TECHNICAL REPORT

ISO/IEC TR 14496-9

Third edition
2009-02-01

Information technology — Coding of audio-visual objects —

Part 9: Reference hardware description

Technologies de l'information — Codage des objets audiovisuels —

Partie 9: Description de matériel de référence

Reference number
ISO/IEC TR 14496-9:2009(E)

© ISO/IEC 2009

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.



COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2009

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

© ISO/IEC 2009 – All rights reserved

Not for Resale

Contents

Page

Foreword v

Introduction vii

1 Scope 1

2 Copyright disclaimer for HDL software modules 1

3 Abbreviated terms 2

4 HDL software availability 2

5 HDL coding format and standards 2
5.1 HDL standards and libraries 2
5.2 Conditions and tools for the synthesis of HDL modules 3
5.3 Conformance with the reference software 3

6 Integrated Framework supporting the "Virtual Socket" between HDL modules described in Part 9 and the MPEG Reference Software (Implementation 1). 3
6.1 Introduction 3
6.2 Addressing 4
6.3 Memory Map 4
6.4 Hardware Accelerator Interface 6
6.5 User Hardware Accelerator Sockets 12

7 Integrated Framework supporting the "Virtual Socket" between HDL modules described in Part 9 and the MPEG Reference Software (Implementation 2). 14
7.1 Introduction 14
7.2 Development Example of a Typical Module : Calc_Sum_Product Module 14
7.3 Second Example of a Typical Module : fifo_transfer module 18
7.4 Integrating the Multi-Modules within the Framework 23
7.5 Calc_Sum_Product Module Controller (memory data transfer) 30
7.6 Simulation of the whole system 44
7.7 Debug Menu 46

8 Integrated Framework supporting the "Virtual Socket" between HDL modules described in Part 9 and the MPEG Reference Software (Implementation 3). 47
8.1 An Integrated Virtual Socket Hardware-Accelerated Co-design Platform for MPEG-4 47
8.2 Reference for Virtual Socket API Function Calls 72
8.3 Tutorial on the Integrated Virtual Socket Hardware-Accelerated Co-design Platform for MPEG-4 Part 9 Implementation 3 95
8.4 An Integration of the MPEG-4 Part 10/AVC DCT/Q Hardware Module into the Virtual Socket Co-design Platform 140
8.5 Migrating Virtual Socket Hardware-Accelerated Co-design Platform From WildCard-II to WildCard-4 153

9 Integrated Framework supporting the "Virtual Socket" between HDL modules described in Part 9 and the MPEG Reference Software: Implementation 4 - Virtual Memory Extension 167
9.1 Introduction 167
9.2 Overview of the "Virtual Socket Platform" implementation 4 167
9.3 Development information 171
9.4 Technical details 172
9.5 How to build the platform 196
9.6 Simulation of the platform 203
9.7 Synthesis of the platform 209
9.8 Building the platform system software 212
9.9 How to use the platform 214

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2009 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.10 Understanding VHDL code ... 216
9.11 Appendix ... 220
9.12 Glossary ... 222

10 HDL MODULES ... 225
10.1 INVERSE QUANTIZER HARDWARE IP BLOCK FOR MPEG-4 PART 2 ... 225
10.2 2-D IDCT HARDWARE IP BLOCK FOR MPEG-4 PART 2 ... 232
10.3 VLD+IQ+IDCT for MPEG-4 Part 2 ... 242
10.4 A SYSTEM C MODEL FOR 2X2 HADAMARD TRANSFORM AND QUANTIZATION FOR MPEG–4 PART 10 ... 247
10.5 A VHDL HARDWARE BLOCK FOR 2X2 HADAMARD TRANSFORM AND QUANTIZATION WITH APPLICATION TO MPEG–4 PART 10 AVC ... 256
10.6 A SYSTEMC MODEL FOR 4X4 HADAMARD TRANSFORM AND QUANTIZATION FOR MPEG-4 PART 10 ... 262
10.7 A VHDL HARDWARE IP BLOCK FOR 4X4 HADAMARD TRANSFORM AND QUANTIZATION FOR MPEG-4 PART 10 AVC ... 269
10.8 A HARDWARE BLOCK FOR THE MPEG-4 PART 10 4X4 DCT-LIKE TRANSFORMATION AND QUANTIZATION ... 275
10.9 A SYSTEMC MODEL FOR THE MPEG-4 PART 10 4X4 DCT-LIKE TRANSFORMATION AND QUANTIZATION ... 281
10.10 A 8X8 INTEGER APPROXIMATION DCT TRANSFORMATION AND QUANTIZATION SYSTEMC IP BLOCK FOR MPEG-4 PART 10 AVC ... 289
10.11 INTEGER APPROXIMATION OF 8X8 DCT TRANSFORMATION AND QUANTIZATION, A HARDWARE IP BLOCK FOR MPEG-4 PART 10 AVC ... 299
10.12 A VHDL CONTEXT-BASED ADAPTIVE VARIABLE LENGTH CODING (CAVLC) IP BLOCK FOR MPEG-4 PART 10 AVC ... 306
10.13 A VERILOG HARDWARE IP BLOCK FOR SA-DCT FOR MPEG-4 ... 311
10.14 A VERILOG HARDWARE IP BLOCK FOR SA-IDCT FOR MPEG-4 ... 322
10.15 A VERILOG HARDWARE IP BLOCK FOR 2D-DCT (8X8) ... 335
10.16 SHAPE CODING BINARY MOTION ESTIMATION HARDWARE ACCELERATION MODULE ... 344
10.17 A SIMD ARCHITECTURE FOR FULL SEARCH BLOCK MATCHING ALGORITHM ... 358
10.18 HARDWARE MODULE FOR MOTION ESTIMATION (4xPE) ... 367
10.19 A IP BLOCK FOR H.264/AVC QUARTER PEL FULL SEARCH VARIABLE BLOCK MOTION ESTIMATION ... 381
10.20 AN IP BLOCK FOR VARIABLE BLOCK SIZE MOTION ESTIMATION IN H.264/MPEG-4 AVC ... 389
10.21 An IP Block for MPEG-4 Part 10 AVC Deblocking Filter ... 400
10.22 A HW BLOCK FOR MPEG-4 PART 10 AVC CONTEXT ADAPTIVE VARIABLE LENGTH CODING (CAVLC) ... 407
10.23 HARDWARE IMPLEMENTATION OF FULL SEARCH MPEG-4 PART 10 AVC MOTION ESTIMATION ... 421

Annex A (Informative) Specification of directory structure for reference SW, HDL and documentation files of MPEG-4 Part 9 Reference HW Description ... 433
A.1 Introduction ... 433
A.2 Directory Structure of TR SW Modules ... 433
CVS Module Name ... 434
Integration Framework Version ... 435
Reference Software Version and Modifications ... 435

Annex B (Informative) Tutorial on Part 9 CVS Client Installation & Operation ... 436

Annex C (Informative) Additional utility software ... 446

Annex D (Informative) Providers of reference hardware code ... 447

Bibliography ... 448

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

In exceptional circumstances, the joint technical committee may propose the publication of a Technical Report of one of the following types:

— type 1, when the required support cannot be obtained for the publication of an International Standard, despite repeated efforts;

— type 2, when the subject is still under technical development or where for any other reason there is the future but not immediate possibility of an agreement on an International Standard;

— type 3, when the joint technical committee has collected data of a different kind from that which is normally published as an International Standard ("state of the art", for example).

Technical Reports of types 1 and 2 are subject to review within three years of publication, to decide whether they can be transformed into International Standards. Technical Reports of type 3 do not necessarily have to be reviewed until the data they provide are considered to be no longer valid or useful.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC TR 14496-9, which is a Technical Report of type 3, was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

This third edition cancels and replaces the second edition (ISO/IEC TR 14496-9:2008) which has been technically revised.

ISO/IEC 14496 consists of the following parts, under the general title *Information technology — Coding of audio-visual objects*:

— *Part 1: Systems*

— *Part 2: Visual*

— *Part 3: Audio*

— *Part 4: Conformance testing*

— *Part 5: Reference software*

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— *Part 6: Delivery Multimedia Integration Framework (DMIF)*

— *Part 7: Optimized reference software for coding of audio-visual objects* [Technical Report]

— *Part 8: Carriage of ISO/IEC 14496 contents over IP networks*

— *Part 9: Reference hardware description* [Technical Report]

— *Part 10: Advanced Video Coding*

— *Part 11: Scene description and application engine*

— *Part 12: ISO base media file format*

— *Part 13: Intellectual Property Management and Protection (IPMP) extensions*

— *Part 14: MP4 file format*

— *Part 15: Advanced Video Coding (AVC) file format*

— *Part 16: Animation Framework eXtension (AFX)*

— *Part 17: Streaming text format*

— *Part 18: Font compression and streaming*

— *Part 19: Synthesized texture streaming*

— *Part 20: Lightweight Application Scene Representation (LASeR) and Simple Aggregation Format (SAF)*

— *Part 21: MPEG-J Graphical Framework eXtension (GFX)*

— *Part 22: Open Font Format*

— *Part 23: Symbolic Music Representation*

— *Part 24: Audio and systems interaction*

— *Part 25: 3D Graphics Compression Model*

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2009 – All rights reserved
Not for Resale

Introduction

The main goal of this Technical Report is to facilitate a more widespread use of the MPEG-4 standard.

Design methodologies of the EDA industry have evolved from schematics to Hardware Description Languages (HDLs) to address the needs of the vast number of gates available on a single device. The increased number of gates allowed more elaborate algorithms to be deployed but also required a shift in design paradigm to handle the complexity created. Through HDLs, more complicated systems could be designed faster through the enabling technology of synthesis of the HDL code towards different silicon technologies where trade offs could be explored. Now the EDA industry again faces challenges where HDLs may not provide the level of abstraction needed for system designers to evaluate system level parameters and complexity issues. There have been a number of tool investigations under way to address this problem. Profiling tools aid in exposing bottlenecks in an abstract way so that early design decisions can be made. C to gates tools allow a C based simulation environment while also enabling direct synthesis to gates for hardware acceleration.

In conclusion, it is the aim of this Technical Report to enable more widespread use of the MPEG-4 standard through reference hardware descriptions and close integration with MPEG-4 Part 7 Optimized Reference Software. Additionally, it is aimed that exposure to such a platform will enable a more systematic way to investigate the complexity of new codecs and open up the algorithm search space with an order of magnitude more compute cycles.

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Information technology — Coding of audio-visual objects —

Part 9: Reference hardware description

1 Scope

This part of ISO/IEC 14496 specifies descriptions of the main video coding tools in hardware description language (HDL) form. Such alternative descriptions to the ones that are reported in ISO/IEC 14496-2, ISO/IEC 14496-5 and ISO/IEC TR 14496-7 correspond to the need of providing the public with conformant standard descriptions that are closer to the starting point of the development of codec implementations than textual descriptions or pure software descriptions. This part of ISO/IEC 14496 contains conformant descriptions of video tools that have been validated within the recommendation ISO/IEC TR 14496-7.

2 Copyright disclaimer for HDL software modules

Each HDL module shall be accompanied by the following copyright disclaimer, to be included in each HDL module and all derivative modules:

```
/*************************************************************************

This software module was originally developed by

   <Family Name>, <Name>, <email address>, <Company Name>

   (date: <month>,<year>)

and edited by: <Family Name>, <Name>,<email address>

This HDL module is an implementation of a part of one or more MPEG-4
tools(ISO/IEC 14496).

ISO/IEC gives users of the MPEG-4 free license to this HDL module or
modifications thereof for use in hardware or software products claiming
conformance to the MPEG-4 Standard.

Those intending to use this HDL module in hardware or software products are
advised that its use may infringe existing patents.

The original developer of this HDL module and his/her company, the subsequent
editors and their companies, and ISO/IEC have no liability for use of this HDL
module or modifications thereof in an implementation.

Copyright is not released for non MPEG-4 Video conforming products.

<Company Name> retains full right to use the code for his/her own purpose, assign
or donate the code to a third party and to inhibit third parties from using the
code for non MPEG standard conforming products.
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

```
This copyright notice must be included in all copies or derivative works.

Copyright (c) <year>.

Module Name:  <module_name>.vhd

Abstract:


Revision History:

*************************************************************************/
```

3 Abbreviated terms

For the purposes of this document, the following abbreviated terms apply:

AV	Audio-Visual
DCT	Discrete Cosine Transform
IDCT	Inverse Discrete Cosine Transform
HDL	Hardware Description language
ISO	International Organization for Standardization
MPEG	Moving Picture Experts Group
Verilog	A Hardware Description Language
VHDL	VHSIC high speed Hardware Description Language
SAD	Sum of Absolute Differences
MAC	Multiply ACcumulate
MAD	Minimum Absolute Difference
SIMD	Single Instruction Multiple Data
DA	Distributive Arithmetic
EDA	Electronic Design and Automation
IEEE	Institute of Electrical and Electronic Engineers
IMEC	Interuniversity Micro Electronic Center
EPFL	École Polytechnique Fédérale de Lausanne

4 HDL software availability

The HDL and System C software modules described in this part of ISO/IEC 14496 are available within the zip file containing this Technical Report. Each module contains a separate directory structure for the source code with a readme.txt file explaining the top level and all files to be included for simulation and synthesis.

5 HDL coding format and standards

5.1 HDL standards and libraries

As the IEEE has several HDL coding standards that are commonly used in hardware reference code (i.e. VHDL1076-1987, VHDL 1164-1993, Verilog 1364-2000, Verilog 1364-1995), the modules constituting this part

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

of ISO/IEC 14496 are made of the latest IEEE standard possible at the time of coding for all reference HDL code. As the IEEE has provided libraries to assist in the use of HDL, only IEEE standard libraries are needed to use the HDL code.

Custom libraries which are specific to the vendor's (Silicon) base library elements are used only if they are freely available for synthesis and simulation and are provided in an accompanying module version of the submitted HDL code using the standard libraries mentioned above.

5.2 Conditions and tools for the synthesis of HDL modules

As there are many choices commercially for HDL synthesis and HDL simulation software tools, specific synthesis or simulation libraries that are used for reference HDL code are properly documented. The same code that is used to synthesize towards an implementation is also used to perform HDL behavioral simulation of the MPEG-4 tool. The code is properly documented with respect to the synthesis and simulation tool (and version) that has been used to perform the work. HDL module codes with multiple synthesis and simulation tools are also possible. In the event a source code modification must be made to support an additional synthesis or simulation tool, an additional source code is provided with proper documentation.

5.3 Conformance with the reference software

HDL reference code provides sufficient test bench code and documentation on how it is conformant with respect to the reference software. To the extent possible, bit and cycle true models are provided which can be used directly in the reference software code for verification. In the case that the reference HDL code is derived from other languages such as: C, C++, System C, Java, it is recommended that that this code and information on the methodology used to generate HDL should be provided to improve verification of conformance of the HDL code.

6 Integrated Framework supporting the “Virtual Socket” between HDL modules described in Part 9 and the MPEG Reference Software (Implementation 1).

6.1 Introduction

The aim of this clause is to document the framework developed by Xilinx Research Labs for the integration of HW modules with the MPEG-4 reference software. The purpose of this virtual socket framework is to create an abstraction between the specific physical layer and specific software driver library to facilitate a reusable hardware/software co-design environment. By acting as an intermediary between specific physical layer bus protocols, the hardware accelerator designer can focus on the acceleration algorithm rather than lower level interface protocols.

The framework of the Virtual Socket allows for 31 addressable hardware accelerators to be present in a single device (see Figure 1). Each specific hardware accelerator will be assigned a bit of the 32-bit hardware identification register and these bit locations shall be assigned to particular MPEG development teams (see Figure 2 for an example containing two accelerators at slots 1 and 6). If an accelerator socket is not present then its bit in the identification register will be de-asserted. Unassigned sockets will also be de-asserted indicating no accelerator is present. In the event that hardware accelerator designers wish to put further identification of their socket they may do so by allocating further identification registers within their socket’s assigned register space.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

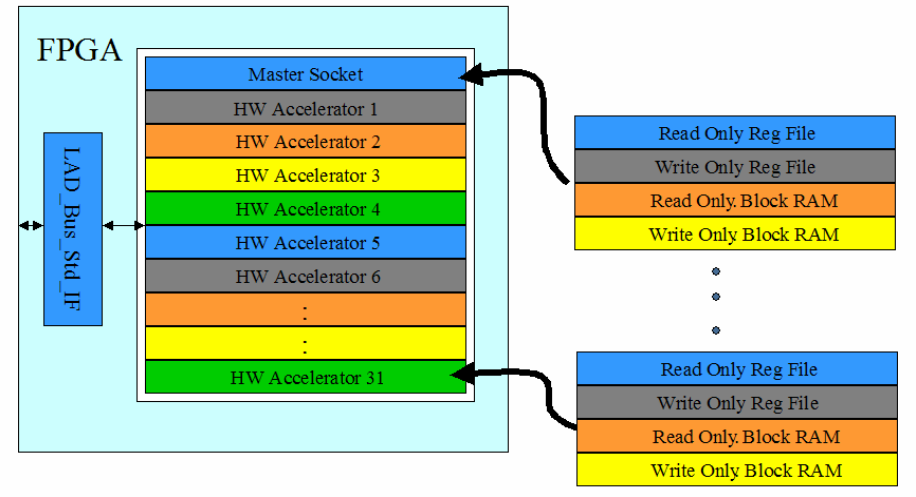


Figure 1 — Block Diagram of Virtual Socket Platform.

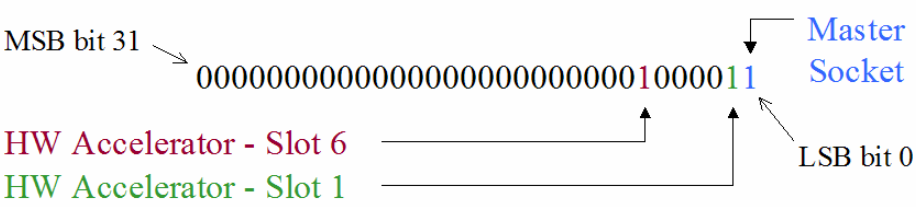


Figure 2 — Example 32-Bit Hardware Identification Register.

6.2 Addressing

The virtual socket provides four strobes that indicate what region of the memory space, register or memory, has been accessed as well as the type of operation, write or read. Although a 16-bit is provided to each socket, the least significant nine bits are only necessary to address within the 512 word assigned memory region. The Virtual Socket API uses macros that assist the software designer in transferring data to and from memory locations.

6.3 Memory Map

Table 1 — Memory Mapping for Register File Allocation.

	Register Read-Only			Register Write-Only	
Socket #	Begin	End		Begin	End
Master	0000	01FF		4000	41FF
1	0200	03FF		4200	43FF
2	0400	05FF		4400	45FF
3	0600	07FF		4600	47FF
4	0800	09FF		4800	49FF
5	0A00	0BFF		4A00	4BFF
6	0C00	0DFF		4C00	4DFF
7	0E00	0FFF		4E00	4FFF

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2009 – All rights reserved
Not for Resale

8	1000	11FF		5000	51FF
9	1200	13FF		5200	53FF
10	1400	15FF		5400	55FF
11	1600	17FF		5600	57FF
12	1800	19FF		5800	59FF
13	1A00	1BFF		5A00	5BFF
14	1C00	1DFF		5C00	5DFF
15	1E00	1FFF		5E00	5FFF
16	2000	21FF		6000	61FF
17	2200	23FF		6200	63FF
18	2400	25FF		6400	65FF
19	2600	27FF		6600	67FF
20	2800	29FF		6800	69FF
21	2A00	2BFF		6A00	6BFF
22	2C00	2DFF		6C00	6DFF
23	2E00	2FFF		6E00	6FFF
24	3000	31FF		7000	71FF
25	3200	33FF		7200	73FF
26	3400	35FF		7400	75FF
27	3600	37FF		7600	77FF
28	3800	39FF		7800	79FF
29	3A00	3BFF		7A00	7BFF
30	3C00	3DFF		7C00	7DFF
31	3E00	3FFF		7E00	7FFF

Table 2 — Memory Mapping for the Block RAM Allocation.

	Memory Read-Only			**Memory Write-Only**	
Socket #	**Begin**	**End**		**Begin**	**End**
Master	8000	81FF		C000	C1FF
1	8200	83FF		C200	C3FF
2	8400	85FF		C400	C5FF
3	8600	87FF		C600	C7FF
4	8800	89FF		C800	C9FF
5	8A00	8BFF		CA00	CBFF
6	8C00	8DFF		CC00	CDFF
7	8E00	8FFF		CE00	CFFF
8	9000	91FF		D000	D1FF
9	9200	93FF		D200	D3FF
10	9400	95FF		D400	D5FF
11	9600	97FF		D600	D7FF
12	9800	99FF		D800	D9FF
13	9A00	9BFF		DA00	DBFF
14	9C00	9DFF		DC00	DDFF
15	9E00	9FFF		DE00	DFFF
16	A000	A1FF		E000	E1FF
17	A200	A3FF		E200	E3FF
18	A400	A5FF		E400	E5FF
19	A600	A7FF		E600	E7FF
20	A800	A9FF		E800	E9FF
21	AA00	ABFF		EA00	EBFF

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

22	AC00	ADFF		EC00	EDFF
23	AE00	AFFF		EE00	EFFF
24	B000	B1FF		F000	F1FF
25	B200	B3FF		F200	F3FF
26	B400	B5FF		F400	F5FF
27	B600	B7FF		F600	F7FF
28	B800	B9FF		F800	F9FF
29	BA00	BBFF		FA00	FBFF
30	BC00	BDFF		FC00	FDFF
31	BE00	BFFF		FE00	FFFF

Table 1 and Table 2 show the memory mapping for the 31 hardware sockets in the virtual socket platform. Note in Figure 1 that the memory is allocated into four distinct sections: 1) read-only register file; 2) write-only register file; 3) read-only block RAM; and 4) write-only block RAM. The allocation size for each type of memory for every HW socket is 512 bytes.

6.4 Hardware Accelerator Interface

Figure 3 shows a typical block diagram of a hardware accelerator socket. Note that input and output block RAMs are provided for input and output data while important flags are mapped to the register file sections, such as start and finish flags.

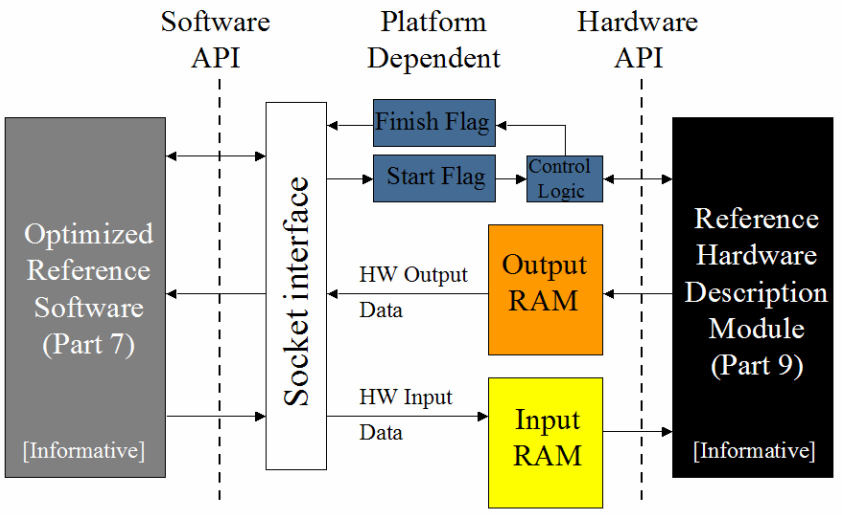


Figure 3 — Block Diagram of Typical Hardware Accelerator.

When a hardware socket is selected for a particular transaction, one of its strobes will be asserted. It is up to the user's particular socket designs whether register or memory regions will be treated differently, however in most cases their behaviour may be identical. The necessary signals to interface to the virtual socket with respect to the hardware accelerator socket are shown in Table 3 below.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2009 – All rights reserved

Table 3 — Hardware Accelerator Socket Interface.

Signal	Length	Direction*	Polarity	Description
Globals	**<2>**			
Clk	1	Input	R	hardware accelerator socket clock
global_reset	1	Input	H	global reset
Strobes	**<4>**			
strobe_reg_read	1	Input	H	read-only register space selected
strobe_reg_write	1	Input	H	write-only register space selected
strobe_ram_read	1	Input	H	read-only memory space selected
strobe_ram_write	1	Input	H	write-only memory space selected
Write Signals	**<50>**			
write_addr	16	Input		write address
data_in	32	Input		data to write into socket
write_valid	1	Input	H	data_in is valid
write_rdy	1	Output	H	socket available to take more write data
Read Signals	**<49>**			
read_addr	16	Input		read address
data_out	32	Output		data for read operation
strobe_out	1	Output	H	data_out has requested data
External Memory Manager	**<92>**			
ZBT_ReadEmpty	1	Input	H	read fifo is empty
ZBT_WriteFull	1	Input	H	write fifo is full
ZBT_ack_job	1	Input	H	job to memory manager accepted
ZBT_wf_grant	1	Input	H	write fifo access granted
ZBT_rf_grant	1	Input	H	read fifo access granted
ZBT_ReadData	32	Input		data read from external memory
ZBT_issue_job	1	Output	H	issue job to memory manager
ZBT_rwb	1	Output	H	job is read = '1' or write = '0'
ZBT_popfifo	1	Output	H	retrieve word of data from read fifo
ZBT_pushfifo	1	Output	H	place data onto write fifo
ZBT_addr	19	Output		address to access data to in external memory
ZBT_dpush	32	Output		data to send to external memory

The user may optionally connect their device to the external memory manager that allows access to either the ZBT SRAM or DDR DRAM (see subclause 6.4.2). The *block move* and *external block move* example VHDL modules demonstrate the basic interface to the virtual socket (see subclause 6.5). Hardware socket designers are strongly encouraged to read this section and use them as building blocks for their own sockets.

6.4.1 Transferring Data To/From a Socket

When a socket detects a write access to it, it should check to see if the write_valid signal is asserted. This signal will indicate that the data present on Data_In is valid and ready to be processed by the socket. Whenever the user is capable of taking data from the virtual socket interface it should drive its write_rdy signal high. This will bring new data to it from the interface. Register or Memory writes to a socket may be multiple words. The write_rdy signal provides a flow control mechanism back to the virtual socket interface. Below are two waveforms demonstrating example writes to register and memory space.

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

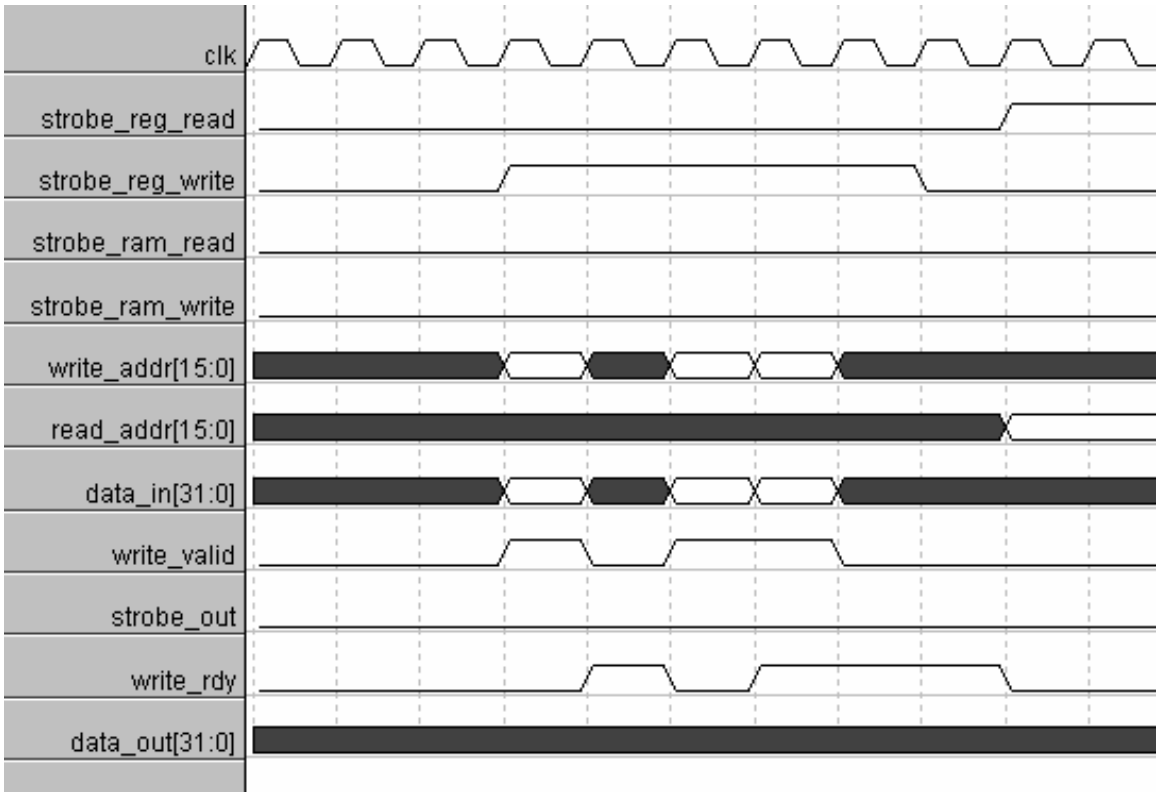


Figure 4 — Timing Diagram of Register Write Operation.

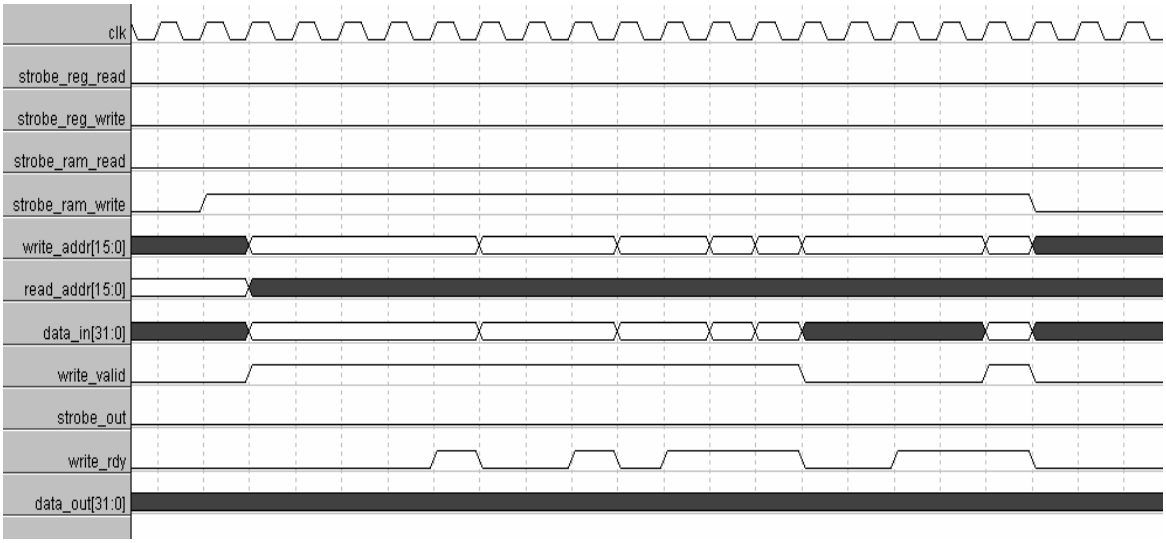


Figure 5 — Timing Diagram of Memory Write Operation.

To perform a read transaction the user should observe a register or memory read strobe asserted. The user has up to 16 clocks to respond to the read transaction by providing the data requested at the read address on data_out and asserting the strobe_out signal. Read operations only return a single word. Samples waveforms are provided below in Figure 6 and Figure 7:

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

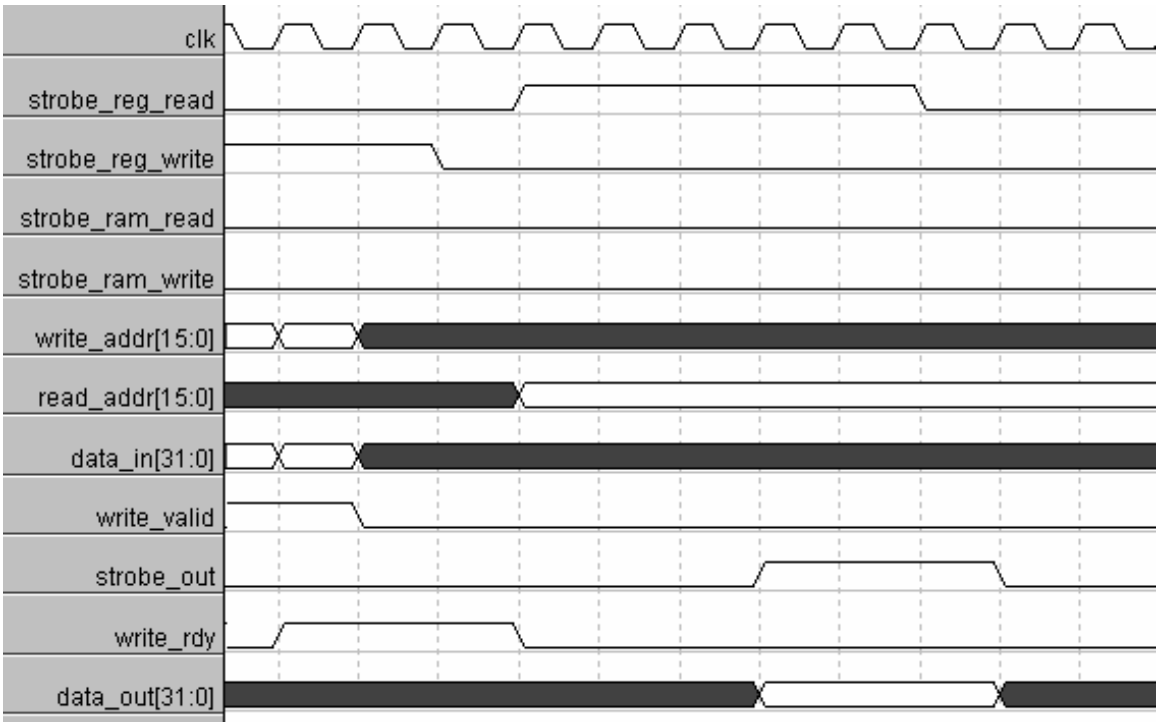


Figure 6 — Timing Diagram of Register Read Operation.

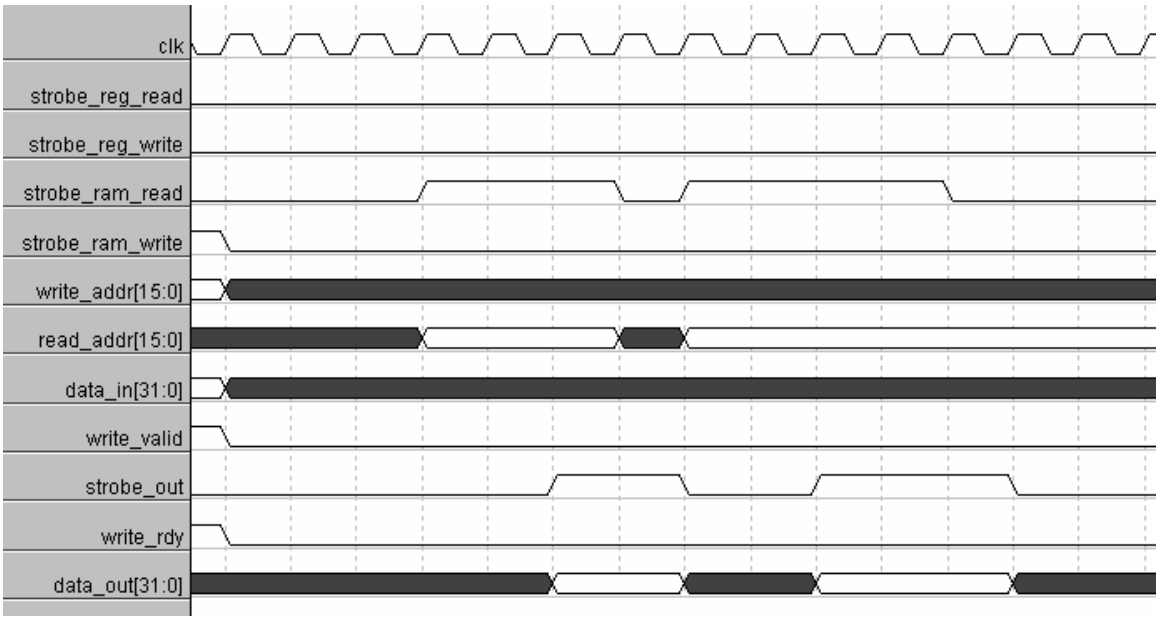


Figure 7 — Timing Diagram of Memory Read Operation.

The hardware socket should implement a simple state machine as shown below in Figure 8.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

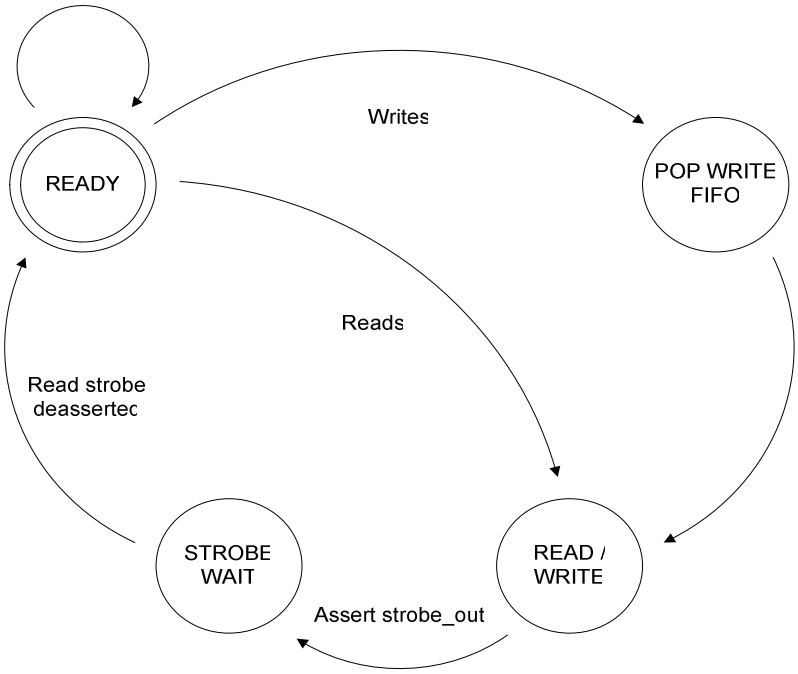


Figure 8 — State Machine of Hardware Accelerator.

6.4.2 External Memory Interface

The external memory manager allows for three hardware accelerator sockets to access the two external memories contained on the WildCard-II, ZBT SRAM and DDR DRAM. The manager arbitrates access to the memory using a round-robin decision method. Sockets requesting access to static or dynamic RAM must first issue a job request to the manager and the manager will respond with an acknowledgement. The socket should then wait until the manager returns either a write or read grant depending on the requested operation. Once a read or write grant is asserted, the socket should either withdraw or deposit (read or write respectively) from the manager. The accesses are for single words only so if a socket requires multiple words, it must request multiple jobs with the manager. An example state machine is provided in Figure 9 demonstrating how the user's socket should interface with the memory manager.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

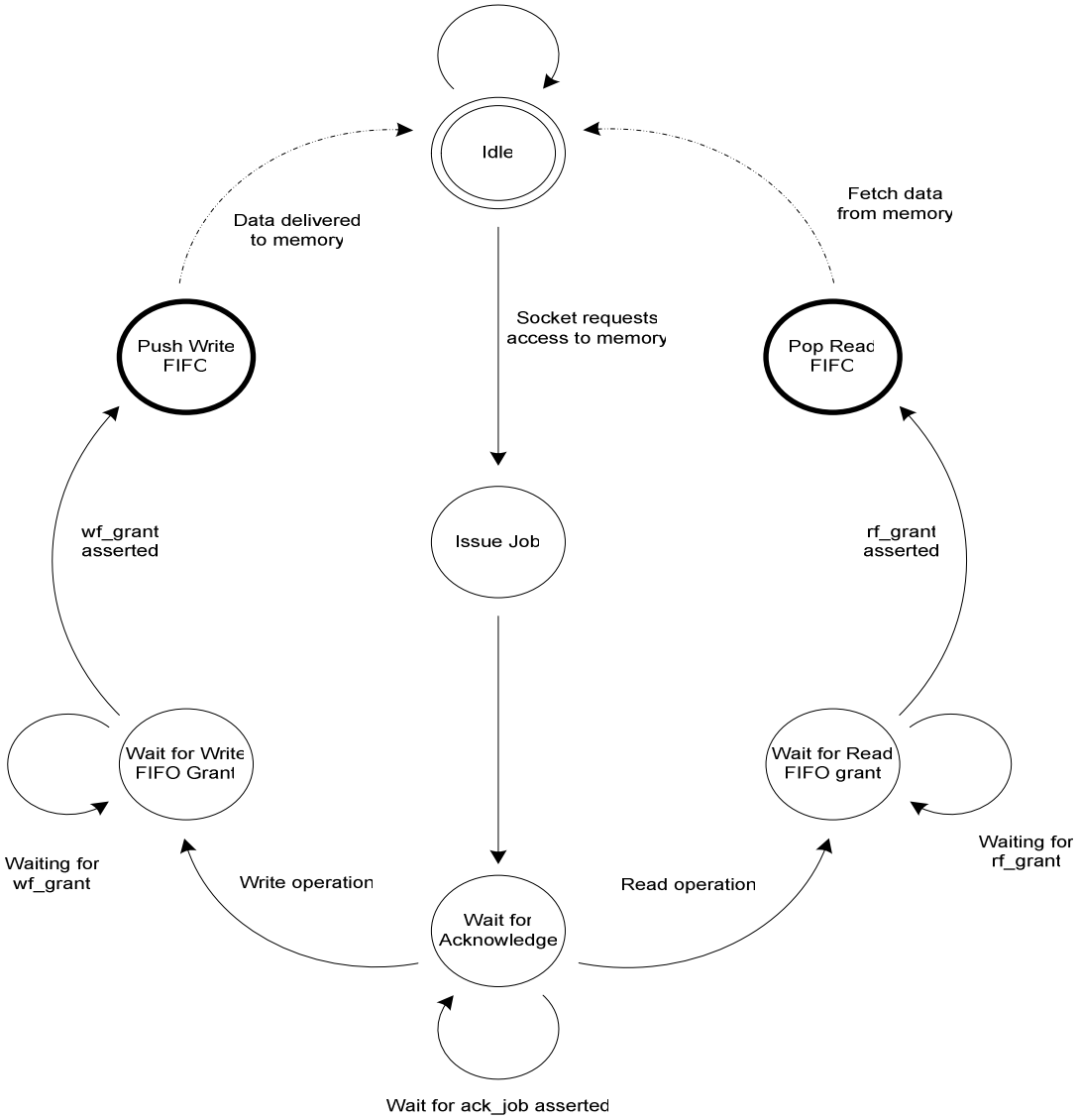


Figure 9 — State Machine of External Memory Interface.

The addressing to the external memories is shown in Figure 10. Note that the user writes a start address value into register location 1 of the master socket. The least-significant nine bits of the memory write value sent to the master socket is then added to the start address register to obtain the final address sent to the external memory.

©ISO/IEC 2009 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

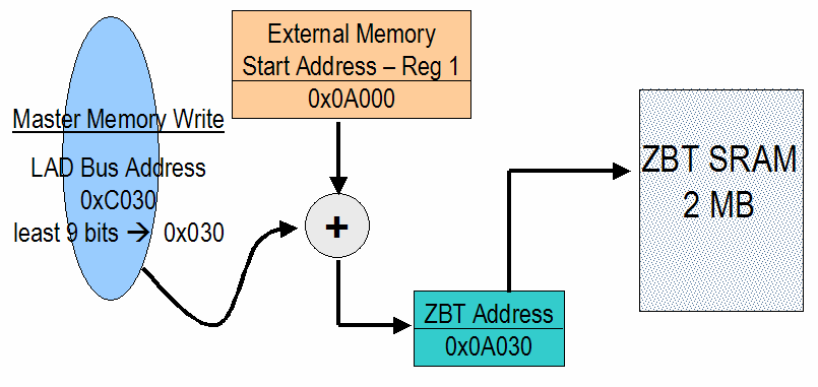


Figure 10 — Addressing Technique Used to Access External Memories.

6.5 User Hardware Accelerator Sockets

Developers of their own hardware accelerator socket should follow the examples listed below to instantiate and activate their own sockets. The VHDL source code for these two examples is provided with the platform and is already pre-connected to the platform. Users are encouraged to copy these examples to use as a template for their own accelerator design. The virtual socket interface is comprised of two major modules that cannot be altered by the user – the master socket and memory manager. These modules must be present to allow software to access external memory as well as provide status information back to control software.

6.5.1 Block Move

The *block move* example connects to the virtual socket interface connections and performs a very simple task - copying the contents of one internal RAM to another. Specifically the block move example copies a region of the Write-Only memory to the Read-Only memory. The example watches for a write to its Start register and begins a loop reading from one memory and writing to the other. Once the block move module has finished this task, it sets its Valid register high. Figure 11 and Figure 12 show the interface and internal block diagram of the block move example, respectively. Note that the interface matches the basic interface listed in Table 3.

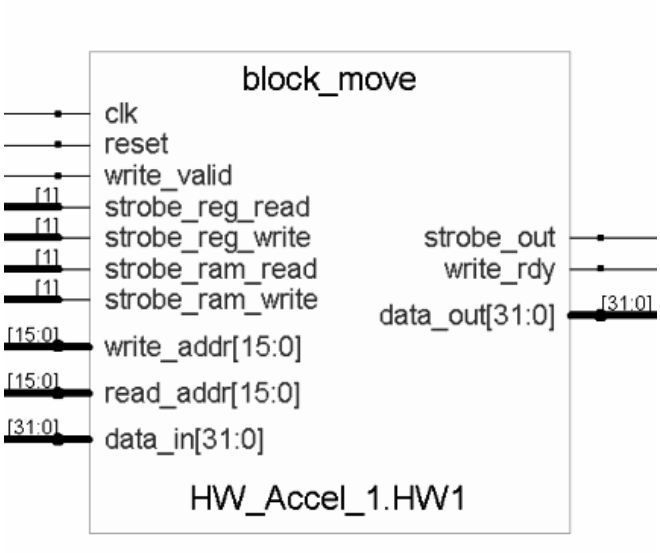


Figure 11 — Interface of Block Move Example.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

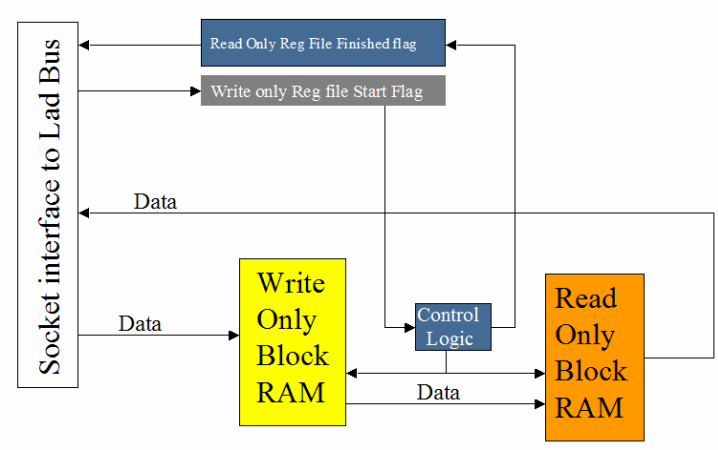


Figure 12 — Block Diagram of Block Move Example.

6.5.2 External Memory Block Move

The *external block move* example VHDL module performs a copy from one region of ZBT SRAM memory to another. Figure 13 shows the interface of this block. It demonstrates the additional connectivity to the external memory manager. This socket provides additional registers to configure the source and destination addresses of the copy and the length of the transfer. A write to the Start register begins the copy and the status of the transfer is provided in the Status register. The number of words moved is present in the high bits of the Status register and the current state of the socket in the lower three bits. When the transfer has completed, the number of words should match the requested length to copy and the state should be all zeroes indicating it has finished.

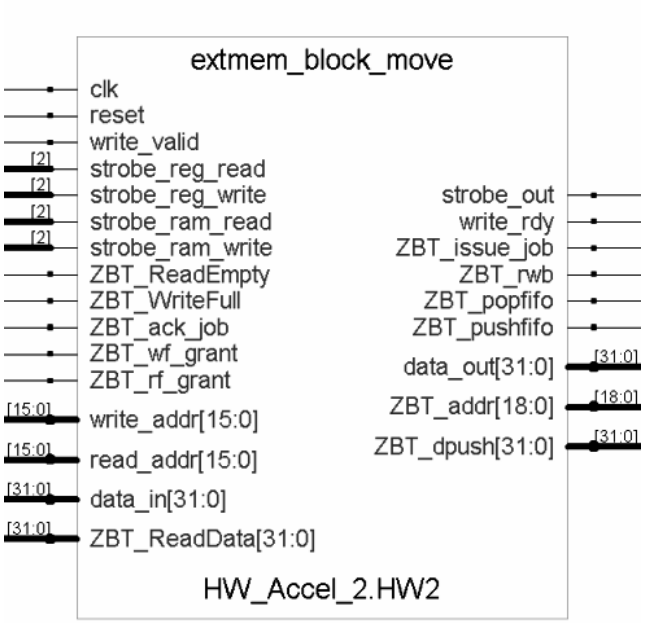


Figure 13 — Interface of External Block Move Example.

© ISO/IEC 2009 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7 Integrated Framework supporting the "Virtual Socket" between HDL modules described in Part 9 and the MPEG Reference Software (Implementation 2).

7.1 Introduction

The aim of this clause is to document the framework developed at the University of Calgary for the integration of HW modules with the MPEG-4 reference software.

The clause is formatted in the form of a tutorial that takes the reader step by step through a typical HDL module development case. After the introduction, two examples are described. The first one permit to transfer from host data into FPGA internal memory block, then to do a simple process and then the data are transferred back to host. The second example is similar, but instead of data processing the data are stored into a internal fifo (designed with FPGA memory blocks) and straight away read back by the host. For both examples, a testbench is descripted to test the two IPs and to help user to understand how host's command words could be interpreted and the processing performed. The following shows the integration of these two examples into the platform and how host controls these IPs. Then it is shown how to use API function and how to use the debug platform.

A typical data follow for an IP accelerator is: The data is transferred from the host computer to FPGA memory block (BRAM) or SRAM memory, (using either direct transfer or DMA protocol) the processing is done, then the results are stored respectively into BRAM or SRAM or (DRAM which is not presented in this document) depending on the source of the original data. The computer host initiates and controls (through software calls) the IP with an IP controller (designed by the user). Different types of controller are detailed into the next sections. The system includes two parts (layers): A software part in C/C++ and a Part in HDL that simulate the IP block. Both share the operation to perform the functionality required. In all examples of this document, it was chosen that the S/W layer receives and responds to the all interrupts (act as Interrupt management). Also the S/W controls the start and acknowledges the operation of the H/W module.

7.2 Development Example of a Typical Module : Calc_Sum_Product Module

In this example, The data are transferred from the host computer to FPGA memory block (BRAM) memory with no DMA, nor SRAM the processing is done, then the results are stored respectively into BRAM. The host initiates and controls the IP with an IP controller. The controller is into the VHDL file calc_sum_product.vhd.

```
entity calc_sum_product is
        generic(no_of_inputs: positive:=64);
        port(
            reset:           in std_logic;
            clock:           in std_logic;
                -----------------------------------------
            module_ready:        out std_logic;
            input_available: in std_logic;
            datain:      in std_logic_vector(7 downto 0);
                -----------------------------------------
            output_available:    out std_logic;
            dataout:         out std_logic_vector(15 downto 0);
                ---------------------------async transfers--
            async_in:        in std_logic;
            async_out:       out std_logic
        );
end calc_sum_product;
```

Figure 14 — Interface of a Typical Module.

The specifications of this calculator:

1. Takes in 64 words-8 bits each, one word at a time (asynchronous transfer), each word is stored into FPGA memory block.
2. It calculates the sum and the product of each two consecutive words. The sum and the product are separately stored in 16-bit words.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3. When the calculation for all the input is done, it begins to output the 64 results, one word at a time, then waits for new input.
4. The interface signals are:
a. Reset. (Input)
b. Clock. (Input)
c. module_ready. (Output) (the module is ready for receiving new input stream)
d. Input_ready (Input) (signals start of the input stream)
e. Output_ready. (Output) (signals start of the output stream)
f. Data_in. (Input)
g. Data_out. (Output)
h. Async_in (Input), Async_out (Output) asynchronous data transfer handshake signals. They are used in inverse sense when data source and data destination switch roles.

Assumptions:

a. Reset is active high.
b. System is positive edge triggered.
c. One clock delay between data and handshake signals.

The second step is to have this module tested by preferably a testbench file similar to the following one shown in Figure 15.

```
------------------------------------------------------------
-- example testbench for the example calculator module
-- Tamer Mohamed and Wael Badawy (badawy@ucalgary.ca)
-- University of Calgary
------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity tb_calc_sum_product is
end tb_calc_sum_product;

architecture stimulus of tb_calc_sum_product is

    type memory8bits is array (natural range <>) of std_logic_vector(7 downto 0);
    type memory16bits is array (natural range <>) of std_logic_vector(15 downto 0);
    type system_modes is (idle, receive_up, receive_dn, calculating, transmit_up, transmit_dn);

    constant m_period: time :=10 ns; -- suggests operation at 100 MHz
    constant tb_period: time :=7 ns;
    constant no_of_in: positive:=16;      -- suggests 16 inputs only

    signal input_array: memory8bits(0 to no_of_in-1);
    signal output_array: memory16bits(0 to no_of_in-1);
    signal current_state: system_modes;

    signal reset, m_clk: std_logic;
    signal tb_clk: std_logic;
    signal module_ready: std_logic;
    signal input_available, output_available: std_logic;
    signal async_in, async_out: std_logic;
    signal datain: std_logic_vector(7 downto 0);
    signal dataout: std_logic_vector(15 downto 0);

    signal initiate_processing: std_logic;        -- external trigger signal

    component calc_sum_product
```

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    generic(no_of_inputs: positive:=64);
        port(
            reset:           in std_logic;
            clock:           in std_logic;
            module_ready:        out std_logic;
            input_available: in std_logic;
            datain:      in std_logic_vector(7 downto 0);
            output_available:    out std_logic;
            dataout:         out std_logic_vector(15 downto 0);
            async_in:        in std_logic;
            async_out:       out std_logic
        );
    end component;

begin

    UUT: calc_sum_product
        generic map (no_of_inputs => no_of_in)
        port map (
            reset => reset,
            clock => m_clk,
            module_ready => module_ready,
            input_available => input_available,
            datain => datain,
            output_available => output_available,
            dataout => dataout,
            async_in => async_in,
            async_out => async_out
        );
                    ---- begin external signals
    input_array
<=(X"00",X"01",X"02",X"03",X"04",X"05",X"06",X"07",X"08",X"09",X"0A",X"0B",X"0C",X"0D",X"0E",X"0F");
    reset <= '1', '0' after 12 ns;
    initiate_processing <= '1', '0' after 50 ns, '1' after 1000 ns, '0' after 1050 ns;

    module_clock: process
    begin
        m_clk <= '0';        wait for m_period/2;
        m_clk <= '1';        wait for m_period/2;
    end process module_clock;

    testbench_clock: process
    begin
        tb_clk <= '0';       wait for tb_period/2;
        tb_clk <= '1';       wait for tb_period/2;
    end process testbench_clock;

                    ---- end external signals

    stimuli: process(reset, tb_clk)
        variable counter: integer range 0 to 63;
    begin
        if reset='1' then
            counter :=0;
            current_state <=idle;
            input_available <='0';
            datain <=(others=>'0');
        elsif rising_edge(tb_clk) then
            case current_state is
                when idle =>
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2009 – All rights reserved

```
                    if initiate_processing='1' then
                        if module_ready='1' then
                            input_available <='1';
                            current_state <= receive_up;
                        end if;
                    end if;
                when receive_up =>
                    if async_out='0' then
                        async_in <='1';      -- latch
                        datain <=input_array(counter);
                        current_state <= receive_dn;
                    end if;
                when receive_dn =>
                    if async_out='1' then
                        async_in <='0';
                        counter :=((counter+1) mod no_of_in);
                        if counter=0 then
                            input_available <='0';
                            current_state <= calculating;
                        else
                            current_state <= receive_up;
                        end if;
                    end if;
                when calculating =>
                    if output_available='1' then
                        current_state <= transmit_up;
                    end if;
                when transmit_up =>
                    if async_out='1' then -- latch
                        output_array(counter) <= dataout;
                        async_in <='1'; --acknowledged
                        current_state <= transmit_dn;
                    end if;
                when transmit_dn =>
                    if async_out='0' then
                        async_in <='0';
                        counter :=((counter+1) mod no_of_in);
                        if counter=0 then
                            current_state <=idle;
                        else
                            current_state <=transmit_up;
                        end if;
                    end if;
            end case;
        end if;

    end process stimuli;

end stimulus;
```

Figure 15 — Testbench of the Typical Module.

The following points should be noted about the structure of the test-bench file:

1. The testbench file is divided into external control part and the “stimuli” process which is responsible for feeding the data to the module. This process; as a state machine, has the same states as the calculator module.
2. The process; *“stimuli”,* is made sensitive to a clock and a reset signals. This provides for maximizing code reuse because this process can be taken directly and integrated as part of the module controller discussed in the following sections.

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3. The sequence of the operations is described into the testbench, therefore in the implementation a component (HW_controller) will be in charge of it. The 16 data words (constant no_of_in := 16;) are stored into a input array, memory (will see for implementation), after the processing the stimuli process is in charged to store the results into the output array memory. The sequence can be split into 3 phases:
a. reading data (received_up and received_dn)
b. processing data (calculating)
c. writing resulting data (transmit_up and transmit_dn)
4. There are two generated clocks in the test-bench: *m_clk* which is the module clock and *tb_clk* which is the clock used for the test-bench process. There is no relation between the two clocks as *m_period* is 10ns and *tb_period* is 7ns. To account for other cases, the two periods should be interchanged.
5. Data transfers are asynchronous using the handshake signals *async_in* and *async_out*.
6. The external triggering signal called "*initiate_processing*" is asserted two times with a delay between the two times it is asserted. The goal of this is to make sure that both the module and its controlling process; (process "stimuli") can run again after going into the idle state. This makes sure that the block after an initial run will not be left in a state that may cause errors during a successive run.
7. To make reviewing the results in the simulator easier, the module is instantiated in the test-bench with only 16 inputs. This is the main use of the generic value here.

Figure 16 shows the simulation of the testbench.

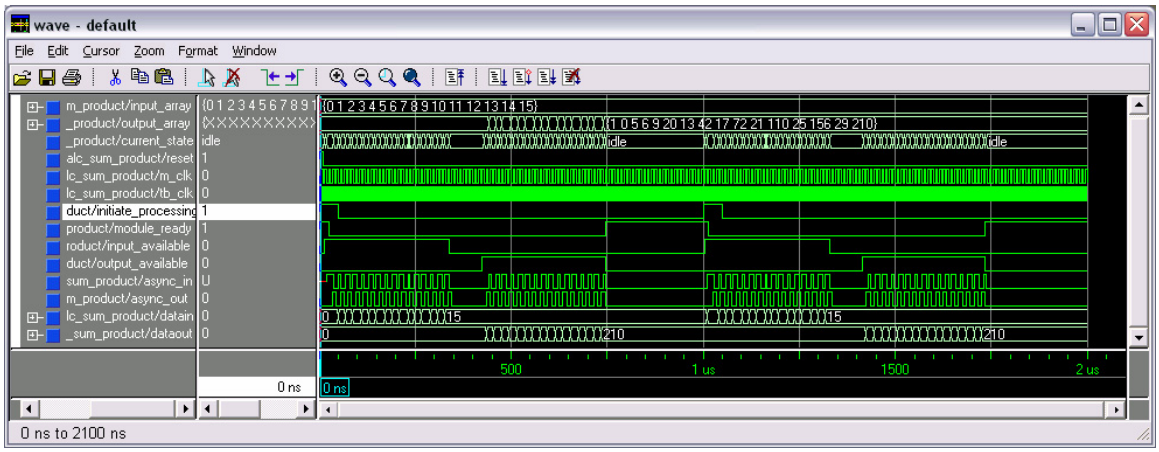


Figure 16 — Simulation of the Testbench for the *calc_sum_product* Module.

The compilation in Modelsim can done by a macro file that can bet similar to the following:

```
-------------------------------------------
-- compilation macro file for target module
-- Tamer Mohamed and Wael Badawy (badawy@ucalgary.ca)
-------------------------------------------

set MODULE_BASE D:/MPEG4/UoC_framework3/vhdl/sim/calc

vlib $MODULE_BASE/calc_lib
vmap calc_lib $MODULE_BASE/calc_lib

vcom -93 -explicit -work calc_lib \
    $MODULE_BASE/calc_sum_product.vhd \
    $MODULE_BASE/tb_calc_sum_product.vhd
```

Figure 17 — "compilation.do" macro file for the module and its testbench.

7.3 Second Example of a Typical Module : fifo_transfer module

In this subclause we will develop a second example and in the following sections we will integrate both developed modules in our framework. Our second module is a FIFO register file.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The specifications of this FIFO register:

1. Its depth is 64 words-16 bits each, accepts one word at a time (asynchronous transfer).
2. The interface signals are:
a. Reset. (Input)
b. Clock. (Input)
c. fifo_empty. (Output) (stored word counter is at 0)
d. fifo_full. (Output) (stored word counter is at maximum)
e. Write_fifo (Input) (signals a word to be loaded into the FIFO)
f. Write_ack (Output) (asynchronous handshake)
g. read_fifo. (Input) (signals topmost word is to be emptied)
h. read_ack (Output) (asynchronous handshake)
i. Datain. (Input)
j. Dataout. (Output)

Assumptions:

a. Reset is active high.
b. System is positive edge triggered.
c. Data is latched with the write_fifo signal.
d. Data writing and data reading are two independent operations, each has its own asynchronous acknowledge signal. This was not the case in the first example.

The following is a description of the module and its test-bench in VHDL reported in Figure 5 and Figure 6. The simulation result for two runs is shown in Figure 20.

Note that the outline of the testbench code is almost identical to the one in the first example. This helps in code reuse and emphasizes the abstraction layer discussed in the next subclause. Also note the use of two clocks.

```
------------------------------------------------------------
-- example FIFO hw module
-- Tamer Mohamed and Wael Badawy (badawy@ucalgary.ca)
-- University of Calgary
------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity fifo_reg is
        generic(
            fifo_depth: positive:=64;
            fifo_width: positive:=8
        );
        port(
            reset:  in std_logic;
            clock:  in std_logic;
            fifo_empty: out std_logic;
            fifo_full:  out std_logic;
                -----------------------------------------
            write_fifo: in std_logic;
            write_ack:  out std_logic;
            datain: in std_logic_vector(fifo_width-1 downto 0);
                -----------------------------------------
            read_fifo:  in std_logic;
            read_ack:   out std_logic;
            dataout:    out std_logic_vector(fifo_width-1 downto 0)
        );
end fifo_reg;
```

Figure 18 — FIFO Module Example (described in fifo_reg.vhd).

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2009 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
------------------------------------------------------------
-- example testbench for the example FIFO module
-- Tamer Mohamed and Wael Badawy (badawy@ucalgary.ca)
-- University of Calgary
------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity tb_fiforeg is
end tb_fiforeg;

architecture stimulus of tb_fiforeg is

    constant m_period: time :=7 ns;
    constant tb_period: time :=10 ns; -- suggests operation at 100 MHz
    constant no_of_in: positive:=16;      -- suggests 16 inputs only
    constant d_wid: positive:=8;

    type memorybits is array (natural range <>) of std_logic_vector(d_wid-1 downto 0);
    type system_modes is (idle, receive_ack, transmit_ack);
    type system_requests is (idle, push_fifo, pop_fifo);

    signal current_state: system_modes;
    signal current_request: system_requests;

    signal reset, m_clk, tb_clk: std_logic;
    signal fifo_empty, fifo_full: std_logic;
    signal read_fifo, read_ack: std_logic;
    signal write_fifo, write_ack: std_logic;
    signal datain: std_logic_vector(d_wid-1 downto 0);
    signal dataout: std_logic_vector(d_wid-1 downto 0);
    signal fifo_data: std_logic_vector(d_wid-1 downto 0);
    signal request_done: std_logic;

    signal initiate_processing: std_logic;        -- external trigger signal
    signal counter1, counter2: integer range 0 to no_of_in-1;
    signal input_array: memorybits(0 to no_of_in-1);
    signal output_array: memorybits(0 to no_of_in-1);

    component fifo_reg is
        generic(
            fifo_depth: positive:=64;
            fifo_width: positive:=8
        );
        port(
            reset:       in std_logic;
            clock:       in std_logic;
            fifo_empty:  out std_logic;
            fifo_full:   out std_logic;
                -----------------------------------------
            write_fifo:  in std_logic;
            write_ack:   out std_logic;
            datain:  in std_logic_vector(fifo_width-1 downto 0);
                -----------------------------------------
            read_fifo:   in std_logic;
            read_ack:    out std_logic;
            dataout:     out std_logic_vector(fifo_width-1 downto 0)
        );
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2009 – All rights reserved

```
    end component;

begin

    UUT: fifo_reg
        generic map (
            fifo_depth => no_of_in,
            fifo_width => d_wid
        )
        port map (
            reset => reset,
            clock => m_clk,
            fifo_empty => fifo_empty,
            fifo_full => fifo_full,
            write_fifo => write_fifo,
            write_ack => write_ack,
            datain => datain,
            read_fifo => read_fifo,
            read_ack => read_ack,
            dataout => dataout
        );
                    ---- begin external signals
    input_array
<=(X"00",X"01",X"02",X"03",X"04",X"05",X"06",X"07",X"08",X"09",X"0A",X"0B",X"0C",X"0D",X"0E",X"0F");
    reset <= '1', '0' after 12 ns;
    initiate_processing <= '1', '0' after 50 ns, '1' after 750 ns, '0' after 800 ns;

    current_request <= push_fifo, idle after 320 ns, pop_fifo after 350 ns, idle after 700 ns,
push_fifo after 750 ns;

    counter1_p: process
    begin
         wait until write_ack='1';
         counter1 <= (counter1+1) mod no_of_in;
    end process counter1_p;
    counter2_p: process
    begin
         wait until read_ack='1';
         counter2 <= (counter2+1) mod no_of_in;
    end process counter2_p;

    fifo_data <= input_array(counter1);
    output_array(counter2) <= dataout;

    module_clock: process
    begin
        m_clk <= '0';        wait for m_period/2;
        m_clk <= '1';        wait for m_period/2;
    end process module_clock;

    testbench_clock: process
    begin
        tb_clk <= '0';       wait for tb_period/2;
        tb_clk <= '1';       wait for tb_period/2;
    end process testbench_clock;
                    ---- end external signals

    stimuli: process(reset, tb_clk)
    begin
```

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
        if reset='1' then
            current_state <=idle;
            write_fifo <='0';
            read_fifo <='0';
            datain <=(others=>'0');
            request_done<='1';
        elsif rising_edge(tb_clk) then
            case current_state is
                when idle =>
                    if current_request=push_fifo then
                        if fifo_full='0' then
                            datain <= fifo_data;
                            write_fifo <='1';
                            current_state <= receive_ack;
                            request_done <='0';
                        end if;
                    elsif current_request=pop_fifo then
                        if fifo_empty='0' then
                        read_fifo <='1';
                            current_state <= transmit_ack;
                            request_done <='0';
                        end if;
                    else
                        write_fifo <='0';
                        read_fifo <='0';
                        request_done <='1';
                    end if;

                when receive_ack =>
                    if write_ack='1' then
                        write_fifo <='0';
                        current_state <= idle;
                    end if;

                when transmit_ack =>
                    if read_ack='1' then
                        read_fifo <='0';
                        current_state <= idle;
                    end if;
            end case;

        end if;

    end process stimuli;

end stimulus;
```

Figure 19 — Testbench for the FIFO Module (described in .vhd).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2009 – All rights reserved

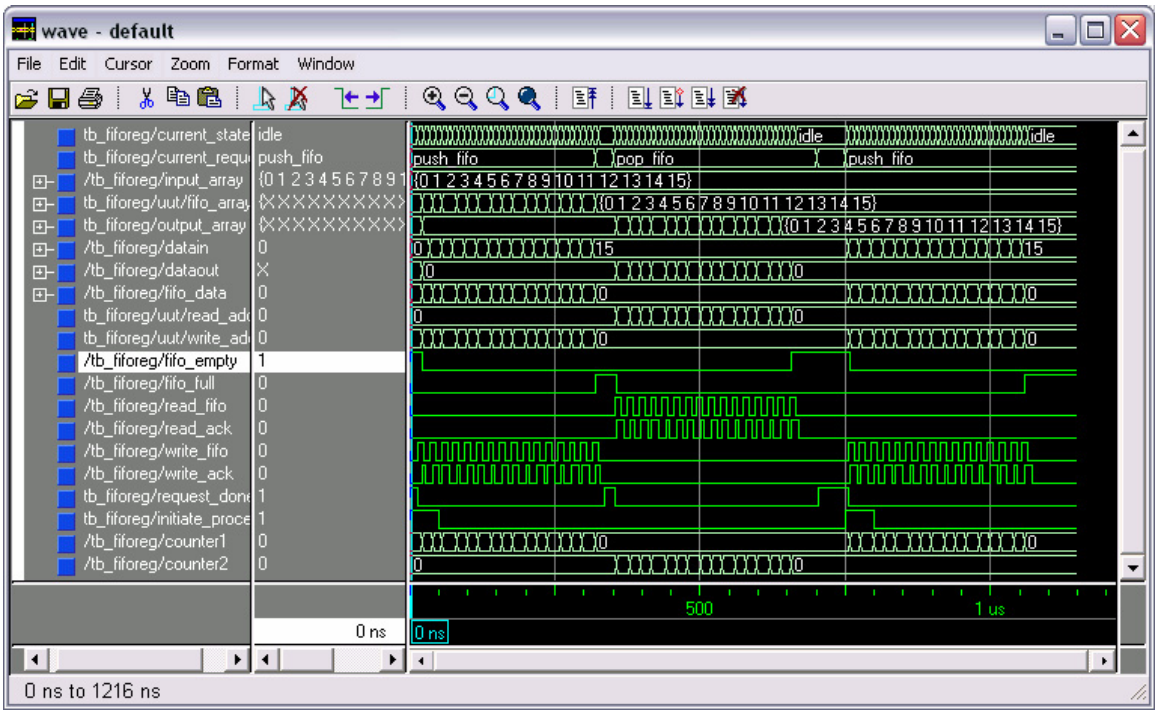


Figure 20 — Simulation of Two Consecutive Runs of the FIFO.

7.4 Integrating the Multi-Modules within the Framework

In this subclause we demonstrate the interation of 2 modules but the system can be scaled up to 16 modules. In the previous part, the connection is established between the HW integration and the API function calls. To obtain information about these functions, please refer to the software platform code (Visual UoCfv3 project) or for simulation to function used with the test vectors (in host_arch.vhd). The results of test vectors can be observed with ModelSim using the script simulation named project_vsim.do. In the simulation, four examples are described, and two of them are the fifo and cal examples.

USER should be warned that:

1. A new processing IP is ALWAYS associated with a controller module which should be designed by the user!!! In this controller, there is at least a LAD interface. Two controllers are provides into the following examples.
2. The second example permits to user IP to access to SRAM.

The HW part of the framework consists of architecture that is mapped to the FPGA. The architecture can be described by Figure 21.

© ISO/IEC 2009 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

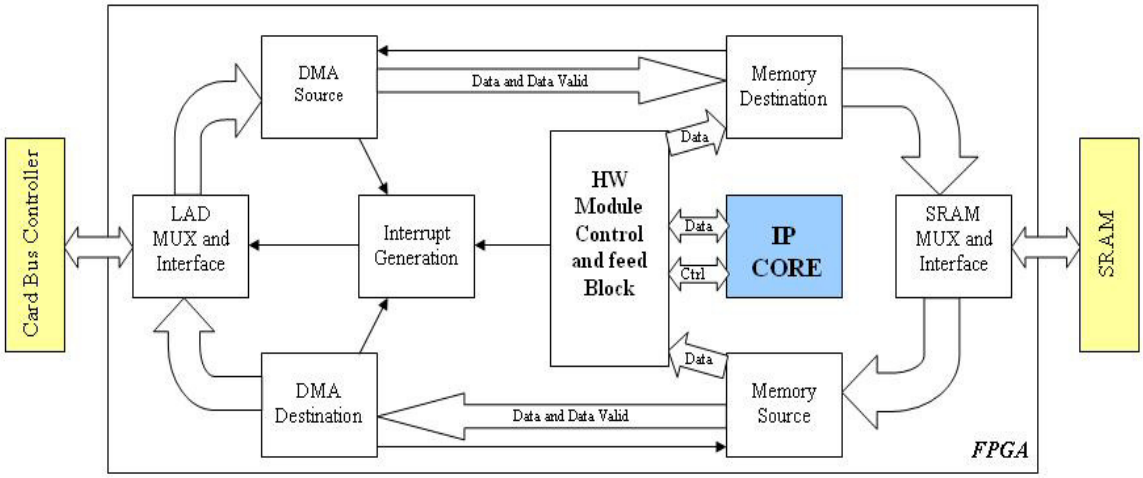


Figure 21 — System architecture mapped to the FPGA.

The “HW module Control and Feed Block” is an abstraction layer whose purpose is to shield the designer of the IP core from the intricate details of the other interfacing blocks required in the HW/SW environment. The other blocks are responsible for interfacing with the PCI bus of the host system and for interfacing with the on board memory chips (SRAM for instance).

Our example system will function properly with the two IP blocks discussed in the previous two sections.

We may choose that the FIFO block be implemented with direct register transfers via the bus and that the calculator takes benefit of the DMA capability. This will illustrate how two different mechanisms of data transfer are implemented through the framework.

We will begin by the simple case of data transfer in the form of one word at a time. The FIFO example suits this type of transfer.

7.4.1 FIFO Module Controller (basic data transfer)

The test-bench file is used as a starting point and is integrated in a simple VHDL template that addresses the following:

1. Interfacing with the local address bus (LAD) through which the host system communicates with the processing element (PE) or FPGA.
2. Passing external control arguments to the “stimuli” process described in the test-bench file and letting it handle the main module.
3. Accessing the card memory if required. (Not the case in this example).
4. Generating an interrupt signal to indicate that the main module finished processing.
5. Assigns values to a status register to indicate the current state of the system. This status register can be queried periodically by the host system if the interrupt signal is masked (which is the case in this example as will be illustrated in the full system).

The description of the controller interface is as follows:

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
entity fifo_hw_module_controller is
   generic(
        BASE_address: std_logic_vector(15 downto 0);
        address_MASK: std_logic_vector(15 downto 0):=X"FFF0"; -- 16 registers
        fifo_size: positive:=16;
        fifo_width: positive:=16
    );
    port(
      reset: in std_logic;
      m_clk: in std_logic;       -- module clock
      b_clk: in std_logic;       -- bus clock
      module_done: out std_logic;
      ------------------------ memory access arbiter
      write_address: out std_logic_vector(20 downto 0);
      read_address: out std_logic_vector(20 downto 0);
      enable_write: out std_logic;
      enable_read : out std_logic;
      access_request: out std_logic;
      access_grant: in std_logic;
      Memory_Source_Data_Valid: in std_logic;
      mem_datain : in std_logic_vector(31 downto 0);
      mem_dataout : out std_logic_vector(31 downto 0);
      ------------------------   interface with LAD bus
      LAD_instrobe: in std_logic;
      LAD_address: in std_logic_vector(15 downto 0);
      LAD_write: in std_logic;
      LAD_datain: in std_logic_vector(31 downto 0);
      LAD_dataout: out std_logic_vector(31 downto 0);
      LAD_strobe_out: out std_logic
    );
end fifo_hw_module_controller;
```

Figure 22 — Interface of the FIFO Controller (declared in fifo_ctrl2.vhd).

The following points should be noticed:

1. Generics: BASE_address and address_MASK provide the parameters for the address bus comparator. Fifo_size and fifo_width are parameters for the main ip module.
2. Signals: The signals are divided into 3 categories: global control (reset and the two clocks), card memory access signals, LAD interface.

The controller file template is divided into 3 main *processes*:

1. LAD interface process.
2. Memory interface process.
3. IP-module interface process. (This is almost the exact form used in the test-bench *"stimuli"* process which is the point in code reuse)

The example here omits the memory access process because it is not used. It will be shown in the second example for the calculator.

The VHDL controller architecture (declared in fifo_ctrl2.vhd) is shown in Figure 23.

```
architecture behav of fifo_hw_module_controller is

    constant zeropad: std_logic_vector(31-fifo_width downto 0):=(others=>'0');
    type system_modes is (idle, receive_ack, transmit_ack);
    type system_requests is (idle, push_fifo, pop_fifo);

    signal current_state: system_modes;
    signal current_request: system_requests;
    signal request_done: std_logic;
```

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    signal status_register: std_logic_vector(7 downto 0);

    --signal reset, m_clk, tb_clk: std_logic;
    signal fifo_full, fifo_empty: std_logic;
    signal read_fifo, read_ack: std_logic;
    signal write_fifo, write_ack: std_logic;
    signal fifo_data: std_logic_vector(fifo_width-1 downto 0);
    signal datain: std_logic_vector(fifo_width-1 downto 0);
    signal dataout: std_logic_vector(fifo_width-1 downto 0);

    component fifo_reg is
        generic(
            fifo_depth: positive:=64;
            fifo_width: positive:=8
        );
        port(
            reset:          in std_logic;
            clock:          in std_logic;
            fifo_empty:     out std_logic;
            fifo_full:      out std_logic;
                ------------------------------------------
            write_fifo:     in std_logic;
            write_ack:      out std_logic;
            datain:     in std_logic_vector(fifo_width-1 downto 0);
                ------------------------------------------
            read_fifo:      in std_logic;
            read_ack:       out std_logic;
            dataout:        out std_logic_vector(fifo_width-1 downto 0)
        );
    end component;

begin
    U_fiforeg: fifo_reg
        generic map (
            fifo_depth => fifo_size,
            fifo_width => fifo_width
        )
        port map (
            reset => reset,
            clock => m_clk,
            fifo_empty => fifo_empty,
            fifo_full => fifo_full,
            write_fifo => write_fifo,
            write_ack => write_ack,
            datain => datain,
            read_fifo => read_fifo,
            read_ack => read_ack,
            dataout => dataout
        );

module_done <= request_done;
status_register <= ( 0=>fifo_empty, 1=>fifo_full, others=>'0');

    -- This module will not access memory (optimized during synthesis)
    write_address <=(others=>'0');
    read_address <=(others=>'0');
    enable_write <='0';
    enable_read <='0';
    access_request <='0';
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2009 – All rights reserved
Not for Resale

```
    mem_dataout <=(others=>'0');
    ------------------------------------------------------------------

LAD_interface: process(reset, b_clk)
begin
  if reset='1' then
      LAD_dataout <=(others=>'0');
      LAD_strobe_out <='0';
      current_request <=idle;
      fifo_data <= (others=>'0');

  elsif rising_edge(b_clk) then
        LAD_strobe_out <='0';
        if (LAD_inStrobe = '1') then
        if ((LAD_Address and address_MASK) = BASE_address) then

            if LAD_Address(2 downto 0)="000" then  -- 000 is data write/read
                    if LAD_Write = '1' then
                        fifo_data <= LAD_datain(fifo_width-1 downto 0);
                        current_request <= push_fifo;
                    else
                        current_request <= pop_fifo;
                        LAD_dataout <= zeropad & dataout;
                        LAD_strobe_out <='1';
                    end if;
            else
                current_request <= idle;
            end if;

            if LAD_Address(2 downto 0)="001" then -- 001 is for control/status
                if LAD_write='0' then
                    LAD_dataout <= X"000000" & status_register;
                    LAD_strobe_out <='1';
                end if;
            end if;

       end if;  -- ends check addressing
     else
            current_request <= idle;
     end if; -- ends check input strobe
   end if;   -- ends reset or bus clock
end process LAD_interface;

    stimuli: process(reset, b_clk)
    begin
        if reset='1' then
            current_state <=idle;
            write_fifo <='0';
            read_fifo <='0';
            datain <=(others=>'0');
            request_done <='1';
        elsif rising_edge(b_clk) then
            case current_state is
                when idle =>
                    if current_request=push_fifo then
                        if fifo_full='0' then
                            datain <= fifo_data;
                            write_fifo <='1';
                            current_state <= receive_ack;
                            request_done <='0';
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
                    end if;
                elsif current_request=pop_fifo then
                    if fifo_empty='0' then
                    read_fifo <='1';
                        current_state <= transmit_ack;
                        request_done <='0';
                    end if;
                else
                    write_fifo <='0';
                    read_fifo <='0';
                    request_done <='1';
                end if;

            when receive_ack =>
                if write_ack='1' then
                    write_fifo <='0';
                    current_state <= idle;
                end if;

            when transmit_ack =>
                if read_ack='1' then
                    read_fifo <='0';
                    current_state <= idle;
                end if;
        end case;
    end if;

end process stimuli;

end behav;
```

Figure 23 — Architecture of the Controller Designed for Integration with Framework

In order to link the host control and IP processing, the exchange process are going to be detailed for this example. Exchanges can processed between the host and a HW module, only if the associated HW controller address is recognized, then control signals are LAD_strobe_out, LAD_instrobe, or signal LAD_write permit data and command transfers. The LAD interface is in charge of these operations, moreover it controls the stimuli process which describes the user operation scheduling, in this example the fifo memory processing. Please note that the fifo memory is designed with FPGA memory blocks.

In this example, the design of LAD interface is connected to API function calls. If the LAD_Address finished by “000” then a read or write could be performed on fifo component depending on signal LAD_Write. If the LAD_Address finished by “001” then status register is then read back, it can be used to schedule data transfer, because the status register represents the fifo status (bit 0 = fifo_empty, bit 1 = fifo_full). This structure is then easy to control with API functions or simulation functions (Figure 24 and Figure 25). In both case, functions are the command word to transfer data or access status register. In this case, the fifo_ctrl_offset is equal to 0x240 (cf UoCfv3.c). Using fifo_ctrl_offset enables to transfer data to fifo memory (because `LAD_Address(2 downto 0)="000"`), using fifo_ctrl_offset+1 enables to access to status_register.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2009 – All rights reserved
Not for Resale

```
void test_fifo() {
    int k;
    DWORD       dDatain[16];
    DWORD       dDataout[16];
    DWORD        control[1];
    WC_RetCode  rc = WC_SUCCESS;
    char operat;
    printf("\n\nPushing 16 word random data to fifo\n");
    for (k=0; k<16; k++) {
        dDatain[k]=rand(); dDatain[k]=(dDatain[k]%100);
        WC_PeRegWrite(TestInfo.DeviceNum,fifo_ctrl_offset,1, dDatain[k]);
        //Function prototype described Annexe1
        WC_PeRegRead(TestInfo.DeviceNum, fifo_ctrl_offset+1, 1, control); // not used
        //Function prototype described Annexe1
        printf("(%x)",control[0]);
    }
    printf("\nPoping 16 word data from fifo\n");
    for (k=0; k<16; k++) {
    dDataout[k]=0;
    WC_PeRegRead(TestInfo.DeviceNum,fifo_ctrl_offset,1,&dDataout[k]);
    WC_PeRegRead(TestInfo.DeviceNum, fifo_ctrl_offset+1, 1, control); // not used
    }
    for(k=0; k<8; k++)   {
        printf("{Tx(%2d):%-5d, Rx:%-5d}\t{Tx(%2d):%-5d, Rx:%-5d}\n",
                2*k,dDatain[2*k],dDataout[2*k],2*k+1,dDatain[2*k+1],dDataout[2*k+1]);
    }
}
```

Figure 24 — API function call (From UoCfv3.c).

```
-------------------------------------------------------testfifo
  report "Testing FIFO move";
 --
--  push then pop
  --
  for i in 1 to 16 loop
    dData(0) :=i;
    WC_PeRegWrite( Cmd_Req, Cmd_Ack, device, fifo_ctrl_offset, 1, dData);
    report "Push ( " & integer'image(i) & ") = " & Hex_To_String(dData(0));
    end loop;

    WC_PeRegRead( Cmd_Req, Cmd_Ack, device, fifo_ctrl_offset+1, 1, dData);
    report "Fifo status: " & Hex_To_String(dData(0));

    for i in 1 to 16 loop
    dData(0) := 0;
    WC_PeRegRead( Cmd_Req, Cmd_Ack, device, fifo_ctrl_offset, 1, dData);
    report "Pop ( " & integer'image(i) & ") = " & Hex_To_String(dData(0));
    end loop;

    WC_PeRegRead( Cmd_Req, Cmd_Ack, device, fifo_ctrl_offset+1, 1, dData);
    report "Fifo status: " & Hex_To_String(dData(0));
    WC_PeRegWrite( Cmd_Req, Cmd_Ack, device, fifo_ctrl_offset, 1, dData);
    report "further Push" & " = " & Hex_To_String(dData(0));
    WC_PeRegRead( Cmd_Req, Cmd_Ack, device, fifo_ctrl_offset, 1, dData);
    report "further Pop"  & " = " & Hex_To_String(dData(0));
```

Figure 25 — Vector test for full system simulation (described in host_arch.vhd).

Copyright International Organization for Standardization © ISO/IEC 2009 All rights reserved
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.5 Calc_Sum_Product Module Controller (memory data transfer)

The test-bench file is used as a starting point and is integrated in a simple VHDL template that addresses the following:

1. Interfacing with the local address bus (LAD) through which the host system communicates with the processing element (PE) or FPGA.
2. Passing external control arguments to the “stimuli” process described in the test-bench file and letting it handle the main module.
3. Accessing the card memory if required. (Which is one of two possible cases in this example).
4. Generating an interrupt signal to indicate that the main module finished processing.
5. Assigns values to a status register to indicate the current state of the system.

The description of the controller interface (defined into calc_ctrlr.vhd) is as follows (Figure 26):

```
entity calc_hw_module_controller is
   generic(
        BASE_address: std_logic_vector(15 downto 0);
        address_MASK: std_logic_vector(15 downto 0):=X"FFE0"; -- 32 registers
        block_width: positive:=4
    );
    port(
      reset: in std_logic;
      m_clk: in std_logic;       -- module clock
      b_clk: in std_logic;       -- bus clock
      module_done: out std_logic;
      ------------------------ memory access arbiter
      write_address: out std_logic_vector(20 downto 0);
      read_address: out std_logic_vector(20 downto 0);
      enable_write: out std_logic;
      enable_read : out std_logic;
      access_request: out std_logic;
      access_grant: in std_logic;
      Memory_Source_Data_Valid: in std_logic;
      mem_datain : in std_logic_vector(31 downto 0);
      mem_dataout : out std_logic_vector(31 downto 0);
      ------------------------   interface with LAD bus
      LAD_instrobe: in std_logic;
      LAD_address: in std_logic_vector(15 downto 0);
      LAD_write: in std_logic;
      LAD_datain: in std_logic_vector(31 downto 0);
      LAD_dataout: out std_logic_vector(31 downto 0);
      LAD_strobe_out: out std_logic
    );
end calc_hw_module_controller;
```

Figure 26 — Controller VHDL Interface for SRAM and BRAM.

The following points should be noticed:

1. Generics: BASE_address and address_MASK provide the parameters for the address bus comparator. Fifo_size and fifo_width are parameters for the main ip module.
2. Signals: The signals are divided into 3 categories: global control (reset and the two clocks), card memory access signals, LAD interface.
3. The interface is identical to the one used for the fifo controller which is intentional to enforce design consistency and code reuse across control modules. This illustrates the authors’ point that most of the effort is concentrated in the development of the ip-module not in writing code for the abstraction layer necessary to interface it with the whole system.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2009 – All rights reserved

The controller file template is divided into 3 main *processes*:

4. LAD interface process.
5. Memory interface process.
6. IP-module interface process. (This is almost the exact form used in the test-bench *"stimuli"* process which is the point in code reuse)

The design of LAD_interface should be designed in a similar way than in 7.4 (i.e. Figure 23, 27, 29). When IP is addressed (in example address between 0x180 and 0x200), as 7.4 depending of control signals, data can be transferred or status register scanned (structure of status register are presented in _Annexe 1). The structure of the LAD_interface is presented in Figure 27. It receives informations (with address and control signals) to control the stimuli process (described in calc_ctrl.vhd) in charge of operations scheduling (data transfer, processing, results transfer). The user can control operations from the host, in this example (Figure 28), the operations realized in API function as follow:

1. Checking of the status register,
2. Data transfer from host into on-board memory,
3. After the Data transfer an interrupt is generated to the S/W layer running on the computer host. This module then control the operation of the HDL IP.Checking of end of processing (test on dDataout[0] which received bit 0 of status register),
4. Results transfer from internal memory to host.

NB: Address_mask2 is used to permit to access to internal memory (for data transfer), moreover if (LAD_Address and address_MASK) = BASE_address but ((LAD_Address and address_mask2) /= BASE_address) then you access to status register (for instance in this case when LAD_address = calc_ctrl_offset+16 as described in Figure 28).

```
LAD_interface: process(reset, b_clk)
    variable counter: integer range 0 to block_width**2-1;
begin
if reset='1' then
    LAD_dataout <=(others=>'0');
    LAD_strobe_out <='0';
    counter:=0;
    start_read_address <= (others=>'0');
    start_write_address <= (others=>'0');
    frame_width <=0;
    frame_height <=0;
    no_frames <=0;
    module_programmed<='0';

elsif rising_edge(b_clk) then
        LAD_strobe_out <='0';
        if (LAD_inStrobe = '1') then
            if ((LAD_Address and address_MASK) = BASE_address) then

                if ((LAD_Address and address_mask2)=BASE_address) then
                    if LAD_Write = '1' then      -- data block
                        input_array(counter) <= LAD_datain(7 downto 0);
                        counter:=(counter+1) mod (block_width**2);
                        if counter=0 then
                            no_frames <= 0;      -- 0 means block mode
                            module_programmed <='1';
                        end if;
                    else
                        LAD_dataout <= X"0000" & output_array(counter);
                        LAD_strobe_out <='1';
                        counter:=(counter+1) mod (block_width**2);
                    end if;
```

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
                else
                    if LAD_write='1' then    -- control block
                        case LAD_Address(1 downto 0) is
                            when"00" =>
                                frame_width <= conv_integer(unsigned(LAD_datain(9 downto 2))); --
divide width by 4
                                frame_height <= conv_integer(unsigned(LAD_datain(19 downto 10)));
                                no_frames <= conv_integer(unsigned(LAD_datain(23 downto 20)));
                            when"01" =>
                                start_read_address <= LAD_datain(20 downto 0);
                            when"10" =>
                                start_write_address <= LAD_datain(20 downto 0);
                            when"11" =>
                                module_programmed <='1';
                            when others => null;
                        end case;
                    else
                        LAD_dataout <= X"000000" & status_register;
                        LAD_strobe_out <='1';
                    end if;
                end if;

            end if;  -- ends check addressing
        else
            module_programmed<='0';
        end if;  -- ends check input strobe
    end if;   -- ends reset or bus clock
end process LAD_interface;
```

Figure 27 — LAD interface described in calc_ctrlr.vhd.

```
////////////////////////////////////////////////////////////////////////////////////////
// function used to call processing (P1#) declared in UoCfv3.c
void test_calc() {
    int k;
    DWORD       dDatain[16];
    DWORD       dDataout[16];
    WC_RetCode  rc = WC_SUCCESS;
        for (k=0; k<16; k++) {
        dDatain[k]=k+1; dDataout[k]=0;
    }
    WC_PeRegRead(TestInfo.DeviceNum, calc_ctrl_offset+16, 1, dDataout);    // Test of the status
register
    printf("\nCalc status %x ",dDataout[0]);
    printf("\n\nMoving 16 word test vector\n");
    WC_PeRegWrite(TestInfo.DeviceNum, calc_ctrl_offset, 16, dDatain);  // 16 data words written
    dDataout[0]=0;
    while(dDataout[0]<1) {
    WC_PeRegRead(TestInfo.DeviceNum, calc_ctrl_offset+16, 1, dDataout);    // Test of the status
register
    //printf("Calc status %x ",dDataout[0]);
    }
    WC_PeRegRead(TestInfo.DeviceNum, calc_ctrl_offset, 16, dDataout);
    for(k=0; k<16; k++) {
        printf("{Tx(%2d):%-5d, Rx:%-5d}\n",
            k,dDatain[k],dDataout[k]);
    }
}
```

Figure 28 — Example to transfer data and receive data with computer host (described in UoCfv.c).

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2009 – All rights reserved

The full code for the controller with SRAM connection is represented in Figure 29. The controller is made to work in two modes: (1) block data mode and (2) memory addressing mode. This is dependent on the programmed number of frames. If it is zero then the *mem_generate_address* process assumes that a data block has been written directly via the LAD bus and just signals the *stimuli* process. On the other hand, if the number of frames is at least one then it is assumed that a at least one whole YUV frame has been written to the SRAM and the *mem_generate_address* process begins to divide each frame in the memory into macro blocks of number of pixels per side equal to *block_width*.

In the block data mode, the host would select data block mode simply by writing the correct amount of data to the first (block_width^2) addresses in the allocated address space for this module. The block size is expected to be much smaller than the actual size.

The memory addressing mode is selected by programming the SRAM read_start, write_start addresses and the number of frames passed to memory by DMA.

The code for this controller might seem complicated, however, the structure is consistent with the idea of code reuse and all the complexity is actually concentrated in the part for calculating how to partition a raster stored YUV image into macro blocks.

```
------------------------------------------------------------
-- Memory access and module controller
------------------------------------------------------------
library IEEE;
   use IEEE.std_logic_1164.all;
   use IEEE.std_logic_unsigned.all;
   use IEEE.std_logic_arith.all;

entity calc_hw_module_controller is
   generic(
        BASE_address: std_logic_vector(15 downto 0);
        address_MASK: std_logic_vector(15 downto 0):=X"FFE0"; -- 32 registers
        block_width: positive:=4
    );
    port(
      reset: in std_logic;
      m_clk: in std_logic;       -- module clock
      b_clk: in std_logic;       -- bus clock
      module_done: out std_logic;
      ------------------------ memory access arbiter
      write_address: out std_logic_vector(20 downto 0);
      read_address: out std_logic_vector(20 downto 0);
      enable_write: out std_logic;
      enable_read : out std_logic;
      access_request: out std_logic;
      access_grant: in std_logic;
      Memory_Source_Data_Valid: in std_logic;
      mem_datain : in std_logic_vector(31 downto 0);
      mem_dataout : out std_logic_vector(31 downto 0);
      ------------------------   interface with LAD bus
      LAD_instrobe: in std_logic;
      LAD_address: in std_logic_vector(15 downto 0);
      LAD_write: in std_logic;
      LAD_datain: in std_logic_vector(31 downto 0);
      LAD_dataout: out std_logic_vector(31 downto 0);
      LAD_strobe_out: out std_logic
    );
end calc_hw_module_controller;
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
architecture behav of calc_hw_module_controller is
    type memory8bits is array (natural range <>) of std_logic_vector(7 downto 0);
    type memory16bits is array (natural range <>) of std_logic_vector(15 downto 0);
    type system_modes is (idle, receive_up, receive_dn, calculating, transmit_up, transmit_dn);

    constant address_mask2:
std_logic_vector:=conv_std_logic_vector(unsigned(address_MASK)+block_width**2,16);

    signal input_array: memory8bits(0 to block_width**2-1);
    signal output_array: memory16bits(0 to block_width**2-1);
    signal current_state: system_modes;

    signal status_register: std_logic_vector(7 downto 0);
    signal control_register: std_logic_vector(7 downto 0);
    signal initiate_processing: std_logic;

    --signal reset, m_clk, tb_clk: std_logic;
    signal module_ready: std_logic;
    signal input_available, output_available: std_logic;
    signal async_in, async_out: std_logic;
    signal datain: std_logic_vector(7 downto 0);
    signal dataout: std_logic_vector(15 downto 0);
    ---- memory access
    signal start_read_address : std_logic_vector(20 downto 0);
    signal start_write_address : std_logic_vector(20 downto 0);
    signal frame_width:  integer range 0 to 255;
    signal frame_height:  integer range 0 to 511;
    signal no_frames: integer range 0 to 15;
    ----
      signal module_programmed: std_logic;

      signal mem_gen_state: integer range 0 to 15; -- state machine of up to 16 states

      signal i:  integer range 0 to 1023;         --10 bits
      signal j:  integer range 0 to 255;          --4 bytes at a time
      signal ii: integer range 0 to block_width-1;
      signal jj: integer range 0 to (block_width/4-1); -- block bytes
      signal voffset: integer range 0 to 2**12-1;
      signal fn: integer range 0 to 22;
      signal YUV: std_logic;
      signal read_addr, write_addr : integer range 0 to 2**20-1;

    component calc_sum_product
    generic(no_of_inputs: positive:=64);
        port(
            reset:           in std_logic;
            clock:           in std_logic;
            module_ready:        out std_logic;
            input_available: in std_logic;
            datain:      in std_logic_vector(7 downto 0);
            output_available:    out std_logic;
            dataout:         out std_logic_vector(15 downto 0);
            async_in:        in std_logic;
            async_out:       out std_logic
        );
    end component;

begin

    U_calc_sum_product: calc_sum_product
```


--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2009 – All rights reserved
Not for Resale

```
        generic map (no_of_inputs => block_width**2)
        port map (
             reset => reset,
             clock => m_clk,
             module_ready => module_ready,
             input_available => input_available,
             datain => datain,
             output_available => output_available,
             dataout => dataout,
             async_in => async_in,
             async_out => async_out
        );

status_register <=(0=> module_ready, 1=> output_available, others=>'0');

LAD_interface: process(reset, b_clk)
    variable counter: integer range 0 to block_width**2-1;
begin
  if reset='1' then
       LAD_dataout <=(others=>'0');
       LAD_strobe_out <='0';
       counter:=0;
       start_read_address <= (others=>'0');
       start_write_address <= (others=>'0');
       frame_width <=0;
       frame_height <=0;
       no_frames <=0;
       module_programmed<='0';

  elsif rising_edge(b_clk) then
        LAD_strobe_out <='0';
        if (LAD_inStrobe = '1') then
             if ((LAD_Address and address_MASK) = BASE_address) then

                  if ((LAD_Address and address_mask2)=BASE_address) then
                       if LAD_Write = '1' then       -- data block
                            input_array(counter) <= LAD_datain(7 downto 0);
                            counter:=(counter+1) mod (block_width**2);
                            if counter=0 then
                               no_frames <= 0; -- 0 means block mode
                               module_programmed <='1';
                            end if;
                       else
                         LAD_dataout <= X"0000" & output_array(counter);
                           LAD_strobe_out <='1';
                           counter:=(counter+1) mod (block_width**2);
                       end if;
                  else
                       if LAD_write='1' then    -- control block
                            case LAD_Address(1 downto 0) is
                                 when"00" =>
             frame_width <= conv_integer(unsigned(LAD_datain(9 downto 2)));
             frame_height <= conv_integer(unsigned(LAD_datain(19 downto 10)));
             no_frames <= conv_integer(unsigned(LAD_datain(23 downto 20)));
                                 when"01" =>
                       start_read_address <= LAD_datain(20 downto 0);
                                 when"10" =>
                       start_write_address <= LAD_datain(20 downto 0);
                                 when"11" =>
                       module_programmed <='1';
```

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
--`,,```,,,,````-`-`,,`,,`,`,,`---
Not for Resale

```
                        when others => null;
                    end case;
                else
                    LAD_dataout <= X"000000" & status_register;
                    LAD_strobe_out <='1';
                end if;
            end if;


        end if;  -- ends check addressing
    else
        module_programmed<='0';
    end if;  -- ends check input strobe
end if;   -- ends reset or bus clock
end process LAD_interface;


    stimuli: process(reset, b_clk)
        variable counter: integer range 0 to block_width**2-1;
    begin
        if reset='1' then
            counter :=0;
            current_state <=idle;
            input_available <='0';
            datain <=(others=>'0');
        elsif rising_edge(b_clk) then
            case current_state is
                when idle =>
                    if initiate_processing='1' then
                        if module_ready='1' then
                            input_available <='1';
                            current_state <= receive_up;
                        end if;
                    end if;
                when receive_up =>
                    if async_out='0' then
                        async_in <='1';      -- latch
                        datain <=input_array(counter);
                        current_state <= receive_dn;
                    end if;
                when receive_dn =>
                    if async_out='1' then
                        async_in <='0';
                           counter :=((counter+1) mod (block_width**2));
                        if counter=0 then
                            input_available <='0';
                            current_state <= calculating;
                        else
                            current_state <= receive_up;
                        end if;
                    end if;
                when calculating =>
                    if output_available='1' then
                        current_state <= transmit_up;
                    end if;
                when transmit_up =>
                    if async_out='1' then -- latch
                        output_array(counter) <= dataout;
                        async_in <='1'; --acknowledged
                        current_state <= transmit_dn;
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2009 – All rights reserved

```
                    end if;
                when transmit_dn =>
                    if async_out='0' then
                        async_in <='0';
                        counter :=((counter+1) mod (block_width**2));
                        if counter=0 then
                            current_state <=idle;
                        else
                            current_state <=transmit_up;
                        end if;
                    end if;
            end case;
        end if;

    end process stimuli;
-------------------------------------------
  memory_generate_address: process(reset,b_clk)
     variable buffercounter: integer range 0 to (block_width**2-1);
     variable base_ad1 : integer range 0 to 2**20-1;
     variable bufferfilled, more_addressing: std_logic;
  begin
    if (reset='1') then
            -- This module will access memory
        enable_write <='0';
        enable_read <='0';
        access_request <='0';

        initiate_processing <='0';

        mem_gen_state<=0;
        module_done <='1';

        i<=0; j<=0; ii<=0; jj<=0;
        voffset<=0;
        YUV<='0'; fn<=0;
        read_addr <=0; write_addr <=0; base_ad1:=0;
        buffercounter:=0;
        bufferfilled:='0'; more_addressing:='1';

    elsif rising_edge(b_clk) then
        case mem_gen_state is
            when 0 => --initialization
                if module_programmed='1' then
                    if no_frames=0 then      -- block mode
                        initiate_processing <='1';
                    else
                        module_done<='0';
                        mem_gen_state<=1;  -- generate address
                        i<=0; j<=0; ii<=0; jj<=0;
                        voffset <=0;
                        YUV<='0'; fn<=0;
                        base_ad1:=conv_integer(unsigned(start_read_address));
                        read_addr<=base_ad1;
                        write_addr<=conv_integer(unsigned(start_write_address));
                        buffercounter:=0;
                        bufferfilled:='0'; more_addressing:='1';
                        enable_read<='1';
                        enable_write<='0';
                        access_request<='1';
                    end if;
```

© ISO/IEC 2009 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
                else
                    enable_read<='0'; enable_write<='0';
                    initiate_processing <='0';
                    module_done<='1';
                    access_request<='0';
                end if;
            when 1 =>    -- remains stuck till gets memory access grant
                if access_grant='1' then
                    mem_gen_state<=2;
                end if;
            when 2 =>    --states 2, 3, 4 for filling an input buffer
               mem_gen_state <= 3;
                if (Memory_Source_Data_Valid='1') then
                input_array(buffercounter*4+3) <= mem_datain(31 downto 24);
                input_array(buffercounter*4+2) <= mem_datain(23 downto 16);
                input_array(buffercounter*4+1) <= mem_datain(15 downto 8);
                input_array(buffercounter*4) <= mem_datain(7 downto 0);
                    if (buffercounter < (block_width**2/4-1)) then
                        buffercounter:= (buffercounter+1) ;
                    else
                      buffercounter:=0;
                      bufferfilled:='1';
                    end if;
                end if;
                if more_addressing='1' then
                  if(jj<(block_width/4-1)) then
                    jj<=(jj+1);
                  else
                    jj<=0;
                    if YUV='0' then
                         voffset <= voffset+frame_width;
                     else
                         voffset <= voffset+frame_width/2;
                     end if;
                     if (ii<block_width-1) then
                       ii<=ii+1;
                     else
                       ii<=0;
                       more_addressing:='0';
                     end if;
                  end if;
                end if;
            when 3 =>
               read_addr <= (base_ad1+voffset+jj+j);
                mem_gen_state<=4;
            when 4 =>
                if (bufferfilled='1') then
                    mem_gen_state<=5; -- enough reading, process
                    initiate_processing <='1';
                    enable_read <= '0';
                else
                     mem_gen_state<=2;   -- end cycle waste
                end if;

            when 5 =>

                mem_gen_state<=6;
                if ( ((j+block_width/4)<frame_width and YUV='0') or
((j+block_width/4)<(frame_width/2))) then
                     j <= j+block_width/4;
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2009 – All rights reserved

```
                else
                     j<=0;
                     base_ad1:=base_ad1+voffset;
                     if (i<frame_height-block_width-1) then
                        i<=i+block_width; --assumption
                     else
                        i<=0; fn<=fn+1;
                        YUV <= not YUV;
                     end if;
                end if;

            when 6 => --go process
                initiate_processing<='0';
                if current_state=idle then
                    mem_gen_state<=7;
                end if;

            when 7 => --writing results to memory
                enable_write<='1';
                mem_dataout <= output_array(buffercounter*2) & output_array(buffercounter*2+1);
                write_addr <= write_addr+1;
                if buffercounter<(block_width**2/2-1) then
                    buffercounter:=(buffercounter+1);
                else
                    buffercounter:=0;
                    voffset<=0;
                    mem_gen_state<=8;
                end if;

            when 8 =>
              enable_write<='0';
              jj<=0;
              if (fn=(no_frames*2)) then
                 mem_gen_state<=10;
              else
                 mem_gen_state<=9;
                 bufferfilled:='0'; more_addressing:='1';
              end if;
            when 9 =>
               enable_read<='1';
                read_addr <= base_ad1+j;
                mem_gen_state<=3;
            when 10 =>
                if (Memory_Source_Data_Valid='0') then
                    mem_gen_state<=0;
                end if;
            when others => null;
        end case;
    end if;

 end process memory_generate_address;
 read_address<=conv_std_logic_vector(read_addr,21);
 write_address<=conv_std_logic_vector(write_addr,21);

end behav;
```

Figure 29 — Architecture of Second Controller Designed for Integration with Framework.

In Figure 29 the orange lines shows the parameter that user needs to transfer to enable user IP to access to data stored into SRAM. As described into calc_ctrlr.vhd, the input data are loaded from the SRAM, then processed, and finally the results transfers into SRAM, a new processing can then start.

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

For example described into host_arch.vhd:

```
    dData(0) := 16#101820#;       -- Mode = 00, frame_width = 0000 1000,

                                  -- frame_height = 00 0000 1000,

-- no_frames = 0001

     dData(1) := 16#0000#;        -- Mode = 01,

-- start_read_address = 0000 0000 0000 1000 0000

        dData(2) := 16#0081#;     -- Mode = 10,

-- start_write_address = 0000 0000 0000 1000 0001

        dData(3) := 16#0003#;     -- Mode = 11,

-- module_programmed ='1';
```

The code for this controller might seem complicated, however, the structure is consistent with the idea of code reuse and all the complexity is actually concentrated in the part for calculating how to partition a raster stored YUV image into macro blocks.

7.5.1 Adding a wrapper for a Verilog module

For instantiating a Verilog ip-core, the wrapper would be exactly the same as in the above two examples. To write the component part we may use the utility “vgencomp” which translates the interface of the Verilog component to VHDL so that it can be instantiated from within a higher level VHDL system. This can also be done manually.

7.5.2 Integrating module controllers within the PE system

The next step is to add the previously described blocks within the PE (processing element) architecture. This involves instantiation of each of the two controllers and connecting them within the system interrupt-chain.

This involves adding/modifying the following code fragments to the “PE system” file:

1. Library declarations.
2. Constants for generics and interrupt signal.
3. Component declaration.
4. VHDL configuration.
5. Component instantiation.
6. Connecting interrupt signal.
7. Updating simulation and synthesis project files.

The details of these steps are as follows

7.5.3 Library declarations

This part is for adding the library declaration for the components and the component controllers defined in the previous sections. The next fragment is added to the library section in the “PE system” VHDL file. It is for 3 blocks: block-move, fifo and calc.

```
////////////////////////////////////////////////////////////////////////////////
// The processing element pe is constituted of tree element
// (instanced in the pe_arch.vhd) first example (no_of_hwas =3):
// 1) calc_hw_module_controller,
```


© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

// 2) fifo_hw_module_controller
// 3) bm_hw_module_controller

```
-----------------------------------
-- hardware accelerator libraries
library calc_lib;
    use calc_lib.all;
library fifo_lib;
    use fifo_lib.all;
library bm_lib;
    use bm_lib.all;
```

Figure 30 — Example of library declarations.

7.5.4 Constants for generics and interrupt signals

For every block we add generics for the base address and the address mask and any other parameters. The optional interrupt signal is hw?_done. The choice of the base address is arbitrary but address spaces should not overlap. The constant “no_of_hwas “ is used to define the memory and LAD bus multiplexers so it must be set correctly.

```
    constant no_of_hwas: positive:=3; -- update

    constant hw1_BASE_ad   : std_logic_vector(15 downto 0) := x"0180";
    constant hw1_MASK   : std_logic_vector(15 downto 0) := x"FFE0";  -- 32 registers
    signal hw1_done: std_logic;

    constant hw2_BASE_ad   : std_logic_vector(15 downto 0) := x"0100";
    constant hw2_MASK   : std_logic_vector(15 downto 0) := x"FFE0";  -- 32 registers
    signal hw2_done: std_logic;

    constant hw3_BASE_ad   : std_logic_vector(15 downto 0) := x"0140";
    constant hw3_MASK   : std_logic_vector(15 downto 0) := x"FF40";  -- 64 registers
    signal hw3_done: std_logic;
```

Figure 31 — Constant declarations. (described in pe_arch.vhd).

7.5.5 Component declaration

This is a step before component instantiation. It is added to the architecture section before the “begin” keyword.

For example the component instantiation for “blockmove“ is as follows.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
component bm_hw_module_controller
   generic(
        BASE_address: std_logic_vector(15 downto 0);
        address_MASK: std_logic_vector(15 downto 0):=X"FF40"; -- 64 registers
        words_per_block: positive:=64
    );
    port(
      reset: in std_logic;
      m_clk: in std_logic;       -- module clock
      b_clk: in std_logic;       -- bus clock
      module_done: out std_logic;
      ----------------------- memory access arbiter
      write_address: out std_logic_vector(20 downto 0);
      read_address: out std_logic_vector(20 downto 0);
      enable_write: out std_logic;
      enable_read : out std_logic;
      access_request: out std_logic;
      access_grant: in std_logic;
      Memory_Source_Data_Valid: in std_logic;
      mem_datain : in std_logic_vector(31 downto 0);
      mem_dataout : out std_logic_vector(31 downto 0);
      -----------------------   interface with LAD bus
      LAD_instrobe: in std_logic;
      LAD_address: in std_logic_vector(15 downto 0);
      LAD_write: in std_logic;
      LAD_datain: in std_logic_vector(31 downto 0);
      LAD_dataout: out std_logic_vector(31 downto 0);
      LAD_strobe_out: out std_logic
    );
end component;
```

Figure 32 — Component instatiations of “blockmove”.

7.5.6 VHDL configuration statements

The following simple VHDL configuration statements should be added before component instantiation. More elaborate types of VHDL configuration can be used.

The code for the three modules is as follows:

```
-------------------------------------------------------VHDL configuration
for u_hw1: calc_hw_module_controller use entity calc_lib.calc_hw_module_controller;
for u_hw2: fifo_hw_module_controller use entity fifo_lib.fifo_hw_module_controller;
for u_hw3: bm_hw_module_controller use entity bm_lib.bm_hw_module_controller;
-------------------------------------------------------
```

Figure 33 — Example of VHDL configuration statements.

7.5.7 Component instantiation

This involves a generic map and a port map. Note that mapping to the LAD multiplexer starts with “4” not zero because there are other 4 bus clients in the system that use the numbers zero to three. Thus, the first “hwa” is bus client number four. However, for memory access, the numbering starts from zero because the memory multiplexer is connected only to the user modules.

Component instantiation for the modules is as follows:

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
u_hw1: calc_hw_module_controller
            generic map(
                BASE_address => hw1_BASE_ad,
                address_MASK => hw1_MASK,
                block_width =>4
            )
            port map(
                reset => elaborate_Reset,
                m_clk => m_Clk,
                b_clk => b_Clk,
                module_done => hw1_done,
      ------------------------ memory access arbiter
                write_address => write_addresses(0),
                read_address => read_addresses(0),
                enable_write => write_requests(0),
                enable_read => read_requests(0),
                access_request => access_requests(0),
                access_grant => access_grants(0),
                Memory_Source_Data_Valid => Memory_Source_Data_Valid,
                mem_datain => Memory_Source_Data_Out,
                mem_dataout => write_datae(0),
      ------------------------   interface with LAD bus
                LAD_instrobe => LAD_instrobe,
                LAD_address => LAD_address,
                LAD_write => LAD_write,
                LAD_datain => LAD_datain,
                LAD_dataout => LAD_Bus_Data_Out_Vector(4),
                LAD_strobe_out => LAD_Bus_Strobe_Out_Vector(4)
            );

    u_hw2: fifo_hw_module_controller
            generic map(
                BASE_address => hw2_BASE_ad,
                address_MASK => hw2_MASK,
                fifo_size => 16,
                fifo_width => 16
            )
            port map(
                reset => elaborate_Reset,
                m_clk => m_Clk,
                b_clk => b_Clk,
                module_done => hw2_done,
      ------------------------ memory access arbiter
                write_address => write_addresses(1),
                read_address => read_addresses(1),
                enable_write => write_requests(1),
                enable_read => read_requests(1),
                access_request => access_requests(1),
                access_grant => access_grants(1),
                Memory_Source_Data_Valid => Memory_Source_Data_Valid,
                mem_datain => Memory_Source_Data_Out,
                mem_dataout => write_datae(1),
      ------------------------   interface with LAD bus
                LAD_instrobe => LAD_instrobe,
                LAD_address => LAD_address,
                LAD_write => LAD_write,
                LAD_datain => LAD_datain,
                LAD_dataout => LAD_Bus_Data_Out_Vector(5),
                LAD_strobe_out => LAD_Bus_Strobe_Out_Vector(5)
            );
```

Figure 34 — Example of components instatiations of modules.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.5.8 Connecting Interrupt signals

The interrupt status register is used to identify which modules are interrupt sources. The signals hw?_done are connected to this register. The signal connection start by number 4 because again there are other predefined 4 blocks as stated in the previous section.

```
    interrupt_status_reg <= (
        0 => DMA_Source_Done,
        1 => Memory_Destination_Done,
        2 => Memory_Source_Done,
        3 => DMA_Destination_Done,
        4 => hw1_done,
        5 => hw2_done,
        6 => hw3_done,
        others => '1'
    );
```

Figure 35 — Interrupt status instatiations.

7.5.9 Updating simulation and synthesis project files

The compilation project file for simulation is located in the “sim” folder. The following is added to the file “project_vcom.do”. The modifications are just calls to the macro do files used in testing the ip-modules. Comment lines start with two hyphens “--”

```
-----------------------------------------------------------------
--next is the ip-cores
--

do $PROJECT_BASE/calc/compile_my_module.do
do $PROJECT_BASE/fifo/compile_my_module.do
do $PROJECT_BASE/blockmove/compile_my_module.do
```

Figure 36

The Synplify synthesis project file is located in the “syn” folder. The following is added to the file “pe.prj”. Comment lines start with a hash “#.”

```
#-------------------------------------------------------------------------
#-     Add your project's PE architecture VHDL file here, as
#-     well as any VHDL or constraint files on which your PE
#-     design depends:

add_file -vhdl -lib calc_lib $PROJECT_BASE/calc/calc_sum_product.vhd
add_file -vhdl -lib calc_lib $PROJECT_BASE/calc/calc_ctrlr.vhd

add_file -vhdl -lib fifo_lib $PROJECT_BASE/fifo/fiforeg2.vhd
add_file -vhdl -lib fifo_lib $PROJECT_BASE/fifo/fifo_ctrlr2.vhd

add_file -vhdl -lib bm_lib $PROJECT_BASE/blockmove/bm_ctrlr.vhd
```

Figure 37

Note that we do not synthesis test-bench files.

Also note that after synthesis, the final step is placement and routing done by Xilinx ISE tool using the batch file “syn/place_and_route.bat”

7.6 Simulation of the whole system

To verify the system operation before synthesis we write a file that simulates the host operations which are mainly interactions with the wildcard board.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2009 – All rights reserved
Not for Resale

We verify each block by sending it the required data and control parameters via the data bus and wait for its response. The system detects the response by querying status registers implemented within the design or by detecting an interrupt. Interrupts are controlled by a software programmable interrupt at hardware address 0x1000. An example host simulation file is given next. The interaction is using API like function calls, namely WC_peregRead and WC_peregWrite and other DMA related API functions.

Note that the host simulation is like a testbench to demonstrate how the whole system should function. Thus this file, and other files that simulate other chips on the WildCard board, are not part of the synthesis project. Modelsim main window is used to mimic a console and the results of the interaction are displayed by a series of “report” statements as shown here.

```
# ** Note: Testing block move
#    Time: 545 ns  Iteration: 0  Instance: /system/u_host
# ** Note: PE Block move Data ( 0) = 00000000
#    Time: 26209424 ps  Iteration: 0  Instance: /system/u_host
# ** Note: PE Block move Data ( 1) = 00000001
#    Time: 26209424 ps  Iteration: 0  Instance: /system/u_host
# ** Note: PE Block move Data ( 2) = 00000002
#    Time: 26209424 ps  Iteration: 0  Instance: /system/u_host
# ** Note: PE Block move Data ( 3) = 00000003
#    Time: 26209424 ps  Iteration: 0  Instance: /system/u_host
# ** Note: PE Block move Data ( 4) = 00000004

# ** Note: Testing calculator
#    Time: 26209424 ps  Iteration: 0  Instance: /system/u_host
# ** Note: Calc Data ( 0) = 1
#    Time: 35449424 ps  Iteration: 0  Instance: /system/u_host
# ** Note: Calc Data ( 1) = 0
#    Time: 35449424 ps  Iteration: 0  Instance: /system/u_host
# ** Note: Calc Data ( 2) = 5
#    Time: 35449424 ps  Iteration: 0  Instance: /system/u_host
# ** Note: Calc Data ( 3) = 6
#    Time: 35449424 ps  Iteration: 0  Instance: /system/u_host
# ** Note: Calc Data ( 4) = 9
#    Time: 35449424 ps  Iteration: 0  Instance: /system/u_host
# ** Note: Calc Data ( 5) = 20
#    Time: 35449424 ps  Iteration: 0  Instance: /system/u_host
# ** Note: Calc Data ( 6) = 13
#    Time: 35449424 ps  Iteration: 0  Instance: /system/u_host
# ** Note: Calc Data ( 7) = 42

# ** Note: Received Interrupt Indicating transfer to SRAM complete
#    Time: 51959 ns  Iteration: 0  Instance: /system/u_host
# ** Note: Retrieving Data by DMA From SRAM
#    Time: 51959 ns  Iteration: 0  Instance: /system/u_host
# ** Note: Received Interrupt Indicating DMA from SRAM complete
#    Time: 59351 ns  Iteration: 0  Instance: /system/u_host
# ** Note: Word(0) Sent :03020100 Received :03020100
#    Time: 59351 ns  Iteration: 0  Instance: /system/u_host
# ** Note: Word(1) Sent :07060504 Received :07060504
#    Time: 59351 ns  Iteration: 0  Instance: /system/u_host
# ** Note: Word(2) Sent :0B0A0908 Received :0B0A0908
#    Time: 59351 ns  Iteration: 0  Instance: /system/u_host
…
# ** Note: End of Iteration 0
#    Time: 59351 ns  Iteration: 0  Instance: /system/u_host
# ** Note: This is the Finish Line
#    Time: 59351 ns  Iteration: 0  Instance: /system/u_host
```

Figure 38

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.7 Debug Menu

An example of a simple debug menu with the ability to add more options is shown here. This menu and its function calls are written in ANSI C as strictly as possible for compatibility with other platforms. The main options cover testing blocks of data to the FPGA, moving blocks of data to the card memory via control blocks on the FPGA and testing other integrated hardware accelerators with different parameters and test vectors.

The data source is either random or through an input file specified in a command line argument. The output of each test is expected to be equivalent to the output from the simulation "host" file. If this is not the case then the report generated by the synthesizer should be revised to inspect for removed/misinterpreted logic.

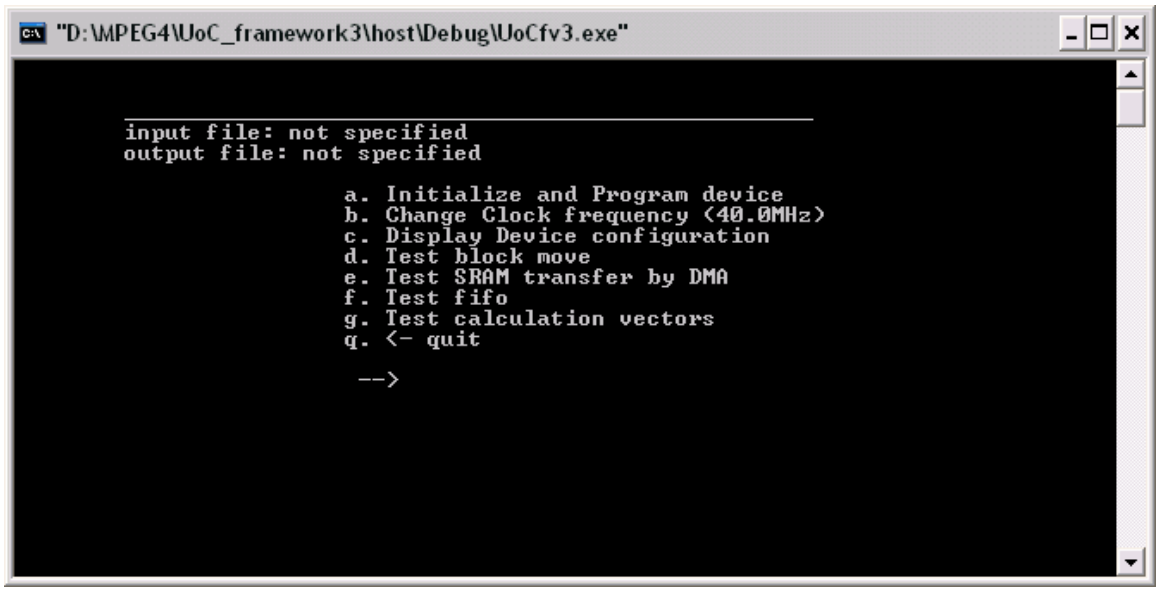


Figure 39 — Display of debug menu.

The submenu a, b are used to configure the platform. The sub-menu c enables to check configuration. The stage b and c are optional. The others sub-menus permit to test the different transfer modes available with the demonstration examples. The examples described into this document are accessible with submenu f and g. Two extra submenu are available to transfer data into SRAM, block move and test SRAM transfer by DMA.

_Annexe 1:

```
////////////////////////////////////////////////////////////////////////////////
// These two functions used to transfer data or parameters are
// their structure is declared as follow in wcdefs.h
WC_PeRegRead ( WC_DeviceNum   device,
            DWORD        dDwordOffset,
            DWORD        numDwords,
            DWORD       *pDestVirtAddr );


WC_PeRegWrite( WC_DeviceNum   device,
            DWORD        dDwordOffset,
            DWORD        numDwords,
            DWORD       *pSrcVirtAddr );

////////////////////////////////////////////////////////////////////////////////
// Structure of status register declared in calc_ctrl.vhd
status_register <=(0=> module_ready, 1=> output_available, 2=> module_programmed, others=>'0');
```


--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8 Integrated Framework supporting the "Virtual Socket" between HDL modules described in Part 9 and the MPEG Reference Software (Implementation 3).

8.1 An Integrated Virtual Socket Hardware-Accelerated Co-design Platform for MPEG-4

8.1.1 Scope

This clause presents the integrated virtual socket hardware-accelerated co-design platform and its conformance testing developed at the University of Calgary for MPEG-4 implementation. In this co-design platform framework, generic system hardware blocks are used for testing, data communication and control signal exchange between a host computer and a configured co-processor based on the concept of virtual socket. These functional blocks employ four types of virtual socket memory spaces inside the platform to transfer data and thus provide the infrastructure for the purpose of user IP blocks that were not specifically designed for interaction with a host computer system. The user IP block can be any hardware module designed to handle a computationally extensive task for MPEG-4 implementation.

The point of this subclause is to stress the concept of having a generic virtual socket co-design platform to illustrate the possibility of rapidly developing a working system for MPEG-4 video applications.

8.1.2 Introduction

In order to take full advantages of the hardware and software design solution, we have developed an integrated co-design platform which is used for MPEG-4 implementation, because a set of the MPEG-4 IP blocks can be integrated into the platform framework. The user IP block will be any hardware module defined by MPEG-4 Part 9 Reference Hardware Description, e.g DCT, IDCT, Quantization, Inverse Quantization, Motion Estimation, Motion Compensation etc. With the integration of such user IP blocks, the developed co-design platform can achieve the ultimate goal for MPEG-4 Part 9 Reference Hardware implementation.

Therefore, the primary concerns for this platform design are (1) to develop an integrated co-design platform framework which can accommodate MPEG-4 IP blocks, (2) integrate some of the video processing IP blocks into the platform framework, and (3) develop prototyping virtual socket API function calls to implement the platform interface features.

While most of the recent co-design solutions for MPEG-4 video codec implementation are based on RISC and DSP architectures, this proposed co-design platform framework is constructed with a simplified architecture based on the conception of virtual socket. The design of the hardware platform framework is prototyped with VHDL and implemented on Annapolis WildCard-II FPGA device.

The software part of the design is done by C/C++, particularly for the design of virtual socket Application Programming Interface (API) function calls. Because the salient feature of co-design is the cooperation of hardware and software modules, the hardware-software portioning for this design is also simplified so that the data flow of the hardware module performance is under the control of host programs.

8.1.3 A Hardware-Software Design Flow

There is a design flow shown in Figure 40 for proposed co-design solution. It describes the overall design process for platform system and how the co-design platform can be developed step by step.

(1) The design flow starts with the definition of system specifications, constraints and requirements. For this co-design platform system, we assume the target design specifications, ultimate goal to implement the MPEG-4 Part 9 Reference Hardware Description, listed in Table 4.

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 4 — Design Specifications.

- Simple Profile Levels 0, 1, 2, 3.
- Resolution up to CIF encoding at 30 fps.
- 4: 2: 0 YCrCb format.
- Hardware Accelerators compatible with MPEG-4 Part 9 Reference Hardware.
- Hardware Platform System Frequency 48MHz.
- The Maximum IP Blocks to be integrated up to 32.
- Register Size: 16 double words for each IP Block.
- Block RAM Size: 512 double words for each IP Block.
- SRAM Size: Total up to 2MB.
- DRAM Size: Total up to 32MB.

(2) An architecture of the platform system is determined according to the design specifications, constraints and requirements. Particularly, the architecture design should take a consideration of the hardware integration for a variety of MPEG-4 hardware accelerators.

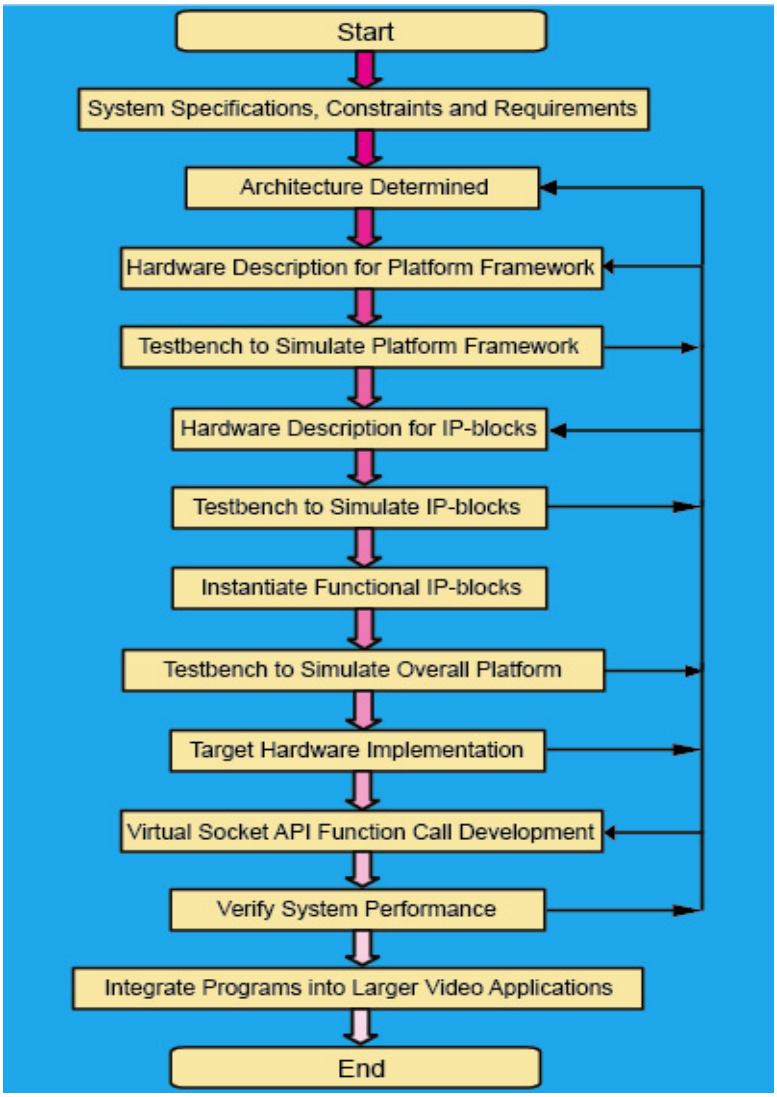


Figure 40 — Hardware-Software Co-design Flowchart.

(3) Write a hardware description for the platform framework with VHDL, and thus generate hardware functional blocks to implement the hardware system features.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

(4) Write the testbench to simulate and verify the platform system features. If the simulation is successfully passed, then go to the next step.

(5) The step from 2 to 4 are concerned with the implementation of platform framework features, and the next steps 5 to 6 are related to the design of user IP blocks. In step 5, a hardware description of the individual IP block is given. Actually, the hardware description for such IP blocks can be prototyped with VHDL or Verilog.

(6) Write the testbench to simulate and verify the functionality of IP blocks. If the simulation is successfully passed, then go to the next step.

(7) After the simulation and verification for IP blocks, the following steps are taken to implement the overall platform system functionality. The step 7 is to instantiate and integrate all the functional IP blocks in the platform framework.

(8) Write the testbench for overall system simulation and verification. If the simulation is successfully passed, then go to the next step.

(9) Target hardware implementation, including system synthesis, FPGA place and route, timing analysis and verification.

(10) Now coming to the software implementation, including the design of virtual socket interface, virtual socket API function calls, and conformance testing programs with the debug menu.

(11) Verify and analyze the overall system performance. If the system specifications are met, then go to the next step.

(12) Integrate all the programs to form a larger video application.

8.1.4 System Block Diagram

The platform system architecture template is depicted in Figure 41. The architecture is based on the WildCard-II FPGA platform device.

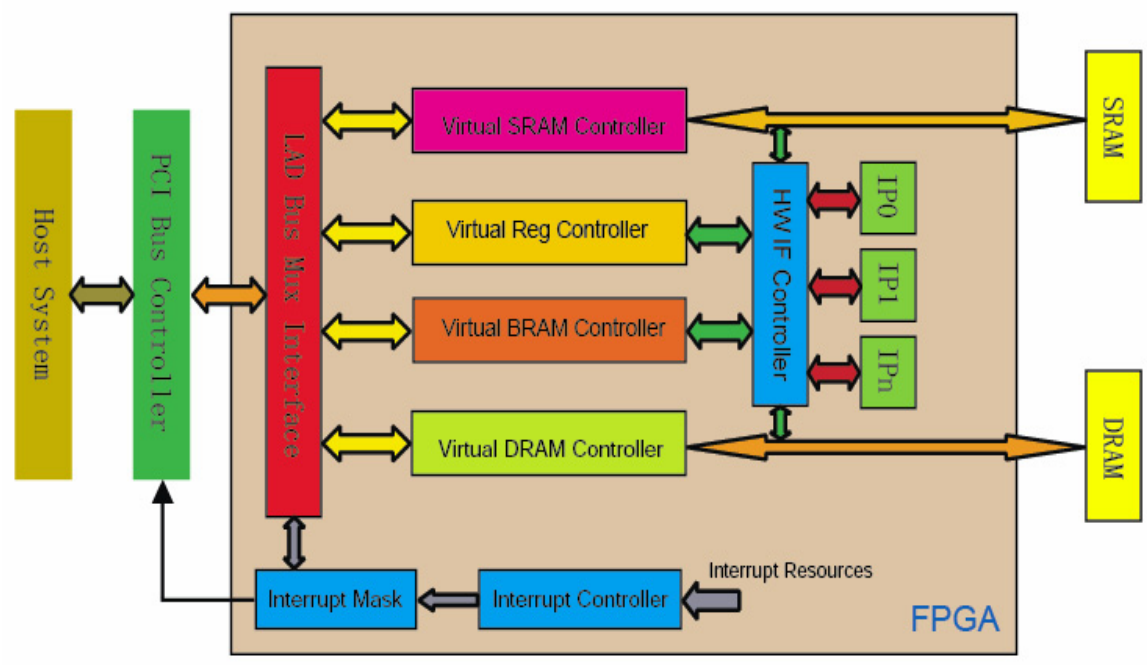


Figure 41 — Virtual Socket Co-design Platform Architecture Template.

It will develop two abstraction layers for the system design. One layer is constructed at the hardware level on WildCard-II FPGA platform which performs the hardware features for the system implementation, and the

© ISO/IEC 2009 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

other is at the software level on host computer system, which deals with a virtual socket interface between the host computer system and hardware platform framework.

The PCI Bus Controller is developed by Annapolis and has been integrated in the WildCard-II. Other than the PCI Bus Controller, the infrastructure for hardware includes FPGA and physical memory spaces. Registers and Block RAM as two types of the memory spaces are integrated in the FPGA as its internal design resources. Other two types of the external memory spaces are ZBT-SRAM and DDR-DRAM. Therefore, there are totally four types of the physical memory spaces inside the platform framework. Besides, the hardware system functional blocks are generated and integrated in the platform framework to perform data communication and signal control between the host computer system and physical memory space. As shown in Figure 41, each generic virtual memory controller is in charge of the address mapping and access operations for corresponding memory storage. All the user IP blocks are connected to a hardware interface controller, which provides a straight way for the user IP blocks, i.e hardware accelerators, to exchange data between the host system and memory spaces. Interrupt signals can be generated when IP blocks finish the data processing or exchange. In this case, the interrupt controller is provided to deal with the user IP block interrupt generation.

8.1.5 The DRAM Access

The design of the platform system is based on Annapolis WildCard-II FPGA device. According to the hardware structure of the WildCard-II [5], there is an external memory space, i.e DDR-DRAM, which is integrated in the card.

Any MPEG-4 video application is truly data dependent, and the user video IP blocks need to access the video data all the time. For this proposed co-design platform system, the video data will be stored in the memory spaces inside the hardware platform. In such case, although Registers, BRAM and SRAM were previously developed as the destination memory spaces which can store the video data, we still need another large-sized memory space, that is DDR-DRAM, to accommodate more video data at a time. Therefore, the DRAM access control becomes very import for the system design.

There will be three modes for DRAM data access according to the platform system design, namely non-DMA DRAM transfer, DMA DRAM transfer and IP block DRAM data access. Accordingly, the hardware development of DRAM access controllers includes the design of Non-DMA DRAM Controller, DMA DRAM Controller and IP Block DRAM Controller. Besides, there will be some virtual socket API function calls developed for the software interface control of the DMA transfer regarding DRAM data access.

8.1.5.1 Interface Control of DRAM Access

Three modes are applied for the DRAM data access. For each of the access mode, we need a set of the corresponding virtual socket API function calls to deal with it. Actually, the following API function calls will be employed, and Figure 42 to Figure 44 show the example software codes for the DRAM data access control.

Mode 1 : Non-DMA DRAM Data Transfer

(1) VSWrite

(2) VSRead

Mode 2 : DMA DRAM Data Transfer

(1) VSDMAWrite

(2) VSDMARead

Mode 3 : IP Block DRAM Data Access

(1) VSWrite / VSDMAWrite

(2) VSRead / VSDMARead

(3) WC_PeRegWrite

(4) HW_Module_Controller

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
int main(int argc, char *argv[])
{
//necessary variable definition and initialization

rc = WC_Open (TestInfo.DeviceNum, 0);

//other possible software processing or operations…

HWAccel = 0;
Offset = 0xf0000000 + Offset_Addr;
Count = 256;
rc = VSWrite (&TestInfo, HWAccel, Offset, Count, Buffer);
rc = VSRead (&TestInfo, HWAccel, Offset, Count, Buffer);

//other possible software processing or operations…

rc = WC_Close (TestInfo.DeviceNum);

return rc;

}
```

Figure 42 — An Example Software Code for DRAM Access Mode 1.

```
int main(int argc, char *argv[])
{
//necessary variable definition and initialization

rc = WC_Open (TestInfo.DeviceNum, 0);

//other possible software processing or operations…

BSDRAM = 2;
StartAddr = 0x00000000;
Count = 256;
rc = VSDMAWrite (&TestInfo, BSDRAM, StartAddr, Count, Buffer);
rc = VSDMARead (&TestInfo, BSDRAM, StartAddr, Count, Buffer);

//other possible software processing or operations…

rc = WC_Close (TestInfo.DeviceNum);

return rc;

}
```

Figure 43 — An Example Software Code for DRAM Access Mode 2.

```
int main(int argc, char *argv[])
{
//necessary variable definition and initialization
rc = WC_Open (TestInfo.DeviceNum, 0);
//other possible software processing or operations…
HWAccel = 0;
Offset = 0xf0000000;
Count = 16;
rc = VSWrite (&TestInfo, HWAccel, Offset, Count, Buffer);
Memory_Type[0] = 0xf0000000;
rc = WC_PeRegWrite (TestInfo.DeviceNum, HW_Mem_HWAccel, 1,
                                    Memory_Type);
HW_IP_Start = 0x00000001;
rc = HW_Module_Controller (&TestInfo, HW_IP_Start);
rc = VSRead (&TestInfo, HWAccel, Offset, Count, Buffer);
//other possible software processing or operations…
rc = WC_Close (TestInfo.DeviceNum);
return rc;

}
```

Figure 44 — An Example Software Code for DRAM Access Mode 3.

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.6 The Extended IP Block Interrupt Mechanism

To take the advantage of user IP block integration features, the platform itself needs an interrupt generation logic to deal with the multiple IP block interrupt signals.

The co-design platform is implemented on Annapolis WildCard-II FPGA platform device, and actually there is only one physical interrupt signal can be generated from the platform to the host computer interface on WildCard-II. Because it is supposed that there will be a maximum of 32 user IP blocks to be integrated in the platform framework, an extended interrupt solution is thus introduced to deal with this issue.

The hardware design of the proposed interrupt mechanism will be prototyped with VHDL, and the software solution for the host interrupt interface control is coded in C/C++ language. Actually, the interrupt mechanism includes an interrupt chain, Interrupt Status Register and Interrupt Mask Register designed in the hardware platform system, while the software design utilizes the virtual socket API function call features [5] to connect interrupts with the host interface. The Interrupt Status Register and Interrupt Mask Register are 32-bit wide, and each register bit represents a corresponding user IP block numbered from 0 to 31.

8.1.6.1 Hardware Design

The hardware design of the interrupt mechanism covers the setup of Interrupt Status Register and Interrupt Mask Register in the platform framework. Furthermore, when user IP block needs to generate an interrupt, the corresponding interrupt signal should be added to an interrupt chain designed within the platform. Figure 45 presents an example interrupt chain prototyped with VHDL, which can deal with 3 user IP blocks in the platform framework. If more user IP blocks are integrated into the platform system, the interrupt chain can be easily expanded.

```
u_int_gen : process (I_Clk, Global_Reset)
begin
  if (Global_Reset = '1') then
Int_Counter <= "10000";

  elsif (rising_edge(I_Clk)) then
    if ((dram_hardware_module_done = '0') or
                (DMA_Destination_Done = '0') or
        (DMA_Source_Done = '0') or
        (Memory_Destination_Done = '0') or
        (Memory_Source_Done = '0') or
        (DRAM_DMA_Destination_Done = '0') or
        (DRAM_DMA_Source_Done = '0') or
        (DRAM_Memory_Destination_Done = '0') or
        (DRAM_Memory_Source_Done = '0') or
        (BRAM_DMA_Destination_Done = '0') or
        (BRAM_DMA_Source_Done = '0') or
        (BRAM_Memory_Destination_Done = '0') or
        (BRAM_Memory_Source_Done = '0') or
        ((Interrupt(0) or Interrupt_Mask(0)) = '0') or
        ((Interrupt(1) or Interrupt_Mask(1)) = '0')) or
        ((Interrupt(2) or Interrupt_Mask(2)) = '0')) then
            Int_Counter <= (others => '0');
    elsif (Interrupt_Done = '0') then
      Int_Counter <= Int_Counter + 1;
    end if;

  end if;
end process;
```

Figure 45 — Platform Interrupt Chain.

In the template codes listed above, “Interrupt” and “Interrupt_Mask” are actually the Interrupt Status Register and Interrupt Mask Register in the platform system. They are initialized and set up elsewhere in the system. Once there is an interrupt generated by a user IP block, the “Interrupt” will be set up accordingly, and the final platform interrupt signal will be determined by “Interrupt_Mask”. If “Interrupt_Mask” is ‘0’, a final interrupt signal is thus generated to the host interface. For example, if user IP block 2 finishes its processing and generates a user interrupt to the platform, the “Interrupt” will record such status, and “Interrupt(2)” is asserted consequently. If “Interrupt_Mask(2)” is ‘0’ in this case, a final platform interrupt will be generated to the host computer interface afterward. Other than the user IP block interrupts shown in the chain, there are some other interrupts added to the chain for platform internal use purpose.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.6.2 Software Design

The software design of the interrupt mechanism includes the initialization of Interrupt Status Register and Interrupt Mask Register, waiting and checking for the interrupt generation, reading Interrupt Status Register to verify which interrupt source is generating an interrupt, and cleaning up the corresponding bit in the Interrupt Status Register if necessary. Figure 46 lists an example software code to control the interrupt interface using virtual socket API function calls. Actually, the following API function calls will be needed:

(1) WC_Open

(2) VSIntStatusWrite

(3) VSIntClear

(4) VSIntCheck

(5) VSIntStatusRead

(6) WC_Close

```
int main(int argc, char *argv[])
{
//necessary variable definition and initialization

rc = WC_Open (TestInfo.DeviceNum, 0);

//other possible software processing or operations…

Interrupt_Status_Register[0] = 0x00000000;
Interrupt_Mask[0] = 0x00000000;
rc = VSIntStatusWrite (&TestInfo, Interrupt_Status_Register, Interrupt_Mask);
rc = VSIntClear (&TestInfo);

//trigger user IP blocks to work…

rc = VSIntCheck (&TestInfo);
rc = VSIntStatusRead (&TestInfo, Interrupt_Status_Register, Interrupt_Mask);

//other possible software processing or operations…

rc = WC_Close (TestInfo.DeviceNum);

return rc;

}
```

Figure 46 — An Example Software Code for Interrupt Interface Control.

8.1.7 The IP Blocks Concurrent Control

To improve the system performance and implement appropriate MPEG-4 functionality, it is very necessary to make the user IP blocks operate in parallel. Actually, this is the concurrent control for the platform design.

Therefore, a platform system concurrent arbitrator needs to be developed to deal with such issue. Because all the user video IP blocks are memory data dependent and connected to the IP hardware interface controller for memory access operations, the platform concurrent arbitrator will work with the IP hardware interface controller to allocate appropriate memory bus for user IP blocks to access the data required.

8.1.7.1 Interface Control of Concurrent Operations

Some virtual socket API function calls [5] are used to control the host interface for the platform concurrent feature. To start the concurrent operations, the host first writes some memory access information to the platform for each user IP block. The memory access information indicates the type of memory space to which the user IP block will have access. A set of the user IP blocks is triggered to work in parallel afterward. When all the IP blocks finish data processing, the host will receive an interrupt signal to indicate the end of the concurrent operations. The utilized virtual socket API function calls are listed following, and Figure 47 shows an example software code to control the concurrent operations for User IP Block 0 and User IP Block 1.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

API Function Calls:

(1) WC_PeRegWrite

(2) HW_Module_Controller

```
int main(int argc, char *argv[])
{
//necessary variable definition and initialization

rc = WC_Open (TestInfo.DeviceNum, 0);

//other possible software processing or operations…

Memory_Type[0] = 0x00000000;
rc = WC_PeRegWrite (TestInfo.DeviceNum, HW_Mem_HWAccel, 1,
                                    Memory_Type);
Memory_Type[0] = 0x50000001;
rc = WC_PeRegWrite (TestInfo.DeviceNum, HW_Mem_HWAccel, 1,
                                    Memory_Type);
HW_IP_Start = 0x00000003;
rc = HW_Module_Controller (&TestInfo, HW_IP_Start);

//other possible software processing or operations…

rc = WC_Close (TestInfo.DeviceNum);

return rc;

}
```

Figure 47 — An Example Software Code for Concurrent Operations.

8.1.8 Virtual Socket Prototype Function Calls

To implement and utilize the features of virtual socket platform, we need define some prototyping API function calls in the host software system. Recently, the improved virtual socket API function calls are defined in Table 5.

Table 5 — Virtual Socket API Function Calls.

Function Name	Description
VSInit	Initialize virtual socket platform
VSSystemReset	Reset virtual socket platform
VSUserResetEnable	Enable a user IP block reset
VSUserResetDisable	Disable a user IP block reset
VSIntEnable	Enable / Disable the interrupt generation for virtual socket platform
VSIntClear	Clear the generated platform interrupt
VSIntCheck	Check the platform interrupt
VSIntStatusWrite	Set up platform interrupt registers
VSIntStatusRead	Read platform interrupt registers
VSWrite	Write data from host to virtual socket platform by non-DMA transfer
VSRead	Read data from virtual socket platform to host by non-DMA transfer
VSDMAInt	Initialize platform DMA channel and data buffer
VSDMAClear	Release platform DMA channel and data buffer
VSDMAWrite	Write data from host to virtual socket platform by DMA transfer
VSDMARead	Read data from virtual socket platform to host by DMA transfer
HW_Module_Controller	Trigger and control user IP blocks to work

The arguments for those prototype functions are listed below:

- VSInit (WC_TestInfo *TestInfo)

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

- VSSystemReset (WC_TestInfo *TestInfo)
- VSUserResetEnable (WC_TestInfo *TestInfo)
- VSUserResetDisable (WC_TestInfo *TestInfo)
- VSIntEnable (WC_TestInfo *TestInfo, boolean IntEnable)
- VSIntClear (WC_TestInfo *TestInfo)
- VSIntCheck (WC_TestInfo *TestInfo)
- VSIntStatusWrite (WC_TestInfo *TestInfo, DWORD *IntStatus, DWORD *IntMask);
- VSIntStatusRead (WC_TestInfo *TestInfo, DWORD *IntStatus, DWORD *IntMask);
- VSWrite (WC_TestInfo *TestInfo, unsigned int HWAccel, unsigned long Offset, unsigned long Count, DWORD *Buffer)
- VSRead (WC_TestInfo *TestInfo, unsigned int HWAccel, unsigned long Offset, unsigned long Count, DWORD *Buffer)
- VSDMAInit (WC_TestInfo *TestInfo, unsigned long Count);
- VSDMAClear (WC_TestInfo *TestInfo);
- VSDMAWrite (WC_TestInfo *TestInfo, unsigned int BSDRAM, unsigned long StartAddr, unsigned long Count, DWORD *Buffer);
- VSDMARead (WC_TestInfo *TestInfo, unsigned int BSDRAM, unsigned long StartAddr, unsigned long Count, DWORD *Buffer);
- HW_Module_Controller (WC_TestInfo *TestInfo, unsigned long HW_IP_Start);

Note: All the functions will return a value defined as WC_RetCode, please refer to the user manual of WildCard II Platform for more details about it [5].

Arguments of the function calls:

```
typedef struct _TestInfo_
{
  WC_DeviceNum DeviceNum;
  WC_DevConfig   DeviceCfg;
  WC_Version       Version;
  DWORD            dIterations;
  float              fClkFreq;
  BOOLEAN         bVerbose;
} WC_TestInfo;
```

TestInfo : A Pointer to the virtual socket platform device

IntEnable : Enable or disable the interrupt generation

IntStatus : Platform Interrupt Status Register

IntMask : Platform Interrupt Mask Register

HWAccel : Hardware accelerator number in virtual socket platform

Offset : Offset of the memory address space

Count : Size of Dwords to read or write

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Buffer : A Pointer to an array of Dwords, Buffer Length ≥ Count

BSDRAM : Indicate which memory space (BRAM / SRAM / DRAM) is accessed

StartAddr : Start address in the memory space from which data are transferred

HW_IP_Start : Indicate which specific user IP block is triggered to work

Note: The highest 2 bits of “Offset” [31:30] indicate which memory space the hardware accelerators (user IP blocks) want to write by non-DMA transfer, while “Offset” [29:28] indicate which memory space the hardware accelerators want to read by non-DMA transfer.

BSDRAM indicates which memory space will be accessed by DMA operations.

Offset[31:30]	Memory Write Accessed
0 0	Register
0 1	BRAM
1 0	SRAM
1 1	DRAM

Offset[29:28]	Memory Read Accessed
0 0	Register
0 1	BRAM
1 0	SRAM
1 1	DRAM

BSDRAM	Memory Accessed
0	BRAM
1	SRAM
2	DRAM

With the definition of prototyping API function calls in the host computer system, users can have access to the physical memory space inside the platform through non-DMA or DMA transfers.

8.1.9 Integration of the User IP blocks

An improved hardware interface controller is provided for the platform framework to integrate a set of the user MPEG-4 IP blocks. All the functional IP blocks are connected to this interface controller. The controller is in charge of memory access operations for IP blocks as well as the interactions with the whole platform framework system. Besides, to control the concurrent performance of IP blocks, an IP block bus arbitrator is designed within the interface controller to allocate appropriate memory operation time for each IP block and thus avoid the memory access conflicts possibly caused by heterogeneous hardware IP modules.

As we have defined that there is a maximum of 32 user IP blocks can be integrated into the platform, it is very likely that each user IP block has its own requirement for the interface between the platform and IP block. To facilitate the platform development and make the integration much easier, we should define a common set of interface signals between the platform and each user IP block. Therefore, we have the following interface signals defined between platform and each user IP block listed in Figure 48.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
entity User_IP_Block is
generic(block_width: positive:=8);
port(
     clk                          : in std_logic;
     reset                        : in std_logic;
     start                        : in std_logic;
     input_valid                  : in std_logic;
     data_in                      : in std_logic_vector(31 downto 0);
     memory_read                  : out std_logic;
     HW_Mem_HWAccel               : out std_logic_vector(31 downto 0);
     HW_Offset                    : out std_logic_vector(31 downto 0);
     HW_Count                     : out std_logic_vector(31 downto 0);
     data_out                     : out std_logic_vector(31 downto 0);
     output_valid                 : out std_logic;
     HW_Mem_HWAccel1              : out std_logic_vector(31 downto 0);
     HW_Offset1                   : out std_logic_vector(31 downto 0);
     HW_Count1                    : out std_logic_vector(31 downto 0);
     output_ack                   : in std_logic;
     Host_Mem_HWAccel             : in std_logic_vector(31 downto 0);
     Host_Offset                  : in std_logic_vector(31 downto 0);
     Host_Count                   : in std_logic_vector(31 downto 0);
     Host_Mem_HWAccel1            : in std_logic_vector(31 downto 0);
     Host_Offset1                 : in std_logic_vector(31 downto 0);
     Host_Count1                  : in std_logic_vector(31 downto 0);
     Mem_Read_Access_Req          : out std_logic;
     Mem_Write_Access_Req         : out std_logic;
     Mem_Read_Access_Release_Req  : out std_logic;
     Mem_Write_Access_Release_Req : out std_logic;
     Mem_Read_Access_Ack          : in std_logic;
     Mem_Write_Access_Ack         : in std_logic;
     Mem_Read_Access_Release_Ack  : in std_logic;
     Mem_Write_Access_Release_Ack : in std_logic;
     done                         : out std_logic
     );
end User_IP_Block;
```

Figure 48 — Interface Signals between Platform and IP blocks.

The signal mapping of the user IP block interface can be:

```
u_myhardware_0 : User_IP_Block_0
port map(
clk                          => I_Clk,
reset                        => User_Global_Reset,
start                        => start(0),
input_valid                  => input_valid(0),
data_in                      => in_block_buffer(0),
memory_read                  => user_memory_read(0),
HW_Mem_HWAccel               => user_HW_Mem_HWAccel(0),
HW_Offset                    => user_HW_Offset(0),
HW_Count                     => user_HW_Count(0),
data_out                     => out_block_buffer(0),
output_valid                 => output_valid(0),
HW_Mem_HWAccel1              => user_HW_Mem_HWAccel1(0),
HW_Offset1                   => user_HW_Offset1(0),
HW_Count1                    => user_HW_Count1(0),
output_ack                   => output_ack(0),
Host_Mem_HWAccel             => user_Host_Mem_HWAccel(0),
Host_Offset                  => user_Host_Offset(0),
Host_Count                   => user_Host_Count(0),
Host_Mem_HWAccel1            => user_Host_Mem_HWAccel1(0),
Host_Offset1                 => user_Host_Offset1(0),
Host_Count1                  => user_Host_Count1(0),
Mem_Read_Access_Req          => user_Mem_Read_Access_Req(0),
Mem_Write_Access_Req         => user_Mem_Write_Access_Req(0),
Mem_Read_Access_Release_Req  => user_Mem_Read_Access_Release_Req(0),
Mem_Write_Access_Release_Req => user_Mem_Write_Access_Release_Req(0),
Mem_Read_Access_Ack          => user_Mem_Read_Access_Ack(0),
Mem_Write_Access_Ack         => user_Mem_Write_Access_Ack(0),
Mem_Read_Access_Release_Ack  => user_Mem_Read_Access_Release_Ack(0),
Mem_Write_Access_Release_Ack => user_Mem_Write_Access_Release_Ack(0),
done                         => user_done(0)
);
```

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
u_myhardware_1 : User_IP_Block_1
port map(
clk                          => I_Clk,
reset                        => User_Global_Reset,
start                        => start(1),
input_valid                  => input_valid(1),
data_in                      => in_block_buffer(1),
memory_read                  => user_memory_read(1),
HW_Mem_HWAccel               => user_HW_Mem_HWAccel(1),
HW_Offset                    => user_HW_Offset(1),
HW_Count                     => user_HW_Count(1),
data_out                     => out_block_buffer(1),
output_valid                 => output_valid(1),
HW_Mem_HWAccel1              => user_HW_Mem_HWAccel1(1),
HW_Offset1                   => user_HW_Offset1(1),
HW_Count1                    => user_HW_Count1(1),
output_ack                   => output_ack(1),
Host_Mem_HWAccel             => user_Host_Mem_HWAccel(1),
Host_Offset                  => user_Host_Offset(1),
Host_Count                   => user_Host_Count(1),
Host_Mem_HWAccel1            => user_Host_Mem_HWAccel1(1),
Host_Offset1                 => user_Host_Offset1(1),
Host_Count1                  => user_Host_Count1(1),
Mem_Read_Access_Req          => user_Mem_Read_Access_Req(1),
Mem_Write_Access_Req         => user_Mem_Write_Access_Req(1),
Mem_Read_Access_Release_Req  => user_Mem_Read_Access_Release_Req(1),
Mem_Write_Access_Release_Req => user_Mem_Write_Access_Release_Req(1),
Mem_Read_Access_Ack          => user_Mem_Read_Access_Ack(1),
Mem_Write_Access_Ack         => user_Mem_Write_Access_Ack(1),
Mem_Read_Access_Release_Ack  => user_Mem_Read_Access_Release_Ack(1),
Mem_Write_Access_Release_Ack => user_Mem_Write_Access_Release_Ack(1),
done                         => user_done(1)
);
.
.
.
--other instantiated user IP blocks
```

Figure 49 — Example of the User IP Block Interface Signal Instantiation.

Interface Signal Descriptions:

clk – the platform system clock input to the user IP block

reset – reset signal for user IP block

start – when the platform triggers user IP block to work, this signal will be asserted

input_valid – when data are ready to be input to the user IP block, this signal is valid

data_in – input data to the user IP block

memory_read – when user IP block needs to read some data from memory space, this signal is asserted for platform to initialize the memory access controllers

HW_Mem_HWAccel – To read data from memory space, user IP block provides the platform this hardware accelerator number for the platform maps the virtual socket address to physical address

HW_Offset – To read data from memory space, user IP block provides the platform this offset for the platform determines the starting address to be accessed

HW_Count – To read data from memory space, user IP block provides the platform this count for the platform determines the data length to be accessed

data_out – output data to the memory space

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

output_valid – when data are ready to be output to the memory space, this signal is valid

HW_Mem_HWAccel1 – To write data to memory space, user IP block provides the platform this hardware accelerator number for the platform maps the virtual socket address to physical address

HW_Offset1 – To write data to memory space, user IP block provides the platform this offset for the platform determines the starting address to be accessed

HW_Count1 – To write data to memory space, user IP block provides the platform this count for the platform determines the data length to be accessed

output_ack – when platform successfully receives the data from user IP block, this signal is asserted to acknowledge it

Host_Mem_HWAccel – Besides the user IP block itself can provide hardware accelerator number to the platform for memory reading operations, the host system can optionally provide the user IP blocks with this initial hardware accelerator number. Therefore, if the user IP block doesn't want to generate its own hardware accelerator number to be provided to the platform, it can just accept this parameter signal as its desiring hardware accelerator number for memory reading operation.

Host_Offset – Provided by host system as an optional parameter signal to generate the necessary memory starting offset address for memory reading operation.

Host_Count – Provided by host system as an optional parameter signal to generate the necessary data counter for memory reading operation.

Host_Mem_HWAccel1 – Similar to Host_Mem_HWAccel, if the user IP block doesn't want to generate its own hardware accelerator number to be provided to the platform, it can just accept this parameter signal as its desiring hardware accelerator number for memory writing operation.

Host_Offset1 – Similar to Host_Offset, this signal is provided by host system as an optional parameter to generate the necessary memory starting offset address for memory writing operation.

Host_Count1 – Similar to Host_Count, this signal is provided by host system as an optional parameter to generate the necessary data counter for memory writing operation.

Mem_Read_Access_Req – User IP block issue this signal to request the memory reading.

Mem_Read_Access_Ack – Platform generates this signal to acknowledge the user request.

Mem_Read_Access_Release_Req – When user IP block finishes the memory reading operations, it asserts this signal to ask for releasing the memory reading.

Mem_Read_Access_Release_Ack – Platform generates this signal to acknowledge the user releasing request, and therefore the memory is released for the reading operations.

Mem_Write_Access_Req – User IP block issue this signal to request the memory writing.

Mem_Write_Access_Ack – Platform generates this signal to acknowledge the user request.

Mem_Write_Access_Release_Req – When user IP block finishes the memory writing operations, it asserts this signal to ask for releasing the memory writing.

Mem_Write_Access_Release_Ack – Platform generates this signal to acknowledge the user releasing request, and therefore the memory is released for the writing operations.

done – the user interrupt signal to the platform

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Data flow between the platform and user IP blocks:

(1) Host computer system writes data to the memory space.

(2) Host computer system issues a command to trigger the relevant user IP block starting to work.

(3) Once the user IP block starts to work, it will issue “Mem_Read_Access_Req” to ask for reading the memory.

(4) Platform accepts the user reading request and if memory is available, it generates an acknowledgement signal “Mem_Read_Access_Ack” to the user IP block.

(5) Once the user IP block receives the acknowledgement signal, it can ask for reading some data directly from the memory space. This request will be performed by asserting “memory_read” together with setting up some other parameter signals, i.e “HW_Mem_HWAccel”, “HW_Offset”, “HW_Count”.

(6) The platform accepts those signals and reads data from the memory space according to the parameters. When the platform finishes each reading, it asserts “input_valid” and the data are ready in the “data_in” to be input to the user IP block.

(7) The user IP block receives the data from the interface.

(8) User IP block will assert “Mem_Read_Access_Release_Req” to ask for releasing the reading operations when finished.

(9) Platform generates an acknowledgement signal “Mem_Read_Access_Release_Ack” to release the reading operations.

(10) Then user IP block will perform the video algorithms or computations to implement the video application functionality.

(11) After video processing finished, user IP block asserts “Mem_Write_Access_Req” to ask for writing operations.

(12) Platform accepts the user writing request and if memory is available, it generates an acknowledgement signal “Mem_Write_Access_Ack” to the user IP block.

(13) Then user IP block asserts “output_valid” together with some other parameters, i.e “HW_Mem_HWAccel1”, “HW_Offset1”, “HW_Count1” to request writing data back to the memory space.

(14) The platform accepts “output_valid”, receives the data from “data_out” and writes data back to the memory space according to the parameter signals. When each writing is finished, the platform asserts “output_ack” to acknowledge the writing process.

(15) User IP block will assert “Mem_Write_Access_Release_Req” to ask for releasing the writing operations when all the processing is finished.

(16) Platform generates an acknowledgement signal “Mem_Write_Access_Release_Ack” to release the writing operations.

(17) A user interrupt signal is asserted to inform the platform that the user IP block finishes all the internal tasks.

(18) A final interrupt signal is asserted (if the interrupt is enabled) by the platform to inform the host computer system that the current task is done.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.1.10 Simulation Tools and Synthesis File

The architecture of the platform system is prototyped with VHDL. It is simulated using ModelSim 5.4e®, synthesized by Synplify 7.6®, and targeting a FPGA (XC2V3000fg676-4) from the Virtex-II family of Xilinx® based on the WildCard-IITM FPGA device.

The following is a synthesis file for implementation of the whole virtual socket platform system.

```
#-------------------------------------------------------------------------
#-
#-  1. Change the PROJECT_BASE and WILDCARD_BASE variables to your
#-      project directory and WILDCARD-II installation directory,
#-      respectively:
#-
#-------------------------------------------------------------------------
set ANNAPOLIS_BASE "c:/annapolis"
set PROJECT_BASE  "$ANNAPOLIS_BASE/wildcard_ii/examples/virtual_socket/vhdl/sim"

#
# Do not modify these lines!
#
set WILDCARD_BASE   "$ANNAPOLIS_BASE/wildcard_ii"
set PE_BASE         "$WILDCARD_BASE/vhdl/pe"
set SIMULATION_BASE "$WILDCARD_BASE/vhdl/simulation"
set TOOLS_BASE      "$WILDCARD_BASE/vhdl/tools"
set SYNTHESIS_BASE  "$WILDCARD_BASE/vhdl/synthesis"

source "$SYNTHESIS_BASE/tcl/wildcard_pe.tcl"

#-------------------------------------------------------------------------
#-
#-  2. Add your project's PE architecture VHDL file here, as
#-      well as any VHDL or constraint files on which your PE
#-      design depends:
#-      NOTE: They must be in the order specified below.
#-
#-------------------------------------------------------------------------

add_file -vhdl -lib pe_lib "$PROJECT_BASE/data_type_definition_package.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/bram_dma_source_entarch.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/bram_memory_destination_entarch.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/bram_memory_source_entarch.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/bram_dma_destination_entarch.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/dma_source_entarch.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/memory_destination_entarch.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/memory_source_entarch.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/dma_destination_entarch.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/dram_dma_source_entarch.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/dram_memory_destination_entarch.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/dram_memory_source_entarch.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/dram_dma_destination_entarch.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/User_IP_Block_0.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/User_IP_Block_1.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/User_IP_Block_2.vhd"
add_file -vhdl -lib pe_lib "$PE_BASE/pe_ent.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/virtual_socket_testing.vhd"

#-------------------------------------------------------------------------
#-
#-  3. Change the result file:
#-
#-------------------------------------------------------------------------
project -result_file "virtual_socket_testing.edf"
#-------------------------------------------------------------------------
#-
#-  4. Add or change the following options according to your
#-      project's needs:
#-------------------------------------------------------------------------

# State machine encoding
set_option -default_enum_encoding sequential

# State machine compiler
set_option -symbolic_fsm_compiler false
```

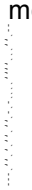
--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
# Resource sharing
set_option -resource_sharing true

# Clock frequency
set_option -frequency 132.000

# Fan-out limit
set_option -fanout_limit 1000

# Hard maximum fanout limit
set_option -maxfan_hard false
```

8.1.11 Conformance Tests

We can set up a conformance testing program to verify the features of the virtual socket co-design platform. The program is coded in C language. There is a debug menu for the program, and users may interactively use debug menu and choose appropriate options to perform the testing.



© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

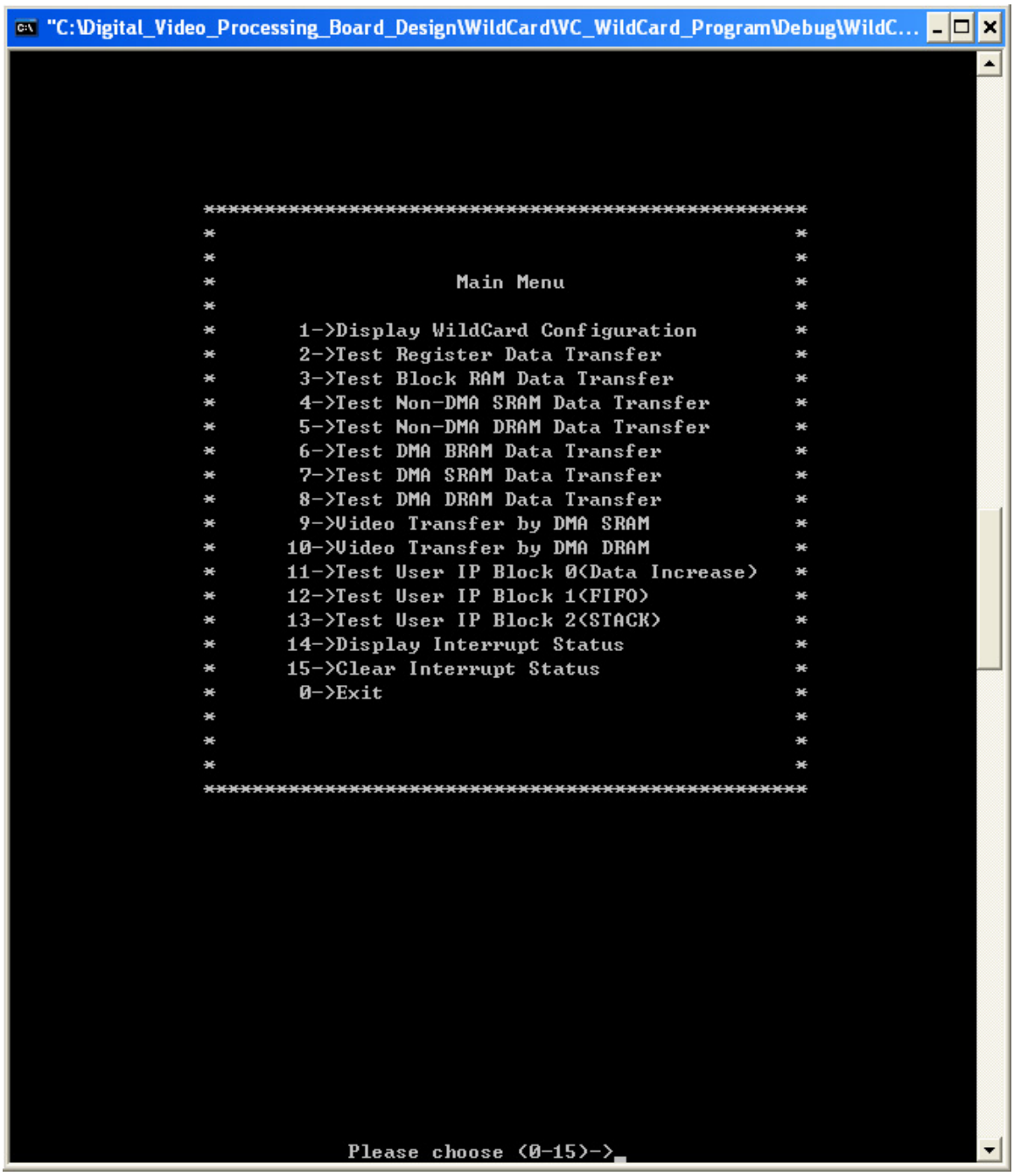


Figure 50 — Debug Menu for Conformance Tests.

Currently, there are 15 feature options included in the debug menu. Those options can test all the 4 types of physical memory space as well as the hardware interface module controller which connects the user IP blocks to the virtual socket platform.

Option 1 : Get the configuration information of the WildCard II Platform such as platform hardware version, FPGA type, FPGA package type, FPGA speed grade, RAM size, RAM access speed, API version, Driver version and Firmware version etc.

Option 2 : Test general registers.

Option 3 : Test Block RAM based on non-DMA operation..

Option 4 : Test SRAM based on non-DMA operation.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Option 5 : Test DRAM based on non-DMA operation.

Option 6 : Test Block RAM based on DMA operation.

Option 7 : Test SRAM based on DMA operation.

Option 8 : Test DRAM based on DMA operation.

Option 9 : Test SRAM video data transfer based on DMA operation. The video frames are in CIF YUV 4:2:0 format.

Option 10: Test DRAM video data transfer based on DMA operation. The video frames are in CIF YUV 4:2:0 format.

Option 11: Test Data Increase Module as user IP block 0.

Option 12: Test FIFO Module as user IP block 1.

Option 13: Test STACK Module as user IP block 2.

Option 14: Display internal user interrupt status register.

Option 15: Cleanup internal user interrupt status register.

The following is an example testing result using option 9 or 10 to write / read data to and from platform through DMA operations.

Currently, there are 15 feature options included in the debug menu. Those options can test all the 4 types of physical memory space as well as the hardware interface module controller which connects the user IP blocks to the virtual socket platform.

Option 1 : Get the configuration information of the WildCard II Platform such as platform hardware version, FPGA type, FPGA package type, FPGA speed grade, RAM size, RAM access speed, API version, Driver version and Firmware version etc.

Option 2 : Test general registers.

Option 3 : Test Block RAM based on non-DMA operation..

Option 4 : Test SRAM based on non-DMA operation.

Option 5 : Test DRAM based on non-DMA operation.

Option 6 : Test Block RAM based on DMA operation.

Option 7 : Test SRAM based on DMA operation.

Option 8 : Test DRAM based on DMA operation.

Option 9 : Test SRAM video data transfer based on DMA operation. The video frames are in CIF YUV 4:2:0 format.

Option 10: Test DRAM video data transfer based on DMA operation. The video frames are in CIF YUV 4:2:0 format.

Option 11: Test Data Increase Module as user IP block 0.

Option 12: Test FIFO Module as user IP block 1.

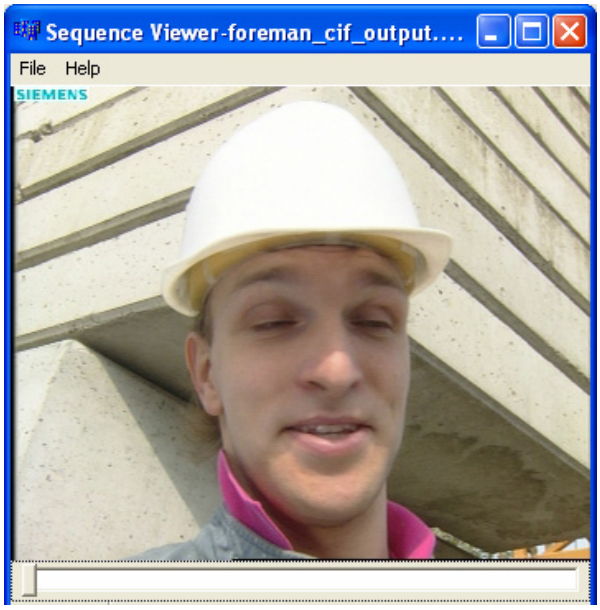
© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Option 13: Test STACK Module as user IP block 2.

Option 14: Display internal user interrupt status register.

Option 15: Cleanup internal user interrupt status register.

The following is an example testing result using option 9 or 10 to write / read data to and from platform through DMA operations.



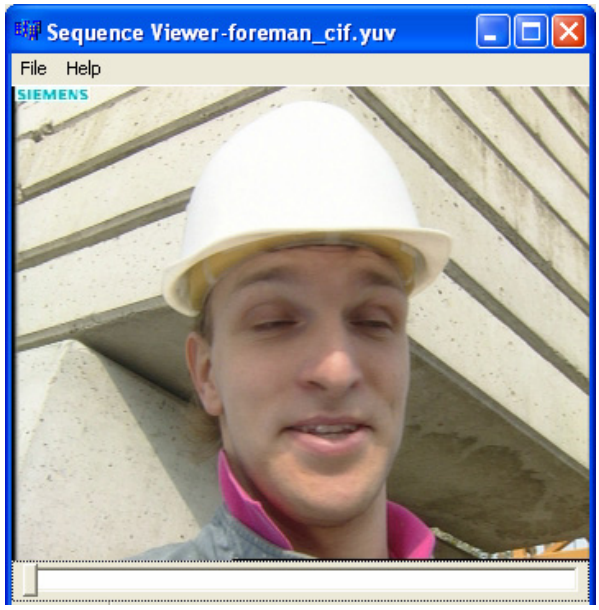


Figure 51 — Write Data to Platform by DMA Transfer. **Figure 52 — Read Data from Platform by DMA Transfer.**

8.1.11.1 The DRAM Access Conformance Test

The conformance testing program will implement all the modes of DRAM data transfer and IP block DRAM data access operations. Figure 53 to Figure 56 show the testing results of non-DMA DRAM transfer, DMA DRAM transfer, DMA video frame data transfer through DRAM, and IP Block DRAM data access, respectively.

© ISO/IEC 2009 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

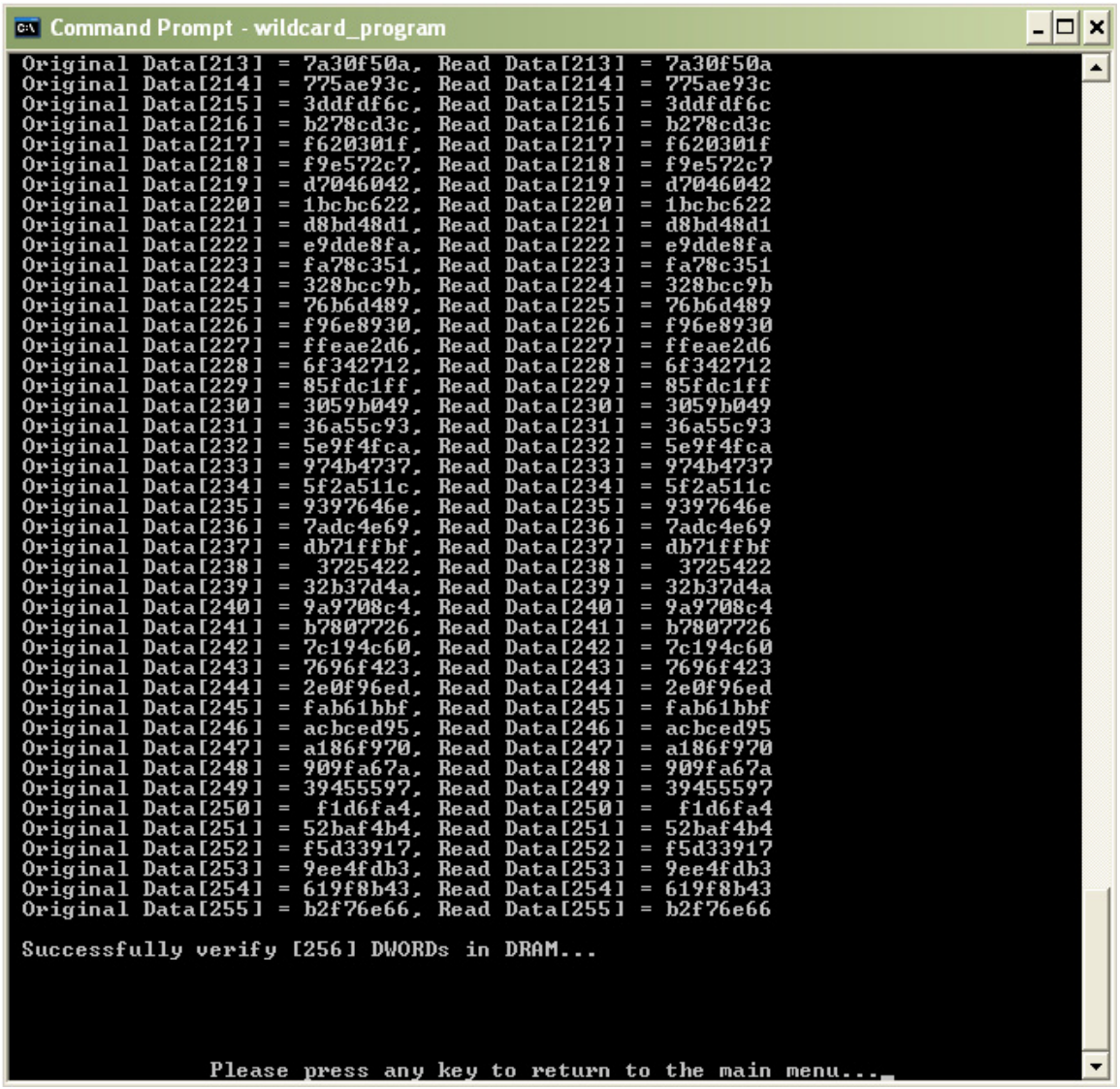


Figure 53 — Non-DMA DRAM Data Transfer.

--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2009 – All rights reserved
Not for Resale

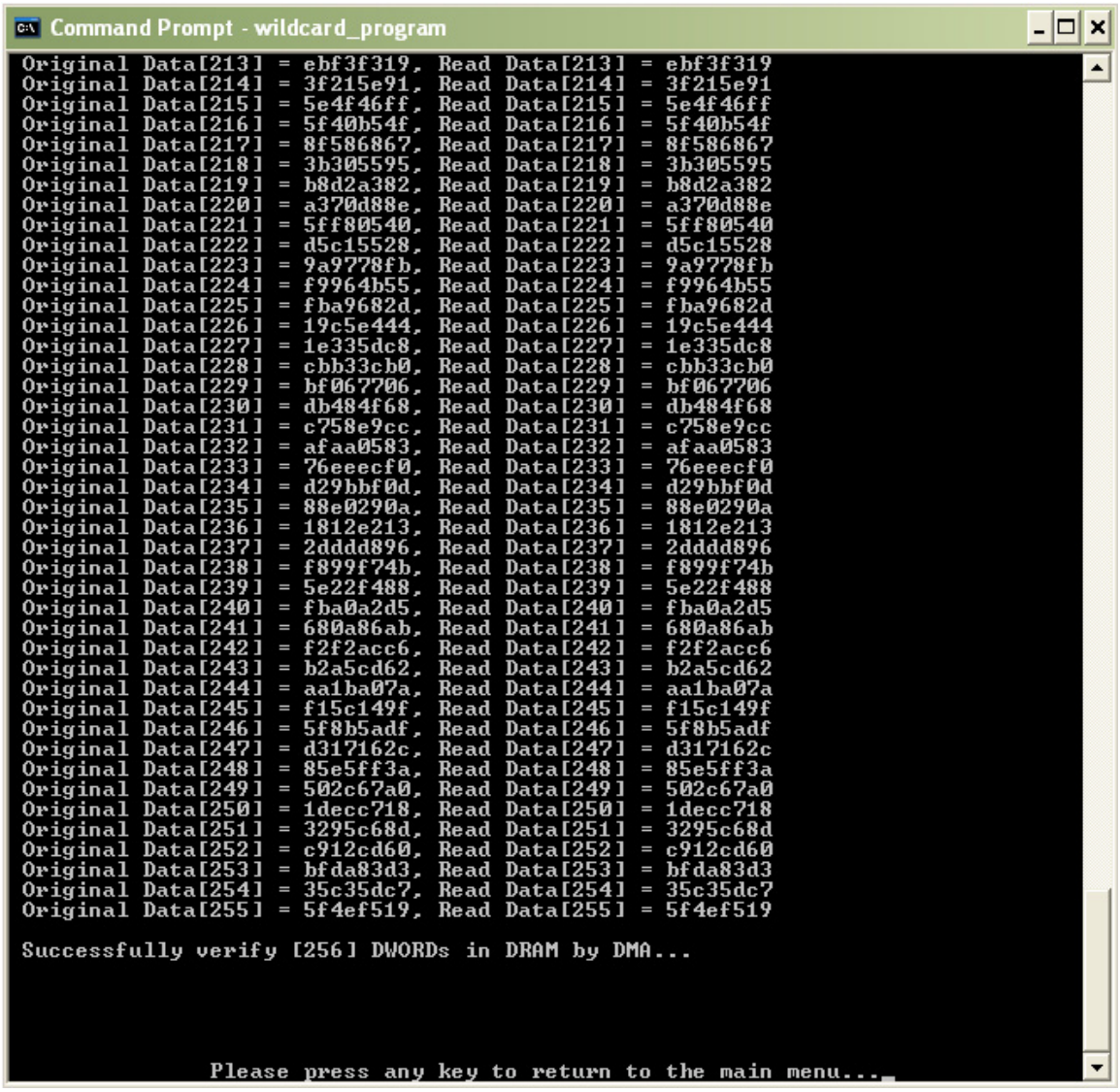


Figure 54 — DMA DRAM Data Transfer.

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

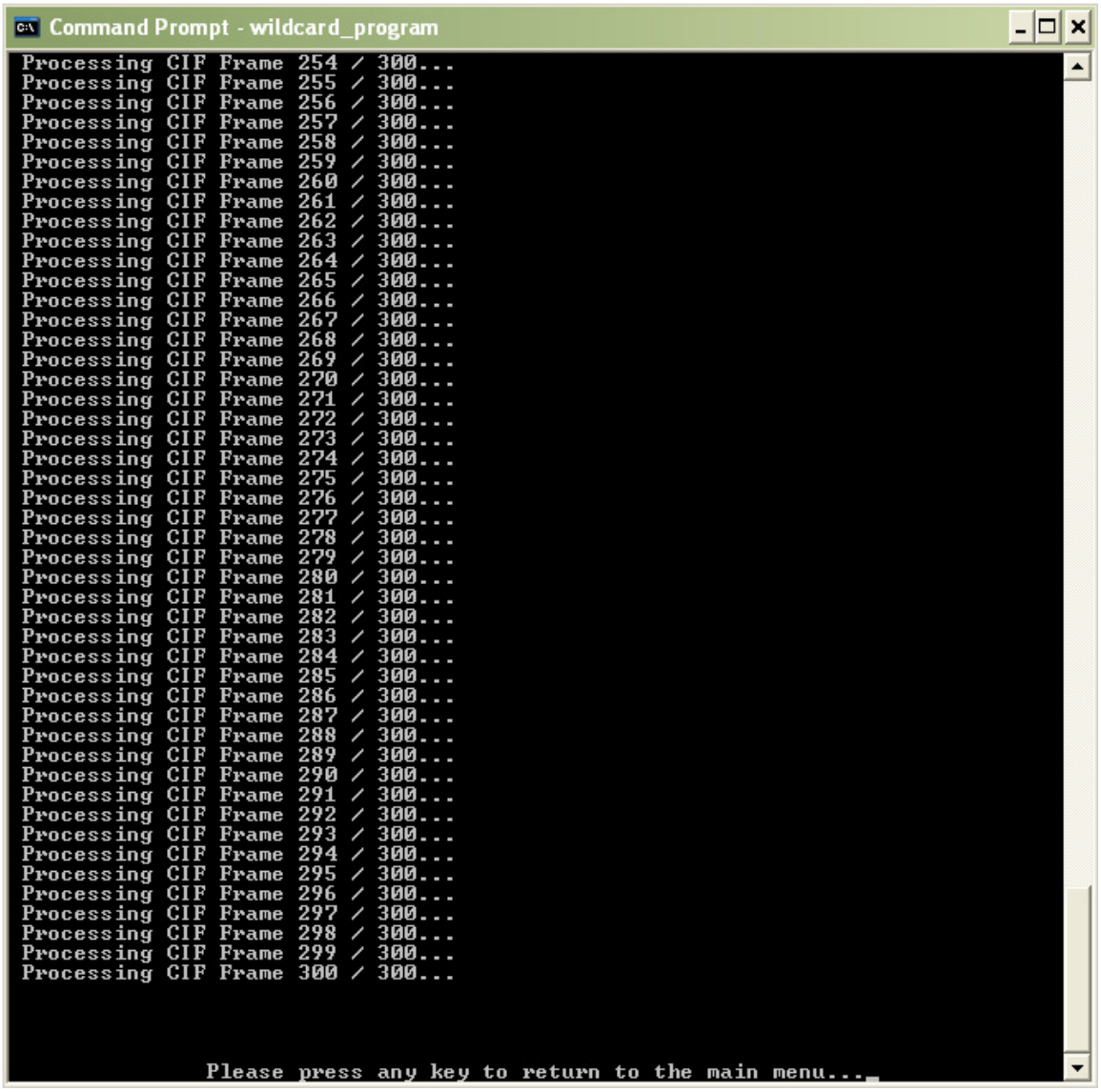


Figure 55 — DMA Video Frame Data Transfer through DRAM.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2009 – All rights reserved

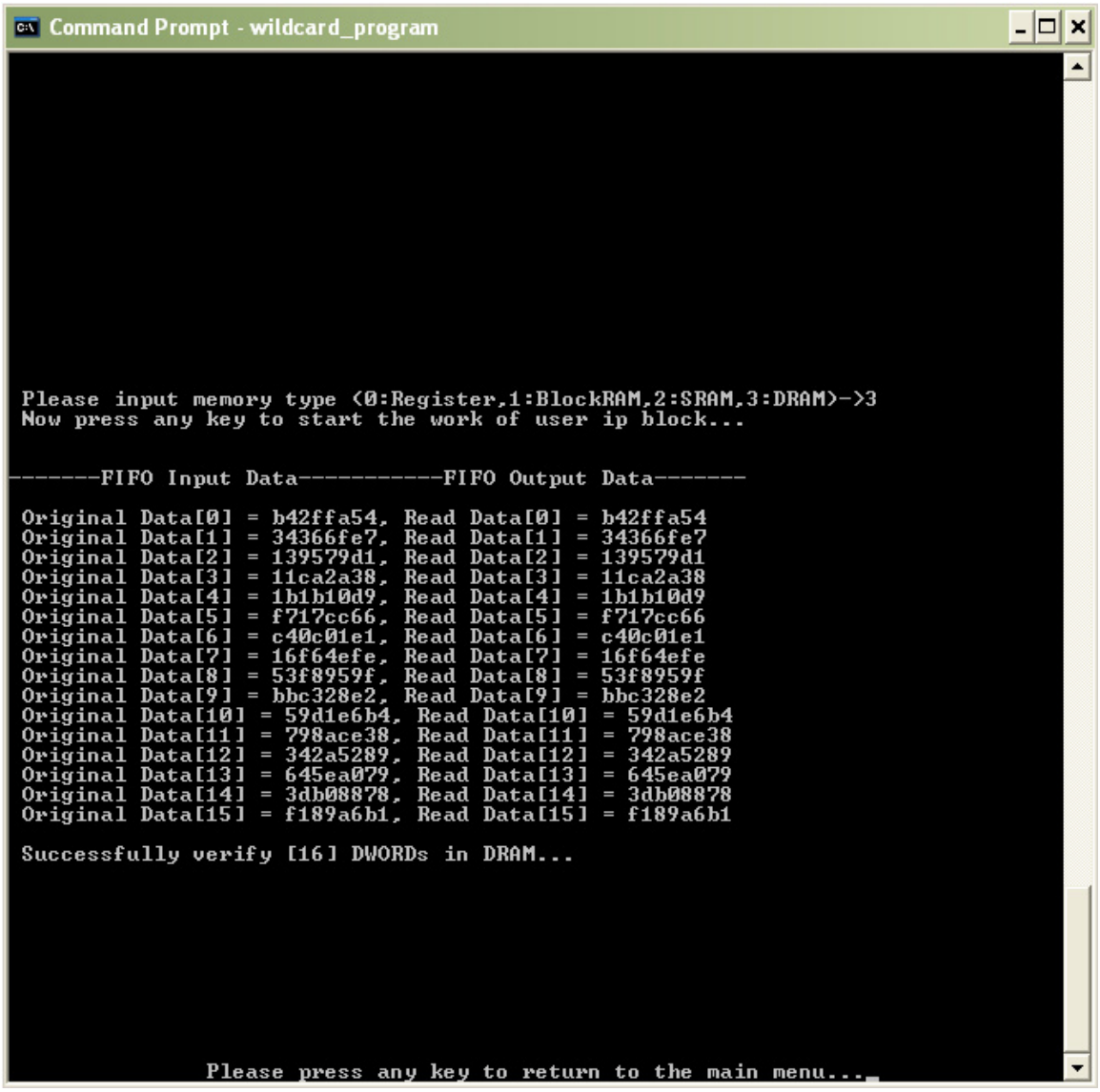


Figure 56 — IP Block DRAM Data Access.

8.1.11.2 The Extended IP Block Interrupt's Conformance Test

A conformance test is used to verify the validity of this proposed extended interrupt mechanism. In the following example test, we assume the user IP block 1 and User IP block 2 generate the interrupts, and the host will check the interrupt status and therefore display the interrupt result on the screen. Furthermore, the host will clean up the interrupt status bits asserted by user IP block 1 and User IP block 2.

© ISO/IEC 2009 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

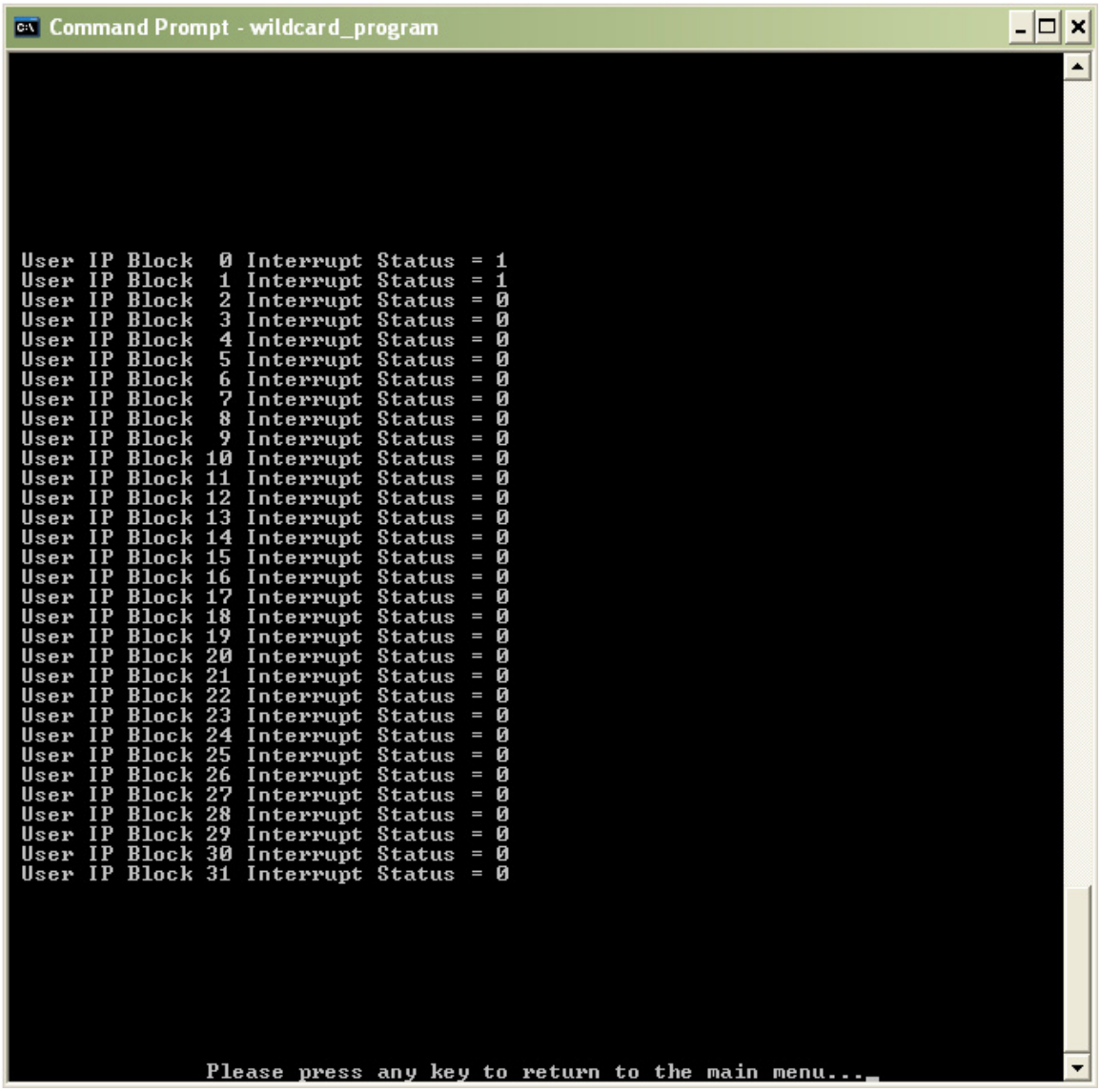


Figure 57 — Interrupts Generated by User IP Block 1 and 2.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2009 – All rights reserved
Not for Resale

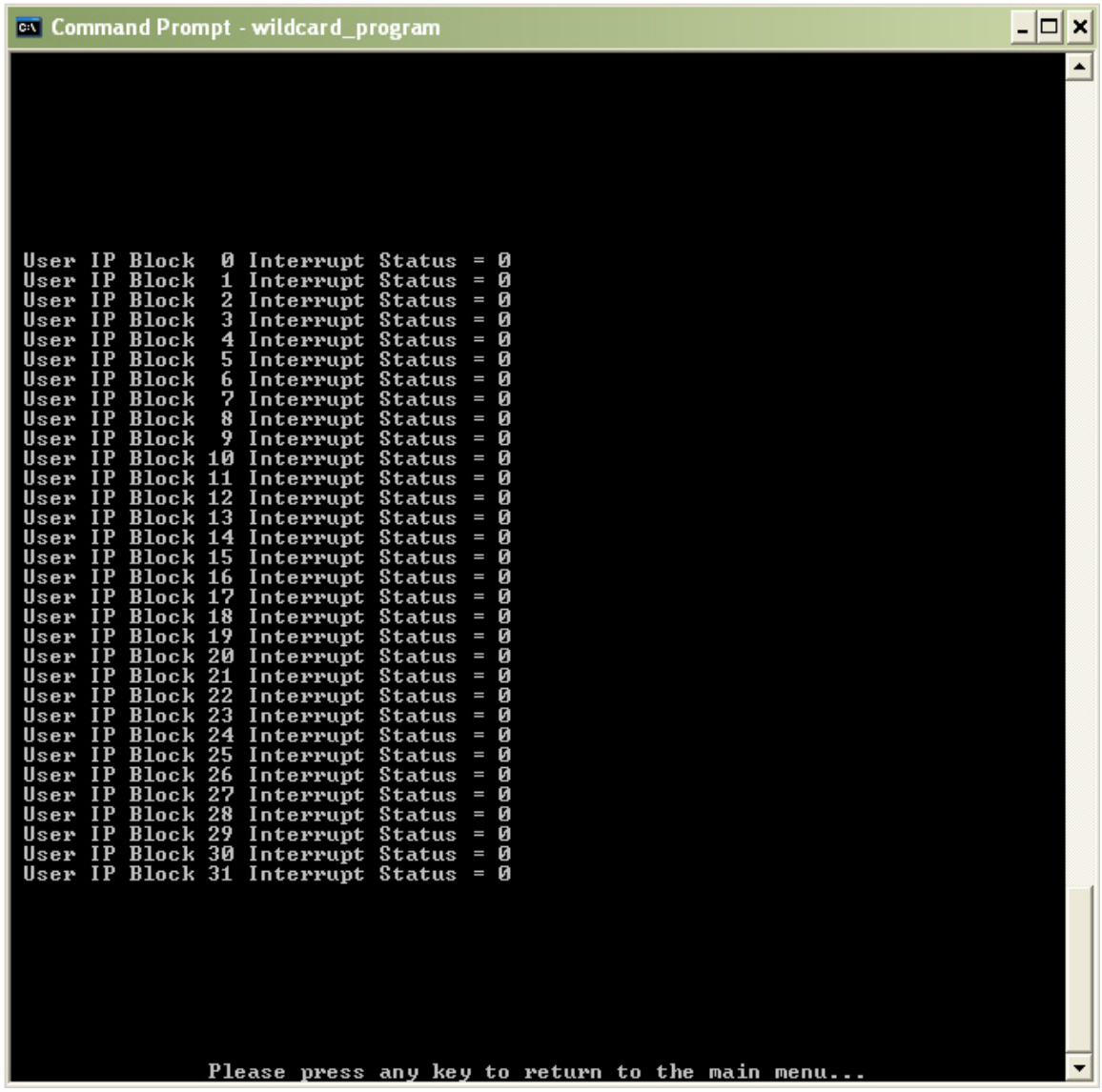


Figure 58 — Interrupts Cleared by Host Interface.

8.1.11.3 The IP Blocks Concurrent Control's Conformance Test

A conformance testing program is set up to implement the verification of platform system concurrent operations. For this example, IP Block 0 accesses Register space to perform the data increase functionality, and IP Block 1 employs BRAM space to implement STACK structure. Figure 59 shows the result transcript of this test.

```
Please input memory type for IP0(0:Register,1:BlockRAM,2:SRAM,3:DRAM)-> 0
Please input memory type for IP1(0:Register,1:BlockRAM,2:SRAM,3:DRAM)-> 1
Now press any key to start the work of user ip blocks...
 Register Data for IP0...
 Original Data[0] = f208c029, Read Data[0] = f208c02a
 Original Data[1] = d2b86784, Read Data[1] = d2b86785
 Original Data[2] = 3cabacd6, Read Data[2] = 3cabacd7
 Original Data[3] = 15925f90, Read Data[3] = 15925f91
 Original Data[4] = 906edaf1, Read Data[4] = 906edaf2
 Original Data[5] = 62ecc1eb, Read Data[5] = 62ecc1ec
 Original Data[6] = 754f12db, Read Data[6] = 754f12dc
 Original Data[7] = 93cfb90c, Read Data[7] = 93cfb90d
 Original Data[8] = dc178124, Read Data[8] = dc178125
 Original Data[9] = 7341c91c, Read Data[9] = 7341c91d
 Original Data[10] = 35379547, Read Data[10] = 35379548
 Original Data[11] = 81d36d12, Read Data[11] = 81d36d13
 Original Data[12] = b9aee443, Read Data[12] = b9aee444
```

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
 Original Data[13] = 3c07e6a6, Read Data[13] = 3c07e6a7
 Original Data[14] = 9d9f7a5a, Read Data[14] = 9d9f7a5b
 Original Data[15] = fec95238, Read Data[15] = fec95239
 Successfully verify [16] DWORDs in Registers...

BRAM Data for IP1...
 Original Data[0] = b6b56e5d, Read Data[0] = 25686c3b
 Original Data[1] = 5fe5ebfc, Read Data[1] = 3fadc230
 Original Data[2] = 3c8ece45, Read Data[2] = 4d9adcd0
 Original Data[3] = bae2660d, Read Data[3] = 6b904944
 Original Data[4] = e2f6f01c, Read Data[4] = 3785314f
 Original Data[5] = a0480732, Read Data[5] = d3775f49
 Original Data[6] =  8bba350, Read Data[6] = dea7bbf6
 Original Data[7] = dacdd878, Read Data[7] = 26927e12
Original Data[8] = 26927e12, Read Data[8] = dacdd878
 Original Data[9] = dea7bbf6, Read Data[9] =  8bba350
 Original Data[10] = d3775f49, Read Data[10] = a0480732
 Original Data[11] = 3785314f, Read Data[11] = e2f6f01c
 Original Data[12] = 6b904944, Read Data[12] = bae2660d
 Original Data[13] = 4d9adcd0, Read Data[13] = 3c8ece45
 Original Data[14] = 3fadc230, Read Data[14] = 5fe5ebfc
 Original Data[15] = 25686c3b, Read Data[15] = b6b56e5d
Successfully verify [16] DWORDs in Block RAM...
```

Figure 59 — Concurrent Operations of Multiple IP Blocks.

8.2 Reference for Virtual Socket API Function Calls

8.2.1 Introduction

Application Programming Interface (API) is a set of functions coded in C language allowing data communication between host computer system and Wildcard II platform. Actually, the virtual socket API function calls here are not simply the Wildcard II API functions. They are constructed on the basic Wildcard II API functions to perform the data communication between the host system and virtual socket platform.

The following virtual socket API function calls are used.

WC_Open Open a WildCard II FPGA device.

WC_Close Close a WildCard II FPGA device.

VSInit Initialize the virtual socket platform.

VSSystemReset Reset the virtual socket platform.

VSUserResetEnable Enable a reset for user IP-blocks.

VSUserResetDisable Disable a reset for user IP-blocks.

VSIntEnable Enable / Disable the platform interrupt generation

VSIntClear Clean up the platform interrupt.

VSIntCheck Check and wait for the platform interrupt generation.

VSIntStatusWrite Set up the interrupt status and mask registers.

VSIntStatusRead Read the interrupt status and mask registers.

VSWrite Write data to memory space by non-DMA operations.

VSRead Read data from memory space by non-DMA operations.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

VSDMAInit	Initialize DMA channel and allocate DMA data buffer.
VSDMAClear	Release DMA channel and data buffer allocation.
VSDMAWrite	Write data to memory space by DMA operations.
VSDMARead	Read data from memory space by DMA operations.
HW_Module_Controller	Trigger and control the user IP-blocks to work.

8.2.2 Virtual Socket Platform Address Mapping

The virtual socket platform memory space is mapped to 4 types of physical storage, i.e Registers, Block RAM, ZBT-SRAM and DDR-DRAM. We define a maximum of 32 user hardware accelerators (IP-blocks) which can be integrated into the platform. The data for each user IP-block can be stored in any type of the physical memory space mentioned above. Currently, we define the size of register space is 16 double words for each user IP-block, BRAM 512 double words for each user IP-block, SRAM 2MB for all the user IP-blocks, and DRAM 32MB for all the user IP-blocks. The size of each memory unit is double word based.

8.2.2.1 Address Mapping for Register Space

Data stored in register space have a maximum size of 16 double words for each user IP-block. Usually, the data stored in register space are some parameters for processing the video data algorithms or hardware modules. Address mapping for register space is listed in Table 6.

Table 6 — Address Mapping for Register Space.

HWAccel Number	Virtual Address	Physical Address	HWAccel Number	Virtual Address	Physical Address
0	0x0000-0x000f	0x0000-0x000f	16	0x2000-0x200f	0x2000-0x200f
1	0x0200-0x020f	0x0200-0x020f	17	0x2200-0x220f	0x2200-0x220f
2	0x0400-0x040f	0x0400-0x040f	18	0x2400-0x240f	0x2400-0x240f
3	0x0600-0x060f	0x0600-0x060f	19	0x2600-0x260f	0x2600-0x260f
4	0x0800-0x080f	0x0800-0x080f	20	0x2800-0x280f	0x2800-0x280f
5	0x0a00-0x0a0f	0x0a00-0x0a0f	21	0x2a00-0x2a0f	0x2a00-0x2a0f
6	0x0c00-0x0c0f	0x0c00-0x0c0f	22	0x2c00-0x2c0f	0x2c00-0x2c0f
7	0x0e00-0x0e0f	0x0e00-0x0e0f	23	0x2e00-0x2e0f	0x2e00-0x2e0f
8	0x1000-0x100f	0x1000-0x100f	24	0x3000-0x300f	0x3000-0x300f
9	0x1200-0x120f	0x1200-0x120f	25	0x3200-0x320f	0x3200-0x320f
10	0x1400-0x140f	0x1400-0x140f	26	0x3400-0x340f	0x3400-0x340f
11	0x1600-0x160f	0x1600-0x160f	27	0x3600-0x360f	0x3600-0x360f
12	0x1800-0x180f	0x1800-0x180f	28	0x3800-0x380f	0x3800-0x380f
13	0x1a00-0x1a0f	0x1a00-0x1a0f	29	0x3a00-0x3a0f	0x3a00-0x3a0f
14	0x1c00-0x1c0f	0x1c00-0x1c0f	30	0x3c00-0x3c0f	0x3c00-0x3c0f
15	0x1e00-0x1e0f	0x1e00-0x1e0f	31	0x3e00-0x3e0f	0x3e00-0x3e0f

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.2.2.2 Address Mapping for BRAM Space

Data stored in Block RAM space have a maximum size of 512 double words for each user IP-block. Address mapping for BRAM space is listed in Table 7.

Table 7 — Address Mapping for BRAM Space.

HWAccel Number	Virtual Address	Physical Address	HWAccel Number	Virtual Address	Physical Address
0	0x0000-0x01ff	0x8000-0x81ff	16	0x2000-0x21ff	0xa000-0xa1ff
1	0x0200-0x03ff	0x8200-0x83ff	17	0x2200-0x23ff	0xa200-0xa3ff
2	0x0400-0x05ff	0x8400-0x85ff	18	0x2400-0x25ff	0xa400-0xa5ff
3	0x0600-0x07ff	0x8600-0x87ff	19	0x2600-0x27ff	0xa600-0xa7ff
4	0x0800-0x09ff	0x8800-0x89ff	20	0x2800-0x29ff	0xa800-0xa9ff
5	0x0a00-0x0bff	0x8a00-0x8bff	21	0x2a00-0x2bff	0xaa00-0xabff
6	0x0c00-0x0dff	0x8c00-0x8dff	22	0x2c00-0x2dff	0xac00-0xadff
7	0x0e00-0x0fff	0x8e00-0x8fff	23	0x2e00-0x2fff	0xae00-0xafff
8	0x1000-0x11ff	0x9000-0x91ff	24	0x3000-0x31ff	0xb000-0xb1ff
9	0x1200-0x13ff	0x9200-0x93ff	25	0x3200-0x33ff	0xb200-0xb3ff
10	0x1400-0x15ff	0x9400-0x95ff	26	0x3400-0x35ff	0xb400-0xb5ff
11	0x1600-0x17ff	0x9600-0x97ff	27	0x3600-0x37ff	0xb600-0xb7ff
12	0x1800-0x19ff	0x9800-0x99ff	28	0x3800-0x39ff	0xb800-0xb9ff
13	0x1a00-0x1bff	0x9a00-0x9bff	29	0x3a00-0x3bff	0xba00-0xbbff
14	0x1c00-0x1dff	0x9c00-0x9dff	30	0x3c00-0x3dff	0xbc00-0xbdff
15	0x1e00-0x1fff	0x9e00-0x9fff	31	0x3e00-0x3fff	0xbe00-0xbfff

8.2.2.3 Address Mapping for SRAM Space

ZBT-SRAM is a large size memory space for video data storage. Actually, each user IP-block can utilize the whole address space of the SRAM for storing the video data. The SRAM size is up to 2M Bytes, i.e 524,288 double words. The address mapping for SRAM space is listed in Table 8.

Table 8 — Address Mapping for SRAM Space.

HWAccel Number	Virtual Address	Physical Address
0 – 31	0x00000000-0x0007ffff	0x00000000-0x0007ffff

8.2.2.4 Address Mapping for DRAM Space

Similar to the ZBT-SRAM space, DDR-DRAM is also a large size memory space for video data storage. Each user IP-block can utilize the whole address space of the DRAM for storing the video data. Currently, the DRAM size can be up to 32M Bytes, i.e 8,388,608 double words. The address mapping for DRAM space is listed in Table 9.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 9 — Address Mapping for DRAM Space.

HWAccel Number	Virtual Address	Physical Address
0 – 31	0x00000000-0x007fffff	0x00000000-0x007fffff

8.2.3 Description of virtual socket API function calls

WC_Open

Please refer to user manual of WildCard II device for details about its function [5].

WC_Close

Please refer to user manual of WildCard II device for details about its function [5].

VSInit

Syntax: *WC_RetCode VSInit(WC_TestInfo *TestInfo);*

Description: Initialize the virtual socket platform.

Arguments: *TestInfo* A point of the index number of a WildCard II FPGA device.

Returns: An enumeration structure *WC_RetCode* [5].

Comments: *VSInit* is used to initialize a virtual socket platform. The process of initialization is to reset platform, get platform device information, load image file to FPGA and set default system clock frequency. The initialization must be performed after opening a WildCard II FPGA device by *WC_Open* function call.

VSSystemReset

Syntax : *WC_RetCode VSSystemReset(WC_TestInfo *TestInfo);*

Description: Reset the virtual socket platform.

Arguments: *TestInfo* A point of the index number of a WildCard II FPGA device.

Returns: An enumeration structure *WC_RetCode* [5].

Comments: *VSSystemReset* is to reset a virtual socket platform. Unlike *VSInit*, this function only issues the system reset signal to the platform. Usually, if users want to reset the virtual socket platform during its work, this function can be employed. In some other cases, however, if users want to load image file other than reset the platform, *VSInit* can be employed.

VSUserResetEnable

Syntax : *WC_RetCode VSUserResetEnable(WC_TestInfo *TestInfo);*

Description: Enable a reset for user IP-blocks.

Arguments: *TestInfo* A point of the index number of a WildCard II FPGA device.

Returns: An enumeration structure *WC_RetCode* [5].

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Comments: *VSUserResetEnable* is used to enable a reset for user IP-blocks. Normally, when a system reset signal is issued by *VSSystemReset*, all the hardware modules, including memory space controller, hardware module controller and user IP-blocks, will be reset. But sometimes it is unnecessary for users to reset their own IP-blocks when a system reset signal is issued. In such case, *VSUserResetEnable* and *VSUserResetDisable* will provide a way for users to enable or shield their own IP-blocks from system reset signal.

See Also: *VSUserResetDisable*

VSUserResetDisable

Syntax : *WC_RetCode VSUserResetDisable(WC_TestInfo *TestInfo);*

Description: Disable a reset for user IP-blocks.

Arguments: *TestInfo* A point of the index number of a WildCard II FPGA device.

Returns: An enumeration structure *WC_RetCode* [5].

Comments: *VSUserResetDisable* is used to shield the user IP-blocks from a system reset signal.

See Also: *VSUserResetEnable*

VSIntEnable

Syntax : *WC_RetCode VSIntEnable(WC_TestInfo *TestInfo,boolean IntEnable);*

Description: Enable or disable the platform interrupt generation.

Arguments: *TestInfo* A point of the index number of a WildCard II FPGA device.

IntEnable A parameter to enable or disable the interrupt generation.

Returns: An enumeration structure *WC_RetCode* [5].

Comments: When users need platform to generate an interrupt signal to computer host system after the hardware modules finish their work, *IntEnable* should be set to “1” to enable the interrupt generation. Otherwise, *IntEnable* is set to “0” to disable the interrupt generation.

VSIntClear

Syntax : *WC_RetCode VSIntClear(WC_TestInfo *TestInfo);*

Description: Clean up the generated platform interrupt signal.

Arguments: *TestInfo* A point of the index number of a WildCard II FPGA device.

Returns: An enumeration structure *WC_RetCode* [5].

Comments: After the platform generates an interrupt signal to the host system, users can issue *VSIntClear* to release the platform interrupt and clean it up.

VSIntCheck

Syntax : *WC_RetCode VSIntCheck(WC_TestInfo *TestInfo);*

Description: Check and wait for the platform interrupt generation.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Arguments: *TestInfo* A point of the index number of a WildCard II FPGA device.

Returns: An enumeration structure *WC_RetCode* [5].

Comments: Before the platform generates an interrupt signal, users can employ this function call to check and wait for the interrupt signal to come. Usually, there is a default INT_TIMEOUT_MS parameter set up in the C language header file to indicate how long time (millisecond) this function call checks and waits for the coming interrupt.

VSIntStatusWrite

Syntax : *WC_RetCode VSIntStatusWrite(WC_TestInfo *TestInfo,*
*DWORD *IntStatus,*
*DWORD *IntMask);*

Description: Set up the platform interrupt status and mask registers.

Arguments: *TestInfo* A point of the index number of a WildCard II FPGA device.

IntStatus Interrupt Status Register

IntMask Interrupt Mask Register

Returns: An enumeration structure *WC_RetCode* [5].

Comments: *VSIntStatusWrite* is used to set up the platform interrupt status register and interrupt mask register. Usually, the users can write all zeros to interrupt status register for cleaning up the platform recorded interrupt status. When an interrupt is generated from the user IP-block, the corresponding bit of the interrupt status register will be overwritten by that interrupt signal and thus set to “1”. If the corresponding bit of the interrupt mask register is to enable the platform interrupt, a final interrupt will be generated from the platform to the host system. The maximum user interrupt resources can be up to 32.

VSIntStatusRead

Syntax : *WC_RetCode VSIntStatusRead(WC_TestInfo *TestInfo,*
DWORD *IntStatus,
DWORD *IntMask);

Description: Read the interrupt status and mask registers.

Arguments: *TestInfo* A point of the index number of a WildCard II FPGA device.

IntStatus Interrupt Status Register

IntMask Interrupt Mask Register

Returns: An enumeration structure *WC_RetCode* [5].

Comments: *VSIntStatusRead* is used to read the platform interrupt status register and interrupt mask register.

See Also: *VSIntStatusWrite*

VSWrite

Syntax : *WC_RetCode VSWrite(WC_TestInfo *TestInfo,*
unsigned int HWAccel,
unsigned long Offset,

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

unsigned long Count,
*DWORD *Buffer);*

Description: Write data to memory space by non-DMA operations.

Arguments: *TestInfo* A point of the index number of a WildCard II FPGA device.

HWAccel Hardware accelerator number (user IP-block number).

Offset Offset of the memory address space

Count Size of Dwords to write

Buffer A Pointer to an array of Dwords, *Buffer* Length ≥ *Count*

Returns: An enumeration structure *WC_RetCode* [5].

Comments: *VSWrite* is used to write data from host system to the memory space by non-DMA operations. There are 4 types of memory address space inside the platform, i.e Register, BRAM, SRAM and DRAM. *VSWrite* can automatically map the virtual socket address space into the physical address space, and thus access all the 4 types of physical memory storage within one function call.

It is assumed that the host writes the data to the memory space belonging to corresponding IP-blocks, as long as the users provide appropriate *HWAccel*, *Offset* and *Count* to the function call. The same assumption is applied for *VSRead*, *VSDMAWrite* and *VSDMARead* function calls listed below.

The maximum number of *HWAccel* is from 0 to 31, which means there are at most 32 user IP-blocks can be integrated into the virtual socket platform. The highest 2 bits of "*Offset*" [31:30] indicate which specific memory space the hardware accelerators (user IP-blocks) want to write, while "*Offset*" [29:28] indicate which specific memory space the hardware accelerators want to read.

Offset[31:30]	Memory Write Accessed
0 0	Register
0 1	BRAM
1 0	SRAM
1 1	DRAM

Offset[29:28]	Memory Read Accessed
0 0	Register
0 1	BRAM
1 0	SRAM
1 1	DRAM

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Usage:

(1) if the user IP-block 2 needs to write 16 double words to the register space with the offset 0 by non-DMA transfer, then write

VSWrite(TestInfo, 2, 0x00000000 + 0, 16, Buffer)

(2) if the user IP-block 5 needs to write 256 double words to the BRAM space with the offset 10 by non-DMA transfer, then write

VSWrite(TestInfo, 5, 0x50008000 + 10, 256, Buffer)

(3) if the user IP-block needs to write 1024 double words to the SRAM space with the offset 0 by non-DMA transfer, then write

VSWrite(TestInfo, 1, 0xa0000000 + 0, 1024, Buffer)

(4) if the user IP-block needs to write 4096 double words to the DRAM space with the offset 1000 by non-DMA transfer, then write

VSWrite(TestInfo, 10, 0xf0000000 + 1000, 4096, Buffer)

Note: To access the individual memory space correctly according to its maximum size, we must have *Offset* + *Count* ≤ Size of Memory Space, and the highest 2 bits of *Offset* [31:30] are set correctly.

VSRead

Syntax : *WC_RetCode VSRead(WC_TestInfo *TestInfo,*
unsigned int HWAccel,
unsigned long Offset,
unsigned long Count,
*DWORD *Buffer);*

Description: Read data from memory space by non-DMA operations.

Arguments: *TestInfo* A point of the index number of a WildCard II FPGA device.

HWAccel Hardware accelerator number (user IP-block number).

Offset Offset of the memory address space

Count Size of Dwords to read

Buffer A Pointer to an array of Dwords, Buffer Length ≥ Count

Returns: An enumeration structure *WC_RetCode* [5].

Comments: *VSRead* is used to read data from the memory space to host system by non-DMA operations. There are 4 types of memory address space inside the platform, i.e Register, BRAM, SRAM and DRAM. *VSRead* can automatically map the virtual socket address space into the physical address space, and thus access all the 4 types of physical memory storage within one function call.

See Also: *VSWrite*

Usage:

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

(1) if the user IP-block 18 needs to read 10 double words from the register space with the offset 5 by non-DMA transfer, then write

VSRead(TestInfo, 18, 0x00000000 + 5, 10, Buffer)

(2) if the user IP-block 27 needs to read 256 double words from the BRAM space with the offset 100 by non-DMA transfer, then write

VSRead(TestInfo, 27, 0x50008000 + 100, 256, Buffer)

(3) if the user IP-block needs to read 2000 double words from the SRAM space with the offset 500 by non-DMA transfer, then write

VSRead(TestInfo, 20, 0xa0000000 + 500, 2000, Buffer)

(4) if the user IP-block needs to read 5000 double words from the DRAM space with the offset 300 by non-DMA transfer, then write

VSRead(TestInfo, 25, 0xf0000000 + 300, 5000, Buffer)

Note: To access the individual memory space correctly according to its maximum size, we must have *Offset* + *Count* ≤ Size of Memory Space, and the *Offset* [29:28] are set correctly.

VSDMAInit

Syntax : *WC_RetCode VSDMAInit(WC_TestInfo *TestInfo,unsigned long Count);*

Description: Initialize DMA channel and allocate DMA data buffer.

Arguments: *TestInfo* A point of the index number of a WildCard II FPGA device.

Count Size of the DMA data buffer.

Returns: An enumeration structure *WC_RetCode* [5].

Comments: *VSDMAInit* is used to initialize the DMA data transfer channels and buffer. "*Count*" indicates the Dword size to be allocated for DMA data transfer buffer.

VSDMAClear

Syntax : *WC_RetCode VSDMAClear(WC_TestInfo *TestInfo);*

Description: Release DMA channel and data buffer allocation.

Arguments: *TestInfo* A point of the index number of a WildCard II FPGA device.

Returns: An enumeration structure *WC_RetCode* [5].

Comments: *VSDMAClear* is used to release the allocated DMA channels and data buffer.

See Also: *VSDMAInit*

VSDMAWrite

Syntax : *WC_RetCode VSDMAWrite(WC_TestInfo *TestInfo,*
unsigned int BSDRAM,
unsigned long StartAddr,

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

unsigned long Count,
*DWORD *Buffer);*

Description: Write data to memory space by DMA operations.

Arguments: *TestInfo* A point of the index number of a WildCard II FPGA device.

BSDRAM Indicate which specific memory space is accessed.

StartAddr Start address in the memory space from which data are transferred.

Count Size of the data to be transferred.

Buffer A Pointer to an array of Dwords, *Buffer* Length ≥ *Count*

Returns: An enumeration structure *WC_RetCode* [5].

Comments: *VSDMAWrite* is used to write data from host system to the memory space by DMA operations. There are 3 types of memory address space inside the platform through which data can be transferred by DMA operations, i.e BRAM, SRAM and DRAM. *VSDMAWrite* can automatically map the virtual socket address space into the physical address space, and thus access all the 3 types of physical memory storage within one function call.

BSDRAM indicates which specific memory space will be accessed.

BSDRAM	Memory Accessed
0	BRAM
1	SRAM
2	DRAM

Usage:

(1) if the user IP-block 4 needs to write 512 double words to the BRAM space with the starting offset 0 by DMA transfer, then write

*VSDMAWrite(TestInfo, 0, 0x50008000 + 0x200 * 4 + 0, 512, Buffer)*

(2) if the user IP-block needs to write 1024 double words to the SRAM space with the starting offset 100 by DMA transfer, then write

VSDMAWrite(TestInfo, 1, 0xa0000000 + 100, 1024, Buffer)

(3) if the user IP-block needs to write 4096 double words to the DRAM space with the starting offset 50 by DMA transfer, then write

VSDMAWrite(TestInfo, 2, 0xf0000000 + 50, 4096, Buffer)

VSDMARead

Syntax : *WC_RetCode VSDMARead(WC_TestInfo *TestInfo,*
unsigned int BSDRAM,
unsigned long StartAddr,

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

unsigned long Count,
*DWORD *Buffer);*

Description: Read data from memory space by DMA operations.

Arguments: *TestInfo* A point of the index number of a WildCard II FPGA device.

BSDRAM Indicate which specific memory space is accessed.

StartAddr Start address in the memory space from which data are transferred.

Count Size of the data to be transferred.

Buffer A Pointer to an array of Dwords, *Buffer* Length ≥ *Count*

Returns: An enumeration structure *WC_RetCode* [5].

Comments: *VSDMARead* is used to read data from the memory space to host system by DMA operations. There are 3 types of memory address space inside the platform through which data can be transferred by DMA operations, i.e BRAM, SRAM and DRAM. *VSDMARead* can automatically map the virtual socket address space into the physical address space, and thus access all the 3 types of physical memory storage within one function call.

BSDRAM indicates which specific memory space will be accessed.

BSDRAM	Memory Accessed
0	BRAM
1	SRAM
2	DRAM

Usage:

(1) if the user IP-block 28 needs to read 256 double words from the BRAM space with the starting offset 100 by DMA transfer, then write

*VSRead(TestInfo, 0, 0x50008000 + 0x0200 * 28 + 100, 256, Buffer)*

(2) if the user IP-block needs to read 2000 double words from the SRAM space with the starting offset 500 by DMA transfer, then write

VSRead(TestInfo, 1, 0xa0000000 + 500, 2000, Buffer)

(3) if the user IP-block needs to read 5000 double words from the DRAM space with the starting offset 300 by DMA transfer, then write

VSRead(TestInfo, 2, 0xf0000000 + 300, 5000, Buffer)

HW_Module_Controller

Syntax : *WC_RetCode HW_Module_Controller(WC_TestInfo *TestInfo,*
unsigned long HW_IP_Start);

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description: Trigger and control the user IP-blocks to work.

Arguments: *TestInfo* A point of the index number of a WildCard II FPGA device.

HW_IP_Start Indicate which specific user IP-block is triggered to work.

Returns: An enumeration structure *WC_RetCode* [5].

Comments: *HW_Module_Controller* is used to trigger and control the user IP-block to work.

Each bit of *HW_IP_Start* represents a corresponding IP-block. Before using this function call to trigger IP-blocks to work, users should write memory access type to the platform for each IP-block to indicate which specific memory space the corresponding IP-block will access, including reading and writing. Please refer to the Appendix, an example program, to get understanding about its operation.

Usage:

(1) if the user wants to implement IP-block 5, then write

HW_IP_Start = 0x00000020;

HW_Module_Controller(TestInfo, HW_IP_Start)

(2) if the user wants to implement IP-block 16, then write

HW_IP_Start = 0x00010000;

HW_Module_Controller(TestInfo, HW_IP_Start)

8.2.4 Build the software

To use the platform system software and issue the virtual socket API function calls to trigger the hardware modules to work, we need build a project for compiling all the C source files necessary for the system implementation. Suppose we have “WildCard_Program_Library_Functions.c” which includes all the necessary virtual socket API function calls, “WildCard_Program_Virtual_Socket.c” which is a main file for system implementation and “WildCard_Program_Virtual_Socket.h” which is a C header file. Then we may take the following simple steps to build a project and compile all the files included in the project.

a. Create a new folder under the root directory of an appropriate hard disk,

e.g C:\Digital_Video_Processing_Board_Design\WildCard.

b. Utilize the VC++ software system to create a new project based on Win32 console application under that folder. After the new project is created, a subfolder will be generated automatically as the project directory.

For example, if we create a new project named “VC_WildCard_Program”, then we will have C:\Digital_Video_Processing_Board_Design\WildCard\VC_WildCard_Program as the project directory.

There should be another subfolder automatically generated by VC++ and thus under this project directory:

C:\Digital_Video_Processing_Board_Design\WildCard\VC_WildCard_Program\Debug

c. Copy “WildCard_Program_Virtual_Socket.c” to the project directory and add it to the project as a C source file.

--`,,`,,`,,`,,`,,`,,`,,`,,`,,`,,`,,`,,`,,---
© ISO/IEC 2009 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

d. Copy “WildCard_Program_Library_Functions.c” to the project directory and add it to the project as a source file.

e. Copy “wc_shared.c” to the project directory and add it to the project as a source file, this file is provided by Annapolis.

f. Copy “wcapi.lib” and “wcapi.dll” to the project directory and add “wcapi.lib” as a library file in the project. Besides, copy “wcapi.dll” to the Debug directory.

g. Copy “WildCard_Program_Virtual_Socket.h” to the project directory.

h. Copy “basetsd.h” to the project directory, this file is provided by the VC++ software system.

i. Copy “wc_shared.h”, “wc_windows.h” and “wcdefs.h” to the project directory, those files are provided by Annapolis.

j. Then build all the files within the project and get the correct compiling and linking information.

k. Create an image file directory:

C:\Digital_Video_Processing_Board_Design\WildCard\VC_WildCard_Program\

Debug\XC2V3000\PKG_FG676

Copy appropriate image file .x86 to the image file directory.

l. Run the execution program. This can be done by open a Dos prompt window under operating system. Enter the Debug directory, there should be an .exe file existing, then run it.

m. If the user doesn’t like to open a Dos window and type Dos command to run the program, another way is to run the execution program directly under VC++ environment. In this case, the user should create a directory:

C:\Digital_Video_Processing_Board_Design\WildCard\VC_WildCard_Program\

XC2V3000\PKG_FG676

Copy appropriate image file .x86 to this folder, then execute the program directly under VC++ system environment.

Note: When users need to use their own C source file as the main program, just replace the “WildCard_Program_Virtual_Socket.c” with user’s own C source file and call the appropriate API functions defined above.

8.2.5 Example

The following is an example debug menu of the conformance test for virtual socket platform.

```
//This is a Demo System for Virtual Socket WildCard II Platform.
//                          Developed by
//*************************************************************
//*       Laboratory for Integrated Video Systems             *
//* Department of Electrical and Computer Engineering  *
//*              University of Calgary                                  *
//*             Calgary, Alberta, Canada                          *
//*************************************************************
//    To contact with the authors, please email to
//       badawy@ucalgary.ca or yiqiu@ucalgary.ca
//                  Date: 2005/11/16
//
//The following are global variables, user-defined variables must not
//conflict with those variables.
//
//   WCII_DmaHandle DMAHandle;
```


© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
//   DWORD HardwareAddress;
//   DWORD GrantedDwords;
//   DWORD *DMABuffer;
//   DWORD Interrupt_Register[1],Interrupt_Mask[1];
//   DWORD save_buffer[32][16];
//   boolean Display;
//
//---------------------------------------------------------------------
#include <stdio.h>
#include <stdlib.h>
#include <memory.h>
#include <time.h>
#include <ctype.h>
#include <dos.h>
#include <conio.h>
#if defined(WIN32)
#include <windows.h>
#include <math.h>
#endif
#include "wcdefs.h"
#include "wc_shared.h"
#include "WildCard_Program_Virtual_Socket.h"
//-------------------------------------------------------
void Clrscr()
{
  int i;
  for (i=0;i<45;i++)
  {
     printf("\n");
  }
}
//-------------------------------------------------------
int main(int argc,char *argv[])
{
  WC_RetCode rc=WC_SUCCESS;
  WC_TestInfo TestInfo;
  unsigned long menu_number,Memory_Type,HW_IP_Start;
  boolean menu_tag;

  unsigned int HWAccel;
  unsigned long Offset;
  unsigned long Count;
  unsigned int BSDRAM;
  unsigned long StartAddr;
  unsigned long i,j,k,m;
  DWORD dControl[4]={0,0,0,0};
  DWORD Buffer0[512],Buffer1[512],Buffer2[CIF_DWORD_SIZE],Buffer3[CIF_DWORD_SIZE];
  FILE *infile, *outfile;
  BYTE TempBuffer[1];

  TestInfo.bVerbose    = DEFAULT_VERBOSITY;
  TestInfo.DeviceNum   = DEFAULT_SLOT_NUMBER;
  TestInfo.dIterations = DEFAULT_ITERATIONS;
  TestInfo.fClkFreq    = DEFAULT_FREQUENCY;

  rc = WC_Open(TestInfo.DeviceNum,0);
  DISPLAY_ERROR(rc);

  Clrscr();
  printf("               ******************************************************\n");
  printf("               *                                                    *\n");
  printf("               *                                                    *\n");
  printf("               *         Virtual Socket Platform Demo System        *\n");
  printf("               *                                                    *\n");
  printf("               *                        Copyright(C) 2005           *\n");
  printf("               *                                                    *\n");
  printf("               *                                                    *\n");
  printf("               *                                                    *\n");
  printf("               *                                                    *\n");
  printf("               *     Laboratory for Integrated Video Systems        *\n");
  printf("               *                                                    *\n");
  printf("               *       Electrical and Computer Engineering          *\n");
  printf("               *                                                    *\n");
  printf("               *                    University of Calgary           *\n");
  printf("               *                                                    *\n");
  printf("               *          Calgary, Alberta, Canada T2N 1N4          *\n");
```

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
printf("                *                                                    *\n");
printf("                *              E-mail: badawy@ucalgary.ca            *\n");
printf("                *                          yiqiu@ucalgary.ca         *\n");
printf("                *                                                    *\n");
printf("                *                                                    *\n");
printf("                ******************************************************\n");
printf("\n");
printf("\n");
printf("\n");
printf("\n");
printf("\n");
printf("\n");
printf("\n");
printf("\n");
printf("                           Please press any key to continue...");
menu_number=getch();

rc=VSInit(&TestInfo);
if (rc != WC_SUCCESS)
{
  rc = WC_Close(TestInfo.DeviceNum);
  return rc;
}

Count=CIF_DWORD_SIZE;
rc=VSDMAInit(&TestInfo,Count);
if (rc != WC_SUCCESS)
{
  printf("\n DMA Error!\n");
  return rc;
}

menu_tag=1;
while (menu_tag)
{
Clrscr();
printf("                ************************************************\n");
printf("                *                                              *\n");
printf("                *                                              *\n");
printf("                *                          Main Menu           *\n");
printf("                *                                              *\n");
printf("                *       1->Display WildCard Configuration      *\n");
printf("                *       2->Test Register Data Transfer         *\n");
printf("                *       3->Test Block RAM Data Transfer        *\n");
printf("                *       4->Test Non-DMA SRAM Data Transfer     *\n");
printf("                *       5->Test Non-DMA DRAM Data Transfer     *\n");
printf("                *       6->Test DMA BRAM Data Transfer         *\n");
printf("                *       7->Test DMA SRAM Data Transfer         *\n");
printf("                *       8->Test DMA DRAM Data Transfer         *\n");
printf("                *       9->Video Transfer by DMA SRAM          *\n");
printf("                *      10->Video Transfer by DMA DRAM          *\n");
printf("                *      11->Test User IP Block 0(Data Increase) *\n");
printf("                *      12->Test User IP Block 1(FIFO)          *\n");
printf("                *      13->Test User IP Block 2(STACK)         *\n");
printf("                *      14->Display Interrupt Status            *\n");
printf("                *      15->Clear Interrupt Status              *\n");
printf("                *       0->Exit                                *\n");
printf("                *                                              *\n");
printf("                *                                              *\n");
printf("                *                                              *\n");
printf("                ************************************************\n");
printf("\n");
printf("\n");
printf("\n");
printf("\n");
printf("\n");
printf("\n");

printf("                             Please choose (0-15)->");
scanf("%d",&menu_number);

switch (menu_number)
{
case 1:Clrscr();
           rc=VSInit(&TestInfo);
           rc=VSSystemReset(&TestInfo);
```


© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
            rc=DisplayConfiguration(TestInfo.DeviceNum);
            for (i=0;i<15;i++) printf("\n");
            printf("               Please press any key to return to the main menu...");
            menu_number=getch();
            break;
case 2:Clrscr();
            for (i=0;i<16;i++)
            {
  Buffer0[i]=rand() | (rand() << 14) | (rand() << 28);
            }
            printf(" Please input the number of HWAccel (0-2)->");
            scanf("%d",&HWAccel);
            printf("\n");
            printf(" Please input the Offset (0-15)->");
            scanf("%d",&Offset);
            Offset=0x00000000 + Offset;
            printf("\n");
            printf(" Please input the Count (1-16)->");
            scanf("%d",&Count);
            printf("\n");
            printf(" Now press any key to start writing to Registers...");
            getch();
            Display = 1;
            rc=VSWrite(&TestInfo,HWAccel,Offset,Count,Buffer0);
            rc=VSRead(&TestInfo,HWAccel,Offset,Count,Buffer0);
            for (i=0;i<15;i++) printf("\n");
            printf("               Please press any key to return to the main menu...");
            menu_number=getch();
            break;
case 3:Clrscr();
            for (i=0;i<512;i++)
            {
  Buffer1[i]=rand() | (rand() << 14) | (rand() << 28);
            }
            printf(" Please input the number of HWAccel (0-2)->");
            scanf("%d",&HWAccel);
            printf("\n");
            printf(" Please input the Offset (0-511)->");
            scanf("%d",&Offset);
            Offset=0x50008000 + Offset;
            printf("\n");
            printf(" Please input the Count (1-512)->");
            scanf("%d",&Count);;
            printf("\n");
            printf(" Now press any key to start writing to Block RAM...");
            getch();
            Display = 1;
            rc=VSWrite(&TestInfo,HWAccel,Offset,Count,Buffer1);
            rc=VSRead(&TestInfo,HWAccel,Offset,Count,Buffer1);
            for (i=0;i<5;i++) printf("\n");
            printf("               Please press any key to return to the main menu...");
            menu_number=getch();
            break;
case 4:Clrscr();
             for (i=0;i<512;i++)
             {
    Buffer2[i]=rand() | (rand() << 14) | (rand() << 28);
             }
             printf(" Please input the number of HWAccel (0-2)->");
             scanf("%d",&HWAccel);
             printf("\n");
             printf(" Please input the Offset (0-511)->");
             scanf("%d",&Offset);
             Offset=0xa0000000 + Offset;
             printf("\n");
             printf(" Please input the Count (1-512)->");
             scanf("%d",&Count);
             printf("\n");
             printf(" Now press any key to start writing to SRAM...");
             getch();
             Display = 1;
             rc=VSWrite(&TestInfo,HWAccel,Offset,Count,Buffer2);
             rc=VSRead(&TestInfo,HWAccel,Offset,Count,Buffer2);
             for (i=0;i<5;i++) printf("\n");
             printf("               Please press any key to return to the main menu...");
             menu_number=getch();
```

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
            break;
case 5:Clrscr();
            for (i=0;i<512;i++)
            {
                Buffer3[i]=rand() | (rand() << 14) | (rand() << 28);
            }
            printf(" Please input the number of HWAccel (0-2)->");
            scanf("%d",&HWAccel);
            printf("\n");
            printf(" Please input the Offset (0-511)->");
            scanf("%d",&Offset);
            Offset=0xf0000000 + Offset;
            printf("\n");
            printf(" Please input the Count (1-512)->");
            scanf("%d",&Count);
            printf("\n");
            printf(" Now press any key to start writing to DRAM...");
            getch();
            Display = 1;
            rc=VSWrite(&TestInfo,HWAccel,Offset,Count,Buffer3);
            rc=VSRead(&TestInfo,HWAccel,Offset,Count,Buffer3);
            for (i=0;i<5;i++) printf("\n");
            printf("              Please press any key to return to the main menu...");
            menu_number=getch();
            break;
case 6:Clrscr();
            for (i=0;i<512;i++)
            {
    Buffer1[i]=rand() | (rand() << 14) | (rand() << 28);
            }
            BSDRAM = 0;
            printf(" Please input the Start Address of BRAM to transfer (0,512,1024)->");
            scanf("%d",&StartAddr);
            StartAddr = 0x50008000 + StartAddr;
            printf("\n");
            printf(" Please input the Count (1-512)->");
            scanf("%d",&Count);
            printf("\n");
            printf(" Now press any key to start DMA BRAM data transfer...");
            getch();
            Display = 1;
            rc=VSDMAWrite(&TestInfo,BSDRAM,StartAddr,Count,Buffer1);
            if (rc != WC_SUCCESS)
            {
                printf("\n DMA Error!\n");
    break;
            }
            rc=VSDMARead(&TestInfo,BSDRAM,StartAddr,Count,Buffer1);
            if (rc != WC_SUCCESS)
            {
                printf("\n DMA Error!\n");
    break;
            }
            for (i=0;i<5;i++) printf("\n");
            printf("              Please press any key to return to the main menu...");
            menu_number=getch();
            break;
case 7:Clrscr();
            for (i=0;i<512;i++)
            {
    Buffer2[i]=rand() | (rand() << 14) | (rand() << 28);
            }
            BSDRAM = 1;
            printf(" Please input the Start Address of SRAM to transfer (0-511)->");
            scanf("%d",&StartAddr);
            StartAddr = 0xa0000000 + StartAddr;
            printf("\n");
            printf(" Please input the Count (1-512)->");
            scanf("%d",&Count);
            printf("\n");
            printf(" Now press any key to start DMA SRAM data transfer...");
            getch();
            Display = 1;
            rc=VSDMAWrite(&TestInfo,BSDRAM,StartAddr,Count,Buffer2);
            if (rc != WC_SUCCESS)
            {
```


© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
                printf("\n DMA Error!\n");
    break;
            }
            rc=VSDMARead(&TestInfo,BSDRAM,StartAddr,Count,Buffer2);
            if (rc != WC_SUCCESS)
            {
                printf("\n DMA Error!\n");
    break;
            }
            for (i=0;i<5;i++) printf("\n");
            printf("              Please press any key to return to the main menu...");
            menu_number=getch();
            break;
case 8:Clrscr();
            for (i=0;i<512;i++)
            {
    Buffer3[i]=rand() | (rand() << 14) | (rand() << 28);
            }
            BSDRAM = 2;
            printf(" Please input the Start Address of DRAM to transfer (0-511)->");
            scanf("%d",&StartAddr);
            StartAddr = 0xf0000000 + StartAddr;
            printf("\n");
            printf(" Please input the Count (1-512)->");
            scanf("%d",&Count);
            printf("\n");
            printf(" Now press any key to start DMA DRAM data transfer...");
            getch();
            Display = 1;
            rc=VSDMAWrite(&TestInfo,BSDRAM,StartAddr,Count,Buffer3);
            if (rc != WC_SUCCESS)
            {
                printf("\n DMA Error!\n");
    break;
            }
            rc=VSDMARead(&TestInfo,BSDRAM,StartAddr,Count,Buffer3);
            if (rc != WC_SUCCESS)
            {
                printf("\n DMA Error!\n");
    break;
            }
            for (i=0;i<5;i++) printf("\n");
            printf("              Please press any key to return to the main menu...");
            menu_number=getch();
            break;
case 9:Clrscr();
            if ((infile=fopen("foreman_cif.yuv","rb"))==NULL)
            {
                printf("Input File does not exist!");
                return rc;
            }
            outfile=fopen("foreman_cif_output_SRAM.yuv","wb+");
            for (i=0;i<30;i++)
            {
                for (j=0;j<CIF_DWORD_SIZE;j++)
                {
        fread(TempBuffer,1,1,infile);
        Buffer2[j]=0;
        Buffer2[j]=Buffer2[j]+TempBuffer[0];
                    fread(TempBuffer,1,1,infile);
                    Buffer2[j]=(Buffer2[j]<<8)+TempBuffer[0];
                    fread(TempBuffer,1,1,infile);
                    Buffer2[j]=(Buffer2[j]<<8)+TempBuffer[0];
        fread(TempBuffer,1,1,infile);
                    Buffer2[j]=(Buffer2[j]<<8)+TempBuffer[0];
    }
                printf("\n Processing CIF Frame %3d / 30...",i+1);
                BSDRAM = 1;
    StartAddr = 0;
                Count = CIF_DWORD_SIZE;
    Display = 0;
    rc=VSDMAWrite(&TestInfo,BSDRAM,StartAddr,Count,Buffer2);
    if (rc != WC_SUCCESS)
    {
                     printf("\n DMA Error!\n");
         break;
```

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    }
                rc=VSDMARead(&TestInfo,BSDRAM,StartAddr,Count,Buffer2);
                if (rc != WC_SUCCESS)
                {
                     printf("\n DMA Error!\n");
          break;
                }
    for (j=0;j<CIF_DWORD_SIZE;j++)
    {
                     TempBuffer[0]=Buffer2[j]>>24;
          fwrite(TempBuffer,1,1,outfile);
                     TempBuffer[0]=(Buffer2[j] & 0x00FF0000)>>16;
                      fwrite(TempBuffer,1,1,outfile);
                     TempBuffer[0]=(Buffer2[j] & 0x0000FF00)>>8;
                      fwrite(TempBuffer,1,1,outfile);
                      TempBuffer[0]=(Buffer2[j] & 0x000000FF);
                      fwrite(TempBuffer,1,1,outfile);
                 }
              }
  fclose(infile);
              fclose(outfile);
  for (i=0;i<5;i++) printf("\n");
              printf("               Please press any key to return to the main menu...");
              menu_number=getch();
  break;
case 10:Clrscr();
  if ((infile=fopen("foreman_cif.yuv","rb"))==NULL)
            {
                printf("Input File does not exist!");
                return rc;
            }
            outfile=fopen("foreman_cif_output_DRAM.yuv","wb+");
            for (i=0;i<30;i++)
            {
                for (j=0;j<CIF_DWORD_SIZE;j++)
                {
        fread(TempBuffer,1,1,infile);
        Buffer3[j]=0;
        Buffer3[j]=Buffer3[j]+TempBuffer[0];
                    fread(TempBuffer,1,1,infile);
                    Buffer3[j]=(Buffer3[j]<<8)+TempBuffer[0];
                    fread(TempBuffer,1,1,infile);
                    Buffer3[j]=(Buffer3[j]<<8)+TempBuffer[0];
        fread(TempBuffer,1,1,infile);
                    Buffer3[j]=(Buffer3[j]<<8)+TempBuffer[0];
    }
                printf("\n Processing CIF Frame %3d / 30...",i+1);
                BSDRAM = 2;
    StartAddr = 0;
                Count = CIF_DWORD_SIZE;
    Display = 0;
    rc=VSDMAWrite(&TestInfo,BSDRAM,StartAddr,Count,Buffer3);
    if (rc != WC_SUCCESS)
    {
                     printf("\n DMA Error!\n");
          break;
    }
                rc=VSDMARead(&TestInfo,BSDRAM,StartAddr,Count,Buffer3);
                if (rc != WC_SUCCESS)
                {
                     printf("\n DMA Error!\n");
          break;
                }
    for (j=0;j<CIF_DWORD_SIZE;j++)
    {
                     TempBuffer[0]=Buffer3[j]>>24;
          fwrite(TempBuffer,1,1,outfile);
                     TempBuffer[0]=(Buffer3[j] & 0x00FF0000)>>16;
                      fwrite(TempBuffer,1,1,outfile);
                     TempBuffer[0]=(Buffer3[j] & 0x0000FF00)>>8;
                      fwrite(TempBuffer,1,1,outfile);
                      TempBuffer[0]=(Buffer3[j] & 0x000000FF);
                      fwrite(TempBuffer,1,1,outfile);
                 }
              }
  fclose(infile);
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
            fclose(outfile);
  for (i=0;i<5;i++) printf("\n");
            printf("               Please press any key to return to the main menu...");
            menu_number=getch();
  break;
case 11:Clrscr();
  Display = 0;
  for (i=0;i<16;i++)
            {
      Buffer0[i]=rand() | (rand() << 14) | (rand() << 28);
          }
          for (i=0;i<16;i++)
          {
    Buffer1[i]=rand() | (rand() << 14) | (rand() << 28);
          }
          for (i=0;i<16;i++)
          {
    Buffer2[i]=rand() | (rand() << 14) | (rand() << 28);
          }
          for (i=0;i<16;i++)
          {
              Buffer3[i]=rand() | (rand() << 14) | (rand() << 28);
          }
          rc=VSWrite(&TestInfo,0,0x00000000,16,Buffer0);
          rc=VSWrite(&TestInfo,0,0x50008000,16,Buffer1);
          rc=VSWrite(&TestInfo,0,0xa0000000,16,Buffer2);
          rc=VSWrite(&TestInfo,0,0xf0000000,16,Buffer3);
          printf(" Please input memory type (0:Register,1:BlockRAM,2:SRAM,3:DRAM)->");
          scanf("%d",&Memory_Type);
          Memory_Type = (Memory_Type << 30) + (Memory_Type << 28) + 0x00000000;
          dControl[0] = Memory_Type;
          rc=WC_PeRegWrite(TestInfo.DeviceNum,HW_Mem_HWAccel,1,dControl);
          dControl[0] = 0x00000000;
          rc=WC_PeRegWrite(TestInfo.DeviceNum,Host_Mem_HWAccel,1,dControl);
          dControl[0] = 0x00000000;
          rc=WC_PeRegWrite(TestInfo.DeviceNum,Host_Offset,1,dControl);
          dControl[0] = 0x00000010;
          rc=WC_PeRegWrite(TestInfo.DeviceNum,Host_Count,1,dControl);
          dControl[0] = 0x00000000;
          rc=WC_PeRegWrite(TestInfo.DeviceNum,Host_Mem_HWAccel1,1,dControl);
          dControl[0] = 0x00000000;
          rc=WC_PeRegWrite(TestInfo.DeviceNum,Host_Offset1,1,dControl);
          dControl[0] = 0x00000010;
          rc=WC_PeRegWrite(TestInfo.DeviceNum,Host_Count1,1,dControl);
          printf(" Now press any key to start the work of user ip block...");
          getch();
          HW_IP_Start = 0x00000001;
          rc=HW_Module_Controller(&TestInfo,HW_IP_Start);
          if (Memory_Type == 0x00000000)
          {
               HWAccel = 0;
               Offset = 0x00000000;
    Count = 16;
    Display = 1;
               rc=VSRead(&TestInfo,HWAccel,Offset,Count,Buffer0);
          }
          if (Memory_Type == 0x50000000)
          {
               HWAccel = 0;
    Offset = 0x50008000;
    Count = 16;
    Display = 1;
               rc=VSRead(&TestInfo,HWAccel,Offset,Count,Buffer1);
          }
          if (Memory_Type == 0xa0000000)
          {
               HWAccel = 0;
    Offset = 0xa0000000;
               Count = 16;
    Display = 1;
    rc=VSRead(&TestInfo,HWAccel,Offset,Count,Buffer2);
          }
          if (Memory_Type == 0xf0000000)
          {
               HWAccel = 0;
    Offset = 0xf0000000;
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      Count = 16;
      Display = 1;
      rc=VSRead(&TestInfo,HWAccel,Offset,Count,Buffer3);
             }
             for (i=0;i<10;i++) printf("\n");
             printf("               Please press any key to return to the main menu...");
             menu_number=getch();
             break;
 case 12:Clrscr();
             Display = 0;
             for (i=0;i<16;i++)
             {
    Buffer0[i]=rand() | (rand() << 14) | (rand() << 28);
             }
             for (i=0;i<16;i++)
             {
    Buffer1[i]=rand() | (rand() << 14) | (rand() << 28);
             }
             for (i=0;i<16;i++)
             {
    Buffer2[i]=rand() | (rand() << 14) | (rand() << 28);
             }
             for (i=0;i<16;i++)
             {
                 Buffer3[i]=rand() | (rand() << 14) | (rand() << 28);
             }
             rc=VSWrite(&TestInfo,1,0x00000000,16,Buffer0);
             rc=VSWrite(&TestInfo,1,0x50008000,16,Buffer1);
             rc=VSWrite(&TestInfo,1,0xa0000000,16,Buffer2);
             rc=VSWrite(&TestInfo,1,0xf0000000,16,Buffer3);
             printf(" Please input memory type (0:Register,1:BlockRAM,2:SRAM,3:DRAM)->");
             scanf("%d",&Memory_Type);
             Memory_Type = (Memory_Type << 30) + (Memory_Type << 28) + 0x00000001;
             dControl[0] = Memory_Type;
             rc=WC_PeRegWrite(TestInfo.DeviceNum,HW_Mem_HWAccel,1,dControl);
             dControl[0] = 0x00000001;
             rc=WC_PeRegWrite(TestInfo.DeviceNum,Host_Mem_HWAccel,1,dControl);
             dControl[0] = 0x00000000;
             rc=WC_PeRegWrite(TestInfo.DeviceNum,Host_Offset,1,dControl);
             dControl[0] = 0x00000010;
             rc=WC_PeRegWrite(TestInfo.DeviceNum,Host_Count,1,dControl);
             dControl[0] = 0x00000001;
             rc=WC_PeRegWrite(TestInfo.DeviceNum,Host_Mem_HWAccel1,1,dControl);
             dControl[0] = 0x00000000;
             rc=WC_PeRegWrite(TestInfo.DeviceNum,Host_Offset1,1,dControl);
             dControl[0] = 0x00000010;
             rc=WC_PeRegWrite(TestInfo.DeviceNum,Host_Count1,1,dControl);
             printf(" Now press any key to start the work of user ip block...");
             getch();
             HW_IP_Start = 0x00000002;
             rc=HW_Module_Controller(&TestInfo,HW_IP_Start);
             if (Memory_Type == 0x00000001)
             {
                  HWAccel = 1;
                  Offset = 0x00000000;
      Count = 16;
      Display = 1;
                  rc=VSRead(&TestInfo,HWAccel,Offset,Count,Buffer0);
             }
             if (Memory_Type == 0x50000001)
             {
                  HWAccel = 1;
      Offset = 0x50008000;
      Count = 16;
      Display = 1;
                  rc=VSRead(&TestInfo,HWAccel,Offset,Count,Buffer1);
             }
             if (Memory_Type == 0xa0000001)
             {
                  HWAccel = 1;
      Offset = 0xa0000000;
                  Count = 16;
      Display = 1;
      rc=VSRead(&TestInfo,HWAccel,Offset,Count,Buffer2);
             }
             if (Memory_Type == 0xf0000001)
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2009 – All rights reserved

```
            {
                HWAccel = 1;
     Offset = 0xf0000000;
     Count = 16;
     Display = 1;
     rc=VSRead(&TestInfo,HWAccel,Offset,Count,Buffer3);
            }
            for (i=0;i<10;i++) printf("\n");
            printf("               Please press any key to return to the main menu...");
            menu_number=getch();
            break;
case 13:Clrscr();
            Display = 0;
            for (i=0;i<16;i++)
            {
    Buffer0[i]=rand() | (rand() << 14) | (rand() << 28);
            }
            for (i=0;i<16;i++)
            {
    Buffer1[i]=rand() | (rand() << 14) | (rand() << 28);
            }
            for (i=0;i<16;i++)
            {
    Buffer2[i]=rand() | (rand() << 14) | (rand() << 28);
            }
            for (i=0;i<16;i++)
            {
                Buffer3[i]=rand() | (rand() << 14) | (rand() << 28);
            }
            rc=VSWrite(&TestInfo,2,0x00000000,16,Buffer0);
            rc=VSWrite(&TestInfo,2,0x50008000,16,Buffer1);
            rc=VSWrite(&TestInfo,2,0xa0000000,16,Buffer2);
            rc=VSWrite(&TestInfo,2,0xf0000000,16,Buffer3);
            printf(" Please input memory type (0:Register,1:BlockRAM,2:SRAM,3:DRAM)->");
            scanf("%d",&Memory_Type);
            Memory_Type = (Memory_Type << 30) + (Memory_Type << 28) + 0x00000002;
            dControl[0] = Memory_Type;
            rc=WC_PeRegWrite(TestInfo.DeviceNum,HW_Mem_HWAccel,1,dControl);
            dControl[0] = 0x00000002;
            rc=WC_PeRegWrite(TestInfo.DeviceNum,Host_Mem_HWAccel,1,dControl);
            dControl[0] = 0x00000000;
            rc=WC_PeRegWrite(TestInfo.DeviceNum,Host_Offset,1,dControl);
            dControl[0] = 0x00000010;
            rc=WC_PeRegWrite(TestInfo.DeviceNum,Host_Count,1,dControl);
            dControl[0] = 0x00000002;
            rc=WC_PeRegWrite(TestInfo.DeviceNum,Host_Mem_HWAccel1,1,dControl);
            dControl[0] = 0x00000000;
            rc=WC_PeRegWrite(TestInfo.DeviceNum,Host_Offset1,1,dControl);
            dControl[0] = 0x00000010;
            rc=WC_PeRegWrite(TestInfo.DeviceNum,Host_Count1,1,dControl);
            printf(" Now press any key to start the work of user ip block...");
            getch();
            HW_IP_Start = 0x00000004;
            rc=HW_Module_Controller(&TestInfo,HW_IP_Start);
            if (Memory_Type == 0x00000002)
            {
                HWAccel = 2;
                Offset = 0x00000000;
     Count = 16;
     Display = 1;
                rc=VSRead(&TestInfo,HWAccel,Offset,Count,Buffer0);
            }
            if (Memory_Type == 0x50000002)
            {
                HWAccel = 2;
     Offset = 0x50008000;
     Count = 16;
     Display = 1;
                rc=VSRead(&TestInfo,HWAccel,Offset,Count,Buffer1);
            }
            if (Memory_Type == 0xa0000002)
            {
                HWAccel = 2;
     Offset = 0xa0000000;
                Count = 16;
     Display = 1;
```

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      rc=VSRead(&TestInfo,HWAccel,Offset,Count,Buffer2);
            }
            if (Memory_Type == 0xf0000002)
            {
                 HWAccel = 2;
      Offset = 0xf0000000;
      Count = 16;
      Display = 1;
      rc=VSRead(&TestInfo,HWAccel,Offset,Count,Buffer3);
            }
            for (i=0;i<10;i++) printf("\n");
            printf("              Please press any key to return to the main menu...");
            menu_number=getch();
            break;
  case 14:Clrscr();
  printf("\n User IP Block  0 Interrupt Status = %1x",(Interrupt_Register[0] & 0x00000001));
            printf("\n User IP Block  1 Interrupt Status = %1x",(Interrupt_Register[0] &
0x00000002) >> 1);
  printf("\n User IP Block  2 Interrupt Status = %1x",(Interrupt_Register[0] & 0x00000004) >> 2);
  printf("\n User IP Block  3 Interrupt Status = %1x",(Interrupt_Register[0] & 0x00000008) >> 3);
  printf("\n User IP Block  4 Interrupt Status = %1x",(Interrupt_Register[0] & 0x00000010) >> 4);
  printf("\n User IP Block  5 Interrupt Status = %1x",(Interrupt_Register[0] & 0x00000020) >> 5);
  printf("\n User IP Block  6 Interrupt Status = %1x",(Interrupt_Register[0] & 0x00000040) >> 6);
  printf("\n User IP Block  7 Interrupt Status = %1x",(Interrupt_Register[0] & 0x00000080) >> 7);
  printf("\n User IP Block  8 Interrupt Status = %1x",(Interrupt_Register[0] & 0x00000100) >> 8);
  printf("\n User IP Block  9 Interrupt Status = %1x",(Interrupt_Register[0] & 0x00000200) >> 9);
  printf("\n User IP Block 10 Interrupt Status = %1x",(Interrupt_Register[0] & 0x00000400) >> 10);
  printf("\n User IP Block 11 Interrupt Status = %1x",(Interrupt_Register[0] & 0x00000800) >> 11);
  printf("\n User IP Block 12 Interrupt Status = %1x",(Interrupt_Register[0] & 0x00001000) >> 12);
  printf("\n User IP Block 13 Interrupt Status = %1x",(Interrupt_Register[0] & 0x00002000) >> 13);
  printf("\n User IP Block 14 Interrupt Status = %1x",(Interrupt_Register[0] & 0x00004000) >> 14);
  printf("\n User IP Block 15 Interrupt Status = %1x",(Interrupt_Register[0] & 0x00008000) >> 15);
  printf("\n User IP Block 16 Interrupt Status = %1x",(Interrupt_Register[0] & 0x00010000) >> 16);
  printf("\n User IP Block 17 Interrupt Status = %1x",(Interrupt_Register[0] & 0x00020000) >> 17);
  printf("\n User IP Block 18 Interrupt Status = %1x",(Interrupt_Register[0] & 0x00040000) >> 18);
  printf("\n User IP Block 19 Interrupt Status = %1x",(Interrupt_Register[0] & 0x00080000) >> 19);
  printf("\n User IP Block 20 Interrupt Status = %1x",(Interrupt_Register[0] & 0x00100000) >> 20);
  printf("\n User IP Block 21 Interrupt Status = %1x",(Interrupt_Register[0] & 0x00200000) >> 21);
  printf("\n User IP Block 22 Interrupt Status = %1x",(Interrupt_Register[0] & 0x00400000) >> 22);
  printf("\n User IP Block 23 Interrupt Status = %1x",(Interrupt_Register[0] & 0x00800000) >> 23);
  printf("\n User IP Block 24 Interrupt Status = %1x",(Interrupt_Register[0] & 0x01000000) >> 24);
  printf("\n User IP Block 25 Interrupt Status = %1x",(Interrupt_Register[0] & 0x02000000) >> 25);
  printf("\n User IP Block 26 Interrupt Status = %1x",(Interrupt_Register[0] & 0x04000000) >> 26);
  printf("\n User IP Block 27 Interrupt Status = %1x",(Interrupt_Register[0] & 0x08000000) >> 27);
  printf("\n User IP Block 28 Interrupt Status = %1x",(Interrupt_Register[0] & 0x10000000) >> 28);
  printf("\n User IP Block 29 Interrupt Status = %1x",(Interrupt_Register[0] & 0x20000000) >> 29);
  printf("\n User IP Block 30 Interrupt Status = %1x",(Interrupt_Register[0] & 0x40000000) >> 30);
  printf("\n User IP Block 31 Interrupt Status = %1x",(Interrupt_Register[0] & 0x80000000) >> 31);
            for (i=0;i<10;i++) printf("\n");
            printf("              Please press any key to return to the main menu...");
            menu_number=getch();
            break;
  case 15:Clrscr();
            Interrupt_Register[0] = 0x00000000;
            rc=VSIntStatusWrite(&TestInfo,Interrupt_Register,Interrupt_Mask);
            rc=VSIntStatusRead(&TestInfo,Interrupt_Register,Interrupt_Mask);
  printf("\n User IP Block  0 Interrupt Status = %1x",(Interrupt_Register[0] & 0x00000001));
  printf("\n User IP Block  1 Interrupt Status = %1x",(Interrupt_Register[0] & 0x00000002) >> 1);
  printf("\n User IP Block  2 Interrupt Status = %1x",(Interrupt_Register[0] & 0x00000004) >> 2);
  printf("\n User IP Block  3 Interrupt Status = %1x",(Interrupt_Register[0] & 0x00000008) >> 3);
  printf("\n User IP Block  4 Interrupt Status = %1x",(Interrupt_Register[0] & 0x00000010) >> 4);
  printf("\n User IP Block  5 Interrupt Status = %1x",(Interrupt_Register[0] & 0x00000020) >> 5);
  printf("\n User IP Block  6 Interrupt Status = %1x",(Interrupt_Register[0] & 0x00000040) >> 6);
  printf("\n User IP Block  7 Interrupt Status = %1x",(Interrupt_Register[0] & 0x00000080) >> 7);
  printf("\n User IP Block  8 Interrupt Status = %1x",(Interrupt_Register[0] & 0x00000100) >> 8);
  printf("\n User IP Block  9 Interrupt Status = %1x",(Interrupt_Register[0] & 0x00000200) >> 9);
  printf("\n User IP Block 10 Interrupt Status = %1x",(Interrupt_Register[0] & 0x00000400) >> 10);
  printf("\n User IP Block 11 Interrupt Status = %1x",(Interrupt_Register[0] & 0x00000800) >> 11);
  printf("\n User IP Block 12 Interrupt Status = %1x",(Interrupt_Register[0] & 0x00001000) >> 12);
  printf("\n User IP Block 13 Interrupt Status = %1x",(Interrupt_Register[0] & 0x00002000) >> 13);
  printf("\n User IP Block 14 Interrupt Status = %1x",(Interrupt_Register[0] & 0x00004000) >> 14);
  printf("\n User IP Block 15 Interrupt Status = %1x",(Interrupt_Register[0] & 0x00008000) >> 15);
  printf("\n User IP Block 16 Interrupt Status = %1x",(Interrupt_Register[0] & 0x00010000) >> 16);
  printf("\n User IP Block 17 Interrupt Status = %1x",(Interrupt_Register[0] & 0x00020000) >> 17);
  printf("\n User IP Block 18 Interrupt Status = %1x",(Interrupt_Register[0] & 0x00040000) >> 18);
  printf("\n User IP Block 19 Interrupt Status = %1x",(Interrupt_Register[0] & 0x00080000) >> 19);
```


© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
  printf("\n User IP Block 20 Interrupt Status = %1x",(Interrupt_Register[0] & 0x00100000) >> 20);
  printf("\n User IP Block 21 Interrupt Status = %1x",(Interrupt_Register[0] & 0x00200000) >> 21);
  printf("\n User IP Block 22 Interrupt Status = %1x",(Interrupt_Register[0] & 0x00400000) >> 22);
  printf("\n User IP Block 23 Interrupt Status = %1x",(Interrupt_Register[0] & 0x00800000) >> 23);
  printf("\n User IP Block 24 Interrupt Status = %1x",(Interrupt_Register[0] & 0x01000000) >> 24);
  printf("\n User IP Block 25 Interrupt Status = %1x",(Interrupt_Register[0] & 0x02000000) >> 25);
  printf("\n User IP Block 26 Interrupt Status = %1x",(Interrupt_Register[0] & 0x04000000) >> 26);
  printf("\n User IP Block 27 Interrupt Status = %1x",(Interrupt_Register[0] & 0x08000000) >> 27);
  printf("\n User IP Block 28 Interrupt Status = %1x",(Interrupt_Register[0] & 0x10000000) >> 28);
  printf("\n User IP Block 29 Interrupt Status = %1x",(Interrupt_Register[0] & 0x20000000) >> 29);
  printf("\n User IP Block 30 Interrupt Status = %1x",(Interrupt_Register[0] & 0x40000000) >> 30);
  printf("\n User IP Block 31 Interrupt Status = %1x",(Interrupt_Register[0] & 0x80000000) >> 31);
              for (i=0;i<10;i++) printf("\n");
              printf("              Please press any key to return to the main menu...");
              menu_number=getch();
              break;
  case 0:menu_tag=0;
              Clrscr();
              printf("
************************************************************\n");
              printf("
*                                                                                *\n");
              printf("
*                                                                                *\n");
              printf("           *      Thanks for using Virtual Socket Demo System!     *\n");
              printf("
*                                                                                *\n");
              printf("
*                                                                                *\n");
              printf("
************************************************************\n");
              for (i=0;i<25;i++) printf("\n");
              break;
  default:
              break;
  }
  }

  rc=VSDMAClear(&TestInfo);
  if (rc != WC_SUCCESS)
  {
    printf("\n DMA Error!\n");
  }
  rc = WC_Close(TestInfo.DeviceNum);
  return(rc);
}
```

8.3 Tutorial on the Integrated Virtual Socket Hardware-Accelerated Co-design Platform for MPEG-4 Part 9 Implementation 3

8.3.1 Scope

The objective of this tutorial subclause is to address the following issues about an integrated virtual socket platform system developed at the University of Calgary for the integration of hardware modules with the MPEG4 reference software.

1. Is there a guide for how to add user MPEG4 compliant IP-blocks to the platform?

2. Is there a guide for how to program and use the platform software?

3. Is there an API C source code that abstracts the Wildcard II function calls away?

4. Is there a debug menu to show all of the supported features in the platform?

5. More questions to be added…

The subclause is formatted in the form of a tutorial that makes the readers get basic knowledge about an integrated virtual socket platform system and takes them step by step through a typical module development case.

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

This subclause includes the following ten subsections.

- Introduction
- Virtual socket API function calls
- Address mapping
- Integration of the user IP-blocks
- Some examples of the user IP-blocks
- A guide for integrating modules within the PE system
- A guide for using platform system software
- Conformance tests
- Reference
- Appendix

8.3.2 Introduction

To facilitate the development of IP-blocks for MPEG4 Part 9 Reference Hardware, we have an integrated virtual socket hardware-accelerated co-design platform. The virtual socket indicates a type of interface that can map virtual memory space to the physical storage. The virtual memory space is described in the software and the physical storage is based on the Wildcard II FPGA device [5]. The goal of the platform design is to address the data communications and control signal exchanges between a host computer system and a reconfigurable FPGA co-processor based on the concept of virtual socket.

The generic platform system employs four types of memory space to transfer data and thus provide the infrastructure for the purpose of user IP-blocks that are not specifically designed for interaction with a host computer system. Actually, the user IP-block can be any hardware module designed to handle a computationally extensive task for MPEG4 applications.

The following is a block diagram for this virtual socket platform.

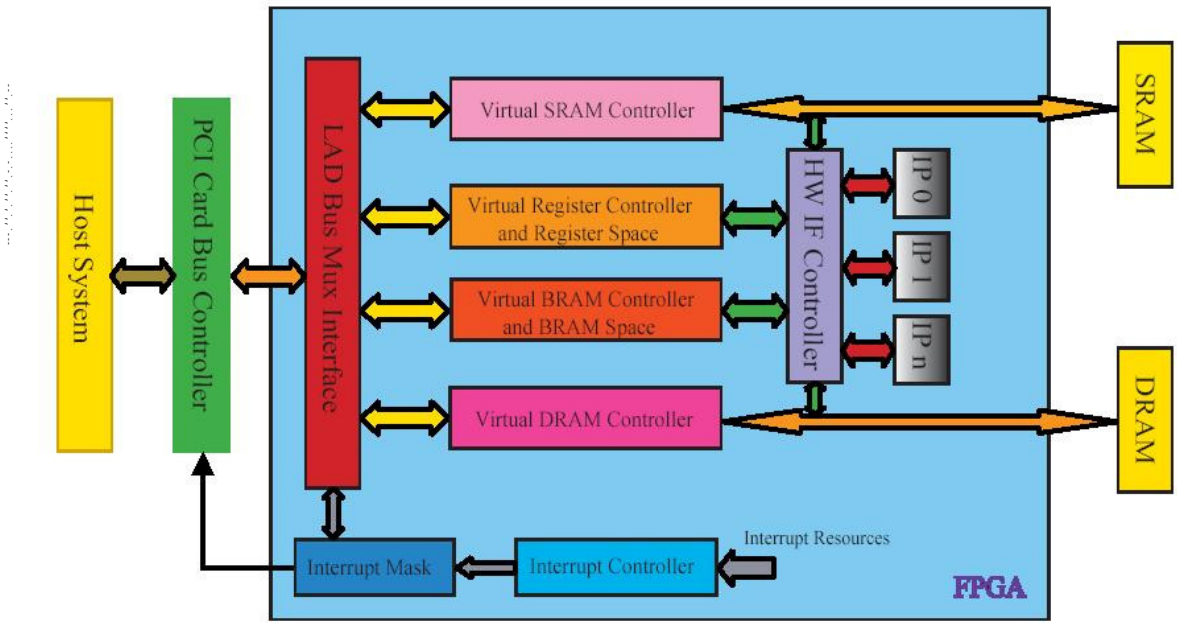


Figure 60 — A block diagram of the virtual socket platform architecture.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The architecture of the virtual socket platform includes PCI Card Bus Controller, FPGA logic and four types of physical memory space, i.e Registers, BlockRAM, ZBT-SRAM and DDR-DRAM. Two types of the memory space, Registers and BRAM, are integrated in the FPGA as the logic resources. Other two types of the memory space, ZBT-SRAM and DDR-DRAM, are external physical storage. Because this platform system design is constructed on the Wildcard II FPGA device, we can directly utilize its PCI Card Bus Controller which is integrated in the Wildcard II.

Some fundamental hardware module controllers are integrated into the virtual socket platform to perform data communication between host computer system and memory spaces. Those hardware module controllers include virtual register controller, virtual BlockRAM controller, virtual SRAM controller and virtual DRAM controller. Each module is in charge of the data transfer as well as the physical memory address mapping for the corresponding memory storage. Besides, to integrate the user IP-blocks into the whole platform PE system, there is a hardware interface controller. The user MPEG4 compliant IP-blocks (e.g DCT, IDCT, Quatizaton, VLC, ME, MC etc.), are connected to this hardware interface controller. It can provide user IP-blocks required data, and IP-blocks process those data and feed them back to the hardware interface controller. After receiving the feed-back data, the hardware interface controller stores the processed data directly into physical memory units. The hardware interface controller provides an easy way for the user IP-blocks to exchange data between the host system and memory spaces. Interrupt signals can be generated when user IP-blocks finish processing or exchanging the data.

For part of the software, the virtual socket API function calls which are coded in C language are set up in the host system. Developers may provide those function calls some parameters, and make the virtual socket API function calls work together with the system hardware modules and automatically map the virtual memory addresses into any type of the physical memory space inside the platform. With such mechanism, the data can be transferred between the host computer system and any type of the physical memory space within only one function call. Anyway, the virtual socket interface is some sort of virtual memory management interface for the hardware-accelerated platform framework.

To verify the functionality of the platform design, a debug menu is provided and interactively test all the features integrated in the virtual socket platform.

8.3.3 Virtual Socket API Function Calls

Application Programming Interface (API) is a set of functions coded in C language allowing data communication between host computer system and Wildcard II platform. Actually, the virtual socket API function calls here are not simply the Wildcard II API functions. They are constructed on the basic Wildcard II API functions to perform the data communication between the host system and virtual socket platform.

Currently, the following virtual socket API function calls are defined.

WC_Open	Open a WildCard II FPGA device.
WC_Close	Close a WildCard II FPGA device.
VSInit	Initialize the virtual socket platform.
VSSystemReset	Reset the virtual socket platform.
VSUserResetEnable	Enable a reset for user IP-blocks.
VSUserResetDisable	Disable a reset for user IP-blocks.
VSIntEnable	Enable / Disable the platform interrupt generation
VSIntClear	Clean up the platform interrupt.
VSIntCheck	Check and wait for the platform interrupt generation.
VSIntStatusWrite	Set up the interrupt status and mask registers.

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

VSIntStatusRead Read the interrupt status and mask registers.

VSWrite Write data to memory space by non-DMA operations.

VSRead Read data from memory space by non-DMA operations.

VSDMAInit Initialize DMA channel and allocate DMA data buffer.

VSDMAClear Release DMA channel and data buffer allocation.

VSDMAWrite Write data to memory space by DMA operations.

VSDMARead Read data from memory space by DMA operations.

HW_Module_Controller Trigger and control the user IP-blocks to work.

Note: Please refer to the “Reference for Virtual Socket API Function Calls” for more details about the description and usage for each virtual socket API function call.

8.3.4 Address Mapping

The virtual socket platform memory space is mapped to 4 types of physical storage, i.e Registers, Block RAM, ZBT-SRAM and DDR-DRAM. We define a maximum of 32 user hardware accelerators (IP-blocks) which can be integrated into the platform. The data for each user IP-block can be stored in any type of the physical memory space mentioned above. Currently, we define the size of register space is 16 double words for each user IP-block, BRAM 512 double words for each user IP-block, SRAM 2MB for all the user IP-blocks, and DRAM 32MB for all the user IP-blocks. The size of each memory unit is double word based.

8.3.4.1 Address Mapping for Register Space

Data stored in register space have a maximum size of 16 double words for each user IP-block. Usually, the data stored in register space are some parameters for processing the video data algorithms or hardware modules. Address mapping for register space is listed in Table 10.

Table 10 — Address Mapping for Register Space.

HWAccel Number	Virtual Address	Physical Address	HWAccel Number	Virtual Address	Physical Address
0	0x0000-0x000f	0x0000-0x000f	16	0x2000-0x200f	0x2000-0x200f
1	0x0200-0x020f	0x0200-0x020f	17	0x2200-0x220f	0x2200-0x220f
2	0x0400-0x040f	0x0400-0x040f	18	0x2400-0x240f	0x2400-0x240f
3	0x0600-0x060f	0x0600-0x060f	19	0x2600-0x260f	0x2600-0x260f
4	0x0800-0x080f	0x0800-0x080f	20	0x2800-0x280f	0x2800-0x280f
5	0x0a00-0x0a0f	0x0a00-0x0a0f	21	0x2a00-0x2a0f	0x2a00-0x2a0f
6	0x0c00-0x0c0f	0x0c00-0x0c0f	22	0x2c00-0x2c0f	0x2c00-0x2c0f
7	0x0e00-0x0e0f	0x0e00-0x0e0f	23	0x2e00-0x2e0f	0x2e00-0x2e0f
8	0x1000-0x100f	0x1000-0x100f	24	0x3000-0x300f	0x3000-0x300f

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9	0x1200-0x120f	0x1200-0x120f	25	0x3200-0x320f	0x3200-0x320f
10	0x1400-0x140f	0x1400-0x140f	26	0x3400-0x340f	0x3400-0x340f
11	0x1600-0x160f	0x1600-0x160f	27	0x3600-0x360f	0x3600-0x360f
12	0x1800-0x180f	0x1800-0x180f	28	0x3800-0x380f	0x3800-0x380f
13	0x1a00-0x1a0f	0x1a00-0x1a0f	29	0x3a00-0x3a0f	0x3a00-0x3a0f
14	0x1c00-0x1c0f	0x1c00-0x1c0f	30	0x3c00-0x3c0f	0x3c00-0x3c0f
15	0x1e00-0x1e0f	0x1e00-0x1e0f	31	0x3e00-0x3e0f	0x3e00-0x3e0f

8.3.4.2 Address Mapping for BRAM Space

Data stored in Block RAM space have a maximum size of 512 double words for each user IP-block. Address mapping for BRAM space is listed in Table 11.

Table 11 — Address Mapping for BRAM Space.

HWAccel Number	Virtual Address	Physical Address	HWAccel Number	Virtual Address	Physical Address
0	0x0000-0x01ff	0x8000-0x81ff	16	0x2000-0x21ff	0xa000-0xa1ff
1	0x0200-0x03ff	0x8200-0x83ff	17	0x2200-0x23ff	0xa200-0xa3ff
2	0x0400-0x05ff	0x8400-0x85ff	18	0x2400-0x25ff	0xa400-0xa5ff
3	0x0600-0x07ff	0x8600-0x87ff	19	0x2600-0x27ff	0xa600-0xa7ff
4	0x0800-0x09ff	0x8800-0x89ff	20	0x2800-0x29ff	0xa800-0xa9ff
5	0x0a00-0x0bff	0x8a00-0x8bff	21	0x2a00-0x2bff	0xaa00-0xabff
6	0x0c00-0x0dff	0x8c00-0x8dff	22	0x2c00-0x2dff	0xac00-0xadff
7	0x0e00-0x0fff	0x8e00-0x8fff	23	0x2e00-0x2fff	0xae00-0xafff
8	0x1000-0x11ff	0x9000-0x91ff	24	0x3000-0x31ff	0xb000-0xb1ff
9	0x1200-0x13ff	0x9200-0x93ff	25	0x3200-0x33ff	0xb200-0xb3ff
10	0x1400-0x15ff	0x9400-0x95ff	26	0x3400-0x35ff	0xb400-0xb5ff
11	0x1600-0x17ff	0x9600-0x97ff	27	0x3600-0x37ff	0xb600-0xb7ff
12	0x1800-0x19ff	0x9800-0x99ff	28	0x3800-0x39ff	0xb800-0xb9ff
13	0x1a00-0x1bff	0x9a00-0x9bff	29	0x3a00-0x3bff	0xba00-0xbbff
14	0x1c00-0x1dff	0x9c00-0x9dff	30	0x3c00-0x3dff	0xbc00-0xbdff
15	0x1e00-0x1fff	0x9e00-0x9fff	31	0x3e00-0x3fff	0xbe00-0xbfff

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.3.4.3 Address Mapping for SRAM Space

ZBT-SRAM is a large size memory space for video data storage. Actually, each user IP-block can utilize the whole address space of the SRAM for storing the video data. The SRAM size is up to 2M Bytes, i.e 524,288 double words. Address mapping for SRAM space is listed in Table 12.

Table 12 — Address Mapping for SRAM Space

HWAccel Number	Virtual Address	Physical Address
0 – 31	0x00000000-0x0007ffff	0x00000000-0x0007ffff

8.3.4.4 Address Mapping DRAM Space

Similar to the ZBT-SRAM space, DDR-DRAM is also a large size memory space for video data storage. Each user IP-block can utilize the whole address space of the DRAM for storing the video data. Currently, the DRAM size can be up to 32M Bytes, i.e 8,388,608 double words. Address mapping for DRAM space is listed in Table 13.

Table 13 — Address Mapping for DRAM Space.

HWAccel Number	Virtual Address	Physical Address
0 – 31	0x00000000-0x007fffff	0x00000000-0x007fffff

8.3.5 Integration of the user IP-blocks

Now we have 4 types of the memory space in the platform as well as the virtual socket API function calls which can be employed to transfer the data through the memory space. Then how the users integrate their own IP-blocks into the platform? As we define that there is a maximum of 32 user IP-blocks can be integrated into the platform, it is very likely that each user IP-block has its own requirement for the interface between the platform and IP-block. To facilitate the platform development and make the integration easy, we should define a common set of interface signals between the platform and each user IP-block. Therefore, we have the following interface signals between the platform and each user IP-block listed in Figure 61:

```
entity User_IP_Block is
generic(block_width: positive:=8);
port(
     clk                          : in std_logic;
     reset                        : in std_logic;
     start                        : in std_logic;
     input_valid                  : in std_logic;
     data_in                      : in std_logic_vector(31 downto 0);
     memory_read                  : out std_logic;
     HW_Mem_HWAccel               : out std_logic_vector(31 downto 0);
     HW_Offset                    : out std_logic_vector(31 downto 0);
     HW_Count                     : out std_logic_vector(31 downto 0);
     data_out                     : out std_logic_vector(31 downto 0);
     output_valid                 : out std_logic;
     HW_Mem_HWAccel1              : out std_logic_vector(31 downto 0);
     HW_Offset1                   : out std_logic_vector(31 downto 0);
     HW_Count1                    : out std_logic_vector(31 downto 0);
     output_ack                   : in std_logic;
     Host_Mem_HWAccel             : in std_logic_vector(31 downto 0);
     Host_Offset                  : in std_logic_vector(31 downto 0);
     Host_Count                   : in std_logic_vector(31 downto 0);
     Host_Mem_HWAccel1            : in std_logic_vector(31 downto 0);
     Host_Offset1                 : in std_logic_vector(31 downto 0);
     Host_Count1                  : in std_logic_vector(31 downto 0);
     Mem_Read_Access_Req          : out std_logic;
     Mem_Write_Access_Req         : out std_logic;
     Mem_Read_Access_Release_Req  : out std_logic;
     Mem_Write_Access_Release_Req : out std_logic;
     Mem_Read_Access_Ack          : in std_logic;
     Mem_Write_Access_Ack         : in std_logic;
```


© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    Mem_Read_Access_Release_Ack  : in std_logic;
    Mem_Write_Access_Release_Ack : in std_logic;
    done                         : out std_logic
    );
end User_IP_Block;
```

Figure 61 — Interface Signals between Platform and IP-blocks.

Assumptions:

a. System is positive edge trigged.

b. Reset is active high.

Interface Signal Descriptions:

clk The platform system clock input to the user IP-block

reset Reset signal for user IP-block

start When the platform triggers user IP-block to work, this signal will be asserted

input_valid When data are ready to be input to the user IP-block, this signal is valid

data_in Input data to the user IP-block

memory_read When user IP-block needs to read some data from memory space, this signal is asserted for platform to initialize the memory access controllers

HW_Mem_HWAccel To read data from memory space, user IP-block provides the platform this hardware accelerator number for the platform maps the virtual socket address to physical address

HW_Offset To read data from memory space, user IP-block provides the platform this offset for the platform determines the starting address to be accessed

HW_Count To read data from memory space, user IP-block provides the platform this count for the platform determines the data length to be accessed

data_out Output data to the memory space

output_valid When data are ready to be output to the memory space, this signal is valid

HW_Mem_HWAccel1 To write data to memory space, user IP-block provides the platform this hardware accelerator number for the platform maps the virtual socket address to physical address

HW_Offset1 To write data to memory space, user IP-block provides the platform this offset for the platform determines the starting address to be accessed

HW_Count1 To write data to memory space, user IP-block provides the platform this count for the platform determines the data length to be accessed

output_ack When platform successfully receives the data from user IP-block, this signal is asserted to acknowledge it

Host_Mem_HWAccel Besides the user IP-block itself can provide hardware accelerator number to the platform for memory reading operations, the host system can optionally provide the user IP-blocks with this initial hardware accelerator number. Therefore, if the user IP-block doesn't want to generate its own hardware accelerator number to be provided to the platform, it can just accept this parameter signal as its desiring hardware accelerator number for memory reading operation.

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Host_Offset Provided by host system as an optional parameter signal to generate the necessary memory starting offset address for memory reading operation.

Host_Count Provided by host system as an optional parameter signal to generate the necessary data counter for memory reading operation.

Host_Mem_HWAccel1 Similar to Host_Mem_HWAccel, if the user IP-block doesn't want to generate its own hardware accelerator number to be provided to the platform, it can just accept this parameter signal as its desiring hardware accelerator number for memory writing operation.

Host_Offset1 Similar to Host_Offset, this signal is provided by host system as an optional parameter to generate the necessary memory starting offset address for memory writing operation.

Host_Count1 Similar to Host_Count, this signal is provided by host system as an optional parameter to generate the necessary data counter for memory writing operation.

Mem_Read_Access_Req User IP-block issue this signal to request the memory reading.

Mem_Read_Access_Ack Platform generates this signal to acknowledge the user request.

Mem_Read_Access_Release_Req When user IP-block finishes the memory reading operations, it asserts this signal to ask for releasing the memory reading.

Mem_Read_Access_Release_Ack Platform generates this signal to acknowledge the user releasing request, and therefore the memory is released for the reading operations.

Mem_Write_Access_Req User IP-block issue this signal to request the memory writing.

Mem_Write_Access_Ack Platform generates this signal to acknowledge the user request.

Mem_Write_Access_Release_Req When user IP-block finishes the memory writing operations, it asserts this signal to ask for releasing the memory writing.

Mem_Write_Access_Release_Ack Platform generates this signal to acknowledge the user releasing request, and therefore the memory is released for the writing operations.

done The user interrupt signal to the platform

Data flow between the platform and user IP-blocks:

(1) Host computer system writes data to the memory space.

(2) Host computer system issues a command to trigger the relevant user IP-block starting to work.

(3) Once the user IP-block starts to work, it will issue "Mem_Read_Access_Req" to ask for reading the memory.

(4) Platform accepts the user reading request and if memory is available, it generates an acknowledgement signal "Mem_Read_Access_Ack" to the user IP-block.

(5) Once the user IP-block receives the acknowledgement signal, it can ask for reading some data directly from the memory space. This request will be performed by asserting "memory_read" together with setting up some other parameter signals, i.e "HW_Mem_HWAccel", "HW_Offset", "HW_Count".

(6) The platform accepts those signals and reads data from the memory space according to the parameters. When the platform finishes each reading, it asserts "input_valid" and the data are ready in the "data_in" to be input to the user IP-block.

(7) The user IP-block receives the data from the interface.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

(8) User IP-block will assert “Mem_Read_Access_Release_Req” to ask for releasing the reading operations when finished.

(9) Platform generates an acknowledgement signal “Mem_Read_Access_Release_Ack” to release the reading operations.

(10) Then user IP-block will perform the video algorithms or computations to implement the video application functionality.

(11) After video processing finished, user IP-block asserts “Mem_Write_Access_Req” to ask for writing operations.

(12) Platform accepts the user writing request and if memory is available, it generates an acknowledgement signal “Mem_Write_Access_Ack” to the user IP-block.

(13) Then user IP-block asserts “output_valid” together with some other parameters, i.e “HW_Mem_HWAccel1”, “HW_Offset1”, “HW_Count1” to request writing data back to the memory space.

(14) The platform accepts “output_valid”, receives the data from “data_out” and writes data back to the memory space according to the parameter signals. When each writing is finished, the platform asserts “output_ack” to acknowledge the writing process.

(15) User IP-block will assert “Mem_Write_Access_Release_Req” to ask for releasing the writing operations when all the processing is finished.

(16) Platform generates an acknowledgement signal “Mem_Write_Access_Release_Ack” to release the writing operations.

(17) A user interrupt signal is asserted to inform the platform that the user IP-block finishes all the internal tasks.

(18) A final interrupt signal is asserted (if the interrupt is enabled) by the platform to inform the host computer system that the current task is done.

The key point of the interaction between user IP-block and platform is that the user IP-block can utilize some parameter signals to ask for reading and writing data through the memory space. In such case, the user IP-block positively asks for accessing the memory space and the platform passively processes the requests from user IP-blocks and thus accesses data through memory space. The memory data access is implemented by the platform itself. User IP-blocks only need to issue the memory requests together with parameter signals to the interface, and other part of the work is taken by the platform. This feature is particularly useful when there is motion estimation or motion compensation hardware modules exiting in the platform.

Here is an example template for user IP-block which can be integrated into the platform. The VHDL codes are shown in Figure 62. Some notes are added to this example. Among the notes, “template” indicates where the template codes are used and “user code” shows where the users should insert their own VHDL codes there. For this example, parameter signals are set up as HW_Mem_HWAccel/HW_Mem_HWAccel1 = 0, HW_Offset/HW_Offset1 = 0, HW_Count/HW_Count1 = 256. The users may change those parameters according to their own requirement in practical cases.

If users choose to use Host_Mem_HWAccel, Host_Offset, Host_Count, Host_Mem_HWAccel1, Host_Offset1, Host_Count1 as their parameter signals, then set up the following in the VHDL codes:

HW_Mem_HWAccel <=Host_ Mem_HWAccel,

HW_Offset <= Host_Offset,

HW_Count <= Host_Count,

HW_Mem_HWAccel1 <= Host_Mem_HWAccel1,

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

HW_Offset1 <= Host_Offset1,

HW_Count1 <= Host_Count1

```
library ieee;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;
entity User_IP_Block_0 is
generic(block_width: positive:=8);
port(
    clk                         : in std_logic;
    reset                       : in std_logic;
    start                       : in std_logic;
    input_valid                 : in std_logic;
    data_in                     : in std_logic_vector(31 downto 0);
    memory_read                 : out std_logic;
    HW_Mem_HWAccel              : out std_logic_vector(31 downto 0);
    HW_Offset                   : out std_logic_vector(31 downto 0);
    HW_Count                    : out std_logic_vector(31 downto 0);
    data_out                    : out std_logic_vector(31 downto 0);
    output_valid                : out std_logic;
    HW_Mem_HWAccel1             : out std_logic_vector(31 downto 0);
    HW_Offset1                  : out std_logic_vector(31 downto 0);
    HW_Count1                   : out std_logic_vector(31 downto 0);
    output_ack                  : in std_logic;
    Host_Mem_HWAccel            : in std_logic_vector(31 downto 0);
    Host_Offset                 : in std_logic_vector(31 downto 0);
    Host_Count                  : in std_logic_vector(31 downto 0);
    Host_Mem_HWAccel1           : in std_logic_vector(31 downto 0);
    Host_Offset1                : in std_logic_vector(31 downto 0);
    Host_Count1                 : in std_logic_vector(31 downto 0);
    Mem_Read_Access_Req         : out std_logic;
    Mem_Write_Access_Req        : out std_logic;
    Mem_Read_Access_Release_Req : out std_logic;
    Mem_Write_Access_Release_Req : out std_logic;
    Mem_Read_Access_Ack         : in std_logic;
    Mem_Write_Access_Ack        : in std_logic;
    Mem_Read_Access_Release_Ack : in std_logic;
    Mem_Write_Access_Release_Ack : in std_logic;
    done                        : out std_logic
     );
end User_IP_Block_0;

architecture behav of User_IP_Block_0 is

type memory32bit is array (natural range <>) of std_logic_vector(31     downto 0);

signal data_buffer  : memory32bit(0 to 255);
signal user_status  : integer range 0 to 31;

begin
  process(reset,clk)
  variable i : integer range 0 to 256;
  variable j : integer range 0 to 256;
  begin
if (reset = '1') then                                --template
  output_valid <= '0';                               --template
i := 0;                                             --template
      j := 0;                                            --template
      user_status <= 0;                                  --template
      memory_read <= '0';                                --template
      Mem_Read_Access_Req <= '0';                        --template
      Mem_Write_Access_Req <= '0';                       --template
      Mem_Read_Access_Release_Req <= '0';                --template
      Mem_Write_Access_Release_Req <= '0';               --template
      done <= '1';                                       --template
elsif rising_edge(clk) then                           --template
      if (start = '1') then                              --template
        case user_status is                              --template
        when 0 =>                                        --template
          done <= '0';                                   --template
```


© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
          user_status <= 1;                                --template
        when 1 =>
          Mem_Read_Access_Req <= '1';                      --template
          user_status <= 2;                                --template
        when 2 =>                                          --template
          if (Mem_Read_Access_Ack = '1') then              --template
            Mem_Read_Access_Req <= '0';                    --template
            user_status <= 3;                              --template
          else                                             --template
            user_status <= 2;                              --template
          end if;
        when 3 =>                                          --template
          HW_Mem_HWAccel <= x"00000000";                   --user code
          HW_Offset <= x"00000000";                        --user code
          HW_Count <= x"00000100";                         --user code
          j := 0;                                          --template
          user_status <= 4;                                --template
        when 4 =>                                          --template
          memory_read <= '1';                              --template
          user_status <= 5;                                --template
        when 5 =>                                          --template
          memory_read <= '0';                              --template
          if (input_valid = '1') then                      --template
            data_buffer(j) <= data_in;                     --template
            j := j + 1;                                    --template
            if (j >= conv_integer(unsigned(HW_Count))) then
              j := 0;                                      --template
              user_status <= 6;                            --template
            else                                           --template
              user_status <= 5;                            --template
            end if;                                        --template
          end if;                                          --template
        when 6 =>
          Mem_Read_Access_Release_Req <= '1';              --template
      user_status <= 7;                                --template
        when 7 =>
          if (Mem_Read_Access_Release_Ack = '1') then    --template
            Mem_Read_Access_Release_Req <= '0';          --template
            user_status <= 8;                            --template
          else                                           --template
            user_status <= 7;                            --template
      end if;                                          --template
when 8 =>                                        --template
          --user algorithms here                           --user code
          user_status <= 9;                                --template
        when 9 =>                                          --template
          Mem_Write_Access_Req <= '1';                     --template
          user_status <= 10;                               --template
        when 10 =>                                         --template
          if (Mem_Write_Access_Ack = '1') then             --template
            Mem_Write_Access_Req <= '0';                   --template
            user_status <= 11;                             --template
          else                                             --template
            user_status <= 10;                             --template
          end if;                                          --template
        when 11 =>                                         --template
          HW_Mem_HWAccel1 <= x"00000000";                  --user code
          HW_Offset1 <= x"00000000";                       --user code
          HW_Count1 <= x"00000100";                        --user code
          j := 0;                                          --template
          user_status <= 12;                               --template
        when 12 =>                                         --template
          data_out <= data_buffer(j);                      --template
          output_valid <= '1';                             --template
          user_status <= 13;                               --template
        when 13 =>                                         --template
          if (output_ack = '1') then                       --template
            j := j + 1;                                    --template
            if (j >= conv_integer(unsigned(HW_Count1))) then
              output_valid <= '0';                         --template
              user_status <= 14;                           --template
            else                                           --template
              data_out <= data_buffer(j);                  --template
              output_valid <= '1';                         --template
              user_status <= 13;                           --template
            end if;                                        --template
```

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
          else                                              --template
            output_valid <= '0';                            --template
            user_status <= 13;                              --template
          end if;                                           --template
        when 14 =>                                          --template
          Mem_Write_Access_Release_Req <= '1';              --template
          user_status <= 15;                                --template
        when 15 =>                                          --template
          if (Mem_Write_Access_Release_Ack = '1') then      --template
            Mem_Write_Access_Release_Req <= '0';            --template
            user_status <= 16;                              --template
          else                                              --template
            user_status <= 15;                              --template
          end if;                                           --template
        when 16 =>                                          --template
          done <= '1';                                      --template
          user_status <= 16;                                --template
        when others => null;                                --template
        end case;                                           --template
Else                                                  --template
        output_valid <= '0';                               --template
        i := 0;                                            --template
        j := 0;                                            --template
        user_status <= 0;                                  --template
        memory_read <= '0';                                --template
        Mem_Read_Access_Req <= '0';                        --template
        Mem_Write_Access_Req <= '0';                       --template
        Mem_Read_Access_Release_Req <= '0';                --template
        Mem_Write_Access_Release_Req <= '0';               --template
        done <= '1';                                       --template
      end if;                                              --template
    end if;                                                --template
  end process;                                             --template
end behav;                                                 --template
```

Figure 62 — An example template for user IP-block.

8.3.6 Some Examples of the user IP-blocks

Through the following example user IP-blocks, users can get more understanding about how to develop and prepare to integrate the hardware modules into the platform.

8.3.6.1 Data Increase Module

The function of this user hardware module is to read 16 double words from BRAM, each double word value is increased by 1, then write back to BRAM with the reverse address order (like Stack output).

Assumptions:

a. System is positive edge trigged.

b. Reset is active high.

c. System clock is fixed to 50MHz, that is 20ns for each clock cycle.

d. Set up HW_Mem_HWAccel = 0, HW_Offset = 0, HW_Count = 16 for reading data from BRAM space.

e. Set up HW_Mem_HWAccel1 = 0, HW_Offset = 0, HW_Count1 = 16 for writing data back to BRAM space.

f. In the host system, user will issue the virtual socket API function call “HW_Module_Controller(…)” with appropriate memory type setting.

The VHDL codes for this example user IP-block is listed in Figure 63.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
library ieee;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_signed.all;
use IEEE.std_logic_arith.all;

entity User_IP_Block_0 is
generic(block_width: positive:=8);
port(
    clk                         : in std_logic;
    reset                       : in std_logic;
    start                       : in std_logic;
    input_valid                 : in std_logic;
    data_in                     : in std_logic_vector(31 downto 0);
    memory_read                 : out std_logic;
        HW_Mem_HWAccel              : out std_logic_vector(31 downto 0);
    HW_Offset                   : out std_logic_vector(31 downto 0);
    HW_Count                    : out std_logic_vector(31 downto 0);
    data_out                    : out std_logic_vector(31 downto 0);
    output_valid                : out std_logic;
    HW_Mem_HWAccel1             : out std_logic_vector(31 downto 0);
    HW_Offset1                  : out std_logic_vector(31 downto 0);
    HW_Count1                   : out std_logic_vector(31 downto 0);
output_ack                  : in std_logic;
    Host_Mem_HWAccel            : in std_logic_vector(31 downto 0);
    Host_Offset                 : in std_logic_vector(31 downto 0);
Host_Count                  : in std_logic_vector(31 downto 0);
Host_Mem_HWAccel1           : in std_logic_vector(31 downto 0);
    Host_Offset1                : in std_logic_vector(31 downto 0);
    Host_Count1                 : in std_logic_vector(31 downto 0);
    Mem_Read_Access_Req         : out std_logic;
    Mem_Write_Access_Req        : out std_logic;
    Mem_Read_Access_Release_Req  : out std_logic;
    Mem_Write_Access_Release_Req : out std_logic;
    Mem_Read_Access_Ack         : in std_logic;
    Mem_Write_Access_Ack        : in std_logic;
    Mem_Read_Access_Release_Ack  : in std_logic;
    Mem_Write_Access_Release_Ack : in std_logic;
    done                        : out std_logic
     );
end User_IP_Block_0;

architecture behav of User_IP_Block_0 is

type memory32bit is array (natural range <>) of std_logic_vector(31 downto 0);
signal data_buffer      : memory32bit(0 to 255);
signal user_status      : integer range 0 to 31 := 0;

begin
  process(reset,clk)
  variable i : integer range 0 to 1024;
  variable j : integer range 0 to 1024;
  variable temp_HW_Count : std_logic_vector(31 downto 0);
  variable temp_HW_Count1 : std_logic_vector(31 downto 0);
  begin
    if (reset = '1') then
      output_valid <= '0';
      i := 0;
      j := 0;
      user_status <= 0;
      memory_read <= '0';
      Mem_Read_Access_Req <= '0';
      Mem_Write_Access_Req <= '0';
      Mem_Read_Access_Release_Req <= '0';
      Mem_Write_Access_Release_Req <= '0';
      done <= '1';
elsif rising_edge(clk) then
      if (start = '1') then
        case user_status is
        when 0 =>
          done <= '0';
          memory_read <= '0';
          output_valid <= '0';
          Mem_Read_Access_Req <= '0';
          Mem_Write_Access_Req <= '0';
          Mem_Read_Access_Release_Req <= '0';
```

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
          Mem_Write_Access_Release_Req <= '0';
       user_status <= 1;
         when 1 =>
          Mem_Read_Access_Req <= '1';
       user_status <= 2;
when 2 =>
          if (Mem_Read_Access_Ack = '1') then
            Mem_Read_Access_Req <= '0';
            user_status <= 3;
          else
            user_status <= 2;
          end if;
        when 3 =>
          HW_Mem_HWAccel <= x"00000000";
          HW_Offset <= x"00000000";
          HW_Count <= x"00000010";
          temp_HW_Count := x"00000010";
          j := 0;
          user_status <= 4;
        when 4 =>
          memory_read <= '1';
          user_status <= 5;
        when 5 =>
          memory_read <= '0';
          if (input_valid = '1') then
            data_buffer(j) <= data_in;
            j := j + 1;
            if (j >= conv_integer(unsigned(temp_HW_Count))) then
              j := 0;
              user_status <= 6;
            else
              user_status <= 5;
            end if;
          end if;
        when 6 =>
          Mem_Read_Access_Release_Req <= '1';
          user_status <= 7;
        when 7 =>
          if (Mem_Read_Access_Release_Ack = '1') then
            Mem_Read_Access_Release_Req <= '0';
            user_status <= 8;
          else
            user_status <= 7;
          end if;
        when 8 =>
          for i in 0 to 15 loop
            data_buffer(15-i) <= data_buffer(i) + 1;
          end loop
          user_status <= 9;
        when 9 =>
          Mem_Write_Access_Req <= '1';
          user_status <= 10;
        when 10 =>
          if (Mem_Write_Access_Ack = '1') then
            Mem_Write_Access_Req <= '0';
            user_status <= 11;
          else
            user_status <= 10;
          end if;
when 11 =>
          HW_Mem_HWAccel1 <= x"00000000";
          HW_Offset1 <= x"00000000";
          HW_Count1 <= x"00000010";
          temp_HW_Count1 := x"00000010";
          j := 0;
          user_status <= 12;
        when 12 =>
          data_out <= data_buffer(j);
          output_valid <= '1';
          user_status <= 13;
        when 13 =>
          if (output_ack = '1') then
            j := j + 1;
            if (j >= conv_integer(unsigned(temp_HW_Count1))) then
              output_valid <= '0';
              user_status <= 14;
```


© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
            else
              data_out <= data_buffer(j);
              output_valid <= '1';
              user_status <= 13;
            end if;
          else
            output_valid <= '0';
            user_status <= 13;
          end if;
        when 14 =>
          Mem_Write_Access_Release_Req <= '1';
          user_status <= 15;
        when 15 =>
          if (Mem_Write_Access_Release_Ack = '1') then
            Mem_Write_Access_Release_Req <= '0';
            user_status <= 16;
          else
            user_status <= 15;
          end if;
        when 16 =>
          done <= '1';
          user_status <= 16;
        when others => null;
        end case;
      else
        output_valid <= '0';
        j := 0;
        user_status <= 0;
        memory_read <= '0';
        Mem_Read_Access_Req <= '0';
        Mem_Write_Access_Req <= '0';
        Mem_Read_Access_Release_Req <= '0';
        Mem_Write_Access_Release_Req <= '0';
        done <= '1';
      end if;
    end if;
  end process;
end behav;
```

Figure 63 — An example of user IP-block.

The test-bench file for this example is listed in Figure 64. To simplify the test-bench design, we use time sequence to simulate the functionality of this module.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;

ENTITY User_IP_Block_0_tb IS
END User_IP_Block_0_tb;

ARCHITECTURE HTWTestBench OF User_IP_Block_0_tb IS

type memory32bit is array (natural range <>) of std_logic_vector(31 downto 0);

COMPONENT User_IP_Block_0
port(
    clk                         : in std_logic;
    reset                       : in std_logic;
    start                       : in std_logic;
    input_valid                 : in std_logic;
    data_in                     : in std_logic_vector(31 downto 0);
    memory_read                 : out std_logic;
    HW_Mem_HWAccel              : out std_logic_vector(31 downto 0);
    HW_Offset                   : out std_logic_vector(31 downto 0);
    HW_Count                    : out std_logic_vector(31 downto 0);
    data_out                    : out std_logic_vector(31 downto 0);
    output_valid                : out std_logic;
    HW_Mem_HWAccel1             : out std_logic_vector(31 downto 0);
    HW_Offset1                  : out std_logic_vector(31 downto 0);
    HW_Count1                   : out std_logic_vector(31 downto 0);
    output_ack                  : in std_logic;
    Host_Mem_HWAccel            : in std_logic_vector(31 downto 0);
    Host_Offset                 : in std_logic_vector(31 downto 0);
```

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    Host_Count                  : in std_logic_vector(31 downto 0);
    Host_Mem_HWAccel1           : in std_logic_vector(31 downto 0);
    Host_Offset1                : in std_logic_vector(31 downto 0);
    Host_Count1                 : in std_logic_vector(31 downto 0);
    Mem_Read_Access_Req         : out std_logic;
    Mem_Write_Access_Req        : out std_logic;
    Mem_Read_Access_Release_Req  : out std_logic;
    Mem_Write_Access_Release_Req : out std_logic;
    Mem_Read_Access_Ack         : in std_logic;
    Mem_Write_Access_Ack        : in std_logic;
    Mem_Read_Access_Release_Ack  : in std_logic;
    Mem_Write_Access_Release_Ack : in std_logic;
    done                        : out std_logic
    );
END COMPONENT;

SIGNAL clk : std_logic;
SIGNAL reset : std_logic;
SIGNAL start : std_logic;
SIGNAL input_valid : std_logic;
SIGNAL data_in : std_logic_vector(31 downto 0);
SIGNAL memory_read : std_logic;
SIGNAL HW_Mem_HWAccel : std_logic_vector(31 downto 0);
SIGNAL HW_Offset : std_logic_vector(31 downto 0);
SIGNAL HW_Count : std_logic_vector(31 downto 0);
SIGNAL data_out : std_logic_vector(31 downto 0);
SIGNAL output_valid : std_logic;
SIGNAL HW_Mem_HWAccel1 : std_logic_vector(31 downto 0);
SIGNAL HW_Offset1 : std_logic_vector(31 downto 0);
SIGNAL HW_Count1 : std_logic_vector(31 downto 0);
SIGNAL output_ack : std_logic;
SIGNAL Host_Mem_HWAccel : std_logic_vector(31 downto 0);
SIGNAL Host_Offset : std_logic_vector(31 downto 0);
SIGNAL Host_Count : std_logic_vector(31 downto 0);
SIGNAL Host_Mem_HWAccel1 : std_logic_vector(31 downto 0);
SIGNAL Host_Offset1 : std_logic_vector(31 downto 0);
SIGNAL Host_Count1 : std_logic_vector(31 downto 0);
SIGNAL Mem_Read_Access_Req : std_logic;
SIGNAL Mem_Read_Access_Ack : std_logic;
SIGNAL Mem_Read_Access_Release_Req : std_logic;
SIGNAL Mem_Read_Access_Release_Ack : std_logic;
SIGNAL Mem_Write_Access_Req : std_logic;
SIGNAL Mem_Write_Access_Ack : std_logic;
SIGNAL Mem_Write_Access_Release_Req : std_logic;
SIGNAL Mem_Write_Access_Release_Ack : std_logic;
SIGNAL done : std_logic;

BEGIN
U1 : User_IP_Block_0
PORT MAP (clk => clk,
          reset => reset,
          start => start,
        input_valid => input_valid,
          data_in => data_in,
          memory_read => memory_read,
          HW_Mem_HWAccel => HW_Mem_HWAccel,
          HW_Offset => HW_Offset,
          HW_Count => HW_Count,
          data_out => data_out,
          output_valid => output_valid,
          HW_Mem_HWAccel1 => HW_Mem_HWAccel1,
          HW_Offset1 => HW_Offset1,
          HW_Count1 => HW_Count1,
          output_ack => output_ack,
          Host_Mem_HWAccel => Host_Mem_HWAccel,
          Host_Offset => Host_Offset,
          Host_Count => Host_Count,
          Host_Mem_HWAccel1 => Host_Mem_HWAccel1,
          Host_Offset1 => Host_Offset1,
          Host_Count1 => Host_Count1,
          Mem_Read_Access_Req => Mem_Read_Access_Req,
          Mem_Read_Access_Ack => Mem_Read_Access_Ack,
          Mem_Read_Access_Release_Req => Mem_Read_Access_Release_Req,
          Mem_Read_Access_Release_Ack => Mem_Read_Access_Release_Ack,
          Mem_Write_Access_Req => Mem_Write_Access_Req,
          Mem_Write_Access_Ack => Mem_Write_Access_Ack,
```


© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
Mem_Write_Access_Release_Req => Mem_Write_Access_Release_Req,
        Mem_Write_Access_Release_Ack => Mem_Write_Access_Release_Ack,
        done => done);

user1 : process
begin
    reset <= '0'; wait for 10ns;
    reset <= '1'; wait for 20ns;
    reset <= '0'; wait;
end process user1;

user2 : process
begin
    clk <= '0'; wait for 10ns;
    clk <= '1'; wait for 10ns;
end process user2;

user3 : process
begin
    start <= '0'; wait for 50ns;
    start <= '1'; wait for 1580ns;
    start <= '0'; wait;
end process user3;

user4 : process
begin
    Mem_Read_Access_Ack <= '0'; wait for 90ns;
    Mem_Read_Access_Ack <= '1'; wait for 20ns;
    Mem_Read_Access_Ack <= '0'; wait;
end process user4;

user5 : process
begin
    Mem_Read_Access_Release_Ack <= '0'; wait for 810ns;
    Mem_Read_Access_Release_Ack <= '1'; wait for 20ns;
    Mem_Read_Access_Release_Ack <= '0'; wait;
end process user5;

user6 : process
begin
    Mem_Write_Access_Ack <= '0'; wait for 870ns;
    Mem_Write_Access_Ack <= '1'; wait for 20ns;
    Mem_Write_Access_Ack <= '0'; wait;
end process user6;

user7 : process
begin
    Mem_Write_Access_Release_Ack <= '0'; wait for 1590ns;
    Mem_Write_Access_Release_Ack <= '1'; wait for 20ns;
    Mem_Write_Access_Release_Ack <= '0'; wait;
end process user7;
user8 : process
variable i : integer range 0 to 15;
begin
    input_valid <= '0'; wait for 150ns;
    for i in 0 to 15 loop
    input_valid <= '0'; wait for 20ns;
    input_valid <= '1'; wait for 20ns;
    end loop;
    input_valid <= '0'; wait;
end process user8;

user9 : process
variable i : integer range 0 to 15;
variable data_array : memory32bit(0 to 15) :=
                      (x"01234567",
                       x"ff2098a4",
                       x"8877efab",
                       x"45ab2100",
                       x"7fff2000",
                       x"0134ddee",
                       x"11112222",
                       x"b4a40912",
                       x"5d7e3623",
                       x"8810aa32",
                       x"eeffdd22",
```

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
                        x"001033ba",
                        x"0fa43e22",
                        x"11001100",
                        x"bbbbaaaa",
                        x"45671001");
begin
    data_in <= x"00000000"; wait for 170ns;
    for i in 0 to 15 loop
    data_in <= data_array(i); wait for 40ns;
    end loop;
    data_in <= x"00000000"; wait;
end process user9;

user10 : process
variable i : integer range 0 to 15;
begin
    output_ack <= '0'; wait for 930ns;
    for i in 0 to 15 loop
    output_ack <= '0'; wait for 20ns;
    output_ack <= '1'; wait for 20ns;
    end loop;
    output_ack <= '0'; wait;
end process user10;

END HTWTestBench;
```

Figure 64 — Testbench for simulation.

According to the functional simulation, if data input to the hardware module are

```
(x"01234567", x"ff2098a4", x"8877efab", x"45ab2100", x"7fff2000",x"0134ddee", x"11112222",
x"b4a40912", x"5d7e3623", x"8810aa32", x"eeffdd22", x"001033ba", x"0fa43e22", x"11001100",
x"bbbbaaaa",x"45671001");
```

The output data back to the memory space are

```
(x"45671002", x"bbbbaaab", x"11001101", x"0fa43e23", x"001033bb", x"eeffdd23", x"8810aa33",
x"5d7e3624", x"b4a40913", x"11112223", x"0134ddef", x"7fff2001", x"45ab2101", x"8877efac",
x"ff2098a5", x"01234568");
```

The results conform to the functionality of the data increase module defined above.

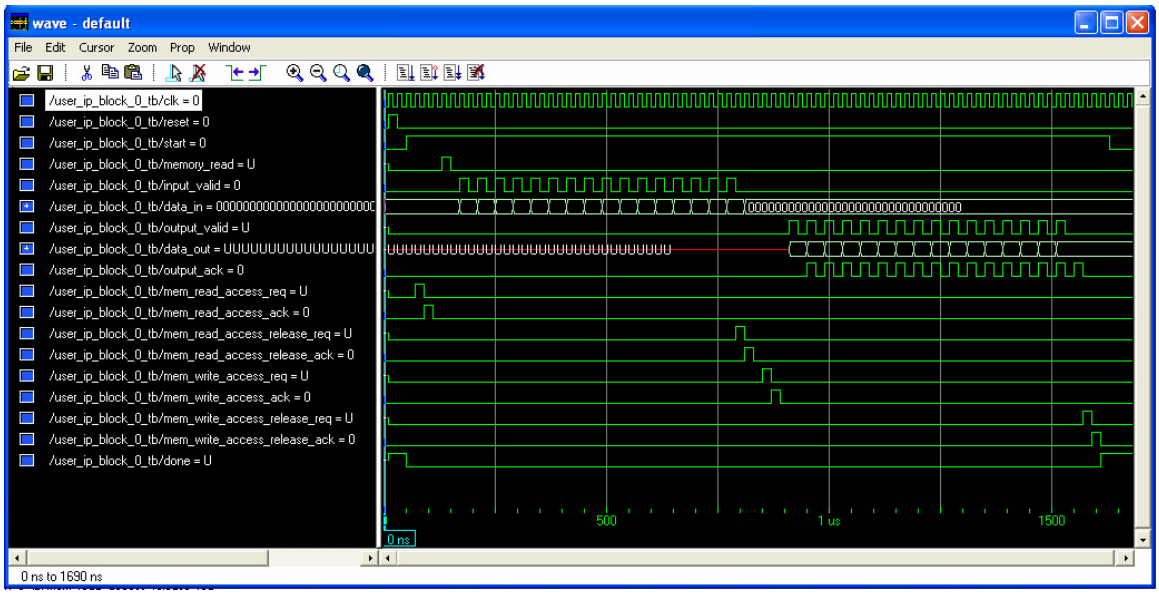


Figure 65 — Simulation of the test-bench for data increase module.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2009 – All rights reserved
Not for Resale

8.3.6.2 Arithmetic Module

The function of this user hardware module is to read 16 double words from SRAM. The higher 16 bits of each double word is operand 1, and the lower 16 bits of each double word is operand 2. We will add operand 1 and operand 2 to generate a new double word called ADD double word. Besides, we will multiply operand 1 by operand 2 to generate another new double word called MUL double word. After the arithmetic processing finished, we write the newly generated double words back to SRAM. Because each original double word generates 2 new double words (ADD and MUL), there are total 32 new double words will be written back to SRAM and overwrite the original data.

Assumptions:

a. System is positive edge trigged.

b. Reset is active high.

c. System clock is fixed to 50MHz, that is 20ns for each clock cycle.

d. Set up HW_Mem_HWAccel = 1, HW_Offset = 0, HW_Count = 16 for reading data from SRAM space.

e. Set up HW_Mem_HWAccel1 = 1, HW_Offset = 0, HW_Count1 = 32 for writing data back to SRAM space.

f. In the host system, user will issue the virtual socket API function call “HW_Module_Controller(…)” with appropriate memory type setting.

The VHDL codes for this example user IP-block is listed in Figure 66.

```
library ieee;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_signed.all;
use IEEE.std_logic_arith.all;

entity User_IP_Block_1 is
generic(block_width: positive:=8);
port(
    clk                          : in std_logic;
    reset                        : in std_logic;
    start                        : in std_logic;
input_valid                 : in std_logic;
data_in                     : in std_logic_vector(31 downto 0);
    memory_read                  : out std_logic;
        HW_Mem_HWAccel               : out std_logic_vector(31 downto 0);
    HW_Offset                    : out std_logic_vector(31 downto 0);
    HW_Count                     : out std_logic_vector(31 downto 0);
    data_out                     : out std_logic_vector(31 downto 0);
output_valid                : out std_logic;
    HW_Mem_HWAccel1              : out std_logic_vector(31 downto 0);
    HW_Offset1                   : out std_logic_vector(31 downto 0);
    HW_Count1                    : out std_logic_vector(31 downto 0);
output_ack                  : in std_logic;
    Host_Mem_HWAccel             : in std_logic_vector(31 downto 0);
    Host_Offset                  : in std_logic_vector(31 downto 0);
Host_Count                  : in std_logic_vector(31 downto 0);
        Host_Mem_HWAccel1            : in std_logic_vector(31 downto 0);
    Host_Offset1                 : in std_logic_vector(31 downto 0);
    Host_Count1                  : in std_logic_vector(31 downto 0);
    Mem_Read_Access_Req          : out std_logic;
    Mem_Write_Access_Req         : out std_logic;
    Mem_Read_Access_Release_Req  : out std_logic;
    Mem_Write_Access_Release_Req : out std_logic;
    Mem_Read_Access_Ack          : in std_logic;
    Mem_Write_Access_Ack         : in std_logic;
    Mem_Read_Access_Release_Ack  : in std_logic;
    Mem_Write_Access_Release_Ack : in std_logic;
    done                         : out std_logic
    );
end User_IP_Block_1;
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
architecture behav of User_IP_Block_1 is

type memory32bit is array (natural range <>) of std_logic_vector(31 downto 0);
signal data_buffer      : memory32bit(0 to 255);
signal user_status      : integer range 0 to 31 := 0;

begin
  process(reset,clk)
  variable i : integer range 0 to 1024;
  variable j : integer range 0 to 1024;
  variable arithmetic_buffer: memory32bit(0 to 31);
  variable temp_data0,temp_data1 : std_logic_vector(31 downto 0);
  variable temp_HW_Count : std_logic_vector(31 downto 0);
  variable temp_HW_Count1 : std_logic_vector(31 downto 0);
  begin
    if (reset = '1') then
      output_valid <= '0';
      i := 0;
      j := 0;
      user_status <= 0;
      memory_read <= '0';
      Mem_Read_Access_Req <= '0';
      Mem_Write_Access_Req <= '0';
      Mem_Read_Access_Release_Req <= '0';
      Mem_Write_Access_Release_Req <= '0';
      done <= '1';
elsif rising_edge(clk) then
  if (start = '1') then
        case user_status is
        when 0 =>
          done <= '0';
          memory_read <= '0';
          output_valid <= '0';
          Mem_Read_Access_Req <= '0';
          Mem_Write_Access_Req <= '0';
          Mem_Read_Access_Release_Req <= '0';
          Mem_Write_Access_Release_Req <= '0';
      user_status <= 1;
when 1 =>
          Mem_Read_Access_Req <= '1';
          user_status <= 2;
        when 2 =>
          if (Mem_Read_Access_Ack = '1') then
            Mem_Read_Access_Req <= '0';
            user_status <= 3;
          else
            user_status <= 2;
          end if;
        when 3 =>
          HW_Mem_HWAccel <= x"00000000";
          HW_Offset <= x"00000000";
          HW_Count <= x"00000010";
          temp_HW_Count := x"00000010";
          j := 0;
          user_status <= 4;
        when 4 =>
          memory_read <= '1';
          user_status <= 5;
        when 5 =>
          memory_read <= '0';
          if (input_valid = '1') then
            data_buffer(j) <= data_in;
            j := j + 1;
            if (j >= conv_integer(unsigned(temp_HW_Count))) then
              j := 0;
              user_status <= 6;
            else
              user_status <= 5;
            end if;
          end if;
        when 6 =>
          Mem_Read_Access_Release_Req <= '1';
          user_status <= 7;
        when 7 =>
          if (Mem_Read_Access_Release_Ack = '1') then
```


--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2009 – All rights reserved
Not for Resale

```
            Mem_Read_Access_Release_Req <= '0';
            user_status <= 8;
          else
            user_status <= 7;
          end if;
        when 8 =>
          for i in 0 to 15 loop
            temp_data0 := data_buffer(i);
            temp_data1(15 downto 0) := temp_data0(31 downto 16);
            temp_data0(31 downto 16) := (others => '0');
            temp_data1(31 downto 16) := (others => '0');
            arithmetic_buffer(2*i) := temp_data0 + temp_data1;
            arithmetic_buffer(2*i+1):=unsigned(temp_data0(15 downto
                          0))*unsigned(temp_data1(15 downto 0));
          end loop;
          user_status <= 9;
when 9 =>
          for i in 0 to 31 loop
            data_buffer(i) <= arithmetic_buffer(i);
          end loop;
          user_status <= 10;
        when 10 =>
          Mem_Write_Access_Req <= '1';
          user_status <= 11;
        when 11 =>
          if (Mem_Write_Access_Ack = '1') then
            Mem_Write_Access_Req <= '0';
            user_status <= 12;
          else
            user_status <= 11;
          end if;
        when 12 =>
          HW_Mem_HWAccel1 <= x"00000000";
          HW_Offset1 <= x"00000000";
          HW_Count1 <= x"00000020";
          temp_HW_Count1 := x"00000020";
          j := 0;
          user_status <= 13;
        when 13 =>
          data_out <= data_buffer(j);
          output_valid <= '1';
          user_status <= 14;
        when 14 =>
          if (output_ack = '1') then
            j := j + 1;
            if (j >= conv_integer(unsigned(temp_HW_Count1))) then
              output_valid <= '0';
              user_status <= 15;
            else
              data_out <= data_buffer(j);
              output_valid <= '1';
              user_status <= 14;
            end if;
          else
            output_valid <= '0';
            user_status <= 14;
          end if;
        when 15 =>
          Mem_Write_Access_Release_Req <= '1';
          user_status <= 16;
        when 16 =>
          if (Mem_Write_Access_Release_Ack = '1') then
            Mem_Write_Access_Release_Req <= '0';
            user_status <= 17;
          else
            user_status <= 16;
          end if;
        when 17 =>
          done <= '1';
          user_status <= 17;
        when others => null;
        end case;
             else
        output_valid <= '0';
        i := 0;
        j := 0;
```

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
        user_status <= 0;
        memory_read <= '0';
        Mem_Read_Access_Req <= '0';
        Mem_Write_Access_Req <= '0';
        Mem_Read_Access_Release_Req <= '0';
        Mem_Write_Access_Release_Req <= '0';
        done <= '1';
      end if;

    end if;
  end process;
end behav;
```

Figure 66 — Another example of user IP-block.

The test-bench file for this example is listed in Figure 67.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;

ENTITY User_IP_Block_1_tb IS
END User_IP_Block_1_tb;

ARCHITECTURE HTWTestBench OF User_IP_Block_1_tb IS

type memory32bit is array (natural range <>) of std_logic_vector(31 downto 0);

COMPONENT User_IP_Block_1
port(
    clk                         : in std_logic;
    reset                       : in std_logic;
    start                       : in std_logic;
    input_valid                 : in std_logic;
    data_in                     : in std_logic_vector(31 downto 0);
    memory_read                 : out std_logic;
    HW_Mem_HWAccel              : out std_logic_vector(31 downto 0);
    HW_Offset                   : out std_logic_vector(31 downto 0);
    HW_Count                    : out std_logic_vector(31 downto 0);
    data_out                    : out std_logic_vector(31 downto 0);
output_valid                : out std_logic;
    HW_Mem_HWAccel1             : out std_logic_vector(31 downto 0);
    HW_Offset1                  : out std_logic_vector(31 downto 0);
    HW_Count1                   : out std_logic_vector(31 downto 0);
output_ack                  : in std_logic;
    Host_Mem_HWAccel            : in std_logic_vector(31 downto 0);
Host_Offset                 : in std_logic_vector(31 downto 0);
Host_Count                  : in std_logic_vector(31 downto 0);
    Host_Mem_HWAccel1           : in std_logic_vector(31 downto 0);
    Host_Offset1                : in std_logic_vector(31 downto 0);
    Host_Count1                 : in std_logic_vector(31 downto 0);
    Mem_Read_Access_Req         : out std_logic;
    Mem_Write_Access_Req        : out std_logic;
    Mem_Read_Access_Release_Req  : out std_logic;
    Mem_Write_Access_Release_Req : out std_logic;
    Mem_Read_Access_Ack         : in std_logic;
    Mem_Write_Access_Ack        : in std_logic;
    Mem_Read_Access_Release_Ack  : in std_logic;
    Mem_Write_Access_Release_Ack : in std_logic;
    done                        : out std_logic
    );
END COMPONENT;

SIGNAL clk : std_logic;
SIGNAL reset : std_logic;
SIGNAL start : std_logic;
SIGNAL input_valid : std_logic;
SIGNAL data_in : std_logic_vector(31 downto 0);
SIGNAL memory_read : std_logic;
SIGNAL HW_Mem_HWAccel : std_logic_vector(31 downto 0);
SIGNAL HW_Offset : std_logic_vector(31 downto 0);
SIGNAL HW_Count : std_logic_vector(31 downto 0);
SIGNAL data_out : std_logic_vector(31 downto 0);
SIGNAL output_valid : std_logic;
SIGNAL HW_Mem_HWAccel1 : std_logic_vector(31 downto 0);
```


© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
SIGNAL HW_Offset1 : std_logic_vector(31 downto 0);
SIGNAL HW_Count1 : std_logic_vector(31 downto 0);
SIGNAL output_ack : std_logic;
SIGNAL Host_Mem_HWAccel : std_logic_vector(31 downto 0);
SIGNAL Host_Offset : std_logic_vector(31 downto 0);
SIGNAL Host_Count : std_logic_vector(31 downto 0);
SIGNAL Host_Mem_HWAccel1 : std_logic_vector(31 downto 0);
SIGNAL Host_Offset1 : std_logic_vector(31 downto 0);
SIGNAL Host_Count1 : std_logic_vector(31 downto 0);
SIGNAL Mem_Read_Access_Req : std_logic;
SIGNAL Mem_Read_Access_Ack : std_logic;
SIGNAL Mem_Read_Access_Release_Req : std_logic;
SIGNAL Mem_Read_Access_Release_Ack : std_logic;
SIGNAL Mem_Write_Access_Req : std_logic;
SIGNAL Mem_Write_Access_Ack : std_logic;
SIGNAL Mem_Write_Access_Release_Req : std_logic;
SIGNAL Mem_Write_Access_Release_Ack : std_logic;
SIGNAL done : std_logic;

BEGIN
U1 : User_IP_Block_1
PORT MAP (clk => clk,
  reset => reset,
  start => start,
  input_valid => input_valid,
  data_in => data_in,
  memory_read => memory_read,
  HW_Mem_HWAccel => HW_Mem_HWAccel,
HW_Offset => HW_Offset,
  HW_Count => HW_Count,
  data_out => data_out,
  output_valid => output_valid,
  HW_Mem_HWAccel1 => HW_Mem_HWAccel1,
  HW_Offset1 => HW_Offset1,
  HW_Count1 => HW_Count1,
  output_ack => output_ack,
  Host_Mem_HWAccel => Host_Mem_HWAccel,
  Host_Offset => Host_Offset,
  Host_Count => Host_Count,
  Host_Mem_HWAccel1 => Host_Mem_HWAccel1,
  Host_Offset1 => Host_Offset1,
  Host_Count1 => Host_Count1,
  Mem_Read_Access_Req => Mem_Read_Access_Req,
  Mem_Read_Access_Ack => Mem_Read_Access_Ack,
  Mem_Read_Access_Release_Req => Mem_Read_Access_Release_Req,
  Mem_Read_Access_Release_Ack => Mem_Read_Access_Release_Ack,
  Mem_Write_Access_Req => Mem_Write_Access_Req,
  Mem_Write_Access_Ack => Mem_Write_Access_Ack,
  Mem_Write_Access_Release_Req => Mem_Write_Access_Release_Req,
  Mem_Write_Access_Release_Ack => Mem_Write_Access_Release_Ack,
  done => done);

user1 : process
begin
    reset <= '0'; wait for 10ns;
    reset <= '1'; wait for 20ns;
    reset <= '0'; wait;
end process user1;

user2 : process
begin
    clk <= '0'; wait for 10ns;
    clk <= '1'; wait for 10ns;
end process user2;

user3 : process
begin
    start <= '0'; wait for 50ns;
    start <= '1'; wait for 2260ns;
    start <= '0'; wait;
end process user3;

user4 : process
begin
    Mem_Read_Access_Ack <= '0'; wait for 90ns;
    Mem_Read_Access_Ack <= '1'; wait for 20ns;
```

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    Mem_Read_Access_Ack <= '0'; wait;
end process user4;
user5 : process
begin
    Mem_Read_Access_Release_Ack <= '0'; wait for 810ns;
    Mem_Read_Access_Release_Ack <= '1'; wait for 20ns;
    Mem_Read_Access_Release_Ack <= '0'; wait;
end process user5;

 user6 : process
begin
    Mem_Write_Access_Ack <= '0'; wait for 890ns;
    Mem_Write_Access_Ack <= '1'; wait for 20ns;
    Mem_Write_Access_Ack <= '0'; wait;
end process user6;

user7 : process
begin
    Mem_Write_Access_Release_Ack <= '0'; wait for 2250ns;
    Mem_Write_Access_Release_Ack <= '1'; wait for 20ns;
    Mem_Write_Access_Release_Ack <= '0'; wait;
end process user7;

user8 : process
variable i : integer range 0 to 15;
begin
    input_valid <= '0'; wait for 150ns;
    for i in 0 to 15 loop
    input_valid <= '0'; wait for 20ns;
    input_valid <= '1'; wait for 20ns;
    end loop;
    input_valid <= '0'; wait;
end process user8;

user9 : process
variable i : integer range 0 to 15;
variable data_array : memory32bit(0 to 15) :=
                      (x"76abc109",
                       x"101021ab",
                       x"07cb0ff4",
                       x"21001200",
                       x"4311abff",
                       x"90bbaa01",
                       x"facb1023",
                       x"0912baed",
                       x"7812ffff",
                       x"01016abf",
                       x"eeddaa00",
                       x"0990abcd",
                       x"22110012",
                       x"7812bbaa",
                       x"99998888",
                       x"11112222");
begin
    data_in <= x"00000000"; wait for 170ns;
    for i in 0 to 15 loop
    data_in <= data_array(i); wait for 40ns;
    end loop;
    data_in <= x"00000000"; wait;
end process user9;

user10 : process
variable i : integer range 0 to 15;
begin
    output_ack <= '0'; wait for 950ns;
    for i in 0 to 31 loop
    output_ack <= '0'; wait for 20ns;
    output_ack <= '1'; wait for 20ns;
    end loop;
    output_ack <= '0'; wait;
end process user10;

END HTWTestBench;
```

Figure 67 — Test-bench for simulation.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

According to the functional simulation, if data input to the hardware module are

```
(x"76abc109",  x"101021ab",  x"07cb0ff4",  x"21001200",  x"4311abff",  x"90bbaa01",  x"facb1023",
x"0912baed",  x"7812ffff",  x"01016abf",  x"eeddaa00",  x"0990abcd",  x"22110012",  x"7812bbaa",
x"99998888",x"11112222");
```

The output data back to the memory space are

```
(x"000137b4",  x"597b1703",  x"000031bb",  x"021ccab0",  x"000017bf",  x"007c527c",  x"00003300",
x"02520000",  x"0000ef10",  x"2d0f28ef",  x"00013abc",  x"601cbebb",  x"00010aee",  x"0fcef9c1",
x"0000c3ff",  x"069f79aa",  x"00017811",  x"781187ee",  x"00006bc0",  x"006b29bf",  x"000198dd",
x"9e9ec200",  x"0000b55d",  x"066ad850",  x"00002223",  x"00026532",  x"000133bc",  x"5804e1f4",
x"00012221", x"51eae148", x"00003333",  x"02468642");
```

The results conform to the functionality of the data increase module defined above.

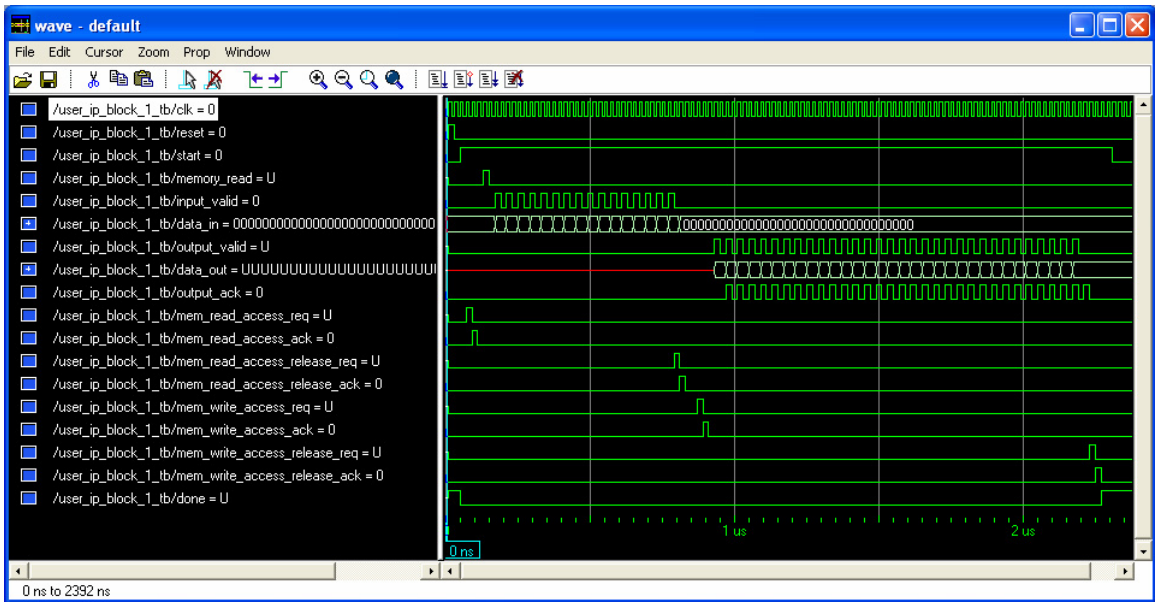


Figure 68 — Simulation of the test-bench for arithmetic module.

8.3.6.3 5.3 An Advanced Module

So far we have two simple examples that can be integrated into the platform. In some other case, however, users are probably interested in how to integrate a more complicated hardware module into the platform. The following example will show how to integrate a practical AVC DCT and Quantization module into the platform. This DCT module is developed by the University of Calgary [25]. The architecture of the DCT module allows the reading and writing operations to be performed separately. Therefore, in this example, a parallel structure will be used for memory access to improve the system performance. It is assumed that DCT module will read the data from SRAM and write back to DRAM. Suppose “HW_Module_Controller(…)” is issued with appropriate memory type setting.

```
library ieee;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_signed.all;
use IEEE.std_logic_arith.all;

entity User_IP_Block_0 is
generic(block_width: positive:=8);
```

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
port(
   clk                          : in std_logic;
   reset                        : in std_logic;
   start                        : in std_logic;
   input_valid                  : in std_logic;
   data_in                      : in std_logic_vector(31 downto 0);
   memory_read                  : out std_logic;
   HW_Mem_HWAccel               : out std_logic_vector(31 downto 0);
   HW_Offset                    : out std_logic_vector(31 downto 0);
   HW_Count                     : out std_logic_vector(31 downto 0);
   data_out                     : out std_logic_vector(31 downto 0);
   output_valid                 : out std_logic;
HW_Mem_HWAccel1              : out std_logic_vector(31 downto 0);
   HW_Offset1                   : out std_logic_vector(31 downto 0);
   HW_Count1                    : out std_logic_vector(31 downto 0);
   output_ack                   : in std_logic;
   Host_Mem_HWAccel             : in std_logic_vector(31 downto 0);
   Host_Offset                  : in std_logic_vector(31 downto 0);
   Host_Count                   : in std_logic_vector(31 downto 0);
   Host_Mem_HWAccel1            : in std_logic_vector(31 downto 0);
   Host_Offset1                 : in std_logic_vector(31 downto 0);
   Host_Count1                  : in std_logic_vector(31 downto 0);
   Mem_Read_Access_Req          : out std_logic;
   Mem_Write_Access_Req         : out std_logic;
   Mem_Read_Access_Release_Req  : out std_logic;
   Mem_Write_Access_Release_Req : out std_logic;
   Mem_Read_Access_Ack          : in std_logic;
   Mem_Write_Access_Ack         : in std_logic;
   Mem_Read_Access_Release_Ack  : in std_logic;
   Mem_Write_Access_Release_Ack : in std_logic;
   done                         : out std_logic
     );
end User_IP_Block_0;

architecture behav of User_IP_Block_0 is

type memory32bit is array (natural range <>) of std_logic_vector(31 downto 0);
signal input_data_buffer      : memory32bit(0 to 255);
signal output_data_buffer     : memory32bit(0 to 255);
signal user_input_status      : integer range 0 to 31;
signal user_output_status     : integer range 0 to 31;

signal DCT_input_valid        : std_logic;
signal DCT_output_valid       : std_logic;
signal DCT_X00_X03            : signed(31 downto 0);
signal DCT_X10_X13            : signed(31 downto 0);
signal DCT_X20_X23            : signed(31 downto 0);
signal DCT_X30_X33            : signed(31 downto 0);
signal DCT_Z00_Z01            : signed(31 downto 0);
signal DCT_Z02_Z03            : signed(31 downto 0);
signal DCT_Z10_Z11            : signed(31 downto 0);
signal DCT_Z12_Z13            : signed(31 downto 0);
signal DCT_Z20_Z21            : signed(31 downto 0);
signal DCT_Z22_Z23            : signed(31 downto 0);
signal DCT_Z30_Z31            : signed(31 downto 0);
signal DCT_Z32_Z33            : signed(31 downto 0);

component CoreTrans
  port(
        CLK          : IN     std_logic;
        QP           : IN     signed (5 DOWNTO 0);
        X00          : IN     signed (7 DOWNTO 0);
        X01          : IN     signed (7 DOWNTO 0);
        X02          : IN     signed (7 DOWNTO 0);
        X03          : IN     signed (7 DOWNTO 0);
        X10          : IN     signed (7 DOWNTO 0);
        X11          : IN     signed (7 DOWNTO 0);
        X12          : IN     signed (7 DOWNTO 0);
        X13          : IN     signed (7 DOWNTO 0);
        X20          : IN     signed (7 DOWNTO 0);
        X21          : IN     signed (7 DOWNTO 0);
        X22          : IN     signed (7 DOWNTO 0);
        X23          : IN     signed (7 DOWNTO 0);
        X30          : IN     signed (7 DOWNTO 0);
        X31          : IN     signed (7 DOWNTO 0);
```


© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
        X32          : IN     signed (7 DOWNTO 0);
        X33          : IN     signed (7 DOWNTO 0);
        input_valid  : IN     std_logic;
        Z00          : OUT    signed (13 DOWNTO 0);
        Z01          : OUT    signed (13 DOWNTO 0);
        Z02          : OUT    signed (13 DOWNTO 0);
        Z03          : OUT    signed (13 DOWNTO 0);
        Z10          : OUT    signed (13 DOWNTO 0);
        Z11          : OUT    signed (13 DOWNTO 0);
        Z12          : OUT    signed (13 DOWNTO 0);
        Z13          : OUT    signed (13 DOWNTO 0);
        Z20          : OUT    signed (13 DOWNTO 0);
        Z21          : OUT    signed (13 DOWNTO 0);
        Z22          : OUT    signed (13 DOWNTO 0);
        Z23          : OUT    signed (13 DOWNTO 0);
        Z30          : OUT    signed (13 DOWNTO 0);
        Z31          : OUT    signed (13 DOWNTO 0);
        Z32          : OUT    signed (13 DOWNTO 0);
        Z33          : OUT    signed (13 DOWNTO 0);
        output_valid : OUT    std_logic
      );
  end component;

begin

  u_myhardware_dct: CoreTrans
  port map(
            CLK          => clk,
            QP           => "000101",
            X00          => DCT_X00_X03(7 downto 0),
            X01          => DCT_X00_X03(15 downto 8),
            X02          => DCT_X00_X03(23 downto 16),
            X03          => DCT_X00_X03(31 downto 24),
            X10          => DCT_X10_X13(7 downto 0),
            X11          => DCT_X10_X13(15 downto 8),
            X12          => DCT_X10_X13(23 downto 16),
            X13          => DCT_X10_X13(31 downto 24),
            X20          => DCT_X20_X23(7 downto 0),
            X21          => DCT_X20_X23(15 downto 8),
            X22          => DCT_X20_X23(23 downto 16),
            X23          => DCT_X20_X23(31 downto 24),
            X30          => DCT_X30_X33(7 downto 0),
            X31          => DCT_X30_X33(15 downto 8),
            X32          => DCT_X30_X33(23 downto 16),
            X33          => DCT_X30_X33(31 downto 24),
            input_valid  => DCT_input_valid,
            Z00          => DCT_Z00_Z01(13 downto 0),
            Z01          => DCT_Z00_Z01(29 downto 16),
            Z02          => DCT_Z02_Z03(13 downto 0),
            Z03          => DCT_Z02_Z03(29 downto 16),
            Z10          => DCT_Z10_Z11(13 downto 0),
            Z11          => DCT_Z10_Z11(29 downto 16),
            Z12          => DCT_Z12_Z13(13 downto 0),
            Z13          => DCT_Z12_Z13(29 downto 16),
            Z20          => DCT_Z20_Z21(13 downto 0),
            Z21          => DCT_Z20_Z21(29 downto 16),
            Z22          => DCT_Z22_Z23(13 downto 0),
            Z23          => DCT_Z22_Z23(29 downto 16),
            Z30          => DCT_Z30_Z31(13 downto 0),
            Z31          => DCT_Z30_Z31(29 downto 16),
            Z32          => DCT_Z32_Z33(13 downto 0),
            Z33          => DCT_Z32_Z33(29 downto 16),
            output_valid => DCT_output_valid
         );

process(reset,clk)
  variable i : integer range 0 to 1024;
  variable j : integer range 0 to 1024;
  variable DCT_Block : integer range 0 to 32767;
  variable temp0,temp1,temp2,temp3 : std_logic_vector(31 downto 0);
  begin
    if (reset = '1') then
      i := 0;
      j := 0;
      DCT_Block := 0;
      user_input_status <= 0;
```

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      memory_read <= '0';
      DCT_input_valid <= '0';
      Mem_Read_Access_Req <= '0';
      Mem_Read_Access_Release_Req <= '0';
elsif rising_edge(clk) then

      if (start = '1') then
        case user_input_status is
        when 0 =>
          memory_read <= '0';
          Mem_Read_Access_Req <= '0';
          Mem_Read_Access_Release_Req <= '0';
          user_input_status <= 1;
        when 1 =>
          Mem_Read_Access_Req <= '1';
          user_input_status <= 2;
        when 2 =>
          if (Mem_Read_Access_Ack = '1') then
            Mem_Read_Access_Req <= '0';
            user_input_status <= 3;
          else
            user_input_status <= 2;
      end if;
when 3 =>
          HW_Mem_HWAccel <= x"00000000";
          HW_Offset <= x"00000000";
          HW_Count <= x"00000004";
          DCT_Block := 6336;
          j := 0;
          user_input_status <= 4;
        when 4 =>
          memory_read <= '1';
          user_input_status <= 5;
        when 5 =>
          memory_read <= '0';
          if (input_valid = '1') then
            input_data_buffer(j) <= data_in;
            HW_Offset <= HW_Offset + 1;
            j := j + 1;
            if (j >= conv_integer(unsigned(HW_Count))) then
              j := 0;
              temp0 := input_data_buffer(0);
              temp1 := input_data_buffer(1);
              temp2 := input_data_buffer(2);
              temp3 := data_in;
              DCT_X00_X03(7 downto 0) <= signed(temp0(7 downto 0));
              DCT_X00_X03(15 downto 8) <= signed(temp0(15 downto 8));
              DCT_X00_X03(23 downto 16) <= signed(temp0(23 downto
                                                        16));
              DCT_X00_X03(31 downto 24) <= signed(temp0(31 downto
                                                        24));
              DCT_X10_X13(7 downto 0) <= signed(temp1(7 downto 0));
              DCT_X10_X13(15 downto 8) <= signed(temp1(15 downto 8));
              DCT_X10_X13(23 downto 16) <= signed(temp1(23 downto
                                                        16));
              DCT_X10_X13(31 downto 24) <= signed(temp1(31 downto
                                                        24));
              DCT_X20_X23(7 downto 0) <= signed(temp2(7 downto 0));
              DCT_X20_X23(15 downto 8) <= signed(temp2(15 downto 8));
              DCT_X20_X23(23 downto 16) <= signed(temp2(23 downto
                                                        16));
              DCT_X20_X23(31 downto 24) <= signed(temp2(31 downto
                                                        24));
              DCT_X30_X33(7 downto 0) <= signed(temp3(7 downto 0));
              DCT_X30_X33(15 downto 8) <= signed(temp3(15 downto 8));
              DCT_X30_X33(23 downto 16) <= signed(temp3(23 downto
                                                        16));
              DCT_X30_X33(31 downto 24) <= signed(temp3(31 downto
                                                        24));
              DCT_input_valid <= '1';
              user_input_status <= 6;
            else
              user_input_status <= 5;
            end if;
          else
            user_input_status <= 5;
```


--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2009 – All rights reserved
Not for Resale

```
          end if;
when 6 =>
          DCT_input_valid <= '0';
          DCT_Block := DCT_Block - 1;
          if (DCT_Block = 0) then
            j := 0;
            user_input_status <= 7;
          else
            j := 0;
            user_input_status <= 4;
          end if;
        when 7 =>
          Mem_Read_Access_Release_Req <= '1';
          user_input_status <= 8;
        when 8 =>
          if (Mem_Read_Access_Release_Ack = '1') then
            Mem_Read_Access_Release_Req <= '0';
            user_input_status <= 9;
          else
            user_input_status <= 8;
          end if;
        when 9 =>
          user_input_status <= 9;
        when others => null;
        end case;
      else
        i := 0;
        j := 0;
        user_input_status <= 0;
        memory_read <= '0';
        DCT_input_valid <= '0';
        Mem_Read_Access_Req <= '0';
        Mem_Read_Access_Release_Req <= '0';
      end if;
    end if;
  end process;

process(reset,clk)
  variable i : integer range 0 to 1024;
  variable j : integer range 0 to 1024;
  variable DCT_Block : integer range 0 to 32767;
  begin
    if (reset = '1') then
      output_valid <= '0';
      i := 0;
      j := 0;
      user_output_status <= 0;
      Mem_Write_Access_Req <= '0';
      Mem_Write_Access_Release_Req <= '0';
      done <= '1';
elsif rising_edge(clk) then

      if (start = '1') then
        case user_output_status is
        when 0 =>
          Mem_Write_Access_Req <= '1';
      user_output_status <= 1;
when 1 =>
          if (Mem_Write_Access_Ack = '1') then
            Mem_Write_Access_Req <= '0';
            user_output_status <= 2;
          else
            user_output_status <= 1;
          end if;
        when 2 =>
          if (DCT_output_valid = '1') then
            done <= '0';
            j := 0;
            output_data_buffer(0) <=
                              conv_std_logic_vector(DCT_Z00_Z01,32);
            output_data_buffer(1) <=
                              conv_std_logic_vector(DCT_Z02_Z03,32);
            output_data_buffer(2) <=
                              conv_std_logic_vector(DCT_Z10_Z11,32);
            output_data_buffer(3) <=
                              conv_std_logic_vector(DCT_Z12_Z13,32);
```

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
          output_data_buffer(4) <=
                              conv_std_logic_vector(DCT_Z20_Z21,32);
          output_data_buffer(5) <=
                              conv_std_logic_vector(DCT_Z22_Z23,32);
          output_data_buffer(6) <=
                              conv_std_logic_vector(DCT_Z30_Z31,32);
          output_data_buffer(7) <=
                              conv_std_logic_vector(DCT_Z32_Z33,32);
          user_output_status <= 3;
        end if;
      when 3 =>
        data_out <= output_data_buffer(j);
        output_valid <= '1';
        user_output_status <= 4;
      when 4 =>
        j := j + 1;
        HW_Offset1 <= HW_Offset1 + 1;
        if (j >= conv_integer(unsigned(HW_Count1))) then
          output_valid <= '0';
          j := 0;
          i := 0;
          DCT_Block := DCT_Block - 1;
          if (DCT_Block = 0) then
            user_output_status <= 5;
          else
            user_output_status <= 2;
          end if;
        else
          data_out <= output_data_buffer(j);
          output_valid <= '1';
          user_output_status <= 4;
        end if;
      when 5 =>
        Mem_Write_Access_Release_Req <= '1';
        user_output_status <= 6;
      when 6 =>
        if (Mem_Write_Access_Release_Ack = '1') then
          Mem_Write_Access_Release_Req <= '0';
          user_output_status <= 7;
        else
          user_output_status <= 6;
        end if;
      when 7 =>
        done <= '1';
        user_output_status <= 7;
      when others => null;
      end case;
    else
      DCT_Block := 6336;
      output_valid <= '0';
      i := 0;
      j := 0;
      user_output_status <= 0;
      Mem_Write_Access_Req <= '0';
      Mem_Write_Access_Release_Req <= '0';
      done <= '1';
      HW_Mem_HWAccel1 <= x"00000000";
      HW_Offset1 <= x"00000000";
      HW_Count1 <= x"00000008";
    end if;
  end if;
end process;

end behav;
```

Figure 69 — Integration of the DCT module.

8.3.7 A guide for integrating modules within the PE system

The next step is to add the user IP-blocks within the PE (processing element) architecture. This involves instantiation of hardware modules and connecting them within the system interrupt-chain.

Suppose we need to integrate 2 user IP-blocks into the platform now, e.g the example modules described above. Before integrating the hardware modules, we must have the following files in hand.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

(1) data_type_definition_package.vhd
(2) bram_dma_source_entarch.vhd
(3) bram_memory_destination_entarch.vhd
(4) bram_memory_source_entarch.vhd
(5) bram_dma_destination_entarch.vhd
(6) dma_source_entarch.vhd
(7) memory_destination_entarch.vhd
(8) memory_source_entarch.vhd
(9) dma_destination_entarch.vhd
(10) dram_dma_source_entarch.vhd
(11) dram_memory_destination_entarch.vhd
(12) dram_memory_source_entarch.vhd
(13) dram_dma_destination_entarch.vhd
(14) virtual_socket_testing.vhd
(15) user_IP_block_0.vhd
(16) user_IP_block_1.vhd
(17) other necessary system synthesis files provided by Annapolis [5]

Among those files, “virtual_socket_testing.vhd” is the main platform VHDL file. Almost all the data signals and hardware module controllers (except DMA controllers) are defined in that file.

Then users can take next several steps and simply add or modify the following code fragments in “virtual_socket_testing.vhd”:

1. Signal declaration.
2. Component declaration.
3. Component instantiation.
4. Interrupt chain
5. Simulation tools and synthesis project file.

The details of these steps are shown below.

8.3.7.1 Signal declaration

Before the component declaration and instantiation, users should declare some necessary signals with the PE architecture in “virtual_socket_testing.vhd”. If users have 2 IP-blocks (Data Increase Module and Arithmetic Module) to be integrated into the platform, the signal declaration is shown below.

```
alias  I_Clk  : std_logic is LAD_Bus_In.RxClk_In.Clk;
type memory32bit is array (natural range <>) of
                          std_logic_vector(31 downto 0);
signal start  : std_logic_vector(1 downto 0);
signal input_valid  : std_logic_vector(1 downto 0);
signal in_block_buffer : memory32bit(0 to 1);
signal user_memory_read : std_logic_vector(1 downto 0);
signal user_HW_Mem_HWAccel : Memory32bit(0 to 1);
signal user_HW_Offset : Memory32bit(0 to 1);
signal user_HW_Count : Memory32bit(0 to 1);
signal out_block_buffer : memory32bit(0 to 1);
signal output_valid : std_logic_vector(1 downto 0);
signal user_HW_Mem_HWAccel1 : Memory32bit(0 to 1);
signal user_HW_Offset1 : Memory32bit(0 to 1);
signal user_HW_Count1 : Memory32bit(0 to 1);
signal output_ack : std_logic_vector(1 downto 0);
signal user_Host_Mem_HWAccel : Memory32bit(0 to 1);
signal user_Host_Offset : Memory32bit(0 to 1);
signal user_Host_Count : Memory32bit(0 to 1);
signal user_Host_Mem_HWAccel1 : Memory32bit(0 to 1);
signal user_Host_Offset1 : Memory32bit(0 to 1);
signal user_Host_Count1 : Memory32bit(0 to 1);
signal user_Mem_Read_Access_Req : std_logic_vector(31 downto 0);
signal user_Mem_Write_Access_Req : std_logic_vector(31 downto 0);
signal user_Mem_Read_Access_Release_Req : std_logic_vector(31 downto 0);
```

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2009 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
signal user_Mem_Write_Access_Release_Req : std_logic_vector(31 downto 0);
signal user_Mem_Read_Access_Ack : std_logic_vector(31 downto 0);
signal user_Mem_Write_Access_Ack : std_logic_vector(31 downto 0);
signal user_Mem_Read_Access_Release_Ack : std_logic_vector(31 downto 0);
signal user_Mem_Write_Access_Release_Ack : std_logic_vector(31 downto 0);
signal user_done : std_logic_vector(1 downto 0);
```

Figure 70 — Signal declaration for user IP-blocks.

8.3.7.2 Component declaration

This is a step before component instantiation. It is added to the architecture section before the “begin” keyword in “virtual_socket_testing.vhd”. For example, the component declaration for “Data Increase Module” and “Arithmetic Module” is as simple as the following.

```
component User_IP_Block_0
generic(block_width: positive:=8);
port(
   clk                          : in std_logic;
   reset                        : in std_logic;
   start                        : in std_logic;
   input_valid                  : in std_logic;
   data_in                      : in std_logic_vector(31 downto 0);
   memory_read                  : out std_logic;
   HW_Mem_HWAccel               : out std_logic_vector(31 downto 0);
   HW_Offset                    : out std_logic_vector(31 downto 0);
   HW_Count                     : out std_logic_vector(31 downto 0);
   data_out                     : out std_logic_vector(31 downto 0);
   output_valid                 : out std_logic;
   HW_Mem_HWAccel1              : out std_logic_vector(31 downto 0);
   HW_Offset1                   : out std_logic_vector(31 downto 0);
   HW_Count1                    : out std_logic_vector(31 downto 0);
   output_ack                   : in std_logic;
   Host_Mem_HWAccel             : in std_logic_vector(31 downto 0);
   Host_Offset                  : in std_logic_vector(31 downto 0);
   Host_Count                   : in std_logic_vector(31 downto 0);
   Host_Mem_HWAccel1            : in std_logic_vector(31 downto 0);
   Host_Offset1                 : in std_logic_vector(31 downto 0);
   Host_Count1                  : in std_logic_vector(31 downto 0);
   Mem_Read_Access_Req          : out std_logic;
   Mem_Write_Access_Req         : out std_logic;
   Mem_Read_Access_Release_Req  : out std_logic;
   Mem_Write_Access_Release_Req : out std_logic;
   Mem_Read_Access_Ack          : in std_logic;
   Mem_Write_Access_Ack         : in std_logic;
   Mem_Read_Access_Release_Ack  : in std_logic;
   Mem_Write_Access_Release_Ack : in std_logic;
   done                         : out std_logic
    );
  end component;
component User_IP_Block_1
generic(block_width: positive:=8);
port(
   clk                          : in std_logic;
   reset                        : in std_logic;
   start                        : in std_logic;
   input_valid                  : in std_logic;
   data_in                      : in std_logic_vector(31 downto 0);
   memory_read                  : out std_logic;
   HW_Mem_HWAccel               : out std_logic_vector(31 downto 0);
   HW_Offset                    : out std_logic_vector(31 downto 0);
   HW_Count                     : out std_logic_vector(31 downto 0);
   data_out                     : out std_logic_vector(31 downto 0);
   output_valid                 : out std_logic;
   HW_Mem_HWAccel1              : out std_logic_vector(31 downto 0);
   HW_Offset1                   : out std_logic_vector(31 downto 0);
   HW_Count1                    : out std_logic_vector(31 downto 0);
   output_ack                   : in std_logic;
   Host_Mem_HWAccel             : in std_logic_vector(31 downto 0);
   Host_Offset                  : in std_logic_vector(31 downto 0);
   Host_Count                   : in std_logic_vector(31 downto 0);
   Host_Mem_HWAccel1            : in std_logic_vector(31 downto 0);
   Host_Offset1                 : in std_logic_vector(31 downto 0);
   Host_Count1                  : in std_logic_vector(31 downto 0);
```


© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   Mem_Read_Access_Req          : out std_logic;
   Mem_Write_Access_Req         : out std_logic;
   Mem_Read_Access_Release_Req  : out std_logic;
   Mem_Write_Access_Release_Req : out std_logic;
   Mem_Read_Access_Ack          : in std_logic;
   Mem_Write_Access_Ack         : in std_logic;
   Mem_Read_Access_Release_Ack  : in std_logic;
   Mem_Write_Access_Release_Ack : in std_logic;
   done                         : out std_logic
    );
  end component;
```

Figure 71 — Component declaration for user IP-blocks.

8.3.7.3 Component instantiation

This involves a generic map and a port map. Component instantiation for the hardware modules is listed below.

```
u_myhardware_0 : User_IP_Block_0
port map(
        clk                          => I_Clk,
        reset                        => User_Global_Reset,
        start                        => start(0),
        input_valid                  => input_valid(0),
        data_in                      => in_block_buffer(0),
        memory_read                  => user_memory_read(0),
        HW_Mem_HWAccel               => user_HW_Mem_HWAccel(0),
        HW_Offset                    => user_HW_Offset(0),
        HW_Count                     => user_HW_Count(0),
        data_out                     => out_block_buffer(0),
        output_valid                 => output_valid(0),
        HW_Mem_HWAccel1              => user_HW_Mem_HWAccel1(0),
        HW_Offset1                   => user_HW_Offset1(0),
        HW_Count1                    => user_HW_Count1(0),
        output_ack                   => output_ack(0),
        Host_Mem_HWAccel             => user_Host_Mem_HWAccel(0),
        Host_Offset                  => user_Host_Offset(0),
        Host_Count                   => user_Host_Count(0),
        Host_Mem_HWAccel1            => user_Host_Mem_HWAccel1(0),
        Host_Offset1                 => user_Host_Offset1(0),
        Host_Count1                  => user_Host_Count1(0),
        Mem_Read_Access_Req          => user_Mem_Read_Access_Req(0),
        Mem_Write_Access_Req         => user_Mem_Write_Access_Req(0),
        Mem_Read_Access_Release_Req  =>
                                  user_Mem_Read_Access_Release_Req(0),
        Mem_Write_Access_Release_Req =>
                                 user_Mem_Write_Access_Release_Req(0),
        Mem_Read_Access_Ack          => user_Mem_Read_Access_Ack(0),
        Mem_Write_Access_Ack         => user_Mem_Write_Access_Ack(0),
        Mem_Read_Access_Release_Ack  =>
                                   user_Mem_Read_Access_Release_Ack(0),
        Mem_Write_Access_Release_Ack =>
                                 user_Mem_Write_Access_Release_Ack(0),
        done                         => user_done(0)
        );
u_myhardware_1 : User_IP_Block_1
port map(
        clk                          => I_Clk,
        reset                        => User_Global_Reset,
        start                        => start(1),
        input_valid                  => input_valid(1),
        data_in                      => in_block_buffer(1),
        memory_read                  => user_memory_read(1),
        HW_Mem_HWAccel               => user_HW_Mem_HWAccel(1),
        HW_Offset                    => user_HW_Offset(1),
        HW_Count                     => user_HW_Count(1),
        data_out                     => out_block_buffer(1),
        output_valid                 => output_valid(1),
        HW_Mem_HWAccel1              => user_HW_Mem_HWAccel1(1),
        HW_Offset1                   => user_HW_Offset1(1),
        HW_Count1                    => user_HW_Count1(1),
        output_ack                   => output_ack(1),
        Host_Mem_HWAccel             => user_Host_Mem_HWAccel(1),
        Host_Offset                  => user_Host_Offset(1),
```

Copyright International Organization for Standardization © ISO/IEC 2009 All rights reserved
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
        Host_Count                  => user_Host_Count(1),
        Host_Mem_HWAccel1           => user_Host_Mem_HWAccel1(1),
        Host_Offset1                => user_Host_Offset1(1),
        Host_Count1                 => user_Host_Count1(1),
        Mem_Read_Access_Req         => user_Mem_Read_Access_Req(1),
        Mem_Write_Access_Req        => user_Mem_Write_Access_Req(1),
        Mem_Read_Access_Release_Req  =>
                                user_Mem_Read_Access_Release_Req(1),
        Mem_Write_Access_Release_Req =>
                               user_Mem_Write_Access_Release_Req(1),
        Mem_Read_Access_Ack         => user_Mem_Read_Access_Ack(1),
        Mem_Write_Access_Ack        => user_Mem_Write_Access_Ack(1),
        Mem_Read_Access_Release_Ack  =>
                                user_Mem_Read_Access_Release_Ack(1),
        Mem_Write_Access_Release_Ack =>
                               user_Mem_Write_Access_Release_Ack(1),
        done                        => user_done(1)
        );
```

Figure 72 — Component instantiation for user IP-blocks.

Note: The “reset” for each user IP-block must be mapped to “User_Global_Reset” which is defined in “virtual_socket_testing.vhd”. The “clk” for each user IP-block can be mapped to the system clock “I_Clk” which is fixed to 50MHz. If users need to change the frequency of “clk”, it should be mapped to “M_Clk” which is a reprogrammable clock. Users can use an API function [5] to set corresponding frequency of “M_Clk”. Anyway, it is suggested that “I_Clk” be preferable.

8.3.7.4 Interrupt chain

Other than the hardware module declaration and instantiation, the interrupt signals generated by user IP-blocks must be added to the platform interrupt chain. Actually, there is a process named “u_int_gen” in “virtual_socket_testing.vhd”, which deals with the interrupt generation function. Users should add appropriate interrupt signals to the chain which is shown below.

```
u_int_gen : process (I_Clk, Global_Reset)
begin
  if (Global_Reset = '1') then
Int_Counter <= "10000";            --Don’t touch this signal
  elsif (rising_edge(I_Clk)) then
    if ((dram_hardware_module_done = '0') or
                (DMA_Destination_Done = '0') or
        (DMA_Source_Done = '0') or
        (Memory_Destination_Done = '0') or
        (Memory_Source_Done = '0') or
        (DRAM_DMA_Destination_Done = '0') or
        (DRAM_DMA_Source_Done = '0') or
        (DRAM_Memory_Destination_Done = '0') or
        (DRAM_Memory_Source_Done = '0') or
        (BRAM_DMA_Destination_Done = '0') or
        (BRAM_DMA_Source_Done = '0') or
        (BRAM_Memory_Destination_Done = '0') or
        (BRAM_Memory_Source_Done = '0') or
        ((Interrupt(0) or Interrupt_Mask(0)) = '0') or
        ((Interrupt(1) or Interrupt_Mask(1)) = '0')) then
           Int_Counter <= (others => '0');  --Don’t touch this signal
    elsif (Interrupt_Done = '0') then
      Int_Counter <= Int_Counter + 1;  --Don’t touch this signal
    end if;
  end if;
end process;
```

Figure 73 — Interrupt chain for platform and user IP-blocks.

Note: Each user IP-block interrupt signal is shielded by its corresponding interrupt mask bit. A final user IP-block related interrupt is determined by its mask bit status. When the mask bit is set to “1”, there will be no interrupt generated for user IP-block. To enable the user IP-block interrupt generation, the mask bit should be set to “0”. Users can set up Interrupt_Mask register in the API function calls.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Besides the user interrupt signals, there are many other DMA related interrupt signals in the chain which are generated by platform itself, users must not change those signals.

8.3.7.5 Simulation of the whole platform

The virtual socket platform is simulated by Mentor Graphics© ModelSim 5.4e® or high version of the simulation tools. The steps show how to simulate the whole platform using simulation tools. Before simulating the platform, users must have the following files in hand.“system_virtual_socket_arch.vhd”, “system_cfg.vhd”, “host_virtual_socket_arch.vhd”, “project_vcom.do” and “project_vsim.do”. Please refer to the reference manual of the WildCard II [5] for more details about the function of each file.

a. Use “project_vcom.do” to compile all the virtual socket platform VHDL files, including the user IP-blocks. The users may modify the project directory in this file for their own applications.

```
------------------------------------------------------------------------
--
--  Model Technology ModelSim(tm) Compilation Macro File
--
--  Follow the itemized instructions below in order to customize
--  the WILDSTAR(tm) simulation model compilation script to meet
--  your application needs.
--
--  To execute this macro file from the ModelSim(tm) VHDL simulator:
--
--    Using the "File->Execute Macro" menu option select this
--    file in the browser window and click on "Open".
--
------------------------------------------------------------------------

------------------------------------------------------------------------
--
--  1. Change the PROJECT_BASE and WILDCARD_BASE variables to your
--     project directory and WILDCARD(tm)-II installation directory,
--     respectively:
--
------------------------------------------------------------------------
set ANNAPOLIS_BASE c:/annapolis
set PROJECT_BASE $ANNAPOLIS_BASE/wildcard_ii/examples/virtual_socket_testing/vhdl/sim

--
--  Do not change these lines!!
--
set WILDCARD_BASE   $ANNAPOLIS_BASE/wildcard_ii
set PE_BASE         $WILDCARD_BASE/vhdl/pe
set SIMULATION_BASE $WILDCARD_BASE/vhdl/simulation
set TOOLS_BASE      $WILDCARD_BASE/vhdl/tools

do $SIMULATION_BASE/simulation_vcom.do
------------------------------------------------------------------------
--
--  2. Add your project's PE architecture VHDL file here, as
--     well as any VHDL files on which your PE design depends:
--
------------------------------------------------------------------------
vcom -93 -explicit -work pe_lib $PROJECT_BASE/data_type_definition_package.vhd
vcom -93 -explicit -work pe_lib $PROJECT_BASE/bram_dma_source_entarch.vhd
vcom -93 -explicit -work pe_lib $PROJECT_BASE/bram_memory_destination_entarch.vhd
vcom -93 -explicit -work pe_lib $PROJECT_BASE/bram_memory_source_entarch.vhd
vcom -93 -explicit -work pe_lib $PROJECT_BASE/bram_dma_destination_entarch.vhd
vcom -93 -explicit -work pe_lib
$PROJECT_BASE/dma_source_entarch.vhd
vcom -93 -explicit -work pe_lib $PROJECT_BASE/memory_destination_entarch.vhd
vcom -93 -explicit -work pe_lib $PROJECT_BASE/memory_source_entarch.vhd
vcom -93 -explicit -work pe_lib $PROJECT_BASE/dma_destination_entarch.vhd
vcom -93 -explicit -work pe_lib $PROJECT_BASE/dram_dma_source_entarch.vhd
vcom -93 -explicit -work pe_lib $PROJECT_BASE/dram_memory_destination_entarch.vhd
vcom -93 -explicit -work pe_lib $PROJECT_BASE/dram_memory_source_entarch.vhd
vcom -93 -explicit -work pe_lib $PROJECT_BASE/dram_dma_destination_entarch.vhd
vcom -93 -explicit -work pe_lib $PROJECT_BASE/User_IP_Block_0.vhd
vcom -93 -explicit -work pe_lib $PROJECT_BASE/User_IP_Block_1.vhd
```

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
vcom -93 -explicit -work pe_lib $PROJECT_BASE/virtual_socket_testing.vhd

------------------------------------------------------------------------
--
--  3. Add your project's IO Connector architecture VHDL file here, as
--     well as any VHDL files on which your IO connector design depends:
--
------------------------------------------------------------------------
--vcom -93 -explicit -work pe_lib $PROJECT_BASE/<IO Architecture>.vhd
------------------------------------------------------------------------
--
--  4. Add your project's host architecture VHDL file here, as
--     well as any VHDL files on which your host design depends:
--
------------------------------------------------------------------------
vcom -93 -explicit -work simulation $PROJECT_BASE/host_virtual_socket_arch.vhd

------------------------------------------------------------------------
--
--  8. Add your project's system configuration VHDL file here:
--
------------------------------------------------------------------------
vcom -93 -explicit -work simulation $PROJECT_BASE/system_virtual_socket_arch.vhd
vcom -93 -explicit -work simulation $PROJECT_BASE/system_cfg.vhd
```

Figure 74 — ModelSim compilation macro file for the project.

b. Use “project_vsim.do” to execute the simulation operations.

```
------------------------------------------------------------------------
--  Copyright (C) 2002, Annapolis Micro Systems, Inc.
--  All Rights Reserved.
--
------------------------------------------------------------------------

------------------------------------------------------------------------
--
--  Model Technology ModelSim(tm) Simulation Macro File
--
--  Follow the itemized instructions below in order to customize
--  the WILDSTAR(tm) simulation model simulation script to meet
--  your application needs.
--
--  To execute this macro file from the ModelSim(tm) VHDL simulator:
--
--    Using the "File->Execute Macro" menu option select this
--    file in the browser window and click on "Open".
--
------------------------------------------------------------------------
vsim -t ps simulation.system_config
```

Figure 75 — ModelSim simulation macro file for the project.

The key point of the virtual socket system simulation is to write appropriate system testbench file. Actually, “host_virtual_socket_arch.vhd” is the testbench file for this simulation. The reference manual of WildCard II [5] lists all the necessary simulated WildCard II API function calls (host APIs). Because virtual socket API function calls are constructed directly on the WildCard II API function calls, users may refer to the “WildCard_Program_Library_Functions.c” and replace the virtual socket API function calls with some appropriate host APIs to trigger and simulate the whole virtual socket platform.

For example, if the following is a paragraph of the C codes as a conformance test.

```
int main(int argc,char *argv[])
{
    WC_RetCode rc=WC_SUCCESS;
WC_TestInfo TestInfo;
DWORD Buffer0[512];
```


© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
rc=VSInit(&TestInfo);
if (rc != WC_SUCCESS)
{
    rc = WC_Close(TestInfo.DeviceNum);
    return rc;
}

for (i=0;i<16;i++)
{
    Buffer0[i]=i;
}
HWAccel = 0;
Offset = 0x00000000;
Count = 16;
Display = 1;
rc=VSWrite(&TestInfo,HWAccel,Offset,Count,Buffer0);
rc=VSRead(&TestInfo,HWAccel,Offset,Count,Buffer0);

rc = WC_Close(TestInfo.DeviceNum);
return(rc);
}
```

Figure 76 — An example of the C codes for testing the platform.

The functionality of the C codes is to test the register data transfer by non-DMA operations. The codes utilize some virtual socket API function calls, i.e “VSInit”, “VSWrite” and “VSRead”. Besides, the original WildCard II API function call “WC_Close” is also used in the C codes. To trigger and simulate the whole platform, users may translate those C codes and replace the necessary virtual socket API function calls with appropriate simulated WildCard II API functions (host APIs), write the testbench codes in “host_virtual_socket_arch.vhd”, compile all the project file using “project_vcom.do” & “project_vsim.do”, and start to simulate the whole platform in the ModelSim environment.

The following is an example testbench file after translating those C codes to the host APIs.

```
------------------------- Library Declarations ------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;

library SIMULATION;
use SIMULATION.Host_Package.all;
use SIMULATION.Clock_Generator_Package.all;

----------------------- Architecture Declaration -------------------

architecture virtual_socket of Host is

begin
  ---------------------------------------------------------
  --
  --  Below is an example of how to use host "API"
  --  functions to send commands to the WILDSTAR(tm) II
  --  board.  Copy this file to your project directory
  --  and modify it to simulate the behavior of the
  --  host portion of your application.
  --
  ---------------------------------------------------------
  main : process

  variable Device              : WC_DeviceNum := 0;
  variable fClkFreqMhz         : float;
  variable Buffer0             : DWORD_array ( 0 to 511 );
  variable Result_Buffer       : DWORD_array ( 0 to 511 );
  variable dControl            : DWORD_array ( 0 to 3 );
  variable bReset              : BOOLEAN := false;
  constant HWAccel             : DWORD := 16#0#;
  constant Offset              : DWORD := 16#0#;
  constant Count               : DWORD := 16#10#;
  constant User_Reset_Disable  : DWORD := 16#fff6#;
  constant DMA_Working         : DWORD := 16#fff7#;

  begin
    ---------------------------------------------------------
```

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    ----  At first, deassert the command request signal --
    ----  Leave this line uncommented.                 --
    -------------------------------------------------------
    Cmd_Req <= FALSE;
    --
    -------------------------------------------------------
    ----  Set the board number to 0.  This is the default --
    ----  configuration.  For multiboard systems, or      --
    ----  single board systems where the board in not in  --
    ----  slot 0, set this variable to the desired board  --
    ----  number as needed.                               --
    -------------------------------------------------------
    --
    Device := 0;
bReset  := false;
    WC_PeReset
    (
      Cmd_Req,
      Cmd_Ack,
      device,
      bReset
);
wait for 100ns;

    bReset  := true;
    WC_PeReset
    (
      Cmd_Req,
      Cmd_Ack,
      device,
      bReset
);
wait for 100ns;

    bReset  := false;
    WC_PeReset
    (
      Cmd_Req,
      Cmd_Ack,
      device,
      bReset
);

    fClkFreqMhz := 50.0;
    WC_ClkSetFrequency
    (
      Cmd_Req,
      Cmd_Ack,
      device,
      fClkFreqMhz
);

WC_IntEnable
(
  Cmd_Req,
  Cmd_Ack,
  device,
  false
);

    WC_IntReset
    (
      Cmd_Req,
      Cmd_Ack,
      Device
);
WC_IntEnable
(
  Cmd_Req,
  Cmd_Ack,
  device,
  true
    );

        dControl(0) := 16#0#;
    WC_PeRegWrite
```


© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
    (
      Cmd_Req,
      Cmd_Ack,
      device,
      User_Reset_Disable,
      1,
      dControl
    );

        for i in 0 to Count-1 loop
      Buffer0(i) := i;
end loop;

    dControl(0) := 16#0#;
    WC_PeRegWrite
    (
      Cmd_Req,
      Cmd_Ack,
      device,
      DMA_Working,
      1,
      dControl
);

    WC_PeRegWrite
    (
      Cmd_Req,
      Cmd_Ack,
      device,
      Offset,
      Count,
      Buffer0
);
report " Finished Writing 16 DWORDs to Registers...";

    dControl(0) := 16#0#;
    WC_PeRegWrite
    (
      Cmd_Req,
      Cmd_Ack,
      device,
      DMA_Working,
      1,
      dControl
);
          for i in 0 to Count-1 loop
       WC_PeRegRead
       (
         Cmd_Req,
         Cmd_Ack,
         device,
         Offset + i,
         1,
         dControl
       );
       Result_Buffer(i) := dControl(0);
     end loop;

          for i in 0 to Count-1 loop
       report "Origianl Data(" & integer'image(i) & ") = " &
       Hex_To_String(Buffer0(i)) & ", Read Data(" & integer'image(i)
       & ") = " & Hex_To_String(Result_Buffer(i));
     end loop;

     report "Successfully Reading 16 DWORDs from Registers...";
     report "Example Complete";

     wait;  -- End of the simulated host program

   end process;

end architecture;
```

Figure 77 — An example of the simulation testbench file host_virtual_socket_arch.vhd.

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.3.7.6 Synthesis project file

The architectures of the platform hardware modules are prototyped using VHDL language, and synthesized using Synplicity© Synplify®. The target technology is the FPGA device (2V3000fg676-4) from the Xilinx© Virtex-II®. The following is a synthesis file for implementation of the whole virtual socket platform system.

```
#-----------------------------------------------------------------------
#-
#-  1. Change the PROJECT_BASE and WILDCARD_BASE variables to your
#-      project directory and WILDCARD-II installation directory,
#-      respectively:
#-
#-----------------------------------------------------------------------
set ANNAPOLIS_BASE "c:/annapolis"
set PROJECT_BASE  "$ANNAPOLIS_BASE/wildcard_ii/examples/virtual_socket/vhdl/sim"

#
# Do not modify these lines!
#
set WILDCARD_BASE   "$ANNAPOLIS_BASE/wildcard_ii"
set PE_BASE         "$WILDCARD_BASE/vhdl/pe"
set SIMULATION_BASE "$WILDCARD_BASE/vhdl/simulation"
set TOOLS_BASE      "$WILDCARD_BASE/vhdl/tools"
set SYNTHESIS_BASE  "$WILDCARD_BASE/vhdl/synthesis"

source "$SYNTHESIS_BASE/tcl/wildcard_pe.tcl"

#-----------------------------------------------------------------------
#-
#-  2. Add your project's PE architecture VHDL file here, as
#-      well as any VHDL or constraint files on which your PE
#-      design depends:
#-      NOTE: They must be in the order specified below.
#-
#-----------------------------------------------------------------------

add_file -vhdl -lib pe_lib "$PROJECT_BASE/data_type_definition_package.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/bram_dma_source_entarch.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/bram_memory_destination_entarch.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/bram_memory_source_entarch.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/bram_dma_destination_entarch.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/dma_source_entarch.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/memory_destination_entarch.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/memory_source_entarch.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/dma_destination_entarch.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/dram_dma_source_entarch.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/dram_memory_destination_entarch.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/dram_memory_source_entarch.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/dram_dma_destination_entarch.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/User_IP_Block_0.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/User_IP_Block_1.vhd"
add_file -vhdl -lib pe_lib "$PE_BASE/pe_ent.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/virtual_socket_testing.vhd"
#-----------------------------------------------------------------------
#-
#-  3. Change the result file:
#-
#-----------------------------------------------------------------------
project -result_file "virtual_socket.edf"
#-----------------------------------------------------------------------
#-
#-  4. Add or change the following options according to your
#-      project's needs:
#-----------------------------------------------------------------------

# State machine encoding
set_option -default_enum_encoding sequential

# State machine compiler
set_option -symbolic_fsm_compiler false

# Resource sharing
set_option -resource_sharing true
```


© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
# Clock frequency
set_option -frequency 132.000

# Fan-out limit
set_option -fanout_limit 1000

# Hard maximum fanout limit
set_option -maxfan_hard false
```

Figure 78 — A synthesis project file for platform implementation.

Note: After all the files are successfully synthesized, users should perform the FPGA routing and image file generation process [5]. The final image file .x86 will be generated and thus downloaded into the FPGA to implement the platform hardware functions.

Due to the limitation of FPGA synthesis resources, although there are maximum of 32 user IP-blocks can be integrated into the platform, it is strongly suggested that for each user IP-block integration, the corresponding function block of Registers and BRAM resources be integrated into the platform. Otherwise, just leave the idle function blocks of Registers and BRAM resources commented and not performed. For example, if users need to integrate 6 user IP-blocks into the platform, they should take the above-mention steps to add 6 user IP-blocks to the platform as well as uncomment appropriate parts of the code in "virtual_socket_testing.vhd" to add corresponding 6 function blocks of Registers and BRAM resources to the platform. After that, users can synthesize the whole platform and generate the final .x86 image file.

Please refer to the comments in "virtual_socket_testing.vhd" to get more understanding about it.

8.3.8 A guide for using platform system software

To use the platform system software and issue the virtual socket API function calls to trigger the hardware modules to work, we need build a project for compiling all the C source files necessary for the system implementation. Suppose we have "WildCard_Program_Library_Functions.c" which includes all the necessary virtual socket API function calls, "WildCard_Program_Virtual_Socket.c" which is a main file for system implementation and "WildCard_Program_Virtual_Socket.h" which is a C header file. Then we may take the following simple steps to build a project and compile all the files included in the project.

a. Create a new folder under the root directory of an appropriate hard disk,

e.g C:\Digital_Video_Processing_Board_Design\WildCard.

b. Utilize the VC++ software system to create a new project based on Win32 console application under that folder. After the new project is created, a subfolder will be generated automatically as the project directory.

For example, if we create a new project named "VC_WildCard_Program", then we will have C:\Digital_Video_Processing_Board_Design\WildCard\VC_WildCard_Program as the project directory.

There should be another subfolder automatically generated by VC++ and thus under this project directory:

C:\Digital_Video_Processing_Board_Design\WildCard\VC_WildCard_Program\Debug

c. Copy "WildCard_Program_Virtual_Socket.c" to the project directory and add it to the project as a C source file.

d. Copy "WildCard_Program_Library_Functions.c" to the project directory and add it to the project as a source file.

e. Copy "wc_shared.c" to the project directory and add it to the project as a source file, this file is provided by Annapolis.

f. Copy "wcapi.lib" and "wcapi.dll" to the project directory and add "wcapi.lib" as a library file in the project. Besides, copy "wcapi.dll" to the Debug directory.

g. Copy "WildCard_Program_Virtual_Socket.h" to the project directory.

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

h. Copy “basetsd.h” to the project directory, this file is provided by the VC++ software system.

i. Copy “wc_shared.h”, “wc_windows.h” and “wcdefs.h” to the project directory, those files are provided by Annapolis.

j. Then build all the files within the project and get the correct compiling and linking information.

k. Create an image file directory:

C:\Digital_Video_Processing_Board_Design\WildCard\VC_WildCard_Program\

Debug\XC2V3000\PKG_FG676

Copy appropriate image file .x86 to the image file directory.

l. Run the execution program. This can be done by open a Dos prompt window under operating system. Enter the Debug directory, there should be an .exe file existing, then run it.

m. If the user doesn’t like to open a Dos window and type Dos command to run the program, another way is to run the execution program directly under VC++ environment. In this case, the user should create a directory:

C:\Digital_Video_Processing_Board_Design\WildCard\VC_WildCard_Program\

XC2V3000\PKG_FG676

Copy appropriate image file .x86 to this folder, then execute the program directly under VC++ system environment.

NOTE When users need to use their own C source file as the main program, just replace the “WildCard_Program_Virtual_Socket.c” with user’s own C source file and call the appropriate API functions defined above.

Please refer to the “Reference for Virtual Socket API Function Calls” for more details.

8.3.9 Conformance Tests

We can set up a conformance testing program to verify the features in the virtual socket platform. The program is coded in C language. There is a debug menu for the program, and we may interactively use debug menu and choose appropriate options to perform the feature testing.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

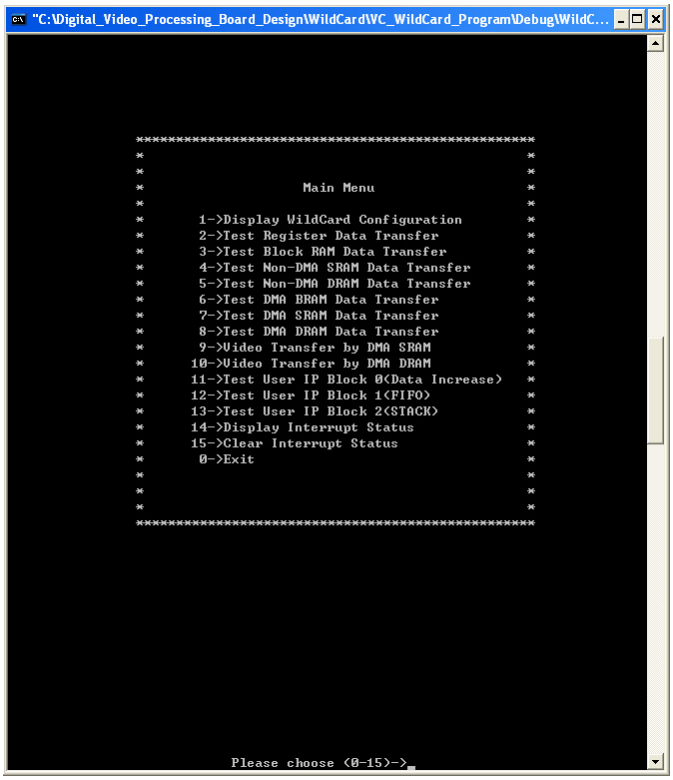


Figure 79 — Debug menu for conformance tests.

Currently, we have 15 feature options included in the debug menu. Those options can test all the 4 types of physical memory space as well as the hardware interface module controller which connects the user IP-blocks to the virtual socket platform.

Option 1 : Get the configuration information of the WildCard II Platform such as platform hardware version, FPGA type, FPGA package type, FPGA speed grade, RAM size, RAM access speed, API version, Driver version and Firmware version etc.

Option 2 : Test general registers.

Option 3 : Test Block RAM based on non-DMA operation..

Option 4 : Test SRAM based on non-DMA operation.

Option 5 : Test DRAM based on non-DMA operation.

Option 6: Test Block RAM based on DMA operation.

Option 7: Test SRAM based on DMA operation.

Option 8: Test DRAM based on DMA operation.

Option 9: Test SRAM video data transfer based on DMA operation. The video frames are in CIF YUV 4:2:0 format.

Option 10: Test DRAM video data transfer based on DMA operation. The video frames are in CIF YUV 4:2:0 format.

Option 11: Test Data Increase Module as user IP-block 0.

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Option 12: Test FIFO Module as user IP-block 1.

Option 13: Test STACK Module as user IP-block 2.

Option 14: Display internal user interrupt status register.

Option 15: Cleanup internal user interrupt status register.

8.3.10 Appendix

The following is a test result of the data transfer rate through SRAM by DMA operation.

```
 Now testing DMA-to-PE...
  BurstSize=128 Iteration=512 Time=0.070000Sec, Transfer Rate=3.744914MB/S
  BurstSize=128 Iteration=1024 Time=0.120000Sec, Transfer Rate=4.369067MB/S
  BurstSize=128 Iteration=2048 Time=0.241000Sec, Transfer Rate=4.350938MB/S
  BurstSize=128 Iteration=4096 Time=0.501000Sec, Transfer Rate=4.185932MB/S
  BurstSize=128 Iteration=8192 Time=0.891000Sec, Transfer Rate=4.707412MB/S
  BurstSize=128 Iteration=16384 Time=1.854000Sec, Transfer Rate=4.524600MB/S
  BurstSize=128 Iteration=32768 Time=3.717000Sec, Transfer Rate=4.513644MB/S
  BurstSize=128 Iteration=65536 Time=7.571000Sec, Transfer Rate=4.431968MB/S
  BurstSize=256 Iteration=512 Time=0.080000Sec, Transfer Rate=6.553600MB/S
  BurstSize=256 Iteration=1024 Time=0.171000Sec, Transfer Rate=6.132023MB/S
  BurstSize=256 Iteration=2048 Time=0.300000Sec, Transfer Rate=6.990507MB/S
  BurstSize=256 Iteration=4096 Time=0.611000Sec, Transfer Rate=6.864655MB/S
  BurstSize=256 Iteration=8192 Time=1.252000Sec, Transfer Rate=6.700166MB/S
  BurstSize=256 Iteration=16384 Time=2.474000Sec, Transfer Rate=6.781413MB/S
  BurstSize=256 Iteration=32768 Time=4.689000Sec, Transfer Rate=7.155989MB/S
  BurstSize=512 Iteration=512 Time=0.090000Sec, Transfer Rate=11.650844MB/S
  BurstSize=512 Iteration=1024 Time=0.190000Sec, Transfer Rate=11.037642MB/S
  BurstSize=512 Iteration=2048 Time=0.361000Sec, Transfer Rate=11.618571MB/S
  BurstSize=512 Iteration=4096 Time=0.811000Sec, Transfer Rate=10.343536MB/S
  BurstSize=512 Iteration=8192 Time=1.613000Sec, Transfer Rate=10.401250MB/S
  BurstSize=512 Iteration=16384 Time=3.185000Sec, Transfer Rate=10.535143MB/S
  BurstSize=1024 Iteration=512 Time=0.171000Sec, Transfer Rate=12.264047MB/S
  BurstSize=1024 Iteration=1024 Time=0.270000Sec, Transfer Rate=15.534459MB/S
  BurstSize=1024 Iteration=2048 Time=0.631000Sec, Transfer Rate=13.294149MB/S
  BurstSize=1024 Iteration=4096 Time=1.182000Sec, Transfer Rate=14.193922MB/S
  BurstSize=1024 Iteration=8192 Time=2.434000Sec, Transfer Rate=13.785716MB/S
  BurstSize=2048 Iteration=512 Time=0.271000Sec, Transfer Rate=15.477137MB/S
  BurstSize=2048 Iteration=1024 Time=0.530000Sec, Transfer Rate=15.827562MB/S
  BurstSize=2048 Iteration=2048 Time=1.092000Sec, Transfer Rate=15.363751MB/S
  BurstSize=2048 Iteration=4096 Time=2.143000Sec, Transfer Rate=15.657691MB/S
  BurstSize=4096 Iteration=512 Time=0.491000Sec, Transfer Rate=17.084741MB/S
  BurstSize=4096 Iteration=1024 Time=0.952000Sec, Transfer Rate=17.623126MB/S
  BurstSize=4096 Iteration=2048 Time=1.963000Sec, Transfer Rate=17.093445MB/S
  BurstSize=8192 Iteration=512 Time=0.931000Sec, Transfer Rate=18.020640MB/S
  BurstSize=8192 Iteration=1024 Time=1.873000Sec, Transfer Rate=17.914806MB/S
  BurstSize=16384 Iteration=512 Time=1.852000Sec, Transfer Rate=18.117944MB/S
  BurstSize=16384 Iteration=1024 Time=3.706000Sec, Transfer Rate=18.108166MB/S
  BurstSize=32768 Iteration=512 Time=3.635000Sec, Transfer Rate=18.461861MB/S
  BurstSize=32768 Iteration=1024 Time=7.250000Sec, Transfer Rate=18.512790MB/S
  BurstSize=65536 Iteration=512 Time=7.211000Sec, Transfer Rate=18.612915MB/S
  BurstSize=131072 Iteration=512 Time=14.360000Sec, Transfer Rate=18.693277MB/S
Now testing DMA-from-PE...
  BurstSize=128 Iteration=512 Time=0.040000Sec, Transfer Rate=6.553600MB/S
  BurstSize=128 Iteration=1024 Time=0.080000Sec, Transfer Rate=6.553600MB/S
  BurstSize=128 Iteration=2048 Time=0.150000Sec, Transfer Rate=6.990507MB/S
  BurstSize=128 Iteration=4096 Time=0.360000Sec, Transfer Rate=5.825422MB/S
  BurstSize=128 Iteration=8192 Time=0.642000Sec, Transfer Rate=6.533184MB/S
  BurstSize=128 Iteration=16384 Time=1.221000Sec, Transfer Rate=6.870277MB/S
  BurstSize=128 Iteration=32768 Time=2.574000Sec, Transfer Rate=6.517955MB/S
  BurstSize=128 Iteration=65536 Time=5.126000Sec, Transfer Rate=6.545929MB/S
  BurstSize=256 Iteration=512 Time=0.050000Sec, Transfer Rate=10.485760MB/S
  BurstSize=256 Iteration=1024 Time=0.090000Sec, Transfer Rate=11.650844MB/S
  BurstSize=256 Iteration=2048 Time=0.190000Sec, Transfer Rate=11.037642MB/S
  BurstSize=256 Iteration=4096 Time=0.371000Sec, Transfer Rate=11.305402MB/S
  BurstSize=256 Iteration=8192 Time=0.651000Sec, Transfer Rate=12.885727MB/S
  BurstSize=256 Iteration=16384 Time=1.323000Sec, Transfer Rate=12.681191MB/S
  BurstSize=256 Iteration=32768 Time=2.714000Sec, Transfer Rate=12.363461MB/S
  BurstSize=512 Iteration=512 Time=0.060000Sec, Transfer Rate=17.476267MB/S
  BurstSize=512 Iteration=1024 Time=0.101000Sec, Transfer Rate=20.763881MB/S
  BurstSize=512 Iteration=2048 Time=0.230000Sec, Transfer Rate=18.236104MB/S
```


© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
BurstSize=512 Iteration=4096 Time=0.421000Sec, Transfer Rate=19.925435MB/S
BurstSize=512 Iteration=8192 Time=0.861000Sec, Transfer Rate=19.485733MB/S
BurstSize=512 Iteration=16384 Time=1.541000Sec, Transfer Rate=21.774453MB/S
BurstSize=1024 Iteration=512 Time=0.070000Sec, Transfer Rate=29.959314MB/S
BurstSize=1024 Iteration=1024 Time=0.110000Sec, Transfer Rate=38.130036MB/S
BurstSize=1024 Iteration=2048 Time=0.250000Sec, Transfer Rate=33.554432MB/S
BurstSize=1024 Iteration=4096 Time=0.510000Sec, Transfer Rate=32.896502MB/S
BurstSize=1024 Iteration=8192 Time=0.882000Sec, Transfer Rate=38.043574MB/S
BurstSize=2048 Iteration=512 Time=0.090000Sec, Transfer Rate=46.603378MB/S
BurstSize=2048 Iteration=1024 Time=0.150000Sec, Transfer Rate=55.924053MB/S
BurstSize=2048 Iteration=2048 Time=0.311000Sec, Transfer Rate=53.946032MB/S
BurstSize=2048 Iteration=4096 Time=0.631000Sec, Transfer Rate=53.176596MB/S
BurstSize=4096 Iteration=512 Time=0.130000Sec, Transfer Rate=64.527754MB/S
BurstSize=4096 Iteration=1024 Time=0.240000Sec, Transfer Rate=69.905067MB/S
BurstSize=4096 Iteration=2048 Time=0.461000Sec, Transfer Rate=72.786187MB/S
BurstSize=8192 Iteration=512 Time=0.211000Sec, Transfer Rate=79.512872MB/S
BurstSize=8192 Iteration=1024 Time=0.400000Sec, Transfer Rate=83.886080MB/S
BurstSize=16384 Iteration=512 Time=0.350000Sec, Transfer Rate=95.869806MB/S
BurstSize=16384 Iteration=1024 Time=0.731000Sec, Transfer Rate=91.804192MB/S
BurstSize=32768 Iteration=512 Time=0.681000Sec, Transfer Rate=98.544587MB/S
BurstSize=32768 Iteration=1024 Time=1.372000Sec, Transfer Rate=97.826332MB/S
BurstSize=65536 Iteration=512 Time=1.342000Sec, Transfer Rate=100.013210MB/S
BurstSize=131072 Iteration=512 Time=2.643000Sec, Transfer Rate=101.564683MB/S
Now testing DMA-to-PE-from-PE...
BurstSize=128 Iteration=512 Time=0.100000Sec, Transfer Rate=5.242880MB/S
BurstSize=128 Iteration=1024 Time=0.180000Sec, Transfer Rate=5.825422MB/S
BurstSize=128 Iteration=2048 Time=0.430000Sec, Transfer Rate=4.877098MB/S
BurstSize=128 Iteration=4096 Time=0.811000Sec, Transfer Rate=5.171768MB/S
BurstSize=128 Iteration=8192 Time=1.494000Sec, Transfer Rate=5.614865MB/S
BurstSize=128 Iteration=16384 Time=3.205000Sec, Transfer Rate=5.234701MB/S
BurstSize=128 Iteration=32768 Time=6.256000Sec, Transfer Rate=5.363560MB/S
BurstSize=128 Iteration=65536 Time=12.678000Sec, Transfer Rate=5.293332MB/S
BurstSize=256 Iteration=512 Time=0.140000Sec, Transfer Rate=7.489829MB/S
BurstSize=256 Iteration=1024 Time=0.231000Sec, Transfer Rate=9.078580MB/S
BurstSize=256 Iteration=2048 Time=0.440000Sec, Transfer Rate=9.532509MB/S
BurstSize=256 Iteration=4096 Time=0.941000Sec, Transfer Rate=8.914567MB/S
BurstSize=256 Iteration=8192 Time=1.844000Sec, Transfer Rate=9.098273MB/S
BurstSize=256 Iteration=16384 Time=3.614000Sec, Transfer Rate=9.284569MB/S
BurstSize=256 Iteration=32768 Time=7.461000Sec, Transfer Rate=8.994621MB/S
BurstSize=512 Iteration=512 Time=0.150000Sec, Transfer Rate=13.981013MB/S
BurstSize=512 Iteration=1024 Time=0.330000Sec, Transfer Rate=12.710012MB/S
BurstSize=512 Iteration=2048 Time=0.531000Sec, Transfer Rate=15.797755MB/S
BurstSize=512 Iteration=4096 Time=1.231000Sec, Transfer Rate=13.628933MB/S
BurstSize=512 Iteration=8192 Time=2.262000Sec, Transfer Rate=14.833966MB/S
BurstSize=512 Iteration=16384 Time=4.815000Sec, Transfer Rate=13.937459MB/S
BurstSize=1024 Iteration=512 Time=0.221000Sec, Transfer Rate=18.978751MB/S
BurstSize=1024 Iteration=1024 Time=0.450000Sec, Transfer Rate=18.641351MB/S
BurstSize=1024 Iteration=2048 Time=0.882000Sec, Transfer Rate=19.021787MB/S
BurstSize=1024 Iteration=4096 Time=1.623000Sec, Transfer Rate=20.674327MB/S
BurstSize=1024 Iteration=8192 Time=3.434000Sec, Transfer Rate=19.542476MB/S
BurstSize=2048 Iteration=512 Time=0.361000Sec, Transfer Rate=23.237141MB/S
BurstSize=2048 Iteration=1024 Time=0.721000Sec, Transfer Rate=23.269370MB/S
BurstSize=2048 Iteration=2048 Time=1.392000Sec, Transfer Rate=24.105195MB/S
BurstSize=2048 Iteration=4096 Time=2.744000Sec, Transfer Rate=24.456583MB/S
BurstSize=4096 Iteration=512 Time=0.621000Sec, Transfer Rate=27.016451MB/S
BurstSize=4096 Iteration=1024 Time=1.201000Sec, Transfer Rate=27.938744MB/S
BurstSize=4096 Iteration=2048 Time=2.444000Sec, Transfer Rate=27.458619MB/S
BurstSize=8192 Iteration=512 Time=1.152000Sec, Transfer Rate=29.127111MB/S
BurstSize=8192 Iteration=1024 Time=2.273000Sec, Transfer Rate=29.524357MB/S
BurstSize=16384 Iteration=512 Time=2.192000Sec, Transfer Rate=30.615358MB/S
BurstSize=16384 Iteration=1024 Time=4.427000Sec, Transfer Rate=30.317987MB/S
BurstSize=32768 Iteration=512 Time=4.316000Sec, Transfer Rate=31.097713MB/S
BurstSize=32768 Iteration=1024 Time=8.633000Sec, Transfer Rate=31.094111MB/S
BurstSize=65536 Iteration=512 Time=8.562000Sec, Transfer Rate=31.351957MB/S
BurstSize=131072 Iteration=512 Time=17.003000Sec, Transfer Rate=31.575070MB/S
```

Figure 80 — A test result of the DMA data transfer rate.

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.4 An Integration of the MPEG-4 Part 10/AVC DCT/Q Hardware Module into the Virtual Socket Co-design Platform

8.4.1 Scope

This subclause presents the integration of a MPEG-4 Part 10/AVC DCT/Q module into a virtual socket co-design platform framework for the conformance testing. The co-design platform is developed to implement MPEG-4 and its video applications. With the integration of MPEG-4 video processing IP blocks into the platform, the above-mentioned goal can be achieved. Furthermore, the platform system performance, in this case, can be tested and compared with those of some other software solutions.

The point of this subclause is to stress how to integrate an AVC/H.264 DCT/Q module into the co-design platform framework, and thus illustrate the possibility of rapidly developing a working system for MPEG-4 video applications.

8.4.2 Introduction

An integrated virtual socket hardware-accelerated co-design platform has been developed recently for the implementation of MPEG-4 and its video applications. The system architecture of that platform is based on Annapolis WildCard-II FPGA platform device.

In the platform, an improved system IP hardware interface controller is specially designed to provide users a way to integrate individual video processing IP block into the platform framework in the purpose of MPEG-4 implementation.

In order to implement the platform system performance testing and analysis, we need integrate an appropriate video processing IP block into the platform, and thus take the conformance test regarding the IP block integration. Because there is a well-developed IP block for MPEG-4 Part 10/AVC DCT and Quantization [25], we will choose to employ that module to be integrated into the platform framework for IP integration and system performance testing purpose.

8.4.3 A Block Diagram of Integration

The block diagram of DCT/Q module integration is illustrated in Figure 81. The architecture is based on WildCard-II FPGA platform device.

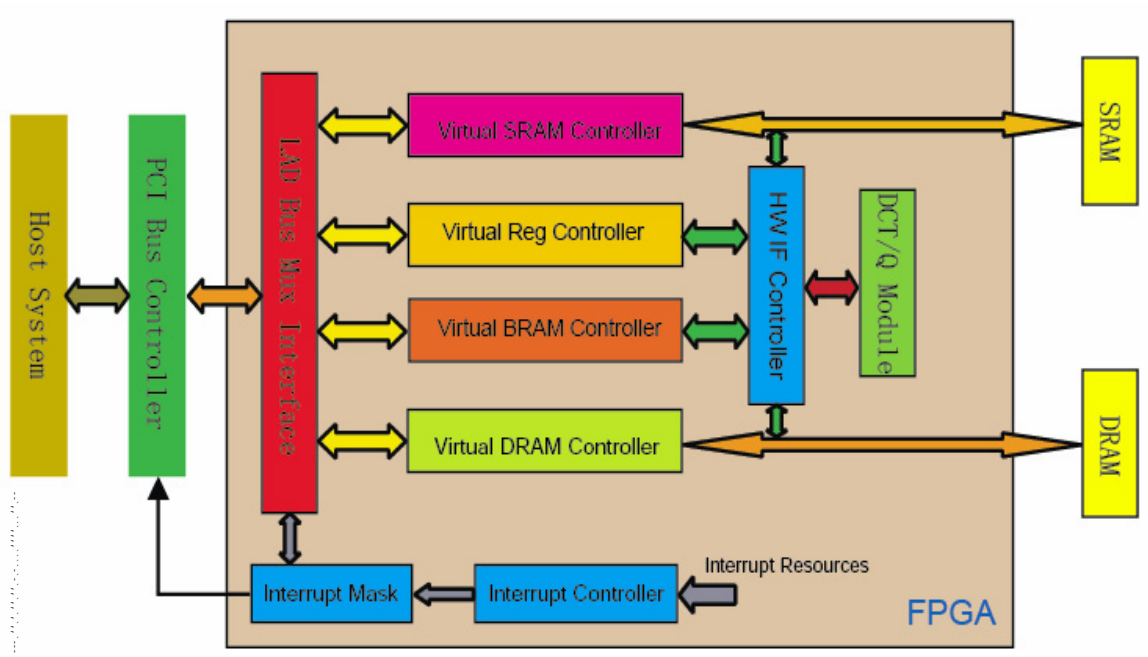


Figure 81 — Architecture Template of DCT/Q Module Integration.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The DCT/Q module will be connected to the platform hardware interface controller. The controller has a set of the interface signals between them to deal with the memory operations required by DCT/Q module.

The architecture of the DCT/Q module [25] allows the memory reading and writing access to be performed separately. Therefore, for this module integration, a parallel module integration structure will be used for memory access to improve the system performance. It is assumed that DCT/Q module will read the data from SRAM and write them back to DRAM. The design of the interface and integration will be prototyped with VHDL.

8.4.4 Module Interface

We have the following interface signals defined between the platform hardware interface controller and DCT/Q module, which are listed in Figure 82.

Interface Signal Descriptions:

clk – the platform system clock input to the DCT/Q module

reset – reset signal for DCT/Q module

start – when the platform triggers DCT/Q module to work, this signal will be asserted

input_valid – when data are ready to be input to the DCT/Q module, this signal is valid

data_in – input data to the DCT/Q module

memory_read – when DCT/Q module needs to read some data from memory space, this signal is asserted for platform to initialize the memory access controllers

```
entity DCT_Quantization is
generic(block_width: positive:=8);
port(
     clk                          : in std_logic;
     reset                        : in std_logic;
     start                        : in std_logic;
     input_valid                  : in std_logic;
     data_in                      : in std_logic_vector(31 downto 0);
     memory_read                  : out std_logic;
     HW_Mem_HWAccel               : out std_logic_vector(31 downto 0);
     HW_Offset                    : out std_logic_vector(31 downto 0);
     HW_Count                     : out std_logic_vector(31 downto 0);
     data_out                     : out std_logic_vector(31 downto 0);
     output_valid                 : out std_logic;
     HW_Mem_HWAccel1              : out std_logic_vector(31 downto 0);
     HW_Offset1                   : out std_logic_vector(31 downto 0);
     HW_Count1                    : out std_logic_vector(31 downto 0);
     output_ack                   : in std_logic;
     Host_Mem_HWAccel             : in std_logic_vector(31 downto 0);
     Host_Offset                  : in std_logic_vector(31 downto 0);
     Host_Count                   : in std_logic_vector(31 downto 0);
     Host_Mem_HWAccel1            : in std_logic_vector(31 downto 0);
     Host_Offset1                 : in std_logic_vector(31 downto 0);
     Host_Count1                  : in std_logic_vector(31 downto 0);
     Mem_Read_Access_Req          : out std_logic;
     Mem_Write_Access_Req         : out std_logic;
     Mem_Read_Access_Release_Req  : out std_logic;
     Mem_Write_Access_Release_Req : out std_logic;
     Mem_Read_Access_Ack          : in std_logic;
     Mem_Write_Access_Ack         : in std_logic;
     Mem_Read_Access_Release_Ack  : in std_logic;
     Mem_Write_Access_Release_Ack : in std_logic;
     done                         : out std_logic
     );
end DCT_Quantization;
```

Figure 82 — DCT/Q Module Interface Signals.

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

HW_Mem_HWAccel – To read data from memory space, DCT/Q module provides the platform this hardware accelerator number for the platform maps the virtual socket address to physical address

HW_Offset – To read data from memory space, DCT/Q module provides the platform this offset for the platform determines the starting address to be accessed

HW_Count – To read data from memory space, DCT/Q module provides the platform this count for the platform determines the data length to be accessed

data_out – output data to the memory space

output_valid – when data are ready to be output to the memory space, this signal is valid

HW_Mem_HWAccel1 – To write data to memory space, DCT/Q module provides the platform this hardware accelerator number for the platform maps the virtual socket address to physical address

HW_Offset1 – To write data to memory space, DCT/Q module provides the platform this offset for the platform determines the starting address to be accessed

HW_Count1 – To write data to memory space, DCT/Q module provides the platform this count for the platform determines the data length to be accessed

output_ack – when platform successfully receives the data from DCT/Q module, this signal is asserted to acknowledge it

Host_Mem_HWAccel – Besides the DCT/Q module itself can provide hardware accelerator number to the platform for memory reading operations, the host system can optionally provide the DCT/Q module with this initial hardware accelerator number. Therefore, if the DCT/Q module doesn't want to generate its own hardware accelerator number to be provided to the platform, it can just accept this parameter signal as its desiring hardware accelerator number for memory reading operation.

Host_Offset – Provided by host system as an optional parameter signal to generate the necessary memory starting offset address for memory reading operation.

Host_Count – Provided by host system as an optional parameter signal to generate the necessary data counter for memory reading operation.

Host_Mem_HWAccel1 – Similar to Host_Mem_HWAccel, if the DCT/Q module doesn't want to generate its own hardware accelerator number to be provided to the platform, it can just accept this parameter signal as its desiring hardware accelerator number for memory writing operation.

Host_Offset1 – Similar to Host_Offset, this signal is provided by host system as an optional parameter to generate the necessary memory starting offset address for memory writing operation.

Host_Count1 – Similar to Host_Count, this signal is provided by host system as an optional parameter to generate the necessary data counter for memory writing operation.

Mem_Read_Access_Req – DCT/Q module issues this signal to request the memory reading.

Mem_Read_Access_Ack – Platform generates this signal to acknowledge the user request.

Mem_Read_Access_Release_Req – When DCT/Q module finishes the memory reading operations, it asserts this signal to ask for releasing the memory reading.

Mem_Read_Access_Release_Ack – Platform generates this signal to acknowledge the user releasing request, and therefore the memory is released for the reading operations.

Mem_Write_Access_Req – DCT/Q module issues this signal to request the memory writing.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Mem_Write_Access_Ack – Platform generates this signal to acknowledge the user request.

Mem_Write_Access_Release_Req – When DCT/Q module finishes the memory writing operations, it asserts this signal to ask for releasing the memory writing.

Mem_Write_Access_Release_Ack – Platform generates this signal to acknowledge the user releasing request, and therefore the memory is released for the writing operations.

done – the DCT/Q module interrupt signal to the platform

8.4.5 Template of the Integration

An example template file to integrate 4 4 DCT/Q module into the platform framework is shown in Figure 83. It is assumed that a whole video frame (luminance value) will be processed by DCT/Q module in this example. The video frame is in the format of CIF 4:2:0, and thus there will be 6336 macroblocks need to be processed.

```
library ieee;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_signed.all;
use IEEE.std_logic_arith.all;

entity DCT_Quantization is
generic(block_width: positive:=8);
port(
   clk                         : in std_logic;
   reset                       : in std_logic;
   start                       : in std_logic;
   input_valid                 : in std_logic;
   data_in                     : in std_logic_vector(31 downto 0);
   memory_read                 : out std_logic;
   HW_Mem_HWAccel              : out std_logic_vector(31 downto 0);
   HW_Offset                   : out std_logic_vector(31 downto 0);
   HW_Count                    : out std_logic_vector(31 downto 0);
   data_out                    : out std_logic_vector(31 downto 0);
   output_valid                : out std_logic;
   HW_Mem_HWAccel1             : out std_logic_vector(31 downto 0);
   HW_Offset1                  : out std_logic_vector(31 downto 0);
   HW_Count1                   : out std_logic_vector(31 downto 0);
   output_ack                  : in std_logic;
   Host_Mem_HWAccel            : in std_logic_vector(31 downto 0);
   Host_Offset                 : in std_logic_vector(31 downto 0);
   Host_Count                  : in std_logic_vector(31 downto 0);
   Host_Mem_HWAccel1           : in std_logic_vector(31 downto 0);
   Host_Offset1                : in std_logic_vector(31 downto 0);
   Host_Count1                 : in std_logic_vector(31 downto 0);
   Mem_Read_Access_Req         : out std_logic;
   Mem_Write_Access_Req        : out std_logic;
   Mem_Read_Access_Release_Req : out std_logic;
   Mem_Write_Access_Release_Req : out std_logic;
   Mem_Read_Access_Ack         : in std_logic;
   Mem_Write_Access_Ack        : in std_logic;
   Mem_Read_Access_Release_Ack : in std_logic;
   Mem_Write_Access_Release_Ack : in std_logic;
   done                        : out std_logic
     );
end DCT_Quantization;
architecture behav of DCT_Quantization is

type memory32bit is array (natural range <>) of std_logic_vector(31 downto 0);
signal input_data_buffer      : memory32bit(0 to 3);
signal output_data_buffer     : memory32bit(0 to 7);
signal user_input_status      : integer range 0 to 31;
signal user_output_status     : integer range 0 to 31;

signal DCT_input_valid        : std_logic;
signal DCT_output_valid       : std_logic;
signal DCT_X00_X03            : signed(31 downto 0);
signal DCT_X10_X13            : signed(31 downto 0);
signal DCT_X20_X23            : signed(31 downto 0);
signal DCT_X30_X33            : signed(31 downto 0);
signal DCT_Z00_Z01            : signed(31 downto 0);
```

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
signal DCT_Z02_Z03            : signed(31 downto 0);
signal DCT_Z10_Z11            : signed(31 downto 0);
signal DCT_Z12_Z13            : signed(31 downto 0);
signal DCT_Z20_Z21            : signed(31 downto 0);
signal DCT_Z22_Z23            : signed(31 downto 0);
signal DCT_Z30_Z31            : signed(31 downto 0);
signal DCT_Z32_Z33            : signed(31 downto 0);

component CoreTrans
  port(
        CLK          : IN     std_logic;
        QP           : IN     signed (5 DOWNTO 0);
        X00          : IN     signed (7 DOWNTO 0);
        X01          : IN     signed (7 DOWNTO 0);
        X02          : IN     signed (7 DOWNTO 0);
        X03          : IN     signed (7 DOWNTO 0);
        X10          : IN     signed (7 DOWNTO 0);
        X11          : IN     signed (7 DOWNTO 0);
        X12          : IN     signed (7 DOWNTO 0);
        X13          : IN     signed (7 DOWNTO 0);
        X20          : IN     signed (7 DOWNTO 0);
        X21          : IN     signed (7 DOWNTO 0);
        X22          : IN     signed (7 DOWNTO 0);
        X23          : IN     signed (7 DOWNTO 0);
        X30          : IN     signed (7 DOWNTO 0);
        X31          : IN     signed (7 DOWNTO 0);
        X32          : IN     signed (7 DOWNTO 0);
        X33          : IN     signed (7 DOWNTO 0);
        input_valid  : IN     std_logic;
        Z00          : OUT    signed (13 DOWNTO 0);
        Z01          : OUT    signed (13 DOWNTO 0);
        Z02          : OUT    signed (13 DOWNTO 0);
        Z03          : OUT    signed (13 DOWNTO 0);
        Z10          : OUT    signed (13 DOWNTO 0);
        Z11          : OUT    signed (13 DOWNTO 0);
        Z12          : OUT    signed (13 DOWNTO 0);
        Z13          : OUT    signed (13 DOWNTO 0);
        Z20          : OUT    signed (13 DOWNTO 0);
        Z21          : OUT    signed (13 DOWNTO 0);
        Z22          : OUT    signed (13 DOWNTO 0);
        X23          : IN     signed (7 DOWNTO 0);
        X30          : IN     signed (7 DOWNTO 0);
        X31          : IN     signed (7 DOWNTO 0);
        X32          : IN     signed (7 DOWNTO 0);
        X33          : IN     signed (7 DOWNTO 0);
        input_valid  : IN     std_logic;
        Z00          : OUT    signed (13 DOWNTO 0);
        Z01          : OUT    signed (13 DOWNTO 0);
        Z02          : OUT    signed (13 DOWNTO 0);
        Z03          : OUT    signed (13 DOWNTO 0);
        Z10          : OUT    signed (13 DOWNTO 0);
        Z11          : OUT    signed (13 DOWNTO 0);
        Z12          : OUT    signed (13 DOWNTO 0);
        Z13          : OUT    signed (13 DOWNTO 0);
        Z20          : OUT    signed (13 DOWNTO 0);
        Z21          : OUT    signed (13 DOWNTO 0);
        Z22          : OUT    signed (13 DOWNTO 0);
        Z23          : OUT    signed (13 DOWNTO 0);
        Z30          : OUT    signed (13 DOWNTO 0);
        Z31          : OUT    signed (13 DOWNTO 0);
        Z32          : OUT    signed (13 DOWNTO 0);
        Z33          : OUT    signed (13 DOWNTO 0);
        output_valid : OUT    std_logic
      );
  end component;

begin

  u_myhardware_dct: CoreTrans
  port map(
            CLK          => clk,
            QP           => "000101",
            X00          => DCT_X00_X03(7 downto 0),
            X01          => DCT_X00_X03(15 downto 8),
            X02          => DCT_X00_X03(23 downto 16),
            X03          => DCT_X00_X03(31 downto 24),
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2009 – All rights reserved
Not for Resale

```
            X10          => DCT_X10_X13(7 downto 0),
            X11          => DCT_X10_X13(15 downto 8),
            X12          => DCT_X10_X13(23 downto 16),
            X13          => DCT_X10_X13(31 downto 24),
            X20          => DCT_X20_X23(7 downto 0),
            X21          => DCT_X20_X23(15 downto 8),
            X22          => DCT_X20_X23(23 downto 16),
            X23          => DCT_X20_X23(31 downto 24),
            X30          => DCT_X30_X33(7 downto 0),
            X31          => DCT_X30_X33(15 downto 8),
            X32          => DCT_X30_X33(23 downto 16),
            X33          => DCT_X30_X33(31 downto 24),
            input_valid  => DCT_input_valid,
            Z00          => DCT_Z00_Z01(13 downto 0),
            Z01          => DCT_Z00_Z01(29 downto 16),
            Z02          => DCT_Z02_Z03(13 downto 0),
            Z03          => DCT_Z02_Z03(29 downto 16),
            Z10          => DCT_Z10_Z11(13 downto 0),
            Z11          => DCT_Z10_Z11(29 downto 16),
            Z12          => DCT_Z12_Z13(13 downto 0),
                         Z13          => DCT_Z12_Z13(29 downto 16),
            Z20          => DCT_Z20_Z21(13 downto 0),
            Z21          => DCT_Z20_Z21(29 downto 16),
            Z22          => DCT_Z22_Z23(13 downto 0),
            Z23          => DCT_Z22_Z23(29 downto 16),
            Z30          => DCT_Z30_Z31(13 downto 0),
            Z31          => DCT_Z30_Z31(29 downto 16),
            Z32          => DCT_Z32_Z33(13 downto 0),
            Z33          => DCT_Z32_Z33(29 downto 16),
            output_valid => DCT_output_valid
          );

process(reset,clk)
  variable i : integer range 0 to 1024;
  variable j : integer range 0 to 1024;
  variable DCT_Block : integer range 0 to 32767;
  variable temp0,temp1,temp2,temp3 : std_logic_vector(31 downto 0);
  begin
    if (reset = '1') then
      i := 0;
      j := 0;
      DCT_Block := 0;
      user_input_status <= 0;
      memory_read <= '0';
      DCT_input_valid <= '0';
      Mem_Read_Access_Req <= '0';
      Mem_Read_Access_Release_Req <= '0';
elsif rising_edge(clk) then

      if (start = '1') then
        case user_input_status is
        when 0 =>
          memory_read <= '0';
          Mem_Read_Access_Req <= '0';
          Mem_Read_Access_Release_Req <= '0';
          user_input_status <= 1;
        when 1 =>
          Mem_Read_Access_Req <= '1';
          user_input_status <= 2;
        when 2 =>
          if (Mem_Read_Access_Ack = '1') then
            Mem_Read_Access_Req <= '0';
            user_input_status <= 3;
          else
            user_input_status <= 2;
      end if;
        when 3 =>
          HW_Mem_HWAccel <= x"00000000";
          HW_Offset <= x"00000000";
          HW_Count <= x"00000004";
          DCT_Block := 6336;
          j := 0;
      user_input_status <= 4;
        when 4 =>
          memory_read <= '1';
      user_input_status <= 5;
```

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
        when 5 =>
          memory_read <= '0';
          if (input_valid = '1') then
            input_data_buffer(j) <= data_in;
            HW_Offset <= HW_Offset + 1;
            j := j + 1;
            if (j >= conv_integer(unsigned(HW_Count))) then
              j := 0;
              temp0 := input_data_buffer(0);
              temp1 := input_data_buffer(1);
              temp2 := input_data_buffer(2);
              temp3 := data_in;
              DCT_X00_X03(7 downto 0) <= signed(temp0(7 downto 0));
              DCT_X00_X03(15 downto 8) <= signed(temp0(15 downto 8));
              DCT_X00_X03(23 downto 16) <= signed(temp0(23 downto 16));
              DCT_X00_X03(31 downto 24) <= signed(temp0(31 downto 24));
              DCT_X10_X13(7 downto 0) <= signed(temp1(7 downto 0));
              DCT_X10_X13(15 downto 8) <= signed(temp1(15 downto 8));
              DCT_X10_X13(23 downto 16) <= signed(temp1(23 downto 16));
              DCT_X10_X13(31 downto 24) <= signed(temp1(31 downto 24));
              DCT_X20_X23(7 downto 0) <= signed(temp2(7 downto 0));
              DCT_X20_X23(15 downto 8) <= signed(temp2(15 downto 8));
              DCT_X20_X23(23 downto 16) <= signed(temp2(23 downto 16));
              DCT_X20_X23(31 downto 24) <= signed(temp2(31 downto 24));
              DCT_X30_X33(7 downto 0) <= signed(temp3(7 downto 0));
              DCT_X30_X33(15 downto 8) <= signed(temp3(15 downto 8));
              DCT_X30_X33(23 downto 16) <= signed(temp3(23 downto 16));
              DCT_X30_X33(31 downto 24) <= signed(temp3(31 downto 24));
              DCT_input_valid <= '1';
              user_input_status <= 6;
            else
              user_input_status <= 5;
            end if;
          else
            user_input_status <= 5;
          end if;
        when 6 =>
          DCT_input_valid <= '0';
          DCT_Block := DCT_Block - 1;
          if (DCT_Block = 0) then
            j := 0;
            user_input_status <= 7;
          else
            j := 0;
            user_input_status <= 4;
          end if;
        when 7 =>
          Mem_Read_Access_Release_Req <= '1';
          user_input_status <= 8;
        when 8 =>
          if (Mem_Read_Access_Release_Ack = '1') then
            Mem_Read_Access_Release_Req <= '0';
            user_input_status <= 9;
          else
            user_input_status <= 8;
          end if;
        when 9 =>
          user_input_status <= 9;
        when others => null;
        end case;
      else
        i := 0;
        j := 0;
        user_input_status <= 0;
        memory_read <= '0';
        DCT_input_valid <= '0';
        Mem_Read_Access_Req <= '0';
        Mem_Read_Access_Release_Req <= '0';
      end if;
    end if;
  end process;

process(reset,clk)
  variable i : integer range 0 to 1024;
  variable j : integer range 0 to 1024;
  variable DCT_Block : integer range 0 to 32767;
```


© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
  begin
    if (reset = '1') then
      output_valid <= '0';
      i := 0;
      j := 0;
      user_output_status <= 0;
      Mem_Write_Access_Req <= '0';
      Mem_Write_Access_Release_Req <= '0';
      done <= '1';
elsif rising_edge(clk) then
      if (start = '1') then
        case user_output_status is
        when 0 =>
          Mem_Write_Access_Req <= '1';
      user_output_status <= 1;
        when 1 =>
          if (Mem_Write_Access_Ack = '1') then
            Mem_Write_Access_Req <= '0';
            user_output_status <= 2;
          else
            user_output_status <= 1;
      end if;
        when 2 =>
          if (DCT_output_valid = '1') then
            done <= '0';
            j := 0;
            output_data_buffer(0) <= conv_std_logic_vector(DCT_Z00_Z01,32);
            output_data_buffer(1) <= conv_std_logic_vector(DCT_Z02_Z03,32);
            output_data_buffer(2) <= conv_std_logic_vector(DCT_Z10_Z11,32);
            output_data_buffer(3) <= conv_std_logic_vector(DCT_Z12_Z13,32);
            output_data_buffer(4) <= conv_std_logic_vector(DCT_Z20_Z21,32);
            output_data_buffer(5) <= conv_std_logic_vector(DCT_Z22_Z23,32);
            output_data_buffer(6) <= conv_std_logic_vector(DCT_Z30_Z31,32);
            output_data_buffer(7) <= conv_std_logic_vector(DCT_Z32_Z33,32);
            user_output_status <= 3;
      end if;
        when 3 =>
          data_out <= output_data_buffer(j);
          output_valid <= '1';
          user_output_status <= 4;
        when 4 =>
          j := j + 1;
          HW_Offset1 <= HW_Offset1 + 1;
          if (j >= conv_integer(unsigned(HW_Count1))) then
            output_valid <= '0';
            j := 0;
            i := 0;
            DCT_Block := DCT_Block - 1;
            if (DCT_Block = 0) then
              user_output_status <= 5;
            else
              user_output_status <= 2;
            end if;
          else
            data_out <= output_data_buffer(j);
            output_valid <= '1';
            user_output_status <= 4;
          end if;
        when 5 =>
          Mem_Write_Access_Release_Req <= '1';
          user_output_status <= 6;
        when 6 =>
          if (Mem_Write_Access_Release_Ack = '1') then
            Mem_Write_Access_Release_Req <= '0';
            user_output_status <= 7;
          else
            user_output_status <= 6;
          end if;
        when 7 =>
          done <= '1';
          user_output_status <= 7;
        when others => null;
        end case;
      else
        DCT_Block := 6336;
        output_valid <= '0';
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
        i := 0;
        j := 0;
        user_output_status <= 0;
        Mem_Write_Access_Req <= '0';
        Mem_Write_Access_Release_Req <= '0';
        done <= '1';
        HW_Mem_HWAccel1 <= x"00000000";
        HW_Offset1 <= x"00000000";
        HW_Count1 <= x"00000008";
      end if;
    end if;
  end process;
end behav;
```

Figure 83 — Integration of the DCT/Q module.

The data flow of the DCT/Q module is as the following:

(1) The host will issue appropriate virtual socket API function calls to set up and trigger the DCT/Q module to work. Particularly, the host will set up SRAM as the source memory space for module data reading and DRAM as the destination memory space for module data writing operations.

(2) The DCT/Q module issues the request for memory reading. Once the request is granted, the module will read 4 4 macroblock data, i.e 4 Dwords, from SRAM. When the reading access is done, it will issue the request to release the memory reading.

(3) The video data will be processed by DCT/Q module. When the data processing of a macroblock is done, the DCT/Q module is ready to output the data to memory space.

(4) The DCT/Q module issues the request for memory writing. Once the request is granted, the module will write the result data, i.e 8 Dwords, to DRAM. When the writing access is finished, it will issue the request to release the memory writing.

(5) If other macroblocks need processing, go back to step 2. Otherwise, the DCT/Q module will generate an interrupt signal to the platform to indicate the finishing of data processing.

(6) The host will check this interrupt, and therefore read corresponding result data from DRAM back to the host memory.

8.4.6 Integration of the User IP blocks

To start working of the DCT/Q module, the following virtual socket API function calls and Annapolis WildCard-II API function calls [5] are needed:

(1) VSInit

(2) VSDMAInit

(3) VSDMAWrite

(4) VSDMARead

(5) VSDMAClear

(6) HW_Module_Controller

(7) WC_Open

(8) WC_Close

(9) WC_PeRegWrite

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

To set up the appropriate memory spaces for DCT/Q module reading and writing, the parameter "Memory_Type" should be as the following:

Memory_Type[31:30]	Memory Space Accessed
1 0	SRAM
Memory_Type[29:28]	**Memory Space Accessed**
1 1	DRAM

Figure 84 — Parameter for Memory Access.

The control of the software data flow:

(1) Open and initialize the platform system, DMA channel and data buffer

(2) Read the video frames and fill out the DMA data buffer

(3) Write frame data to SRAM by DMA transfer

(4) Trigger the DCT/Q module to work

(5) Wait and check the platform interrupt generation

(6) Read data from DRAM back to the host memory

(7) Other possible processing or operations, then clean up the DMA channel and data buffer, close the platform

The example software codes are listed in Figure 85.

```
int main(int argc, char *argv[])
{
//necessary variable definition and initialization

rc = WC_Open (TestInfo.DeviceNum, 0);
rc = VSInit (&TestInfo);
Count = CIF_DWORD_SIZE;
rc = VSDMAInit (&TestInfo, Count);

//open a video frame file, and read data from the file to the DMA data buffer
BSDRAM = 1;
StartAddr = 0;
rc = VSDMAWrite (&TestInfo, BSDRAM, StartAddr, Count, Buffer);

Memory_Type[0] = 0xB0000000;
rc = WC_PeRegWrite (TestInfo.DeviceNum, HW_Mem_HWAccel, 1,
                                    Memory_Type);
HW_IP_Start = 0x00000001;
rc = HW_Module_Controller (&TestInfo, HW_IP_Start);

BSDRAM = 2;
StartAddr = 0;
rc = VSDMARead (&TestInfo, BSDRAM, StartAddr, 2*Count, Buffer);

//other possible processing or operations

rc = VSDMAClear (&TestInfo);
rc = WC_Close (TestInfo.DeviceNum);

return rc;

}
```

Figure 85 — An Example of the DCT/Q Software Codes.

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.4.7 Simulation Tools and Synthesis File

The architecture of the platform system and DCT/Q module are prototyped with VHDL. They are simulated using ModelSim 5.4e®, synthesized by Synplify 7.6®, and targeting a FPGA (XC2V3000fg676-4) from the Virtex-II family of Xilinx® based on the WildCard-IITM FPGA device.

The following is a synthesis file for implementation of the whole virtual socket platform system and integration of the DCT/Q module.

```
#-------------------------------------------------------------------------
#-
#-  1. Change the PROJECT_BASE and WILDCARD_BASE variables to your
#-      project directory and WILDCARD-II installation directory,
#-      respectively:
#-
#-------------------------------------------------------------------------
set ANNAPOLIS_BASE "c:/annapolis"
set PROJECT_BASE  "$ANNAPOLIS_BASE/wildcard_ii/examples/virtual_socket/vhdl/sim"

#
# Do not modify these lines!
#
set WILDCARD_BASE   "$ANNAPOLIS_BASE/wildcard_ii"
set PE_BASE         "$WILDCARD_BASE/vhdl/pe"
set SIMULATION_BASE "$WILDCARD_BASE/vhdl/simulation"
set TOOLS_BASE      "$WILDCARD_BASE/vhdl/tools"
set SYNTHESIS_BASE  "$WILDCARD_BASE/vhdl/synthesis"

source "$SYNTHESIS_BASE/tcl/wildcard_pe.tcl"

#-------------------------------------------------------------------------
#-
#-  2. Add your project's PE architecture VHDL file here, as
#-      well as any VHDL or constraint files on which your PE
#-      design depends:
#-      NOTE: They must be in the order specified below.
#-
#-------------------------------------------------------------------------

add_file -vhdl -lib pe_lib "$PROJECT_BASE/data_type_definition_package.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/bram_dma_source_entarch.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/bram_memory_destination_entarch.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/bram_memory_source_entarch.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/bram_dma_destination_entarch.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/dma_source_entarch.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/memory_destination_entarch.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/memory_source_entarch.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/dma_destination_entarch.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/dram_dma_source_entarch.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/dram_memory_destination_entarch.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/dram_memory_source_entarch.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/dram_dma_destination_entarch.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/User_IP_Block_0.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE1/add_flow.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE1/adder_flow.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE1/addr_flow.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE1/check0_flow.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE1/coretrans_struct.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE1/cx0_struct.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE1/cx_ct_struct.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE1/cx_struct.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE1/dum_struct.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE1/matrixmult_struct.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE1/mreg_flow.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE1/mult_struct.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE1/qpopr_struct.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE1/quantizer_struct.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE1/reg_flow.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE1/shifter_struct.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE1/shr1_untitled.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE1/shr15_untitled.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE1/sub_flow.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE1/subr_flow.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE1/subt_flow.vhd"
```


© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
add_file -vhdl -lib pe_lib "$PROJECT_BASE1/subtractor_flow.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE1/xct_struct.vhd"
add_file -vhdl -lib pe_lib "$PE_BASE/pe_ent.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/virtual_socket_testing.vhd"

#-----------------------------------------------------------------------
#-
#-  3. Change the result file:
#-
#-----------------------------------------------------------------------
project -result_file "virtual_socket_testing.edf"
#-----------------------------------------------------------------------
#-
#-  4. Add or change the following options according to your
#-      project's needs:
#-----------------------------------------------------------------------

# State machine encoding
set_option -default_enum_encoding sequential

# State machine compiler
set_option -symbolic_fsm_compiler false

# Resource sharing
set_option -resource_sharing true

# Clock frequency
set_option -frequency 132.000

# Fan-out limit
set_option -fanout_limit 1000

# Hard maximum fanout limit
set_option -maxfan_hard false
```

8.4.8 Conformance Test

A conformance test will be taken with regard to the DCT/Q module integration and the platform system performance in this case. The testing result is shown in Figure 86.

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

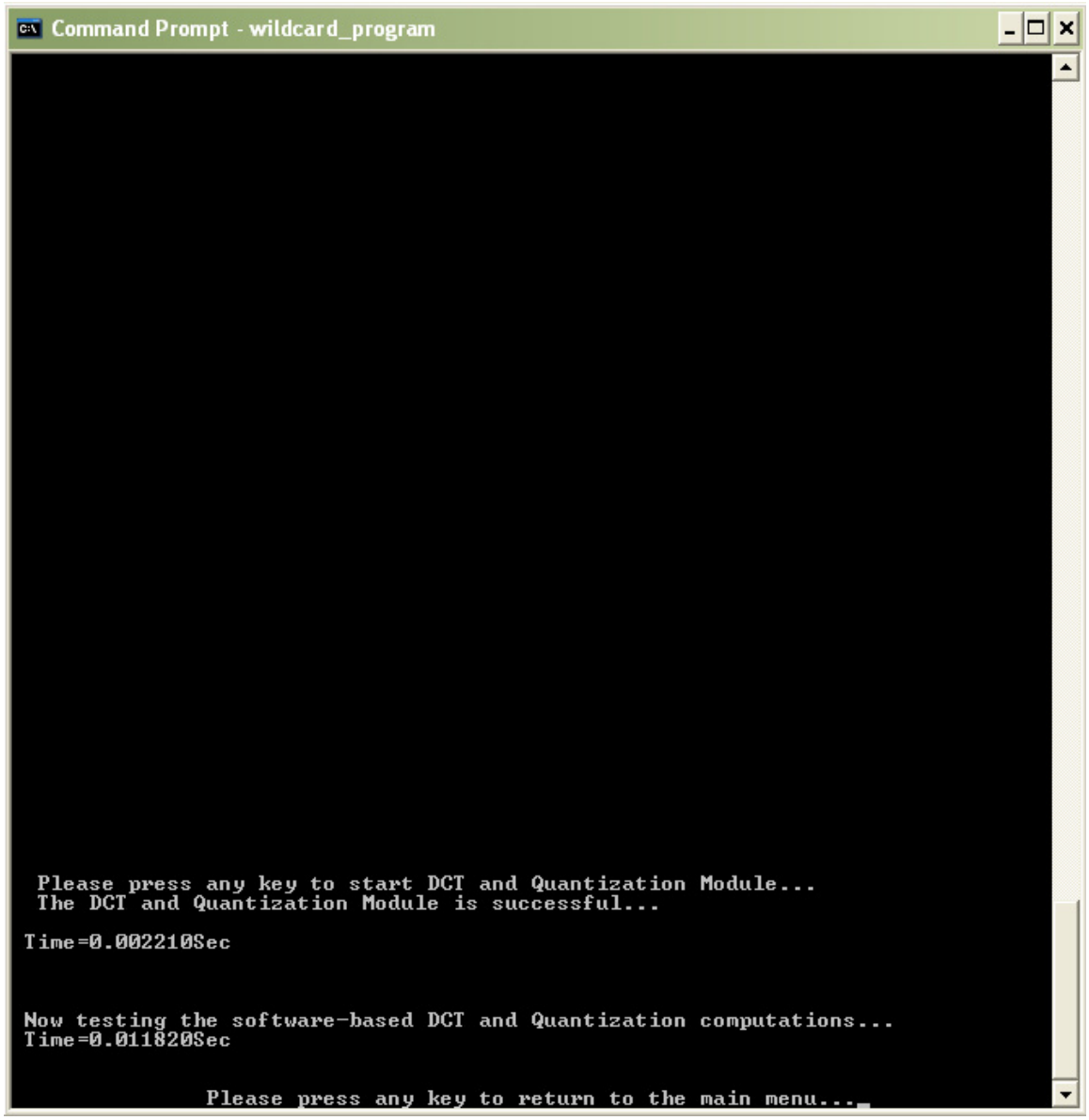


Figure 86 — Conformance Test of DCT/Q Module Integration.

It is assumed that the platform system frequency is 48MHz or 20.83ns for one clock cycle. Normally, it takes the platform two clock cycles for one memory access operation (DWORD based). The architecture of the DCT/Q module allows the memory reading and writing to be performed in parallel, and DCT/Q computation is pipelined. The interface of the DCT/Q module requires at least 4 double words for input and 8 double words for output. Therefore, the critical operation time is determined by the output or 8 2 = 16 system clock cycles for one macroblock data (16 pixels) processing. Generally, it needs the platform to add extra 1 clock cycle for each of the macroblock data processing.

Therefore, the time required for the platform system to encode a whole CIF frame (luminance 352 288 = 101,376 pixels) can be calculated as follows:

Time Required per CIF Frame (Luminance)

=Time per Block Number of Blocks per Frame

=20.83ns (16 + 1)

=2.244ms

--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2009 – All rights reserved
Not for Resale

This result is 14 times less than the 33.3ms standard time (assuming 30 fr/sec) required for MPEG-4 frame encoding.

If we follow the given equations [25] to get DCT and Quantization coefficients based on a pure software computation, the result shows that the software-based solution takes much more time to process a whole video frame than the co-design platform does, even though the host CPU clock frequency, for this example 2.0GHz, is much faster than the hardware system clock.

The testing result which is shown in Figure 86 quite conforms to the theoretical calculations.

8.5 Migrating Virtual Socket Hardware-Accelerated Co-design Platform From WildCard-II to WildCard-4

8.5.1 Scope

This subclause presents a migration of the virtual socket hardware-accelerated platform from WildCard-II to WildCard-4 and implementation results of design resources utilized by the virtual socket co-design platform on WildCard-II and WildCard-4. Through the migration, a previously developed virtual socket platform can be easily expanded or reused without significant design modifications. The migration process actually demonstrates the flexible features for developing a co-design platform.

The point of this subclause is to stress the concept of a system design reuse to get a new generic virtual socket co-design platform for illustrating the possibility of rapidly developing a working system for MPEG-4 video applications.

8.5.2 Introduction

An integrated virtual socket hardware-accelerated co-design platform was previously developed for MPEG-4 video applications. The platform was designed and implemented based on a FPGA device WildCard-II [5]. Through the proposed platform design, user MPEG-4 video IP blocks can be integrated into the platform to accelerate and improve the performance of MPEG-4 video codec application systems.

To integrate more IP blocks into the platform and perform advanced MPEG-4 video codec functions, however, it seems that the WildCard-II cannot meet the platform system requirement because of its limited design resources that can be used. A main solution to this issue is to transplant the previous platform design from WildCard-II to a new platform. Thus, a WildCard-4 FPGA device [5] will be used for accommodating the transplanted platform design.

Actually, this migration process is called platform reuse, which is a major flexible feature for the proposed co-design platform development. With the platform migration and reuse, both of the platform designers and IP block developers can easily reuse their previously well-designed hardware modules and software application programs without significant design modifications. Therefore, the platform can keep all the features that have been developed, leave more design resources to be utilized by user IP blocks, and hence greatly shorten the development and debugging time for a new platform with improved performance.

The platform includes both of the hardware and software design. The hardware platform framework integrates four types of memory controllers, including Register, BRAM, SRAM and DRAM controllers. In some cases, however, users do not need to utilize all of those memory controllers to access the data. Therefore, to reduce the resource utilization of platform framework itself and save more resources for user IP block integrations, the platform framework design needs to be trimmed down. In such case, different platform design cases can be applied for different user IP block integration requirements.

For example, if a user utilizes BRAM to store the video frame data and does not access SRAM or DRAM memory space, only BRAM memory controller is necessary to be integrated into the platform framework. In other word, the platform system design can be split into different functional combinations. Each functional combination can be applied for one user design case.

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.5.3 Previous WildCard-II Platform

The previous virtual socket co-design platform was implemented on WildCard-II. The primary concerns for that platform design were (1) to develop an integrated co-design platform framework which can accommodate MPEG-4 IP blocks, (2) integrate some of the video processing IP blocks into the platform framework, and (3) develop prototyping virtual socket API function calls to implement the platform interface features.

While most of the recent co-design solutions for MPEG-4 video codec implementation have been based on RISC and DSP architectures, the previous co-design platform framework was constructed with a simplified architecture based on the conception of virtual socket. The design of the hardware platform framework was prototyped with VHDL.

The software part of the design was written in C/C++, particularly for the design of virtual socket Application Programming Interface (API) function calls. Because the salient feature of co-design is the cooperation of hardware and software modules, the hardware-software portioning for that platform design was also simplified so that the data flow of the hardware module performance was under the control of host programs.

A system architecture template is depicted in Figure 87, and system functional block diagram is shown in Figure 88.

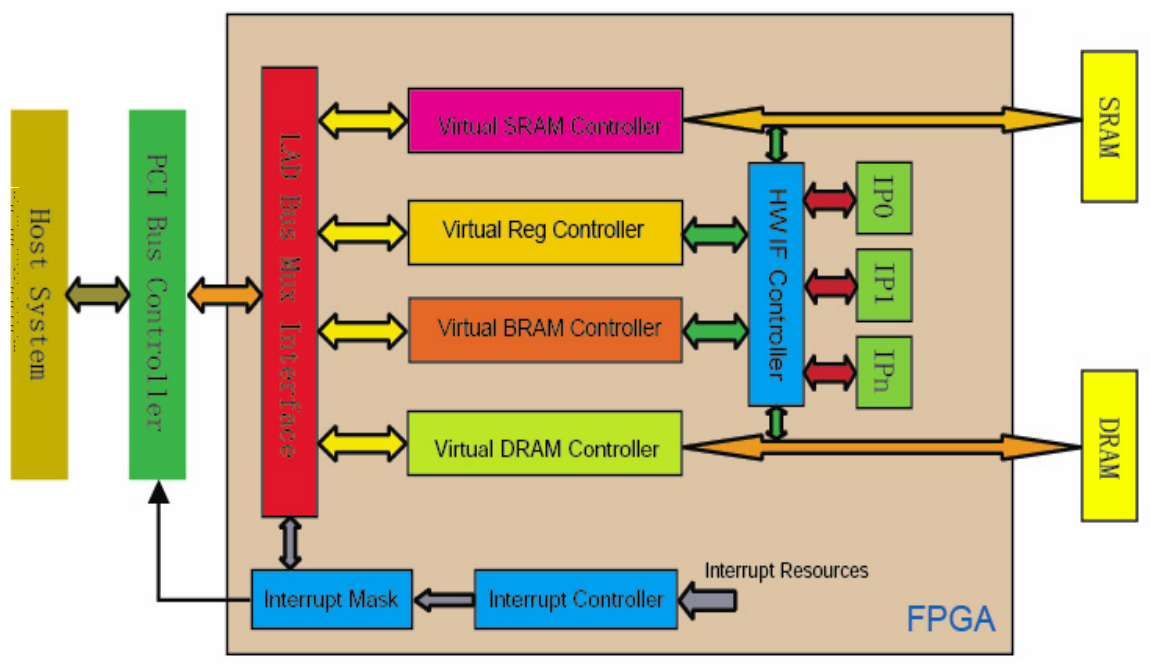


Figure 87 — Virtual Socket Co-design Platform Architecture Template.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2009 – All rights reserved

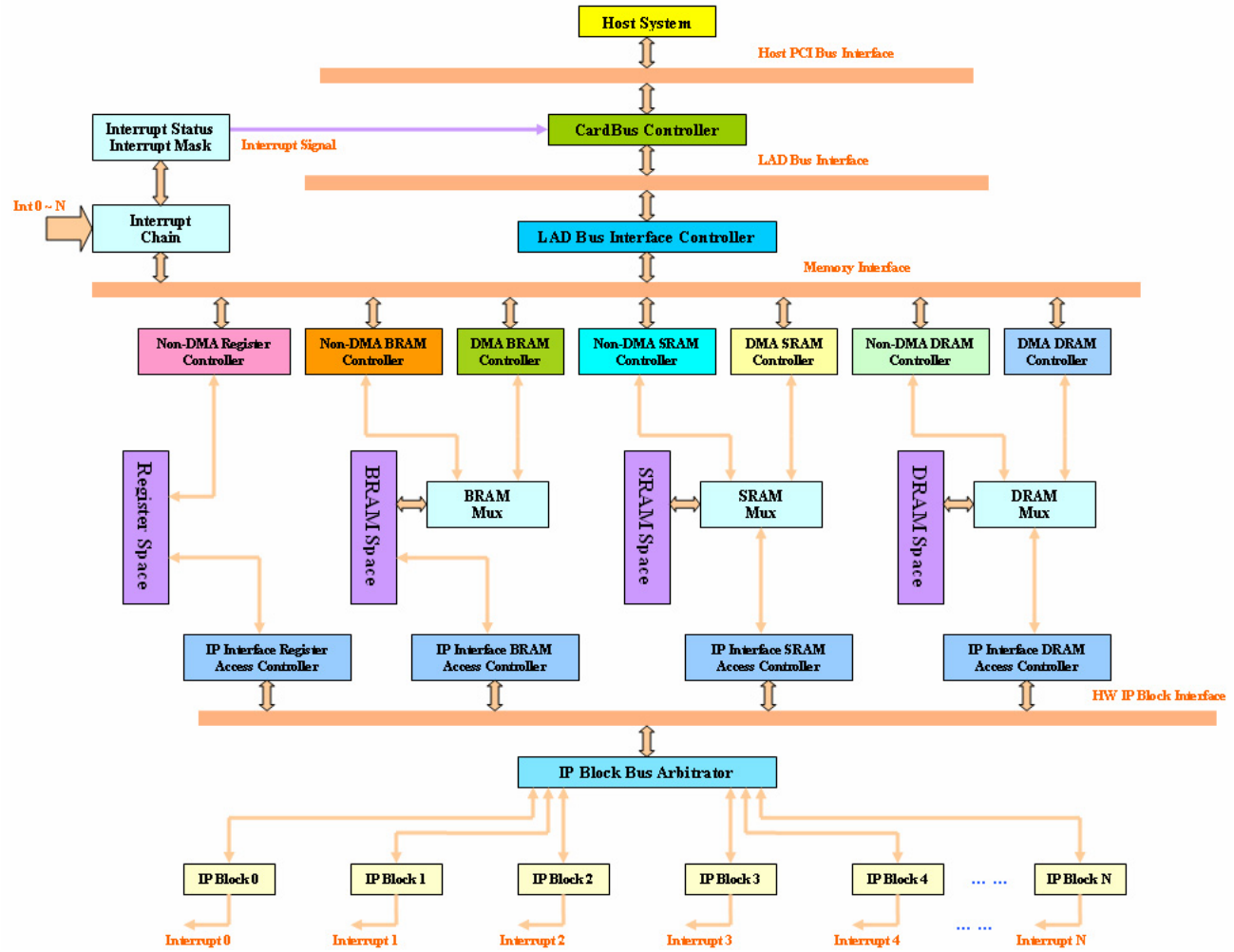


Figure 88 — Platform Functional Block Diagram.

It developed two abstraction layers for system design. One layer is constructed at the hardware level on WildCard-II which performs hardware features for system implementation, and the other is at the software level on host computer system, which deals with a virtual socket interface between the host and hardware platform framework.

The PCI Bus Controller has been developed by Annapolis and is integrated in the WildCard-II. Other than the PCI Bus Controller, the infrastructure for hardware includes FPGA and physical memory spaces. Registers and Block RAM as two types of the memory spaces are integrated in the FPGA as its internal design resources. Other two types of the external memory spaces are ZBT-SRAM and DDR-DRAM. Therefore, there are totally four types of the physical memory spaces inside the platform framework.

Moreover, the hardware functional blocks are generated and integrated in the platform framework to perform data communication and signal control between the host and physical memory spaces. As shown in Figure 87 and Figure 88, each generic virtual memory controller is in charge of the address mapping and access operations for corresponding memory storage. All the user IP blocks are connected to a hardware interface controller, which provides a straight way for the user IP blocks, i.e hardware accelerators, to exchange data between the host system and memory spaces. Interrupt signals can be generated when IP blocks finish the data processing or exchange. In this case, the interrupt controller is designed to deal with the user IP block interrupt generation.

Table 14 and Table 15 list all the hardware and software features developed for that platform system.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 – All rights reserved
155
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 14 — Platform Hardware Features.

Hardware Feature		Description
Non-DMA Data Transfer	Register	Data transfer through Register space by non-DMA mode
	BRAM	Data transfer through Block RAM space by non-DMA mode
	SRAM	Data transfer through SRAM space by non-DMA mode
	DRAM	Data transfer through DRAM space by non-DMA mode
DMA Data Transfer	BRAM	Data transfer through Block RAM space by DMA mode
	SRAM	Data transfer through SRAM space by DMA mode
	DRAM	Data transfer through DRAM space by DMA mode
IP Block Integration		Potentially up to 32 user IP blocks can be integrated into the platform
Concurrent Operation		IP blocks operate in parallel mode if applicable
Extended IP Block Interrupt Handling		An extended interrupt mechanism to deal with multiple IP block interrupt generation

Table 15 — Platform Software Features.

Virtual Socket API Function Call	Description
VSInit	Initialize virtual socket platform
VSSystemReset	Reset virtual socket platform
VSUserResetEnable	Enable a user IP block reset
VSUserResetDisable	Disable a user IP block reset
VSIntEnable	Enable / Disable the interrupt generation for virtual socket platform
VSIntClear	Clear the generated platform interrupt
VSIntCheck	Check the platform interrupt
VSIntStatusWrite	Set up platform interrupt status and mask registers
VSIntStatusRead	Read platform interrupt status and mask registers
VSWrite	Write data from host to virtual socket platform by non-DMA transfer mode
VSRead	Read data from virtual socket platform to host by non-DMA transfer mode
VSDMAInit	Initialize platform DMA transfer channel and data buffer
VSDMAClear	Release platform DMA transfer channel and data buffer
VSDMAWrite	Write data from host to virtual socket platform by DMA transfer mode
VSDMARead	Read data from virtual socket platform to host by DMA transfer mode
HW_Module_Controller	Trigger and control user IP blocks to work

8.5.4 WildCard-II and WildCard-4 FPGA Platforms

To transplant the previous co-design platform from WildCard-II to WildCard-4, we need to know the functional features and differences between those two FPGA platforms. It is true that both WildCard-II and WildCard-4 are developed by Annpolis Micro Systems. The WildCard-II includes a Xilinx Virtex-II FPGA XC2V3000-4-FG676, while WildCard-4 uses a more advanced Xilinx Virtex-4 FPGA XC4VSX35-10-FG668 as its main feature. Table 16 lists their functional features, similarities and differences.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2009 – All rights reserved
Not for Resale

Table 16 — WildCard-II and WildCard-4 Functional Features.

WildCard-II		WildCard-4	
PCMCIA CardBus	8.0 Compliant	**PCMCIA CardBus**	8.0 Compliant
PE	XC2V3000-4	**PE**	XC4VSX35-10
ZBT SRAM	2MB	**ZBT SRAM**	8MB
DDR DRAM	64MB	**DDR DRAM**	128MB
CoreFire Support	YES	**CoreFire Support**	YES
Digital I/O Port	14 bits TTL or 7 bits LVDS	**Digital I/O Port**	14 bits TTL or 7 bits LVDS
A/D Capability	12-bit, 65MHz	**A/D Capability**	12-bit, 65MHz
LAD Bus Clock	48MHz	**LAD Bus Clock**	40MHz
Programmable Clock	24MHz ~ 200MHz	**Programmable Clock**	40MHz ~ 270MHz
Globally Distributed Clocks	16	**Globally Distributed Clocks**	32
Slices	14,336	**Slices**	15,360
Xtreme DSP Slices	N/A	**Xtreme DSP Slices**	192
Block RAM	1,728 Kbits	**Block RAM**	3,456 Kbits
DCM	12	**DCM**	8
PMCD	N/A	**PMCD**	4

From the Table 16, we can know that WildCard-4 has a more advanced performance than WildCard-II, particularly in areas of the memory sizes (SRAM, DRAM and BRAM) and design resources that can be utilized.

8.5.5 Migration from WildCard-II to WildCard-4

The migration process for the proposed platform system is pretty easy, because most of the previous platform VHDL codes can be reused directly on WildCard-4 without modification. Besides, all the software virtual socket API function calls can also be kept without change.

The only modifications are related to the WildCard-4 memory interfaces in the hardware platform framework. The modifications will cover the address mapping of Register and memory interface signal control of SRAM and DRAM.

8.5.5.1 Address Mapping of WildCard-4 Registers

The address mapping of WildCard-II Register space is shown in Table 17. The previous physical address space of the Registers is from 0x0000 to 0x3e0f, and each hardware accelerator occupies a consecutive 16 Dwords Register space.

When migrated to WildCard-4, however, the address mapping of the Registers will be changed. The default WildCard-4 LAD Bus interface has some registers built into the PE design. They are located in a reserved section of memory space from 0x0000 to 0x000f. In such case, the users cannot use the register space from 0x0000 to 0x000f. Therefore, the address mapping of WildCard-4 Register space needs to be relocated, and it will be defined in Table 18. The WildCard-4 physical address space of the Registers will be from 0x1000 to 0x4e0f, and each hardware accelerator still occupies a consecutive 16 Dwords Register space.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 17 — Address Mapping of Registers on WildCard-II.

HWAccel Number	Virtual Address	Physical Address	HWAccel Number	Virtual Address	Physical Address
0	0x0000-0x000f	0x0000-0x000f	16	0x2000-0x200f	0x2000-0x200f
1	0x0200-0x020f	0x0200-0x020f	17	0x2200-0x220f	0x2200-0x220f
2	0x0400-0x040f	0x0400-0x040f	18	0x2400-0x240f	0x2400-0x240f
3	0x0600-0x060f	0x0600-0x060f	19	0x2600-0x260f	0x2600-0x260f
4	0x0800-0x080f	0x0800-0x080f	20	0x2800-0x280f	0x2800-0x280f
5	0x0a00-0x0a0f	0x0a00-0x0a0f	21	0x2a00-0x2a0f	0x2a00-0x2a0f
6	0x0c00-0x0c0f	0x0c00-0x0c0f	22	0x2c00-0x2c0f	0x2c00-0x2c0f
7	0x0e00-0x0e0f	0x0e00-0x0e0f	23	0x2e00-0x2e0f	0x2e00-0x2e0f
8	0x1000-0x100f	0x1000-0x100f	24	0x3000-0x300f	0x3000-0x300f
9	0x1200-0x120f	0x1200-0x120f	25	0x3200-0x320f	0x3200-0x320f
10	0x1400-0x140f	0x1400-0x140f	26	0x3400-0x340f	0x3400-0x340f
11	0x1600-0x160f	0x1600-0x160f	27	0x3600-0x360f	0x3600-0x360f
12	0x1800-0x180f	0x1800-0x180f	28	0x3800-0x380f	0x3800-0x380f
13	0x1a00-0x1a0f	0x1a00-0x1a0f	29	0x3a00-0x3a0f	0x3a00-0x3a0f
14	0x1c00-0x1c0f	0x1c00-0x1c0f	30	0x3c00-0x3c0f	0x3c00-0x3c0f
15	0x1e00-0x1e0f	0x1e00-0x1e0f	31	0x3e00-0x3e0f	0x3e00-0x3e0f

Table 18 — Address Mapping of Registers on WildCard-4.

HWAccel Number	Virtual Address	Physical Address	HWAccel Number	Virtual Address	Physical Address
0	0x0000-0x000f	0x1000-0x100f	16	0x2000-0x200f	0x3000-0x300f
1	0x0200-0x020f	0x1200-0x120f	17	0x2200-0x220f	0x3200-0x320f
2	0x0400-0x040f	0x1400-0x140f	18	0x2400-0x240f	0x3400-0x340f
3	0x0600-0x060f	0x1600-0x160f	19	0x2600-0x260f	0x3600-0x360f
4	0x0800-0x080f	0x1800-0x180f	20	0x2800-0x280f	0x3800-0x380f
5	0x0a00-0x0a0f	0x1a00-0x1a0f	21	0x2a00-0x2a0f	0x3a00-0x3a0f
6	0x0c00-0x0c0f	0x1c00-0x1c0f	22	0x2c00-0x2c0f	0x3c00-0x3c0f
7	0x0e00-0x0e0f	0x1e00-0x1e0f	23	0x2e00-0x2e0f	0x3e00-0x3e0f
8	0x1000-0x100f	0x2000-0x200f	24	0x3000-0x300f	0x4000-0x400f
9	0x1200-0x120f	0x2200-0x220f	25	0x3200-0x320f	0x4200-0x420f
10	0x1400-0x140f	0x2400-0x240f	26	0x3400-0x340f	0x4400-0x440f
11	0x1600-0x160f	0x2600-0x260f	27	0x3600-0x360f	0x4600-0x460f
12	0x1800-0x180f	0x2800-0x280f	28	0x3800-0x380f	0x4800-0x480f
13	0x1a00-0x1a0f	0x2a00-0x2a0f	29	0x3a00-0x3a0f	0x4a00-0x4a0f
14	0x1c00-0x1c0f	0x2c00-0x2c0f	30	0x3c00-0x3c0f	0x4c00-0x4c0f
15	0x1e00-0x1e0f	0x2e00-0x2e0f	31	0x3e00-0x3e0f	0x4e00-0x4e0f

8.5.5.2 SRAM Interface

The previous WildCard-II SRAM interface signals are defined in Table 19.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 19 — WildCard-II SRAM Interface Control Signals.

```
type SRAM_Std_IF_In_Type is record
Data_In      : std_logic_vector(35 downto 0);
Data_Valid   : std_logic;
DMC_Clocked  : std_logic;
end record;

type SRAM_Std_IF_Out_Type record
Addr         : std_logic_vector(20 downto 0);
Data_Out     : std_logic_vector(35 downto 0);
Write        : std_logic;
Strobe       : std_logic;
Sleep        : std_logic;
DCM_Reset    : std_logic;
end record;
```

The WildCard-4 SRAM interface signals are defined in Table 20.

Table 20 — WildCard-4 SRAM Interface Control Signals.

```
type sram_in_type is record
data_in              : std_logic_vector(35 downto 0);
data_in_valid        : std_logic;
end record;

type sram_out_type record
address              : std_logic_vector(20 downto 0);
data_out             : std_logic_vector(35 downto 0);
byte_write_enable_n  : std_logic_vector(3 downto 0);
write                : std_logic;
read                 : std_logic;
sleep                : std_logic;
end record;
```

The “byte_write_enable_n”, “write” and “read” should be set up appropriately on WildCard-4 when previous SRAM interface controllers are migrated. After all, the difference between WildCard-II and WildCard-4 SRAM interfaces is minor, and they can be easily settled.

8.5.5.3 DRAM Interface

The WildCard-II and WildCard-4 DRAM interface signals are defined in Table 21 and Table 22. Although the difference between those interface signals is not significant, a DCM module must be added to the platform to provide WildCard-4 DRAM controller with a synchronous clock to the LAD Bus controller. Without such synchronous clock generated by DCM, the DMA data transfer to and from the DRAM memory space cannot be performed. The synchronous clock is defined in Table 23. This clock is generated with an instantiation of DCM module [6].

Table 21 — WildCard-II DRAM Interface Control Signals.

```
type DRAM_Std_IF_In_Type is record
Data_In      : std_logic_vector(63 downto 0);
Data_Valid   : std_logic;
Ready        : std_logic;
Init_Done    : std_logic;
end record;

type DRAM_Std_IF_Out_Type record
Addr         : std_logic_vector(22 downto 0);
Data_Out     : std_logic_vector(63 downto 0);
Write        : std_logic;
Strobe       : std_logic;
end record;
```

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 22 — WildCard-4 DRAM Interface Control Signals.

```
type dram_in_type is record
data_in              : std_logic_vector(63 downto 0);
data_in_available    : std_logic;
write_rdy            : std_logic;
read_rdy             : std_logic;
end record;

type dram_out_type record
addr                 : std_logic_vector(23 downto 0);
data_out             : std_logic_vector(63 downto 0);
data_in_ce           : std_logic;
write                : std_logic;
read                 : std_logic;
end record;
```

Table 23 — DCM Synchronous Clock for WildCard-4 DDR2 DRAM Interface.

```
User_DCM : AMS_DCM
generic map (
              ATTR_DLL_FREQUENCY_MODE => "Low"
                                )
port map (
           Clock_In                => I_Clk,
           Clock_Out               => I_Clk_i,
           Clock_Out_2x            => open,
           Clock_Out_2x_180        => open,
           Clock_Out_90            => I_Clk_i_90,
           Clock_Out_180           => open,
           Clock_Out_270           => open,
           Clock_Out_DV            => open,
           Clock_FB                => I_Clk_o,
           DSS_Enable              => '0',
           PS_IncDec               => '0',
           PS_Enable               => '0',
           PS_Clk                  => '0',
           PS_Done                 => open,
           Reset                   => Global_Reset,
           Locked                  => open,
           Status                  => open
         );

User_BUFG_1: BUFG
port map (
           I   => I_Clk_i,
           O   => I_Clk_o
         );

User_BUFG_2: BUFG
port map (
           I   => I_Clk_i_90,
           O   => I_Clk_o_90
         );

dram_interface_inst: dram_interface
port map (
           pads             => pads.dram,
           reset            => Global_Reset,
           p_clock.clock    => I_Clk,
           p_clock.clock90  => I_Clk_o_90,
           lad_clock        => I_Clk,
           user_in          => DRAM_In,
           user_out         => DRAM_Out
         );
```


© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.5.6 Resources for WildCard-II Design Cases

8.5.6.1 Register + BRAM

Resources	Utilized	Total on FPGA	Percentage
BUFGMUX	1	16	6%
DCM	1	12	8%
LOCed DCM	1	1	100%
External IOB	43	484	8%
LOCed IOB	43	43	100%
RAMB16	4	96	4%
SLICE	**3626**	**14336**	**25%**
STARTUP	1	1	100%

8.5.6.2 Register + SRAM

Resources	Utilized	Total on FPGA	Percentage
BUFGMUX	2	16	12%
DCM	2	12	16%
LOCed DCM	2	2	100%
External IOB	111	484	22%
LOCed IOB	111	111	100%
RAMB16	1	96	1%
SLICE	**3853**	**14336**	**26%**
STARTUP	1	1	100%

8.5.6.3 Register + DRAM

Resources	Utilized	Total on FPGA	Percentage
BUFGMUX	3	16	18%
DCM	2	12	16%
LOCed DCM	2	2	100%
External IOB	105	484	21%
LOCed IOB	105	105	100%
RAMB16	6	96	6%
SLICE	**5115**	**14336**	**35%**
STARTUP	1	1	100%

8.5.6.4 Register + BRAM + SRAM

Resources	Utilized	Total on FPGA	Percentage
BUFGMUX	2	16	12%
DCM	2	12	16%
LOCed DCM	2	2	100%
External IOB	111	484	22%
LOCed IOB	111	111	100%
RAMB16	5	96	5%
SLICE	**4860**	**14336**	**33%**
STARTUP	1	1	100%

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.5.6.5 Register + BRAM + DRAM

Resources	Utilized	Total on FPGA	Percentage
BUFGMUX	3	16	18%
DCM	2	12	16%
LOCed DCM	2	2	100%
External IOB	105	484	21%
LOCed IOB	105	105	100%
RAMB16	10	96	10%
SLICE	**6144**	**14336**	**42%**
STARTUP	1	1	100%

8.5.6.6 Register + BRAM + SRAM + DRAM

Resources	Utilized	Total on FPGA	Percentage
BUFGMUX	4	16	25%
DCM	3	12	25%
LOCed DCM	3	3	100%
External IOB	173	484	35%
LOCed IOB	173	173	100%
RAMB16	11	96	11%
SLICE	**7566**	**14336**	**52%**
STARTUP	1	1	100%

8.5.7 Resources for WildCard-4 Design Cases

8.5.7.1 Register + BRAM

Resources	Utilized	Total on FPGA	Percentage
BUFG	1	32	3%
ILOGIC	37	448	8%
External IOB	43	448	9%
LOCed IOB	43	43	100%
OLOGIC	38	448	8%
RAMB16	4	192	2%
SLICE	**3720**	**15360**	**24%**
SLICEM	64	7680	1%
STARTUP	1	1	100%

8.5.7.2 Register + SRAM

Resources	Utilized	Total on FPGA	Percentage
BUFG	1	32	3%
ILOGIC	69	448	15%
External IOB	109	448	24%
LOCed IOB	109	109	100%
OLOGIC	103	448	22%
RAMB16	1	192	1%
SLICE	**3885**	**15360**	**25%**
SLICEM	64	7680	1%
STARTUP	1	1	100%

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.5.7.3 Register + DRAM

Resources	Utilized	Total on FPGA	Percentage
BUFG	4	32	12%
DCM_ADV	2	8	25%
IDELAYCTRL	4	16	25%
LOCed IDELAYCTRL	4	4	100%
ILOGIC	71	448	15%
External IOB	112	448	25%
LOCed IOB	112	112	100%
OLOGIC	101	448	22%
RAMB16	1	192	1%
SLICE	**4817**	**15360**	**31%**
SLICEM	304	7680	3%
STARTUP	1	1	100%

8.5.7.4 Register + BRAM + SRAM

Resources	Utilized	Total on FPGA	Percentage
BUFG	1	32	3%
ILOGIC	69	448	15%
External IOB	109	448	24%
LOCed IOB	109	109	100%
OLOGIC	103	448	22%
RAMB16	5	192	2%
SLICE	**4896**	**15360**	**31%**
SLICEM	108	7680	1%
STARTUP	1	1	100%

8.5.7.5 Register + BRAM + DRAM

Resources	Utilized	Total on FPGA	Percentage
BUFG	4	32	12%
DCM_ADV	2	8	25%
IDELAYCTRL	4	16	25%
LOCed IDELAYCTRL	4	4	100%
ILOGIC	71	448	15%
External IOB	112	448	25%
LOCed IOB	112	112	100%
OLOGIC	101	448	22%
RAMB16	5	192	2%
SLICE	**5988**	**15360**	**38%**
SLICEM	244	7680	3%
STARTUP	1	1	100%

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.5.7.6 Register + BRAM + SRAM + DRAM

Resources	Utilized	Total on FPGA	Percentage
BUFG	4	32	12%
DCM_ADV	2	8	25%
IDELAYCTRL	4	16	25%
LOCed IDELAYCTRL	4	4	100%
ILOGIC	103	448	22%
External IOB	178	448	39%
LOCed IOB	178	178	100%
OLOGIC	166	448	37%
RAMB16	6	192	3%
SLICE	**7245**	**15360**	**47%**
SLICEM	218	7680	2%
STARTUP	1	1	100%

Note:

– The WildCard-II and WildCard-4 platform design cases are synthesized by Synplify 8.1 and configured by Xilinx ISE 8.1i SP3. The utilized resources shown above are generated by Xilinx post-synthesis tools.

– All design cases have successfully passed the conformance tests.

– When Register or BRAM is employed, 16 Dwords × 3 = 48 Dwords (Register space), or 512 Dwords × 3 = 1.5K Dwords (BRAM space) are pre-defined in the platform. Those pre-defined memory spaces consume part of the utilized resources listed above.

– No user IP block is integrated in the platform for all cases. When user module needs to be integrated, an extra 6% ~ 10% of the resources will be consumed for each IP wrapper due to the IP interface interactions, IP Bus Arbitrator, Interrupt control etc.

8.5.8 Simulation Tools and Synthesis File

After the migration, the new platform design is simulated using ModelSim 5.4e®, synthesized by Synplify 8.1®, and targeting a FPGA (XC4VSX35-10-FG668) from the Virtex-4 family of Xilinx® based on the WildCard-4TM FPGA device.

The following is a synthesis file for implementation of the whole virtual socket platform system.

```
##############################################################################
#   1. Change the ANNAPOLIS_BASE variable to WILDCARD 4(tm)
#       installation directory.
##############################################################################
set ANNAPOLIS_BASE  "c:/Annapolis_IV"

##############################################################################
#   2. Change the PROJECT_BASE variable to your project directory.
##############################################################################
set PROJECT_BASE    "c:/Annapolis_IV/wildcard4/examples/virtual_socket_testing/vhdl/sim"

##############################################################################
#   Do not modify these lines.
##############################################################################
set WILDCARD4_BASE  "$ANNAPOLIS_BASE/wildcard4"
set SCRIPT_BASE     "$WILDCARD4_BASE/scripts/synthesis"
source              "$SCRIPT_BASE/wildcard4.tcl"

##############################################################################
#   3. Add your project's PE architecture VHDL file here, as
#       well as any VHDL or constraint files on which your PEX
#       design depends:
##############################################################################
```


© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
add_file -vhdl -lib WILDCARD4_LIB "$PROJECT_BASE/data_type_definition_package.vhd"
add_file -vhdl -lib WILDCARD4_LIB "$PROJECT_BASE/fifo_tools_package.vhd"
add_file -vhdl -lib WILDCARD4_LIB "$PROJECT_BASE/bram_dma_source_entarch.vhd"
add_file -vhdl -lib WILDCARD4_LIB "$PROJECT_BASE/bram_memory_destination_entarch.vhd"
add_file -vhdl -lib WILDCARD4_LIB "$PROJECT_BASE/bram_memory_source_entarch.vhd"
add_file -vhdl -lib WILDCARD4_LIB "$PROJECT_BASE/bram_dma_destination_entarch.vhd"
add_file -vhdl -lib WILDCARD4_LIB "$PROJECT_BASE/dma_source_entarch.vhd"
add_file -vhdl -lib WILDCARD4_LIB "$PROJECT_BASE/memory_destination_entarch.vhd"
add_file -vhdl -lib WILDCARD4_LIB "$PROJECT_BASE/memory_source_entarch.vhd"
add_file -vhdl -lib WILDCARD4_LIB "$PROJECT_BASE/dma_destination_entarch.vhd"
add_file -vhdl -lib WILDCARD4_LIB "$PROJECT_BASE/dram_dma_source_entarch.vhd"
add_file -vhdl -lib WILDCARD4_LIB "$PROJECT_BASE/dram_memory_destination_entarch.vhd"
add_file -vhdl -lib WILDCARD4_LIB "$PROJECT_BASE/dram_memory_source_entarch.vhd"
add_file -vhdl -lib WILDCARD4_LIB "$PROJECT_BASE/dram_dma_destination_entarch.vhd"
add_file -vhdl -lib WILDCARD4_LIB "$PROJECT_BASE/User_IP_Block_0.vhd"
add_file -vhdl -lib WILDCARD4_LIB "$PROJECT_BASE/User_IP_Block_1.vhd"
add_file -vhdl -lib WILDCARD4_LIB "$PROJECT_BASE/User_IP_Block_2.vhd"
add_file -vhdl -lib WILDCARD4_LIB "$PROJECT_BASE/fifoctlr_ic_v2.vhd"
add_file -vhdl -lib WILDCARD4_LIB "$PROJECT_BASE/virtual_socket_testing.vhd"

################################################################################
#   4. Change the result file name.
################################################################################
project -result_file "virtual_socket_testing.edf"

################################################################################
#   5. Add or change the following options according to your
#       project's needs:
################################################################################
# FPGA part type
set_option -technology VIRTEX4
set_option -part XC4VSX35
set_option -package FF668

# FPGA speed grade
set_option -speed_grade -10

# State machine encoding
set_option -default_enum_encoding sequential

# State machine compiler
set_option -symbolic_fsm_compiler false

# I/O Insertion
set_option -disable_io_insertion true

# Resource sharing
set_option -resource_sharing true

# Clock frequency
set_option -frequency 200.000

# Fan-out limit
set_option -fanout_limit 1000

# Hard maximum fanout limit
set_option -maxfan_hard false

# Turn off auto constrain
set_option -auto_constrain_io 0

#automatic place and route (vendor) options
set_option -write_apr_constraint 0
```

8.5.9 Conformance Tests

Because the previously developed conformance testing debug menu can be kept and reused without much modifications (when compiling the software programs, only some header and library files need to be replaced by WildCard-4 related files), the users can perform the testing for new platform very easily.

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

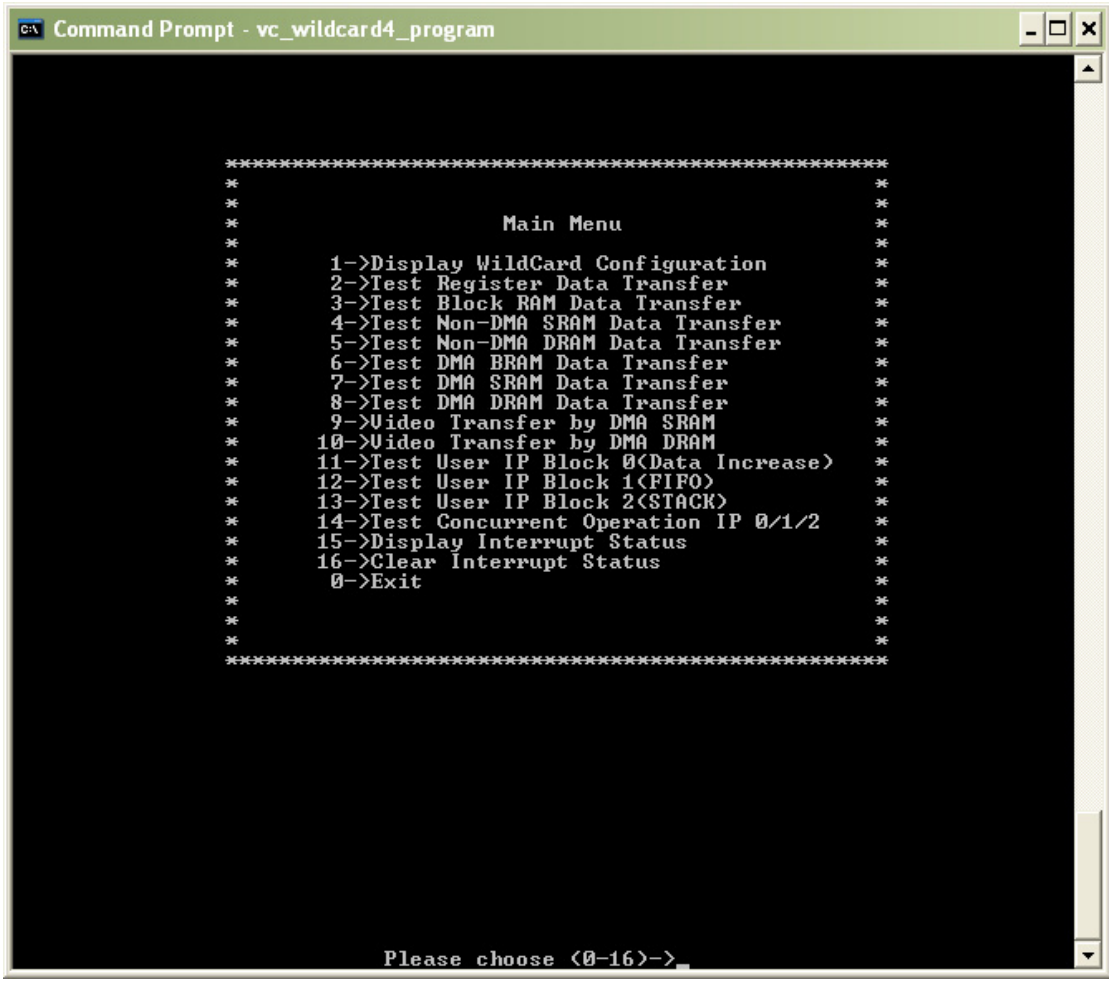


Figure 89 — Debug Menu for WildCard-4 Platform.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9 Integrated Framework supporting the "Virtual Socket" between HDL modules described in Part 9 and the MPEG Reference Software: Implementation 4 - Virtual Memory Extension

9.1 Introduction

The Virtual Memory Extension has been created to aid in the rapid emulation, test, and verification of FPGA designs for complex multimedia standards. It addresses the data communications and controls signal exchanges between a host computer system and a reconfigurable FPGA co-processor based on the concept of virtual socket.

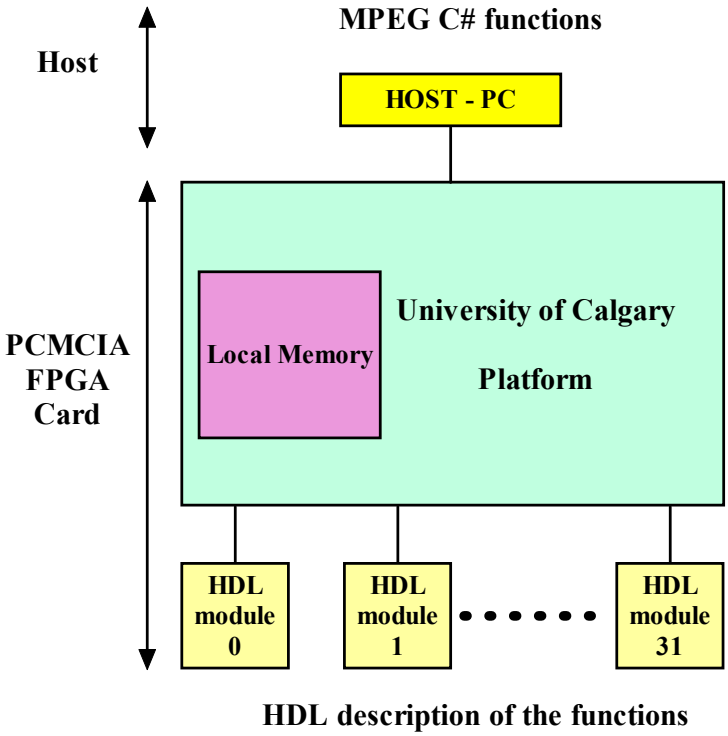


Figure 90 — Relations between the Host, the UoC platform and HDL IP-Blocks.

The objective of this tutorial clause is to address the following issues about an integrated virtual socket platform system developed at the University of Calgary and Ecole Polytechnique Fédérale de Lausanne (EPFL) for the integration of hardware modules with the MPEG4 reference software.

The document is formatted in the form of a tutorial that makes the readers get basic knowledge about an integrated virtual socket platform system and takes them step by step through a typical module development case.

9.2 Overview of the "Virtual Socket Platform" implementation 4

9.2.1 Hardware support

The hardware support is the **Annapolis WildCard-II (WCII)** which is actually a PCMCIA-card that contains a Xilinx FPGA (xc2v3000-fg676) and additional circuits to deal with the PCI bus and to provide some external memory. In particular it contains 64Mb of DDR SDRAM, 2Mb of ZBT SRAM and user analog/digital I/O. The communication between the host and the FPGA is done by what is called the Local Address Data bus (LAD).

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

All what we need to know about this LAD is clearly explained in the WCII Reference Manual [5]. The Processing Element (PE) is the Xilinx FPGA : the UoC platform is implemented in this part.

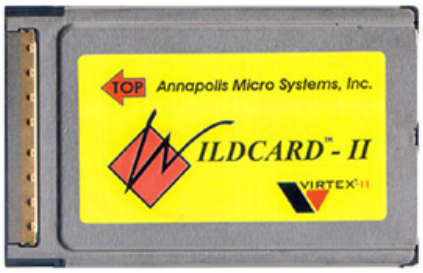


Figure 91 — Visual view of the Annapolis Wildcard II PCMCIA card.

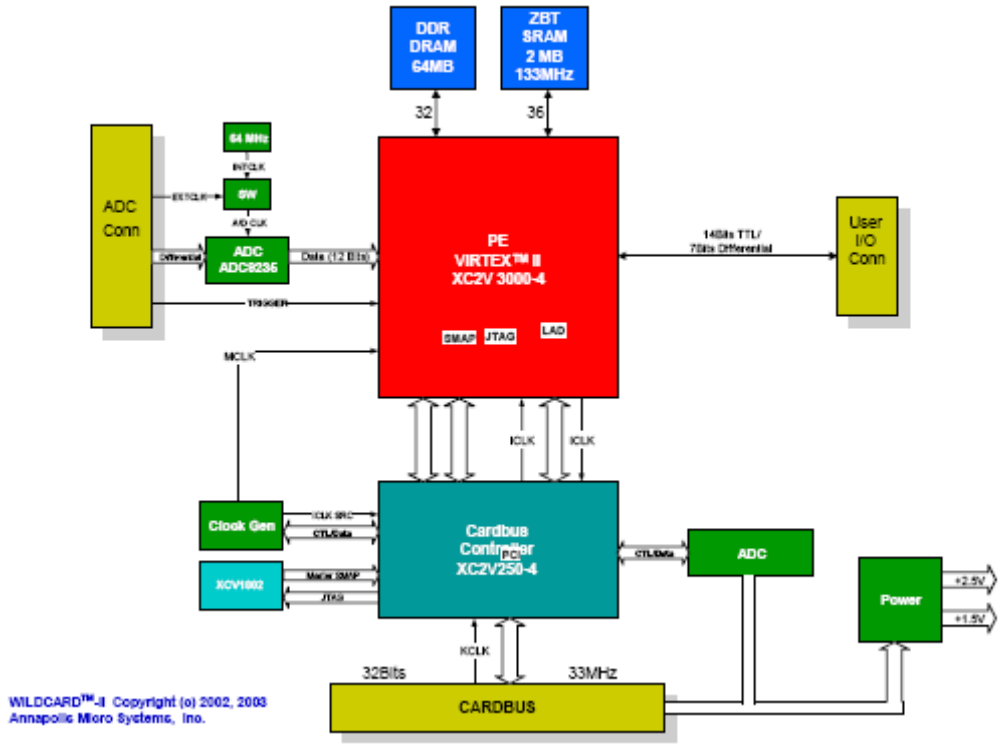


Figure 92 — Architecture of the card Annapolis Wildcard II.

9.2.2 Architecture of the platform

The generic platform system employs four types of memory space to transfer data and thus provide the infrastructure for the purpose of user IP-blocks that are not specifically designed for interaction with a host computer system. Actually, the user IP-block can be any hardware module designed to handle a computationally extensive task for MPEG4 applications.

In a general view, the platform contains **four memory spaces available by both sides: the Host and the IP-Blocks**. There are controllers and a LAD Interface to allow reading / writing data in the local memory from the

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Host. There is a Hardware Interface Controller to send the data from the local memory requested to the IP. The following is a block diagram for this virtual socket platform:

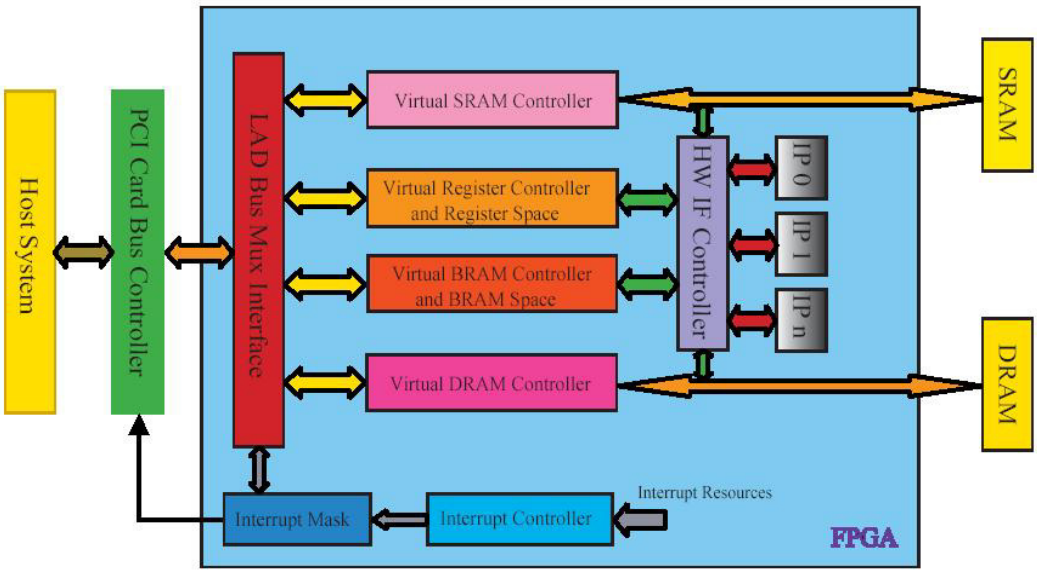


Figure 93 — A block diagram of the virtual socket platform architecture.

LAD Bus Mux Interface, Virtual Register Controller, Virtual BRAM Controller, Virtual SRAM Controller, Virtual DRAM Controller are responsible of transferring data from the Host to the different memory spaces.

HW IF Controller is the interface responsible to send to the IP the data contained in the local memory.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

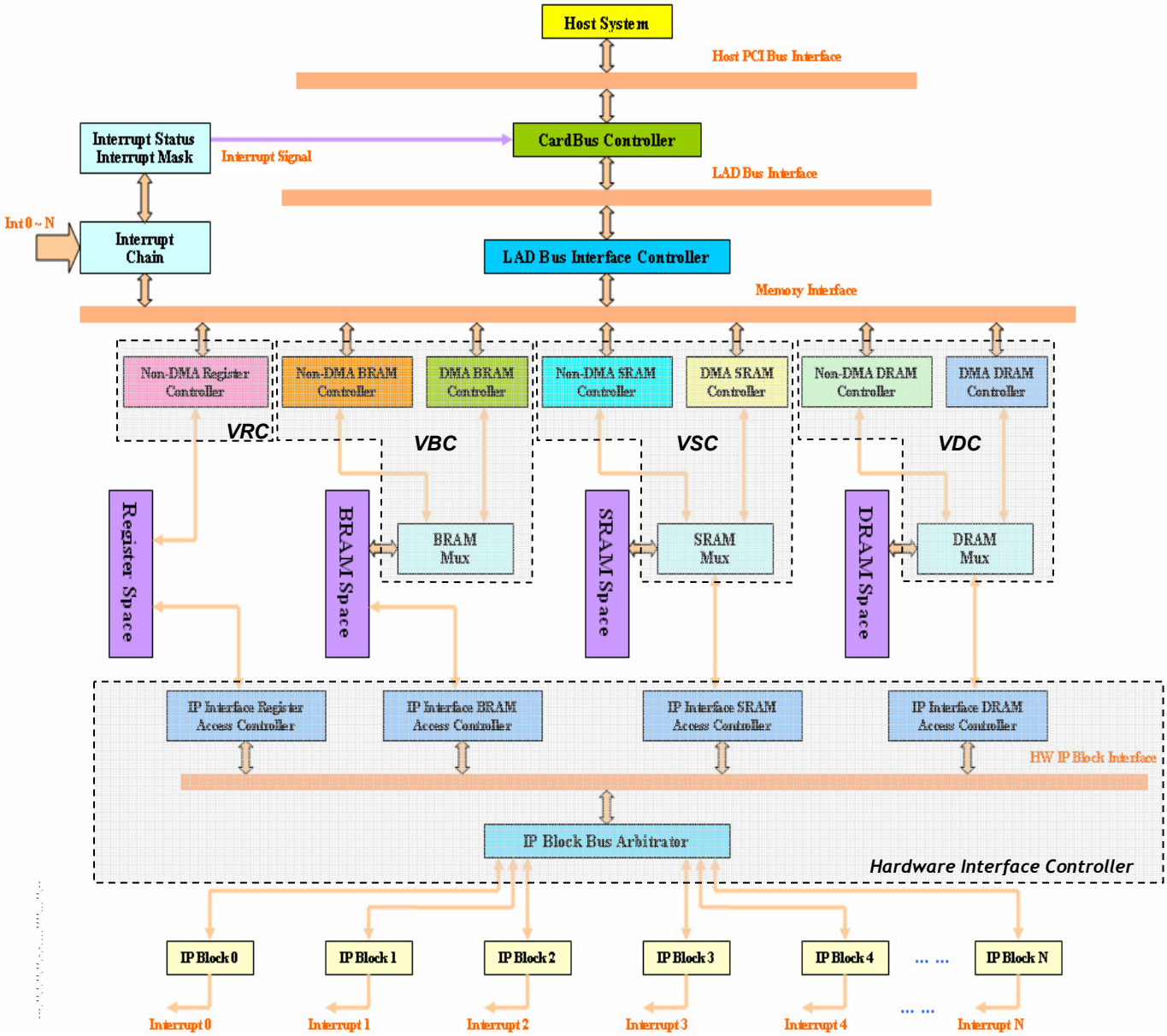


Figure 94 — Precise view of the architecture of the original platform.

Captions:

VRC : Virtual Register Controller
VBC : Virtual BRAM Controller
VSC : Virtual SRAM Controller
VDC : Virtual DRAM Controller

The architecture of the virtual socket platform includes PCI Card Bus Controller, FPGA logic and four types of physical memory space, i.e Registers, BlockRAM, ZBT-SRAM and DDR-DRAM. Two types of the memory space, Registers and BRAM, are integrated in the FPGA as the logic resources. Other two types of the memory space, ZBT-SRAM and DDR-DRAM, are external physical storage. Because this platform system design is constructed on the Wildcard II FPGA device, we can directly utilise its PCI Card Bus Controller which is integrated in the Wildcard II.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Some fundamental hardware module controllers are integrated into the virtual socket platform to perform data communication between host computer system and memory spaces. Those hardware module controllers include virtual register controller, virtual BlockRAM controller, virtual SRAM controller and virtual DRAM controller. Each module is in charge of the data transfer as well as the physical memory address mapping for the corresponding memory storage. Besides, to integrate the user IP-blocks into the whole platform PE system, there is a hardware interface controller. The user MPEG4 compliant IP-blocks (e.g DCT, IDCT, Quatizaton, VLC, ME, MC etc.), are connected to this hardware interface controller. It can provide user IP-blocks required data and IP-blocks process those data and feed them back to the hardware interface controller. After receiving the feed-back data, the hardware interface controller stores the processed data directly into physical memory units. The hardware interface controller provides an easy way for the user IP-blocks to exchange data between the host system and memory spaces. Interrupt signals can be generated when user IP-blocks finishes processing or exchanging the data.

For part of the software, the virtual socket API function calls which are coded in C language are set up in the host system. Developers may provide those functions calls some parameters, and make the virtual socket API function calls work together with the system hardware modules and automatically map the virtual memory addresses into any type of the physical memory space inside the platform. With such mechanism, the data can be transferred between the host computer system and any type of the physical memory space within only one function call. **Anyway, the virtual socket interface is some sort of virtual memory management interface for the hardware-accelerated platform framework.**

To verify the functionality of the platform design, a debug menu is provided and interactively test all the features integrated in the virtual socket platform.

9.2.3 Modes of operations

The HDL designer is free to choose between two modes of operations: the **Explicit Mode** and the **Virtual Mode**.

When an IP-Block requests data, it sends some information to the platform thanks to an interface made of a given number and well-defined signals. Among these signals, there is one called *Virt_mem_access*. If this signal is at low level (‘0’) when the request is sent to the platform, the platform is in the Explicit Mode. At the contrary, if this signal is at high level (‘1’), the platform operates in the Virtual Mode.

9.2.3.1 Explicit mode

The Explicit Mode implies that the **programmer transfers manually the data** requested by the IP from the main memory (host) to the local memory (FPGA). The IP can request data coming from the four types of memory: register space, BRAM, SRAM of DRAM. This mode implies that the addresses given to the IP for asking data are the physical addresses in the local memory.

When asking for data to the platform, the *virt_mem_access* signal must be at low level: *virt_mem_access = ‘0’.*

9.2.3.2 Virtual mode

The Virtual Mode implies that the **programmer doesn’t care about memory management**: the library will transfers automatically the data requested by the IP from the main memory (host) to the local memory (FPGA). In this mode, only register space and BRAM can be used. This mode implies that the addresses given to the IP for asking data are the virtual addresses in the main memory on the host, in the virtual memory space of the program. It was like the IP see directly the main memory of the host (see Figure 94 for illustration).

When asking for data to the platform, the *virt_mem_access* signal must be at high level: *virt_mem_access = ‘1’.*

9.3 Development information

The original platform (Virtual Socket) has been developed by:

University of Calgary (UoC) - Electrical and Computer Engineering

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Laboratory for Integrated Video Systems (LIVS)
Calgary, Alberta, Canada T2N 1N4
Yifeng Qiu and Wael Badawy
E-mail: yiqiu@ucalgary.ca and badawy@ucalgary.ca

The Virtual Memory Extension of the platform has been developed by:

Ecole Polytechnique Fédérale de Lausanne (EPFL) – Informatique et communications
Laboratoire d'Architecture des Processeurs (LAP)
Lausanne, Suisse
Christophe Lucarz and Miljan Vuletić
E-mail: christophe.lucarz@epfl.ch and miljan.vuletic@epfl.ch

9.4 Technical details

9.4.1 Hardware side

9.4.1.1 Memories

The virtual socket platform memory space is mapped to 4 types of physical storage, i.e Registers, Block RAM, ZBT-SRAM and DDR-DRAM. We define a maximum of 32 user hardware accelerators (IP-blocks) which can be integrated into the platform. The data for each user IP-block can be stored in any type of the physical memory space mentioned above (only in Explicit Mode). Currently, we define the size of register space is 16 double words for each user IP-block, BRAM 512 double words for each user IP-block, SRAM 2MB for all the user IP-blocks, and DRAM 32MB for all the user IP-blocks. The size of each memory unit is double word based (32 bits).

- **Address Mapping for Register Space**

Data stored in register space have a maximum size of 16 double words for each user IP-block. Usually, the data stored in register space are some parameters for processing the video data algorithms or hardware modules. Address mapping for register space is listed in the figure below.

HWAccel Number	Virtual Address*	Physical Address	HWAccel Number	Virtual Address*	Physical Address
0	0x0000-0x000f	0x0000-0x000f	16	0x2000-0x200f	0x2000-0x200f
1	0x0200-0x020f	0x0200-0x020f	17	0x2200-0x220f	0x2200-0x220f
2	0x0400-0x040f	0x0400-0x040f	18	0x2400-0x240f	0x2400-0x240f
3	0x0600-0x060f	0x0600-0x060f	19	0x2600-0x260f	0x2600-0x260f
4	0x0800-0x080f	0x0800-0x080f	20	0x2800-0x280f	0x2800-0x280f
5	0x0a00-0x0a0f	0x0a00-0x0a0f	21	0x2a00-0x2a0f	0x2a00-0x2a0f
6	0x0c00-0x0c0f	0x0c00-0x0c0f	22	0x2c00-0x2c0f	0x2c00-0x2c0f
7	0x0e00-0x0e0f	0x0e00-0x0e0f	23	0x2e00-0x2e0f	0x2e00-0x2e0f
8	0x1000-0x100f	0x1000-0x100f	24	0x3000-0x300f	0x3000-0x300f
9	0x1200-0x120f	0x1200-0x120f	25	0x3200-0x320f	0x3200-0x320f
10	0x1400-0x140f	0x1400-0x140f	26	0x3400-0x340f	0x3400-0x340f
11	0x1600-0x160f	0x1600-0x160f	27	0x3600-0x360f	0x3600-0x360f
12	0x1800-0x180f	0x1800-0x180f	28	0x3800-0x380f	0x3800-0x380f
13	0x1a00-0x1a0f	0x1a00-0x1a0f	29	0x3a00-0x3a0f	0x3a00-0x3a0f
14	0x1c00-0x1c0f	0x1c00-0x1c0f	30	0x3c00-0x3c0f	0x3c00-0x3c0f
15	0x1e00-0x1e0f	0x1e00-0x1e0f	31	0x3e00-0x3e0f	0x3e00-0x3e0f

Figure 95 — Address Mapping for Register Space.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

- **Address Mapping for BRAM Space**

Data stored in Block RAM space have a maximum size of 512 double words for each user IP-block. Address mapping for BRAM space is listed in the figure below.

HWAccel Number	Virtual Address*	Physical Address	HWAccel Number	Virtual Address*	Physical Address
0	0x0000-0x01ff	0x8000-0x81ff	16	0x2000-0x21ff	0xa000-0xa1ff
1	0x0200-0x03ff	0x8200-0x83ff	17	0x2200-0x23ff	0xa200-0xa3ff
2	0x0400-0x05ff	0x8400-0x85ff	18	0x2400-0x25ff	0xa400-0xa5ff
3	0x0600-0x07ff	0x8600-0x87ff	19	0x2600-0x27ff	0xa600-0xa7ff
4	0x0800-0x09ff	0x8800-0x89ff	20	0x2800-0x29ff	0xa800-0xa9ff
5	0x0a00-0x0bff	0x8a00-0x8bff	21	0x2a00-0x2bff	0xaa00-0xabff
6	0x0c00-0x0dff	0x8c00-0x8dff	22	0x2c00-0x2dff	0xac00-0xadff
7	0x0e00-0x0fff	0x8e00-0x8fff	23	0x2e00-0x2fff	0xae00-0xafff
8	0x1000-0x11ff	0x9000-0x91ff	24	0x3000-0x31ff	0xb000-0xb1ff
9	0x1200-0x13ff	0x9200-0x93ff	25	0x3200-0x33ff	0xb200-0xb3ff
10	0x1400-0x15ff	0x9400-0x95ff	26	0x3400-0x35ff	0xb400-0xb5ff
11	0x1600-0x17ff	0x9600-0x97ff	27	0x3600-0x37ff	0xb600-0xb7ff
12	0x1800-0x19ff	0x9800-0x99ff	28	0x3800-0x39ff	0xb800-0xb9ff
13	0x1a00-0x1bff	0x9a00-0x9bff	29	0x3a00-0x3bff	0xba00-0xbbff
14	0x1c00-0x1dff	0x9c00-0x9dff	30	0x3c00-0x3dff	0xbc00-0xbdff
15	0x1e00-0x1fff	0x9e00-0x9fff	31	0x3e00-0x3fff	0xbe00-0xbfff

Figure 96 — Address Mapping for BRAM Space.

- **Address Mapping for SRAM Space**

ZBT-SRAM is a large size memory space for video data storage. Actually, each user IP-block can utilize the whole address space of the SRAM for storing the video data. The SRAM size is up to 2M Bytes, i.e. 524,288 double words. Address mapping for SRAM space is listed in the figure below.

HWAccel Number	Virtual Address*	Physical Address
0 – 31	0x00000000-0x0007ffff	0x00000000-0x0007ffff

Figure 97 — Address Mapping for SRAM Space.

- **Address Mapping DRAM Space**

Similar to the ZBT-SRAM space, DDR-DRAM is also a large size memory space for video data storage. Each user IP-block can utilize the whole address space of the DRAM for storing the video data. Currently, the DRAM size can be up to 32M Bytes, i.e. 8,388,608 double words. Address mapping for DRAM space is listed in the figure below.

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

HWAccel Number	Virtual Address*	Physical Address
0 – 31	0x00000000-0x007fffff	0x00000000-0x007fffff

Figure 98 — Address Mapping for DRAM Space.

(*) **Note:** what the designers of the original platform called “virtual address” in these tables has not the same meaning as the “virtual address” in the part concerning Virtual Memory Extension.

When the Virtual Mode is running, the IP-Block generates a virtual address: an address which points to data in the user virtual memory space, the one seen by the program in use on the Host.

At the contrary, “Virtual address” used in these tables (or in the documentation of Virtual Socket, see the references) is still a physical address. This *semi-virtual* address must be put in relation with an other register of the platform (*mem_access)* to get the entire physical address.

9.4.1.2 Virtual memory extension

The aim of the Virtual Memory Extension is to **provide to the IPs the virtual memory abstraction and shared address space with user software.**

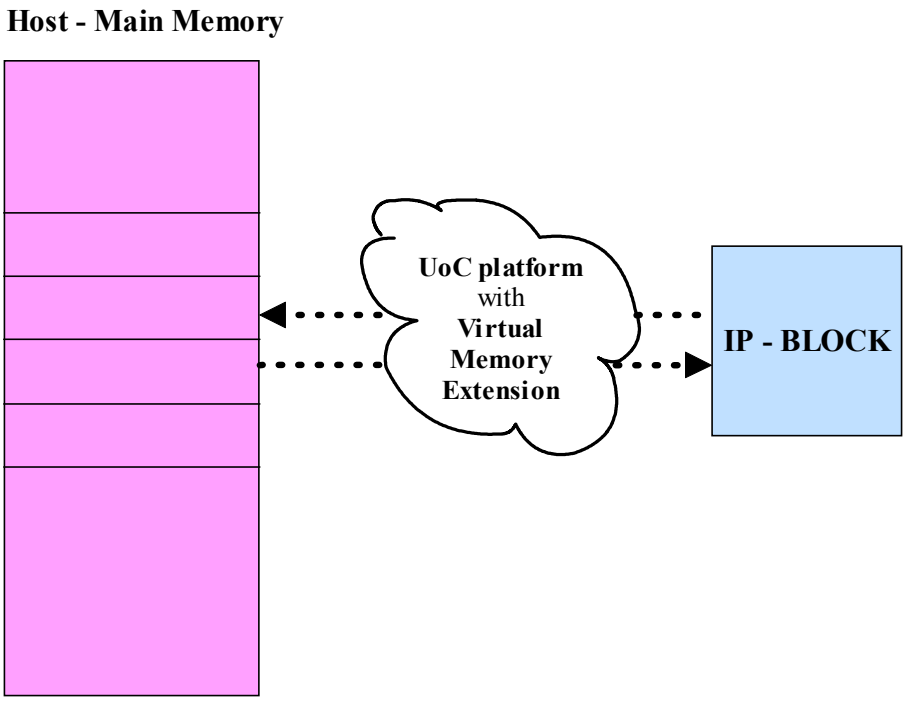


Figure 99 — The IP-Block sees directly the main memory of the host.

This clause provides all the hardware technical details about the Virtual Memory Extension.

To handle such functionality, news components must be added to the original platform: the WMU and the Virtual Memory Controller. How theses components are integrated into the platform and how they communicate with it to reach this functionality ?

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

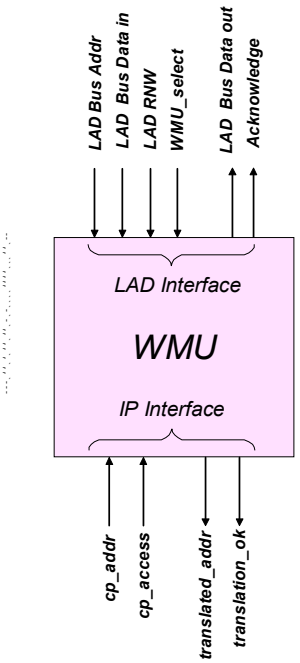
9.4.1.2.1 The WMU

The **WMU is an hardware component responsible for handling virtual memory accesses** requested by the IPs. It operates the translation of the virtual address into the physical address.

The WMU has two interfaces :

- the **LAD interface** to communicate with the Host
- the **IP interface** to obtain the virtual address requested by the IP and to send to the platform the translated address, the physical one.

The WMU is composed of one major element, the **Translation Lookaside Buffer (TLB)**. This latter is constituted of an essential component, the **Content Addressable Memory (CAM)**. The Translation Lookaside Buffer is a buffer (or cache) that contains parts of the page table which translate from virtual into physical addresses. This buffer has a fixed number of entries and is used to improve the speed of virtual address translation. The buffer is typically a content addressable memory (CAM) in which the search key is the virtual address and the search result is a physical address. If the CAM search yields a match the translation is known and the match data are used. If no match exists, the VMW library will intervene to copy the data required and write the new entry in the page table in the TLB



LAD Interface:

- Address Bus
- Data In Bus
- Read / Write signal
- Chip Select signal
- Data out Bus

IP Interface:

- Address *from* the IP
- Valid signal *from* the IP
- Address *to* the platform
- Valid signal *to* the platform
- Acknowledgement signal

Figure 100 — The WMU and its interfaces.

9.4.1.2.1.1 LAD Interface

The WMU is connected directly to the LAD Bus interface. Through this interface, the VMW library can access control, status and address registers of the WMU. In the case of an unsuccessful address translation, the address register contains the IP-generated address that caused the fault. The VMW library obtains this address, copy the requested data in the BRAM, and update the Translation Lookaside Buffer (TLB – a table accessible through LAD, containing the most recent virtual page translations) of the WMU. The virtualization layer screens the programmer and the hardware designer of these events. For the moment, we do not use Access Indicator Register (AIR) and Access Mask Register (AMR). Their purpose is to support prefetching, but this is the subject of the future work. The following table is a summary of the LAD addresses assigned to the WMU:

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 24 — LAD addresses used by the WMU.

LAD Address	Data
0x6000	Control Register
0x6001	Status Register
0x6002	Address Register
0x6003	Address Mask Register
0x6004	Access Indicator Register
0x6020 – 0x603F	32 TLB entries

9.4.1.2.1.2 IP Interface

The WMU needs to obtain the virtual address requested by the IP. Thanks to the Virtual Memory Controller, the WMU receives the virtual address requested by the IP on the signal *cp_addr* and its strobe *cp_access*. Then the WMU translates the virtual address into a physical address : *translated_addr* and its strobe *trans_addr_ok*. These signals are send to the platform thanks to the Virtual Memory Controller. The exact protocol will be explained more in details in the following sections.

9.4.1.2.1.3 WMU registers

All the registers of the WMU are written or read form the LAD Bus. You can get more information about the commands to write (*WC_PeRegWrite*) or read (*WC_PeRegRead)* registers in the WILDCARD™-II Reference Manual [5].

Control register is used to configure the WMU :

- it configures the page size the TLB will deal with. In the current version of the platform, there is only one page size which is supported by the platform : pages of 2kB, it means an physical offset of 11 bits.
- It sets/unsets the hold state of the TLB. This state locks the TLB and waits for a write from the host. A read from the host is only possible in this state. The host must put the TLB in the hold state in order to do a read or a write.

Here are the commands to write the control register :

- the WMU is configured in mode “000”, i.e. a page size of 11 bits.

```
dControl[0] = 0x0;
WC_PeRegWrite(TestInfo.DeviceNum,WMU_Control_Register,1,dControl);
```

- the TLB is put in the hold state

```
dControl[0] = 0x2;
WC_PeRegWrite(TestInfo.DeviceNum,WMU_Control_Register,1,dControl);
```

Layout of the control register :

Status register is used to obtain the state of the WMU. Only one bit of the status register is used : the miss bit. When the TLB cannot match the address provided by the IP for translation, the WMU raises the miss bit, situated at bit 1 in the register. If the status register of the WMU is read and equal to 0x2, it means that a miss interrupt raised. Here is the command to read this register :

```
WC_PeRegRead(TestInfo.DeviceNum,WMU_Status_Register,1,dControl);
```

Address register is used to obtain the last address the WMU received from the Virtual Memory Controller. This register is read-only. Here is the command to read this register :

```
WC_PeRegRead(TestInfo.DeviceNum,WMU_Address_Register,1,dControl);
```


© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.4.1.2.1.4 Translation Lookaside Buffer Description

The writing and the reading of the TLB is done thanks to the address 0x6020 to 0x603F as detailed in the table 5. It is available from the host. The TLB is a buffer (or cache) that contains parts of the page table which translates from virtual into physical addresses. We need to operate writing and reading operations to support the Virtual memory feature.

There are two specific patterns, according to the type of operations, writing or reading. The figures below show how must be organised the data which is going to be written in the TLB and which is the signification of the data received in a read operation.

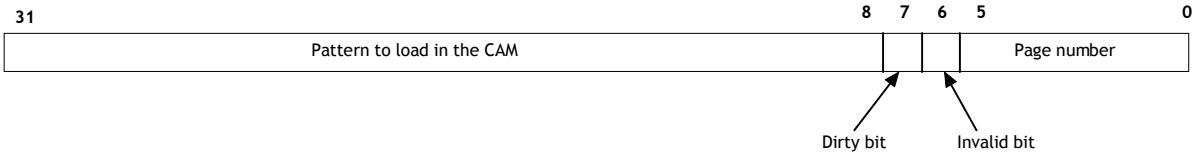


Figure 101 — Write pattern.

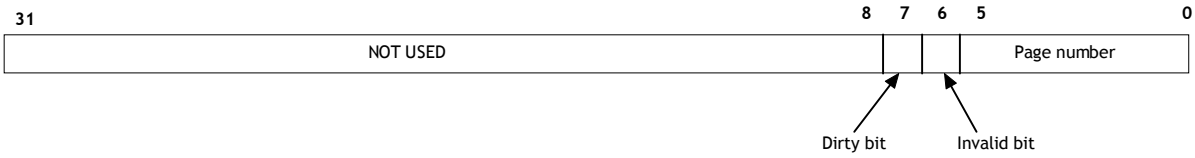


Figure 102 — Read pattern.

9.4.1.2.1.5 Translation mechanism

One page of local memory is composed of 2 kB. In order to access each data, the offset part of the address must be represented with 11 bits. (2^{11} = 2048). The other part of the address is transformed by the WMU into a physical page which represent the page number of the local memory. The figure below shows in details how the virtual address is transformed into the physical address by the WMU:

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2009 – All rights reserved
177
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

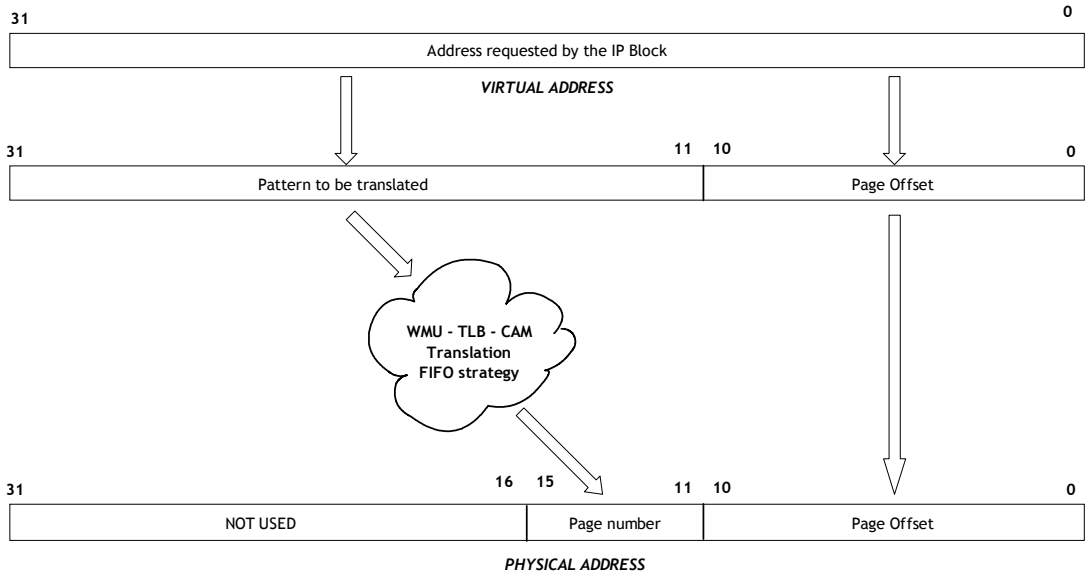


Figure 103 — Translation mechanism.

9.4.1.2.2 The Virtual Memory Controller

The Virtual Memory Controller is an **intermediate component between the Hardware Interface Controller of the platform, the WMU and the IP-Block**. The Virtual Memory Controller looks sort of like a hub: it manages Virtual Memory features.

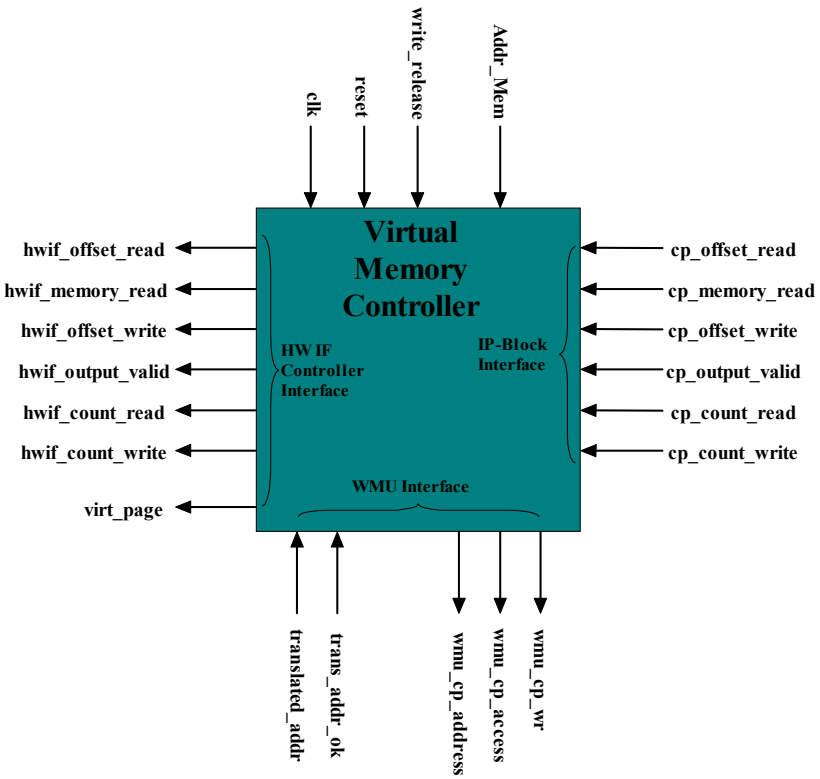


Figure 104 — Virtual Memory Controller entity.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2009 – All rights reserved
Not for Resale

The Virtual Memory Controller is comparable to a hub in the sense that it directs signals from one component towards an other without modifying them significantly. It makes the connexions between three components : the WMU, the Hardware Interface Controller (platform) and the IP-Block.

The Virtual Memory Controller operates at the interface between the IP and the platform. In the Explicit mode, it is like the Virtual Memory Controller doesn't exist, and we get exactly the original platform. In the Virtual mode,

- the Virtual Memory Controller is charged to detour some signals of the standard interface between the IP and the platform into the WMU, and to send back to the platform correct information obtained from the translation mechanism in order that the platform knows which data must be send/received to/by which IP.
- It directs also the IP requests towards the WMU. Because of the page boundary problem, it sends sometimes two requests to the WMU for one single request of the IP.

9.4.1.2.3 The integration into the interface

Originally, the communication between the IPs and the platform is performed by the Hardware Interface Controller. Each IP has a sort of connection slot to this interface. The signals of this interface are the same for all IPs.

In order to have a virtual memory support for each IP, we have to slightly modify the interface. The goal was to integrate the WMU in the platform such a way it modifies as less as possible the interface between the IP and the Hardware Interface Controller.

9.4.1.2.4 Definition of an IP request.

When the IP asks for data access (read or write) it generates four signals. For a read request, the IP asserts the following signals :

- *hw_mem_hwaccel* : IP number
- *hw_offset :* reading address
- *hw_count :* number of data to read
- *memory_read :* strobe signal of the request

For a write request, the IP asserts the following signals :

- *hw_mem_hwaccel1* : IP number
- *hw_offset1 :* writing address
- *hw_count1 :* number of data to write
- *output_valid :* strobe signal of the request and the data.

IPs generate addresses for writing and reading in the local memory and the corresponding signals which validate the writing/reading addresses. **The principle is to detour thanks to the Virtual Memory Controller these signals into the WMU when an IP initiates a virtual memory access**. In turn, the WMU does its task to translate the addresses. It forms a physical address, composed of the physical offset (corresponding to the original Virtual Socket BRAM address space) and a physical page and transfers it to the Hardware Interface Controller thanks to the Virtual Memory Controller, with an additional signal which corresponds to the original *memory_read* signal.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

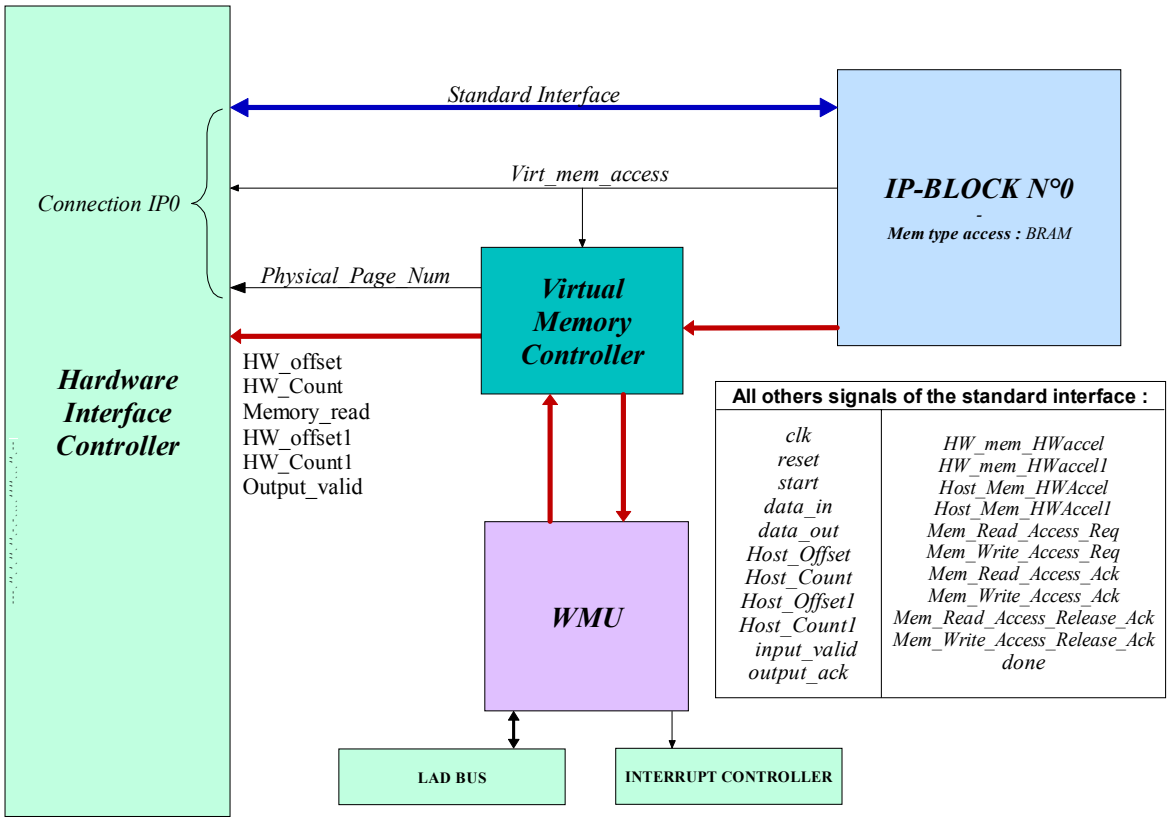


Figure 105 — Integration of the WMU into the platform.

We modify the interface between the IPs and the Hardware Interface Controller by adding only one signal: V*irt_Mem_Access*. Another signal is generated by the WMU as an input to the Hardware Interface Controller: the *phys_page_num* which indicates the number of the page in the BRAM space where the requested data are stored. The other signals are not modified and are represented in the following figure as the “Standard Interface”.

9.4.1.3 The page boundary problem

The BRAM memory space is composed of 32 pages of 512 Dwords (2 Kb). In the Explicit Mode (corresponding to the original platform), the maximum request of one IP is 2 kB, i.e. one entire page of the BRAM. We want the same feature for the Virtual Mode.

The address generated by the IP-Block is a virtual address which refers to data in the user memory space on the host. This address is not necessarily aligned with one page of 2 kB in the main memory. Imagine a request of one IP which is composed of an address which points to data in the middle of a page and has count of 2kB. The request will involve two pages in the main memory as seen in the figure bellow.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

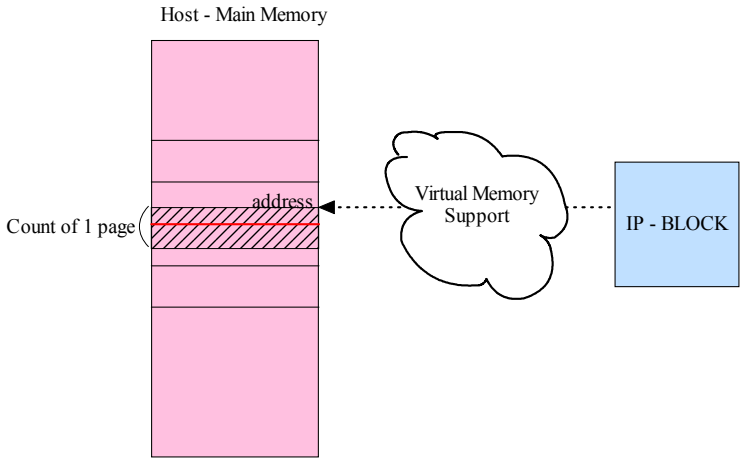


Figure 106 — Page boundary problem.

Let's take an example to illustrate the problem. The IP wants to read 512 Dwords (2 kB), it generates the following signals:

- *hw_offset* = 0x12FAC8
- *hw_count = 0x200*

This address is sent to the WMU to get translated. The page containing the address is not yet in the local memory, so the VMW will copy the entire page from main memory to local memory. The page copied is from the address 0x12F800 to 0x130000. The IP wants to read data from address 0x12FAC8 with a count of 2kB : 0x12FAC8 + 0x800 = 0x1302C8. Remember that the IP sends only one request, and the platform sends automatically all the data to the IP. But in our case, only the data from address 0x12F800 to 0x130000 have been copied in the local memory and the data from address 0x130000 to 0x1302C8 are not available in the local memory because it involves an other page in the main memory which have been not copied in the local memory.

The solution to this problem is to check if the IP request crosses a page boundary. The verification is done by the component Virtual Memory Controller. If this latter detects an over page boundary access, it generates two requests to the WMU for one single request of the IP. The first request to the WMU deals with the first page, making the VMW transfer the first page of the main memory in the local memory and the second request deals with the second involved page, making the VMW copy the second page of data. So that, thanks to the second request of the Virtual Memory Controller, the data from 0x130000 to 0X1302C8 will be available in the local memory and can be send correctly to the IP.

9.4.1.4 IP-Blocks

- *How the IP-Blocks are defined into the platform ?*

As we define that there is a maximum of 32 user IP-blocks can be integrated into the platform, it is very likely that each user IP-block has its own requirement for the interface between the platform and IP-block. To facilitate the platform development and make the integration easy, we should define a **common set of interface signals between the platform and each user IP-block**. Therefore, we have the following interface signals between the platform and each user IP-block listed bellow :

```
entity User_IP_Block_1 is
generic(block_width: positive:=8);
port(
      clk                         : in std_logic;
      reset                       : in std_logic;
      start                       : in std_logic;
      input_valid                 : in std_logic;
      data_in                     : in std_logic_vector(31 downto 0);
      memory_read                 : out std_logic;
      HW_Mem_HWAccel              : out std_logic_vector(31 downto 0);
      HW_Offset                   : out std_logic_vector(31 downto 0);
      HW_Count                    : out std_logic_vector(31 downto 0);
```

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      data_out                    : out std_logic_vector(31 downto 0);
      output_valid                : out std_logic;
      HW_Mem_HWAccel1             : out std_logic_vector(31 downto 0);
      HW_Offset1                  : out std_logic_vector(31 downto 0);
      HW_Count1                   : out std_logic_vector(31 downto 0);
      output_ack                  : in std_logic;
      Host_Mem_HWAccel            : in std_logic_vector(31 downto 0);
      Host_Offset                 : in std_logic_vector(31 downto 0);
      Host_Count                  : in std_logic_vector(31 downto 0);
      Host_Mem_HWAccel1           : in std_logic_vector(31 downto 0);
      Host_Offset1                : in std_logic_vector(31 downto 0);
      Host_Count1                 : in std_logic_vector(31 downto 0);
      Mem_Read_Access_Req         : out std_logic;
      Mem_Write_Access_Req        : out std_logic;
      Mem_Read_Access_Release_Req : out std_logic;
      Mem_Write_Access_Release_Req: out std_logic;
      Mem_Read_Access_Ack         : in std_logic;
      Mem_Write_Access_Ack        : in std_logic;
      Mem_Read_Access_Release_Ack : in std_logic;
      Mem_Write_Access_Release_Ack: in std_logic;
      Virt_Mem_Access             : out std_logic;
      done                        : out std_logic
    );
end User_IP_Block_1;
```

Figure 107 — IP-Block interface.

Assumptions:
a. System is positive edge triggered.
b. Reset is active high.

Interface signals description:

Table 25 — Interface signals description.

clk	the platform system clock input to the user IP-block
reset	reset signal for user IP-block
start	when the platform triggers user IP-block to work, this signal will be asserted
input_valid	when data are ready to be input to the user IP-block, this signal is valid
data_in	input data to the user IP-block
memory_read	when user IP-block needs to read some data from memory space, this signal is asserted for platform to initialise the memory access controllers
HW_Mem_HWAccel	To read data from memory space, user IP-block provides the platform this hardware accelerator number for the platform maps the virtual socket address to physical address
HW_Offset	To read data from memory space, user IP-block provides the platform this offset for the platform determines the starting address to be accessed
HW_Count	To read data from memory space, user IP-block provides the platform this count for the platform determines the data length to be accessed
data_out	output data to the memory space
output_valid	when data are ready to be output to the memory space, this signal is valid
HW_Mem_HWAccel1	To write data to memory space, user IP-block provides the platform this hardware accelerator number for the platform maps the virtual socket address to physical address
HW_Offset1	To write data to memory space, user IP-block provides the platform this offset for the platform determines the starting address to be accessed
HW_Count1	To write data to memory space, user IP-block provides the platform this count for the platform determines the data length to be accessed
output_ack	when platform successfully receives the data from user IP-block, this signal is asserted to acknowledge it
Host_Mem_HWAccel	Besides the user IP-block itself can provide hardware accelerator number to the platform for memory reading operations, the host system can optionally provide the user IP-blocks with this initial hardware accelerator number. Therefore, if the

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

	user IP-block doesn't want to generate its own hardware accelerator number to be provided to the platform, it can just accept this parameter signal as its desiring hardware accelerator number for memory reading operation.
Host_Offset	Provided by host system as an optional parameter signal to generate the necessary memory starting offset address for memory reading operation.
Host_Count	Provided by host system as an optional parameter signal to generate the necessary data counter for memory reading operation.
Host_Mem_HWAccel1	Similar to Host_Mem_HWAccel, if the user IP-block doesn't want to generate its own hardware accelerator number to be provided to the platform, it can just accept this parameter signal as its desiring hardware accelerator number for memory writing operation.
Host_Offset1	Similar to Host_Offset, this signal is provided by host system as an optional parameter to generate the necessary memory starting offset address for memory writing operation.
Host_Count1	Similar to Host_Count, this signal is provided by host system as an optional parameter to generate the necessary data counter for memory writing operation.
Mem_Read_Access_Req	User IP-block issue this signal to request the memory reading.
Mem_Read_Access_Ack	Platform generates this signal to acknowledge the user request.
Mem_Read_Access_Release_Req	When user IP-block finishes the memory reading operations, it asserts this signal to ask for releasing the memory reading.
Mem_Read_Access_Release_Ack	Platform generates this signal to acknowledge the user releasing request, and therefore the memory is released for the reading operations.
Mem_Write_Access_Req	User IP-block issue this signal to request the memory writing.
Mem_Write_Access_Ack	Platform generates this signal to acknowledge the user request.
Mem_Write_Access_Release_Req	When user IP-block finishes the memory writing operations, it asserts this signal to ask for releasing the memory writing.
Mem_Write_Access_Release_Ack	Platform generates this signal to acknowledge the user releasing request, and therefore the memory is released for the writing operations.
Virt_Mem_Access	To choose between the Original mode or Virtual Memory mode
done	the user interrupt signal to the platform

- *How the IP-Blocks communicate with the platform ?*

The key point of the interaction between user IP-block and platform is that the user IP-block can utilise some parameter signals to ask for reading and writing data through the memory space. In such case, the user IP-block positively asks for accessing the memory space and the platform passively processes the requests from user IP-blocks and thus accesses data through memory space. The memory data access is implemented by the platform itself. **User IP-blocks only need to issue the memory requests together with parameter signals to the interface, and other part of the work is taken by the platform.**

A precise protocol exists between the IP and the platform to make them communicate together and exchange data. Here are two examples of the protocol :

- the first example uses **Explicit Mode**: this is the original protocol. The IP asks to read some data, the platform sends to the IP the data requested. Then the IP asks to write some data, the platform receives from the IP the data send.
- The second example uses **Virtual Mode**: this is the extended protocol. Parameters passing appears and is realised through the register space of the IP in use. The IP asks to read the parameters in the register space. The platform sends the parameters to the IP. The IP gets configured. Then the IP asks to read some data by generating a virtual address, the platform sends the data requested to the IP with the support of the VMW library which does all the memory management. Then the IP asks to write data, the platform receives the data sent by the IP. If there's some memory management to do, the VMW will intervene.

Example 1: Explicit Mode

Data flow between the platform and user IP-blocks for a read and a write cycle.

(1) Host computer system writes data to the memory space.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

(2) Host computer system issues a command to trigger the relevant user IP-block starting to work.

(3) Once the user IP-block starts to work, it will issue “Mem_Read_Access_Req” to ask for reading the memory.

(4) Platform accepts the user reading request and if memory is available, it generates an acknowledgement signal “Mem_Read_Access_Ack” to the user IP-block.

(5) Once the user IP-block receives the acknowledgement signal, it can ask for reading some data directly from the memory space. This request will be performed by asserting “memory_read” together with setting up some other parameter signals, i.e. “HW_Mem_HWAccel”, “HW_Offset”, “HW_Count”.

(6) The platform accepts those signals and reads data from the memory space according to the parameters. When the platform finishes each reading, it asserts “input_valid” and the data are ready in the “data_in” to be input to the user IP-block.

(7) The user IP-block receives the data from the interface.

(8) User IP-block will assert “Mem_Read_Access_Release_Req” to ask for releasing the reading operations when finished.

(9) Platform generates an acknowledgement signal “Mem_Read_Access_Release_Ack” to release the reading operations.

(10) Then user IP-block will perform the video algorithms or computations to implement the video application functionality.

(11) After video processing finished, user IP-block asserts “Mem_Write_Access_Req” to ask for writing operations.

(12) Platform accepts the user writing request and if memory is available, it generates an acknowledgement signal “Mem_Write_Access_Ack” to the user IP-block.

(13) Then user IP-block asserts “output_valid” together with some other parameters, i.e. “HW_Mem_HWAccel1”, “HW_Offset1”, “HW_Count1” to request writing data back to the memory space.

(14) The platform accepts “output_valid”, receives the data from “data_out” and writes data back to the memory space according to the parameter signals. When each writing is finished, the platform asserts “output_ack” to acknowledge the writing process.

(15) User IP-block will assert “Mem_Write_Access_Release_Req” to ask for releasing the writing operations when all the processing is finished.

(16) Platform generates an acknowledgement signal “Mem_Write_Access_Release_Ack” to release the writing operations.

(17) A user interrupt signal is asserted to inform the platform that the user IP-block finishes all the internal tasks.

(18) A final interrupt signal is asserted (if the interrupt is enabled) by the platform to inform the host computer system that the current task is done.

Example 2: Virtual Mode

Data flow between the platform and user IP-blocks for a read and a write cycle.

(1) The VMW library initialises the WMU and writes parameters in the register space allocated to the given IP.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

(2) The VMW manager writes the appropriate *Mem_Type* for user IP block to access the register space. It is necessary for the platform to know which memory (among BRAM, SRAM, DRAM and the registers) the IP wants to access.

(3) The VMW issues a command to trigger the execution transfer. The user IP Block starts reading the parameters.

(4) IP block issues *Mem_Read_Access_Req* to the HW Interface Controller

(5) The HW Interface Controller accepts the user request and generates the acknowledgement signal *Mem_Read_Access_Ack* to the user IP block.

(6) Once the user IP block receives the acknowledgement, it starts reading from its allocated register space to get the parameters (*Virt_Mem_Access* is set '0' – we don't use virtual memory feature for parameters exchange). To access its predefined slot in the register space, the IP block generates *Memory_Read* together with setting up other relevant signals, i.e., *HW_Mem_HWAccel*, *HW_Offset and HW_Count*. Given that the emplacement is predefined, the IP designer doesn't have to modify these values in this step.

(8) The HW Interface Controller accepts the request and reads the data from the register space, and sends it back to the IP block. For each DWORD read, the platform asserts *Input_Valid,* and the data are available on the *Data_In*

(9) The user IP block receives the data (the parameters) from the IP interface

(10) When finished, User IP block asserts *Mem_Read_Access_Release_Req* to ask releasing the read port.

(11) Platform generates the acknowledgement signal *Mem_Read_Access_Release_Ack* to the user IP block.

(12) The IP configures itself with the parameters it just received.

(13) The VMW manager writes the appropriate *Mem_Type* for user IP block to access the BRAM space.

(14) The VMW issues a command to trigger the user IP block start the computation.

(15) IP block issues *Mem_Read_Access_Req* to the HW Interface Controller

(16) The HW Interface Controller accepts the user request and generates an acknowledgement signal *Mem_Read_Access_Ack* to the user IP block.

(17) Once the user IP block receives the acknowledgement, it starts reading from the virtual memory space by asserting *Virt_Mem_Access*. The IP block generates also *Memory_Read* together with setting up other relevant signals, i.e. *HW_Mem_HWAccel*, *HW_Offset and HW_Count*. These signals are sent to the WMU.

(18) The WMU gets the virtual address from the IP block. If the translation is not possible in the TLB, it generates an interrupt that is handled by the VMW library. The VMW library copies the requested page into the BRAM and updates the page table. Then the WMU provides the HW Interface Controller with the following translated signals: *Memory_Read*, *HW_Mem_HWAccel*, *HW_Offset*, *HW_Count* and *phys_page_num*.

(19) The HW Interface Controller accepts the request and reads the data from the BRAM space, and sends data to the IP block. For each DWORD read, the platform asserts *Input_Valid* and the data are available on the *Data_In*.

(20) The user IP block receives the data from the IP interface.

(21) When finished, User IP block asserts *Mem_Read_Access_Release_Req* to ask for releasing the read port.

(22) Platform generates an acknowledgement signal *Mem_Read_Access_Release_Ack* to the user IP block.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

(23) Then user IP block performs the computations.

(24) After the computation is over, the user IP block asserts *Mem_Write_Access_Req* to ask for writing data back. This signal is sent to the HW Interface Controller.

(25) Platform generates the acknowledgement signal *Mem_Write_Access_Ack* to the user IP block.

(26) The user IP block asserts *Output_Valid* together with other signals, i.e. *HW_Mem_HWAccel1*, *HW_Offset1*, *HW_Count1* to request writing data back to BRAM. These signals are sent to the WMU. If the translation is possible (the BRAM contains the translated physical page), the WMU generates the *Output_Valid* and sends the translated address to the HW Interface Controller. If the translation is not possible, the VMW library will intervene.

(27) The HW Interface Controller accepts *Output_Valid* and receives the data from *Data_Out*, writes data back to the BRAM space. After each DWORD writing, the platform asserts *Output_Ack* and send to IP block to acknowledge the writing operations.

(28) The user IP block asserts *Mem_Write_Access_Release_Req* to ask for releasing the write operations.

(29) Platform generates acknowledgement signal *Mem_Write_Access_Release_Ack* to release the write operations.

(30) The user IP block asserts the interrupt signal to notify the platform about the task completion.

9.4.2 Software side

9.4.2.1 Virtual Socket API

Application Programming Interface (API) is a set of functions coded in C language allowing data communication between host computer system and Wildcard II platform. Actually, the virtual socket API function calls here are not simply the Wildcard II API functions. They are constructed on the basic Wildcard II API functions to perform the data communication between the host system and virtual socket platform.

The following virtual socket API function calls are defined:

WC_Open	Open a WildCard II FPGA device.
WC_Close	Close a WildCard II FPGA device.
VSInit	Initialise the virtual socket platform.
VSSystemReset	Reset the virtual socket platform.
VSUserResetEnable	Enable a reset for user IP-blocks.
VSUserResetDisable	Disable a reset for user IP-blocks.
VSIntEnable	Enable / Disable the platform interrupt generation
VSIntClear	Clean up the platform interrupt.
VSIntCheck	Check and wait for the platform interrupt generation.
VSIntStatusWrite	Set up the interrupt status and mask registers.
VSIntStatusRead	Read the interrupt status and mask registers.
VSWrite	Write data to memory space by non-DMA operations.
VSRead	Read data from memory space by non-DMA operations.
VSDMAInit	Initialise DMA channel and allocate DMA data buffer.
VSDMAClear	Release DMA channel and data buffer allocation.
VSDMAWrite	Write data to memory space by DMA operations.
VSDMARead	Read data from memory space by DMA operations.
HW_Module_Controller	Trigger and control the user IP-blocks to work.

- **WC_Open**
  Please refer to user manual of WildCard II device [5] for details about its function.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

- **WC_Close**

  Please refer to user manual of WildCard II device [5] for details about its function.

- **VSInit**

Syntax:	WC_RetCode VSInit(WC_TestInfo *TestInfo);
Description:	Initialize the virtual socket platform.
Arguments:	TestInfo A point of the index number of a WildCard II FPGA device.
Returns:	An enumeration structure WC_RetCode [5].
Comments:	VSInit is used to initialize a virtual socket platform. The process of initialization is to reset platform, get platform device information, load image file to FPGA and set default system clock frequency. The initialization must be performed after opening a WildCard II FPGA device by WC_Open function call

- **VSSystemReset**

Syntax:	WC_RetCode VSSystemReset(WC_TestInfo *TestInfo);
Description:	Reset the virtual socket platform.
Arguments:	TestInfo A point of the index number of a WildCard II FPGA device.
Returns:	An enumeration structure WC_RetCode [5].
Comments:	VSSystemReset is to reset a virtual socket platform. Unlike VSInit, this function only issues the system reset signal to the platform. Usually, if users want to reset the virtual socket platform during its work, this function can be employed. In some other cases, however, if users want to load image file other than reset the platform, VSInit can be employed.

- **VSUserResetEnable**

Syntax:	WC_RetCode VSUserResetEnable(WC_TestInfo *TestInfo);
Description:	Enable a reset for user IP-blocks.
Arguments:	TestInfo A point of the index number of a WildCard II FPGA device.
Returns:	An enumeration structure WC_RetCode [5].
Comments:	VSUserResetEnable is used to enable a reset for user IP-blocks. Normally, when a system reset signal is issued by VSSystemReset, all the hardware modules, including memory space controller, hardware module controller and user IP-blocks, will be reset. But sometimes it is unnecessary for users to reset their own IP-blocks when a system reset signal is issued. In such case, VSUserResetEnable and VSUserResetDisabe will provide a way for users to enable or shield their own IP-blocks from system reset signal. See Also: VSUserResetDisable

- **VSUserResetDisable**

Syntax:	WC_RetCode VSUserResetDisable(WC_TestInfo *TestInfo);
Description:	Disable a reset for user IP-blocks.
Arguments:	TestInfo A point of the index number of a WildCard II FPGA device.
Returns:	An enumeration structure WC_RetCode [5].
Comments:	VSUserResetDisable is used to shield the user IP-blocks from a system reset signal. See Also: VSUserResetEnable

- **VSIntEnable**

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Syntax:	WC_RetCode VSIntEnable(WC_TestInfo *TestInfo,boolean IntEnable);
Description:	Enable or disable the platform interrupt generation.
Arguments:	TestInfo A point of the index number of a WildCard II FPGA device. IntEnable A parameter to enable or disable the interrupt generation.
Returns:	An enumeration structure WC_RetCode [5].
Comments:	When users need platform to generate an interrupt signal to computer host system after the hardware modules finish their work, IntEnable should be set to "1" to enable the interrupt generation. Otherwise, IntEnable is set to "0" to disable the interrupt generation.

- **VSIntClear**

Syntax:	WC_RetCode VSIntClear(WC_TestInfo *TestInfo);
Description:	Clean up the generated platform interrupt signal.
Arguments:	TestInfo A point of the index number of a WildCard II FPGA device.
Returns:	An enumeration structure WC_RetCode [5].
Comments:	After the platform generates an interrupt signal to the host system, users can issue VSIntClear to release the platform interrupt and clean it up.

- **VSIntCheck**

Syntax:	WC_RetCode VSIntCheck(WC_TestInfo *TestInfo);
Description:	Check and wait for the platform interrupt generation.
Arguments:	TestInfo A point of the index number of a WildCard II FPGA device.
Returns:	An enumeration structure WC_RetCode [5].
Comments:	Before the platform generates an interrupt signal, users can employ this function call to check and wait for the interrupt signal to come. Usually, there is a default INT_TIMEOUT_MS parameter set up in the C language header file to indicate how long time (millisecond) this function call checks and waits for the coming interrupt.

- **VSIntStatusWrite**

Syntax:	WC_RetCode VSIntStatusWrite(WC_TestInfo *TestInfo, DWORD *IntStatus, DWORD *IntMask);
Description:	Set up the platform interrupt status and mask registers.
Arguments:	TestInfo A point of the index number of a WildCard II FPGA device. IntStatus Interrupt Status Register IntMask Interrupt Mask Register
Returns:	An enumeration structure WC_RetCode [5].
Comments:	VSIntStatusWrite is used to set up the platform interrupt status register and interrupt mask register. Usually, the users can write all zeros to interrupt status register for cleaning up the platform recorded interrupt status. When an interrupt is generated from the user IP-block, the corresponding bit of the interrupt status register will be overwritten by that interrupt signal and thus set to "1". If the corresponding bit of the interrupt mask register is to enable the platform interrupt, a final interrupt will be generated from the platform to the host system. The maximum user interrupt resources can be up to 32.

- **VSIntStatusRead**

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Syntax:	WC_RetCode VSIntStatusRead(WC_TestInfo *TestInfo, DWORD *IntStatus, DWORD *IntMask);
Description:	Read the interrupt status and mask registers.
Arguments:	TestInfo A point of the index number of a WildCard II FPGA device. IntStatus Interrupt Status Register IntMask Interrupt Mask Register
Returns:	An enumeration structure WC_RetCode [5].
Comments:	VSIntStatusRead is used to read the platform interrupt status register and interrupt mask register. See Also: VSIntStatusWrite

- **VSWrite**

Syntax:	WC_RetCode VSWrite(WC_TestInfo *TestInfo, unsigned int HWAccel, unsigned long Offset, unsigned long Count, DWORD *Buffer);
Description:	Write data to memory space by non-DMA operations.
Arguments:	TestInfo A point of the index number of a WildCard II FPGA device. HWAccel Hardware accelerator number (user IP-block number). Offset Offset of the memory address space Count Size of Dwords to write Buffer A Pointer to an array of Dwords, Buffer Length ≥ Count
Returns:	An enumeration structure WC_RetCode [5].
Comments:	VSWrite is used to write data from host system to the memory space by non-DMA operations. There are 4 types of memory address space inside the platform, i.e. Register, BRAM, SRAM and DRAM. VSWrite can automatically map the virtual socket address space into the physical address space, and thus access all the 4 types of physical memory storage within one function call. It is assumed that the host writes the data to the memory space belonging to corresponding IP-blocks, as long as the users provide appropriate HWAccel, Offset and Count to the function call. The same assumption is applied for VSRead, VSDMAWrite and VSDMARead function calls listed below. The maximum number of HWAccel is from 0 to 31, which means there are at most 32 user IP-blocks can be integrated into the virtual socket platform. The highest 2 bits of "Offset" [31:30] indicate which specific memory space the hardware accelerators (user IP-blocks) want to write, while "Offset" [29:28] indicate which specific memory space the hardware accelerators want to read.

Offset[31:30]	Memory Write Accessed
0 0	Register
0 1	BRAM
1 0	SRAM
1 1	DRAM

Offset[29:28]	Memory Read Accessed
0 0	Register
0 1	BRAM
1 0	SRAM
1 1	DRAM

Usage:	(1) if the user IP-block 2 needs to write 16 double words to the register space with the offset 0 by non-DMA transfer, then write *VSWrite(TestInfo, 2, 0x00000000 + 0, 16, Buffer)* (2) if the user IP-block 5 needs to write 256 double words to the BRAM space with the

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

	offset 10 by non-DMA transfer, then write *VSWrite(TestInfo, 5, 0x50008000 + 10, 256, Buffer)* (3) if the user IP-block needs to write 1024 double words to the SRAM space with the offset 0 by non-DMA transfer, then write *VSWrite(TestInfo, 1, 0xa0000000 + 0, 1024, Buffer)* (4) if the user IP-block needs to write 4096 double words to the DRAM space with the offset 1000 by non-DMA transfer, then write *VSWrite(TestInfo, 10, 0xf0000000 + 1000, 4096, Buffer)* Note: To access the individual memory space correctly according to its maximum size, we must have Offset + Count ≤ Size of Memory Space, and the highest 2 bits of Offset [31:30] are set correctly.

- **VSRead**

Syntax:	WC_RetCode VSRead(WC_TestInfo *TestInfo, unsigned int HWAccel, unsigned long Offset, unsigned long Count, DWORD *Buffer);
Description:	Read data from memory space by non-DMA operations.
Arguments:	TestInfo A point of the index number of a WildCard II FPGA device. HWAccel Hardware accelerator number (user IP-block number). Offset Offset of the memory address space Count Size of Dwords to read Buffer A Pointer to an array of Dwords, Buffer Length ≥ Count
Returns:	An enumeration structure WC_RetCode [5].
Comments:	VSRead is used to read data from the memory space to host system by non-DMA operations. There are 4 types of memory address space inside the platform, i.e. Register, BRAM, SRAM and DRAM. VSRead can automatically map the virtual socket address space into the physical address space, and thus access all the 4 types of physical memory storage within one function call. See Also: VSWrite
Usage:	(1) if the user IP-block 18 needs to read 10 double words from the register space with the offset 5 by non-DMA transfer, then write *VSRead(TestInfo, 18, 0x00000000 + 5, 10, Buffer)* (2) if the user IP-block 27 needs to read 256 double words from the BRAM space with the offset 100 by non-DMA transfer, then write *VSRead(TestInfo, 27, 0x50008000 + 100, 256, Buffer)* (3) if the user IP-block needs to read 2000 double words from the SRAM space with the offset 500 by non-DMA transfer, then write *VSRead(TestInfo, 20, 0xa0000000 + 500, 2000, Buffer)* (4) if the user IP-block needs to read 5000 double words from the DRAM space with the offset 300 by non-DMA transfer, then write *VSRead(TestInfo, 25, 0xf0000000 + 300, 5000, Buffer)* Note: To access the individual memory space correctly according to its maximum size, we must have Offset + Count ≤ Size of Memory Space, and the Offset [29:28] are set correctly.

- **VSDMAInit**

Syntax:	WC_RetCode VSDMAInit(WC_TestInfo *TestInfo,unsigned long Count);
Description:	Initialize DMA channel and allocate DMA data buffer.
Arguments:	TestInfo A point of the index number of a WildCard II FPGA device. Count Size of the DMA data buffer.
Returns:	An enumeration structure WC_RetCode [5].
Comments:	VSDMAInit is used to initialize the DMA data transfer channels and buffer. “Count” indicates the Dword size to be allocated for DMA data transfer buffer.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

- **VSDMAClear**

Syntax:	WC_RetCode VSDMAClear(WC_TestInfo *TestInfo);
Description:	Release DMA channel and data buffer allocation.
Arguments:	TestInfo A point of the index number of a WildCard II FPGA device.
Returns:	An enumeration structure WC_RetCode [5].
Comments:	VSDMAClear is used to release the allocated DMA channels and data buffer. See Also: VSDMAInit

- **VSDMAWrite**

Syntax:	WC_RetCode VSDMAWrite(WC_TestInfo *TestInfo, unsigned int BSDRAM, unsigned long StartAddr, unsigned long Count, DWORD *Buffer);
Description:	Write data to memory space by DMA operations.
Arguments:	TestInfo A point of the index number of a WildCard II FPGA device. BSDRAM Indicate which specific memory space is accessed. StartAddr Start address in the memory space from which data are transferred. Count Size of the data to be transferred. Buffer A Pointer to an array of Dwords, Buffer Length ≥ Count
Returns:	An enumeration structure WC_RetCode [5].
Comments:	VSDMAWrite is used to write data from host system to the memory space by DMA operations. There are 3 types of memory address space inside the platform through which data can be transferred by DMA operations, i.e. BRAM, SRAM and DRAM. VSDMAWrite can automatically map the virtual socket address space into the physical address space, and thus access all the 3 types of physical memory storage within one function call. BSDRAM indicates which specific memory space will be accessed.
Usage:	(1) if the user IP-block 4 needs to write 512 double words to the BRAM space with the starting offset 0 by DMA transfer, then write *VSDMAWrite(TestInfo, 0, 0x50008000 + 0x200 * 4 + 0, 512, Buffer)* (2) if the user IP-block needs to write 1024 double words to the SRAM space with the starting offset 100 by DMA transfer, then write *VSDMAWrite(TestInfo, 1, 0xa0000000 + 100, 1024, Buffer)* (3) if the user IP-block needs to write 4096 double words to the DRAM space with the starting offset 50 by DMA transfer, then write *VSDMAWrite(TestInfo, 2, 0xf0000000 + 50, 4096, Buffer)*

BSDRAM	Memory Accessed
0	BRAM
1	SRAM
2	DRAM

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

- **VSDMARead**

Syntax:	WC_RetCode VSDMARead(WC_TestInfo *TestInfo, unsigned int BSDRAM, unsigned long StartAddr, unsigned long Count, DWORD *Buffer);
Description:	Read data from memory space by DMA operations.
Arguments:	TestInfo A point of the index number of a WildCard II FPGA device. BSDRAM Indicate which specific memory space is accessed. StartAddr Start address in the memory space from which data are transferred. Count Size of the data to be transferred. Buffer A Pointer to an array of Dwords, Buffer Length ≥ Count
Returns:	An enumeration structure WC_RetCode [5].
Comments:	VSDMARead is used to read data from the memory space to host system by DMA operations. There are 3 types of memory address space inside the platform through which data can be transferred by DMA operations, i.e. BRAM, SRAM and DRAM. VSDMARead can automatically map the virtual socket address space into the physical address space, and thus access all the 3 types of physical memory storage within one function call. BSDRAM indicates which specific memory space will be accessed.
Usage:	(1) if the user IP-block 28 needs to read 256 double words from the BRAM space with the starting offset 100 by DMA transfer, then write *VSRead(TestInfo, 0, 0x50008000 + 0x0200 * 28 + 100, 256, Buffer)* (2) if the user IP-block needs to read 2000 double words from the SRAM space with the starting offset 500 by DMA transfer, then write *VSRead(TestInfo, 1, 0xa0000000 + 500, 2000, Buffer)* (3) if the user IP-block needs to read 5000 double words from the DRAM space with the starting offset 300 by DMA transfer, then write *VSRead(TestInfo, 2, 0xf0000000 + 300, 5000, Buffer)*

BSDRAM	Memory Accessed
0	BRAM
1	SRAM
2	DRAM

- **HW_Module_Controller**

Syntax:	WC_RetCode HW_Module_Controller(WC_TestInfo *TestInfo, unsigned long HW_IP_Start);
Description:	Trigger and control the user IP-blocks to work.
Arguments:	TestInfo A point of the index number of a WildCard II FPGA device. HW_IP_Start Indicate which specific user IP-block is triggered to work.
Returns:	An enumeration structure WC_RetCode [5].
Comments:	HW_Module_Controller is used to trigger and control the user IP-block to work. Each bit of HW_IP_Start represents a corresponding IP-block. Before using this function call to trigger IP-blocks to work, users should write memory access type to the platform for each IP-block to indicate which specific memory space the corresponding IP-block will access, including reading and writing. Please refer to the Appendix, an example program, to get understanding about its operation.
Usage:	(1) if the user wants to implement IP-block 5, then write *HW_IP_Start = 0x00000020;* *HW_Module_Controller(TestInfo, HW_IP_Start)* (2) if the user wants to implement IP-block 16, then write *HW_IP_Start = 0x00010000;* *HW_Module_Controller(TestInfo, HW_IP_Start)*

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.4.2.2 Virtual Memory Extension library

The software support of the Virtual Memory Extension is named the Virtual Manager Window Library (VMW library). It is composed of functions which manage all the transfers between the host memory and the local memory. The library relies on the FPGA card driver, the Wildcard II API. The University of Calgary framework contains a software API based also on the card driver. The Virtual Socket API proposes a list of functions which manages the platform: configuration, data transfers, IP handling … The VMW library uses both of these API and also WIN32 for memory management.

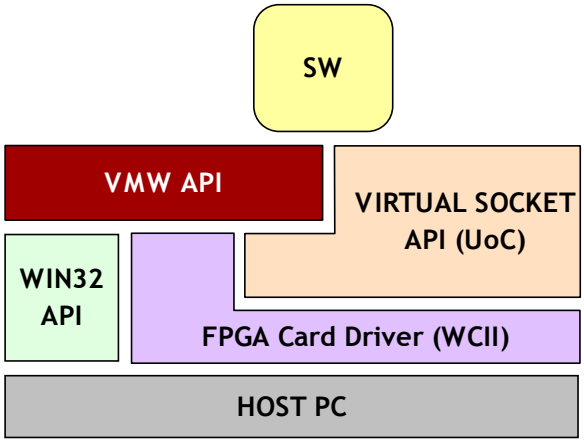


Figure 108 — The VMW library.

- **VMW_Init**

Syntax:	int VMW_Init(void)
Description:	the function initialises the Virtual Manager Window
Arguments:	none
Returns:	-1 when an error occurs, 0 else
Comments:	It creates and starts an interrupt thread. When an interrupt occurs, the thread treats immediately this interrupt. The functions initialises the WMU with the mode “000” to have 11 bits for the offset. It continues with the initialisation of the TLB.

- **VMW_stop**

Syntax:	void VMW_Stop(void)
Description:	the function stops the Virtual Manager Window
Arguments:	none
Returns:	none
Comments:	the function stops the interrupt thread

- **IP_Start**

Syntax:	int IP_start(unsigned int IP_num, IP_Param_Struct *IP_Param)
Description:	The function start a given IP to work
Arguments:	the number of the IP to use (IP_num) and a pointer to the parameter structure(*IP_Param)
Returns:	-1 when an error occurs, 0 else
Comments:	the function starts by clearing the finish bit of the IP ready to work. The function starts a first time the IP to pass parameters through register memory space. It sets the

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

	memory access type register accordingly. The function starts a second time the IP to make the work. The memory access type register is set in order that the IP have access to BRAM. The function finishes only when the finish interrupt signal of the given IP rises. This is a locking function

- **Interrupt_Proc**

Syntax:	DWORD WINAPI Interrupt_Proc(LPVOID lpParam)
Description:	Heart of the library : this function responds to the interruptions of the platform
Arguments:	compulsory parameters to link this function to the interrupt thread

Detailed description of the algorithm

The function manages two types of interrupts:

- *the finish interrupt*

When an IP has finished its work, the platform raises this interrupt. The IP must be started twice. One start to pass the parameters and one start to make the real work. So we have to distinguish if the parameters were just passed or if the real job is done.

If the parameters are passed, we just clear the finish bit and wait for the next interrupt.

If the job is finished, the function copies all the dirty pages. A dirty page is a page in the local memory which has been written by one IP and not yet copied back in the main memory of the Host. The function determines which IP has just finished. It informs the *IP_start* function that the IP finished its work by setting an event. This event is required by the *IP_start* function to get unlocked. The corresponding interrupt bit is cleared.

- *the miss interrupt*

When the WMU is unable to translate the address received by the IP, the function raises this interrupt. It puts the WMU in the hold state, necessary to manage correctly the WMU. It clears the miss interrupt bit. It obtains the missed address by reading the WMU. It computes the pattern that must be written in the TLB. It gets a free TLB entry by using the FIFO strategy. It checks if this entry is free (i.e. not dirty) otherwise copies the page back. Then it copies one page of data from the host memory to the local memory. It updates the software TLB structure in order to hold in main memory the patterns written in the TLB because once written, it is impossible to get back the patterns written). It writes the TLB at the free entry with the pattern computed before and exits the WMU from the hold state.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

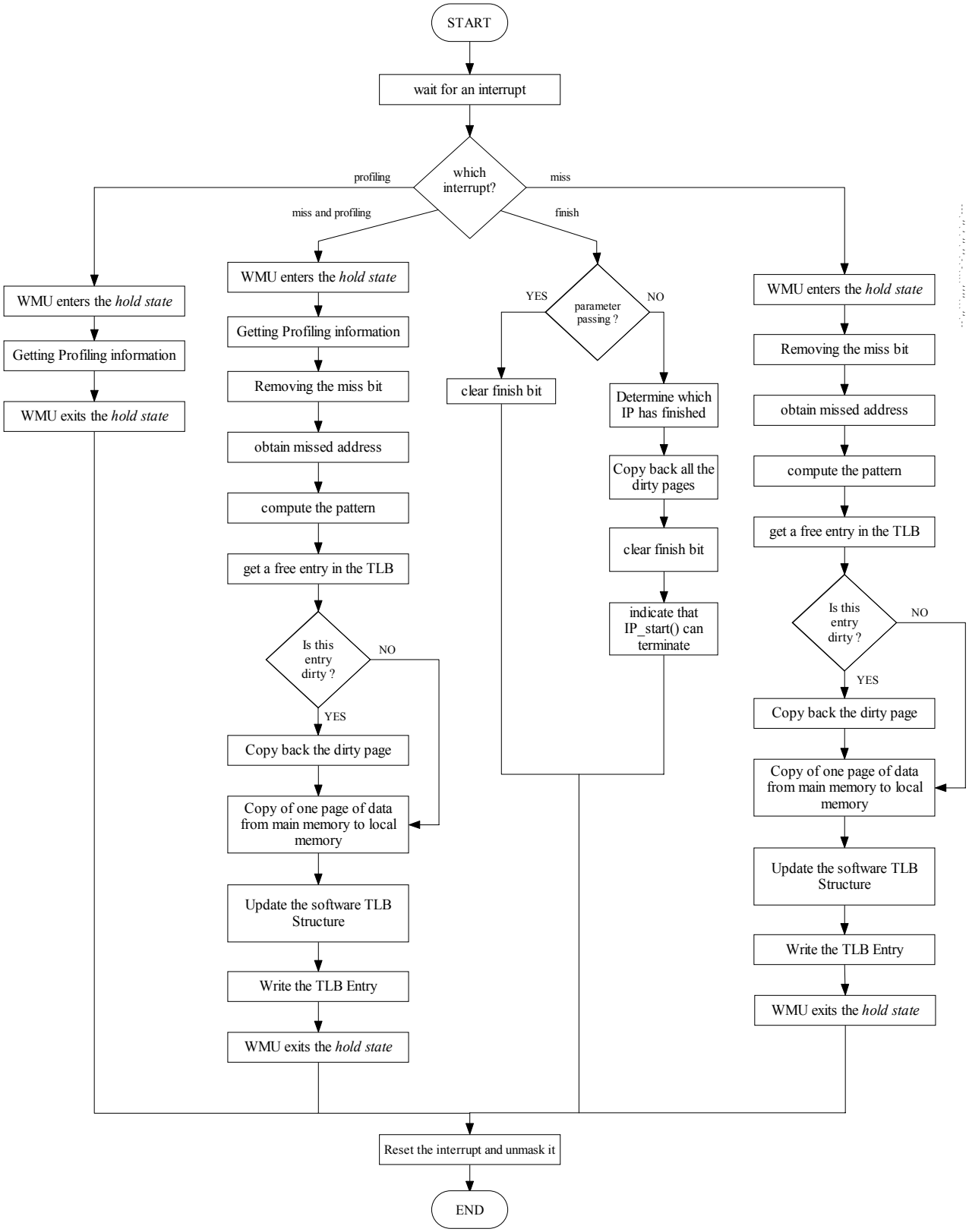


Figure 109 — Algorithm of the Interrupt_proc function.

© ISO/IEC 2009 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.5 How to build the platform

You are an IP-designer and you want to know how to integrate your IP-Block in the platform and make the system work on the FPGA card. Here are the steps to follow :

• First, you must **get acquainted with the communication protocol** between the IP-Block and the platform. IP Finite State Machine templates are presented in this subclause for the two modes (Explicit and Virtual modes) to have an idea to how implement the communication protocol presented in part 9.4.1.4 in VHDL.

• Then, you have to **simulate the whole platform** with your IP-Block and be sure the communications between your IP and the platform occurs as you expected.

• You have to **synthesise you design** to get the bitstream file to program the FPGA on the card.

• Finally, you have to **build the software** project and write your C source code which allows you to use the card.

9.5.1 Integration of IP-Blocks

Imagine you have your IP-Block ready. You have tested it independently and it works. Your core has a given number of IN/OUT specific signals. Now is the time for integrating this IP in the platform with a well-defined interface. What you designed is what we will call the IP core. Around this core, you need a wrapper to integrate your IP core in the platform with the specific interface.

The wrapper looks like a Finite State Machine (FSM) which manages the transfers of data between the platform and your IP thanks to the well-defined interface. Thanks to this FSM, your IP core will be started to do the work with the data available.

According to the operating mode of the platform, we gives here two examples of the FSM. But this FSM must be entirely modified specifically to the IP core and must respond to the demands of the IP core in terms of data. Nevertheless, the steps of the protocol to read or to write data must be rigorously respected. The numbers of reading and writing requests are set by the IP-designer, specifically to its IP core.

9.5.2 IP FSM template for Explicit Mode

Here is an example template for user IP-block which can be integrated into the platform. The VHDL codes are shown in Figure 110. Some notes are added to this example. Among the notes, “template” indicates where the template codes are used and “user code” shows where the users should insert their own VHDL codes there. For this example, parameter signals are set up as

```
HW_Mem_HWAccel/HW_Mem_HWAccel1 = 0,
HW_Offset/HW_Offset1 = 0,
HW_Count/HW_Count1 = 256.
```

The users may change those parameters according to their own requirement in practical cases.

If users choose to use Host_Mem_HWAccel, Host_Offset, Host_Count, Host_Mem_HWAccel1, Host_Offset1, Host_Count1 as their parameter signals, then set up the following in the VHDL codes:

```
HW_Mem_HWAccel <= Host_ Mem_HWAccel,
HW_Offset <= Host_Offset,
HW_Count <= Host_Count,
HW_Mem_HWAccel1 <= Host_Mem_HWAccel1,
HW_Offset1 <= Host_Offset1,
HW_Count1 <= Host_Count1


library ieee;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;
entity User_IP_Block_0 is
generic(block_width: positive:=8);
```


© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
port(
    clk                          : in std_logic;
    reset                        : in std_logic;
    start                        : in std_logic;
    input_valid                  : in std_logic;
    data_in                      : in std_logic_vector(31 downto 0);
    memory_read                  : out std_logic;
    HW_Mem_HWAccel               : out std_logic_vector(31 downto 0);
    HW_Offset                    : out std_logic_vector(31 downto 0);
    HW_Count                     : out std_logic_vector(31 downto 0);
    data_out                     : out std_logic_vector(31 downto 0);
    output_valid                 : out std_logic;
    HW_Mem_HWAccel1              : out std_logic_vector(31 downto 0);
    HW_Offset1                   : out std_logic_vector(31 downto 0);
    HW_Count1                    : out std_logic_vector(31 downto 0);
    output_ack                   : in std_logic;
    Host_Mem_HWAccel             : in std_logic_vector(31 downto 0);
    Host_Offset                  : in std_logic_vector(31 downto 0);
    Host_Count                   : in std_logic_vector(31 downto 0);
    Host_Mem_HWAccel1            : in std_logic_vector(31 downto 0);
    Host_Offset1                 : in std_logic_vector(31 downto 0);
    Host_Count1                  : in std_logic_vector(31 downto 0);
    Mem_Read_Access_Req          : out std_logic;
    Mem_Write_Access_Req         : out std_logic;
    Mem_Read_Access_Release_Req  : out std_logic;
    Mem_Write_Access_Release_Req : out std_logic;
    Mem_Read_Access_Ack          : in std_logic;
    Mem_Write_Access_Ack         : in std_logic;
    Mem_Read_Access_Release_Ack  : in std_logic;
    Mem_Write_Access_Release_Ack : in std_logic;
    done                         : out std_logic
     );
end User_IP_Block_0;

architecture behav of User_IP_Block_0 is

type memory32bit is array (natural range <>) of std_logic_vector(31     downto 0);

signal data_buffer  : memory32bit(0 to 255);
signal user_status  : integer range 0 to 31;

begin
  process(reset,clk)
  variable i : integer range 0 to 256;
  variable j : integer range 0 to 256;
  begin
if (reset = '1') then                                        --template
  output_valid <= '0';                                       --template
      i := 0;                                                    --template
      j := 0;                                                    --template
      user_status <= 0;                                          --template
      memory_read <= '0';                                        --template
      Mem_Read_Access_Req <= '0';                                --template
      Mem_Write_Access_Req <= '0';                               --template
      Mem_Read_Access_Release_Req <= '0';                        --template
      Mem_Write_Access_Release_Req <= '0';                       --template
      done <= '1';                                               --template
elsif rising_edge(clk) then                                  --template
      if (start = '1') then                                      --template
        case user_status is                                      --template
        when 0 =>                                                --template
          done <= '0';                                           --template
          user_status <= 1;                                      --template
        when 1 =>                                    --template
          Mem_Read_Access_Req <= '1';                            --template
          user_status <= 2;                                      --template
        when 2 =>                                                --template
          if (Mem_Read_Access_Ack = '1') then                    --template
            Mem_Read_Access_Req <= '0';                          --template
            user_status <= 3;                                    --template
          else                                                   --template
            user_status <= 2;                                    --template
          end if;                                    --template
        when 3 =>                                        --template
          HW_Mem_HWAccel <= x"00000000";                     --USER CODE
          HW_Offset <= x"00000000";                          --USER CODE
```

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      HW_Count <= x"00000100";                          --USER CODE
      j := 0;                                               --template
      user_status <= 4;                                     --template
    when 4 =>
      memory_read <= '1';                                   --template
      user_status <= 5;                                     --template
    when 5 =>                                               --template
      memory_read <= '0';                                   --template
      if (input_valid = '1') then                           --template
        data_buffer(j) <= data_in;                          --template
        j := j + 1;                                         --template
        if (j >= conv_integer(unsigned(HW_Count))) then
          j := 0;                                           --template
          user_status <= 6;                                 --template
        else                                                --template
          user_status <= 5;                                 --template
        end if;                                             --template
      end if;                                               --template
    when 6 =>
      Mem_Read_Access_Release_Req <= '1';                   --template
  user_status <= 7;                                       --template
    when 7 =>
      if (Mem_Read_Access_Release_Ack = '1') then           --template
        Mem_Read_Access_Release_Req <= '0';                 --template
        user_status <= 8;                                   --template
      else                                                  --template
        user_status <= 7;                                   --template
  end if;                                                 --template
    when 8 =>                                               --template
      --USER ALGORITHMS HERE                            --USER CODE
      user_status <= 9;                                     --template
    when 9 =>                                               --template
      Mem_Write_Access_Req <= '1';                          --template
      user_status <= 10;                                    --template
    when 10 =>                                              --template
      if (Mem_Write_Access_Ack = '1') then                  --template
        Mem_Write_Access_Req <= '0';                        --template
        user_status <= 11;                                  --template
      else                                                  --template
        user_status <= 10;                                  --template
      end if;                                               --template
    when 11 =>                                              --template
      HW_Mem_HWAccel1 <= x"00000000";                   --USER CODE
      HW_Offset1 <= x"00000000";                        --USER CODE
      HW_Count1 <= x"00000100";                         --USER CODE
      j := 0;                                               --template
      user_status <= 12;                                    --template
    when 12 =>                                              --template
      data_out <= data_buffer(j);                           --template
      output_valid <= '1';                                  --template
      user_status <= 13;                                    --template
    when 13 =>                                              --template
      if (output_ack = '1') then                            --template
        j := j + 1;                                         --template
        if (j >= conv_integer(unsigned(HW_Count1))) then    --template
          output_valid <= '0';                              --template
          user_status <= 14;                                --template
        else                                                --template
          data_out <= data_buffer(j);                       --template
          output_valid <= '1';                              --template
          user_status <= 13;                                --template
        end if;                                             --template
      else                                                  --template
        output_valid <= '0';                                --template
        user_status <= 13;                                  --template
      end if;                                               --template
    when 14 =>                                              --template
      Mem_Write_Access_Release_Req <= '1';                  --template
      user_status <= 15;                                    --template
    when 15 =>                                              --template
      if (Mem_Write_Access_Release_Ack = '1') then          --template
        Mem_Write_Access_Release_Req <= '0';                --template
        user_status <= 16;                                  --template
      else                                                  --template
        user_status <= 15;                                  --template
      end if;                                               --template
```


© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
        when 16 =>                                              --template
          done <= '1';                                          --template
          user_status <= 16;                                    --template
        when others => null;                                    --template
        end case;                                               --template
            Else                                                --template
        output_valid <= '0';                                    --template
        i := 0;                                                 --template
        j := 0;                                                 --template
        user_status <= 0;                                       --template
        memory_read <= '0';                                     --template
        Mem_Read_Access_Req <= '0';                             --template
        Mem_Write_Access_Req <= '0';                            --template
        Mem_Read_Access_Release_Req <= '0';                     --template
        Mem_Write_Access_Release_Req <= '0';                    --template
        done <= '1';                                            --template
      end if;                                                   --template

    end if;                                                     --template
  end process;                                                  --template
end behav;                                                      --template
```

Figure 110 — An example template for user IP-block in Explicit Mode.

9.5.3 IP FSM template for Virtual Mode

Here is described an example template for an IP-block (using the Virtual Mode) which can be integrated into the platform

```
library ieee;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;

entity User_IP_Block_1 is
generic(block_width: positive:=8);
port(
      clk                          : in std_logic;
      reset                        : in std_logic;
      start                        : in std_logic;
      input_valid                  : in std_logic;
      data_in                      : in std_logic_vector(31 downto 0);
      memory_read                  : out std_logic;
      HW_Mem_HWAccel               : out std_logic_vector(31 downto 0);
      HW_Offset                    : out std_logic_vector(31 downto 0);
      HW_Count                     : out std_logic_vector(31 downto 0);
      data_out                     : out std_logic_vector(31 downto 0);
      output_valid                 : out std_logic;
      HW_Mem_HWAccel1              : out std_logic_vector(31 downto 0);
      HW_Offset1                   : out std_logic_vector(31 downto 0);
      HW_Count1                    : out std_logic_vector(31 downto 0);
      output_ack                   : in std_logic;
      Host_Mem_HWAccel             : in std_logic_vector(31 downto 0);
      Host_Offset                  : in std_logic_vector(31 downto 0);
      Host_Count                   : in std_logic_vector(31 downto 0);
      Host_Mem_HWAccel1            : in std_logic_vector(31 downto 0);
      Host_Offset1                 : in std_logic_vector(31 downto 0);
      Host_Count1                  : in std_logic_vector(31 downto 0);
      Mem_Read_Access_Req          : out std_logic;
      Mem_Write_Access_Req         : out std_logic;
      Mem_Read_Access_Release_Req  : out std_logic;
      Mem_Write_Access_Release_Req : out std_logic;
      Mem_Read_Access_Ack          : in std_logic;
      Mem_Write_Access_Ack         : in std_logic;
      Mem_Read_Access_Release_Ack  : in std_logic;
      Mem_Write_Access_Release_Ack : in std_logic;
      Virt_Mem_Access              : out std_logic;
      done                         : out std_logic
     );
end User_IP_Block_1;

architecture behav of User_IP_Block_1 is
```

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
type memory32bit is array (natural range <>) of std_logic_vector(31 downto 0);
signal data_buffer      : memory32bit(0 to 255);
signal param_buffer     : memory32bit(0 to 15);
signal user_status      : integer range 0 to 31;


-- ******* Example of IP parameters ***************************************
signal virtual_access   : std_logic_vector(31 downto 0) := (others => '0') ;
signal address_read     : std_logic_vector(31 downto 0) := (others => '0') ;
signal address_write    : std_logic_vector(31 downto 0) := (others => '0') ;
signal count            : std_logic_vector(31 downto 0) := (others => '0') ;
************************************************************************

begin
  process(reset,clk)
  variable i : integer range 0 to 1024;
  variable j : integer range 0 to 1024;
  variable temp_HW_Count : std_logic_vector(31 downto 0);
  variable temp_HW_Count1 : std_logic_vector(31 downto 0);
  variable count_temp : integer range 0 to 50 ; -- 50 is arbitrary
  begin
    if (reset = '1') then
      output_valid <= '0';
      i := 0;
      j := 0;
      user_status <= 0;
      memory_read <= '0';
      HW_Mem_HWAccel <= (others => '0') ;
      HW_Offset <= (others => '0') ;
      HW_Count <= (others => '0') ;
      data_out <= (others => '0') ;
      HW_Mem_HWAccel1 <= (others => '0') ;
      HW_Offset1 <= (others => '0') ;
      HW_Count1 <= (others => '0') ;
      temp_HW_Count := (others => '0') ;
      temp_HW_Count1 := (others => '0') ;
      count_temp := 0 ;
      Mem_Read_Access_Req <= '0';
      Mem_Write_Access_Req <= '0';
      Mem_Read_Access_Release_Req <= '0';
      Mem_Write_Access_Release_Req <= '0';
      Virt_Mem_Access <= '0';
      done <= '1';
    elsif rising_edge(clk) then

      if (start = '1') then             -- first start of the IP (pass parameters)
        case user_status is
        when 0 =>
          done <= '0';
          memory_read <= '0';
          output_valid <= '0';
          Mem_Read_Access_Req <= '0';
          Mem_Write_Access_Req <= '0';
          Mem_Read_Access_Release_Req <= '0';
          Mem_Write_Access_Release_Req <= '0';
          Virt_Mem_Access <= '0';       -- Explicit mode to pass parameters
          user_status <= 1;
        when 1 =>
          Mem_Read_Access_Req <= '1';
          user_status <= 2;
        when 2 =>
          if (Mem_Read_Access_Ack = '1') then
            Mem_Read_Access_Req <= '0';
            user_status <= 3;
          else
            user_status <= 2;
          end if;
        when 3 =>
          HW_Mem_HWAccel <= x"00000001";    -- IP N°1
          HW_Offset <= x"00000000" ;        -- Offset 0 in the register space
          HW_Count <= x"00000010";          -- read 16 registers in the reg space
          temp_HW_Count := x"00000010";
          j := 0;
          user_status <= 4;
          count_temp := 8 ;
        when 4 =>
```


© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
          memory_read <= '1';
          user_status <= 5;
        when 5 =>
          memory_read <= '0';
          if (input_valid = '1') then
            param_buffer(j) <= data_in;
            j := j + 1;
            if (j >= conv_integer(unsigned(temp_HW_count))) then
              j := 0;
              user_status <= 6;
            else
              user_status <= 5;
            end if;
          end if;
        when 6 =>
          Mem_Read_Access_Release_Req <= '1';
          user_status <= 7;
        when 7 =>
          if (Mem_Read_Access_Release_Ack = '1') then
            Mem_Read_Access_Release_Req <= '0';
            user_status <= 8;
          else
            user_status <= 7;
          end if;
        when 8 =>
        -- ***** CONFIGURATION OF THE IP WITH THE PARAMETERS TRANSFERED ********
          virtual_access <= param_buffer(0) ;             -- USER CODE
          address_read <= param_buffer(1) ;               -- USER CODE
          address_write <= param_buffer(2) ;              -- USER CODE
          count <= param_buffer(3) ;                  -- USER CODE
 -- ************************************************************************
          done <= '1';
          user_status <= 9;
        when 9 =>
          user_status <= 10 ;
        when 10 =>
          if (start = '1') then
-- wait for starting the IP a second time (work), VMW sets Mem_access type for BRAM -- in the soft
before restarting
             done <= '0';
             memory_read <= '0';
             output_valid <= '0';
             Mem_Read_Access_Req <= '0';
             Mem_Write_Access_Req <= '0';
             Mem_Read_Access_Release_Req <= '0';
             Mem_Write_Access_Release_Req <= '0';
             Virt_Mem_Access <= virtual_access(0) ; -- USER CODE : virtual/explicit
             user_status <= 11 ;
          else
             user_status <= 10 ;
          end if ;
        when 11 =>
          Mem_Read_Access_Req <= '1';
          user_status <= 12;
        when 12 =>
          if (Mem_Read_Access_Ack = '1') then
            Mem_Read_Access_Req <= '0';
            user_status <= 13;
          else
            user_status <= 12;
          end if;
        when 13 =>                    -- READ REQUEST
          HW_Mem_HWAccel <= x"00000001";   -- USER CODE : IP n°1
          HW_Offset <= address_read;       -- USER CODE : reading address
          HW_Count <= count;               -- USER CODE : number of data to process
          temp_HW_Count := count;
          j := 0;
          user_status <= 14;
        when 14 =>
          memory_read <= '1';
          user_status <= 15;
        when 15 =>
          memory_read <= '0';
          if (input_valid = '1') then
            data_buffer(j) <=  data_in ;
            j := j + 1;
```

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      if (j >= conv_integer(unsigned(temp_HW_Count))) then
        j := 0;
        user_status <= 16;
      else
        user_status <= 15;
      end if;
    end if;
  when 16 =>
    Mem_Read_Access_Release_Req <= '1';
    user_status <= 17;
  when 17 =>
    if (Mem_Read_Access_Release_Ack = '1') then
      Mem_Read_Access_Release_Req <= '0';
      user_status <= 18;
    else
      user_status <= 17;
    end if;
  when 18 =>
  -- *********** PROCESSING OF THE DATA *****************
  --                                                     *
  --                 USER CODE                           *
  --                                                     *
  -- *****************************************************
    user_status <= 19;
  when 19 =>
    Mem_Write_Access_Req <= '1';
    user_status <= 20;
  when 20 =>
    if (Mem_Write_Access_Ack = '1') then
      Mem_Write_Access_Req <= '0';
      user_status <= 21;
    else
      user_status <= 20;
    end if;
  when 21 =>                    -- WRITE REQUEST
    HW_Mem_HWAccel1 <= x"00000001";    -- USER CODE : IP n°1
    HW_Offset1 <= address_write ;      -- USER CODE : writing address
    HW_Count1 <= count;                -- USER CODE :  nb of data to process
    temp_HW_Count1 := count;
    j := 0;
    user_status <= 22;
  when 22 =>
    data_out <= data_buffer(j);
    output_valid <= '1';
    user_status <= 23;
  when 23 =>
    if (output_ack = '1') then
      j := j + 1;
      if (j >= conv_integer(unsigned(temp_HW_Count1))) then
        output_valid <= '0';
        user_status <= 24;
      else
        data_out <= data_buffer(j);
        output_valid <= '1';
        user_status <= 23;
      end if;
    else
      output_valid <= '0';
      user_status <= 23;
    end if;
  when 24 =>
    Mem_Write_Access_Release_Req <= '1';
    user_status <= 25;
  when 25 =>
    if (Mem_Write_Access_Release_Ack = '1') then
      Mem_Write_Access_Release_Req <= '0';
      user_status <= 26;
    else
      user_status <= 25;
    end if;
  when 26 =>
    done <= '1';
    user_status <= 26;
  when others => null;
  end case;
else
```


© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
        output_valid <= '0';
        i := 0;
        j := 0;
        user_status <= user_status ;
        memory_read <= '0';
        Mem_Read_Access_Req <= '0';
        Mem_Write_Access_Req <= '0';
        Mem_Read_Access_Release_Req <= '0';
        Mem_Write_Access_Release_Req <= '0';
        Virt_Mem_Access <= '0';
        done <= '1';
      end if;

    end if;
  end process;
end behav;
```

Figure 111 — An example template for user IP-block in Virtual Mode.

9.6 Simulation of the platform

9.6.1 Configuration

The virtual socket platform is simulated by **Mentor Graphics© ModelSim 5.4e® or high version** of the simulation tools. This part shows how to simulate the whole platform using simulation tools.

Before simulating the platform, users must have the following files in hand:

system_virtual_socket_arch.vhd	This file defines the top-level architecture for the whole VHDL model system. By default it instantiates one host component and one WILDCARD™-II board component.
system_cfg.vhd	Many of the components in the VHDL simulation environment are configurable. The system_cfg.vhd file contains a VHDL configuration statement that adds the ability to enable or disable the SRAM and DRAM simulation models, and to set the architectures for the PE, Host and I/O connectors components.
host_virtual_socket_arch.vhd	This file is the template for creating the simulated host program. It defines the architecture for the Host component on the System level of the VHDL model.

(*Source Templates Files : \annapolis\wildcard_ii\vhdl\template\sim*)

Please refer to the reference manual of the WildCard II [5] for more details about the function of each file.

9.6.2 Simulation process

A general macro file (.do) has been created to simplify the simulation process : "*simulation_process.do"*. This file relies on the files "*project_vcom.do*" and "*project_vsim.do*" given by the Annapolis support (*Source : \annapolis\wildcard_ii\vhdl\template\sim*).

Use "*simulation_process.do"* to compile all the virtual socket platform VHDL files, including the user IP-blocks. The users may modify the project directory in this file for their own applications. It executes also the simulation operations.

To execute this macro file from the ModelSim(tm) VHDL simulator: using the "*File->Execute Macro*" menu option select this file in the browser window and click on "*Open*".

```
-----------------------------------------------------------------------
--
--  Model Technology ModelSim(tm) Compilation Macro File
--
--  Follow the itemised instructions below in order to customise
--  the WILDCARD(tm) simulation model compilation script to meet
--  your application needs.
```

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
--
--  To execute this macro file from the ModelSim(tm) VHDL simulator:
--
--    Using the "File->Execute Macro" menu option select this
--    file in the browser window and click on "Open".
--
-------------------------------------------------------------------------
-----------------
-- COMPILATION --
-----------------
-------------------------------------------------------------------------
--
--  1. Change the PROJECT_BASE and ANNAPOLIS_BASE variables to your
--     project directory and Annapolis installation directory,
--     respectively:
--
-------------------------------------------------------------------------
set ANNAPOLIS_BASE c:/annapolis
set PROJECT_BASE $ANNAPOLIS_BASE/wildcard_ii/examples/virtual_socket_testing/vhdl/sim
set SCRIPT_BASE $ANNAPOLIS_BASE/wildcard_ii/examples/virtual_socket_testing/vhdl/modelsim

--
--  Do not change these lines!!
--
set WILDCARD_BASE   $ANNAPOLIS_BASE/wildcard_ii
set PE_BASE         $WILDCARD_BASE/vhdl/pe
set SIMULATION_BASE $WILDCARD_BASE/vhdl/simulation
set TOOLS_BASE      $WILDCARD_BASE/vhdl/tools

do $SIMULATION_BASE/simulation_vcom.do

-------------------------------------------------------------------------
--
--  2. Add your project's PE architecture VHDL file here, as
--     well as any VHDL files on which your PE design depends:
--
-------------------------------------------------------------------------

---- Files from UoC ----
vcom -93 -explicit -work pe_lib $PROJECT_BASE/data_type_definition_package.vhd
vcom -93 -explicit -work pe_lib $PROJECT_BASE/bram_dma_source_entarch.vhd
vcom -93 -explicit -work pe_lib $PROJECT_BASE/bram_memory_destination_entarch.vhd
vcom -93 -explicit -work pe_lib $PROJECT_BASE/bram_memory_source_entarch.vhd
vcom -93 -explicit -work pe_lib $PROJECT_BASE/bram_dma_destination_entarch.vhd
vcom -93 -explicit -work pe_lib $PROJECT_BASE/dma_source_entarch.vhd
vcom -93 -explicit -work pe_lib $PROJECT_BASE/memory_destination_entarch.vhd
vcom -93 -explicit -work pe_lib $PROJECT_BASE/memory_source_entarch.vhd
vcom -93 -explicit -work pe_lib $PROJECT_BASE/dma_destination_entarch.vhd
vcom -93 -explicit -work pe_lib $PROJECT_BASE/dram_dma_source_entarch.vhd
vcom -93 -explicit -work pe_lib $PROJECT_BASE/dram_memory_destination_entarch.vhd
vcom -93 -explicit -work pe_lib $PROJECT_BASE/dram_memory_source_entarch.vhd
vcom -93 -explicit -work pe_lib $PROJECT_BASE/dram_dma_destination_entarch.vhd
vcom -93 -explicit -work pe_lib $PROJECT_BASE/User_IP_Block_0.vhd
vcom -93 -explicit -work pe_lib $PROJECT_BASE/User_IP_Block_1.vhd
vcom -93 -explicit -work pe_lib $PROJECT_BASE/User_IP_Block_2.vhd


---- Files for the WMU and TLB ----
vcom -93 -explicit -work pe_lib $PROJECT_BASE/wmu.vhd
vcom -93 -explicit -work pe_lib $PROJECT_BASE/tlb.vhd
vcom -93 -explicit -work pe_lib $PROJECT_BASE/COMPARE_4.vhd
vcom -93 -explicit -work pe_lib $PROJECT_BASE/COUNT_16.vhd
vcom -93 -explicit -work pe_lib $PROJECT_BASE/Decode_1.vhd
vcom -93 -explicit -work pe_lib $PROJECT_BASE/Decode_2.vhd
vcom -93 -explicit -work pe_lib $PROJECT_BASE/Decode_3.vhd
vcom -93 -explicit -work pe_lib $PROJECT_BASE/Decode_4.vhd
vcom -93 -explicit -work pe_lib $PROJECT_BASE/Encode_1_MSB.vhd
vcom -93 -explicit -work pe_lib $PROJECT_BASE/Encode_2_MSB.vhd
vcom -93 -explicit -work pe_lib $PROJECT_BASE/Encode_3_MSB.vhd
vcom -93 -explicit -work pe_lib $PROJECT_BASE/Encode_4_LSB.vhd
vcom -93 -explicit -work pe_lib $PROJECT_BASE/Encode_4_MSB.vhd
vcom -93 -explicit -work pe_lib $PROJECT_BASE/INIT_SRL16_AND.vhd
vcom -93 -explicit -work pe_lib $PROJECT_BASE/CAM_SRL16.vhd
vcom -93 -explicit -work pe_lib $PROJECT_BASE/CAM_16WORDS.vhd
vcom -93 -explicit -work pe_lib $PROJECT_BASE/CAM_generic_word.vhd
vcom -93 -explicit -work pe_lib $PROJECT_BASE/CAM_top.vhd
```


© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
---- Files for the Virtual Memory Controller ----
vcom -93 -explicit -work pe_lib $PROJECT_BASE/Virtual Memory Controller.vhd

---- Top design File ----
vcom -93 -explicit -work pe_lib $PROJECT_BASE/virtual_socket_testing.vhd


------------------------------------------------------------------------
--
--  3. Add your project's IO Connector architecture VHDL file here, as
--     well as any VHDL files on which your IO connector design depends:
--
------------------------------------------------------------------------
--vcom -93 -explicit -work simulation $PROJECT_BASE/<IO Architecture>.vhd


------------------------------------------------------------------------
--
--  4. Add your project's host architecture VHDL file here, as
--     well as any VHDL files on which your host design depends:
--
------------------------------------------------------------------------
vcom -93 -explicit -work simulation $PROJECT_BASE/host_virtual_socket_arch.vhd


------------------------------------------------------------------------
--
--  5. Add your project's system configuration VHDL file here:
--
------------------------------------------------------------------------
vcom -93 -explicit -work simulation $PROJECT_BASE/system_virtual_socket_arch.vhd
vcom -93 -explicit -work simulation $PROJECT_BASE/system_cfg.vhd

-----------------
-- SIMULATION  --
-----------------
vsim -t ps simulation.system_config

---------------------
-- WAVE generation --
---------------------
view wave
do $SCRIPT_BASE/IP.do

-----------
-- RUN  ---
-----------
run
```

Figure 112 — ModelSim compilation macro file for the project.

The file is composed of many parts :

- *compilation* : this part compiles all the files of the platform in the library *pe_lib* and all the files necessary to the simulation environment.
- *simulation* : it starts the simulation of the whole environment
- *wave generation* : it constructs the waveform window for simulation. The principal signals of the platform are present to visualise the waveforms.
- *Run* : starts the simulation (be careful to the time of simulation)

9.6.3 Simulating software

The key point of the virtual socket system simulation is to write appropriate system testbench file. Actually, "***host_virtual_socket_arch.vhd***" **is the testbench file for this simulation**. The reference manual of WildCard II [5] lists all the necessary simulated WildCard II API function calls (host APIs). Because virtual socket API function calls [48] are constructed directly on the WildCard II API function calls, users may refer to the "WildCard_Program_Library_Functions.c" and replace the virtual socket API function calls with some appropriate host APIs to trigger and simulate the whole virtual socket platform.

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

For example, if the following is a paragraph of the C codes as a conformance test.

```
int main(int argc,char *argv[])
{
    WC_RetCode rc=WC_SUCCESS;
WC_TestInfo TestInfo;
DWORD Buffer0[512];

rc=VSInit(&TestInfo);
if (rc != WC_SUCCESS)
{
    rc = WC_Close(TestInfo.DeviceNum);
    return rc;
}

for (i=0;i<16;i++)
{
    Buffer0[i]=i;
}
HWAccel = 0;
Offset = 0x00000000;
Count = 16;
Display = 1;
rc=VSWrite(&TestInfo,HWAccel,Offset,Count,Buffer0);
rc=VSRead(&TestInfo,HWAccel,Offset,Count,Buffer0);

rc = WC_Close(TestInfo.DeviceNum);
return(rc);
}
```

Figure 113 — An example of the C codes for testing the platform.

The functionality of the C codes is to test the register data transfer by non-DMA operations. The codes utilise some virtual socket API function calls, i.e. "*VSInit*", "*VSWrite*" and "*VSRead*". Besides, the original WildCard II API function call "*WC_Close*" is also used in the C codes. To trigger and simulate the whole platform, users may translate those C codes and replace the necessary virtual socket API function calls [48] with appropriate simulated WildCard II API functions (host APIs), write the testbench codes in "*host_virtual_socket_arch.vhd*", compile all the project file using "simulation_process.do" and start to simulate the whole platform in the ModelSim environment.

The following is an example testbench file after translating those C codes to the host APIs.

```
------------------------- Library Declarations ---------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;

library SIMULATION;
use SIMULATION.Host_Package.all;
use SIMULATION.Clock_Generator_Package.all;

------------------------ Architecture Declaration -------------------

architecture virtual_socket of Host is

begin
  ----------------------------------------------------------
  --
  --  Below is an example of how to use host "API"
  --  functions to send commands to the WILDSTAR(tm) II
  --  board.  Copy this file to your project directory
  --  and modify it to simulate the behaviour of the
  --  host portion of your application.
  --
  ----------------------------------------------------------
  main : process

  variable Device              : WC_DeviceNum := 0;
  variable fClkFreqMhz         : float;
  variable Buffer0             : DWORD_array ( 0 to 511 );
```


© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
  variable Result_Buffer       : DWORD_array ( 0 to 511 );
  variable dControl            : DWORD_array ( 0 to 3 );
  variable bReset              : BOOLEAN := false;
  constant HWAccel             : DWORD := 16#0#;
  constant Offset              : DWORD := 16#0#;
  constant Count               : DWORD := 16#10#;
  constant User_Reset_Disable  : DWORD := 16#fff6#;
  constant DMA_Working         : DWORD := 16#fff7#;

  begin
    ------------------------------------------------------
    ----  At first, deassert the command request signal --
    ----  Leave this line uncommented.                  --
    ------------------------------------------------------
    Cmd_Req <= FALSE;
    --
    ------------------------------------------------------------
    ----  Set the board number to 0.  This is the default --
    ----  configuration.  For multiboard systems, or      --
    ----  single board systems where the board in not in  --
    ----  slot 0, set this variable to the desired board  --
    ----  number as needed.                               --
    ------------------------------------------------------------
    --
    Device := 0;

        bReset  := false;
    WC_PeReset
    (
      Cmd_Req,
      Cmd_Ack,
      device,
      bReset
);
wait for 100ns;

    bReset  := true;
    WC_PeReset
    (
      Cmd_Req,
      Cmd_Ack,
      device,
      bReset
);
wait for 100ns;

    bReset  := false;
    WC_PeReset
    (
      Cmd_Req,
      Cmd_Ack,
      device,
      bReset
);

    fClkFreqMhz := 50.0;
    WC_ClkSetFrequency
    (
      Cmd_Req,
      Cmd_Ack,
      device,
      fClkFreqMhz
);

WC_IntEnable
(
  Cmd_Req,
  Cmd_Ack,
  device,
  false
);

    WC_IntReset
    (
      Cmd_Req,
      Cmd_Ack,
```

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      Device
);

WC_IntEnable
(
  Cmd_Req,
  Cmd_Ack,
  device,
  true
    );

        dControl(0) := 16#0#;
    WC_PeRegWrite
    (
      Cmd_Req,
      Cmd_Ack,
      device,
      User_Reset_Disable,
      1,
      dControl
    );

       for i in 0 to Count-1 loop
      Buffer0(i) := i;
end loop;

    dControl(0) := 16#0#;
    WC_PeRegWrite
    (
      Cmd_Req,
      Cmd_Ack,
      device,
      DMA_Working,
      1,
      dControl
);

    WC_PeRegWrite
    (
      Cmd_Req,
      Cmd_Ack,
      device,
      Offset,
      Count,
      Buffer0
);
report " Finished Writing 16 DWORDs to Registers...";

    dControl(0) := 16#0#;
    WC_PeRegWrite
    (
      Cmd_Req,
      Cmd_Ack,
      device,
      DMA_Working,
      1,
      dControl
);


          for i in 0 to Count-1 loop
       WC_PeRegRead
       (
         Cmd_Req,
         Cmd_Ack,
         device,
         Offset + i,
         1,
         dControl
       );
       Result_Buffer(i) := dControl(0);
     end loop;

          for i in 0 to Count-1 loop
       report "Original Data(" & integer'image(i) & ") = " &
       Hex_To_String(Buffer0(i)) & ", Read Data(" & integer’image(i)
```


© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
      & ") = " & Hex_To_String(Result_Buffer(i));
    end loop;

    report "Successfully Reading 16 DWORDs from Registers...";
    report "Example Complete";

    wait;  -- End of the simulated host program

  end process;

end architecture;
```

Figure 114 — An example of the simulation testbench file host_virtual_socket_arch.vhd.

Your design works with the whole platform. Now, let's see now how to synthesise it in order to program the FPGA.

9.7 Synthesis of the platform

The synthesis process is composed of two phases. A first phase to transform VHDL Files into an .EDF file (Electronic Design Electronic Interchange Format). It is a industry-standard netlist. The second is to transform this standard netlist into a bitstream necessary to program the FPGA. The design flow of Xilinx© is used.

9.7.1 Phase 1 : having .EDF netlist file

The architectures of the platform hardware modules are prototyped using VHDL language, and synthesised using **Synplicity© Synplify®**. The target technology is the FPGA device (2V3000fg676-4) from the Xilinx© Virtex-II®. The following is a synthesis file for implementation of the whole virtual socket platform system.

```
#-------------------------------------------------------------------------
#-
#-  1. Change the PROJECT_BASE and WILDCARD_BASE variables to your
#-      project directory and WILDCARD-II installation directory,
#-      respectively:
#-
#-------------------------------------------------------------------------
set ANNAPOLIS_BASE "c:/annapolis"
set PROJECT_BASE  "$ANNAPOLIS_BASE/wildcard_ii/examples/virtual_socket/vhdl/sim"

#
# Do not modify these lines!
#
set WILDCARD_BASE   "$ANNAPOLIS_BASE/wildcard_ii"
set PE_BASE         "$WILDCARD_BASE/vhdl/pe"
set SIMULATION_BASE "$WILDCARD_BASE/vhdl/simulation"
set TOOLS_BASE      "$WILDCARD_BASE/vhdl/tools"
set SYNTHESIS_BASE  "$WILDCARD_BASE/vhdl/synthesis"

source "$SYNTHESIS_BASE/tcl/wildcard_pe.tcl"

#-------------------------------------------------------------------------
#-
#-  2. Add your project's PE architecture VHDL file here, as
#-      well as any VHDL or constraint files on which your PE
#-      design depends:
#-      NOTE: They must be in the order specified below.
#-
#-------------------------------------------------------------------------

add_file -vhdl -lib pe_lib "$PROJECT_BASE/data_type_definition_package.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/bram_dma_source_entarch.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/bram_memory_destination_entarch.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/bram_memory_source_entarch.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/bram_dma_destination_entarch.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/dma_source_entarch.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/memory_destination_entarch.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/memory_source_entarch.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/dma_destination_entarch.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/dram_dma_source_entarch.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/dram_memory_destination_entarch.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/dram_memory_source_entarch.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/dram_dma_destination_entarch.vhd"
```

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
--add_file -vhdl -lib pe_lib "$PROJECT_BASE/User_IP_Block_0.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/User_IP_Block_1.vhd"
add_file -vhdl -lib pe_lib "$PE_BASE/pe_ent.vhd"

add_file -vhdl -lib pe_lib "$PROJECT_BASE/CAM_16WORDS.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/CAM_generic_word.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/CAM_SRL16.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/CAM_top.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/COMPARE_4.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/COUNT_16.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/Decode_1.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/Decode_2.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/Decode_3.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/Decode_4.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/Encode_1_MSB.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/Encode_2_MSB.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/Encode_3_MSB.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/Encode_4_LSB.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/Encode_4_MSB.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/INIT_SRL16_AND.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/Virtual Memory Controller.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/tlb.vhd"
add_file -vhdl -lib pe_lib "$PROJECT_BASE/wmu.vhd"

add_file -vhdl -lib pe_lib "$PROJECT_BASE/virtual_socket_testing.vhd"

#-------------------------------------------------------------------------
#-
#-  3. Change the result file:
#-
#-------------------------------------------------------------------------
project -result_file "virtual_socket.edf"
#-------------------------------------------------------------------------
#-
#-  4. Add or change the following options according to your
#-      project's needs:
#-------------------------------------------------------------------------

# State machine encoding
set_option -default_enum_encoding sequential

# State machine compiler
set_option -symbolic_fsm_compiler false

# Resource sharing
set_option -resource_sharing true

# Clock frequency
set_option -frequency 132.000

# Fan-out limit
set_option -fanout_limit 1000

# Hard maximum fanout limit
set_option -maxfan_hard false
```

Figure 115 — A synthesis project file for platform implementation.

Notes:

• This project file includes only IP-Block 1.

• Be careful to add the top design file in last position in the list of the files added.

• Due to the limitation of FPGA synthesis resources, although there are maximum of 32 user IP-blocks can be integrated into the platform, it is strongly suggested that for each user IP-block integration, the corresponding function block of Registers and BRAM resources be integrated into the platform. Otherwise, just leave the idle function blocks of Registers and BRAM resources commented and not performed. For example, if users need to integrate 6 user IP-blocks into the platform, they should take the above-mention steps to add 6 user IP-blocks to the platform as well as uncomment appropriate parts of the code in "*virtual_socket_testing.vhd*" to add corresponding 6 function blocks of Registers and BRAM resources to the platform. After that, users can synthesise the whole platform and generate the final .x86 image file.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.7.2 Phase 2: having .X86 bitstream file.

- **Design Flow**

After all the files are successfully synthesised, users should perform the FPGA routing and image file generation process [5]. The final image file .x86 will be generated and thus downloaded into the FPGA to implement the platform hardware functions.

Copy this directory *\annapolis\wildcard_ii\vhdl\par\bin* in your directory where stands the EDIF file (.EDF). Then execute this batch file "*design_flow.bat*" in a Dos prompt window.

Here is the batch file code :

```
::*********************************************************
:: Batch File
::
:: Author : Christophe Lucarz
:: Laboratory : LAP - EPFL
:: Description : command lines to transform the EDIF (.edf) file into the bitstream ::
file (.X86) using :: the Design Flow of Xilinx
:: Date : June 2006
::*********************************************************

edif2ngd -p xc2v3000fg676-4 virtual_socket_testing.edf virtual_socket_testing.ngo
touch virtual_socket_testing.ucf
xilperl c:/annapolis/wildcard_ii/vhdl/par/bin/pad_loc.pl -i virtual_socket_testing.edf -t
c:/annapolis/wildcard_ii/vhdl/par/pad_loc/pe_pad_loc.dat -o virtual_socket_testing.pad_ucf
rm -f combined.ucf
cat virtual_socket_testing.pad_ucf virtual_socket_testing.ucf > combined.ucf
ngdbuild -sd . -uc combined.ucf virtual_socket_testing.ngo virtual_socket_testing.ngd
rm -f virtual_socket_testing.pcf
map -o virtual_socket_testing.ncd -pr b -k 6 -r virtual_socket_testing
xilperl -p -e "s/SCHEMATIC END ;/ENABLE = REG_SR_CLK ;\nSCHEMATIC END ;/;" virtual_socket_testing.pcf
> virtual_socket_testing.tmp
mv virtual_socket_testing.tmp virtual_socket_testing.pcf
mv virtual_socket_testing.ncd virtual_socket_testing
cp virtual_socket_testing virtual_socket_testing.ncd
par -ol high -w virtual_socket_testing virtual_socket_testing virtual_socket_testing.pcf
trce -v 20 virtual_socket_testing virtual_socket_testing.pcf -o virtual_socket_testing
bitgen -w virtual_socket_testing.ncd virtual_socket_testing virtual_socket_testing.pcf
promgen -w -p hex -o virtual_socket_testing.dum -u 0 virtual_socket_testing
peutil.exe virtual_socket_testing.hex virtual_socket_testing.x86
```

Figure 116 — Code of the batch file for synthesis.

Details on the Xilinx Commands

EDIF2NGD

The EDIF2NGD program allows you to read an EDIF (Electronic Design Interchange Format) 2 0 0 file into the Xilinx development system toolset. EDIF2NGD converts an industry-standard EDIF netlist to an NGO file—a Xilinx-specific format. The EDIF file includes the hierarchy of the input schematic. The output NGO file is a binary database describing the design in terms of the components and hierarchy specified in the input design file. After you convert the EDIF file to an NGO file, you run NGDBuild to create an NGD file, which expands the design to include a description reduced to Xilinx primitives.

NGDBuild

NGDBuild reads in a netlist file in EDIF or NGC format and creates a Xilinx Native Generic Database (NGD) file that contains a logical description of the design in terms of logic elements, such as AND gates, OR gates, decoders, flip-flops, and RAMs.

MAP

The MAP program maps a logical design to a Xilinx FPGA. The input to MAP is an NGD file, which is generated using the NGDBuild program. The NGD file contains a logical description of the design that

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

includes both the hierarchical components used to develop the design and the lower level Xilinx primitives. The NGD file also contains any number of NMC (macro library) files, each of which contains the definition of a physical macro.

PAR

After you create a Native Circuit Description (NCD) file with the MAP program, you can place and route that design file using PAR. PAR accepts a mapped NCD file as input, places and routes the design, and outputs an NCD file to be used by the bitstream generator (BitGen).

TRCE

TRACE provides static timing analysis of an FPGA design based on input timing constraints.

BitGen

BitGen produces a bitstream for Xilinx device configuration. After the design is completely routed, it is necessary to configure the device so that it can execute the desired function. This is done using files generated by BitGen, the Xilinx bitstream generation program. BitGen takes a fully routed NCD (native circuit description) file as input and produces a configuration bitstream—a binary file with a .bit extension.

PROMGen

PROMGen formats a BitGen-generated configuration bitstream (BIT) file into a PROM format file. The PROM file contains configuration data for the FPGA device. PROMGen converts a BIT file into one of three PROM formats: MCS-86 (Intel), EXORMAX (Motorola), or TEKHEX (Tektronix). It can also generate a binary or hexadecimal file format.

PEUTIL

PEUTIL is a PE image file format converter from Annapolis Micro Systems. The supported input file formats are : .mcs, .hex, .x86, .m68, .xh and the output file format are : .x86, .m68 and .xh.

- **Uploading the bitstream file in the FPGA**

In your root directory of your software project, where are all the executable files are, it must have two directories : *\XC2V3000\PKG_FG676.* Inside theses directories, copy the appropriate image file .x86.

In this example, I copy the bitstream file in the following directory :

C:\Digital_Video_Processing_Board_Design\WildCard\VC_WildCard_Program\Debug\XC2V3000\PKG_FG676

More details in the next subclause about the software file structure.

9.8 Building the platform system software

To use the platform system software and issue the virtual socket API function calls (Explicit mode) or the VMW library function calls (Virtual Mode) to trigger the hardware modules to work, we need build a project for compiling all the C source files necessary for the system implementation.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The details of the files to be integrated in the project :

Library Functions Files	
WildCard_Program_Library_Functions.c	includes Virtual Socket API function calls
WildCard_Program_Virtual_Socket.h	C header file for virtual socket API
VMW_library.c	Includes Virtual Memory Extension API function calls
Virtual_Memory_Extension.h	C header file for the VMW library
User main Program Files :	
WildCard_Program_Virtual_Socket.c	User main file for system implementation
IP_using_VMW.c	User main program for the example of copying vectors.

Then we may take the following simple steps to build a project and compile all the files included in the project.

a. Create a new folder under the root directory of an appropriate hard disk, e.g. *C:\Digital_Video_Processing_Board_Design\WildCard.*

b. Utilize the VC++ software system to create a new project based on Win32 console application under that folder. After the new project is created, a subfolder will be generated automatically as the project directory.

For example, if we create a new project named "*VC_WildCard_Program*", then we will have *C:\Digital_Video_Processing_Board_Design\WildCard\VC_WildCard_Program* as the project directory. There should be another subfolder automatically generated by VC++ and thus under this project directory: *C:\Digital_Video_Processing_Board_Design\WildCard\VC_WildCard_Program\Debug*

c. Copy "*WildCard_Program_Virtual_Socket.c*" to the project directory and add it to the project as a C source file. This file contains the main function. (See the note for more information).

d. Copy "*WildCard_Program_Library_Functions.c*" and "*VMW_library.c*" to the project directory and add it to the project as a source file.

e. Copy "*wc_shared.c*" to the project directory and add it to the project as a source file, this file is provided by Annapolis.

f. Copy "*wcapi.lib*" and "*wcapi.dll*" to the project directory and add "*wcapi.lib*" as a library file in the project. Besides, copy "*wcapi.dll*" to the Debug directory.

g. Copy "*WildCard_Program_Virtual_Socket.h*" and "*Virtual_Memory_Extension.h*" to the project directory.

h. Copy "*basetsd.h*" to the project directory, this file is provided by the VC++ software system.

i. Copy "*wc_shared.h*", "*wc_windows.h*" and "*wcdefs.h*" to the project directory, those files are provided by Annapolis.

j. Then build all the files within the project and get the correct compiling and linking information.

k. Create an image file directory:

C:\Digital_Video_Processing_Board_Design\WildCard\VC_WildCard_Program\Debug\XC2V3000\PKG_FG676

Copy appropriate image file .x86 to the image file directory.

l. Run the execution program. This can be done by open a Dos prompt window under operating system. Enter the Debug directory, there should be an .exe file existing, then run it.

m. If the user doesn't like to open a Dos window and type Dos command to run the program, another way is to run the execution program directly under VC++ environment. In this case, the user should create a directory:

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

C:\Digital_Video_Processing_Board_Design\WildCard\VC_WildCard_Program\XC2V3000\PKG_FG676

Copy appropriate image file .x86 to this folder, then execute the program directly under VC++ system environment.

NOTE When users need to use their own C source file as the main program, just replace the "*WildCard_Program_Virtual_Socket.c*" with user's own C source file and call the appropriate API functions defined above. If the user wants to test the Virtual Memory Extension with the example of copying vector, just replace "*WildCard_Program_Virtual_Socket.c*" by "*IP_using_VMW.c*". The user can use this file as template for his own C source file.

9.9 How to use the platform

9.9.1 Virtual socket program : conformance test

We can set up a conformance testing program (*WildCard_Program_Virtual_Socket.exe*) to verify the features in the virtual socket platform. The program is coded in C language. There is a debug menu for the program, and we may interactively use debug menu and choose appropriate options to perform the feature testing.

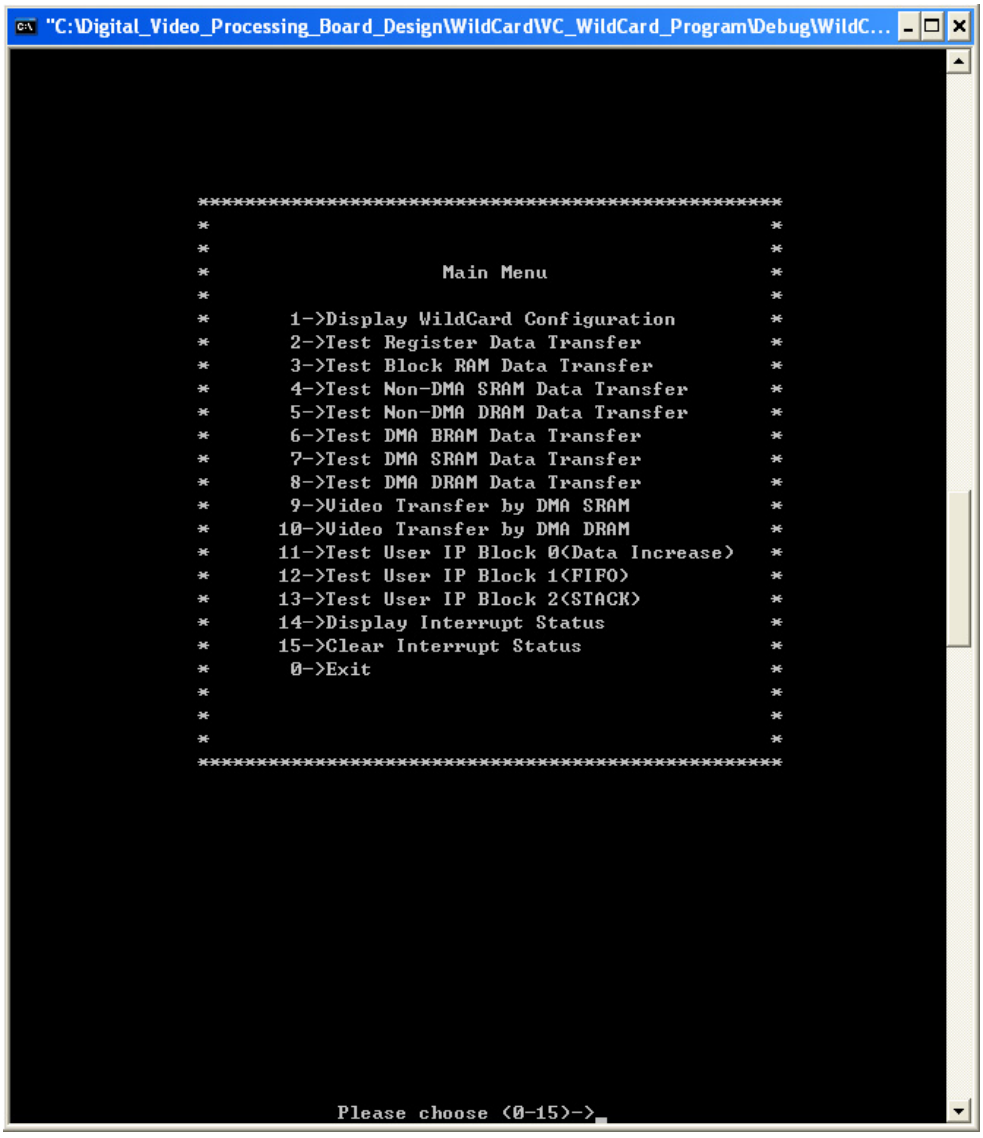


Figure 117 — Debug menu for conformance tests.

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2009 – All rights reserved

Currently, we have 15 feature options included in the debug menu. Those options can test all the 4 types of physical memory space as well as the hardware interface module controller which connects the user IP-blocks to the virtual socket platform.

Option 1 :	Get the configuration information of the WildCard II Platform such as platform hardware version, FPGA type, FPGA package type, FPGA speed grade, RAM size, RAM access speed, API version, Driver version and Firmware version etc.
Option 2	Test general registers.
Option 3	Test Block RAM based on non-DMA operation..
Option 4	Test SRAM based on non-DMA operation.
Option 5	Test DRAM based on non-DMA operation.
Option 6	Test Block RAM based on DMA operation.
Option 7	Test SRAM based on DMA operation.
Option 8	Test DRAM based on DMA operation.
Option 9	Test SRAM video data transfer based on DMA operation. The video frames are in CIF YUV 4:2:0 Format
Option 10	Test DRAM video data transfer based on DMA operation. The video frames are in CIF YUV 4:2:0 Format
Option 11	Test Data Increase Module as user IP-block 0.
Option 12	Test FIFO Module as user IP-block 1.
Option 13	Test STACK Module as user IP-block 2.
Option 14	Display internal user interrupt status register.
Option 15	Cleanup internal user interrupt status register.

9.9.2 Example of copying vectors

Given two tables of a given size and composed of 32 bits words in the main memory on the host, in the virtual memory space : TableA and TableB. We want the IP to copy TableA in TableB. How this trivial task will be treated in the two different modes ?

9.9.2.1 Explicit Mode

The programmer transfers manually the data from the main memory to local memory. To do the transfers, the programmer can use the Virtual Socket functions, especially *VSWrite*, *VSDMAWrite*.

The programmer starts its IP using a function from Virtual Socket API : *HW_Module_Controller.* The IP must generate addresses to get the data from local memory. Theses addresses can be written in the VHDL code of the FSM of the IP. This means you must put the data corresponding to this IP always at the same place in the local memory. Remember that in the Explicit Mode, only one page of the BRAM is available for the IP-Block : for example, the IP-Block n°5 has its BRAM memory space composed of the page n°5, from physical address 0x8a00 to 0x8bff (512 Dwords = 2kB). It has not access to all the BRAM space, as in the Virtual Mode. An other solution to give the addresses to the IP is to write them in the register space and to make a read of the register by the IP and configure it accordingly to the parameters. This implies to make two starts of the IP (have a look to the FSM IP template in Virtual Mode).

To recover the data from the local memory, the programmer can use the Virtual Socket functions, especially *VSRead, VSDMARead*.

9.9.2.2 Virtual Mode

The programmer doesn't care about the memory management. It is the VMW library which manages all the data transfers. You want to treat some data which are on the main memory, on the Host. Nothing is more easy with the Virtual Mode. You just have to pass the addresses as parameters by using the register memory space. Then the IP generates theses addresses and get the data requested. Note we are talking about the addresses of the data in the user software memory space. The programmer doesn't have to deal with any physical address in the local memory.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.10 Understanding VHDL code

The first part “Files system” shows what are the constitutive files of the platform, and which components there are related. The second part details more precisely the addresses of the LQD bus used by the platform. Then, we’ll detail the code of the top design file “*virtual_socket_testing.vhd*” and some others components which needs some explanations.

9.10.1 Files System

The whole virtual socket platform framework consists of the following VHDL programs:

Modules files of the original platform

VHDL Program	Description
data_type_definition_package.vhd	Define some useful data types for platform
bram_dma_source_entarch.vhd	BRAM DMA Components to be instantiated in platform framework
bram_memory_destination_entarch.vhd	
bram_memory_source_entarch.vhd	
bram_dma_destination_entarch.vhd	
dma_source_entarch.vhd	SRAM DMA Components to be instantiated in platform framework
memory_destination_entarch.vhd	
memory_source_entarch.vhd	
dma_destination_entarch.vhd	
dram_dma_source_entarch.vhd	DRAM DMA Components to be instantiated in platform framework
dram_memory_source_entarch.vhd	
dram_memory_source_entarch.vhd	
dram_dma_destination_entarch.vhd	

Modules files of the Virtual Memory Extension

wmu.vhd	WMU component
tlb.vhd	TLB component (inside WMU)
Virtual Memory Controller.vhd	Virtual Memory Controller component
cam_16words.vhd	TLB relative component : Content Addressable Memory (CAM)
cam_generic_word.vhd	
cam_srl16.vhd	
cam_top.vhd	
compare_4.vhd	
count_16.vhd	
decode_1.vhd	
decode_2.vhd	
decode_3.vhd	
decode_4.vhd	

IP-Blocks files and relatives

user_ip_block_0.vhd	IP Blocks 0 Component
user_ip_block_1.vhd	IP Blocks 1 Component
user_ip_block_2.vhd	IP Blocks 2 Component

Top Level Design :

virtual_socket_testing.vhd	Main codes of platform framework

Figure 118 — Sources files of the platform.

--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2009 – All rights reserved
Not for Resale

9.10.2 Top design file

"*virtual_socket_testing.vhdl*" is the main program for platform framework design. All of the hardware features are included in it. The table bellow summarises which process deals with which Functional Block in Figure 94.

Process Name	Related Functional Blocks in the Diagram
U_LAD_CTRL	LAD Bus Interface Controller, Non-DMA Register Controller, Non-DMA BRAM Controller, Non-DMA SRAM Controller, Non-DMA DRAM Controller, Interrupt Status and Mask setup, IP Interface Register Access Controller, IP Interface BRAM Access Controller, IP Interface SRAM Access Controller, IP Interface DRAM Access Controller
U_Arbitrator	IP Block Bus Arbitrator
U_Strobe_Out	LAD Bus Interface Controller
U_DRAM_Data_In_Valid	Non-DMA DRAM Controller, IP Interface DRAM Access Controller
U_SRAM_Mux	SRAM Mux
U_DRAM_Mux	DRAM Mux
U_RAM_0 to U_RAM_31	32 instantiated templates for BRAM Space
U_Int_Gen	Interrupt Chain
U_BRAM_DMA_Source U_BRAM_Memory_Destination U_BRAM_Memory_Source U_BRAM_DMA_Destination	DMA BRAM Controller
U_DMA_Source U_Memory_Destination U_Memory_Source U_DMA_Destination	DMA SRAM Controller
U_DRAM_DMA_Source U_DRAM_Memory_Destination U_DRAM_Memory_Source U_DRAM_DMA_Destination	DMA DRAM Controller
Other Combinational Logic	LAD Bus Interface Controller, BRAM Mux

Figure 119 — Functional Blocks in the virtual_socket_testing.vhd main VHDL file.

9.10.3 Lad Bus

The physical LAD bus addresses used by virtual socket platform are :

0x0000 – 0x3e0f	**Register Space**
See table on memory mapping	
0x8000 – 0xbffff	**BRAM Space**
See table on memory mapping	
0xfff0 – 0xffff	**Platform Control Registers**
0xfff0	INTERRUPT_STATUS
0xfff1	INTERRUPT_MASK
0xfff2	HW_Mem_HWAccel
0xfff6	USER_RESET_DISABLE
0xfff7	DMA_Working
0xfff8	HW_Start
0xfff9	SRAM_Address
0xfffa	SRAM_Start

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

0xfffb	DRAM_Address
0xfffc	DRAM_Data_Low
0xfffd	DRAM_Data_High
0xfffe	DRAM_Start_Write
0xffff	DRAM_Start_Read
0xffb0 – 0xffb5	**IP Block Interface Parameter Control**
0xffb0	Host_Mem_HWAccel
0xffb1	Host_Offset
0xffb2	Host_Count
0xffb3	Host_Mem_HWAccel1
0xffb4	Host_Offset1
0xffb5	Host_Count1
0xffc0 – 0xffcf	**DRAM DMA Control Registers**
0xffc0	bram_dma_source_offset
0xffc4	bram_memory_destination_offset
0xffc6	bram_memory_source_offset
0xffc8	bram_dma_destination_offset
0xffd0 – 0xffdf	**DRAM DMA Control Registers**
0xffd0	dram_dma_source_offset
0xffd4	dram_memory_destination_offset
0xffd6	dram_memory_source_offset
0xffd8	dram_dma_destination_offset
0xffe0 – 0xffef	**SRAM DMA Control Registers**
0xffe0	dma_source_offset
0xffe4	memory_destination_offset
0xffe6	memory_source_offset
0xffe8	dma_destination_offset
0x6000 – 603F	**WMU Control Register**
0x6000	Control register
0x6001	Status register
0x6002	Address register
0x6003	Access Indicator register
0x6004	Access Mask register
0x6020-0x603F	32 TLB entries

Figure 120 — LAD Bus addresses.

9.10.4 The Virtual Memory Controller

The following figure details precisely the FSM of the Virtual Memory Controller.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

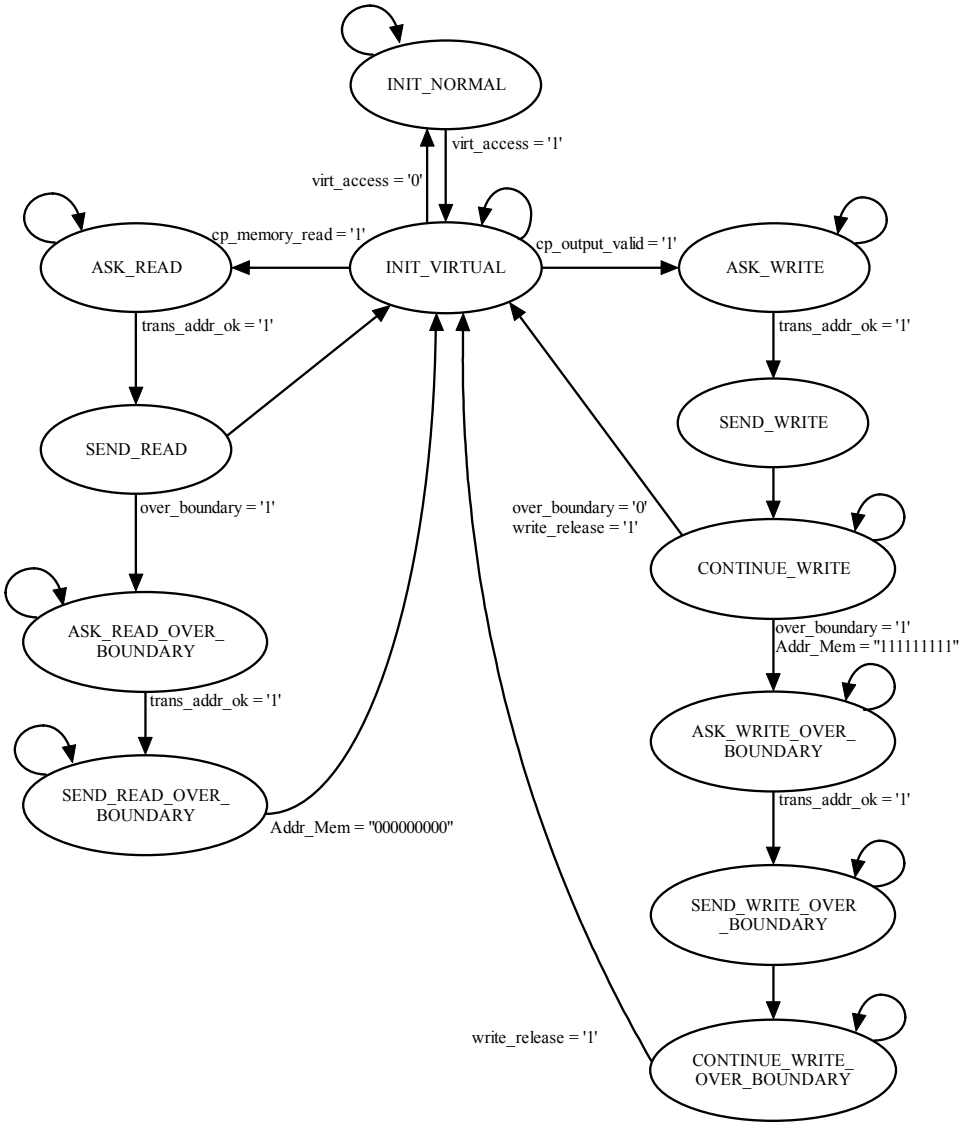


Figure 121 — Virtual Memory Controller Finite State Machine.

Details on the states of the FSM

- **Idle states**

INIT_NORMAL: this is the initial state of the Virtual Memory Controller. In this case, the entries from the IP are directly connected to the platform. In this state, it was like the component doesn't exist to make the platform work in its original way, without Virtual Memory Extension.

INIT_VIRTUAL: when the *virt_mem_access* signal is asserted, the Virtual Memory Controller enters this state. The Virtual Memory Controller detours the signals from the IP to the WMU. When it receives a read request, it goes in the *ask_read* state. When it receives a write request, if goes in the *ask_write* state.

- **Read states**

ASK_READ: the Virtual Memory Controller sends to the WMU the address requested. It goes into the *send_read* state when the address has been translated by the WMU. Otherwise, it stays in the same state.

SEND_READ: the Virtual Memory Controller sends to the platform the offset address, the read request signal (*memory_read*) and the count. If the Virtual Memory Controller detects a cross of the page boundary, it goes in the *ask_read_over_boundary* state, otherwise returns in the *init_virtual* state.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

ASK_READ_OVER_BOUNDARY: the Virtual Memory Controller will send the second part of the initial request (the first part has been already done in the last steps). It sends to the WMU the computed address corresponding to the following page.

SEND_READ_OVER_BOUNDARY: the Virtual Memory Controller sends to the platform the offset address, the read request signal (*memory_read*) and the count for the second part of the initial request. It returns in the *init_virtual* state when the transfer is finished.

- **Write states**

ASK_WRITE: the Virtual Memory Controller sends to the WMU the address requested. It goes into the *send_write* state when the address has been translated by the WMU. Otherwise, it stays in the same state.

SEND_WRITE: the Virtual Memory Controller sends to the platform the offset address, the write request signal (*output_valid*) and the count. It jumps directly to the *continue_write* state.

CONTINUE_WRITE: The signal *output_valid* has two functions. The first is to validate an write request from the IP and the other one is to validate the transferred data. For theses reasons, a new state must be created to distinguish the request of the IP and the transfer of data. This state is to allow the transfer of the data through the interface. If the Virtual Memory Controller detects a cross of the page boundary, it goes in the *ask_write_over_boundary* state, otherwise goes returns in the *init_virtual* state.

ASK_WRITE_OVER_BOUNDARY: the Virtual Memory Controller will send the second part of the initial request (the first part has been already done in the last steps). It sends to the WMU the computed address corresponding to the following page.

SEND_WRITE_OVER_BOUNDARY: the Virtual Memory Controller sends to the platform the offset address, the write request signal (*output_valid*) and the count for the second part of the initial request. It jumps directly to the *continue_write_over_boundary* state.

CONTINUE_WRITE_OVER_BOUNDARY: the state has the same function as the *continue_write* state, it allows the transfer of the data through the interface, for the second part of the request.

9.11 Appendix

9.11.1 Copying vector program

The following is an example of an user program of the copying vector example using the Virtual Memory Extension.

```
#include <stdio.h>
#include <stdlib.h>
#include <memory.h>
#include <time.h>
#include <ctype.h>
#include <dos.h>
#include <conio.h>
#if defined(WIN32)
#include <windows.h>
#include <math.h>
#endif
#include "wcdefs.h"
#include "wc_shared.h"
#include "WildCard_Program_Virtual_Socket.h"
#include "Virtual_Memory_Extension.h"


int main(int argc,char *argv[])
{
/******* User Variables *******/
  unsigned long i, result ;
```


© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
  int unsigned size ;
  DWORD *TableA,*TableB;
  boolean error ;

/*****************************************/
/*       CONFIGURING THE PLATFORM        */
/*****************************************/
  // Virtual Socket
  Platform_Init();
  // Virtual Memory Extension
  VMW_Init() ;


// ***********************************************************************
// *                          USER PROGRAM SPACE
// ***********************************************************************
  printf("\n\n 1/ Size of the vector to copy : ");
  scanf("%d",&size);

  // Allocate space for an array : TableA
  TableA = malloc(sizeof (DWORD) * size);
  if (TableA == NULL)
   exit(EXIT_FAILURE);

  // initialisation table A
  for (i=0 ; i < size ; i++) {
      TableA[i] = rand() | (rand() << 14) | (rand() << 28) ;
      //printf("\n TableA[%d] = %x ", i, TableA[i]);
  }

  // Allocate space for an array : TableB
  TableB = malloc(sizeof (DWORD) * size);
  if (TableB == NULL)
   exit(EXIT_FAILURE);

  // showing the address generated
  printf("\n Adresse de TableA : 0x%x", TableA);
  printf("\n Adresse de TableB : 0x%x", TableB);

  // we set the parameters
  IP_Param.nb_param = 4 ;                    // number of parameters
  IP_Param.Param[0] = 1 ;                    // virtual memory access (virtual = 1 ; normal = 0)
  IP_Param.Param[1] = TableA ;               // reading address
  IP_Param.Param[2] = TableB ;               // writing address
  IP_Param.Param[3] = size ;             // count

  // we start the IP (locking call)
  result = IP_start(1, &IP_Param) ;
  if (result == 0) {
      printf("\n\n IP_start has correctly finished \n");
  }
  else {
      printf("\n\n IP_start didn't finished correctly \n");
  }


  // Check results

  printf("\n **************RESULTS*************\n");
  error = FALSE ;
  for (i=0 ; i < size ; i++) {
      if (TableA[i] != TableB[i]) {
        printf("\n Error at index %d", i);
        error = TRUE ;
      }
  }

  if (error == TRUE) printf("\n Error detected : example failed");
  else printf("\n Vectors are identical - Example Complete");
  printf("\n*********************************\n");

  free(TableA);
  free(TableB);


// ********************  END OF USER PROGRAM SPACE ********************
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
/************************************/
/*       CLOSING THE PLATFORM       */
/************************************/
  // Virtual Memory Extension
  VMW_Stop();
  // Virtual Socket
  Platform_Stop();


}
```

Figure 122 — Copying vector program.

9.12 Glossary

B

BRAM memory space

The BRAM space is one the four type of memory available on the platform. The BRAM space is one of the two on-chip memory space, the other one being the register space. In the Virtual Memory Extension, the BRAM is the one used for the automatic data transfers. SRAM and DRAM are not used in the extended platform.

C

CAM

Unlike standard computer memory (RAM) in which the user supplies a memory address and the RAM returns the data word stored at that address, a Content Addressable Memory is designed such that the user supplies a data word and the CAM searches its entire memory to see if that data word is stored anywhere in it. If the data word is found, the CAM returns the address where the word was found.

D

Dirty page

A dirty page is relative to the BRAM memory space. In the Virtual Mode, a page of the BRAM is dirty is when this page has been modified by one IP-Block and not copied back in the main memory of the host.

E

Extended platform

The extended platform is in fact the Virtual Socket platform with the virtual memory feature.

H

Hardware call function

This is the software function which is used to call the hardware IP-Block. The hardware function call is here named *Start_IP*.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Host

The host is here the personal computer (PC). Given that the IP-Block must be tested in a real environment (i.e. the C algorithm), the IP-Block must be started in the algorithm which is coded in C and is executed on the PC.

I

IP – Block

IP-Block stands for *Intellectual Property Block*. The user IP-block can be any hardware module designed to handle a computationally extensive task for MPEG4 applications. It is composed of a core and a wrapper to have the appellation IP-Block. Each IP-Block has the same interface to communicate with the platform.

IP request

When the IP asks for data access (read or write) it generates four signals. For a read request, the IP will assert the following signals:

- *hw_mem_hwaccel*: IP number
- *hw_offset:* reading address
- *hw_count:* number of data to read
- *memory_read:* strobe signal of the request

For a write request, the IP will assert the following signals:

- *hw_mem_hwaccel1* : IP number
- *hw_offset1:* writing address
- *hw_count1:* number of data to write
- *output_valid:* strobe signal of the request and the data.

L

Local Address Data Bus (LAD)

Bus between the CardBus controller and the Processing Element (PE). The LAD Bus is the primary means of communicating with the host system. Using the LAD bus, the PE can send and receive programmed I/O and DMA data to and from the host.

O

Original platform

One calls original platform to talk about the Virtual Socket platform before the Virtual Memory Extension.

P

Platform

We call platform the link between the host-PC and the IP-Blocks. It is composed of local memories and controllers to handle data transfers between the host and the local memory and between the local memory and the IP-Blocks.

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Physical address

The physical address is relative to the local memory space of the FPGA, on the card. The physical address is composed of the offset in one page of the BRAM space and the page number. The address is generated by the WMU which transforms the virtual address to get it.

S

Virtual Memory Controller

The Virtual Memory Controller is an intermediate component between the Hardware Interface Controller of the platform, the WMU and the IP-Block. The Virtual Memory Controller looks sort of like a hub: it manages Virtual Memory features.

T

TLB

A **Translation Lookaside Buffer** is a buffer (or cache) that contains parts of the page table which translate from virtual into physical addresses. This buffer has a fixed number of entries and is used to improve the speed of virtual address translation. The buffer is typically a content addressable memory (CAM) in which the search key is the virtual address and the search result is a physical address. If the CAM search yields a match the translation is known and the match data are used. If no match exists, the VMW library will intervene to copy the data required and write the new entry in the page table in the TLB.

U

UoC platform

University of Calgary Platform has the same meaning as the Virtual Socket Platform.

V

Virtual Socket

The Virtual Socket is the platform developed by the University of Calgary to aid in the rapid emulation, test, and verification of FPGA designs for complex multimedia standards.

Virtual address

The virtual address is relative to the host virtual memory space. These addresses are generated by the IP to get data. Thanks to the platform and its extension, the virtual address is handled by hardware and software and data are sent to the IP without any intervention of the programmer. The address is transformed into the physical address thanks to the WMU.

Virtual Memory Extension

It is the extension of the original platform; it consists in the possibility for the IP-Block to use virtual addresses to get data on the host memory space.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

W

WMU

The **Window Manager Unit** is a hardware component responsible for handling virtual memory accesses requested by the IPs. It operates the translation of the virtual address into the physical address.

10 HDL MODULES

10.1 INVERSE QUANTIZER HARDWARE IP BLOCK FOR MPEG-4 PART 2

10.1.1 Abstract description of the module

This subclause documents a high performance implementation of an MPEG-4 Inverse Quantizer (INVQ) in a Virtex™-II FPGA.

10.1.2 Module specification

10.1.2.1 MPEG 4 part: ***2***
10.1.2.2 Profile: ***All***
10.1.2.3 Level addressed: ***All***
10.1.2.4 Module Name: ***INVQ***
10.1.2.5 Module latency: ***2 clock cycles***
10.1.2.6 Module data throughput: ***1.48M blocks/sec***
10.1.2.7 Max clock frequency: ***98MHz***
10.1.2.8 Resource usage:
10.1.2.8.1 CLB Slices: ***318***
10.1.2.8.2 Slices Flip Flops: ***80***
10.1.2.8.3 4 Input LUTS: ***578***
10.1.2.8.4 Multipliers: **11**
10.1.2.8.5 External memory: ***none***
10.1.2.9 Revision: ***1.00***
10.1.2.10 Authors: ***A. Navarro, A. Silva, O. Nunes, C. Aragao***
10.1.2.11 Creation Date: ***25/06/2004***
10.1.2.12 Modification Date: ***12/10/2004***

10.1.3 Introduction

With this hardware solution the MPEG-4 Inverse Quantization of blocks of 64 elements is performed in about 0.6µs, using 2% of all available slices in the Virtex™-II XC2V3000-4 FPGA [1]. We have simulated InvQ module using the designing tool (ISE) 6.1. This module can be integrated with others in order to efficiently decode MPEG-4 video.

10.1.4 Functional Description

10.1.4.1 Functional description details

Next figure shows the implementation of the MPEG-4 Inverse Quantizer. It is assumed that the circuit is fed by input quantized coefficients (data_in) and outputs (data_out) dequantized serial coefficients.

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.1.4.2 I/O Diagram

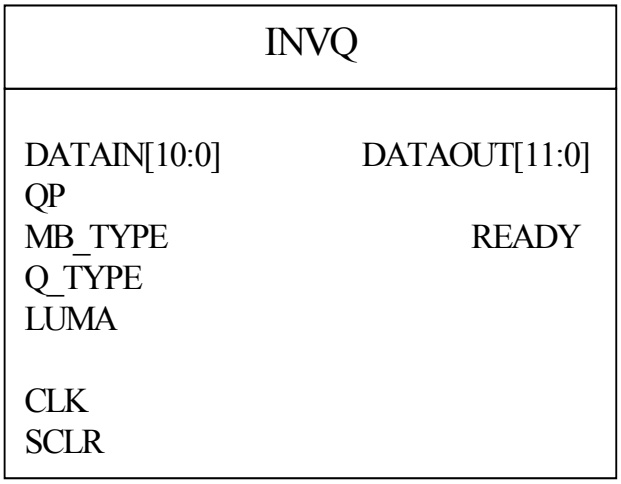


Figure 123 — MPEG-4 Inverse Quantizer Block Diagram.

10.1.4.3 I/O Ports Description

Table 26

Port Name	Port Width	Direction	Description
DATAIN[10:0]	11	Input	Quantized DCT coefficient data input
QP	1	Input	Quantization parameter
MB_TYPE	1	Input	Macroblock type flag
Q_TYPE	1	Input	Quantization type flag
LUMA	1	Input	Luma Blocks flag
CLK	1	Input	System clock
SCLR	1	Input	System sync reset
DATAOUT[B:0]	12	Output	Block data out
READY	1	Output	Valid data at DATAOUT

10.1.5 Algorithm

Quantization consists of selectively discarding visual information without introducing significant visual loss. The quantization process removes viewer's imperceptible visual information. The selective discarding of visual information, which is ignored by the human visual system, represents one of the key processes in image and video compression systems, reducing storage requirements and improving bandwidth. Besides, quantization reduces the number of bits required to represent the DCT coefficients and is the primary source of data loss in image/video compression algorithms. However, lesser loss is only possible with a perfect match between the source statistics and the quantization function [1][2]. A quantizer can be either a constant scalar to be applied to a set of DCT coefficients or an 8x8 matrix in which each of its elements is applied to each spatial corresponding coefficient.

When the DCT is applied to an 8x8 block of pixels, the result is a set of spatial frequency components. Since the human visual system is less sensitive to higher frequency details than lower frequency, the reduction (quantization) of the accuracy of the higher spatial frequency does not affect the reconstructed image quality significantly and thus an additional compression is achieved. Similarly, since the human visual system is less

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

sensitive to colour components than to brightness (luminance), quantization of colour components can be coarser.

Due to the quantization process, the lower value coefficients will tend to zero. The resulting zero coefficients will be properly encoded. Every loss video coding standard employs a quantization block. Let us now move to describe the quantization functions within MPEG-4 framework.

10.1.5.1 MPEG-4 Quantization

In MPEG-4 [3], it is possible to apply two quantization processes. The first, "MPEG Quantization" (subclause 10.1.5.3) is derived from the MPEG-2 video standard, and the second, "H.263 Quantization" (subclause 10.1.5.4), was used in recommendation ITU-T H.263. At the encoder side, it is decided which of the two methods is used, and the quantization method used is sent to the decoder as side information. In addition, the DC coefficient of an 8x8 block coded in INTRA mode is quantized using a fixed quantizer step size.

The quantization step size is controlled by a specific parameter: the quantizer_scale, Qp, which can take values from 1 to $2^{\text{quant_precision}} - 1$, and it is encoded once per VOP. The parameter quant_precision specifies the number of bits used to represent quantizer parameters and can assume values between 3 and 9. If the parameter not_8_bit is set to 0, meaning no transmission of quant_precision, then quant_precision assumes the default value of 5.

Before IDCT takes place, the resulting coefficients, from the Inverse Quantization F"[i][j], are saturated, as expressed by,

$$F'[i][j] = \begin{cases} 2^{\text{bits_per_pixel}+3} - 1, & \text{if } F''[i][j] > 2^{\text{bits_per_pixel}+3} - 1 \\ F''[i][j], & -2^{\text{bits_per_pixel}+3} \le F''[i][j] \le 2^{\text{bits_per_pixel}+3} - 1 \\ -2^{\text{bits_per_pixel}+3}, & \text{if } F''[i][j] < -2^{\text{bits_per_pixel}+3} \end{cases}$$

Equation 1

10.1.5.2 Intra DC Coefficient Quantization

The DC coefficients of INTRA coded macroblocks (MBs) are quantized using an optimized, nonlinear quantization method, where the value of the quantization step size, dc_scaler, is a function of Qp, as shown in Table 27.

Table 27 — Quantization step size, dc_scaler.

quantizer_scale, Qp	1 - 4	5 - 8	9 - 24	25 – 31
dc_scaler(luminance)	8	2Qp	$Qp + 8$	$2Qp - 16$
dc_scaler(chrominance)	8	$(Qp + 13)/2$	$(Qp + 13)/2$	$Qp - 6$

The DC InvQ is then carried out as follows:

$$F''[0][0] = QF[0][0] \text{dc_scaler}$$

Equation 2

where QF[0][0] denotes the quantized DC coefficients.

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.1.5.3 MPEG Quantization

As mentioned above, the advantage of MPEG quantization is that the encoder can take into account the properties of the human visual system. Thus, the MPEG quantization method allows the adaptation of the quantization step size individually for each transform coefficient through the use of weighting matrices.

MPEG-4 defines different quantization matrices for INTRA and for INTER coded macroblocks, as shown bellow, in Figure 123. Furthermore, either default matrices or new defined matrices can be applied. In the latter case, the new matrices are transmitted to the receiver.

$$\begin{bmatrix} 8 & 17 & 18 & 19 & 21 & 23 & 25 & 27 \\ 17 & 18 & 19 & 21 & 23 & 25 & 27 & 28 \\ 20 & 21 & 22 & 23 & 24 & 26 & 28 & 30 \\ 21 & 22 & 23 & 24 & 26 & 28 & 30 & 32 \\ 22 & 23 & 24 & 26 & 28 & 30 & 32 & 35 \\ 23 & 24 & 26 & 28 & 30 & 32 & 35 & 38 \\ 25 & 26 & 28 & 30 & 32 & 35 & 38 & 41 \\ 27 & 28 & 30 & 32 & 35 & 38 & 41 & 45 \end{bmatrix}$$

(a)

$$\begin{bmatrix} 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 \\ 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 \\ 18 & 19 & 20 & 21 & 22 & 23 & 24 & 25 \\ 19 & 20 & 21 & 22 & 23 & 24 & 26 & 27 \\ 20 & 21 & 22 & 23 & 25 & 26 & 27 & 28 \\ 21 & 22 & 23 & 24 & 26 & 27 & 28 & 30 \\ 22 & 23 & 24 & 26 & 27 & 28 & 30 & 31 \\ 23 & 24 & 25 & 27 & 28 & 30 & 31 & 33 \end{bmatrix}$$

(b)

Figure 124 — Default weighting matrices in MPEG Quantization for: (a) INTRA coded MBs, (b) INTER coded MBs.

The inverse quantization is performed according to the following equation:

$$F''[i][j] = \begin{cases} 0, & \text{if } QF[i][j] = 0 \\ ((2.QF[i][j] + k) \times W[i][j] \times \text{quantiser_scale})/16, & \text{if } QF[i][j] \neq 0 \end{cases}$$

Equation 3

where:

$$k = \begin{cases} 0 & \text{for intra coded blocks} \\ \text{sign}(QF[i][j]) & \text{for inter coded blocks} \end{cases}$$

QF[i][j] denotes the quantized coefficients. W[i][j] is the weighting matrix.

In this quantization method, it should be applied mismatch control. All reconstructed and saturated coefficients F'[i][j] in the block shall be summed. This value is tested, and a change to coefficient F'[7][7] shall be made according to,

$$F[7][7] = \begin{cases} F'[7][7], & \text{if sum is odd} \\ \left.\begin{cases} F'[7][7] - 1, & \text{if } F'[7][7] \text{ is odd} \\ F'[7][7] + 1, & \text{if } F'[7][7] \text{ is even} \end{cases}\right\} & \text{if sum is even} \end{cases}$$

Equation 4

10.1.5.4 H.263 Quantization

This method does not apply the weighting matrix technique and therefore the computational complexity is decreased. Nevertheless, it does not achieve as good performance as the previous method. It does not allow the optimization of the encoder through the application of adaptive quantization inside an 8x8 block, since the quantization step is the same for all the coefficients (frequency) in a block.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The inverse quantization follows the equation,

$$F''[i][j] = \begin{cases} 0, & \text{if } QF[i][j] = 0 \\ (2.|QF[i][j]|+1) \times \text{quantiser_scale}, & \text{if } QF[i][j] \neq 0, \text{quantiser_scale is odd} \\ ((2.|QF[i][j]|+1) \times \text{quantiser_scale}) - 1, & \text{if } QF[i][j] \neq 0, \text{quantiser_scale is odd} \end{cases}$$

Equation 5

Then the sign of QF[i][j] is incorporated according to,

$$F''[i][j] = \text{Sign}(QF[i][j]) \times |F''[i][j]|$$

Equation 6

10.1.6 Implementation

Figure 125 shows the structure of the MPEG-4 Inverse Quantizer.

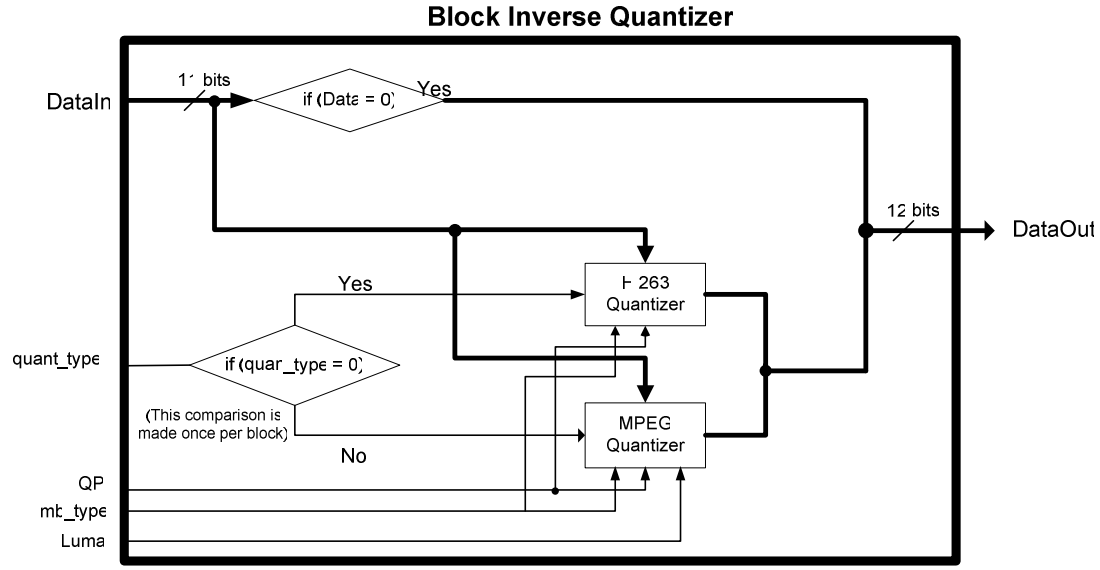


Figure 125 — MPEG-4 Inverse Quantizer Structure.

10.1.6.1 Interfaces

Next figure shows the implementation of the MPEG-4 Inverse Quantizer. It is assumed that the circuit is fed by input quantized coefficients (data_in) and outputs (data_out) dequantized serial coefficients.

© ISO/IEC 2009 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

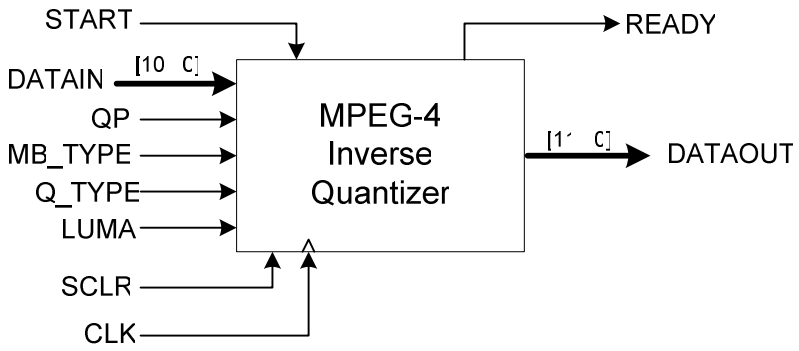


Figure 126 — MPEG-4 Inverse Quantizer Block Diagram.

SCLR – at level high reset all the flip-flops on the design.
CLK = 98.056MHz.
START – is asserted to high to starts the reading of data. This signal is asserted when the input data (DATAIN) pins are valid, and remains in level high while reading the 64 elements.
QP – Quantizer scale
MB_TYPE – Indicates if the block type (Intra or Inter)
Q_TYPE – Indicates quantization type (H.263 or MPEG-4)
LUMA – Indicates if the block is of luminance
READY – is asserted to high when the inverse quantization of the elements of a block is complete.

10.1.6.2 Timing Diagrams
The timing diagram is shown in next figure,

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2009 – All rights reserved
Not for Resale

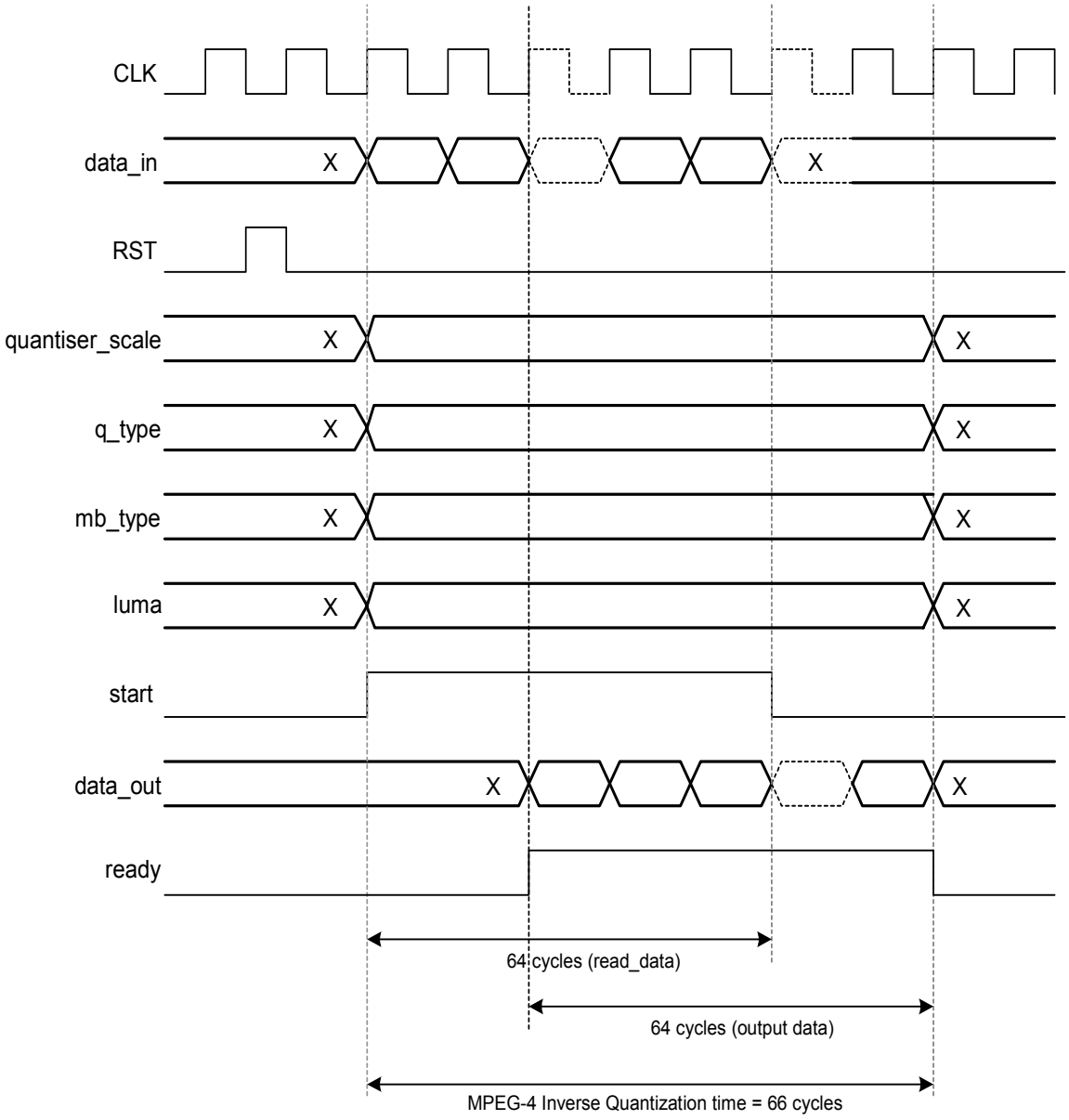


Figure 127 — Timing Diagram.

10.1.7 Results of Performance & Resource Estimation

The design tool used in this work was ISE 6.1. The obtained report from the synthesis tool, XST, is presented below:

Device utilization summary:

Selected Device : 2v3000fg676-4

Number of Slices:	*318*	*out of*	*14336*	*2%*
Number of Slices Flip Flops	*80*	*out of*	*28672*	*0%*
Number of 4 input LUTs:	*578*	*out of*	*28672*	*2%*
Number of MULT18X18s:	*11*	*out of*	*96*	*11%*

Timing Summary:

Speed Grade: -4

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Minimum period: 10.918ns (Maximum Frequency: 98.056MHz)
Minimum input arrival time before clock: 33.798ns
Maximum output required time after clock:5.446ns
Maximum combinational path delay: No path found

10.1.8 API calls from reference software

N/A

10.1.9 Conformance Testing

10.1.9.1 Reference software type, version and input data set
N/A

10.1.9.2 API vector conformance
N/A

10.1.9.3 End to end conformance (conformance of encoded bitstreams or decoded pictures)
N/A

10.1.10 Limitations

The performance of the quantizer could be optimized, performing the computations in parallel.

10.2 2-D IDCT HARDWARE IP BLOCK FOR MPEG-4 PART 2

10.2.1 Abstract description of the module

The Inverse Discrete Cosine Transform (IDCT) is one of the most computation-intensive parts of video coding/decoding process. Therefore, a fast hardware based IDCT implementation is crucial to speed-up real time video processing.

10.2.2 Module specification

10.2.2.1 MPEG 4 part:	**2 (Video)**
10.2.2.2 Profile:	***All Natural Video Profiles***
10.2.2.3 Level addressed:	***All***
10.2.2.4 Module Name:	***IDCT***
10.2.2.5 Module latency:	***64 clocks***
10.2.2.6 Module data throughput:	**1.52 *M blocks/sec***
10.2.2.7 Max clock frequency:	**194.128 MHz**
10.2.2.8 Resource usage:	
10.2.2.8.1 CLB Slices:	***9806***
10.2.2.8.2 Block RAMs:	**none**
10.2.2.8.3 Multipliers:	***64***
10.2.2.8.4 External memory:	***none***
10.2.2.8.5 4 input LUTs	***18535***
10.2.2.9 Revision:	***1.0***
10.2.2.10 Authors:	***A. Navarro, A. Silva, O. Nunes, C. Aragao, M. Santos***
10.2.2.11 Creation Date:	***9/06/2004***
10.2.2.12 Modification Date:	***14/04/2005***

10.2.3 Introduction

This subclause describes a high performance IDCT implementation in a Virtex™-II FPGA [1]. Our solution performance for an 8x8 coefficients is about 51.77ns with an occupation at most of 64% of all available hardware resources in the FPGA and with a precision satisfying IEEE Standard 1180-1990 [10] and Annex A of [11].

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.2.4 Functional Description

10.2.4.1 I/O Diagram

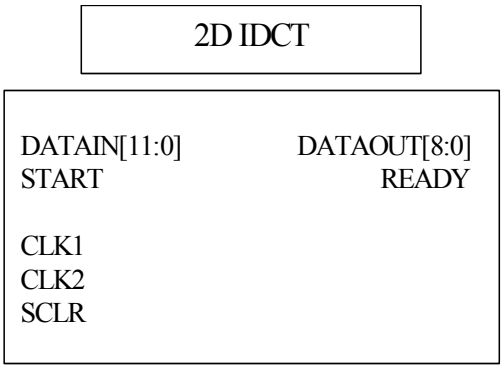


Figure 128

10.2.4.2 I/O Ports Description

Table 28

Port Name	Port Width	Direction	Description
DATAIN[11:0]	12	Input	DCT coefficient data input
START	3	Input	Input data mode flags
CLK	1	Input	System clock
RST	1	Input	System sync reset
DATAOUT[8:0]	9	Output	Block data out
READY	1	Output	Valid data pixels at DATAOUT

10.2.5 Algorithm

10.2.5.1 Discrete Cosine Transform

Most of the hybrid motion compensated video coding standards use a well known discrete cosine transform (DCT) at the encoder to remove redundancy from video random processes. Being the central part of many image coding applications, all DCT based video algorithms or standards will benefit from a DCT (IDCT) fast computation. Several floating-point DCT (IDCT) calculation algorithms have been proposed, and usually can be classified into two classes: indirect and direct methods. The former computes the DCT through a FFT or other transforms and the latter through matrix factorization or recursive computation.

When direct methods are chosen to calculate (NxN)-point 2-D DCTs, the conventional approach follows the row-column method which requires 2N sets of N-point 1-D DCTs. However, true 2-D techniques are more efficient than the conventional row-column approach. Feig and Winograd [11] proposed a matrix factorization algorithm of 2-D DCT matrix which is, as far as we know the fastest 2-D DCT algorithm.

Feig and Winograd proposed an algorithm for the factorization of the DCT matrix. According to [11], the DCT matrix can be represented as a matricial product given by:

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

$$C = D.P.B_1.B_2.M.A_1.A_2.A_3$$

Equation 7

where D is a diagonal matrix whose diagonal elements are {0.3536; 0.2549; 0.2706; 0.3007; 0.3536; 0.4500; 0.6533; 1.2815}. M is also composed of real values $\gamma(k)=\cos(\pi k/8)$ and P is a permutation matrix. The matrices needed to perform Equation 7 are given by:

$$B_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 \end{bmatrix} \quad B_2 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 \end{bmatrix},$$

$$A_1 = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, A_2 = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix},$$

Equation 8

$$A_3 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad M = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \gamma_4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -\gamma_2 & 0 & -\gamma_6 & 0 \\ 0 & 0 & 0 & 0 & 0 & \gamma_4 & 0 & 0 \\ 0 & 0 & 0 & 0 & -\gamma_6 & 0 & \gamma_2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

where $\gamma_i = \cos\left(\frac{2\pi i}{32}\right)$.

The computation of the 2-D DCT on 8x8 points involves the product of the matrix $C \otimes C$ given by,

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2009 – All rights reserved
Not for Resale

$$C \otimes C = (D.P.B_1.B_2.M.A_1.A_2.A_3) \otimes (D.P.B_1.B_2.M.A_1.A_2.A_3)$$

Equation 9

with a 64-pixel vector X_{64}. A standard result about tensor products allows us to rearrange Equation 9 into

$$C \otimes C = (D \otimes D)(P \otimes P)(B_1 \otimes B_1)(B_2 \otimes B_2)(M \otimes M)(A_1 \otimes A_1)(A_2 \otimes A_2)(A_3 \otimes A_3)$$

Equation 10

The matrix factorization for the 2-D DCT proposed by Feig-Winograd can be re-arranged in order to compute the 2-D IDCT.

10.2.5.2 Feig-Winograd IDCT algorithm

Matrix C is orthogonal since its inverse is equal to its transpose. Furthermore, as D is diagonal and M is symmetric, we have:

$$C^{-1} = A_3^T.A_2^T.A_1^T.M.B_2^T.B_1^T.P^T.D$$

Equation 11

The 2-D IDCT can be calculated as:

$$(C^{-1} \otimes C^{-1}).X_{64} =$$
$$(A_3^T \otimes A_3^T)(A_2^T \otimes A_2^T)(A_1^T \otimes A_1^T)(M \otimes M)(B_2^T \otimes B_2^T)(B_1^T \otimes B_1^T)(P^T \otimes P^T)(D \otimes D).X_{64}$$

Equation 12

and since P is a permutation matrix, Equation 12 can be transformed into:

$$(C^{-1} \otimes C^{-1})X_{64} =$$
$$(A_3^T \otimes A_3^T)(A_2^T \otimes A_2^T)(A_1^T \otimes A_1^T)(M \otimes M)(B_2^T \otimes B_2^T)(B_1^T P^T \otimes B_1^T P^T)(D \otimes D)$$

Equation 13

where X64 is the 64-point vector with DCT coefficients.

The above equation yields an algorithm for the inverse scaled-DCT computation. Multiplication by $D \otimes D$ is a simple pointwise multiplication (which can be incorporated in pre-processing stages), $P^T \otimes P^T$ is a matrix permutation and multiplication of $B_1^T \otimes B_1^T$, $B_2^T \otimes B_2^T$, $A_1^T \otimes A_1^T$, $A_2^T \otimes A_2^T$, $A_3^T \otimes A_3^T$ by X_{64} involves only additions, altogether requires 416 additions. Multiplication by $M \otimes M$ needs 54 multiplications, 6 shifts and 46 additions. All together, the algorithm requires 54 multiplications, 462 additions and 6 shifts. A more detailed explanation about the computation of the IDCT can be found in [11].

Efficient implementation of the IDCT requires fixed-point implementations resulting in less silicon area and power consumption. However, in fixed-point implementation, there is an inherent accuracy problem due to finite word length.

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The elements of matrix M are real numbers. Therefore, the multiplications involving the matrix $\mathrm{M} \otimes \mathrm{M}$ are replaced by a sequence of sums and shifts. Several precisions to these constants were performed and tested in order to produce an IDCT implementation which fulfils the conditions imposed by the IEEE standard.

The elements of matrix M, are approximated by [12]:

$$\gamma_2 = 1 - 2^{-4} - 2^{-6} + 2^{-9} + 2^{-14}$$
$$\gamma_4 = 1 - 2^{-2} - 2^{-5} - 2^{-7} - 2^{-8} + 2^{-14}$$
$$\gamma_6 = 2^{-2} + 2^{-3} + 2^{-7} - 2^{-13}$$
$$\gamma_2 + \gamma_6 = 1 + 2^{-2} + 2^{-4} - 2^{-8} + 2^{-9}$$
$$\gamma_2 - \gamma_6 = 2^{-1} + 2^{-5} + 2^{-7} + 2^{-9} + 2^{-13} + 2^{-14}$$
$$\gamma_4 / 2 = 2^{-2} + 2^{-3} - 2^{-6} - 2^{-8} - 2^{-9} + 2^{-13}$$
$$\gamma_4 \gamma_6 = 2^{-2} + 2^{-6} + 2^{-8} + 2^{-10} + 2^{-14}$$

Equation 14

10.2.6 Implementation

Figure 129 shows the structure of our IDCT implementation.

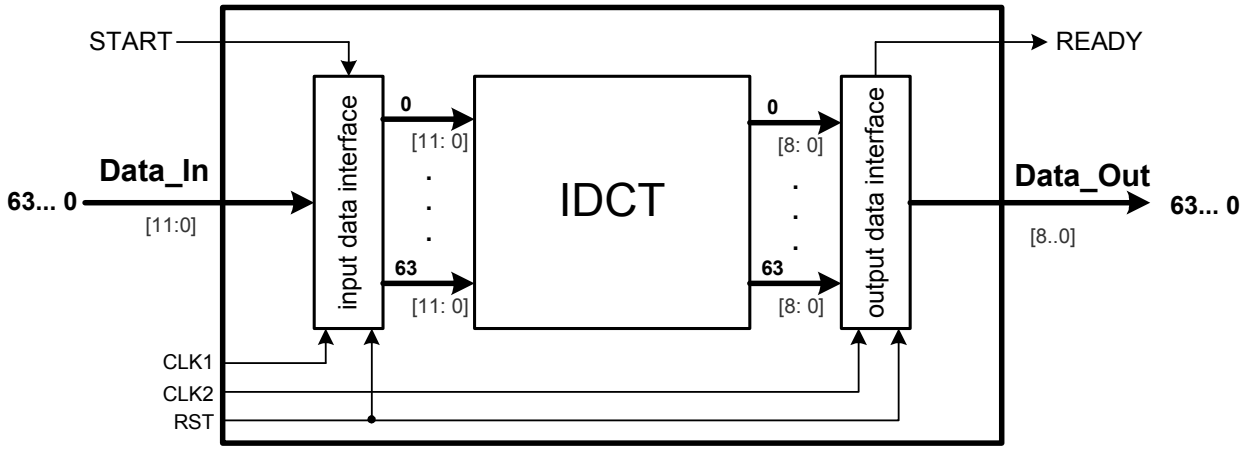


Figure 129 — IDCT Block Diagram.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

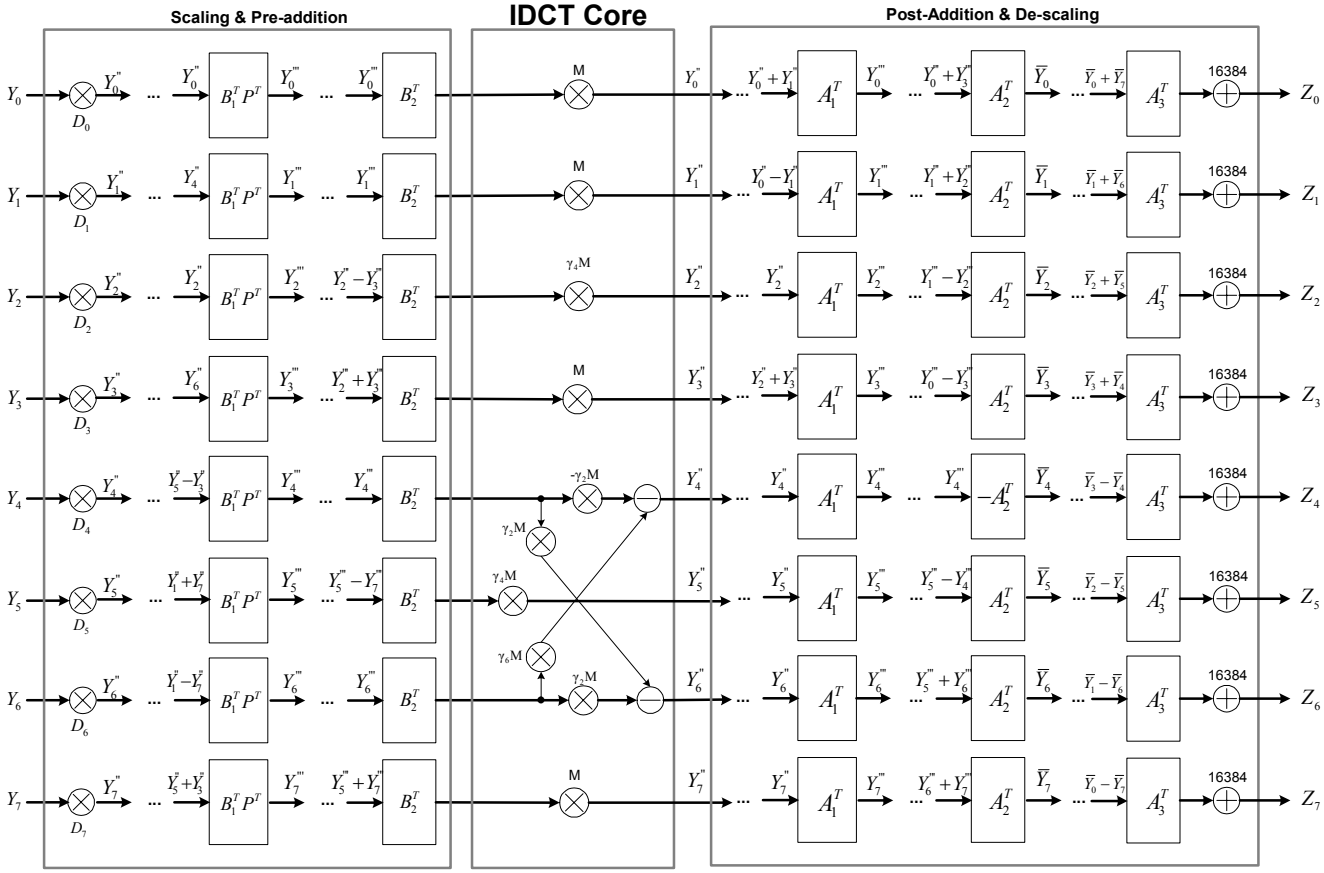


Figure 130 — IDCT Implementation on the FPGA.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

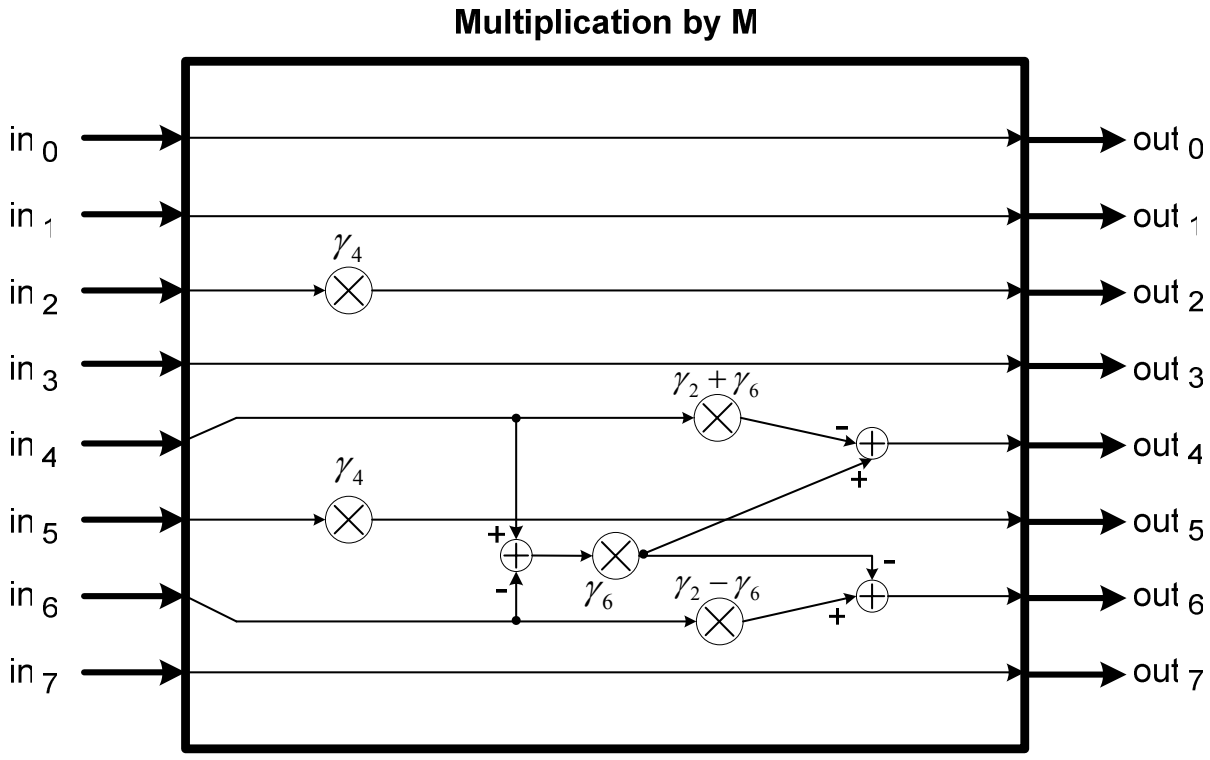


Figure 131 — Implementation of the multiplication of M by an 8-point vector.

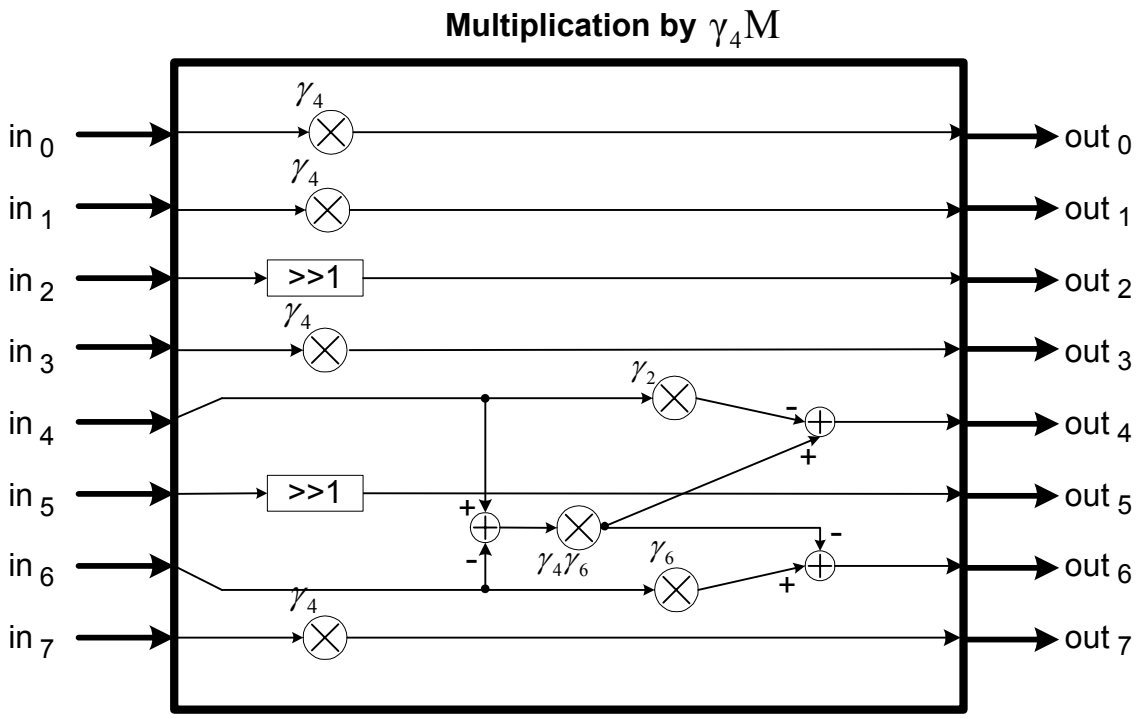


Figure 132 — Implementation of the multiplication of γ_4 M by an 8-point vector.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2009 – All rights reserved
Not for Resale

10.2.6.1 Interfaces

RST – at level high reset all the flip-flops on the circuit.
START – is asserted to high to starts reading data. This signal is asserted when the input data (Data_in) pins are valid and remains in level high while reading all 64 elements.
READY – is asserted to high when the IDCT computation is complete and output data (Data_out) pins are valid.
CLK1 = 219.250MHz.
CLK2 = 194.128MHz.

Firstly, the input data go through an input interface transforming serial data into parallel (converts 64 serial coefficients into parallel). As long as input data elements are available, the IDCT block loads and computes them in pipelining. Once the IDCT is computed, it follows a parallel-to-serial conversion which transforms the 64 parallel elements into 9 bit-64 serial elements.

We should note that input/output data elements are processed in series since this is the common approach in practical implementations.

10.2.6.2 Timing Diagrams

The timing diagram is shown in Figure 133, below.

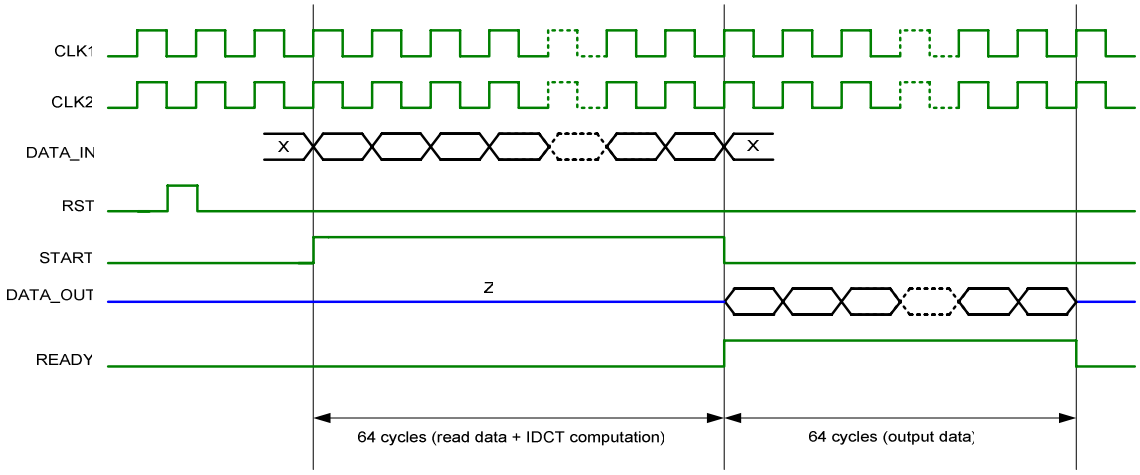


Figure 133 — Timing Diagram.

10.2.7 Results of Performance & Resource Estimation

Since our IDCT implementation is based on fixed-point arithmetic, the internal precision of the computations was adjusted in order to produce an IDCT implementation compliant with the IEEE 1180-1990 Standard [10] with the modifications provided by MPEG-4 Annex A of [3]. The standard specification of the IDCT function [10] defines a set of input data and requires that an IDCT implementation satisfies a set of conditions. The following table show the IDCT precision obtained in our implementation.

© ISO/IEC 2009 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 29 — Precision results of the proposed IDCT implementation according IEEE 1180-1990 Standard conditions.

IEEE 1180-1990 Test			Results
Test	**Interval**	**Error type**	
1	[-256, +255] Sign = +1	Peak Error	1
		Worst mse	0.020960
		Overall mse	0.017245
		Worst mean error	0.000603
		Overall mean error	0.000102
2	[-5 ,+5] Sign = +1	Peak Error	1
		Worst mse	0.000517
		Overall mse	0.000384
		Worst mean error	0.000385
		Overall mean error	0.000007
3	[-384, +383] Sign = +1	Peak Error	1
		Worst mse	0.020231
		Overall mse	0.016314
		Worst mean error	0.000483
		Overall mean error	0.000084
4	[-256, +255] Sign = -1	Peak Error	1
		Worst mse	0.020946
		Overall mse	0.017223
		Worst mean error	0.000435
		Overall mean error	0.000069
5	[-5 ,+5] Sign = -1	Peak Error	1
		Worst mse	0.000503
		Overall mse	0.000383
		Worst mean error	0.000375
		Overall mean error	0.000007
6	[-384, +383] Sign = -1	Peak Error	1
		Worst mse	0.020234
		Overall mse	0.016316
		Worst mean error	0.000379
		Overall mean error	0.000051

By using the approach of performing the IDCT calculation in parallel, computation time of all 64 DCT coefficients is limited by the maximum combinational path delay, which in this case is 51.77ns.

The design tool used in this work was ISE 6.1. The report of the synthesis tool, XST, obtained is presented below:

Device utilization summary:

Selected Device : 2v3000fg676-4

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
Number of Slices:                  9285   out of   14336    64%
Number of 4 input LUTs:           18120   out of   28672    63%
Number of MULT18X18s:                64   out of      96    66%

Timing Summary:
---------------
Speed Grade: -4
Minimum period: No path found
Minimum input arrival time before clock: No path found
Maximum output required time after clock: No path found
Maximum combinational path delay: 51.767ns
```

10.2.8 API calls from reference software

```
if (WC_rc == WC_SUCCESS) {
        /* READ AND WRITE */

        // Resets WC_IDCT
        pWriteBuffer[0] = 0x21;
        WC_rc = WC_PeRegWrite(TestInfo.DeviceNum, 0, 1, &pWriteBuffer);
        pWriteBuffer[0] = 0x01;
        WC_rc = WC_PeRegWrite(TestInfo.DeviceNum, 0, 1, &pWriteBuffer);

        for (index=0; index < 64 ; index ++)
        {
        pWriteBuffer[0] = 0x11;
        WC_rc = WC_PeRegWrite(TestInfo.DeviceNum, 0, 1, &pWriteBuffer);

           pWriteBuffer[0] = coef[index];
           WC_rc = WC_PeRegWrite(TestInfo.DeviceNum, 1, 1, pWriteBuffer);

        pWriteBuffer[0] = 0x10;
        WC_rc = WC_PeRegWrite(TestInfo.DeviceNum, 0, 1, pWriteBuffer);

        }

        pWriteBuffer[0] = 0x01;
        WC_rc = WC_PeRegWrite(TestInfo.DeviceNum, 0, 1, pWriteBuffer);


        for (index=0; index < 64 ; index ++)
        {
        pWriteBuffer[0] = 0x0;
        WC_rc = WC_PeRegWrite(TestInfo.DeviceNum, 2, 1, pWriteBuffer);

        pWriteBuffer[0] = 0x1;
        WC_rc = WC_PeRegWrite(TestInfo.DeviceNum, 2, 1, pWriteBuffer);

        WC_rc = WC_PeRegRead(TestInfo.DeviceNum, 3, 1, pWriteBuffer);
        coef[index] = (short)((pWriteBuffer[0] > 256) ? (pWriteBuffer[0] - 512) : pWriteBuffer[0]);

        }
```

N/A.

10.2.9 Conformance Testing

10.2.9.1 Reference software type, version and input data set

Our functional testing was performed on the MPEG-4 main profile software xvid (1.0.3). The video

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

test sequences is “Foreman”.

10.2.9.2 API vector conformance

The test vectors used are QCIF test sequences.

10.2.9.3 End to end conformance (conformance of encoded bitstreams or decoded pictures)

xvid_decraw - raw mpeg4 bitstream decoder.

Command line: xvid_decraw.exe -i foreman_qcif_30.bit -d -c i420

Input MPEG-4 bitstream: foreman_qcif_30.bit

Output YUV file: output.yuv

10.2.10 Limitations

FPGA occupation area is the main limitation of the design.

10.3 VLD+IQ+IDCT for MPEG-4 Part 2

10.3.1 Abstract description of the module

This clause describes a module that implements three stages of the MPEG-4 decoding process. It decodes all variable length codes (VLC) associated to DCT coefficients included in a coded input stream, performs the inverse quantization (IQ) of the decoded coefficients and finally performs an inverse transform of those coefficients using a high performance inverse discrete cosine transform (IDCT).

The module is implemented on a Xilinx VIRTEX™-II FPGA. The VHDL code of this module has been uploaded to the MPEG CVS.

10.3.2 Module specification

10.3.2.1	**MPEG 4 part:**	**9**
10.3.2.2	**Profile :**	**All**
10.3.2.3	**Level addressed:**	**All**
10.3.2.4	**Module Name:**	**VLD+IQ+IDCT**
10.3.2.5	**Module latency:**	**5<n.cycles<108**
10.3.2.6	**Module data throughput:**	**<21 Mcodes/sec, >1 Mcodes/s**
10.3.2.7	**Max clock frequency:**	**65.4MHz**
10.3.2.8	**Resource usage:**	
10.3.2.8.1	**LUTs**	**26569**
10.3.2.8.2	**Multipliers**	**7**
10.3.2.8.3	**BRAMs**	**1**
10.3.2.9	**Revision:**	**1.0**
10.3.2.10	**Authors:**	**M. Santos, A. Silva, A. Navarro**
10.3.2.11	**Creation Date:**	**23/03/2006**
10.3.2.12	**Modification Date:**	**23/03/2006**

10.3.3 Introduction

This module is composed by three independent sub-modules that are connected to each other to perform the following stages of the MPEG-4 decoding process: Variable Length Decoding (VLD), Inverse Quantization (IQ) and Inverse Discrete Cosine Transform (IDCT). Those sub-modules are described in previously submitted documents.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The VLD sub-module decodes the coefficients VLC codes of a coded input stream returning the decoded coefficients in zig-zag scanning. The following sub-module, IQ, reads each coefficient available in the VLD output bus. As soon as the IQ sub-module quantizes one coefficient passes it to the IDCT sub-module. After the IDCT sub-module transforms one coefficient, it will signal the output of the module in order to validate the decoded coefficient.

10.3.4 Functional Description

The decoding process is started when the START input signal is set to high. Then the VLD module collects 30 bits from the input stream. Afterwards, the module will search in its internal look-up table (LUT) one VLC code that matches with the incoming stream regarding if it is an INTRA or INTER block (INTRA_INTER signal) and if the stream is in the short_video_header form (SHORTVIDEO signal). After the match is done, the module will analyse more bits of the previous loaded 30 bits. If the module does not find a complete VLC code in the remaining bits, it will request more 30 input bits from the bitstream parser for further decoding (MORE signal). Each time a code is decoded it is passed to the IQ sub-module. The IQ sub-module will quantize the incoming value according to the type of block (INTRA, INTER) if it is a CHROMA or LUMA block, quantize type (SHORTVIDEO) and quantize scale (QP). After inverse quantisation is done the IDCT sub-module will transform all the 64 coefficients and signalling the output READY when it's done.

The module also outputs the number of decoded bits (signal BITS) to the device that is feeding the module, in order to acknowledge how much bits the device should skip to stream another 30 bits to the module.

10.3.4.1 I/O Diagram

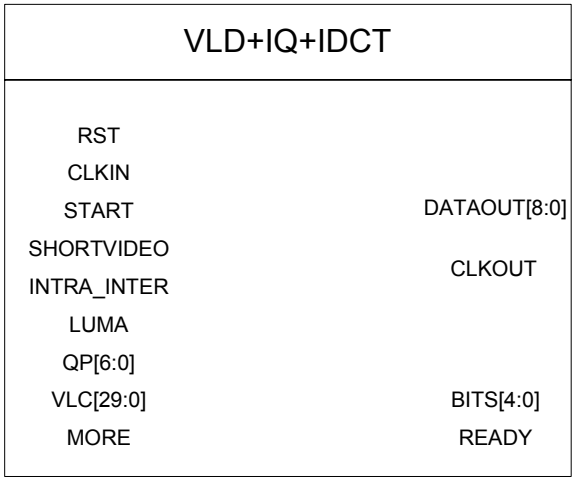


Figure 134 — I/O Diagram.

10.3.4.2 I/O Ports Description

- RST – at level high, it resets all the flip-flops and state machines on the module.
- CLKIN – system clock for encoded input stream load.
- START – is asserted high to start reading data. This signal is asserted when the input data (VLC) pins are valid and remains in level high only one CLKIN clock cycle.
- SHORTVIDEO – is asserted to high to signalling input stream with short_video_header=1. It should be stable till READY signal is high.
- INTRA_INTER – is asserted high to select INTER blocks and LOW to select INTRA blocks. It should be stable till READY signal is high.
- LUMA – is asserted high to select CHROMA blocks and LOW to select LUMA blocks. It should be stable till READY signal is high.
- QP[6:0] – Quantizer scale. It should be stable till READY signal is high.
- VLC[29:0] – is the input stream with variable length codes (VLCs). It should be stable till MORE signal is high.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

- MORE – when high the module is requesting more VLC bits. When low the module is in the decoding process or the decoding process is done.
- DATAOUT[8:0] – decoded coefficients (pixels). Valid only when READY is high.
- CLKOUT – system clock for data pixels in the bus DATAOUT.
- BITS[4:0] – number of bits to skip in the input stream for the next VLC decoding. Valid only when MORE is high.
- READY – valid data pixels at DATAOUT[8:0] when READY is high for the current block.

10.3.4.3 I/O Ports Description

Port Name	Port Width	Direction	Description
RST	1	Input	Reset the module
CLKIN	1	Input	Clock to load the input stream in VLC
START	1	Input	Start decoding if START is high
SHORTVIDEO	1	Input	Select short_video_header if high
INTRA_INTER	1	Input	Select intra blocks (low) or inter blocks (high)
LUMA	1	Input	Select luma blocks (low) or chroma blocks (high)
QP	7	Input	Quantization parameter
VLC	30	Input	Input stream
MORE	1	Output	More bits to feed
DATAOUT	9	Output	Block data out
CLKOUT	1	Input	Clock to read the data pixels of DATAOUT
BITS	5	Output	Number of bits to skip to next stream position
READY	1	Output	Valid data pixels at DATAOUT

10.3.5 Algorithm

The module VLD+IQ+IDCT uses individual modules, serially connected to each other.

10.3.6 Implementation

The following figure shows the block diagram of the VLD+IQ+IDCT module.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

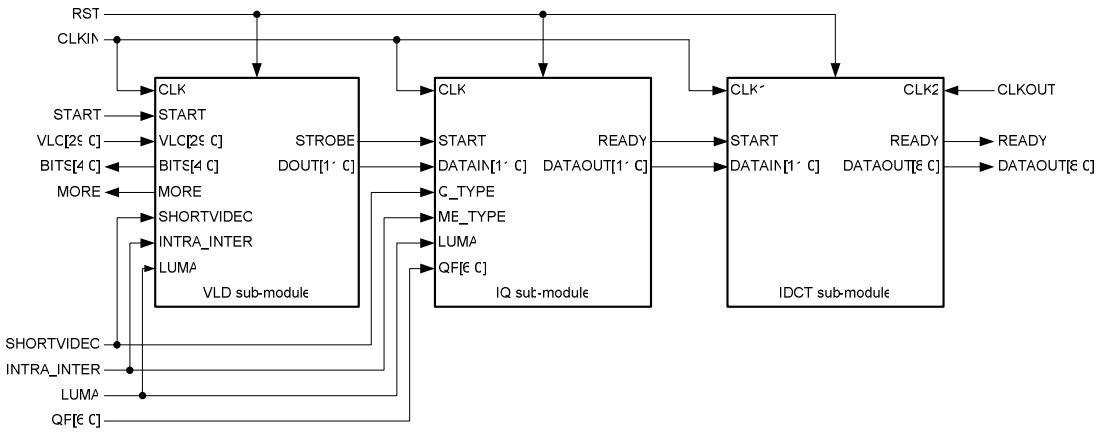


Figure 135 — VLD+IQ+IDCT Block Diagram.

10.3.6.1 Interfaces

The next figure shows the interfaces of the implemented module.

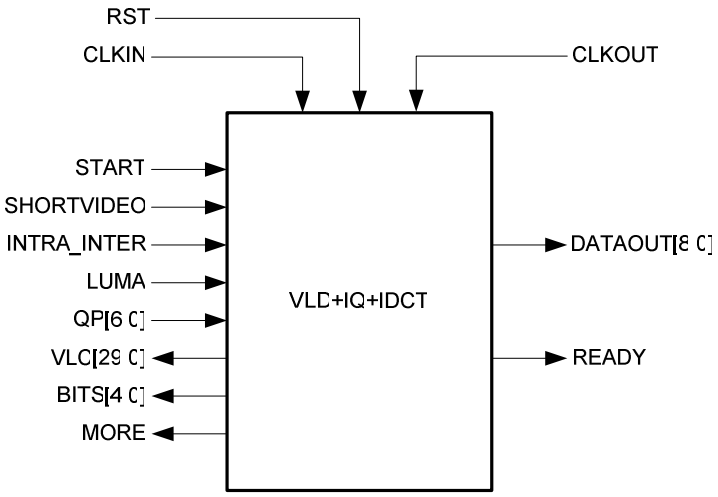


Figure 136 — VLD+IQ+IDCT module interfaces.

- RST – at level high reset all the flip-flops and state machines on the module.
- CLKIN – system clock for encoded input stream load.
- START – is asserted high to starts reading data. This signal is asserted when the input data (VLC) pins are valid and remains in level high only one CLKIN clock cycle.
- SHORTVIDEO – is asserted to high to signalling input stream with short_video_header=1. It should be stable till READY signal is high.
- INTRA_INTER – is asserted high to select INTER blocks and LOW to select INTRA blocks. It should be stable till READY signal is high.
- LUMA – is asserted high to select CHROMA blocks and LOW to select LUMA blocks. It should be stable till READY signal is high.
- QP[6:0] – Quantizer scale. It should be stable till READY signal is high.
- VLC[29:0] – is the input stream with variable length codes (VLCs). It should be stable till MORE signal is high.
- MORE – when high the module is requesting more VLC bits. When low the module is in the decoding process or the decoding process is done.
- DATAOUT[8:0] – decoded coefficients (pixels). Valid only when READY is high.

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

- CLKOUT – system clock for data pixels in DATAOUT bus.
- BITS[4:0] – number of bits to skip in the input stream for the next VLC decoding. Valid only when MORE is high.
- READY – valid data pixels at DATAOUT[8:0] when READY is high for the current block.

10.3.6.2 Timing Diagrams

The timing diagram of the implemented block is shown in the next figure.

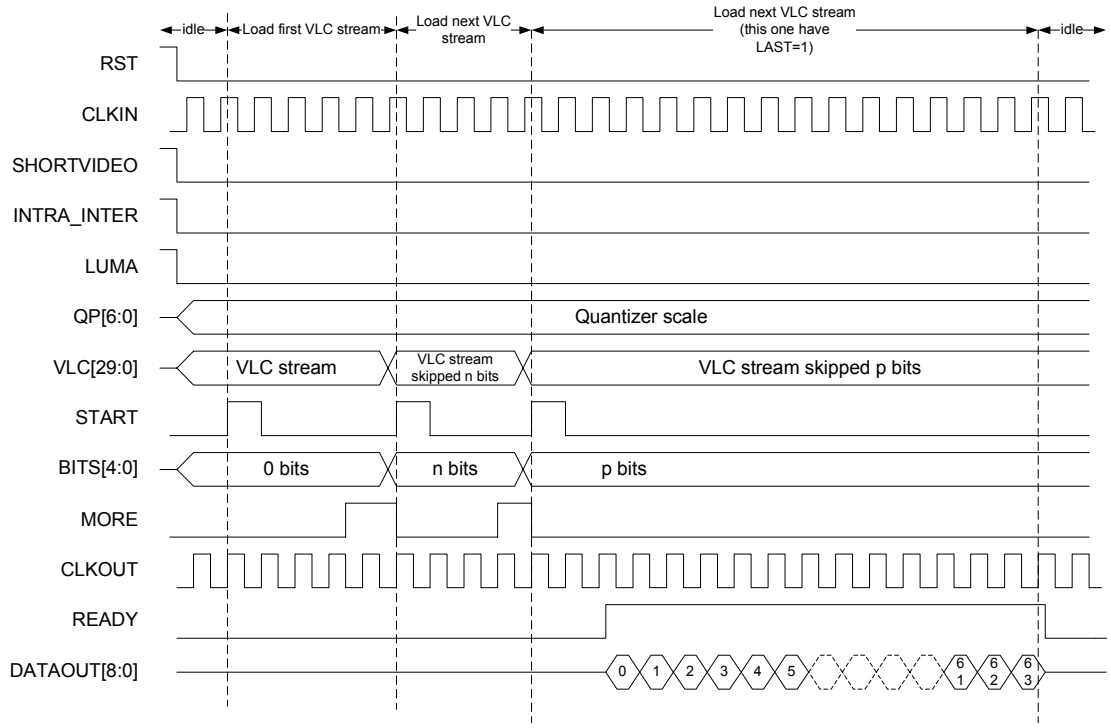


Figure 137 — Timing Diagram.

The diagram shows an example of three VLC streams loaded to the module. The first one corresponds to a stream where the DC coefficient will be decoded and the last VLC stream is such that the decoding process will reveal LAST = 1. After this the READY signal will go high and the values at DATAOUT are ready to be read.

10.3.7 Results of Performance & Resource Estimation

The design tool used in this work was ISE 7.1i. The obtained report from the synthesis tool, XST, is presented below:

Device utilization summary:

Selected Device : 2v3000fg676-4

Number of LUTs:	*26569*	*out of*	*28672*	*92%*
Number of BRAMs:	*1*	*out of*	*96*	*1%*
Number of Multipliers:	*7*	*out of*	*96*	*7%*

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Timing Summary:

Speed Grade: -4

Minimum period: 15.30ns (Maximum Frequency: 65.4MHz)

10.3.8 Limitations

The performance of this module depends greatly in the performance of the sub-modules used. For example, the VLD module could be optimized by searching in parallel the matches for codes with length of 2 thru 12. With this optimization the maximum number of cycles to perform the VLD would be bounded by the maximum number of entries in the LUT plus the clock cycles required initiating the process and finishing it. Also in the IQ sub-module if the computation are done in parallel the performance could rise up significantly.

10.4 A SYSTEM C MODEL FOR 2X2 HADAMARD TRANSFORM AND QUANTIZATION FOR MPEG–4 PART 10

10.4.1 Abstract description of the module

This subclause describes a SystemC model of the 2x2 Hadamard transform that is applied to the DC coefficients of the four 4x4 blocks of each chroma component as described in the MPEG –4 Part 10 Advanced Video Coding (AVC) standard. A VLSI prototype for the quantization process that is accompanied with the transform operation is provided as well. The implemented transform represents a level in the hierarchical transform adopted in the new AVC standard. The transform is computed using add operations only. This reduces the computational requirements of the design.

10.4.2 Module specification

10.4.2.1	**MPEG 4 part:**	***10***
10.4.2.2	**Profile :**	***All***
10.4.2.3	**Level addressed:**	***All***
10.4.2.4	**Module Name:**	***2x2 Hadamard (SystemC)***
10.4.2.5	**Module latency:**	***N/A***
10.4.2.6	**Module data throughput:**	***A 2x2 parallel quantized transform coefficients matrix/ CC***
10.4.2.7	**Max clock frequency:**	***N/A***
10.4.2.8	**Resource usage:**	
10.4.2.8.1	**CLB Slices:**	***N/A***
10.4.2.8.2	**DFFs or Latches:**	***N/A***
10.4.2.8.3	**Function Generators:**	***N/A***
10.4.2.8.4	**External Memory:**	***N/A***
10.4.2.8.5	**Number of Gates:**	***N/A***
10.4.2.9	**Revision:**	***1.00***
10.4.2.10	**Authors:**	***Ihab Amer, Wael Badawy, and Graham Jullien***
10.4.2.11	**Creation Date:**	***July 2004***
10.4.2.12	**Modification Date:**	***October 2004***

10.4.3 Introduction

Digital video streaming is increasingly gaining higher reputation due to the noticeable progress in the efficiency of various digital video-coding techniques. This raises the need for an industry standard for compressed video representation with substantially increased coding efficiency and enhanced robustness to network environments [13].

In 2001, the Joint Video Team (JVT) was formed to represent the cooperation between the ITU-T Video Coding Expert Group (VCEG) and the ISO/IEC Moving Picture Expert Group (MPEG) aiming for the development of a new Recommendation/International Standard.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The ITU-T video coding standards are called recommendations, and they are denoted with H.26x. The ISO/IEC standards are denoted with MPEG –x [14]. Hence, the name MPEG –4 Part 10 “AVC” (or H.264) is given to the new standard for coding of natural video images that is currently being finalized by the JVT [15].

The main objective behind the AVC project is to develop a “block to basics” approach where simple and straightforward design using well-known building blocks is used [14].

AVC shares common features with other existing standards, while at the same time, it has a number of new features that distinguish it from conventional standards. For instance, AVC offers good video quality at high and low bit rates. It is also characterized by error resilience and network friendliness [16]-[19].

The new standard does not use the traditional 8x8 Discrete Cosine Transform (DCT) as the basic transform. Instead, a novel hierarchy of transforms is introduced. The used transforms can be computed exactly in integer arithmetic, thus avoiding inverse transform mismatch problem [21].

Moreover, the used transforms can be computed without multiplications, just additions and shifts, in 16-bit arithmetic. This minimizes the computational complexity significantly. Besides, the quantization operation uses multiplications avoiding unsynthesizable divisions.

In the present contribution, a hardware prototype for the 2x2 Hadamard transform and quantization that is applied to the DC coefficients of the four 4x4 blocks of each chroma component in the AVC standard is introduced. The transform is computed using add operations only, which reduces the computational requirements of the design.

10.4.4 Functional Description

10.4.4.1 Functional description details

In this subclause, the hardware prototype of the 2x2 Hadamard transform and quantization adopted by the AVC standard is introduced. It is used for the coding of the DC coefficients of the four 4x4 blocks of each chroma component.

10.4.4.2 I/O Diagram

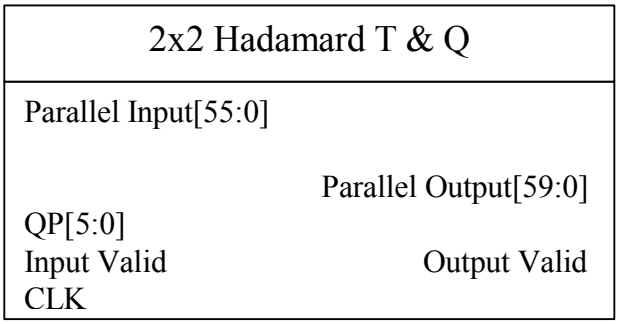


Figure 138

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.4.4.3 I/O Ports Description

Table 30

Port Name	Port Width	Direction	Description
Parallel Input[55:0]	56	Input	2x2 matrix of DC coefficients
QP[5:0]	6	Input	Quantization Parameter
Input Valid	1	Input	Flag indicating that input is valid
CLK	1	Input	System clock
Parallel Output[59:0]	60	Output	2x2 parallel quantized transform coefficients matrix
Output Valid	1	Output	Flag indicating that output is valid

10.4.5 Algorithm

A hierarchical transform is adopted in the MPEG-4.Part10 / AVC standard. A block diagram showing the hierarchy of transform before the quantization process in the encoder-side is given in Figure 139.

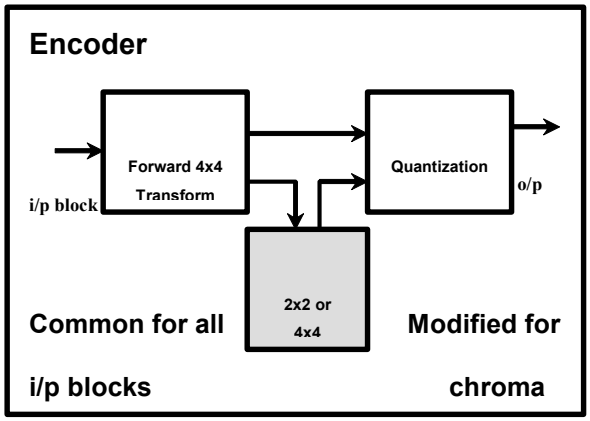


Figure 139 — Hierarchical transform and quantization in AVC standard.

A forward transform is first applied to the input 4x4 block. This transform represents an integer orthogonal approximation to the DCT. It allows for bit-exact implementation for all encoders and decoders [21].

Intra-16 prediction modes and chroma intra modes are intended for coding of smooth areas. Therefore, in order to decrease the reconstruction error, the DC coefficients undergo a second transform with the result that we have transform coefficients covering the whole macroblock [16].

An additional 2x2 transform is also applied to the DC coefficients of the four 4x4 blocks of each chroma component. The gray box in Figure 139 represents this additional transform. The cascading of block transforms is equivalent to an extension to the length of the transform functions [21]. This results in an increase in the reconstruction accuracy.

In conventional standards, the second level transform is the same as the first level transform. The current draft specifies just a Hadamard transform to the second level. No performance loss is observed over the standard video test sets [22][24].

The Hadamard transform formula that is applied to a 2x2 array (W) of DC coefficients of one of the chroma components is shown in Equation 15.

$$Y = HWH^T$$

Equation 15

where the matrix H is given by Equation 16.

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

$$H = H^T = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Equation 16

The quantization process for chroma or intra-16 luma differs from the corresponding process in other modes of operation. The formulas for post-scaling and quantization of transformed chroma DC coefficients are shown in Equation 17 and Equation 18.

$$qbits = 15 + (QP \ DIV \ 6)$$

Equation 17

$$Z_{ij} = round(Y_{ij} . \frac{MF}{2^{qbits}})$$

Equation 18

where QP is a quantization parameter that enables the encoder to accurately and flexibly control the tradeoff between bit rate and quality. It can take any value from 0 up to 51. Z_{ij} is an element in the output quantized DC coefficients matrix. MF is a multiplication factor that depends on QP as shown in Table 31.

Table 31 — Multiplication Factor (MF).

QP	MF
0	26214
1	23831
2	20165
3	18725
4	16384
5	14564

The factor MF remains unchanged for $QP > 5$. It can be calculated using Equation 19.

$$MF_{QP>5} = MF_{QP=QP \bmod 6}$$

Equation 19

Equation (4) can be represented in pure integer arithmetic as shown in Equation 20 and Equation 21.

$$\left|Z_{ij}\right| = SHR(\left|Y_{ij}\right| .MF + f \, , \, qbits)$$

Equation 20

$$Sign(Z_{ij}) = Sign(Y_{ij})$$

Equation 21

where $SHR()$ is a procedure that right-shifts the result of its first argument a number of bits equal to its second argument. f is defined in the reference model software as $2^{qbits} / 3$ for intra blocks and $2^{qbits} / 6$ for inter blocks [15].

--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2009 – All rights reserved
Not for Resale

10.4.6 Implementation

The illustrated architecture can be integrated on the same chip with another architecture that performs the initial 4x4 forward transform [25].

This architecture is designed to perform pipelined operations. Therefore, with the exception of the first 2x2 input block, the architecture can output a whole coded block with each clock pulse. The design does not contain memory elements. Instead, a redundancy in computational elements shows up. This introduces an example of performance-area tradeoff.

Figure 140 shows the flow of signals between the two main stages of the design, the transformer and the quantizer.

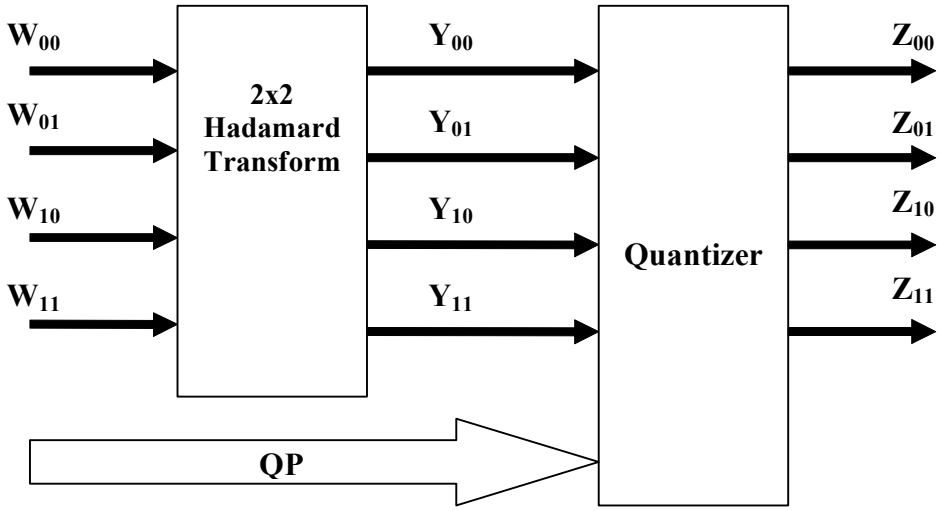


Figure 140 — The two main stages of the architecture.

A flow graph of the used 2x2 Hadamard transform is shown in Figure 141.

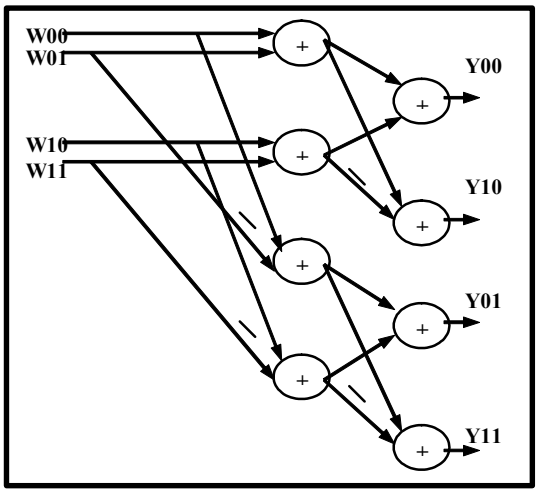


Figure 141 — Flow Graph for 2x2 Hadamard transform.

The quantization block consists of three different blocks, each having its specific task. A detailed diagram of the quantizer showing its three different blocks is shown in Figure 142.

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

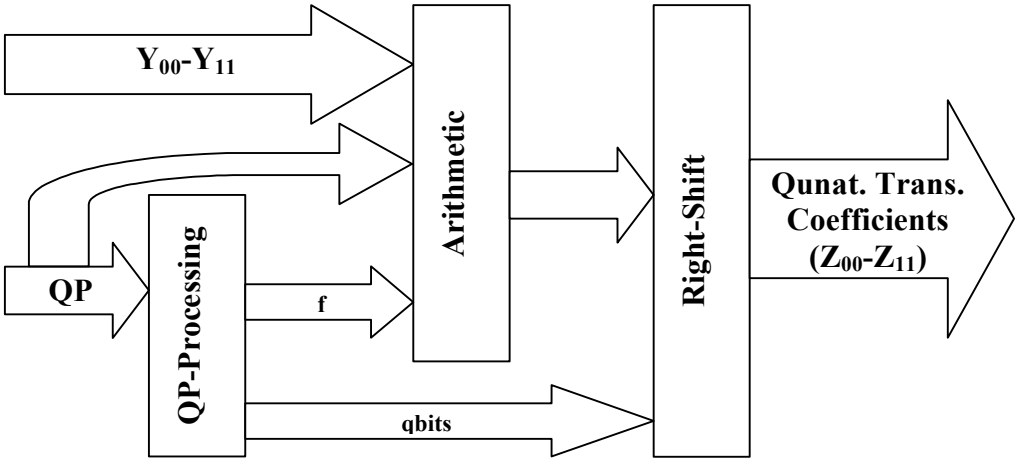


Figure 142 — A detailed diagram of the quantizer.

The QP-Processing block is responsible for using the input QP to calculate the values of $qbits$ and f. The Arithmetic block contains sub-blocks for performing multiplication and addition operations. Finally, the Right-Shift block shifts the output from the Arithmetic block a number of bits equal to qbits.

10.4.6.1 Interfaces

N/A.

10.4.6.2 Register File Access

Please refer to subclause 10.4.8.

10.4.6.3 Timing Diagrams

N/A.

10.4.7 Results of Performance & Resource Estimation

Behavioural simulation shows that the designed architecture functionally complies with the reference software. The architecture was embedded in JM 8.5 and its output stream was identical to the output from the original software. Figure 143 gives a comparison between the outputs before and after the embedding the SystemC block.

10.4.8 API calls from reference software

The switching between software and hardware is controlled by flags that can be reset to avoid switching, hence the software flow will execute normally bypassing the SystemC block. An example of a HW-block call from SW is as follows:

```
/*~~~~~~~~~~~~~~~~~Hardware/Software Switching~~~~~~~~~~~~~~~~~~*/
 if(H2_HW_ACCELERATOR){
      sc_hadamard_2(img->m7, m1, firstHW_Call);
      firstHW_Call = 0;
 }
 else
      sw_hadamard_2(img->m7, m1);
/*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
```


© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale



(a) **(b)**

Figure 143 — (a) Output before embedding the SystemC block(b) Output after embedding the SystemC block.

10.4.9 Conformance Testing

10.4.9.1 Reference software type, version and input data set

Our functional testing was performed on the AVC (MPEG-4 Part 10) reference software (JM8.5). The video test sequences are “Miss America” and “Foreman”.

10.4.9.2 API vector conformance

The test vectors used are QCIF test sequences.

10.4.9.3 End to end conformance (conformance of encoded bitstreams or decoded pictures)

The end-to-end encoder conformance test is evaluated via a mixed C and SystemC environment using the JM 8.5 software reference model. Figure 144 and Figure 145 show that the results obtained before and after using the hardware accelerators are identical.

© ISO/IEC 2009 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
Freq. for encoded bitstream       : 30
Hadamard transform                : Used
Image format                      : 176x144
Error robustness                  : Off
Search range                      : 16
No of ref. frames used in P pred  : 10
Total encoding time for the seq.  : 3.876 sec
Total ME time for sequence        : 1.041 sec
Sequence type                     : IPPP (QP: I 28, P 28)
Entropy coding method             : CAVLC
Profile/Level IDC                 : (66,30)
Search range restrictions         : none
RD-optimized mode decision        : used
Data Partitioning Mode            : 1 partition
Output File Format                : H.264 Bit Stream File Format
------------------ Average data all frames  -----------------------------------
SNR Y(dB)                         : 40.59
SNR U(dB)                         : 39.24
SNR V(dB)                         : 39.77
Total bits                        : 12408 (I 10896, P  1344, NVB 168)
Bit rate (kbit/s)  @ 30.00 Hz     : 124.08
Bits to avoid Startcode Emulation : 0
Bits for parameter sets           : 168
```

Figure 144 — Summary of results reported by JM 8.5 before embedding the SystemC block.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2009 – All rights reserved

```
Freq. for encoded bitstream      : 30
Hadamard transform               : Used
Image format                     : 176x144
Error robustness                 : Off
Search range                     : 16
No of ref. frames used in P pred  : 10
Total encoding time for the seq.  : 4.526 sec
Total ME time for sequence       : 1.060 sec
Sequence type                    : IPPP (QP: I 28, P 28)
Entropy coding method            : CAVLC
Profile/Level IDC                : (66,30)
Search range restrictions        : none
RD-optimized mode decision       : used
Data Partitioning Mode           : 1 partition
Output File Format               : H.264 Bit Stream File Format
------------------ Average data all frames  -----------------------------------
SNR Y(dB)                        : 40.59
SNR U(dB)                        : 39.24
SNR V(dB)                        : 39.77
Total bits                       : 12408 (I 10896, P  1344, NVB 168)
```

Figure 145 — Summary of results reported by JM 8.5 after embedding the SystemC block.

10.4.10 Limitations

Increased area is the main limitation of the design.

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.5 A VHDL HARDWARE BLOCK FOR 2X2 HADAMARD TRANSFORM AND QUANTIZATION WITH APPLICATION TO MPEG–4 PART 10 AVC

10.5.1 Abstract description of the module

This subclause describes a hardware prototype for the 2x2 Hadamard transform and quantization that is applied to the DC coefficients of the four 4x4 blocks of each chroma component in the MPEG-4.Part10 / AVC standard. The transform is computed using add operations only, which reduces the computational requirements of the design.

10.5.2 Module specification

10.5.2.1 MPEG 4 part: ***10***
10.5.2.2 Profile : ***All***
10.5.2.3 Level addressed: ***All***
10.5.2.4 Module Name: ***2x2 Hadamard (VHDL)***
10.5.2.5 Module latency: ***355.2 ns***
10.5.2.6 Module data throughput: ***A 2x2 parallel quantized transform coefficients matrix/sec***
10.5.2.7 Max clock frequency: ***42.4 MHz***
10.5.2.8 Resource usage:
10.5.2.8.1 CLB Slices: ***1016***
10.5.2.8.2 DFFs or Latches: ***1095***
10.5.2.8.3 Function Generators: ***2032***
10.5.2.8.4 External Memory: ***none***
10.5.2.8.5 Number of Gates: ***1981***
10.5.2.9 Revision: ***1.00***
10.5.2.10 Authors: ***Ihab Amer, Wael Badawy, and Graham Jullien***
10.5.2.11 Creation Date: ***July 2004***
10.5.2.12 Modification Date: ***October 2004***

10.5.3 Introduction

The new MPEG-4 Part 10 AVC standard does not use the traditional 8x8 Discrete Cosine Transform (DCT) as the basic transform. Instead, a novel hierarchy of transforms is introduced. The used transforms can be computed exactly in integer arithmetic, thus avoiding inverse transform mismatch problem [21].

Moreover, the used transforms can be computed without multiplications, just additions and shifts, in 16-bit arithmetic. This minimizes the computational complexity significantly. Besides, the quantization operation uses multiplications avoiding unsynthesizable divisions.

A VHDL hardware prototype for the 2x2 Hadamard transform and quantization that is applied to the DC coefficients of the four 4x4 blocks of each chroma component in the AVC standard is described. The transform is computed using add operations only, which reduces the computational requirements of the design.

10.5.4 Functional Description

10.5.4.1 Functional description details

In this subclause, a VHDL hardware prototype of the 2x2 Hadamard transform and quantization adopted by the AVC standard is described. It is used for the coding of the DC coefficients of the four 4x4 blocks of each chroma component.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.5.4.2 I/O Diagram

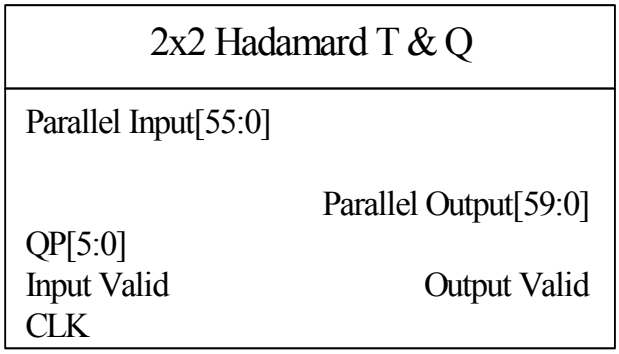


Figure 146

10.5.4.3 I/O Ports Description

Table 32

Port Name	Port Width	Direction	Description
Parallel Input[55:0]	56	Input	2x2 matrix of DC coefficients
QP[5:0]	6	Input	Quantization Parameter
Input Valid	1	Input	Flag indicating that input is valid
CLK	1	Input	System clock
Parallel Output[59:0]	60	Output	2x2 parallel quantized transform coefficients matrix
Output Valid	1	Output	Flag indicating that output is valid

10.5.5 Algorithm

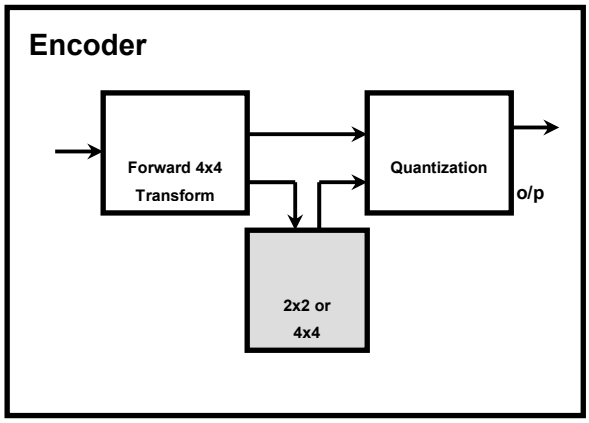


Figure 147 — Hierarchical transform and quantization in AVC standard.

A hierarchical transform is adopted in the MPEG-4.Part10 / AVC standard. A block diagram showing the hierarchy of transform before the quantization process in the encoder-side is given in Figure 147.

A forward transform is first applied to the input 4x4 block. This transform represents an integer orthogonal approximation to the DCT. It allows for bit-exact implementation for all encoders and decoders [21].

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Intra-16 prediction modes and chroma intra modes are intended for coding of smooth areas. Therefore, in order to decrease the reconstruction error, the DC coefficients undergo a second transform with the result that we have transform coefficients covering the whole macroblock [16].

An additional 2x2 transform is also applied to the DC coefficients of the four 4x4 blocks of each chroma component. The gray box in Figure 147 represents this additional transform. The cascading of block transforms is equivalent to an extension to the length of the transform functions [21]. This results in an increase in the reconstruction accuracy.

In conventional standards, the second level transform is the same as the first level transform. The current draft specifies just a Hadamard transform to the second level. No performance loss is observed over the standard video test sets [22][24].

The Hadamard transform formula that is applied to a 2x2 array (W) of DC coefficients of one of the chroma components is shown in Equation 22.

$$Y = HWH^T$$

Equation 22

where the matrix H is given by Equation 23.

$$H = H^T = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Equation 23

The quantization process for chroma or intra-16 luma differs from the corresponding process in other modes of operation. The formulas for post-scaling and quantization of transformed chroma DC coefficients are shown in Equation 24 and Equation 25.

$$qbits = 15 + (QP \ DIV \ 6)$$

Equation 24

$$Z_{ij} = round(Y_{ij} \cdot \frac{MF}{2^{qbits}})$$

Equation 25

where QP is a quantization parameter that enables the encoder to accurately and flexibly control the tradeoff between bit rate and quality. It can take any value from 0 up to 51. Z_{ij} is an element in the output quantized DC coefficients matrix. MF is a multiplication factor that depends on QP as shown in Table 33.

Table 33 — Multiplication Factor (MF).

QP	MF
0	26214
1	23831
2	20165
3	18725
4	16384
5	14564

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The factor MF remains unchanged for $QP > 5$. It can be calculated using Equation 26.

$$MF_{QP>5} = MF_{QP=QP \bmod 6}$$

Equation 26

Equation 25 can be represented in pure integer arithmetic as shown in Equation 27 and Equation 28.

$$\left|Z_{ij}\right| = SHR(\left|Y_{ij}\right|.MF + f,\ qbits)$$

Equation 27

$$Sign(Z_{ij}) = Sign(Y_{ij})$$

Equation 28

where $SHR()$ is a procedure that right-shifts the result of its first argument a number of bits equal to its second argument. f is defined in the reference model software as $2^{qbits}/3$ for intra blocks and $2^{qbits}/6$ for inter blocks [15].

10.5.6 Implementation

The illustrated architecture can be integrated on the same chip with another architecture that performs the initial 4x4 forward transform [25].

This architecture is designed to perform pipelined operations. Therefore, with the exception of the first 2x2 input block, the architecture can output a whole coded block with each clock pulse. The design does not contain memory elements. Instead, a redundancy in computational elements shows up. This introduces an example of performance-area tradeoff.

Figure 148 shows the flow of signals between the two main stages of the design, the transformer and the quantizer.

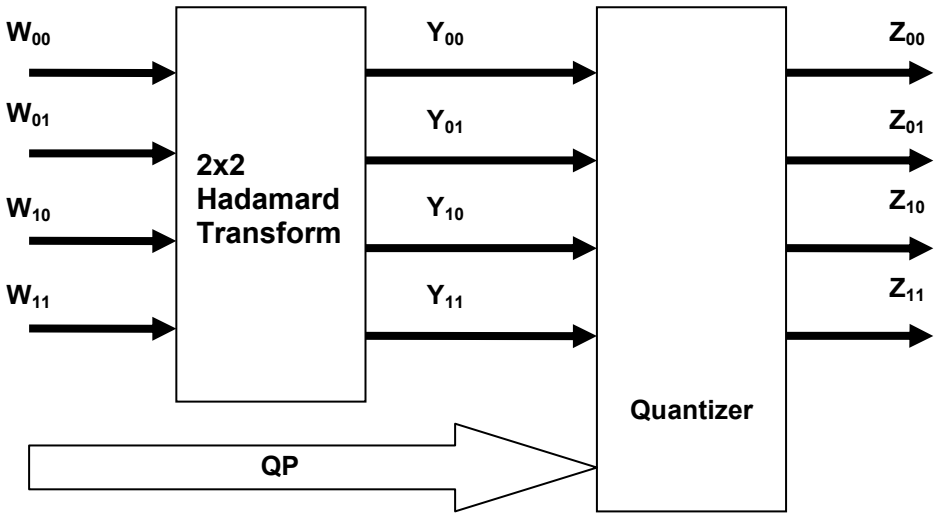


Figure 148 — The two main stages of the developed architecture.

Figure 149 — Flow Graph for 2x2 Hadamard transform.

A flow graph of the used 2x2 Hadamard transform is shown in Figure 149. The quantization block consists of three different blocks, each having its specific task. A detailed diagram of the quantizer showing its three different blocks is shown in Figure 150.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

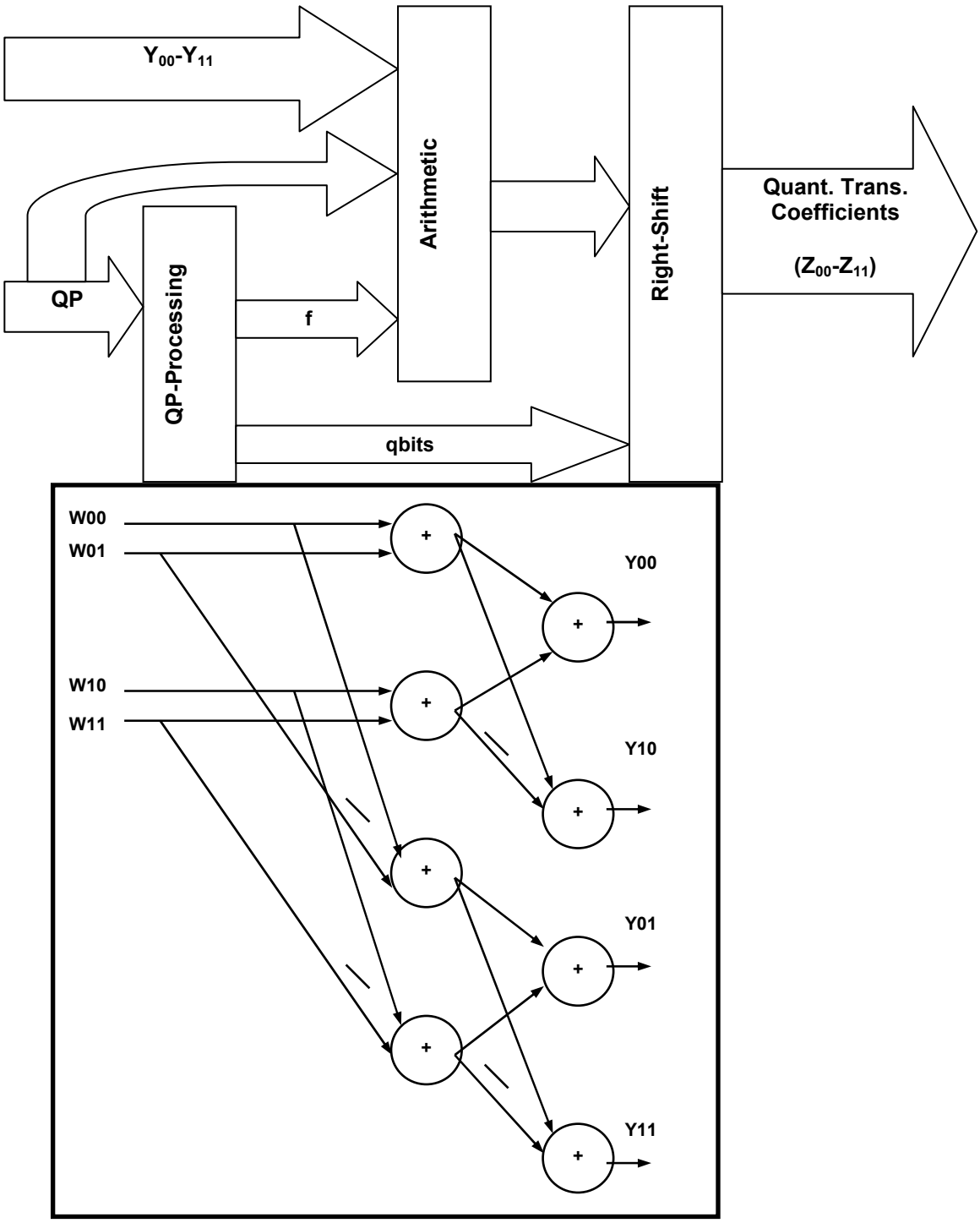


Figure 150 — A detailed diagram of the quantizer.

The QP-Processing block is responsible for using the input QP to calculate the values of $qbits$ and f. The Arithmetic block contains sub-blocks for performing multiplication and addition operations. Finally, the Right-Shift block shifts the output from the Arithmetic block a number of bits equal to qbits.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.5.6.1 Interfaces

10.5.6.2 Register File Access

Refer to subclause 10.5.4.

10.5.6.3 Timing Diagrams

N/A.

10.5.7 Results of Performance & Resource Estimation

The architecture mentioned in subclause 10.5.4 is represented using VHDL language. It is simulated using the Mentor Graphics© ModelSim 5.4® simulation tool, and synthesized using Leonardo Spectrum®.

The target technology is the FPGA device (2V3000fg676) from the Virtex-II family of Xilinx© due to its availability and its large number of I/O pins.

The critical path is estimated by the synthesis tool to be 23.68 ns. This is equivalent to a maximum operating frequency of 42.4 MHz. The chip outputs a whole 2x2 coded block with each clock pulse (except for the first block). Thus, the design can be easily integrated with the forward 4x4 transform architecture without damaging its performance. Hence, the resulting architecture satisfies the real-time constraints required by different digital video applications such as HDTV.

Table 34

Critical Path (ns)	CLK Freq. (MHz)	# of Gates	# of I/O Ports
23.68	42.4	1981	123
# of Nets	**# of DFF's or Latches**	**# Function Generators**	**# of CLB Slices**
310	1095	2032	1016

The results obtained leads to the suggestion of taking the input serially to reduce the consumed area, integrating other different operations on the same chip, or targeting other applications that use more complicated-higher resolution video formats.

10.5.8 API calls from reference software

N/A.

10.5.9 Conformance Testing

The architecture has not been tested on the platform but its simulation results conforms with the software results.

10.5.9.1 Reference software type, version and input data set

N/A.

10.5.9.2 API vector conformance

N/A.

10.5.9.3 End to end conformance (conformance of encoded bitstreams or decoded pictures)

N/A.

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.5.10 Limitations

Increased area is the main limitation of the design.

10.6 A SYSTEMC MODEL FOR 4X4 HADAMARD TRANSFORM AND QUANTIZATION FOR MPEG-4 PART 10

10.6.1 Abstract description of the module

This subclause presents a SystemC hardware model for the 4×4 Hadamard transform and quantization that is applied to the DC coefficients of the luma component when the macroblock is encoded in 16×16 intra prediction mode. The implemented transform represents the second level in the transformation hierarchy, which is adopted by the MPEG-4 Part 10 standard. It comes after the forward 4×4 integer approximation of the DCT transform.

10.6.2 Module specification

10.6.2.1 MPEG 4 part: ***10***
10.6.2.2 Profile : ***All***
10.6.2.3 Level addressed: ***All***
10.6.2.4 Module Name: ***4x4 Hadamard (SystemC)***
10.6.2.5 Module latency: ***N/A***
10.6.2.6 Module data throughput: ***A 4x4 parallel quantized transform coefficients matrix/CC***
10.6.2.7 Max clock frequency: ***N/A***
10.6.2.8 Resource usage:
10.6.2.8.1 CLB Slices: ***N/A***
10.6.2.8.2 DFFs or Latches: ***N/A***
10.6.2.8.3 Function Generators: ***N/A***
10.6.2.8.4 External Memory: ***N/A***
10.6.2.8.5 Number of Gates: ***N/A***
10.6.2.9 Revision: ***1.00***
10.6.2.10 Authors: ***Ihab Amer, Wael Badawy, and Graham Jullien***
10.6.2.11 Creation Date: ***July 2004***
10.6.2.12 Modification Date: ***October 2004***

10.6.3 Introduction

In the present contribution, a hardware prototype for the 4×4 Hadamard transform that is applied to the DC coefficients of the luma component when the macroblock is encoded in 16×16 intra prediction mode is introduced. The proposed architecture is developed to use only add operations to reduce the computational requirements for the transform.

10.6.4 Functional Description

10.6.4.1 Functional description details

This subclause introduces the hardware prototype of the 4×4 Hadamard transform and quantization adopted by the MPEG-4 Part 10 AVC standard. It is applied to the DC coefficients of the sixteen 4×4 blocks of the luma component. The architecture uses 4×4 parallel input block.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.6.4.2 I/O Diagram

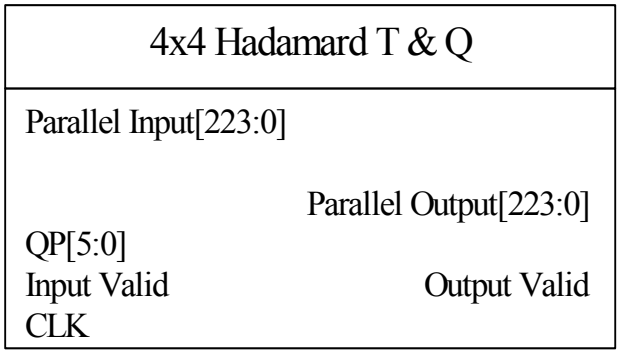


Figure 151

10.6.4.3 I/O Ports Description

Table 35

Port Name	Port Width	Direction	Description
Parallel Input[223:0]	224	Input	4x4 matrix of DC coefficients
QP[5:0]	6	Input	Quantization Parameter
Input Valid	1	Input	Flag indicating that input is valid
CLK	1	Input	System clock
Parallel Output[223:0]	224	Output	4x4 parallel quantized transform coefficients matrix
Output Valid	1	Output	Flag indicating that output is valid

10.6.5 Algorithm

A hierarchical transform is adopted in the MPEG-4 Part 10 standard. Figure 152 gives a block diagram showing the hierarchy of transform before the quantization process in the encoder-side.

Step 1 is an integer orthogonal approximation to the Discrete Cosine Transform (DCT) with a 4×4 input block, which allows for bit-exact implementation for all encoders and decoders [21].

Step 2 is a 4×4 Hadamard transform to the DC coefficients (from Step 1). It reduces the reconstruction error for intra-16 prediction mode. The cascading of block transforms is equivalent to an extension to the length of the transform functions [21].

© ISO/IEC 2009 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

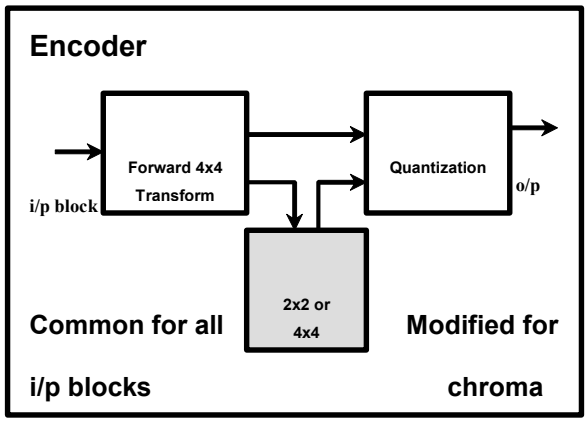


Figure 152 — Hierarchical transform and quantization in AVC standard.

The Hadamard transform formula that is applied to a 4×4 array (*W*) of DC coefficients of the luma component is shown in Equation 29. The output coefficients are divided by 2 (with rounding).

$$Y = (HWH^T)/2$$

Equation 29

The Matrix *H* is given by Equation 30.

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix}$$

Equation 30

The formulas for post-scaling and quantization of transformed intra-16 mode luma DC coefficients expressed in integer arithmetic are shown in Equation 31, Equation 32, and Equation 33.

$$qbits = 15 + (QP \ DIV \ 6)$$

Equation 31

$$|Z_{ij}| = SHR(|Y_{ij}|.MF + 2f, qbits + 1)$$

Equation 32

$$Sign(Z_{ij}) = Sign(Y_{ij})$$

Equation 33

QP is a quantization parameter that enables the encoder to control the trade-off between bit rate and quality. It can take any integer value from 0 up to 51. Z_{ij} is an element in the output quantized DC coefficients matrix. MF is a multiplication factor in order to avoid any division operation. It depends on QP as shown in Table 36. *SHR()* is a procedure that right-shifts the result of its first argument a number of bits equal to its second argument. *f* is defined in the reference model software as $2^{qbits}/3$ for intra blocks and $2^{qbits}/6$ for Inter blocks [15].

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 36 — Multiplication Factor (*MF*).

QP	MF
0	13107
1	11916
2	10082
3	9362
4	8192
5	7282

The factor MF does not change for $QP > 5$. It can be calculated using Equation 34.

$$MF_{QP>5} = MF_{QP=QP \bmod 6}$$

Equation 34

10.6.6 Implementation

The architecture can be integrated on the same chip with the forward 4×4 transform that is adopted by the AVC standard and/or with the 2×2 Hadamard transform that is applied to the DC coefficients of the four 4×4 blocks of each chroma component [25][26].

The architecture uses pipelined stages which increase the throughput. At steady state, the proposed architecture outputs an encoded block at each clock pulse. The architecture does not contain memory elements. Instead, a redundancy in computational elements is used.

Figure 153 shows the flow of signals between the two main stages of the design, the transformer and the quantizer.

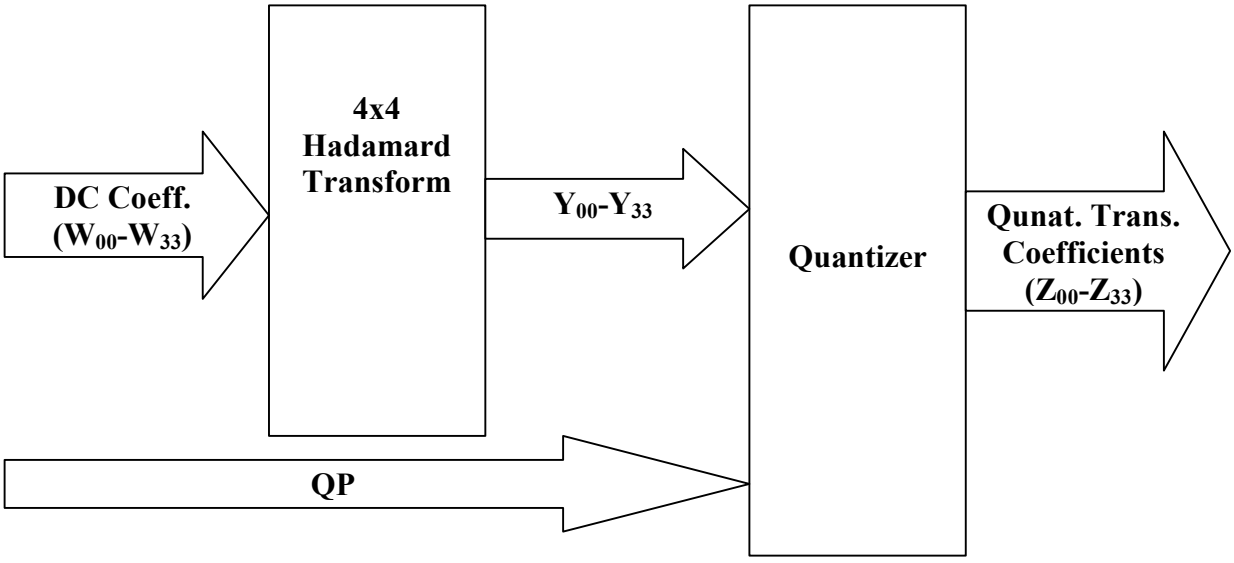


Figure 153 — Flow of signals between the two main stages of the design.

A flow graph of the used 4×4 Hadamard transform is shown in Figure 154. The transformation is performed in two stages. Each of them is responsible for multiplying two 4×4 matrices. Each is composed of four identical butterfly-adders. Its function is to perform a group of additions. In Figure 154 (a), the first butterfly-adder block

© ISO/IEC 2009 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

in the first sub-block is shown, while in Figure 154 (b), the first butterfly-adder block in the next sub-block is shown. The Transform block is the hardware implementation that corresponds to Equation 29.

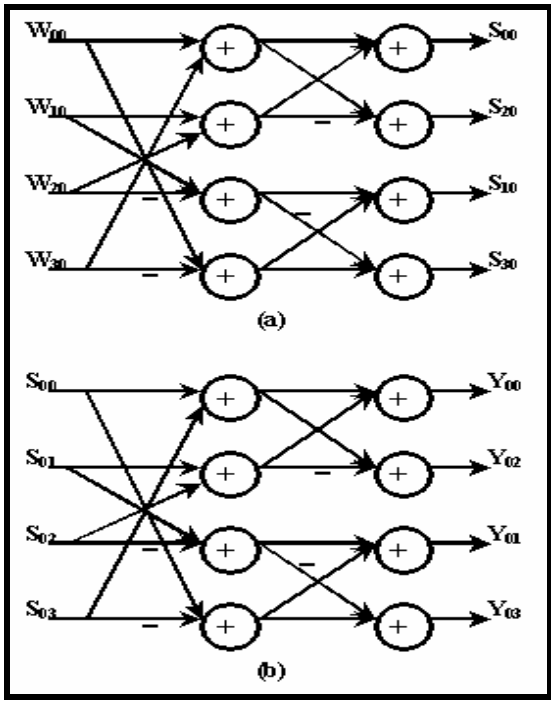


Figure 154 — First butterfly-adder block in: (a) First sub-block, (b) Second sub-block.

Figure 155 gives a detailed description of the quantizer. Quantization is performed in three different stages, each having its specific task. In the *QP-Processing* stage, QP is used to calculate the values of *qbits*, *f_by_2* as well as *MF*, which is a multiplication factor that is based on *QP* as shown in Table 36. The *Arithmetic* block is responsible for performing multiplication and addition operations. Finally, the *Right-Shift* block shifts the output from the Arithmetic block a number of bits equal to *qbits+1.*

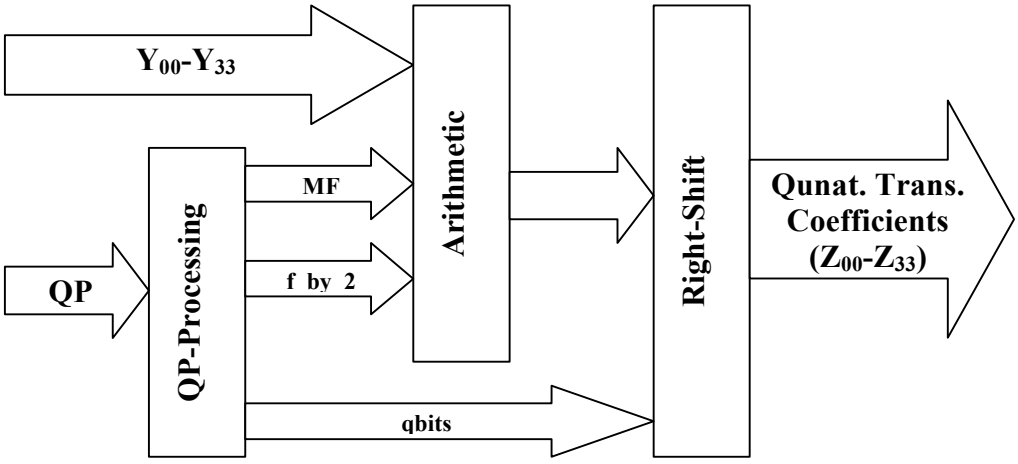


Figure 155 — A detailed diagram of the quantizer.

10.6.6.1 Interfaces

10.6.6.2 Register File Access

Please refer to subclause 10.6.4.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.6.6.3 Timing Diagrams

N/A.

10.6.7 Results of Performance & Resource Estimation

Behavioural simulation shows that the designed architecture functionally complies with the reference software. The architecture was embedded in JM 8.5 and its output stream was identical to the output from the original software. Figure 156 gives a comparison between the outputs before and after the embedding the SystemC block.



(a) **(b)**

Figure 156 — (a) Output before embedding the SystemC block (b) Output after embedding the SystemC block.

10.6.8 API calls from reference software

The switching between software and hardware is controlled by flags that can be reset to avoid switching, hence the software flow will execute normally bypassing the SystemC block. An example of a HW-block call from SW is as follows:

```
/*================Hardware/Software Switching================*/

    if(H4_HW_ACCELERATOR){

        sc_hadamard_4(M4, firstHW_Call);

        firstHW_Call = 0;

    }

    else

        sw_hadamard_4(M4);

/*===========================================================*/
```

10.6.9 Conformance Testing

10.6.9.1 Reference software type, version and input data set

Our functional testing was performed on ITU-T Rec. H.264 | ISO/IEC 14496-10 reference software (JM8.5). The video test sequences are “Miss America” and “Foreman”.

© ISO/IEC 2009 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.6.9.2 API vector conformance

The test vectors used are QCIF test sequences.

10.6.9.3 End to end conformance (conformance of encoded bitstreams or decoded pictures)

The end-to-end encoder conformance test is evaluated via a mixed C and SystemC environment using the JM 8.5 software reference model. Figure 157 and Figure 158 show that the results obtained before and after using the hardware accelerators are identical.

```
Freq. for encoded bitstream      : 30
Hadamard transform               : Used
Image format                     : 176x144
Error robustness                 : Off
Search range                     : 16
No of ref. frames used in P pred : 10
Total encoding time for the seq. : 3.876 sec
Total ME time for sequence       : 1.041 sec
Sequence type                    : IPPP (QP: I 28, P 28)
Entropy coding method            : CAVLC
Profile/Level IDC                : (66,30)
Search range restrictions        : none
RD-optimized mode decision       : used
Data Partitioning Mode           : 1 partition
Output File Format               : H.264 Bit Stream File Format
------------------ Average data all frames  ----------------------------------
SNR Y(dB)                        : 40.59
SNR U(dB)                        : 39.24
SNR V(dB)                        : 39.77
```

Figure 157 — Summary of results reported by JM 8.5 before embedding the SystemC block.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
Freq. for encoded bitstream      : 30
Hadamard transform               : Used
Image format                     : 176x144
Error robustness                 : Off
Search range                     : 16
No of ref. frames used in P pred  : 10
Total encoding time for the seq.  : 4.136 sec
Total ME time for sequence       : 1.181 sec
Sequence type                    : IPPP (QP: I 28, P 28)
Entropy coding method            : CAVLC
Profile/Level IDC                : (66,30)
Search range restrictions        : none
RD-optimized mode decision       : used
Data Partitioning Mode           : 1 partition
Output File Format               : H.264 Bit Stream File Format
------------------ Average data all frames  -----------------------------------
SNR Y(dB)                        : 40.59
SNR U(dB)                        : 39.24
SNR V(dB)                        : 39.77
```

Figure 158 — Summary of results reported by JM 8.5 after embedding the SystemC block.

10.6.10 Limitations

Increased area is the main limitation of the design.

10.7 A VHDL HARDWARE IP BLOCK FOR 4X4 HADAMARD TRANSFORM AND QUANTIZATION FOR MPEG-4 PART 10 AVC

10.7.1 Abstract description of the module

This subclause presents a hardware prototype for the 4x4 Hadamard transform and quantization that is applied to the DC coefficients of the luma component when the macroblock is encoded in 16 x 16 intra prediction mode. The implemented transform represents the second level in the transformation hierarchy that it is adopted by the MPEG-4 Part 10 AVC standard. It comes after the forward 4 x 4 integer approximation of the DCT transform. The architecture is prototyped and simulated using ModelSim 5.4®. It is synthesized using Leonardo Spectrum®. The results show that the architecture satisfies the real-time constraints required by different digital video applications.

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.7.2 Module specification

10.7.2.1 MPEG 4 part: ***10***
10.7.2.2 Profile : ***All***
10.7.2.3 Level addressed: ***All***
10.7.2.4 Module Name: ***4x4 Hadamard (VHDL)***
10.7.2.5 Module latency: ***375.76 ns***
10.7.2.6 Module data throughput: ***A 4x4 parallel quantized transform coefficients matrix/sec***
10.7.2.7 Max clock frequency: ***36.6 MHz***
10.7.2.8 Resource usage:
10.7.2.8.1 CLB Slices: ***4019***
10.7.2.8.2 DFFs or Latches: ***3877***
10.7.2.8.3 Function Generators: ***8038***
10.7.2.8.4 External Memory: ***none***
10.7.2.8.5 Number of Gates: ***7890***
10.7.2.9 Revision: ***1.00***
10.7.2.10 Authors: ***Ihab Amer, Wael Badawy, and Graham Jullien***
10.7.2.11 Creation Date: ***July 2004***
10.7.2.12 Modification Date: ***October 2004***

10.7.3 Introduction

In the present contribution, a hardware prototype for the 4×4 Hadamard transform that is applied to the DC coefficients of the luma component when the macroblock is encoded in 16×16 intra prediction mode is introduced. The proposed architecture is developed to use only add operations to reduce the computational requirements for the transform.

10.7.4 Functional Description

10.7.4.1 Functional description details

This subclause introduces the proposed hardware prototype of the 4×4 Hadamard transform and quantization adopted by the MPEG-4 Part 10 standard. It is applied to the DC coefficients of the sixteen 4×4 blocks of the luma component. The proposed architecture uses 4×4 parallel input block.

10.7.4.2 I/O Diagram

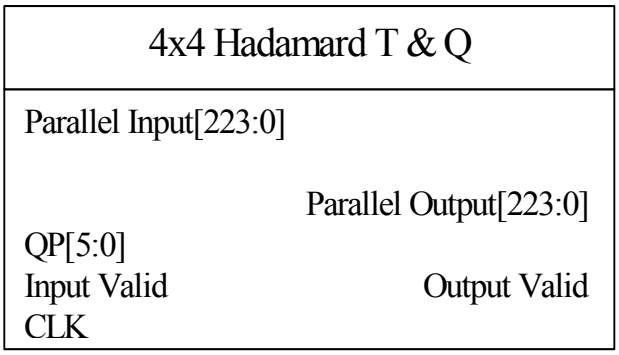


Figure 159

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.7.4.3 I/O Ports Description

Table 37

Port Name	Port Width	Direction	Description
Parallel Input[223:0]	224	Input	4x4 matrix of DC coefficients
QP[5:0]	6	Input	Quantization Parameter
Input Valid	1	Input	Flag indicating that input is valid
CLK	1	Input	System clock
Parallel Output[223:0]	224	Output	4x4 parallel quantized transform coefficients matrix
Output Valid	1	Output	Flag indicating that output is valid

10.7.5 Algorithm

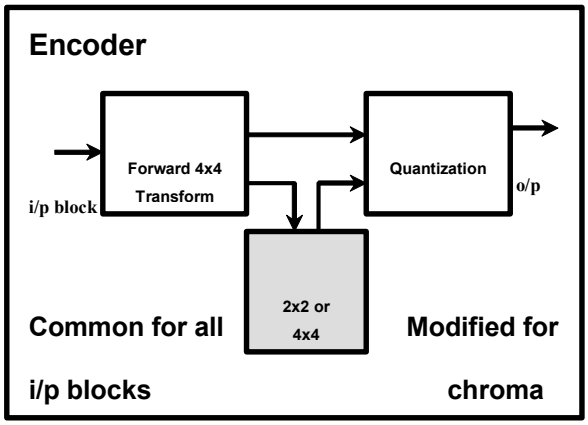


Figure 160 — Hierarchical transform and quantization in AVC standard.

A hierarchical transform is adopted in the MPEG-4 Part 10 standard. Figure 160 provides a block diagram showing the hierarchy of transform before the quantization process in the encoder-side.

Step 1 is an integer orthogonal approximation to the Discrete Cosine Transform (DCT) with a 4×4 input block, which allows for bit-exact implementation for all encoders and decoders [21].

Step 2 is a 4×4 Hadamard transform to the DC coefficients (from Step 1). It reduces the reconstruction error for intra-16 prediction mode. The cascading of block transforms is equivalent to an extension to the length of the transform functions [21].

The Hadamard transform formula that is applied to a 4×4 array (W) of DC coefficients of the luma component is shown in Equation 35. The output coefficients are divided by 2 (with rounding).

$$Y = (HWH^T)/2$$

Equation 35

The Matrix H is given by Equation 36.

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix}$$

Equation 36

The formulas for post-scaling and quantization of transformed intra-16 mode luma DC coefficients expressed in integer arithmetic are shown in Equation 37, Equation 38, and Equation 39.

$$qbits = 15 + (QP \ DIV \ 6)$$

Equation 37

$$|Z_{ij}| = SHR(|Y_{ij}|.MF + 2f, qbits + 1)$$

Equation 38

$$Sign(Z_{ij}) = Sign(Y_{ij})$$

Equation 39

QP is a quantization parameter that enables the encoder to control the trade-off between bit rate and quality. It can take any integer value from 0 up to 51. Z_{ij} is an element in the output quantized DC coefficients matrix. MF is a multiplication factor in order to avoid any division operation. It depends on QP as shown in Table 38. *SHR()* is a procedure that right-shifts the result of its first argument a number of bits equal to its second argument. *f* is defined in the reference model software as $2^{qbits}/3$ for intra blocks and $2^{qbits}/6$ for Inter blocks [15].

Table 38 — Multiplication Factor (*MF*).

QP	**MF**
0	13107
1	11916
2	10082
3	9362
4	8192
5	7282

The factor MF does not change for $QP > 5$. It can be calculated using Equation 40.

$$MF_{QP>5} = MF_{QP=QP \bmod 6}$$

Equation 40

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.7.6 Implementation

The architecture can be integrated on the same chip with the forward 4×4 transform that is adopted by the AVC standard and/or with the 2×2 Hadamard transform that is applied to the DC coefficients of the four 4×4 blocks of each chroma component [25][26].

The architecture uses pipelined stages which increase the throughput. At steady state, the proposed architecture outputs an encoded block at each clock pulse. The architecture does not contain memory elements. Instead, a redundancy in computational elements is used.

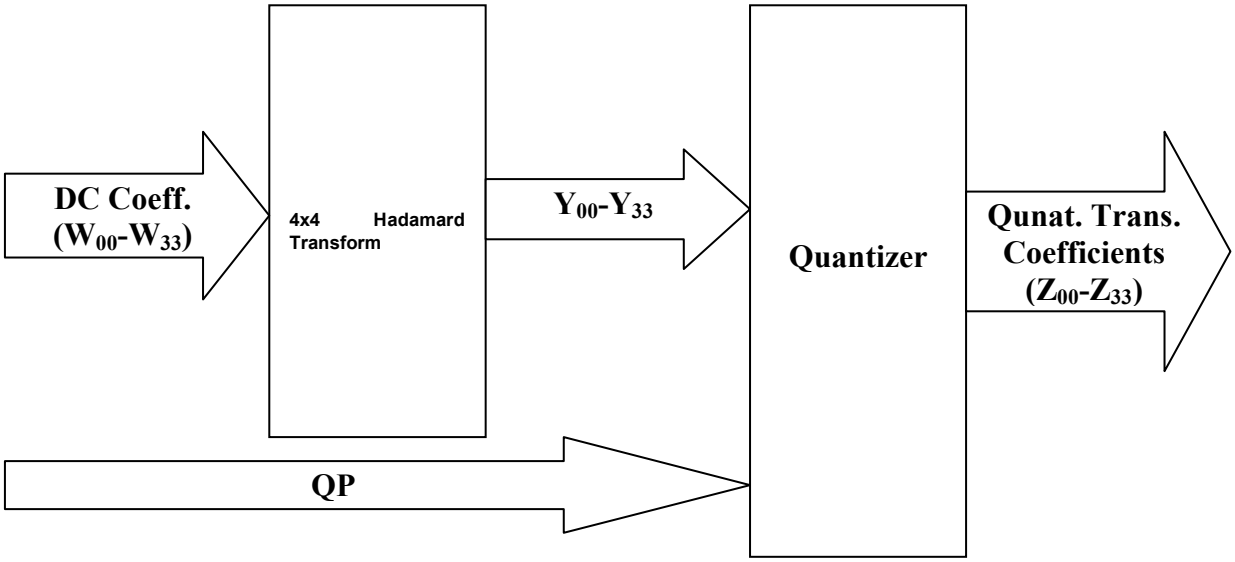


Figure 161 — Flow of signals between the two main stages of the design, the transformer and the quantizer.

A flow graph of the used 4×4 Hadamard transform is shown in Figure 161. The transformation is performed in two stages. Each of them is responsible for multiplying two 4×4 matrices. Each is composed of four identical butterfly-adders. Its function is to perform a group of additions. In Figure 162 (a), the first butterfly-adder block in the first sub-block is shown, while in Figure 162 (b), the first butterfly-adder block in the next sub-block is shown. The Transform block is the hardware implementation that corresponds to Equation 35.

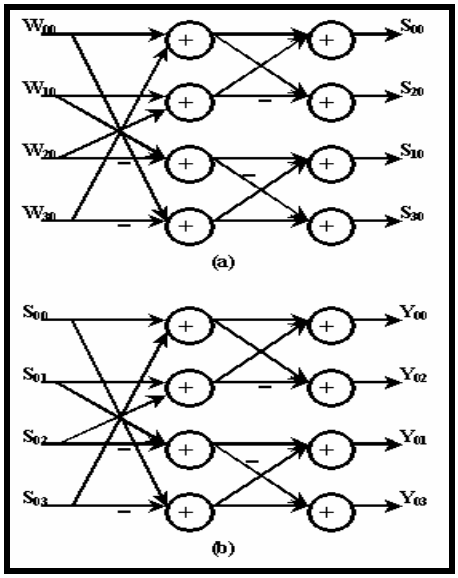


Figure 162 — First butterfly-adder block in: (a) First sub-block, (b) Second sub-block.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

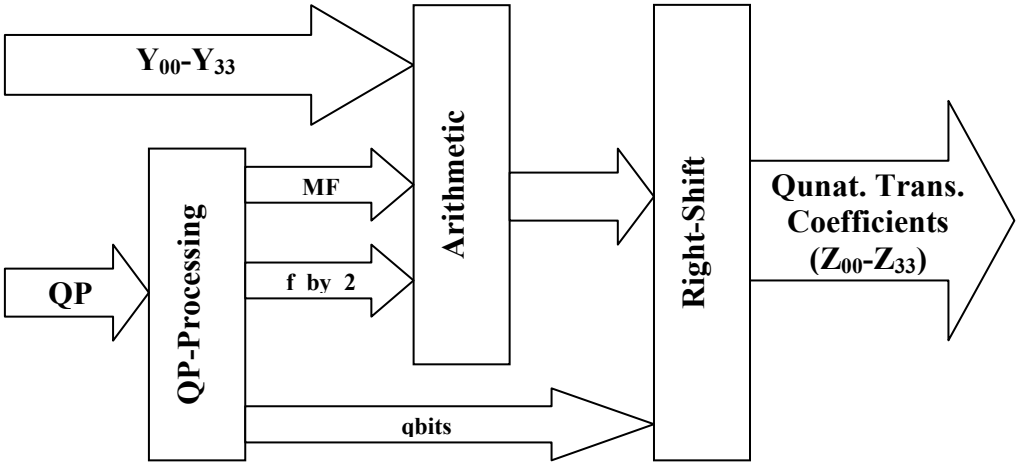


Figure 163 — A detailed diagram of the quantizer.

Figure 163 gives a detailed description of the quantizer. Quantization is performed in three different stages, each having its specific task. In the *QP-Processing* stage, QP is used to calculate the values of *qbits*, *f_by_2* as well as *MF*, which is a multiplication factor that is based on *QP* as shown in Table 38. The *Arithmetic* block is responsible for performing multiplication and addition operations. Finally, the *Right-Shift* block shifts the output from the Arithmetic block a number of bits equal to *qbits+1*.

10.7.6.1 Interfaces

N/A.

10.7.6.2 Register File Access

Please refer to subclause 10.7.4.

10.7.6.3 Timing Diagrams

N/A.

10.7.7 Results of Performance & Resource Estimation

The architecture is prototyped using VHDL language and simulated using the Mentor Graphics© ModelSim 5.4®, then synthesized using Leonardo Spectrum®.

The architecture is a hardware reference model for the MPEG-4 Part 10 AVC and the target implementation technology is the FPGA device (2V3000fg676) from the Virtex-II family of Xilinx©.

Table 39 summarizes the performance of the prototyped architecture. The critical path is 26.84 *ns*, which is equivalent to a maximum clock frequency of 36.6 MHz. The proposed prototype provides a 4×4 encoded block (16 pixels into 16 quantized transform coefficients) with each clock pulse at steady state. Therefore the latency to encode a CIF frame (325×288 pixels) is calculated as follows:

Time required per CIF frame =

Time required per block $\times$ *Number of blocks per frame*

$$= 26.84\ ns \times \frac{Number\ of\ \ pixels\ per\ frame}{Number\ of\ \ pixels\ per\ block}$$

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

$$= 26.84\ ns \times \frac{(352 \times 288)\ pixels\ per\ frame}{(4 \times 4)\ pixels\ per\ block}$$

$$\simeq 0.17\ ms$$

Table 39 — Performance of the prototyped architecture.

Critical Path (ns)	CLK Freq. (MHz)	# of Gates	# of I/O Ports
26.84	36.6	7890	455
# of Nets	**# of DFF's or Latches**	**# Function Generators**	**# of CLB Slices**
910	3877	8038	4019

The system allows the computation of 195 CIF frames per second at 36.6 Mhz. Similarly, it encodes a whole High Definition Television (HDTV) frame of a 720×1280 pixels resolution, and a 60 *frames/sec* frame rate in 1.54 *ms*. This is about 10.7 times less than the 16.6 *ms* standard time. Hence, the proposed architecture is suitable to be used in even higher resolution systems than the HDTV systems.

10.7.8 API calls from reference software

N/A.

10.7.9 Conformance Testing

The architecture has not been tested on the platform but its simulation results conforms with the software results.

10.7.9.1 Reference software type, version and input data set

N/A.

10.7.9.2 API vector conformance

N/A.

10.7.9.3 End to end conformance (conformance of encoded bitstreams or decoded pictures)

N/A.

10.7.10 Limitations

Increased area is the main limitation of the design.

10.8 A HARDWARE BLOCK FOR THE MPEG-4 PART 10 4X4 DCT-LIKE TRANSFORMATION AND QUANTIZATION

10.8.1 Abstract description of the module

The 4x4 forward DCT transform adopted in the MPEG-4 Part 10 (AVC) standard is an integer orthogonal approximation to the DCT. This allows for bit-exact implementation for all encoders and decoders. Another important feature in the new standard is the removal of the computationally expensive multiplications that appears in the conventional standards, which are based on the traditional DCT formulation.

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.8.2 Module specification

10.8.2.1	**MPEG 4 part:**	***10***
10.8.2.2	**Profile :**	***All***
10.8.2.3	**Level addressed:**	***All***
10.8.2.4	**Module Name:**	***4x4 DCT-Like (VHDL)***
10.8.2.5	**Module latency:**	***481.1 ns***
10.8.2.6	**Module data throughput:**	***A 4x4 parallel quantized transform coefficients matrix/sec***
10.8.2.7	**Max clock frequency:**	***34.8 MHz***
10.8.2.8	**Resource usage:**	
10.8.2.8.1	**CLB Slices:**	***3204***
10.8.2.8.2	**DFFs or Latches:**	***4156***
10.8.2.8.3	**Function Generators:**	***6407***
10.8.2.8.4	**External Memory:**	***none***
10.8.2.8.5	**Number of Gates:**	***6212***
10.8.2.9	**Revision:**	***1.00***
10.8.2.10	**Authors:**	***Ihab Amer, Wael Badawy, and Graham Jullien***
10.8.2.11	**Creation Date:**	***July 2004***
10.8.2.12	**Modification Date:**	***October 2004***

10.8.3 Introduction

The AVC standard does not use the traditional 8×8 Discrete Cosine Transform (DCT) as the basic transform. Instead, a new 4×4 transform is introduced that can be computed exactly in integer arithmetic, thus avoiding inverse transform mismatch problem [21].

The transform can be computed without multiplications, just additions and shifts, in 16-bit arithmetic. This minimizes the computational complexity significantly. Besides, the quantization operation uses multiplications avoiding unsynthesisable divisions.

To develop a hardware video codec for AVC, a VLSI architecture is required. Surveying the literature shows that there are few architectures that prototype the new 4×4 transformation.

10.8.4 Functional Description

10.8.4.1 Functional description details

This subclause introduces the proposed hardware prototype of the 4×4 forward transform and quantization adopted by the MPEG-4 Part 10 AVC standard. It is applied to the parallel 4×4 input pixels blocks of the luma component. A block diagram of the architecture showing its inputs and outputs is given in Figure 164.

10.8.4.2 I/O Diagram

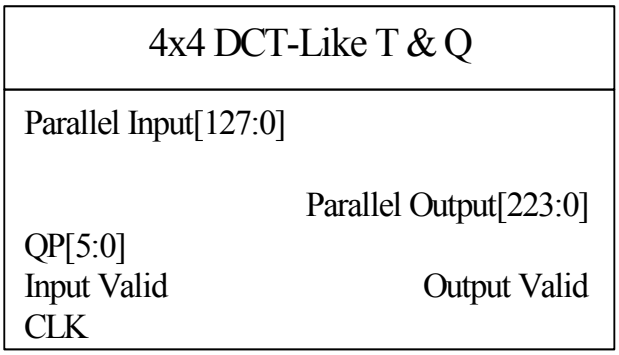


Figure 164 — A block diagram of the hardware architecture.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.8.4.3 I/O Ports Description

Table 40

Port Name	Port Width	Direction	Description
Parallel Input[127:0]	128	Input	4x4 parallel matrix of pixels
QP[5:0]	6	Input	Quantization Parameter
Input Valid	1	Input	Flag indicating that input is valid
CLK	1	Input	System clock
Parallel Output[223:0]	224	Output	4x4 parallel quantized transform coefficients matrix
Output Valid	1	Output	Flag indicating that output is valid

10.8.5 Algorithm

The encoder transform formula that is proposed by the JVT to be applied to an input 4×4 block is shown in Equation 41.

$$W = C_f X C_f^{\,T}$$

Equation 41

where the Matrix C_f is given by Equation 42.

$$C_f = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix}$$

Equation 42

In Equation 42, the absolute values of all the coefficients of the C_f matrix are either 1 or 2. Thus, the transform operation represented by Equation 41 can be computed using signed additions and left-shifts only to avoid expensive multiplications.

The post-scaling and quantization formulas are shown in Equation 43 and Equation 44.

$$qbits = 15 + (QP\ DIV\ 6)$$

Equation 43

$$Z_{ij} = round(W_{ij} \cdot \frac{MF}{2^{qbits}})$$

Equation 44

where QP is a quantization parameter that enables the encoder to accurately and flexibly control the trade-off between bit rate and quality. It can take any integer value from 0 up to 51. Z_{ij} is an element in the matrix that

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

results from the quantization process. MF is a multiplication factor that depends on QP and the position (i, j) of the element in the matrix as shown in Table 41.

Table 41 — Multiplication Factor (*MF*).

QP	(i, j) ∈ {(0, 0), (2, 0), (2, 2), (0, 2)}	(i, j) ∈ {(1, 1), (1, 3), (3, 1), (3, 3)}	Other Positions
0	13107	5243	8066
1	11916	4660	7490
2	10082	4194	6554
3	9362	3647	5825
4	8192	3355	5243
5	7282	2893	4559

The factor MF remains unchanged for $QP > 5$. It can be calculated using Equation 45.

$$MF_{QP>5} = MF_{QP=QP \bmod 6}$$

Equation 45

Equation 44 can be represented using integer arithmetic as follows:

$$\left|Z_{ij}\right| = SHR(\left|W_{ij}\right|.MF + f, qbits)$$

Equation 46

$$Sign(Z_{ij}) = Sign(W_{ij})$$

Equation 47

where *SHR()* is a procedure that right-shifts the result of its first argument a number of bits equal to its second argument. *f* is defined in the reference model software as $2^{qbits}/3$ for Intra blocks and $2^{qbits}/6$ for Inter blocks [15].

10.8.6 Implementation

The architecture is designed to perform pipelined operations. Therefore, with the exception of the first 4×4 input block; the illustrated architecture can output a whole encoded block with each clock pulse. The architecture does not contain memory elements. Instead, a redundancy in computational elements shows up. This introduces an example of performance-area tradeoff.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

A detailed description of the architecture is shown in Figure 165. The architecture is composed of three main stages: The Register File stage, The Transform and the QP-Processing stage, and The Quantization stage.

Data is initially captured from the outside environment and stored in the Register File. Then the 4×4 input block is passed to the Transform block. This block consists of two cascaded sub-blocks. Each of them is responsible of multiplying two 4×4 matrices and is composed of four identical butterfly-adder blocks. Its operation is to perform a group of additions and shifts Figure 166(a) shows the first butterfly-adder block in the first sub-block, while Figure 166 (b) shows the first butterfly-adder block in the next sub-block. The Transform block is the hardware implementation that corresponds to Equation 41. At the same time, the QP-Processing block is responsible for calculating *f*, *qbits*, and determining P_1, P_2, and P_3, which are the values of the multiplication factors at the three different groups of positions in the matrix as shown in Table 41. Finally, the Quantization process takes place in the last-stage block. The integer division by six that is required for implementing Equation 42 and Equation 45 is implemented by recursive subtraction.

Signed numbers are represented in the whole architecture using the standard signed two's complement representation.

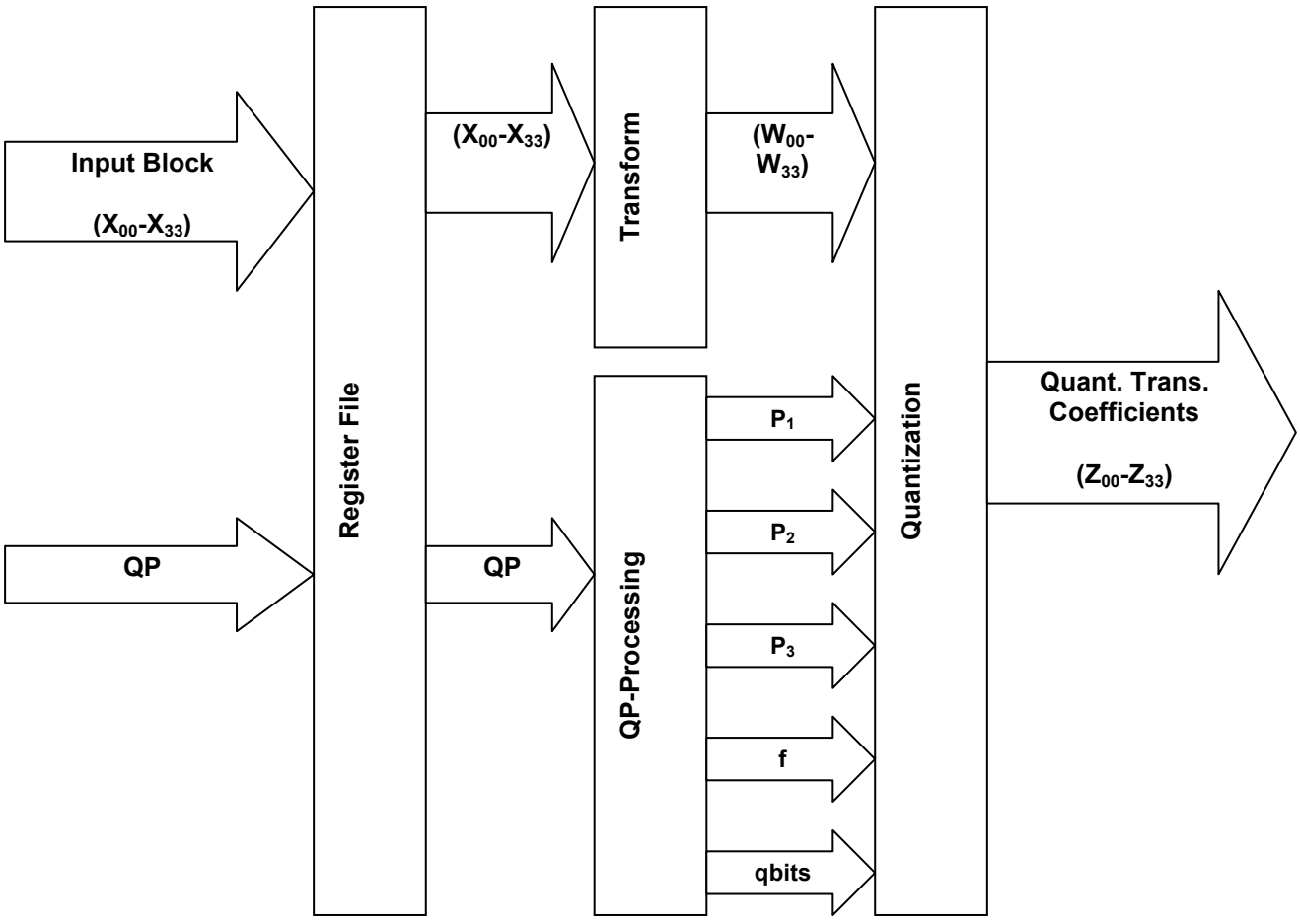


Figure 165 — A detailed block diagram of the hardware architecture.

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

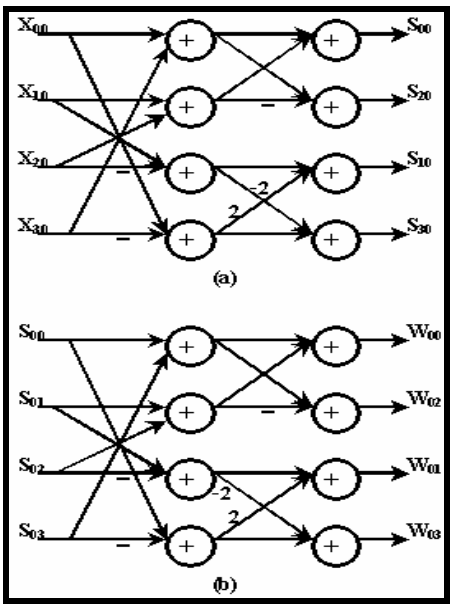


Figure 166 — First butterfly-adder block in: (a) First sub-block, (b) Second sub-block.

10.8.6.1 Interfaces

N/A.

10.8.6.2 Register File Access

Please refer to subclause 10.8.4.

10.8.6.3 Timing Diagrams

N/A.

10.8.7 Results of Performance & Resource Estimation

The architecture for the MPEG-4 part 10 AVC 4×4 transformation is prototyped using VHDL language. It is simulated using the Mentor Graphics© ModelSim 5.4® simulation tool, and synthesized using Leonardo Spectrum®. The target technology is the FPGA device (2V3000fg676) from the Virtex-II family of Xilinx©.

The correctness of the implemented architecture is also checked. This is done by passing different input patterns to the architecture and comparing the output with the results obtained by passing the same inputs to the equations of subclause 10.8.5.

Table 42 summarizes the performance of the prototyped architecture.

Table 42 — Performance of the prototyped architecture.

Critical Path (ns)	Clk Freq. (MHz)	# Of Gates	# Of Ports	# Of Nets
28.3	34.8	6212	359	718
# Dff's or Latches	**# Func. Generators**	**# CLB Slices**	**# B. Box Adders**	**# B. Box Subtr.**
4156	6407	3204	8	8

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The critical path is estimated by the synthesis tool to be 28.3 *ns*. Since the chip outputs a whole 4×4 encoded block with each clock pulse (except for the first block), therefore the time required to encode a whole CIF frame (325×288 pixels) can be calculated as follows:

Time required per CIF frame =

Time required per block $\times$ *Number of blocks per frame*

$$= 28.3\ ns \times \frac{Number\ of\ pixels\ per\ frame}{Number\ of\ pixels\ per\ block}$$

$$= 28.3\ ns \times \frac{(352 \times 288)\ pixels\ per\ frame}{(4 \times 4)\ pixels\ per\ block}$$

$$\simeq 0.18\ ms$$

This value is 185 times less than the 33.3 *ms* standard time (assuming 29.97 *frames/sec*) required for frame encoding. Similarly, it can be shown that the time required to encode a whole High Definition Television (HDTV) frame of a 720×1280 pixels resolution, and a 60 *frames/sec* frame rate is 1.63 *ms*, which is about 10 times less than the 16.6 *ms* standard time. This leads to the suggestion of taking the input serially, integrating other operations on the same encoder chip, or targeting other applications that use more complicated-higher resolution video formats.

10.8.8 API calls from reference software

N/A.

10.8.9 Conformance Testing

The architecture has not been tested on the platform but its simulation results conforms with the software results.

10.8.9.1 Reference software type, version and input data set

N/A.

10.8.9.2 API vector conformance

N/A.

10.8.9.3 End to end conformance (conformance of encoded bitstreams or decoded pictures)

N/A.

10.8.10 Limitations

Increased area is the main limitation of the design.

10.9 A SYSTEMC MODEL FOR THE MPEG-4 PART 10 4X4 DCT-LIKE TRANSFORMATION AND QUANTIZATION

10.9.1 Abstract description of the module

This subclause presents a SystemC hardware prototype of the AVC transformation. The proposed architecture uses only add and shift operations to reduce the computational requirements for the 4×4 transform. The architecture is developed to be used in high-resolution applications such as High Definition Television (HDTV) and Digital Cinema.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.9.2 Module specification

10.9.2.1 MPEG 4 part: ***10***
10.9.2.2 Profile : ***All***
10.9.2.3 Level addressed: ***All***
10.9.2.4 Module Name: ***4x4 DCT-Like (VHDL)***
10.9.2.5 Module latency: ***N/A***
10.9.2.6 Module data throughput: ***A 4x4 parallel quantized transform coefficients matrix/ CC***
10.9.2.7 Max clock frequency: ***N/A***
10.9.2.8 Resource usage:
10.9.2.8.1 CLB Slices: ***N/A***
10.9.2.8.2 DFFs or Latches: ***N/A***
10.9.2.8.3 Function Generators: ***N/A***
10.9.2.8.4 External Memory: ***N/A***
10.9.2.8.5 Number of Gates: ***N/A***
10.9.2.9 Revision: ***1.00***
10.9.2.10 Authors: ***Ihab Amer, Wael Badawy, and Graham Jullien***
10.9.2.11 Creation Date: ***July 2004***
10.9.2.12 Modification Date: ***October 2004***

10.9.3 Introduction

The AVC standard does not use the traditional 8×8 Discrete Cosine Transform (DCT) as the basic transform. Instead, a new 4×4 transform is introduced that can be computed exactly in integer arithmetic, thus avoiding inverse transform mismatch problem [21].

The transform can be computed without multiplications, just additions and shifts, in 16-bit arithmetic. This minimizes the computational complexity significantly. Besides, the quantization operation uses multiplications avoiding unsynthesisable divisions.

To develop a hardware video codec for AVC, a VLSI architecture is required. Surveying the literature shows that there are few architectures that prototype the new 4×4 transformation.

10.9.4 Functional Description

10.9.4.1 Functional description details

This subclause introduces the proposed hardware prototype of the 4×4 forward transform and quantization adopted by the MPEG-4 Part 10 standard. It is applied to the parallel 4×4 input pixels blocks of the luma component. A block diagram of the architecture showing its inputs and outputs is given in Figure 167.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.9.4.2 I/O Diagram

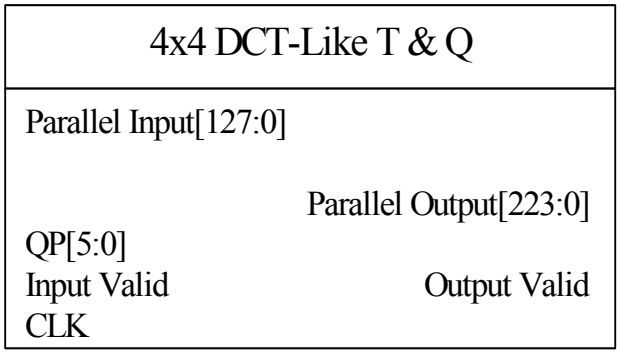


Figure 167 — A block diagram of the proposed hardware architecture.

10.9.4.3 I/O Ports Description

Table 43

Port Name	Port Width	Direction	Description
Parallel Input[127:0]	128	Input	4x4 parallel matrix of pixels
QP[5:0]	6	Input	Quantization Parameter
Input Valid	1	Input	Flag indicating that input is valid
CLK	1	Input	System clock
Parallel Output[223:0]	224	Output	4x4 parallel quantized transform coefficients matrix
Output Valid	1	Output	Flag indicating that output is valid

10.9.5 Algorithm

The encoder transform formula that is proposed by the JVT to be applied to an input 4×4 block is shown in Equation 48.

$$W = C_f X C_f^{\,T}$$

Equation 48

where the Matrix C_f is given by Equation 49.

$$C_f = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix}$$

Equation 49

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

In Equation 49, the absolute values of all the coefficients of the C_f matrix are either 1 or 2. Thus, the transform operation represented by Equation 48 can be computed using signed additions and left-shifts only to avoid expensive multiplications.

The post-scaling and quantization formulas are shown in Equation 50 and Equation 51.

$$qbits = 15 + (QP\ DIV\ 6)$$

Equation 50

$$Z_{ij} = round(W_{ij} . \frac{MF}{2^{qbits}})$$

Equation 51

where QP is a quantization parameter that enables the encoder to accurately and flexibly control the trade-off between bit rate and quality. It can take any integer value from 0 up to 51. Z_{ij} is an element in the matrix that results from the quantization process. MF is a multiplication factor that depends on QP and the position (i, j) of the element in the matrix as shown in Table 44.

Table 44 — Multiplication Factor (*MF*).

QP	(i, j) ∈ {(0, 0), (2, 0), (2, 2), (0, 2)}	(i, j) ∈ {(1, 1), (1, 3), (3, 1), (3, 3)}	Other Positions
0	13107	5243	8066
1	11916	4660	7490
2	10082	4194	6554
3	9362	3647	5825
4	8192	3355	5243
5	7282	2893	4559

The factor MF remains unchanged for $QP > 5$. It can be calculated using Equation 52.

$$MF_{QP>5} = MF_{QP=QP \bmod 6}$$

Equation 52

Equation 51 can be represented using integer arithmetic as follows:

$$|Z_{ij}| = SHR(|W_{ij}| .MF + f, qbits)$$

Equation 53

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

$$Sign(Z_{ij}) = Sign(W_{ij})$$

Equation 54

where *SHR()* is a procedure that right-shifts the result of its first argument a number of bits equal to its second argument. *f* is defined in the reference model software as $2^{qbits} / 3$ for Intra blocks and $2^{qbits} / 6$ for Inter blocks [15].

10.9.6 Implementation

The architecture is designed to perform pipelined operations. Therefore, with the exception of the first 4×4 input block; the illustrated architecture can output a whole encoded block with each clock pulse. The architecture does not contain memory elements. Instead, a redundancy in computational elements shows up. This introduces an example of performance-area tradeoff.

A detailed description of the architecture is shown in Figure 168. The architecture is composed of three main stages: The Register File stage, The Transform and the QP-Processing stage, and The Quantization stage.

Data is initially captured from the outside environment and stored in the Register File. Then the 4×4 input block is passed to the Transform block. This block consists of two cascaded sub-blocks. Each of them is responsible of multiplying two 4×4 matrices and is composed of four identical butterfly-adder blocks. Its operation is to perform a group of additions and shifts. Figure 169 (a) shows the first butterfly-adder block in the first sub-block, while Figure 169 (b) shows the first butterfly-adder block in the next sub-block. The Transform block is the hardware implementation that corresponds to Equation 48. At the same time, the QP-Processing block is responsible for calculating *f*, *qbits*, and determining P_1, P_2, and P_3, which are the values of the multiplication factors at the three different groups of positions in the matrix as shown in Table 44. Finally, the Quantization process takes place in the last-stage block. The integer division by six that is required for implementing Equation 49 and Equation 52 is implemented by recursive subtraction.

Signed numbers are represented in the whole architecture using the standard signed two's complement representation.

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

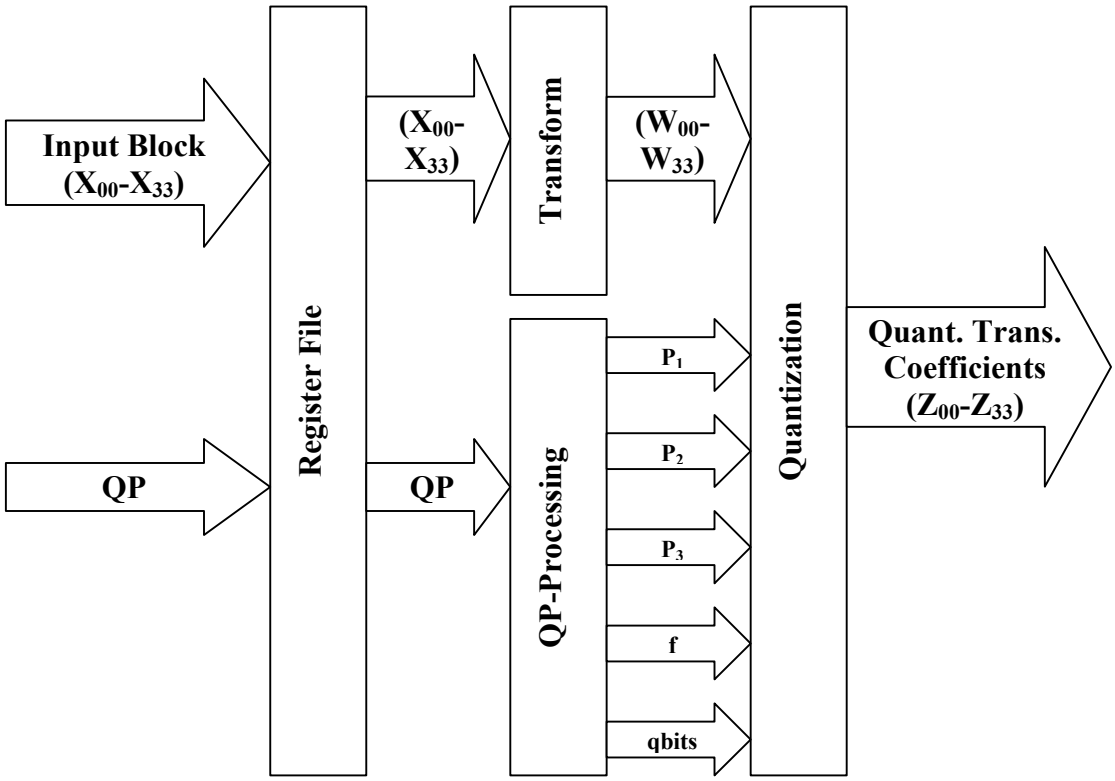


Figure 168 — A detailed block diagram of the hardware architecture.

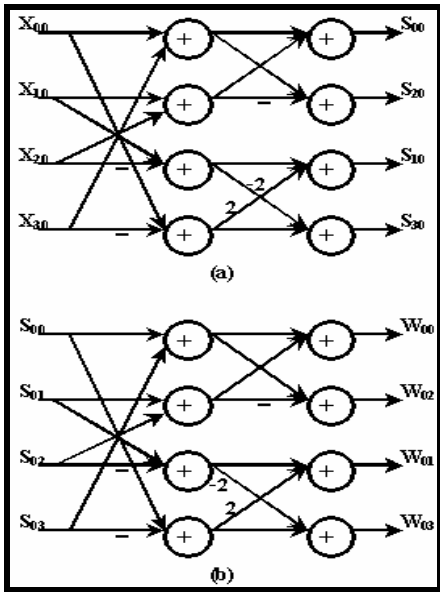


Figure 169 — First butterfly-adder block in (a) First sub-block, (b) Second sub-block.

10.9.6.1 Interfaces

N/A.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.9.6.2 Register File Access

Please refer to subclause 10.9.4.

10.9.6.3 Timing Diagrams

N/A.

10.9.7 Results of Performance & Resource Estimation

Behavioural simulation shows that the designed architecture functionally complies with the reference software. The architecture was embedded in JM 8.5 and its output stream was identical to the output from the original software. Figure 170 gives a comparison between the outputs before and after the embedding the SystemC block.



(a) **(b)**

Figure 170 — (a) Output before embedding the SystemC block (b) Output after embedding the SystemC block.

10.9.8 API calls from reference software

The switching between software and hardware is controlled by flags that can be reset to avoid switching, hence the software flow will execute normally bypassing the SystemC block. An example of a HW-block call from SW is as follows:

```
/*----------------Hardware/Software Switch --------------*/

 if(DCT_HW_ACCELERATOR){

       sc_dct(img->m7, 0, 0, firstHW_Call);

       firstHW_Call = 0;

 }

 else

     sw_dct(img->m7, 0, 0);

/*-------------------------------------------------------------*/
```

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.9.9 Conformance Testing

10.9.9.1 Reference software type, version and input data set

The functional testing was performed on the MPEG-4 Part 10 AVC reference software (JM8.5). The video test sequences are "Miss America" and "Foreman".

10.9.9.2 API vector conformance

The test vectors used are QCIF test sequences.

10.9.9.3 End to end conformance (conformance of encoded bitstreams or decoded pictures)

The end-to-end encoder conformance test is evaluated via a mixed C and SystemC environment using the JM 8.5 software reference model. Figure 171 and Figure 172 show that the results obtained before and after using the hardware accelerators are identical.

```
Freq. for encoded bitstream      : 30
Hadamard transform               : Used
Image format                     : 176x144
Error robustness                 : Off
Search range                     : 16
No of ref. frames used in P pred : 10
Total encoding time for the seq. : 3.876 sec
Total ME time for sequence       : 1.041 sec
Sequence type                    : IPPP (QP: I 28, P 28)
Entropy coding method            : CAVLC
Profile/Level IDC                : (66,30)
Search range restrictions        : none
RD-optimized mode decision       : used
Data Partitioning Mode           : 1 partition
Output File Format               : H.264 Bit Stream File Format
------------------ Average data all frames  -----------------------------------
SNR Y(dB)                        : 40.59
SNR U(dB)                        : 39.24
SNR V(dB)                        : 39.77
```

Figure 171 — Summary of results reported by JM 8.5 before embedding the SystemC block.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
Freq. for encoded bitstream      : 30
Hadamard transform               : Used
Image format                     : 176x144
Error robustness                 : Off
Search range                     : 16
No of ref. frames used in P pred  : 10
Total encoding time for the seq.  : 37.165 sec
Total ME time for sequence       : 0.985 sec
Sequence type                    : IPPP (QP: I 28, P 28)
Entropy coding method            : CAVLC
Profile/Level IDC                : (66,30)
Search range restrictions        : none
RD-optimized mode decision       : used
Data Partitioning Mode           : 1 partition
Output File Format               : H.264 Bit Stream File Format
------------------ Average data all frames  -----------------------------------
SNR Y(dB)                        : 40.59
SNR U(dB)                        : 39.24
SNR V(dB)                        : 39.77
Total bits                       : 12408 (I 10896, P  1344, NVB 168)
```

Figure 172 — Summary of results reported by JM 8.5 after embedding the SystemC block.

10.9.10 Limitations

Increased area is the main limitation of the design.

10.10 A 8X8 INTEGER APPROXIMATION DCT TRANSFORMATION AND QUANTIZATION SYSTEMC IP BLOCK FOR MPEG-4 PART 10 AVC

10.10.1 Abstract description of the module

The recently approved digital video standard known as AVC promises to be an excellent video format for use with a large range of applications. Real-time encoding/decoding is a main requirement for adoption of the

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

standard to take place in the consumer marketplace. Transformation and quantization in AVC are relatively less complex than their correspondences in other video standards. Nevertheless, for real-time operation, a speedup is required for such processes. Especially after the recent proposal to use an 8x8 integer approximation of Discrete Cosine Transform (DCT) to give significant compression performance at Standard Definition (SD) and High Definition (HD) resolutions. This subclause is to propose a SystemC prototype of a high-performance hardware implementation of the AVC simplified 8x8 transformation and quantization. The results show that the architecture satisfies the real-time constraints required by different digital video applications.

10.10.2 Module specification

10.10.2.1 MPEG 4 part: ***10***
10.10.2.2 Profile : ***All***
10.10.2.3 Level addressed: ***All***
10.10.2.4 Module Name: ***8x8 DCT-like (SystemC)***
10.10.2.5 Module latency: ***N/A***
10.10.2.6 Module data throughput: ***An 8x8 parallel quantized transform coefficients matrix/ CC***
10.10.2.7 Max clock frequency: ***N/A***
10.10.2.8 Resource usage:
10.10.2.8.1 CLB Slices: ***N/A***
10.10.2.8.2 DFFs or Latches: ***N/A***
10.10.2.8.3 Function Generators: ***N/A***
10.10.2.8.4 External Memory: ***N/A***
10.10.2.8.5 Number of Gates: ***N/A***
10.10.2.9 Revision: ***1.00***
10.10.2.10 Authors: ***Ihab Amer, Wael Badawy, and Graham Jullien***
10.10.2.11 Creation Date: ***March 2005***
10.10.2.12 Modification Date: ***March 2005***

10.10.3 Introduction

Unlike current standards, the usual floating-point 8x8 DCT is not the basic transformation in AVC. Instead, a new transformation hierarchy is introduced that can be computed exactly in integer arithmetic. This eliminates any mismatch issues between the encoder and the decoder in the inverse transform [14][21]. In the initial AVC standard, which was completed in May 2003, the transformation is primarily 4x4 in shape, which helps reduce blocking and ringing artifacts.

In July 2004, a new amendment called the Fidelity Range Extensions (FRExt, Amendment I) was added to the AVC standard. This amendment is currently receiving wide attention in the industry. It actually demonstrates further coding efficiency against current video coding standards, potentially by as much as 3:1 for some key applications. The FRExt project produced a suite of some new profiles collectively called High profiles. Beside supporting all features of the prior Main profile, all the High profiles support an adaptive transform-block size and perceptual quantization scaling matrices [29]. In fact, the concept of adaptive transform-block size has proven to be an efficient coding tool within AVC video coding layer design. This led to the proposal of a seamless integration of a new 8x8 integer approximation of DCT (and prediction modes) into the specification with the least possible amount of technical and syntactical changes.

10.10.4 Functional Description

10.10.4.1 Functional description details

This subclause introduces the proposed hardware prototype of the 8×8 forward transform and quantization adopted by FRExt in the MPEG-4 Part 10 standard. It is applied to the parallel 8×8 input pixels blocks of the luma component. A block diagram of the architecture showing its inputs and outputs is given in Figure 173.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.10.4.2 I/O Diagram

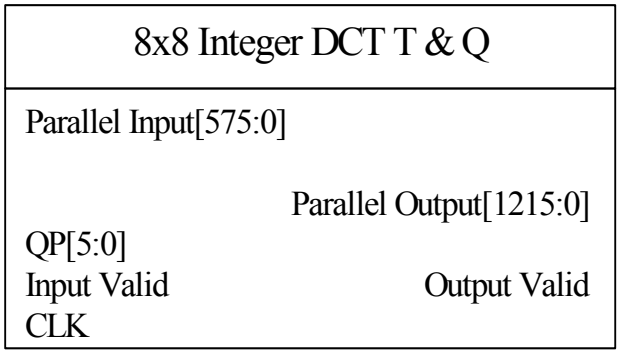


Figure 173 — A block diagram of the hardware architecture.

10.10.4.3 I/O Ports Description

Table 45

Port Name	Port Width	Direction	Description
Parallel Input[127:0]	575	Input	8x8 parallel matrix of pixels
QP[5:0]	6	Input	Quantization Parameter
Input Valid	1	Input	Flag indicating that input is valid
CLK	1	Input	System clock
Parallel Output[223:0]	1215	Output	8x8 parallel quantized transform coefficients matrix
Output Valid	1	Output	Flag indicating that output is valid

10.10.5 Algorithm

An integer approximation of 8x8 DCT was proposed in FREext to be added to the JVT specification based on the fact that at SD resolutions and above, the use of block sizes smaller than 8x8 is limited. This transform is applied to each block in the luminance component of the input video stream. It allows for bit-exact implementation for all encoders and decoders. In spite of being more complex compared to the 4x4 DCT-like transform that is adopted by the initial AVC specification, the proposed transform gives excellent compression performance when used for high-resolution video streams using a number of operations comparable to the number of operations required for the corresponding four 4x4 blocks using the fast butterfly implementation of the existing 4x4 transform.

The 2-D forward 8x8 transform is computed in a separable way as a 1-D horizontal (row) transform followed by a 1-D vertical (column) transform as shown in Equation 55.

$$W = C_f X C_f^T$$

Equation 55

where the Matrix C_f is given by Equation 56.

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

$$C_f = \begin{bmatrix} 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\ 12 & 10 & 6 & 3 & -3 & -6 & -10 & -12 \\ 8 & 4 & -4 & -8 & -8 & -4 & 4 & 8 \\ 10 & -3 & -12 & -6 & 6 & 12 & 3 & -10 \\ 8 & -8 & -8 & 8 & 8 & -8 & -8 & 8 \\ 6 & -12 & 3 & 10 & -10 & -3 & 12 & -6 \\ 4 & -8 & 8 & -4 & -4 & 8 & -8 & 4 \\ 3 & -6 & 10 & -12 & 12 & -10 & 6 & -3 \end{bmatrix} .1/8$$

Equation 56

Each of the 1-D transforms is computed using 3-stages fast butterfly operations as follows:

Stage 1:

a[0] = x[0] + x[7];

a[1] = x[1] + x[6];

a[2] = x[2] + x[5];

a[3] = x[3] + x[4];

a[5] = x[0] - x[7];

a[6] = x[1] - x[6];

a[7] = x[2] - x[5];

a[8] = x[3] - x[4];

Stage 2:

b[0] = a[0] + a[3];

b[1] = a[1] + a[2];

b[2] = a[0] - a[3];

b[3] = a[1] - a[2];

b[4] = a[5] + a[6] + ((a[4]>>1) + a[4]);

b[5] = a[4] – a[7] – ((a[6]>>1) + a[6]);

b[6] = a[4] + a[7] – ((a[5]>>1) + a[5]);

b[7] = a[5] – a[6] + ((a[7]>>1) + a[7]);

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Stage 3:

```
w[0] = b[0] + b[1];

w[1] = b[2] + (b[3]>>1);

w[2] = b[0] - b[1];

w[3] = (b[2]>>1) - b[3];

w[4] = b[4] + (b[7]>>2);

w[5] = b[5] + (b[6]>>2);

w[6] = b[6] – (b[5]>>2);

w[7] = -b[7] + (b[4]>>2);
```

Hence, the 2-D transform operation can be implemented using signed additions and right-shifts only, avoiding expensive multiplications. The post-scaling and quantization formulas are shown in Equation 57, Equation 58 and Equation 59.

$$qbits = 15 + (QP\ DIV\ 6)$$

Equation 57

$$\left|Z_{ij}\right| = SHR(\left|W_{ij}\right|.MF + f, qbits + 1)$$

Equation 58

$$Sign(Z_{ij}) = Sign(W_{ij})$$

Equation 59

where *QP* is a quantization parameter that enables the encoder to accurately and flexibly control the trade-off between bit rate and quality. It can take any integer value from 0 up to 51. Z_{ij} is an element in the quantized transform coefficients matrix. *MF* is a multiplication factor that depends on *(m = QP mod 6)* and the position *(i, j)* of the element in the matrix as shown in Table 46. *SHR()* is a procedure that right-shifts the result of its first argument a number of bits equal to its second argument. *f* is defined in the reference model software as $2^{qbits}/3$ for Intra blocks and $2^{qbits}/6$ for Inter blocks [15][28].

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 46 — Multiplication Factor (*MF*).

m	$(i, j) \in G_0$	$(i, j) \in G_1$	$(i, j) \in G_2$	$(i, j) \in G_3$	$(i, j) \in G_4$	$(i, j) \in G_5$
0	13107	11428	20972	12222	16777	15481
1	11916	10826	19174	11058	14980	14290
2	10082	8943	15978	9675	12710	11985
3	9362	8228	14913	8931	11984	11295
4	8192	7346	13159	7740	10486	9777
5	7282	6428	11570	6830	9118	8640

*G_0: i = [0, 4], j = [0, 4]

G_1: i = [1, 3, 5, 7], j = [1, 3, 5, 7]

G_2: i = [2, 6], j = [2, 6]

G_3: (i = [0, 4], j = [1, 3, 5, 7]) ∩ (i = [1, 3, 5, 7], j = [0, 4])

G_4: (i = [0, 4], j = [2, 6]) ∩ (i = [2, 6], j = [0, 4])

G_5: (i = [2, 6], j = [1, 3, 5, 7]) ∩ (i = [1, 3, 5, 7], j = [2, 6])

10.10.6 Implementation

The proposed architecture uses 8x8 parallel blocks, QP, a synchronizing clock, and an enabling signal (Input Valid) as inputs. It outputs the quantized transform coefficients and the signal Output Valid.

The architecture is designed to perform pipelined operations, which drastically reduces the required memory resources and accesses, avoids any stall states, and dramatically improves the throughput of the architecture. Figure 174 gives a detailed block diagram of the proposed architecture showing the flow of signals between the main stages of the design.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

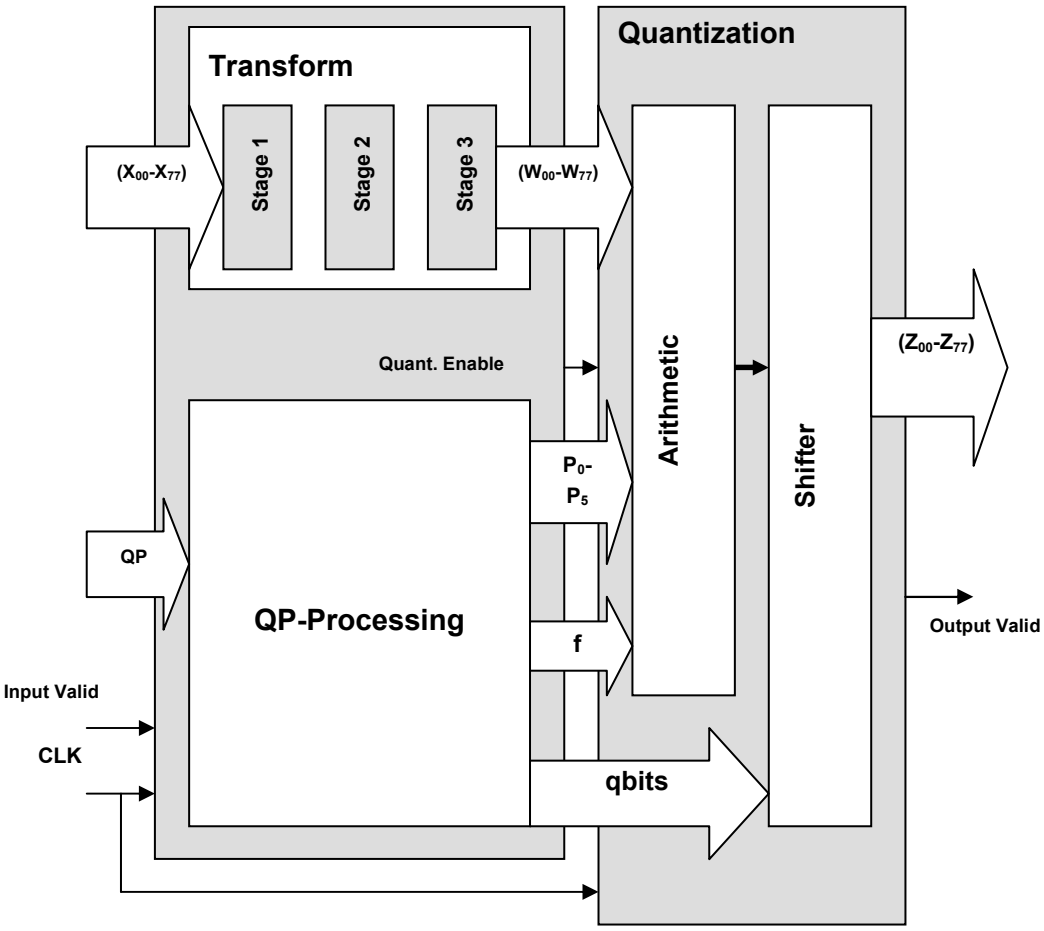


Figure 174 — A detailed block diagram of the hardware architecture.

The architecture is composed of two main stages. The first one contains two blocks; the Transform block, which is composed of the three stages of the fast butterfly operations mentioned in subclause 10.10.5, and the QP-Processing block, which is responsible for calculating the intermediate variables needed for quantization, such as *f*, *qbits*, and (P_0 – P_5), which are the values of the multiplication factors at the six different groups of positions in the matrix as shown in Table 46. Finally, the Quantization process takes place in the second main stage of the design. This is done by performing the addition and multiplication operations in the Arithmetic block, and finally the shifting operations in the Shifter block.

10.10.6.1 Interfaces

N/A.

10.10.6.2 Register File Access

Please refer to subclause 10.10.4.

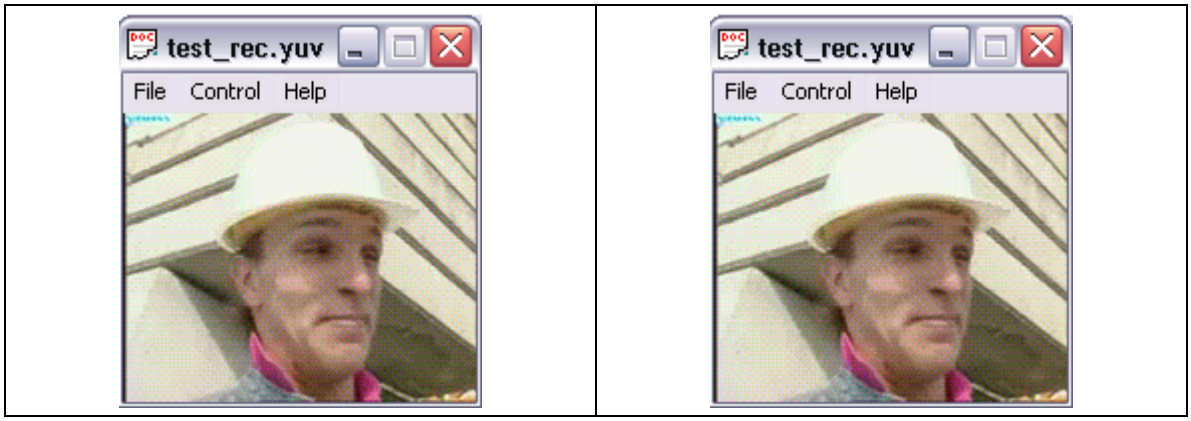
10.10.6.3 Timing Diagrams

N/A.

10.10.7 Results of Performance & Resource Estimation

Behavioural simulation shows that the designed architecture functionally complies with the reference software. The architecture was embedded in JM FRExt 2.2 and its output stream was identical to the output from the original software. Figure 175 gives a comparison between the outputs before and after the embedding the SystemC block.

© ISO/IEC 2009 All rights reserved
295
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale



(b) **(b)**

Figure 175 — (a) Output before embedding the SystemC block (b) Output after embedding the SystemC block.

10.10.8 API calls from reference software

The switching between software and hardware is controlled by flags that can be reset to avoid switching, hence the software flow will execute normally bypassing the SystemC block. An example of a HW-block call from SW is as follows:

```
if(DCT_8x8_HW_ACCELERATOR){
        sc_dct_8x8(img->m7, firstHW_Call);
        firstHW_Call = 0;
}
else{
    sw_dct_8x8(img->m7, m6);
}
```

10.10.9 Conformance Testing

10.10.9.1 Reference software type, version and input data set

Our functional testing was performed on the AVC (MPEG-4 Part 10) reference software (JM FRExt 2.2). The video test sequences are “Miss America” and “Foreman”.

10.10.9.2 API vector conformance

The test vectors used are QCIF test sequences.

10.10.9.3 End to end conformance (conformance of encoded bitstreams or decoded pictures)

The end-to-end encoder conformance test is evaluated via a mixed C and SystemC environment using the JM FRExt 2.2 software reference model. Figure 176 and Figure 177 show that the results obtained before and after using the hardware accelerators are identical.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
Parsing Configfile encoder.cfg.....................................................

-------------------------------JM FREXT ver.2.2-------------------------------

 Input YUV file                 : foreman_part_qcif.yuv

 Output H.264 bitstream         : test.264

 Output YUV file                : test_rec.yuv

 YUV Format                     : YUV 4:2:0

 Frames to be encoded I-P/B     : 2/1

 PicInterlace / MbInterlace     : 0/0

 Transform8x8Mode               : 1

-------------------------------------------------------------------------------

  Frame  Bit/pic WP QP   SnrY    SnrU    SnrV    Time(ms) MET(ms) Frm/Fld   I D

0000(NVB)     176

0000(IDR)   21784 0 28 37.4332 41.3158 43.0858     1301       0     FRM

0002(P)      8816 0 28 36.8903 40.8079 42.3439     2294     321     FRM    18

0001(B)      2656 0 30 36.1340 41.0615 42.8278     4537    1261     FRM     0 1

-------------------------------------------------------------------------------

 Total Frames:  3 (2)

 LeakyBucketRate File does not exist; using rate calculated from avg. rate

 Number Leaky Buckets: 8

    Rmin     Bmin     Fmin

  166275    21784    21784
```

Figure 176 — Summary of results reported by JM FRExt 2.2 before embedding the SystemC block.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

```
Parsing Configfile encoder.cfg.....................................

-------------------------------JM FREXT ver.2.2-------------------------------

Input YUV file                  : foreman_part_qcif.yuv

Output H.264 bitstream          : test.264

Output YUV file                 : test_rec.yuv

YUV Format                      : YUV 4:2:0

Frames to be encoded I-P/B      : 2/1

PicInterlace / MbInterlace      : 0/0

Transform8x8Mode                : 1

-------------------------------------------------------------------------------

Frame  Bit/pic WP QP   SnrY    SnrU    SnrV    Time(ms) MET(ms) Frm/Fld   I D

0000(NVB)     176

0000(IDR)   21784 0 28 37.4332 41.3158 43.0858    26999       0     FRM

0002(P)      8816 0 28 36.8903 40.8079 42.3439    47598     692     FRM    18

0001(B)      2656 0 30 36.1340 41.0615 42.8278    39216    1700     FRM     0 1

-------------------------------------------------------------------------------

Total Frames:  3 (2)

LeakyBucketRate File does not exist; using rate calculated from avg. rate

Number Leaky Buckets: 8

Rmin     Bmin     Fmin

166275    21784    21784
```

Figure 177 — Summary of results reported by JM FRExt 2.2 after embedding the SystemC block.

10.10.10 Limitations

Increased area is the main limitation of the design.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.11 INTEGER APPROXIMATION OF 8X8 DCT TRANSFORMATION AND QUANTIZATION, A HARDWARE IP BLOCK FOR MPEG-4 PART 10 AVC

10.11.1 Abstract

The recently approved digital video standard known as AVC promises to be an excellent video format for use with a large range of applications. Real-time encoding/decoding is a main requirement for adoption of the standard to take place in the consumer marketplace. Transformation and quantization in AVC are relatively less complex than their correspondences in other video standards. Nevertheless, for real-time operation, a speedup is required for such processes. Especially after the recent proposal to use an 8x8 integer approximation of Discrete Cosine Transform (DCT) to give significant compression performance at Standard Definition (SD) and High Definition (HD) resolutions. This subclause is to propose a high-performance hardware implementation of the AVC simplified 8x8 transformation and quantization. The results show that the architecture satisfies the real-time constraints required by different digital video applications.

10.11.2 Module specification

10.11.2.1 MPEG 4 part: ***10***
10.11.2.2 Profile : ***All***
10.11.2.3 Level addressed: ***All***
10.11.2.4 Module Name: ***8x8 DCT-Like (VHDL)***
10.11.2.5 Module latency: ***204.4 ns***
10.11.2.6 Module data throughput: ***An 8x8 parallel quantized transform coefficients matrix/ CC***
10.11.2.7 Max clock frequency: ***68.5 MHz***
10.11.2.8 Resource usage:
10.11.2.8.1 IO Register Bits: ***1219***
10.11.2.8.2 Non IO Register Bits: **16893**
10.11.2.8.3 LUTs: **29018**
10.11.2.8.4 Global Clock Buffers: ***1***
10.11.2.8.5 External memory: ***none***
10.11.2.9 Revision: ***1.00***
10.11.2.10 Authors: ***Ihab Amer, Wael Badawy, and Graham Jullien***
10.11.2.11 Creation Date: ***March 2005***
10.11.2.12 Modification Date: ***March 2005***

10.11.3 Introduction

Unlike current standards, the usual floating-point 8x8 DCT is not the basic transformation in AVC. Instead, a new transformation hierarchy is introduced that can be computed exactly in integer arithmetic. This eliminates any mismatch issues between the encoder and the decoder in the inverse transform [14][21]. In the initial AVC standard, which was completed in May 2003, the transformation is primarily 4x4 in shape, which helps reduce blocking and ringing artifacts.

In July 2004, a new amendment called the Fidelity Range Extensions (FRExt, Amendment I) was added to the AVC standard. This amendment is currently receiving wide attention in the industry. It actually demonstrates further coding efficiency against current video coding standards, potentially by as much as 3:1 for some key applications. The FRExt project produced a suite of some new profiles collectively called High profiles. Beside supporting all features of the prior Main profile, all the High profiles support an adaptive transform-block size and perceptual quantization scaling matrices [29]. In fact, the concept of adaptive transform-block size has proven to be an efficient coding tool within AVC video coding layer design. This led to the proposal of a seamless integration of a new 8x8 integer approximation of DCT (and prediction modes) into the specification with the least possible amount of technical and syntactical changes.

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.11.4 Functional Description

10.11.4.1 Functional description details

This subclause introduces the proposed hardware prototype of the 8×8 forward transform and quantization adopted by FREXt in the MPEG-4 Part 10 standard. It is applied to the parallel 8×8 input pixels blocks of the luma component. A block diagram of the architecture showing its inputs and outputs is given in Figure 178.

10.11.4.2 I/O Diagram

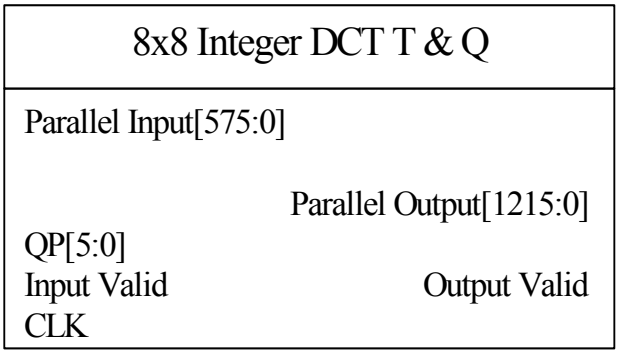


Figure 178 — A block diagram of the proposed hardware architecture.

10.11.4.3 I/O Ports Description

Table 47

Port Name	Port Width	Direction	Description
Parallel Input[127:0]	575	Input	8x8 parallel matrix of pixels
QP[5:0]	6	Input	Quantization Parameter
Input Valid	1	Input	Flag indicating that input is valid
CLK	1	Input	System clock
Parallel Output[223:0]	1215	Output	8x8 parallel quantized transform coefficients matrix
Output Valid	1	Output	Flag indicating that output is valid

10.11.5 Algorithm

An integer approximation of 8x8 DCT was proposed in FREXt to be added to the JVT specification based on the fact that at SD resolutions and above, the use of block sizes smaller than 8x8 is limited. This transform is applied to each block in the luminance component of the input video stream. It allows for bit-exact implementation for all encoders and decoders. In spite of being more complex compared to the 4x4 DCT-like transform that is adopted by the initial AVC specification, the proposed transform gives excellent compression performance when used for high-resolution video streams using a number of operations comparable to the number of operations required for the corresponding four 4x4 blocks using the fast butterfly implementation of the existing 4x4 transform.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The 2-D forward 8x8 transform is computed in a separable way as a 1-D horizontal (row) transform followed by a 1-D vertical (column) transform as shown in Equation 60.

$$W = C_f X C_f^T$$

Equation 60

where the Matrix C_f is given by Equation 61.

$$C_f = \begin{bmatrix} 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\ 12 & 10 & 6 & 3 & -3 & -6 & -10 & -12 \\ 8 & 4 & -4 & -8 & -8 & -4 & 4 & 8 \\ 10 & -3 & -12 & -6 & 6 & 12 & 3 & -10 \\ 8 & -8 & -8 & 8 & 8 & -8 & -8 & 8 \\ 6 & -12 & 3 & 10 & -10 & -3 & 12 & -6 \\ 4 & -8 & 8 & -4 & -4 & 8 & -8 & 4 \\ 3 & -6 & 10 & -12 & 12 & -10 & 6 & -3 \end{bmatrix} .1/8$$

Equation 61

Each of the 1-D transforms is computed using 3-stages fast butterfly operations as follows:

Stage 1:

a[0] = x[0] + x[7];

a[1] = x[1] + x[6];

a[2] = x[2] + x[5];

a[3] = x[3] + x[4];

a[5] = x[0] - x[7];

a[6] = x[1] - x[6];

a[7] = x[2] - x[5];

a[8] = x[3] - x[4];

Stage 2:

b[0] = a[0] + a[3];

b[1] = a[1] + a[2];

b[2] = a[0] - a[3];

b[3] = a[1] - a[2];

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
b[4] = a[5] + a[6] + ((a[4]>>1) + a[4]);

b[5] = a[4] – a[7] – ((a[6]>>1) + a[6]);

b[6] = a[4] + a[7] – ((a[5]>>1) + a[5]);

b[7] = a[5] – a[6] + ((a[7]>>1) + a[7]);
```

Stage 3:

```
w[0] = b[0] + b[1];

w[1] = b[2] + (b[3]>>1);

w[2] = b[0] - b[1];

w[3] = (b[2]>>1) - b[3];

w[4] = b[4] + (b[7]>>2);

w[5] = b[5] + (b[6]>>2);

w[6] = b[6] – (b[5]>>2);

w[7] = -b[7] + (b[4]>>2);
```

Hence, the 2-D transform operation can be implemented using signed additions and right-shifts only, avoiding expensive multiplications. The post-scaling and quantization formulas are shown in Equation 62, Equation 63 and Equation 64 Equations.

$$qbits = 15 + (QP\ DIV\ 6)$$

Equation 62

$$\left|Z_{ij}\right| = SHR(\left|W_{ij}\right|.MF + f, qbits + 1)$$

Equation 63

$$Sign(Z_{ij}) = Sign(W_{ij})$$

Equation 64

where *QP* is a quantization parameter that enables the encoder to accurately and flexibly control the trade-off between bit rate and quality. It can take any integer value from 0 up to 51. Z_{ij} is an element in the quantized transform coefficients matrix. *MF* is a multiplication factor that depends on *(m = QP mod 6)* and the position *(i, j)* of the element in the matrix as shown in Table 48. *SHR()* is a procedure that right-shifts the result of its first argument a number of bits equal to its second argument. *f* is defined in the reference model software as $2^{qbits}/3$ for Intra blocks and $2^{qbits}/6$ for Inter blocks [15][28].

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 48 — Multiplication Factor (*MF*).

m	(i, j) ∈ G_0	(i, j) ∈ G_1	(i, j) ∈ G_2	(i, j) ∈ G_3	(i, j) ∈ G_4	(i, j) ∈ G_5
0	13107	11428	20972	12222	16777	15481
1	11916	10826	19174	11058	14980	14290
2	10082	8943	15978	9675	12710	11985
3	9362	8228	14913	8931	11984	11295
4	8192	7346	13159	7740	10486	9777
5	7282	6428	11570	6830	9118	8640

*G_0: i = [0, 4], j = [0, 4]

G_1: i = [1, 3, 5, 7], j = [1, 3, 5, 7]

G_2: i = [2, 6], j = [2, 6]

G_3: (i = [0, 4], j = [1, 3, 5, 7]) ∩ (i = [1, 3, 5, 7], j = [0, 4])

G_4: (i = [0, 4], j = [2, 6]) ∩ (i = [2, 6], j = [0, 4])

G_5: (i = [2, 6], j = [1, 3, 5, 7]) ∩ (i = [1, 3, 5, 7], j = [2, 6])

10.11.6 Implementation

The proposed architecture uses 8x8 parallel blocks, QP, a synchronizing clock, and an enabling signal (Input Valid) as inputs. It outputs the quantized transform coefficients and the signal Output Valid.

The architecture is designed to perform pipelined operations, which drastically reduces the required memory resources and accesses, avoids any stall states, and dramatically improves the throughput of the architecture. Figure 179 gives a detailed block diagram of the architecture showing the flow of signals between the main stages of the design.

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

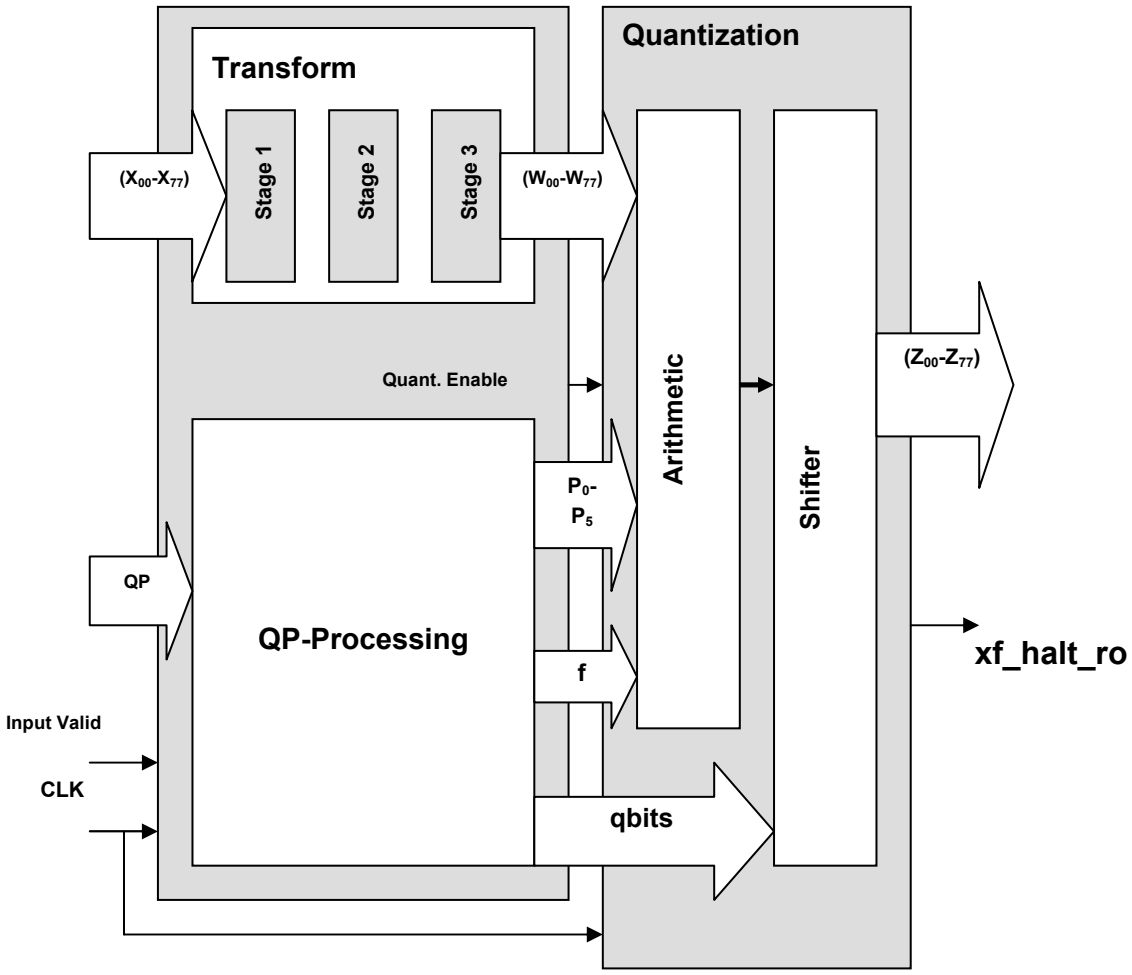


Figure 179 — A detailed block diagram of the hardware architecture.

The architecture is composed of two main stages. The first one contains two blocks; the Transform block, which is composed of the three stages of the fast butterfly operations mentioned in subclause 10.11.5, and the QP-Processing block, which is responsible for calculating the intermediate variables needed for quantization, such as *f*, *qbits*, and ($P_0 – P_5$), which are the values of the multiplication factors at the six different groups of positions in the matrix as shown in Table 48. Finally, the Quantization process takes place in the second main stage of the design. This is done by performing the addition and multiplication operations in the Arithmetic block, and finally the shifting operations in the Shifter block.

10.11.6.1 Interfaces

N/A.

10.11.6.2 Register File Access

Please refer to subclause 10.11.4.

10.11.6.3 Timing Diagrams

N/A.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.11.7 Results of Performance & Resource Estimation

The architecture of the AVC simplified 8x8 transformation is prototyped using VHDL language. It is simulated using the Mentor Graphics© ModelSim 5.4® simulation tool, and synthesized using Synplify Pro 7.1® from Synplicity©. The target technology is the FPGA device XC2V4000 (BF957 package) from the Virtex-II family of Xilinx©.

Table 49 summarizes the performance of the prototyped architecture.

Table 49 — Performance of the architecture.

Critical Path (ns)	CLK Freq. (MHz)	# of i/p Buffers	# of o/p Buffers
14.598	68.5	583	1217
# of I/O Reg. Bits	**#of Reg. Bits not inc. (I/O)**	**Total # of LUT**	**# of clock buffers**
1219	16893	29018	1

A 14.598 *ns* critical path is estimated by the synthesis tool. Since at steady state, the architecture outputs a whole 8x8 encoded block with each clock pulse, therefore the time required to encode a whole SD frame of 704 $\times$ 480 pixels can be calculated as follows:

Time required per CIF frame =

Time required per block $\times$ *Number of blocks per frame*

$$= 14.598\ ns \times \frac{Number\ of\ pixels\ per\ frame}{Number\ of\ pixels\ per\ block}$$

$$= 14.598\ ns \times \frac{(704 \times 480)\ pixels\ per\ frame}{(8 \times 8)\ pixels\ per\ block}$$

$$\simeq 77.1\ \mu s$$

This value is about 216 times less than the 16.67 *ms* time required for continuous motion (assuming a refresh rate of 60 *frames/sec*). Similarly, it can be shown that the time required to encode a whole High Definition Television (HDTV) frame of a 720×1280 pixels resolution, and a 60 *frames/sec* frame rate is 0.21 *ms*, which is about 79 times less than the 16.6 *ms* time required for continuous motion. Hence, the introduced architecture satisfies the real-time constraints for SD, HD, and even higher resolution video formats.

10.11.8 API calls from reference software

N/A.

10.11.9 Conformance Testing

The architecture has not been tested on the platform but its simulation results conforms with the software results.

10.11.9.1 Reference software type, version and input data set

N/A.

10.11.9.2 API vector conformance

N/A.

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.11.9.3 End to end conformance (conformance of encoded bitstreams or decoded pictures)

N/A.

10.11.10 Limitations

Increased area is the main limitation of the design.

10.12 A VHDL CONTEXT-BASED ADAPTIVE VARIABLE LENGTH CODING (CAVLC) IP BLOCK FOR MPEG-4 PART 10 AVC

10.12.1 Abstract

This subclause presents a VHDL model for Context-based Adaptive Variable Length Coding (CAVLC). This scheme is a part of the lossless compression process as described in the MPEG-4 Part 10 standard. It is applied to the quantized transform coefficients of the luminance component during the entropy coding process. The developed architecture is prototyped and simulated using ModelSim 5.4®. It is synthesized using Synplify Pro 7.1®.

10.12.2 Module specification

10.12.2.1 MPEG 4 part:	***10***
10.12.2.2 Profile :	***All***
10.12.2.3 Level addressed:	***All***
10.12.2.4 Module Name:	***CAVLC***
10.12.2.5 Module latency:	***Approx 1 us***
10.12.2.6 Module data throughput:	***1 single-block encoded bitstream/sec***
10.12.2.7 Max clock frequency:	***31.9 MHz***
10.12.2.8 Resource usage:	
10.12.2.8.1 IO Register Bits:	***442***
10.12.2.8.2 Non IO Register Bits:	**15622**
10.12.2.8.3 LUTs:	**84902**
10.12.2.8.4 Global Clock Buffers:	***1***
10.12.2.8.5 External memory:	***none***
10.12.2.9 Revision:	***1.00***
10.12.2.10 Authors:	***Ihab Amer, Wael Badawy, and Graham Jullien***
10.12.2.11 Creation Date:	***July 2004***
10.12.2.12 Modification Date:	***October 2004***

10.12.3 Introduction

The Entropy Coding block in the MPEG-4 Part 10 standard exploits the statistical properties of the data being encoded. It is based on assigning shorter codewords to symbols that occur with higher probabilities, and longer codewords to symbols with less frequent occurrences. Entropy coding represents the lossless part in the AVC encoding process. In combination with previous transformations and quantizations, it can result in significantly increased compression ratio [14][16]. All syntax elements are coded using Exponential Golomb variable length codes with a regular construction. For quantized transform coefficients, either CAVLC, or CABAC is used depending on the entropy coding mode [28][32].

10.12.4 Functional Description

10.12.4.1 Functional description details

This subclause introduces the proposed hardware prototype of CAVLC that is adopted by the MPEG-4 Part 10 standard. The proposed architecture uses 4×4 parallel input blocks. It also takes the number of non-zero coefficients in the left-hand and upper previously coded blocks (*nA* and *nB*) as inputs. It gives the encoded

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

bitstream as an output. A block diagram of the architecture showing its inputs and outputs is given in Figure 180.

10.12.4.2 I/O Diagram

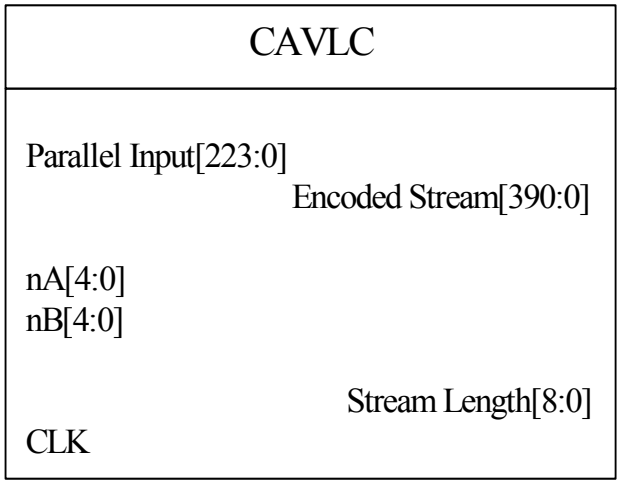


Figure 180 — A block diagram of the hardware architecture.

10.12.4.3 I/O Ports Description

Table 50

Port Name	Port Width	Direction	Description
Parallel Input[223:0]	224	Input	4x4 parallel quantized transform coefficients matrix
nA	1	Input	Number of non-zero coefficients in the left-hand previously coded block
nB	1	Input	Number of non-zero coefficients in the upper previously coded block
CLK	1	Input	System clock
Encoded Stream[390:0]	391	Output	Encoded bitstream
Stream Length	9	Output	Encoded bitstream length

10.12.5 Algorithm

In CAVLC, VLC tables for various elements are switched depending on previously coded elements. This results in an improvement in the coding efficiency compared with schemes that use a single VLC table [14].

In order to exploit the existence of many zeros in a block of quantized transform coefficients, they should be reordered in a way that gives long runs of zeros. Reordering in a zigzag fashion as shown in Figure 181 is used for this purpose.

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

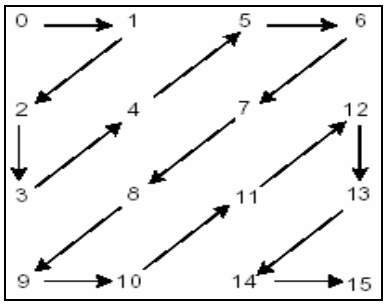


Figure 181 — Zigzag scanning.

CAVLC is designed to take advantage of several characteristics of quantized 4×4 blocks of transform coefficients such as [28][32]:

- Existence of long runs of zeros after zigzag scanning.
- Highest non-zero coefficients after zigzag scanning are often sequences of +/-1. Hence, CAVLC codes the number of high-frequency +/-1 coefficients (trailing ones) in a compact way.
- The number of non-zero coefficients in neighbouring blocks is correlated. Thus, the choice of look-up tables relies on the number of non-zero coefficients in neighbouring blocks.
- The level (magnitude) of non-zero coefficients is usually higher at the start of the reordered array (near the DC coefficient) and lower towards the higher frequencies. Therefore, the choice of VLC look-up tables for the level parameter depends on recently coded level magnitudes.

CAVLC proceeds as follows:

- Encode the number of coefficients and trailing ones (coef_token).
- Encode the sign of each trailing one.
- Encode the levels of the remaining non-zero coefficients.
- Encode the total number of zeros before the last non-zero coefficient.
- Encode each run of zeros.

10.12.6 Implementation

This architecture is designed to perform pipelined operations. Hence, at steady state, it outputs a bitstream representation of a whole block with each clock pulse. The architecture does not contain memory elements. Instead, a redundancy in computational elements shows up. This represents an example of performance-area tradeoff.

A detailed description of the architecture is shown in Figure 182. First, the *zigzag scan* block reorder the 4×4 input block of quantized transform coefficients. It also calculates the average of the number of non-zero coefficients in the left-hand and upper previously coded blocks (*nC*), and outputs a signal *CfStatus*, which is a 16-bit signal with each bit assigned either the value ‘0’ or ‘1’ depending on the value of the corresponding element in the zigzag ordered list whether it is zero or not.

The block *Cftoken* outputs the total number if non-zero coefficients *NumCf*, the number of trailing ones *TrOnes*, and their signs *TrOnesSgn*. The *Z-Work* block calculates the total number of zeros before the last coefficient (*ZTotal*). It also scans the ordered coefficients in reverse order and calculates the number of zeros after each non-zero coefficient as well as the length or the zero-run before it. The *Final stage* is the critical block in the design. A detailed description of the *Final stage* block is given in Figure 183.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

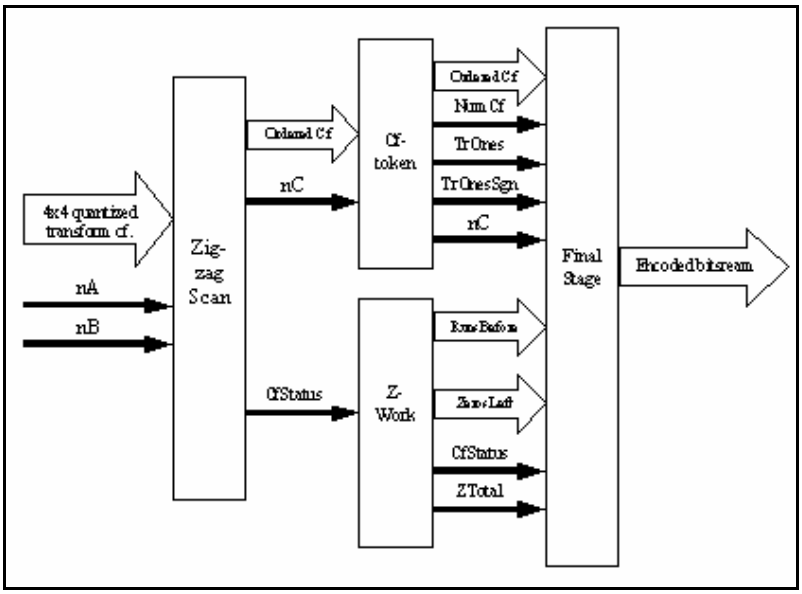


Figure 182 — A detailed block diagram of the hardware architecture.

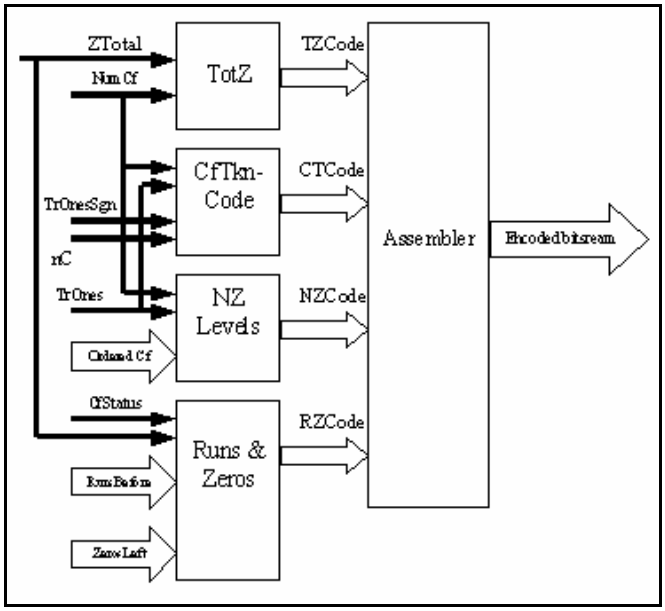


Figure 183 — A block diagram of the final stage.

The block *TotZ* outputs the codeword for *Ztotal* depending on the value of *NumCf*. The block *CfTknCode* outputs the codeword for the coefficient token depending on the values of *NumCf* and *TrOnes*, then it attach *TrOnesSgn* to it. The blocks *NZ Levels* and *Runs & Zeros* calculate the codewords for the non-zero levels and the runs of zeros respectively in the way described in [28]. Finally the *Assembler* block concatenates all the generated codewords in a single encoded bitstream.

10.12.6.1.1 Interfaces

N/A.

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.12.6.1.2 Register File Access

Please refer to subclause 10.12.4.

10.12.6.2 Timing Diagrams

N/A.

10.12.7 Results of Performance & Resource Estimation

The architecture for the AVC CAVLC is prototyped using VHDL language. It is simulated using the Mentor Graphics© ModelSim 5.4® simulation tool, and synthesized using Synplify Pro 7.1®. The target technology is the FPGA device (2V8000bf957) from the Virtex-II family of Xilinx©.

Table 51 summarizes the performance of the prototyped architecture.

Table 51 — Performance of the CALVC architecture.

Critical Path (ns)	CLK Freq. (MHz)	# of i/p Buffers	# of o/p Buffers
31.326	31.9	234	400
# of I/O Reg. Bits	**#of Reg. Bits not inc. (I/O)**	**Total # of LUT**	**# of SRL16**
442	15622	84902	258

The critical path is estimated by the synthesis tool to be 31.326 *ns*. This is equivalent to a maximum operating frequency of 31.9 MHz. Since at steady state, the chip outputs the encoded bitstream for a whole 4×4 block of quantized transform coefficients with each clock pulse, therefore the time required to encode a whole CIF frame (325 $\times$ 288 pixels) can be calculated as follows:

Time required per CIF frame =

Time required per block $\times$ *Number of blocks per frame*

$$= 31.326\ ns \times \frac{Number\ of\ \ pixels\ per\ frame}{Number\ of\ \ pixels\ per\ block}$$

$$= 31.326\ ns \times \frac{(352 \times 288)\ pixels\ per\ frame}{(4 \times 4)\ pixels\ \ per\ block}$$

$$\simeq 0.2\ ms$$

This value is 166.5 times less than the 33.3 *ms* standard time (assuming 29.97 *frames/sec*) required for frame encoding. Similarly, it can be shown that the time required to encode a whole High Definition Television (HDTV) frame of a 720×1280 pixels resolution, and a 60 *frames/sec* frame rate is 1.8 *ms*, which is about 9.2 times less than the 16.6 *ms* standard time. Therefore, the resulting architecture satisfies the real-time constraints required by different digital video applications with a noticeable margin. This leads to the suggestion of integrating other operations on the same encoder chip such as the hierarchal transform and quantization that is adopted by the AVC standard [25][26], taking the input serially, using memory elements, or targeting other applications that use more complicated-higher resolution video formats.

10.12.8 API calls from reference software

N/A.

10.12.9 Conformance Testing

The architecture has not been tested on the platform but its simulation results conforms with the software results.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.12.9.1.1 Reference software type, version and input data set

N/A.

10.12.9.1.2 API vector conformance

N/A.

10.12.9.2 End to end conformance (conformance of encoded bitstreams or decoded pictures)

N/A.

10.12.10 Limitations

Increased area is the main limitation of the design.

10.13 A VERILOG HARDWARE IP BLOCK FOR SA-DCT FOR MPEG-4

10.13.1 Abstract description of the module

This subclause describes a hardware architecture for the MPEG-4 Shape Adaptive Discrete Cosine Transform (SA-DCT) tool. The architecture exploits the fact that video object shape texture data vectors are variable in length by definition to reduce circuit node switching and minimise processing latency. The SA-DCT requires additional processing steps over the conventional block-based 8x8 DCT and this architecture exploits the shape information to minimise the impact of this additional overhead to give the benefits of object-based encoding without a significant increase in computational burdens. The proposed SA-DCT architecture leverages state-of-the-art techniques used to develop hardware for block-based DCT transforms to extend the capability to shape adaptive processing without a corresponding increase in complexity.

10.13.2 Module specification

10.13.2.1 MPEG 4 part: **2 (Video)**
10.13.2.2 Profile: **Advanced Coding Efficiency (ACE)**
10.13.2.3 Level addressed: **L1, L2, L3, L4**
10.13.2.4 Module Name: ***sadct_top***
10.13.2.5 Module latency: **Between the minimum of 72 cycles (when only a single VOP pel in the 8x8 block) and the maximum of 142 cycles when the block is fully opaque.**
10.13.2.6 Module data throughput: **Approx 35.14 Mpixels/sec (with a clock of 78.2 MHz)**
10.13.2.7 Max clock frequency: **Approx 78.2 MHz**
10.13.2.8 Resource usage:
10.13.2.8.1 CLB Slices: **2605**
10.13.2.8.2 Block RAMs: **None**
10.13.2.8.3 Multipliers: **None**
10.13.2.8.4 External memory: **SRAM on WildCard (Capacity of 2MB 133MHz) QCIF 38016 Bytes per texture frame (YUV), 31680 Bytes per alpha frame and sub-sampled alpha, CIF, 152064 Bytes per texture frame (YUV) 126720 Bytes per alpha frame and sub-sampled alpha.**
10.13.2.8.5 Other metrics **Equivalent Gate Count = 39854**
10.13.2.9 Revision: **v4.0**
10.13.2.10 Authors: **Andrew Kinane**
10.13.2.11 Creation Date: **October 2004**
10.13.2.12 Modification Date: **January 2006**

10.13.3 Introduction

This subclause describes a power efficient architecture that can leverage any state-of-the-art implementation of the 1D variable *N*-point Discrete Cosine Transform (DCT) to compute MPEG-4's Shape Adaptive DCT (SA-DCT) tool. The SA-DCT algorithm was originally formulated in response to the MPEG-4 requirement for video object based texture coding [37], and it builds upon the 8x8 2D DCT computation by including extra

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

processing steps that manipulate the video object's shape information. The gain is increased compression efficiency at the cost of additional computation. This work focuses on absorbing these additional SA-DCT specific processing stages in an efficient manner in terms of power consumption, computation latency and silicon area. In this way, the gains associated with the SA-DCT are achieved with minimal impact on hardware resources. The SA-DCT is one of the most computationally demanding blocks in an MPEG-4 video codec, therefore energy-efficient implementations are important – especially on battery powered wireless platforms. Power consumption issues require resolution for mobile MPEG-4 hardware solutions to become viable. A more in depth discussion on the SA-DCT algorithm and the parameters influencing power dissipation in digital circuits may be found in [37] and [38] respectively.

The main principles behind the design of this architecture are as follows:

- The SA-DCT algorithm has been analysed and the computation steps have been re-formulated and merged on the premise that if there are less operations to be carried out, there will be less switching and hence less energy dissipation.

- Since the computational load of the SA-DCT algorithm depends entirely on the VOP shape, the circuit switching activity and processing latency is proportional to the amount of VOP pels in a particular 8x8 block.

- In general, registers are only switched if absolutely necessary depending on a particular computation using clock gating techniques.

- The processing latency of the module is minimised without excessive use of parallelism to permit a lower operating frequency and voltage to lower power dissipation.

- The coefficient computation datapath has been serialised to reduce area. The design computes coefficients serially from k = N-1 down to k = 0. The same datapath is shared for both vertical and horizontal data processing.

10.13.4 Functional Description

10.13.4.1 Functional description details

The *sadct_top* module has been implemented with sub-modules that compute various stages of the SA-DCT algorithm. A system block diagram is shown in Figure 184.

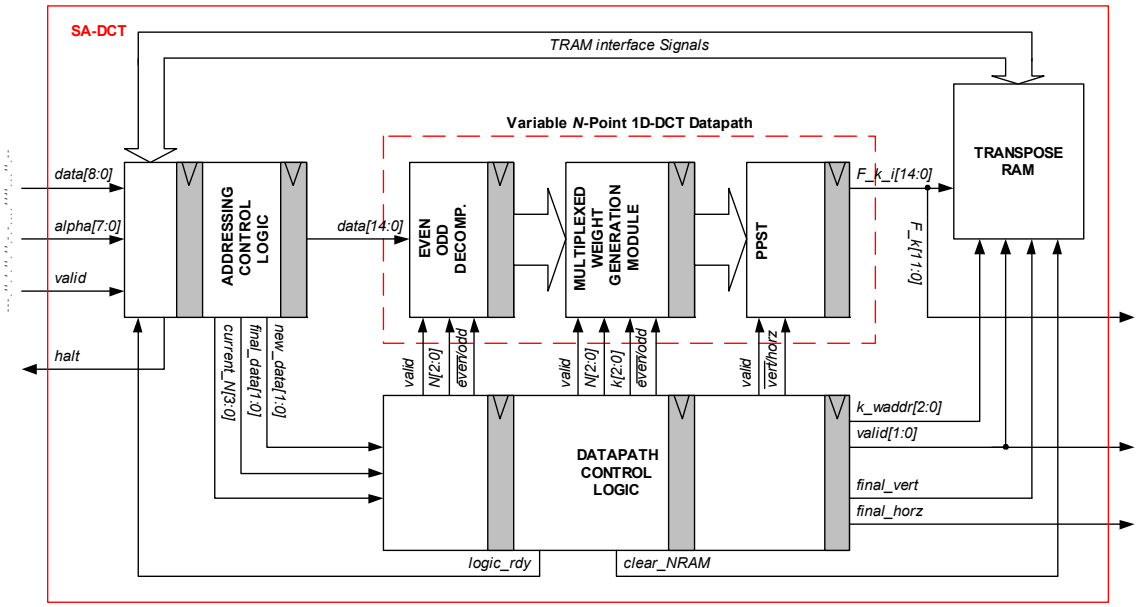


Figure 184 — SA-DCT System Architecture.

Top-level SA-DCT architecture is shown in Figure 184, comprising of the TRAM and datapath with their associated control logic. For all modules local clock gating is employed based on the computation being carried out to avoid wasted power. The addressing control logic (ACL) reads the pixel and shape information serially into a set of interleaved pixel vector buffers that store the pixel data and evaluate *N* with minimal switching avoiding explicit vertical packing. When loaded, a vector is then passed to the variable *N*-point 1D DCT module, which computes all *N* coefficients serially starting with *F[N-1]* down to *F[0]*. This is achieved

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

using even/odd decomposition (EOD), followed by adder-based distributed arithmetic using a multiplexed weight generation module (MWGM) and a partial product summation tree (PPST). The TRAM has a 64 word capacity, and when storing data here the index *k* is manipulated to store the value at address *8*k + N_horz[k]*. Then *N_horz[k]* is incremented by 1. In this way when an entire block has been vertically transformed, the TRAM has the resultant data stored in a horizontally packed manner with the horizontal *N* values ready immediately without shifting. The ACL addresses the TRAM to read the appropriate row data and the datapath is re-used to compute the final SA-DCT coefficients that are routed to the module output.

A serial coefficient computation scheme has been chosen because it facilitates simpler shape information parsing and hence simpler data interpretation and addressing. Also, the area of the datapath is better compared to a parallel scheme although the processing latency increases slightly. The reason for only a slight increase is the algorithmic subsuming of the SA-DCT packing stages.

A more detailed description of the architecture and behavioural steps may be found in [39].

10.13.4.2 I/O Diagram

The top-level I/O signals of the *sadct_top* module are summarised in Figure 185.

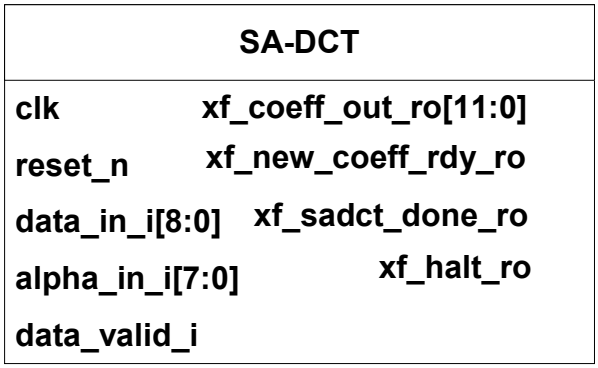


Figure 185 — Top Level I/O Ports.

10.13.4.3 I/O Ports Description

Table 52

Port Name	Port Width	Direction	Description
clk	1	Input	System clock
reset_n	1	Input	Asynchronous active-low reset
data_in_i[8:0]	9	Input	Serial port for reading VOP block texture data (pixels in INTRA mode and pixel differences in INTER mode)
alpha_in_i[7:0]	8	Input	Serial port for reading VOP block alpha data
data_valid_i	1	Input	Active-high signal when asserted indicates that valid VOP texture data and co-located alpha data is present on the input data ports
xf_coeff_out_ro[11:0]	12	Output	SA-DCT coefficient output port
xf_new_coeff_rdy_ro	1	Output	Active-high signal when asserted indicates that a valid SA-DCT coefficient is present on the output data port

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

xf_sadct_done_ro	1	Output	Active-high pulse signal that indicates the final VOP coefficient for a particular block is on the output port if xf_new_coeff_rdy is also asserted. If asserted and xf_new_coeff_rdy is de-asserted a transparent VOP block has been detected
xf_halt_ro	1	Output	Halt external data routing to core when control logic is busy

10.13.4.3.1 Parameters (generic)

Table 53

Parameter Name	Type	Range	Description
T_TRANSPOSE	Integer	4	Bit width of fractional part of intermediate SA-DCT coefficients (after vertical transformation)

10.13.4.3.2 Parameters (constants)

N/A.

10.13.5 Algorithm

The algorithm implemented by this module is the SA-DCT required for object-based texture encoding of video objects for MPEG-4 core profile and above. The SA-DCT is less regular compared to the 8x8 block-based DCT since its processing decisions are entirely dependent on the shape information associated with each individual block. The 8x8 DCT requires 16 1D 8-point DCT computations if implemented using the row-column approach. Each 1D transformation has a fixed length of 8, with fixed basis functions. This is amenable to hardware implementation since the data path is fixed and all parameters are constant. The SA-DCT requires up to 16 1D N-point DCT computations where N $\in$ {2,3…8} (N $\in$ {0,1} are trivial cases). In general N can vary across the possible 16 computations depending on the shape. With the SA-DCT the basis functions vary with N, complicating hardware implementation.

A sample SA-DCT computation showing each high-level processing stage is shown in Figure 186, of which there are 6 in total.

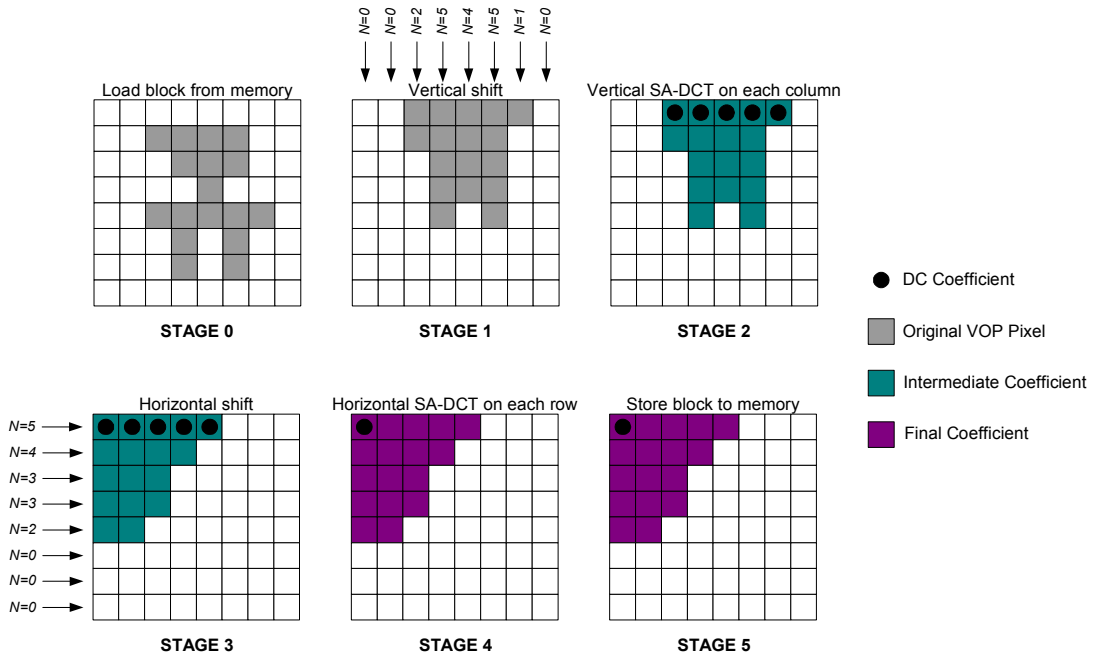


Figure 186 — Example showing SA-DCT computation stages.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2009 – All rights reserved
Not for Resale

Additional non-trivial shifting and packing stages are required for the SA-DCT that are unnecessary for the conventional 8x8 DCT. In summary, the SA-DCT processing stages are:

- Stage 0 – Load input block data from memory
- Stage 1 – Vertically shift VOP pels
- Stage 2 – Vertical *N*-point 1D DCT on each column
- Stage 3 – Horizontally shift intermediate vertical coefficients
- Stage 4 – Horizontal *N*-point 1D DCT on each row of intermediate coefficients
- Stage 5 – Store final coefficient block data to external memory

The block-based 8x8 DCT does not require stages 1 and 3. In addition, stages 0 and 5 are somewhat trivial for an 8x8 DCT since the amount of data being loaded and stored is fixed. With the SA-DCT, this amount varies depending on the alpha mask so there is scope for adapting the number of processing steps based on the shape information to achieve minimum processing latency.

10.13.6 Implementation

The architecture *sadct_top* has been implemented using Verilog HDL in a structural style with RTL sub-modules as summarised in Figure 184. Full details of the internal architectural structure are given in [39]. The module has been integrated with an adapted version of the multiple IP-core hardware accelerated software system framework developed by the University of Calgary. The entire system along with host software calls has been implemented on a Windows 2000 laptop with the Annapolis PCMCIA FPGA (Xilinx Virtex-II XC2V 3000-4) prototyping platform installed. The main alterations were to the hardware module controller (to comply with the interface shown in Figure 185). Also, since the alpha information is required for SA-DCT processing, the host software was altered to store the alpha information along with the texture information in the SRAM on the prototyping platform.

The design flow followed is summarised in Figure 187. The SA-DCT core was coded in Verilog at RTL level and simulated with a testbench using ModelSim SE v6.0a. The original design was in SystemC but due to the discontinuation of the Synopsys SystemC Compiler tool, direct Verilog was adopted instead.

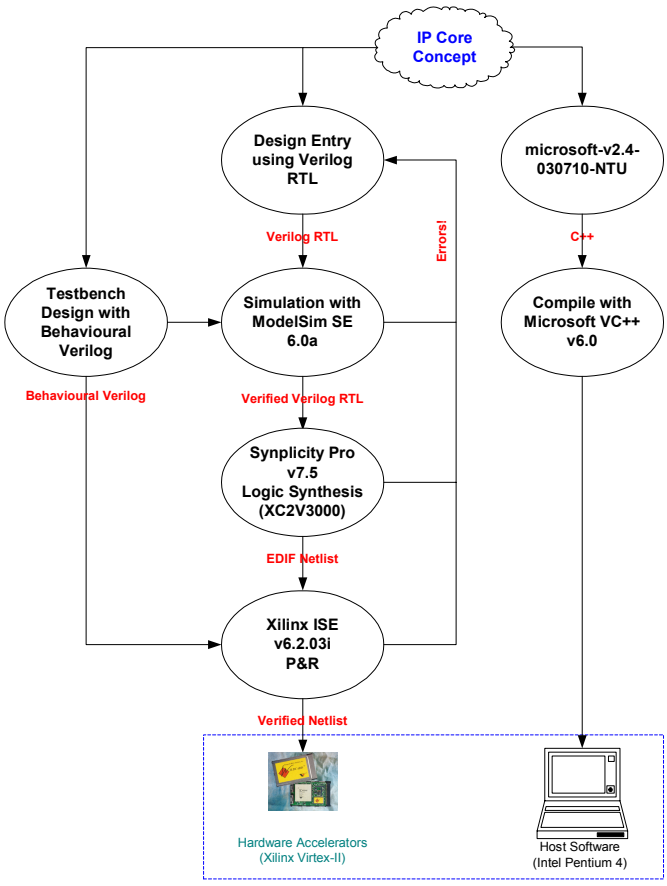


Figure 187 — Design flow from concept to implementation.

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Once verified, the Verilog RTL of the SA-DCT core was integrated with an adapted version of the VHDL multiple IP-core integration framework developed by the University of Calgary. The only HDL module that required major modification was the hardware module controller, which interfaces the SA-DCT core with the rest of the integration framework. The entire HDL system was synthesised with Synplicity Pro (v7.5) targeting the WildCard Xilinx Virtex-II (XC2V3000) FPGA. Xilinx ISE (v6.203i) was used to place and route the netlist created by Synplicity Pro. The host software used is the Microsoft MPEG-4 Part 7 Optimised Video Reference Software (version microsoft-v2.4-030710-NTU) as hosted by the National Chiao-Tung University, Taiwan [40].

10.13.6.1 Interfaces

The input ports apart form the clock and reset signal are driven by the hardware module controller to serially read VOP block alpha and texture data in a column wise raster manner from the SRAM (via the memory source block and the hardware module controller). When *data_valid_i* is asserted (active-high) by the hardware module controller, this indicates that valid pixel information is present on *data_in_i[7:0]* and its co-located alpha value is present on *alpha_in_i[7:0]*. When *xf_new_coeff_rdy_ro* is asserted (active-high) the module is indicating to the hardware module controller that a new SA-DCT coefficient is present on the *xf_coeff_out_ro[11:0]* output port. The hardware module controller writes coefficients back to the SRAM via the memory destination module. The signal *xf_sadct_done_ro* is used to indicate to the hardware module controller that the final VOP coefficient for a particular block is present on *xf_coeff_out_ro[11:0]* (if *xf_new_coeff_rdy_ro* is asserted in the same cycle) or if a fully transparent block has been detected (if *xf_new_coeff_rdy_ro* is de-asserted in the same cycle). This extra handshaking signal is necessary since the amount of VOP coefficients in a particular block can vary depending on the shape ($64 \geq M \geq 0$) and it is efficient to exploit this fact for processing latency gains.

10.13.6.2 Register File Access

Four 32-bit master socket registers are programmed directly by the host software to configure parameters for the SA-DCT core. These registers are used to configure the hardware module controller.

Table 54

Register Name	Range	Description
dControl[0][31:0]	[0][9:0]	Frame width
	[0][19:10]	Frame Height
	[0][23:20]	Number of frames that are read/written at a time
	[0][31:0]	Undefined
dControl[1][31:0]	[1][20:0]	SRAM read start address
	[1][31:21]	Undefined
dControl[2][31:0]	[2][20:0]	SRAM write start address
	[2][31:21]	Undefined
dControl[3][31:0]	[3][0]	Configures SA-DCT hardware module controller addressing mode. ‘1’ => 4:2:0 YUV alpha mode. ‘0’ => Debug Mode.
	[3][31:1]	Undefined

The host software programs these registers after the frame data has been written to the SRAM. They are written by using a WildCard API function *WC_PeRegWrite*. This function writes each of the four 32-bit values into the hardware register file configuration registers at a specific offset according to the specific hardware accelerator being strobed. The abridged code listing in Figure 190 shows how the API function is called.

10.13.6.3 Timing Diagrams

Figure 188 shows a simplified I/O timing diagram for the SA-DCT architecture, and the worst case processing latency is 142 clock cycles. This worst case occurs if the alpha block is fully opaque.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2009 – All rights reserved
Not for Resale

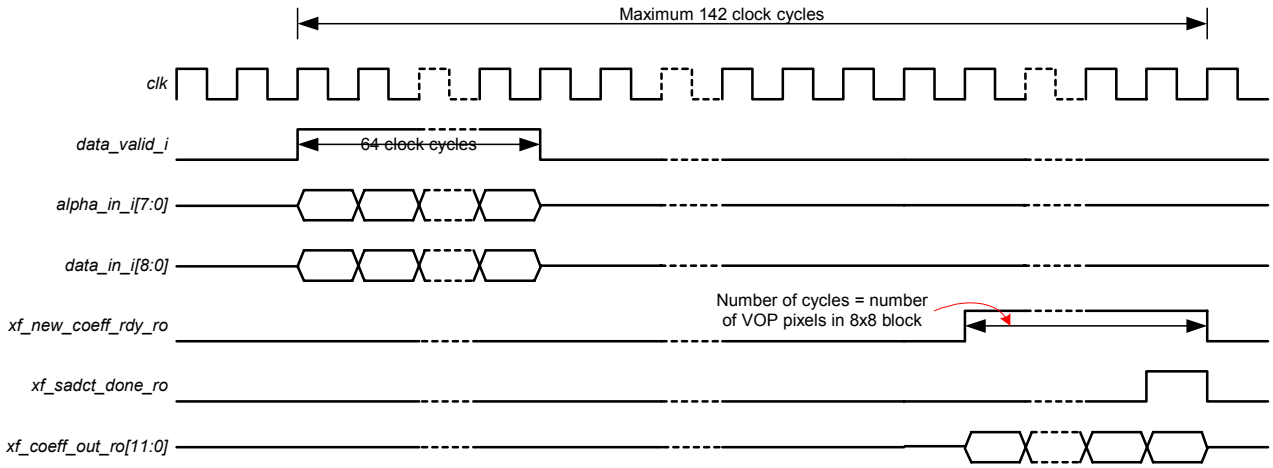


Figure 188 — Sample I/O Ports Timing Diagram.

10.13.7 Results of Performance & Resource Estimation

The module has been integrated with the University of Calgary's integration framework and implemented on the Annapolis WildCard FPGA prototyping platform with associated host calling software. The IP core has been implemented using Verilog RTL and verified with ModelSim SE v6.0a. The Verilog RTL was then synthesised using Synplicity Pro (version 7.5) followed by place and route using Xilinx ISE (version 6.2.03i). The synthesis and place & route scripts were adapted from those proposed by the University of Calgary. To obtain resource usage information for the IP core itself a synthesis run was carried out with the IP core only without the surrounding integration framework and pin assignments (since the IP core is not connected to any FPGA pins directly). An abridged version of the mapping report is given in the following code listing:

```
Release 6.3.03i Map G.38
Xilinx Mapping Report File for Design 'sadct_top'

Design Information
------------------
Command Line   : C:/Xilinx/bin/nt/map.exe -intstyle ise -p XC2V3000-FG676-4 -cm
area -pr b -k 4 -c 100 -tx off -o sadct_top_map.ncd sadct_top.ngd sadct_top.pcf
Target Device  : x2v3000
Target Package : fg676
Target Speed   : -4
Mapper Version : virtex2 -- $Revision: 1.16.8.2 $
Mapped Date    : Mon Nov 28 12:54:23 2005

Design Summary
--------------
Number of errors:      0
Number of warnings:    0
Logic Utilization:
  Number of Slice Flip Flops:       1,745 out of  28,672    6%
  Number of 4 input LUTs:           3,589 out of  28,672   12%
Logic Distribution:
  Number of occupied Slices:        2,605 out of  14,336   18%
  Number of Slices containing only related logic:   2,605 out of   2,605  100%
  Number of Slices containing unrelated logic:          0 out of   2,605    0%
       *See NOTES below for an explanation of the effects of unrelated logic
Total Number 4 input LUTs:          3,607 out of  28,672   12%
  Number used as logic:             3,589
  Number used as a route-thru:         18

  Number of bonded IOBs:               35 out of     484    7%
  Number of GCLKs:                      1 out of      16    6%

Total equivalent gate count for design:  39,854
Additional JTAG gate count for IOBs:  1,680
Peak Memory Usage:  142 MB
```

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
Section 13 - Additional Device Resource Counts
-----------------------------------------------
Number of JTAG Gates for IOBs = 35
Number of Equivalent Gates for Design = 39,854
Number of RPM Macros = 0
Number of Hard Macros = 0
CAPTUREs = 0
BSCANs = 0
STARTUPs = 0
PCILOGICs = 0
DCMs = 0
GCLKs = 1
ICAPs = 0
18X18 Multipliers = 0
Block RAMs = 0
TBUFs = 0
Total Registers (Flops & Latches in Slices & IOBs) not driven by LUTs = 1318
IOB Dual-Rate Flops not driven by LUTs = 0
IOB Dual-Rate Flops = 0
IOB Slave Pads = 0
IOB Master Pads = 0
IOB Latches not driven by LUTs = 0
IOB Latches = 0
IOB Flip Flops not driven by LUTs = 0
IOB Flip Flops = 0
Unbonded IOBs = 0
Bonded IOBs = 35
Total Shift Registers = 0
Static Shift Registers = 0
Dynamic Shift Registers = 0
16x1 ROMs = 0
16x1 RAMs = 0
32x1 RAMs = 0
Dual Port RAMs = 0
MUXFs = 855
MULT_ANDs = 8
4 input LUTs used as Route-Thrus = 18
4 input LUTs = 3589
Slice Latches not driven by LUTs = 0
Slice Latches = 0
Slice Flip Flops not driven by LUTs = 1318
Slice Flip Flops = 1745
Slices = 2605
Number of LUT signals with 4 loads = 30
Number of LUT signals with 3 loads = 125
Number of LUT signals with 2 loads = 527
Number of LUT signals with 1 load = 2604
NGM Average fanout of LUT = 2.55
NGM Maximum fanout of LUT = 120
NGM Average fanin for LUT = 3.2073
Number of LUT symbols = 3589
Number of IPAD symbols = 20
Number of IBUF symbols = 20
```

Figure 189

At CIF resolution with a frame rate of 30 fps requires 71280 blocks to be processed per second. This implies that the SA-DCT should be capable of processing a single 8x8 block in approximately 14μs. Given that the worst-case number of cycles for the IP core to process a block is 142 cycles the IP core must run at approximately 11MHz at worst to maintain real-time constraints. The place and route report generated by ISE indicates a theoretical operating frequency of approximately 78MHz so the IP core should be able to handle real time processing of CIF sequences quite comfortably. Operating at 78 MHz the module is capable of processing at least 35.14Mpixels/s.

10.13.8 API calls from reference software

The hardware acceleration framework with the integrated SA-DCT IP core has been integrated with Microsoft MPEG-4 Part 7 Optimised Video Reference Software (version microsoft-v2.4-030710-NTU). As can be seen from the following code listing, the hardware accelerator for the SA-DCT is called in file *sadct.cpp* in the class

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

CFwdSADCT. Based on a pre-processor directive, the function *DCU_SADCT_HWA* is called and this looks after initiating the appropriate protocols with the SA-DCT hardware accelerator on the WildCard-II FPGA.

```
Void CFwdSADCT::apply(const Int* rgiSrc, Int nColSrc, Int* rgiDst, Int nColDst, const
PixelC* rgchMask, Int nColMask, Int *lx)
{
  if (rgchMask)
  {
    prepareMask(rgchMask, nColMask);
    prepareInputBlock(m_in, rgiSrc, nColSrc);

        // Schueuer HHI: added for fast_sadct
        #ifdef _FAST_SADCT_
        fast_transform(m_out, lx, m_in, m_mask, m_N, m_N);
        #elif _DCU_SADCT_HWA_
              DCU_SA_DCT_HWA(m_out, lx, m_in, m_mask, m_N, m_N);
        #else
        transform(m_out, lx, m_in, m_mask, m_N, m_N);
        #endif

        copyBack(rgiDst, nColDst, m_out, lx);
  }
  else
        CBlockDCT::apply(rgiSrc, nColSrc, rgiDst, nColDst, NULL, 0, NULL);
}
```

Figure 190

10.13.9 Conformance Testing

10.13.9.1 Reference software type, version and input data set

The hardware acceleration framework with the integrated SA-DCT IP core has been integrated with Microsoft MPEG-4 Part 7 Optimised Video Reference Software (version microsoft-v2.4-030710-NTU). The parameter file used to configure the encoder is shown below (*Source.FilePrefix* changes depending on the test sequence name). Figure 191 shows an uncompressed CIF resolution frame (from the akiyo sequence), the software-only reconstructed VOP frame and the SA-DCT hardware accelerated reconstructed VOP frame. The conformance test verifies that the software-only bitstream and the hardware accelerated bitstream are identically bit accurate.



Figure 191 — Example frames for conformance testing of the akiyo sequence.

```
Version = 904                       // parameter file version

// When VTC is enabled, the VTC parameter file is used instead of this one.
VTC.Enable = 0
VTC.Filename = ""
VersionID[0] = 2              // object stream version number (1 or 2)

Source.Width = 352
Source.Height = 288
Source.FirstFrame = 0
Source.LastFrame = 9
Source.ObjectIndex.First = 255
Source.ObjectIndex.Last = 255
Source.FilePrefix = "akiyo_cif"
Source.Directory = "."
```

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
Source.BitsPerPel = 8
Source.Format [0] = "420"            // One of "444", "422", "420"
Source.FrameRate [0] = 10
Source.SamplingRate [0] = 1

Output.Directory.Bitstream = ".\cmp"
Output.Directory.DecodedFrames = ".\rec"

Not8Bit.Enable = 0
Not8Bit.QuantPrecision = 5

RateControl.Type [0] = "None"        // One of "None", "MP4", "TM5"
RateControl.BitsPerSecond [0] = 50000

Scalability [0] = "None"         // One of "None", "Temporal", "Spatial"
Scalability.Temporal.PredictionType [0] = 0       // Range 0 to 4
Scalability.Temporal.EnhancementType [0] = "Full" // One of "Full", "PartC", "PartNC"
Scalability.Spatial.EnhancementType [0] = "PartC" // One of "Full", "PartC", "PartNC"
Scalability.Spatial.PredictionType [0] = "PBB"    // One of "PPP", "PBB"
Scalability.Spatial.Width [0] = 352
Scalability.Spatial.Height [0] = 288
Scalability.Spatial.HorizFactor.N [0] = 2 // upsampling factor N/M
Scalability.Spatial.HorizFactor.M [0] = 1
Scalability.Spatial.VertFactor.N [0] = 2  // upsampling factor N/M
Scalability.Spatial.VertFactor.M [0] = 1
Scalability.Spatial.UseRefShape.Enable [0] = 0
Scalability.Spatial.UseRefTexture.Enable [0] = 0
Scalability.Spatial.Shape.HorizFactor.N [0] = 2   // upsampling factor N/M
Scalability.Spatial.Shape.HorizFactor.M [0] = 1
Scalability.Spatial.Shape.VertFactor.N [0] = 2    // upsampling factor N/M
Scalability.Spatial.Shape.VertFactor.M [0] = 1

Quant.Type [0] = "H263"              // One of "H263", "MPEG"

GOV.Enable [0] = 0
GOV.Period [0] = 0               // Number of VOPs between GOV headers

Alpha.Type [0] = "Binary"            // One of "None", "Binary", "Gray", "ShapeOnly"
Alpha.MAC.Enable [0] = 0
Alpha.ShapeExtension [0] = 0         // MAC type code
Alpha.Binary.RoundingThreshold [0] = 0
Alpha.Binary.SizeConversion.Enable [0] = 0
Alpha.QuantStep.IVOP [0] = 16
Alpha.QuantStep.PVOP [0] = 16
Alpha.QuantStep.BVOP [0] = 16
Alpha.QuantDecouple.Enable [0] = 0
Alpha.QuantMatrix.Intra.Enable [0] = 0
Alpha.QuantMatrix.Intra [0] = {}     // { insert 64 comma-separated values }
Alpha.QuantMatrix.Inter.Enable [0] = 0
Alpha.QuantMatrix.Inter [0] = {}     // { insert 64 comma-separated values }

Texture.IntraDCThreshold [0] = 0     // See note at top of file
Texture.QuantStep.IVOP [0] = 16
Texture.QuantStep.PVOP [0] = 16
Texture.QuantStep.BVOP [0] = 16
Texture.QuantMatrix.Intra.Enable [0] = 0
Texture.QuantMatrix.Intra [0] = {}       // { insert 64 comma-separated values }
Texture.QuantMatrix.Inter.Enable [0] = 0
Texture.QuantMatrix.Inter [0] = {}       // { insert 64 comma-separated values }
Texture.SADCT.Enable [0] = 1

Motion.RoundingControl.Enable [0] = 1
Motion.RoundingControl.StartValue [0] = 0
Motion.PBetweenICount [0] = -1
Motion.BBetweenPCount [0] = 2
Motion.SearchRange [0] = 16
Motion.SearchRange.DirectMode [0] = 2    // half-pel units
Motion.AdvancedPrediction.Enable [0] = 0
Motion.SkippedMB.Enable [0] = 1
Motion.UseSourceForME.Enable [0] = 1
Motion.DeblockingFilter.Enable [0] = 0
Motion.Interlaced.Enable [0] = 0
Motion.Interlaced.TopFieldFirst.Enable [0] = 0
Motion.Interlaced.AlternativeScan.Enable [0] = 0
Motion.ReadWriteMVs [0] = "Off"          // One of "Off", "Read", "Write"
Motion.ReadWriteMVs.Filename [0] = "MyMVFile.dat"
```


--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
Motion.QuarterSample.Enable [0] = 0

Trace.CreateFile.Enable [0] = 1
Trace.DetailedDump.Enable [0] = 1

Sprite.Type [0] = "None"          // One of "None", "Static", "GMC"
Sprite.WarpAccuracy [0] = "1/2"           // One of "1/2", "1/4", "1/8", "1/16"
Sprite.Directory = "\\swinder1\sprite\brea\spt"
Sprite.Points [0] = 0                 // 0 to 4, or 0 to 3 for GMC
Sprite.Points.Directory = "\\swinder1\sprite\brea\pnt"
Sprite.Mode [0] = "Basic"             // One of "Basic", "LowLatency", "PieceObject", "PieceUpdate"

ErrorResil.RVLC.Enable [0] = 0
ErrorResil.DataPartition.Enable [0] = 0
ErrorResil.VideoPacket.Enable [0] = 0
ErrorResil.VideoPacket.Length [0] = 0
ErrorResil.AlphaRefreshRate [0] = 1

Newpred.Enable [0] = 0
Newpred.SegmentType [0] = "VideoPacket"       // One of "VideoPacket", "VOP"
Newpred.Filename [0] = "example.ref"
Newpred.SliceList [0] = "0"

RRVMode.Enable [0] = 0                // Reduced resolution VOP mode
RRVMode.Cycle [0] = 0

Complexity.Enable [0] = 1             // Global enable flag
Complexity.EstimationMethod [0] = 1       // 0 or 1
Complexity.Opaque.Enable [0] = 1
Complexity.Transparent.Enable [0] = 1
Complexity.IntraCAE.Enable [0] = 1
Complexity.InterCAE.Enable [0] = 1
Complexity.NoUpdate.Enable [0] = 1
Complexity.UpSampling.Enable [0] = 1
Complexity.IntraBlocks.Enable [0] = 1
Complexity.InterBlocks.Enable [0] = 1
Complexity.Inter4VBlocks.Enable [0] = 1
Complexity.NotCodedBlocks.Enable [0] = 1
Complexity.DCTCoefs.Enable [0] = 1
Complexity.DCTLines.Enable [0] = 1
Complexity.VLCSymbols.Enable [0] = 1
Complexity.VLCBits.Enable [0] = 1
Complexity.APM.Enable [0] = 1
Complexity.NPM.Enable [0] = 1
Complexity.InterpMCQ.Enable [0] = 1
Complexity.ForwBackMCQ.Enable [0] = 1
Complexity.HalfPel2.Enable [0] = 1
Complexity.HalfPel4.Enable [0] = 1
Complexity.SADCT.Enable [0] = 1
Complexity.QuarterPel.Enable [0] = 1

VOLControl.Enable [0] = 0
VOLControl.ChromaFormat [0] = 0
VOLControl.LowDelay [0] = 0
VOLControl.VBVParams.Enable [0] = 0
VOLControl.Bitrate [0] = 0            // 30 bits
VOLControl.VBVBuffer.Size [0] = 0     // 18 bits
VOLControl.VBVBuffer.Occupancy [0] = 0        // 26 bits
```

Figure 192

10.13.9.2 API vector conformance

At API level the test vectors used are the CIF and QCIF test sequences as defined by the MPEG-4 Video Verification Model [41].

10.13.9.3 End to end conformance (conformance of encoded bitstreams or decoded pictures)

End to end conformance has been completed and it has been verified that the bitstreams produced by the encoder with and without SA-DCT hardware acceleration are identical.

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.13.10 Limitations

The only limitation associated with this module is that block data must be fed serially to it in a vertical raster manner. If bandwidth was sufficient and parallel data was available this would only require a re-work of the input buffer architecture.

10.14 A VERILOG HARDWARE IP BLOCK FOR SA-IDCT FOR MPEG-4

10.14.1 Abstract Description of the module

This subclause describes a hardware architecture for the MPEG-4 Shape Adaptive Inverse Discrete Cosine Transform (SA-IDCT) tool. The architecture exploits the fact that video object shape texture data vectors are variable in length by definition to reduce circuit node switching and processing latency. The SA-IDCT requires additional processing steps over the conventional block-based 8x8 IDCT and this architecture exploits the shape information to minimise the impact of this additional overhead to give the benefits of object-based encoding without a significant increase in computational burdens. The proposed SA-IDCT architecture leverages state-of-the-art techniques used to develop hardware for block-based IDCT transforms to extend the capability to shape adaptive processing without a corresponding increase in complexity.

10.14.2 Module specification

10.14.2.1 MPEG 4 part:
2 (Video)
10.14.2.2 Profile :
Advanced Coding Efficiency (ACE)
10.14.2.3 Level addressed:
L1, L2, L3, L4
10.14.2.4 Module Name:
isadct_top
10.14.2.5 Module latency:
Between the minimum of 125 cycles (when only a single VOP pel in the 8x8 block) and the maximum of 188 cycles when the block is fully opaque.

10.14.2.6 Module data throughput:
Approx 25.63 Mpixel/s (with clock of 75.3 MHz)
10.14.2.7 Max clock frequency:
Approx 75.3 MHz
10.14.2.8 Resource usage:
10.14.2.8.1 CLB Slices:
2763
10.14.2.8.2 Block RAMs:
None
10.14.2.8.3 Multipliers:
None
10.14.2.8.4 External memory:
SRAM on WildCard-II (Capacity of 2MB 133MHz) QCIF: 76032 Bytes per texture frame (YUV coefficients) 31680 Bytes per alpha frame and sub-sampled alpha. CIF: 304128 Bytes per texture frame (YUV coefficients). 126720 Bytes per alpha frame and sub-sampled alpha
10.14.2.8.5 Other metrics:
Equivalent Gate Count = 42787
10.14.2.9 Revision:
v1.0
10.14.2.10 Authors:
Andrew Kinane
10.14.2.11 Creation Date:
January 2006
10.14.2.12 Modification Date:
January 2006

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.14.3 Introduction

This work proposes an SA-IDCT architecture that offers a good trade-off between area, speed and power consumption. The SA-DCT/IDCT algorithms were originally formulated in response to the MPEG-4 requirement for video object based texture coding [37], and they extend the 8x8 2D DCT/IDCT algorithms by including extra processing steps that manipulate the video object's shape information. The gain is increased compression efficiency at the cost of additional computation. This work focuses on absorbing these additional SA-IDCT specific processing stages in an efficient manner in terms of power consumption, computation latency and silicon area. In this way, the gains associated with the SA-IDCT are achieved without significant impact on hardware resources. The SA-IDCT is one of the most computationally demanding blocks in an MPEG-4 video codec, therefore energy-efficient implementations are important – especially on battery powered wireless platforms. Power consumption issues require resolution for mobile MPEG-4 hardware solutions to become viable.

The main principles behind the design of this architecture are as follows:

- The SA-IDCT algorithm has been analysed and the computation steps have been re-formulated and merged on the premise that if there are less operations to be carried out, there will be less switching and hence less energy dissipation.
- Since the computational load of the SA-IDCT algorithm depends entirely on the VOP shape, the circuit switching activity and processing latency is proportional to the amount of VOP pels in a particular 8x8 block.
- Even/odd decomposition of SA-IDCT basis matrices to reduce computational complexity.
- Reconfiguring adder-based distributed arithmetic implementation of a variable *N*-point IDCT processor avoiding power consumptive hardware multipliers.
- An efficient addressing scheme is used for parsing the shape information to schedule the processing sequence and the reconstruction of the resultant pixel block.
- General low power properties such as a pipelined/interleaved datapath and local clock gating controlled by the processing load requirements.

10.14.4 Functional Description

10.14.4.1 Functional Description Details

The *isadct_top* module has been implemented with sub-modules that compute various stages of the SA-IDCT algorithm. A system block diagram is shown in Figure 193.

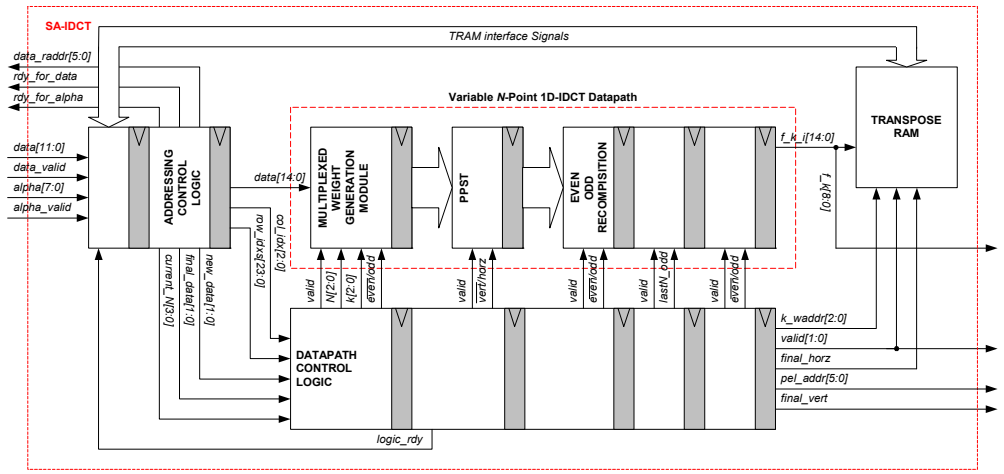


Figure 193 — SA-IDCT System Architecture.

The top-level SA-IDCT architecture in Figure 193 comprises of the TRAM and datapath with their associated control logic. It has a similar architecture to the SA-DCT proposed previously [39]. For all modules, local clock gating is employed based on the computation being carried out to avoid wasted power. The addressing

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

control logic (ACL) firstly parses the 8 × 8 shape information in a vertical raster fashion to interpret the horizontal and vertical N values as well as saving the shape pattern into interleaved buffer register files. Each register file is 128 bits wide (16 × 4-bits for N values plus 64 bits for shape pattern). Interleaving the data increases throughput since when one 8 × 8 block is being processed by the SA-IDCT logic, the ACL can begin to parse the next block into the alternate interleaved buffer. The ACL then uses the parsed shape information to read the input SADCT coefficient data from memory – this requires num_vo_pels memory accesses where num_vo_pels is the subset of pels in the 8 × 8 block that form part of the VO. The SA-DCT coefficients are parsed in a horizontal raster fashion. Each successive row vector is stored in an alternate interleaved buffer to improve latency.

When loaded, a vector is then passed to the variable N-point 1D IDCT module, which computes all N reconstructed pixels serially in a ping-pong fashion (i.e. f[N − 1],f[0],f[N − 2],f[1],. . .). The variable N-point 1D IDCT module is a five stage pipeline and employs adder-based distributed arithmetic using a multiplexed weight generation module (MWGM) and a partial product summation tree (PPST), followed by even-odd recomposition. The MWGM and the PPST are very similar in architecture to the corresponding modules used for the previously proposed SA-DCT architecture [39]. The even-odd recomposition module has a three clock cycle latency whereas the corresponding decomposition module used for the SA-DCT has only a single cycle latency. This accounts for the additional two cycles over the variable N-point 1D DCT module.

The TRAM has a 64 word capacity, and when storing data the index k is manipulated to store the value at address 8*N_vert[k] + k after which N_vert[k] is incremented by 1. The TRAM addressing scheme used by the ACL looks after the horizontal and vertical re-alignment steps by using the parsed shape information. The data read from the TRAM is routed to the datapath which is re-used to compute the final reconstructed pixels that are driven to the module output along with its address in the 8 × 8 block according to the shape information.

10.14.4.2 I/O Diagram

The top-level I/O signals of the isadct_top module are summarised in Figure 194.

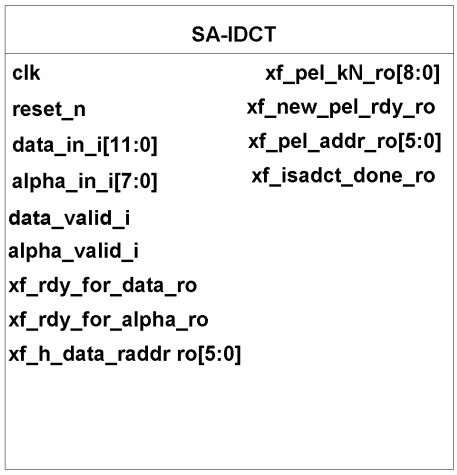


Figure 194 — Top Level I/O Ports.

10.14.4.3 I/O Ports Description

Table 55

Port Name	Port Width	Direction	Description
clk	1	Input	System clock
reset_n	1	Input	Asynchronous active-low reset
data_in_i[11:0]	12	Input	Serial port for reading VOP SA-DCT coefficients.
alpha_in_i[7:0]	8	Input	Serial port for reading VOP block alpha data
data_valid_i	1	Input	Active-high signal when asserted indicates that a valid VOP SA-DCT coefficient is present on the input data bus

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

alpha_valid_i	1	Input	Active-high signal when asserted indicates that a valid alpha data is present on the input alpha bus
xf_rdy_for_data_ro	1	Output	Active-high signal when asserted indicates that a new SA-DCT coefficient read address is on the output bus
xf_rdy_for_alpha_ro	1	Output	Active-high signal when asserted indicates that the IP core is ready to parse a new alpha block
xf_h_data_raddr_ro[5:0]	6	Output	SA-DCT coefficient address bus (horizontal raster)
xf_pel_kN_ro[8:0]	9	Output	Reconstructed pixel output port
xf_new_pel_rdy_ro	1	Output	Active-high signal when asserted indicates that a valid reconstructed pixel is present on the output data port
xf_isadct_done	1	Output	Active-high pulse signal that indicates the final reconstructed pixel for a particular block is on the output port if xf_new_pel_rdy_ro is also asserted. If asserted and xf_new_pel_rdy_ro is de-asserted a transparent VOP block has been detected
xf_pel_addr_ro[5:0]	6	Output	Reconstructed pixel block address

10.14.4.3.1 Parameters (generic)

Table 56

Parameter Name	Type	Range	Description
T_TRANSPOSE	Integer	4	Bit width of fractional part of intermediate reconstructed pixels (after horizontal inverse transformation)

10.14.4.3.2 Parameters (constants)

N/A.

10.14.5 Algorithm

The algorithm implemented by this module is the SA-IDCT required for object-based texture encoding of video objects for MPEG-4 ACE profile and above. The SA-IDCT is less regular compared to the 8x8 block-based IDCT since its processing decisions are entirely dependent on the shape information associated with each individual block. The 8 × 8 IDCT requires 16 1D 8-point IDCT computations if implemented using the row-column approach. Each 1D transformation has a fixed length of 8, with fixed basis functions. This is amenable to hardware implementation since the data path is fixed and all parameters are constant. The SA-IDCT requires up to 16 1D N-point DCT computations where $N \in \{2,3\ldots8\}$ ($N \in \{0,1\}$ are trivial cases). In general N can vary across the possible 16 computations depending on the shape. With the SA-IDCT the basis functions vary with N, complicating hardware implementation.

A sample SA-IDCT computation showing each high-level processing stage is shown in Figure 195, of which there are 6 in total.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

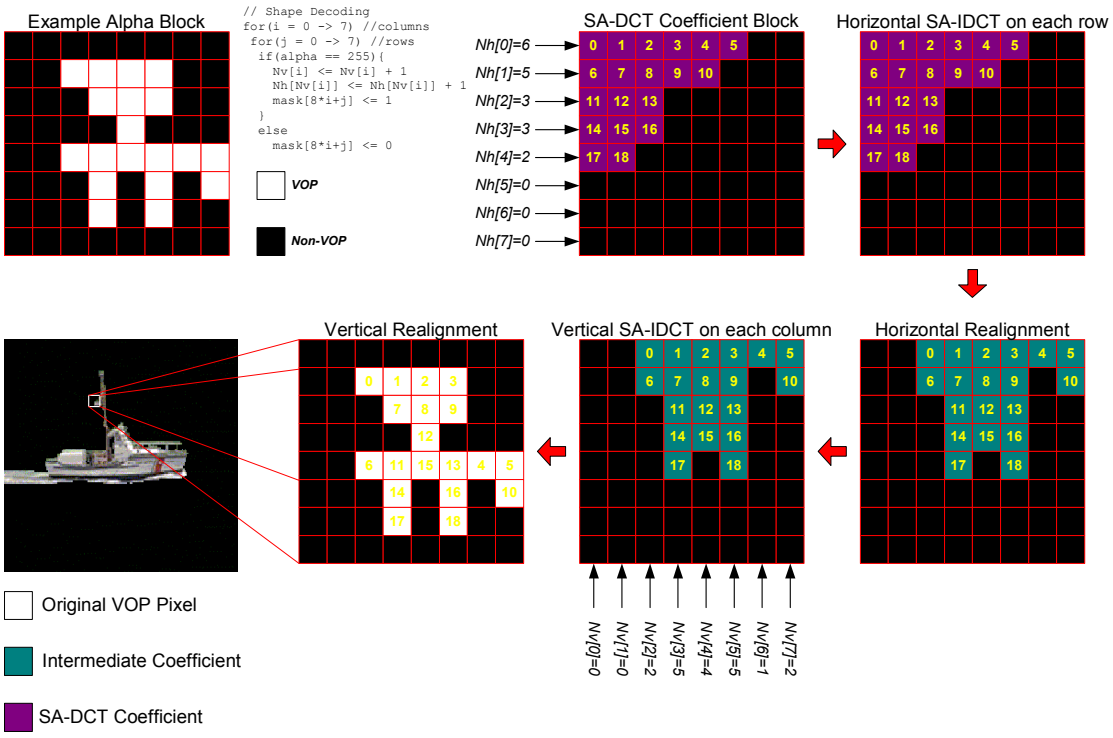


Figure 195 — Example showing SA-IDCT computation stages.

Additional non-trivial alignment stages are required for the SA-IDCT that are unnecessary for the conventional 8 × 8 IDCT. In summary, the SA-IDCT processing stages are:

- Stage 0 – Parse the 8 × 8 shape block
- Stage 1 – Read appropriate SA-DCT coefficient block from memory
- Stage 2 – Horizontal *N*-point 1D IDCT on each row
- Stage 3 – Horizontally re-align intermediate reconstructed pixels
- Stage 4 – Vertical *N*-point 1D IDCT on each column of intermediate reconstructed pixels
- Stage 5 – Vertical re-alignment of final reconstructed pixels to original block addresses

The block-based 8 × 8 IDCT does not require stages 0, 3 and 5. With the SA-IDCT, the processing load varies depending on the alpha mask so there is scope for adapting the number of processing steps based on the shape information to achieve minimum processing latency.

10.14.6 Implementation

The architecture *isadct_top* has been implemented using Verilog HDL in a structural style with RTL sub-modules as summarised in Figure 193. The module has been integrated with an adapted version of the multiple IP-core hardware accelerated software system framework developed by the University of Calgary. The entire system along with host software calls has been implemented on a Windows 2000 laptop with the Annapolis PCMCIA FPGA (Xilinx Virtex-II XC2V 3000-4) prototyping platform installed. The main alterations were to the hardware module controller (to comply with the interface shown in Figure 194). Also, since the alpha information is required for SA-IDCT processing, the host software was altered to store the alpha information along with the texture information in the SRAM on the prototyping platform.

The design flow followed is summarised in Figure 196. The SA-IDCT core was coded in Verilog at RTL level and simulated with a testbench using ModelSim SE v6.0a.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2009 – All rights reserved

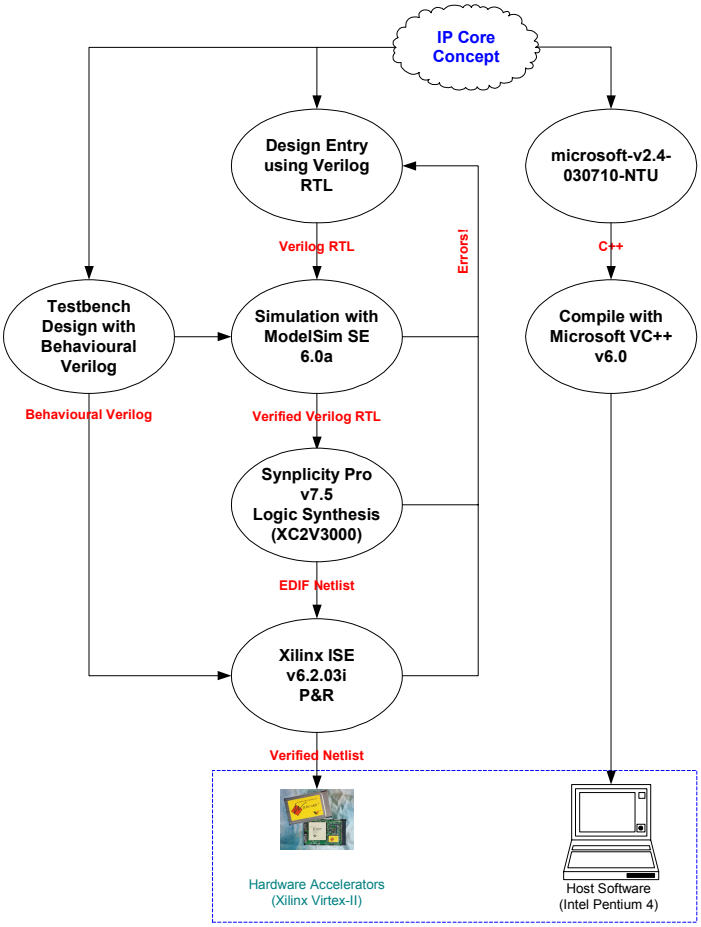


Figure 196 — Design flow from concept to implementation

Once verified, the Verilog RTL of the SA-IDCT core was integrated with an adapted version of the VHDL multiple IP-core integration framework developed by the University of Calgary. The only HDL module that required major modification was the hardware module controller, which interfaces the SA-IDCT core with the rest of the integration framework. The entire HDL system was synthesised with Synplicity Pro (v7.5) targeting the WildCard Xilinx Virtex-II (XC2V3000) FPGA. Xilinx ISE (v6.203i) was used to place and route the netlist created by Synplicity Pro. The host software used is the Microsoft MPEG-4 Part 7 Optimised Video Reference Software (version microsoft-v2.4-030710-NTU) as hosted by the National Chiao-Tung University, Taiwan [40].

10.14.6.1 Interfaces

The input ports apart form the clock and reset signal are driven by the hardware module controller to serially read VOP block alpha (vertical raster) and coefficient data (horizontal raster) from the SRAM (via the memory source block and the hardware module controller). When the IP core asserts *xf_rdy_for_alpha_ro* this indicates that it is ready to process a new block. The hardware module controller serially streams the next alpha block on *alpha_in_i[7:0]* to the IP core and asserts *alpha_valid_i* to indicate cycles where valid alpha data is on the bus. The IP core parses this information, and interprets the *N* values for each of the rows and columns, as well as recording the shape pattern in a 64 bit register. At this point the IP core knows the shape pattern, so it can read the SA-DCT coefficients from memory via the hardware module controller. This is achieved by driving the coefficient block address on *xf_h_data_raddr_ro[5:0]* and asserting *xf_rdy_for_data_ro*. The hardware module controller read the appropriate coefficients from the SRAM and responds by asserting *data_valid_i* and placing the data on *data_in_i[11:0]*.

When *xf_new_pel_rdy_ro* is asserted (active-high) the module is indicating to the hardware module controller that a new reconstructed pixel is present on the *xf_pel_kN_ro[8:0]* output port, with its corresponding block address present on *xf_pel_addr_ro[5:0]*. The hardware module controller writes pixels back to the SRAM via the memory destination module. The signal *xf_isadct_done_ro* is used to indicate to the hardware module controller that the final VOP coefficient for a particular block is present on *xf_pel_kN_ro[8:0]* (if *xf_new_pel_rdy_ro* is asserted in the same cycle) or if a fully transparent block has been detected (if

© ISO/IEC 2009 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

xf_new_pel_rdy_ro is de-asserted in the same cycle). This extra handshaking signal is necessary since the amount of VOP pixels in a particular block can vary depending on the shape ($64 \geq M \geq 0$) and it is efficient to exploit this fact for processing latency gains.

10.14.6.2 Register File Access

Four 32-bit master socket registers are programmed directly by the host software to configure parameters for the SA-IDCT core. These registers are used to configure the hardware module controller.

Table 57

Register Name	Range	Description
dControl[0][31:0]	[0][9:0]	Frame width
	[0][19:10]	Frame Height
	[0][23:20]	Number of frames that are read/written at a time
	[0][31:0]	Undefined
dControl[1][31:0]	[1][20:0]	SRAM read start address
	[1][31:21]	Undefined
dControl[2][31:0]	[2][20:0]	SRAM write start address
	[2][31:21]	Undefined
dControl[3][31:0]	[3][31:0]	Configure hardware module controller for debug mode or normal mode. Debug mode allows host software to monitor the hardware module controller state machine

The host software programs these registers after the frame data has been written to the SRAM. They are written by using a WildCard API function *WC_PeRegWrite*. This function writes each of the four 32-bit values into the hardware register file configuration registers at a specific offset according to the specific hardware accelerator being strobed. The abridged code listing in subclause 10.14.8 shows how the API function is called.

10.14.6.3 Timing Diagrams

Figure 197 shows a simplified I/O timing diagram for the SA-IDCT architecture, and the worst case processing latency is 188 clock cycles. This worst case occurs if the alpha block is fully opaque.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

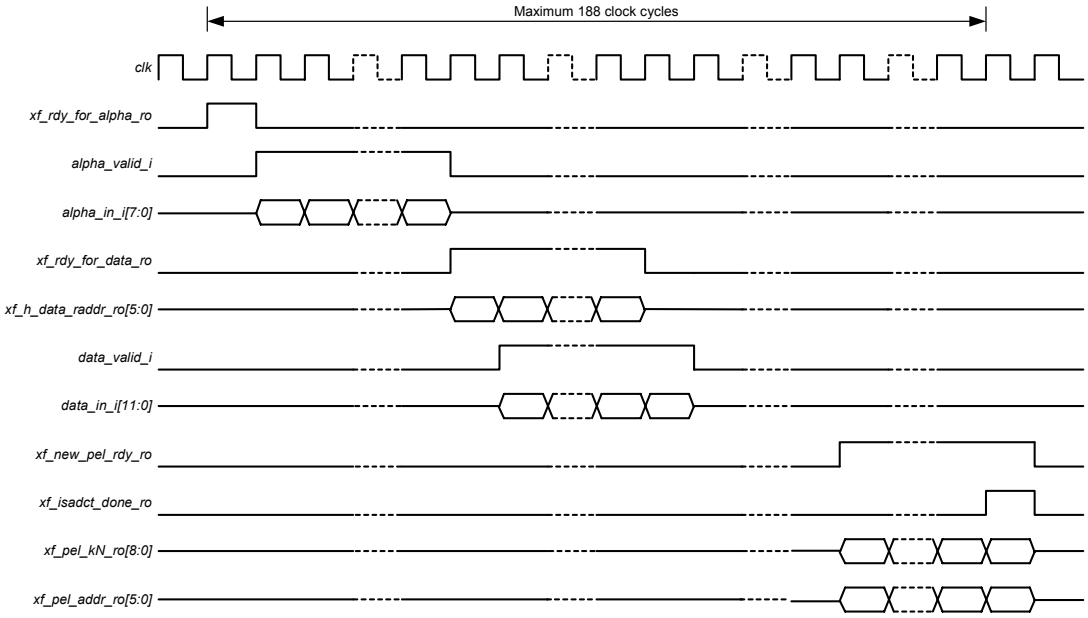


Figure 197 — Sample I/O Timing Diagram

10.14.7 Results of Performance & Resource Estimation

The module has been integrated with the University of Calgary's integration framework and the MPEG-4 Part 7 optimised reference software implemented on the Annapolis WildCard FPGA prototyping platform with associated host calling software. The IP core has been implemented using Verilog RTL and verified using ModelSim SE v6.0a. The Verilog RTL was then synthesised using Synplicity Pro (version 7.5) followed by place and route using Xilinx ISE (version 6.2.03i). The synthesis and place & route scripts were adapted from those proposed by the University of Calgary. To obtain resource usage information for the IP core itself a synthesis run was carried out with the IP core only without the surrounding integration framework and pin assignments (since the IP core is not connected to any FPGA pins directly). An abridged version of the mapping report is given in the following code listing:

```
Release 6.3.03i Map G.38
Xilinx Mapping Report File for Design 'isadct_top'

Design Information
------------------
Command Line   : C:/Xilinx/bin/nt/map.exe -intstyle ise -p XC2V3000-FG676-4 -cm
area -pr b -k 4 -c 100 -tx off -o isadct_top_map.ncd isadct_top.ngd
isadct_top.pcf
Target Device  : x2v3000
Target Package : fg676
Target Speed   : -4
Mapper Version : virtex2 -- $Revision: 1.16.8.2 $
Mapped Date    : Thu Nov 17 18:01:13 2005

Design Summary
--------------
Number of errors:      0
Number of warnings:    0
Logic Utilization:
  Number of Slice Flip Flops:       1,977 out of  28,672    6%
  Number of 4 input LUTs:           3,768 out of  28,672   13%
Logic Distribution:
  Number of occupied Slices:        2,763 out of  14,336   19%
  Number of Slices containing only related logic:   2,763 out of   2,763  100%
```

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
  Number of Slices containing unrelated logic:          0 out of   2,763    0%
        *See NOTES below for an explanation of the effects of unrelated logic
Total Number 4 input LUTs:          3,774 out of  28,672   13%
  Number used as logic:             3,768
  Number used as a route-thru:          6

  Number of bonded IOBs:               49 out of     484   10%
  Number of GCLKs:                      1 out of      16    6%

Total equivalent gate count for design:  42,787
Additional JTAG gate count for IOBs:  2,352
Peak Memory Usage:  144 MB

Section 13 - Additional Device Resource Counts
-----------------------------------------------
Number of JTAG Gates for IOBs = 49
Number of Equivalent Gates for Design = 42,787
Number of RPM Macros = 0
Number of Hard Macros = 0
CAPTUREs = 0
BSCANs = 0
STARTUPs = 0
PCILOGICs = 0
DCMs = 0
GCLKs = 1
ICAPs = 0
18X18 Multipliers = 0
Block RAMs = 0
TBUFs = 0
Total Registers (Flops & Latches in Slices & IOBs) not driven by LUTs = 1315
IOB Dual-Rate Flops not driven by LUTs = 0
IOB Dual-Rate Flops = 0
IOB Slave Pads = 0
IOB Master Pads = 0
IOB Latches not driven by LUTs = 0
IOB Latches = 0
IOB Flip Flops not driven by LUTs = 0
IOB Flip Flops = 0
Unbonded IOBs = 0
Bonded IOBs = 49
Total Shift Registers = 0
Static Shift Registers = 0
Dynamic Shift Registers = 0
16x1 ROMs = 0
16x1 RAMs = 0
32x1 RAMs = 0
Dual Port RAMs = 0
MUXFs = 753
MULT_ANDs = 5
4 input LUTs used as Route-Thrus = 6
4 input LUTs = 3768
Slice Latches not driven by LUTs = 0
Slice Latches = 0
Slice Flip Flops not driven by LUTs = 1315
Slice Flip Flops = 1977
Slices = 2763
Number of LUT signals with 4 loads = 94
Number of LUT signals with 3 loads = 123
Number of LUT signals with 2 loads = 652
Number of LUT signals with 1 load = 2521
```


© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
NGM Average fanout of LUT = 2.85
NGM Maximum fanout of LUT = 147
NGM Average fanin for LUT = 3.2869
Number of LUT symbols = 3768
Number of IPAD symbols = 24
Number of IBUF symbols = 24
```

At CIF resolution with a frame rate of 30 fps requires 71280 blocks to be processed per second. This implies that the SA-IDCT should be capable of processing a single 8 × 8 block in approximately 14μs. Given that the worst-case number of cycles for the IP core to process a block is 188 cycles, the IP core must run at approximately 14MHz at worst to maintain real-time constraints. The place and route report generated by ISE indicates a theoretical operating frequency of approximately 75.3MHz so even allowing for some error in the report the IP core should be able to handle real time processing of CIF sequences quite comfortably. Operating at 75.3 MHz the module is capable of processing at least 25.63Mpixel/s.

10.14.8 API calls from reference software

The hardware acceleration framework with the integrated SA-IDCT IP core has been integrated with Microsoft MPEG-4 Part 7 Optimised Video Reference Software (version microsoft-v2.4-030710-NTU) [40]. As can be seen from the following code listing, the hardware accelerator for the SA-IDCT is called in file *sadct.cpp* in the class *CInvSADCT*. Based on a pre-processor directive, the function *DCU_ISADCT_HWA* is called and this looks after initiating the appropriate protocols with the SA-IDCT hardware accelerator on the WildCard-II FPGA.

```
Void CInvSADCT::apply (const PixelI* rgiSrc, Int nColSrc, PixelI* rgiDst, Int nColDst, const PixelC*
rgchMask, Int nColMask)
{
    if (rgchMask) {
        // boundary block assumed, for details refer to the comment
        // inside the body of the other apply method.
        prepareMask(rgchMask, nColMask);
        prepareInputBlock(m_in, rgiSrc, nColSrc);

// Schueuer HHI: added for fast_sadct
#ifdef _FAST_SADCT_
        fast_transform(m_out, m_in, m_mask, m_N, m_N);
#elif _DCU_ISADCT_HWA_
    DCU_SA_IDCT_HWA(m_out, m_in, m_mask, m_N, m_N);
#else
        transform(m_out, m_in, m_mask, m_N, m_N);
#endif
…
```

10.14.9 Conformance Testing

10.14.9.1 Reference software type, version and input data set

The hardware acceleration framework with the integrated SA-IDCT IP core has been integrated with Microsoft MPEG-4 Part 7 Optimised Video Reference Software (version microsoft-v2.4-030710-NTU). The parameter file used to configure the encoder is shown below (*Source.FilePrefix* changes depending on the test sequence name). Figure 198 shows an uncompressed CIF resolution frame (from the coastguard sequence), the software-only reconstructed VOP frame and the SA-IDCT hardware accelerated reconstructed VOP frame. The conformance test verifies that the software-only bitstream and the hardware accelerated bitstream are identically bit accurate.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2009 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

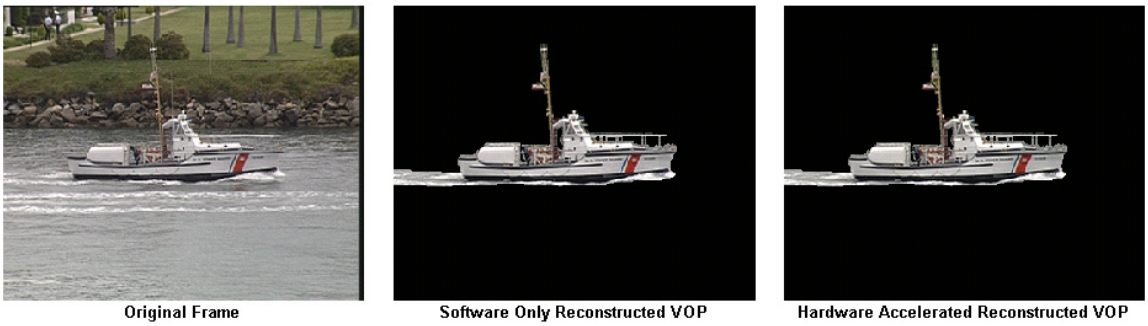


Figure 198 — Example frames for conformance testing of the coastguard sequence.

```
Version = 904                       // parameter file version

// When VTC is enabled, the VTC parameter file is used instead of this one.
VTC.Enable = 0
VTC.Filename = ""

VersionID[0] = 2             // object stream version number (1 or 2)

Source.Width = 352
Source.Height = 288
Source.FirstFrame = 0
Source.LastFrame = 299
Source.ObjectIndex.First = 255
Source.ObjectIndex.Last = 255
Source.FilePrefix = "nh_cg1_cif"
Source.Directory = "."
Source.BitsPerPel = 8
Source.Format [0] = "420"            // One of "444", "422", "420"
Source.FrameRate [0] = 10
Source.SamplingRate [0] = 1

Output.Directory.Bitstream = ".\cmp"
Output.Directory.DecodedFrames = ".\rec"

Not8Bit.Enable = 0
Not8Bit.QuantPrecision = 5

RateControl.Type [0] = "None"         // One of "None", "MP4", "TM5"
RateControl.BitsPerSecond [0] = 50000

Scalability [0] = "None"          // One of "None", "Temporal", "Spatial"
Scalability.Temporal.PredictionType [0] = 0       // Range 0 to 4
Scalability.Temporal.EnhancementType [0] = "Full" // One of "Full", "PartC", "PartNC"
Scalability.Spatial.EnhancementType [0] = "PartC" // One of "Full", "PartC", "PartNC"
Scalability.Spatial.PredictionType [0] = "PBB"    // One of "PPP", "PBB"
Scalability.Spatial.Width [0] = 352
Scalability.Spatial.Height [0] = 288
Scalability.Spatial.HorizFactor.N [0] = 2 // upsampling factor N/M
Scalability.Spatial.HorizFactor.M [0] = 1
Scalability.Spatial.VertFactor.N [0] = 2  // upsampling factor N/M
Scalability.Spatial.VertFactor.M [0] = 1
Scalability.Spatial.UseRefShape.Enable [0] = 0
Scalability.Spatial.UseRefTexture.Enable [0] = 0
Scalability.Spatial.Shape.HorizFactor.N [0] = 2   // upsampling factor N/M
Scalability.Spatial.Shape.HorizFactor.M [0] = 1
```


© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
Scalability.Spatial.Shape.VertFactor.N [0] = 2    // upsampling factor N/M
Scalability.Spatial.Shape.VertFactor.M [0] = 1

Quant.Type [0] = "H263"              // One of "H263", "MPEG"

GOV.Enable [0] = 0
GOV.Period [0] = 0               // Number of VOPs between GOV headers

Alpha.Type [0] = "Binary"            // One of "None", "Binary", "Gray", "ShapeOnly"
Alpha.MAC.Enable [0] = 0
Alpha.ShapeExtension [0] = 0         // MAC type code
Alpha.Binary.RoundingThreshold [0] = 0
Alpha.Binary.SizeConversion.Enable [0] = 0
Alpha.QuantStep.IVOP [0] = 16
Alpha.QuantStep.PVOP [0] = 16
Alpha.QuantStep.BVOP [0] = 16
Alpha.QuantDecouple.Enable [0] = 0
Alpha.QuantMatrix.Intra.Enable [0] = 0
Alpha.QuantMatrix.Intra [0] = {}     // { insert 64 comma-separated values }
Alpha.QuantMatrix.Inter.Enable [0] = 0
Alpha.QuantMatrix.Inter [0] = {}     // { insert 64 comma-separated values }

Texture.IntraDCThreshold [0] = 0     // See note at top of file
Texture.QuantStep.IVOP [0] = 16
Texture.QuantStep.PVOP [0] = 16
Texture.QuantStep.BVOP [0] = 16
Texture.QuantMatrix.Intra.Enable [0] = 0
Texture.QuantMatrix.Intra [0] = {}       // { insert 64 comma-separated values }
Texture.QuantMatrix.Inter.Enable [0] = 0
Texture.QuantMatrix.Inter [0] = {}       // { insert 64 comma-separated values }
Texture.SADCT.Enable [0] = 1

Motion.RoundingControl.Enable [0] = 1
Motion.RoundingControl.StartValue [0] = 0
Motion.PBetweenICount [0] = -1
Motion.BBetweenPCount [0] = 2
Motion.SearchRange [0] = 16
Motion.SearchRange.DirectMode [0] = 2     // half-pel units
Motion.AdvancedPrediction.Enable [0] = 0
Motion.SkippedMB.Enable [0] = 1
Motion.UseSourceForME.Enable [0] = 1
Motion.DeblockingFilter.Enable [0] = 0
Motion.Interlaced.Enable [0] = 0
Motion.Interlaced.TopFieldFirst.Enable [0] = 0
Motion.Interlaced.AlternativeScan.Enable [0] = 0
Motion.ReadWriteMVs [0] = "Off"           // One of "Off", "Read", "Write"
Motion.ReadWriteMVs.Filename [0] = "MyMVFile.dat"
Motion.QuarterSample.Enable [0] = 0

Trace.CreateFile.Enable [0] = 1
Trace.DetailedDump.Enable [0] = 1

Sprite.Type [0] = "None"          // One of "None", "Static", "GMC"
Sprite.WarpAccuracy [0] = "1/2"           // One of "1/2", "1/4", "1/8", "1/16"
Sprite.Directory = "\\swinder1\sprite\brea\spt"
Sprite.Points [0] = 0                // 0 to 4, or 0 to 3 for GMC
Sprite.Points.Directory = "\\swinder1\sprite\brea\pnt"
Sprite.Mode [0] = "Basic"            // One of "Basic", "LowLatency", "PieceObject", "PieceUpdate"

ErrorResil.RVLC.Enable [0] = 0
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
ErrorResil.DataPartition.Enable [0] = 0
ErrorResil.VideoPacket.Enable [0] = 0
ErrorResil.VideoPacket.Length [0] = 0
ErrorResil.AlphaRefreshRate [0] = 1

Newpred.Enable [0] = 0
Newpred.SegmentType [0] = "VideoPacket"       // One of "VideoPacket", "VOP"
Newpred.Filename [0] = "example.ref"
Newpred.SliceList [0] = "0"

RRVMode.Enable [0] = 0               // Reduced resolution VOP mode
RRVMode.Cycle [0] = 0

Complexity.Enable [0] = 1            // Global enable flag
Complexity.EstimationMethod [0] = 1      // 0 or 1
Complexity.Opaque.Enable [0] = 1
Complexity.Transparent.Enable [0] = 1
Complexity.IntraCAE.Enable [0] = 1
Complexity.InterCAE.Enable [0] = 1
Complexity.NoUpdate.Enable [0] = 1
Complexity.UpSampling.Enable [0] = 1
Complexity.IntraBlocks.Enable [0] = 1
Complexity.InterBlocks.Enable [0] = 1
Complexity.Inter4VBlocks.Enable [0] = 1
Complexity.NotCodedBlocks.Enable [0] = 1
Complexity.DCTCoefs.Enable [0] = 1
Complexity.DCTLines.Enable [0] = 1
Complexity.VLCSymbols.Enable [0] = 1
Complexity.VLCBits.Enable [0] = 1
Complexity.APM.Enable [0] = 1
Complexity.NPM.Enable [0] = 1
Complexity.InterpMCQ.Enable [0] = 1
Complexity.ForwBackMCQ.Enable [0] = 1
Complexity.HalfPel2.Enable [0] = 1
Complexity.HalfPel4.Enable [0] = 1
Complexity.SADCT.Enable [0] = 1
Complexity.QuarterPel.Enable [0] = 1

VOLControl.Enable [0] = 0
VOLControl.ChromaFormat [0] = 0
VOLControl.LowDelay [0] = 0
VOLControl.VBVParams.Enable [0] = 0
VOLControl.Bitrate [0] = 0           // 30 bits
VOLControl.VBVBuffer.Size [0] = 0    // 18 bits
VOLControl.VBVBuffer.Occupancy [0] = 0       // 26 bits
```

10.14.9.2 API vector conformance

At API level the test vectors used are the CIF and QCIF test sequences as defined by the MPEG-4 Video Verification Model [41].

10.14.9.3 End to end conformance (conformance of encoded bitstreams or decoded pictures)

End to end conformance has been completed and it has been verified that the bitstreams produced by the encoder with and without SA-IDCT hardware acceleration are identical.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.15 A VERILOG HARDWARE IP BLOCK FOR 2D-DCT (8X8)

10.15.1 Abstract description of the module

This code for 2D-DCT (8x8) is implemented based on one of the recent proposed architecture, called the New Distributed Arithmetic architecture (NEDA). The advantage of NEDA architecture is that it can be implemented with only adders and some shift registers at final stage. The HDL code is written in Verilog HDL.

10.15.2 Module specification

10.15.2.1 MPEG 4 part:	***4***
10.15.2.2 Profile :	***All***
10.15.2.3 Level addressed:	***All***
10.15.2.4 Module Name:	***2D-DCT***
10.15.2.5 Module latency:	***Approx 1.48 us***
10.15.2.6 Module data throughput:	***1 transformed coefficient ()/clock cycle***
10.15.2.7 Max clock frequency:	***86.7 MHz***
10.15.2.8 Resource usage:	
10.15.2.8.1 CLB Slices:	***1411***
10.15.2.8.2 Block RAMs:	***1***
10.15.2.8.3 Multipliers:	***none***
10.15.2.8.4 External memory:	***none***
10.15.2.8.5 Other Metrics:	***none***
10.15.2.9 Revision:	***1.00***
10.15.2.10 Authors:	***Wael Badawy, and Graham Jullien***
10.15.2.11 Creation Date:	***July 2002***
10.15.2.12 Modification Date:	***October 2004***

10.15.3 Introduction

One of the basic building modules of any video or image coder is the Transform Coding block. The purpose of this block is to compress the parts of the image into a more correlated form so it would be more efficiently coded. MPEG-4 employs the Discrete Cosine Transform (DCT) as its transform coder. There are many existing architectures and hardware realizations for the DCT. But the high demands of future applications for MPEG-4 require a very high-throughput architecture for the DCT. Most existing architectures are either based on Multiply/Accumulate units or ROM, which are relatively slow and provide low-throughput. A very high-throughput and high-speed architecture for the DCT is of extreme importance in future applications.

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.15.4 Functional Description

10.15.4.1 Functional description details

10.15.4.2 I/O Diagram

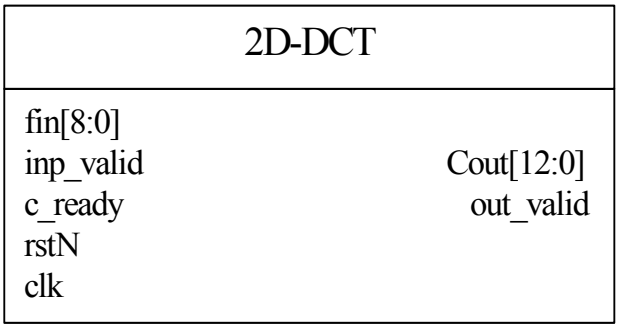


Figure 199

10.15.4.3 I/O Ports Description

Table 58

Port Name	Port Width	Direction	Description
fin[8:0]	9	Input	9-bit input to the module
inp_valid	1	Input	A high for one clock cycle indicates the start of first input of the 64 byte input tile
out_valid	1	Output	A high for one clock cycle indicates the start of the first 2D DCT output coefficient, followed by 63 more
clk	1	Input	Clock of the module
c_ready	1	Output	Encoded bitstream
rstN	1	Input	Reset to the module
Cout	13	Output	Output transformed coefficient

10.15.5 Algorithm

The 8x1 point DCT {F(u): u = [0,7]} for a given real input sequence {f(x): x = [0,7]} is defined as:

$$F(u) = c(u)\sum_{x=0}^{7} f(x)\cdot\cos\frac{u\pi(2x+1)}{2\times 8}$$

$$where\ c(u) = \begin{cases} (1/\sqrt{2}) & \text{, for u = 0} \\ 1 & \text{, otherwise} \end{cases}$$

Equation 65

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Rewriting the above equation in matrix forms, we get:

$$F(u) = \sum_{x=0}^{7} [A_u(x)] \cdot f(x) = [A_u(0) \cdots A_u(7)] \cdot \begin{bmatrix} f(0) \\ \vdots \\ f(7) \end{bmatrix}$$

where $A_u(x) = c(u) \cdot \cos(u\pi((2x)+1)/(2\times 8))$ and $x = 0,1,2,3,4,5,6,7$.

Equation 66

The above inner product of input samples by the DCT coefficients is implemented using NEDA algorithm, which is described in the next section.

10.15.5.1 NEDA Algorithm

This technique applies the concept of DA in a new way. Instead of distributing the inputs as in conventional DA, it distributes the coefficients. The mathematical derivation of the algorithm is as follows.

If DA precision is chosen to be (M-N+1) bit for a fixed value of u, real number coefficients $A_u(x)$ in eqn.2 can be represented in 2' compliment format, where M is the index of sign bit, and N is the index of the least significant bit. For simplicity $A_u(x)$ is assumed A_x.

$$A_x = -A_{x,M} \cdot 2^M + \sum_{i=N}^{M-1} A_{x,i} \cdot 2^i$$

Equation 67

where A_x is x^{th} coefficient, and $A_{x,i}$ is the jth bit of the xth coefficient $A_{x,i}$ can be either zero or one. Substituting Equation 67 in Equation 66 , we have.

$$F(u) = \sum_{x=0}^{7} \left(-A_{x,M} \cdot 2^M + \sum_{i=N}^{M-1} A_{x,i} \cdot 2^i\right) \cdot f(x)$$

Equation 68

Rearranging the terms and combining the constants we have:

$$F(u) = -2^M \cdot \left(\sum_{x=0}^{7} A_{x,M} \cdot f(m)\right) + \sum_{i=N}^{M-1} 2^i \cdot \left(\sum_{x=0}^{7} A_{x,i} \cdot f(m)\right)$$

Equation 69

Equation 69 can be rewritten in the matrix representation as follows:

© ISO/IEC 2009 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

$$F(u) = [\ 2^N\ 2^{N+1} \ldots 2^M\ ] \begin{pmatrix} A_0^N & A_1^N & \cdots & A_7^N \\ A_0^{N+1} & A_1^{N+1} & \cdots & A_7^{N+1} \\ . & . & & . \\ . & . & & . \\ . & . & & . \\ -A_0^M & -A_1^M & \cdots & -A_7^N \end{pmatrix} \begin{pmatrix} f(0) \\ f(1) \\ . \\ . \\ . \\ f(7) \end{pmatrix}$$

$$F(u) = [\ 2^N\ 2^{N+1} \ldots 2^M\ ] \cdot \begin{pmatrix} F_{u,N} \\ F_{u,N+1} \\ . \\ . \\ . \\ \text{inv}\ (F_{u,M}) \end{pmatrix}$$

Equation 70

Since matrix A consists of 0's and 1's, following computation consists of only addition operations. Therefore matrix A is referred as Adder Matrix.

$$\begin{pmatrix} F_{u,N} \\ F_{u,N+1} \\ . \\ . \\ . \\ \text{inv}(F_{u,M}) \end{pmatrix} = [A] \cdot \begin{pmatrix} f(0) \\ f(1) \\ \\ . \\ . \\ f(7) \end{pmatrix}$$

Equation 71

The final stage of computing Equation 70 can be realized with shifting and addition. In NEDA, only one adder and one shift register implement the final stage. NEDA architecture is described in next clause.

The following example is presented to more clearly illustrate different steps of NEDA algorithm. The main aim is to find the DCT value for F(0), given the following 8 inputs :

$$\begin{pmatrix} f(0) \\ f(1) \\ f(2) \\ f(3) \\ f(4) \\ f(5) \\ f(6) \\ f(7) \end{pmatrix} = \begin{pmatrix} 2 \\ 5 \\ 14 \\ 3 \\ 8 \\ 11 \\ 3 \\ 5 \end{pmatrix}$$

First, the DCT coefficients $[A_0(0) \ldots A_0(7)]$ are calculated and represented in 2' complement format assuming the DA precision is 13 bits.

$A_0 = [A_0(0), A_0(1), A_0(2), A_0(3), A_0(4), A_0(5), A_0(6), A_0(7)] = [1/2\sqrt{2}, 1/2\sqrt{2}, \ldots.. 1/2\sqrt{2}]$

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Substituting each 1/2√2 with the 2' complement representation we get:

$$A_0 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{matrix} \leftarrow F_{u,-12} \\ \leftarrow F_{u,-11} \\ \leftarrow F_{u,-10} \\ \leftarrow F_{u,-9} \\ \leftarrow F_{u,-8} \\ \leftarrow F_{u,-7} \\ \leftarrow F_{u,-6} \\ \leftarrow F_{u,-5} \\ \leftarrow F_{u,-4} \\ \leftarrow F_{u,-3} \\ \leftarrow F_{u,-2} \\ \leftarrow F_{u,-1} \\ \leftarrow F_{u,0} \end{matrix}$$

$$\begin{matrix} \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow \\ f(0) & f(1) & f(2) & f(3) & f(4) & f(5) & f(6) & f(7) \end{matrix}$$

The bottom row of A_0 consists of the sign bits of $A_0(x)$, and the top row are LSB's of $A_0(x)$. Each row shows which inputs need to be added to get the associated $F_{u,i}$,where i = 0, …, -12. All zero rows mean no addition is needed. In our example $F_{0,-12} = F_{0,-11} =$

$F_{0,-10} = F_{0,-8} = F_{0,-6} = F_{0,-3} = F_{0,-1} = F_{0,0} = 0$,which need no further calculations, while

$$F_{0,-9} = F_{0,-7} = F_{0,-5} = F_{0,-4} = F_{0,-2} = f(0) + f(1) + \ldots\ldots + f(7) = \sum_{x=0}^{7} f(x) \quad \text{which need 7 additions.}$$

A Direct mapping to hardware requires 35 additions while eliminating the redundant adders, reduce the number of additions to 7. The butterfly structure for A_0 is shown in Figure 200.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

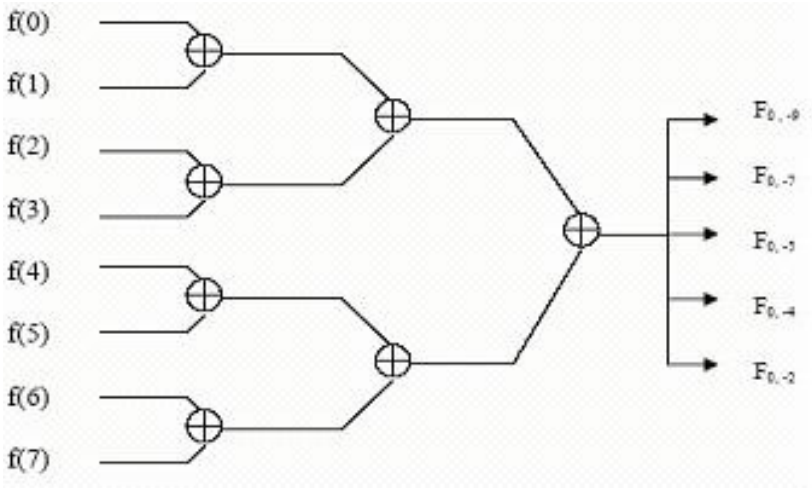


Figure 200

The last step is to calculate the following:

$$F(0) = [2^{-12}\ 2^{-11}\ \dots\ 2^{0}] \begin{pmatrix} F_{0,-12} \\ F_{0,-11} \\ F_{0,-10} \\ F_{0,-9} \\ F_{0,-8} \\ F_{0,-7} \\ . \\ . \\ . \\ . \\ . \\ \text{inv}(F_{0,0}) \end{pmatrix} = [2^{-12}\ 2^{-11}\ \dots\ 2^{0}] \begin{pmatrix} 0 \\ 0 \\ 0 \\ 51 \\ 0 \\ 51 \\ 0 \\ 51 \\ 51 \\ 0 \\ 51 \\ 0 \\ 0 \end{pmatrix}$$

where inv(.) means the two's complement of value $F_{0,0}$. It can be performed serially with one adder and one right shift register starting from $F_{0,-12}$.

Assuming the inputs are 9-bit long, we will need a 25 bits register in order not to loose any data. The format of the output, which is a real number, is considered Q13.12. It means 12 bits are considered for the decimal part and 13 bits for the integer part.

10.15.5.2 NEDA Algorithm

The 8x8 point DCT $\{F(u_1, u_2): u_1, u_2 = [0,7]\}$ for a given real input sequence $\{ f(x_1, x_2): x_1, x_2 = [0,7]\}$ is defined as:

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

$$F(u1,u2) = c(u1)\cdot c(u2)\sum_{x1=0}^{7}\sum_{x2=0}^{7} f(x1,x2)\cdot \cos\frac{u_1\pi(2x_1+1)}{2\times 8}\cos\frac{u_2\pi(2x_2+1)}{2\times 8}$$

$$where\ c(u1), c(u2) = \begin{cases} (1/\sqrt{2}) & \text{, for u1 \& u2 = 0} \\ 1 & \text{, otherwise} \end{cases}$$

10.15.6 Implementation

Direct implementation of above equation requires 8^4 multiplications and additions. By using the separability property of the DCT, the 2D-DCT can be calculated using eight 1D-DCT$_s$ operating on the rows of the block, followed by another 8 1D-DCT$_s$ operating on the column of the resulting coefficients of the first stage. Figure 201 shows the architecture. The length of the inputs and the outputs are standardized by the IEEE standards committee. While the precision of the intermediate results is left as a design decision.

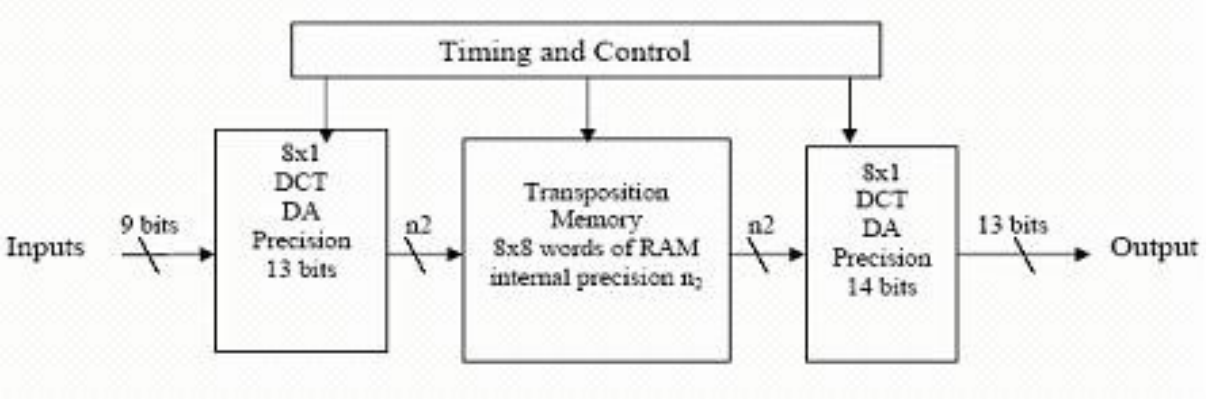


Figure 201

Using the above structure the design of the 8×8 DCT is simplified into the design of two similar 8×1 DCT modules. Each DCT module is implemented based on NEDA architecture.

10.15.6.1 Interfaces

N/A.

10.15.6.2 Register File Access

Please refer to subclause 10.15.4.

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.15.6.3 Timing Diagrams

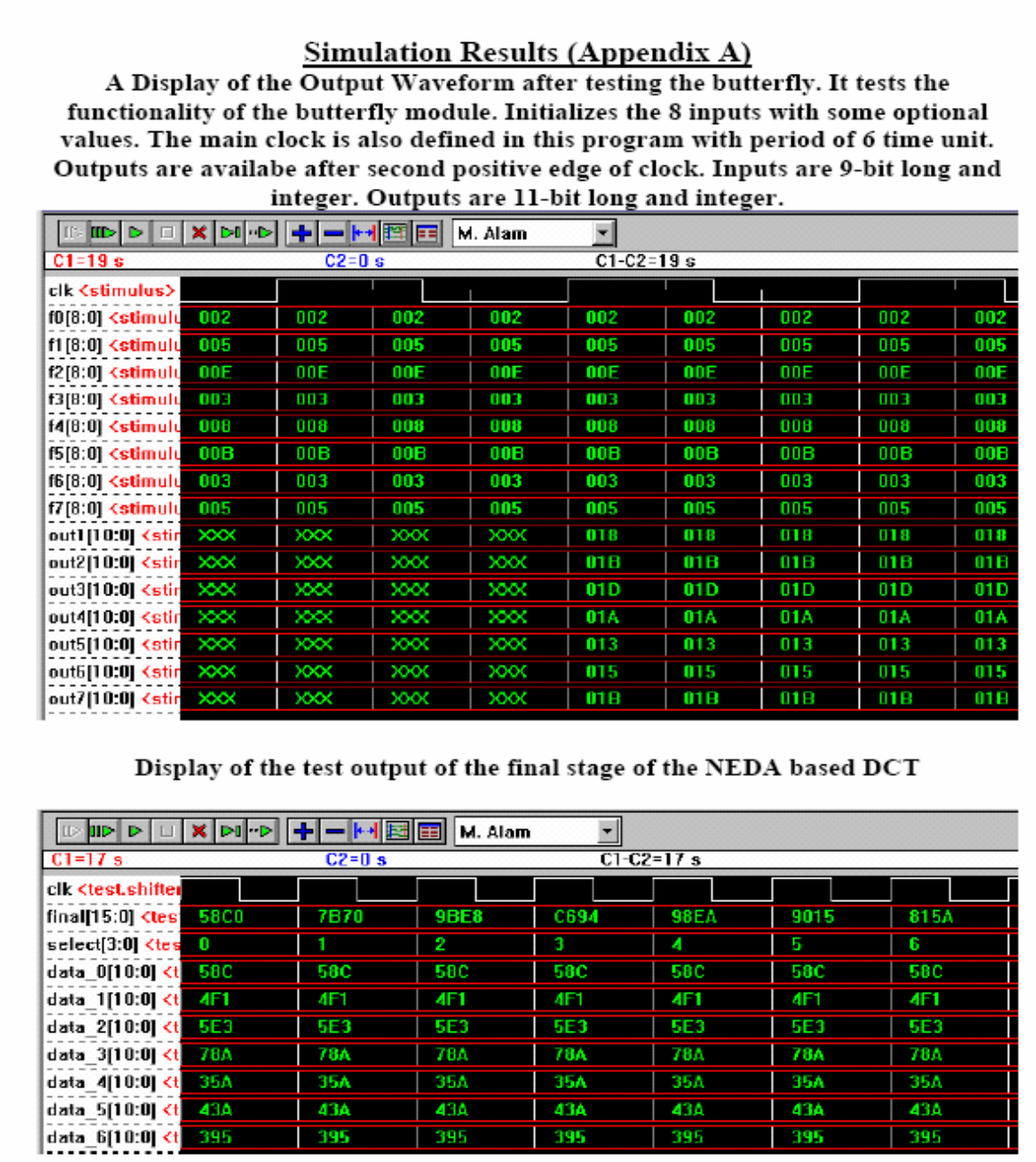


Figure 202

10.15.7 Results of Performance & Resource Estimation

- Using NEDA, great reduction in hardware and power consumption and we have less number of adders and also free from multiplication and subtraction operations.

- We can use two 1-D DCT modules to implement 2-D DCT and this is the easiest approach to implement low power fast 2D-DCT core.

- Reproduction of image highly depend upon AC component of IDCT matrix, so if we need more closer reproduction of image then include more AC components and select quantization matrix accordingly.

--`,,```,,,,````-`-,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2009 – All rights reserved
Not for Resale

- This architecture can be synthesized on FPGA using Xilinx FPGA Spartan-II family chipset.

10.15.8 API calls from reference software

In the source code file “text_code_mb.c”, the file beginning is modified with the following additions:

```
/* added for part 9 work R. Turney, Xilinx Research Labs */
#include "wcdefs.h"
#include "wc_shared.h"
#include "myhost.h"
//#define _Hang  /*Turn on the BlockDCT(), Added by ShihHao Wang, NCTU*/
#define _Hardware_DCT /*use hardware acceleration, Added by Tamer S. Mohamed */
```

Then the function implementation for Block DCT; in the same source code file, is modified by adding a compiler directive on the function body and adding a function call that invokes the wildcard II. This function is defined in two files that are added to the project of the optimized software: “myhost.h” and “myhost.c”

```
Int BlockDCT(int in[][8],int *coeff)
#if defined(_Hardware_DCT)
{
    WC_p_Main(0,in,coeff,0); /* This function contains only this line! */
    return 0;
}
#else
{
...
}
```

10.15.9 Conformance Testing

The accuracy measurement requirement dictates less than a value 2 difference between the floating and fixed-point implementations. Analysis of the above results reveals less than a 0.8750 difference. However, in order to fulfill the complete accuracy requirement it is recommended that the HDL reference architecture fulfill the accuracy requirements given in IEEE 1180-1990, in addition to those mentioned in ISO/IEC 14496-2:2001(E).

In a separate c-based test-bench, the software implementation of BlockDCT was compared to the hardware invocation with several samples of random data. The output from both the software and the hardware implementations was identical. Then two versions of the overall encoder project were tested; one with the software only implementation and another with the hardware-implemented DCT. The encoded bit stream was compared and was found to be identical. We thus conclude that the implementation successfully conforms to the requirement.

10.15.10 Limitations

The main limitation found in this implementation was the overhead associated with the call for BlockDCT function. The hardware call augments the normal overhead associated with the software function call, it augments it with the overhead of invoking the driver of the hardware accelerator and further operating system associated operation and data transfers using the main PCI bus with limited clock frequency compared to the software only case which relies on bus speeds that are typically much higher than the PCI bus speed (33MHz-50MHz) in a modern PC system.

© ISO/IEC 2009 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.16 SHAPE CODING BINARY MOTION ESTIMATION HARDWARE ACCELERATION MODULE

10.16.1 Abstract description of the module

This subclause describes an efficient implementation of a binary motion estimation module for MPEG-4 binary shape coding. The principal benefit of the proposed design is the reduction of computation complexity through the use of an innovative binary SAD cancellation architecture. Moreover, when combined with the use of run length coded binary pixel addressing and reformulated SAD calculation further operations are eliminated since only relevant data is processed. Overall this leads to throughput improvements and dynamic power savings. Static power is indirectly reduced since less area is required compared to conventional binary motion estimation implementations.

10.16.2 Module specification

10.16.2.1 MPEG 4 part: ***2 (Video)***
10.16.2.2 Profile : ***Core and above***
10.16.2.3 Level addressed: ***L1, L2***
10.16.2.4 Module Name: ***BME_4xPE***
10.16.2.5 Module latency: **Source data dependant**
Minimum: 3133 cycles
Maximum: 35869 cycles
(Search window of -15/+16)
10.16.2.6 Module data throughput: **Source data dependant**
Minimum ~ 2.8kbps (@115MHz)
Maximum ~ 32kbps (@115MHz)
10.16.2.7 Max clock frequency: ***115Mhz***
10.16.2.8 Resource usage:
10.16.2.8.1 CLB Slices: **808**
10.16.2.8.2 Block RAMs: **0**
10.16.2.8.3 Multipliers: **0**
10.16.2.8.4 External memory: ***VOP search window memory***
10.16.2.8.5 Other metrics **Equivalent Gate Count 28965**
10.16.2.9 Revision: ***2.0***
10.16.2.10 Authors: ***Daniel Larkin, Andrew Kinane***
10.16.2.11 Creation Date: ***20 August 2004***
10.16.2.12 Modification Date: ***29 March 2006***

10.16.3 Introduction

In general, with binary valued alpha pixels the SAD formula for a 16 X 16 pixel macroblock is:

$$SAD(B_{curr}, B_{ref}) = \sum_{i=1}^{16}\sum_{j=1}^{16}\left|\mathrm{B_{curr}(i,j)\ XOR\ B_{ref}\ (i,j)}\right|$$

Equation 72

Where B_{curr} is the block under consideration in the current BAP and B_{ref} is the block at the current search location in the search BAP. Due to the binary valued nature of the source data inherent redundancies can be exploited to improved throughput and power consumption. The motion within the BABs exhibits a high degree of non-uniformity. By employing early termination techniques the processing overhead can be reduced. Early SAD termination means that in certain block matches it is possible to cancel all further operations for that block match because the partial SAD result accumulated so far is larger than the minimum SAD found so far within the search window. Further processing of that particular reference BAB will only make the SAD result larger. Therefore if the partial SAD result is greater than the minimum SAD, then the final SAD result will also be greater than the minimum. Exploiting this fact allows processing to terminate early and the search strategy to move onto the next candidate block.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

A further characteristic that can be exploited becomes apparent by observing that there are unnecessary memory accesses and operations when both B_{curr} and B_{ref} pixels have the same value. This happens because the XOR in Equation 1 gives a zero result when both $B_{curr}(i,j)$ and $B_{ref}(i,j)$ have the same value. To exploit this we propose using run length encoding (RLE), thereby accessing only relevant data. However to use the RLE the SAD calculation must be reformulated, this reformulation is described in detail in [42] and simplifies to the following:

$$SAD = TOTref - TOTcurr + 2\sum DIFFcurr_{rle}$$

Equation 73

This is beneficial from a hardware and low power perspective because:

- TOT_{curr} is calculated only once per search
- TOT_{ref} can be updated in one clock cycle, after initial calculation
- Incremental addition of $DIFF_{curr}$ allows
  - Early Termination if the current minimum SAD is exceeded
  - Not Accessing Irrelevant Data

The run length code is generated for the current block, during the first match when SAD cancellation is not possible. In situations where it is beneficial to use the locations of the black pixels rather than the white pixels, an alternative form of equation 2 is available which uses an inverse version of the run length codes. This inverse run length SAD calculation derivation and the facility to use a further cancellation (TOT_{ref} underflow) method are described in detail in [42].

10.16.4 Functional Description

10.16.4.1 I/O Diagram

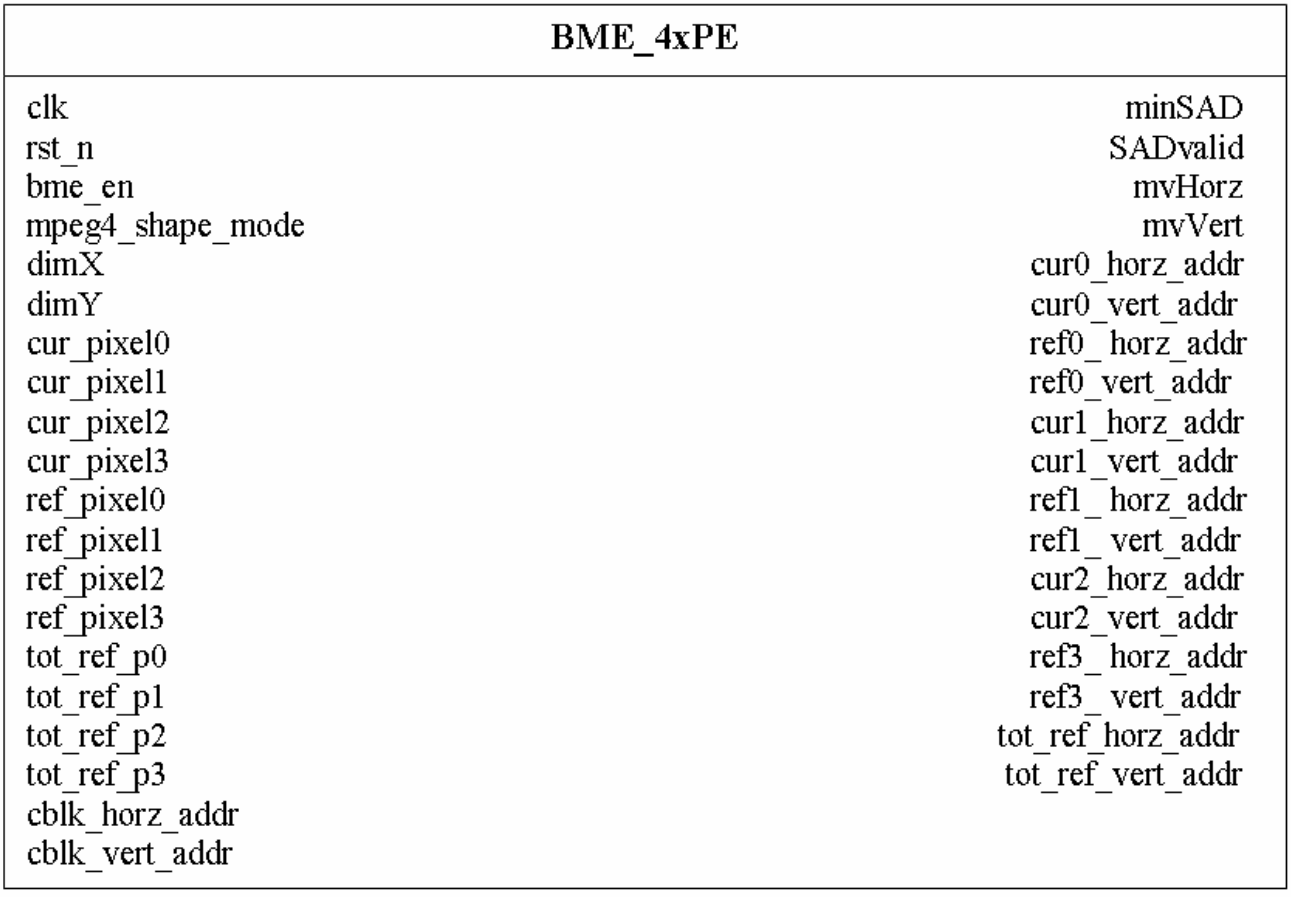


Figure 203

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.16.4.2 I/O Ports Description

Table 59

Port Name	Port Width	Direction	Description
clk	1	Input	System clock
rst_n	1	Input	System async reset
bme_en	1	Input	BME module enable
mpeg4_shape_mode	1	Input	Control flag to indicate only a single block should be carried out
dimX	ADR_BUS_SIZE	Input	Word length of Frame/VOP horizontal dimension
dimY	ADR_BUS_SIZE	Input	Word length of Frame/VOP vertical dimension
cur_pixel0	1	Input	Pixel value addressed from current block
ref_pixel0	1	Input	Pixel value addressed from reference block
cur_pixel1	1	Input	Pixel value addressed from current block
ref_pixel1	1	Input	Pixel value addressed from reference block
cur_pixel2	1	Input	Pixel value addressed from current block
ref_pixel2	1	Input	Pixel value addressed from reference block
cur_pixel3	1	Input	Pixel value addressed from current block
ref_pixel3	1	Input	Pixel value addressed from reference block
tot_ref_p0	1	Input	Pre-fetched pixel value from the reference BAB, which is used to update TOTref for PE0
tot_ref_p1	1	Input	Pre-fetched pixel value from the reference BAB, which is used to update TOTref for PE1
tot_ref_p2	1	Input	Pre-fetched pixel value from the reference BAB, which is used to update TOTref for PE2
tot_ref_p3	1	Input	Pre-fetched pixel value from the reference BAB, which is used to update TOTref for PE3
cblk_horz_addr	ADR_BUS_SIZE	Input	Horizontal address of the first pixel in the current block, used in MPEG4 shape coding mode.
cblk_vert_addr	ADR_BUS_SIZE	Input	Vertical address of the first pixel in the current block, used in MPEG4 shape coding mode.
minSAD	7	Output	Minimum calculated SAD value for the processed search window
SADvalid	1	Output	Handshake signal to indicate minimum SAD is valid for reading
mvHorz	5	Output	Horizontal Motion Vector Associated with the minimum SAD
mvVert	5	Output	Vertical Motion Vector Associated with the minimum SAD
cur0_horz_addr	ADR_BUS_SIZE	Output	Horizontal address of pixel in the current alpha block for PE0
cur0_vert_addr	ADR_BUS_SIZE	Output	Vertical address of pixel in the current alpha block for PE0

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

cur1_horz_addr	ADR_BUS_SIZE	Output	Horizontal address of pixel in the current alpha block for PE1
cur1_vert_addr	ADR_BUS_SIZE	Output	Vertical address of pixel in the current alpha block for PE1
cur2_horz_addr	ADR_BUS_SIZE	Output	Horizontal address of pixel in the current alpha block for PE2
cur2_vert_addr	ADR_BUS_SIZE	Output	Vertical address of pixel in the current alpha block for PE2
cur3_horz_addr	ADR_BUS_SIZE	Output	Horizontal address of pixel in the current alpha block for PE3
cur3_vert_addr	ADR_BUS_SIZE	Output	Vertical address of pixel in the current alpha block for PE3
ref0_horz_addr	ADR_BUS_SIZE	Output	Horizontal address of pixel in the reference alpha block for PE0
ref0_vert_addr	ADR_BUS_SIZE	Output	Vertical address of pixel in the reference alpha block for PE0
ref1_horz_addr	ADR_BUS_SIZE	Output	Horizontal address of pixel in the reference alpha block for PE1
ref1_vert_addr	ADR_BUS_SIZE	Output	Vertical address of pixel in the reference alpha block for PE1
ref2_horz_addr	ADR_BUS_SIZE	Output	Horizontal address of pixel in the reference alpha block for PE2
ref2_vert_addr	ADR_BUS_SIZE	Output	Vertical address of pixel in the reference alpha block for PE2
ref3_horz_addr	ADR_BUS_SIZE	Output	Horizontal address of pixel in the reference alpha block for PE3
ref3_vert_addr	ADR_BUS_SIZE	Output	Vertical address of pixel in the reference alpha block for PE3
tot_ref_horz_addr	ADR_BUS_SIZE	Output	Horizontal address for pre-fetched pixel for the TOTref update
tot_ref_vert_addr	ADR_BUS_SIZE	Output	Vertical address for pre-fetched pixel for the TOTref update.

10.16.4.3 Parameters (Generic)

Table 60

Parameter Name	Type	Range	Description
ADR_BUS_SIZE	INT	11	Word length of horizontal and vertical BAP pointers
DIM_WIDTH	INT	10	Word length for horizontal dimension of BAP/frame
DIM_HEIGHT	INT	10	Word length for vertical dimension of BAP/frame
MAX_SRCH_WINDOW	INT	16	Search Window
MAX_HORZ_BLK_SIZE	INT	16	Horizontal Alpha block size
MAX_VERT_BLK_SIZE	INT	16	Vertical Alpha block size

© ISO/IEC 2009 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.16.4.4 Parameters (Constants)

N/A.

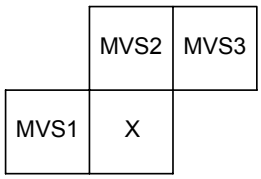
10.16.5 Algorithm

A comprehensive review of binary shape coding in MPEG-4 is presented in [43]. It is generally accepted that motion estimation for shape is the most computationally intensive block within binary shape encoding [44]. Approximately 90% of the resources required in a shape encoder are consumed by binary motion estimation (BME). This is our motivation for accelerating this block.

Motion estimation for shape differs somewhat from conventional texture motion estimation. Firstly a motion vector predictor for shape (MVPS) is found by examining neighbouring shape and texture macroblocks. The first valid motion vector in the sequence [MVS1, MVS2, MVS3, MV1, MV2, MV3] is chosen as the predictor (where MVSx is the motion vector for shape and MVx is the texture motion vector). The position of these candidate motion vector predictors is depicted in Figure 204. A BAB is considered to have an invalid MVS if the BAB is transparent or is an intra block. In addition the MV of a texture macroblock is invalid if the macroblock is transparent, the current VOP is a B-VOP or if the current video object has binary information only and no texture information. If no neighbouring vector is valid the MVPS is set to zero. Once the MVPS motion compensated (MC) BAB is retrieved it is compared against the current macroblock. If the error between each 4x4 subblock of the MVPS MC BAB and the current BAB is less than a predefined threshold (AlphaTH), the motion vector predictor can be used directly. If the MVPS MC BAB error is not less than the threshold a motion vector for shape (MVS) is required. If MVS is required, it proceeds in a conventional fashion with a search window usually of +/- 16 pixels around the MVPS macroblock using any search strategy. This aspect is in contrast to texture motion estimation where the search is around the co-located macroblock in the reference VOP. At each candidate BME search position a distortion metric is evaluated. Typically the sum of absolute differences (SAD) is used due to its optimum trade off between complexity and efficiency. Once the minimum SAD is located in the search window a final motion vector difference for Shape (MVDS) is calculated as follows:

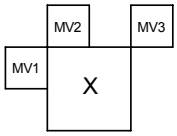
MVDS = MVS − MVPS.

Using prior Shape Motion Vectors



X= Colocated position of the current BAB in the reference Binary Alpha Plane.

Using prior Texture Motion Vectors



X= Colocated position of the current BAB in the reference texture VOP.

Figure 204 — Position of candidate MVPS.

10.16.6 Implementation

The architecture can be implemented with varying degrees of parallelism using differing number of processing elements. The current implementation uses 4 PEs, and the blocks are described in the following subsections.

10.16.6.1 BME_CTRL

The BME module is configurable to operate in a number of different ways including with/without SAD cancellation, with/without run length coding and with/without TOTref underflow cancellation. It is the function of the BME_CTRL block to send the necessary control signals to PAGU_NXPE, bme_sad_NxPE and Update blocks and monitor the status of these blocks.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

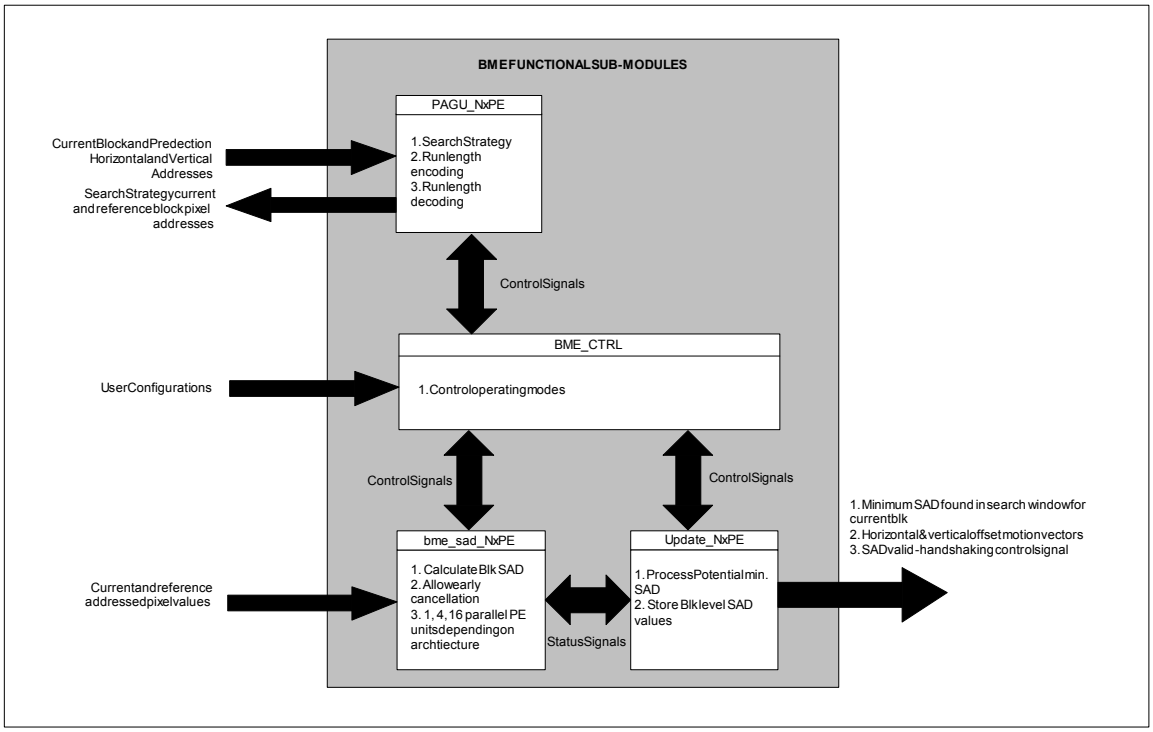


Figure 205 — Functional sub modules within the BME.

10.16.6.2 PAGU_NxPE

The Pixel Address Generation Unit has the following basic functionality:

- During the first block match, run length encoding is generated from the pixels within the current block.
- Block match addresses are generate from the Search strategy sub module.
- If applicable run length code pairs are fetched from memory and decoding occurs

10.16.6.3 BME_SAD_NxPE

The basic functionality of the *bme_sad_NxPE* module is to calculate the SAD between two alpha blocks. This calculation can proceed on a pixel by pixel basis or can use run length coding to access only those pixels, which contribute to the actual final SAD value.

Figure 206 shows a detailed view of the SAD Processing Element. At the first clock cycle the minimum SAD encountered so far is loaded into DACC_REG. During the next cycle TOTcurr / TOTref is added to DACC REG (depending if TOTref [MSB] is 0 or 1 respectively). On the next clock cycle DACC_REG is de-accumulated by TOTref / TOTcurr again depending on whether TOTref [MSB] is 0 or 1 respectively. If a sign change occurs at this point the minimum SAD has already been exceeded and no further processing is required. If a sign change has not occurred the PAGU retrieves the next run length code from memory. If TOTcurr[msb] = 0 the run length pair code is processed unmodified. On the other hand if TOTref [msb] = 1 the inverse run length code is processed. In either case the run length code processing results in an X, Y macroblock address. The X,Y address is used to retrieve the relevant pixel from the reference BAB and the current BAB. The pixel values are XORed and the result is left shifted by one place and then subtracted from the DACC_REG. If a sign change occurs, early termination is possible. If not the remaining pixels in the current run length code are processed. If the SAD calculation is not cancelled, subsequent run length codes for the current BAB are fetched from memory and the processing repeats.

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

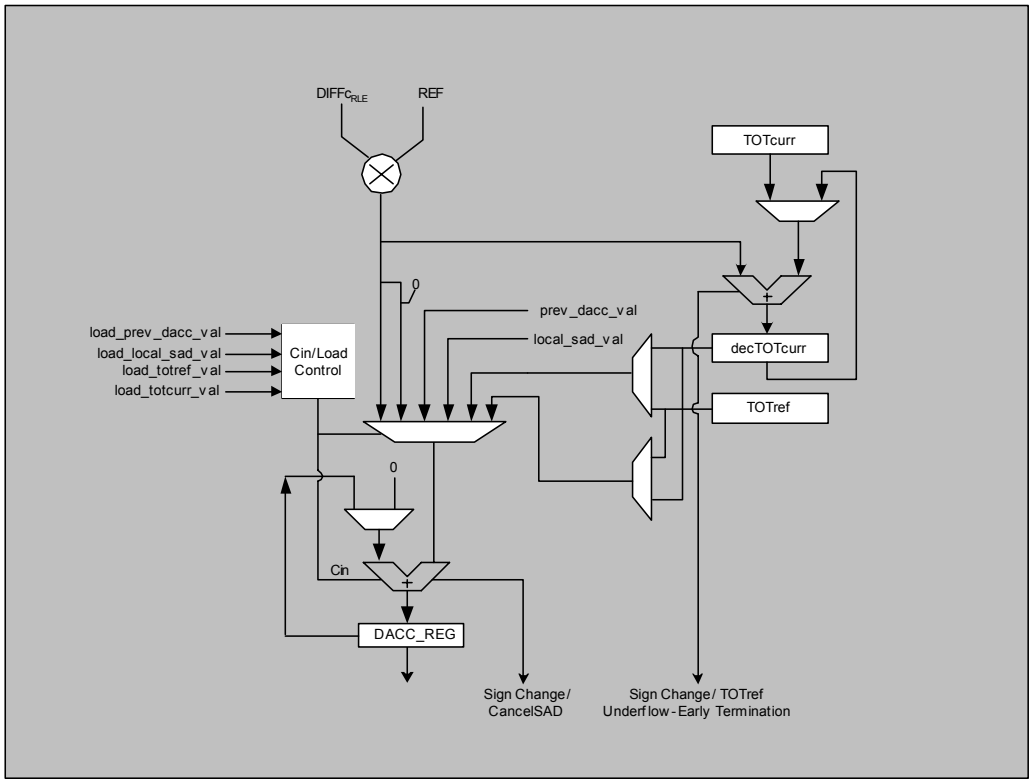


Figure 206 — Run length Binary SAD PE.

10.16.6.4 Update_NxPE

When SAD cancellation does not occur it is necessary to examine the PE SAD values and see if a new minimum SAD has been found. Since PE SAD calculation can take up to TOTcurr/Inverse TOTcurr + 2 steps to complete, it is possible to run the update stage in parallel with a new block match. For the 1xPE architecture the update logic is trivial.

Figure 207 shows the structure of the 4xPE update logic. Sequential type processing is adopted, which takes at most 11 cycles to complete. Each PE sad value is accumulated in the $TOTAL_DACC_REG$, if after this the value is positive a new block level minimum SAD has been found. The block level minimum SAD levels must now be updated. In the 16xPE architecture to prevent excessive stalling the sequential update is replaced by an adder tree structure.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

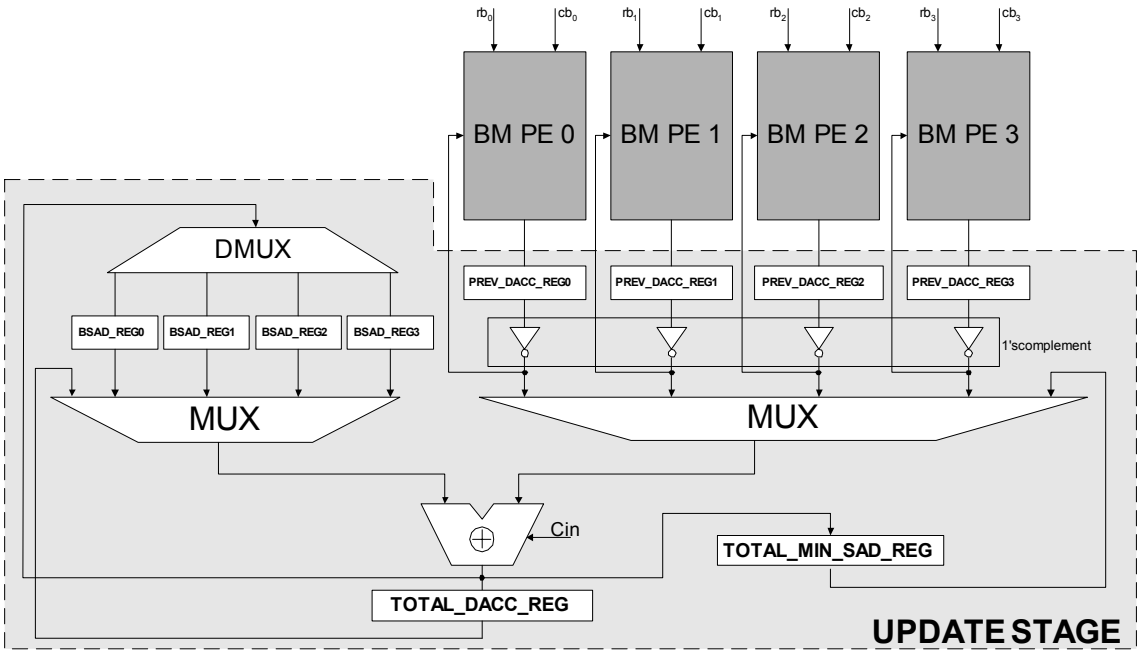


Figure 207 — BME 4xPE Update Logic.

10.16.6.5 Interfaces

Figure 208 shows a timing diagram describing the relationships between the input and output ports. The BAP/frame dimensions along with MPEG4_SHAPE_MODE should be set up prior to enabling the module via BME_EN. One clock cycle after BME_EN goes high, valid output addresses are generated. The system should respond to these requests by providing the appropriate pixel values on the subsequent clock cycle. When SADvalid pulses high all block matches in the search window have been completed, therefore minimum SAD (minSAD) and the associated motion vectors (mvHorz & mvVert) can be read.

10.16.6.6 Timing Diagrams

Figure 208 shows a timing diagram describing the relationships between input and output ports.

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

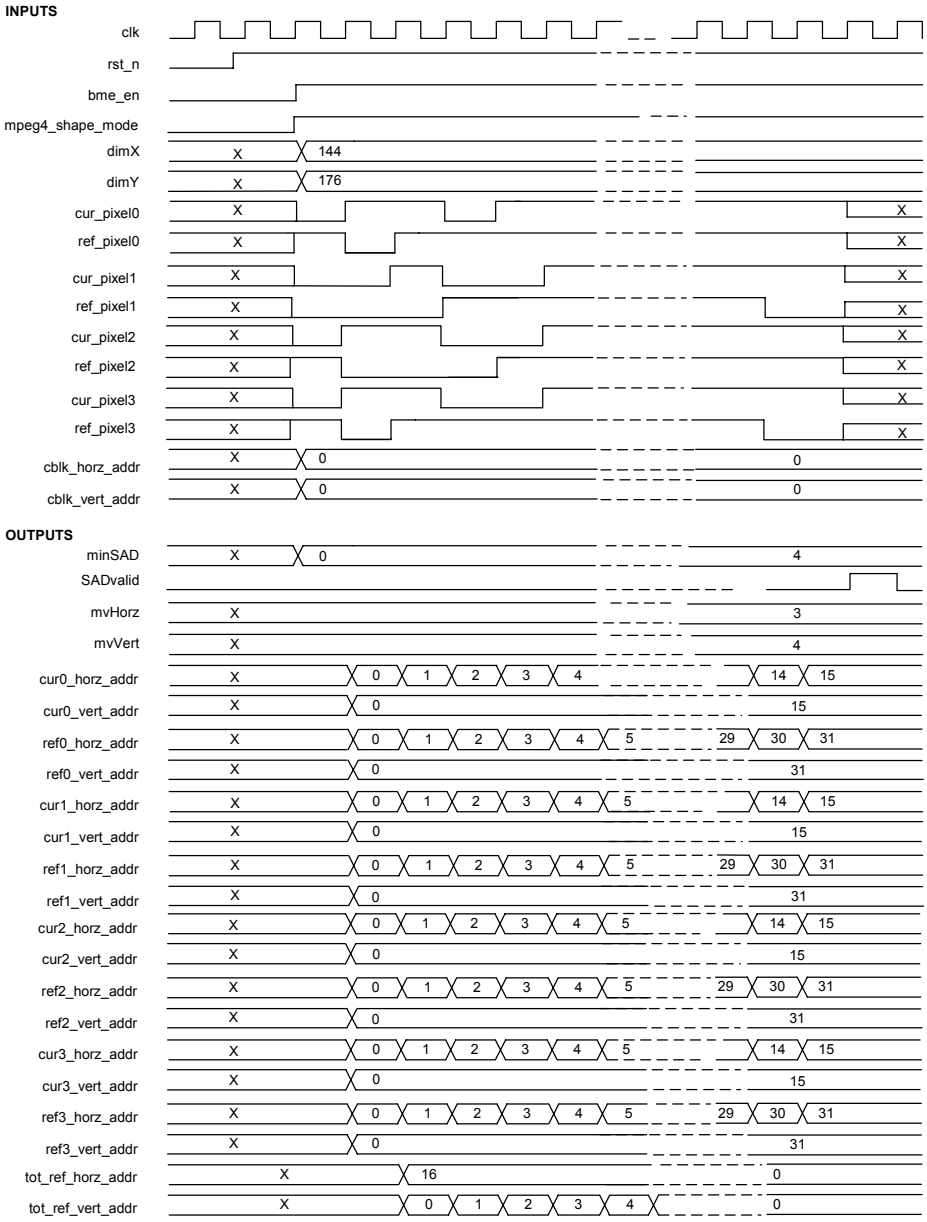


Figure 208 — BME Input & Output Timing Diagram.

10.16.6.7 Register File Access

Four 32-bit master socket registers are programmed directly by the host software to configure parameters for the BME_4xPE core. These registers are used to configure the hardware module controller.

Register Name	Range	Description
dControl[0][31:0]	[0][9:0]	Frame width
	[0][19:10]	Frame Height
	[0][23:20]	Number of frames that are read/written at a time
	[0][31:0]	Undefined
dControl[1][31:0]	[1][20:0]	SRAM read start address
	[1][31:21]	Undefined

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

dControl[2][31:0]	[2][20:0]	SRAM write start address
	[2][31:21]	Undefined
dControl[3][31:0]	[3][31:0]	Configure hardware module controller for debug mode or normal mode. Debug mode allows host software to monitor the hardware module controller state machine

The host software programs these registers after the frame data has been written to the SRAM. They are written by using a WildCard API function WC_PeRegWrite. This function writes each of the four 32-bit values into the hardware register file configuration registers at a specific offset according to the specific hardware accelerator being strobed. The abridged code listing in 10.16.8 shows how the API function is called.

10.16.7 Results of Performance & Resource Estimation

The module has been integrated with the University of Calgary's integration framework on the Annapolis WildCard FPGA prototyping platform with associated host calling software. The IP core has been implemented using Verilog RTL and verified using ModelSim SE v6.0a. The Verilog RTL was synthesised using Synplicity Pro 7.5, followed by place and route using Xilinx ISE (version 6.2.03i). The synthesis and place & route scripts were adapted from those proposed by the University of Calgary. To obtain resource usage information for the IP core itself a synthesis run was carried out with the IP core only without the surrounding integration framework and pin assignments (since the IP core is not connected to any FPGA pins directly). An abridged version of the mapping report is given in the following code listing:

```
Release 6.3.03i Map G.38
Xilinx Mapping Report File for Design 'bme_4xPE_top'

Design Information
------------------
Command Line   : C:/Xilinx/bin/nt/map.exe -intstyle ise -p XC2V3000-BG728-6 -cm
area -pr b -k 4 -c 100 -tx off -o bme_4xPE_top_map.ncd bme_4xPE_top.ngd
bme_4xPE_top.pcf
Target Device  : x2v3000
Target Package : bg728
Target Speed   : -6
Mapper Version : virtex2 -- $Revision: 1.16.8.2 $
Mapped Date    : Fri Mar 31 13:19:47 2006


Design Summary
--------------
Number of errors:      0
Number of warnings:    0
Logic Utilization:
  Number of Slice Flip Flops:         586 out of  28,672    2%
  Number of 4 input LUTs:           1,215 out of  28,672    4%
Logic Distribution:
  Number of occupied Slices:          808 out of  14,336    5%
  Number of Slices containing only related logic:     808 out of     808  100%
  Number of Slices containing unrelated logic:          0 out of     808    0%
        *See NOTES below for an explanation of the effects of unrelated logic
Total Number 4 input LUTs:          1,409 out of  28,672    4%
  Number used as logic:             1,215
  Number used as a route-thru:         82
  Number used for 32x1 RAMs:          112
    (Two LUTs used per 32x1 RAM)

  Number of bonded IOBs:              173 out of     516   33%
  Number of GCLKs:                      1 out of      16    6%
```

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2009 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
Total equivalent gate count for design:  28,965
Additional JTAG gate count for IOBs:  8,304
Peak Memory Usage:  123 MB

Section 13 - Additional Device Resource Counts
-----------------------------------------------
Number of JTAG Gates for IOBs = 173
Number of Equivalent Gates for Design = 28,965
Number of RPM Macros = 0
Number of Hard Macros = 0
CAPTUREs = 0
BSCANs = 0
STARTUPs = 0
PCILOGICs = 0
DCMs = 0
GCLKs = 1
ICAPs = 0
18X18 Multipliers = 0
Block RAMs = 0
TBUFs = 0
Total Registers (Flops & Latches in Slices & IOBs) not driven by LUTs = 407
IOB Dual-Rate Flops not driven by LUTs = 0
IOB Dual-Rate Flops = 0
IOB Slave Pads = 0
IOB Master Pads = 0
IOB Latches not driven by LUTs = 0
IOB Latches = 0
IOB Flip Flops not driven by LUTs = 0
IOB Flip Flops = 0
Unbonded IOBs = 0
Bonded IOBs = 173
Total Shift Registers = 0
Static Shift Registers = 0
Dynamic Shift Registers = 0
16x1 ROMs = 0
16x1 RAMs = 0
32x1 RAMs = 56
Dual Port RAMs = 0
MUXFs = 108
MULT_ANDs = 70
4 input LUTs used as Route-Thrus = 82
4 input LUTs = 1215
Slice Latches not driven by LUTs = 0
Slice Latches = 0
Slice Flip Flops not driven by LUTs = 407
Slice Flip Flops = 586
Slices = 808
Number of LUT signals with 4 loads = 17
Number of LUT signals with 3 loads = 35
Number of LUT signals with 2 loads = 338
Number of LUT signals with 1 load = 692
NGM Average fanout of LUT = 2.41
NGM Maximum fanout of LUT = 56
NGM Average fanin for LUT = 3.0831
Number of LUT symbols = 1215
Number of IPAD symbols = 27
Number of IBUF symbols = 27
```


© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.16.8 API calls from reference software

The hardware acceleration framework with the integrated BME_4xPE IP core has been integrated with Microsoft MPEG-4 Part 7 Optimised Video Reference Software (version microsoft-v2.4-030710-NTU). As can be seen from the following code listing, the hardware accelerator for BME is called in file *shenc.cpp* in the class *CvideoObjectEncoder*. Based on a pre-processor directive, the function DCU_BME_4xPE_HWA is called and this looks after initiating the appropriate protocols with the BME_4xPE hardware accelerator on the WildCard-II FPGA.

```
Int CVideoObjectEncoder::codeInterShape (PixelC* ppxlcSrcFrm,
                      CVOPU8YUVBA* pvopcRefQ,
                      CMBMode* pmbmd,
                      const ShapeMode& shpmdColocatedMB,
                          const CMotionVector* pmv,
                      CMotionVector* pmvBY,
                      CoordI iX, CoordI iY,
                      Int iMBX, Int iMBY)
{
.
.
.
if(m_volmd.volType == ENHN_LAYER && m_volmd.bSpatialScalability == 1 &&       m_volmd.iHierarchyType
== 0 && m_volmd.iEnhnType != 0 && m_volmd.iuseRefShape == 1 && m_vopmd.vopPredType == PVOP)
{
    mvShapeMVP.setToZero();
    memset((void *)m_puciPredBAB->pixels(),255, MC_BAB_SIZE*MC_BAB_SIZE);
}
else
{
#ifdef _DCU_BME_4xPE_HWA_
    DCU_BME_4xPE_HWA(pvopcRefQ,&mvShape, mvShapeMVP.trueMVHalfPel (), iX, iY);
#else
    blkmatchForShape(pvopcRefQ,&mvShape, mvShapeMVP.trueMVHalfPel (), iX, iY);
#endif
    assert (mvShape.iMVX != NOT_MV && mvShape.iMVY != NOT_MV);
    *pmvBY = mvShape;
    mvBYD = mvShape - mvShapeMVP;

       .
       .
}
.
.
.
}
```

10.16.9 Conformance Testing

10.16.9.1 Reference software type, version and input data set

The hardware acceleration framework with the integrated BME_4xPE core has been integrated with Microsoft MPEG-4 Part 7 Optimised Video Reference Software (version microsoft-v2.4-030710-NTU). The parameter file used to configure the encoder is shown below (Source.FilePrefix changes depending on the test sequence name).

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
Version = 904                       // parameter file version

// When VTC is enabled, the VTC parameter file is used instead of this one.
VTC.Enable = 0
VTC.Filename = ""

VersionID[0] = 2             // object stream version number (1 or 2)

Source.Width = 352
Source.Height = 288
Source.FirstFrame = 0
Source.LastFrame = 299
Source.ObjectIndex.First = 255
Source.ObjectIndex.Last = 255
Source.FilePrefix = "nh_cg1_cif"
Source.Directory = "."
Source.BitsPerPel = 8
Source.Format [0] = "420"            // One of "444", "422", "420"
Source.FrameRate [0] = 10
Source.SamplingRate [0] = 1

Output.Directory.Bitstream = ".\cmp"
Output.Directory.DecodedFrames = ".\rec"

Not8Bit.Enable = 0
Not8Bit.QuantPrecision = 5

RateControl.Type [0] = "None"         // One of "None", "MP4", "TM5"
RateControl.BitsPerSecond [0] = 50000

Scalability [0] = "None"          // One of "None", "Temporal", "Spatial"
Scalability.Temporal.PredictionType [0] = 0       // Range 0 to 4
Scalability.Temporal.EnhancementType [0] = "Full" // One of "Full", "PartC", "PartNC"
Scalability.Spatial.EnhancementType [0] = "PartC" // One of "Full", "PartC", "PartNC"
Scalability.Spatial.PredictionType [0] = "PBB"    // One of "PPP", "PBB"
Scalability.Spatial.Width [0] = 352
Scalability.Spatial.Height [0] = 288
Scalability.Spatial.HorizFactor.N [0] = 2 // upsampling factor N/M
Scalability.Spatial.HorizFactor.M [0] = 1
Scalability.Spatial.VertFactor.N [0] = 2  // upsampling factor N/M
Scalability.Spatial.VertFactor.M [0] = 1
Scalability.Spatial.UseRefShape.Enable [0] = 0
Scalability.Spatial.UseRefTexture.Enable [0] = 0
Scalability.Spatial.Shape.HorizFactor.N [0] = 2   // upsampling factor N/M
Scalability.Spatial.Shape.HorizFactor.M [0] = 1
Scalability.Spatial.Shape.VertFactor.N [0] = 2    // upsampling factor N/M
Scalability.Spatial.Shape.VertFactor.M [0] = 1

Quant.Type [0] = "H263"               // One of "H263", "MPEG"

GOV.Enable [0] = 0
GOV.Period [0] = 0                // Number of VOPs between GOV headers

Alpha.Type [0] = "Binary"             // One of "None", "Binary", "Gray", "ShapeOnly"
Alpha.MAC.Enable [0] = 0
Alpha.ShapeExtension [0] = 0          // MAC type code
Alpha.Binary.RoundingThreshold [0] = 0
Alpha.Binary.SizeConversion.Enable [0] = 0
Alpha.QuantStep.IVOP [0] = 16
Alpha.QuantStep.PVOP [0] = 16
```


--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
Alpha.QuantStep.BVOP [0] = 16
Alpha.QuantDecouple.Enable [0] = 0
Alpha.QuantMatrix.Intra.Enable [0] = 0
Alpha.QuantMatrix.Intra [0] = {}      // { insert 64 comma-separated values }
Alpha.QuantMatrix.Inter.Enable [0] = 0
Alpha.QuantMatrix.Inter [0] = {}      // { insert 64 comma-separated values }

Texture.IntraDCThreshold [0] = 0      // See note at top of file
Texture.QuantStep.IVOP [0] = 16
Texture.QuantStep.PVOP [0] = 16
Texture.QuantStep.BVOP [0] = 16
Texture.QuantMatrix.Intra.Enable [0] = 0
Texture.QuantMatrix.Intra [0] = {}        // { insert 64 comma-separated values }
Texture.QuantMatrix.Inter.Enable [0] = 0
Texture.QuantMatrix.Inter [0] = {}        // { insert 64 comma-separated values }
Texture.SADCT.Enable [0] = 1

Motion.RoundingControl.Enable [0] = 1
Motion.RoundingControl.StartValue [0] = 0
Motion.PBetweenICount [0] = -1
Motion.BBetweenPCount [0] = 2
Motion.SearchRange [0] = 16
Motion.SearchRange.DirectMode [0] = 2     // half-pel units
Motion.AdvancedPrediction.Enable [0] = 0
Motion.SkippedMB.Enable [0] = 1
Motion.UseSourceForME.Enable [0] = 1
Motion.DeblockingFilter.Enable [0] = 0
Motion.Interlaced.Enable [0] = 0
Motion.Interlaced.TopFieldFirst.Enable [0] = 0
Motion.Interlaced.AlternativeScan.Enable [0] = 0
Motion.ReadWriteMVs [0] = "Off"           // One of "Off", "Read", "Write"
Motion.ReadWriteMVs.Filename [0] = "MyMVFile.dat"
Motion.QuarterSample.Enable [0] = 0

Trace.CreateFile.Enable [0] = 1
Trace.DetailedDump.Enable [0] = 1

Sprite.Type [0] = "None"          // One of "None", "Static", "GMC"
Sprite.WarpAccuracy [0] = "1/2"           // One of "1/2", "1/4", "1/8", "1/16"
Sprite.Directory = "\\swinder1\sprite\brea\spt"
Sprite.Points [0] = 0                 // 0 to 4, or 0 to 3 for GMC
Sprite.Points.Directory = "\\swinder1\sprite\brea\pnt"
Sprite.Mode [0] = "Basic"             // One of "Basic", "LowLatency", "PieceObject", "PieceUpdate"

ErrorResil.RVLC.Enable [0] = 0
ErrorResil.DataPartition.Enable [0] = 0
ErrorResil.VideoPacket.Enable [0] = 0
ErrorResil.VideoPacket.Length [0] = 0
ErrorResil.AlphaRefreshRate [0] = 1

Newpred.Enable [0] = 0
Newpred.SegmentType [0] = "VideoPacket"   // One of "VideoPacket", "VOP"
Newpred.Filename [0] = "example.ref"
Newpred.SliceList [0] = "0"

RRVMode.Enable [0] = 0                // Reduced resolution VOP mode
RRVMode.Cycle [0] = 0

Complexity.Enable [0] = 1             // Global enable flag
Complexity.EstimationMethod [0] = 1       // 0 or 1
```

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
Complexity.Opaque.Enable [0] = 1
Complexity.Transparent.Enable [0] = 1
Complexity.IntraCAE.Enable [0] = 1
Complexity.InterCAE.Enable [0] = 1
Complexity.NoUpdate.Enable [0] = 1
Complexity.UpSampling.Enable [0] = 1
Complexity.IntraBlocks.Enable [0] = 1
Complexity.InterBlocks.Enable [0] = 1
Complexity.Inter4VBlocks.Enable [0] = 1
Complexity.NotCodedBlocks.Enable [0] = 1
Complexity.DCTCoefs.Enable [0] = 1
Complexity.DCTLines.Enable [0] = 1
Complexity.VLCSymbols.Enable [0] = 1
Complexity.VLCBits.Enable [0] = 1
Complexity.APM.Enable [0] = 1
Complexity.NPM.Enable [0] = 1
Complexity.InterpMCQ.Enable [0] = 1
Complexity.ForwBackMCQ.Enable [0] = 1
Complexity.HalfPel2.Enable [0] = 1
Complexity.HalfPel4.Enable [0] = 1
Complexity.SADCT.Enable [0] = 1
Complexity.QuarterPel.Enable [0] = 1
        :

VOLControl.Enable [0] = 0
VOLControl.ChromaFormat [0] = 0
VOLControl.LowDelay [0] = 0
VOLControl.VBVParams.Enable [0] = 0
VOLControl.Bitrate [0] = 0           // 30 bits
VOLControl.VBVBuffer.Size [0] = 0    // 18 bits
VOLControl.VBVBuffer.Occupancy [0] = 0    // 26 bits
```

10.16.9.2 API vector conformance

At API level the test vectors used are the CIF and QCIF test sequences as defined by the MPEG-4 Video Verification Model.

10.16.9.3 End to end conformance (conformance of encoded bitstreams or decoded pictures)

The bistream generated by the software only encoder is identical to the bitstream generated by the encoder hardware accelerated by the BME_4xPE module for binary motion estimation as part of the shape encoder.

10.16.10 Limitations

N/A.

10.17 A SIMD ARCHITECTURE FOR FULL SEARCH BLOCK MATCHING ALGORITHM

10.17.1 Abstract description of the module

This subclause presents an SIMD architecture for full pixel Exhaustive Search Block Matching Algorithm (ESBMA). This module is part of the MPEG-4 Part 9: Reference Hardware Description. The developed module is prototyped and simulated using ModelSim 5.4®. It is synthesized using Synplify Pro 7.1®. The module processes 26.7 CIF frame /sec using the max clock frequency. This module utilizes 20% of the register bits, 16% of the Block RAMs, and 16% of the LUTs in Xilinx Virtex II FPGA XC2V3000-4.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.17.2 Module specification

10.17.2.1 MPEG 4 Part: *9*
10.17.2.2 Profile: *Simple profile*
10.17.2.3 Level addressed: *All*
10.17.2.4 Module Name: *ME_architecture*
10.17.2.5 Module latency: *11594 clock cycles*
10.17.2.6 Module data throughput: *10592 motion vector/sec using the max clock freq.*
10.17.2.7 Max clock frequency: *122.8 MHz*
10.17.2.8 Resource usage:
10.17.2.8.1 LUT: *4676 (16% of the available LUT in* XC2V3000-4*)*
10.17.2.8.2 Block RAMs: *16 (16% of the avail. block RAMs in* XC2V3000-4*)*
10.17.2.8.3 Multipliers: *0*
10.17.2.8.4 External memory: *0*
10.17.2.8.5 Register bits not including I/O: *5887* (20% *of the aval. register bits in* XC2V3000-4)
10.17.2.9 Revision: *1.0*
10.17.2.10 Authors: *Mohammed Sayed and Wael Badawy*
10.17.2.11 Creation Date: *April 2005*
10.17.2.12 Modification Date:

10.17.3 Introduction

Video compression techniques exploit the spatial and temporal redundancy of the video signals. Different video coding standards have been introduced to meet the different requirements of streaming video sequences, especially over low bandwidth networks. MPEG-4, as one of the latest video coding standards, is still using the block-based motion estimation and compensation coding technique due to its simplicity. In this technique the current frame is divided into non-overlapped blocks and the video motion is represented by the translation of these blocks with respect to a reference frame. The motion estimation process generates one motion vector for each block with horizontal and vertical components using the block-matching algorithm (BMA).

In BMA, a searching process is done for each block in the current frame to find the best matching block to it in a reference frame as shown in Figure 209. Then the block motion vector is estimated from the position difference between the two matched blocks. The BMA suffers from high computational cost and high memory requirements. To reduce the computational cost, the searching process is usually limited to certain search area as shown in Figure 209. Different matching criteria can be used among which the sum of absolute difference (SAD) is the most common in motion estimation architectures [27] due to its simplicity and suitability for VLSI implementation In addition, different search strategies have been proposed in the literature, such as full search, three-step search [27], and cross search [27]. Full search strategy produces high video quality but it suffers from high computational cost and high memory requirements. Equation 74 and Equation 75 show the SAD matching criterion:

$$SAD(d_1,d_2)=\sum_{(n_1,,n_2)\in B}\left|s_1(n_1,n_2,k)-s_2(n_1+d_1,n_2+d_2,k+1)\right|$$

Equation 74

$$\left[d_1\,d_2\right]^T=\arg\min_{(d_1,d_2)}SAD(d_1,d_2)$$

Equation 75

Where $s_1(n_1,n_2,k)$ is the pixel value at (n_1,n_2) in frame k and $s_2(n_1+d_1,n_2+d_2,k+1)$ is the pixel value at (n_1+d_1,n_2+d_2) in frame k+1, d_1 and d_2 are the horizontal and vertical motion vectors respectively.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

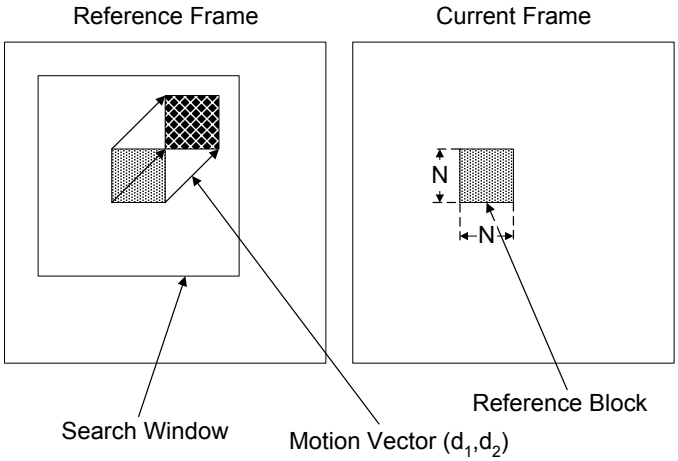


Figure 209 — Block Matching Algorithm.

10.17.4 Functional Description

10.17.4.1 Functional description details

The proposed architecture processes CIF format video sequences (i.e. 352x288 pixels frame size) with 16x16 pixels block size and ±15 pixels search range. It uses the full search block matching algorithm with sum of absolute difference SAD matching criterion.

10.17.4.2 I/O Diagram

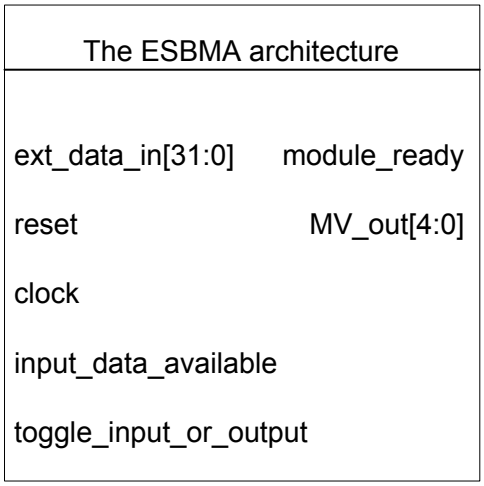


Figure 210

10.17.4.3 I/O Ports Description

Table 61

Port Name	Port width	Direction	Description
ext_data_in	32	Input	External data input
Reset	1	Input	Reset signal

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Clock	1	Input	Clock
input_data_available	1	Input	Inform the module that the input data is available
toggle_input_or_output	1	Input	Change the input type from SW to RB or change the output type from horizontal MV to vertical MV
module_ready	1	Output	Motion vector ready
MV_out	5	Output	The motion vector output

10.17.4.3.1 Parameters (generic)

Not applicable

10.17.4.3.2 Parameters (constants)

Not applicable

10.17.5 Algorithm

The proposed architecture uses the exhaustive search block matching algorithm with ±15 pixels search range and 16x16 pixels block size.

10.17.6 Implementation

The proposed architecture processes CIF format video sequences (i.e. 352x288 pixels frame size) with 16x16 pixels block size and ±15 pixels search range. The architecture consists of an embedded SRAM with 46x46 bytes size, 31 processing elements, one reference block memory, and one comparison unit as shown in Figure 211. The architecture searches for the 16x16 block with the minimum SAD in the 46x46 search window stored in the embedded SRAM and generates horizontal and vertical motion vector for that block. The generated motion vector is 10 bits width; 5 bits for the horizontal motion vector and 5 bits for the vertical one. The architecture reads the search window texture from the reference frame and the reference block texture from the current frame. Both of the reference and the current frames are stored in the external SRAM.

© ISO/IEC 2009 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

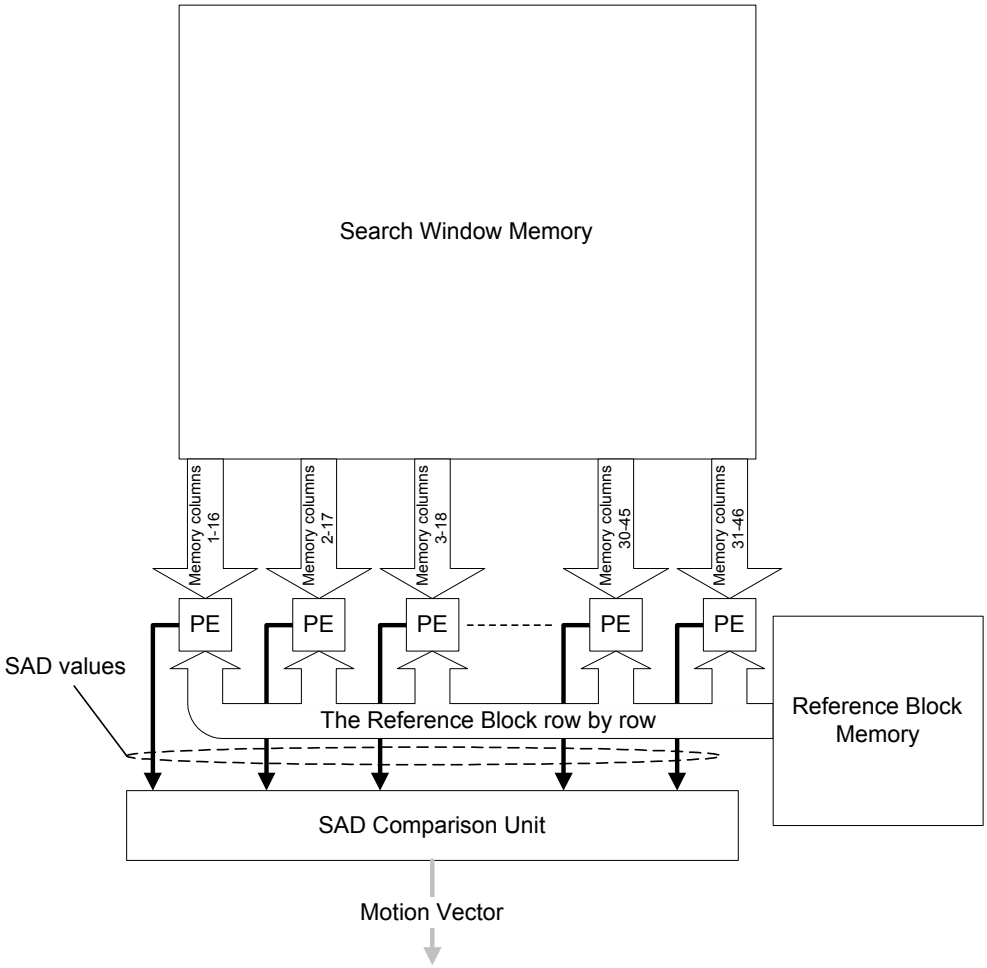


Figure 211 — Block diagram of the architecture.

The processing elements are working in parallel as single instruction multiple data (SIMD) architecture. The processing elements compute the SAD values for the candidate blocks. The SAD comparison unit finds the minimum SAD and evaluates the horizontal and vertical motion vectors. A search window of 46x46 pixels and a block of 16x16 pixels are used, which means with full-search block matching algorithm we have 961 (31x31) candidate blocks. Accessing the search window memory is done row by row. This means that the reference block is compared with all the candidate blocks in the first row simultaneously. Then it is compared with all the candidate blocks in the second row simultaneously and so on for the following rows of candidate blocks.

Table 62 — The memory columns and the processing elements connection configuration.

Processing element	The connected memory columns
PE1	1,2,3,…,16
PE2	2,3,4,…,17
PE3	3,4,5,…,18
PE4	4,5,6,…,19
PE5	5,6,7,…,20
PE6	6,7,8,…,21

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

PE7	7,8,9,…,22
PE8	8,9,10,…,23
PE9	9,10,11,…,24
PE10	10,11,12,…,25
PE11	11,12,13,…,26
PE12	12,13,14,…27
PE 13	13,14,15,…,28
PE14	14,15,16,…,29
PE15	15,16,17,…,30
PE16	16,17,18,…,31
PE17	17,18,19,…,32
PE18	18,19,20,…,33
PE19	19,20,21,…,34
PE20	20,21,22,…,35
PE21	21,22,23,…,36
PE22	22,23,24,…,37
PE23	23,24,25,…,38
PE24	24,25,26,…,39
PE25	25,26,27,…,40
PE26	26,27,28,…,41
PE27	27,28,29,…,42
PE28	28,29,30,…,43
PE29	29,30,31,…,44
PE30	30,31,32,…,45
PE31	31,32,33,…,46

The reference block is stored in a 16x16 memory. The reference block pixels are feed row by row to the processing elements as shown in Figure 211. One processing element is used per each column of candidate blocks. The 46 memory columns are connected to the 31 processing elements according to the connection configuration shown in Table 62. The processing element consists of four subtractors and four accumulators as shown in Figure 212. The processing element has two inputs one row from the candidate block and one

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

row from the reference block. To accumulate the absolute value needed in the SAD computations, the first two adders add or subtract the subtractors' outputs according to their sign bit.

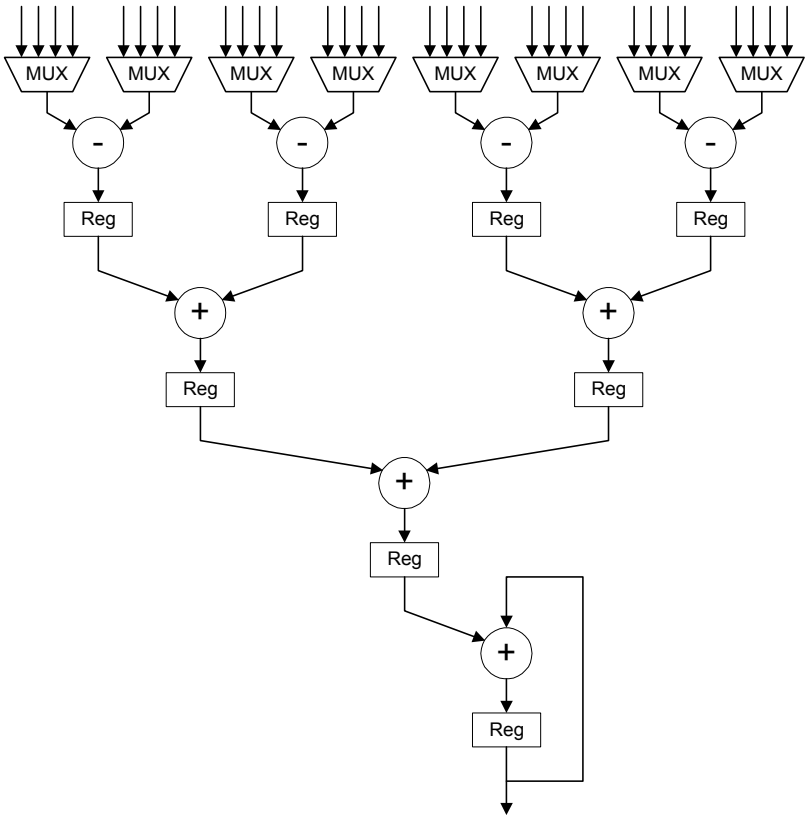


Figure 212 — Block diagram of one processing element.

Figure 213 shows block diagram of the SAD comparison unit. This part compares between the 31 SAD values computed by the processing elements and generates the required horizontal and vertical motion vectors. The inputs to this part are stored in the 31 registers shown in Figure 213, which enables the SAD comparison unit to compare between the computed SAD values while the processing elements compute the SAD values for the next row of blocks. The horizontal and the vertical motion vector registers act as pointers to the position of the block with minimum SAD. The final value of the horizontal and the vertical motion vectors lay between -15 and 15. The algorithm of the SAD comparison unit is shown in Figure 214, where i and j are the outputs of the horizontal and the vertical counters respectively.

The operation of the proposed architecture can be explained as follows: For each block in the current frame (i.e. reference block), 1) read and store the corresponding search window texture in the search window memory, 2) read and store the reference block texture in the reference block memory, 3) compute the SAD values for one row of candidate blocks, 4) find the block with minimum SAD value among the computed SAD values, 5) repeat steps 3 and 4 for all the candidate blocks (row by row), 6) generate the horizontal and the vertical motion vectors for the reference block.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

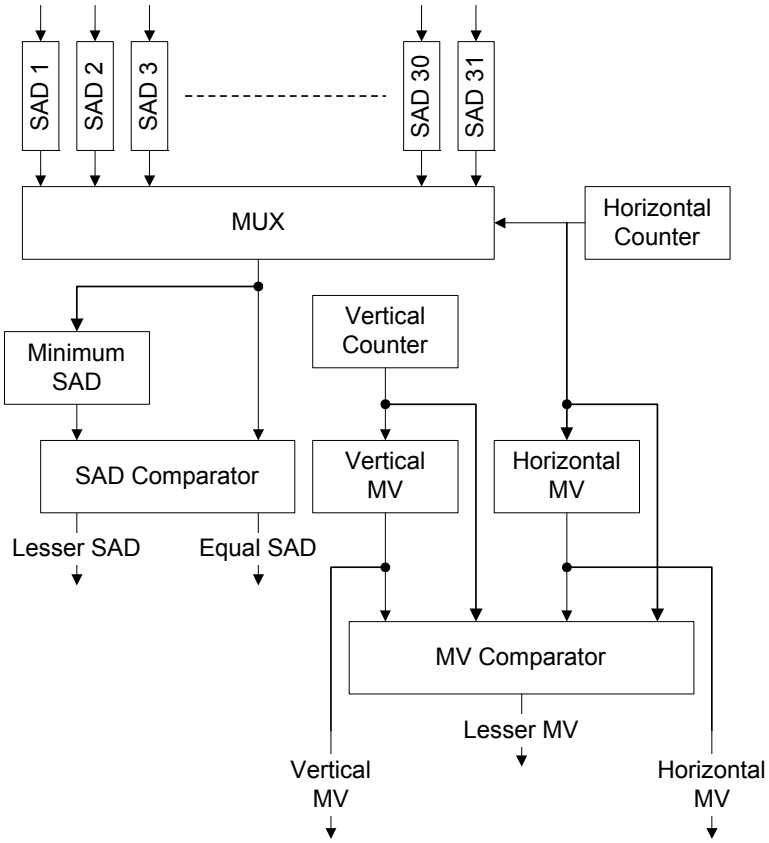


Figure 213 — Block diagram of the SAD comparison unit.

```
if (SAD < minimum_SAD)
{
    minimum_SAD = SAD;
    Horizontal_MV = i;
    Vertical_MV = j;
}
else if (SAD == minimum_SAD)
```

Figure 214 — The algorithm of the SAD comparison unit.

10.17.6.1 Interfaces

Description of I/O interfaces

10.17.6.2 Register File Access

N/A

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.17.6.3 Timing Diagrams

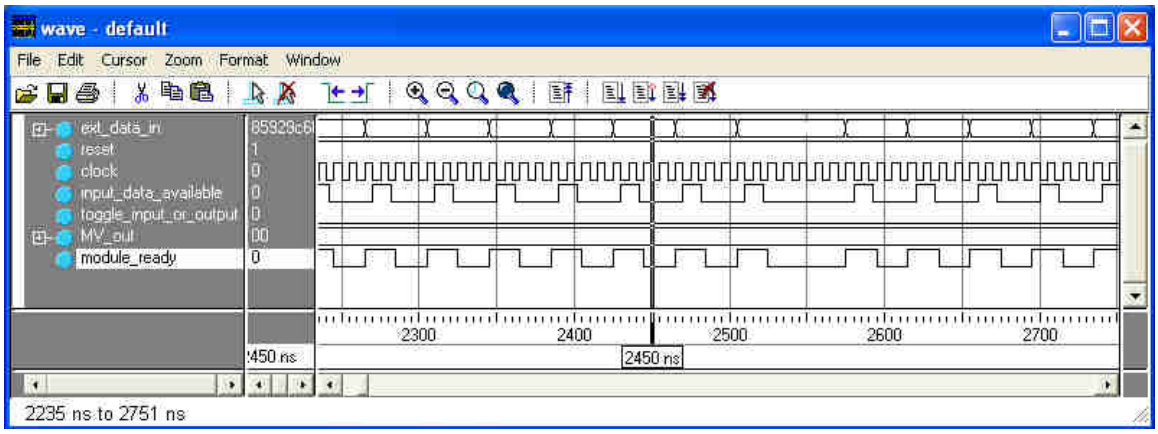


Figure 215 — Writing the search window texture.

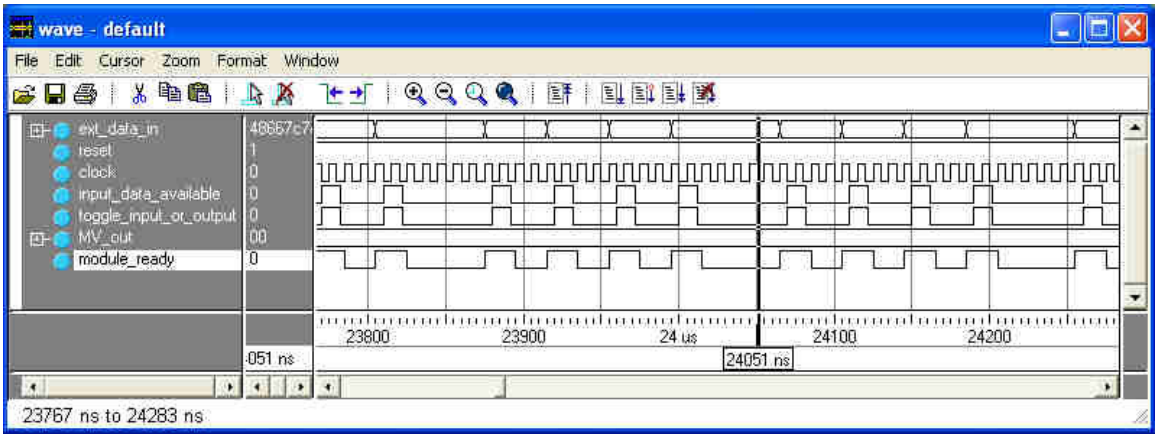


Figure 216 — Writing the reference block texture.

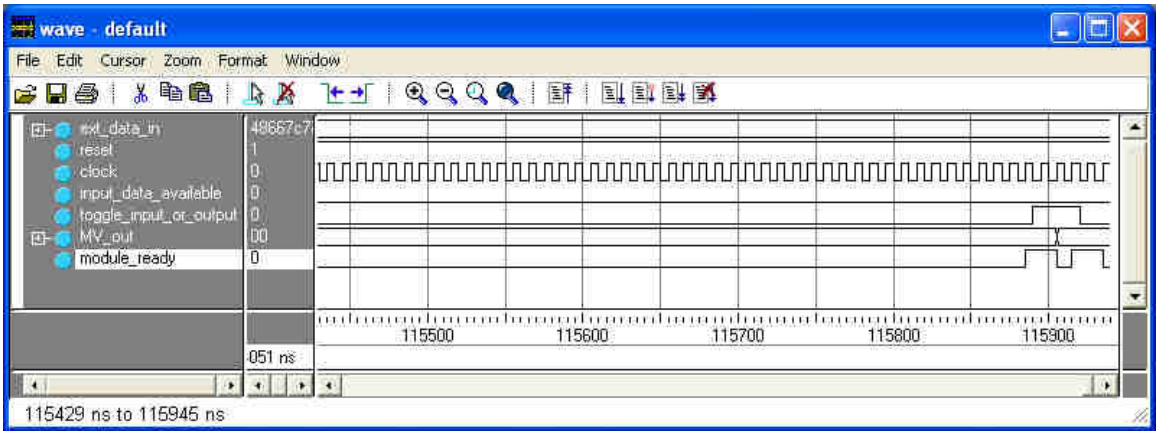


Figure 217 — Reading the estimated motion vector.

10.17.7 Results of Performance & Resource Estimation

The motion estimation module has been prototyped, simulated and synthesized for Xilinx Virtex II FPGA XC2V3000-4. Using the max clock frequency (122.8 MHz), the proposed architecture needs 94.41 µs to process one block with 16x16 pixels size. This module utilizes 20% of the register bits, 16% of the Block RAMs, and 16% of the LUTs in Xilinx Virtex II FPGA XC2V3000-4, which is the processing element in the Annapolis Wildcard II. The proposed architecture processes one CIF video frame (i.e. 352x288 pixels) in 37.38 ms such that it can process up to 26.74 CIF video frames per second.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.17.8 API calls from reference software

N/A

10.17.9 Conformance Testing

The architecture has not been tested on the platform but its simulation results conforms with the software results.

10.17.9.1 Reference software type, version and input data set

N/A.

10.17.9.2 API vector conformance

N/A.

10.17.9.3 End to end conformance (conformance of encoded bitstreams or decoded pictures)

N/A.

10.17.10 Limitations

N/A.

10.18 HARDWARE MODULE FOR MOTION ESTIMATION (4xPE)

10.18.1 Abstract description of the module

This subclause describes a hardware acceleration module for the 4xPE MPEG-4 Motion Estimation architecture. The 4xPE hardware acceleration module is a low-power low-area motion estimation architecture. Its basic Processing Elements exploit the SAD cancellation mechanism in order to remove the redundant SAD operations. It also uses pixel subsampling to split the macroblock information into equal size blocks and this way balance the computational complexity between the Processing Elements that carry out in parallel the SAD calculations at sub-block level. This architecture is normally used for fast exhaustive motion estimation, that is it generates the optimum motion vectors and minimum SAD value. However, fast heuristical motion estimation implementations can be designed to work with the architecture described here, wherein reduced pixel (pixel subsampled) information is used to calculate sub-optimal motion vectors (i.e. sub-optimal match with sub-optimal SAD value).

10.18.2 Module specification

10.18.2.1 MPEG 4 part: **2 (Video)**
10.18.2.2 Profile: **Simple and above**
10.18.2.3 Level addressed: **L1, L2**
10.18.2.4 Module Name: ***ME_4xPE***
10.18.2.5 Module latency: By module latency it is meant the period of time taken to generate motion vectors (MV) for each macroblock. Namely, it is the time difference between the moment when the first set of inputs (pels) are provided to design's inputs and the moment when the first set of output values (MVs) are calculated and provided on the output signals. However, the module latency of the ME_4xPE architecture is variable and depends on the nature of the video input (i.e. its motion level). This is due to the adaptive nature of the SAD cancellation mechanism employed in ME_4xPE. For the extreme case when the SAD cancellation mechanism is disabled a match will be carried out every 64 steps, i.e. 64x4=256 SAD operations carried out in 4 parallel processing elements (PE). For a 15x15 match positions ([+7, -7] positions around the current macroblock) a number of 225 matches have to be carried out. This translates into 225x64 = 14400 clock cycles. The output will be a relative MV with its X and Y components that can have values between [-7, 7]. Thus, 4 bits will be necessary for each MV component, that is 8bits/MV=1Byte/MV. If a MV is generated every 14400th clock cycle, then one could estimate a maximum module latency of 145us to calculate a MV for each

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

macroblock at a maximum clock frequency of 99-100MHz listed below for a Virtex2 technology. However, over 90% of SAD operations can be removed by employing the SAD cancellation mechanism. For example, this is the case for a typical mobile conference video test sequence (akiyo.qcif). Consequently, at least a 10 times improvement is achieved roughly in terms of module latency bringing it to approx **14us** under the same technological conditions.

10.18.2.6 Module data throughput: A minimum **6.9KB/s** (kilobytes per second) data throughput is calculated based on the maximum (no SAD cancellation) module latency estimated above. However, with SAD cancellation it is believed that an order of magnitude improvement can be achieved, that is approx. **70KB/s.**

10.18.2.7 Max clock frequency: Approx 99.3 MHz (critical path of 10.1ns)

10.18.2.8 Resource usage: Please note that the figures below represent only the ME datapath without the search window memory which will be implemented in the hardware module controller during the Wildcard integration process. A 31x31 = 961Bytes (31 = 15 match positions vertically or horizontally + 16pels macroblock size, respectively) search window memory has to be implemented, but it is outside the scope of this document.

10.18.2.8.1 CLB Slices: 636 out of 14336 (4% of a Virtex 2 - xc2v3000 device)
10.18.2.8.2 Block RAMs: None for the moment, though a 31x31
10.18.2.8.3 Multipliers: None
10.18.2.8.4 External memory: SRAM needed on WildCard: 2x25344 Bytes for 2 x luminance frames in QCIF format and 2x101376 Bytes for 2 x luminance frames in CIF format.
10.18.2.8.5 Other metrics Equivalent Gate Count = 10828
10.18.2.9 Revision: v1.0
10.18.2.10 Authors: Valentin Muresan
10.18.2.11 Creation Date: October 2004
10.18.2.12 Modification Date: October 2004

10.18.3 Introduction

This subclause describes a low-power low-area hardware acceleration architecture for one of the most computationally intensive video processing algorithms – motion estimation (ME). The algorithm's behaviour (e.g. SAD cancellation, pixel subsampling) is exploited in order to remove redundant operations, hence eliminating unwanted dynamic power consumption. Also, the area taken by the architecture is sensibly smaller than other architectures previously proposed in the literature, thus static power is also reduced.

ME's high computational requirements are addressed by implementing in HW a SAD cancellation mechanism. Due to the fact that this approach is based on re-mapping and partitioning the video content by means of pixel subsampling (see Figure 218), only architectures with a 2^{2*n} number of Pes can be implemented. However, cases for when n = 3 or 4 are rather extreme where the architecture effectively becomes a 2D systolic array. This subclause describes the implementation of an architecture which has 4 Pes (Figure 219) and is named ME_4xPE. The main principles behind the design of this architecture are as follows:

- The ME algorithm has been analysed and the computation steps have been re-formulated and merged on the premise that if there are less operations to be carried out, there will be less switching and hence less energy dissipation. Hence, a SAD cancellation mechanism is considered an effective approach to achieve the above;
- Since the computational load of the SAD-cancellation mechanism depends entirely on the video characteristics, the circuit swithing activity and processing latency is proportional to the amount of motion in the video frames;
- The processing latency of the module is large because it does not make excessive use of parallelism. However, other variations of the proposed architecture, with more Pes or pipelined structures are being implemented and the power efficiency will be traded-off for speed (smaller latency and higher throughput);
- To get a maximum of effectiveness, the pixel subsampling technique is employed in order to balance the workload throughout the Pes (see Figure 218);

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

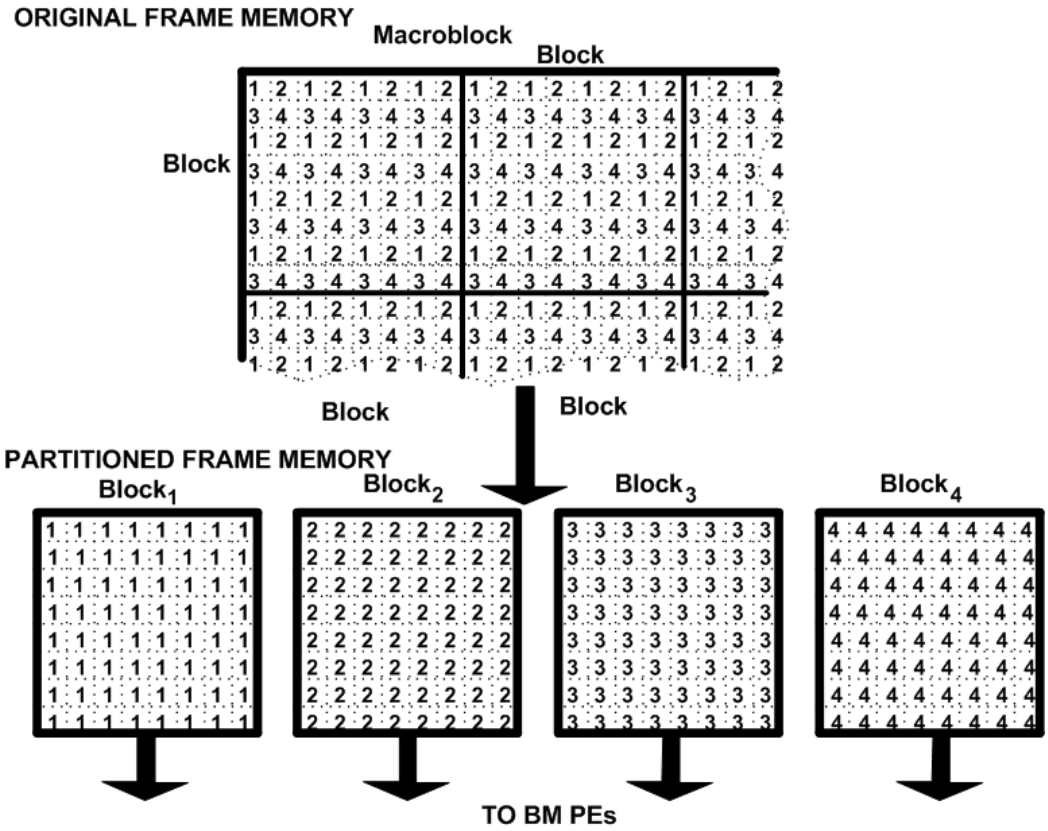


Figure 218 — Video Data Re-mapping and Partitioning.

10.18.4 Functional Description

10.18.4.1 Functional description details

The *ME_4xPE* module has been implemented with two main sub-modules that search the minSADs in a search window: a circular search strategy (*bm_search_strategy* RTL module) and the actual ME datapath (*bm_adaptive_4xPE_core* RTL module). A conceptual diagram of the *bm_adaptive_4xPE_core* module is shown in Figure 219. A more detailed description of the overall ME architecture is given in [42].

© ISO/IEC 2009 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

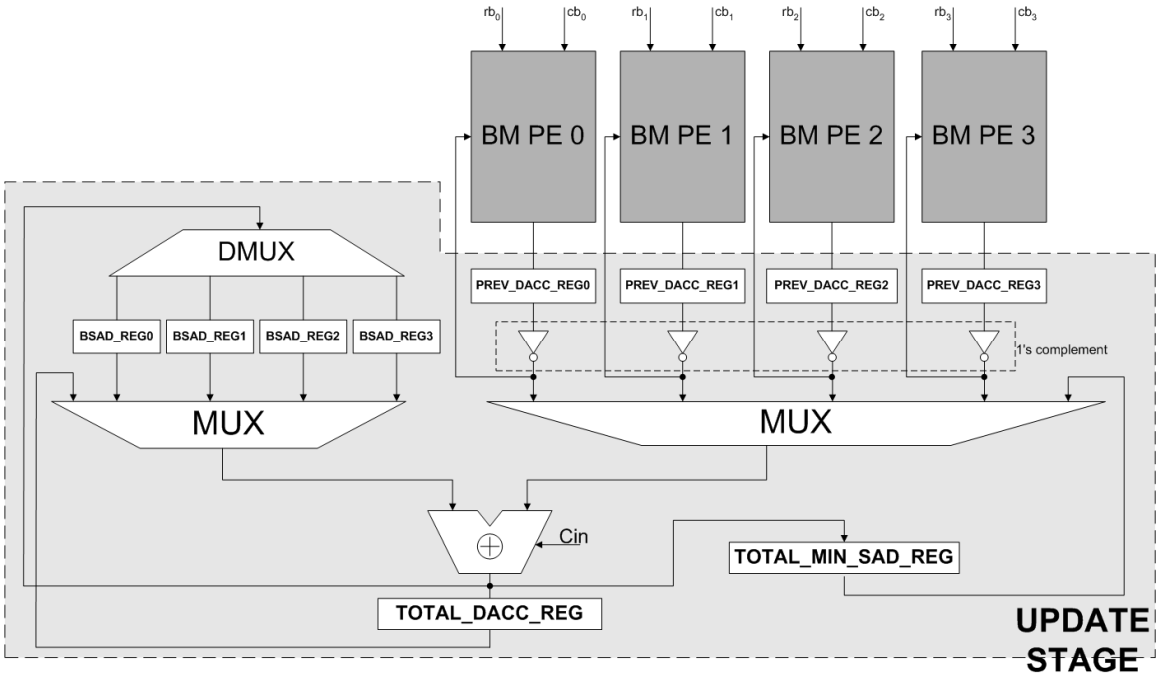


Figure 219 — 4xPE Architecture = 4BM PEs + Update Stage.

Figure 220 depicts a detailed view of a Block Matching (BM) Processing Element (PE) employed above. A SAD calculation implies a subtraction, an absolute and an accumulation operation. Since relative values to the current *minSAD* and *minBSAD_k* (block-level) values are calculated, a de-accumulation function is used instead. The absolute difference is de-accumulated from the *bk_dacc_reg* register (de-accumulator) which is at the center of the bottom shaded block. At each moment the *bk_dacc_reg* stores the appropriate relative (to the current *minSAD*) block-level SAD value and signals immediately with its sign bit if it becomes negative. The initial value stored in the *bk_dacc_reg* at the beginning of each best match search is the corresponding *minBSAD_k* value and is brought through the *bk_local_sad_reg* inputs. For the first match, when a minimum SAD has not been calculated yet, a maximum value is brought instead till the *minBSAD_k* values are initialized. Any time all the *bk_dacc_reg* become negative they signal a SAD-cancellation condition and the update stage is kept idle. If this condition is not met before the end of the block match (64-cycles for the 4xPE architecture), then the result of the *bk_dacc_reg* is transferred in the corresponding *mk_prev_dacc(K)_reg* register in the update stage before the PEs are committed to a new block-match. The circuitry within the top shaded block represents the absolute-difference logic. The output of the absolute-difference generates in parallel both non-inverted and inverted (1's complement) versions of the difference result in order to be able to select the absolute-difference based on subtraction's sign output. The pixel values are brought sequentially from the appropriate bank of the ME memory to the *bk_cur_in* (current block) and *bk_prev_in* (reference block) inputs. The *bk_cur_in* is inverted to 2's complement (1's complement and *C_in = 1* in order to get *bk_cur_in's* negative value. The shaded block in the middle is the control logic that provides the de-accumulator with various inputs based on the function executed: firstly, to de-accumulate the absolute-difference provided through either of the two left-most inputs of the *4:1 Mux*, secondly, to initialize *bk_dacc_reg* through the *bk_local_sad_reg* inputs with the corresponding current *minBSAD_k* value, and, thirdly, to correct the relative (de-accumulated) SAD value stored in the *bk_dacc_reg* through the *bk_prev_dacc_reg* inputs when the update stage deems it necessary.

--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2009 – All rights reserved
Not for Resale

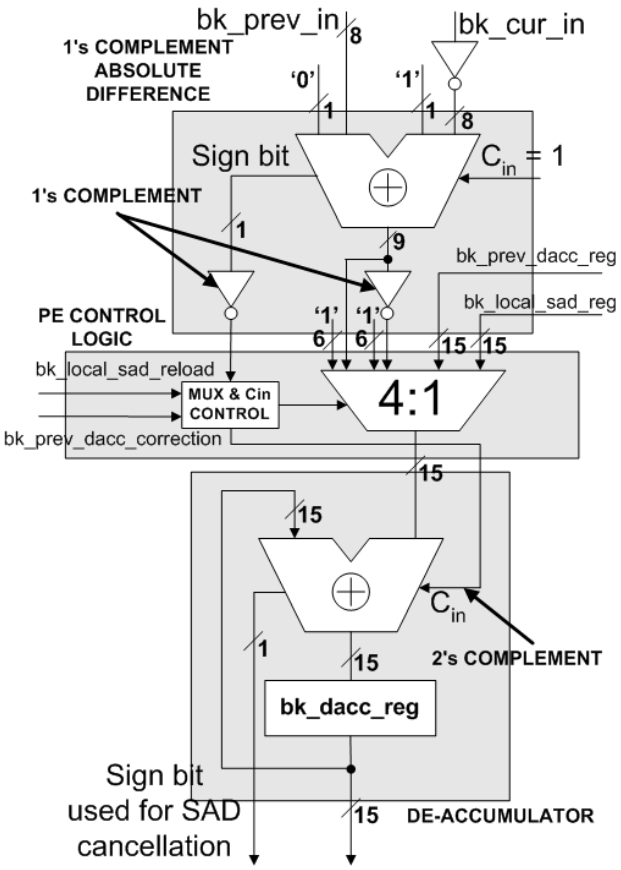


Figure 220 — Block Matching Processing Element (BM PE).

The update stage can be carried out in parallel with the next match's operations executed in the block-level datapaths because it takes at most 11 cycles. Therefore, a pure sequential scheduling of the update stage operations is implemented in the update stage hardware and is described in Figure 220. There are three possible update stage execution scenarios: first, when it is idle most of the, second, when the update is launched at the end of a match, but after 5 steps the global SAD relative to the minSAD turns out to be negative and no update is deemed necessary, third, when after 5 steps the relative SAD is positive and an update of the block-level SAD values and the total (macroblock-level) SAD value is carried in the rest of 6 steps (see [42] for a more detailed description).

10.18.4.2 I/O Diagram

The top-level I/O signals of the *ME_4xPE* module are summarised in Figure 221.

© ISO/IEC 2009 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

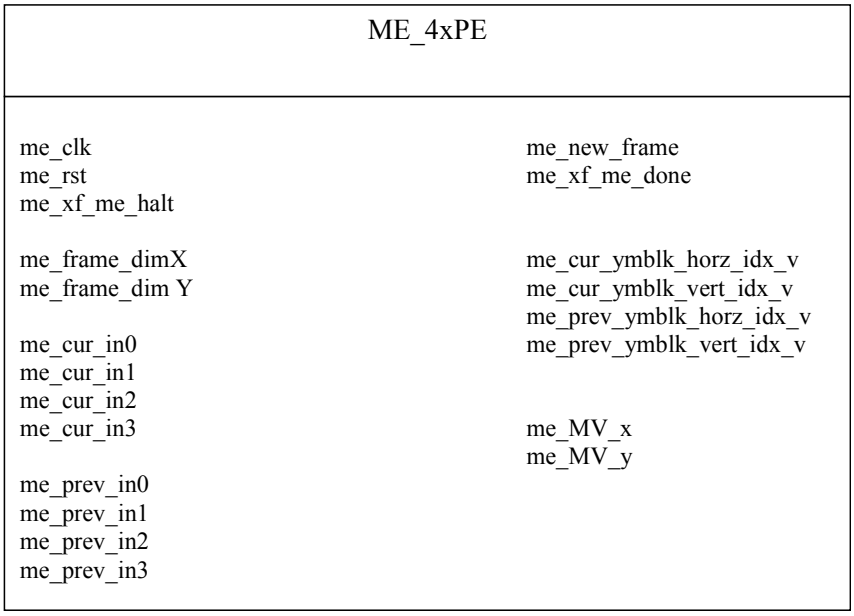


Figure 221 — Top Level I/O Ports.

10.18.4.3 I/O Ports Description

Table 63

Port Name	Port Width	Type	Description
me_clk	1	Input	System clock
me_rst	1	Input	Asynchronous active-high reset
me_xf_me_halt	1	Input	Handshaking signal controlled by the memory controller that tells ME-4xPE to wait as the memory data is not ready yet
me_frame_dimX	DIM_WIDTH	Input	Frame horizontal dimension
me_frame_dimY	DIM_WIDTH	Input	Frame vertical dimension
me_cur_in_[0..3]	4x8	Input	In the 4xPE architecture there are 4 pixels addressed from the current block
me_prev_in_[0..3]	4x8	Input	DITTO for the previous block to match
me_xf_me_done	1	Output	Handshaking signal driven by ME_4xPE that tells the memory controller that it can fetch the new set of pel data
me_new_frame	1	Output	Handshaking signal that tells the memory controller that a whole new frame has to be fetched
me_cur_ymblk_horz_idx_v	SEARCH_ADR_BUS_SIZE +MATCH_ADR_BUS_SIZE	Output	Horizontal address of the 4 pixels (fetched in parallel from the 4 pixel subsampled remapped sub-frames/sub-blocks) in the current block
me_cur_ymblk_vert_idx_v	SEARCH_ADR_BUS_SIZE +MATCH_ADR_BUS_SIZE	Output	Vertical address of the 4 pixels (fetched in parallel from the 4 pixel subsampled remapped sub-frames/sub-blocks) in the current block

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

me_prev_ymblk_horz_idx_v	SEARCH_ADR_BUS_SIZE +MATCH_ADR_BUS_SIZE	Output	Horizontal address of the 4 pixels (fetched in parallel from the 4 pixel subsampled remapped sub-frames/sub-blocks) in the previous block
me_prev_ymblk_vert_idx_v	SEARCH_ADR_BUS_SIZE +MATCH_ADR_BUS_SIZE	Output	Vertical address of the 4 pixels (fetched in parallel from the 4 pixel subsampled remapped sub-frames/sub-blocks) in the previous block
me_MV_x	MV_WIDTH	Output	Horizontal motion vector associated with the minimum SAD
me_MV_y	MV_WIDTH	Output	Vertical motion vector associated with the minimum SAD

10.18.4.3.1 Parameters (generic)

N/A.

10.18.4.3.2 Parameters (constants)

Table 64

Parameter Name	Type	Value	Description
SEARCH_ADR_BUS_SIZE	INT	6	The macroblock-level horizontal and vertical address bus size/width. The current value is sufficient for the address space of a CIF format
MATCH_ADR_BUS_SIZE	INT	3	The block-level horizontal and vertical address bus size/width. Because a block has 8x8=64 pixels in ME_4xPE, 3 bits are enough for the vertical and horizontal indexes
DIM_WIDTH	INT	9	The bit-width of the horizontal and vertical frame dimensions
PIXEL_DATA_SIZE	INT	8	Luminance pixels bit-width
MAXVAL	INT	0x3fff	The maximum value that the block-level SAD registers are initialized with at each new search
NR_SUBLOCKS	INT	4	The number of Processing Elements
MATCH_COUNTUP	INT	63	The number of SAD operations for a full (uncancelled) match: 0-63 = 64 cycles
UPC_Xth_STATE	INT	0-11	Update Control FSM's state machines
CIRC_SEARCH_LAPS	INT	6	Circular Search Strategy's number of laps ([0..6] = 6+1, i.e. [+7, -7] horizontal and vertical range)
DACC_REG_WIDTH	INT	15	bk_dacc_reg's bit width
LAST_MACK_I	INT	160	The horizontal position of the last macroblock in a QCIF video test sequence's frame
LAST_MACK_J	INT	128	The horizontal position of the last macroblock in a QCIF video test sequence's frame

These constants are defined in the SystemC hardware model of ME_4xPE. However, after the RTL compilation with the help of Synopsys's SystemC the output is RTL Verilog code where these constants are hardwired with the given constant values.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.18.5 Algorithm

The SAD cancellation mechanism has been proposed so far in the motion estimation algorithm only in the context of a fully serial software implementation. At each cycle, the current sum of absolute difference total is compared with the current best SAD for the current search area. If the former is greater than the latter the current SAD calculation can be terminated at this point. This method reduces the number of total operations required to find the motion vectors by discounting worse matches early on, thus saving on the operations which would have been required for finding the exact SAD for these worse matches. Reducing

the number of operations results in a reduction of power drain (circuit switching activity) and at the same time a shortening of the overall motion estimation time.

10.18.6 Implementation

The architecture *ME_4xPE* has been implemented using SystemC in a structural style with nine RTL sub-modules at different hierarchical levels as in Figure 222. Full details of the internal architectural structure are given in [42]. The module is going to be integrated next with the multiple IP-core hardware accelerated software system framework developed by the University of Calgary and is going to be implemented on a Windows XP laptop with the Annapolis PCMCIA FPGA (Xilinx Virtex-II XC2V 3000-4) prototyping platform installed. The search window memory and current macroblock memory will be implemented within the hardware module controller that is going to interface the *ME_4xPE* module with the virtual socket.

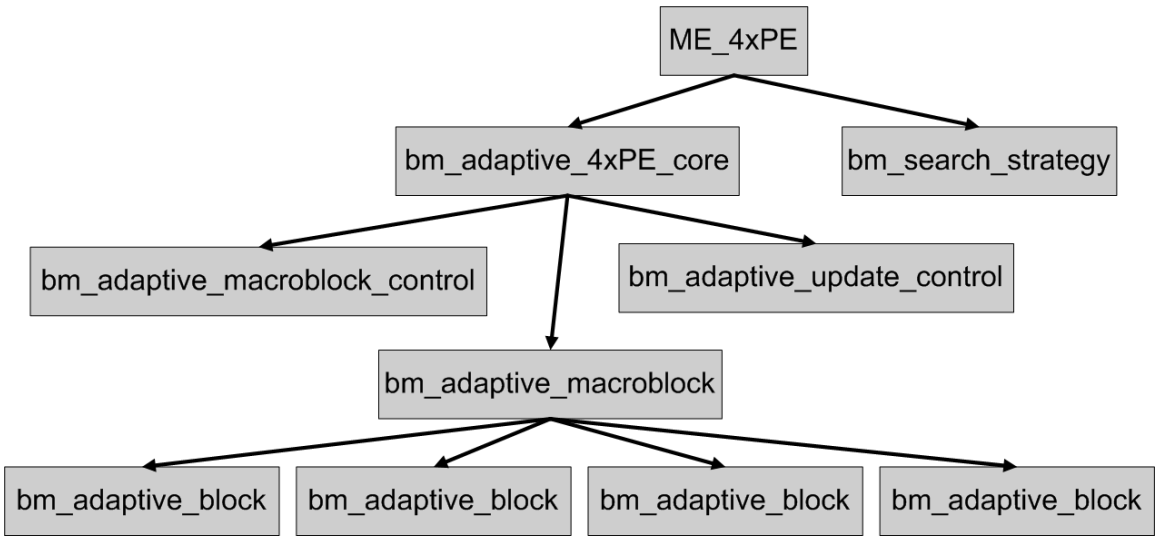


Figure 222 — SystemC Modules Hierarchy of ME_4xPE.

The synthesis flow employed so far is depicted in Figure 223. VisualC++ 6.0 is used in the first instance to model an RTL representation of *ME_4xPE* in SystemC. The output of this design capture stage is then compiled with Synopsys's SystemC compiler (2003.12 SP1). If errors have been encountered during this compilation, those errors were mainly due to the fact that the SystemC description does not meet the RTL description guidelines. Thus, the *RTL SystemC Modeling* is repeated till all the RTL related errors are eliminated. The Verilog code is co-simulated using Synopsys VCS (2003.12 SP1) to guarantee correct functionality in the translated Verilog files. Once an RTL Verilog code is generated it is imported in Synplify PRO 7.5 and the *Synthesis* process is carried out taking into account the implementation constraints set by the designer. An EDIF representation of the synthesized RTL Verilog code is then exported to ISE 6.2.03i which carries out the final *Place&Route* stage that accomplishes the implementation process for a Wildard Xilinx Virtex 2 FPGA technology.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

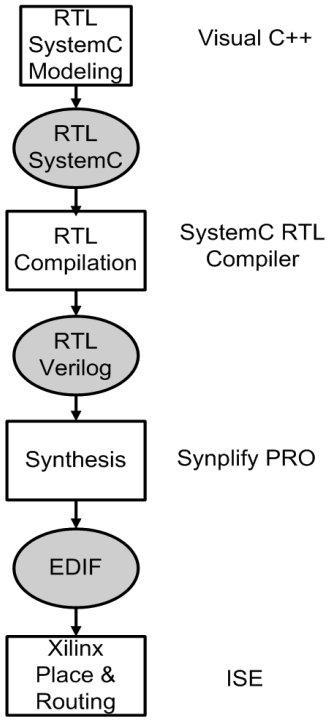


Figure 223 — Synthesis Flow.

10.18.6.1 Interfaces

N/A.

10.18.6.2 Register File Access

N/A.

10.18.6.3 Timing Diagrams

Figure 224 and Figure 225 depict the first match and the first match/update overlap of the first set of macroblocks in the first frame of the container_qcif.yuv video test sequence. The most important input, output and control signals in these two scenarios are listed in the waveform diagrams. In Figure 225 the update control process could be noticed at the bottom of the depicted signals list.

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

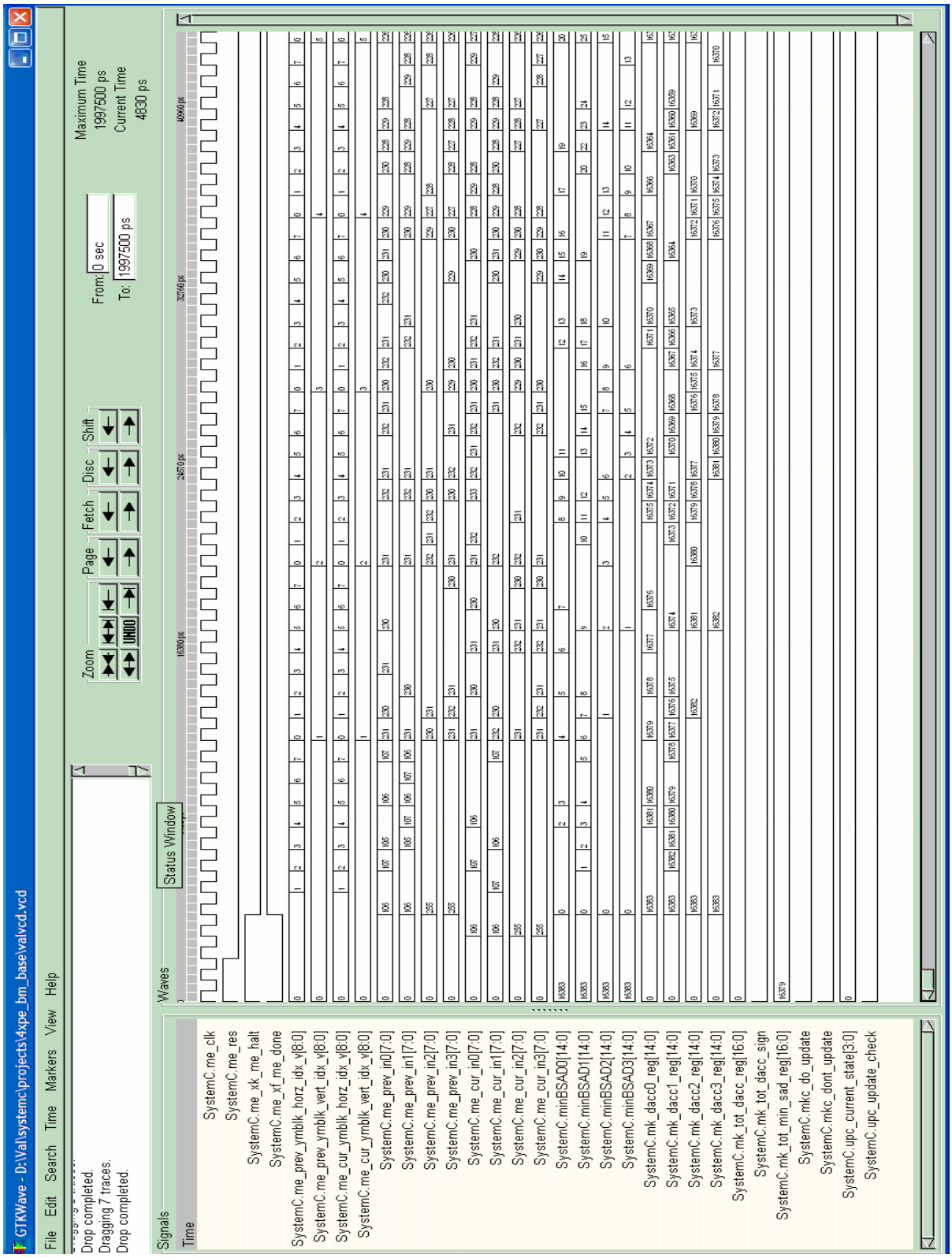


Figure 224 — Sample of First Match Timing Diagram.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2009 – All rights reserved

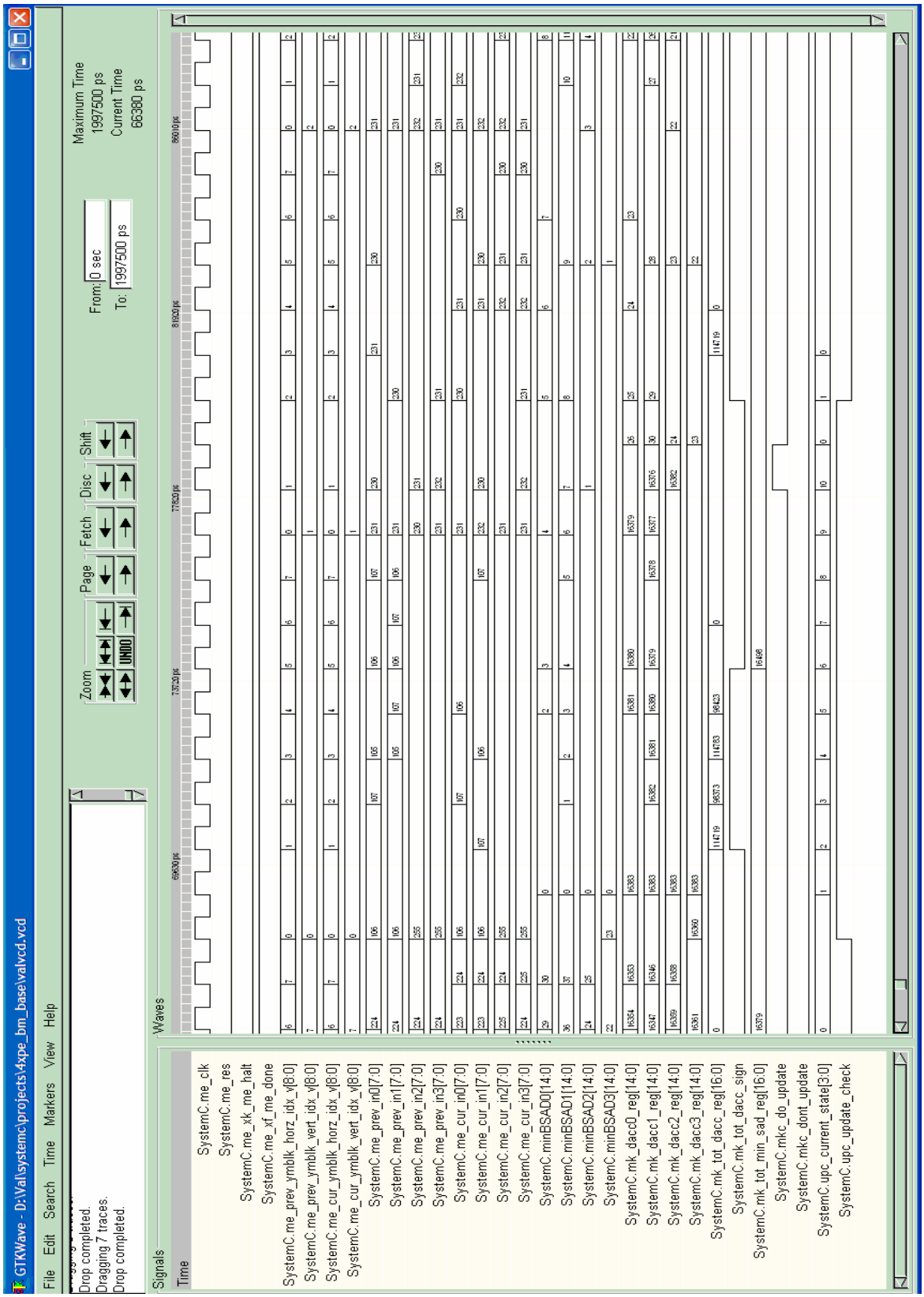


Figure 225 — Sample of First Match/Update Timing Diagram.

10.18.7 Results of Performance & Resource Estimation

Below is an excerpt of the final resource results reported by ISE:

© ISO/IEC 2009 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
Release 6.2.03i Map G.31a
Xilinx Mapping Report File for Design 'ME_4xPE'
Design Information
------------------
Command Line   : D:/Xilinx/bin/nt/map.exe -intstyle ise -p XC2V3000-FG676-4 -cm
area -pr b -k 4 -c 100 -tx off -o ME_4xPE_map.ncd ME_4xPE.ngd ME_4xPE.pcf
Target Device  : x2v3000
Target Package : fg676
Target Speed   : -4
Mapper Version : virtex2 -- $Revision: 1.16.8.1 $
Mapped Date    : Fri Oct 15 09:49:02 2004

Design Summary
--------------
Number of errors:      0
Number of warnings:    1
Logic Utilization:
  Total Number Slice Registers:       395 out of  28,672    1%
    Number used as Flip Flops:                391
    Number used as Latches:                     4
  Number of 4 input LUTs:             858 out of  28,672    2%
Logic Distribution:
  Number of occupied Slices:          650 out of  14,336    4%
  Number of Slices containing only related logic:     650 out of     650  100%
  Number of Slices containing unrelated logic:          0 out of     650    0%
        *See NOTES below for an explanation of the effects of unrelated logic
Total Number 4 input LUTs:            949 out of  28,672    3%
  Number used as logic:               858
  Number used as a route-thru:         91

  Number of bonded IOBs:              169 out of     484   34%
    IOB Flip Flops:                     1
    IOB Latches:                        1
  Number of GCLKs:                      3 out of      16   18%

Total equivalent gate count for design:  10,828
Additional JTAG gate count for IOBs:  8,112
Peak Memory Usage:  121 MB

Section 13 - Additional Device Resource Counts
-----------------------------------------------
Number of JTAG Gates for IOBs = 169
Number of Equivalent Gates for Design = 10,828
Number of RPM Macros = 0
Number of Hard Macros = 0
CAPTUREs = 0
```


--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
BSCANs = 0
STARTUPs = 0
PCILOGICs = 0
DCMs = 0
GCLKs = 3
ICAPs = 0
18X18 Multipliers = 0
Block RAMs = 0
TBUFs = 0
Total Registers (Flops & Latches in Slices & IOBs) not driven by LUTs = 301
IOB Dual-Rate Flops not driven by LUTs = 0
IOB Dual-Rate Flops = 0
IOB Slave Pads = 0
IOB Master Pads = 0
IOB Latches not driven by LUTs = 1
IOB Latches = 1
IOB Flip Flops not driven by LUTs = 1
IOB Flip Flops = 1
Unbonded IOBs = 0
Bonded IOBs = 169
Total Shift Registers = 0
Static Shift Registers = 0
Dynamic Shift Registers = 0
16x1 ROMs = 0
16x1 RAMs = 0
32x1 RAMs = 0
Dual Port RAMs = 0
MUXFs = 29
MULT_ANDs = 139
4 input LUTs used as Route-Thrus = 91
4 input LUTs = 858
Slice Latches not driven by LUTs = 4
Slice Latches = 4
Slice Flip Flops not driven by LUTs = 295
Slice Flip Flops = 391
Slices = 650
Number of LUT signals with 4 loads = 9
Number of LUT signals with 3 loads = 14
Number of LUT signals with 2 loads = 377
Number of LUT signals with 1 load = 417
NGM Average fanout of LUT = 2.28
NGM Maximum fanout of LUT = 221
NGM Average fanin for LUT = 2.8042
Number of LUT symbols = 858
Number of IPAD symbols = 131
Number of IBUF symbols = 131
```

Figure 226

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The excerpt related to the maximum achievable frequency that was taken from the report generated by Synplify PRO is given next:

```
Performance Summary
*******************



Worst slack in design: -3.406

                                                  Requested     Estimated     Requested     Estimated
Starting Clock                                    Frequency     Frequency     Period        Period
-----------------------------------------------------------------------------------------------------
ME_4xPE|core.Macro_block.mk_gated_clk_inferred_clock     150.0 MHz     176.6 MHz     6.667     5.662
ME_4xPE|core.cr_gated_clk_inferred_clock                150.0 MHz     99.3 MHz      6.667     10.073
=====================================================================================================
```

Figure 227

10.18.8 API calls from reference software

N/A.

10.18.9 Conformance Testing

N/A.

10.18.9.1 Reference software type, version and input data set

The hardware acceleration framework with the integrated 4xPE ME IP core is currently being integrated with Microsoft MPEG-4 Part 7 Optimised Video Reference Software (version microsoft-v2.4-030710-NTU).

10.18.9.2 API vector conformance

N/A.

10.18.9.3 End to end conformance (conformance of encoded bitstreams or decoded pictures)

End to end conformance has not yet been completed. The software implementation of ME in the reference software will be run a test sequences. During the run the relevant data will be collected. Then the software implementation of ME will be replaced by SW API calls to the ME hardware on the integration framework again gathering the relevant data. Then analysing the two generated results comparison from a conformance and performance perspective will be carried out.

10.18.10 Limitations

An possible limitation of this module is that the module latency increases with the video data which involves a lot of motion. However, the current figures show that real time MPEG-4 based multimedia application needs (at 30 frames/s) are achievable for the given technology. Moreover, architecture variations as 16xPE can be found to be a better trade off for larger frame formats. However, the large power saving gains are significantly traded-off by the reduction in speed. This fact proves two points:

- Under the given technological limitations the *ME_4xPE* can be successfully used for security related motion detection-based application, where the motion in the frame sequence is usually lower;
- In order to meet the real-time constraints of a high-quality MPEG-4 encoding application for larger video frame formats, more than one *ME_4xPE* module can be employed to run in parallel. This will have an impact

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

on size of the search window and current macroblock memory architectures to be designed in the hardware module controller. However, even if a larger multi-*ME_4xPE* based architecture will obviously need more resources (area) to achieve the real time needs, the number of SAD operations will be still significantly reduced, and overall the same level of operation removal (over 90%) can be achieved with clear impact on the power saving achievements. This will be the target of our future research efforts.

10.19 A IP BLOCK FOR H.264/AVC QUARTER PEL FULL SEARCH VARIABLE BLOCK MOTION ESTIMATION

10.19.1 Abstract description of the module

This subclause describes a Verilog model for H.264/AVC quarter pel full search variable block motion estimation. The architecture is capable of calculating all 41 motion vectors required by the various size blocks, supported by H.264/AVC, in parallel. The architecture is prototyped and simulated using ModelSim 5.4. It is synthesized by Xilinx 6.2 ISE development tools for VirtexII FPGA XC2V3000. The prototype is capable of processing CIF frame sequences in real time considering 5 reference frames within the search range of -3.75 to +4.00 at a clock speed of 120MHz. The maximum speed of the architecture is around 150MHz.

10.19.2 Module specification

10.19.2.1 MPEG 4 part: ***10***
10.19.2.2 Profile : ***All***
10.19.2.3 Level addressed: ***All***
10.19.2.4 Module Name: **ME_AVC**
10.19.2.5 Module latency: **2,071 clock cycles**
10.19.2.6 Module data throughput: **2,954,805 motion vector/sec at max clock frequency**
10.19.2.7 Max clock frequency: ***149.2MHz***
10.19.2.8 Resource usage:
10.19.2.8.1 CLB Slices: ***8,951***
10.19.2.8.2 DFFs or Latches: ***13,091***
10.19.2.8.3 LUTs: **13,857**
10.19.2.8.4 BRAMs: **39**
10.19.2.8.5 Number of Gates: **225K**
10.19.2.9 Revision: ***1.00***
10.19.2.10 Authors: ***Choudhury A. Rahman and Wael Badawy***
10.19.2.11 Creation Date: ***December 2004***
10.19.2.12 Modification Date: ***December 2004***

10.19.3 Introduction

The newest international video coding standard has been finalized in May 2003. It is approved both by ITU-T as Recommendation H.264 and ISO/IEC as International Standard 14496-10 (MPEG-4 part 10) Advanced Video Coding (AVC) [13]. This new standard H.264/AVC is designed for application in the areas such as broadcast, interactive or serial storage on optical and magnetic devices such as DVDs, video-on-demand or multimedia streaming, multimedia messaging etc. over ISDN, DSL, Ethernet, LAN, wireless and mobile networks. Some new features of the standard that enable enhanced coding efficiency by accurately predicting the values of the content of a picture to be encoded are variable block-size, quarter-sample-accuracy and multiple reference picture for motion estimation and compensation [16]. In addition to improved prediction methods, other parts of the design are also enhanced for improved coding efficiency including small block-size transform, hierarchical block transform, exact-match inverse transform, arithmetic entropy coding etc. While the scope of the standard is limited to the decoder by imposing restrictions on the bitstream and syntax, and defining the decoding process of the syntax elements such that every decoder conforming to the standard will produce similar output when given an encoded bitstream that conforms to the constraints of the standard, there is a considerable flexibility in designing an encoder for AVC to optimize implementations in a manner appropriate to the intended application.

The new features such as variable block-size, quarter-sample-accuracy and multiple reference frames increase the complexity and computation load of motion estimation greatly in H.264/AVC encoder. Experimental results have shown that motion estimation can consume 60% for 1 reference frame to 80% for 5 reference frames of the total encoding time of AVC codec. Due to this reason, in order to get real time

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

performance (30 frames per second) from a AVC encoder, parallel processing must be exploited in the architecture. So far, there have been a very few VLSI implementations [33][34] for H.264/AVC motion estimation considering variable block size. But none of them is particularly suitable considering real time frame processing, multiple reference frames and fractional pel accuracy. In this subclause, a quarter pixel full search variable block motion estimation architecture has been proposed that can process all the required motion vectors for H.264/AVC encoder in parallel. Experimental results have shown that the architecture can process in real time up to 5 reference frames at a clock speed of 120MHz.

10.19.4 Functional Description

10.19.4.1 Functional description details

The architecture process CIF format video sequences (i.e. 352x288 pixels frame size) with 16x16 pixels block size and -3.5 to +4.0 search range. It uses full search block matching algorithm with SAD as the matching criteria.

10.19.4.2 I/O Diagram

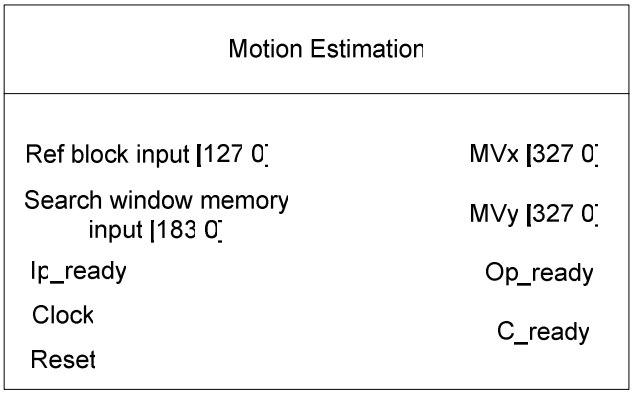


Figure 228 — I/O Diagram.

10.19.4.3 I/O Ports Description

Table 65

Port Name	Port Width	Direction	Description
Ref block input	128	Input	One row of 16x16 reference block's pixels input.
Search window memory input	184	Input	Search window memory inputs
Ip_valid	1	Input	Flag indicating that inputs are valid
Clock	1	Input	System clock
Reset	1	Input	System reset
MVx	328	Output	Output of 41 motion vectors in horizontal direction.
MVy	328	Output	Output of 41 motion vectors in vertical direction.
Op_ready	1	Output	Flag indicating that outputs is valid
C_ready	1	Output	Flag indicating that core is ready for next reference block.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.19.5 Algorithm

Motion estimation is the basic bandwidth compression method adopted in the video coding standards. In H.264/AVC the motion estimation method is further refined with the new features like variable block size, multiple reference frames and quarter pixel accuracy. Up to 5 reference frames can be used along with 7 block patterns: 16x16, 16x8, 8x16, 8x8, 8x4, 4x8 and 4x4 in AVC as shown in Figure 229. Compared to fixed size block and singe reference frame, the new method provides better estimation of small and irregular motion fields and allows better adaptation of motion boundaries resulting in a reduced number of bits required for coding prediction errors.

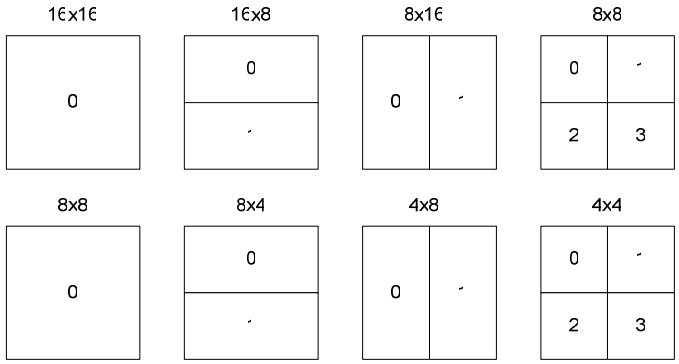


Figure 229 — The various block sizes in H.264/AVC

The block matching algorithm (BMA) is the most implemented one in real time for motion estimation [35]. The algorithm is composed of two parts: matching criterion and searching strategy. In our proposed architecture, sum of absolute difference (SAD) and full search (FS) have been chosen for matching criterion and search strategy, respectively. SAD can be expressed in terms of equation as follows:

$$SAD(dx,dy) = \sum_{i=0}^{M-1}\sum_{j=0}^{N-1}\left|a(i,j) - b(i+dx, j+dy)\right|$$

$$\left(MV_x, MV_y\right) = \left(dx, dy\right)\Big|_{\min SAD(dx,dy)}$$

Equation 76

In Equation 76, a(i,j) and b(i,j) are the pixels of the reference and candidate blocks, respectively. dx and dy are the displacement of the candidate block within the search window. MxN is the size of the reference block and (MVx, MVy) is the motion vector pair of the block.

10.19.6 Implementation

The proposed architecture for quarter pel full search block motion estimation is shown in Figure 230. The architecture composed of single port block RAMs for search window and 16x16 reference block, 8 processing units, shift registers comparing unit and address generator (AG). The search window size of 92x92 pixels (quarter pel) has been chosen for prototyping for which the motion vector for the 16x16 size block lays between -3.75 to +4.00. So, there are 23x23 integer pel positions for which there are 64(8x8) 16x16 candidate blocks. Therefore, the total number of 16x16 candidate blocks considering quarter pel accuracy is 64x4x4 = 1024. The search window has been partitioned into 23 4x92 size block RAMs for parallel processing. This is shown in Figure 231 and it requires a total memory bandwidth of 184(23x8) bits. The address generator generates addresses for the reference and candidate blocks. These addresses are fed into the search window memory, reference block memory and comparing unit.

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

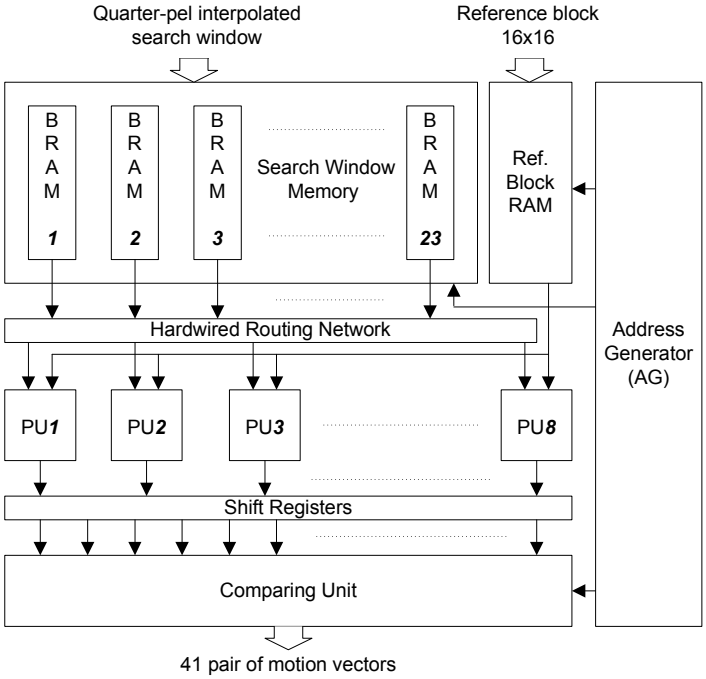


Figure 230 — The motion estimation architecture.

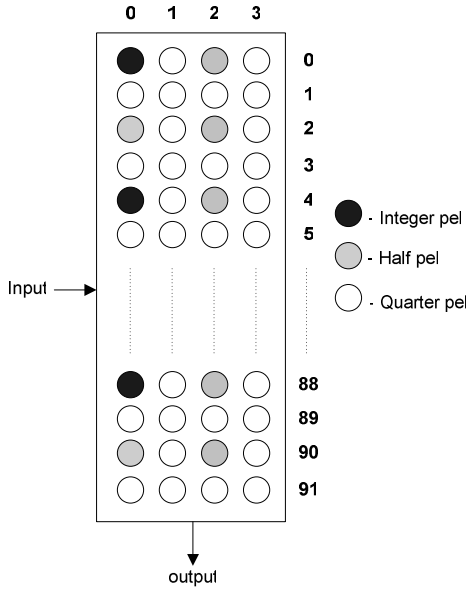


Figure 231 — BRAM for search window memory.

The hardwired routing network connects the search window memory with the PUs. The input / output connections of the routing network are shown in Table 66. Figure 232 shows the C-type address generation algorithm for the AG. This algorithm generates addresses for the search window memory (SW MEM), reference block memory (REF MEM) and Hx, Vx for the processing units fed into the comparing unit. Each (Hx, Vx) pair represents motion vector and it addresses the top left corner point of a 4x4 candidate block shown in Figure 233. This means the motion vectors of all the possible size blocks can be represented by the combination of these Hx, Vx values.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Table 66 — Hardwired routing network's input / output connections.

Input (BRAM)	Output							
	PU1	PU2	PU3	PU4	PU5	PU6	PU7	PU8
1	1							
2	2	1						
3	3	2	1					
4	4	3	2	1				
5	5	4	3	2	1			
6	6	5	4	3	2	1		
7	7	6	5	4	3	2	1	
8	8	7	6	5	4	3	2	1
9	9	8	7	6	5	4	3	2
10	10	9	8	7	6	5	4	3
11	11	10	9	8	7	6	5	4
12	12	11	10	9	8	7	6	5
13	13	12	11	10	9	8	7	6
14	14	13	12	11	10	9	8	7
15	15	14	13	12	11	10	9	8
16	16	15	14	13	12	11	10	9
17		16	15	14	13	12	11	10
18			16	15	14	13	12	11
19				16	15	14	13	12
20					16	15	14	13
21						16	15	14
22							16	15
23								16

```
For c = 0 to 31 {
  For add_h = 0 to 3 {
    For add_v = 0 to 15 {

      SW MEM address = c + add_v*4 + add_h*92;
      REF MEM address = add_v;

      Hx for PU(y) = (add_h + x*16 + y*4 - 15)/4;
      Vx for all PU = (c + x*16 - 15)/4;
        //where, x = {0, 1, .. 3} and y = {0, 1, .. 7}
} } }
```

Figure 232 — Algorithm for address generator.

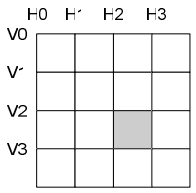


Figure 233 — 4x4 blocks within a 16x16 candidate block and their corresponding addresses. For an example, the address of the gray shaded 4x4 block is (H2, V2).

The PU structure is shown in Figure 234. It has 16 processing elements (PE) shown in Figure 235. The PE is composed of one subtractor, one selectable adder / subtractor and two registers. The subtractor subtracts the values of the candidate and reference block pixels. The MSB of the result of this subtractor selects the functionality of the adder / subtractor unit. If the result of the subtractor unit is negative (MSB = 1), the adder / subtractor unit subtracts the result from the value stored in register R1 and vice versa. After each 4th cycle the accumulated value is loaded into R2 and R1 value is cleared. This means the output of each group of 4 PEs (16 PEs arranged in 4 groups) after summation of the PE outputs in that group gives the SAD value of a 4x4 block. These SAD values are passed through delay registers (D) that are triggered in every 4th cycle. Therefore, after the 16th cycle the SAD values of all the 4x4 candidate blocks are available to the inputs of the routing networks. The routing networks I, II and III are then used to connect these inputs to the four stage adder networks for computing the SAD values of the candidate blocks of other sizes, i.e., 8x4, 4x8, 8x8, 16x8,

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8x16 and 16x16. Therefore, 8 PUs compute all 41 SAD values of 8 16x16 candidate blocks of one row in parallel for each add_h value (Figure 232). This means, each complete cycle of add_h values results all SAD values of 8x4 = 32 16x16 candidate blocks of one row. This is repeated 32 times, controlled by the value of c (Figure 232) to complete motion estimation of the entire search window. add_v in Figure 232 controls the row addresses of the reference and candidate block.

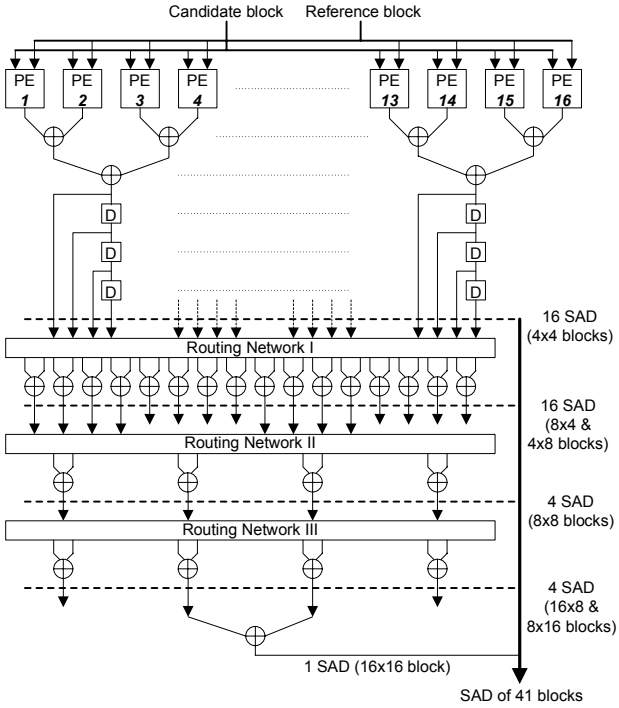


Figure 234 — Processing unit (PU).

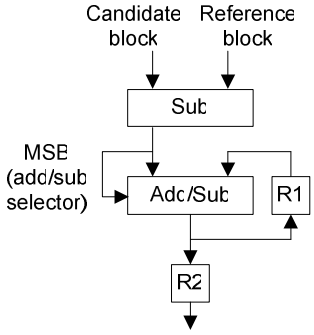


Figure 235 —Processing element (PE).

There are 41 parallel in serial out shift registers, one of which is shown in Figure 236. Each of these takes SAD values of one particular type / size of block from all PUs as inputs and makes them serially available to the comparing unit.

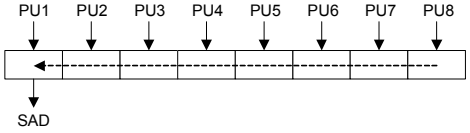


Figure 236 — Parallel in serial out shift registers.

The comparing unit is composed of 41 comparing elements (CE), one of which is shown in Figure 237. Each shift registers output is connected to one of these CEs. CE is composed of one comparator and two registers, one of which stores the minimum SAD for comparison and the other is triggered for storing the motion vector (Hx, Vx) from AG when the input SAD is less than the previous stored minimum SAD value.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

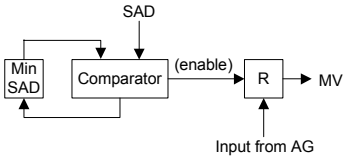


Figure 237 — Comparing element (CE).

The min SAD is initialized with the biggest possible SAD value at the beginning of motion estimation for each reference block. So, the output of the comparing unit gives the motion vectors of all possible candidate blocks (41 in total) at the end of search of the search window. The multiplication and division operations in AG (Figure 232) are implemented by hardwired shifts except add_h*92 for which stored pre-computed values are used. The subtraction and division operations are done for sign and quarter pel adjustments, respectively.

10.19.6.1 Interfaces

N/A.

10.19.6.2 Register File Access

N/A.

10.19.6.3 Timing Diagrams

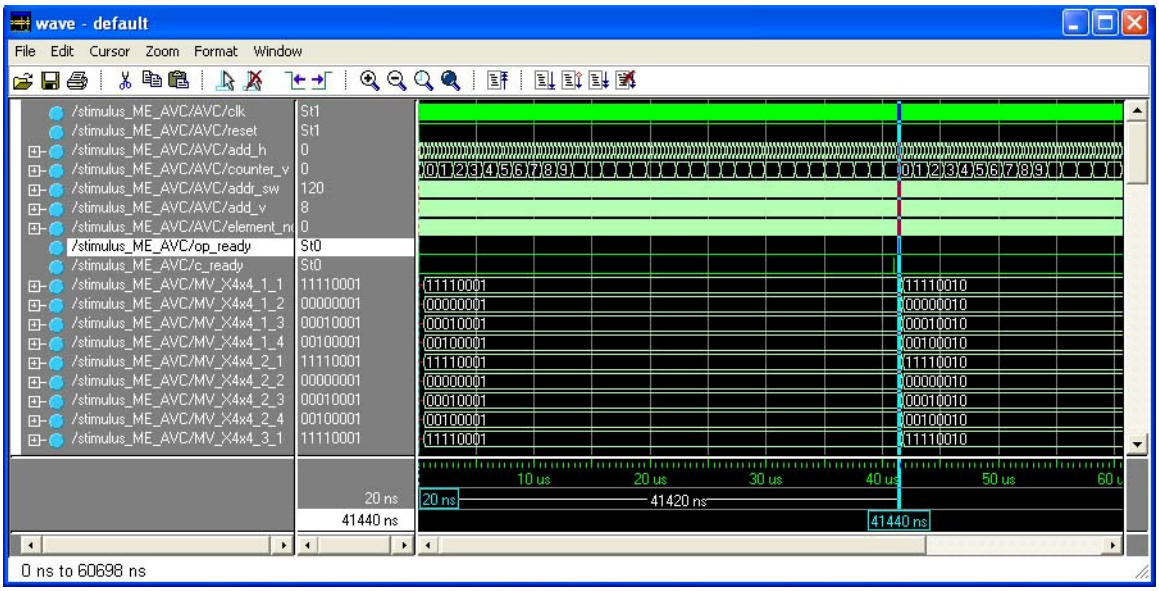


Figure 238 — This diagram shows the latency of the core. Clock period is 20ns for which the core latency is 41420ns. This gives a latency of 2071 clock cycles.

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

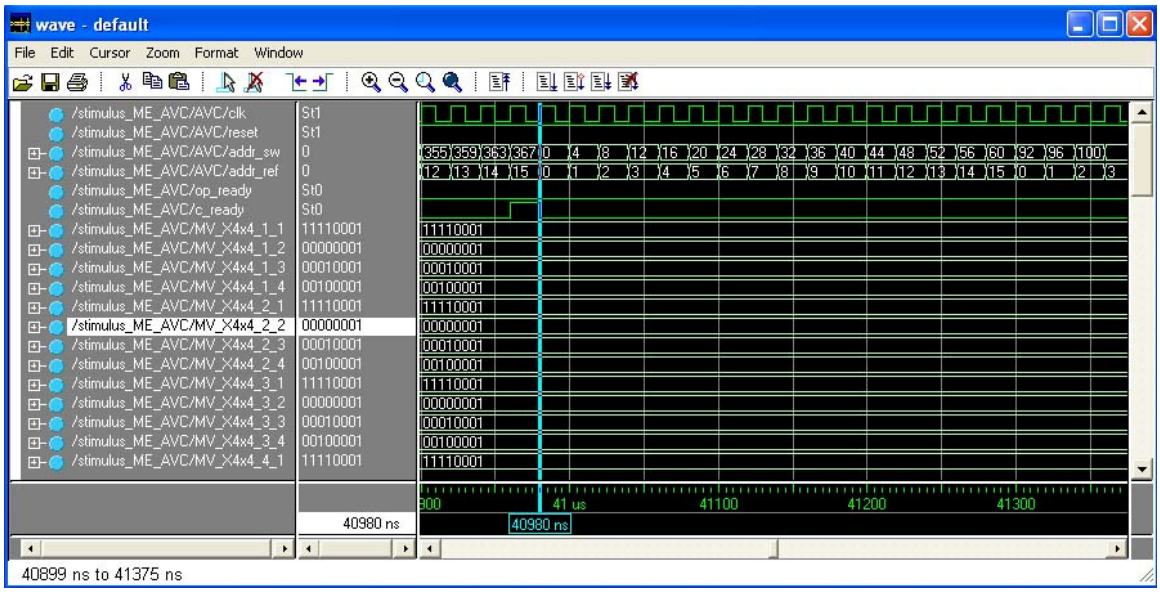


Figure 239 — This diagram shows the generated address locations of reference block memory (addr_ref) and quarter pel interpolated search window memory (addr_sw).

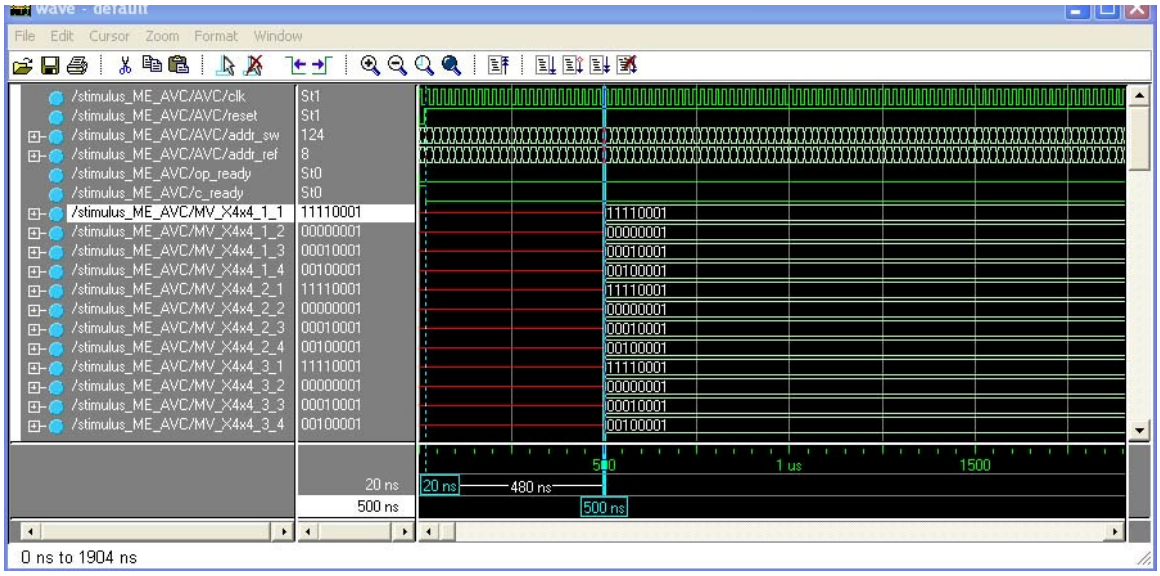


Figure 240 — This diagram shows the setup time of the core. Setup time is 480ns for a clock period of 20ns, i.e., 24 clock cycles.

10.19.7 Results of Performance & Resource Estimation

The architecture has been prototyped in Verilog HDL, simulated and synthesized by Xilinx ISE development tools for Virtex2 device family. The maximum speed was found to be around 150MHz. Simulation result conforms real time processing of CIF (352x288) frame sequences. Under a clock speed of 120MHz, the core can compute in real time the motion vectors of all various size blocks with 5 reference frames.

10.19.8 API calls from reference software

N/A.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.19.9 Conformance Testing

The architecture has not been tested on the platform because of the limited resource availability but its simulation results conforms with the actual results.

10.19.9.1 Reference software type, version and input data set

N/A.

10.19.9.2 API vector conformance

N/A.

10.19.9.3 End to end conformance (conformance of encoded bitstreams or decoded pictures)

N/A.

10.19.10 Limitations

Increased area is the main limitation of the design.

10.20 AN IP BLOCK FOR VARIABLE BLOCK SIZE MOTION ESTIMATION IN H.264/MPEG-4 AVC

10.20.1 Abstract description of the module

This subclause presents an efficient real time variable block size motion estimation (VBSME) IP block. The architecture provides motion vectors for each 16x16 block and its 40 sub-blocks (i.e. 41 motion vectors). The architecture is an SIMD architecture integrated with embedded SRAMs on one chip. The developed IP block is prototyped and simulated using ModelSim 5.4®. It is synthesized using Xilinx ISE 6®. The proposed module processes 31 CIF frames /sec using 122 MHz clock frequency. This module utilizes 68% of the register bits, 16% of the Block RAMs, and 94% of the LUTs in Xilinx Virtex II FPGA XC2V3000-4.

10.20.2 Module specification

10.20.2.1 MPEG 4 part: ***10***
10.20.2.2 Profile : ***Simple profile***
10.20.2.3 Level addressed: ***All***
10.20.2.4 Module Name: ***ME_architecture***
10.20.2.5 Module latency: ***14989 clock cycles***
10.20.2.6 Module data throughput: ***12276*41 motion vector/s using the max clock freq.***
10.20.2.7 Max clock frequency: ***122 MHz***
10.20.2.8 Resource usage:
10.20.2.8.1 LUT: **27089 (94% of the available LUT in XC2V3000-4)**
10.20.2.8.2 Block RAMs: **16 (16% of the available block RAMs in XC2V3000-4)**
10.20.2.8.3 Multipliers: **0**
10.20.2.8.4 External memory: **0**
10.20.2.8.5 Register bits not including I/O 19627 (68% of the available register bits in XC2V3000-4)
10.20.2.9 Revision: ***1.0***
10.20.2.10 Authors: ***Mohammed Sayed and Wael Badawy***
10.20.2.11 Creation Date: ***October 2005***
10.20.2.12 Modification Date:

10.20.3 Introduction

Traditional video coding standards such as MPEG-2 and H.261 use fixed block size motion estimation. In this technique the video frame is divided into non-overlapped square blocks with fixed size and the video motion is

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2009 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

represented by the translational motion of these blocks. However, if the block size is too large it may have areas from more than one moving object and so it fails to represent such type of motion, which increases the coding residues. On the other hand, if the block size is too small, the number of motion vectors increases and so the video coding becomes inefficient. One of the solutions to these contradicting requirements is to use different block sizes. MPEG-4 and H.263 tried to solve that problem by using two block sizes 16x16 or 8x8 pixels but these two options are not enough.

MPEG-4 Advanced Video Coding and the emerging video coding standard AVC [13] adopt variable block size motion estimation. They use seven different block sizes for better motion representation. This technique improves the video quality and the transmission bit rate at the cost of increasing the computational cost of video coding. The seven used block sizes are 16x16, 16x8, 8x16, 8x8, 8x4, 4x8, and 4x4 as shown in Figure 241.

In block-based motion estimation the block matching algorithm (BMA) is used, where a searching process is done for each block in the current frame to find the best matching block in a reference frame. Then the block motion vector is estimated from the position difference between the two matched blocks. The BMA suffers from high computational cost and high memory requirements. To reduce the computational cost, the searching process is usually limited to certain search area. Different search strategies have been proposed in the literature, such as full search, three-step search [27], and cross search [27]. Full search strategy produces high video quality but it suffers from high computational cost and high memory requirements. In addition, different matching criteria can be used among which the sum of absolute difference (SAD) is the most common in motion estimation architectures [27] due to its simplicity and suitability for VLSI implementation. Equation 77 and Equation 78 show the SAD matching criterion:

$$SAD(d_1, d_2) = \sum_{(n_1, n_2) \in B} \left| s_1(n_1, n_2, k) - s_2(n_1 + d_1, n_2 + d_2, k+1) \right|$$

Equation 77

$$\left[d_1 \; d_2 \right]^T = \arg \min_{(d_1, d_2)} SAD(d_1, d_2)$$

Equation 78

Using the BMA for each of the sub-blocks shown in Figure 241 separately leads to a huge implementation cost in terms of gates count and silicon area. To reduce this implementation cost the SAD values for the smallest blocks (i.e. 4x4) can be used to compute the SAD values for the larger blocks by adding them together. The proposed architecture builds on our previously presented architecture for fixed block size motion estimation. The architecture presented estimates 41 motion vectors for the 41 sub-block shown in Figure 241 instead of estimating one motion vector for 8x8 block size.

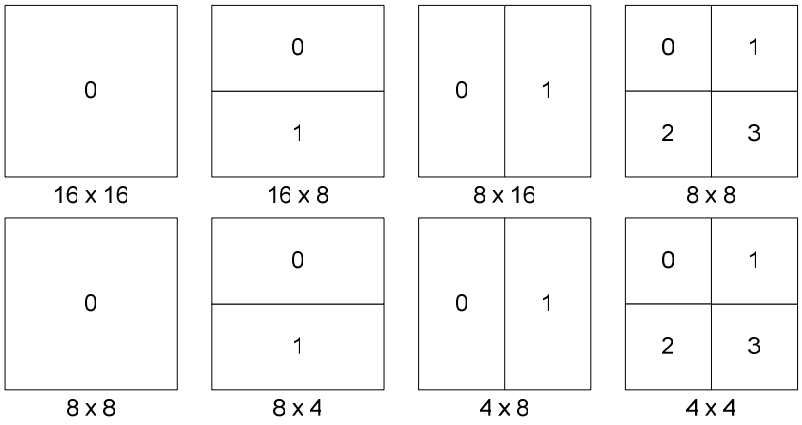


Figure 241 — The various block sizes in H.264/AVC.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

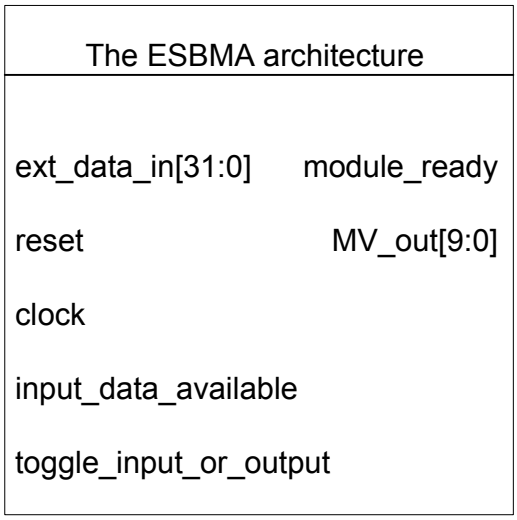
10.20.4 Functional Description

10.20.4.1 Functional description details

The architecture processes CIF format video sequences (i.e. 352x288 pixels frame size) with 16x16 pixels block size and ±15 pixels search range. It supports 7 block sizes to enhance the video quality as required by the AVC standard. The different partitions of the 16x16 microblock are shown in Figure 241. The architecture estimates motion vectors for each 16x16 block and its 40 sub-blocks (i.e. 41 motion vectors) with ±15 search range. Which sub-blocks will be selected or combined together (e.g. 8x4 blocks with 4x4 blocks) is out of the architecture scope. The architecture uses the full search block matching algorithm with sum of absolute difference (SAD) matching criterion.

10.20.4.2 I/O Diagram

Table 67 — I/O Diagram.



10.20.4.3 I/O Ports Description

Table 68 — Port description.

Port Name	Port width	Direction	Description
ext_data_in	32	Input	External data input
Reset	1	Input	Reset signal
Clock	1	Input	Clock
input_data_available	1	Input	Inform the module that the input data is available
toggle_input_or_output	1	Input	Change the input type from SW to RB or change the output type from horizontal MV to vertical MV
module_ready	1	Output	Motion vectors ready
MV_out	10	Output	The motion vectors output

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.20.4.3.1 Parameters (generic)

N/A.

10.20.4.3.2 Parameters (constants)

N/A.

10.20.5 Algorithm

The architecture implements the exhaustive search block matching algorithm with ±15 pixels search range and 16x16 pixels block size. Each 16x16 block has 41 sub-blocks as shown in Figure 241.

10.20.6 Implementation

The architecture processes CIF format video sequences (i.e. 352x288 pixels frame size) with variable block sizes and ±15 pixels search range. 16x16 is the largest block size while 4x4 is the smallest one. The architecture consists of an embedded SRAM with 46x46 bytes size, 31 processing elements, one reference block memory, and 16 comparison units as shown in Figure 242. The architecture searches for the 41 sub-blocks with the minimum SADs in a 46x46 search window and generates horizontal and vertical motion vector for each block. The architecture reads the search window texture from the reference frame and the 16x16 reference block texture from the current frame and stores them in the local memories.

The processing elements are working in parallel as a (SIMD) architecture. They compute the SAD values for the candidate blocks. The SAD comparison units find the minimum SAD and estimate the horizontal and vertical motion vectors. A search window of 46x46 pixels and a block of 16x16 pixels are used, which means with full-search BMA there are 961 (31x31) candidate blocks for each sub-block. Accessing the search window memory is done row by row. This means that the reference block is compared with all the candidate blocks in the first row simultaneously. Then it is compared with all the candidate blocks in the second row simultaneously and so on for the following rows of candidate blocks. The reference block pixels are feed row by row to the processing elements as shown in Figure 242. One processing element is used per each column of candidate blocks. The 46 memory columns are connected to the 31 processing elements according to the connection configuration shown in Table 69.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

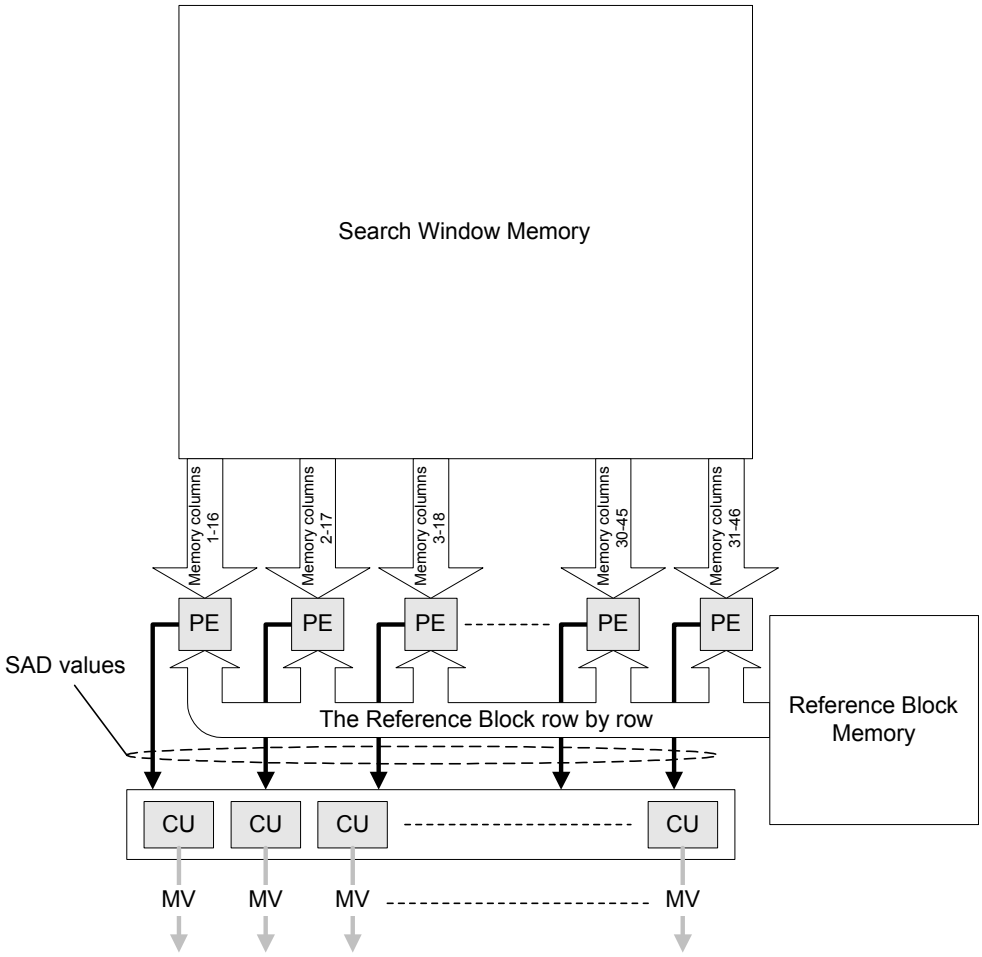


Figure 242 — Block diagram of the architecture.

The processing element consists of four subtractors, four accumulators and one adder as shown in Figure 243. The processing element has two inputs one row from the candidate block and one row from the reference block. To accumulate the absolute value needed in the SAD computations, the accumulators add or subtract the subtractors' outputs according to their sign bit. The 16 SAD values for the 4x4 blocks are computed in four stages four in each stage as can be noted from the processing element block diagram. Then at each stage the four SAD values are used to find the SAD values for larger blocks.

Table 69 — The memory columns and the processing elements connection configuration.

Processing element	The connected memory columns
PE1	1,2,3,…,16
PE2	2,3,4,…,17
PE3	3,4,5,…,18
PE4	4,5,6,…,19
PE5	5,6,7,…,20
PE6	6,7,8,…,21

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

PE7	7,8,9,…,22
PE8	8,9,10,…,23
PE9	9,10,11,…,24
PE10	10,11,12,…,25
PE11	11,12,13,…,26
PE12	12,13,14,…27
PE 13	13,14,15,…,28
PE14	14,15,16,…,29
PE15	15,16,17,…,30
PE16	16,17,18,…,31
PE17	17,18,19,…,32
PE18	18,19,20,…,33
PE19	19,20,21,…,34
PE20	20,21,22,…,35
PE21	21,22,23,…,36
PE22	22,23,24,…,37
PE23	23,24,25,…,38
PE24	24,25,26,…,39
PE25	25,26,27,…,40
PE26	26,27,28,…,41
PE27	27,28,29,…,42
PE28	28,29,30,…,43
PE29	29,30,31,…,44
PE30	30,31,32,…,45
PE31	31,32,33,…,46

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

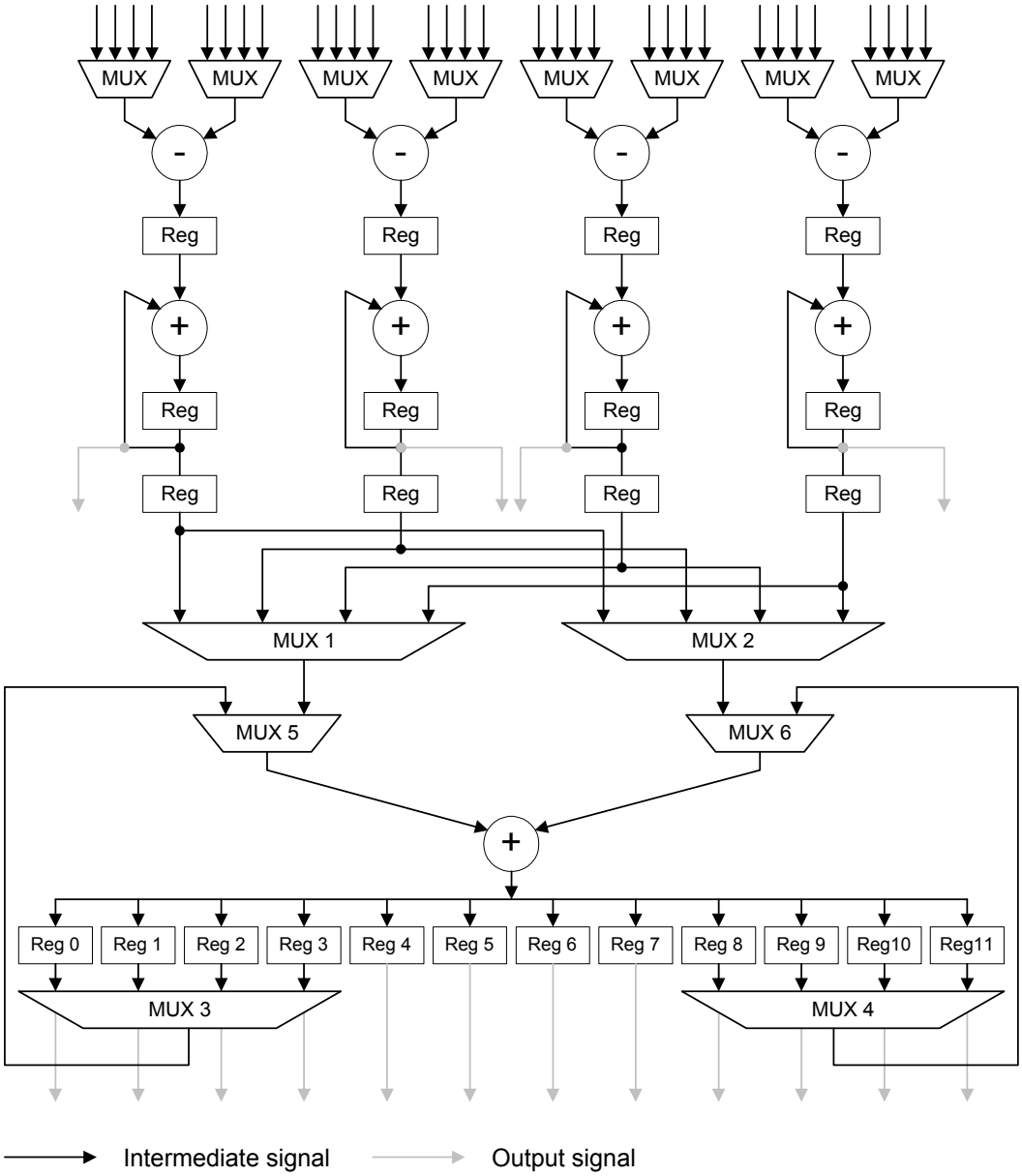


Figure 243 — Block diagram of one processing element.

Figure 244 shows the blocks partitioning which can be used with Table 70 to explain the SAD computations sequence of the 41 sub-blocks. For example, in stage 1 after finding the SAD values for blocks 0,1,2, and 3 of the 4x4 blocks, the SAD values for blocks 0 and 1 of the 8x4 blocks are computed and the 4x4 SAD values are stored to be used in the next stage to find the SAD values of blocks 0,1,2,3 of the 4x8 blocks. The computing sequence of the 41 SAD values over the four stages is shown in Table 70. In Table 70 the SAD values for number of the blocks are computed in two stages in the first stage the SAD value of the upper half is stored to be used in the second stage to find the SAD value of the whole block.

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

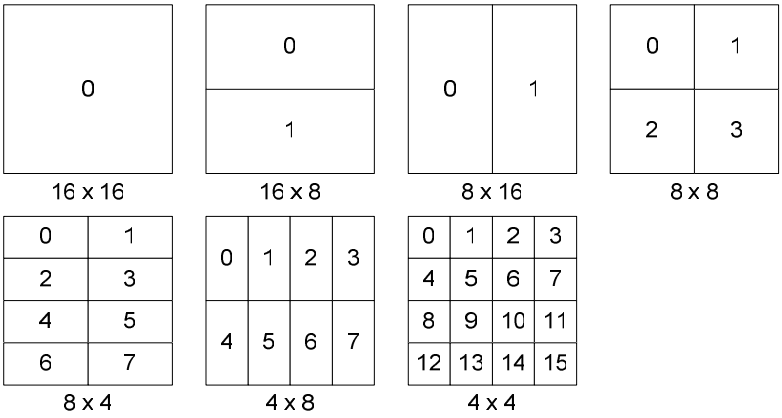


Figure 244 — The blocks partitioning.

Table 70 — The computing sequence of the 41 sad values over the four stages.

Stage	4x4	4x8	8x4	8x8	8x16	16x8	16x16
1	0,1,2,3	Half 0,1,2,3	0,1	-	-	-	-
2	4,5,6,7	0,1,2,3	2,3	0,1	Half 0,1	0	-
3	8,9,10,11	Half 4,5,6,7	4,5	-	-	-	-
4	12,13,14,15	4,5,6,7	6,7	2,3	0,1	1	0

The SAD comparison part of the architecture has 16 SAD comparison units (CUs). The SAD CUs are classified as follows: 4 CUs for the 4x4 blocks, 2 CUs for the 8x4 blocks, 4 CUs for the 4x8 blocks, 2 CUs for the 8x8 blocks, 1 CU for the 16x8 blocks, 2 CUs for the 8x16 blocks and 1 CU for the 16x16 blocks. This classification is due to the four processing stages explained in Table II. For example, the SAD values of the 4x8 blocks are computed four by four blocks (i.e. blocks 0,1,2,3 then blocks 4,5,6,7), such that the SAD values of four blocks are ready simultaneously so four CUs are needed.

The CU compares between the 31 SAD values computed by the processing elements and generates the required horizontal and vertical motion vectors. It has 31 comparison node distributed over five levels as shown in Figure 245. Each comparison node has two inputs (i.e. two SAD values) from the upper level and it compares between them and generates 1-bit result signal and it also passes the lesser SAD value to the lower level. The comparison results from the five levels are connected to the multiplexer structure shown in Figure 246. These multiplexers are used to ascend over the comparison tree to generate the horizontal motion vector. The generated motion vector is unsigned (i.e. from 1 to 31). The sign adjustment unit subtracts 16 from the unsigned horizontal motion vector to generate a final motion vector between -15 and +15.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2009 – All rights reserved
Not for Resale

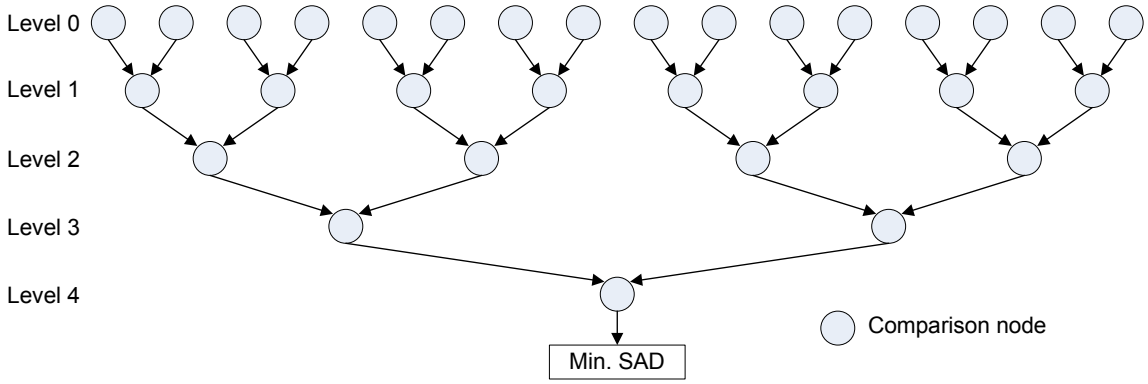


Figure 245 — The comparison tree of one SAD comparison unit.

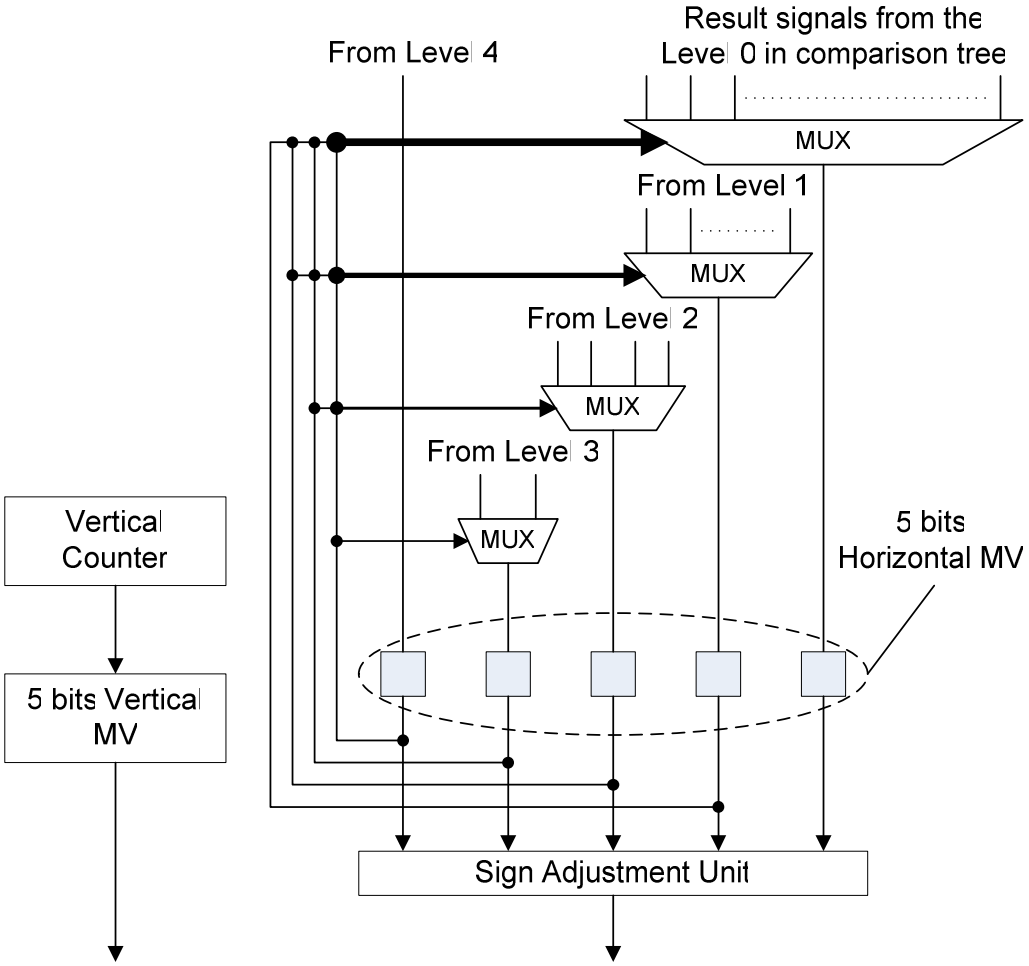


Figure 246 — Generating the horizontal and the vertical motion vectors.

The operation of the proposed architecture can be explained as follows: For each 16x16 block in the current frame (i.e. reference block), 1) read and store the corresponding search window texture in the search window memory, 2) read and store the 16x16 reference block texture in the reference block memory, 3) compute the SAD values for the 41 sub-block in one row of 31 candidate blocks, 4) find the blocks with minimum SAD values among the computed SAD values, 5) repeat steps 3 and 4 for all the candidate blocks (row by row), 6)

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

generate the horizontal and the vertical motion vectors for the 41 sub-blocks. Figure 247 shows a flow chart describes the operation of the proposed architecture.

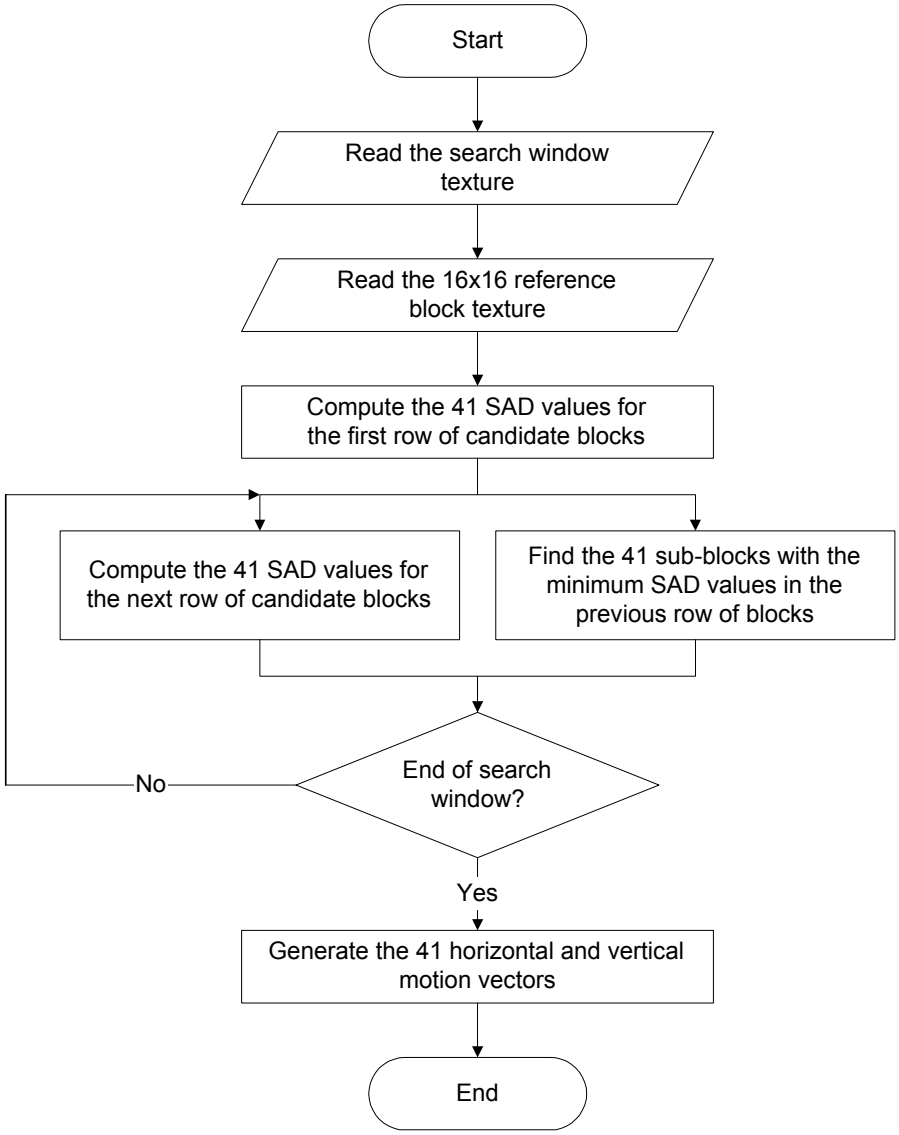


Figure 247 — Flow chart describes the operation of the architecture.

10.20.6.1 Interfaces

N/A.

10.20.6.2 Register File Access

N/A.

10.20.6.3 Timing Diagrams

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

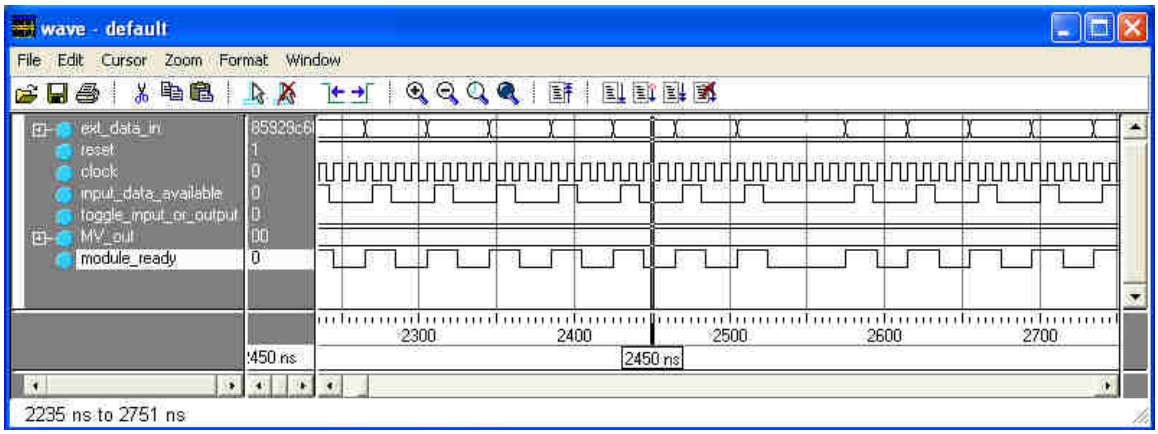


Figure 248 — Writing the search window texture.

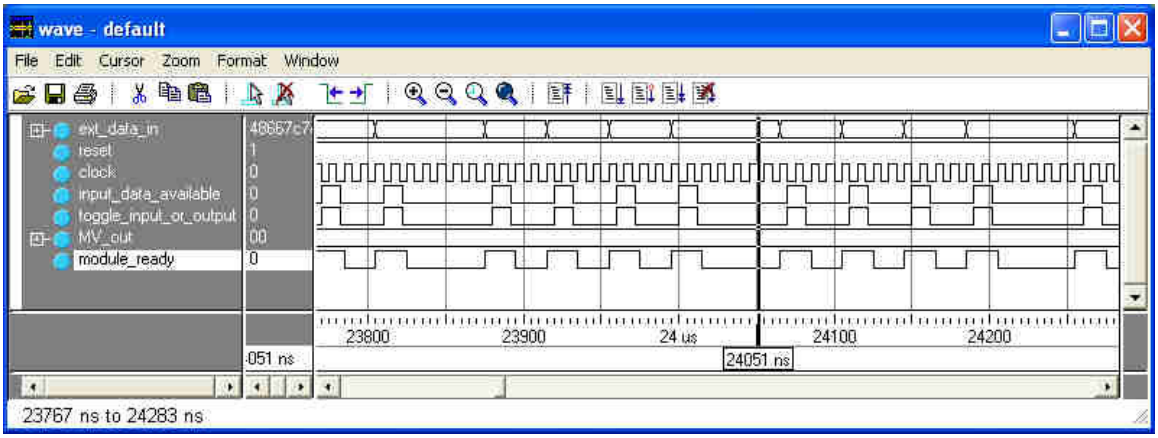


Figure 249 — Writing the reference block texture.

10.20.7 Results of Performance & Resource Estimation

The module has been prototyped, simulated and synthesized for Xilinx Virtex II FPGA XC2V3000-4. Using the max clock frequency (122 MHz), the proposed architecture needs 81.4 µs to generate the required 41 motion vectors for one 16x16 block. This module utilizes 68% of the register bits, 16% of the Block RAMs, and 94% of the LUTs in Xilinx Virtex II FPGA XC2V3000-4, which is the processing element in the Annapolis Wildcard II. The proposed architecture processes one CIF video frame (i.e. 352x288 pixels) in 32.23 ms such that it can process up to 31 CIF video frames per second.

10.20.8 API calls from reference software

N/A.

10.20.9 Conformance Testing

The architecture has not been tested on the platform because of the limited resource availability but its simulation results conforms with the software results.

10.20.9.1 Reference software type, version and input data set

N/A.

10.20.9.2 API vector conformance

N/A.

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.20.9.3 End to end conformance (conformance of encoded bitstreams or decoded pictures)

N/A.

10.20.10 Limitations

N/A.

10.21 An IP Block for MPEG-4 Part 10 AVC Deblocking Filter

10.21.1 Abstract

This subclause describes a module that implements the deblocking filter used in MPEG-4 AVC [13] video coding standard. The deblocking filter and the motion estimator are two of the most computational expensive modules of MPEG-4 AVC. This module is running in a dedicated hardware namely on Xilinx VIRTEX™-II FPGA [1].

10.21.2 Module specification

10.21.2.1 MPEG 4 part: **10**
10.21.2.2 Profile : **All**
10.21.2.3 Level addressed: **All**
10.21.2.4 Module Name: **Deblocking**
10.21.2.5 Module latency: **17**
10.21.2.6 Module data throughput: **10 Mblocks/sec**
10.21.2.7 Max clock frequency: **40MHz**
10.21.2.8 Resource usage:
10.21.2.8.1 LUTs **1456**
10.21.2.8.2 Multipliers **0**
10.21.2.8.3 BRAMs **3**
10.21.2.9 Revision: **1.0**
10.21.2.10 Authors: **M. Santos, A. Silva, A. Navarro**
10.21.2.11 Creation Date: **19/04/2006**
10.21.2.12 Modification Date: **19/04/2006**

10.21.3 Introduction

The H.264/MPEG-4 AVC is a block based video coding standard and therefore producing some blocking artifacts [46]. Also, the motion compensated prediction is a source of those blocking artifacts that could result in disturbing video anomalies. Those type of artifacts are also typical of others block based video coding techniques. Despite, in H.264/MPEG-4 AVC the IDCT is a 4x4 sample transform that reduces in some degree the block effect, the use of a deblocking filter can significantly increase the coding complexity.

As in H.264/MPEG-4 AVC, the deblocking filter is inside the coding loop, it is mandatory that all 4x4 blocks should be filtered by the deblocking filter. This requires some intense computations mainly because the filter is an adaptive filter. So, to reduce some computation effort from the main processor the purposed module performs the H.264/MPEG-4 AVC deblocking filter in a FPGA.

The module receives all the unfiltered pixels of a block and then performs the deblocking filtering based on the parameters introduced into the module. After 17 clocks latency the module signals that the deblocking is done and is ready for a new deblocking process.

10.21.4 Functional Description

The deblocking filter process is started when the START input is set to high. Then the module collects all the blocks pixels and the parameters regarding the filtering conditions. The collected pixels are stored in RAMs in a strategic manner like in [47]. In this way, and because the filter is based in combinational logic, only the clocks cycles used to store and load into the RAMs are accounted for the latency of the module.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Firstly, the block horizontal boundaries are filtered. The module loads the pixels information from one of the 3 RAMs and passes all pixels to the filter sub-module and the results are stored in a buffer. This buffer is especially build to store those results in a manner that when the module performs the vertically boundary filtering and requires new pixels, they are already stored in the right form. In another words, if a normal RAM was used after the horizontal filtering process, all the filtered pixels should be read again and stored in a way that filter sub-module now receives the vertically boundaries. Following this approach, a lot of clock cycles were avoided.

The filtering process is controlled by finite state machines (FSM) and some look-up-tables (LUT) are used to reduce the logic needed for calculations. When the module performs all the horizontally and vertically filtering, it signals it and is ready to start a new deblocking process.

10.21.4.1 I/O Diagram

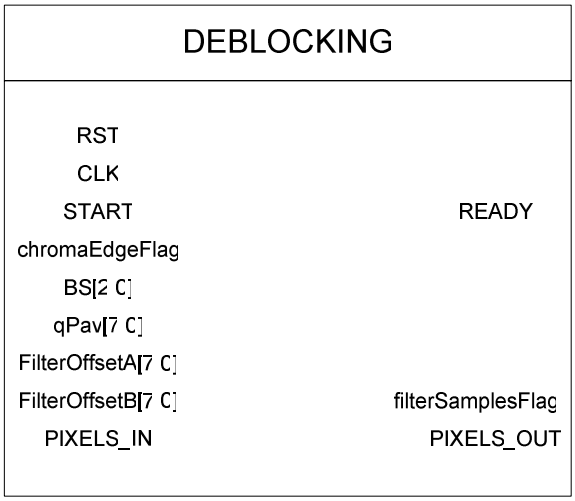


Figure 250 — I/O diagram.

I/O Ports Description

- RST – at level high, it resets all the flip-flops and state machines on the module.
- CLK – system clock.
- START – is asserted high to start the deblocking process. It must be held at high level during block data loading.
- chromaEdgeFlag – selection between luma pixels (low) or chroma pixels (high). It should be stable for the entire pixel load.
- BS[2:0] – defines the boundaries strength.
- qPav[7:0] – defines the average quantization parameter.
- FilterOffsetA[7:0] – filter offset for addressing the α and t_{c0} parameters.
- FilterOffsetB[7:0] – filter offset for addressing the β parameter.
- PIXELS_IN – is the port where the pixel are loaded. This is a 64 bit port with that is split in two 32 bit sections, the PIXEL_IN_P section and the PIXEL_IN_Q section. The PIXEL_IN_P section is composed by 4 eight bit pixels corresponding to the left pixels boundary. And the PIXEL_IN_Q section is composed by 4 eight bit pixels corresponding to the right pixels boundary.
- PIXELS_OUT – is the port where the filtered pixels are read. This is a 64 bit port with that is split in two 32 bit sections, the PIXEL_OUT_P section and the PIXEL_OUT_Q section. The PIXEL_OUT_P section is composed by 4 eight bit pixels corresponding to the left pixels boundary. And the PIXEL_OUT_Q section is composed of 4 eight bit pixels corresponding to the right pixels boundary.
- filterSamplesFlag – if high means that the pixels were filtered.
- READY – valid data pixels at PIXELS_OUT[8:0] when READY is high.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2009 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.21.4.2 I/O Ports Description

Port Name	Port Width	Direction	Description
RST	1	Input	Reset the module
CLK	1	Input	Clock to load the input stream in VLC
START	1	Input	Start deblocking if START is high
chromaEdgeFlag	1	Input	H.264/MPEG-4 AVC chromaEdgeFlag
BS	3	Input	Boundary strength
qPav	8	Input	H.264/MPEG-4 AVC average quantization parameter
FilterOffsetA	8	Input	H.264/MPEG-4 AVC FilterOffsetA
FilterOffsetB	8	Input	H.264/MPEG-4 AVC FilterOffsetB
PIXELS_IN	64	Input	Pixels data in
PIXELS_OUT	64	Output	Filtered pixels data out
filterSampleFlag	1	Output	H.264/MPEG-4 AVC filterSampleFlag
READY	1	Output	Valid data pixels at PIXELS_OUT

10.21.5 Algorithm

The following figure describes the algorithm employed.

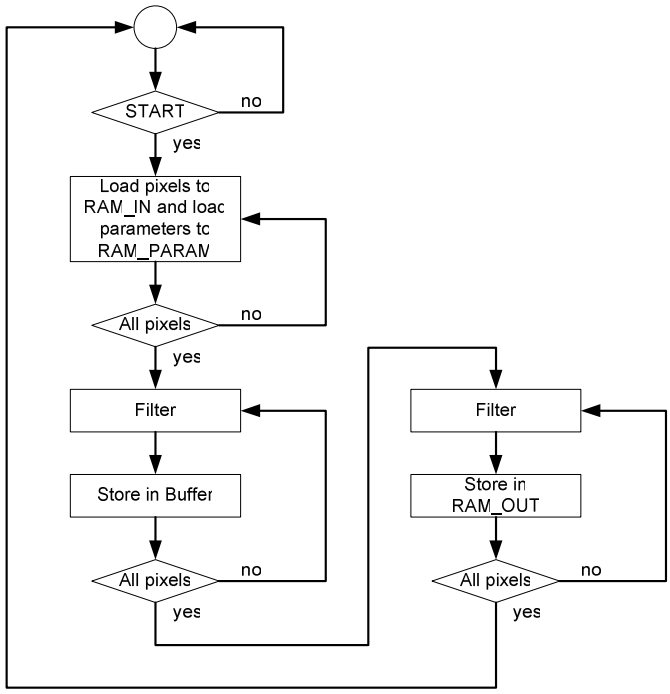


Figure 251 — Deblocking algorithm.

When START is high the deblocking process begins. The module will store the incoming pixels in a strategic way in the RAM_IN that is a 32 bits width RAM. The strategic way consists in storing in each RAM position the 8 pixels (or 4 pixels if it is a luma block) corresponding to the horizontal boundary in this order p3, p2, p1, p0,

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

q0, q1, q2 and q3. Note that p3, p2, p1, p0, q0, q1, q2 and q3 are the same has defined in [13] for the deblocking filter. Besides, in parallel, the parameters for each block are stored in the RAM_PARAM.

When all the pixels are loaded, the module applies the filter sub-module to all the RAM_IN position according to the data in RAM_PARAM to do the horizontal filtering boundary process. Then, the result is stored in a special buffer. The buffer stores the data in such a way that the pixels data is already organized for the vertically filter boundaries. Following this approach, the module saves clock cycles required to re-load the pixel data for the vertical filtering process.

After the horizontal filtering is done, the module starts the vertically filtering boundaries to all the pixels boundaries and stores the results into the RAM_OUT memory. The vertical filtering uses the same filter sub-module like the horizontal filtering, but in the later case, the data fed into the filter is in the buffer and the output result is stored into the RAM_OUT.

10.21.6 Implementation

The following figure shows the block diagram of the Deblocking module.

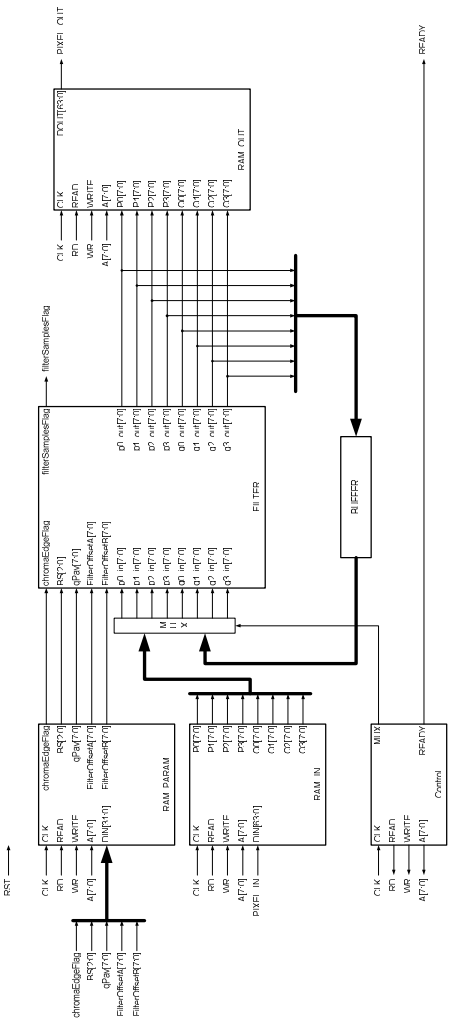


Figure 252 — Deblocking block digram.

© ISO/IEC 2009 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.21.6.1 Interfaces

Next figure shows the interfaces of the implemented module.

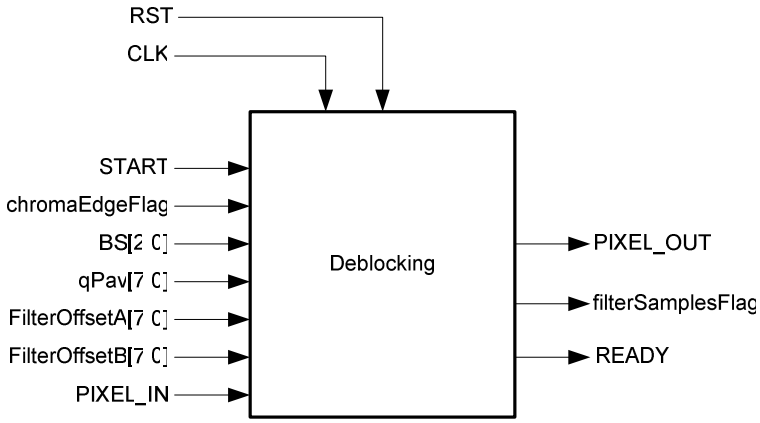


Figure 253 — Deblocking module interfaces.

- RST – at level high, it resets all the flip-flops and state machines on the module.
- CLK – system clock.
- START – is asserted high to start the deblocking process. And it must high for the load of all pixel data.
- chromaEdgeFlag – selection between luma pixels (low) or chroma pixels (high). It should be stable for the entire pixel load.
- BS[2:0] – defines the boundaries strength.
- qPav[7:0] – defines the average quantization parameter.
- FilterOffsetA[7:0] – filter offset for addressing the α and t_{c0} parameters.
- FilterOffsetB[7:0] – filter offset for addressing the β parameter.
- PIXELS_IN – is the port where the pixel are loaded. This is a 64 bit port with that is split in two 32 bit sections, the PIXEL_IN_P section and the PIXEL_IN_Q section. The PIXEL_IN_P section is composed by 4 eight bit pixels corresponding to the left pixels boundary. And the PIXEL_IN_Q section is composed by 4 eight bit pixels corresponding to the right pixels boundary.
- PIXELS_OUT – is the port where the filtered pixels are read. This is a 64 bit port with that is split in two 32 bit sections, the PIXEL_OUT_P section and the PIXEL_OUT_Q section. The PIXEL_OUT_P section is composed by 4 eight bit pixels corresponding to the left pixels boundary. And the PIXEL_OUT_Q section is composed by 4 eight bit pixels corresponding to the right pixels boundary.
- filterSamplesFlag – if high means that the pixels were filtered.
- READY – valid data pixels at PIXELS_OUT[8:0] when READY is high.

10.21.6.2 Timing Diagrams

The timing diagram of the implemented module is shown in the next figure.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

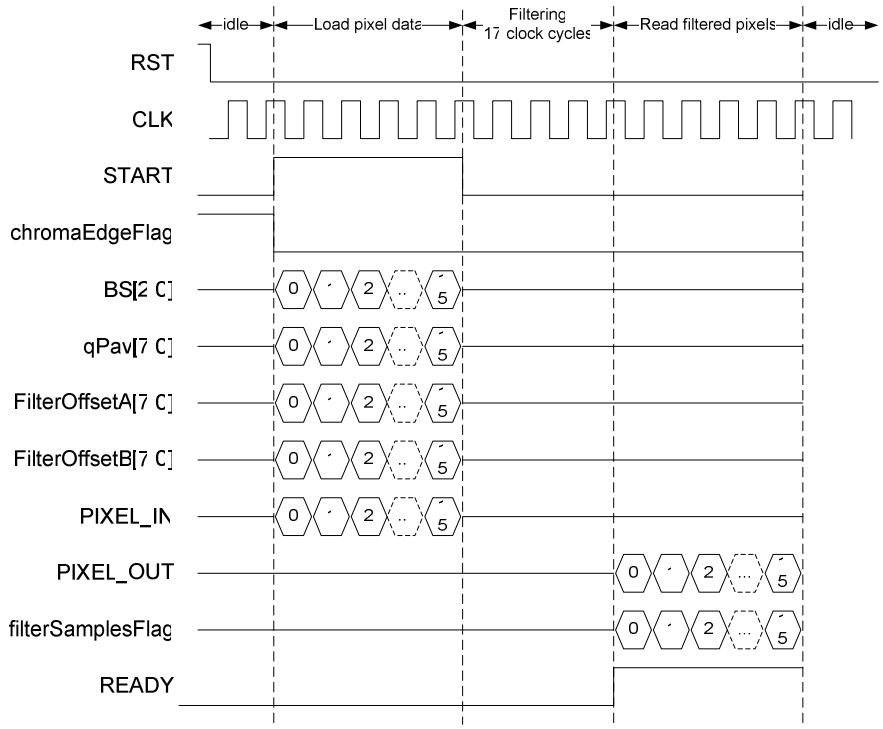


Figure 254 — Timing diagram.

The timing diagram, above, shows an example of a deblocking filtering of chroma pixels (chromaEdgeFlag = 0). In the first 16 clock cycles and when START is high, the pixels are loaded into the RAM_IN memory. After loading, the filtering process takes place. This process needs 17 clock cycles to compute all filtered pixels.

When the filtering process is done, the module sets the READY signal into the high state, READY = 1. Thus, the module is ready to retrieve the filtered pixels.

10.21.7 Results of Performance & Resource Estimation

The design tool used in this work was ISE 7.1i. The obtained report from the synthesis tool, XST, is presented below:

Device utilization summary:

Selected Device : 2v3000fg676-4

Number of LUTs: *1456 out of 28672 5%*

Number of BRAMs: *3 out of 96 3%*

Number of Multipliers: *0 out of 96 0%*

Timing Summary:

Speed Grade: -4

Minimum period: 25ns (Maximum Frequency: 40MHz)

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.21.8 API calls from reference software

```
DWORD WC_DEBLOCKING(WC_Struct wildcard, int *p, int *q, int chromaEdgeFlag, int BS, int
qPav, int FilterOffsetA, int FilterOffsetB) {

    DWORD WC_rc;

DWORD config, ready, start;

config = (chromaEdgeFlag << 28) | (BS << 24)

| (qPav << 16)

| (FilterOffsetB << 8)

| (FilterOffsetA);

WC_rc = WC_PeRegWrite(wildcard.DeviceNum, 0x80, 1, &config);



WC_rc = WC_PeRegWrite(wildcard.DeviceNum, 0x0, 8, &p);

WC_rc = WC_PeRegWrite(wildcard.DeviceNum, 0x8, 8, &q);



start = 1;

WC_rc = WC_PeRegWrite(wildcard.DeviceNum, 0x10, 1, &start);



ready = 0;

while (ready == 0) {

WC_rc = WC_PeRegRead(wildcard.DeviceNum, 0x11, 16, &ready);

}



    WC_rc = WC_PeRegRead(wildcard.DeviceNum, 0x0, 8, &p);

    WC_rc = WC_PeRegRead(wildcard.DeviceNum, 0x8, 8, &q);

return WC_rc;

}
```

10.21.9 Conformance Testing

10.21.9.1 Reference software type, version and input data set

The reference software used was JM 10.2 and the test sequence was test.264.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.21.9.2 API vector conformance

The test vectors used are AVC QCIF test sequences.

10.21.9.3 End to end conformance (conformance of encoded bitstreams or decoded pictures)

Command line: ldecod

Output YUV file: test_dec.yuv

10.21.10 Limitations

The maximum operate frequency of the module is limited by the filter sub-module because it is only composed of combinational logic. This sub-module has a relative long path that reduces the maximum clock rate. Anyway, the proposed approach results in a very low resource usage of the FPGA and the filtering filter needs only 17 clock cycles, neglecting the required time to load 4x4 pixels into the H.264/AVC deblocking filter.

10.22 A HW BLOCK FOR MPEG-4 PART 10 AVC CONTEXT ADAPTIVE VARIABLE LENGTH CODING (CAVLC)

10.22.1 Abstract

This subclause presents a Verilog model for the H.264/AVC context adaptive variable length coding (CAVLC). The proposed architecture uses a split and shared VLC look up table for minimizing the implementation area. The architecture is prototyped in Verilog, simulated and synthesized by ModelSim and Xilinx ISE development tools, respectively. The target device for synthesis was Xilinx Virtex II FPGA (2v3000fg676-4).

10.22.2 Module specification

10.22.2.1 MPEG 4 part:	**10**
10.22.2.2 Profile :	**All**
10.22.2.3 Level addressed:	**All**
10.22.2.4 Module Name:	**HW_CAVLC**
10.22.2.5 Module latency:	**18 clock cycles**
10.22.2.6 Module data throughput:	**715.35 ns (avg.) per 4x4 blocks of quantized transform coefficients at max clock frequency.**
10.22.2.7 Max clock frequency:	**60.11 MHz**
10.22.2.8 Resource usage:	
10.22.2.8.1 CLB Slices:	**866**
10.22.2.8.2 DFFs or Latches:	**576**
10.22.2.8.3 LUTs:	**1,533**
10.22.2.8.4 Bonded IOBs:	**143**
10.22.2.8.5 Number of Gates:	**15,363**
10.22.2.9 Revision:	**1.00**
10.22.2.10 Authors:	**Choudhury A. Rahman and Wael Badawy**
10.22.2.11 Creation Date:	**August 2006**
10.22.2.12 Modification Date:	**August 2006**

10.22.3 Introduction

The growing demand in video quality and low bit-rate raises the need of new video standards. Therefore, the new standards are emerging with more complex compression and coding algorithms. Software implementation of these coders often does not satisfy the real-time processing requirements. Therefore, a HW or HW/SW co-design based coders are gaining popularity.

The Joint Video Team (JVT) was formed in 2001 to collaborate between the ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC 14496-10 AVC Moving Pictures Experts Group (MPEG). The JVT finalized the new video coding standard H.264/Advance Video Coding (AVC) in May 2003 [14]. The H.264/AVC

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

achieves 39%, 49% and 64% bit-rate reduction compared to the prior standards: MPEG-4 [4], H.263 [49] and MPEG-2 [45], respectively [50]. This coding gain comes mainly from prediction. Inter prediction enhanced by motion estimation with quarter pixel accuracy, multiple reference frames and variable block sizes have significantly reduced the bit-rate of the new standard. Moreover, the intra prediction has also lifted the coding performance. The advanced entropy coding method (CAVLC or CABAC) used in the standard further improves the performance.

The CAVLC entropy coding method is used for coding quantized transform coefficients of the residual image. It is highly context adaptive that results in very good coding efficiency. This subclause presents a split and shared VLC look up table (for minimizing the implementation area) based real-time CAVLC architecture. The architecture is prototyped in Verilog, simulated and synthesized by ModelSim and Xilinx ISE development tools, respectively.

10.22.4 Functional Description

10.22.4.1 Functional description details

The proposed architecture processes 4×4 blocks of quantized transform coefficients of the residual images. It takes N_U, N_L, N_U_valid and N_L_valid as parameters (refer to subclause 10.22.4.3) for computing the CAVLC stream. The architecture outputs the CAVLC stream serially through the output port Op_data.

10.22.4.2 I/O Diagram

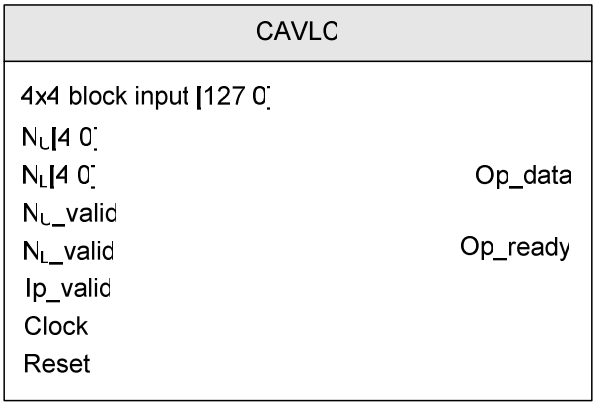


Figure 255 — I/O Diagram.

10.22.4.3 I/O Ports Description

Table 71 — I/O Ports Description.

Port Name	Port Width	Direction	Description
4x4 block input	128	Input	Quantized 4x4 block of transform coefficient input.
N_U	5	Input	The number of total coefficients in the upper block
N_L	5	Input	The number of total coefficients in the left hand block
N_U_valid	1	Input	Flag indicating that upper block is available

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

N_L_valid	1	Input	Flag indicating that left hand block is available
Ip_valid	1	Input	Flag indicating that input is valid
Clock	1	Input	System clock
Reset	1	Input	System reset
Op_data	1	Output	Serial output of the encoded bit-stream.
Op_ready	1	Output	Flag indicating that output is valid

10.22.5 Algorithm

In the proposed H.264/AVC standard, two entropy coding methods are included, namely context-based adaptive variable length coding (CAVLC) [14][52][28] and context-based adaptive binary arithmetic coding (CABAC) [53]. Both of these two coding methods adaptive features improve performance relative to prior standard designs.

Figure 256 shows the block diagram of a H.264/AVC encoder. The entropy coding block is the last block in this diagram that generates bit-stream of the encoded residual. It is the lossless part of the encoder and is based on assigning shorter codewards to symbols that occur most frequently and longer codewards to symbols that occur less frequently. Depending on entropy coding mode either CAVLC or CABAC can be chosen. If CAVLC is chosen residual block data (quantized transform coefficients) are coded using CAVLC and other variable length coded units are coded using Exp-Golomb codes.

CAVLC uses multiple VLC tables. The VLC tables are switched depending on previously coded elements. This results improved coding efficiency compared to using single VLC table.

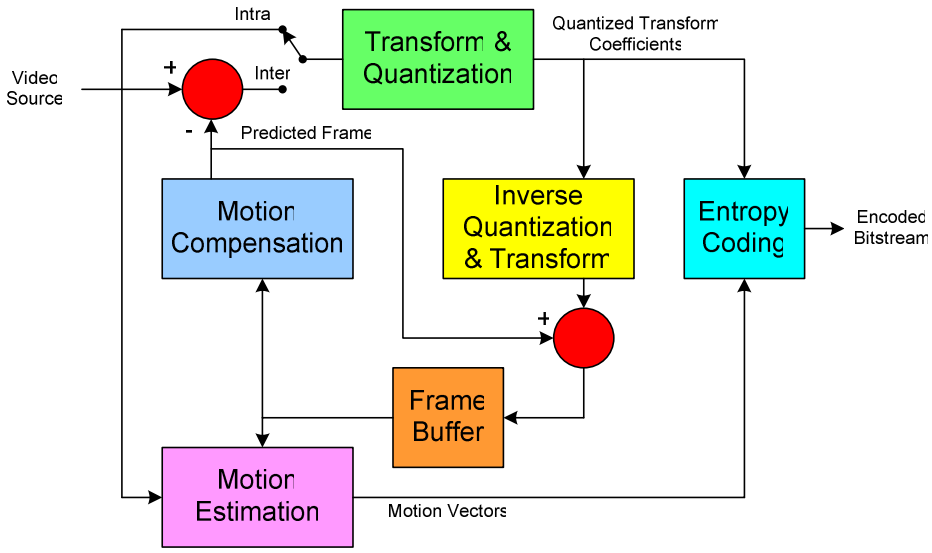


Figure 256 — Block diagram of a H.264/AVC encoder.

10.22.5.1 Context- based Adaptive Variable Length Coding (CAVLC)

The CAVLC encodes the quantized transform coefficient of the residual images. The transform coefficients are first scanned in a zigzag fashion as shown in Figure 257.

© ISO/IEC 2009 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

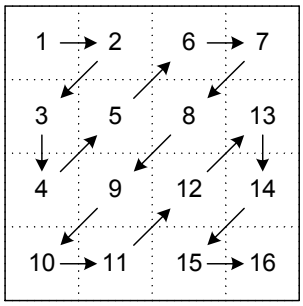


Figure 257 — Zigzag scanning.

The zigzag scanning is done to produce long runs on zeros. The CAVLC is designed observing the following characteristics (Table 72) of these reordered 4×4 blocks of transform coefficients:

Table 72 — Characteristics of the reordered transform coefficients.

Characteristics	Advantage taken
The quantized transform coefficient blocks are sparse that produce runs of zeros between nonzero coefficients.	Runs of zeros are compactly represented using run level coding.
The nonzero coefficients are often sequence of ± 1s.	A number of ± 1 coefficients (trailing ones') are compactly represented.
The number of nonzero coefficients in the neighboring blocks is correlated.	VLC table is chosen in which shorter codeward is used for the possible number of nonzero coefficient of the current block.
The absolute value of the nonzero coefficients are larger at the beginning of the reordered stream.	VLC table is chosen depending on recently coded nonzero coefficient to produce shorter codeward for the current coefficient.

CAVLC encoder encodes the elements listed in Table 73 of a block of transform coefficients in order:

Table 73 — Elements of a CAVLC bit-stream.

Elements	Description
coeff_token	Encodes the number of nonzero coefficients (TotalCoeff) and trailing ones (TrailingOnes) – one per block
sign_trail	Sign of the trailing ± 1s – one per trailing one
levels	Encodes the nonzero coefficients – one per nonzero coefficient excluding trailing ones
total_zeros	Encodes the total number of zeros occurring after the first nonzero coefficient – one per block
run_before	Encodes number of zeros preceding each nonzero coefficient in reverse zigzag order

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.22.5.2 coeff_token

The first VLC in the CAVLC stream is coeff_token. It encodes both the total number of nonzero coefficients and the number of trailing ones. Total coefficient can have a value between 0 to 16 and the number of trailing ones can have a value between 0 to 3. If the block has more than 3 trailing ones the rest are coded as normal coefficient. There are three VLC tables and one FLC table to choose from and the choice depends on the number of total coefficients in the left hand block (N_L) and upper block (N_U) of the current block. A parameter N is calculated for this purpose following the pseudo code shown in Figure 258.

if upper block and left hand block is available

$$N = \left(\frac{N_L + N_U}{2} \right)$$

else if upper block is available

$$N = N_U$$

else if left hand block is available

$$N = N_L$$

else

$$N = 0$$

Figure 258 — Pseudo code for calculating N from N_L and N_U.

The choice of the table for coeff_token is made as follows:

Table 74 — Choice of coeff_token table.

Value of N	Table
N < 2	VLC 1
2 ≤ N < 4	VLC 2
4 ≤ N < 8	VLC 3
N ≥ 8	FLC

The VLC tables are biased from small to large number of total nonzero coefficients in a block so that coeff_token can be coded with smallest possible codeward. The FLC is a 6 bit code. The least significant 2 bit represents the value of trailing ones and the rest represents the total number of nonzero coefficient in the current block.

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.22.5.3 sign_trail

The sign of each trailing one included in the coeff_token is coded with a single bit. Bit 0 is assigned for +1 and bit 1 is assigned for -1. The sign_trail is coded in reverse of the zigzag ordered stream so that highest frequency trailing one comes first.

10.22.5.4 levels

The nonzero coefficients (excluding trailing ones) are encoded in reverse order of a zigzag ordered block. The level (sign and magnitude) code comprises a prefix and a suffix. Initially the length of the suffix is set to 0 unless there are more than 10 nonzero coefficients and less than 3 trailing ones. In that case the length of the suffix is initialized to 1. If the magnitude of the coefficient is larger than a predefined threshold, the length of the suffix is incremented by 1 except for the first coefficient, in which case the length of the suffix is always incremented by 1 (if the suffix length was initialized to 0, otherwise not). This means the VLC for nonzero coefficients depends on previously coded levels to ensure shorter coderwards are assigned.

10.22.5.5 total_zeros

total_zeros encodes the total number of zeros preceding to the highest frequency nonzero coefficient in the zigzag order array.

10.22.5.6 run_before

run_before encodes the sum of zeros preceding each nonzero coefficient starting from the highest frequency nonzero coefficient (in reverse order). In two cases the run_before of a coefficient need not to be encoded. These two cases are: if there are no more zeros left and if this is the last (lowest frequency) coefficient of which run_before is left to encode.

10.22.6 Implementation

The block diagram of the proposed architecture is shown in Figure 259. It takes the quantized 4x4 block of transform coefficient, NU and NL as its input and outputs the CAVLC bit-stream.

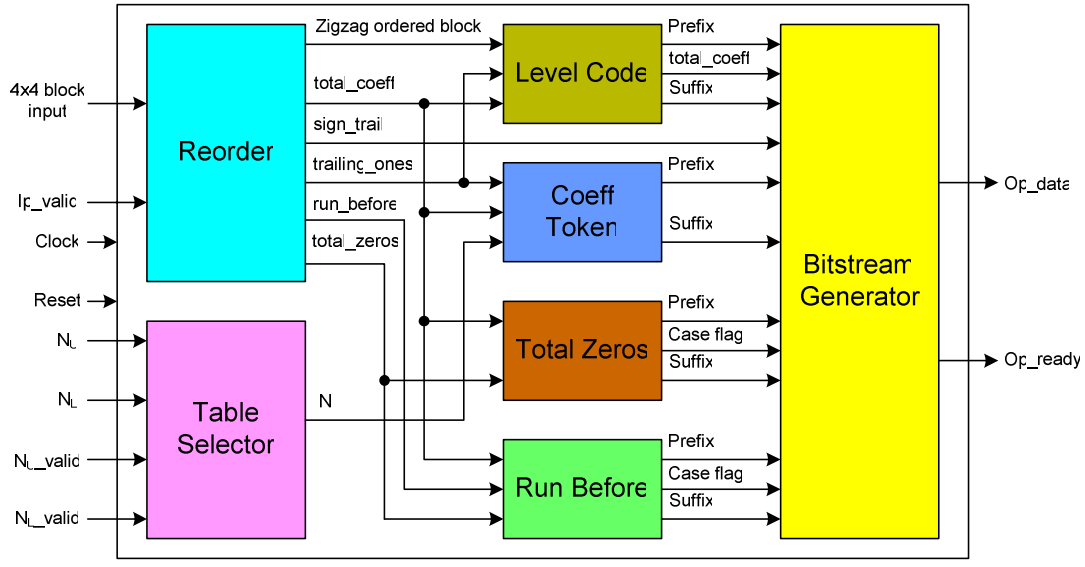


Figure 259 — The proposed CAVLC architecture.

The block diagram shows that the proposed architecture is composed of the following blocks: Reorder block, Table Selector block, Coeff Token block, Level Code block, Total Zeros block, Run Before block and Bitstream Generator block. The functions of each of the blocks are discussed in the following subclauses.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.22.6.1 Reorder block

The main function of this block is zigzag scanning. In addition, this block also performs the following: calculates the total number of nonzero coefficients, total_zeros, total number of trailing ones, run_before value of each nonzero coefficient and the sign_trail. The sign_trail is a 3 bit register, initialized to 0 (000). While zigzag scanning, the reorder block pushes bit 0 (for +1) or bit 1 (for -1) from MSB in reverse zigzag order. There is a mux to select the output bit of this sign_trail driven by the total number of trailing ones. If trailing_ones = 3, LSB is chosen as the output bit, so that when sign_trail is chosen by the bitstream generator block for output, the right shifted (at each clock pulse) 3 bit sign_trail register can output the correct VLC. When the input data is ready, the Ip_valid flag is set to 1, asking the core to process the current block of transform coefficients.

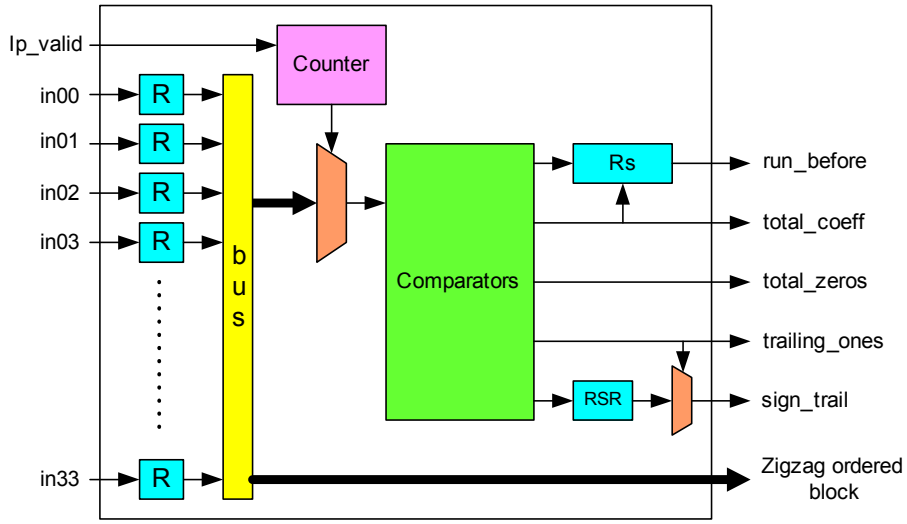


Figure 260 — The reorder block.

10.22.6.2 Table selector block

Table Selector is a combinational logic block. The function of this block is to calculate N to select the appropriate table for coeff_token. Its inputs are N_U, N_L, N_U_valid and N_L_valid and output is N.

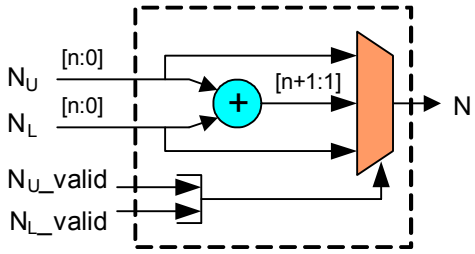


Figure 261 — The table selector block.

10.22.6.3 Coeff token block

In order to reduce the area of the proposed architecture the look up tables are split. This is done by observing the VLCs. For total_coeff > 2, each code in the VLC table contains a number of zeros (prefix) followed by a 4 bit value (suffix). A 4 bit value can represent 0-15. Since none of the VLCs contains a prefix greater than 12, same 4 bit table can be used to represent the prefix as well as the suffix. Table 75 shows an example of the split table VLC when total_coeff = 3.

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

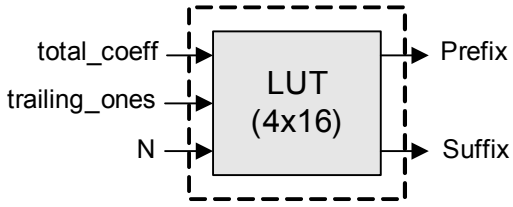


Figure 262 — The coeff token block.

Table 75 — Prefix and suffix values for coeff_token when total_coeff = 3.

total_coeff	VLC Table	*trailing_ones*	Prefix (4 bit)	Suffix (4 bit)
3	1	0	0101	0111
		1	0100	0110
		2	0011	0101
		3	0001	0011
	2	0	0011	0111
		1	0010	1010
		2	0010	1001
		3	0000	0101
	3	0	0010	1000
		1	0001	1100
		2	0001	1110
		3	0000	1100

There is a special case when total_coeff ≤ 2. In this case some of the VLCs are represented by less than 4 bits. When VLCs have less the 4 bits, the prefix signals the number of bits is needed to be extracted from the MSB of suffix to represent the VLC. This is shown in Table 76. The table shows the case when total_coeff = 0 and trailing_ones = 0. In this case, when VLC table 2 is chosen, the prefix indicates the first two MSB bits of suffix are needed to be extracted. So the VLC is 11.

Table 76 — Prefix and suffix values for coeff_token when total_coeff = 0.

total_coeff	VLC Table	*trailing_ones*	Prefix (4 bit)	Suffix (4 bit)
0	1	0	0001	1000
	2	0	0010	1100
	3	0	0100	1111

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2009 – All rights reserved
Not for Resale

10.22.6.4 Level code block

The function of the level code block is to encode the nonzero coefficients excluding the trailing ones. The blocks inputs are the zigzag ordered block, total_coeff and the trailing_ones. The format of the level code is shown in the Figure 264.

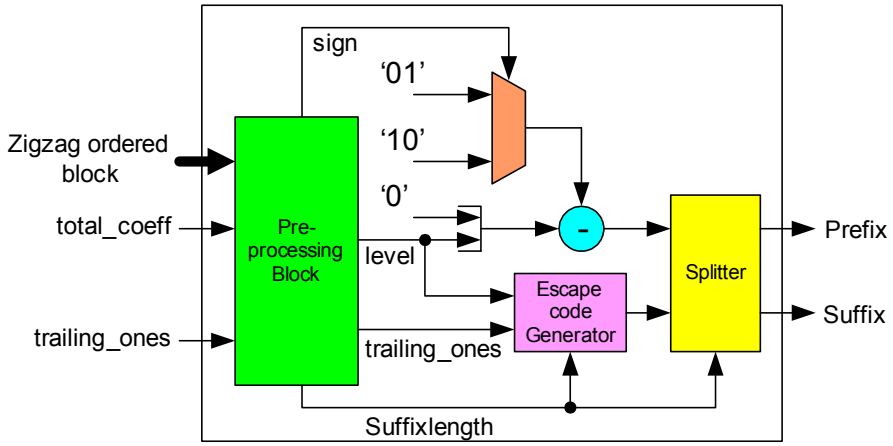


Figure 263 — The level code block.

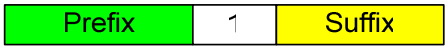


Figure 264 — The format of the level VLC.

The prefix is an array of zeros and the suffix is an array of bits; the size of which may vary from 0 to 12. The block outputs the prefix and suffix. As prefix it outputs the number of zeros that the code has for a level and as suffix it outputs the array of bits preceding the in between bit 1. The block applies 3 steps to generate level code shown in Figure 265.

Step 1: Left shift 1 bit to the magnitude of the level

Level code = (Level magnitude << 1)

Step 2: If the sign of the level is positive *Sub* = 2, otherwise *Sub* = 1.

Level code = Level code - Sub

Step 3: Right shift *Suffixlength* amount bits to get the final *Level code*.

Level code = Level code >> Suffixlength

Figure 265 — Pseudo code to generate level code.

© ISO/IEC 2009 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Suffixlength are set inside the block for each level. The escape code levels are handled in a special way. Extra combinational logics are applied to generate the level code (prefix and suffix) for these levels.

10.22.6.5 Total zeros block

Split table VLC (VLC is split to prefix and suffix) is also applied to generate total_zeros VLC. As in coeff_token, the VLC of total_zeros is composed of a number of zeros (prefix) followed by a 3 bit value (suffix). Some of the VLCs contain less than 3 bits. These VLCs are treated in a special way similar to the special case of coeff_token. Table 77 shows the VLC table when total_coeff = 12. The case flag indicates whether or not the code has to be treated in a special way. When case flag is 0, the prefix indicates how many 0s precede the suffix. For an example, when total_zeros = 1, the prefix is 1 (001) and suffix is 001. So, the VLC is 0001. When total_zeros = 2, case flag is 1. So it is a special case. In this case, the prefix indicates the number of bits is needed to be extracted from the MSB of suffix to represent the VLC. Since prefix is 2 (010) and suffix is 010, the VLC is 01 for this case. Since the prefix is always less than 8, same 3 bit table can be used to represent the prefix and the suffix. This helps in reducing the area in the proposed architecture.

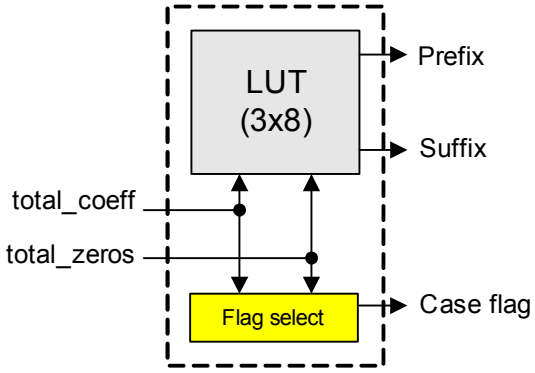


Figure 266 — The total zero block.

Table 77 — Prefix and suffix values for total_zeros when total_coeff = 12.

total_coeff	*total_zeros*	Prefix (3 bit)	Suffix (3 bit)	Case flag
12	0	001	000	0
	1	001	001	0
	2	010	010	1
	3	001	100	1
	4	000	001	0

10.22.6.6 Run before block

Run before block encodes number of zeros preceding each nonzero coefficient in reverse zigzag order. The blocks inputs are the run_before values of each nonzero coefficient in reverse zigzag order from the reorder block, total_coeff and total_zeros. Split table technique is also applied in this block. Same 3 bit look up table is used for the prefix and suffix in a similar way to the total zero block.

Table 78 shows the VLC table when run_before = 1. Case flag = 1 indicates special case as before that is handled in a way similar to the previous section.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

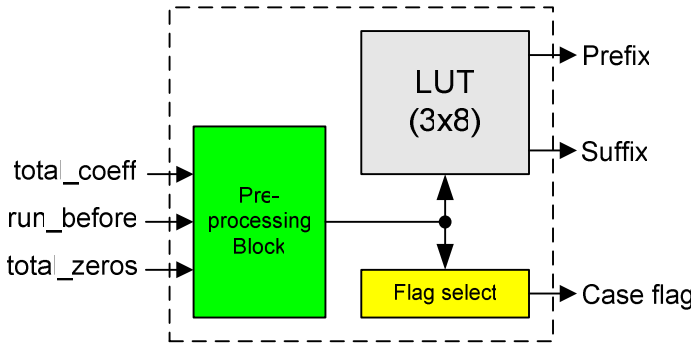


Figure 267 — The run before block.

Table 78 — Prefix and suffix values for run_before when run_before = 1.

Run_before	Zeros left	Prefix (3 bit)	Suffix (3 bit)	Case flag
1	1	001	000	1
	2	010	010	1
	3	010	100	1
	4	010	100	1
	5	010	100	1
	6	000	000	0
	>6	000	110	0

There is only one exception. 3 bit prefix can represent only up to 7. But when run_before = 14, there are 8 zeros preceding suffix. This exception is tackled by setting case flag = 1 and prefix = 0 (000).

10.22.6.7 Bitstream generator block

Bitstream generator block takes the outputs of all blocks and generates the CAVLC code. It has a 1 bit serial data output port (Op_data). The structure of the code is shown in the Figure 269.

© ISO/IEC 2009 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

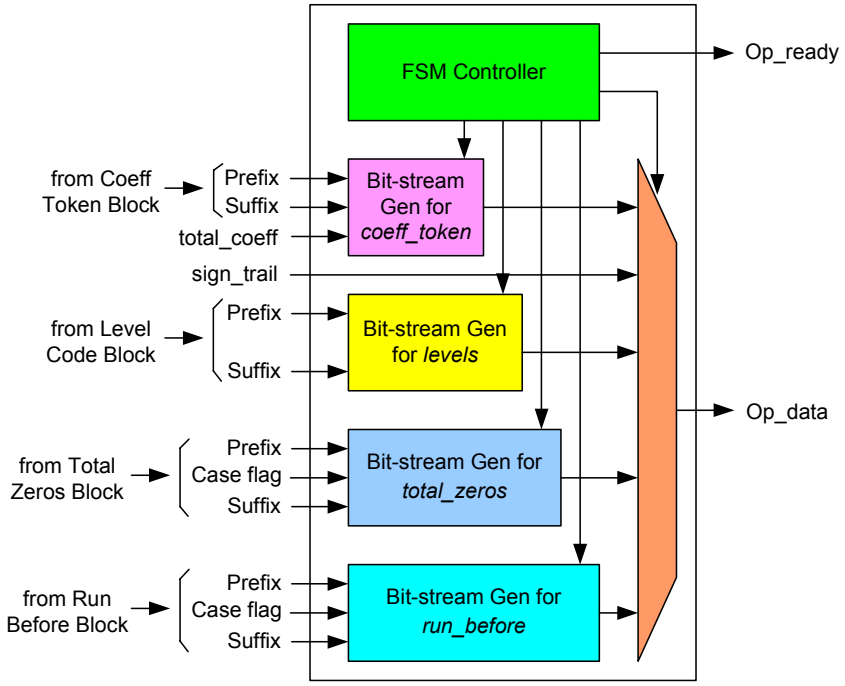


Figure 268 — The bitstream generator block.

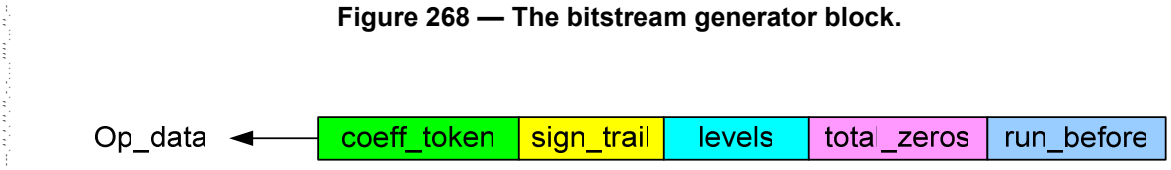


Figure 269 — The structure of the CAVLC code.

The first VLC that it outputs is coeff_token. Before outputting the code, the Op_ready flag is set to 1 indicating that valid CAVLC code is going to be available at Op_data port from the following clock pulse. If the value of total_coeff > 2, it outputs prefix number of 0s first and then the suffix serially from MSB to LSB, otherwise it outputs prefix number of bits from MSB to LSB of the suffix serially. Next it selects sign_trail for output that is already processed by the reorder block. After that it selects the outputs of the level code block. The level code is also composed of prefix and suffix. In addition, it has a bit “1” in between. So, while outputting the level code, it first outputs the prefix number of 0s, then a single bit “1” and at the end, the suffix serially from MSB to LSB. The VLC for total_zeros comes next. For this, the block selects the outputs of total zeros block as inputs. If the case flag is 0, the bitstream generator outputs the prefix number of zeros followed by the suffix serially from MSB to LSB. Otherwise, it outputs prefix number of bits from MSB to LSB of the suffix serially at each clock pulse. The final code in the CAVLC stream is the run_before. The block selects the outputs of the run before block as inputs. The run_before codes are generated in a similar way to the total_zeros code with only one exception - if the case flag is 1 and prefix is 0 (000), the block outputs eight 0s, followed by the suffix. The block sets the Op_ready flag to 0 indicating that the core has finished outputting the current CAVLC code and it is now ready to process the next block of transform coefficients.

10.22.6.8 Interfaces

Please refer to subclause 10.22.4.

10.22.6.9 Register File Access

N/A.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2009 – All rights reserved
Not for Resale

10.22.6.10 Timing Diagrams

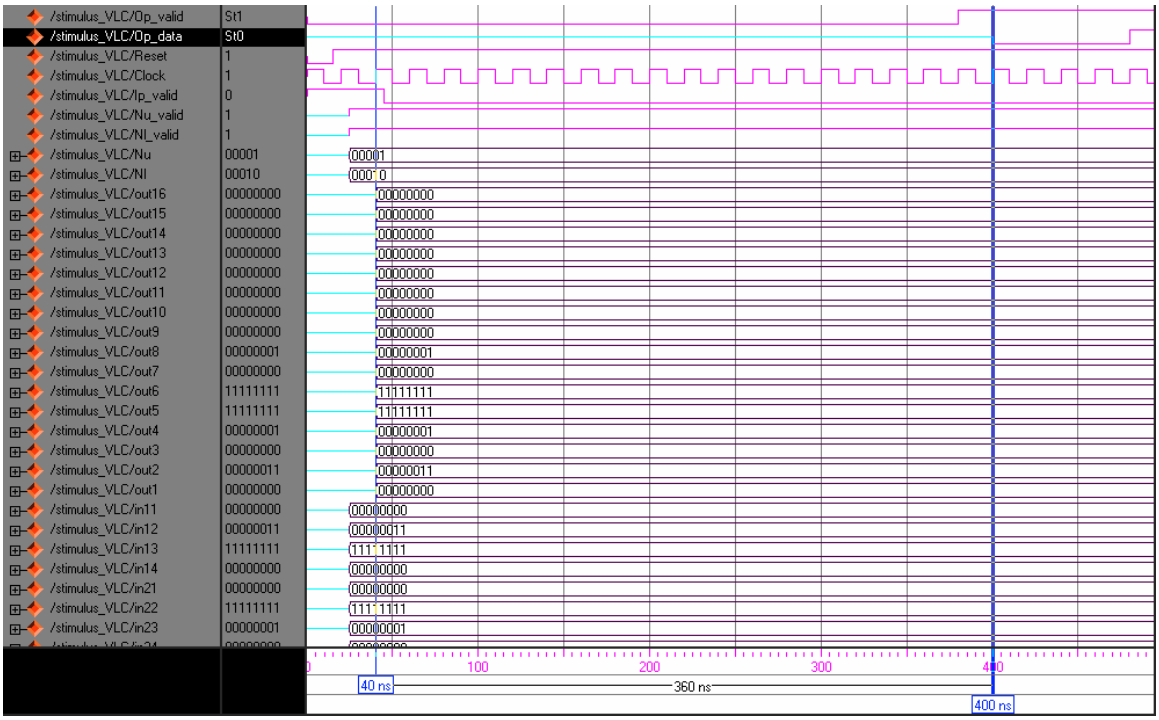


Figure 270 — The diagram shows the latency of the core. Clock period is 20ns for which the core latency is 360ns. This gives a latency of 18 clock cycles.

10.22.7 Results of Performance & Resource Estimation

The proposed CAVLC architecture is coded in Verilog HDL, simulated and synthesized by ModelSim 6.0 and Xilinx ISE development tools 6.2, respectively. The target device for synthesis was the Xilinx Virtex II FPGA (2v3000fg676-4). ModelSim simulator shows that the core setup time after being reset is 360ns with a 20ns period clock which is equivalent to 18 clock cycles. The prototype outputs the CAVLC code serially. Since the length of the code is variable, average time is taken from the simulation for calculating the system's performance. Table 79 shows the synthesis results.

Table 79 — Synthesis results of the proposed architecture.

	Resource usage	Utilization summary of a Xilinx 2v3000fg676-4 device
CLB Slices	866	6%
DFFs or Latches	576	2%
LUTs	1,533	5%
Bonded IOBs	143	29%

The gate count of the architecture is 15,363. Maximum core speed is 60.11 MHz that gives a critical path of 16.636 ns. In an average, around 43 clock cycles are needed for outputting the CAVLC for one 4x4 block of

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

quantized transform coefficients. So, in order to output the CAVLC of a single block the core takes 43 × 16.636 = 715.35 ns. There are (352 × 288) / (4 × 4) = 6,336 blocks of transform coefficients in a CIF frame. So, the core takes 715.35 × 6336 = ~ 4.6 ms to process a CIF frame. This gives Table 80 that shows the processing time of a CIF and QCIF image sequence by the proposed prototype. The table clarifies that the core is more than 7 times faster than the minimum speed threshold for real time CIF video sequence.

Table 80 — Processing time of video sequences.

Image	1 frame (ms)	30 frames (ms)
QCIF	1.15	34.50
CIF	4.60	138.00

10.22.8 API calls from reference software

N/A.

10.22.9 Conformance Testing

N/A.

10.22.9.1 Reference software type, version and input data set

N/A.

10.22.9.2 API vector conformance

N/A.

10.22.9.3 End to end conformance (conformance of encoded bitstreams or decoded pictures)

N/A.

10.22.10 Limitations

Since the architecture can process the CIF frame sequence in real-time without pipelining, the pipelining has not been introduced in the proposed architecture. It also helped in reducing the size of the architecture. However, for HDTV applications the pipelining will surely play a key role for speeding up the architecture to satisfy the real-time threshold. We are currently working on this area.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.23 HARDWARE IMPLEMENTATION OF FULL SEARCH MPEG-4 PART 10 AVC MOTION ESTIMATION

10.23.1 Abstract

The motion estimation is one of the most computational intensive tasks of the H.264/MPEG-4 AVC [14] video coding standard. It could consume about 60% of the total coding time using one reference frame and 80% of the time with 5 references frames [54]. The developed module performs the motion estimation using the full-search block-matching algorithm with early termination for H.264/MPEG-4 AVC in a Xilinx VIRTEX™-II FPGA [1].

10.23.2 Module specification

10.23.2.1 MPEG 4 part: **10**
10.23.2.2 Profile : **All**
10.23.2.3 Level addressed: **All**
10.23.2.4 Module Name: **Motion**
10.23.2.5 Module latency: **361**
10.23.2.6 Module data throughput: **1 Million MBs (16x16)/sec**
10.23.2.7 Max clock frequency: **50MHz**
10.23.2.8 Resource usage:
10.23.2.8.1 LUTs **3590**
10.23.2.8.2 Multipliers **0**
10.23.2.8.3 BRAMs **4**
10.23.2.9 Revision: **1.0**
10.23.2.10 Authors: **M. Santos, A. Navarro**
10.23.2.11 Creation Date: **19/10/2006**
10.23.2.12 Modification Date: **19/10/2006**

10.23.3 Introduction

In H.264/MPEG-4 AVC, motion estimation has new features like variable block size, multiple reference images, motion predictive vectors, quarter-pixel resolution, etc. These new features require a relative greater computational power and complexity than MPEG-2. Practical results demonstrate that H.264/AVC motion estimation consumes about 60% of the coding time process with one reference image and about 80% for multiple reference frames [54].

The module presented in this document, performs H.264/MPEG-4 AVC motion estimation that could be used to accelerate the coding process of H.264 video coding standard. This module uses the full-search block-matching algorithm with early termination. Although it is slower than [55], it guarantees that the best block match is found and therefore obtaining the greatest Peak to Signal Noise Ratio (PSNR) of the reconstructed video.

10.23.3.1 Functional Description

The module starts when the START input is set to high. Then it collects all the pixels for the current image, a 16×16 block, and the pixels for the reference image, a 48×48 block, into two separate memories. One memory that is outside of the FPGA, DDR-SDRAM, is used to store the 48×48 block (reference image). The current 16x16 pixel block is stored on internal memory, BRAM (on-chip memory).

Once all the pixels are loaded, then the module starts to calculate the Sum of Absolute Differences (SAD) in blocks of 16×16 pixels. As the module implements the full-search algorithm, all pixels of the current image block are used in the calculations unless an early termination occurs, thereby, as soon as it is detected that the current SAD is above the value of any previous SAD then the reference block is not a good candidate. It skips to the next 16×16 block increasing the speed of the process.

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

When the module finds the best match of the reference frame with the current frame it starts to calculate the motion vectors.

After the motion vectors are calculated the module signals READY high indicating that the process is done.

10.23.3.2 I/O Diagram

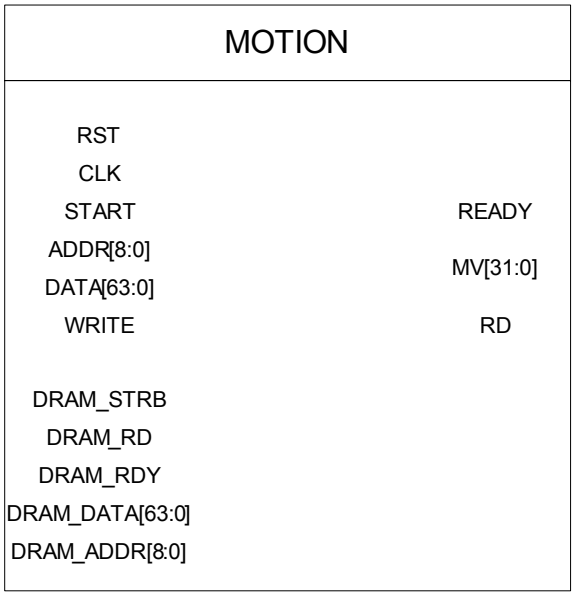


Figure 271 — FPGA I/O diagram.

10.23.3.3 I/O Ports Description

- RST – at level high, it resets all the flip-flops and state machines on the module.
- CLK – system clock.
- START – is asserted high to start the motion estimation process.
- ADDR[8:0] – address lines for reading/writing pixels and motion vectors.
- DATA[63:0] – data lines for loading pixels.
- WRITE – is asserted high to load pixel data. It must be low for reading motion vectors.
- RD – when high it reads the motion vectors. Only valid when READY is high.
- MV[31:0] – motion vectors.
- READY – valid data pixels at PIXELS_OUT[8:0] when READY is high.
- DRAM_STRB – DDR-SDRAM controller strobe.
- DRAM_RD – DDR-SDRAM controller read.
- DRAM_RDY – DDR-SDRAM controller ready.
- DRAM_DATA[63:0] – DDR-SDRAM controller data in/out.
- DRAM_ADDR[8:0] – DDR-SDRAM controller address.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.23.3.4 I/O Ports Description

Port Name	Port Width	Direction	Description
RST	1	Input	Reset the module
CLK	1	Input	Clock to load the input stream in VLC
START	1	Input	Start motion estimation if START is high
ADDR	9	Input	Address for the pixel and motion vectors data
DATA	64	Input	Pixel data
WRITE	1	Input	Write signal
RD	8	Input	Read signal
MV	32	Output	Motion vectors
READY	1	Output	Valid data at MV port
DRAM_STRB	1	Output	DDR-SDRAM controller strobe
DRAM_RD	1	Output	DDR-SDRAM controller read
DRAM_RDY	1	Input	DDR-SDRAM controller ready
DRAM_DATA	64	Input/Output	DDR-SDRAM controller data
DRAM_ADDR	9	Output	DDR-SDRAM controller address

10.23.4 Algorithm

The motion estimation exploits the temporal redundancy of a video sequence and the full-search block-matching algorithm produces the best result [54].

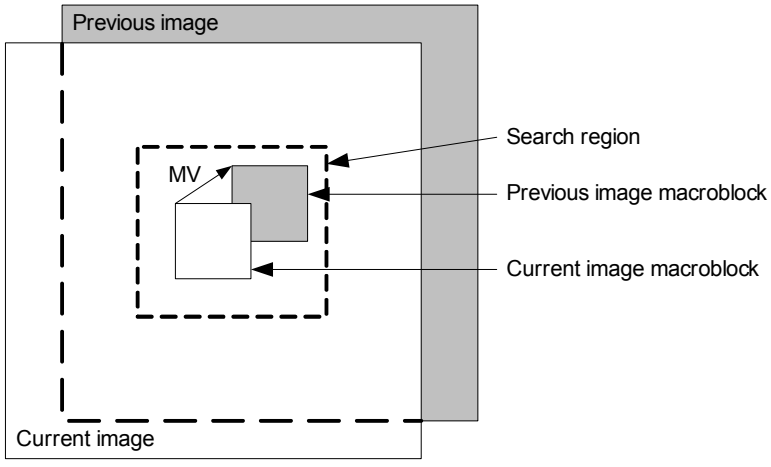
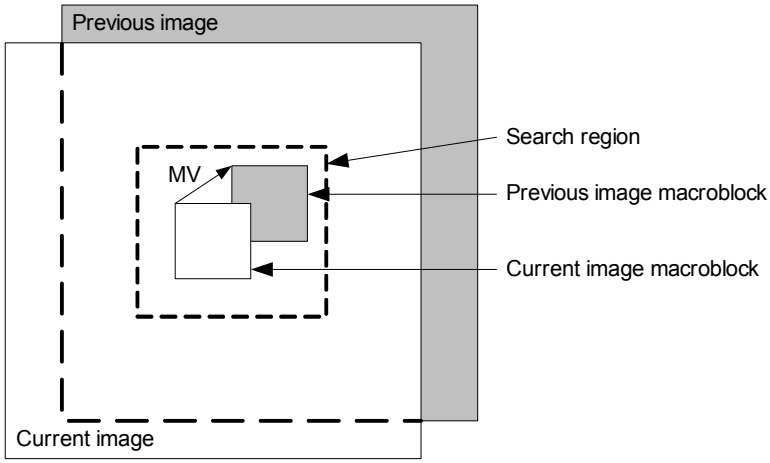


Figure 272 — Block matching.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale



As illustrated in

Figure 272, the motion estimation is done in two steps. First, the sum of absolute differences, $SAD_{(i,j)}$, is calculated for each displacement (motion vector). Then, it is calculated the minimum value, SAD_{MIN}, of all previously calculated SADs.

$$SAD_{(i,j)} = \sum_{k=0}^{N-1}\sum_{l=0}^{N-1}\left|C(k,l) - R(i+k, j+l)\right| \quad \textbf{(79)}$$

$$SAD_{MIN} = \min(SAD(i,j)) \quad \textbf{(80)}$$

where *C(k,l)* is the current block and *R(i+k,j+l)* is the reference block.

The motion estimation in H.264/MPEG-4 AVC was improved by using variable block size motion estimation unlike previous MPEG coding standards. The macroblocks in H.264 are segmented into two modes, macroblock mode, Figure 273-a), and 8×8 mode, Figure 273-b).

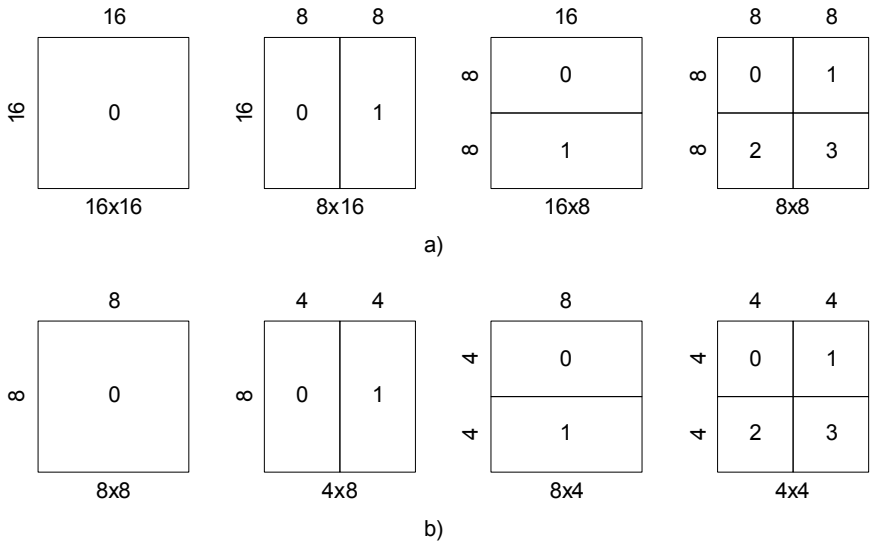


Figure 273 — a) Macroblock mode b) 8x8 mode.

This segmentation results in a total of 41 sub-blocks, i.e., each macroblock has a total of 41 motion vectors. And the present module, calculates those 41 motion vectors using the full-search block-matching algorithm with early termination. Although this module is slower than [55] it guarantees that the best matching region is found yielding better PSNR values and compression. The use of early termination makes the search process

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

faster. If, during the search process, the current SAD value is greater than any previous one, then the module skips to the next search and thus reducing the number of cycles to perform the full search algorithm.

The search window is a 48×48 pixel block that is three times the size of 16×16 macroblock. And each 16×16 macroblock is divided in 4×4 blocks, Figure 274.

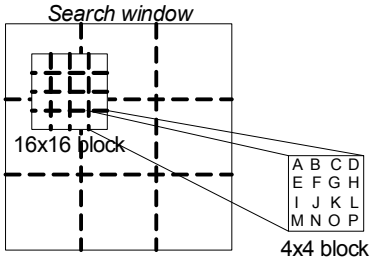


Figure 274 — Search window.

In each 4×4 block each pixel is labelled in alphabetically order. And the search window has pixels of the reference image where it must be found a 16×16 block that best matches with the current 16×16 block.

The developed algorithm is shown in Figure 275.

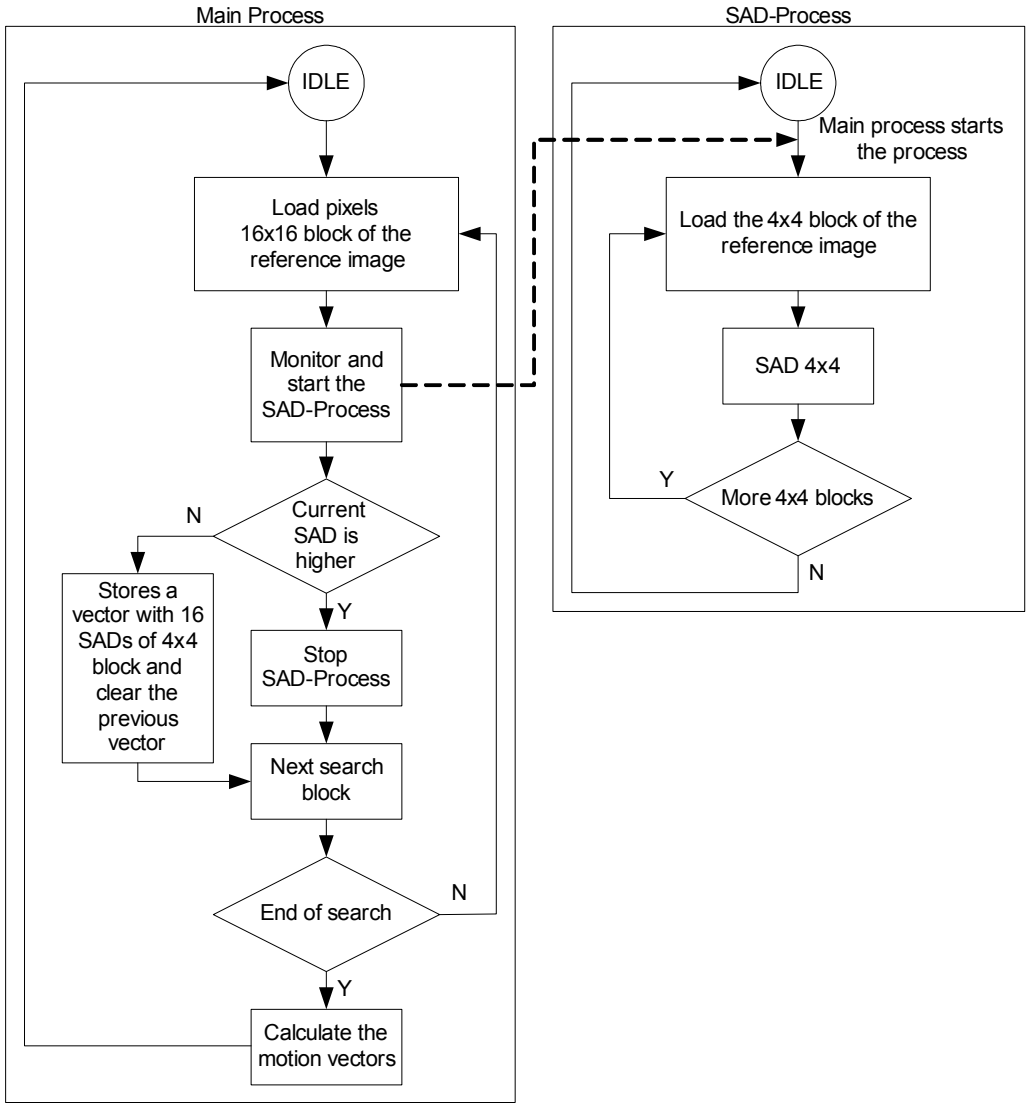


Figure 275 — Motion Estimation algorithm.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2009 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

There are two concurrent processes running in parallel, the main and the SAD. The main process starts by loading the 16×16 pixels of the search window and the SAD-process starts running. The SAD-process stores one of the 16 blocks of 4×4 pixels of the reference 16×16 block and calculates the SAD of a 4×4 block. This calculation is done in a single clock cycle. Mean while, the main process is storing the remaining 4×4 SADs that the SAD-process process. It must be noted that the 16×16 block of the current image has been previously loaded.

Internally, the SAD-process will successively calculate the sum of all 16 4×4 SADs of a 16×16 block. This value is constantly passed to the main process to be used for early termination. Each time this value is higher than the previous one, the SAD-process is stopped and the main process loads the next pixels of the search block.

The main process has 2 SAD vectors with 16 positions each. The first has 16 current 4×4 SADs and the other 16 previous calculated 4×4 SADs. If the current SAD is lower than the previous, then the actual SAD vector takes the place of the previous SAD vector and vice-versa. This is done to keep track of the valid 16 SAD values.

After all 16 4×4 SADs of the reference 16×16 block are processed, the main process verifies whether there are more 16×16 blocks in the search window. If yes, then the main process loads the next 16×16 block. If not, the main process calculates the motion vectors.

As the final SAD vector of the main process has the lowest sixteen 4×4 SADs, i.e, the SADs values corresponding to the best match, the main process calculates the remaining SADs: 4×8, 8×4, 8×8, 16×8, 8×16 and 16×16, Figure 276.

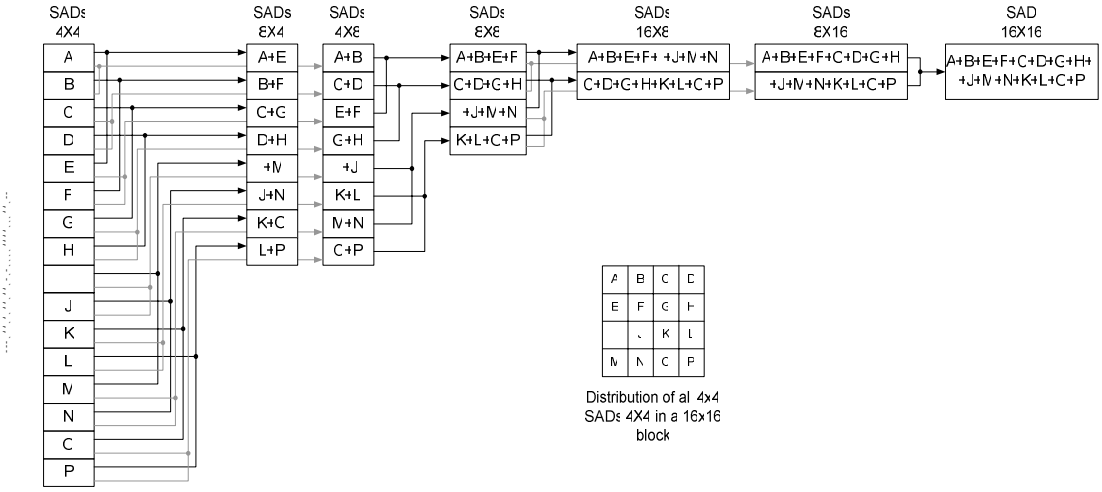


Figure 276 — SADs 4x4, 8x4, 8x8, 16x8, 8x16 and 16x16.

The operations of all these SADs are calculated in a single clock cycle.

10.23.5 Implementation

Figure 277 shows the block diagram of the motion estimation module.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

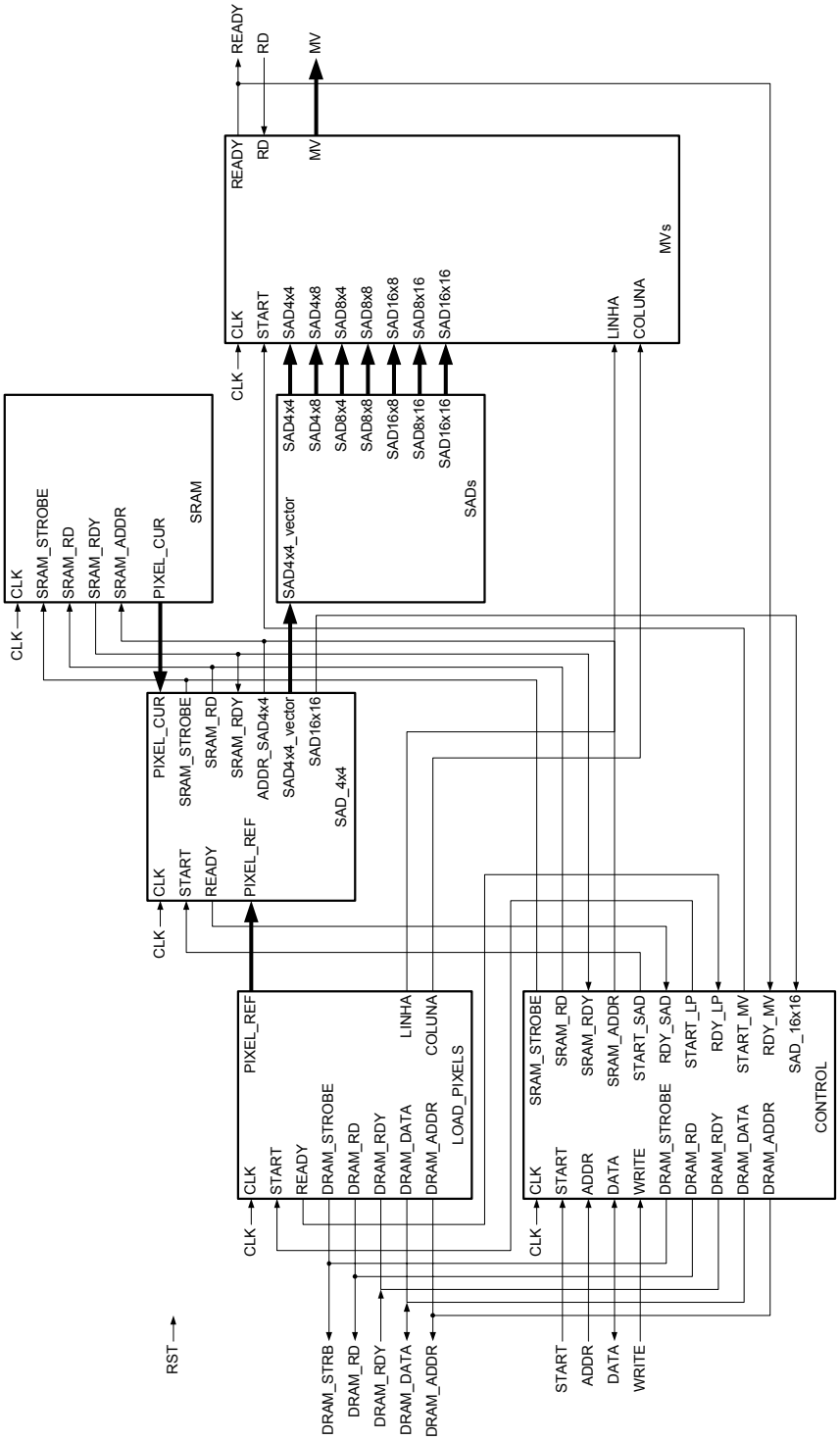


Figure 277 — Motion estimation block diagram.

The motion estimation module starts when START is set to high (START = 1) in the CONTROL sub-module. Then the pixel data for the current block and search window, reference block, should be loaded. The pixels of the search window are stored in the DDR-SDRAM of the WildCard-II device (WCII) and the pixels from the current block are loaded into the BRAMs of the VIRTEX-II FPGA. Those BRAMs are static random-access memories, SRAM (on-chip). After all pixels are loaded, then the motion estimation process of begins.

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

First, the sub-module CONTROL sets START_LP to high (STAR_LP = 1) for the sub-module LOAD_PIXELS start loading the pixels corresponding to the 16×16 block of the search window. This sub-module LOAD_PIXELS has all the logic for addressing (DRAM_ADDR) the DDR-SDRAM controller inside of the WCII. The loading of pixels is considered done when the LOAD_PIXEL module sets READY to high. At the same time, the sub-module CONTROL detects that the pin RDY_LP is high (RDY_LP = 1) and starts the calculation of the SAD values setting START_SAD to high. Now, the CONTROL sub-module will monitor the SAD_16x16 signal that indicates the actual best match value. This signal is used for the early termination.

When the best match is found, the CONTROL sub-module starts the MV sub-module that calculates the motion vectors, START_MV is then set to high (START_MV = 1). The MV sub-module will use the signals LINHA and COLUNA to calculate the motions vectors related to the motion of the 16×16 current block in relation to the search window.

When READY is valid (READY = 1) on the MV sub-module, the process of motion estimation is done. To read the resulting motion vectors, the RD pin should be set high (RDY = 1). Then, the 41 motion vectors could be read from the MV port of the module.

The sub-module SRAM is a module with four 16×32 static memories (BRAMs) where each memory stores the 16×16 pixels of the current block.

Those memories are organized in such a manner that the calculation of a 4×4 SAD is performed in one clock cycle, Figure 278.

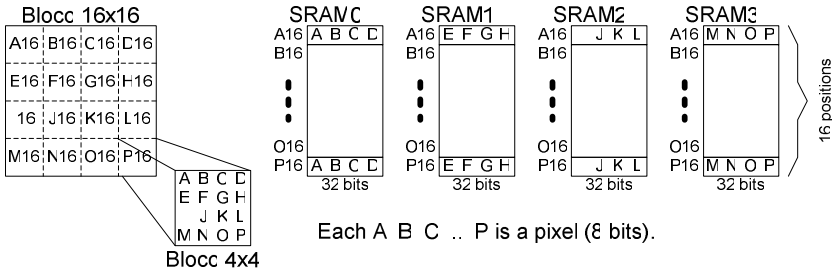


Figure 278 — Organization of the 4 BRAMs.

The LOAD_PIXELS sub-module loads the pixels of the DDR_SDRAM memory in such a way that, the data in the input of the PIXEL_REF signal of the SAD sub-module is in the same structure and order as in the PIXEL_CUR pins.

During the process of loading the pixels of the DDR-SDRAM, the sub-module LOAD_PIXELS stores in his pins COLUNA and LINHA the information of the localization of the actual test of the best match that will be used to calculate the motion vectors.

10.23.5.1 Interfaces

Figure 279 shows the interfaces of the implemented module.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

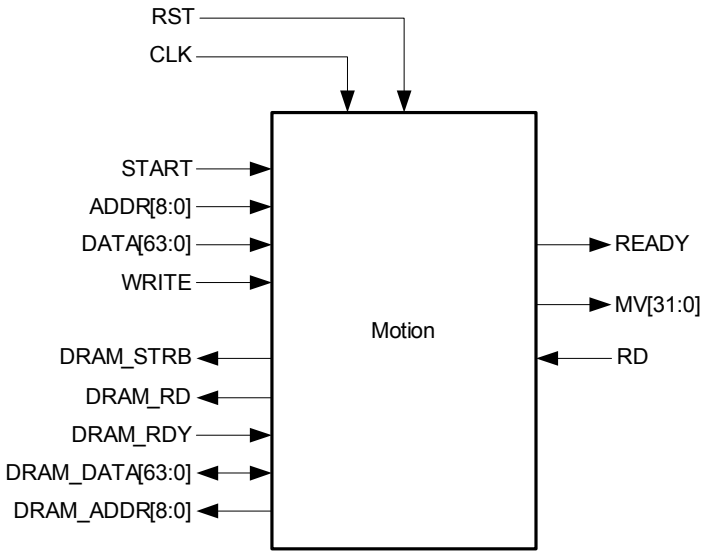


Figure 279 — Motion module interface.

- RST – at level high, it resets all the flip-flops and state machines on the module.
- CLK – system clock.
- START – is asserted high to start the motion estimation process. And it must high for the load of all pixel data.
- ADDR[8:0] – address lines for read/write of the pixels and motion vectors.
- DATA[63:0] – data lines for load of the pixels.
- WRITE – is asserted high to load the pixel data. It must be low for reading of the motion vectors.
- RD – when high it reads the motion vectors. Only valid when READY is high.
- MV[31:0] – motion vectors.
- READY – valid data pixels at PIXELS_OUT[8:0] when READY is high.
- DRAM_STRB – DDR-SDRAM controller strobe.
- DRAM_RD – DDR-SDRAM controller read.
- DRAM_RDY – DDR-SDRAM controller ready.
- DRAM_DATA[63:0] – DDR-SDRAM controller data in/out.
- DRAM_ADDR[8:0] – DDR-SDRAM controller address.

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.23.5.2 Timing Diagrams

The timing diagram of the implemented block is shown in Figure 280.

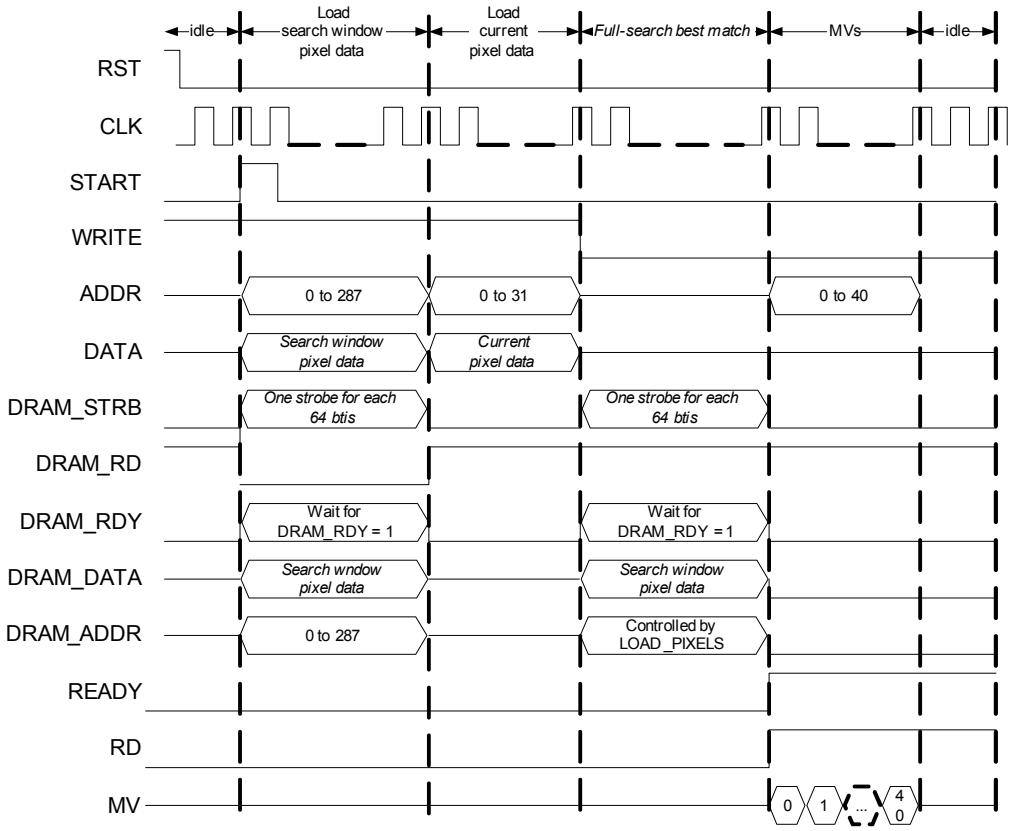


Figure 280 — Timing diagram.

It is necessary 2048 searches to perform the full-search matching in a 48×48 search window and a 16×16 block of the current image. If, for each search, it is used 8 clock cycles excluding the latency of the DDR-SDRAM controller then the total number of clocks cycles are 16364 clock cycles. But, because it's used the early termination algorithm, the number of cycles could be much reduced. Because, if the search is done from the centre of the block to the outside in spiral then the probability of the early termination could be very high reducing the number of clock cycles.

10.23.6 Results of Performance & Resource Estimation

The design tool used in this work was ISE 7.1i. The obtained report from the synthesis tool, XST, is presented below:

```
Device utilization summary:
---------------------------
Selected Device:        2v3000fg676-4
Number of LUTs:         3590   out of    28672     5%
Number of BRAMs:        4      out of    96        3%
Number of Multipliers:     0      out of    96     0%

Timing Summary:
---------------
Speed Grade: -4
Minimum period: 20ns (Maximum Frequency: 50MHz)
```


© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

10.23.7 API calls from reference software

```
DWORD WC_SAD(WC_Struct wildcard, int *r, int *o, int *sad)
{
  DWORD WC_rc;

  DWORD pr[4], po[4];

  pr[0] = (r[0] << 24) | (r[1] << 16) | (r[2] << 8) | r[3];
  pr[1] = (r[4] << 24) | (r[5] << 16) | (r[6] << 8) | r[7];
  pr[2] = (r[8] << 24) | (r[9] << 16) | (r[10] << 8) | r[11];
  pr[3] = (r[12] << 24) | (r[13] << 16) | (r[14] << 8) | r[15];

  po[0] = (o[0] << 24) | (o[1] << 16) | (o[2] << 8) | o[3];
  po[1] = (o[4] << 24) | (o[5] << 16) | (o[6] << 8) | o[7];
  po[2] = (o[8] << 24) | (o[9] << 16) | (o[10] << 8) | o[11];
  po[3] = (o[12] << 24) | (o[13] << 16) | (o[14] << 8) | o[15];

  WC_rc = WC_PeRegWrite(wildcard.DeviceNum, 0x10, 4, &pr);
  WC_rc = WC_PeRegWrite(wildcard.DeviceNum, 0x20, 4, &po);

  WC_rc = WC_PeRegRead(wildcard.DeviceNum, 0x30, 1, &sad);

  return WC_rc;
}

DWORD WC_SADl(WC_Struct wildcard, int *r, int *o, int *sad)
{
  DWORD WC_rc;

  DWORD pr[4], po[4];

  pr[0] = (r[0] << 24) | (r[1] << 16) | (r[2] << 8) | r[3];
  pr[1] = 0;
  pr[2] = 0;
  pr[3] = 0;

  po[0] = (o[0] << 24) | (o[1] << 16) | (o[2] << 8) | o[3];
  po[1] = 0;
  po[2] = 0;
  po[3] = 0;

  WC_rc = WC_PeRegWrite(wildcard.DeviceNum, 0x10, 4, &pr);
  WC_rc = WC_PeRegWrite(wildcard.DeviceNum, 0x20, 4, &po);

  WC_rc = WC_PeRegRead(wildcard.DeviceNum, 0x30, 1, sad);

  return WC_rc;
}


DWORD WC_Motion(WC_Struct wildcard, int *r, int *o, int *mv)
{
  DWORD WC_rc;

  DWORD pr[4], po[4];

  DWORD start, ready;

  pr[0] = (r[0] << 24) | (r[1] << 16) | (r[2] << 8) | r[3];
  pr[1] = (r[4] << 24) | (r[5] << 16) | (r[6] << 8) | r[7];
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
  pr[2] = (r[8] << 24) | (r[9] << 16) | (r[10] << 8) | r[11];
  pr[3] = (r[12] << 24) | (r[13] << 16) | (r[14] << 8) | r[15];

  po[0] = (o[0] << 24) | (o[1] << 16) | (o[2] << 8) | o[3];
  po[1] = (o[4] << 24) | (o[5] << 16) | (o[6] << 8) | o[7];
  po[2] = (o[8] << 24) | (o[9] << 16) | (o[10] << 8) | o[11];
  po[3] = (o[12] << 24) | (o[13] << 16) | (o[14] << 8) | o[15];

  WC_rc = WC_PeRegWrite(wildcard.DeviceNum, 0x10, 8, &pr);
  WC_rc = WC_PeRegWrite(wildcard.DeviceNum, 0x20, 8, &po);

  start = 1;
  WC_rc = WC_PeRegWrite(wildcard.DeviceNum, 0x10, 1, &start);

  ready = 0;
  while (ready == 0)
  {
    WC_rc = WC_PeRegRead(wildcard.DeviceNum, 0x10, 1, &ready);
  }

  WC_rc = WC_PeRegRead(wildcard.DeviceNum, 0x0, 16, &mv);

  return WC_rc;
}
```

10.23.8 Conformance Testing

10.23.8.1 Reference software type, version and input data set

The reference software used was JM 10.2 and the test sequence was foreman_part_qcif.yuv.

10.23.8.2 API vector conformance

The test vectors used are H.264 QCIF test sequences.

10.23.8.3 End to end conformance (conformance of encoded bitstreams or decoded pictures)

Command line: lencod

Output H.264 file: foreman_part_qcif.264

10.23.9 Limitations

This module only does integer motion compensation, but could be used for full search quarter-pixel resolution motion estimation if interpolation is done outside of the module.

The maximum operating frequency of the module is limited by the SAD sub-module because it is only composed of combinational logic. This sub-module has a relative long path that reduces the maximum clock rate. Anyway, the proposed approach results in a relative low resource usage of the FPGA with a reasonable maximum operation clock.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex A
(Informative)

Specification of directory structure for reference SW, HDL and documentation files of MPEG-4 Part 9 Reference HW Description

A.1 Introduction

This Annex specifies the standard directory structure of all SW and HW components that are documented in this Technical Report. It also specifies some additional documentation that could be included in any future Edition of this TR.

A.2 Directory Structure of TR SW Modules

```
Virtual_Socket/
|               |
|               |
|               Implementation_1/
|               |               |
|               |               version1/...
|               |               |
|               |               version2/...
|               |               |
|               |               version3/... under development
|               |               |
|               Implementation_2/
|               |               |
|               |               version1/...
|               |               |
|               |               version2/...
|               |               |
|               |               version3/... under development
|               |               |
|               Implementation_3/
|               |               |
|               |               version1/...
|
MPEG_Reference_Software/
|                       |
|                        microsoft-v2.4-030710-NTU/...
|
...
|
IP_CVS_MODULE/
              |
              README.TXT
              |
              doc/
              |
              hdl/
              |   |
              |   /rtl
              |   |
              |   /testbench
              |
              sw/
```

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
                |   |
                |   /stimulus_data
                |   |
                |   /modified
                |   |
                |   /additional
                |
               synthesis/
                |          |
                |          /constraints
                |          |
                |          /netlists
                |          |
                |          /logs/
                |          |     |
                |          |     /synplify
                |          |     |
                |          |     /ISE
                |          |
                |          /scripts
                |                |
                |                /synplify
                |                |
                |                /ISE
               simulation/
                          |
                          /data
                          |
                          /scripts
```

Notes:

- Directories and files highlighted in red in the above structure are compulsory – all others are optional.
- ***IP_CVS_MODULE*** corresponds to the CVS module name on the CVS server for a given IP core. This name *must* correspond to the "Module Name" given in the Technical Report at section 10.X.4.2.
- ***README.TXT*** gives an overview of the contents of the module and any guidelines the author deems necessary.
- ***doc*** could be used for any further documentation or diagrams associated with the module.
- ***hdl*** contains the RTL for the IP core and any testbench HDL for block-level and system level verification.
- ***sw*** contains any modified files from a specifically referenced version of the MPEG reference software code and any additional software files used (e.g. APIs) for conformance testing. This must correspond to a specific version of the reference software. Optional files include data stimulus or parameter/configuration files.
- ***synthesis*** contains synthesis scripts, constraints, netlists etc.
- ***simulation*** contains any stimulus, do files or scripts used for HDL testbenches associated with the module.

A.2.1.1 Additional Section in Contribution Document (e.g. SA-DCT)

CVS Module Name

The module name on the CVS server is found at subsection 10.X.2.4 in the MPEG-4 Part 9 Technical Report. For example, one module is ***SA_DCT***. This is to be found in the root directory on the CVS server. In this directory there is a file called ***README.txt,*** which details how to use this module. It describes

- Where to find the RTL (for the module and the framework).
- Where to find the synthesis scripts and how to execute them.
- How to launch the reference software and verify the conformance test.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Integration Framework Version

The ***SA_DCT*** module has been integrated with version 1 of the "virtual socket" Implementation 2 (Calgary Framework). This can be found in directory **`Virtual_Socket/Implementation_2/Version1`**.

Reference Software Version and Modifications

For example, the ***SA_DCT*** module has undergone conformance testing with the MPEG-4 Part 7 C++ reference software implementation (version microsoft-v2.4-030710-NTU). This can be found in the ***`MPEG_Reference_Software/microsoft-v2.4-030710-NTU/`*** directory. See ***`README.txt`***, which details how to launch the software.

The files modified to include hardware acceleration of this module are listed. The modified files are found in the ***`modified`*** directory. For instance, the original reference files adjusted to hardware accelerate the SA-DCT:

- **`microsoft-v2.4-030710-NTU/app/encoder/encoder.cpp`**
- **`microsoft-v2.4-030710-NTU/sys/dct.hpp`**
- **`microsoft-v2.4-030710-NTU/tools/sadct/sadct.cpp`**

Additional software files that have been created that specify the API to communicate with the hardware are found in the ***`additional`*** directory. For example, these are included in the reference software for the SA-DCT module as follows:

- **`microsoft-v2.4-030710-NTU/tools/sadct/hardware_sadct.c`**
- **`microsoft-v2.4-030710-NTU/tools/sadct/hardware_sadct.h`**
- **`microsoft-v2.4-030710-NTU/tools/sadct/wc_shared.c`**
- **`microsoft-v2.4-030710-NTU/tools/sadct/wc_shared.h`**
- **`microsoft-v2.4-030710-NTU/tools/sadct/wc_windows.h`**
- **`microsoft-v2.4-030710-NTU/tools/sadct/wcdefs.h`**

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex B
(Informative)

Tutorial on Part 9 CVS Client Installation & Operation

B.1.1 Introduction

Concurrent Versions System (CVS) is a version control system which allows to keep old versions of files (usually source code), keep a log of who, when, and why changes occurred, etc. For the ISG group of MPEG committee the CVS server is located at EPFL. The CVS server at EPFL (*lsipc25.epfl.ch*) has an operation and configuration manual (N6508). The manual describes how to configure the clients *PuTTY* [7] and *WinCVS* [8] in order to access the server. Yet this part of the annex describes the installation and configuration of the client software with some basic CVS commands in more details for those, who is dealing with a CVS server for the first time.

B.1.2 Tools for accessing the EPFL CVS repository

Access to the EPFL CVS repository is only possible via SSH protocol of version 2. DSA public key method is used in this process. For windows users two free utilities needed to be downloaded and installed in the system in order to access. These two utilities are – *PuTTY* and *WinCVS*. *PuTTY* is the background SSH engine for *WinCVS*. Apart from these two tools there are three other tools needed. These are *PuTTYgen* for key generation, *Peagent* for authentication, and *Plink* for linking *PuTTY* with *WinCVS*.

PuTTY, *PuTTYgen*, *Peagent* and *Plink* can be downloaded from http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html. After downloading copy the programs to a directory, e.g. “c:\PuTTY” and list it in the windows PATH environment variable so that the commands can be executed from any paths.

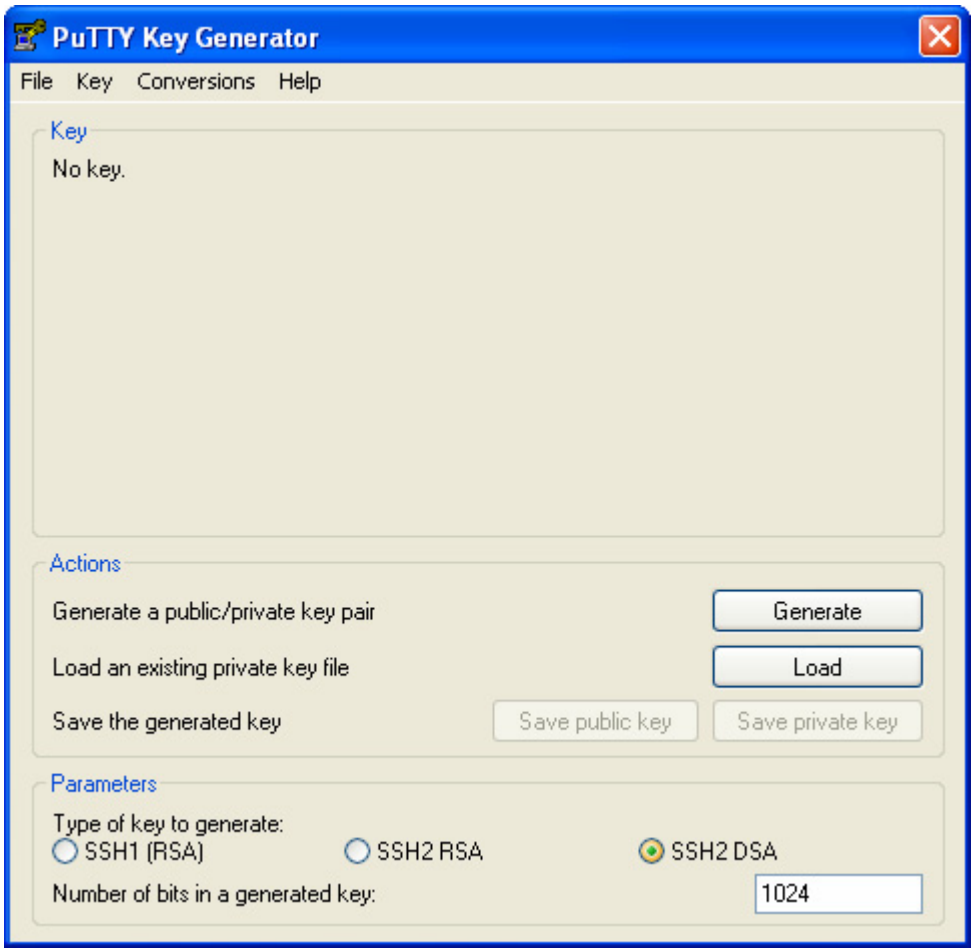
WinCVS can be downloaded from http://www.wincvs.org or http://cvsgui.sourceforge.net. In this tutorial the version of *WinCVS* that has been used is WinCVS 1.3.17.2 Beta 17 (Build 2).

B.1.2.1 Generating public and private keys for PuTTY

The following figure shows the *PuTTYgen* window. The default number of key bits is 1024. Select “SSH2 DSA” in the “Type of key to generate” options of the “Parameters”. Then click on the “Generate” button.

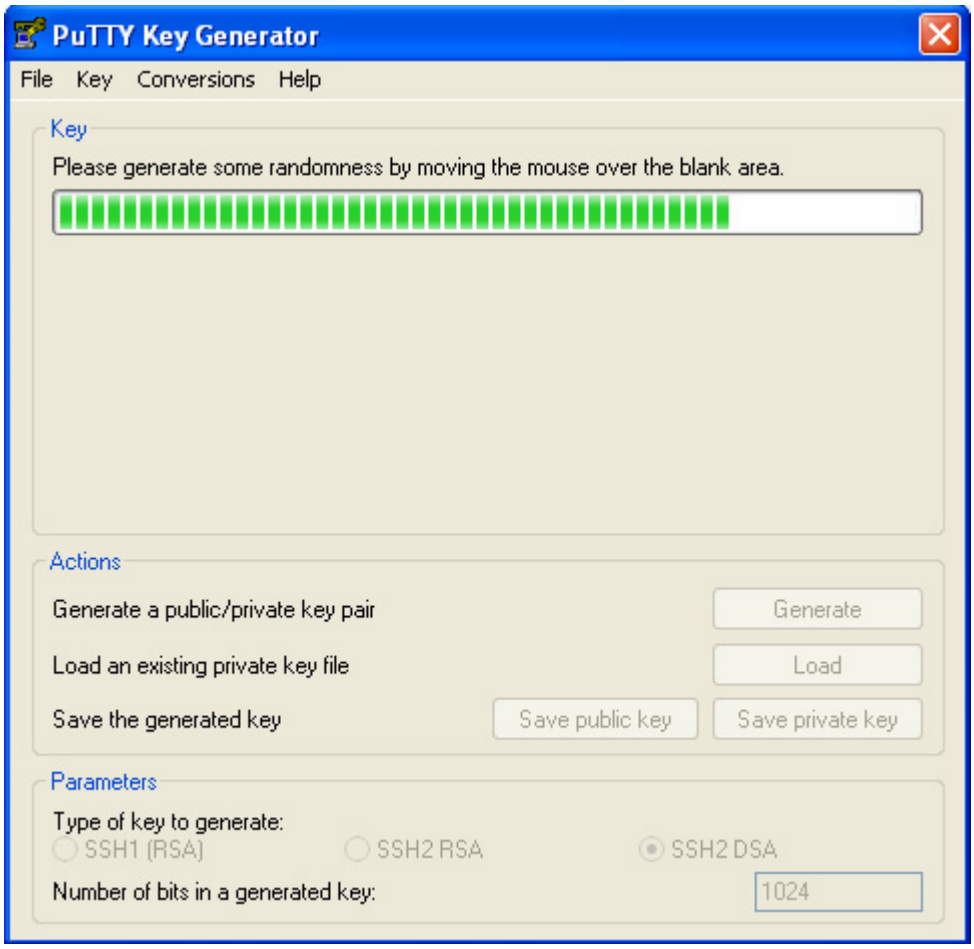
--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale



PuTTYgen will then ask the user to move mouse over the blank area in the key box as shown in the following figure so that it can generate public and private keys with randomness.

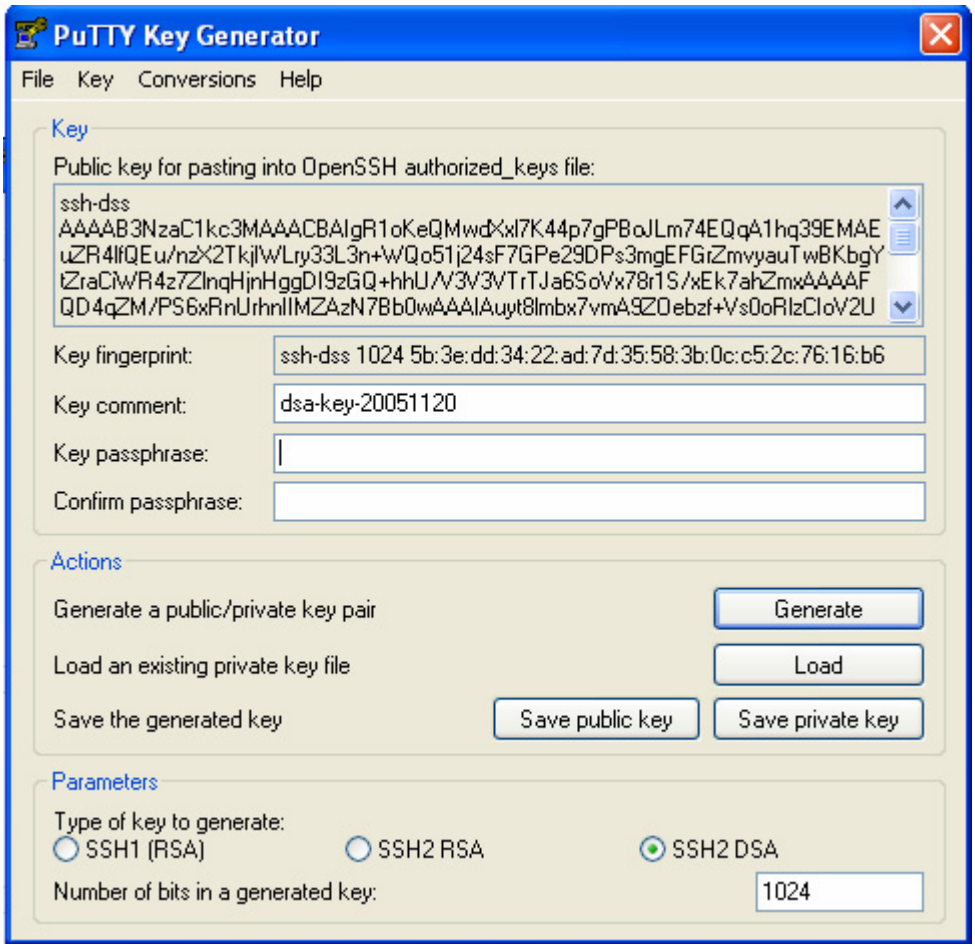
--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2009 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale



After generating the keys, the following screen will appear. Enter a passphrase for the private key and confirm it. Then click on the “Save private key” button. Enter a name for the private key and click “Save”. Now click on the “Save public key” button, enter a name and click “Save”. This ends the process of generation of public and private keys.

Send your public key and your preferred username to all the CVS manager email addresses and wait till your access to the CVS server is granted. Then proceed to the next section.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale



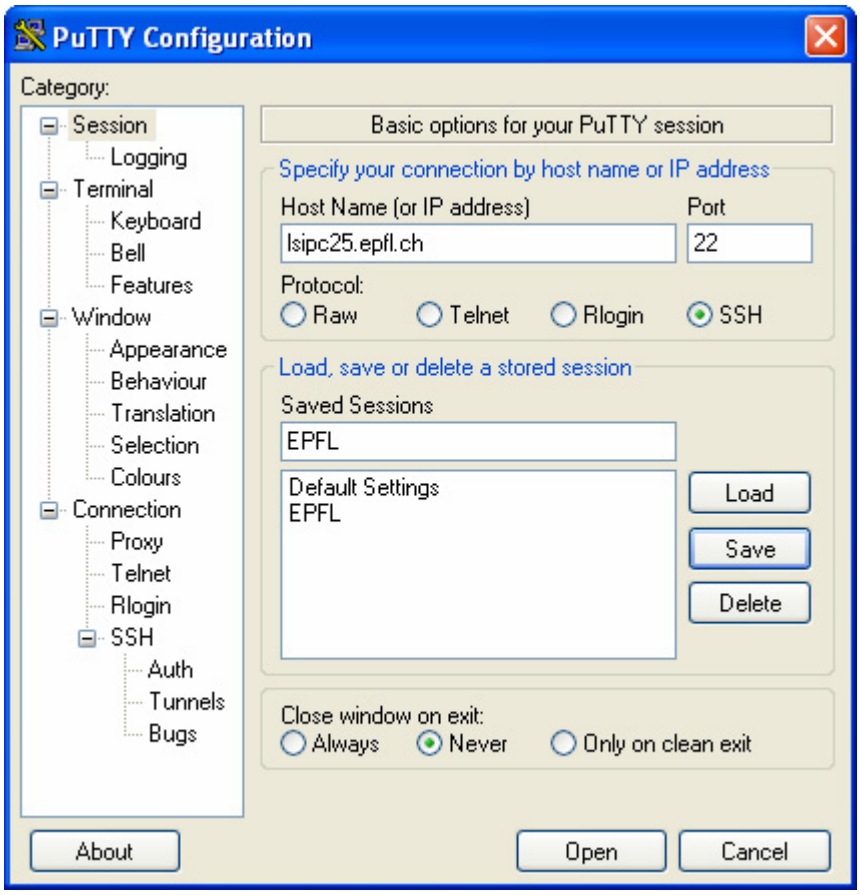
B.1.2.2 Configuring PuTTY

Execute *Peagent*. It is a TSR program and puts an icon on the Windows tray. Right click on the icon for a menu and then left click on “Add Key”. Now locate the private key, select it and click “Open”. It will ask for the passphrase. Enter the passphrase and click “OK”. The *Peagent* (*PuTTY* authentication agent) has now been set up.

Run *PuTTY*, the window in the following figure will appear. In the “Host Name” field write “lsipc25.epfl.ch”, Port 22. Select SSH as the Protocol. Then click on “Never” in the “Close window on exit” options. Now, give a name of the session, e.g. “EPFL” and click on “Save” button.

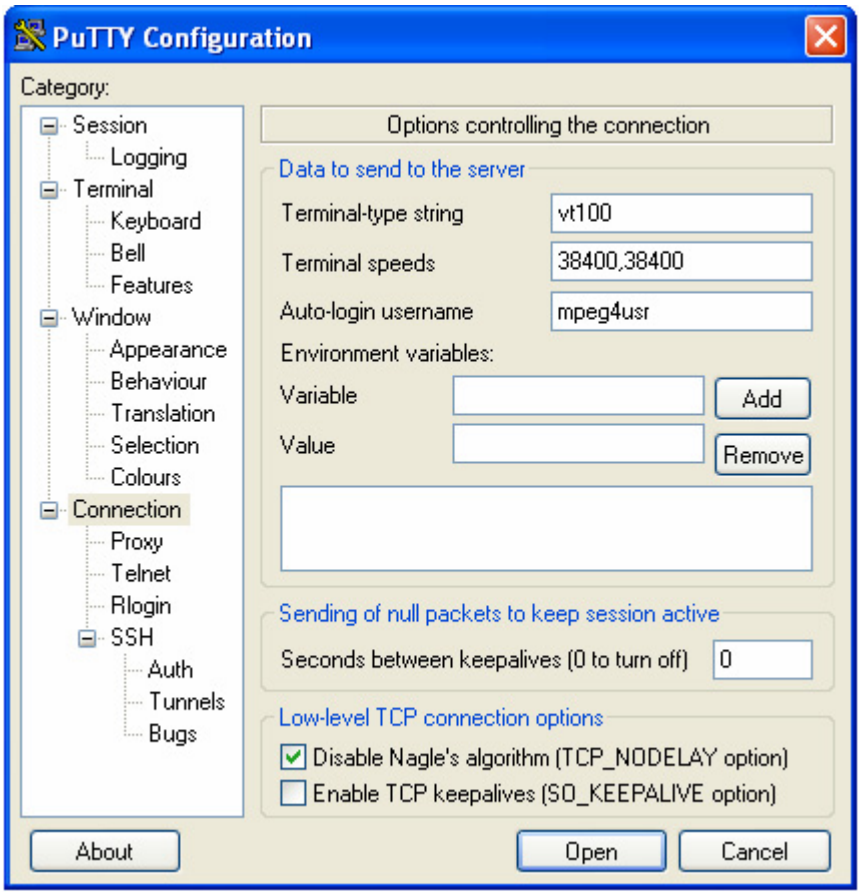
© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale



Now select “Connection” in the “Category” list. It will bring the following window. Put the preferred username, e.g. “mpeg4usr” in the “Auto-login username” field.

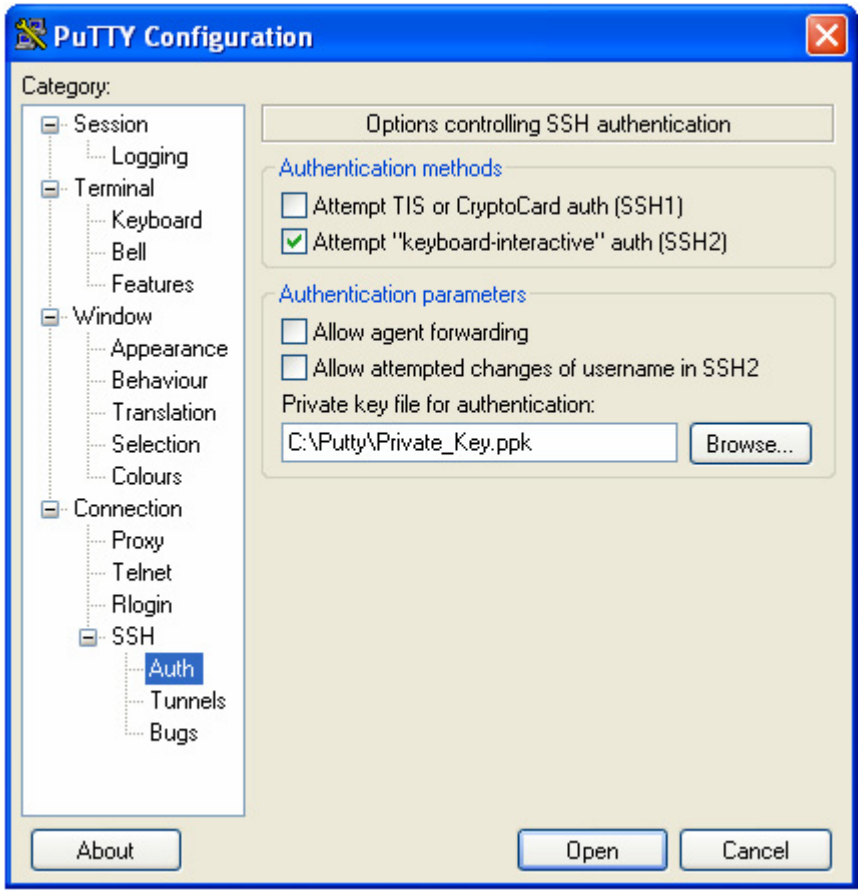
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2009 – All rights reserved



Select “Auth” in the “Category” list and browse and select the private key, e.g. “C:\Putty\Private_Key.ppk” as shown in the following figure. Now, select “Session” in the “Category”, click on “Save” button and exit *Putty*. This completes the configuration of *Putty*.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale



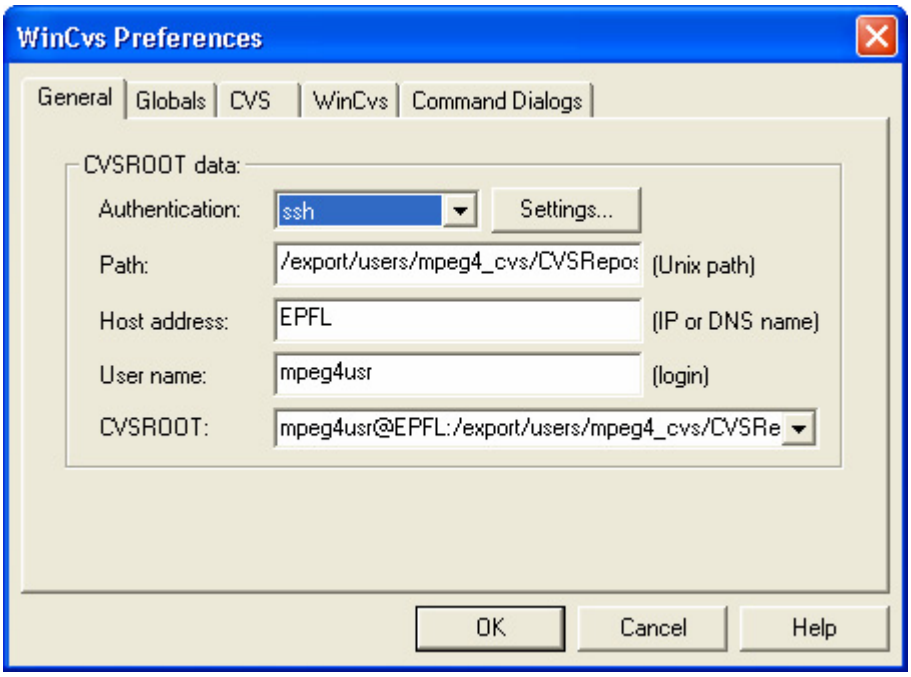
B.1.2.3 WinCVS setup

This section describes the setup and configuration of the *WinCVS* client. After installing the software, click on "Admin → Preferences". This will bring the "WinCvs Preferences" dialog box as shown in the following figure. Select "ssh" as the authentication protocol, type "EPFL" (the name that you used in the "Saved session" of *PuTTY*) in the "Host address" and the preferred username, e.g. "mpeg4usr" in the "User name". In the CVSROOT field, enter

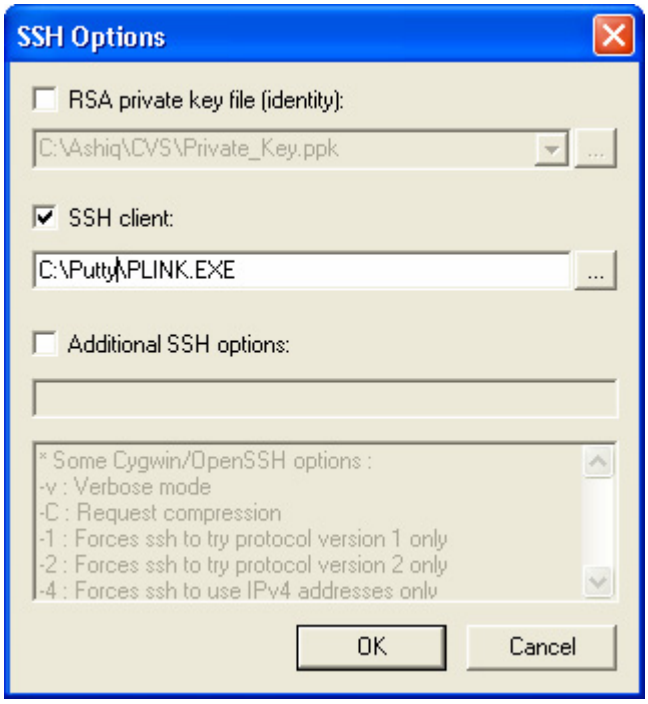
<*your-username*>@<*your-cvssession*>:/export/users/mpeg4_cvs/CVSRepository

where, <*your-username*> is the preferred username and
<*your-cvssession*> is the name of the "Saved session" entered in *PuTTY*

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2009 – All rights reserved



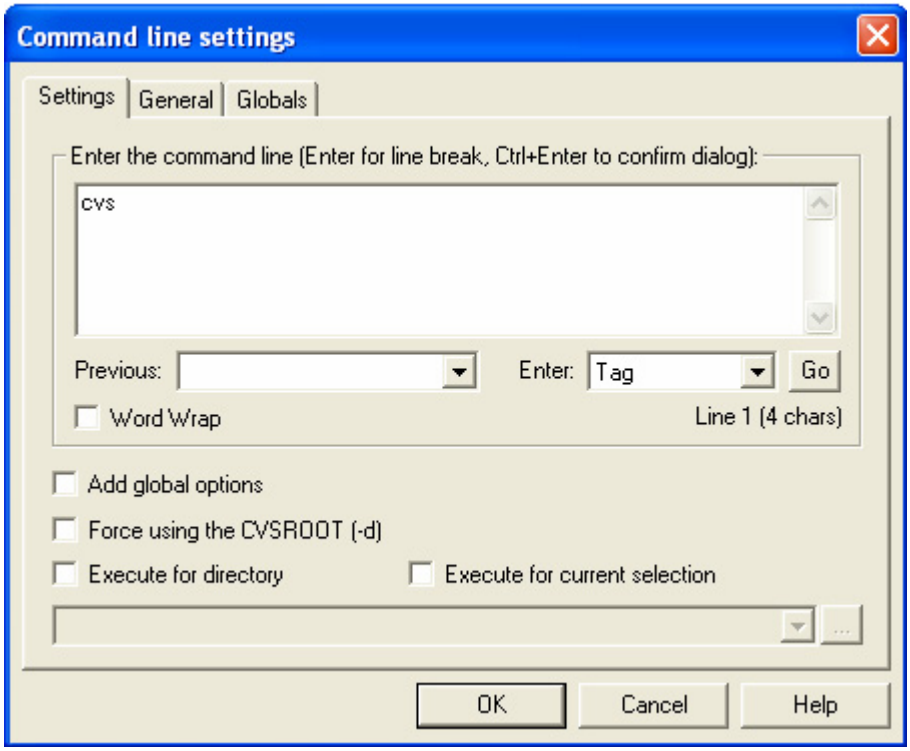
Now click on “Settings” button to open a second dialog box, and enter “plink.exe” in the “SSH client” field as shown in the following figure.



The *WinCVS* has now been setup. To work with it one should use the command line. To bring it, click on “Admin → Command Line”. The following dialog box will appear.

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale



To execute the command in a specific directory click on the “Execute for directory” and browse and select the directory from local hard drive. Now on the “Enter the command line” field the CVS commands can be entered and executed. This is discussed in the following section.

B.1.3 Basic CVS commands

Usage: **cvs** [cvs-options] **command** [command-options] [files...]

Some of the most commonly used CVS commands are:

cvs **checkout** *filename*: Checkout sources for editing

cvs **add** *filename*: Adds a new file/directory to the repository

cvs **commit** *filename*: Checks files into the repository

cvs **remove** *filename*: Remove a file/directory from the repository

cvs **update** *filename*: Brings work tree in sync with repository

cvs **diff** *filename*: Runs diffs between revisions

cvs **history** *filename*: Shows status of files and users

cvs **status** *filename*: Status info on the revisions

cvs **release** *filename*: Status info on the revisions

The top 5 commands are discussed below. To find a detail list and description of the CVS commands and options please refer to [10].

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2009 – All rights reserved

B.1.3.1 Checkout

Make a working directory (specified in the “Execute for directory” of the command line settings) containing copies of the source files specified by *modules/filename*. You must execute checkout before using most of the other CVS commands, since most of them operate on your working directory. *checkout* is used to create directories. The top-level directory created is always added to the directory where checkout is invoked, and usually has the same name as the specified module.

B.1.3.2 Add

Use the *add* command to create a new file or directory in the source repository. The files or directories specified with add must already exist in the current directory. The added files are not placed in the source repository until you use *commit* to make the change permanent.

B.1.3.3 Commit

Use *commit* when you want to incorporate changes from your working source files into the source repository. If you don't specify particular files to commit, all of the files in your working current directory are examined. *commit* is careful to change in the repository only those files that you have really changed. By default, files in subdirectories are also examined and committed if they have changed.

B.1.3.4 Remove

Use this command to declare that you wish to remove files from the source repository. Like most CVS commands, *remove* works on files in your working directory, not directly on the repository. As a safeguard, it also requires that you first erase the specified files from your working directory. The files are not actually removed until you apply your changes to the repository with *commit*.

B.1.3.5 Update

After you have run *checkout* to create your private copy of source from the common repository, other developers will continue changing the central source. From time to time, when it is convenient in your development process, you can use the *update* command from within your working directory to reconcile your work with any revisions applied to the source repository since your last *checkout* or *update*.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2009 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex C
(Informative)

Additional utility software

Software that appears in this Annex has proven to be useful to the developers of the standard but is not a normative reference implementation.

Software used for simulation of HDL is Model Technology's MTI 5.8 C version.

Software used for synthesis of HDL is Synplicity's Synplify Pro 7.5.1 version.

Software used for place and route of HDL is Xilinx ISE 6.1.03.

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex D
(Informative)

Providers of reference hardware code

The following organizations have contributed software referenced in this part of ISO/IEC 14496.

Xilinx Research Labs
University of Calgary, Canada
Dublin City University, Ireland
University of Alveiro Portugal
École Polytechnique Fédérale de Lausanne

© ISO/IEC 2009 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Bibliography

[1] Xilinx Inc., "Virtex™-II Platform FPGAs: Complete Data Sheet", document DS031, October 2003.

[2] Y. Shoham and A. Gersho, "Efficient bit allocation for an arbitrary set of quantizers", *IEEE Transactions on Audio, Speech and Signal Processing*, Vol. 36, No. 9, September 1988, pp. 1445-1453.

[3] A. Navarro, P. Gouveia, and A. Silva, "Delta Rate Control for DV Coding Standard", *IEEE Symposium on Consumer Electronics*, September 2004, Reading, UK.

[4] ISO/IEC 14496-2:2004, *Information technology – Coding of audio-visual objects – Part 2: Visual*.

[5] Annapolis Micro Systems, "WILDCARD™-II Reference Manual", document 12968-000 Revision 2.6, January 2004.

[6] Annapolis Micro Systems, "WILDCARD™-II & 4 Reference Manual", document 12968-0000 Revision 3.2, December 2005.

[7] Xilinx Inc., "Virtex-4 User Guide", document UG070 (v1.5), March 2006.

[8] "PuTTY: A Free Telnet/SSH Client", http://www.chiark.greenend.org.uk/~sgtatham/putty/, valid on 2007-10-17.

[9] "WinCVS/CvsGui", http://www.wincvs.org, valid on 2007-10-17.

[10] "Reference manual for CVS commands", http://www.cs.utah.edu/dept/old/texinfo/cvs/cvs_18.html, valid on 2007-10-17.

[11] IEEE Std 1180-1990, "IEEE Standard Specifications for the implementations of 8x8 Inverse Discrete Cosine Transform", December 1990.

[12] E. Feig and S. Winograd. "Fast algorithms for the discrete cosine transform", *IEEE Transactions on Signal Processing*, Vol. 40, No. 9, September 1992, pp. 2174-2193.

[13] A. Silva, P. Gouveia, and A. Navarro, "Fast Multiplication-free QWDCT for DV coding standard", *IEEE Transactions on Consumer Electronics*, Vol. 50, No. 1, February 2004, pp 180-187.

[14] ISO/IEC 14496-10:2005, *Information technology – Coding of audio-visual objects – Part 10: Advanced Video Coding*.

[15] UB Video, Inc., "Emerging H.264 Standard: Overview and TMS320DM642-Based Solutions for Real-Time Video Applications", December 2002, available online at http://www.seeddsp.com/pdf/soft/c600003208.pdf, valid on 2007-10-17.

[16] I. E. G. Richardson, "H.264/MPEG-4 Part 10: Transform & Quantization", Vcodex, March 2003, available online at http://www.vcodex.com/h264.html, valid on 2007-10-17.

[17] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC Video Coding Standard", *IEEE Transactions on Circuits and Systems For Video Technology*, Vol. 13, No. 7, July 2003, pp. 560-576.

[18] K. Denolf, C. Blanch, G. Lafruit, and A. Bormans, "Initial memory complexity analysis of the AVC codec", *IEEE Workshop on Signal Processing Systems*, October 2002, pp. 222-227.

[19] T. Stockhammer, M. M. Hannuksela, T. Wiegand, "H.264/AVC in wireless environments", *IEEE Transactions on Circuits and Systems For Video Technology*, Vol. 13, No. 7, July 2003, pp. 657-673.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

[20] M. Horowitz, A. Joch, F. Kossentini, A. Hallapuro, "H.264/AVC Baseline Profile Decoder Complexity Analysis", *IEEE Transactions on Circuits and Systems For Video Technology*, Vol. 13, No. 7, July 2003, pp. 704-716.

[21] H. S. Malvar, A. Hallapuro, M. Karczewicz, and L. Kerofsky, "Low-Complexity Transform and Quantization in H.264/AVC", *IEEE Transactions on Circuits and Systems For Video Technology*, Vol. 13, No. 7, July 2003, pp. 598-603.

[22] R. Schafer, T. Wiegand, and H. Schwarz, "The Emerging H.264/AVC Standard", *EBU Technical Review*, January 2003.

[23] H. Malvar, A. Hallapuro, M. Karczewicz, and L. Kerfosky, "Low-Complexity Transform and Quantization with 16-bit Arithmetic for H.26L", *IEEE International Conference on Image Processing*, Rochester, New York, September 2002.

[24] A. Hallapuro, M. Karczewicz, and H. Malvar, "Low Complexity Transform and Quantization – Part II: Extensions", Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG, doc. JVT –B039r2, February 2002.

[25] I. Amer, W. Badawy, and G. Jullien, "Hardware Prototyping for The H.264 4x4 Transformation", *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing*, Montreal, Quebec, Canada, Vol. 5, pp. 77-80, May 2004.

[26] I. Amer, W. Badawy, and G. Jullien, "A VLSI Prototype for Hadamard Transform with Application to MPEG-4 Part 10", *IEEE International Conference on Multimedia and Expo*, Taipei, Taiwan, June 2004.

[27] A. M. Tekalp, "Digital Video Processing", Prentice-Hall Inc., New Jersey, USA, 1995.

[28] I. E. G. Richardson, "H.264 and MPEG-4 Video Compression: Video Coding for Next-generation Multimedia", John Wiley & Sons Ltd., Sussex, England, December 2003.

[29] G. Sullivan, P. Topiwala, and A. Luthra, "The H.264 Advanced Video Coding Standard: Overview and Introduction to the Fidelity Range Extensions," *SPIE Conference on Application of Digital Image Processing XXVII*, Colorado, USA, August 2004.

[30] I. Amer, W. Badawy, and G. Jullien, "Towards MPEG-4 Part 10 System On Chip: A VLSI Prototype For Context-Based Adaptive Variable Length Coding (CAVLC)," *IEEE Workshop on Signal Processing Systems*, Austin, Texas, USA, October 2004.

[31] I. Amer, W. Badawy, and G. Jullien, "A Proposed Hardware Reference Model for Spatial Transformation and Quantization in H.264," *Journal of Visual Communication and Image Representation*, Special Issue on Emerging H.264/AVC Video Coding Standard.

[32] I. E. G. Richardson, "H.264/MPEG-4 Part 10: Variable Length Coding", October 2002, available online at http://www.vcodex.com/h264.html, valid on 2007-10-17.

[33] Y. W. Huang et. al., "Hardware architecture design for variable block size motion estimation in MPEG-4 AVC/JVT/ITU-T H.264," *Proceedings of the 2003 International Symposium on CAS*, ISCAS '03, pp. II-796-II-799, May 2003.

[34] S. Y. Yap and J. V. McCanny, "A VLSI architecture for variable block size video motion estimation," *IEEE Transactions on CAS II*, Vol. 51, No. 7, July 2004.

[35] P. Kuhn, "Algorithms, Complexity Analysis and VLSI Architectures for MPEG-4 Motion Estimation", Kluwer Academic Publishers, Boston, 1999.

[36] Pereira F., et. al., "The MPEG-4 Book", Prentice Hall PTR, 2002.

[37] T. Sikora and B. Makai, "Shape-Adaptive DCT for Generic Coding of Video", *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 5, No. 1, February 1995, pp 59 – 62.

© ISO/IEC 2009 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

[38] A. P. Chandrakasan and R. W. Brodersen, "Minimizing Power Consumption in Digital CMOS Circuits," *Proceedings of the IEEE*, pp. 498-523, April 1995.

[39] A. Kinane, et. al., “An Optimal Adder-Based Hardware Architecture for the DCT/SA-DCT”, *Proceedings SPIE Video Communications and Image Processing (VCIP)*, Beijing, China, July 2005.

[40] MPEG-4 Part 7 Optimized Reference Software, “microsoft-v2.5-040207-NTU”, available online at http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html.

[41] L. Weiping, et. al., “MPEG-4 Video Verification Model version 18.0”, ISO/IEC JTC1/SC29/WG11 N 3908, Pisa, Italy, January 2001.

[42] D. Larkin, V. Muresan and N. O'Connor, “A Low Complexity Hardware Architecture for Motion Estimation”, *IEEE International Symposium on Circuits and Systems (ISCAS)*, Kos, Greece, 21-24 May 2006.

[43] N. Brady, “MPEG-4 standardized methods for the compression of arbitrarily shaped video objects,” *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 9, No. 8, December 1999.

[44] Hao-Chieh Chang, Yung-Chi Chang, Yi-Chu Wang, Wei-Ming Chao, and Liang-Gee Chen, “VLSI architecture design of MPEG-4 shape coding,” *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 12, No. 9, September 2002.

[45] S. Eckart and C. Fogg, “ISO/IEC MPEG-2 Software Video Codec”, *Proceedings of the SPIE Conference on Digital Video Compressing*, 1995, pp 100 – 109.

[46] P. List, A. Joch, J. Lainema, G. Bjontegaard and M. Karczewicz, “Adaptive Deblocking Filter”, *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 13, No. 7, pp. 614-619, July 2003.

[47] Yu-Wen Huang, To-Wei Chen, Bing-Yu Hsieh, Tu-Chih Wang, Te-Hao Chang and Liang-Gee Chen, “Architecture design for deblocking filter in H.264/JVC/AVC”, *Proceedings of IEEE ICME 2003*, Vol. 1, pp. 693-696, January 2006.

[48] Y. Qiu and W. Badawy, “Reference for Virtual Socket API Function Calls”, ISO/IEC JTC1/SC29/WG11/M12788, Bangkok, Thailand, January 2006.

[49] ITU-T Recommendation H.263, “Video coding for low bit rate communication”, 1998.

[50] ISO/IEC 13818-2:2000, *Information technology – Generic coding of moving pictures and associated audio information: Video*.

[51] A. Joch, F. Kossentini, H. Schwarz, T. Wiegand and G. J. Sullivan, “Performace comparison of video coding standards using Lagrangian coder control“, *Proceedings of IEEE International Conference on Image Processing 2002*, pp. 501-504.

[52] G. Bjøntegaard and K. Lillevold, “Context-adaptive VLC (CVLC) coding of coefficients”, JVT of ISO/IEC MPEG and ITU-T VCEG document, JVT-C028, Fairfax, Virginia, USA, May 2002.

[53] D. Marpe, H. Schwartz and T. Wiegand, “Context-based Adaptive Binary Arithmetic Coding in the H.264/AVC Video Compression Standard“, *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 13, No. 7, pp. 620-636, July 2003.

[54] Y. Yeow and J. V. McCanny, “A VLSI Architecture for Variable Block Size Video Motion Estimation”, *IEEE Transactions on Circuits and Systems*, Vol. 51 No. 7, pp. 384-389, July 2004.

[55] C. A. Rahman and W. Badawy, “UMHexagonS Algorithm Based Motion Estimation Architecture for H.264/AVC”, *Proceedings of the 9th International Database Engineering & Application Symposium (IDEAS’05)*, July 2005.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

ICS 35.040

Price based on 432 pages

© ISO/IEC 2009 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale