H.264 INTRA FRAME CODER SYSTEM DESIGN

ÖZGÜR TAŞDİZEN

SABANCI UNIVERSITY

SPRING 2005

H.264 INTRA FRAME CODER SYSTEM DESIGN

by
ÖZGÜR TAŞDİZEN

Submitted to the Graduate School of Engineering and Natural Sciences
in partial fulfillment of
the requirements for the degree of
Master of Science

Sabancı University
Spring 2005

H.264 INTRA FRAME CODER SYSTEM DESIGN

APPROVED BY:

Assist. Prof. Dr. İlker Hamzaoğlu …….

(Thesis Supervisor)

Assist. Prof. Dr. Ayhan Bozkurt …….

Assist. Prof. Dr. Hasan Ateş …….

DATE OF APPROVAL: …….

ABSTRACT

Recently, a new international standard for video compression named H.264 / MPEG4 Part 10 is developed. This new standard offers significantly better video compression efficiency than previous International standards. However, this coding gain comes with an increase in encoding complexity. This makes it impossible to implement a real-time H.264 video coder using a state-of-the-art embedded processor alone. Therefore, in this thesis, we developed an FPGA-based H.264 intra frame coder system for portable applications targeting level 2.0 of baseline profile.

As part of the system, we first designed a high performance and low cost hardware architecture for real-time implementation of forward transform and quantization and inverse transform and quantization algorithms used in H.264 / MPEG4 Part 10 video coding standard in Verilog HDL. The design is first verified with RTL simulations using Mentor Graphics Modelsim. It is then verified to work on a Xilinx Virtex II FPGA on an ARM Versatile Platform development board.

We then designed the top-level H.264 Intra Frame Coder System targeting 30 fps CIF encoding. The system consists of search, mode decision and coding parts. The mode decision part implements a Hadamard Transform based mode decision algorithm. The coding part is implemented by integrating Transform-Quant module with CAVLC and Intra Prediction modules. The top-level design is verified with RTL simulations using Mentor Graphics Modelsim.

The complete H.264 Intra Frame Coder System is verified to work on an ARM Versatile Platform development board. The verification includes first capturing an RGB image, converting it into YCbCr format, partitioning the image into macroblocks, and writing it into an SRAM using the software running on ARM9EJ-S processor. Then, the intra frame coder hardware mapped to the Xilinx Virtex II FPGA using Lenoardo Spectrum and Xilinx ISE is used to encode the image and reconstruct it. The conversion of reconstructed image into raster scan order and RGB color domain is then performed by software running on ARM9EJ-S processor. The reconstructed image is then displayed on a color LCD panel for visual verification.

.

ÖZET

Yakın tarihte H.264 / MPEG4 Part 10 isimli yeni bir uluslararası standart geliştirilmiştir. Bu yeni standart, kendinden önceki standartlara göre belirgin şekilde daha iyi sıkıştırma verimi sunmaktadır. Fakat bu kodlama verimi beraberinde kodlama karmaşıklığını da getirmiştir. Bu karmaşıklık H.264 video kodlayıcısının son teknoloji gömülü işlemcilerle gerçek zamanlı uygulanmasını imkansız kılmaktadır. Bu yüzden, bu tez çalışmasında, taşınabilir uygulamalar için taban profilinin 2.0 düzeyini hedefleyen FPGA tabanlı H.264 intra çerçeve kodlayıcı sistemi geliştirilmiştir.

Öncelikle, sistemin bir parçası olarak H.264 / MPEG4 Part 10 standardında kullanılan dönüşüm ve nicemleme ile ters dönüşüm ve ters nicemleme algoritmaları için yüksek performanslı ve düşük maliyetli bir donanım mimarisi Verilog HDL kullanılarak tasarlanmıştır. Tasarımın doğrulama işlemi ilk olarak Mentor Graphics Modelsim benzetim programında "RTL" benzetimleri ile, daha sonra da "Xilinx Virtex II" FPGA'sının bulunduğu "ARM Versalite Platform" isimli geliştirme ortamında gerçekleştirilmiştir.

Saniyede 30 tane CIF resmini kodlamayı hedefleyen H.264 intra çerçeve kodlayıcı sistemi gerekli diğer bloklarıyla birlikte tasarlanmıştır. Sistem araştırma, moda karar verme ve kodlama bölümlerinden oluşmaktadır. Moda karar verme ünitesi "Hadamard" dönüşümü tabanlı moda karar verme algoritmasını gerçekleştirmektedir. Kodlama kısmı dönüşüm - nicemleme modülüyle birlikte CAVLC ve intra tahmin etme modülleri ile birlikte entegre edilmiştir. Sistem Mentor Graphics Modelsim benzetim programında "RTL" benzetimleri ile doğrulanmıştır.

Bütün H.264 intra çerçeve kodlayıcı sisteminin çalışması "ARM Versalite Platform" geliştirme ortamında doğrulanmıştır. Doğrulama işlemi, ARM9EJ-S işlemcisinde çalışan yazılımların bir resmi RGB biçiminde elde etmesini, bunu YCbCr biçimine dönüştürmesini, resmi makro bloklara bölmesini ve SRAM hafızasına yazmasını kapsamaktadır. Daha sonra "Lenoardo Spectrum" ve "Xilinx ISE" programları kullanılarak "Xilinx Virtex II" FPGA'sına yüklenen H.264 intra çerçeve kodlayıcı donanımı ile resim kodlanmış ve yeniden oluşturulmuştur. Yeniden oluşturulan resmin televizyon taraması şekline ve RGB renklerine dönüştürülmesi ARM9EJ-S işlemcisinde çalışan yazılım aracılığıyla olmuştur. Yeniden oluşturulan resim renkli LCD ekranda gösterilerek görsel anlamda doğrulanmıştır

To my Family
to Grandmoms
and to Sedef…

ACKNOWLEDGEMENTS

First of all I would like to thank Dr. Ilker Hamzaoglu who gave me the chance to do this interesting work. I appreciate very much for his constant guidance, suggestions, and help during this project.

I also want to thank Dr. Yucel Altunbasak, who played an important role in initiating H.264 research project at Sabanci University; Dr. Hasan Ates, who assisted us with H.264 algorithm design and Dr. Ayhan Bozkurt, who participiated in my thesis jury.

I also want to thank my partners in H.264 research project for valuable discussions and feedback throughout the project; Esra Sahin, Sinan Yalcin, Mehmet Guney, Mustafa Parlak. In addition to these friends during my two years study at Sabanci University, I am grateful for the company of my friends Akin Unal, A. Ozgur Cakmak, Can Sumer.

I am of course indebted to my whole family, without them I would not be in such a position and special thanks to Sedef Cakir for her friendship.

Lastly, my acknowledgements go to Sabanci University for supporting our project.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

AHB	Advanced High-speed Bus
AMBA	Advanced Microcontroller Bus Architecture
ARM	Advanced RISC Machines
ASIC	Application Specific Integrated Circuit
ASP	Advanced Simple Profile (MPEG24)
AVC	Advanced Video Coding
BOPS	Billion Operations Per Second
CABAC	Context-Adaptive Binary Arithmetic Coding
CAVLC	Context Adaptive Variable Length Coding
CIF	Common Intermediate Format
CISC	Complex Instruction Set Computer
CLB	Configurable Logic Block
CODEC	Coder, Decoder Pair
CPU	Central Processing Unit
DCT	Discrete Cosine Transform
Dff	D Flip Flop
DLL	Delay Locked Loop
DMA	Direct Memory Access
DSP	Digital Signal Processor
DVD	Digital Versatile Disc
ETM	Embedded Trace Macrocell
FCC	U.S. Federal Communications Commission
FPGA	Field Programmable Gate Array
GPIO	General Purpose Input Output
HDL	Hardware Description Language
HDTV	High Definition Television
HHR	Horizontal High Definition
HLP	High Latency Profile (H.263++)
ISDN	Integrated Services Digital Network

ISO/IEC	International Standards Organization, International Electrotechnical Commission
ITU-T	International Telecommunications Union, Telecommunications Standardization Sector
JTAG	Joint Test Access Group
JVT	Joint Video Team
LAN	Local Area Network
LCD	Liquid Crystal Display
LED	Light Emitting Diode
MB	Macroblock
MP@ML	Main Profile at Main Level
MPEG	Motion Picture Experts Group
MPMC	Multi-Port Memory Controller
NAL	Network Abstraction Layer
PCI	Peripheral Component Interconnect
PLD	Programmable Logic Device
PLL	Phase Locked Loop
PSNR	Peak Signal Noise Ratio
PVT	Process Voltage Temperature
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
RTC	Real Time Clock
RTP	Real-Time Protocol
QCIF	Quadrature Common Intermediate Format
QP	Quantization Parameter
QVGA	Quarter Video Graphics Array
SAD	Sum of Absolute Difference
SATD	Sum of Absolute Transformed Difference
SCI	Smart Card Interface
SDRAM	Dynamic Random Access Memory
SDTV	Standard Definition Television
SHDTV	Super High Definition Television
SI	Switching Intra
SRAM	Static Random Access Memory

SP	Switching Prediction
SSMC	Synchronous Static Memory Controller
SSP	Synchronous Serial Port
SVGA	Super Video Graphics Array
UART	Universal Asynchronous Receive Transmit
UMC	United Microelectronic Corporation
USB	Universal Serial Bus
VCEG	Video Coding Experts Group
VCD	Video Compact Disc
VCL	Video Coding Layer
VGA	Video Graphics Array
VIC	Vectored Interrupt Controller
VLC	Variable-Length Coder

CHAPTER 1

INTRODUCTION

1.1 Motivation

Video compression systems are used in many commercial products, from consumer electronic devices such as digital camcorders, cellular phones to video teleconferencing and video broadcasting systems. These applications make the video compression hardware devices an inevitable part of our lives. To improve the performance of the existing applications and to enable the applicability of video compression to new applications a new international standard for video compression is developed. This latest video compression standard H.264 / MPEG4 Part 10 is standardized by both ISO and ITU. H.264 offers significantly better video compression efficiency than previous international standards. Compared to other video compression standards, H.264 performs nearly 50% better than the next best video compression standard.

The H.264 standard includes a Video Coding Layer (VCL), which efficiently represents the video content, and a Network Abstraction Layer (NAL), which formats the VCL representation of the video and provides header information in a manner suitable for transportation by particular transport layers or storage media [1]. This thesis is concerned with the Video Coding Layer of the standard. In this thesis, we studied this block based video coding layer and developed an efficient H.264 Intra Frame Coder System.

The following sub sections explain why video compression is required, why H.264 standard is implemented and why FPGA-based implementation is chosen.

1.1.1 The Need for Video Compression

Digital video contains huge amount of data and in order to be represented with fewer number of bits it needs to be compressed. The emergence of video compression techniques addresses this problem and makes it feasible to store high quality video on a limited-storage space or transmit it within the limited-bandwidth that networks can provide.

The continuous development in technology makes production of image sensors having up to 16M pixels possible. 16M pixels create an image with a resolution of 4096 x 4096 pixels. Considering even a CIF sized image, 352 x 288 pixels, is enough to understand why compression is absolutely required. In full color depth (true color format) 24 bits per pixel are used to carry the color information and in reduced format, for example in 4:2:0 format, 12 bits per pixel are needed to carry the color information. That means in 4:2:0 format, 1188 bytes, which is equal to 1,16 MB, are required to represent a CIF image. For 30 frames per second CIF sized video, 35640 bytes, which is equal to 34,8 MB, are required to represent one second video. This clearly shows that, to save storage space or to reduce required transmission bandwidth (or time) video compression is a must.

1.1.2 Reasons for Implementing H.264 Standard

H.264 is the latest and most complex video coding standard and as mentioned before, compared to the previous next best standard it typically uses half of the bit rate to encode the same video stream or if it uses the same bit rate to code a video stream, the quality of the decoded video almost doubles.

Video conferencing on Internet, image telephony, digital TV (video broadcasting) digital storage (DVD), set-top-boxes and video-on-demand are the targeted applications of H.264 standard. To emphasize the significance of H.264 standard, digital video broadcasting and DVD storage can be examined. Current digital TV channels support

SDTV format with a resolution of 640x480, which requires 4.3 Mbps per channel when coded with MPEG-2. HDTV has a resolution of 1920x1080 pixels and it requires 19 Mbps per channel when coded with MPEG-2. H.264 saves bandwidth and makes HDTV economically feasible; SDTV format will require 1.5-2 Mbps when coded with H.264 and HDTV format will require 6-9 Mbps when coded with H.264. A single sided, single layer DVD has a storage capability of 4.7 GB and maximum bit rate for DVD playback is 10 Mbps. DVD movies are coded with MPEG-2 for standard definition format. If movies are coded with H.264 then even high definition movies will fit in a 4.7 GB DVD.

1.1.3 Reasons for FPGA-based Implementation

In order to perform 50% better than previous best video compression standard, H.264 encoder requires very high computational power which makes its real time implementation very challenging. There are four possible H.264 encoder implementations; general purpose processor, Digital Signal Processor (DSP), ASIC or FPGA implementations. The goal of this thesis is the implementation of real time coding of 30 CIF sized frames per second corresponding to baseline profile of H.264 standard. Since this implementation targets portable applications, its area and power consumption should also be small. In the following paragraphs, we analyze the suitability of the four alternative implementations for our target application.

The first alternative is using general purpose processors without any hardware acceleration. Because of their architecture, requiring more than one cycle to execute an application-specific operation, general purpose processors may not be capable of real time encoding of a video stream using H.264 standard. As an example, “Using Intel™ VTune™ software running on an Intel Pentium™ III 1.0 GHz general-purpose CPU with 512 MB of memory, achieving H.264/AVC SD with a main profile encoding solution would require approximately 1,600 billion operations per second (BOPS)” [2]. Another example is to implement the full search algorithm for H.264 intra prediction a RISC processor has to

work at least at 522 MHz clock frequency [3]. In addition, general purpose processors have large area and power consumption which makes them unsuitable for implementing an H.264 baseline encoder for portable applications.

The second alternative is using a DSP core. In general, DSP cores offer better performance than general purpose processors for signal processing applications. However, a single DSP core may not be enough for implementing an H.264 encoder. In addition, DSPs also have larger area and power consumption in comparison to ASICs and FPGAs. The power dissipation of a well-executed FPGA design is typically about 20% of the power consumption of a software based DSP system operating at the same sample rate [4]. Therefore, DPSs are not suitable for our goal of implementing an H.264 baseline encoder for portable applications.

The third alternative is an ASIC implementation. ASICs offer high performance with low area and power consumption. They are suitable for real-time implementation of an H.264 encoder and they can satisfy our goal of implementing an H.264 baseline encoder for portable applications. However, ASIC design requires significant human resources and time to market for an ASIC design is long. In addition, after the design is finished, it takes several weeks for the fabrication which can be an iterative process before the silicon works. Finally, unless a mass production of thousands of units is targeted an ASIC solution may not be cost effective.

The last alternative is an FPGA based implementation. The maximum clock frequency that can be achieved on an FPGA for a design is lower compared to the same design implemented as an ASIC with the same process technology. However, the recent FPGAs with full-custom embedded multipliers and block RAMs are suitable for real-time implementation of an H.264 encoder. Even though FPGAs have higher area and power consumption than ASICs, they can still satisfy our goal of implementing an H.264 baseline encoder for portable applications. In addition, FPGA design requires less human resources than ASIC design and time to market for an FPGA design is short. Once the RTL design is finished, the design can be verified on the FPGA immediately. FPGAs are also cost effective unless mass production of thousands of units is targeted. Therefore, in this thesis, we preferred an FPGA based implementation.

1.2 Thesis Organization

The organization of this thesis is as follows:

Chapter 2 starts with a brief overview of video coding and presents the history of video coding standards. After introducing basic terminology of video coding, it explains main features of H.264 baseline profile. Finally, it compares H.264 with previous standards.

Chapter 3 explains Transform - Quant and Inverse Quant - Inverse Transform design in detail. First, it introduces transform and quantization algorithms of H.264. Then it describes the designed hardware in detail and the implementation results are given. CAVLC Hardware and Intra Prediction Hardware are also discussed in this chapter.

Chapter 4 explains the top-level Intra Frame Coder Hardware. The modules used in the Intra Frame Coder Hardware and their scheduling are explained.

Chapter 5 gives general information about ARM Versatile/PB926EJ-S platform and Xilinx Virtex II 8000 FPGA. It then describes the Intra Frame Coder System and its design process.

Chapter 6 presents the conclusions and the future work.

CHAPTER 2

OVERVIEW OF H.264 VIDEO CODING

The latest video compression standard, known as H.264 or MPEG-4 Part 10, is developed with the collaboration of MPEG and VCEG groups. Aiming to develop a better video coding standard, MPEG and VCEG groups join together and form Joint Video Team (JVT). Hence this new video coding standard is called with two different names. H.264 / MPEG-4 Part 10 is not the only standard established by the collaboration of MPEG and VCEG groups. Between 1990 and 1994, JVT worked to develop a standard for DTV and SDTV applications and created H.262 / MPEG-2 standard [1,5]. Figure 2.1 shows the development of video coding standards and their designer groups in chronological order [5].

2.1 MPEG Standards

Motion Picture Experts Group (MPEG) is an ISO/IEC working group, established in 1988 to develop standards for digital audio and video formats. The interests of this group are the applications requiring higher bit rates like coding for moving pictures and digital storage. Examples for these application areas are VCD, DVD, SDTV and HDTV.

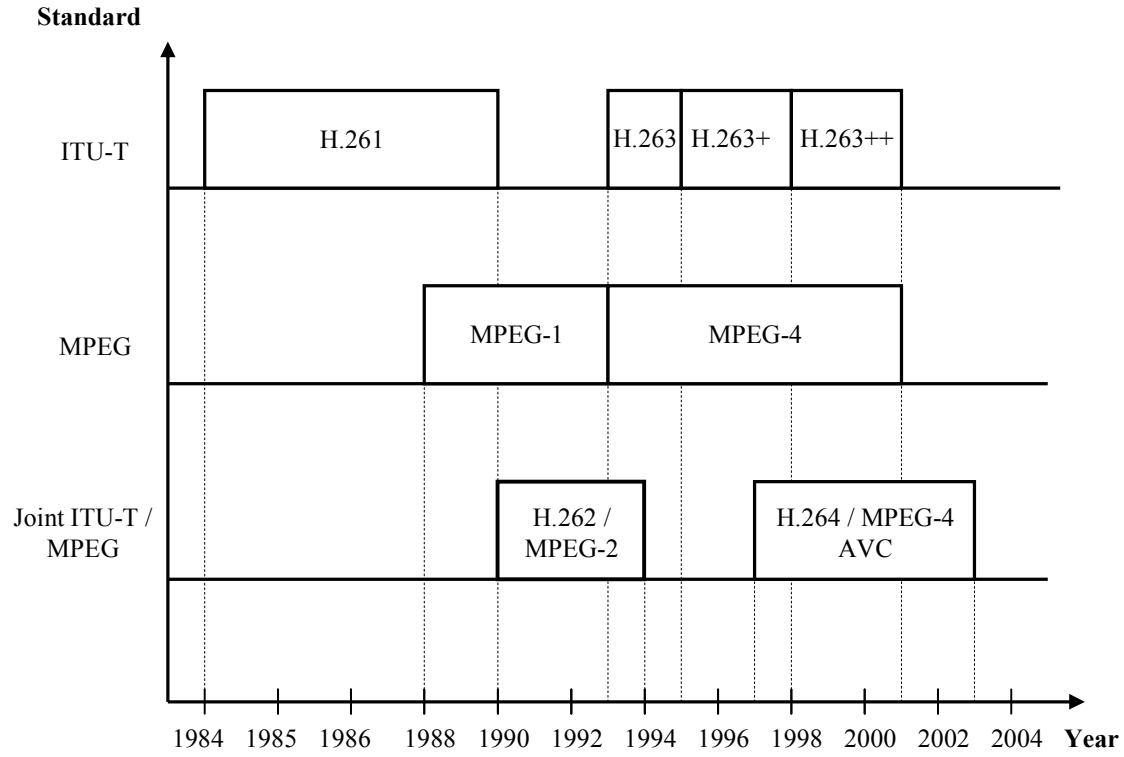

Figure 2.1 Development Years of Video Coding Standards

MPEG group has developed three video compression standards. The first MPEG standard, MPEG-1, is finalized in 1992. MPEG-1 is used for storing audiovisual data on CD-ROM (VCD). It is optimized for video quality at bit rates of 1.1Mbps to 1.5Mbps and has a video quality of nearly the same as VHS cassettes. Three years after the finalization of MPEG-1, MPEG-2 standard is developed. Because MPEG 2 is a standard developed by the collaboration of MPEG and VCEG groups, it is explained in section 2.3. MPEG group then worked on a standard targeting the application field of interactive video. This effort resulted in finalizing MPEG-4 in October 1998 and in the first months of 1999 MPEG-4 became an International Standard. The completely backward compatible extensions under the title of MPEG-4 Version 2 were frozen at the end of 1999, and it received the formal International Standard Status in early 2000. MPEG-4 is successful on three fields: digital television, interactive graphics applications and interactive media [1].

2.2 ITU-T Standards

Video Coding Experts Group (VCEG) is an ITU-T working group. The interests of VCEG are the applications requiring low bit rates, e.g. audiovisual applications such as video conferencing, wireless video, and image telephony.

The first standard by VCEG, H.261, is developed between 1984 and 1990. This video compression standard has been specifically designed for video telecommunication applications like videoconferencing and video telephony over ISDN telephone lines. Since the second standard by VCEG, H.262, is developed in collaboration with MPEG group, it is explained in section 2.3.

The next standard by VCEG, H.263, requires half the bandwidth to achieve the same video quality as H.261. H.263 video compression standard is now widely used in videoconferencing systems and it has largely replaced H.261. It can be applied for a wide range of bit rates up to 20Mbps and it supports five resolutions. In addition to QCIF and CIF that were supported by H.261, the SQCIF, 4CIF, and 16CIF resolutions are also supported. H.263 uses half pixel precision for motion compensation as opposed to integer pixel precision used in H.261. Real Time Transport Protocol (RTP) is used for packing the H.263 video streams for transportation over packet networks. H.263+ is the second version of H.263. It has several additional features and negotiable additional modes. It supplies SNR scalability as well as spatial and temporal scalability. It has custom source formats. It uses advanced intra coding to improve the compression efficiency by using spatial prediction of DCT coefficient values. H.263++ is the third version of H.263. It is completed in 2001 and it includes three new modes to boost the coding efficiency, reduce delay and improve error resilience [6].

2.3 Joint ITU-T / MPEG Standards

The first outcome of the MPEG and VCEG collaboration is H.262 / MPEG-2. It is similar to MPEG-1, but it also supports interlaced video. It is designed for bit rates between 1.5 and 15 Mbps. It is not optimized for bit-rates less than 1 Mbps. It outperforms MPEG-1 at 3 Mbps and above. All decoders in compliance with H.262 / MPEG-2 standard are capable of playing back MPEG-1 video streams. MPEG-2 has a wider usage because it supports a wide range of resolutions and bit rates. MPEG-2 standard made the DTV, SDTV, HDTV, and DVD applications possible.

After finalizing the H.263 standard for video telephony in 1995, VCEG started to work on two further development areas: a short-term effort to add extra features to H.263, which resulted in the new versions of H.263 standard, and a long-term effort to develop a new standard for low bit rate visual communications. The long-term effort led to the draft H.26L standard, providing significantly better video compression efficiency than previous ITU-T standards. In 2001, MPEG recognized the possible benefits of H.26L and a Joint Video Team (JVT) was established with experts from MPEG and VCEG. JVT developed the draft H.26L model into a full International Standard called Advanced Video Coding (AVC). The new standard is also called ISO MPEG4 Part 10 and ITU-T H.264 [1].

2.4 H.264 Video Coding Standard

The first draft of H.264 / MPEG-4 Part 10 standard is completed in May of 2003. This thesis refers to the algorithms specified in this version of the standard [8,9,10]. Same as the previous video compression standards, the H.264 standard does not specify all the algorithms that will be used in an encoder. Instead, it defines the syntax of the encoded bit stream and functionality of the decoder that can decode this bit stream. H.264 has currently three different profiles; baseline, main and extended, and each profile has 14 levels. A profile is a set of algorithmic features and a level shows encoding capability such as picture

size and frame rate. Baseline profile has lower latency and it is used for wireless video applications and video conferencing. Main profile introduces additional algorithms to increase the compression efficiency and it is used for video broadcasting. Extended profile is used for video streaming applications. Table 2.1 shows the performances of all levels in H.264 standard [10]. In this table "p" stands for progressive and "i" stands for interlaced frames. The complexity of an encoder increases with the increasing level. Since, in this thesis, an H.264 Intra Frame Coder System implementing level 2.0 of baseline profile is designed, the features of H.264 baseline profile are explained below.

Table 2.1 Performances of H.264 Levels

Level	**Performance**
1.0	QCIF @ 15 fps
1.1	QCIF @ 30 fps
1.2	CIF @ 15 fps
2.0	CIF @ 30 fps
2.1	HHR @ 15 or 30 fps
2.2	SDTV @ 15 fps
3.0	SDTV 720x480x20i. , 720x576x25i 10Mbps (max)
3.1	1280x720x30p. , SVGA (800x600) 50+p.
3.2	1280x720x60p.
4.0	HDTV: 1920x1080x30i. , 1280x720x60p. , 2kx1x30p. 20 Mbps (max)
4.1	HDTV: 1920x1080x30i. , 1280x720x60p. , 2kx1x30p. 50 Mbps (max)
5.0	SHDTV / D-Cinema: 1920x1080x60p. ,2,5kx2k..
5.1	SHDTV / D-Cinema:4kx2k..

YCbCr Color Space and 4:2:0 Sampling:

The human visual system appears to distinguish scene content in terms of brightness and color information individually, and with greater sensitivity to the details of brightness than color. Video transmission methods are designed to take advantage of this by using YCbCr color space. YCbCr format divides the 24 bit long pixel data into three 8 bit long components called Y, Cb, and Cr. Luma components, Y, represents brightness and chroma components, Cb and Cr, represent the extent to which the color differs from gray toward blue and red, respectively. Since the human visual system is more sensitive to luma than chroma, H.264 standard uses 4:2:0 sampling. In 4:2:0 sampling, for every four luma (Y) samples, there are two chroma samples; one Cb and one Cr [8].

Progressive Video:

In progressive video, video signal is sampled as a sequence of whole frames. Frames are not divided into fields [8].

I and P Pictures:

I pictures can only contain intra coded macroblocks. P pictures can contain both intra and inter coded macroblocks. In addition, there can also be skipped macroblocks in P pictures [8,9].

Integer Transform:

Transform algorithm is based on a 4x4 integer transform. It is guaranteed by the standard that 16-bit arithmetic is enough to implement the transform algorithm. The algorithm does not include any floating point operations; it only needs integer addition and binary shift operations. In this way, a possible drift between encoder and decoder is avoided. H.264 is the first standard to attain exact equality of decoded video content from all decoders [8,9,10]. Detailed information about transform algorithm is given in chapter 3.

Non-uniform Quantization:

H.264 standard uses a non-uniform quantizer. Quantization parameter can take a value between 0-51. The quantization step size doubles for an increment of 6 in

quantization parameter. That means an increment of 1 in quantization parameter results in 12.2% increment in quantization step size. The quantization algorithm requires an integer multiplication [8,9,10]. Detailed information about quantization algorithm is given in chapter 3.

CAVLC:

Context Adaptive Variable Length Coding (CAVLC) algorithm encodes transformed and quantized residual luminance and chrominance data. CAVLC uses multiple tables for a syntax element. It adapts to the current context by selecting one of these tables for a given syntax element based on the already transmitted syntax elements. Information other than the residual data is coded using Exp-Golomb code words [8,9,10]. Additional information about CAVLC is given in chapter 3.

Loop Filter:

A loop filter, called deblocking filter, is used to enhance the video quality by reducing the blocking artifacts in decoded frames. Deblocking filter flattens the blocking artifacts around each 4x4 block and macroblock boundary without disturbing the sharpness of the picture [8,9,10].

Intra Prediction:

H.264 has two intra coding types for luma samples, intra 4×4 and intra 16×16, and one for chroma samples, intra 8x8. 4x4, 8x8 and 16x16 indicate the size of the intra-predicted block. There are 9 prediction modes for intra 4x4, and 4 prediction modes for intra 16x16 and intra 8x8 prediction types. Using a large number of prediction modes improves the prediction accuracy and reduces bit rate by 10-15% [8,9,10]. Additional information about intra prediction is given in chapter 3.

Variable Block Size Motion Estimation:

H.264 standard supports using different block sizes for motion estimation and compensation. A macroblock can be partitioned into 16x16, 16x8, 8x16, 8x8, 8x4, 4x8 or

4x4 blocks. This improves the performance of motion estimation and contributes to reducing the required bit rate for coding the video signal [8,9,10].

Sub-pel Accurate Motion Estimation:

In addition to integer-pel accurate motion estimation, H.264 standard allows using half-pel and quarter-pel accurate motion estimations to improve the performance of motion estimation algorithm at the expense of additional computational complexity [8,9,10].

Multiple Reference Frames:

H.264 allows using multiple reference frames by motion estimation. This enables the encoder to search the best match for the current macroblock in a wider set of pictures than just using the last encoded picture. This improves the performance of motion estimation algorithm [9].

2.5 H.264 Encoder Flow

As shown in Figure 2.2, an H.264 encoder has a forward path and a reconstruction path. The reconstruction path consists of the modules in the dashed rectangle. The forward path is used to encode a video frame by using intra and inter predictions and to create the bit stream. The reconstruction path is used to decode the encoded frame and to reconstruct the decoded frame. Since a decoder never gets original images, but rather works on the decoded frames, reconstruction path in the encoder ensures that both encoder and decoder use identical reference frames for intra and inter prediction. This avoids possible encoder – decoder mismatches [1,8,10].

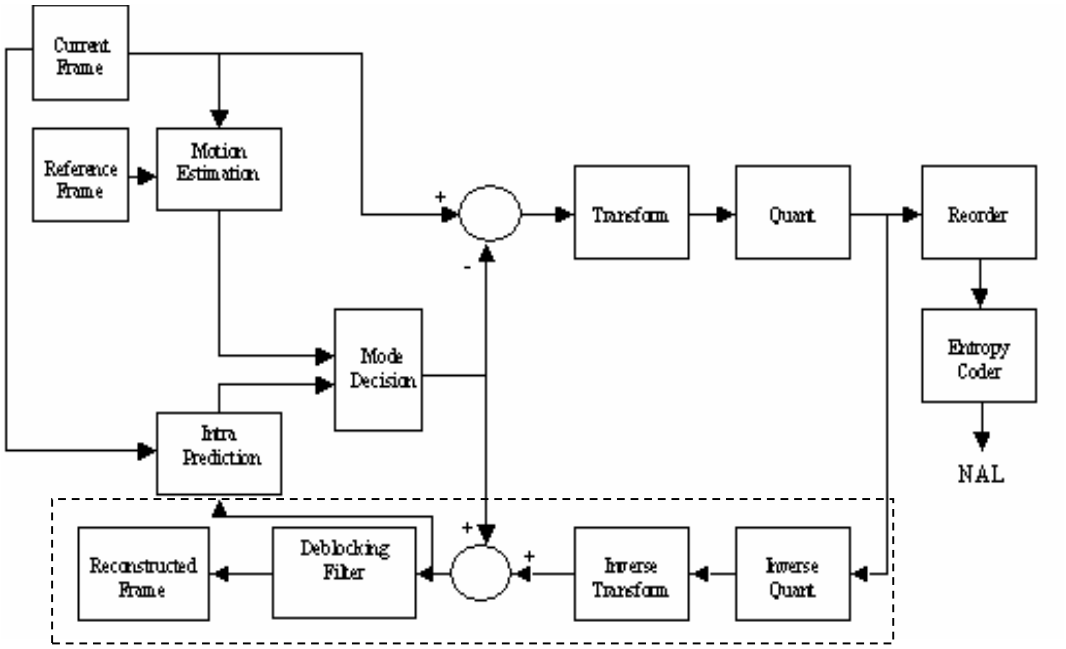

Figure 2.2 Block Diagram of H.264 Encoder

Forward path starts with partitioning the input frame into macroblocks. Each macroblock is encoded in intra or inter mode depending on the mode decision. In both intra and inter modes, the current macroblock is predicted from the reconstructed frame. Intra mode generates the predicted macroblock based on spatial redundancy, whereas inter mode, generates the predicted macroblock based on temporal redundancy. Mode decision compares the required amount of bits to encode a macroblock and the quality of the decoded macroblock for both of these modes and chooses the mode with better quality and bit-rate performance. In either case, intra or inter mode, the predicted macroblock is subtracted from the current macroblock to generate the residual macroblock. Residual macroblock is transformed using 4x4 and 2x2 integer transforms. Transformed residual data is quantized and quantized transform coefficients are re-ordered in a zig-zag scan order. The reordered quantized transform coefficients are entropy encoded. The entropy-encoded coefficients together with header information, such as macroblock prediction mode and quantization step size, form the compressed bit stream. The compressed bit stream is passed to network abstraction layer (NAL) for storage or transmission [1,8,10].

Reconstruction path begins with inverse quantization and inverse transform

operations. The quantized transform coefficients are inverse quantized and inverse transformed to generate the reconstructed residual data. Since quantization is a lossy process, inverse quantized and inverse transformed coefficients are not identical to the original residual data. The reconstructed residual data are added to the predicted pixels in order to create the reconstructed frame. A deblocking filter is applied to reduce the effects of blocking artifacts in the reconstructed frame [1,8,10].

2.6 Comparison of H.264 with Previous Standards

A comparison of the coding efficiency of H.264 standard with the coding efficiency of MPEG-4, MPEG-2 and H.263 standards based on several video sequences is given in [1]. The video sequences used in this study have distinct spatial and temporal characteristics. The video encoders used in this study are optimized for best performance gain with regards to their rate-distortion efficiency using Lagrangian techniques. In order to make a fair comparison between the encoders, a unique and efficient coder control is used. The H.264 encoder implemented the main profile. The H.263 encoder implemented the High Latency Profile (HLP). For both these encoders, five reference frames are used. The H.263 encoder is used quarter-pel-accurate motion compensation with global motion compensation enabled. In addition, the recommended deblocking filter is applied as a post-processing step. The MPEG-4 Visual encoder implemented the Advanced Simple Profile (ASP). The MPEG-2 Visual encoder is generated bit streams at the MP@ML conformance point.

The average bit-rate savings provided by each encoder, relative to all the other tested encoders over the entire set of video sequences are given in Table 2-2. It can clearly be seen that H.264 significantly outperforms all the other standards.

Table 2.2 Average Bit Rate Savings of H.264

Coder	MPEG-4 ASP	H.263 HLP	MPEG-2
H.264	38.62%	48.80%	64.46%
MPEG-4 ASP	-	16.65%	42.95%
H.263 HLP	-	-	30.61%

Another performance comparison of H.264 against MPEG-4 and MPEG-2 is given in Figure 2.3 [7]. Bandwidth requirement, storage space and download time of a 90 minute long DVD quality movie is calculated for all these standards. Download time is calculated with respect to a download speed of 700 Kbps. Once again, the performance advantage of H.264 encoder over the other encoders can clearly be seen.

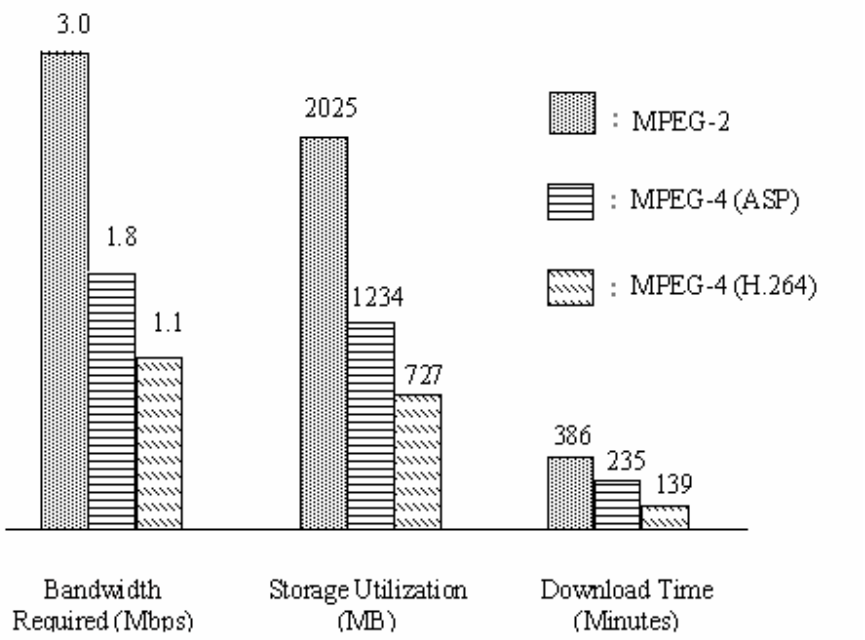

Figure 2.3 Performance Comparisons for 90-Minute DVD Quality Movie

In addition to having a significant performance advantage over previous standards, H.264 standard has two other advantages; error resilience and network friendliness. Unlike previous standards, each transmission packet coded by H.264 can be decoded without depending on the information on the other packets. This makes H.264 a more error resilient

video coding standard [8]. Previous standards targeted transport protocols usually in a circuit switched, bit stream oriented environment. However, H.264 is designed by realizing the importance of packet-based data over fixed and wireless networks and this resulted in a more network friendly video coding structure [8].

CHAPTER 3

HARDWARE ARCHITECURES FOR H.264 INTRA FRAME CODER MODULES

3.1 Transform - Quant and Inverse Quant – Inverse Transform

The basic transform coding process in H.264 is similar to that of previous standards. The process includes a forward transform and quantization followed by zig-zag ordering and entropy coding. The transform coded residual data is also reconstructed. The reconstruction process includes an inverse quantization and inverse transform followed by motion compensation. The reconstructed data before deblocking filter is used for intra prediction in current frame, and the reconstructed data after deblocking filter is used for motion estimation in future frames.

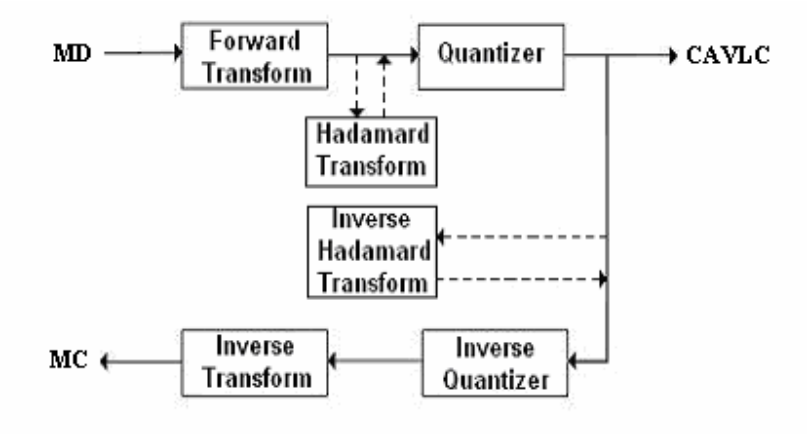

Figure 3.1 Block Diagram of Transform and Quant Algorithms

A detailed flow of the transform and quantization algorithms is presented in Figure 3.1. The input to the forward transform algorithm is a 4x4 block of residual data obtained by subtracting the prediction from the original image data. The transform and quantization algorithms process the blocks in a macroblock as explained in the following sections, and send the resulting data to entropy coder and reconstruction process in the order shown in Figure 3.2.

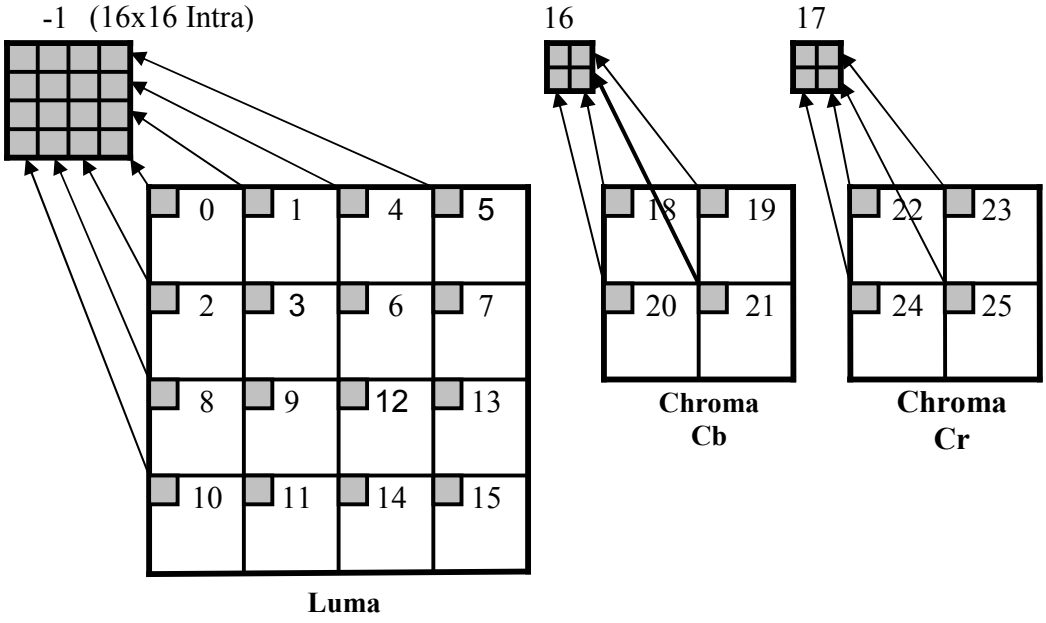

Figure 3.2 Processing Order of Blocks in a Macroblock

3.1.1 Transform Algorithm Overview

H.264 transform algorithm uses four different transform matrices shown in Figure 3.3; 4x4 forward integer, 4x4 Hadamard, 2x2 Hadamard, and 4x4 inverse integer [8,9, 10]. Since 4x4 and 2x2 Hadamard transform matrices are symmetric, inverse Hadamard transform matrices are same as forward Hadamard transform matrices.

In the transform coding process, 4x4 integer transform is applied to all the blocks independent of their prediction type and mode. As shown in Figure 3.2, 4x4 block -1 is

formed by the transformed DC coefficients of 4x4 luminance blocks for the macroblocks that are coded in 16x16 Intra mode, and 2x2 blocks 16 and 17 are formed by the transformed DC coefficients of 4x4 chrominance blocks for all the macroblocks. After the 4x4 integer transform, 4x4 Hadamard transform is applied to block -1 and 2x2 Hadamard transform is applied to blocks 16 and 17.

In the reconstruction process, 4x4 inverse Hadamard transform is applied to block -1, and 2x2 inverse Hadamard transform is applied to blocks 16 and 17. After the inverse Hadamard transforms, 4x4 inverse integer transform is applied to all the blocks independent of their prediction type and mode.

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \begin{bmatrix} x0 & x1 & x2 & x3 \\ x4 & x5 & x6 & x7 \\ x8 & x9 & 10 & x11 \\ x12 & x13 & x14 & x15 \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix}$$

(a)

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} z0 & z1 & z2 & z3 \\ z4 & z5 & z6 & z7 \\ z8 & z9 & z10 & z11 \\ z12 & z13 & z14 & z15 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix}$$

(b)

$$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} z0 & z1 \\ z2 & z3 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

(c)

$$\begin{bmatrix} 1 & 1 & 1 & 1/2 \\ 1 & 1/2 & -1 & -1 \\ 1 & -1/2 & -1 & 1 \\ 1 & -1 & 1 & -1/2 \end{bmatrix} \begin{bmatrix} y0 & y1 & y2 & y3 \\ y4 & y5 & y6 & y7 \\ y8 & y9 & y10 & y11 \\ y12 & y13 & y14 & y15 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1/2 & -1/2 & -1 \\ 1 & -1 & -1 & 1 \\ 1/2 & -1 & 1 & -1/2 \end{bmatrix}$$

(d)

Figure 3.3 Matrices Used in H.264 Transform Algorithm:

(a) 4x4 Forward Integer Transform,

(b) 4x4 Hadamard Transform,

(c) 2x2 Hadamard Transform,

(d) 4x4 Inverse Integer Transform

.

Matrices used for H.264 transform algorithm do not require floating point operations. It can clearly be seen that only binary shift and integer addition / subtraction operations are needed to realize the transform and inverse transform operations of H.264 standard. Consequently, encoder and decoder mismatches are avoided. . As mentioned in chapter 2, there is no drift between the encoded video and the decoded video. In addition, there is no need for a multiplier for implementing these transform and inverse transform operations.

3.1.2 Quantization Algorithm Overview

A quantization parameter (QP), calculated by a rate control algorithm, is used for determining the quantization step size of transform coefficients in H.264 [10,12,13]. There are 52 quantization parameter values. These values are arranged so that an increase of 1 in quantization parameter means an increase of quantization step size by approximately 12%. An increase of quantization step size by approximately 12% means a reduction of bit rate by approximately 12%. Quantization parameters and corresponding quantization step sizes are given in table 3.1. The quality of images reduces with the increase in the quantization parameter, which also results in lower bit consumption.

Table 3.1 Quantization Parameters and Step Sizes

QP	**0**	**1**	**2**	**3**	**4**	**5**	**6**
Qstep	0.625	0.6875	0.8125	0.875	1	1.125	1.250
QP	**18**	**24**	**30**	**36**	**42**	**48**	**51**
Qstep	5	10	20	40	80	160	224

Quantization of AC coefficients is done by using the equation (3.1) and the quantization of DC coefficients is done by using the equation (3.2).

$$|Zij| = (|Wij|.MF + f) >> qbits, \quad sign(Zij) = sign(Wij) \tag{3.1}$$

$$|Zij| = (|Yij|.MF + 2f) >> (qbits + 1), \text{sign}(Zij) = \text{sign}(Yij) \quad (3.2)$$

Wij is the result of 4x4 forward integer transformation and Yij is the result of 4x4 Hadamard transform. Prior to quantization, the result of 4x4 Hadamard transform is divided by 2 and rounded. MF is multiplication factor and depends on QP and positions of pixels of a 4x4 block. Table 3.2 shows the positions of a 4x4 block and table 3.3 lists the corresponding multiplication factors. Floor (QP/6) indicates that quantization parameter is divided by six and then the result is truncated. In equations (3.1) and (3.2), f is a parameter used to avoid rounding errors and it depends on prediction type of the block and QP, qbits is a variable depending on QP.

Table 3.2 Positions of a 4x4 Block

(0,0)	(0,1)	(0,2)	(0,3)
(1,0)	(1,1)	(1,2)	(1,3)
(2,0)	(2,1)	(2,2)	(2,3)
(3,0)	(3,1)	(3,2)	(3,3)

Inverse quantization of AC coefficients is done by using the equation (3.3).) If QP is greater than or equal to12, inverse quantization of DC coefficients after the 4x4 inverse Hadamard Transform is done by using the equation (3.4). Otherwise, if QP is less than 12, inverse quantization of DC coefficients after the 4x4 inverse Hadamard Transform is done by using the equation (3.5).

$$W'ij = Zij.V.2^{\text{floor}(QP/6)} \quad (3.3)$$

$$W'ij = Wqij.V.2^{\text{floor}(QP/6) - 2} \quad (3.4)$$

$$W'ij = [\ Wqij.V + 2^{1 - \text{floor}(QP/6)}\ ] >> (2\text{-floor}\ (QP/6)) \quad (3.5)$$

Table 3.3 Multiplication Factors

Floor (QP/6)	Positions (0,0), (2,0), (2,2), (0,2)	Positions (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

Table 3.4 Rescaling Factors

Floor (QP/6)	Positions (0,0), (2,0), (2,2), (0,2)	Positions (1,1), (1,3), (3,1),(3,3)	Other Positions
0	10	16	13
1	11	18	14
2	13	20	16
3	14	23	18
4	16	25	20
5	18	29	23

Zij is the result of forward quantization and Wqij is the result of inverse Hadamard transformation. V is the rescaling factor, which depends on QP and positions of pixels of a 4x4 block. Table 3.4 shows the rescaling factors.

3.1.3 Proposed Hardware Architecture

It is stated in the standard that 16-bit arithmetic is enough to realize the transform algorithm and proposed hardware is designed based on this. The proposed hardware architecture includes an input register file, a reconfigurable datapath and its control unit, internal register files and an output register file. The reconfigurable datapath and the register files are shown in Figure 3.4 [11]. The reconfigurable datapath is designed for implementing forward and inverse 4x4 and hadamard transform, and quant algorithms. Even though only one multiplier is used in the reconfigurable datapath, the proposed hardware performs transform, quant, inverse quant and inverse transform operations for a macroblock, in the worst case, in 2500 clock cycles. The worst-case occurs for the macroblocks that are coded in 16x16 Intra mode. Therefore, the proposed high performance and low cost hardware can process 30 VGA frames per second at 90 MHz. 384x9 bit input register file stores residual data for a macroblock that will be transform coded including both luminance and chrominance blocks.

The part of the datapath above the dashed line performs transform and inverse transform operations. The registers, adders and shifters in this part of the datapath are shared by forward and inverse transform operations. When the hardware is used to perform forward transform, the control unit configures the datapath to perform the forward transform operations. When it is used to perform inverse transform, the control unit configures the datapath to perform the inverse transform operations. The first row of multiplexers is used for selecting the proper inputs for transform operations. They select the data from input register file for forward transform operations and the data from IQIT register file for inverse transform operations. The second row of multiplexers is used for selecting the proper input data for the first and the second matrix multiplications. They select the data from the first row of multiplexers for the first matrix multiplication operations and the data from register 0, register 1, register 2, register 3 for the second matrix multiplication operations.

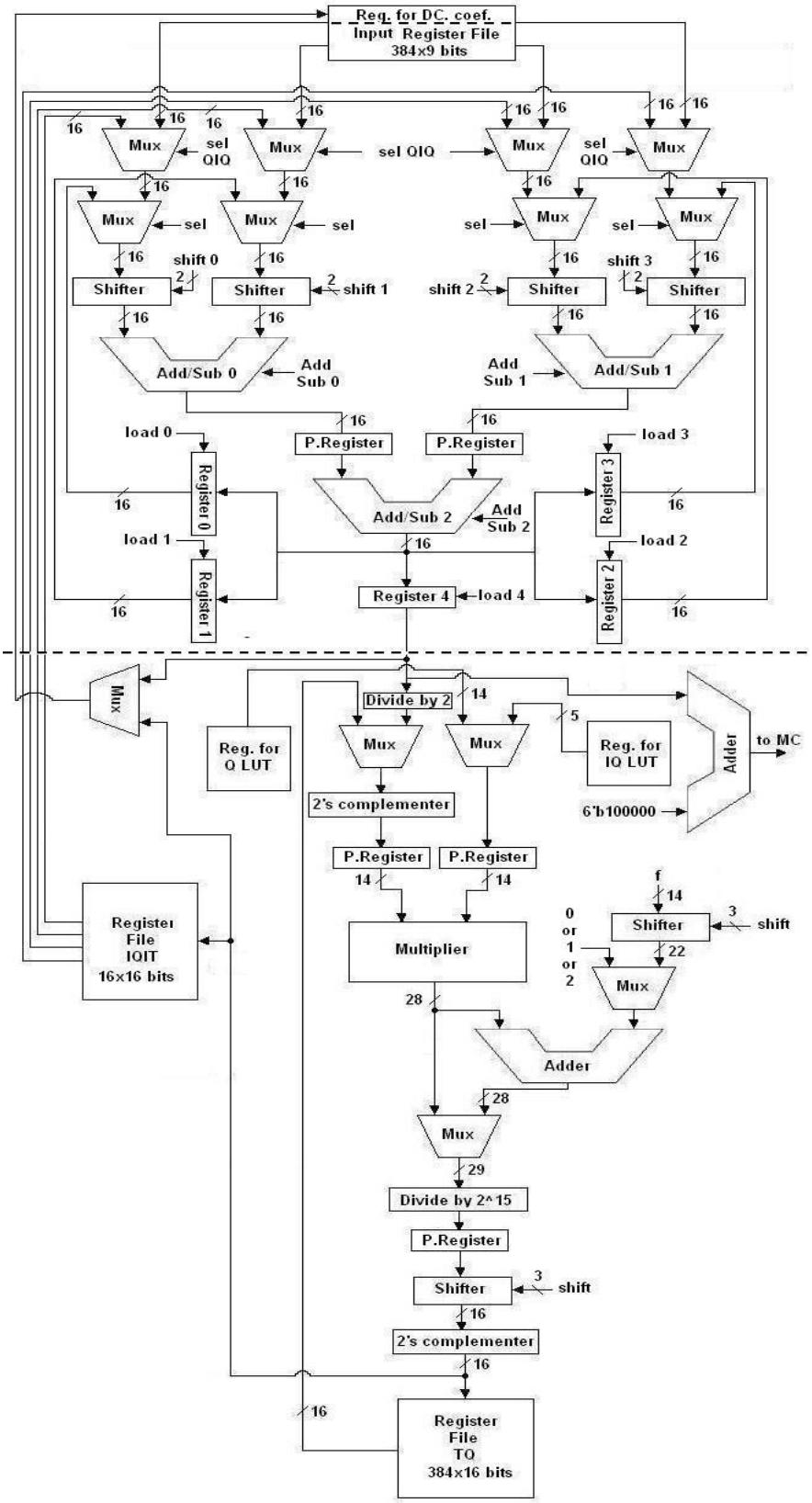

Figure 3.4 Transform – Quant Hardware

Shifters are one bit shifters used for shifting left (multiply by 2) for forward transform operations and for shifting right (divide by 2) for inverse transform operations.

For transform operations, three adder/subtractors are used to achieve high performance with low hardware cost. The forward integer matrix multiplications, inverse integer matrix multiplications and Hadamard Matrix Multiplications are shown in Figure 3.5, Figure 3.6 and Figure 3.7, respectively. The matrix multiplication operations for forward and inverse hadamard transform, which are exactly the same, are given in Figure 3.7.

In Figure 3.5, the four values (x0+x4+x8+x12), (x1+x5+x9+x13), (x2+x6+x10+x14) and (x3+x7+x11+x15) are the results of first forward integer matrix multiplication and they are used for calculating the first row of the result matrix containing the transform coefficients. Similarly, the equations for calculating the transform coefficients in each remaining row of the result matrix have four common values that are used to calculate the corresponding transform coefficients. Therefore, 16-bit registers register 0, register 1, register 2, and register3 are used to store these four common values, i.e. the results of first matrix multiplications. This reduces both the number of cycles and the power consumption of both forward and inverse transform operations. The same method is used to implement the matrix multiplication operations shown in Figure 3.6 and Figure 3.7 as well.

Since the order of the equations used to perform the matrix multiplications for 4x4 and 2x2 Hadamard transforms are not important for functional correctness, we have used the order that gives the lowest power consumption. When the equations are calculated in the order given in Figure 3.7, both the operations (addition or subtraction) performed by adder/subtractor 0 and adder/subtractor 1 and their inputs stay the same between first and second cycles and between third and fourth cycles while calculating each row. Since their inputs and the operations they perform stay the same for two consecutive clock cycles, their outputs stay the same as well. This avoids unnecessary switching activity resulting in lower power consumption for both forward and inverse Hadamard transforms.

P. Registers are pipelining registers used to achieve 80 MHz clock frequency in a 2V8000ff1152 Xilinx Virtex II FPGA with speed grade 5. Register 4 stores the results of forward or inverse transform operations.

[(x0+x4+x8+x12) + (x1+x5+x9+x13) + (x2+x6+x10+x14) + (x3+x7+x11+x15)]
[2*(x0+x4+x8+x12) + (x1+x5+x9+x13) - ((x2+x6+x10+x14) + 2*(x3+x7+x11+x15))]
[(x0+x4+x8+x12) - (x1+x5+x9+x13) - ((x2+x6+x10+x14) - (x3+x7+x11+x15))]
[(x0+x4+x8+x12) - 2*(x1+x5+x9+x13) + 2*(x2+x6+x10+x14) - (x3+x7+x11+x15)]

[(2*x0+x4-x8-2*x12) + (2*x1+x5-x9-2*x13) + (2*x2+x6-x10-2*x14) + (2*x3+x7-x11-2*x15)]
[2*(2*x0+x4-x8-2*x12) + (2*x1+x5-x9-2*x13) – ((2*x2+x6-x10-2*x14) + 2*(2*x3+x7-x11-2*x15))]
[(2*x0+x4-x8-2*x12) - (2*x1+x5-x9-2*x13) – ((2*x2+x6-x10-2*x14) - (2*x3+x7-x11-2*x15))]
[(2*x0+x4-x8-2*x12) - 2*(2*x1+x5-x9-2*x13) + 2*(2*x2+x6-x10-2*x14) - (2*x3+x7-x11-2*x15)]

[(x0-x4-x8+x12) + (x1-x5-x9+x13) + (x2-x6-x10+x14) + (x3-x7-x11+x15)]
[2*(x0-x4-x8+x12) + (x1-x5-x9+x13) – ((x2-x6-x10+x14) + 2*(x3-x7-x11+x15))]
[(x0-x4-x8+x12) - (x1-x5-x9+x13) – ((x2-x6-x10+x14) - (x3-x7-x11+x15))]
[(x0-x4-x8+x12) - 2*(x1-x5-x9+x13) + 2*(x2-x6-x10+x14) - (x3-x7-x11+x15)]

(x0-2*x4+2*x8-x12) + (x1-2*x5+2*x9-x13) + (x2-2*x6+2*x10-x14) + (x3-2*x7+2*x11-x15)]
2*(x0-2*x4+2*x8-x12) + (x1-2*x5+2*x9-x13) – ((x2-2*x6+2*x10-x14) + 2*(x3-2*x7+2*x11-x15))]
(x0-2*x4+2*x8-x12) - (x1-2*x5+2*x9-x13) – ((x2-2*x6+2*x10-x14) - (x3-2*x7+2*x11-x15))]
(x0-2*x4+2*x8-x12) - 2*(x1-2*x5+2*x9-x13) + 2*(x2-2*x6+2*x10-x14) - (x3-2*x7+2*x11-x15)]

Figure 3.5 Forward Integer Matrix Operations

[(y0 + y4 + y8 + y12/2) + (y1 + y5 + y9 + y13/2) + (y2 + y6 + y10 + y14/2) + 1/2 * (y3 + y7 + y11 + y15/2)]
[(y0 + y4 + y8 + y12/2) + 1/2 * (y1 + y5 + y9 + y13/2) – ((y2 + y6 + y10 + y14/2) + (y3 + y7 + y11 + y15/2))]
[(y0 + y4 + y8 + y12/2) - 1/2 * (y1 + y5 + y9 + y13/2) - ((y2 + y6 + y10 + y14/2) - (y3 + y7 + y11 + y15/2))]
[(y0 + y4 + y8 + y12/2) - (y1 + y5 + y9 + y13/2) + (y2 + y6 + y10 + y14/2) - 1/2 * (y3 + y7 + y11 + y15/2)]

[(y0 + y4/2 - y8 - y12) + (y1 + y5/2 - y9 - y13) + (y2 + y6/2 - y10 - y14) + 1/2 * (y3 + y7/2 - y11 - y15)]
[(y0 + y4/2 - y8 - y12) + 1/2 * (y1 + y5/2 - y9 - y13) - ((y2 + y6/2 - y10 - y14) + (y3 + y7/2 - y11 - y15))]
[(y0 + y4/2 - y8 - y12) - 1/2 * (y1 + y5/2 - y9 - y13) - ((y2 + y6/2 - y10 - y14) - (y3 + y7/2 - y11 - y15))]
[(y0 + y4/2 - y8 - y12) - (y1 + y5/2 - y9 - y13) + (y2 + y6/2 - y10 - y14) - 1/2 * (y3 + y7/2 - y11 - y15)]

[(y0 - y4/2 - y8 + y12) + (y1 - y5/2 - y9 + y13) + (y2 - y6/2 - y10 + y14) + 1/2 * (y3 - y7/2 - y11 + y15)]
[(y0 - y4/2 - y8 + y12) + 1/2 * (y1 - y5/2 - y9 + y13) – ((y2 - y6/2 - y10 + y14) + (y3 - y7/2 - y11 + y15))]
[(y0 - y4/2 - y8 + y12) - 1/2 * (y1 - y5/2 - y9 + y13) - ((y2 - y6/2 - y10 + y14) - (y3 - y7/2 - y11 + y15))]
[(y0 - y4/2 - y8 + y12) - (y1 - y5/2 - y9 + y13) + (y2 - y6/2 - y10 + y14) - 1/2 * (y3 - y7/2 - y11 + y15)]

[(y0 - y4 + y8 - y12/2) + (y1 - y5 + y9 - y13/2) + (y2 - y6 + y10 - y14/2) + 1/2 * (y3 - y7 + y11 - y15/2)]
[(y0 - y4 + y8 - y12/2) + 1/2 * (y1 - y5 + y9 - y13/2) – ((y2 - y6 + y10 - y14/2) + (y3 - y7 + y11 - y15/2))]
[(y0 - y4 + y8 - y12/2) - 1/2 * (y1 - y5 + y9 - y13/2) - ((y2 - y6 + y10 - y14/2) - (y3 - y7 + y11 - y15/2))]
[(y0 - y4 + y8 - y12/2) - (y1 - y5 + y9 - y13/2) + (y2 - y6 + y10 - y14/2) - 1/2 * (y3 - y7 + y11 - y15/2)]

Figure 3.6 Inverse Integer Matrix Operations

[(z0+z4+z8+z12) + (z1+z5+z9+z13) + (z2+z6+z10+z14) + (z3+z7+z11+z15)]
[(z0+z4+z8+z12) + (z1+z5+z9+z13) – ((z2+z6+z10+z14) + (z3+z7+z11+z15))]
[(z0+z4+z8+z12) - (z1+z5+z9+z13) – ((z2+z6+z10+z14) - (z3+z7+z11+z15))]
[(z0+z4+z8+z12) - (z1+z5+z9+z13) + (z2+z6+z10+z14) - (z3+z7+z11+z15)]

[(z0+z4-z8-z12) + (z1+z5-z9-z13) + (z2+z6-z10-z14) + (z3+z7-z11-z15)]
[(z0+z4-z8-z12) + (z1+z5-z9-z13) – ((z2+z6-z10-z14) + (z3+z7-z11-z15))]
[(z0+z4-z8-z12) - (z1+z5-z9-z13) – ((z2+z6-z10-z14) - (z3+z7-z11-z15))]
[(z0+z4-z8-z12) - (z1+z5-z9-z13) + (z2+z6-z10-z14) - (z3+z7-z11-z15)]

[(z0-z4-z8+z12) + (z1-z5-z9+z13) + (z2-z6-z10+z14) + (z3-z7-z11+z15)]
[(z0-z4-z8+z12) + (z1-z5-z9+z13) – ((z2-z6-z10+z14) + (z3-z7-z11+z15))]
[(z0-z4-z8+z12) - (z1-z5-z9+z13) – ((z2-z6-z10+z14) - (z3-z7-z11+z15))]
[(z0-z4-z8+z12) - (z1-z5-z9+z13) + (z2-z6-z10+z14) - (z3-z7-z11+z15)]

[(z0-z4+z8-z12) + (z1-z5+z9-z13) + (z2-z6+z10-z14) + (z3-z7+z11-z15)]
[(z0-z4+z8-z12) + (z1-z5+z9-z13) – ((z2-z6+z10-z14) + (z3-z7+z11-z15))]
[(z0-z4+z8-z12) - (z1-z5+z9-z13) – ((z2-z6+z10-z14) - (z3-z7+z11-z15))]
[(z0-z4+z8-z12) - (z1-z5+z9-z13) + (z2-z6+z10-z14) - (z3-z7+z11-z15)]

Figure 3.7 Forward and Inverse Hadamard Matrix Operations

The part of the datapath below the dashed line performs forward and inverse quantization operations. The registers, adders, shifters and the multiplier in this part of the datapath are shared by forward and inverse quant operations. When the hardware is used to perform forward quantization, the control unit configures the datapath to perform the forward quant operations. When it is used to perform inverse quantization, the control unit configures the datapath to perform the inverse quant operations.

Register 4 contains the input data for the quantization and inverse quantization operations. P. Registers are pipelining registers used to achieve 80 MHz clock frequency in a 2V8000ff1152 Xilinx Virtex II FPGA with speed grade 5.

The multiplier used in the datapath is a 14x14 unsigned multiplier. Two multiplexers are used for selecting the proper inputs for the multiplier. One of the multiplexers is used to select either a transformed or inverse transformed value coming from register 4 or a quantized value coming from the output register file TQ. The other multiplexer is used to select either a value from quant lookup table or a value from inverse quant lookup table.

The adder at the output of the multiplier and the shifter at one of the inputs of the adder are used to avoid rounding errors that can happen during scaling and rescaling operations.

The 3-bit shifter at the output of the multiplier is used to perform scaling and rescaling operations depending on the value of qbits parameter. The result of the shift operation is converted into two's complement form and stored in the output register file TQ.

The transform and quant operations are executed in a pipelined manner. After a transform coefficient is computed, in the next cycle, this coefficient is quantized in the quant part of the datapath and a new transform coefficient is computed in the transform part of the datapath. Transform and quant operations for a 4x4 block takes 44 clock cycles. Since only one multiplier is used in the datapath, quant and inverse quant operations cannot be pipelined. Inverse transform and inverse quant operations for a 4x4 block take 59 cycles. After all the transform coefficients in a block are quantized, inverse quantization starts followed by inverse transform. Inverse transform operations start when the inverse quantization operations of a 4x4 block are completed. Therefore, inverse quant and transform operations for a 4x4 block take more clock cycles than the forward transform and quant operations for a 4x4 block.

3.1.4 Implementation Results

A high performance and low cost hardware architecture for real-time implementation of H.264 forward transform and quantization and inverse transform and quantization algorithms is developed. The proposed architecture is implemented in Verilog HDL. The implementation is verified with RTL simulations using Mentor Graphics ModelSim SE. The Verilog RTL is then synthesized to a 2V8000ff1157 Xilinx Virtex II FPGA with speed grade 5 using Mentor Graphics Leonardo Spectrum [14]. The resulting netlist is placed and routed to the same FPGA using Xilinx ISE Series 5.2i. The FPGA implementation including input and output register files as well is placed and routed at 81 MHz under worst-case PVT conditions. Since, in the worst-case, it takes 2500 clock cycles to process a MB, the FPGA implementation can code 27 VGA frames (640x480) per second. The FPGA implementation is verified to work in a Xilinx Virtex II FPGA on an Arm Versatile Platform development board.

The FPGA implementation including input and output register files as well used the following FPGA resources; 4054 Function Generators, 2027 CLB Slices, 1 Block Multiplier, and 583 Dffs /Latches, i.e. 4.35% of Function Generators, 4.35% of CLB Slices, 0.60% of Block Multipliers, and 0.61% of Dffs /Latches. The FPGA implementation excluding input and output register files used the following FPGA resources; 2497 Function Generators, 1249 CLB Slices, 1 Block Multiplier, and 581 Dffs /Latches, i.e. 2.68% of Function Generators, 2.68% of CLB Slices, 0.60% of Block Multipliers, and 0.61% of Dffs /Latches [11]. The Verilog RTL is also synthesized to Virtual Silicon UMC 0.18µ standard-cell library using Synopsys Design Compiler. The synthesis results are presented in Table 1 [11]. The netlist excluding input and output register files has an area of 23K gates. The netlist is verified to work at 210 MHz under worst-case PVT conditions with post synthesis simulations. This 0.18µ ASIC implementation can code 70 VGA frames (640x480) per second.

A hardware architecture only for real-time implementation of H.264 forward and inverse transform algorithms is presented in [15]. This hardware achieves higher performance than our hardware design at the expense of a much higher hardware cost. Our

hardware design is a more cost-effective solution for portable applications. They use 16 adders and 16 internal register files in their datapath as opposed to 3 adders and 6 internal register files in the transform part of our datapath. Their datapath has an area of 6538 gates in TSMC 0.35µ technology. Our datapath, on the other hand, has an area of 2904 gates in AMS 0.35µ technology.

Table 3.5 ASIC Implementation Results

	Critical Path Delay [ns]	Area [Gate Count]
Transform part of the Datapath	2.77	1978
Datapath	4.78	12773
Datapath + Control Unit	4.8	23162
Datapath + Control Unit + Input Register File + Output Register File TQ	4.8	130505

3.2 CAVLC

We have taken CAVLC implementation from Esra Sahin's work [16] and integrated it into our design. A brief explanation about the CAVLC hardware is given below.

CAVLC algorithm is used to encode transformed and quantized residual luminance and chrominance blocks in a macroblock in the order shown in Figure 3.2 Block -1 is formed by the DC coefficients of 4x4 luminance blocks only for the macroblocks that are coded in 16x16 Intra Mode. Blocks 16 and 17 are formed by the DC coefficients of 4x4 chrominance blocks for all the macroblocks. All the transformed and quantized 4x4 and 2x2 blocks for a macroblock are given as inputs to CAVLC algorithm in the order shown in Figure 3.2. CAVLC algorithm processes each 4x4 block in zig-zag scan order and each 2x2 block in raster scan order. It encodes each block in the following five steps [9, 10, 12].

Step 1: It generates coeff_token, the variable length code that encodes both the number of non-zero coefficients (TotalCoeff) and the number of trailing ±1 values

(TrailingOnes) in a block. Since the highest non-zero coefficients after the zig-zag scan are often sequences of ±1, CAVLC algorithm encodes the number of high-frequency ±1 coefficients (TrailingOnes) in coeff_token. Since the number of non-zero coefficients in neighbouring blocks is correlated, CAVLC algorithm generates coeff_token for a block context adaptively. It uses one of the four different VLC tables for generating the coeff_token for a block based on the number of non-zero coefficients in the neighbouring blocks as follows. It first calculates a parameter nC based on the number of non-zero coefficients in the left-hand and upper previously coded blocks, nA and nB respectively. If upper and left blocks nB and nA are both available (i.e. in the same coded slice), nC = round ((nA + nB) /2). If only the upper is available, nC = nB; if only the left block is available, nC = nA; if neither is available, nC = 0. As a special case, for 2x2 dc chroma blocks, nC is always set to -1. It, then, selects the VLC table that will be used for generating the coeff_token based on the value of nC.

Step 2: It encodes the sign of each TrailingOne with a single bit in reverse order starting with the highest-frequency TrailingOne.

Step 3: It encodes the level (sign and magnitude) of each remaining non-zero coefficient in the block in reverse order starting with the highest frequency coefficient and working back towards the DC coefficient. The codeword for a level consists of a prefix and a suffix. Since the magnitude of non-zero coefficients tends to be larger near the DC coefficient and smaller towards the higher frequencies, CAVLC algorithm adapts the suffix length for the level parameter depending on recently coded level magnitudes. It sets the suffix length for the first level, except in some special cases, to 0. It then increments the current suffix length, if the magnitude of the current level is larger than a predefined threshold for this suffix length. CAVLC algorithm generates the code length and the codeword for the current level based on its suffix length. When the suffix length for a level is 0, its codeword does not include a suffix. Otherwise, the codeword for the level includes a suffix. The codeword for a level always includes a prefix, but the prefix for a level is generated using different equations in the two cases; when the suffix length for the level is 0 versus when the suffix length for the level is greater than 0 [12].

Step 4: It encodes the total number of zeros before the last non-zero coefficient (Total_Zeros) using a VLC table.

Step 5: It encodes the number of zeros preceding each non-zero coefficient (Run_Before) in reverse order starting with the highest-frequency coefficient. Since after transformation and quantization, blocks typically contain mostly zeros, CAVLC algorithm uses run-length coding to represent strings of zeros compactly.

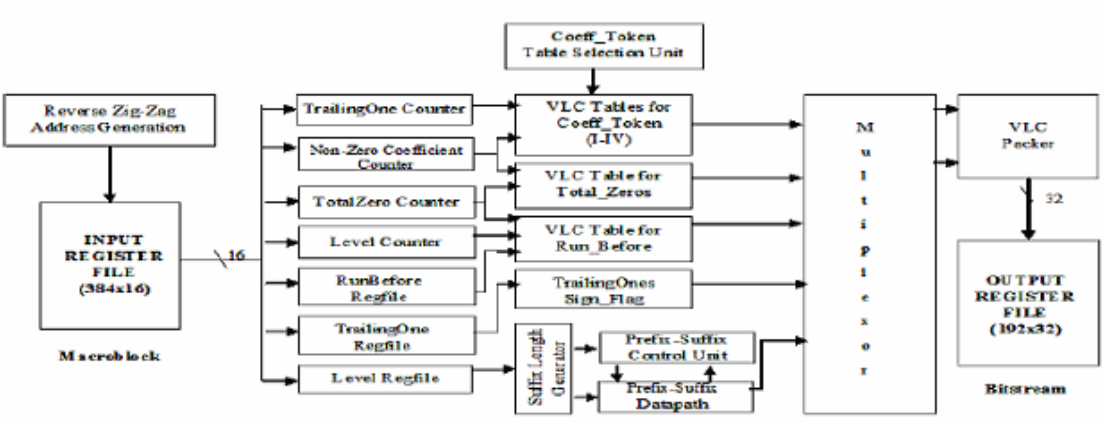

Figure 3.8 Block Diagram of CAVLC Hardware

The proposed hardware architecture for H.264 CAVLC algorithm is shown in Figure 3.8. The proposed hardware performs context-adaptive variable length coding for a macroblock, in the worst case, in 2880 clock cycles. The worst-case occurs for the macroblocks that have no zero coefficients and trailing ±1 coefficients [16].

The CAVLC design is synthesized to a 2v8000ff1157 Xilinx Virtex II FPGA with speed grade 5 using Mentor Graphics Leonardo Spectrum. The resulting netlist is placed and routed to the same FPGA using Xilinx ISE Series 5.2i. The FPGA implementation including input and output register files as well is placed and routed at 76 MHz. Since, in the worst-case, it takes 2880 clock cycles to process a MB, the FPGA implementation can code 22 VGA frames (640x480) per second [16].

3.3 Intra Prediction

Intra predictor hardware used in this thesis is taken from Esra Sahin's work [17]. The intra predictor hardware predicts the values of pixels in the current block from the values of neighboring pixels. For the luma samples, prediction may be formed for each 4x4 sub block or for a 16x16 macroblock. There are 9 prediction modes for each 4x4 luma block and 4 prediction modes for a 16x16 luma block. For chroma samples, the same modes used for 16x16 luma blocks are applied to the 8x8 chroma blocks. 4x4 intra prediction modes are shown in Figure 3.9 and16x16 and 8x8 prediction modes are shown in Figure 3.10 [8,9]. The arrows in Figure 3.9 and 3.10 indicate the direction of prediction in each mode.

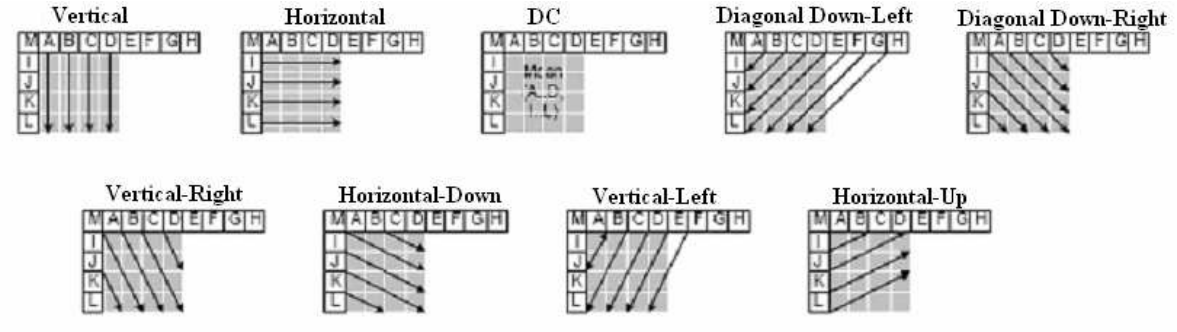

Figure 3.9 4x4 Intra Prediction Modes

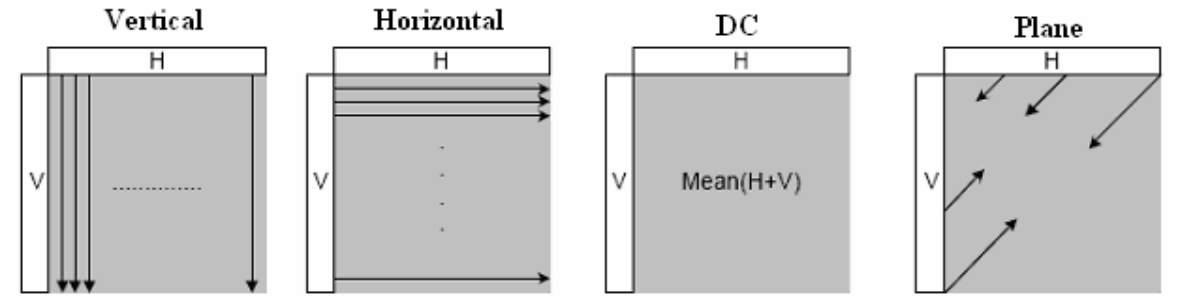

Figure 3.10 16x16 and 8x8 Intra Prediction Modes

Only one intra prediction mode, 4x4 DC mode, is explained in this thesis; all intra modes are explained in [8,9]. In DC mode, as shown in Table 3.6 and Figure 3.11, the

average value of the neighboring pixels are found out. In Figure 3.11, 3-bit shift left operation is symbolized with " << 3 ", which is identical to dividing with 8 and truncating the result. 16x16 and 8x8 DC modes are implemented in the same way; the only difference is the block size.

Table 3.6 Prediction Samples for 4x4 Modes

M	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**
I	a	b	c	d				
J	e	f	g	h				
K	i	j	k	l				
L	m	n	o	p				

$$a = (A + B + C + D + I + J + K + L) << 3$$
$$b = (A + B + C + D + I + J + K + L) << 3$$
$$c = (A + B + C + D + I + J + K + L) << 3$$
$$d = (A + B + C + D + I + J + K + L) << 3$$
$$e = (A + B + C + D + I + J + K + L) << 3$$
$$f = (A + B + C + D + I + J + K + L) << 3$$
$$g = (A + B + C + D + I + J + K + L) << 3$$
$$h = (A + B + C + D + I + J + K + L) << 3$$
$$i = (A + B + C + D + I + J + K + L) << 3$$
$$j = (A + B + C + D + I + J + K + L) << 3$$
$$k = (A + B + C + D + I + J + K + L) << 3$$
$$l = (A + B + C + D + I + J + K + L) << 3$$
$$m = (A + B + C + D + I + J + K + L) << 3$$
$$n = (A + B + C + D + I + J + K + L) << 3$$
$$o = (A + B + C + D + I + J + K + L) << 3$$
$$p = (A + B + C + D + I + J + K + L) << 3$$

Figure 3.11 DC Mode Equations

The intra predictor hardware is shown in Figure 3.12 [17]. It has three different datapaths and control units for 16x16 luma, 8x8 chroma and 4x4 luma modes. Datapaths consist of adders and shifters. Since intra predictor predicts the pixels in the current block

from neighboring pixels of this block, neighboring buffers are used to store the neighboring pixels.

Intra predictor can only use a prediction mode if the neighboring pixels used for predicting the pixels in the current block based on this mode are available for prediction. For example, for the macroblocks in the first row of an image, vertical mode and plane mode cannot be used, because upper neighbors of these macoroblocks are not available. For the macroblocks in the first column of an image left neighbors are not available, this means these macroblocks horizontal and plane modes cannot be used. The availability information for neighboring pixels is provided to the intra predictor hardware as an input, and it performs the prediction based on this availability information.

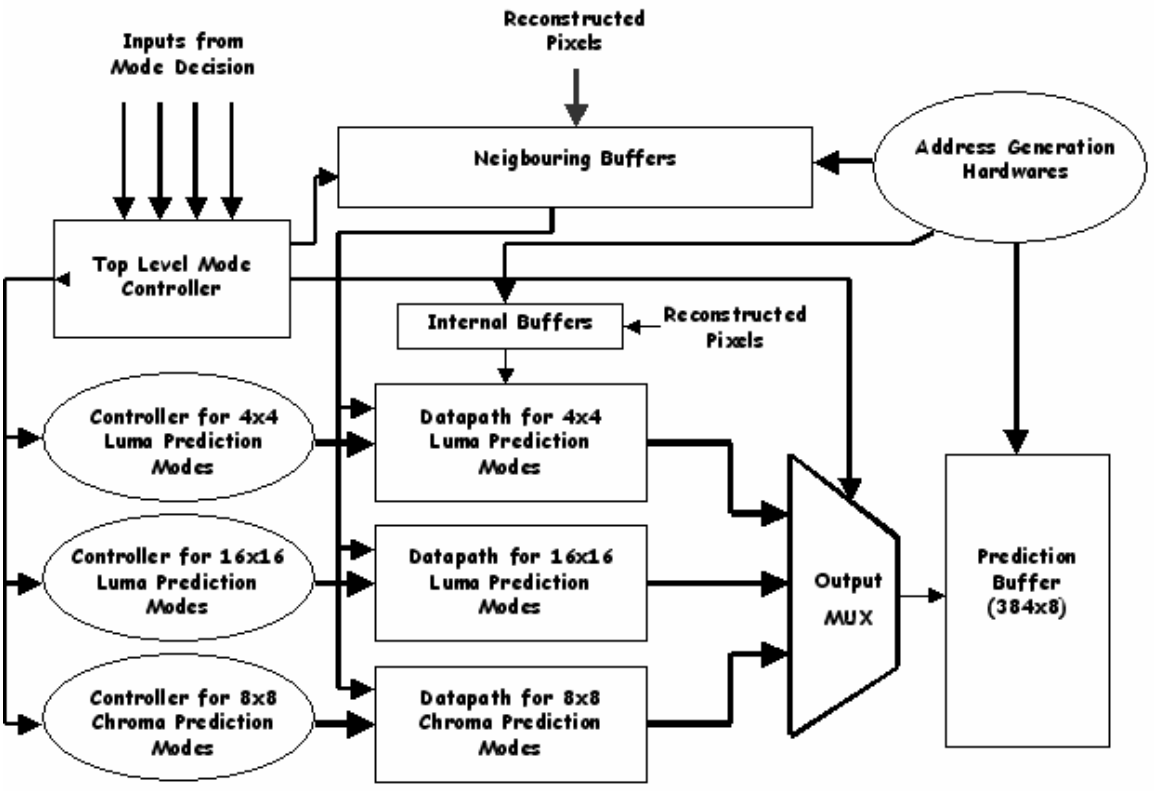

Figure 3.12 Intra Prediction Hardware

CHAPTER 4

TOP LEVEL H.264 INTRA FRAME CODER HARDWARE

H.264 Intra Frame Coder Hardware consists of a search hardware and a coder hardware, and they work in a pipelined manner. Figure 4.1 shows the top-level block diagram of H.264 Intra Frame Coder Hardware and Figure 4.2 shows the top-level scheduling.

Figure 4.1 Top Level Block Diagram

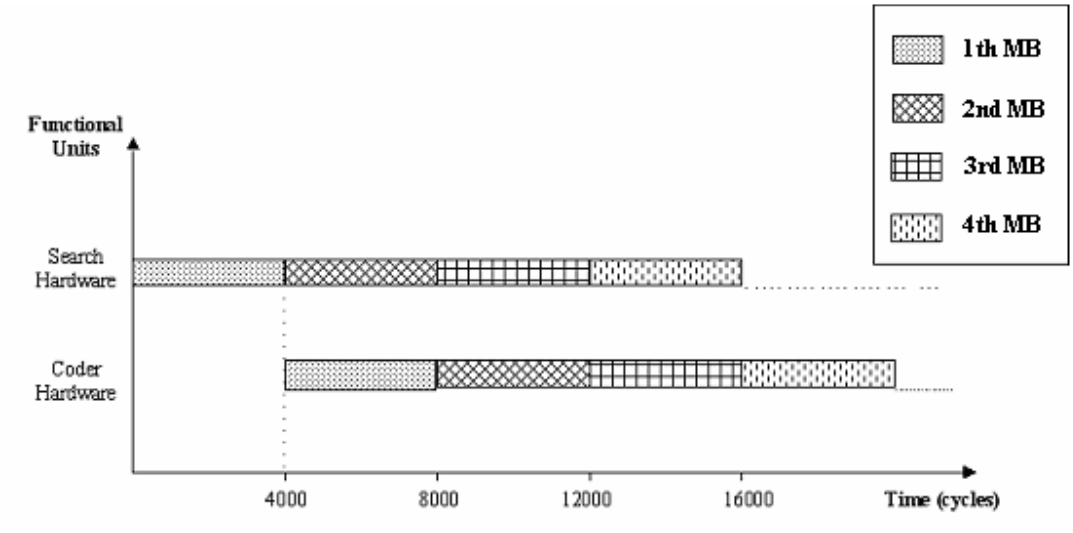

Figure 4.2 Top Level Scheduling

H.264 Intra Frame Coder Hardware works as follows. The first macroblock (MB) of the input frame is loaded to the system and search hardware works on this MB. After the search hardware determines the best mode for the first MB, the first MB is loaded to the current macroblock register file in the coder hardware. As soon as this loading operation finishes, the coder hardware starts to code the first MB based on the selected mode and search hardware starts to work on the second MB. The entire image is processed macroblock by macroblock in this order.

In this thesis, we want to design an H.264 Intra Frame Coder Hardware for portable applications targeting level 2.0 of baseline profile. This requires encoding 30 CIF frames per second (fps). Since a CIF size image has 352x288 pixels, corresponding to 22x18=396 MBs, this requires processing 11800 MBs per second. In Table 4.1, maximum number of clock cycles in which a MB must be coded for different clock frequencies are given. Since search and coder hardware finish their work in 4000 clock cycles, running the H.264 Intra Frame Coder Hardware at 50 MHz is enough for satisfying our target.

Table 4.1 Available Clock Cycles for Different Clock Frequencies

Level	#of MBs per second	@30Mhz	@40Mhz	@50Mhz	@55Mhz	@60Mhz	@65Mhz	@70Mhz
2.0	11880	2525	3367	4208	4629	5050	5471	5892

4.1 Search Hardware

The block diagram of the search hardware is shown in Figure 4.3. Search Hardware includes the intra prediction, residue, high-speed Hadamard transform and mode decision functional units. Register files used in the search part of the H.264 Intra Frame Coder Hardware are current macroblock and intra predicted macroblock register files.

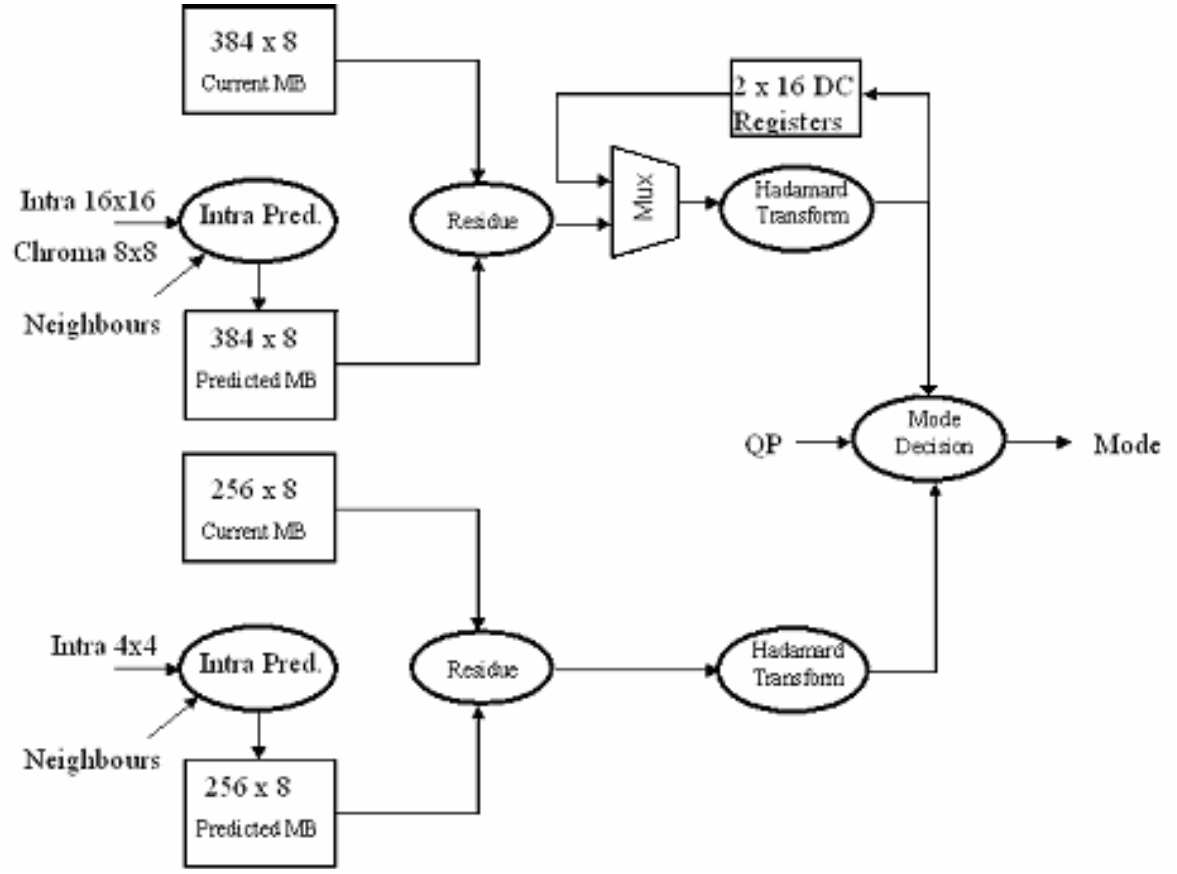

Figure 4.3 Block Diagram of Search Hardware

Mode decision algorithm implemented in the search hardware is same as the algorithm implemented in JM Software when there is no R-D optimization [12,18]. The algorithm is explained below.

Intra 4x4 Mode Decision:

1. For each 4x4 block, apply Hadamard, compute SATD.
2. For each 4x4 block, compute $Cost_{4x4}$ = SATD + 4λR, where R=0 for most probable mode and R=1 for other modes.
3. Choose the mode with minimum cost
4. Repeat this procedure for all 4x4 subblocks
5. Compute total cost for the MB: Σ $Cost_{4x4}$

Lambda:

img->qp : QP parameter (12 <= QP <= 51)

λ = lambda_mode = lambda_motion = QP2QUANT[max(0,img->qp-SHIFT_QP)];

where SHIFT_QP is equal to 12 and QP2QUANT table is given below:

QP2QUANT[40]= { 1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 4, 4, 4, 5, 6, 6, 7, 8, 9,10,11,13,14, 16,18,20,23,25,29,32,36, 40,45,51,57,64,72,81,91 };

Intra 16x16 Mode Decision:

1. Perform 4x4 Hadamard for each 4x4 subblock
2. Extract all DC coefficients from each transformed block, divide these values by 2, form a 4x4 block and apply Hadamard to this DC block.
3. Sum up the absolute values of all the AC coefficients and Hadamard transformed (and scaled) DC coefficients, and take this sum as total cost $Cost_{16x16}$.
4. Repeat this procedure for all modes and choose the 16x16 mode with smallest cost.

Intra 8x8 Mode Decision:

1. Perform 4x4 Hadamard for each 4x4 subblock
2. Sum up the absolute values of all the Hadamard transformed coefficients and take this sum as total cost $Cost_{8x8}$.
3. Repeat this procedure for all modes and choose the 8x8 mode with smallest cost.

Intra 4x4 vs Intra 16x16 Mode Decision:

Compare $Cost_{16x16}$ with (Σ $Cost_{4x4}+24\lambda$), and choose the one with smaller cost.

In the search hardware, there are two parts operating in parallel in order to complete the mode decision process faster. The upper part shown in Figure 4.3 is used for finding the cost of 16x16 luma and 8x8 chroma modes. The size of register files used in this part is 384 x 8, because they are used for storing both luma and chroma samples of a MB. The lower part shown in the same figure is used for finding the cost of all 4x4 modes for each 4x4 luma block of a MB. The size of register files used in this part is 256 x 8, because they are used for storing only luma samples of a MB. Mode decision module uses the results produced by these two parts to determine the mode with lowest cost and sends this mode information to the coder hardware.

Before the intra prediction module is used to predict the pixels of a MB, the neighboring buffers in the intra prediction module are filled with original image data available in the current MB register file. The top-level control of the search hardware then sends the block number and type information to the intra prediction module and starts it for each available mode.

The upper part of the search hardware starts working by computing the cost of the 16x16 DC mode for luma samples. When intra prediction completes the prediction based on 16x16 DC mode, the result of this prediction is used by residue and Hadamard transform modules. Intra 16x16 mode decision requires storing DC coefficients in a register, because Hadamard transform has to be applied to these coefficients again. The multiplexer before the Hadamard transform hardware selects between DC coefficients and coefficients from the residue block. As the residue and Hadamard transform hardware are working on the predicted MB based on 16x16 DC mode, the top-level control of the search hardware starts the intra prediction module for the 8x8 DC mode for chroma samples. When the intra prediction completes the prediction based on 8x8 DC mode for chroma samples, the result of this prediction is used by residue and Hadamard transform modules and the top-level control of the search hardware starts the intra prediction module for the next available 16x16 mode for luma samples. This process continues for all the available modes for 16x16 luma and 8x8 chroma samples. In this way, computing the costs of 16x16 and 8x8 modes of a MB are overlapped.

4.1.1 High Speed Hadamard Transform Hardware

In order to complete the Sum of Absolute Transformed Difference (SATD) operations faster, a high speed Hadamard transform hardware is designed. The designed hardware is shown in Figure 4.4. This hardware implements the same algorithm shown in Figure 3.7. The difference is that 15 adder / subtractor units are used in this hardware as opposed to 3 adder /subtractor units used in the hardware shown in Figure 3.4. The adder / subtractors used in this hardware are 13-bit adder / subtractors. The hardware includes two pipelining registers to improve the maximum clock frequency. A register is used in the hardware to store the result of accumulating the transformed difference data. The proposed hardware finishes SATD operations of a 4x4 block in 18 clock cycles.

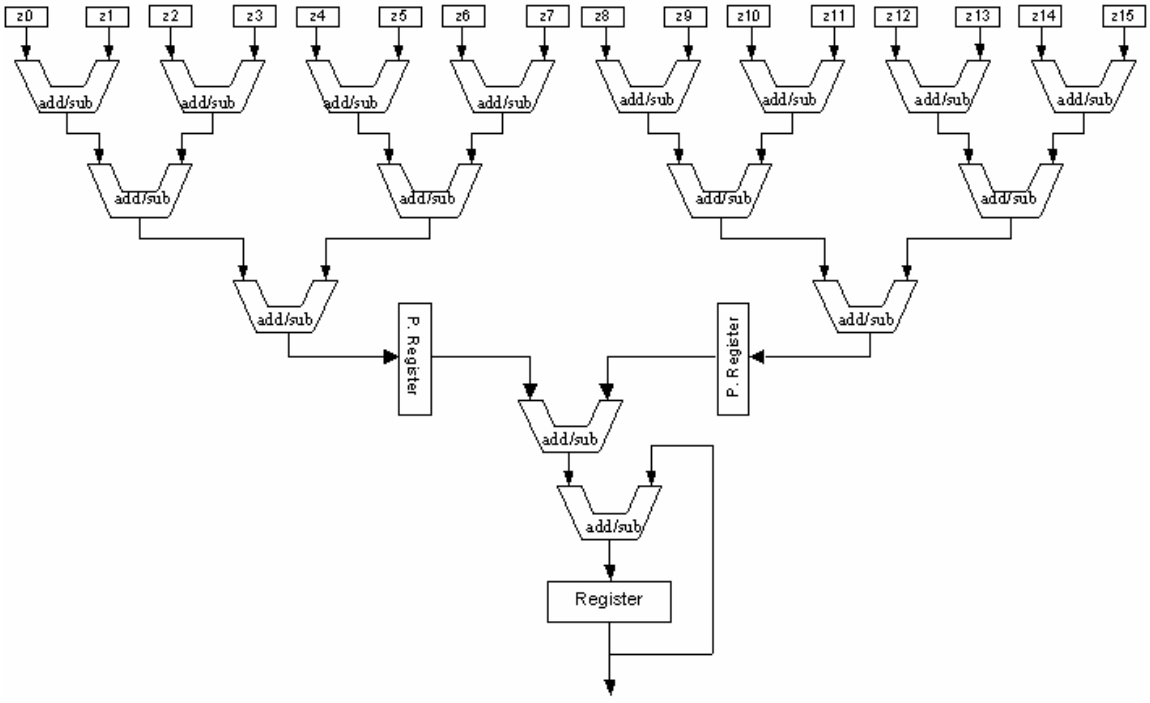

Figure 4.4 Block Diagram of High Speed Hadamard Transform Hardware

4.1.2 Mode Decision Hardware

The hardware shown in Figure 4.5 is designed for comparing the cost of selected intra 16x16 mode with the total cost of selected 4x4 modes. In order to avoid using a multiplier, λ is shifted to left by 3 bits and λ is added to this shifted value three times to obtain 24xλ. Then, the total cost of selected 4x4 modes is added to 24xλ and the result of this addition is subtracted from the cost of selected 16x16 mode. If the result is positive, mode decision hardware selects 16x16 mode, otherwise it selects 4x4 modes.

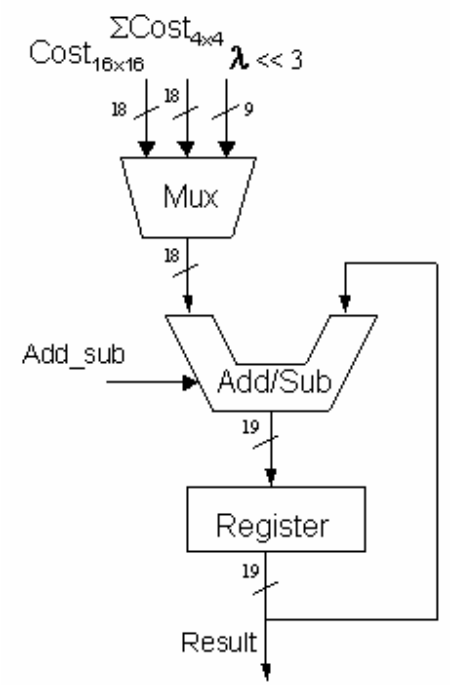

Figure 4.5 Block Diagram of Mode Decision Hardware

Table 4.2 Mode Decision Hardware Register Content

Cycle Number	Register Content
1	8 x λ
2	16 x λ
3	24 x λ
4	$\Sigma Cost_{4x4}$ + 24 x λ
5	$Cost_{16x16} - (\Sigma Cost_{4x4} + 24 \times \lambda)$

4.2 Coder Hardware

The block diagram of the Coder Hardware is shown in Figure 4.6. The transform, quant, inverse transform, inverse quant, Hadamard transform, inverse Hadamard transform, CAVLC and intra prediction modules in the coder hardware are explained in chapter 3. Residue block creates the residual data by taking the difference between current macroblock and intra predicted macroblock, and it loads the residual data to the input register file of the Transform – Quant Hardware. Reconstruction block reads the result of inverse quantization and inverse transform Hardware and the intra predicted macroblock as inputs. It adds them and clips the result to the [0-255] range, and loads the result to the neighboring pixel buffers in the intra prediction hardware and to the reconstructed macroblock register file.

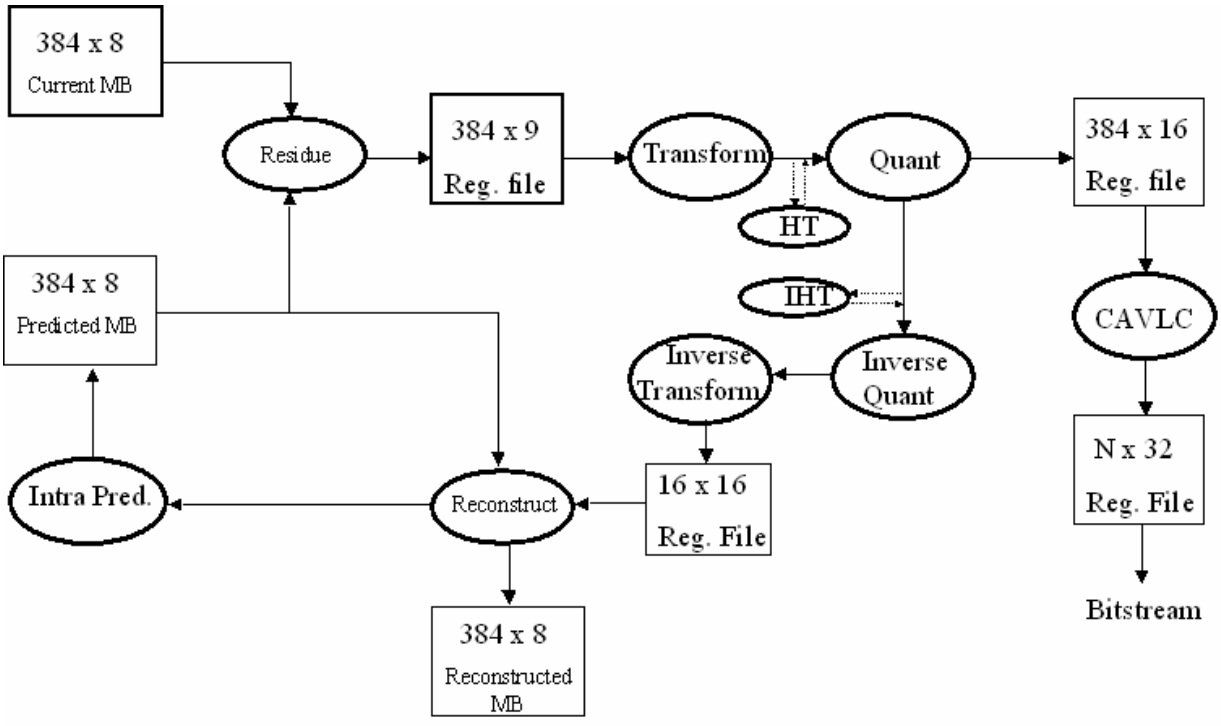

Figure 4.6 Block Diagram of Coder Hardware

Since there are 256 luminance pixels and 128 chrominance pixels in a MB, and each pixel value is in the range [0-255], 384x8 register files are used for storing the current MB, the intra predicted MB and the reconstructed MB. A 384x9 register file is used for storing the 9-bit signed residual data. A 384x16 register file is used for storing the 16-bit signed quantized transform coefficients of a MB. A 16x16 register file is used for storing the 16-bit signed inverse transformed coefficients of a 4x4 block. A 192x32 register file is used for storing the generated bitstream.

The scheduling of the Coder Hardware for a MB that will be coded with 4x4 intra modes is shown in Figure 4.7. For a MB that will be coded with 4x4 intra modes, first the intra predictor hardware fills the predicted macroblock register file with the predicted pixels based on the 4x4 prediction modes. Then, the residue block subtracts the predicted MB from the current MB. When the first 4x4 block of residual data is available, Transform – Quant module starts to work. The results of Transform - Quant module, i.e. quantized transform coefficients, are loaded to the input register file of CAVLC hardware. As soon as the quantized transform coefficients of the first 4x4 block are ready, CAVLC and Transform – Quant modules start to work. The output of CAVLC module, i.e. the generated bit stream, is stored in the output register file of CAVLC hardware. As soon as Transform – Quant module finishes inverse quant and inverse transform operations for the first 4x4 block, i.e. when the inverse transformed coefficients of first 4x4 block are ready, reconstruction block starts to work. After the first 4x4 block of a MB is coded and reconstructed, the coder hardware starts to work on the second 4x4 block. In this way, all 4x4 blocks in a MB are coded and reconstructed.

The scheduling of the Coder Hardware for a MB that will be coded with a 16x16 intra mode is shown in Figure 4.8. For a MB that will be coded with a 16x16 intra mode, Hadamard Transform has to be applied to DC coefficients after 4x4 integer transforms. Therefore, inverse quant and inverse transform, CAVLC and reconstruction operations for the MB can only start after the Hadamard transform finishes.

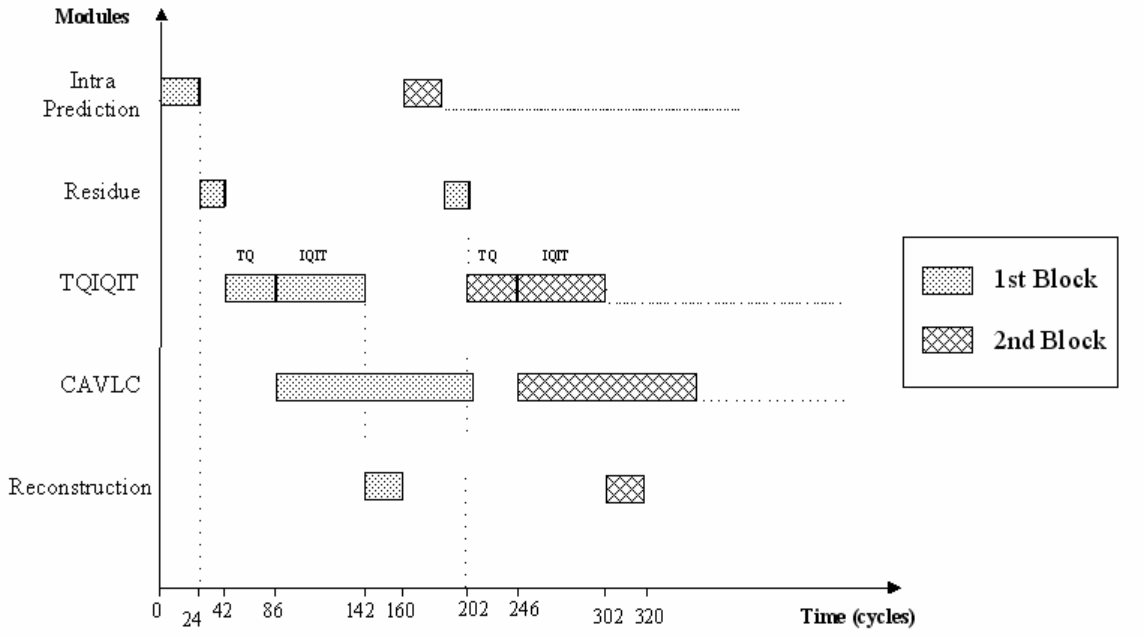

Figure 4.7 Coder Hardware Scheduling for 4x4 Intra Modes

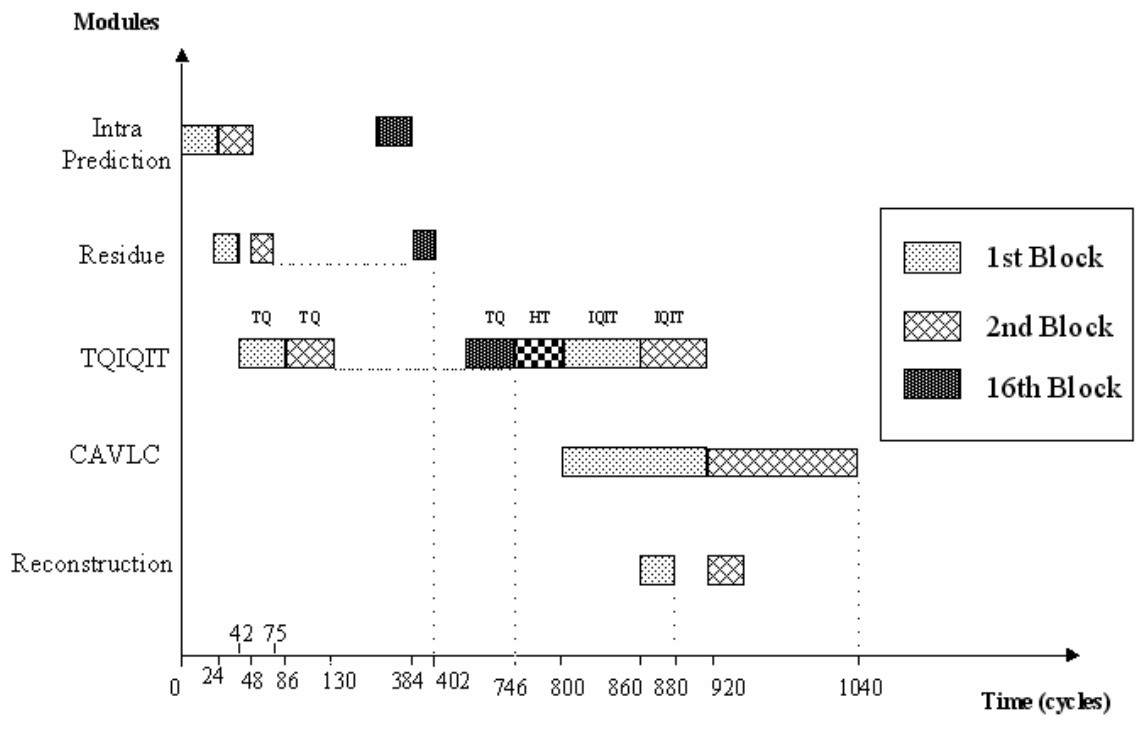

Figure 4.8 Coder Hardware Scheduling for 16x16 Intra Modes

CHAPTER 5

H.264 INTRA FRAME CODER SYSTEM

5.1 System Overview

The development environment consists of a PC connected to ARM Versatile/PB926EJ-S board through ARM Multi-ICE debugger, a logic tile mounted on the Versatile/PB926EJ-S board and a color LCD panel. PC is used to create the bit stream, which is loaded to the FPGA on the logic tile. Design methodology to create the resulting bit stream is explained in section 5.3. The software running on this PC to debug the system is AXD Debugger from ARM Developer Suite 1.2 and ARM Multi ICE Server V.2.2.6 is used to communicate with the development board. A Color LCD panel is used to display the original and reconstructed images for visual verification. The development environment is shown in Figure 5.1 [19].

The Versatile/PB926EJ-S board contains a development chip including an ARM 9 processor, a bus matrix and a number of peripheral interfaces. There are volatile and non-volatile memories on the board. The board also has a two million gate Xilinx Virtex II FPGA, XC2V2000, which implements more peripheral interfaces. Figure 5.2 shows the peripherals of Versatile/PB926EJ-S development board [19]. The controllers for some of these interfaces are implemented in the development chip such as color LCD and GPIO controllers. The controllers for the other interfaces are implemented in the XC2V2000 Xilinx Virtex II FPGA such as PCI interface, smart card interface, multimedia card interface, keyboard interface, mouse interface, universal asynchronous receiver-transmitter (UART) interface and character LCD interface. In addition, XC2V2000 FPGA controls

registers for status, ID, onboard switches, LEDs, and clock control [19]. A bit file implementing these controllers is loaded to the XC2V2000 FPGA at the start up by default.

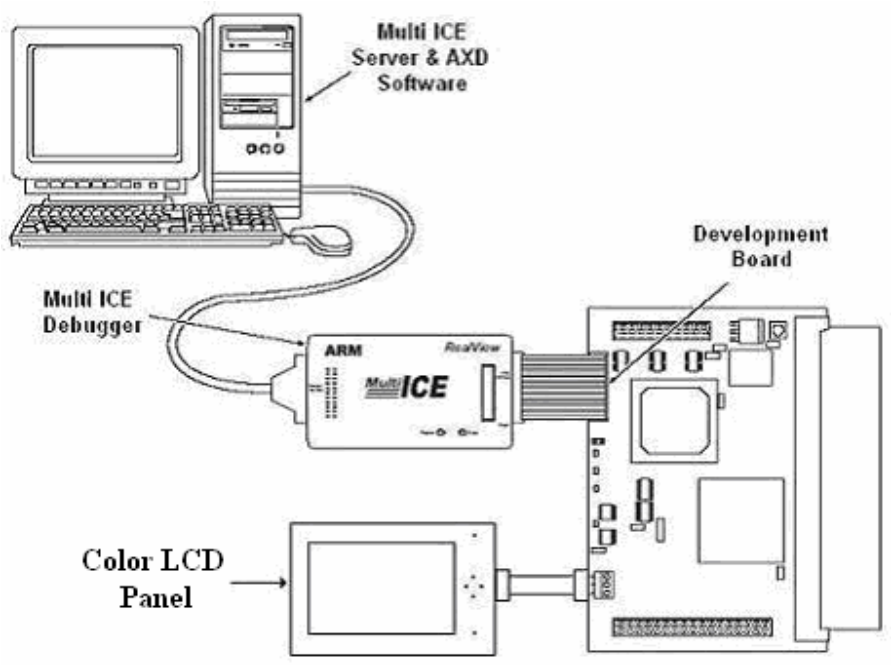

Figure 5.1 Development Environment

The board has a connector for Joint Test Access Group (JTAG) port which is used for configuring the FPGAs and PLDs on the board and for debugging the system. The board also has logic analyzer connectors for AHB monitor and Trace port. The signal activity of the system buses can be seen with a logic analyzer connected to the AHB monitor. The contents of the internal registers can be seen with a logic analyzer connected to the Trace port without stopping the system.

Versatile board offers the possibility of using one or more Real View logic tiles which can be configured to implement custom-designed logic. HDRX, HDRY and HDRZ, shown as logic tile expansion in Figure 5.2, form together the logic tile connection. Logic tiles can be added to the system by mounting them on top of these connectors. The logic tiles include Xilinx Virtex II FPGAs to implement additional hardware in the system [19].

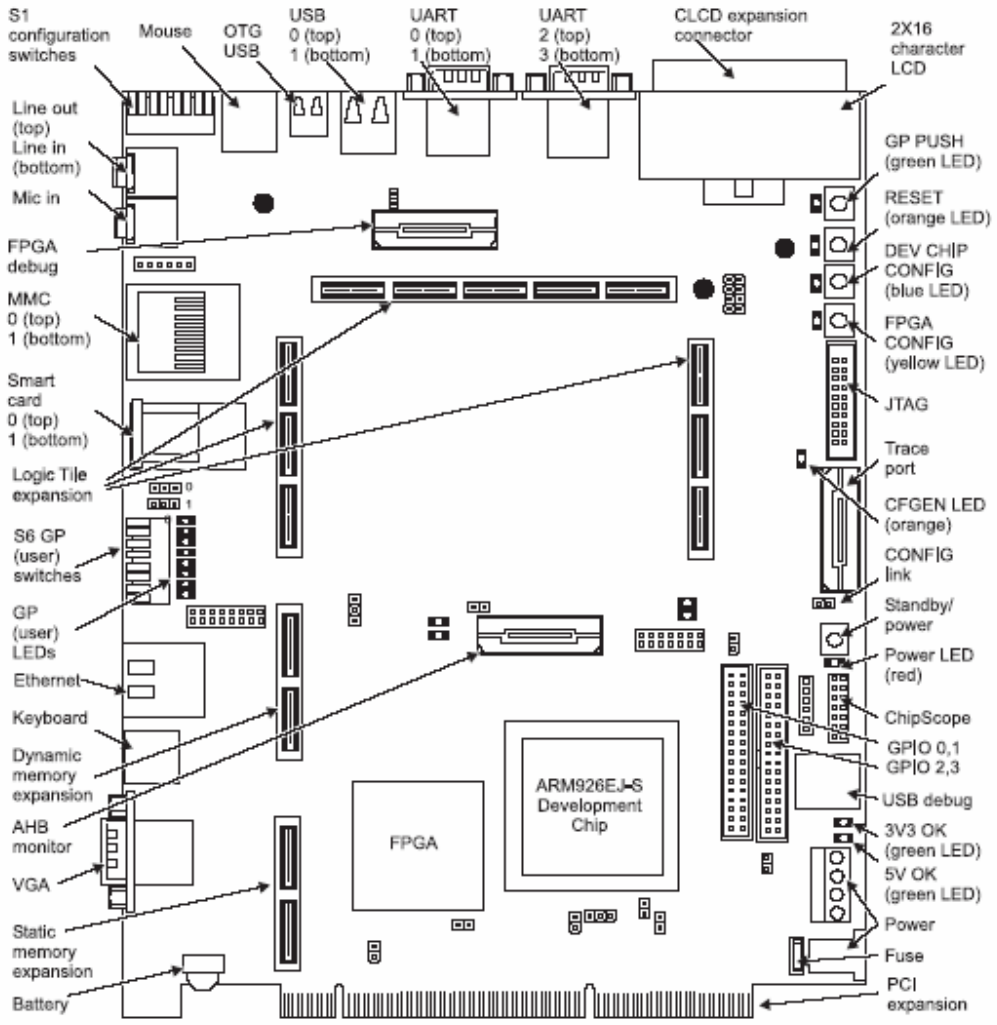

Figure 5.2 Versatile/PB926EJ-S Development Board

There are also programmable clock generators on the Versatile/PB926EJ-S board and the board contains clock multiplexers, so that different clock signals can be used. The clock multiplexers can be configured so that the clocks are driven either by the baseboard or the logic tile [20]. There are three major clock signals, which drive the ARM processor core, modules connected to the buses in the development chip and modules connected to the buses outside the development chip. Default frequencies for these clocks are 210 MHz for ARM CPU, 70 MHz for internal buses and 35 MHz for external buses and the maximum working frequencies for these clocks are 216 MHz, 75 MHz and 43 MHz, respectively. System buses can be clocked with the same clock signal or with different clock signals.

Synchronous mode means using the same clock signal for the buses. Asynchronous mode means using different clock signals for the buses. This mode is ideal for implementing modules on these buses which have different maximum working clock frequencies [19].

5.1.1 Development Chip

ARM926EJ-S Development Chip, the block diagram of which is shown in Figure 5.3, is equipped with ARM926EJ-S processor that supports 32-bit ARM and 16-bit Thumb instructions sets, tightly coupled memories for code (32KB) and data (32KB), cache memory for code (32KB) and data (32KB), memory management unit, multi-layer bus matrix that provides simultaneous transfers, MOVE video encoding coprocessor, MBX graphics accelerator, Multi-Port Memory Controller (MPMC) for accessing dynamic memory, Synchronous Static Memory Controller (SSMC) for accessing static (SRAM or flash) memory, VFP9 Vector Floating Point coprocessor, two external AHB master bridges and one external AHB slave bridge, AHB monitor for detailed analysis of bus activity, DMA controller, Vectored Interrupt Controller (VIC), Color LCD controller, three UARTs, Synchronous Serial Port (SSP), Smart Card Interface (SCI), four eight-bit GPIOs, Real Time Clock (RTC), two programmable timers, watchdog timer, Embedded Trace Macrocell (ETM9), Embedded-ICE logic for JTAG debugging, Phase-Locked Loop (PLL) [19].

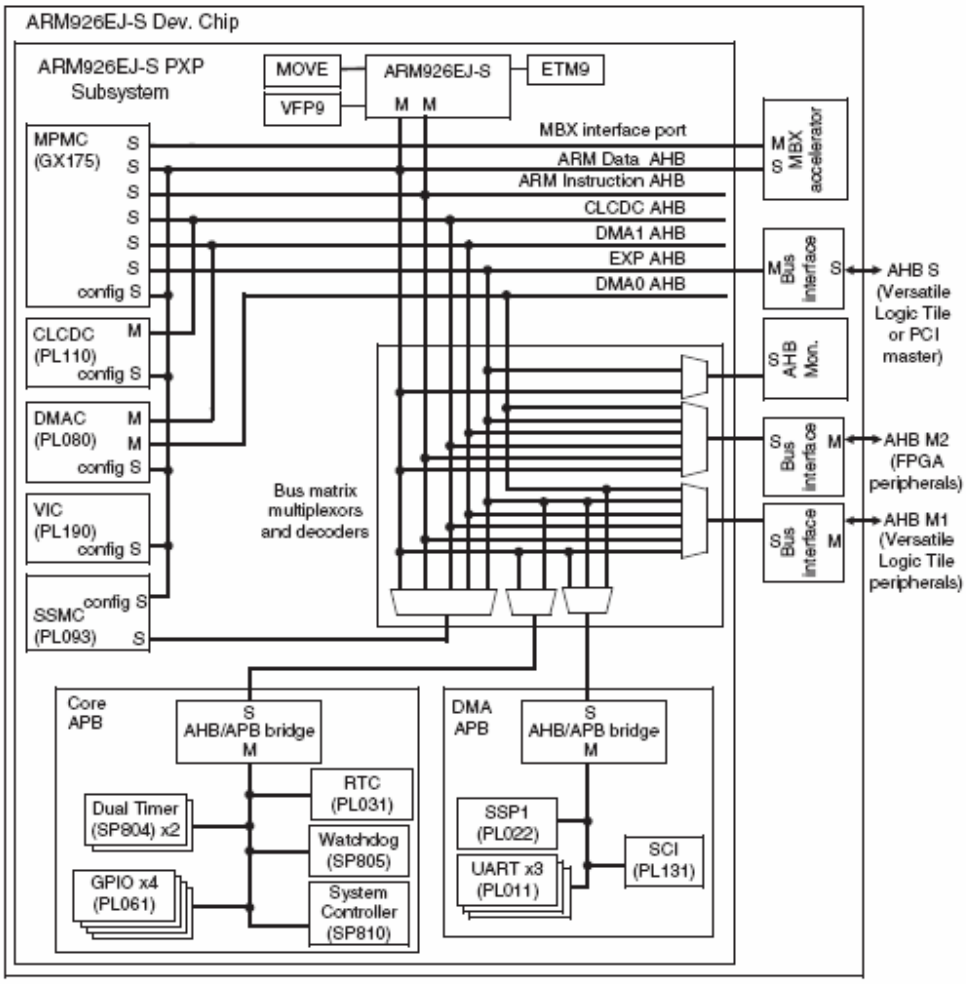

Figure 5.3 Block Diagram of Development Chip

5.1.2 Memory

Versatile board contains 128 MB of 32-bit wide SDRAM, 2 MB of 32-bit wide static RAM, 64 MB of 32-bit wide NOR flash, 64 MB of 16-bit wide NAND flash memories. There are memory expansion ports for static and dynamic expansion memories. Using these ports, if required, up to 320 MB of static memory in a static memory expansion board and up to 256 MB of SDRAM in a dynamic memory expansion board can be added to system memory. Figure 5.4 shows system memories with their connections and mapping [19].

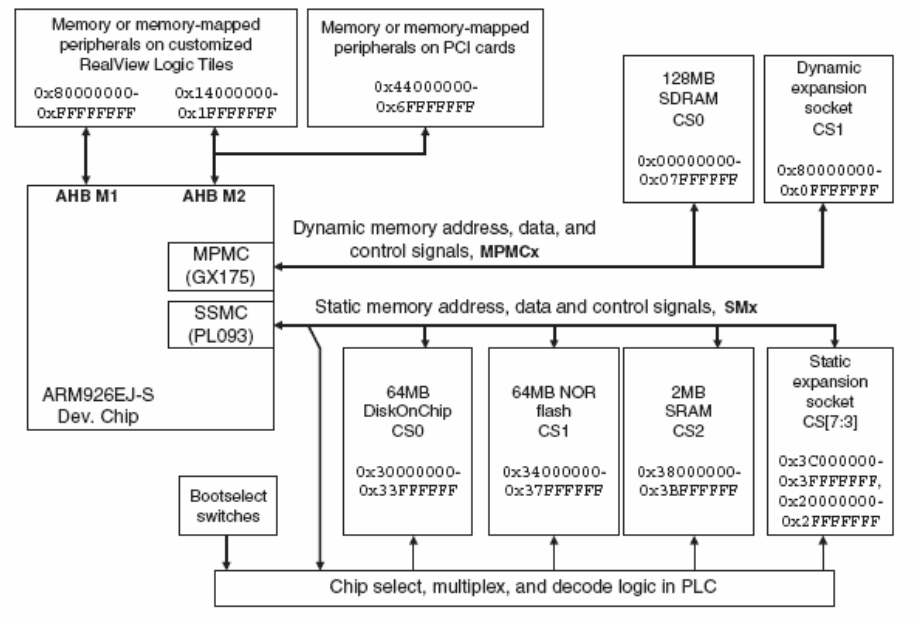

Figure 5.4 System Memory Map

5.1.3 Bus Architecture

As shown in Figure 5.5, the development chip on the Versatile/PB926EJ-S interfaces external peripherals using three external buses; AHB M1, AHB M2 and AHB S [19, 20, 22] AHB M1 is an external master bus, so the development chip behaves as a bus master. The AHB M1 is connected to the baseboard FPGA and logic tiles, but the default baseboard FPGA configuration does not connect any slaves to this bus. AHB M2 is an external master bus, so the development chip behaves as a bus master. The AHB M2 is connected to the baseboard FPGA and logic tiles. The default baseboard FPGA configuration has its slaves connected to this bus. AHB S is an external slave bus, so the development chip behaves as a slave on this bus. The AHB S bus is connected to the baseboard FPGA and logic tiles. The default baseboard FPGA configuration has the PCI master connected to this bus. An internal bus matrix is used in the development chip. The AHB M1 and AHB M2 bridges inside the development chip are slaves of the bus matrix,

mapped to fixed addresses in the memory map. The AHB S bridge is a master of the bus matrix, and has its own bus layer.

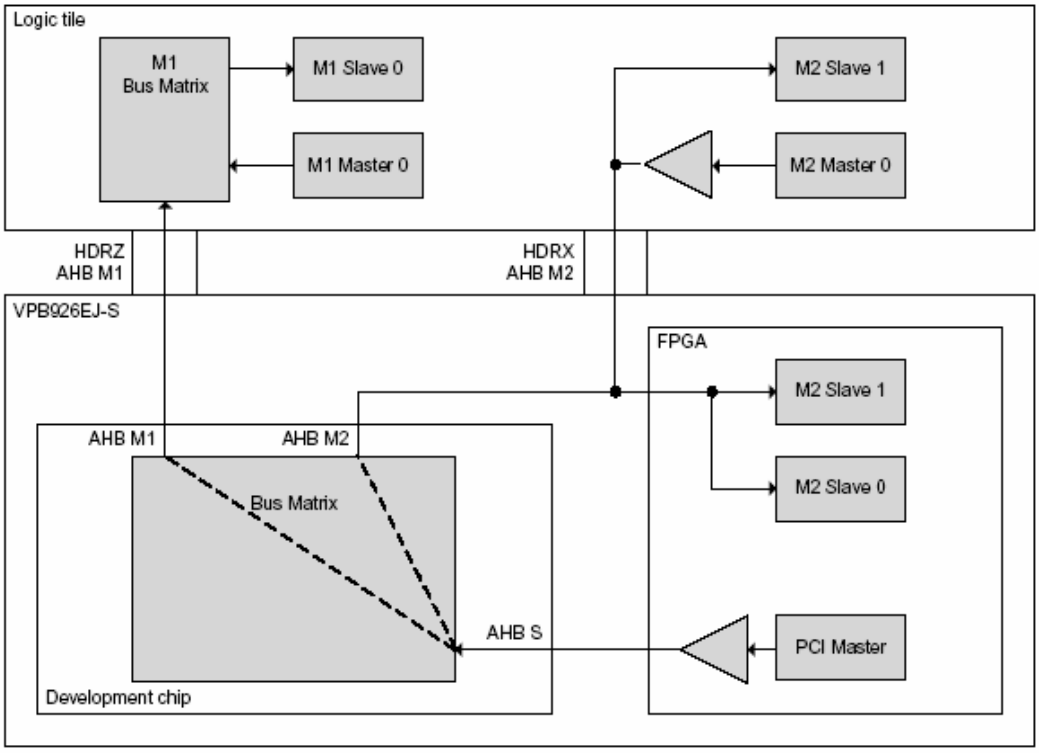

Figure 5.5 Bus Architecture

5.1.4 Logic Tile

As shown in Figure 5.6, an eight million gate Xilinx Virtex II FPGA, XC2V8000, exists on the logic tile [21]. Other than the FPGA, the logic tile includes two 1MB SRAM memories, LEDs, switches and a pushbutton.

The XC2V8000 Xilinx Virtex II FPGA uses a flip chip technology based 40x40 mm package. The package has 1108 user I/Os. In the FPGA, there are 168 18-bit x 18-bit embedded multipliers, 168 block RAMs, providing a total of 3024Kbits RAM, 1456Kbits of distributed RAM and 12 digital clock management blocks based on DLL technology.

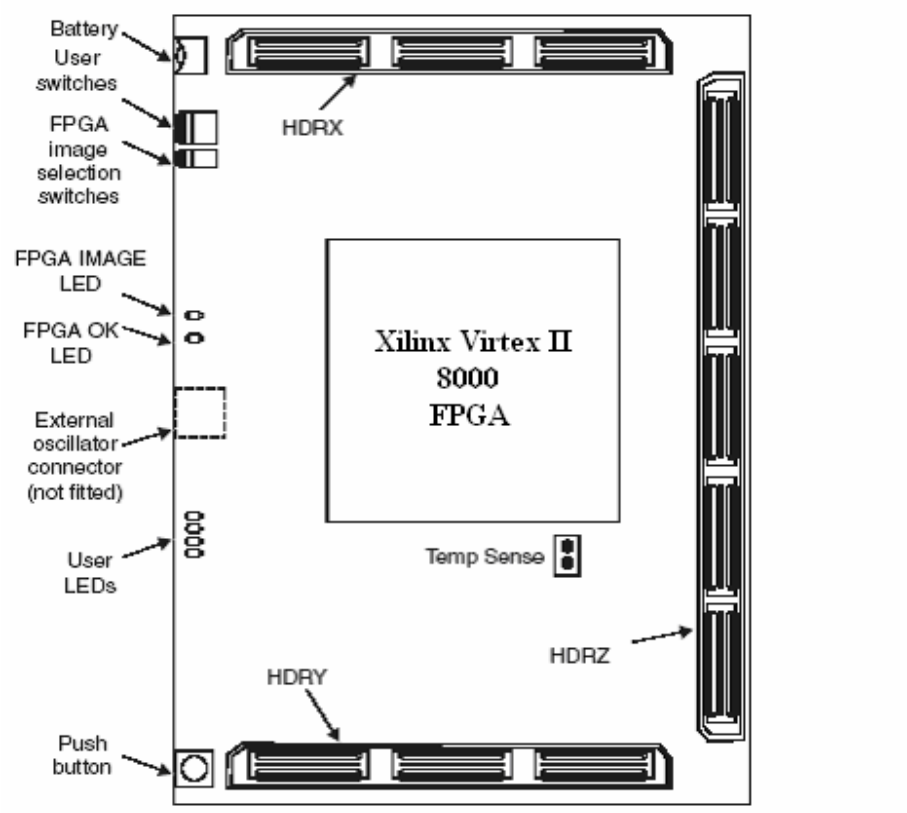

Figure 5.6 Logic Tile

The embedded multipliers take two 18-bit 2's complement inputs and produce a 36-bit result. Block RAMs are a dual-port RAMs with two independently clocked and controlled synchronous read/write ports that access a common storage area. The size of each block RAM is 18Kbits with configurable port widths from 1 bit to 36 bits. Distributed RAMs use look-up tables as memory and they have two ports; one for read operations and one for read and write operations. Distributed RAM is an efficient way of building small, fast, localized memory structures within FPGA.

5.2 Software Implementation

Software implementation includes Matlab and C codes. The flow of the operations done by software is shown in Figure 5.7. First, the input image is captured with Matlab in RGB format and the captured image is loaded to the SRAM on the logic tile using ARM AXD debugger software. This image is then converted from RGB format to YCbCr format, partitioned into MBs, and converted to 4:2:0 sampling format by the software running on ARM processor. The equations used for RGB to YcbCr and YCbCr to RGB color domain conversions are shown in Figure 5.8. At this point, designed hardware, implemented in the Xilinx Virtex II FPGA on the logic tile, starts to process the image MB by MB. Throughout this operation, processed MBs are read from reconstructed MB register file and stored in the SRAM. When coding of the image finishes, the output image is reconstructed in raster-scan order from processed MBs, the resulting image is converted from YCbCr format to RGB format and loaded to a location in the SRAM by the software running on ARM processor. The color LCD controller points to this location, and it reads the image from this location and displays it on the color LCD panel.

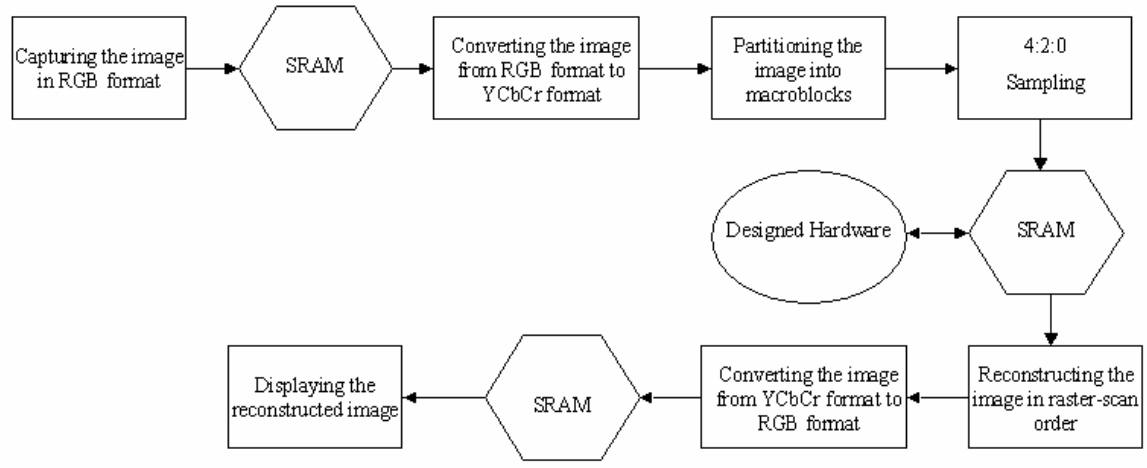

Figure 5.7 Operations Done by Software

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & 0.5 \\ 0.5 & -0.419 & -0.081 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad \begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.402 \\ 1 & -0.344 & -0.714 \\ 1 & 1.772 & 0 \end{bmatrix} \begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix}$$

(a) **(b)**

Figure 5.8 Color Domain Conversions **(a)** RGB to YcbCr, **(b)** YCbCr to RGB

5.3 Hardware Implementation

As shown in Figure 5.9 (a), hardware description languages are commonly used to define the behavior of an FPGA. The design is then compiled to generate a netlist and this netlist is then mapped to the actual FPGA. In this thesis, as shown in Figure 5.9 (b), Verilog HDL, Mentor Graphics Leonardo Spectrum and Xilinx ISE tools are used.

Intra Frame Coder Hardware modules are implemented in Verilog HDL. In order to achieve the targeted speed with low area usage, resource allocation, pipelining and critical path optimization techniques are used. Power consumption is also tried to be reduced during the design process. After completing the Verilog modules, they are compiled with Leonardo Spectrum Level 3 from Mentor Graphics with high effort for speed constraint. In this thesis, optimization for speed is preferred rather than optimization for area. The netlist is then placed and routed using Xilinx ISE Series 5.2i with high effort for speed constraint. The Xilinx ISE tool is also used for generating the bit file from this netlist.

FPGA design is an iterative process. If the synthesis and place & route tools cannot meet the user constraints, than the Verilog RTL code may need to be modified to meet these constraints. In our design, when the tools could not meet the target clock frequency, we have used the critical path optimization and pipelining techniques to speed up the design.

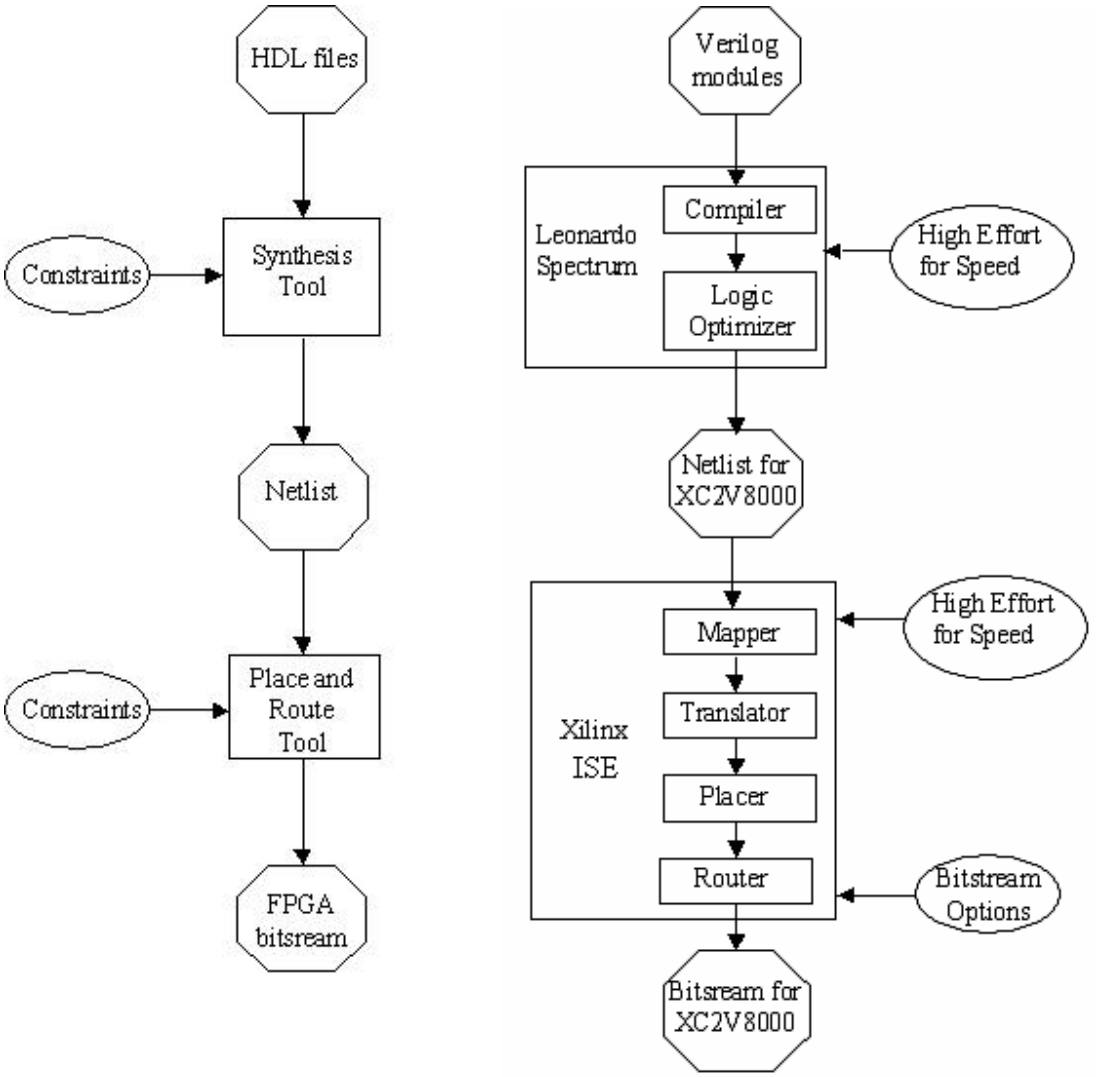

Figure 5.9 FPGA Design Flow: **(a)** Generic, **(b)** Specific

The H.264 Intra Frame Coder Hardware is synthesized at 61.4 MHz. The FPGA resource utilization of the synthesized design is given in Table 5.1. Total equivalent gate count of the design is 1,051,458 and an additional number of 20,211 JTAG gates are added to the design due to I/O blocks. The design is placed & routed at 53.8 MHz which satisfies our target speed.

Table 5.1 Device Utilization for XC2V8000 FPGA

Resources	Used	Available	Utilization
Ios	418	1108	37.73%
Global Buffers	2	16	12.50%
Function Generators	21404	93184	22.97%
CLB Slices	10702	46592	22.97%
Dffs/Latches	3881	96508	4.02%
Block RAMs	1	168	0.60%
Block Multipliers	1	168	0.60%

As shown in Figure 5.10, H.264 Intra Frame Coder Hardware is integrated into the Xilinx Virtex II FPGA on the logic tile of the ARM Versatile/PB926EJ-S board as a master of the AHB S bus. We used synchronous bus clocking, because our design utilizes only one AHB bus. The bit file including the H.264 Intra Frame Coder Hardware is loaded to the Xilinx Virtex II FPGA using the Progcards utility which in turn uses Multi-ICE Server Software and Multi-ICE Debugger Hardware.

A picture of the development environment is given in Figure 5.11. H.264 Intra Frame Coder Hardware, implemented in the Xilinx Virtex II FPGA, is started by pressing the pushbutton on the logic tile after the system reset. The switches on the logic tile are used to input the quant parameter to the hardware. Intra Frame Coder Hardware reads the input image from the SRAM memory using AHB bus protocol, performs compression and reconstruction, and writes the reconstructed image to the SRAM memory again using the AHB bus protocol [22]. The reconstructed image is then displayed on the color LCD panel for visual verification. H.264 Intra Frame Coder System is tested with different quant parameters and the result is as expected; the quality of the image displayed on the color LCD panel reduces with the increasing quant parameter value.

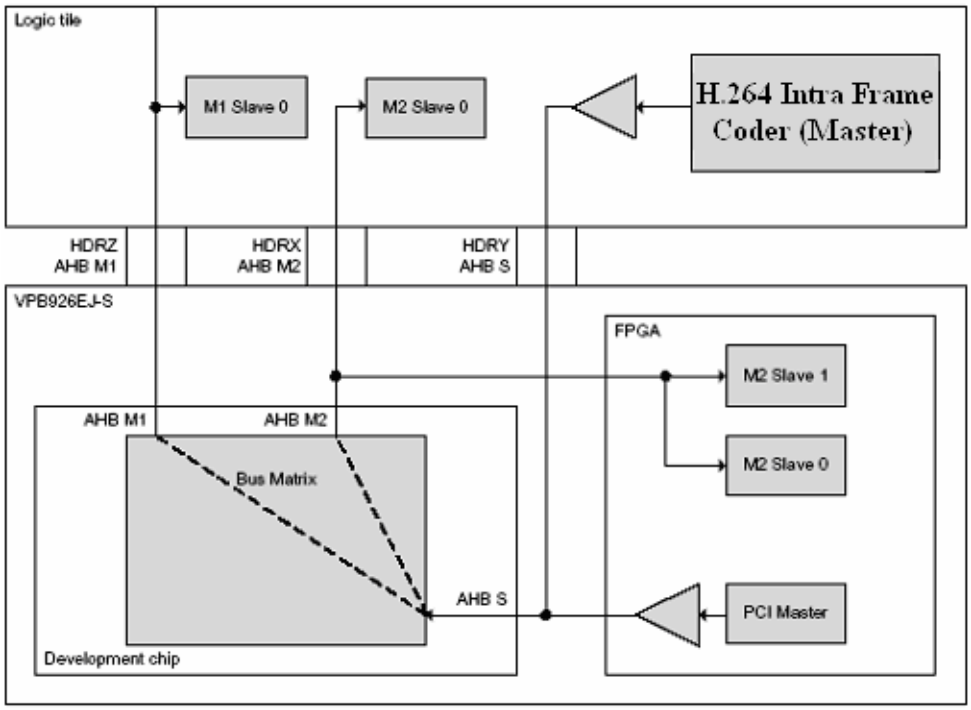

Figure 5.10 Integration of Designed Hardware into ARM Versatile Board

Figure 5.11 Picture of the Development Environment

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

In this thesis, we developed an FPGA-based H.264 intra frame coder system for portable applications targeting level 2.0 of baseline profile. As part of the system, we first designed a high performance and low cost hardware architecture for real-time implementation of forward transform and quantization and inverse transform and quantization algorithms used in H.264 / MPEG4 Part 10 video coding standard in Verilog HDL. The architecture is based on a reconfigurable datapath with only one multiplier. The design is first verified with RTL simulations using Mentor Graphics Modelsim. It is then verified to work at 81 MHz on a Xilinx Virtex II FPGA and at 210 MHz in a 0.18µ ASIC implementation. The FPGA and ASIC implementations can code 28 and 74 VGA frames (640x480) per second respectively.

We then designed the top-level H.264 Intra Frame Coder System targeting 30 fps CIF encoding. The system consists of search, mode decision and coding parts. The mode decision part implements a Hadamard Transform based mode decision algorithm. The coding part is implemented by integrating Transform-Quant module with CAVLC and Intra Prediction modules. The top-level design is verified with RTL simulations using Mentor Graphics Modelsim. The top-level design is synthesized for 61.4MHz and placed & routed for 53.8MHz. The total equivalent gate count of the design is 1,051,458. Device utilization for XC2V8000 FPGA is as follows; 37.73% of IOs, 12.50% of global buffers, 22.97% of function generators, 22.97% of CLB slices, 4.02% Dffs or latches, 0.60% of block RAMs and 0.60% of block multipliers are used.

The complete H.264 Intra Frame Coder System is verified to work on an ARM Versatile Platform development board. The verification includes first capturing an RGB image, converting it into YCbCr format, partitioning the image into macroblocks, and writing it into an SRAM using the software running on ARM9EJ-S processor. Then, the intra frame coder hardware mapped to the Xilinx Virtex II FPGA using Lenoardo Spectrum and Xilinx ISE is used to encode the image and reconstruct it. The conversion of reconstructed image into raster scan order and RGB color domain is then performed by software running on ARM9EJ-S processor. The reconstructed image is then displayed on a color LCD panel for visual verification.

6.2 Future Work

In order to guarantee that our design exactly matches the H.264 standard, further verification must be done. The best method for this is decoding the bit stream generated by the intra frame coder using an H.264 compliant decoder, e.g. Joint Model (JM) Software. In order to do this, header generation functionality should be added to the intra frame coder system. Low-power techniques such as clock gating can be used in the design to lower the power consumption of the system. The FPGA-based implementation can be easily modified as an ASIC implementation and prototypes can be fabricated. A camera can be integrated to the current system for real-time video capture and coding. The intra frame coder system can be extended to a complete H.264 video coding system by integrating motion estimation, motion compensation, deblocking filter, intra vs. inter mode decision and rate control units.

REFERENCES

[1] R. Schäfer, T. Wiegand and H. Schwarz, "The Emerging H.264/AVC Standard", *EBU Technical Review*, January 2003

[2] C. Chung, "Implementing the H.264/AVC Video Coding Standard on FPGAs", *Xcell Journal*, Winter 2004

[3] Y. W. Huang, B. Y. Hsieh, T. C. Chen *and L. G. Chen "*Hardware Architecture Design for H.264/AVC Intra Frame Coder*", Proc. of IEEE ISCAS*, 2004

[4] Brian Dipert, "FPGAs "DiSP"lay Their Processing Prowess", *EDN Articles*, October 2002

[5] A. Jerbi, H.264 MPEG-4 AVC Video Coding Standard, ubvideo, November 2003

[6] Y. Zhao, Complexity Management for Video Encoders, PhD Thesis, Robert Gordon University, March 2004

[7] http://www.envivio.cn/images/products/h264FactSheet.pdf

[8] T. Wiegand, G. J. Sullivan, G. Bjøntegaard, and A. Luthra "Overview of the H.264/AVC Video Coding Standard", *IEEE Trans. on Circuits and Systems for Video Technology vol*. 13, no. 7, pp. 560–576, July 2003

[9] I. Richardson, H.264 and MPEG-4 Video Compression*,* Wiley, 2003

[10] Joint Video Team (JVT) of ITU-T VCEG and ISO/IEC MPEG, Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification, ITU-T Rec. H.264 and ISO/IEC 14496-10 AVC, May 2003

[11] O.Tasdizen, I. Hamzaoglu, "A High Performance And Low Cost Hardware Architecture for H.264 Transform And Quantization Algorithms", *13th European Signal Processing Conference*, September 2005

[12] Joint Video Team (JVT) of ITU-T VCEG and ISO/IEC MPEG, *Joint Model (JM) Reference Software Version 9.2*, http://bs.hhi.de/suehring/

[13] H. Malvar, A. Hallapuro, M. Karczewicz. and L. Kerofsky, "Low-Complexity Transform and Quantization in H.264 / AVC", *IEEE Trans. on Circuits and Systems for Video Technology, vol. 13, no. 7, pp. 598–603*, July 2003.

[14] Xilinx Inc., *Virtex-II™ Platform FPGAs: Complete Data Sheet DS031*, http://www.xilinx.com, March 2004

[15] T. C. Wang, Y. W. Huang, H. C. Fang, and L. G. Chen*, "*Parallel 4x4 2D Transform and Inverse Transform Architecture for MPEG-4 AVC / H.264*", Proc. of IEEE ISCAS*, 2003

[16] E. Sahin, I. Hamzaoglu, "A High Performance and Low Power Hardware Architecture for H.264 CAVLC Algorithm", *13th European Signal Processing Conference*, September 2005

[17] Personal Communication with Esra Sahin

[18] Personal Communication with Hasan Ates

[19] Versatile Platform Baseboard for ARM926EJ-S User Guide, http://www.arm.com, May 2004

[20] Application Note 119: Implementing AHB Peripherals in Logic Tiles, http://www.arm.com, September 2004

[21] Versatile / LT-XC2V4000+ User Guide, http://www.arm.com, May 2004

[22] AMBA 2.0 Specification, http://www.arm.com, 1999