2005

Novel VLSI Architecture for Quantization and Variable Length Coding for H-264/AVC Video Compression Standard

Suneetha Kosaraju

Follow this and additional works at: http://scholarworks.rit.edu/theses

Recommended Citation

This Thesis is brought to you for free and open access by the Thesis/Dissertation Collections at RIT Scholar Works. It has been accepted for inclusion in Theses by an authorized administrator of RIT Scholar Works. For more information, please contact ritscholarworks@rit.edu.
Novel VLSI Architecture for Quantization and Variable Length Coding for H-264/AVC Video Compression Standard

By

Suneetha Kosaraju

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of

Master of Science

In

Electrical Engineering

Approved By:

Kenneth W. Hsu

Dr. Kenneth W. Hsu
Primary Advisor – R.I.T. Dept. of Computer Engineering

Pratapa Reddy

Dr. Pratapa Reddy
Secondary Advisor – R.I.T. Dept. of Computer Engineering

Edward Brown

Dr. Edward Brown
Secondary Advisor – R.I.T. Dept. of Electrical Engineering

Robert J. Bowman

Dr. Robert J. Bowman
Department Head – R.I.T. Dept. of Electrical Engineering

Department of Electrical Engineering
Kate Gleason College of Engineering
Rochester Institute of Technology
Rochester, NY
December 2005
Thesis Release Permission Form

Rochester Institute of Technology
Kate Gleason College of Engineering

Title: Novel VLSI Architecture for Quantization and Variable Length Coding for H-264/AVC Video Compression Standard

I, Suneetha Kosaraju, hereby grant permission to the Wallace Memorial Library to reproduce my thesis in whole or part.

Suneetha Kosaraju

Date
I would like to thank all of my advisors Dr. Kenneth Hsu, Dr. Pratapa Reddy, and Dr. Edward Brown for giving of their time and knowledge; and especially Dr. Kenneth Hsu for his valuable guidance at each and every step towards the successful completion of the thesis. I would also like to thank my family and friends for their support and encouragement in every walk of life.
Abstract

Integrated multimedia systems process text, graphics, and other discrete media such as digital audio and video streams. In an uncompressed state, graphics, audio and video data, especially moving pictures, require large transmission and storage capacities which can be very expensive. Hence video compression has become a key component of any multimedia system or application. The ITU (International Telecommunications Union) and MPEG (Moving Picture Experts Group) have combined efforts to put together the next generation of video compression standard, the H.264/MPEG-4 Part10/AVC, which was finalized in 2003. The H.264/AVC uses significantly improved and computationally intensive compression techniques to maximize performance. H.264/AVC compliant encoders achieve the same reproduction quality as encoders that are compliant with the previous standards while requiring 60% or less of the bit rate [2].

This thesis aims at designing two basic blocks of an ASIC capable of performing the H.264 video compression. These two blocks, the Quantizer, and Entropy Encoder implement the Baseline Profile of the H.264/AVC standard. The architecture is implemented in Register Transfer Level HDL and synthesized with Synopsys Design Compiler using TSMC 0.25μm technology, giving us an estimate of the hardware requirements in real-time implementation. The quantizer block is capable of running at 309MHz and has a total area of 785K gates with a power requirement of 88.59mW. The entropy encoder unit is capable of running at 250 MHz and has a total area of 49K gates with a power requirement of 2.68mW. The high speed that is achieved in this thesis simply indicates that the two blocks Quantizer and Entropy Encoder can be used as IP embedded in the HDTV systems.
# Table of Contents

Acknowledgements ........................................................................................................ iii

Abstract ........................................................................................................................ iv

List of Figures ................................................................................................................ vii

List of Tables ................................................................................................................ ix

List of Equations ............................................................................................................ x

Glossary ........................................................................................................................ xi

1 Introduction .................................................................................................................. 1
   1.1 Video Compression ................................................................................................. 1
   1.1.1 Compression ...................................................................................................... 1
   1.1.2 Video Compression ............................................................................................ 1
   1.1.3 Spatial and Temporal Compression .................................................................... 3
   1.1.4 Sampling .......................................................................................................... 4
   1.1.5 Image and Video Compression ........................................................................... 5
   1.1.6 Video Encoder ................................................................................................... 6
   1.1.7 Video Decoder .................................................................................................. 7
   1.2 Thesis Objective .................................................................................................... 9
   1.3 Thesis Chapter Overview ..................................................................................... 10

2 Literature Review ....................................................................................................... 11
   2.1 Standards of Video Compression ........................................................................... 11
      2.1.1 Standardization groups .................................................................................. 11
      2.1.2 Related standards ......................................................................................... 12
   2.2 H.264/AVC Video Compression ........................................................................... 13
      2.2.1 NAL and VCL .............................................................................................. 14
      2.2.2 Macroblocks and Slices ............................................................................... 15
      2.2.3 H.264/AVC Encoder ..................................................................................... 17
         2.2.3.1 Intra Prediction ....................................................................................... 19
         2.2.3.2 Inter Prediction ...................................................................................... 19
         2.2.3.3 Motion Estimation ............................................................................... 21
         2.2.3.4 Tree Structured Motion Compensation ............................................... 21
         2.2.3.5 Transform and Quantization ................................................................. 23
         2.2.3.6 Reordering ........................................................................................... 25
         2.2.3.7 Entropy Coding .................................................................................... 26
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2.3.8 De-blocking Filter</td>
<td>27</td>
</tr>
<tr>
<td>2.2.4 H.264/AVC Decoder</td>
<td>28</td>
</tr>
<tr>
<td>2.2.5 H.264/AVC Profiles</td>
<td>29</td>
</tr>
<tr>
<td>2.2.6 Performance Comparison</td>
<td>30</td>
</tr>
<tr>
<td>3 Design Procedure and Algorithms</td>
<td>32</td>
</tr>
<tr>
<td>3.1 Quantizer Unit</td>
<td>32</td>
</tr>
<tr>
<td>3.1.1 Hardware Implementation</td>
<td>37</td>
</tr>
<tr>
<td>3.2 Entropy Encoder Unit</td>
<td>40</td>
</tr>
<tr>
<td>3.2.1 Hardware Implementation</td>
<td>43</td>
</tr>
<tr>
<td>4 Synthesizable HDL Model</td>
<td>45</td>
</tr>
<tr>
<td>4.1 Quantizer Unit</td>
<td>45</td>
</tr>
<tr>
<td>4.1.1 Designware Pipelined Multiplier</td>
<td>45</td>
</tr>
<tr>
<td>4.1.2 Designware Adder</td>
<td>46</td>
</tr>
<tr>
<td>4.1.3 Designware Incrementer</td>
<td>46</td>
</tr>
<tr>
<td>4.2 Entropy Encoder Unit</td>
<td>47</td>
</tr>
<tr>
<td>4.2.1 huff_en component</td>
<td>47</td>
</tr>
<tr>
<td>5 Testing and Results</td>
<td>54</td>
</tr>
<tr>
<td>5.1 Quantizer Unit</td>
<td>54</td>
</tr>
<tr>
<td>5.1.1 Testing</td>
<td>54</td>
</tr>
<tr>
<td>5.1.2 Synthesis Results</td>
<td>54</td>
</tr>
<tr>
<td>5.2 Entropy Encoder Unit</td>
<td>67</td>
</tr>
<tr>
<td>5.2.1 Testing</td>
<td>67</td>
</tr>
<tr>
<td>5.2.2 Synthesis Results</td>
<td>67</td>
</tr>
<tr>
<td>6 Conclusion</td>
<td>98</td>
</tr>
<tr>
<td>6.1 Conclusion</td>
<td>98</td>
</tr>
<tr>
<td>6.2 Suggestions for Improvement</td>
<td>99</td>
</tr>
<tr>
<td>6.3 Future Work</td>
<td>100</td>
</tr>
<tr>
<td>Bibliography</td>
<td>101</td>
</tr>
</tbody>
</table>
List of Figures

1.1 Spatial and Temporal Sampling ......................................................... 4
1.2 Video Encoder Block Diagram .......................................................... 6
2.1 International Standards Bodies .......................................................... 12
2.2 H.264/AVC in a Transport Environment ............................................ 15
2.3 Subdivision of a Frame into Slices .................................................... 16
2.4 H.264/AVC Encoder .......................................................................... 17
2.5 Macroblock Partitions for Motion Estimation and Compensation ............ 22
2.6 Scanning Order of Residual Blocks Within a Macroblock ..................... 25
2.7 Zigzag Scan Order ........................................................................... 26
2.8 H.264/AVC Decoder ........................................................................... 28
3.1 Hardware Implementation of H.264/AVC Quantizer ................................ 37
3.2 Basic Data Path of Quantizer .............................................................. 38
3.3 Zigzag Ordering ................................................................................ 40
3.4 Huffman Encoding Example ................................................................ 41
3.5 Huffman Encoding Architecture ........................................................ 43
4.1 Designware Pipelined Multiplier ......................................................... 45
4.2 Designware Adder ............................................................................. 46
4.3 Designware Incrementer ..................................................................... 47
4.4 I/Os for huff_en Block ..................................................................... 47
4.5 VHDL Architecture for huff_en Block ................................................. 48
4.6 VHDL Architecture huff_en_input Block ........................................... 49
4.7 VHDL Architecture for huff_en_shift Block ....................................... 50
4.8 VHDL Architecture for huff_en_merge Block ..................................... 50
4.9 VHDL Architecture for huff_en_arb Block ....................................... 52
4.10 VHDL Architecture for huff_en_output Block ............... 53
5.1 Netlist of quantizer Unit ............................................. 65
5.2 Netlist of quantizer_data_path Unit ............................. 66
5.3 Netlist of huffman_en Unit ........................................... 92
5.4 Netlist of huffman_en_arb Unit ..................................... 93
5.4 Netlist of huffman_en_input Unit ................................... 94
5.4 Netlist of huffman_en_merge Unit ................................. 95
5.4 Netlist of huffman_en_output Unit .................................. 96
5.4 Netlist of huffman_en_shift Unit ................................... 97
List of Tables

1.1 Typical Transmission and Storage Capacities ............................... 2
1.2 Video Frame Rates ..................................................................... 5
2.1 Comparison of H.264 Entropy Encoding Approaches ............... 27
2.2 Average Bit Rate Savings Compared with Various Prior Decoding Schemes ................. 31
3.1 Quantization Step Size ............................................................ 33
3.2 Position Factor Look Up Table .................................................. 34
3.3 Multiplication Factor MF .......................................................... 35
3.4 Huffman Table ................................................................. 41
3.5 AC- Coefficient Coding ............................................................ 42
5.1 Quantizer Results ................................................................. 64
5.2 Entropy Encoder Results ...................................................... 91
List of Equations

3.1 Basic Equation of Quantizer ................................. 32
3.2 Algorithm of Quantizer Including PF .......................... 34
3.3 Divisionless Equation ........................................... 34
3.4 Unsigned Integer Arithmetic Implementation .................. 35
3.5 DC Luma Quantization ........................................... 36
3.6 DC Chroma Quantization ......................................... 36
Glossary

ASIC | Application Specific Integrated Circuit – A specialized hardware designed aimed at a specific application.

AVC | Advanced Video Coding – The latest video compression standard.

CABAC | Context Adaptive Binary Arithmetic Coding. A highly-efficient entropy encoding standard used in the H.264/AVC Main Profile.

CAVLC | Context Adaptive Variable Length Coding. An improved, context adaptive version of VLC used in the H.264/AVC Baseline Profile.

CODEC | Video enCODer DECoder pair.

DCT | Discrete Cosine Transform. A matrix transform commonly used to convert image data from the spatial domain into the frequency domain.

DVD | Digital Versatile Disk. A popular optical disk storage technology used for videos and other applications that require large amounts of storage.

H.264/AVC | Video coding standard approved in Spring 2003 by both ISO/IEC and ITU-T. Delivers significantly better compression than previous standards such as MPEG-2.

HDTV | High definition television. A number of high-quality resolutions standardized for television use. Includes 1080x720 and 1920x1080 resolutions.

HVS | Human Visual System – A term that encapsulates the manner which humans sample and process visual stimuli.

IDR | Instantaneous Data Refresh. A frame that signals the reference picture list that any previous reference frames will no longer be needed.

ISO/IEC | International Standards Organization/International Electrotechnical Commission. ISO is an international body responsible for creating and maintaining a wide range of standards. The IEC is the commission specifically responsible for electrical products and components, including MPEG video compression standards.

MPEG  Moving Picture Experts Group. The group within ISO/IEC that is responsible for adopting and defining video compression standards.

MPEG-2  Video coding standard created by the MPEG group; used extensively for cable television broadcasting and DVDs.

NAL  Network Abstraction Layer. The layer in H.264/AVC that defines how video payloads are stored or transmitted.

PSNR  Peak Signal-to-Noise Ratio. A measure of the objective quality of an image.

QCIF  Quarter-resolution Common Image Format. Defines an image size of 176 pixels wide by 144 pixels high.

QP  Quantization Parameter – Scaling factor used by the encoder during quantization.

RAM  Random Access Memory. Type of reusable data storage that can be accessed in any order.

RBSP  Raw Bit Sequence Payload. The payload containing the actual packet information inside a NAL unit.

VCEG  Video Coding Experts Group. A group from the ITU-T responsible for adopting and defining video compression standards.

VCL  Video coding layer. The layer in the H.264/AVC standard that contains actual video information.

VHDL  Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (HDL). A popular language used for modeling and describing hardware.

VHS  Video Home System. The tape format used in most consumer Video Cassette Recorders (VCRs).
Chapter 1

Introduction

1.1 Video Compression

1.1.1 Compression

Compression is a reversible conversion of data to a format that requires fewer bits, so that the data can be stored or transmitted more efficiently. If the inverse of the process, decompression, produces an exact replica of the original data, then the compression is lossless. This type of compression is useful when the data has a high priority such as medical images. Lossy compression, usually applied to image data, does not allow reproduction of an exact replica of the original image, but it is more efficient. While lossless video compression is possible, in practice it is virtually never used because lossless compression methods can only achieve a modest amount of compression of image and video signals, hence all standard video data rate reduction involves discarding data.

1.1.2 Video Compression

Video compression deals with the compression of digital video data. With the widespread adoption of technologies such as digital television, Internet streaming video and DVD-Video, video compression has become an essential component of broadcast and entertainment media. The goal of video compression algorithm is to achieve efficient
compression whilst minimizing the distortion introduced by the compression process.

Video compression has two important benefits.

- It makes it possible to use digital video in transmission and storage environments that would not support uncompressed (‘raw’) video. For example, a 2-hour uncompressed movie requires over 194Gbytes of storage, equivalent to 42 DVDs or 304 CD-ROMs. [3]

- It enables more efficient use of transmission and storage resources. If a high bitrate transmission channel is available, then it is advantageous to send high-resolution compressed video or multiple compressed video channels than to send a single, low-resolution, uncompressed stream.

Table 1.1 lists typical capacities of popular storage media and transmission networks.

<table>
<thead>
<tr>
<th>Media/Network</th>
<th>Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ethernet LAN</td>
<td>Max 10 Mbps/Typical 1-2 Mbps</td>
</tr>
<tr>
<td>ISDN-2</td>
<td>128 kbps</td>
</tr>
<tr>
<td>V.90 modem</td>
<td>56 kbps downstream/33 kbps upstream</td>
</tr>
<tr>
<td>DVD-5</td>
<td>4.7 Gbytes</td>
</tr>
<tr>
<td>CD-ROM</td>
<td>640 Mbytes</td>
</tr>
</tbody>
</table>

Table 1.1 Typical Transmission/Storage Capacities [3]
1.1.3 Spatial and Temporal Compression

When considering video signals with 30 frames per second, the amount of data to be transmitted and stored increases significantly. Transmission and storage of these huge amounts of data calls for an effective means of compression. This can be achieved by removing redundancy in the temporal (known as interframe or temporal compression), spatial (known as intraframe or spatial compression), and/or frequency domains. These take advantage of the fact that human eye and brain (Human Visual System) are more sensitive to lower frequencies and so the image is still recognizable despite the fact that much of the information has been removed.

Spatial compression is applied to a single frame of video, and compresses the image much like a single image is compressed. The degree of spatial compression affects the overall video quality. Frames compressed with spatial compression are called intraframes.

Temporal compression takes advantage of the fact that consecutive frames of video often contain much of the same pixel data. By identifying differences between consecutive frames of video, and by just transmitting the frame differences, temporal compression can dramatically decrease video data size. Frames compressed with temporal compression are called interframes.
A typical spatial and temporal sampling scenario is as shown in Figure 1.1.

![Diagram of Moving scene with spatial and temporal sampling](image)

**Figure 1.1 Spatial and Temporal Sampling [3]**

A video keyframe is a complete frame of video – not just the computed differences between two frames. Keyframes are used as reference points for subsequent interframes.

1.1.4. Sampling

A digital image may be generated by sampling an analogue video signal at regular intervals. The visual quality of the image is influenced by the number of sampling points. More sampling points give a better image quality; however more sampling points require higher storage capacity. A moving video image is formed by sampling the video
signal temporally. A higher temporal sampling rate (frame rate) gives a smoother appearance to motion in the video scene but requires more samples to be captured and stored. Table 1.2 shows the various video frame rates and the corresponding appearance of video.

<table>
<thead>
<tr>
<th>Video frame rate</th>
<th>Appearance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Below 10 frames per second</td>
<td>‘Jerky’, unnatural appearance to movement</td>
</tr>
<tr>
<td>10-20 frames per second</td>
<td>Slow movements appear OK; rapid movement is clearly jerky</td>
</tr>
<tr>
<td>20-30 frames per second</td>
<td>Movement is reasonably smooth</td>
</tr>
<tr>
<td>50-60 frames per second</td>
<td>Movement is very smooth</td>
</tr>
</tbody>
</table>

**Table 1.2 Video Frame Rates [3]**

1.1.5. Image and Video Compression

A device or a program that compresses a signal is an encoder and a device or a program that decompresses a signal is a decoder. An enCODer/DECoder pair is a CODEC. The CODEC represents the original video sequence by a model (an efficient coded representation that can be used to reconstruct an approximation of the video data). Ideally, the model should represent the sequence using as few bits as possible with as high fidelity as possible. These two goals (compression efficiency and high quality) are usually conflicting, because a lower compressed bit rate typically produces reduced
image quality at the decoder. Hence there is always a tradeoff between bit rate and quality of the image.

1.1.6. Video Encoder

A video encoder consists of three main functional units:

- a temporal model
- a spatial model
- an entropy encoder

![Video Encoder Block Diagram](image)

**Figure 1.2 Video Encoder Block Diagram [1]**

The input to the temporal model is an uncompressed video sequence. It reduces temporal redundancy by exploiting the similarities between neighboring video frames. The output of the temporal model is a residual frame and set of model parameters, typically a set of motion vectors describing how motion was compensated.
The input to the spatial model is the residual frame. The spatial model makes use of the similarities between neighboring samples in the residual frame to reduce spatial redundancy. This is achieved by applying a transform to the residual samples and quantizing the results. The transform converts the samples into another domain in which they are represented by transform coefficients. These coefficients are quantized to remove insignificant values, leaving a small number of significant coefficients that provide a more compact representation of the residual frame. The output of the spatial model is a set of quantized transform coefficients.

The parameters of the temporal and spatial model are compressed by the entropy encoder. This removes statistical redundancy in the data and produces a compressed bit stream or file that may be transmitted and/or stored. A compressed sequence consists of coded motion vector parameters, coded residual coefficients and header information.

1.1.7 Video Decoder

The video decoder reconstructs a video frame from the compressed bit stream. The coefficients and motion vectors are decoded by an entropy decoder after which spatial model is decoded to reconstruct a version of the residual frame. The decoder uses the motion vector parameters, together with one or more previously decoded frames, to create a prediction of the current frame. The frame itself is reconstructed by adding the residual frame to this prediction.

The majority of video CODECs in use today conform to one of the international standards for video coding. The ISO JPEG and MPEG-2 standards have the biggest
impact: JPEG has become one of the most widely used formats for still image storage and MPEG-2 is widely used for digital television and DVD-video systems.

With the continual development of video applications in recent years, there has been an ongoing demand for better compression performance i.e. to deliver better picture quality with a smaller bit rate. The H.264/AVC (Advanced Video Coding) standard was developed to improve on current compression standards like MPEG-1, -2, and -4. The main goals of this standardization effort are to develop a simple and straightforward video coding design, with enhanced compression performance and to provide a “network friendly” video representation which addresses “conversational” (video telephony) and “nonconversational” (storage, broadcast or streaming) applications. Its design provides the most current balance between the coding efficiency, implementation complexity, and cost-based on state of VLSI design technology (ASICS, FPGAs, DSPs). H.264/AVC is based on block transforms and motion compensated predictive coding, but uses improved coding techniques as compared to previous coding standards including:

- Multiple reference frames
- Intra-frame prediction
- Quarter pixel precision motion compensation
- More block sizes for motion compensation
- A 4x4 integer transform that approximates the DCT with a much simpler algorithm
- In-loop deblocking filter to remove blocking artifacts and increase final picture quality.
- Improved entropy coding with CABAC and CAVLC
Error resilience tools for maintaining video quality in error-prone broadcasting

These coding techniques provide better video compression than previous standards [5]. H.264/AVC compliant encoders achieve the same reproduction quality as encoders that are compliant with the previous standards while requiring 60% or less of the bit rate [2], making it much more effective for delivering high-quality video over cable, satellite, and telecom networks. However, this improved compression requires significantly more processing power than previous video standards [6]. Because of this increased complexity, the widespread adoption of H.264/AVC may be limited unless efficient and cost-effective hardware implementations are developed for real-time encoding and decoding of high-resolution video [7].

While reference software is available to demonstrate the expected results of the encoding or decoding process, verifying individual stages of a hardware design is difficult. Designers have to develop the complete encoder or decoder to verify its operation. Not only does this make it difficult to identify and correct errors in a new hardware design, but also prevents new designers from focusing on the development of hardware for a single stage. Thus verifying the designs has become a major challenge for the hardware designers trying to develop hardware implementations of H.264/AVC encoders or decoders.

1.2 Thesis Objective

This thesis aims at designing two basic blocks of an ASIC capable of performing the H.264/AVC video compression. These two blocks, the Quantizer, and the Entropy Encoder implement the Baseline Profile of the H.264/AVC standard. The quantizer has
been modeled using Verilog HDL and the entropy encoder was modeled using VHDL. The HDL used for modeling is synthesizable, giving us an estimate of the hardware requirements in real-time implementation. Synthesis was done using Synopsys Design Compiler, which is CMOS based and gave a reasonable idea of the speed, size and power requirement when implemented as an ASIC. The quantizer makes use of the DWARE components from Synopsys standard library to optimize its speed. The entropy encoder makes use of the GTECH generic library available with Synopsys. The constraints were prioritized so that speed was the important factor. After speed was maximized, the area was reduced as much as possible without affecting the speed, by applying area constraint. These blocks were designed to be easily expandable in the future to include the features of other H.264/AVC profiles, the Main and the Extended Profiles. Both blocks were verified using testbenches, providing individual module verification.

1.3 Thesis Chapter Overview
This thesis starts with an overview of different standardization bodies and the different compression standards developed by them in Chapter 2. The H.264/AVC standard is then explained in detail. This chapter also deals with the important changes made in H.264/AVC over other standards and the performance comparison. Chapter 3 deals with the design algorithms used and the hardware implementation of these algorithms. Chapter 4 discusses the details of the HDL implementation of these blocks. Chapter 5 deals with the testing and presenting of results. Finally, Chapter 6 concludes the thesis with a discussion of future work that could be done and suggestions for improvement.
Chapter 2

Literature Review

This chapter briefly discusses the different standardization groups and the related standards that have been developed by them. It also gives us a brief description of the H.264/AVC video compression techniques. The improvements made in H.264/AVC as compared to the previous standards were also discussed.

2.1 Standards of Video Compression

2.1.1 Standardization Groups

A video coding standard describes the syntax for representing compressed video data and the procedure for decoding this data as well. Over the last two decades, two standard bodies have developed a series of standards for video compression techniques. They are

- International Standards Organization (ISO)
- International Telecommunications Union (ITU)

There are two working groups for each of these standard bodies which are responsible for the development of the standards for video compression. They are

- Moving Picture Experts Group (MPEG) of the ISO
- Video Coding Experts Group (VCEG) of the ITU-T
2.1.2 Related Standards

The popular standards developed by the MPEG are JPEG and JPEG-2000 for still images, MPEG-1, MPEG-2 and MPEG-4 for moving video (digital television and DVD-video systems).

The popular standards developed by VCEG are H.261, H.263 and H.26L standards. H.261 was originally developed for videoconferencing over the ISDN, but H.261 and H.263 are now widely used for real-time video communications over a range of networks including the Internet.

Figure 2.1 shows the International standards bodies and the video standards produced by these bodies, targeting a wide range of applications from video teleconferencing to TV broadcasting and DVDs.
The ITU-T is responsible for the H-series of video standards, which especially target video conferencing applications. Their most recent video conferencing standard, H.264, has undergone two major revisions to produce H.263++ (also called H.263 High Latency Profile (HLP)). The MPEG series of video standards have especially targeted high-end video applications. MPEG-2 is currently used for DVDs and broadcast television. MPEG-4 ASP (Advanced Simple Profile), also called MPEG-4 Version 1, was developed primarily for Internet video streaming applications.

Beginning in 1997 the two groups combined efforts to put together the next generation of video compression standard. MPEG-2 (also known as H.262) and H.264/AVC are the only two video standards ever to be developed jointly by ITU-T and ISO/IEC. H.264/AVC, which was approved in May 2003, has achieved bit-rate savings by a factor of two as compared with existing standards such as MPEG-2 video [7]. H.264/AVC addresses the full range of video applications, from low-bandwidth wireless uses, low-and high-definition television, video streaming over the Internet, high-quality DVD content, and extremely high-quality video for use in movie theaters [9].

### 2.2 H.264/AVC Video Compression

H.264/AVC is the latest standard for video compression with the goals of enhanced compression efficiency, network friendly video representation for interactive (video telephony) and non-interactive applications (broadcast, streaming, storage, video on demand). H.264/AVC follows the basic video encoding and decoding steps, but additional techniques are included that allow H.264/AVC to achieve 30-70% better compression than MPEG-2, as well as substantial perceptual quality improvements [2].
2.2.1 NAL and VCL

The H.264 standard defines the bit stream protocol. The bit stream is divided and processed in two layers:

- Network Abstraction Layer (NAL)
- Video Coding Layer (VCL)

NAL is directed towards making the bitstream transmission compliant and VCL defines the actual format that encoded video data must adhere to. It is the responsibility of NAL to encapsulate the data produced by VCL.

The H.264 NAL defines NAL units that packet the coded video data. A NAL unit consists of a single header byte and corresponding payload. The first bit of the header is always zero, bits 1-2 represent the NAL reference ID, and bits 3-7 identify what type of data is contained within the appended payload. NAL units are categorized into VCL and non-VCL units. The payload of a VCL unit contains actual encoded video data that translates into frames. The payload of a non-VCL unit contains information that describes the format of the data stream.

The VCL contains the actual encoded video frames. The H.264 is a block-based hybrid decoding standard [9], i.e. the image is broken down into rectangular blocks and both temporal and spatial predictions are performed. The residual of the predictions themselves are sent or stored as the payload within a NAL unit. Figure 2.2 shows H.264/AVC in a transport environment.
2.2.2 Macroblocks and Slices

The H.264 defines a macroblock as a 16x16 luminance region and its corresponding 8x8 chrominance values. One of the major advances that the H.264/AVC offers is the ability to encode sub-blocks down to 4x4 for motion prediction. A series of macroblocks are grouped together into a slice. An image may be composed of a single or several slices. Furthermore, slices that share properties can be combined into slice groups. Slice groups have no geometric constraints and the number of macroblocks per slice need not be constant within a picture. Figure 2.3 shows the subdivision of a frame into slices and slice groups.
Macroblocks within a slice are processed in a raster scan order. The slices are decoded in the order that they are read or received.

The H.264/AVC standard defines five types of slices I, P, SI, SP, and B. A coded picture may be composed of different types of slices. A Baseline Profile bit stream may include only I and P slices. The Main or Extended Profile coded picture may contain a mixture of I, P and B slices. An I slice contains only I macroblocks that are encoded using both inter and intra prediction. A macroblock may be compressed using either algorithm. The encoder determines which method yields the highest compression rate and groups them into slices accordingly. Intra prediction is aimed at removing spatial redundancy and uses adjacent previously encoded frames. P slice contains P macroblocks and/or I macroblocks. B slice contains B macroblocks and/or I macroblocks. SP slice contains P and/or I macroblocks. SI contains SI macroblocks (a special type of intra coded macroblock). SP and SI slices facilitate switching between coded streams.
2.2.3 H.264/AVC Encoder

The encoder includes two dataflow paths, a forward path and a reconstruction path. With the exception of deblocking filter, most of the basic functional elements are present in the previous standards, but important changes occur in the details of each functional block. The basic building blocks of a H.264/AVC encoder are shown in Figure 2.4.

![Figure 2.4 H.264/AVC Encoder](image)

**Figure 2.4 H.264/AVC Encoder [1]**

Encoder (Forward Path): An input frame is processed in units of macroblock. Each macroblock is encoded in intra or inter mode, and for each block in the macroblock, a prediction ‘P’ is formed based on the reconstructed picture samples. The term “block”
is used to denote a macroblock partition or sub-macroblock partition (inter coding) or a 16x16 or 4x4 block of luma samples and associated chroma samples (intra coding). In intra mode, ‘P’ is formed from samples in the current slice that have previously encoded, decoded and reconstructed. In inter mode, ‘P’ is formed by motion-compensated prediction from one or two reference pictures. In Figure 2.4, the reference picture is shown as the previous encoded picture F'_{n-1}. The prediction ‘P’ is subtracted from the current block to produce a residual block D_n that is transformed and quantized to give X, a set of quantized transform coefficients which are reordered and entropy encoded. The entropy-encoded coefficients, together with side information required to decode each block within the macroblock form the compressed bitstream which is passed to a Network Abstraction Layer (NAL) for transmission or storage [1].

Encoder (Reconstruction Path): The encoder reconstructs each block in a macroblock to provide a reference for future predictions. The coefficients X are rescaled (Q^{-1}) and inverse transformed (T^{-1}) to produce a difference block D'_n. The prediction block ‘P’ is added to D'_n to create a reconstructed block uF'_n (u indicates that it is unfiltered). A filter is applied to reduce the effects of blocking distortion and the reconstructed reference picture is created from a series of blocks F'_n [1].

Decoder: The decoder receives a compressed bitstream from the NAL and entropy decodes the data elements to produce a set of quantized coefficients X. These are scaled and inverse transformed to give D'_n. Using the header information decoded from the bitstream, the decoder creates a prediction block ‘P’, identical to the original prediction ‘P’ formed in the encoder. ‘P’ is added to D'_n to produce uF'_n which is filtered to create each decoded block F'_n.
2.2.3.1 Intra Prediction

When a block or macroblock is coded in intra mode, a prediction block is formed based on previously encoded and reconstructed blocks in the same frame. This prediction is subtracted from the current macroblock or block and the result of the subtraction (residual) is compressed and transmitted to the decoder, together with the information required for the decoder to repeat the prediction process. The decoder creates an identical prediction and adds this to the decoded residual or block. The encoder bases its prediction on encoded and decoded image samples (rather than on original video frame samples) in order to ensure that the encoder and decoder predictions are identical. Intra prediction may occur at both the luma macroblock (16x16) and sub-block levels.

2.2.3.2 Inter Prediction

Inter Prediction creates a prediction model from one or more previously encoded video frames using block based motion compensation. It aims at removing temporal redundancies in a video sequence. Inter prediction macroblocks must reside in P-slices and require a history of previously encode frames to be kept in memory. The encoder manages the reference frame buffer and communicates to the decoder via the bit-stream regarding what images to keep in its buffer. The availability of multiple reference frames for motion compensation is a new feature offered with the H.264/AVC standard. It proves most useful in sequences with repetitive motion or appearances.

For inter prediction a 16x16 macroblock can be partitioned into any 4x4 multiple. If the macroblock is broken into four 8x8 blocks, an additional field is added to the bit
stream for each sub block to specify whether or not and how the 8x8 sub block is partitioned. Chroma blocks are divided according to their luma counterpart, i.e. the largest chroma block is 8x8 and the smallest is 2x2. The chroma block is half the resolution of the luma. Each macroblock partition has a motion vector and a reference number associated with it. For an 8x8 partition, only one reference frame may be used. All four 4x4 blocks within an 8x8 partition must all use the same reference frame. The reference frame number specifies which frame the prediction used and the vector correlates to the block used within the referenced frame. If the encoder decides to divide a macroblock into 4x4 partitions, it must send sixteen motion vectors and reference frame numbers. It is upto the encoder to balance the tradeoff between the cost of transmitting/storing motion vectors and the savings of accurate motion prediction that results in low energy residuals [11].

Important differences from earlier standards include support for a range of block sizes (from 16x16 down to 4x4) for motion compensation, support for multiple reference frames (reference frame can be chosen from a set of ‘n’ frames), intra-frame prediction, quarter sample resolution in the luma component, and an in-loop deblocking filter (used to remove blocking-distortion).
2.2.3.3 Motion Estimation

Since multiple video frames are displayed each second, long sequence of image frames can contain very similar data. Motion estimation compares the sequence of image frames in a video to find temporal redundancies and only encode the changes that occur between frames. These changes are often confined to specific portions of the image where movement is occurring, allowing motion estimation techniques to result in a large decrease in the video stream size [17].

Three different types of picture frames can be encoded by the motion estimation block: I, P, and B. I-frames are coded independently from any other frames. These provide a baseline reference for the other frames to be decoded from. Because they include a full picture frame worth of data, they can only be compressed moderately. P-frames are predictively coded picture frames, encoded with reference to previous I- or P-frames. B-frames (bi-directionally predictive-coded frames) are the most highly compressed type of frame, making reference to both past and future I- or P-frames in the video sequence [5].

2.2.3.4 Tree Structured Motion Compensation

H.264/AVC supports motion compensation block sizes ranging from 16x16 to 4x4 luminance samples with many options between the two. Figure 2.5 shows the different macroblock partitions for motion estimation and compensation.
The luminance component of each macroblock (16x16) may be split up in four ways as shown in Figure 2.5: 16x16, 16x8, 8x16, 8x8. Each of the sub-divided regions is a macroblock partition. If the 8x8 mode is chosen, each of the four 8x8 macroblock partitions within the macroblock may be split in a further four ways as shown in Figure 2.5: 8x8, 8x4, 4x8, 4x4 (known as macroblock sub-partitions). These partitions and sub-partitions give rise to a large number of possible combinations within each macroblock. This method of partitioning macroblocks into motion compensated sub-blocks of varying size is known as tree structured motion compensation. A separate motion vector is required for each partition or sub-partition. Each motion vector must be coded and transmitted; in addition, the choice of partition(s) must be encoded in the compressed bit-stream. Choosing a large partition size (e.g. 16x16, 16x8, 8x16) means that a small number of bits are required to signal the choice of motion vector(s) and the type of partition; however, the motion compensated residual may contain a significant amount of
energy in frame areas with high detail. Choosing a small partition size (e.g. 8x4, 4x4 ) may give a lower energy residual after motion compensation but requires a large number of bits to signal the motion vectors and the choice of partition(s). The choice of partition size has a significant impact on the compression performance. In general, a large partition size is appropriate for homogeneous areas of the frame and a small partition size may be beneficial for detailed areas [3].

2.2.3.5 Transform and Quantization

Many a time, the spatial domain is not the most efficient place to work in, it is quite difficult to separate high frequency data in spatial domain. The transform stage transforms the image data from the spatial domain into the frequency domain such as Fourier or Discrete Cosine. The idea is that high frequencies in an image may be removed without risking the integrity of the image. The H.264/AVC uses an integer version of the Discrete Cosine Transform (DCT). This transformation reorders the block data according to its frequency grouping low frequency information together. High frequency data shows up as edges or boundaries while low frequency data resides in smooth regions. Removing low frequency or DC energy results in a drastically different image whereas high frequency energy may be removed without affecting the integrity of the image. Thus low frequency data has high priority while the high frequency data has low priority. By transforming images from spatial domain into the frequency domain, low priority data may be easily removed. DCT requires floating point arithmetic which complicates hardware, hence to simplify the transform, H.264/AVC standard defines three transforms that only require simple 16-bit integer arithmetic.
The purpose of quantization is to remove the components of the transformed data that are unimportant (high frequency coefficients) to the visual appearance of the image and to retain the visually important components (low frequency components). Removing high frequency coefficients removes image information but maintains most of the perceptual quality since human eye cannot distinguish high frequency detail very well. Once removed the less important components cannot be replaced and so quantization is a lossy process. The amount of quantization can be adjusted depending on the desired image quality and compression rate. The quantizer step size between successive rescaled values is the critical parameter used to control image quality and compression in an image or video CODEC. If the step size is large, the range of quantized values is small and can be highly compressed during transmission, but the rescaled values are a rough approximation to the original signal. If the step size is small, the re-scaled values match the original signal more closely but the larger range of quantized values reduces compression efficiency. A scalar quantizer maps one sample of the input signal to one quantized output value and a vector quantizer maps a group of input samples (a vector) to a group of quantized values.

H.264 uses a scalar quantizer. After the transform and quantization stages, coefficients are in order from lower frequency to higher frequency. Since higher frequency coefficients tend to be zero, this ordering produces a considerable coding improvement in the entropy coding stage.
2.2.3.6 Reordering

Reordering is to group the data into groups of nonzero and zero coefficients. Efficient representation of zero coefficients is done before entropy encoding. In the encoding path after transform and quantization a 16x16 macroblock consists of 16-4x4 luma coefficient blocks and 8-4x4 chroma coefficient blocks as shown in Figure 2.6.

![Figure 2.6 Scanning Order of Residual Blocks within a Macroblock [1]](image)

If the macroblock was compressed using 16x16 intra prediction then an additional 4x4 and 2-2x2 coefficient blocks are created from the DC coefficients. In such cases, the blocks are sent to the entropy encoder starting with block -1 and finishing with block 25. Otherwise blocks -1, 16 and 17 do not exist and are therefore excluded. The actual coefficients in a 4x4 block are sent in a zigzag scan order as shown in Figure 2.7.
Frame macroblocks are sent in zigzag order and field macroblocks are sent in field scan order.

2.2.2.7 Entropy Coding

Entropy encoding techniques are aimed at bit-level information. Entropy coding compresses the final serial data stream by mapping frequently used symbols to actual bit codes. The most frequently occurring symbols are mapped to shorter bit codes, while less frequently occurring symbols are mapped to longer bit codes. This lossless encoding reduces the bandwidth of the final video stream while allowing the data to be completely reconstructed after transmission.

H.264/AVC offers improved entropy coding to compress the final data bit stream. Instead of the older Variable Length Coding (VLC) used by MPEG-2, H.264/AVC offers two new entropy coding techniques, called Context Adaptive Binary Arithmetic Coding (CABAC) and Context Adaptive Variable Length Coding (CAVLC). CABAC uses arithmetic coding with non-integer codewords to allow greater bit rate reduction. It is capable of adapting to different probability distributions of data in order to better correlate the current bit patterns. CAVLC offers some of the entropy coding...
improvements of CABAC without all of the hardware complexity. CAVLC is a more adaptive version of VLC with multiple code tables that can be used on the current context of the video data. A comparison of the entropy coding types is shown in Table 2.1.

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>VLC</th>
<th>CABAC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Where it is used</td>
<td>MPEG-2, MPEG-4, ASP</td>
<td>H.264/MPEG-4/AVC (high efficiency option)</td>
</tr>
<tr>
<td>Probability distribution</td>
<td>Static: probabilities never change</td>
<td>Adaptive: Adjusts probabilities based on actual data</td>
</tr>
<tr>
<td>Leverages correlation between symbols</td>
<td>No: conditional probabilities ignored</td>
<td>Yes: exploits symbol correlations by using contexts</td>
</tr>
<tr>
<td>Noninteger code words</td>
<td>No: low coding efficiency for high coding symbols</td>
<td>Yes: exploits arithmetic coding which generates non-integer code words for higher efficiency.</td>
</tr>
</tbody>
</table>

Table 2.1 Comparison of H.264 Entropy Coding Approaches [10]

2.2.3.8 Deblocking Filter

A deblocking filter is implemented in the H.264/AVC standard to reduce artifacts produced by various compression techniques. By partitioning the frame into macroblocks the decoded image may have blocked artifacts. A higher degree of decoding will increase the likelihood of “blocked” images. The deblocking filter operates on both 16x16 pixel macroblocks and on 4x4 pixel block boundaries. For
macroblocks, the filter reduces artifacts caused by different types of motion or intra estimation being used in adjacent blocks. The filter also helps to remove artifacts caused by transform/quantization of adjacent 4x4 blocks or from motion vector differences. The deblocking filter operates on the two pixels on either side of a boundary using a context adaptive non-linear filter [1]. The exact filter and filter strength used are dynamically chosen according to the macroblock content and encoding method.

### 2.2.4 H.264/AVC Decoder

The decoder receives a compressed bitstream from the NAL and entropy decodes the data elements to produce a set of quantized coefficients X. These are scaled and inverse transformed to give $D_n'$. Using the header information decoded from the bit stream, the decoder creates a prediction block ‘P’, identical to the original prediction ‘P’ formed in the encoder. ‘P’ is added to $D_n'$ to produce $uF_n'$ which is filtered to create each decoded block $F_n'$. Figure 2.8 shows the block diagram of an H.264/AVC decoder.

![Figure 2.8 H.264/AVC Decoder [1]](image-url)
2.2.5 H.264/AVC Profiles

To address the large range of applications considered by H.264/AVC, three profiles have been defined. Each profile adds a level of flexibility and complexity. They are:

- Baseline Profile
- Main Profile
- Extended Profile

Baseline Profile: Typically considered the simplest profile, includes all the H.264/AVC tools with the exception of the following tools: B-slices, weighted prediction, field (interlaced) coding, picture/macroboblock adaptive switching between frame and field coding, SP/SI slices and slice data partitioning. This profile targets applications with low complexity and low delay requirements. The Baseline Profile supports intra and inter prediction using I and P frames and entropy coding using Context Adaptive Variable Length Coding (CAVLC).

Main Profile: Supports together with the Baseline Profile a core set of tools; however, regarding Baseline, Main does exclude redundant pictures features while including B slices, weighted prediction, field coding, picture/macroboblock adaptive switching between frame and field coding, and CABAC. This profile typically allows the best quality at the cost of higher complexity (especially due to the B-slices and CABAC) and delay. The Main Profile includes support for interlaced video, inter prediction using B frames and weighted prediction, and entropy coding using Context Adaptive Binary Arithmetic Coding (CABAC).

Extended Profile: This profile is a superset of the Baseline Profile supporting all tools in the specification with the exception of CABAC. The SP/SI slices and slice data
partitioning tools are only included in this profile [12]. The Extended Profile does not support interlaced video or CABAC, but adds modes for efficient switching between coded bitstreams and improved error resilience using data partitioning.

2.2.6 Performance Comparison

Though H.264/AVC has the same basic blocks as most of the other CODECs, the difference lies in the details of each block. Some of the major improvements in H.264/AVC over the previous standards are given below.

Motion Estimation

H.264/AVC introduces smaller block sizes, greater flexibility in block shapes, and greater precision in motion vectors. This can result in a much higher temporal compression because of the improved motion prediction that can be accomplished. H.264/AVC also introduces the ability to use multiple reference frames for motion estimation.

Intra Estimation

Intra estimation is a new feature added by H.264/AVC. Intra estimation can be used to spatially compress an image when motion estimation does not give good results. This works particularly well on flat backgrounds where the image changes in some consistent way [5].

Transform

The integer transform used by H.264/AVC approximates the DCT, but uses substantially simpler arithmetic. Smaller block sizes of 4x4 pixels are encoded and decoded rather than 8x8 blocks, resulting in less blocking or ringing artifacts when compressed by the quantization stage and therefore resulting in a better image quality. The integer transform
matrix coefficients have been adjusted to be integers or simple ratios (such as ½) so that no multiplications are needed in the transform stage (a scaling multiplication is done in the quantization stage). This means that all arithmetic for the transform can be accomplished using additions and shifts [5].

Quantization

The scalar quantizer used by H.264/AVC also avoids any division or floating-point arithmetic, enabling simpler integer arithmetic to be used. The quantization stage uses mostly shifts and additions and only one multiplication per coefficient. The quantization stage incorporates the post- and pre-scaling factors for the integer transform.

Because of the algorithm changes highlighted above, H.264/AVC compliant encoders achieve the same reproduction quality as encoders that are compliant with the previous standards while requiring 60% or less of the bit rate [2]. The bit rates for TV or HD video (at broadcast and DVD quality) are reduced by a factor of between 2.25 and 2.5 when using H.264/AVC coding [7]. Table 2.2 shows the average bitrate savings of each encoder relative to all other tested encoders over the entire set of sequences.

<table>
<thead>
<tr>
<th>Coder</th>
<th>MPEG-4 ASP</th>
<th>H.263 HLP</th>
<th>MPEG-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>H.264/AVC</td>
<td>38.62%</td>
<td>48.80%</td>
<td>64.46%</td>
</tr>
<tr>
<td>MPEG-4 ASP</td>
<td>---</td>
<td>16.65%</td>
<td>42.95%</td>
</tr>
<tr>
<td>H.263 HLP</td>
<td>---</td>
<td>---</td>
<td>30.61%</td>
</tr>
</tbody>
</table>

Table 2.2 Average bit-rate savings compared with prior coding schemes [7].
Chapter 3

Design Procedure and Algorithms

3.1 Quantizer Unit

As mentioned earlier, data contained in an image is prioritized according to frequency. The low frequency data has high priority while the high frequency data has low priority. By transforming images from spatial domain into frequency domain, low priority data may be easily removed. Quantization maps the transformed data to a reduced range of values so that the signal can be transmitted using fewer bits than the original signal. The process is lossy, since it involves rounding fractional number to the nearest integer. The basic algorithm of a quantizer is shown in Equation 3.1.

\[
Z_{i,j} = \text{round} \left( \frac{Y_{i,j}}{Q_{\text{step}}} \right) \quad \text{----> Equation 3.1}
\]

**Equation 3.1 Basic Equation of Quantization [1]**

In the above Equation 3.1, \( Y_{i,j} \) is the input matrix, \( Q_{\text{step}} \) is the quantizer step size, and \( Z_{ij} \) is the output matrix. The rounding operation need not round to the nearest integer; for example, rounding towards smaller integers can give perceptual quality improvements [1]. H.264/AVC uses three transforms depending on the type of residual data: DC luma transformed array of coefficients (4x4 matrix) in intra macroblocks predicted in 16x16 mode, DC chroma transformed array of coefficients (2x2 matrix) in any macroblock and
a residual data transformed array of coefficients (4x4 matrix) for all the other blocks in the residual data.

**Residual mode**

The basic forward quantization algorithm is described in Equation 3.1. A total of 52 values of Qstep are supported and indexed by the input value QP. Each increase in QP corresponds to a 12.5% increase in Qstep and Qstep doubles in size for every increment of 6 in QP. Quantization step size values are shown in Table 3.1.

<table>
<thead>
<tr>
<th>QP</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td>QStep</td>
<td>0.625</td>
<td>0.6875</td>
<td>0.8125</td>
<td>0.875</td>
<td>1.000</td>
<td>1.125</td>
<td>1.250</td>
<td>1.375</td>
<td>1.625</td>
<td>1.750</td>
<td>2.000</td>
<td>2.25</td>
<td>2.5</td>
<td>…</td>
</tr>
<tr>
<td>QP</td>
<td>…</td>
<td>18</td>
<td>…</td>
<td>24</td>
<td>…</td>
<td>30</td>
<td>…</td>
<td>36</td>
<td>…</td>
<td>42</td>
<td>…</td>
<td>48</td>
<td>…</td>
<td>51</td>
</tr>
<tr>
<td>QStep</td>
<td>…</td>
<td>5</td>
<td>…</td>
<td>10</td>
<td>…</td>
<td>20</td>
<td>…</td>
<td>40</td>
<td>…</td>
<td>80</td>
<td>…</td>
<td>160</td>
<td>…</td>
<td>224</td>
</tr>
</tbody>
</table>

**Table 3.1 Quantization Step Sizes in H.264/AVC CODEC [1]**

The wide range of Qstep makes it possible for the encoder to control the tradeoff accurately and flexibly between bit rate and quality [1]. The values of QP can be different for luma and chroma, both parameters are in the range 0-51 and the default is that the chroma parameter QPc is derived from QPy so that QPc is less than QPy for QPy >30.

The quantizer also incorporates a post scaling factor PF from the previous transform block. PF is $a^2$, $ab/2$ or $b^2/4$ depending on the position (i, j), determined according to Table 3.2 where $a = \frac{1}{2}$ and $b = \sqrt{2/5}$. 

33
Incorporating PF gives us Equation 3.2

\[
Z_{ij} = \text{round} \left( W_{ij} \frac{PF}{Q\text{step}} \right) \quad \text{----> Equation 3.2}
\]

**Equation 3.2. Algorithm of Quantizer Including Position Factor (PF) [1]**

In Equation 3.2, \(W_{ij}\) is the unweighted input matrix, PF is the position factor, and Qstep is the quantization step size. In order to simplify the arithmetic, the factor \((PF/Q\text{step})\) is implemented as a multiplication by a factor \(MF\) and a right shift, thus avoiding division operations.

\[
\frac{MF}{2^{q\text{bits}}} = \frac{PF}{Q\text{step}} \quad \text{and} \quad q\text{bits} = 15 + \text{floor} \left( \frac{Qr}{6} \right)
\]

\[
Z_{i,j} = \text{round} \left( W_{i,j} \frac{MF}{2^{q\text{bits}}} \right) \quad \text{----> Equation 3.3}
\]

**Equation 3.3 Divisionless Equation [1]**

<table>
<thead>
<tr>
<th>Position</th>
<th>PF</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0, 0), (2, 0), (0, 2) or (2, 2)</td>
<td>(a^2)</td>
</tr>
<tr>
<td>(1, 1), (1, 3), (3, 1) or (3, 3)</td>
<td>(b^2/4)</td>
</tr>
<tr>
<td>Other</td>
<td>(ab/2)</td>
</tr>
</tbody>
</table>

Table 3.2 Position Factor (PF) Look-Up Table [1]
Since the division operation in Equation 3.3 is an integer power of two, it can be implemented as a simple right shift resulting in the final unsigned integer arithmetic version shown in Equation 3.4.

\[
|Z_{i,j}| = \left( |W_{i,j}| MF + f \right) \gg q\text{bits} \\
\text{sign}(Z_{i,j}) = \text{sign}(W_{i,j})
\]  
\[\text{-----}\rightarrow \text{Equation 3.4}\]

**Equation 3.4 Unsigned Integer Arithmetic Implementation [1]**

In Equation 3.4 \(\gg\) indicates a binary shift right. The factor \(f\) represents dead zone compensation factor. In the reference model, \(f\) is \(2^{q\text{bits}}/3\) for intra blocks and \(2^{q\text{bits}}/6\) for inter blocks. The factors \(f\), \(q\text{bits}\), and \(MF\) are fixed, known values. For hardware implementation, they will be precalculated and placed in LUTs (Look Up Table) indexed by \(QP\). Table 3.3 shows the Multiplication Factor (MF) for different values of \((i, j)\).

<table>
<thead>
<tr>
<th>(QP)</th>
<th>Positions</th>
<th>((0,0),(2,0),(2,2),(0,2))</th>
<th>Positions</th>
<th>((1,1),(1,3),(3,1),(3,3))</th>
<th>Other positions</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>13107</td>
<td>5243</td>
<td>8066</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>11916</td>
<td>4660</td>
<td>7490</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>10082</td>
<td>4194</td>
<td>6554</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>9362</td>
<td>3647</td>
<td>5825</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>8192</td>
<td>3355</td>
<td>5243</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>7282</td>
<td>2893</td>
<td>4559</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 3.3. Multiplication Factor (MF) [1]**
4x4 luma DC Quantization and 2x2 chroma DC coefficient quantization

If the macroblock is encoded in 16x16 intra mode, then the quantization algorithm is slightly altered for the quantization of the input matrix because of a difference in the previous transform block. The algorithm for the DC luma transformed array is shown in Equation 3.5.

\[
\left| Z_{D(i,j)} \right| = \left( \left| Y_{D(i,j)} \right| MF_{(i,j)} + 2 f \right) \gg (q\text{bits} + 1)
\]

\[
\text{sign} \left( Z_{D(i,j)} \right) = \text{sign} \left( Y_{D(i,j)} \right)
\]

-----\rightarrow \text{Equation 3.5}

**Equation 3.5 DC Luma Quantization [1]**

MF\(_{(i,j)}\) is the multiplication factor for the position \((i, j)\), \(f\) and \(q\) bits are as defined before.

The 2x2 chroma coefficient quantization algorithm is shown in Equation 3.6. The input and output matrices are 2x2 instead of 4x4 and the MF for the position vector \((0,0)\) is used.

\[
\left| Z_{D(i,j)} \right| = \left( \left| Y_{D(i,j)} \right| MF_{(0,0)} + 2 f \right) \gg (q\text{bits} + 1)
\]

\[
\text{sign} \left( Z_{D(i,j)} \right) = \text{sign} \left( Y_{D(i,j)} \right)
\]

-----\rightarrow \text{Equation 3.6}

**Equation 3.6 DC Chroma Quantization [1]**
3.1.1 Hardware Implementation

H.264/AVC uses scalar quantization algorithm that has been specifically designed with hardware implementation in mind. Hence, the quantizer can be implemented without the use of floating point arithmetic or integer division. Figure 3.1 shows the hardware implementation of the H.264/AVC quantizer.

![Figure 3.1 Hardware Implementation of H.264/AVC Quantizer](image)

The hardware implementation of the quantizer consists of two pieces, the data paths and the LUT. The first stage of the pipeline will accept the block input and use it to look up the required values for the coefficients MF, f, and qbits. These coefficients, along with the input matrix will then be fed through an array of 16 pipelined multipliers. The final stage will then implement the add and shift.

The LUT accepts as inputs QP and the current value of mode. From this the value of MF for each data path is determined, as well as the values of f and QP_div_6 (15 + QP_div_6 = qbits). In this implementation, the LUT was modeled using Verilog case statements and constants providing all the required values in 1 clock cycle. Since QP is limited to a maximum of 52 values, these mathematical calculations do not need to be
performed in hardware. To improve the performance of the LUTs, the results of QP divided by 6 and QP mod 6 are precalculated and built into the case statements.

The outputs from the LUT along with the delayed Y input (Y was delayed one cycle to match the delay from the lookup table.) were then fed into an array of sixteen data paths. Each data path received one element of the 4x4 input matrix as well as the corresponding MF, f, QP_div_6, and mode values. The basic data path is shown in Figure 3.2.

![Figure 3.2 Basic Data Path of Quantizer [18]](image)

In Figure 3.2, vertical bars indicate pipelined registers. The full data path consists of eight pipeline stages, six for the multiplication and two for the add and shift. The multiplication is implemented with a Wallace Tree Multiplier. Multiplication of the 16-bit Y value and 14-bit MF yields a 30-bit result.

In early implementations, a one stage thirty bit adder was used, accomplishing the full add and shift in one cycle. This proved to be the critical path within the data path and was redesigned to use the implementation in Figure 3.2. In this implementation, the 30-bit addition is broken into two 15-bit additions, thus greatly reducing the time needed to
propagate carries across the addition. Additionally, since qbits is defined as 15 + QP_div_6, the result of the addition will always be right shifted a minimum of 15 places. Consequently, only the carry out of the lower half of the addition is required. The second stage of the add and shift portion, then propagates the carry from the lower half to the upper using a 16-bit incrementer circuit. Both the adders and the incrementer are implemented as Fast Carry Look-Ahead Adders. The carry out from the lower half is used as the select line to a bank of multiplexers to choose either the incremented or non-incremented value for shifting. Finally, a barrel shifter is used to right shift the value by QP_div_6 places to produce the final output.

In addition to the basic data path described above, a mode input is provided to select between various quantization modes. When in DC luma or DC chroma modes, the mode input is used as select line for left shifting the f input one place and then right shifting the final output one additional place. When operating in 2x2 DC chroma mode, the outputs from the 12 unnecessary datapaths are ignored, the current implementation does not explicitly shut them off.

Finally, if this design were to be combined with the hardware implementation of the preceding transform block, total latency could be reduced from nine cycles to eight. Currently, no processing is done on the Y input during the first cycle when the LUT is being accessed. Since all the LUT inputs are control lines not dependent upon Y, the LUT access could be done in parallel with the last stage or stages of the transform block, thus reducing the latency from nine clock cycles to eight clock cycles.
3.2 Entropy Encoder Unit

A Huffman entropy encoder maps each input symbol into a variable length codeword based on the probability of occurrence of different symbols. The constraints on variable length codeword are that it must (i) contain an integer number of bits and (ii) be uniquely decodable (i.e. the decoder must be able to identify each code word without ambiguity. In entropy encoding, the coefficients of the incoming matrix are read in zigzag order as shown in Figure 3.3.

![Zigzag Ordering](image)

**Figure 3.3 Zigzag Ordering [1]**

This ordering is arranged based on the increasing spatial frequency. Since many of the higher frequency values will be zero, this zigzag pattern will be beneficial for the coding scheme used.

Initially, a new DC-coefficient is determined using differential pulse-code modulation (DPCM). This is determined by taking the difference between the current DC-coefficient and the DC-coefficient of the previous 8x8 block used. If there was no previous block, then the previous value is set to 0. For 8-bit gray-scale pixel values, the
maximum size value from the DCT was determined to be 11 bits plus a sign bit. A
difference magnitude coding, SSSS, is then determined from the following table. The
SSSS value is a 4-bit value representing the size of the value. Table 3.4 shows the SSSS
value and the corresponding Huffman Codes.

<table>
<thead>
<tr>
<th>SSSS</th>
<th>Difference</th>
<th>Code Length</th>
<th>Huffman Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>00</td>
</tr>
<tr>
<td>1</td>
<td>-1,1</td>
<td>3</td>
<td>010</td>
</tr>
<tr>
<td>2</td>
<td>-3,-2,2,3</td>
<td>3</td>
<td>011</td>
</tr>
<tr>
<td>3</td>
<td>-7,-4,4,7</td>
<td>3</td>
<td>100</td>
</tr>
<tr>
<td>4</td>
<td>-15,-8,8,15</td>
<td>3</td>
<td>101</td>
</tr>
<tr>
<td>5</td>
<td>-31,-16,16,31</td>
<td>3</td>
<td>110</td>
</tr>
<tr>
<td>6</td>
<td>-63,-32,32,63</td>
<td>4</td>
<td>1110</td>
</tr>
<tr>
<td>7</td>
<td>-127,-64,64,127</td>
<td>5</td>
<td>11110</td>
</tr>
<tr>
<td>8</td>
<td>-255,-128,128,-255</td>
<td>6</td>
<td>1111110</td>
</tr>
<tr>
<td>9</td>
<td>-511,-256,256,-511</td>
<td>7</td>
<td>11111110</td>
</tr>
<tr>
<td>10</td>
<td>-1023,-512,512,-1023</td>
<td>8</td>
<td>111111110</td>
</tr>
<tr>
<td>11</td>
<td>-2047,-1024,1024,-2047</td>
<td>9</td>
<td>1111111110</td>
</tr>
</tbody>
</table>

Table 3.4 Huffman Table [13]

The coding obtained from the above table is then encoded using Huffman tables.
Huffman tables are based primarily on the probabilities of the values used. The more
frequently used values have the shortest codes assigned to them. There is no standard or
default table that is always used. The value is finally encoded using the Huffman value
for the difference magnitude with the sign bit attached to the end. The value is attached
afterwards but without the most significant bit (MSB) since the code itself represents the
size of the value. The example in Figure 3.4 shows how a value is encoded.

<table>
<thead>
<tr>
<th>Value</th>
<th>Sign bit</th>
<th>Value (binary)</th>
<th>Size</th>
<th>Huffman Code</th>
<th>Value - MSB (binary)</th>
<th>Output (code + sign + (value - MSB))</th>
</tr>
</thead>
<tbody>
<tr>
<td>-74</td>
<td>1</td>
<td>1001010</td>
<td>7</td>
<td>11110</td>
<td>001010</td>
<td>111101001010</td>
</tr>
</tbody>
</table>

Figure 3.4 Huffman Encoding Example
The AC-coefficients are coded slightly differently using an 8-bit value represented as RRRRSSSS. The run length, 4-bit RRRR value, is the number of zeros preceding a non-zero value using the zigzag format of reading a matrix described earlier. The non-zero value is then coded by size, 4-bit SSSS value, as was described for the difference magnitude. Table 3.5 shows the possible combinations for the AC-coefficient coding.

<table>
<thead>
<tr>
<th>RRRR</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EOB</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>0A</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>1A</td>
</tr>
<tr>
<td>2</td>
<td>X</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
<td>29</td>
<td>2A</td>
</tr>
<tr>
<td>3</td>
<td>X</td>
<td>31</td>
<td>32</td>
<td>33</td>
<td>34</td>
<td>35</td>
<td>36</td>
<td>37</td>
<td>38</td>
<td>39</td>
<td>3A</td>
</tr>
<tr>
<td>4</td>
<td>X</td>
<td>41</td>
<td>42</td>
<td>43</td>
<td>44</td>
<td>45</td>
<td>46</td>
<td>47</td>
<td>48</td>
<td>49</td>
<td>4A</td>
</tr>
<tr>
<td>5</td>
<td>X</td>
<td>51</td>
<td>52</td>
<td>53</td>
<td>54</td>
<td>55</td>
<td>56</td>
<td>57</td>
<td>58</td>
<td>59</td>
<td>5A</td>
</tr>
<tr>
<td>6</td>
<td>X</td>
<td>61</td>
<td>62</td>
<td>63</td>
<td>64</td>
<td>65</td>
<td>66</td>
<td>67</td>
<td>68</td>
<td>69</td>
<td>6A</td>
</tr>
<tr>
<td>7</td>
<td>X</td>
<td>71</td>
<td>72</td>
<td>73</td>
<td>74</td>
<td>75</td>
<td>76</td>
<td>77</td>
<td>78</td>
<td>79</td>
<td>7A</td>
</tr>
<tr>
<td>8</td>
<td>X</td>
<td>81</td>
<td>82</td>
<td>83</td>
<td>84</td>
<td>85</td>
<td>86</td>
<td>87</td>
<td>88</td>
<td>89</td>
<td>8A</td>
</tr>
<tr>
<td>9</td>
<td>X</td>
<td>91</td>
<td>92</td>
<td>93</td>
<td>94</td>
<td>95</td>
<td>96</td>
<td>97</td>
<td>98</td>
<td>99</td>
<td>9A</td>
</tr>
<tr>
<td>10</td>
<td>X</td>
<td>A1</td>
<td>A2</td>
<td>A3</td>
<td>A4</td>
<td>A5</td>
<td>A6</td>
<td>A7</td>
<td>A8</td>
<td>A9</td>
<td>AA</td>
</tr>
<tr>
<td>11</td>
<td>X</td>
<td>B1</td>
<td>B2</td>
<td>B3</td>
<td>B4</td>
<td>B5</td>
<td>B6</td>
<td>B7</td>
<td>B8</td>
<td>B9</td>
<td>BA</td>
</tr>
<tr>
<td>12</td>
<td>X</td>
<td>C1</td>
<td>C2</td>
<td>C3</td>
<td>C4</td>
<td>C5</td>
<td>C6</td>
<td>C7</td>
<td>C8</td>
<td>C9</td>
<td>CA</td>
</tr>
<tr>
<td>13</td>
<td>X</td>
<td>D1</td>
<td>D2</td>
<td>D3</td>
<td>D4</td>
<td>D5</td>
<td>D6</td>
<td>D7</td>
<td>D8</td>
<td>D9</td>
<td>DA</td>
</tr>
<tr>
<td>14</td>
<td>X</td>
<td>E1</td>
<td>E2</td>
<td>E3</td>
<td>E4</td>
<td>E5</td>
<td>E6</td>
<td>E7</td>
<td>E8</td>
<td>E9</td>
<td>EA</td>
</tr>
<tr>
<td>15</td>
<td>X</td>
<td>F1</td>
<td>F2</td>
<td>F3</td>
<td>F4</td>
<td>F5</td>
<td>F6</td>
<td>F7</td>
<td>F8</td>
<td>F9</td>
<td>FA</td>
</tr>
</tbody>
</table>

Table 3.5 AC-Coefficient Coding [13]

There are two cases where the size of the value can be zero. This can occur when the run length has 16 zeros so the RRRRSSSS value will be F0 (ZRL). No more than 16 values can be coded together using this format. If there is a run of more than 16 zeros, then it must be split up into intervals of 16. The 00 (EOB) value is used when there are less than 16 values remaining in the block and they are all 0 [14]. The maximum size for an AC-coefficient is 10 bits plus a sign bit. These RRRRSSSS values are then encoded with a
Huffman table as was described for the difference magnitude. The value output is the Huffman code with the value, without the MSB, attached to the end of it.

### 3.2.1 Hardware Implementation

The Huffman encoding is done using a 5-stage pipeline. The Huffman lookup tables are included between the input and shift stages and the arbiter contains a buffer. Figure 3.5 describes the structure of the Huffman Encoder.

![Huffman Encoding Architecture](image)

**Figure 3.5 Huffman Encoding Architecture [16]**

The value from the quantization stage enters the input stage of the Huffman encoder in the appropriate zigzag ordering. Here the previous DC-coefficient is stored so that the DPCM can be used to determine the difference magnitude. This stage also collects the number of zeros coming in and determines the size of the non-zero value to get the AC-coefficient code. An address is determined to get the correct Huffman code from the look-up table. The input value minus the MSB is passed on to the next stage.

The Huffman lookup table is comprised of 256x21-bit entries. Sixteen bits are used to allow for the maximum size code and five bits are necessary to store the size of the code since the codes are of variable length. The address generated for AC-coefficients is just the RRRRSSSS value obtained for the zero-run and the size of the non-zero value. The difference magnitude Huffman codes are stored in the values not
used by the AC-coefficients (hexadecimal addresses: 0x0B-0x10, 0x1B-0x20). These addresses are determined by the input stage.

The shift stage receives the value from the input and the Huffman code from the table. This stage then shifts the values so that the MSB is in the leftmost position in the registers. These values are then sent on to the merge stage. The merge stage combines the Huffman code and the value into a 27-bit maximum value, 16 bits maximum for the code and 11 bits maximum for the sign bit and magnitude minus the MSB. This merged value is sent on to the arbiter.

The arbiter combines the merged value with the total bit count of the merged value into a 32-bit value. This value is stored in a buffer, which is necessary to help increase the throughput. The size of the buffer used is 128x32 bits. This large size buffer is not necessary for encoding; however, the decoding stage will share these buffers and they need to be 128 words in length. Therefore, the entire buffer is used since it is available.

The output stage receives data from the buffer through the arbiter. This stage outputs the coded image 8-bits at a time. Since the output stage could receive values larger than 8-bits in length from the arbiter, it needs to stall the pipeline until the entire value is sent out. The buffer allows the rest of the pipeline to continue undisturbed while the output stage takes multiple clock cycles to output the entire value from the arbiter. The buffer clears up when the input stage starts receiving sequences of zeros. This is because the input stage does not output something until a non-zero value or 16 zeros have been obtained. The output of the encoder will be the compressed version of the image video.
Chapter 4

Synthesizable HDL Model

4.1 Quantizer Unit

As mentioned earlier, the hardware implementation of the quantizer consists of two pieces: the data paths and the LUT. In this implementation, the LUT was modeled using Verilog case statements and constants providing all the required values in one clock cycle. The barrel shifter and multiplexer are implemented together as a single Verilog case statement. Two separate case statements are used because the multiplication factors are assigned based upon QP mod 6, while the dead zone compensation and shifting are based upon QP div by 6. To improve the performance of the LUTs, the results of QP div 6 and QP mod 6 are precalculated and built into the case statements.

4.1.1 Designware Pipelined Multiplier

The multiplication of the 16-bit Y value and the 14-bit MF is implemented using Wallace Tree Multiplier available as Synopsys’ Designware component. The block diagram of the Designware 6-stage pipelined multiplier is as shown in Figure 4.1 [15].

![Figure 4.1 Designware Pipelined Multiplier](15)
This multiplier DW02_mult_6_stage multiples the operand A by B to produce the product (PRODUCT) with a latency of five clock cycles [15].

4.1.2 Designware Adder

The multiplication of the 16-bit Y value and 14-bit MF yields a 30-bit result. This 30-bit result is broken into two 15-bit additions. Both the adders are implemented as this Designware Fast Carry Look Ahead Adder. The block diagram of the Designware adder is as shown in Figure 4.2 [15].

![Designware Adder](image_url)

**Figure 4.2 Designware Adder [15]**

This adder DW01_add adds 2 operands A and B with a carry-in CI to produce the output SUM with a carry-out CO.

4.1.3 Designware Incrementer

The second stage of the add and shift portion propagates the carry from the lower half to the upper half using a 16-bit incrementer circuit. This adder is implemented as this Designware Incrementer. The block diagram of the Designware Incrementer is as shown in Figure 4.3 [15].
4.2 Entropy Encoder Unit

4.2.1 huff_en.vhd

The encoder component does the Huffman encoding of the value coming in. The input values are from the quantization stage.

The output values are 8-bit values representing the encoded version of the image. The signals with “fifo” at the beginning go to the shared buffers. They consist of the read and write address and data lines and the write enable line. The hold_out signal is set high.
when the buffers fill up and the pipeline can not handle anymore data at the time. The huff_en block is broken up into a 5-stage pipeline.

\[\text{Figure 4.5. VHDL Architecture for huff_en Block [16]}\]

huff_en_input.vhd

The huff_en_input component takes the input values and generates the look-up address to get the appropriate Huffman code. To do the DPCM coding for the DC-coefficient, the previous 8x8 block DC-coefficient is stored here to generate the difference that is coded. The sequence of input values is always one DC-coefficient followed by 63 AC-coefficients, so this is kept track of with an internal counter. This component determines the size of the value and the number of preceding zeros that come in to generate the RRRRSSSSS value for the coding. For the difference magnitude, only the SSSS value is necessary. The RRRRSSSSS value becomes the address to the Huffman table. The addresses for the difference magnitude are encoded as described in Chapter 3. The output values consist of the original value, the size of the value, and the address to the table.
huff_en_tab1.vhd, huff_en_tab2.vhd, huff_en_tab3.vhd

Each of these tables is 256x8 bits in size and allow for 256x24 bits of data to be stored; however only 21 bits are necessary to store the Huffman codes. The maximum code value is 16 bits in size and the remaining 5 bits represent the size of the code (0-16). The huff_en_tab_1 block stores the sizes of the codes in the 5 least significant bits. The code is right justified in the 16 combined bits of huff_tab_2 and huff_tab_3.

huff_en_shift.vhd

The huff_en_shift component takes the input value and removes the MSB from it since that bit will be encoded in the Huffman code itself. It then left justifies both the value and the code in their respective fields. The value field is now 11 bits without the MSB and the code is 16 bits in size. The values and their sizes are sent to the next stage.
The `huff_en_merge` block takes the value and code and merges them into a maximum 27-bit value. The code is first followed by the value and it is left justified in the 27-bit field. The two count values are added and also sent to the output as a 5-bit value.

Figure 4.7. VHDL Architecture for huff_en_shift Block

Figure 4.8. VHDL Architecture for huff_en_merge Block
huff_en_arb.vhd

The huff_en_arb block does all the interactions with the buffer. It takes the value and size coming in and combines it into a 32-bit value where the first 5 bits are the size and the remaining 27 bits are the value.

The done_in signal must also be placed in the buffer so that it can be matched with the appropriate value. If the value size is 27 bits, then the five bits representing the size are encoded as 11111. When the size is less than 27 bits in size, the done_in signal is placed in the LSB of the value going to the buffer. When the buffer values are read, the done_out signal will be decoded from this format. Since the maximum size value is 27, the 11111 value will only be used when the size is 27 and done_in was set.

The arbiter stores the read and write address into the buffer. It determines whether the buffer is full with the use of a wrap bit. Initially, both addresses and wrap bits are set to 0. Whenever the address wraps around from the bottom of the buffer back to the top, the wrap bit is flipped. When the valid_in line goes high, the value_in lines are written to the buffer and the write address is incremented. The valid_out signal goes high when the addresses are different from each other indicating that there is data in the buffer. The value associated with the current read address is sent on to the next stage. If the hold_in signal from the next stage is set, no value is sent and the read address remains where it is while the buffer can still be filling up. The hold_out signal is set high when the buffer fills up. This is determined when both read and write addresses are equal but the wrap bits are different. The hold_out bit stalls the DCT, quantization, and the previously mentioned Huffman encoding components until the output stage can send out
the data to free up the buffer. When both read and write addresses and wrap bits are equal, then the buffer is empty and nothing is done.

![Diagram](image)

**Figure 4.9. VHDL Architecture for huff_en_arb Block**

huff_en_output.vhd

The huff_en_output block takes the values from the huff_en_arb component and outputs the values 8 bits at a time. Initially, the internal storage register and the register count are set to 0. A value is obtained from the arbiter along with the size. If the size is less than 8 bits, then it is stored in the internal register and the total count is updated with the size to be used in the next clock cycle. On the next clock cycle, the input is attached to the end of the value in the internal register through a shifter. If the sum is greater than 8, then 8 bits are output, the merged value removes the 8 bits output, and the remaining bits are placed in the internal register. The internal register is 27 bits in size and when it can not
accommodate anymore input data, it sets the hold_out signal high to the arbiter until it can clear out the register. This sequence continues on for the entire frame.

Figure 4.10. VHDL Architecture for huff_en_output Block

When the last value of the frame is to be sent, the done_in signal will be high. When this occurs, the hold_out signal is set high until the internal register and the input value are completely transmitted to signify the end of the frame. This is also the case where the block will transmit data when there are less then 8 bits to work with. All the bits of the last byte of the encoded data are not necessarily part of the actual data. The last byte of data will have the done_out signal set high to indicate the end of the image. Once the registers are cleared, the block can continue on with the next frame.
Chapter 5

Testing and Results

5.1 Quantizer Unit

5.1.1 Testing

A testbench was implemented for verifying the quantizer. To test the design, behavioral code was written to generate inputs for the quantizer. The inputs were fed to the pipelined RTL implementation as well as single cycle behavioral model. The outputs from these two implementations were then compared. In the case of mismatch, the bit of the 4-bit by 4-bit fail signal that corresponds to the failed output was set high. This allowed for easy identification of not only what cycle contained the error, but also which element in the output matrix was incorrect.

5.1.2 Synthesis

The quantizer unit was synthesized using Synopsys Design Compiler. It makes use of the Designware library components available with Synopsys. The worst case constraints were prioritized so that speed was the most important factor and after speed was maximized, the area was reduced adding additional constraints without affecting the speed of the circuit. The worst case constraints for the technology library used are shown below. The operating temperature is set to be $125^\circ C$, and the voltage is set to be 1.62 V.
Report: library
Library: ssc_core_slow
Version: X-2005.09
Date: Tue Dec 20 16:19:46 2005

Library Type: Technology
Comments: Operating condition (125.00 C, 1.62 V, slow)
Time Unit: 1ns

Capacitive Load Unit: 1.000000pf
Pulling Resistance Unit: 1kilo-ohm
Voltage Unit: 1V
Current Unit: 1mA
Dynamic Energy Unit: 1.000000pJ (derived from V,C units)

Operating Conditions:

Operating Condition Name: slow_125_1.62
Library: ssc_core_slow
Process: 1.00
Temperature: 125.00
Voltage: 1.62
Interconnect Model: balanced_tree

Operating Condition Name: slow_125_1.62_WCT
Library: ssc_core_slow
Process: 1.00
Temperature: 125.00
Voltage: 1.62
Interconnect Model: worst_case_tree

Input Voltages:

No input_voltage groups specified

Output Voltages:

No output_voltage groups specified

default_wire_load_capacitance: 0.000170
default_wire_load_resistance: 0.000271

A set_dont_touch attribute was applied on the design once optimized so that it will not be reoptimized when all the components will be put together for an ASIC.

Synthesis was performed in two steps. The first step consisted of a top down synthesis of the data path. In the synthesis runs, it was determined that the data path was significantly faster than the LUTs. To take advantage of this extra slack in the data paths,
an area constraint was applied to trade speed for area inside the data path. The second step then treated the sixteen data paths as a single unit and synthesized the top level including the LUTs. For both steps, the same design constraints were used to specify the clock period, clock uncertainty, operating conditions, and input and output constraints. The high power requirement of the quantizer block can be attributed to its high operating frequency of 309MHz, due to which the dynamic power dissipation increases drastically.

\[ P_{\text{dynamic}} = CVf^2 \]

where \( C \) is the load capacitance

\( V \) is the operating voltage

\( f \) is the frequency of operation

The synthesis results of both the data path and the quantizer unit are shown below.

Timing Report for quantizer_data_path:

*****************************************************************************
Report : timing
 -path full
 -delay max
 -max_paths 1
Design quant_data_path
Version: X-2005.09
Date : Sat Dec 10 15:26:23 2005
*****************************************************************************

Operating Conditions: slow_125_l.62 Library: ssc_core_slow
Wire Load Model Mode: enclosed

Startpoint: y_s8_reg[1]
 (rising edge-triggered flip-flop clocked by my_clock)
Endpoint: y_s9_reg[4]
 (rising edge-triggered flip-flop clocked by my_clock)
Path Group: my_clock
Path Type: max

<table>
<thead>
<tr>
<th>Des/Clust/Port</th>
<th>Wire Load Model</th>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>quant_data_path</td>
<td>10KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>increment_DW01_inc_1</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td></td>
<td>Incr</td>
<td>Path</td>
</tr>
<tr>
<td>---------------</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>clock my_clock (rise edge)</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>clock network delay (ideal)</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>y_s8_reg[1]/CLK (fdflc3)</td>
<td>0.00</td>
<td>0.00 r</td>
</tr>
<tr>
<td>y_s8_reg[1]/QN (fdflc3)</td>
<td>0.49</td>
<td>0.49 r</td>
</tr>
<tr>
<td>propagate/inl[1] (increment)</td>
<td>0.00</td>
<td>0.49 r</td>
</tr>
<tr>
<td>propagate/U1/A[1] (increment_DW01_inc_l)</td>
<td>0.00</td>
<td>0.49 r</td>
</tr>
<tr>
<td>propagate/U1/U128/Y (or2c6)</td>
<td>0.14</td>
<td>0.63 f</td>
</tr>
<tr>
<td>propagate/U1/U100/Y (and2c9)</td>
<td>0.18</td>
<td>0.81 r</td>
</tr>
<tr>
<td>propagate/U1/U79/Y (inv1a3)</td>
<td>0.17</td>
<td>0.98 f</td>
</tr>
<tr>
<td>propagate/U1/U101/Y (and2c3)</td>
<td>0.25</td>
<td>1.23 r</td>
</tr>
<tr>
<td>propagate/U1/U117/Y (and2a3)</td>
<td>0.26</td>
<td>1.49 r</td>
</tr>
<tr>
<td>propagate/U1/U84/Y (xor2a3)</td>
<td>0.30</td>
<td>1.79 f</td>
</tr>
<tr>
<td>propagate/U1/SUM[7] (increment_DW01_inc_l)</td>
<td>0.00</td>
<td>1.79 f</td>
</tr>
<tr>
<td>propagate/sum[7] (increment)</td>
<td>0.00</td>
<td>1.79 f</td>
</tr>
<tr>
<td>U575/Y (or2c1)</td>
<td>0.25</td>
<td>2.04 r</td>
</tr>
<tr>
<td>U564/Y (or3d1)</td>
<td>0.30</td>
<td>2.34 f</td>
</tr>
<tr>
<td>U682/Y (aol1f2)</td>
<td>0.32</td>
<td>2.66 r</td>
</tr>
<tr>
<td>U1054/Y (or3c2)</td>
<td>0.14</td>
<td>2.80 f</td>
</tr>
<tr>
<td>U681/Y (inv1a1)</td>
<td>0.18</td>
<td>2.98 r</td>
</tr>
<tr>
<td>y_s9_reg[4]/D (fdflc1)</td>
<td>0.00</td>
<td>2.98 r</td>
</tr>
</tbody>
</table>

Data arrival time: 2.98

<table>
<thead>
<tr>
<th></th>
<th>Incr</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock my_clock (rise edge)</td>
<td>3.23</td>
<td>3.23</td>
</tr>
<tr>
<td>clock network delay (ideal)</td>
<td>0.00</td>
<td>3.23</td>
</tr>
<tr>
<td>clock uncertainty</td>
<td>-0.10</td>
<td>3.13</td>
</tr>
<tr>
<td>y_s9_reg[4]/CLK (fdflc1)</td>
<td>0.00</td>
<td>3.13 r</td>
</tr>
<tr>
<td>library setup time</td>
<td>-0.15</td>
<td>2.98</td>
</tr>
<tr>
<td>data required time</td>
<td>2.98</td>
<td></td>
</tr>
</tbody>
</table>

Data required time: 2.98
Data arrival time: -2.98

Slack (MET): 0.00

Area Report of quantizer_data_path:

**********************************************************
Report : area
Design : quant_data_path
Version: X-2005.09
Date : Sat Dec 10 14:40:53 2005
**********************************************************

Library(s) Used:

ssc_core_slow (File: /home/sxk7568/eecc631/CHIP_2002.05/risc_design/core_slow.db)

Number of ports: 74
Number of nets: 767
Number of cells: 649
Number of references: 39

Combinational area: 19625.082031
Noncombinational area: 19572.253906
Net Interconnect area: undefined (Wire load has zero net area)

Total cell area: 39197 480469

Power Report of quantizer_data_path:

******************************************************************************
Report : power
-analysis_effort low
Design : quant_data_path
Version: X-2005.09
Date : Sat Dec 10 14:40:54 2005
******************************************************************************

Library(s) Used:

   ssc_core_slow (File: /home/sxk7568/eecc631/CHIP_2002.05/risc_design/core_slow.db)

Information: The cells in your design are not characterized for internal power (PWR-229)

Operating Conditions: slow_125_1.62 Library: ssc_core_slow
Wire Load Model Mode: enclosed

Design          Wire Load Model     Library
------------------------------------------
quant_data_path  5KGATES             ssc_core_slow
mult             5KGATES             ssc_core_slow
mult_DW02_mult_6_stage_0  5KGATES ssc_core_slow
add_1            5KGATES             ssc_core_slow
add_1_DW01_add_0  5KGATES             ssc_core_slow
add_0            5KGATES             ssc_core_slow
add_0_DW01_add_0  5KGATES             ssc_core_slow
increment        5KGATES             ssc_core_slow
increment_DW01_inc_0  5KGATES ssc_core_slow

Global Operating Voltage = 1.62
Power-specific unit information:
Voltage Units = 1V
Capacitance Units = 1.000000pf
Time Units = 1ns
Dynamic Power Units = 1mW (derived from V,C,T units)
Leakage Power Units = Unitless

Cell Internal Power = 0.0000 mW (0%)
Net Switching Power = 3.9156 mW (100%)
Total Dynamic Power = 3.9156 mW (100%)
Cell Leakage Power = 0.0000

Synthesis Results for Quantizer:

Timing Report for quantizer:

******************************************************************************
Report : timing
-path full
-delay max
-max_paths 1
Design : quantizer
Version: X-2005.09
Date : Sat Dec 10 15:11:26 2005
******************************************************************************

# A fanout number of 1000 was used for high fanout net computations.

Operating Conditions: slow_125_1.62 Library: ssc_core_slow
Wire Load Model Mode: enclosed

Startpoint: path21/QP_s8_reg[0]
  (rising edge-triggered flip-flop clocked by my_clock)
Endpoint: path21/y_s9_reg[1]
  (rising edge-triggered flip-flop clocked by my_clock)
Path Group: my_clock
Path Type: max

<table>
<thead>
<tr>
<th>Des/Clust/Port</th>
<th>Wire Load Model</th>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>quantizer</td>
<td>160K Gates</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>quant_data_path_6</td>
<td>10K Gates</td>
<td>ssc_core_slow</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Point</th>
<th>Incr</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock my_clock (rise edge)</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>clock network delay (ideal)</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>path21/QP_s8_reg[0]/CLK (fdflc3)</td>
<td>0.00</td>
<td>0.00 r</td>
</tr>
<tr>
<td>path21/QP_s8_reg[0]/QN (fdflc3)</td>
<td>0.57</td>
<td>0.57 r</td>
</tr>
<tr>
<td>path21/U1636/Y (clk1b3)</td>
<td>0.36</td>
<td>0.94 f</td>
</tr>
<tr>
<td>path21/U1302/Y (or2c9)</td>
<td>0.35</td>
<td>1.28 r</td>
</tr>
<tr>
<td>path21/U1668/Y (clk1b3)</td>
<td>0.31</td>
<td>1.59 f</td>
</tr>
<tr>
<td>path21/U1296/Y (or2c2)</td>
<td>0.23</td>
<td>1.82 r</td>
</tr>
<tr>
<td>path21/U1295/Y (or3d6)</td>
<td>0.13</td>
<td>1.94 f</td>
</tr>
<tr>
<td>path21/U1279/Y (and2c1)</td>
<td>0.44</td>
<td>2.38 r</td>
</tr>
<tr>
<td>path21/U1675/Y (or3c2)</td>
<td>0.26</td>
<td>2.64 r</td>
</tr>
<tr>
<td>path21/U1677/Y (mx2d2)</td>
<td>0.15</td>
<td>2.79 f</td>
</tr>
<tr>
<td>path21/U1412/Y (or3d1)</td>
<td>0.18</td>
<td>2.97 r</td>
</tr>
<tr>
<td>path21/y_s9_reg[1]/D (fdflc1)</td>
<td>0.00</td>
<td>2.97 r</td>
</tr>
</tbody>
</table>
data arrival time  2.97

clock my_clock (rise edge)  3.23  3.23
clock network delay (ideal)  0.00  3.23
clock uncertainty -0.10  3.13
path21/y_s9_reg[1]/CLK (dfdlc1)  0.00  3.13  r
library setup time  -0.16  2.97
data required time  2.97

-----------------------------------------------
data required time  2.97
data arrival time -2.97
-----------------------------------------------
slack (MET)  0.00

Area Report of quantizer:

******************************************************
Report : area
Design    quantizer
Version: X-2005.09
Date    Sat Dec 10 15:11:25 2005
******************************************************

Library(s) Used:

  ssc_core_slow (File: /home/sxk7568/eecc631/CHIP_2002.05/risc_design/core_slow.db)

Number of ports:  522
Number of nets:  888
Number of cells:  382
Number of references:  99

Combinational area:  347315.968750
Noncombinational area:  438004.531250
Net Interconnect area:  undefined (Wire load has zero net area)

Total cell area:  785275.312500

Power Report of quantizer:

******************************************************
Report : power
         -analysis_effort low
Design    quantizer
Version: X-2005.09
Date    Sat Dec 10 15:11:25 2005
******************************************************

Library(s) Used:
ssc_core_slow (File: /home/sxk7568/eecc631/CHIP_2002.05/risc_design/core_slow.db)

Information: The cells in your design are not characterized for internal power. (PWR-229)

Operating Conditions: slow_125_1.62 Library: ssc_core_slow
Wire Load Model Mode: enclosed

<table>
<thead>
<tr>
<th>Design</th>
<th>Wire Load Model</th>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>quantizer</td>
<td>160KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>quant_data_path_15</td>
<td>10KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>mult_15</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>mult_DW02_mult_6_stage_0_15</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>add_17</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>add_17_DW01_add_0</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>add_33</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>add_33_DW01_add_0</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>increment_15</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>increment_15_DW01_inc_0</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>quant_data_path_14</td>
<td>10KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>mult_14</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>mult_DW02_mult_6_stage_0_14</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>add_16</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>add_16_DW01_add_0</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>add_32</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>add_32_DW01_add_0</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>increment_14</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>increment_14_DW01_inc_0</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>quant_data_path_13</td>
<td>10KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>mult_13</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>mult_DW02_mult_6_stage_0_13</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>add_15</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>add_15_DW01_add_0</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>add_31</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>add_31_DW01_add_0</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>increment_13</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>increment_13_DW01_inc_0</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>quant_data_path_12</td>
<td>10KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>mult_12</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>mult_DW02_mult_6_stage_0_12</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>add_14</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>add_14_DW01_add_0</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>add_30</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>add_30_DW01_add_0</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>increment_12</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>increment_12_DW01_inc_0</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>quant_data_path_11</td>
<td>10KGATES</td>
<td>ssc_core_slow</td>
</tr>
</tbody>
</table>
| Operation                        | GATEs     | Module  
|---------------------------------|-----------|---------
| mult_11                         | 5KGATES   | ssc_core_slow  
| mult_DW02_mult_6_stage_0_11     | 5KGATES   | ssc_core_slow  
| add_13                          | 5KGATES   | ssc_core_slow  
| add_13_DW01_add_0               | 5KGATES   | ssc_core_slow  
| add_29                          | 5KGATES   | ssc_core_slow  
| add_29_DW01_add_0               | 5KGATES   | ssc_core_slow  
| increment_11                    | 5KGATES   | ssc_core_slow  
| increment_11_DW01_inc_0         | 5KGATES   | ssc_core_slow  
| quant_data_path_10              | 10KGATES  | ssc_core_slow  
| mult_10                         | 5KGATES   | ssc_core_slow  
| mult_DW02_mult_6_stage_0_10     | 5KGATES   | ssc_core_slow  
| add_12                          | 5KGATES   | ssc_core_slow  
| add_12_DW01_add_0               | 5KGATES   | ssc_core_slow  
| add_28                          | 5KGATES   | ssc_core_slow  
| add_28_DW01_add_0               | 5KGATES   | ssc_core_slow  
| increment_10                    | 5KGATES   | ssc_core_slow  
| increment_10_DW01_inc_0         | 5KGATES   | ssc_core_slow  
| quant_data_path_9               | 10KGATES  | ssc_core_slow  
| mult_9                          | 5KGATES   | ssc_core_slow  
| mult_DW02_mult_6_stage_0_9      | 5KGATES   | ssc_core_slow  
| add_11                          | 5KGATES   | ssc_core_slow  
| add_11_DW01_add_0               | 5KGATES   | ssc_core_slow  
| add_27                          | 5KGATES   | ssc_core_slow  
| add_27_DW01_add_0               | 5KGATES   | ssc_core_slow  
| increment_9                     | 5KGATES   | ssc_core_slow  
| increment_9_DW01_inc_0          | 5KGATES   | ssc_core_slow  
| quant_data_path_8               | 10KGATES  | ssc_core_slow  
| mult_8                          | 5KGATES   | ssc_core_slow  
| mult_DW02_mult_6_stage_0_8      | 5KGATES   | ssc_core_slow  
| add_10                          | 5KGATES   | ssc_core_slow  
| add_10_DW01_add_0               | 5KGATES   | ssc_core_slow  
| add_26                          | 5KGATES   | ssc_core_slow  
| add_26_DW01_add_0               | 5KGATES   | ssc_core_slow  
| increment_8                     | 5KGATES   | ssc_core_slow  
| increment_8_DW01_inc_0          | 5KGATES   | ssc_core_slow  
| quant_data_path_7               | 10KGATES  | ssc_core_slow  
| mult_7                          | 5KGATES   | ssc_core_slow  
| mult_DW02_mult_6_stage_0_7      | 5KGATES   | ssc_core_slow  
| add_9                           | 5KGATES   | ssc_core_slow  
| add_9_DW01_add_0                | 5KGATES   | ssc_core_slow  
| add_25                          | 5KGATES   | ssc_core_slow  
| add_25_DW01_add_0               | 5KGATES   | ssc_core_slow  
| increment_7                     | 5KGATES   | ssc_core_slow  
| increment_7_DW01_inc_0          | 5KGATES   | ssc_core_slow  
| quant_data_path_6               | 10KGATES  | ssc_core_slow  
| mult_6                          | 5KGATES   | ssc_core_slow  
| mult_DW02_mult_6_stage_0_6      | 5KGATES   | ssc_core_slow  
| add_8                           | 5KGATES   | ssc_core_slow  
| add_8_DW01_add_0                | 5KGATES   | ssc_core_slow  

62
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_24</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>add_24_DW01_add_0</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>increment_6</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>increment_6_DW01_inc_0</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>quant_data_path_5</td>
<td>10KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>mult_5</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>mult_DW02_mult_6_stage_0_5</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>add_7</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>add_7_DW01_add_0</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>add_23</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>add_23_DW01_add_0</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>increment_5</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>increment_5_DW01_inc_0</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>quant_data_path_4</td>
<td>10KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>mult_4</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>mult_DW02_mult_6_stage_0_4</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>add_6</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>add_6_DW01_add_0</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>add_22</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>add_22_DW01_add_0</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>increment_4</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>increment_4_DW01_inc_0</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>quant_data_path_3</td>
<td>10KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>mult_3</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>mult_DW02_mult_6_stage_0_3</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>add_5</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>add_5_DW01_add_0</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>add_21</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>add_21_DW01_add_0</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>increment_3</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>increment_3_DW01_inc_0</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>quant_data_path_2</td>
<td>10KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>mult_2</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>mult_DW02_mult_6_stage_0_2</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>add_4</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>add_4_DW01_add_0</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>add_20</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>add_20_DW01_add_0</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>increment_2</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>increment_2_DW01_inc_0</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>quant_data_path_1</td>
<td>10KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>mult_1</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>mult_DW02_mult_6_stage_0_1</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>add_3</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>add_3_DW01_add_0</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>add_19</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>add_19_DW01_add_0</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>increment_1</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>increment_1_DW01_inc_0</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>quant_data_path_0</td>
<td>10KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>mult_0</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>mult_0_DW02_mult_6_stage_1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Global Operating Voltage = 1.62
Power-specific unit information:
  Voltage Units = 1V
  Capacitance Units = 1.000000pf
  Time Units = 1ns
  Dynamic Power Units = 1mW (derived from V,C,T units)
  Leakage Power Units = Unitless

Cell Internal Power = 0.0000 mW (0%)
Net Switching Power = 88.5918 mW (100%)
Total Dynamic Power = 88.5918 mW (100%)

Results in tabular form:

<table>
<thead>
<tr>
<th>Component</th>
<th>Speed (MHz)</th>
<th>Area (# of gates)</th>
<th>Power (mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>quantizer</td>
<td>309</td>
<td>785275</td>
<td>88.5918</td>
</tr>
<tr>
<td>quantizer_data_path</td>
<td>309</td>
<td>39197</td>
<td>3.9156</td>
</tr>
</tbody>
</table>

Table 5.1 Quantizer Results
Figure 5.1 Netlist of quantizer Unit
Figure 5.2 Netlist of quantizer\_data\_path Unit
5.2 Entropy Encoder

5.2.1 Testing

The entropy encoder unit was tested by writing a test bench which provides input values to the unit under test and output values are observed to test the functionality. Random values and sequences of zeros were sent through the encoder to generate different codes. The output of the huffman encoder was checked to make sure that it outputs eight bits at a time without losing or adding bits. The arbiter and output blocks were checked to see that the hold_out signal was generated correctly indicating that the buffers were full. The data flow consistency was verified by changing the hold_in signal to fill up the buffers and check for the output values.

5.2.2 Synthesis

The entropy encoder was synthesized using Synopsys Design Compiler. It makes use of the GTECH library components available with Synopsys. This GTECH library is CMOS based and gives us good idea of the size of the chip if made into an ASIC. The constraints were prioritized so that speed was the most important factor and after speed was maximized, area was reduced adding additional constraints without affecting the speed of the circuit. A set_dont_touch attribute was applied on the design once optimized so that it will not be reoptimized when all the components will be put together for an ASIC.
The timing analysis showed that the worst case path was huff_en_input block. This component determined the size of the input value and generated the address to the lookup table to obtain the Huffman code to the output.

Some of the synthesized results of the Huffman encoder with their critical paths highlighted are shown below.

**Huffman_en Block:**

**Timing Report:**

```
****************************************
Report : timing
  -path full
  -delay max
  -max_paths 1
Design  huff_en
Version: X-2005.09
Date    : Mon Dec 12 10:27:45 2005
****************************************
```

**Operating Conditions:** slow_125_1.62    **Library:** ssc_core_slow

**Wire Load Model Mode:** enclosed

Startpoint: INP/huff_addr_reg[1]
            (rising edge-triggered flip-flop clocked by my_clock)

Endpoint:  SHIFT/huff_out_reg[6]
            (rising edge-triggered flip-flop clocked by my_clock)

Path Group: my_clock
Path Type: max

<table>
<thead>
<tr>
<th>Des/Clust/Port</th>
<th>Wire Load Model</th>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>huff_en</td>
<td>10KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>huff_en_tab_3</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>huff_en_shift</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Point</th>
<th>Incr</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock my_clock (rise edge)</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>clock network delay (ideal)</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>INP/huff_addr_reg[1]/CLK (fdef2a15)</td>
<td>0.00</td>
<td>0.00 r</td>
</tr>
<tr>
<td>INP/huff_addr_reg[1]/Q (fdef2a15)</td>
<td>0.74</td>
<td>0.74 r</td>
</tr>
<tr>
<td>INP/huff_addr[1] (huff_en_input)</td>
<td>0.00</td>
<td>0.74 r</td>
</tr>
<tr>
<td>HET3/address [1] (huff_en_tab_3)</td>
<td>0.00</td>
<td>0.74 r</td>
</tr>
<tr>
<td>HET3/U366/Y (clk1k3)</td>
<td>0.18</td>
<td>0.92 f</td>
</tr>
<tr>
<td>HET3/U319/Y (and2c6)</td>
<td>0.24</td>
<td>1.16 r</td>
</tr>
<tr>
<td>HET3/U529/Y (or3d2)</td>
<td>0.27</td>
<td>1.43 f</td>
</tr>
<tr>
<td>HET3/U341/Y (inv1a3)</td>
<td>0.42</td>
<td>1.85 r</td>
</tr>
<tr>
<td>HET3/U365/Y (and2c15)</td>
<td>0.11</td>
<td>1.96 f</td>
</tr>
<tr>
<td>HET3/U364/Y (or2c15)</td>
<td>0.14</td>
<td>2.10 r</td>
</tr>
</tbody>
</table>
HET3/U344/Y (inv1a3) 0.10 2.20 f
HET3/U447/Y (or3d3) 0.20 2.40 r
HET3/U122/Y (or2a3) 0.22 2.62 r
HET3/U534/Y (and2c3) 0.11 2.73 f
HET3/U70/Y (ao2i3) 0.30 3.02 r
HET3/U375/Y (oalf2) 0.10 3.12 f
HET3/U466/Y (or3d1) 0.27 3.39 r
HET3/value[6] (huff_en_tab_3) 0.00 3.39 r
SHIFT/huff_in[6] (huff_en_shift) 0.00 3.39 r
SHIFT/U223/Y (and2a3) 0.25 3.64 r
SHIFT/huff_out_reg[6]/D (fdef2a3) 0.00 3.64 r
data arrival time 3.64

clock my_clock (rise edge) 4.00 4.00
clock network delay (ideal) 0.00 4.00
clock uncertainty -0.10 3.90
SHIFT/huff_out_reg[6]/CLK (fdef2a3) 0.00 3.90 r
library setup time -0.26 3.64
data required time 3.64

--------------------------------------
data required time 3.64
data arrival time -3.64
--------------------------------------

slack (MET) 0.00

Area Report:

********************************************************************************
Report : area
Design : huff_en
Version: X-2005.09
Date : Mon Dec 12 10:27:45 2005
********************************************************************************

Library(s) Used:

ssc_core_slow (File:
/home/sxk7568/eecc631/CHIP_2002.05/risc_design/core_slow.db)

Number of ports: 108
Number of nets: 288
Number of cells: 34
Number of references: 19

Combinational area: 30764.642578
Noncombinational area: 17509.160156
Net Interconnect area: undefined (Wire load has zero net area)

Total cell area: 48274.058594
Power Report:

************************************************
Report : power
-analysis_effort low
Design : huff_en
Version: X-2005.09
Date Mon Dec 12 10:27:45 2005
************************************************

Library(s) Used:

ssc_core_slow (File: /home/sxk7568/eecc631/CHIP_2002.05/risc_design/core_slow.db)

Information: The cells in your design are not characterized for internal power. (PWR-229)

Operating Conditions: slow_125_1.62 Library: ssc_core_slow
Wire Load Model Mode: enclosed

<table>
<thead>
<tr>
<th>Design</th>
<th>Wire Load Model</th>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>huff_en</td>
<td>10KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>huff_en_tab_1</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>huff_en_tab_2</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>huff_en_tab_3</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>huff_en_input</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>huff_en_input_DW01_sub_4</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>huff_en_input_DW01_sub_5</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>huff_en_input_DW01_inc_0</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>huff_en_shift</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>huff_en_arb</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>huff_en_arb_DW01_inc_1</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>huff_en_arb_DW01_inc_0</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>huff_en_merge</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>huff_en_merge_DW01_add_0</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>huff_en_output</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>huff_en_output_DW01_add_1</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
</tbody>
</table>

Global Operating Voltage = 1.62
Power-specific unit information
Voltage Units = 1V
Capacitance Units = 1.000000pf
Time Units = 1ns
Dynamic Power Units = 1mW (derived from V,C,T units)
Leakage Power Units = Unitless
Cell Internal Power = 0.0000 mW (0%)
Net Switching Power = 2.6889 mW (100%)

Total Dynamic Power = 2.6889 mW (100%)
Cell Leakage Power = 0.0000

**Huff_en_arb Block:**

Timing Report:

******************************************************************************
Report: timing
-path full
-delay max
-max_paths 1
Design: huff_en_arb
Version: X-2005.09
Date: Mon Dec 12 10:25:05 2005
******************************************************************************

Operating Conditions: slow_125_1.62 Library: ssc_core_slow
Wire Load Model Mode: enclosed

Startpoint: reset (input port clocked by my_clock)
Endpoint: valid_rd_reg
    (rising edge-triggered flip-flop clocked by my_clock)
Path Group: my_clock
Path Type: max

<table>
<thead>
<tr>
<th>Des/Clust/Port</th>
<th>Wire Load Model</th>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>huff_en_arb</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Point</th>
<th>Incr</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock my_clock (rise edge)</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>clock network delay (ideal)</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>input external delay</td>
<td>0.60</td>
<td>0.60 r</td>
</tr>
<tr>
<td>reset (in)</td>
<td>0.12</td>
<td>0.72 r</td>
</tr>
<tr>
<td>U490/Y (buf1a9)</td>
<td>0.25</td>
<td>0.96 r</td>
</tr>
<tr>
<td>U405/Y (inv1a6)</td>
<td>0.17</td>
<td>1.13 f</td>
</tr>
<tr>
<td>U409/Y (buf1a9)</td>
<td>0.36</td>
<td>1.49 f</td>
</tr>
<tr>
<td>U413/Y (or2c1)</td>
<td>0.28</td>
<td>1.78 r</td>
</tr>
<tr>
<td>U478/Y (ao1f9)</td>
<td>0.36</td>
<td>2.14 f</td>
</tr>
<tr>
<td>U492/Y (or2c1)</td>
<td>0.24</td>
<td>2.38 r</td>
</tr>
<tr>
<td>U531/Y (ao1d2)</td>
<td>0.34</td>
<td>2.71 f</td>
</tr>
<tr>
<td>U317/Y (xor2b2)</td>
<td>0.32</td>
<td>3.03 r</td>
</tr>
<tr>
<td>U522/Y (or2c1)</td>
<td>0.18</td>
<td>3.22 f</td>
</tr>
<tr>
<td>U499/Y (and3d3)</td>
<td>0.25</td>
<td>3.47 r</td>
</tr>
<tr>
<td>U406/Y (and2a6)</td>
<td>0.18</td>
<td>3.65 r</td>
</tr>
<tr>
<td>U462/Y (or2c3)</td>
<td>0.09</td>
<td>3.75 f</td>
</tr>
<tr>
<td>valid_rd_reg/D (fdf2a3)</td>
<td>0.00</td>
<td>3.75 f</td>
</tr>
</tbody>
</table>
data arrival time 3.75

clock my_clock (rise edge) 4.00 4.00
clock network delay (ideal) 0.00 4.00
clock uncertainty -0.10 3.90
valid_rd_reg/CLK (fdf2a3) 0.00 3.90 r
library setup time -0.15 3.75
data required time 3.75

-----------------------------
data required time 3.75
data arrival time -3.75
-----------------------------
slack (MET) 0.00

Area Report:

******************************************************************************
Report : area
Design : huff_en_arb
Version: X-2005.09
Date : Mon Dec 12 10:25:05 2005
******************************************************************************

Library(s) Used:

 ssc_core_slow (File: /home/sxk7568/eecc631/CHIP_2002.05/risc_design/core_slow.db)

Number of ports: 151
Number of nets: 362
Number of cells: 281
Number of references: 44

Combinational area: 2800.044678
Noncombinational area: 5948.455566
Net Interconnect area: undefined (Wire load has zero net area)
Total cell area: 8748.490234

Power Report:

******************************************************************************
Report : power
-analysis_effort low
Design : huff_en_arb
Version: X-2005.09
Date : Mon Dec 12 10:25:05 2005
******************************************************************************

Library(s) Used:
ssc_core_slow (File: /home/sxk7568/eecc631/CHIP_2002.05/risc_design/core_slow.db)

Information: The cells in your design are not characterized for internal power. (PWR-229)

Operating Conditions: slow_125_1.62    Library: ssc_core_slow
Wire Load Model Mode: enclosed

<table>
<thead>
<tr>
<th>Design</th>
<th>Wire Load Model</th>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>huff_en_arb</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>huff_en_arb_DW01_inc_1</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>huff_en_arb_DW01_inc_0</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
</tbody>
</table>

Global Operating Voltage = 1.62
Power-specific unit information:
- Voltage Units = 1V
- Capacitance Units = 1.000000pf
- Time Units = 1ns
- Dynamic Power Units = 1mW  (derived from V,C,T units)
- Leakage Power Units = Unitless

Cell Internal Power = 0.0000 mW  (0%)
Net Switching Power = 666.9734 uW  (100%)
-------------------
Total Dynamic Power = 666.9734 uW  (100%)
Cell Leakage Power = 0.0000

Huff_en_input Block:

Timing Report:

*******************************************************************************
Report : timing
- path full
- delay max
- max_paths 1
Design   huff_en_input
Version: X-2005.09
Date : Mon Dec 12 10:23:49 2005
*******************************************************************************

Operating Conditions: slow_125_1.62    Library: ssc_core_slow
Wire Load Model Mode: enclosed

Startpoint: value_in[2]
            (input port clocked by my_clock)
Endpoint:  huff_addr_reg[2]
          (rising edge-triggered flip-flop clocked by my_clock)
Path Group: my_clock
Path Type: max

<table>
<thead>
<tr>
<th>Des/Clust/Port</th>
<th>Wire Load Model</th>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>huff_en_input</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>huff_en_input_DW01_sub_5</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
</tbody>
</table>

Point

<table>
<thead>
<tr>
<th>Path</th>
<th>Incr</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock my_clock (rise edge)</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>clock network delay (ideal)</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>input external delay</td>
<td>0.60</td>
<td>0.60 f</td>
</tr>
<tr>
<td>value_in[2] (in)</td>
<td>0.06</td>
<td>0.66 f</td>
</tr>
<tr>
<td>U702/Y (buf1a9)</td>
<td>0.25</td>
<td>0.90 f</td>
</tr>
<tr>
<td>sub_118/minus/B[2] (huff_en_input_DW01_sub_5)</td>
<td>0.00</td>
<td>0.90 f</td>
</tr>
<tr>
<td>sub_118/minus/U9/Y (clk1b2)</td>
<td>0.16</td>
<td>1.07 r</td>
</tr>
<tr>
<td>sub_118/minus/U137/Y (or2c6)</td>
<td>0.13</td>
<td>1.20 f</td>
</tr>
<tr>
<td>sub_118/minus/U136/Y (or2c6)</td>
<td>0.11</td>
<td>1.31 r</td>
</tr>
<tr>
<td>sub_118/minus/U58/Y (or3d3)</td>
<td>0.13</td>
<td>1.44 f</td>
</tr>
<tr>
<td>sub_118/minus/U29/Y (or2c3)</td>
<td>0.18</td>
<td>1.62 r</td>
</tr>
<tr>
<td>sub_118/minus/U135/Y (or3d6)</td>
<td>0.12</td>
<td>1.74 f</td>
</tr>
<tr>
<td>sub_118/minus/U132/Y (or3d6)</td>
<td>0.09</td>
<td>1.83 r</td>
</tr>
<tr>
<td>sub_118/minus/U123/Y (xor2b3)</td>
<td>0.26</td>
<td>2.09 f</td>
</tr>
<tr>
<td>sub_118/minus/DIFF[10] (huff_en_input_DW01_sub_5)</td>
<td>0.00</td>
<td>2.09 f</td>
</tr>
<tr>
<td>U729/Y (clk1b6)</td>
<td>0.12</td>
<td>2.20 r</td>
</tr>
<tr>
<td>U748/Y (oa4e3)</td>
<td>0.27</td>
<td>2.48 r</td>
</tr>
<tr>
<td>U941/Y (or2c6)</td>
<td>0.13</td>
<td>2.61 f</td>
</tr>
<tr>
<td>U954/Y (and3d6)</td>
<td>0.18</td>
<td>2.79 r</td>
</tr>
<tr>
<td>U835/Y (or3d6)</td>
<td>0.18</td>
<td>2.97 f</td>
</tr>
<tr>
<td>U957/Y (ao2e3)</td>
<td>0.29</td>
<td>3.26 r</td>
</tr>
<tr>
<td>U777/Y (inv1a6)</td>
<td>0.09</td>
<td>3.35 f</td>
</tr>
<tr>
<td>U613/Y (ao1d3)</td>
<td>0.19</td>
<td>3.54 f</td>
</tr>
<tr>
<td>U612/Y (or2c6)</td>
<td>0.11</td>
<td>3.64 r</td>
</tr>
<tr>
<td>huff_addr_reg[2]/D (fdef2a3)</td>
<td>0.00</td>
<td>3.64 r</td>
</tr>
</tbody>
</table>

data arrival time: 3.64

clock my_clock (rise edge): 4.00

clock network delay (ideal): 0.00

clock uncertainty: -0.10

huff_addr_reg[2]/CLK (fdef2a3): 0.00

library setup time: -0.26

data required time: 3.64

data required time: 3.64

data arrival time: 3.64

slack (MET): 0.00
Area Report:

************************************************
Report : area
Design : huff_en_input
Version: X-2005.09
Date : Mon Dec 12 10:23:49 2005
************************************************

Library(s) Used:

  ssc_core_slow (File: /home/sxk7568/eecc631/CHIP_2002.05/risc_design/core_slow.db)

Number of ports:  44  
Number of nets:    498 
Number of cells:  451  
Number of references: 102

Combinational area:  10700.830078 
Noncombinational area: 3457.140381 
Net Interconnect area: undefined (Wire load has zero net area)

Total cell area:  14157.979492

Power Report:

************************************************
Report : power
  -analysis_effort low
Design : huff_en_input
Version: X-2005.09
Date : Mon Dec 12 10:23:49 2005
************************************************

Library(s) Used:

  ssc_core_slow (File: /home/sxk7568/eecc631/CHIP_2002.05/risc_design/core_slow.db)

Information: The cells in your design are not characterized for internal power. (PWR-229)

Operating Conditions: slow_125_1.62  Library: ssc_core_slow
Wire Load Model Mode: enclosed

<table>
<thead>
<tr>
<th>Design</th>
<th>Wire Load Model</th>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>huff_en_input</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>huff_en_input_DW01_sub_5</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>huff_en_input_DW01_sub_4</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
</tbody>
</table>
Global Operating Voltage = 1.62

Power-specific unit information:
- Voltage Units = 1V
- Capacitance Units = 1.000000pf
- Time Units = 1ns
- Dynamic Power Units = 1mW (derived from V,C,T units)
- Leakage Power Units = Unitless

Cell Internal Power = 0.0000 mW (0%)
Net Switching Power = 1.4460 mW (100%)

Total Dynamic Power = 1.4460 mW (100%)
Cell Leakage Power = 0.0000

**Huff_en_merge Block:**

Timing Report:

************************************************
Report . timing
- path full
- delay max
- max_paths 1
Design : huff_en_merge
Version: X-2005.09
Date : Mon Dec 12 10:19:15 2005
************************************************

Operating Conditions: slow_125_l.62 Library: ssc_core_slow
Wire Load Model Mode: enclosed

Startpoint: huff_len[1]
  (input port clocked by my_clock)
Endpoint: value_out_reg[12]
  (rising edge-triggered flip-flop clocked by my_clock)
Path Group: my_clock
Path Type: max

<table>
<thead>
<tr>
<th>Des/Clust/Port</th>
<th>Wire Load Model</th>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>huff_en_merge</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Point</th>
<th>Incr</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock my_clock (rise edge)</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>clock network delay (ideal)</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>input external delay</td>
<td>0.60</td>
<td>0.60 r</td>
</tr>
</tbody>
</table>
huff_len[1] (in) 0.12 0.72 r
U449/Y (buf1a9) 0.25 0.97 r
U456/Y (inv1a3) 0.17 1.14 f
U405/Y (and2c6) 0.27 1.41 r
U461/Y (or2c3) 0.34 1.74 f
U516/Y (invla9) 0.41 2.15 r
U369/Y (oa4f3) 0.11 2.26 f
U494/Y (or2c1) 0.25 2.52 r
U613/Y (oa2i2) 0.13 2.65 f
U612/Y (oa1f3) 0.26 2.91 r
value_out_reg[12]/D (fdef2a2) 0.00 2.91 r
data arrival time 2.91

clock my_clock (rise edge) 3.30 3.30
clock network delay (ideal) 0.00 3.30
clock uncertainty -0.10 3.20
value_out_reg[12]/CLK (fdef2a2) 0.00 3.20 r
library setup time -0.28 2.92
data required time 2.92

--------------------------------------------
data required time 2.92
data arrival time -2.91
--------------------------------------------

slack (MET) 0.00

Area Report:

*****************************************************************************************
Report : area
Design : huff_en_merge
Version: X-2005.09
Date : Mon Dec 12 10:19:15 2005
*****************************************************************************************

Library(s) Used:

                              ssc_core_slow (File: /home/sxk7568/eecc631/CHIP_2002.05/risc_design/core_slow.db)

Number of ports:    75
Number of nets:     468
Number of cells:    421
Number of references: 59

Combinational area:  5669.749512
Noncombinational area:  2303.159912
Net Interconnect area:  undefined (Wire load has zero net area)

Total cell area:  7972.909668
Power Report:

****************************************
Report : power
    -analysis_effort low
Design : huff_en_merge
Version: X-2005.09
Date : Mon Dec 12 10:19:15 2005
****************************************

Library(s) Used:

    ssc_core_slow (File: /home/sxk7568/eecc631/CHIP_2002.05/risc_design/core_slow.db)

Information: The cells in your design are not characterized for internal power. (PWR-229)

Operating Conditions: slow_125_1.62 Library: ssc_core_slow
Wire Load Model Mode: enclosed

Design       Wire Load Model       Library
---------------------------------------
huff_en_merge  5KGATES       ssc_core_slow
huff_en_merge_DW01_add_0  5KGATES       ssc_core_slow

Global Operating Voltage = 1.62
Power-specific unit information.
    Voltage Units = 1V
    Capacitance Units = 1.000000pf
    Time Units = 1ns
    Dynamic Power Units = 1mW (derived from V,C,T units)
    Leakage Power Units = Unitless

    Cell Internal Power = 0.0000 mW (0%)
    Net Switching Power = 1.2707 mW (100%)
-------------------
    Total Dynamic Power = 1.2707 mW (100%)
    Cell Leakage Power = 0.0000
Huff_en_output Block:

Timing Report:

******************************************************************************
Report : timing
  -path full
  -delay max
  -max_paths 1
Design  huff_en_output
Version: X-2005.09
Date : Mon Dec 12 10:17:41 2005
******************************************************************************

Operating Conditions: slow_125_1.62  Library: ssc_core_slow
Wire Load Model Mode: enclosed

Startpoint: value_in_len[0]
  (input port clocked by my_clock)
  Path Group: my_clock
  Path Type: max

<table>
<thead>
<tr>
<th>Des/Clust/Port</th>
<th>Wire Load Model</th>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>huff_en_output</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>huff_en_output_DW01_add_l</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Point</th>
<th>Incr</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock my_clock (rise edge)</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>clock network delay (ideal)</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>input external delay</td>
<td>0.60</td>
<td>0.60 r</td>
</tr>
<tr>
<td>value_in_len[0] (in)</td>
<td>0.10</td>
<td>0.70 r</td>
</tr>
<tr>
<td>U572/Y (clk1a3)</td>
<td>0.23</td>
<td>0.93 r</td>
</tr>
<tr>
<td>add_154/plus/B[0] (huff_en_output_DW01_add_l)</td>
<td>0.00</td>
<td>0.93 r</td>
</tr>
<tr>
<td>add_154/plus/U36/Y (and2a3)</td>
<td>0.19</td>
<td>1.12 r</td>
</tr>
<tr>
<td>add_154/plus/U29/Y (or2c3)</td>
<td>0.12</td>
<td>1.24 f</td>
</tr>
<tr>
<td>add_154/plus/U5/Y (or3d6)</td>
<td>0.16</td>
<td>1.40 r</td>
</tr>
<tr>
<td>add_154/plus/U24/Y (oa1c9)</td>
<td>0.12</td>
<td>1.51 f</td>
</tr>
<tr>
<td>add_154/plus/U38/Y (aole6)</td>
<td>0.19</td>
<td>1.71 r</td>
</tr>
<tr>
<td>add_154/plus/U11/Y (inv1a3)</td>
<td>0.08</td>
<td>1.79 f</td>
</tr>
<tr>
<td>add_154/plus/U37/Y (aolf3)</td>
<td>0.18</td>
<td>1.96 r</td>
</tr>
<tr>
<td>add_154/plus/SUM[5] (huff_en_output_DW01_add_l)</td>
<td>0.00</td>
<td>1.96 r</td>
</tr>
<tr>
<td>U911/Y (clk1b3)</td>
<td>0.11</td>
<td>2.07 f</td>
</tr>
<tr>
<td>U623/Y (or2c6)</td>
<td>0.14</td>
<td>2.21 r</td>
</tr>
<tr>
<td>U621/Y (oa1a2)</td>
<td>0.20</td>
<td>2.41 r</td>
</tr>
<tr>
<td>U622/Y (and2a6)</td>
<td>0.24</td>
<td>2.65 r</td>
</tr>
<tr>
<td>U624/Y (or2c9)</td>
<td>0.11</td>
<td>2.76 f</td>
</tr>
<tr>
<td>U563/Y (inv1a9)</td>
<td>0.12</td>
<td>2.88 r</td>
</tr>
<tr>
<td>U638/Y (inv1a15)</td>
<td>0.14</td>
<td>3.02 f</td>
</tr>
<tr>
<td>U630/Y (clk1b3)</td>
<td>0.16</td>
<td>3.18 r</td>
</tr>
<tr>
<td>U565/Y (or2c9)</td>
<td>0.14</td>
<td>3.32 f</td>
</tr>
<tr>
<td>U932/Y (ao2i3)</td>
<td>0.30</td>
<td>3.62 r</td>
</tr>
</tbody>
</table>
Area Report:

***************************************************************
Report : area
Design : huff_en_output
Version: X-2005.09
Date : Mon Dec 12 10:17:41 2005
***************************************************************

Library(s) Used:

    ssc_core_slow (File: /home/sxk7568/eecc631/CHIP_2002.05/risc_design/core_slow.db)

Number of ports: 48
Number of nets: 511
Number of cells: 467
Number of references: 66

Combinational area: 5259.431152
Noncombinational area: 2980.510010
Net Interconnect area: undefined (Wire load has zero net area)

Total cell area: 8239.940430

Power Report:

***************************************************************
Report : power
    -analysis_effort low
Design : huff_en_output
Version: X-2005.09
Date : Mon Dec 12 10:17:41 2005
***************************************************************

Library(s) Used:
ssc_core_slow (File: /home/sxk7568/eecc631/CHIP_2002.05/risc_design/core_slow.db)

Information: The cells in your design are not characterized for internal power. (PWR-229)

Operating Conditions: slow_125 1.62 Library: ssc_core_slow
Wire Load Model Mode: enclosed

<table>
<thead>
<tr>
<th>Design</th>
<th>Wire Load Model</th>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>huff_en_output</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
<tr>
<td>huff_en_output_DW01_add_1</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
</tbody>
</table>

Global Operating Voltage = 1.62
Power-specific unit information:
- Voltage Units = 1V
- Capacitance Units = 1.000000pf
- Time Units = 1ns
- Dynamic Power Units = 1mW (derived from V,C,T units)
- Leakage Power Units = Unitless

Cell Internal Power = 0.0000 mW (0%)
Net Switching Power = 873.5944 uW (100%)
Total Dynamic Power = 873.5944 uW (100%)
Cell Leakage Power = 0.0000

**Huff_en_shift Block:**

Timing Report:

*******************************
Report : timing
- path full
- delay max
- max_paths 1
Design : huff_en_shift
Version: X-2005.09
Date : Mon Dec 12 10:14:59 2005
*******************************

Operating Conditions: slow_125 1.62 Library: ssc_core_slow
Wire Load Model Mode: enclosed

Startpoint: value_len[3]
  (input port clocked by my_clock)
Endpoint: value_out_reg[8]
  (rising edge-triggered flip-flop clocked by my_clock)
Path Group: my_clock
Path Type: max

<table>
<thead>
<tr>
<th>Des/Clust/Port</th>
<th>Wire Load Model</th>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>huff_en_shift</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Point</th>
<th>Incr</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock my_clock (rise edge)</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>clock network delay (ideal)</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>input external delay</td>
<td>0.60</td>
<td>0.60 r</td>
</tr>
<tr>
<td>value_len[3] (in)</td>
<td>0.12</td>
<td>0.72 r</td>
</tr>
<tr>
<td>U220/Y (clk1b15)</td>
<td>0.26</td>
<td>0.97 f</td>
</tr>
<tr>
<td>U260/Y (and2c9)</td>
<td>0.14</td>
<td>1.12 r</td>
</tr>
<tr>
<td>U284/Y (buf1a15)</td>
<td>0.17</td>
<td>1.29 r</td>
</tr>
<tr>
<td>U335/Y (or3d6)</td>
<td>0.21</td>
<td>1.50 f</td>
</tr>
<tr>
<td>U248/Y (ao4f3)</td>
<td>0.25</td>
<td>1.75 r</td>
</tr>
<tr>
<td>U282/Y (oa2i3)</td>
<td>0.11</td>
<td>1.86 f</td>
</tr>
<tr>
<td>U225/Y (oalc3)</td>
<td>0.27</td>
<td>2.12 r</td>
</tr>
<tr>
<td>value_out_reg[8]/D (fdef2a9)</td>
<td>0.00</td>
<td>2.12 r</td>
</tr>
<tr>
<td>data arrival time</td>
<td></td>
<td>2.12</td>
</tr>
</tbody>
</table>

| clock my_clock (rise edge)   | 2.50 | 2.50 |
| clock network delay (ideal)  | 0.00 | 2.50 |
| clock uncertainty            | -0.10| 2.40 |
| value_out_reg[8]/CLK (fdef2a9)| 0.00 | 2.40 r |
| library setup time           | -0.28| 2.12 |
| data required time            |      | 2.12 |

| data required time            |      | 2.12 |
| data arrival time             | -2.12|      |

slack (MET) 0.00

Area Report:

*****************************************************************************************
Report : area
Design : huff_en_shift
Version: X-2005.09
Date    : Mon Dec 12 10:14:58 2005
*****************************************************************************************

Library(s) Used:

ssc_core_slow (File: /home/sxk7568/eecc631/CHIP_2002.05/risc_design/core_slow.db)

<table>
<thead>
<tr>
<th>Number of ports:</th>
<th>80</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of nets:</td>
<td>214</td>
</tr>
<tr>
<td>Number of cells:</td>
<td>173</td>
</tr>
<tr>
<td>Number of references:</td>
<td>53</td>
</tr>
</tbody>
</table>
Combinational area: 2064.682861
Noncombinational area: 2611.739990
Net Interconnect area: undefined (Wire load has zero net area)
Total cell area: 4676.419922

Power Report:

************************************************************************************
Report : power
   -analysis_effort low
Design : huff_en_shift
Version: X-2005.09
Date : Mon Dec 12 10:14:59 2005
*************************************************************************************

Library(s) Used:

   ssc_core_slow (File: /home/sxk7568/eecc631/CHIP_2002.05/risc_design/core_slow.db)

Information: The cells in your design are not characterized for internal power. (PWR-229)

Operating Conditions: slow_125_1.62 Library: ssc_core_slow
Wire Load Model Mode: enclosed

Design       Wire Load Model       Library
---------------------------------------------
huff_en_shift  5KGATES            ssc_core_slow

Global Operating Voltage = 1.62
Power-specific unit information :
   Voltage Units = 1V
   Capacitance Units = 1.000000pf
   Time Units = 1ns
   Dynamic Power Units = 1mW (derived from V,C,T units)
   Leakage Power Units = Unitless

   Cell Internal Power = 0.0000 mW (0%)
   Net Switching Power = 798.1906 uW (100%)

   Total Dynamic Power = 798.1906 uW (100%)
   Cell Leakage Power = 0.0000
**Huffman_en_tab_1 Block:**

**Timing Report:**

```
****************************************
Report : timing
    -path full
    -delay max
    -max_paths 1
Design : huff_en_tab_1
Version: X-2005.09
Date : Fri Dec 9 21:13:15 2005
****************************************

Operating Conditions: slow_125_1.62   Library: ssc_core_slow
Wire Load Model Mode: enclosed
```

Startpoint: address[4] (input port)
Endpoint: value[4] (output port)
Path Group: (none)
Path Type: max

<table>
<thead>
<tr>
<th>Des/Clust/Port</th>
<th>Wire Load Model</th>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>huff_en_tab_1</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Point</th>
<th>Incr</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>input external delay</td>
<td>0.00</td>
<td>0.00 f</td>
</tr>
<tr>
<td>address[4] (in)</td>
<td>0.00</td>
<td>0.00 f</td>
</tr>
<tr>
<td>U620/Y (inv1a1)</td>
<td>0.28</td>
<td>0.28 r</td>
</tr>
<tr>
<td>U545/Y (and2c3)</td>
<td>0.25</td>
<td>0.54 f</td>
</tr>
<tr>
<td>U560/Y (and3d1)</td>
<td>0.50</td>
<td>1.03 r</td>
</tr>
<tr>
<td>U517/Y (aolf2)</td>
<td>0.24</td>
<td>1.28 f</td>
</tr>
<tr>
<td>U521/Y (and2c2)</td>
<td>0.30</td>
<td>1.58 r</td>
</tr>
<tr>
<td>U555/Y (ao2i1)</td>
<td>0.19</td>
<td>1.77 f</td>
</tr>
<tr>
<td>U554/Y (ao4a1)</td>
<td>0.40</td>
<td>2.17 f</td>
</tr>
<tr>
<td>U551/Y (oa2i1)</td>
<td>0.48</td>
<td>2.65 r</td>
</tr>
<tr>
<td>U550/Y (mx2a1)</td>
<td>0.31</td>
<td>2.96 r</td>
</tr>
<tr>
<td>U548/Y (or2c1)</td>
<td>0.13</td>
<td>3.09 f</td>
</tr>
<tr>
<td>value[4] (out)</td>
<td>0.00</td>
<td>3.09 f</td>
</tr>
<tr>
<td>data arrival time</td>
<td></td>
<td>3.09</td>
</tr>
</tbody>
</table>

(Path is unconstrained)
Area Report:

************************************************************************************
Report : area
Design : huff_en_tab_1
Version: X-2005.09
Date : Fri Dec  9 21:13:15 2005
************************************************************************************

Library(s) Used:

    ssc_core_slow (File: 
    /home/sxk7568/eecc631/CHIP_2002.05/risc_design/core_slow.db)

Number of ports: 16
Number of nets: 120
Number of cells: 111
Number of references: 31

Combinational area: 1242.010742
Noncombinational area: 0.000000
Net Interconnect area: undefined (Wire load has zero net area)

Total cell area: 1242.010010

Power Report:

************************************************************************************
Report : power
    -analysis_effort low
Design : huff_en_tab_1
Version: X-2005.09
Date : Fri Dec  9 21:13:15 2005
************************************************************************************

Library(s) Used:

    ssc_core_slow (File:
    /home/sxk7568/eecc631/CHIP_2002.05/risc_design/core_slow.db)

Information: The cells in your design are not characterized for internal power. (PWR-229)

Operating Conditions: slow_125_l.62 Library: ssc_core_slow
Wire Load Model Mode: enclosed

Design       Wire Load Model       Library
-----------------------------
huff_en_tab_1   5KGATES            ssc_core_slow

Global Operating Voltage = 1.62
Power-specific unit information :
Voltage Units = 1V
Capacitance Units = 1.000000pf
Time Units = 1ns
Dynamic Power Units = 1mW (derived from V,C,T units)
Leakage Power Units = Unitless

Cell Internal Power = 0.0000 mW (0%)
Net Switching Power = 734.1196 uW (100%)
Total Dynamic Power = 734.1196 uW (100%)
Cell Leakage Power = 0.0000

Huffman_en_tab2 Block:

Timing Report:

******************************************************************************************
Report : timing
-path full
-delay max
-max_paths 1
Design : huff_en_tab_2
Version: X-2005.09
Date : Fri Dec 9 21:17:05 2005
***************************************************************************************

Operating Conditions: slow_125_1.62   Library: ssc_core_slow
Wire Load Model Mode: enclosed

Startpoint: address[3] (input port)
Endpoint: value[0] (output port)
Path Group: (none)
Path Type: max

<table>
<thead>
<tr>
<th>Des/Clust/Port</th>
<th>Wire Load Model</th>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>huff_en_tab_2</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Point</th>
<th>Incr</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>input external delay</td>
<td>0.00</td>
<td>0.00 r</td>
</tr>
<tr>
<td>address[3] (in)</td>
<td>0.00</td>
<td>0.00 r</td>
</tr>
<tr>
<td>U175/Y (inv1a2)</td>
<td>0.15</td>
<td>0.15 f</td>
</tr>
<tr>
<td>U213/Y (or2c1)</td>
<td>0.51</td>
<td>0.66 r</td>
</tr>
<tr>
<td>U168/Y (inv1a2)</td>
<td>0.31</td>
<td>0.97 f</td>
</tr>
<tr>
<td>U207/Y (or2c1)</td>
<td>0.54</td>
<td>1.51 r</td>
</tr>
<tr>
<td>U196/Y (or3d1)</td>
<td>0.30</td>
<td>1.81 f</td>
</tr>
<tr>
<td>U195/Y (oa4e1)</td>
<td>0.38</td>
<td>2.19 r</td>
</tr>
<tr>
<td>U187/Y (ao2i1)</td>
<td>0.27</td>
<td>2.47 f</td>
</tr>
<tr>
<td>U186/Y (oa1f1)</td>
<td>0.54</td>
<td>3.01 r</td>
</tr>
<tr>
<td>U184/Y (or3d1)</td>
<td>0.43</td>
<td>3.44 f</td>
</tr>
</tbody>
</table>
U183/Y (ao2a1) 0.58 4.02 f
U181/Y (oa2i1) 0.45 4.47 r
U180/Y (ao2i1) 0.34 4.82 f
U178/Y (oa2i1) 0.49 5.31 r
U177/Y (ao2i1) 0.20 5.51 f
value[0] (out) 0.00 5.51 f
data arrival time 5.51

(Path is unconstrained)

Area Report:

*******************************************************************************
Report : area
Design : huff_en_tab_2
Version: X-2005.09
Date : Fri Dec  9 21:17:05 2005
*******************************************************************************

Library(s) Used:

ssc_core_slow (File: /home/sxk7568/eecc631/CHIP_2002.05/risc_design/core_slow.db)

Number of ports: 16
Number of nets: 55
Number of cells: 47
Number of references: 23

Combinational area: 541.920044
Noncombinational area: 0.000000
Net Interconnect area: undefined (Wire load has zero net area)

Total cell area: 541.919983

Power Report:

*******************************************************************************
Report : power
   -analysis_effort low
Design : huff_en_tab_2
Version: X-2005.09
Date : Fri Dec  9 21:17:05 2005
*******************************************************************************

Library(s) Used:

ssc_core_slow (File: /home/sxk7568/eecc631/CHIP_2002.05/risc_design/core_slow.db)
Information: The cells in your design are not characterized for internal power. (PWR-229)

Operating Conditions: slow_125_1.62 Library: ssc_core_slow
Wire Load Model Mode: enclosed

Design | Wire Load Model | Library
---|---|---
huff_en_tab_2 | 5KGATES | ssc_core_slow

Global Operating Voltage = 1.62
Power-specific unit information:
- Voltage Units = 1V
- Capacitance Units = 1.000000pf
- Time Units = 1ns
- Dynamic Power Units = 1mW (derived from V,C,T units)
- Leakage Power Units = Unitless

Cell Internal Power = 0.0000 mW (0%)
Net Switching Power = 344.0723 uW (100%)
Total Dynamic Power = 344.0723 uW (100%)
Cell Leakage Power = 0.0000

**Huffman_en_tab_3 Block:**

Timing Report:

****************************************
Report: timing
-path full
-delay max
-max_paths 1
Design: huff_en_tab_3
Version: X-2005.09
Date: Fri Dec 9 21:19:45 2005
****************************************

Operating Conditions: slow_125_1.62 Library: ssc_core_slow
Wire Load Model Mode: enclosed

Startpoint: address[1] (input port)
Endpoint: value[5] (output port)
Path Group: (none)
Path Type: max

Des/Clust/Port | Wire Load Model | Library
---|---|---
huff_en_tab_3 | 5KGATES | ssc_core_slow
<table>
<thead>
<tr>
<th>Point</th>
<th>Incr</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>input external delay</td>
<td>0.00</td>
<td>0.00 f</td>
</tr>
<tr>
<td>address[1] (in)</td>
<td>0.00</td>
<td>0.00 f</td>
</tr>
<tr>
<td>U393/Y (clkb3)</td>
<td>0.30</td>
<td>0.30 r</td>
</tr>
<tr>
<td>U392/Y (and2c2)</td>
<td>0.29</td>
<td>0.59 f</td>
</tr>
<tr>
<td>U331/Y (or3d2)</td>
<td>0.33</td>
<td>0.92 r</td>
</tr>
<tr>
<td>U309/Y (or2c3)</td>
<td>0.19</td>
<td>1.11 f</td>
</tr>
<tr>
<td>U308/Y (and2c3)</td>
<td>0.22</td>
<td>1.33 r</td>
</tr>
<tr>
<td>U468/Y (inv1a1)</td>
<td>0.18</td>
<td>1.51 f</td>
</tr>
<tr>
<td>U373/Y (and2c3)</td>
<td>0.37</td>
<td>1.88 r</td>
</tr>
<tr>
<td>U372/Y (and2a3)</td>
<td>0.32</td>
<td>2.20 r</td>
</tr>
<tr>
<td>U448/Y (and2b1)</td>
<td>0.41</td>
<td>2.61 r</td>
</tr>
<tr>
<td>U443/Y (oa1f1)</td>
<td>0.18</td>
<td>2.79 f</td>
</tr>
<tr>
<td>U338/Y (oa2i6)</td>
<td>0.64</td>
<td>3.43 r</td>
</tr>
<tr>
<td>U439/Y (ao2i1)</td>
<td>0.15</td>
<td>3.58 f</td>
</tr>
<tr>
<td>U438/Y (inv1a1)</td>
<td>0.35</td>
<td>3.92 r</td>
</tr>
<tr>
<td>U366/Y (ao2i2)</td>
<td>0.29</td>
<td>4.21 f</td>
</tr>
<tr>
<td>U417/Y (or2b1)</td>
<td>0.26</td>
<td>4.47 f</td>
</tr>
<tr>
<td>U414/Y (oa2i1)</td>
<td>0.44</td>
<td>4.91 r</td>
</tr>
<tr>
<td>U413/Y (or3d1)</td>
<td>0.21</td>
<td>5.12 f</td>
</tr>
<tr>
<td>value[5] (out)</td>
<td>0.00</td>
<td>5.12 f</td>
</tr>
<tr>
<td>data arrival time</td>
<td></td>
<td>5.12</td>
</tr>
</tbody>
</table>

(Path is unconstrained)

Area Report:

****************************************
Report : area
Design : huff_en_tab_3
Version: X-2005.09
Date : Fri Dec 9 21:19:44 2005
****************************************

Library(s) Used:

    ssc_core_slow (File: /home/sxk7568/eecc631/CHIP_2002.05/risc_design/core_slow.db)

Number of ports: 16
Number of nets: 243
Number of cells: 235
Number of references: 35

Combinational area: 2669.594482
Noncombinational area: 0.000000
Net Interconnect area: undefined (Wire load has zero net area)

Total cell area: 2669.590088
Power Report:

*****************************************************************************
Report : power
   -analysis_effort low
Design : huff_en_tab_3
Version: X-2005.09
Date : Fri Dec 9 21:19:45 2005
*****************************************************************************

Library(s) Used:

   ssc_core_slow (File: /home/sxk7568/eecc631/CHIP_2002.05/risc_design/core_slow.db)

Information: The cells in your design are not characterized for internal power. (PWR-229)

Operating Conditions: slow_125_1.62   Library: ssc_core_slow
Wire Load Model Mode: enclosed

<table>
<thead>
<tr>
<th>Design</th>
<th>Wire Load Model</th>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>huff_en_tab_3</td>
<td>5KGATES</td>
<td>ssc_core_slow</td>
</tr>
</tbody>
</table>

Global Operating Voltage = 1.62
Power-specific unit information :
   Voltage Units = 1V
   Capacitance Units = 1.000000pf
   Time Units = 1ns
   Dynamic Power Units = 1mW   (derived from V,C,T units)
   Leakage Power Units = Unitless

   Cell Internal Power = 0.0000 mW   (0%)
   Net Switching Power = 1.5007 mW   (100%)
-------------------
Total Dynamic Power = 1.5007 mW   (100%)
   Cell Leakage Power = 0.0000
The above results are summarized in the table below:

<table>
<thead>
<tr>
<th>Component</th>
<th>Speed (MHz)</th>
<th>Area (# of gates)</th>
<th>Power (mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>huff_en</td>
<td>250</td>
<td>48274</td>
<td>2.6889</td>
</tr>
<tr>
<td>huff_en_arb</td>
<td>250</td>
<td>8748</td>
<td>0.6667</td>
</tr>
<tr>
<td>huff_en_input</td>
<td>250</td>
<td>14157</td>
<td>1.4460</td>
</tr>
<tr>
<td>huff_en_merge</td>
<td>303</td>
<td>7972</td>
<td>1.2707</td>
</tr>
<tr>
<td>huff_en_output</td>
<td>250</td>
<td>8239</td>
<td>0.8735</td>
</tr>
<tr>
<td>huff_en_shift</td>
<td>400</td>
<td>4676</td>
<td>0.7981</td>
</tr>
<tr>
<td>huff_en_tab_1</td>
<td>250</td>
<td>1242</td>
<td>0.7341</td>
</tr>
<tr>
<td>huff_en_tab_2</td>
<td>250</td>
<td>541</td>
<td>0.3440</td>
</tr>
<tr>
<td>huff_en_tab_3</td>
<td>250</td>
<td>2669</td>
<td>1.5007</td>
</tr>
</tbody>
</table>

**Table 5.2 Entropy Encoder Results**

The following figures show the gate level netlist of each of the components of encoder.
Figure 5.3 Netlist of huff-en Unit
Figure 5.4 Netlist of huff_en_arb Unit
Figure 5.5 Netlist of huff_en_input Unit
Figure 5.6 Netlist of huff_en_merge Unit
Figure 5.7 Netlist of huff_en_output Unit
Figure 5.8 Netlist of huff_en_shift Unit
Chapter 6

Conclusion

6.1 Conclusion

Integrated multimedia systems process text, graphics, and other discrete media such as digital audio and video streams. In an uncompressed state, graphics, audio and video data, especially moving pictures, require large transmission and storage capacities which can be very expensive. Hence video compression has become a key component of any multimedia system or application. The H.264/AVC video coding standard was developed as a joint effort by both the ITU-T VCEG and ISO/IEC MPEG. The H.264/AVC uses significantly improved and computationally intensive compression techniques to maximize performance and aims to meet the needs of practical multimedia communication applications. The success of practical implementation of H.264/AVC depends on careful design of the CODEC and effective choices of coding parameters.

The Quantizer and the Entropy Encoder blocks have been designed and successfully implemented as an ASIC. These two blocks, the Quantizer, and the Entropy Encoder implement the Baseline Profile of the H.264/AVC standard. The architecture is implemented in Register Transfer Level HDL and synthesized with Synopsys Design Compiler using TSMC 0.25μm technology, giving us an estimate of the hardware requirements in real-time implementation. The quantizer uses DWARE components from Synopsys standard library for optimizing the speed and the entropy encoder uses GTECH library available with Synopsys. Both the blocks were designed as pipelined structures which helped in optimizing the operating speed of the blocks. The quantizer
block is capable of running at 309MHz and has a total area of 785K gates with a power requirement of 88.59mW. The entropy encoder unit is capable of running at 250 MHz and has a total area of 49K gates with a power requirement of 2.68mW. These blocks can be integrated with other building blocks of H.264/AVC encoder to implement an ASIC, used for real-time video compression. It gives us a clear idea of the hardware complexity, the operating speed, and the power requirements in real-time implementation. The high speed that is achieved in this thesis simply indicates that the two blocks Quantizer and Entropy Encoder can be used as IP embedded in the HDTV systems.

6.2 Suggestions for Improvement

In the design of quantizer, the current implementation of the LUTs uses basic combinational logic and constants. A ROM implementation would be significantly more efficient. A pipelined implementation of the LUTs along with relaxed area requirements on data path could increase the speed to a certain extent. This quantizer can produce one output per cycle and has a latency of nine clock cycles; the latency could be reduced to eight clock cycles if the LUT can be implemented in parallel with the last stage of the transform block. Finally when operating in 2x2 DC chroma mode, twelve of the sixteen data paths are unused. In the current implementation, these paths continue running, but their outputs are simply ignored. Significant power savings could be achieved by gating off the clocks to these data paths and leaving them idle when not in use.
In the design of entropy encoder, it will be more beneficial to be able to store different Huffman tables. Secondly the use of DWARE components would have significantly improved the speed of the circuit.

The operating speed of both the designs can be improved by shrinking the process. The current design uses TSMC 0.25\(\mu\) technology.

6.3 Future Work

This thesis is easily expandable. There is a scope to improve the Huffman Coding scheme to Context Adaptive Variable Length Coding (CAVLC). This would be beneficial in terms of effective coding. Once the entropy encoder is replaced with CAVLC, both these units can be used as building blocks of real-time H.264/AVC Main Profile encoder design. This design can be expanded to implement the Main and Extended Profiles of H.264/AVC encoder.
Bibliography


