A RECONFIGURABLE APPROXIMATE JPEG ENCODER IMPLEMENTED ON FPGA PLATFORM

A Thesis

by

SHRIJA

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Jiang Hu
Co-Chair of Committee,	Duncan M. Walker
Committee Member,	Peng Li
Head of Department,	Miroslav M. Begovic

December 2016

Major Subject: Computer Engineering

ABSTRACT

Approximate computing is a blooming field of research, which involves a compromise of an application's accuracy to make it more efficient. It necessarily involves a deliberate effort to make an application imprecise in order to conserve some resource. While the techniques for approximation may include using approximate functional units, approximate storage hardware, approximate network communication or executing algorithms with varying precision, in this thesis, we mainly aim to use approximate arithmetic units in a JPEG Encoder Core.

Using imprecise arithmetic units requires an identification of the applications which are resilient to errors. JPEG compression is inherently a lossy compression as the degradation in image quality isn't perceptible to the end users for a significant compression ratio. This tolerance of JPEG compression to a loss of data is exploited to replace the accurate computationally intensive blocks of the JPEG encoding algorithm with their imprecise counterparts. The approximate hardware design proposed in this thesis is also reconfigurable, which implies that the end user can select the level of approximation desired based on the loss in accuracy allowed.

The reconfigurable approximate JPEG hardware is also implemented on the FPGA platform. The FPGA emulation of the approximate and the precise versions of the JPEG Encoder core is done on the Altera DE1-SoC with Cyclone V device and an ARM based Hard Processor System. The emulation provided a maximum area reduction by 45%, maximum delay reduction by 29% and a maximum power reduction by 28% for an approximation of 8 least significant bits. For approximation of more than 8 least significant bits, the peak-signal-to-noise ratio of the decompressed image is very low, resulting in poor image quality.

DEDICATION

To my Parents Mrs.& Mr. Sinha, Sneha and Vishal.

ACKNOWLEDGEMENTS

I would like to express my gratitude to my thesis advisor and committee chair Dr. Jiang Hu for his continuous guidance, untiring support and immense knowledge and for keeping me motivated throughout the last two years. I also wish to thank Dr. Duncan Walker and Dr. Peng Li for serving on my committee and for their meticulous feedback on my work.

I express my deepest appreciation for the faculty at Texas A&M, who have helped me develop a strong knowledge base and acquire the required skill set for my research through the coursework. I also thank my family, who have always been a source of encouragement and have inspired me to follow my dreams. Last but not the least, I want to thank all my friends, who have been a constant support and have kept me smiling through this entire process.

NOMENCLATURE

FPGA	Field Programmable Gate Array
DSP	Digital Signal Processing
DCT	Discrete Cosine Transform
RLE	Run Length Encoding
IDCT	Inverse Discrete Cosine Transform
SoC	System on Chip
ARM	Advanced RISC Machine
HPS	Hard Processor System
BRAM	Block Random Access Memory
ROM	Read Only Memory
RTL	Register Transfer Level
HDL	Hardware Description Language
RCA	Ripple Carry Adder
CLA	Carry Lookahead Adder
GDA	Gracefully Degrading Accuracy Configurable Adder
ACA	Accuracy Configurable Adder
ALM	Adaptive Logic Module
JTAG	Joint Test Action Group

TABLE OF CONTENTS

Page

ABSTRACT . . . ii

DEDICATION . . . iii

ACKNOWLEDGEMENTS . . . iv

NOMENCLATURE . . . v

TABLE OF CONTENTS . . . vi

LIST OF FIGURES . . . viii

LIST OF TABLES . . . x

1. INTRODUCTION . . . 1

   1.1 Motivation Behind Approximate Computing . . . 2
   1.2 Issues and Limitations of Approximate Computing . . . 3
   1.3 Design Overview . . . 4

2. PREVIOUS WORK . . . 6

   2.1 Approximate Arithmetic Units . . . 6
      2.1.1 Approximate Adders . . . 7
      2.1.2 Approximate Multipliers . . . 8

3. BACKGROUND . . . 9

   3.1 JPEG Compression . . . 9
      3.1.1 Color Space Transformation . . . 11
      3.1.2 Discrete Cosine Transform . . . 11
      3.1.3 Quantization . . . 12
      3.1.4 Entropy Coding . . . 13
   3.2 FPGA . . . 15
   3.3 Project Description . . . 16
      3.3.1 Standard Cell Library . . . 16
      3.3.2 Softwares Used . . . 16
      3.3.3 Approach . . . 17

4. APPROXIMATE ARITHMETIC UNIT DESIGN 21

4.1 Approximate Adder Design 21
4.2 Approximate Multiplier Design 22
4.3 Experimental Results 23

5. APPROXIMATE JPEG HARDWARE DESIGN 26

5.1 Approximate DCT Design 26
5.2 Approximate Quantizer Design 26
5.3 Approximate Encoder Design 27
5.4 Experimental Results 27

6. FPGA IMPLEMENTATION 31

6.1 FPGA Specifications 31
6.2 IP Cores 32
6.2.1 General Purpose Input-Output 32
6.2.2 Block RAM 33
6.3 Implementation Details 33
6.3.1 Image Data 34
6.3.2 System Schematic 34
6.3.3 Programming the FPGA 35
6.3.4 Bitstream Capture 36
6.4 System Validation 36
6.4.1 Functional Verification 36
6.4.2 Area Reduction 37
6.4.3 Power Saving 37
6.4.4 Delay Reduction 38
6.4.5 Clock Speed 39
6.4.6 PSNR Degradation 40
6.4.7 Approximation with Different Compression Ratios 41

7. CONCLUSION AND FUTURE WORK 43

REFERENCES 44

LIST OF FIGURES

FIGURE Page

3.1 Block Diagram of JPEG Encoder 10

3.2 Comparison of Original and Reconstructed Image 10

3.3 Horizontal and Vertical Frequencies of 2D-DCT 13

3.4 Zigzag Sequencing of the Image Components 14

3.5 Altera DE1-SoC Board with Cyclone V FPGA 15

3.6 Block Diagram of Approximate JPEG Encoder 18

4.1 Approximate Adder Design 22

4.2 Approximate Multiplier Design 23

4.3 Power Delay Product of Approximate Arithmetic Units Vs. Number of Approximate LSBs 25

5.1 Percentage Delay Reduction Vs. Number of Approximate LSBs . . . 29

5.2 Percentage Power Saving Vs. Number of Approximate LSBs 29

5.3 Percentage Area Reduction Vs. Number of Approximate LSBs 30

6.1 Internal Block Diagram of the DE1-SoC Board 31

6.2 Block Diagram of the GPIO IP Core 32

6.3 IP Blocks of Single and Dual Port RAM 33

6.4 Flow Diagram of FPGA Implementation 35

6.5 Schematic of the FPGA Implementation 36

6.6 Simulation Results 37

6.7 Block Diagram of an ALM 38

6.8 Plot of the Comparison Metrics Vs. Number of Approximated LSBs . 40

LIST OF TABLES

TABLE — Page

4.1 Comparison of Accurate and Approximate Adder Design 24

4.2 Comparison of Accurate and Approximate Multiplier Design 24

5.1 Comparison of Accurate and Approximate DCT Blocks 28

5.2 Comparison of Accurate and Approximate Quantizer Blocks 28

5.3 Comparison of Accurate and Approximate Encoder Blocks 28

6.1 Area Comparison of Accurate and Approximate Designs 38

6.2 Power Comparison of Accurate and Approximate Designs 39

6.3 Delay Comparison of Accurate and Approximate Designs 39

6.4 Clock Speed Comparison of Accurate and Approximate Designs . . . 40

6.5 Comparison of PSNR Degradation for Accurate and Approximate Designs 41

6.6 Comparison of PSNR Degradation with the No. of Approximate LSBs for Different Compression Ratios 42

1. INTRODUCTION

In today's world, extensive applications like social media, scientific computing etc. are witnessing a rapid growth with respect to user demands. Most of these applications are computationally intensive and have an enormous storage requirement. It is expected that in the coming years, the rate of increase in the amount of information being processed might far exceed the rate of increase of the available resources. As a result, the disposition of the required resources would not be possible. With the increasing demands for performance and the scarcity of the available resources, approximate computing is an encouraging solution to the inevitable dilemma of balancing the performance requirement and the meagre resources.

By exploring the applications that are tolerant to error and don't have strict requirements of accurate results, approximate computing can be employed to facilitate resource savings. However, this also implies that suitable quality metrics should be used to estimate the degradation in quality weighed against the improvement in system parameters. Only a level of accuracy can be compromised to get an improvement in the efficiency of the application. This necessitates the critical evaluation of the errors introduced in the application due to approximation. So, approximate computing is a multi dimensional area of research and each and every aspect has to be given its due importance.

This section is a short introduction to the work presented in this thesis. Initially, the need for the growth of approximate computing as a field of research is presented, followed by the challenges faced in developing an approximate system. The last section describes a brief overview of the design implemented in this thesis.

1.1 Motivation Behind Approximate Computing

The motivation for approximate computing comes from the fact that in many applications, like, multimedia, signal processing, machine learning, etc., the amount of accuracy being provided by the computing system is more than the amount of accuracy required by the user. Hence, instead of performing exact computations and using excessive resources, it is possible to introduce a small approximation and save resources, while maintaining the quality of service at the acceptable level.

Many of the real time applications have an intrinsic scope of approximation. Some applications are prone to noisy input or frequent breakdown of the hardware. Yet others have unequal loading at different points of time. All these applications present an alluring platform to use approximate computing. In these applications, loss of accuracy is anyways unavoidable. Using approximation, however, facilitates resource saving at the cost of imprecise results.

There are some applications, which are resilient to loss of accuracy due to the inability of the humans to percieve the difference between an accurate and an inaccurate computation. Such applications, like multimedia applications, have a lot of scope for approximation. Even if the results are inaccurate, human senses might not be able to percieve the difference and the quality of service might be acceptable. Most of these applications use computationally intensive DSP blocks, hence, using approximate hardware would provide a lot of resource savings.

Another property of the applications which provides a scope for approximation is redundancy. If the results of the application are correlated, either with respect to time or with respect to space, approximation can be used to increase the efficiency. Memoization is a well known technique which stores the results of an expensive computation or function call to be returned later if the same input vectors are applied.

Other software approaches can also be used to optimize the application.

Some applications don't have a single correct result but multiple correct results are returned, for example, in search engines. Other applications can save a lot of resource by employing precision scaling and skipping some iterations of the loop. All these applications provide the motivation for approximate computing.

1.2 Issues and Limitations of Approximate Computing

Approximate computing helps save resources, however, it also comes with some challenges. There are issues which are to be addressed before the practicability of any application employing approximate computing.

The first limitation of approximate computing is that not all applications have a scope of approximation. Some applications, like, real time tracking systems, high accuracy measurement systems, etc. have a demand for high accuracy, as a result, using approximate computations might be catastrophic. Also, not all approximation strategies provide a good range of approximation. Some strategies have a very limited range of approximation and hence cannot be used in many applications.

Using approximate computing also necessitates the use of proper quality metrics to weigh the loss of accuracy introduced. If suitable quality metrics aren't used, there is a chance that the output quality may be degraded beyond the acceptable level, however, it isn't sampled properly due to the use of an improper metric. For example, an approximate image processing hardware might give a bad image quality if only the power consumption and not the PSNR is used as a metric to measure the performance.

Another challenge is that using approximation may introduce a lot of overhead. For example, the approximate reconfigurable adder proposed by [27] has a smaller delay as compared to the accurate adder, however, the extra combinational logic in the design to achieve a smaller delay leads to an overhead in area and power consumption. The overhead may also be in the form of the amount of effort required to design an approximate system.

Also, the strategies used for approximation are very specific to the type of application they can be used on. Hence, a lot of background work is required to determine the suitable technique for approximation. Using an approximate version of an application would also require a continuous monitoring of the output quality. The errors introduced due to approximation, hence, should be within acceptable bounds, else, the output would fall below the acceptable level of degradation and would result in poor quality of service. So, a lot of factors have to be kept in mind before developing an application with approximation.

1.3 Design Overview

As mentioned above, the error-resilience of applications provides a scope for approximation. The work presented in this thesis aims at using the error-resilience of JPEG compression algorithm to develop a reconfigurable approximate hardware for JPEG compression, which can be reconfigured for different levels of approximation. The reconfigurable hardware is also implemented on FPGA platform.

JPEG compression is a lossy compression algorithm, and is hence resilient to a loss of accuracy. As proposed in [24], a PSNR value of 30dB or greater is considered acceptable. The most computationally intensive step in JPEG compression is the Discrete Cosine Transform (DCT) , which expresses data points as the sum of cosine functions oscillating at varying fundamental frequencies. The DCT step uses a large number of arithmetic units to perform the computations. After DCT, the next step is quantization, which aims at mapping a large set of data to smaller set, thus reducing the file size. The Quantizer block in JPEG compression also requires a lot of arithmetic units.

The idea presented in this thesis is to use approximate arithmetic units instead of the accurate ones in the DCT and the Quantizer blocks of JPEG compression and hence reduce the delay of the compression as well as have power and area savings. Also, the approximate arithmetic units, namely, the approximate adder and approximate multiplier are made reconfigurable, which implies that the user can choose the level of approximation and the degradation in the output quality that can be tolerated.

The approximate arithmetic units introduce some error in the least significant bits of the result, as the least significant bits carry lesser weight than the most significant bits. So, small errors in the LSBs can be introduced without causing much difference in the result. The adders and the multipliers are reconfigurable in the sense that the number of LSBs of the result that would be computed approximately can be configured by the user based on the level of output degradation allowed or the required amount of reduction in delay.

The reconfigurable hardware designed is then implemented on the Altera DE1-SoC FPGA. Different versions with different levels of approximation are implemented and the output is verified. The FPGA emulation also validates the reduction in latency, increase in the clock frequency and the savings in the area and power of the approximate version as compared to the accurate version. The FPGA implementation of the approximate version can be used as a platform for characterization of new approximate arithmetic units as well as for further research in using approximate computing to optimize performance.

2. PREVIOUS WORK

The earliest works in approximate computing can be traced back to the Floating Point computations, where the results are frequently rounded-off to reduce the storage costs while compromising the accuracy. Since that time, numerous works have been done to develop an approximate system to trade-off accuracy for the improvement of some other parameter.

The works on approximate computing can be broadly grouped into two catagories, namely software approaches to approximate computing and hardware approaches to approximate computing. The salient techniques for approximation used by the software approaches are executing an imprecise version of an algorithm in the program, altering the precision of the intermediate computations to reduce storage cost, reducing computational cost by skipping some loop iterations, memoization and using many imprecise versions of a program. The techniques used for hardware approaches to approximation, as dicussed in [22] are mainly using faulty or inaccurate hardware, using voltage scaling, approximate memory structures, skipping memory accesses, approximations in neural networks and SIMD architectures, etc. The work presented in [1] partitions the application into error-resilient and error-sensitive parts and then uses approximation models for the error-resilient part of the application.

Our research is limited to the use of approximate arithmetic units, so, in the rest of this section, the previous works on the approximate arithmetic units is discussed.

2.1 Approximate Arithmetic Units

A number of works have proposed using approximate arithmetic units, namely, adders and multipliers, in place of the accurate ones in computationally intensive applications. As the least significant bits carry less weight than the most significant bits, it is feasible to introduce approximation in the least significant bits of the result. The following subsections list some prior works on approximate adders and multipliers respectively.

2.1.1 Approximate Adders

One of the earliest works in approximate adder design, [16], comprises of the reduction in error introduced in a Ripple Carry Adder due to voltage scaling. Four different voltages are supplied to the RCA to reduce the switching power, while making sure that the critical path isn't violated. The adder proposed by [26] deterministically produces inaccurate results for specific input vectors. The work presented in [5] introduces various configurations of imprecise multi bit adders comprising full adder cells, namely simplified Mirror Adder (MA), MA Approximation 1, MA Approximation 2 and MA Approximation 3, which have a simpler transistor level schematic, and hence have lesser delay and significant power and area savings with a negligible loss of accuracy. With the advent of approximate as well as probabilistic adders, new metrics were required to prove the potency of these adders. [18] proposes Mean Error Distance (MED), Normalised Error Distance (NED) and Sequential Probability Transition Matrices (SPTMs) as the new metrics to evaluate the approximate and probabilistic adders. A logic to reduce the errors due to approximation, known as the conditional bound logic for the least significant bits is proposed in [21]. Another work on imprecise adders with reduced transistor level complexity is [6], which proposes five approximate versions of the conventional Mirror Adder.

While most of the works listed above have a fixed bit-width approximation, there are some works on reconfigurable approximate adders as well. [15] proposes an accuracy configurable adder, which can operate in both precise and approximate modes owing to the facility of configuring the accuracy of the results at runtime.

[27] also proposes a gracefully degrading accuracy configurable adder, which can be configured at runtime. A critical evaluation of all the approximate adders is discussed in [14], while [8] presents a comprenensive evaluation of the energy efficiency of approximate computing paradigm.

2.1.2 Approximate Multipliers

The work proposed in [7] introduces an approximate multiplier design by using approximate adders which have shorter critical paths as their carry propagation is limited to a single stage. The approximate multiplier has lesser power consumption and shorter delay and also has an error recovery mechanism. The approximate multiplier proposed by [19] is a modified Wallace Tree Multiplier with reduced number of partial product stages. Another approximate multiplier for low power DSP applications is the one proposed by [4], which uses a broken booth multiplier. The proposed multiplier replaces all the least significant bits of the partial products upto the vertical breaking level to zero.

The above section outlines the prior work done in the approximate arithmetic unit design. However, there are many more facets to approximate computing than just circuit design. [23] lays an emphasis on some meta functions that are inherently error resilient and hence are fluently scalable under voltage over-scaling when implemented in hardware. Approximate computing has been widely adopted in memory architecture as well, like in [17], the fidelity of the stored data in an SRAM is traded off for a reduction in power consumed to retain the data .

3. BACKGROUND

This chapter contains some relevant background which is needed to understand the work presented in this thesis. The main focus of this thesis is to develop a platform for reconfigurable approximate application. The platform used is the Altera De1-SoC FPGA and the application inculcating approximation is the JPEG Encoder. The section 3.1 covers a description of the JPEG Compression algorithm and the major steps involved in the encoding scheme. The next section gives a brief description of the FPGA platform and the IP cores used in the implementation of the approximate hardware. The methodology of the project has been discussed in the last section.

The HDL used in the description of the JPEG hardware is *Verilog* [12]. *Verilog* is a hardware description language, standardized as IEEE 1364 and is used in the modelling and verification of electronic circuits. The description of the JPEG Encoder core [20] is done at the register transfer level of abstraction. This level of abstraction models the circuits in terms of the flow of data between the hardware registers and the logical and arithmetic operations performed on the data. The description of the hardware in *Verilog* is synthesizable, which means that the design can be physically realized by using some synthesis softwares.

3.1 JPEG Compression

JPEG stands for *Joint Photographic Expert Group*, which is the name of the committee that created JPEG as well as other still picture coding standards. The first JPEG standard was issued by the committee in the year 1992. The block diagram of the baseline JPEG Encoder is shown in Figure 3.1. The rest of this section explains the various blocks of the JPEG Compression.

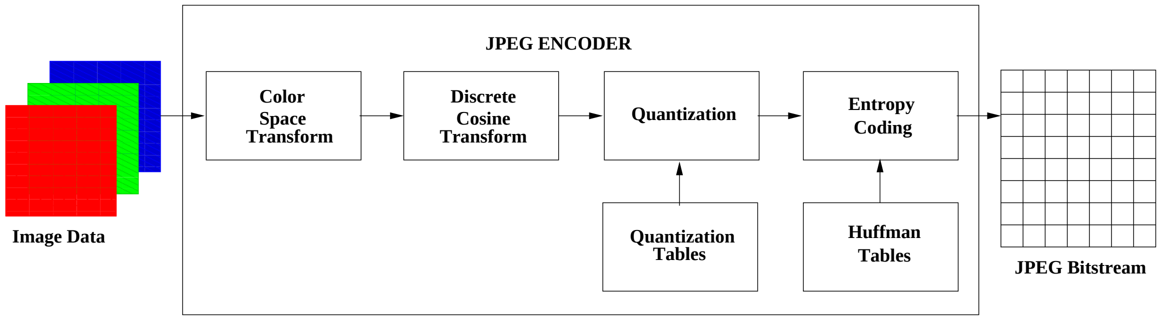
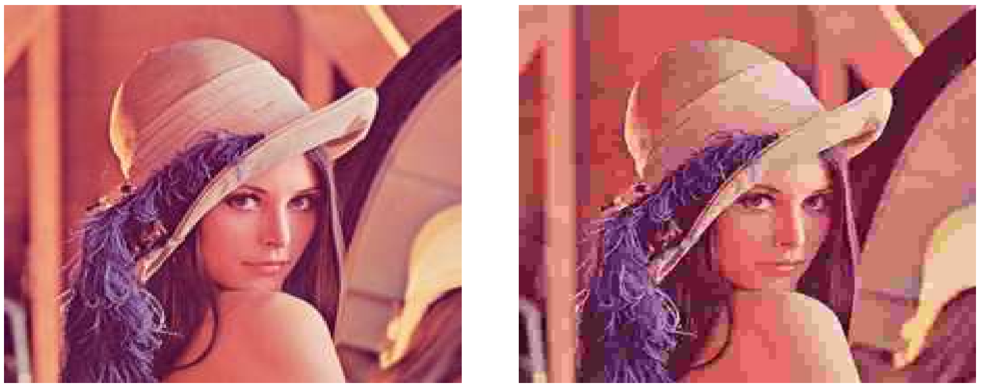


Figure 3.1: Block Diagram of JPEG Encoder

JPEG Compression is inherently lossy in nature, which implies that there is some loss of pixel data in the compression process and the reconstructed image doesn't have the same pixel values as the original image. As a result, JPEG can't be used in the images in which a sharp contrast between the neighboring pixels can change the image information. JPEG should also not be used in cases where an exact reproduction of images is required. The Figure 3.2 shows the comparison between the original and the reconstructed images.



(a) Original Image *(b) Reconstructed Image*

Figure 3.2: Comparison of Original and Reconstructed Image

3.1.1 Color Space Transformation

The first step in the JPEG compression algorithm is the color space transformation. In this step, the image is transformed from the RGB color space to a different color space, called the YCbCr space. The Y component represents the brightness of the pixels and the Cb and Cr components represents the chrominance, which is the color information of a picture. This transformation is done as it provides greater image compression without a significant loss in the perceptual image quality. The brightness information is confined to a single channel, which makes the compression more efficient as the image quality is governed by the brightness information. After the conversion to YCbCr space, the spatial resolution of the Cb and Cr components is reduced, in a process called "Downsampling" or "Chroma Sampling".

3.1.2 Discrete Cosine Transform

After the color space transform, the image is broken to blocks of 8X8 pixels and each 8X8 block of each component is then transformed into frequency domain representation, by using normalized 2D-Discrete Cosine Transformation. The DCT operation transforms the data points to a sum of cosine signals of varying fundamental frequencies. The DCT is used in the compression algorithm as it has a very strong energy compaction property, which means that it redistributes the signal energy into a very small number of coefficients, owing to the way its basis function is formed. The following formula is used for 2D-DCT:

$$X_k = \sum_{n=0}^{N-1} x_n cos(\frac{\pi}{N}(n + \frac{1}{2})k) \tag{3.1}$$

Prior to the computation of DCT, each 8X8 block has to be centered around 0. Since each data point lies in the interval [0,255], 128 has to be subtracted from each data point to center the values around 0. For the 8X8 matrix, the DCT is performed by the following formula:

$$DY = T * Y * T^{-1} \tag{3.2}$$

where, T is the DCT matrix and T^{-1} is its inverse. Y is the matrix representation of the 8x8 pixel block. DY is the resultant matrix after the 2D DCT. The top-left entry of the 8X8 block is the DC coefficient and it represents the basic hue of the entire block. The rest of the 63 coefficients are the AC coefficients. The DCT stage confines most of the signal to one corner of the result. This stage also increases the bit-depth of the data, however, this isn't a major concern, since only a small subset of the image data is in the DCT stage at a given time. The Figure 3.3 shows the combination of the horizontal and vertical frequencies of a 2D-DCT.

3.1.3 Quantization

Quantization is the next step in the JPEG encoding algorithm. The basis of this step is the inability of the human eye to percieve the exact strength of a high frequency brightness variation. This encourages the reduction of the signal strength of the high frequency components in the image matrix. In this step, each high frequency component is divided by a particular constant that is specific to that component. The division makes the value smaller and then the result is rounded off to the nearest integer. Most of the values are reduced to zero, and the rest are reduced to small positive integers. This step is a lossy step, as once a value has been rounded off to an integer or reduced to zero, it can't be reverted to the original value. Hence, some of the image information is lost in this step. Both the previous steps, the DCT and the color transformation are completely reversible and the data can be fully recovered. The quantization step makes the compression algorithm lossy in

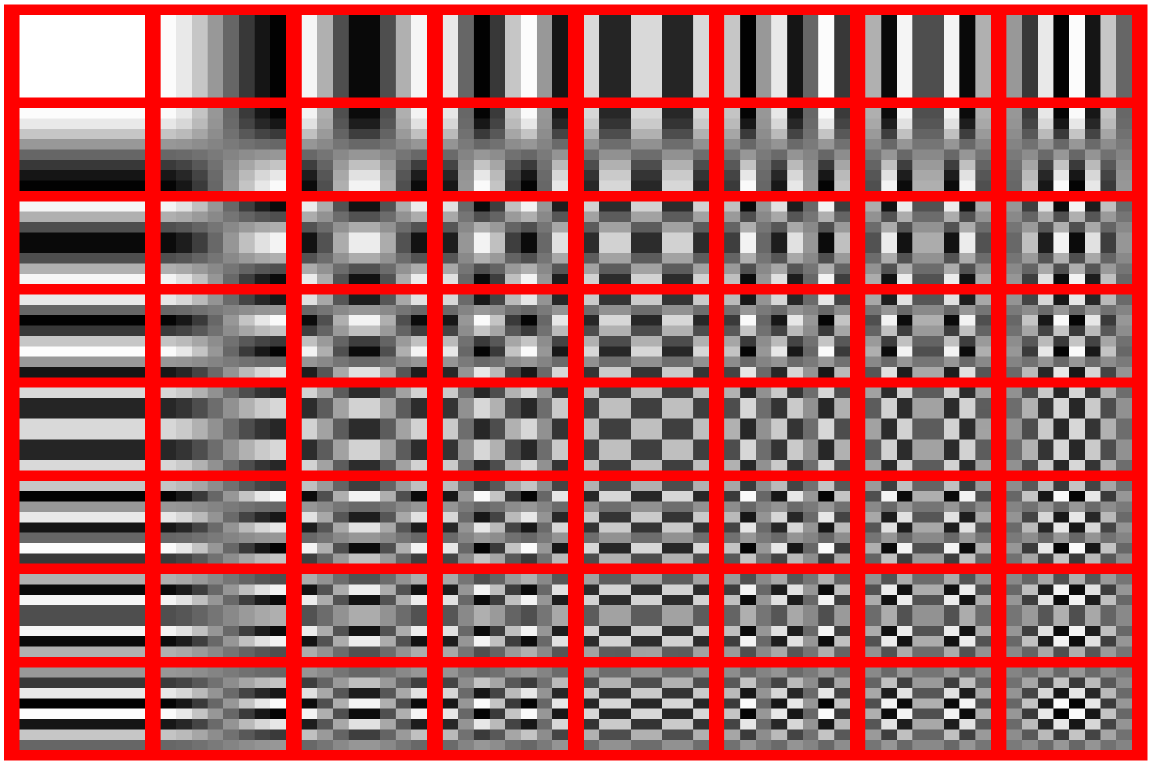


Figure 3.3: Horizontal and Vertical Frequencies of 2D-DCT

nature. The quantization step makes the entropy coding easier as well, since, the greater the recurrence of the data, the easier it is to encode it.

3.1.4 Entropy Coding

The final step of the JPEG compression is the Entropy Coding step, which is again a lossless compression step. The entropy coding step employs zigzag sequencing of the image components, followed by the run length encoding, after which, the data is compressed by applying Huffman Coding.

3.1.4.1 Zigzag Sequencing

This step involves arranging the image components in a zigzag sequence. The figure 3.4 shows the sequence in which the image components are arranged.

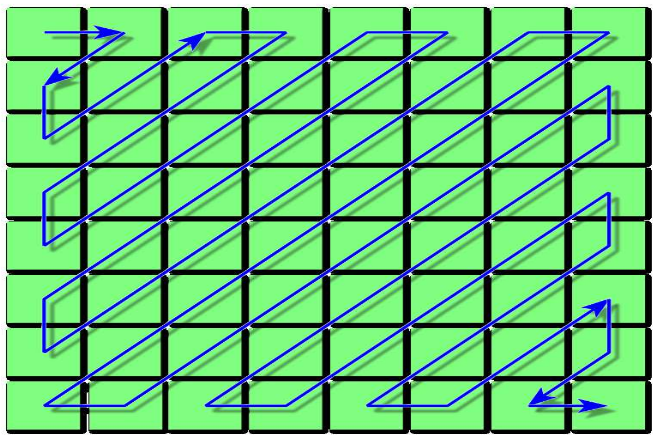


Figure 3.4: Zigzag Sequencing of the Image Components

3.1.4.2 Run Length Encoding

Run Length Encoding is a form of encoding in which the similar data are grouped together. In this process, the number of similar data points are counted and then the group of similar data points are stored just as a single data and the count value.

3.1.4.3 Huffman Coding

Huffman Coding step involves the use of Huffman tables, which is derived from the probability and the frequency of occurence of each value. It creates a binary tree of nodes, along with their probabilities. In the subsequent stages, two nodes of smallest frequency are grouped together, until only a single node is left, which is the Huffman Tree. This stage produces the encoded JPEG bitstream.

The JPEG decoder in turn inverses all the above stages to finally produce the image pixel data in the RGB space. However, some image data is lost in the compression process, so, the actual image data can't be recovered.

3.2 FPGA

FPGA stands for Field Programmable Gate Array. It is an integrated circuit, which can be configured by the user to implement the design of their choice. The system description is done in an HDL, either *Verilog* or *VHDL*. The FPGA used in the hardware implementation for this thesis is the Altera DE1-SoC board [11] with Cyclone V device [2] and Arm cortex-9 dual core hard processor system. Figure 3.5 shows the picture of how the DE1-SoC board looks like.

The FPGA is fabricated on a 28 nm technology and has a large number of well defined hard IP cores. All the peripherals and the connectors of the FPGA is labelled in the figure 3.5.

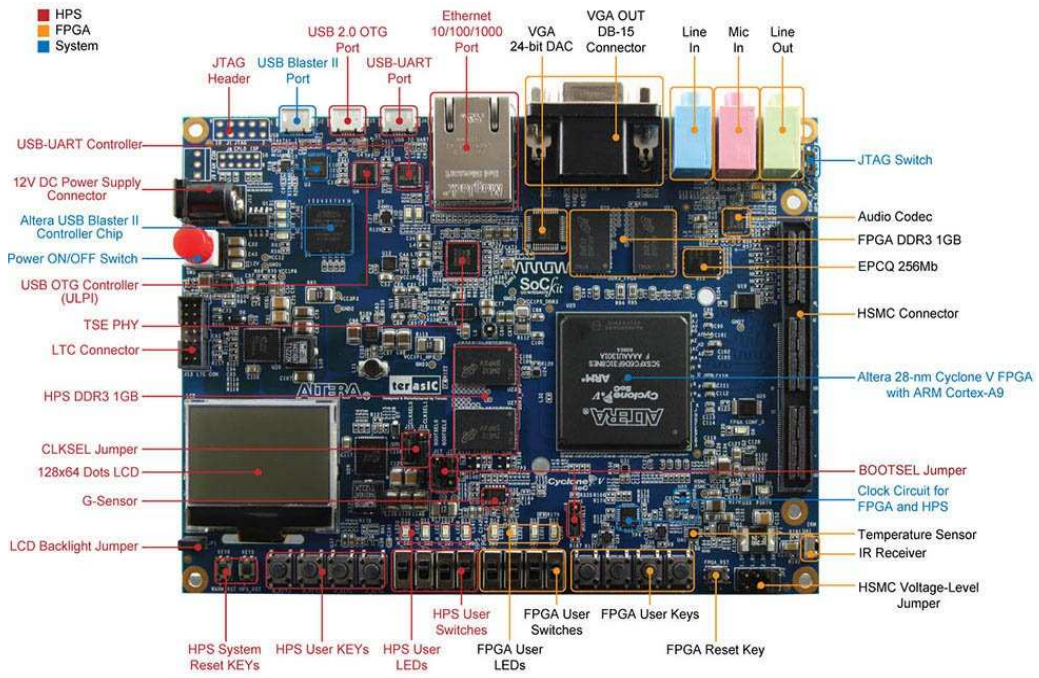


Figure 3.5: Altera DE1-SoC Board with Cyclone V FPGA

3.3 Project Description

This section provides a description of the project implementation as well as the resources used in the development of the approximate hardware. The approximate design is first implemented in ASIC and then on the FPGA. The ASIC implementation of the approximate JPEG hardware is done using the 45 nm standard cell library and the FPGA implementation is done on the Altera DE1-SoC board having 28 nm fabric.

3.3.1 Standard Cell Library

The standard cell library used in the synthesis and the timing analysis of the accurate as well as the approximate JPEG hardware is the Nangate 45 nm Open Cell Library. It is based on the NCSU FreePDK45 process kit. It is an open-source and non-manufacturable process technology, which represents the actual geometry design rules of the technology.

3.3.2 Softwares Used

This section gives a brief introduction of the softwares used in the development of the entire system.

3.3.2.1 Design Compiler

Design Compiler [9] is a synthesis tool developed by Synopsys. It optimizes the high level description of a design and syntesizes it to the gate level netlist. It can synthesize both *Verilog* and *VHDL* HDLs. It has the following four steps:

• *Analyze :* This step checks for the syntax of the HDL files.

• *Elaborate :* This step checks the whole design to make sure that the code is synthesizable, the sub-designs are connected correctly and there are no erroneous implied circuits.

- *Compile :* This step creates the gate level netlist from the HDL files.
- *Save :* This step saves the generated netlist.

Also, this tool reports the area and the power consumed by the design. Both dynamic and leakage power are reported.

3.3.2.2 PrimeTime

PrimeTime [10] is a Synopsys tool to perform the static timing analysis on a gate level description of a design. The accuracy of the timing reports generated by PrimeTime is comparable to HSPICE simulations.

3.3.2.3 Quartus Prime

Quartus Prime [3] is a design software for Altera FPGAs and SoCs. The software performs all the steps from the design entry to simulation and verification, advanced synthesis and optimization, place and route and generation of the programing files.

3.3.2.4 Modelsim

Modelsim is an HDL simulation environment developed by Mentor Graphics. It is used to simulate the designs described in both *Verilog* and *VHDL* HDLs.

3.3.3 Approach

The RTL of the precise JPEG hardware in verilog is first synthesized using the 45nm Nangate Open Cell Library using Synopsys Design Compiler.The synthesis helps in the characterization of the design in terms of area and power. The synthesized gate level netlist is then analyzed in PrimeTime to get the delay of the critical path. After characterization of the precise hardware, the approximate hardware is first implemented and then characterized.

3.3.3.1 Implementation of Approximate Hardware

For the implementation of approximate JPEG hardware, the adders and multipliers in the precise design are replaced by the approximate adders and multipliers respectively. In the JPEG compression algorithm, the DCT and the Quantizer blocks present the scope of approximation. Figure 3.6 shows the implementation of the approximate JPEG encoder block.

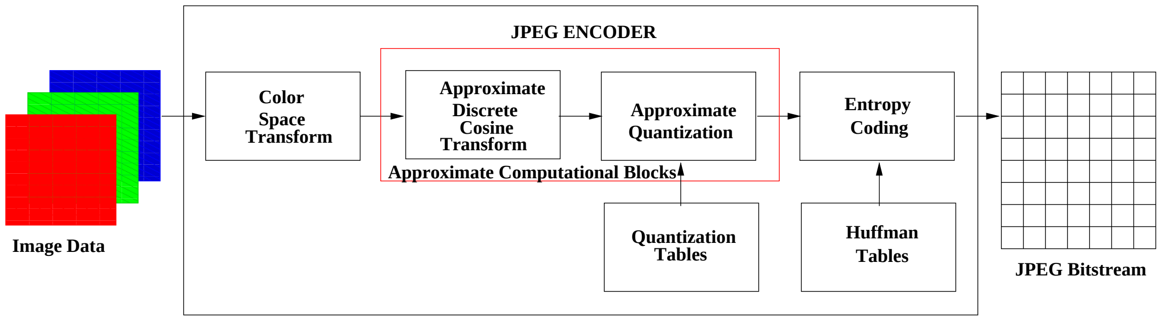


Figure 3.6: Block Diagram of Approximate JPEG Encoder

- *Approximate Adder :* The approximate adder used is based on the reconfigurable adders proposed in [27],with the difference that the number of approximate LSBs are variable, as used in [25]. So, the number of LSBs of the result to be approximated can be configured in the RTL and the resulting adder has area, power and delay characteristics proportional to the number of bits approximated. If a greater reduction in area, power and delay is required, more number of least significant bits in the result should be approximated.

- *Approximate Multiplier :* The approximate multiplier used comprises of variable number of approximated bits. The multiplier can be configured to have n number

of approximate LSBs. Based on the number of bits to be approximated, the result of the multiplication is truncated and the rest of the LSBs are replaced by 0. This method, hence reduces the effort required in the computation of the LSBs of the multiplier result in precise mode. Based on the reduction in delay and the induced inaccuracy, the multiplier can be set to approximate a specific number of LSBs.
Each color space component has a DCT and a quantizer block. Each DCT block has 73 32X32 bit multipliers and 64 32 bit adders. Every quantizer block has 64 32X32 bit multipliers. So, the entire encoder part has 411 32X32 bit multipliers and 192 32 bit adders, which need to be replaced by approximate multipliers and adders. Also, every individual multiplier and adder can be separately configured to approximate a specified number of LSBs of the result.

The approximate hardware is then synthesized and the timing analysis is done to characterize the design in terms of area, delay and power consumption. The results obtained are then compared to the results obtained from the precise hardware.

3.3.3.2 FPGA Implementation

The precise and approximate designs are implemented on the Altera DE1-SoC FPGA, which has Cyclone V device, with ARM based dual core Cortex-A9 embedded processor.The FPGA fabrication is done on 28 nm technology. The specification and development of the design is done using the Quartus II software.

• *System Specification and Development :* The RTL for the JPEG encoder is imported as a block. A fractional PLL present on the board is used to generate the clock of the desired frequency. Since the design is in a single clock domain, no synchronizer circuits are required. To input the image data, the data input pins are mapped to a single port RAM, where, the image data is stored in form of a memory initialization file. The rest of the input pins are mapped to the GPIO pins or the on-board pushbuttons and switches. The output of the encoder block, is again mapped to a single port RAM, to write the encoded bitstream in the memory. The rest of the outputss are mapped to the GPIO pins or the on-board LEDs.

• *System Validation :* The input and output of the system can be read by probing the memory units. Once, the precise and the approximate designs are implemented on the FPGA, the area, delay and power comparision between the accurate and the approximate systems is done based on the resource utilization report from the FPGA and the power analyzer tools.

4. APPROXIMATE ARITHMETIC UNIT DESIGN

This section explains the design of the basic approximate arithmetic units used in this thesis. As mentioned in section 2, there has been a lot of work in the design of approximate adders and the approximate adders have been used to design the approximate multipliers as well, as multiplication requires the addition of the partial products. The approximate adder circuit used in this thesis has been borrowed from [27] and the approximate multiplier used is designed by the truncation of the the original product term.

4.1 Approximate Adder Design

The approximate adder, as presented in [27] is an accuracy configurable adder which gracefully degrades the accuracy of the adder. The adder is named as *GDA* or the Gracefully Degrading Accuracy Configurable Adder. The idea behind this adder is that it consists of sub-adders. Each sub-adder unit is a k-bit adder. The reconfiguration property comes from the idea that the bit length of the sub-adders are configurable. Between each sub-adder unit, there are multiplexers, which choose the carry-in to the next sub-adder unit. So, the multiplexer can either choose the carry-in from the carry-out of the previous sub-adder unit, like the traditional ripple carry adder, or take the carry-in prediction.

The carry-in prediction is based on the *Generate* and *Propagate* bits generation from a Carry Lookahead Adder. Preceding t bits are observed for carry in prediction. If the carry-in predicted is correct, the result is accurate, other wise the result is inaccurate, as the carry-in prediction is assumed to be '0'. The reason of making the carry-in prediction to be '0' is that as large numbers of bits are observed for making the carry-in prediction, the carry-in propagates to the carry-in prediction only if the *Propagate* bits of all the bits being observed is ‘1’. For large number of bits being observed, the probability of all the *Propagate* bits being ‘1’ is low, hence, the carry-in can be assumed to be ‘0’. The figure 4.1 shows the schematic diagram of the approximate adder.

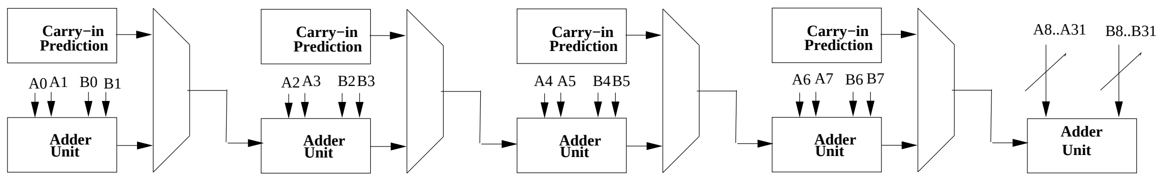


Figure 4.1: Approximate Adder Design

4.2 Approximate Multiplier Design

As mentioned earlier, the approximate multiplier is the truncated form of the accurate multiplier. The reconfiguration comes from the idea that the number of bits required to be approximated can be fed to the design and the all the LSBs till the number of bits to be approximated are replaced by ‘0’. The multiplier and the multiplicand are truncated first to implement this multiplier.

If there are *2n* bits to be approximated, and the multiplier and the multiplicand have *ma* and *mb* bits respectively, they are truncated to contain *ma-n* and *mb-n* bits respectively and then the multiplication is performed. The product obtained is then retained in the *ma+mb-2n* MSBs and the *2n* LSB positions are filled with ‘0’. This way, the *maXmb* multiplication is replaced to *ma-nXmb-n* bit multiplication. The next section presents the results of characterization of the approximate adder and multiplier units. The figure 4.2 explains the algorithm of the approximate multiplier.

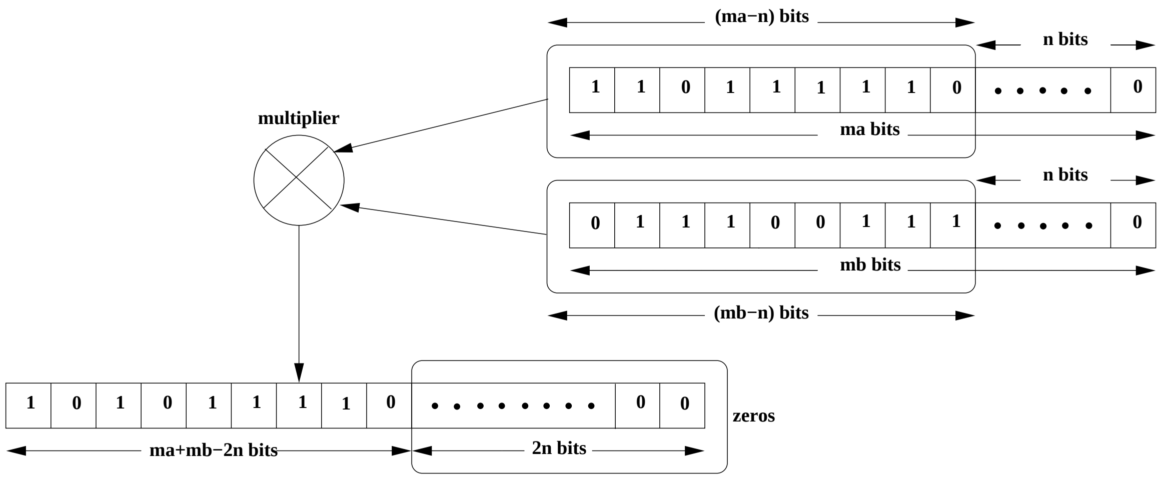


Figure 4.2: Approximate Multiplier Design

4.3 Experimental Results

This section presents the results of characterization of the approximate adder and multiplier units. Table 4.1 shows the comparison of the area reduction, power savings and delay reduction of the accurate and the approximate adder designs with different number of bits approximated. Table 4.2 shows the similar comparison for accurate and approximate multiplier designs.

In the approximate adder, the power consumption increases with the increase in the number of approximated bits. However, the delay reduces. In the approximate multiplier, the power and delay both reduce with the increase in the number of approximated bits. To quantify the effectiveness of the approximate circuits in the reduction of delay and power, power-delay product is used as a metric for comparison. The power delay product is defined by the following formula:

$$PDP = P_D * t_p \tag{4.1}$$

where, P_D is the dynamic power consumed by the circuit and t_p is the propagation delay. Figure 4.3 shows the graph of the power delay product vs. the number of approximated bits for both approximate adder and approximate multiplier circuits.

No. of Bits Approximated	Area (in μm^2)	Leakage Power (in μW)	Dynamic Power (in μW)	Delay (in ns)
0	1567.90	45.99	93.51	4.71
2	1632.89	47.56	96.32	4.68
4	1665.33	48.11	99.56	4.53
6	1712.96	49.03	103.42	4.35
8	1753.04	51.19	107.98	4.11

Table 4.1: Comparison of Accurate and Approximate Adder Design

No. of Bits Approximated	Area (in μm^2)	Leakage Power (in μW)	Dynamic Power (in μW)	Delay (in ns)
0	2700.43	47.01	117.52	5.03
2	2372.99	41.28	101.51	4.79
4	2065.76	35.93	87.18	4.54
6	1778.74	30.91	73.43	4.29
8	1513.54	26.31	61.18	4.08

Table 4.2: Comparison of Accurate and Approximate Multiplier Design

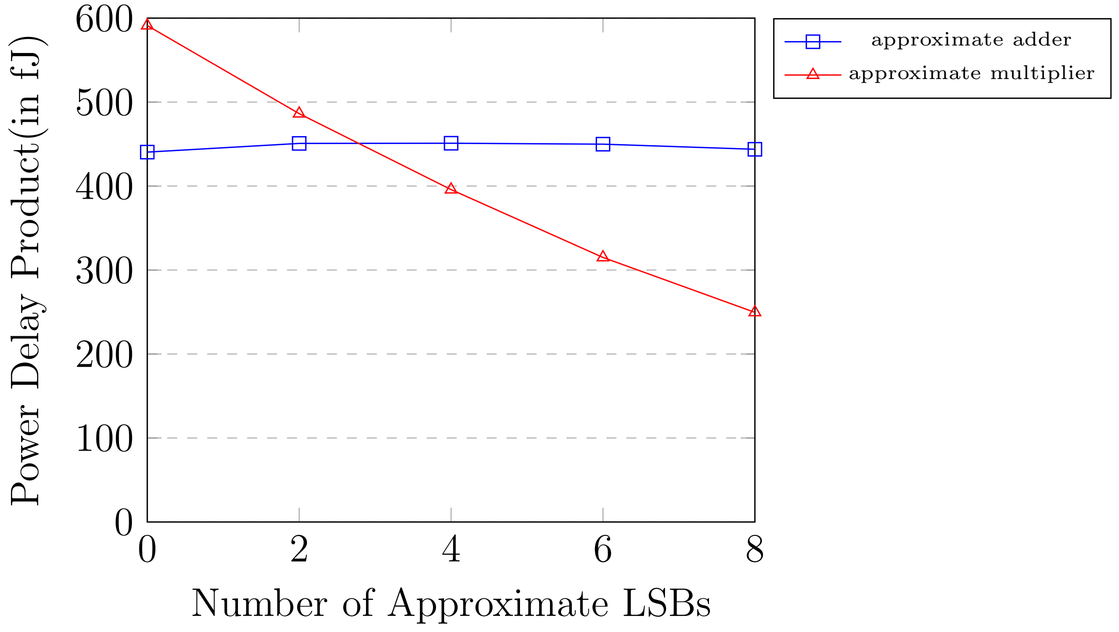


Figure 4.3: Power Delay Product of Approximate Arithmetic Units Vs. Number of Approximate LSBs

5. APPROXIMATE JPEG HARDWARE DESIGN

This section describes the design of the approximate JPEG hardware. After the approximate adder and multiplier units are designed, the DCT and the quantizer blocks of the JPEG encoder need to have their adders and multipliers replaced by the approximate ones to develop the approximate JPEG encoder core. We first discuss the design of the approximate DCT block, followed by the design of the approximate quantizer block and then the design of the approximate encoder core. The last section presents the results of characterization of the DCT and quantizer block and the encoder core in general.

5.1 Approximate DCT Design

As explained earlier, the JPEG block comprises of the color space transformation block. After the color space transformation from RGB to YCbCr, each channel has a DCT block. Each DCT block has 64 32-bit adders, 64 32X32 bit multipliers and 9 32X8 bit multipliers. In total, one DCT block has 73 multipliers and 64 adders. As there are three channels, hence, in total, there are 192 adders and 219 multipliers in the DCT block. Each of these adders and multipliers have to be replaced by the approximate adders and mutipliers respectively.

The adders and multipliers are all reconfigurable. Also, each adder and each multiplier cell can be configured independent of each other to have different levels of approximation, depending on the level of approximation required.

5.2 Approximate Quantizer Design

After the DCT block, the quantizer block is approximated. Each DCT block is succeeded by a quantizer block. So, there are three quantizer blocks in the encoder core. Each quantizer has 64 32X32 bit multipliers, therefore, the quantizer block has a total of 192 32X32 bit multipliers. All these multipliers are replaced with the approximate multiplier units. All these multipliers too, can be configured independent of each other to have different levels of approximation.

5.3 Approximate Encoder Design

After the DCT and the quantizer blocks are approximated, the entire approximate encoder core is developed by connecting the blocks as per the algorithm. The encoder core has a total of 192 approximate adders and 412 approximate multipliers. The approximate encoder core has area and delay reduction and power savings, which is presented in the next sction.

5.4 Experimental Results

In this section, we discuss the results of characterization of the DCT and the quantizer blocks and the encoder core. The synthesis is done by Design Compiler and the timing analysis is done in PrimeTime using the 45nm Nangate Open Cell Library.

Table 5.1 shows the comparison of the area reduction, power savings and delay reduction of the accurate and the approximate DCT blocks. The same comparison metrics are tabulated for the accurate and approximate quantizer blocks in table 5.2. The table 5.3 shows the results when both the quantizer and the DCT blocks are replaced for the entire encoder core. However, only the result for the accurate design and the design with 8 LSBs approximated is shown here.

The figure 5.1 shows the percentage reduction in delay plotted against the number of approximated LSBs, figure 5.2 shows the percentage savings in power verses the number of approximated LSBs and the figure 5.3 elaborates the percentage reduction in area with the number of approximated LSBs.

No. of Bits Approximated	Area (in μm^2)	Leakage Power (in mW)	Dynamic Power (in mW)	Delay (in ns)
0	162023.24	2.58	1.89	5.05
2	143882.05	2.24	1.69	4.91
4	123509.38	2.01	1.64	4.87
6	107925.77	1.78	1.55	4.71
8	93536.50	1.56	1.46	4.53

Table 5.1: Comparison of Accurate and Approximate DCT Blocks

No. of Bits Approximated	Area (in μm^2)	Leakage Power (in μW)	Dynamic Power (in μW)	Delay (in ns)
0	17280.15	366.19	700.32	4.76
2	16058.41	339.49	657.55	4.72
4	14832.69	313.46	617.76	4.66
6	13609.36	286.66	569.12	4.65
8	12390.27	259.73	519.12	4.63

Table 5.2: Comparison of Accurate and Approximate Quantizer Blocks

No. of Bits Approximated	Area (in μm^2)	Leakage Power (in mW)	Dynamic Power (in mW)	Delay (in ns)
0	629971.78	10.79	11.73	5.67
8	417035.32	7.48	9.84	5.29

Table 5.3: Comparison of Accurate and Approximate Encoder Blocks

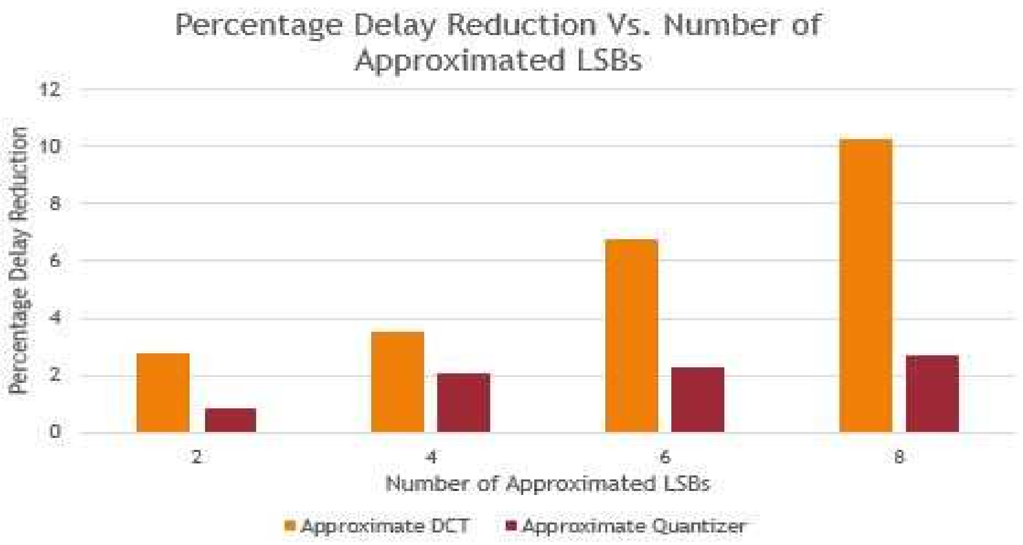


Figure 5.1: Percentage Delay Reduction Vs. Number of Approximate LSBs

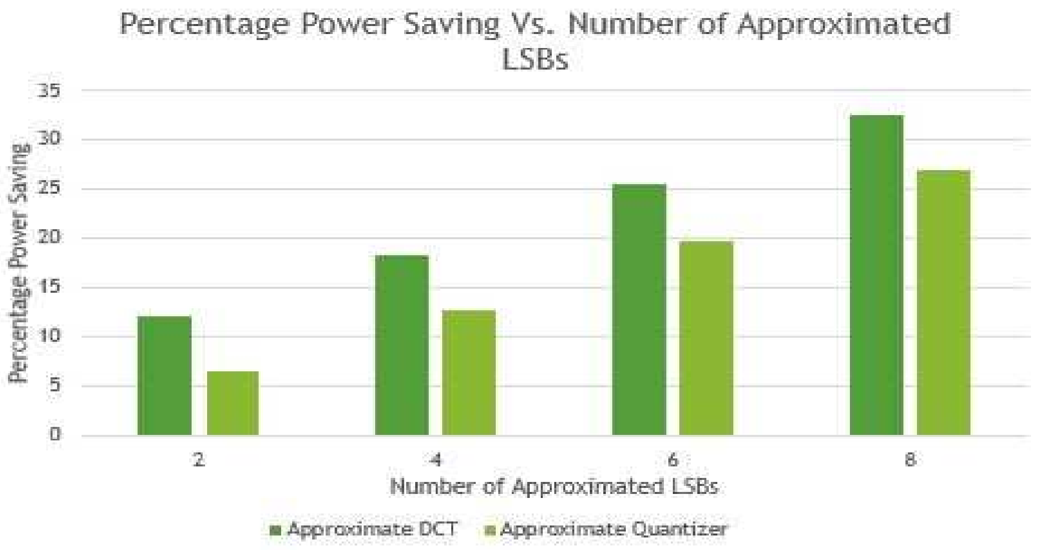


Figure 5.2: Percentage Power Saving Vs. Number of Approximate LSBs

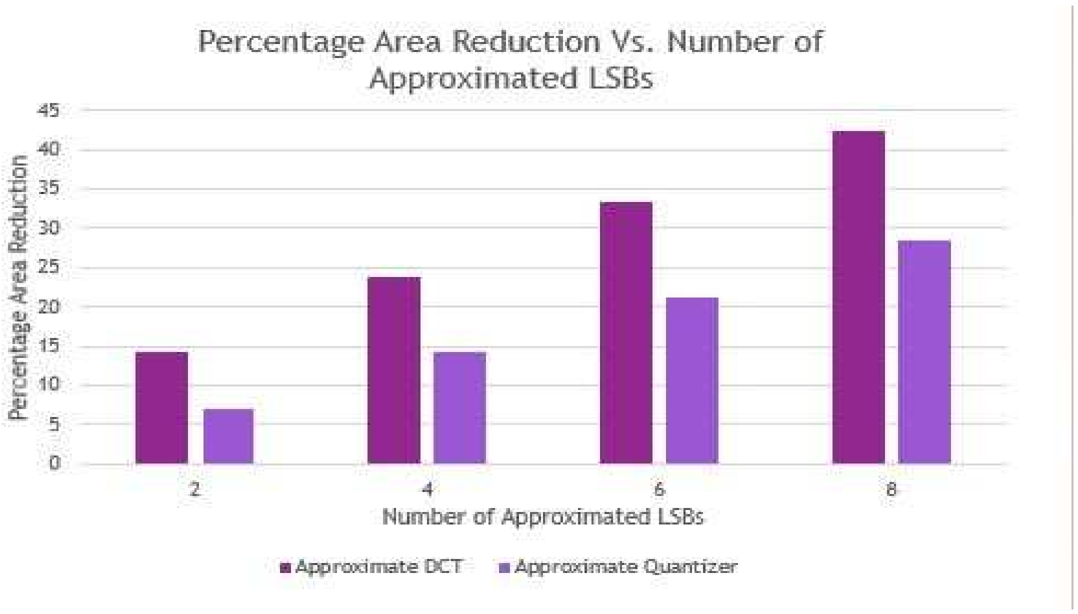


Figure 5.3: Percentage Area Reduction Vs. Number of Approximate LSBs

6. FPGA IMPLEMENTATION

This chapter describes the FPGA implementation of the JPEG hardware in detail. The first section presents the specifications of the FPGA board, the next section details the IP cores of the FPGA, the next section presents the implementation details and finally, we have the results of the system validation.

6.1 FPGA Specifications

The FPGA prototyping of the JPEG hardware has been done on the Altera DE1-SoC board having Cyclone V device and ARM Cortex-9 dual core processor. It has 457 GPIO pins and around 33000 Adaptive Logic Modules (ALMs). It also has M10K internal memory blocks. The figure 6.1 shows the internal block diagram of the DE1-SoC board.

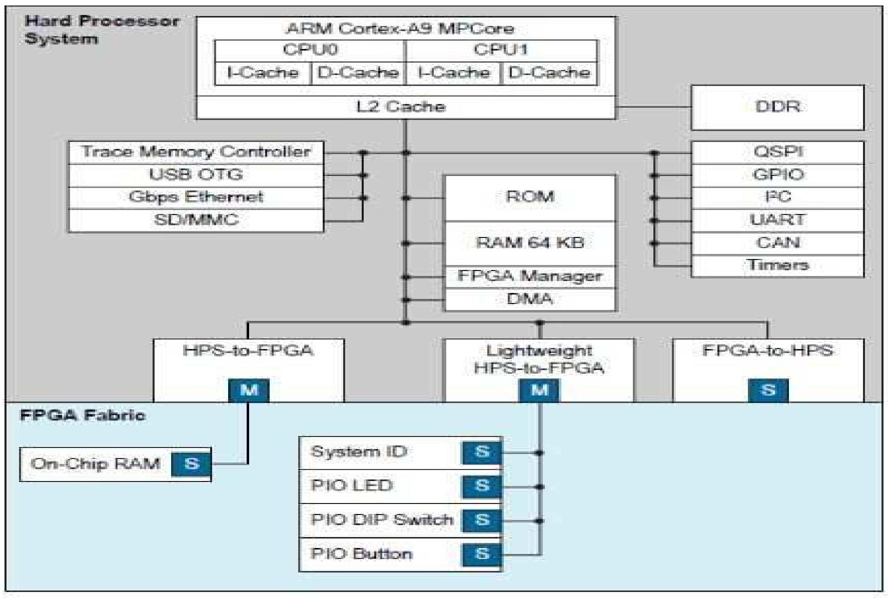


Figure 6.1: Internal Block Diagram of the DE1-SoC Board

6.2 IP Cores

The Intellectual Property core, or an IP core refers to a block of logic which can be re-used in a design as it is. Here, the IP cores being discussed are the property of Altera and are used in the FPGA implementation.

6.2.1 General Purpose Input-Output

There are 457 GPIO pins available in the Cyclone V device, which are highly configurable. They have a weak pull up and programmable bus hold. The differential output voltage and the pre-emphasis are also programmable. The components of the GPIO IP core are the DDIO (Double Data Rate Input Output), that can double or half the data rate of the chaannel, the Delay Chains to introduce delays and help in timing closure and the I/O buffers to connect the pads to the FPGA.Figure 6.2 shows the internal block diagram of the GPIO IP core.

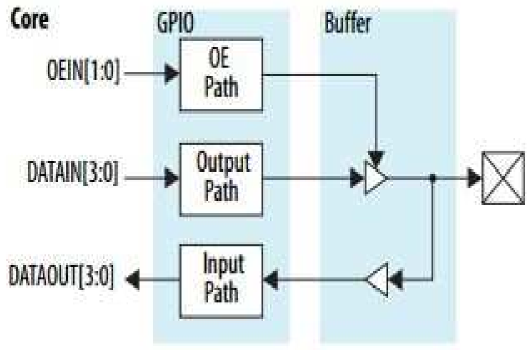
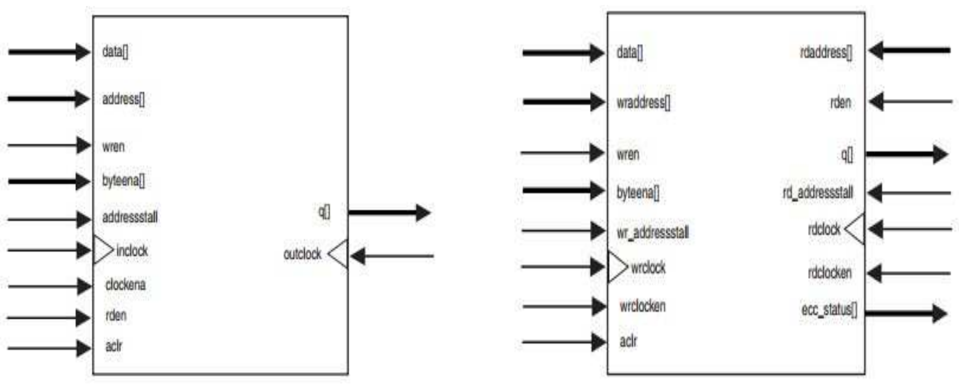


Figure 6.2: Block Diagram of the GPIO IP Core

6.2.2 Block RAM

Block RAMs are the dedicated two port memories present in the FPGA which contain several kilobits of memory. The Altera DE1-SoC has multiple blocks of RAM, which can be used to perform read and write operations or else can just be used as a ROM. The Altera IP cores have single port as well as dual port RAMs. In a single port RAM, read and write operations share the address, whereas, in a dual port RAM, the read and write operations, each have a dedicated address port. Figure 6.3 shows the difference in the IP block of the single and dual port RAMs.



(a) Single Port RAM *(b) Dual Port RAM*

Figure 6.3: IP Blocks of Single and Dual Port RAM

6.3 Implementation Details

This section elaborates the actual system implementation of the FPGA. Figure 6.4 shows the flow diagram of the FPGA implementaion. After the design has been specified in verilog, it is compiled to implement on the program the FPGA. The compilation has the following steps:

- *Analysis and Synthesis:* This step runs a syntax check as well as synthesizes the design, i.e., creates a gate level netlist of the design.

• *Functional Verification:* This step takes the input vectors from the testbench and then produces the output which has to be verified to be correct. If the output is not as expected, a change in the design logic might be required.

• *Fitter:* This function takes the pin assignments as the input and runs the place and route operation on it.

• *Assembler:* This operation creates a pattern of bits out of the program to be downloaded onto the FPGA.

• *Timing Analysis:* This operation performs the timing analysis and gives the details about the maximum clock frequency and timing closure.

6.3.1 Image Data

The 512X512 pixel data of the image is broken into 8X8 blocks and each component value (RGB) of 8 bits each are concatenated into a 24 bit data. The data is then stored in a Block RAM configured as a ROM. An address counter is used to generate a new ROM address every clock cycle for the data input to the JPEG encoder.

6.3.2 System Schematic

The system schematic is a block description file or a *.bdf*, which comprises of a PLL to geenerate the clock, a ROM to input the data, the address counter, the JPEG encoder core and a dual port RAM where the generated bitstream is dumped. There is only a single global clock in the design, so, no synchronizers are needed. The figure 6.5 shows the high level schematic of the interconnections between the different blocks for the FPGA implementation.

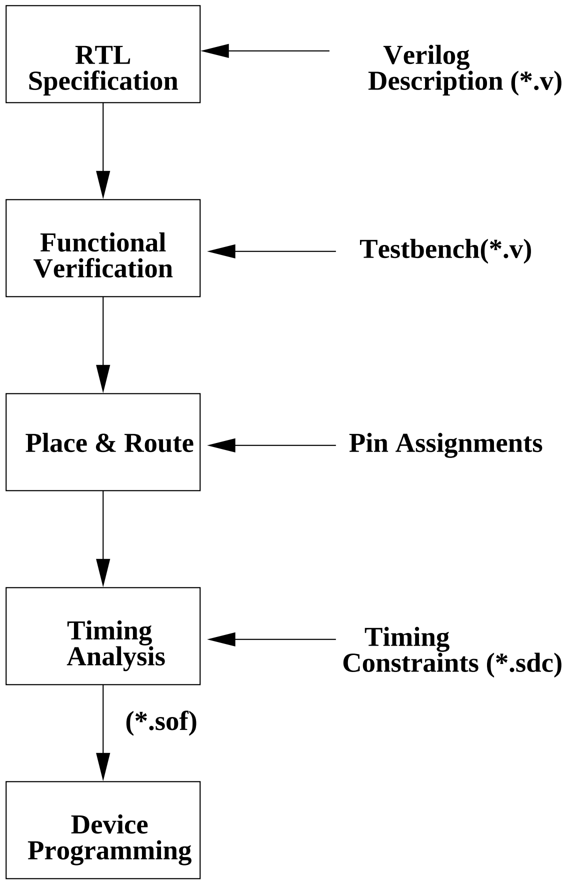


Figure 6.4: Flow Diagram of FPGA Implementation

6.3.3 Programming the FPGA

After the description of the system schematic in *.bdf*, the system is compiled and after the place and route operation, the object file is created, which contains the bitsream to program the FPGA. The SRAM object file, or the *.sof* file is then downloaded on the FPGA to program it.

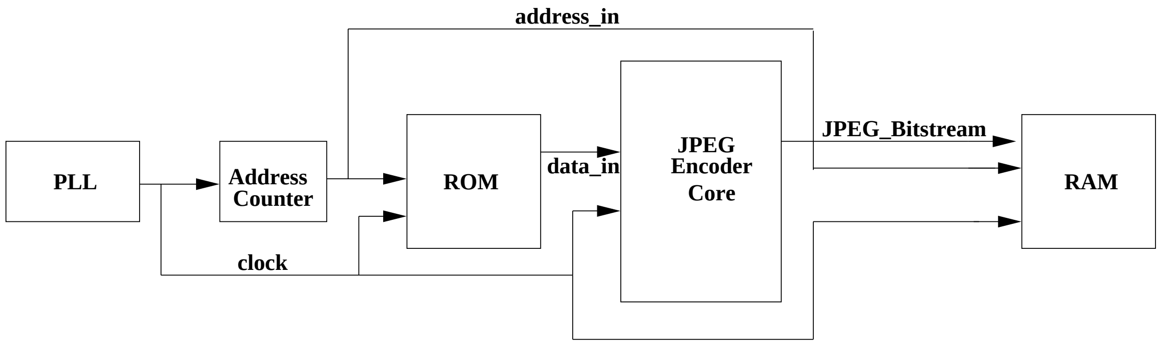


Figure 6.5: Schematic of the FPGA Implementation

6.3.4 Bitstream Capture

After the program is downloaded to the FPGA, the block RAMs are probed to read the loaded image data. The system is then emulated on the FPGA and the generated JPEG bitstream is captured by probing the dual port RAM.

6.4 System Validation

After the implementation on FPGA, the design is then validated to check for the functional correction, area and delay reduction and power savings.

6.4.1 Functional Verification

The pre-silicon functional verification is done by simulating the system on Modelsim and observing the waveforms to verify the desired working of the FPGA. The post implementation testing is done by probing the block RAM units to read the generated bitstream and check for its correctness. Figure 6.6 shows the results from a simulation run.

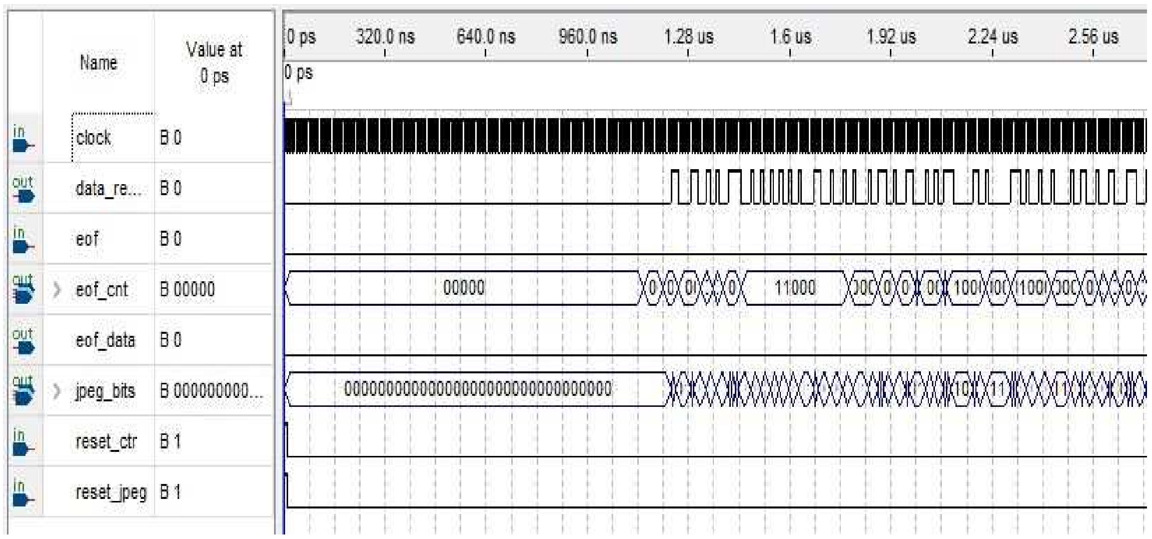


Figure 6.6: Simulation Results

6.4.2 Area Reduction

Area occupied by a design over FPGA is estimated by the number of ALMs (Adaptive Logic Modules) used in the implementation of the design over the FPGA. Cyclone V devices use a 28nm ALM as the basic building block of the logic fabric. It uses an 8-input fracturable LUT, with 4 dedicated registers to improve the timing closure of the circuit.The figure 6.7 shows the block diagram of an ALM. The different accurate and approximate versions of the JPEG encoder core was implemented on the FPGA.The table 6.1 shows the results obtained by the implementation on the FPGA.

6.4.3 Power Saving

The power consumed by the design over the FPGA is reported by the PowerPlay tool, which is the power analysis and optimization tool in Quartus II design software. It takes the clock assignments and the placement and routing details of the design as input and then gives the power consumption by a design on the FPGA. It

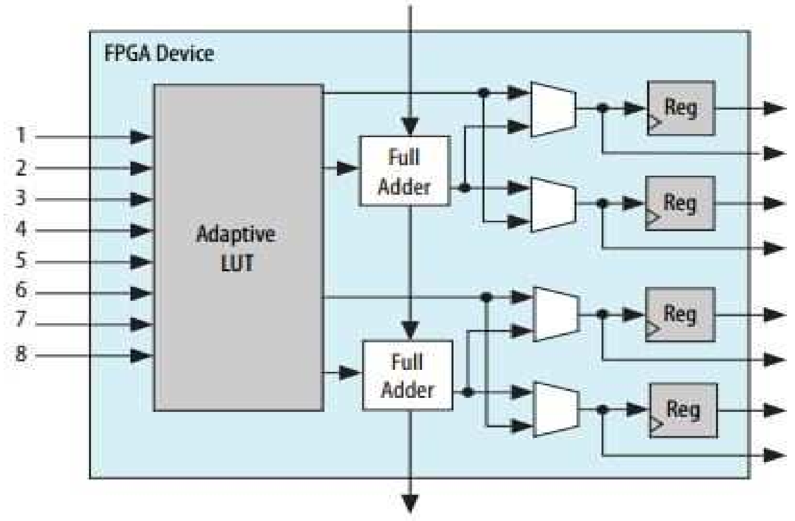


Figure 6.7: Block Diagram of an ALM

Area Elements	Accurate Design	No. of Approximate LSBs in Approximate Design			
		2	4	6	8
No. of ALMs Used	32398	29365	26714	23511	17780
No. of Registers Used	33249	30271	28833	27100	25339
No. of DSP Blocks Used	112	87	56	34	0

Table 6.1: Area Comparison of Accurate and Approximate Designs

gives a report on the total dynamic power and static power consumed by the design implementation. The total power dissipation is the sum of the total dynamic and static power along with the I/O power consumption. The results obtained are shown in the table 6.2.

6.4.4 Delay Reduction

Delay comparison is obtained by the worst case delay of the critical path by the timing constraints, clock frequency and the slack reports from the timing analysis. The timing analysis is done over two timing models and the results are shown in

Power Elements	Accurate Design	No. of Approximate LSBs in Approximate Design			
		2	4	6	8
Core Dynamic Power Dissipation (in mW)	611.58	537.22	439.06	371.36	318.06
Core Static Power Dissipation (in mW)	418.94	418.47	418.49	417.50	417.35
Total Power Dissipation (in mW)	1042.47	967.81	870.10	788.86	747.53

Table 6.2: Power Comparison of Accurate and Approximate Designs

table 6.3.

Timing Model	Accurate Design	No. of Approximate LSBs in Approximate Design			
		2	4	6	8
1100 mV 85 C Model (in ns)	9.273	9.112	8.830	8.241	6.592
1100 mV 0 C Model (in ns)	9.164	9.102	8.751	8.008	6.522

Table 6.3: Delay Comparison of Accurate and Approximate Designs

6.4.5 Clock Speed

The TimeQuest Timing Analyzer tool, which is the timing analysis tool in the Quartus II software, reports the maximum allowable clock speed for a design. The background calculation is based on the setup and hold time analysis between the source and the destination ports/registers in the same clock domain. It takes the netlist of the design as well as the placement and routing details as inputs with the time constraints mentioned by the user. Table 6.4 enlists the comparison of the clock speeds between the accurate and approximate designs.

The figure 6.8 shows all the comparison metrics plotted against the number of LSBs approximated.

Timing Model	Accurate Design	No. of Approximate LSBs in Approximate Design			
		2	4	6	8
1100 mV 85 C Model (in MHz)	88.71	94.73	102.68	109.64	120.86
1100 mV 0 C Model (in MHz)	89.57	95.33	102.81	111.06	121.05

Table 6.4: Clock Speed Comparison of Accurate and Approximate Designs

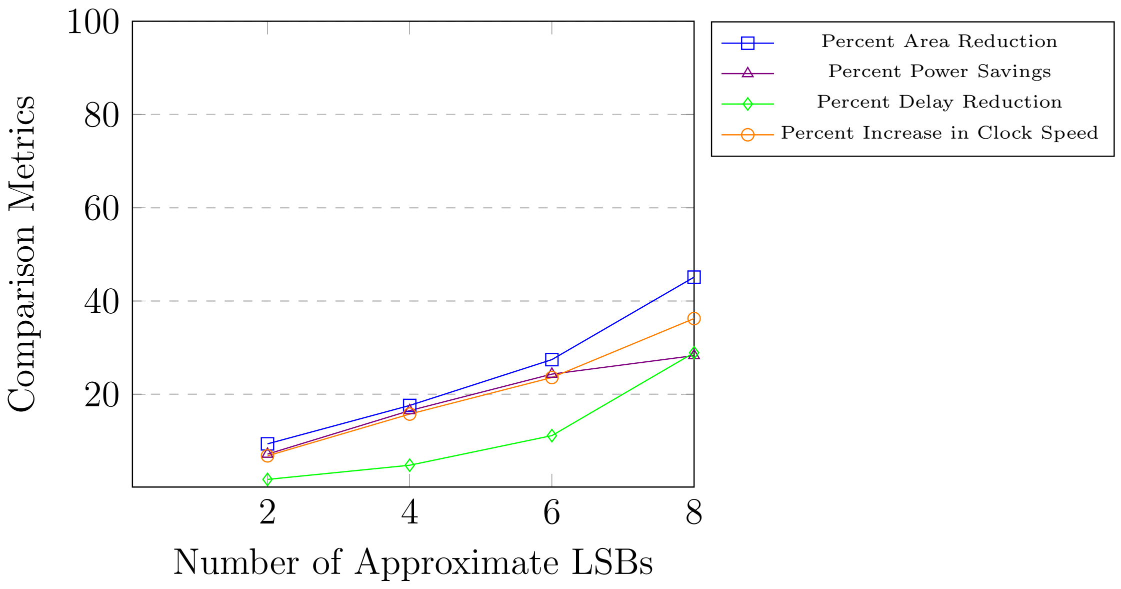


Figure 6.8: Plot of the Comparison Metrics Vs. Number of Approximated LSBs

6.4.6 PSNR Degradation

The output stored in the RAM can be read out to a .hex file. The .hex file can then be imported in MATLAB [13] to compare the error in terms of peak-signal-to-noise-ratio (PSNR) between the accurate and the approximate designs. Table 6.5 shows the comparison of the PSNR degradation between the accurate and the approximate designs.

No. of Approximate LSBs	PSNR (in dB)		
	ja.jpg	lena.jpg	peppers.jpg
0	42.35	32.40	39.85
2	32.68	28.75	27.93
4	29.45	25.11	24.56
6	22.77	22.98	21.87
8	18.78	15.17	17.91

Table 6.5: Comparison of PSNR Degradation for Accurate and Approximate Designs

6.4.7 Approximation with Different Compression Ratios

As mentioned previously, some of the image data is lost in the process of quantization. Quantization involves the division of each frequency component after DCT in the 8X8 pixel block with their respective constants in the 8X8 quantization matrix, represented by Q, and then rounding off the result. The compression ratio is controlled by the elements of the quantization matrix, as, larger values in the quantization matrix produce greater compression ratios and smaller PSNR values. The quantization matrix most widely used in standard JPEG compression corresponds to a compression ratio of 15:1 and is shown below:

$$Q = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

To obtain larger compression, we need to increase the value of the coefficients of Q. The quantization matrix shown below, for example, produces a compression ratio of 46:1.

$$Q = \begin{bmatrix} 80 & 55 & 50 & 80 & 120 & 200 & 255 & 305 \\ 60 & 60 & 70 & 95 & 130 & 290 & 300 & 275 \\ 70 & 65 & 80 & 120 & 200 & 285 & 345 & 280 \\ 70 & 85 & 110 & 145 & 255 & 435 & 400 & 310 \\ 90 & 110 & 185 & 280 & 340 & 545 & 515 & 385 \\ 120 & 175 & 275 & 320 & 405 & 520 & 565 & 460 \\ 245 & 320 & 390 & 435 & 515 & 605 & 600 & 505 \\ 360 & 460 & 475 & 490 & 560 & 500 & 515 & 495 \end{bmatrix}$$

Table 6.6 shows the comparison of the PSNR degradation with the number of approximated LSBs for different compression ratios.

No. of Approximate LSBs	PSNR (in dB) for Different Compression Ratios		
	2.5:1	15:1	46:1
0	42.35	29.53	20.71
2	32.68	20.12	11.81
4	29.45	17.34	9.15
6	22.77	15.20	7.23
8	18.78	12.97	4.50

Table 6.6: Comparison of PSNR Degradation with the No. of Approximate LSBs for Different Compression Ratios

7. CONCLUSION AND FUTURE WORK

An approximate reconfigurable JPEG encoder was designed and implemented on the FPGA platform in this thesis. The experimental results show a marked improvement in the area, power and delay metrics of the approximate application over the accurate application. However, the PSNR keeps reducing with the increasing the number of approximated bits. So, the value of PSNR ratio required should be carefully weighed against the amount of area and power savings and the delay reduction. Also, as the approximation discussed in this thesis is not dependent on the image, some images might not show a huge PSNR degradation even for a large number of approximated bits and vice versa. Different variations of the number of bits approximated for each individual adder and multiplier units too might be required for better PSNR depending on the image.

The current work has just been done in the JPEG compression, the future scope of the work is to extend it to the motion picture or the MPEG encoding algorithm. An extension to MPEG would be applying the JPEG approximation to each and every frame of the motion picture to achieve the desired signal level with efficient resource usage. Also, the work discussed in this thesis is concerned with the image independent approximation. As mentioned above, different images might have different scope for approximation. So, image dependent approximation techniques should be explored. As the discrete cosine transform block is the most computationally intensive step, the other applicatons that use the DCT can also be optimized by using the approximate DCT computation instead of the accurate one. Also, the FPGA platform could be interfaced to a VGA monitor to have a display of the images in real time to make the PSNR degradation more intuitive for the user.

REFERENCES

[1] Vinay K. Chippa, Srimat T. Chakradhar, Kaushik Roy, and Anand Raghunathan. Analysis and characterization of inherent application resilience for approximate computing. In *Proceedings of the 50th Annual Design Automation Conference*, DAC '13, pages 1–9. IEEE, 2013.

[2] Altera Corporation. Altera Documentation. `https://www.altera.com/en_US/pdfs/literature/hb/cyclone-v/cv_51001.pdf`. [Accessed: June 10, 2016].

[3] Altera Corporation. Quartus Prime Standard Edition Handbook. `https://www.altera.com/en_US/pdfs/literature/hb/qts/qts-qps-handbook.pdf`. [Accessed: October 31, 2015].

[4] Farzad Farshchi, Muhammad Saeed Abrisham, and Sied Mehdi Fakhraie. New approximate multiplier for low power digital signal processing. In *17th CSI International Symposium on Computer Architecture and Digital Systems*, CADS '13, pages 42–47. IEEE, 2013.

[5] Vaibhav Gupta, Debabrata Mohapatra, Sang Phill Park, Anand Raghunathan, and Kaushik Roy. IMPACT: Imprecise adders for low-power approximate computing. In *Proceedings of the 17th IEEE/ACM International Symposium on Low-power Electronics and Design*, ISLPED '11, pages 409–414. IEEE Press, 2011.

[6] Vaibhav Gupta, Debabrata Mohapatra, Anand Raghunathan, and Kaushik Roy. Low-power digital signal processing using approximate adders. *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(1):124–137, 2013.

[7] Jie Han, Cong Liu, and Fabrizio Lombardi. A low-power, high-performance approximate multiplier with configurable partial error recovery. In *Proceedings of the Conference on Design, Automation & Test in Europe*, DATE '14, pages 1–4. EDAA, 2014.

[8] Jie Han and Michael Orshansky. Approximate computing: An emerging paradigm for energy-efficient design. In *Proceedings - 2013 18th IEEE European Test Symposium*, ETS '13, pages 1–6. IEEE, 2013.

[9] Synopsys Inc. Design Vision User Guide. `http://beethoven.ee.ncku.edu.tw/testlab/course/VLSIdesign_course/course_96/Tool/Design_Vision_User_Guide.pdf`. [Accessed: September 15, 2005].

[10] Synopsys Inc. PrimeTime. `http://www.synopsys.com/tools/implementation/signoff/documents/primetime_ds.pdf`. [Accessed: March 2, 2016].

[11] Terasic Technologies Inc. DE1-SoC User Manual. `ftp://ftp.altera.com/up/pub/Altera_Material/Boards/DE1-SoC/DE1_SoC_User_Manual.pdf`. [Accessed: March 14, 2014].

[12] The IEEE Inc. IEEE Standard Verilog Hardware Description Language. `https://inst.eecs.berkeley.edu/~cs150/fa06/Labs/verilog-ieee.pdf`. [Accessed: September 28, 2001].

[13] The MathWorks Inc. MATLAB Documentation. `http://www.mathworks.com/help/matlab/`. [Accessed: September 5, 2014].

[14] Honglan Jiang, Jie Han, and Fabrizio Lombardi. A comparative review and evaluation of approximate adders. In *Proceedings of the 25th Edition on Great Lakes Symposium on VLSI*, GLSVLSI '15, pages 343–348. ACM, 2015.

[15] Andrew B. Kahng and Seokhyeong Kang. Accuracy-configurable adder for approximate arithmetic designs. In *Proceedings of the 49th Annual Design Automation Conference*, DAC '12, pages 820–825. ACM, 2012.

[16] Zvi M. Kedem, Vincent J. Mooney, Kirthi Krishna Muntimadugu, and Krishna V. Palem. An approach to energy-error tradeoffs in approximate ripple carry adders. In *Proceedings of the 17th IEEE/ACM International Symposium on Low-power Electronics and Design*, ISLPED '11, pages 211–216. IEEE Press, 2011.

[17] Animesh Kumar. *SRAM Leakage-Power Optimization Framework: a System Level Approach*. PhD thesis, University of California at Berkeley, 2008.

[18] Jinghang Liang, Jie Han, and Fabrizio Lombardi. New metrics for the reliability of approximate and probabilistic adders. *Transactions on Computers*, 62(9):1760–1771, 2013.

[19] Chia-Hao Lin and Ing-Chao Lin. High accuracy approximate multiplier with error correction. In *31st International Conference on Computer Design*, ICCD '13, pages 33–38. IEEE, 2013.

[20] David Lundgren. JPEG Encoder Core. `http://opencores.org/project,mkjpeg`. [Accessed: September 25, 2014].

[21] Jin Miao, Ku He, Andreas Gerstlauer, and Michael Orshansky. Modeling and synthesis of quality-energy optimal approximate adders. In *Proceedings of the International Conference on Computer-Aided Design*, ICCAD '12, pages 728–735. IEEE, 2012.

[22] Sparsh Mittal. A survey of techniques for approximate computing. *ACM Computing Surveys*, 48(4):1–33, 2016.

[23] Debabrata Mohapatra, Vinay K Chippa, Anand Raghunathan, and Kaushik Roy. Design of voltage-scalable meta-functions for approximate computing. In *Design, Automation & Test in Europe Conference & Exhibition*, DATE '14, pages 1–6. IEEE, 2011.

[24] Abbas Rahimi, Andrea Marongiu, Rajesh K. Gupta, and Luca Benin. A variability-aware OpenMP environment for efficient execution of accuracy-configurable computation on shared-FPU processor clusters. In *Proceedings of the 9th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, CODES+ISSS '13, pages 1–10. IEEE Press, 2013.

[25] Farhana Sharmin Snigdha, Deepashree Sengupta, Jiang Hu, and Sachin S. Sapatnekar. Optimal design of JPEG hardware under the approximate computing paradigm. In *Proceedings of the 53rd Annual Design Automation Conference*, DAC '16, pages 1–6. ACM, 2016.

[26] Ajay K. Verma, Philip Brisk, and Paolo Ienne. Variable latency speculative addition: A new paradigm for arithmetic circuit design. In *Design, Automation & Test in Europe Conference & Exhibition*, DATE '08, pages 1250–1255. ACM, 2008.

[27] Rong Ye, Ting Wang, Feng Yuan, Rakesh Kumar, and Qiang Xu. On reconfiguration-oriented approximate adder design and its application. In *Proceedings of the International Conference on Computer-Aided Design*, ICCAD '13, pages 48–54. IEEE, 2013.