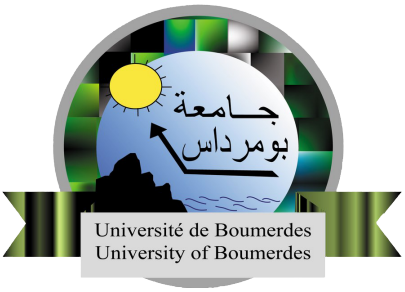
People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research
University M'Hamed BOUGARA – Boumerdes



Institute of Electrical and Electronic Engineering
Department of Electronics

Project Report Presented in Partial Fulfilment of
the Requirements for the Degree of

License

In Electrical and Electronic Engineering

Title:

JPEG Compression Algorithm on FPGA

Presented by:

- **Abdelmadjid DAHMANI**
- **Mohamed Nasreddine BELDJEMEL**

Supervisor:

- **Dr. Ahmed MAACHE**

Registration number: 2021/2022

JPEG Compression Algorithm on FPGA

Abdelmadjid DAHMANI, Mohamed Nasreddine BELDJEMEL

Abstract

Image compression is a widely used technique to reduce the size of images. It is a type of data compression that takes advantage of the visual perception and the statistical properties of an image. This topic is extremely important as images takes considerable amount of storage. Not only this increases the cost to storage, the cost to transmit also increases which also affects computational speeds.

Images compressors and decompressors (CODECs) are usually implemented in software. However, much higher speeds can be achieved by implementing a CODEC as a digital circuit. Customizing the hardware for a specific algorithm or application improves performance significantly. Customized hardware also does not suffer from any overhead compared to general purpose designs like microprocessors.

The two common ways to implement such hardware design is ASICs (Application Specific Integrated Circuit) and FPGAs (Field Programmable Gate Array). FPGA was chosen for this project as it is easily reconfigurable while also benefiting from the speed improvements compared to software design.

The JPEG (Joint Photographic Experts Group) compression algorithm is the most widely used image compression in the world. It was chosen for this project as it is well defined and has many educational resources available. The algorithm was implemented on the Intel Altera DE1 Cylone V Board. The ESP32 microcontroller was used to bridge between the FPGA and PC (Personal Computer) as an interface. It was programmed with C++. The supporting software in the PC was programmed with python.

The JPEG core runs on a 25MHz clock with a parallel design where each 8 by 8 block of image takes from 200 to 700 clock cycles based on the complexity of the block. Regardless of how fast the JPEG Core is, the design's total performance was limited by intrinsic restrictions in the interface to the FPGA. The design uses 82% of the Cyclone V logic elements.

Acknowledgements

Praise to Allah the Almighty and most merciful for giving us the wisdom, persistence and potency to complete this project. We revere the patronage and moral support extended with love by our families, especially our parents whose financial and constant passionate encouragement made this undertaking a reality.

We thank our colleagues, WameedhSC and people whose assistance and contributions are greatly acknowledged.

Contents

List of Figures

List of Tables

Introduction

The term "JPEG" is an acronym for the Joint Photographic Experts Group, which created the standard in 1992 [1]. The standard specifies process for converting source image data to compressed image data with it's implementation and vice versa (i.e. the decompression process). The standard describes both lossy and lossless CODECs.

The baseline JPEG is a lossy image compression that typically achieves 10:1 compression ratio. It is based on the discrete cosine transform (DCT) which is an embrassingly parallel workload. This makes it a perfect candidate to be accelerated with a digital design.

Objective

The objective is to implement the baseline JPEG algorithm on the Cyclone V FPGA using VHDL. Through this project, we plan to analyze the results and compare them with the software implementation. With this approach, we can figure out the most optimal way to implement this algorithm. Our aim of this project is no more than educational purposes. Digital design is an extremely complex and vast field and we consider this project to be first step into it.

Structure of the work

Chapter one is background knowledge about image compression and JPEG with chapter two describing the inner workings of the baseline JPEG algorithm. Followed by chapter three which is the hardware implementation on the Cyclone V FPGA. Chapter four analyzes the results and discuss the pros and cons of the design and ending with chapter five being the final conclusion and future works that will build on top of this project.

Chapter 1

Background

1.1 Image Compression

Digital images, whether taken from a camera or designed using software, contains large amounts of data when stored as a bitmap. This slows down computations considerably.

Modern technologies such as deep learning requires thousands of high resolution images to achieve accurate results. Using the raw bitmap format for images becomes extremely expensive in terms of storage and performance. Therefore, image compression [2] is essential to reduce the overhead on storage mediums, network and data processing.

Computer scientists studied image compression techniques for decades and different algorithms were developed and improved over the years. There are two types of image compression:

- **Lossless Compression :** reduces the file size without a loss in quality. This allows the perfect reconstruction of the original image from the compressed data.
- **Lossy Compression :** reduces the file size by eliminating redundant data. This makes the compressed data smaller in size without a noticeable loss in the image quality.

1.2 The Human Visual System

The Human Visual System (HVS) [3] describes the way humans perceives an image. In general, the human eye does not perceive the high frequency components and the colors of the image as compared to the low frequency components and the brightness of the image.

Low frequency components corresponds to slowly varying color that creates the general shape of the image. High frequency components represents fine details that can be ignored to a certain degree.

The same concept occurs with brightness and colors. the human eye perceives the general brightness better than the colors of the image.

Thus, we can take advantage of the HVS by reducing both the data that represents high frequency components and the colors of the image. With this method, the image size is reduced significantly without causing a noticeable loss in quality of the image.

1.3 Color Format

Pixels in an image can be represented in various formats. RGB (Red, Blue, Green) and YCbCr (Luminance, Chrominance Blue, Chrominance Red) are two commonly used color format. RGB is used more than YCbCr as the majority of modern displays uses RGB as the input format. This does not mean that RGB is superior to YCbCr as both formats have their advantages and uses. In fact, YCbCr is the preferred color format in the JPEG standard.

YCbCr separates luminance (brightness) and chrominance (color) of the pixel whereas in RGB, brightness is incorporated with the color values for each channel. The Y component is the luminance and Cb, Cr components are for the blue and red chrominance. Separating luminance and chrominance is important for image compression as we can specifically reduce the colors without affecting the overall brightness of the image.

1.4 JPEG Compression

JPEG, standing for "Joint Photographic Experts Group", is the most widely used image file format in the world. The JPEG standard [1] provides an efficient image compression algorithm that can be modified in various ways to fit the desired application needs. Including the baseline JPEG algorithm which is based on the sequential mode, there are 4 modes of operation

1.4.1 Sequential Mode

The baseline JPEG method is used in the sequential DCT mode of operation. This method can provide excellent compression ratios while sacrificing image quality. The output block has a single DC component and 63 AC components after each input 8x8 block of pixels is transformed to frequency domain using the DCT. Then the block is quantized, which reduces the entropy of the image data significantly. The DC component is then encoded predictively using the difference between the current and previous DC values. The final step is entropy encoding. Only Huffman encoding models are used in this mode, not the arithmetic encoding models used in other JPEG extensions.

The baseline JPEG focuses on removing the redundancies in the image data. The main two types of redundancies are:

- **Coding redundancy :** refers to the entropy of the image. In the sense that more bits are used to convey the information contained in the image. JPEG achieves this through Huffman Encoding.

- **Psycho-visual redundancy :** refers to the fact that the human visual system perceives images in such a way that removing specific details results in a nearly indistinguishable images for humans. JPEG achieves this through Quantization.

1.4.2 Progressive Mode

JPEG compression based on progressive DCT has two complementary coding algorithms. The purpose of this mode is to display low-quality photos that gradually improve during compression. Spectral-selection and successive approximation are the two different methods used. During decompression, the image will gradually come into focus. This approach is utilized in applications that require these qualities.

1.4.3 Lossless Mode

Simply said, when comparing the original picture to the reproduced image, this mode of JPEG experiences no loss. This approach employs predictive, differential coding rather than the discrete cosine transform and quantization. Although this approach does not reach large compression ratios, it is appropriate for some applications that demand exceptionally accurate picture reproduction.

1.4.4 Hierarchical Mode

The hierarchical JPEG extension is a multi-stage compression methodology where each dimension is down sampled and then encoded using progressive, sequential, or lossless encoding methods. To reconstruct the source image, the encoded data stream must be decoded and up-sampled.

Chapter 2

Baseline JPEG Algorithm

The Baseline algorithm is the most basic version of JPEG. The input image is assumed to be 8 bit per channel (24 bit for 3 channels). The algorithm splits the image into 8x8 blocks of pixels. Each block goes through various stages and outputs a byte stream representing the compressed block. The blocks are serially input in raster order.

Baseline JPEG consists of various stages. Each RGB input block goes through color conversion to YCbCr, level shift, discrete cosine transform, quantization, zigzag scan and entropy coding based on run-length and Huffman encoding.

2.1 Color Conversion And Level Shift

The color conversion from RGB to YCbCr is described with the following equations that follow the ITU-T T.871 standard [4].

$$Y' = 0 + (0.299 \cdot R) + (0.587 \cdot G) + (0.114 \cdot B) \tag{2.1}$$
$$Cb = 128 - (0.168736 \cdot R) - (0.331264 \cdot G) + (0.5 \cdot B) \tag{2.2}$$
$$Cr = 128 + (0.5 \cdot R) - (0.418688 \cdot G) - (0.081312 \cdot B) \tag{2.3}$$

The resulting values from these equations are 8 bits with a range from 0 to 255. However, the discrete cosine transform performs better if the range is from -128 to 127 as this increases symmetry in the data. Any symmetry will reduce entropy resulting in better compression. Therefore, a level shift is done by subtracting 128 from each channel.

2.2 The Discrete Cosine Transform

DCT is a widely used transformation technique in signal processing and data compression. The transformation was introduced by Nasir Ahmed along with his PhD student T. Raj Natarajan and friend K. R. Rao in 1974 [5]. They originally intended DCT to be used in image compression.

Extensive amount of work and many research papers on DCT emerged since 1974. This reasearch alongside the original paper of Ahmed served as the basis for the JPEG compression algorithm in 1992 [1].

DCT represents a given input signal with series of cosine terms. It is essentially a special case of the Fourier Transform. JPEG uses the two-dimensional version of DCT and takes 8x8 block of pixels as input.

The result is an 8x8 coefficients matrix for each cosine term with different frequencies named as the basis functions shown in Figure 2.1. The 1st element at row 0 and column 0 is called the DC term as it corresponds to a frequency of 0 in both X and Y axis. The DC term is essentially the average value of the entire block. The rest 63 terms are called the AC terms, each term has a specific frequency on the X and Y axis.

Combining all of the terms with each one of them scaled by its coefficient can perfectly replicate any 8x8 block of an image. Each term represents the resulting pixel pattern when the term value is maximum and the rest of the terms are set 0.

DCT is important because it allows us to separate low frequency components from high frequency ones. This separation allows to eliminate high frequency components without affecting low frequency ones.

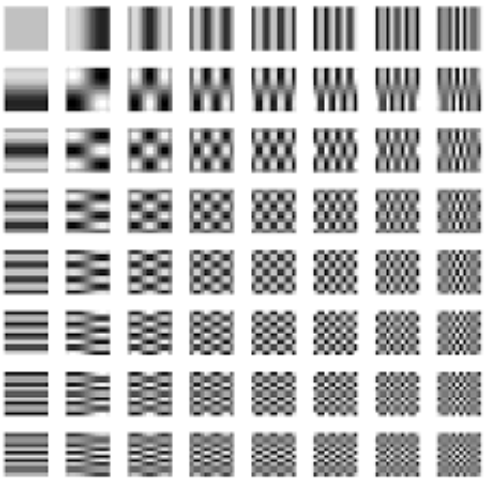


Figure 2.1: DCT Basis Functions

2.3 Quantization

Quantization is the step where redundancies are eliminated or reduced improving the compression considerably. Quantization targets the components that are not perceivable to the human eyes, specifically the high frequency components and the chrominance componenets.

Quantization is essentially dividing the coefficient matrix that resulted from DCT by an 8x8 matrix called a quantization table. Larger values in the quantization table reduce the final file size but increases visual artifacts and destroys potentially

visible details. In the other hand, smaller values results in bad compressions but eliminates visual artifacts. In short, desiging the table effectively, by quantizing the less important components harshly, is the key to improving compression without any visible loss in quality.

In baseline JPEG, quantization tables can be customized depending on the target application but designing an optimal table can be quite tricky. Fortunately, Recommendation T-81 from ITU (JPEG Standard) [1] provides two prescribed quantization tables in Annex K. For improved compression, we separate the luminance table from the chrominance table, the chrominance table has larger values as it is less important compared to the luminance component. The 2 tables from Annex K are shown in Tables 2.1 and 2.2.

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

Table 2.1: Luminance quantization table

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

Table 2.2: Chrominance quantization table

2.4 Zigzag Scan

Zigzag is the process of reordering the DCT coefficients according to their frequency. The resulting sequence of coefficients is approximately ordered from the lowest frequency to the highest frequency.

Zigzag scan results in increased number of consecutive zero coefficients after quantization which improves the performance of the run-length encoder considerably. The zigzag order is shown in Figure 2.2

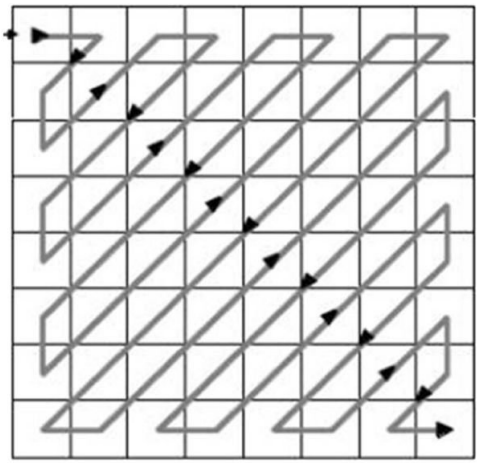


Figure 2.2: Zigzag scan

2.5 DC Difference Encoding

We can further improve the encoding by encoding the difference between a DC coefficient of a block and the previous DC of the block instead of encoding the coefficient as it is. The DC term usually has the biggest coefficient thus takes more bits to encode. The previous DC coefficient of the first block is assumed to be 0.

$$DIFF = DC_i - DC_{i-1} \tag{2.4}$$

2.6 Entropy Encoding

Entropy in information theory is the idea of quantifying the amount of information in a given message. It is a complex but extremely important field to study as it is highly used in various fields such as data compression, machine learning and data transmission.

Entropy is calculated using probabilities as the intuition behind it is the idea of measuring how much surprise there is in an event or message. A rare event is high information or surprising while a common event is low information.

In JPEG, there is more entropy in the low frequency terms and less entropy in the higher frequency terms after DCT, quantization and zigzag scan. In other words, it is extremely common for the data to have multiple consecutive zeros toward the end and this makes it requires less bits to transmit the data.

2.6.1 Run Length Encoding

Run length encoding is the idea of assigning a code that represents the run-length and size of a non-zero value. Run-length is the number of zero values before a non-zero value. The size is the number of bits required to store the value with no losses. In JPEG, the non-zero value is one of the 64 coefficients following quantization and zigzag scan. The DC coefficient is not included to this process.

Run-length encoding is essentially assuming that one specific value or symbol appears often in a data stream. This is a general example of zero suppression techniques. In JPEG, zero is a value that appears very often in a quantized block of coefficients. Especially at the end of the block where most of the high frequency coefficients are set to 0.

2.6.2 Huffman Encoding

Huffman encoding is an algorithm developed by David A. Huffman while he was a Sc.D. student at MIT, and published in the 1952 paper "A Method for the Construction of Minimum-Redundancy Codes" [6]. It is a compression technique that takes symbols (e.g. characters, DCT coefficients, etc.) and encodes them with variable length codes that are assigned according to statistical probabilities. A frequently used symbol will be encoded with a code that takes up only a couple bits, while symbols that are rarely used are represented by symbols that take more bits to encode.

The variable length codes are uniquely decodable prefix codes. Uniquely decodable means each code can not be confused with another code. Prefix code means each value has code before it in the data stream. The output of this algorithm is a table consisting of variable length codes for each run-length/size category.

JPEG uses 4 different Huffman table for each type of coefficients. These type are specifically luminance DC, luminance AC, chrominance DC and chrominance AC coefficients. These tables can be optimally derived by generating a Huffman tree using the exact frequency or probability that a symbol or value will occur. However, Annex K of Recommendation T-81 from ITU (JPEG Standard) [1] provides Huffman tables that has been developed from a large set of video images for each type of coefficient. These tables are not always optimal but they can be useful for many applications.

Chapter 3

Hardware Implementation

3.1 System Overview

The architecture of this project is divided into three parts, as shown in the block diagram shown in Figure 3.1, which are PC, FPGA and ESP32.

The FPGA is the Altera Cyclone V found in the DE1 board. The FPGA is the main part of this project as it contains the JPEG Core. The JPEG Core is an IP core described with VHDL that contains all of the essential components to perform the baseline JPEG algorithm. The JPEG Core takes an 8x8 block of pixels sent from PC to FPGA through SPI (Serial Peripheral Interface) and outputs 512 bits that represents a compressed 8x8 block through SPI back to the PC.

The ESP32 [7] microcontroller is programmed with C++ to essentially be a bridge between the FPGA and PC. Its' job is to receive data from the FPGA using SPI protocol and send it to the PC using USB protocol and vice versa. This is required as SPI is not supported in PC.

PC is responsible of the construction of the final JPEG file. This is done using a software written in python that receives compressed 8x8 block from the ESP32 through USB and writes to the output file.

Figure 3.1: System Overview

3.2 JPEG Core

The JPEG core is the component that does the actual compression. It is implemented as a finite state machine that ensures that the data is valid in every stage. It is composed of many components but the main ones are as shown in the block diagram of Figure 3.2.

The image is first converted from RGB to YCbCr on the PC and then a YCbCr 8x8 block is transferred to the FPGA. The 8x8 YCbCr block becomes input to the DCT unit which outputs 64 signal of 11 bit in size that represents the DCT coefficients. The Quantizer and Zigzag Units manipulate the same 704 bits outputted by the DCT Block and then passes it into the Encoder. The Encoder is the final stage where it encodes the input 704 bits and outputs a variable length output of maximum size of 512 bits. The 512 bits are transferred to the PC through SPI.

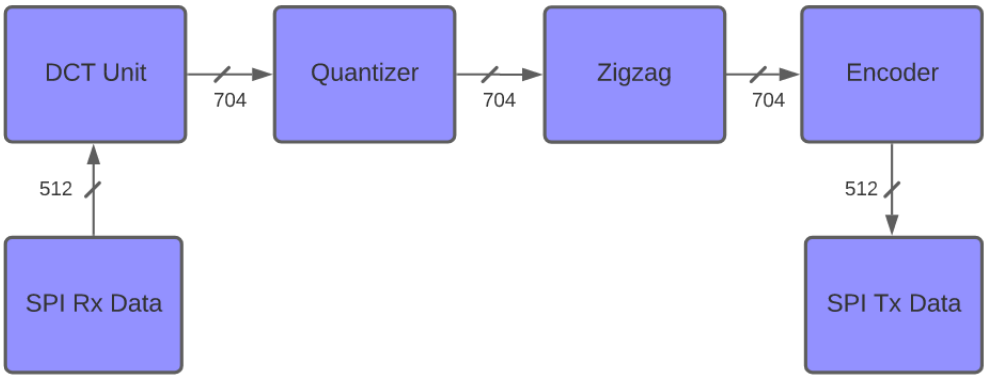


Figure 3.2: Core Overview

3.2.1 Discrete Cosine Transform Unit

The DCT Unit takes advantage of the fact that each DCT coefficient can be calculated separately from the other 63 terms. The unit is designed with 64 DCT multiply-accumulate units as shown in Figure 3.3. The DCT unit implements the Forward 2D-DCT equation described as followed:

$$F(u,v) = \frac{1}{4} C_u C_v \sum_{x=0}^{7} \sum_{y=0}^{7} f(x,y) \cos \frac{(2x+1)u\pi}{16} \cos \frac{(2y+1)v\pi}{16} \tag{3.1}$$

$$C_u, C_v = \begin{cases} \frac{1}{\sqrt{2}} & \text{if } u, v = 0. \\ 1 & \text{otherwise.} \end{cases} \tag{3.2}$$

Each multiply-accumulate unit has its own (u,v) that corresponds to the row and column of each coefficient, making the two cosines, Cu and Cv constant. All of the constants are calculated using the equations of the Forward DCT and implemented as lookup-tables. This makes the entire DCT a simple multiplication of a constant by one of the 64 pixels in 8x8 block and then the result is added to a sum. Each clock cycle, the pixel value changes in raster order and a multiply-accumulate operation is performed. This operation happens simultaneously on all 64 multiply-accumulate

units. After 64 clock cycles, an 8x8 block of coefficients is outputted and ready to be input to the next stage.

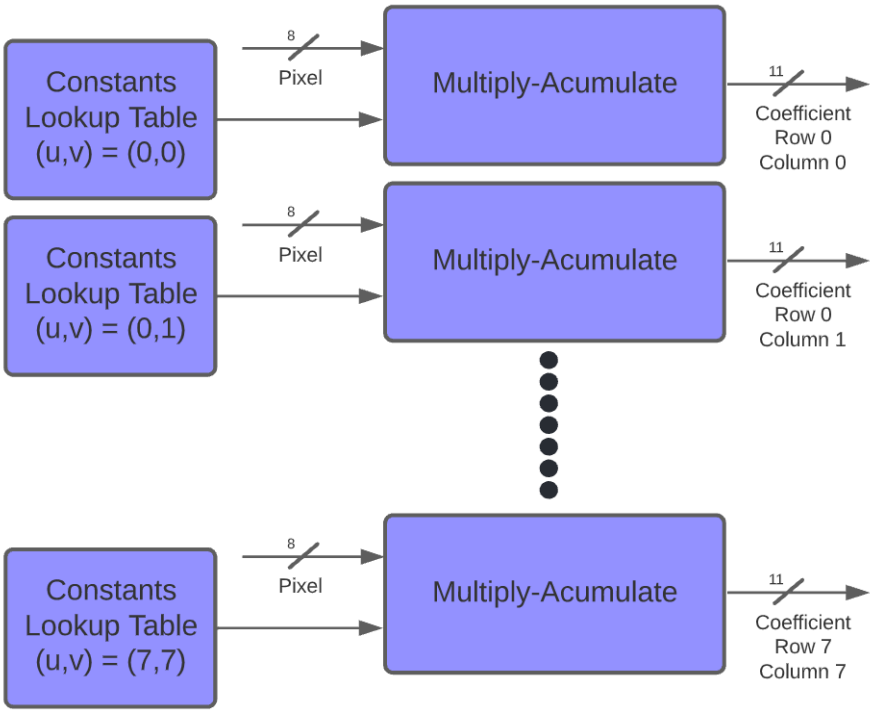


Figure 3.3: DCT Block

3.2.2 Quantizer Unit

The Quantizer unit is combinational circuit that takes 64 coefficients from the DCT Block as input. The quantization tables used are shown in tables 2.1 and 2.2. They are implemented as a lookup table with channel input used to select between the luminance and chrominance tables. The Quantizer unit is represented in Figure 3.4.

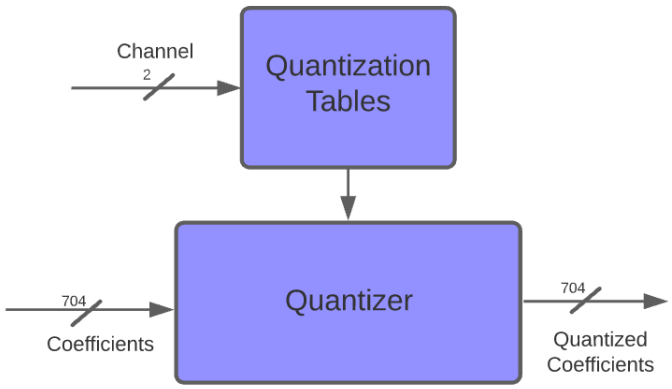


Figure 3.4: Quantizer unit

3.2.3 Zigzag Unit

The Zigzag unit is an extremely simple combinational circuit that reorder the coefficients using multiple multiplexers. The Zigzag unit is represented in Figure 3.5.

Figure 3.5: Zigzag unit

3.2.4 Encoder Unit

The Encoder unit is a combination of the DC difference, run-length encoder and Huffman encoder. The Encoder unit is represented in Figure 3.6.

The previous block's DC coefficient is subtracted by the current block DC coefficient. The encoder unit proceeds to assign a code that corresponds to the size category of the coefficient. The result is a variable length code (VLC) concatenated with the variable length integer (VLI) with a maximum total size of 27 bits. For AC terms, the run-length encoder keeps a counter of how many zero values there is before non-zero value. The run-length encoder compares one of the coefficients with 0 each clock cycle. Once a coefficient is non-zero, the encoder unit proceeds to assign a code that corresponds to the run-length and size of this non-zero coefficient. The result follows the same format as before.

The final result is 512 bit signal that is constructed by concatenating all of the variable length output one after another. This is done by consecutively applying a shift on the 27 bit signal variable length output followed by an OR operation with the 512 bit long final output. The number of clock cycles required to encode a block of coefficients is unpredictable. It ranges from 132 clock cycles up to 640 clock cycles. The best case scenario is when every AC coefficient is zero. The worst case scenario is when every coefficient is non-zero.

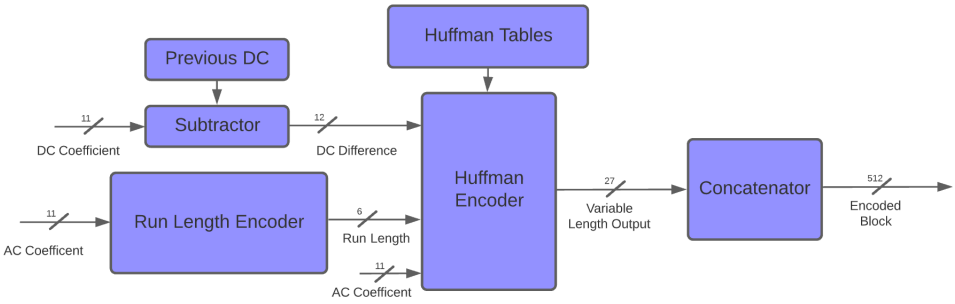


Figure 3.6: Encoder unit

3.2.5 SPI Master on FPGA

The Serial Peripheral Interface (SPI) [8] is a full-duplex communication protocol used primarily in embedded systems. SPI is a synchronous protocol based on a single master/multiple slaves architecture. The basic form of the SPI bus has 4 wires.

- **MOSI :** Master Out Slave In.
- **MISO :** Master In Slave Out.
- **SCLK :** Serial Clock.
- **SS :** Slave Select.

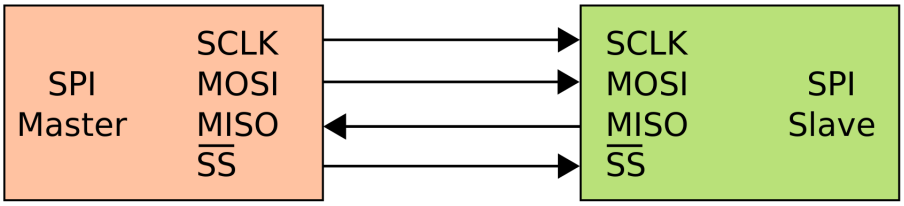


Figure 3.7: Basic SPI Bus

The JPEG Core can successfully compress a bitmap image but transmitting the results to the PC to analyze them is not possible without an interface. SPI was chosen for this instead of USB or I^2C (Inter-Integrated Circuit) for many reasons:

- **Transfer Speed :** 10 times faster than I^2C.
- **Transaction Size :** Larger transaction size than I^2C.
- **Design Complexity :** SPI is extremely simple to implement on the FPGA compared to USB or I^2C.

The SPI Master unit is used to transmit 512 bits outputted from the Encoder unit to the ESP32. SCLK is configured to be a 5MHz with SPI Mode 0(CPOL = 0, CPHA = 0) and MSB first. Once the encoded data is valid, SS goes low signaling the start of the transaction.

3.3 SPI Slave on ESP32

The ESP32 [7] is a low-cost microcontroller designed and manufactured by Espressif Systems. For this project, ESP32 DevKitC development board is used. The ESP32 supports many build systems and programming languages. C/C++ is the most reliable and widely used language for this platform. The ESP32 supports WiFI, Bluetooth, SPI, I^2C , UART (Universal Asynchronous Reciever Transmitter) and many other useful peripherals. All its peripherals are well documented and can be configured to fit the desired application.

The SPI peripheral is easily configurable and very reliable. It supports both master and slave mode. In slave mode, the user can configure SPI Mode (CPOL, CPHA), transaction length and MSB/LSB first.

Once a transaction is completed, the ESP32 proceeds to transmit the same 64 bytes received to the PC via UART. However, PC does not support UART and USB is required to communicate with it. A solution to this problem is to use a UART to USB Converter which is already available on the ESP32 DevKitC development board.

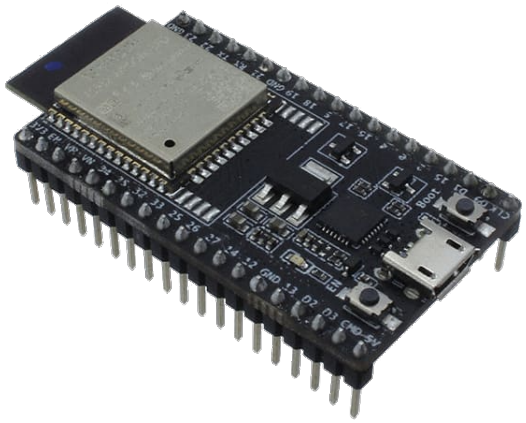


Figure 3.8: ESP32 DevKitC

Chapter 4

Results and Discussion

4.1 Results

The performance of a JPEG compression algorithm varies a lot as different input images will have different compression ratio and speed. This is directly caused by the entropy encoder. Depending on the complexity of an 8x8 block, it can take from 8μs up to 28μs per block. Similarly, the size of the encoded data of each block also ranges from 2 bits up to 1664 bits. Besides the input image data, changing any of the quantization tables or the Huffman tables will also cause the performance to change.

Another factor that limits compression speed of an image is the maximum throughput of the interface chosen. In this case, The JPEG core has to wait 50ms between each block to start the compression on the next one and this is mainly due to the software overhead of the ESP32. This means that the compression time per block is 50ms instead of 8μs up to 28μs per block. The JPEG core uses around 82% of the FPGA's logic elements and this is mainly due to the parallel design of the DCT unit.

Flow Status	Successful - Thu May 5 01:41:24 2022
Quartus Prime Version	21.1.0 Build 842 10/21/2021 SJ Lite Edition
Revision Name	jpeg
Top-level Entity Name	jpeg
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	26,347 / 32,070 (82 %)
Total registers	4510
Total pins	15 / 457 (3 %)
Total virtual pins	0
Total block memory bits	737,280 / 4,065,280 (18 %)
Total DSP Blocks	0 / 87 (0 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

Figure 4.1: Quartus Prime Compilation Report

Figure 4.2 is an extremely simple 16x16 image that was used as a test image. This image was used instead of large and complex images because of the limited SPI throughput and debugging simplicity. The BMP (bitmap) file size is 824 and JPEG compressed file by the FPGA is 644 bytes. However, the raw pixel data on the BMP file is 768 byte with 56 bytes of BMP header data. Similarly, the raw pixel data on the JPEG file is 19 bytes only with 625 bytes of header. The header data is 625 bytes because it includes the entirety of the Huffman tables including the non used codes. If optimized Huffman tables were used, the total image file would be reduced significantly. However, optimizing Huffman tables is a tedious process that requires extra complicated hardware that builds Huffman trees based on input data and traverse each one of them. Looking only at the pixel data, the compression ratio is 40:1 for this image. Different input with complex shapes will result in a lower compression ratio. Table 4.1 summarizes the results.

File Format	File Size	Header Size	Pixel Data Size
BMP	824 byte	56 byte	768 byte
JPEG	644 byte	625 byte	19 byte

Table 4.1: Compression Results Summary

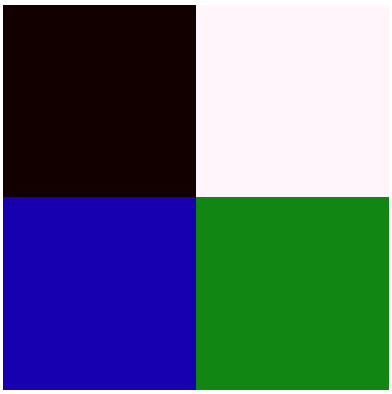


Figure 4.2: Test image used for JPEG compression

4.2 Discussion

Measuring the speed of the JPEG core independently, 8us up to 28us per block is relatively fast but this comes at many trade-offs. JPEG is an extremely flexible and configurable algorithm. Baseline JPEG is one variant described by the JPEG Standard. Different encoders and DCT types can be used and each variant have its own advantages and disadvantages. Even baseline JPEG can be configured by changing quantization tables and Huffman tables. Achieving this level of flexibility is hard when implemented purely as a digital circuit and extremely expensive in terms of logic elements.

Another trade-off that comes with implementing JPEG as digital circuit and specifically the entropy encoder is the design complexity. Entropy encoder is hard to debug

and this is mainly due to the nature of the output. It only takes 1 bit to be wrong out of the entire bit-stream for the image to become completely corrupted. There is usually no obvious hint to the problem just by looking at the output encoded data. There are cases where an error can go undetected as it can only occur for a specific type of input data. Timing related issues such as meta-stability can also be the reason why the design failed. In such case, having limited debugging tools with a wide range of potential failure reasons makes it extremely hard to develop.

Implementing the entropy encoder as a software program has a lot of advantages with little disadvantages over the FPGA implementation. Besides having access to a wide range of debugging tools and eliminating timing issues, the entropy encoder can be highly configurable. This allows the ability to change Huffman tables to better match the input data. Optimizing Huffman tables reduces the size of the image significantly. The only disadvantage of implementing the encoder as a software is losing speed compared to the FPGA. However, the loss in speed is not significant as the encoding process is inherently sequential which is ideal for microprocessors with higher clock frequencies.

On the other hand, DCT and quantization benefited the most when implemented as a digital circuit. They are considered an embarrassingly parallel workload which means it can be separated in to a number of parallel computations. The number of parallel computations can theoretically vary from 1 to 64. This parameter can be chosen based on the target FPGA as each one has different number of logic elements and DSP (Digital Signal Processor) resources.

Chapter 5

Conclusion and Future work

JPEG compression algorithm is a combination of both parallel and sequential algorithms. This means that half of the algorithm will be perform better when implemented as hardware and the other half will be perform better when implemented as software on a higher clock rate microprocessor. Specifically, DCT and quantization are ideal to be implemented as an IP core while entropy encoding will perform better as a software implementation.

Future work will involve:

- Designing a fully featured JPEG CODEC on the FPGA.
- Creating a custom Linux image for the HPS (Hard Processor System).
- Using AXI (Advanced eXtensible Interface) instead of SPI to increase throughput between FPGA and HPS.
- Implementing entropy encoding/decoding on the HPS as a software program.
- Expand the project to Motion-JPEG.

Bibliography

[1] ITU. “T.81 : Information technology - Digital compression and coding of continuous-tone still images - Requirements and guidelines”. In: (September 1992).

[2] *”Introduction to Data Compression (The Morgan Kaufmann Series in Multimedia Information and Systems) 4th edition”*. ”Burlington, Massachusetts”: ”Morgan Kaufmann”, October 30, 2012.

[3] *Eye, Brain, and Vision (Scientific American Library)”*. ”New York”: ”W. H. Freeman; 2nd edition”, May 15, 1995.

[4] ITU. “T.871 : Information technology - Digital compression and coding of continuous-tone still images: JPEG File Interchange Format (JFIF)”. In: (September 11, 2012).

[5] N. Ahmed, T. Natarajan, and K.R. Rao. “Discrete Cosine Transform”. In: *IEEE Transactions on Computers* C-23.1 (1974), pp. 90–93. DOI: 10.1109/T-C.1974.223784.

[6] David A. Huffman. “A Method for the Construction of Minimum-Redundancy Codes”. In: *Proceedings of the IRE* 40.9 (1952), pp. 1098–1101. DOI: 10.1109/JRPROC.1952.273898.

[7] Espressif. *ESP32*. URL: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/index.html.

[8] Texas Instruments. “KeyStone Architecture Serial Peripheral Interface (SPI)”. In: (March 2012).