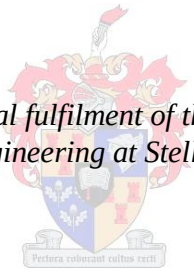
The Design and Implementation of a Video Compression Development Board

by
Suliman Alalait



Thesis presented in partial fulfilment of the requirements for the degree Master of Science in Engineering at Stellenbosch University

Supervisor: Mr Willem Smit
Department of Electrical Engineering

March 2011

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: March 2011

ABSTRACT

This thesis describes the design and implementation of a video compression development board as a standalone embedded system.

The board can capture images, encode them and stream out a video to a destination over a wireless link. This project was done to allow users to test and develop video compression encoders that are designed for UAV applications.

The board was designed to use an ADSP-BF533 Blackfin DSP from Analog Devices with the purpose of encoding images, which were captured by a camera module and then streamed out a video through a WiFi module. Moreover, an FPGA that has an interface to a logic analyzer, the DSP, the camera and the WiFi module, was added to accommodate other future uses, and to allow for the debugging of the board.

The board was tested by loading a H.264 BP/MP encoder from Analog Devices to the DSP, where the DSP was integrated with the camera and the WiFi module. The test was successful and the board was able to encode a 2 MegaPixel picture at about 2 frames per second with a data rate of 186 Kbps. However, as the frame rate was only 2 frames per second, the video was somewhat jerky.

It was found that the encoding time is a system limitation and that it has to be improved in order to increase the frame rate. A proposed solution involves dividing the captured picture into smaller segments and encoding each segment in parallel. Thereafter, the segments can be packed and streamed out. Further performance issues about the proposed structure are presented in the thesis.

ACKNOWLEDGEMENTS

I would like to thank my supervisor Mr. Willem Smit for his time, guidance and for providing all the necessary equipment to complete the project.

To Peter Koeppen and Clive Bowerman for their ideas and advice.

To Bernard Visser for his friendship and advice.

To Dr. Sami Alhamidi, I say; thank you for making this happen.

To Sharef Neemat for the unlimited support throughout writing my thesis and for Regine Lord for editing my thesis.

I would also like to thank my parents for their support and patience throughout the duration of my studies.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS AND ABBREVIATIONS

3D	3 Dimension
ACC	Average Camera Cycles
ADSP-BF533	BF533
ADSP-BF548	BF548
ADSP-BF561	BF561
AEC	Average Encoding Cycles
AFS	Average Frame Size
AVC	Advanced Video Coding
AWC	Average WiFi Cycles
BGA	Ball Grid Array
BP	Baseline Profile
CABAC	Context Adaptive Binary Arithmetic Coding
CAVLC	Context Adaptive Variable Length Coding
CBR	Constant Bit Rate
CCD	Charge Coupled Device
CCLK	Core CLK
CLK	Clock
CLL	Critical Line Length
CMOS	Complementary Metal Oxide Semiconductor
CPLD	Complex PLD
DCT	Discrete Cosine Transform
DMA	Direct Memory Access
DPLD	Decode PLD
DSP	Digital Signal Processing

ACRONYMS AND ABBREVIATIONS

EBIU	External Bus Interface Unit
EMIF	External Memory Interface
FPGA	Field Programmable Gate Arrays
Fps	frames per second
GND	Ground
GOB	Groups of Blocks
GPIO	General Purpose I/O
HW	Hardware
IC	Integrated Circuit
I/O	Input/Output
ISO	International Organization for Standardization
ISP	In System Programmability
ISR	Interrupt Service Routine
ITU-R	International Telecommunication Union-Radiocommunications
ITU-T	International Telecommunication Union-Telecommunication
JPEG	Joint Photographic Experts Group
JTAG	Joint Test Action Group
LANs	Local Area Networks
LED	Light Emitting Diode
LQFP	Low-profile Quad Flat Package
MC	Motion Compensation
MDMA	Memory DMA
ME	Motion Estimation

ACRONYMS AND ABBREVIATIONS

MP	Main Profile
MPEG	Moving Picture Experts Group
MV	Motion Vector
PCB	Printed Circuit Board
PLD	Programmable Logic Device
PLL	Phase Locked Loop
PPI	Parallel Peripheral Interface
PSD4256G6V	PSD
Q&DCT	Quantization and Discrete Cosine Transform
QP	Quantization Parameter
RF	Radio Frequency
RGB	Red, Green and Blue
RLC	Run Length Coding
SCCB	Serial Camera Control Bus
SCLK	System CLK
SDRAM	Synchronous Dynamic Random Access Memory
SPI	Serial Peripheral Interface
SPORT	Serial Port
SRAM	Static Random Access Memory
SW	Software
TCP	Transmission Control Protocol
TV	Television
UART	Universal Asynchronous Receiver Transmitter
UAV	Unmanned Aerial Vehicle

ACRONYMS AND ABBREVIATIONS

UDP	User Datagram Protocol
USB	Universal Serial Bus
UXGA	Ultra eXtended Graphics Array
VBR	Variable Bit Rate
VCEG	Video Coding Experts Group
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuits
VLC	Variable-Length Coding
WiFi	Wireless Fidelity
WQXGA	Wide Quad eXtended Graphics Array
XGA	eXtended Graphics Array
YCbCr	Luminance, Chrominance-Blue and Chrominance-Red
YUV	YCbCr

1. Chapter 1: Introduction

The objective of this project is to design a video compression development board for an Unmanned Aerial Vehicle (UAV) application. This chapter covers the background of the main system components, namely the video compression concept and the camera sensors. The project requirements are also discussed in this chapter.

1.1 Background

1.1.1 Video Compression Concept

Due to the limitation in digital video transmission bandwidth and storage space, digital video compression algorithms are needed to achieve lower costs and the desired performance.

The basic idea of video compression is to remove redundancy in the video stream, so that the entire video can be transmitted in fewer bits [4] and [5].

A video clip consists of a series of still images or frames. There are two kind of redundancy in videos, namely, spatial and temporal. Spatial redundancy refers to the redundant information that is contained in a single frame, while temporal redundancy represents the redundant information across multiple frames [4], [5] and [6]. Figure 1.1 is an overview of a generic video compression encoder [8].

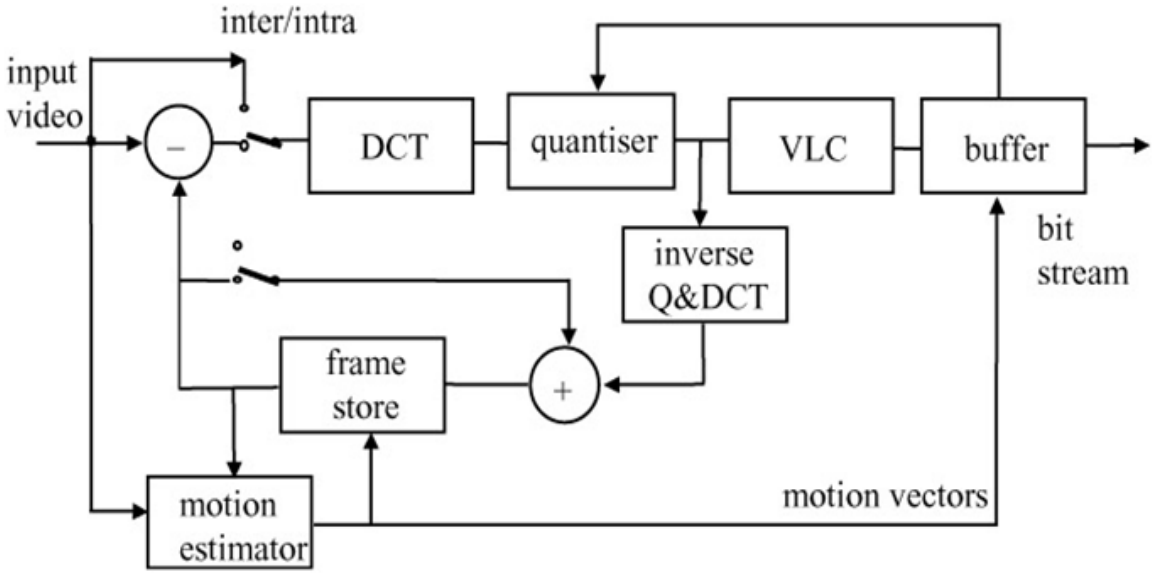


Figure 1.1: From [8]. Generic video compression Encoder. DCT is the Discrete Cosine Transform, VLC is the Variable-Length Coding and Q&DCT is the Quantization and Discrete Cosine Transform.

1.1.1.1 Spatial Redundancy

Spatial redundancy is also known as Intra-frame redundancy; it works similarly to the Joint Photographic Experts Group (JPEG) format. JPEG is a so-called lossy compression algorithm, which eliminates the information to which the human eye is not sensitive (in other words, high frequency detail), by converting frames to the frequency domain and removing high frequency detail. This means that some information is lost from the original image [4], [5] and [6]. How JPEG works is described briefly below. The JPEG algorithm in general consists of four steps:

- Converting the RGB image format to the YCbCr format

  First, the image must be prepared for compression by being converted from Red, Green and Blue (RGB) to Luminance, Chrominance-Blue and Chrominance-Red (YCbCr). Each pixel in RGB (0-255, 0-255, 0-255) is 24 bits, and 0-255

represents the intensity for each colour. Thus, in RGB there are 16 million different combinations of colours, most of which cannot be distinguished by the human eye: this implies that it is redundant information which could be omitted. It has been found that the human eye is more sensitive to brightness (Luminance) than to colours (Chrominance) [7]. Therefore, the YCbCr format separates the luminance (Y) that can be represented in a high resolution, from the chrominance (Cb and Cr), which can be subsampled. International Telecommunication Union – Radiocommunications (ITU-R) BT.601 standard recommends the following equations for transforming RGB to YCbCr [4] and [5]:

$$Y = 0.299(R) + 0.587(G) + 0.144(B)$$

$$Cb = 0.564(B - Y)$$

$$Cr = 0.713(R - Y)$$

For Ultra eXtended Graphics Array (UXGA) video at 1600x1200 pixels, there are ((1600x1200)x3)x8 = 46,080,000 bits in RGB form. In YCbCr luminance is sampled one-for-one based on RGB form; therefore, there are 1600x1200 luminance samples. If one were to subsample the red and blue chrominance values by 2, there would be (800x600)x2 chrominance samples. The total number of bits in the YCbCr format are thus ((1600x1200) + ((800x600)x2))) x 8 = 23,040,000 bits, which is less than 46,080,000 bits in RGB form.

- Discrete Cosine Transform (DCT)

  After converting the image to YCbCr, it must be divided into small blocks, with each block normally made up of 8 x 8 pixels. Then, each block is transformed into the frequency domain using the DCT equation, based on the fact that human eyes are more sensitive to the information in the low frequency range than the

high frequency range. DCT thus identifies and separates this information from the rest of the information [4], [5] and [6].

- Quantization

  Many bits are omitted in this step. Quantization represents the DCT high frequency coefficients with fewer bits than the low frequency coefficients. Therefore, dequantizing the quantized block will be close but not identical to the original block. It has been found that, after quantization, most of the DCT high frequency coefficients are zeros [4], [5] and [6].

- Encoding

  Different kinds of encoding are available in the market. Two are presented here, namely, Run Length Coding (RLC) and Variable-Length Coding (VLC). RLC converts a sequence of values into a sequence of symbols (Run, Value). Value is the value, while Run is the count of this value (see Figure 1.2).

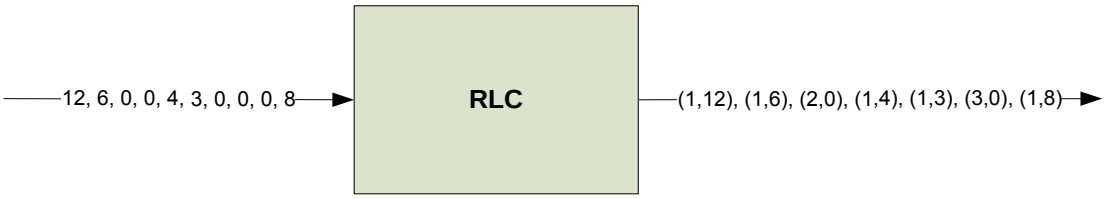


Figure 1.2: From [9]. General RLC

Different formats could be used in RLC. For example, if the input has a long run of zeros, as may be the case with JPEG after Quantization, then a different format is used: Value would be used to encode only the nonzero values, while Run would be used to encode the number of zeros preceding that value (see Figure 1.3).

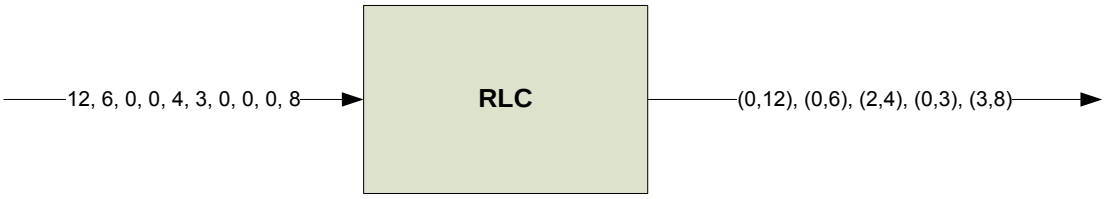


Figure 1.3: RLC in JPEG

The next step is VLC, which converts symbols to a series of codewords. The symbols that occur most frequently are encoded with a codeword consisting of fewer bits, while the symbols that occur less frequently are encoded with a codeword consisting of more bits. Therefore, with VLC the required average number of bits to encode a symbol is less, which leads to fewer bits for encoding the whole image or frame [4], [6] and [9].

1.1.1.2 Temporal Redundancy

Temporal redundancy is also known as Inter-frame redundancy. The three most popular encoded frames are introduced here, before explaining the technique that will be used herein to reduce Temporal Redundancy [4] and [5]:

I-frames: I-frames are Intra frames; they are encoded by using information from the same picture, which means that only the spatial redundancy is reduced and not the temporal redundancy. I-frames provide random access points along the stream because the decoder does not need any reference frames to reconstruct I-frames [4], [5] and [10].

P-frames: P-frames are forward predicted frames. In other words, they can be predicted from the last I- or P-frame. They are encoded by comparing the present frame with the past frame, and the differences between them are then encoded

and transmitted. The decoder would not be able to reconstruct P-frames without a reference frame. P-frames provide more compression than I-frames because only the difference between two frames is being encoded and transmitted [4], [5] and [10].

B-frames: B-frames are bidirectional, in that they can be predicted from the previous I- or P-frame as well as from the next I- or P-frame. The encoding would thus need to compare both past and future frames to compute the difference between them and the present frame. B-frames provide the most compression of all three types of frames, but require more processing time [4], [5] and [10].

P-frames and B-frames are Inter frames (unlike I-frames, which are Intra frames), because they reduce the temporal redundancy. Prediction is the technique that is being used to reduce temporal redundancy. The prediction process comprises of two processes, namely, Motion Estimation (ME) and Motion Compensation (MC) [11]. ME searches for the best Motion Vector (MV) that points to a block of pixels, usually called macro block, in the previous or next frame that is the closest identical to the present macro block. Then, MC calculates the difference between the two matching MBs by using the above MV. The new macro block then can be processed via DCT, Quantization and RLC and VLC before it is transmitted across the channel, along with the MV [11].

High compression is achieved if ME finds the ultimate MV for all the blocks, in other words, if the blocks match each other closely. However, ME requires more processor cycles than any other step in video compression algorithms [6] and [11].

The selection of a ME technique affects the processor performance and the video quality. As a result, commercial available encoders keep details of how the ME is implemented classified [6].

1.1.2 Camera Sensors

Today, camera sensors are either Charge Coupled Devices (CCD) or Complementary Metal Oxide Semiconductors (CMOS).

CCD sensors work by converting light into electronic charges, and then transferring these electronic charges to an output amplifier to convert them to voltage. The CCD structure consists of X number of parallel columns (parallel registers), which represent the CCD image area; in each column, there are Y number of pixels. The CCD structure also consists of Z number of serial registers, which are horizontal to the columns and the output amplifier at the end of the serial registers. The electron charges in the last pixel of each column are transferred to the serial registers. Thereafter, the electron charges in the serial registers are transferred one at a time to the output amplifier to convert them to voltage. As soon as the output amplifier has finished converting all the electron charges in the serial registers, then the steps are repeated, starting by transferring the electron charges in the next pixels to the serial registers from each column. CCD sensors are analogue chips because the pixels on the chip cannot be digitized. The step after the output amplifier is made up of a circuit that converts the voltage to a digital signal in the camera; once all pixels have been digitized, they can be stored as a single image file. The basic CCD structure is illustrated in Figure 1.4 [1] and [2].

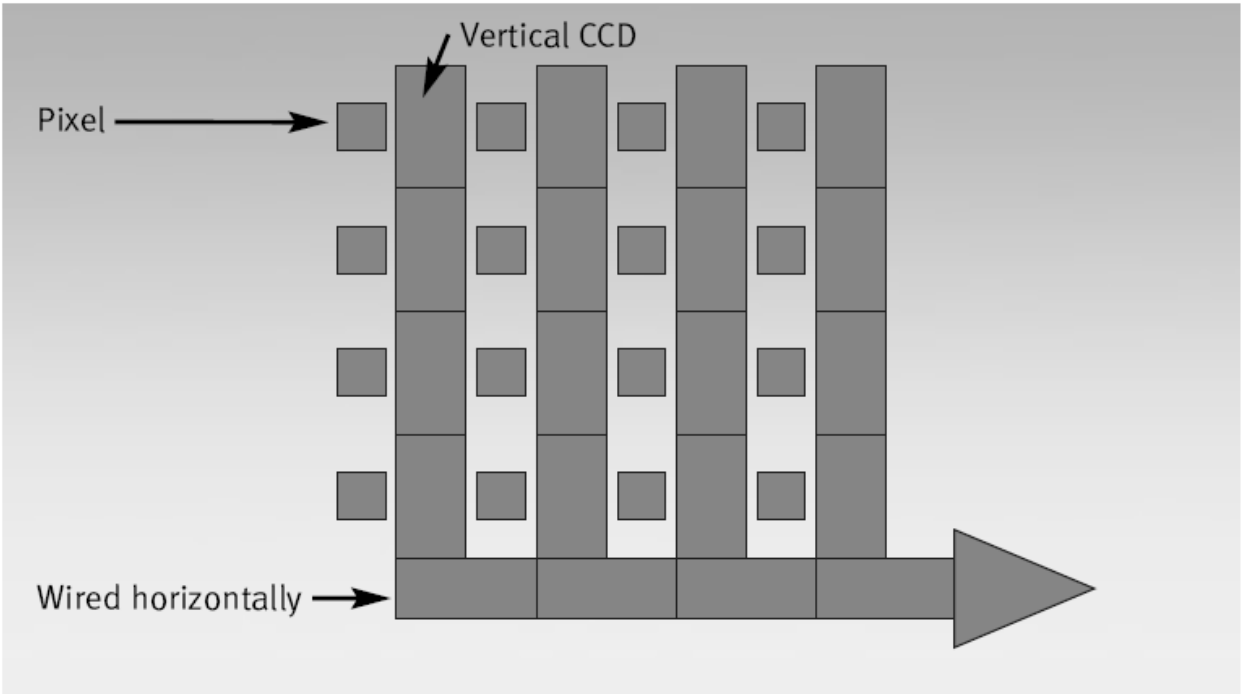


Figure 1.4: Modified from [2]. CCD structure

The numbers of electrons in each pixel are affected by the intensity of light and by exposure time. Therefore, when both the intensity of light and the exposure time are high, the number of electrons will be high, and vice versa [3].

CMOS sensors have exactly the same structure as CCD sensors; except that electrons charges are amplified and converted to voltage inside each pixel and then transported across the chip (see Figure 1.5). Thereafter, an additional circuit inside the chip converts the voltage to a digital signal [1] and [2].

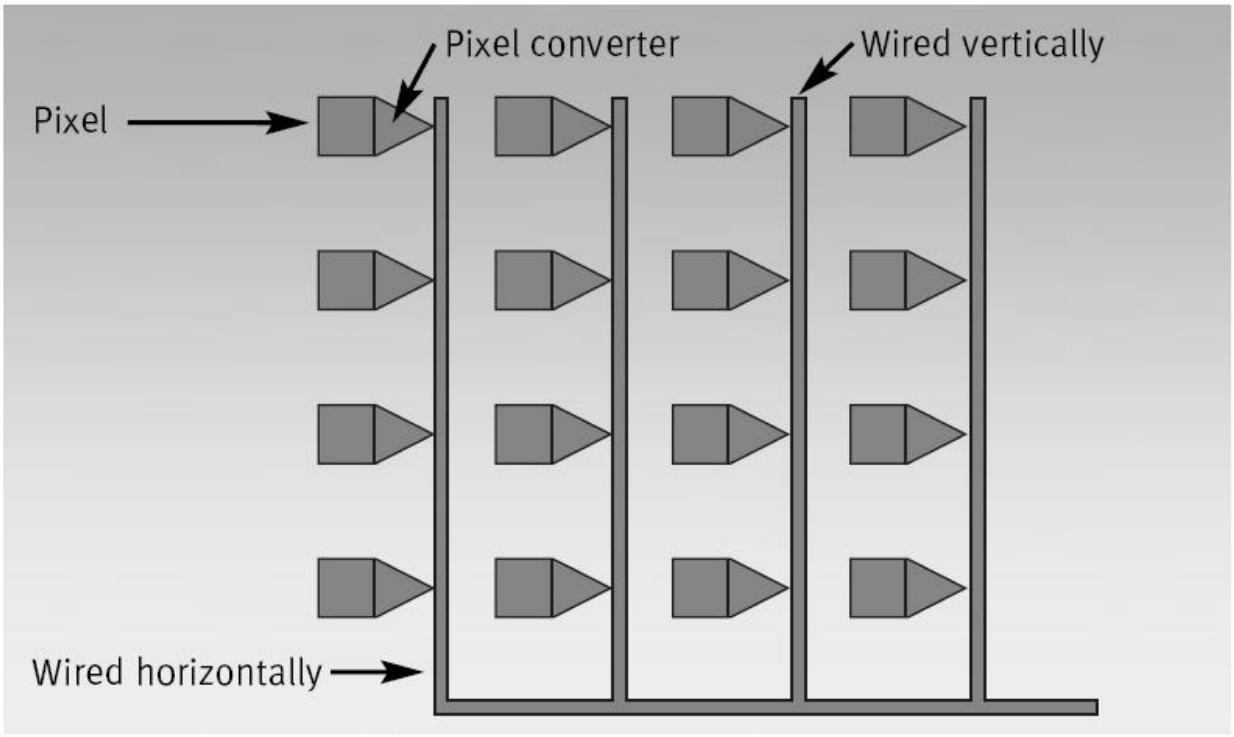


Figure 1.5: Modified from [2]. CMOS structure

CMOS sensors consume less power than CCD sensors, because they need very little power to transfer the voltage. As a result, even when the size of the CMOS sensor is increased, it consumes the same amount of power as a smaller sensor, as long as there is no increase in the numbers of channels [2]. A CCD sensor, however, needs power to transfer electronic charges across the chip; therefore, the larger the size of the sensor, the more power it needs (see Figure 1.6). The more power the CCD needs, the greater the advantage of using a CMOS sensor instead. A CCD could consume as much as 100 times more power than an equivalent CMOS [2].

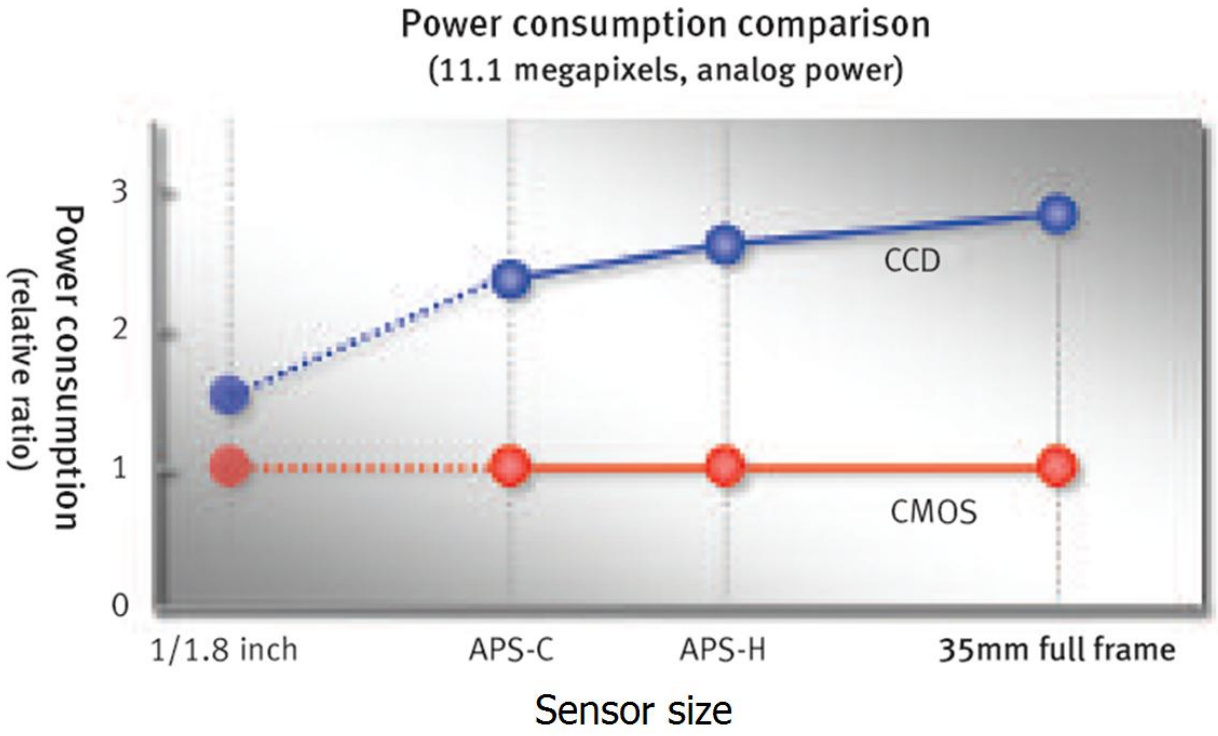


Figure 1.6: Modified from [2]. Comparison between CCD and CMOS in terms of size and power consumption

CCD sensors create low noise images because there is less on-chip circuitry; they also create high quality images because of a special manufacturing process. CMOS sensors, however, have numerous transistors next to the photodiode, which leads to poor image quality “this might not be true for the past year as Sony announced their Exmor sensor, which uses CMOS technology and give high quality images”. Conversely, though, CMOS chips can be cheaply manufactured at any silicon production line because they use standard Integrated Circuit (IC) production technology whereas CCD needs special machines [2].

1.2 Project Requirements

The board is required to carry out three main tasks: capturing images, encoding them, and transmitting them over a radio link. Figure 1.7 depicts a high-level system block diagram.

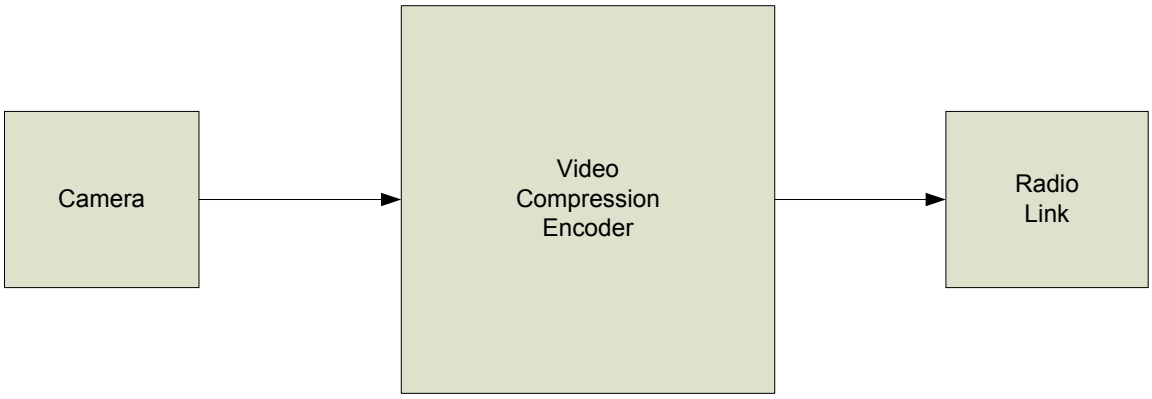


Figure 1.7: High-level system block diagram

1.2.1 Camera and Video Compression Encoder Requirements

The target UAV uses a 60° degree lens, flies at a speed of 50 km/h and at a height of 100 m above the ground. The camera picture size and the amount of change between one picture and the other have to be calculated while taking these numbers into consideration in order to choose the camera and to design the encoder.

The picture size is measured in pixels. The more pixels make up a picture of a particular size, the more accurate and detailed that picture will be. Put differently, the more pixels make up a particular image, the smaller the pixels need to be, and thus the more accurate the picture will be. There are some standard picture sizes, such as eXtended Graphics Array (XGA), UXGA and Wide Quad eXtended Graphics Array (WQXGA). The

number and size of the pixels for each of these standard picture sizes had to be measured, before the optimal size for this project could be determined. The following equation was used to measure the pixel size:

$$One\ Pixel = \frac{H \times \tan\left(D \ \frac{\pi}{180}\right)}{X} \qquad 1.1$$

Where H is the UAV height, D is half of the lens degree and X is the number of pixels in one row divided by 2 (see Figure 1.8).

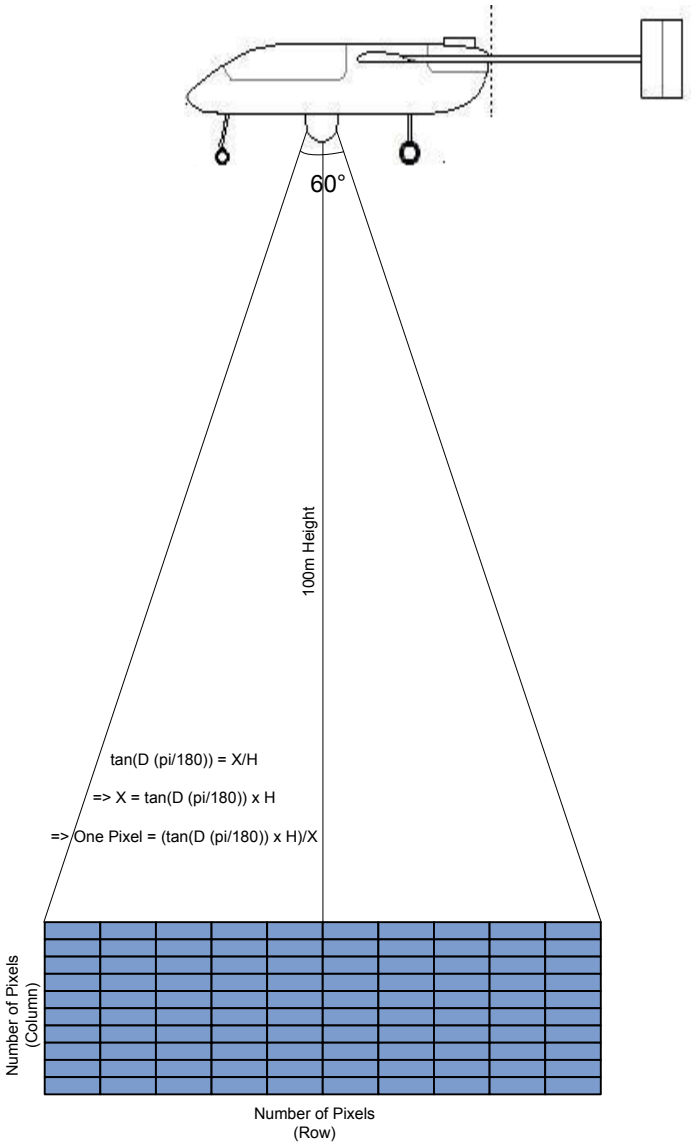


Figure 1.8: Relationship between picture size, UAV lens degree and UAV height

- XGA is (1024 (row) x 768 (column)), therefore X will be equal to 1024/2. So, one pixel will be (100 tan (30 $\frac{\pi}{180}$)) / 512 = 0.1128 m. As 0.1128 m is a large size, the picture will look blurry and lack detail.
- UXGA is (1600 x 1200), therefore X will be equal to 1600/2. So, one pixel will be (100 tan (30 $\frac{\pi}{180}$)) / 800 = 0.0722 m. The picture with 0.0722 m will be clearer than one with 0.1128 m.

- WQXGA is (2560 x 1600), therefore X will be equal to 2560/2. So, one pixel will be (100 tan (30 $\frac{\pi}{180}$)) / 1280 = 0.0451 m. The picture with 0.0451 m will be clearer than XGA and UXGA but it will require more processing time.

From visualisation, the UXGA picture from a 100 m height is acceptable. Given the above, the UXGA is the best option for this project because the picture quality would be acceptable and because it does not require high processing time.

Therefore, it was decided to choose a camera that would be able to capture 2 MegaPixel frame sizes at the quickest frames per second (fps) available and to design an encoder that would be able to encode 2 MegaPixel frame sizes at the quickest possible fps rate; this is because it was found that video jerkiness has an inverse relationship with the frame rate; put differently, a higher frame rate means a smoother video. Moreover, it was decided that a CMOS camera would be the best option, because cost and power consumption were more important than high quality images (see Section 1.1.2) and because most MegaPixel sensors use CMOS technology.

From equation (1.1), one can see that the pixel size has an inverse relationship with the number of pixels in one row and a direct correlation with the UAV height and lens degree (see Figure 1.9).

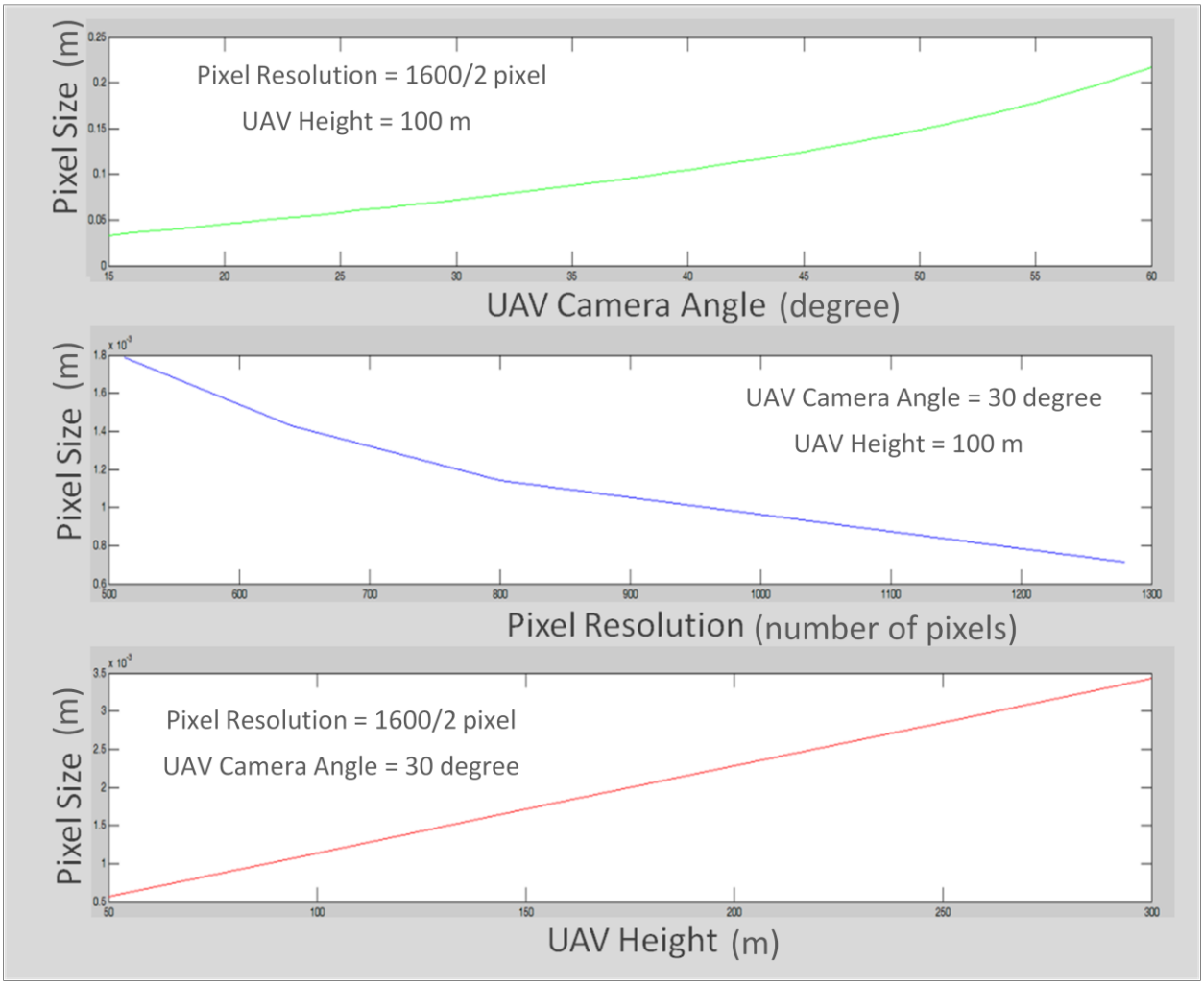


Figure 1.9: Relationship between pixel size, number of pixels in one row, UAV height and UAV lens degree.

To specify the amount of change between one picture and the other, it is necessary to know when a new scene appears. UXGA frames can be divided into Groups of Blocks (GOB), where each GOB consists of 100 macro blocks and each macro block consists of 16x16 pixels. Therefore, UXGA has 75 GOB and 7500 macro blocks (see Figure 1.10) [11].

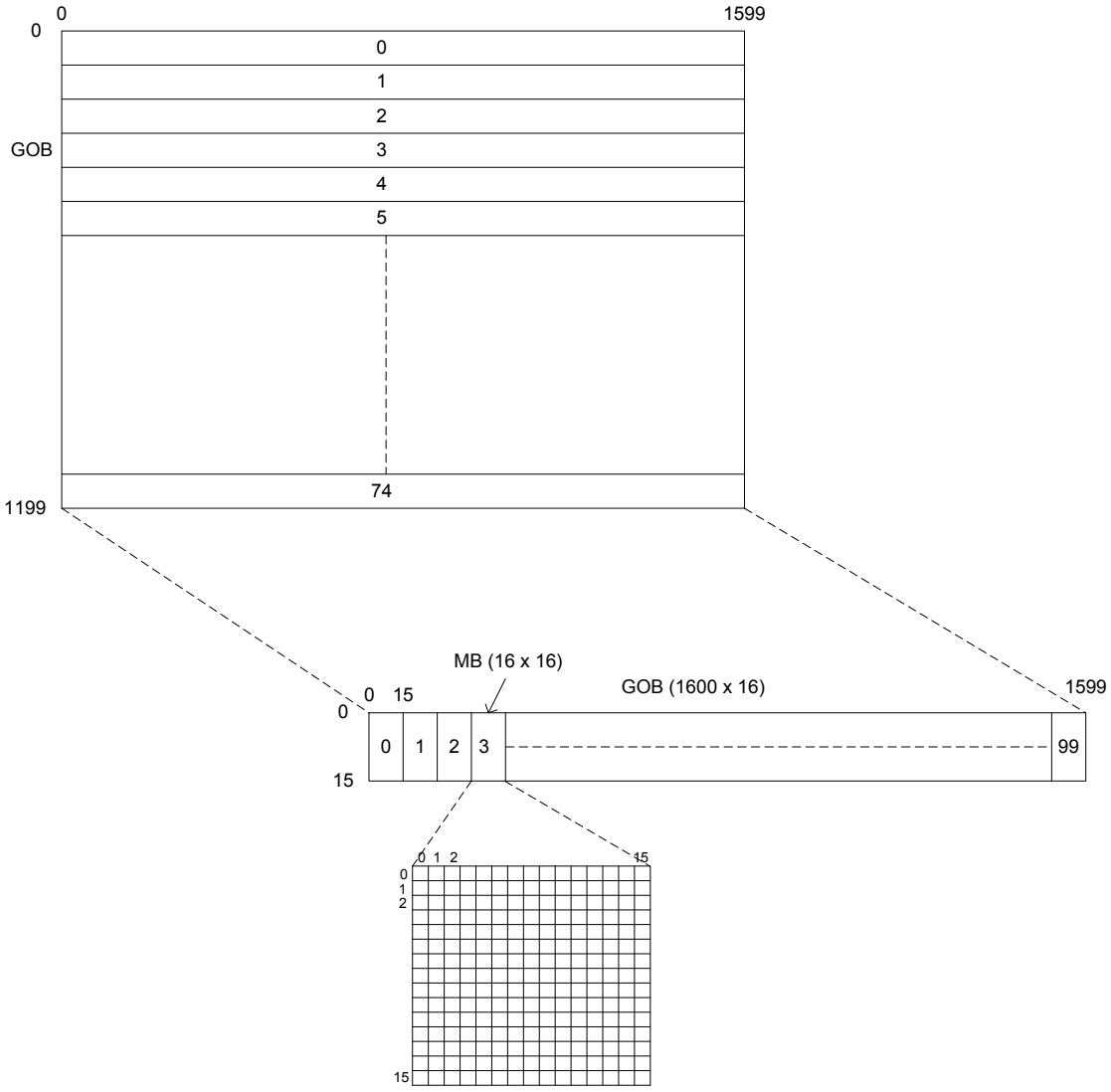


Figure 1.10: Modified from [11]. Division of UXGA

The relevant mathematical calculations are presented below:

1) One UXGA frame is 1600 pixels long along the X-axis, and each pixel is 0.072 m => 1600(0.072) = 115.2 m.
2) One macro block is 16 pixels long => 16(0.072) = 1.152 m.
3) One UXGA frame has 100 macro blocks in one GOB => 115.2m = 100 macro blocks.
4) The UAV flies 50 km in one hour => in one second, it flies

$$\frac{(50\ (1000))m}{(60\ minutes \times 60\ seconds)seconds} = 13.88\ m/s.$$

From the above, the following can be concluded (see Figure 1.11):

- There is a new scene of 13.88 m every second.
- The number of macro blocks that will be moved from one frame to the following frame in one GOB every one second is (115.2 – 13.88)/1.152 ≈ 88 macro blocks, and thus for the whole frame 88 x 75 = 6600 macro blocks.
- The number of new macro blocks in one GOB every one second is 13.88/1.152 ≈ 12 macro blocks, and thus for the whole frame 12 x 75 = 900 macro blocks.

The 6600 macro blocks in the new picture would be encoded as Inter-block which means more compression (see Section 1.1.1.2), whereas most of the new 900 macro blocks would be encoded as Intra-block depending on the ME. In other words, ME would search for the new macro blocks in the old picture to find corresponding macro blocks, and most probably it will not find any unless if there was an object which has moved from the old picture to the new picture. Consequently, high compression rate has an inverse relationship with the number of the new macro blocks.

$$Number\ of\ the\ new\ macro\ block = \frac{UAV\ speed}{16\ (Pixel\ size)}$$

$$Pixel\ size\ from\ equation\ (1.1) = \frac{H \tan (D\ \frac{\pi}{180})}{No.\ of\ pixels\ along\ the\ X_{axis}}$$

$$=> \quad Number\ of\ the\ new\ macro\ block = \frac{\text{UAV speed}}{16\ (\frac{H \tan\left(D\ \frac{\pi}{180}\right)}{No.of\ pixels\ along\ the\ X_{axis}})}$$

$$=> \quad Number\ of\ the\ new\ macro\ block = \frac{No.of\ pixels\ long\ the\ X_{axis}\ \times\ \text{UAV speed}}{16\ (H \tan\left(D\ \frac{\pi}{180}\right))}$$

From all of the above, the following can be concluded: The bigger the degree of the lens, the bigger the area it can photograph or film. Also, the higher the UAV flies (the higher its altitude above the ground), the more space the lens can capture. In other words, a 60° lens would capture less area at a height of 100 m than at 300 m. Therefore, the higher the UAV flies, or the bigger the degree of the lens is, the larger the picture size must be. The slower or the higher the UAV flies, the less new macro block will be which would result in a higher compression rate and vice versa.

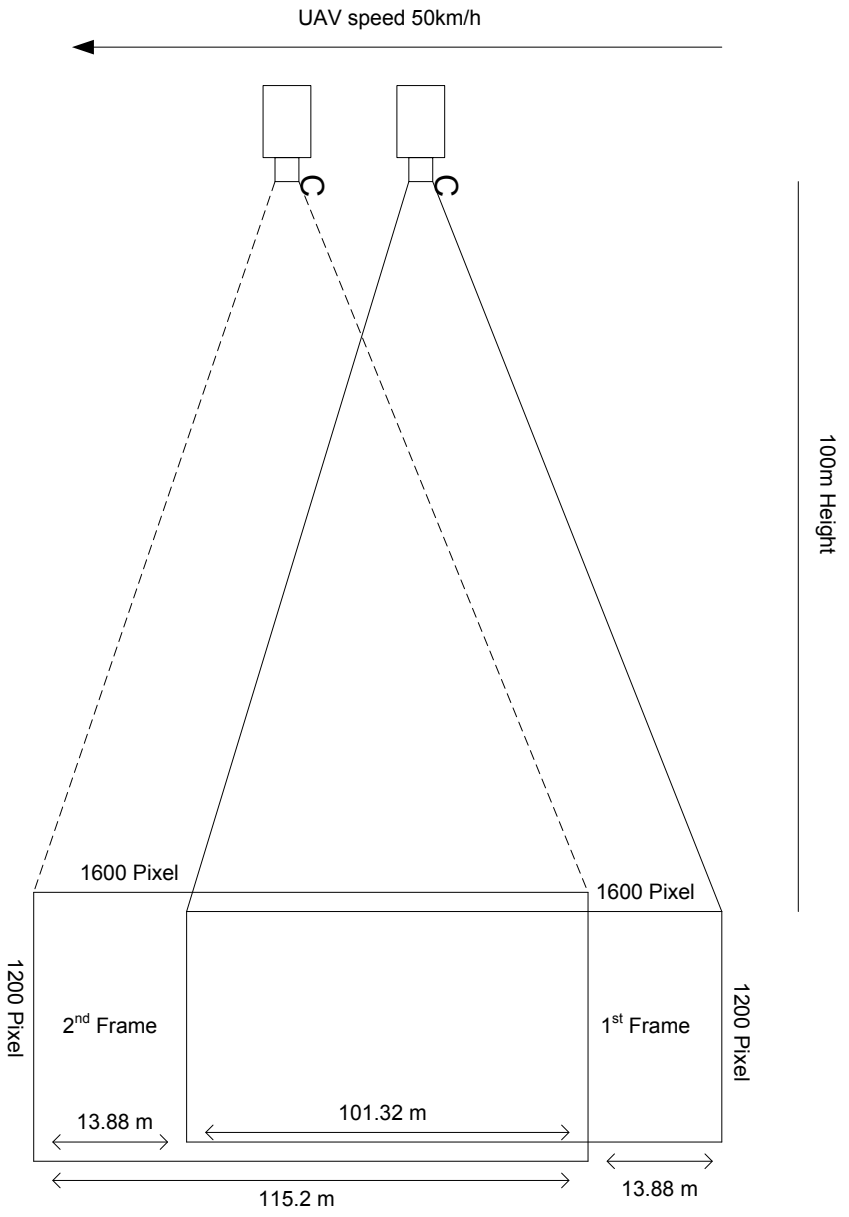


Figure 1.11: Time interval for a new scene

1.2.2 Data Link Requirements

A standard plug-and-play 2.4 GHz Wireless Fidelity (WiFi) module is required for this project in order to simulate the RF (Radio Frequency) part in the UAV. This will determine the bit rate at which the video would be transmitted over a wireless link.

1.3 Conclusion

A background of the main components of the project and its requirements has been presented in this chapter. The requirements of the project can be summarised as follows:

1. CMOS camera with a 2 MegaPixel frame size at the quickest fps available.
2. An encoder that can encode a 2 MegaPixel frame size at the quickest fps possible.
3. A standard plug-and-play 2.4 GHz WiFi module.

The following chapters of the thesis are structured as follows: Chapter 2 looks at the different solutions of the problem studied herein. The design and implementation phase of the project are presented in Chapter 3, as well as the schematics and the layout of the Printed Circuit Board (PCB). Chapter 4 explains the design of the software (SW) and hardware (HW) debugging. The results are presented in Chapter 5. Finally, the conclusions and possible improvements are summarised in Chapter 6.

2. Chapter 2: Encoder Platform Selections

2.1 Introduction

Video compression encoders could be implemented by means of Digital Signal Processors (DSPs) or Field Programmable Gate Arrays (FPGAs). FPGAs can outperform DSPs in many ways because they are reconfigurable HW which allows the implementation of multiple functions in parallel. However, FPGAs are more complex to develop for and they are usually more expensive than DSPs. Conversely, in some applications, DSPs may perform slower than FPGAs because they deal with code and instructions, which have to be fetched and then executed. However, DSPs are easy to develop for and are considered less complex, therefore, if it is possible to meet the system requirements using a DSP rather than an FPGA, a DSP solution is usually preferred.

Initially, the idea was to implement the video encoder in FPGA because this would achieve a faster compression rate than would be possible with DSP. A high compression rate is dependent on the success of ME and MC, and it has been found that processors spent most of the time executing ME and MC (see Section 1.1.1.2). Therefore, to save time, it was decided to implement only ME and MC and then to integrate these with a JPEG encoder. ME and MC were thus implemented for a 16x16 pixel MBs, but the development stopped due to the high cost of a JPEG encoder. The code was designed using Actel Libero IDE v8.0 (see Appendix E.2). Video compression encoders are very complex to implement in FPGA and there was not enough information to write the encoder in VHDL (VHSIC [Very High Speed Integrated Circuits] Hardware Description Language). Thus, due to the complexity of implementing an encoder and time pressure, it was decided to look for an encoder that would be able to encode up to 2 MegaPixel at the quickest fps rate available.

2.2 Comparison of Options

A few encoders written in VHDL were found. Conversely, C code encoders are widely available, but they are limited in the resolution that they can encode.

2.2.1 Field Programmable Gate Array (FPGA)

FPGA encoders are available in the market but are very expensive. For example, the Xilinx and CAST encoder intellectual property cores will cost US$ 20,000 and US$ 48,000 respectively, whereas Ocean Logic will charge a five- or six-digit amount in Euros.

2.2.2 Digital Signal Processing (DSP)

Many companies provide encoders that can run on their DSPs, such as Texas Instruments, Freescale and Analog Devices. According to my search, up to January 2009 Texas Instruments and Freescale had encoders that were able to encode up to D1 (720X480) resolution only. Also, it was found that Analog Devices was the only company that could provide encoders with up to 5 MegaPixel frame sizes. As all Analog Devices encoders are available for free evaluation, one such encoder was chosen for this project.

2.2.3 Analog Devices

Analog Devices offers four video encoders: H.264 Baseline Profile (BP), H.264 BP/ Main Profile (MP), Moving Picture Experts Group (MPEG-2) and MPEG-4 Visual.

H.264 BP/MP is the newer version of H.264 BP, whereas MPEG-4 Visual is the newer version of MPEG-2. All of these encoders have been demonstrated on ADSP-BF533, ADSP-BF561, ADSP-BF527 and ADSP-BF548 Blackfin DSPs. Therefore, a comparison has to be made between MPEG-4 visual and H.264, also known as Advanced Video Coding (AVC), in order to choose the most suitable encoder. Furthermore, a comparison between the Blackfin DSPs has to be made, in order to choose the most suitable DSP.

2.2.4 MPEG-4 Visual and H.264 (AVC)

MPEG-4 Visual (Part 2 of the MPEG-4) was standardized in 1999 by MPEG which is a working group of the International Organization for Standardization (ISO). MPEG-4 Visual is more flexible than H.264, as it provides a wide range of techniques to accommodate different types of applications that require high quality video such as Television (TV), movies, 3 Dimension (3D) and other applications [5].

Conversely, H.264 concentrates on efficiency of the compression and transmission with features to support reliable and robust transmission over networks. Also, one of the H.264 applications is streaming of live videos. H.264 (MPEG-4 Part 10) was standardized in 2003 by the Video Coding Experts Group (VCEG) which is a working group of the International Telecommunication Union-Telecommunication (ITU-T) and MPEG [5].

It was decided that H.264 would be more suitable for this project than MPEG-4 Visual because one of the features of H.264 is its transmission efficiency. Furthermore, H.264 provides reliable and robust transmission over networks.

2.2.5 Overview of Analog Devices' H.264 BP/MP Encoder

The H.264 BP/MP encoder is a library, which encodes video to the H.264 video bit stream standard. It can take input data from a real-time video capturing device, such as a camera. The encoder processes a single frame at a time and produces an elementary bitstream for that frame. The input video format can be the YUV422 format progressive raw from the CMOS sensors, which is what this project requires. It supports I-, P- & B-frames and can encode up to a frame size of 5 MegaPixels. Furthermore, the frame rate can be from 2 to 30 fps, and it has the flexibility to choose between Variable Bit Rate (VBR) and Constant Bit Rate (CBR) control, (see Figure 2.1) [12].

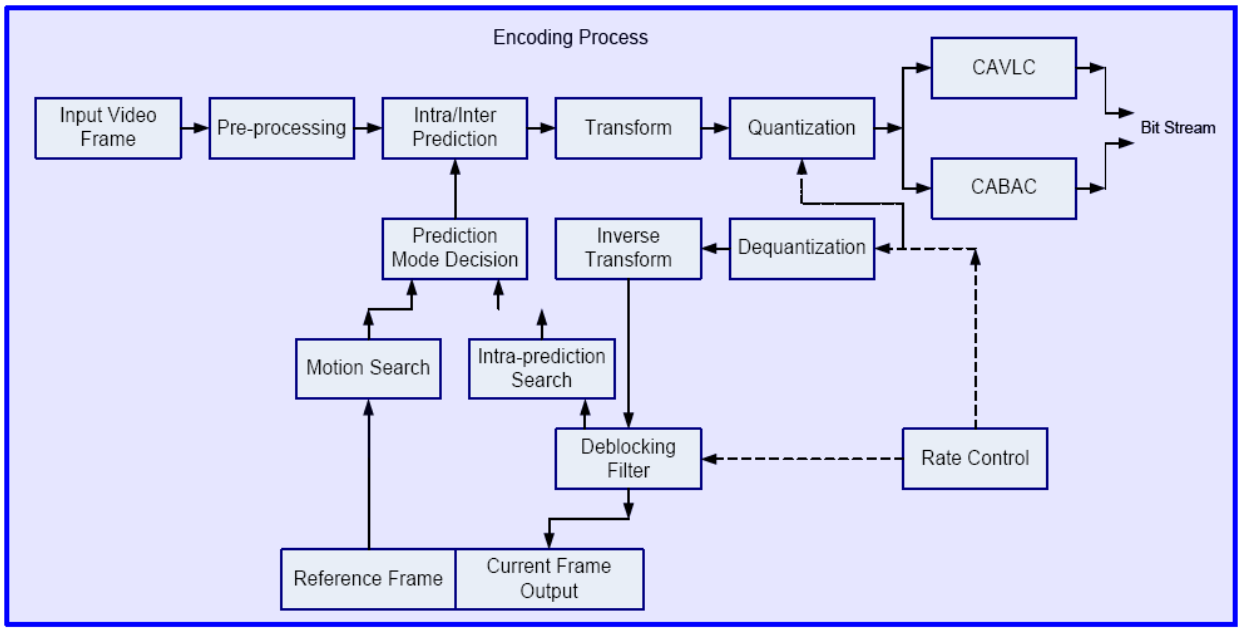


Figure 2.1: From [12]. H.264 BP/MP Encoder. CABAC is the Context Adaptive Binary Arithmetic Coding and CAVLC is the Context Adaptive Variable Length Coding.

2.2.6 Blackfin DSPs

ADSP-BF533 (BF533), ADSP-BF527 and ADSP-BF548 (BF548) have almost the same features; all of them provide up to 600 MHz processing speed. In terms of memory and peripherals, BF548 has the biggest on-chip memory of up to 324 Kbytes and it is the most peripheral-rich amongst the abovementioned DSPs [13], [14] and [15].

ADSP-BF561 (BF561) is a dual Blackfin core DSP, with each core able to operate at up to 600 MHz. BF561 has two independent Blackfin cores, which will result in less processing time than a normal Blackfin DSP. Also, BF561 has up to 328 Kbytes on-chip memory [16].

Although BF561 is the best processor among them, it is only available in the Ball Grid Array (BGA) package as well as together with the other processors, except ADSP-BF533. BF-533, however, is available in the BGA and Low-profile Quad Flat Package (LQFP) packages. LQFP is simpler than BGA and can be visually inspected so it is easier to debug. Furthermore, in most of the BGAs, bringing out the pins is not easy. Also, LQFP is easy to populate, whereas BGA requires very complex process to be populated. However, the BGA package is better from an electrical point of view and it is usually smaller than LQFPs.

As the board constructed for the purposes of this study is a development board, it was decided that the LQFP package would be more appropriate. Also, the university does not have the facilities that guarantee the successful population of BGA packages. Therefore, the ADSP-BF533 was chosen as the DSP for this project.

2.2.7 Blackfin ADSP-BF533 Overview

The 176-LQFP package of BF533 was chosen for this project. It provides a core clock of up to 400 MHz (CCLK), a system clock of 133 MHz (SCLK) and up to 148 Kbytes of on-chip memory. The BF533 has an external memory controller, which provides a glueless connection to a bank of Synchronous Dynamic Random Access Memory (SDRAM), as well as up to four banks of asynchronous memory devices. The system peripherals include a Universal Asynchronous Receiver Transmitter (UART) port, a Serial Peripheral Interface (SPI) port, two Serial Ports (SPORTs), 16 General-Purpose Input/Output (I/O) pins (GPIO), a real-time clock, a watchdog timer, and a Parallel Peripheral Interface (PPI). Furthermore, the BF533 has four memory-to-memory Direct Memory Access (DMAs) and eight peripheral DMAs, and it provides the options of booting from either SPI or external memory (see Figure 2.2) [13].

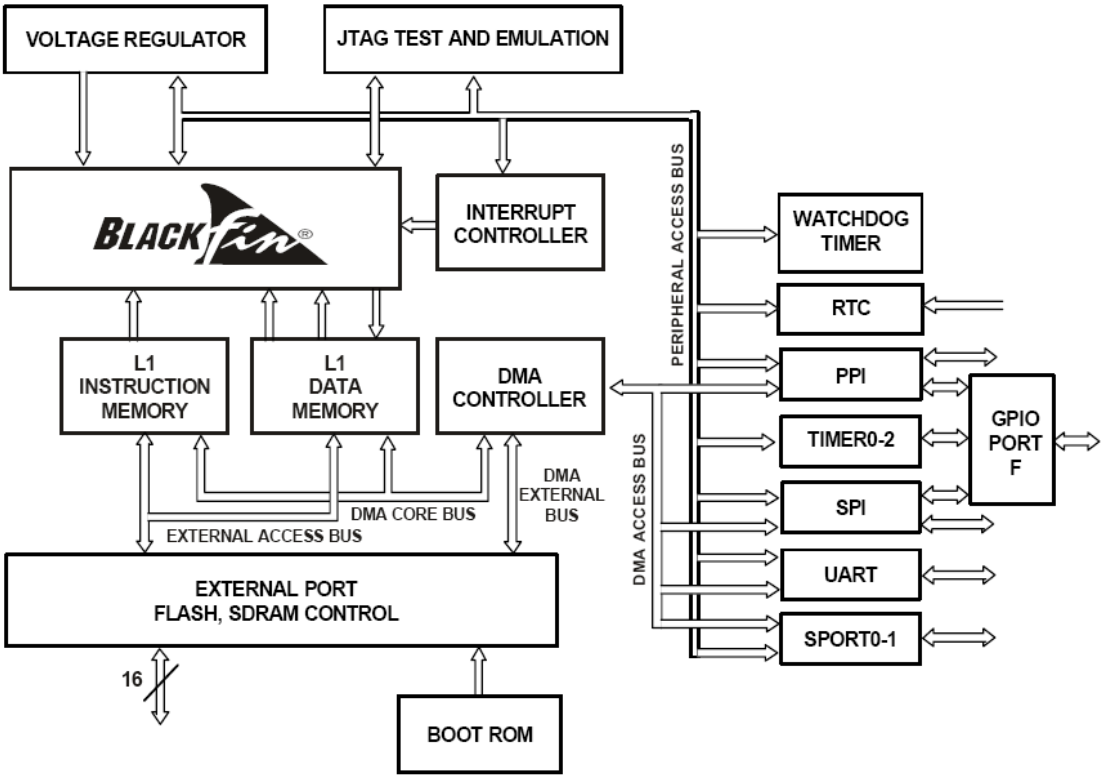


Figure 2.2: From [13]. BF533 architecture

2.3 Conclusion

Options have been introduced in this chapter and it was explained why Analog Devices' H.264 (MPEG-4 Part 10) encoder was chosen together with the ADSP-BF533 Blackfin DSP. In addition to the reasons discussed above, the decision was influenced by the fact that Analog Devices provides evaluation kits, application notes and software examples, which can be very useful as a starting point.

3. Chapter 3: Design and Implementation

3.1 High Level Description

The Analog Devices' evaluation kit ADSP-BF533 EZ-KIT Lite was used as a basis for the design, but it was extensively modified to accommodate the requirements of this project [17]. This evaluation kit was chosen because it had demonstrated the use of the H.264 encoder on the Analog Devices BF533 DSP.

The camera captures frames at a specified rate that is set via the DSP, before it then passes the data to the DSP through a PPI. The DSP encodes the incoming frames by using H.264 encoder software. These encoded frames are passed to the WiFi module through a SPI and finally the WiFi module streams the encoded frames to the host destination. This process has to happen in real time. Furthermore, ADM3202 was replaced with a USB-UART module because most of the new laptops do not have a UART port. Figure 3.1 illustrates the new design. Each block in Figure 3.1 is described in detail in this Chapter.

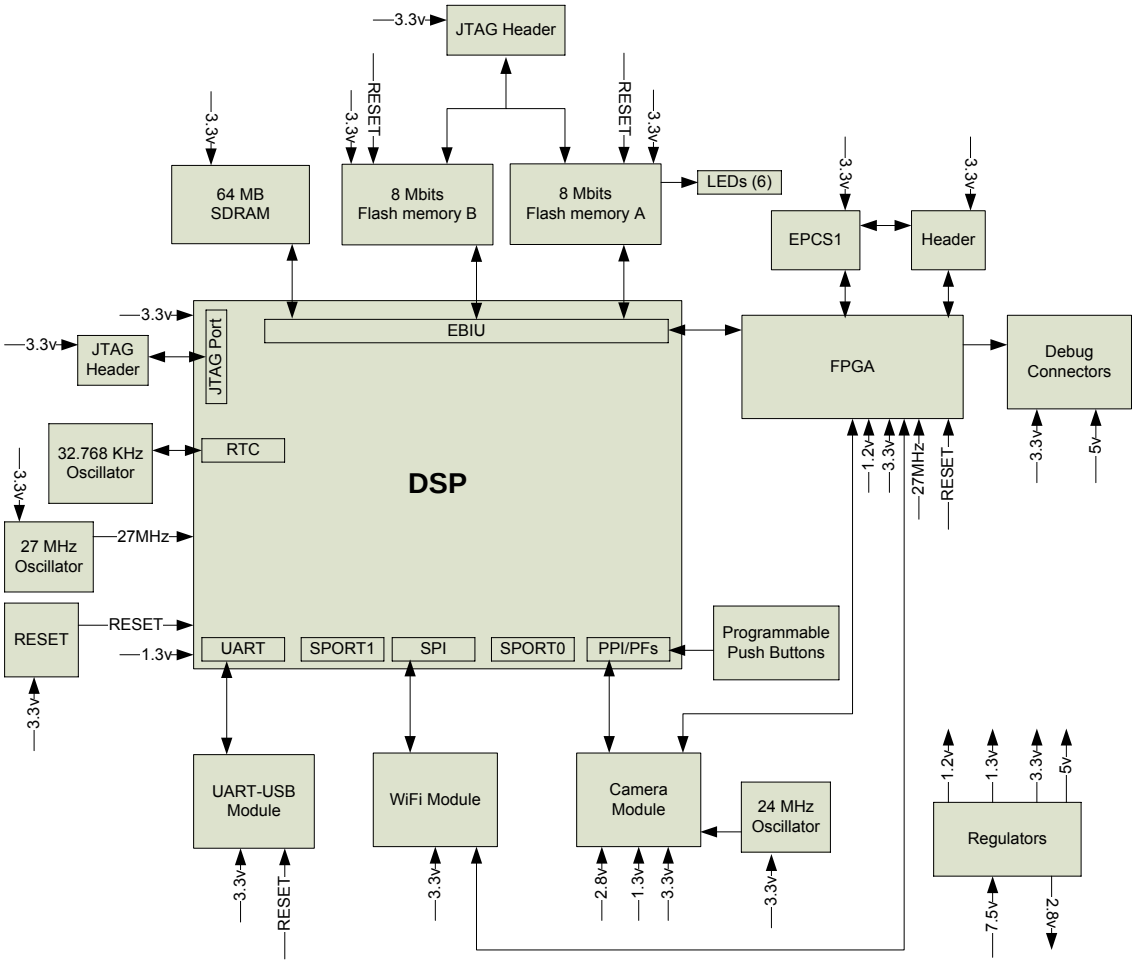


Figure 3.1 : System design

3.1.1 Rules Followed During Schematic Design

First, any input pins that were not being used were pulled low; this was done to protect the chips on the board. Secondly, pads were placed for any unused output pins which may still be useful in the future. Thirdly, one polarized capacitor of 100 uF was put onto each chip, and one normal capacitor of 0.1 uF was used for each Vcc pin for decoupling. Fourthly, 0 Ω resistors were placed between the output of the regulators and the rest of the board. The reason for this is that the power on the board could be

tested by supplying one voltage at a time. Finally, test points for the critical signals were placed for examination.

3.1.2 DSP Block (Blackfin ADSP-BF533)

The DSP block is the core of the project, and in it all system processing and control functions of the system are implemented. The Blackfin ADSP-BF533 is the selected DSP for this project (see Section 2.2.6), and the H.264 encoder was implemented to run on this block.

The BF533 can be programmed by using the Joint Test Action Group (JTAG) port through the $\overline{EMU}$, TMS, TCK, TDI, TDO and $\overline{TRST}$ signals. The JTAG circuit was implemented by following the method presented in the EE-68 application note (see Figure 3.2) [18].

The core voltage is 1.3v, whereas the I/O voltage of the BF533 is 3.3v. Also, the BF533 has four modes for booting by means of BMODE0 and BMODE1 pins (see Table 3.1). Therefore, a two-pin switch was connected to the BF533 to enable the user to choose the desirable boot mode (see Figure 3.2) [13] and [17].

Table 3.1 : From [13]. BMODES

BMODE1 – 0	Description
00	Execute from 16-bit external memory (bypass boot ROM)
01	Boot from 8-bit or 16-bit FLASH
10	Boot from serial master connected to SPI
11	Boot from serial slave EEPROM /flash (8-,16-, or 24-bit address range, or Atmel AT45DB041, AT45DB081, or AT45DB161serial flash)

A buck converter circuit was implemented for the BF533 to complete the power management system using the BF533 data sheet [13] and the EE-228 application note [19] (see Figure 3.2).

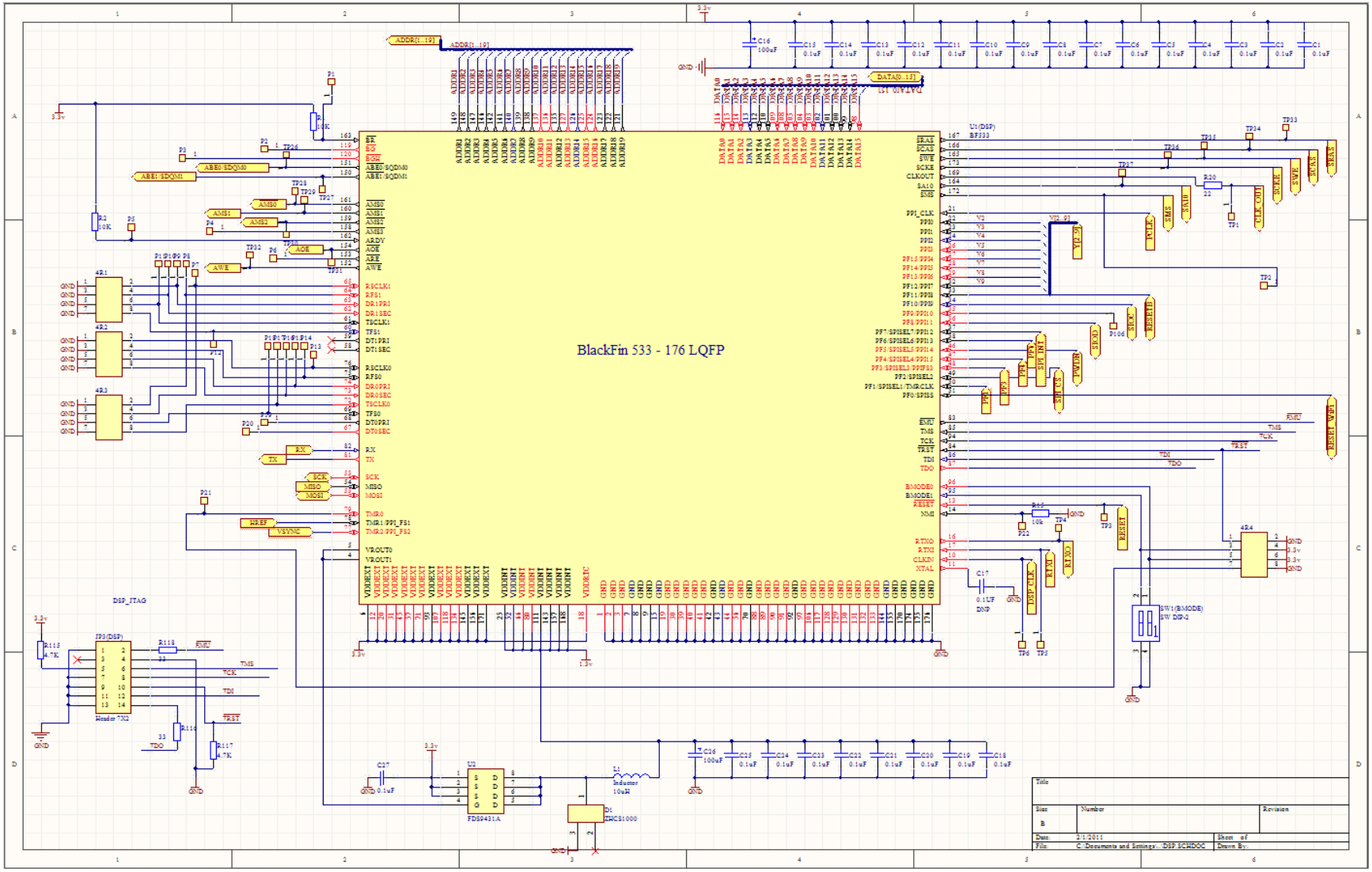


Figure 3.2 : DSP schematic sheet

3.1.3 Flash Memories and SDRAM

The BF533 boots from the flash memory block. Analog Devices have demonstrated the use of the H.264 encoder by using two PSD4256G6V (PSD) chips from STMicroelectronics and MT48LC32M16A2TG from Micron. Therefore, these same memory chips were used in this project. Furthermore, the board is intended to be used for development, where the PSD4256G6V can also be used as a standard flash memory or Static Random Access Memory (SRAM) as peripherals for the BF533 [20]. The MT48LC32M16A2TG SDRAM provides 64 Mbytes of storage for the BF533. Consequently, for the UXGA frame size with 2 bytes for each pixel, the MT48LC32M16A2TG can store up to 16 frames [22]. Moreover, Analog Devices also provides the drivers for these chips, which saved some time while writing the SW.

The PSDs (Flash A and Flash B) consists of 8 Mbits flash memory, 256 Kbits SRAM and 3000 gates of Programmable Logic Device (PLD) including Complex PLD (CPLD) and Decode PLD (DPLD) [20]. The DPLD can be programmed to manage the memory banks instead of the BF533, and the CPLD can be programmed to manage ports A, B, C and D of the flash. There are six programmable Light Emitting Diodes (LED) on the board connected to port B in Flash A, which can be programmed by CPLD. Furthermore, the PSD is a full chip with In System Programmability (ISP), and with a built-in JTAG serial port, which allows easy testing and programming by means of a low-cost FlashLINK cable [20]. Port E of the PSDs is the JTAG port; six signals are used for the JTAG connection TMS, TCK, TDI, TDO, TSTAT and $\overline{TERR}$. The JTAG circuit was implemented by following the steps presented in the AN1153 application note [21]. Also, a NAND and an inverter circuit were implemented to be able to reset the Flash memories from either JTAG or a reset push button (see Figure 3.3).

The PSDs and MT48LC32M16A2TG (SDRAM) were connected to the BF533 by means of an External Bus Interface Unit (EBIU). The EBIU is a 16-bit interface that provides a glueless connection [13].

The following signals are connected to interface BF533 with the PSDs (see Figure 3.3) [13], [17] and [20]:

Input Signals:

- $\overline{\text{ABE0}}/\text{SDQM0}$ and $\overline{\text{ABE1}}/\text{SDQM1}$: Byte enabled and data masks for Async/Sync access.
- $\overline{\text{AMS2} - 0}$: Memory bank select.
- $\overline{\text{AWE}}$: Write enable.
- $\overline{\text{AOE}}$: Read enable
- ADDR[1..19]: Address bus.

Bidirectional Signals:

- DATA[0..15]: Data bus.

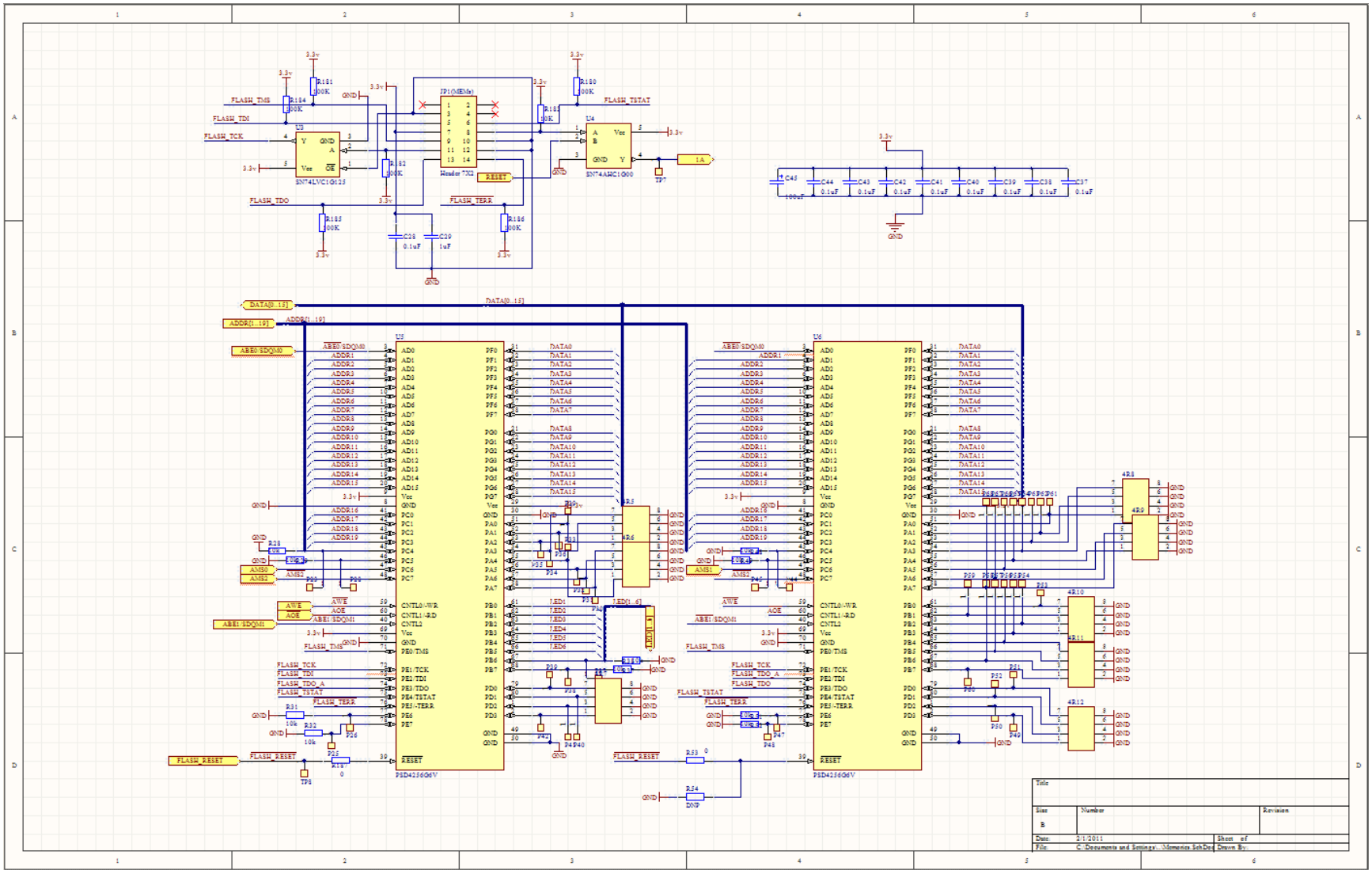


Figure 3.3 : PSDs schematic sheet

Each PSD has 8 Mbit of primary flash, 512 Kbit of secondary flash and 256 Kbit of SRAM (see Table 3.2) [20].

Table 3.2 : From [20]. PSD memory block size and organization

	Primary Flash Memory		Secondary Flash Memory		SRAM	
Sector Number	Sector Size (Bytes)	Sector Select Signal	Sector Size (Bytes)	Sector Select Signal	SRAM Size (Bytes)	SRAM Select Signal
0	64K	FS0	16K	CSBOOT0	32K	RS0
1	64K	FS1	8K	CSBOOT1		
2	64K	FS2	8K	CSBOOT2		
3	64K	FS3	32K	CSBOOT3		
4	64K	FS4				
5	64K	FS5				
6	64K	FS6				
7	64K	FS7				
8	64K	FS8				
9	64K	FS9				
10	64K	FS10				
11	64K	FS11				
12	64K	FS12				
13	64K	FS13				
14	64K	FS14				
15	64K	FS15				
Total	1024K	16 Sectors	64K	4 Sectors	32K	

The BF533 has four reserved asynchronous memory banks of 8 Mbit for each bank (see Figure 3.4) [13]. Therefore, $\overline{\mathrm{AMS0}}$ and $\overline{\mathrm{AMS1}}$ were assigned for the primary flash of Flash A and Flash B respectively. $\overline{\mathrm{AMS2}}$ was assigned for the secondary flash, and SRAM in the both PSDs (Flash A and Flash B) (see Table 3.3).

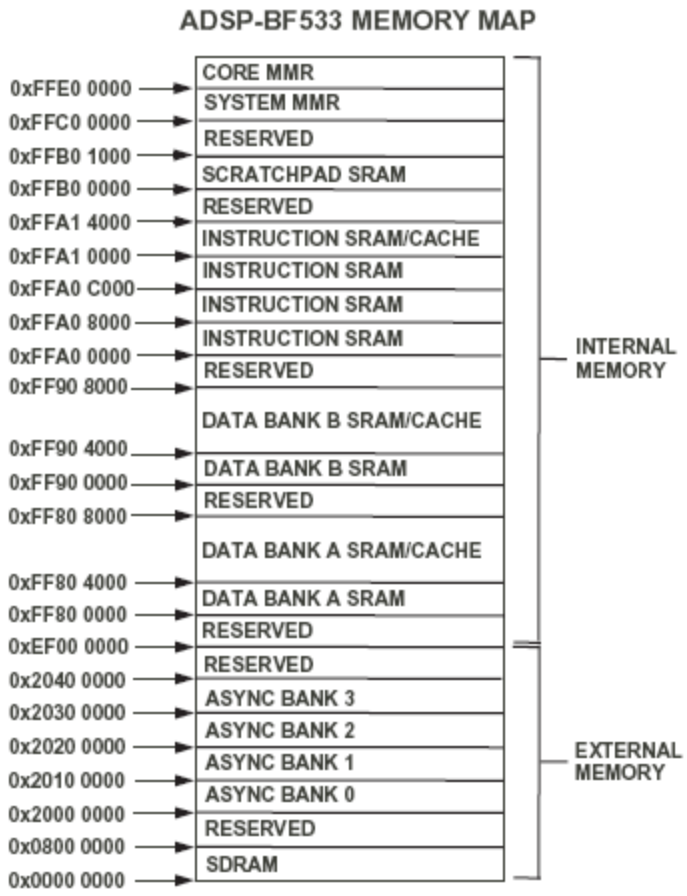


Figure 3.4 : From [32]. BF533 memory map

Table 3.3 : DSP flash memory map

Address	***Assigned to***
$0x2000\ 0000$ to $0x200F\ FFFF^{(1)}(\overline{AMS0})$	Primary flash of Flash A (1Mbyte)
$0x2010\ 0000$ to $0x201F\ FFFF^{(1)}(\overline{AMS1})$	Primary flash of Flash B (1 Mbyte)
$0x2020\ 0000$ to $0x2020\ FFFF^{(1)}(\overline{AMS2})$	Secondary flash of Flash A (64 Kbyte)
$0x2024\ 0000$ to $0x2024\ 7FFF^{(1)}(\overline{AMS2})$	SRAM of Flash A (32 Kbyte)
$0x2027\ 0000$ to $0x2027\ 00FF^{(1)}(\overline{AMS2})$	I/O of Flash A (256 Byte)
$0x2028\ 0000$ to $0x2028\ FFFF^{(1)}(\overline{AMS2})$	Secondary flash of Flash B (64 Kbyte)

$0x202C\ 0000$ to $0x202C\ 7FFF^{(1)}(\overline{AMS2})$	SRAM of Flash B (32 Kbyte)
$0x202E\ 0000$ to $0x202E\ 00FF^{(1)}(\overline{AMS2})$	I/O of Flash B (256 Byte)

(1): These numbers are the same numbers in Flash_A.abl and flash_b.abl files in the PSD4256G_ConfigFiles folder from Analog Devices. The numbers have not been modified because the PSDs were configured by loading flash_a.obj and flash_b.obj files from the PSD4256G_ConfigFiles folder from Analog Devices, and therefore these numbers are compatible with the config files.

The SDRAM block is the main memory of the system. The following signals are connected to interface the BF533 with the MT48LC32M16A2TG (see Figure 3.5), [13], [17] and [22]:

Input Signals:

- $\overline{ABE0}/SDQM0$ and $\overline{ABE1}/SDQM1$: Byte enabled and data mask for Async/Sync access.
- SA10: Connected to address 10 pin.
- $\overline{SRAS}$: Row address strobe.
- $\overline{SCAS}$: Column address strobe.
- $\overline{SWE}$: Write enable.
- $\overline{SMS}$: Bank select.
- SCKE: Clock enable.
- CLKOUT: Connected to CLK pin.
- ADDR[1..19]: Address bus.

Bidirectional Signals:

- DATA[0..15]: Data bus.

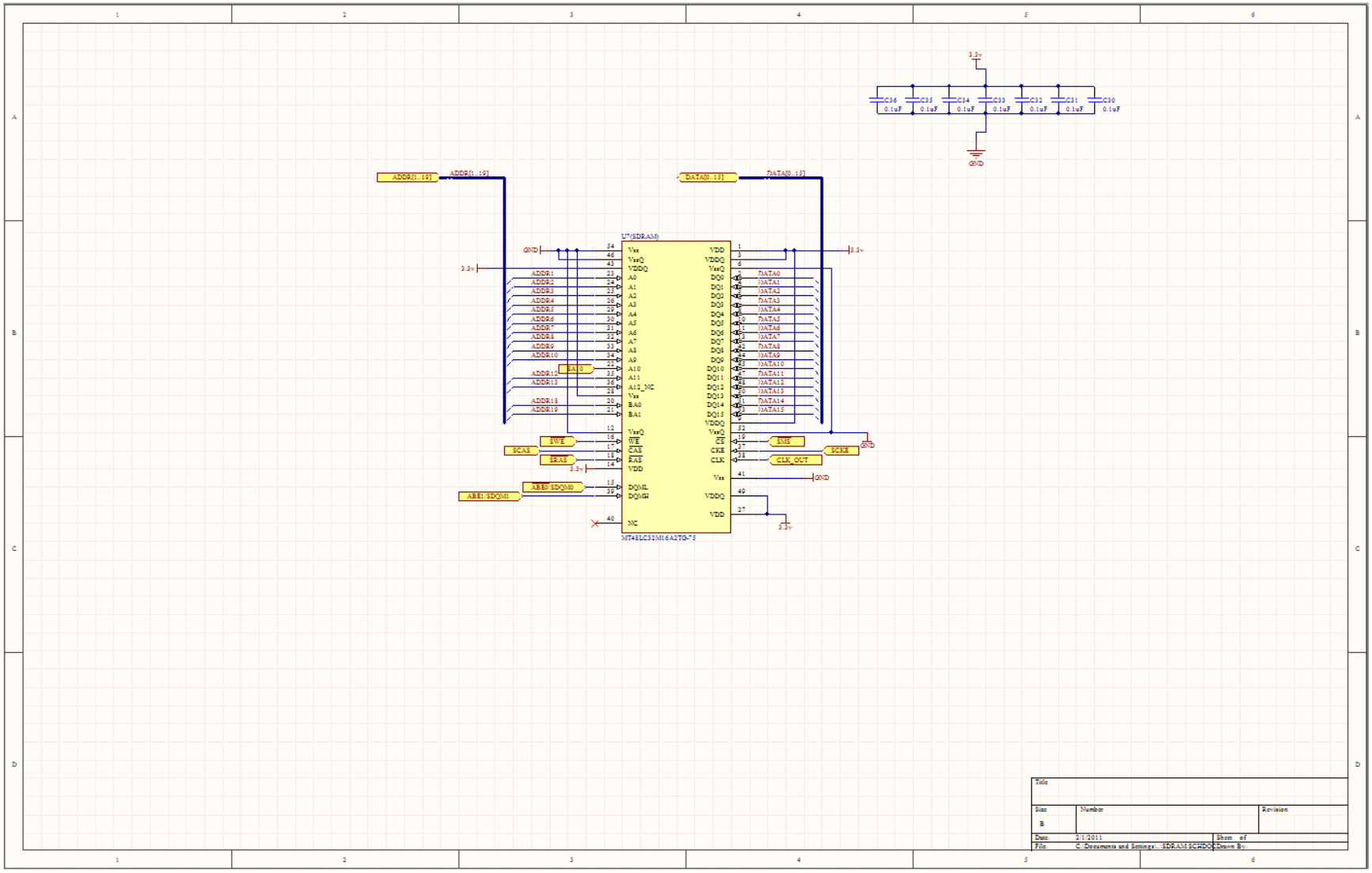


Figure 3.5 : SDRAM schematic sheet

3.1.4 Camera Module Block

The camera module block is responsible for taking pictures and then sending them to the BF533. The OV2640 camera module was chosen for this particular project because it uses CMOS technology and because it provides up to UXGA frame size at 15 frames per second (fps). All of these meet the requirements of having a 2 MegaPixel CMOS camera (see Section 1.2.1). In addition, the OV2640 camera module can give an output format of YUV (422/420)/YCbCr422, which is accepted by the H.264 encoder (see Section 2.2.5). Lastly, the OV2640 module is available from OmniVision for a reasonable price. OV2640 might not be the best quality camera compared to Kodak or Micron cameras, but it was available and cost a reasonable price. Kodak cameras were expensive, whereas Micron cameras were not available.

The OV2640 module is connected to the BF533 through the PPI because PPI can connect directly to the video encoders and video source. The PPI has up to three frame synchronization pins, up to 16 data pins and an input clock pin [13].

All required OV2640 functions are programmable by means of the Serial Camera Control Bus (SCCB) [23]. By looking at the SCCB Functional Specification document from OmniVision, one can see that SCCB is the I^2C protocol [24]. YUV (422/420)/YCbCr422 is an 8 bit output from OV2640, therefore only 8 data lines are connected to the BF533.

The following signals are connected to interface BF533 with the OV2640 [23], [25] and [13] (see Figure 3.6):

Input Signals:

- SIOC: SCCB serial interface clock.
- $\overline{\text{RESETB}}$: Reset.
- PWDN: Power down mode enable.

Bidirectional Signals:

- SIOD: SCCB serial interface data.

Output Signals:

- VSYNC: Vertical synchronization.
- HREF: Horizontal reference.
- PCLK: Pixel clock.
- Y[2..9]: Data output.

As the OV2640 module runs at 24 MHz, a 24 MHz crystal chip was thus used for the implementation. The OV2640 circuit was implemented following the OmniVision serial camera control bus functional specification [23], the camera data sheet [25] and the BF533 data sheet [13].

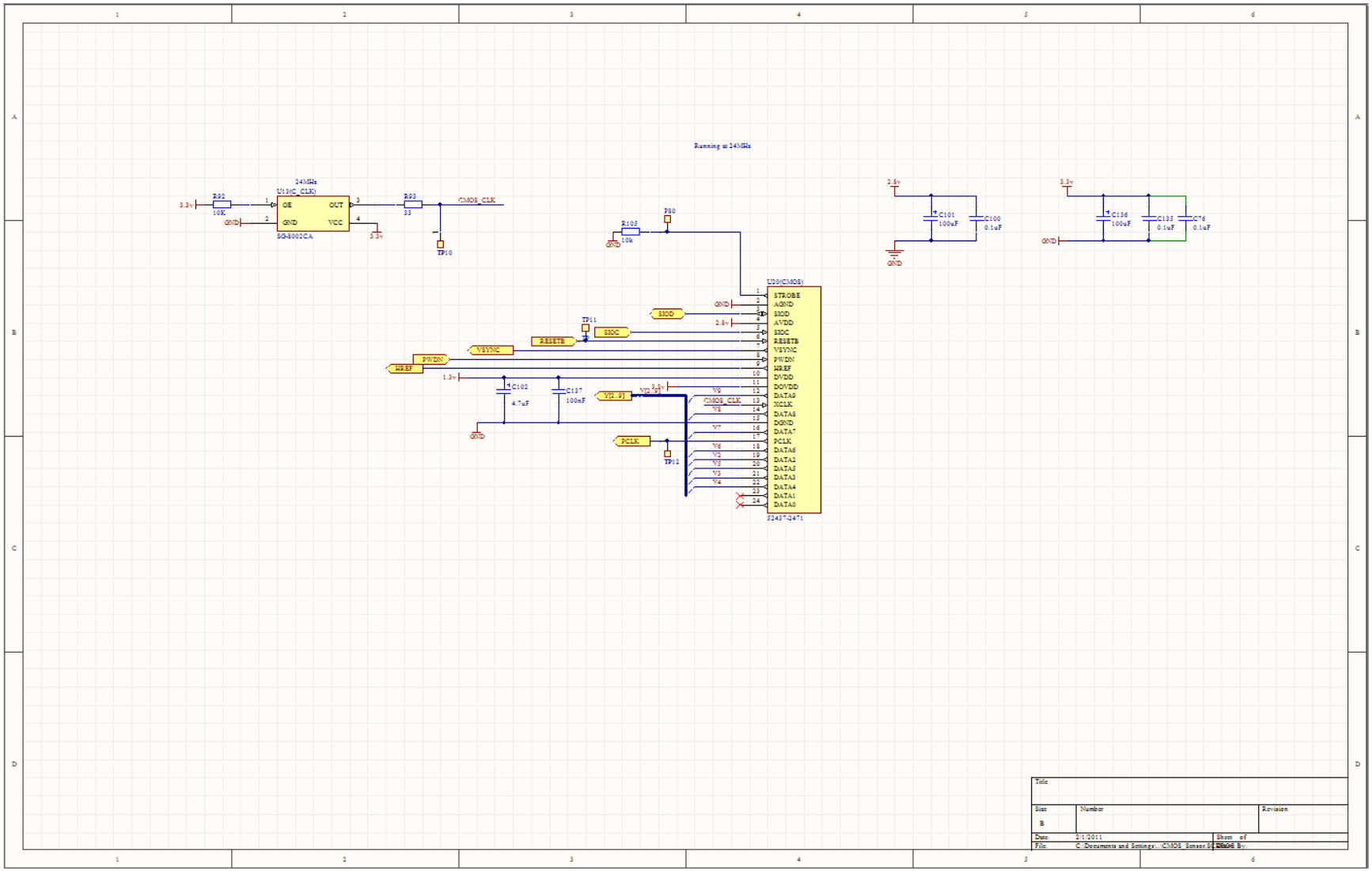


Figure 3.6 : Camera module schematic sheet

3.1.5 WiFi Module Block

The WiFi module block streams the data to the network. iW- SM2144N1-EU-0 (Nano WiReach) from ConnectOne was chosen for this project. Nano WiReach is a WiFi module that connects serial devices to 802.11b/g Wireless Local Area Networks (LANs). Moreover, it supports up to 10 simultaneous TCP (Transmission Control Protocol)/UDP (User Datagram Protocol) sockets, which can be used for streaming out the video. NanoWiReach can be programmed by sending commands that are specified in the AT+i programmer's manual document from ConnectOne which eliminates the need to write complex drivers. Nano WiReach is considered a plug-and-play module, which was particularly important for this project, given the time constraints [26] and [27].

The Nano WiReach module offers an UART interface and an SPI interface. As the UART interface was already being used, the Nano WiReach was connected to the BF533 by means of the SPI. Six signals were connected between the module and the BF533. BF533 was the master whereas the module was the slave, because the BF533 sent the encoded frames to the module, which then streams them out to the host destination.

The following signals are connected to interface BF533 with the Nano WiReach (see Figure 3.7):

Input Signals:

- SCK: SPI CLK.
- MOSI: Master out slave in.
- $\overline{\text{SPI\_CS}}$: Chip select.
- $\overline{\text{RESET\_WiFi}}$: Reset the module.

Output Signals:

- SPI_INT: Interrupt to inform the BF533 that the module has data on its buffer.
- MISO: Master in slave out.

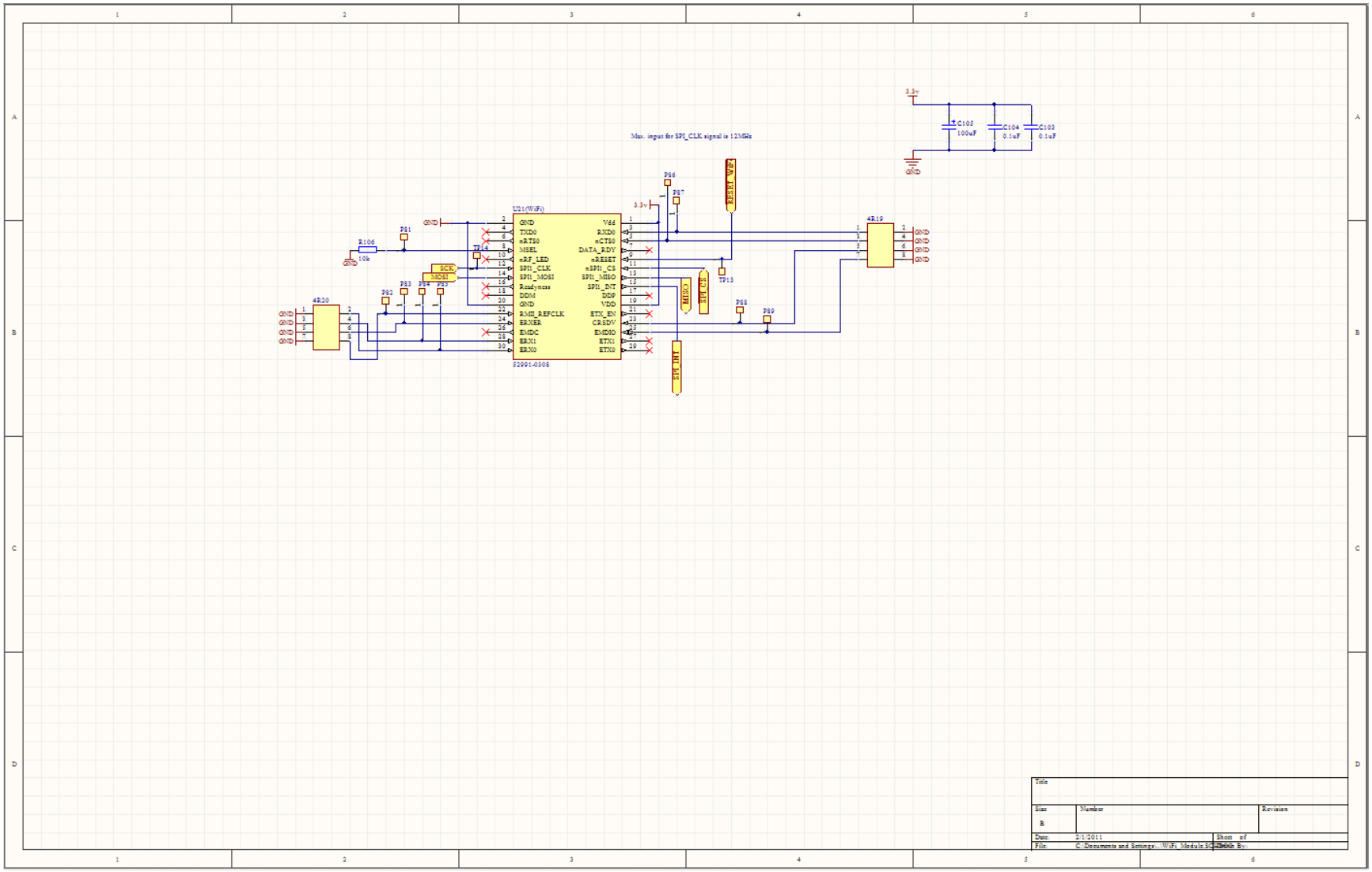


Figure 3.7 : WiFi schematic sheet

3.1.6 UART-USB Module Block

The UART-USB module block connects the BF533 UART interface to a USB interface. The UM232R was chosen for this purpose; it needs a single supply voltage range from 3.3 v to 5.25 v and the clock circuit is integrated onto the device. Moreover, it has both transmit and receive LED drive signals. The UM232R is also a fully integrated module, which means that no external components are required [28].

The following signals are connected to interface BF533 with the UM232R:

Input Signals:

- TX: UART transmit from BF533.

Output Signals:

- RX: UART receive to BF533.

Two LEDs are connected to the module, one for transmitting and one for receiving. 390 Ω resistors are connected in series with the LEDs to protect them and to give an acceptable brightness (see Figure 3.8). Voltage drops at TX LED and RX LED are 2.2 v and 2.1 v respectively, and the maximum current is 30 mA [29]:

$$V = I \times R \Rightarrow I = \frac{Vcc - Voltage\ Drop(TX\ LED)}{R} \Rightarrow I = \frac{3.3 - 2.2}{390} = 2.8\ mA$$

$$V = I \times R \Rightarrow I = \frac{Vcc - Voltage\ Drop(RX\ LED)}{R} \Rightarrow I = \frac{3.3 - 2.1}{390} = 3\ mA$$

Since 2.8 mA and 3 mA are less than 30 mA, they will not damage the LEDs and will give an acceptable brightness.

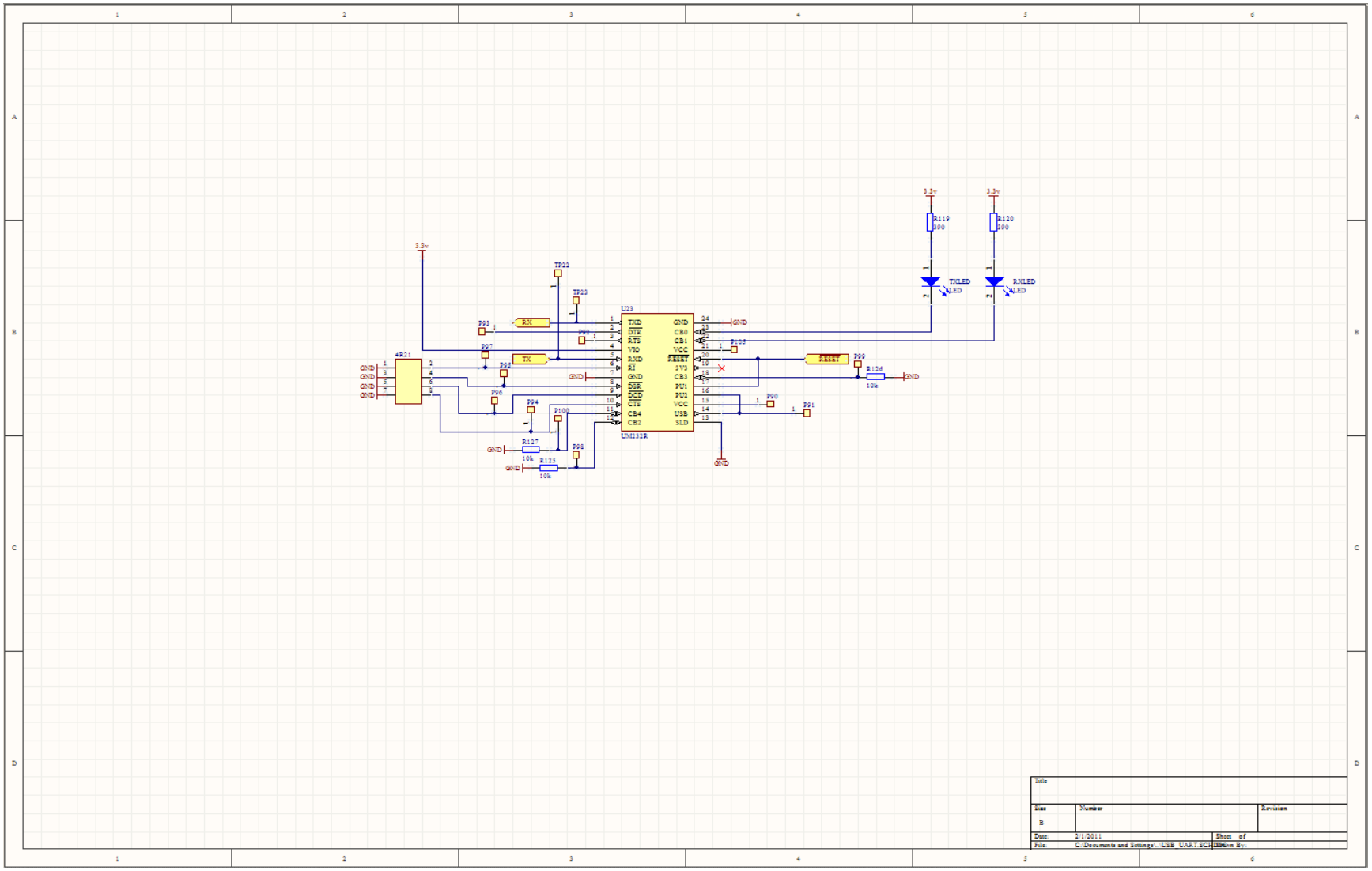


Figure 3.8 : UART-USB module schematic sheet

3.1.7 FPGA and Debug Connectors

The FPGA was not intended to be part of the scope of this project. The FPGA was used to provide a simple method of debugging the board and to make provision for future use. Consequently, all the address and data lines, camera interface and WiFi interface were connected to the FPGA before being routed to the debug connectors through the IDT74FCT3244 and IDT74FCT62244 buffers to protect the FPGA. The FPGA clock can be either 27 MHz or the DSP clock out. Four memory control signals $\overline{\mathrm{AWE}}$, $\overline{\mathrm{AOE}}$, $\overline{\mathrm{AMS1}}$ and $\overline{\mathrm{AMS0}}$ were connected to be able to use the FPGA as a memory (see Figure 3.9). Cyclone 2 EP2C5Q208C7N from Altera was chosen because it was available and because it can do the required tasks of the project.

For future use, the FPGA can be used for modifying the camera data signals before sending them to the DSP, for managing the WiFi module, or for acting as a memory.

The serial configuration device EPCS1 was connected to the FPGA and a connector to provide in-system programming [30] and [31]. The FPGA boots from the EPCS1.

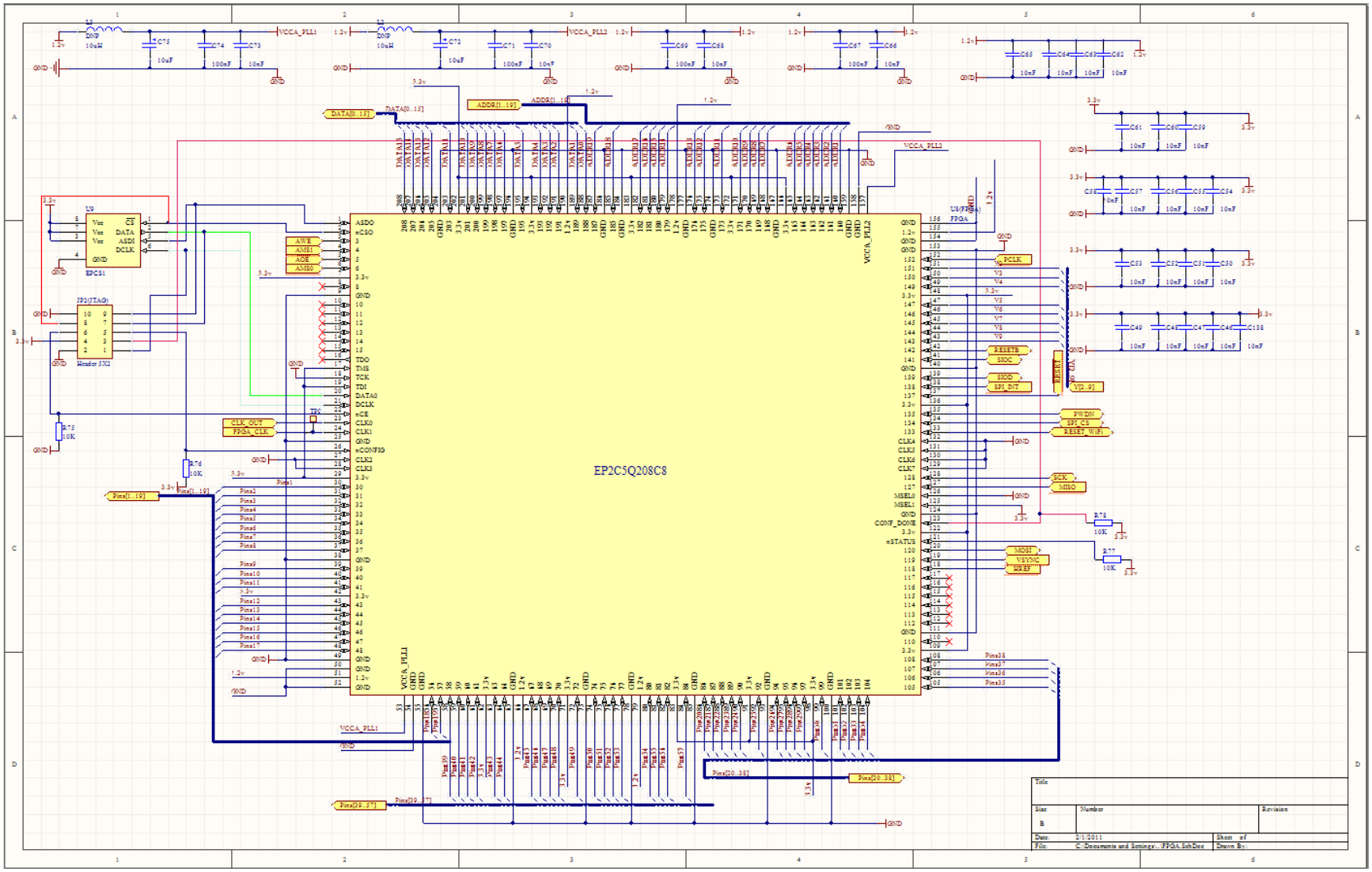


Figure 3.10 : FPGA schematic sheet

3.1.8 Clock System

The system runs on 27 MHz and 24 MHz. The DSP runs on 27 MHz. A 1-to-10 clock driver was added to the circuit to ensure that the components receive the clock signal at the same time. The camera module runs on 24 MHz (see Figure 3.10).

A 27 MHz clock was used as a safe starting point because the BF533 evaluation kit used 27 MHz [17]. It was found that 24 MHz might be better, though, because this means that the CCLK and SCLK of the BF533 can increase up to 384 MHz and 128 MHz respectively, whereas a 27 MHz can increase them up to 378 MHz and 126 MHz respectively (see Section 2.2.7) [32]. In addition, the system would use only one crystal rather than two. Calculations in respect of the 24 MHz are presented in Chapter 6.

A 32.768 KHz real time clock crystal was implemented in the same way as was specified in the BF533 data sheet [13].

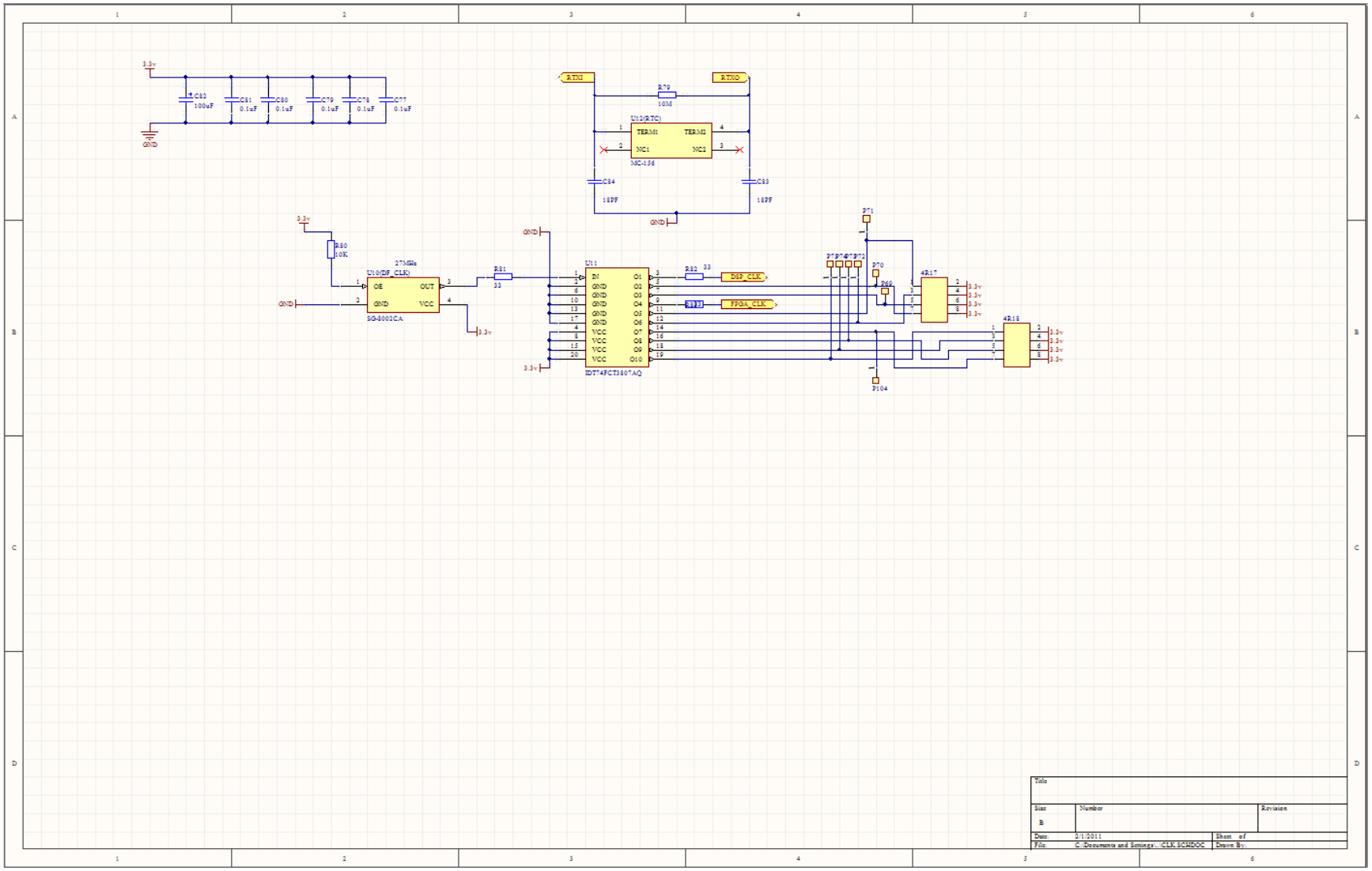


Figure 3.9 : CLKs schematic sheet

3.1.9 Reset, Push Buttons and LEDs

Four programmable push buttons, six programmable LEDs, one reset button with an LED and one power LED were implemented.

The reset button was connected to the ADM708SAR, which is a chip that provides an active low debounced manual reset input and an active high and low reset output. The reset signal was connected to the BF533, the FPGA, the PSDs and the UART-USB module. The BF533 resets the rest of the components on the board, namly, SDRAM, OV2640 and Nano WiReach. In addition, the reset signal was connected to the LED through the IDT74FCT3244 buffer; for the LED to turn ON whenever the reset button is pushed.

Six programmable LEDs were connected to Flash A, which does not provide enough current [20] to turn the LEDs ON, through the IDT74FCT3244 buffer to provide enough current to turn the LEDs ON [17].

The power LED was connected to 7 v for the LED to turn ON whenever there was power supplied to the board.

The voltage drops at the Reset LED (colour RED), the programmable LEDs (colour YELLOW) and the power LED (colour GREEN) were 1.7 v, 2.1 v and 2.2 v respectively, and the maximum current was 30 mA [29] as per the following calculations:

$$V = I \times R \Rightarrow I = \frac{Vcc - Voltage\ Drop(RESET\ LED)}{R} \Rightarrow I = \frac{3.3 - 1.7}{270} = 5.9\ mA$$

$$V = I \times R \Rightarrow I = \frac{Vcc - Voltage\ Drop(YELLOW\ LED)}{R} \Rightarrow I = \frac{3 - 2.1}{270} = 3.3\ mA$$

$$V = I \times R \Rightarrow I = \frac{Vcc - Voltage\ Drop(POWER\ LED)}{R} \Rightarrow I = \frac{7 - 2.2}{680} = 7\ mA$$

5.9 mA, 3.3 mA and 7 mA will thus be able to turn ON the LEDs without damaging them.

Four programmable push buttons were connected to a debouncing circuit, which is in the form of a low pass filter, to remove high frequency components from the mechanic switching of the push buttons. The push buttons were then connected to the BF533 through an inverter and a switch to disable or enable them (see Figure 3.11) [17].

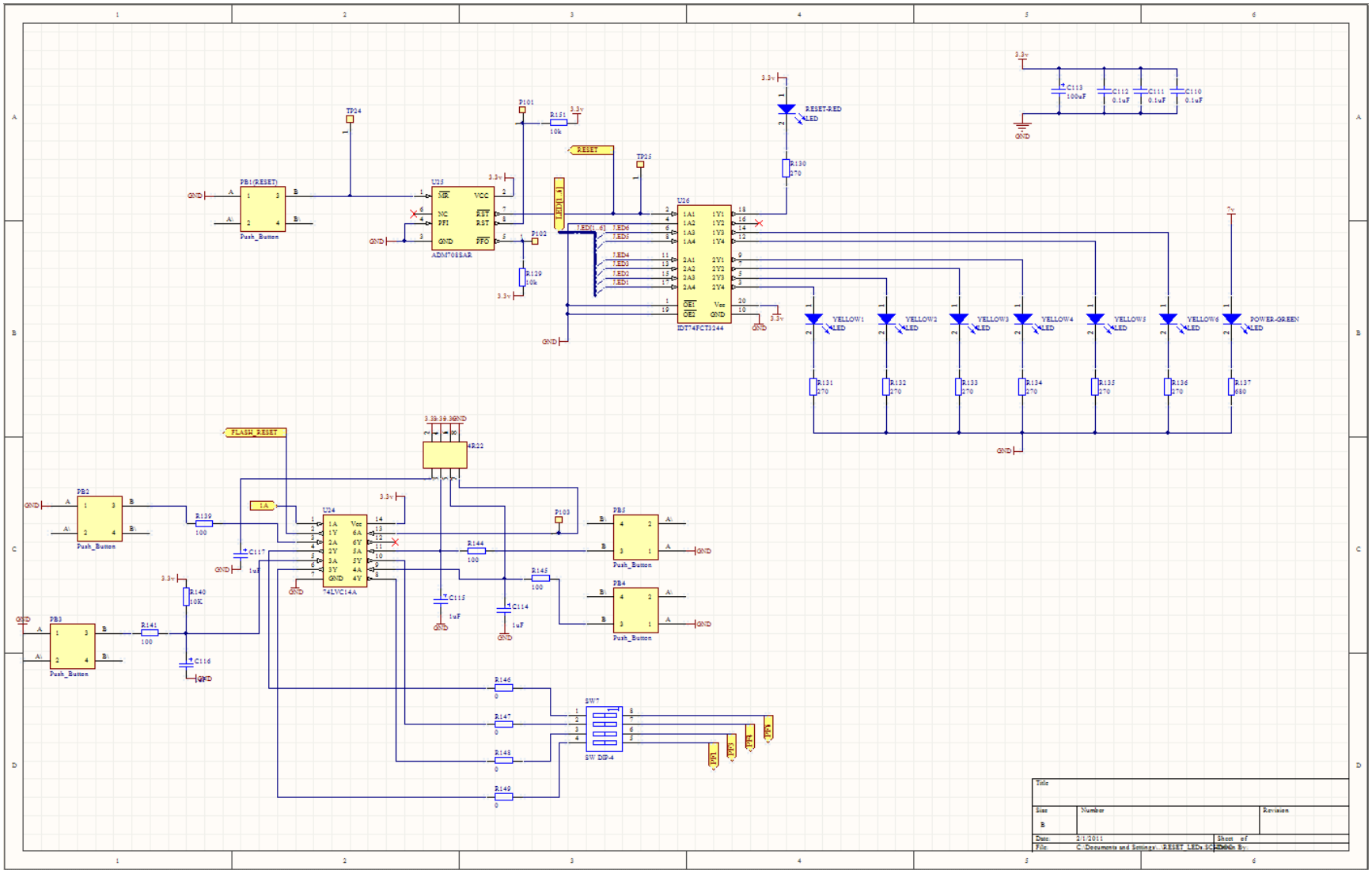


Figure 3.10 : Push buttons and LEDs schematic sheet

3.1.10 Regulators Block

The Regulator Blocks are responsible for supplying the right voltages to the components. The board input voltage is 7 v, six regulators have been used to manage the power distribution of the system (see Figure 3.11).

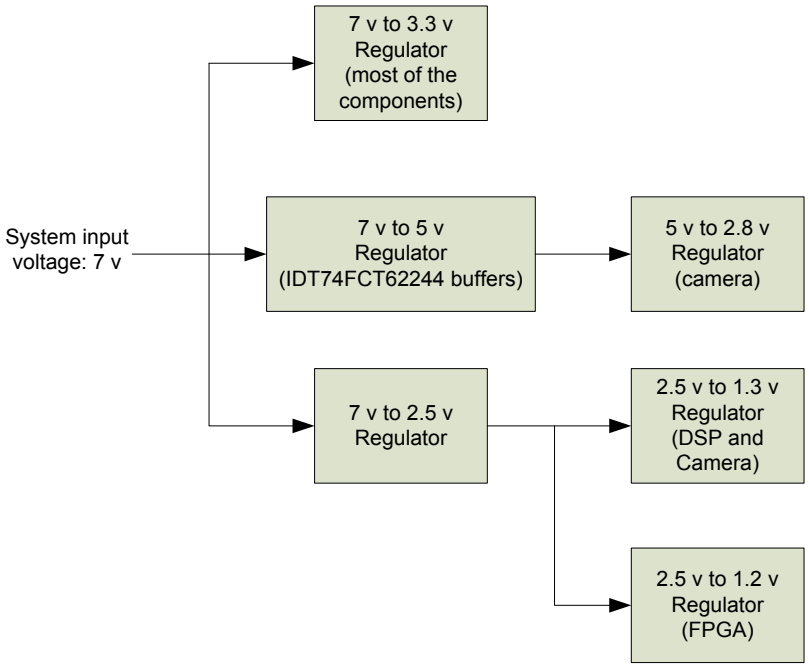


Figure 3.11: Power Distribution System

The technique, which was used to select the regulators, was to follow a start backwards approach, in which we assumed that all the components that share the same voltage were working at the same time at the maximum current rate (which is unlikely to happen). Those currents were added together and a regulator was chosen that could supply the required voltage with the required current. 0 Ω resistors were placed between the output of the regulators and the rest of the board. The reason for this was to be able to test the power in the board by supplying one voltage at a time.

The maximum current ratings of each of the 3.3 v devices were calculated as follows: two flash memories will consume 50 mA, the SDRAM 255 mA, the FPGA 100 mA, the DSP 308 mA and the WiFi will consume 280 mA. Adding these currents together resulted in a consumption of almost 1.1 A. Therefore, the PTN78000W switching regulator from Texas Instruments was chosen because it can output up to 1.5 A, and because its input voltage can vary from 7 v to 36 v. The output voltage was set by using a single external resistor; it can be set to any value within the range of 2.5 v to 12.6 v. The required output voltage for the chips described above was 3.3 v, which lies within the regulator voltage range. To set the output voltage, the following equation from the PTN78000W data sheet was used [33]:

$$Rset = 54.9\ k\Omega\ \times \frac{1.25\ v}{Vo - Vmin} -\ Rp$$

Vmin and Rp are given from the data sheet. Vo equals 3.3 v, Vmin equals 2.5 v and Rp equals 6.49 kΩ so, Rset will equal to 79.29 kΩ. The input voltage range can be from 7 v to 33 v according to the PTN78000W data sheet. 7 v is the input voltage of the board, which lies on the voltage range of this regulator (see Figure 3.12).

The core voltage of the FPGA was 1.2 v, whereas the DSP was 1.3 v and the maximum current rating for each was 0.5 A. Two ADP1715 from Analog Devices were chosen, because they can output up to 0.5 A, and because two of its common usage applications are for DSPs and FPGAs. The regulators output voltage was set by means of two external resistors; it can be set to any value within the range from 0.8 v to 5 v. The required output voltages were 1.2 v and 1.3 v, which lie within the output ranges of the regulators. The ADP1715 data sheet provided the following equation for setting the required output voltages [34]:

$$V_{out} = 0.8(1 +\ R_1/R_2\ )$$

For the FPGA, V_{out} was 1.2 v, resulting in R_1/R_2 being equal to a ratio of 0.5. For the DSP, V_{out} was 1.3 v, resulting in R_1/R_2 being equal to a ratio of 0.625. Furthermore, the ADP1715 presented the following equation for calculating the power dissipation of the chip (see Figure 3.12) [34]:

$$P_D = (V_{in} - V_{out}) \times I_{load}$$

Where P_D is the power dissipation, V_{in} and V_{out} are input and output voltages respectively, and I_{load} is the load current. Assuming that I_{load} is the maximum, which is 0.5 A, V_{in} is 2.5 v and V_{out} is 1.2 v, then P_D will be equal to 0.65 w. The same applies for V_{out} 1.3 v, where P_D will be 0.6 w. 0.65 w and 0.6 w are acceptable and fall within the limitations of the ADP1715 as per the data sheet [34].

The ADP1715 (1.3 v) and ADP1715 (1.2 v) regulators need an input of 2.5 v and the maximum current required from each of them is 0.5 A. Thus, a regulator that can supply 2.5 v and 1 A was required to supply the two regulators. Therefore, the PTN78000W was chosen to supply the 2.5 v [33]. As a result, the PTN78000W received a 7 v input and will output 2.5 v for the two regulators (see Figure 3.12).

The camera module requires 2.8 v and a maximum current of 10 mA. The ADP1715 was also chosen to supply the camera. V_{out} is 2.8 v, V_{in} is 5 v and I_{load} is 10 mA, resulting in R_1/R_2 being 2.5 and P_D being 0.022 w, which falls within the limitations of the ADP1715 as per the data sheet (see Figure 3.12) [34].

Three IDT74FCT62244 buffers require 5 v and a maximum current rating for each is 120 mA. In addition, the ADP1715 (2.8 v) needs 5 v and a maximum current of 10 mA. Adding all the currents together will result in a sum of 370 mA. Therefore, REG103-5 from Texas Instruments was chosen because it can output up to 500 mA, because the

output voltage is fixed at 5 v and because the input voltage can range from $V_{out} + 0.7$ to 15 v. The power dissipation can be calculated by using the same equation as above, I_{load} is 370 mA, V_{in} is 7.5 v and V_{out} is 5 v so, P_D will equal to 0.925 w, which is within the limitations (see Figure 3.12). The DDPAK package of REG103-5 was chosen due to the PCB heat sink configuration [35].

The input voltage of the board goes through a circuit before it enters the regulators. This circuit consists of a fuse for over-current protection, bulk capacitors to filter the incoming supply, a diode for reverse polarity, and finally a bleed resistor to discharge the capacitors when the board is off (see Figure 3.12).

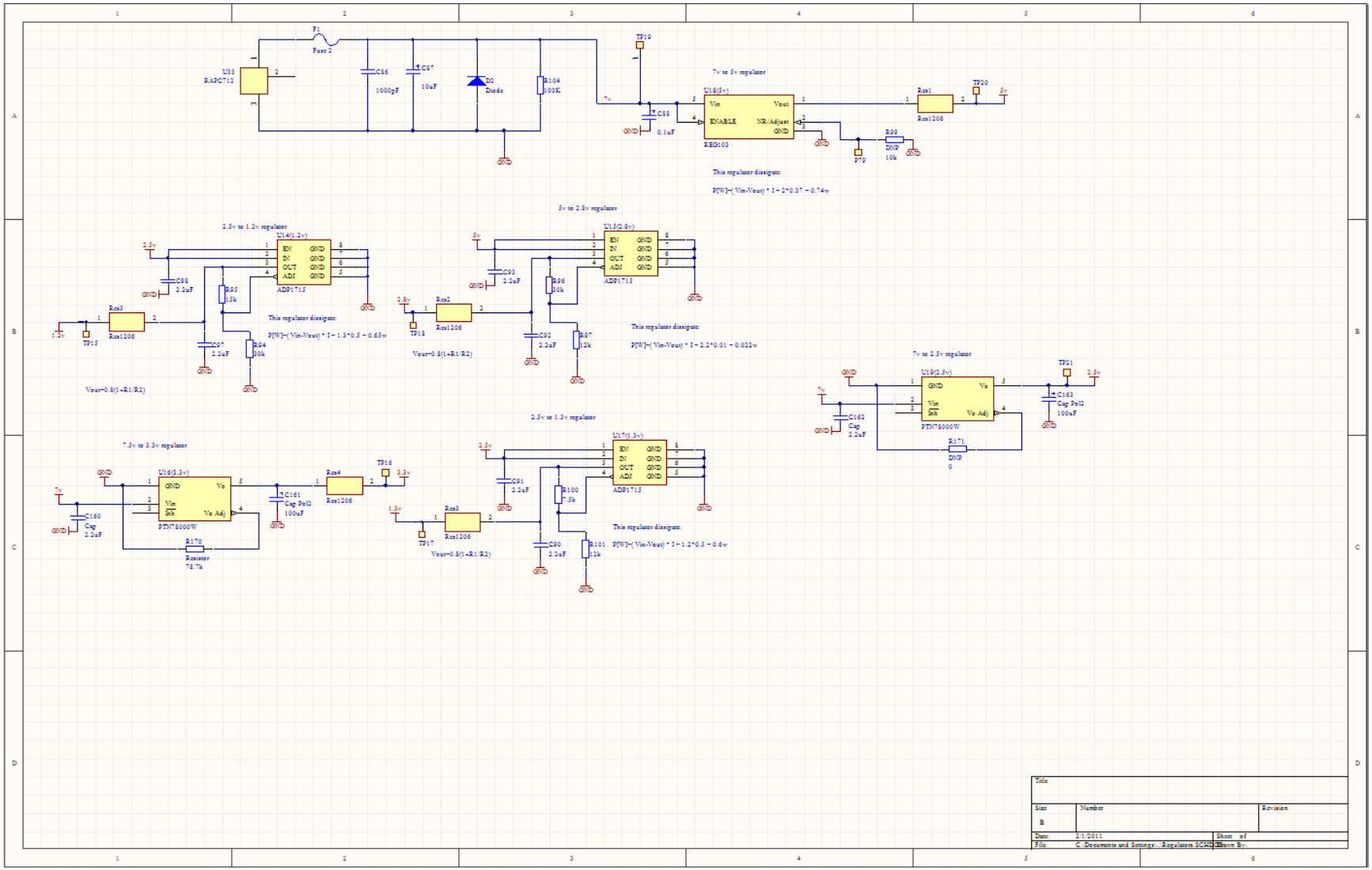


Figure 3.12 : Regulators schematic sheet

3.1.11 Level Translator

The system does not need any level translators between the chips. All the I/O voltages are compatible throughout the design (see Tables 3.4, 3.5 and 3.6).

Table 3.4 : I/O voltages between the BF533, PSDs and SDRAM

BF533	*PSD4256G6V*	*MT48LC32M16A2TG*
$V_{IH} = 2v\ to\ 3.6v$	$V_{OH} = 2.3v\ to\ 2.4v$	$V_{OH} = 2.4v$
$V_{OH} = 2.4v$	$V_{IH} = 2.31v\ to\ 3.8v$	$V_{IH} = 2v\ to\ 3.6v$
$V_{IL} = -0.3v\ to\ 0.6v$	$V_{OL} = 0.15v\ to\ 0.45v$	$V_{OL} = 0.4v$
$V_{OL} = 0.4v$	$V_{IL} = -0.5v\ to\ 0.8v$	$V_{IL} = -0.3v\ to\ 0.8v$

Table 3.5 : I/O voltages between the BF533 and the camera, WiFi and UART-USB modules

BF533	*Camera Module*	*WiFi Module*	*UART-USB Module*
$V_{IH} = 2v\ to\ 3.6v$	$V_{OH} = 2.2v.^{(1)}$	$V_{OH} = 2.9v\ to\ 3.1v$	$V_{OH} = 2.2v\ to\ 3.2v$
$V_{OH} = 2.4v$	$V_{IH} = 1.26v\ to\ 2.3v$	$V_{IH} = 2v\ to\ 3.6v$	$V_{IH} = Min(1.5v)$
$V_{IL} = -0.3v\ to\ 0.6v$	$V_{OL} = 0.18v$	$V_{OL} = 0.2\ to\ 0.4v$	$V_{OL} = -0.3v\ to\ 0.6v$
$V_{OL} = 0.4v$	$V_{IL} = -0.5v\ to\ 0.54v$	$V_{IL} = -0.3v\ to\ 0.8v$	$V_{IL} = Max(1v)$

(1): They specify V_{OH} = 2.2v in the original data sheet of the camera but they also specify 1.62v as minimum V_{OH} in the latest data sheet so, at the moment it is working but there might be a risk that it will not work in a different environment.

Table 3.6 : I/O voltages between the BF533, FPGA, ADM708, 74LVC14A and IDT74FCT3807

BF533	*FPGA*	*ADM708SAR*	*74LVC14A*	*IDT74FCT3807*
$V_{IH} = 2v\ to\ 3.6v$	$V_{OH} = Min(2.4v)$	$V_{OH} = 2.64v$	$V_{OH} = 3.1v$	$V_{OH} = 2.4v\ to\ 3v$
$V_{OH} = 2.4v$	$V_{IH} = Min(1.7v)$	N/A	$V_{IH} = 1.1v\ to\ 2v$	$V_{IH} = 2v\ to\ 5.5v$
$V_{IL} = -0.3v\ to\ 0.6v$	$V_{OL} = Max(0.45v)$	$V_{OL} = 0.3v$	$V_{OL} = 0.2v$	$V_{OL} = 0.2\ to\ 0.5v$
$V_{OL} = 0.4v$	$V_{IL} = Max(0.8v)$	N/A	$V_{IL} = -0.8v\ to\ 1.5v$	$V_{IL} = -0.5v\ to\ 0.8v$

3.2 PCB Layout

The board consisted four layers: two signal layers, a power layer and a ground (GND) layer (see Figure 3.13). The board was designed using Altium Designer Winter 2009, and it was manufactured by Trax company. A top and bottom views of the final prototype are presented in Figure 3.14 and Figure 3.15.

The following rules were followed when laying out the board:

- First, the critical line length (CLL) was calculated, which meant that none of the tracks should exceed that length, otherwise we would have had to deal with high speed digital design issues. CLL was calculated by using the following equation [36]:

$$CLL = \frac{T_{rf}}{2\,S}$$

Where

T_{rf} is the rise and fall time (10% to 90%).

The BF533 is the fastest chip on the board so the calculations were based on the T_{rf} of the BF533. The BF533 has four different driver types for the output pins, and each driver type has its own T_{rf}. By examining the BF533 data sheet we assumed that the worst case for the load capacitance is 50 pF at 3.3 v. Therefore, when investigating the T_{rf} diagrams in the BF533 data sheet, one can see that the worst cases are output pins of driver A ($T_{rf} = 4\,ns$) and the

CLK_OUT pin, which is driver B ($T_{rf} = 3.3\,ns$). $T_{rf} = 4\,ns$ was used to calculate the critical line length for all the tracks except CLK_OUT [13].

S is trace velocity. Trax uses FR4 material for manufacturing the PCB and the typical trace velocity for FR4 is $2\ ns/feet$ [37]:

$$For\ all\ the\ tracks: CLL = \frac{4}{2(2)} = 1\ feet = 12000\ mils$$

$$For\ CLK\_OUT: CLL = \frac{3.3}{2(2)} = 0.33\ feet = 9900\ mils$$

- The critical signals, such as CLK and some of the control signals, were routed first, making them as short as possible [38].
- The buck converter circuit tracks of the BF533 had to be as short as possible and as thick as possible [19].
- The GND and Vcc tracks were made as thick as possible and as short as possible, because they draw a lot of current (especially GND). In addition, the holes of their vias were made 1 mm.
- The address and data lines were routed randomly and not next to each other to avoid the cross talk problems.
- The GND layer was placed closer to the top layer because it draws more current than the bottom layer.
- The decoupling capacitors were placed as close as possible to the Vcc pins.
- The minimum track width was calculated for each track, which meant that each track should not be below that width, using the following equation [39]:

$$\text{Required Trace Width (Mils)} = \frac{\text{Copper Area (Square Mils)}}{(\text{Cu Thickness (Oounce per Square Feet)} \times 1.378)}$$

Where

$$\text{Cu Thickness} = 0.5 \text{ (oz per sq. ft.)}$$

$$\text{Copper Area for top and bottom layer} = \left(\frac{\text{Current}}{0.0647 * (\text{Temperature Rise})^{0.4281}} \right)^{\left(\frac{1}{0.6732}\right)}$$

$$\text{Copper Area for internal layers} = \left(\frac{\text{Current}}{0.015 * (\text{Temperature Rise})^{0.5453}} \right)^{\left(\frac{1}{0.7349}\right)}$$

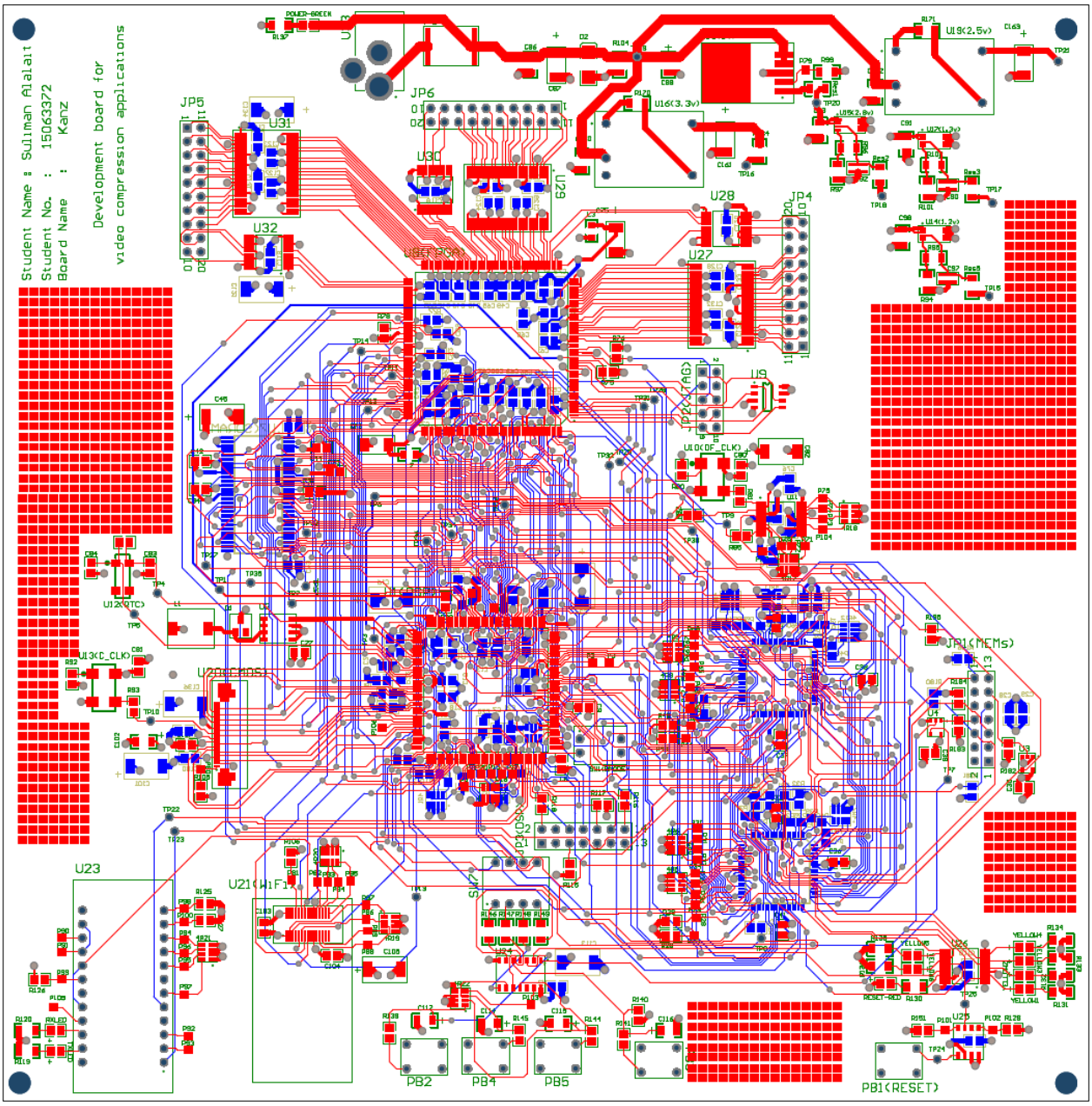


Figure 3.13 : The top and bottom layer of the PCB design (a 0805 footprint was placed all over the unused space for future use)

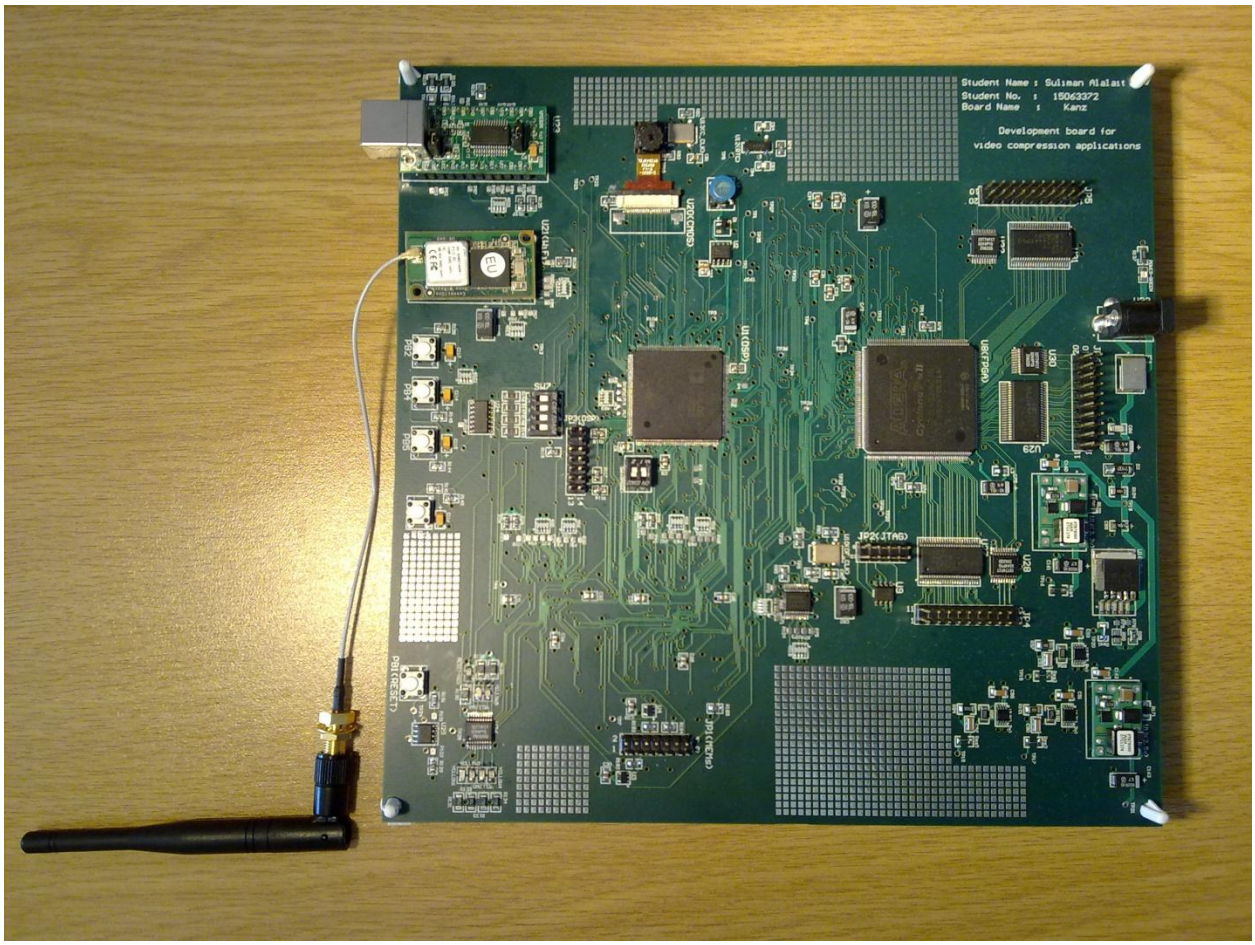


Figure 3.14: Top view of the final prototype

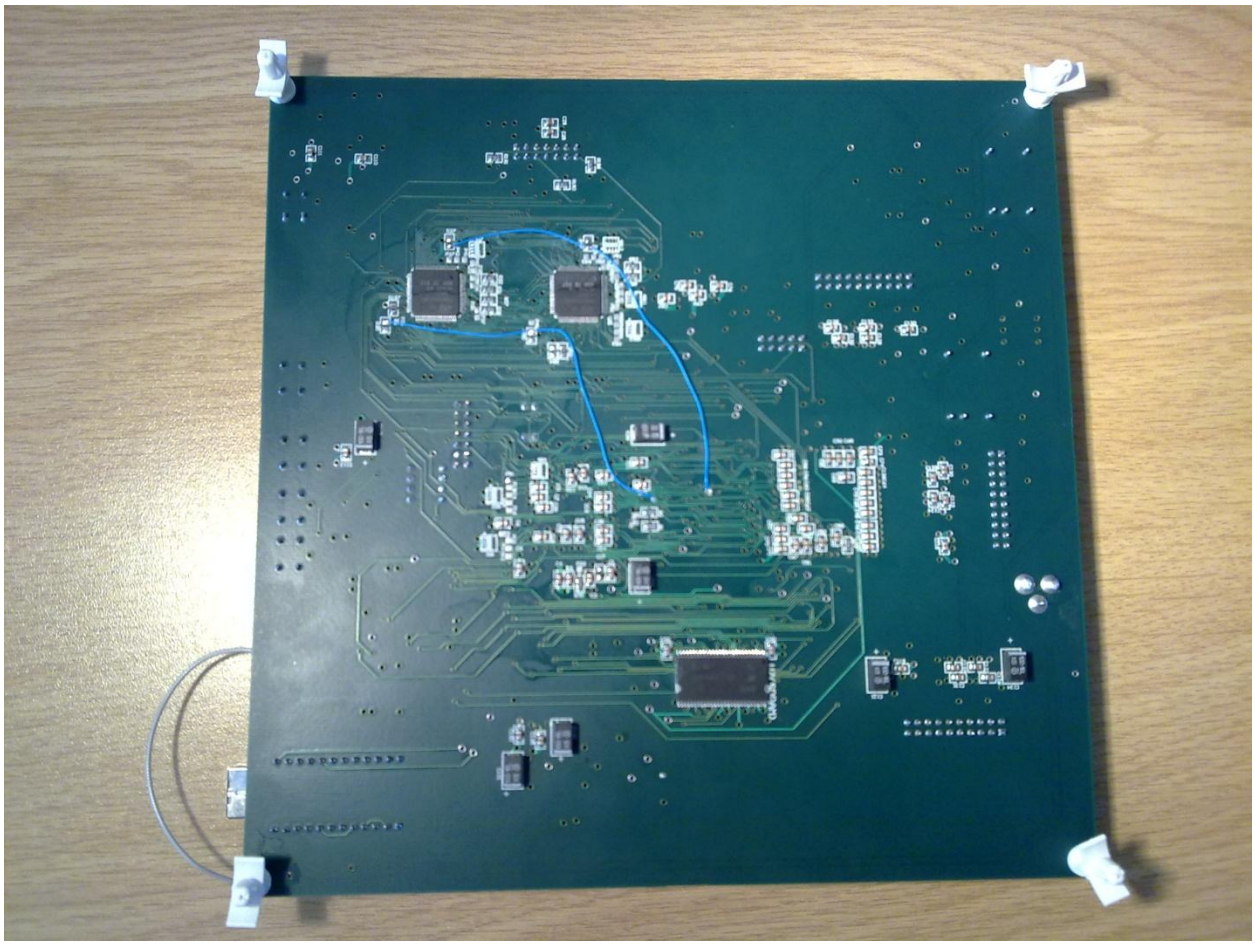


Figure 3.15: Bottom view of the final prototype

3.3 Conclusion

The system design was depicted in this chapter in Figure 3.1. Subsequently, the implementation of each sub-system was described, and hardware selections were justified. Furthermore, this chapter listed the rules that were followed during the schematic design and PCB lay-out phases. Finally, a picture of the final prototype board was presented. In summery, the prototype board contains 29 components, 6 connectors, 2 switches, 5 push buttons, 10 LEDs and 240 miscellaneous components (resistors, capacitors, inductors, diodes and fuse). The next chapter will cover the software design and hardware debugging.

4. Chapter 4: Software Design and Hardware Debugging

4.1 Software Overview

The structure of the SW was designed in such a way that it was as simple and straightforward as possible. Three main cycles were identified in the software, namely, camera cycles, encoding cycles and WiFi cycles. Camera cycles refer to the time it takes to set up DMA and PPI to start transferring the frame from the camera to SDRAM; encoding cycles are the time it takes to encode one frame; and WiFi cycles are the time it takes to transmit one frame.

The software process starts by initializing the hardware. This is followed by, grabbing the first frame from the camera and waiting until the grabbing is done. After that, the encoding of the first frame starts while the second frame is being grabbed at the same time. Thereafter, the encoded frame is transmitted. This same process repeats for the second frame. In other words, while the first frame is being encoded, the second frame is grabbed, this happens in a ping-pong manner (see Figure 4.1). The DMA is responsible for grabbing the frames from the camera.

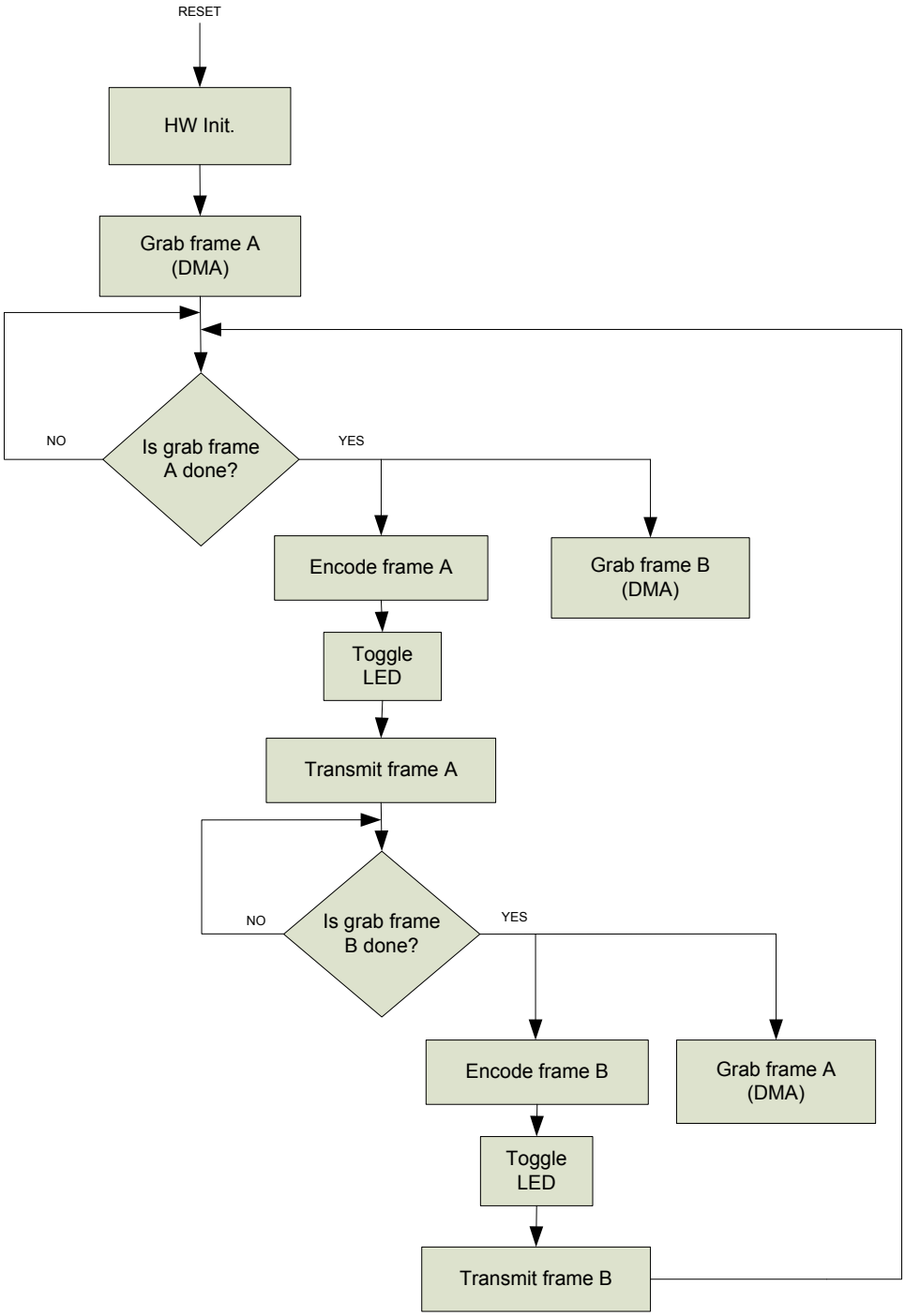


Figure 4.1 : SW flowchart

4.1.1 Hardware Initialization

Hardware initialization involves setting up all the registers that will change during the process (see Figure 4.2).

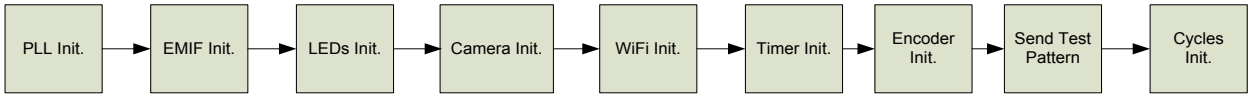


Figure 4.2 : Hardware initialization

4.1.1.1 Phase Locked Loop (PLL) Initialization Function

This sets up the core voltage of BF533 to 1.3 v, the CCLK to the maximum, which is 378 MHz (27 MHz × 14) and the SCLK to the maximum, which is 126 MHz (CCLK ÷ 3).

The registers programmed in this function are VR_CTL, PLL_DIV and PLL_CTL [32].

4.1.1.2 External Memory Interface (EMIF) Initialization Function

This sets up the registers of the EBIU to enable all the banks. It also sets up the registers of SDRAM refresh rate control, SDRAM memory bank control and SDRAM memory global control.

The registers programmed in this function are EBIU_AMBCTL, EBIU_AMGCTL, EBIU_SDRRC, EBIU_SDBCTL and EBIU_SDGCTL [32].

4.1.1.3 LEDs Initialization

This sets port B of Flash A as output to turn the LEDs ON or OFF. Thereafter, the first LED is turned ON to indicate that BF533 has power.

The registers programmed in this function are Direction and Data-Out registers of Flash A port B [22].

4.1.1.4 Camera Initialization Function

The I^2C protocol has to be initialised first, and BF533 must be set as a master. Then the power down mode of the camera must be removed by pulling the PWDN pin low and bringing the camera out of reset by pulling the $\overline{\text{RESETB}}$ pin high [23].

There are two different sets of register banks on the OV2640 module. Register 0xFF controls which sets are accessible. When register 0xFF = 00, Table 12 of the camera datasheet is effective. When register 0xFF = 01, Table 13 is effective. As part of the initialization, register 12 in Table 13 sets to 8, which initiates a system reset: this means that all the registers are set to factory default values, after which the chip resumes normal operation. Then the values of all the registers in Table 12 and 13 are set for the correct operation, such as setting up the size to UXGA, setting PCLK to the maximum etc...[25].

Thereafter, DMA is initialised to transfer the frame from PPI to SDRAM. Each pixel is two bytes, therefore X_COUNT has to be 16 bit and X_MODIFY has to be 2 and the same for Y_COUNT and Y_MODIFY. Then, the PPI interface is initialised to receive a frame of 1600x1200 pixels.

The registers programmed in this function are FIO_DIR, FIO_INEN, FIO_FLAG_S, FIO_FLAG_C, FIO_FLAG_D, DMA0_X_COUNT, DMA0_X_MODIFY, DMA0_Y_COUNT, DMA0_Y_MODIFY, DMA0_PERIPHERAL_MAP, DMA0_CONFIG, PPI_FRAME, PPI_COUNT, PPI_DELAY and PPI_CONTROL [32].

4.1.1.5 WiFi Initialization Function

First, the WiFi module must be reset by pulling the $\overline{\text{RESET\_WiFi}}$ pin low and then bringing it out of reset by pulling the same pin high. Then, SPI is initialised by setting the baud rate register, control register and SPI flag register. BF533 is the master and the maximum SPI CLK of the WiFi module is 12 MHz. Therefore, the baud rate register must be set to the maximum, which is 10.5 MHz (SCLK ÷ 12) [26].

The registers programmed in this function are SPI_BAUD, SPI_CTL and SPI_FLG [32].

Thereafter, the module is configured by sending commands in a specific format as specified in the programmer's manual [27]. Configuring the module means setting up the WiFi communication channel to Ad-Hoc mode, the destination wireless LAN Service Set Identifier (SSID), the IP address of the module and number of addresses to be allocated in the IP pool of the module. Then a UDP (User Datagram Protocol) socket must be opened to allow streaming and to set the remote system's address.

The WiFi module was programmed by sending the following commands "AT+iWLCH=1\r\n", "AT+iWLSI=!KANZ\r\n", "AT+iIPA=192.168.10.1\r\n", "AT+iDPSZ=1\r\n", "AT+iDOWN\r\n", "AT+iUP=2\r\n" and "AT+iSUDP:192.168.10.2,3139\r\n" [27].

4.1.1.6 Timer Initialization Function

This sets up TIMER0 and the Interrupt Service Routine (ISR) for measuring activity, such as data throughput or frame per second; it also toggles one of the LEDs every second.

The registers programmed in this function are TIMER0_CONFIG, TIMER0_PERIOD, TIMER0_WIDTH, TIMER_ENABLE, SIC_IAR0, SIC_IAR1, SIC_IAR2 and SIC_IMASK [32].

4.1.1.7 Encoder Initialization Function

This prepares the encoder for encoding based on the step-by-step guide for integrating the H.264 BP/MP Encoder library into the application as described in Section 3.2 of the H264 encoder developer's guide document [12]. Steps 1 and 2 were implemented in the *h264enc.c* file, which include *adi_h264e_codec.h* and *adi_codec.h* and declare all the required buffers and variables. Steps 3 to 8 were implemented in this function *H264_Init*. They include setting up the encoder input parameters *ADICodecConfigInputParam* (where features of the encoder are enabled or disabled, thus affecting the quality of the video and processing cycle), setting up the memory blocks, setting up the MDMA (Memory DMA) configuration, calling the *ADIH264ECodecNew()* function to create a new encoder instance, configuring the encoder instance with input parameters, configuring the encoder video input format to UYVY 422 and calling the sequence header process function to process the sequence header before getting the header encoded from the encoder [12]. Finally, the *H264_Init* function returns the start address of the header output stream and size.

4.1.1.8 Send Test Pattern

SMPlayer does not play the output video directly; and it was found that SMPlayer only starts playing the video when it has first been sending a test pattern. The UDP packet size is limited to 2048 bytes and normally the video output size is about 12 Kbyte; as a result, the video is divided to 2048 bytes [27]. It seems that SMPlayer cannot recognise the beginning of the video, because it has been divided among the packets. However, the test pattern is less than 2048 bytes, along with the header that *H264_Init* function returned (see Section 4.1.1.7). Therefore, a test pattern file has to be streamed first via *WiReach_SendUdpBuf* function (more details about this function are presented in Section 4.1.5).

The test pattern was obtained by setting the registers in the camera to send only the test pattern, and then transferring it to the SDRAM through the PPI. Once this was done, the test pattern was dumped out from the SDRAM to the “test1600x1200.txt” file.

4.1.1.9 Initialization of Cycles

This initialises a cycle count for the encoder, camera and WiFi module to keep track of how many cycles each one utilizes. Afterwords, these numbers would be printed out for use in calculating of the bit rate and some other values.

4.1.2 Grab Frame Function

This grabs frame A by setting the frame starts address, configuring and enabling the DMA and enabling PPI. The DMA is responsible for grabbing the frames from the camera.

The registers programmed in this function are DMA0_START_ADDR, DMA0_CONFIG and PPI_CONTROL [32].

4.1.3 Is Grab Frame Done Function

This involves checking the DMA_IRQ_STATUS register: if it is done, then PPI must be disabled, returning 1, otherwise returning 0.

The registers programmed in this function are DMA0_IRQ_STATUS and PPI_CONTROL [32].

4.1.4 Encode Frame Function

This involves passing a two pointer, one for the input video frame and one for where the output stream should be stored. This procedure causes the library encoder function to be executed, based on the step-by-step guide for integrating the H.264 BP/MP Encoder library into the application, according to Section 3.2 in the H264 encoder developer's guide document [12]. Steps 9 and 10 were implemented in this function known as *H264_Encode*, which assigned the input video frame to the instance and called the encoder to process the frame [12]. The *H264_Encode* function then returns the start address of the output stream and size.

4.1.5 Transmit Frame Function

The WiFi module can only send UDP packets of up to 2048 bytes. Therefore, a while loop was implemented to send only 2048 bytes at a time until the whole output

stream has been sent. This function *WiReach_SendUdpBuf* sends one packet of 2048 bytes through the SPI_TDBR register and then waits for acknowledgement through the SPI_RDBR register. *WiReach_SendUdpBuf* function returns the number of bytes that have been sent.

One WiFi command was used in this function, namly, "AT+iSSND:0,%04u:" [27] and two registers which were used, namly, SPI_TDBR and SPI_RDBR [32].

4.2 PSD4256G6V Software

The PLDs in PSD4256G6V were programmed by installing *flash_a.obj* and *flash_b.obj* configuration files from the example code folder of Analog Devices. The Anaolg Devices folder that was used to program the PSDs are covered in Appendix E.3.

4.3 Software Design Tools

The tools that have been used in the project are VisualDSP++ version 5 update 7 and ADZS-USB-ICE emulator from Analog Devices to program BF533 DSP. PSDsoft Express and Flash Link FL-101D from STMicroelectronics were used to program the PSDs. SMPlayer, which was recommended by Analog Devices, was used to play the output stream video [12].

4.4 Hardware Debugging

No voltage could be measured at the output of the REG103-5 when the power was first connected to the board because the ENABLE pin on the REG103-5 was faultily grounded. To rectify this, the ENABLE pin was connected to the Vin pin [35], which led to the right voltages being measured at all the affected output regulators.

When the regulators were connected to the rest of the board and power was applied for the second time, the zener diode in the buck converter circuit was burned, causing smoke to come out of it. This was because the footprint of the zener diode had been placed the wrong way around. The zener diode had to be replaced. The camera also became very hot because the footprint of the camera connector was the wrong way around, and the camera had to be replaced with a new one as well.

The BF533 JTAG emulator was connected to the board with the purpose of loading an example application to test the SDRAM and Flash memories. The SDRAM test was successful. However, when an application was loaded to the BF533 to flash some LEDs through Flash A, it was unsuccessful. After that, *flash_a.obj* and *flash_b.obj* configuration files from Analog Devices were loaded into Flash A and Flash B through their own JTAG, but the LEDs still did not flash. All the tracks between the BF533 and PSDs were checked, and it was found that two control signals $\overline{\mathrm{ABE0}}/\mathrm{SDQM0}$ and $\overline{\mathrm{ABE1}}/\mathrm{SDQM1}$ were not connected. Therefore, two wires were manually placed for each PSD. When the code was loaded again, the test was successful.

The UART-USB module was emitting considerable heat when the USB cable was first plugged in; this was because the footprint was the wrong way around. Unfortunately, the UART-USB module was a thru-hole module and thus the process of replacing the module was difficult and time consuming. However, as the module was not critical for the project, it was left out.

The $\overline{\mathrm{AMS1}}$ and $\overline{\mathrm{AMS0}}$ signals should not have been connected to the FPGA because $\overline{\mathrm{AMS0}}$ was reserved for Flash A and $\overline{\mathrm{AMS1}}$ for Flash B (see Table 3.3). The FPGA has 15 Kbyte of memory [30]. Instead, $\overline{\mathrm{AMS2}}$ or $\overline{\mathrm{AMS3}}$ should have been connected, because $\overline{\mathrm{AMS2}}$ still has approximately 831 Kbyte left, which is more than enough for the FPGA. The signal $\overline{\mathrm{AMS3}}$ was not even used. At the moment, $\overline{\mathrm{AMS1}}$ could be used for the FPGA because Flash B is not being used.

The JTAG circuit of the FPGA was not implemented, and the only way of programming it was through the EPCS1. This is because the FPGA JTAG signals were dropped off errantly and not connected (see Figure 3.12). Consequently, it is currently very difficult to program the FPGA, and since it is not critical for this project, the FPGA was never used.

4.5 Conclusion

In this chapter, a flow chart of the software system design was presented as well as a detailed description for each block. Thereafter, the PSD4256G6V software was explained, and the SW design tools that were used for the project implementation were presented. Some errors were discovered in the board design, and those were listed in the hardware debugging section. System testing and results will be discussed in the next chapter.

5. Chapter 5: System Testing and Results

5.1 Introduction

Users can adjust the quality of the video or processing time by tuning the numbers in the configuration parameters, which can enable or disable some of the features [12].

Analog Devices recommended three profiles, namely, Movie profile, Low Cost Surveillance profile and Surveillance profile. In the Movie profile, quality is more important than low processing time; in the Low Cost Surveillance profile, low processing time is more important than quality; and in the Surveillance profile, a trade-off between the two is important [12].

A sample test for each of these profiles was done to prove that the structure of the system was indeed working, as well as to identify any obstacles or bottlenecks, and to overcome these.

5.2 System Tests

5.2.1 Configuration Parameters

The test was done for the UXGA frame size by setting the parameters of *ADICodecConfigInputParam* in the *init_encoder_settings* function. Most of the parameters were kept at the default values, as per the code example of Analog Devices except five parameters, namely, iFramerate, iBitrate, iSearchComplexity, iScdConfig and iRcConfig [12]:

- iFramerate was set to the maximum, which is 30.
- iBitrate was set to the maximum, which is 2 Mbps.

For each profile, Analog Devices suggested values for iSearchComplexity, iScdConfig and iRcConfig parameters. iSearchComplexity configures the complexity of the encoder, iScdConfig configures the scene change detector, and iRcConfig configures the bit rate controller [12].

5.2.2 Movie Profile Parameters

iSearchComplexity was set to 0x6A1322D7, which set the motion search range to 64x32 and enabled the following: half pixel ME search, diagonal half pixel ME search, quarter pixel ME search, higher complexity ME search, alternate ME, predicted skip MBs, and diagonal intra4 prediction modes. It also set the maximum number of search depth to the default value 2 (see Appendix D) [12].

iScdConfig was set to 0x10A08932, which enables the scene change detection (see Appendix D) [12].

iRcConfig was set to 0x542CCF5, which sets the rate control sensitivity index to 5, the minimum QP (Quantization Parameter) to 15, and the maximum QP to 51, while disabling slice based rate control for I-frames and enabling variable GOP at scene change (see Appendix D) [12].

5.2.3 Surveillance Profile Parameters

iSearchComplexity was set to 0x6A112091, which sets the motion search range to 32x16 and enables and disables the following: enable half pixel ME search, disable diagonal half pixel ME search, disable quarter pixel ME search, enable higher complexity ME search, disable alternate ME, enable predicted skip MBs, and disable diagonal intra4 prediction modes. It also sets the maximum number of search depth to the default value 2 (see Appendix D) [12].

iScdConfig was set to 0x38932, which disables the scene change detection (see Appendix D) [12].

iRcConfig was set to 0x440CCF5, which sets the rate control sensitivity index to 5, the minimum QP to 15 and the maximum QP to 51, while disabling slice based rate control for I-frames and disabling variable GOP at scene change (see Appendix D) [12].

5.2.4 Low Cost Surveillance Profile Parameters

iSearchComplexity was set to 0x6A1124D1, which sets the motion search range to 32x16 and sets the following: enable half pixel ME search, disable diagonal half pixel ME search, disable quarter pixel ME search, enable higher complexity ME search, enable alternate ME, enable predicted skip MBs, and disable diagonal intra4 prediction modes. It also sets the maximum number of search depth to the default value 2 (see Appendix D) [12].

iScdConfig was set to 0x38932, which disables the scene change detection (see Appendix D) [12].

iRcConfig was set to 0x440CCF5, which sets the rate control sensitivity index to 5, the minimum QP to 15 and the maximum QP to 51, while disabling the slice based rate control for I-frames and disabling variable GOP at scene change (see Appendix D) [12].

5.2.5 Analog Devices' Profiles Test

The average encoding cycles, WiFi cycles, camera cycles and frame size of 200 frames were printed out for each profile to be able to calculate more or less the actual bit rate and frame rate for each profile (see Table 5.1).

Table 5.1 : Analog Devices' profile test results

Movie Profile		Surveillance Profile		Low Cost Surveillance Profile	
Average Encoding Cycles (AEC)	177403533 Cycles	Average Encoding Cycles (AEC)	167117555 Cycles	Average Encoding Cycles (AEC)	157524020 Cycles
Average WiFi Cycles (AWC)	30949197 Cycles	Average WiFi Cycles (AWC)	30933404 Cycles	Average WiFi Cycles (AWC)	31388640 Cycles
Average Camera Cycles (ACC)	399 Cycles	Average Camera Cycles (ACC)	401 Cycles	Average Camera Cycles (ACC)	401 Cycles
Average Frame Size (AFS)	12102 Bytes	Average Frame Size (AFS)	12063 Bytes	Average Frame Size (AFS)	12310 Bytes
Calculated Results:		Calculated Results:		Calculated Results:	

Cycles per pixel based on encoding cycles = $\frac{AEC}{(1600 \times 1200)^{(1)}}$	92 Cycles/pel	Cycles per pixel based on encoding cycles = $\frac{AEC}{(1600 \times 1200)^{(1)}}$	87 Cycles/pel	Cycles per pixel based on encoding cycles = $\frac{AEC}{(1600 \times 1200)^{(1)}}$	82 Cycles/pel
Encoding time = $\frac{AEC}{CCLK^{(2)}}$	0.46 Seconds	Encoding time = $\frac{AEC}{CCLK^{(2)}}$	0.44 Seconds	Encoding time = $\frac{AEC}{CCLK^{(2)}}$	0.41 Seconds
fps based on encoding time only = $\frac{1}{Encoding\ time}$	2.13 fps	fps based on encoding time only = $\frac{1}{Encoding\ time}$	2.26 fps	fps based on encoding time only = $\frac{1}{Encoding\ time}$	2.39 fps
Actual time $\approx \frac{(AEC+AWC+ACC)}{CCLK^{(2)}}$	0.55 Seconds	Actual time $\approx \frac{(AEC+AWC+ACC)}{CCLK^{(2)}}$	0.52 Seconds	Actual time $\approx \frac{(AEC+AWC+ACC)}{CCLK^{(2)}}$	0.49 Seconds
fps based on actual time = $\frac{1}{Actual\ time}$	1.81 fps	fps based on actual time = $\frac{1}{Actual\ time}$	1.9 fps	fps based on actual time = $\frac{1}{Actual\ time}$	2 fps
Real Bit Rate $\approx (AFS \times 8) \times fps\ based\ on\ actual\ time$	176 Kbps	Real Bit Rate $\approx (AFS \times 8) \times fps\ based\ on\ actual\ time$	185 Kbps	Real Bit Rate $\approx (AFS \times 8) \times fps\ based\ on\ actual\ time$	197 Kbps

(1): UXGA frame size is 1600*1200.

(2): CCLK = 14*27MHz .

It was found that the encoding cycles are dependent on the configuration parameters. Some features add more cycles while others add less. WiFi cycles are dependent on the output frame size, and thus the bigger the size, the higher the number of cycles, and the smaller the size, the lower the number of cycles. Camera cycles are more or less the same on all the profiles, because their aim is to ensure that the previous frame is grabbed before reconfiguring DMA and PPI to grab the next frame.

It was also found that the input frame size is 3750 Kbytes and that the average output frame size is almost 12 Kbytes; therefore, the compression ratio of the encoder is more or less 312:1. The quality of the video was good for all the profiles, but very jerky because the actual frame rate was only 2 fps. It is generally recommended that the video frame rate is at least 6 or 7 fps to be smooth enough for the human eyes. There were not any significant differences in terms of video quality between the profiles, nor were there significant differences in terms of the speed. More focused testing in terms of characterising the encoder parameters would need to be done to acquire the optimal numbers to suit the target application, but this lies outside the scope of this project.

It was also found that the encoding cycles consume almost 85% of the processing time, whereas the WiFi cycles used up 14% and the camera cycles only 0.1%, which is almost negligible. Therefore, to improve the actual frame rate, which would result in a less jerky video, the encoding cycles and WiFi cycles would have to be smaller. Ideas on how to improve both encoding cycles and WiFi cycles are presented in Chapter 6.

5.3 Conclusion

Testing of the three main steps of the system proved that the system is indeed working: UXGA frames were captured, compressed and then sent through a radio link. Furthermore, the bottlenecks of the system were identified and defined. Ideas and suggestions on how to improve on these are presented in the following chapter.

6. Chapter 6: Improvements and Conclusion

6.1 Improvements and Future Work

The ideas and recommendations that are presented in this chapter focus on overcoming the obstacles and problems of the system; the two most important of these were found to be encoding time and transmitting time. Furthermore, some suggestions are presented with regard to improvements in the hardware.

In the sections below, suggestions are first made with regard to the selection of future hardware; thereafter, ideas are presented on how to improve the present hardware and software; finally, an entirely new structure for a product that could really be used for an UAV is explained below.

6.1.1 Hardware Selections

A single 24 MHz crystal clock could have been used instead of two crystal clocks. 24 MHz is slightly better (see Section 3.1.8) than 27 MHz, but not significantly. For example, if 24 MHz had been used, then the encoding time for the Movie profile would be 0.461988367 seconds instead of 0.469321516 seconds (see Table 5.1).

Also, a single 32 Megabit SPI serial flash (SST25VF032B) could have been used instead of two 8 Megabit PSD425. SST25VF032B has only four signals to route instead of 16 data lines, 19 address lines, 4 control signals and 2 memory select; furthermore, SST25VF032B is much cheaper and smaller than PSD425. The UART protocol could have been used for the WiFi module instead of SPI, and the SPI protocol could have been used for SST25VF032B. Conversely, the UART-USB module could not have been used; it is nice to have but not critical to the structure of the system. Therefore, the advantages of having SST25VF032B outweigh the advantages of the UART-USB module.

Using one 24 MHz clock and one 32 Megabit SPI serial flash would have resulted in the need for fewer components and easier routing.

6.1.2 Recommended Work Using the Present Hardware and Software

Encoding cycles and WiFi cycles are the main bottlenecks of the system. Nothing can be done about the encoding cycles because we do not have access to the encoder itself, except that we could have used a smaller frame. This would have resulted in fewer encoding cycles, which will be discussed in Section 6.1.3.

The problem with the WiFi module was that the DSP could not transmit the next packet until it had received an acknowledgement. Using a DMA to transmit the encoded frame would have added software complexity, however; there would also not have been a significant difference or improvement because the DSP still needed to monitor the acknowledgements and then reconfigure the DMA.

The SPORT protocol could be used to send the encoded frame to the FPGA, which could then send it to the WiFi module; in this way, grabbing, encoding and transmitting could happen in parallel. Such a new software structure would still consist of three main cycles, but the WiFi cycles would be limited to the time it takes to set up DMA and SPORT to start transferring the frame from the SDRAM to the FPGA, which is almost the same as the length of the camera cycles (see Figure 6.1). Therefore, the WiFi cycles would be more or less 400 cycles, just like the camera cycles. If the same test as that of the Movie profiles in Section 5.2.5 was used to test the new structure, then the actual time would be 0.469 seconds rather than 0.551 seconds; the actual frame rate would be 2.13 fps rather than 1.81 fps (see Table 5.1 and Table 6.1). In other words, the new structure would improve the speed of the system but not significantly.

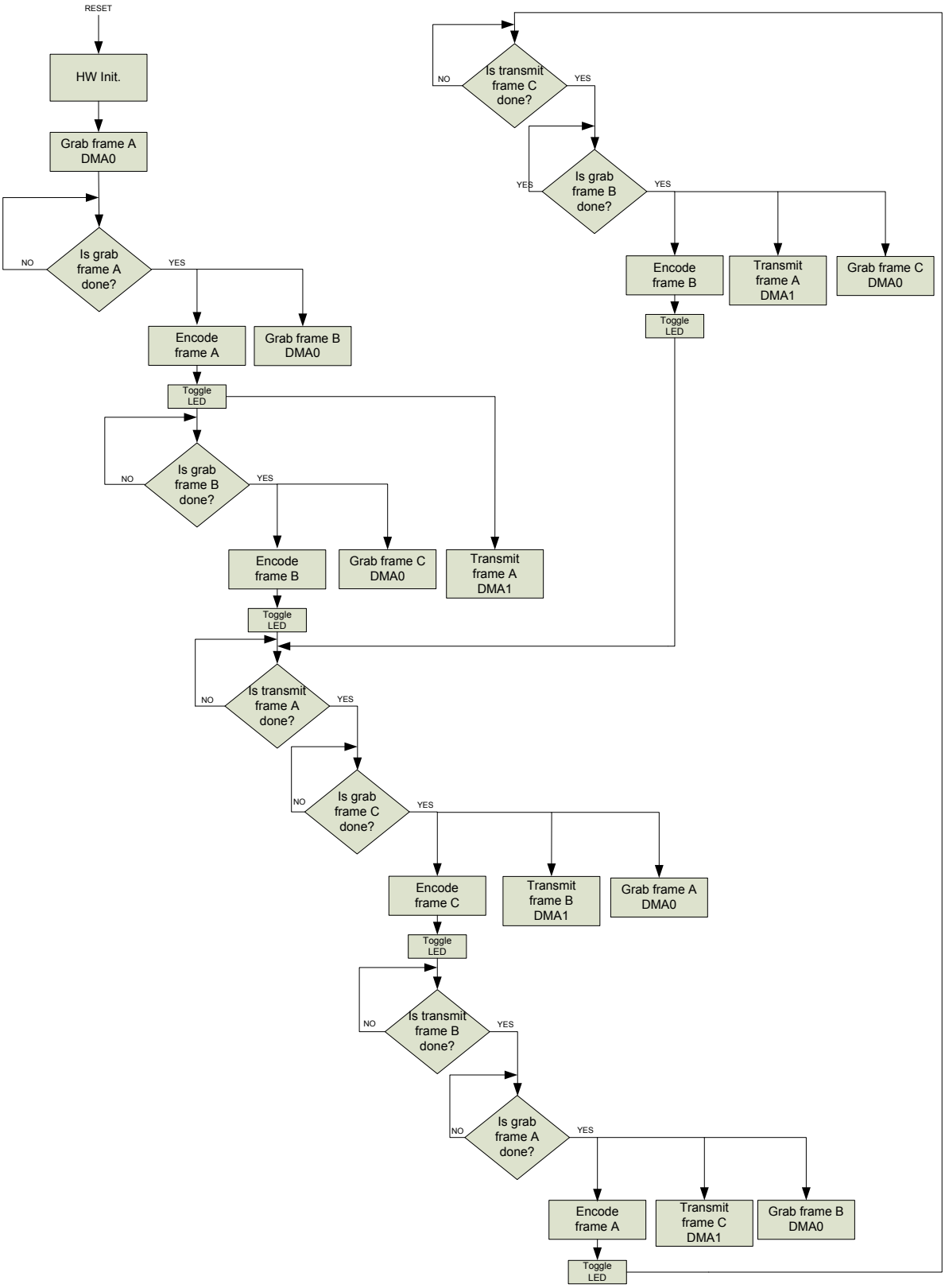


Figure 6.1 : Recommended software structure

Table 6.1 : Movie profile with only 400 WiFi cycles

Movie Profile	
Average Encoding Cycles (AEC)	177403533 Cycles
Average WiFi Cycles (AWC)	400 Cycles
Average Camera Cycles (ACC)	400 Cycles
Average Frame Size (AFS)	12102 Bytes
Expected Results:	
Cycles per pixel based on encoding cycles $= \frac{AEC}{(1600 \times 1200)^{(1)}}$	92 Cycles/pel
Encoding time $= \frac{AEC}{CCLK^{(1)}}$	0.469 Seconds
fps based on encoding time only $= \frac{1}{Encoding\ time}$	2.13 fps
Actual time $\approx \frac{(AEC+AWC+ACC)}{CCLK^{(1)}}$	0.469 Seconds
fps based on actual time $= \frac{1}{Actual\ time}$	2.13 fps
Real Bit Rate $\approx (AFS \times 8) \times fps\ based\ on\ actual\ time$	207 Kbps

(1): UXGA frame size is 1600*1200.

(2): CCLK = 14*27MHz .

The SPI protocol is already connected between the FPGA and WiFi module, but wires still have to be placed between the SPORT pins in the BF533 and FPGA.

6.1.3 Recommended New Hardware and Software Structure

In order to make the system faster, the encoding time has to be low and the only way to achieve this is to use smaller frames. This means that the picture must be divided into small pieces, using multiple DSPs to process them in parallel.

The proposed new structure would have four DSPs, each with its own SDRAM, one FPGA, Camera and WiFi module (see Figure 6.2). Consequently, the DSPs would be able to boot from the FPGA using the SPI protocol, to obtain the captured frames from the camera using the PPI protocol, to get data in and out from the SDRAM using the EBIU protocol, and to send the encoded frames to the FPGA using the SPORT protocol. Then, the FPGA could pack all four encoded streams and send these to the WiFi module by means of the UART protocol. Finally, the WiFi module could stream the packets to the destination via the UDP protocol.

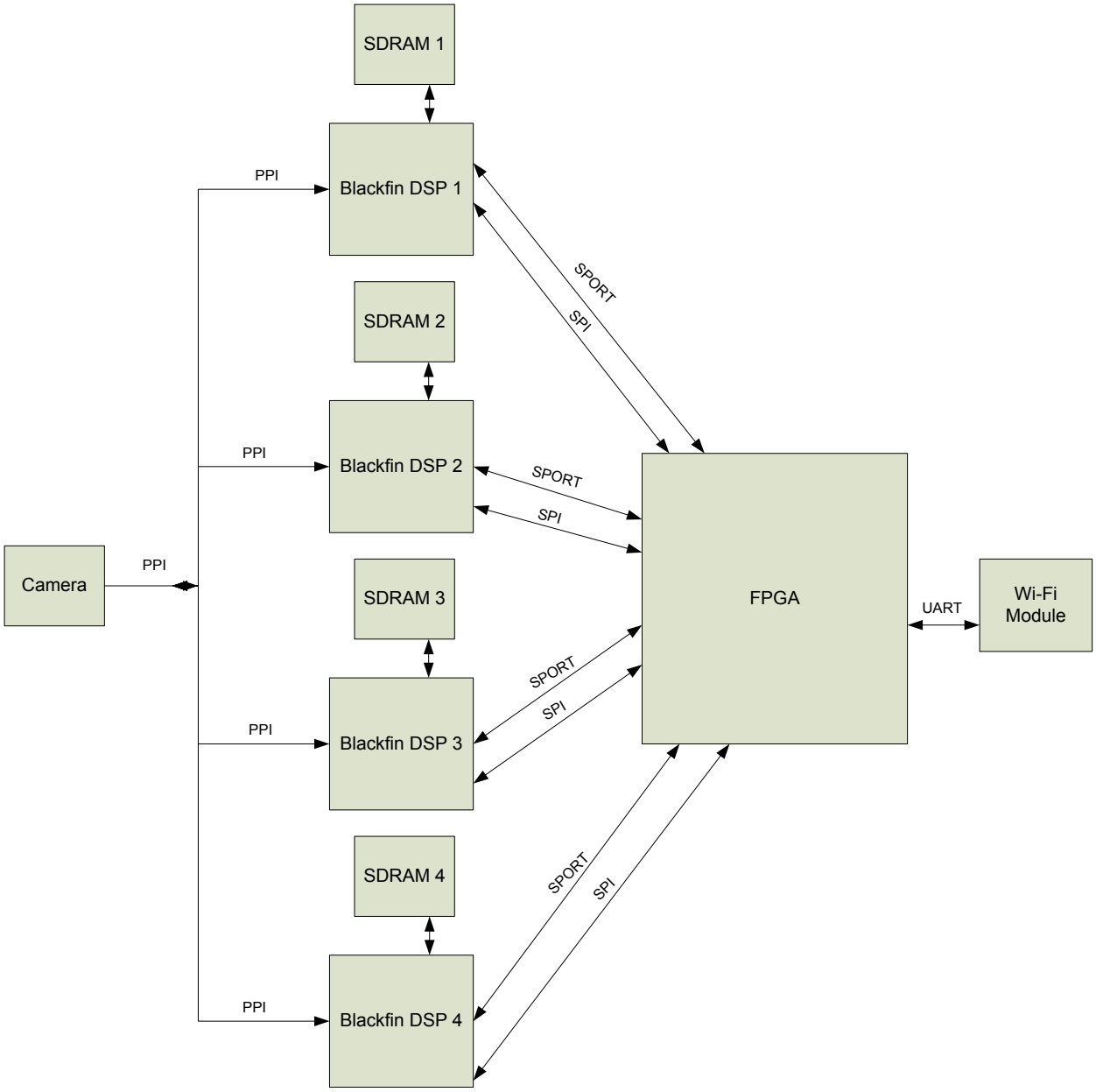


Figure 6.2 : Recommended new hardware structure

The UXGA frame could be divided into four strips, with each strip 1600 pixels wide and 300 pixels high (see Figure 6.3).

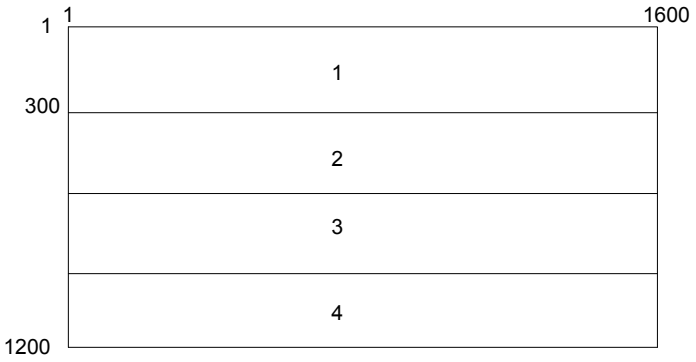


Figure 6.3 : UXGA frame divided to four strips

The UXGA frame could be stored on all four SDRAM, but each DSP can only process one strip. For each DSP, the width could be set to 1600 pixels and the height to 300 pixels in the ADICodecConfigInputParam of the encoder. The start address of strip one would then be sent to the encoder in DSP1, the start address of strip two would be sent to the encoder in DSP2 and so on. The software structure in each of the DSPs would be the same as the structure set out in Section 6.1.2 (see Figure 6.1), and thus the camera cycles and WiFi cycles would be more or less 400 cycles. A test was done by setting the width to 1600 pixels and the height to 300 pixels for the Movie profile in the ADICodecConfigInputParam of the encoder, using the present hardware and software to specify more or less the encoding cycles. For the results, see Table 6.2.

Table 6.2 : Movie profile output using the recommended hardware and software structure

Movie Profile	
Average Encoding Cycles (AEC)	54412538 Cycles
Average WiFi Cycles (AWC)	400 Cycles
Average Camera Cycles (ACC)	400 Cycles
Average Frame Size (AFS)	8328 Bytes
Expected Results:	
Cycles per pixel based on encoding cycles = $\frac{AEC}{(1600 \times 1200)^{(1)}}$	28 Cycles/pel
Encoding time = $\frac{AEC}{CCLK^{(2)}}$	0.143 Seconds
fps based on encoding time only = $\frac{1}{Encoding\ time}$	6.94 fps
Actual time ≈ $\frac{(AEC+AWC+ACC)}{CCLK^{(2)}}$	0.143 Seconds
fps based on actual time = $\frac{1}{Actual\ time}$	6.94 fps
Real Bit Rate ≈ $(AFS \times 8) \times fps\ based\ on\ actual\ time$	463 Kbps

(1): UXGA frame size is 1600*1200.

(2): CCLK = 14*27MHz .

It was found that the new structure would be much faster than the present structure. Encoding time would be only 0.143 seconds, compared to 0.469 seconds in the present system, and the actual frames per second would be 6.94 fps rather than 1.81 fps.

However, there are two concerns that need to be investigated in the proposed new structure. Firstly, the bit rate is almost 12 times the present bit rate. Secondly, it must be assessed how much distortion would occur between the lines that separate the strips and whether this is acceptable or not.

In the proposed new structure, the UXGA frame could be divided into four quarters, rather than strips, each 800 pixels wide and 600 pixels high (see Figure 6.4). However, this would make it very difficult to store them in the right order in the SDRAM, because after the first 800 pixels, 800 pixels would have to be skipped in order to get to the new line.

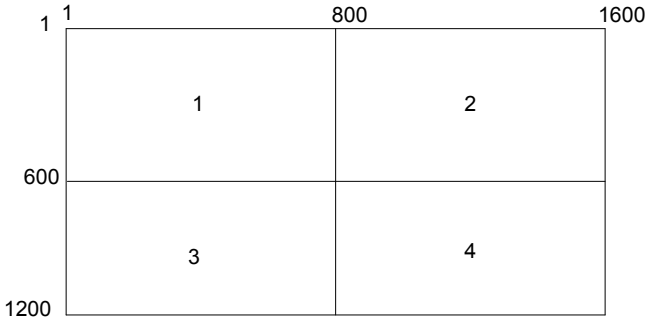


Figure 6.4 : UXGA frame divided to four quarters

6.2 Conclusion

A development board was designed for testing real time video compression encoders that are designed for UAV applications, with the aim of allowing users to test and develop encoders before the actual design of the board that would be used for the UAV.

The thesis began by discussing the background of the main components of the project, namely, camera sensors and video compression. Thereafter, the requirements were identified with regard to the target application, namely, the UAV. Chapter 2 discussed how the project should be implemented, what options were available, and which one would be the best in the current situation.

The design and implementation were examined in Chapter 3. The chapter also listed the rules that were followed during the design of the schematic and the layout. The board mainly consists of a 2 MegaPixel camera, a 400 MHz Blackfin DSP where the video compression encoders would be running, two 8 Mbit flash, one 64 Mbyte SDRAM, an FPGA for future use and a WiFi module that simulates the radio link in the UAV.

The software design and structure were presented in Chapter 4. Some faults, which were discovered during hardware and software debugging, are listed in Section 4.4.

Various sample tests were done on the board by using the H.264 encoder library from Analog Devices; the results of these are discussed in Chapter 5. It was found that the board compressed 2 MegaPixel images at approximately 2 fps, and that it was able to stream out the video at around 186 Kbps. The quality of the video was good but jerky. These tests made it possible to identify the system bottlenecks, which were found to be encoding and transmitting times.

This chapter made various suggestions with regard to possible improvements on the present hardware and software structure. In addition, a new structure was proposed to overcome the bottlenecks, and to speed up the system. Concerns about the new structure, which would need to be investigated, were identified.

7. References:

[1] “Image Sensor”. [Online]. Available:

http://en.wikipedia.org/wiki/Image_sensor#CCD_vs_CMOS

[Accessed in November 2010].

[2] “Canon’s Full Frame CMOS Sensors: The Finest Tools for Digital Photography”. White Paper. [Online]. Available:
http://www.usa.canon.com/uploadedimages/FCK/Image/White%20Papers/Canon_CMOS_WP.pdf

[Accessed in November 2010].

[3] “Charge-Coupled Device (CCD) image sensors”. [Online]. Available:

http://www.kodak.com/global/plugins/acrobat/en/business/ISS/supportdocs/chargeCoupledDevice.pdf

[Accessed in November 2010].

[4] Raymond Westwater and Borko Furht. *Real-Time Video Compression, Techniques and Algorithms*. Kluwer Academic Publishers, 1997.

[5] Iain E. G. Richardson. *H.264 and MPEG-4 Video Compression*. Wiley, 2003.

[6] “How Video Compression Works”. [Online]. Available:

http://www.eetimes.com/design/signal-processing-dsp/4017518/How-video-compression-works

[Accessed in November 2010].

[7] “Setting the Record Straight”. [Online]. Available:

http://www.canopus.com.cn/pdf/yuv_vs_rgb.pdf

[Accessed in November 2010].

[8] “Principles of Video Compression”. [Online]. Available:

discovery.bits-pilani.ac.in/~vimalsp/267MMC/**Lectures**/**Lecture-12**.**ppt**

(Accessed in November 2010).

[9] Al Bovik. *Handbook of Image and Video Processing*. Academic Press, 2000.

[10] “MPEG Video Compression Technique”. [Online]. Available:

http://vsr.informatik.tu-chemnitz.de/~jan/MPEG/HTML/mpeg_tech.html

[Accessed in November 2010].

[11] Steve Richmond. “A Low-Power Design of Motion Estimation Blocks for Low Bit-Rate Wireless Video Communications,” M.S, thesis, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, United States of America, 2001.

[12] Analog Devices. H.264 BP/MP Encoder Developer’s Guide for Blackfin ADSP-BF5xx Processors. Revision 3.0.0, 2009.

[13] Analog Devices. ADSP-BF531/ ADSP-BF532/ ADSP-BF533 Embedded Processor Data Sheet. Revision E, 2007.

[14] Analog Devices. ADSP-BF522/ ADSP-BF523/ ADSP-BF524/ ADSP-BF525/ ADSP-BF526/ ADSP-BF527 Blackfin Embedded Processor Data Sheet. Revision B, 2010.

[15] Analog Devices. ADSP-BF542/ ADSP-BF544/ ADSP-BF547/ ADSP-BF548/ ADSP-BF549 Blackfin Embedded Processor Data Sheet. Revision C, 2010.

[16] Analog Devices. ADSP-BF561 Blackfin Embedded Symmetric Multiprocessor Data Sheet. Revision E, 2009.

[17] Analog Devices. ADSP-BF533 EZ-KIT Lite Evaluation System Manual. Revision 3.1, 2007.

[18] Analog Devices. Engineer-to-Engineer Note EE-68: Analog Devices JTAG Emulation Technical Reference. Revision 10, 2008.

[19] Analog Devices. Engineer-to-Engineer Note EE-228: Switching Regulator Design Considerations for ADSP-BF533 Blackfin Processors. Revision 1, 2005.

[20] STMicroelectronics. PSD4256G6V Data Sheet. Revision 3, 2004.

[21] STMicroelectronics. Application Note AN1153: M88x3Fxx Program Peripheral JTAG Information. January, 2000.

[22] Micron Technology. 512 Mb SDRAM Data Sheet. Revision L, 2000.

[23] OmniVision. OmniVision Serial Camera Control Bus (SCCB) Functional Specification. Revision 2.1, 2003

[24] "I^2C Bus". [Online]. Available:

http://tmd.havit.cz/Papers/I2C.pdf

[Accessed in November 2010].

[25] OmniVision. OV2640 Color CMOS UXGA (2.0 MegaPixel) CameraChip Data Sheet. Revision 2.1, 2006.

[26] ConnectOne. Nano WiReach Data Sheet. Revision 1.26, 2008.

[27] ConnectOne. AT+i Programmer's Manual. Revision 8.32, 2008.

[28] FTDI. UM232R USB-Serial UART Development Module Data Sheet. Revision 1.02, 2006.

[29] Yellow Stone. Surface Mount Chip LED Lamps Data Sheet.

[30] Altera. Section I. Cyclone II Device Family Data Sheet. 2007/2008.

[31] Altera. Serial Configuration Devices (EPCS1, EPCS4, EPCS16, EPCS64 and EPCS128) Data Sheet. Revision 3.3, 2009.

[32] Analog Devices. ADSP-BF533 Blackfin Processor Hardware Reference. Revision 3.3, 2008.

[33] Texas Instruments. PTN78000W 1.5-A, Wide-Input Adjustable Switching Regulator Data Sheet. 2008

[34] Analog Devices. ADP1715/ADP1716 500 mA, Low Dropout, CMOS Linear Regulator Data Sheet. Revision 0, 2006.

[35] Texas Instruments. REG103-5 DMOS 500 mA, Low Dropout Regulator Data Sheet. 2005.

[36] James Edger Buchanan, Bert D. Buchanan. *CMOS/TTL Digital System Design*. McGraw-Hill, 1990.

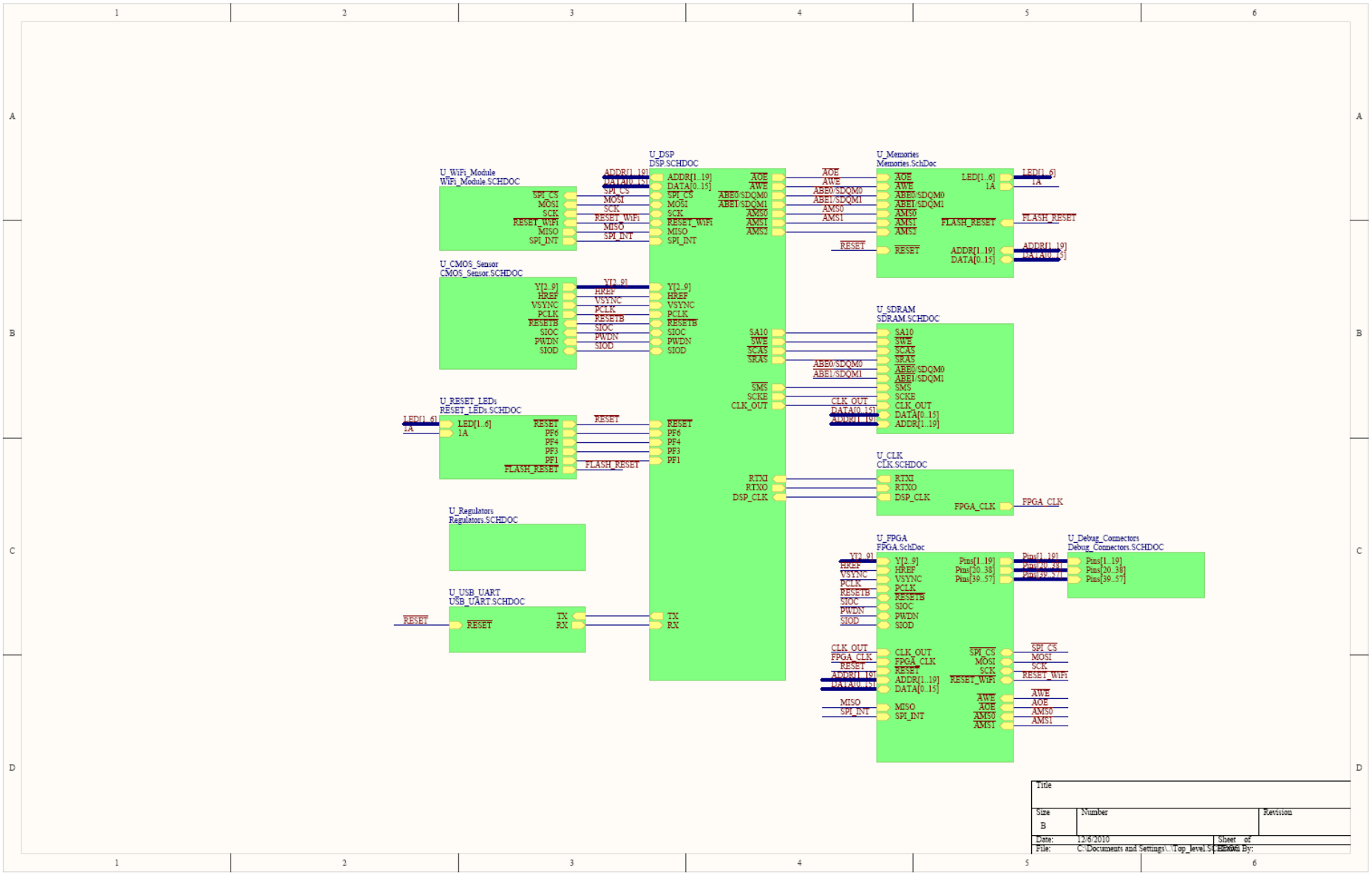
[37] “Fundamentals of SI (Part 2) – Transmission Lines”. [Online]. Available: http://blogs.mentor.com/hyperblog/blog/tag/velocity-of-propagation/ [Accessed in November 2010].

[38] Analog Devices. Engineer-to-Engineer Note EE-281: Hardware Design Checklist for the Blackfin Processors. Revision 2, 2008.

[39] “PCB Track width Calculator”. [Online]. Available: www.**ceda**.in/.../**Track**%20Width%20**calculator**.xls [Accessed in November 2010].

Appendix A: Top Level Schematic Sheet

The top level schematic layout is presented in this appendix. A higher resolution PDF for all the schematic sheets and the original project files are included on the CD (see Appendix E).


U_DSP
DSP.SCHDOC
U_WiFi_Module
WiFi_Module.SCHDOC
U_CMOS_Sensor
CMOS_Sensor.SCHDOC
U_RESET_LEDs
RESET_LEDs.SCHDOC
U_Regulators
Regulators.SCHDOC
U_USB_UART
USB_UART.SCHDOC
U_Memories
Memories.SchDoc
U_SDRAM
SDRAM.SCHDOC
U_CLK
CLK.SCHDOC
U_FPGA
FPGA.SchDoc
U_Debug_Connectors
Debug_Connectors.SCHDOC
Title
Size
B
Number
Revision
Date:
12/6/2010
Sheet of
File:
C:\Documents and Settings\...\Top_level.SchDoc
Drawn By:

Appendix B: PCB Layout

The PCB manufacturing layers are presented in this appendix. A higher resolution PDF of all the PCB layers and the original files are included on the CD (see Appendix E).

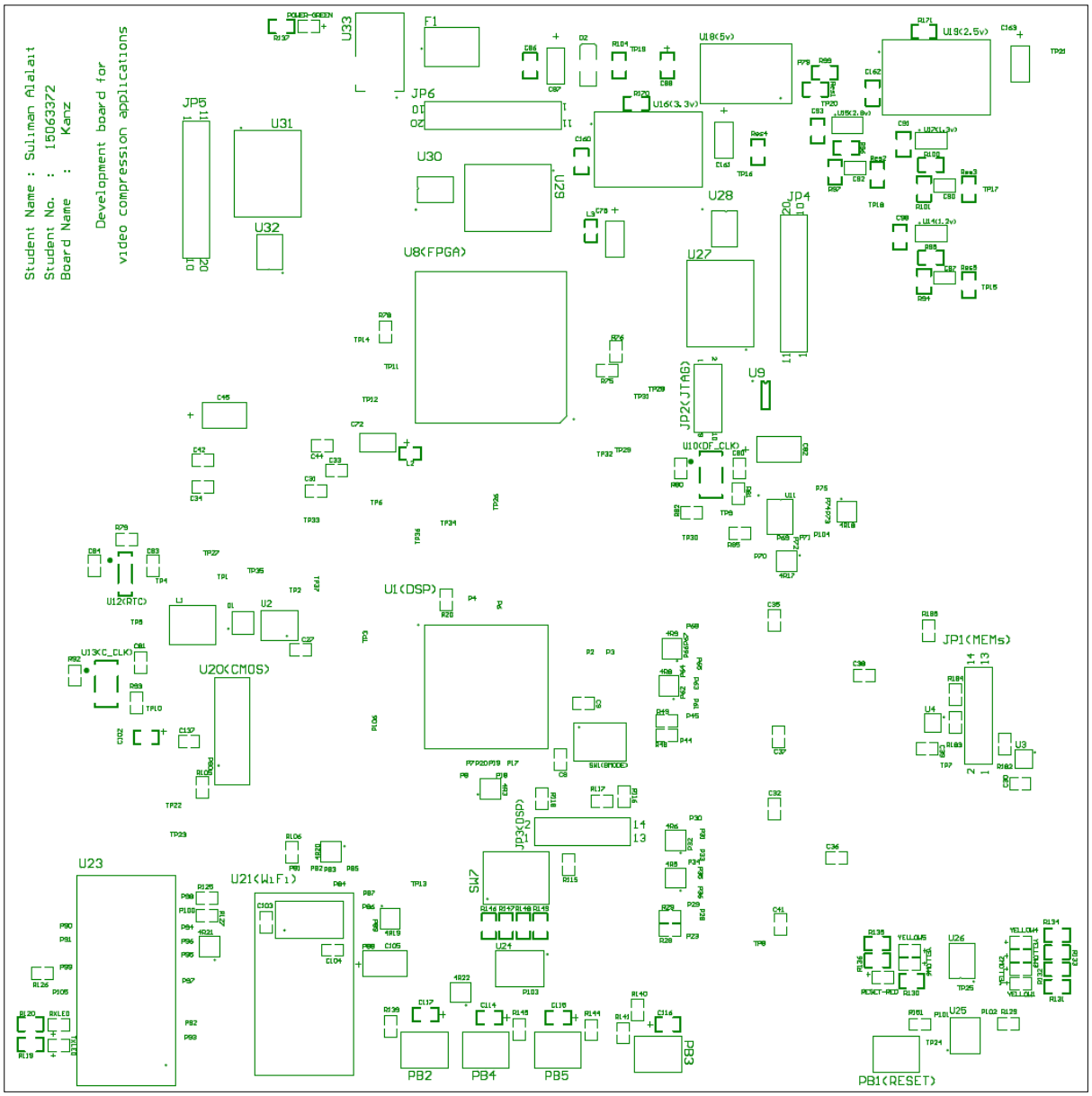


Figure B.1: Top overlay

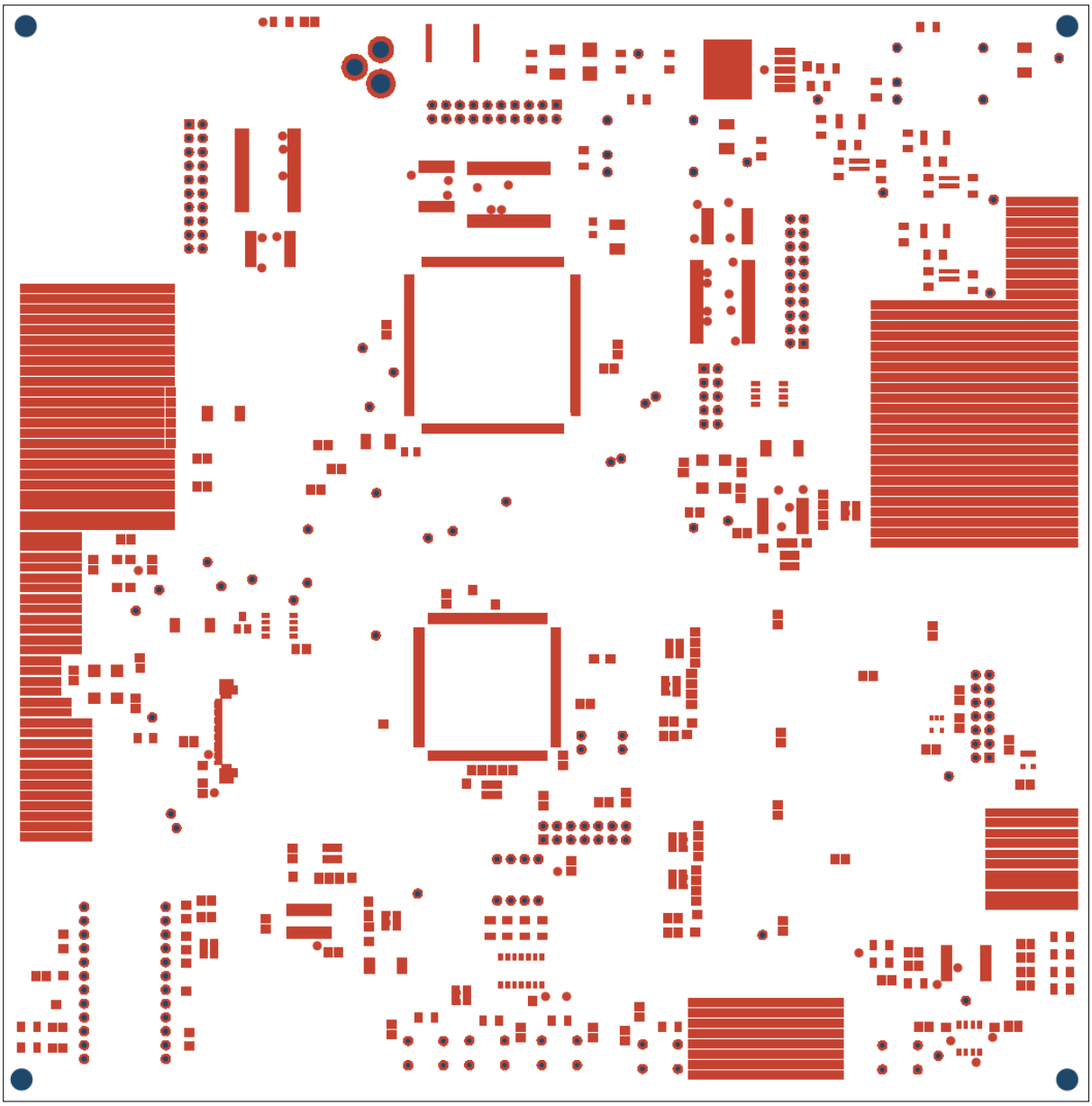


Figure B.2: Top solder mask

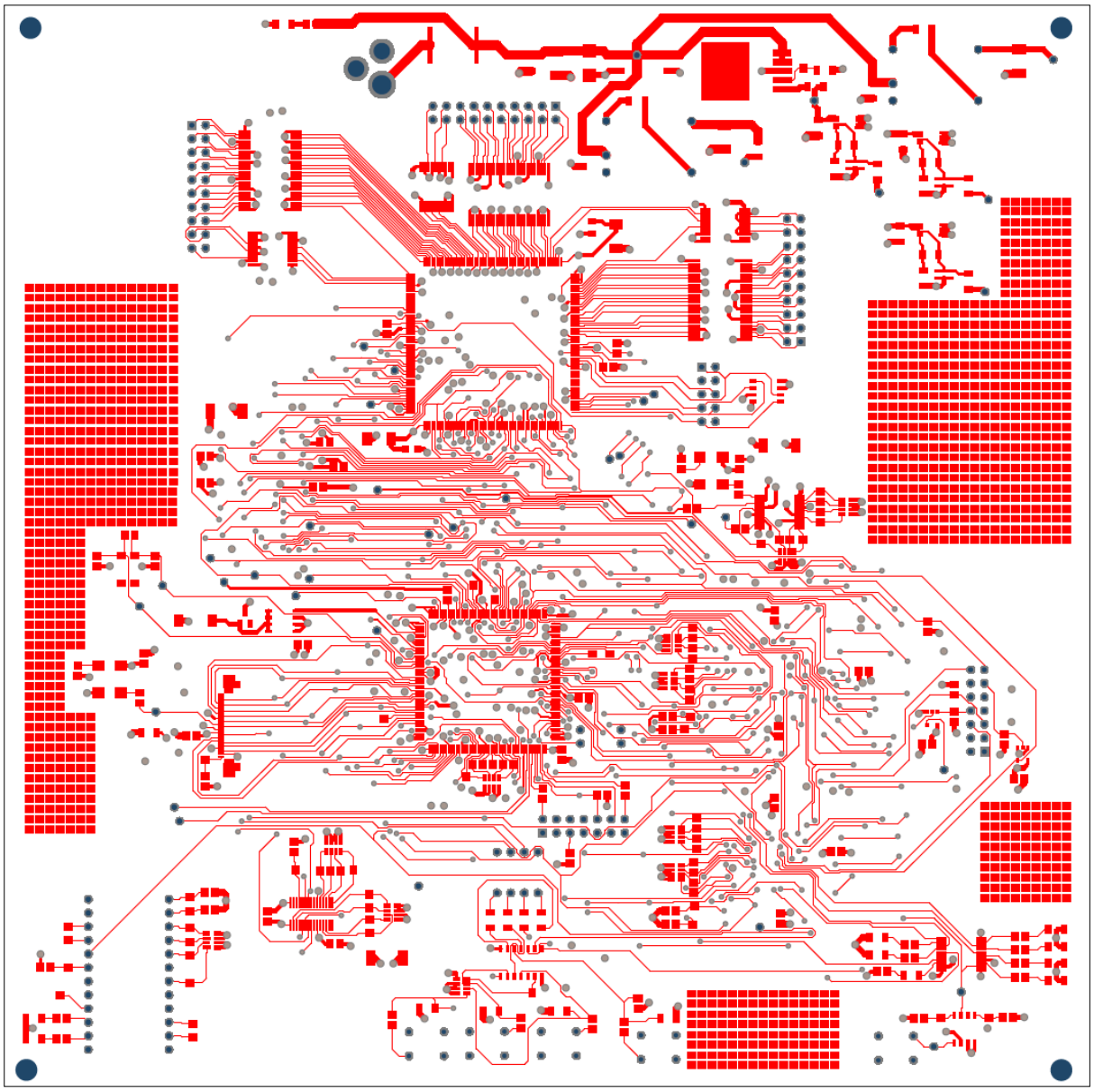


Figure B.3: Top layer routing

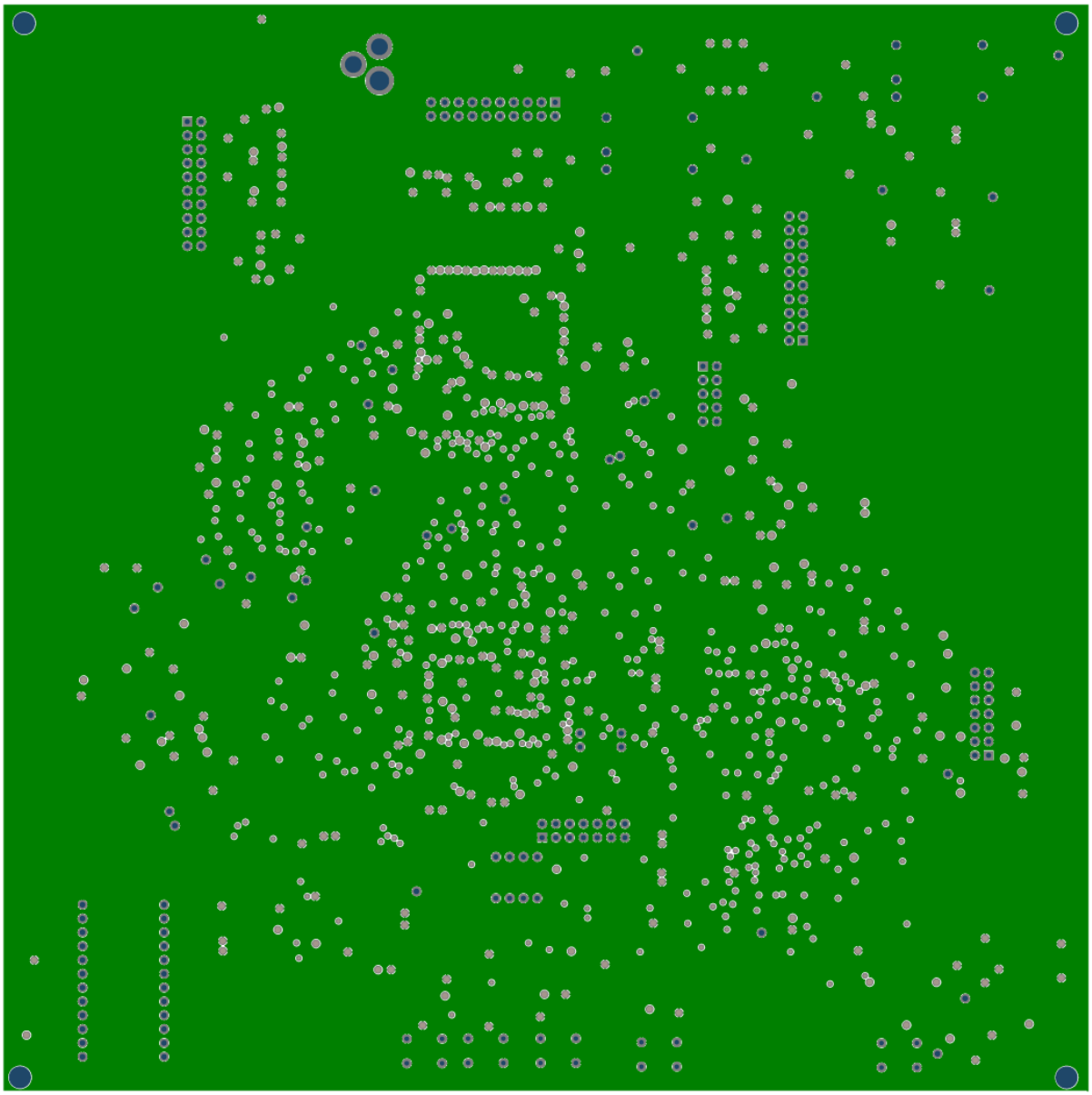


Figure B.4: Ground layer

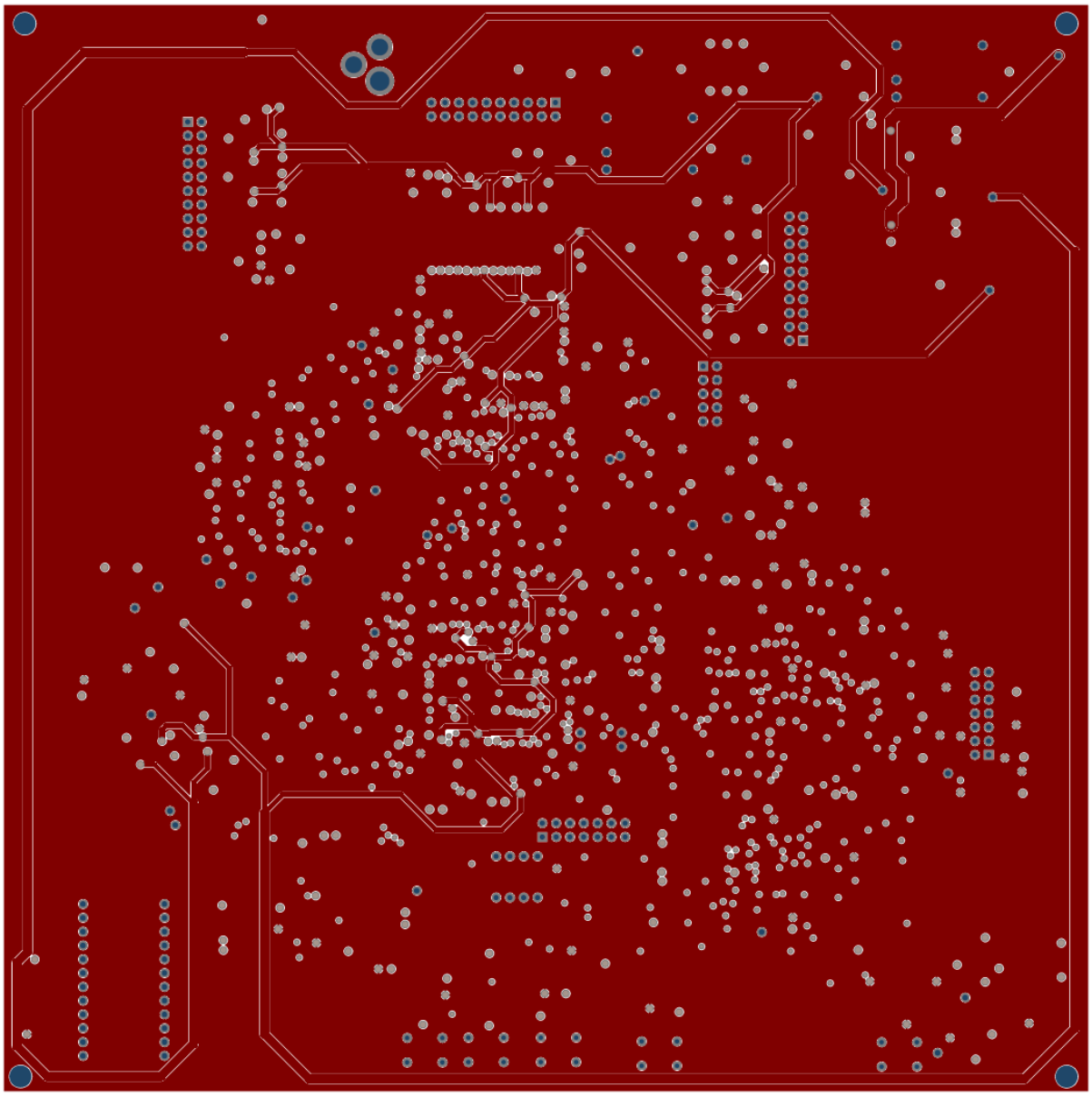


Figure B.5: Power layer

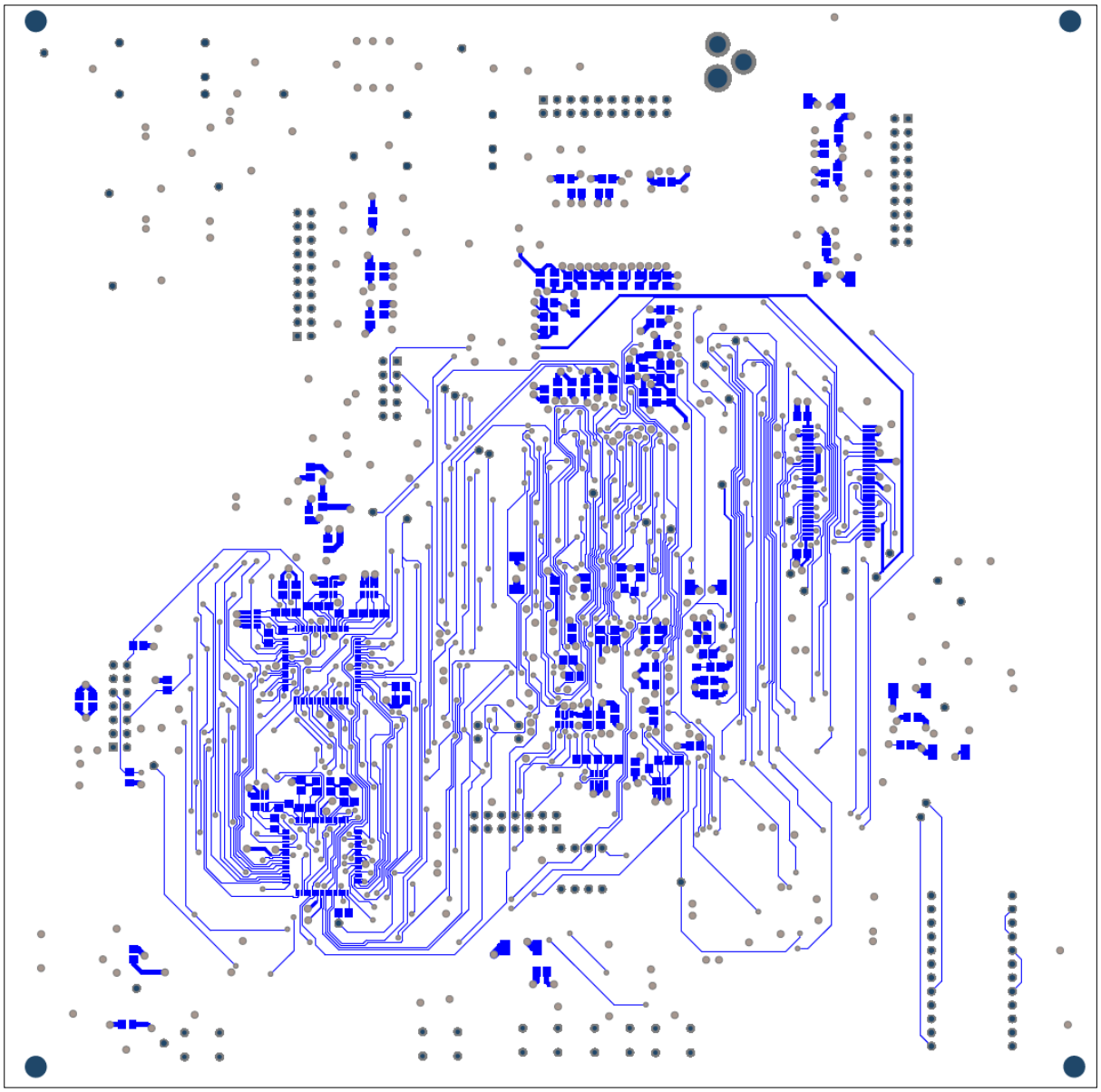


Figure B.6: Bottom layer routing (mirrored)

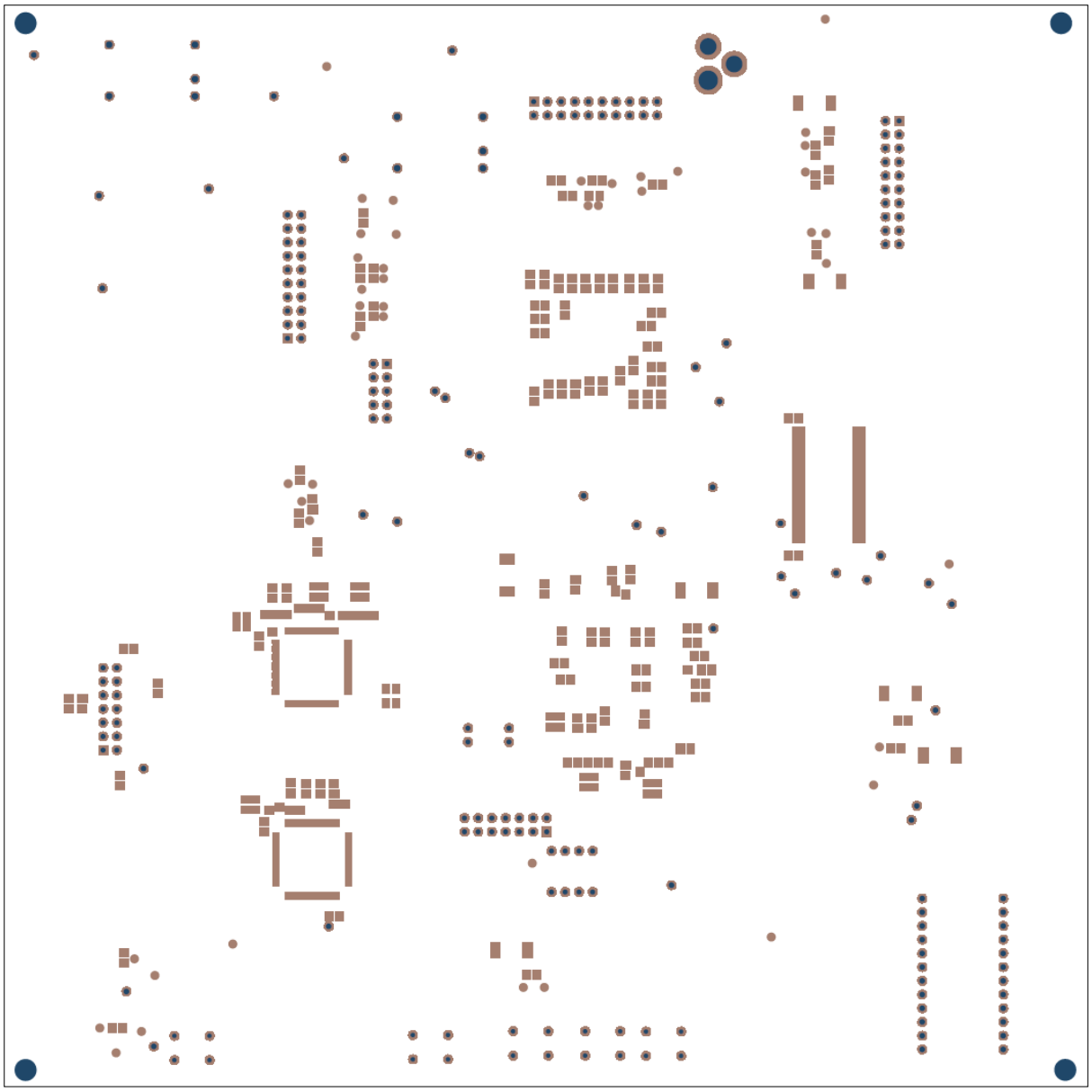


Figure B.7: Bottom solder mask (mirrored)

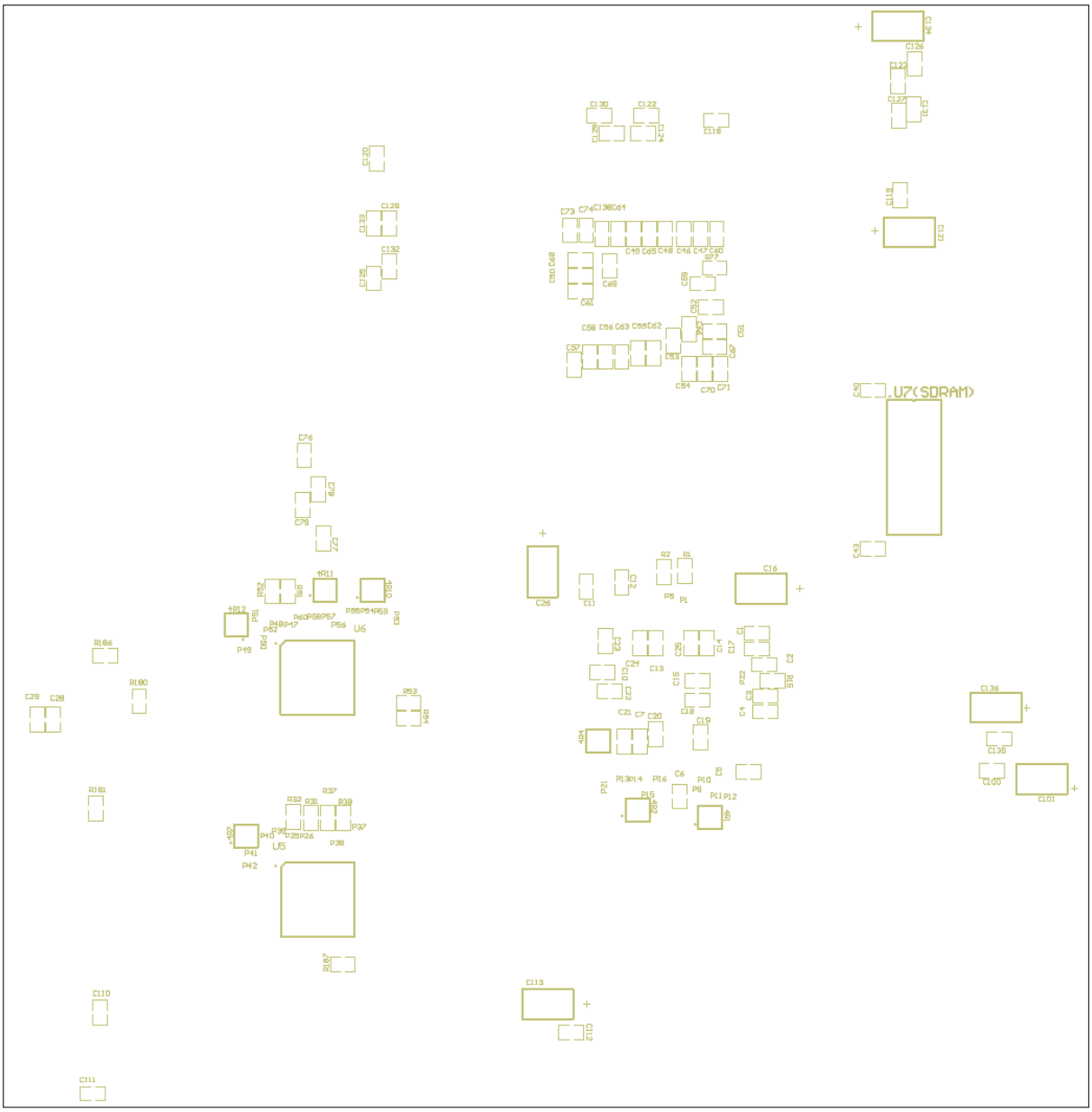


Figure B.8: Bottom overlay (mirrored)

Appendix C: Bill of Materials

#	Designator	Quantity	Company	Description	Value	Part Number
1	4R1-4R12, 4R17-4R22	18	Vishay	Resistor Array(10K)	10 K	ACASA1100A2200P500
2	C1-C15, C18-C25,C27, C28, C30-C44, C76-C81, C100, C103, C104, C110-C112, C118-C120, C122-C133, C135	69	TDK	Ceramic Capacitor	0.1 uF	C2012X7R1H104K
3	C29	1	TDK	Ceramic Capacitor	1 uF	C2012X7R1H105K
4	C16, C26, C45, C82, C101, C105, C113, C121, C134, C136	10	Vishay	Tantalum Capacitor	100 uF	293D107X9016E2T
5	C17	1	TDK	Ceramic Capacitor	DNP	C2012X7R1H104K
5	R54	1	RS	Resistor	DNP	
6	C46-C66, C68, C70, C73, C138	25	TDK	Ceramic Capacitor	10 nF	C2012X7R1H103K
7	C67, C69, C71, C74, C137	5	TDK	Ceramic Capacitor	100 nF	C2012X7R1H104K
8	C72, C75, C87	3	Vishay	Tantalum Capacitor	10 uF	593D106X9016C2TE3
9	C83, C84	2	AVX	Ceramic Capacitor	18 pF	08055A180JAT2A
10	C86	1	AVX	Ceramic Capacitor	1000 pF	12065A102JAT2A
11	C88	1	Vishay	Tantalum Capacitor	0.1 uF	TR3C107K010C0200
12	C90, C92, C97	3	TDK	Ceramic Capacitor	2.2 uF	C1632X5R1A225K
13	C91, C93, C98	3	TDK	Ceramic Capacitor	2.2 uF	C3216X7R1C225K
14	C102	1	Vishay	Tantalum Capacitor	4.7 uF	293D475X9016A2TE3
15	C114-C117	4	AVX	Tantalum Capacitor	1 uF	TAJA105K020R
16	C160, C162	2	TDK	Ceramic Capacitor	2.2 uF	C3216X7R1C225K
17	C161, C163	2	Vishay	Tantalum Capacitor	100 uF	TR3C107K010C0200
18	D1	1	ZETEX	Zener Diode		ZHCS1000
19	D2	1	Vishay	Default Diode		DO-214AA (S2A)
20	F1	1	Mouser	Fuse		SMD250F-2
21	JP1 (MEMs), JP3 (DSP)	2	RS	Header, 7-Pin, Dual row		
22	JP2 (JTAG)	1	RS	Header, 5-Pin, Dual row		
23	JP4, JP5, JP6	3	RS	Header, 10-Pin, Dual row		
24	L1	1	TDK	Inductor	10 uH	SLF7045T-100M1R3-PF
25	L2, L3	2	Vishay	Inductor	10 uH	
26	PB1 (RESET), PB2-PB5	5	Panasonic	Push Button		EVQPAD04M
	POWER-GREEN, RESET-RED, RXLED, TXLED, YELLOW1-YELLOW6	10	Mantech	LEDs		
28	R99, R171	1	RS	Resistor	DNP	
	R1, R2, R15, R28, R29, R31, R32, R37, R38, R48, R49, R51, R52, R75-R78, R80, R92, R106, R125-R127, R129, R140, R151, R183	27	RS	Resistor	10 K	
30	R20	1	Mouser	Resistor	22	

31	R53, R187	2	Mouser	Resistor	0	
32	R79	1	Mouser	Resistor	10 M	
33	R81, R82, R85, R93, R116, R118	6	Mouser	Resistor	33	
34	R94, R96	2	Mouser	Resistor	30 K	
35	R95	1	Mouser	Resistor	15 K	
36	R97, R101	2	Mouser	Resistor	12 K	
37	R100	1	Mouser	Resistor	7.5 K	
38	R104	1	Mouser	Resistor	100 K	
39	R115, R117	2	Mouser	Resistor	4.7 K	
40	R119, R120	2	Mouser	Resistor	390	
41	R130-R136	7	Mouser	Resistor	270	
42	R137	1	Mouser	Resistor	680	
43	R139, R141, R144, R145	4	Mouser	Resistor	100	
44	R146, R147, R148, R149	4	Mouser	Resistor	0	
45	R170	1	Mouser	Resistor	75 K	
47	R180, R181, R182, R184, R185, R186	6	Mouser	Resistor	100 K	
48	Res1-Res5	5	Mouser	Resistor	DNP	
49	SW1 (BMODE)	1	C&K	DIP Switch		SDA02H0BD
50	SW7	1	C&K	DIP Switch		SDA04H0BD
	U1 (DSP)	1	Analog Devices	Blackfin Embedded Processor		ADSP-BF533
	U2	1	Fairchild	P-Channel 2.5V Specified MOSFET		FDS9431A
53	U3	1	TI	INVERTER		SN74LVC1G125
	U4	1	TI	Single 2-Input Positive NAND Gate		SN74AHC1G00
55	U5, U6	2	STMicroelectronics	Flash Memory		PSD4256G6V
56	U7(SDRAM)	1	Micron	SDRAM		MT48LC32M16A2
57	U8(FPGA)	1	Altera	FPGA		EP2C5Q208C7N
58	U9	1	Altera	Serial Configuration Devices		EPCS1
59	U10(DF_CLK), U13(C_CLK)	2	Epson Toyocom	Crystal Oscillator		SG-8002
60	U11	1	IDT	1:10 Clock Driver		IDT74FCT3807
61	U12(RTC)	1	Epson Toyocom	Crystal Unit		MC-156
62	U14(1.2v), U15(2.8v), U17(1.3v)	3	Analog Devices	500 mA Low-Dropout Regulator		'ADP1715 Adjustable
63	U16(3.3v), U19(2.5v)	2	TI	Switching Regulator		PTN78000W-EUS
64	U18(5v)	1	TI	500 mA Low-Dropout Regulator		REG103-5
65	U20(CMOS)	1	Molex	Camera Connector		52437-2471
66	U21(WiFi)	1	Molex	WiFi Connector		52991-0308
67	U23	1	FTDI	USB-Serial UART		UM232R
68	U24	1	Philips Semiconductor	Hex Inverting Schmitt-Trigger		74LVC14A
69	U25	1	Analog Devices	+3 V, Voltage Monitoring uP Supervisory Circuit		ADM708SAR
70	U27, U29, U31	3	IDT	16-Bit Buffer/Line Drive		IDT74FCT162244

71	U28, U30, U32, U26	4	IDT	3.3V Octal Buffer/Line Driver		IDT74FCT3244
72	U33	1	RS	Power Connector		RAPC712

Table C.1: Bill of materials

Appendix D: H.264 Encoder Developer's Guide

The parameter flags presented in this appendix are extracted from the H.264 BP/MP encoder developer's guide for Blackfin ADSP-BF5xx processors document [12].

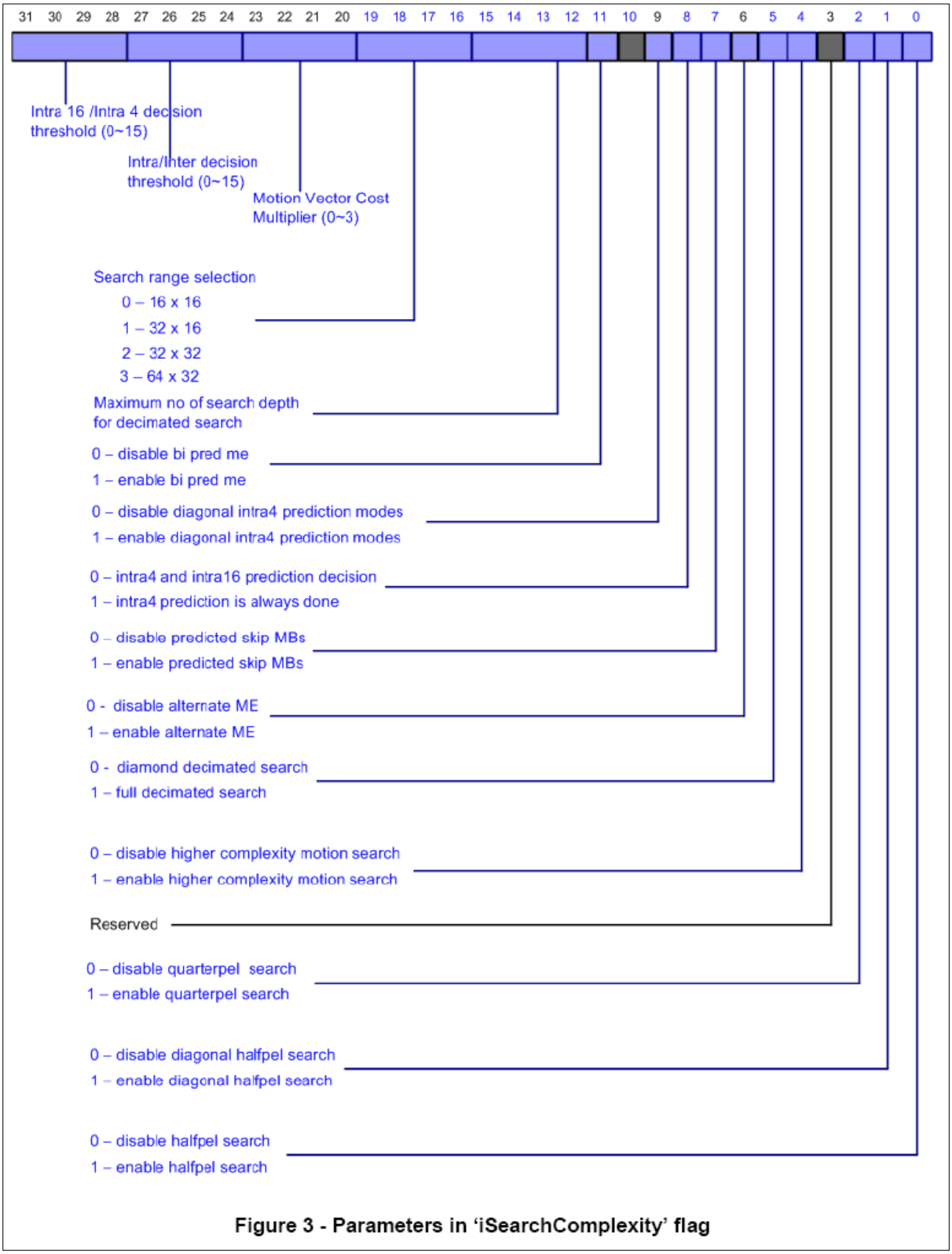


Figure 3 - Parameters in 'iSearchComplexity' flag

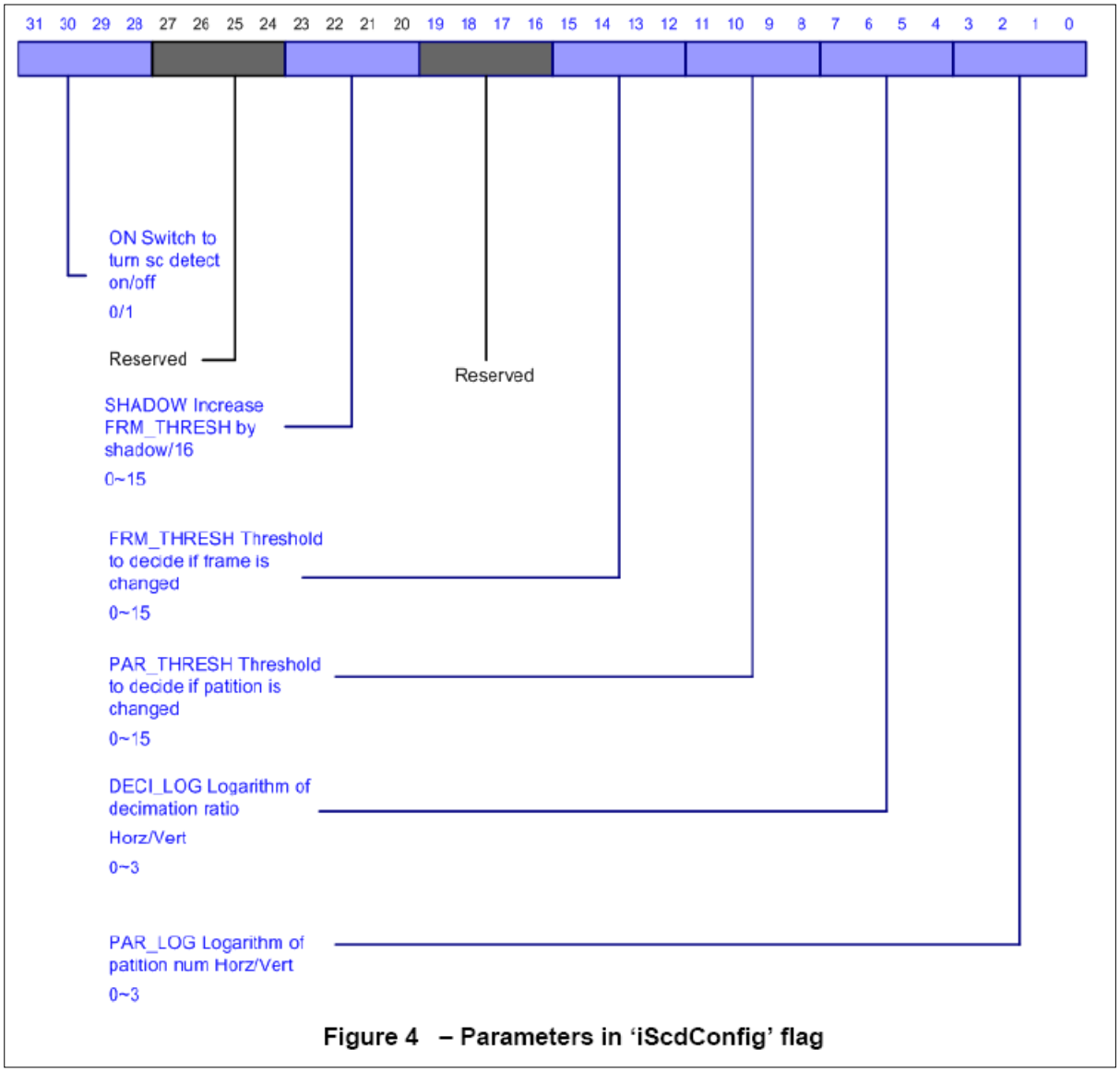


Figure 4 – Parameters in 'iScdConfig' flag

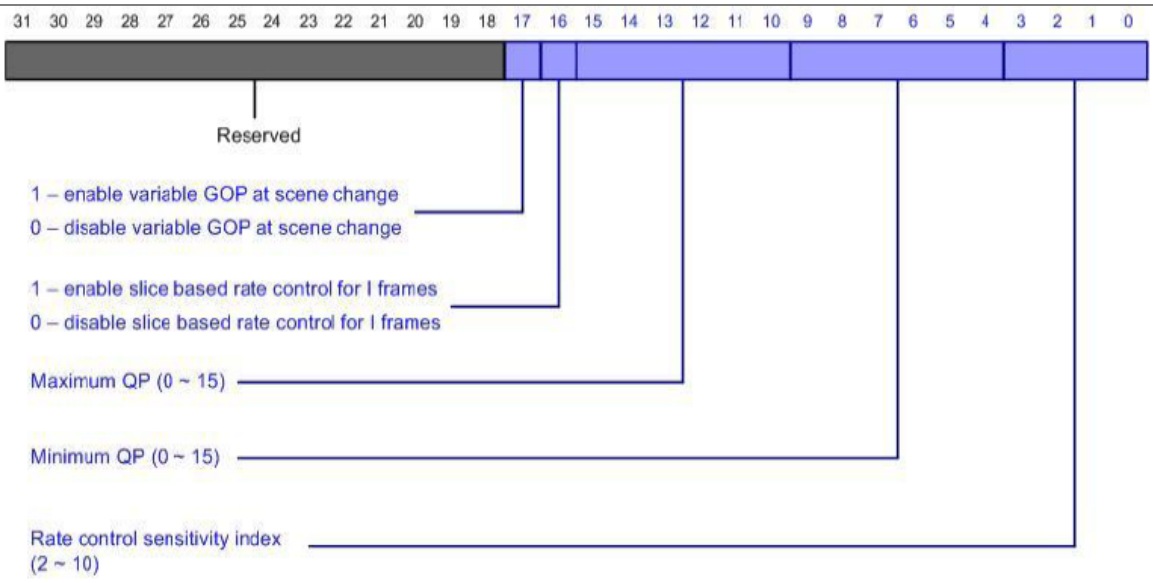


Figure 5 – Parameters in 'iRcConfig' flag

Appendix E: CD Manual

The root directory of the CD contains the H264 Encoder Developer Guide document, bill of materials in an Excel sheet format and the following folders:

- Analog Devices BF533 DSP
- Board Schematics & PCB Altium Files
- Camera Module
- Data Sheets
- Schematics and PCB Layers in PDF Format
- WiFi Module
- Source Code

The Analog Devices BF533 DSP folder contains the BF533 data sheet, hardware reference manual and evaluation kit manual files as well as the EE-68, EE-228 and EE281 application notes.

The Board Schematics & PCB Altium Files folder contains all the schematic libraries, PCB libraries, schematic sheets and the PCB layout that which were created using Altium Designer Winter 2009. The project file name is *VideoCompression_V3.PrjPcb*.

The Camera Module folder contains the OV2640 data sheet and the SCCB document.

The Data Sheets folder contains all the components data sheets that were used in this project.

The Schematics and PCB Layers in PDF Format folder contains all the schematic sheets and PCB layers in high resolution in PDF format.

The WiFi Module folder contains the Nano WiReach data sheet and AT+i programmers manual document.

The Source Code folder contains the project C and H source code, ME and MC VHDL source code and the PSD4256G6V configuration files.

E.1: C and H Source Code

The project C and H source code resides in the KANZ folder under the Source Code folder. The project file name is *KANZ.dpj*, (see Figure E.1).

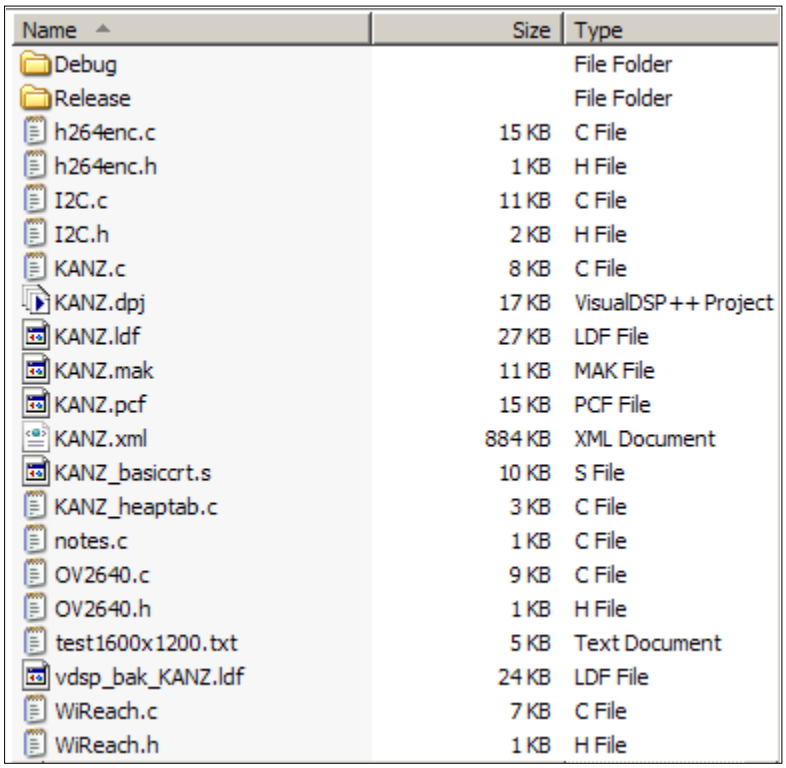


Figure E.1: KANZ subdirectory

E.2: ME_MC VHDL Source Code

The ME and MC VHDL source code resides in the hdl subfolder of the ME_MC folder, all under the Source Code folder. The project file name is *ME_MC.prj*, (see Figure E.2).

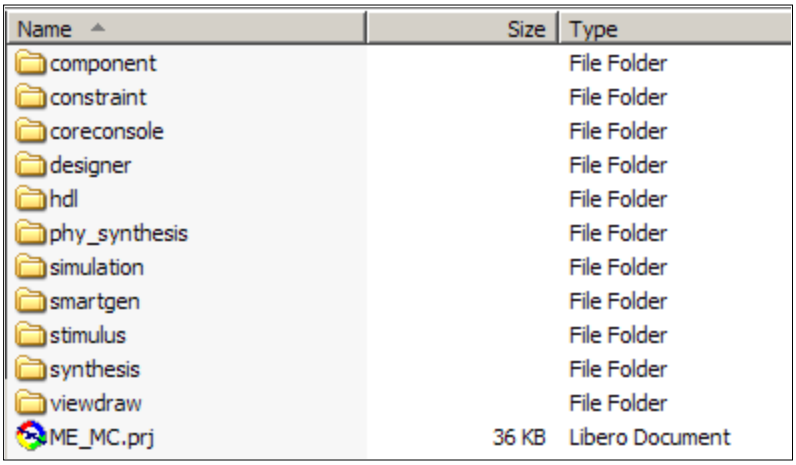


Figure E.2: ME_MC subdirectory.

E.3: PSD4256G6V Configuration Files

The *flash_a.obj* and *flash_b.obj* files that were loaded onto the PSDs reside in Flash_A and Flash_B folders respectively under the PSD4256G_ConfigFiles folder, all under the Source Code folder.