A SCALABLE IMAGE/VIDEO PROCESSING PLATFORM WITH APPROXIMATE FPGA DESIGN

by

Yunxiang, Zhang, B.S.

THESIS

Presented to the Faculty of

The University of Houston-Clear Lake

In Partial Fulfillment

Of the Requirements

For the Degree

MASTER OF SCIENCE

in Computer Engineering

THE UNIVERSITY OF HOUSTON-CLEAR LAKE

DECEMBER, 2018

A SCALABLE IMAGE/VIDEO PROCESSING PLATFORM WITH APPROXIMATE DESIGN

by

Yunxiang, Zhang, B.S.

APPROVED BY

Xiaokun Yang, PhD, Chair

Lei Wu, PhD, Committee Member

Jiang Lu, PhD, Committee Member

APPROVED/RECEIVED BY THE COLLEGE OF SCIENCE AND ENGINEERING:

Said Bettayeb, PhD, Associate Dean

Ju H. Kim, PhD, Dean

Dedication

I dedicate this dissertation to my friends and my parents. Without their encourage, understanding, and most of all love, the completion of this work would not have been possible.

Acknowledgments

First and foremost, I am grateful to my major advisor, Dr. Xiaokun Yang, for being friendly, caring, supportive, and help in numerous ways. Without his support, I could not have done what I was able to do. He was very generous in sharing his experiences on electrical and computer engineering, academic life and beyond. He is not only my adviser, but also, a friend inspiring me for the rest of my life.

Next, I would like to thank the members of my committee, Dr. Jiang Lu and Dr. Lei Wu for their support and suggestions in improving the quality of this dissertation. It is truly honored to have such great fantastic and knowledgeable professors serving as my committee members.

I would also like to thank all the lab mates and members at the Advance Digital System Design (ADSD) Laboratory for creating an amazing working environment, and thank my friends, Archit Gajjar and Cui Xue, for their assistance on work related to my research.

Furthermore, I would also like to acknowledge the research support provided from the Department of Computer Engineering at University of Houston-Clear Lake, and the dissertation year fellowship from the graduate school during my dissertation research.

Finally, I want to thank my family for their unconditional love, faith, and encouragement.

ABSTRACT

A SCALABLE IMAGE/VIDEO PROCESSING PLATFORM WITH

Yunxiang, Zhang, B.S.

University of Houston-Clear Lake, 2018

Thesis Chair: Xiaokun Yang, PhD

This dissertation presents a scalable image/video platform with approximate computing design on Field-Programmable Gate Array (FPGA). The platform is able to capture images in real time with a low-cost OV7670 camera and display the original, in-process and final results of images on a VGA-interfaced monitor. To make the platform reusable and expandable, the design with Verilog Hardware Description Language (HDL) and the verification environment including six Open Verification Components (OVCs) are provided. Compared to prior works, our proposed work achieves the least FPGA resource cost (753 Look Up Tables (LUTs) and 277 Registers) on the design of a Camera-FPGA-VGA platform. Furthermore, we present a novel approximate design library with FPGA and provide several slice-energy cost solutions corresponding to different application constrains. Specifically three approximations of multipliers and two approximations of adders, along with the exact designs,

are presented and integrated as twelve benchmarks to implement RGB to grayscale conversion as a case study. Experimental results show that the minimum slice-energy cost, integrated with approximate#2 adder and approximate#3 multiplier, achieves 25.17% slice-energy saving compared with the exact design by sacrificing the quality of results as 5.69% error for multiplier and 2.85% for adder.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Scalable Image/Video Processing Platform

To date, computer vision applications are growing rapidly, bringing many challenges of computation speed and power consumption on traditional software based frameworks such as object and facial recognition [25], [12]. As a result of the advantages such as programmability and parallel computing on pure hardware design, the implementation on Field-Programmable Gate Array (FPGA) is becoming widely used in many applications of image/video processing [9].

In prior works, such designs on FPGA were mostly intended on high-level synthesis (HLS) design and sometimes involved with software and GPU. For example, Ref. [16] proposed a real-time image acquisition design by using LabVIEW with GPU-based acceleration which is able to sustain the rate of data acquisition. Similarly, Ref. [28] presented an implementation of a camera with LabVIEW frame grabber, a mask generating by MatLab, and an image processing design on LabVIEW FPGA. This implementation is able to prove a system easily by directly using the block based design, but it is hard to customize or improve the system because the block libraries are usually not open source to users.

Under this context, many researchers have proposed their works of register-transfer level (RTL) designs on image/video processing. For example, Mike Field's OV7670-FPGA-VGA project [8] has been widely reused and expanded into several prototypes on new research ideas related to the image/video processing systems. This open source code is written by VHDL and performed on Zedboard FPGA. Moreover, a reconfigurable platform preforming edge detection by interfacing an OV7610 camera as an image input and VGA as the result output, has been presented in Ref. [2]. Since

the FPGA resource cost and power consumption have not been provided in these two works, it is not able to compare and estimate the design performance.

Ref. [1] proposed an open source project by using an OV7670 camera on FPGA. This project is writen by using VHDL and implemented on an Altera DE2-115 FPGA board. It consists of 1,616 logic elements and 818 registers. The main concern of this project is the lack of a verification environment, making the project being difficult to be reused and expanded. To avoid the problems aforementioned, in this thesis we proposed a Camera-FPGA-VGA data path on a Nexys-4 FPGA board named as 'A Scalable Image/Video Processing Platform' including both the design and verification environment. By interfacing a low-cost OV7670 camera with the image/video input on the Nexys-4 board, it is able to display the original and processed images on a VGA-interfaced monitor. Our contributions from this thesis include:

- We presented not only Design-Under-Test (DUT), but also a verification environment containing tcl script, filelist, and testbench with six Open Verification Models (OVCs) such as Bus Function Models (BFMs) and scoreboards [39], in order to make the design reusable and expandable. The implementation is writen by Verilog Hardware Description Language (HDL) and preformed on a Nexys 4 FPGA board, and the verification environment is able to be run automatically on ModelSim.
- We presented a verification environment in order to automatically check the functions of the System on Chip (SoC), by providing three BFMs and three scoreboards. A tcl script is also provided to control the scoreboards to compare data between DUT and golden models, aiming to increase the re-usability and reliability of the open source designs.

- We presented a 640 × 480 resolution VGA display with showing up to four 320 × 240 resolution images at the same time. In such a way the original images, the in-process images, and the final results of the images are able to be shown in the same window.
- We presented design performance in terms of slice count and power consumption. Compared with prior works, our proposed work archive the least FPGA resource cost - 753 slices Look-up-tables (LUTs) and 277 slices as register. And the power consumption is around 220 mW for displaying and processing a single frame of color images.

1.2 Approximate Design on Combinational Circuits

In the design of energy-efficient digital systems, approximate computing is considered as a possible solution in many application domains. Some of the operations used in this area are intrinsically error-tolerant, such as multimedia, recognition, and data mining [15], [31], [11], [43], and [30]. For these kinds of applications, approximate computing is served as an important part to reduce the design area, power consumption, and computation delay in digital systems. This is a tradeoff between accuracy and performance, in order to sacrificing accuracy to gain better performance in energy efficiency.

Recently IBM [24], Intel [23], Microsoft [4], [3], [6], and a lot of companies and research groups [29], [5], [22], and [27] proposed some effective results on approximate computing. However, most of them are limited to the software applications. Due to the advantages of the pure hardware implementations, such as the reconfigurability and hardware parallelism, we believe that FPGAs will be adopted in the future or the next-generation of high-performance SoC. For example, adders have been commonly

considered for the approximate implementation as one of the important components in the circuit design [38]. Some of the approximate adder has been discussed in [19] and [21]. These works focused on the subcomponent designs, however, the impact of the approximations on structural implementations has not been considered. In most of the applications, the improvement on system-level has a greater potential to improve the circuit and system performance [34], [33]. Ref. [20] designed the approximate adder to produce the radix-8 booth encoding $3\times$ with error reduction. Ref. [18] used 2×2 approximate multiplier blocks to compute the final results. And Ref. [14] had introduced approximate speculative adders used in a multiplier. In this thesis, thus, we propose a set of approximate FPGA design components, in order to find a better balance between accuracy and power cost, by providing a wide rang of solutions for different energy-quality tradeoffs corresponding to different applications.

One of the big challenges of this integrated system is the hardware programmability on FPGA. Basically, FPGA development needs to balance resource cost with algorithm accuracy and extensive hand-coding in RTL. To fill the gap between software programming and hardware design and provide more RTL choices for different project requirements, we presented a basic FPGA design library including adders and multipliers with five different accuracy levels of components: exact design (EX), approximate design #1 (AP1), approximate design #2 (AP2), approximate design #3 (AP3) and approximate design #4 (AP4). More specifically, the main contributions of this thesis are:

- We presented five approximation degrees of adders and multipliers and estimated the average/maximum error distances with considering all the possible test cases.

- We evaluated the tradeoffs between design accuracy and resource cost, by which users are able to choose computation components depending on different applications and requirement.
- We employed different approximate components to implement applications, for both algorithm and RTL designs.
- We evaluated the real hardware performance including slice count, maximum operational frequency, and power consumption.

1.3 Advance Approximate application on Sequential Circuit

In order to reduce the computational cost and improve the energy efficiency, approximate design on FPGA platforms has been widely used in many application domains, such as artificial intelligent [13], edge computing [10, 32, 36], and Internet-of-Things (IoT) security [41], [37].

The inaccurate implementation potentially provides an opportunity to find the minimum FPGA cost in terms of slice number and energy consumption corresponding to different quality of constrains. The main challenges are: 1) the combinational circuit design on FPGA is mapped to look-up-tables (LUTs), leading to uncertainty to determine the power consumption and the maximum operational frequency (MOF); 2) different from the improvement on one sub-components, the combination of multiple approximations of computational components affects the energy cost and slice utilization, making the tradeoff between accuracy and slice-energy reduction difficult.

To find the tradeoff between quality of the results and energy cost on FPGA is one of the important points driving the research of approximate computing. Previous works in such field have mainly focused on combinational circuit design [17], and some of the researchers concentrated on optimizing the flow chart to reduce the energy con-

sumption [42]. In this thesis we provided a design of a sequential circuit with twelve approximations, providing a wide range of quality-cost tradeoffs. Additionally the earlier researches on improving FPGA design focused on either the low-energy technology [7], [40] or the low-cost architecture [33]. However, most embedded chips are operating in cost-limited and energy-constrained environments such as energy harvesting powered platforms and micro-controllers, in which the increasing of resource cost and power consumption will villainously shorten the lifetime of the systems. To overcome this issue, Ref. [44] proposed four approximations of addition models and evaluated the quality of results on a histogram equalization algorithm. The idea was simulated on Matlab so the hardware performance was not estimated. Therefore, our work focuses on proposing several approximate multipliers and employing the approximate adders as well, and more important, the slice-energy savings are estimated and demonstrated on an FPGA platform.

Under this background, this thesis paper proposes several approximations of adders and multipliers, and further applies all the components on a sequential circuit design of color to grayscale converter. More specifically, the main contributions are:

- We presented a fixed-point design on RGB to Grayscale converter (RGB2Grayscale) with a artix-7 FPGA platform. Then three approximations of fixed-point multipliers, and two approximations of adders are proposed and applied in this prototype. The tradeoff between quality of results and resource cost is statically analyzed with different implementations.
- We implemented twelve RGB2Grayscale converters with different approximations of RTL design, and synthesized the DUT with a Nexys-4-artix-7 . The performance in terms of slice count and dynamic energy consumption were estimated using the performance evaluation methodology [35].

- We evaluated the cost saving on weighted slice count and energy dissipation using a slice-energy metric in our work. Experimental results show that the minimum slice-energy reduction can reach 25.17% compared with the exact design.

1.4 Structure Of The Dissertation

The rest of this dissertation is organized as follows. The chapter 2 presents a scalable image/video platform with approximate computing design on Field-Programmable Gate Array (FPGA). The chapter 3 presents a novel hierarchical synthesis for approximating FPGA components library. The section 3.6 discusses the advance approximate application that providing several slice-energy cost solutions corresponding to different application constrains. Finally, in Chapter 4, we conclude this dissertation and discuss possible future work.

CHAPTER 2

A SCALABLE IMAGE/VIDEO PROCESSING PLATFORM

2.1 Proposed Work

The design architecture of the image/video processing platform is discussed in this section. In addition, the verification environment of this platform is presented to make the open source platform expandable and reusable to other researchers.

2.2 Design Architecture

A scalable image/video processing FPGA platform proposed in this section is shown in Fig. 2.2. The platform is able to capture image/video by using a low-cost camera, process the images, and display them to a VGA-interfaced monitor in real time. This platform includes an OV7670 camera, a Nexys-4 FPGA board, and a monitor with VGA port. The FPGA platform is designed with Verilog HDL, including three sub modules: I2C Controller, Image Capture and VGA Master. The I2C Controller is the control module of the OV7670 camera, by using the I2C protocol to set functional registers. After being configured the camera enables to capture and send images pixel by pixel through the 'VSYNC-HREF-DATA' interface.

The Image Capture module receives and stores the image data into four memory blocks, named the 'Frame Buffer'; each block can store one 320 $\times$ 240 resolution image. Then, the VGA output module reads the data from buffers and sends them to a VGA-interfaced monitor. In this platform, the full screen monitor (640 $\times$ 480 resolution) is splited into four regions (320 $\times$ 240 resolution) as shown in Fig. 2.1. The 'Region0' is used to display the data from 'FrameBuffer0'. 'Region1', 'Region2', and 'Region3' are applied to display the data processed by three different algorithms:

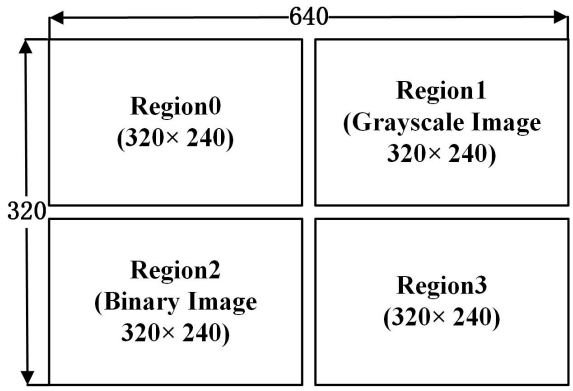


Figure 2.1: *640 × 480 Window with Four 320 × 240 Regions*

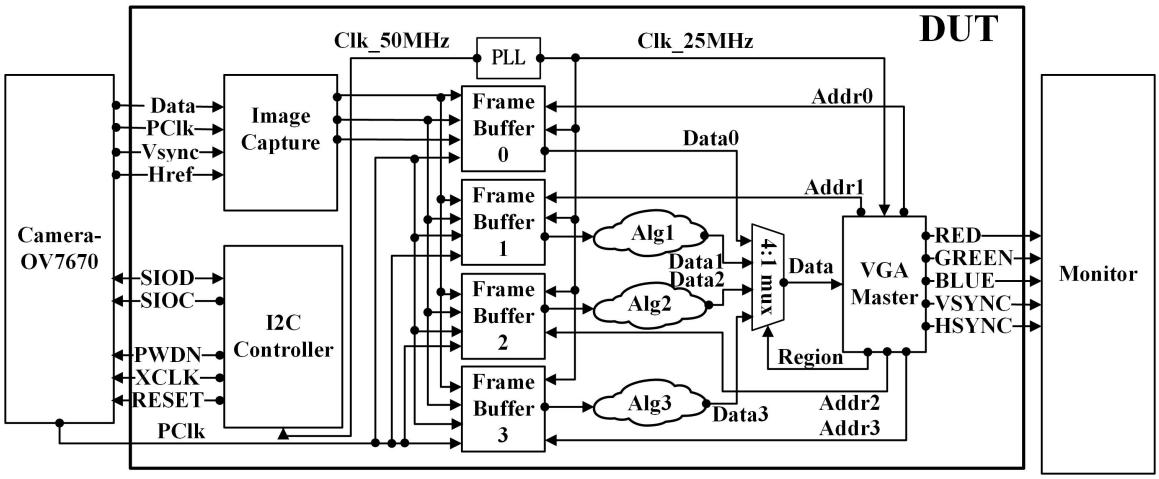


Figure 2.2: *Design Architecture*

'Alg1', 'Alg2', and 'Alg3'. Notice that multiple clock cycles or memory blocks might be needed based on the complexity of the algorithms.

2.3 Verification Environment

The open source packet shown in Fig. 2.3 includes a synthesizable design with Verilog HDL, a configurable verification environment, and FPGA synthesis files. In particular, the green box shows the design tree with three levels, from top datapath to bottom submodules. The orange box represents the synthesis files for FPGA

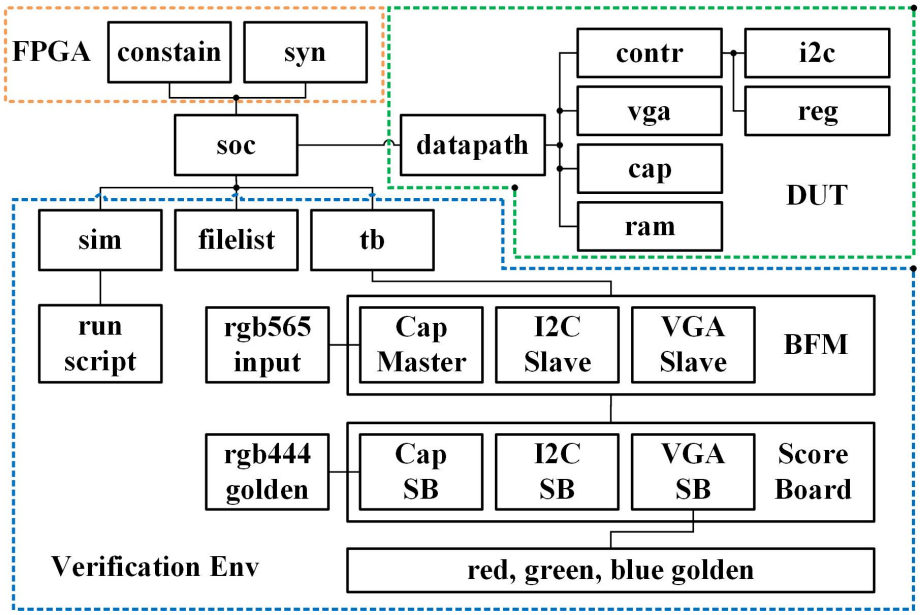


Figure 2.3: *FPGA Design, Verification, and Synthesis*

implementation, including a constrain file and a FPGA netlist. Notice that the synthesis files are also able to be used to generate performance results in terms of slice count and power cost. The most important contributions of this section is the verification environment shawn in the blue box. The verification environment contains a tcl script, a file list, and a testbench. The tcl script is created to configure the design and verification models working in different modes.

The testbench provides three BFMs-Capture Master, I2C Slave, VGA Slave, and three Scoreboards (SB)-Capture SB, I2C SB, VGA SB, as OVCs. For instance, the DUT receives images from the capture master BFM. The input image data stored in 'rgb565_input.txt' file is in the RGB565 format-5-bit red pixel, 6-bit green pixel, and 5-bit blue pixel. The BFM is able to tranfer 8 bits of the data in each clock cycle. Therefore, a 16-bit pixel requires two clock cycles to finish the transmission. Similarly, the I2C slave BFM is designed to receive and response to commands from

the I2C master. The VGA slave is created to collect the images data from the VGA interface.

Since there are three BFMs, then there are three SBs paired with. For example, the VGA scoreboard compares the red, green, and blue pixels driven by VGA master with the golden data in 'golden_r.txt', 'golden_g.txt', and 'golden_b.txt' files. Likewise, the I2C scoreboard compares each command received by the I2C slave BFM with the original register configuration. The capture scoreboard verifies the data stored into memory blocks with the golden data from 'golden_rgb444.txt' file. All the DUT are discussed in the section. 2.4.

2.4 Design Under Test(DUT)

All the design submodules and Intellectual Properties (IPs) are introduced in this section, including the I2C Master, the Image Capture slave, the VGA Master, the clock PLL, and Frame Buffers.

2.5 OV7670 Controller

The design of the OV7670 Controller is shown in Fig. 2.5. The most important module of this controller is the OV7670 register and the I2C sender. All register setting values for OV7670 camera are generated in the module OV7670 register. These register values are sent by the module I2C sender. I2C sender generates the clock line (SIOC) and the data line (SIOD). These data lines follow the I2C-interface. When the SIOD signal is being pulled low and the SIOC signal continues being high the camera initiates data transfer. After the SIOC signal pulled low, the SIOD signal begins to send the first data bit. The camera receives the first data bit when the SIOC is pulled high again. This process repeats until the camera receives a stop signal. The stop

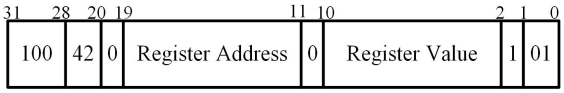


Figure 2.4: *Signal 'data_sr'*

signal is received when the SIOC signal is pulled high, and is followed by the SIOD signal being pulled high. In the OV7670 datasheet [26], the I2C slave has an 8-bit ID to specify the write command as 'ox42' and the read command as 'ox43'.

In the I2C sender, multi-bit signal 'data_sr' is sequentially driven on SIOD. This 32-bit signal consists of 3-bit '100', 8-bit hex '42', 1-bit '0', 8-bit register address, 1-bit '0', 8-bits register value, 1-bit '0' and 2-bit '01' as shown in Fig. 2.4. Another signal called 'busy_sr' shows the situation of data writing. For example, when data is just stored into 'data_sr', 'busy_sr' should be 32 bit hex 'FFFFFFFF'. Then after one bit data is sent, 'busy_sr', the least significant bit of 'data_sr' signal becomes zero. This process repeats until all bits of 'busy_sr' become zero. After 'busy_sr' becomes zero, the 'token' signal becomes one and is sent to OV7670 register module to ask for next register value. To make the process correct, SIOC must follow the I2C protocol. This protocol was described in detail in the previous paragraph. The design detail of the I2C sender is shown in Fig. 2.6.

The register addresses and values are saved in the OV7670 register module shown in Fig. 2.5. Those register values are all from the Table 5 of OV7670 datasheet [26]. By trying the register values in the datasheet, the OV7670 camera is not working functionally as expected. Many of the register settings shown in the datasheet are without detail description. Thankfully, Mike Field [8] with help from Chirs Wilson, designed the necessary register values. Based on his design, some of the register values are changed to fit this platform. For example, register COM7 at address 12 write-in binary value '00010100' instead of '00000100'. The only change here is the fifth bit

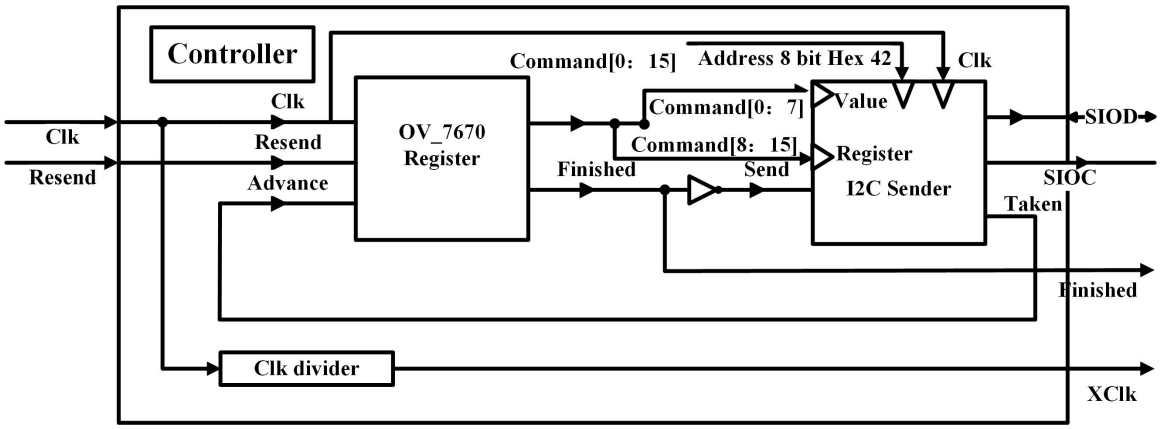


Figure 2.5: *OV7670 Controler*

which controls the output frame size as QVGA (320×240 resolution). The register HSTART, HSTOP, VSTART, VSTOP, HREF and VREF are also change to make the right timing as the QVGA output.

2.6 OV7670 Capture

The design of OV7670 Capture is shown in Fig. 2.8. Each set of data needs 4 clock cycles shown in Table. 2.1. The address signal is named 'addr' and the next address signal is named 'addr_next'. The 'wr_hold' signal is a 2-bit signal which holds the horizontal ref value, named as 'href', from the previous clock cycle. The 'd_latch' is a hold signal of the input data, and only on the third clock cycle does it have all the RGB 565 format data. And at the fourth clock cycle, 'dout' signal will consist of 'd_latch' [15:12], [10:7], [4:1]. And then the write enable (we) signal pulls high. The signal 'dout' is not equal to 'd_latch' because VGA output can only display the most significant 4 bits of the RGB. The vertical sync signal named as 'vsync' is initialized at low. When 'vsync' signal is pulled high, all the signals reset and the capture process begins. The value of horizontal reference signal (Href) is held 'wr_hold'. The

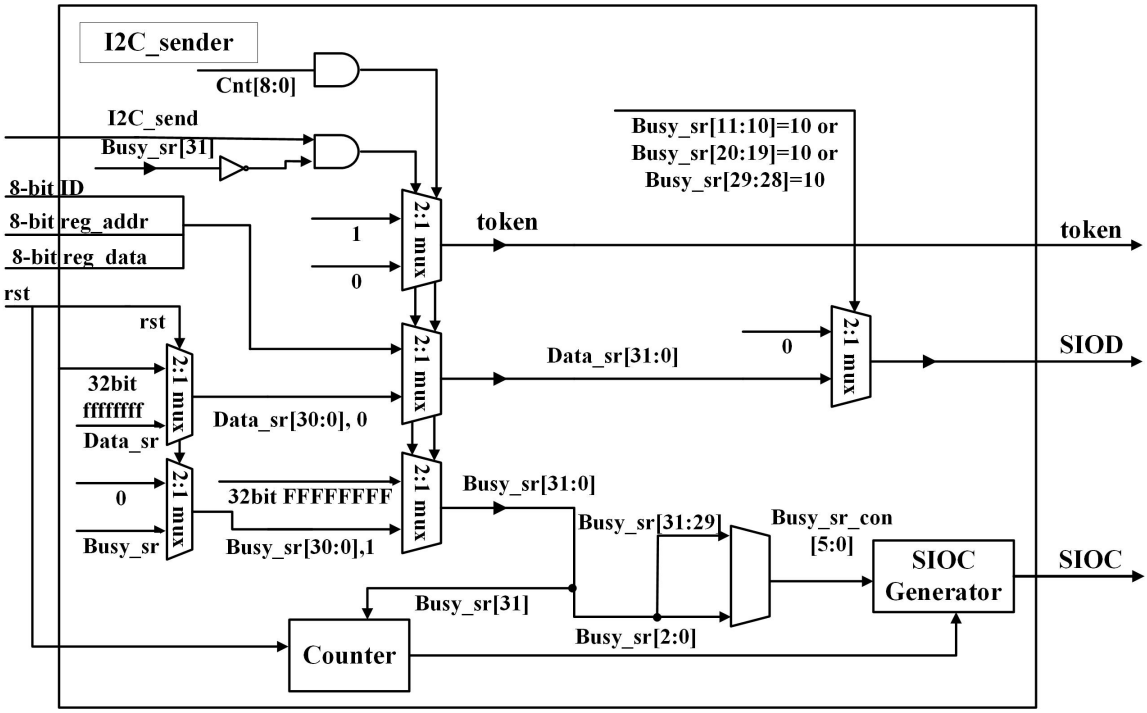


Figure 2.6: *OV7670 I2C Sender*

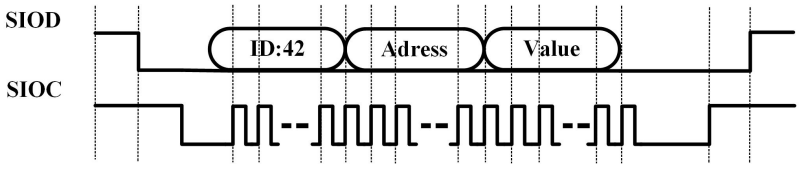


Figure 2.7: *I2C Timing*

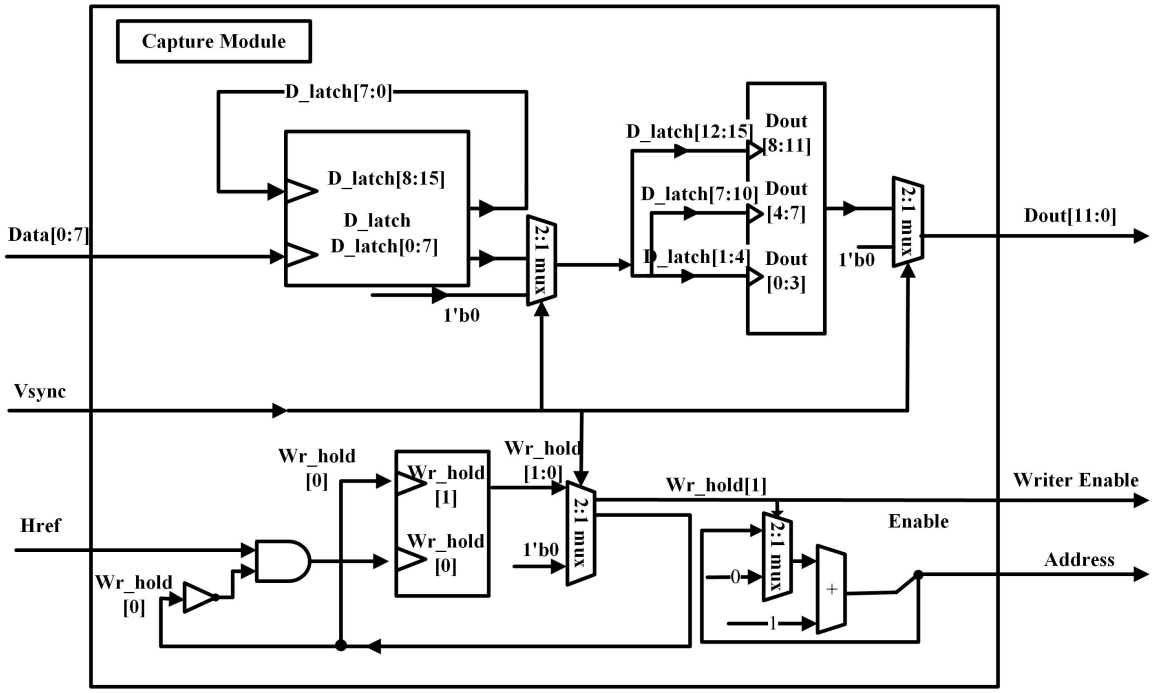


Figure 2.8: *OV7670 Capture*

Href	wr_hold	D_latch	we	addr	addr_next
x	XX	XXXXXXXXXXXXXXXX	X	XXXX	XXXX
1	X1	XXXXXXXXRRRRRGGG	X	XXXX	addr
0	10	RRRRRGGGGGGBBBBB	X	addr	addr
x	0X	GGGBBBBBXXXXXXXX	1	addr	addr+1

Table 2.1: *Data Transfer of RGB 565 Format*

'wr_hold[0]' is the current value of the 'Href' and the 'wr_hold[1]' is the previous of the 'Href'.

In this platform, two types of frame buffers are designed for simulation and FPGA implementation. The reason is that Xilinx Vivado uses logic cells on the FPGA board instead of the RAM if the buffer is not designed by block memory generator. The number of logic cells in the FPGA is not enough to implement all the buffers. However, if the frame buffer is generated by the block memory generator, ModelSim can not

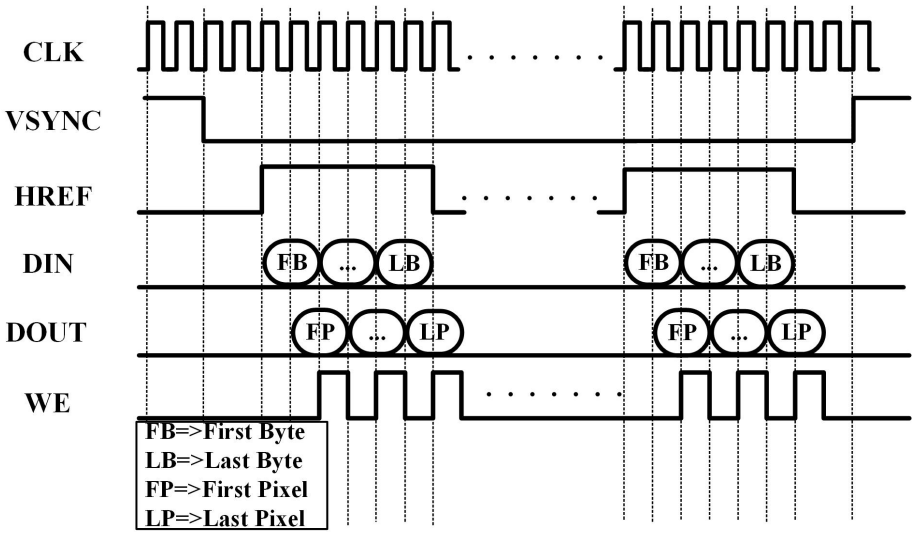


Figure 2.9: *OV7670 Capture Timing*

use the files to run the simulation. This conflict can be solved by separating the buffer design into a simulation design and an implementation IP.

2.7 VGA Master

The following Fig. 2.10 and Table. 2.2, display the design of the VGA port. There are 5 signals output to monitor: red data, green data, blue data, horizontal sync(Hsync), and vertical sync(Vsync). The display is enabled when receiving a logic high from Hsync and Vsync. The VGA refresh rate can be in between 50 Hz to 120 Hz based on different input sync signal. The sync signal timing required for a 640×480 resolution at a 60Hz refresh rate is shown in Table. 2.2. The signal Vsync is counted by the number of the lines, and Hsync is counted by the number of pixels in each of the line. More specifically, there are several processes that make up one sync process of Hsync. These are the display time (DIS), pulse width (PW), front porch (FP) and back porch (BP). Each process has different multiple parameters it needs to follow.

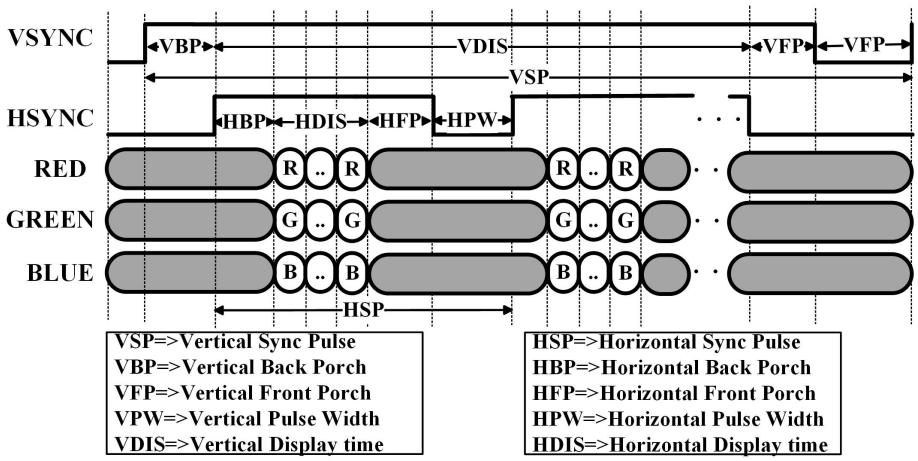


Figure 2.10: *Nexys-4 VGA timing*

Fig. 2.10 displays the timing detail of Hsync and Vsync. These two signals follow the VGA protocol, which states that only during the DIS process monitor can receive pixel data from red, green and blue channel. Following those details, the design of the VGA Master needs two counters. One counter counts the number of clock cycles for the horizontal lines and the other counter counts the number of the vertical lines. Fig. 2.1 displays the final result of this platform. There are four different regions of the output showing the frame data from different frame buffers. The 2-bit signal called 'Region' is designed to control the delivery of frame data. In the VGA master module, the counter of horizontal lines and vertical lines are being used to separate the frame data of the designed region. When the frame data is ready to go, the Vsync signal needs to wait for the delay back porch (BP) then pulls high. To finish the first frame, Vsync need to wait for front porch (FP) then pulls low. Similar to the Hsync, but Hsync repeats the step for each horizontal line.

Symbol	VYNC			HSYNC	
	Time (us)	Clock	Lines	Time (us)	Clock
SP	16,700	416,800	521	32	800
DIS	15,360	384,000	480	25.6	640
PW	64	1,600	2	3.84	96
FP	320	8,000	10	0.64	16
BP	928	23,200	29	1.92	48

Table 2.2: *VGA Timing Table*

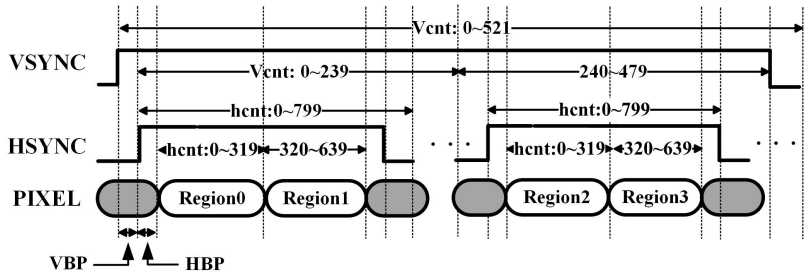


Figure 2.11: *Display timing for different Regions*

2.8 Display Regions

The 640 × 480 display window can simultaneously display four 320 × 240 images. To place each image into the correct position, two counters are designed to recognize the timing. The counter 'hcnt' for the horizontal sync timing and the counter 'vcnt' for the vertical sync timing. Fig. 2.11 shows that when 'vcnt' is in between 0 to 239 Region0 and Region1 will be selected. And when 'vcnt' is in between 240 to 479 Region2 and the Region3 will be selected. Likewise, while 'hcnt' is in between 0 to 319 the Region0 and the Region2 will be selected. Region1 and Region3 will be selected when 'hcnt' is in between 320 to 639 counts.

2.9 IPs of PLL and Frame Buffer

In the top module of this design, five IPs are generated for FPGA implementation on Xilinx Vivado. These five IPs are one PLL and four Frame Buffers. Fig. 2.12(a)

Clock	Input	Form	Freq-(MHz)
Clk_100MHz	Clock Generator	FPGA	100
Clk_50MHz	I2C Master	PLL output	50
Clk_25MHz	VGA Master	PLL output	25
PClk	image caputre slave	OV7670 Camera	20
SIOC	Camera Register	I2C Master	0.2

Table 2.3: *Clocks For The DUT*

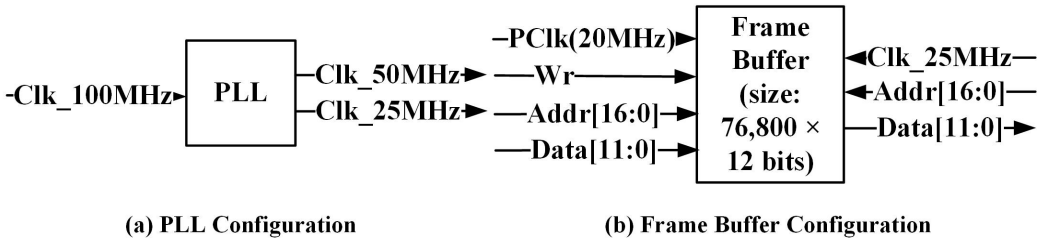


Figure 2.12: *IP module*

shows the design configuration of both the PLL and the Frame Buffer. The PLL is used to divide the original 100MHz clock rate into 50MHz and 25MHz clock rate. The 50MHz clock rate is applied to the I2C master and the 25MHz clock rate is applied to the VGA master. Table 2.3 summarizes the result of this process.

Fig. 2.12(b) displays the process diagram of the frame buffer. The calculation of the size of the frame buffer is based on 320 × 240 × 12bits = 76,800 × 12bits. The write clock rate is the pixel clock from the OV7670 called 'PClk'. This clock rate is usually around 20MHz. The read clock rate has to be the same as the VGA Master, meaning that the clock rate is exactly 25MHz.

2.10 Experimental Result

The verification result and implementation result are discussed in this section.

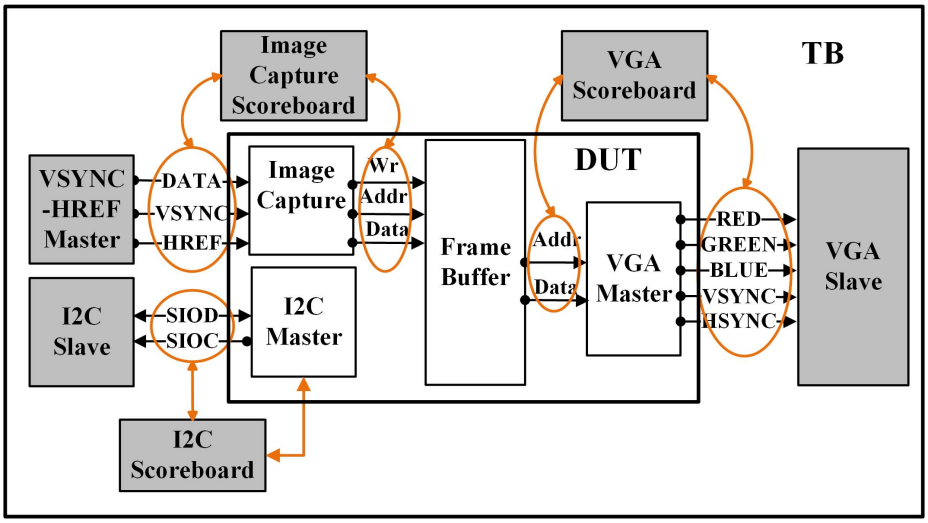


Figure 2.13: *Verification Overview*

2.11 Design Verification

To verify the proposed platform, a verification environment is provided in this section. Fig. 2.13 shows the overview of this verification environment. Before verifying the whole system, it is necessary to check that each individual module is fully functional. Fig. 2.13 shows the three scoreboards that are designed to check each input and output following the design protocol. The subsections below shows the verification detail of each module.

2.12 I2C Controller Verification

To verify the I2C controller in ModelSim, a run script is designed to include all the files and display the important signals in the waveform. The I2C scoreboard sends 25 Mhz clock and a '0' rst signal to the I2C Master module, and concurrently monitors the outputs signal SIOD and SIOC. After receives the clock and '0' rst signals, the I2C master begins to generate the signal SIOD and SIOC. I2C slave is designed as a memory blcok meant to receive and store the register data from the I2C master by

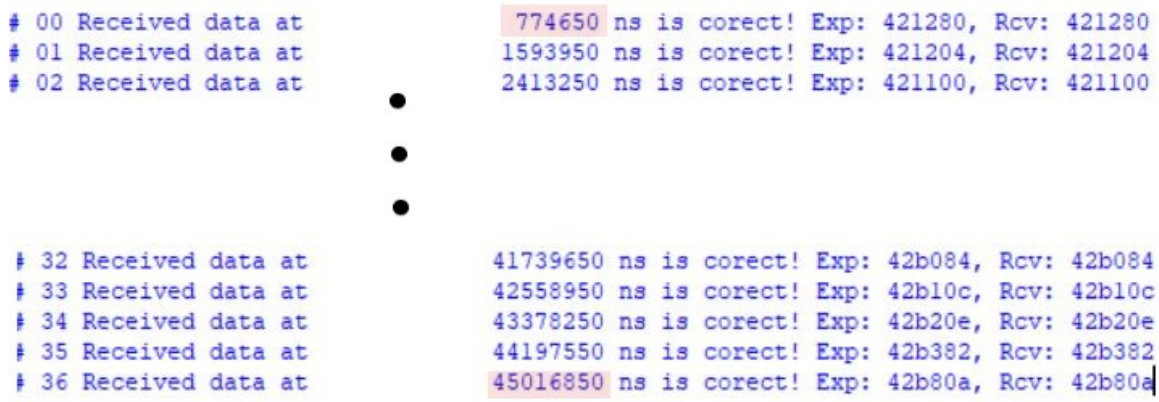


Figure 2.14: *Controller Verification Result*

the I2C protocol. On each received register data, the I2C scoreboard will output the count number of this data, received time, the exact data and the data displayed to the ModelSim transcript. In this simulation case, 54 register data sets are received correctly. First data is received at 774650 ns and the last data at 45016850 ns or 0.045 seconds.

2.13 OV7670 Capture Verification

The register setting of OV7670 camera mentioned in section 2.5, the capture module needs to transfer RGB 565 data to RGB 444. Shown in Fig. 2.13, VSYNC-HREF Master is required to send the RGB 565 data, VSYNC and HREF. All the signals need to follow the same timing displayed in OV7670 camera datasheet. This is shown in Fig. 2.9. A file named "RGB565_Golden.txt" that is produced by Matlab from a regular RGB image used as the golden model of input. The golden model of the ouput is produced from the same image but in the RGB 444 format, named as "RGB444_Golden.txt". The size of this image is 320 $\times$ 240 resolution which same as the register settings. The OV7670 capture scoreboard compares the output data to the golden model when write enable is high. The result are in the ModelSim

```
# 00bf8 Received data at          15998550 ns is corect! Exp: ffe, Rcv: ffe
# 00bf9 Received data at          15998750 ns is corect! Exp: ffe, Rcv: ffe
# 00bfa Received data at          15998950 ns is corect! Exp: ffe, Rcv: ffe
# 00bfb Received data at          15999150 ns is corect! Exp: ffe, Rcv: ffe
# 00bfc Received data at          15999350 ns is corect! Exp: ffe, Rcv: ffe
# 00bfd Received data at          15999550 ns is corect! Exp: ffe, Rcv: ffe
# 00bfe Received data at          15999750 ns is corect! Exp: ffe, Rcv: ffe
# 00bff Received data at          15999950 ns is corect! Exp: ffe, Rcv: ffe
```

Figure 2.15: *Capture Verification Result*

transcript which are displayed in Fig.2.15. To receive one frame requires 15999950 ns or 0.016 seconds.

2.14 VGA Master Verification

The VGA Master module is needed to verify the output signal of vsync and hsync. This two signals follow the VGA protocol which was discussed in section 2.7. The VGA Slave receive the data from the VGA Master, place the data by vsync and hsync signal and then save the data into txt file. The VGA Scoreboard collects the input and output signal from the VGA Master. The sync signal timing should follow the Table. 2.2, if it does not there can be a dislocation in the .txt file which compared to the golden model. The ModelSim transcript shows that the running time is 15999640 ns or 0.0159 seconds per frame. For the 60Hz VGA refresh rate, the time for each frame should be 0.01667 seconds which is very close to the result from the previous simulation.

2.15 SOC System Verification

After all the module included in this design pass verification, a system verification is needed to ensure the design works in system level. In the verification of System On Chip (SOC), all the modules are combined to test as a system. So testbench module

```
# 12bf9 Received data at              25330360 ns is corect! Exp: red pixel - a, gree
n pixel - 5, blue pixel - 0; Rcv: red pixel - a, green pixel - 5, blue pixel - 0
# 12bfa Received data at              25330400 ns is corect! Exp: red pixel - a, gree
n pixel - 5, blue pixel - 1; Rcv: red pixel - a, green pixel - 5, blue pixel - 1
# 12bfb Received data at              25330440 ns is corect! Exp: red pixel - a, gree
n pixel - 6, blue pixel - 1; Rcv: red pixel - a, green pixel - 6, blue pixel - 1
# 12bfc Received data at              25330480 ns is corect! Exp: red pixel - a, gree
n pixel - 6, blue pixel - 2; Rcv: red pixel - a, green pixel - 6, blue pixel - 2
# 12bfd Received data at              25330520 ns is corect! Exp: red pixel - b, gree
n pixel - 7, blue pixel - 3; Rcv: red pixel - b, green pixel - 7, blue pixel - 3
# 12bfe Received data at              25330560 ns is corect! Exp: red pixel - d, gree
n pixel - a, blue pixel - 6; Rcv: red pixel - d, green pixel - a, blue pixel - 6
# 12bff Received data at              25330600 ns is corect! Exp: red pixel - e, gree
n pixel - b, blue pixel - 7; Rcv: red pixel - e, green pixel - b, blue pixel - 7
```

Figure 2.16: *Soc Verification Result*

is designed to receive control data and give feedback data in the correct timing. For reasons mentioned previously, the RAM used in this system is designed for simulation only. Fig. 2.13 displays overview of the verification environment.

In the testbench module, we designed 4 memory block; one store input data and the rest are use to store output. This output is considered to the golden model. The input data is a 320 $\times$ 240 resolution image in RGB 565 format. The output golden model is the same image but in RGB 444 format. In addition, it is separated in three channels: R, G, and B. The register values are sent first, after all 54 register values are set into the camera, a signal called 'finished' goes high, and the testbench process to send the image data, VSYNC, and HREF. Following this the 'address' signal, 'write-enable' signal, and image data in RGB 444 format are sent to the RAM called 'framebuffer'. The VGA Master is going to generate the address from the counter. This address will then be used to read the data from the 'framebuffer'. Then we processed to compare the RGB 444 data to the golden model stored in the memory block. The result displayed in ModelSim transcript are shown in Fig. 2.16. As displayed, the results show that our plat from functional correctly.

Name	Slice LUTs	Slice Register	RAM	IO
blk_mem0	134	11	26.5	0
blk_mem1	134	11	26.5	0
blk_mem2	134	11	26.5	0
blk_mem3	134	11	26.5	0
image_capture	2	43	0	0
ov7670_controller	87	90	0	0
vga	134	100	0	0
Top	753	277	0	34

Table 2.4: *Resource Cost of The Platform*

Resource Cost	[1]	[28]	Our Work
Devices	Altera DE-115	Xilinx Virtex 5	Xilinx Nexys 4
Slice LUTs	1,616	10,283	753
Slice Registers	818	9,974	277
Block RAMs	-	52	106
IOs	94	-	34

Table 2.5: *Resource Cost Comparison*

2.16 Experimental Result

In this work, the simulation was performed on Mentor Graphic ModelSim 10.4d and the synthesis/implementation on Xilinx Vivado. The test device utilized was a Nexys-4 FPGA. Eventually the power consumption result are analysed by XPower Analyzer [35].

2.17 Resource Cost

The resource cost are determined by slice count, RAM utilization, and the number of IOs as shown in the Table 2.4. In the last row shows, the total number of slice LUTs are 753, slice registers are 277, RAM unit are 106, and IO port are 34 used in this work.

TP(mW)	SP(mW)	DP(mW)					
		Clock	Signal	Logic	BRAM	PLL	I/O
220	102	2	4	1	10	97	4

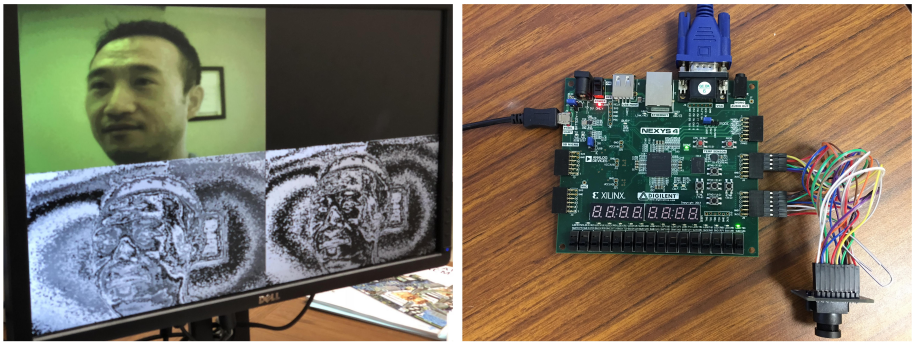
Table 2.6: *Power Consumption on Nexys 4 FPGA*

Moreover, the Table 2.5 compares the resource cost of this design with the existing work [28], [1]. The third column shows the resource cost of image acquisition and processing using LabView FPGA [28]. Obviously, the design uses much more hardware resource compared to the RTL designs in the second and fourth columns.

The implementation resource cost of color to grayscale conversion and edge detection on Altera DE-115 FPGA board [1] is showm in the second column. Compared with our proposed design shown in the fourth column, Ref. [1] has 53.4% more LUTs and 66.1% more registers. It also uses more than twice of the IOs than our platform. Generally, the high number of logics and IOs increases the switching activities of signals resulting in high power consumption on FPGA design.

2.18 Power consumption

Table 2.6 shows the total power consumption (TP) is 220mW, the static power consumption (SP) is 102mW, and the dynamic power consumption (DP) is 118mW. Due to the reduced number of logics and IOs, the power measured from the toggle rate of clocks, signals, logics, and IOs is only 11mW or 9.3% of DP, as shown in the third, fourth, and fifth columns. The rest of power consumption is comes mainly from the BRAM and the PLL. This result to totally 107mW or 90.7% of DP. The two previous works, [1] and [28], did not estimate the power cost. Therefore, a comparison with our work is not possible



(a) FPGA Results with Original Color Image in Region0, Black in Region1, Grayscale in Region2, and Binary in Region3

(b) FPGA with OV7670 Camera and VGA

Figure 2.17: *FPGA Prototype*

2.19 FPGA Prototype

After programming the design netlist on Xilinx Nexys 4 FPGA, by Xilinx Vivado. Fig. 2.17 shows the demo of displaying original video, and the enhanced images in grayscale and binary. Notice that the region not utilized in this demo shows all black pixels.

2.20 Summary

This chapter presents a scalable image/video processing platform on FPGAs containing not only open source design code but also a verification environment. The power consumption and slice count of our work is significantly reduced when compared to the prior works. The most important here is that the reusable and the expandable to a diverse range of algorithm in image processing and computer vision of this platform. In future, we expect our work will lead to multiple designs and give some contribution on research and education of this area.

CHAPTER 3

APPROXIMATE DESIGN

3.1 Hierarchical Synthesis of Approximate Design

This section presents a novel hierarchical synthesis for FPGA based adders and multipliers. Our proposed work is able to implement the multiplier design with the following contributions: 1) providing four types of single-bit approximate adders employed by implementing the multiplications, 2) presenting many approximations of the multiplier, enable to cover a wide range of energy-quality tradeoffs, 3) substituting corresponding bits of the approximate multiplier considering the quality constrains of the results. The novel hierarchical synthesis of approach has been integrated into our prior project, an FPGA-IoTmesh system in the field of fog computing for hardware acceleration. Combining the merits of reconfigurability of FPGAs and long-distance connection of CSRmesh technology, this work creates a diverse range of applications such as approximate designs at the network edge, as well as showing a demo for Internet-of-Things (IoT) connections covering an entire building.

3.2 Proposed Works

In this subsection, we start from the fundamental single-bit adders' design. Then, four approximate additions are adopted to implement the 4-bit multipliers as a case study. The tradeoffs between accuracy and resource cost are further estimated.

3.2.0.1 Approximate adders

The sum and carry bits, denoted as Sum and Cout, of the conventional single-bit full adder can be expressed as

$$Sum(EX) = A'B'C + A'BC' + AB'C' + ABC. \tag{3.1}$$

$$Cout(EX) = AC + BC + AB. \tag{3.2}$$

where A and B represent 2 single-bit inputs and C indicates the carry-in bit.

In what follow, we modify the K-map of the basic single-bit adder in order to reduce the gate count. As an example shown in Fig. 3.1(a), the Sum result is modified from 1 to 0 and the Cout result is changed from 0 to 1 when A=1, B=0 and C=0. After plotting the maximum group of 1's on the map, the algebraic expressions can be simplified as

$$Sum(AP1) = A'BC' + ABC + A'B'C. \tag{3.3}$$

$$Cout(AP1) = A + BC. \tag{3.4}$$

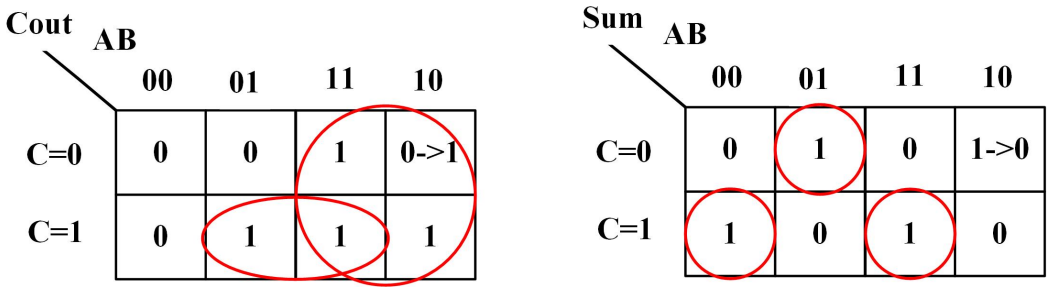
Comparing with the conventional full adder design, the AP1 design simplifies algebraic expressions so as to reduce the hardware cost and power consumption. Likewise, Fig. 3.1(b), 3.1(c), and 3.1(d) also show the modified K-maps of the other three different approximate adders. In the same way the algebraic expressions can be rewritten as

$$Sum(AP2) = A'C + BC + A'B \tag{3.5}$$

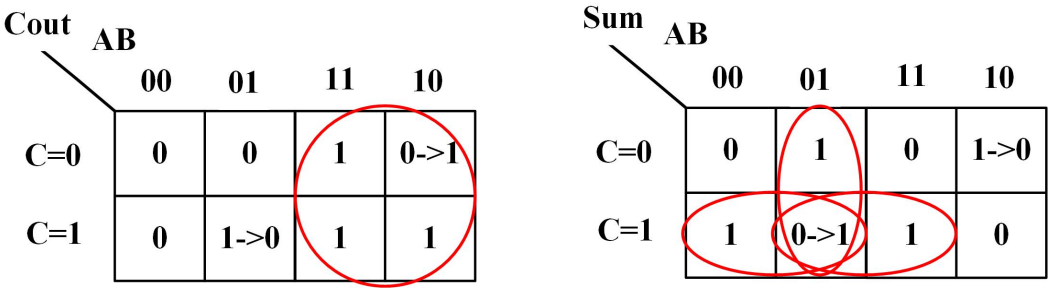
$$Cout(AP2) = A \tag{3.6}$$

$$Sum(AP3) = B + A'C \tag{3.7}$$

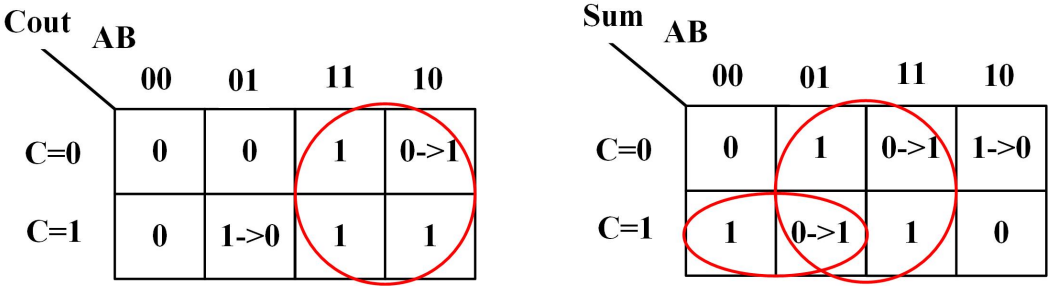
$$Cout(AP3) = A \tag{3.8}$$



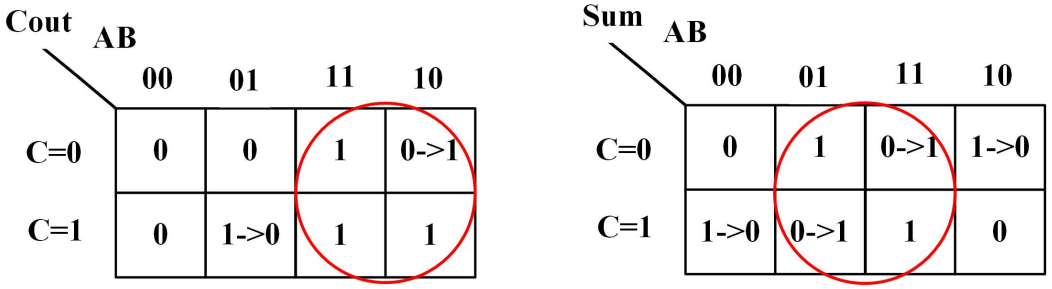
(a) AP1 Adder's K-map



(b) AP2 Adder's K-map



(c) AP3 Adder's K-map



(d) AP4 Adder's K-map

Figure 3.1: *Algebraic expressions of Approximate Additions*

$$Sum(AP4) = B \tag{3.9}$$

$$Cout(AP4) = A \tag{3.10}$$

It can be observed that the AP1 expression costs the largest number of gate count and the AP4 consumes the least in the four approximate designs. In theory, the implementation with more resource cost, is likely to achieve higher accuracy and vice-versa, which will be proved in the following subsection.

3.2.0.2 Approximate multipliers' design and evaluation

Based on the aforementioned adders' design, the 4-bit unsigned multipliers can be implemented in this subsection. Basically, a binary multiplication requires shifting and adding, as shown in Fig. 3.2. In this figure, all the product bits, from the Least Significant Bit (LSB) to the Most Significant Bit (MSB), are calculated by exact adders.

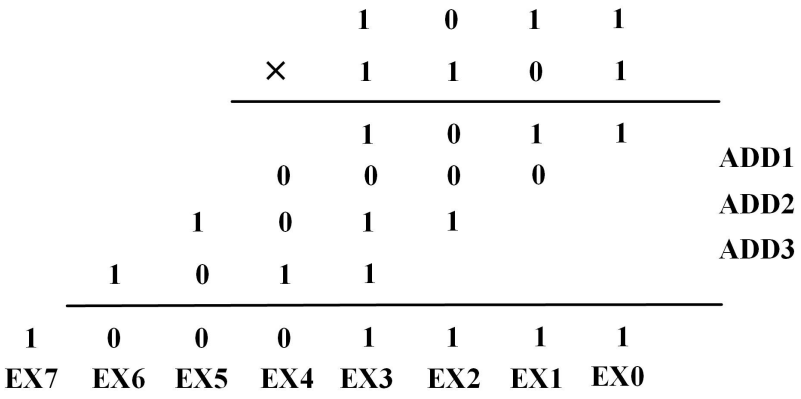
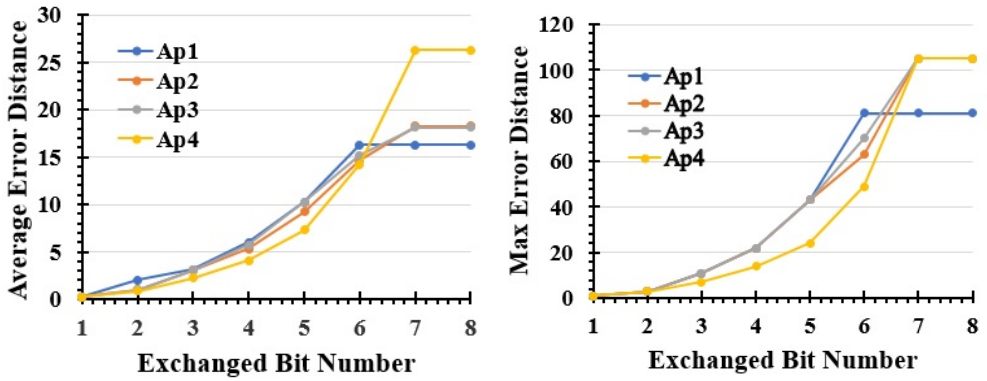


Figure 3.2: *4-bit Multiplication*

To estimate the accuracy of the approximate multiplications, we replaced the exact single-bit adders by approximate adders (AP1, AP2, AP3, and AP4) from LSB to MSB. The Error Distance (ED), formulated as $ED = Absolute(R - R*)$ where



(a) Average Error Distance (b) Maximum Error Distance

Figure 3.3: *Error Data*

R represents the exact result and R* indicates the approximate result, is applied to evaluate the multiplications' accuracy. Since the 4-bit multiplication has $2^4 \times 2^4 = 256$ possible combined inputs, the Average Error Distance (AED) can be written as

$$AED = \frac{\sum_{i=0}^{255} ED_i}{256}; \tag{3.11}$$

Experimental results in Fig. 3.3(a) demonstrate our expectation, that the AP1 based design achieves the minimum average error distance in all the approximate implementations, it costs the most gate count however. For example, in the case of replacing all the 8-bit additions, the average error distances are 16.25, 18.23, 18.09, and 26.25, using AP1, AP2, AP3, and AP4, respectively.

The Maximum Error Distance (MED) is also applied to estimate the worst case of the approximate designs, which can be formulated as

$$MED = max\{ED_0, ED_1, ED_2, ..., ED_{255}\}; \tag{3.12}$$

As shown in Fig. 3.3(b), the worst case for each approximate design happens when all the 8-bit additions are modified. For example, when they are replaced with AP1 and AP2, the maximum error distances are 81 and 105, respectively.

3.3 Implementation

As a case study, the histogram equalization algorithm is used to estimate the performance of the approximate computations in this subsection. Generally the histogram equalization is used to enhance the contrast of images by transforming the values in an intensity image, or the values in the colormap of an indexed image, so that the histogram of the output image approximately matches a specified histogram.

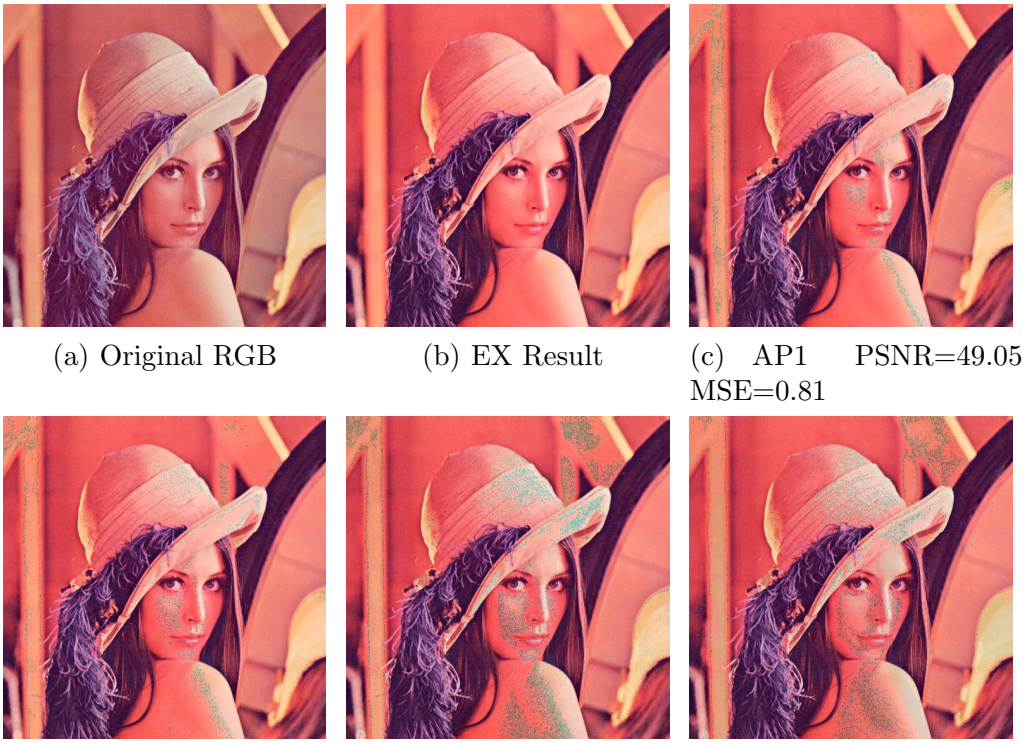
The pseudocode for implementing the histogram equalization algorithm is depicted in Algo. 1. It basically contains two procedures. In *procedure*#1 we count the number of each grayscale pixel. Then, the histogram results are computed in *procedure*#2 as the division of the approximation of multiplications over the size of the image. Finally the regulated results are distributed to each pixel in order to achieve a better image with improved contrast.

Algorithm 1 The Histogram Equalization Algorithm Design with Approximations

Input: *Pixel array* $p[i](i = 1 - 256)$; *pixel number* $s[i]$ of $value = i - 1$;
Output: $Configurated\ pixels : s[j]$;

1: **procedure** #1 - FIND THE NUMBER OF GRAYSCALE PIXELS:$(s[i] - p[i])$
2: $initialize : s(0) = p(0)$;
3: **for** $(i = 1; i \leq 256; i = i + 1)$ **do**
4: $s[i] = p[i] + s(i - 1)$;
5: **procedure** #2 - HISTOGRAM EQUALIZATION:$(s[i])$
6: **for** $(j = 1; j \leq 256; j = i + 1)$ **do**
7: $s[j] = \frac{approx\_comp(s[j]) \times 256}{Height \times Width}$;
8: if $(s[j] < 256)$ $s[j] = 256$;

Fig. 3.4 and 3.5 show the results of RGB and grayscale images, respectively. To demonstrate the difference, we use the Peak Signal-to-Noise Ratio (PSNR), a term for the ratio between the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation, as one of the performance estimation parameters. It can be formulated as:



(a) Original RGB (b) EX Result (c) AP1 PSNR=49.05 MSE=0.81

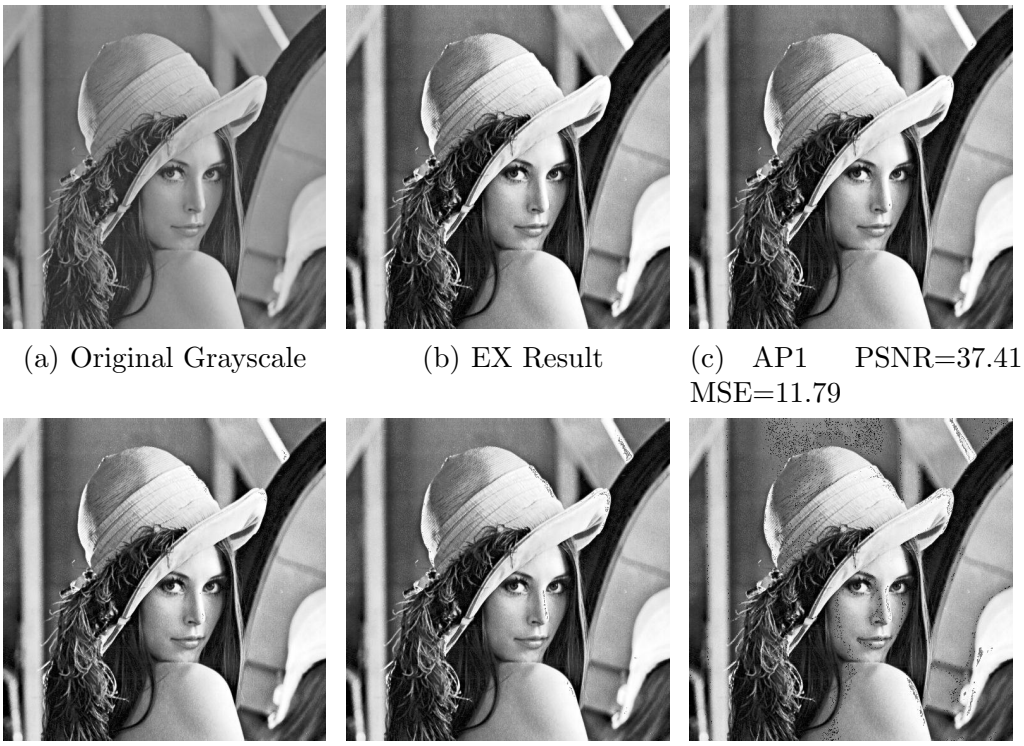
(d) AP2: PSNR=47.34 MSE=1.99 (e) AP3: PSNR=42.13 MSE=3.98 (f) AP4: PSNR=36.07 MSE=16.08

Figure 3.4: *Equalized RGB Image*

$$PSNR = 10 \times \log_{10} \frac{255^2}{MSE}; \tag{3.13}$$

where MSE is the mean squared error. Typical values for the PSNR in lossy image and video compression are between 30 and 50 dB, provided the bit depth is 8 bits, where higher is better.

First, we compare the Fig. 3.4(a) and Fig. 3.4(b). It can be observed that the equalized rgb image achieves higher contrast compared with the original image using the exact multipliers. From Fig. 3.4(c) to Fig. 3.4(f), the quality of the results has been degraded but the contrast is still enhanced compared with the Fig. 3.4(a). The higher approximations of the multipliers are employed, the worse quality of the results.



(a) Original Grayscale (b) EX Result (c) AP1 PSNR=37.41 MSE=11.79

(d) AP2 PSNR=37.02 MSE=12.91 (e) AP3 PSNR=33.65 MSE=28.06 (f) AP4 PSNR=32.62 MSE=35.53

Figure 3.5: *Equalized Grayscale Image*

Similarly, Fig. 3.5 depicts the experimental results of the grayscale images. Note that the higher approximation degrees of the multipliers are employed, the lower of the PSNRs are achieved.

3.4 Experimental Results

Using the four approximate multipliers in the histogram equalization, the histograms of RGB images and the histograms of grayscale images are shown in Fig. 3.6 and Fig. 3.7, respectively. The goal of using the histogram equalization is to make the images to use entire range of values available to them.

Basically, histogram equalization is a nonlinear normalization that stretches the area of histogram with high abundance intensities and compresses the area with low

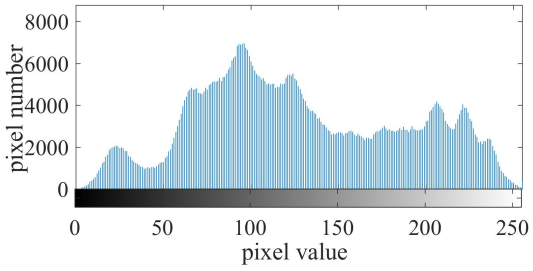
abundance intensities. As an example shown in Fig. 3.7, the equalized grayscale image is flattened in Fig. 3.7(b) compared with the original Fig.r̃efsubfig:hisorgrorg. More important, the quality of the equalized results of Fig. 3.7(c)–Fig. 3.7(f) are not as good as Fig. 3.7(b) due to the imprecise multiplications, however, the images are normalized to the original image in Fig. 3.7(a). In some of the application domains with a tolerance of errors, the imprecise results are acceptable within the quality bound, and the resource cost and power consumption can be significantly reduced due to the approximating designs.

3.5 Conclusion

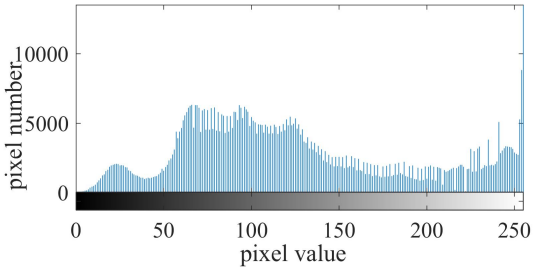
In this section, we presented a new methodology to design approximate adders and multipliers that exchange the single-bit computation based on different applications. The experimental results show that our proposed work achieves similar results to the exact design on a very common image processing algorithm, the histogram equalization. As a tradeoff, the hardware resource cost can be significantly reduced due to the imprecise computation on FPGA. Our future work will keep developing more approximate design components in our approximate library and focus on more complicate algorithm on image processing and data mining.

3.6 Advance Approximate application

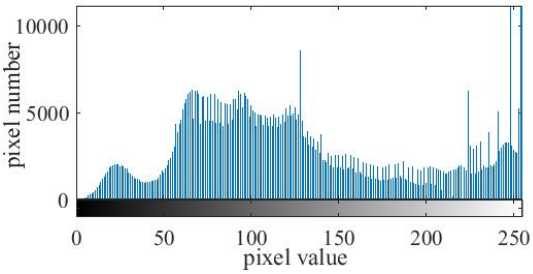
In the chapter one, we developed a multiplier by using adder. In this chapter we propose a advanced design of multiplier. Compare to the design from last chapter, we have improved the accuracy and energy consumption. To find the accuracy and energy difference are the most important point of approximate computing. So we



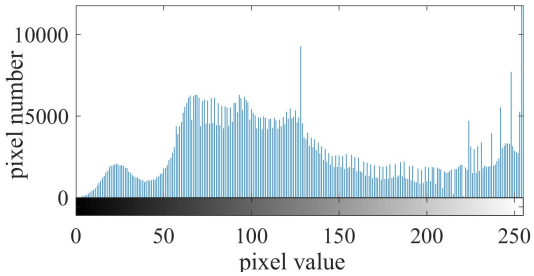
(a) Histogram of Original RGB image



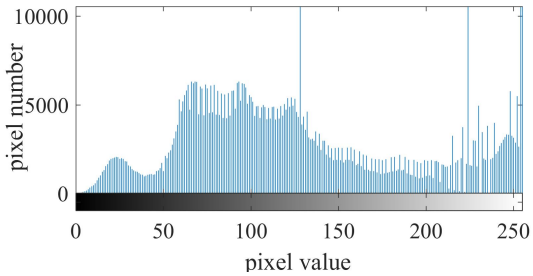
(b) Histogram of Equalized RGB image



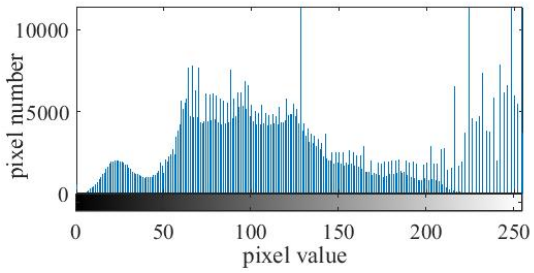
(c) Histogram of Equalized RGB image



(d) Histogram of Equalized RGB image

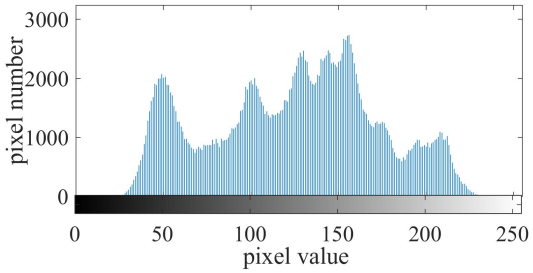


(e) Histogram of Equalized RGB image

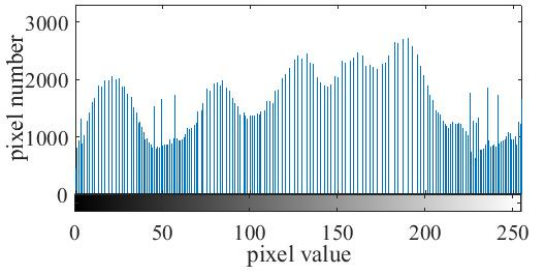


(f) Histogram of Equalized RGB image

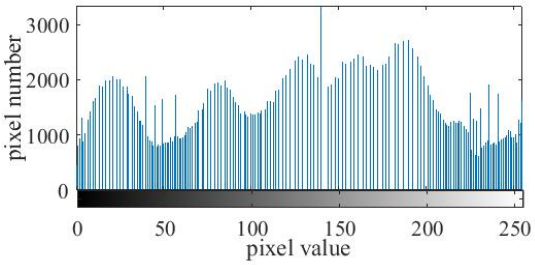
Figure 3.6: *Comparison of Histograms of RGB Images*



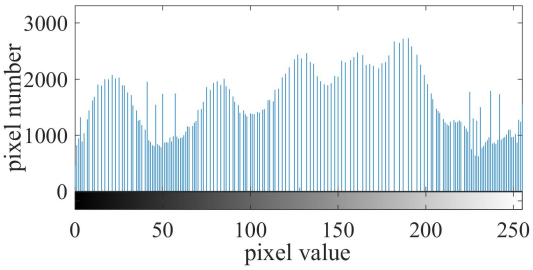
(a) Histogram of Original Grayscale Image



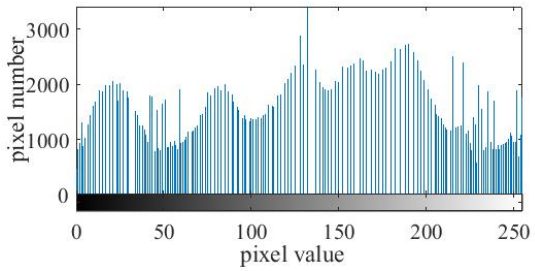
(b) Histogram of Equalized Grayscale Image



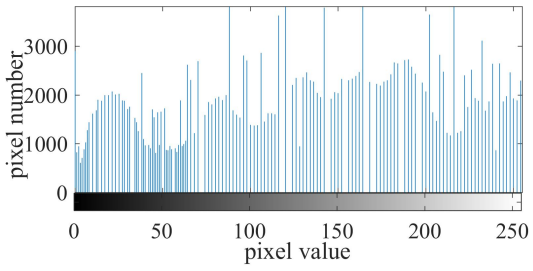
(c) Histogram of Equalized Grayscale Image



(d) Histogram of Equalized Grayscale Image



(e) Histogram of Equalized Grayscale Image



(f) Histogram of Equalized Grayscale Image

Figure 3.7: *Comparison of Histograms of Grayscale Images*

will use both Matlab simulation and FPGA application to find out the accuracy and energy consumption.

3.7 Slice-Energy Saving on FPGA Platform with Approximate Computing

3.7.0.1 Energy Consumption on FPGA

Generally energy can be computed by running time and power as Energy = Power × Time. Thus to reduce the energy cost we can either increasing the Maximum Operating Frequency (MOF) or decreasing the power consumption. However, decreasing the clock frequency can lower the system speed and cost more time on each cycle, resulting an increasing of energy dissipation. Thus the power reduction is mainly considered in this work to save energy. Since the static power on FPGAs is dependent on the specific designs, in this work we focus on dynamic power estimation, which can be expressed as

$$P_{dyn} = (V^2) \times \sum_{i=1}^{n} C_{eff-i} \times U_i \times f_i \tag{3.14}$$

where the total switching capacitance is the product of its effective capacitance C_{eff-i}, the number of instances in the design U_i , and the average switching frequency across all the instances f_i including the logic, signals, and IOs. The dynamic power of switching all instances of resource i is the product of (V^2) and its switching capacitance. From Eq. (1), one of the most effective ways to lower the dynamic power is reducing number of logic, IOs, and signals employed in a digital system design.

3.7.0.2 A Case Study of RGB2Grayscale Converter

As a case study, the RGB2Grayscale converter is implemented as different approximations of designs in our work. Basically a grayscale pixel can be computed as a summation of 29.89% of the red pixel, 58.7% of green, and 11.4% of blue, which is

written as below

$$Grayscale = o.2989 \times r + 0.587 \times g + 0.114 \times b. \tag{3.15}$$

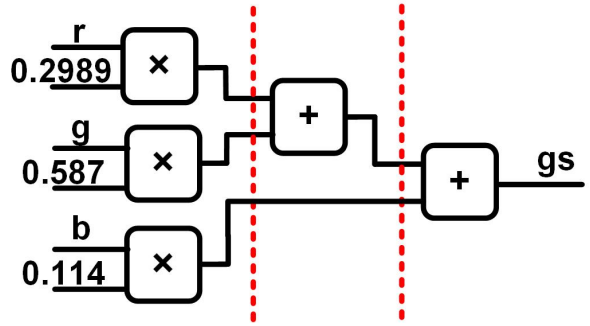
Fig. 3.8(a) shows the design structure with three floating-point multipliers and two adders. Although the floating-point multiplier is precise in results but it costs much more slices compared to the fixed-point designs. Therefore, this paper presents a fixed-point design shown in Fig. 3.8(b) by multiplying 2^8 for the three floating-point constants then rounding the fractions up, and finally dividing by 2^8 after the addition.

$$Grayscale = (76 \times r + 150 \times g + 30 \times b)/256. \tag{3.16}$$

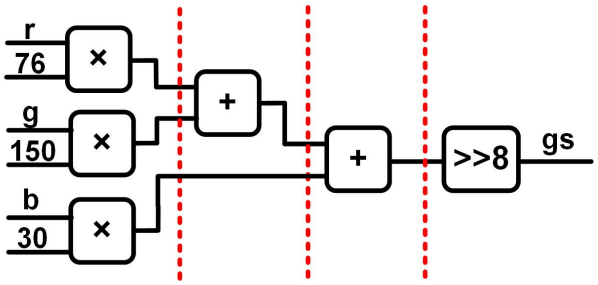
Furthermore, we present and apply several approximations of multipliers and adders in the design, in order to trade the accuracy for the energy efficiency. Theoretically the higher approximation of the multiplier, the higher energy efficiency can be achieved.

3.7.0.3 Approximate Multiplier

In this subsection, four approximations of 2×2-bit multiplier are presented. Notice that any multi-bit multiplication, denoted as WW-bit, can be integrated by four $W/2 \times W/2$-bit designs. As an example shown in Fig. 3.9, two multi-bit inputs A and B can be represented as $(A_H A_L)$ and $(B_H B_L)$ with the MSB AH and BH, and the LSB AL and BL. Then the sum of four partial products, denoted as $ALBL$, $A_H \times B_L$, $A_L \times B_H$ and $A_H \times B_H$, is the final product of WW-bit multiplication. In what follows, we propose three approximate 2×2-bit multipliers for different energy dissipation corresponding different quality constrains. Before discussing the approximate design, the exact 2×2-bit multiplication can be implemented using the K-map shown in Fig. 3.10 and written as In a 2×2 multiplier, there have two 2-bits input and one 4-bits output. Two inputs denoted as 'A' and 'B', the output called 'Mulout'. The



(a) Floating-Point



(b) Fixed-Point

Figure 3.8: *Design Structures of RGB2Grayscale Coverter*

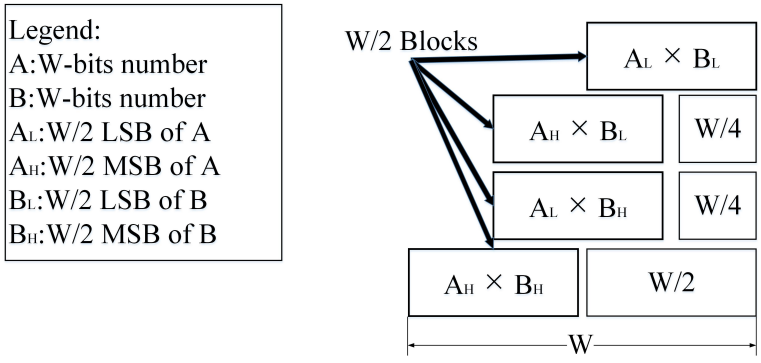


Figure 3.9: *Increase the size of Multiplier*

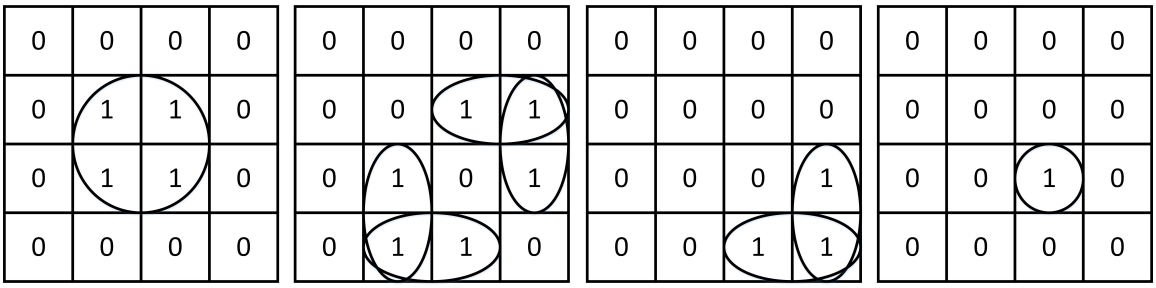


Figure 3.10: *Exact multiplier K-map*

conventional 2×2 multiplier can be expected as

$$Mulout[0](EX) = A[0]B[0]. \tag{3.17a}$$

$$Mulout[1](EX) = A[1]'A[0]B[1] + A[0]B[1]B[0]' + A[1]B[1]'B[0] + A[1]A[0]'B[0]. \tag{3.17b}$$

$$Mulout[2](EX) = A[1]B[1]B[0]' + A[1]A[0]'B[1]. \tag{3.17c}$$

$$Mulout[3](EX) = A[1]A[0]B[1]B[0]. \tag{3.17d}$$

where Mul[3], Mul[2], Mul[1], and Mul[0] are the four bits of the products, from the MSB to LSB. And A[1:0] and B[0] and the 2-bit input of the multiplier.

The corresponding design structure of the exact multiplier is shown in Fig. 3.11, requiring sixteen AND gates and four OR gates. More specifically, the MSB bit computation takes three AND gate and the LSB takes one AND gate. The middle bit Mul[2] needs four AND gates and one OR gate, and the Mul[1] bit requires eight AND gates and three OR gates. Since the LSB only uses one AND gate, it does not need to be simplified. To reduce the gate count for Mul[3:1], we present some approximations by modifying three bits from 0 and 1 in the K-map. For example shown in Fig. 3.12, we change 0 to 1 for the case A[1:0]=11 and B[1:0]=01, leading to a simple boolean expression as

$$Mulout[3](AP1) = A[1]A[0]B[0]. \tag{3.18}$$

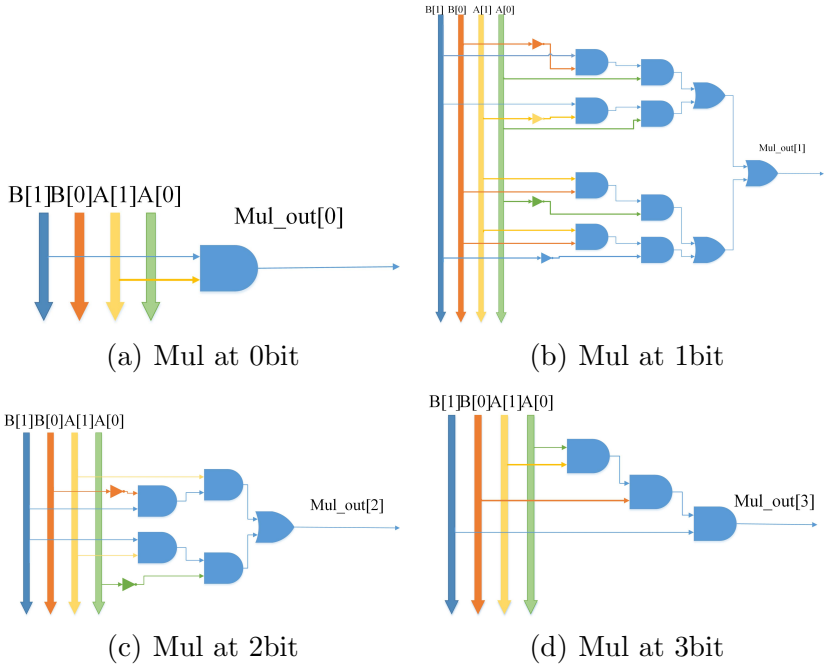


Figure 3.11: *Exact Multiplier Design*

Comparing to Fig. 3.11(d) with the exact Mul[3] bit computation, the approximate design in Fig. 3.12 reduces one AND gate within the criticial path and one AND gate for the totoal gate count as well, which theoritically would achieve a higher MOF and lower power dissipation.

Similarly, the exact computation of Mul[2] in Fig. 3.11(c) can be improved in resource cost by modifying 0 to 1 at A[1:0]=11 and B[1:0]=11, as shown in Fig. 3.13. The boolean expression is rewritten as

$$Mulout[2](AP2) = A[1]B[1]. \tag{3.19}$$

After the optimization, the total gate cost is reduced from five gates to one gate. In other words, the approximation sacrifices one bit error for saving 80% gate numbers compared to the Eact design.

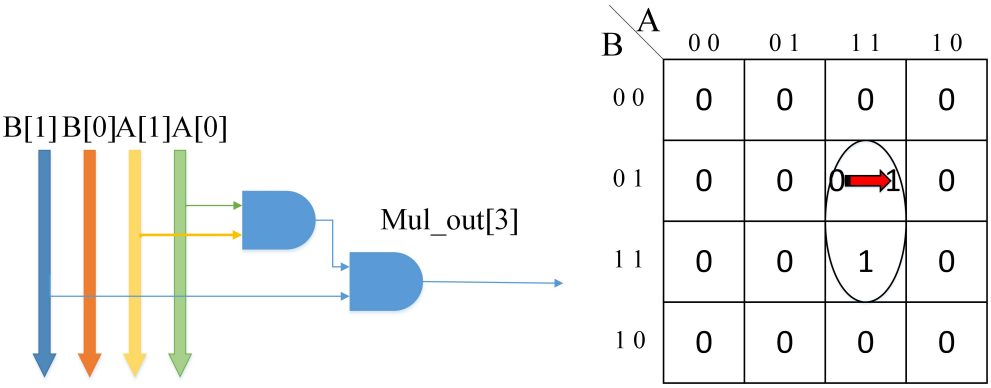


Figure 3.12: *Approximate Design for Mul[3]*

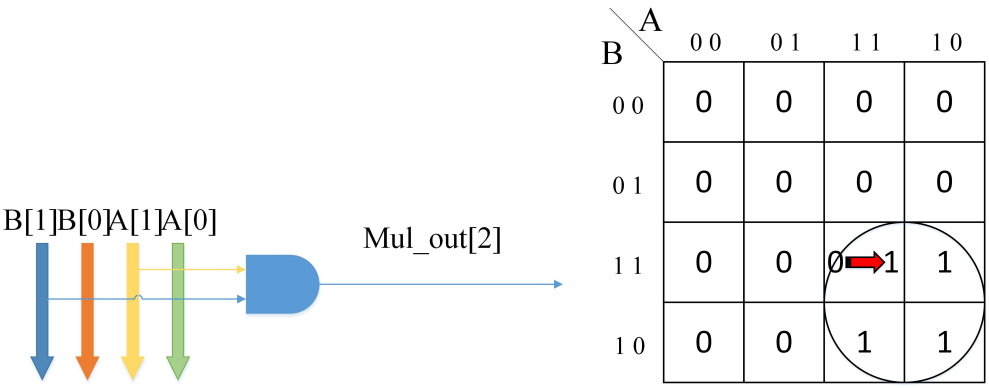


Figure 3.13: *Approximate Design for Mul[2]*

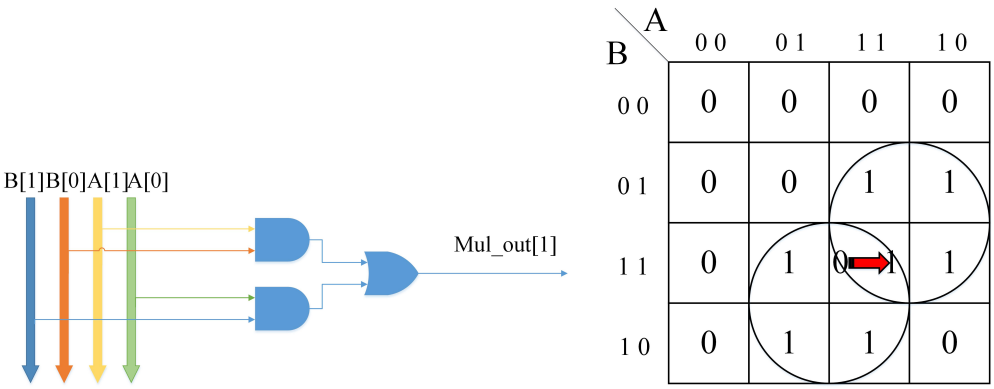


Figure 3.14: *Approximate Design for Mul[1]*

Finally, the Mul[1] is simplifed as the boolean expression below with changing 0 to 1 for the case of A[1:0]=11 and B[1:0]=11.

$$Mulout[1](AP3) = A[1]B[0] + B[1]A[0]. \tag{3.20}$$

Comparing to the exact design on Mul[1] shown in Fig. 3.11(b), the approximate design shown in Fig. 3.14 significantly reduces the gate count by 72.7% with one bit error tallerance.

In conclusion, we present three different approximations of 2×2-bit multiplier design combining the three approximate bit computations. In Eq. 3.17, Mul[1] is replaced by AP_Mul[1] as the first approximation (AP#1), Mul[2:1] is substituted with AP_Mul[2:1] as the second approximation (AP#2), and Mul[3:1] is replaced by with AP_Mul[3:1] as the third approximation (AP#3). Therefore, the total gate counts for the exact design and AP#1 AP#3 designs can be summarized in Table 3.1. It can be observed that the resource cost is saved by 40%, 60%, 65%, respectively, for AP#1, AP#2, AP#3, compared with the exact design, leading to a significant slice and energy saving by employing inexact computing. Notice that the combinational design on FPGA is based on LUT not logic gate, so the results might be a little difference, which is proved in Section 3.9.

Designs	Hardware cost
EX MUL	16 AND + 4 OR
AP#1 MUL	10 AND + 2 OR
AP#2 MUL	7 AND + 1 OR
AP#3 MUL	6 AND + 1 OR

Table 3.1: *Gate Cost of 2 × 2 bit Multiplier*

3.7.0.4 Approximate Adder

Generally, the multi-bit adders can be simply integrated with several single-bit adders. The exact single-bit adder can be expressed as

$$Cout = (ab) + (bc) + (ac) \tag{3.21a}$$

$$Sum = (abc) + (abc) + (abc) + (abc) \tag{3.21b}$$

Likewise, two approximate single-bit adders can be rewritten as

$$AP1Cout = a + (bc) \tag{3.22a}$$

$$AP1Sum = (abc) + (abc) + (abc) \tag{3.22b}$$

and

$$AP1Cout = a \tag{3.23a}$$

$$AP1Sum = (ac) + (bc) + (a'c) \tag{3.23b}$$

where a and b are the two inputs, and c is the carry in bit. AP1cout and AP1sum are the carry out bit and summation bit for the first approximate adder, and AP2cout and AP2sum are bits the second approximate adder. The static analysis of the approximate adder is shown in Table 3.2. Compared to the exact adder design, it

Designs	Hardware cost
EX Adder	7 AND + 2 OR
AP#1 Adder	4 AND + 2 OR
AP#2 Adder	3 AND + 1 OR

Table 3.2: *Single-Bits Adder Resource Cost*

can be observed that the gate counts are saved by 77.8% and 44.4%, respectively, for using AP#1 and AP#2.

3.8 Static Evaluation

This subsection evaluates the quality of the results by using exact design and approximate implementations. In Fig. 3.15, six grayscales image converted by different approximations of RGB2Grayscale designs are depicted. Fig. 3.15(a) shows the quality of result using the exact design. Fig. 3.15(b), Fig. 3.15(c) and Fig. 3.15(d) depict the results employing the AP#1, AP#2, and AP#3 multipliers respectively. And Fig. 3.15(e)and Fig. 3.15(f) show the results with AP#1 and AP#2 adders respectively. It is not clear to see the difference between images in Fig. 3.15 by human eyes, so the error rate is further graphed in Fig. 3.16 As depicted in Fig. 3.16(a), the horizontal axis represents the different approximates of the multipliers, and the vertical axis indicates the error rate for each specific implementation. It can be observed that the error rates for converting the color image into grayscale image are 0.999%, 3.243%, and 5.64%, respectively, by using AP#1, AP#2, and AP#3 multipliers. In Fig. 3.16(b), the error rate decreases by replacing less number of addition bits, which is represented by the horizontal axis from the third bit (3b) to the least bit (1b). It is obvious that the higher bits have more error effect compared to the lower bits. To keep the error tolerance acceptable (less than 4%), therefore, only the least three bits are considered in this benchmark. When replacing all the three bits with AP#1 and

AP#2 adders, the error rates are 1.43% and 2.85%, respectively. The error rates drop to around 1% with replacing the least signifiant two bits for both AP#1 and AP#2 adders, and the error rates are less than 1% when replacing the least significant bit.

3.9 FPGA Implementation and Simulation

In this subsection, first the register transfer level (RTL) design and verification with Verilog hardware description language (HDL) is discussed. The Mentor Graphic ModelSim is used as the simulator, and the Xilinx Vivado 2018 with the target device Nexys-4 is employed as the synthesis tool. In our work, the FPGA performance evaluation methodology is adopted to estimate the system performance in terms of slice count, speed, and power dissipation [35].

3.9.0.1 FPGA Design Flow

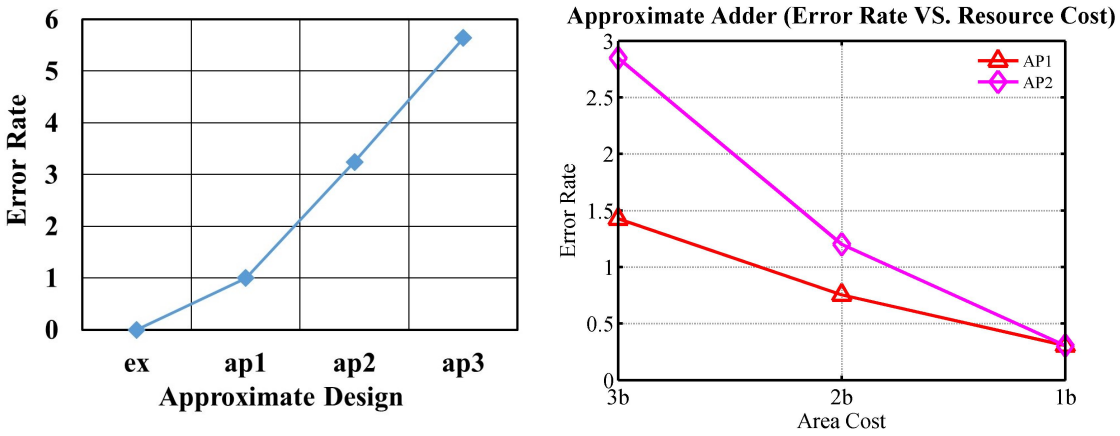
After simulation, the toggle activities of signals, IOs, and logic are collected by Value Changed Dump (VCD) files. And then after synthesis, the practical results are summarized in the third and fourth columns of Table 3. It can be observed that with the same adder design, the higher approximations of the multiplier implementation, the less number of the slices are needed. Similarly, when using the same multiplier, higher approximation of adders consume less number of FPGA slices. The MOF is decided by the critical path delay. However, since the combinational circuit mapping on FPGA is based on the Look-Up-Table (LUT), the critical paths for all the approximations of implementations are the same, resulting in the same MOF as 271.326 MHz. Finally, we use XPower Analyzer to estimate the realistic power consumption. Xilinx Power Analyzer evaluates the power with the Native Circuit Description (NCD) file generated by ISE and the specific simulation VCD file. As the power con-

(a) Exact (b) AP#1 Mul

(c) AP#2 Mul (d) AP#3 Mul

(e) AP#1 Add (f) AP#2 Add

Figure 3.15: *Approximate multiplier simulation result*



(a) AP Mul (b) AP Add

Figure 3.16: *Error rate VS. Approximations*

sumption shown in the sixth column in Table 3, the dynamic power decreases with the increasing of approximations of the adders or multipliers. Some of the power consumptions are the same because the toggle rates of slices are similar to each other in this benchmark. By simply multiplying dynamic power by the reciprocal of MOF, the dynamic energy is computed in the seventh column. Since the MOF are the same for all the approximate designs, the dynamic energy dissipations have the same trend of the power cost.

3.9.0.2 FPGA Slice-Energy cost

In order to find the optimal cost saving in terms of slice number and energy consumption corresponding to the specific quality bound, in this subsection we present a novel performance metric, denoted as slice-energy cost, as below

$$Slice - Energy = S^x \times E^y. \tag{3.24}$$

where 'S' and 'E' represent the FPGA cost in terms of slice count and energy dissipation. 'x' is the weight of slice count, and 'y' is the weight of FPGA energy consumption. The weights x and y are between 0 and 1, and the summation of the

DUT No.	AP Adder	AP Mul	Slice of Regsiter	Slice of LUT	DP(mW)	DE(pJ)
1	Ex	Ex	700	879	13	4.79
2	Ex	Ap1	684	843	12	4.42
3	Ex	Ap2	652	799	12	4.42
4	Ex	Ap3	640	783	12	4.42
5	Ap1	Ex	698	874	13	4.79
6	Ap1	Ap1	682	838	12	4.42
7	Ap1	Ap2	650	794	12	4.42
8	Ap1	Ap3	638	778	12	4.42
9	Ap2	Ex	538	720	10	3.69
10	Ap2	Ap1	522	692	10	3.69
11	Ap2	Ap2	506	672	10	3.69
12	Ap2	Ap3	494	656	10	3.69

Table 3.3: *Slice count and dynamic power and energy*

weights is considered as a constant value equal to 1, in order to decide the tradeoff between cost savings. By varying x and y, we can target the performance on a specific application. For example, setting x = y = 1/2 leads to equal weighting, x = 1 targets the small-size design, and y = 1 targets the low-energy optimization. Generally, the slice-energy cost saving should be minimized in order to find the optimal design with different weight configurations. As an example for equally setting the two weights as 1/2, the minimum slice-energy cost occurs at the No. 12 design with the highest approximation of multiplier (AP#3) and the highest approximation of adder (AP#2) as shown in Fig. 3.17.

3.10 Summary

In this paper, twelve approximations of RGB2Grayscale converters are implemented on an FPGA platform, by proposing and integrating three approximations of multipliers, two approximations of adders, along with exact designs as well. The implementations shown on an FPGA demo provide a wide range of slice-energy saving corresponding to different quality constrains. By a 5.69% quality decreasing for

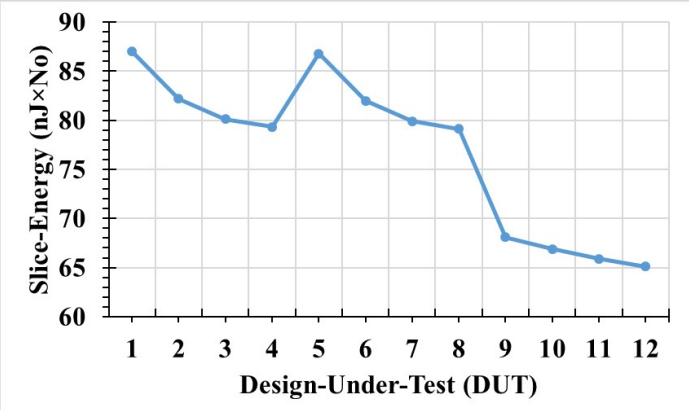


Figure 3.17: *Approximate No.4 multiplier structure*

multiplication and 2.85% for addition, the dynamic energy can be reduced to 77.04% and the slice count can be saved to 72.83% compared to the exact design. Our future work is to implement the face detection algorithm with many approximations, in order to speed up the system and reduce the energy consumption.

CHAPTER 4

CONCLUSIONS AND FUTURE WORK

In this chapter, we summarize our contributions presented in this dissertation. We then discuss the possible directions for our future research work.

4.1 Summary

This chapter presents a scalable image/video processing platform on FPGAs containing not only open source design code but also a verification environment. The power consumption and slice count of our work is significantly reduced when compared to the prior works. The most important here is that the reusable and the expandable to a diverse range of algorithm in image processing and computer vision of this platform.

More important, twelve approximations of RGB2Grayscale converters are implemented on an FPGA platform, by proposing and integrating three approximations of multipliers, two approximations of adders, along with exact designs as well. The implementations shown on an FPGA demo provide a wide range of slice-energy saving corresponding to different quality constrains. By a 5.69% quality decreasing for multiplication and 2.85% for addition, the dynamic energy can be reduced to 77.04% and the slice count can be saved to 72.83% compared to the exact design.

4.2 Future Work

Consider of the reusable and expandable of the platform proposed in this dissertation, in future, we can extend our platform and approximate library on a variety of algorithms and systems to explore the energy tradeoff. Such as FPGA-IoTmesh

system, FPGA-video processing system, or FPGA-deep learning system. We expect our work will lead to multiple designs and give some contribution on research and education of this area.

BIBLIOGRAPHY

[1] C. Ababei and et al. Open source digital camera on field programmable gate arrays. *Intl. Journal of Handheld Computing Research (IJHCR)*, 7(4):30–40, 2016.

[2] M. Birla. Fpga based reconfigurable platform for complex image processing. *2006 IEEE Intl. Conf. on Electro/Information Technology*, pages 204–209, May 2006.

[3] J. Bornholt, T. Mytkowicz, and K. S. McKinley. Uncertain: A first order type for uncertain data. *Proceedings of the 19th Intl. Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 51–66, 2014.

[4] J. Bornholt, T. Mytkowicz, and K. S. McKinley. Uncertain: Abstractions for uncertain hardware and software. *IEEE Micro*, 35(3):132–143, 2015.

[5] V. Chippa, S. Chakradhar, and K. Roy. Analysis and characterization of inherent application resilience for approximate computing. *ACM/EDAC/IEEE 50rd Design Automation Conference (DAC)*, 2013.

[6] H. Esmaeilzadeh, A. Sampson, and L. Ceze. Architecture support for disciplined approximate programming. *ACM SIGPLAN Notices- ASPLOS '2012*, 47(4): 301–312, 2012.

[7] M. Fan, Q. Han, and X. Yang. Energy minimization for on-line real-time scheduling with reliability awareness. *Elsevier Journal of Systems and Software (JSS)*, 127:168–176, May 2017. doi: 10.1016/j.jss.2017.02.004.

[8] Mike Field. Zedboard ov7670. `http://hamsterworks.co.nz/mediawiki/index.php/Zedboard_OV7670`.

[9] A. Gajjar and et al. An fpga synthesis of face detection algorithm using haar classifiers. *Intl. Conf. on Algorithms Computing and Systems (ICACS2018)*, pages 133–137, July 2018.

[10] A. Gajjar, Y. Zhang, and X. Yang. Demo abstract: A smart building system integrated with an edge computing algorithm and iot mesh networks. *The Second ACM/IEEE Symposium on Edge Computing (SEC)*, 35, 2017.

[11] A. Gajjar, X. Yang, and et. al. Mesh-iot based system for large-scale environment. *5th Annual Conf. on Computational Science and Computational Intelligence (CSCI2018)*, 2018.

[12] H. He and et al. Dual long short-term memory networks for sub-character representation learning. *The 15th Intl. Conf. on Information Technology-New Generations (ITNG-2018)*, Jan 2018.

[13] H. He, L. Wu, X. Yang, and et al. Dual long short-term memory networks for sub-character representation learning. *The 15th Intl. Conference on Information Technology-New Generations (ITNG)*, pages 1–6, Jan 2018.

[14] J. Huang, J. Lach, and G. Robins. A methodology for energy-quality tradeoff using imprecise hardware. *DAC 2012*, pages 504–509, 2012.

[15] J.Han and M. Orshansky. Approximate computing: An emerging paradigm for energy-efficient design. *IEEE ETS*, 2013.

[16] K. Jin and et al. High-speed fpga-gpu processing for 3d-oct imaging. *Intl. Conf. on Computer and Communications (ICCC)*, pages 2085–2088, March 2018.

[17] A. Kahng and S. Kang. Accuracy-configurable adder for approximate arithmetic designs. *The 49th Design Automation Conference (DAC2012)*, pages 820–825, 2012. doi: 10.1145/2228360.2228509.

[18] P. Kulkarni, P. Gupta, and M. Ercegovac. Trading accuracy for power with an underdesigned multiplier architecture. *24th IEEE Intl. Conf. on VLSI Design*, pages 346–351, 2011.

[19] J. Liang, J. Han, and F. Lombardi. New metrics for the reliability of approximate and probabilistic adders. *IEEE Transactions on Computers*, 62(9):1760–1771, 2013.

[20] S. Lu. Speeding up processing with approximation circuits. *Computer*, 37(3): 67–73, 2004.

[21] J. Miao, K. He, A. Gerstlauer, and M. Orshansky. Modeling and synthesis of quality-energy optimal approximate adders. *ICCAD 2012*, pages 728–735, 2012.

[22] S. Misailovic, M. Carbin, S. Achour, and Z. Qi. Chisel: Reliability- and accuracy-aware optimization of approximate computational kernels. *Proceedings of the 2014 ACM Intl Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA)*, pages 309–328, 2014.

[23] A. K. Mishra, R. Barik, and S. Paul. iact: A software-hardware framework for understanding the scope of approximate computing. *Workshop on Approximate Computing Across the System Stack (WACAS)*, 2014.

[24] R. Nair. Big data needs approximate computing: Technical perspective. *ACM Communications*, (1):58–104, 2015.

[25] L. Nwosu and et al. Deep convolutional neural network for facial expression recognition using facial parts. *15th IEEE Intl Conf. on Dependable Autonomic and Secure Computing*, Feb 2018.

[26] *OV7670/OV7171 CMOS VGA(640X480) CAMERACHIP with OmniPixel Technology.* OmniVision, 7 2005. Version 1.01.

[27] G. Pekhimenko, D. Koutra, and K. Qian. Approximate computing: Application analysis and hardware design, 2010. www.cs.cmu.edu/ gpekhime/Projects/15740/paper.pdf.

[28] S. Rahangdale and et al. MBSEM image acquisition and image processing in LabView FPGA. *2016 Intl. Conf. on Systems, Signals and Image Processing (IWSSIP)*, pages 1–4, July 2016.

[29] M. Shafique, R. Hafiz, and S. Rehman. Cross-layer approximate computing: From logic to architectures. *ACM/EDAC/IEEE 53rd Design Automation Conference (DAC)*, 2016.

[30] J. Thota, P. Vangali, and X. Yang. Prototyping an autonomous eye-controlled system (AECS) using raspberry-pi on wheelchairs. *Intl. Journal of Compt. Applications (IJCA)*, 158(8):1–7, 2017.

[31] P. Vangali and X. Yang. A compression algorithm design and simulation for processing large volumes of data from wireless sensor networks. *Communications on Applied Electronics (CAE)*, 7(4):1–5, June 2017.

[32] X.Yang and X.He. Demo abstract: Establishing a BLE mesh network with fabricated csrmesh devices. *The Second ACM/IEEE Symposium on Edge Computing (SEC)*, 34, July 2017.

[33] X. Yang and J. Andrian. A high performance on-chip bus (MSBUS) design and verification. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst. (TVLSI)*, 23(7): 1350–1354, July 2015.

[34] X. Yang and J. Andrian. An advanced bus architecture for aes-encrypted high-performance embedded systems. *US20170302438A1*, 2017.

[35] X. Yang and et al. A novel bus transfer mode: Block transfer and a performance evaluation methodology. *Elsevier Integration the VLSI Journal*, 53:23–33, Jan 2016.

[36] X. Yang and et. al. A vision of fog systems with integrating fpgas and ble mesh network. *Journal of Communications*, 2018.

[37] X. Yang and W. Wen. Design of a pre-scheduled data bus (DBUS) for advanced encryption standard (AES) encrypted system-on-chips (socs). *The 22nd Asia and South Pacific Design Automation Conference (ASP-DAC 2017)*, pages 1–6, Feb 2017. doi: 10.1109/ASPDAC.2017.7858373.

[38] X. Yang and N. Wu. Design of a bio-feedback digital system (bfs) using 33-step training table for cardio equipment. *The 8th Intl. Conference on Applied Human Factors and Ergonomics (AHFE 2017)*, 603:53–64, June 2017.

[39] X. Yang, X. Niu, and J. Fan. Mixed-signal system-on-chip (soc) verification based on system verilog model. *The 45th Southeastern Symposium on System Theory (SSST 2013)*, pages 17–21, March 2013.

[40] X. Yang, N. Wu, and J. Andrian. Comparative power analysis of an adaptive bus encoding method on the MBUS structure. *Journal of VLSI Design*, 2017: 1–7, May 2017. doi: 10.1155/2017/4914301.

[41] X. Yang, W. Wen, and M. Fan. Improving AES core performance via an advanced IBUS protocol. *ACM Journal on Emerging Technologies in Computing (ACM JETC)*, 14(1):61–63, Jan 2018. doi: 10.1145/3110713.

[42] K. Muntimadugu et al Z. Kedem, V. Mooney. Optimizing energy to minimize errors in dataflow graphs using approximate adders. *The 2010 Intl. Conference on Compilers, Architectures and Synthesis for Embedded Systems*, pages 177–186, 2010. doi: 10.1145/1878921.1878948.

[43] K. Zeng, N. Wu, X. Yang, and K. K. Yen. Fhcc: A soft hierarchical clustering approach for collaborative filtering recommendation. *Intl. Journal of Data Mining & Knowledge Management Process (IJDKP)*, 6(3), May 2016.

[44] Y. Zhang, X. Yang, L. Wu, and et al. Hierarchical synthesis of approximate multiplier design for field-programmable gate arrays (FPGA)-CSRmesh system. *Intl. Journal of Compt. Applications (IJCA)*, 180(17):1–7, Feb 2018. doi: 10.5120/ijca2018916380.

VITA

Yunxiang Zhang

2015	B.S., Electrical Engineering University of Tennessee, Martin Martin, TN
2018	M.S., Computer Engineering University of Houston Clear Lake Houston, TX

PUBLICATIONS

Y. Zhang, X. Yang, L. Wu, K. Sha, et. al., "Exploring Slice-Energy Saving on An Video Processing FPGA Platform with Approximate Computing" Intl. Conference on Algorithms, Compting and Systems (ICACS2018), PP. 138-143 , 2018.

Y. Zhang, X. Yang, etc., "Hierarchical Synthesis of Approximate Multiplier Design for Field-Programmable Gate Arrays (FPGA)-CSRmesh System," Intl. Journal of Compt. Applications (IJCA), Vol. 180, No. 17 PP. 1-7, Feb. 2018 %doi> 10.5120/ijca2018916380

X. Yang, **Y. Zhang**, et. al., "A Scalable Image/Video Processing Platform with Open Source Design and Verification Environment," 20th Intl. Symposium on Quality Electronic Design (ISQED 2019) , Under Review, 2018.

A. Gajjar, X. Yang , **Y. Zhang**, et. al.,"An FPGA Synthesis of Face Detection Algorithm using HAAR Classifiers," Intl. Conference on Algorithms, Computing and Systems (ICACS 2018), Under Review, 2018.

X. Yang, **Y. Zhang**, W. Wen, and M. Fan, "A Case Study of Self-Organization Algorithms for High-Efficiency System-on-Chips Integration," IEEE Intl. Conf. on Autonomic Computing (ICAC 2017) – Workshop on Feedback Computing, Accepted, In Press, July, 2017.

A. Gajjar, **Y. Zhang**, and X. Yang, "A Smart Building System Integrated with An Edge Computing Algorithm and IoT Mesh Networks," The Second ACM/IEEE Symposium on Edge Computing (SEC 2017), Article No. 35, Oct. 2017 %doi >10.1145/3132211.3132462

Y. Zhang, and X. Yang, "Exploring Approximate Designs for FPGA-Based Edge Computing," Houston Robotics & AI Day, Houston, TX, US, July 2017.

Y. Zhang, and X. Yang, "A Novel Fog Computing Acceleration Method: Approximate FPGA Design on Computation Components," 2017 Innovation/Automation Dual Conference, Houston, TX, US, Oct 2017.

X. Yang, **Y. Zhang**, et. al., "Learning-on-Chip: Facial Detection with Approximations of FPGA Computing," 2018 Robotics & AI Day, UHCL, Aug. 03, 2018.