3-2008

Hardware Algorithm Implementation for Mission Specific Processing

Jason W. Shirley

Hardware Algorithm Implementation For Mission Specific Processing

THESIS

Jason W. Shirley, Captain, USAF

AFIT/GE/ENG/08-27

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the United States Government.

AFIT/GE/ENG/08-27

Hardware Algorithm Implementation For Mission Specific Processing

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Electrical Engineering

Jason W. Shirley, B.S.E.E.

Captain, USAF

March 2008

AFIT/GE/ENG/08-27

Hardware Algorithm Implementation For Mission Specific Processing

Jason W. Shirley, B.S.E.E.
Captain, USAF

Approved:

/signed/	27 Feb 2008
Yong C. Kim, PhD (Chairman)	date
/signed/	27 Feb 2008
LaVern A. Starman, PhD, Maj (Member)	date
/signed/	27 Feb 2008
Guna S. Seetharaman, PhD (Member)	date
/signed/	27 Feb 2008
Juan R. Vasquez, PhD (Member)	date

Abstract

There is a need to expedite the process of designing military hardware to stay ahead of the adversary. The core of this project was to build reusable, synthesizeable libraries to make this a possibility. In order to build these libraries, `Matlab`® commands and functions, such as Conv2, Round, Floor, Pinv, etc., had to be converted into reusable VHDL modules. These modules make up reusable libraries for the Mission Specific Process (MSP) which will support AFRL/RY.

The MSP allows the VLSI design process to be completed in a mere matter of days or months using an FPGA or ASIC design, as opposed to the current way of developing a system which can take 1-2 years to complete. By having the libraries built, the components can be implemented in an FPGA or ASIC design over and over again. The libraries make it possible to make upgrades to weapons systems to meet the ever-changing needs the War Fighter faces. MSP makes it possible to develop various algorithms, including algorithms implemented in `Matlab`® . The MSP libraries were built and tested using `TSMC 250-nm`®technology library from the Taiwan Semiconductor Manufacturing Company. They were also synthesized for an FPGA. The modules were all synthesized using the CAD tools from `Cadence`®and `Mentor Graphics`®. Power, area, and delay results for each module were presented.

Acknowledgements

First and foremost, I would like to thank my Lord and Savior Jesus Christ for giving me the wisdom and knowledge to complete my thesis. I also owe a large debt of gratitude to my wife and daughter for their understanding and patience during my research at the Air Force Institute of Technology (AFIT). I would like to thank Dr. Yong Kim for his wisdom, knowledge, and helpfulness during my research as he guided and encouraged me throughout my research process. I would like to thank Dr. Juan Vasquez and Douglas Summers-Stay for sponsoring me, and I hope that the research that I have gathered is useful to the Air Force Research Lab (AFRL) AFRL/RY. Finally, I'd like to thank my fellow students for all of their encouragement and knowledge during my research efforts: Sam Stone, Benjamin Hancey, Eric Lam, James Crouch, and Gene Hockenberry.

Jason W. Shirley

Table of Contents

Page

Page

List of Figures

Figure Page

List of Tables

List of Abbreviations

Abbreviation		Page
AFIT	Air Force Institute of Technology	v
AFRL	Air Force Research Lab	v
VLSI	Very Large-Scale Integration	1
MSP	Mission Specific Process	1
VHDL	VHSIC Hardware Description Language	1
FPGA	Field Programmable Gate Array	3
ASIC	Application Specific Integrated Circuits	3
TSMC	Taiwan Semiconductor Manufacturing Company	4
Conv2	Two Dimensional Convolution	4
Pinv	Pseudoinverse	4
RTL	Register Transfer Level	5
DoD	Department of Defense	11
SRAM	Static Random Access Memory	13
CLB	Configurable Logic Block	44
DSP	Digital Signal Process	52
DPM	Dynamic Power Management	54
DC	Direct Current	55
AC	Alternating Current	55
Vp	Peak Voltage	55
Vth	Threshold Voltage	56
P	Power	56
V	Voltage	56
I	Current	56

Hardware Algorithm Implementation For Mission Specific Processing

I. Introduction

In today's advancing technologies, devices are shrinking and densities of integrations are increasing. With these trends come many new challenges in designing integrated electronic circuits and systems. To achieve high performance (Power, Speed, Dynamic Range, etc.) in new integrated circuits for next-generation systems; new methodologies must be created, adopted, and executed.

The objective of this research is to examine and investigate current difficulties/problems associated with modeling and fabricating Very Large-Scale Integration (VLSI) circuits, then provide reusable library cells for AFRL/RY directorate's Mission Specific Process (MSP) in Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL). These reusable libraries can be broken into three variations which are as follows: optimizing power, minimizing area, and minimizing delay. By having these libraries built and ready to go, system requirements can be upgraded and changed in a matter of days instead of months or even years. These reusable libraries make it possible to meet the changing requirements of the operational environment. The MSP makes it flexible for system changes to be implemented quickly into a system. The variations between power, area, and delay can be generalized as the points on an equilateral triangle as seen in Figure 1.1. The points of the triangle represent the priority optimal design for the given circuit. For example, if you want to optimize the power, the other two parameters will not be the priority. This will cause the design to have minimal power consumption at the expense of the final de-

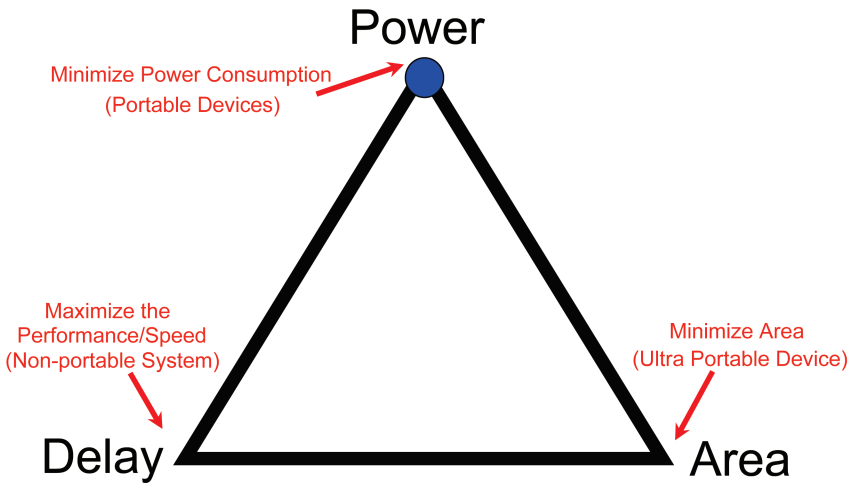


Figure 1.1: Power, Area, and Delay Triangle.

sign being large or slow. The blue dot in the triangle represents where the priority lies, meeting design specifications.

The idea behind AFRL/RY directorate's MSP is a threefold approach to designing circuits. First, it takes time for the design to be developed and fabricated before it can get into the hands of the customer. The design to market can take up to 1-2 years depending on the technology and the level of difficultly of the circuit design. In addition to the amount of time it takes to develop the product, it can be obsolete in only a few years. This is what makes MSP so unique; it is made up of pre-built synthesizable reusable libraries so the designer doesn't have to start from scratch. The designer can quickly use the pre-built libraries and make new ones when specifications change and add them to the library.

Second, if modifications are required for additional features to be added to the product, then you have to wait for the redesign to take place. You have to pay the vendor again for the changes they make and wait for the new design to be delivered. This is where MSP comes in to reduce the cost and schedule of redesigning a circuit just to make a few changes for additional specifications. The use of the reusable libraries makes designing a circuit much easier than current methodologies. In a

sense, it is like putting a Lego®set together. By having all the pre-built parts it's just a matter of integrating them together for your personal application.

Thirdly, a Field Programmable Gate Array (FPGA) can replace an Application Specific Integrated Circuit (ASIC) design, unless you want absolute performance. The use of MSP makes it flexible for the designer to develop prototypes faster and cheaper for an FPGA as opposed to an ASIC design. Having MSP in your tool box can save time and money for the next generation of a circuit design. Therefore, the War Fighter will have new equipment in the field in a matter of days as opposed to the old way of doing business, which could take 1-2 years for a weapons system to be developed.

This project will support AFRL/RY in their development of a target tracking project, where the circuits and digital circuits are to be implemented in one chip. The goals are to provide reliable building blocks as portable synthesizable reusable libraries. These libraries will enable the War Fighter to get an upgradable weapon system in a matter of a few days or hours and to keep up-to-date with the ever-changing Global War on Terrorism.

Building these reusable libraries and making them usable for FPGA or ASIC designs is beneficial in many ways. For instance, programming the design on an FPGA makes it possible to protect the design against enemy hands. Anti-tamper methods on the FPGA board protects the design from being discovered by the enemy. Additionally, placing the designs on an FPGA allows the weapons system to be implemented quicker than sending the design to a foundry for fabrication as an ASIC design, which can take months due to long lead times that the foundries put in their schedule. Once the design is created on an ASIC chip it has to be thoroughly tested to find faults in the fabrication process. All these design steps take time, especially if there are problems with the completed design.

1.1 Specific Issue

The War on Terrorism has made the military solider depend on today's technology of global positioning systems, radar systems, and different communication devices. These devices, being mobile or not, are required for them to operate in the field and communicate with Command and Control. For this reason it is essential that these libraries be built and perfected.

With any circuit design there are three key parmeters that designers face when designing circuits, and they are as follows: power, area, and delay. The designer has to make tradeoffs between these three parmeters to meet their specific design constraints. Every commercial or military application has its own specifications for power, area, and delay.

The VLSI technology continues to place more and more transistors on a single chip. This allows the chips to become more powerful in computing power as the area of the chips remain small. The chips constantly require electrical power to keep them operational which makes it difficult for the War Fighter to do their mission without wondering if their batteries are going to sustain throughout their mission. There is a need to be able to run longer missions and have longer lasting equipment that doesn't require battery change-outs in the middle of a critical mission. This low power optimization will be using a 250-nm technology library [15] from the Taiwan Semiconductor Manufacturing Company (TSMC). This library will provide a starting point to develop circuits that have lower power consumption for the future.

1.2 Problem Statement

The problem is to take the AFRL/RY Optical Flow Dense Algorithm written in `Matlab`® and convert the commands into synthesizable reusable library modules written in VHDL. We will be laying the foundation with these synthesizable reusable libraries for other weapons systems that require `Matlab`® commands such as, Round, Floor, Two Dimensional Convolution (Conv2), and Pseudoinverse (Pinv), etc. The

Figure 1.2: A Sample of Target Tracking Imagery [1].

Optical Flow Dense Algorithm can provide various libraries to handle Unmanned Aerial Vehicles for image processing. An example of target tracking imagery can be seen in [1] Figure 1.2. The small rectangles show targets that can potentially be tracked using Optical Flow.

1.3 Scope and Assumptions

It is assumed that the reader has knowledge about the VLSI technology and understands VHDL, scripting, and intergrating the script in `Cadence`®software program or `Modelsim`®. The main software programs that will be used for this research are `Modelsim`®, `Mentor Graphics`®, and `Cadence`®Companies' software tools. The simulations will be run on `Modelsim`®to verify the Register Transfer Level(RTL) coding. `Cadence`®and `Mentor Graphics`®software tools will be used to verify that the modules are synthesizable.

1.4 Thesis Organization

Chapter 2 of this thesis will give background information required to understand the technology options that are available for use to reduce power consumption. Each option will be briefly explained and the main focus of the research project option will be expounded into further details and discussions. Also, background information will

be given to support the design decisions used in Chapter 3. Chapter 3 will discuss the theory and methods that were used for this thesis project. Chapter 4 will look at the results that were gathered throughout this research and will be analyzed and discussed. Finally, Chapter 5 will discuss future work and topics. The `Modelsim`®VHDL code will be located in Appendix C.

II. Background

This chapter will give an overview of the background information used throughout this research. The MSP idea makes it flexible enough to accommodate the ever-changing Air Force missions. This section will present an overview of the VLSI Design Process as well as a more in depth view of the process. We will also discuss FPGA vs. ASIC Risk, MSP Design Reusability, and Optical Flow.

2.1 Overview of VLSI Design Process

The VLSI Design Process can be summed up in the following three main steps: architecture, verification, and implementation and can be seen in Figure 2.1. The architecture is made up of three methods to design a circuit which are power, area, and delay (speed). Each architecture has it pros and cons for design implementation depending on what you are trying to achieve. The use of the circuit will drive what architecture you should use when designing it.

It is important to verify that your expected results match your simulated results. Once you have determined your design is working properly through simulations it is time to implement your design on an FPGA or fabricated circuit such as an ASIC.

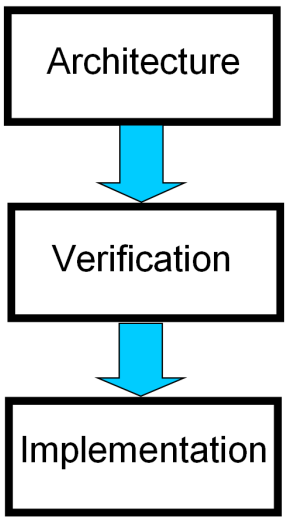


Figure 2.1: Three Main Steps for VLSI Design Process

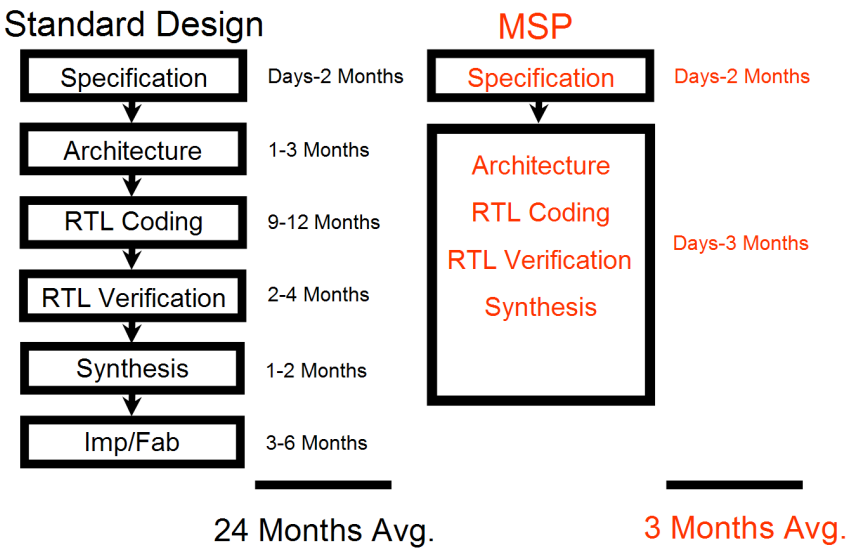


Figure 2.2: Six Steps for VLSI Design Process

The fabrication process for a circuit can take anywhere from 3 to 6 months at the foundry. In the next section, the VLSI Design Process will be broken down for a more magnified look at the steps involved to build an operational design.

2.2 VLSI Design Process

The typical VLSI Design Process can be broken into the following six steps and can be seen in Figure 2.2:

1. Specification
2. Architecture
3. RTL Coding
4. RTL Verification
5. Synthesis
6. Implementation/Fabrication

Each of these steps will be discussed further for an understanding of what is required before a design can be implemented into the field as an operational weapons system.

2.2.1 Specification. The specification for any design comes from the customer who has a specific need for a project they want built and implemented. There are three different design strategies for circuits - custom, ASIC, and FPGA. The customer lists the criteria they want for their system, which could be a specific size (area) of the circuit, a certain power usage, or delay (speed) of the design, or even that they want the design written in VHDL. The vendor (designer) will discuss with the customer the possibilities based on the technology available at the time the original specification was created.

2.2.2 Architecture. There are three different architectures that can be implemented in any design. The three architectures are power, area, and delay (speed).

2.2.2.1 Power. Clock control plays a major role when designing a circuit to reduce power consumption. Reducing the speed of the clock for a circuit will reduce the switching activity. This results in power savings by limiting the amount of switching activity that takes place. Besides reducing the speed of the clock for the circuit, designers have also proposed "clock gating [13] by modifying the design of the existing energy recovery clocked flip-flops to incorporate a power saving feature that eliminates any energy loss on the internal clock and other nodes of the flip-flops." According, to Steve Kilts, in his book, Advanced FPGA Design, he suggests the following:

> The most effective and widely used technique for lowering the dynamic power dissipation in synchronous digital circuits is to dynamically disable the clock in specific regions that do not need to be active at particular stages in the data flow [9].

It would be ideal to have the circuit temporarily turn the clock off when a particular section of the circuit is not required to reduce power consumption. These are a few techniques that are available for use to reduce power consumption. Additional power reduction techniques can be found in Appendix A.

2.2.2.2 Area. Another architecture that can be used when developing a circuit design is area. Ways to implement reduction in area for a circuit depends on picking the correct topology. Topology that focus on reducing the area size of a circuit can be attained by reusing [9] "the logic resources to the greatest extent possible, often at the expense of throughput(speed)." When you want to increase the delay for a design you need to pipeline your design. However, to reduce the area you need to do the opposite of creating pipelines, you need to roll them up to be able to use the available resources. Also, it is a good idea, according to Steve Kilts, to [9] "share logic resources between different functional operations."

2.2.2.3 Delay (Speed). The third architecture that can be used when designing a circuit is speed. There are three ways to describe speed in regards to circuit design - they are throughput, latency, and timing. In the book Advanced FPGA Design [9], Kilts gives a description for each one of these.

> A high-throughput design is one that is concerned with the steady-state data rate but less concerned about the time any specific piece of data requires to propagate through the design (latency). A low-latency design is one that passes the data from the input to the output as quickly as possible by minimizing processing delays. Timing refers to the clock speed of a design. The maximum delay between any two sequential elements in a design will determine the max clock speed [9].

A key factor that should be considered when designing a circuit for speed is to use pipelining wherever possible to increase your throughput. The idea of pipelining is very similar to how an assembly line works. Each member is continually performing their specific task; they finish one, pass it on to the next station, and get another. The result is that a completed product is continuously produced. Pipelining [9] is a way to increase throughput for a process. Another [9] technique is to have the system run things in parallel to speed up the process. This technique is most useful when doing math calculations in a design.

2.2.3 RTL Coding. The third step in the VLSI Design Process is RTL Coding. RTL Coding is nothing more than what language you will code your design in. There are different Hardware Description Languages (HDL) that can be used to make an RTL module. The most commonly used RTL modeling languages are the following:

1. VHDL
2. Verilog
3. System Verilog

VHDL is the adapted hardware description languages for the Department of Defense (DoD). For this thesis we will code our design in VHDL.

2.2.4 RTL Verification. Step four in the VLSI Design Process is RTL Verification. RTL Verification is when you verify that your VHDL code simulates correctly. The simulation waveform generated in `Modelsim`®gives you a waveform to show the circuit is functioning. If this waveform gives you the expected results then your code and RTL are working properly.

2.2.5 Synthesis/Manual Layout. A flexible synthesis tool, such as Leonardo Spectrum by `Mentor Graphics`®Tool, allows a synthesizable HDL design to be synthesized for both FPGA and ASIC. The software tool uses optimization algorithms to determine the best floor design, place, and route of the design. The synthesis step has made it possible for an FPGA to only need the design to be download to the FPGA board to verify whether the simulation results match the synthesis results and meet the specifications that were stated at the beginning of the VLSI Design Process. Unlike the FPGA design, after the ASIC design is synthesized, it still needs to be sent to a foundry to be fabricated. The use of FPGA eliminates the fabrication step, it is economically inexpensive to own and can be reconfigured for a new design in a matter of seconds, depending on the size of the design. Therefore, if there was a problem with the timing of the design and it didn't meet the specifications of the design, the

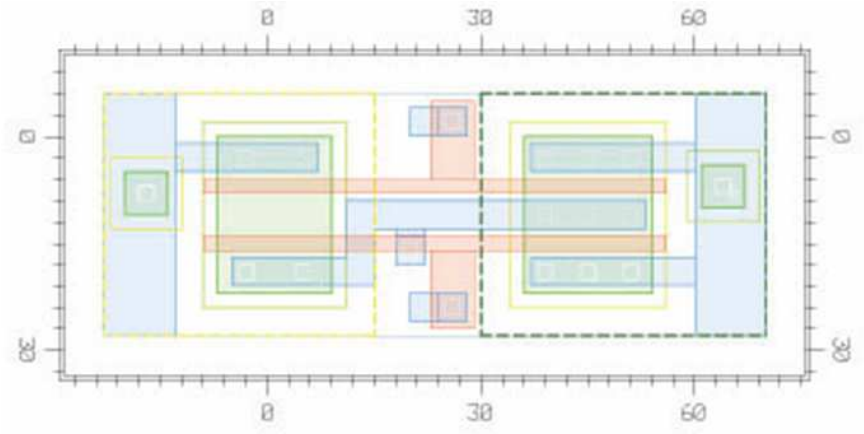


Figure 2.3: Nand Gate Layout

designer would be aware of the problem and begin to correct the timing issue. However, the ASIC design wouldn't find this problem until the design was tested after it was received from the foundry.

When developing a custom design, a manual layout could be used. This would require you to hand place every piece of the circuit in a location that is determined by you. Then you would have to route all the cell wires together. This is a very tedious and time consuming process. The designer has to worry about design rule violations such as placing the cells too close together or too far apart causing timing delays. An example of a handmade design of a Nand Gate is seen in Figure 2.3. This Nand Gate is only one gate. If you have a complicated design, in the range of hundreds of millions of gates, you can see how this task can be extremely time consuming. Once the gates have been layed out, then you have to determine if they are in the optimal position for the place and route step. This design lacks scalability when new technologies are created, the old layout may not be usable at all in ASICs. However, ASIC designs will have a higher absolute performance than FPGA designs.

2.2.6 Implementation/Fabrication. Implementation/Fabrication is the final step in the VLSI Design Process. Implementation/Fabrication consists of the design being synthesized for an ASIC design, otherwise it would be a manual layout design. The design is finalized and the circuit is sent on as a computer file to the foundry for

the fabrication process. At the foundry, several different metal layers and polysilicon layers will be used to create the design. The design will also need a set of test vectors to be sent with it to the foundry. This is required to determine if the design is working correctly. If the design comes back correct, then this ASIC or custom design can only be used for this given application. On the other hand, if the design comes back not working properly, then the designer will have to analyze the circuit and determine where in the six step process the design failed. This basically leaves the designer at square one. All of this time has been invested into a product that does not work. Even if the change is found within a few days it will still be 3 to 6 months from the time they send it to the foundry before they will see it again. Therefore, making an ASIC design is very costly and time consuming for the design process. Unlike the ASIC, the FPGA can be designed and implemented in an shorter time span and is an economical, reusable, and reconfiguable product.

2.3 FPGA vs. ASIC Risk

Now that the VLSI design cycle has been thoroughly discussed, let us examine the risk that is involved with building a circuit to meet an FPGA or ASIC design. The FPGA is an inexpensive design option that will only cost a few thousand dollars for a board. The Static Random Access Memory (SRAM) FPGA is reconfigurable in that once you put one design on the board you can take the program off the board and reprogram the board for a completely new design with no new costs involved. Once the design is synthesizable and thoroughly tested, it requires less time to become operational. There is also a higher chance that your FPGA design will synthesize and work the first time, unlike the ASIC design, saving a company time and money. On the other hand, an ASIC design is an expensive investment and will require substantially more time and money to be implemented. For example, let us say that a company is implementing a new design and they choose to take the route of an ASIC design. They could invest a million dollars into the program before it is even fabricated due to the labor intensive effort required. If the design fails once, it comes back from fabrication

to engineers who will thoroughly look at the design to understand why it failed. Once the failure has been determined, it may cost an additional $100,000 from the time the engineers determine the problem to the time it is refabricated and works. Also, this design can only be used for this single application. If this application was for a satellite that was in space and it failed, the satellite would lose that functionality in the system. However, if the design was on an FPGA, the design could be reprogrammed in a matter of minutes to keep the system functioning. As you can see, there are several risks that need to be considered when deciding what design solution to use. In this thesis, we will create a set of synthesizable VHDL libraries targeting FPGAs and ASICs. However, we are putting emphasis on the FPGAs due to extra time and cost required for us to fabricate ASIC solutions for validation purposes.

2.4 MSP Design Reusability

Now you have seen the design steps required for a VLSI circuit design and how there are two different approaches to reach a solution. MSP offers design reusability by creating multiple modules of the same functioning task that focus on optimizing power, area, and delay modules. A typical design takes an average of 24 months to complete. The MSP design can take 3 months on average to complete, due to the synthesizable reusable libraries. Each design is unique to itself with different variables that depend on the length of time required to complete a design. Some of the variables that need to be considered are as follows: complexity factor, can you reuse anything from a previous design, is the task fully defined, experience of the designer, etc. A typical design timeline can be seen in Figure 2.4. The MSP libraries include the Architecture, RTL Coding, RTL Verification, and Synthesis. Using these MSP libraries can save time, money, and resources when designing a circuit. Chang and Aguan [3] state how important it is to have reusable VHDL modules in their journal article “Design-for-reusability” in VHDL.

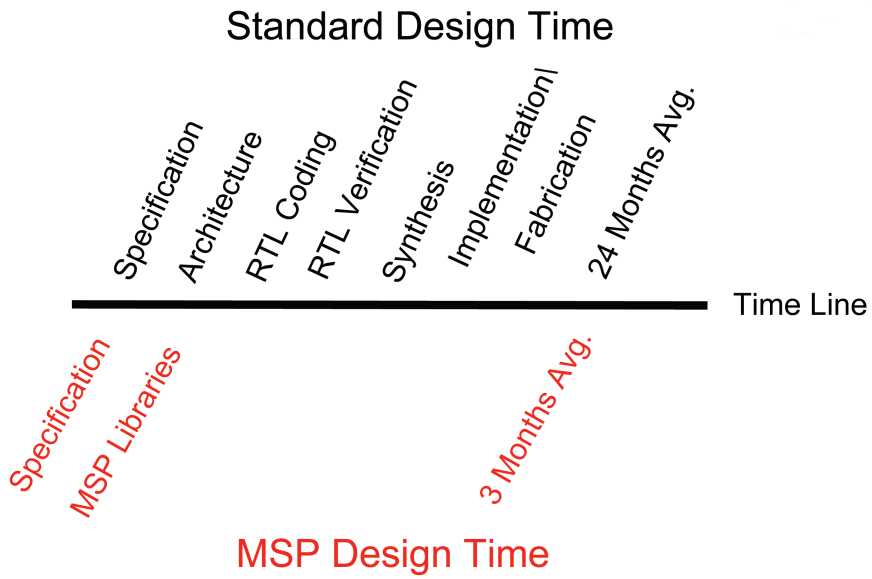


Figure 2.4: Standard Design Time vs. MSP Design Time

The reuse of electronic components can improve productivity in system design. However, without careful planning, components are rarely designed for reuse.

There is also more than one option when designing a multiplier, adder, or controller. You can build a behavior combinational multiplier, structurally combinational multiplier, behavior sequential multiplier, structurally sequential multiplier, or even a Booth multiplier just to name a few. They all have strengths and weaknesses for a design. The MSP idea is to build reusable libraries to meet the needs of the designers. A multiplier is a good example of this. The designer can build three variations of a multiplier to have one optimized for power, one for area, and one for delay (speed). When a new requirement is generated and it requires the use of a multiplier, the reusable libraries of power, area, and delay will be available. Since these modules have already been optimized, there is no additional work for the designer. This concept goes back to the the power, area, and delay triangle that was shown in Figure 1.1. The pre-built modules make it possible for the designer to only have to choose which module will work best for the specifications they were given. This can be seen in Table 2.1 where there are four types of multiplier designs showing their results for power, area, and delay. These results came from the software tool `Cadence`®. The

Table 2.1: Summary of Maximum Clock Frequency for Modules

Type of Multiplier	Power mW	Area mm^2	Max. Freq. MHz	Constrained
Booth	37.17	0.1537	53.99	Yes
Combinational Behavior	7.35	0.2043	53.09	No
Combinational Structural	9.13	0.2422	28.23	No
Sequential Behavior	64.87	0.2151	37.31	Yes

constrained column in the table gives a clear picture of what power, area, and maximum clock frequency can be achieved. When combinational designs are synthesized in `Cadence`®there is no clock used to determine how fast the design will run accurately. Therefore, `Cadence`®gives a best guess estimate for the power, area, and delay. As you can see, the Booth Multiplier has the fastest delay but the third largest amount of power consumption. One multiplier design may have the smallest area, but the slowest delay. It is up to the designer to pick which module will work best for their application.

If you take this a step further and expand to other modules like adders, subtractors, and controllers, you will have generated an arsenal of modules that have been optimized for power, area, and delay. This will be like picking out a new car. What options do you want for a new car - power windows, cd player, full size spare tire, etc.? Having these libraries built, reconfigurable, and reusable makes designing a weapon system simpler and saves time, money, and resources. Also, using an FPGA makes it easier to upgrade the system at a lower cost while applying less time. There will be several different variations of the same module focusing on optimized power, area, and delay. For example, if a design calls for fast multiplications to be performed, the designer has the option to choose the multiplier that has been optimized for speed from the reusable libraries that are already built.

In this thesis we will be implementing a Booth multiplier. The Booth multiplier is based on adding, subtracting, and shifting the binary values several times. Andrew Booth [5] noticed this can be achieved by having a lookup table to determine if the binary value needs to be added, subtracted, or do no operation and only shift the

binary values one place. The Booth algorithm is summarized in points 1-3 below [5]. By using a lookup table, it is possible to quickly determine what operation needs to be performed. The best thing about the Booth multiplier is the option it has to perform no operation if there is a run of "00" or "11" in the number being multiplied. This do nothing option reduces the number of operations, saving clock cycles required to compute the multiplication. For example, if the bit width is 32 bits, it will only require 16 clock cycles to complete the multiplication. For this thesis we will implement a 2 bit shift Booth multiplier to cut the clock cycles in half from the given number of input bits.

Booth Multiplier Algorithm Summary:

1. Examine each pair of digits in the multiplier, creating the first pair by appending a dummy '0' at the least significant end.

   If the pair is 01, add the multiplicand.

   If the pair is 10, subtract the multiplicand.

   Otherwise, do nothing.

2. Shift both partial product and multiplier one place to the right, allowing the next pair of digits to be examined.

3. Repeat as many times as there are digits in the multiplier [5].

2.5 Optical Flow

The Optical Flow algorithm compares two images together to see what is different between the two images. Some applications where optical flow are used are change detection, computer vision, pattern recognition, tracking targets, and image processing. Optical Flow is best described by Horn and Schunck [7] in their paper, "Determining Optical Flow."

> Optical flow is the distribution of apparent velocities of movement of brightness patterns in an image. Optical flow can arise from relative mo-

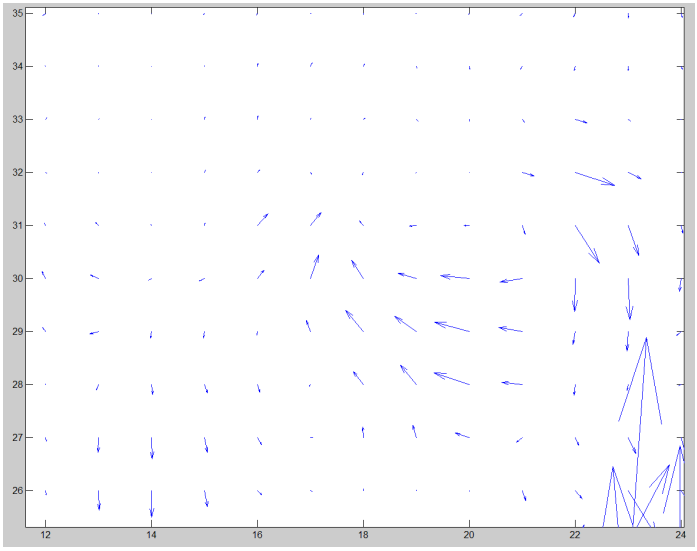


Figure 2.5: Optical Flow Vectors for a 250 × 400 Image

tion of objects and the viewer. Consequently, optical flow can give important information about the spatial arrangement of the objects viewed and the rate of change of this arrangement.

There are two [7] [10] well regarded methods for calculating optical flow - are Horn-Schunck and Lucas-Kanade methods. The Horn-Schunck [7] method looks at the difference between the brightness and contrasts between the two images to estimate what changes have occurred. The changes are represented in a vector field to show the direction of motion the image is moving compared to the first image. The Lucas-Kanade method [10] makes use of the "spatial intensity gradient of the images." Using the Horn-Schunck or the Lucas-Kanade method requires large amounts of calculations to determine if there is a change between the first image and the second image. In this project, we will look at the Lucas-Kanade method to determine an optical flow solution between two images. An example of a vector field that was produced between two images using the Lucas-Kanade method is shown in Figure 2.5. These results were attained from `Matlab`® comparing two images that are 250 × 400 pixels.

2.6 Chapter Summary

The background information for Chapter 2 covered the following: VLSI Design Process, FPGA vs. ASIC Risk, MSP Design Reusability, and Optical Flow. Several low power, area, and delay (speed) implementations were also looked at and discussed in the architecture section of the VLSI Design Flow section. MSP has shown the importance of creating many different modules for optimized power, area, and delay that will perform the same task. Two different methods to perform Optical Flow have been discussed, which were the Lucas-Kanade and Horn-Schunck Optical Flow methods. We will build reusable libraries in VHDL for the Lucas-Kanade method so the libraries can later build a complete Lucas-Kanade Optical Flow system. This image processing can be used in part of the target tracking project. These ideas will be discussed further in Chapter 3 to develop this thesis project.

III. Methodology

The methodology used to convert the Matlab® commands that generated the Dense Optical Flow will be discussed in this chapter. The research goal is to convert the Matlab® commands, located in Appendix B, used in the Dense Optical Flow to build reusable VHDL libraries such as, Conv2, Matrix Transpose, Round, Floor, and Pinv, etc. The goal is to demonstrate functioning MSP modules that are reusable for image processing such as Optical Flow, DSP, computer vision, pattern recognition, tracking targets, change detection, etc. These MSP libraries will be the key for the foundation of these applications.

The Dense Optical Flow Matlab® command functions will be created and demonstrated using smaller modules. These modules will make up the parts for the Matlab® command that can later be used to build an Optical Flow system using MSP modules. An example of some of the Matlab® commands that are used in Optical Flow code can be seen in Listing III.1.

Listing III.1:

```
% Example of using the conv2 command
hResult = conv2(im, mask);

% Example of using the matrix transpose command
curFx = curFx';

% Example of using the Pinv command
U = pinv(A'*A)*A'*curFt;

% Example of using the round command
uIn = round(uIn);

% Example of using the floor command
halfWindow = floor(windowSize/2);
```

To create these commands, smaller modules were designed around the Matlab® Dense Optical Flow written by Sohaib Khan [8] using the Lucas-Kanade algorithm [10].

The reason for creating individual modules and making them configurable components is to be able to use a generic module and shape it to meet the needs of a specific design. We will assume that the image sizes will not change while the data is being recorded with a camera and the modules will be set to receive grayscale images. This parameter is configurable to meet future needs. The configurability parameters of many of the individual modules can be seen in Table 3.1.

Table 3.1: Configurable parameters

Parameter	Description
address_width	Bit width of matrix size (row, column)
data_width	Bit width of pixel depth values

3.1 Overall Design

The overall design of the Dense Optical Flow was broken into smaller parts called modules. These modules were designed to take on the same characteristics as the `Matlab`® commands. There are two different designs for the Reduce Matrix Function that will be developed in VHDL. These two designs will look at large area and small power consumption between the designs. These modules were built using VHDL and tested with test benches to prove that they functioned properly. Other `Matlab`® commands that will be created in VHDL are the following: Conv2, Round, Floor, Matrix Transpose, Compute Derivatives, Reduce Matrix Function, Adder, Multiplier, Divider, Subtractor, and Pinv. Also, the goal is to implement the `Matlab`® functions as VHDL modules. For instance, the 2×2 matrix called A has the following parameters while the 2×2 matrix called B has the following parameters:

$$\text{Matrix A} \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \quad \text{Matrix B} \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$$

The following `Matlab`® code calculates the two-dimensional convolution of Matrices A and B as seen in Equation 3.1. The resulting answer is stored in the 3×3 Matrix C.

$$C = Conv2(A, B) \tag{3.1}$$

Matrix C produces the following result:

$$\text{Matrix C} \begin{pmatrix} 5 & 16 & 12 \\ 22 & 60 & 40 \\ 21 & 52 & 32 \end{pmatrix}$$

These results match the results from the VHDL convolution module with 0% error. The other Matlab® commands or functions were converted to VHDL to build the reusable module libraries to lay the foundation for Dense Optical Flow algorithm. The Matlab® commands that are only specific to Matlab® were developed and function properly include: Conv2, Matrix Transpose, Round, Floor, and Pinv.

3.2 Two Dimensional Convolution

The two dimensional convolution Matlab® command Conv2 can be seen in Equation 3.1. The Conv2 to Matlab® command is the backbone for the image processing that has to take place in an application such as Optical Flow. Other Matlab® functions that were developed by Sohaib Khan, such as a Reduce Function and Computing Derivatives, can not be developed without the use of the Conv2 command. Therefore, the Conv2 command was the first module that was created.

Matrix C is created from the size of Matrix A and B. For example, Matrix C is [D+F-1, E+G-1]. These parameters come from the size of Matrix A[D,E] and Matrix B[F,G]. The algorithm for the Conv2 can be seen in Equation 3.2.

$$C(n_1, n_2) = \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} a(k_1, k_2) b(n_1 - k_1, n_2 - k_2) \tag{3.2}$$

From the Matlab® Conv2 algorithm, VHDL modules were written and implemented having the same characteristics as the Matlab® command using a state

machine to take the place of the nested for-loops. The Conv2 Matlab® command requires the following modules to be built in VHDL in order for it to function:

1. Memory Module
2. Multiplier Module
3. Control Module
4. Multiplexer Module
5. Adder Module
6. Register Module

Once these modules are built, they will be connected together, like connecting Legos®together, to build the Conv2 Module for VHDL. Simulations will be run to test the VHDL Conv2 and the results will be compared to the Matlab® Conv2 command.

3.2.1 Memory. Picture images are made up of grayscale values between 0-255 integer values that makeup the brightness of the image. These values are used to discern between adjacent pixel colors of the images. A picture image is nothing more than a two-dimensional matrix that has rows and columns and is loaded with integer values to represent the brightness of the pixels. The memory method that will be used is to initially generate a matrix larger than the image that is going to have the Conv2 command performed on it. This is required when the Conv2 command is finished computing, the resultant matrix is larger than the initial image matrix size. Therefore, to take into account the matrix being larger than the initial image size, the memory module will initially be filled with all zero's. This oversized matrix eliminated the problem of the Conv2 command from stepping out of bounds when computing the results for the Conv2 command. The matrix can then be filled with the required grayscale values that represent the image.

3.2.2 Multiplier. The initial Multiplier Module that will be developed will only be a basic behavior model. The second Multiplier Module that will be developed

will be a 2-bit Booth Multiplier. This multiplier will be able to cut the clock cycle in half for calculating the results. For instance, the data input for the pixel value for the image is 8 bits, but we need to take into account the fractional parts of numbers that are multiplied together. The `Matlab`® Optical Flow uses a 1×5 mask matrix that is made up of a Gaussian distribution. The Mask 1×5 matrix has the following values:

$$\text{Mask Matrix} \begin{pmatrix} .05 & .25 & .4 & .25 & .05 \end{pmatrix}$$

When these calculations are performed, the results give a fractional part to the resultant. Therefore, we will use fixed-point notation to represent the fractional part of the number. In doing so, we will allocate 16 bits for the fractional part to represent the Gaussian distribution trying to minimize the error that will be caused since the values .05 or .4 will be only a close proximation of these values. This leaves us with 16 bits for the fraction part and 8 bits for the grayscale images yielding a total of 24 bits required to represent the images. We will initially set the image data width to 24 bits. However, this parameter is configurable to a larger bit width, if required.

3.2.3 Control. The Control Module could be implemented using a for-loop but these loops cannot typically be used for algorithmic iterations in synthesizable code [9]. Therefore, a state machine will be built to unroll the iterative loop that will increase throughput for the design.

3.2.4 Multiplexer, Adder, Register. The Multiplexer Module is used to switch between the loading of images and the Control Module, and is connected to the Memory Module. Once the image is loaded into the Memory, the Multiplexer will switch from a load mode to only communicate with the control logic. The Adder Module is used to add the values together once the multiplication is completed for a set of values. The Conv2 `Matlab`® commands require the sum of matrix positions to be multiplied and added together. The Register Module locks in the values that have been added together into its register. Once all the calculations have been completed,

the Control Module will send a signal to Matrix C so the data in the Register Module can be written into Matrix C at location (0,0).

3.3 Matrix Transpose

Part of the `Matlab`® code transposes the input image. The following modules are required to build the Matrix Transpose Module:

1. Memory Module
2. Control Module
3. Multiplexer Module

Since the goal of the MSP is to create reusable libraries, creating the Matrix Transpose Module has been simplified because the Memory Module was already developed while creating the Conv2 command. Therefore, the only new module required to be built is a state machine Control Module to transpose a matrix.

3.4 Compute Derivatives

Another `Matlab`® function module that needs to be developed is one that is able to compute derivatives. The Fx and Fy derivatives are used to determine the edge detection of an object that has moved between Image1 and Image2. The Ft derivative is the summation of Image1 and Image2. You can picture it as placing Image2 on top of Image1, which forms the Ft derivative. The Compute Derivatives Module is built around the `Matlab`® Conv2 command. Again, the MSP reusable libraries make it possible for the creation of the Compute Derivative function by being able to recycle the Memory and Conv2 modules. The equation seen in Equation 3.3 is from the Compute Derivatives function that was written in `Matlab`® . This equation requires the Conv2 command to be used twice to multiply Image1 by a fixed 2×2 matrix and added together with the result from Image2 being Conv2 with a fixed 2×2 matrix.

$$Fx = Conv2(Image1, 0.25[-11; -11]) + Conv2(Image2, 0.25[-11; -11]); \quad (3.3)$$

The following modules are required to build the Compute Derivative Module:

1. Memory Module
2. Control Module
3. Conv2 Module

The Matlab® Compute Derivatives Function code can be seen in Listing III.2.

Listing III.2:

```
[fx, fy, ft] = ComputeDerivatives(im1, im2);

function [fx, fy, ft] = ComputeDerivatives(im1, im2);
% ComputeDerivatives Compute horizontal, vertical and
% time derivative between two gray-level images.

if (size(im1,1) ~= size(im2,1)) | (size(im1,2) ~= size(im2,2))
   error('input images are not the same size');
end;

if (size(im1,3)~=1) | (size(im2,3)~=1)
   error('method only works for gray-level images');
end;


fx = conv2(im1,0.25*[-1 1; -1 1])+ conv2(im2, 0.25*[-1 1; -1 1]);
fy = conv2(im1,0.25*[-1 -1; 1 1])+ conv2(im2, 0.25*[-1 -1; 1 1]);
ft = conv2(im1,0.25*ones(2))+conv2(im2, -0.25*ones(2));

% make same size as input
fx=fx(1:size(fx,1)-1, 1:size(fx,2)-1);
fy=fy(1:size(fy,1)-1, 1:size(fy,2)-1);
ft=ft(1:size(ft,1)-1, 1:size(ft,2)-1);
```

3.5 Reduce Matrix Function

When dealing with image processing, the images need to be down sampled in size. This down sampling is part of the Optical Flow. We will use the terminology Reduce Matrix Function instead of down sampling. The Reduce Matrix Function is another Matlab® function that needs to be created in VHDL. The Reduce Matrix Function Module takes the initial image and reduces it in size by one half. For example, an initial image size of 250×400 will be 125×200 after performing the Reduce Matrix Function. Here is another example of how the MSP design thought process uses the created reusable libraries to develop another module. The key module in image processing is the Conv2 Module; again, it is used to build another module. The following modules are required to build the Reduce Matrix Function Module:

1. Conv2 Module
2. Control Module
3. Reduce Control Horizonal Module
4. Reduce Control Vertical Module
5. Memory Module
6. Multiplexer Module
7. Adder Module
8. Register Module

The Matlab® Reduce Matrix Function code can been seen in Listing III.3.

Listing III.3:

```
1 function smallIm = Reduce(im)
2 % REDUCE Compute smaller layer of Gaussian Pyramid
3
4 % Sohaib Khan, Feb 16, 2000
5
6 %Algo
```

```
% Gaussian mask = [0.05 0.25 0.4 0.25 0.05]
% Apply 1d mask to alternate pixels along each row of image
% apply 1d mask to each pixel along alternate columns
% of resulting image

mask = [0.05 0.25 0.4 0.25 0.05];

hResult = conv2(im, mask);
hResult = hResult(:,3:size(hResult,2)-2);
hResult = hResult(:, 1:2:size(hResult,2));

vResult = conv2(hResult, mask');
vResult = vResult(3:size(vResult,1)-2, :);
vResult = vResult(1:2:size(vResult,1),:);

smallIm = vResult;
```

3.6 Round/Floor

The Round and Floor commands are predefined Matlab® commands that do not exist in VHDL. The Round Module will take an input and round it up or down depending on what the fractional part of the number is. If, for example, the number is 4.5, the Round Module will round the number to 5 and if the number is 4.49 it will round the number to 4. On the other hand, the Floor command will take the floor of a number. For example, if the value is 5.9 it will floor the value to 5.

3.7 Pseudoinverse (Pinv)

The Pseudoinverse, also known as Pinv, is used to calculate the inverse of a matrix. Not all matrices have an inverse, therefore, the Pseudoinverse is used to find a close matrix inverse for a matrix. The Pseudoinverse [12] Equation 3.4 is the equation used to calculate the Pseudoinverse for Matrix A and the answer is stored in Y. The Matlab® code uses the equation seen in Equation 3.5 to do this

calculation. The method used to build the Pseudoinverse Module will be to use the Matrix Transpose module and multiply it by the original Matrix A. While that is being done, we will multiply Matrix Transpose by B_{Vector}, where the B_{Vector} is the Ft derivative. Doing these two multiplications in parallel, we will perform the matrix inverse on the resultants using the LU-Factorization method [6] to find the inverse. The Pseudoinverse Module will include the following modules for its development:

$$Y = (A^T A)^{-1} A^T \tag{3.4}$$

$$Y = (A^T A)^{-1} (A^T B_{Vector}) \tag{3.5}$$

1. Memory Module
2. Control Module
3. Multiplier Module
4. Divider Module
5. Adder Module
6. Matrix Transpose Module
7. Register Module

3.8 Synthesis/Timing

The `Xilinx`®Virtex-4 SX ML402 FPGA will be the target FPGA to attain the power, area, and delay of the circuits. The Virtex-4 is a generalized moderate cost FPGA that is a practical standard FPGA for DSP applications. The Virtex-4 ML402 is a good evaluation board with a wide range of applications such as DSP and low power. The `Precision RTL`®RTL 2007a.8 and `Xilinx`®ISE 9.2 are the two synthesis tools that will be used.

In one method, the two dimensional convolution module is built using sequential and combinational modules that make up the Conv2. The Conv2 module does many

multiplications and additions to compute the result for Matrix C. These sub-modules are built using combinational logic. Therefore, if the clock runs too fast, the multiplications and additions will not be completed before the next clock cycle, causing the wrong result to be stored in Matrix C. One of the ways to eliminate this problem is to lower the clock speed.

A second method would be to place a pipeline register between the multiplier and adder to increase throughput by increasing the clock speed. Timing is important for Dense Optical Flow due to the number of calculations required to complete two simple 2×2 matrices to create a 3×3 matrix. Timing simulations will show how fast the system can run to increase throughput. Timing simulations can also determine if it is required to add additional pipeline registers to increase throughput. The timing simulations can also determine how slow the clock can run without including additional registers or hardware. By slowing the clock speed to a minimal speed, power can be saved in the design. All these things rely on the timing simulations. The `Mentor Graphics`®tools will also play a role in attaining the results for Chapter 4 of this thesis project.

3.9 Testing Procedure

There are several ways to perform testing on these modules that will be developed. First, `Modelsim`®version 6.3c will be used to develop the VHDL code. The `Modelsim`®software will also be used to generate test benches to show the functionality of modules. Once the VHDL modules are functioning, they will be run through the `Precision RTL`®2007a.8 and `Xilinx`®ISE 9.2 software programs for synthesis to look at power, area, and delay of the circuits. This will give an estimate of how to reduce high power draws that occur due to the high amounts of calculations required by Dense Optical Flow. This may require that the circuit clock speeds be slowed down to reduce power, which in turn can minimize circuit size and slow the speeds of the system being tested.

3.10 Chapter Summary

The idea of breaking the `Matlab`® commands and functions into synthesizable reusable library modules that are used in image processing were discussed. These modules will support AFRL/RY later when they want to create the Optical Flow as a system by taking the synthesizable reusable libraries and connecting them together to build an Optical Flow that can be fully designed in VHDL. The synthesis, timing, and testing procedures were introduced and discussed. The software that will be used to carry out this methodology was discussed and how it will be used. Each sub-module was thoroughly tested to ensure there will be no errors when final assembly of the sub-modules are connected to create an MSP module such as a Conv2, Matrix Transpose, Round, Floor, and Pinv, etc.

IV. Analysis and Results

This chapter will discuss the overview design of the MSP modules that were discussed in the methodology section. We will also look at the power, area, and delay for the MSP modules. We will also look at the error analysis calculated between the `Matlab`® code and the VHDL modules.

4.1 Overview of MSP Modules

The overarching module that needed to be created was the Conv2 Module. This module was used in the development of the Reduce Matrix Function and Computing Derivatives. The Top-Level Design 1 for the Reduce Matrix Function can be seen in Figure 4.1. The Reduce Matrix Function requires the use of Conv2 Module to perform the Conv2 `Matlab`® command. When using the Conv2 command in `Matlab`® the matrix size grows larger due to the Conv2 algorithm. The hControl/vControl Reduce Module are used to reduce the oversized matrix that is created when Conv2 is performed. The Reduce Matrix Function is used to reduce a matrix size in half. For example, an image the size of 6×8 reduces to a 3×4. The Conv2 Module will generate a 6×12. The hControl Reduce Module then reduces the 6×12 matrix down to 6×4. The Conv2 Module will generate a 10×4 which will be reduced to 3×4 using the vControl Reduce Module. The final image is stored in the Memory Module as a 3×4.

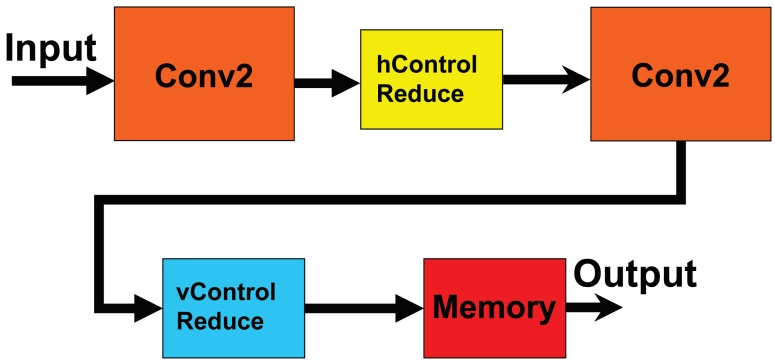


Figure 4.1: Reduce Matrix Function Top Level Design 1

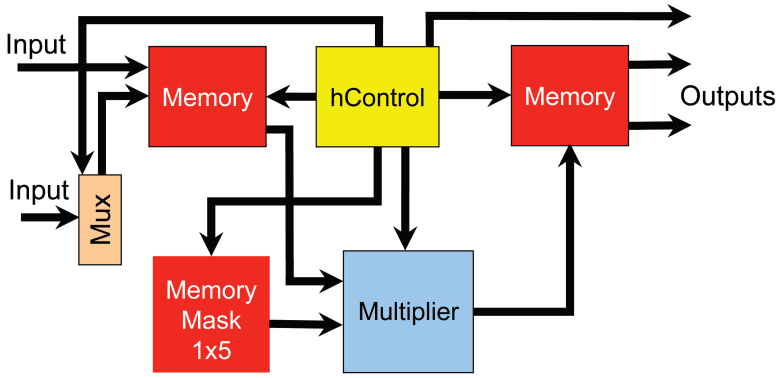


Figure 4.2: Conv2 Sub-Modules Design

The modules that were used in the the Top-Level Design 1 for the Conv2 Module can be seen in Figure 4.2. These modules make up the Conv2 Module. The first Conv2 Module uses a 1×5 Mask matrix opposed to the 5×1 Mask matrix that is used in the second Conv2 Module. The Conv2 Module design uses Multiplexer, Memory, Multiplier and Controller Modules. They are connected together to achieve the result for Conv2 `Matlab`® command.

The Behavior Multiplier can be seen in Figure 4.3. The three modules, Behavior Multiplier, Adder, and Register are required in the development of the Conv2 command.

The modules that are required to build a 2-bit Booth Multiplier Module can be seen in Figure 4.4. The Booth Multiplier Module requires the input values to be loaded into the Multiplicand and Product Module. The Control Module is used to communicate between the other modules.

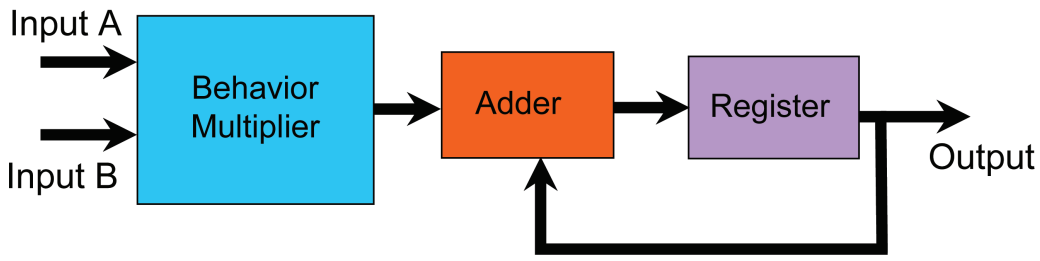


Figure 4.3: Behavior Multiplier Module Design

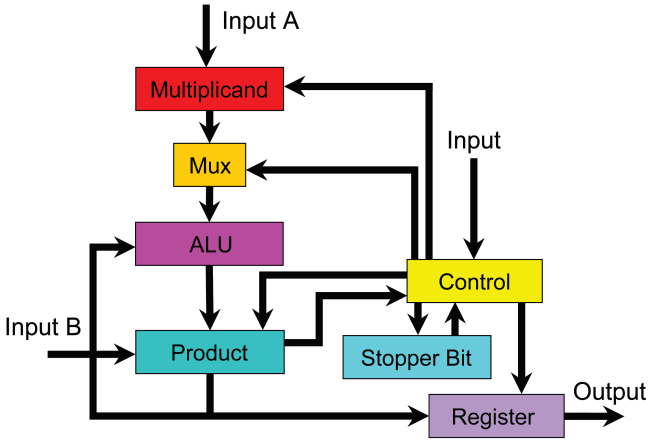


Figure 4.4: Booth Multiplier Module Design

The Top-Level Design 2 that was used to develop the Reduce Matrix Function can be seen in Figure 4.5. This design approach eliminates the use of a second Multiplier Module reducing power consumption and minimizing area compared to Top-Level Design 1. Further analysis will be discussed later in Chapter 4 to see the differences between the two design methods. Every large scale design requires a Control Module to communicate between the other modules. The Control Module is basically the "brains" for the module.

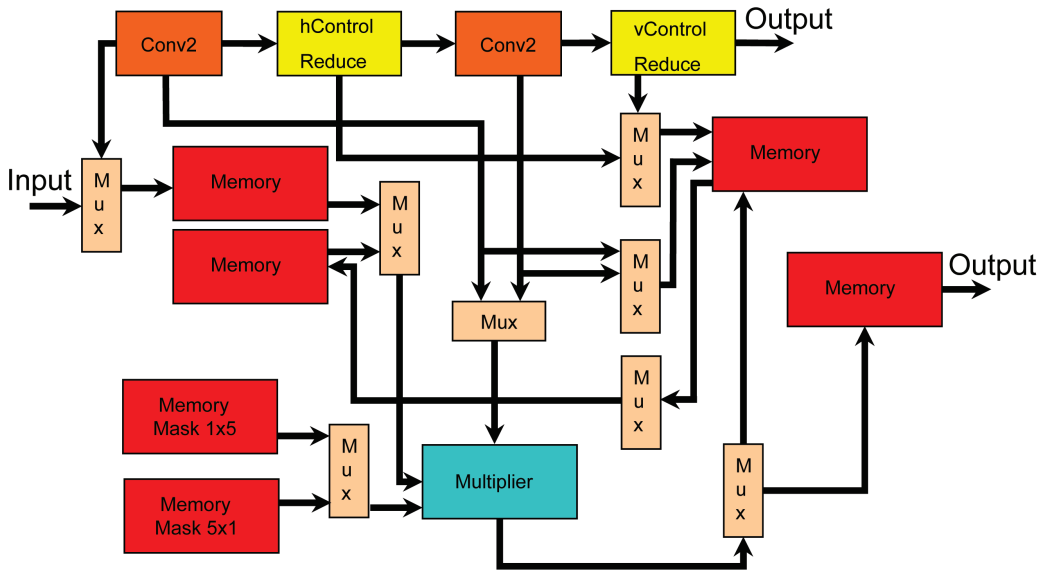


Figure 4.5: Reduce Matrix Function Top Level Design 2

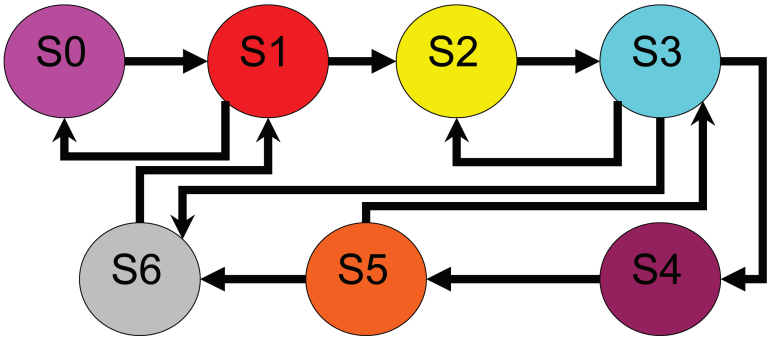


Figure 4.6: Multiplier State Machine Design

The Control Module that was used for the Behavior Multiplier Module uses seven states for its state machine. The Control Module is used to communicate between the Multiplier Module and the other modules. The seven states that the Control Module uses to communicate with the Behavior Multiplier Module can be seen in Figure 4.6. The Booth Multiplier, on the other hand, requires that the Control Module be built using 24 states for its state machine if using a 32 bit input. The Booth Multiplier state machine can be seen in Figure 4.7. The Booth Multiplier is performing its shifts, adds, and subtraction operations during States S4-S20 for a 32 bit number. If the input number was to be reduced to 24 bits, it would require 4 less states for the Controller to use.

The Compute Derivatives Top-Level Design can be seen in Figure 4.8. Again, you can see how the Conv2 Module was used to build the Compute Derivatives Module. This function requires the use of the Conv2, Controller, Adder, and a Memory

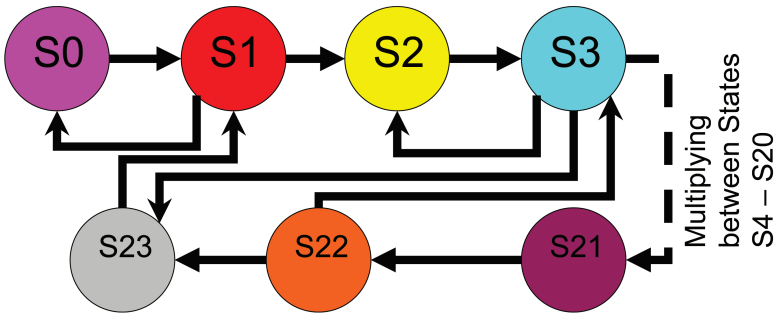


Figure 4.7: Booth Multiplier State Machine Design

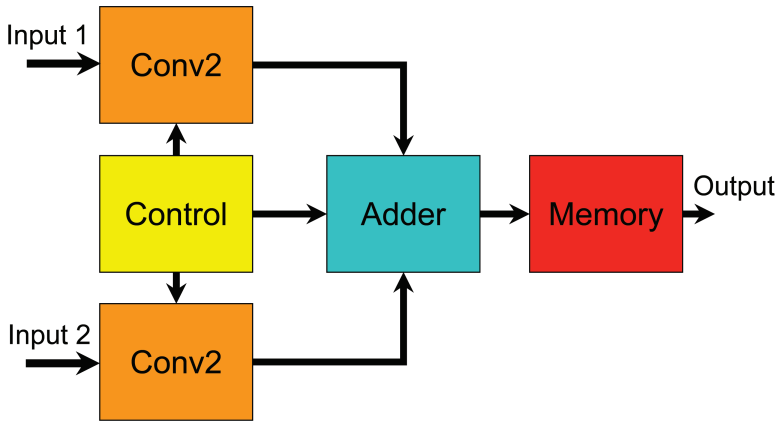


Figure 4.8: Compute Derivatives Top-Level Design

to build the derivative module in VHDL. The MSP Design effort payed off in the creation of the MSP Compute Derivatives Modules by being able to reuse previously designed modules.

The Matrix Transpose Module requires the use of a Multiplexer, Controller, and Memory. The Matrix Transpose Module receives data through the Multiplexer which loads the Memory. The Controller is used to transpose the values from the Memory to Memory Transpose. The Top-Level design of the Matrix Transpose Module can be seen in Figure 4.9.

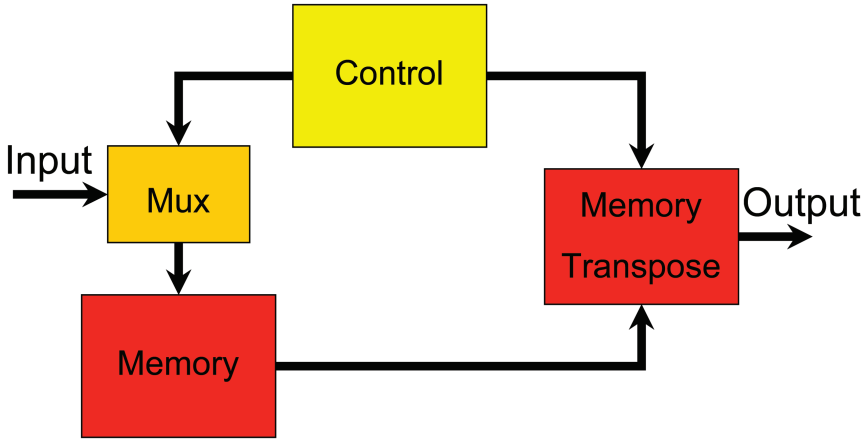


Figure 4.9: Matrix Transpose Top-Level Design

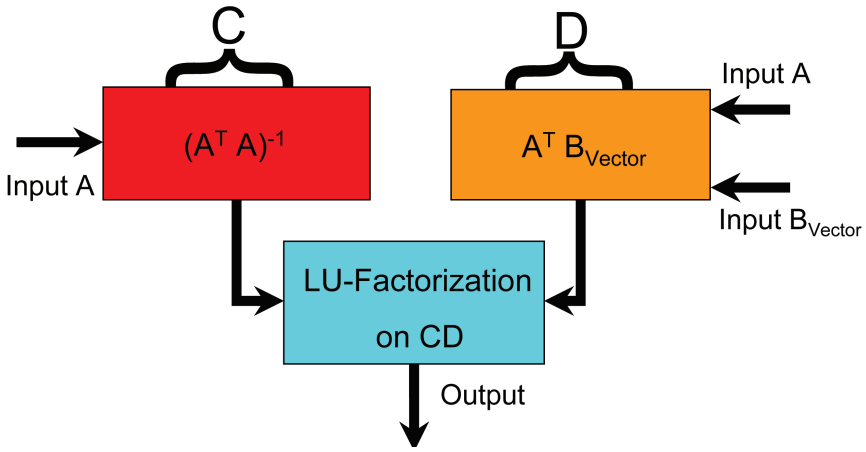


Figure 4.10: Pseudoinverse Top-Level Design

The Pseudoinverse (Pinv) Module can be broken into 3 Modules. The $(A^TA)^{-1}$ is one module, the A^TB_{Vector} the other module, and finally the module that will perform the LU-Factorization. The Pinv Module Top-Level Design can be seen Figure 4.10.

4.2 Error Analysis

When creating the VHDL modules to have the same functionality as the `Matlab`® commands and functions, error is introduced. Part of this error is introduced and can be seen from the Mask Matrix 1×5 or its transpose 5×1 that have the Gaussian Distributions values in the Memory Matrix. The values .05 and .40 can not exactly be represented in binary numbers when only using a limited amount of bits to represent these values. We used 16 bits to represent the fractional part of the Mask Matrix values. Therefore, when the Conv2 Module multiplies the image by the mask, the resultant will have error. This is due to the fact that the Mask Values were not being exactly compared to the `Matlab`® values. The close approximation values that were used for the Mask Matrix and `Matlab`® values can be seen in Table 4.1. The error analysis equation that will be used is $\%Error = \frac{VHDL-MATLAB}{MATLAB} \times 100$.

The error analysis that was shown for the Reduce Matrix Function for the VHDL compared to the `Matlab`® can be seen in Figure 4.11. The largest amount of error

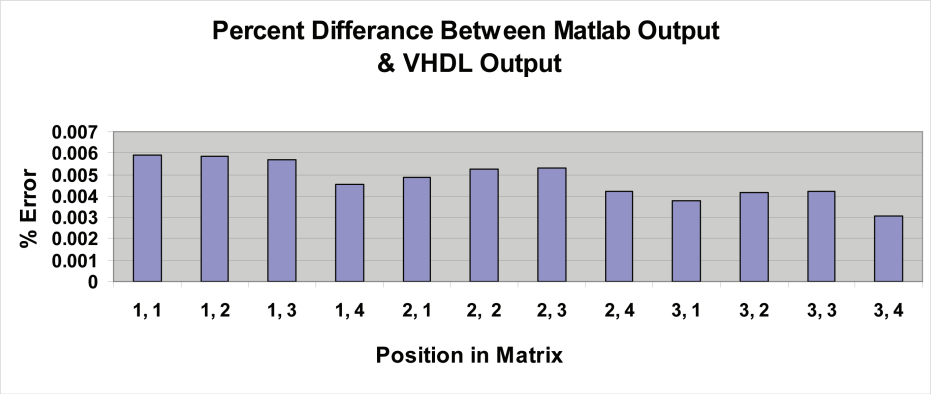


Figure 4.11: Reduce Matrix Function `Matlab`® vs. VHDL Percent Error

Table 4.1: Comparison between `Matlab`® and VHDL Gaussian Distribution

`Matlab`® Gaussian Distribution	0.05	0.25	0.4	0.25	0.05
VHDL Gaussian Distribution	0.049987793	0.25	0.399993896	0.25	0.049987793
% Error Difference	0.024414	0	0.001526	0	0.024414

that occurred was .0059% for the sample 6×8 data set that we used. The 6×8 data set that was used can be seen in the Input Data Matrix. This data set was arbitrarily chosen. The Reduce Matrix Function reduces the 6×8 matrix down to 3×4, but not before the 6×8 has the Conv2 Module performed on it. The x-axis represents the positions in the the 3×4 matrix. The y-axis shows the percent error between the `Matlab`® output and the VHDL output.

$$\text{Input Data Matrix} \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 \\ 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \\ 41 & 42 & 43 & 44 & 45 & 46 & 47 & 48 \\ 25 & 26 & 27 & 28 & 29 & 30 & 31 & 32 \\ 33 & 34 & 35 & 36 & 37 & 38 & 39 & 40 \end{pmatrix}$$

The Compute Derivatives for Fx, Fy, and Ft had zero percent error compared to the `Matlab`® code. This is due to the fact that the Compute Derivative equation

used a positive or negative 2×2 matrix with the values .25 which can be represented in binary with zero percent error. Also, the Matrix Transpose Module had zero error because it took the original Matrix A and transposed it to become Matrix A^T. There was no loss of data precision when transposing Matrix A.

The sample test data set that was used to test the Pseudoinverse Module was Matrix A 9×2 and Matrix B_{Vector} 9×1. The comparison between Matlab® and VHDL results can be seen in Table 4.2. The results are stored in a 2×1 matrix. The largest percent error from the sample test data was 2.65%. A small amount of error occurs from the divider that is used to calculate the Pseudoinverse. This small error gets magnified when the value is multiplied. This intermediate result is divided and multiplied again as part of the LU-Factorization in order to find the Pseudoinverse.

$$\text{Matrix A} \begin{pmatrix} 6 & 8 \\ 1 & 3 \\ 10 & 12 \\ 5 & 7 \\ 15 & 14 \\ 4 & 3 \\ 12 & 13 \\ 0 & 11 \\ 2 & 17 \end{pmatrix} \quad \text{Matrix } B_{Vector} \begin{pmatrix} 1 \\ 5 \\ 7 \\ 9 \\ 10 \\ 15 \\ 20 \\ 2 \\ 8 \end{pmatrix}$$

Table 4.2: Comparison between Matlab® and VHDL Pseudoinverse Module

	Matrix Position (1, 1)	Matrix Position (2, 1)
Matlab® Results	0.731775	0.305488
VHDL Results	0.716705	0.313827
% Error Difference	2.10	2.65

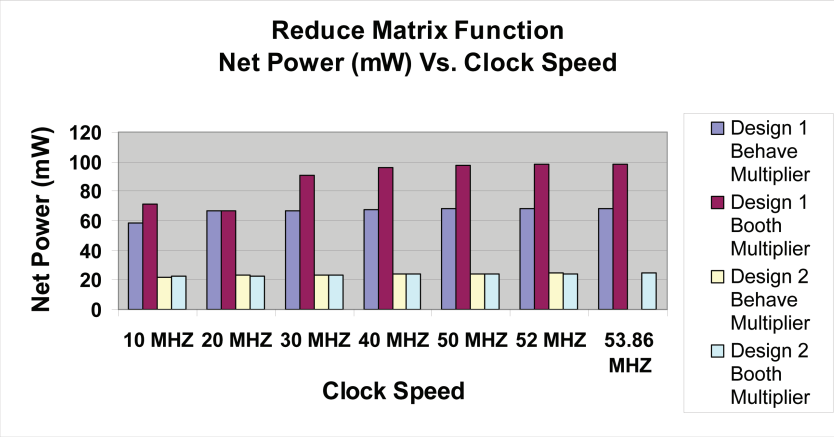


Figure 4.12: Reduce Matrix Function Module Net Power(mW) vs. Clock Speed

4.3 Power, Area, and Delay

In the power, area, and delay section we will look at the results that were attained from the MSP modules implemented using the `Cadence`®synthesis tool. The `Cadence`®software tool did not specifically target an FPGA or an ASIC for the results that were attained.

First, we will look at the results that were attained from the Reduce Matrix Function, seen in Figure 4.12. From the excel chart, it is clear that the Top-Level Design 2 approach requires less net power to be used for the Reduce Matrix Function Module. The power savings that occurs for Top-Level Design 2 over Top-Level Design 1 is at least 2.8 times smaller. In Figure 4.12 the `Cadence`®software also stated that the fastest delay for Top-Level Design 1 using a behavior Multiplier Module ran at 52.02 MHz. The other designs ran at 53.86 MHz.

In Figure 4.13 it is clear that Top Level Design 2 uses a smaller area in the design, up to 54.6% smaller, compared to the Behavior Multiplier Top-Level Design 1 and Booth Multiplier Top-Level Design 2.

The results for Fx, Fy, and Ft can be seen in Figure 4.14 for Power vs. Clock Speed using the `Cadence`®software. As expected, as you increase the clock speed, the net power increases. The clock speeds and area that were attained for the excel charts

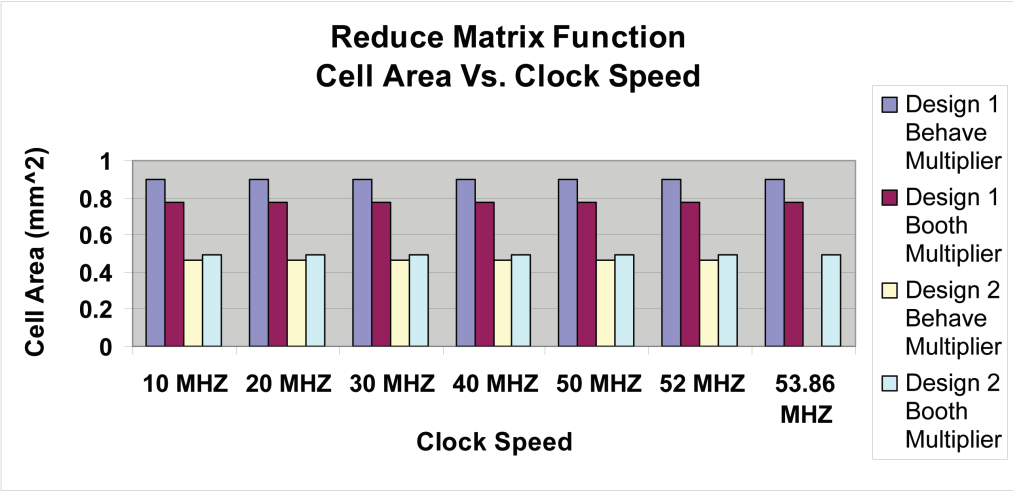


Figure 4.13: Reduce Matrix Function Module Cell Area vs. Clock Speed

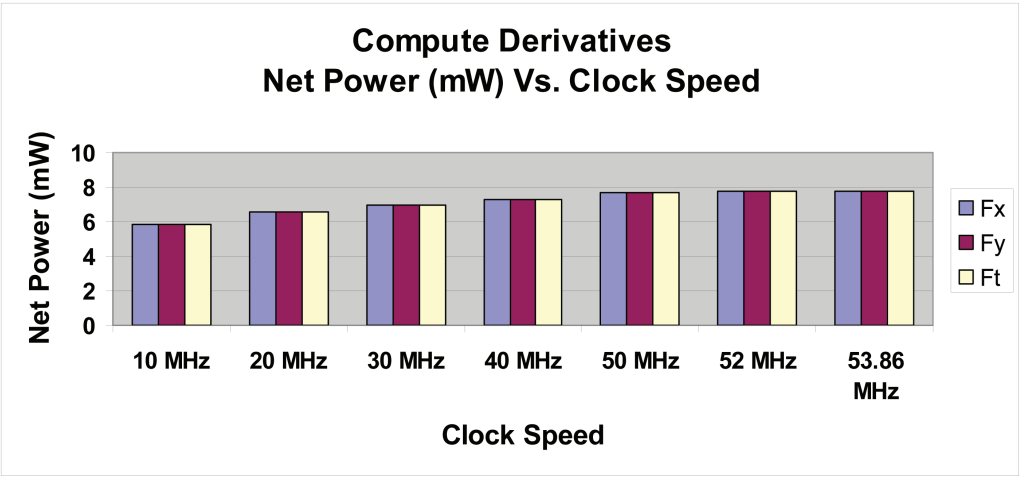


Figure 4.14: Compute Derivatives Module Net Power(mW) vs. Clock Speed

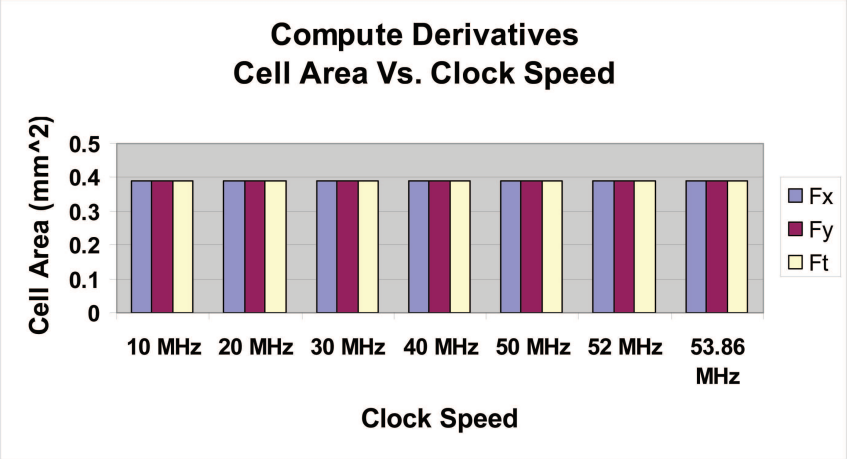


Figure 4.15: Compute Derivatives Module Cell Area vs. Clock Speed

are only estimations generated from the `Cadence`®software. Every software package uses different algorithms to attain the power, area, and delay.

The area for the Compute Derivatives was unchanged as the speed increases as shown in Figure 4.15. For example, you are able to achieve the same cell area for 10 MHz design or a 52 MHz design.

The Matrix Transpose Module achieved the results for Power vs. Clock Speed in Figure 4.16. The net power increased as the speed increased which is what is

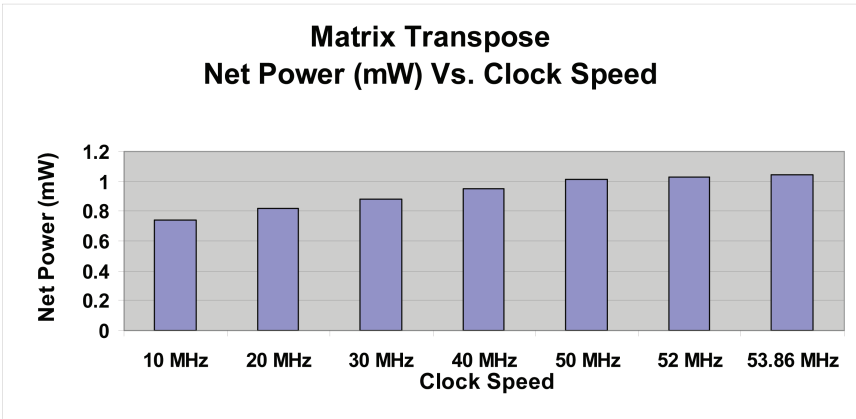


Figure 4.16: Matrix Transpose Module Net Power(mW) vs. Clock Speed

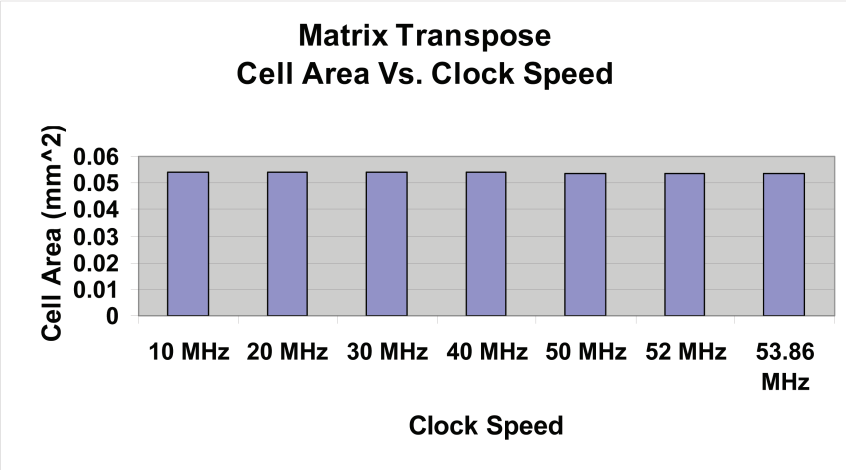


Figure 4.17: Matrix Transpose Module Cell Area vs. Clock Speed

expected. The cell area was unchanged when the clock speed was increased. This can be seen in Figure 4.17.

The Pseudoinverse Module achieved the results for Power vs. Clock Speed in Figure 4.18. The net power increased as the clock speed increased which is what is expected. The cell area was unchanged when the clock speed was increased. This can be seen in Figure 4.19.

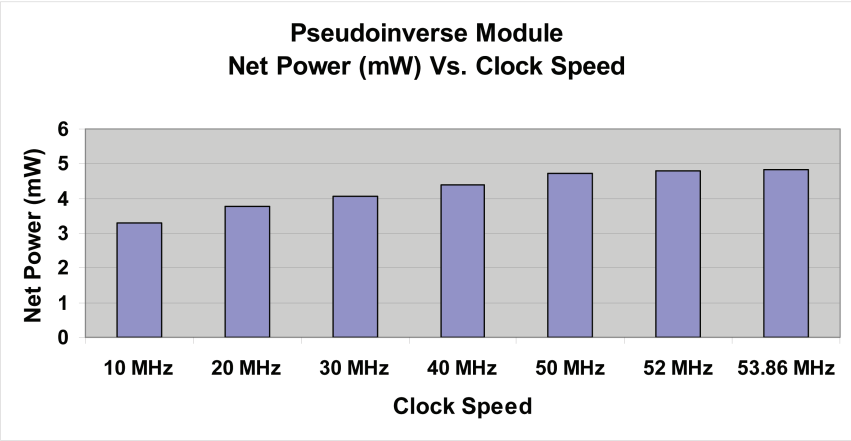


Figure 4.18: Pseudoinverse Module Net Power(mW) vs. Clock Speed

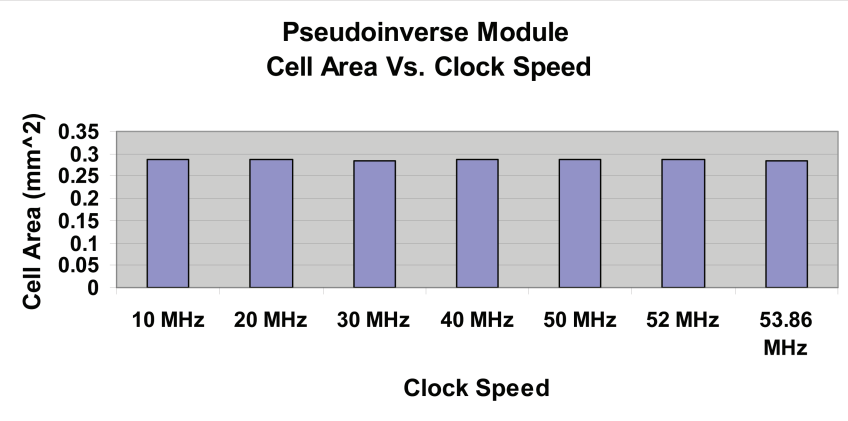


Figure 4.19: Pseudoinverse Module Cell Area vs. Clock Speed

4.4 Synthesis

In the synthesis section, all the modules were targeted for the `Xilinx`®Virtex-4 SX ML402 FPGA that uses 90nm technology. The device used was $4VSX35FF668$, and a Speed Grade of -10 was used to attain area and speed for the synthesis. The `Xilinx`®Virtex-4 board isn't a state-of-the art board and therefore is very cost efficient. Without a doubt, high-end FPGA systems will work much faster (2-3 times the speed) than the current one with Configurable Logic Blocks (CLB) to spare. Like any synthesis software tool, each one uses different algorithms for optimal power, area, and delay for designs. That is why the results in this section will not match the results from the `Cadence`®software since it uses 250 nm technology to attain the area and delay for the modules. The Reduce Matrix Function Module was synthesized using the `Precision RTL`®software, which is a `Mentor Graphics`®product.

The Reduce Matrix Function Module for the Behavior Multiplier Top-Level Design 1 results can be seen in Table 4.3. The constraint for the frequency used was 76 MHz. After synthesis, the maximum clock speed/frequency was 76.482 MHz. Therefore, the synthesis had no timing violations. The Reduce Matrix Function used only 10.24% of the FPGA area. There are a total of 15360 CLB slices available for the targeted FPGA.

Table 4.3: FPGA Synthesize Results for Top-Level 1, Behavior Multiplier

Resources	Used	Available	Utilization %
Inputs & Outputs	264	448	59.93
Global Buffers	1	32	3.13
Function Generators	3145	30720	10.24
CLB Slices	1573	15360	10.24
Dffs or Latches	2027	31616	6.41
Block RAMs	5	192	2.60
DSP48s	10	192	5.21

Table 4.4: FPGA Synthesize Results for Top-Level 2, Behavior Multiplier

Resources	Used	Available	Utilization %
Inputs & Outputs	166	448	37.05
Global Buffers	1	32	3.13
Function Generators	2725	30720	8.87
CLB Slices	1362	15360	8.87
Dffs or Latches	1670	31616	5.28
Block RAMs	4	192	2.08
DSP48s	8	192	4.17

The Reduce Matrix Function Module for the Behavior Multiplier Top-Level Design 2 results can be seen in Table 4.4. The constraint for the frequency used was 73 MHz. After synthesis, the maximum clock speed/frequency was 73.115 MHz. Therefore, the synthesis had no timing violations. Design 2 only requires 8.87% of the CLB slices saving 15.49% more area than Design 1.

The Reduce Matrix Function Module for the Booth Multiplier Top-Level Design 1 results can be seen in Table 4.5. The constraint for the frequency used was 100 MHz. After synthesis, the maximum clock speed/frequency was 101.75 MHz. Therefore, the synthesis had no timing violations. Design 1 only requires 11.32% of the CLB slices. This results in a 10% increase in area over the behavior design. The Booth design has a 39.1% increase in maximum clock frequency over the Behavior Design 1.

The Reduce Matrix Function Module for the Booth Multiplier Top-Level Design 2 results can be seen in Table 4.6. The constraint for the frequency used was 100 MHz. After synthesis, the maximum clock speed/frequency was 100.422 MHz. Therefore,

Table 4.5: FPGA Synthesize Results for Top-Level 1, Booth Multiplier

Resources	Used	Available	Utilization %
Inputs & Outputs	268	448	59.82
Global Buffers	1	32	3.13
Function Generators	3475	30720	11.31
CLB Slices	1738	15360	11.32
Dffs or Latches	2387	31616	7.55
Block RAMs	5	192	2.60
DSP48s	6	192	3.13

Table 4.6: FPGA Synthesize Results for Top-Level 2, Booth Multiplier

Resources	Used	Available	Utilization %
Inputs & Outputs	173	448	38.62
Global Buffers	1	32	3.13
Function Generators	2887	30720	9.40
CLB Slices	1444	15360	9.40
Dffs or Latches	1867	31616	5.91
Block RAMs	4	192	2.08
DSP48s	6	192	3.13

the synthesis had no timing violations. Design 2 only requires 9.40% of the CLB slices. This results in a 6% increase in area over the Behavior Design. Although, the Booth design has a 37.34% increase in maximum clock frequency over the Behavior Design 2. Design 2 for the Booth is 20.36% smaller than Design 1 using a Booth multiplier.

The Compute Derivatives Fx, Fy, and Ft Modules synthesis results can be seen in Table 4.7, Table 4.8, and Table 4.9 respectively. The constraint for Compute Derivative Fx Module had a frequency of 105 MHz. After synthesis, the maximum clock speed/frequency was 105.876 MHz. Therefore, the synthesis had no timing violations.

The constraint for Compute Derivative Fy Module in Table 4.8 had a frequency of 105 MHz. After synthesis, the maximum clock speed/frequency was 105.876 MHz. Therefore, the synthesis had no timing violations.

Table 4.7: FPGA Synthesize Results for Compute Derivative Fx Module

Resources	Used	Available	Utilization %
Inputs & Outputs	106	448	23.66
Global Buffers	1	32	3.13
Function Generators	3439	30720	11.19
CLB Slices	1720	15360	11.20
Dffs or Latches	2187	31616	6.92
Block RAMs	5	192	2.60
DSP48s	4	192	2.08

Table 4.8: FPGA Synthesize Results for Compute Derivative Fy Module

Resources	Used	Available	Utilization %
Inputs & Outputs	106	448	23.66
Global Buffers	1	32	3.13
Function Generators	3439	30720	11.19
CLB Slices	1720	15360	11.20
Dffs or Latches	2187	31616	6.92
Block RAMs	5	192	2.60
DSP48s	4	192	2.08

The constraint for Compute Derivative Ft Module in Table 4.9 had a frequency of 101 MHz. After synthesis, the maximum clock speed/frequency was 101.75 MHz. Therefore, the synthesis had no timing violations.

Table 4.9: FPGA Synthesize Results for Compute Derivative Ft Module

Resources	Used	Available	Utilization %
Inputs & Outputs	106	448	23.66
Global Buffers	1	32	3.13
Function Generators	3368	30720	10.96
CLB Slices	1684	15360	10.96
Dffs or Latches	2183	31616	6.90
Block RAMs	5	192	2.60
DSP48s	4	192	2.08

The constraint for Matrix Transpose Module in Table 4.10 had a frequency of 158 MHz. After synthesis, the maximum clock speed/frequency was 158.856 MHz. Therefore, the synthesis had no timing violations. The Matrix Transpose Module only required 3.22% CLB slices to be used.

Table 4.10: FPGA Synthesize Results for Matrix Transpose Module

Resources	Used	Available	Utilization %
Inputs & Outputs	68	448	15.18
Global Buffers	1	32	3.13
Function Generators	990	30720	3.22
CLB Slices	495	15360	3.22
Dffs or Latches	588	31616	1.86
Block RAMs	2	192	1.04
DSP48s	2	192	1.04

The constraint for the Floor Module had a frequency of 471 MHz. After synthesis, the maximum clock speed/frequency was 471.032 MHz. Therefore, the synthesis had no timing violations and the results can be seen in Table 4.11. The Floor module only uses .01% of the CLB slices.

Table 4.11: FPGA Synthesize Results for Floor Module

Resources	Used	Available	Utilization %
Inputs & Outputs	9	448	2.01
Global Buffers	1	32	3.13
Function Generators	4	30720	0.01
CLB Slices	2	15360	0.01
Dffs or Latches	4	31616	0.01
Block RAMs	0	192	0.00
DSP48s	0	192	0.00

The constraint for the Round Module had a frequency of 148 MHz. After synthesis, the maximum clock speed/frequency was 148.236 MHz. Therefore, the synthesis had no timing violations and the results can be seen in Table 4.12. The Round Module uses 3.67% of the CLB slices.

The constraint for the Pseudoinverse Module had a frequency of 33 MHz. After synthesis, the maximum clock speed/frequency was 34.58 MHz. Therefore, the synthesis had no timing violations and the results can be seen in Table 4.13. The Pseudoinverse Module uses 28.27% of the CLB slices.

The summary of maximum clock speed can be seen in Table 4.14. The `Precision RTL`®software tool showed similar results as the `Cadence`®software tool. The Reduce

Table 4.12: FPGA Synthesize Results for Round Module

Resources	Used	Available	Utilization %
Inputs & Outputs	68	448	15.18
Global Buffers	1	32	3.13
Function Generators	1128	30720	3.67
CLB Slices	564	15360	3.67
Dffs or Latches	582	31616	1.84
Block RAMs	1	192	0.52
DSP48s	2	192	1.04

Table 4.13: FPGA Synthesize Results for Pseudoinverse Module

Resources	Used	Available	Utilization %
Inputs & Outputs	339	448	75.67
Global Buffers	2	32	6.25
Function Generators	8698	30720	28.31
CLB Slices	4349	15360	28.31
Dffs or Latches	4986	31616	15.77
Block RAMs	14	192	7.29
DSP48s	22	192	11.46

Table 4.14: Summary of Maximum Clock Speed for Modules

Module Name	Maximum Clock Speed MHz
Reduce Matrix Function Top-Level 1 Behavior Multiplier	76.48
Reduce Matrix Function Top-Level 2 Behavior Multiplier	73.11
Reduce Matrix Function Top-Level 1 Booth Multiplier	101.75
Reduce Matrix Function Top-Level 2 Booth Multiplier	100.42
Compute Derivative Fx	105.87
Compute Derivative Fy	105.87
Compute Derivative Ft	101.75
Matrix Transpose	158.85
Floor	471.03
Round	148.23
Pseudoinverse	34.58

Matrix Function Booth Multiplier Design outperforms the Behavior Multiplier by 39.1%.

The summary of the net power in Watts can be seen in Table 4.15. The `Precision RTL`®software tool showed similar results as the `Cadence`®software tool.

Table 4.15: Summary of Net Power Watts for Modules

Module Name	Net Power Watts
Reduce Matrix Function Top-Level 1 Behavior Multiplier	0.672
Reduce Matrix Function Top-Level 2 Behavior Multiplier	0.662
Reduce Matrix Function Top-Level 1 Booth Multiplier	0.703
Reduce Matrix Function Top-Level 2 Booth Multiplier	0.687
Compute Derivative Fx	0.700
Compute Derivative Fy	0.708
Compute Derivative Ft	0.700
Matrix Transpose	0.668
Floor	0.636
Round	0.660
Pseudoinverse	0.646

4.5 Chapter Summary

In this chapter we discussed the overview of the designs that were used to build the MSP synthesizable reusable libraries. Top-Level Design schematics were used to show how the modules connected together. Error analysis was discussed to see what error was introduced by using the Gaussian Distribution for the Mask Matrix Modules. These modules used values that were close approximations to the `Matlab`® values. The error results showed only a maximum of .0059% error for our arbitrary data set. The power, area, and delay section used the `Cadence`®software to attain the results for each module. It was noticed that as the delay increased the power consumption increased. In some cases, the smallest cell area occurred when the delay was at maximum. The synthesis section used the `Precision RTL`®software to target the `Xilinx`®Virtex-4 SX ML402 FPGA to attain area and speed for the different modules.

V. Conclusions

5.1 Summary of the Project

The problem was to convert Matlab® commands and functions into VHDL components. These commands are used in image processing such as Optical Flow. To fully implement Lucas-Kanade Optical Flow using these libraries may take up to six months. The foundation has been laid through the synthesizable reusable libraries. The development of these MSP modules have shortened the development time for an FPGA or ASIC design. The MSP modules were implemented, validated, and are fully synthesizable and functional reusable libraries. Chapter 2 discussed the VLSI Design Process and how long it takes to get the design completed. Architecture techniques were used in the development of the MSP libraries. The risks were discussed and developed for why FPGAs are the right solution for getting a design field quickly. Chapter 3 discussed the methodology and design structure that was used to aid in the creation of the MSP libraries. Chapter 4 discussed the analysis and results that were achieved from the design methodology. These libraries make it possible for the designer to save time, money, and resources in the VLSI Design Process. The modules that were created are reconfigurable to meet the needs the of the ever-changing Air Force. These modules can be used for future Air Force projects that require change detection, computer vision, pattern recognition, tracking targets, and image processing.

5.2 Future Work

The designs of the MSP Modules were shown to work successfully through simulations and were synthesized through the RTL software Cadence®and Precision RTL®software tools. The Cadence®software focused results on using the 250-nm technology library from the Taiwan Semiconductor Manufacturing Company. The Precision RTL®software tools targeted designs for the Xilinx®Virtex-4 SX ML402 FPGA. This section will discuss three future research topics.

5.2.1 Improve Module Designs. The modules designed are just the beginning. The great thing about this project is that it provided baseline modules that will be able to be enhanced for many projects to come. For example, the Reduce Matrix Function Module could have pipelines and parallization added to it to improve the delay of the Module. This would require that the Conv2 Module also have additional multipliers, controllers, pipelining, and parallization. The Memory Module could have enhanced features to accept multiple reads and writes during the same clock cycle. By breaking the Control Module into smaller control units it will assist in the parallization of how quickly the multiplications can be achieved. This will result in a faster delay, and the area of the design will grow larger due to the additional hardware. These are just a few examples; the possibilities are endless.

5.2.2 Power, Area, and Delay. Optimized power, area, and delay modules can be developed for each `Matlab`® command or function that makes up the Lucas-Kanade method. These additional modules will give the designer more flexibility and options to chose from for their next design. For example, their arsenal can have three different versions of a multiplier: Multiplier_Power Module, Multiplier_Area Module, and Multiplier_Delay Module. The designer will have the option to mix and match these modules together for future needs of the Air Force.

5.2.3 Complex Numbers. Currently, the modules are based on using fixed-point notation. Enhancements and variations can be made to handle floating point and complex numbers. A floating point number will give more range than using fixed-point. However, complex numbers can open the doors for potential uses such as Digital Signal Process (DSP), image processing, tracking targets, etc. When using DSP applications, complex numbers are used. DSP requires complex numbers to be able to work in the time and frequency domain using Laplace transforms.

Appendix A. Reducing Power Consumption

This section will discuss other ways to reduce power consumption for designing an architecture that requires low power.

A.1 Clock Speed/Clock Gating

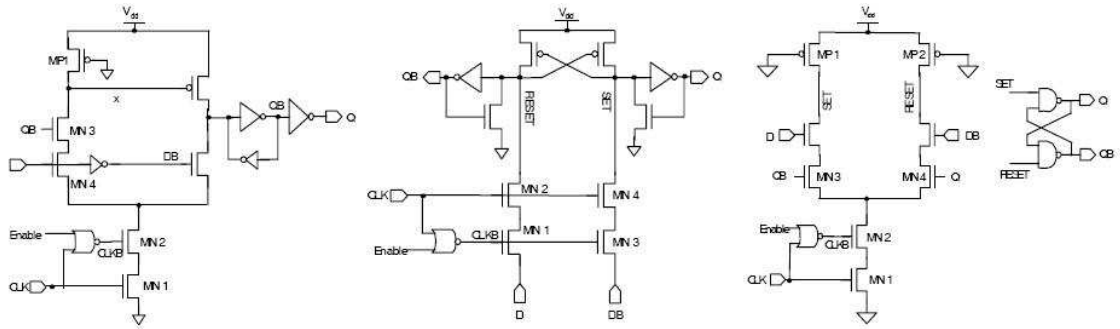
One of the six different options used to reduce the amount of power in a circuit is to reduce the speed of the clock. Reducing the speed of the clock for a circuit will reduce the switching activity. This results in power savings by limiting the amount of switching activity that has to take place. Besides reducing the speed of the clock for the circuit, designers have also proposed.

> Clock gating by modifying the design of the existing energy recovery clocked flip-flops to incorporate a power saving feature that eliminates any energy loss on the internal clock and other nodes of the flip-flops. Applying this approach of clock gating for a system that has 1000 flip-flops with 50% percent data switching eliminates the total power of the circuit by 47% and also found that during sleep mode the flip-flops reduce their power by (1000x) and has negligible amounts of overhead when the flip-flops are in active mode [13].

The proposed method that achieved these results [13] "inserted the gating feature inside the flip-flops themselves." This design can be seen in Figure A.1. Figure A.1a is Single-Ended Conditional Capturing Energy Recovery (SCCER), Figure A.1b is Static Differential Energy Recovery (SDER), and Figure A.1c is Differential Conditional Capturing Energy Recovery (DCCER) flip flop.

Another option to reducing the power of the clock is to have an asynchronous circuit design, also known as a clockless design [2]. This design eliminates the use of having a clock saving power. Arsalan and Shams state the following about asynchronous designs [2]:

> The asynchronous circuit design has been long regarded as a way for solving the problem of synchronous circuit design such as clock skew, worst case delay, and heavy global clock loading. When designed carefully, asynchronous circuits can be more power efficient as compared to their synchronous counterparts.



(a) Clock gating SCCER (b) Clock gating SDER (c) Clock gating DCCER

Figure A.1: Energy Recovery Clocked Flip-Flops with Clock Gating [13]

A.2 Turning Off The Circuit

A second option to reduce power consumption is to set up a circuit to have processes idle or turned off when not in use. This approach to reducing power consumption can be broken into three parts: timeout-based, predictive, and stochastic [4]. The timeout-based is self explanatory. When a task is not required to be used over a given time period it will timeout until it is required again. The predictive scheme can use an algorithm to predict when the circuit should turn different portions on and off, depending on it's requirements. The predictive scheme, or partially shutting down the system when it is not in use, will reduce the power consumption. Partial shutdown will put a section of the system into idle or sleep mode depending on the operation that is being performed. If a process is set into idle mode it will wake up without waiting to reboot. If a process is set into sleep mode it will be turned off when not in use. The downsides are that the idle mode will require more power to keep the system readily available and the sleep mode will take longer for the section of the circuit to transition to run mode. The third approach to reducing power consumption is stochastic. This "approach formulates policy optimization of Dynamic Power Management (DPM) as an optimization problem under uncertainty rather than trying to eliminate uncertainty by prediction" [4].

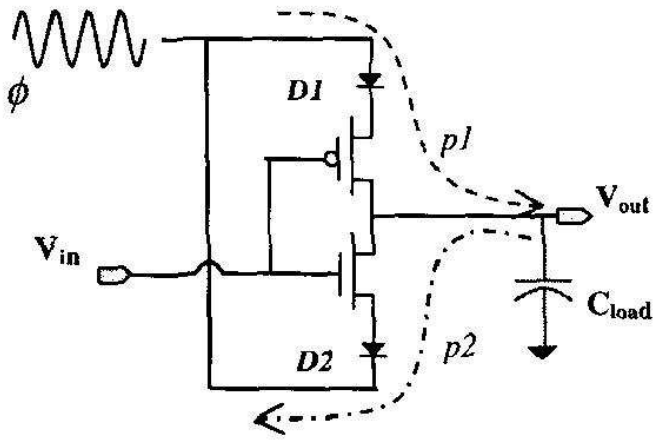


Figure A.2: Clock Recovery Circuit

A.3 Recycle Power

Thirdly, recycling the power in the circuit is another possible solution to reducing power consumption. The switching activity for a circuit is the major cause for power consumption by having the circuit constantly charging and discharging across the load and parasitic capacitance [16]. If you were to recycle the power of the circuit back into it's voltage supply you could save up to 80% power consumption [16], instead of sending the power discharging back into ground where it will be lost forever. Most circuits use Direct Current (DC) power supplies to turn them on. If you use Alternating Current (AC) power supplies instead you can recycle the power for your circuit. This approach can be seen in Figure A.2 [16] using two diodes and P channel and N channel transistors to keep the load from discharging. As the power source supplies a sinusoidal waveform the signal is 0 volts and the P channel transistor is turned on and begins charging the load capacitor. This causes the diode connected to the P channel to be just below the Peak Voltage (Vp) by one diode. During all this, the N channel transistor is turned off. The diode that is connected to the P channel transistor holds the load voltage to Vp. During the Vp of the sinusoidal waveform the N channel transistor is turned on and the P channel transistor is turned off. The P and N channel transistors are turned off during the negative cycle of the sinusoidal signal [16].

A.4 Redesign Logic Of The Circuit

Another option is to reduce the amount of floating point operations in the circuit by replacing them with fixpoint representation. It's even possible to have a trade off between parallel and serial logic design for the circuit. Power can be saved by using parallization, pipelining, or a systolic architecture approach. Parallization gives you the option to be able to slow down the clock speed to still compute data through the circuit. The slowing down of the clock speed reduces the clock switching resulting in power savings. For example, a sequential design requires a clock that will result in continuous switching activity that will cause the power consumption to increase over a design that are combinational. In a sequential circuit, adders and multipliers are a good example of how additional switching activity will increase power consumption. Reducing the power consumption for a sequential circuit could reduce the clock speed of the circuit.

A.5 Higher Level of Integration

Changing the way the circuit is integrated can help reduce power. Performing a System on Chip approach, non-cmos, or even reducing the lamda size of the circuit from 250 nm to 90 nm, for example, can reduce the power. Reducing the technology size of design reduces the Threshold Voltage (Vth) required to turn on the transistors. As seen in the Equation A.1, reducing the voltage or current will cause the power usage to go down. P represents power (P), V represents voltage (V), and I stands for Current (I). Lowering the Vth decreases power, resulting in a power savings.

$$P = VI \tag{A.1}$$

A.6 Dynamic Power Management

Finally, dynamic power management is another approach to reduce power consumption in a circuit design. It has been stated [14] that "Dynamic power management is one of the most popular and successful low power design techniques in

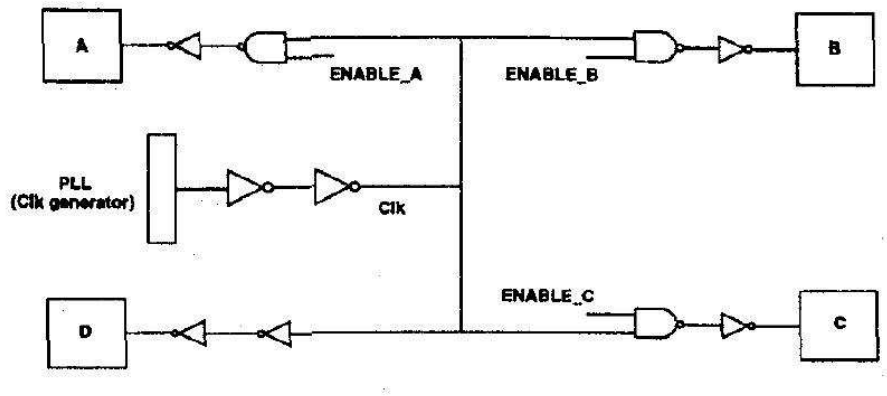


Figure A.3: Illustration of Clock Gating

commercial integrated circuits". This approach to saving power is built around the idea of conserving power in the circuit for activities that don't need to be operating. The approach is also easy to build into a circuit design or even older design structures. One of the best features of using DPM is that in most cases the area of the system will not increase much in size and the performance will not be reduced. The mechanisms for DPM are a collection of techniques using the following areas in the circuit to reduce the power consumption: Clock Gating, Qualified System Latches, Guarded Evaluation, Bus Deactivation, and Self-timed Techniques. Now lets take a look at each one of these power saving strategies in further detail.

A.6.1 Clock Gating. Clock Gating was touched on early but it's still important to note that when dealing with circuits or microprocessors that high performance is typically trying to be achieved. Performance in a design requires that the clock of the system operate as quickly as possible. This in turn causes the most power consumption for the system [14] by the amount of switching activity that is taking place. Also, the clock tree for a system puts an additional load on the system causing increased power consumption. The idea is to eliminate unnecessary activity on segments of the clock signal by gating these with special qualifying signals. This can been seen in Figure A.3.

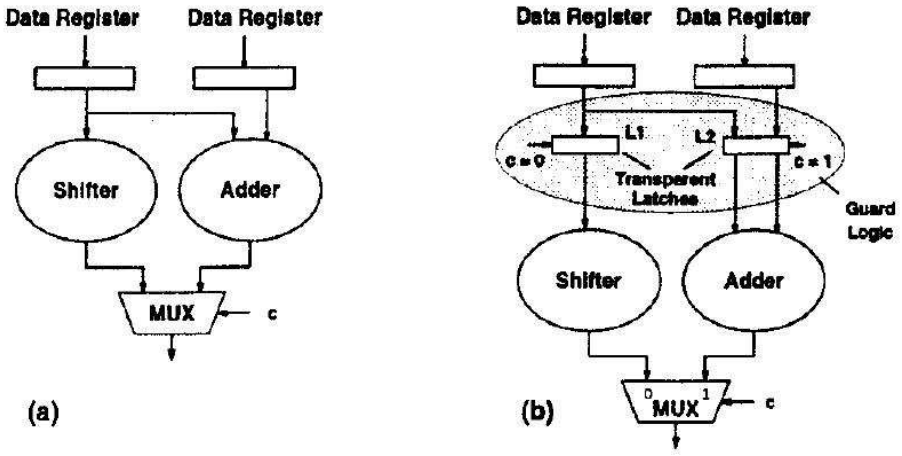


Figure A.4: Guarded evaluation for a dual function ALU: (a) Original design, (b) Guarded design

A.6.2 Qualified System Latches. According to Tiwari [14] "regular system latches have a data input and a control input that is usually connected to the system clock. A qualified latch has an additional input, the "enable" input, which determines if the inputs to the latch will be read or not when the clock is asserted." The use of these latches can have a better power savings than regular latches. The reasoning behind this is that the regular latches use power every time the system clock is switching low to high, even when the data is not changing across the latch.

A.6.3 Guarded Evaluation. From the previous two subsections you saw how the clock on the system requires large amounts of power to operate. Guarded evaluation comes into play when you are unable to slow the clock or can not make any further corrections to reduce the amount of power the clock is drawing. For instance, [14] "if the combinational block at the output of a system latch is known to be doing no useful work, power can be canned by disabling transitions from entering the block." It can also be beneficial if you have a set of [14] "modules that share a set of inputs". Being able to turn on only the module that is used will save power from going to a module that is not being used. This is seen in Figure A.4.

A.6.4 Bus Deactivation. According to Tiwari [14] "the idea is to restrict the enabling condition on tristate buffers driving a bus, such that the bus is not driven on cycles when it is known that its results will not be used. Such unnecessary activity on heavily loaded buses can lead to significant power wastage."

A.6.5 Self-timed Techniques. Self-timed techniques are useful with on-chip memories to reduce power. When dealing with microprocessors the memory can cause large power consumption as data is being transferred back and forth to the cache with on-chip memories systems. "A low-power optimization is to pre-charge the bit lines and activate the sense-amps using short pulses. These pulses are generated only when changes are detected on the address lines and a read or write is going to be initiated [14]."

A.7 Synthesize

The synthesis approach in this research will be using Cadence Software CAD tools. The paper [11] "Leakage Power Optimization Flow", written by Sirsi, discusses three different Power Flow models. The Power Flow model that worked the best for reducing power consumption for clock speed of 200Mhz, technology 130 nm, and cell instances 120K was his Power Flow 3 which uses a RTL Compiler and System on Chip Encounter. Siri's power flow model is illustrated in Figure A.5. In general, the flow taken from Cadence's support documentation has the block diagrams broken into simpler blocks. The most important thing for a good design is the design flow used for the system. Writing a good script to address these challenges plays a large role with using the Cadence Tools. The first step is to set up your environment settings along with what technology library that will be used. Load the VHDL file and perform elaboration on the design. Apply constraints such as timing, design rule constraints, defining input and output delay, and etc. Once these have been set it's a good idea to add optimization setting to the design. Now you're ready for the final steps of the script to synthesize, perform analysis, and export the design as a RTL file.

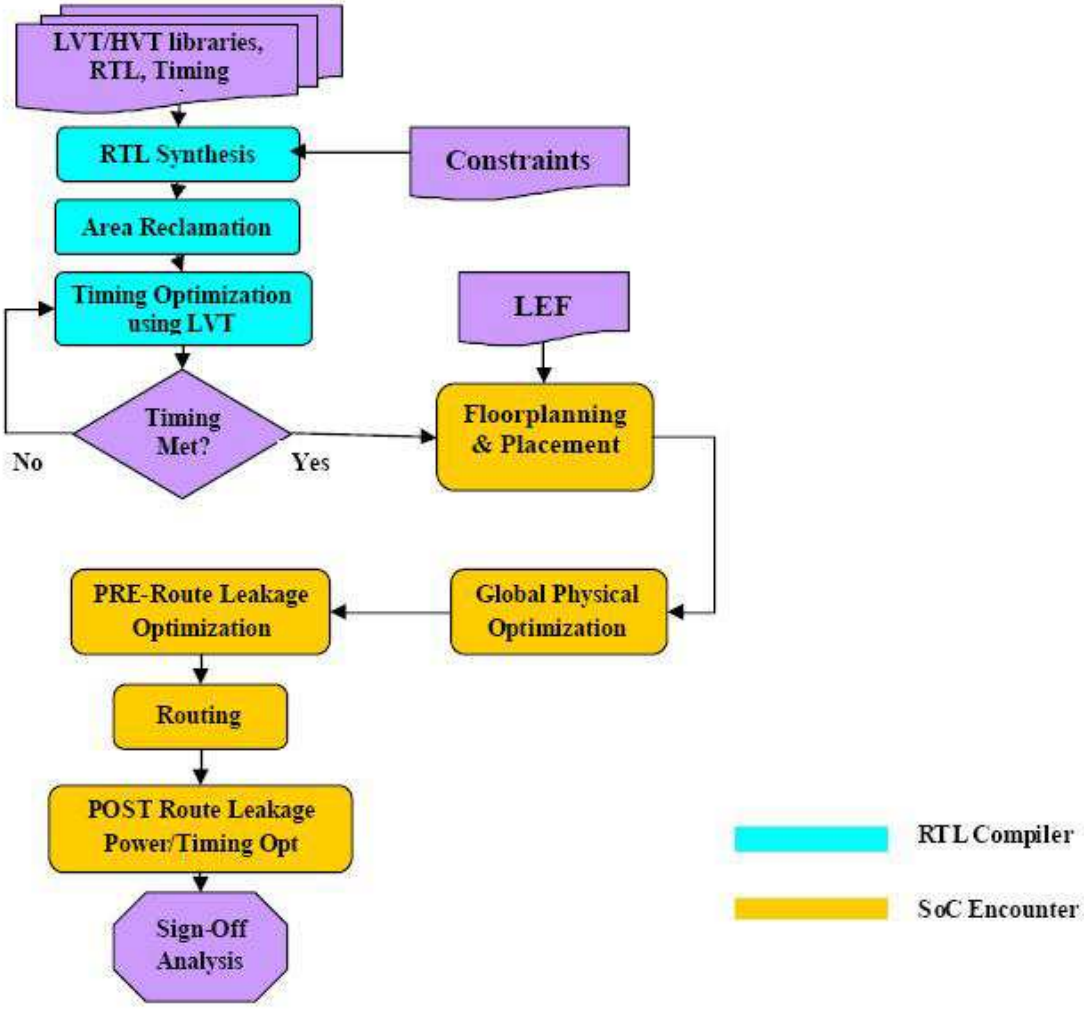


Figure A.5: Leakage Power Optimization Flow

Appendix B. Matlab® Code

This section has the Matlab® code that was written by Sohai Khan [8] for Optical Flow using the Lucas-Kanade method.

B.1 HierarchicalLK

Listing B.1: The HierarchicalLK.m Matlab® file.(appendix2/HierarchicalLK.m)

```
function [u,v,cert] = HierarchicalLK(im1, im2, numLevels, ...
    windowSize, iterations, display)
% HIERARCHICALLK        Hierarchical Lucas Kanade
% (using pyramids)
% [u,v]=HierarchicalLK(im1, im2, numLevels, windowSize, iterations...
    , display)
% Tested for pyramids of height 1, 2, 3 only...
% operation with pyramids of height 4 might be unreliable
%
%        Use quiver(u, -v, 0) to view the results
%
% NUMLEVELS  Pyramid Levels (typical value 3)
% WINDOWSIZE Size of smoothing window (typical value 1-4)
% ITERATIONS   number of iterations (typical value 1-5)
% DISPLAY      1 to display flow fields (1 or 0)
%
%Uses: Reduce, Expand
%
% Sohaib Khan
%   edited 05-15-03 (Yaser)
% yaser@cs.ucf.edu
%
% [1]   B.D. Lucas and T. Kanade, "An Iterative Image Registration
% technique, with an Application to Stero Vision," Int'l Joint
% Conference Artifical Intelligence, pp. 121-130, 1981.

if (size(im1,1)~=size(im2,1)) | (size(im1,2)~=size(im2,2))
    error('images are not same size');
end;

if (size(im1,3) ~= 1) | (size(im2, 3) ~= 1)
    error('input should be gray level images');
end;


% check image sizes and crop if not divisible
if (rem(size(im1,1), 2^(numLevels - 1)) ~= 0)
    warning('image will be cropped in height, size of output will ...
        be smaller than input!');
    im1 = im1(1:(size(im1,1) - rem(size(im1,1), 2^(numLevels - 1))...
        ), :);
    im2 = im2(1:(size(im1,1) - rem(size(im1,1), 2^(numLevels - 1))...
        ), :);
```

```
39 end;
40
41 if (rem(size(im1,2), 2^(numLevels - 1)) ~= 0)
42     warning('image will be cropped in width, size of output will ...
           be smaller than input!');
43     im1 = im1(:, 1:(size(im1,2) - rem(size(im1,2), 2^(numLevels - ...
           1))));
44     im2 = im2(:, 1:(size(im1,2) - rem(size(im1,2), 2^(numLevels - ...
           1))));
45 end;
46
47 %Build Pyramids
48 pyramid1 = im1;
49 pyramid2 = im2;
50
51 for i=2:numLevels
52     im1 = reduce(im1);
53     im2 = reduce(im2);
54     pyramid1(1:size(im1,1), 1:size(im1,2), i) = im1;
55     pyramid2(1:size(im2,1), 1:size(im2,2), i) = im2;
56 end;
57
58 % base level computation
59 disp('Computing Level 1');
60 baseIm1 = pyramid1(1:(size(pyramid1,1)/(2^(numLevels-1))), 1:(size...
       (pyramid1,2)/(2^(numLevels-1))), numLevels);
61 baseIm2 = pyramid2(1:(size(pyramid2,1)/(2^(numLevels-1))), 1:(size...
       (pyramid2,2)/(2^(numLevels-1))), numLevels);
62 [u,v] = LucasKanade(baseIm1, baseIm2, windowSize);
63
64 for r = 1:iterations
65     [u, v] = LucasKanadeRefined(u, v, baseIm1, baseIm2);
66 end
67
68 %propagating flow 2 higher levels
69 for i = 2:numLevels
70     disp(['Computing Level ', num2str(i)]);
71     uEx = 2 * imresize(u,size(u)*2);   % use appropriate expand ...
           function (gaussian, bilinear, cubic, etc).
72     vEx = 2 * imresize(v,size(v)*2);
73
74     curIm1 = pyramid1(1:(size(pyramid1,1)/(2^(numLevels - i))), ...
           1:(size(pyramid1,2)/(2^(numLevels - i))), (numLevels - i)...
           +1);
75     curIm2 = pyramid2(1:(size(pyramid2,1)/(2^(numLevels - i))), ...
           1:(size(pyramid2,2)/(2^(numLevels - i))), (numLevels - i)...
           +1);
76
77     [u, v] = LucasKanadeRefined(uEx, vEx, curIm1, curIm2);
78
79     for r = 1:iterations
80         [u, v, cert] = LucasKanadeRefined(u, v, curIm1, curIm2);
```

```
81      end
82  end;
83
84  if (display==1)
85      figure, quiver(reduce((reduce(medfilt2(flipud(u),[5 5])))), -...
            reduce((reduce(medfilt2(flipud(v),[5 5])))), 0), axis equal
86  end
```

B.2 Reduce

Listing B.2: The Reduce.m Matlab® file.(appendix2/Reduce.m)

```
1  function smallIm = Reduce(im)
2  % REDUCE        Compute smaller layer of Gaussian Pyramid
3
4  % Sohaib Khan, Feb 16, 2000
5
6  %Algo
7  %Gaussian mask = [0.05 0.25 0.4 0.25 0.05]
8  % Apply 1d mask to alternate pixels along each row of image
9  % apply 1d mask to each pixel along alternate columns of
10 % resulting image
11
12
13 mask = [0.05 0.25 0.4 0.25 0.05];
14
15 hResult = conv2(im, mask);
16 hResult = hResult(:,3:size(hResult,2)-2);
17 hResult = hResult(:, 1:2:size(hResult,2));
18
19 vResult = conv2(hResult, mask');
20 vResult = vResult(3:size(vResult,1)-2, :);
21 vResult = vResult(1:2:size(vResult,1),:);
22
23 smallIm = vResult;
```

B.3 LucasKanade

Listing B.3: The LucasKanade.m Matlab® file.(appendix2/LucasKanade.m)

```
1  function [u, v] = LucasKanade(im1, im2, windowSize);
2  % LucasKanade  lucas kanade algorithm, without pyramids
3  % (only 1 level);
4
5  % REVISION: NaN vals are replaced by zeros
6
7  [fx, fy, ft] = ComputeDerivatives(im1, im2);
8
9  u = zeros(size(im1));
10 v = zeros(size(im2));
```

```
11
12 halfWindow = floor(windowSize/2);
13 for i = halfWindow+1:size(fx,1)-halfWindow
14    for j = halfWindow+1:size(fx,2)-halfWindow
15       curFx = fx(i-halfWindow:i+halfWindow, j-halfWindow:j+...
            halfWindow);
16       curFy = fy(i-halfWindow:i+halfWindow, j-halfWindow:j+...
            halfWindow);
17       curFt = ft(i-halfWindow:i+halfWindow, j-halfWindow:j+...
            halfWindow);
18
19       curFx = curFx';
20       curFy = curFy';
21       curFt = curFt';
22
23
24       curFx = curFx(:);
25       curFy = curFy(:);
26       curFt = -curFt(:);
27
28       A = [curFx curFy];
29
30       U = pinv(A'*A)*A'*curFt;
31
32       u(i,j)=U(1);
33       v(i,j)=U(2);
34    end;
35 end;
36
37 u(isnan(u))=0;
38 v(isnan(v))=0;
39
40 %u=u(2:size(u,1), 2:size(u,2));
41 %v=v(2:size(v,1), 2:size(v,2));
42
43 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
44 function [fx, fy, ft] = ComputeDerivatives(im1, im2);
45 %ComputeDerivatives Compute horizontal, vertical and time
46 %derivative between two gray-level images.
47
48 if (size(im1,1) ~= size(im2,1)) | (size(im1,2) ~= size(im2,2))
49    error('input images are not the same size');
50 end;
51
52 if (size(im1,3)~=1) | (size(im2,3)~=1)
53    error('method only works for gray-level images');
54 end;
55
56
57 fx = conv2(im1,0.25* [-1 1; -1 1]) + conv2(im2, 0.25*[-1 1; -1 1])...
    ;
```

```
fy = conv2(im1, 0.25*[-1 -1; 1 1]) + conv2(im2, 0.25*[-1 -1; 1 1])...
    ;
ft = conv2(im1, 0.25*ones(2)) + conv2(im2, -0.25*ones(2));

% make same size as input
fx=fx(1:size(fx,1)-1, 1:size(fx,2)-1);
fy=fy(1:size(fy,1)-1, 1:size(fy,2)-1);
ft=ft(1:size(ft,1)-1, 1:size(ft,2)-1);
```

B.4 LucasKanadeRefined

Listing B.4: The LucasKanadeRefined.m Matlab® file.
(appendix2/LucasKanadeRefined.m)

```
function [u,v,cert] = LucasKanadeRefined(uIn, vIn, im1, im2);
% Lucas Kanade Refined computes lucas kanade flow at the current
% level given previous estimates current implementation is only
% for a 3x3 window


%[fx, fy, ft] = ComputeDerivatives(im1, im2);

uIn = round(uIn);
vIn = round(vIn);
%uIn = uIn(2:size(uIn,1), 2:size(uIn, 2)-1);
%vIn = vIn(2:size(vIn,1), 2:size(vIn, 2)-1);


u = zeros(size(im1));
v = zeros(size(im2));

% to compute derivatives, use a 5x5 block...
% the resulting derivative will be 5x5...
% take the middle 3x3 block as derivative
for i = 3:size(im1,1)-2
   for j = 3:size(im2,2)-2
    %  if uIn(i,j)~=0
    %     disp('ha');
    %  end;

      curIm1 = im1(i-2:i+2, j-2:j+2);
      lowRindex = i-2+vIn(i,j);
      highRindex = i+2+vIn(i,j);
      lowCindex = j-2+uIn(i,j);
      highCindex = j+2+uIn(i,j);

      if (lowRindex < 1)
         lowRindex = 1;
         highRindex = 5;
      end;

```

```
38        if (highRindex > size(im1,1))
39           lowRindex = size(im1,1)-4;
40           highRindex = size(im1,1);
41        end;
42
43        if (lowCindex < 1)
44           lowCindex = 1;
45           highCindex = 5;
46        end;
47
48        if (highCindex > size(im1,2))
49           lowCindex = size(im1,2)-4;
50           highCindex = size(im1,2);
51        end;
52
53        if isnan(lowRindex)
54           lowRindex = i-2;
55           highRindex = i+2;
56        end;
57
58        if isnan(lowCindex)
59           lowCindex = j-2;
60           highCindex = j+2;
61        end;
62
63
64
65        curIm2 = im2(lowRindex:highRindex, lowCindex:highCindex);
66
67        [curFx, curFy, curFt]=ComputeDerivatives(curIm1, curIm2);
68        curFx = curFx(2:4, 2:4);
69        curFy = curFy(2:4, 2:4);
70        curFt = curFt(2:4, 2:4);
71
72        curFx = curFx';
73        curFy = curFy';
74        curFt = curFt';
75
76
77        curFx = curFx(:);
78        curFy = curFy(:);
79        curFt = -curFt(:);
80
81        A = [curFx curFy];
82
83        U = pinv(A'*A)*A'*curFt;
84
85        u(i,j)=U(1);
86        v(i,j)=U(2);
87
88        cert(i,j) = rcond(A'*A);
89
```

```
90      end;
91  end;
92
93  u = u+uIn;
94  v = v+vIn;
95
96
97  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
98  function [fx, fy, ft] = ComputeDerivatives(im1, im2);
99  %ComputeDerivatives Compute horizontal, vertical and time
100 %derivative between two gray-level images.
101
102 if (size(im1,1) ~= size(im2,1)) | (size(im1,2) ~= size(im2,2))
103     error('input images are not the same size');
104 end;
105
106 if (size(im1,3)~=1) | (size(im2,3)~=1)
107     error('method only works for gray-level images');
108 end;
109
110
111 fx = conv2(im1,0.25* [-1 1; -1 1]) + conv2(im2, 0.25*[-1 1; -1 1])...
        ;
112 fy = conv2(im1, 0.25*[-1 -1; 1 1]) + conv2(im2, 0.25*[-1 -1; 1 1])...
        ;
113 ft = conv2(im1, 0.25*ones(2)) + conv2(im2, -0.25*ones(2));
114
115 % make same size as input
116 fx=fx(1:size(fx,1)-1, 1:size(fx,2)-1);
117 fy=fy(1:size(fy,1)-1, 1:size(fy,2)-1);
118 ft=ft(1:size(ft,1)-1, 1:size(ft,2)-1);
```

B.5 Expand

Listing B.5: The Expand.m Matlab® file.(appendix2/Expand.m)

```
1  function largeIm = Expand(im);
2  %EXPAND Compute large layer of Gaussian pyramid
3
4  % Sohaib Khan, Feb 16, 2000
5
6  %Algo
7  % Gaussian mask = [0.05 0.25 0.4 0.25 0.05]
8  % Insert zeros in every alternate row position and conv with mask
9  % insert zeros in every alternate clmn position in result
10 % and conv with mask'
11
12 mask = 2*[0.05 0.25 0.4 0.25 0.05];
13 % factor of 2 is there because
14 % each pixel gets contribution either
15 % from 0.05, 0.4, 0.05  or from 0.25, 0.25
```

```
16
17 % insert zeros in every alternate position in each row
18 rowZeros = [im; zeros(size(im))];
19 rowZeros = reshape(rowZeros, size(im,1), 2*size(im,2));
20
21 %conv with horiz mask
22 newIm = conv2(rowZeros, mask);
23 newIm = newIm(:,3:size(newIm,2)-2);
24
25 % insert zeros in every alternate position in each col
26 colZeros = newIm';
27 colZeros = [colZeros; zeros(size(colZeros))];
28 colZeros = reshape(colZeros, size(colZeros,1)/2, 2*size(colZeros...
      ,2));
29 colZeros = colZeros';
30
31 largeIm=conv2(colZeros, mask');
32 largeIm=largeIm(3:size(largeIm,1)-2,:);
```

This section has the Top-Level Designs that were coded in VHDL to convert the Matlab® commands and functions.

C.1 Reduce Matrix Function Top-Level 1 Behavior Multiplier

Listing C.1: The ReduceConv2Module.vhd VHDL file. (appendix3/ReduceConv2Module.vhd)

```
1  -- Capt. Jason Shirley
2  -- This is the Reduce Matrix Function Top-Level Design 1 using the
3  -- Behavior Multiplier. These Modules are structually connected.
4
5
6  library ieee;
7  use ieee.std_logic_1164.all;
8  use ieee.std_logic_unsigned.all;
9  use ieee.std_logic_arith.all;
10 use ieee.numeric_std.all;
11
12 ----------------------------------------------------------------------
13
14 entity reduceconv2_module is
15
16    generic (address_width: integer := 8;
17             data_width: integer := 24);
18
19    port (clk                          : in std_logic;
20          reset                        : in std_logic;
21          enable_matrixA               : in std_logic;
22          enable_conv2_module          : in std_logic;
23          read_address_matrixA         : in std_logic_vector (...
               address_width - 1 downto 0);
24          read_address_matrixReduce    : in std_logic_vector (...
               address_width - 1 downto 0);
25          input_matrixA_values         : in std_logic_vector (...
               data_width - 1 downto 0);
26          output_data_matrixC          : out std_logic_vector(...
               data_width - 1 downto 0);
27          reduce_module_finished       : out std_logic;
28          input_to_matrixC_hresult     : out std_logic_vector(...
               data_width - 1 downto 0);
29          writeEnable_matrixC_hresult  : out std_logic;
30          address_matrixA_hresult      : out std_logic_vector(...
               address_width - 1 downto 0);
31          address_matrixB_hresult      : out std_logic_vector(...
               address_width - 1 downto 0);
32          address_matrixC_hresult      : out std_logic_vector(...
               address_width - 1 downto 0);
33          matrixA_output_hresult       : out std_logic_vector(...
               data_width - 1 downto 0);
```

```
34          matrixB_output_hresult      : out std_logic_vector(...
               data_width - 1 downto 0);
35          control_adder_signal_hresult: out std_logic;
36            input_to_matrixC_vresult    : out std_logic_vector(...
                 data_width - 1 downto 0);
37          writeEnable_matrixC_vresult : out std_logic;
38          address_matrixA_vresult     : out std_logic_vector(...
               address_width - 1 downto 0);
39          address_matrixB_vresult     : out std_logic_vector(...
               address_width - 1 downto 0);
40          address_matrixC_vresult     : out std_logic_vector(...
               address_width - 1 downto 0);
41          matrixA_output_vresult      : out std_logic_vector(...
               data_width - 1 downto 0);
42          matrixB_output_vresult      : out std_logic_vector(...
               data_width - 1 downto 0);
43          control_adder_signal_vresult: out std_logic
44            );
45
46 end reduceconv2_module;
47
48 ----------------------------------------------------------------------
49
50 architecture structure_reduceconv2_module of reduceconv2_module is
51
52 component hresultconv2_module is
53
54    generic (address_width: integer := 8;
55             data_width: integer := 24);
56
57    port (clk                    : in std_logic;
58          reset                  : in std_logic;
59          enable_matrixA         : in std_logic;
60          enable_conv2_module    : in std_logic;
61          read_address_matrixA   : in std_logic_vector (...
               address_width - 1 downto 0);
62          read_address_matrixC   : in std_logic_vector (...
               address_width - 1 downto 0);
63          input_matrixA_values   : in std_logic_vector (data_width -...
                1 downto 0);
64          output_data_matrixC    : out std_logic_vector(data_width -...
                1 downto 0);
65          enable_reduce          : out std_logic;
66          input_to_matrixC       : out std_logic_vector(data_width -...
                1 downto 0);
67          writeEnable_matrixC    : out std_logic;
68          address_matrixA        : out std_logic_vector(...
               address_width - 1 downto 0);
69          address_matrixB        : out std_logic_vector(...
               address_width - 1 downto 0);
70          address_matrixC        : out std_logic_vector(...
               address_width - 1 downto 0);
```

```
71          matrixA_output         : out std_logic_vector(data_width -...
                1 downto 0);
72          matrixB_output         : out std_logic_vector(data_width -...
                1 downto 0);
73          control_adder_signal   : out std_logic
74               );
75
76  end component hresultconv2_module;
77
78  ------------------------------------------------------------------
79
80  component hresultreducecontrol is
81
82          generic (address_width: integer := 8;
83                   n1: integer := 5;
84                   n2: integer := 11);
85  -- address_width must be an even number
86  -- The hResultReduceControl module reduces the orignal 6x8
87  --input matrix in half from the number of columns from the
88  -- hResultControl module. Example 6 x 12 will be reduce down
89  -- to 6x4 n1 is the row position and n2 is the column position.
90  -- you want n1 and n2 to be one size smaller then the 6 x 12
91  -- therefore n1 should equal = 5 since you start at zero
92  -- and n2 should equal = 11 since you start at zero
93  -- The hResultReduceControl module reduces the orignal 6x8
94  -- input matrix in half from the number of columns from the
95  -- hResultControl module. Example 6 x 12 will be reduce down
96  -- to 6x4 n1 is the row position and n2 is the column position.
97  -- you want n1 and n2 to be one size smaller then the 6 x 12
98  -- therefore n1 should equal = 5 since you start at zero
99  -- and n2 should equal = 11 since you start at zero
100
101    port (clk                          : in std_logic;
102          reset                        : in std_logic;
103          enable_control               : in std_logic;
104          writeEnable_matrix           : out std_logic;
105          enable_vresult               : out std_logic;
106          read_address_hResult         : out std_logic_vector (...
                address_width - 1 downto 0);
107          write_address_reducehResult: out std_logic_vector (...
                address_width - 1 downto 0)
108          );
109
110 end component hresultreducecontrol;
111 ------------------------------------------------------------------
112
113 component vresultconv2_module is
114
115    generic (address_width: integer := 8;
116             data_width: integer := 24);
117
118    port (clk                    : in std_logic;
```

```
119         reset                 : in std_logic;
120         enable_matrixA        : in std_logic;
121         enable_conv2_module   : in std_logic;
122         read_address_matrixA  : in std_logic_vector (...
               address_width - 1 downto 0);
123         read_address_matrixC  : in std_logic_vector (...
               address_width - 1 downto 0);
124         input_matrixA_values  : in std_logic_vector (data_width -...
                1 downto 0);
125         output_data_matrixC   : out std_logic_vector(data_width -...
                1 downto 0);
126         enable_reduce         : out std_logic;
127         input_to_matrixC      : out std_logic_vector(data_width -...
                1 downto 0);
128         writeEnable_matrixC   : out std_logic;
129         address_matrixA       : out std_logic_vector(...
               address_width - 1 downto 0);
130         address_matrixB       : out std_logic_vector(...
               address_width - 1 downto 0);
131         address_matrixC       : out std_logic_vector(...
               address_width - 1 downto 0);
132         matrixA_output        : out std_logic_vector(data_width -...
                1 downto 0);
133         matrixB_output        : out std_logic_vector(data_width -...
                1 downto 0);
134         control_adder_signal  : out std_logic
135          );
136
137 end component vresultconv2_module;
138
139 ------------------------------------------------------------------
140 component vresultreducecontrol is
141
142        generic (address_width: integer := 8;
143                 n1: integer := 9;
144                 n2: integer := 3);
145 -- address_width must be an even number
146    port (clk                       : in std_logic;
147         reset                      : in std_logic;
148         enable_control             : in std_logic;
149         writeEnable_matrix         : out std_logic;
150         enable_vresult             : out std_logic;
151         read_address_vResult       : out std_logic_vector (...
               address_width - 1 downto 0);
152         write_address_reducevResult: out std_logic_vector (...
               address_width - 1 downto 0)
153         );
154
155 end component vresultreducecontrol;
156
157 ------------------------------------------------------------------
158 component mem_matrix is
```

```
159
160         generic (address_width: integer := 8;
161                  data_width: integer := 24);
162
163    port (clk                    : in std_logic;
164          reset                  : in std_logic;
165          writeEnable            : in std_logic;
166               matrix_values          : in std_logic_vector (...
                    data_width - 1 downto 0);
167               write_address          : in std_logic_vector (...
                    address_width - 1 downto 0);
168               read_address           : in std_logic_vector (...
                    address_width - 1 downto 0);
169               output_matrix_values : out std_logic_vector(...
                    data_width - 1 downto 0)
170               );
171
172 end component mem_matrix;
173
174 ---------------------------------------------------------------------
175 signal enable_reduce_h_to_enable_control, ...
       writeEnable_matrix_to_enable_matrixA: std_logic;
176 signal enable_vresult_to_enable_conv2_module, ...
       enable_reduce_v_to_enable_control: std_logic;
177 signal writeEnable_matrix_to_writeEnable: std_logic;
178 signal write_address_reducevResult_to_write_address:...
       std_logic_vector (address_width - 1 downto 0);
179 signal read_address_hResult_to_read_address_matrixC:...
       std_logic_vector (address_width - 1 downto 0);
180 signal read_addressvResult_to_read_address_matrixC:...
       std_logic_vector (address_width - 1 downto 0);
181 signal write_address_reducehResult_to_read_address_matrixA:...
       std_logic_vector (address_width - 1 downto 0);
182 signal output_data_matrixC_to_input_matrixA_values: ...
       std_logic_vector(data_width - 1 downto 0);
183 signal output_data_matrixC_to_matrix_values: std_logic_vector(...
       data_width - 1 downto 0);
184
185 begin
186 hresultconv2_module1: hresultconv2_module port map (clk => clk,
187 reset => reset,
188 enable_matrixA => enable_matrixA,
189 enable_conv2_module => enable_conv2_module,
190 read_address_matrixA => read_address_matrixA,
191 read_address_matrixC =>
192 read_address_hResult_to_read_address_matrixC,
193 input_matrixA_values => input_matrixA_values,
194 output_data_matrixC =>
195 output_data_matrixC_to_input_matrixA_values,
196 enable_reduce  => enable_reduce_h_to_enable_control,
197 input_to_matrixC => input_to_matrixC_hresult,
198 writeEnable_matrixC => writeEnable_matrixC_hresult,
```

```
199 address_matrixA => address_matrixA_hresult,
200 address_matrixB => address_matrixB_hresult,
201 address_matrixC => address_matrixC_hresult,
202 matrixA_output => matrixA_output_hresult,
203 matrixB_output => matrixB_output_hresult,
204 control_adder_signal => control_adder_signal_hresult);
205
206 hresultreducecontrol1: hresultreducecontrol port map(clk => clk,
207 reset => reset,
208 enable_control => enable_reduce_h_to_enable_control,
209 writeEnable_matrix=> writeEnable_matrix_to_enable_matrixA,
210 enable_vresult => enable_vresult_to_enable_conv2_module,
211 read_address_hResult =>
212 read_address_hResult_to_read_address_matrixC,
213 write_address_reducehResult =>
214 write_address_reducehResult_to_read_address_matrixA);
215
216 vresultconv2_module1: vresultconv2_module port map(clk => clk,
217 reset => reset,
218 enable_matrixA => writeEnable_matrix_to_enable_matrixA,
219 enable_conv2_module => enable_vresult_to_enable_conv2_module,
220 read_address_matrixA =>
221 write_address_reducehResult_to_read_address_matrixA,
222 read_address_matrixC =>
223 read_addressvResult_to_read_address_matrixC,
224 input_matrixA_values =>
225 output_data_matrixC_to_input_matrixA_values,
226 output_data_matrixC => output_data_matrixC_to_matrix_values,
227 enable_reduce => enable_reduce_v_to_enable_control,
228 input_to_matrixC => input_to_matrixC_vresult,
229 writeEnable_matrixC => writeEnable_matrixC_vresult,
230 address_matrixA => address_matrixA_vresult,
231 address_matrixB => address_matrixB_vresult,
232 address_matrixC => address_matrixC_vresult,
233 matrixA_output => matrixA_output_vresult,
234 matrixB_output => matrixB_output_vresult,
235 control_adder_signal => control_adder_signal_vresult);
236
237 vresultreducecontrol1 :vresultreducecontrol port map(clk => clk,
238 reset => reset,
239 enable_control => enable_reduce_v_to_enable_control,
240 writeEnable_matrix => writeEnable_matrix_to_writeEnable,
241 enable_vresult => reduce_module_finished,
242 read_address_vResult =>
243 read_addressvResult_to_read_address_matrixC,
244 write_address_reducevResult =>
245 write_address_reducevResult_to_write_address);
246
247 mem_matrixReduce: mem_matrix port map (clk => clk,
248 reset => reset,
249 writeEnable  => writeEnable_matrix_to_writeEnable,
250 matrix_values => output_data_matrixC_to_matrix_values,
```

```
write_address => write_address_reducevResult_to_write_address,
read_address => read_address_matrixReduce,
output_matrix_values => output_data_matrixC);

end structure_reduceconv2_module;
```

C.2 Reduce Matrix Function Top-Level 2 Behavior Multiplier

Listing C.2: The ReduceConv2Module2.vhd VHDL file. (appendix3/ReduceConv2Module2.vhd)

```
-- Capt. Jason Shirley
-- This is the Reduce Matrix Function Top-Level Design 2 using the
-- Behavior Multiplier. These Modules are structually connected.


library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
use ieee.numeric_std.all;

--------------------------------------------------------------------

entity reduceconv2_module2 is

   generic (address_width: integer := 8;
            data_width: integer := 24);

   port (clk                       : in std_logic;
         reset                     : in std_logic;
         enable_matrixA            : in std_logic;
         enable_conv2_module       : in std_logic;
         read_address_matrixA      : in std_logic_vector (...
            address_width - 1 downto 0);
         read_address_matrixReduce: in std_logic_vector (...
            address_width - 1 downto 0);
         input_matrixA_values      : in std_logic_vector (...
            data_width - 1 downto 0);
         reduce_module_finished    : out std_logic;
         output_data_matrixReduce : out std_logic_vector(...
            data_width - 1 downto 0)
           );

end reduceconv2_module2;

--------------------------------------------------------------------

architecture structure_reduceconv2_module2 of reduceconv2_module2 ...
   is

```

```
36 component hresultcontrol2 is
37
38    generic (address_width: integer := 8;
39             n1: integer := 5;
40                  n2: integer := 11);
41 -- address_width must be an even number
42
43         port (clk                    : in std_logic;
44                 reset                  : in std_logic;
45                 enable_control         : in std_logic;
46                 update_reg_enable      : out std_logic;
47                 writeEnable_matrixC    : out std_logic;
48                 reset_reg              : out std_logiC;
49                 enable_reduce          : out std_logic;
50                 read_address_matrixA   : out std_logic_vector (...
                      address_width - 1 downto 0);
51                 read_address_matrixB   : out std_logic_vector (...
                      address_width - 1 downto 0);
52                 write_address_matrixC : out std_logic_vector (...
                      address_width - 1 downto 0);
53                 mux_enable             : out std_logic
54                  );
55
56 end component hresultcontrol2;
57
58 ----------------------------------------------------------------
59 component hresultreducecontrol2 is
60
61         generic (address_width: integer := 8;
62                  n1: integer := 5;
63                  n2: integer := 11);
64 -- address_width must be an even number
65 -- The hResultReduceControl module reduces the orignal 6x8
66 -- input matrix in half from the number of columns from the
67 -- hResultControl module. Example 6 x 12 will be reduce down
68 -- to 6x4 n1 is the row position and n2 is the column position.
69 -- you want n1 and n2 to be one size smaller then the 6 x 12
70 -- therefore n1 should equal = 5 since you start at zero
71 -- and n2 should equal = 11 since you start at zero
72 -- The hResultReduceControl module reduces the orignal 6x8
73 -- input matrix in half from the number of columns from the
74 -- hResultControl module. Example 6 x 12 will be reduce down
75 -- to 6x4 n1 is the row position and n2 is the column position.
76 -- you want n1 and n2 to be one size smaller then the 6 x 12
77 -- therefore n1 should equal = 5 since you start at zero
78 -- and n2 should equal = 11 since you start at zero
79
80    port (clk                         : in std_logic;
81          reset                       : in std_logic;
82          enable_control              : in std_logic;
83          writeEnable_matrix          : out std_logic;
84          enable_vresult              : out std_logic;
```

```
85          read_address_hResult       : out std_logic_vector (...
               address_width - 1 downto 0);
86          write_address_reducehResult: out std_logic_vector (...
               address_width - 1 downto 0);
87          mux_enable                 : out std_logic
88          );
89
90  end component hresultreducecontrol2;
91  ----------------------------------------------------------------
92  component vresultcontrol2 is
93
94     generic (address_width: integer := 8;
95              n1: integer := 9;
96                   n2: integer := 3);
97  -- address_width must be an even number
98
99     port (clk                   : in std_logic;
100          reset                  : in std_logic;
101          enable_control         : in std_logic;
102          update_reg_enable      : out std_logic;
103          writeEnable_matrixC    : out std_logic;
104          reset_reg              : out std_logiC;
105          enable_reduce          : out std_logic;
106          read_address_matrixA   : out std_logic_vector (...
               address_width - 1 downto 0);
107          read_address_matrixB   : out std_logic_vector (...
               address_width - 1 downto 0);
108          write_address_matrixC : out std_logic_vector (...
               address_width - 1 downto 0)
109          );
110
111 end component vresultcontrol2;
112
113 ----------------------------------------------------------------
114 component vresultreducecontrol2 is
115
116         generic (address_width: integer := 8;
117                  n1: integer := 9;
118                  n2: integer := 3);
119 -- address_width must be an even number
120    port (clk                         : in std_logic;
121          reset                       : in std_logic;
122          enable_control              : in std_logic;
123          writeEnable_matrix          : out std_logic;
124          enable_vresult              : out std_logic;
125          read_address_vResult        : out std_logic_vector (...
               address_width - 1 downto 0);
126          write_address_reducevResult: out std_logic_vector (...
               address_width - 1 downto 0)
127          );
128
129 end component vresultreducecontrol2;
```

```
130 ------------------------------------------------------------------
131 component mux2to1 is
132
133    generic (address_width: integer := 8);
134
135    port(sel         : in std_logic;
136         input1      : in std_logic_vector(address_width - 1 downto ...
                0);
137         input2      : in std_logic_vector(address_width - 1 downto ...
                0);
138         mux_output : out std_logic_vector(address_width - 1 downto...
                 0)
139         );
140
141 end component mux2to1;
142
143 ------------------------------------------------------------------
144 component mux1to2data_width is
145
146    generic (data_width: integer := 24);
147
148    port(sel          : in std_logic;
149         input        : in std_logic_vector(data_width - 1 downto 0)...
                ;
150         mux_output1 : out std_logic_vector(data_width - 1 downto ...
                0);
151         mux_output2 : out std_logic_vector(data_width - 1 downto ...
                0)
152         );
153
154 end component mux1to2data_width;
155
156 ------------------------------------------------------------------
157 component mux2to1data_width is
158
159    generic (data_width: integer := 24);
160
161    port(sel         : in std_logic;
162         input1      : in std_logic_vector(data_width - 1 downto 0);
163         input2      : in std_logic_vector(data_width - 1 downto 0);
164         mux_output : out std_logic_vector(data_width - 1 downto 0)
165         );
166
167 end component;
168 ------------------------------------------------------------------
169 component mux6to3enable is
170
171    port(sel         : in std_logic;
172         input1      : in std_logic;
173         input2      : in std_logic;
174         input3      : in std_logic;
175         input4      : in std_logic;
```

```
176         input5      : in std_logic;
177         input6      : in std_logic;
178         mux_output1 : out std_logic;
179         mux_output2 : out std_logic;
180         mux_output3 : out std_logic
181         );
182
183 end component;
184
185 -------------------------------------------------------------------
186 component mem_matrix is
187
188    generic (address_width: integer := 8;
189             data_width: integer := 24);
190
191    port (clk                  : in std_logic;
192          reset                : in std_logic;
193          writeEnable          : in std_logic;
194          matrix_values        : in std_logic_vector (data_width - ...
                1 downto 0);
195          write_address        : in std_logic_vector (address_width...
                - 1 downto 0);
196          read_address         : in std_logic_vector (address_width...
                - 1 downto 0);
197          output_matrix_values : out std_logic_vector(data_width - ...
                1 downto 0)
198            );
199
200 end component mem_matrix;
201
202 -------------------------------------------------------------------
203 component mask1x5mem_matrix is
204
205    generic (address_width: integer := 8;
206             data_width: integer := 24);
207
208    port(clk           : in std_logic;
209         reset         : in std_logic;
210           read_address  : in std_logic_vector (address_width - 1 ...
                downto 0);
211           output_matrix_values: out std_logic_vector(data_width - ...
                1 downto 0)
212           );
213
214 end component mask1x5mem_matrix;
215
216 -------------------------------------------------------------------
217 component mask5x1mem_matrix is
218
219    generic (address_width: integer := 8;
220             data_width: integer := 24);
221
```

```
222     port(clk            : in std_logic;
223          reset          : in std_logic;
224            read_address   : in std_logic_vector (address_width - 1 ...
                 downto 0);
225            output_matrix_values: out std_logic_vector(data_width - ...
                 1 downto 0)
226            );
227
228 end component mask5x1mem_matrix;
229
230 ------------------------------------------------------------------
231 component conv2 is
232
233     generic (data_width: integer := 24);
234
235     port (clk                    : in std_logic;
236           reset                  : in std_logic;
237           update_reg_enable      : in std_logic;
238           input_matrixA_values   : in std_logic_vector (data_width - ...
                 1 downto 0);
239           input_matrixB_values   : in std_logic_vector (data_width - ...
                 1 downto 0);
240           output_data_matrixC    : out std_logic_vector (data_width -...
                 1 downto 0)
241            );
242
243 end component conv2;
244
245 ------------------------------------------------------------------
246
247 signal update_reg_enable_to_mux_input1, ...
      control_writeEnable_memC_to_WriteEnableMatrixC_mux_input3: ...
      std_logic;
248 signal control_reset_reg_to_hvcontrol_reset_mux_input2, ...
      enable_reduce_to_hResultReduceControl: std_logic;
249 signal mux_enable_to_mux_select, writeEnable_matrix_to_MatrixA2, ...
      enable_vresult_to_vResultControl : std_logic;
250 signal mux_enable_to_mux_select_matrixA2, ...
      update_reg_enable_to_mux_input4: std_logic;
251 signal control_writeEnable_memC_to_mux_input6: std_logic;
252 signal control_reset_reg_to_mux_input5, ...
      enable_reduce_to_vResultReduceControl_enable : std_logic;
253 signal writeEnable_matrix_to_MatrixReduce, ...
      mux_output_to_conv2_update_reg: std_logic;
254 signal mux_output_to_conv2_reset, ...
      mux_out_writeEnableMatrixC_to_writeEnableMatrixC: std_logic;
255 signal read_address_matrixA_to_mux_input1, ...
      read_address_matrixB_to_mask1x5 : std_logic_vector (...
      address_width - 1 downto 0);
256 signal write_address_matrixC_to_WriteAddressMatrix_mux_input1, ...
      write_address_reducehResult_to_mux_input1_matrixA2_address : ...
      std_logic_vector (address_width - 1 downto 0);
```

```
257 signal read_address_hResult_to_mux_input1_readAddressMatrixC, ...
       write_address_matrixC_to_WriteAddressMatrix_mux_input2 : ...
       std_logic_vector (address_width - 1 downto 0);
258 signal write_address_reducevResult_to_mux_input2_matrixA2_address,...
        read_address_matrixB_to_mask5x1 : std_logic_vector (...
       address_width - 1 downto 0);
259 signal read_address_vResult_to_mux_input2_readAddressMatrixC, ...
       write_address_reducevResult_to_MatrixReduce: std_logic_vector (...
       address_width - 1 downto 0);
260 signal mux_ouput_to_matrixA1,mux_output_to_matrixA2_address: ...
       std_logic_vector (address_width - 1 downto 0);
261 signal read_address_matrixC_to_address_matrixC, ...
       mux_output_WriteAddressMatrixC_to_write_address_matrixC: ...
       std_logic_vector (address_width - 1 downto 0);
262 signal output_matrix_values_to_mux_input1_matrixA_values, ...
       output_matrix_values_to_mux_input2_matrixA_values: ...
       std_logic_vector(data_width - 1 downto 0);
263 signal mux_output_matrixA_values_to_conv2_input1, ...
       output_matrix_values_to_mux_input1_matrixB_values: ...
       std_logic_vector(data_width - 1 downto 0);
264 signal output_matrix_values_to_mux_input2_matrixB_values, ...
       mux_output_matrixB_to_conv2_input2: std_logic_vector(data_width...
        - 1 downto 0);
265 signal output_matrixC_values_to_mux_input_values, ...
       mux_output_MatrixC_values_to_matrixA2_values: std_logic_vector(...
       data_width - 1 downto 0);
266 signal mux_output_MatrixC_values_to_matrixReduce_values: ...
       std_logic_vector(data_width - 1 downto 0);
267 signal output_data_matrixC_to_matrixC_values: std_logic_vector(...
       data_width - 1 downto 0);
268
269 begin
270
271 hresultcontrol1: hresultcontrol2 port map (clk => clk,
272 reset => reset,
273 enable_control => enable_conv2_module,
274 update_reg_enable => update_reg_enable_to_mux_input1,
275 writeEnable_matrixC =>
276 control_writeEnable_memC_to_WriteEnableMatrixC_mux_input3,
277 reset_reg => control_reset_reg_to_hvcontrol_reset_mux_input2,
278 enable_reduce => enable_reduce_to_hResultReduceControl,
279 read_address_matrixA => read_address_matrixA_to_mux_input1,
280 read_address_matrixB => read_address_matrixB_to_mask1x5,
281 write_address_matrixC =>
282 write_address_matrixC_to_WriteAddressMatrix_mux_input1,
283 mux_enable => mux_enable_to_mux_select);
284
285 ------------------------------------------------------------------
286 hresultreducecontrol1: hresultreducecontrol2 port map(clk => clk,
287 reset => reset,
288 enable_control => enable_reduce_to_hResultReduceControl,
289 writeEnable_matrix => writeEnable_matrix_to_MatrixA2,
```

```
290 enable_vresult => enable_vresult_to_vResultControl,
291 read_address_hResult =>
292 read_address_hResult_to_mux_input1_readAddressMatrixC,
293 write_address_reducehResult =>
294 write_address_reducehResult_to_mux_input1_matrixA2_address,
295 mux_enable => mux_enable_to_mux_select_matrixA2);
296
297 ------------------------------------------------------------------
298 vresultcontrol1: vresultcontrol2 port map (clk => clk,
299 reset => reset,
300 enable_control => enable_vresult_to_vResultControl,
301 update_reg_enable => update_reg_enable_to_mux_input4,
302 writeEnable_matrixC => control_writeEnable_memC_to_mux_input6,
303 reset_reg => control_reset_reg_to_mux_input5,
304 enable_reduce => enable_reduce_to_vResultReduceControl_enable,
305 read_address_matrixA =>
306 write_address_reducevResult_to_mux_input2_matrixA2_address,
307 read_address_matrixB => read_address_matrixB_to_mask5x1,
308 write_address_matrixC =>
309 write_address_matrixC_to_WriteAddressMatrix_mux_input2);
310
311 ------------------------------------------------------------------
312 vresultreducecontrol1 :vresultreducecontrol2 port map(clk => clk,
313 reset => reset,
314 enable_control => enable_reduce_to_vResultReduceControl_enable,
315 writeEnable_matrix => writeEnable_matrix_to_MatrixReduce,
316 enable_vresult => reduce_module_finished,
317 read_address_vResult =>
318 read_address_vResult_to_mux_input2_readAddressMatrixC,
319 write_address_reducevResult =>
320 write_address_reducevResult_to_MatrixReduce);
321
322 ------------------------------------------------------------------
323 mux2to1_matrixA1: mux2to1 port map(sel => enable_matrixA,
324 input1 => read_address_matrixA_to_mux_input1,
325 input2 => read_address_matrixA,
326 mux_output => mux_ouput_to_matrixA1);
327
328 ------------------------------------------------------------------
329 mux2to1_matrixA_values: mux2to1data_width port map(
330 sel => mux_enable_to_mux_select,
331 input1 => output_matrix_values_to_mux_input1_matrixA_values,
332 input2 => output_matrix_values_to_mux_input2_matrixA_values,
333 mux_output => mux_output_matrixA_values_to_conv2_input1);
334
335 ------------------------------------------------------------------
336 mux2to1_matrixB_values: mux2to1data_width port map(
337 sel => mux_enable_to_mux_select,
338 input1 => output_matrix_values_to_mux_input1_matrixB_values,
339 input2 => output_matrix_values_to_mux_input2_matrixB_values,
340 mux_output => mux_output_matrixB_to_conv2_input2);
341
```

```
342 ----------------------------------------------------------------
343 mux6to3_Enable1: mux6to3enable port map (
344 sel => mux_enable_to_mux_select,
345 input1 => update_reg_enable_to_mux_input1,
346 input2 => control_reset_reg_to_hvcontrol_reset_mux_input2,
347 input3 =>
348 control_writeEnable_memC_to_WriteEnableMatrixC_mux_input3,
349 input4 => update_reg_enable_to_mux_input4,
350 input5 => control_reset_reg_to_mux_input5,
351 input6 => control_writeEnable_memC_to_mux_input6,
352 mux_output1 => mux_output_to_conv2_update_reg,
353 mux_output2 => mux_output_to_conv2_reset,
354 mux_output3 => mux_out_writeEnableMatrixC_to_writeEnableMatrixC);
355
356 ----------------------------------------------------------------
357 mux2to1_ReadAddressMatrixC: mux2to1 port map(
358 sel => mux_enable_to_mux_select_matrixA2,
359 input1 => read_address_hResult_to_mux_input1_readAddressMatrixC,
360 input2 => read_address_vResult_to_mux_input2_readAddressMatrixC,
361 mux_output => read_address_matrixC_to_address_matrixC);
362
363 ----------------------------------------------------------------
364 mux2to1_WriteAddressMatrixC: mux2to1 port map(
365 sel => mux_enable_to_mux_select,
366 input1 => write_address_matrixC_to_WriteAddressMatrix_mux_input1,
367 input2 => write_address_matrixC_to_WriteAddressMatrix_mux_input2,
368 mux_output =>
369 mux_output_WriteAddressMatrixC_to_write_address_matrixC);
370
371 ----------------------------------------------------------------
372 mux1to2_matrixA2_ReduceMatrix_values: mux1to2data_width port map(
373 sel => mux_enable_to_mux_select_matrixA2,
374 input => output_matrixC_values_to_mux_input_values,
375 mux_output1 => mux_output_MatrixC_values_to_matrixA2_values,
376 mux_output2 => mux_output_MatrixC_values_to_matrixReduce_values);
377
378 ----------------------------------------------------------------
379 mem_matrixA1: mem_matrix port map (clk => clk, reset => reset,
380 writeEnable => enable_matrixA,
381 matrix_values => input_matrixA_values,
382 write_address => mux_ouput_to_matrixA1,
383 read_address => mux_ouput_to_matrixA1,
384 output_matrix_values =>
385 output_matrix_values_to_mux_input1_matrixA_values);
386
387 ----------------------------------------------------------------
388 mem_matrixA2: mem_matrix port map (clk => clk, reset => reset,
389 writeEnable => writeEnable_matrix_to_MatrixA2,
390 matrix_values => mux_output_MatrixC_values_to_matrixA2_values,
391 write_address =>
392 write_address_reducehResult_to_mux_input1_matrixA2_address,
393 read_address =>
```

```
394 write_address_reducevResult_to_mux_input2_matrixA2_address,
395 output_matrix_values =>
396 output_matrix_values_to_mux_input2_matrixA_values);
397
398 ----------------------------------------------------------------
399 mask1x5mem_matrix0: mask1x5mem_matrix port map (clk => clk,
400 reset => reset,
401 read_address => read_address_matrixB_to_mask1x5,
402 output_matrix_values =>
403 output_matrix_values_to_mux_input1_matrixB_values);
404
405 ----------------------------------------------------------------
406 mask5x1mem_matrix0: mask5x1mem_matrix port map (clk => clk,
407 reset => reset,
408 read_address => read_address_matrixB_to_mask5x1,
409 output_matrix_values =>
410 output_matrix_values_to_mux_input2_matrixB_values);
411
412 ----------------------------------------------------------------
413 conv2_1: conv2 port map (clk => clk,
414 reset => mux_output_to_conv2_reset,
415 update_reg_enable => mux_output_to_conv2_update_reg,
416 input_matrixA_values => mux_output_matrixA_values_to_conv2_input1,
417 input_matrixB_values  => mux_output_matrixB_to_conv2_input2,
418 output_data_matrixC => output_data_matrixC_to_matrixC_values);
419
420 ----------------------------------------------------------------
421 mem_matrixC: mem_matrix port map (clk => clk,
422 reset => reset,
423 writeEnable  => mux_out_writeEnableMatrixC_to_writeEnableMatrixC,
424 matrix_values => output_data_matrixC_to_matrixC_values,
425 write_address =>
426 mux_output_WriteAddressMatrixC_to_write_address_matrixC,
427 read_address => read_address_matrixC_to_address_matrixC,
428 output_matrix_values =>
429 output_matrixC_values_to_mux_input_values);
430
431 ----------------------------------------------------------------
432 mem_matrixReduce: mem_matrix port map (clk => clk,
433 reset => reset,
434 writeEnable  => writeEnable_matrix_to_MatrixReduce,
435 matrix_values => mux_output_MatrixC_values_to_matrixReduce_values,
436 write_address => write_address_reducevResult_to_MatrixReduce,
437 read_address => read_address_matrixReduce,
438 output_matrix_values => output_data_matrixReduce);
439
440 end structure_reduceconv2_module2;
```

C.3 Reduce Matrix Function Top-Level 1 Booth Multiplier

Listing C.3: The ReduceConv2ModuleBooth.vhd VHDL file. (appendix3/ReduceConv2ModuleBooth.vhd)

```
1  -- Capt. Jason Shirley
2  -- This is the Reduce Matrix Function Top Level Design 1 using the
3  -- Booth Multiplier.  These Modules are structually connected.
4
5
6  library ieee;
7  use ieee.std_logic_1164.all;
8  use ieee.std_logic_unsigned.all;
9  use ieee.std_logic_arith.all;
10 use ieee.numeric_std.all;
11
12 ------------------------------------------------------------------
13
14 entity reduceconv2_modulebooth is
15
16    generic (address_width: integer := 8;
17             data_width: integer := 24);
18
19    port (clk                          : in std_logic;
20          reset                        : in std_logic;
21          enable_matrixA               : in std_logic;
22          enable_conv2_module          : in std_logic;
23          read_address_matrixA         : in std_logic_vector (...
               address_width - 1 downto 0);
24          read_address_matrixReduce    : in std_logic_vector (...
               address_width - 1 downto 0);
25          input_matrixA_values         : in std_logic_vector (...
               data_width - 1 downto 0);
26          output_data_matrixC          : out std_logic_vector(...
               data_width - 1 downto 0);
27          reduce_module_finished       : out std_logic;
28          input_to_matrixC_hresult     : out std_logic_vector(...
               data_width - 1 downto 0);
29          writeEnable_matrixC_hresult : out std_logic;
30          address_matrixA_hresult      : out std_logic_vector(...
               address_width - 1 downto 0);
31          address_matrixB_hresult      : out std_logic_vector(...
               address_width - 1 downto 0);
32          address_matrixC_hresult      : out std_logic_vector(...
               address_width - 1 downto 0);
33          matrixA_output_hresult       : out std_logic_vector(...
               data_width - 1 downto 0);
34          matrixB_output_hresult       : out std_logic_vector(...
               data_width - 1 downto 0);
35          control_adder_signal_hresult: out std_logic;
36               input_to_matrixC_vresult    : out std_logic_vector(...
                    data_width - 1 downto 0);
37          writeEnable_matrixC_vresult : out std_logic;
```

```
38          address_matrixA_vresult     : out std_logic_vector(...
               address_width - 1 downto 0);
39          address_matrixB_vresult     : out std_logic_vector(...
               address_width - 1 downto 0);
40          address_matrixC_vresult     : out std_logic_vector(...
               address_width - 1 downto 0);
41          matrixA_output_vresult      : out std_logic_vector(...
               data_width - 1 downto 0);
42          matrixB_output_vresult      : out std_logic_vector(...
               data_width - 1 downto 0);
43          control_adder_signal_vresult: out std_logic;
44          mult_ready_hresult          : out std_logic;
45          mult_done_hresult           : out std_logic;
46          mult_ready_vresult          : out std_logic;
47          mult_done_vresult           : out std_logic
48               );
49
50 end reduceconv2_modulebooth;
51
52 ----------------------------------------------------------------
53
54 architecture structure_reduceconv2_modulebooth of ...
      reduceconv2_modulebooth is
55
56 component hresultboothconv2_module is
57
58    generic (address_width: integer := 8;
59             data_width: integer := 24);
60
61    port (clk                   : in std_logic;
62          reset                 : in std_logic;
63          enable_matrixA        : in std_logic;
64          enable_conv2_module   : in std_logic;
65          read_address_matrixA  : in std_logic_vector (...
               address_width - 1 downto 0);
66          read_address_matrixC  : in std_logic_vector (...
               address_width - 1 downto 0);
67          input_matrixA_values  : in std_logic_vector (data_width -...
                1 downto 0);
68          output_data_matrixC   : out std_logic_vector(data_width -...
                1 downto 0);
69          enable_reduce         : out std_logic;
70          input_to_matrixC      : out std_logic_vector(data_width -...
                1 downto 0);
71          writeEnable_matrixC   : out std_logic;
72          address_matrixA       : out std_logic_vector(...
               address_width - 1 downto 0);
73          address_matrixB       : out std_logic_vector(...
               address_width - 1 downto 0);
74          address_matrixC       : out std_logic_vector(...
               address_width - 1 downto 0);
```

```
75          matrixA_output         : out std_logic_vector(data_width -...
                1 downto 0);
76          matrixB_output         : out std_logic_vector(data_width -...
                1 downto 0);
77          control_adder_signal   : out std_logic;
78          mult_ready             : out std_logic;
79          mult_done              : out std_logic
80               );
81
82  end component hresultboothconv2_module;
83
84  ----------------------------------------------------------------
85
86  component hresultreducecontrolbooth is
87
88          generic (address_width: integer := 8;
89                   n1: integer := 5;
90                   n2: integer := 11);
91  -- address_width must be an even number
92  -- The hResultReduceControl module reduces the orignal 6x8
93  -- input matrix in half from the number of columns from the
94  -- hResultControl module. Example 6 x 12 will be reduce down
95  -- to 6x4 n1 is the row position and n2 is the column position.
96  -- you want n1 and n2 to be one size smaller then the 6 x 12
97  -- therefore n1 should equal = 5 since you start at zero
98  -- and n2 should equal = 11 since you start at zero
99  -- The hResultReduceControl module reduces the orignal 6x8
100 -- input matrix in half from the number of columns from the
101 -- hResultControl module. Example 6 x 12 will be reduce down
102 -- to 6x4 n1 is the row position and n2 is the column position.
103 -- you want n1 and n2 to be one size smaller then the 6 x 12
104 -- therefore n1 should equal = 5 since you start at zero
105 -- and n2 should equal = 11 since you start at zero
106
107    port (clk                          : in std_logic;
108          reset                        : in std_logic;
109          enable_control               : in std_logic;
110          writeEnable_matrix           : out std_logic;
111          enable_vresult               : out std_logic;
112          read_address_hResult         : out std_logic_vector (...
                address_width - 1 downto 0);
113          write_address_reducehResult: out std_logic_vector (...
                address_width - 1 downto 0)
114          );
115
116 end component hresultreducecontrolbooth;
117 ----------------------------------------------------------------
118
119 component vresultboothconv2_module is
120
121    generic (address_width: integer := 8;
122             data_width: integer := 24);
```

```
123
124     port (clk                    : in std_logic;
125           reset                  : in std_logic;
126           enable_matrixA         : in std_logic;
127           enable_conv2_module    : in std_logic;
128           read_address_matrixA   : in std_logic_vector (...
                 address_width - 1 downto 0);
129           read_address_matrixC   : in std_logic_vector (...
                 address_width - 1 downto 0);
130           input_matrixA_values   : in std_logic_vector (data_width -...
                  1 downto 0);
131           output_data_matrixC    : out std_logic_vector(data_width -...
                  1 downto 0);
132           enable_reduce          : out std_logic;
133           input_to_matrixC       : out std_logic_vector(data_width -...
                  1 downto 0);
134           writeEnable_matrixC    : out std_logic;
135           address_matrixA        : out std_logic_vector(...
                 address_width - 1 downto 0);
136           address_matrixB        : out std_logic_vector(...
                 address_width - 1 downto 0);
137           address_matrixC        : out std_logic_vector(...
                 address_width - 1 downto 0);
138           matrixA_output         : out std_logic_vector(data_width -...
                  1 downto 0);
139           matrixB_output         : out std_logic_vector(data_width -...
                  1 downto 0);
140           control_adder_signal   : out std_logic;
141           mult_ready             : out std_logic;
142           mult_done              : out std_logic
143                );
144
145 end component vresultboothconv2_module;
146
147 ------------------------------------------------------------------
148 component vresultreducecontrolbooth is
149
150         generic (address_width: integer := 8;
151                  n1: integer := 9;
152                  n2: integer := 3);
153 -- address_width must be an even number
154     port (clk                        : in std_logic;
155           reset                      : in std_logic;
156           enable_control             : in std_logic;
157           writeEnable_matrix         : out std_logic;
158           enable_vresult             : out std_logic;
159           read_address_vResult       : out std_logic_vector (...
                 address_width - 1 downto 0);
160           write_address_reducevResult: out std_logic_vector (...
                 address_width - 1 downto 0)
161           );
162
```

```
163 end component vresultreducecontrolbooth;
164
165 ----------------------------------------------------------------
166 component mem_matrix is
167
168         generic (address_width: integer := 8;
169                  data_width: integer := 24);
170
171    port (clk                  : in std_logic;
172          reset                : in std_logic;
173          writeEnable          : in std_logic;
174            matrix_values        : in std_logic_vector (data_width ...
                 - 1 downto 0);
175            write_address        : in std_logic_vector (...
                 address_width - 1 downto 0);
176            read_address         : in std_logic_vector (...
                 address_width - 1 downto 0);
177            output_matrix_values : out std_logic_vector(data_width ...
                 - 1 downto 0)
178            );
179
180 end component mem_matrix;
181
182 ----------------------------------------------------------------
183
184 signal enable_reduce_h_to_enable_control, ...
    writeEnable_matrix_to_enable_matrixA: std_logic;
185 signal enable_vresult_to_enable_conv2_module, ...
    enable_reduce_v_to_enable_control: std_logic;
186 signal writeEnable_matrix_to_writeEnable: std_logic;
187 signal write_address_reducevResult_to_write_address:...
    std_logic_vector (address_width - 1 downto 0);
188 signal read_address_hResult_to_read_address_matrixC:...
    std_logic_vector (address_width - 1 downto 0);
189 signal read_addressvResult_to_read_address_matrixC:...
    std_logic_vector (address_width - 1 downto 0);
190 signal write_address_reducehResult_to_read_address_matrixA:...
    std_logic_vector (address_width - 1 downto 0);
191 signal output_data_matrixC_to_input_matrixA_values: ...
    std_logic_vector(data_width - 1 downto 0);
192 signal output_data_matrixC_to_matrix_values: std_logic_vector(...
    data_width - 1 downto 0);
193
194 begin
195 hresultboothconv2_module1: hresultboothconv2_module port map (
196 clk => clk,
197 reset => reset,
198 enable_matrixA => enable_matrixA,
199 enable_conv2_module => enable_conv2_module,
200 read_address_matrixA => read_address_matrixA,
201 read_address_matrixC =>
202 read_address_hResult_to_read_address_matrixC,
```

```
203 input_matrixA_values => input_matrixA_values,
204 output_data_matrixC =>
205 output_data_matrixC_to_input_matrixA_values,
206 enable_reduce  => enable_reduce_h_to_enable_control,
207 input_to_matrixC => input_to_matrixC_hresult,
208 writeEnable_matrixC => writeEnable_matrixC_hresult,
209 address_matrixA => address_matrixA_hresult,
210 address_matrixB => address_matrixB_hresult,
211 address_matrixC => address_matrixC_hresult,
212 matrixA_output => matrixA_output_hresult,
213 matrixB_output => matrixB_output_hresult,
214 control_adder_signal => control_adder_signal_hresult,
215 mult_ready => mult_ready_hresult,
216 mult_done => mult_done_hresult);
217
218 hresultreducecontrolbooth1: hresultreducecontrolbooth port map(
219 clk => clk,
220 reset => reset,
221 enable_control => enable_reduce_h_to_enable_control,
222 writeEnable_matrix=> writeEnable_matrix_to_enable_matrixA,
223 enable_vresult => enable_vresult_to_enable_conv2_module,
224 read_address_hResult =>
225 read_address_hResult_to_read_address_matrixC,
226 write_address_reducehResult =>
227 write_address_reducehResult_to_read_address_matrixA);
228
229 vresultboothconv2_module1: vresultboothconv2_module port map(
230 clk => clk,
231 reset => reset,
232 enable_matrixA => writeEnable_matrix_to_enable_matrixA,
233 enable_conv2_module => enable_vresult_to_enable_conv2_module,
234 read_address_matrixA =>
235 write_address_reducehResult_to_read_address_matrixA,
236 read_address_matrixC =>
237 read_addressvResult_to_read_address_matrixC,
238 input_matrixA_values =>
239 output_data_matrixC_to_input_matrixA_values,
240 output_data_matrixC => output_data_matrixC_to_matrix_values,
241 enable_reduce => enable_reduce_v_to_enable_control,
242 input_to_matrixC => input_to_matrixC_vresult,
243 writeEnable_matrixC => writeEnable_matrixC_vresult,
244 address_matrixA => address_matrixA_vresult,
245 address_matrixB => address_matrixB_vresult,
246 address_matrixC => address_matrixC_vresult,
247 matrixA_output => matrixA_output_vresult,
248 matrixB_output => matrixB_output_vresult,
249 control_adder_signal => control_adder_signal_vresult,
250 mult_ready => mult_ready_vresult,
251 mult_done => mult_done_vresult);
252
253 vresultreducecontrolbooth1 :vresultreducecontrolbooth port map(
254 clk => clk,
```

```
255 reset => reset,
256 enable_control => enable_reduce_v_to_enable_control,
257 writeEnable_matrix => writeEnable_matrix_to_writeEnable,
258 enable_vresult => reduce_module_finished,
259 read_address_vResult =>
260 read_addressvResult_to_read_address_matrixC,
261 write_address_reducevResult =>
262 write_address_reducevResult_to_write_address);
263
264 mem_matrixReduce: mem_matrix port map (clk => clk,
265 reset => reset,
266 writeEnable  => writeEnable_matrix_to_writeEnable,
267 matrix_values => output_data_matrixC_to_matrix_values,
268 write_address => write_address_reducevResult_to_write_address,
269 read_address => read_address_matrixReduce,
270 output_matrix_values => output_data_matrixC);
271
272 end structure_reduceconv2_modulebooth;
```

C.4 Reduce Matrix Function Top-Level 2 Booth Multiplier

Listing C.4: The ReduceConv2ModuleBooth2.vhd VHDL file. (appendix3/ReduceConv2ModuleBooth2.vhd)

```
1  -- Capt. Jason Shirley
2  -- This is the Reduce Matrix Function Top-Level Design 2 using the
3  -- Booth Multiplier.  These Modules are structually connected.
4
5
6  library ieee;
7  use ieee.std_logic_1164.all;
8  use ieee.std_logic_unsigned.all;
9  use ieee.std_logic_arith.all;
10 use ieee.numeric_std.all;
11
12 ----------------------------------------------------------------------
13
14 entity reduceconv2_modulebooth2 is
15
16    generic (address_width: integer := 8;
17             data_width: integer := 24);
18
19    port (clk                      : in std_logic;
20          reset                    : in std_logic;
21          enable_matrixA           : in std_logic;
22          enable_conv2_module      : in std_logic;
23          read_address_matrixA     : in std_logic_vector (...
               address_width - 1 downto 0);
24          read_address_matrixReduce: in std_logic_vector (...
               address_width - 1 downto 0);
```

```
25          input_matrixA_values     : in std_logic_vector (...
               data_width - 1 downto 0);
26          reduce_module_finished   : out std_logic;
27          output_data_matrixReduce : out std_logic_vector(...
               data_width - 1 downto 0);
28          mult_ready_hresult       : out std_logic;
29          mult_done_hresult        : out std_logic;
30          mult_ready_vresult       : out std_logic;
31          mult_done_vresult        : out std_logic
32               );
33
34  end reduceconv2_modulebooth2;
35
36  ----------------------------------------------------------------------
37  architecture structure_reduceconv2_modulebooth2 of ...
       reduceconv2_modulebooth2 is
38
39  component hresultcontrolbooth2 is
40
41     generic (address_width: integer := 8;
42                   n1: integer := 5;
43                  n2: integer := 11);
44  -- address_width must be an even number
45  -- n1 and n2 are the size of the input matrix conv2 with the
46  -- 1x5 mask example, input 6x8 matrix, mask 1x5, new conv2 matrix
47  -- equals Conv2 (n1xn2)= (6+1-1)x (8+5-1), therefore n1 = 6,
48  -- n2 = 12, but the matrix starts at zero so, n1 = (6-1) = 5,
49  -- n2 = (12-1) = 11.
50
51     port (clk                    : in std_logic;
52           reset                  : in std_logic;
53           enable_control         : in std_logic;
54           update_reg_enable      : out std_logic;
55           writeEnable_matrixC    : out std_logic;
56           reset_reg              : out std_logic;
57           enable_reduce          : out std_logic;
58           execute                : out std_logic;
59           read_address_matrixA   : out std_logic_vector (...
                address_width - 1 downto 0);
60           read_address_matrixB   : out std_logic_vector (...
                address_width - 1 downto 0);
61           write_address_matrixC : out std_logic_vector (...
                address_width - 1 downto 0);
62           mux_enable             : out std_logic
63           );
64
65  end component hresultcontrolbooth2;
66
67  ----------------------------------------------------------------------
68  component hresultreducecontrolbooth2 is
69
```

```
70          generic (address_width: integer := 8; -- address_width ...
               must be an even number
71                   n1: integer := 5;
72                   n2: integer := 11);
73  -- address_width must be an even number
74  -- The hResultReduceControl module reduces the orignal 6x8
75  -- input matrix in half from the number of columns from the
76  -- hResultControl module. Example 6 x 12 will be reduce down
77  -- to 6x4 n1 is the row position and n2 is the column position.
78  -- you want n1 and n2 to be one size smaller then the 6 x 12
79  -- therefore n1 should equal = 5 since you start at zero
80  -- and n2 should equal = 11 since you start at zero
81  -- The hResultReduceControl module reduces the orignal 6x8
82  -- input matrix in half from the number of columns from the
83  -- hResultControl module. Example 6 x 12 will be reduce down
84  -- to 6x4 n1 is the row position and n2 is the column position.
85  -- you want n1 and n2 to be one size smaller then the 6 x 12
86  -- therefore n1 should equal = 5 since you start at zero
87  -- and n2 should equal = 11 since you start at zero
88
89     port (clk                          : in std_logic;
90           reset                        : in std_logic;
91           enable_control               : in std_logic;
92           writeEnable_matrix           : out std_logic;
93           enable_vresult               : out std_logic;
94           read_address_hResult         : out std_logic_vector (...
                address_width - 1 downto 0);
95           write_address_reducehResult: out std_logic_vector (...
                address_width - 1 downto 0);
96           mux_enable                   : out std_logic
97           );
98
99  end component hresultreducecontrolbooth2;
100 ------------------------------------------------------------------
101
102 component vresultcontrolbooth2 is
103
104    generic (address_width: integer := 8;
105                   n1: integer := 9;
106                   n2: integer := 3);
107 -- address_width must be an even number
108 -- n1 and n2 are the size of the input matrix conv2 with the
109 -- 5x1 mask example, input 6x4 matrix, mask 5x1, new conv2
110 -- matrix equals Conv2 (n1xn2)= (6+5-1)x (4+1-1),
111 -- therefore n1 = 10, n2 = 4, but the matrix starts at
112 -- zero so, n1 = (10-1) = 9, n2 = (4-1) = 3.
113
114    port (clk                     : in std_logic;
115          reset                   : in std_logic;
116          enable_control          : in std_logic;
117          update_reg_enable       : out std_logic;
118          writeEnable_matrixC     : out std_logic;
```

```
119          reset_reg              : out std_logic;
120          enable_reduce          : out std_logic;
121          execute                : out std_logic;
122          read_address_matrixA   : out std_logic_vector (...
                address_width - 1 downto 0);
123          read_address_matrixB   : out std_logic_vector (...
                address_width - 1 downto 0);
124          write_address_matrixC  : out std_logic_vector (...
                address_width - 1 downto 0)
125          );
126
127 end component vresultcontrolbooth2;
128
129 ----------------------------------------------------------------------
130
131 component vresultreducecontrolbooth2 is
132
133         generic (address_width: integer := 8;
134                  n1: integer := 9;
135                  n2: integer := 3);
136 -- address_width must be an even number
137    port (clk                         : in std_logic;
138          reset                       : in std_logic;
139          enable_control              : in std_logic;
140          writeEnable_matrix          : out std_logic;
141          enable_vresult              : out std_logic;
142          read_address_vResult        : out std_logic_vector (...
                address_width - 1 downto 0);
143          write_address_reducevResult: out std_logic_vector (...
                address_width - 1 downto 0)
144          );
145
146 end component vresultreducecontrolbooth2;
147 ----------------------------------------------------------------------
148 component mux2to1 is
149
150    generic (address_width: integer := 8);
151
152    port(sel         : in std_logic;
153         input1      : in std_logic_vector(address_width - 1 downto ...
                0);
154         input2      : in std_logic_vector(address_width - 1 downto ...
                0);
155         mux_output : out std_logic_vector(address_width - 1 downto...
                0)
156         );
157
158 end component mux2to1;
159
160 ----------------------------------------------------------------------
161 component mux1to2data_width is
162
```

```
163     generic (data_width: integer := 24);
164
165     port(sel          : in std_logic;
166          input        : in std_logic_vector(data_width - 1 downto 0)...
                ;
167          mux_output1 : out std_logic_vector(data_width - 1 downto ...
                0);
168          mux_output2 : out std_logic_vector(data_width - 1 downto ...
                0)
169          );
170
171 end component mux1to2data_width;
172
173 ----------------------------------------------------------------------
174 component mux2to1data_width is
175
176     generic (data_width: integer := 24);
177
178     port(sel          : in std_logic;
179          input1       : in std_logic_vector(data_width - 1 downto 0);
180          input2       : in std_logic_vector(data_width - 1 downto 0);
181          mux_output : out std_logic_vector(data_width - 1 downto 0)
182          );
183
184 end component;
185 ----------------------------------------------------------------------
186 component mux8to4enable is
187
188     port(sel          : in std_logic;
189          input1       : in std_logic;
190          input2       : in std_logic;
191          input3       : in std_logic;
192          input4       : in std_logic;
193          input5       : in std_logic;
194          input6       : in std_logic;
195          input7       : in std_logic;
196          input8       : in std_logic;
197          mux_output1 : out std_logic;
198          mux_output2 : out std_logic;
199          mux_output3 : out std_logic;
200          mux_output4 : out std_logic
201          );
202
203 end component;
204 ----------------------------------------------------------------------
205 component mux2to4enable is
206
207      port(sel          : in std_logic;
208           input1       : in std_logic;
209           input2       : in std_logic;
210           mux_output1 : out std_logic;
211           mux_output2 : out std_logic;
```

```
212          mux_output3 : out std_logic;
213          mux_output4 : out std_logic
214          );
215
216 end component;
217
218 ----------------------------------------------------------------------
219 component mem_matrix is
220
221    generic (address_width: integer := 8;
222             data_width: integer := 24);
223
224    port (clk                  : in std_logic;
225          reset                : in std_logic;
226          writeEnable          : in std_logic;
227          matrix_values        : in std_logic_vector (data_width - ...
                1 downto 0);
228          write_address        : in std_logic_vector (address_width...
                - 1 downto 0);
229          read_address         : in std_logic_vector (address_width...
                - 1 downto 0);
230          output_matrix_values : out std_logic_vector(data_width - ...
                1 downto 0)
231            );
232
233 end component mem_matrix;
234
235 ----------------------------------------------------------------------
236 component mask1x5mem_matrix is
237
238    generic (address_width: integer := 8;
239             data_width: integer := 24);
240
241    port(clk           : in std_logic;
242         reset         : in std_logic;
243           read_address  : in std_logic_vector (address_width - 1 ...
                downto 0);
244           output_matrix_values: out std_logic_vector(data_width - ...
                1 downto 0));
245
246 end component mask1x5mem_matrix;
247
248 ----------------------------------------------------------------------
249 component mask5x1mem_matrix is
250
251    generic (address_width: integer := 8;
252             data_width: integer := 24);
253
254    port(clk           : in std_logic;
255         reset         : in std_logic;
256           read_address  : in std_logic_vector (address_width - 1 ...
                downto 0);
```

```
257           output_matrix_values: out std_logic_vector(data_width - ...
                 1 downto 0)
258           );
259
260 end component mask5x1mem_matrix;
261
262 ----------------------------------------------------------------------
263 component boothconv2 is
264
265    generic (data_width: integer := 24);
266
267    port (clk                    : in std_logic;
268          reset                  : in std_logic;
269          execute                : in std_logic;
270            update_reg_enable      : in std_logic;
271            input_matrixA_values   : in std_logic_vector (data_width...
                   - 1 downto 0);
272            input_matrixB_values   : in std_logic_vector (data_width...
                   - 1 downto 0);
273            output_data_matrixC    : out std_logic_vector (...
                  data_width - 1 downto 0);
274            mult_ready             : out std_logic;
275            mult_done              : out std_logic
276            );
277
278 end component boothconv2;
279
280 ----------------------------------------------------------------------
281
282 signal update_reg_enable_to_mux_input1, ...
      control_writeEnable_memC_to_WriteEnableMatrixC_mux_input3: ...
      std_logic;
283 signal control_reset_reg_to_hvcontrol_reset_mux_input2, ...
      enable_reduce_to_hResultReduceControl: std_logic;
284 signal mux_enable_to_mux_select, writeEnable_matrix_to_MatrixA2, ...
      enable_vresult_to_vResultControl : std_logic;
285 signal mux_enable_to_mux_select_matrixA2, ...
      update_reg_enable_to_mux_input5: std_logic;
286 signal control_writeEnable_memC_to_mux_input7: std_logic;
287 signal control_reset_reg_to_mux_input6, ...
      enable_reduce_to_vResultReduceControl_enable : std_logic;
288 signal writeEnable_matrix_to_MatrixReduce, ...
      mux_output_to_conv2_update_reg: std_logic;
289 signal mux_output_to_conv2_reset, ...
      mux_out_writeEnableMatrixC_to_writeEnableMatrixC: std_logic;
290 signal read_address_matrixA_to_mux_input1, ...
      read_address_matrixB_to_mask1x5 : std_logic_vector (...
      address_width - 1 downto 0);
291 signal write_address_matrixC_to_WriteAddressMatrix_mux_input1, ...
      write_address_reducehResult_to_mux_input1_matrixA2_address : ...
      std_logic_vector (address_width - 1 downto 0);
```

```
signal read_address_hResult_to_mux_input1_readAddressMatrixC, ...
   write_address_matrixC_to_WriteAddressMatrix_mux_input2 : ...
   std_logic_vector (address_width - 1 downto 0);
signal write_address_reducevResult_to_mux_input2_matrixA2_address,...
    read_address_matrixB_to_mask5x1 : std_logic_vector (...
   address_width - 1 downto 0);
signal read_address_vResult_to_mux_input2_readAddressMatrixC, ...
   write_address_reducevResult_to_MatrixReduce: std_logic_vector (...
   address_width - 1 downto 0);
signal mux_ouput_to_matrixA1,mux_output_to_matrixA2_address: ...
   std_logic_vector (address_width - 1 downto 0);
signal read_address_matrixC_to_address_matrixC, ...
   mux_output_WriteAddressMatrixC_to_write_address_matrixC: ...
   std_logic_vector (address_width - 1 downto 0);
signal output_matrix_values_to_mux_input1_matrixA_values, ...
   output_matrix_values_to_mux_input2_matrixA_values: ...
   std_logic_vector(data_width - 1 downto 0);
signal mux_output_matrixA_values_to_conv2_input1, ...
   output_matrix_values_to_mux_input1_matrixB_values: ...
   std_logic_vector(data_width - 1 downto 0);
signal output_matrix_values_to_mux_input2_matrixB_values, ...
   mux_output_matrixB_to_conv2_input2: std_logic_vector(data_width...
    - 1 downto 0);
signal output_matrixC_values_to_mux_input_values, ...
   mux_output_MatrixC_values_to_matrixA2_values: std_logic_vector(...
   data_width - 1 downto 0);
signal mux_output_MatrixC_values_to_matrixReduce_values: ...
   std_logic_vector(data_width - 1 downto 0);
signal output_data_matrixC_to_matrixC_values: std_logic_vector(...
   data_width - 1 downto 0);
signal execute_to_mux_input4, execute_to_mux_input8, ...
   mux_out_execute_to_boothconv2: std_logic;
signal mult_ready_to_mux_input1, mult_done_to_mux_input2: ...
   std_logic;


begin

hresultcontrolbooth1: hresultcontrolbooth2 port map (clk => clk,
reset => reset,
enable_control => enable_conv2_module,
update_reg_enable => update_reg_enable_to_mux_input1,
writeEnable_matrixC =>
control_writeEnable_memC_to_WriteEnableMatrixC_mux_input3,
reset_reg => control_reset_reg_to_hvcontrol_reset_mux_input2,
enable_reduce => enable_reduce_to_hResultReduceControl,
execute => execute_to_mux_input4,
read_address_matrixA => read_address_matrixA_to_mux_input1,
read_address_matrixB => read_address_matrixB_to_mask1x5,
write_address_matrixC =>
write_address_matrixC_to_WriteAddressMatrix_mux_input1,
mux_enable => mux_enable_to_mux_select);
```

```
323
324 ------------------------------------------------------------------
325 hresultreducecontrolbooth1: hresultreducecontrolbooth2 port map(
326 clk => clk,
327 reset => reset,
328 enable_control => enable_reduce_to_hResultReduceControl,
329 writeEnable_matrix => writeEnable_matrix_to_MatrixA2,
330 enable_vresult => enable_vresult_to_vResultControl,
331 read_address_hResult =>
332 read_address_hResult_to_mux_input1_readAddressMatrixC,
333 write_address_reducehResult =>
334 write_address_reducehResult_to_mux_input1_matrixA2_address,
335 mux_enable => mux_enable_to_mux_select_matrixA2);
336
337 ------------------------------------------------------------------
338 vresultcontrolbooth1: vresultcontrolbooth2 port map (clk => clk,
339 reset => reset,
340 enable_control => enable_vresult_to_vResultControl,
341 update_reg_enable => update_reg_enable_to_mux_input5,
342 writeEnable_matrixC => control_writeEnable_memC_to_mux_input7,
343 reset_reg => control_reset_reg_to_mux_input6,
344 enable_reduce => enable_reduce_to_vResultReduceControl_enable,
345 execute => execute_to_mux_input8,
346 read_address_matrixA =>
347 write_address_reducevResult_to_mux_input2_matrixA2_address,
348 read_address_matrixB => read_address_matrixB_to_mask5x1,
349 write_address_matrixC =>
350 write_address_matrixC_to_WriteAddressMatrix_mux_input2);
351
352 ------------------------------------------------------------------
353 vresultreducecontrolbooth1 :vresultreducecontrolbooth2 port map(
354 clk => clk,
355 reset => reset,
356 enable_control => enable_reduce_to_vResultReduceControl_enable,
357 writeEnable_matrix => writeEnable_matrix_to_MatrixReduce,
358 enable_vresult => reduce_module_finished,
359 read_address_vResult =>
360 read_address_vResult_to_mux_input2_readAddressMatrixC,
361 write_address_reducevResult =>
362 write_address_reducevResult_to_MatrixReduce);
363
364 ------------------------------------------------------------------
365 mux2to1_matrixA1: mux2to1 port map(sel => enable_matrixA,
366 input1 => read_address_matrixA_to_mux_input1,
367 input2 => read_address_matrixA,
368 mux_output => mux_ouput_to_matrixA1);
369
370 ------------------------------------------------------------------
371 mux2to1_matrixA_values: mux2to1data_width port map(
372 sel => mux_enable_to_mux_select,
373 input1 => output_matrix_values_to_mux_input1_matrixA_values,
374 input2 => output_matrix_values_to_mux_input2_matrixA_values,
```

```
375 mux_output => mux_output_matrixA_values_to_conv2_input1);
376
377 -------------------------------------------------------------------
378 mux2to1_matrixB_values: mux2to1data_width port map(
379 sel => mux_enable_to_mux_select,
380 input1 => output_matrix_values_to_mux_input1_matrixB_values,
381 input2 => output_matrix_values_to_mux_input2_matrixB_values,
382 mux_output => mux_output_matrixB_to_conv2_input2);
383
384 -------------------------------------------------------------------
385 mux8to4_Enable1: mux8to4enable port map (
386 sel => mux_enable_to_mux_select,
387 input1 => update_reg_enable_to_mux_input1,
388 input2 => control_reset_reg_to_hvcontrol_reset_mux_input2,
389 input3 =>
390 control_writeEnable_memC_to_WriteEnableMatrixC_mux_input3,
391 input4 => execute_to_mux_input4,
392 input5 => update_reg_enable_to_mux_input5,
393 input6 => control_reset_reg_to_mux_input6,
394 input7 => control_writeEnable_memC_to_mux_input7,
395 input8 => execute_to_mux_input8,
396 mux_output1 => mux_output_to_conv2_update_reg,
397 mux_output2 => mux_output_to_conv2_reset,
398 mux_output3 => mux_out_writeEnableMatrixC_to_writeEnableMatrixC,
399 mux_output4 => mux_out_execute_to_boothconv2);
400
401 -------------------------------------------------------------------
402 mux2to4_Enable1: mux2to4enable port map (
403 sel => mux_enable_to_mux_select,
404 input1 => mult_ready_to_mux_input1,
405 input2 => mult_done_to_mux_input2,
406 mux_output1 => mult_ready_hresult,
407 mux_output2 => mult_done_hresult,
408 mux_output3 => mult_ready_vresult,
409 mux_output4 => mult_done_vresult);
410
411 -------------------------------------------------------------------
412 mux2to1_ReadAddressMatrixC: mux2to1 port map(
413 sel => mux_enable_to_mux_select_matrixA2,
414 input1 => read_address_hResult_to_mux_input1_readAddressMatrixC,
415 input2 => read_address_vResult_to_mux_input2_readAddressMatrixC,
416 mux_output => read_address_matrixC_to_address_matrixC);
417
418 -------------------------------------------------------------------
419 mux2to1_WriteAddressMatrixC: mux2to1 port map(
420 sel => mux_enable_to_mux_select,
421 input1 => write_address_matrixC_to_WriteAddressMatrix_mux_input1,
422 input2 => write_address_matrixC_to_WriteAddressMatrix_mux_input2,
423 mux_output =>
424 mux_output_WriteAddressMatrixC_to_write_address_matrixC);
425
426 -------------------------------------------------------------------
```

```
427 mux1to2_matrixA2_ReduceMatrix_values: mux1to2data_width port map(
428 sel => mux_enable_to_mux_select_matrixA2,
429 input => output_matrixC_values_to_mux_input_values,
430 mux_output1 => mux_output_MatrixC_values_to_matrixA2_values,
431 mux_output2 => mux_output_MatrixC_values_to_matrixReduce_values);
432
433 ----------------------------------------------------------------
434 mem_matrixA1: mem_matrix port map (clk => clk, reset => reset,
435 writeEnable => enable_matrixA,
436 matrix_values => input_matrixA_values,
437 write_address => mux_ouput_to_matrixA1,
438 read_address => mux_ouput_to_matrixA1,
439 output_matrix_values =>
440 output_matrix_values_to_mux_input1_matrixA_values);
441
442 ----------------------------------------------------------------
443 mem_matrixA2: mem_matrix port map (clk => clk, reset => reset,
444 writeEnable => writeEnable_matrix_to_MatrixA2,
445 matrix_values => mux_output_MatrixC_values_to_matrixA2_values,
446 write_address =>
447 write_address_reducehResult_to_mux_input1_matrixA2_address,
448 read_address =>
449 write_address_reducevResult_to_mux_input2_matrixA2_address,
450 output_matrix_values =>
451 output_matrix_values_to_mux_input2_matrixA_values);
452
453 ----------------------------------------------------------------
454 mask1x5mem_matrix0: mask1x5mem_matrix port map (clk => clk,
455 reset => reset,
456 read_address => read_address_matrixB_to_mask1x5,
457 output_matrix_values =>
458 output_matrix_values_to_mux_input1_matrixB_values);
459
460 ----------------------------------------------------------------
461 mask5x1mem_matrix0: mask5x1mem_matrix port map (clk => clk,
462 reset => reset,
463 read_address => read_address_matrixB_to_mask5x1,
464 output_matrix_values =>
465 output_matrix_values_to_mux_input2_matrixB_values);
466
467 ----------------------------------------------------------------
468 boothconv2_1: boothconv2 port map (clk => clk,
469 reset => mux_output_to_conv2_reset,
470 execute => mux_out_execute_to_boothconv2,
471 update_reg_enable => mux_output_to_conv2_update_reg,
472 input_matrixA_values => mux_output_matrixA_values_to_conv2_input1,
473 input_matrixB_values  => mux_output_matrixB_to_conv2_input2,
474 output_data_matrixC => output_data_matrixC_to_matrixC_values,
475 mult_ready => mult_ready_to_mux_input1,
476 mult_done => mult_done_to_mux_input2);
477
478 ----------------------------------------------------------------
```

```
479 mem_matrixC: mem_matrix port map (clk => clk, reset => reset,
480 writeEnable  => mux_out_writeEnableMatrixC_to_writeEnableMatrixC,
481 matrix_values => output_data_matrixC_to_matrixC_values,
482 write_address =>
483 mux_output_WriteAddressMatrixC_to_write_address_matrixC,
484 read_address => read_address_matrixC_to_address_matrixC,
485 output_matrix_values =>
486 output_matrixC_values_to_mux_input_values);
487
488 ----------------------------------------------------------------
489 mem_matrixReduce: mem_matrix port map (clk => clk, reset => reset,
490 writeEnable  => writeEnable_matrix_to_MatrixReduce,
491 matrix_values => mux_output_MatrixC_values_to_matrixReduce_values,
492 write_address => write_address_reducevResult_to_MatrixReduce,
493 read_address => read_address_matrixReduce,
494 output_matrix_values => output_data_matrixReduce);
495
496 end structure_reduceconv2_modulebooth2;
497 ----------------------------------------------------------------
```

C.5 Compute Derivatives Fx using Booth Multiplier

Listing C.5: The FxComputeDerivativesBoothConv2Module.vhd VHDL file. (appendix3/FxComputeDerivativesBoothConv2Module.vhd)

```
1  -- Capt. Jason Shirley
2  -- This is the FxComputeDerivatives Module using the
3  -- Booth Multiplier.These Modules are structually connected
4
5
6  library ieee;
7  use ieee.std_logic_1164.all;
8  use ieee.std_logic_unsigned.all;
9  use ieee.std_logic_arith.all;
10 use ieee.numeric_std.all;
11
12 ----------------------------------------------------------
13
14 entity fxcomputederivativesboothconv2_module is
15
16    generic (address_width: integer := 8;
17             data_width: integer := 24);
18
19    port (clk                    : in std_logic;
20          reset                  : in std_logic;
21          enable_matrix_A_B      : in std_logic;
22          enable_conv2_module    : in std_logic;
23          read_address_matrixA   : in std_logic_vector (...
               address_width - 1 downto 0);
24          read_address_matrixB   : in std_logic_vector (...
               address_width - 1 downto 0);
```

```
25          read_address_matrixC  : in std_logic_vector (...
               address_width - 1 downto 0);
26          input_matrixA_values  : in std_logic_vector (data_width -...
               1 downto 0);
27          input_matrixB_values  : in std_logic_vector (data_width -...
               1 downto 0);
28          output_data_matrixC   : out std_logic_vector(data_width -...
               1 downto 0);
29          enable_fxcomputederivative_complete : out std_logic;
30          writeEnable_matrix_conv2A : out std_logic;
31          writeEnable_matrix_conv2B : out std_logic;
32          mult_ready_conv2A     : out std_logic;
33          mult_done_conv2A      : out std_logic;
34          mult_ready_conv2B     : out std_logic;
35          mult_done_conv2B      : out std_logic
36               );
37
38 end fxcomputederivativesboothconv2_module;
39
40 ----------------------------------------------------------
41
42 architecture structure_fxcomputederivativesboothconv2_module of ...
      fxcomputederivativesboothconv2_module is
43
44 component boothconv2_module is
45
46    generic (address_width: integer := 8;
47             data_width: integer := 24);
48
49    port (clk                   : in std_logic;
50          reset                 : in std_logic;
51          enable_matrixA        : in std_logic;
52          enable_conv2_module   : in std_logic;
53          read_address_matrixA  : in std_logic_vector (...
               address_width - 1 downto 0);
54          read_address_matrixC  : in std_logic_vector (...
               address_width - 1 downto 0);
55          input_matrixA_values  : in std_logic_vector (data_width -...
               1 downto 0);
56          output_data_matrixC   : out std_logic_vector(data_width -...
               1 downto 0);
57          enable_complete       : out std_logic;
58          writeEnable_matrixC   : out std_logic;
59          mult_ready            : out std_logic;
60          mult_done             : out std_logic
61               );
62
63 end component boothconv2_module;
64
65 ----------------------------------------------------------
66 component adder is
67
```

```
68          generic (data_width: integer := 24);
69
70     port (input_mult_data  : in std_logic_vector (data_width - 1 ...
           downto 0);
71              input_reg_data   : in std_logic_vector (data_width - 1 ...
                   downto 0);
72              output_data      : out std_logic_vector (data_width - 1...
                    downto 0)
73 );
74
75 end component adder;
76 ------------------------------------------------------------
77 component fxfyftcontroladder is
78
79          generic (address_width: integer := 8; -- address_width ...
               must be an even number
80                    n1: integer := 5;
81                   n2: integer := 7);
82
83 -- address_width must be an even number
84 -- n1 and n2 are the size of the input matrix for example 6x8
85 -- matrix starts at zero so, n1 = (6-1) = 5, n2 = (8-1) = 7.
86
87     port (clk                                  : in std_logic;
88           reset                                : in std_logic;
89           enable_control_conv2A                : in std_logic;
90           enable_control_conv2B                : in std_logic;
91           writeEnable_matrix_image1_image2     : out std_logic;
92           enable_fxfy_complete                 : out std_logic;
93           read_address_matrix_image1           : out std_logic_vector...
                 (address_width - 1 downto 0);
94           read_address_matrix_image2           : out std_logic_vector...
                 (address_width - 1 downto 0);
95           write_address_matrix_image1_image2 : out std_logic_vector...
                 (address_width - 1 downto 0)
96           );
97
98 end component fxfyftcontroladder;
99 ------------------------------------------------------------
100 component mem_matrix is
101
102          generic (address_width: integer := 8;
103                   data_width: integer := 24);
104
105     port (clk                    : in std_logic;
106           reset                  : in std_logic;
107           writeEnable            : in std_logic;
108             matrix_values          : in std_logic_vector (data_width ...
                 - 1 downto 0);
109             write_address          : in std_logic_vector (...
                 address_width - 1 downto 0);
```

```
110             read_address          : in std_logic_vector (...
                   address_width - 1 downto 0);
111             output_matrix_values : out std_logic_vector(data_width ...
                   - 1 downto 0)
112             );
113
114 end component mem_matrix;
115
116 ----------------------------------------------------------
117
118 signal enable_complete_to_enable_control_conv2A, ...
       enable_complete_to_enable_conrol_conv2B: std_logic;
119 signal writeEnable_matrix_image1_image2_to_writeEnable: std_logic;
120 signal mux_ouput_matrixA, mux_ouput_matrixB :std_logic_vector (...
       address_width - 1 downto 0);
121 signal control_matrixA_to_muxA_input1, ...
       control_matrixB_to_muxB_input1:std_logic_vector (address_width ...
       - 1 downto 0);
122 signal write_address_matrix_image1_image2_to_writeEnable:...
       std_logic_vector (address_width - 1 downto 0);
123 signal read_address_matrix_image1_to_read_address_conv2A, ...
       read_address_matrix_image2_to_read_address_conv2B:...
       std_logic_vector (address_width - 1 downto 0);
124 signal output_data_matrix_conv2A_to_input_mult_data, ...
       output_data_matrix_conv2B_to_input_reg_data: std_logic_vector(...
       data_width - 1 downto 0);
125 signal output_data_to_mem_matrix_input_values: std_logic_vector(...
       data_width - 1 downto 0);
126
127 begin
128
129 boothconv2_module1: boothconv2_module port map (clk => clk,
130 reset => reset,
131 enable_matrixA => enable_matrix_A_B,
132 enable_conv2_module => enable_conv2_module,
133 read_address_matrixA => read_address_matrixA,
134 read_address_matrixC =>
135 read_address_matrix_image1_to_read_address_conv2A,
136 input_matrixA_values => input_matrixA_values,
137 output_data_matrixC =>
138 output_data_matrix_conv2A_to_input_mult_data,
139 enable_complete => enable_complete_to_enable_control_conv2A,
140 writeEnable_matrixC => writeEnable_matrix_conv2A,
141 mult_ready => mult_ready_conv2A,
142 mult_done => mult_done_conv2A);
143
144 boothconv2_module2: boothconv2_module port map (clk => clk,
145 reset => reset,
146 enable_matrixA => enable_matrix_A_B,
147 enable_conv2_module => enable_conv2_module,
148 read_address_matrixA => read_address_matrixB,
149 read_address_matrixC =>
```

```
150 read_address_matrix_image2_to_read_address_conv2B,
151 input_matrixA_values => input_matrixB_values,
152 output_data_matrixC =>
153 output_data_matrix_conv2B_to_input_reg_data,
154 enable_complete => enable_complete_to_enable_conrol_conv2B,
155 writeEnable_matrixC => writeEnable_matrix_conv2B,
156 mult_ready => mult_ready_conv2B,
157 mult_done => mult_done_conv2B);
158
159 fxfyftcontroladder0: fxfyftcontroladder port map (clk => clk,
160 reset => reset,
161 enable_control_conv2A => enable_complete_to_enable_control_conv2A,
162 enable_control_conv2B => enable_complete_to_enable_conrol_conv2B,
163 writeEnable_matrix_image1_image2 =>
164 writeEnable_matrix_image1_image2_to_writeEnable,
165 enable_fxfy_complete => enable_fxcomputederivative_complete,
166 read_address_matrix_image1 =>
167 read_address_matrix_image1_to_read_address_conv2A,
168 read_address_matrix_image2 =>
169 read_address_matrix_image2_to_read_address_conv2B,
170 write_address_matrix_image1_image2 =>
171 write_address_matrix_image1_image2_to_writeEnable);
172
173 mem_matrixC: mem_matrix port map (clk => clk,
174 reset => reset,
175 writeEnable =>
176 writeEnable_matrix_image1_image2_to_writeEnable,
177 matrix_values => output_data_to_mem_matrix_input_values,
178 write_address =>
179 write_address_matrix_image1_image2_to_writeEnable,
180 read_address => read_address_matrixC,
181 output_matrix_values => output_data_matrixC);
182
183 adder1: adder port map (input_mult_data =>
184 output_data_matrix_conv2A_to_input_mult_data,
185 input_reg_data => output_data_matrix_conv2B_to_input_reg_data,
186 output_data => output_data_to_mem_matrix_input_values);
187
188 end structure_fxcomputederivativesboothconv2_module;
```

C.6 Compute Derivatives Fy using Booth Multiplier

Listing C.6: The FyComputeDerivativesBoothConv2Module.vhd VHDL file. (appendix3/FyComputeDerivativesBoothConv2Module.vhd)

```
1  -- Capt. Jason Shirley
2  -- This is the FyComputeDerivatives Module using the
3  -- Booth Multiplier. These Modules are structually connected
4
5
6  library ieee;
7  use ieee.std_logic_1164.all;
8  use ieee.std_logic_unsigned.all;
9  use ieee.std_logic_arith.all;
10 use ieee.numeric_std.all;
11
12 ----------------------------------------------------------------
13
14 entity fycomputederivativesboothconv2_module is
15
16    generic (address_width: integer := 8;
17             data_width: integer := 24);
18
19    port (clk                    : in std_logic;
20          reset                  : in std_logic;
21          enable_matrix_A_B      : in std_logic;
22          enable_conv2_module    : in std_logic;
23          read_address_matrixA   : in std_logic_vector (...
               address_width - 1 downto 0);
24          read_address_matrixB   : in std_logic_vector (...
               address_width - 1 downto 0);
25          read_address_matrixC   : in std_logic_vector (...
               address_width - 1 downto 0);
26          input_matrixA_values   : in std_logic_vector (data_width -...
                1 downto 0);
27          input_matrixB_values   : in std_logic_vector (data_width -...
                1 downto 0);
28          output_data_matrixC    : out std_logic_vector(data_width -...
                1 downto 0);
29          enable_fycomputederivative_complete : out std_logic;
30          writeEnable_matrix_conv2A : out std_logic;
31          writeEnable_matrix_conv2B : out std_logic;
32          mult_ready_conv2A      : out std_logic;
33          mult_done_conv2A       : out std_logic;
34          mult_ready_conv2B      : out std_logic;
35          mult_done_conv2B       : out std_logic
36               );
37
38 end fycomputederivativesboothconv2_module;
39
40 ----------------------------------------------------------------
41
```

```
42 architecture structure_fycomputederivativesboothconv2_module of ...
       fycomputederivativesboothconv2_module is
43
44 component boothconv2_module is
45
46    generic (address_width: integer := 8;
47             data_width: integer := 24);
48
49    port (clk                    : in std_logic;
50          reset                  : in std_logic;
51          enable_matrixA         : in std_logic;
52          enable_conv2_module    : in std_logic;
53          read_address_matrixA   : in std_logic_vector (...
              address_width - 1 downto 0);
54          read_address_matrixC   : in std_logic_vector (...
              address_width - 1 downto 0);
55          input_matrixA_values   : in std_logic_vector (data_width -...
               1 downto 0);
56          output_data_matrixC    : out std_logic_vector(data_width -...
               1 downto 0);
57          enable_complete        : out std_logic;
58          writeEnable_matrixC    : out std_logic;
59          mult_ready             : out std_logic;
60          mult_done              : out std_logic
61               );
62
63 end component boothconv2_module;
64
65 ----------------------------------------------------------------
66
67 component adder is
68
69         generic (data_width: integer := 24);
70
71    port (input_mult_data   : in std_logic_vector (data_width - 1 ...
          downto 0);
72             input_reg_data    : in std_logic_vector (data_width - 1 ...
                 downto 0);
73             output_data       : out std_logic_vector (data_width - 1...
                  downto 0)
74             );
75
76 end component adder;
77 ----------------------------------------------------------------
78
79 component fxfyftcontroladder is
80
81         generic (address_width: integer := 8;
82                  n1: integer := 5;
83                 n2: integer := 7);
84 -- address_width must be an even number
85 -- n1 and n2 are the size of the input matrix for example 6x8
```

```
86  -- matrix starts at zero so, n1 = (6-1) = 5, n2 = (8-1) = 7.
87
88     port (clk                                  : in std_logic;
89           reset                                : in std_logic;
90           enable_control_conv2A                : in std_logic;
91           enable_control_conv2B                : in std_logic;
92           writeEnable_matrix_image1_image2     : out std_logic;
93           enable_fxfy_complete                 : out std_logic;
94           read_address_matrix_image1           : out std_logic_vector...
                 (address_width - 1 downto 0);
95           read_address_matrix_image2           : out std_logic_vector...
                 (address_width - 1 downto 0);
96           write_address_matrix_image1_image2 : out std_logic_vector...
                 (address_width - 1 downto 0)
97           );
98
99  end component fxfyftcontroladder;
100 ------------------------------------------------------------------
101
102 component mem_matrix is
103
104         generic (address_width: integer := 8;
105                  data_width: integer := 24);
106
107    port (clk                    : in std_logic;
108          reset                  : in std_logic;
109          writeEnable            : in std_logic;
110               matrix_values          : in std_logic_vector (...
                     data_width - 1 downto 0);
111               write_address          : in std_logic_vector (...
                     address_width - 1 downto 0);
112               read_address           : in std_logic_vector (...
                     address_width - 1 downto 0);
113               output_matrix_values : out std_logic_vector(...
                     data_width - 1 downto 0)
114               );
115
116 end component mem_matrix;
117
118 ------------------------------------------------------------------
119
120 signal enable_complete_to_enable_control_conv2A, ...
       enable_complete_to_enable_conrol_conv2B: std_logic;
121 signal writeEnable_matrix_image1_image2_to_writeEnable: std_logic;
122 signal mux_ouput_matrixA, mux_ouput_matrixB :std_logic_vector (...
       address_width - 1 downto 0);
123 signal control_matrixA_to_muxA_input1, ...
       control_matrixB_to_muxB_input1:std_logic_vector (address_width ...
       - 1 downto 0);
124 signal write_address_matrix_image1_image2_to_writeEnable:...
       std_logic_vector (address_width - 1 downto 0);
```

```
125 signal read_address_matrix_image1_to_read_address_conv2A, ...
       read_address_matrix_image2_to_read_address_conv2B:...
       std_logic_vector (address_width - 1 downto 0);
126 signal output_data_matrix_conv2A_to_input_mult_data, ...
       output_data_matrix_conv2B_to_input_reg_data: std_logic_vector(...
       data_width - 1 downto 0);
127 signal output_data_to_mem_matrix_input_values: std_logic_vector(...
       data_width - 1 downto 0);
128
129 begin
130
131 boothconv2_module1: boothconv2_module port map (clk => clk,
132 reset => reset,
133 enable_matrixA => enable_matrix_A_B,
134 enable_conv2_module => enable_conv2_module,
135 read_address_matrixA => read_address_matrixA,
136 read_address_matrixC =>
137 read_address_matrix_image1_to_read_address_conv2A,
138 input_matrixA_values => input_matrixA_values,
139 output_data_matrixC =>
140 output_data_matrix_conv2A_to_input_mult_data,
141 enable_complete => enable_complete_to_enable_control_conv2A,
142 writeEnable_matrixC => writeEnable_matrix_conv2A,
143 mult_ready => mult_ready_conv2A,
144 mult_done => mult_done_conv2A);
145
146 boothconv2_module2: boothconv2_module port map (clk => clk,
147 reset => reset,
148 enable_matrixA => enable_matrix_A_B,
149 enable_conv2_module => enable_conv2_module,
150 read_address_matrixA => read_address_matrixB,
151 read_address_matrixC =>
152 read_address_matrix_image2_to_read_address_conv2B,
153 input_matrixA_values => input_matrixB_values,
154 output_data_matrixC =>
155 output_data_matrix_conv2B_to_input_reg_data,
156 enable_complete => enable_complete_to_enable_conrol_conv2B,
157 writeEnable_matrixC => writeEnable_matrix_conv2B,
158 mult_ready => mult_ready_conv2B,
159 mult_done => mult_done_conv2B);
160
161 fxfyftcontroladder0: fxfyftcontroladder port map (clk => clk,
162 reset => reset,
163 enable_control_conv2A => enable_complete_to_enable_control_conv2A,
164 enable_control_conv2B => enable_complete_to_enable_conrol_conv2B,
165 writeEnable_matrix_image1_image2 =>
166 writeEnable_matrix_image1_image2_to_writeEnable,
167 enable_fxfy_complete => enable_fycomputederivative_complete,
168 read_address_matrix_image1 =>
169 read_address_matrix_image1_to_read_address_conv2A,
170 read_address_matrix_image2 =>
171 read_address_matrix_image2_to_read_address_conv2B,
```

```
172 write_address_matrix_image1_image2 =>
173 write_address_matrix_image1_image2_to_writeEnable);
174
175 mem_matrixC: mem_matrix port map (clk => clk,
176 reset => reset,
177 writeEnable =>
178 writeEnable_matrix_image1_image2_to_writeEnable,
179 matrix_values => output_data_to_mem_matrix_input_values,
180 write_address =>
181 write_address_matrix_image1_image2_to_writeEnable,
182 read_address => read_address_matrixC,
183 output_matrix_values => output_data_matrixC);
184
185 adder1: adder port map (
186 input_mult_data => output_data_matrix_conv2A_to_input_mult_data,
187 input_reg_data => output_data_matrix_conv2B_to_input_reg_data,
188 output_data => output_data_to_mem_matrix_input_values);
189
190 end structure_fycomputederivativesboothconv2_module;
```

C.7 Compute Derivatives Ft using Booth Multiplier

Listing C.7: The FtComputeDerivativesBoothConv2Module.vhd VHDL file. (appendix3/FtComputeDerivativesBoothConv2Module.vhd)

```
1  -- Capt. Jason Shirley
2  -- This is the FtComputeDerivatives Module using the
3  -- Booth Multiplier. These Modules are structually connected
4
5
6  library ieee;
7  use ieee.std_logic_1164.all;
8  use ieee.std_logic_unsigned.all;
9  use ieee.std_logic_arith.all;
10 use ieee.numeric_std.all;
11
12 ----------------------------------------------------------
13
14 entity ftcomputederivativesboothconv2_module is
15
16    generic (address_width: integer := 8;
17             data_width: integer := 24);
18
19    port (clk                    : in std_logic;
20          reset                  : in std_logic;
21          enable_matrix_A_B      : in std_logic;
22          enable_conv2_module    : in std_logic;
23          read_address_matrixA   : in std_logic_vector (...
               address_width - 1 downto 0);
24          read_address_matrixB   : in std_logic_vector (...
               address_width - 1 downto 0);
```

```
25          read_address_matrixC  : in std_logic_vector (...
               address_width - 1 downto 0);
26          input_matrixA_values  : in std_logic_vector (data_width -...
               1 downto 0);
27          input_matrixB_values  : in std_logic_vector (data_width -...
               1 downto 0);
28          output_data_matrixC   : out std_logic_vector(data_width -...
               1 downto 0);
29          enable_ftcomputederivative_complete : out std_logic;
30          writeEnable_matrix_conv2A : out std_logic;
31          writeEnable_matrix_conv2B : out std_logic;
32          mult_ready_conv2A     : out std_logic;
33          mult_done_conv2A      : out std_logic;
34          mult_ready_conv2B     : out std_logic;
35          mult_done_conv2B      : out std_logic
36            );
37
38 end ftcomputederivativesboothconv2_module;
39
40 ----------------------------------------------------------
41
42 architecture structure_ftcomputederivativesboothconv2_module of ...
      ftcomputederivativesboothconv2_module is
43
44 component boothconv2_image1_module is
45
46    generic (address_width: integer := 8;
47             data_width: integer := 24);
48
49    port (clk                  : in std_logic;
50          reset                : in std_logic;
51          enable_matrixA       : in std_logic;
52          enable_conv2_module  : in std_logic;
53          read_address_matrixA  : in std_logic_vector (...
               address_width - 1 downto 0);
54          read_address_matrixC  : in std_logic_vector (...
               address_width - 1 downto 0);
55          input_matrixA_values  : in std_logic_vector (data_width -...
               1 downto 0);
56          output_data_matrixC   : out std_logic_vector(data_width -...
               1 downto 0);
57          enable_complete       : out std_logic;
58          writeEnable_matrixC   : out std_logic;
59          mult_ready            : out std_logic;
60          mult_done             : out std_logic
61            );
62
63 end component boothconv2_image1_module;
64
65 ----------------------------------------------------------
66 component boothconv2_image2_module is
67
```

```
68     generic (address_width: integer := 8;
69              data_width: integer := 24);
70
71     port (clk                    : in std_logic;
72           reset                  : in std_logic;
73           enable_matrixA         : in std_logic;
74           enable_conv2_module    : in std_logic;
75           read_address_matrixA   : in std_logic_vector (...
                address_width - 1 downto 0);
76           read_address_matrixC   : in std_logic_vector (...
                address_width - 1 downto 0);
77           input_matrixA_values   : in std_logic_vector (data_width -...
                1 downto 0);
78           output_data_matrixC    : out std_logic_vector(data_width -...
                1 downto 0);
79           enable_complete        : out std_logic;
80           writeEnable_matrixC    : out std_logic;
81           mult_ready             : out std_logic;
82           mult_done              : out std_logic
83             );
84
85 end component boothconv2_image2_module;
86
87 ------------------------------------------------------------
88
89 component adder is
90
91         generic (data_width: integer := 24);
92
93     port (input_mult_data  : in std_logic_vector (data_width - 1 ...
          downto 0);
94           input_reg_data   : in std_logic_vector (data_width - 1 ...
                downto 0);
95           output_data      : out std_logic_vector (data_width - 1...
                downto 0)
96           );
97
98 end component adder;
99 ------------------------------------------------------------
100 component fxfyftcontroladder is
101
102         generic (address_width: integer := 8;
103                  n1: integer := 5;
104                 n2: integer := 7);
105 -- address_width must be an even number
106 -- n1 and n2 are the size of the input matrix for example 6x8
107 -- matrix starts at zero so, n1 = (6-1) = 5, n2 = (8-1) = 7.
108
109     port (clk                                  : in std_logic;
110           reset                                : in std_logic;
111           enable_control_conv2A                : in std_logic;
112           enable_control_conv2B                : in std_logic;
```

```
113         writeEnable_matrix_image1_image2   : out std_logic;
114         enable_fxfy_complete               : out std_logic;
115         read_address_matrix_image1         : out std_logic_vector...
                (address_width - 1 downto 0);
116         read_address_matrix_image2         : out std_logic_vector...
                (address_width - 1 downto 0);
117         write_address_matrix_image1_image2 : out std_logic_vector...
                (address_width - 1 downto 0)
118         );
119
120 end component fxfyftcontroladder;
121 ------------------------------------------------------------
122 component mem_matrix is
123
124        generic (address_width: integer := 8;
125                 data_width: integer := 24);
126
127   port (clk                  : in std_logic;
128         reset                : in std_logic;
129         writeEnable          : in std_logic;
130           matrix_values        : in std_logic_vector (data_width ...
                - 1 downto 0);
131           write_address        : in std_logic_vector (...
                address_width - 1 downto 0);
132           read_address         : in std_logic_vector (...
                address_width - 1 downto 0);
133           output_matrix_values : out std_logic_vector(data_width ...
                - 1 downto 0)
134           );
135
136 end component mem_matrix;
137
138 ------------------------------------------------------------
139
140 signal enable_complete_to_enable_control_conv2A, ...
      enable_complete_to_enable_conrol_conv2B: std_logic;
141 signal writeEnable_matrix_image1_image2_to_writeEnable: std_logic;
142 signal mux_ouput_matrixA, mux_ouput_matrixB :std_logic_vector (...
      address_width - 1 downto 0);
143 signal control_matrixA_to_muxA_input1, ...
      control_matrixB_to_muxB_input1:std_logic_vector (address_width ...
      - 1 downto 0);
144 signal write_address_matrix_image1_image2_to_writeEnable:...
      std_logic_vector (address_width - 1 downto 0);
145 signal read_address_matrix_image1_to_read_address_conv2A, ...
      read_address_matrix_image2_to_read_address_conv2B:...
      std_logic_vector (address_width - 1 downto 0);
146 signal output_data_matrix_conv2A_to_input_mult_data, ...
      output_data_matrix_conv2B_to_input_reg_data: std_logic_vector(...
      data_width - 1 downto 0);
147 signal output_data_to_mem_matrix_input_values: std_logic_vector(...
      data_width - 1 downto 0);
```

```
148
149 begin
150
151 boothconv2_image1_module1: boothconv2_image1_module port map (
152 clk => clk,
153 reset => reset,
154 enable_matrixA => enable_matrix_A_B,
155 enable_conv2_module => enable_conv2_module,
156 read_address_matrixA => read_address_matrixA,
157 read_address_matrixC =>
158 read_address_matrix_image1_to_read_address_conv2A,
159 input_matrixA_values => input_matrixA_values,
160 output_data_matrixC =>
161 output_data_matrix_conv2A_to_input_mult_data,
162 enable_complete => enable_complete_to_enable_control_conv2A,
163 writeEnable_matrixC => writeEnable_matrix_conv2A,
164 mult_ready => mult_ready_conv2A,
165 mult_done => mult_done_conv2A);
166
167 boothconv2_image2_module2: boothconv2_image2_module port map (
168 clk => clk,
169 reset => reset,
170 enable_matrixA => enable_matrix_A_B,
171 enable_conv2_module => enable_conv2_module,
172 read_address_matrixA => read_address_matrixB,
173 read_address_matrixC =>
174 read_address_matrix_image2_to_read_address_conv2B,
175 input_matrixA_values => input_matrixB_values,
176 output_data_matrixC =>
177 output_data_matrix_conv2B_to_input_reg_data,
178 enable_complete => enable_complete_to_enable_conrol_conv2B,
179 writeEnable_matrixC => writeEnable_matrix_conv2B,
180 mult_ready => mult_ready_conv2B,
181 mult_done => mult_done_conv2B);
182
183 fxfyftcontroladder0: fxfyftcontroladder port map (clk => clk,
184 reset => reset,
185 enable_control_conv2A =>
186 enable_complete_to_enable_control_conv2A,
187 enable_control_conv2B =>
188 enable_complete_to_enable_conrol_conv2B,
189 writeEnable_matrix_image1_image2 =>
190 writeEnable_matrix_image1_image2_to_writeEnable,
191 enable_fxfy_complete => enable_ftcomputederivative_complete,
192 read_address_matrix_image1 =>
193 read_address_matrix_image1_to_read_address_conv2A,
194 read_address_matrix_image2 =>
195 read_address_matrix_image2_to_read_address_conv2B,
196 write_address_matrix_image1_image2 =>
197 write_address_matrix_image1_image2_to_writeEnable);
198
199 mem_matrixC: mem_matrix port map (clk => clk, reset => reset,
```

```
200 writeEnable => writeEnable_matrix_image1_image2_to_writeEnable,
201 matrix_values => output_data_to_mem_matrix_input_values,
202 write_address =>
203 write_address_matrix_image1_image2_to_writeEnable,
204 read_address => read_address_matrixC,
205 output_matrix_values => output_data_matrixC);
206
207 adder1: adder port map (input_mult_data =>
208 output_data_matrix_conv2A_to_input_mult_data,
209 input_reg_data => output_data_matrix_conv2B_to_input_reg_data,
210 output_data => output_data_to_mem_matrix_input_values);
211
212 end structure_ftcomputederivativesboothconv2_module;
```

C.8 Matrix Transpose

Listing C.8: The TransposeMatrixModule.vhd VHDL file.
(appendix3/TransposeMatrixModule.vhd)

```
1  -- Capt. Jason Shirley
2  -- This is the Transpose Matrix Module
3  -- These Modules are structually connected
4
5
6  library ieee;
7  use ieee.std_logic_1164.all;
8  use ieee.std_logic_unsigned.all;
9  use ieee.std_logic_arith.all;
10 use ieee.numeric_std.all;
11
12 ------------------------------------------------------------
13
14 entity transpose_matrix_module is
15
16    generic (address_width: integer := 8;
17             data_width: integer := 24);
18
19    port (clk                           : in std_logic;
20          reset                         : in std_logic;
21          enable_transpose              : in std_logic;
22          enable_matrix_original        : in std_logic;
23          read_address_matrix_original  : in std_logic_vector (...
               address_width - 1 downto 0);
24          input_matrix_original_values  : in std_logic_vector (...
               data_width - 1 downto 0);
25          read_address_matrix_transpose : in std_logic_vector (...
               address_width - 1 downto 0);
26          output_data_matrix_transpose  : out std_logic_vector(...
               data_width - 1 downto 0);
27          enable_transpose_complete     : out std_logic
28            );
```

```
29
30 end transpose_matrix_module;
31
32 ------------------------------------------------------------
33
34 architecture structure_transpose_matrix_module of ...
      transpose_matrix_module is
35
36 component transposecontrol is
37
38         generic (address_width: integer := 8;
39                  n1: integer := 5;
40                 n2: integer := 7);
41 -- address_width must be an even number
42 -- n1 and n2 are the size of the input matrix for example 6x8
43 -- matrix starts at zero so, n1 = (6-1) = 5, n2 = (8-1) = 7.
44
45    port (clk                              : in std_logic;
46          reset                            : in std_logic;
47          enable_control                   : in std_logic;
48          writeEnable_matrix_transpose     : out std_logic;
49          enable_transpose_complete        : out std_logic;
50          read_address_matrix_original     : out std_logic_vector (...
               address_width - 1 downto 0);
51          write_address_matrix_transpose : out std_logic_vector (...
               address_width - 1 downto 0)
52          );
53
54 end component transposecontrol;
55
56
57 ------------------------------------------------------------
58 component mem_matrix is
59
60         generic (address_width: integer := 8;
61                  data_width: integer := 24);
62
63    port (clk                   : in std_logic;
64          reset                 : in std_logic;
65          writeEnable           : in std_logic;
66            matrix_values         : in std_logic_vector (data_width ...
                 - 1 downto 0);
67            write_address         : in std_logic_vector (...
                 address_width - 1 downto 0);
68            read_address          : in std_logic_vector (...
                 address_width - 1 downto 0);
69            output_matrix_values : out std_logic_vector(data_width ...
                 - 1 downto 0)
70            );
71
72 end component mem_matrix;
73
```

```
74 ------------------------------------------------------------
75 component mux2to1 is
76
77     generic (address_width: integer := 8);
78
79     port(sel         : in std_logic;
80          input1      : in std_logic_vector(address_width - 1 downto...
                0);
81          input2      : in std_logic_vector(address_width - 1 downto...
                0);
82          mux_output : out std_logic_vector(address_width - 1 ...
               downto 0)
83          );
84
85 end component mux2to1;
86
87 ------------------------------------------------------------
88
89 signal writeEnable_matrix_transpose_to_mem_matrix_transpose: ...
       std_logic;
90 signal control_matrix_original_to_mux_input1:std_logic_vector (...
       address_width - 1 downto 0);
91 signal control_address_transpose_to_transpose_matrix:...
       std_logic_vector (address_width - 1 downto 0);
92 signal mux_output_to_original_matrix:std_logic_vector (...
       address_width - 1 downto 0);
93 signal output_matrix_original_values_to_matrix_transpose_values: ...
       std_logic_vector(data_width - 1 downto 0);
94
95 begin
96
97 transposecontrol1: transposecontrol port map (clk => clk,
98 reset => reset,
99 enable_control => enable_transpose,
100 writeEnable_matrix_transpose =>
101 writeEnable_matrix_transpose_to_mem_matrix_transpose,
102 enable_transpose_complete => enable_transpose_complete,
103 read_address_matrix_original =>
104 control_matrix_original_to_mux_input1,
105 write_address_matrix_transpose =>
106 control_address_transpose_to_transpose_matrix);
107
108 mem_matrix_orginal: mem_matrix port map (clk => clk,
109 reset => reset,
110 writeEnable => enable_matrix_original,
111 matrix_values => input_matrix_original_values,
112 write_address => mux_output_to_original_matrix,
113 read_address => mux_output_to_original_matrix,
114 output_matrix_values =>
115 output_matrix_original_values_to_matrix_transpose_values);
116
117 mem_matrix_transpose: mem_matrix port map (clk => clk,
```

```
118 reset => reset,
119 writeEnable =>
120 writeEnable_matrix_transpose_to_mem_matrix_transpose,
121 matrix_values =>
122 output_matrix_original_values_to_matrix_transpose_values,
123 write_address => control_address_transpose_to_transpose_matrix,
124 read_address => read_address_matrix_transpose,
125 output_matrix_values => output_data_matrix_transpose);
126
127 mux2to1_matrix_orginal: mux2to1 port map (
128 sel => enable_matrix_original,
129 input1 => control_matrix_original_to_mux_input1,
130 input2 => read_address_matrix_original,
131 mux_output => mux_output_to_original_matrix);
132
133 end structure_transpose_matrix_module;
```

C.9 Pseudoinverse

Listing C.9: The PseudoinverseModule.vhd VHDL file.
(appendix3/PseudoinverseModule.vhd)

```
1  -- Capt. Jason Shirley
2  -- This is the PseudoinverseModule
3
4
5  library ieee;
6  use ieee.std_logic_1164.all;
7  use ieee.std_logic_unsigned.all;
8  use ieee.numeric_std.all;
9  --use ieee.std_logic_signed.all;
10 --use ieee.std_logic_arith.all;
11
12 ----------------------------------------------------------------
13
14 entity xresultbooth_module is
15
16    generic (address_width: integer := 8;
17             data_width: integer := 32);
18
19    port (clk                    : in std_logic;
20          reset                  : in std_logic;
21          enable_matrixAtA       : in std_logic; -- load A matrix ...
               value
22          enable_AtA_module      : in std_logic; -- start AtA
23          enable_matrixBvector   : in std_logic; -- load B matrix ...
               value
24          read_address_matrixA   : in signed (address_width - 1 ...
               downto 0);
25          read_address_matrixBvectors : in signed (address_width - ...
               1 downto 0);
```

```
26          read_address_matrixX    : in signed (address_width -1 ...
               downto 0);
27          input_matrixA_values    : in signed (data_width - 1 downto...
                0);
28          input_matrixBvectors_values : in signed(data_width - 1 ...
               downto 0);
29          output_data_matrixX     : out signed(data_width - 1 downto...
                0);
30          enable_finished         : out std_logic
31               );
32
33 end xresultbooth_module;
34
35 ----------------------------------------------------------------
36
37 architecture structure_xresultbooth_module of xresultbooth_module ...
      is
38
39 component yresultbooth_module is
40
41    generic (address_width: integer := 8;
42             data_width: integer := 32);
43
44    port (clk                     : in std_logic;
45          reset                   : in std_logic;
46          enable_matrixAtA        : in std_logic; -- load A matrix ...
               value
47          enable_AtA_module       : in std_logic; -- start AtA
48          enable_matrixBvector    : in std_logic; -- load B matrix ...
               value
49          read_address_matrixA    : in signed (address_width - 1 ...
               downto 0);
50          read_address_matrixBvectors : in signed (address_width - ...
               1 downto 0);
51          read_address_matrixU    : in signed (address_width - 1 ...
               downto 0);
52          read_address_matrixY    : in signed (address_width - 1 ...
               downto 0);
53          input_matrixA_values    : in signed (data_width - 1 downto...
                0);
54          input_matrixBvectors_values : in signed(data_width - 1 ...
               downto 0);
55          output_data_matrixY     : out signed(data_width - 1 downto...
                0);
56          output_data_matrixU     : out signed(data_width - 1 downto...
                0);
57          enable_finished         : out std_logic
58               );
59
60 end component yresultbooth_module;
61 ----------------------------------------------------------------
62 component xcontrolbooth is
```

```
63
64          generic (address_width: integer := 8;
65                   data_width: integer := 32);
66
67     port (clk                       : in std_logic;
68           reset                     : in std_logic;
69           enable_control            : in std_logic;
70           enable_predivider_register : out std_logic;
71           enable_postdivider_register: out std_logic;
72                enable_checkneg        : out std_logic;
73                enable_divider         : out std_logic;
74                mux_load_divider       : out std_logic;
75           muxU                      : out std_logic;
76           muxUXY                    : out signed(1 downto 0);
77           writeEnable_matrixX       : out std_logic;
78           enable_finished           : out std_logic;
79           execute                   : out std_logic;
80           read_address_matrixU      : out signed (address_width - 1 ...
                downto 0);
81           write_address_matrixX     : out signed (address_width - 1 ...
                downto 0);
82           read_address_matrixY      : out signed (address_width - 1 ...
                downto 0)
83             );
84
85 end component xcontrolbooth;
86
87 ----------------------------------------------------------------
88 component reciprocal_top Is
89     Generic(high_bit : natural := 31; fraction_size : natural := ...
           16);
90     Port(clk, reset, load, mux_control : In std_logic;
91        data_in : In signed(high_bit Downto 0); --signed
92        data_out : Out signed(high_bit Downto 0); --signed
93        overflow : Out std_logic);
94 end component reciprocal_top;
95
96 ----------------------------------------------------------------
97 component mux2to1data_width is
98
99     generic (data_width: integer := 32);
100
101    port(sel         : in std_logic;
102         input1      : in signed(data_width - 1 downto 0);
103         input2      : in signed(data_width - 1 downto 0);
104         mux_output  : out signed(data_width - 1 downto 0)
105         );
106
107 end component mux2to1data_width;
108
109 ----------------------------------------------------------------
110 component mux3to1data_width is
```

```
111
112     generic (data_width: integer := 32);
113
114     port(sel         : in signed (1 downto 0);
115          input1      : in signed (data_width - 1 downto 0);
116          input2      : in signed (data_width - 1 downto 0);
117          input3      : in signed (data_width - 1 downto 0);
118          mux_output : out signed (data_width - 1 downto 0)
119          );
120
121 end component mux3to1data_width;
122
123 ------------------------------------------------------------------
124 component mem_matrix is
125
126    generic (address_width: integer := 8;
127             data_width: integer := 32);
128
129    port (clk                   : in std_logic;
130          reset                 : in std_logic;
131          writeEnable           : in std_logic;
132          matrix_values         : in signed (data_width - 1 downto ...
                0);
133          write_address         : in signed (address_width - 1 ...
                downto 0);
134          read_address          : in signed (address_width - 1 ...
                downto 0);
135          output_matrix_values : out signed(data_width - 1 downto ...
                0)
136            );
137
138 end component mem_matrix;
139
140 ------------------------------------------------------------------
141 component booth2bit is
142
143         generic (data_width: integer := 32;
144                  shift_size: integer:= 8);
145
146    port (clk                   : in std_logic;
147          reset                 : in std_logic;
148          execute               : in std_logic;
149          input_matrixA_values : in signed(data_width - 1 downto 0)...
                ;
150          input_matrixB_values : in signed(data_width - 1 downto 0)...
                ;
151          mult_ready            : out std_logic;
152          booth2bit_output      : out signed(data_width - 1 downto ...
                0);
153          mult_done             : out std_logic
154          );
155
```

```
156 end component booth2bit;
157
158 -------------------------------------------------------------
159 component checkneg is
160
161         generic (data_width: integer := 32);
162
163    port (clk          : in std_logic;
164          enable       : in std_logic;
165          matrix_value : in signed (data_width - 1 downto 0);
166          output_value : out signed(data_width - 1 downto 0)
167          );
168
169 end component checkneg;
170
171 -------------------------------------------------------------
172 component adder is
173
174         generic (data_width: integer := 32);
175
176    port (input_mult_data  : in signed (data_width - 1 downto 0);
177               input_reg_data   : in signed (data_width - 1 downto ...
                      0);
178               output_data      : out signed (data_width - 1 downto...
                       0)
179               );
180
181 end  component adder;
182
183 -------------------------------------------------------------
184 component dff is
185
186         generic (data_width: integer := 32);
187
188 port (clk          : in std_logic;
189       enable       : in std_logic;
190       input_data   : in signed (data_width - 1 downto 0);
191       output_data : out signed (data_width - 1 downto 0)
192       );
193
194 end component dff;
195
196 -------------------------------------------------------------
197
198 signal enable_finished_to_xcontrol,...
      enable_predividercontrol_to_predivider_dff : std_logic;
199 signal enable_postdivider_registercontrol_to_postdivider_dff: ...
      std_logic;
200 signal enable_divider_to_load, mux_load_divider_to_mux_control: ...
      std_logic;
201 signal execute_to_execute, muxU_to_muxU, ...
      writeEnable_matrix_to_write_memoryX: std_logic;
```

```
202 signal muxUXY_to_muxUXY: signed (1 downto 0);
203 signal output_data_to_data_input, data_out_to_post_divider_dff: ...
      signed (data_width - 1 downto 0);
204 signal output_data_to_mux_input1, ...
      output_data_matrixU_to_predivider_dff: signed (data_width - 1 ...
      downto 0);
205 signal read_address_matrixU_control_to_read_addres_matrixU: signed...
       (address_width - 1 downto 0);
206 signal read_address_matrixY_control_to_read_addres_matrixY: signed...
       (address_width - 1 downto 0);
207 signal write_address_matrixX_control_to_write_address_matrixX: ...
      signed (address_width - 1 downto 0);
208 signal mux_output_to_boothinputA, output_data_matrixY_to_muxinput1...
      : signed (data_width - 1 downto 0);
209 signal booth2bit_output_to_input_value_memoryX: signed (data_width...
       - 1 downto 0);
210 signal mux_output_to_booth_inputB, output_data_to_muxinput3: ...
      signed (data_width - 1 downto 0);
211 signal checkneg_output_value_to_input_data_dff, ...
      dff_output_to_adder_inputA: signed (data_width - 1 downto 0);
212 signal enable_checkneg_to_checkneg: std_logic;
213
214
215 begin
216
217 yresultbooth_module1 :yresultbooth_module port map (clk => clk,
218 reset => reset,
219 enable_matrixAtA => enable_matrixAtA,
220 enable_AtA_module => enable_AtA_module,
221 enable_matrixBvector => enable_matrixBvector,
222 read_address_matrixA => read_address_matrixA,
223 read_address_matrixBvectors => read_address_matrixBvectors,
224 read_address_matrixU =>
225 read_address_matrixU_control_to_read_addres_matrixU,
226 read_address_matrixY =>
227 read_address_matrixY_control_to_read_addres_matrixY,
228 input_matrixA_values => input_matrixA_values,
229 input_matrixBvectors_values => input_matrixBvectors_values,
230 output_data_matrixY => output_data_matrixY_to_muxinput1,
231 output_data_matrixU => output_data_matrixU_to_predivider_dff,
232 enable_finished => enable_finished_to_xcontrol);
233
234 xcontrolbooth1 :xcontrolbooth port map(clk => clk,
235 reset => reset,
236 enable_control => enable_finished_to_xcontrol,
237 enable_predivider_register =>
238 enable_predividercontrol_to_predivider_dff,
239 enable_postdivider_register =>
240 enable_postdivider_registercontrol_to_postdivider_dff,
241 enable_checkneg => enable_checkneg_to_checkneg,
242 enable_divider => enable_divider_to_load,
243 mux_load_divider => mux_load_divider_to_mux_control,
```

```
244 muxU => muxU_to_muxU,
245 muxUXY => muxUXY_to_muxUXY,
246 writeEnable_matrixX => writeEnable_matrix_to_write_memoryX,
247 enable_finished => enable_finished,
248 execute => execute_to_execute,
249 read_address_matrixU =>
250 read_address_matrixU_control_to_read_addres_matrixU,
251 write_address_matrixX =>
252 write_address_matrixX_control_to_write_address_matrixX,
253 read_address_matrixY =>
254 read_address_matrixY_control_to_read_addres_matrixY);
255
256 reciprocal_top1: reciprocal_top port map (clk => clk,
257 reset => reset,
258 load => enable_divider_to_load,
259 mux_control => mux_load_divider_to_mux_control,
260 data_in => output_data_to_data_input,
261 data_out => data_out_to_post_divider_dff,
262 overflow => open);
263
264 mux2to1data_widthY: mux2to1data_width port map
265 sel => muxU_to_muxU,
266 input1 => output_data_to_mux_input1,
267 input2 => output_data_matrixU_to_predivider_dff,
268 mux_output => mux_output_to_boothinputA);
269
270 mux3to1data_widthUXY: mux3to1data_width port map
271 (sel => muxUXY_to_muxUXY,
272 input1 => output_data_matrixY_to_muxinput1,
273 input2 => booth2bit_output_to_input_value_memoryX,
274 input3 => output_data_to_muxinput3,
275 mux_output => mux_output_to_booth_inputB);
276
277 mem_matrixX: mem_matrix port map (clk => clk,
278 reset => reset,
279 writeEnable => writeEnable_matrix_to_write_memoryX,
280 matrix_values => booth2bit_output_to_input_value_memoryX,
281 write_address =>
282 write_address_matrixX_control_to_write_address_matrixX,
283 read_address => read_address_matrixX,
284 output_matrix_values => output_data_matrixX);
285
286 booth2bit1: booth2bit port map(clk => clk,
287 reset => reset,
288 execute => execute_to_execute,
289 input_matrixA_values => mux_output_to_boothinputA,
290 input_matrixB_values => mux_output_to_booth_inputB,
291 mult_ready => open,--
292 booth2bit_output => booth2bit_output_to_input_value_memoryX,
293 mult_done => open);--
294
295 adder1: adder port map(
```

```
296 input_mult_data => dff_output_to_adder_inputA,
297 input_reg_data => output_data_matrixY_to_muxinput1,
298 output_data => output_data_to_muxinput3);
299
300 checkneg1: checkneg port map(clk => clk,
301 enable => enable_checkneg_to_checkneg,
302 matrix_value => booth2bit_output_to_input_value_memoryX,
303 output_value => checkneg_output_value_to_input_data_dff);
304
305 dffpredivider: dff port map(clk => clk,
306 enable => enable_predividercontrol_to_predivider_dff,
307 input_data => output_data_matrixU_to_predivider_dff,
308 output_data => output_data_to_data_input);
309
310 dffpostdivider: dff port map(clk => clk,
311 enable =>
312 enable_postdivider_registercontrol_to_postdivider_dff,
313 input_data => data_out_to_post_divider_dff,
314 output_data => output_data_to_mux_input1);
315
316 dffpostcheckneg: dff port map(clk => clk,
317 enable =>
318 enable_postdivider_registercontrol_to_postdivider_dff,
319 input_data => checkneg_output_value_to_input_data_dff,
320 output_data => dff_output_to_adder_inputA);
321
322 end structure_xresultbooth_module;
323
324 ------------------------------------------------------------------------...
```

Bibliography

1. AFRL/SN, Sensors Directorate. "Data Collection Supports Sensors Development". $http://www.wpafb.af.mil/news/story\_print.asp?id = 123033809$, June 2006.
2. Arsalan, Muhammad and Maitham Shams. "Asynchronous Adiabatic Logic". 3720–3723. 2007. VL:.
3. Chang, J. Morris and S. Kagan Agun. "Design-for-reusability in VHDL". October 2001.
4. Chen, WenZhi and Huan Zheng. "Analysis of Power Saving Effect for Dynamic Power Management". 1–4. 2006. VL:.
5. Collin, Andrew. "Andrew Booth Computers at Birkbeck College". http://www.cs.man.ac.uk./CCS/res/res05.htm#e.
6. Edwards, Larson. *Elementary Linear Algebra*. Houghton Mifflin, 4 ed. edition, 2000. ISBN 0-395-96717-1.
7. Horn, B.K.P and B. G. Schunck. "Determining Optical Flow". volume 17, 185–203. 1981.
8. Khan, Sohai. "Lucas-Kanade". http://www.cs.ucf.edu/ vision/Code/Optical_Flow/Lucas%20Kanade.zip, May 2003.
9. Kilts, Steve. *Advanced FPGA Design*. Wiley, 2007. ISBN 978-0-470-05437-6.
10. Lucas, B.D and T. Kanade. "An iterative image registration technique with an application to stereo vision". 121–130. 1981.
11. Sirsi, Gurudev Bhat. "Leakage Power Optimization Flow", Sep 2004.
12. Strang, Gilbert. *Linear Algebra and its Applcations*. Thomson Brooks/Cole, 4 ed. edition, 2006. ISBN 0-03-010567-6.
13. Tirumalashetty, Vishwanadh and Hamid Mahmoodi. "Clock Gating and Negative Edge Triggering for Energy Recovery Clock". 1141–1144. 2007. VL:.
14. Tiwari, V., R. Donnelly, S. Malik, and R. Gonzalez. "Dynamic power management for microprocessors: a case study". 185–192. 1997. VL:.
15. Virginia Tech, Dr. Dong S. Ha. "http://www.vtvt.ece.vt.edu/vlsidesign/download.php".
16. Yeung, W. K., Cheong-Fat Chan, Chiu-Sing Choy, and Kong-Pang Pun. "Clock recovery circuit with adiabatic technology (quasi-static CMOS logic)". volume 2, II–185–II–187 vol.2. 2003.

Vita

Captain Jason W. Shirley graduated from Valencia High School in Placentia, CA in 1997. He attended Fullerton Junior College for three years before transferring to California State Polytechnic University of Pomona as an Electrical Engineering student, and graduated in 2003 with a Bachelor's of Science in Electrical Engineering. Shortly after, Captain Shirley joined the US Air Force and was commissioned as an officer in September 2003 through Officer's Training School. Captain Shirley's first assignment was at the Oklahoma City Air Logistics Center at Tinker Air Force Base, OK. During his tour at Tinker he earned a Master's of Science in Aerospace Administration from Southeastern Oklahoma State University in 2006. His second assignment was at the Air Force Institute of Technology (AFIT) to complete a Master's of Science in Electrical Engineering. He will graduate in March of 2008.

Permanent address: 2950 Hobson Way
Air Force Institute of Technology
Wright-Patterson AFB, OH 45433

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From – To)*
27-03-2008	**Master Thesis**	May 2006-March 2008

4. TITLE AND SUBTITLE

HARDWARE ALGORITHM IMPLEMENTATION FOR MISSION SPECIFIC PROCESSING

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Shirley, Jason W., Captain, USAF

5d. PROJECT NUMBER
ENG 07-312

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765 DSN: 785-3636

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GE/ENG/08-27

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Douglas Summers-Stay, 937-674-9033, Douglas.Summers-Stay@WPAFB.AF.MIL
AFRL/RYAT (AFMC)
2241 Avionics Circle
WPAFB, OH 45433-7765

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

13. SUPPLEMENTARY NOTES

14. ABSTRACT

There is a need to expedite the process of designing military hardware to stay ahead of the adversary. The core of this project was to build reusable, synthesizeable libraries to make this a possibility. In order to build these libraries, Matlab commands and functions, such as Conv2, Round, Floor, Pinv, etc., had to be converted into reusable VHDL modules. These modules make up reusable libraries for the Mission Specific Process (MSP) which will support AFRL/RY. The MSP allows the VLSI design process to be completed in a mere matter of days or months using an FPGA or ASIC design, as opposed to the current way of developing a system which can take 1-2 years to complete. By having the libraries built, the components can be implemented in an FPGA or ASIC design over and over again. The libraries make it possible to make upgrades to weapons systems to meet the ever-changing needs the War Fighter faces. MSP makes it possible to develop various algorithms, including algorithms implemented in Matlab. The MSP libraries were built and tested using TSMC 250-nm technology library from the Taiwan Semiconductor Manufacturing Company. They were also synthesized for an FPGA. The modules were all synthesized using the CAD tools from Cadence and Mentor Graphics. Power, area, and delay results for each module were presented.

15. SUBJECT TERMS
Mission Specific Process, Field Programmable Gate Array, Optical Flow, VHDL

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
REPORT	**ABSTRACT**	**c. THIS PAGE**			**Yong C. Kim, PhD (ENG)**
U	**U**	**U**	**UU**	142	**19b. TELEPHONE NUMBER** *(Include area code)* (937) 255-3636, ext 4620; email: Yong.Kim@afit.edu