DEVELOPMENT OF A SMART CAMERA SYSTEM USING A SYSTEM ON MODULE FPGA

by

Connor Aquila Dack

A thesis submitted in partial fulfillment
of the requirements for the degree

of

Master of Science

in

Electrical Engineering

MONTANA STATE UNIVERSITY
Bozeman, Montana

April, 2017

©COPYRIGHT

by

Connor Aquila Dack

2017

All Rights Reserved

ACKNOWLEDGEMENTS

I would like to acknowledge the faculty and staff of the Electrical and Computer Engineering Department, especially my advisor Dr. Ross Snider, as well as my family and friends for their continued support and encouragement throughout my undergraduate and graduate education.

Funding Acknowledgment

This work was kindly supported by the Montana Research and Economic Development Initiative, Montana Board of Research and Commercialization Technology, and Resonon, Inc.

TABLE OF CONTENTS

1. INTRODUCTION AND BACKGROUND 1

Levels of Integration 1
Integrated Circuits 2
FPGAs 2
Why Custom FPGA Systems 2
SoC FPGAs 3
SoC FPGA SoMs 3
Project Overview 3
Smart Camera Solutions 4
Previous Work 5
Exploratory Senior Design Team 5
Monica Whitaker's Research 5
Sorting Senior Design Team 6
Current Work 6
Background 7
Hyperspectral Imaging 8
Resonon Optical System 9
Sorting Systems 9
Printed Circuit Board Design 9
Schematic Capture 10
Board Layout, Layers, and Routing 11
Signal Interference 11
Computer Aided Design Files 12
Assembly 12
FPGA Design 13
Asynchronous and Synchronous Designs 13
Three Process State Machine 13
Current State Transition Process 14
Next State Logic Process 14
Output Logic Process 14
Simulation Setup 15

2. CMOS SENSOR FOR THE HYPERSPECTRAL IMAGER 16

Hyperspectral Image Sensor PCBs 16
Hyperspectral Image Sensor Prototype Board 16
FPGA Mezzine Connector Prototype Board 24
Prototype Sensor Housing 29
Hyperspectral Image Sensor Control 30

TABLE OF CONTENTS — CONTINUED

Camera Control State Machine 30
Start States 32
Read States 32
Exposure States 33
End Exposure States 33
Camera Control State Machine Implementation 34
Camera Control State Machine Simulation 34
Camera Control Avalon Wrapper 35
Hyperspectral Image Pixel Preprocessing 35
Pixel Preprocessing Component 36
Implementation 36
Simulation 37
Pixel Averaging 37
Implementation 37
Simulation 38
Pixel Location 39
Implementation 39
Simulation 40
Pixel Parallelization 40
Implementation 40
Prototype Sensor Board Verification 45

3. LINE SCAN OBJECT TRACKING 48

Camera Link Interface 48
Prototype Camera Link Interface Board 48
Line Scan Camera Interface 52
Camera Link Interface 53
Module 1 - Pixel to Data Array Conversion 53
Implementation 53
Simulation 54
Object Tracking Algorithm 55
Module 2 - Tap to Sub-segment Conversion 55
Implementation 56
Simulation 58
Module 3 - Sub-segment to Segment Conversion 59
Implementation 60
Simulation 63
Module 4 - Segment to Object Conversion 63
Implementation 64

TABLE OF CONTENTS — CONTINUED

Simulation 68
Hyperspectral Object Packet Creation 68
Implementation 68
Simulation 69
Module 5 - Vertical Center Calculation 69
Implementation 70
Simulation 71
Air Jet Object Packet and Interfacing with the Air Jets 71
Implementation 72
Simulation 72

4. PROTOTYPE INTEGRATION 73

Arria 10 SoM PCB Design 73
Arria 10 SoM Hyperspectral Image Sensor Daughter Card 73
Algorithm Integration 75
Physical Setup 76
Continuing Development 78

5. CONCLUSION 80

REFERENCES CITED 81

APPENDICES 83

APPENDIX A: Prototype Image Sensor 84
APPENDIX B: Line Scan Camera and Object Tracking 213
APPENDIX C: Arria 10 SoM Image Sensor 316

LIST OF TABLES

Table		Page
2.1	Prototype Image Sensor Stack Up	20
2.2	Prototype FMC Stack Up	27
3.1	Prototype Waterfall Board Stack Up	51
A.1	Number of Clock Cycles for Exposure Register Description	90
A.2	Row Number Exposure Register Description	90
A.3	Command Register Description	90
A.4	Status Register Description	91

LIST OF FIGURES

Figure		Page
1.1	The System Level Block Diagram	4
1.2	The Arria 10 SoM	7
1.3	An example of hyperspectral line scan images over several frames	8
2.1	The MT9M413 Block Diagram	17
2.2	The Unrouted Prototype Sensor Board	19
2.3	The Routed Prototype Sensor Board	21
2.4	Example Extended Trace Length	22
2.5	The Top Side Populated Prototype Sensor Board	23
2.6	The Bottom Side Populated Prototype Sensor Board	23
2.7	The Unrouted Prototype FMC Connection Board	26
2.8	The Routed Prototype FMC Connection Board	27
2.9	The FMC Board connected to the FMC to FMC Board	28
2.10	The 3D Printed Prototype Sensor Housing	30
2.11	The Camera Control State Machine	31
2.12	The Camera Control State Machine Simulation Waveform	34
2.13	The Pixel Averaging Simulation Waveform	38
3.1	The Unrouted Prototype Waterfall Board	50
3.2	The Routed Prototype Waterfall Board	52
3.3	A single object test image	54
3.4	A multiple object test image	58
3.5	Waveforms from the Module 2 Test Bench	59
4.1	The Unrouted Arria 10 SoM Sensor Board	75
4.2	The prototype air jet sorting system	77
4.3	View of what the cameras will be seeing	78

ABSTRACT

Imaging systems can now produce more data than conventional PCs with frame grabbers can process in real-time. Moving real-time custom computation as close as possible to the image sensor will alleviate the bandwidth bottle-neck of moving data multiple times through buffers in conventional PC systems, which are also computation bottlenecks. An example of a high bandwidth, high computation application is the use of hyperspectral imagers for sorting applications. Hyperspectral imagers capture hundreds of colors ranging from the visible spectrum to the infrared. This masters thesis continues the development of the hyperspectral smart camera by integrating the image sensor with a field programmable gate array (FPGA) and by developing an object tracking algorithm for use during the sorting process, with the goal of creating a single compact embedded solution. An FPGA is a hardware programmable integrated circuit that can be reprogrammed depending on the application. The prototype integration involves the development of a custom printed circuit board to connect the data and control lines between the sensor, the FPGA, and the control code to read data from the sensor. The hyperspectral data is processed on the FPGA and is combined with the object edges to make a decision on the quality of the object. The object edges are determined using a line scan camera, which provides data via the Camera Link interface, and a custom object tracking algorithm. The object tracking algorithm determines the horizontal edges and center of the object while also tracking the vertical edges and center of the object. The object information is then passed to the air jet sorting subsystem which ejects bad objects. The solution is to integrate the hyperspectral image sensor, the two processing algorithms, and Camera Link interface into a single, compact unit by implementing the design on the Intel Arria 10 System on Module with custom printed circuit boards.

1

INTRODUCTION AND BACKGROUND

As cameras continue to develop and provide more data, new systems need to be designed to process the data. Currently, the data from the camera is being passed to a Linux computer, through a frame grabber, for processing. The data is transferred using different serial protocols such as USB 3.0 or Camera Link. Between the operating system on the Linux machine and the transfer protocol, the limiting factor is the processing unit. The target of this research project is to remove the need for the cables between the camera and the processing unit by combining the image sensor with a custom processing unit to create a smart camera. One application of the smart camera would be commercial sorting.

Commercial sorting is necessary to determine good product from bad. In Montana, a major crop is the lentil which must be sorted before being sold. By sending the crops to a large, off farm, sorting company, the farmer has less control over the quality of the crops and usually has only one sorting done. The focus of the Hyperspectral Sorting System is to develop a single solution that can be used in a barn on the source farm, where the farmer has control of the sorting and can determine the quality of the product before it leaves the farm. This project, also, creates spin off products such as the high speed hyperspectral imager that will be utilized by Resonon, Inc.

Levels of Integration

Custom embedded processing applications require different solutions to achieve the best result. The solution could use an integrated circuit, such as a microcontroller,

a field programmable gate array (FPGA), an FPGA system on chip (SoC), or an FPGA SoC System on Module (SoM). Each of these solutions allow for different flexibility ranging from cheap, simple solutions to custom, parallel solutions.

Integrated Circuits

Integrated Circuits, such as microcontrollers, are specific processing chips that provide simple solutions for a single application. The design selects a current microprocessor or designs a custom microprocessor to implement the processing. A microprocessor implements a fixed algorithm at a low cost. These devices are for mass producing a single solution. FPGAs, on the other hand, allow for the flexibility of implementing different processing algorithms.

FPGAs

Custom FPGAs are hardware programmable chips that allow for the creation individualized parallel processing solutions. FPGA fabric is programmed using a hardware description language (HDL) such as VHDL or Verilog.

Why Custom FPGA Systems? Custom FPGA Systems provide for the flexibility to change the computation algorithms without recreating the custom PCB or integrated processors. If a customer wants to sort wheat instead of lentils, the system has to be retrained with new sorting coefficients while the algorithm remains the same. If a customer wanted to use a different sorting algorithm, the front-end processing and board connections would remain the same, while the hardware programmable algorithm is changed. Current FPGAs are combined with a dual ARM core processor on the silicon die, creating a System on Chip (SoC) FPGA.

SoC FPGAs

SoC FPGA contain the FPGA programmable fabric as well as a dual ARM Core Central Processing Unit (CPU). The FPGA fabric handles the pixel processing while the ARM CPU provides the user interface and access to necessary registers. The Line Scan System is being prototyped using the Altera Arria 5 SoC Development Board, with the FPGA fabric tracking the objects and the ARM CPU setting control registers and system variables. The Hyperspectral Image System was prototyped on the Arria 10 SoC, with the FPGA fabric handling the computational algorithm and the ARM processor implementing the user interface. The final design of the Hyperspectral Image System will be implement on the Arria 10 SoC System on Module (SoM).

SoC FPGA SoMs

An SoC FPGA SoM is an FPGA with the dual ARM Core CPU implemented on a single printed circuit board with the necessary supplementary components such as the power regulation circuitry, the DDR4 DRAM, and the connectors. The Arria 10 SoM has four FPGA Mezzine Connectors (FMC), two on the top and two on the bottom. The FMC connectors on the bottom will be used to connect different sensors to the FPGA through custom printed circuit board.

Project Overview

The Hyperspectral Smart Camera Sorting System consists of two cameras and a single FPGA processing unit. The high level block diagram of the system can be seen in Figure 1.1. This thesis focuses on the blocks in red, including the custom CMOS image sensor, the custom CMOS interface and pixel preprocessing, as well as the object tracking algorithm.

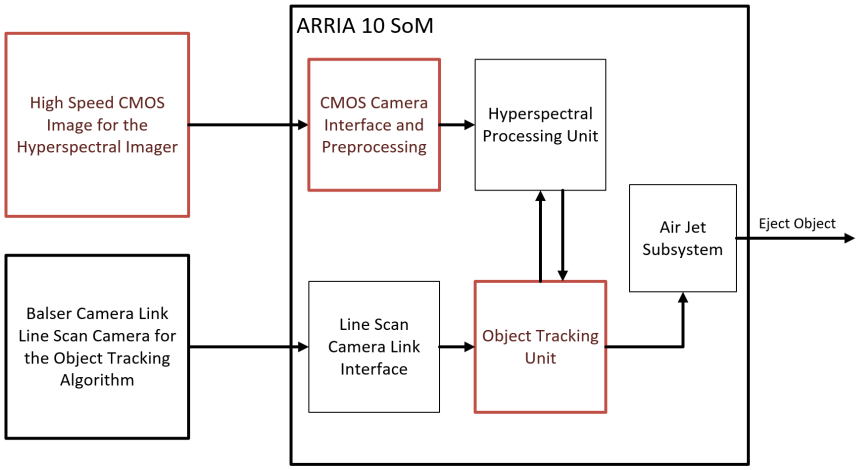


Figure 1.1: The smart camera sorting system block diagram showing the connection of the different subsystems with the red blocks being the focus of this thesis.

On the left, there are two cameras which provide data to the system. The first camera is the custom CMOS image sensor which required custom printed circuit boards and control code. The second camera is the line scan camera which provides the data for the object tracking algorithm. The data from the CMOS image sensor is passed to the pixel preprocessing block before being passed to the hyperspectral processing unit. The line scan data tracks the edges of the objects before passing the information to the hyperspectral processing unit. The hyperspectral processing unit makes a decision about the object before the information is passed to the air jet subsystem.

Smart Camera Solutions

Smart cameras are image sensors combined with internal real time processing. This allows for the system to analyze the data using a video processor or microprocessor [1]. As these systems continue to advance, smart cameras are being designed with FPGA accelerators to help handle the data loads [2]. Current FPGA smart

camera systems connect the video sensor to the FPGA through a frame grabber, which parallelizes the data. A second solution is to connect the FPGA directly to the image sensor [3], which is how the smart camera in the current research at Montana State is designed.

Previous Work

The smart camera system originally started with an Exploratory Senior Design Team, including Monica Whitaker and myself, in 2014. The project was then continued by two graduate students, Monica Whitaker and myself, as well as a second senior design team in 2015. Monica Whitaker continued the work of the Exploratory Senior Design Team. The second senior design team began the implementation of the Air Jet Sorting System.

Exploratory Senior Design Team. The project started with a proof of concept senior design project, where the team implemented the logistic regression classification algorithm using fixed-point calculations on the Arria 5 Field Programmable Gate Array (FPGA) Development Board. The team established that the Arria 5 Development was capable of implementing the logistic regression classification algorithm as was currently implemented on the Linux computer. The design combined a Basler Area Scan Camera, the Arria 5 FPGA, and the on board DDR3 DRAM. The Basler Area Scan Camera provided the data to the Arria 5 FPGA. The DDR3 DRAM provided the light and dark data necessary for the computation algorithm.

Monica Whitaker's Research. Monica Whitaker continued the development of the smart camera by moving the logistic regression algorithm from fixed-point calculations to floating-point calculation by implementing the design on an Arria 10 SoC FPGA [4]. During the transition to floating-point operations, a design decision

was made to rotate the custom sensor 90 degrees to simplify the computation process. The rotation was necessary for the design of the hyperspectral image processing block that computes the probability of each spectral bin of one physical pixel before moving on to the next pixel [4]. The logistic regression algorithm was verified using simulated data with MATLAB. The object decision algorithm was also implemented on the Arria 10. The object decision algorithm combined the hyperspectral classification data along with the object edges data. The object edges were passed from the Arria 5 to the Arri 10 and back using the serial lite transfer protocol.

Sorting Senior Design Team. The Sorting Senior Design Team began the implementation of the object tracking algorithm. The team designed the initial algorithm to track falling objects using a Basler Line Scan Camera and the Arria 5 Development Board. The team implemented the design to track two falling objects at a time, however there could potentially be up to 64 falling objects. The algorithm was verified in simulation.

Current Work

This thesis focuses on the development of a smart camera built on a system on module FPGA. This Arria 10 SoM is ideal for image processing due to its processing capabilities, small form factor, and connectors. The Arria 10 SoM is able to achieve 1.5 trillion single precision floating point operations per second (FLOPS). The Arria 10 SoM is 3.4 x 3.8 inches which allows the computation to be placed with in the camera itself. As shown in Figure 1.2, the module also has FPGA Mezzanine Connectors on the top and bottom of the board, allowing the CMOS image sensor to be connected directly to the FPGA removing the need for cables.

The first chapter of this thesis describes the custom CMOS Image Sensor for the high speed hyperspectral imager including the design and build of the

Figure 1.2: The Arria 10 SoM that will be used in the development of the smart camera system [5].

Hyperspectral Image Sensor Prototype Boards, the camera control VHDL code, and the pixel preprocessing. The second chapter focuses on the development of an object tracking algorithm using the data from a monochrome Camera Link line scan camera. The third chapter discusses the integration of the Hyperspectral System with the Object Tracking System, to provide a single commercial sorting embedded processing solution.

Background

The following section discusses the necessary background information to understand the project, including the definition hyperspectral imaging, current sorting systems, printed circuit board design, and FPGA design.

Hyperspectral Imaging

Traditional cameras capture images in three color bands, red, blue, and green. These colors are then blended to create the colors that humans see. In contrast, hyperspectral imagers are able to capture a wider band of colors through the use of a diffraction method to spread the spectrum across the image sensor pixels. Figure 1.3 shows a three dimensional data cube defining the pixels. The spatial pixels are represented in the x and y dimensions while the spectral bins, the color data defining each pixel, are represented in the z dimension.

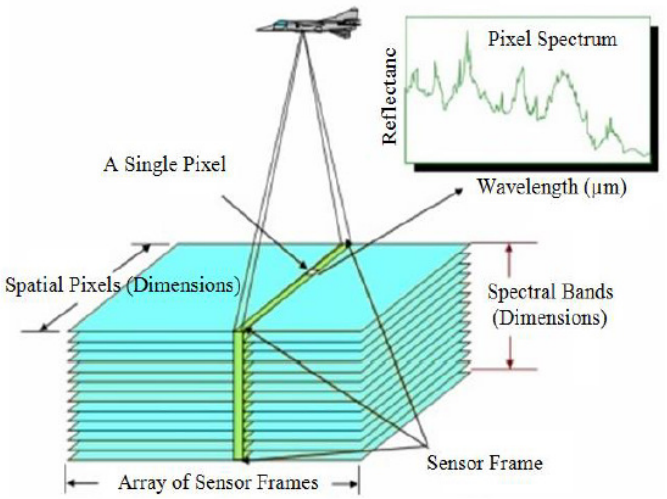


Figure 1.3: An example of hyperspectral line scan images over several frames (lines) [6].

A hyperspectral camera is built on a monochrome area scan camera. The monochrome area scan camera captures a single-color value per pixel. When combined with the hyperspectral optical system, the light of a single pixel is spread across the row values of the sensor. The hyperspectral lens creates a line scan camera where the horizontal pixels are the spatial pixels and the vertical pixels become the spectral bins. To create the data cube in Figure 1.3, each line of spatial data is saved and combine with the previous lines [7].

The hyperspectral processing algorithm requires the luminance of the pixels to be normalized. This normalization requires the white and dark correction values. The white data matrix is created by imaging a white background. The dark matrix is created by capturing an image with the lens cap on. The correction matrixes are then stored on the FPGA DRAM to be used during the hyperspectral computation block [4].

Resonon Optical System. Resonon designs and develops the front end optical systems that creates the hyperspectral imagers. Traditionally, Resonon buys off the shelf monochrome cameras and fits their optical systems to them. During this project, Resonon will be provided with a custom high speed imager to incorporate with their optical system, creating a unique, one of a kind high speed hyperspectral image sensor.

Sorting Systems

Resonon‘s current hyperspectral imagers are used in robotic sorting systems. These systems use the data from the hyperspectral imagers to determine good and bad products. Product is run on a conveyer belt before the bad products are then removed with a robotic vacuum arm. These sorting systems are expensive and require room to run the conveyer belt. The new system will implement a free fall sorting system that uses high speed valves to eject product with air jets. The new system will require custom printed circuit boards to implement the custom hardware.

Printed Circuit Board Design

Printed circuit boards are used to implement custom circuits. In this project, the printed circuit boards are used to connect the custom image sensor and the line scan camera to the FPGA or to provide input/output interfaces to the FPGA.

Printed circuit board (PCB) design is the process of connecting an electrical circuit through copper traces on individual layers which are separated by insulated cores. Each printed circuit board is created in even layers starting with the top and bottom layers. If more layers are necessary to complete the routing or to provide dedicated power or ground planes, an even number of layers is added between the top and bottom layer. In a simple four layer board, the top layer is used for routing and component placement. The inner layers are used as either power planes, routing layers, or a combination of both. The bottom layer is used as a routing and component layer. As the number of layers increases, the cost of the board increases. The goal for the PCB design is to limit the number of layers, while reducing interference on the signal traces. PCB design begins with the schematic capture, followed by component placement, routing, creation of computer aided design files, and ending with assembly.

Schematic Capture. The first step in the PCB design process is to design the circuit that will be implemented on the PCB. The schematic is created in schematic capture software, such as Mentor Graphics DxDesigner. First, the major components of the design are selected and any new components are created as a schematic symbol and connected to a PCB footprint. The schematic symbol defines the circuit and allows for the connection of nets between pads. The PCB footprint is the physical layout of the pins that connect the component leads to the copper traces through landing pads. Second, the data and control signals are connected through nets. Third, components are connected to the various power rails, using decoupling capacitors where recommended. Finally, power regulation circuitry, including the input voltage range and the necessary output voltage rails, is designed, and added to the circuit. With the schematic drawn, the second step in the PCB design process is to layout the components on the top and bottom layers of the board.

Board Layout, Layers, and Routing. To begin the second step of the PCB design process, the schematic is moved from DxDesigner to Mentor Graphics PADs Layout, the PCB design software. This imports all the footprints and the connections from the schematic. The components are laid out starting with either the major components or with components that have a specific location. With the components in place, the third step in the PCB design process is to determine the number of layers and the layer type. This information is recorded in the layer stack up table.

With the boards layers determined, the fourth step is to route the connections between the components. This involves drawing the copper lines, adding vias, and drawing the copper planes to connect power and ground. The major concern during this step to reduce trace lengths to reduces signal interference.

Signal Interference. Copper traces transmit electrical signals between pads. The signals transmit data and control information. Trace length, reflections, cross talk, or electro-magnetic interference can affect the signals. Trace length is the length of copper that the signal must travel from one end to the other. As the signals move faster, shorter traces are required to reduce transmission line effects. Besides the transmission line effects, the trace lengths must be matched so that parallel data arrives at the destination at the same time. A reflection is created when an electrical signal travels around a corner or through a via. The reflection causes voltage to return to the source which could damage components. Cross talk is caused when magnetic waves induce an electrical current on a second trace. Cross talk could cause bits to change resulting in incorrect data or control signals. Electro-magnetic interference may cause bits to change and is reduced by following common practices, such as using decoupling capacitors on power rails and having return planes close to the signal traces.

Computer Aided Design Files. With the signals routed and power connected to the components, the fifth step in the PCB design process is to create the Computer Aided Design (CAD) Files. The CAD files include the layer files, the drill file, and the drill drawing. The layer files include the copper layers, the silkscreen layers, and the soldermask layers. The copper layers are the top, inner routing and plane layers, and the bottom layers. These layers define the copper traces and pours. The silkscreen layers define the printed text on the board. The soldermask layers define where soldermask is added to the board. The drill file defines the size and location of the drill holes on the board. The drill drawing is a PDF document that contains information about the drill holes, the layer stack up, and any specific information for the Fabrication House. With the CAD files created, they are sent to a Fabrication House to build the PCBs.

Assembly. With the PCB designed and created, the final step is to solder the components to the board. Soldering by hand is either done using a reflow oven or using a soldering iron. The reflow oven is used for the surface mount parts. The first step is to put a small amount of solder paste on the component pads. With the solder paste down, the next step is to place the component on the pads. The final step is to place the board with the components into the reflow oven. The reflow oven, first bakes out the flux in the solder paste before melting the solder to create the electrical connection. The soldering iron is used for through hole components. The pin and pad are heated until the solder melts to the pin and pad, creating the electrical connection. If it is not possible to assemble the board by hand, due to component packaging, the board and components are sent to any assembly house. The assembly house uses a pick and place machine to place the components. The finished PCBs are integrated with control code.

FPGA Design

FPGA design is the implementation of the control code and the processing algorithms to complete the desired tasks using a hardware description language (HDL). The control code is implement using state machines. The state machines provide the control signals to the image sensor and the line scan camera. The state machines are implemented using the three-process method. All of the VHDL code is then verified in simulation using MATLAB and the HDL Verification Toolbox.

Asynchronous and Synchronous Designs

FPGAs systems can be implemented using asynchronous or synchronous designs. Asynchronous designs do not use clocks to control the data flow. Instead the results are updated when the inputs change using combinational logic. As the combinational logic increases in complexity, the maximum frequency of the system is reduced because of the longer path the data take through the system.

Synchronous designs use clocks and processes to control the data flow from one register to the next. These systems usually implement pipelines to compute the results. A pipeline computes data using a series of steps with the previous step feeding the next step. With the pipeline registering intermediate results, the maximum clock frequency of the system increases. However, with a pipeline it takes a certain number of clock cycles to fill data before the first result is ready. This is the latency of the system in terms of clock cycles.

Three Process State Machine

The Three Process State Machine is a good coding practice to create a state machine. The method allows for easy implementation, understanding, and debugging.

The first process is the current state transition process. The second process is the next state logic process. The third process is the output logic process.

The Three Process State Machine was implemented by defining a custom signal type, state_type to define the different states. Every state is named and defined in this type. Two signals, the current state and the next state, are defined as state_type signals. This allows the current state and the next state to be one of the named states.

Current State Transition Process. The current state transition process updates the current state signal from the next state signal. The current state transition process is triggered on the system reset or a clock signal. If the reset is triggered, the current state is updated to the start state. On the rising edge of the clock, the current state is assigned the next state.

Next State Logic Process. The next state logic process defines the next state based on the current state and any input signals. The process is triggered on the change of the current state or on the change of any input signal. The current state is checked using a case statement. The input signals are checked using if statements within the different states.

Output Logic Process. The output logic process defines the output signal of each state. The process is triggered on the change of the current state, which is checked using a case statement. The three process state machine was verified using simulation.

Simulation Setup

The VHDL components were simulated using the MATLAB HDL Verification Toolbox which drives the input signals in ModelSim. The VHDL component input signals are driven using a MATLAB script. The MATLAB script uses the fixed-point toolbox to create the input values. The output signals are saved during the simulation for comparison with MATLAB fixed-point simulation. The simulation wave forms are used to verify internal signals, such as current state and next state signals.

CMOS SENSOR FOR THE HYPERSPECTRAL IMAGER

Currently, Resonon purchases off the shelf monochrome area scan cameras to combine with their optical system to create hyperspectral imagers. These cameras connect to the computer via USB 3.0 where the classification occurs. The USB 3.0 protocol limits the frame rate or the spatial resolution of the camera. Instead, a commercial monochrome area scan sensor was purchased to replace the commercial area scan camera. The sensor selected is the Aptina MT9M413 1.3 Megapixel [8]. The sensor was selected because of the physical pixel size and the frame rate of the camera. The pixels are 12um square, which allows Resonon to expose more light to each pixel. The sensor produces 500 frames per second with 1020 spatial pixels, which doubles the frame rate of Resonons current hyperspectral imagers.

The image sensor is connected to the Arria 10 FPGA through two custom printed circuit boards (PCB), the Hyperspectral Image Sensor Prototype PCB and the FMC Prototype PCB. The image sensor is controlled using VHDL code. The data from the sensor goes through a preprocessing step before being passed to the Hyperspectral Computation Unit, which classifies each pixel before making a decision about the object.

Hyperspectral Image Sensor PCBs

The following section discusses the design and creation of the Hyperspectral Prototype Sensor Board and FMC Connection Board.

Hyperspectral Image Sensor Prototype Board

The Hyperpsectral Image Sensor Prototype PCB is a board that connects the Aptina MT9M413 1.3 Megapixel monochrome image sensor to two 100-pin headers.

The two 100-pin headers are then connected to the FPGA Mezzanine Connector Board which connects the data and control lines to the FPGA. A block diagram of the sensor can be seen in Figure 2.1. The sensor produces 500 images per second that are 1280 pixels wide by 1024 pixels tall. The image data generates ten ten-bit pixels in parallel moving across each row before moving down to the next row. The data is passed over 100 single ended transmission lines while control signals are transmitted and received over 23 single ended lines.

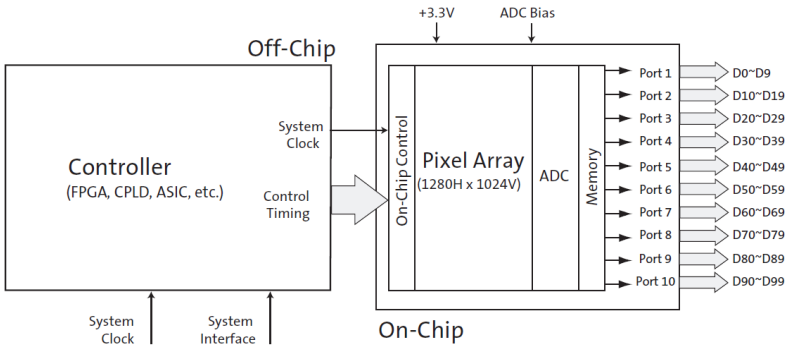


Figure 2.1: The MT9M413 Block Diagram showing a control unit, an Arria 10 SoC FPGA, connected to the MT9M413 image sensor via a custom printed circuit board [8].

With 100 single ended data lines and 23 control lines, the first step to create the schematic is to route the data lines from the sensor to the headers which connect to the FPGA Mezzine Connector Baord (FMC Board). The goal for routing was to reduce cross talk on the ribbon cables that connect the Sensor Board to the FMC Board by having ground signals between every data line and control lines. This solution would require approximately 250 pins. Instead of using three 100-pin headers and having unused connections, two 100-pin headers were implemented. When assigning the signals from the sensor to the connector, correlated cross talk was reduced by having uncorrelated signals routed close together. For example, tap 0 bit 0 and tap 0 bit 1

were not routed to two neighboring header pins. Instead, tap 0 bit 0 and tap 9 bit 9 were routed to neighboring header pins. The signals from the sensor were connected by hand to the header pins before being drawn in Mentor Graphics DxDesigner. This approach did not account for traces having to travel across the PCB, which resulted in longer matching trace lengths. With the data and control signals connected, the next step was to implement the power circuitry for the sensor.

The sensor requires an analog +3.3V power rail and a digital +3.3V power rail along with analog ground and digital ground. The FMC Board provides the analog and digital power rails though two two-pin headers reducing the amount of circuitry on the sensor board. The analog voltage provides power for the analog processing circuitry including the column buffers and the analog to digital converters [8]. The analog voltage is also connected to a set of reference voltage pins through potentiometers. These reference pins include VCLAMP3, VLN1, VLP, VREF1, VREF2, VRSTPIX, VREF3, and VREF4. VCLAMP3 is the dark offset cancellation negative input reference voltage and it is set to 0.0 volts [8]. VLN1 is the bias setting for the pixel source bias operating current and it is set to 1.0 volts [8]. VLP is the bias setting for the bias voltage for the column source follower operating current and it is set to 0.0 volts [8]. VREF1 is the reference voltage for the analog to digital converters and it is set to 1.0 volts. The voltage of VREF1 sets the maximum voltage of the input signal from the pixel array [8]. VREF2 is the reference voltage used for the calibration of the analog to digital converts. VREF2 is set to 0.8 volts [8]. VRSTPIX is the power supply for the pixel array and it is set to 2.7 volts using the potentiometer [8]. VREF3 is the dark offset cancellation positive input reference and is set to 0.6 volts [8]. VREF4 is the analog to digital input value which is one quarter of VREF1, 0.25 volts [8]. VLN1, VLP, and VREF4 are generated on chip but may be change by supplying external biasing voltages. Potentiometers are used to fine tune

the voltage based off of the input from the analog power supply that is +3.3V plus or minus 0.3V. All power pins, both analog and digital, are decoupled with capacitors ranging from 0.1uF to 100uF. The digital +3.3V provides power for the digital pad ring [8].

With the data and control signals connected and the power for the reference voltages and power rails connected, it is time to create the layout for the printed circuit board. The first step is to place the components on the board. The image sensor was placed in the center of the board with the decoupling capacitors placed as close as possible to the power pins. The decoupling capacitors used are 0603 parts, which were soldered by hand. The data and control headers were placed on the sides of the board while the power connectors were placed at the top of the board. The potentiometers for the analog reference voltages were placed at the bottom of the board. Figure 2.2 shows the layout of the sensor board with all of the nets removed for easy visibility.

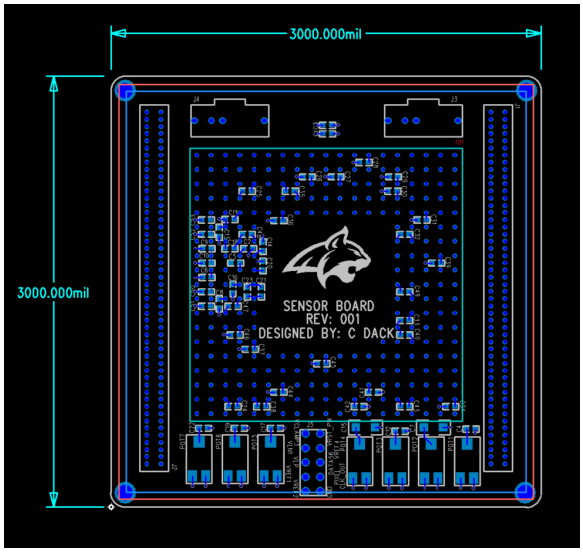


Figure 2.2: The unrouted sensor board showing the placement of the sensor, the decoupling capacitors, and the headers. Note: the image sensor is on the bottom of the board.

With the parts placed, the layer stack up was determined. For the Prototype Sensor Board, 10 layers was necessary to complete the routing of the board. The top layer is a component and routing layer. The second layer is the analog power plane. The third and fourth layers are internal routing layers. The fifth layer is the ground plane. The sixth and seventh layers are internal routing layers. The eighth layer is the digital power layer. The ninth layer is an internal routing layer. The bottom layer is a component and routing layer. The board is 62mils thick and Table 2.1 shows the thickness and type of each layer. All the layers contain 1.0 ounces of copper.

Table 2.1: Prototype Image Sensor Stack Up

Layer number	Layer Name	Copper Weight	Copper Thickness	Prepreg and Core Material	Prepreg and Pressed Core Thickness	Trace Width
				Soldermask - Green		
1	Top	1.0oz	0.00135			0.005
				FR-4 (Er = 4.5)	0.0056	
2	AVDD	1.0oz	0.00135			
				FR-4 (Er = 4.5)	0.0050	
3	Layer 3	1.0oz	0.00135			0.005
				FR-4 (Er = 4.5)	0.0052	
4	Layer 4	1.0oz	0.00135			0.005
				FR-4 (Er = 4.5)	0.0050	
5	GND	1.0oz	0.00135			
				FR-4 (Er = 4.5)	0.0078	
6	Layer 6	1.0oz	0.00135			0.005
				FR-4 (Er = 4.5)	0.0050	
7	Layer 7	1.0oz	0.00135			0.005
				FR-4 (Er = 4.5)	0.0052	
8	DVDD	1.0oz	0.00135			
				FR-4 (Er = 4.5)	0.0050	
9	Layer 9	1.0oz	0.00135			0.005
				FR-4 (Er = 4.5)	0.0056	
10	Bottom	1.0oz	0.00135			0.005
				Soldermask - Green		

With the components in place and layers defined, routing is the next step starting with the power planes. All the through hole pins that are power or ground are connect to their respective planes when the copper pour is created. With the planes drawn, the next step is to route the decoupling capacitors to the correct power pins. With the component placement, short traces connect the capacitor's surface mount pads to the sensor's power or ground pins on the top layer. The third step is to route the data and control signals. The 100 data signals require matching trace lengths, so that the bits arrive at the headers at the same time. This is necessary to ensure the data

arriving at the FPGA belongs with the current pixel not with the previous or the next pixel. The control signals must have matching trace lengths so that the control data arriving at the image sensor is correct. Because of the constraints on the trace lengths and the number of signals to route, auto router was used to route the signals. The results can be seen in Figure 2.3.

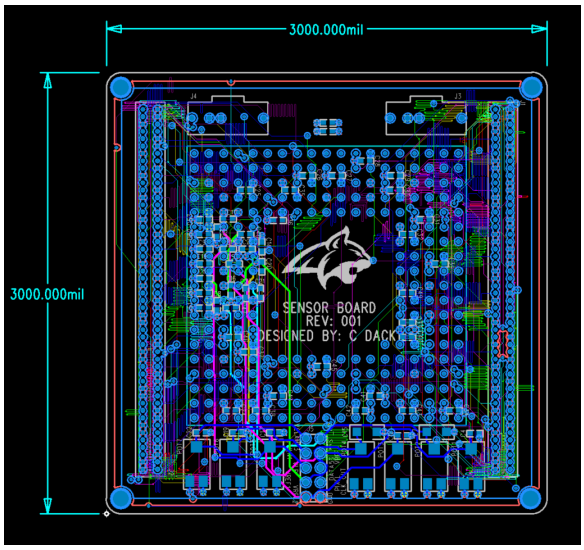


Figure 2.3: The routed sensor board showing the traces on the different layers. The board is a 10 layer board with data and control traces running 2.8965 and 2.99818 inches, respectively.

The auto routing tool routed the longest traces first and then added extra length to the shorter traces. The addition length was added by weaving the trace back and forth to achieve the desired trace length. An example can be seen in Figure 2.4, where the trace has been double back on itself to extend the trace length so that it matches the rest of the data lines. All of the data traces are 2.8965 inches in length and all of the control traces are 2.99818 inches in length.

With the routing finished, the design files were sent to the fabrication house for production. Two printed circuit boards were manufactured and sent back. One board was then populated by hand using the reflow oven and a soldering iron. First,

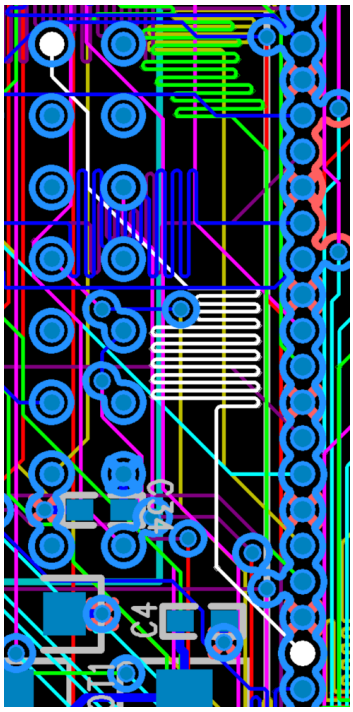


Figure 2.4: The trace length has been extended by routing the trace so that it doubles back on itself.

the surface mount parts, capacitors, resistors, and potentiometers, were solder to the board using solder paste and the reflow over. Second, the image sensor and the pin headers were solder to the board using a solder iron and a tin-lead solder. The difficultly, due to pin spacing, was heating up the pin and the pad to allow the solder to create a good electrical connection.

Figure 2.5 shows the top side of the sensor board with the surface mount parts and the headers for the ribbon cables and for power cables. Figure 2.6 shows the bottom side of the board with the image sensor and the through hole pins.

With the prototype sensor board assembled and it is ready for testing with the FPGA FMC Board and the Sensor Control Code.

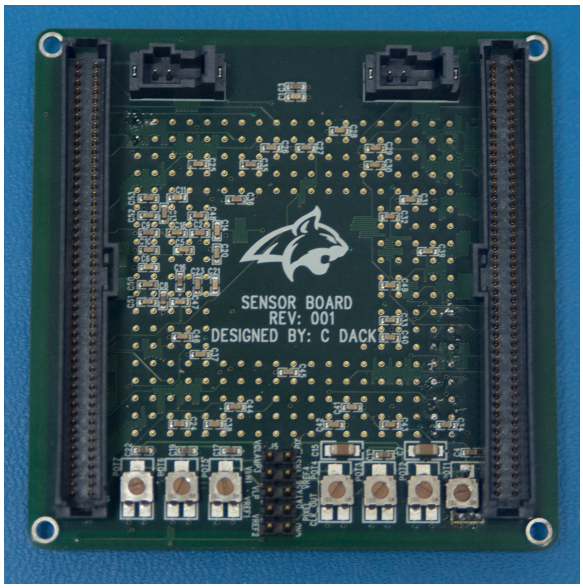


Figure 2.5: The populated sensor board with all of the components soldered using the reflow over or by hand.

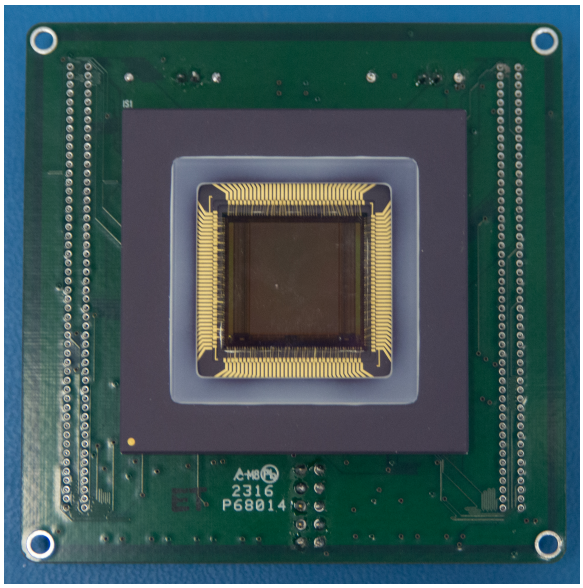


Figure 2.6: The populated sensor board showing the image sensor.

FPGA Mezzine Connector Prototype Board

With the sensor board designed, the FPGA Mezzine Connector Prototype Board (FMC Prototype Board) is designed next. The FMC Prototype Board which will connect the image sensor prototype board to the FPGA through the FPGA Mezzine Connector (FMC). The ribbon cable from the image sensor board connects to the 100-pin headers on the FMC Prototype Board. The data signals from the sensor are at a +3.3V and they need to be stepped down to +1.8V to be read by the FPGA. The control signals from the FPGA are at +1.8V and need to stepped up to +3.3V to read by the sensor. With these requirements, the first step is to create the schematic for the FMC Prototype Board.

The schematic was designed to connect the data and controls signal to the FPGA through the FMC. With the ground signals predefined by the FMC specification [9], the data and controls signal were assigned to the remaining pins. Most the signal pins are connected in pairs because of the layout of the FMC pins. The goal for theses connection pairs was to reduce the cross talk between correlated pixels, while no consideration was taken to reduce the trace lengths. With the FMC pin assignments determined, the connections on the level translators were assigned by going down each of the 100 pin headers, with the goal to keep the signals routed to the same chip. The issue was that there were data signals interlaced with control signals which are translated differently.

With the signal assigned to the pins, the level translators must be powered with 1.8V and digital 3.3V rails. The 1.8V rail was supplied by a bench top power supply through a two-pin header. The 3.3V rail was also supplied by the bench top power supply and connected via a two-pin header. The 3.3V rail was used to supply the digital and analog power to the image sensor. The 3.3V digital rail was run from the 3.3V power plane through a two-pin header and over to the image sensor board. The

analog 3.3V rail was created by filtering the 3.3V digital rail through a low pass filter created using decoupling capacitors, a ferrite bead, and a resistor.

The FMC Board also required SMA connectors to communication between the Arria V and the Arria 10 Development Boards. Initially, the Arria V was going to handle the monochrome line scan camera, the logistic regression algorithm, and the air jet control system, while the Arria 10 was going to handle the hyperspectral imager and the object classification algorithm. The Arria 10 was going to make the decision on whether to keep or reject an object, while the Arria V was going to handle the ejection of the bad object. The SMA connectors connect the boards transceivers together to allow for high speed serial communication. The communication system was designed during the previous stage of the project [4]. Along with the SMA connectors, a clock generator was placed on the board to provide the correct clock frequency for the transceiver transfer protocol.

With the schematic complete, the components are position on the board. First, the FMC connector was placed on the bottom side of the board. This is the connection to FPGA and must be orientated correctly to mate with the Arria 10 Engineering Sample Board. Next, the 100 pin header connectors where placed on the edge of the PCB, which allows for the most room for the remaining components. Third, the level translators were placed on the board with the goal to reduce the length of the traces. The final step is to place the remaining components, including the SMA connectors, the clock circuitry, the power filtering, and the power connectors. The location of the FMC connector along with the components can be seen in Figure 2.7.

With the components laid out on the board, the next step is to determine the board layers. For the FMC board, a ten-layer board was created. The first, or top, layer is a signal and component layer. This layer contains the 100 pin headers, the level translators, SMA and associated clock circuitry, and the power connectors. The

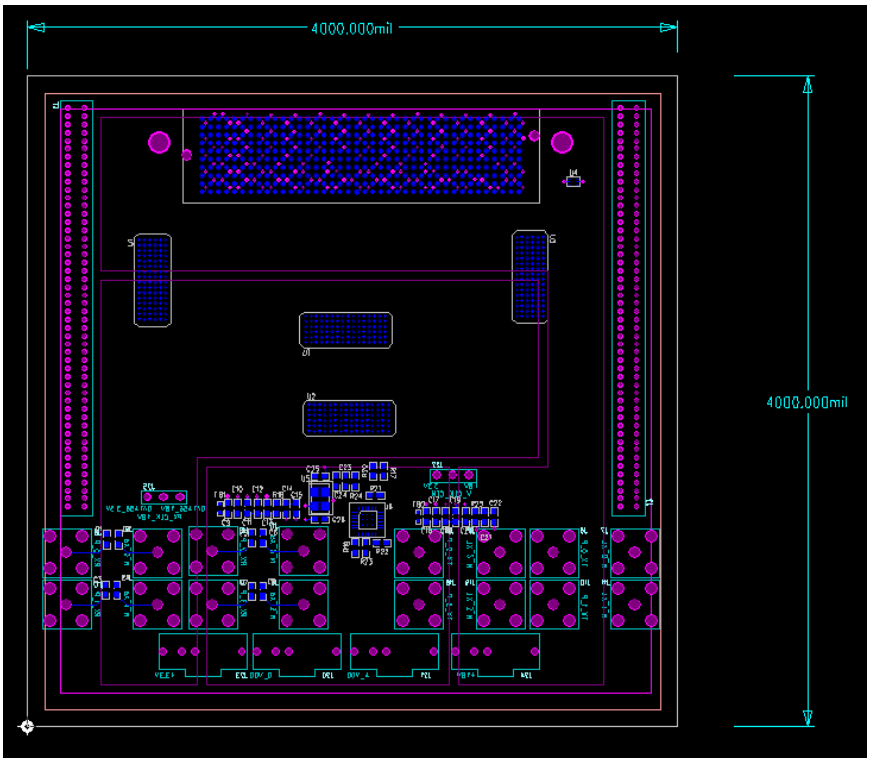


Figure 2.7: The unrouted FMC Connection Board, showing the location of the header pins, the level translators, and the power filtering circuitry.

second layer is the +3.3V analog power plane. The third and fourth layers are signal internal routing layers. The fifth layer is the ground plane. The sixth and seventh layers are internal signal routing layers. The eighth layer is the digital 3.3V power plane. The ninth layer is an internal signal routing layer. The ten, or bottom, layer is a component and routing layer. The bottom layer contains the FMC connector. The layers stack up table can be found in Table 2.2.

With the layers defined, the next step is to use the auto-routing tool to route the board. This is necessary to keep all of the data and control traces the same length. The SMAs, clock circuitry, and power filtering was routed by hand. The routed FMC Connection Board can be seen in Figure 2.8.

Table 2.2: Prototype FMC Stack Up

Layer number	Layer Name	Copper Weight	Copper Thickness	Prepreg and Core Material	Prepreg and Pressed Core Thickness	Trace Width
				Soldermask - Green		
1	Top	1.0oz	0.00135			0.005
				FR-4 (Er = 4.5)	0.0056	
2	AVDD	1.0oz	0.00135			
				FR-4 (Er = 4.5)	0.0050	
3	Layer 3	1.0oz	0.00135			0.005
				FR-4 (Er = 4.5)	0.0052	
4	Layer 4	1.0oz	0.00135			0.005
				FR-4 (Er = 4.5)	0.0050	
5	GND	1.0oz	0.00135			
				FR-4 (Er = 4.5)	0.0078	
6	Layer 6	1.0oz	0.00135			0.005
				FR-4 (Er = 4.5)	0.0050	
7	Layer 7	1.0oz	0.00135			0.005
				FR-4 (Er = 4.5)	0.0052	
8	DVDD	1.0oz	0.00135			
				FR-4 (Er = 4.5)	0.0050	
9	Layer 9	1.0oz	0.00135			0.005
				FR-4 (Er = 4.5)	0.0056	
10	Bottom	1.0oz	0.00135			0.005
				Soldermask - Green		

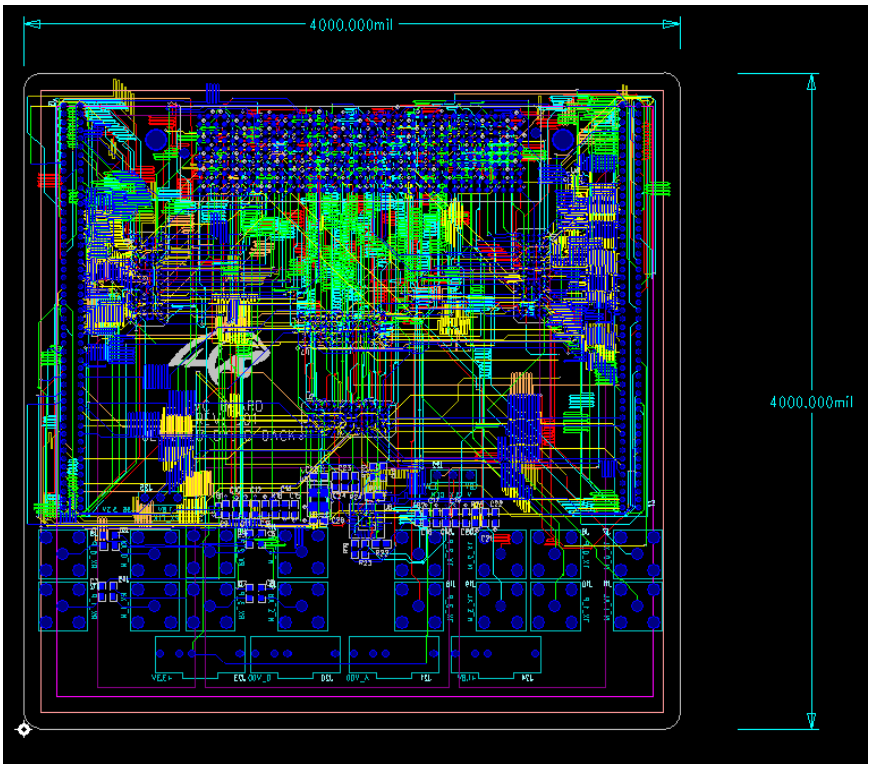


Figure 2.8: The routed FMC Connection Board, showing the matching length traces and the power planes.

With the board routed, the CAD files were sent to a fabrication house before being assemble by an assembly house. The assembly house was necessary because the 32-bit level shifters are packaged in ball grid arrays. These chips were placed on the board using a pick and place machine. The completed and assembled board can be seen in Figure 2.9. With the FMC Board assembled, the FMC connector was rotated

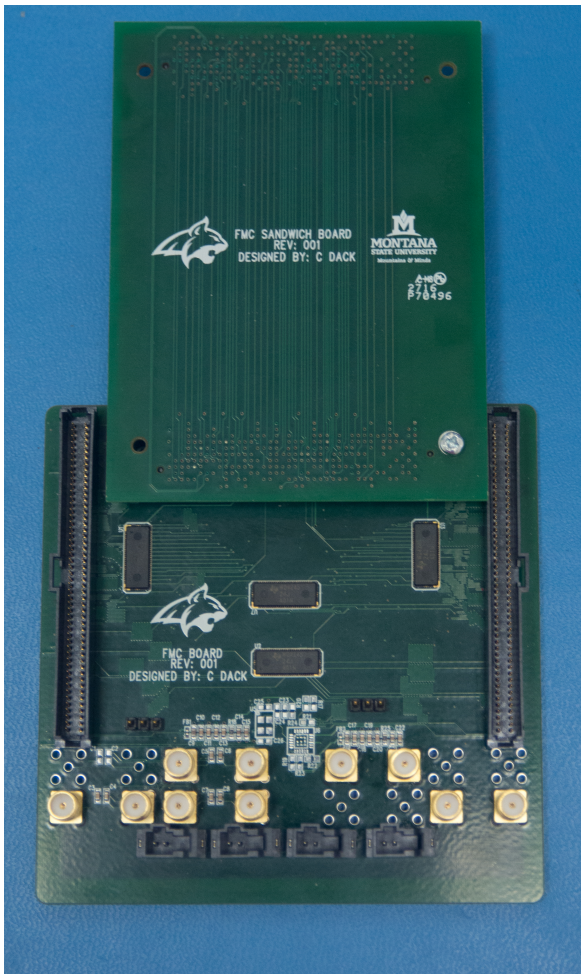


Figure 2.9: The FMC Board connected to the FMC to FMC Board.

180 degrees and could not be correctly connected to the Arria 10 Engineering Sample Board. Because of this error, an FMC to FMC translation board was created. The board has a high pin count FMC female connector and a high pin count FMC male

connector. The female connector connects to the FMC Board and male connector connects to the Arria 10 Engineering Sample Board. The orientation of the male connector was verified using paper modeling.

With the FMC Board connected to the Arria 10 Engineering Sample Board, using the FMC to FMC connection board, a prototype sensor housing was designed to hold the sensor and the lens in the correct positions.

Prototype Sensor Housing

With the image sensor solder to a custom PCB and connected to the FPGA through the FMC Connection Prototype Board, the next step is to design a Prototype Sensor Housing. The housing design was broken into a series of pieces and designed in Solidworks.

The housing was designed in four parts: the C-Mount holder, the front mount, the back mount, and the bottom holder. The lens used in the prototype is a standard C-Mount lens. The C-Mount lens requires a threaded mounting hole. The threaded hole has 1.0 inch thread and 32 threads per inch [10]. The inside edge of the mounting hole was set to 17.526 millimeters from the sensor [10] creating the optical image on the sensor. The lens mount will be replaced with the Resonon Hyperspectral Optical System in the next phase of the project. The front sensor mount connects the lens mount to the front of the image sensor prototype board. The mount is 0.125 inches thick, the thickness of the image sensor. The back mount sandwiches the Prototype Sensor Board between the front mount and itself. The back mount has cut outs for the 100 pin headers and the power connectors. The bottom mount holds the lens mount, front mount, Prototype Sensor Board, the back mount vertically, and supports for the FMC Board to prevent bending and damage.

The housing was printed using a conventional 3D printer, extruding the filament. The assembled Prototype Housing can be seen in Figure 2.10. The vertical section of the housing was designed to be held together using screws and the gaps were sealed using black electrical tape. The entire vertical section slides into a grove printed into the bottom mounting piece.

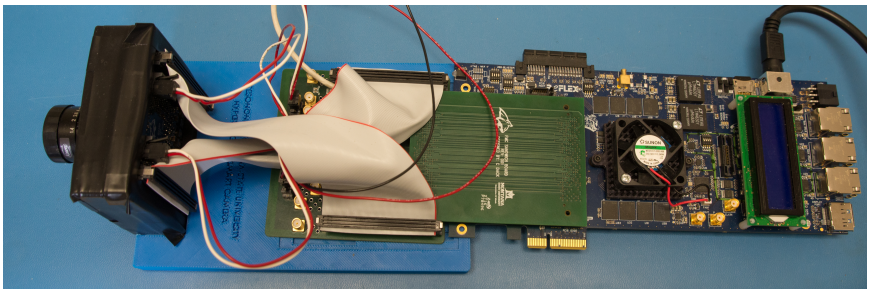


Figure 2.10: The 3D printed Prototype Sensor Housing using a conventional 3D Printer.

With the prototype boards assembled, fit into the housing and connected, the next step is to implement the control code.

Hyperspectral Image Sensor Control

The following section describes the Hyperspectral Image Sensor Control Logic using the Camera Control State Machine which sends and monitors the control signals.

Camera Control State Machine

The Camera Control State Machine provides the output signals to control the image sensor. The state machine diagram can be seen in Figure 2.11 which shows each state, the transitions between states, and the output of the states. The state machine is broken down into four sections: The Start States, The Read States, The Exposure States, and The End Exposure States. The Start States are the states that occur

Figure 2.11: The Camera Control State Machine, showing the follow through the states and the output signals of each state.

during initialization or reset. The Read States occur during every row. The Exposure states occur when the row address is equal to the exposure row address value. The End Exposure States occur at the end of the frame, when the row address is equal to 1023. The state machine controls the PG_n, TX_n, and LD_Shift_n signals. PG_n resets the photodectors which begins the integration [8]. TX_n ends the integration by transferring the charge from the photodectors to memory [8]. LD_Shift_n moves the data from the analog to digital converts to the output registers [8].

Start States. There are five Start States. The first state, S0_Start, is the initial state that occurs on power up. The state sets the calibration start signal to '0'and the sensor calibration interrupt signal to '0'. The system stays in this state for two clock cycles before moving to the next state, S1_Start. During S1_Start, the calibration start signal is set to '1'. The state machine then moves to S2_Start once the calibration is complete. Calibration is complete when the calibration done signal is set to '0'. In state S2_Start, the PG signal is set to '0'and the sensor calibration interrupt is set to '1'. After 64 clock cycles, the PG signal is set to '1'during state S3_Start. The system waits 'M'clock cycles to complete the exposure before moving to S4_Start. S4_Start sets the TX signal to '0'. After 64 clock cycles, the system moves to S5_Start. During S5_Start, the TX signal is set to '1'. This completes the Start Section of the Camera Control State Machine.

Read States. The system moves from S5_Start to S0_Read after one clock cycle. During S0_Read, the system sets the row address to the next row. If this is the first time entering the state, the row address is set to 0. The data read enable signal is set to '0'and the LD Shift signal is set to '0'. If the user has set the Sensor Calibration Register, the system moves to S0_Start. Otherwise the system moves to S1_Read after two clock cycles. S1_Read sets the row start signal to '0'. The next state is S2_Read.

S2_Read sets the data valid signal to '1'. The system will move to S3_Read on the next clock cycle, unless the row address is equal to the exposure row address or is equal to 1023. In S3_Read, the row start signal is set to '1'. The system moves to S4_Read after 126 clock cycles. S4_Read sets the PG signal to '0'. The next state is S5_Read. In S5_Read, the PG signal is set to '1', the data read enable signal is set to '1', the data valid signal is set to '0'and the LD Shift signal is set to '0'. The system the moves back to S0_Read on the next clock cycle.

Exposure States. If the row address is equal to the exposure address, the system moves from S2_Read to S0_Exposure. The Exposure States send the commands to begin the next exposure while also reading the data from the current row. S0_Exposure sets the row start signal to '1'. After 62 clock cycles, the system moves to S1_Exposure and sets the PG signal to '0'. The system then waits 64 clock cycles before moving to S2_Exposure and setting the PG signal to '1'. On the next clock cycle, the system moves to S3_Exposure. S3_Exposure sets the data read enable signal to '1', the data valid signal to '0', and the LD Shift signal to '1'. The system then returns to S0_Read.

End Exposure States. If the row address is equal to the 1023, the system moves from S2_Read to S0_End_Exposure. In S0_End_Exposure, the row start signal is set to '1'. The system then waits 62 clock cycles before moving to S1_End_Exposure and setting the TX signal to '0'. After waiting 64 clock cycles, the system moves to S2_End_Exposure. In S2_End_Exposure, the TX signal is set to '1'and the PG signal is set to '0'. The system then moves to S3_End_Exposure. S3_End_Exposure sets the PG signal to '1', the data read enable signal to '1', the data valid signal to '0'and LD Shift signal to '1'. The system then returns to S0_Read on the next clock cycle.

Camera Control State Machine Implementation

The current state and next state signals were defined as state_type_camera_control. The current state is initialized to a power start state and resets to this state. The current state is update on reset or on the rising edge of the pixel clock. The current state moves from power start to S0_Start. The next state is determined using a case statement to check the current stat and if statements to check the inputs. The output signals are defined based on the current state.

Camera Control State Machine Simulation. The Camera Control State Machine was verified using the MATLAB HDL Verification toolbox. The first step in the verification process is to determine that the system is moving through the correct states. The calibration done signal is generated in MATLAB to simulate the signal from the sensor. Figure 2.12 shows the system moving from S4_Start through five states to S3_Read.

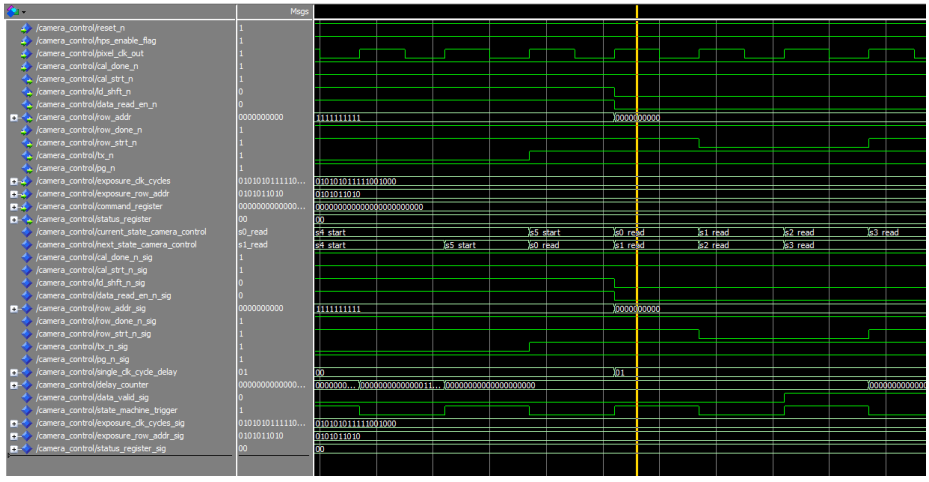


Figure 2.12: The Camera Control State Machine showing the movement from S4_Start to S3_Read.

With the correct current state transition verified, the second step is to verify the output signals from each state. The output signals, specifically, PG_n, TX_n,

and LD_Shift_n are verified in the simulation by comparing the current state with the output value.

Camera Control Avalon Wrapper

The Camera Control State Machine is implemented within the Camera Control Avalon Wrapper. The Camera Control Avalon Wrapper implements the Avalon interface to connect the state machine to the HPS within a custom Qsys component. The wrapper contains both read and write signals to allow the user to set or read registers including the exposure length, the command registers, or the status register.

Hyperspectral Image Pixel Preprocessing

Resonons optical system spreads the light across the pixels on the monochrome image sensor. Resonon requires more light per spectral bin than spectral resolution. To achieve this requirement, the pixel data is averaged, reducing the number of spectral bins but increasing the amount of light per spectral bin.

The Hyperspectral Sensor Pixel Preprocessing Component are the initial computation steps which are camera specific and broken into three steps. The first step is Pixel Averaging, where 8 pixels are summed together. The second step is to add the pixel location information to the pixel data. The final step is to combine the pixel data with the light and dark correction values, which is stored in DRAM data, to create a super pixel. The light and dark correction values are used to normalize the image data creating a uniform brightness level for all the pixels. The super pixel contains the following information.

1. The Spectral Bin Number

2. The Pixel Number

3. The Spectral Bin Data

4. The Light Correction Value

5. The Dark Correction Value

The super pixels are then parallelized into groups of five, during the combination step, before being passed to the computation unit.

Pixel Preprocessing Component

The Pixel Preprocessing Component instantiates two Pixel Averaging components, the Pixel Location component, and the Pixel Parallelization component.

Implementation. With ten pixels arriving in parallel on a single clock cycle, some data is registered before being summed. On the first clock cycle, eight pixels are added while two pixels are registered. On the second clock edge, the two registered pixels are added with six new pixels. This requires four pixels to be registered and saved until the following clock cycle. On the third clock cycle, the four old pixels are summed with four new pixels, resulting in six pixels being registered until the next clock cycle. On the fourth clock cycle, the six registered pixels are summed with two new pixels requiring eight pixels to be registered. Instead of registering these eight pixels, they are summed using a second averaging component. On the fourth clock cycle, two results are passed to the pixel location and pixel parallelization blocks. The following pseudo code shows the data being registered before being passed to the computation block.

```
--Registering the Data
if HPS_Enable =1 and Data_Valid = 1 then
    repeat every 4 clock cycles
        if clock edge 1 then
            pass on pixels 0-7
            register pixels 8 and 9
```

```
7           else if clock edge 2 then
8               pass on registered pixels 8-9 with pixels 0-5
9               register pixels 6-9
10          else if clock edge 3 then
11              pass on registered pixels 6-9 with pixels 0-3
12              register pixels 4-9
13          else if cock edge 4 then
14              pass on registered pixel 4-9 with pixels 0-1
15              pass on pixels 3-9 to second averaging block
16          end
17      end
18  end
```

```
1   --Computation Block
2   if HPS_Enable =1 and Data_Valid = 1 then
3       averaged_spectral_pixel = pixel_0 + pixel_1 + pixel_2 +
            pixel_3 + pixel_4 + pixel_5 + pixel_6 + pixel_7
4       location_pixel = spectral_bin_number & pixel_number &
            averaged_spectral_pixel
5       super_pixel = spectral_bin_number & pixel_number &
            averaged_spectra_pixel & light_ value & dark_value
6   end
```

Simulation. To verify the Pixel Preprocessing Component, a device under test (DUT) VHDL file was created. The DUT file instantiated the single DRAM FIFO and filled it with simulated data. The data was written with the correct location information but with constant light and dark values. When the FIFO was full, the HPS Enable Flag was set to initialize the system. The DUT file also, divided the super pixel into the five individual data packets to be passed out to MATLAB.

Pixel Averaging

The Pixel Averaging step sums eight 10-bit pixels to create a single 13-bit pixel.

Implementation. The addition is completed using a three-level summing tree. The first level adds pixels one and two together, pixels three and four together, pixels five and six together, and pixels seven and eight together. The additions are complete in parallel in one clock cycle. The second level of the tree adds the results of pixel

one and two with the result of pixel three and four, and the results of pixel five and six with the result of pixel seven and eight. These two additions are completed in parallel. The third level of the tree adds the results of the second level of the tree to compute the resulting pixel value. The Pixel Averaging step takes four clock cycles to complete.

By summing eight pixels together, the signal to noise ratio of the hyperspectral data is increased. The addition reduces the number of spectral bins from 1240 to 155. With the addition tree implemented, the VHDL code was simulated for verification.

Simulation. The Pixel Averaging step was verified using the MATLAB HDL Verification toolbox. For this simulation, random ten-bit data was generated using the MATLAB fixed-point toolbox and used as the inputs to the component. The Pixel Averaging step computed the sum over three clock cycles and the results were compared to the MATLAB script computing the same values using the fixed point toolbox. The resulting wave forms can be seen in Figure 2.13. Because the simulation only returns the final value for comparison, the intermediate pipeline steps were computed by hand and verified using the results from the simulation wave form.

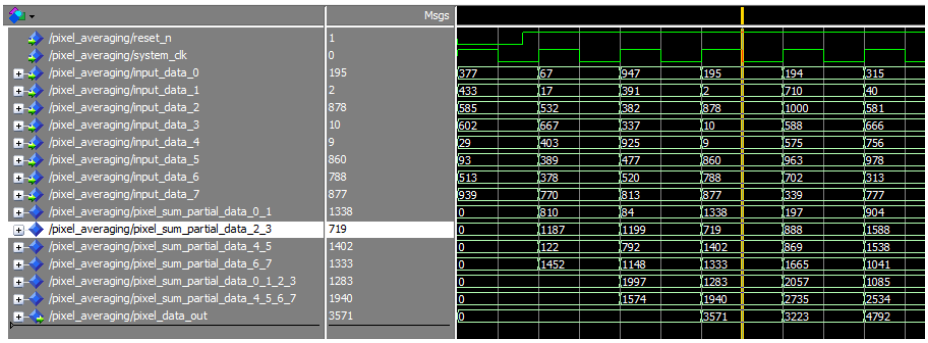


Figure 2.13: The verification of the summing step, Pixel Averaging, using ModelSim.

With the Pixel Averaging verified, the pixels are passed to the Pixel Location block.

Pixel Location

The Pixel Location block adds the spatial pixel number and the spectral bin number to each pixel. This information is used during the Pixel Parallelization step to verify the correct DRAM data is being joined with the current pixel.

Implementation. The Pixel Location is tracked using a single loop to count the pixel number and the spectral bin number. The pixel number and the spectral bin number begin at 1. Once the component is enabled by the HPS, the counting process updates on every clock edge. Initially, the process counts five clock cycles, waiting for the first valid pixel. On the fifth clock cycle, the pixel data and location valid flag is set to '1'. This signifies that the pixel data has been summed and is ready for the pixel location. The pixel data and location valid flag triggers the counting steps, implemented using a series of if statements, to begin.

The counter if statements begins with the end of frame condition, where the pixel number is equal to 256, spectral bin number is equal to 155, and a new frame is going to begin. If these conditions are met, the pixel number and row number are reset to one and the new frame flag is set. Also, the pixel data and location outputs are assigned. If the end of frame condition is not met, the end of pixel condition, the spectral bin number equal to 155, is checked. If the end of pixel condition is met, the spectral bin number is reset, the pixel number is incremented by 1 and the output signals are assigned. If neither of the above conditions are met, the spectral bin value is incremented by 1 unless the double output is needed. If the double output is needed, the spectral bin value is increased by two.

The double output is tracked using a separate process. The process counts the number of clock cycles and reset the counter to zero if the counter equals three.

Otherwise, the counter increments by one. When the counter reaches three, then double output flag is enabled.

Simulation. The pixel location information was verified using the HDL Verification toolbox in MATLAB. The input to the system was the results from the pixel addition block. The component was checked for the correct location values, roll over, and double output. With the pixel location data verified, the pixel data packet is sent to Pixel Parallelization component for packaging with the DRAM data.

Pixel Parallelization

The Pixel Parallelization component combines the pixel data packet with the DRAM data. The light and dark correction matrices are stored in DRAM before being read and stored in a first in, first out (FIFO) buffer. The design and verification of this component was completed by Monica Whitaker [4].

Implementation. The Pixel Parallelization component is implemented using three state machines and five FIFO buffers. The first state machine is the DRAM Read State Machine that reads the light and dark data from the single DRAM FIFO and saves the data in five individual FIFOs. The second state machine, the FIFO Read State Machine, reads the DRAM data from the five individual FIFOs. The final state machine is the Location Check State Machine that checks the location information before combining the DRAM and pixel data into a super pixel.

The DRAM Read state machine was designed using the three process implementation. The first process, the DRAM Start Process, is triggered on reset and on the DRAM Clock. On reset, the process sets the current state to the initial read state, S_Start_DRAM_Read. On the rising edge of the DRAM clock and if the system is

enabled by the HPS, the current state is updated to the next state. The next state is determined in the second process, the DRAM State Transition Process.

The DRAM State Transition process updates the next state signal. The component moves from S_Start_DRAM_Read to S0_DRAM_Read. The state machine then moves to S1_DRAM Read if the DRAM 1 Data FIFO is not full. Otherwise the component stays in the same state. The DRAM Data FIFO stores the light and dark data to be used in the Location Check State Machine. In the S1_DRAM_Read state, the component could either move to the next state, S2_DRAM_Read, or stay in the current state. The system will move to the next state if the DRAM 2 Data FIFO is not full. The system moves to S3_DRAM_Read from S2_DRAM_Read if the DRAM 3 Data FIFO is not full. The state machine will move to S4_DRAM_Read from S3_DRAM_Read if the DRAM 4 Data FIFO is not full. From S4_DRAM_Read, the system will return to S0_DRAM_read if the DRAM 0 Data FIFO is not full. The state machine will not move to the next state if there was no room for new data. With the next state logic defined, the third process is the output logic.

The output logic procudes the output signals based on the current state of the system. Each state triggers a read from the single DRAM FIFO and stores the data into five individual FIFOs. The read occurs if the DRAM FIFO is not empty and if there is room in the individual FIFOs for the data. The read and write signals for the different buffers are controlled using a case statement to determine the current state and if statements to determine the correct read and write signals. A read from the single DRAM FIFO was completed by setting the s_buffer_read_signal to a '1'. A write to any of the individual FIFOs was completed with by setting the dram_wrreq_'x'signal to a '1', where 'x'ranges from zero to four depending on which FIFO is being written to. With the DRAM data moved from the single FIFO and parallelized in five FIFOs, the next step is to read the individual FIFO data.

The DRAM data is read out of the individual FIFOs using a state machine, implemented using the three process structure. The first process is the FIFO Read Start Process and is almost identical to the DRAM Read Start Process. The two difference are that the component is triggered on the component clock not the DRAM clock and the process resets to the state S3_FIFO_read instead of S_Start_DRAM_Read.

The second process is the FIFO Read State Transition Process which assigns the next state based on the current state. The current state is checked with a case statement. The state machine moves from S0_FIFO_Read to S1_FIFO_Read to S2_FIFO_Read to S3_FIFO_Read and back to S0_FIFO_Read. With the next state logic defined, the output signals are determined and assigned in the third process.

The output signals are assigned during the FIFO Read Output Process. Depending on the current state, again checked with a case statement, the output signals are assigned to read from the different FIFOs. The data from the different FIFOs are sent to the Location Check State Machine to be combined with the pixel data. S0_FIFO_Read reads from the DRAM 1 FIFO by asserting the signal dram_rdreq_to '1', if the DRAM 1 FIFO is not empty. It takes one clock cycle to read the data from the FIFO. S1_FIFO_Read reads data from the DRAM 2 FIFO. S2_FIFO_Read reads data from the DRAM 3 FIFO and the DRAM 4 FIFO. This provides the two sets of DRAM data to the Location Check State Machine when there are two pixels from the pipeline. S3_FIFO_Read reads data from the DRAM 0 FIFO. All the data saved in the DRAM FIFOs are used during the Location Check State Machine.

FThe Location Check State Machine is a five-state state machine built using the three-process method. The first is the Location Check Start Process. On reset, this process updates the current state to S_Start_Location_Check. On the rising edge of

the system clock, the current state is updated to the next state. The next state is determined in the Location Check Transition Process.

The Location Check Transition Process updates the next state based on the current state. S_Start_Location_Check moves to state S1_Location_Check. S1_Location_Check moves to state S2_Location_Check. S2_Location_Check moves to state S3_Location_Check. S3_Location_Check moves to state S0_Location_Check. S0_Location_Check returns to state S1_Location_Check. With the state transitions established, the outputs of the state machine were assigned using the Location Check Output Process.

The Location Check Output Process checks the pixel location of the DRAM data and compares the data to the pixel location data of the incoming pixel before assigning an output. In the S_Start_Location_Check state, the current pixel location is compared to the location value from the light and dark values from DRAM 0 FIFO. If the locations match, the light and dark values are combined with the current pixel information which is assigned to data register zero. The super pixel valid flag, the new frame flag and the clear pixel flag are cleared. If the locations do not match, data register zero, the super pixel valid flag, and the new frame flag cleared. The clear pixel flag is set and the pixel location check is set to the current pixel number. Until the pixel number updates, no new data will be sent to the computation unit and the clear pixel flag will remain high.

In S0_Location_Check, the clear pixel flag is checked first. If the clear pixel flag is set to a zero, then the current pixel location is compared to the location of the data from the DRAM 0 FIFO. If the locations match, then data register 0 is updated and the super pixel valid flag, the new frame flag, and the clear pixel flag are cleared. If the locations do not match, then the data register zero, the super pixel valid flag, and the new frame flag are cleared. The current pixel number is saved to the pixel

location check register and the clear pixel flag is set. If the clear pixel flag is set to a one, then the pixel location check register is checked. If the pixel location check register equals the last pixel in the frame, then the new pixel number is checked. If the new pixel number is zero, then the location of the new pixel is compared to the location of the light and dark correction data from DRAM 0 FIFO. If the locations match, data register zero is updated and the correct flags are cleared. If the location does not match, then data register zero and the correct flags are cleared, the pixel number is registered and the clear pixel flag is set. If the pixel location check register is in the middle of the frame, then the check looks for the next pixel number. If the pixel number is one more than the pixel location check register, the light and dark location is compared to the current pixel location. If the locations match, then data register zero is updated and the flags are cleared. If the location does not match, then data register zero is cleared, the pixel number is saved, and the clear pixel flag is set. If registered pixel number does not match any of the above conditions, then data register zero is set cleared and the clear pixel flag remains high.

In S1_Location_Check, if the clear pixel flag is a zero, the location of the current pixel is compared to the location of the light and dark data from DRAM 1 FIFO. If the locations match, data register one is created by combining the current pixel with the light and dark data. Data register zero is concatenated to the end of data register one. The super pixel valid flag, the new frame flag, and the clear pixel flag are cleared. If the locations do not match, the pixel number is registered, the clear pixel flag is set, and the super pixel and new frame flags are cleared. If the clear pixel flag is a one, then data register one, the super pixel valid flag, and the new frame flag are cleared. The clear pixel flag is set. The next pixel conditions are not checked because the new pixel will begin in the S0_Location_Check state.

In S2_Location_Check, if the clear pixel flag is a zero, the location of the current pixel is compared to the location of the light and dark data from DRAM 2 FIFO. If the locations match, data register two is created by combining the current pixel data with the light and dark data and concatenating data register one to the end. The the super pixel valid and the new frame flags are cleared. If the location does not match, then the data register two is cleared, the flags are cleared, the pixel number is saved, and the clear pixel flag is set. If the clear pixel flag is a one, data register two is set to zero and the clear pixel flag is set.

In S3_Location_Check, if the clear pixel flag is a zero, the location of the two current pixels are compared to the location of the light and dark data from DRAM 3 FIFO and DRAM 4 FIFO. If both locations match, then the location of first data packet is checked. If the first location is equal to zero, the new frame flag is set. If not, the new frame flag is cleared. Either way, data register three is created by combining the light and dark data with the pixel data and concatenating data register two to the end. The super pixel flag is set and the clear pixel flag is cleared. If the locations do not match, data register three is cleared and the super pixel valid and the new frame flags are cleared. The current pixel number is saved and the clear pixel flag is set. If the clear pixel flag is a one, then data register three, the super pixel valid flag, and the new frame flag are cleared while the clear pixel flag is set.

Prototype Sensor Board Verification

With the prototype sensor board completely assembled and connected to the FPGA, the sensor control can be verified. The first step in the process is to implement the Camera Control Avalon component within a Qsys system along with the memory block component. The memory block component connects on-chip RAM with the JTAG Master Interface, so that the data can be read by an external command line

interface. Three processes are implemented to capture and write the data to the on-chip RAM.

The process, register data, stores the valid image data, and sets the memory block pixel valid flag to a one. The process is triggered on reset and on the rising edge of the sensor clock. On reset, the pixel data and the memory block pixel valid flag are cleared. On the rising edge of the sensor clock, the data valid flag is checked. If the data valid flag is a one, then the current pixel information is saved into a 100-bit vector containing all ten pixels and the memory block pixel valid flag is set to a one. Otherwise, the pixel data and the memory block pixel valid flag are cleared. The data valid flag is set in the second process, the data valid process.

The data valid process determines when the sensor data is valid by checking the calibration done input signal from the sensor. The process is triggered on reset and on the rising edge of the sensor clock. On reset, the data valid flag, the calibration done flag, and the calibration done delay counter are cleared. On the rising edge of the sensor clock, the calibration done signal is checked. If the calibration done signal is a zero, then calibration done flag is set to a one. Also, on the rising edge of the sensor clock, the calibration done flag and the calibration delay counter are checked. If the calibration done flag is a one, the calibration delay counter is checked. If the calibration delay counter is 100, the data valid flag is set. Otherwise, the data valid flag is cleared and the delay counter is implemented by one. The data valid flag is used in the third process.

The third process updates the memory block address. The process is triggered on reset and on the rising edge of the camera clock. On reset, the memory address is cleared. On the rising edge of the camera clock, the data valid flag is checked. If the data valid flag is a one, then the memory address is checked. If the memory address is

at its maximum, then the memory address rolls over to zero. Otherwise, the memory address is incremented by one.

The system is currently under verification using the Signal Tap Logic Analyzer to monitor internal command signals and to see the returned command signals.

LINE SCAN OBJECT TRACKING

The Waterfall Sorting Subsystem implements a line scan camera to provide the data for the object tracking algorithm. In this prototype, the line scan camera is connected to the Arria 5 SoC FPGA Development Board through a custom PCB using the FMC connector. The object tracking algorithm determines the edges of falling objects and provides the information to the Hyperspectral Processing Subsystem. The object tracking algorithm also provides the center of object information to the Air Jet Subsystem.

Camera Link Interface

The line scan camera selected for this project is the Basler Racer raL8192-80km, a monochrome line scan camera producing 8,000 pixels per line and running at 80,000 lines per second [11]. This results in the camera being able to generating 640 megabytes of data per second. The data is being transmitted over the Camera Link protocol. Because of the protocol, a custom PCB was required to connect the Camera Link interface to FPGA. [12]

Prototype Camera Link Interface Board

The first step is to design the schematic for the Prototype Waterfall Interface Board. The board is designed to connect to the Altera Arria 5 Development Board through the FMC connector. The line scan camera is connected to the PCB through two 26-pin mini Camera Link SDR connectors. The 26 pins provide the camera data through low voltage differential signaling (LVDS) pairs. The differential pairs are connected to deserailizer integrated circuits (IC) that convert the serial data to parallel data. Three deseralizing ICs were placed on the board to handle the full

Camera Link interface with each deserialization chip handling one of the Camera Link channels. The channels transmit 24-bits of serial data over four LVDS pairs. The parallel data is then connected to the FPGA through an FMC connector. The parallel data lines were routed with the goal of shortening the trace lengths between the deserialization chips and the FMC connector. With the camera data routed, a 40-pin header was added to the schematic to connect to the air jet control lines.

With the schematic completed, the next step in the is to place the components on the board. The first component placed on the board is the high pin count male FMC connector which must be orientated correctly on the bottom layer to connect to the high pin count female FMC connector on the Arria 5 Development Board. The orientation of the FMC connector was verified using a paper model. With the FMC connector in place, the 40-pin connector was placed above the FMC connector on the top layer. Third, the mini Camera Link connectors were placed on the edge of the board. Fourth, the deserialization IC were placed between the Camera Link connectors and the FMC connector. The goal was to limit the trace length of the LVDS pairs and the parallel data lines. Finally, the decoupling capacitors for the deserialization chips and the termination resistors for the LVDS pairs are placed on the top and bottom layers. The component placement can be seen in the Unrouted Waterfall Board in Figure 3.1.

Third, the layer stack up was determined to provide power to the components and to provide layers to route the data signals. Table 3.1 provides the layer stack up for the Prototype Waterfall Board. The first layer, top layer, is a component and routing layer containing, the deserialization ICs, the 40-pin header, the mini Camera Link connectors, and half of the termination resistors. The second layer is an internal routing layer. The third layer is the ground plane. The fourth and fifth layers are internal routing layers. The sixth layer is the power plane, providing +3.3V to the

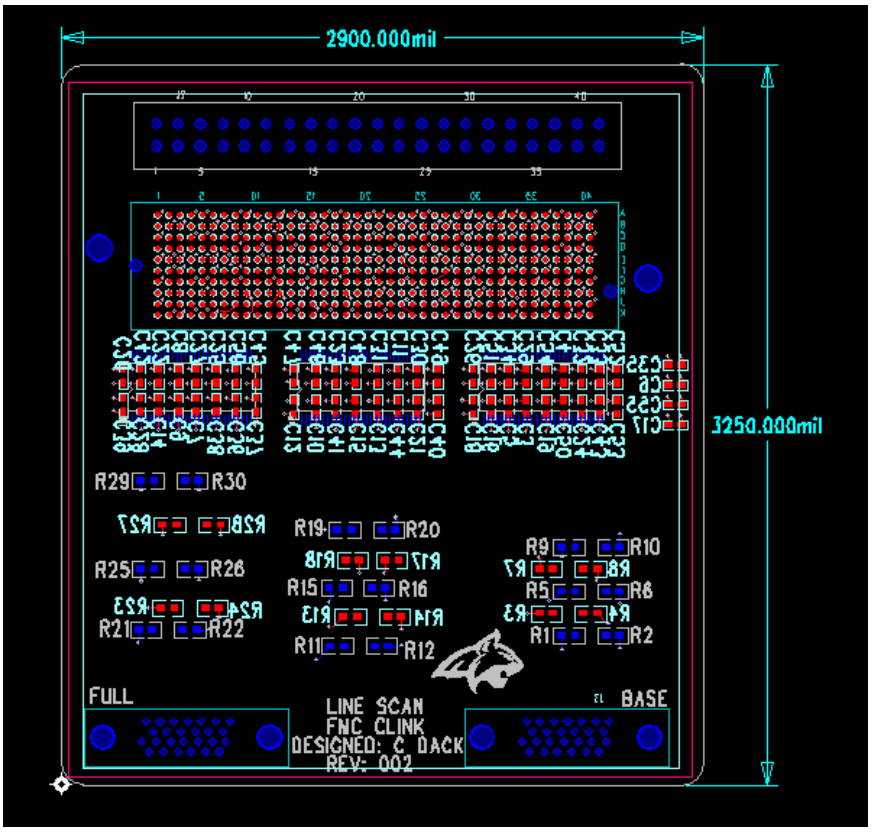


Figure 3.1: The unrouted waterfall board showing the placement of the FMC connector, the Camera Link connectors, and the Air Jet header.

components on the board. The seventh layer is an internal routing layer. The eighth layer, the bottom layer, is a component and routing layer, containing the decoupling capacitors, the FMC connector, and half of the termination resistors. All of the routing layers have a minimum trace width of 5 mils and all layer contain 1.0 oz of copper.

With the layer stack up defined, the fourth step is to route the board. The LVDS lines coming from the Camera Link cables must be the same length so that the data arrives at the deserialization IC at the same time. The LVDS pairs were routed by hand starting with the traces that had the longest distance between pins. The shorter traces were matched by adding additional copper length. The parallel

Table 3.1: Prototype Waterfall Board Stack Up

Layer number	Layer Name	Copper Weight	Copper Thickness	Prepreg and Core Material	Prepreg and Pressed Core Thickness	Trace Width
				Soldermask - Green		
1	Top	1.0oz	0.00135			0.005
				FR-4 (Er = 4.5)	0.0074	
2	Layer 2	1.0oz	0.00135			
				FR-4 (Er = 4.5)	0.0080	
3	GND	1.0oz	0.00135			0.005
				FR-4 (Er = 4.5)	0.0070	
4	Layer 4	1.0oz	0.00135			0.005
				FR-4 (Er = 4.5)	0.0080	
5	Layer 5	1.0oz	0.00135			
				FR-4 (Er = 4.5)	0.0070	
6	VCC	1.0oz	0.00135			0.005
				FR-4 (Er = 4.5)	0.0080	
7	Layer 7	1.0oz	0.00135			0.005
				FR-4 (Er = 4.5)	0.0074	
8	Bottom	1.0oz	0.00135			
				Soldermask - Green		

data lines from the deserialization ICs to the FPGA must also be the same length. These parallel lines were routed using the auto routing tool. The resulting traces can be seen on the Routed Prototype Waterfall Board in Figure 3.2.

With the board designed and routed, the board was sent to a fabrication house for construction. The boards were then assembled by hand using a soldering iron and the reflow oven. With the boards assembled, the Waterfall Prototype Board was connected to the Arria V Development Board. The first step is to verify that data is moving from the camera to the FPGA. During the initial test, a mistake with the schematic was found. Pins 27 and 28 were grounded. Pin 27, however, is the zeroth bit of the parallel data packed. Pin 28 is still ground.

Before designing a new revision of the Waterfall Prototype Board, the Camera Link transmission lines were routed to the transceiver receive pins of the FMC connector through the 40-pin header. The goal is to use the deserialization ability of the FPGA transceivers to create the parallel data. The parallel data is then passed to the Object Tracking Algorithm.

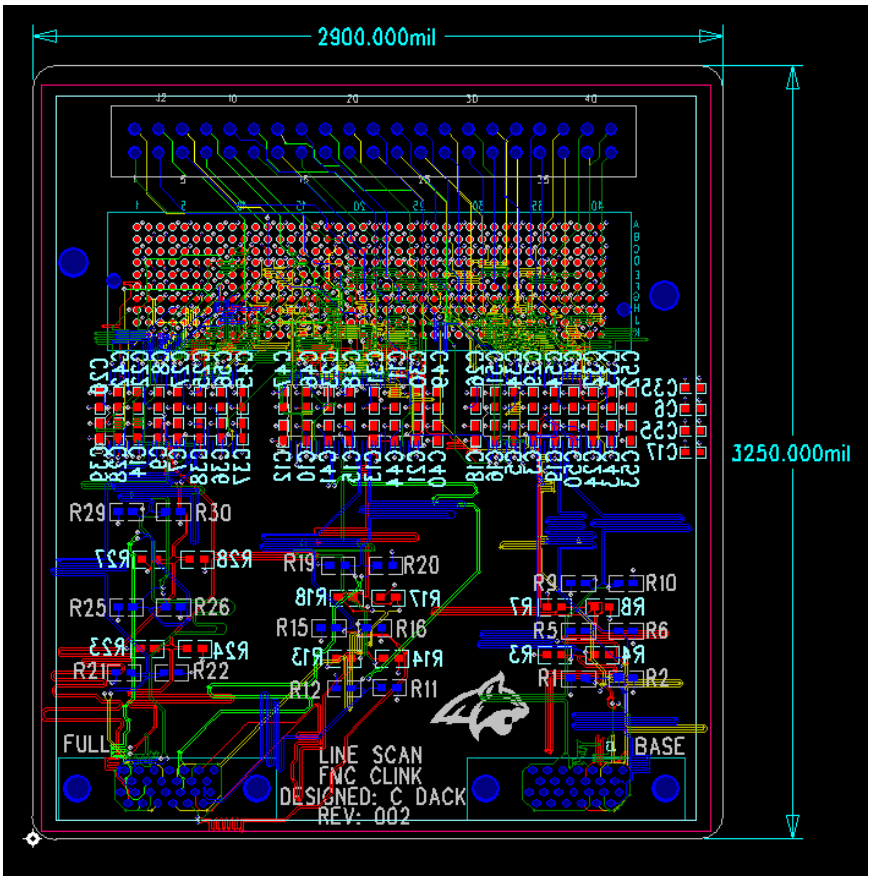


Figure 3.2: The routed waterfall board showing the matching trace lengths.

Line Scan Camera Interface

The camera interface section discusses the Camera Link Interface and Module 1. The Camera Link Interface implements the Camera Link protocol to move data from the line scan camera into the FPGA. Module 1 converts the 10 8-bit parallel pixels into a single Boolean ten-bit vector. The Camera Interface component is camera specific depending on the data transfer protocol and the number of pixels coming in in parallel.

Camera Link Interface

The Camera Link protocol transmits the camera data over LVDS pairs. The original goal was to implement the deserialization of camera data using integrated circuits. However, due to the schematic mistake, the LVDS pairs were routed to the transceiver receive pairs of the FPGA. The data will then be deserialized using the FPGA before being passed to Module 1. The Camera Link Interface component is currently under development.

Module 1 - Pixel to Data Array Conversion

Module 1 converts the pixel value into a single bit signifying if there is valid object. Ten pixels arrive on a single clock edge, are converted in parallel, and saved into a single ten-bit array. The ten-bit is then passed to the Object Tracking Algorithm.

Implementation. With the physical camera set up, the objects being scanned are falling in front of a solid colored background. Since the camera is a monochrome camera, a white background is necessary. Each pixel is a grey value between 0-255, the maximum value represented by eight bits. If the pixel value is found to be within a specific threshold, say 25-220 for example, the pixel is represented by a one, otherwise the pixel is represented with a zero.

The pixel comparison is implemented using a single process that is triggered on reset and on the pixel clock. On reset, the output array is cleared. On the rising edge of the pixel clock, the comparison is run on each pixel in parallel. The pixel comparison is completed using a set of if statements that checks whether the pixel value is within the threshold. If the pixel is within the threshold, then the corresponding bit in the output array is set to one. If the pixel is outside the threshold,

the corresponding bit is set to zero. The output ten-bit vector is then passed to the Object Tracking Subsystem.

Simulation. With Module 1 being within the Camera Interface component, the verification of Module 1 uses a test image that is 2040 pixels wide and 512 lines tall. The image is a simple gray circle on a white background as seen in Figure 3.3. The gray circle represents the lentil after it has been scanned by the monochrome line scan camera.

Figure 3.3: An example test image where the gray circle is the lentil and the white is the background data. The lentil is 125 pixels wide while the image is 2040 pixels wide.

With the test image created, the first step is to parse the data into individual lines. The lines are then broken into ten-pixel blocks and are converted to the eight-bit fixed-point data using the MATLAB fixed-point toolbox. The fixed point pixel data is used to drive the inputs to the Module 1 block and the results are saved. The results are verified by recreating the VHDL logic using fixed-point simulation. The fixed-point input data is compared to the fixed-point threshold data. If the pixel data is within the threshold, then the corresponding bit is set by adding

$$result = result + 2^{(9-pixel_location)} \tag{3.1}$$

to the result. For pixel zero, the value added to the result is 512, representing the 10th bit, or most significant bit, in the array. For pixel nine, the value added to the result is one, representing the 0th bit, or the least significant bit, in the array. The fixed point simulation data is compared to the VHDL result for accuracy. The fixed-point simulation is used to create the input data for Module 2.

Object Tracking Algorithm

The Object Tracking Subsystem is the VHDL component that handles tracking the edges of the object, sending information about the edges of the object to the Hyperspectral Computation Subsystem, and passing information about objects that will be ejected to the Air Jet Subsystem. The Object Tracking Algorithm is implemented using four subcomponents. The first subcomponent is Module 2, that converts the pixel array data into sub-segments. The second component is Module 3, which converts the sub-segment data into segment data. The third component, Module 4, converts the segment information into object information. The object information is passed to the Hyperspectral Computation System and to Module 5. The fourth subcomponent is Module 5, which computes the vertical center of the object. The vertical center of the object is combined with the object information from Module 4, to create the Air Jet Packet.

Module 2 - Tap to Sub-segment Conversion

Module 2 converts the incoming ten-bit vector into a sub-segment. A sub-segment is the start and end values of the valid data in the ten-bit vector from Module 1. The start and end values could range between zero and nine depending on the location of the ones in the vector. Module 2 also tracks and resets the line number.

Module 2 is broken down into two parts: the sub-segment creation and the line counter and reset. The sub-segments created are used to create the object information per line and then the objects. The line counter keeps track of the current line for use in determining the vertical center of the object. The line counter is also used during the Air Jet subsystem being developed by the Air Jet Senior Design Team. The line counter is occasionally reset when the line reset flags are cleared and there are no objects in the processing pipeline. The line counter reset flag is also sent to the Hyperspectral Computation Subsystem to keep both systems synchronized.

Implementation. The sub-segments are created from the ten-bit tap vector that comes from Module 1. A process is used to implement a for loop that determines the sub-segments. The process is triggered on reset and on the rising edge of the component clock. On reset, the output vectors are cleared. On the rising edge of the component clock, the process first checks if the system is enabled by the HPS. If the system is enabled, the process sets all the internal variables to their initial condition. Then the process begins the ten-step for loop, which is iterated on every clock edge. The for loop first checks to see if there is valid object data, a one, in the ten-bit vector. If there is valid data, the next check is the last bit variable. The last bit variable is set to a one if the last bit checked was a one. The last bit variable is set to a zero if the last bit checked was a zero. If the last bit variable is a one, the current bit value is appended to the sub-segment. This is accomplished by increasing the end value of the sub-segment by one. The object found flag is also set during this check. The component can handle up to two sub-segments at one time.

The active sub-segment is determined during the creation of a new sub-segment. If there is valid data but the last bit flag is a zero, then a new sub-segment must be created. The sub-segment pointer is set to ‘00’when the process begins. If a new

sub-segment is created, the sub-segment pointer is set to '01'. If a second new sub-segment is created, the sub-segment pointer is incremented to '10'. Theoretically, a third sub-segment could be created with the sub-segment pointer being set to '11', however the second sub-segment would only be a few pixels wide. Each pixel from the line scan camera is 50um. The object represented by the sub-segment would be a maximum of 300um. The lentils that are being sorted are 2.54 mm in diameter, which covers 2,000-pixels.

The second process in Module 2 is the line counter process. This process tracks the line number, setting the end of line flag, and resetting the line number. The line number is incremented at the end of each line when the end of line flag is set. The end of the line is determined using two counters. The first counter, pixel counter, is updated on the rising edge of the system clock, if the sub-segment ready flag is set. The pixel counter counts from zero to the number of pixels in the area of interest. If the pixel counter is 240, then three if statements are checked and the end of line flag is set to a one. First, the object found flag, the line reset variable, the line empty counter, the transceiver FIFO empty flag, and the air jet FIFO empty flag are checked. If the object found flag is a one, or the line reset variable is a one, or the line empty counter equals eight, the ratio of the number of line scan lines per hyperspectral line, and both transceiver FIFOs are empty, then the line empty variable is set to zero. Otherwise, the line empty variable is incremented by one. The line empty variable tracks the number of empty line scan lines. Second, the process checks the current line counter value. If the line counter is equal to the maximum line counter value, the counter rolls over to zero, the pixel counter is cleared, the line reset variable and the line reset signal are set to one. If the line counter is not at the maximum value, then line empty variable and the transceiver FIFO empty flags are checked. If the line empty variable is eight and the FIFOs are empty, the line number is cleared, the

pixel counter is reset, the line reset variable and signal are set to one. Otherwise, the line number is incremented by one, the pixel counter is cleared, and the line reset flags are set to zero. Third, the process checks the line reset signal. If the line reset signal is a one, then line reset signal is cleared and the end of line signal is cleared. If the pixel counter has not reached the end of the area of interest, then end of line flag is set to zero, the line reset signal is set to zero, and the pixel counter is incremented by one.

Simulation. Module 2 is the first component to be verified in the object tracking pipeline. A smaller test image can be used since the system is looking at objects and not the actual camera data. Figure 3.4 shows a test image that was used for verification of the Object Tracking System. The test image contains three gray lentils on a 1000 pixel wide image and 512 lines tall.

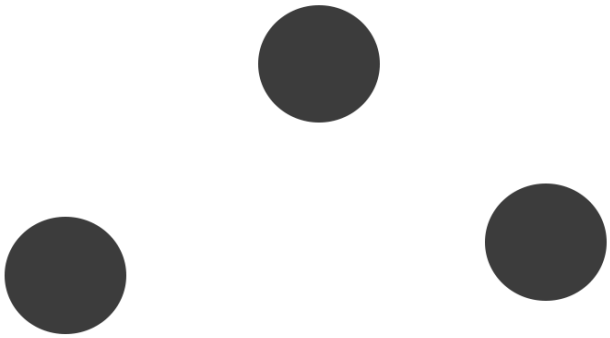


Figure 3.4: A multiple object test image to verify the object tracking algorithm. This image targets the creation of multiple objects to be passed to the hyperspectral image processing unit as well as the air jet component.

The MATLAB script first implements the same parsing and array creation algorithm implemented in the Module 1 test bench. This allows the inputs to be

generated without the need to rerun the Module 1 simulation. With the test data created, the test bench drives the Module 2 VHDL component. The resulting wave forms are displayed in ModelSim, where they are visually verified. The results are also returned to MATLAB for use in future test benches.

Figure 3.5 shows the input data from MATLAB as well as the clock and reset lines. The yellow line shows the input data of "000000011". This results in the start value of this sub-segment being 8 and the end value being 9. The next piece of data contains the information "1111111111". This results in the start value of 0 an the end value of 9. The third piece of data contains the information "1100000000". This results in a start value of 0 and an end value of 1. The start and end values are saved, so that they can be passed to Module 3.

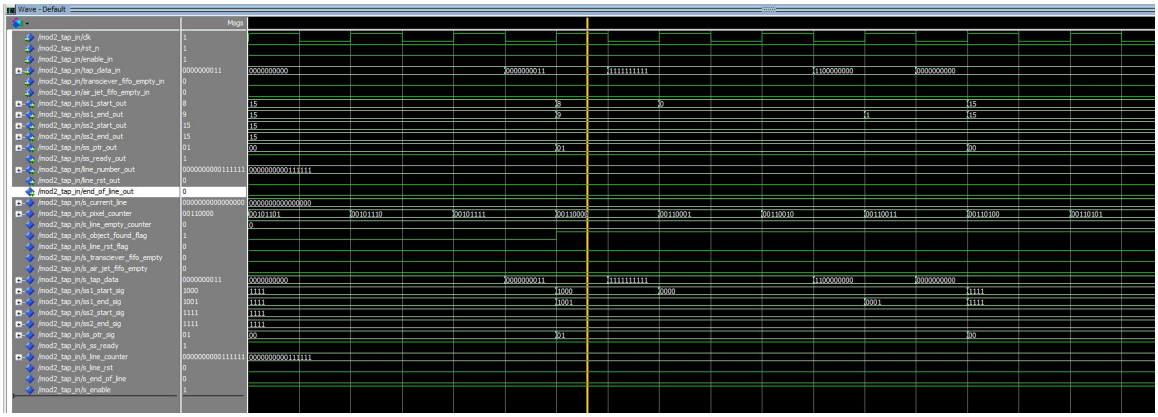


Figure 3.5: An example of the simulation wave forms from the Module 2 Test Bench.

Module 3 - Sub-segment to Segment Conversion

Module 3 combines the sub-segment information into a single segment by comparing the edges of each sub-segment. If the edges are both '1', then the sub-segments are combined. The edges of the segments are also assigned a pixel number. The segment information is passed to Module 4 to be converted into objects.

Implementation. Module 3 uses a control process to coordinate the flow of data and a component to create the segment information.

The control process, sub-segment to segment process, controls the flow of data into the component. The process is triggered on reset and on the rising edge of the component clock. On reset, the sub-segment pointers, the segment pointer, the last bit flag, and the tap counter are cleared. On the rising edge of the component clock, the process checks the sub-segment ready flag. If the sub-segment ready flag is a zero, then the sub-segment pointers, the segment pointer, the last bit flag, and the tap counter are cleared. If the sub-segment ready flag is a one, then the sub-segment one start value is checked. If the start value is zero, then the first bit on the sub-segment is a one and the first bit variable is set to one. Otherwise, the first bit variable is set to zero. Also, the end of line flag is checked. If the end of line flag is a one, then the sub-segment pointer and the segment pointer are checked.

If the sub-segment pointer is either one or two and the segment pointer is less than 1023, then the last bit flag and the first bit variable are checked. The last bit flag is determined by checking the end value of the previous data block. If the last bit flag is a one and the first bit is a one, then the sub-segment one route signal is set to one and the segment pointer value is checked. If the segment pointer is 1022, then the sub-segment one end value is checked. If the sub-segment one end value is not nine, then the segment pointer is set to 1023. Otherwise, the sub-segment pointer is set to one. Either way, the sub-segment two route signal is cleared. If the segment pointer is less than 1022, then the sub-segment pointer is checked. If the sub-segment pointer is two, then the sub-segment two route signal is set to one and the segment pointer is set to two. If the sub-segment pointer is not two, then the sub-segment two route signal is cleared. If the last bit flag and the first bit variable are not one, then the segment pointer is checked. If the segment pointer is equal to 1022, then

the segment pointer is set to the 1023 and the sub-segment route signals are cleared. If the segment pointer is less than 1022, then the sub-segment one route signal is assigned the segment pointer plus one and the segment pointer is incremented by one. If the sub-segment pointer is two, then the sub-segment two route signal is set to the segment pointer plus two and the segment pointer is incremented by two. If the sub-segment pointer does not equal two, then the sub-segment two pointer is cleared. If the sub-segment pointer does not equal one or two and the segment pointer is not equal to 1023, then the sub-segment one and two route signals are cleared and the segment pointer is set to zero.

If the end of line flag is a zero, then sub-segment pointer value and the segment pointer value are checked. If the sub-segment pointer value is one or two and the segment pointer is less than the maximum value, then the last bit flag and the first bit variable are checked. If the last bit flag is a one and the first bit variable is a one, then the sub-segment one route signal is set to the segment pointer value and the segment pointer value is checked. If the segment pointer value is one less, then the maximum value, the sub-segment one end value is checked. If the sub-segment one end value is nine, then the segment pointer is set to the maximum value. Otherwise, the segment pointer is registered. Either way, the sub-segment two route signal is cleared. If the segment pointer is less than 1022, then the sub-segment pointer is checked. If the sub-segment pointer is two, then the sub-segment two route signal is set to segment pointer plus two and the segment pointer is incremented by two. If the sub-segment two pointer is not two, then the sub-segment two route signal is cleared. If the last bit flag and the first bit variable are not one, then the segment pointer is checked. If the segment pointer is 1022, then the segment pointer is set to 1023, and the sub-segment one and two route signals are set to zero. If the segment pointer is not 1022, then the sub-segment one route signal is set to the segment pointer plus

one and the segment pointer is incremented by one. If the sub-segment pointer is two, then sub-segment two route signal is set to the segment pointer plus two and the segment pointer is incremented by two. The segment pointer is only assigned at the end of all the checks in the process. If the sub-segment pointer is not two, then the sub-segment two route signal is cleared.

Also, the process checks the sub-segment one end value and the sub-segment two end values. If either of these end values equal nine, then the last bit flag is set to one. Otherwise, the last bit flag is cleared. On every clock cycle, the pixel counter is incremented by ten unless the pixel counter is equal to the end of the area of interest minus ten. In this case, the pixel counter is reset to zero.

The process provides the control signals for the internal subcomponent, Module 3 Segment Information. Module 3 Segment Information uses a single process to determine the start, middle, and end values of the segment information. The start, middle, and end values are pixel numbers in the area of interest.

The process, update segment, is triggered on reset and the rising edge of the component clock. On reset, the segment active signal, and the segment start, middle, and end variables are cleared to zero. On the rising edge of the component clock, the end of line flag is checked. If the end of line flag is a one, then the segment active signal, and the segment start, middle, and end variables are cleared. Otherwise, the segment active flag is checked. If the segment active flag is a one, the sub-segment one and sub-segment two flags are checked. If the sub-segment one flag is a one, then the segment end variable is assigned the pixel counter plus the sub-segment one end value. If the sub-segment two flag is a one, then the segment end variable is assigned the pixel counter plus the sub-segment two end value. If the segment active flag is a zero, then the sub-segment one active and sub-segment two active flags are checked. If either of these flags are set, then the segment active flag is set. If the sub-segment

active one flag is set, then the segment start value is set to the pixel counter plus the sub-segment one start value and the end value is set to the pixel counter plus the sub-segment one end value. If the sub-segment active two flag is set, then the segment start value is set to the pixel counter plus the sub-segment two start value and the end value is set to the pixel counter plus the sub-segment two end value. With the segment variables updated, the segment middle value is computed by subtracting the end value from the start value, dividing by two using a right shift, and adding the result to the start value. The middle value is computed on every clock cycle regardless on new data. The start, middle, and end variables are then assigned to the outputs.

The Module 3 Segment Information component is implemented using a generate statement to create a component for every ten pixels in the array of interest. This allows the system to track one object per ten-pixel block. The active signals are set in an array using a process. The process walks through a for loop and sets the active array cell to one if the for loop counter value equals either the sub-segment one route signal or the sub-segment two route signal. The segment start, middle, and end values are stored and passed out of Module 3 in an array. Module 4 uses the arrays to create the objects.

Simulation. Module 3 was initially verified in simulation using a single object and is currently being verified with the multiple object test image. The pixel data from Module 2 was used as the input for Module 3 and the results were saved using MATLAB. The segment information was verified per line by checking the results with the test image.

Module 4 - Segment to Object Conversion

Module 4 converts the segments into objects, assigning object numbers. The object information is produced per line and the information is passed serially to the

Camera Relations component, which creates the Hyperspectral Object Packet, and to Module 5, which computes the vertical center of the object.

Implementation. Module 4 is implemented using six processes and a subcomponent. The first process is the data register process, which registers the current line when the end of line flag is a one. The second process is the pipeline array data process, which steps through the current line data and sends it to the subcomponent in series. The third process is the new object process, which signifies a new object. The fourth process is the no data process, which checks if there is no data in the current and previous pixel lines. The fifth process is the data valid process which sets the data valid signal. The sixth process is the data out process which serializes the object data for use in the Module 5 and in the Air Jet Packet Creation process.

The data register process is triggered on reset and on the rising edge of the component clock. On reset, the previous line start and end registers are cleared. On the rising edge of the component clock and if the end of line flag is a one, the current line start and end values are saved to the previous line start and end registers.

The pipeline data process is triggered on reset and on the rising edge of the component clock. On reset, the segment start, middle, and end single values are set to zero, the valid data flag and the Module 4 segment route enable signals are cleared. The segment start single value is a signal that registers one cell of the start value input array. The data is used with in the subcomponent. On the rising edge of the component clock, the end of line flag is checked and the valid data flag is checked. If the end of line flag is a one, then the valid data flag and the Module 4 segment route enable are set to one. If the valid data flag is a one, then the array counter is checked and the segment start, middle, and end single values are assigned. If the array counter is equal to the maximum number of objects, then array counter

is set to one, the valid data flag is cleared, and the Module 4 write enable signal is cleared. If the array counter is not equal to the maximum number of objects, then the array counter is implemented by one. The segment start single value is assigned the value located at the array counter in the segment start array from Module 3. Similar assignments are made with the middle and end single values.

The new object process determines if there is a new object to begin tracking. The process is triggered on reset and on the rising edge of the component clock. On reset, the current object active signal is cleared and the object ready flag is cleared. On the rising edge of the component clock, the data valid flag is checked. If the data valid flag is one, then a for loop is implemented to determine if there is a new object. The for loop runs from one to the maximum number of objects minus one. In this case, the for loop ends at 63. The for loop counter is used to check different values that are stored in arrays. The first check is of the object done array. If the object done flag is a one, then the current object flag for the object is cleared and the object ready flag is cleared. If the object done flag is zero, then the current object active flag is checked. If the current object active flag is a one, then the current object active flag remains a one and the object ready flag is set. If the current object active flag is zero, then the process checks if two objects become active at the same time. This is determined by check the current active object flag at the current loop counter and the following loop counter along with the active object signal at the current loop counter and the following loop counter. If the current active object flags are zero and the active object signals are one, then a new object is created. The new object is created by setting the current object active flag at the current loop counter to one. The object ready flag is set to one. Otherwise, the current object active flag is not updated and the object ready flag is set to zero.

The Module 4 subcomponent, segment route, is the component that checks if the current data belong to an active object. If the data does not belong to an active object, a new object is created. The component implements a single process to check the new data. The process is triggered on reset and on the rising edge of the system clock. On reset, the data in object and the no match flags are cleared. On the rising edge of the component clock, the HPS enable flag is checked. If the enable flag is set, then the active object flag is checked. If the active object flag is a one, then the object done flag is checked. If the object done flag is a one, then the active object, the data in object, and the no match flags are cleared. Otherwise, the current line data is checked. If the current line data is zero, then the active object flag is registered and the data in object and the no match flags are cleared. If the current line data is between the last line start and end object values, then the active object flag is registered, the data in object flag is set, and the no match flag is cleared. If the object active flag is a zero, then the current line data is checked. If the current line data is zero, then there is no object and the active object, the data in object, and the no match flags are cleared. Otherwise, there is a new object and the active object flag is set, the data in object flag is cleared, and the no match flag is set.

The Module 4 Subcomponent is implemented using a generate statement to create a segment route component for each possible object to be routed. This allows the comparison of the current data to be run on every piece of data from the previous line in parallel. The input and output data is saved in arrays. The previous line input data is passed in using a single array, while the current line data is passed in serially. The output data from these subcomponents are passed to the data output process.

The no data process checks if there is new data in the current or previous pixel scan line. The process is triggered on reset and on the rising edge of the component clock. On reset, the no data flag is cleared. On the rising edge of the component

clock. The array data valid flag is checked. If the array data valid flag is a one, then the current line start, middle, and end values along with the last line start and end values are checked. If there is no data in these arrays, meaning all the cell in each array are zero, then the no data flag is set to a one. The no data flag is used to determine the end of an object. Otherwise, the no data flag is set to a zero.

The data valid process is triggered on reset and on the rising edge of the component clock. On reset, the data valid signal is cleared. On the rising edge of the component clock, the array data valid flag is checked. If the array data valid flag is a one, then the data valid flag is set to a one. The data valid flag is checked in the data output process.

The data output process serials the object data for use in Module 5 and in the creation of the Air Jet Packets. The data output process is triggered on reset and on the rising edge of the component clock. On reset, the Module 5 ready flag and the Module 4 write enable signals are cleared, the line scan data packet is set to zero, and the hyperspectral line counter is set to one. On the rising edge of the component clock, the data valid flag is checked. If the data valid flag is a one, then the process counter is checked. If the process counter is equal to the maximum number of objects, the process counter is reset to one and the object done flag is cleared. Otherwise the process counter is incremented by one. Also, the active object flag, the no match flag and the no data flag are checked. If the active object flag is a one, the no match flag is a zero, and the no data flag is a one, then the object done flag is set, the Module 4 write enable is cleared, and the line scan packet is created with the done condition. The done condition is used in Module 5 to trigger the computation of the vertical center of the object. Otherwise, the active object flag, the data in object flag, and the new object flag are checked. If the active object flag is a one and the data in object flag or the new object flag is a one, then Module 5 ready flag is set, the Module 4

write enable is set, and the Line Scan Data Packet is created. The Line Scan Packet is created by concatenating the object number with the line number with the start, middle, and end values of the object. The object number is the current value of the process counter number when the object is first created. If the active object and the object done flags are set to one, then the hyperspectral line counter is checked. If the hyperspectral line counter is five, then the hyperspectral line counter is set to one and the object done flag is cleared. Otherwise, the hyperspectral line counter is incremented by one and the object done flag is registered. If the active object and the object done flags are not set to one, then the active object and the object done flags are checked. If the active object flag is a one and the object done flag is a zero, then the line scan packet is set to zero and the Module 5 ready signal is set to zero. Otherwise, the Module 5 ready flag and the Module 4 write enable are cleared, the line scan data packet is set to zero, and the object done flag is cleared.

Simulation. Module 4 has been verified with one object and is currently being verified with multiple objects. The control state machines, Module 4 subcomponent, and the data output packet are being simulated for the correct signals and timing.

Hyperspectral Object Packet Creation

The Hyperspectral Object Packet is created by converting the object information from the line scan frame of reference to the hyperspectral frame of reference. The edges of the object are serially passed to the Camera Relations block that converts the information. The Camera Relation component was developed and tested by Monica Whitaker [4] before being implemented in the Line Scan Object Tracking Algorithm.

Implementation. The conversion from the line scan frame of reference to the hyperspectral frame of reference is completed by using a single process, triggered

on reset and on the rising edge of the component clock. The object information is passed to the component in a packet containing the object number, the line number, the start edge and the end edge. The start edge is the left most edge of the object and the end edge is the right most edge of the object. The line number is multiplied by 0.125, the number of line scan lines per hyperspectral line. The object edges are converted by multiplying by the line scan pixel to hyperspectral pixel ratio, 0.125. On reset, the hyperspectral packet is cleared. On the rising edge of the system clock, the hyperspectral packet is created. If the line reset flag is set, then the hyperspectral packet is given a specific bit array to signify a line reset. Otherwise, the edges of the object are checked to determine if the object is within the hyperspectral pixels. If the object starts before the hyperspectral pixels, then the object is ignored. If the object is after the hyperspectral region, the object is ignored. If the object is within the hyperspectral pixel region, the object is kept. The hyperspectral packet is created on every clock edge by concatenating the hyperspectral line number, a string of zeros, the object number, the hyperspectral start value and the hyperspectral end value. The packet is saved in a FIFO, which is read by the Arria 5 transceiver block.

Simulation. The component was simulated and verified using random MATLAB test data, before being verified with the results from Module 4.

Module 5 - Vertical Center Calculation

The object information from every line is saved in a memory block based on the object number and addressed by the line number. Module 5 computes the vertical center of the object by subtracting the last line number of the object from the first line number of the object and dividing by two. Every time a new line of data is passed into the component, the first line array indexed by the object number is checked. If the first line array cell is equal to all ones, then the value is saved in the first line

object. Otherwise, the new line data will be saved in the last line array. The last line array is updated on every clock cycle unless, the object edge information is equal to the object done condition. In this case, the intermediate flag is set, the last line minus first line subtraction is completed, and the object number is save.

When the intermediate flag is set, it triggers the component to compute the vertical center and to creates the vertical data object packet. This data packet contains the object number, the first line number, the center line number, and the last line number. The center of the line is computed by shifting the subtraction result from the first line and the last line right by one bit and adding the result to the first line number. The Module 5 write enable bit is also set during this step.

Implementation. Module 5 was implemented using two arrays and two processes. The arrays store the first line and the last line information based on the object number. The arrays contain one cell for every object that could be tracked because multiple objects could be tracked at once. The cells in the first line array are initialized to FFFFFFFF while the cells in the last line array are initialized to zero.

The first process stores the line data for the object and when the object is complete, computes the subtraction of the line information. The process is triggered on reset and the rising edge of the component clock. On reset, the arrays are reinitialized to their starting values. On the rising edge of the component clock, the process checks for the object done condition. If this condition is met, then the process sets the intermediate flag to a one, computes the subtraction, and stores the current line number. If the end object condition is not met, then the process checks the value of the first line array at the object number. If the value of the array is equal to FFFFFFFF, then the current line number is saved to this location. Otherwise, the

current line number is saved to the last line array. If the Module 5 write enable flag, set in the second process, is a one, then the intermediate flag is cleared.

The second process computes the vertical center of the object, creates the vertical object packet, and sets the Module 5 write enable signal. The process is triggered on reset and on the rising edge of the component clock. On reset, the vertical object packet and the Module 5 write enable signal are cleared. On the rising edge of the component clock and if the intermediate flag is set, then the vertical object packet is created and the Module 5 write enable signal is set to a one. The vertical object packet is created by concatenating the object number, the first line number, the center line number, and the last line number. The center line number is created by shifting the subtraction result by one bit and adding the result with the first line value. Otherwise, the vertical object packet is cleared and the Module 5 write enable is set to zero.

Simulation. Module 5 was verified with a single object using the results from Module 4 as the inputs to the component. The system computed the center of the object which was confirmed by computing the vertical center result by hand. The last line value was off by a single line, but because of rounding error, the vertical center computed was correctly.

Air Jet Object Packet and Interfacing with the Air Jets

The Air Jet Object Packet contains the object number, the horizontal center, and the vertical center of the object. This information is passed to the Air Jet subsystem through a FIFO. The air jet write enable signal triggers the current air jet packet to be written to the FIFO. The Air Jet subsystem then reads the object information and tracks the object before turning on the correct air jet to eject the object.

Implementation. The Air Jet Object Packet is created within a process in the Object Tracking component. The Air Jet Packet Process is triggered on the reset and the rising edge of the component clock. On reset, the Air Jet Packet and the air jet write enable signal are cleared. On the rising edge of the component clock, the Module 5 write enable signal is checked. If the write enable signal is a one, then the Air Jet Object Packet is created by combining the object number, the horizontal center, and the vertical center of the object. The object number and the vertical center come from the vertical object packet, the output of Module 5. The horizontal center is read from the memory block using the object number and the vertical center line number. The object number selects the correct memory block and the vertical line number address the memory block. The air jet packet is then sent to the top level along with the air jet write enable that is set. If the write enable is zero, the Air Jet Packet and the air jet write enable signal are cleared.

Simulation. The Air Jet Object Packet was verified as a part of the complete Object Tracking Algorithm simulation. The simulation started by inputting data into Module 2 and ended with the verification of the Air Jet Object Packet.

PROTOTYPE INTEGRATION

With the Hyperspectral System and the Object Tracking System verified, the two systems were combined to create the single embedded solution using the Arria 10 System on Module FPGA. This integration requires a custom printed circuit board to connect the image sensor to the Arria 10 System on Module as well as a custom printed circuit board to connect to the line scan camera. The processing algorithms have been combined on the Arria 10, eliminating the need for the transceivers to pass data between the computation systems. The custom sorting system will then be integrated with the Air Jet System.

Arria 10 SoM PCB Design

The embedded sorting system is created by joining the Arria 10 System on Module with three custom printed circuit boards. The first printed circuit board will join the image sensor to the FPGA using two FMC connectors. The second board is the Camera Link board which will connect the line scan camera to the FPGA through one FMC connector and the Camera Link protocol. The final printed circuit will provide power for the entire system, specifically the image sensor, the Arria 10 FPGA, and the line scan camera as well as the JTAG interface along with other connections to connect to and program the FPGA.

Arria 10 SoM Hyperspectral Image Sensor Daughter Card

While the image sensor board is under verification, the design work for the Arria 10 SoM Hyperspectral Image Sensor Daughter Card can begin. The first step is to create the schematic for the Arria 10 SoM Sensor Board by combining the Prototype Sensor Board and the FMC Prototype Board. The data signals were first sketched

with the goal of reducing the trace length. The data signals travel from the image sensor to a voltage translator, where the data signals are translated from 3.3V to 1.8V. The 1.8V signals enters one of two FMC connectors before heading to the FPGA. The control signal travel from the FPGA through the FMC connect to a voltage translator to go from 1.8V to 3.3V before entering the sensor. Some attention was paid to what signals were next to each other on the connector but this was not the driving factor for the design.

With the data and control signals defined, the next step is to determine the power circuitry. The sensor needs analog and digital +3.3V rails, the level translators need digital 1.8V and 3.3V volts. The SoM FMC connectors provide 12V power rails to the connected boards. Instead of designing power regulators on the sensor board, power is provided through a six-pin surface mount header. The header pin provides three ground connections, a digital +3.3V power rail, a digital +1.8V power rail, and an analog +3.3V power rail. The analog will not be connected initially. Instead filtering circuitry on the sensor board will filter the digital +3.3V power rail to an analog +3.3V power rail. The power header will be connected to a dedicated power board in the stack. The power board will provide power for the Arria 10 SoM, Hyperspectral Image Sensor, the CameraLink board and power for the line scan camera.

With the schematic completed, the components were placed on the board. First, the FMC connectors were placed on the top layer and locked. This ensured that the SoM Sensor Board will mate with the Arria 10 SoM Board correctly. With the FMC connectors in place, the image sensor was placed in the center of the board on the bottom layer. This is necessary for correct component population during assembly. After the sensor is in place, the level translators, capacitors, resistors, and potentiometers are placed on the board. Figure 4.1 shows the unrouted SoM Sensor Board highlighting the location of the surface mount components.

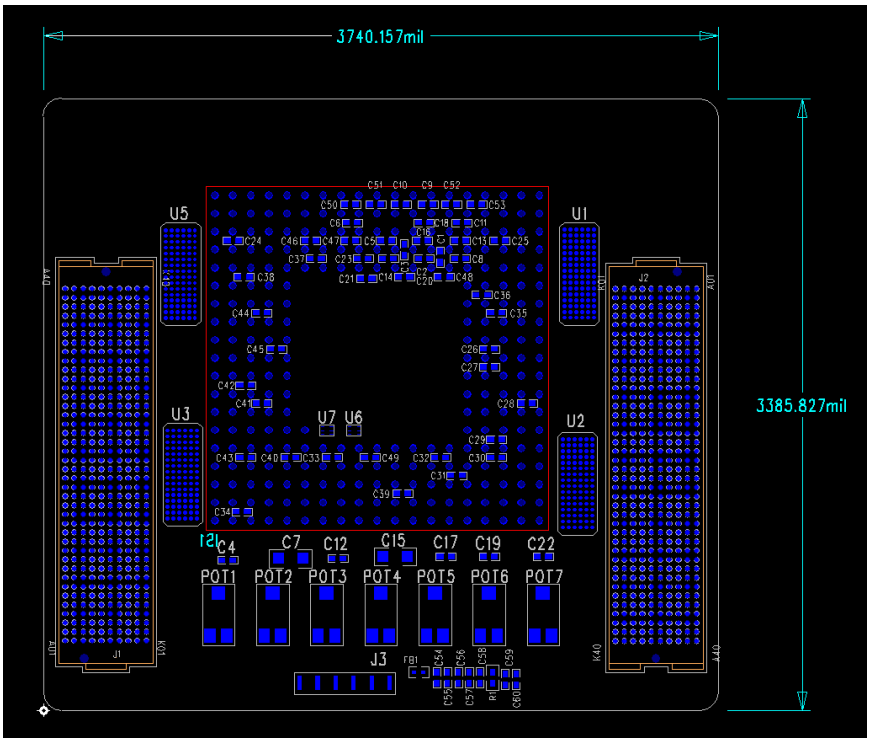


Figure 4.1: The Unrouted SoM Sensor Board showing the location of the FMC connectors, the image sensor, the surface mount parts.

With the components placed on the board, the next step is to determine the number of layers and to route the traces. The sensor board is still under development while the Camera Link Board and the Power Board are still in the design phase. Along with the custom print circuit boards, the algorithms need to be combined into a single solution.

Algorithm Integration

The Hyperspectral Processing Component was combined with the Object Tracking Component, to create the single processing solution. The integration was completed by combining the top-level files of the Computation System, the Hyperspectral Camera Interface, the Object Tracking System, and the Line Scan

Camera Interface. The challenge was to ensure that the correct signals were passed to the different components. By moving the Object Tracking System to the Arria 10, the transceiver blocks were removed from the computation project. The entire system is under development and will be verified as part of the continuing work on the project, once the user interface has been implemented using the HPS. With the printed circuit boards designed and the algorithms integrated on to a single FPGA, the next step is to integrate the sensors and the computation system with the physical sorting system.

Physical Setup

The following section discusses the physical setup, including the feed system, the air jet subsystem, and the camera location.

Figure 4.2 shows the prototype commercial sorting system. Starting from the top moving down, the product to be sorted is placed in the hopper. The hopper then feeds the vibrating tray, where the height of the hopper feed tube can be adjusted based on the product being sorted. For this system, the feed tube is set to distribute the lentils in a single layer. The single layer is necessary so that no lentils overlap during scanning.

As the lentils are moving down the vibrating tray, eventually they fall of the end and are imaged by the hyperspectal and line scan cameras. The Basler Monochrome Line Scan camera produces a maximum of 8,192 pixels at 80,000 lines per second. The line scan camera has been scaled down to 2,048 pixels at 80,000 lines per second. The camera is set to run in an eight-bit ten-pixel configuration. The line scan camera is connected to the FPGA via the Camera Link Interface. The hyperspectral imager is built on using a custom image sensor and a lens built by Resonon Inc. The sensor produces 1,020 pixels at 500 frames per second. The area of interest for the

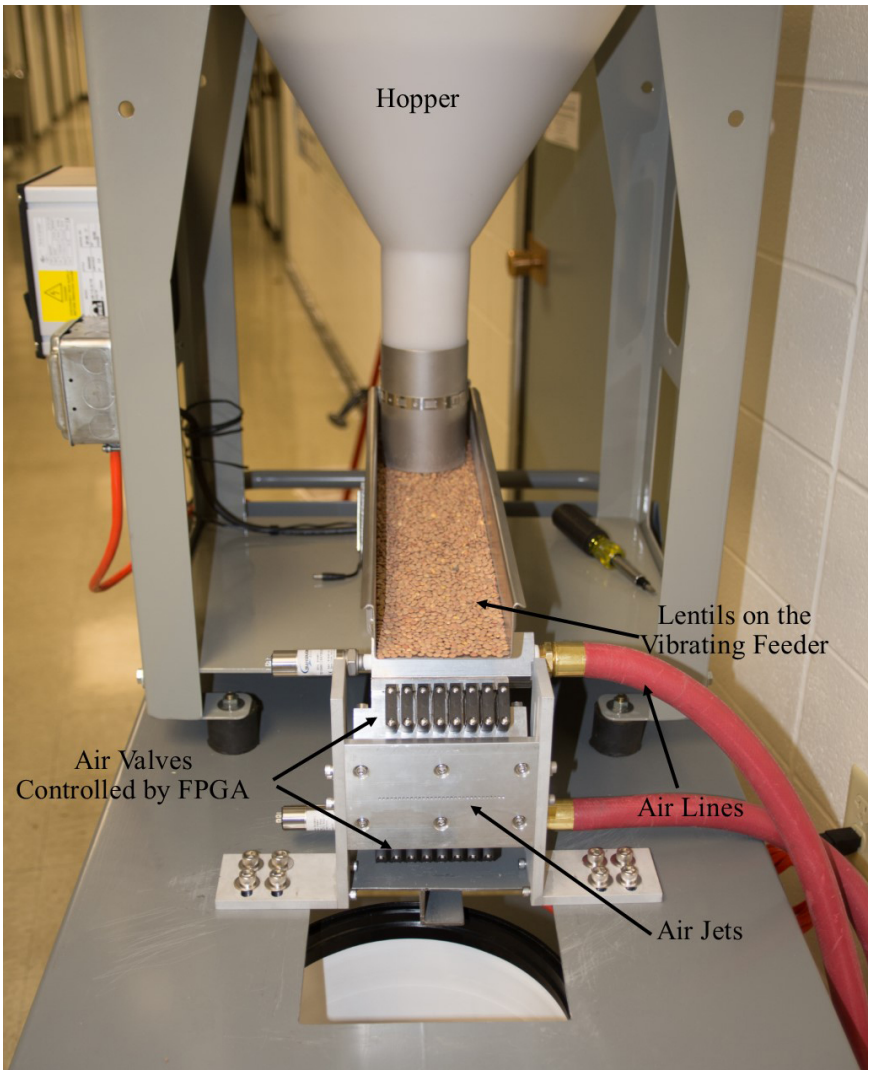


Figure 4.2: This figure shows the set up of the prototype commercial sorting system including the hopper, the vibrating tray, the air valves, the air jets and the air lines.

hyperspectral imager is 250 pixels at 2000 frames per second. The ten-bit pixel data is passed to the FPGA over 100 single-ended data lines which enter the Arria 10 System on Module using two high pin count FMC connectors. The cameras are located across from the falling lentils.

The cameras image the same line of pixel data. Figure 4.3 shows the pixel line scanned by both cameras. The data is sent to the FPGA for processing before a

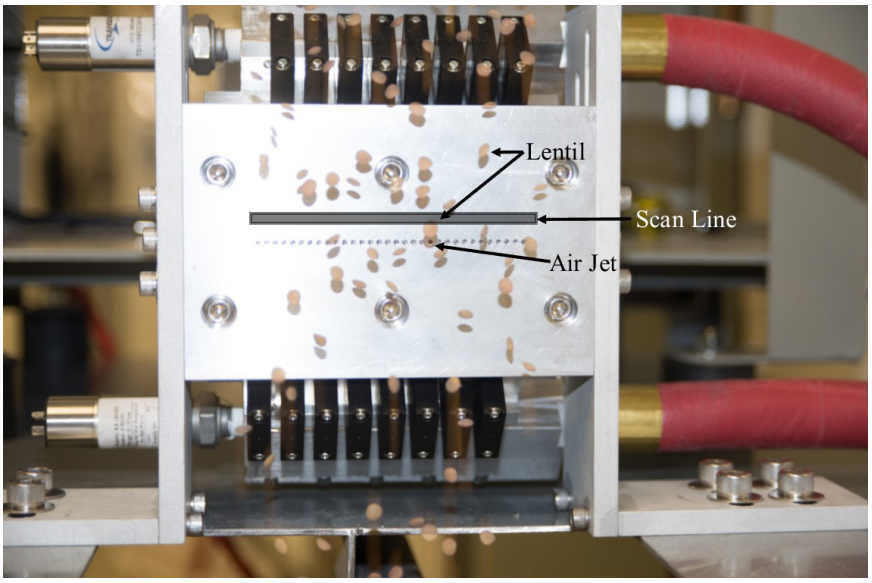


Figure 4.3: A view from the location of the hyperspectral and line scan camera location.

decision is made on whether to keep or reject the lentil. There are 32 air jets spread over the four-inch wide vibrating tray. The air jets are controlled by information from the Object Tracking System and a custom timing system being developed by the Air Jet Sorting Senior Design Team.

As the research moves toward commercialization, there is still development that needs to be completed.

Continuing Development

To reach commercialization, a complete system needs to be verified. This will require a user interface as well as integration with Resonons optical system. The user interface will be implemented on the Linux operating system running on the HPS and will provide access to the control and status registers of the different subsystem. With the user interface and the optical system, the sorting system can be trained to

sort good product from bad product. With the training complete, the system will then undergo final device testing.

CONCLUSION

As the system moves to a commercial product, this thesis focused on the integration of the CMOS image sensor for the high speed hyperspectral imager as well as the development of the object tracking algorithm. The CMOS image sensor combined with the FPGA will become a stand alone high speed hyperspectral imager for Resonon. However, as shown with the former years on the project and the remain functionality and integration that needs to be completed, this is one step in the development of the Smart Camera Sorting System.

REFERENCES CITED

[1] W. Wolf, B. Ozer, and T. Lv, "Smart cameras as embedded systems," *IEEE Computer*, vol. 35, pp. 48–53, 2002.

[2] M. Lesser, S. Miller, and Y. Haiqian, "Smart camera based on reconfigurable hardware enables diverse real-time applications," *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, pp. 147–155, 2004.

[3] F. Dias, F. Berry, and J. Serot, "Hardware, design and implementation issues on a fpga-based smart camera," *Distriuted Smart Cameras, First ACM/IEEE International Conference*, pp. 20–26, 2007.

[4] M. Whitaker, "Development of a smart camera system on an fpga," Montana State University, 2015.

[5] (2015) Flyer for the arria 10 system on module fpga. Reflex CES. [Online]. Available: https://rocketboards.org/foswiki/pub/Documentation/A10ACHILLESInstantDevKitArria10SOMStarterBoard/Flyer%20Achilles.pdf

[6] G. Lokman and G. Yilmaz, "Hyperspectral image classification using support vector neural network algorithm," pp. 239–243, 2015.

[7] "What is spectral imaging and when should i use it?" White Paper, Resonon.

[8] (2015) Mt9m413: 1.3mp cmos digital image sensor features. Aptina Imaging. [Online]. Available: https://media.digikey.com/pdf/Data%20Sheets/APTINA%20PDFs/MT9M413_DS.pdf

[9] (2008) Ansi/vita 57.1 fpga mezzanine card (fmc) standard. VITA. [Online]. Available: http://www.vita.com/fmc

[10] (2009) Iso 10935:2009. International Organization of Standardization. [Online]. Available: https://www.iso.org/standard/51305.html

[11] (2008) Basler racer user's manual for camera link cameras. Basler. [Online]. Available: http://www.baslerweb.com/en/products/cameras/line-scan-cameras/racer/ral8192-80km

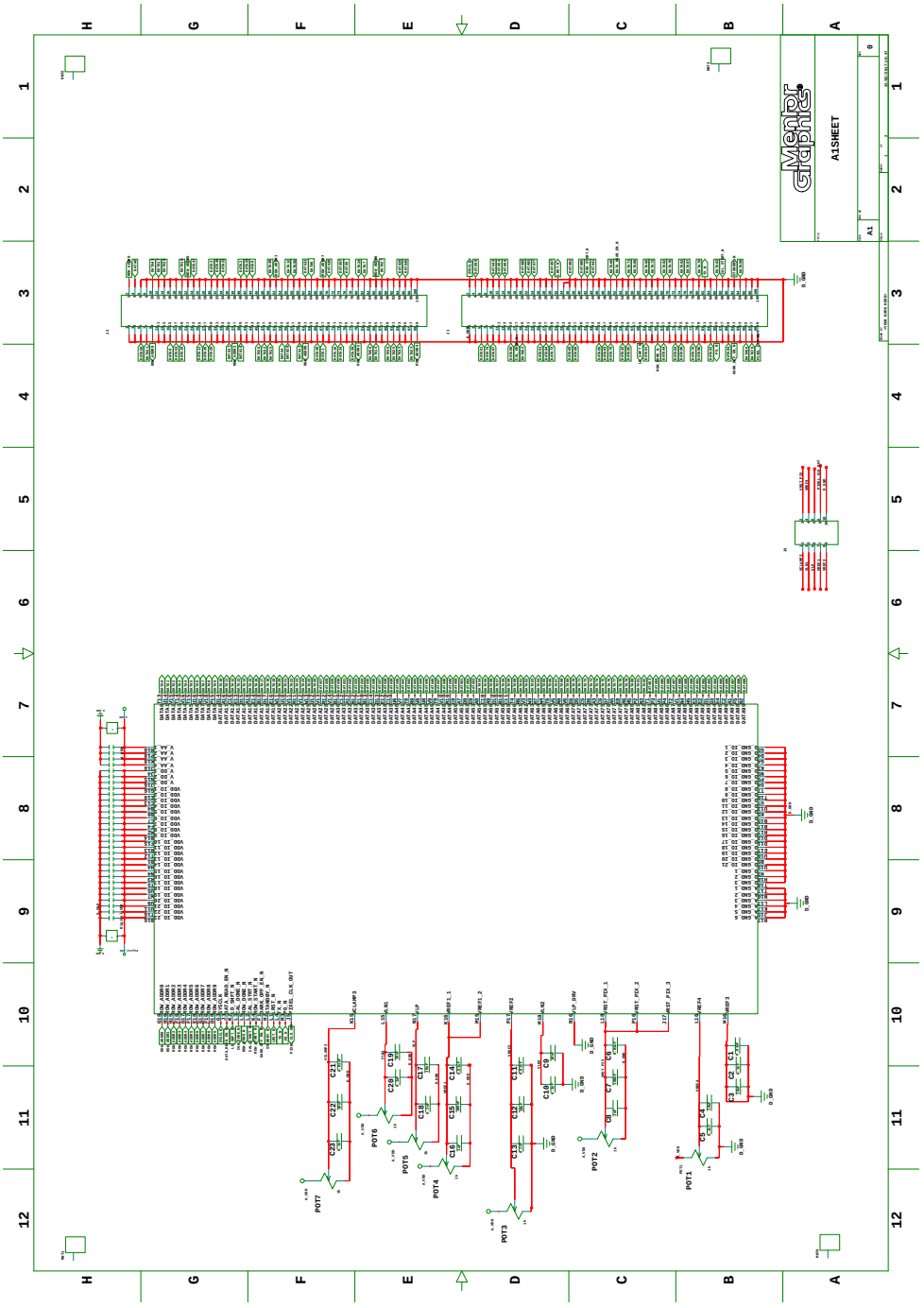
[12] "Camera link technology brief," Basler Vision Technologies, 2001.

APPENDICES

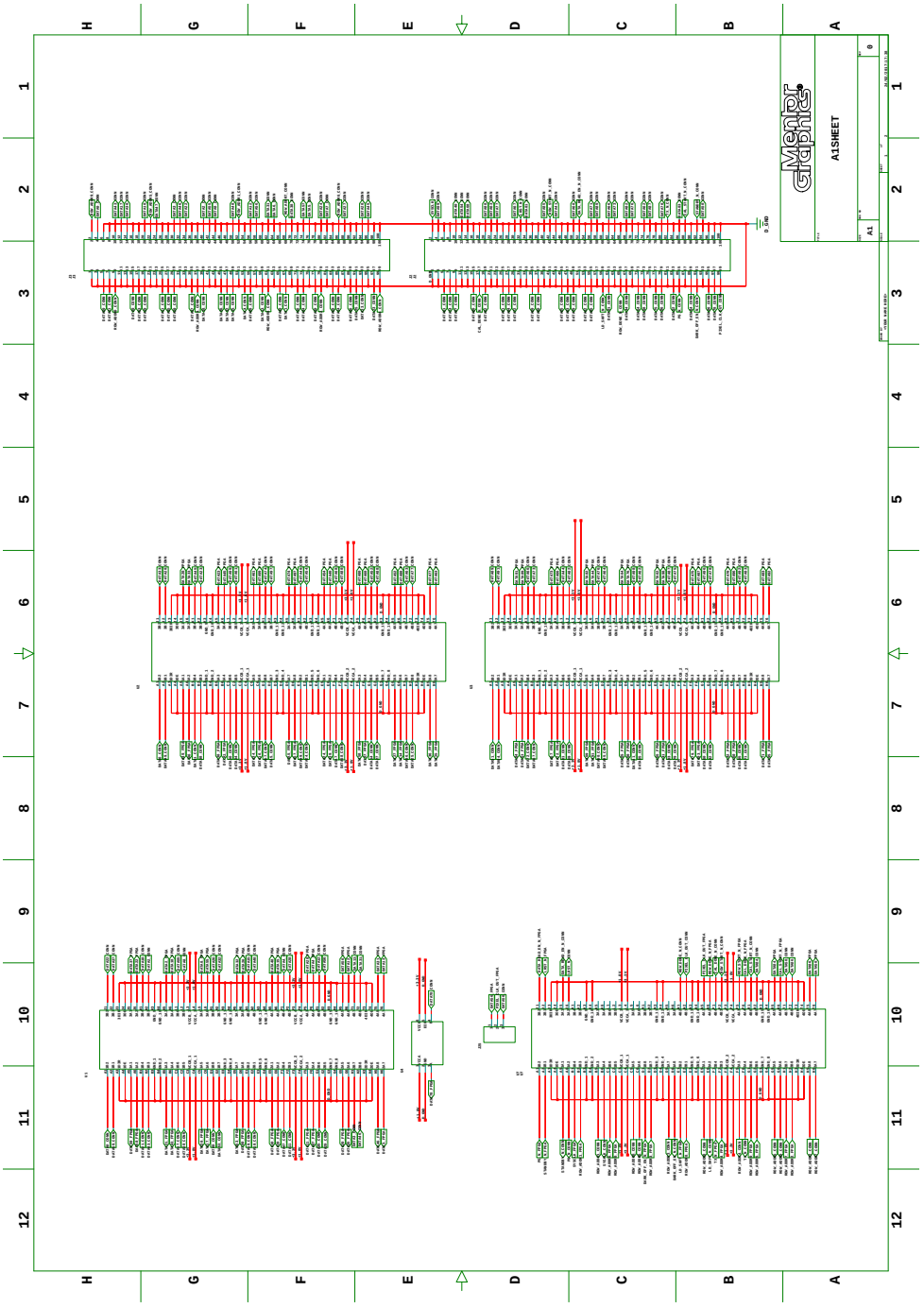
APPENDIX A

PROTOTYPE IMAGE SENSOR

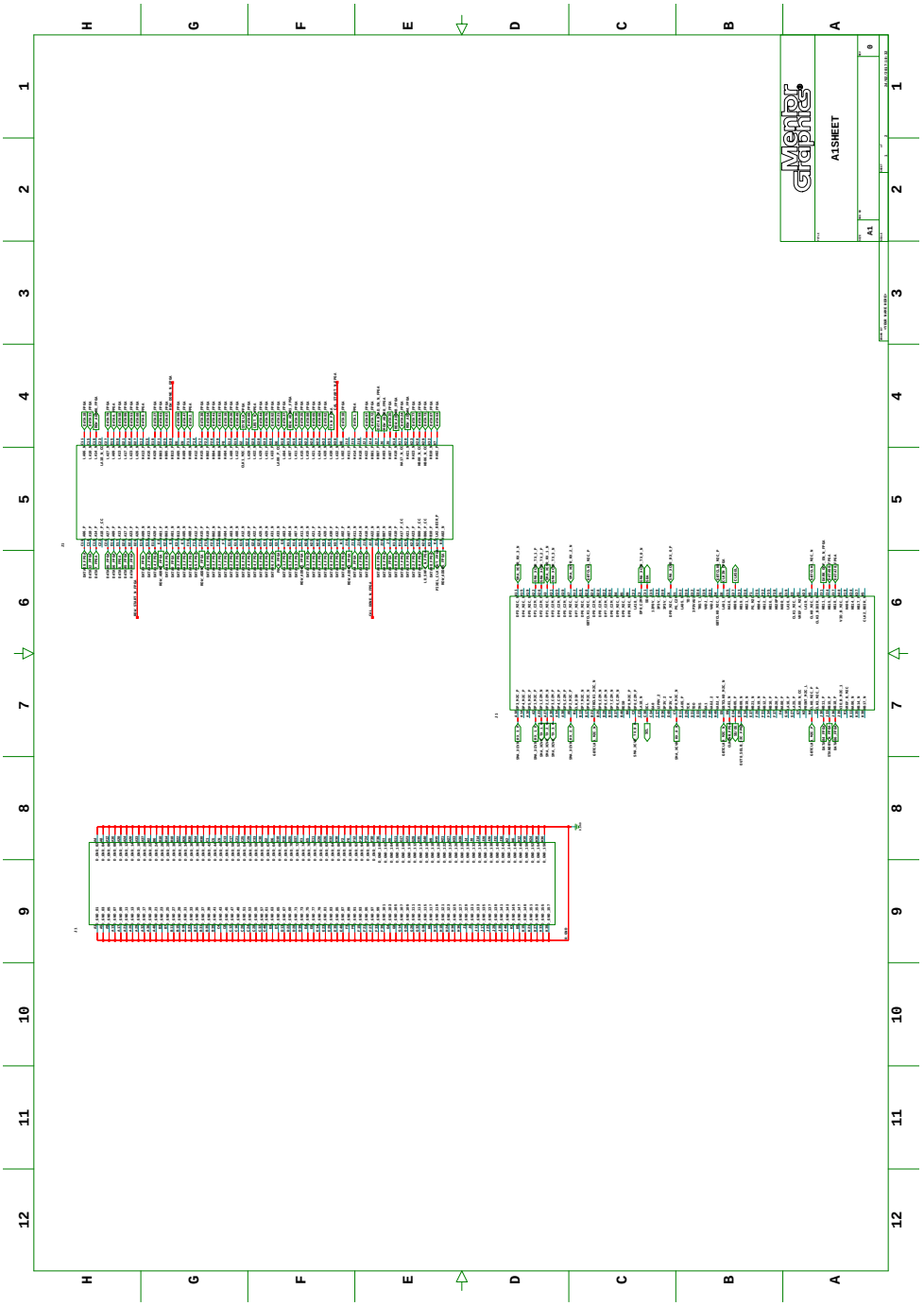
Prototype Image Sensor Board Schematic



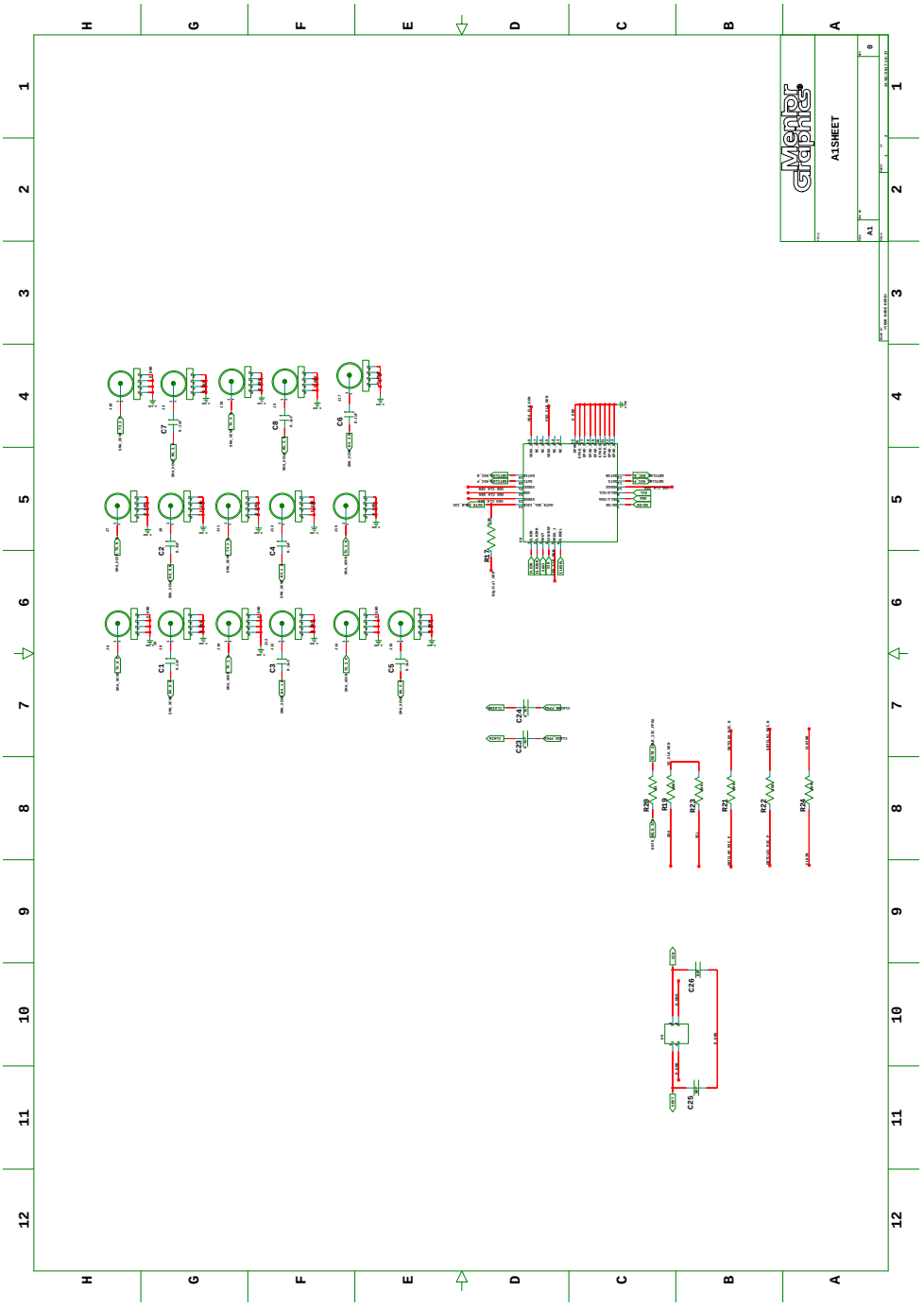
Prototype FMC Board



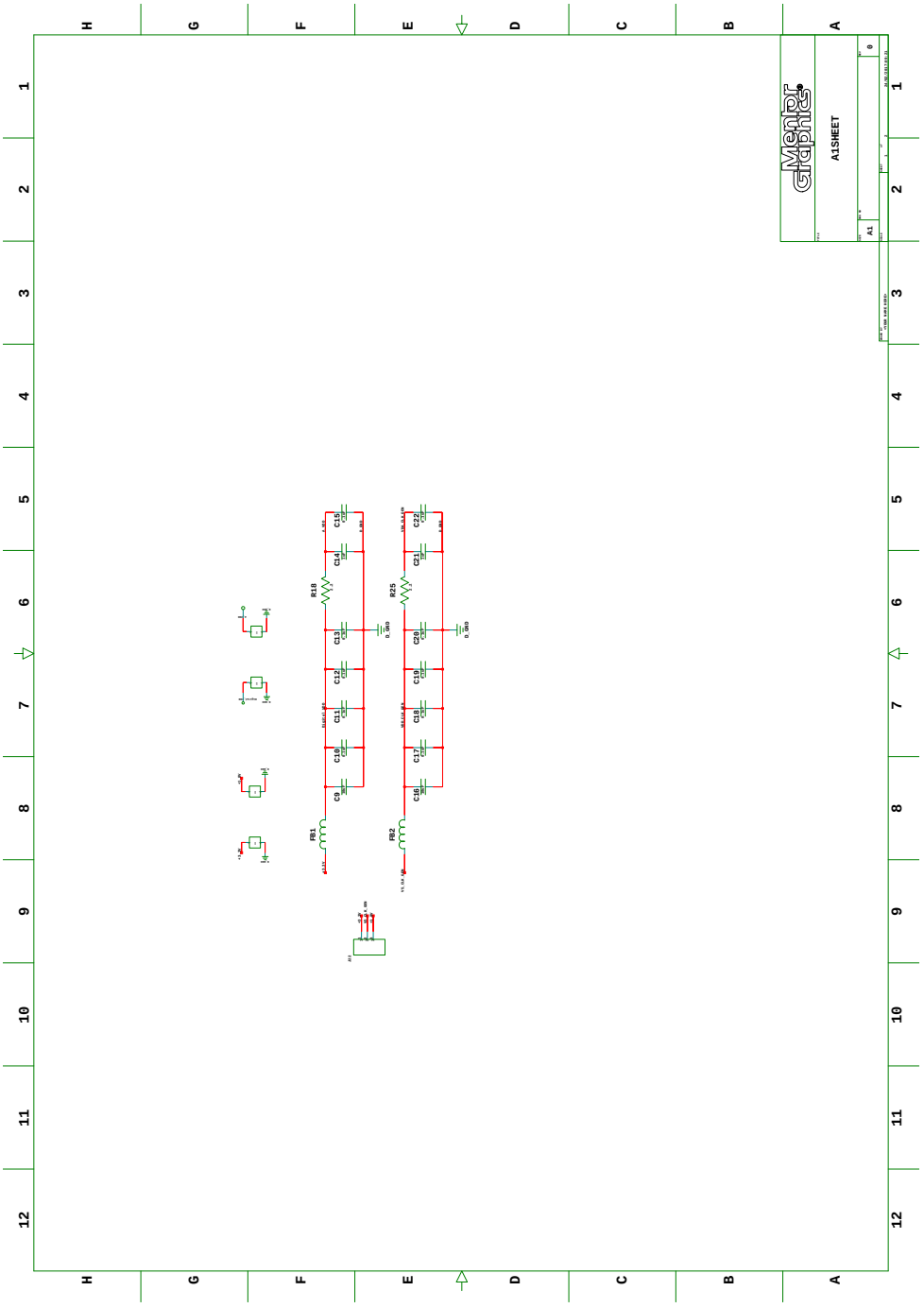
Prototype FMC Board



Prototype FMC Board



Prototype FMC Board



Prototype Image Sensor Register Descriptions

Table A.1: Number of Clock Cycles for Exposure Register Description

	MSB																													LSB
Bits	31			28	27			24	23			20	19			16	15			12	11			8	7			4	3			0
R/W	-	-	-	-	-	-	-	-	-	-	-	-	-	-	**I**	**I**	**I**	**I**	**I**	**I**	**I**	**I**	**I**	**I**	**I**	**I**	**I**	**I**	**I**	**I**	**I**	**I**
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table A.2: Row Number Exposure Register Description

	MSB																													LSB
Bits	31			28	27			24	23			20	19			16	15			12	11			8	7			4	3			0
R/W	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	**I**	**I**	**I**	**I**	**I**	**I**	**I**	**I**
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table A.3: Command Register Description

	MSB																													LSB
Bits	31			28	27			24	23			20	19			16	15			12	11			8	7			4	3			0
R/W	**I**	**I**	**I**	**I**	**I**	**I**	**I**	**I**	**I**	**I**	**I**	**I**	**I**	**I**	**I**	**I**	**I**	**I**	**I**	**I**	**I**	**I**	**I**	**I**	**I**	**I**	**I**	**I**	**I**	**I**	**I**	**I**
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table A.4: Status Register Description

	MSB																										LSB	
Bits	31			28	27			24	23			20	19			16	15			12	11			8	7			4	3		0
R/W	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	**I**	**I**
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Prototype Image Sensor VHDL Code

```
1  ----------------------------------------------------------------------------
2  --
3  --! @file        top.vhd
4  --! @brief       Top Level Design File used to test the camera
5  --               interface with the DRAM
6  --
7  --! @author      Connor Dack
8  --! @date        May 2015
9  --! @copyright   Copyright (C) 2015 Ross K. Snider and Connor Dack
10 --
11 --   This program is free software: you can redistribute it
12 --   and/or modify it under the terms of the GNU General
13 --   Public License as published by the Free Software Foundation,
14 --   either version 3 of the License, or (at your option) any
15 --   later version.
16 --
17 --   This program is distributed in the hope that it will be
18 --   useful, but WITHOUT ANY WARRANTY; without even the implied
19 --   warranty of MERCHANTABILITY or FITNESS for A PARTICULAR
20 --   PURPOSE. See the GNU General Public License for more details.
21 --
22 --   You should have received a copy of the GNU General Public
23 --   License along with this program.  If not, see
24 --   <http://www.gnu.org/licenses/>.
25 --
26 --   Connor Dack
27 --   Electrical and Computer Engineering
28 --   Montana State University
29 --   610 Cobleigh Hall
30 --   Bozeman, MT 59717
31 --   connor.dack@msu.montana.edu
32 --
33 ----------------------------------------------------------------------------
34
35 library IEEE;
36 use IEEE.STD_LOGIC_1164.ALL;
37 use IEEE.NUMERIC_STD.ALL;
```

```
38
39 use work.Sensor_Package.ALL;
40
41 entity top is
42     port(reset_n          : in std_logic;
43          clk_100mhz_1     : in std_logic;
44          dram_clk         : in std_logic;
45          data_out_0 : out std_logic_vector (SUPER_PIXEL_SIZE -1
46                                                    downto 0);
47          data_out_1 : out std_logic_vector (SUPER_PIXEL_SIZE -1
48                                                    downto 0);
49          data_out_2 : out std_logic_vector (SUPER_PIXEL_SIZE -1
50                                                    downto 0);
51          data_out_3 : out std_logic_vector (SUPER_PIXEL_SIZE -1
52                                                    downto 0);
53          data_out_4 : out std_logic_vector (SUPER_PIXEL_SIZE -1
54                                                  downto 0));
55 end top;
56
57 architecture rtl of top is
58
59     component camera_preprocessing is
60         port(rst_n              : in std_logic;
61              component_clk      : in std_logic;
62              dram_clk           : in std_logic;
63              hps_enable_flag    : in std_logic;
64              data_0_in          : in std_logic_vector(
65                                   INPUT_DATA_SIZE  - 1 downto 0);
66              data_1_in          : in std_logic_vector(
67                                   INPUT_DATA_SIZE - 1 downto 0);
68              data_2_in          : in std_logic_vector(
69                                   INPUT_DATA_SIZE - 1 downto 0);
70              data_3_in          : in std_logic_vector(
71                                   INPUT_DATA_SIZE - 1 downto 0);
72              data_4_in          : in std_logic_vector(
73                                   INPUT_DATA_SIZE - 1 downto 0);
74              data_5_in          : in std_logic_vector(
75                                   INPUT_DATA_SIZE - 1 downto 0);
76              data_6_in          : in std_logic_vector(
77                                   INPUT_DATA_SIZE - 1 downto 0);
78              data_7_in          : in std_logic_vector(
79                                   INPUT_DATA_SIZE - 1 downto 0);
80              data_8_in          : in std_logic_vector(
81                                   INPUT_DATA_SIZE - 1 downto 0);
82              data_9_in          : in std_logic_vector(
83                                   INPUT_DATA_SIZE - 1 downto 0);
84              buffer_empty_in    : in std_logic;
85              buffer_read_out    : out std_logic;
```

```
86              buffer_readdata_in : in std_logic_vector(
87                                       DRAM_DATA_SIZE-1 downto 0);
88              data_packet_out    : out std_logic_vector(((
89                                       SUPER_PIXEL_SIZE *
90                                       NUMBER_OF_PARALLEL_CHANNELS)
91                                       - 1) downto 0);
92              pixel_data_and_location_valid_flag_out :
93                                                    out std_logic;
94              super_pixel_valid_flag_out : out std_logic;
95              clear_pixel_flag_out       : out std_logic;
96              new_frame_flag_out         : out std_logic);
97      end component;
98
99      component dual_clock_fifo is
100         generic(
101             enable_ecc             : string := "FALSE";
102             lpm_numwords           : natural;
103             lpm_width              : natural;
104             lpm_widthu             : natural;
105             rdsync_delaypipe       : natural;
106             wrsync_delaypipe       : natural);
107         port (
108             data       : in  std_logic_vector(lpm_width - 1 downto
109                           0) := (others => 'X'); -- datain
110             wrreq      : in  std_logic := 'X'; -- wrreq
111             rdreq      : in  std_logic := 'X'; -- rdreq
112             wrclk      : in  std_logic := 'X'; -- wrclk
113             rdclk      : in  std_logic := 'X'; -- rdclk
114             aclr       : in std_logic  := '0';
115             q          : out std_logic_vector(lpm_width - 1 downto
116                                       0); -- dataout
117             rdempty    : out std_logic; -- rdempty
118             wrfull     : out std_logic; -- wrfull
119             rdfull     : out std_logic;
120             wrempty    : out std_logic;
121             rdusedw    : out std_logic_vector(lpm_widthu - 1
122                                                   downto 0);
123             wrusedw    : out std_logic_vector(lpm_widthu - 1
124                                                   downto 0);
125             eccstatus : out std_logic_vector(1 downto 0));
126     end component dual_clock_fifo;
127
128     component memory_block is
129         generic (
130             num_elements_a  : natural;
131             num_elements_b  : natural;
132             size_address_a  : natural;
133             size_address_b  : natural;
```

```
134             size_word_a     : natural;
135             size_word_b     : natural;
136             mem_init        : string := "UNUSED"
137         );
138         port (
139             address_a : in  std_logic_vector(size_address_a-1
140                                                 downto 0);
141             address_b : in  std_logic_vector(size_address_b-1
142                                                 downto 0);
143             clock_a   : in  std_logic  := '1';
144             clock_b   : in  std_logic;
145             data_a    : in  std_logic_vector(size_word_a-1
146                                                 downto 0);
147             data_b    : in  std_logic_vector(size_word_b-1
148                                                 downto 0);
149             wren_a    : in  std_logic  := '0';
150             wren_b    : in  std_logic  := '0';
151             q_a       : out std_logic_vector(size_word_a-1
152                                                 downto 0);
153             q_b       : out std_logic_vector(size_word_b-1
154                                                 downto 0)
155         );
156     end component memory_block;
157
158     constant data_packet_3_end : natural := SUPER_PIXEL_SIZE *
159                                     NUMBER_OF_PARALLEL_CHANNELS
160                                     - SUPER_PIXEL_SIZE;
161     constant data_packet_2_end : natural := SUPER_PIXEL_SIZE *
162                                     NUMBER_OF_PARALLEL_CHANNELS
163                                     - (2 * SUPER_PIXEL_SIZE);
164     constant data_packet_1_end : natural := SUPER_PIXEL_SIZE *
165                                     NUMBER_OF_PARALLEL_CHANNELS
166                                     - (3 * SUPER_PIXEL_SIZE);
167     constant data_packet_0_end : natural := SUPER_PIXEL_SIZE *
168                                     NUMBER_OF_PARALLEL_CHANNELS
169                                     - (4 * SUPER_PIXEL_SIZE);
170
171     constant pixel_condition : unsigned (DRAM_LOCATION_SIZE- 1
172                     downto 0) :=  to_unsigned((NUMBER_OF_PIXELS)
173                     - 1 , DRAM_LOCATION_SIZE);
174     constant spectral_bin_condition: unsigned (DRAM_LOCATION_SIZE
175                     - 1 downto 0) := to_unsigned(
176                     NUMBER_OF_SPECTRAL_BINS, DRAM_LOCATION_SIZE);
177
178     constant pixel_counter_zeros : std_logic_vector(
179                                 DRAM_LOCATION_SIZE - 1 downto 0)
180                                 := (others => '0');
181     constant spectral_bin_counter_zeros : unsigned(
```

```
182                                     DRAM_LOCATION_SIZE - 1 downto 0)
183                                     := (others => '0');
184
185     signal address : std_logic_vector (7 downto 0) := "00000000";
186
187     -- Signals for Simulating the DRAM by writing to the FIFO
188     signal dram_test_data : std_logic_vector(DRAM_DATA_SIZE - 1
189                                     downto 0) := (others => '0');
190     signal wrreq_signal : std_logic := '0';
191     signal wrfull_signal : std_logic := '0';
192     signal delay_one_cycle_flag : std_logic := '1';
193     signal light_data : std_logic_vector(LIGHT_CORRECT_SIZE - 1
194                                     downto 0) := (others => '0');
195     signal dark_data  : std_logic_vector(DARK_CORRECT_SIZE  - 1
196                                     downto 0) := (others => '1');
197
198     -- Pixel Location Counting Signals
199     signal pixel_counter : unsigned (DRAM_LOCATION_SIZE - 1
200                                    downto 0) := pixel_condition;
201     signal spectral_bin_counter : unsigned (DRAM_LOCATION_SIZE -
202                                     1 downto 0):=
203                                     spectral_bin_condition;
204
205     -- Camera Clock
206     signal camera_interface_clk : std_logic;
207
208     -- Enables and Flags
209     signal new_frame_flag           : std_logic := '0';
210     signal super_pixel_valid_flag   : std_logic := '0';
211     signal pixel_data_and_location_valid_flag_signal : std_logic
212                                                      := '0';
213     signal clear_pixel_flag_signal : std_logic := '0';
214     signal hps_enable_flag          : std_logic := '0';
215
216     -- DRAM FIFO Signals
217     signal s_buffer_empty     : std_logic;
218     signal s_buffer_read      : std_logic;
219     signal s_buffer_readdata : std_logic_vector(
220                                     DRAM_DATA_SIZE - 1 downto 0)
221                                     := (others => '0');
222
223     -- Super Pixel FIFO Signals -- will change names upon
224     -- combination of projects
225     signal data_packet_out     : std_logic_vector(((
226                                     SUPER_PIXEL_SIZE *
227                                     NUMBER_OF_PARALLEL_CHANNELS) -
228                                     1) downto 0) := (others => '0');
229     signal data_to_computation: std_logic_vector(((
```

```
230                                         SUPER_PIXEL_SIZE *
231                                         NUMBER_OF_PARALLEL_CHANNELS)
232                                         - 1) downto 0);
233     signal super_pixel_rdreq  : std_logic := '0';
234     signal super_pixel_full   : std_logic := '0';
235     signal super_pixel_empty  : std_logic := '0';
236
237     -- Input signals from on-chip memory
238     signal input_pixel_data_0 : std_logic_vector(INPUT_DATA_SIZE
239                                                  - 1 downto 0) :=
240                                                  (others => '0');
241     signal input_pixel_data_1 : std_logic_vector(INPUT_DATA_SIZE
242                                                  - 1 downto 0) :=
243                                                  (others => '0');
244     signal input_pixel_data_2 : std_logic_vector(INPUT_DATA_SIZE
245                                                  - 1 downto 0) :=
246                                                  (others => '0');
247     signal input_pixel_data_3 : std_logic_vector(INPUT_DATA_SIZE
248                                                  - 1 downto 0) :=
249                                                  (others => '0');
250     signal input_pixel_data_4 : std_logic_vector(INPUT_DATA_SIZE
251                                                  - 1 downto 0) :=
252                                                  (others => '0');
253     signal input_pixel_data_5 : std_logic_vector(INPUT_DATA_SIZE
254                                                  - 1 downto 0) :=
255                                                  (others => '0');
256     signal input_pixel_data_6 : std_logic_vector(INPUT_DATA_SIZE
257                                                  - 1 downto 0) :=
258                                                  (others => '0');
259     signal input_pixel_data_7 : std_logic_vector(INPUT_DATA_SIZE
260                                                  - 1 downto 0) :=
261                                                  (others => '0');
262     signal input_pixel_data_8 : std_logic_vector(INPUT_DATA_SIZE
263                                                  - 1 downto 0) :=
264                                                  (others => '0');
265     signal input_pixel_data_9 : std_logic_vector(INPUT_DATA_SIZE
266                                                  - 1 downto 0) :=
267                                                  (others => '0');
268
269     constant zeros : std_logic_vector(INPUT_DATA_SIZE - 1 downto
270                                       0) := (others => '0');
271     signal wrusedw : std_logic_vector(10 - 1 downto 0);
272
273   begin
274
275     -- TESTING ONLY - REMOVE WHEN BUILDING COMPLETE PROJECT------
276     -- Address Loop to change the address and to store read data
277      address_randomization : process (clk_100mhz_1, reset_n)
```

```
278         begin
279             if(reset_n = '0') then
280                 address <= "00000000";
281             elsif(rising_edge(clk_100mhz_1)) then
282                 if(hps_enable_flag = '1') then
283                     if(address = "11111111") then
284                         address <= "00000000";
285                     else
286                         address <= std_logic_vector(unsigned(
287                                                     address) + 1);
288                     end if;
289                 end if;
290             end if;
291         end process;
292 
293      camera_interface_clk <= clk_100mhz_1;
294 
295         dram_fifo_for_location_check : component dual_clock_fifo
296             generic map(
297                 lpm_numwords            => 1024,
298                 lpm_width               => DRAM_DATA_SIZE,
299                 lpm_widthu              => 10,
300                 rdsync_delaypipe        => 3,
301                 wrsync_delaypipe        => 3)
302             port map (
303                 data                    => dram_test_data,
304                 wrreq                   => wrreq_signal,
305                 rdreq                   => s_buffer_read,
306                 wrclk                   => dram_clk,
307                 rdclk                   => dram_clk,
308                 q                       => s_buffer_readdata,
309                 rdempty                 => s_buffer_empty,
310                 wrfull                  => wrfull_signal,
311                 aclr                    => '0',
312                 eccstatus               => open,
313                 rdfull                  => open,
314                 rdusedw                 => open,
315                 wrempty                 => open,
316                 wrusedw                 => wrusedw);
317 
318      pixel_input_0_wirer : component memory_block
319         generic map (
320             num_elements_a  => 256,
321             num_elements_b  => 256,
322             size_address_a  => 8,
323             size_address_b  => 8,
324             size_word_a     => INPUT_DATA_SIZE,
325             size_word_b     => INPUT_DATA_SIZE,
```

```
326             mem_init        => "E:/Users/Connor/Graduate_School/
                   Research/VHDL_Projects/
                   Camera_Simulation_8_Pixel_Averaging/ProjectFiles/
                   MATLAB/Top_Verification/Simulated_Data_Tap_1.mif"
327         )
328         port map (
329             address_a       => address,
330             address_b       => address,
331             clock_a         => camera_interface_clk,
332             clock_b         => camera_interface_clk,
333             data_a          => zeros,
334             data_b          => zeros,
335             wren_a          => '0',
336             wren_b          => '0',
337             q_a             => input_pixel_data_0,
338             q_b             => open);
339
340         pixel_input_1_wirer : component memory_block
341         generic map (
342             num_elements_a  => 256,
343             num_elements_b  => 256,
344             size_address_a  => 8,
345             size_address_b  => 8,
346             size_word_a     => INPUT_DATA_SIZE,
347             size_word_b     => INPUT_DATA_SIZE,
348             mem_init        => "E:/Users/Connor/Graduate_School/
                   Research/VHDL_Projects/
                   Camera_Simulation_8_Pixel_Averaging/ProjectFiles/
                   MATLAB/Top_Verification/Simulated_Data_Tap_2.mif"
349         )
350         port map (
351             address_a       => address,
352             address_b       => address,
353             clock_a         => camera_interface_clk,
354             clock_b         => camera_interface_clk,
355             data_a          => zeros,
356             data_b          => zeros,
357             wren_a          => '0',
358             wren_b          => '0',
359             q_a             => input_pixel_data_1,
360             q_b             => open);
361
362         pixel_input_2_wirer : component memory_block
363         generic map (
364             num_elements_a  => 256,
365             num_elements_b  => 256,
366             size_address_a  => 8,
367             size_address_b  => 8,
```

```
368             size_word_a     => INPUT_DATA_SIZE,
369             size_word_b     => INPUT_DATA_SIZE,
370             mem_init        => "E:/Users/Connor/Graduate_School/
                    Research/VHDL_Projects/
                    Camera_Simulation_8_Pixel_Averaging/ProjectFiles/
                    MATLAB/Top_Verification/Simulated_Data_Tap_3.mif"
371         )
372         port map (
373             address_a       => address,
374             address_b       => address,
375             clock_a         => camera_interface_clk,
376             clock_b         => camera_interface_clk,
377             data_a          => zeros,
378             data_b          => zeros,
379             wren_a          => '0',
380             wren_b          => '0',
381             q_a             => input_pixel_data_2,
382             q_b             => open);
383
384         pixel_input_3_wirer : component memory_block
385         generic map (
386             num_elements_a  => 256,
387             num_elements_b  => 256,
388             size_address_a  => 8,
389             size_address_b  => 8,
390             size_word_a     => INPUT_DATA_SIZE,
391             size_word_b     => INPUT_DATA_SIZE,
392             mem_init        => "E:/Users/Connor/Graduate_School/
                    Research/VHDL_Projects/
                    Camera_Simulation_8_Pixel_Averaging/ProjectFiles/
                    MATLAB/Top_Verification/Simulated_Data_Tap_4.mif"
393         )
394         port map (
395             address_a       => address,
396             address_b       => address,
397             clock_a         => camera_interface_clk,
398             clock_b         => camera_interface_clk,
399             data_a          => zeros,
400             data_b          => zeros,
401             wren_a          => '0',
402             wren_b          => '0',
403             q_a             => input_pixel_data_3,
404             q_b             => open);
405
406         pixel_input_4_wirer : component memory_block
407         generic map (
408             num_elements_a  => 256,
409             num_elements_b  => 256,
```

```
410             size_address_a  => 8,
411             size_address_b  => 8,
412             size_word_a     => INPUT_DATA_SIZE,
413             size_word_b     => INPUT_DATA_SIZE,
414             mem_init        => "E:/Users/Connor/Graduate_School/
                    Research/VHDL_Projects/
                    Camera_Simulation_8_Pixel_Averaging/ProjectFiles/
                    MATLAB/Top_Verification/Simulated_Data_Tap_5.mif"
415         )
416         port map (
417             address_a       => address,
418             address_b       => address,
419             clock_a         => camera_interface_clk,
420             clock_b         => camera_interface_clk,
421             data_a          => zeros,
422             data_b          => zeros,
423             wren_a          => '0',
424             wren_b          => '0',
425             q_a             => input_pixel_data_4,
426             q_b             => open);
427
428         pixel_input_5_wirer : component memory_block
429         generic map (
430             num_elements_a  => 256,
431             num_elements_b  => 256,
432             size_address_a  => 8,
433             size_address_b  => 8,
434             size_word_a     => INPUT_DATA_SIZE,
435             size_word_b     => INPUT_DATA_SIZE,
436             mem_init        => "E:/Users/Connor/Graduate_School/
                    Research/VHDL_Projects/
                    Camera_Simulation_8_Pixel_Averaging/ProjectFiles/
                    MATLAB/Top_Verification/Simulated_Data_Tap_6.mif"
437         )
438         port map (
439             address_a       => address,
440             address_b       => address,
441             clock_a         => camera_interface_clk,
442             clock_b         => camera_interface_clk,
443             data_a          => zeros,
444             data_b          => zeros,
445             wren_a          => '0',
446             wren_b          => '0',
447             q_a             => input_pixel_data_5,
448             q_b             => open);
449
450         pixel_input_6_wirer : component memory_block
451         generic map (
```

```
452             num_elements_a  => 256,
453             num_elements_b  => 256,
454             size_address_a  => 8,
455             size_address_b  => 8,
456             size_word_a     => INPUT_DATA_SIZE,
457             size_word_b     => INPUT_DATA_SIZE,
458             mem_init        => "E:/Users/Connor/Graduate_School/
                   Research/VHDL_Projects/
                   Camera_Simulation_8_Pixel_Averaging/ProjectFiles/
                   MATLAB/Top_Verification/Simulated_Data_Tap_7.mif"
459         )
460         port map (
461             address_a       => address,
462             address_b       => address,
463             clock_a         => camera_interface_clk,
464             clock_b         => camera_interface_clk,
465             data_a          => zeros,
466             data_b          => zeros,
467             wren_a          => '0',
468             wren_b          => '0',
469             q_a             => input_pixel_data_6,
470             q_b             => open);
471
472         pixel_input_7_wirer : component memory_block
473         generic map (
474             num_elements_a  => 256,
475             num_elements_b  => 256,
476             size_address_a  => 8,
477             size_address_b  => 8,
478             size_word_a     => INPUT_DATA_SIZE,
479             size_word_b     => INPUT_DATA_SIZE,
480             mem_init        => "E:/Users/Connor/Graduate_School/
                   Research/VHDL_Projects/
                   Camera_Simulation_8_Pixel_Averaging/ProjectFiles/
                   MATLAB/Top_Verification/Simulated_Data_Tap_8.mif"
481         )
482         port map (
483             address_a       => address,
484             address_b       => address,
485             clock_a         => camera_interface_clk,
486             clock_b         => camera_interface_clk,
487             data_a          => zeros,
488             data_b          => zeros,
489             wren_a          => '0',
490             wren_b          => '0',
491             q_a             => input_pixel_data_7,
492             q_b             => open);
493
```

```
494         pixel_input_8_wirer : component memory_block
495         generic map (
496             num_elements_a  => 256,
497             num_elements_b  => 256,
498             size_address_a  => 8,
499             size_address_b  => 8,
500             size_word_a     => INPUT_DATA_SIZE,
501             size_word_b     => INPUT_DATA_SIZE,
502             mem_init        => "E:/Users/Connor/Graduate_School/
                   Research/VHDL_Projects/
                   Camera_Simulation_8_Pixel_Averaging/ProjectFiles/
                   MATLAB/Top_Verification/Simulated_Data_Tap_9.mif"
503         )
504         port map (
505             address_a       => address,
506             address_b       => address,
507             clock_a         => camera_interface_clk,
508             clock_b         => camera_interface_clk,
509             data_a          => zeros,
510             data_b          => zeros,
511             wren_a          => '0',
512             wren_b          => '0',
513             q_a             => input_pixel_data_8,
514             q_b             => open);
515
516         pixel_input_9_wirer : component memory_block
517         generic map (
518             num_elements_a  => 256,
519             num_elements_b  => 256,
520             size_address_a  => 8,
521             size_address_b  => 8,
522             size_word_a     => INPUT_DATA_SIZE,
523             size_word_b     => INPUT_DATA_SIZE,
524             mem_init        => "E:/Users/Connor/Graduate_School/
                   Research/VHDL_Projects/
                   Camera_Simulation_8_Pixel_Averaging/ProjectFiles/
                   MATLAB/Top_Verification/Simulated_Data_Tap_10.mif"
525         )
526         port map (
527             address_a       => address,
528             address_b       => address,
529             clock_a         => camera_interface_clk,
530             clock_b         => camera_interface_clk,
531             data_a          => zeros,
532             data_b          => zeros,
533             wren_a          => '0',
534             wren_b          => '0',
535             q_a             => input_pixel_data_9,
```

```
536             q_b               => open);
537
538         write_dram_test_fifo_proc : process (dram_clk, reset_n)
539         begin
540             if(reset_n = '0') then
541                 dram_test_data <= std_logic_vector(
542                         spectral_bin_counter_zeros) &
543                         pixel_counter_zeros & light_data &
544                         dark_data;
545             elsif(rising_edge(dram_clk)) then
546                 if(wrfull_signal = '0' and delay_one_cycle_flag =
547                                                         '1') then
548                   if(pixel_condition = pixel_counter and
549                         spectral_bin_condition =
550                         spectral_bin_counter) then
551                     pixel_counter <= (others => '0');
552                     spectral_bin_counter <= to_unsigned(1,
553                                             DRAM_LOCATION_SIZE);
554                     dram_test_data <= std_logic_vector(
555                                 spectral_bin_counter_zeros) &
556                                 pixel_counter_zeros & light_data
557                                 & dark_data;
558                   elsif(spectral_bin_condition =
559                                         spectral_bin_counter) then
560                     pixel_counter <= pixel_counter + 1;
561                     spectral_bin_counter <= to_unsigned(1,
562                                             DRAM_LOCATION_SIZE);
563                     dram_test_data <= std_logic_vector(
564                                     spectral_bin_counter_zeros) &
565                                     std_logic_vector(pixel_counter
566                                     + 1) & light_data & dark_data;
567                   else
568                     spectral_bin_counter <= spectral_bin_counter
569                                                         + 1;
570                     dram_test_data<= std_logic_vector(
571                                         spectral_bin_counter) &
572                                         std_logic_vector(
573                                         pixel_counter) &
574                                         light_data & dark_data;
575                   end if;
576               end if;
577           end if;
578       end process;
579
580         write_request_proc : process (dram_clk, reset_n)
581           begin
582             if(reset_n = '0') then
583               hps_enable_flag           <= '0';
```

```
584                     wrreq_signal              <= '0';
585                     delay_one_cycle_flag      <= '1';
586                 elsif(rising_edge(dram_clk)) then
587                     if(wrfull_signal = '0') then
588                         wrreq_signal          <= '1';
589                         delay_one_cycle_flag <= '1';
590                     else
591                         hps_enable_flag       <= '1';
592                         wrreq_signal          <= '0';
593                         delay_one_cycle_flag <= '0';
594                     end if;
595                 end if;
596             end process;
597
598         -- Instantate the camera preprocessing block
599         camera_preprocessing_wirer : component camera_preprocessing
600             port map (rst_n                 => reset_n,
601                       component_clk         => camera_interface_clk,
602                       dram_clk              => dram_clk,
603                       hps_enable_flag       => hps_enable_flag,
604                       data_0_in             => input_pixel_data_0,
605                       data_1_in             => input_pixel_data_1,
606                       data_2_in             => input_pixel_data_2,
607                       data_3_in             => input_pixel_data_3,
608                       data_4_in             => input_pixel_data_4,
609                       data_5_in             => input_pixel_data_5,
610                       data_6_in             => input_pixel_data_6,
611                       data_7_in             => input_pixel_data_7,
612                       data_8_in             => input_pixel_data_8,
613                       data_9_in             => input_pixel_data_9,
614                       buffer_empty_in       => s_buffer_empty,
615                       buffer_read_out       => s_buffer_read,
616                       buffer_readdata_in    => s_buffer_readdata,
617                       data_packet_out       => data_packet_out,
618                       pixel_data_and_location_valid_flag_out =>
619                           pixel_data_and_location_valid_flag_signal,
620                       super_pixel_valid_flag_out =>
621                                                 super_pixel_valid_flag,
622                       clear_pixel_flag_out =>clear_pixel_flag_signal,
623                       new_frame_flag_out    => new_frame_flag);
624
625         -- Cross the clock domain between the Camera Interface and
626         -- the computition unit with a fifo
627         super_pixel_fifo_wirer : component dual_clock_fifo
628                 generic map(
629                     lpm_numwords            => 16,
630                     lpm_width               => SUPER_PIXEL_SIZE * 5,
631                     lpm_widthu              => 4,
```

```
632                 rdsync_delaypipe        => 4,
633                 wrsync_delaypipe        => 4)
634             port map (
635                 data                    => data_packet_out,
636                 wrreq                   => super_pixel_valid_flag,
637                 rdreq                   => super_pixel_rdreq,
638                 wrclk                   => camera_interface_clk,
639                 rdclk                   => dram_clk,
640                 q                       => data_to_computation,
641                 rdempty                 => super_pixel_empty,
642                 wrfull                  => super_pixel_full,
643                 aclr                    => open,
644                 eccstatus               => open,
645                 rdfull                  => open,
646                 rdusedw                 => open,
647                 wrempty                 => open,
648                 wrusedw                 => open);
649
650     -- Divide up the parallel output and pass out of the system
651     data_out_4 <= data_packet_out((SUPER_PIXEL_SIZE *
652                                    NUMBER_OF_PARALLEL_CHANNELS - 1)
653                                    downto data_packet_3_end);
654     data_out_3 <= data_packet_out(data_packet_3_end - 1 downto
655                                                   data_packet_2_end);
656     data_out_2 <= data_packet_out(data_packet_2_end - 1 downto
657                                                   data_packet_1_end);
658     data_out_1 <= data_packet_out(data_packet_1_end - 1 downto
659                                                   data_packet_0_end);
660     data_out_0 <= data_packet_out(data_packet_0_end -1 downto 0);
661
662 end architecture;
```

```
1  ----------------------------------------------------------------------------
2  --
3  --! @file        top.vhd
4  --! @brief
5  --
6  --! @author      Connor Dack
7  --! @date        March 2017
8  --! @copyright   Copyright (C) 2015 Ross K. Snider and Connor Dack
9  --
10 --   This program is free software: you can redistribute it
11 --   and/or modify it under the terms of the GNU General
12 --   Public License as published by the Free Software Foundation,
13 --   either version 3 of the License, or (at your option) any
14 --   later version.
15 --
16 --   This program is distributed in the hope that it will be
```

```
17 --   useful, but WITHOUT ANY WARRANTY; without even the implied
18 --   warranty of MERCHANTABILITY or FITNESS for A PARTICULAR
19 --   PURPOSE. See the GNU General Public License for more details.
20 --
21 --   You should have received a copy of the GNU General Public
22 --   License along with this program.  If not, see
23 --   <http://www.gnu.org/licenses/>.
24 --
25 --   Connor Dack
26 --   Electrical and Computer Engineering
27 --   Montana State University
28 --   610 Cobleigh Hall
29 --   Bozeman, MT 59717
30 --   connor.dack@msu.montana.edu
31 --
32 ----------------------------------------------------------------------------
33
34 library IEEE;
35 use IEEE.STD_LOGIC_1164.ALL;
36 use IEEE.NUMERIC_STD.ALL;
37
38 use work.Sensor_Package.ALL;
39
40 ----------------------------------------------------------------------------
41 --
42 --! @brief     top
43 --
44 --! @param     clk_100mhz       :  Input - 100 MHz Clock
45 --! @param     pixel_clk        :  Input - Pixel Clock
46 --! @param     sysclk           : Output - System Clock
47 --! @param     signal_tap_clk   : Output - Signal Tap Clock
48 --! @param     reset            :  Input - Active Low Reset
49 --! @param     pixel_0          :  Input - Tap 0
50 --! @param     pixel_1          :  Input - Tap 1
51 --! @param     pixel_2          :  Input - Tap 2
52 --! @param     pixel_3          :  Input - Tap 3
53 --! @param     pixel_4          :  Input - Tap 4
54 --! @param     pixel_5          :  Input - Tap 5
55 --! @param     pixel_6          :  Input - Tap 6
56 --! @param     pixel_7          :  Input - Tap 7
57 --! @param     pixel_8          :  Input - Tap 8
58 --! @param     pixel_9          :  Input - Tap 9
59 --! @param     cal_done_n       :  Input - Calibration Done Signal
60 --! @param     cal_strt_n       : Output - Calibration Start
61 --! @param     dark_off_en_n    : Output - Dark Off Enable
62 --! @param     ld_shft_n        : Output - LD Shift Signal
63 --! @param     data_read_en_n   : Output - Data Read Enable
64 --! @param     lrst_n           : Output - Hard Reset
```

```
65  --! @param     row_addr          : Output - Row Address
66  --! @param     row_done_n        :  Input - Row Done Signal
67  --! @param     row_strt_n        : Output - Row Start Signal
68  --! @param     standby_n         : Output - Standby
69  --! @param     tx_n              : Output - TX Signal
70  --! @param     pg_n              : Output - PG Signal
71  --
72  ------------------------------------------------------------------------------
73
74  entity top is
75      port(clk_100mhz          : in  std_logic;
76           pixel_clk           : in  std_logic;
77           sysclk              : out std_logic;
78           signal_tap_clk      : out std_logic;
79           reset               : in  std_logic := '0';
80           pixel_0             : in  std_logic_vector(9 downto 0);
81           pixel_1             : in  std_logic_vector(9 downto 0);
82           pixel_2             : in  std_logic_vector(9 downto 0);
83           pixel_3             : in  std_logic_vector(9 downto 0);
84           pixel_4             : in  std_logic_vector(9 downto 0);
85           pixel_5             : in  std_logic_vector(9 downto 0);
86           pixel_6             : in  std_logic_vector(9 downto 0);
87           pixel_7             : in  std_logic_vector(9 downto 0);
88           pixel_8             : in  std_logic_vector(9 downto 0);
89           pixel_9             : in  std_logic_vector(9 downto 0);
90           cal_done_n          : in  std_logic;
91           cal_strt_n          : out std_logic;
92           dark_off_en_n       : out std_logic;
93           ld_shft_n           : out std_logic;
94           data_read_en_n      : out std_logic;
95           lrst_n              : out std_logic;
96           row_addr            : out std_logic_vector(9 downto 0);
97           row_done_n          : in  std_logic;
98           row_strt_n          : out std_logic;
99           standby_n           : out std_logic;
100          tx_n                : out std_logic;
101          pg_n                : out std_logic);
102 end entity;
103
104 architecture rtl of top is
105
106     -- Internal Component Signals
107     signal memory_block_address           : std_logic_vector(
108                                              11 downto 0) := (
109                                              others => '0');
110     signal memory_block_pixel_data_valid  : std_logic;
111     signal camera_control_hps_enable_flag : std_logic := '0';
112     signal s_pixel_data                   : std_logic_vector(
```

```
113                                                 99 downto 0) := (
114                                                 others => '0');
115     constant c_ones                           : std_logic_vector(
116                                                 11 downto 0) := (
117                                                 others => '1');
118     signal cal_done_delay_counter             : integer := 0;
119     constant clk_cycles_after_cal_done        : integer := 100;
120     signal f_cal_done                         : std_logic := '0';
121     signal f_data_valid                       : std_logic := '0';
122     signal pll_lock                           : std_logic;
123     signal s_pixel_clk                        : std_logic;
124     signal s_clk_100mhz                       : std_logic;
125     signal reset_n                            : std_logic := '1';
126     signal s_lrst_n                           : std_logic := '1';
127     signal s_delay_1clk                       : std_logic := '0';
128     signal s_sysclk                           : std_logic := 'X';
129     signal delay_counter                      : integer range 0 to
130                                                 1000000000 := 0;
131     signal s_cal_done_n                       : std_logic;
132     signal s_cal_strt_n                       : std_logic;
133     signal s_dark_off_en_n                    : std_logic;
134     signal s_ld_shft_n                        : std_logic;
135     signal s_data_read_en_n                   : std_logic;
136     signal s_row_addr                         : std_logic_vector
137                                                 (9 downto 0);
138     signal s_row_done_n                       : std_logic;
139     signal s_row_strt_n                       : std_logic;
140     signal s_standby_n                        : std_logic;
141     signal s_tx_n                             : std_logic;
142     signal s_pg_n                             : std_logic;
143
144     -- Camera Control Qsys System
145     component camera_control_qsys is
146         port (
147             camera_control_hps_enable_flag    : in  std_logic
148                                               := 'X';
149             camera_control_cal_done_n_in      : in  std_logic
150                                               := 'X';
151             camera_control_cal_strt_n_out     : out std_logic;
152             camera_control_dark_off_en_n_out  : out std_logic;
153             camera_control_ld_shft_n_out      : out std_logic;
154             camera_control_data_read_en_n_out : out std_logic;
155             camera_control_row_addr_out       : out
156                                               std_logic_vector(9
157                                               downto 0);
158             camera_control_row_done_n_in      : in  std_logic
159                                               := 'X';
160             camera_control_row_strt_n_out     : out std_logic;
```

```
161             camera_control_standby_n_out       : out std_logic;
162             camera_control_tx_n_out            : out std_logic;
163             camera_control_pg_n_out            : out std_logic;
164             camera_control_pixel_clk_clk       : in  std_logic
165                                                := 'X';
166             clk_clk                            : in  std_logic
167                                                := 'X';
168             memory_block_pixel_data_in         : in
169                                                std_logic_vector(99
170                                                downto 0) :=
171                                                (others => 'X');
172             memory_block_pixel_data_valid_in   : in  std_logic
173                                                := 'X';
174             memory_block_memory_a_address_in   : in
175                                                std_logic_vector(11
176                                                downto 0) :=
177                                                (others => 'X');
178             memory_block_pixel_clk             : in  std_logic
179                                                := 'X';
180             reset_reset_n                      : in  std_logic
181                                                := 'X'
182         );
183     end component camera_control_qsys;
184
185     -- Camera Clock PLL
186     component camera_clock is
187         port (
188             rst      : in  std_logic := 'X'; -- reset
189             refclk   : in  std_logic := 'X'; -- clk
190             outclk_0 : out std_logic;         -- clk
191             outclk_1 : out std_logic         -- clk
192         );
193     end component camera_clock;
194
195     -- Pixel Clock PLL
196     component pixel_input_clock is
197         port (
198             inclk  : in  std_logic := 'X'; -- inclk
199             outclk : out std_logic         -- outclk
200         );
201     end component pixel_input_clock;
202
203     begin
204     -- Component Mappings
205     pxl_clk : component pixel_input_clock
206         port map (
207             inclk  => pixel_clk,   --  altclkctrl_input.inclk
208             outclk => s_pixel_clk  -- altclkctrl_output.outclk
```

```
209         );
210
211     sys_100mhz_clk : component pixel_input_clock
212         port map (
213             inclk  => clk_100mhz,    -- altclkctrl_input.inclk
214             outclk => s_clk_100mhz   -- altclkctrl_output.outclk
215         );
216
217     sys_clk : component camera_clock
218         port map (
219             rst      => reset,          -- reset.reset
220             refclk   => clk_100mhz,     -- refclk.clk
221             outclk_0 => s_sysclk,       -- outclk0.clk
222             outclk_1 => signal_tap_clk -- outclk1.clk
223         );
224
225     u0 : component camera_control_qsys
226         port map (
227             camera_control_hps_enable_flag     =>
228                                 camera_control_hps_enable_flag,
229             camera_control_cal_done_n_in       => s_cal_done_n,
230             camera_control_cal_strt_n_out      => s_cal_strt_n,
231             camera_control_dark_off_en_n_out   => s_dark_off_en_n,
232             camera_control_ld_shft_n_out       => s_ld_shft_n,
233             camera_control_data_read_en_n_out=> s_data_read_en_n,
234             camera_control_row_addr_out        => s_row_addr,
235             camera_control_row_done_n_in       => s_row_done_n,
236             camera_control_row_strt_n_out      => s_row_strt_n,
237             camera_control_standby_n_out       => s_standby_n,
238             camera_control_tx_n_out            => s_tx_n,
239             camera_control_pg_n_out            => s_pg_n,
240             camera_control_pixel_clk_clk       => s_sysclk,
241             clk_clk                            => s_clk_100mhz,
242             memory_block_pixel_data_in         => s_pixel_data,
243             memory_block_pixel_data_valid_in   =>
244                                  memory_block_pixel_data_valid,
245             memory_block_memory_a_address_in   =>
246                                        memory_block_address,
247             memory_block_pixel_clk             => s_sysclk,
248             reset_reset_n                      => reset_n
249         );
250
251     -- Register the pixel data
252     register_data : process(s_sysclk, reset_n)
253         begin
254             if(reset_n <= '0') then -- On reset,
255                 s_pixel_data <= (others => '0');
256                 -- clear the pixel data
```

```
257             elsif(rising_edge(s_sysclk)) then
258             -- On the rising edge of the system clock,
259                 if(f_data_valid = '1') then
260                 -- if the data is valid,
261                     s_pixel_data <= pixel_0 & pixel_1 & pixel_2 &
262                                     pixel_3 & pixel_4 & pixel_5 &
263                                     pixel_6 & pixel_7 & pixel_8 &
264                                     pixel_9;
265                     -- Register the pixel data into a single
266                     -- standard logic vector of 100 bits
267                     memory_block_pixel_data_valid <= '1';
268                     -- set the memory block valid flag
269                 else -- Otherwise
270                     s_pixel_data <= (others => '0');
271                     -- clear the pixel data
272                     memory_block_pixel_data_valid <= '0';
273                     -- clear the memory block valid flag
274                 end if;
275             end if;
276         end process;
277 
278         -- Memory Address Counter
279         mem_address_counter : process(s_sysclk, reset_n)
280             begin
281                 if(reset_n <= '0') then -- On reset,
282                     memory_block_address <= (others => '0');
283                     -- Clear the memory block address
284                 elsif(rising_edge(s_sysclk)) then
285                 -- On the rising edge of the system clock
286                     if(f_data_valid = '1') then
287                     -- If the data valid flag is set, then
288                         if(memory_block_address = c_ones) then
289                         -- If the memory address maxed out,
290                             memory_block_address <= (others => '0');
291                             -- Reset the memory block to zero
292                         else -- Otherwise,
293                             memory_block_address <= std_logic_vector(
294                                 unsigned(memory_block_address) + 1);
295                             -- increment the address counter by 1
296                         end if;
297                     end if;
298                 end if;
299             end process;
300 
301         -- Data valid process
302         data_valid_process : process(s_sysclk, reset_n)
303             begin
304                 if(reset_n <= '0') then -- On reset,
```

```
305                 f_data_valid <= '0';
306                 -- clear the data valid flag
307                 f_cal_done   <= '0';
308                 -- clear the calibration done flag
309                 cal_done_delay_counter <= 0;
310                 -- clear the calibration delay counter
311             elsif(rising_edge(s_sysclk)) then
312             -- On the rising edge of the system clock
313                 if(cal_done_n = '0') then
314                 -- if the calibration done signal is a zero, then
315                     f_cal_done <= '1';
316                     -- set the calibration done flag
317                 end if;
318                 if(f_cal_done = '1' and cal_done_delay_counter =
319                         clk_cycles_after_cal_done) then
320                 -- If the calibration done flag is set and the
321                 -- delay counter reached the count value, then
322                     f_data_valid <= '1';
323                     -- set the data valid flag
324                 else -- Otherwise,
325                     f_data_valid <= '0';
326                     -- clear the data valid flag
327                     cal_done_delay_counter <=
328                          cal_done_delay_counter + 1;
329                     -- increment the counter by 1
330                 end if;
331             end if;
332         end process;
333 
334         -- The hard reset process
335         lrst_n_proc : process(s_sysclk, reset_n)
336             begin
337                 if(reset_n <= '0') then -- On reset,
338                     s_lrst_n <= '1';
339                     -- set the lrst signal to 1
340                     s_delay_1clk <= '0';
341                     -- clear the delay counter
342                     camera_control_hps_enable_flag <= '0';
343                     -- clear the enable flag
344                  elsif(rising_edge(s_sysclk)) then
345                  -- On the rising edge of the system clock
346                     if(delay_counter = 1000000000) then
347                     -- If the delay counter reaches the above value
348                         s_lrst_n <= '1';
349                         -- set the lrst signal to 1
350                         camera_control_hps_enable_flag <= '1';
351                         -- set the simulated hps enable flag
352                     elsif(delay_counter = 999999998) then
```

```
353                 -- If the counter reaches the above value,
354                     s_lrst_n <= '0';
355                     -- set the lrst signal to 0
356                     delay_counter <= delay_counter + 1;
357                     -- increment the delay counter
358                 else -- Otherwise
359                     delay_counter <= delay_counter + 1;
360                     -- increment the delay counter
361                 end if;
362             end if;
363         end process;
364 
365     -- Assign the output signals
366     lrst_n          <= s_lrst_n;
367     sysclk          <= s_sysclk;
368     s_cal_done_n    <= cal_done_n;
369     cal_strt_n      <= not s_cal_strt_n;
370     dark_off_en_n   <= s_dark_off_en_n;
371     ld_shft_n       <= s_ld_shft_n;
372     data_read_en_n  <= s_data_read_en_n;
373     row_addr        <= s_row_addr;
374     s_row_done_n    <= row_done_n;
375     row_strt_n      <= s_row_strt_n;
376     standby_n       <= s_standby_n;
377     tx_n            <= s_tx_n;
378     pg_n            <= s_pg_n;
379 
380 end architecture;
```

```
1  ------------------------------------------------------------------------------
2  --
3  --! @file        camera_control_avalon.vhd
4  --! @brief       Avalon Wrapper for camera_control.vhd
5  --! @details     The avlaon wrapper allows easy implementation
6  --!              with the HPS in the Qsys system.
7  --
8  --! @author      Connor Dack
9  --! @date        May 2015
10 --! @copyright   Copyright (C) 2015 Ross K. Snider and Connor Dack
11 --
12 --   This program is free software: you can redistribute it
13 --   and/or modify it under the terms of the GNU General
14 --   Public License as published by the Free Software Foundation,
15 --   either version 3 of the License, or (at your option) any
16 --   later version.
17 --
18 --   This program is distributed in the hope that it will be
19 --   useful, but WITHOUT ANY WARRANTY; without even the implied
```

```
20 --   warranty of MERCHANTABILITY or FITNESS for A PARTICULAR
21 --   PURPOSE. See the GNU General Public License for more details.
22 --
23 --   You should have received a copy of the GNU General Public
24 --   License along with this program.  If not, see
25 --   <http://www.gnu.org/licenses/>.
26 --
27 --   Connor Dack
28 --   Electrical and Computer Engineering
29 --   Montana State University
30 --   610 Cobleigh Hall
31 --   Bozeman, MT 59717
32 --   connor.dack@msu.montana.edu
33 --
34 ----------------------------------------------------------------------------
35
36 library IEEE;
37 use IEEE.STD_LOGIC_1164.ALL;
38 use IEEE.NUMERIC_STD.ALL;
39
40 use work.Sensor_Package.ALL;
41 ----------------------------------------------------------------------------
42 --
43 --! @brief     camera_control_avalon
44 --!
45 --! @param     rst_n                 -  Input: Active low reset
46 --! @param     clk                   -  Input: System Clock
47 --! @param     hps_enable_flag       -  Input: HPS Enable Signal
48 --! @param     pixel_clk_in          -  Input: Sensor Clock
49 --! @param     cal_done_n_in         -  Input: Cal Done Signal
50 --! @param     cal_strt_n_out        - Output: Cal Strt Signal
51 --! @param     dark_off_en_n_out     - Output: Dark Off Enable
52 --! @param     ld_shft_n_out         - Output: LD Shift Signal
53 --! @param     data_read_en_n_out    - Output: Data Read Enable
54 --! @param     row_addr_out          - Output: Row Address
55 --! @param     row_done_n_in         -  Input: Row Done Signal
56 --! @param     row_strt_n_out        - Output: Row Start Signal
57 --! @param     standby_n_out         - Output: Standby Signal
58 --! @param     tx_n_out              - Output: TX Signal
59 --! @param     pg_n_out              - Output: PG Signal
60 --! @param     avs_s1_read           -  Input: Avalon Read Enable
61 --! @param     avs_s1_readdata       - Output: Avalon Read Data
62 --! @param     avs_s1_write          -  Input: Avalon Write Enable
63 --! @param     avs_s1_address        -  Input: Avalon address
64 --! @param     avs_s1_writedat       - Outptu: Avalon Write Data
65 ----------------------------------------------------------------------------
66
67 entity camera_control_avalon is
```

```
68      port(rst_n                :  in std_logic;
69           clk                  :  in std_logic;
70           hps_enable_flag      :  in std_logic;
71           pixel_clk_in         :  in std_logic;
72           cal_done_n_in        :  in std_logic;
73           cal_strt_n_out       : out std_logic;
74           dark_off_en_n_out    : out std_logic;
75           ld_shft_n_out        : out std_logic;
76           data_read_en_n_out   : out std_logic;
77           row_addr_out         : out std_logic_vector(9 downto 0);
78           row_done_n_in        :  in std_logic;
79           row_strt_n_out       : out std_logic;
80           standby_n_out        : out std_logic;
81           tx_n_out             : out std_logic;
82           pg_n_out             : out std_logic;
83           avs_s1_read          :  in std_logic;
84           avs_s1_readdata      : out std_logic_vector(31 downto 0);
85           avs_s1_write         :  in std_logic;
86           avs_s1_address       :  in std_logic_vector(1 downto 0);
87           avs_s1_writedata     :  in std_logic_vector(31 downto 0));
88  end camera_control_avalon;
89
90  architecture rtl of camera_control_avalon is
91
92      component camera_control is
93          port(
94           rst_n                :  in std_logic;
95           -- Input: Active low reset
96           hps_enable_flag_in   :  in std_logic;
97           -- Input: HPS Enable Flag
98           pixel_clk_in         :  in std_logic;
99           -- Input: Pixel Clock Input
100          cal_done_n_in        :  in std_logic;
101          -- Input: Cal Done Signal
102          cal_strt_n_out       : out std_logic;
103          -- Output: Cal Start Signal
104          dark_off_en_n_out    : out std_logic;
105          -- Output: Dark Off Enable
106          ld_shft_n_out        : out std_logic;
107          -- Output: LD Shift Signal
108          data_read_en_n_out   : out std_logic;
109          -- Output: Data Read Enable
110          row_addr_out         : out std_logic_vector(9 downto 0);
111          -- Output: Row Address
112          row_done_n_in        :  in std_logic;
113          -- Input: Row Done Signal
114          row_strt_n_out       : out std_logic;
115          -- Output: Row Start Signal
```

```
116         standby_n_out       : out std_logic;
117         -- Output: Standby Signal
118         tx_n_out            : out std_logic;
119         -- Output: TX Control Signal
120         pg_n_out            : out std_logic;
121         -- Output: PG Control Signal
122         exposure_clk_cycles_in:in std_logic_vector(17 downto 0);
123         --  Input: Exposure Value
124         exposure_row_addr_in:  in std_logic_vector(
125                                PIXEL_ADDRESS_SIZE - 1 downto 0);
126         --  Input: Exposure Row Addr
127         command_register_in :  in std_logic_vector(31 downto 0);
128         --  Input: Command Register
129         status_register_in  :  in std_logic_vector( 1 downto 0);
130         --  Input: Status Register
131         busy_flag_out       : out std_logic);
132         -- Output: Busy Flag Signal
133     end component camera_control;
134
135     type state_type_camera_control_avalon is (running, pending,
136                                               busy);
137     signal s_current_state_camera_control_avalon :
138                                state_type_camera_control_avalon
139                                := running;
140     signal s_next_state_camera_control_avalon    :
141                                state_type_camera_control_avalon;
142
143     constant c_exposure_row_addr_init   : integer
144                                         := NUMBER_OF_PIXELS / 2;
145     constant c_exposure_clk_cycles_init : integer
146                           := (c_exposure_row_addr_init * 130) -
                                 2;
147
148     signal s_exposure_clk_cycles : std_logic_vector(17 downto 0)
149                                := std_logic_vector(to_unsigned(
150                                c_exposure_clk_cycles_init, 18));
151                                -- 66558 clock cycles - 511 rows
152     signal s_exposure_row_addr   : std_logic_vector(
153                                  PIXEL_ADDRESS_SIZE - 1 downto 0)
154                                := std_logic_vector(to_unsigned(
155                                c_exposure_row_addr_init, P
156                                IXEL_ADDRESS_SIZE));
157                                -- 511: The middle of the frame
158     signal r_command    : std_logic_vector(31 downto 0)
159                                         := (others => '0');
160     signal r_status     : std_logic_vector( 1 downto 0)
161                                         := (others => '0');
162     signal s_wre                : std_logic;
```

```
163     signal s_addr                : std_logic_vector(1 downto 0);
164     signal s_re                  : std_logic;
165     signal s_read_data           : std_logic_vector(31 downto 0);
166     signal s_busy_flag           : std_logic;
167     signal s_state_machine_trigger  : std_logic := '0';
168     constant c_zeros             : std_logic_vector(32 -
169                                    PIXEL_ADDRESS_SIZE - 1 downto 0)
170                                    := (others => '0');
171
172     signal s_pixel_clk       : std_logic;
173     signal s_cal_done_n      : std_logic;
174     signal s_cal_strt_n      : std_logic;
175     signal s_dark_off_en_n   : std_logic;
176     signal s_ld_shft_n       : std_logic;
177     signal s_data_read_en_n  : std_logic;
178     signal s_row_addr        : std_logic_vector(9 downto 0);
179     signal s_row_done_n      : std_logic;
180     signal s_row_strt_n      : std_logic;
181     signal s_standby_n       : std_logic;
182     signal s_tx_n            : std_logic;
183     signal s_pg_n            : std_logic;
184
185     begin
186        -- Assign the avalon signals to internal component signals
187         s_wre           <= avs_s1_write;
188         s_addr          <= avs_s1_address;
189         s_re            <= avs_s1_read;
190         avs_s1_readdata <= s_read_data;
191
192        -- The instantiation of the camera control component
193         camera_cntrl : component camera_control
194             port map(
195                 rst_n                   => rst_n,
196                 hps_enable_flag_in      => hps_enable_flag,
197                 pixel_clk_in            => s_pixel_clk,
198                 cal_done_n_in           => s_cal_done_n,
199                 cal_strt_n_out          => s_cal_strt_n,
200                 dark_off_en_n_out       => s_dark_off_en_n,
201                 ld_shft_n_out           => s_ld_shft_n,
202                 data_read_en_n_out      => s_data_read_en_n,
203                 row_addr_out            => s_row_addr,
204                 row_done_n_in           => s_row_done_n,
205                 row_strt_n_out          => s_row_strt_n,
206                 standby_n_out           => s_standby_n,
207                 tx_n_out                => s_tx_n,
208                 pg_n_out                => s_pg_n,
209                 exposure_clk_cycles_in  => s_exposure_clk_cycles,
210                 exposure_row_addr_in    => s_exposure_row_addr ,
```

```
211                 command_register_in     => r_command,
212                 status_register_in      => r_status,
213                 busy_flag_out           => s_busy_flag);
214
215         -- The avalon control process - triggered off of the
216         -- system clock
217         avalon_proc : process (clk)
218             begin
219                 if(rising_edge(clk)) then
220                 -- On the rising edge of the system clock
221                     case s_addr is
222                         when "00" => -- Address 0
223                         -- Exposure Clock Cycles
224                             if(s_wre = '1' and s_re = '0') then
225                             -- Write the value of the exposure
226                             -- clock cycles
227                                 s_exposure_clk_cycles
228                                 <= avs_s1_writedata(17 downto 0);
229                             elsif(s_wre = '0' and s_re = '1')then
230                             -- Read the value of the exposure
231                             -- clock cycles
232                                 s_read_data <= "00000000000000"
233                                         & s_exposure_clk_cycles;
234                             end if;
235                         when "01" => -- Address 1
236                         -- Exposure Row Addresses
237                             if(s_wre = '1' and s_re = '0') then
238                             -- write exposure row address
239                                 s_exposure_row_addr <=
240                                             avs_s1_writedata
241                                             (PIXEL_ADDRESS_SIZE
242                                             - 1 downto 0);
243                             elsif(s_wre = '0' and s_re = '1')then
244                             -- read exposure row address
245                                 s_read_data <= c_zeros &
246                                             s_exposure_row_addr;
247                             end if;
248                         when "10" => -- Address 2
249                         -- Command Register
250                             if(s_wre = '1' and s_re = '0') then
251                             -- write the command register
252                                 r_command <= avs_s1_writedata;
253                             elsif(s_wre = '0' and s_re = '1')then
254                             -- read the command register
255                                 s_read_data <= r_command;
256                             end if;
257                         when "11" => -- Address 3
258                         -- Status Register
```

```
259                         if(s_wre = '0' and s_re = '1') then
260                         -- read the status register
261                             s_read_data <= x"0000000" & "00"
262                                         & r_status;
263                         end if;
264                     when others =>
265                 end case;
266             end if;
267         end process;
268
269     -- Camera Control Avalon Start Process - Triggered off of
270     -- Clock and Reset
271     -- Updates the current state on the rising edge of the
272     -- system clock
273     camera_control_avalon_start_proc : process (clk, rst_n)
274       begin
275         if (rst_n = '0') then
276             s_current_state_camera_control_avalon <= running;
277         elsif (rising_edge(clk)) then
278             s_current_state_camera_control_avalon
279                         <= s_next_state_camera_control_avalon;
280             s_state_machine_trigger
281                             <= not s_state_machine_trigger;
282         end if;
283      end process;
284
285     -- The camera control avalon next state process
286     -- Triggered on the current state, state machine trigger,
287     -- write enable, or the busy flag
288     camera_control_avalon_trans_proc : process (
289                         s_current_state_camera_control_avalon,
290                         s_state_machine_trigger,
291                         s_wre, s_busy_flag)
292      begin
293         if(hps_enable_flag = '1') then
294             case s_current_state_camera_control_avalon is
295                 when running =>
296                     if(s_wre = '1') then
297                         case s_addr is
298                             when "10"   =>
299                             s_next_state_camera_control_avalon
300                                             <= pending;
301                             when others =>
302                             s_next_state_camera_control_avalon
303                                             <= running;
304                         end case;
305                     end if;
306                 when pending =>
```

```
307                         if(s_busy_flag = '1') then
308                             s_next_state_camera_control_avalon
309                                                     <= busy;
310                         end if;
311                     when busy =>
312                         if(s_busy_flag = '0') then
313                             s_next_state_camera_control_avalon
314                                                   <= running;
315                         end if;
316                     when others =>
317                 end case;
318             end if;
319         end process;
320 
321         -- Camera Control Avalon Output Process - Triggered off of
322         -- current state
323         -- Sets the status registers for the HPS
324          camera_control_alavon_output_proc : process
325                             (s_current_state_camera_control_avalon)
326          begin
327             case s_current_state_camera_control_avalon is
328                 when running => r_status <= "00";
329                 when pending => r_status <= "01";
330                 when busy    => r_status <= "10";
331                 when others  =>
332             end case;
333          end process;
334 
335 end architecture;
```

```
1  ----------------------------------------------------------------------
2  --
3  --! @file        camera_control.vhd
4  --! @brief       The hyperspectral image sensor control code.
5  --! @details     Provides the camera control signals to the top
6  --!              level through the camera control state machines.
7  --
8  --! @author      Connor Dack
9  --! @date        May 2015
10 --! @copyright   Copyright (C) 2015 Ross K. Snider and Connor Dack
11 --
12 --   This program is free software: you can redistribute it
13 --   and/or modify it under the terms of the GNU General
14 --   Public License as published by the Free Software Foundation,
15 --   either version 3 of the License, or (at your option) any
16 --   later version.
17 --
18 --   This program is distributed in the hope that it will be
```

```
19 --   useful, but WITHOUT ANY WARRANTY; without even the implied
20 --   warranty of MERCHANTABILITY or FITNESS for A PARTICULAR
21 --   PURPOSE. See the GNU General Public License for more details.
22 --
23 --   You should have received a copy of the GNU General Public
24 --   License along with this program.  If not, see
25 --   <http://www.gnu.org/licenses/>.
26 --
27 --   Connor Dack
28 --   Electrical and Computer Engineering
29 --   Montana State University
30 --   610 Cobleigh Hall
31 --   Bozeman, MT 59717
32 --   connor.dack@msu.montana.edu
33 --
34 ----------------------------------------------------------------------------
35
36 library IEEE;
37 use IEEE.STD_LOGIC_1164.ALL;
38 use IEEE.NUMERIC_STD.ALL;
39
40 use work.Sensor_Package.ALL;
41 ----------------------------------------------------------------------------
42 --
43 --! @brief     camera_control
44 --!
45 --! @param     rst_n                   -  Input: Active low reset
46 --! @param     hps_enable_flag_in      -  Input: HPS Enable Flag
47 --! @param     pixel_clk_out           -  Input: Pixel Clock Input
48 --! @param     cal_done_n_in           -  Input: Cal Done Signal
49 --! @param     cal_strt_n_out          - Output: Cal Start Signal
50 --! @param     dark_off_en_n_out       - Output: Dark Off Enable
51 --! @param     ld_shft_n_out           - Output: LD Shift Signal
52 --! @param     data_read_en_n_out      - Output: Data Read Enable
53 --! @param     row_addr_out            - Output: Row Address
54 --! @param     row_done_n_in           -  Input: Row Done Signal
55 --! @param     row_strt_n_out          - Output: Row Start Signal
56 --! @param     standby_n_out           - Output: Standby Signal
57 --! @param     tx_n_out                - Output: TX Control Signal
58 --! @param     pg_n_out                - Output: PG Control Signal
59 --! @param     exposure_clk_cycles_in  -  Input: Exposure Value
60 --! @param     exposure_row_addr_in    -  Input: Exposure Row Addr
61 --! @param     command_register_in     -  Input: Command Register
62 --! @param     status_register_in      -  Input: Status Register
63 --! @param     busy_flag_out           - Output: Busy Flag Signal
64 ----------------------------------------------------------------------------
65
66 entity camera_control is
```

```
67      port(rst_n                  :  in std_logic;
68           hps_enable_flag_in     :  in std_logic;
69           pixel_clk_out          :  in std_logic;
70           cal_done_n_in          :  in std_logic;
71           cal_strt_n_out         : out std_logic;
72           dark_off_en_n_out      : out std_logic;
73           ld_shft_n_out          : out std_logic;
74           data_read_en_n_out     : out std_logic;
75           row_addr_out           : out std_logic_vector(9 downto 0);
76           row_done_n_in          :  in std_logic;
77           row_strt_n_out         : out std_logic;
78           standby_n_out          : out std_logic;
79           tx_n_out               : out std_logic;
80           pg_n_out               : out std_logic;
81           exposure_clk_cycles_in:in std_logic_vector(17 downto 0);
82           exposure_row_addr_in:  in std_logic_vector(
83                                  PIXEL_ADDRESS_SIZE - 1 downto 0);
84           command_register_in :  in std_logic_vector(31 downto 0);
85           status_register_in  :  in std_logic_vector( 1 downto 0);
86           busy_flag_out          : out std_logic);
87  end camera_control;
88
89  architecture camera_control_arch of camera_control is
90
91
92      type state_type_camera_control is (power_up, s0_start,
93                           s1_start, s2_start, s3_start, s4_start,
94                           s5_start, s0_read, s1_read, s2_read,
95                           s3_read, s4_read, s5_read, s0_exposure,
96                           s1_exposure, s2_exposure, s3_exposure,
97                           s0_end_exposure, s1_end_exposure,
98                           s2_end_exposure, s3_end_exposure);
99      -- The camera control state machine states
100     signal s_current_state_camera_control :
101                            state_type_camera_control := power_up;
102     -- The current state signal for the camera control
103     -- state machine
104     signal s_next_state_camera_control:state_type_camera_control;
105     -- The next state signal
106
107     -- Internal Component Signals
108     signal s_cal_done_n          : std_logic;
109     signal s_cal_strt_n          : std_logic := '1';
110     signal s_dark_off_en_n       : std_logic;
111     signal s_ld_shft_n           : std_logic := '1';
112     signal s_data_read_en_n      : std_logic := '1';
113     signal s_row_addr            : unsigned(9 downto 0) :=
114                           to_unsigned(NUMBER_OF_PIXELS - 1, 10);
```

```
115     signal s_row_done_n          : std_logic;
116     signal s_row_strt_n          : std_logic := '1';
117     signal s_standby_n           : std_logic;
118     signal s_tx_n                : std_logic := '1';
119     signal s_pg_n                : std_logic := '1';
120     signal s_single_clk_cycle_delay : unsigned(1 downto 0)
121                                           := (others => '0');
122     signal s_delay_counter          : unsigned(19 downto 0)
123                                           := (others => '0');
124     signal s_data_valid          : std_logic := '0';
125     signal s_state_machine_trigger  : std_logic := '0';
126     signal s_exposure_clk_cycles: unsigned(17 downto 0);
127     signal s_exposure_row_addr   : unsigned(PIXEL_ADDRESS_SIZE - 1
128                                                   downto 0);
129     signal s_status_register     : std_logic_vector(1 downto 0);
130     signal s_busy_flag           : std_logic;
131
132     begin
133
134     -- Assign the input of the component to internal signals
135     s_cal_done_n          <= cal_done_n_in;
136     s_exposure_clk_cycles <= unsigned(exposure_clk_cycles_in);
137     s_exposure_row_addr   <= unsigned(exposure_row_addr_in);
138
139     -- The camera control state movement process
140     camera_control_start_proc : process (pixel_clk_out, rst_n)
141         begin
142             if (rst_n = '0') then  -- On reset,
143                 s_current_state_camera_control <= s0_start;
144                 -- Set the current state to the start state
145             elsif (rising_edge(pixel_clk_out)) then
146             -- On the rising edge of the pixel clock
147                 s_current_state_camera_control <=
148                                         s_next_state_camera_control
                                            ;
149                 -- Update to the next state
150                 s_state_machine_trigger      <=
151                                       not s_state_machine_trigger;
152                 -- Change the state machine trigger state
153             end if;
154         end process;
155
156     -- The camera next state logic - updated when the current
157     -- state changes, the state machine trigger changes, the
158     -- row address changes, the cal done state changes, or the
159     -- status registers changes
160     camera_control_trans_proc : process (
```

```
161                                     s_current_state_camera_control
                                            ,
162                                     s_state_machine_trigger,
163                                     s_row_addr, s_cal_done_n,
164                                     s_status_register)
165         begin
166             if(hps_enable_flag_in = '1') then
167             -- If the system is enabled by the HPS, then
168                 case s_current_state_camera_control is
169               -- Update the next state based on the current state
170                     when power_up =>
171                         s_next_state_camera_control <= s0_start;
172                     when s0_start =>
173                         if(s_single_clk_cycle_delay = 1) then
174                         s_next_state_camera_control <= s1_start;
175                           s_single_clk_cycle_delay  <=
176                                                 (others => '0');
177                         else
178                           s_next_state_camera_control<= s0_start;
179                             s_single_clk_cycle_delay <=
180                                     s_single_clk_cycle_delay + 1;
181                         end if;
182                     when s1_start =>
183                         if(s_single_clk_cycle_delay = 1 and
184                                         s_cal_done_n = '0') then
185                         s_next_state_camera_control <= s2_start;
186                         s_single_clk_cycle_delay <=
187                                                 (others => '0');
188                         elsif(s_cal_done_n = '0') then
189                         s_next_state_camera_control <= s1_start;
190                         s_single_clk_cycle_delay <=
191                                     s_single_clk_cycle_delay + 1;
192                         else
193                         s_next_state_camera_control <= s1_start;
194                         end if;
195                     when s2_start =>
196                         if(s_delay_counter = 62) then
197                         s_next_state_camera_control <= s3_start;
198                         s_delay_counter <= (others => '0');
199                         else
200                         s_next_state_camera_control <= s2_start;
201                         s_delay_counter <= s_delay_counter + 1;
202                         end if;
203                     when s3_start =>
204                         if(s_delay_counter =
205                                       s_exposure_clk_cycles) then
206                         s_next_state_camera_control <= s4_start;
207                         s_delay_counter <= (others => '0');
```

```
208                 else
209                  s_next_state_camera_control <= s3_start;
210                  s_delay_counter <= s_delay_counter + 1;
211                 end if;
212             when s4_start =>
213                 if(s_delay_counter = 62) then
214                  s_next_state_camera_control <= s5_start;
215                  s_delay_counter <= (others => '0');
216                 else
217                  s_next_state_camera_control <= s4_start;
218                  s_delay_counter <= s_delay_counter + 1;
219                 end if;
220             when s5_start =>
221                   s_next_state_camera_control <= s0_read;
222             when s0_read  =>
223                 if(status_register_in(0) = '1') then
224                     case command_register_in is
225                         when x"00000001" =>
226                          s_next_state_camera_control <=
227                                                 s0_start;
228                         when others     =>
229                       end case;
230                 elsif(s_single_clk_cycle_delay = 1) then
231                   s_next_state_camera_control <= s1_read;
232                 else
233                   s_next_state_camera_control <= s0_read;
234                     s_single_clk_cycle_delay <=
235                             s_single_clk_cycle_delay + 1;
236                 end if;
237             when s1_read  =>
238                   s_next_state_camera_control <= s2_read;
239             when s2_read  =>
240                 if(s_row_addr = s_exposure_row_addr) then
241                     s_next_state_camera_control <=
242                                             s0_exposure;
243                 elsif(s_row_addr = NUMBER_OF_PIXELS - 1)
244                                                 then
245                     s_next_state_camera_control <=
246                                         s0_end_exposure;
247                 else
248                   s_next_state_camera_control <= s3_read;
249                 end if;
250             when s3_read  =>
251                 if(s_delay_counter = 125) then
252                   s_next_state_camera_control <= s4_read;
253                     s_delay_counter <= (others => '0');
254                 else
255                   s_next_state_camera_control <= s3_read;
```

```
256                 s_delay_counter <= s_delay_counter + 1;
257               end if;
258           when s4_read  =>
259                 s_next_state_camera_control <= s5_read;
260           when s5_read  =>
261                 s_next_state_camera_control <= s0_read;
262           when s0_exposure =>
263               if(s_delay_counter = 60) then
264                   s_next_state_camera_control <=
265                                                 s1_exposure;
266                   s_delay_counter <= (others => '0');
267               else
268                 s_next_state_camera_control <=
269                                                 s0_exposure;
270                 s_delay_counter <= s_delay_counter + 1;
271               end if;
272           when s1_exposure =>
273               if(s_delay_counter = 62) then
274                   s_next_state_camera_control <=
275                                                 s2_exposure;
276                   s_delay_counter <= (others => '0');
277               else
278                 s_next_state_camera_control <=
279                                                 s1_exposure;
280                 s_delay_counter <= s_delay_counter + 1;
281               end if;
282           when s2_exposure =>
283             s_next_state_camera_control <= s3_exposure;
284           when s3_exposure =>
285                 s_next_state_camera_control <= s0_read;
286           when s0_end_exposure =>
287               if(s_delay_counter = 60) then
288                   s_next_state_camera_control <=
289                                               s1_end_exposure;
290                   s_delay_counter <= (others => '0');
291               else
292                 s_next_state_camera_control <=
293                                               s0_end_exposure;
294                 s_delay_counter <= s_delay_counter + 1;
295               end if;
296           when s1_end_exposure =>
297               if(s_delay_counter = 62) then
298                   s_next_state_camera_control <=
299                                               s2_end_exposure;
300                   s_delay_counter <= (others => '0');
301               else
302                 s_next_state_camera_control <=
303                                               s1_end_exposure;
```

```
304                         s_delay_counter <= s_delay_counter + 1;
305                     end if;
306                 when s2_end_exposure =>
307                     s_next_state_camera_control
308                                         <= s3_end_exposure;
309                 when s3_end_exposure =>
310                     s_next_state_camera_control <= s0_read;
311                 when others =>
312                     s_next_state_camera_control <= s0_start;
313             end case;
314         end if;
315     end process;
316
317     -- The camera control output logic - Update if the current
318     -- State changes
319     -- These are the signals that control the iamge sensor
320     camera_control_output_proc : process
321                                  (s_current_state_camera_control)
322         begin
323             case s_current_state_camera_control is
324             -- Change the output logic based on the current state
325                 when s0_start       =>
326                     s_busy_flag  <= '1';
327                     s_cal_strt_n <= '0';
328                     s_row_addr   <= "1111111111";
329                     s_status_register(1) <= '1';
330                     s_status_register(0) <= '0';
331                 when s1_start       =>
332                     s_cal_strt_n <= '1';
333                 when s2_start       =>
334                     s_pg_n       <= '0';
335                     s_status_register(1) <= '0';
336                 when s3_start       =>
337                     s_pg_n       <= '1';
338                 when s4_start       =>
339                     s_tx_n       <= '0';
340                     if(command_register_in = x"00000001") then
341                         s_status_register(0) <= '1';
342                     end if;
343                 when s5_start       =>
344                     s_tx_n       <= '1';
345                     s_busy_flag  <= '0';
346                 when s0_read        =>
347                     if(s_row_addr = NUMBER_OF_PIXELS - 1) then
348                         s_row_addr <= (others => '0');
349                     else
350                         s_row_addr <= s_row_addr + 1;
351                     end if;
```

```
352                     s_ld_shft_n <= '0';
353                     s_data_read_en_n <= '0';
354                 when s1_read          =>
355                     s_row_strt_n <= '0';
356                 when s2_read          =>
357                     s_data_valid <= '1';
358                 when s3_read          =>
359                     s_row_strt_n <= '1';
360                 when s4_read          =>
361                     s_pg_n <= '0';
362                 when s5_read          =>
363                     s_pg_n <= '1';
364                     s_ld_shft_n <= '1';
365                     s_data_read_en_n <= '1';
366                     s_data_valid <= '0';
367                 when s0_exposure      =>
368                     s_row_strt_n <= '1';
369                 when s1_exposure      =>
370                     s_pg_n <= '0';
371                 when s2_exposure      =>
372                     s_pg_n <= '1';
373                 when s3_exposure      =>
374                     s_ld_shft_n <= '1';
375                     s_data_read_en_n <= '1';
376                     s_data_valid <= '0';
377                 when s0_end_exposure =>
378                     s_row_strt_n <= '1';
379                 when s1_end_exposure =>
380                     s_tx_n <= '0';
381                 when s2_end_exposure =>
382                     s_tx_n <= '1';
383                     s_pg_n <= '0';
384                 when s3_end_exposure =>
385                     s_pg_n <= '1';
386                     s_ld_shft_n <= '1';
387                     s_data_read_en_n <= '1';
388                     s_data_valid <= '0';
389                 when others =>
390             end case;
391         end process;
392
393     -- Assign the outputs of the component
394     cal_strt_n_out      <= s_cal_strt_n;
395     row_addr_out        <= std_logic_vector(s_row_addr);
396     ld_shft_n_out       <= s_ld_shft_n;
397     data_read_en_n_out <= s_data_read_en_n;
398     s_row_done_n        <= row_done_n_in;
399     row_strt_n_out      <= s_row_strt_n;
```

```
400     tx_n_out            <= s_tx_n;
401     pg_n_out            <= s_pg_n;
402     busy_flag_out       <= s_busy_flag;
403
404 end architecture;
```

```
1  ----------------------------------------------------------------------------
2  --
3  --! @file        camera_preprocessing_DUT.vhd
4  --! @brief       MATLAB Camera Interface Wrapper
5  --
6  --! @author      Connor Dack
7  --! @date        May 2015
8  --! @copyright   Copyright (C) 2015 Ross K. Snider and Connor Dack
9  --
10 --   This program is free software: you can redistribute it
11 --   and/or modify it under the terms of the GNU General
12 --   Public License as published by the Free Software Foundation,
13 --   either version 3 of the License, or (at your option) any
14 --   later version.
15 --
16 --   This program is distributed in the hope that it will be
17 --   useful, but WITHOUT ANY WARRANTY; without even the implied
18 --   warranty of MERCHANTABILITY or FITNESS for A PARTICULAR
19 --   PURPOSE. See the GNU General Public License for more details.
20 --
21 --   You should have received a copy of the GNU General Public
22 --   License along with this program.  If not, see
23 --   <http://www.gnu.org/licenses/>.
24 --
25 --   Connor Dack
26 --   Electrical and Computer Engineering
27 --   Montana State University
28 --   610 Cobleigh Hall
29 --   Bozeman, MT 59717
30 --   connor.dack@msu.montana.edu
31 --
32 ----------------------------------------------------------------------------
33
34 library IEEE;
35 use IEEE.STD_LOGIC_1164.ALL;
36 use IEEE.NUMERIC_STD.ALL;
37
38 use work.Sensor_Package.ALL;
39
40 entity camera_preprocessing_dut is
41    port(-- reset_n
42         reset_n     : in std_logic;
```

```
43          -- System Clock
44          system_clk    : in std_logic;
45          -- DRAM Clock
46          dram_clk      : in std_logic;
47          -- HPS Enable Flag - Go When '1'
48          -- f_hps_enable : in std_logic;
49          -- Pixel Data Inputs
50          input_data_0 : in std_logic_vector (INPUT_DATA_SIZE - 1
51                                                  downto 0);
52          input_data_1 : in std_logic_vector (INPUT_DATA_SIZE - 1
53                                                  downto 0);
54          input_data_2 : in std_logic_vector (INPUT_DATA_SIZE - 1
55                                                  downto 0);
56          input_data_3 : in std_logic_vector (INPUT_DATA_SIZE - 1
57                                                  downto 0);
58          input_data_4 : in std_logic_vector (INPUT_DATA_SIZE - 1
59                                                  downto 0);
60          input_data_5 : in std_logic_vector (INPUT_DATA_SIZE - 1
61                                                  downto 0);
62          input_data_6 : in std_logic_vector (INPUT_DATA_SIZE - 1
63                                                  downto 0);
64          input_data_7 : in std_logic_vector (INPUT_DATA_SIZE - 1
65                                                  downto 0);
66          input_data_8 : in std_logic_vector (INPUT_DATA_SIZE - 1
67                                                  downto 0);
68          input_data_9 : in std_logic_vector (INPUT_DATA_SIZE - 1
69                                                  downto 0);
70          -- Pixel Data and Location Flag
71          pixel_data_and_location_valid_flag : out std_logic;
72          -- Packet Valid Flag
73          super_pixel_valid_flag : out std_logic;
74          -- Location Error Flag
75          clear_pixel_flag       : out std_logic;
76          -- New Frame Flag
77          new_frame_flag  : out std_logic;
78          -- MATLAB Results
79          data_out_0 : out std_logic_vector (SUPER_PIXEL_SIZE -1
80                                                  downto 0);
81          data_out_1 : out std_logic_vector (SUPER_PIXEL_SIZE -1
82                                                  downto 0);
83          data_out_2 : out std_logic_vector (SUPER_PIXEL_SIZE -1
84                                                  downto 0);
85          data_out_3 : out std_logic_vector (SUPER_PIXEL_SIZE -1
86                                                  downto 0);
87          data_out_4 : out std_logic_vector (SUPER_PIXEL_SIZE -1
88                                                  downto 0));
89  end camera_preprocessing_dut;
90
```

```
91  architecture rtl of camera_preprocessing_dut is
92
93    component camera_preprocessing is
94          port(rst_n                : in std_logic;
95               component_clk        : in std_logic;
96               dram_clk             : in std_logic;
97               hps_enable_flag      : in std_logic;
98               data_0_in            : in std_logic_vector(
99                                       INPUT_DATA_SIZE - 1 downto 0);
100              data_1_in            : in std_logic_vector(
101                                      INPUT_DATA_SIZE - 1 downto 0);
102              data_2_in            : in std_logic_vector(
103                                      INPUT_DATA_SIZE - 1 downto 0);
104              data_3_in            : in std_logic_vector(
105                                      INPUT_DATA_SIZE - 1 downto 0);
106              data_4_in            : in std_logic_vector(
107                                      INPUT_DATA_SIZE - 1 downto 0);
108              data_5_in            : in std_logic_vector(
109                                      INPUT_DATA_SIZE - 1 downto 0);
110              data_6_in            : in std_logic_vector(
111                                      INPUT_DATA_SIZE - 1 downto 0);
112              data_7_in            : in std_logic_vector(
113                                      INPUT_DATA_SIZE - 1 downto 0);
114              data_8_in            : in std_logic_vector(
115                                      INPUT_DATA_SIZE - 1 downto 0);
116              data_9_in            : in std_logic_vector(
117                                      INPUT_DATA_SIZE - 1 downto 0);
118              buffer_empty_in      : in std_logic;
119              buffer_read_out      : out std_logic;
120              buffer_readdata_in   : in std_logic_vector(
121                                         DRAM_DATA_SIZE-1 downto 0);
122              data_packet_out      : out std_logic_vector(((
123                                         SUPER_PIXEL_SIZE *
124                                         NUMBER_OF_PARALLEL_CHANNELS)
125                                         - 1) downto 0);
126              pixel_data_and_location_valid_flag_out :
127                                                  out std_logic;
128              super_pixel_valid_flag_out : out std_logic;
129              clear_pixel_flag_out       : out std_logic;
130              new_frame_flag_out         : out std_logic);
131     end component;
132
133     component dram_fifo_for_location_check is
134         port (
135             data    : in  std_logic_vector(DRAM_DATA_SIZE - 1
136                           downto 0) := (others => 'X'); -- datain
137             wrreq   : in  std_logic := 'X'; -- wrreq
138             rdreq   : in  std_logic := 'X'; -- rdreq
```

```
139             wrclk    : in  std_logic := 'X'; -- wrclk
140             rdclk    : in  std_logic := 'X'; -- rdclk
141             q        : out std_logic_vector(DRAM_DATA_SIZE - 1
142                                 downto 0); -- dataout
143             rdempty  : out std_logic;       -- rdempty
144             wrfull   : out std_logic        -- wrfull
145         );
146     end component dram_fifo_for_location_check;
147
148     constant c_test_condition : integer := 208;
149
150   constant c_pixel_condition : unsigned (DRAM_LOCATION_SIZE - 1
151                                     downto 0) := to_unsigned((
152                                     NUMBER_OF_PIXELS) - 1,
153                                     DRAM_LOCATION_SIZE);
154   constant c_spectral_bin_condition : unsigned(
155                                       DRAM_LOCATION_SIZE - 1
156                                       downto 0) := to_unsigned
157                                       (NUMBER_OF_SPECTRAL_BINS,
158                                       DRAM_LOCATION_SIZE);
159     constant c_pixel_counter_zeros : std_logic_vector(
160                                      DRAM_LOCATION_SIZE - 1
161                                      downto 0) := (others =>
162                                      '0');
163     constant c_spectral_bin_counter_zeros : unsigned(
164                                       DRAM_LOCATION_SIZE - 1
165                                       downto 0) := (others =>
166                                       '0');
167
168     constant c_data_packet_3_end : natural := SUPER_PIXEL_SIZE *
169                                       NUMBER_OF_PARALLEL_CHANNELS
170                                       - SUPER_PIXEL_SIZE;
171     constant c_data_packet_2_end : natural := SUPER_PIXEL_SIZE *
172                                       NUMBER_OF_PARALLEL_CHANNELS
173                                       - (2 * SUPER_PIXEL_SIZE);
174     constant c_data_packet_1_end : natural := SUPER_PIXEL_SIZE *
175                                       NUMBER_OF_PARALLEL_CHANNELS
176                                       - (3 * SUPER_PIXEL_SIZE);
177     constant c_data_packet_0_end : natural := SUPER_PIXEL_SIZE *
178                                       NUMBER_OF_PARALLEL_CHANNELS
179                                       - (4 * SUPER_PIXEL_SIZE);
180
181
182     signal f_hps_enable : std_logic := '0';
183
184     -- DRAM INPUTS
185     signal s_buffer_empty     : std_logic := '0';
186     signal s_buffer_read      : std_logic := '0';
```

```
187     signal s_buffer_readdata : std_logic_vector(DRAM_DATA_SIZE -
188                                   1 downto 0) := (others => '0');
189
190     signal s_data_packet : std_logic_vector (((SUPER_PIXEL_SIZE *
191                                   NUMBER_OF_PARALLEL_CHANNELS) - 1)
192                                   downto 0) := (others => '0');
193
194     signal s_wrreq  : std_logic := '1';
195     signal s_wrfull : std_logic := '0';
196     signal f_delay_one_cycle : std_logic := '1';
197     signal s_light_data : std_logic_vector(LIGHT_CORRECT_SIZE - 1
198                                   downto 0) := (others => '0');
199     signal s_dark_data  : std_logic_vector(DARK_CORRECT_SIZE  - 1
200                                   downto 0) := (others => '1');
201
202     signal s_pixel_counter : unsigned (DRAM_LOCATION_SIZE - 1
203                                   downto 0) := (others => '0');
204   signal s_spectral_bin_counter : unsigned (DRAM_LOCATION_SIZE
205                                    - 1 downto 0) := to_unsigned(
206                                    1, DRAM_LOCATION_SIZE);
207     signal s_dram_test_data : std_logic_vector(DRAM_DATA_SIZE - 1
208                                  downto 0) := std_logic_vector(
209                                  c_spectral_bin_counter_zeros) &
210                                  c_pixel_counter_zeros &
211                                  s_light_data & s_dark_data;
212
213     signal s_test : integer range 0 to 500 := 0;
214     signal f_test : std_logic := '0';
215
216     begin
217
218     write_dram_test_fifo_proc : process (dram_clk, reset_n)
219     begin
220         if(reset_n = '0') then
221             s_dram_test_data <= std_logic_vector(
222                                  c_spectral_bin_counter_zeros) &
223                                  c_pixel_counter_zeros
224                                 & s_light_data & s_dark_data;
225         elsif(rising_edge(dram_clk)) then
226             if(s_wrfull = '0' and f_delay_one_cycle = '1') then
227                 if(c_pixel_condition = s_pixel_counter and
228                   c_spectral_bin_condition =
229                   s_spectral_bin_counter) then
230                     s_pixel_counter <= (others => '0');
231                     s_spectral_bin_counter <= to_unsigned(1,
232                                               DRAM_LOCATION_SIZE);
233                     s_dram_test_data <= std_logic_vector(
234                                         c_spectral_bin_counter_zeros)
```

```
235                                     & c_pixel_counter_zeros
236                                     & s_light_data & s_dark_data;
237                 elsif(c_spectral_bin_condition =
238                                     s_spectral_bin_counter) then
239                  s_pixel_counter <= s_pixel_counter + 1;
240                  s_spectral_bin_counter  <= to_unsigned(1,
241                                                 DRAM_LOCATION_SIZE);
242                  s_dram_test_data <= std_logic_vector(
243                                     c_spectral_bin_counter_zeros)
244                                     & std_logic_vector(
245                                     s_pixel_counter + 1)
246                                     & s_light_data & s_dark_data;
247                 else
248                  s_spectral_bin_counter <= s_spectral_bin_counter
249                                                             + 1;
250                  s_dram_test_data <= std_logic_vector(
251                                       s_spectral_bin_counter) &
252                                       std_logic_vector(
253                                       s_pixel_counter) &
254                                       s_light_data & s_dark_data;
255                end if;
256             end if;
257          end if;
258          end process;
259
260       write_request_proc : process (dram_clk, reset_n)
261          begin
262              if(reset_n = '0') then
263                  f_hps_enable          <= '0';
264                  s_wrreq               <= '1';
265                  f_delay_one_cycle     <= '1';
266              elsif(rising_edge(dram_clk)) then
267                  if(s_wrfull = '0') then
268                      s_wrreq           <= '1';
269                      f_delay_one_cycle <= '1';
270                  else
271                      f_hps_enable      <= '1';
272                      s_wrreq           <= '0';
273                      f_delay_one_cycle <= '0';
274                  end if;
275              end if;
276          end process;
277
278       u0 : component dram_fifo_for_location_check
279          port map (
280              data    => s_dram_test_data,
281              wrreq   => s_wrreq,
282              rdreq   => s_buffer_read,
```

```
283             wrclk   => dram_clk,
284             rdclk   => dram_clk,
285             q       => s_buffer_readdata,
286             rdempty => s_buffer_empty,
287             wrfull  => s_wrfull
288         );
289
290     camera_interface_wirer : component camera_preprocessing
291     port map(rst_n                  => reset_n,
292              component_clk          => system_clk,
293              dram_clk               => dram_clk,
294              hps_enable_flag        => f_hps_enable,
295              data_0_in              => input_data_0,
296              data_1_in              => input_data_1,
297              data_2_in              => input_data_2,
298              data_3_in              => input_data_3,
299              data_4_in              => input_data_4,
300              data_5_in              => input_data_5,
301              data_6_in              => input_data_6,
302              data_7_in              => input_data_7,
303              data_8_in              => input_data_8,
304              data_9_in              => input_data_9,
305              buffer_empty_in        => s_buffer_empty,
306              buffer_read_out        => s_buffer_read,
307              buffer_readdata_in     => s_buffer_readdata,
308              data_packet_out        => s_data_packet,
309              pixel_data_and_location_valid_flag_out =>
310                              pixel_data_and_location_valid_flag,
311              super_pixel_valid_flag_out =>super_pixel_valid_flag,
312              clear_pixel_flag_out => clear_pixel_flag,
313              new_frame_flag_out   => new_frame_flag);
314
315     -- Divide up the parallel output and pass out of the system
316     data_out_4 <= s_data_packet((SUPER_PIXEL_SIZE *
317                                  NUMBER_OF_PARALLEL_CHANNELS - 1)
318                                  downto c_data_packet_3_end);
319     data_out_3 <= s_data_packet(c_data_packet_3_end - 1 downto
320                                 c_data_packet_2_end);
321     data_out_2 <= s_data_packet(c_data_packet_2_end - 1 downto
322                                 c_data_packet_1_end);
323     data_out_1 <= s_data_packet(c_data_packet_1_end - 1 downto
324                                 c_data_packet_0_end);
325     data_out_0 <= s_data_packet(c_data_packet_0_end- 1 downto 0);
326
327 end architecture;
```

```
1 ---------------------------------------------------------------------
2 --
```

```
3  --! @file        camera_preprocessing.vhd
4  --! @brief       The hyperspectral image sensor control code.
5  --! @details     Provides the camera control signals to the top
6  --!              level through the camera control state machines.
7  --
8  --! @author      Connor Dack
9  --! @date        May 2015
10 --! @copyright   Copyright (C) 2015 Ross K. Snider and Connor Dack
11 --
12 --   This program is free software: you can redistribute it
13 --   and/or modify it under the terms of the GNU General
14 --   Public License as published by the Free Software Foundation,
15 --   either version 3 of the License, or (at your option) any
16 --   later version.
17 --
18 --   This program is distributed in the hope that it will be
19 --   useful, but WITHOUT ANY WARRANTY; without even the implied
20 --   warranty of MERCHANTABILITY or FITNESS for A PARTICULAR
21 --   PURPOSE. See the GNU General Public License for more details.
22 --
23 --   You should have received a copy of the GNU General Public
24 --   License along with this program.  If not, see
25 --   <http://www.gnu.org/licenses/>.
26 --
27 --   Connor Dack
28 --   Electrical and Computer Engineering
29 --   Montana State University
30 --   610 Cobleigh Hall
31 --   Bozeman, MT 59717
32 --   connor.dack@msu.montana.edu
33 --
34 ----------------------------------------------------------------------------
35
36 library IEEE;
37 use IEEE.STD_LOGIC_1164.ALL;
38 use IEEE.NUMERIC_STD.ALL;
39
40 use work.Sensor_Package.ALL;
41
42 ----------------------------------------------------------------------------
43 --
44 --! @brief     camera_preprocessing
45 --
46 --! @param     rst_n             : Input - Active Low Reset
47 --! @param     component_clk     : Input - Component Clock
48 --! @param     dram_clk          : Input - DRAM Clock
49 --! @param     hps_enable_flag   : Input - HPS Enable
50 --! @param     data_0_in         : Input - Tap 0
```

```
51 --! @param     data_1_in         : Input - Tap 1
52 --! @param     data_2_in         : Input - Tap 2
53 --! @param     data_3_in         : Input - Tap 3
54 --! @param     data_4_in         : Input - Tap 4
55 --! @param     data_5_in         : Input - Tap 5
56 --! @param     data_6_in         : Input - Tap 6
57 --! @param     data_7_in         : Input - Tap 7
58 --! @param     data_8_in         : Input - Tap 8
59 --! @param     data_9_in         : Input - Tap 9
60 --! @param     buffer_empty_in  : Input - Buffer Empty Signal
61 --! @param     buffer_read_out  : Output - Buffer Read Request
62 --! @param     buffer_readdata_in   : Output - Buffer Read Data
63 --! @param     data_packet_out      : Output - Data Packet
64 --! @param     pixel_data_and_location_valid_flag_out : Output
65 --!                                                    - Valid
66 --! @param     super_pixel_valid_flag_out : Output - Data Valid
67 --! @param     clear_pixel_flag_out        : Output
68 --!                                          - Clear Pixel Flag
69 --! @param     new_frame_flag_out          :Output - New Frame Flag
70 --
71 ----------------------------------------------------------------------------
72
73 entity camera_preprocessing is
74    port(rst_n               : in std_logic;
75         component_clk       : in std_logic;
76         dram_clk            : in std_logic;
77         hps_enable_flag     : in std_logic;
78         data_0_in           : in std_logic_vector(INPUT_DATA_SIZE
79                                                   - 1 downto 0);
80         data_1_in           : in std_logic_vector(INPUT_DATA_SIZE
81                                                   - 1 downto 0);
82         data_2_in           : in std_logic_vector(INPUT_DATA_SIZE
83                                                   - 1 downto 0);
84         data_3_in           : in std_logic_vector(INPUT_DATA_SIZE
85                                                   - 1 downto 0);
86         data_4_in           : in std_logic_vector(INPUT_DATA_SIZE
87                                                   - 1 downto 0);
88         data_5_in           : in std_logic_vector(INPUT_DATA_SIZE
89                                                   - 1 downto 0);
90         data_6_in           : in std_logic_vector(INPUT_DATA_SIZE
91                                                   - 1 downto 0);
92         data_7_in           : in std_logic_vector(INPUT_DATA_SIZE
93                                                   - 1 downto 0);
94         data_8_in           : in std_logic_vector(INPUT_DATA_SIZE
95                                                   - 1 downto 0);
96         data_9_in           : in std_logic_vector(INPUT_DATA_SIZE
97                                                   - 1 downto 0);
98         buffer_empty_in     : in std_logic;
```

```
99          buffer_read_out     : out std_logic;
100         buffer_readdata_in  : in std_logic_vector(
101                                   DRAM_DATA_SIZE-1 downto 0);
102         data_packet_out     : out std_logic_vector(((
103                                   SUPER_PIXEL_SIZE *
104                                   NUMBER_OF_PARALLEL_CHANNELS)
105                                   - 1) downto 0);
106         pixel_data_and_location_valid_flag_out : out std_logic;
107         super_pixel_valid_flag_out : out std_logic;
108         clear_pixel_flag_out       : out std_logic;
109         new_frame_flag_out         : out std_logic);
110 end camera_preprocessing;
111
112 architecture rtl of camera_preprocessing is
113
114     component pixel_averaging is
115        port(rst_n           : in std_logic;
116             component_clk   : in std_logic;
117             hps_enable_flag : in std_logic;
118             data_0_in : in std_logic_vector(INPUT_DATA_SIZE - 1
119                                                 downto 0);
120             data_1_in : in std_logic_vector(INPUT_DATA_SIZE - 1
121                                                 downto 0);
122             data_2_in : in std_logic_vector(INPUT_DATA_SIZE - 1
123                                                 downto 0);
124             data_3_in : in std_logic_vector(INPUT_DATA_SIZE - 1
125                                                 downto 0);
126             data_4_in : in std_logic_vector(INPUT_DATA_SIZE - 1
127                                                 downto 0);
128             data_5_in : in std_logic_vector(INPUT_DATA_SIZE - 1
129                                                 downto 0);
130             data_6_in : in std_logic_vector(INPUT_DATA_SIZE - 1
131                                                 downto 0);
132             data_7_in : in std_logic_vector(INPUT_DATA_SIZE - 1
133                                                 downto 0);
134             pixel_data_out : out std_logic_vector (DATA_SIZE - 1
135                                                   downto 0));
136     end component;
137
138     component pixel_location is
139         port(component_clk   : in std_logic;
140          rst_n           : in std_logic;
141              hps_enable_flag : in std_logic;
142          pixel_data_1_in : in std_logic_vector (DATA_SIZE - 1
143                                                 downto 0);
144              pixel_data_2_in : in std_logic_vector (DATA_SIZE - 1
145                                                 downto 0);
146          pixel_address_and_data_0_out : out std_logic_vector(
```

```
147                                 (DATA_PACKAGE_SIZE) - 1 downto 0);
148             pixel_address_and_data_1_out : out std_logic_vector(
149                                 (DATA_PACKAGE_SIZE) - 1 downto 0);
150                 pixel_data_and_location_valid_flag_out :
151                                                       out std_logic;
152                 start_reading_flag_out : out std_logic;
153                 new_frame_flag_out     : out std_logic);
154     end component;
155
156     component pixel_parallelization is
157         port(rst_n          : in std_logic;
158                 component_clk   : in std_logic;
159                 dram_clk        : in std_logic;
160                 hps_enable_flag : in std_logic;
161                 pixel_address_and_data_0_in : in std_logic_vector(
162                                 (DATA_PACKAGE_SIZE) - 1 downto 0);
163                 pixel_address_and_data_1_in : in std_logic_vector(
164                                 (DATA_PACKAGE_SIZE) - 1 downto 0);
165                 buffer_empty_in    : in std_logic;
166                 buffer_read_out    : out std_logic;
167                 buffer_readdata_in : in std_logic_vector(
168                                   DRAM_DATA_SIZE - 1 downto 0);
169                 data_packet_out    : out std_logic_vector(((
170                                   SUPER_PIXEL_SIZE *
171                                   NUMBER_OF_PARALLEL_CHANNELS)
172                                   - 1) downto 0);
173                 pixel_data_and_location_valid_flag_in :in std_logic;
174                 start_reading_flag_in      : in std_logic;
175                 clear_pixel_flag_out       : out std_logic;
176                 new_frame_flag_out         : out std_logic;
177                 super_pixel_valid_flag_out : out std_logic);
178     end component;
179
180     signal s_data_0_in : std_logic_vector(INPUT_DATA_SIZE
181                                     - 1 downto 0);
182     signal s_data_1_in : std_logic_vector(INPUT_DATA_SIZE
183                                     - 1 downto 0);
184     signal s_data_2_in : std_logic_vector(INPUT_DATA_SIZE
185                                     - 1 downto 0);
186     signal s_data_3_in : std_logic_vector(INPUT_DATA_SIZE
187                                     - 1 downto 0);
188     signal s_data_4_in : std_logic_vector(INPUT_DATA_SIZE
189                                      - 1 downto 0);
190     signal s_data_5_in : std_logic_vector(INPUT_DATA_SIZE
191                                     - 1 downto 0);
192     signal s_data_6_in : std_logic_vector(INPUT_DATA_SIZE
193                                     - 1 downto 0);
194     signal s_data_7_in : std_logic_vector(INPUT_DATA_SIZE
```

```
195                                         - 1 downto 0);
196     signal s_data_8_in : std_logic_vector(INPUT_DATA_SIZE
197                                         - 1 downto 0);
198     signal s_data_9_in : std_logic_vector(INPUT_DATA_SIZE
199                                          - 1 downto 0);
200     signal s_buffer_empty    : std_logic;
201     signal s_buffer_read     : std_logic;
202     signal s_buffer_readdata : std_logic_vector(
203                                DRAM_DATA_SIZE-1 downto 0);
204     signal s_pixel_data_0 , s_pixel_data_1 , s_pixel_data_2 ,
205            s_pixel_data_3 , s_pixel_data_4 , s_pixel_data_5 ,
206            s_pixel_data_6 , s_pixel_data_7 : std_logic_vector
207            (INPUT_DATA_SIZE - 1 downto 0) := (others => '0');
208     signal s_pixel_data_8 , s_pixel_data_9 , s_pixel_data_10 ,
209            s_pixel_data_11 , s_pixel_data_12 , s_pixel_data_13 ,
210            s_pixel_data_14 , s_pixel_data_15 : std_logic_vector
211            (INPUT_DATA_SIZE - 1 downto 0) := (others => '0');
212
213     -- State machine for controlling the inputs to the components
214     type register_states is (S0, S1, S2, S3, S_Start);
215     signal current_state       : register_states := S_Start;
216     signal next_state          : register_states;
217
218     -- Registers required to align the 8 pixels and
219     -- register the remaining
220     signal s_pixel_register_s0_0 : std_logic_vector(
221                                    INPUT_DATA_SIZE - 1 downto 0)
222                                    := (others => '0');
223     signal s_pixel_register_s0_1 : std_logic_vector(
224                                    INPUT_DATA_SIZE - 1 downto 0)
225                                    := (others => '0');
226     signal s_pixel_register_s0_2 : std_logic_vector(
227                                    INPUT_DATA_SIZE - 1 downto 0)
228                                    := (others => '0');
229     signal s_pixel_register_s0_3 : std_logic_vector(
230                                    INPUT_DATA_SIZE - 1 downto 0)
231                                    := (others => '0');
232     signal s_pixel_register_s0_4 : std_logic_vector(
233                                    INPUT_DATA_SIZE - 1 downto 0)
234                                    := (others => '0');
235     signal s_pixel_register_s0_5 : std_logic_vector(
236                                    INPUT_DATA_SIZE - 1 downto 0)
237                                    := (others => '0');
238     signal s_pixel_register_s0_6 : std_logic_vector(
239                                    INPUT_DATA_SIZE - 1 downto 0)
240                                    := (others => '0');
241     signal s_pixel_register_s0_7 : std_logic_vector(
242                                    INPUT_DATA_SIZE - 1 downto 0)
```

```
243                                     := (others => '0');
244     signal s_pixel_register_s0_8 : std_logic_vector(
245                                     INPUT_DATA_SIZE - 1 downto 0)
246                                     := (others => '0');
247     signal s_pixel_register_s0_9 : std_logic_vector(
248                                     INPUT_DATA_SIZE - 1 downto 0)
249                                     := (others => '0');
250
251     signal s_pixel_register_s1_0 : std_logic_vector(
252                                     INPUT_DATA_SIZE - 1 downto 0)
253                                     := (others => '0');
254     signal s_pixel_register_s1_1 : std_logic_vector(
255                                     INPUT_DATA_SIZE - 1 downto 0)
256                                     := (others => '0');
257     signal s_pixel_register_s1_2 : std_logic_vector(
258                                     INPUT_DATA_SIZE - 1 downto 0)
259                                     := (others => '0');
260     signal s_pixel_register_s1_3 : std_logic_vector(
261                                     INPUT_DATA_SIZE - 1 downto 0)
262                                     := (others => '0');
263     signal s_pixel_register_s1_4 : std_logic_vector(
264                                     INPUT_DATA_SIZE - 1 downto 0)
265                                     := (others => '0');
266     signal s_pixel_register_s1_5 : std_logic_vector(
267                                     INPUT_DATA_SIZE - 1 downto 0)
268                                     := (others => '0');
269     signal s_pixel_register_s1_6 : std_logic_vector(
270                                     INPUT_DATA_SIZE - 1 downto 0)
271                                     := (others => '0');
272     signal s_pixel_register_s1_7 : std_logic_vector(
273                                     INPUT_DATA_SIZE - 1 downto 0)
274                                     := (others => '0');
275     signal s_pixel_register_s1_8 : std_logic_vector(
276                                     INPUT_DATA_SIZE - 1 downto 0)
277                                     := (others => '0');
278     signal s_pixel_register_s1_9 : std_logic_vector(
279                                     INPUT_DATA_SIZE - 1 downto 0)
280                                     := (others => '0');
281
282     signal s_pixel_register_s2_0 : std_logic_vector(
283                                     INPUT_DATA_SIZE - 1 downto 0)
284                                     := (others => '0');
285     signal s_pixel_register_s2_1 : std_logic_vector(
286                                     INPUT_DATA_SIZE - 1 downto 0)
287                                     := (others => '0');
288     signal s_pixel_register_s2_2 : std_logic_vector(
289                                     INPUT_DATA_SIZE - 1 downto 0)
290                                     := (others => '0');
```

```
291     signal s_pixel_register_s2_3 : std_logic_vector(
292                                    INPUT_DATA_SIZE - 1 downto 0)
293                                    := (others => '0');
294     signal s_pixel_register_s2_4 : std_logic_vector(
295                                    INPUT_DATA_SIZE - 1 downto 0)
296                                    := (others => '0');
297     signal s_pixel_register_s2_5 : std_logic_vector(
298                                    INPUT_DATA_SIZE - 1 downto 0)
299                                    := (others => '0');
300     signal s_pixel_register_s2_6 : std_logic_vector(
301                                    INPUT_DATA_SIZE - 1 downto 0)
302                                    := (others => '0');
303     signal s_pixel_register_s2_7 : std_logic_vector(
304                                    INPUT_DATA_SIZE - 1 downto 0)
305                                    := (others => '0');
306     signal s_pixel_register_s2_8 : std_logic_vector(
307                                    INPUT_DATA_SIZE - 1 downto 0)
308                                    := (others => '0');
309     signal s_pixel_register_s2_9 : std_logic_vector(
310                                    INPUT_DATA_SIZE - 1 downto 0)
311                                    := (others => '0');
312
313     signal s_pixel_register_s3_0 : std_logic_vector(
314                                    INPUT_DATA_SIZE - 1 downto 0)
315                                    := (others => '0');
316     signal s_pixel_register_s3_1 : std_logic_vector(
317                                    INPUT_DATA_SIZE - 1 downto 0)
318                                    := (others => '0');
319     signal s_pixel_register_s3_2 : std_logic_vector(
320                                    INPUT_DATA_SIZE - 1 downto 0)
321                                    := (others => '0');
322     signal s_pixel_register_s3_3 : std_logic_vector(
323                                    INPUT_DATA_SIZE - 1 downto 0)
324                                    := (others => '0');
325     signal s_pixel_register_s3_4 : std_logic_vector(
326                                    INPUT_DATA_SIZE - 1 downto 0)
327                                    := (others => '0');
328     signal s_pixel_register_s3_5 : std_logic_vector(
329                                    INPUT_DATA_SIZE - 1 downto 0)
330                                    := (others => '0');
331     signal s_pixel_register_s3_6 : std_logic_vector(
332                                    INPUT_DATA_SIZE - 1 downto 0)
333                                    := (others => '0');
334     signal s_pixel_register_s3_7 : std_logic_vector(
335                                    INPUT_DATA_SIZE - 1 downto 0)
336                                    := (others => '0');
337     signal s_pixel_register_s3_8 : std_logic_vector(
338                                    INPUT_DATA_SIZE - 1 downto 0)
```

```
339                                     := (others => '0');
340     signal s_pixel_register_s3_9 : std_logic_vector(
341                                     INPUT_DATA_SIZE - 1 downto 0)
342                                     := (others => '0');
343
344
345     -- Data out signals
346     signal s_pixel_data_0_summed, s_pixel_data_1_summed :
           std_logic_vector(
347                                                 DATA_SIZE - 1 downto
348                                                 0) := (OTHERS => '0');
349     signal s_pixel_address_and_data_0, s_pixel_address_and_data_1
350                             : std_logic_vector (DATA_PACKAGE_SIZE
351                             - 1 downto 0):=(OTHERS => '0');
352     signal f_pixel_data_and_location_valid : std_logic := '0';
353
354     -- Flags from the different components
355     signal f_super_pixel_valid          : std_logic := '0';
356     signal f_new_frame_parallelization  : std_logic := '0';
357     signal f_new_frame_pixel_location   : std_logic := '0';
358     signal f_start_reading              : std_logic := '0';
359     signal f_input_valid                : std_logic := '0';
360     signal f_clear_pixel                : std_logic := '0';
361
362     begin
363
364     s_data_0_in       <= data_0_in;
365     s_data_1_in       <= data_1_in;
366     s_data_2_in       <= data_2_in;
367     s_data_3_in       <= data_3_in;
368     s_data_4_in       <= data_4_in;
369     s_data_5_in       <= data_5_in;
370     s_data_6_in       <= data_6_in;
371     s_data_7_in       <= data_7_in;
372     s_data_8_in       <= data_8_in;
373     s_data_9_in       <= data_9_in;
374     s_buffer_empty    <= buffer_empty_in;
375     s_buffer_readdata <= buffer_readdata_in;
376
377     -- Start process - Update on the clock edge
378     start_proc : process (component_clk, rst_n)
379         begin
380             if (rst_n = '0') then
381                 current_state <= S_Start;
382             elsif (rising_edge(component_clk)) then
383                 if(hps_enable_flag = '1') then
384                     if(f_input_valid = '1') then
385                         current_state <= next_state;
```

```
386                     else
387                         f_input_valid <= '1';
388                     end if;
389                 end if;
390             end if;
391         end process;
392     -- Transition Process - What is the next state
393     state_trans_proc : process (current_state)
394         begin
395             case current_state is
396                 when S_Start => next_state <= S3;
397                 when S0      => next_state <= S1;
398                 when S1      => next_state <= S2;
399                 when S2      => next_state <= S3;
400                 when S3      => next_state <= S0;
401                 when others  => next_state <= S_Start;
402             end case;
403         end process;
404     -- Output process - Set the inputs and the regsisters
405     state_output_proc : process (current_state)
406         begin
407             case current_state is
408                     when S_Start =>
409                        s_pixel_data_0 <= (others => '0');
410                        s_pixel_data_1 <= (others => '0');
411                        s_pixel_data_2 <= (others => '0');
412                        s_pixel_data_3 <= (others => '0');
413                        s_pixel_data_4 <= (others => '0');
414                        s_pixel_data_5 <= (others => '0');
415                        s_pixel_data_6 <= (others => '0');
416                        s_pixel_data_7 <= (others => '0');
417
418                        s_pixel_data_8 <= (others => '0');
419                        s_pixel_data_9 <= (others => '0');
420                        s_pixel_data_10 <= (others => '0');
421                        s_pixel_data_11 <= (others => '0');
422                        s_pixel_data_12 <= (others => '0');
423                        s_pixel_data_13 <= (others => '0');
424                        s_pixel_data_14 <= (others => '0');
425                        s_pixel_data_15 <= (others => '0');
426
427                        s_pixel_register_s0_0 <= (others => '0');
428                        s_pixel_register_s0_1 <= (others => '0');
429                        s_pixel_register_s0_2 <= (others => '0');
430                        s_pixel_register_s0_3 <= (others => '0');
431                        s_pixel_register_s0_4 <= (others => '0');
432                        s_pixel_register_s0_5 <= (others => '0');
433                        s_pixel_register_s0_6 <= (others => '0');
```

```
434                 s_pixel_register_s0_7 <= (others => '0');
435                 s_pixel_register_s0_8 <= (others => '0');
436                 s_pixel_register_s0_9 <= (others => '0');
437 
438                 s_pixel_register_s1_0 <= (others => '0');
439                 s_pixel_register_s1_1 <= (others => '0');
440                 s_pixel_register_s1_2 <= (others => '0');
441                 s_pixel_register_s1_3 <= (others => '0');
442                 s_pixel_register_s1_4 <= (others => '0');
443                 s_pixel_register_s1_5 <= (others => '0');
444                 s_pixel_register_s1_6 <= (others => '0');
445                 s_pixel_register_s1_7 <= (others => '0');
446                 s_pixel_register_s1_8 <= (others => '0');
447                 s_pixel_register_s1_9 <= (others => '0');
448 
449                 s_pixel_register_s2_0 <= (others => '0');
450                 s_pixel_register_s2_1 <= (others => '0');
451                 s_pixel_register_s2_2 <= (others => '0');
452                 s_pixel_register_s2_3 <= (others => '0');
453                 s_pixel_register_s2_4 <= (others => '0');
454                 s_pixel_register_s2_5 <= (others => '0');
455                 s_pixel_register_s2_6 <= (others => '0');
456                 s_pixel_register_s2_7 <= (others => '0');
457                 s_pixel_register_s2_8 <= (others => '0');
458                 s_pixel_register_s2_9 <= (others => '0');
459 
460                 s_pixel_register_s3_0 <= (others => '0');
461                 s_pixel_register_s3_1 <= (others => '0');
462                 s_pixel_register_s3_2 <= (others => '0');
463                 s_pixel_register_s3_3 <= (others => '0');
464                 s_pixel_register_s3_4 <= (others => '0');
465                 s_pixel_register_s3_5 <= (others => '0');
466                 s_pixel_register_s3_6 <= (others => '0');
467                 s_pixel_register_s3_7 <= (others => '0');
468                 s_pixel_register_s3_8 <= (others => '0');
469                 s_pixel_register_s3_9 <= (others => '0');
470 
471             when S0 =>
472                 s_pixel_data_0 <= s_pixel_register_s3_0;
473                 s_pixel_data_1 <= s_pixel_register_s3_1;
474                 s_pixel_data_2 <= s_pixel_register_s3_2;
475                 s_pixel_data_3 <= s_pixel_register_s3_3;
476                 s_pixel_data_4 <= s_pixel_register_s3_4;
477                 s_pixel_data_5 <= s_pixel_register_s3_5;
478                 s_pixel_data_6 <= s_pixel_register_s3_6;
479                 s_pixel_data_7 <= s_pixel_register_s3_7;
480 
481                 s_pixel_register_s0_0 <= s_data_0_in;
```

```
482                 s_pixel_register_s0_1 <= s_data_1_in;
483                 s_pixel_register_s0_2 <= s_data_2_in;
484                 s_pixel_register_s0_3 <= s_data_3_in;
485                 s_pixel_register_s0_4 <= s_data_4_in;
486                 s_pixel_register_s0_5 <= s_data_5_in;
487                 s_pixel_register_s0_6 <= s_data_6_in;
488                 s_pixel_register_s0_7 <= s_data_7_in;
489                 s_pixel_register_s0_8 <= s_data_8_in;
490                 s_pixel_register_s0_9 <= s_data_9_in;
491
492             when S1 =>
493                 s_pixel_data_0 <= s_pixel_register_s3_8;
494                 s_pixel_data_1 <= s_pixel_register_s3_9;
495                 s_pixel_data_2 <= s_pixel_register_s0_0;
496                 s_pixel_data_3 <= s_pixel_register_s0_1;
497                 s_pixel_data_4 <= s_pixel_register_s0_2;
498                 s_pixel_data_5 <= s_pixel_register_s0_3;
499                 s_pixel_data_6 <= s_pixel_register_s0_4;
500                 s_pixel_data_7 <= s_pixel_register_s0_5;
501
502                 s_pixel_register_s1_0 <= s_data_0_in;
503                 s_pixel_register_s1_1 <= s_data_1_in;
504                 s_pixel_register_s1_2 <= s_data_2_in;
505                 s_pixel_register_s1_3 <= s_data_3_in;
506                 s_pixel_register_s1_4 <= s_data_4_in;
507                 s_pixel_register_s1_5 <= s_data_5_in;
508                 s_pixel_register_s1_6 <= s_data_6_in;
509                 s_pixel_register_s1_7 <= s_data_7_in;
510                 s_pixel_register_s1_8 <= s_data_8_in;
511                 s_pixel_register_s1_9 <= s_data_9_in;
512
513             when S2 =>
514                 s_pixel_data_0 <= s_pixel_register_s0_6;
515                 s_pixel_data_1 <= s_pixel_register_s0_7;
516                 s_pixel_data_2 <= s_pixel_register_s0_8;
517                 s_pixel_data_3 <= s_pixel_register_s0_9;
518                 s_pixel_data_4 <= s_pixel_register_s1_0;
519                 s_pixel_data_5 <= s_pixel_register_s1_1;
520                 s_pixel_data_6 <= s_pixel_register_s1_2;
521                 s_pixel_data_7 <= s_pixel_register_s1_3;
522
523                 s_pixel_register_s2_0 <= s_data_0_in;
524                 s_pixel_register_s2_1 <= s_data_1_in;
525                 s_pixel_register_s2_2 <= s_data_2_in;
526                 s_pixel_register_s2_3 <= s_data_3_in;
527                 s_pixel_register_s2_4 <= s_data_4_in;
528                 s_pixel_register_s2_5 <= s_data_5_in;
529                 s_pixel_register_s2_6 <= s_data_6_in;
```

```
530                 s_pixel_register_s2_7 <= s_data_7_in;
531                 s_pixel_register_s2_8 <= s_data_8_in;
532                 s_pixel_register_s2_9 <= s_data_9_in;
533 
534               when S3 =>
535                s_pixel_data_0 <= s_pixel_register_s1_4;
536                s_pixel_data_1 <= s_pixel_register_s1_5;
537                s_pixel_data_2 <= s_pixel_register_s1_6;
538                s_pixel_data_3 <= s_pixel_register_s1_7;
539                s_pixel_data_4 <= s_pixel_register_s1_8;
540                s_pixel_data_5 <= s_pixel_register_s1_9;
541                s_pixel_data_6 <= s_pixel_register_s2_0;
542                s_pixel_data_7 <= s_pixel_register_s2_1;
543 
544                s_pixel_data_8  <= s_pixel_register_s2_2;
545                s_pixel_data_9  <= s_pixel_register_s2_3;
546                s_pixel_data_10 <= s_pixel_register_s2_4;
547                s_pixel_data_11 <= s_pixel_register_s2_5;
548                s_pixel_data_12 <= s_pixel_register_s2_6;
549                s_pixel_data_13 <= s_pixel_register_s2_7;
550                s_pixel_data_14 <= s_pixel_register_s2_8;
551                s_pixel_data_15 <= s_pixel_register_s2_9;
552 
553                s_pixel_register_s3_0 <= s_data_0_in;
554                s_pixel_register_s3_1 <= s_data_1_in;
555                s_pixel_register_s3_2 <= s_data_2_in;
556                s_pixel_register_s3_3 <= s_data_3_in;
557                s_pixel_register_s3_4 <= s_data_4_in;
558                s_pixel_register_s3_5 <= s_data_5_in;
559                s_pixel_register_s3_6 <= s_data_6_in;
560                s_pixel_register_s3_7 <= s_data_7_in;
561                s_pixel_register_s3_8 <= s_data_8_in;
562                s_pixel_register_s3_9 <= s_data_9_in;
563 
564               when others =>
565                  s_pixel_data_0 <= (others => '0');
566                  s_pixel_data_1 <= (others => '0');
567                  s_pixel_data_2 <= (others => '0');
568                  s_pixel_data_3 <= (others => '0');
569                  s_pixel_data_4 <= (others => '0');
570                  s_pixel_data_5 <= (others => '0');
571                  s_pixel_data_6 <= (others => '0');
572                  s_pixel_data_7 <= (others => '0');
573 
574                  s_pixel_data_8 <= (others => '0');
575                  s_pixel_data_9 <= (others => '0');
576                  s_pixel_data_10 <= (others => '0');
577                  s_pixel_data_11 <= (others => '0');
```

```
578                 s_pixel_data_12 <= (others => '0');
579                 s_pixel_data_13 <= (others => '0');
580                 s_pixel_data_14 <= (others => '0');
581                 s_pixel_data_15 <= (others => '0');
582 
583                 s_pixel_register_s0_0 <= (others => '0');
584                 s_pixel_register_s0_1 <= (others => '0');
585                 s_pixel_register_s0_2 <= (others => '0');
586                 s_pixel_register_s0_3 <= (others => '0');
587                 s_pixel_register_s0_4 <= (others => '0');
588                 s_pixel_register_s0_5 <= (others => '0');
589                 s_pixel_register_s0_6 <= (others => '0');
590                 s_pixel_register_s0_7 <= (others => '0');
591                 s_pixel_register_s0_8 <= (others => '0');
592                 s_pixel_register_s0_9 <= (others => '0');
593 
594                 s_pixel_register_s1_0 <= (others => '0');
595                 s_pixel_register_s1_1 <= (others => '0');
596                 s_pixel_register_s1_2 <= (others => '0');
597                 s_pixel_register_s1_3 <= (others => '0');
598                 s_pixel_register_s1_4 <= (others => '0');
599                 s_pixel_register_s1_5 <= (others => '0');
600                 s_pixel_register_s1_6 <= (others => '0');
601                 s_pixel_register_s1_7 <= (others => '0');
602                 s_pixel_register_s1_8 <= (others => '0');
603                 s_pixel_register_s1_9 <= (others => '0');
604 
605                 s_pixel_register_s2_0 <= (others => '0');
606                 s_pixel_register_s2_1 <= (others => '0');
607                 s_pixel_register_s2_2 <= (others => '0');
608                 s_pixel_register_s2_3 <= (others => '0');
609                 s_pixel_register_s2_4 <= (others => '0');
610                 s_pixel_register_s2_5 <= (others => '0');
611                 s_pixel_register_s2_6 <= (others => '0');
612                 s_pixel_register_s2_7 <= (others => '0');
613                 s_pixel_register_s2_8 <= (others => '0');
614                 s_pixel_register_s2_9 <= (others => '0');
615 
616                 s_pixel_register_s3_0 <= (others => '0');
617                 s_pixel_register_s3_1 <= (others => '0');
618                 s_pixel_register_s3_2 <= (others => '0');
619                 s_pixel_register_s3_3 <= (others => '0');
620                 s_pixel_register_s3_4 <= (others => '0');
621                 s_pixel_register_s3_5 <= (others => '0');
622                 s_pixel_register_s3_6 <= (others => '0');
623                 s_pixel_register_s3_7 <= (others => '0');
624                 s_pixel_register_s3_8 <= (others => '0');
625                 s_pixel_register_s3_9 <= (others => '0');
```

```
626                 end case;
627         end process;
628 
629     -- Averages Pixels 1-8, 9-16, 17-24, 25-32
630     pixel_averaging_wirer_1 : component pixel_averaging
631         port map (rst_n          => rst_n,
632                   component_clk  => component_clk,
633                   hps_enable_flag => hps_enable_flag,
634                   data_0_in      => s_pixel_data_0,
635                   data_1_in      => s_pixel_data_1,
636                   data_2_in      => s_pixel_data_2,
637                   data_3_in      => s_pixel_data_3,
638                   data_4_in      => s_pixel_data_4,
639                   data_5_in      => s_pixel_data_5,
640                   data_6_in      => s_pixel_data_6,
641                   data_7_in      => s_pixel_data_7,
642                   pixel_data_out => s_pixel_data_0_summed);
643 
644       -- Averages Pixels 33-40
645       pixel_averaging_wirer_2 : component pixel_averaging
646         port map (rst_n          => rst_n,
647                   component_clk  => component_clk,
648                   hps_enable_flag => hps_enable_flag,
649                   data_0_in      => s_pixel_data_8,
650                   data_1_in      => s_pixel_data_9,
651                   data_2_in      => s_pixel_data_10,
652                   data_3_in      => s_pixel_data_11,
653                   data_4_in      => s_pixel_data_12,
654                   data_5_in      => s_pixel_data_13,
655                   data_6_in      => s_pixel_data_14,
656                   data_7_in      => s_pixel_data_15,
657                   pixel_data_out => s_pixel_data_1_summed);
658 
659     --  Adds the location data to the pixel data
660     pixel_location_wirer : component pixel_location
661         port map(rst_n           => rst_n,
662                  component_clk   => component_clk,
663                  hps_enable_flag => hps_enable_flag,
664                  pixel_data_1_in => s_pixel_data_0_summed,
665                  pixel_data_2_in => s_pixel_data_1_summed,
666                  pixel_address_and_data_0_out =>
667                                     s_pixel_address_and_data_0,
668                  pixel_address_and_data_1_out =>
669                                     s_pixel_address_and_data_1,
670                  pixel_data_and_location_valid_flag_out =>
671                        f_pixel_data_and_location_valid,
672                  start_reading_flag_out =>
673                                        f_start_reading,
```

```
674                     new_frame_flag_out =>
675                             f_new_frame_pixel_location);
676
677     -- Parallelizes all 5 super pixel data packets into
678     -- single packet
679     pixel_parallelization_wirer : component pixel_parallelization
680         port map(rst_n => rst_n,
681                  component_clk => component_clk,
682                  dram_clk => dram_clk,
683                  hps_enable_flag => hps_enable_flag,
684                  pixel_address_and_data_0_in =>
685                                  s_pixel_address_and_data_0,
686                  pixel_address_and_data_1_in =>
687                                  s_pixel_address_and_data_1,
688                  buffer_empty_in    => s_buffer_empty,
689                  buffer_read_out    => s_buffer_read,
690                  buffer_readdata_in => s_buffer_readdata,
691                  data_packet_out => data_packet_out,
692                  pixel_data_and_location_valid_flag_in =>
693                        f_pixel_data_and_location_valid,
694                  start_reading_flag_in      => f_start_reading,
695                  clear_pixel_flag_out       => f_clear_pixel,
696                  new_frame_flag_out         =>
697                          f_new_frame_parallelization,
698                  super_pixel_valid_flag_out =>
699                                     f_super_pixel_valid);
700
701     -- Route the flags to the outputs
702     clear_pixel_flag_out <= f_clear_pixel;
703     pixel_data_and_location_valid_flag_out <=
704                       f_pixel_data_and_location_valid;
705     new_frame_flag_out  <= f_new_frame_parallelization;
706     buffer_read_out     <= s_buffer_read;
707     super_pixel_valid_flag_out <= f_super_pixel_valid;
708
709 end architecture;
```

```
1  ----------------------------------------------------------------------
2  --
3  --! @file        pixel_averaging.vhd
4  --! @brief       The pixel averaging pipeline.
5  --
6  --! @author      Connor Dack
7  --! @date        May 2015
8  --! @copyright   Copyright (C) 2015 Ross K. Snider and Connor Dack
9  --
10 --   This program is free software: you can redistribute it
11 --   and/or modify it under the terms of the GNU General
```

```
12 --   Public License as published by the Free Software Foundation,
13 --   either version 3 of the License, or (at your option) any
14 --   later version.
15 --
16 --   This program is distributed in the hope that it will be
17 --   useful, but WITHOUT ANY WARRANTY; without even the implied
18 --   warranty of MERCHANTABILITY or FITNESS for A PARTICULAR
19 --   PURPOSE. See the GNU General Public License for more details.
20 --
21 --   You should have received a copy of the GNU General Public
22 --   License along with this program.  If not, see
23 --   <http://www.gnu.org/licenses/>.
24 --
25 --   Connor Dack
26 --   Electrical and Computer Engineering
27 --   Montana State University
28 --   610 Cobleigh Hall
29 --   Bozeman, MT 59717
30 --   connor.dack@msu.montana.edu
31 --
32 ----------------------------------------------------------------------------
33
34 library IEEE;
35 use IEEE.STD_LOGIC_1164.ALL;
36 use IEEE.NUMERIC_STD.ALL;
37
38 use work.Sensor_Package.ALL;
39
40 ----------------------------------------------------------------------------
41 --
42 --! @brief     pixel_averaging
43 --!
44 --! @param     rst_n                   -  Input: Active low reset
45 --! @param     component_clk           -  Input: Component Clock
46 --! @param     hps_enable_flag         -  Input: HPS Enable
47 --! @param     data_0_in               -  Input: Data 0
48 --! @param     data_1_in               -  Input: Data 1
49 --! @param     data_2_in               -  Input: Data 2
50 --! @param     data_3_in               -  Input: Data 3
51 --! @param     data_4_in               -  Input: Data 4
52 --! @param     data_5_in               -  Input: Data 5
53 --! @param     data_6_in               -  Input: Data 6
54 --! @param     data_7_in               -  Input: Data 7
55 --! @param     data_8_in               -  Input: Data 8
56 --! @param     input_data_9            -  Input: Data 9
57 --! @param     pixel_data_out          - Output: Summed Pixel Data
58 --
59 ----------------------------------------------------------------------------
```

```
60
61 entity pixel_averaging is
62     port(rst_n           : in std_logic;
63          component_clk   : in std_logic;
64          hps_enable_flag : in std_logic;
65          data_0_in : in std_logic_vector(INPUT_DATA_SIZE - 1
66                                                  downto 0);
67          data_1_in : in std_logic_vector(INPUT_DATA_SIZE - 1
68                                                  downto 0);
69          data_2_in : in std_logic_vector(INPUT_DATA_SIZE - 1
70                                                  downto 0);
71          data_3_in : in std_logic_vector(INPUT_DATA_SIZE - 1
72                                                  downto 0);
73          data_4_in : in std_logic_vector(INPUT_DATA_SIZE - 1
74                                                  downto 0);
75          data_5_in : in std_logic_vector(INPUT_DATA_SIZE - 1
76                                                  downto 0);
77          data_6_in : in std_logic_vector(INPUT_DATA_SIZE - 1
78                                                  downto 0);
79          data_7_in : in std_logic_vector(INPUT_DATA_SIZE - 1
80                                                  downto 0);
81          pixel_data_out : out std_logic_vector (DATA_SIZE - 1
82                                                 downto 0));
83 end entity;
84
85 architecture pixel_averaging_arch of pixel_averaging is
86
87     -- Internal Signals for the inputs
88     signal s_data_0 : std_logic_vector(INPUT_DATA_SIZE -1
89                                                  downto 0);
90     signal s_data_1 : std_logic_vector(INPUT_DATA_SIZE -1
91                                                  downto 0);
92     signal s_data_2 : std_logic_vector(INPUT_DATA_SIZE -1
93                                                  downto 0);
94     signal s_data_3 : std_logic_vector(INPUT_DATA_SIZE -1
95                                                  downto 0);
96     signal s_data_4 : std_logic_vector(INPUT_DATA_SIZE -1
97                                                  downto 0);
98     signal s_data_5 : std_logic_vector(INPUT_DATA_SIZE -1
99                                                  downto 0);
100    signal s_data_6 : std_logic_vector(INPUT_DATA_SIZE -1
101                                                 downto 0);
102    signal s_data_7 : std_logic_vector(INPUT_DATA_SIZE -1
103                                                 downto 0);
104    -- Signals for the different steps in the addition tree
105    signal s_pixel_sum_partial_data_0_1 : unsigned(
106                                    REQUIRED_DATA_ACCURACY - 1
107                                    downto 0) := (others => '0');
```

```
108     signal s_pixel_sum_partial_data_2_3 : unsigned(
109                                         REQUIRED_DATA_ACCURACY - 1
110                                         downto 0) := (others => '0');
111     signal s_pixel_sum_partial_data_4_5 : unsigned(
112                                         REQUIRED_DATA_ACCURACY - 1
113                                         downto 0) := (others => '0');
114     signal s_pixel_sum_partial_data_6_7 : unsigned(
115                                         REQUIRED_DATA_ACCURACY - 1
116                                         downto 0) := (others => '0');
117     signal s_pixel_sum_partial_data_0_1_2_3 : unsigned(
118                                         REQUIRED_DATA_ACCURACY - 1
119                                         downto 0) := (others => '0');
120     signal s_pixel_sum_partial_data_4_5_6_7 : unsigned(
121                                         REQUIRED_DATA_ACCURACY - 1
122                                         downto 0) := (others => '0');
123
124     -- Result Signal
125     signal s_pixel_data :std_logic_vector(DATA_SIZE -1 downto 0);
126
127     begin
128
129     -- Assign he input signals to internal signals
130     s_data_0 <= data_0_in;
131     s_data_1 <= data_1_in;
132     s_data_2 <= data_2_in;
133     s_data_3 <= data_3_in;
134     s_data_4 <= data_4_in;
135     s_data_5 <= data_5_in;
136     s_data_6 <= data_6_in;
137     s_data_7 <= data_7_in;
138
139     ------------------------------------------------------------------
140     -- Addition Tree: Level One
141     ------------------------------------------------------------------
142     -- Adds pixel data 0 and 1, 2 and 3, 4 and 5, and 6 and 7
143     -- The results are registered on the rising edge of the
144     -- component clock
145     pixel_averaging_step_1 : process(rst_n,component_clk)
146         begin
147             if(rst_n = '0') then
148                 s_pixel_sum_partial_data_0_1 <= (others => '0');
149                 s_pixel_sum_partial_data_2_3 <= (others => '0');
150                 s_pixel_sum_partial_data_4_5 <= (others => '0');
151                 s_pixel_sum_partial_data_6_7 <= (others => '0');
152             elsif(rising_edge(component_clk)) then
153                 if(hps_enable_flag = '1') then
154                     s_pixel_sum_partial_data_0_1 <= resize(
155                                         unsigned(s_data_0),
```

```
156                                     REQUIRED_DATA_ACCURACY) +
157                                     resize(unsigned(s_data_1),
158                                     REQUIRED_DATA_ACCURACY);
159                     s_pixel_sum_partial_data_2_3 <= resize(
160                                     unsigned(s_data_2),
161                                     REQUIRED_DATA_ACCURACY) +
162                                     resize(unsigned(s_data_3),
163                                     REQUIRED_DATA_ACCURACY);
164                     s_pixel_sum_partial_data_4_5 <= resize(
165                                     unsigned(s_data_4),
166                                     REQUIRED_DATA_ACCURACY) +
167                                     resize(unsigned(s_data_5),
168                                     REQUIRED_DATA_ACCURACY);
169                     s_pixel_sum_partial_data_6_7 <= resize(
170                                     unsigned(s_data_6),
171                                     REQUIRED_DATA_ACCURACY) +
172                                     resize(unsigned(s_data_7),
173                                     REQUIRED_DATA_ACCURACY);
174                 end if;
175             end if;
176         end process;
177
178     ------------------------------------------------------------------
179     -- Addition Tree: Level Two
180     ------------------------------------------------------------------
181     -- Adds pixel data 0_1 and 2_3, 4_5 and 6_7
182     -- The results are registered on the rising edge
183     -- of the component clock
184     pixel_averaging_step_2 : process(rst_n,component_clk)
185         begin
186             if(rst_n = '0') then
187              s_pixel_sum_partial_data_0_1_2_3 <= (others => '0');
188              s_pixel_sum_partial_data_4_5_6_7 <= (others => '0');
189             elsif(rising_edge(component_clk)) then
190                 if(hps_enable_flag = '1') then
191                     s_pixel_sum_partial_data_0_1_2_3 <=
192                                     s_pixel_sum_partial_data_0_1 +
193                                     s_pixel_sum_partial_data_2_3;
194                     s_pixel_sum_partial_data_4_5_6_7 <=
195                                     s_pixel_sum_partial_data_4_5 +
196                                     s_pixel_sum_partial_data_6_7;
197                 end if;
198             end if;
199         end process;
200
201     ------------------------------------------------------------------
202     -- Addition Tree: Level Three (Final)
203     ------------------------------------------------------------------
```

```
204     -- Adds pixel data 0_1_2_3 and 4_5_6_7
205     -- The results are registered on the rising edge of the
206     -- component clock
207     pixel_averaging_step_3 : process(rst_n,component_clk)
208         begin
209             if(rst_n = '0') then
210                 s_pixel_data <= (others => '0');
211             elsif(rising_edge(component_clk)) then
212                 if(hps_enable_flag = '1') then
213                     s_pixel_data <= std_logic_vector(resize(
214                                 s_pixel_sum_partial_data_0_1_2_3 +
215                                 s_pixel_sum_partial_data_4_5_6_7,
216                                   DATA_SIZE));
217                 end if;
218             end if;
219         end process;
220
221     -- Assign the output
222     pixel_data_out <= s_pixel_data;
223 end architecture;
```

```
1  ----------------------------------------------------------------------
2  --
3  --! @file        pixel_location.vhd
4  --! @brief       Adds the location data to the pixel data
5  --
6  --! @author      Connor Dack
7  --! @date        May 2015
8  --! @copyright   Copyright (C) 2015 Ross K. Snider and Connor Dack
9  --
10 --   This program is free software: you can redistribute it
11 --   and/or modify it under the terms of the GNU General
12 --   Public License as published by the Free Software Foundation,
13 --   either version 3 of the License, or (at your option) any
14 --   later version.
15 --
16 --   This program is distributed in the hope that it will be
17 --   useful, but WITHOUT ANY WARRANTY; without even the implied
18 --   warranty of MERCHANTABILITY or FITNESS for A PARTICULAR
19 --   PURPOSE. See the GNU General Public License for more details.
20 --
21 --   You should have received a copy of the GNU General Public
22 --   License along with this program.  If not, see
23 --   <http://www.gnu.org/licenses/>.
24 --
25 --   Connor Dack
26 --   Electrical and Computer Engineering
27 --   Montana State University
```

```
28 --   610 Cobleigh Hall
29 --   Bozeman, MT 59717
30 --   connor.dack@msu.montana.edu
31 --
32 ----------------------------------------------------------------------------------

33
34 library IEEE;
35 use IEEE.STD_LOGIC_1164.ALL;
36 use IEEE.NUMERIC_STD.ALL;
37
38 use work.Sensor_Package.ALL;
39 ----------------------------------------------------------------------------------
40 --
41 --! @brief     pixel_location
42 --!
43 --! @param     component_clk              -  Input: Component Clock
44 --! @param     rst_n                      -  Input: Active low reset
45 --! @param     pixel_data_1_in            -  Input: Data in 1
46 --! @param     pixel_data_2_in            -  Input: Data in 2
47 --! @param     pixel_address_and_data_0_out - Output: Data Out 0
48 --! @param     pixel_address_and_data_1_out - Output: Data Out 1
49 --! @param     pixel_data_and_location_valid_flag_out - Output:
    Valid
50 --! @param     start_reading_flag_out       - Output: Read Enabled
51 --! @param     new_frame_flag_out           - Output: New Frame
    Flag
52 --
53 ----------------------------------------------------------------------------------

54
55 entity pixel_location is
56   port(component_clk    : in std_logic;
57        rst_n            : in std_logic;
58            hps_enable_flag : in std_logic;
59        pixel_data_1_in : in std_logic_vector (DATA_SIZE - 1
60                                                downto 0);
61            pixel_data_2_in : in std_logic_vector (DATA_SIZE - 1
62                                                downto 0);
63        pixel_address_and_data_0_out : out std_logic_vector(
64                                (DATA_PACKAGE_SIZE) - 1 downto 0);
65        pixel_address_and_data_1_out : out std_logic_vector(
66                                (DATA_PACKAGE_SIZE) - 1 downto 0);
67            pixel_data_and_location_valid_flag_out : out std_logic;
68            start_reading_flag_out : out std_logic;
69            new_frame_flag_out     : out std_logic);
70 end entity;
71
72 architecture rtl of pixel_location is
73
```

```
74    --Frame Size converted from Integer to Binary
75    constant c_pixel_condition : unsigned(PIXEL_ADDRESS_SIZE - 1
76                                  downto 0) := to_unsigned
77                                  ((NUMBER_OF_PIXELS)- 1,
78                                  PIXEL_ADDRESS_SIZE);
79    constant c_spectral_bin_condition : unsigned (
80                                     SPECTRAL_BIN_ADDRESS_SIZE - 1
81                                     downto 0) := to_unsigned(
82                                     NUMBER_OF_SPECTRAL_BINS,
83                                     SPECTRAL_BIN_ADDRESS_SIZE);
84      constant c_pixel_counter_zeros : std_logic_vector(
85                                      PIXEL_ADDRESS_SIZE - 1
86                                     downto 0) := (others => '0');
87      constant c_spectral_bin_counter_zeros: unsigned(
88                                          SPECTRAL_BIN_ADDRESS_SIZE
89                                             - 1 downto 0) :=
90                                             (others => '0');
91
92      -- Signals for tracking the pixel location
93      signal s_pixel_counter : unsigned (PIXEL_ADDRESS_SIZE - 1
94                                  downto 0) := c_pixel_condition;
95    signal s_spectral_bin_counter : unsigned(
96                                     SPECTRAL_BIN_ADDRESS_SIZE - 1
97                                     downto 0) :=
98                                     c_spectral_bin_condition;
99
100     -- Internal Component Signals
101
102   signal s_pixel_data_1 : std_logic_vector (DATA_SIZE - 1
103                                                 downto 0);
104     signal s_pixel_data_2 : std_logic_vector (DATA_SIZE - 1
105                                                 downto 0);
106
107     -- More signals for the pixel loaction
108     signal s_pixel_data_and_location_valid_flag :std_logic:= '0';
109     -- used to start the pixel parallelization block
110     signal s_delay_counter         : integer range 0 to 5 := 0;
111     -- delay the necessary amount to get the first result from
112     -- the averaging block
113     signal f_start_reading: std_logic := '0';
114     -- asserts a signal to signal a new frame
115     signal s_second_output_counter  : integer range 0 to 5 := 0;
116     -- delay to align the second output every four clock cycles
117
118     -- Output Signals
119   signal s_pixel_address_and_data_0 : std_logic_vector(
120                              (DATA_PACKAGE_SIZE) - 1 downto 0);
121   signal s_pixel_address_and_data_1 : std_logic_vector(
```

```
122                                 (DATA_PACKAGE_SIZE) - 1 downto 0);
123
124     begin
125
126           s_pixel_data_1 <= pixel_data_1_in;
127           s_pixel_data_2 <= pixel_data_2_in;
128
129           -- Process to count clock cycles to know
130           -- when to use the second output
131          second_output_cntr_process : process(component_clk, rst_n)
132               begin
133                    if(rst_n = '0') then
134                         s_second_output_counter <= 0;
135                    elsif(rising_edge(component_clk)) then
136                         -- Make sure the HPS has enabled the system
137                         -- and the data is valid
138                         if(hps_enable_flag = '1' and
139                           s_pixel_data_and_location_valid_flag = '1')
140                           then
141                             if(s_second_output_counter = 3) then
142                             -- count 4 clock cycles
143                                  s_second_output_counter <= 0;
144                                  -- reset the counter
145                             else
146                                  -- Increase the counter by 1
147                                  s_second_output_counter <=
148                                                 s_second_output_counter
149                                                 + 1;
150                             end if;
151                         end if;
152                    end if;
153               end process;
154
155           -- Pixel Location Process
156       pixel_location_loop : process (component_clk, rst_n)
157           begin
158               -- Reset Conditions
159               if(rst_n = '0') then
160                    s_pixel_counter        <= c_pixel_condition;
161                    s_spectral_bin_counter <=
162                                           c_spectral_bin_condition;
163                    s_pixel_data_and_location_valid_flag <= '0';
164                    s_delay_counter             <=  0 ;
165                    f_start_reading        <= '0';
166                    s_pixel_address_and_data_0  <= (others => '0');
167                    s_pixel_address_and_data_1  <= (others => '0');
168               -- On the rising edge of the component clock,
169               -- add the pixel location
```

```
170             elsif(rising_edge(component_clk)) then
171                 -- Make sure the HPS Enabled the system
172                 if(hps_enable_flag = '1') then
173                     -- If the counter is greater than four
174                     -- (5 clock cycles), Set the data and
175                     -- location flag
176                     if(4 < s_delay_counter) then
177                       s_pixel_data_and_location_valid_flag <= '1';
178                     -- After four clock cycles, set the read flag
179                     elsif(4 = s_delay_counter) then
180                         f_start_reading <= '1';
181                         s_delay_counter <= s_delay_counter + 1;
182                     -- Count clock cycles until reaching four
183                     else
184                         s_delay_counter <= s_delay_counter + 1;
185                     end if;
186
187                     -- At the end of the frame, roll over
188                     -- the counters
189                     if(c_pixel_condition = s_pixel_counter and
190                        c_spectral_bin_condition =
191                        s_spectral_bin_counter and
192                        s_pixel_data_and_location_valid_flag='1')
193                        then
194
195                       s_pixel_counter <= (others => '0');
196                       s_spectral_bin_counter <= to_unsigned(1,
197                                         SPECTRAL_BIN_ADDRESS_SIZE);
198                       s_pixel_address_and_data_0 <=
199                                         std_logic_vector(
200                                         c_spectral_bin_counter_zeros)
201                                        & c_pixel_counter_zeros
202                                        & s_pixel_data_1;
203                       s_pixel_address_and_data_1 <=
204                                         std_logic_vector(
205                                         c_spectral_bin_counter_zeros
206                                        + 1) & c_pixel_counter_zeros
207                                        & s_pixel_data_2;
208                       f_start_reading <= '1';
209                      -- set the new frame flag
210
211                     -- At the end of the pixel, reset the bin and
212                     -- increase the pixel location by 1
213                     elsif(c_spectral_bin_condition =
214                               s_spectral_bin_counter and
215                               s_pixel_data_and_location_valid_flag
216                              = '1')then
217
```

```
218                     s_pixel_counter <= s_pixel_counter + 1;
219                     s_spectral_bin_counter  <= to_unsigned(1,
220                                         SPECTRAL_BIN_ADDRESS_SIZE);
221
222                     -- Output the pixel data
223                     s_pixel_address_and_data_0 <=
224                                 std_logic_vector(
225                                 c_spectral_bin_counter_zeros) &
226                                 std_logic_vector(s_pixel_counter
227                                 + 1) & s_pixel_data_1;
228                     s_pixel_address_and_data_1 <=
229                                 std_logic_vector(
230                                 c_spectral_bin_counter_zeros + 1)
231                                 & std_logic_vector(s_pixel_counter
232                                 + 1) & s_pixel_data_2;
233                     f_start_reading <= '0';
234                     -- clear the flag
235
236                   -- Increment the bin counter
237                   elsif(s_pixel_data_and_location_valid_flag
238                                                 = '1') then
239                     -- If outputting two values, increment
240                     -- the bin counter by 2
241                     if(s_second_output_counter = 3) then
242                       s_spectral_bin_counter <=
243                                         s_spectral_bin_counter + 2;
244                     -- Increment the bin counter by 1
245                     else
246                       s_spectral_bin_counter <=
247                                         s_spectral_bin_counter + 1;
248                     end if;
249
250                     -- Output the pixel data
251                     s_pixel_address_and_data_0 <=
252                         std_logic_vector(s_spectral_bin_counter)
253                         & std_logic_vector(s_pixel_counter)
254                         & s_pixel_data_1;
255                     s_pixel_address_and_data_1 <=
256                         std_logic_vector(s_spectral_bin_counter
257                         + 1) & std_logic_vector(s_pixel_counter)
258                         & s_pixel_data_2;
259                     f_start_reading <= '0';
260                     -- clear the flag
261                   end if;
262                 end if;
263               end if;
264             end process;
265
```

```
266         -- Assign signals to output ports
267         pixel_address_and_data_0_out <=
268                                         s_pixel_address_and_data_0;
269         pixel_address_and_data_1_out <=
270                                         s_pixel_address_and_data_1;
271         new_frame_flag_out <= f_start_reading;
272         pixel_data_and_location_valid_flag_out <=
273                                 s_pixel_data_and_location_valid_flag;
274         start_reading_flag_out  <= f_start_reading;
275
276 end architecture;
```

```
1  ----------------------------------------------------------------------
2  --
3  --! @file        pixel_parallelization.vhd
4  --! @brief       The pixel parallelization step.
5  --! @details     Parallelizes the pixels and combines the DRAM
6  --!              data based on the location data.
7  --
8  --! @author      Connor Dack
9  --! @date        May 2015
10 --! @copyright   Copyright (C) 2015 Ross K. Snider and Connor Dack
11 --
12 --   This program is free software: you can redistribute it
13 --   and/or modify it under the terms of the GNU General
14 --   Public License as published by the Free Software Foundation,
15 --   either version 3 of the License, or (at your option) any
16 --   later version.
17 --
18 --   This program is distributed in the hope that it will be
19 --   useful, but WITHOUT ANY WARRANTY; without even the implied
20 --   warranty of MERCHANTABILITY or FITNESS for A PARTICULAR
21 --   PURPOSE. See the GNU General Public License for more details.
22 --
23 --   You should have received a copy of the GNU General Public
24 --   License along with this program.  If not, see
25 --   <http://www.gnu.org/licenses/>.
26 --
27 --   Connor Dack
28 --   Electrical and Computer Engineering
29 --   Montana State University
30 --   610 Cobleigh Hall
31 --   Bozeman, MT 59717
32 --   connor.dack@msu.montana.edu
33 --
34 ----------------------------------------------------------------------
35
36 library IEEE;
```

```
37 use IEEE.STD_LOGIC_1164.ALL;
38 use IEEE.NUMERIC_STD.ALL;
39
40 use work.Sensor_Package.ALL;
41 ----------------------------------------------------------------------------
42 --
43 --! @brief     pixel_parallelization
44 --
45 --! @param     rst_n                          - Input: Active low rst_n
46 --! @param     component_clk                  - Input: Component Clk
47 --! @param     dram_clk                       - Input: DRAM Clk
48 --! @param     hps_enable_flag                - Input: HPS Enable
49 --! @param     pixel_address_and_data_0_in - Input: DRAM Data 1
50 --! @param     pixel_address_and_data_1_in - Input: DRAM Data 2
51 --! @param     buffer_empty_in                - Input: Buffer Empty
52 --! @param     buffer_read_out                - Output: Buffer Read En
53 --! @param     buffer_readdata_in             - Input: Buffer Read Data
54 --! @param     data_packet_out                - Output: Data Packet
55 --! @param     pixel_data_and_location_valid_flag_in - Input: Flag
56 --! @param     start_reading_flag_in    - Input: Start Reading Flag
57 --! @param     clear_pixel_flag_out     - Output: Clear Pixel Flag
58 --! @param     new_frame_flag_out          - Output: New Frame Flag
59 --! @param     super_pixel_valid_flag_out- Output: Data Valid Flag
60 --
61 ----------------------------------------------------------------------------
62
63 entity pixel_parallelization is
64    port(rst_n        : in std_logic;
65         component_clk    : in std_logic;
66         dram_clk         : in std_logic;
67         hps_enable_flag  : in std_logic;
68         pixel_address_and_data_0_in : in std_logic_vector(
69                                (DATA_PACKAGE_SIZE) - 1 downto 0);
70         pixel_address_and_data_1_in : in std_logic_vector(
71                                (DATA_PACKAGE_SIZE) - 1 downto 0);
72         buffer_empty_in     : in std_logic;
73         buffer_read_out     : out std_logic;
74         buffer_readdata_in  : in std_logic_vector(DRAM_DATA_SIZE
75                                                   - 1 downto 0);
76         data_packet_out     : out std_logic_vector(((
77                                  SUPER_PIXEL_SIZE *
78                                  NUMBER_OF_PARALLEL_CHANNELS) - 1)
79                                  downto 0);
80         pixel_data_and_location_valid_flag_in : in std_logic;
81         start_reading_flag_in       : in std_logic;
82         clear_pixel_flag_out        : out std_logic;
83         new_frame_flag_out          : out std_logic;
84         super_pixel_valid_flag_out  : out std_logic);
```

```
85  end entity;
86
87  architecture rtl of pixel_parallelization is
88
89      -- Internal Input Signals
90      signal s_pixel_address_and_data_0 : std_logic_vector(
91                                  (DATA_PACKAGE_SIZE) - 1 downto 0);
92      signal s_pixel_address_and_data_1 : std_logic_vector(
93                                  (DATA_PACKAGE_SIZE) - 1 downto 0);
94      signal s_buffer_empty     : std_logic;
95      signal s_buffer_readdata  : std_logic_vector(DRAM_DATA_SIZE
96                                                      - 1 downto 0);
97      signal s_pixel_data_and_location_valid_flag : std_logic;
98      signal f_start_reading    : std_logic;
99
100     -- Internal Signals
101     signal f_first_data          : std_logic := '0';
102     signal f_super_pixel_valid   : std_logic := '0';
103     signal f_new_frame           : std_logic := '0';
104     signal f_clear_pixel         : std_logic := '1';
105
106     signal s_data_register_0 : std_logic_vector((
107                                   SUPER_PIXEL_SIZE - 1) downto 0)
108                                   := (others => '0');
109     signal s_data_register_1 : std_logic_vector(((
110                                   SUPER_PIXEL_SIZE * 2)- 1) downto
111                                   0) := (others => '0');
112     signal s_data_register_2 : std_logic_vector(((
113                                   SUPER_PIXEL_SIZE * 3)- 1) downto
114                                   0) := (others => '0');
115     signal s_data_register_3 : std_logic_vector(((
116                                   SUPER_PIXEL_SIZE * 5)- 1) downto
117                                   0) := (others => '0');
118
119     signal s_dram_data_0      : std_logic_vector(DRAM_DATA_SIZE
120                                                      - 1 downto 0);
121     signal s_dram_data_1      : std_logic_vector(DRAM_DATA_SIZE
122                                                      - 1 downto 0);
123     signal s_dram_data_2      : std_logic_vector(DRAM_DATA_SIZE
124                                                      - 1 downto 0);
125     signal s_dram_data_3      : std_logic_vector(DRAM_DATA_SIZE
126                                                      - 1 downto 0);
127     signal s_dram_data_4      : std_logic_vector(DRAM_DATA_SIZE
128                                                      - 1 downto 0);
129
130     signal s_dram_rdreq_0      : std_logic := '0';
131     signal s_dram_rdreq_1      : std_logic := '0';
132     signal s_dram_rdreq_2      : std_logic := '0';
```

```
133     signal s_dram_rdreq_3      : std_logic := '0';
134     signal s_dram_rdreq_4      : std_logic := '0';
135
136     signal s_dram_wrreq_0      : std_logic := '0';
137     signal s_dram_wrreq_1      : std_logic := '0';
138     signal s_dram_wrreq_2      : std_logic := '0';
139     signal s_dram_wrreq_3      : std_logic := '0';
140     signal s_dram_wrreq_4      : std_logic := '0';
141
142     signal s_dram_full_0       : std_logic := 'X';
143     signal s_dram_full_1       : std_logic := 'X';
144     signal s_dram_full_2       : std_logic := 'X';
145     signal s_dram_full_3       : std_logic := 'X';
146     signal s_dram_full_4       : std_logic := 'X';
147
148     signal s_dram_empty_0      : std_logic := 'X';
149     signal s_dram_empty_1      : std_logic := 'X';
150     signal s_dram_empty_2      : std_logic := 'X';
151     signal s_dram_empty_3      : std_logic := 'X';
152     signal s_dram_empty_4      : std_logic := 'X';
153
154     signal s_buffer_read : std_logic := '0';
155
156     signal s_pixel_location_check : unsigned(PIXEL_ADDRESS_SIZE
157                                 - 1 downto 0) := to_unsigned((
158                                 NUMBER_OF_PIXELS) - 1,
159                                 PIXEL_ADDRESS_SIZE);
160     constant c_spectral_bin_zeros : std_logic_vector(
161                                 SPECTRAL_BIN_ADDRESS_SIZE - 1
162                                 downto 0) := (others => '0');
163     constant c_pixel_zeros        : std_logic_vector(
164                                 PIXEL_ADDRESS_SIZE - 1 downto
165                                 0) := (others => '0');
166     constant c_pixel_condition    : unsigned(PIXEL_ADDRESS_SIZE
167                                 - 1 downto 0) := to_unsigned((
168                                 NUMBER_OF_PIXELS) - 1,
169                                 PIXEL_ADDRESS_SIZE);
170
171     -- State Machine Types
172     type state_location_check is (S0_location_check,
173     S1_location_check, S2_location_check, S3_location_check,
174     S_Start_location_check);
175     type state_FIFO_read is (S0_FIFO_read, S1_FIFO_read,
176     S2_FIFO_read, S3_FIFO_read);
177     type state_DRAM_read is (S0_DRAM_read, S1_DRAM_read,
178     S2_DRAM_read, S3_DRAM_read, S4_DRAM_read, S_Start_DRAM_read);
179
180     -- State Machine Signals
```

```
181   signal s_current_state_location_check : state_location_check
182                                         := S_Start_location_check;
183     signal s_next_state_location_check    : state_location_check;
184     signal s_current_state_fifo_read : state_FIFO_read
185                                              := S3_FIFO_read;
186     signal s_next_state_fifo_read    : state_FIFO_read;
187     signal s_current_state_dram_read : state_DRAM_read
188                                                := S_Start_DRAM_read;
189     signal s_next_state_dram_read    : state_DRAM_read;
190
191    component dual_clock_fifo is
192         generic(
193             enable_ecc             : string := "FALSE";
194             lpm_numwords           : natural;
195             lpm_width              : natural;
196             lpm_widthu             : natural;
197             rdsync_delaypipe       : natural;
198             wrsync_delaypipe       : natural);
199         port (
200             data      : in  std_logic_vector(lpm_width - 1
201                           downto 0) := (others => 'X'); -- datain
202             wrreq     : in  std_logic  := 'X';          -- wrreq
203             rdreq     : in  std_logic  := 'X';          -- rdreq
204             wrclk     : in  std_logic  := 'X';          -- wrclk
205             rdclk     : in  std_logic  := 'X';          -- rdclk
206             aclr      : in std_logic   := '0';
207             q         : out std_logic_vector(lpm_width - 1
208                             downto 0);  -- dataout
209             rdempty   : out std_logic;  -- rdempty
210             wrfull    : out std_logic;  -- wrfull
211             rdfull    : out std_logic;
212             wrempty   : out std_logic;
213             rdusedw   : out std_logic_vector(lpm_widthu - 1
214                             downto 0);
215             wrusedw   : out std_logic_vector(lpm_widthu - 1
216                             downto 0);
217             eccstatus : out std_logic_vector(1 downto 0));
218     end component dual_clock_fifo;
219
220     begin
221     -- Assign inputs to internal signals
222     s_pixel_address_and_data_0 <= pixel_address_and_data_0_in;
223     s_pixel_address_and_data_1 <= pixel_address_and_data_1_in;
224     s_buffer_empty             <= buffer_empty_in;
225     s_buffer_readdata          <= buffer_readdata_in;
226     s_pixel_data_and_location_valid_flag <=
227                           pixel_data_and_location_valid_flag_in;
228     f_start_reading            <= start_reading_flag_in;
```

```
229
230     -- The DRAM State Machine - Current State to Next State
231     -- Logic - Trigger on the DRAM Clock and Reset
232     dram_start_proc : process (dram_clk, rst_n)
233      begin
234       if (rst_n = '0') then
235         s_current_state_dram_read <= S_Start_DRAM_Read;
236       elsif (rising_edge(dram_clk)) then
237                 if(hps_enable_flag = '1') then
238                     s_current_state_dram_read
239                                         <= s_next_state_dram_read;
240                 end if;
241       end if;
242     end process;
243
244     -- The DRAM Transition Logic - Trigger on the Current State
245     -- or any of the DRAM Full Signals
246     dram_state_trans_proc : process (s_current_state_dram_read,
247                                      s_dram_full_0,s_dram_full_1,
248                                      s_dram_full_2,s_dram_full_3,
249                                      s_dram_full_4)
250     begin
251             case s_current_state_dram_read is
252                 when S_Start_DRAM_Read =>
253                           s_next_state_dram_read <= S0_DRAM_Read;
254                 when S0_DRAM_Read      =>
255                     if(s_dram_full_1 = '0') then
256                         s_next_state_dram_read <= S1_DRAM_Read;
257                     else
258                         s_next_state_dram_read <= S0_DRAM_Read;
259                     end if;
260                 when S1_DRAM_Read      =>
261                     if(s_dram_full_2 = '0') then
262                         s_next_state_dram_read <= S2_DRAM_Read;
263                     else
264                         s_next_state_dram_read <= S1_DRAM_Read;
265                     end if;
266                 when S2_DRAM_Read      =>
267                     if(s_dram_full_3 = '0') then
268                         s_next_state_dram_read <= S3_DRAM_Read;
269                     else
270                         s_next_state_dram_read <= S2_DRAM_Read;
271                     end if;
272                 when S3_DRAM_Read      =>
273                     if(s_dram_full_4 = '0') then
274                         s_next_state_dram_read <= S4_DRAM_Read;
275                     else
276                         s_next_state_dram_read <= S3_DRAM_Read;
```

```
277                     end if;
278                 when S4_DRAM_Read       =>
279                     if(s_dram_full_0 = '0') then
280                         s_next_state_dram_read <= S0_DRAM_Read;
281                     else
282                         s_next_state_dram_read <= S4_DRAM_Read;
283                     end if;
284                 when others => s_next_state_dram_read
285                                         <= S_Start_DRAM_Read;
286             end case;
287     end process;
288
289     -- The DRAM Output Procees - Trigger on the current state,
290     -- the HPS enable, or any of the DRAM Full signals
291     dram_state_output_proc : process (s_current_state_dram_read,
292                                       hps_enable_flag,s_dram_full_0,
293                                       s_dram_full_1, s_dram_full_2,
294                                       s_dram_full_3, s_dram_full_4)
295     begin
296             case s_current_state_dram_read is
297                 when S_Start_DRAM_Read =>
298                     if(hps_enable_flag = '1') then
299                         s_dram_wrreq_0 <= '0';
300                         s_dram_wrreq_1 <= '0';
301                         s_dram_wrreq_2 <= '0';
302                         s_dram_wrreq_3 <= '0';
303                         s_dram_wrreq_4 <= '0';
304                         s_buffer_read  <= '1';
305                     else
306                         s_dram_wrreq_0 <= '0';
307                         s_dram_wrreq_1 <= '0';
308                         s_dram_wrreq_2 <= '0';
309                         s_dram_wrreq_3 <= '0';
310                         s_dram_wrreq_4 <= '0';
311                         s_buffer_read  <= '0';
312                      end if;
313                 when S0_DRAM_Read =>
314                     if(s_buffer_empty = '0' and
315                                           s_dram_full_0 = '0') then
316                         s_dram_wrreq_0 <= '1';
317                         s_dram_wrreq_1 <= '0';
318                         s_dram_wrreq_2 <= '0';
319                         s_dram_wrreq_3 <= '0';
320                         s_dram_wrreq_4 <= '0';
321                         s_buffer_read  <= '1';
322                      else
323                         s_dram_wrreq_0 <= '0';
324                         s_dram_wrreq_1 <= '0';
```

```
325                     s_dram_wrreq_2 <= '0';
326                     s_dram_wrreq_3 <= '0';
327                     s_dram_wrreq_4 <= '0';
328                     s_buffer_read  <= '0';
329                 end if;
330             when S1_DRAM_Read =>
331                 if(s_buffer_empty = '0' and
332                                     s_dram_full_1 = '0') then
333                     s_dram_wrreq_0 <= '0';
334                     s_dram_wrreq_1 <= '1';
335                     s_dram_wrreq_2 <= '0';
336                     s_dram_wrreq_3 <= '0';
337                     s_dram_wrreq_4 <= '0';
338                     s_buffer_read  <= '1';
339                 else
340                     s_dram_wrreq_0 <= '0';
341                     s_dram_wrreq_1 <= '0';
342                     s_dram_wrreq_2 <= '0';
343                     s_dram_wrreq_3 <= '0';
344                     s_dram_wrreq_4 <= '0';
345                     s_buffer_read  <= '0';
346                 end if;
347             when S2_DRAM_Read =>
348                 if(s_buffer_empty = '0' and
349                                     s_dram_full_2 = '0') then
350                     s_dram_wrreq_0 <= '0';
351                     s_dram_wrreq_1 <= '0';
352                     s_dram_wrreq_2 <= '1';
353                     s_dram_wrreq_3 <= '0';
354                     s_dram_wrreq_4 <= '0';
355                     s_buffer_read  <= '1';
356                 else
357                     s_dram_wrreq_0 <= '0';
358                     s_dram_wrreq_1 <= '0';
359                     s_dram_wrreq_2 <= '0';
360                     s_dram_wrreq_3 <= '0';
361                     s_dram_wrreq_4 <= '0';
362                     s_buffer_read  <= '0';
363                 end if;
364             when S3_DRAM_Read =>
365                 if(s_buffer_empty = '0' and
366                                     s_dram_full_3 = '0') then
367                     s_dram_wrreq_0 <= '0';
368                     s_dram_wrreq_1 <= '0';
369                     s_dram_wrreq_2 <= '0';
370                     s_dram_wrreq_3 <= '1';
371                     s_dram_wrreq_4 <= '0';
372                     s_buffer_read  <= '1';
```

```
373                     else
374                         s_dram_wrreq_0 <= '0';
375                         s_dram_wrreq_1 <= '0';
376                         s_dram_wrreq_2 <= '0';
377                         s_dram_wrreq_3 <= '0';
378                         s_dram_wrreq_4 <= '0';
379                         s_buffer_read  <= '0';
380                     end if;
381                 when S4_DRAM_Read =>
382                     if(s_buffer_empty = '0' and
383                                         s_dram_full_4 = '0') then
384                         s_dram_wrreq_0 <= '0';
385                         s_dram_wrreq_1 <= '0';
386                         s_dram_wrreq_2 <= '0';
387                         s_dram_wrreq_3 <= '0';
388                         s_dram_wrreq_4 <= '1';
389                         s_buffer_read  <= '1';
390                     else
391                         s_dram_wrreq_0 <= '0';
392                         s_dram_wrreq_1 <= '0';
393                         s_dram_wrreq_2 <= '0';
394                         s_dram_wrreq_3 <= '0';
395                         s_dram_wrreq_4 <= '0';
396                         s_buffer_read  <= '0';
397                     end if;
398                 when others =>
399                         s_dram_wrreq_0 <= '0';
400                         s_dram_wrreq_1 <= '0';
401                         s_dram_wrreq_2 <= '0';
402                         s_dram_wrreq_3 <= '0';
403                         s_dram_wrreq_4 <= '0';
404                         s_buffer_read  <= '0';
405             end case;
406         end process;
407
408     s_buffer_read <= s_buffer_read;
409
410     -- DRAM DATA FIFO
411     dram_data_0 : component dual_clock_fifo
412             generic map(
413                 lpm_numwords      => 1024,
414                 lpm_width         => DRAM_DATA_SIZE,
415                 lpm_widthu        => 10,
416                 rdsync_delaypipe => 3,
417                 wrsync_delaypipe => 3)
418             port map (
419                 data      => s_buffer_readdata,
420                 wrreq     => s_dram_wrreq_0,
```

```
421                 rdreq      => s_dram_rdreq_0,
422                 wrclk      => dram_clk,
423                 rdclk      => component_clk,
424                 q          => s_dram_data_0,
425                 rdempty    => s_dram_empty_0,
426                 wrfull     => s_dram_full_0,
427                 aclr       => open,
428                 eccstatus  => open,
429                 rdfull     => open,
430                 rdusedw    => open,
431                 wrempty    => open,
432                 wrusedw    => open);
433
434     -- DRAM DATA FIFO
435     dram_data_1 : component dual_clock_fifo
436             generic map(
437                 lpm_numwords      => 1024,
438                 lpm_width         => DRAM_DATA_SIZE,
439                 lpm_widthu        => 10,
440                 rdsync_delaypipe => 3,
441                 wrsync_delaypipe => 3)
442             port map (
443                 data       => s_buffer_readdata,
444                 wrreq      => s_dram_wrreq_1,
445                 rdreq      => s_dram_rdreq_1,
446                 wrclk      => dram_clk,
447                 rdclk      => component_clk,
448                 q          => s_dram_data_1,
449                 rdempty    => s_dram_empty_1,
450                 wrfull     => s_dram_full_1,
451                 aclr       => open,
452                 eccstatus  => open,
453                 rdfull     => open,
454                 rdusedw    => open,
455                 wrempty    => open,
456                 wrusedw    => open);
457
458     -- DRAM DATA FIFO
459     dram_data_2 : component dual_clock_fifo
460             generic map(
461                 lpm_numwords      => 1024,
462                 lpm_width         => DRAM_DATA_SIZE,
463                 lpm_widthu        => 10,
464                 rdsync_delaypipe => 3,
465                 wrsync_delaypipe => 3)
466             port map (
467                 data       => s_buffer_readdata,
468                 wrreq      => s_dram_wrreq_2,
```

```
469                 rdreq      => s_dram_rdreq_2,
470                 wrclk      => dram_clk,
471                 rdclk      => component_clk,
472                 q          => s_dram_data_2,
473                 rdempty    => s_dram_empty_2,
474                 wrfull     => s_dram_full_2,
475                 aclr       => open,
476                 eccstatus  => open,
477                 rdfull     => open,
478                 rdusedw    => open,
479                 wrempty    => open,
480                 wrusedw    => open);
481     -- DRAM DATA FIFO
482     dram_data_3 : component dual_clock_fifo
483             generic map(
484                 lpm_numwords      => 1024,
485                 lpm_width         => DRAM_DATA_SIZE,
486                 lpm_widthu        => 10,
487                 rdsync_delaypipe => 3,
488                 wrsync_delaypipe => 3)
489             port map (
490                 data       => s_buffer_readdata,
491                 wrreq      => s_dram_wrreq_3,
492                 rdreq      => s_dram_rdreq_3,
493                 wrclk      => dram_clk,
494                 rdclk      => component_clk,
495                 q          => s_dram_data_3,
496                 rdempty    => s_dram_empty_3,
497                 wrfull     => s_dram_full_3,
498                 aclr       => open,
499                 eccstatus  => open,
500                 rdfull     => open,
501                 rdusedw    => open,
502                 wrempty    => open,
503                 wrusedw    => open);
504
505     -- DRAM DATA FIFO
506     dram_data_4 : component dual_clock_fifo
507             generic map(
508                 lpm_numwords      => 1024,
509                 lpm_width         => DRAM_DATA_SIZE,
510                 lpm_widthu        => 10,
511                 rdsync_delaypipe => 3,
512                 wrsync_delaypipe => 3)
513             port map (
514                 data       => s_buffer_readdata,
515                 wrreq      => s_dram_wrreq_4,
516                 rdreq      => s_dram_rdreq_4,
```

```
517                 wrclk      => dram_clk,
518                 rdclk      => component_clk,
519                 q          => s_dram_data_4,
520                 rdempty    => s_dram_empty_4,
521                 wrfull     => s_dram_full_4,
522                 aclr       => open,
523                 eccstatus  => open,
524                 rdfull     => open,
525                 rdusedw    => open,
526                 wrempty    => open,
527                 wrusedw    => open);
528
529   -- FIFO Next State Process - Trigger on the Component Clock
530   fifo_read_start_proc : process (component_clk, rst_n)
531     begin
532       if (rst_n = '0') then
533         s_current_state_fifo_read <= S3_FIFO_read;
534       elsif (rising_edge(component_clk)) then
535             if(f_start_reading = '1' and
536                                    hps_enable_flag = '1') then
537                 s_current_state_fifo_read <=
538                                          s_next_state_fifo_read
                                             ;
539             end if;
540       end if;
541    end process;
542
543    -- FIFO Transition Process - Trigger on the Current State
544    fifo_read_state_trans_proc : process(
          s_current_state_fifo_read)
545    begin
546           case s_current_state_fifo_read is
547               when S0_FIFO_read =>
548                         s_next_state_fifo_read <= S1_FIFO_read
                             ;
549               when S1_FIFO_read =>
550                         s_next_state_fifo_read <= S2_FIFO_read
                             ;
551               when S2_FIFO_read =>
552                         s_next_state_fifo_read <= S3_FIFO_read
                             ;
553               when S3_FIFO_read =>
554                         s_next_state_fifo_read <= S0_FIFO_read
                             ;
555               when others       =>
556                         s_next_state_fifo_read <= S0_FIFO_read
                             ;
557           end case;
```

```
558     end process;
559
560     -- FIFO Output Logic - Trigger on the Current State, Start
561     -- flag, or any of the DRAM full flags
562     fifo_read_state_output_proc :process(
            s_current_state_fifo_read,
563                                 f_start_reading, s_dram_full_0,
564                                 s_dram_full_1, s_dram_full_2,
565                                 s_dram_full_3, s_dram_full_4)
566     begin
567             case s_current_state_fifo_read is
568                 when S0_FIFO_read =>
569                     if(s_dram_empty_1 = '0') then
570                         s_dram_rdreq_0 <= '0';
571                         s_dram_rdreq_1 <= '1';
572                         s_dram_rdreq_2 <= '0';
573                         s_dram_rdreq_3 <= '0';
574                         s_dram_rdreq_4 <= '0';
575                     else
576                         s_dram_rdreq_0 <= '0';
577                         s_dram_rdreq_1 <= '0';
578                         s_dram_rdreq_2 <= '0';
579                         s_dram_rdreq_3 <= '0';
580                         s_dram_rdreq_4 <= '0';
581                     end if;
582                 when S1_FIFO_read =>
583                     if(s_dram_empty_2 = '0') then
584                         s_dram_rdreq_0 <= '0';
585                         s_dram_rdreq_1 <= '0';
586                         s_dram_rdreq_2 <= '1';
587                         s_dram_rdreq_3 <= '0';
588                         s_dram_rdreq_4 <= '0';
589                     else
590                         s_dram_rdreq_0 <= '0';
591                         s_dram_rdreq_1 <= '0';
592                         s_dram_rdreq_2 <= '0';
593                         s_dram_rdreq_3 <= '0';
594                         s_dram_rdreq_4 <= '0';
595                     end if;
596                 when S2_FIFO_read =>
597                     if(s_dram_empty_3 = '0' and
598                                         s_dram_empty_4 = '0') then
599                         s_dram_rdreq_0 <= '0';
600                         s_dram_rdreq_1 <= '0';
601                         s_dram_rdreq_2 <= '0';
602                         s_dram_rdreq_3 <= '1';
603                         s_dram_rdreq_4 <= '1';
604                     else
```

```
605                         s_dram_rdreq_0 <= '0';
606                         s_dram_rdreq_1 <= '0';
607                         s_dram_rdreq_2 <= '0';
608                         s_dram_rdreq_3 <= '0';
609                         s_dram_rdreq_4 <= '0';
610                     end if;
611                 when S3_FIFO_read =>
612                     if(s_dram_empty_0 = '0') then
613                         s_dram_rdreq_0 <= '1';
614                         s_dram_rdreq_1 <= '0';
615                         s_dram_rdreq_2 <= '0';
616                         s_dram_rdreq_3 <= '0';
617                         s_dram_rdreq_4 <= '0';
618                     else
619                         s_dram_rdreq_0 <= '0';
620                         s_dram_rdreq_1 <= '0';
621                         s_dram_rdreq_2 <= '0';
622                         s_dram_rdreq_3 <= '0';
623                         s_dram_rdreq_4 <= '0';
624                     end if;
625                 when others =>
626                         s_dram_rdreq_0 <= '0';
627                         s_dram_rdreq_1 <= '0';
628                         s_dram_rdreq_2 <= '0';
629                         s_dram_rdreq_3 <= '0';
630                         s_dram_rdreq_4 <= '0';
631             end case;
632         end process;
633 
634     -- Set a Flag when the first piece of Data is ready
635     first_data_flag_process : process (rst_n, component_clk)
636         begin
637             if(rst_n = '0') then
638                 f_first_data <= '0';
639             elsif(rising_edge(component_clk)) then
640                 if(s_pixel_data_and_location_valid_flag = '1' and
641                                         hps_enable_flag = '1') then
642                     f_first_data <= '1';
643                 end if;
644             end if;
645         end process;
646 
647     -- Location Check Next State Process - Trigger on the
648     -- component clock
649     location_check_start_proc : process (component_clk, rst_n)
650      begin
651       if (rst_n = '0') then
652         s_current_state_location_check
```

```
653                                         <= S_Start_location_check;
654       elsif (rising_edge(component_clk)) then
655               if(f_first_data = '1' and
656                                         hps_enable_flag = '1') then
657                   s_current_state_location_check
658                                       <= s_next_state_location_check;
659               end if;
660       end if;
661   end process;
662
663   -- Location Check Transition Process
664   location_check_state_trans_proc : process(
665                                   s_current_state_location_check)
666   begin
667           case s_current_state_location_check is
668               when S_Start_location_check =>
669                   s_next_state_location_check <=
670                                               S1_location_check;
671               when S0_location_check =>
672                    s_next_state_location_check <=
673                                               S1_location_check;
674               when S1_location_check =>
675                   s_next_state_location_check <=
676                                               S2_location_check;
677               when S2_location_check =>
678                   s_next_state_location_check <=
679                                               S3_location_check;
680               when S3_location_check =>
681                   s_next_state_location_check <=
682                                               S0_location_check;
683               when others                 =>
684                   s_next_state_location_check <=
685                                               S0_location_check;
686           end case;
687   end process;
688
689   -- Location Check Output Logic - Checks the location of the
690   -- DRAM data and combines it with the pixel data
691   location_check_state_output_proc : process(
692                   s_current_state_location_check, f_first_data)
693   begin
694           case s_current_state_location_check is
695                   when S_Start_location_check =>
696                       if(hps_enable_flag = '1') then
697                           if(s_pixel_address_and_data_0(
698                              DATA_PACKAGE_SIZE - 1 downto
699                              DATA_PACKAGE_SIZE -
700                              SPECTRAL_BIN_ADDRESS_SIZE) =
```

```
701                 s_dram_data_0((DRAM_DATA_SIZE -
702                 DRAM_LOCATION_SIZE) +
703                 SPECTRAL_BIN_ADDRESS_SIZE - 1
704                 downto DRAM_DATA_SIZE -
705                 DRAM_LOCATION_SIZE) and
706                 s_pixel_address_and_data_0(
707                 DATA_PACKAGE_SIZE -
708                 SPECTRAL_BIN_ADDRESS_SIZE - 1
709                 downto DATA_PACKAGE_SIZE -
710                 SPECTRAL_BIN_ADDRESS_SIZE -
711                 PIXEL_ADDRESS_SIZE) =
712                 s_dram_data_0((DRAM_DATA_SIZE -
713                 (2 * DRAM_LOCATION_SIZE)) +
714                 PIXEL_ADDRESS_SIZE - 1 downto
715                 DRAM_DATA_SIZE - (2 *
716                 DRAM_LOCATION_SIZE))) then
717                     s_data_register_0    <=
718                         s_pixel_address_and_data_0
719                         & s_dram_data_0(
720                         LIGHT_CORRECT_SIZE +
721                         DARK_CORRECT_SIZE - 1
722                         downto 0);
723                     f_super_pixel_valid <= '0';
724                     f_new_frame         <= '0';
725                     f_clear_pixel       <= '0';
726                 else
727                     s_data_register_0       <=
728                                     (others => '0');
729                     f_super_pixel_valid     <= '0';
730                     f_new_frame             <= '0';
731                     s_pixel_location_check  <=
732                             unsigned(
733                             s_pixel_address_and_data_0
734                             (DATA_PACKAGE_SIZE -
735                             SPECTRAL_BIN_ADDRESS_SIZE
736                             - 1 downto
737                             DATA_PACKAGE_SIZE -
738                             SPECTRAL_BIN_ADDRESS_SIZE
739                             - PIXEL_ADDRESS_SIZE));
740                     f_clear_pixel           <= '1';
741                 end if;
742             else
743                 s_data_register_0           <=
744                                   (others => '0');
745                 f_super_pixel_valid         <= '0';
746                 f_new_frame                 <= '0';
747                 s_pixel_location_check      <=
748                   unsigned(s_pixel_address_and_data_0
```

```
749                         (DATA_PACKAGE_SIZE -
750                         SPECTRAL_BIN_ADDRESS_SIZE - 1
751                         downto DATA_PACKAGE_SIZE -
752                         SPECTRAL_BIN_ADDRESS_SIZE -
753                         PIXEL_ADDRESS_SIZE));
754                     f_clear_pixel                 <= '1';
755                 end if;
756             when S0_location_check =>
757                 if(f_clear_pixel = '0') then
758                     if(s_pixel_address_and_data_0(
759                         DATA_PACKAGE_SIZE - 1 downto
760                         DATA_PACKAGE_SIZE -
761                         SPECTRAL_BIN_ADDRESS_SIZE) =
762                         s_dram_data_0((DRAM_DATA_SIZE -
763                         DRAM_LOCATION_SIZE) +
764                         SPECTRAL_BIN_ADDRESS_SIZE - 1
765                         downto DRAM_DATA_SIZE -
766                         DRAM_LOCATION_SIZE) and
767                         s_pixel_address_and_data_0(
768                         DATA_PACKAGE_SIZE -
769                         SPECTRAL_BIN_ADDRESS_SIZE - 1
770                         downto DATA_PACKAGE_SIZE -
771                         SPECTRAL_BIN_ADDRESS_SIZE -
772                         PIXEL_ADDRESS_SIZE) =
773                         s_dram_data_0((DRAM_DATA_SIZE -
774                         (2 * DRAM_LOCATION_SIZE)) +
775                         PIXEL_ADDRESS_SIZE - 1 downto
776                         DRAM_DATA_SIZE - (2 *
777                         DRAM_LOCATION_SIZE))) then
778                             s_data_register_0    <=
779                                 s_pixel_address_and_data_0
780                                 & s_dram_data_0(
781                                 LIGHT_CORRECT_SIZE +
782                                 DARK_CORRECT_SIZE - 1
783                                 downto 0);
784                             f_super_pixel_valid <= '0';
785                             f_new_frame         <= '0';
786                             f_clear_pixel       <= '0';
787                     else
788                         s_data_register_0       <=
789                                             (others => '0');
790                         f_super_pixel_valid     <= '0';
791                         f_new_frame             <= '0';
792                         s_pixel_location_check  <=
793                               unsigned(
794                               s_pixel_address_and_data_0(
795                               DATA_PACKAGE_SIZE -
796                               SPECTRAL_BIN_ADDRESS_SIZE -
```

```
                                1 downto DATA_PACKAGE_SIZE
                                - SPECTRAL_BIN_ADDRESS_SIZE
                                - PIXEL_ADDRESS_SIZE));
                            f_clear_pixel           <= '1';
                        end if;
                    else
                        if(s_pixel_location_check =
                                c_pixel_condition) then
                            if(s_pixel_address_and_data_0(
                               DATA_PACKAGE_SIZE -
                               SPECTRAL_BIN_ADDRESS_SIZE - 1
                               downto DATA_PACKAGE_SIZE -
                               SPECTRAL_BIN_ADDRESS_SIZE -
                               PIXEL_ADDRESS_SIZE) =
                               c_pixel_zeros) then
                                if(s_pixel_address_and_data_0
                                  (DATA_PACKAGE_SIZE - 1
                                  downto DATA_PACKAGE_SIZE -
                                  SPECTRAL_BIN_ADDRESS_SIZE)
                                  = s_dram_data_0((
                                  DRAM_DATA_SIZE -
                                  DRAM_LOCATION_SIZE) +
                                  SPECTRAL_BIN_ADDRESS_SIZE -
                                  1 downto DRAM_DATA_SIZE -
                                  DRAM_LOCATION_SIZE) and
                                  s_pixel_address_and_data_0(
                                  DATA_PACKAGE_SIZE -
                                  SPECTRAL_BIN_ADDRESS_SIZE -
                                  1 downto DATA_PACKAGE_SIZE
                                  - SPECTRAL_BIN_ADDRESS_SIZE
                                  - PIXEL_ADDRESS_SIZE) =
                                  s_dram_data_0((
                                  DRAM_DATA_SIZE - (2 *
                                  DRAM_LOCATION_SIZE)) +
                                  PIXEL_ADDRESS_SIZE - 1
                                  downto DRAM_DATA_SIZE -
                                  (2 * DRAM_LOCATION_SIZE)))
                                  then
                                      s_data_register_0 <=
                                   s_pixel_address_and_data_0
                                       & s_dram_data_0(
                                       LIGHT_CORRECT_SIZE +
                                       DARK_CORRECT_SIZE - 1
                                       downto 0);
                                      f_super_pixel_valid
                                                  <= '0';
                                      f_new_frame    <= '0';
                                      f_clear_pixel <= '0';
```

```
845                 else
846                     s_data_register_0    <=
847                                 (others => '0');
848                     f_super_pixel_valid <=
849                                             '0';
850                     f_new_frame    <= '0';
851                     f_clear_pixel <= '1';
852                 end if;
853             else
854                 s_data_register_0    <=
855                             (others => '0');
856                 f_super_pixel_valid <= '0';
857                 f_new_frame          <= '0';
858                 f_clear_pixel        <= '1';
859             end if;
860         elsif(unsigned(
861                 s_pixel_address_and_data_0(
862                 DATA_PACKAGE_SIZE -
863                 SPECTRAL_BIN_ADDRESS_SIZE -
864                 1 downto DATA_PACKAGE_SIZE -
865                 SPECTRAL_BIN_ADDRESS_SIZE -
866                 PIXEL_ADDRESS_SIZE)) =
867                 s_pixel_location_check + 1)
868                 then
869             if(s_pixel_address_and_data_0(
870                 DATA_PACKAGE_SIZE - 1 downto
871                 DATA_PACKAGE_SIZE -
872                 SPECTRAL_BIN_ADDRESS_SIZE) =
873                 s_dram_data_0((DRAM_DATA_SIZE
874                 - DRAM_LOCATION_SIZE) +
875                 SPECTRAL_BIN_ADDRESS_SIZE - 1
876                 downto DRAM_DATA_SIZE -
877                 DRAM_LOCATION_SIZE) and
878                 s_pixel_address_and_data_0(
879                 DATA_PACKAGE_SIZE -
880                 SPECTRAL_BIN_ADDRESS_SIZE - 1
881                 downto DATA_PACKAGE_SIZE -
882                 SPECTRAL_BIN_ADDRESS_SIZE -
883                 PIXEL_ADDRESS_SIZE) =
884                 s_dram_data_0((DRAM_DATA_SIZE
885                 - (2 * DRAM_LOCATION_SIZE)) +
886                 PIXEL_ADDRESS_SIZE - 1 downto
887                 DRAM_DATA_SIZE - (2 *
888                 DRAM_LOCATION_SIZE))) then
889                 s_data_register_0    <=
890                     s_pixel_address_and_data_0
891                     & s_dram_data_0(
892                     LIGHT_CORRECT_SIZE +
```

```
893                                 DARK_CORRECT_SIZE - 1
894                                 downto 0);
895                             f_super_pixel_valid  <= '0';
896                             f_new_frame          <= '0';
897                             f_clear_pixel        <= '0';
898                         else
899                             s_data_register_0    <=
900                                         (others => '0');
901                             f_super_pixel_valid <= '0';
902                             f_new_frame          <= '0';
903                             f_clear_pixel        <= '1';
904                             s_pixel_location_check <=
905                              unsigned(
906                              s_pixel_address_and_data_0(
907                             DATA_PACKAGE_SIZE -
908                             SPECTRAL_BIN_ADDRESS_SIZE -
909                             1 downto DATA_PACKAGE_SIZE -
910                             SPECTRAL_BIN_ADDRESS_SIZE -
911                             PIXEL_ADDRESS_SIZE));
912                         end if;
913                     else
914                         s_data_register_0    <=
915                                         (others => '0');
916                         f_super_pixel_valid <= '0';
917                         f_new_frame          <= '0';
918                         f_clear_pixel        <= '1';
919                     end if;
920                 end if;
921             when S1_location_check =>
922                 if(f_clear_pixel = '0') then
923                     if(s_pixel_address_and_data_0(
924                        DATA_PACKAGE_SIZE - 1 downto
925                        DATA_PACKAGE_SIZE -
926                        SPECTRAL_BIN_ADDRESS_SIZE) =
927                        s_dram_data_1((DRAM_DATA_SIZE -
928                        DRAM_LOCATION_SIZE) +
929                        SPECTRAL_BIN_ADDRESS_SIZE - 1
930                        downto DRAM_DATA_SIZE -
931                        DRAM_LOCATION_SIZE) and
932                        s_pixel_address_and_data_0(
933                        DATA_PACKAGE_SIZE -
934                        SPECTRAL_BIN_ADDRESS_SIZE - 1
935                        downto DATA_PACKAGE_SIZE -
936                        SPECTRAL_BIN_ADDRESS_SIZE -
937                        PIXEL_ADDRESS_SIZE) =
938                        s_dram_data_1((DRAM_DATA_SIZE -
939                        (2 * DRAM_LOCATION_SIZE)) +
940                        PIXEL_ADDRESS_SIZE - 1 downto
```

```
941                     DRAM_DATA_SIZE - (2 *
942                     DRAM_LOCATION_SIZE))) then
943                         s_data_register_1   <=
944                             s_pixel_address_and_data_0
945                             & s_dram_data_1(
946                             LIGHT_CORRECT_SIZE +
947                             DARK_CORRECT_SIZE - 1
948                             downto 0) &
949                             s_data_register_0;
950                         f_super_pixel_valid <= '0';
951                         f_new_frame         <= '0';
952                         f_clear_pixel       <= '0';
953                     else
954                         s_data_register_1       <=
955                                         (others => '0');
956                         f_super_pixel_valid     <= '0';
957                         f_new_frame             <= '0';
958                         f_clear_pixel           <= '1';
959                         s_pixel_location_check <=
960                             unsigned(
961                             s_pixel_address_and_data_0(
962                             DATA_PACKAGE_SIZE -
963                             SPECTRAL_BIN_ADDRESS_SIZE -
964                             1 downto DATA_PACKAGE_SIZE -
965                             SPECTRAL_BIN_ADDRESS_SIZE -
966                             PIXEL_ADDRESS_SIZE));
967                     end if;
968                 else
969                     s_data_register_1 <= (others => '0');
970                     f_super_pixel_valid <= '0';
971                     f_new_frame         <= '0';
972                     f_clear_pixel       <= '1';
973                 end if;
974             when S2_location_check =>
975                 if(f_clear_pixel = '0') then
976                     if(s_pixel_address_and_data_0(
977                         DATA_PACKAGE_SIZE - 1 downto
978                         DATA_PACKAGE_SIZE -
979                         SPECTRAL_BIN_ADDRESS_SIZE) =
980                         s_dram_data_2((DRAM_DATA_SIZE -
981                         DRAM_LOCATION_SIZE) +
982                         SPECTRAL_BIN_ADDRESS_SIZE - 1
983                         downto DRAM_DATA_SIZE -
984                         DRAM_LOCATION_SIZE) and
985                         s_pixel_address_and_data_0(
986                         DATA_PACKAGE_SIZE -
987                         SPECTRAL_BIN_ADDRESS_SIZE - 1
988                         downto DATA_PACKAGE_SIZE -
```

```
                    SPECTRAL_BIN_ADDRESS_SIZE -
                    PIXEL_ADDRESS_SIZE) =
                    s_dram_data_2((DRAM_DATA_SIZE -
                    (2 * DRAM_LOCATION_SIZE)) +
                    PIXEL_ADDRESS_SIZE - 1
                    downto DRAM_DATA_SIZE - (2 *
                    DRAM_LOCATION_SIZE))) then
                        s_data_register_2    <=
                            s_pixel_address_and_data_0
                            & s_dram_data_2(
                            LIGHT_CORRECT_SIZE +
                            DARK_CORRECT_SIZE - 1
                            downto 0) &
                            s_data_register_1;
                        f_super_pixel_valid <= '0';
                        f_new_frame         <= '0';
                    else
                        s_data_register_2       <=
                                        (others => '0');
                        f_super_pixel_valid     <= '0';
                        f_new_frame             <= '0';
                        f_clear_pixel           <= '1';
                        s_pixel_location_check <=
                            unsigned(
                            s_pixel_address_and_data_0(
                            DATA_PACKAGE_SIZE -
                            SPECTRAL_BIN_ADDRESS_SIZE -
                            1 downto DATA_PACKAGE_SIZE -
                            SPECTRAL_BIN_ADDRESS_SIZE -
                            PIXEL_ADDRESS_SIZE));
                    end if;
                else
                    s_data_register_2 <= (others => '0');
                    f_super_pixel_valid  <= '0';
                    f_new_frame          <= '0';
                    f_clear_pixel        <= '1';
                end if;
            when S3_location_check =>
                if(f_clear_pixel = '0') then
                    if(s_pixel_address_and_data_0(
                        DATA_PACKAGE_SIZE - 1 downto
                        DATA_PACKAGE_SIZE -
                        SPECTRAL_BIN_ADDRESS_SIZE) =
                        s_dram_data_3((DRAM_DATA_SIZE -
                        DRAM_LOCATION_SIZE) +
                        SPECTRAL_BIN_ADDRESS_SIZE - 1
                        downto DRAM_DATA_SIZE -
                        DRAM_LOCATION_SIZE) and
```

```
1037                s_pixel_address_and_data_0(
1038            DATA_PACKAGE_SIZE -
1039            SPECTRAL_BIN_ADDRESS_SIZE - 1
1040            downto DATA_PACKAGE_SIZE -
1041            SPECTRAL_BIN_ADDRESS_SIZE -
1042            PIXEL_ADDRESS_SIZE) =
1043            s_dram_data_3((DRAM_DATA_SIZE -
1044            (2 * DRAM_LOCATION_SIZE)) +
1045            PIXEL_ADDRESS_SIZE - 1 downto
1046            DRAM_DATA_SIZE -
1047            (2 * DRAM_LOCATION_SIZE)) and
1048            s_pixel_address_and_data_1
1049            (DATA_PACKAGE_SIZE - 1 downto
1050            DATA_PACKAGE_SIZE -
1051            SPECTRAL_BIN_ADDRESS_SIZE) =
1052            s_dram_data_4((DRAM_DATA_SIZE -
1053            DRAM_LOCATION_SIZE) +
1054            SPECTRAL_BIN_ADDRESS_SIZE - 1
1055            downto DRAM_DATA_SIZE -
1056            DRAM_LOCATION_SIZE) and
1057            s_pixel_address_and_data_1(
1058            DATA_PACKAGE_SIZE -
1059            SPECTRAL_BIN_ADDRESS_SIZE - 1
1060            downto DATA_PACKAGE_SIZE -
1061            SPECTRAL_BIN_ADDRESS_SIZE -
1062            PIXEL_ADDRESS_SIZE) =
1063            s_dram_data_4((DRAM_DATA_SIZE -
1064            (2 * DRAM_LOCATION_SIZE)) +
1065            PIXEL_ADDRESS_SIZE - 1 downto
1066            DRAM_DATA_SIZE - (2 *
1067            DRAM_LOCATION_SIZE))) then
1068                if(s_data_register_2(
1069                    SUPER_PIXEL_SIZE -1 downto
1070                    SUPER_PIXEL_SIZE -
1071                    SPECTRAL_BIN_ADDRESS_SIZE)
1072                    = c_spectral_bin_zeros and
1073                    s_data_register_2(
1074                    SUPER_PIXEL_SIZE -
1075                    SPECTRAL_BIN_ADDRESS_SIZE
1076                    -1 downto SUPER_PIXEL_SIZE
1077                    -SPECTRAL_BIN_ADDRESS_SIZE
1078                    - PIXEL_ADDRESS_SIZE) =
1079                    c_pixel_zeros) then
1080                     f_new_frame      <= '1';
1081                else
1082                     f_new_frame      <= '0';
1083                end if;
1084                    s_data_register_3 <=
```

```
                                s_pixel_address_and_data_1
                                & s_dram_data_4(
                                LIGHT_CORRECT_SIZE +
                                DARK_CORRECT_SIZE - 1
                                downto 0) &
                                s_pixel_address_and_data_0
                                & s_dram_data_3(
                                LIGHT_CORRECT_SIZE +
                                DARK_CORRECT_SIZE - 1
                                downto 0) &
                                s_data_register_2;
                            f_clear_pixel       <= '0';
                            f_super_pixel_valid <= '1';
                        else
                            s_data_register_3       <=
                                            (others => '0');
                            f_super_pixel_valid     <= '0';
                            f_new_frame             <= '0';
                            f_clear_pixel           <= '1';
                            s_pixel_location_check  <=
                                    unsigned(
                                    s_pixel_address_and_data_0
                                    (DATA_PACKAGE_SIZE -
                                    SPECTRAL_BIN_ADDRESS_SIZE
                                    - 1 downto
                                    DATA_PACKAGE_SIZE -
                                    SPECTRAL_BIN_ADDRESS_SIZE
                                    - PIXEL_ADDRESS_SIZE));
                        end if;
                    else
                        s_data_register_3 <= (others => '0');
                        f_super_pixel_valid   <= '0';
                        f_new_frame           <= '0';
                        f_clear_pixel         <= '1';
                    end if;
                when others =>
                    s_data_register_0 <= (others => '0');
                    s_data_register_1 <= (others => '0');
                    s_data_register_2 <= (others => '0');
                    s_data_register_3 <= (others => '0');
                    f_super_pixel_valid<= '0';
                    f_new_frame       <= '0';
                    f_clear_pixel     <= '1';
                    s_pixel_location_check <= unsigned(
                                s_pixel_address_and_data_0(
                                DATA_PACKAGE_SIZE -
                                SPECTRAL_BIN_ADDRESS_SIZE -
                                1 downto DATA_PACKAGE_SIZE -
```

```
1133                                                SPECTRAL_BIN_ADDRESS_SIZE -
1134                                                PIXEL_ADDRESS_SIZE) );
1135                         end case;
1136             end process;
1137
1138         -- Assign the output signal
1139         buffer_read_out             <= s_buffer_read;
1140         clear_pixel_flag_out        <= f_clear_pixel;
1141         new_frame_flag_out          <= f_new_frame;
1142         super_pixel_valid_flag_out <= f_super_pixel_valid;
1143         data_packet_out             <= s_data_register_3;
1144
1145 end architecture;
```

Prototype Image Sensor MATLAB Code

```
%-------------------------------------------------------------------------
% Camera_Control_Avalon_TB.m
%-------------------------------------------------------------------------
clear;
clc;
%-------------------------------------------------------------------------

%-------------------------------------------------------------------------
% Controls from the Sensor
%-------------------------------------------------------------------------
cal_done_n          = fi(1, 0, 1, 0);
row_done_n          = fi(1, 0, 1, 0);

%-------------------------------------------------------------------------
% User Input - Exposure Timing
%-------------------------------------------------------------------------
exposure_time = 1/750; % seconds - enter by user
%                          Minimum Exposure Time = 1/500 seconds
%--Computations done on the HPS-----------------------------------------
sensor_frequency  = 66e6; % 66 MHz
exact_clk_cycles  = exposure_time * sensor_frequency;
Nrows             = round(1023 - ((exact_clk_cycles + 2) / 130));
rounded_clk_cycles  = round((1023-Nrows)*130 -2);
exposure_clk_cycles = fi(rounded_clk_cycles, 0, 18, 0);
exposure_row_addr   = fi(Nrows, 0, 10, 0);
command_register    = fi(0, 0, 24, 0);

hps_enable_flag     = fi(0, 0, 1, 0);
avs_s1_read         = fi(0, 0, 1, 0);
avs_s1_write        = fi(0, 0, 1, 0);
avs_s1_address      = fi(0, 0, 2, 0);
avs_s1_writedata    = fi(0, 0, 32, 0);

%-------------------------------------------------------------------------
% Code for the Simulation
%-------------------------------------------------------------------------
camera_control_avalon_hdl  = hdlcosim_camera_control_avalon;
hps_enable_clk_edge        = 10;
cal_done_clk_edge          = hps_enable_clk_edge + 200;
cal_start_clk_edge         = 100000;
Nclock_edges               = 1000000;

%--Simulate VHDL Code---------------------------------------------------
for clk=1:Nclock_edges
```

```
45 %     if(clk == 2)
46 %         avs_s1_write        = fi(1, 0, 1, 0);
47 %         avs_s1_address      = fi(0, 0, 2, 0);
48 %         avs_s1_writedata    = fi(exposure_clk_cycles, 0, 24, 0);
49 %     end
50 %
51 %     if(clk == 3)
52 %         avs_s1_write        = fi(1, 0, 1, 0);
53 %         avs_s1_address      = fi(1, 0, 2, 0);
54 %         avs_s1_writedata    = fi(exposure_row_addr, 0, 24, 0);
55 %     end
56
57     % Trigger the system
58     if(clk >= hps_enable_clk_edge)
59         hps_enable_flag     = fi(1, 0, 1, 0);
60     else
61         hps_enable_flag     = fi(0, 0, 1, 0);
62     end
63
64     % Simulate the cal_done_n signal from the sensor
65     if(clk == cal_done_clk_edge || clk==cal_done_clk_edge + 1 ...
66        || clk == cal_start_clk_edge + cal_done_clk_edge || clk...
67        == cal_start_clk_edge + cal_done_clk_edge + 1)
68         cal_done_n = fi(0, 0, 1, 0);
69     else
70         cal_done_n = fi(1, 0, 1, 0);
71     end
72
73     % Simulate the cal_done_n signal from the sensor after the
74     % new calibration
75     if(clk == cal_start_clk_edge)
76         avs_s1_write        = fi(1, 0, 1, 0);
77         avs_s1_address      = fi(2, 0, 2, 0);
78         avs_s1_writedata    = fi(1, 0, 32, 0);
79     elseif(clk > cal_start_clk_edge + 1)
80         avs_s1_write        = fi(0, 0, 1, 0);
81     end
82
83     [cal_strt_n, dark_off_en_n, ld_shft_n, data_read_en_n, ...
84      row_addr, row_strt_n, standby_n, tx_n, pg_n] =          ...
85      step(camera_control_avalon_hdl, hps_enable_flag,        ...
86      cal_done_n, row_done_n, avs_s1_read, avs_s1_write,      ...
87      avs_s1_address, avs_s1_writedata);
88 end
89
90 save results.mat
91 clear;
92 load results.mat
```

```
%-------------------------------------------------------------------------
% Camera_Control_TB.m
%-------------------------------------------------------------------------
clear;
clc;
%-------------------------------------------------------------------------

%-------------------------------------------------------------------------
% Controls from the Sensor
%-------------------------------------------------------------------------
cal_done_n          = fi(1, 0, 1, 0);
row_done_n          = fi(1, 0, 1, 0);

%-------------------------------------------------------------------------
% User Input - Exposure Timing
%-------------------------------------------------------------------------
exposure_time = 1/750; % seconds - enter by user
%                         - Minimum Exposure Time = 1/500 seconds
%--Computations done on the HPS-----------------------------------------
sensor_frequency = 66e6; % 66 MHz
exact_clk_cycles = exposure_time * sensor_frequency;
Nrows = round(1023 - ((exact_clk_cycles + 2) / 130));
rounded_clk_cycles = round((1023-Nrows)*130 -2);
exposure_clk_cycles = fi(rounded_clk_cycles, 0, 18, 0);
exposure_row_addr   = fi(Nrows, 0, 10, 0);
command_register  = fi(0, 0, 24, 0);

%-------------------------------------------------------------------------
% Code for the Simulation
%-------------------------------------------------------------------------
camera_control_hdl  = hdlcosim_camera_control;
cal_done_clk_edge = 200;
cal_start_clk_edge = 100000;
Nclock_edges = 1000000;
%---Simulate VHDL Code---------------------------------------------------
for clk=1:Nclock_edges
    % Simulate the cal_done_n signal from the sensor
    if(clk == cal_done_clk_edge || clk == cal_done_clk_edge+1 ...
       || clk == cal_start_clk_edge + cal_done_clk_edge || clk...
       == cal_start_clk_edge + cal_done_clk_edge + 1)
        cal_done_n = fi(0, 0, 1, 0);
    else
        cal_done_n = fi(1, 0, 1, 0);
    end

%     if(clk == cal_start_clk_edge)
%         command_register = fi(1, 0, 24, 0);
%     elseif(clk > cal_start_clk_edge + 130)
```

```
%           command_register = fi(0, 0, 24, 0);
%       end

    [cal_strt_n, dark_off_en_n, ld_shft_n, data_read_en_n,     ...
    row_addr, row_strt_n, standby_n, tx_n, pg_n,               ...
    sensor_calibration_irq] = step(camera_control_hdl,         ...
    cal_done_n, row_done_n, exposure_clk_cycles,               ...
    exposure_row_addr, command_register);
end

clear;
```

```
%-------------------------------------------------------------------------
% Simulated_Hyperspectral_Data.m
%-------------------------------------------------------------------------
function Simulated_Hyperspectral_Data(Total_Memory_Size,       ...
                Number_of_Data_Channels, Input_Data_Size,      ...
                Number_of_Fractional_Bits, Signed_Or_Unsigned)
%-------------------------------------------------------------------------
% Code to Generate Random Numbers from 0-65535 and
% convert them to binary
a = 0;
b = (2 ^ Input_Data_Size ) - 1;
Number_of_Pixels_Per_Tap = Total_Memory_Size/ ...
                                        Number_of_Data_Channels;

Data_Fix_Point_Tap_0 = fi(zeros(1,Number_of_Pixels_Per_Tap),   ...
                          Signed_Or_Unsigned, Input_Data_Size, ...
                          Number_of_Fractional_Bits);

Data_Fix_Point_Tap_1 = fi(zeros(1,Number_of_Pixels_Per_Tap),   ...
                          Signed_Or_Unsigned, Input_Data_Size, ...
                          Number_of_Fractional_Bits);

Data_Fix_Point_Tap_2 = fi(zeros(1,Number_of_Pixels_Per_Tap),   ...
                          Signed_Or_Unsigned, Input_Data_Size, ...
                          Number_of_Fractional_Bits);

Data_Fix_Point_Tap_3 = fi(zeros(1,Number_of_Pixels_Per_Tap),   ...
                          Signed_Or_Unsigned, Input_Data_Size, ...
                          Number_of_Fractional_Bits);

Data_Fix_Point_Tap_4 = fi(zeros(1,Number_of_Pixels_Per_Tap),   ...
                          Signed_Or_Unsigned, Input_Data_Size, ...
                          Number_of_Fractional_Bits);

Data_Fix_Point_Tap_5 = fi(zeros(1,Number_of_Pixels_Per_Tap),   ...
                          Signed_Or_Unsigned, Input_Data_Size, ...
```

```
37                          Number_of_Fractional_Bits);
38
39  Data_Fix_Point_Tap_6 = fi(zeros(1,Number_of_Pixels_Per_Tap),   ...
40                          Signed_Or_Unsigned, Input_Data_Size,   ...
41                          Number_of_Fractional_Bits);
42
43                          Signed_Or_Unsigned, Input_Data_Size,   ...
44                          Number_of_Fractional_Bits);
45
46  Data_Fix_Point_Tap_8 = fi(zeros(1,Number_of_Pixels_Per_Tap),   ...
47                          Signed_Or_Unsigned, Input_Data_Size,   ...
48                          Number_of_Fractional_Bits);
49
50  Data_Fix_Point_Tap_9 = fi(zeros(1,Number_of_Pixels_Per_Tap),   ...
51                          Signed_Or_Unsigned, Input_Data_Size,   ...
52                          Number_of_Fractional_Bits);
53
54  for i=1:Number_of_Pixels_Per_Tap
55      %Tap 0-------------------------------------------------------------
56      R_Tap_0 = a + (b-a).*rand(1,1);
57      R_Ints_Tap_0 = uint16(R_Tap_0);
58      Data_Fix_Point_Tap_0(i) = fi(R_Ints_Tap_0,                    ...
59                          Signed_Or_Unsigned, Input_Data_Size,...
60                          Number_of_Fractional_Bits);
61      %------------------------------------------------------------------
62      %Tap 1-------------------------------------------------------------
63      R_Tap_1 = a + (b-a).*rand(1,1);
64      R_Ints_Tap_1 = uint16(R_Tap_1);
65      Data_Fix_Point_Tap_1(i) = fi(R_Ints_Tap_1,                    ...
66                          Signed_Or_Unsigned, Input_Data_Size,...
67                          Number_of_Fractional_Bits);
68      %------------------------------------------------------------------
69      %Tap 2-------------------------------------------------------------
70      R_Tap_2 = a + (b-a).*rand(1,1);
71      R_Ints_Tap_2 = uint16(R_Tap_2);
72      Data_Fix_Point_Tap_2(i) = fi(R_Ints_Tap_2,                    ...
73                          Signed_Or_Unsigned, Input_Data_Size,...
74                          Number_of_Fractional_Bits);
75      %------------------------------------------------------------------
76      %Tap 3-------------------------------------------------------------
77      R_Tap_3 = a + (b-a).*rand(1,1);
78      R_Ints_Tap_3 = uint16(R_Tap_3);
79      Data_Fix_Point_Tap_3(i) = fi(R_Ints_Tap_3,                    ...
80                          Signed_Or_Unsigned, Input_Data_Size,...
81                          Number_of_Fractional_Bits);
82      %------------------------------------------------------------------
83      %Tap 4-------------------------------------------------------------
84      R_Tap_4 = a + (b-a).*rand(1,1);
```

```
85      R_Ints_Tap_4 = uint16(R_Tap_4);
86      Data_Fix_Point_Tap_4(i) = fi(R_Ints_Tap_4,                    ...
87                              Signed_Or_Unsigned, Input_Data_Size,...
88                              Number_of_Fractional_Bits);
89      %-----------------------------------------------------------------
90      % Tap 5-----------------------------------------------------------
91      R_Tap_5 = a + (b-a).*rand(1,1);
92      R_Ints_Tap_5 = uint16(R_Tap_5);
93      Data_Fix_Point_Tap_5(i) = fi(R_Ints_Tap_5,                    ...
94                              Signed_Or_Unsigned, Input_Data_Size,...
95                              Number_of_Fractional_Bits);
96      %-----------------------------------------------------------------
97      % Tap 6-----------------------------------------------------------
98      R_Tap_6 = a + (b-a).*rand(1,1);
99      R_Ints_Tap_6 = uint16(R_Tap_6);
100     Data_Fix_Point_Tap_6(i) = fi(R_Ints_Tap_6,                    ...
101                             Signed_Or_Unsigned, Input_Data_Size,...
102                             Number_of_Fractional_Bits);
103     %-----------------------------------------------------------------
104     % Tap 7-----------------------------------------------------------
105     R_Tap_7 = a + (b-a).*rand(1,1);
106     R_Ints_Tap_7 = uint16(R_Tap_7);
107     Data_Fix_Point_Tap_7(i) = fi(R_Ints_Tap_7,                    ...
108                             Signed_Or_Unsigned, Input_Data_Size,...
109                             Number_of_Fractional_Bits);
110     %-----------------------------------------------------------------
111     % Tap 8-----------------------------------------------------------
112     R_Tap_8 = a + (b-a).*rand(1,1);
113     R_Ints_Tap_8 = uint16(R_Tap_8);
114     Data_Fix_Point_Tap_8(i) = fi(R_Ints_Tap_8,                    ...
115                             Signed_Or_Unsigned, Input_Data_Size,...
116                             Number_of_Fractional_Bits);
117     %-----------------------------------------------------------------
118     % Tap 9-----------------------------------------------------------
119     R_Tap_9 = a + (b-a).*rand(1,1);
120     R_Ints_Tap_9 = uint16(R_Tap_9);
121     Data_Fix_Point_Tap_9(i) = fi(R_Ints_Tap_9,                    ...
122                             Signed_Or_Unsigned, Input_Data_Size,...
123                             Number_of_Fractional_Bits);
124     %-----------------------------------------------------------------
125 end
126 %-----------------------------------------------------------------
127 % Create the .mif files
128 %-----------------------------------------------------------------
129 fixpt2alteramif(Data_Fix_Point_Tap_0,                             ...
130             'Simulated_Data_Tap_1.mif', Number_of_Pixels_Per_Tap);
131 fixpt2alteramif(Data_Fix_Point_Tap_1,                             ...
132             'Simulated_Data_Tap_2.mif', Number_of_Pixels_Per_Tap);
```

```
133 fixpt2alteramif(Data_Fix_Point_Tap_2,                                ...
134             'Simulated_Data_Tap_3.mif', Number_of_Pixels_Per_Tap);
135 fixpt2alteramif(Data_Fix_Point_Tap_3,                                ...
136             'Simulated_Data_Tap_4.mif', Number_of_Pixels_Per_Tap);
137 fixpt2alteramif(Data_Fix_Point_Tap_4,                                ...
138             'Simulated_Data_Tap_5.mif', Number_of_Pixels_Per_Tap);
139 fixpt2alteramif(Data_Fix_Point_Tap_5,                                ...
140             'Simulated_Data_Tap_6.mif', Number_of_Pixels_Per_Tap);
141 fixpt2alteramif(Data_Fix_Point_Tap_6,                                ...
142             'Simulated_Data_Tap_7.mif', Number_of_Pixels_Per_Tap);
143 fixpt2alteramif(Data_Fix_Point_Tap_7,                                ...
144             'Simulated_Data_Tap_8.mif', Number_of_Pixels_Per_Tap);
145 fixpt2alteramif(Data_Fix_Point_Tap_8,                                ...
146             'Simulated_Data_Tap_9.mif', Number_of_Pixels_Per_Tap);
147 fixpt2alteramif(Data_Fix_Point_Tap_9,                                ...
148             'Simulated_Data_Tap_10.mif',Number_of_Pixels_Per_Tap);
149
150 save data.mat
151 end
```

```
1  %-------------------------------------------------------------------------
2  % Pixel_Averaging_TB.m
3  %-------------------------------------------------------------------------
4  clear;
5  clc;
6  F = fimath;
7  F.SumWordLength  = 13;
8  F.MaxSumWordLength = 13;
9  %pause on;
10 %-------------------------------------------------------------------------
11 %
12 %-------------------------------------------------------------------------
13 Number_of_Data_Channels       = 10;
14 Total_Memory_Size             = 2560;
15 Number_of_Pixels_Per_Channel = Total_Memory_Size /                    ...
16                                  Number_of_Data_Channels;
17 Number_of_Pixels              = 1024;
18 Pixel_Address_Size            = 11;
19 Number_of_Bins                = 160;
20 Bin_Address_Size              = 8;
21
22 Number_of_Average_Result_Bits = 13;
23 Target_Data_Size               = 16;
24 Number_of_Extra_Zeros          = Target_Data_Size -                   ...
25                                  Number_of_Average_Result_Bits - 1;
26 Input_Data_Size                = 10;
27 Data_Packet_Size               = Pixel_Address_Size +                 ...
28                                  Bin_Address_Size +                   ...
```

```
29                                      Number_of_Extra_Zeros +         ...
30                                      Number_of_Average_Result_Bits;
31
32  Number_of_Fractional_Bits      = 0;
33  Signed_Or_Unsigned             = 0;
34
35  % Simulated_Hyperspectral_Data(Total_Memory_Size,
36  % Number_of_Data_Channels, Input_Data_Size,
37  %Number_of_Fractional_Bits, Signed_Or_Unsigned)
38  Data_In_1_Double  = csvread('Simulated_Data_Tap_1.csv', 0, 0, ...
39                      [0 0 0 (Number_of_Pixels_Per_Channel - 1)]);
40  Data_In_2_Double  = csvread('Simulated_Data_Tap_2.csv', 0, 0, ...
41                      [0 0 0 (Number_of_Pixels_Per_Channel - 1)]);
42  Data_In_3_Double  = csvread('Simulated_Data_Tap_3.csv', 0, 0, ...
43                      [0 0 0 (Number_of_Pixels_Per_Channel - 1)]);
44  Data_In_4_Double  = csvread('Simulated_Data_Tap_4.csv', 0, 0, ...
45                      [0 0 0 (Number_of_Pixels_Per_Channel - 1)]);
46  Data_In_5_Double  = csvread('Simulated_Data_Tap_5.csv', 0, 0, ...
47                      [0 0 0 (Number_of_Pixels_Per_Channel - 1)]);
48  Data_In_6_Double  = csvread('Simulated_Data_Tap_6.csv', 0, 0, ...
49                      [0 0 0 (Number_of_Pixels_Per_Channel - 1)]);
50  Data_In_7_Double  = csvread('Simulated_Data_Tap_7.csv', 0, 0, ...
51                      [0 0 0 (Number_of_Pixels_Per_Channel - 1)]);
52  Data_In_8_Double  = csvread('Simulated_Data_Tap_8.csv', 0, 0, ...
53                      [0 0 0 (Number_of_Pixels_Per_Channel - 1)]);
54  Data_In_9_Double  = csvread('Simulated_Data_Tap_9.csv', 0, 0, ...
55                      [0 0 0 (Number_of_Pixels_Per_Channel - 1)]);
56  Data_In_10_Double = csvread('Simulated_Data_Tap_10.csv',0, 0, ...
57                      [0 0 0 (Number_of_Pixels_Per_Channel - 1)]);
58
59  %-------------------------------------------------------------------
60  % Compile the VHDL Design with new simulated data
61  %-------------------------------------------------------------------
62  %compile_hdl_design_top();
63  %-------------------------------------------------------------------
64  % Launch ModelSim
65  %-------------------------------------------------------------------
66  %launch_hdl_simulator_top();
67  %pause;
68  %-------------------------------------------------------------------
69  % HdlCosimulation System Object creation (this Matlab function
70  % was created by the cosimWizard).
71  %-------------------------------------------------------------------
72  % top_hdl = hdlcosim_top;
73  pixel_averaging_hdl = hdlcosim_pixel_averaging;
74
75  Nclock_edges                = 25;
76  Number_of_Data_Points       = 256;
```

```
77  latency                     = 4;
78
79  Error_Counter_1             = 1;
80  Error_Counter_2             = 1;
81  Error_Counter_3             = 1;
82  Error_Counter_4             = 1;
83  Error_Counter_5             = 1;
84
85  %--Allocate Memory-----------------------------------------------------
86  Data_In_0 = fi(zeros(1,Number_of_Pixels_Per_Channel),          ...
87                 Signed_Or_Unsigned, Input_Data_Size,            ...
88                 Number_of_Fractional_Bits, F);
89  Data_In_1 = fi(zeros(1,Number_of_Pixels_Per_Channel),          ...
90                 Signed_Or_Unsigned, Input_Data_Size,            ...
91                 Number_of_Fractional_Bits, F);
92  Data_In_2 = fi(zeros(1,Number_of_Pixels_Per_Channel),          ...
93                 Signed_Or_Unsigned, Input_Data_Size,            ...
94                 Number_of_Fractional_Bits, F);
95  Data_In_3 = fi(zeros(1,Number_of_Pixels_Per_Channel),          ...
96                 Signed_Or_Unsigned, Input_Data_Size,            ...
97                 Number_of_Fractional_Bits, F);
98  Data_In_4 = fi(zeros(1,Number_of_Pixels_Per_Channel),          ...
99                 Signed_Or_Unsigned, Input_Data_Size,            ...
100                Number_of_Fractional_Bits, F);
101 Data_In_5 = fi(zeros(1,Number_of_Pixels_Per_Channel),          ...
102                Signed_Or_Unsigned, Input_Data_Size,            ...
103                Number_of_Fractional_Bits, F);
104 Data_In_6 = fi(zeros(1,Number_of_Pixels_Per_Channel),          ...
105                Signed_Or_Unsigned, Input_Data_Size,            ...
106                Number_of_Fractional_Bits, F);
107 Data_In_7 = fi(zeros(1,Number_of_Pixels_Per_Channel),          ...
108                Signed_Or_Unsigned, Input_Data_Size,            ...
109                Number_of_Fractional_Bits, F);
110
111 input_history_0 =fi(zeros(Nclock_edges,Number_of_Data_Points),...
112                     Signed_Or_Unsigned, Input_Data_Size,       ...
113                     Number_of_Fractional_Bits, F);
114 input_history_1 =fi(zeros(Nclock_edges,Number_of_Data_Points),...
115                     Signed_Or_Unsigned, Input_Data_Size,       ...
116                     Number_of_Fractional_Bits, F);
117 input_history_2 =fi(zeros(Nclock_edges,Number_of_Data_Points),...
118                     Signed_Or_Unsigned, Input_Data_Size,       ...
119                     Number_of_Fractional_Bits, F);
120 input_history_3 =fi(zeros(Nclock_edges,Number_of_Data_Points),...
121                     Signed_Or_Unsigned, Input_Data_Size,       ...
122                     Number_of_Fractional_Bits, F);
123 input_history_4 =fi(zeros(Nclock_edges,Number_of_Data_Points),...
124                     Signed_Or_Unsigned, Input_Data_Size,       ...
```

```
125                     Number_of_Fractional_Bits, F);
126 input_history_5 =fi(zeros(Nclock_edges,Number_of_Data_Points),...
127                     Signed_Or_Unsigned, Input_Data_Size,       ...
128                     Number_of_Fractional_Bits, F);
129 input_history_6 =fi(zeros(Nclock_edges,Number_of_Data_Points),...
130                     Signed_Or_Unsigned, Input_Data_Size,       ...
131                     Number_of_Fractional_Bits, F);
132 input_history_7 =fi(zeros(Nclock_edges,Number_of_Data_Points),...
133                     Signed_Or_Unsigned, Input_Data_Size,       ...
134                     Number_of_Fractional_Bits, F);
135
136 output_history_0 = fi(zeros(Nclock_edges,                     ...
137                    Number_of_Data_Points), Signed_Or_Unsigned,...
138                    Target_Data_Size, Number_of_Fractional_Bits);
139
140 Addition_Result =fi(zeros(Nclock_edges,Number_of_Data_Points),...
141                     Signed_Or_Unsigned, Target_Data_Size,      ...
142                     Number_of_Fractional_Bits, F);
143
144 Difference_0 = zeros(Nclock_edges , Number_of_Data_Points);
145
146 Error_Location_1 = zeros(Nclock_edges , Number_of_Data_Points);
147 %-------------------------------------------------------------------
148
149 %---Generate Fixed Point Values-------------------------------------
150 for i = 1:1:Number_of_Pixels_Per_Channel
151     Data_In_0(i) = fi(Data_In_1_Double(i), Signed_Or_Unsigned,...
152                     Input_Data_Size, Number_of_Fractional_Bits);
153     Data_In_1(i) = fi(Data_In_2_Double(i), Signed_Or_Unsigned,...
154                     Input_Data_Size, Number_of_Fractional_Bits);
155     Data_In_2(i) = fi(Data_In_3_Double(i), Signed_Or_Unsigned,...
156                     Input_Data_Size, Number_of_Fractional_Bits);
157     Data_In_3(i) = fi(Data_In_4_Double(i), Signed_Or_Unsigned,...
158                     Input_Data_Size, Number_of_Fractional_Bits);
159     Data_In_4(i) = fi(Data_In_5_Double(i), Signed_Or_Unsigned,...
160                     Input_Data_Size, Number_of_Fractional_Bits);
161     Data_In_5(i) = fi(Data_In_6_Double(i), Signed_Or_Unsigned,...
162                     Input_Data_Size, Number_of_Fractional_Bits);
163     Data_In_6(i) = fi(Data_In_7_Double(i), Signed_Or_Unsigned,...
164                     Input_Data_Size, Number_of_Fractional_Bits);
165     Data_In_7(i) = fi(Data_In_8_Double(i), Signed_Or_Unsigned,...
166                     Input_Data_Size, Number_of_Fractional_Bits);
167 end
168 %-------------------------------------------------------------------
169
170 %---Simulate VHDL Code----------------------------------------------
171 for cycles=1:Nclock_edges
172     for clki = 1:1:Number_of_Data_Points
```

```
173         [data_packet_out_1] = step(pixel_averaging_hdl, ...
174                                    Data_In_0(clki), Data_In_1(clki), ...
175                                    Data_In_2(clki), Data_In_3(clki), ...
176                                    Data_In_4(clki), Data_In_5(clki), ...
177                                    Data_In_6(clki), Data_In_7(clki));
178         input_history_0(cycles,clki)  = Data_In_0(clki);
179         input_history_1(cycles,clki)  = Data_In_1(clki);
180         input_history_2(cycles,clki)  = Data_In_2(clki);
181         input_history_3(cycles,clki)  = Data_In_3(clki);
182         input_history_4(cycles,clki)  = Data_In_4(clki);
183         input_history_5(cycles,clki)  = Data_In_5(clki);
184         input_history_6(cycles,clki)  = Data_In_6(clki);
185         input_history_7(cycles,clki)  = Data_In_7(clki);
186
187         output_history_0(cycles,clki) = data_packet_out_1;
188     end
189 end
190 %-------------------------------------------------------------------------
191
192 %---Generate the MATLAB Calculated Results-------------------------------
193
194 for clki=1:1:Nclock_edges
195     start_counter = latency;
196     for counter = start_counter:1:Number_of_Data_Points+latency-1
197         Addition_Result_0_1 = input_history_0(clki,counter -3)...
198                             + input_history_1(clki,counter -3);
199         Addition_Result_2_3 = input_history_2(clki,counter -3)...
200                             + input_history_3(clki,counter -3);
201         Addition_Result_4_5 = input_history_4(clki,counter -3)...
202                             + input_history_5(clki,counter -3);
203         Addition_Result_6_7 = input_history_6(clki,counter -3)...
204                             + input_history_7(clki,counter -3);
205
206         Addition_Result_0_1_2_3 = Addition_Result_0_1 + ...
207                                                 Addition_Result_2_3;
208         Addition_Result_4_5_6_7 = Addition_Result_4_5 + ...
209                                                 Addition_Result_6_7;
210
211         if(counter == Number_of_Data_Points + 3)
212             Addition_Result(clki + 1, latency - 1) = ...
213                Addition_Result_0_1_2_3 + Addition_Result_4_5_6_7;
214         elseif(counter == Number_of_Data_Points + 2)
215             Addition_Result(clki + 1, latency - 2) = ...
216                Addition_Result_0_1_2_3 + Addition_Result_4_5_6_7;
217         elseif(counter == Number_of_Data_Points + 1)
218             Addition_Result(clki + 1, latency - 3) = ...
219                Addition_Result_0_1_2_3 + Addition_Result_4_5_6_7;
220         else
```

```
            Addition_Result(clki, counter) = ...
                Addition_Result_0_1_2_3 + Addition_Result_4_5_6_7;
        end
    end
end

Simulated = sprintf('Simulated');
MATLAB = sprintf('MATLAB');
Difference = sprintf('Differnce,');
fprintf('%10s, %10s, %10s\n',Simulated,MATLAB, Difference);
for clki=1:1:Nclock_edges
    if(clki == 1)
        counter_start = 4;
    else
        counter_start = 1;
    end
    for counter = counter_start:1:Number_of_Data_Points
        Difference_0(clki, counter) =  double(Addition_Result( ...
        clki, counter)) - double(output_history_0(clki, counter));

        if(Difference_0(clki , counter) ~= 0)
            Error_Location_1(clki,counter) = Error_Counter_1;
            fprintf('%10x, %10x, %10d, %10d\n', ...
                    double(output_history1(clki , counter)),...
                    double(Addition_Result(clki , counter)),...
                    Difference_0(clki , counter), ...
                    Error_Location_1(Nclock_edges,counter));
            fprintf('---------------------------------------\n');
            Error_Counter_1 = Error_Counter_1 + 1;
        end

    end
end
```

```
%-------------------------------------------------------------------------
% Camera_Interface_TB.m
%-------------------------------------------------------------------------
clear;
clc;
F = fimath;
F.SumWordLength  = 13;
F.MaxSumWordLength = 13;

%-------------------------------------------------------------------------
% Call the function Simulated_Hyperspectral_Data to create new
% random data for the test
%-------------------------------------------------------------------------
Number_of_Parallel_Channels  = 5;
```

```
15 Number_of_Data_Channels       = 10;
16 Number_of_Pixels_Per_Channel = 256;
17 Total_Memory_Size             = Number_of_Pixels_Per_Channel * ...
18                                     Number_of_Data_Channels;
19
20 Number_of_Pixels             = 1024;
21 Pixel_Address_Size           = 10;
22 Pixel_Counter                = Number_of_Pixels - 1;
23 Number_of_Bins               = 160;
24 Bin_Address_Size             = 8;
25 Bin_Counter                  = Number_of_Bins;
26
27 Number_of_Average_Result_Bits = 13;
28 Target_Data_Size              = 16;
29 Number_of_Extra_Zeros         = Target_Data_Size -              ...
30                                    Number_of_Average_Result_Bits;
31 Input_Data_Size               = 10;
32 Data_Packet_Size              = Pixel_Address_Size +            ...
33                                        Bin_Address_Size +     ...
34                                             Target_Data_Size;
35 Super_Pixel_Size              = Data_Packet_Size + 64 ;
36
37 Number_of_Fractional_Bits     = 0;
38 Signed_Or_Unsigned            = 0;
39
40 %-------------------------------------------------------------------
41 % Create New Test Data
42 %-------------------------------------------------------------------
43
44 % Simulated_Hyperspectral_Data(Total_Memory_Size,                 ...
45 %                     Number_of_Data_Channels, Input_Data_Size, ...
46 %                     Number_of_Fractional_Bits,                ...
47 %                     Signed_Or_Unsigned);
48 % This function will create new data for the On-Chip Memory
49 % and for the CSV Files read below
50 load data.mat
51
52 %-------------------------------------------------------------------
53 % HdlCosimulation System Object creation (this Matlab function
54 % was created by the cosimWizard).
55 %-------------------------------------------------------------------
56 camera_interface_dut_hdl = hdlcosim_camera_interface_dut;
57
58 Nclock_edges               = 50;
59 Number_of_Data_Points      = Number_of_Pixels_Per_Channel;
60 latency_data_point         = 11;
61 latency_clock_edges        =  2;
62
```

```
Error_Counter_0             = 1;
Error_Counter_1             = 1;
Error_Counter_2             = 1;
Error_Counter_3             = 1;
Error_Counter_4             = 1;

%---Allocate Memory-----------------------------------------------------------
input_history_0 =  fi(zeros(Nclock_edges ,                                  ...
                   Number_of_Data_Points), Signed_Or_Unsigned,  ...
                   Input_Data_Size, Number_of_Fractional_Bits, F);
input_history_1  = fi(zeros(Nclock_edges ,                                  ...
                   Number_of_Data_Points), Signed_Or_Unsigned,  ...
                   Input_Data_Size, Number_of_Fractional_Bits, F);
input_history_2  = fi(zeros(Nclock_edges ,                                  ...
                   Number_of_Data_Points), Signed_Or_Unsigned,  ...
                   Input_Data_Size, Number_of_Fractional_Bits, F);
input_history_3  = fi(zeros(Nclock_edges ,                                  ...
                   Number_of_Data_Points), Signed_Or_Unsigned,  ...
                   Input_Data_Size, Number_of_Fractional_Bits, F);
input_history_4  = fi(zeros(Nclock_edges ,                                  ...
                   Number_of_Data_Points), Signed_Or_Unsigned,  ...
                   Input_Data_Size, Number_of_Fractional_Bits, F);
input_history_5  = fi(zeros(Nclock_edges ,                                  ...
                   Number_of_Data_Points), Signed_Or_Unsigned,  ...
                   Input_Data_Size, Number_of_Fractional_Bits, F);
input_history_6  = fi(zeros(Nclock_edges ,                                  ...
                   Number_of_Data_Points), Signed_Or_Unsigned,  ...
                   Input_Data_Size, Number_of_Fractional_Bits, F);
input_history_7  = fi(zeros(Nclock_edges ,                                  ...
                   Number_of_Data_Points), Signed_Or_Unsigned,  ...
                   Input_Data_Size, Number_of_Fractional_Bits, F);
input_history_8  = fi(zeros(Nclock_edges ,                                  ...
                   Number_of_Data_Points), Signed_Or_Unsigned,  ...
                   Input_Data_Size, Number_of_Fractional_Bits, F);
input_history_9  = fi(zeros(Nclock_edges ,                                  ...
                   Number_of_Data_Points), Signed_Or_Unsigned,  ...
                   Input_Data_Size, Number_of_Fractional_Bits, F);

output_history_0 = fi(zeros(Nclock_edges ,                                  ...
                   Number_of_Data_Points), Signed_Or_Unsigned,  ...
                   Super_Pixel_Size, Number_of_Fractional_Bits, F);
output_history_1 = fi(zeros(Nclock_edges ,                                  ...
                   Number_of_Data_Points), Signed_Or_Unsigned,  ...
                   Super_Pixel_Size, Number_of_Fractional_Bits, F);
output_history_2 = fi(zeros(Nclock_edges ,                                  ...
                   Number_of_Data_Points), Signed_Or_Unsigned,  ...
                   Super_Pixel_Size, Number_of_Fractional_Bits, F);
output_history_3 = fi(zeros(Nclock_edges ,                                  ...
```

```
                    Number_of_Data_Points), Signed_Or_Unsigned,  ...
                    Super_Pixel_Size, Number_of_Fractional_Bits, F);
output_history_4 = fi(zeros(Nclock_edges ,                       ...
                    Number_of_Data_Points), Signed_Or_Unsigned,  ...
                    Super_Pixel_Size, Number_of_Fractional_Bits, F);

Simulated_Result_0 = fi(zeros(Nclock_edges ,                     ...
                    Number_of_Data_Points), Signed_Or_Unsigned,  ...
                    Super_Pixel_Size, Number_of_Fractional_Bits, F);
Simulated_Result_1 = fi(zeros(Nclock_edges ,                     ...
                    Number_of_Data_Points), Signed_Or_Unsigned,  ...
                    Super_Pixel_Size, Number_of_Fractional_Bits, F);

Difference_0 = zeros(Nclock_edges , Number_of_Data_Points);
Difference_1 = zeros(Nclock_edges , Number_of_Data_Points);
Difference_2 = zeros(Nclock_edges , Number_of_Data_Points);
Difference_3 = zeros(Nclock_edges , Number_of_Data_Points);
Difference_4 = zeros(Nclock_edges , Number_of_Data_Points);

Error_Location_0 = zeros(Nclock_edges , Number_of_Data_Points);
Error_Location_1 = zeros(Nclock_edges , Number_of_Data_Points);
Error_Location_2 = zeros(Nclock_edges , Number_of_Data_Points);
Error_Location_3 = zeros(Nclock_edges , Number_of_Data_Points);
Error_Location_4 = zeros(Nclock_edges , Number_of_Data_Points);
%-------------------------------------------------------------------

%---Simulate VHDL Code-----------------------------------------------
for cycles = 1:Nclock_edges
    for clki = 1:1:Number_of_Data_Points
       [pixel_data_and_location_valid_flag, packet_valid_flag,...
        clear_pixel_flag, new_frame_flag, data_packet_out_0,  ...
        data_packet_out_1, data_packet_out_2,                 ...
        data_packet_out_3, data_packet_out_4] = step(         ...
        camera_interface_dut_hdl,  Data_Fix_Point_Tap_0(clki),...
        Data_Fix_Point_Tap_1(clki),Data_Fix_Point_Tap_2(clki),...
        Data_Fix_Point_Tap_3(clki),Data_Fix_Point_Tap_4(clki),...
        Data_Fix_Point_Tap_5(clki),Data_Fix_Point_Tap_6(clki),...
        Data_Fix_Point_Tap_7(clki),Data_Fix_Point_Tap_8(clki),...
        Data_Fix_Point_Tap_9(clki));
       input_history_0(cycles,clki) = Data_Fix_Point_Tap_0(clki);
       input_history_1(cycles,clki) = Data_Fix_Point_Tap_1(clki);
       input_history_2(cycles,clki) = Data_Fix_Point_Tap_2(clki);
       input_history_3(cycles,clki) = Data_Fix_Point_Tap_3(clki);
       input_history_4(cycles,clki) = Data_Fix_Point_Tap_4(clki);
       input_history_5(cycles,clki) = Data_Fix_Point_Tap_5(clki);
       input_history_6(cycles,clki) = Data_Fix_Point_Tap_6(clki);
       input_history_7(cycles,clki) = Data_Fix_Point_Tap_7(clki);
       input_history_8(cycles,clki) = Data_Fix_Point_Tap_8(clki);
```

```
159         input_history_9(cycles,clki) = Data_Fix_Point_Tap_9(clki);
160
161         output_history_0(cycles,clki) = data_packet_out_0;
162         output_history_1(cycles,clki) = data_packet_out_1;
163         output_history_2(cycles,clki) = data_packet_out_2;
164         output_history_3(cycles,clki) = data_packet_out_3;
165         output_history_4(cycles,clki) = data_packet_out_4;
166     end
167 end
168 %-------------------------------------------------------------------
169 Light_Value = fi(0,0,32,0);
170 Dark_Value  = fi(4294967295,0,32,0);
171 latency_data_minus_2 = latency_data_point - 2;
172 latency_data_minus_3 = latency_data_point - 3;
173 counter_output = 0;
174 step_1 = mod(latency_data_point     ,                              ...
175                                 Number_of_Parallel_Channels - 1);
176 step_2 = mod(latency_data_point + 1,                               ...
177                                 Number_of_Parallel_Channels - 1);
178 step_3 = mod(latency_data_point + 2,                               ...
179                                 Number_of_Parallel_Channels - 1);
180 step_4 = mod(latency_data_point + 3,                               ...
181                                 Number_of_Parallel_Channels - 1);
182 %---Generate the MATLAB Calculated Results-------------------------
183 for clki=latency_clock_edges:1:Nclock_edges
184     for counter = latency_data_point:1:Number_of_Data_Points +...
185                                             latency_data_point - 1
186         if(Pixel_Counter == Number_of_Pixels - 1 &&                 ...
187                                     Number_of_Bins == Bin_Counter)
188             Pixel_Counter = 0;
189             Bin_Counter = 0;
190
191             Pixel_Counter_FI = fi(Pixel_Counter,                   ...
192                       Signed_Or_Unsigned, Pixel_Address_Size,...
193                       Number_of_Fractional_Bits);
194             Bin_Counter_FI_0 = fi(Bin_Counter,                     ...
195                       Signed_Or_Unsigned, Bin_Address_Size,    ...
196                       Number_of_Fractional_Bits);
197             Bin_Counter_FI_1 = fi(Bin_Counter + 1,                 ...
198                       Signed_Or_Unsigned, Bin_Address_Size,    ...
199                       Number_of_Fractional_Bits);
200
201             counter_output = counter_output + 1;
202             Bin_Counter = Bin_Counter + 1;
203         elseif(Number_of_Bins == Bin_Counter)
204             Pixel_Counter = Pixel_Counter + 1;
205             Bin_Counter = 0;
206
```

```
207             Pixel_Counter_FI = fi(Pixel_Counter,                ...
208                       Signed_Or_Unsigned, Pixel_Address_Size,...
209                       Number_of_Fractional_Bits);
210             Bin_Counter_FI_0 = fi(Bin_Counter,                  ...
211                       Signed_Or_Unsigned, Bin_Address_Size,   ...
212                       Number_of_Fractional_Bits);
213             Bin_Counter_FI_1 = fi(Bin_Counter + 1,              ...
214                       Signed_Or_Unsigned, Bin_Address_Size,   ...
215                       Number_of_Fractional_Bits);
216
217             counter_output = counter_output + 1;
218             Bin_Counter = Bin_Counter + 1;
219         else
220             Pixel_Counter_FI = fi(Pixel_Counter,                ...
221                       Signed_Or_Unsigned, Pixel_Address_Size,...
222                       Number_of_Fractional_Bits);
223             Bin_Counter_FI_0 = fi(Bin_Counter,                  ...
224                       Signed_Or_Unsigned, Bin_Address_Size,   ...
225                       Number_of_Fractional_Bits);
226             Bin_Counter_FI_1 = fi(Bin_Counter + 1,              ...
227                       Signed_Or_Unsigned, Bin_Address_Size,   ...
228                       Number_of_Fractional_Bits);
229             if(counter_output == 3)
230                 Bin_Counter = Bin_Counter + 2;
231                 counter_output = 0;
232             else
233                 Bin_Counter = Bin_Counter + 1;
234                 counter_output = counter_output + 1;
235             end
236         end
237
238         Zeros_FI = fi(0,0,Number_of_Extra_Zeros,0);
239
240         if(counter==Number_of_Data_Points+latency_data_point - 1)
241             Addition_Result_0_1   = input_history_4(clki + 1, ...
242                                 1) + input_history_5(clki + 1, 1);
243             Addition_Result_2_3   = input_history_6(clki + 1, ...
244                                 1) + input_history_7(clki + 1, 1);
245             Addition_Result_4_5   = input_history_8(clki + 1, ...
246                                 1) + input_history_9(clki + 1, 1);
247             Addition_Result_6_7   = input_history_0(clki + 1, ...
248                                 2) + input_history_1(clki + 1, 2);
249             Addition_Result_8_9   = input_history_2(clki + 1, ...
250                                 2) + input_history_3(clki + 1, 2);
251             Addition_Result_10_11 = input_history_4(clki + 1, ...
252                                 2) + input_history_5(clki + 1, 2);
253             Addition_Result_12_13 = input_history_6(clki + 1, ...
254                                 2) + input_history_7(clki + 1, 2);
```

```
255             Addition_Result_14_15 = input_history_8(clki + 1, ...
256                                 2) + input_history_9(clki + 1, 2);
257         elseif(counter == Number_of_Data_Points +                ...
258                                             latency_data_point - 2)
259             Addition_Result_0_1   = input_history_6(clki,        ...
260                             counter - latency_data_minus_2) + ...
261                             input_history_7(clki, counter -   ...
262                             latency_data_minus_2);
263             Addition_Result_2_3   = input_history_8(clki,        ...
264                             counter - latency_data_minus_2) + ...
265                             input_history_9(clki, counter -   ...
266                             latency_data_minus_2);
267             Addition_Result_4_5   = input_history_0(clki + 1, ...
268                                 1) + input_history_1(clki + 1, 1);
269             Addition_Result_6_7   = input_history_2(clki + 1, ...
270                                 1) + input_history_3(clki + 1, 1);
271             Addition_Result_8_9   = fi(0, Signed_Or_Unsigned, ...
272                    Input_Data_Size, Number_of_Fractional_Bits,...
273                    F) + fi(0, Signed_Or_Unsigned,             ...
274                    Input_Data_Size, Number_of_Fractional_Bits,F);
275             Addition_Result_10_11 = fi(0, Signed_Or_Unsigned, ...
276                    Input_Data_Size, Number_of_Fractional_Bits,...
277                    F) + fi(0, Signed_Or_Unsigned,             ...
278                    Input_Data_Size, Number_of_Fractional_Bits,F);
279             Addition_Result_12_13 = fi(0, Signed_Or_Unsigned, ...
280                    Input_Data_Size, Number_of_Fractional_Bits,...
281                    F) + fi(0, Signed_Or_Unsigned,             ...
282                    Input_Data_Size, Number_of_Fractional_Bits,F);
283             Addition_Result_14_15 = fi(0, Signed_Or_Unsigned, ...
284                    Input_Data_Size, Number_of_Fractional_Bits,...
285                    F) + fi(0, Signed_Or_Unsigned,             ...
286                    Input_Data_Size, Number_of_Fractional_Bits,F);
287         else
288             if(mod(counter, Number_of_Parallel_Channels - 1)==...
289                                                       step_1)
290                 Addition_Result_0_1 = input_history_0(clki,   ...
291                         counter - latency_data_minus_3) +     ...
292                         input_history_1(clki, counter -       ...
293                         latency_data_minus_3);
294                 Addition_Result_2_3 = input_history_2(clki,   ...
295                         counter - latency_data_minus_3) +     ...
296                         input_history_3(clki, counter -       ...
297                         latency_data_minus_3);
298                 Addition_Result_4_5 = input_history_4(clki,   ...
299                         counter - latency_data_minus_3) +     ...
300                         input_history_5(clki, counter -       ...
301                         latency_data_minus_3);
302                 Addition_Result_6_7 = input_history_6(clki,   ...
```

```
303                     counter - latency_data_minus_3) +     ...
304                     input_history_7(clki, counter -       ...
305                     latency_data_minus_3);
306                 Addition_Result_8_9   = fi(0,              ...
307                     Signed_Or_Unsigned, Input_Data_Size,  ...
308                     Number_of_Fractional_Bits, F) + fi(0, ...
309                     Signed_Or_Unsigned, Input_Data_Size,  ...
310                     Number_of_Fractional_Bits, F);
311                 Addition_Result_10_11 = fi(0,             ...
312                     Signed_Or_Unsigned, Input_Data_Size,  ...
313                     Number_of_Fractional_Bits, F) + fi(0, ...
314                     Signed_Or_Unsigned, Input_Data_Size,  ...
315                     Number_of_Fractional_Bits, F);
316                 Addition_Result_12_13 = fi(0,             ...
317                     Signed_Or_Unsigned, Input_Data_Size,  ...
318                     Number_of_Fractional_Bits, F) + fi(0, ...
319                     Signed_Or_Unsigned, Input_Data_Size,  ...
320                     Number_of_Fractional_Bits, F);
321                 Addition_Result_14_15 = fi(0,             ...
322                     Signed_Or_Unsigned, Input_Data_Size,  ...
323                     Number_of_Fractional_Bits, F) + fi(0, ...
324                     Signed_Or_Unsigned, Input_Data_Size,  ...
325                     Number_of_Fractional_Bits, F);
326             elseif(mod(counter, Number_of_Parallel_Channels - ...
327                                                 1) == step_2)
328                 Addition_Result_0_1 = input_history_8(clki,   ...
329                     counter - latency_data_minus_2) +     ...
330                     input_history_9(clki, counter -       ...
331                     latency_data_minus_2);
332                 Addition_Result_2_3 = input_history_0(clki,   ...
333                     counter - latency_data_minus_3) +     ...
334                     input_history_1(clki, counter -       ...
335                     latency_data_minus_3);
336                 Addition_Result_4_5 = input_history_2(clki,   ...
337                     counter - latency_data_minus_3) +     ...
338                     input_history_3(clki, counter -       ...
339                     latency_data_minus_3);
340                 Addition_Result_6_7 = input_history_4(clki,   ...
341                     counter - latency_data_minus_3) +     ...
342                     input_history_5(clki, counter -       ...
343                     latency_data_minus_3);
344                 Addition_Result_8_9   = fi(0,              ...
345                     Signed_Or_Unsigned, Input_Data_Size,  ...
346                     Number_of_Fractional_Bits, F) + fi(0, ...
347                     Signed_Or_Unsigned, Input_Data_Size,  ...
348                     Number_of_Fractional_Bits, F);
349                 Addition_Result_10_11 = fi(0,             ...
350                     Signed_Or_Unsigned, Input_Data_Size,  ...
```

```
351                     Number_of_Fractional_Bits, F) + fi(0, ...
352                     Signed_Or_Unsigned, Input_Data_Size, ...
353                     Number_of_Fractional_Bits, F);
354                 Addition_Result_12_13 = fi(0, ...
355                     Signed_Or_Unsigned, Input_Data_Size, ...
356                     Number_of_Fractional_Bits, F) + fi(0, ...
357                     Signed_Or_Unsigned, Input_Data_Size, ...
358                     Number_of_Fractional_Bits, F);
359                 Addition_Result_14_15 = fi(0, ...
360                     Signed_Or_Unsigned, Input_Data_Size, ...
361                     Number_of_Fractional_Bits, F) + fi(0, ...
362                     Signed_Or_Unsigned, Input_Data_Size, ...
363                     Number_of_Fractional_Bits, F);
364             elseif(mod(counter, Number_of_Parallel_Channels - ...
365                                                 1) == step_3)
366                 Addition_Result_0_1 = input_history_6(clki, ...
367                     counter - latency_data_minus_2) + ...
368                     input_history_7(clki, counter - ...
369                     latency_data_minus_2);
370                 Addition_Result_2_3 = input_history_8(clki, ...
371                     counter - latency_data_minus_2) + ...
372                     input_history_9(clki, counter - ...
373                     latency_data_minus_2);
374                 Addition_Result_4_5 = input_history_0(clki, ...
375                     counter - latency_data_minus_3) + ...
376                     input_history_1(clki, counter - ...
377                     latency_data_minus_3);
378                 Addition_Result_6_7 = input_history_2(clki, ...
379                     counter - latency_data_minus_3) + ...
380                     input_history_3(clki, counter - ...
381                     latency_data_minus_3);
382                 Addition_Result_8_9   = fi(0, ...
383                     Signed_Or_Unsigned, Input_Data_Size, ...
384                     Number_of_Fractional_Bits, F) + fi(0, ...
385                     Signed_Or_Unsigned, Input_Data_Size, ...
386                     Number_of_Fractional_Bits, F);
387                 Addition_Result_10_11 = fi(0, ...
388                     Signed_Or_Unsigned, Input_Data_Size, ...
389                     Number_of_Fractional_Bits, F) + fi(0, ...
390                     Signed_Or_Unsigned, Input_Data_Size, ...
391                     Number_of_Fractional_Bits, F);
392                 Addition_Result_12_13 = fi(0, ...
393                     Signed_Or_Unsigned, Input_Data_Size, ...
394                     Number_of_Fractional_Bits, F) + fi(0, ...
395                     Signed_Or_Unsigned, Input_Data_Size, ...
396                     Number_of_Fractional_Bits, F);
397                 Addition_Result_14_15 = fi(0, ...
398                     Signed_Or_Unsigned, Input_Data_Size, ...
```

```
399                     Number_of_Fractional_Bits, F) + fi(0, ...
400                     Signed_Or_Unsigned, Input_Data_Size, ...
401                     Number_of_Fractional_Bits, F);
402             elseif(mod(counter, Number_of_Parallel_Channels - ...
403                                                 1) == step_4)
404                 Addition_Result_0_1 = input_history_4(clki, ...
405                         counter - latency_data_minus_2) + ...
406                         input_history_5(clki, counter - ...
407                         latency_data_minus_2);
408                 Addition_Result_2_3 = input_history_6(clki, ...
409                         counter - latency_data_minus_2) + ...
410                         input_history_7(clki, counter - ...
411                         latency_data_minus_2);
412                 Addition_Result_4_5 = input_history_8(clki, ...
413                         counter - latency_data_minus_2) + ...
414                         input_history_9(clki, counter - ...
415                         latency_data_minus_2);
416                 Addition_Result_6_7 = input_history_0(clki, ...
417                         counter - latency_data_minus_3) + ...
418                         input_history_1(clki, counter - ...
419                         latency_data_minus_3);
420                 Addition_Result_8_9   = input_history_2(clki, ...
421                         counter - latency_data_minus_3) + ...
422                         input_history_3(clki, counter - ...
423                         latency_data_minus_3);
424                 Addition_Result_10_11 = input_history_4(clki, ...
425                         counter - latency_data_minus_3) + ...
426                         input_history_5(clki, counter - ...
427                         latency_data_minus_3);
428                 Addition_Result_12_13 = input_history_6(clki, ...
429                         counter - latency_data_minus_3) + ...
430                         input_history_7(clki, counter - ...
431                         latency_data_minus_3);
432                 Addition_Result_14_15 = input_history_8(clki, ...
433                         counter - latency_data_minus_3) + ...
434                         input_history_9(clki, counter - ...
435                         latency_data_minus_3);
436             end
437         end
438
439         Addition_Result_0_1_2_3 = Addition_Result_0_1 + ...
440                                             Addition_Result_2_3;
441         Addition_Result_4_5_6_7 = Addition_Result_4_5 + ...
442                                             Addition_Result_6_7;
443         Addition_Result_8_9_10_11   = Addition_Result_8_9   + ...
444                                         Addition_Result_10_11;
445         Addition_Result_12_13_14_15 = Addition_Result_12_13 + ...
446                                         Addition_Result_14_15;
```

```
447
448         Addition_Result_1 = Addition_Result_0_1_2_3 +          ...
449                                         Addition_Result_4_5_6_7;
450         Addition_Result_2 = Addition_Result_8_9_10_11 +        ...
451                                       Addition_Result_12_13_14_15;
452
453         if(counter == Number_of_Data_Points + (                ...
454                                         latency_data_point - 1))
455             Simulated_Result_0(clki + 1, latency_data_point    ...
456                     - 1) = bitconcat(Bin_Counter_FI_0,         ...
457                     Pixel_Counter_FI, Zeros_FI,                ...
458                     Addition_Result_1, Light_Value, Dark_Value);
459             Simulated_Result_1(clki + 1, latency_data_point    ...
460                     - 1) = bitconcat(Bin_Counter_FI_1,         ...
461                     Pixel_Counter_FI, Zeros_FI,                ...
462                     Addition_Result_2, Light_Value, Dark_Value);
463         elseif(counter == Number_of_Data_Points + (            ...
464                                         latency_data_point - 2))
465             Simulated_Result_0(clki + 1, latency_data_point    ...
466                     - 2) = bitconcat(Bin_Counter_FI_0,         ...
467                     Pixel_Counter_FI, Zeros_FI,                ...
468                     Addition_Result_1, Light_Value, Dark_Value);
469             Simulated_Result_1(clki + 1, latency_data_point    ...
470                     - 2) = bitconcat(Bin_Counter_FI_1,         ...
471                     Pixel_Counter_FI, Zeros_FI,                ...
472                     Addition_Result_2, Light_Value, Dark_Value);
473         elseif(counter == Number_of_Data_Points + (            ...
474                                         latency_data_point - 3))
475             Simulated_Result_0(clki + 1, latency_data_point    ...
476                     - 3) = bitconcat(Bin_Counter_FI_0,         ...
477                     Pixel_Counter_FI, Zeros_FI,                ...
478                     Addition_Result_1, Light_Value, Dark_Value);
479             Simulated_Result_1(clki + 1, latency_data_point    ...
480                     - 3) = bitconcat(Bin_Counter_FI_1,         ...
481                     Pixel_Counter_FI, Zeros_FI,                ...
482                     Addition_Result_2, Light_Value, Dark_Value);
483         elseif(counter == Number_of_Data_Points + (            ...
484                                         latency_data_point - 4))
485             Simulated_Result_0(clki + 1, latency_data_point    ...
486                     - 4) = bitconcat(Bin_Counter_FI_0,         ...
487                     Pixel_Counter_FI, Zeros_FI,                ...
488                     Addition_Result_1, Light_Value, Dark_Value);
489             Simulated_Result_1(clki + 1, latency_data_point    ...
490                     - 4) = bitconcat(Bin_Counter_FI_1,         ...
491                     Pixel_Counter_FI, Zeros_FI,                ...
492                     Addition_Result_2, Light_Value, Dark_Value);
493         elseif(counter == Number_of_Data_Points + (            ...
494                                         latency_data_point - 5))
```

```
495             Simulated_Result_0(clki + 1, latency_data_point    ...
496                     - 5) = bitconcat(Bin_Counter_FI_0,         ...
497                     Pixel_Counter_FI, Zeros_FI,                ...
498                     Addition_Result_1, Light_Value, Dark_Value);
499             Simulated_Result_1(clki + 1, latency_data_point    ...
500                     - 5) = bitconcat(Bin_Counter_FI_1,         ...
501                     Pixel_Counter_FI, Zeros_FI,                ...
502                     Addition_Result_2, Light_Value, Dark_Value);
503         elseif(counter == Number_of_Data_Points + (            ...
504                                         latency_data_point - 6))
505             Simulated_Result_0(clki + 1, latency_data_point    ...
506                     - 6) = bitconcat(Bin_Counter_FI_0,         ...
507                     Pixel_Counter_FI, Zeros_FI,                ...
508                     Addition_Result_1, Light_Value, Dark_Value);
509             Simulated_Result_1(clki + 1, latency_data_point    ...
510                     - 6) = bitconcat(Bin_Counter_FI_1,         ...
511                     Pixel_Counter_FI, Zeros_FI,                ...
512                     Addition_Result_2, Light_Value, Dark_Value);
513         elseif(counter == Number_of_Data_Points + (            ...
514                                         latency_data_point - 7))
515             Simulated_Result_0(clki + 1, latency_data_point    ...
516                     - 7) = bitconcat(Bin_Counter_FI_0,         ...
517                     Pixel_Counter_FI, Zeros_FI,                ...
518                     Addition_Result_1, Light_Value, Dark_Value);
519             Simulated_Result_1(clki + 1, latency_data_point    ...
520                     - 7) = bitconcat(Bin_Counter_FI_1,         ...
521                     Pixel_Counter_FI, Zeros_FI,                ...
522                     Addition_Result_2, Light_Value, Dark_Value);
523         elseif(counter == Number_of_Data_Points + (            ...
524                                         latency_data_point - 8))
525             Simulated_Result_0(clki + 1, latency_data_point    ...
526                     - 8) = bitconcat(Bin_Counter_FI_0,         ...
527                     Pixel_Counter_FI, Zeros_FI,                ...
528                     Addition_Result_1, Light_Value, Dark_Value);
529             Simulated_Result_1(clki + 1, latency_data_point    ...
530                    - 8) = bitconcat(Bin_Counter_FI_1,          ...
531                    Pixel_Counter_FI, Zeros_FI,                 ...
532                    Addition_Result_2, Light_Value, Dark_Value);
533         elseif(counter == Number_of_Data_Points + (            ...
534                                         latency_data_point - 9))
535             Simulated_Result_0(clki + 1, latency_data_point    ...
536                     - 9) = bitconcat(Bin_Counter_FI_0,         ...
537                     Pixel_Counter_FI, Zeros_FI,                ...
538                     Addition_Result_1, Light_Value, Dark_Value);
539             Simulated_Result_1(clki + 1, latency_data_point    ...
540                     - 9) = bitconcat(Bin_Counter_FI_1,         ...
541                     Pixel_Counter_FI, Zeros_FI,                ...
542                     Addition_Result_2, Light_Value, Dark_Value);
```

```
543         elseif(counter == Number_of_Data_Points + (                ...
544                                         latency_data_point - 10))
545             Simulated_Result_0(clki + 1, latency_data_point    ...
546                     - 10) = bitconcat(Bin_Counter_FI_0,        ...
547                     Pixel_Counter_FI, Zeros_FI,                ...
548                     Addition_Result_1, Light_Value, Dark_Value);
549             Simulated_Result_1(clki + 1, latency_data_point    ...
550                     - 10) = bitconcat(Bin_Counter_FI_1,        ...
551                     Pixel_Counter_FI, Zeros_FI,                ...
552                     Addition_Result_2, Light_Value, Dark_Value);
553         else
554             Simulated_Result_0(clki, counter) = bitconcat(     ...
555                         Bin_Counter_FI_0, Pixel_Counter_FI,...
556                         Zeros_FI, Addition_Result_1,           ...
557                         Light_Value, Dark_Value);
558             Simulated_Result_1(clki, counter) = bitconcat(     ...
559                         Bin_Counter_FI_1, Pixel_Counter_FI,...
560                         Zeros_FI, Addition_Result_2,           ...
561                         Light_Value, Dark_Value);
562         end
563      end
564 end
565
566
567 Simulated = sprintf('Simulated');
568 MATLAB = sprintf('MATLAB');
569 Difference = sprintf('Differnce');
570 fprintf('%10s, %10s, %10s\n--------------------------------\n'...
571                                     ,Simulated,MATLAB, Difference);
572
573 for clki=latency_clock_edges:1:Nclock_edges
574     if(clki == latency_clock_edges)
575         counter_start = latency_data_point;
576     else
577         counter_start = 3;
578     end
579     for counter = counter_start:4:Number_of_Data_Points
580         if (counter == 255)
581             Difference_0(clki, counter) =  double(             ...
582                    Simulated_Result_0(clki, counter))     -    ...
583                    double(output_history_0(clki, counter));
584             Difference_1(clki, counter) =  double(             ...
585                    Simulated_Result_0(clki, counter + 1)) -    ...
586                    double(output_history_1(clki, counter));
587             Difference_2(clki, counter) =  double(             ...
588                      Simulated_Result_0(clki + 1, 1)) -    ...
589                      double(output_history_2(clki, counter));
590             Difference_3(clki, counter) =  double(             ...
```

```
591                     Simulated_Result_0(clki + 1, 2)) -    ...
592                     double(output_history_3(clki, counter));
593         Difference_4(clki, counter) = double(             ...
594                     Simulated_Result_1(clki + 1, 2)) -    ...
595                     double(output_history_4(clki, counter));
596     else
597         Difference_0(clki, counter) = double(             ...
598                 Simulated_Result_0(clki, counter)) -      ...
599                 double(output_history_0(clki, counter));
600         Difference_1(clki, counter) = double(             ...
601                 Simulated_Result_0(clki, counter + 1)) - ...
602                 double(output_history_1(clki, counter));
603         Difference_2(clki, counter) = double(             ...
604                 Simulated_Result_0(clki, counter + 2)) - ...
605                 double(output_history_2(clki, counter));
606         Difference_3(clki, counter) = double(             ...
607                 Simulated_Result_0(clki, counter + 3)) - ...
608                 double(output_history_3(clki, counter));
609         Difference_4(clki, counter) = double(             ...
610                 Simulated_Result_1(clki, counter + 3)) - ...
611                 double(output_history_4(clki, counter));
612     end
613
614     if(Difference_0(clki , counter) ~= 0)
615         Error_Location_0(clki,counter) = Error_Counter_0;
616         fprintf('Diff_0\n%10x, %10x, %10d, %10d\n',       ...
617         double(output_history_0(clki , counter)),         ...
618         double(Simulated_Result_0(clki , counter)),       ...
619         Difference_0(clki , counter),                     ...
620         Error_Location_0(clki,counter));
621         fprintf('---------------------------------------\n');
622         Error_Counter_0 = Error_Counter_0 + 1;
623     end
624
625     if(Difference_1(clki , counter) ~= 0)
626         Error_Location_1(clki,counter) = Error_Counter_1;
627         fprintf('Diff_1\n%10x, %10x, %10d, %10d\n',       ...
628         double(output_history_1(clki , counter)),         ...
629         double(Simulated_Result_0(clki , counter + 1)),   ...
630         Difference_1(clki , counter),                     ...
631         Error_Location_1(clki,counter));
632         fprintf('---------------------------------------\n');
633         Error_Counter_1 = Error_Counter_1 + 1;
634     end
635
636     if(Difference_2(clki , counter) ~= 0)
637         Error_Location_2(clki,counter) = Error_Counter_2;
638         if(counter == 255)
```

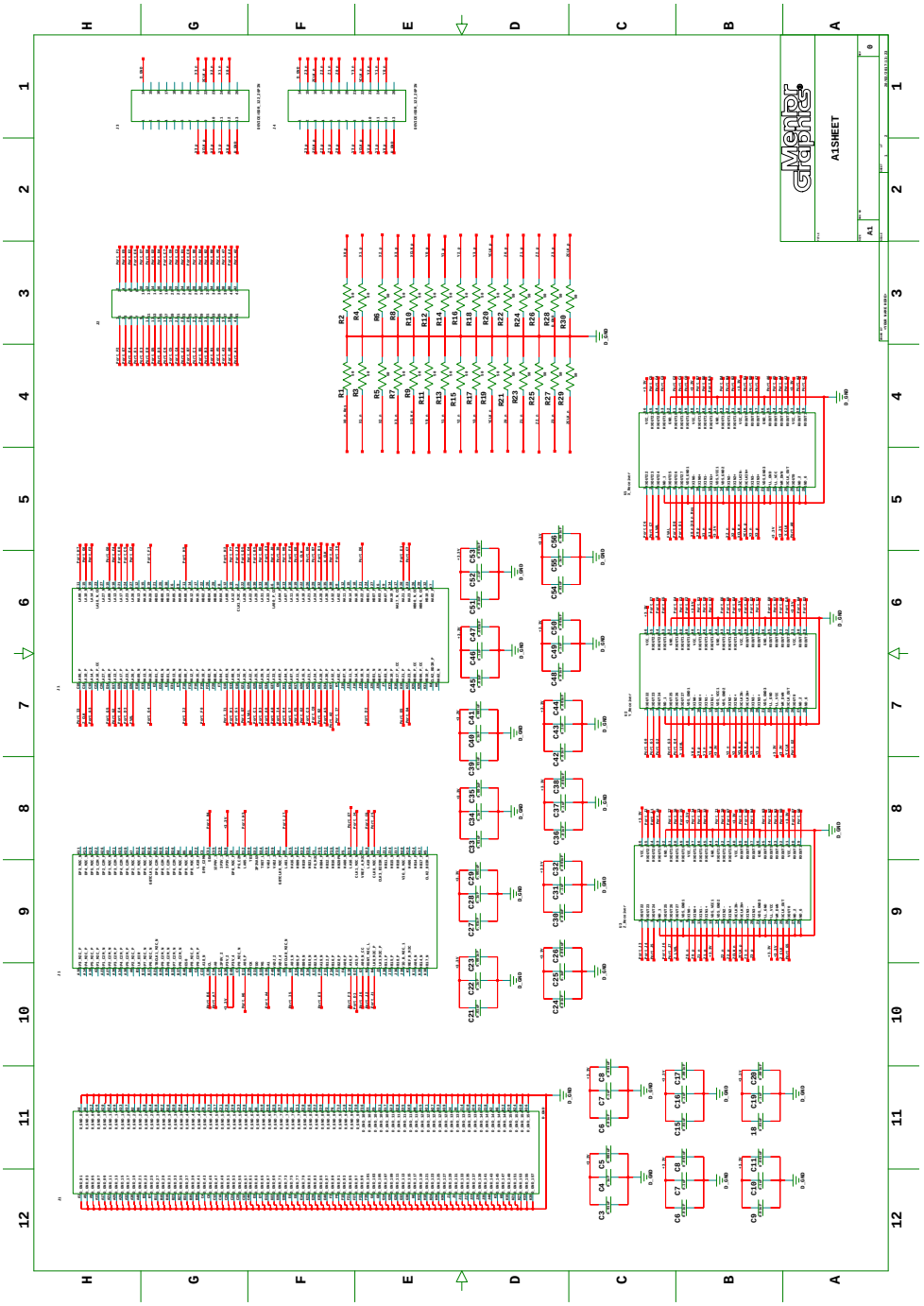
```
639                 fprintf('Diff_2\n%10x, %10x, %10d, %10d\n',    ...
640                 double(output_history_2(clki , counter)),      ...
641                 double(Simulated_Result_0(clki +1, 1)),        ...
642                 Difference_2(clki , counter),                  ...
643                 Error_Location_2(clki,counter));
644             else
645                 fprintf('Diff_2\n%10x, %10x, %10d, %10d\n',    ...
646                 double(output_history_2(clki , counter)),      ...
647                 double(Simulated_Result_0(clki, counter + 2)),...
648                 Difference_2(clki , counter),                  ...
649                 Error_Location_2(clki,counter));
650             end
651             fprintf('---------------------------------------\n');
652             Error_Counter_2 = Error_Counter_2 + 1;
653         end
654
655         if(Difference_3(clki , counter) ~= 0)
656             Error_Location_3(clki,counter) = Error_Counter_3;
657             if(counter == 255)
658                 fprintf('Diff_3\n%10x, %10x, %10d, %10d\n',    ...
659                 double(output_history_3(clki , counter)),      ...
660                 double(Simulated_Result_0(clki+1, 2)),         ...
661                 Difference_3(clki , counter),                  ...
662                 Error_Location_3(clki,counter));
663             else
664                 fprintf('Diff_3\n%10x, %10x, %10d, %10d\n',    ...
665                 double(output_history_3(clki , counter)),      ...
666                 double(Simulated_Result_0(clki, counter)),     ...
667                 Difference_3(clki , counter),                  ...
668                 Error_Location_3(clki,counter));
669             end
670             fprintf('---------------------------------------\n');
671             Error_Counter_3 = Error_Counter_3 + 1;
672         end
673
674         if(Difference_4(clki , counter) ~= 0)
675             Error_Location_4(clki,counter) = Error_Counter_4;
676             if(counter == 255)
677                 fprintf('Diff_4\n%10x, %10x, %10d, %10d\n',    ...
678                 double(output_history_4(clki , counter)),      ...
679                 double(Simulated_Result_1(clki+1 , 2)),        ...
680                 Difference_4(clki , counter),                  ...
681                 Error_Location_4(clki,counter));
682             else
683                 fprintf('Diff_4\n%10x, %10x, %10d, %10d\n',    ...
684                     double(output_history_4(clki , counter)),...
685                     double(Simulated_Result_1(clki , counter ...
686                     + 3)), Difference_4(clki , counter),       ...
```

```
687                         Error_Location_4(clki,counter));
688                 end
689                 fprintf('---------------------------------------\n');
690                 Error_Counter_4 = Error_Counter_4 + 1;
691             end
692         end
693 end
694
695 save results.mat
696 clear
697 load results.mat
698 %-------------------------------------------------------------------------
```

APPENDIX B

LINE SCAN CAMERA AND OBJECT TRACKING

Waterfall Sandwich Prototype Board Schematic



Line Scan Camera and Object Tracking VHDL Code

```
1  ----------------------------------------------------------------------------
2  --
3  --! @file         user_controls.vhd
4  --! @brief        Contains user defined constants for the line scan
5  --!               system.
6  --! @details
7  --
8  --! @author       Connor Dack
9  --! @date         November 2016
10 --! @copyright    Copyright (C) 2015 Ross K. Snider and Connor Dack
11 --
12 --   This program is free software: you can redistribute it
13 --   and/or modify it under the terms of the GNU General
14 --   Public License as published by the Free Software Foundation,
15 --   either version 3 of the License, or (at your option) any
16 --   later version.
17 --
18 --   This program is distributed in the hope that it will be
19 --   useful, but WITHOUT ANY WARRANTY; without even the implied
20 --   warranty of MERCHANTABILITY or FITNESS for A PARTICULAR
21 --   PURPOSE. See the GNU General Public License for more details.
22 --
23 --   You should have received a copy of the GNU General Public
24 --   License along with this program.  If not, see
25 --   <http://www.gnu.org/licenses/>.
26 --
27 --   Connor Dack
28 --   Electrical and Computer Engineering
29 --   Montana State University
30 --   610 Cobleigh Hall
31 --   Bozeman, MT 59717
32 --   connor.dack@msu.montana.edu
33 --
34 ----------------------------------------------------------------------------
35
36 library IEEE;
37 use IEEE.STD_LOGIC_1164.all;
38 use IEEE.NUMERIC_STD.all;
39 use IEEE.math_real.all;
40
41 use work.Sensor_Package.all;
42
43 package USER_CONTROLS is
44     constant AOI                            : integer := 1000;
```

```
45      -- Area of Interest -- Number of Pixels in a Line
46      constant NUMBER_OF_TAPS             : integer := 10;
47      constant AOI_LESS_10                : integer := AOI -
48                                                       NUMBER_OF_TAPS;
49      -- The number of pixels on a line - 10. Must be a multiple of
50      -- 10 and maxes at 8150.
51      constant TAPS_IN_AOI                : integer := AOI /
52                                                       NUMBER_OF_TAPS;
53      -- Number of Taps in the Area of Interest
54      constant AOI_ADDRESS_SIZE           : integer := 10;
55      constant MAX_OBJECT_NUMBER          : integer
56      := MAX_OBJECT_NUMBER;      -- 64
57      constant OBJECT_ADDRESS_SIZE        : integer
58      := OBJECT_ADDRESS_SIZE;  -- Number of bits needed to
59                               -- represent the max number of objects
60      constant PIXEL_ADDRESS_SIZE         : integer
61      := PIXEL_ADDRESS_SIZE;    -- Hyperspectral
62      constant NUMBER_OF_PIXELS           : integer
63      := NUMBER_OF_PIXELS;      -- Hyperspectral
64      constant TRANSCEIVER_DATA_SIZE      : integer := 64;
65      constant CLOCKS_PER_LS              : integer := 1250; -- The
66    -- number of clock cycles it takes for a line scan to complete
67      constant LINE_NUMBER_ADDRESS_SIZE   : integer := 16;
68
69      constant LINE_TO_HYPER_PIXEL_RATIO : unsigned(31 downto 0)
70      := "0010" & x"0000000"; --line/hyperspectral - 50u/pixel /
71      -- 400u/pixel = 0.125
72      constant LINE_TO_HYPER_LINE_RATIO   : unsigned(31 downto 0)
73      := "0010" & x"0000000"; --line/hyperspectral - 50u/line  /
74      -- 400u/line  = 0.125
75      constant HYPER_TO_LINE_LINE_RATIO   : integer := 7;
76      -- hyperspectral/line - 400u/line / 50u/line
77
78      type ss_inSeg_array is array(1 to TAPS_IN_AOI) of STD_LOGIC;
79    -- 816 because 816 taps in a line. Change to be taps in a line
80      type seg_info_array is array(1 to TAPS_IN_AOI) of
81      STD_LOGIC_VECTOR(AOI_ADDRESS_SIZE - 1 downto 0);
82  end package;
83
84  package body USER_CONTROLS is
85
86  end package body;
```

```
1   ------------------------------------------------------------------------
2   --
3   --! @file        top.vhd
4   --! @brief
5   --
```

```
6  --! @author      Connor Dack
7  --! @date        May 2015
8  --! @copyright   Copyright (C) 2015 Ross K. Snider and Connor Dack
9  --
10 --   This program is free software: you can redistribute it
11 --   and/or modify it under the terms of the GNU General
12 --   Public License as published by the Free Software Foundation,
13 --   either version 3 of the License, or (at your option) any
14 --   later version.
15 --
16 --   This program is distributed in the hope that it will be
17 --   useful, but WITHOUT ANY WARRANTY; without even the implied
18 --   warranty of MERCHANTABILITY or FITNESS for A PARTICULAR
19 --   PURPOSE. See the GNU General Public License for more details.
20 --
21 --   You should have received a copy of the GNU General Public
22 --   License along with this program.  If not, see
23 --   <http://www.gnu.org/licenses/>.
24 --
25 --   Connor Dack
26 --   Electrical and Computer Engineering
27 --   Montana State University
28 --   610 Cobleigh Hall
29 --   Bozeman, MT 59717
30 --   connor.dack@msu.montana.edu
31 --
32 ----------------------------------------------------------------------------
33 LIBRARY IEEE;
34 use IEEE.STD_LOGIC_1164.all;
35
36 use work.USER_CONTROLS.all;
37
38 ----------------------------------------------------------------------------
39 --
40 --! @brief     top
41 --
42 --! @param     fmc_clk_m2c_n      :  Input - Camera Link Pixel Line
43 --! @param     fmc_clk_m2c_p      :  Input - Camera Link Pixel Line
44 --! @param     fmc_ga             :  Input - Camera Link Pixel Line
45 --! @param     fmc_gpio_0         :  Input - Camera Link Pixel Line
46 --! @param     fmc_gpio_1         :  Input - Camera Link Pixel Line
47 --! @param     fmc_gpio_2         :  Input - Camera Link Pixel Line
48 --! @param     fmc_gpio_3         :  Input - Camera Link Pixel Line
49 --! @param     fmc_gpio_4         :  Input - Camera Link Pixel Line
50 --! @param     fmc_gpio_5         :  Input - Camera Link Pixel Line
51 --! @param     fmc_gpio_6         :  Input - Camera Link Pixel Line
52 --! @param     fmc_gpio_7         :  Input - Camera Link Pixel Line
53 --! @param     fmc_la_rx_clk_n    :  Input - Camera Link Pixel Line
```

```
54 --! @param     fmc_la_rx_clk_p    :  Input - Camera Link Pixel Line
55 --! @param     fmc_la_rx_n        :  Input - Camera Link Pixel Line
56 --! @param     fmc_la_rx_p        :  Input - Camera Link Pixel Line
57 --! @param     fmc_la_tx_clk_n    :  Input - Camera Link Pixel Line
58 --! @param     fmc_la_tx_clk_p    :  Input - Camera Link Pixel Line
59 --! @param     fmc_la_tx_n        :  Input - Camera Link Pixel Line
60 --! @param     fmc_la_tx_p        :  Input - Camera Link Pixel Line
61 --! @param     fmc_prsnt          :  Input - Camera Link Pixel Line
62 --! @param     fmc_scl            :  Input - Camera Link Pixel Line
63 --! @param     fmc_sda            :  Input - Camera Link Pixel Line
64 --! @param     user_pb_fpga       :  Input - Camera Link Pixel Line
65 --! @param     object_packet_out  : Output - Object Packet
66 --! @param     ignore_transceiver_obj_out  : Output - Write Enable
67 --! @param     system_mask_out_4_out : Output- Air Jet Control Sig
68 --! @param     air_jet_packet_out     : Output- Air Jet Packet
69 --! @param     air_jet_wre_out        : Output- Air Jet Write En
70 --! @param     current_line_out       : Output- Current Line
71 --
72 ----------------------------------------------------------------------------
73
74 entity top is
75     port(
76         fmc_clk_m2c_n    : in  std_logic_vector( 1 downto 0);
77         fmc_clk_m2c_p    : in  std_logic_vector( 1 downto 0);
78         fmc_ga           : in  std_logic_vector( 1 downto 0);
79         fmc_gpio_0       : in  std_logic;
80         fmc_gpio_1       : in  std_logic;
81         fmc_gpio_2       : in  std_logic;
82         fmc_gpio_3       : in  std_logic;
83         fmc_gpio_4       : in  std_logic;
84         fmc_gpio_5       : in  std_logic;
85         fmc_gpio_6       : in  std_logic;
86         fmc_gpio_7       : in  std_logic;
87         fmc_la_rx_clk_n  : in  std_logic;
88         fmc_la_rx_clk_p  : in  std_logic;
89         fmc_la_rx_n      : in  std_logic_vector(15 downto 0);
90         fmc_la_rx_p      : in  std_logic_vector(15 downto 0);
91         fmc_la_tx_clk_n  : in  std_logic;
92         fmc_la_tx_clk_p  : in  std_logic;
93         fmc_la_tx_n      : in  std_logic_vector(17 downto 0);
94         fmc_la_tx_p      : in  std_logic_vector(17 downto 0);
95         fmc_prsnt        : in  std_logic;
96         fmc_scl          : in  std_logic;
97         fmc_sda          : in  std_logic;
98
99         user_pb_fpga     : in  std_logic_vector( 1 downto 0);
100
101        object_packet_out          : out std_logic_vector(
```

```
102                                             TRANSCEIVER_DATA_SIZE
103                                             - 1 downto 0);
104         ignore_transceiver_obj_out : out std_logic;
105         system_mask_out_4_out      : out std_logic_vector(
106                                                   31 downto 0);
107         air_jet_packet_out         : out std_logic_vector(
108                                             OBJECT_ADDRESS_SIZE
109                                             + AOI_ADDRESS_SIZE +
110                                             LINE_NUMBER_ADDRESS_SIZE
111                                             - 1 downto 0);
112         air_jet_wre_out            : out std_logic;
113         current_line_out           : out std_logic_vector(
114                                              LINE_NUMBER_ADDRESS_SIZE
115                                              - 1 downto 0));
116 end entity;
117
118 architecture behaviour of top is
119
120     -- Component Decelerations
121     component camera_interface is
122         port(clk    :  in  std_logic;
123              rst_n  :  in  std_logic;
124
125              positive_threshold: in std_logic_vector(7 downto 0);
126              negative_threshold: in std_logic_vector(7 downto 0);
127
128              fmc_clk_m2c_n    : in  std_logic_vector( 1 downto 0);
129              fmc_clk_m2c_p    : in  std_logic_vector( 1 downto 0);
130              fmc_ga           : in  std_logic_vector( 1 downto 0);
131              fmc_gpio_0       : in  std_logic;
132              fmc_gpio_1       : in  std_logic;
133              fmc_gpio_2       : in  std_logic;
134              fmc_gpio_3       : in  std_logic;
135              fmc_gpio_4       : in  std_logic;
136              fmc_gpio_5       : in  std_logic;
137              fmc_gpio_6       : in  std_logic;
138              fmc_gpio_7       : in  std_logic;
139              fmc_la_rx_clk_n  : in  std_logic;
140              fmc_la_rx_clk_p  : in  std_logic;
141              fmc_la_rx_n      : in  std_logic_vector(15 downto 0);
142              fmc_la_rx_p      : in  std_logic_vector(15 downto 0);
143              fmc_la_tx_clk_n  : in  std_logic;
144              fmc_la_tx_clk_p  : in  std_logic;
145              fmc_la_tx_n      : in  std_logic_vector(17 downto 0);
146              fmc_la_tx_p      : in  std_logic_vector(17 downto 0);
147              fmc_prsnt        : in  std_logic;
148              fmc_scl          : in  std_logic;
149              fmc_sda          : in  std_logic;
```

```
150
151             pixel_array : out std_logic_vector(0 to
152                                                NUMBER_OF_TAPS - 1);
153             fval        : out std_logic;
154             x_lval      : out std_logic);
155     end component;
156
157     component object_tracking is
158         port(clk                          :  in std_logic;
159              rst_n                        :  in std_logic;
160              data_valid_in                :  in std_logic;
161              pixel_data_in                :  in std_logic_vector(0
162                                              to NUMBER_OF_TAPS - 1);
163              transciever_fifo_empty_in    :  in std_logic;
164              air_jet_fifo_empty_in        :  in std_logic;
165              linescan_offset_in           :  in std_logic_vector(
166                                                AOI_ADDRESS_SIZE -
167                                                1 downto 0);
168              object_packet_out            : out std_logic_vector(
169                                                TRANSCEIVER_DATA_SIZE
170                                                - 1 downto 0);
171              ignore_transceiver_obj_out   : out std_logic;
172              air_jet_packet_out           : out std_logic_vector(
173                                             OBJECT_ADDRESS_SIZE
174                                             + AOI_ADDRESS_SIZE +
175                                             LINE_NUMBER_ADDRESS_SIZE
176                                             - 1 downto 0);
177              air_jet_wre_out              : out std_logic;
178              current_line_out             : out std_logic_vector(
179                                             LINE_NUMBER_ADDRESS_SIZE
180                                             - 1 downto 0));
181     end component;
182
183     component pixel_clock is
184         port(
185             inclk  : in  std_logic := 'X'; -- inclk
186             outclk : out std_logic         -- outclk
187         );
188     end component;
189
190      component transceiver_fifo is
191         port(data    : in  std_logic_vector(63 downto 0)
192                                             := (others => 'X');
193              wrreq   : in  std_logic                    := 'X';
194              rdreq   : in  std_logic                    := 'X';
195              wrclk   : in  std_logic                    := 'X';
196              rdclk   : in  std_logic                    := 'X';
197              q       : out std_logic_vector(63 downto 0);
```

```
198             rdempty : out std_logic;
199             wrfull  : out std_logic);
200     end component transceiver_fifo;
201
202     component air_jet_fifo is
203         port(data    : in  std_logic_vector(31 downto 0)
204                                             := (others => 'X');
205             wrreq    : in  std_logic                := 'X';
206             rdreq    : in  std_logic                := 'X';
207             wrclk    : in  std_logic                := 'X';
208             rdclk    : in  std_logic                := 'X';
209             q        : out std_logic_vector(31 downto 0);
210             rdempty  : out std_logic;
211             wrfull   : out std_logic);
212     end component air_jet_fifo;
213
214     -- Internal Signals
215     signal pixel_0      : std_logic_vector(7 downto 0);
216     signal pixel_1      : std_logic_vector(7 downto 0);
217     signal pixel_2      : std_logic_vector(7 downto 0);
218     signal pixel_3      : std_logic_vector(7 downto 0);
219     signal pixel_4      : std_logic_vector(7 downto 0);
220     signal pixel_5      : std_logic_vector(7 downto 0);
221     signal pixel_6      : std_logic_vector(7 downto 0);
222     signal pixel_7      : std_logic_vector(7 downto 0);
223     signal pixel_8      : std_logic_vector(7 downto 0);
224     signal pixel_9      : std_logic_vector(7 downto 0);
225
226     signal fval         : std_logic;
227     signal x_lval       : std_logic;
228
229     signal x_clk        : std_logic;
230     signal y_clk        : std_logic;
231     signal z_clk        : std_logic;
232     signal combined_clk : std_logic;
233     signal pixel_clk    : std_logic;
234
235     signal rst_n : std_logic := '1';
236
237
238     signal pixel_array:std_logic_vector(0 to NUMBER_OF_TAPS - 1);
239
240     -- Object Thresholds
241     signal positive_threshold : std_logic_vector(7 downto 0)
242                                                 := x"FA";
243     signal negative_threshold : std_logic_vector(7 downto 0)
244                                                 := x"00";
245
```

```
246     -- Linescan to Hyperspectral Ratios
247     signal linescan_offset : std_logic_vector(AOI_ADDRESS_SIZE
248                                                 - 1 downto 0);
249     signal pixel_ratio     : std_logic_vector(31 downto 0);
250     signal lines_ratio     : std_logic_vector(31 downto 0);
251
252     -- Transciever FIFO
253     signal S_object_packet : std_logic_vector(
254                               TRANSCEIVER_DATA_SIZE-1  downto 0);
255     signal S_object_packet_out : std_logic_vector(
256                               TRANSCEIVER_DATA_SIZE-1  downto 0);
257     signal s_ignore_trans_obj  : std_logic;
258     signal transciever_fifo_rdreq : std_logic := '0';
259     signal transciever_fifo_read_clk : std_logic;
260     signal data_to_arria_10 : std_logic_vector(
261                              TRANSCEIVER_DATA_SIZE - 1 downto 0);
262     signal transciever_fifo_empty    : std_logic;
263     signal transciever_fifo_full     : std_logic;
264
265     -- Air Jet Team Information
266     signal s_air_jet_packet_out : std_logic_vector(
267                                     OBJECT_ADDRESS_SIZE +
268                                     AOI_ADDRESS_SIZE +
269                                     LINE_NUMBER_ADDRESS_SIZE
270                                     - 1 downto 0);
271     signal s_air_jet_wre_out    : std_logic;
272     signal s_current_line       : std_logic_vector(
273                                     LINE_NUMBER_ADDRESS_SIZE
274                                     - 1 downto 0);
275
276     -- Air Jet FIFO
277     signal s_object_air_jet     : std_logic_vector(
278                                     OBJECT_ADDRESS_SIZE +
279                                     AOI_ADDRESS_SIZE +
280                                     LINE_NUMBER_ADDRESS_SIZE
281                                     - 1 downto 0);
282     signal s_air_jet_fifo_empty : std_logic;
283     signal s_air_jet_fifo_full  : std_logic;
284     signal s_air_jet_fifo_rdreq : std_logic := '0';
285
286 begin
287   rst_n <= '1';
288
289     -- Port Maps
290     clk_ctrl : pixel_clock port map(
291         inclk           => combined_clk,
292         outclk          => pixel_clk
293     );
```

```
294
295     x_clk_ctrl : pixel_clock port map(
296         inclk           => fmc_la_tx_n(14),
297         outclk          => x_clk
298     );
299
300     y_clk_ctrl  : pixel_clock port map(
301         inclk           => fmc_la_tx_p(9),
302         outclk          => y_clk
303     );
304
305     z_clk_ctrl  : pixel_clock port map(
306         inclk           => fmc_la_tx_p(3),
307         outclk          => z_clk
308     );
309
310     waterfall_interface : camera_interface port map(
311         clk                 => pixel_clk,
312         rst_n               => rst_n,
313         positive_threshold  => positive_threshold,
314         negative_threshold  => negative_threshold,
315         fmc_clk_m2c_n       => fmc_clk_m2c_n,
316         fmc_clk_m2c_p       => fmc_clk_m2c_p,
317         fmc_ga              => fmc_ga,
318         fmc_gpio_0          => fmc_gpio_0,
319         fmc_gpio_1          => fmc_gpio_1,
320         fmc_gpio_2          => fmc_gpio_2,
321         fmc_gpio_3          => fmc_gpio_3,
322         fmc_gpio_4          => fmc_gpio_4,
323         fmc_gpio_5          => fmc_gpio_5,
324         fmc_gpio_6          => fmc_gpio_6,
325         fmc_gpio_7          => fmc_gpio_7,
326         fmc_la_rx_clk_n     => fmc_la_rx_clk_n,
327         fmc_la_rx_clk_p     => fmc_la_rx_clk_p,
328         fmc_la_rx_n         => fmc_la_rx_n,
329         fmc_la_rx_p         => fmc_la_rx_p,
330         fmc_la_tx_clk_n     => fmc_la_tx_clk_n,
331         fmc_la_tx_clk_p     => fmc_la_tx_clk_p,
332         fmc_la_tx_n         => fmc_la_tx_n,
333         fmc_la_tx_p         => fmc_la_tx_p,
334         fmc_prsnt           => fmc_prsnt,
335         fmc_scl             => fmc_scl,
336         fmc_sda             => fmc_sda,
337         pixel_array         => pixel_array,
338         fval                => fval,
339         x_lval              => x_lval
340     );
341
```

```
342     sys_object_tracking : object_tracking port map(
343         clk                       => pixel_clk,
344         rst_n                     => rst_n,
345         data_valid_in                 => fval,
346         pixel_data_in                 => pixel_array,
347         transciever_fifo_empty_in  => transciever_fifo_empty,
348         air_jet_fifo_empty_in      => s_air_jet_fifo_empty,
349         linescan_offset_in         => linescan_offset,
350         object_packet_out          => S_object_packet,
351         ignore_transceiver_obj_out => s_ignore_trans_obj,
352         air_jet_packet_out         => s_air_jet_packet_out,
353         air_jet_wre_out            => s_air_jet_wre_out,
354         current_line_out           => s_current_line);
355
356     object_packet_out             <= S_object_packet;
357     ignore_transceiver_obj_out    <= s_ignore_trans_obj;
358     air_jet_packet_out            <= s_air_jet_packet_out;
359     air_jet_wre_out               <= s_air_jet_wre_out;
360     current_line_out              <= s_current_line;
361
362       transciever_fifo_u0 : component transceiver_fifo
363         port map(data    => S_object_packet   ,
364                  wrreq   => s_ignore_trans_obj,
365                  rdreq   => transciever_fifo_rdreq,
366                  wrclk   => pixel_clk,
367                  rdclk   => transciever_fifo_read_clk,
368                  q       => s_object_packet_out,
369                  rdempty => transciever_fifo_empty,
370                  wrfull  => transciever_fifo_full);
371
372      air_jet_fifo_u0 : component air_jet_fifo
373         port map(data    => s_air_jet_packet_out,
374                  wrreq   => s_air_jet_wre_out,
375                  rdreq   => s_air_jet_fifo_rdreq,
376                  wrclk   => pixel_clk,
377                  rdclk   => pixel_clk,
378                  q       => s_object_air_jet,
379                  rdempty => s_air_jet_fifo_empty,
380                  wrfull  => s_air_jet_fifo_full);
381
382     combined_clk <= x_clk AND y_clk AND z_clk;
383 end architecture;
```

```
1  ----------------------------------------------------------------------
2  --
3  --! @file        object_tracking.vhd
4  --! @brief       Implementation of the object tracking algorithm.
5  --! @details
```

```
6  --
7  --! @author      Connor Dack
8  --! @date        November 2016
9  --! @copyright   Copyright (C) 2015 Ross K. Snider and Connor Dack
10 --
11 --   This program is free software: you can redistribute it
12 --   and/or modify it under the terms of the GNU General
13 --   Public License as published by the Free Software Foundation,
14 --   either version 3 of the License, or (at your option) any
15 --   later version.
16 --
17 --   This program is distributed in the hope that it will be
18 --   useful, but WITHOUT ANY WARRANTY; without even the implied
19 --   warranty of MERCHANTABILITY or FITNESS for A PARTICULAR
20 --   PURPOSE. See the GNU General Public License for more details.
21 --
22 --   You should have received a copy of the GNU General Public
23 --   License along with this program.  If not, see
24 --   <http://www.gnu.org/licenses/>.
25 --
26 --   Connor Dack
27 --   Electrical and Computer Engineering
28 --   Montana State University
29 --   610 Cobleigh Hall
30 --   Bozeman, MT 59717
31 --   connor.dack@msu.montana.edu
32 --
33 ----------------------------------------------------------------------------
34
35 LIBRARY IEEE;
36 use IEEE.STD_LOGIC_1164.all;
37 use IEEE.NUMERIC_STD.all;
38
39 use work.USER_CONTROLS.all;
40
41 ----------------------------------------------------------------------------
42 --
43 --! @brief       object_tracking
44 --! @details
45 --!
46 --! @param      clk                          :  Input: Component Clock
47 --! @param      rst_n                        :  Input: Active low reset
48 --! @param      data_valid_in                :  Input: Data Valid Flag
49 --! @param      pixel_data_in                :  Input: Pixel Data Vector
50 --! @param      transciever_fifo_empty_in    : Input: FIFO Empty Flag
51 --! @param      air_jet_fifo_empty_in        : Input: FIFO Empty Flag
52 --! @param      linescan_offset_in           : Input: Linescan offset
53 --! @param      object_packet_out            : Output: Data packet
```

```
54 --! @param      ignore_transceiver_obj_out : Output: ignore object
55 --! @param      air_jet_packet_out         : Output: Air Jet Packet
56 --! @param      air_jet_wre_out            : Output: Air Jet Wr En
57 --! @param      current_line_out           : Output: Current Line
58 --
59 ----------------------------------------------------------------------------------
60
61 entity object_tracking is
62     port(clk                           :  in std_logic;
63          rst_n                         :  in std_logic;
64          data_valid_in                 :  in std_logic;
65          pixel_data_in                 :  in std_logic_vector(0 to
66                                             NUMBER_OF_TAPS - 1);
67          transciever_fifo_empty_in     :  in std_logic;
68          air_jet_fifo_empty_in         :  in std_logic;
69          linescan_offset_in            :  in std_logic_vector(
70                                             AOI_ADDRESS_SIZE -
71                                             1 downto 0);
72          object_packet_out             : out std_logic_vector(
73                                             TRANSCEIVER_DATA_SIZE
74                                             - 1 downto 0);
75          ignore_transceiver_obj_out    : out std_logic;
76          air_jet_packet_out            : out std_logic_vector(
77                                             OBJECT_ADDRESS_SIZE
78                                             + AOI_ADDRESS_SIZE +
79                                             LINE_NUMBER_ADDRESS_SIZE
80                                             - 1 downto 0);
81          air_jet_wre_out               : out std_logic;
82          current_line_out              : out std_logic_vector(
83                                             LINE_NUMBER_ADDRESS_SIZE
84                                             - 1 downto 0));
85 end entity;
86
87 architecture behaviour of object_tracking is
88
89     -- Component Decleration for Module 2
90     component mod2_tap_in is
91         port(clk                          : in std_logic;
92              rst_n                        : in std_logic;
93              enable_in                    : in std_logic;
94              tap_data_in                  : in std_logic_vector(0 to
95                                               NUMBER_OF_TAPS - 1);
96              transciever_fifo_empty_in    : in std_logic;
97              air_jet_fifo_empty_in        : in std_logic;
98              ss1_start_out                : out std_logic_vector(3
99                                                          downto 0);
100             ss1_end_out                  : out std_logic_vector(3
101                                                         downto 0);
```

```
102             ss2_start_out               : out std_logic_vector(3
103                                                 downto 0);
104             ss2_end_out                 : out std_logic_vector(3
105                                                 downto 0);
106             ss_ptr_out                  : out std_logic_vector(1
107                                                 downto 0);
108             ss_ready_out                : out std_logic;
109             line_number_out             : out std_logic_vector(
110                         LINE_NUMBER_ADDRESS_SIZE - 1 downto 0);
111             line_rst_out                : out std_logic;
112             end_of_line_out             : out std_logic);
113     end component;
114
115     -- Component Decleration for Module 3
116     component mod3_ss2seg is
117         port(clk            :  in std_logic;
118             rst_n           :  in std_logic;
119             ss_ready_in     :  in std_logic;
120             ss_ptr_in       :  in std_logic_vector(1 downto 0);
121             ss1_start_in    :  in std_logic_vector(3 downto 0);
122             ss1_end_in      :  in std_logic_vector(3 downto 0);
123             ss2_start_in    :  in std_logic_vector(3 downto 0);
124             ss2_end_in      :  in std_logic_vector(3 downto 0);
125             line_number_in  :  in std_logic_vector(
126                                     LINE_NUMBER_ADDRESS_SIZE
127                                     - 1 downto 0);
128             line_rst_in     :  in std_logic;
129             end_of_line_in  :  in std_logic;
130             seg_start_out   : out seg_info_array;
131             seg_mid_out     : out seg_info_array;
132             seg_end_out     : out seg_info_array;
133             seg_ptr_out     : out std_logic_vector(9 downto 0));
134     end component;
135
136     -- Component Decleration for Module 4
137     component mod4_seg2obj is
138         port(clk            :  in std_logic;
139          rst_n          :  in std_logic;
140             line_number             :  in std_logic_vector(
141                                       LINE_NUMBER_ADDRESS_SIZE - 1
142                                       downto 0);
143          end_of_line        :  in std_logic;
144             seg_start_in            :  in seg_info_array;
145             seg_mid_in              :  in seg_info_array;
146             seg_end_in              :  in seg_info_array;
147          mod_5_trigger_out      : out std_logic;
148             line_scan_packet_out    : out std_logic_vector(
149                                       OBJECT_ADDRESS_SIZE +
```

```
150                                         LINE_NUMBER_ADDRESS_SIZE +
151                                         (3 * AOI_ADDRESS_SIZE) - 1
152                                         downto 0);
153             mod_4_wr_en_out         : out std_logic);
154     end component;
155
156     -- Component Decleration for Camera Relations Component
157     component camera_relations is
158         port(clk                        : in std_logic;
159              rst_n                      : in std_logic;
160              line_reset                 : in std_logic;
161              linescan_offset            : in std_logic_vector(
162                                         AOI_ADDRESS_SIZE-1 downto 0);
163              line_scan_packet_in        : in std_logic_vector((
164                                         OBJECT_ADDRESS_SIZE +
165                                         LINE_NUMBER_ADDRESS_SIZE +
166                                         AOI_ADDRESS_SIZE +
167                                         AOI_ADDRESS_SIZE) - 1
168                                         downto 0);
169              hyperspectral_packet_out : out std_logic_vector(
170                                         TRANSCEIVER_DATA_SIZE - 1
171                                         downto 0);
172              transciever_obj_wre_out  : out std_logic);
173     end component;
174
175     -- Component Decleration for a generic memory block
176     component memory_block is
177         generic (
178             num_elements_a  : natural := 65536;
179             num_elements_b  : natural := 65536;
180             size_address_a  : natural := 16;
181             size_address_b  : natural := 16;
182             size_word_a     : natural := 32;
183             size_word_b     : natural := 32;
184             mem_init        : string := "UNUSED");
185         port (
186             address_a       : in  std_logic_vector(size_address_a
187                                                    - 1 downto 0);
188             address_b       : in  std_logic_vector(size_address_b
189                                                    - 1 downto 0);
190             clock_a         : in  std_logic  := '1';
191             clock_b         : in  std_logic  := '1';
192             data_a          : in  std_logic_vector(size_word_a -
193                                                     1 downto 0);
194             data_b          : in  std_logic_vector(size_word_b -
195                                                     1 downto 0);
196             wren_a          : in  std_logic  := '0';
197             wren_b          : in  std_logic  := '0';
```

```
198             q_a                 : out std_logic_vector(size_word_a -
199                                                         1 downto 0);
200             q_b                 : out std_logic_vector(size_word_b -
201                                                         1 downto 0));
202     end component;
203
204     -- Component Decleration for Module 5
205     component mod5_calc_y_center is
206         port(clk            :  in std_logic;
207          rst_n              :  in std_logic;
208               mod_5_trigger_in     :  in std_logic;
209               line_scan_packet_in  :  in std_logic_vector(
210                                          OBJECT_ADDRESS_SIZE +
211                                          LINE_NUMBER_ADDRESS_SIZE +
212                                          (3 * AOI_ADDRESS_SIZE) - 1
213                                          downto 0);
214               object_y_data_out    : out std_logic_vector(
215                                          OBJECT_ADDRESS_SIZE + (3 *
216                                          LINE_NUMBER_ADDRESS_SIZE)
217                                          - 1 downto 0);
218               mod_5_wr_en_out      : out std_logic);
219     end component;
220
221     -- Signals for component inputs
222     signal s_data_valid                 : std_logic;
223     signal s_pixel_array                : std_logic_vector(0 to
224                                           NUMBER_OF_TAPS - 1);
225     signal s_transciever_fifo_empty     : std_logic;
226     signal s_air_jet_fifo_empty         : std_logic;
227     signal s_linescan_offset            : std_logic_vector(
228                                           AOI_ADDRESS_SIZE - 1
229                                           downto 0);
230
231     -- Outputs of mod2
232     signal s_ss1_start, s_ss1_end : std_logic_vector(3 downto 0);
233     signal s_ss2_start, s_ss2_end : std_logic_vector(3 downto 0);
234     signal s_ss_ptr               : std_logic_vector(1 downto 0);
235     signal s_ss_ready             : std_logic;
236     signal s_line_number          : std_logic_vector(
237                                           LINE_NUMBER_ADDRESS_SIZE
238                                           - 1 downto 0);
239     signal s_line_rst             : std_logic := '1';
240     signal s_end_of_line          : std_logic := '0';
241
242     -- Outputs of mod3
243     signal eol_flag     : std_logic;
244     signal seg_ptr      : std_logic_vector(9 downto 0);
245     signal s_seg_start  : seg_info_array := (others =>
```

```
246                                                 (others => '0'));
247     signal s_seg_end    : seg_info_array := (others =>
248                                                 (others => '0'));
249     signal s_seg_mid    : seg_info_array := (others =>
250                                                 (others => '0'));
251
252     -- Outputs of mod4
253     type array_of_line_numbers   is array(1 to MAX_OBJECT_NUMBER
254                                  ) of std_logic_vector(
255                                  LINE_NUMBER_ADDRESS_SIZE -
256                                  1 downto 0);
257     type array_of_wr_en          is array(1 to MAX_OBJECT_NUMBER
258                                  ) of std_logic;
259     type array_of_data           is array(1 to MAX_OBJECT_NUMBER
260                                  ) of std_logic_vector(3 *
261                                  AOI_ADDRESS_SIZE - 1 downto 0);
262     type array_of_center_x_value is array(0 to MAX_OBJECT_NUMBER
263                                  ) of std_logic_vector(
264                                  AOI_ADDRESS_SIZE - 1 downto 0);
265
266     signal s_mod_5_trigger      : std_logic;
267     signal s_line_scan_packet   : std_logic_vector(
268                                   OBJECT_ADDRESS_SIZE +
269                                   LINE_NUMBER_ADDRESS_SIZE +
270                                   (3 * AOI_ADDRESS_SIZE) - 1
271                                   downto 0);
272     signal s_mod_4_wr_en        : std_logic := '0';
273
274     signal s_array_of_mod_4_line_numbers : array_of_line_numbers
275                                            := (others =>
276                                               (others => '0'));
277     signal s_array_of_mod_4_wr_en        : array_of_wr_en
278                                            := (others => '0');
279     signal s_array_of_mod_4_data         : array_of_center_x_value
280                                            := (others =>
281                                               (others => '0'));
282
283      -- Object Information
284     signal s_transceiver_line_scan_packet : std_logic_vector(
285                                             OBJECT_ADDRESS_SIZE +
286                                             LINE_NUMBER_ADDRESS_SIZE
287                                             + (2 * AOI_ADDRESS_SIZE)
288                                             - 1 downto 0);
289     signal s_ignore_transceiver_obj         : std_logic;
290
291     -- Outputs of mod5
292     signal s_object_y_data : std_logic_vector(OBJECT_ADDRESS_SIZE
293                              + (3 * LINE_NUMBER_ADDRESS_SIZE) - 1
```

```
294                                 downto 0);
295     signal s_mod_5_data : std_logic_vector(OBJECT_ADDRESS_SIZE
296                                 + (3 * LINE_NUMBER_ADDRESS_SIZE) - 1
297                                 downto 0);
298     signal s_array_of_mod_5_line_numbers : array_of_line_numbers
299                                         := (others =>
300                                             (others => '0'));
301     signal s_array_of_mod_5_data    : array_of_center_x_value
302                                         := (others =>
303                                             (others => '0'));
304     signal s_mod_5_wr_en_mem_block : std_logic := '0';
305     signal s_mod_5_wr_data         : std_logic_vector(
306                                      AOI_ADDRESS_SIZE - 1
307                                      downto 0) := (others => '0');
308     signal s_mod_5_wr_en : std_logic := '0';
309     signal s_air_jet_wre : std_logic := '0';
310     signal f_delay_one_clk_cycle : std_logic := '0';
311     signal s_air_jet_packet : std_logic_vector(
312                                         OBJECT_ADDRESS_SIZE
313                                         + AOI_ADDRESS_SIZE +
314                                         LINE_NUMBER_ADDRESS_SIZE
315                                         - 1 downto 0);
316
317 begin
318     -- Assign the internal signal components
319     s_data_valid             <= data_valid_in;
320     s_pixel_array            <= pixel_data_in;
321     s_transciever_fifo_empty <= transciever_fifo_empty_in;
322     s_air_jet_fifo_empty     <= air_jet_fifo_empty_in;
323     s_linescan_offset        <= linescan_offset_in;
324
325     -- Route the signals to Module 2
326     mod2 : mod2_tap_in port map(
327         clk                         => clk,
328         rst_n                       => rst_n,
329         enable_in                   => s_data_valid,
330         tap_data_in                 => s_pixel_array,
331         transciever_fifo_empty_in   => s_transciever_fifo_empty,
332         air_jet_fifo_empty_in       => s_air_jet_fifo_empty,
333         ss1_start_out               => s_ss1_start,
334         ss1_end_out                 => s_ss1_end,
335         ss2_start_out               => s_ss2_start,
336         ss2_end_out                 => s_ss2_end,
337         ss_ptr_out                  => s_ss_ptr,
338         ss_ready_out                => s_ss_ready,
339         line_number_out             => s_line_number,
340         line_rst_out                => s_line_rst,
341         end_of_line_out             => s_end_of_line);
```

```
342
343     -- Route the signals to Module 3
344     mod3 : mod3_ss2seg port map(
345         clk              => clk,
346         rst_n            => rst_n,
347         ss_ready_in      => s_ss_ready,
348         ss_ptr_in        => s_ss_ptr,
349         ss1_start_in     => s_ss1_start,
350         ss1_end_in       => s_ss1_end,
351         ss2_start_in     => s_ss2_start,
352         ss2_end_in       => s_ss2_end,
353         line_number_in   => s_line_number,
354         line_rst_in      => s_line_rst,
355         end_of_line_in   => s_end_of_line,
356         seg_start_out    => s_seg_start,
357         seg_mid_out      => s_seg_mid,
358         seg_end_out      => s_seg_end,
359         seg_ptr_out      => seg_ptr);
360
361     -- Route the signals to Module 4
362     mod4 : mod4_Seg2Obj port map(
363         clk                      => clk,
364         rst_n                    => rst_n,
365         line_number              => s_line_number,
366         end_of_line              => s_end_of_line,
367         seg_start_in             => s_seg_start,
368         seg_mid_in               => s_seg_mid,
369         seg_end_in               => s_seg_end,
370         mod_5_trigger_out        => s_mod_5_trigger,
371         line_scan_packet_out     => s_line_scan_packet,
372         mod_4_wr_en_out          => s_mod_4_wr_en);
373
374     -- Create the Line Scan Packet that is sent to the
375     -- Hyperspectral Processing unit
376     s_transceiver_line_scan_packet <= s_line_scan_packet(
377                                       OBJECT_ADDRESS_SIZE +
378                                       LINE_NUMBER_ADDRESS_SIZE +
379                                       (3 * AOI_ADDRESS_SIZE) - 1
380                                       downto AOI_ADDRESS_SIZE);
381
382     -- Assign the infomration from Module 4 to write the object
383     -- information to the memory block
384     mod_4_memory_write_process : process(clk,rst_n)
385         begin
386             if(rst_n = '0') then -- On reset,
387                 s_array_of_mod_4_line_numbers <=(others =>
388                                                  (others => '0'));
389                 s_array_of_mod_4_wr_en <=(others => '0');
```

```
390                 s_array_of_mod_4_data  <=(others =>
391                                           (others => '0'));
392             elsif(rising_edge(clk)) then
393                 if(s_mod_4_wr_en = '1') then
394                     s_array_of_mod_4_line_numbers(to_integer(
395                         unsigned(s_line_scan_packet(
396                         OBJECT_ADDRESS_SIZE +
397                         LINE_NUMBER_ADDRESS_SIZE + (3 *
398                         AOI_ADDRESS_SIZE) - 1 downto
399                         LINE_NUMBER_ADDRESS_SIZE + (3 *
400                         AOI_ADDRESS_SIZE)))))
401                                 <= s_line_scan_packet(
402                                   LINE_NUMBER_ADDRESS_SIZE +
403                                   (3 * AOI_ADDRESS_SIZE) - 1
404                                   downto (3 * AOI_ADDRESS_SIZE));
405                     s_array_of_mod_4_wr_en(to_integer(unsigned(
406                           s_line_scan_packet(OBJECT_ADDRESS_SIZE
407                           + LINE_NUMBER_ADDRESS_SIZE + (3 *
408                           AOI_ADDRESS_SIZE) - 1 downto
409                           LINE_NUMBER_ADDRESS_SIZE + (3 *
410                           AOI_ADDRESS_SIZE))))) <= '1';
411                     s_array_of_mod_4_data(to_integer(unsigned(
412                         s_line_scan_packet(OBJECT_ADDRESS_SIZE +
413                         LINE_NUMBER_ADDRESS_SIZE + (3 *
414                         AOI_ADDRESS_SIZE) - 1 downto
415                         LINE_NUMBER_ADDRESS_SIZE + (3 *
416                         AOI_ADDRESS_SIZE)))))
417                                 <= s_line_scan_packet((2 *
418                                     AOI_ADDRESS_SIZE) - 1 downto
419                                     AOI_ADDRESS_SIZE);
420                 else
421                     s_array_of_mod_4_line_numbers <= (others =>
422                                                 (others => '0'));
423                     s_array_of_mod_4_wr_en <=(others => '0');
424                     s_array_of_mod_4_data  <=(others =>
425                                                 (others => '0'));
426                 end if;
427             end if;
428         end process;
429
430     -- Route the signals to Camera Relations
431     line_scan_to_hyperspectral : camera_relations  port map (
432           clk                   => clk,
433           rst_n                 => rst_n,
434           line_reset            => s_line_rst,
435           linescan_offset       => s_linescan_offset,
436           line_scan_packet_in   => s_transceiver_line_scan_packet,
437           hyperspectral_packet_out => object_packet_out,
```

```
438             transciever_obj_wre_out  => s_ignore_transceiver_obj);
439
440         -- Generate the correct number of memory blocks
441         memory_block_generate : for i in 1 to MAX_OBJECT_NUMBER
442                                                         - 1 generate
443             object_data : memory_block generic map (
444                 num_elements_a  => 65536,
445                 num_elements_b  => 65536,
446                 size_address_a  => LINE_NUMBER_ADDRESS_SIZE,
447                 size_address_b  => LINE_NUMBER_ADDRESS_SIZE,
448                 size_word_a     => AOI_ADDRESS_SIZE,
449                 size_word_b     => AOI_ADDRESS_SIZE,
450                 mem_init        => "UNUSED" )
451             port map(
452                 address_a => s_array_of_mod_4_line_numbers(i),
453                 clock_a   => clk,
454                 data_a    => s_array_of_mod_4_data(i),
455                 wren_a    => s_array_of_mod_4_wr_en(i),
456                 q_a       => open,
457                 address_b => s_array_of_mod_5_line_numbers(i),
458                 data_b    => s_mod_5_wr_data,
459                 wren_b    => s_mod_5_wr_en_mem_block,
460                 clock_b   => clk,
461                 q_b       => s_array_of_mod_5_data(i));
462         end generate;
463
464
465         mod_5_memory_read_process : process(clk,rst_n)
466             begin
467                 if(rst_n = '0') then -- On reset,
468                     s_array_of_mod_5_line_numbers <= (others =>
469                                                     (others => '0'));
470                 elsif(rising_edge(clk)) then
471                     if(s_mod_5_wr_en = '1') then
472                         s_array_of_mod_5_line_numbers(to_integer(
473                               unsigned(s_object_y_data(
474                               OBJECT_ADDRESS_SIZE + (3 *
475                               LINE_NUMBER_ADDRESS_SIZE) - 1
476                               downto (3 *LINE_NUMBER_ADDRESS_SIZE)))))
477                                     <= s_object_y_data((2 *
478                                     LINE_NUMBER_ADDRESS_SIZE) - 1
479                                     downto LINE_NUMBER_ADDRESS_SIZE);
480                             -- Determine the address to read the
481                             -- horizontal center from memory
482                     else
483                         s_array_of_mod_5_line_numbers <= (others =>
484                                                     (others => '0'));
485                     end if;
```

```
486             end if;
487         end process;
488 
489 
490         -- Route the signals to Module 5
491         mod_5 : mod5_calc_y_center port map(
492             clk                 => clk,
493             rst_n               => rst_n,
494             mod_5_trigger_in    => s_mod_5_trigger,
495             line_scan_packet_in => s_line_scan_packet,
496             object_y_data_out   => s_object_y_data,
497             mod_5_wr_en_out     => s_mod_5_wr_en);
498 
499         -- Create the air jet packet using a process
500         air_jet_packet_process : process(clk,rst_n)
501             begin
502                 if(rst_n = '0') then -- On reset,
503                     f_delay_one_clk_cycle <= '0';
504                     -- clear the delay flag
505                     s_mod_5_data <= (others => '0');
506                     -- clear the registered data
507                     s_air_jet_wre <= '0';
508                     -- Clear the Air Jet Write Enable
509                     s_air_jet_packet <= (others => '0');
510                     -- Clear the air jet packet
511                 elsif(rising_edge(clk)) then
512                 -- On the rising edge of the component clock
513                     if(s_mod_5_wr_en = '1') then
514                     -- If the Mod 5 write enable is a one,
515                         f_delay_one_clk_cycle <= '1';
516                         --set the delay flag
517                         s_mod_5_data <= s_object_y_data;
518                         -- register the vertical data
519                         s_air_jet_wre <= '0';
520                         -- clear the air jet write enable
521                     elsif(f_delay_one_clk_cycle <= '1') then
522                     -- If the delay flag is set, then
523                         s_air_jet_wre <= '1';
524                         -- Set the air jet write enable
525                         s_air_jet_packet <= s_mod_5_data(
526                                             OBJECT_ADDRESS_SIZE + (3 *
527                                             LINE_NUMBER_ADDRESS_SIZE) - 1
528                                             downto (3 *
529                                             LINE_NUMBER_ADDRESS_SIZE)) &
530                                             s_array_of_mod_5_data(to_integer(
531                                             unsigned(s_mod_5_data(
532                                             OBJECT_ADDRESS_SIZE + (3 *
533                                             LINE_NUMBER_ADDRESS_SIZE) - 1
```

```
534                                     downto (3 *
535                                     LINE_NUMBER_ADDRESS_SIZE))))
536                                     & s_mod_5_data((2 *
537                                     LINE_NUMBER_ADDRESS_SIZE) - 1
538                                     downto LINE_NUMBER_ADDRESS_SIZE);
539                         -- create the air jet packet
540                         f_delay_one_clk_cycle <= '0';
541                         -- clear the delay flag
542                     else -- Otherwise,
543                         s_air_jet_wre <= '0';
544                         -- clear the air jet write enable
545                         s_air_jet_packet <= (others => '0');
546                         -- clear the air jet packet
547                         s_mod_5_data <= (others => '0');
548                         -- clear the registered data
549                         f_delay_one_clk_cycle <= '0';
550                         -- clear the delay flag
551                     end if;
552                 end if;
553             end process;
554 
555 
556         -- Assign the outputs
557         current_line_out            <= s_line_number;
558         ignore_transceiver_obj_out <= s_ignore_transceiver_obj;
559         air_jet_wre_out             <= s_air_jet_wre;
560         air_jet_packet_out          <= s_air_jet_packet;
561 
562 end architecture;
```

```
1  ----------------------------------------------------------------------------
2  --
3  --! @file        mod1_pixel_to_array.vhd
4  --! @brief       Module 1 is the pixel to array conversion.
5  --! @details     Module 1 looks at each individual pixel and
6  --!              converts the pixel value into a 1 if the
7  --!              monochrome data is within the given threshold.
8  --!              The conversion is done using subtraction. The
9  --!              module is looking for a dark object on a white
10 --!              background.
11 --
12 --! @author      Connor Dack
13 --! @date        November 2016
14 --! @copyright   Copyright (C) 2015 Ross K. Snider and Connor Dack
15 --
16 --   This program is free software: you can redistribute it
17 --   and/or modify it under the terms of the GNU General
18 --   Public License as published by the Free Software Foundation,
```

```
19 --   either version 3 of the License, or (at your option) any
20 --   later version.
21 --
22 --   This program is distributed in the hope that it will be
23 --   useful, but WITHOUT ANY WARRANTY; without even the implied
24 --   warranty of MERCHANTABILITY or FITNESS for A PARTICULAR
25 --   PURPOSE. See the GNU General Public License for more details.
26 --
27 --   You should have received a copy of the GNU General Public
28 --   License along with this program.  If not, see
29 --   <http://www.gnu.org/licenses/>.
30 --
31 --   Connor Dack
32 --   Electrical and Computer Engineering
33 --   Montana State University
34 --   610 Cobleigh Hall
35 --   Bozeman, MT 59717
36 --   connor.dack@msu.montana.edu
37 --
38 ----------------------------------------------------------------------------
39
40 library IEEE;
41 use IEEE.STD_LOGIC_1164.all;
42 use IEEE.NUMERIC_STD.all;
43
44 use work.USER_CONTROLS.all;
45
46 ----------------------------------------------------------------------------
47 --
48 --! @brief     mod1_pixel_to_array
49 --! @details
50 --!
51 --! @param     clk                     -  Input: System Clock
52 --! @param     rst                     -  Input: Active low reset
53 --!                                          signal from HPS
54 --! @param     positive_threshold_in   -  Input:  The upper threshold
55 --!                                          - 255 is pure white
56 --! @param     negative_threshold_in   -  Input:  The lower threshold
57 --!                                          -   0 is pure black
58 --! @param     pixel_0_in              -  Input: Pixel 0 is a standard
59 --!                                       logic value between 0-255
60 --! @param     pixel_1_in              -  Input: Pixel 1 is a standard
61 --!                                       logic value between 0-255
62 --! @param     pixel_2_in              -  Input: Pixel 2 is a standard
63 --!                                       logic value between 0-255
64 --! @param     pixel_3_in              -  Input: Pixel 3 is a standard
65 --!                                       logic value between 0-255
66 --! @param     pixel_4_in              -  Input: Pixel 4 is a standard
```

```
67  --!                                          logic value between 0-255
68  --! @param     pixel_5_in           -  Input: Pixel 5 is a standard
69  --!                                          logic value between 0-255
70  --! @param     pixel_6_in           -  Input: Pixel 6 is a standard
71  --!                                          logic value between 0-255
72  --! @param     pixel_7_in           -  Input: Pixel 7 is a standard
73  --!                                          logic value between 0-255
74  --! @param     pixel_8_in           -  Input: Pixel 8 is a standard
75  --!                                          logic value between 0-255
76  --! @param     pixel_9_in           -  Input: Pixel 9 is a standard
77  --!                                          logic value between 0-255
78  --! @param     pixel_array_data_out- Output:  The pixel array data
79  --!                                    is the output data with the
80  --!                                  MSB (0) representing pixel 0
81  --!                                and LSB (9) representing pixel 9
82  --
83  ----------------------------------------------------------------------------
84  entity mod1_pixel_to_array is
85    port(clk                    :  in  std_logic;
86         rst                    :  in  std_logic;
87         positive_threshold_in  :  in  std_logic_vector(7 downto 0);
88         negative_threshold_in  :  in  std_logic_vector(7 downto 0);
89         pixel_0_in             :  in  std_logic_vector(7 downto 0);
90         pixel_1_in             :  in  std_logic_vector(7 downto 0);
91         pixel_2_in             :  in  std_logic_vector(7 downto 0);
92         pixel_3_in             :  in  std_logic_vector(7 downto 0);
93         pixel_4_in             :  in  std_logic_vector(7 downto 0);
94         pixel_5_in             :  in  std_logic_vector(7 downto 0);
95         pixel_6_in             :  in  std_logic_vector(7 downto 0);
96         pixel_7_in             :  in  std_logic_vector(7 downto 0);
97         pixel_8_in             :  in  std_logic_vector(7 downto 0);
98         pixel_9_in             :  in  std_logic_vector(7 downto 0);
99         pixel_array_data_out   :  out std_logic_vector(0    to 9));
100 end entity;
101
102
103 architecture rtl of mod1_pixel_to_array is
104
105     signal s_positive_threshold : std_logic_vector(7 downto 0)
106                 := (others => '0'); -- Positive threshold signal
107     signal s_negative_threshold : std_logic_vector(7 downto 0)
108                 := (others => '0'); -- Negative threshold signal
109     signal s_pixel_0            : std_logic_vector(7 downto 0)
110                 := (others => '0'); -- Pixel 0 signal
111     signal s_pixel_1            : std_logic_vector(7 downto 0)
112                 := (others => '0'); -- Pixel 1 signal
113     signal s_pixel_2            : std_logic_vector(7 downto 0)
114                 := (others => '0'); -- Pixel 2 signal
```

```
115     signal s_pixel_3            : std_logic_vector(7 downto 0)
116                 := (others => '0'); -- Pixel 3 signal
117     signal s_pixel_4            : std_logic_vector(7 downto 0)
118                 := (others => '0'); -- Pixel 4 signal
119     signal s_pixel_5            : std_logic_vector(7 downto 0)
120                 := (others => '0'); -- Pixel 5 signal
121     signal s_pixel_6            : std_logic_vector(7 downto 0)
122                 := (others => '0'); -- Pixel 6 signal
123     signal s_pixel_7            : std_logic_vector(7 downto 0)
124                 := (others => '0'); -- Pixel 7 signal
125     signal s_pixel_8            : std_logic_vector(7 downto 0)
126                 := (others => '0'); -- Pixel 8 signal
127     signal s_pixel_9            : std_logic_vector(7 downto 0)
128                 := (others => '0'); -- Pixel 9 signal
129     signal s_pixel_array_data   : std_logic_vector(0     to 9)
130                 := (others => '0'); -- Pixel Array Data Output
131
132 begin
133     -- Assign the input signal to the internal component signals
134     s_positive_threshold <= positive_threshold_in; -- Positive
135                                                    -- threshold signal
136     s_negative_threshold <= negative_threshold_in; -- Negative
137                                                    -- threshold signal
138     s_pixel_0            <= pixel_0_in;          -- Pixel 0 signal
139     s_pixel_1            <= pixel_1_in;          -- Pixel 1 signal
140     s_pixel_2            <= pixel_2_in;          -- Pixel 2 signal
141     s_pixel_3            <= pixel_3_in;          -- Pixel 3 signal
142     s_pixel_4            <= pixel_4_in;          -- Pixel 4 signal
143     s_pixel_5            <= pixel_5_in;          -- Pixel 5 signal
144     s_pixel_6            <= pixel_6_in;          -- Pixel 6 signal
145     s_pixel_7            <= pixel_7_in;          -- Pixel 7 signal
146     s_pixel_8            <= pixel_8_in;          -- Pixel 8 signal
147     s_pixel_9            <= pixel_9_in;          -- Pixel 9 signal
148
149     -- The process to convert the pixel data into the pixel array
150     pixel_value_to_bit_proc : process(clk, rst)
151         begin
152             if(rst = '1') then -- On reset,
153                 s_pixel_array_data <= (others => '0');
154                 -- Clear the pixel data array output
155             elsif(rising_edge(clk)) then -- Trigger on the rising
156                                          -- edge of the system clock,
157                 if(s_pixel_0 > s_negative_threshold and s_pixel_0
158                   < s_positive_threshold) then -- If pixel 0 is
159                   -- with in the threshold, then
160                     s_pixel_array_data(0) <= '1'; -- Set bit 0 in
161                     -- the output array to 1
162                 else -- Otherwise,
```

```
163                 s_pixel_array_data(0) <= '0'; -- Clear bit 0
164             end if;
165             if(s_pixel_1 > s_negative_threshold and s_pixel_1
166                < s_positive_threshold) then -- If pixel 1 is
167                -- with in the threshold, then
168                 s_pixel_array_data(1) <= '1'; -- Set bit 1 in
169                 -- the output array to 1
170             else -- Otherwise,
171                 s_pixel_array_data(1) <= '0'; -- Clear bit 1
172             end if;
173             if(s_pixel_2 > s_negative_threshold and s_pixel_2
174                < s_positive_threshold) then -- If pixel 2 is
175                -- with in the threshold, then
176                 s_pixel_array_data(2) <= '1'; -- Set bit 2 in
177                 -- the output array to 1
178             else -- Otherwise,
179                 s_pixel_array_data(2) <= '0';  -- Clear bit 2
180             end if;
181             if(s_pixel_3 > s_negative_threshold and s_pixel_3
182                < s_positive_threshold) then -- If pixel 3 is
183                -- with in the threshold, then
184                 s_pixel_array_data(3) <= '1'; -- Set bit 3 in
185                 -- the output array to 1
186             else -- Otherwise,
187                 s_pixel_array_data(3) <= '0'; -- Clear bit 3
188             end if;
189             if(s_pixel_4 > s_negative_threshold and s_pixel_4
190                < s_positive_threshold) then -- If pixel 4 is
191                -- with in the threshold, then
192                 s_pixel_array_data(4) <= '1'; -- Set bit 4 in
193                -- the output array to 1
194             else -- Otherwise,
195                 s_pixel_array_data(4) <= '0'; -- Clear bit 4
196             end if;
197             if(s_pixel_5 > s_negative_threshold and s_pixel_5
198                < s_positive_threshold) then -- If pixel 5 is
199                -- with in the threshold, then
200                 s_pixel_array_data(5) <= '1'; -- Set bit 5 in
201                 -- the output array to 1
202             else -- Otherwise,
203                 s_pixel_array_data(5) <= '0'; -- Clear bit 5
204             end if;
205             if(s_pixel_6 > s_negative_threshold and s_pixel_6
206                < s_positive_threshold) then -- If pixel 6 is
207                -- with in the threshold, then
208                 s_pixel_array_data(6) <= '1'; -- Set bit 6 in
209                 -- the output array to 1
210             else -- Otherwise,
```

```
211                     s_pixel_array_data(6) <= '0'; -- Clear bit 6
212                 end if;
213                 if(s_pixel_7 > s_negative_threshold and s_pixel_7
214                    < s_positive_threshold) then -- If pixel 7 is
215                    -- with in the threshold, then
216                     s_pixel_array_data(7) <= '1'; -- Set bit 7 in
217                     -- the output array to 1
218                 else -- Otherwise,
219                     s_pixel_array_data(7) <= '0'; -- Clear bit 7
220                 end if;
221                 if(s_pixel_8 > s_negative_threshold and s_pixel_8
222                    < s_positive_threshold) then -- If pixel 8 is
223                    -- with in the threshold, then
224                     s_pixel_array_data(8) <= '1'; -- Set bit 8 in
225                     -- the output array to 1
226                 else -- Otherwise,
227                     s_pixel_array_data(8) <= '0'; -- Clear bit 8
228                 end if;
229                 if(s_pixel_9 > s_negative_threshold and s_pixel_9
230                    < s_positive_threshold) then -- If pixel 9 is
231                    -- with in the threshold, then
232                     s_pixel_array_data(9) <= '1'; -- Set bit 9 in
233                     -- the output array to 1
234                 else -- Otherwise,
235                     s_pixel_array_data(9) <= '0'; -- Clear bit 9
236                 end if;
237             end if;
238         end process;
239
240     -- Assign the pixel array data signal to the output
241     pixel_array_data_out <= s_pixel_array_data;
242
243 end architecture;
```

```
1  ----------------------------------------------------------------------
2  --
3  --! @file        mod2_tap_in.vhd
4  --! @brief       Convert the tap data into pixel location values.
5  --! @details     Module 2 converst the incoming pixel data array
6  --               into start and end values within the 10 taps. The
7  --               start and end data form a sub-segment.
8  --
9  --! @author      Connor Dack
10 --! @date        November 2016
11 --! @copyright   Copyright (C) 2015 Ross K. Snider and Connor Dack
12 --
13 --   This program is free software: you can redistribute it
14 --   and/or modify it under the terms of the GNU General
```

```
15 --   Public License as published by the Free Software Foundation,
16 --   either version 3 of the License, or (at your option) any
17 --   later version.
18 --
19 --   This program is distributed in the hope that it will be
20 --   useful, but WITHOUT ANY WARRANTY; without even the implied
21 --   warranty of MERCHANTABILITY or FITNESS for A PARTICULAR
22 --   PURPOSE. See the GNU General Public License for more details.
23 --
24 --   You should have received a copy of the GNU General Public
25 --   License along with this program.  If not, see
26 --   <http://www.gnu.org/licenses/>.
27 --
28 --   Connor Dack
29 --   Electrical and Computer Engineering
30 --   Montana State University
31 --   610 Cobleigh Hall
32 --   Bozeman, MT 59717
33 --   connor.dack@msu.montana.edu
34 --
35 ----------------------------------------------------------------------------
36
37 library IEEE;
38 use IEEE.STD_LOGIC_1164.all;
39 use IEEE.NUMERIC_STD.all;
40 use IEEE.math_real.all;
41
42 use work.USER_CONTROLS.all;
43
44 ----------------------------------------------------------------------------
45 --
46 --! @brief       mod2_tap_in
47 --! @details     Module 2 converst the incoming pixel data array
48 --               into start and end values within the 10 taps. The
49 --               start and end data form a sub-segment.
50 --!
51 --! @param       clk                          - System Clock
52 --! @param       rst_n                        - Active low reset
53 --!                                             signal from HPS
54 --! @param       transciever_fifo_empty_in    - '1' represents empty
55 --! @param       air_jet_fifo_empty_in        - '1' represents empty
56 --! @param       enable_in                    - Indicate that 10-bit
57 --!     tap_data contains valid information. 0/1 = inVALID/VALID
58 --! @param       tap_data_in                  - The 10-pixel
59 --!     information represented as a 10-bit vector. MSB=0, LSB=9
60 --! @param       ss1_start_out                - Starting index of
61 --!                                 Subsegment 1. RANGE: 0-9, 15=UNUSED
62 --! @param       ss1_end_out                  - Ending index of
```

```
63  --!                                 Subsegment 1. RANGE: 0-9, 15=UNUSED
64  --! @param       ss2_start_out                - Starting index of
65  --!                                 Subsegment 2. RANGE: 0-9, 15=UNUSED
66  --! @param       ss2_end_out                  - Ending index of
67  --!                                 Subsegment 2. RANGE: 0-9, 15=UNUSED
68  --! @param       ss_ptr_out                   - Amount of Subsegments
69  --!                AND the current Subsegment #. RANGE: 1-2, 0=NONE
70  --! @param       ss_ready_out               - Data is Valid When '1'
71  --! @param       line_number_out            - Current line number
72  --! @param       line_rst_out               - Line reset flag
73  --! @param       end_of_line_out            - End of line flag
74  --
75  ----------------------------------------------------------------------------
76
77  entity mod2_tap_in is
78      port(clk                          :  in std_logic;
79           rst_n                        :  in std_logic;
80           enable_in                    :  in std_logic;
81           tap_data_in                  :  in std_logic_vector(0 to
82                                                 NUMBER_OF_TAPS - 1);
83           transciever_fifo_empty_in    :  in std_logic;
84           air_jet_fifo_empty_in        :  in std_logic;
85           ss1_start_out                : out std_logic_vector(3
86                                                              downto 0);
87           ss1_end_out                  : out std_logic_vector(3
88                                                              downto 0);
89           ss2_start_out                : out std_logic_vector(3
90                                                              downto 0);
91           ss2_end_out                  : out std_logic_vector(3
92                                                              downto 0);
93           ss_ptr_out                   : out std_logic_vector(1
94                                                              downto 0);
95           ss_ready_out                 : out std_logic;
96           line_number_out              : out std_logic_vector(
97                                 LINE_NUMBER_ADDRESS_SIZE - 1 downto 0);
98           line_rst_out                 : out std_logic;
99           end_of_line_out              : out std_logic);
100 end entity;
101
102 architecture behaviour of mod2_tap_in is
103
104     constant c_line_counter_max : unsigned(
105               LINE_NUMBER_ADDRESS_SIZE - 1 downto 0) := (others
106                    => '1'); -- the maximum possible line counter
107
108     -- Inner Component Signals
109     signal s_current_line      : unsigned(LINE_NUMBER_ADDRESS_SIZE
110                                  - 1 downto 0) := (others => '0');
```

```
111     signal s_pixel_counter       : unsigned(integer(trunc(log2(
112                                  real(TAPS_in_AOI)))) + 1 downto 0)
113                                  := (others => '0');
114     signal s_line_empty_counter : integer range 0 to 8 := 0;
115     -- A counter to make sure a complete hyperspectral line has
116     -- passed and there is no data
117     signal s_object_found_flag  : std_logic := '0';
118   -- Set if an object is found, used in the line reset conditions
119     signal s_line_rst_flag       : std_logic := '0';      --
120     signal s_transciever_fifo_empty : std_logic := '0';
121     -- Transciever FIFO Empty Flag, checked in the
122     -- line reset conditions
123     signal s_air_jet_fifo_empty : std_logic := '0';
124   -- Air Jet FIFO Empty Flag, checked in the line reset conditions
125     signal s_tap_data: std_logic_vector(0 to NUMBER_OF_TAPS - 1);
126
127     -- Output Signals
128     signal ss1_start_sig         : std_logic_vector( 3 downto 0)
129                                                        := "1111";
130         -- Starting index of Subsegment 1.  RANGE: 0-9, 15=UNUSED
131     signal ss1_end_sig           : std_logic_vector( 3 downto 0)
132                                                        := "1111";
133           -- Ending index of Subsegment 1. RANGE: 0-9, 15=UNUSED
134     signal ss2_start_sig         : std_logic_vector( 3 downto 0)
135                                                        := "1111";
136         -- Starting index of Subsegment 2. RANGE: 0-9, 15=UNUSED
137     signal ss2_end_sig           : std_logic_vector( 3 downto 0)
138                                                        := "1111";
139         -- Ending   index of Subsegment 2. RANGE: 0-9, 15=UNUSED
140     signal ss_ptr_sig            : std_logic_vector( 1 downto 0)
141                   :=   "00";  -- Amount of Subsegments AND the
142                       -- current Subsegment #. RANGE: 1-2, 0=NONE
143     signal s_ss_ready            : std_logic := '0';
144     -- Subsegment ready flag, used in Module 3
145     signal s_line_counter      : unsigned(LINE_NUMBER_ADDRESS_SIZE
146                 - 1 downto 0) := (others => '0'); -- Line counter
147     signal s_line_rst            : std_logic := '0';
148     -- Line reset flag - sent to the hyperspectral system as well
149     signal s_end_of_line         : std_logic := '0';
150     -- End of line flag, triggers later modules
151     signal s_enable              : std_logic := '0';
152     -- Component enable signal
153
154 begin
155      s_transciever_fifo_empty <= transciever_fifo_empty_in;
156     -- Assign the input signals to the internal component signals
157      s_air_jet_fifo_empty     <= air_jet_fifo_empty_in;
158     -- Assign the input signals to the internal component signals
```

```
159      s_enable                <= enable_in;
160    -- Assign the input signals to the internal component signals
161      s_tap_data              <= tap_data_in;
162    -- Assign the input signals to the internal component signals
163
164    -- The process to determine the subsegments from the tap data
165    find_ss     : process(clk, rst_n)
166        variable ss1_start_var : std_logic_vector(3 downto 0)
167                    := "1111"; -- Starting index of Subsegment 1.
168                               -- RANGE: 0-9, 15=UNUSED
169        variable ss1_end_var   : std_logic_vector(3 downto 0)
170                    := "1111"; -- Ending   index of Subsegment 1.
171                               -- RANGE: 0-9, 15=UNUSED
172        variable ss2_start_var : std_logic_vector(3 downto 0)
173                    := "1111"; -- Starting index of Subsegment 2.
174                               -- RANGE: 0-9, 15=UNUSED
175        variable ss2_end_var   : std_logic_vector(3 downto 0)
176                    := "1111"; -- Ending   index of Subsegment 2.
177                               -- RANGE: 0-9, 15=UNUSED
178        variable ss_ptr_var    : std_logic_vector(1 downto 0)
179                    :=   "00"; -- Amount of Subsegments AND the
180                              -- current Subsegment #. RANGE: 1-2, 0=NONE
181        variable last_bit_var  : std_logic := '0'; -- Flag set if
182                                 -- the last value in the tap is a '1'
183    begin
184        if (rst_n = '0') then -- Reset case - set the variables
185  -- to their inital values and assign them to the correct signals
186            ss1_start_var   := "1111";
187            -- Reset the sub-segment 1 start variable
188            ss1_end_var     := "1111";
189            -- Reset the sub-segment 1 end variable
190            ss2_start_var   := "1111";
191            -- Reset the sub-segment 2 start variable
192            ss2_end_var     := "1111";
193            -- Reset the sub-segment 2 end variable
194            ss_ptr_var      :=   "00";
195            -- Clear the sub-segment pointer
196            last_bit_var    :=    '0';
197            -- Clear the last bit variable flag
198            s_ss_ready      <=    '0';
199            -- Clear the sub-segment ready flag
200            ss1_start_sig   <= ss1_start_var; -- Assign the sub-
201    -- segment 1 start variable to the sub-segment 1 start signal
202            ss1_end_sig     <= ss1_end_var;   -- Assign the sub-
203    -- segment 1 end variable to the sub-segment 1 end signal
204            ss2_start_sig   <= ss2_start_var; -- Assign the sub-
205    -- segment 2 start variable to the sub-segment 2 start signal
206            ss2_end_sig     <= ss2_end_var;   -- Assign the sub-
```

```
207     -- segment 2 end variable to the sub-segment 2 end signal
208             ss_ptr_sig      <= ss_ptr_var;    -- Assign the sub-
209     -- segment pointer variable to the sub-segment pointer signal
210       elsif (rising_edge(clk)) then
211       -- Trigger on the rising  edge of the system clock
212           if (s_enable = '1') then -- if the system is in
213             -- enabled, set the variables to their inital value
214               ss1_start_var   := "1111";
215               -- Reset the sub-segment 1 start variable
216               ss1_end_var     := "1111";
217               -- Reset the sub-segment 1 end variable
218               ss2_start_var   := "1111";
219               -- Reset the sub-segment 2 start variable
220               ss2_end_var     := "1111";
221               -- Reset the sub-segment 2 end variable
222               ss_ptr_var      :=   "00";
223               -- Clear the sub-segment pointer
224               last_bit_var    :=    '0';
225               -- Clear the last bitvariable flag
226               for i in 0 to NUMBER_OF_TAPS - 1 loop
227               -- Run the following loop for the number of taps
228                   if (s_tap_data(i) = '1') then
229                   -- If the tap data is a '1', then
230                       if (last_bit_var = '1') then
231                       -- If the previous tap data is a '1' then
232                           -- Append to existing Subsegment
233                           if (ss_ptr_var = "01") then
234                       -- If the sub-segment pointer is "01" then
235                               ss1_end_var := std_logic_vector(
236                               UNSIGNED(ss1_end_var) + "0001");
237                    -- Increment the sub-segment 1 end variable by 1
238                               s_object_found_flag <= '1';
239                               -- Set the object found flag
240                           elsif (ss_ptr_var = "10") then
241                       -- If the sub-segment pointer is "10" then
242                               ss2_end_var := std_logic_vector(
243                               UNSIGNED(ss2_end_var) + "0001");
244                    -- Increment the sub-segment 2 end variable by 1
245                               s_object_found_flag <= '1';
246                               -- Set the object found flag
247                           end if;
248                       else -- If the tap data is a '1' and the
249                       -- previous tap data is a '0' then
250                           -- Create a new Subsegment
251                           if(ss_ptr_var = "00") then
252                           -- Create Subsegment #1
253                               ss_ptr_var    := "01"; -- Set the
254                                  -- sub-segment pointer is "01"
```

```
255                                 ss1_start_var:= std_logic_vector(
256                                 to_unsigned(i, 4)); -- Set the
257   -- sub-segment 1 start variable to i - location in the for loop
258                                 ss1_end_var  := std_logic_vector(
259                                 to_unsigned(i, 4)); -- Set the
260                                -- sub-segment 1 end variable to i
261                                 s_object_found_flag <= '1';
262                                 -- Set the object found flag
263                               elsif (ss_ptr_var = "01") then
264                               -- Create Subsegment #2
265                                 ss_ptr_var   := "10"; -- Set the
266                                 -- sub-segment pointer to "10"
267                                 ss2_start_var:= std_logic_vector(
268                                 to_unsigned(i, 4)); -- Set the
269                                -- sub-segment 2 start variable to i
270                                 ss2_end_var  := std_logic_vector(
271                                 to_unsigned(i, 4)); -- Set the
272                                -- sub-segment 2 end variable to i
273                                 s_object_found_flag <= '1';
274                                 -- Set the object found flag
275                               end if;
276                             end if;
277                           elsif(s_end_of_line = '1') then
278         -- If it is the end of the line and there is no tap data,
279                             s_object_found_flag <= '0'; -- Clear the
280                                                 -- object found flag
281                           end if;
282                           last_bit_var := s_tap_data(i); -- Update the
283                                                 -- last bit variable
284                       end loop;
285           -- When the loop is done, set the subsegment ready flag
286                       s_ss_ready <= '1';
287                     else -- If the system is not enabled, reset all of
288                          -- the variables and signals
289                       ss1_start_var       := "1111"; -- Reset the sub-
290                                              -- segment 1 start variable
291                       ss1_end_var         := "1111"; -- Reset the sub-
292                                              -- segment 1 end variable
293                       ss2_start_var       := "1111"; -- Reset the sub-
294                                              -- segment 2 start variable
295                       ss2_end_var         := "1111"; -- Reset the sub-
296                                              -- segment 2 end variable
297                       ss_ptr_var          :=   "00"; -- Clear the sub-
298                                              -- segment pointer
299                       last_bit_var        :=     '0'; -- Clear the last
300                                                -- bit variable flag
301                       s_ss_ready          <=     '0'; -- Clear the sub-
302                                              -- segment ready flag
```

```
303                 s_object_found_flag <=     '0'; -- Clear the
304                                             -- object found flag
305             end if; -- Assign the variables to the output signals
306             ss1_start_sig   <= ss1_start_var; -- Assign the sub-
307     -- segment 1 start variable to the sub-segment 1 start signal
308             ss1_end_sig     <= ss1_end_var;   -- Assign the sub-
309     -- segment 1 end variable to the sub-segment 1 end signal
310             ss2_start_sig   <= ss2_start_var; -- Assign the sub-
311     -- segment 2 start variable to the sub-segment 2 start signal
312             ss2_end_sig     <= ss2_end_var;   -- Assign the sub-
313     -- segment 2 end variable to the sub-segment 2 end signal
314             ss_ptr_sig      <= ss_ptr_var;    -- Assign the sub-
315     -- segment pointer variable to the sub-segment pointer signal
316         end if;
317     end process;
318
319 -- Process to count the line numbers and to reset when necessary
320     line_counter_proc : process (clk, rst_n)
321         variable v_line_empty_counter : integer range 0 to 8:= 0;
322         -- Variable to count the number of empty line scan lines
323         variable v_line_rst : std_logic := '0'; -- Variable for
324                                                 -- the line reset flag
325     begin
326         if(rst_n = '0') then -- On reset, clear register
327             s_end_of_line  <=  '0'; -- clear the end of line flag
328             s_line_counter <= (others => '0');
329             -- reset the line counter
330         elsif (rising_edge(clk)) then
331         -- Trigger on the rising edge of the system clock
332             if(s_ss_ready = '1') then
333             -- If the sub-segment are ready, then
334                 if(s_pixel_counter = TAPS_in_AOI - 1) then
335         -- If the pixel counter reaches the end of the line, then
336                     if(s_object_found_flag = '1' or v_line_rst =
337                             '1' or (v_line_empty_counter =
338                             HYPER_TO_LINE_LINE_RATIO and
339                             s_transciever_fifo_empty = '0'
340                             and s_air_jet_fifo_empty = '0')) then
341                 -- If there is an object somewhere in the system,
342                         v_line_empty_counter := 0;
343                         -- reset the line empty counter
344                     else
345                         v_line_empty_counter :=
346                                         v_line_empty_counter + 1;
347                         -- Increment the line empty counter by 1
348                     end if;
349                     if(s_line_counter = c_line_counter_max) then
350                   -- If the line counter is at the maximum value,
```

```
351                     s_line_counter <= (others => '0');
352                     -- Reset the line counter to 0
353                     s_pixel_counter <= (others => '0');
354                     -- Reset the pixel counter
355                     v_line_rst   := '1';
356                     -- Set the line reset flag variable
357                     s_line_rst   <= '1';
358                     -- Set the line reset flag signal
359                 elsif(v_line_empty_counter =
360                       HYPER_TO_LINE_LINE_RATIO and
361             -- If there have been 8 empty line scan lines and
362                       s_transciever_fifo_empty = '1' and
363                       s_air_jet_fifo_empty = '1') then
364                       -- both fifos are empty, then
365                     s_line_counter <= (others => '0');
366                     -- Reset the line counter to 0
367                     s_pixel_counter <= (others => '0');
368                     -- Reset the pixel counter
369                     v_line_rst   := '1';
370                     -- Set the line reset flag variable
371                     s_line_rst   <= '1';
372                     -- Set the line reset flag signal
373                 else -- Otherwise
374                     s_line_counter  <= s_line_counter + 1;
375                     -- Increment the line counter by 1
376                     s_pixel_counter <= (others => '0');
377                     -- Reset the pixel counter
378                     s_line_rst    <= '0';
379                     -- Clear the line reset flag signal
380                     v_line_rst    := '0';
381                     -- Clear the line reset flag variable
382                 end if;
383                 if(s_line_rst = '1') then
384                 -- If the line reset flag signal is set,
385                     s_line_rst    <= '0';
386                     -- Clear the line reset flag signal
387                     s_end_of_line <=  '0';
388                     -- Clear the end of line flag
389                 end if;
390                 s_end_of_line <=  '1';
391                 -- Set the end of line flag
392             else -- Otherwise
393                 s_end_of_line <= '0';
394                 -- Clear the end of line flag
395                 s_line_rst    <= '0';
396                 -- Clear the line reset flag signal
397                 s_pixel_counter <= s_pixel_counter + 1;
398                 -- Increment the pixel counter by 1
```

```
399                 end if;
400             end if;
401         end if;
402     end process;
403
404 -- Assign all of the internal signals to the corresponding output
405     ss1_start_out   <= ss1_start_sig; -- Assign the sub-segment 1
406                                       -- start signal to the output
407     ss1_end_out     <= ss1_end_sig;   -- Assign the sub-segment 1
408                                       -- end signal to the output
409     ss2_start_out   <= ss2_start_sig; -- Assign the sub-segment 2
410                                       -- start signal to the output
411     ss2_end_out     <= ss2_end_sig;   -- Assign the sub-segment 2
412                                       -- end signal to the output
413     ss_ptr_out      <= ss_ptr_sig;    -- Assign the sub-segment
414                                    -- pointer signal to the output
415     ss_ready_out    <= s_ss_ready;    -- Assign the sub-segment
416                                       -- ready signal to the output
417     line_rst_out    <= s_line_rst;    -- Assign the line number
418                                       -- reset flag to the output
419     line_number_out <= std_logic_vector(s_line_counter);
420                          -- Assign the line counter to the output
421     end_of_line_out <= s_end_of_line; -- Assign the end of line
422                                       -- signal to the output
423
424 end architecture;
```

```
1  --------------------------------------------------------------------------------
2  --
3  --! @file        mod3_ss2seg.vhd
4  --! @brief       Module 3 sub-segment to segment conversion.
5  --! @details
6  --
7  --! @author      Connor Dack
8  --! @date        May 2015
9  --! @copyright   Copyright (C) 2015 Ross K. Snider and Connor Dack
10 --
11 --   This program is free software: you can redistribute it
12 --   and/or modify it under the terms of the GNU General
13 --   Public License as published by the Free Software Foundation,
14 --   either version 3 of the License, or (at your option) any
15 --   later version.
16 --
17 --   This program is distributed in the hope that it will be
18 --   useful, but WITHOUT ANY WARRANTY; without even the implied
19 --   warranty of MERCHANTABILITY or FITNESS for A PARTICULAR
20 --   PURPOSE. See the GNU General Public License for more details.
21 --
```

```
22 --   You should have received a copy of the GNU General Public
23 --   License along with this program.  If not, see
24 --   <http://www.gnu.org/licenses/>.
25 --
26 --   Connor Dack
27 --   Electrical and Computer Engineering
28 --   Montana State University
29 --   610 Cobleigh Hall
30 --   Bozeman, MT 59717
31 --   connor.dack@msu.montana.edu
32 --
33 ----------------------------------------------------------------------------
34 library IEEE;
35 use IEEE.STD_LOGIC_1164.all;
36 use IEEE.NUMERIC_STD.all;
37
38 use work.USER_CONTROLS.all;
39
40 ----------------------------------------------------------------------------
41 --
42 --! @brief       mod3_ss2seg
43 --! @details
44 --!
45 --! @param       clk             - Input: System Clock
46 --! @param       rst_n           - Input: Active low reset signal
47 --! @param       ss_ready_in     - Input: Sub-sgement ready flag
48 --! @param       ss_ptr_in       - Input: sub-segment pointer flag
49 --! @param       ss1_start_in    - Input: sub-segment 1 start value
50 --! @param       ss1_end_in      - Input: sub-segment 1 end value
51 --! @param       ss2_start_in    - Input: sub-segment 2 start value
52 --! @param       ss2_end_in      - Input: sub-segment 2 end value
53 --! @param       line_number_in  - Input: The current line number
54 --! @param       line_rst_in     - Input: The line scan line reset
55 --!                                       flag
56 --! @param       end_of_line_in  - Input: End of line flag
57 --! @param       seg_start_out   - Output: Array of start values
58 --!                               of the objects on the current line
59 --! @param       seg_mid_out     - Output: Array of middle values
60 --!                               of the objects on the current line
61 --! @param       seg_end_out     - Output: Array of end values of
62 --!                                 the objects on the current line
63 --! @param       seg_ptr_out     - Output:
64 --
65 ----------------------------------------------------------------------------
66
67 entity mod3_ss2seg is
68     port(clk             :  in std_logic;
69          rst_n           :  in std_logic;
```

```
70          ss_ready_in     :  in std_logic;
71          ss_ptr_in       :  in std_logic_vector(1 downto 0);
72          ss1_start_in    :  in std_logic_vector(3 downto 0);
73          ss1_end_in      :  in std_logic_vector(3 downto 0);
74          ss2_start_in    :  in std_logic_vector(3 downto 0);
75          ss2_end_in      :  in std_logic_vector(3 downto 0);
76          line_number_in  :  in std_logic_vector(
77                                      LINE_NUMBER_ADDRESS_SIZE
78                                      - 1 downto 0);
79          line_rst_in     :  in std_logic;
80          end_of_line_in  :  in std_logic;
81          seg_start_out   : out seg_info_array;
82          seg_mid_out     : out seg_info_array;
83          seg_end_out     : out seg_info_array;
84          seg_ptr_out     : out std_logic_vector(9 downto 0));
85  end entity;
86
87  architecture rtl of mod3_ss2seg is
88
89      component mod3_seg_info is
90          port(clk            :  in std_logic;
91               rst_n          :  in std_logic;
92               end_of_line_in :  in std_logic;
93               tap_counter_in :  in unsigned(AOI_ADDRESS_SIZE - 1
94                                          downto 0);
95               ss1_active_in  :  in std_logic;
96               ss1_start_in   :  in std_logic_vector(3 downto 0);
97               ss1_end_in     :  in std_logic_vector(3 downto 0);
98               ss2_active_in  :  in std_logic;
99               ss2_start_in   :  in std_logic_vector(3 downto 0);
100              ss2_end_in     :  in std_logic_vector(3 downto 0);
101              seg_start_out  : out std_logic_vector(
102                                 AOI_ADDRESS_SIZE - 1  downto 0);
103              seg_mid_out    : out std_logic_vector(
104                                 AOI_ADDRESS_SIZE - 1  downto 0);
105              seg_end_out    : out std_logic_vector(
106                                 AOI_ADDRESS_SIZE - 1  downto 0));
107     end component;
108
109     component unit_delay_4bit is
110         port(clk        :  in std_logic;
111              rst_n      :  in std_logic;
112              clk_enable:  in std_logic;
113              to_delay   :  in std_logic_vector(3 downto 0);
114              ce_out     : out std_logic;
115              out1       : out std_logic_vector(3 downto 0));
116     end component;
117
```

```
118     signal s_ss_ready     : std_logic;
119     signal s_ss_ptr       : std_logic_vector(1 downto 0);
120     signal s_ss1_start    : std_logic_vector(3 downto 0);
121     signal s_ss1_end      : std_logic_vector(3 downto 0);
122     signal s_ss2_start    : std_logic_vector(3 downto 0);
123     signal s_ss2_end      : std_logic_vector(3 downto 0);
124     signal s_line_number  : std_logic_vector(
125                             LINE_NUMBER_ADDRESS_SIZE - 1 downto 0);
126     signal s_line_rst     : std_logic;
127     signal s_end_of_line  : std_logic;
128     signal s_last_bit     : std_logic := '0';
129     -- Saves the value of left-most pixel from previous tap
130     signal s_ss1_route    : std_logic_vector(9 downto 0)
131                                           := "0000000000";
132     -- Counts the Active Segments
133     signal s_ss2_route    : std_logic_vector(9 downto 0)
134                                           := "0000000000";
135     -- Counts the Active Segments
136     signal s_seg_ptr      : std_logic_vector(9 downto 0)
137                                           := "0000000000";
138     -- Counts the Active Segments
139     signal ls_ptr         : unsigned(AOI_ADDRESS_SIZE -1 downto 0)
140                                     := (others => '0');
141     signal s_tap_counter : integer range 0 to AOI_LESS_10 := 0;
142     -- Integer tap counter - used to know the pixel location
143     -- on the line
144     signal s_ss1_active   : ss_inSeg_array := (others => '0');
145     -- Sub-sgement 1 active array
146     signal s_ss2_active   : ss_inSeg_array := (others => '0');
147     -- Sub-segment 2 active array
148     signal s_seg_start    : seg_info_array := (others =>
149                                           (others => '0'));
150     -- Array of start values of the objects on the current line
151     signal s_seg_end      : seg_info_array := (others =>
152                                           (others => '0'));
153     -- Array of middle values of the objects on the current line
154     signal s_seg_mid      : seg_info_array := (others =>
155                                           (others => '0'));
156     -- Array of end values of the objects on the current line
157
158     signal ss1_start_delayed, ss1_end_delayed,
159            -- The values of Subsegment 1x clock cycle after it
160            -- has been provided (comes in 1x too early)
161            ss2_start_delayed, ss2_end_delayed :
162                                     std_logic_vector(3 downto 0);
163
164 begin
165     -- Assign the input signals to the internal component signals
```

```
166     s_ss_ready     <= ss_ready_in;  -- Sub-segment ready flag
167     s_ss_ptr       <= ss_ptr_in;    -- Sub-segment pointer signal
168     s_ss1_start    <= ss1_start_in; -- Sub-segment one start value
169     s_ss1_end      <= ss1_end_in;   -- Sub-segment one end value
170     s_ss2_start    <= ss2_start_in; -- Sub-segment two start value
171     s_ss2_end      <= ss2_end_in;   -- Sub-segment two end value
172     s_line_number <= line_number_in; -- The current line number
173     s_line_rst     <= line_rst_in;    -- The line reset flag
174     s_end_of_line <= end_of_line_in; -- The end of line flag
175
176   -- ss2seg : The control process to covert the sub-segments on
177   -- the line to segments on the line
178     ss2seg : process(clk, rst_n)
179         variable v_first_bit : std_logic;
180         -- Variable to save the first bit of each sub-segment
181     begin
182         if (rst_n = '0') then -- Active low reset
183             s_ss1_route   <= "0000000000";
184             -- Clear the sub-segment one route pointer
185             s_ss2_route   <= "0000000000";
186             -- Clear the sub-segment two route pointer
187             s_seg_ptr     <= "0000000000";
188             -- Clear the segment pointer
189             s_last_bit    <= '0';
190             -- Clear the last bit of the tap register
191             s_tap_counter <=  0 ;
192             -- Clear the tap counter
193         elsif(rising_edge(clk)) then
194         -- Trigger on the rising edge of the system clock
195             if (s_ss_ready = '1') then
196             -- If the sub-segment ready flag is set, then
197                 if (s_ss1_start = "0000") then
198                 -- If the sub-segment one start value is 0, then
199                     v_first_bit := '1';
200                     -- set the first bit variable to '1'
201                 else -- otherwise,
202                     v_first_bit := '0';
203                     -- set the first bit variable to '0'
204                 end if;
205                 if(s_end_of_line = '1') then
206                 -- If the end of line flag is set, then
207                     if((s_ss_ptr = "01" or s_ss_ptr = "10") and
208                         s_seg_ptr < "1111111111") then
209                         -- If either segment is active and the
210                         -- segment pointer is not at the maximum,
211                         -- then
212                         if (s_last_bit = '1' and
213                             v_first_bit = '1') then
```

```
214                 -- If the last bit is a '1' and the
215                 --first bit is a '1', then
216                 s_ss1_route <= "0000000001";
217                 -- Route sub-segment into current
218                 -- Segment
219                 if(s_seg_ptr = "1111111110") then
220                 -- If the segment pointer is about
221                 -- to roll over, then
222                     if(s_ss1_end /= "1001") then
223                     -- If the sub-segment 1 end
224                     -- value is 9, then
225                         s_seg_ptr <= "1111111111";
226                         -- increment the segment
227                         -- pointer to the max value -
228                         -- this is the end
229                     else -- Otherwise
230                         s_seg_ptr <= "0000000001";
231                       -- Set the segment pointer to 1
232                     end if;
233                     s_ss2_route <= "0000000000";
234                     -- Clear the sub-segment 2 route
235                     -- signal
236                 else -- Otherwise
237                     if(s_ss_ptr = "10") then
238                     -- if the sub-semgent pointer is
239                     -- 2, then
240                         s_ss2_route <= "0000000010";
241                         -- Route sub-semgnet into the
242                         -- next segment
243                         s_seg_ptr   <= "0000000010";
244                         -- Set the segment pointer
245                         -- to 2
246                     else -- Otherwise
247                         s_ss2_route <= "0000000000";
248                         -- Clear the sub-segment 2
249                         -- route signal
250                     end if;
251                 end if;
252             else -- Otherwise
253                 if(s_seg_ptr = "1111111110") then
254                 -- If seg_ptr = "1111111110" and
255                 -- no rollover, then
256                     s_seg_ptr   <= "1111111111";
257                    -- Set the segment pointer to
258                    -- the end condition
259                     s_ss1_route <= "0000000000";
260                    -- Clear the sub-segment 1
261                    -- route pointer
```

```
262                         s_ss2_route <= "0000000000";
263                         -- Clear the sub-segment 2
264                         -- route pointer
265                     else -- Otherwise
266                         s_ss1_route <= "0000000001";
267                         -- Set the sub-segment route
268                         -- signal to 1
269                         s_seg_ptr   <= "0000000001";
270                         -- Set the segment pointer to 1
271                         if(s_ss_ptr = "10") then
272                         -- If the sub-segment pointer is
273                         -- 2, then
274                             s_ss2_route <= "0000000010";
275                             -- Route sub-segment into
276                             -- then next segment
277                             -- Segment 2
278                             s_seg_ptr   <= "0000000010";
279                             -- Set the segment pointer
280                             -- to 2
281                         else -- Otherwise
282                             s_ss2_route <= "0000000000";
283                             -- Clear the sub-segment 2
284                             -- route signal
285                         end if;
286                     end if;
287                 end if;
288             else -- Otherwise
289                 s_ss1_route <= "0000000000";
290                 -- Clear the sub-segment 1 route signal
291                 s_ss2_route <= "0000000000";
292                 -- Clear the sub-segment 2 route sginal
293                 s_seg_ptr   <= (others => '0');
294                 -- Register the current state of the
295                 -- segment pointer
296             end if;
297         else --
298             if((s_ss_ptr = "01" or s_ss_ptr = "10") and
299                 s_seg_ptr < "1111111111") then
300                 if(s_last_bit = '1' and
301                    v_first_bit = '1') then
302                     s_ss1_route <= s_seg_ptr;
303                     -- Route SS into current Segment
304                     if(s_seg_ptr = "1111111110") then
305                         if(s_ss1_end /= "1001") then
306                             s_seg_ptr <= "1111111111";
307                         else
308                             s_seg_ptr <= s_seg_ptr;
309                         end if;
```

```
310                         s_ss2_route <= "0000000000";
311                     else
312                         if(s_ss_ptr = "10") then
313                             s_ss2_route <=
314                                     std_logic_vector(
315                                     unsigned(s_seg_ptr)
316                                     + "0000000001");
317                             -- Route SS into NEXT Segment
318                             s_seg_ptr   <=
319                                      std_logic_vector(
320                                      unsigned(s_seg_ptr)
321                                      + "0000000001");
322                         else
323                             s_ss2_route <= "0000000000";
324                         end if;
325                     end if;
326                 else
327                     if(s_seg_ptr = "1111111110") then
328                         s_seg_ptr   <= "1111111111";
329                         s_ss1_route <= "0000000000";
330                         s_ss2_route <= "0000000000";
331                     else
332                         s_ss1_route <= std_logic_vector(
333                                        unsigned(s_seg_ptr)
334                                        + "0000000001");
335                         -- Route SS into NEXT Segment
336                         s_seg_ptr   <= std_logic_vector(
337                                        unsigned(s_seg_ptr)
338                                        + "0000000001");
339                         if(s_ss_ptr = "10") then
340                             s_ss2_route <=
341                                     std_logic_vector(
342                                     unsigned(s_seg_ptr)
343                                     + "0000000010");
344                             -- Route SS into NEXT Segment
345                             s_seg_ptr   <=
346                                     std_logic_vector(
347                                     unsigned(s_seg_ptr)
348                                     + "0000000010");
349                         else
350                             s_ss2_route <= "0000000000";
351                         end if;
352                     end if;
353                 end if;
354             else
355                 s_ss1_route <= "0000000000";
356                 s_ss2_route <= "0000000000";
357                 s_seg_ptr   <= s_seg_ptr;
```

```
358                     end if;
359                 end if;
360                 if(s_ss1_end = "1001" or s_ss2_end = "1001") then
361                     s_last_bit <= '1';
362                 else
363                     s_last_bit <= '0';
364                 end if;
365                 if (s_tap_counter = AOI_LESS_10) then
366                     s_tap_counter <= 0;
367                 else
368                     s_tap_counter <= s_tap_counter + 10;
369                 end if;
370             else
371                 s_ss1_route   <= "0000000000";
372                 s_ss2_route   <= "0000000000";
373                 s_seg_ptr     <= "0000000000";
374                 s_last_bit    <= '0';
375                 s_tap_counter <=  0 ;
376             end if;
377         end if;
378     end process;
379
380
381     ss_active_logic : process(s_ss1_route, s_ss2_route)
382         begin
383             for i in 1 to TAPS_IN_AOI loop
384                 if(i = to_integer(unsigned(s_ss1_route))) then
385                     s_ss1_active(i) <= '1';
386                 else
387                     s_ss1_active(i) <= '0';
388                 end if;
389
390                 if(i = to_integer(unsigned(s_ss2_route))) then
391                     s_ss2_active(i) <= '1';
392                 else
393                     s_ss2_active(i) <= '0';
394                 end if;
395             end loop;
396         end process;
397
398     ls_ptr <= to_unsigned(s_tap_counter,AOI_ADDRESS_SIZE);
399
400     seg_generate : for i in 1 to TAPS_IN_AOI generate
401         seg : mod3_seg_info port map(
402             clk             => clk,
403             rst_n           => rst_n,
404             end_of_line_in  => s_end_of_line,
405             tap_counter_in  => ls_ptr,
```

```
406             ss1_active_in  => s_ss1_active(i),
407             ss1_start_in   => ss1_start_delayed(3 downto 0),
408             ss1_end_in     => ss1_end_delayed(3 downto 0),
409             ss2_active_in  => s_ss2_active(i),
410             ss2_start_in   => ss2_start_delayed(3 downto 0),
411             ss2_end_in     => ss2_end_delayed(3 downto 0),
412             seg_start_out  => s_seg_start(i),
413             seg_mid_out    => s_seg_mid(i),
414             seg_end_out    => s_seg_end(i));
415     end generate;
416
417
418     ss1_start_delay : unit_delay_4bit port map(
419             clk             => clk,
420             rst_n           => rst_n,
421             clk_enable      => '1',
422             to_delay        => s_ss1_start,
423             out1            => ss1_start_delayed
424     );
425
426     ss1_end_delay : unit_delay_4bit port map(
427             clk             => clk,
428             rst_n           => rst_n,
429             clk_enable      => '1',
430             to_delay        => s_ss1_end,
431             out1            => ss1_end_delayed
432     );
433
434     ss2_start_delay : unit_delay_4bit port map(
435             clk             => clk,
436             rst_n           => rst_n,
437             clk_enable      => '1',
438             to_delay        => s_ss2_start,
439             out1            => ss2_start_delayed
440     );
441
442     ss2_end_delay : unit_delay_4bit port map(
443             clk             => clk,
444             rst_n           => rst_n,
445             clk_enable      => '1',
446             to_delay        => s_ss2_end,
447             out1            => ss2_end_delayed
448     );
449
450     delay_segptr : process (clk, rst_n) -- 1x clock delay
451      begin
452         if(rst_n = '0') then
453             seg_ptr_out <= "0000000000";
```

```
454             elsif(rising_edge(clk)) then
455                 seg_ptr_out <= s_seg_ptr;
456             end if;
457         end process;
458 
459         seg_start_out <= s_seg_start;
460         seg_mid_out   <= s_seg_mid;
461         seg_end_out   <= s_seg_end;
462 
463 end architecture;
```

```
1  ----------------------------------------------------------------------------
2  --
3  --! @file        mod3_seg_info.vhd
4  --! @brief
5  --! @details
6  --
7  --! @author      Connor Dack
8  --! @date        May 2015
9  --! @copyright   Copyright (C) 2015 Ross K. Snider and Connor Dack
10 --
11 --   This program is free software: you can redistribute it
12 --   and/or modify it under the terms of the GNU General
13 --   Public License as published by the Free Software Foundation,
14 --   either version 3 of the License, or (at your option) any
15 --   later version.
16 --
17 --   This program is distributed in the hope that it will be
18 --   useful, but WITHOUT ANY WARRANTY; without even the implied
19 --   warranty of MERCHANTABILITY or FITNESS for A PARTICULAR
20 --   PURPOSE. See the GNU General Public License for more details.
21 --
22 --   You should have received a copy of the GNU General Public
23 --   License along with this program.  If not, see
24 --   <http://www.gnu.org/licenses/>.
25 --
26 --   Connor Dack
27 --   Electrical and Computer Engineering
28 --   Montana State University
29 --   610 Cobleigh Hall
30 --   Bozeman, MT 59717
31 --   connor.dack@msu.montana.edu
32 --
33 ----------------------------------------------------------------------------
34 
35 library IEEE;
36 use IEEE.STD_LOGIC_1164.all;
37 use IEEE.NUMERIC_STD.all;
```

```
38
39 use work.USER_CONTROLS.all;
40
41 --
   ------------------------------------------------------------------------------

42 --
43 --! @brief      mod3_seg_info
44 --! @details
45 --!
46 --! @param      clk             -  Input: System Clock
47 --! @param      rst_n           -  Input: Active low reset signal
48 --! @param      end_of_line_in  -  Input: End of Line flag
49 --! @param      tap_counter_in  -  Input: Tap Counter
50 --! @param      ss1_active_in   -  Input: Sub-segment 1 active flag
51 --! @param      ss1_start_in    -  Input: Sub-segment 1 start value
52 --! @param      ss1_end_in      -  Input: Sub-segment 1 end value
53 --! @param      ss2_active_in   -  Input: Sub-segment 2 active flag
54 --! @param      ss2_start_in    -  Input: Sub-segment 2 start value
55 --! @param      ss2_end_in      -  Input: Sub-segment 2 end value
56 --! @param      seg_start_out   - Output: Segment start value
57 --! @param      seg_mid_out     - Output: Segment middle value
58 --! @param      seg_end_out     - Output: Segment end value
59 --
60 ------------------------------------------------------------------------------
61 entity mod3_seg_info is
62     port(clk             :  in std_logic;
63          rst_n           :  in std_logic;
64          end_of_line_in  :  in std_logic;
65          tap_counter_in  :  in unsigned(AOI_ADDRESS_SIZE - 1
66                                                      downto 0);
67          ss1_active_in   :  in std_logic;
68          ss1_start_in    :  in std_logic_vector( 3 downto 0);
69          ss1_end_in      :  in std_logic_vector( 3 downto 0);
70          ss2_active_in   :  in std_logic;
71          ss2_start_in    :  in std_logic_vector( 3 downto 0);
72          ss2_end_in      :  in std_logic_vector( 3 downto 0);
73          seg_start_out   : out std_logic_vector(AOI_ADDRESS_SIZE -
74                                                  1 downto 0);
75          seg_mid_out     : out std_logic_vector(AOI_ADDRESS_SIZE -
76                                                  1 downto 0);
77          seg_end_out     : out std_logic_vector(AOI_ADDRESS_SIZE -
78                                                  1 downto 0));
79 end entity;
80
81 architecture rtl of mod3_seg_info is
82
83     -- Internal component signal decelarations
```

```
84      signal s_seg_active   : std_logic := '0';
85      -- Keeps track of Segment ON/OFF
86      signal s_end_of_line : std_logic :='0';  -- End of line flag
87      signal s_tap_counter:unsigned(AOI_ADDRESS_SIZE - 1 downto 0);
88      --
89      signal s_ss1_active   : std_logic;
90      -- Sub-segment 1 active flag
91      signal s_ss1_start    : std_logic_vector(3 downto 0);
92      -- Sub-segment 1 start value
93      signal s_ss1_end      : std_logic_vector(3 downto 0);
94      -- Sub-segment 1 end value
95      signal s_ss2_active   : std_logic;
96      -- Sub-segment 2 active flag
97      signal s_ss2_start    : std_logic_vector(3 downto 0);
98      -- Sub-segment 2 start value
99      signal s_ss2_end      : std_logic_vector(3 downto 0);
100     -- Sub-segment 2 end value
101     signal s_seg_start    : std_logic_vector(AOI_ADDRESS_SIZE - 1
102                                     downto 0) := (others => '0');
103     -- Segment start value
104     signal s_seg_mid      : std_logic_vector(AOI_ADDRESS_SIZE - 1
105                                     downto 0) := (others => '0');
106     -- Segment middle value
107     signal s_seg_end      : std_logic_vector(AOI_ADDRESS_SIZE - 1
108                                     downto 0) := (others => '0');
109     -- Segment end value
110
111     -- Internal component constant decelerations
112     constant c_zeros      : unsigned(AOI_ADDRESS_SIZE - 4 - 1
113                                     downto 0) := (others => '0');
114
115 begin
116     -- Assign the incoming signals to internal component signals
117     s_end_of_line <= end_of_line_in; -- Signal for the end of
118                                      -- line flag
119     s_tap_counter <= tap_counter_in; -- Signal of the tap counter
120     s_ss1_active  <= ss1_active_in;  -- Sub-segment 1 active flag
121     s_ss1_start   <= ss1_start_in;   -- Sub-segment 1 start value
122     s_ss1_end     <= ss1_end_in;     -- Sub-segment 1 end value
123     s_ss2_active  <= ss2_active_in;  -- Sub-segment 2 active flag
124     s_ss2_start   <= ss2_start_in;   -- Sub-segment 2 start value
125     s_ss2_end     <= ss2_end_in;     -- Sub-segment 2 end value
126
127
128     update_segment: process(clk, rst_n)
129         variable v_seg_start : std_logic_vector(AOI_ADDRESS_SIZE
130                                     - 1 DOWNTO 0) := s_seg_start;
131         -- Internal process variable for the segment start value
```

```
132         -- that will change reguardless of the clock edge
133         variable v_seg_mid    : std_logic_vector(AOI_ADDRESS_SIZE
134                                 - 1 DOWNTO 0) := s_seg_mid;
135         -- Internal process variable for the segment middle value
136         -- that will change reguardless of the clock edge
137         variable v_seg_end    : std_logic_vector(AOI_ADDRESS_SIZE
138                                 - 1 DOWNTO 0) := s_seg_end;
139         -- Internal process variable for the segment end value
140         -- that will change reguardless of the clock edge
141     begin
142         if(rst_n = '0') then -- On reset,
143             s_seg_active <= '0';
144             -- Clear the active flag
145             v_seg_start  := (others => '0');
146             -- Clear the segment start value
147             v_seg_mid    := (others => '0');
148             -- Clear the segment middle value
149             v_seg_end    := (others => '0');
150             -- Clear the segment end value
151         elsif(rising_edge(clk)) then
152         -- Trigger on the rising edge of the system clock
153             if(s_end_of_line = '1') then
154             -- At the end of the line, reset the signals
155             -- and variables
156                 s_seg_active <= '0';
157                 -- Clear the active flag
158                 v_seg_start  := (others => '0');
159                 -- Clear the segment start value
160                 v_seg_mid    := (others => '0');
161                 -- Clear the segment middle value
162                 v_seg_end    := (others => '0');
163                 -- Clear the segment end value
164             else -- Otherwise,
165                 if(s_seg_active = '1') then
166                 -- If the segment is active, then
167                     if (s_ss1_active = '1') then
168                     -- If sub-segment 1 is active, then
169                         v_seg_end:=std_logic_vector(s_tap_counter
170                                 + (c_zeros & unsigned(
171                                 s_ss1_end)));
172                         -- Increase the segment end value by
173                         -- adding the current pixel location with
174                         -- the sub-segment 1 end value
175                     end if;
176                     if (s_ss2_active = '1') then
177                     -- If sub-segment 2 is active,
178                     -- then - Might not be necessary
179                         v_seg_end:=std_logic_vector(s_tap_counter
```

```
180                     + (c_zeros & unsigned(s_ss2_end)));
181                     -- Increase the segment end value by
182                     -- adding the current pixel location with
183                     -- the sub-segment 2 end value
184                 end if;
185             else -- Otherwise, create a new segement
186                 if (s_ss1_active = '1' OR s_ss2_active = '1')
187                  then
188                 -- If either sub-segments are active, then
189                     s_seg_active <= '1';
190                     -- Set the segment active flag
191                 end if;
192                 if (s_ss1_active = '1') then
193                 -- If sub-segment 1 is active, then
194                     v_seg_start := std_logic_vector(
195                                    s_tap_counter + (c_zeros
196                                    & unsigned(s_ss1_start)));
197                     -- The new segment start is equal to the
198                     -- pixel value plus the sub-segment 1
199                     -- start value
200                     v_seg_end   := std_logic_vector(
201                                     s_tap_counter + (c_zeros
202                                     & unsigned(s_ss1_end)));
203                     -- The new segment end is equal to the
204                     -- pixel value plus the sub-segment 1
205                     -- end value
206                 elsif (s_ss2_active = '1') then
207                 -- Else if sub-segment 2 is active, then
208                     v_seg_start := std_logic_vector(
209                                    s_tap_counter + (c_zeros
210                                    & unsigned(s_ss2_start)));
211                     -- The new segment start is equal to the
212                     -- pixel value plus the sub-segment 2
213                     --start value
214                     v_seg_end   := std_logic_vector(
215                                     s_tap_counter + (c_zeros
216                                     & unsigned(s_ss2_end)));
217                     -- The new segment end is equal to the
218                     --pixel value plus the sub-segment 2 end
219                     -- value
220                 end if;
221             end if;
222             -- Assign the updated values on every clock edge
223             s_seg_start     <= v_seg_start;
224             -- Assign the segment start variable
225             -- to the segment start signal
226             s_seg_mid       <= v_seg_mid;
227             -- Assign the segment middle variable
```

```
228                 -- to the segment middle signal
229                 s_seg_end       <= v_seg_end;
230                 -- Assign the segment end variable
231                 -- to the segment end signal
232             end if;
233             -- Calculate seg_mid on every clock cycle
234             -- regardless of new data
235             v_seg_mid := std_logic_vector(unsigned(v_seg_end) -
236                                           unsigned(v_seg_start));
237             -- Subtract the end value from the start value
238             v_seg_mid := '0' & v_seg_mid(AOI_ADDRESS_SIZE - 1
239                                          downto 1);
240             -- Shift the result of the subtraction right by
241             -- one bit to divide by 2
242             v_seg_mid := std_logic_vector(unsigned(v_seg_mid) +
243                                           unsigned(v_seg_start));
244             -- Add the shifted result to the start value to get
245             -- the middle value
246             else
247             -- Verify that this change can be done in simulation
248                 s_seg_active <= s_seg_active;
249                 v_seg_start  := v_seg_start;
250                 v_seg_mid    := v_seg_mid;
251                 v_seg_end    := v_seg_end;
252             end if;
253             -- Assign the updated values on every clock edge
254             s_seg_start     <= v_seg_start;
255             -- Assign the segment start variable to
256             -- the segment start signal
257             s_seg_mid       <= v_seg_mid;
258             -- Assign the segment middle variable to
259             -- the segment middle signal
260             s_seg_end       <= v_seg_end;
261             -- Assign the segment end variable to the
262             -- segment end signal
263 --      end if;
264     end process;
265
266     -- Assign the output signals
267     seg_start_out <= s_seg_start; -- Segment start value output
268     seg_mid_out   <= s_seg_mid;   -- Segment middle value output
269     seg_end_out   <= s_seg_end;   -- Segment end value output
270
271 end architecture;
```

```
1 ------------------------------------------------------------------------
2 --
3 --! @file        mod4_seg2obj.vhd
```

```
4  --! @brief       Convert the segments into object with an object
5  --!              number.
6  --! @details
7  --
8  --! @author      Connor Dack
9  --! @date        November 2016
10 --! @copyright   Copyright (C) 2015 Ross K. Snider and Connor Dack
11 --
12 --   This program is free software: you can redistribute it
13 --   and/or modify it under the terms of the GNU General
14 --   Public License as published by the Free Software Foundation,
15 --   either version 3 of the License, or (at your option) any
16 --   later version.
17 --
18 --   This program is distributed in the hope that it will be
19 --   useful, but WITHOUT ANY WARRANTY; without even the implied
20 --   warranty of MERCHANTABILITY or FITNESS for A PARTICULAR
21 --   PURPOSE. See the GNU General Public License for more details.
22 --
23 --   You should have received a copy of the GNU General Public
24 --   License along with this program.  If not, see
25 --   <http://www.gnu.org/licenses/>.
26 --
27 --   Connor Dack
28 --   Electrical and Computer Engineering
29 --   Montana State University
30 --   610 Cobleigh Hall
31 --   Bozeman, MT 59717
32 --   connor.dack@msu.montana.edu
33 --
34 ------------------------------------------------------------------------
35
36
37 LIBRARY IEEE;
38 USE IEEE.STD_LOGIC_1164.ALL;
39 USE IEEE.NUMERIC_STD.ALL;
40
41 USE work.USER_CONTROLS.ALL;
42
43 ------------------------------------------------------------------------
44 --
45 --! @brief       mod4_seg2obj
46 --! @details
47 --!
48 --! @param    clk                 -  Input: System Clock
49 --! @param    rst_n               -  Input: Active low reset
50 --! @param    line_number         -  Input: Current line number
51 --! @param    end_of_line      -  Input: Indicate that
```

```
52 --!                                                  Segments need to be
53 --!                                                  added
54 --! @param    seg_start_in          - Input: Data from the current
55 --!                                                  line - start points
56 --! @param    seg_mid_in            - Input: Data from the current
57 --!                                                  line - mid points
58 --! @param    seg_end_in            - Input: Data from the current
59 --!                                                  line - end points
60 --! @param    mod_5_trigger_out     - Output: Triggers Mod_5
61 --! @param    line_scan_packet_out - Output: The Line Scan Packet
62 --!                                                  to be sent to the
63 --!                                                  Transceiver FIFO
64 --! @param    mod_4_wr_en_out       - Output: Write Enable for the
65 --!                                                  Transceiver FIFO
66 --
67 ----------------------------------------------------------------------------
68 entity mod4_seg2obj is
69   port(clk             :  in std_logic;
70        rst_n           :  in std_logic;
71            line_number           :  in std_logic_vector(
72                                     LINE_NUMBER_ADDRESS_SIZE - 1
73                                     downto 0);
74        end_of_line       :  in std_logic;
75            seg_start_in          :  in seg_info_array;
76            seg_mid_in            :  in seg_info_array;
77            seg_end_in            :  in seg_info_array;
78        mod_5_trigger_out     : out std_logic;
79            line_scan_packet_out  : out std_logic_vector(
80                                     OBJECT_ADDRESS_SIZE +
81                                     LINE_NUMBER_ADDRESS_SIZE +
82                                     (3 * AOI_ADDRESS_SIZE) - 1
83                                     downto 0);
84            mod_4_wr_en_out       : out std_logic);
85 end entity;
86
87 architecture rtl of mod4_seg2obj is
88
89    type array_of_object_numbers is array (1 to MAX_OBJECT_NUMBER)
90       of std_logic_vector(OBJECT_ADDRESS_SIZE - 1 downto 0);
91
92     component mod4_seg_route is
93        port (clk                  :  in std_logic;
94              rst_n             :  in std_logic;
95              enable               :  in std_logic;
96              seg_start_in     :  in std_logic_vector(
97                                    AOI_ADDRESS_SIZE - 1 downto 0);
98              seg_mid_in        :  in std_logic_vector(
99                                    AOI_ADDRESS_SIZE - 1 downto 0);
```

```
100             seg_end_in          :  in std_logic_vector(
101                                    AOI_ADDRESS_SIZE - 1 downto 0);
102             obj_active_in       :  in std_logic;
103             obj_last_x_start_in :  in std_logic_vector(
104                                    AOI_ADDRESS_SIZE - 1 downto 0);
105             obj_last_x_end_in   :  in std_logic_vector(
106                                    AOI_ADDRESS_SIZE - 1 downto 0);
107             obj_done_in         :  in std_logic;
108         obj_active_out      : out std_logic;
109             inObj_out           : out std_logic;
110             no_match_out        : out std_logic);
111     end component;
112
113     signal s_mod_5_ready        : std_logic := '0';
114     signal s_line_scan_packet   : std_logic_vector(
115                                    OBJECT_ADDRESS_SIZE +
116                                    LINE_NUMBER_ADDRESS_SIZE +
117                                    AOI_ADDRESS_SIZE +
118                                    AOI_ADDRESS_SIZE +
119                                    AOI_ADDRESS_SIZE - 1
120                                    downto 0) := (others => '0');
121     signal s_mod_4_wr_en        : std_logic := '0';
122     signal s_seg_start          : seg_info_array := (others =>
123                                                     (others => '0'));
124     signal s_seg_mid            : seg_info_array := (others =>
125                                                     (others => '0'));
126     signal s_seg_end            : seg_info_array := (others =>
127                                                     (others => '0'));
128     signal s_current_seg_start  : seg_info_array := (others =>
129                                                     (others => '0'));
130     signal s_current_seg_mid    : seg_info_array := (others =>
131                                                     (others => '0'));
132     signal s_current_seg_end    : seg_info_array := (others =>
133                                                     (others => '0'));
134     signal s_last_x_start       : seg_info_array := (others =>
135                                                     (others => '0'));
136     signal s_last_x_end         : seg_info_array := (others =>
137                                                     (others => '0'));
138     signal s_seg_start_single   : std_logic_vector(AOI_ADDRESS_SIZE
139                                   - 1 downto 0) := (others => '0');
140     signal s_seg_mid_single     : std_logic_vector(AOI_ADDRESS_SIZE
141                                   - 1 downto 0) := (others => '0');
142     signal s_seg_end_single     : std_logic_vector(AOI_ADDRESS_SIZE
143                                   - 1 downto 0) := (others => '0');
144     signal s_valid_data_array   : std_logic := '0';
145     signal s_array_counter      : integer range 1 to
146                                   MAX_OBJECT_NUMBER := 1;
147     signal s_active_objects     : std_logic_vector(1 to
```

```
148                                     MAX_OBJECT_NUMBER) := (others
149                                     => '0');
150     signal s_inObj_out          : std_logic_vector(1 to
151                                     MAX_OBJECT_NUMBER) := (others
152                                     => '0');
153     signal s_new_obj            : std_logic_vector(1 to
154                                     MAX_OBJECT_NUMBER) := (others
155                                     => '0');
156     signal s_no_match           : std_logic_vector(1 to
157                                     MAX_OBJECT_NUMBER) := (others
158                                     => '0');
159     signal s_mod4_seg_route_en  : std_logic := '0';
160     signal s_mod4_new_obj_en    : std_logic := '0';
161
162     signal l                    : integer range 1 to
163                                             MAX_OBJECT_NUMBER := 1;
164     signal s_current_objs       : std_logic_vector(1 to
165                                     MAX_OBJECT_NUMBER) := (others
166                                     => '0');
167     signal s_data_valid         : std_logic := '0';
168     signal s_obj_done           : std_logic_vector(1 to
169                                     MAX_OBJECT_NUMBER) := (others
170                                     => '0');
171     constant c_line_counter     : integer := 4;
172     signal s_hyperspectral_line_counter : integer range 1 to
173                                             c_line_counter + 1 := 1;
174     signal f_object_done        : std_logic := '0';
175     signal f_object_ready       : std_logic := '0';
176     signal f_new_object         : std_logic := '0';
177     signal f_delay              : std_logic := '0';
178     signal k                    : integer range 1 to
179                                     MAX_OBJECT_NUMBER := 1;
180     constant c_done             : std_logic_vector((3 *
181                                      AOI_ADDRESS_SIZE) - 1
182                                      downto 0) := (others => '1');
183     signal f_no_data : std_logic := '0';
184     constant c_current_seg : seg_info_array := (others =>
185                                                 (others => '0'));
186
187 begin
188     -- Assign the input signals to internal component signals
189     s_seg_start <= seg_start_in;-- saves the current start values
190     s_seg_mid   <= seg_mid_in; -- saves the current middle values
191     s_seg_end   <= seg_end_in; -- saves the current end values
192
193     --Register the Previous Array Day on at the end of every line
194     register_data_array : process (clk,rst_n)
195         begin
```

```
196             if(rst_n = '0') then -- Clear the registers
197                 s_last_x_start <= (others => (others => '0'));
198                 -- the starting values of the objects on the
199                 -- previous line
200                 s_last_x_end   <= (others => (others => '0'));
201                 -- the ending values of the objects on the
202                 -- previous line
203             elsif(rising_edge(clk)) then
204             -- Trigger on the rising edge of the system clock
205                 if(end_of_line = '1') then
206                 -- wait until the end of the line to
207                 -- register the new data
208                     s_current_seg_start <= s_seg_start;
209                     s_current_seg_mid   <= s_seg_mid;
210                     s_current_seg_end   <= s_seg_end;
211                     s_last_x_start <= s_current_seg_start;
212                     -- saves the previous start values
213                     s_last_x_end   <= s_current_seg_end;
214                     -- saves the previous end values
215                 end if;
216             end if;
217         end process;
218
219     -- Walk through the current start, middle and end values
220     -- using a counter
221     pipeline_data_array : process(clk,rst_n)
222         begin
223             if(rst_n = '0') then
224             -- Reset the start, middle, end and enable values
225                 s_seg_start_single  <= (others => '0');
226                 -- clear the singles start value
227                 s_seg_mid_single    <= (others => '0');
228                 -- clear the single middle value
229                 s_seg_end_single    <= (others => '0');
230                 -- clear the single end value
231                 s_valid_data_array  <= '0';
232                 -- clear the data array valid flag
233                 s_mod4_seg_route_en <= '0';
234                 -- clear the route enable flag
235             elsif(rising_edge(clk)) then
236             -- Trigger on the rising edge of the system clock
237                 if(end_of_line = '1') then
238                 -- wait until the end of line signal to set the
239                 -- valid data array flag and the route enable flag
240                     s_valid_data_array  <= '1';
241                     -- Flag will trigger the counter
242                     s_mod4_seg_route_en <= '1';
243                     -- Flag will trigger the segment rounting
```

```
244                     -- component
245                 end if;
246                 if(s_valid_data_array = '1') then
247                 -- If the data is valid, then -
248                     if(s_array_counter = MAX_OBJECT_NUMBER) then
249                     -- If the counter has reached the maximum
250                     -- number of objects, then -
251                         s_array_counter     <= 1;
252                         -- Reset the array counter
253                         s_valid_data_array  <= '0';
254                         -- Clear the data arrray valid flag
255                         -- completed the current line of data
256                         s_mod4_seg_route_en <= '0';
257                         -- Clear the segment route enable flag
258                         -- stop the component because there is
259                         -- no new data
260                     else -- Otherwise
261                         s_array_counter <= s_array_counter + 1;
262                         -- Increment the array counter by 1
263                     end if;
264                     -- The registered data will be used during
265                     -- the segment routing step
266                     s_seg_start_single <= s_current_seg_start(
267                                           s_array_counter);
268                     -- Take the start value at the array counter
269                     -- and register it to a signal
270                     s_seg_mid_single   <= s_current_seg_mid(
271                                           s_array_counter);
272                     -- Take the middle value at the array counter
273                     -- and register it to a signal
274                     s_seg_end_single   <= s_current_seg_end(
275                                           s_array_counter);
276                     -- Take the end value at the array counter
277                     -- and register it to a signal
278                 end if;
279             end if;
280         end process;
281
282     -- Set the new object enable flag to create a new object
283     mod4_new_obj_en_proc : process(clk,rst_n)
284         begin
285             if(rst_n = '0') then
286             -- On reset, clear the new object enable flag
287                 s_mod4_new_obj_en <= '0';
288                 -- Clear the new object enable flag
289             elsif(rising_edge(clk)) then
290             -- Trigger on the rising edge of the system clock
291                 if(s_mod4_seg_route_en = '1') then
```

```
292                     s_mod4_new_obj_en <= '1';
293                 else
294                     s_mod4_new_obj_en <= '0';
295                 end if;
296             end if;
297         end process;
298 
299         -- Generate statement to compute the different object
300         -- information in parallel
301         seg_routing_gen_1 : for i in 1 to MAX_OBJECT_NUMBER generate
302             seg_routing : mod4_seg_route port map(
303                     clk                 => clk,
304                     rst_n           => rst_n,
305                     enable              => s_mod4_seg_route_en,
306                     seg_start_in        => s_seg_start_single,
307                     seg_mid_in          => s_seg_mid_single,
308                     seg_end_in          => s_seg_end_single,
309                     obj_active_in       => s_current_objs(i),
310                     obj_last_x_start_in => s_last_x_start(i),
311                     obj_last_x_end_in   => s_last_x_end(i),
312                     obj_done_in         => s_obj_done(i),
313                     obj_active_out      => s_active_objects(i),
314                     inObj_out           => s_inObj_out(i),
315                     no_match_out        => s_no_match(i));
316         end generate;
317 
318         -- New Object Process
319         new_object_process : process(clk, rst_n)
320             begin
321                 if(rst_n = '0') then -- On reset,
322                     s_current_objs <= (others => '0');
323                     -- Clear the current active object flag array
324                     f_new_object   <= '0';
325                     -- Clear the new object flag
326                     f_object_ready <= '0';
327                     -- Clear the object ready flag
328                     k              <= 1;
329                     -- Clear the process counter
330                 elsif(rising_edge(clk)) then
331                 -- On the rising edge of the component clock
332                     if(s_data_valid = '1') then
333                     -- If the data valid signal is a one, then
334                         if(f_new_object = '1') then
335                         -- If the new object flag is a one, then
336                             k <= 1;
337                             -- Clear the process counter
338                         else -- Otherwise,
339                             if(k = MAX_OBJECT_NUMBER) then
```

```
340                         -- If the process counter is at the max,
341                         -- then
342                             k <= 1;
343                             -- Clear the process counter
344                         else -- Otherwise,
345                             k <= k + 1;
346                             -- Increment the process counter
347                         end if;
348                         if(s_obj_done(k) = '1') then
349                         -- If the object done flag is one, then
350                             s_current_objs(k) <= '0';
351                             -- set the object active flag to zero
352                         elsif(s_current_objs(k) = '1') then
353                         -- If the current object is active, then
354                             s_current_objs(k) <= '1';
355                             -- Don't change the falg
356                             f_object_ready <= '1';
357                             -- Set the object ready flag
358                         elsif((s_active_objects(k) = '1' and
359                                 s_current_objs(k) = '0') and
360                                (s_active_objects(k+1) = '1' and
361                                 s_current_objs(k+1) = '0')) then
362                          -- If a new object is found, then
363                             s_current_objs(k) <= '1';
364                             -- Set the current object flag
365                             f_object_ready <= '1';
366                             -- Set the object ready flag
367                         else -- Otherwise
368                             f_object_ready <= '0';
369                             -- Clear the object ready flag
370                         end if;
371                     end if;
372                 else -- Otherwise,
373                     f_new_object <= '0';
374                     -- clear the new object flag
375                     k <= 1;
376                     -- Reset the process counter
377                 end if;
378             end if;
379         end process;
380
381     -- Data valid process that sets the data valid flag
382     data_valid_process : process(clk, rst_n)
383         begin
384             if(rst_n = '0') then -- On reset,
385                 s_data_valid <= '0';
386                 -- clear the data valid flag
387             elsif(rising_edge(clk)) then
```

```
388             -- On the rising edge of the component clock
389                 if(s_valid_data_array = '1') then
390                 -- If the valid data flag is a one, then
391                     if(f_delay = '1') then
392                         s_data_valid <= '1';
393                         -- set the data valid flag
394                     else
395                         f_delay <= '1';
396                     end if;
397                 else -- Otherwise,
398                     f_delay <= '0';
399                     s_data_valid <= '0';
400                     -- clear the data flag
401                 end if;
402             end if;
403         end process;
404 
405     -- No data process
406     no_data_proc : process(clk, rst_n)
407         begin
408             if(rst_n = '0') then -- On reset, then
409                 f_no_data <= '0'; -- clear the no data flag
410             elsif(rising_edge(clk)) then
411             -- On the rising edge of the component clock
412                 if(s_valid_data_array = '1') then
413                 -- If the data valid flag is a one, then
414                     if(s_current_seg_start = c_current_seg and
415                        s_current_seg_mid = c_current_seg and
416                        s_current_seg_end = c_current_seg and
417                        s_last_x_start = c_current_seg and
418                        s_last_x_end = c_current_seg) then
419                     -- If there is no data in the current or
420                     -- previous pixel lines
421                         f_no_data <= '1';
422                         -- Set the no data flag to a one
423                     else -- Otherwise
424                         f_no_data <= '0';
425                         -- Clear the no data flag to a zero
426                     end if;
427                 else -- Otherwise
428                     f_no_data <= '0';
429                     -- Clear the no data flag
430                 end if;
431             end if;
432         end process;
433 
434     -- Create the data output packets
435     data_out_process : process(clk, rst_n)
```

```
436         begin
437             if(rst_n = '0') then -- On reset,
438             -- clear the following signals
439                 s_mod_5_ready       <= '0';
440                 s_line_scan_packet <= (others => '0');
441                 s_mod_4_wr_en       <= '0';
442                 s_hyperspectral_line_counter <= 1;
443             elsif(rising_edge(clk)) then
444             -- On the rising edge of the component clock,
445                 if(f_object_ready = '1') then
446                     f_object_done <= '1';
447                 end if;
448                 if(f_object_done = '1') then
449                 -- check the data valid flag, if it is a 1, then
450                     if(l = MAX_OBJECT_NUMBER - 1) then
451                     -- If the counter equals the maximum number
452                     -- of objects, then
453                         l <= 1;
454                         -- reset the counter
455                         f_object_done <= '0';
456                     else -- Otherwise
457                         l <= l + 1; -- increment the counter by 1
458                     end if;
459                     if(s_active_objects(l) = '1' and
460                        s_no_match(l) = '0' and
461                        f_no_data = '1') then
462                     -- If the active object is set and the no
463                     -- match flag is zero, then set the object
464                     -- done flag
465                         s_obj_done(l) <= '1';
466                         s_mod_4_wr_en <= '0';
467                         s_line_scan_packet <= std_logic_vector(
468                                               to_unsigned(l,
469                                               OBJECT_ADDRESS_SIZE))
470                                              & line_number & c_done;
471                         s_mod_5_ready       <= '1';
472                     elsif(s_active_objects(l) = '1' and
473                       (s_inObj_out(l) = '1' or s_current_objs(l)
474                        = '1')and s_no_match(l) = '0') then
475                        -- there is data for an object
476                         s_line_scan_packet <= std_logic_vector(
477                                               to_unsigned(l,
478                                               OBJECT_ADDRESS_SIZE))
479                                               & line_number &
480                                               s_current_seg_start(l) &
481                                               s_current_seg_mid(l) &
482                                               s_current_seg_end(l);
483                         -- Create the object packet
```

```
484                         s_mod_5_ready      <= '1';
485                         -- Set the Mod 5 Ready flag
486                         s_mod_4_wr_en      <= '1';
487                         -- Set the Mod 4 write enable signal
488                     elsif(s_active_objects(l) = '1' and
489                           s_obj_done(l) = '1') then
490                         -- If there is an active object and
491                         -- the object done flag is set, then
492                         if(s_hyperspectral_line_counter =
493                            c_line_counter + 1) then
494                             -- If the delay counter is 5, then
495                             -- Clear the object done flag and
496                             -- reset the line counter
497                             s_hyperspectral_line_counter <= 1;
498                             s_obj_done(l) <= '0';
499                             s_mod_4_wr_en <= '0';
500                         else -- Otherwise,
501                         -- Increment the line counter and
502                         -- register the object done flag
503                             s_hyperspectral_line_counter <=
504                                 s_hyperspectral_line_counter + 1;
505                             s_obj_done(l) <= '1';
506                             s_mod_4_wr_en <= '0';
507                         end if;
508                     elsif(s_active_objects(l) = '1' and
509                                         s_obj_done(l) = '0') then
510                     -- If the object active flag is set and the
511                     -- object done flag is zero, then clear the
512                     -- output packet and the mod 5 ready signal
513                         s_line_scan_packet <= (others => '0');
514                         s_mod_5_ready      <= '0';
515                         s_mod_4_wr_en      <= '0';
516                     else -- Otherwise,
517                     -- Clear the output signals
518                         s_line_scan_packet <= (others => '0');
519                         s_mod_5_ready      <= '0';
520                         s_mod_4_wr_en      <= '0';
521                         s_obj_done(l)      <= '0';
522                     end if;
523                 end if;
524             end if;
525         end process;
526
527         -- Assign the output signals
528         mod_5_trigger_out    <= s_mod_5_ready;
529         line_scan_packet_out <= s_line_scan_packet;
530         mod_4_wr_en_out      <= s_mod_4_wr_en;
531
```

```
532 end architecture;

1  ----------------------------------------------------------------------------
2  --
3  --! @file        mod4_seg_route.vhd
4  --! @brief
5  --! @details
6  --
7  --! @author      Connor Dack
8  --! @date        November 2016
9  --! @copyright   Copyright (C) 2015 Ross K. Snider and Connor Dack
10 --
11 --   This program is free software: you can redistribute it
12 --   and/or modify it under the terms of the GNU General
13 --   Public License as published by the Free Software Foundation,
14 --   either version 3 of the License, or (at your option) any
15 --   later version.
16 --
17 --   This program is distributed in the hope that it will be
18 --   useful, but WITHOUT ANY WARRANTY; without even the implied
19 --   warranty of MERCHANTABILITY or FITNESS for A PARTICULAR
20 --   PURPOSE. See the GNU General Public License for more details.
21 --
22 --   You should have received a copy of the GNU General Public
23 --   License along with this program.  If not, see
24 --   <http://www.gnu.org/licenses/>.
25 --
26 --   Connor Dack
27 --   Electrical and Computer Engineering
28 --   Montana State University
29 --   610 Cobleigh Hall
30 --   Bozeman, MT 59717
31 --   connor.dack@msu.montana.edu
32 --
33 ----------------------------------------------------------------------------
34
35 LIBRARY IEEE;
36 USE IEEE.STD_LOGIC_1164.ALL;
37 USE IEEE.NUMERIC_STD.ALL;
38
39 USE work.USER_CONTROLS.ALL;
40
41 ----------------------------------------------------------------------------
42 --
43 --! @brief       mod4_seg_route
44 --! @details
45 --!
46 --! @param   clk                    -   Input: System Clock
```

```
47 --! @param   rst_n                - Input: Active low reset
48 --! @param   enable               - Input: Submodual Enable Signal
49 --! @param   seg_start_in         - Input: Data from the current
50 --!                                       line - start points
51 --! @param   seg_mid_in           - Input: Data from the current
52 --!                                       line - mid points
53 --! @param   seg_end_in           - Input: Data from the current
54 --!                                       line - end points
55 --! @param   obj_active_in        - Input: Structure Active Signal
56 --! @param   obj_last_x_start_in - Input: Data from the last line
57 --! @param   obj_last_x_end_in   - Input: Data from the last line
58 --! @param   obj_done_in          - Input: Signal that the object
59 --!                                       has been ejected
60 --! @param   obj_active_out      - Output:
61 --! @param   inObj_out           - Output: If the data is in this
62 --!                                       object, inobj_out = 1
63 --! @param   no_match_out        - Output: If no match was found
64 --!                                       for the data,
65 --!                                       no_match_out = 1
66 --
67 ----------------------------------------------------------------------
68 entity mod4_seg_route is
69   port (clk                    :  in std_logic;
70         rst_n          :  in std_logic;
71         enable                 :  in std_logic;
72         seg_start_in      :  in std_logic_vector(
73                                   AOI_ADDRESS_SIZE - 1
74                                   downto 0);
75         seg_mid_in       :  in std_logic_vector(
76                                   AOI_ADDRESS_SIZE - 1
77                                   downto 0);
78         seg_end_in       :  in std_logic_vector(
79                                   AOI_ADDRESS_SIZE - 1
80                                   downto 0);
81         obj_active_in     :  in std_logic;
82         obj_last_x_start_in   :  in std_logic_vector(
83                                   AOI_ADDRESS_SIZE - 1
84                                   downto 0);
85         obj_last_x_end_in   :  in std_logic_vector(
86                                   AOI_ADDRESS_SIZE - 1
87                                   downto 0);
88         obj_done_in           :  in std_logic;
89       obj_active_out        : out std_logic;
90         inObj_out       : out std_logic;
91         no_match_out      : out std_logic);
92 end entity;
93
94 architecture seg_route_arch of mod4_seg_route is
```

```
95
96      -- Internal component signals for the input and outputs of
97      -- the component
98    signal s_inObj            : std_logic;
99      signal s_seg_start        : std_logic_vector(AOI_ADDRESS_SIZE
100                                                  - 1 downto 0);
101     signal s_seg_mid          : std_logic_vector(AOI_ADDRESS_SIZE
102                                                  - 1 downto 0);
103     signal s_seg_end          : std_logic_vector(AOI_ADDRESS_SIZE
104                                                  - 1 downto 0);
105     signal s_obj_last_x_start : std_logic_vector(AOI_ADDRESS_SIZE
106                                                  - 1 downto 0);
107     signal s_obj_last_x_end   : std_logic_vector(AOI_ADDRESS_SIZE
108                                                  - 1 downto 0);
109     signal s_obj_done         : std_logic;
110   signal s_obj_active       : std_logic := '0';
111     signal s_enable           : std_logic := '0';
112     signal s_no_match         : std_logic := '0';
113
114     -- Constant signal of zeros for comparison
115     constant c_zeros          : std_logic_vector(AOI_ADDRESS_SIZE
116                                                  - 1 downto 0)
117                                                  := (others => '0');
118
119 begin
120     -- Assign the inputs to internal component signals
121     s_seg_start        <= seg_start_in;
122     -- assign the start value
123     s_seg_mid          <= seg_mid_in;
124     -- assign the middle value
125     s_seg_end          <= seg_end_in;
126     -- assign the end value
127     s_obj_last_x_start <= obj_last_x_start_in;
128     -- assign the previous start value
129     s_obj_last_x_end   <= obj_last_x_end_in;
130     -- assign the previous end value
131     s_enable           <= enable;
132     -- assign the enable flag
133   ------------------------------------------------------------------
134   -- About the route process
135   ------------------------------------------------------------------
136   -- Finds which object the segment is in.
137     --    Either sets or clears the in object flag,
138     --    depending if an object was found or not
139     --    Either sets or clears the no match flag,
140     --    depending if an ojbect was found or not
141   ------------------------------------------------------------------
142   route: process(clk,rst_n)
```

```
143   begin
144     if (rst_n = '0') then
145         -- On reset, clear the in object signal
146         -- and the no match flag
147       s_inObj    <= '0'; -- Clear the in object flag
148             s_no_match <= '0'; -- Clear the no match flag
149     elsif(rising_edge(clk)) then
150         -- Trigger on the rising edge of the system clock
151             if(s_enable = '1') then
152             -- Trigger on the enable signal
153                 -- Check for within the active objects range
154                 if (obj_active_in = '1') then
155                 -- If the object is active, then
156                     if(s_obj_done = '1') then
157                     -- If the object has been ejected,
158                     -- clear the information about the data
159                         s_obj_active <= '0';
160                         -- Clear the object flag so that it can
161                         -- be reused
162                         s_inObj      <= '0';
163                         -- Clear the in object flag, data was
164                         -- not a match
165                         s_no_match   <= '0';
166                         -- Clear the no match flag, data was
167                         -- not in this object
168
169                     -- Check to see if the current data is
170                     -- within the active object
171                     elsif(s_seg_start >= s_obj_last_x_start and
172                           s_seg_start <= s_obj_last_x_end) then
173                           -- Check the start value of the object
174                         s_obj_active <= '1';
175                         -- Don't change the flag state
176                         s_inObj      <= '1';
177                         -- It was found, set the in object flag
178                         s_no_match   <= '0';
179                         -- It was found, clear the no match flag
180                     elsif(s_seg_mid >= s_obj_last_x_start and
181                           s_seg_mid <= s_obj_last_x_end) then
182                           -- Check the middle value of the object
183                         s_obj_active <= '1';
184                         -- Don't change the flag state
185                         s_inObj      <= '1';
186                         -- It was found, set the in object flag
187                         s_no_match   <= '0';
188                         -- It was found, clear the no match flag
189                     elsif(s_seg_end >= s_obj_last_x_start
190                           and s_seg_end <= s_obj_last_x_end) then
```

```
191                             -- Check the end value of the object
192                         s_obj_active <= '1';
193                         -- Don't change the flag state
194                         s_inObj      <= '1';
195                         -- It was found, set the in object flag
196                         s_no_match   <= '0';
197                         -- It was found, clear the no match flag
198                     end if;
199                 elsif(s_seg_start = c_zeros and s_seg_mid =
200                      c_zeros and s_seg_end = c_zeros) then
201                      -- If the incoming object data is zero, then
202                     s_obj_active <= '0';
203                     -- There is no object data, clear the
204                     -- object active flag
205                     s_inObj      <= '0';
206                     -- There is no object data, clear the in
207                     -- object flag
208                     s_no_match   <= '0';
209                     -- There is no object data, clear the no
210                     -- match flag
211                 else -- Otherwise, there is object data but
212                      --no object was found
213                     s_obj_active <= '1';
214                   -- Set the object active flag for the next pass
215                     s_inObj      <= '0';
216                   -- Clear the in object flag
217                     s_no_match   <= '1';
218                   -- Set the no match flag to create a new object
219                 end if;
220             end if;
221         end if;
222   end process;
223
224   -- assign outputs
225     obj_active_out <= s_obj_active;
226     -- is '1' when an object is active
227   inObj_out      <= s_inObj;
228     -- is '1' when the data is in an active object
229   no_match_out   <= s_no_match;
230     -- is '1' when no match was found for the data
231
232 end architecture;
```

```
1 ----------------------------------------------------------------------
2 --
3 --! @file        camera_relations.vhd
4 --! @brief       Converts from the line scan space to the
5 --!               the hyperspectral space
```

```
6  --! @details
7  --
8  --! @author      Monica Whitaker
9  --! @date        September 2016
10 --! @copyright   Copyright (C) 2015 Ross K. Snider,Monica Whitaker
11 --!              and Connor Dack
12 --
13 --   This program is free software: you can redistribute it
14 --   and/or modify it under the terms of the GNU General
15 --   Public License as published by the Free Software Foundation,
16 --   either version 3 of the License, or (at your option) any
17 --   later version.
18 --
19 --   This program is distributed in the hope that it will be
20 --   useful, but WITHOUT ANY WARRANTY; without even the implied
21 --   warranty of MERCHANTABILITY or FITNESS for A PARTICULAR
22 --   PURPOSE. See the GNU General Public License for more details.
23 --
24 --   You should have received a copy of the GNU General Public
25 --   License along with this program.  If not, see
26 --   <http://www.gnu.org/licenses/>.
27 --
28 --   Connor Dack
29 --   Electrical and Computer Engineering
30 --   Montana State University
31 --   610 Cobleigh Hall
32 --   Bozeman, MT 59717
33 --   connor.dack@msu.montana.edu
34 --
35 ----------------------------------------------------------------------------
36
37 library IEEE;
38 use IEEE.STD_LOGIC_1164.ALL;
39 use IEEE.NUMERIC_STD.ALL;
40
41 use work.USER_CONTROLS.ALL;
42 ----------------------------------------------------------------------------
43 --
44 --! @brief     camera_relations
45 --!
46 --! @param     clk                       -  Input: System Clock
47 --! @param     rst_n                     -  Input: Active Low Reset
48 --! @param     line_reset                -  Input: Line Reset Flag
49 --! @param     linescan_offset           -  Input: Offset Value
50 --! @param     line_scan_packet_in       -  Input: Data In
51 --! @param     hyperspectral_packet_out  - Output: Data Out
52 --! @param     transciever_obj_wre_out   - Output: FIFO Write En
53 ----------------------------------------------------------------------------
```

```
54
55  entity camera_relations is
56      port(clk                        : in std_logic;
57           rst_n                      : in std_logic;
58           line_reset                 : in std_logic;
59           linescan_offset            : in std_logic_vector(
60                                        AOI_ADDRESS_SIZE-1 downto 0);
61           line_scan_packet_in        : in std_logic_vector((
62                                        OBJECT_ADDRESS_SIZE +
63                                        LINE_NUMBER_ADDRESS_SIZE +
64                                        AOI_ADDRESS_SIZE +
65                                        AOI_ADDRESS_SIZE) - 1
66                                        downto 0);
67           hyperspectral_packet_out   : out std_logic_vector(
68                                        TRANSCEIVER_DATA_SIZE - 1
69                                        downto 0);
70           transciever_obj_wre_out    : out std_logic);
71  end entity;
72
73
74  architecture rtl of camera_relations is
75
76      signal s_linescan_offset       : std_logic_vector(
77                                           AOI_ADDRESS_SIZE - 1
78                                           downto 0);
79      signal s_line_number           : std_logic_vector(
80                                           LINE_NUMBER_ADDRESS_SIZE
81                                           - 1 downto 0) :=
82                                           (others => '0');
83      signal s_start_in              : std_logic_vector(
84                                           AOI_ADDRESS_SIZE - 1
85                                           downto 0) :=
86                                           (others => '0');
87      signal s_end_in                : std_logic_vector(
88                                           AOI_ADDRESS_SIZE - 1
89                                           downto 0) := (
90                                           others => '0');
91      signal s_object_number         : std_logic_vector(
92                                           OBJECT_ADDRESS_SIZE - 1
93                                           downto 0);
94      signal s_transciever_obj_wre   : std_logic;
95      signal s_hyperspectral_packet  : std_logic_vector(
96                                        TRANSCEIVER_DATA_SIZE - 1);
97
98
99      signal s_hyperspectral_line_number : unsigned(
100                                       LINE_NUMBER_ADDRESS_SIZE + 32
101                                       - 1 downto 0);
```

```
102     signal s_hyperspectral_start, s_hyperspectral_end : unsigned(
103                                     AOI_ADDRESS_SIZE + 32
104                                     downto 0);
105
106     constant c_zeros                : std_logic_vector(
107                                     TRANSCEIVER_DATA_SIZE - 1
108                                     downto
109                                     LINE_NUMBER_ADDRESS_SIZE +
110                                     OBJECT_ADDRESS_SIZE +
111                                     PIXEL_ADDRESS_SIZE +
112                                     PIXEL_ADDRESS_SIZE) :=
113                                     (others => '0');
114     constant c_ones                 : std_logic_vector(
115                                     TRANSCEIVER_DATA_SIZE - 1
116                                     downto
117                                     LINE_NUMBER_ADDRESS_SIZE +
118                                     OBJECT_ADDRESS_SIZE +
119                                     PIXEL_ADDRESS_SIZE +
120                                     PIXEL_ADDRESS_SIZE)
121                                     := (others => '1');
122     constant c_line_number_reset    : unsigned(
123                                     LINE_NUMBER_ADDRESS_SIZE +
124                                     32 - 1 downto 0) :=
125                                     (others => '0');
126     constant c_object_number_reset : std_logic_vector(
127                                     OBJECT_ADDRESS_SIZE - 1
128                                     downto 0):=(others => '0');
129     constant c_start_signal_reset   : std_logic_vector(
130                                     PIXEL_ADDRESS_SIZE - 1
131                                     downto 0):=(others => '0');
132     constant c_end_signal_reset     : std_logic_vector(
133                                     PIXEL_ADDRESS_SIZE - 1
134                                     downto 0):=(others => '1');
135
136     constant c_buffer_64_bits       : std_logic_vector(
137                                     TRANSCEIVER_DATA_SIZE -
138                                     LINE_NUMBER_ADDRESS_SIZE +
139                                     OBJECT_ADDRESS_SIZE +
140                                     PIXEL_ADDRESS_SIZE +
141                                     PIXEL_ADDRESS_SIZE - 1
142                                     downto 0):=(others => '0');
143
144 begin
145
146     registers : process(clk, rst_n)
147     begin
148         if(rst_n = '0') then
149             s_linescan_offset <= (others => '0');
```

```
150         elsif(rising_edge(clk)) then
151             s_linescan_offset <= linescan_offset;
152         end if;
153     end process;
154 
155     s_object_number <= line_scan_packet_in(OBJECT_ADDRESS_SIZE +
156                        LINE_NUMBER_ADDRESS_SIZE +
157                        AOI_ADDRESS_SIZE + AOI_ADDRESS_SIZE - 1
158                        downto LINE_NUMBER_ADDRESS_SIZE +
159                        AOI_ADDRESS_SIZE + AOI_ADDRESS_SIZE);
160     s_line_number   <= line_scan_packet_in(
161                        LINE_NUMBER_ADDRESS_SIZE +
162                        AOI_ADDRESS_SIZE + AOI_ADDRESS_SIZE - 1
163                        downto AOI_ADDRESS_SIZE +
164                        AOI_ADDRESS_SIZE);
165     s_start_in      <= line_scan_packet_in(AOI_ADDRESS_SIZE +
166                        AOI_ADDRESS_SIZE - 1 downto
167                        AOI_ADDRESS_SIZE);
168     s_end_in        <= line_scan_packet_in(AOI_ADDRESS_SIZE - 1
169                        downto 0);
170 
171     s_hyperspectral_line_number <= unsigned(s_line_number) *
172                                    unsigned(
173                                    LINE_TO_HYPER_LINE_RATIO);
174     s_hyperspectral_start       <= unsigned(signed('0' &
175                                    s_start_in) + signed(
176                                    s_linescan_offset)) *
177                                    unsigned(
178                                    LINE_TO_HYPER_PIXEL_RATIO);
179     s_hyperspectral_end         <= unsigned(signed('0' &s_end_in)
180                                    + signed(s_linescan_offset))
181                                    * unsigned(
182                                    LINE_TO_HYPER_PIXEL_RATIO);
183 
184 
185     --multiply integer by fraction, keep integer part
186     trans_packet_generation: process(clk, rst_n)
187     begin
188         if(rst_n = '0') then
189             s_transciever_obj_wre <= '0';
190             s_hyperspectral_packet <= (others => '0');
191         elsif(rising_edge(clk)) then
192             if(line_reset = '1') then
193                 s_hyperspectral_packet <= std_logic_vector(
194                                           s_hyperspectral_line_number(
195                                           LINE_NUMBER_ADDRESS_SIZE +
196                                           32-1 downto 32)) & c_ones &
197                                           c_object_number_reset &
```

```
198                                     c_start_signal_reset & c_
199                                     end_signal_reset;
200                 s_transciever_obj_wre    <= '1';
201             else
202                 -- negative offset, start pixel occurs before
203                 -- start of hyperspectral pixels
204                 if(s_linescan_offset(AOI_ADDRESS_SIZE-1) = '1'
205                    and signed('0'& s_start_in) < abs(signed(
206                    s_linescan_offset))) then
207                     s_transciever_obj_wre <= '0';
208                 -- positive offset, object outside of
209                 -- hyperspectral view
210                 elsif(s_hyperspectral_start(32+PIXEL_ADDRESS_SIZE
211                       downto 32) > to_unsigned(NUMBER_OF_PIXELS
212                       - 1,PIXEL_ADDRESS_SIZE+1) or
213                       s_hyperspectral_end( 32+ PIXEL_ADDRESS_SIZE
214                       downto 32) > to_unsigned(
215                       NUMBER_OF_PIXELS - 1,PIXEL_ADDRESS_SIZE+1))
216                       then
217                     s_transciever_obj_wre <= '0';
218                 else
219                     s_transciever_obj_wre <= '1';
220                 end if;
221                 s_hyperspectral_packet <= std_logic_vector(
222                                     s_hyperspectral_line_number(
223                                     LINE_NUMBER_ADDRESS_SIZE+32-
224                                     1 downto 32)) & c_zeros &
225                                     s_object_number &
226                                     std_logic_vector(
227                                     s_hyperspectral_start(32+
228                                     PIXEL_ADDRESS_SIZE-1
229                                     downto 32)) &
230                                     std_logic_vector(
231                                     s_hyperspectral_end(32+
232                                     PIXEL_ADDRESS_SIZE-1
233                                     downto 32));
234             end if;
235         end if;
236     end process;
237
238     -- Assign the output signals
239     transciever_obj_wre_out  <= s_transciever_obj_wre;
240     hyperspectral_packet_out <= s_hyperspectral_packet;
241
242 end architecture;
```

```
1 ------------------------------------------------------------------------
2 --
```

```
3  --! @file        mod5_calc_y_center.vhd
4  --! @brief       Compute the vertical center of the object.
5  --! @details
6  --
7  --! @author      Connor Dack
8  --! @date        November 2016
9  --! @copyright   Copyright (C) 2015 Ross K. Snider and Connor Dack
10 --
11 --   This program is free software: you can redistribute it
12 --   and/or modify it under the terms of the GNU General
13 --   Public License as published by the Free Software Foundation,
14 --   either version 3 of the License, or (at your option) any
15 --   later version.
16 --
17 --   This program is distributed in the hope that it will be
18 --   useful, but WITHOUT ANY WARRANTY; without even the implied
19 --   warranty of MERCHANTABILITY or FITNESS for A PARTICULAR
20 --   PURPOSE. See the GNU General Public License for more details.
21 --
22 --   You should have received a copy of the GNU General Public
23 --   License along with this program.  If not, see
24 --   <http://www.gnu.org/licenses/>.
25 --
26 --   Connor Dack
27 --   Electrical and Computer Engineering
28 --   Montana State University
29 --   610 Cobleigh Hall
30 --   Bozeman, MT 59717
31 --   connor.dack@msu.montana.edu
32 --
33 ------------------------------------------------------------------------------
34
35
36 LIBRARY IEEE;
37 USE IEEE.STD_LOGIC_1164.ALL;
38 USE IEEE.NUMERIC_STD.ALL;
39
40 USE work.USER_CONTROLS.ALL;
41
42 ------------------------------------------------------------------------------
43 --
44 --! @brief       mod5_calc_y_center
45 --! @details
46 --!
47 --! @param       clk             -  Input: System Clock
48 --! @param       rst_n           -  Input: Active low reset
49 --! @param       mod_5_trigger_in     -  Input: Enable Trigger
50 --! @param       line_scan_packet_in  -  Input: Object Number,
```

```
51 --                                          Current Line Number,
52 --                                          Start X, End X, Middle X
53 --! @param       object_y_data_out    - Output: Object Number,
54 --                                          First LN, Middle LN,
55 --                                          Last LN
56 --! @param       mod_5_wr_en_out      - Output: Write Enable
57 --
58 ----------------------------------------------------------------------------
59
60 LIBRARY IEEE;
61 USE IEEE.STD_LOGIC_1164.ALL;
62 USE IEEE.NUMERIC_STD.ALL;
63
64 use work.user_controls.all;
65
66 entity mod5_calc_y_center is
67   port(clk          :  in std_logic;
68        rst_n        :  in std_logic;
69           mod_5_trigger_in    :  in std_logic;
70           line_scan_packet_in :  in std_logic_vector(
71                                     OBJECT_ADDRESS_SIZE +
72                                     LINE_NUMBER_ADDRESS_SIZE +
73                                     (3 * AOI_ADDRESS_SIZE) - 1
74                                     downto 0);
75           object_y_data_out   : out std_logic_vector(
76                                     OBJECT_ADDRESS_SIZE + (3 *
77                                     LINE_NUMBER_ADDRESS_SIZE) - 1
78                                     downto 0);
79           mod_5_wr_en_out     : out std_logic);
80 end entity;
81
82 architecture rtl of mod5_calc_y_center is
83
84     type array_of_object_info is array(1 to MAX_OBJECT_NUMBER) of
85                                     std_logic_vector(
86                                     LINE_NUMBER_ADDRESS_SIZE - 1
87                                     downto 0);
88
89     signal s_ojbect_first_line : array_of_object_info :=
90                                     (others => (others => '1'));
91     signal s_object_last_line  : array_of_object_info :=
92                                     (others => (others => '0'));
93     signal s_completed_obj_num : std_logic_vector(
94                                      OBJECT_ADDRESS_SIZE - 1
95                                      downto 0) := (others => '0');
96     signal s_mod_5_trigger     : std_logic := '0';
97     signal s_intermediate_flag : std_logic := '0';
98     signal s_mod_5_wr_en       : std_logic := '0';
```

```
99      signal s_y_subtract         : std_logic_vector(
100                                   LINE_NUMBER_ADDRESS_SIZE - 1
101                                   downto 0) := (others => '0');
102     signal s_object_y_data      : std_logic_vector(
103                                   OBJECT_ADDRESS_SIZE + (3 *
104                                   LINE_NUMBER_ADDRESS_SIZE) - 1
105                                   downto 0) := (others => '0');
106
107     constant c_first_line_compare : std_logic_vector(
108                                     LINE_NUMBER_ADDRESS_SIZE - 1
109                                     downto 0) := (others => '1')
                                        ;
110     constant c_done_compare       : std_logic_vector((3 *
111                                     AOI_ADDRESS_SIZE) - 1
112                                     downto 0) := (others => '1');
113
114 begin
115     -- Assign the internal component signals
116     s_mod_5_trigger <= mod_5_trigger_in;
117
118     -- Compute the vertical subtraction using a process
119     compute_y_subtract : process(clk, rst_n)
120         begin
121             if(rst_n = '0') then -- On reset
122               s_ojbect_first_line <= (others => (others => '1'));
123               -- Set the first line array cells to all 1s
124               s_object_last_line  <= (others => (others => '0'));
125               -- Clear the last line array cells
126             elsif(rising_edge(clk)) then
127             -- On the rising edge of the component clock
128                 if(s_mod_5_trigger = '1') then
129                 -- If the Mod 5 enable flag high, then
130                     if(line_scan_packet_in((3 * AOI_ADDRESS_SIZE)
131                             - 1 downto 0) = c_done_compare) then
132                     -- If the object is complete, then
133                         s_intermediate_flag <= '1';
134                         -- Set the intermediate ste plage
135                         s_y_subtract <= std_logic_vector(unsigned
136                                         (s_object_last_line(
137                                         to_integer(unsigned(
138                                         line_scan_packet_in(
139                                         OBJECT_ADDRESS_SIZE +
140                                         LINE_NUMBER_ADDRESS_SIZE
141                                         + (3 * AOI_ADDRESS_SIZE)
142                                         - 1 downto
143                                         LINE_NUMBER_ADDRESS_SIZE
144                                         + (3 * AOI_ADDRESS_SIZE))
145                                         )))) - unsigned(
```

```
146                                 s_ojbect_first_line(
147                                 to_integer(unsigned(
148                                 line_scan_packet_in(
149                                 OBJECT_ADDRESS_SIZE +
150                                 LINE_NUMBER_ADDRESS_SIZE
151                                 + (3 * AOI_ADDRESS_SIZE)
152                                 - 1 downto
153                                 LINE_NUMBER_ADDRESS_SIZE
154                                 + (3 * AOI_ADDRESS_SIZE))
155                                 ))))));
156                     -- Compute the subtraction between the
157                     -- last line value and the first line
158                     -- value
159                     s_completed_obj_num <=line_scan_packet_in
160                                 (OBJECT_ADDRESS_SIZE +
161                                 LINE_NUMBER_ADDRESS_SIZE +
162                                 (3 * AOI_ADDRESS_SIZE) - 1
163                                 downto
164                                 LINE_NUMBER_ADDRESS_SIZE +
165                                 (3 * AOI_ADDRESS_SIZE));
166                     -- Register the object number
167                 elsif(s_ojbect_first_line(to_integer(unsigned
168                      (line_scan_packet_in(OBJECT_ADDRESS_SIZE
169                      + LINE_NUMBER_ADDRESS_SIZE + (3 *
170                      AOI_ADDRESS_SIZE) - 1 downto
171                      LINE_NUMBER_ADDRESS_SIZE + (3 *
172                      AOI_ADDRESS_SIZE))))) =
173                      c_first_line_compare) then
174                 -- If the first line value at the object
175                 -- number is all ones, then
176                     s_ojbect_first_line(to_integer(unsigned(
177                     line_scan_packet_in(OBJECT_ADDRESS_SIZE +
178                     LINE_NUMBER_ADDRESS_SIZE + (3 *
179                     AOI_ADDRESS_SIZE) - 1 downto
180                     LINE_NUMBER_ADDRESS_SIZE + (3 *
181                     AOI_ADDRESS_SIZE)))))) <=
182                             line_scan_packet_in(
183                             LINE_NUMBER_ADDRESS_SIZE + (3 *
184                             AOI_ADDRESS_SIZE) - 1 downto (3 *
185                             AOI_ADDRESS_SIZE));
186                 -- Save the current line number in the first
187                 -- line array at the object number
188                     s_y_subtract <= (others => '0');
189                 else -- Otherwise
190                     s_object_last_line(to_integer(unsigned(
191                     line_scan_packet_in(OBJECT_ADDRESS_SIZE +
192                     LINE_NUMBER_ADDRESS_SIZE + (3 *
193                     AOI_ADDRESS_SIZE) - 1 downto
```

```
194                         LINE_NUMBER_ADDRESS_SIZE + (3 *
195                         AOI_ADDRESS_SIZE)))) <=
196                         line_scan_packet_in(
197                         LINE_NUMBER_ADDRESS_SIZE + (3 *
198                         AOI_ADDRESS_SIZE) - 1 downto (3 *
199                         AOI_ADDRESS_SIZE));
200                     -- Save the object number to the last line
201                     -- array at the object number
202                         s_y_subtract <= (others => '0');
203                     end if;
204                 end if;
205                 if(s_mod_5_wr_en = '1') then
206                 -- If the Mod 5 write enable is a one, then
207                     s_intermediate_flag <= '0';
208                     -- clear the intermediate flag
209                 end if;
210             end if;
211         end process;
212
213     -- Compute the vertical center and create the output packet
214     compute_y_center : process(clk, rst_n)
215         begin
216             if(rst_n = '0') then -- On reset,
217                 s_object_y_data <= (others => '0');
218                 -- clear the object packet
219             elsif(rising_edge(clk)) then
220             -- On the rising edge of the component clock,
221                 if(s_intermediate_flag = '1') then
222                 -- If the intermediate flag is set, then
223                     s_object_y_data  <= s_completed_obj_num &
224                                         s_ojbect_first_line(
225                                         to_integer(unsigned(
226                                         s_completed_obj_num)))
227                                         & std_logic_vector(
228                                         unsigned(
229                                         s_ojbect_first_line(
230                                         to_integer(unsigned(
231                                         s_completed_obj_num)))
232                                         ) + unsigned('0' &
233                                         s_y_subtract(
234                                         LINE_NUMBER_ADDRESS_SIZE
235                                         - 1 downto 1))) &
236                                         s_object_last_line(
237                                         to_integer(unsigned(
238                                         s_completed_obj_num)));
239                     -- Create the object packet by combining the
240                     -- object number with the first line value,
241                     -- the vertical center, and the last line
```

```
242                     -- value
243                     s_mod_5_wr_en <= '1';
244                     -- Set the Mod 5 write enable
245                 else -- Otherwise,
246                      s_object_y_data <= (others => '0');
247                     s_mod_5_wr_en <= '0';
248                     -- Clear the Mod 5 write enable
249                 end if;
250             end if;
251         end process;
252 
253     -- Assign the outputs
254     object_y_data_out <= s_object_y_data;
255     mod_5_wr_en_out <= s_mod_5_wr_en;
256 
257 end architecture;
```

Line Scan Camera and Object Tracking MATLAB Code

```
%-------------------------------------------------------------------------
% Mod1_TB.m
%-------------------------------------------------------------------------
clear;
clc;
%-------------------------------------------------------------------------
% Import the Test Image
%-------------------------------------------------------------------------
fileName = 'Test_Image_2.bmp';
importfile(fileName);

%-------------------------------------------------------------------------
% Constants and Variables for the Test Bench
%-------------------------------------------------------------------------
number_of_pixels            = 2040;
number_of_pixels_divide_10 = 204;
number_of_lines             = 512;
Number_of_Fractional_Bits  = 0;
Signed_Or_Unsigned          = 0;
Input_Data_Size             = 8;
Target_Data_Size            = 10;

positive_threshold          = 250;
negative_threshold          = 0;
Difference_0 = zeros(number_of_lines,number_of_pixels_divide_10);
ErrorLocation1=zeros(number_of_lines,number_of_pixels_divide_10);

positive_threshold_fixed = fi(positive_threshold,              ...
                          Signed_Or_Unsigned, Input_Data_Size,...
                          Number_of_Fractional_Bits);
negative_threshold_fixed = fi(negative_threshold,              ...
                          Signed_Or_Unsigned, Input_Data_Size,...
                          Number_of_Fractional_Bits);

Data_In_0 = fi(zeros(number_of_lines,                          ...
             number_of_pixels_divide_10), Signed_Or_Unsigned, ...
             Input_Data_Size, Number_of_Fractional_Bits);
Data_In_1 = fi(zeros(number_of_lines,                          ...
             number_of_pixels_divide_10), Signed_Or_Unsigned, ...
             Input_Data_Size, Number_of_Fractional_Bits);
Data_In_2 = fi(zeros(number_of_lines,                          ...
             number_of_pixels_divide_10), Signed_Or_Unsigned, ...
             Input_Data_Size, Number_of_Fractional_Bits);
Data_In_3 = fi(zeros(number_of_lines,                          ...
```

```
45              number_of_pixels_divide_10), Signed_Or_Unsigned, ...
46              Input_Data_Size, Number_of_Fractional_Bits);
47  Data_In_4 = fi(zeros(number_of_lines,                          ...
48              number_of_pixels_divide_10), Signed_Or_Unsigned, ...
49              Input_Data_Size, Number_of_Fractional_Bits);
50  Data_In_5 = fi(zeros(number_of_lines,                          ...
51              number_of_pixels_divide_10), Signed_Or_Unsigned, ...
52              Input_Data_Size, Number_of_Fractional_Bits);
53  Data_In_6 = fi(zeros(number_of_lines,                          ...
54              number_of_pixels_divide_10), Signed_Or_Unsigned, ...
55              Input_Data_Size, Number_of_Fractional_Bits);
56  Data_In_7 = fi(zeros(number_of_lines,                          ...
57              number_of_pixels_divide_10), Signed_Or_Unsigned, ...
58              Input_Data_Size, Number_of_Fractional_Bits);
59  Data_In_8 = fi(zeros(number_of_lines,                          ...
60              number_of_pixels_divide_10), Signed_Or_Unsigned, ...
61              Input_Data_Size, Number_of_Fractional_Bits);
62  Data_In_9 = fi(zeros(number_of_lines,                          ...
63              number_of_pixels_divide_10), Signed_Or_Unsigned, ...
64              Input_Data_Size, Number_of_Fractional_Bits);
65
66  result = fi(zeros(number_of_lines,number_of_pixels_divide_10),...
67             Signed_Or_Unsigned, Target_Data_Size,               ...
68             Number_of_Fractional_Bits);
69  result_matlab= zeros(number_of_lines,number_of_pixels_divide_10);
70  result_matlab_fi = fi(zeros(number_of_lines,                   ...
71                     number_of_pixels_divide_10),                ...
72                     Signed_Or_Unsigned, Target_Data_Size,       ...
73                     Number_of_Fractional_Bits);
74
75  test_image_recreate = zeros(number_of_lines , number_of_pixels);
76  k = 0;
77  %---Generate Fixed Point Values-----------------------------------
78  for i = 1:number_of_lines
79      for j = 1:10:number_of_pixels
80          k = k + 1;
81          Data_In_0(i,k) = fi(cdata(i,j + 0),Signed_Or_Unsigned,...
82                       Input_Data_Size, Number_of_Fractional_Bits);
83          Data_In_1(i,k) = fi(cdata(i,j + 1),Signed_Or_Unsigned,...
84                        Input_Data_Size, Number_of_Fractional_Bits);
85          Data_In_2(i,k) = fi(cdata(i,j + 2),Signed_Or_Unsigned,...
86                        Input_Data_Size, Number_of_Fractional_Bits);
87          Data_In_3(i,k) = fi(cdata(i,j + 3),Signed_Or_Unsigned,...
88                        Input_Data_Size, Number_of_Fractional_Bits);
89          Data_In_4(i,k) = fi(cdata(i,j + 4),Signed_Or_Unsigned,...
90                        Input_Data_Size, Number_of_Fractional_Bits);
91          Data_In_5(i,k) = fi(cdata(i,j + 5),Signed_Or_Unsigned,...
92                        Input_Data_Size, Number_of_Fractional_Bits);
```

```
93          Data_In_6(i,k) = fi(cdata(i,j + 6),Signed_Or_Unsigned,...
94                              Input_Data_Size, Number_of_Fractional_Bits);
95          Data_In_7(i,k) = fi(cdata(i,j + 7),Signed_Or_Unsigned,...
96                              Input_Data_Size, Number_of_Fractional_Bits);
97          Data_In_8(i,k) = fi(cdata(i,j + 8),Signed_Or_Unsigned,...
98                              Input_Data_Size, Number_of_Fractional_Bits);
99          Data_In_9(i,k) = fi(cdata(i,j + 9),Signed_Or_Unsigned,...
100                             Input_Data_Size, Number_of_Fractional_Bits);
101     end
102     k = 0;
103 end
104
105 save data.mat
106
107 %-------------------------------------------------------------------------
108 % HdlCosimulation System Object creation (this Matlab function
109 % was created by the cosimWizard).
110 %-------------------------------------------------------------------------
111 mod1_pixel_to_array_hdl = hdlcosim_mod1_pixel_to_array;
112
113 %---Simulate VHDL Code---------------------------------------------------
114 for i = 1:number_of_lines
115     for j = 1:1:number_of_pixels_divide_10
116         [data_packet_out_1] = step(mod1_pixel_to_array_hdl,    ...
117            positive_threshold_fixed, negative_threshold_fixed,...
118               Data_In_0(i,j), Data_In_1(i,j), Data_In_2(i,j), ...
119               Data_In_3(i,j), Data_In_4(i,j), Data_In_5(i,j), ...
120               Data_In_6(i,j), Data_In_7(i,j), Data_In_8(i,j), ...
121               Data_In_9(i,j));
122         result(i,j) = data_packet_out_1;
123     end
124 end
125
126 % %---MATLAB Results-----------------------------------------------------
127 start_location = 1;
128 for i = 1:number_of_lines
129     for j = start_location:1:number_of_pixels_divide_10
130         if(j == number_of_pixels_divide_10)
131             if(Data_In_0(i,j) > negative_threshold_fixed &&    ...
132                Data_In_0(i,j) < positive_threshold_fixed)
133                 result_matlab(i+1,1) = 512;
134             end
135             if(Data_In_1(i,j) > negative_threshold_fixed &&    ...
136                Data_In_1(i,j) < positive_threshold_fixed)
137                 result_matlab(i+1,1) = result_matlab(i+1,1) +256;
138             end
139             if(Data_In_2(i,j) > negative_threshold_fixed &&    ...
140                Data_In_2(i,j) < positive_threshold_fixed)
```

```
141                 result_matlab(i+1,1) = result_matlab(i+1,1) +128;
142             end
143             if(Data_In_3(i,j) > negative_threshold_fixed &&   ...
144                Data_In_3(i,j) < positive_threshold_fixed)
145                 result_matlab(i+1,1) = result_matlab(i+1,1) + 64;
146             end
147             if(Data_In_4(i,j) > negative_threshold_fixed &&   ...
148                Data_In_4(i,j) < positive_threshold_fixed)
149                 result_matlab(i+1,1) = result_matlab(i+1,1) + 32;
150             end
151             if(Data_In_5(i,j) > negative_threshold_fixed &&   ...
152                Data_In_5(i,j) < positive_threshold_fixed)
153                 result_matlab(i+1,1) = result_matlab(i+1,1) + 16;
154             end
155             if(Data_In_6(i,j) > negative_threshold_fixed &&   ...
156                Data_In_6(i,j) < positive_threshold_fixed)
157                 result_matlab(i+1,1) = result_matlab(i+1,1) + 8;
158             end
159             if(Data_In_7(i,j) > negative_threshold_fixed &&   ...
160                Data_In_7(i,j) < positive_threshold_fixed)
161                 result_matlab(i+1,1) = result_matlab(i+1,1) + 4;
162             end
163             if(Data_In_8(i,j) > negative_threshold_fixed &&   ...
164                Data_In_8(i,j) < positive_threshold_fixed)
165                 result_matlab(i+1,1) = result_matlab(i+1,1) + 2;
166             end
167             if(Data_In_9(i,j) > negative_threshold_fixed &&   ...
168                Data_In_9(i,j) < positive_threshold_fixed)
169                 result_matlab(i+1,1) = result_matlab(i+1,1) + 1;
170             end
171         else
172             if(Data_In_0(i,j) > negative_threshold_fixed &&   ...
173                Data_In_0(i,j) < positive_threshold_fixed)
174                 result_matlab(i,j+1) = 512;
175             end
176             if(Data_In_1(i,j) > negative_threshold_fixed &&   ...
177                Data_In_1(i,j) < positive_threshold_fixed)
178                 result_matlab(i,j+1) = result_matlab(i,j+1) +256;
179             end
180             if(Data_In_2(i,j) > negative_threshold_fixed &&   ...
181                Data_In_2(i,j) < positive_threshold_fixed)
182                 result_matlab(i,j+1) = result_matlab(i,j+1) +128;
183             end
184             if(Data_In_3(i,j) > negative_threshold_fixed &&   ...
185                Data_In_3(i,j) < positive_threshold_fixed)
186                 result_matlab(i,j+1) = result_matlab(i,j+1) + 64;
187             end
188             if(Data_In_4(i,j) > negative_threshold_fixed &&   ...
```

```
189                 Data_In_4(i,j) < positive_threshold_fixed)
190                  result_matlab(i,j+1) = result_matlab(i,j+1) + 32;
191             end
192             if(Data_In_5(i,j) > negative_threshold_fixed &&   ...
193                 Data_In_5(i,j) < positive_threshold_fixed)
194                  result_matlab(i,j+1) = result_matlab(i,j+1) + 16;
195             end
196             if(Data_In_6(i,j) > negative_threshold_fixed &&   ...
197                 Data_In_6(i,j) < positive_threshold_fixed)
198                  result_matlab(i,j+1) = result_matlab(i,j+1) + 8;
199             end
200             if(Data_In_7(i,j) > negative_threshold_fixed &&   ...
201                 Data_In_7(i,j) < positive_threshold_fixed)
202                  result_matlab(i,j+1) = result_matlab(i,j+1) + 4;
203             end
204             if(Data_In_8(i,j) > negative_threshold_fixed &&   ...
205                 Data_In_8(i,j) < positive_threshold_fixed)
206                  result_matlab(i,j+1) = result_matlab(i,j+1) + 2;
207             end
208             if(Data_In_9(i,j) > negative_threshold_fixed &&   ...
209                 Data_In_9(i,j) < positive_threshold_fixed)
210                  result_matlab(i,j+1) = result_matlab(i,j+1) + 1;
211             end
212         end
213     end
214 end
215
216 Error_Counter_1 = 0;
217
218 Simulated = sprintf('Simulated');
219 MATLAB = sprintf('MATLAB');
220 Difference = sprintf('Differnce,');
221 fprintf('%10s, %10s, %10s\n',Simulated,MATLAB,Difference);
222 for clki=1:1:number_of_lines
223     if(clki == 1)
224         start = 2;
225     else
226         start = 1;
227     end;
228     for counter = start:1:number_of_pixels_divide_10
229        Difference_0(clki, counter) =  result_matlab(clki,      ...
230                        counter) - double(result(clki, counter));
231
232        if(Difference_0(clki , counter) ~= 0)
233             ErrorLocation1(clki,counter) = Error_Counter_1;
234             fprintf('%10x, %10x, %10d, %10d\n', double(result(...
235             clki , counter)), double(result_matlab (clki ,     ...
236             counter)), Difference_0(clki , counter),           ...
```

```
237                 ErrorLocation1(clki,counter));
238                 fprintf('-------------------------------------\n');
239                 Error_Counter_1 = Error_Counter_1 + 1;
240             end
241
242         end
243     end
244
245     save test.mat
246     clear;
247     % clc;
248     load test.mat
```

```
1   %-------------------------------------------------------------------------
2   % Mod2_TB.m
3   %-------------------------------------------------------------------------
4   clear;
5   clc;
6   % load pixel_array.mat
7   %-------------------------------------------------------------------------
8   % Import the Test Image
9   %-------------------------------------------------------------------------
10  fileName = '..\..\..\TestImages\Test_Image_4.bmp';
11  importfile(fileName);
12
13  % ------------------------------------------------------------------------
14  % Constants and Variables for the Test Bench
15  % ------------------------------------------------------------------------
16  number_of_pixels           = 1000;
17  number_of_pixels_divide_10 = number_of_pixels / 10;
18  number_of_lines            = 512;
19  Number_of_Fractional_Bits  = 0;
20  Signed_Or_Unsigned         = 0;
21  Input_Data_Size            = 8;
22  Target_Data_Size           = 10;
23
24  Difference_0 = zeros(number_of_lines,number_of_pixels_divide_10);
25
26  Error_Location_1 = zeros(number_of_lines,                          ...
27                                      number_of_pixels_divide_10);
28
29  positive_threshold         = 250;
30  positive_threshold_fixed = fi(positive_threshold,                  ...
31                          Signed_Or_Unsigned, Input_Data_Size,  ...
32                          Number_of_Fractional_Bits);
33
34  negative_threshold         = 0;
35  negative_threshold_fixed = fi(negative_threshold,                  ...
```

```
36                          Signed_Or_Unsigned, Input_Data_Size, ...
37                          Number_of_Fractional_Bits);
38
39  Data_In_0 = fi(zeros(number_of_lines,                          ...
40                number_of_pixels_divide_10), Signed_Or_Unsigned,...
41                Input_Data_Size, Number_of_Fractional_Bits);
42  Data_In_1 = fi(zeros(number_of_lines,                          ...
43                number_of_pixels_divide_10), Signed_Or_Unsigned,...
44                Input_Data_Size, Number_of_Fractional_Bits);
45  Data_In_2 = fi(zeros(number_of_lines,                          ...
46                number_of_pixels_divide_10), Signed_Or_Unsigned,...
47                Input_Data_Size, Number_of_Fractional_Bits);
48  Data_In_3 = fi(zeros(number_of_lines,                          ...
49                number_of_pixels_divide_10), Signed_Or_Unsigned,...
50                Input_Data_Size, Number_of_Fractional_Bits);
51  Data_In_4 = fi(zeros(number_of_lines,                          ...
52                number_of_pixels_divide_10), Signed_Or_Unsigned,...
53                Input_Data_Size, Number_of_Fractional_Bits);
54  Data_In_5 = fi(zeros(number_of_lines,                          ...
55                number_of_pixels_divide_10), Signed_Or_Unsigned,...
56                Input_Data_Size, Number_of_Fractional_Bits);
57  Data_In_6 = fi(zeros(number_of_lines,                          ...
58                number_of_pixels_divide_10), Signed_Or_Unsigned,...
59                Input_Data_Size, Number_of_Fractional_Bits);
60  Data_In_7 = fi(zeros(number_of_lines,                          ...
61                number_of_pixels_divide_10), Signed_Or_Unsigned,...
62                Input_Data_Size, Number_of_Fractional_Bits);
63  Data_In_8 = fi(zeros(number_of_lines,                          ...
64                number_of_pixels_divide_10), Signed_Or_Unsigned,...
65                Input_Data_Size, Number_of_Fractional_Bits);
66  Data_In_9 = fi(zeros(number_of_lines,                          ...
67                number_of_pixels_divide_10), Signed_Or_Unsigned,...
68                Input_Data_Size, Number_of_Fractional_Bits);
69
70  result = fi(zeros(number_of_lines,number_of_pixels_divide_10),...
71                Signed_Or_Unsigned, Target_Data_Size,           ...
72                Number_of_Fractional_Bits);
73  result_matlab =zeros(number_of_lines,number_of_pixels_divide_10);
74  pixel_array_input_fi = fi(zeros(number_of_lines,               ...
75                number_of_pixels_divide_10), Signed_Or_Unsigned,...
76                Target_Data_Size, Number_of_Fractional_Bits);
77
78  result_ss1_start = fi(zeros(number_of_lines,                   ...
79                      number_of_pixels_divide_10),             ...
80                      Signed_Or_Unsigned, 4,                   ...
81                      Number_of_Fractional_Bits);
82  result_ss1_end   = fi(zeros(number_of_lines,                   ...
83                      number_of_pixels_divide_10),             ...
```

```
                    Signed_Or_Unsigned, 4,                          ...
                    Number_of_Fractional_Bits);
result_ss2_start = fi(zeros(number_of_lines,                        ...
                    number_of_pixels_divide_10),                    ...
                    Signed_Or_Unsigned, 4,                          ...
                    Number_of_Fractional_Bits);
result_ss2_end   = fi(zeros(number_of_lines,                        ...
                    number_of_pixels_divide_10),                    ...
                    Signed_Or_Unsigned, 4,                          ...
                    Number_of_Fractional_Bits);
result_ss_ptr    = fi(zeros(number_of_lines,                        ...
                    number_of_pixels_divide_10),                    ...
                    Signed_Or_Unsigned, 2,                          ...
                    Number_of_Fractional_Bits);
result_ss_ready  = fi(zeros(number_of_lines,                        ...
                    number_of_pixels_divide_10),                    ...
                    Signed_Or_Unsigned, 1,                          ...
                    Number_of_Fractional_Bits);
result_line_number = fi(zeros(number_of_lines,                      ...
                    number_of_pixels_divide_10),                    ...
                    Signed_Or_Unsigned, 16,                         ...
                    Number_of_Fractional_Bits);
result_line_rst    = fi(zeros(number_of_lines,                      ...
                    number_of_pixels_divide_10),                    ...
                    Signed_Or_Unsigned, 1,                          ...
                    Number_of_Fractional_Bits);
result_end_of_line = fi(zeros(number_of_lines,                      ...
                    number_of_pixels_divide_10),                    ...
                    Signed_Or_Unsigned, 1,                          ...
                    Number_of_Fractional_Bits);

test_image_recreate = zeros(number_of_lines , number_of_pixels);
k = 0;

%---Module 2 Inputs Generation-------------------------------------
for i = 1:number_of_lines
    for j = 1:10:number_of_pixels
        k = k + 1;
        Data_In_0(i,k) = fi(cdata(i,j + 0),Signed_Or_Unsigned,...
                    Input_Data_Size, Number_of_Fractional_Bits);
        Data_In_1(i,k) = fi(cdata(i,j + 1),Signed_Or_Unsigned,...
                    Input_Data_Size, Number_of_Fractional_Bits);
        Data_In_2(i,k) = fi(cdata(i,j + 2),Signed_Or_Unsigned,...
                    Input_Data_Size, Number_of_Fractional_Bits);
        Data_In_3(i,k) = fi(cdata(i,j + 3),Signed_Or_Unsigned,...
                    Input_Data_Size, Number_of_Fractional_Bits);
        Data_In_4(i,k) = fi(cdata(i,j + 4),Signed_Or_Unsigned,...
                    Input_Data_Size, Number_of_Fractional_Bits);
```

```
132         Data_In_5(i,k) = fi(cdata(i,j + 5),Signed_Or_Unsigned,...
133                         Input_Data_Size, Number_of_Fractional_Bits);
134         Data_In_6(i,k) = fi(cdata(i,j + 6),Signed_Or_Unsigned,...
135                         Input_Data_Size, Number_of_Fractional_Bits);
136         Data_In_7(i,k) = fi(cdata(i,j + 7),Signed_Or_Unsigned,...
137                         Input_Data_Size, Number_of_Fractional_Bits);
138         Data_In_8(i,k) = fi(cdata(i,j + 8),Signed_Or_Unsigned,...
139                         Input_Data_Size, Number_of_Fractional_Bits);
140         Data_In_9(i,k) = fi(cdata(i,j + 9),Signed_Or_Unsigned,...
141                         Input_Data_Size, Number_of_Fractional_Bits);
142     end
143     k = 0;
144 end
145
146 start_location = 1;
147 for i = 1:number_of_lines
148     for j = start_location:1:number_of_pixels_divide_10
149         if(j == number_of_pixels_divide_10)
150             if(Data_In_0(i,j) > negative_threshold_fixed &&   ...
151                         Data_In_0(i,j) < positive_threshold_fixed)
152                 result_matlab(i+1,1) = 512;
153             end
154             if(Data_In_1(i,j) > negative_threshold_fixed &&   ...
155                         Data_In_1(i,j) < positive_threshold_fixed)
156                 result_matlab(i+1,1) = result_matlab(i+1,1) +256;
157             end
158             if(Data_In_2(i,j) > negative_threshold_fixed &&   ...
159                         Data_In_2(i,j) < positive_threshold_fixed)
160                 result_matlab(i+1,1) = result_matlab(i+1,1) +128;
161             end
162             if(Data_In_3(i,j) > negative_threshold_fixed &&   ...
163                         Data_In_3(i,j) < positive_threshold_fixed)
164                 result_matlab(i+1,1) = result_matlab(i+1,1) + 64;
165             end
166             if(Data_In_4(i,j) > negative_threshold_fixed &&   ...
167                         Data_In_4(i,j) < positive_threshold_fixed)
168                 result_matlab(i+1,1) = result_matlab(i+1,1) + 32;
169             end
170             if(Data_In_5(i,j) > negative_threshold_fixed &&   ...
171                         Data_In_5(i,j) < positive_threshold_fixed)
172                 result_matlab(i+1,1) = result_matlab(i+1,1) + 16;
173             end
174             if(Data_In_6(i,j) > negative_threshold_fixed &&   ...
175                         Data_In_6(i,j) < positive_threshold_fixed)
176                 result_matlab(i+1,1) = result_matlab(i+1,1) + 8;
177             end
178             if(Data_In_7(i,j) > negative_threshold_fixed &&   ...
179                         Data_In_7(i,j) < positive_threshold_fixed)
```

```
180                 result_matlab(i+1,1) = result_matlab(i+1,1) + 4;
181             end
182             if(Data_In_8(i,j) > negative_threshold_fixed &&   ...
183                     Data_In_8(i,j) < positive_threshold_fixed)
184                 result_matlab(i+1,1) = result_matlab(i+1,1) + 2;
185             end
186             if(Data_In_9(i,j) > negative_threshold_fixed &&   ...
187                     Data_In_9(i,j) < positive_threshold_fixed)
188                 result_matlab(i+1,1) = result_matlab(i+1,1) + 1;
189             end
190         else
191             if(Data_In_0(i,j) > negative_threshold_fixed &&   ...
192                     Data_In_0(i,j) < positive_threshold_fixed)
193                 result_matlab(i,j+1) = 512;
194             end
195             if(Data_In_1(i,j) > negative_threshold_fixed &&   ...
196                     Data_In_1(i,j) < positive_threshold_fixed)
197                 result_matlab(i,j+1) = result_matlab(i,j+1) +256;
198             end
199             if(Data_In_2(i,j) > negative_threshold_fixed &&   ...
200                     Data_In_2(i,j) < positive_threshold_fixed)
201                 result_matlab(i,j+1) = result_matlab(i,j+1) +128;
202             end
203             if(Data_In_3(i,j) > negative_threshold_fixed &&   ...
204                     Data_In_3(i,j) < positive_threshold_fixed)
205                 result_matlab(i,j+1) = result_matlab(i,j+1) + 64;
206             end
207             if(Data_In_4(i,j) > negative_threshold_fixed &&   ...
208                     Data_In_4(i,j) < positive_threshold_fixed)
209                 result_matlab(i,j+1) = result_matlab(i,j+1) + 32;
210             end
211             if(Data_In_5(i,j) > negative_threshold_fixed &&   ...
212                     Data_In_5(i,j) < positive_threshold_fixed)
213                 result_matlab(i,j+1) = result_matlab(i,j+1) + 16;
214             end
215             if(Data_In_6(i,j) > negative_threshold_fixed &&   ...
216                     Data_In_6(i,j) < positive_threshold_fixed)
217                 result_matlab(i,j+1) = result_matlab(i,j+1) + 8;
218             end
219             if(Data_In_7(i,j) > negative_threshold_fixed &&   ...
220                     Data_In_7(i,j) < positive_threshold_fixed)
221                 result_matlab(i,j+1) = result_matlab(i,j+1) + 4;
222             end
223             if(Data_In_8(i,j) > negative_threshold_fixed &&   ...
224                     Data_In_8(i,j) < positive_threshold_fixed)
225                 result_matlab(i,j+1) = result_matlab(i,j+1) + 2;
226             end
227             if(Data_In_9(i,j) > negative_threshold_fixed &&   ...
```

```
                    Data_In_9(i,j) < positive_threshold_fixed)
                result_matlab(i,j+1) = result_matlab(i,j+1) + 1;
            end
        end
    end
end

save pixel_array.mat
s_transceiver_empty = fi(0, Signed_Or_Unsigned, 1,                ...
                                        Number_of_Fractional_Bits);
s_air_jet_fifo_empty = fi(0, Signed_Or_Unsigned, 1,               ...
                                        Number_of_Fractional_Bits);
s_enable = fi(1, Signed_Or_Unsigned,1,Number_of_Fractional_Bits);
pixel_array_input_fi = fi(result_matlab, Signed_Or_Unsigned,   ...
                    Target_Data_Size, Number_of_Fractional_Bits);
%-------------------------------------------------------------------
% HdlCosimulation System Object creation (this Matlab function
% was created by the cosimWizard).
%-------------------------------------------------------------------
mod2_tap_inhdl = hdlcosim_mod2_tap_in;

%---Simulate VHDL Code-----------------------------------------------
for i = 1:number_of_lines
    for j = 1:1:number_of_pixels_divide_10
        [ss1_start, ss1_end, ss2_start, ss2_end, ss_ptr, ...
        ss_ready, line_number, line_rst, end_of_line] = step(...
        mod2_tap_inhdl, s_enable, pixel_array_input_fi(i,j),...
        s_transceiver_empty, s_air_jet_fifo_empty);
        result_ss1_start(i,j)    = ss1_start;
        result_ss1_end(i,j)      = ss1_end;
        result_ss2_start(i,j)    = ss2_start;
        result_ss2_end(i,j)      = ss2_end;
        result_ss_ptr(i,j)       = ss_ptr;
        result_ss_ready(i,j)     = ss_ready;
        result_line_number(i,j) = line_number;
        result_line_rst(i,j)     = line_rst;
        result_end_of_line(i,j) = end_of_line;
    end
end

%---Save the results of the simulation to be used by Mod 3--------
save mod2_results.mat
clear;
load mod2_results.mat
```

```
%-------------------------------------------------------------------
% Mod3_TB.m
%-------------------------------------------------------------------
```

```
clear;
clc;
load ../mod2_tap_in/mod2_results.mat %Load the results from Mod 2

mod2_ss1_start = fi(zeros(number_of_lines ,                    ...
                    number_of_pixels_divide_10),               ...
                    Signed_Or_Unsigned, Target_Data_Size,      ...
                    Number_of_Fractional_Bits);
mod2_ss1_end   = fi(zeros(number_of_lines ,                    ...
                    number_of_pixels_divide_10),               ...
                    Signed_Or_Unsigned, Target_Data_Size,      ...
                    Number_of_Fractional_Bits);
mod2_ss2_start = fi(zeros(number_of_lines ,                    ...
                    number_of_pixels_divide_10),               ...
                    Signed_Or_Unsigned, Target_Data_Size,      ...
                    Number_of_Fractional_Bits);
mod2_ss2_end   = fi(zeros(number_of_lines ,                    ...
                    number_of_pixels_divide_10),               ...
                    Signed_Or_Unsigned, Target_Data_Size,      ...
                    Number_of_Fractional_Bits);
mod2_ss_ptr    = fi(zeros(number_of_lines ,                    ...
                    number_of_pixels_divide_10),               ...
                    Signed_Or_Unsigned, 2,                     ...
                    Number_of_Fractional_Bits);
mod2_ss_ready  = fi(zeros(number_of_lines ,                    ...
                    number_of_pixels_divide_10),               ...
                    Signed_Or_Unsigned, 1,                     ...
                    Number_of_Fractional_Bits);
mod2_line_number = fi(zeros(number_of_lines ,                  ...
                    number_of_pixels_divide_10),               ...
                    Signed_Or_Unsigned, 16,                    ...
                    Number_of_Fractional_Bits);
mod2_line_rst    = fi(zeros(number_of_lines ,                  ...
                    number_of_pixels_divide_10),               ...
                    Signed_Or_Unsigned, 1,                     ...
                    Number_of_Fractional_Bits);
mod2_end_of_line = fi(zeros(number_of_lines ,                  ...
                    number_of_pixels_divide_10),               ...
                    Signed_Or_Unsigned, 1,                     ...
                    Number_of_Fractional_Bits);

mod2_ss1_start = result_ss1_start;
mod2_ss1_end   = result_ss1_end;
mod2_ss2_start = result_ss2_start;
mod2_ss2_end   = result_ss2_end;
mod2_ss_ptr    = result_ss_ptr;
mod2_ss_ready  = result_ss_ready;
mod2_line_number = result_line_number;
```

```
52 mod2_line_rst     = result_line_rst;
53 mod2_end_of_line = result_end_of_line;
54
55
56 mod3_seg_start_out = fi(zeros(number_of_lines ,                    ...
57                         number_of_pixels_divide_10 ,               ...
58                         number_of_pixels_divide_10),               ...
59                         Signed_Or_Unsigned, 8,                     ...
60                         Number_of_Fractional_Bits);
61 mod3_seg_mid_out    = fi(zeros(number_of_lines ,                   ...
62                         number_of_pixels_divide_10 ,               ...
63                         number_of_pixels_divide_10),               ...
64                         Signed_Or_Unsigned, 8,                     ...
65                         Number_of_Fractional_Bits);
66 mod3_seg_out_out    = fi(zeros(number_of_lines ,                   ...
67                         number_of_pixels_divide_10 ,               ...
68                         number_of_pixels_divide_10),               ...
69                         Signed_Or_Unsigned, 8,                     ...
70                         Number_of_Fractional_Bits);
71 seg_ptr_out    = fi(zeros(number_of_lines ,                        ...
72                    number_of_pixels_divide_10),                    ...
73                    Signed_Or_Unsigned, Target_Data_Size,           ...
74                    Number_of_Fractional_Bits);
75 eol_out        = fi(zeros(number_of_lines ,                        ...
76                    number_of_pixels_divide_10),                    ...
77                    Signed_Or_Unsigned, 1,                          ...
78                    Number_of_Fractional_Bits);
79
80 mod3_ss2seg_hdl = hdlcosim_mod3_ss2seg;
81
82 %---Simulate VHDL Code-------------------------------------------
83 for i = 1:number_of_lines
84     for j = 1:1:number_of_pixels_divide_10
85         [seg_start_out, seg_mid_out, seg_end_out,   ...
86     seg_ptr_out(i,j)] = step(mod3_ss2seg_hdl,   ...
87     mod2_ss_ready(i,j), mod2_ss_ptr(i,j),       ...
88     mod2_ss1_start(i,j), mod2_ss1_end(i,j),     ...
89     mod2_ss2_start(i,j), mod2_ss2_end(i,j),     ...
90     mod2_line_number(i,j), mod2_line_rst(i,j),  ...
91     mod2_end_of_line(i,j));
92         mod3_seg_start_out(i,j,:) = seg_start_out;
93         mod3_seg_mid_out(i,j,:)   = seg_mid_out;
94         mod3_seg_out_out(i,j,:)   = seg_end_out;
95     end
96     i
97 end
98
99 %---Save the results of the simulation to be used by Mod 4-------
```

```
save mod3_results.mat
clear;
load mod3_results.mat
```

```
%-------------------------------------------------------------------------
% Mod4_TB.m
%-------------------------------------------------------------------------
clear;
clc;
%---Import the data from Module 2-----------------------------------------
load ../mod2_tap_in/mod2_results.mat

%---Allocate Memory Space-------------------------------------------------
mod2_ss1_start = fi(zeros(number_of_lines,                          ...
                    number_of_pixels_divide_10),                    ...
                    Signed_Or_Unsigned,                             ...
                    Target_Data_Size,                               ...
                    Number_of_Fractional_Bits);
mod2_ss1_end   = fi(zeros(number_of_lines,                          ...
                    number_of_pixels_divide_10),                    ...
                    Signed_Or_Unsigned,                             ...
                    Target_Data_Size,                               ...
                    Number_of_Fractional_Bits);
mod2_ss2_start = fi(zeros(number_of_lines,                          ...
                    number_of_pixels_divide_10),                    ...
                    Signed_Or_Unsigned,                             ...
                    Target_Data_Size,                               ...
                    Number_of_Fractional_Bits);
mod2_ss2_end   = fi(zeros(number_of_lines,                          ...
                    number_of_pixels_divide_10),                    ...
                    Signed_Or_Unsigned,                             ...
                    Target_Data_Size,                               ...
                    Number_of_Fractional_Bits);
mod2_ss_ptr    = fi(zeros(number_of_lines,                          ...
                    number_of_pixels_divide_10),                    ...
                    Signed_Or_Unsigned, 2,                          ...
                    Number_of_Fractional_Bits);
mod2_ss_ready  = fi(zeros(number_of_lines,                          ...
                    number_of_pixels_divide_10),                    ...
                    Signed_Or_Unsigned, 1,                          ...
                    Number_of_Fractional_Bits);
mod2_line_number = fi(zeros(number_of_lines,                        ...
                    number_of_pixels_divide_10),                    ...
                    Signed_Or_Unsigned, 32,                         ...
                    Number_of_Fractional_Bits);
mod2_line_rst    = fi(zeros(number_of_lines,                        ...
                    number_of_pixels_divide_10),                    ...
                    Signed_Or_Unsigned, 1,                          ...
```

```
45                      Number_of_Fractional_Bits);
46  mod2_end_of_line = fi(zeros(number_of_lines,                    ...
47                      number_of_pixels_divide_10),                ...
48                      Signed_Or_Unsigned, 1,                      ...
49                      Number_of_Fractional_Bits);
50
51  mod2_ss1_start = result_ss1_start;
52  mod2_ss1_end   = result_ss1_end;
53  mod2_ss2_start = result_ss2_start;
54  mod2_ss2_end   = result_ss2_end;
55  mod2_ss_ptr    = result_ss_ptr;
56  mod2_ss_ready  = result_ss_ready;
57  mod2_line_number = result_line_number;
58  mod2_line_rst    = result_line_rst;
59  mod2_end_of_line = result_end_of_line;
60
61
62  mod3_seg_start_out = fi(zeros(number_of_lines,                  ...
63                          number_of_pixels_divide_10,             ...
64                          number_of_pixels_divide_10),            ...
65                          Signed_Or_Unsigned, 8,                  ...
66                          Number_of_Fractional_Bits);
67  mod3_seg_mid_out   = fi(zeros(number_of_lines,                  ...
68                          number_of_pixels_divide_10,             ...
69                          number_of_pixels_divide_10),            ...
70                          Signed_Or_Unsigned, 8,                  ...
71                          Number_of_Fractional_Bits);
72  mod3_seg_out_out   = fi(zeros(number_of_lines,                  ...
73                          number_of_pixels_divide_10,             ...
74                          number_of_pixels_divide_10),            ...
75                          Signed_Or_Unsigned, 8,                  ...
76                          Number_of_Fractional_Bits);
77  seg_ptr_out    = fi(zeros(number_of_lines,                      ...
78                          number_of_pixels_divide_10),            ...
79                          Signed_Or_Unsigned, Target_Data_Size,   ...
80                          Number_of_Fractional_Bits);
81  eol_out        = fi(zeros(number_of_lines,                      ...
82                          number_of_pixels_divide_10),            ...
83                          Signed_Or_Unsigned, 1,                  ...
84                          Number_of_Fractional_Bits);
85
86  mod3_mod4_seg2obj_hdl = hdlcosim_mod3_mod4_wrapper;
87
88  OBJECT_ADDRESS_SIZE = 6;
89  AOI_ADDRESS_SIZE    = 8;
90
91  mod_5_trigger_out    = fi(zeros(number_of_lines,                ...
92                          number_of_pixels_divide_10),            ...
```

```
                          Signed_Or_Unsigned, 1,                    ...
                          Number_of_Fractional_Bits);
line_scan_packet_out = fi(zeros(number_of_lines,                    ...
                          number_of_pixels_divide_10),              ...
                          Signed_Or_Unsigned,                       ...
                          OBJECT_ADDRESS_SIZE + 32 +                ...
                          AOI_ADDRESS_SIZE + AOI_ADDRESS_SIZE,      ...
                          Number_of_Fractional_Bits);
mod_4_wr_en_out      = fi(zeros(number_of_lines,                    ...
                          number_of_pixels_divide_10),              ...
                          Signed_Or_Unsigned, 1,                    ...
                          Number_of_Fractional_Bits);

%---Simulate VHDL Code-------------------------------------------------
for i = 1:number_of_lines
    for j = 1:1:number_of_pixels_divide_10
        [mod_5_trigger_out(i,j), line_scan_packet_out(i,j),         ...
         mod_4_wr_en_out(i,j)] = step(mod3_mod4_seg2obj_hdl,        ...
         mod2_ss_ready(i,j), mod2_ss_ptr(i,j),                      ...
         mod2_ss1_start(i,j), mod2_ss1_end(i,j),                    ...
         mod2_ss2_start(i,j), mod2_ss2_end(i,j),                    ...
         mod2_line_number(i,j), mod2_line_rst(i,j),                 ...
         mod2_end_of_line(i,j));

    end
end

%---Save the Results to be used in Mod 5-------------------------------
save mod4_results.mat
clear;
load mod4_results.mat
```

```
%-------------------------------------------------------------------------
% Mod5_TB.m
%-------------------------------------------------------------------------

clear;
clc;

%---Import the results of Module 4-------------------------------------
load ../mod4_seg2obj/mod4_results.mat

mod5_calc_y_center_hdl = hdlcosim_mod5_calc_y_center;

object_y_data_out = fi(zeros(number_of_lines,                       ...
                          number_of_pixels_divide_10,               ...
                          number_of_pixels_divide_10),              ...
                          Signed_Or_Unsigned, 54,                   ...
```

```
                        Number_of_Fractional_Bits);
mod_5_wr_en_out    = fi(zeros(number_of_lines,                    ...
                        number_of_pixels_divide_10,               ...
                        number_of_pixels_divide_10),              ...
                        Signed_Or_Unsigned, 1,                    ...
                        Number_of_Fractional_Bits);

%---Simulate VHDL Code-------------------------------------------------
for i = 1:number_of_lines
    for j = 1:1:number_of_pixels_divide_10
       [object_y_data_out(i,j),mod_5_wr_en_out(i,j)] =            ...
          step(mod5_calc_y_center_hdl, mod_5_trigger_out(i,j),...
          line_scan_packet_out(i,j));
    end
end

%---Save the results---------------------------------------------------
save mod5_results.mat
clear;
load mod5_results.mat
```

```
%-------------------------------------------------------------------------
% Object_Tracking_TB.m
%-------------------------------------------------------------------------
clear;
clc;
% load pixel_array.mat
%-------------------------------------------------------------------------
% Import the Test Image
%-------------------------------------------------------------------------
fileName = '..\..\..\TestImages\Test_Image_5.bmp';
importfile(fileName);

% ------------------------------------------------------------------------
% Constants and Variables for the Test Bench
% ------------------------------------------------------------------------
number_of_pixels           = 1000;
number_of_pixels_divide_10 = number_of_pixels / 10;
number_of_lines            = 512;
Number_of_Fractional_Bits  = 0;
Signed_Or_Unsigned         = 0;
Input_Data_Size            = 8;
Target_Data_Size           = 10;

Difference_0 = zeros(number_of_lines,number_of_pixels_divide_10);

Error_Location_1 = zeros(number_of_lines,                         ...
                                    number_of_pixels_divide_10);
```

```
28
29 positive_threshold         = 250;
30 positive_threshold_fixed = fi(positive_threshold,                ...
31                          Signed_Or_Unsigned, Input_Data_Size,  ...
32                          Number_of_Fractional_Bits);
33
34 negative_threshold         = 0;
35 negative_threshold_fixed = fi(negative_threshold,                ...
36                          Signed_Or_Unsigned, Input_Data_Size,  ...
37                          Number_of_Fractional_Bits);
38
39 Data_In_0 = fi(zeros(number_of_lines,                            ...
40               number_of_pixels_divide_10), Signed_Or_Unsigned,...
41               Input_Data_Size, Number_of_Fractional_Bits);
42 Data_In_1 = fi(zeros(number_of_lines,                            ...
43               number_of_pixels_divide_10), Signed_Or_Unsigned,...
44               Input_Data_Size, Number_of_Fractional_Bits);
45 Data_In_2 = fi(zeros(number_of_lines,                            ...
46               number_of_pixels_divide_10), Signed_Or_Unsigned,...
47               Input_Data_Size, Number_of_Fractional_Bits);
48 Data_In_3 = fi(zeros(number_of_lines,                            ...
49               number_of_pixels_divide_10), Signed_Or_Unsigned,...
50               Input_Data_Size, Number_of_Fractional_Bits);
51 Data_In_4 = fi(zeros(number_of_lines,                            ...
52               number_of_pixels_divide_10), Signed_Or_Unsigned,...
53               Input_Data_Size, Number_of_Fractional_Bits);
54 Data_In_5 = fi(zeros(number_of_lines,                            ...
55               number_of_pixels_divide_10), Signed_Or_Unsigned,...
56               Input_Data_Size, Number_of_Fractional_Bits);
57 Data_In_6 = fi(zeros(number_of_lines,                            ...
58               number_of_pixels_divide_10), Signed_Or_Unsigned,...
59               Input_Data_Size, Number_of_Fractional_Bits);
60 Data_In_7 = fi(zeros(number_of_lines,                            ...
61               number_of_pixels_divide_10), Signed_Or_Unsigned,...
62               Input_Data_Size, Number_of_Fractional_Bits);
63 Data_In_8 = fi(zeros(number_of_lines,                            ...
64               number_of_pixels_divide_10), Signed_Or_Unsigned,...
65               Input_Data_Size, Number_of_Fractional_Bits);
66 Data_In_9 = fi(zeros(number_of_lines,                            ...
67               number_of_pixels_divide_10), Signed_Or_Unsigned,...
68               Input_Data_Size, Number_of_Fractional_Bits);
69
70 result = fi(zeros(number_of_lines,number_of_pixels_divide_10),...
71               Signed_Or_Unsigned, Target_Data_Size,             ...
72               Number_of_Fractional_Bits);
73 result_matlab =zeros(number_of_lines,number_of_pixels_divide_10);
74 pixel_array_input_fi = fi(zeros(number_of_lines,                 ...
75               number_of_pixels_divide_10), Signed_Or_Unsigned,...
```

```
76                 Target_Data_Size, Number_of_Fractional_Bits);
77
78  result_ss1_start = fi(zeros(number_of_lines,                    ...
79                        number_of_pixels_divide_10),              ...
80                        Signed_Or_Unsigned, 4,                    ...
81                        Number_of_Fractional_Bits);
82  result_ss1_end   = fi(zeros(number_of_lines,                    ...
83                        number_of_pixels_divide_10),              ...
84                        Signed_Or_Unsigned, 4,                    ...
85                        Number_of_Fractional_Bits);
86  result_ss2_start = fi(zeros(number_of_lines,                    ...
87                        number_of_pixels_divide_10),              ...
88                        Signed_Or_Unsigned, 4,                    ...
89                        Number_of_Fractional_Bits);
90  result_ss2_end   = fi(zeros(number_of_lines,                    ...
91                        number_of_pixels_divide_10),              ...
92                        Signed_Or_Unsigned, 4,                    ...
93                        Number_of_Fractional_Bits);
94  result_ss_ptr    = fi(zeros(number_of_lines,                    ...
95                        number_of_pixels_divide_10),              ...
96                        Signed_Or_Unsigned, 2,                    ...
97                        Number_of_Fractional_Bits);
98  result_ss_ready  = fi(zeros(number_of_lines,                    ...
99                        number_of_pixels_divide_10),              ...
100                       Signed_Or_Unsigned, 1,                    ...
101                       Number_of_Fractional_Bits);
102 result_line_number = fi(zeros(number_of_lines,                  ...
103                       number_of_pixels_divide_10),              ...
104                       Signed_Or_Unsigned, 16,                   ...
105                       Number_of_Fractional_Bits);
106 result_line_rst    = fi(zeros(number_of_lines,                  ...
107                       number_of_pixels_divide_10),              ...
108                       Signed_Or_Unsigned, 1,                    ...
109                       Number_of_Fractional_Bits);
110 result_end_of_line = fi(zeros(number_of_lines,                  ...
111                       number_of_pixels_divide_10),              ...
112                       Signed_Or_Unsigned, 1,                    ...
113                       Number_of_Fractional_Bits);
114
115 test_image_recreate = zeros(number_of_lines , number_of_pixels);
116 k = 0;
117
118 %--Module 2 Inputs Generation-------------------------------------
119 for i = 1:number_of_lines
120     for j = 1:10:number_of_pixels
121         k = k + 1;
122         Data_In_0(i,k) = fi(cdata(i,j + 0),Signed_Or_Unsigned,...
123                         Input_Data_Size, Number_of_Fractional_Bits);
```

```
124         Data_In_1(i,k) = fi(cdata(i,j + 1),Signed_Or_Unsigned,...
125                         Input_Data_Size, Number_of_Fractional_Bits);
126         Data_In_2(i,k) = fi(cdata(i,j + 2),Signed_Or_Unsigned,...
127                         Input_Data_Size, Number_of_Fractional_Bits);
128         Data_In_3(i,k) = fi(cdata(i,j + 3),Signed_Or_Unsigned,...
129                         Input_Data_Size, Number_of_Fractional_Bits);
130         Data_In_4(i,k) = fi(cdata(i,j + 4),Signed_Or_Unsigned,...
131                         Input_Data_Size, Number_of_Fractional_Bits);
132         Data_In_5(i,k) = fi(cdata(i,j + 5),Signed_Or_Unsigned,...
133                         Input_Data_Size, Number_of_Fractional_Bits);
134         Data_In_6(i,k) = fi(cdata(i,j + 6),Signed_Or_Unsigned,...
135                         Input_Data_Size, Number_of_Fractional_Bits);
136         Data_In_7(i,k) = fi(cdata(i,j + 7),Signed_Or_Unsigned,...
137                         Input_Data_Size, Number_of_Fractional_Bits);
138         Data_In_8(i,k) = fi(cdata(i,j + 8),Signed_Or_Unsigned,...
139                         Input_Data_Size, Number_of_Fractional_Bits);
140         Data_In_9(i,k) = fi(cdata(i,j + 9),Signed_Or_Unsigned,...
141                         Input_Data_Size, Number_of_Fractional_Bits);
142     end
143     k = 0;
144 end
145
146 start_location = 1;
147 for i = 1:number_of_lines
148     for j = start_location:1:number_of_pixels_divide_10
149         if(j == number_of_pixels_divide_10)
150             if(Data_In_0(i,j) > negative_threshold_fixed &&   ...
151                         Data_In_0(i,j) < positive_threshold_fixed)
152                 result_matlab(i+1,1) = 512;
153             end
154             if(Data_In_1(i,j) > negative_threshold_fixed &&   ...
155                         Data_In_1(i,j) < positive_threshold_fixed)
156                 result_matlab(i+1,1) = result_matlab(i+1,1) +256;
157             end
158             if(Data_In_2(i,j) > negative_threshold_fixed &&   ...
159                         Data_In_2(i,j) < positive_threshold_fixed)
160                 result_matlab(i+1,1) = result_matlab(i+1,1) +128;
161             end
162             if(Data_In_3(i,j) > negative_threshold_fixed &&   ...
163                         Data_In_3(i,j) < positive_threshold_fixed)
164                 result_matlab(i+1,1) = result_matlab(i+1,1) + 64;
165             end
166             if(Data_In_4(i,j) > negative_threshold_fixed &&   ...
167                         Data_In_4(i,j) < positive_threshold_fixed)
168                 result_matlab(i+1,1) = result_matlab(i+1,1) + 32;
169             end
170             if(Data_In_5(i,j) > negative_threshold_fixed &&   ...
171                         Data_In_5(i,j) < positive_threshold_fixed)
```

```
172                 result_matlab(i+1,1) = result_matlab(i+1,1) + 16;
173             end
174             if(Data_In_6(i,j) > negative_threshold_fixed &&   ...
175                     Data_In_6(i,j) < positive_threshold_fixed)
176                 result_matlab(i+1,1) = result_matlab(i+1,1) + 8;
177             end
178             if(Data_In_7(i,j) > negative_threshold_fixed &&   ...
179                     Data_In_7(i,j) < positive_threshold_fixed)
180                 result_matlab(i+1,1) = result_matlab(i+1,1) + 4;
181             end
182             if(Data_In_8(i,j) > negative_threshold_fixed &&   ...
183                     Data_In_8(i,j) < positive_threshold_fixed)
184                 result_matlab(i+1,1) = result_matlab(i+1,1) + 2;
185             end
186             if(Data_In_9(i,j) > negative_threshold_fixed &&   ...
187                     Data_In_9(i,j) < positive_threshold_fixed)
188                 result_matlab(i+1,1) = result_matlab(i+1,1) + 1;
189             end
190         else
191             if(Data_In_0(i,j) > negative_threshold_fixed &&   ...
192                     Data_In_0(i,j) < positive_threshold_fixed)
193                 result_matlab(i,j+1) = 512;
194             end
195             if(Data_In_1(i,j) > negative_threshold_fixed &&   ...
196                     Data_In_1(i,j) < positive_threshold_fixed)
197                 result_matlab(i,j+1) = result_matlab(i,j+1) +256;
198             end
199             if(Data_In_2(i,j) > negative_threshold_fixed &&   ...
200                     Data_In_2(i,j) < positive_threshold_fixed)
201                 result_matlab(i,j+1) = result_matlab(i,j+1) +128;
202             end
203             if(Data_In_3(i,j) > negative_threshold_fixed &&   ...
204                     Data_In_3(i,j) < positive_threshold_fixed)
205                 result_matlab(i,j+1) = result_matlab(i,j+1) + 64;
206             end
207             if(Data_In_4(i,j) > negative_threshold_fixed &&   ...
208                     Data_In_4(i,j) < positive_threshold_fixed)
209                 result_matlab(i,j+1) = result_matlab(i,j+1) + 32;
210             end
211             if(Data_In_5(i,j) > negative_threshold_fixed &&   ...
212                     Data_In_5(i,j) < positive_threshold_fixed)
213                 result_matlab(i,j+1) = result_matlab(i,j+1) + 16;
214             end
215             if(Data_In_6(i,j) > negative_threshold_fixed &&   ...
216                     Data_In_6(i,j) < positive_threshold_fixed)
217                 result_matlab(i,j+1) = result_matlab(i,j+1) + 8;
218             end
219             if(Data_In_7(i,j) > negative_threshold_fixed &&   ...
```

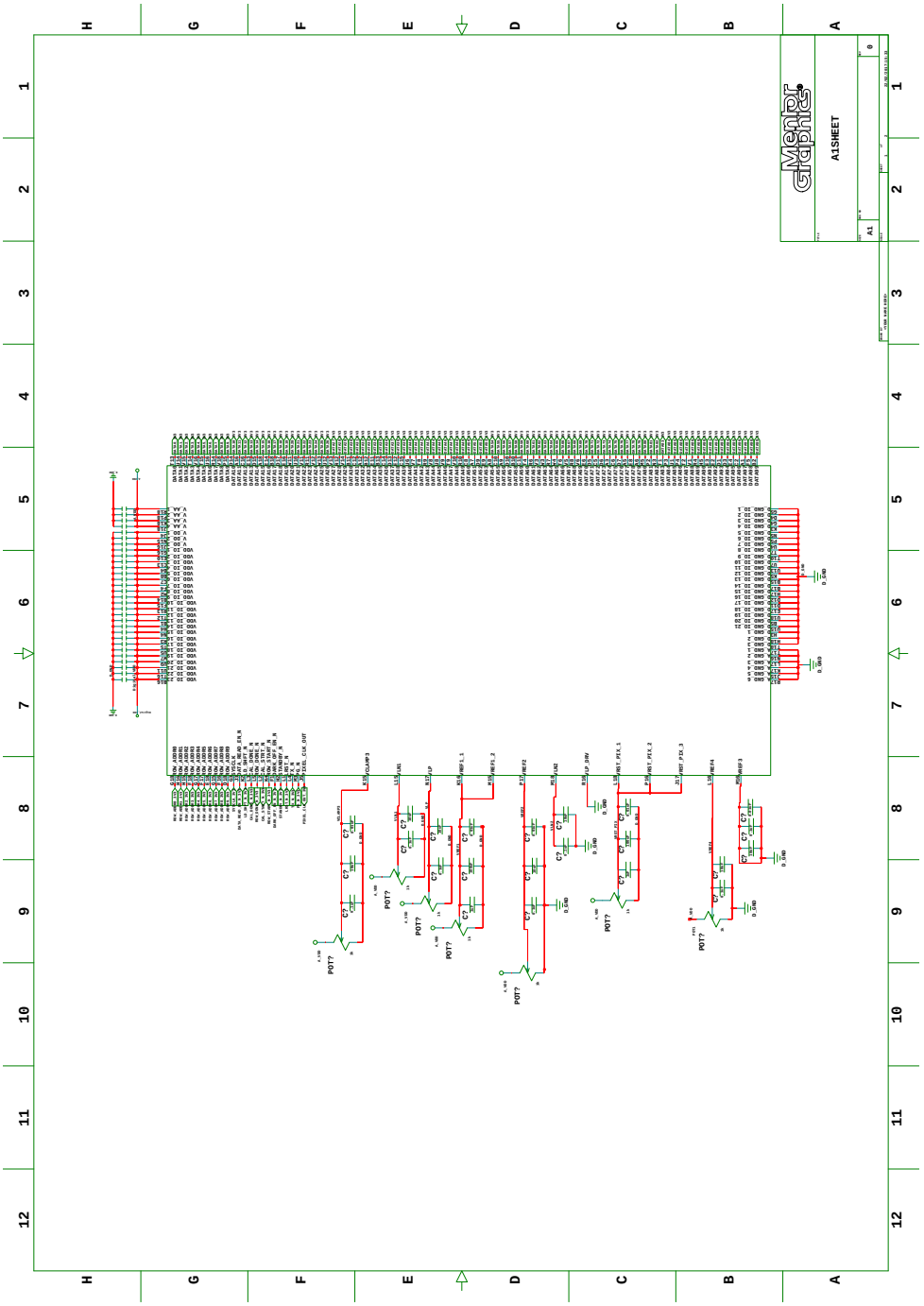
```
                    Data_In_7(i,j) < positive_threshold_fixed)
                result_matlab(i,j+1) = result_matlab(i,j+1) + 4;
            end
            if(Data_In_8(i,j) > negative_threshold_fixed &&   ...
                    Data_In_8(i,j) < positive_threshold_fixed)
                result_matlab(i,j+1) = result_matlab(i,j+1) + 2;
            end
            if(Data_In_9(i,j) > negative_threshold_fixed &&   ...
                    Data_In_9(i,j) < positive_threshold_fixed)
                result_matlab(i,j+1) = result_matlab(i,j+1) + 1;
            end
        end
    end
end

save pixel_array.mat
%---Other Inputs-----------------------------------------------------
data_valid_fi = fi(1, Signed_Or_Unsigned, 1,                    ...
                                        Number_of_Fractional_Bits);
s_transceiver_empty = fi(0, Signed_Or_Unsigned, 1,              ...
                                        Number_of_Fractional_Bits);
s_air_jet_fifo_empty = fi(0, Signed_Or_Unsigned, 1,             ...
                                        Number_of_Fractional_Bits);
s_enable = fi(1, Signed_Or_Unsigned,1,Number_of_Fractional_Bits);
linescan_offset_fi = fi(1, Signed_Or_Unsigned,10,               ...
                                        Number_of_Fractional_Bits);
pixel_array_input_fi = fi(result_matlab, Signed_Or_Unsigned,    ...
                    Target_Data_Size, Number_of_Fractional_Bits);

%---Allocate Memory for the Outputs----------------------------------
object_packet_out_fi = fi(zeros(number_of_lines,                ...
                         number_of_pixels_divide_10),           ...
                         Signed_Or_Unsigned, 64,                ...
                         Number_of_Fractional_Bits);
mod_5_wr_en_out_fi   = fi(zeros(number_of_lines,                ...
                         number_of_pixels_divide_10),           ...
                         Signed_Or_Unsigned, 1,                 ...
                         Number_of_Fractional_Bits);
air_jet_packet_out_fi= fi(zeros(number_of_lines,                ...
                         number_of_pixels_divide_10),           ...
                         Signed_Or_Unsigned, 6 + 10 + 16 ,      ...
                         Number_of_Fractional_Bits);
air_jet_wre_out_fi   = fi(zeros(number_of_lines,                ...
                         number_of_pixels_divide_10),           ...
                         Signed_Or_Unsigned, 1,                 ...
                         Number_of_Fractional_Bits);
current_line_out_fi  = fi(zeros(number_of_lines,                ...
                         number_of_pixels_divide_10),           ...
```

```
268                         Signed_Or_Unsigned, 16,                 ...
269                         Number_of_Fractional_Bits);
270
271 %-------------------------------------------------------------------
272 % HdlCosimulation System Object creation (this Matlab function
273 % was created by the cosimWizard).
274 %-------------------------------------------------------------------
275 object_tracking_inhdl = hdlcosim_object_tracking;
276
277 %--Simulate VHDL Code-----------------------------------------------
278 for i = 1:number_of_lines
279     for j = 1:1:number_of_pixels_divide_10
280         [object_packet_out_fi(i,j), mod_5_wr_en_out_fi(i,j),   ...
281          air_jet_packet_out_fi(i,j), air_jet_wre_out_fi(i,j), ...
282          current_line_out_fi(i,j)] =                           ...
283                step(object_tracking_inhdl, data_valid_fi,     ...
284                pixel_array_input_fi(i,j), s_transceiver_empty,...
285                s_air_jet_fifo_empty, linescan_offset_fi);
286     end
287 end
288
289
290 save object_tracking_results.mat
291 clear;
292 load object_tracking_results.mat
```

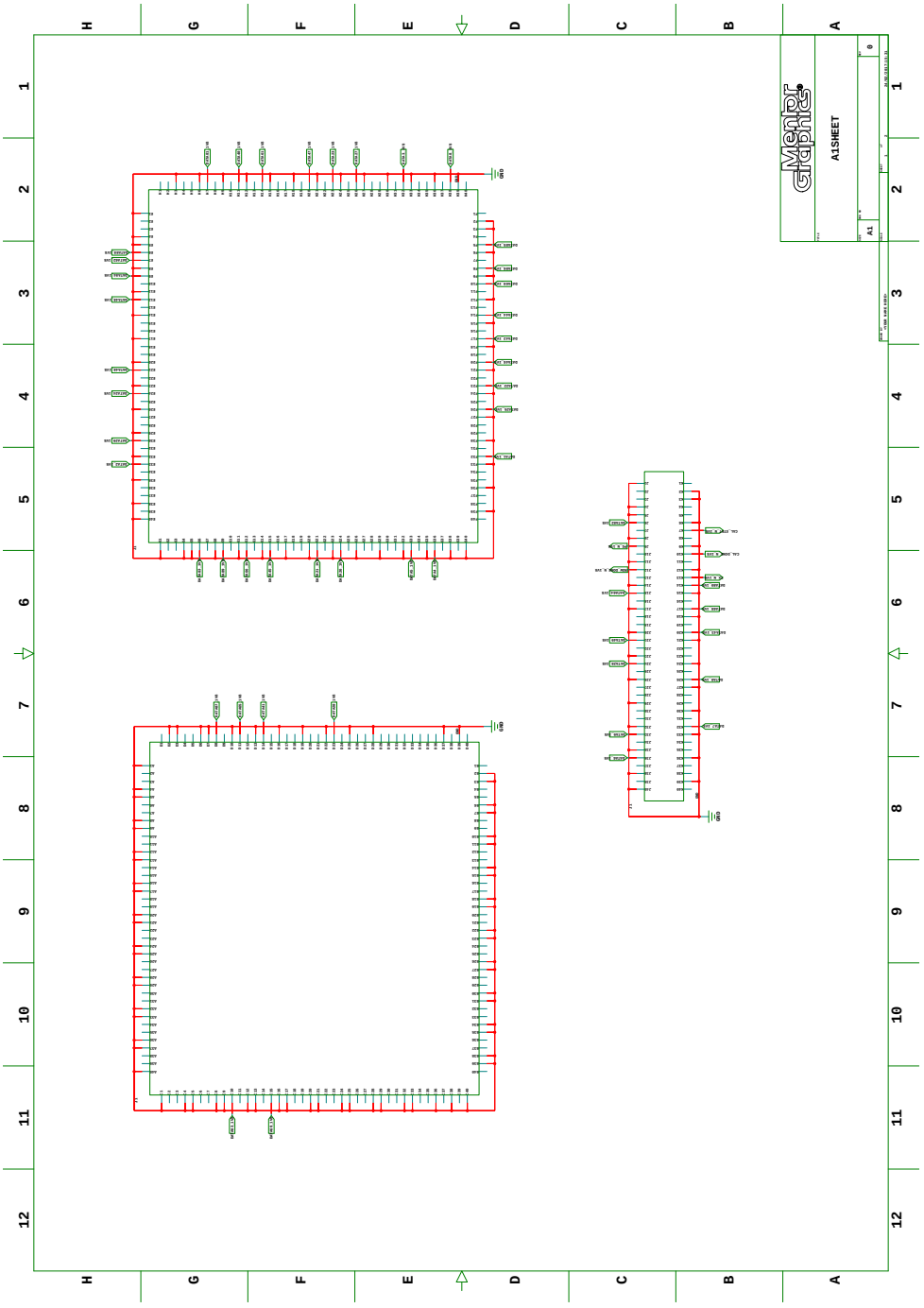
APPENDIX C

ARRIA 10 SOM IMAGE SENSOR

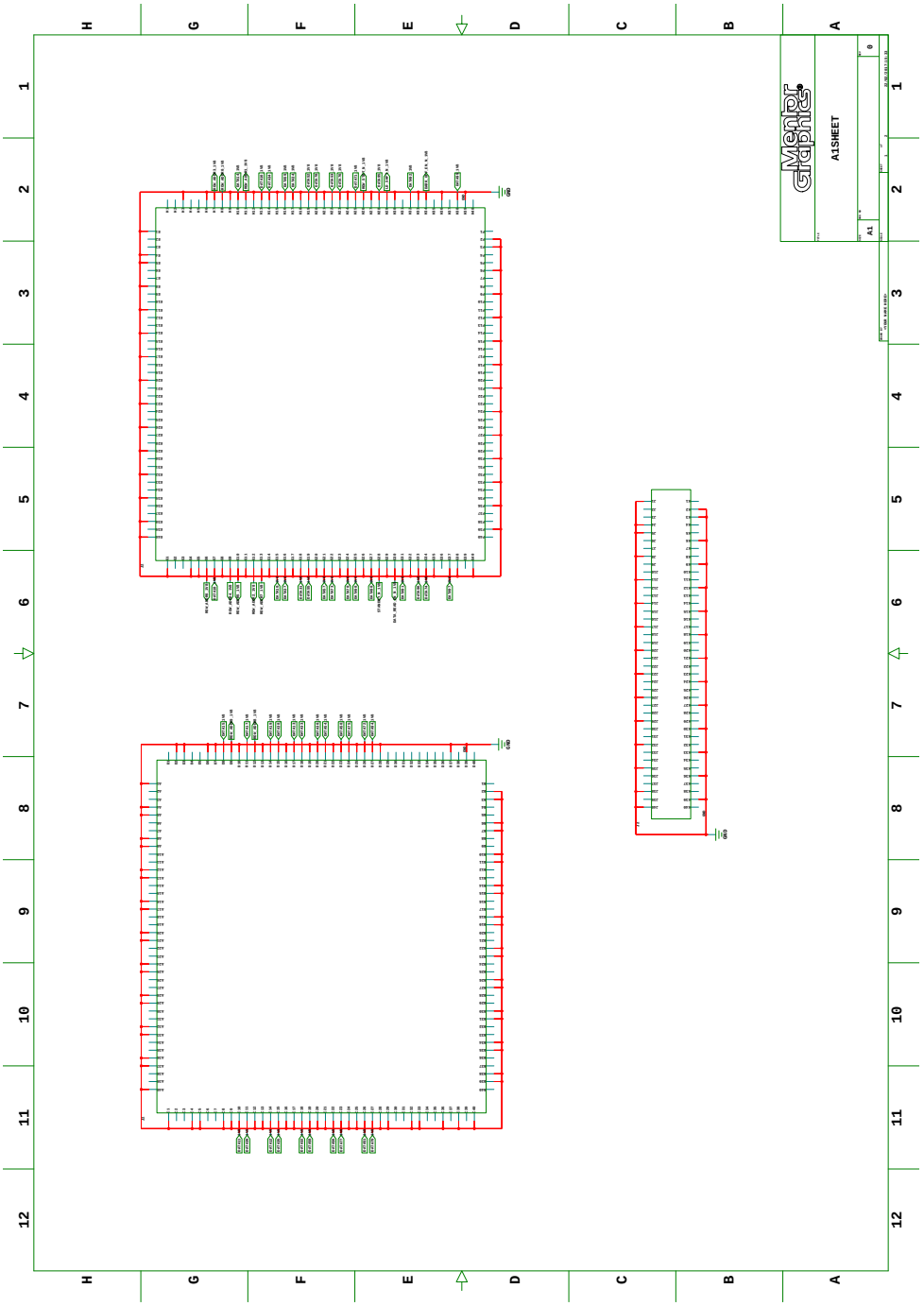
Arria 10 SoM Image Sensor Board Schematic



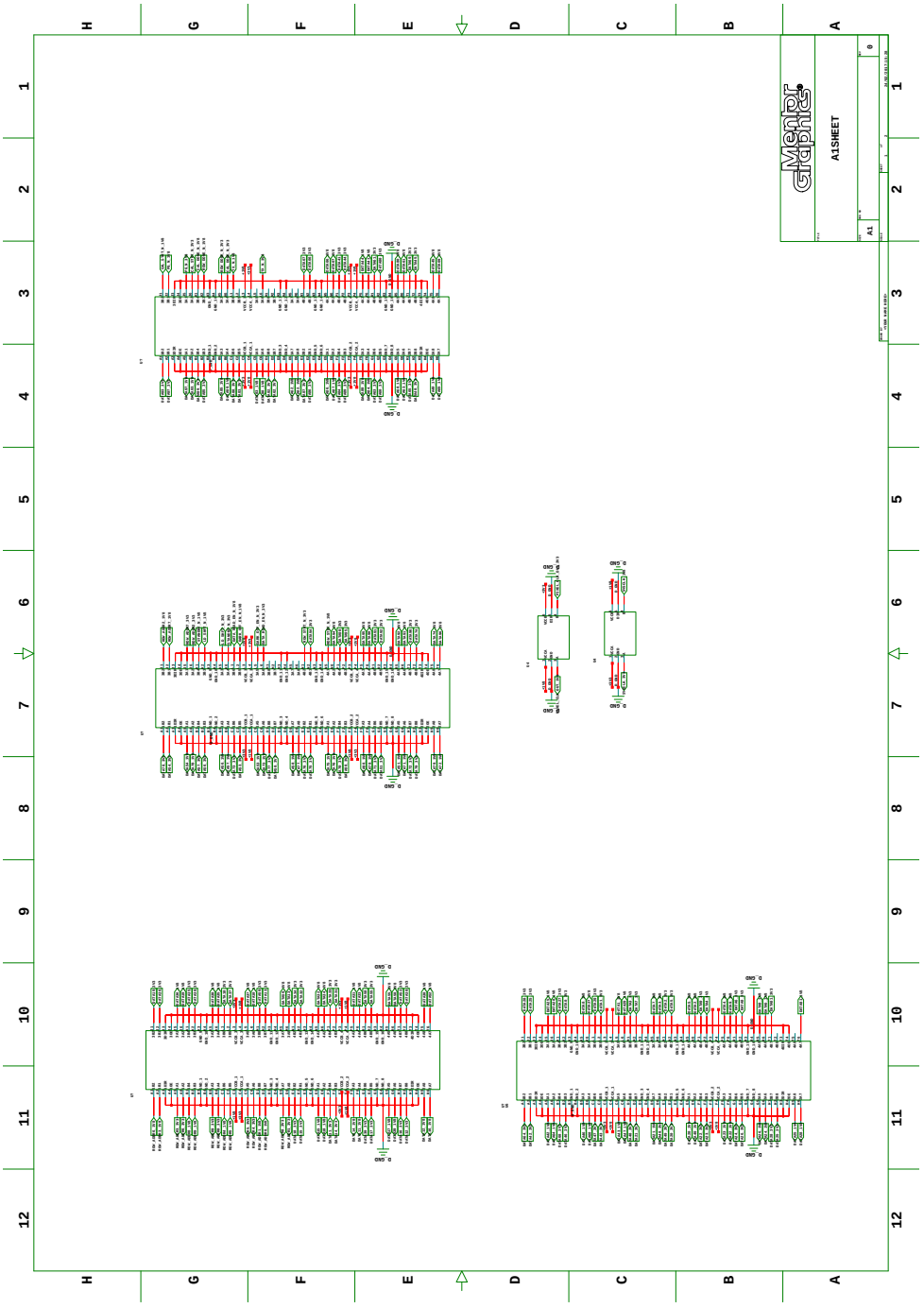
Arria 10 SoM Image Sensor Board Schematic



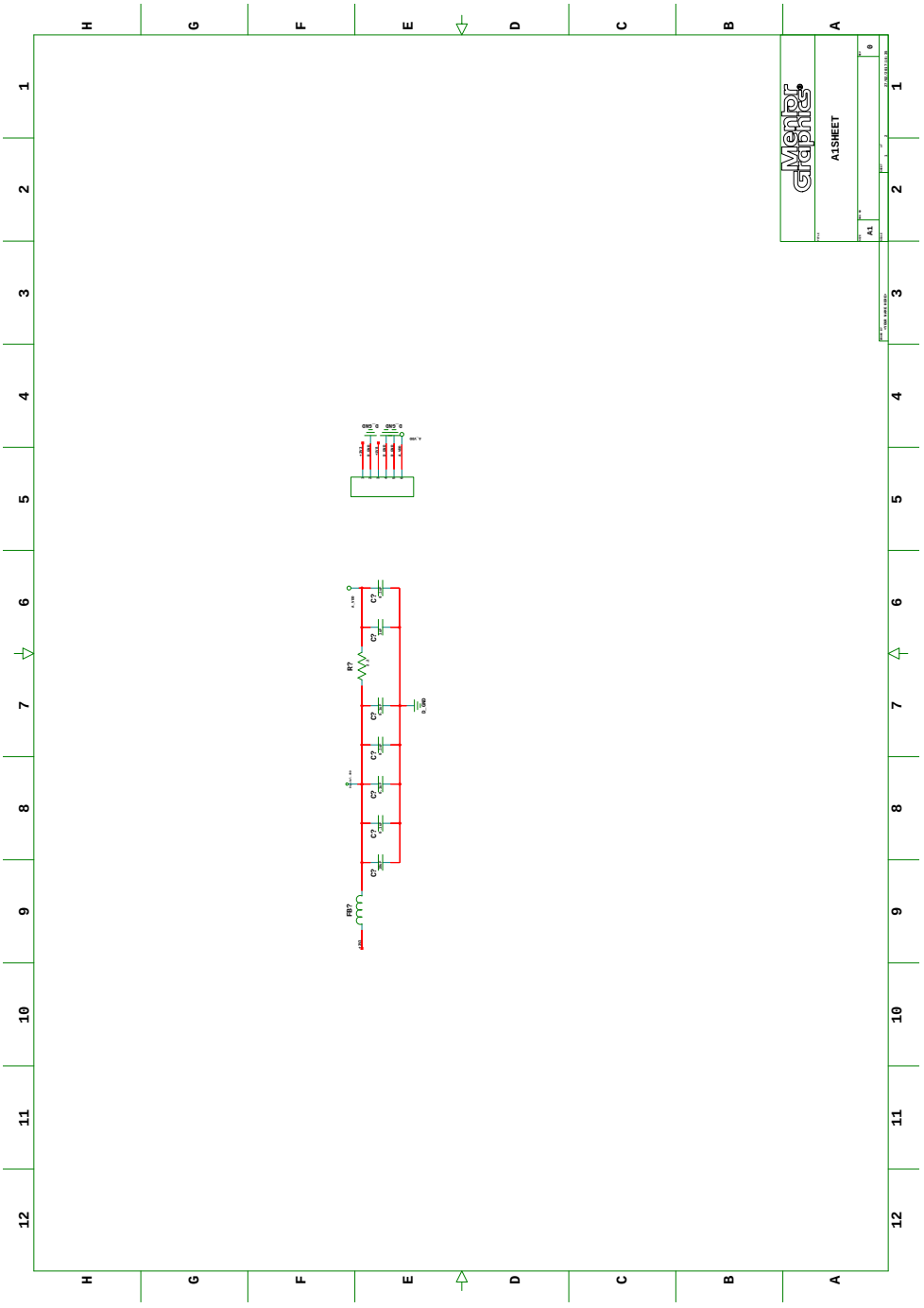
Arria 10 SoM Image Sensor Board Schematic



Arria 10 SoM Image Sensor Board Schematic



Arria 10 SoM Image Sensor Board Schematic



Arria 10 SoM VHDL Code

```
1  ----------------------------------------------------------------------------
2  --
3  --! @file        top.vhd
4  --! @brief       Arria 10 SoM Top Level File
5  --
6  --! @author      Connor Dack
7  --! @date        April 2017
8  --! @copyright   Copyright (C) 2015 Ross K. Snider and Connor Dack
9  --
10 --   This program is free software: you can redistribute it
11 --   and/or modify it under the terms of the GNU General
12 --   Public License as published by the Free Software Foundation,
13 --   either version 3 of the License, or (at your option) any
14 --   later version.
15 --
16 --   This program is distributed in the hope that it will be
17 --   useful, but WITHOUT ANY WARRANTY; without even the implied
18 --   warranty of MERCHANTABILITY or FITNESS for A PARTICULAR
19 --   PURPOSE. See the GNU General Public License for more details.
20 --
21 --   You should have received a copy of the GNU General Public
22 --   License along with this program.  If not, see
23 --   <http://www.gnu.org/licenses/>.
24 --
25 --   Connor Dack
26 --   Electrical and Computer Engineering
27 --   Montana State University
28 --   610 Cobleigh Hall
29 --   Bozeman, MT 59717
30 --   connor.dack@msu.montana.edu
31 --
32 ----------------------------------------------------------------------------
33
34 library ieee;
35 use ieee.std_logic_1164.all;
36 use ieee.numeric_std.all;
37 use ieee.math_real.all;
38
39 use work.Sensor_Package.all;
40 use work.user_controls.all;
41
42 ----------------------------------------------------------------------------
43 --
44 --! @brief     top
```

```
45 --
46 --! @param    clk_50mhz              :  input : 50 MHz input clock
47 --! @param    mem_a                  : output : DRAM Memory Signals
48 --! @param    mem_ba                 : output : DRAM Memory Signals
49 --! @param    mem_ck                 : output : DRAM Memory Signals
50 --! @param    mem_ck_n               : output : DRAM Memory Signals
51 --! @param    mem_cke                : output : DRAM Memory Signals
52 --! @param    mem_cs_n               : output : DRAM Memory Signals
53 --! @param    mem_dq                 :  inout : DRAM Memory Signals
54 --! @param    mem_dqs                :  inout : DRAM Memory Signals
55 --! @param    mem_dqs_n              :  inout : DRAM Memory Signals
56 --! @param    mem_dbi_n              :  inout : DRAM Memory Signals
57 --! @param    mem_odt                : output : DRAM Memory Signals
58 --! @param    mem_par                : output : DRAM Memory Signals
59 --! @param    mem_bg                 : output : DRAM Memory Signals
60 --! @param    mem_reset_n            : output : DRAM Memory Signals
61 --! @param    mem_alert_n            :  input : DRAM Memory Signals
62 --! @param    mem_act_n              : output : DRAM Memory Signals
63 --! @param    mem_ref_clk            :  input : DRAM Memory Signals
64 --! @param    mem_oct_rzqin          :  input : DRAM Memory Signals
65 --! @param    pixel_0                :  input : Pixel 0
66 --! @param    pixel_1                :  input : Pixel 1
67 --! @param    pixel_2                :  input : Pixel 2
68 --! @param    pixel_3                :  input : Pixel 3
69 --! @param    pixel_4                :  input : Pixel 4
70 --! @param    pixel_5                :  input : Pixel 5
71 --! @param    pixel_6                :  input : Pixel 6
72 --! @param    pixel_7                :  input : Pixel 7
73 --! @param    pixel_8                :  input : Pixel 8
74 --! @param    pixel_9                :  input : Pixel 9
75 --! @param    camera_clk             :  input : Camera Clock
76 --! @param    hps_clk                :  input : HPS Input Clock
77 --! @param    hps_reset              :  input : HPS Reset
78 --! @param    result_data            : output : Data Result
79 --! @param    result_ready           :  input : Result Ready Flag
80 --! @param    result_valid           : output : Result Valid Flag
81 --! @param    hps_enable_flag        :  input : HPS Enable
82 --! @param    pixel_clk_in           :  input : Pixel Clock
83 --! @param    cal_done_n_in          :  input : Calibration Done
84 --! @param    cal_strt_n_out         : output : Calibration Start
85 --! @param    dark_off_en_n_out      : output : Dark Off Enable
86 --! @param    ld_shft_n_out          : output : LD Shift
87 --! @param    data_read_en_n_out     : output : Data Read Enable
88 --! @param    row_addr_out           : output : Row Address
89 --! @param    row_done_n_in          :  input : Row Done
90 --! @param    row_strt_n_out         : output : Row Start
91 --! @param    standby_n_out          : output : Standby
92 --! @param    tx_n_out               : output : TX Signal
```

```
93  --! @param     pg_n_out              : output : PG Signal
94  --
95  ------------------------------------------------------------------------------
96
97  entity top is
98      port(clk_50mhz          :    in std_logic;
99           mem_a              :   out std_logic_vector(16 downto 0);
100          mem_ba             :   out std_logic_vector( 1 downto 0);
101          mem_ck             :   out std_logic_vector( 0 downto 0);
102          mem_ck_n           :   out std_logic_vector( 0 downto 0);
103          mem_cke            :   out std_logic_vector( 0 downto 0);
104          mem_cs_n           :   out std_logic_vector( 0 downto 0);
105          mem_dq             : inout std_logic_vector(71 downto 0);
106          mem_dqs            : inout std_logic_vector( 8 downto 0);
107          mem_dqs_n          : inout std_logic_vector( 8 downto 0);
108          mem_dbi_n          : inout std_logic_vector( 8 downto 0);
109          mem_odt            :   out std_logic_vector( 0 downto 0);
110          mem_par            :   out std_logic_vector( 0 downto 0);
111          mem_bg             :   out std_logic_vector( 0 downto 0);
112          mem_reset_n        :   out std_logic_vector( 0 downto 0);
113          mem_alert_n        :    in std_logic_vector( 0 downto 0);
114          mem_act_n          :   out std_logic_vector( 0 downto 0);
115          mem_ref_clk        :    in std_logic;
116          mem_oct_rzqin      :    in std_logic;
117          pixel_0            :    in std_logic_vector(
118                                      INPUT_DATA_SIZE-1 downto 0);
119          pixel_1            :    in std_logic_vector(
120                                      INPUT_DATA_SIZE-1 downto 0);
121          pixel_2            :    in std_logic_vector(
122                                      INPUT_DATA_SIZE-1 downto 0);
123          pixel_3            :    in std_logic_vector(
124                                      INPUT_DATA_SIZE-1 downto 0);
125          pixel_4            :    in std_logic_vector(
126                                      INPUT_DATA_SIZE-1 downto 0);
127          pixel_5            :    in std_logic_vector(
128                                      INPUT_DATA_SIZE-1 downto 0);
129          pixel_6            :    in std_logic_vector(
130                                      INPUT_DATA_SIZE-1 downto 0);
131          pixel_7            :    in std_logic_vector(
132                                      INPUT_DATA_SIZE-1 downto 0);
133          pixel_8            :    in std_logic_vector(
134                                      INPUT_DATA_SIZE-1 downto 0);
135          pixel_9            :    in std_logic_vector(
136                                      INPUT_DATA_SIZE-1 downto 0);
137          camera_clk         :    in std_logic;
138          hps_clk            :    in std_logic;
139          hps_reset          :    in std_logic;
140          result_data        :   out std_logic_vector(
```

```
141                                         OBJECT_ADDRESS_SIZE+ natural(
142                                         trunc(log2(real(
143                                         NUMBER_OF_CLASSES))))+32-1
144                                         downto 0);
145         result_ready      :    in std_logic;
146         result_valid      :   out std_logic;
147         hps_enable_flag   :    in std_logic;
148         pixel_clk_in      :    in std_logic;
149         cal_done_n_in     :    in std_logic;
150         cal_strt_n_out    :   out std_logic;
151         dark_off_en_n_out :   out std_logic;
152         ld_shft_n_out     :   out std_logic;
153         data_read_en_n_out :  out std_logic;
154         row_addr_out      :   out std_logic_vector(9 downto 0);
155         row_done_n_in     :    in std_logic;
156         row_strt_n_out    :   out std_logic;
157         standby_n_out     :   out std_logic;
158         tx_n_out          :   out std_logic;
159         pg_n_out          :   out std_logic;
160         fmc_clk_m2c_n     :    in std_logic_vector( 1 downto 0);
161         fmc_clk_m2c_p     :    in std_logic_vector( 1 downto 0);
162         fmc_ga            :    in std_logic_vector( 1 downto 0);
163         fmc_gpio_0        :    in std_logic;
164         fmc_gpio_1        :    in std_logic;
165         fmc_gpio_2        :    in std_logic;
166         fmc_gpio_3        :    in std_logic;
167         fmc_gpio_4        :    in std_logic;
168         fmc_gpio_5        :    in std_logic;
169         fmc_gpio_6        :    in std_logic;
170         fmc_gpio_7        :    in std_logic;
171         fmc_la_rx_clk_n   :    in std_logic;
172         fmc_la_rx_clk_p   :    in std_logic;
173         fmc_la_rx_n       :    in std_logic_vector(15 downto 0);
174         fmc_la_rx_p       :    in std_logic_vector(15 downto 0);
175         fmc_la_tx_clk_n   :    in std_logic;
176         fmc_la_tx_clk_p   :    in std_logic;
177         fmc_la_tx_n       :    in std_logic_vector(17 downto 0);
178         fmc_la_tx_p       :    in std_logic_vector(17 downto 0);
179         fmc_prsnt         :    in std_logic;
180         fmc_scl           :    in std_logic;
181         fmc_sda           :    in std_logic;
182         user_pb_fpga      :    in std_logic_vector( 1 downto 0));
183 end entity;
184
185 architecture rtl of top is
186
187     ------------------------------------------------------------------
188     -- Component declarations
```

```
189     -----------------------------------------------------------------------
190     component jtag_master is
191         port (
192             clk_clk           : in  std_logic := '0';
193             clk_reset_reset   : in  std_logic := '0';
194             master_address    : out std_logic_vector(31 downto 0);
195             master_readdata   : in  std_logic_vector(31 downto 0)
196                               := (others => '0');
197             master_read       : out std_logic;
198             master_write      : out std_logic;
199             master_writedata  : out std_logic_vector(31 downto 0);
200             master_waitrequest : in  std_logic   := '0';
201             master_readdatavalid : in  std_logic := '0';
202             master_byteenable : out std_logic_vector(3 downto 0);
203             master_reset_reset : out std_logic
204         );
205     end component;
206
207
208     component ddr_mem is
209         port (
210             clk_clk                : in std_logic := '0';
211             emif_0_mem_mem_ck      : out std_logic_vector(0
212                                                   downto 0);
213             emif_0_mem_mem_ck_n    : out std_logic_vector(0
214                                                   downto 0);
215             emif_0_mem_mem_a       : out std_logic_vector(16
216                                                   downto 0);
217             emif_0_mem_mem_act_n   : out std_logic_vector(0
218                                                   downto 0);
219             emif_0_mem_mem_ba      : out std_logic_vector(1
220                                                   downto 0);
221             emif_0_mem_mem_bg      : out std_logic_vector(0
222                                                   downto 0);
223             emif_0_mem_mem_cke     : out std_logic_vector(0
224                                                   downto 0);
225             emif_0_mem_mem_cs_n    : out std_logic_vector(0
226                                                   downto 0);
227             emif_0_mem_mem_odt     : out std_logic_vector(0
228                                                   downto 0);
229             emif_0_mem_mem_reset_n : out std_logic_vector(0
230                                                   downto 0);
231             emif_0_mem_mem_par     : out std_logic_vector(0
232                                                   downto 0);
233             emif_0_mem_mem_alert_n :  in std_logic_vector(0
234                                       downto 0):= (others => '0');
235             emif_0_mem_mem_dqs     : inout std_logic_vector(8
236                                       downto 0):= (others => '0');
```

```
237             emif_0_mem_mem_dqs_n    : inout std_logic_vector(8
238                                       downto 0):= (others => '0');
239             emif_0_mem_mem_dq       : inout std_logic_vector(71
240                                       downto 0):= (others => '0');
241             emif_0_mem_mem_dbi_n    : inout std_logic_vector(8
242                                       downto 0):= (others => '0');
243             emif_0_oct_oct_rzqin    :    in std_logic  := '0';
244             emif_0_status_local_cal_success : out std_logic;
245             emif_0_status_local_cal_fail    : out std_logic;
246             memory_master_driver_0_buffer_empty_export    : out
247                                                     std_logic;
248             memory_master_driver_0_buffer_output_export   : out
249                                 std_logic_vector(127 downto 0);
250             memory_master_driver_0_buffer_rd_clk_clk      :  in
251                                                 std_logic := '0';
252             memory_master_driver_0_buffer_read_en_export :  in
253                                                 std_logic := '0';
254             memory_master_driver_0_csr_write          :  in
255                                                 std_logic := '0';
256             memory_master_driver_0_csr_address        :  in
257                                               std_logic_vector(31
258                                               downto 0) := (
259                                               others => '0');
260             memory_master_driver_0_csr_writedata      :  in
261                                               std_logic_vector(
262                                               127 downto 0) := (
263                                               others => '0');
264             memory_master_driver_0_csr_waitrequest:out std_logic;
265             read_start_export :  in std_logic := '0';
266             reset_reset_n     :  in std_logic := '0';
267             write_clk_clk     : out std_logic);
268     end component ddr_mem;
269
270     component camera_control_avalon is
271         port(rst_n              :  in std_logic;
272              clk                :  in std_logic;
273              hps_enable_flag    :  in std_logic;
274              pixel_clk_in       :  in std_logic;
275              cal_done_n_in      :  in std_logic;
276              cal_strt_n_out     : out std_logic;
277              dark_off_en_n_out  : out std_logic;
278              ld_shft_n_out      : out std_logic;
279              data_read_en_n_out : out std_logic;
280              row_addr_out    : out std_logic_vector(9 downto 0);
281              row_done_n_in   :  in std_logic;
282              row_strt_n_out  : out std_logic;
283              standby_n_out   : out std_logic;
284              tx_n_out        : out std_logic;
```

```
285             pg_n_out         : out std_logic;
286             avs_s1_read      :  in std_logic;
287             avs_s1_readdata  : out std_logic_vector(31 downto 0);
288             avs_s1_write     :  in std_logic;
289             avs_s1_address   :  in std_logic_vector(1 downto 0);
290             avs_s1_writedata: in std_logic_vector(31 downto 0));
291     end component;
292
293     component camera_preprocessing is
294         port(rst_n                : in std_logic;
295             component_clk         : in std_logic;
296             dram_clk              : in std_logic;
297             hps_enable_flag       : in std_logic;
298             data_0_in             : in std_logic_vector(
299                                     INPUT_DATA_SIZE - 1 downto 0);
300             data_1_in             : in std_logic_vector(
301                                     INPUT_DATA_SIZE - 1 downto 0);
302             data_2_in             : in std_logic_vector(
303                                     INPUT_DATA_SIZE - 1 downto 0);
304             data_3_in             : in std_logic_vector(
305                                     INPUT_DATA_SIZE - 1 downto 0);
306             data_4_in             : in std_logic_vector(
307                                     INPUT_DATA_SIZE - 1 downto 0);
308             data_5_in             : in std_logic_vector(
309                                     INPUT_DATA_SIZE - 1 downto 0);
310             data_6_in             : in std_logic_vector(
311                                     INPUT_DATA_SIZE - 1 downto 0);
312             data_7_in             : in std_logic_vector(
313                                     INPUT_DATA_SIZE - 1 downto 0);
314             data_8_in             : in std_logic_vector(
315                                     INPUT_DATA_SIZE - 1 downto 0);
316             data_9_in             : in std_logic_vector(
317                                     INPUT_DATA_SIZE - 1 downto 0);
318             buffer_empty_in       : in std_logic;
319             buffer_read_out       : out std_logic;
320             buffer_readdata_in    : in std_logic_vector(
321                                        DRAM_DATA_SIZE-1 downto 0);
322             data_packet_out       : out std_logic_vector(((
323                                        SUPER_PIXEL_SIZE *
324                                        NUMBER_OF_PARALLEL_CHANNELS)
325                                        - 1) downto 0);
326             pixel_data_and_location_valid_flag_out :
327                                                  out std_logic;
328             super_pixel_valid_flag_out : out std_logic;
329             clear_pixel_flag_out       : out std_logic;
330             new_frame_flag_out         : out std_logic);
331     end component;
332
```

```
333     component computation_unit is
334         generic(TOTAL_INPUT_SIZE     : natural :=
335                                        NUMBER_OF_PARALLEL_CHANNELS
336                                        * SUPER_PIXEL_SIZE;
337                 WORD_SIZE            : natural := 32;
338                 RESULTS_MEMORY_SIZE  : natural := 64
339         );
340         port(linescan_clk          : in std_logic;
341              rst_n                 : in std_logic;
342              hps_reset             : in std_logic;
343              line_rst_n            : in std_logic;
344              linescan_obj          : in std_logic_vector(
345                                      PIXEL_ADDRESS_SIZE * 2 +
346                                      OBJECT_ADDRESS_SIZE +
347                                      WORD_SIZE - 1 downto 0);
348              regression_clk        : in std_logic;
349              pixel_read_en         : out std_logic;
350              pixel_read_empty      : in std_logic;
351              super_pixel_in        : in std_logic_vector(
352                                       TOTAL_INPUT_SIZE - 1
353                                       downto 0);
354              fast_clk              : in std_logic;
355              clear_pixel           : in std_logic;
356              object_decision       : out std_logic_vector(
357                                       OBJECT_ADDRESS_SIZE + 1 - 1
358                                       downto 0);
359              hps_clk               : in std_logic;
360              sink_result_data    : out std_logic_vector(
361                                    OBJECT_ADDRESS_SIZE + natural(
362                                    trunc(log2(real(
363                                    NUMBER_OF_CLASSES)))) +
364                                    WORD_SIZE-1 downto 0);
365              sink_result_ready   : in std_logic;
366              sink_result_valid   : out std_logic;
367              hps_avs_s1_read     :  in std_logic;
368              hps_avs_s1_write    :  in std_logic;
369              hps_avs_s1_address  :  in std_logic_vector(
370                                                    31 downto 0);
371              hps_avs_s1_readdata : out std_logic_vector(
372                                                    31 downto 0);
373              hps_avs_s1_writedata : in std_logic_vector(
374                                                    31 downto 0));
375     end component;
376
377     component air_jet_fifo is
378         port(data    : in  std_logic_vector(31 downto 0)
379                                    := (others => 'X');
380              wrreq   : in  std_logic := 'X';
```

```
381             rdreq    : in  std_logic := 'X';
382             wrclk    : in  std_logic := 'X';
383             rdclk    : in  std_logic := 'X';
384             q        : out std_logic_vector(31 downto 0);
385             rdempty  : out std_logic;
386             wrfull   : out std_logic);
387     end component air_jet_fifo;
388
389     component dual_clock_fifo is
390         generic(
391             lpm_numwords            : natural;
392             lpm_width               : natural;
393             lpm_widthu              : natural;
394             rdsync_delaypipe        : natural;
395             wrsync_delaypipe        : natural);
396         port(
397             data      : in  std_logic_vector(lpm_width - 1
398                                     downto 0) := (others => 'X');
399             wrreq     : in  std_logic := 'X';
400             rdreq     : in  std_logic := 'X';
401             wrclk     : in  std_logic := 'X';
402             rdclk     : in  std_logic := 'X';
403             aclr      : in std_logic  := '0';
404             q         : out std_logic_vector(lpm_width - 1
405                                                 downto 0);
406             rdempty   : out std_logic;
407             wrfull    : out std_logic;
408             rdfull    : out std_logic;
409             wrempty   : out std_logic;
410             rdusedw   : out std_logic_vector(lpm_widthu - 1
411                                                 downto 0);
412             wrusedw   : out std_logic_vector(lpm_widthu - 1
413                                                 downto 0);
414             eccstatus : out std_logic_vector(1 downto 0));
415     end component dual_clock_fifo;
416
417     component camera_interface_line_scan is
418         port(clk    :  in  std_logic;
419              rst_n  :  in  std_logic;
420              positive_threshold: in std_logic_vector(7 downto 0);
421              negative_threshold: in std_logic_vector(7 downto 0);
422              fmc_clk_m2c_n    : in  std_logic_vector( 1 downto 0);
423              fmc_clk_m2c_p    : in  std_logic_vector( 1 downto 0);
424              fmc_ga           : in  std_logic_vector( 1 downto 0);
425              fmc_gpio_0       : in  std_logic;
426              fmc_gpio_1       : in  std_logic;
427              fmc_gpio_2       : in  std_logic;
428              fmc_gpio_3       : in  std_logic;
```

```
429             fmc_gpio_4       : in  std_logic;
430             fmc_gpio_5       : in  std_logic;
431             fmc_gpio_6       : in  std_logic;
432             fmc_gpio_7       : in  std_logic;
433             fmc_la_rx_clk_n  : in  std_logic;
434             fmc_la_rx_clk_p  : in  std_logic;
435             fmc_la_rx_n      : in  std_logic_vector(15 downto 0);
436             fmc_la_rx_p      : in  std_logic_vector(15 downto 0);
437             fmc_la_tx_clk_n  : in  std_logic;
438             fmc_la_tx_clk_p  : in  std_logic;
439             fmc_la_tx_n      : in  std_logic_vector(17 downto 0);
440             fmc_la_tx_p      : in  std_logic_vector(17 downto 0);
441             fmc_prsnt        : in  std_logic;
442             fmc_scl          : in  std_logic;
443             fmc_sda          : in  std_logic;
444             pixel_array : out std_logic_vector(0 to
445                                                NUMBER_OF_TAPS - 1);
446             fval         : out std_logic;
447             x_lval       : out std_logic);
448     end component;
449
450     component object_tracking_line_scan is
451         port(clk                          :  in std_logic;
452             rst_n                         :  in std_logic;
453             data_valid_in                 :  in std_logic;
454             pixel_data_in                 :  in std_logic_vector(0
455                                               to NUMBER_OF_TAPS - 1);
456             transciever_fifo_empty_in     :  in std_logic;
457             air_jet_fifo_empty_in         :  in std_logic;
458             linescan_offset_in            :  in std_logic_vector(
459                                                AOI_ADDRESS_SIZE -
460                                                1 downto 0);
461             object_packet_out             : out std_logic_vector(
462                                               TRANSCEIVER_DATA_SIZE
463                                               - 1 downto 0);
464             ignore_transceiver_obj_out    : out std_logic;
465             air_jet_packet_out            : out std_logic_vector(
466                                              OBJECT_ADDRESS_SIZE
467                                              + AOI_ADDRESS_SIZE +
468                                              LINE_NUMBER_ADDRESS_SIZE
469                                              - 1 downto 0);
470             air_jet_wre_out               : out std_logic;
471             current_line_out              : out std_logic_vector(
472                                              LINE_NUMBER_ADDRESS_SIZE
473                                              - 1 downto 0));
474     end component;
475
476     component computation_pll
```

```
477         port (
478             locked    : out std_logic;
479             outclk_0  : out std_logic;
480             outclk_1  : out std_logic;
481             outclk_2  : out std_logic;
482             refclk    : in  std_logic := '0';
483             rst       : in  std_logic := '0');
484     end component;
485
486     component reset_synchro is
487         port(
488             clk          : in std_logic;
489             reset_in     : in std_logic;
490             reset_out    : out std_logic
491         );
492     end component;
493
494     ------------------------------------------------------------------
495     -- Constant and Type declarations
496     ------------------------------------------------------------------
497     constant C_SYS_CLK_PERIOD : time       := 10 ns;
498     constant PIXEL_CONDITION  : unsigned(DRAM_LOCATION_SIZE - 1
499                                 downto 0) := to_unsigned((
500                                 NUMBER_OF_PIXELS) - 1,
501                                 DRAM_LOCATION_SIZE);
502     constant SPECTRAL_BIN_CONDITION : unsigned(DRAM_LOCATION_SIZE
503                                 - 1 downto 0) := to_unsigned(
504                                 NUMBER_OF_SPECTRAL_BINS,
505                                 DRAM_LOCATION_SIZE);
506     constant PIXEL_COUNTER_ZEROS : std_logic_vector(
507                                  DRAM_LOCATION_SIZE - 1 downto 0)
508                                  := (others => '0');
509     constant SPECTRAL_BIN_COUNTER_ZEROS : unsigned(
510                                 DRAM_LOCATION_SIZE - 1 downto 0)
511                                 := (others => '0');
512     constant NUMBER_OF_LANES    : natural := 1;
513     constant LANEWIDTH          : natural := 32;
514
515
516     ------------------------------------------------------------------
517     -- Signal declarations
518     ------------------------------------------------------------------
519     signal read_counter : integer range 0 to 10;
520
521
522     signal s_ed_status_local_cal_success : std_logic;
523     signal s_ed_status_local_cal_fail : std_logic;
524     signal s_buffer_empty : std_logic;
```

```
525     signal s_buffer_read : std_logic := '0';
526     signal s_buffer_readdata : std_logic_vector(127 downto 0);
527     signal s_read_start : std_logic;
528
529     signal s_dram_address     : std_logic_vector(31 downto 0);
530     signal s_dram_waitrequest : std_logic;
531     signal s_dram_write       : std_logic;
532     signal s_dram_write_clk   : std_logic;
533     signal s_dram_writedata   : std_logic_vector(127 downto 0);
534
535     signal dram_counter : integer range 0 to NUMBER_OF_PIXELS
536                                   * NUMBER_OF_SPECTRAL_BINS + 1;
537
538     -- Camera Interface Signals
539     signal s_pixel_0 : std_logic_vector(INPUT_DATA_SIZE-1
540                                                         downto 0);
541     signal s_pixel_1 : std_logic_vector(INPUT_DATA_SIZE-1
542                                                         downto 0);
543     signal s_pixel_2 : std_logic_vector(INPUT_DATA_SIZE-1
544                                                         downto 0);
545     signal s_pixel_3 : std_logic_vector(INPUT_DATA_SIZE-1
546                                                         downto 0);
547     signal s_pixel_4 : std_logic_vector(INPUT_DATA_SIZE-1
548                                                         downto 0);
549     signal s_pixel_5 : std_logic_vector(INPUT_DATA_SIZE-1
550                                                         downto 0);
551     signal s_pixel_6 : std_logic_vector(INPUT_DATA_SIZE-1
552                                                         downto 0);
553     signal s_pixel_7 : std_logic_vector(INPUT_DATA_SIZE-1
554                                                         downto 0);
555     signal s_pixel_8 : std_logic_vector(INPUT_DATA_SIZE-1
556                                                         downto 0);
557     signal s_pixel_9 : std_logic_vector(INPUT_DATA_SIZE-1
558                                                         downto 0);
559
560     -- Camera Clock
561     signal resetcount       : std_logic_vector(5 downto 0)
562                             := (others => '0');
563     signal powerOnReset     : std_logic;
564     signal reset, reset_n   : std_logic;
565     signal reset_in_synchro : std_logic;
566     signal reset_data       : std_logic;
567     signal reset_data_n     : std_logic;
568     signal locked_pll_Q     : std_logic;
569     signal locked_pll_Q2    : std_logic;
570     signal clk_100MHz_buf   : std_logic;
571
572     -- Enables and Flags
```

```
573     signal new_frame_flag : std_logic := '0';
574     signal super_pixel_valid_flag : std_logic := '0';
575     signal s_pixel_data_and_location_valid_flag:std_logic := '0';
576     signal s_clear_pixel_flag : std_logic := '0';
577
578     -- Super Pixel FIFO Signals
579     signal data_packet_out : std_logic_vector(((SUPER_PIXEL_SIZE*
580                              NUMBER_OF_PARALLEL_CHANNELS) - 1)
581                              downto 0) := (others => '0');
582     signal data_to_computation: std_logic_vector(((
583                                 SUPER_PIXEL_SIZE *
584                                 NUMBER_OF_PARALLEL_CHANNELS) - 1)
585                                 downto 0);
586     signal super_pixel_rdreq  : std_logic := '0';
587     signal super_pixel_full   : std_logic := '0';
588     signal super_pixel_empty  : std_logic := '0';
589
590     -- Write DRAM Value Signals
591     signal light_data : std_logic_vector(LIGHT_CORRECT_SIZE
592                         - 1 downto 0) := (others => '0');
593     signal dark_data  : std_logic_vector(DARK_CORRECT_SIZE
594                         - 1 downto 0) := (others => '1');
595
596
597     signal pixel_counter : unsigned(DRAM_LOCATION_SIZE - 1
598                            downto 0) := PIXEL_CONDITION;
599     signal spectral_bin_counter : unsigned(DRAM_LOCATION_SIZE - 1
600                            downto 0) := SPECTRAL_BIN_CONDITION;
601
602     signal comp_pll_locked, fast_clk : std_logic;
603     signal regress_clk, linescan_clk : std_logic;
604
605     signal s_object_decision : std_logic_vector(
606                                OBJECT_ADDRESS_SIZE+1-1 downto 0);
607     signal object_valid : std_logic;
608     signal data_ready : std_logic;
609
610     signal stat_rr_link : std_logic;
611
612     signal linescan_obj : std_logic_vector(PIXEL_ADDRESS_SIZE * 2
613                           + OBJECT_ADDRESS_SIZE + 32-1 downto 0);
614     signal crc_err : std_logic;
615
616     signal g_reset, g_reset_n   : std_logic;
617
618     signal avs_s1_read          : std_logic;
619     signal avs_s1_write         : std_logic;
620     signal avs_s1_address       : std_logic_vector(31 downto 0);
```

```
621     signal avs_s1_readdata      : std_logic_vector(31 downto 0);
622     signal avs_s1_writedata     : std_logic_vector(31 downto 0);
623
624     signal avs_s2_read          : std_logic;
625     signal avs_s2_write         : std_logic;
626     signal avs_s2_address       : std_logic_vector(31 downto 0);
627     signal avs_s2_readdata      : std_logic_vector(31 downto 0);
628     signal avs_s2_writedata     : std_logic_vector(31 downto 0);
629
630     -- Object Tracking Signals ------------------------------------
631     signal pixel_0_line_scan    : std_logic_vector(7 downto 0);
632     signal pixel_1_line_scan    : std_logic_vector(7 downto 0);
633     signal pixel_2_line_scan    : std_logic_vector(7 downto 0);
634     signal pixel_3_line_scan    : std_logic_vector(7 downto 0);
635     signal pixel_4_line_scan    : std_logic_vector(7 downto 0);
636     signal pixel_5_line_scan    : std_logic_vector(7 downto 0);
637     signal pixel_6_line_scan    : std_logic_vector(7 downto 0);
638     signal pixel_7_line_scan    : std_logic_vector(7 downto 0);
639     signal pixel_8_line_scan    : std_logic_vector(7 downto 0);
640     signal pixel_9_line_scan    : std_logic_vector(7 downto 0);
641
642     signal fval         : std_logic;
643     signal x_lval       : std_logic;
644
645     signal x_clk        : std_logic;
646     signal y_clk        : std_logic;
647     signal z_clk        : std_logic;
648     signal combined_clk : std_logic;
649     signal pixel_clk    : std_logic;
650
651     signal pixel_array:std_logic_vector(0 to NUMBER_OF_TAPS - 1);
652
653     -- Object Thresholds
654     signal positive_threshold : std_logic_vector(7 downto 0)
655                                                     := x"FA";
656     signal negative_threshold : std_logic_vector(7 downto 0)
657                                                     := x"00";
658
659     -- Linescan to Hyperspectral Ratios
660     signal linescan_offset : std_logic_vector(AOI_ADDRESS_SIZE -
661                                                 1 downto 0);
662     signal pixel_ratio     : std_logic_vector(31 downto 0);
663     signal lines_ratio     : std_logic_vector(31 downto 0);
664
665     -- Transciever FIFO
666     signal S_object_packet          : std_logic_vector(
667                                       TRANSCEIVER_DATA_SIZE - 1
668                                       downto 0);
```

```
669     signal S_object_packet_out        : std_logic_vector(
670                                         TRANSCEIVER_DATA_SIZE - 1
671                                         downto 0);
672     signal s_ignore_trans_obj         : std_logic;
673     signal transciever_fifo_rdreq     : std_logic := '0';
674     signal transciever_fifo_read_clk  : std_logic;
675     signal data_to_arria_10           : std_logic_vector(
676                                        TRANSCEIVER_DATA_SIZE - 1
677                                        downto 0);
678     signal transciever_fifo_empty     : std_logic;
679     signal transciever_fifo_full      : std_logic;
680
681     -- Air Jet Team Information
682     signal s_air_jet_packet_out : std_logic_vector(
683                                   OBJECT_ADDRESS_SIZE +
684                                   AOI_ADDRESS_SIZE +
685                                   LINE_NUMBER_ADDRESS_SIZE - 1
686                                   downto 0);
687     signal s_air_jet_wre_out    : std_logic;
688     signal s_current_line       : std_logic_vector(
689                                   LINE_NUMBER_ADDRESS_SIZE - 1
690                                   downto 0);
691
692     -- Air Jet FIFO
693     signal s_object_air_jet     : std_logic_vector(
694                                   OBJECT_ADDRESS_SIZE +
695                                   AOI_ADDRESS_SIZE +
696                                   LINE_NUMBER_ADDRESS_SIZE - 1
697                                   downto 0);
698     signal s_air_jet_fifo_empty : std_logic;
699     signal s_air_jet_fifo_full  : std_logic;
700     signal s_air_jet_fifo_rdreq : std_logic := '0';
701
702     signal cal_done_delay_counter          : integer := 0;
703     constant clk_cycles_after_cal_done     : integer := 100;
704     signal f_cal_done                      : std_logic := '0';
705     signal f_data_valid                    : std_logic := '0';
706
707 begin
708
709     test_master : jtag_master
710     port map(
711         clk_clk              => clk_50mhz,
712         clk_reset_reset      => reset,
713         master_address       => avs_s1_address,
714         master_readdata      => avs_s1_readdata,
715         master_read          => avs_s1_read,
716         master_write         => avs_s1_write,
```

```
717             master_writedata      => avs_s1_writedata,
718             master_waitrequest    => '0',
719             master_readdatavalid  => '1',
720             master_byteenable     => open,
721             master_reset_reset    => open
722         );
723
724 ---------------------------------------------------------------
725 -- PowerOnReset
726 ---------------------------------------------------------------
727         process(clk_50mhz)
728         begin
729             if(rising_edge(clk_50mhz)) then
730                 if(resetcount = "111111") then
731                     powerOnReset <= '0';
732                 else
733                     powerOnReset <= '1';
734                     resetcount <= std_logic_vector(
735                                             unsigned(resetcount) + 1);
736                 end if;
737             end if;
738         end process;
739
740 ---------------------------------------------------------------
741 -- Create Reset
742 ---------------------------------------------------------------
743         process(clk_50mhz, powerOnReset)
744         begin
745             if(powerOnReset = '1') then
746                 reset <= '1';
747                 reset_n <= '0';
748             elsif(rising_edge(clk_50mhz)) then
749                 locked_pll_Q <= (comp_pll_locked);
750                 locked_pll_Q2 <= locked_pll_Q;
751                 if(locked_pll_Q2 = '1') then
752                     reset <= '0';
753                     reset_n <= '1';
754                 else
755                     reset <= '1';
756                     reset_n <= '0';
757                 end if;
758             end if;
759         end process;
760
761         reset_synchro2_i : reset_synchro
762         port map(
763             clk         => regress_clk,
764             reset_in    => reset,
```

```
765         reset_out    => g_reset
766     );
767 
768     g_reset_n <= not g_reset;
769 
770     reset_in_synchro <= reset or not (stat_rr_link) or not (
771                                     s_ed_status_local_cal_success);
772 
773     reset_synchro_i : reset_synchro
774     port map(
775         clk          => linescan_clk,
776         reset_in     => reset_in_synchro,
777         reset_out    => reset_data
778     );
779     reset_data_n <= not reset_data;
780 
781     clock_gen : computation_pll
782     port map(
783         locked       => comp_pll_locked,
784         outclk_0     => fast_clk,  --210--300 MHz
785         outclk_1     => regress_clk, --70--100 MHz
786         outclk_2     => linescan_clk, --140 MHz
787         refclk       => clk_50mhz,
788         rst          => powerOnReset
789     );
790 
791     camera_cntrl : camera_control_avalon
792         port map(
793             rst_n               => reset_n,
794             clk                 => camera_clk,
795             hps_enable_flag     => hps_enable_flag,
796             pixel_clk_in        => pixel_clk_in,
797             cal_done_n_in       => cal_done_n_in,
798             cal_strt_n_out      => cal_strt_n_out,
799             dark_off_en_n_out   => dark_off_en_n_out,
800             ld_shft_n_out       => ld_shft_n_out,
801             data_read_en_n_out  => data_read_en_n_out,
802             row_addr_out        => row_addr_out,
803             row_done_n_in       => row_done_n_in,
804             row_strt_n_out      => row_strt_n_out,
805             standby_n_out       => standby_n_out ,
806             tx_n_out            => tx_n_out,
807             pg_n_out            => pg_n_out,
808             avs_s1_read         => avs_s2_read,
809             avs_s1_readdata     => avs_s2_readdata,
810             avs_s1_write        => avs_s2_write,
811             avs_s1_address      => avs_s2_address(1 downto 0),
812             avs_s1_writedata    => avs_s2_writedata);
```

```
813
814     u0 : component ddr_mem
815     port map (
816         clk_clk => mem_ref_clk,
817         emif_0_mem_mem_ck => mem_ck,
818         emif_0_mem_mem_ck_n => mem_ck_n,
819         emif_0_mem_mem_a => mem_a,
820         emif_0_mem_mem_act_n => mem_act_n,
821         emif_0_mem_mem_ba => mem_ba,
822         emif_0_mem_mem_bg => mem_bg,
823         emif_0_mem_mem_cke => mem_cke,
824         emif_0_mem_mem_cs_n => mem_cs_n,
825         emif_0_mem_mem_odt => mem_odt,
826         emif_0_mem_mem_reset_n => mem_reset_n,
827         emif_0_mem_mem_par => mem_par,
828         emif_0_mem_mem_alert_n => mem_alert_n,
829         emif_0_mem_mem_dqs => mem_dqs,
830         emif_0_mem_mem_dqs_n => mem_dqs_n,
831         emif_0_mem_mem_dq => mem_dq,
832         emif_0_mem_mem_dbi_n => mem_dbi_n,
833         emif_0_oct_oct_rzqin => mem_oct_rzqin,
834         emif_0_status_local_cal_success =>
835                                     s_ed_status_local_cal_success,
836         emif_0_status_local_cal_fail =>
837                                       s_ed_status_local_cal_fail,
838         memory_master_driver_0_buffer_empty_export =>
839                                                   s_buffer_empty,
840         memory_master_driver_0_buffer_output_export =>
841                                                    s_buffer_readdata,
842         memory_master_driver_0_buffer_rd_clk_clk =>
843                                                     s_dram_write_clk,
844         memory_master_driver_0_buffer_read_en_export =>
845                                                      s_buffer_read,
846         memory_master_driver_0_csr_write => s_dram_write,
847         memory_master_driver_0_csr_address => s_dram_address,
848         memory_master_driver_0_csr_writedata => s_dram_writedata,
849         memory_master_driver_0_csr_waitrequest =>
850                                                    s_dram_waitrequest,
851         read_start_export=> hps_enable_flag,
852         reset_reset_n => reset_n,
853         write_clk_clk => s_dram_write_clk);
854
855     ---------------------------------------------------------------
856     -- Port Maps
857     ---------------------------------------------------------------
858     camera_preprocessing_wirer : component camera_preprocessing
859     port map(rst_n               => reset_n,
860              component_clk       => camera_clk,
```

```
861             dram_clk            => mem_ref_clk,
862             hps_enable_flag     => hps_enable_flag,
863             data_0_in           => s_pixel_0,
864             data_1_in           => s_pixel_1,
865             data_2_in           => s_pixel_2,
866             data_3_in           => s_pixel_3,
867             data_4_in           => s_pixel_4,
868             data_5_in           => s_pixel_5,
869             data_6_in           => s_pixel_6,
870             data_7_in           => s_pixel_7,
871             data_8_in           => s_pixel_8,
872             data_9_in           => s_pixel_9,
873             buffer_empty_in     => s_buffer_empty,
874             buffer_read_out     => s_buffer_read,
875             buffer_readdata_in => s_buffer_readdata,
876             data_packet_out     => data_packet_out,
877             pixel_data_and_location_valid_flag_out =>
878                             s_pixel_data_and_location_valid_flag,
879             super_pixel_valid_flag_out =>super_pixel_valid_flag,
880             clear_pixel_flag_out => s_clear_pixel_flag,
881             new_frame_flag_out   => new_frame_flag);
882
883     super_pixel_fifo_wirer : component dual_clock_fifo
884     generic map(
885         lpm_numwords            => 16,
886         lpm_width               => SUPER_PIXEL_SIZE * 5,
887         lpm_widthu              => 4,
888         rdsync_delaypipe        => 4,
889         wrsync_delaypipe        => 4
890     )
891     port map(
892         data                    => data_packet_out,
893         wrreq                   => super_pixel_valid_flag,
894         rdreq                   => super_pixel_rdreq,
895         wrclk                   => camera_clk,
896         rdclk                   => regress_clk,
897         q                       => data_to_computation,
898         rdempty                 => super_pixel_empty,
899         wrfull                  => super_pixel_full,
900         aclr                    => open,
901         eccstatus               => open,
902         rdfull                  => open,
903         rdusedw                 => open,
904         wrempty                 => open,
905         wrusedw                 => open);
906
907     computation : computation_unit
908     generic map(
```

```
909         TOTAL_INPUT_SIZE     => NUMBER_OF_PARALLEL_CHANNELS *
910                                                 SUPER_PIXEL_SIZE,
911         WORD_SIZE            => 32,
912         RESULTS_MEMORY_SIZE => 1024 -- must be power of 2
913     )
914     port map(linescan_clk           => linescan_clk,
915              rst_n                  => g_reset_n,
916              hps_reset              => hps_reset,
917              line_rst_n             => reset_data_n,
918              linescan_obj           => S_object_packet(
919                                        PIXEL_ADDRESS_SIZE * 2 +
920                                        OBJECT_ADDRESS_SIZE + 32-1
921                                        downto 0),
922              regression_clk         => regress_clk,
923              pixel_read_en          => super_pixel_rdreq,
924              pixel_read_empty       => super_pixel_empty,
925              super_pixel_in         => data_to_computation,
926              fast_clk               => fast_clk,
927              clear_pixel            => s_clear_pixel_flag,
928              object_decision        => s_object_decision,
929              hps_clk                => hps_clk,
930              sink_result_data       => result_data,
931              sink_result_ready      => result_ready,
932              sink_result_valid      => result_valid,
933              hps_avs_s1_read        => avs_s1_read,
934              hps_avs_s1_write       => avs_s1_write,
935              hps_avs_s1_address     => avs_s1_address,
936              hps_avs_s1_readdata    => avs_s1_readdata,
937              hps_avs_s1_writedata   => avs_s1_writedata);
938
939     waterfall_interface : camera_interface_line_scan port map(
940         clk                 => pixel_clk,
941         rst_n               => reset_n,
942         positive_threshold  => positive_threshold,
943         negative_threshold  => negative_threshold,
944         fmc_clk_m2c_n       => fmc_clk_m2c_n,
945         fmc_clk_m2c_p       => fmc_clk_m2c_p,
946         fmc_ga              => fmc_ga,
947         fmc_gpio_0          => fmc_gpio_0,
948         fmc_gpio_1          => fmc_gpio_1,
949         fmc_gpio_2          => fmc_gpio_2,
950         fmc_gpio_3          => fmc_gpio_3,
951         fmc_gpio_4          => fmc_gpio_4,
952         fmc_gpio_5          => fmc_gpio_5,
953         fmc_gpio_6          => fmc_gpio_6,
954         fmc_gpio_7          => fmc_gpio_7,
955         fmc_la_rx_clk_n     => fmc_la_rx_clk_n,
956         fmc_la_rx_clk_p     => fmc_la_rx_clk_p,
```

```
957         fmc_la_rx_n          => fmc_la_rx_n,
958         fmc_la_rx_p          => fmc_la_rx_p,
959         fmc_la_tx_clk_n      => fmc_la_tx_clk_n,
960         fmc_la_tx_clk_p      => fmc_la_tx_clk_p,
961         fmc_la_tx_n          => fmc_la_tx_n,
962         fmc_la_tx_p          => fmc_la_tx_p,
963         fmc_prsnt            => fmc_prsnt,
964         fmc_scl              => fmc_scl,
965         fmc_sda              => fmc_sda,
966         pixel_array          => pixel_array,
967         fval                 => fval,
968         x_lval               => x_lval);
969
970     sys_object_tracking : object_tracking_line_scan port map(
971         clk                          => pixel_clk,
972         rst_n                        => reset_data_n,
973         data_valid_in                => fval,
974         pixel_data_in                => pixel_array,
975         transciever_fifo_empty_in    => transciever_fifo_empty,
976         air_jet_fifo_empty_in        => s_air_jet_fifo_empty,
977         linescan_offset_in           => linescan_offset,
978         object_packet_out            => S_object_packet,
979         ignore_transceiver_obj_out => s_ignore_trans_obj,
980         air_jet_packet_out           => s_air_jet_packet_out,
981         air_jet_wre_out              => s_air_jet_wre_out,
982         current_line_out             => s_current_line);
983
984
985     air_jet_fifo_u0 : component air_jet_fifo
986         port map(data    => s_air_jet_packet_out,
987                  wrreq   => s_air_jet_wre_out,
988                  rdreq   => s_air_jet_fifo_rdreq,
989                  wrclk   => pixel_clk,
990                  rdclk   => pixel_clk,
991                  q       => s_object_air_jet,
992                  rdempty => s_air_jet_fifo_empty,
993                  wrfull  => s_air_jet_fifo_full);
994
995
996     process(regress_clk)
997     begin
998         if(rising_edge(regress_clk)) then
999             if(data_ready = '1' and unsigned(s_object_decision)
1000                     /= to_unsigned(0,OBJECT_ADDRESS_SIZE+1)) then
1001                 object_valid <= '1';
1002             else
1003                 object_valid <= '0';
1004             end if;
```

```
1005            end if;
1006        end process;
1007
1008        -- Register the pixel data
1009        register_data : process(camera_clk, reset_n)
1010            begin
1011                if(reset_n <= '0') then
1012                    s_pixel_0 <= (others => '0');
1013                    s_pixel_1 <= (others => '0');
1014                    s_pixel_2 <= (others => '0');
1015                    s_pixel_3 <= (others => '0');
1016                    s_pixel_4 <= (others => '0');
1017                    s_pixel_5 <= (others => '0');
1018                    s_pixel_6 <= (others => '0');
1019                    s_pixel_7 <= (others => '0');
1020                    s_pixel_8 <= (others => '0');
1021                    s_pixel_9 <= (others => '0');
1022                elsif(rising_edge(camera_clk)) then
1023                    if(f_data_valid = '1') then
1024                        s_pixel_0 <= pixel_0;
1025                        s_pixel_1 <= pixel_1;
1026                        s_pixel_2 <= pixel_2;
1027                        s_pixel_3 <= pixel_3;
1028                        s_pixel_4 <= pixel_4;
1029                        s_pixel_5 <= pixel_5;
1030                        s_pixel_6 <= pixel_6;
1031                        s_pixel_7 <= pixel_7;
1032                        s_pixel_8 <= pixel_8;
1033                        s_pixel_9 <= pixel_9;
1034                    else
1035                        s_pixel_0 <= (others => '0');
1036                        s_pixel_1 <= (others => '0');
1037                        s_pixel_2 <= (others => '0');
1038                        s_pixel_3 <= (others => '0');
1039                        s_pixel_4 <= (others => '0');
1040                        s_pixel_5 <= (others => '0');
1041                        s_pixel_6 <= (others => '0');
1042                        s_pixel_7 <= (others => '0');
1043                        s_pixel_8 <= (others => '0');
1044                        s_pixel_9 <= (others => '0');
1045                    end if;
1046                end if;
1047            end process;
1048
1049         -- Data valid process
1050        data_valid_process : process(camera_clk, reset_n)
1051            begin
1052                if(reset_n <= '0') then
```

```
1053                f_data_valid <= '0';
1054                f_cal_done   <= '0';
1055                cal_done_delay_counter <= 0;
1056            elsif(rising_edge(camera_clk)) then
1057                if(cal_done_n_in = '0') then
1058                    f_cal_done <= '1';
1059                end if;
1060                if(f_cal_done = '1' and cal_done_delay_counter =
1061                        clk_cycles_after_cal_done) then
1062                    f_data_valid <= '1';
1063                else
1064                    f_data_valid <= '0';
1065                    cal_done_delay_counter <=
1066                    cal_done_delay_counter + 1;
1067                end if;
1068            end if;
1069        end process;
1070
1071    combined_clk <= x_clk AND y_clk AND z_clk;
1072 end architecture;
```