ECE 6276 DSP Hardware System Design (Fall 2018)

Project Report

Edge Detection and VGA

Team Number: 10

Suvrat Krishna Mishra	**903423604**
Clancy Jembia	**903433543**
Abhinav Himanshu	**903424420**
Steve Medie	**903433377**

Date: 12/04/2018

Table of Contents

1 Introduction

In an image, an edge is the boundary between two homogenous regions. Applying edge detection to an image means identifying all edges on the image, as shown in figure 1.

Figure 1 Edge detection on an image

Edge detection has many applications such as reducing unnecessary information in the image while preserving the fundamental image structure (compression) and extracting important features in the image (shapes) as shown in figure 2. This is greatly used in image recognition and computer vision.

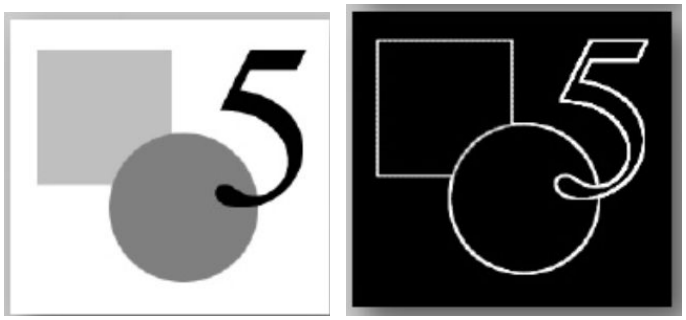

Figure 2 extracting features in an image

An application of edge detection include, identifying objects immersed in images, more precisely, a self driving car identifying a couple of signs it sees on a road and comparing it to its database and chooses an appropriate action.

There are several ways of performing edge detection. To understand how to carry out edge detection, consider the ideal step edges in 1 and 2 dimensions as shown in figure 3.

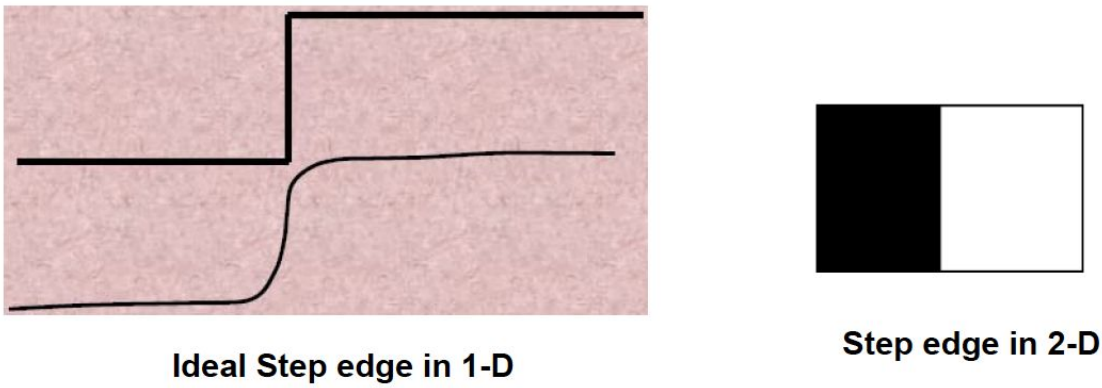

Figure 3 step edges in 1-D and 2-D

In one dimension, an edge is characterized by a very high value in the derivative, or the point where the second derivative is zero (maximum/minimum in first derivative) as shown in figure 4. These are the fundamental principles behind most methods of edge detection.

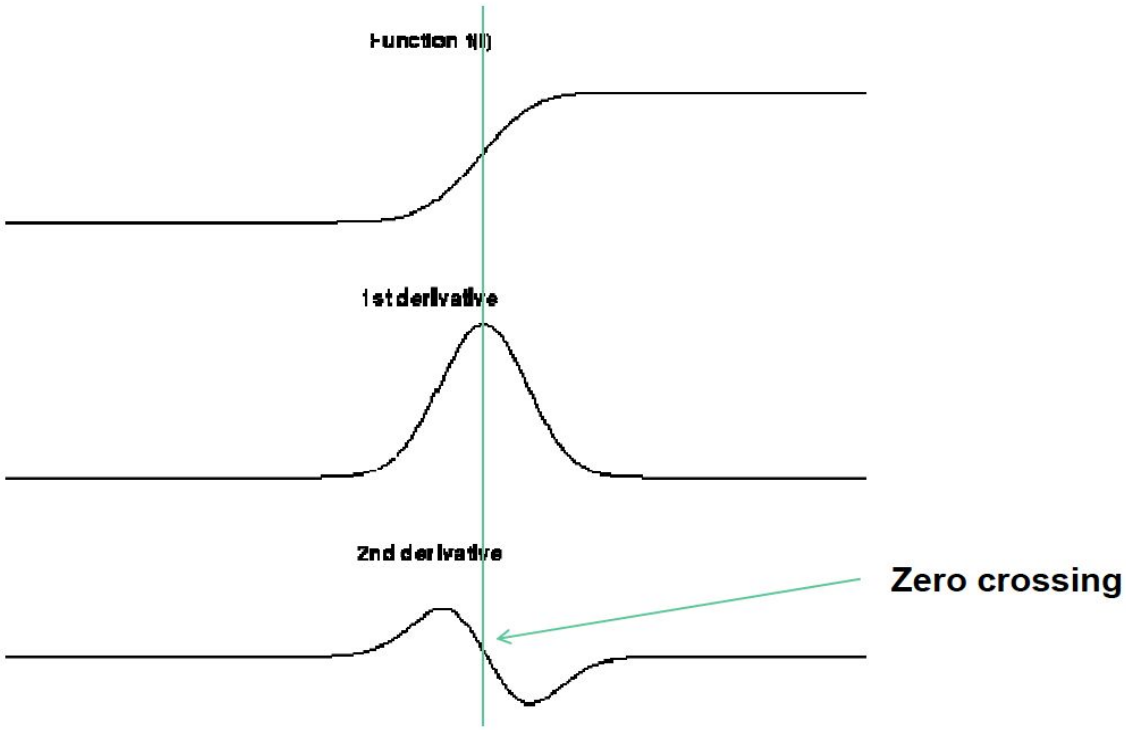

Figure 4 fundamental principles for edge detection

Now when we consider a continuous function (in one dimension), it is easy to obtain a continuous derivative function and identify an edge at the extrema of this function but given samples of this function, the derivative at each point can be inferred by computing the difference between the next and previous values and dividing by the difference between the x-values of the samples as shown in figure 5.

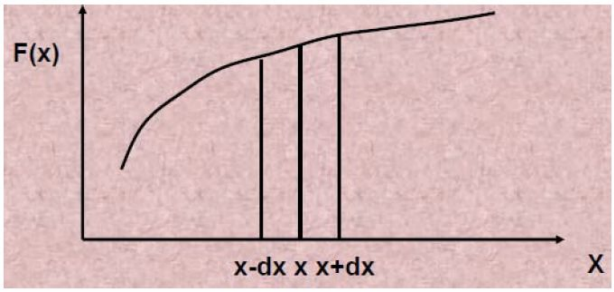

Figure 5 Inferring the derivative using samples

There are multiple methods of edge detection based either on zero crossing or on the first derivative being high or the zero crossing of the second derivative.

The method we are using to perform edge filtering is based on the first derivative being higher than a certain threshold. The simply estimate in our From figure 5, for example, f'(x) = (f(x+dx)-f(x-dx))/2dx. The method we are using in this lab is called the Sobel method and it passes each pixel of the image through the following filters:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

$$G = |G_x| + |G_y|$$

Figure 6 Sobel method filters

This filter performs a calculation similar to the approximate of the derivative from figure 5 in the x and y directions. The magnitude is then approximated to be the sum of the absolute values and this is compared to an appropriate threshold to determine whether there is an edge.

2 Features

- 100 Mhz operating frequency of the module allows this module to work in real-time applications.
- This module provides user to change the threshold value during the run-time.
- Module provides higher level of threshold values, 11 bit threshold value for comparison provides better differentiability between smooth and edgy regions.
- Module is capable of handling steaming images (tested and verified) or even videos (if used with faster producer).
- Architecture used for convolution only requires single lookup for each pixel value making this module very favourable for streaming application.
- Indication of Data values, State of the System, VGA ON signal is provided using LEDs on the FPGA board.
- The architecture has throughput of 1 cycle after initial latency of W+2 cycles, where W is width of image.

3 Target Specifications

Input type: A grayscale image of 256x256 (we also have Matlab code to resize convert to grayscale if required on Matlab before sending to architecture)

Input size: 65536 Bytes i.e. Weight x Height of image, each pixel is one byte.

Input format: Input is given using UART at 230400 baud rate

Input Way : Using UART module

Output type: Binary image representing edgy image.

Output Size: It is actually stored in RAM 65536 bits for 256x256 image.

Output format: Output is displayed using VGA

Output Way: Output Image is displayed on VGA screen

Target Frequency: Architecture works at 100Mhz but for VGA we require 25Mhz, we use PLL for that.

Throughput: Throughput of system is one cycle after initial latency of W+2 cycles .i.e. 258 for 256x256 image.

Sampling period : Even though edge detector module can work at 100Mhz, the sampling period is limited by the baud rate of UART which is 230400 bps. So sampling period for each 8 bit word is 47.734 usec.

4 Design Architecture

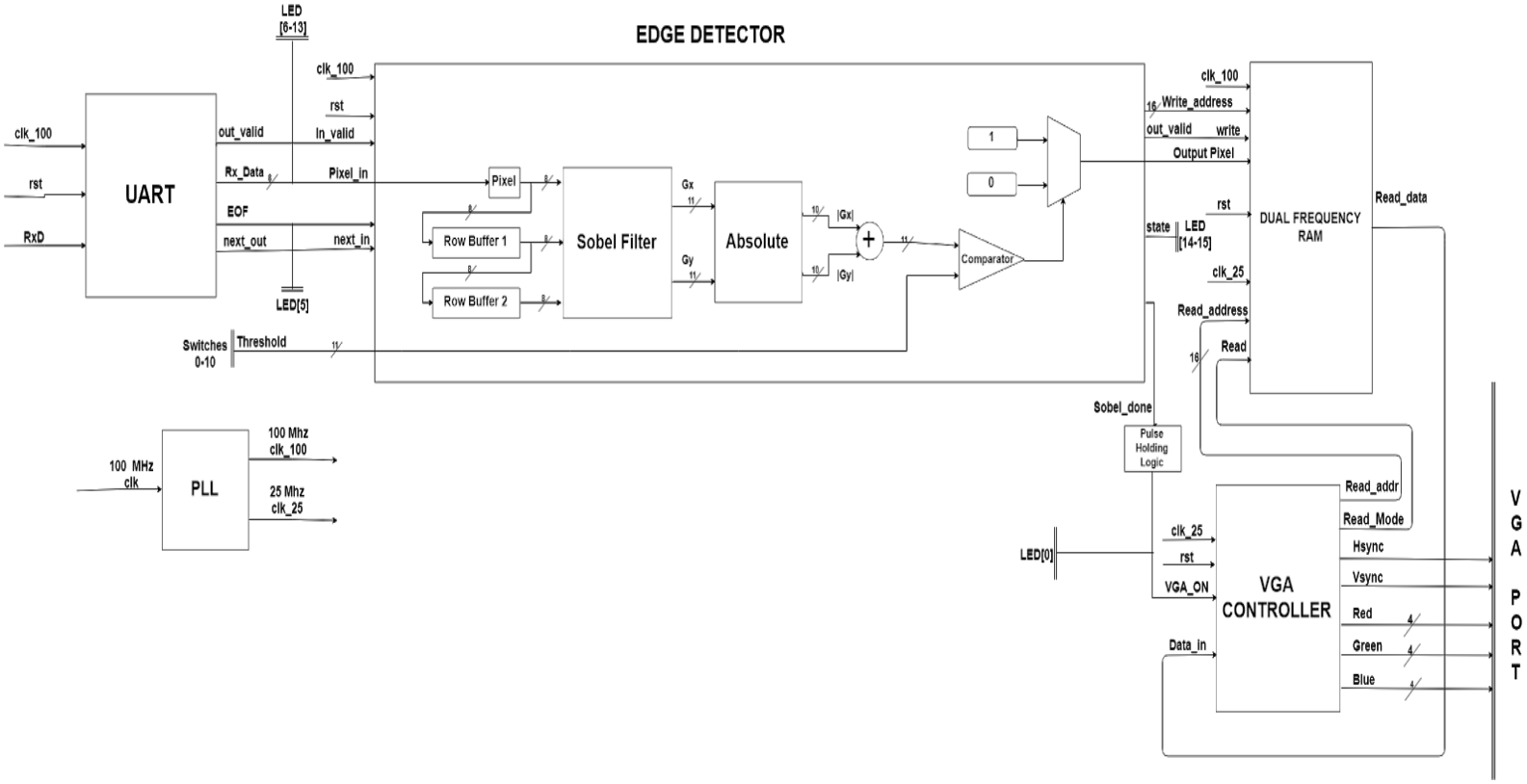

Figure 7 Overall architecture of Image Edge Detector

[More clear view of this Architecture]

- Designed Architecture can detect edges in images using Sobel Filter.
- Input Image to this architecture is provided in raster fashion, and output image which contains edges is stored in RAM in raster fashion.
- Architecture works at 100Mhz frequency and PLL is used to generate 25 Mhz clock for VGA.
- Architecture requires single memory lookup for each pixel value
- Valid values are produced after a latency of W+2 cycles, after which throughput of system is 1.
- Architecture for edge detector is effective even for streaming images or videos if used with faster producer than UART.
- Architecture gives the ability to user to change the threshold on the fly using switches (0-10) present on the board, architecture also provides more levels of threshold values to differentiate a smooth region from edgy region.
- Architecture assumes zero above the top row of image for calculation of top row, since row buffers are initially zero this condition is implicit for the first **image** with this architecture, for the bottom row edge detector module sends a busy signal to producer module (Busy signal is not required with UART because time taken by UART to form one word is very much longer

than time taken by edge detector module to compute last row because it is producing its own pixels) and inserts a zero in row buffers by itself, this helps in calculating bottom row by assuming zero after the image. This in a way also clears the Row buffers for next image, but anyways after the first image is done a flush signal is explicitly provided to clear the Row buffers. Shifting mechanism of Row buffers implicitly assumes that image is folded for calculation of Left and Right boundary of image.

This architecture follows the principle of "Simple is better".

Flow of this architecture:

256x256 grayscale image is sent from Matlab via UART, UART samples the asynchronous data and forms an 8 bit word which represents a pixel. This 8 bit word is given to an edge detector module, edge detector and UART are communicating using handshaking signals. The 8 bit word received by edge detector modules goes inside "Row Shifter" which provides a mechanism for convolution on image with only single lookup for each pixel value. Three pixels at a time are transferred to "Sobel Filter" module which already has other relevant pixel values to calculate the Sobel gradient Gx and Gy. These 11 bit Gradient values are transferred to "Absolute" module where |Gx|+|Gy| is computed. The resulting 10 bit number is than transferred to "Comp_bin" module where this number is compared with a user defined threshold at runtime and binary value representing an edge or not is outputted along with valid signal. The edge detector also produce the write address signal which are used to store data in RAM, out_valid signal of edge_detector is used to trigger the write operation in RAM. RAM is designed to write at 100Mhz while read at 25Mhz. This solves the problem of frequency mismatch between VGA and the edge detector module. After the finish of computation of first image VGA is turned ON and the stays ON thereafter. The VGA controller reads the data from the RAM at 25Mhz and display it on the screen.

Control of this architecture:

The control of the edge detector module works depending upon three states ("00" ,"01","11"). The state is also indicated on the Board via LED[14-15]. State "00" implies that either module is in idle state or has not received sufficient number of pixels to start a valid computation. This state stays for W+2 cycles (assuming all valid inputs, which of course is not the case, W is width of image). In this state output produced is always invalid. State "01" implies that module has sufficient pixels to produce valid output samples, during this state edge detector module has throughput of one. In case of invalid signal the module just ignores the sample and do not shift the data. The EOF(End Of Frame) signal indicates that input image is finished, which also pushes the edge_detetor module to State "11", where the module is busy and provides its data own it is own that is zero this state stays for W cycles. In this state module always produces a valid output. During this state module truly works at 100Mhz because module is producing its own data and does not dependent UART. As a matter of fact UART is so slow compared to edge detector module that it computes the complete last row before even UART forms a next 8 bit word, this makes the next_in signal needless in current implementation, but we are anyways keeping it if in case our module is used with faster producers. After computing the image edege_detector sends a done pulse signal, which is used for starting the VGA. The logic for generating address to write in RAM also depends on the states. After complete computation of image module switches its state back to "00" and waits for valid pixel of next image.

4.1 uArch 1: UART (myreceiver.vhd)

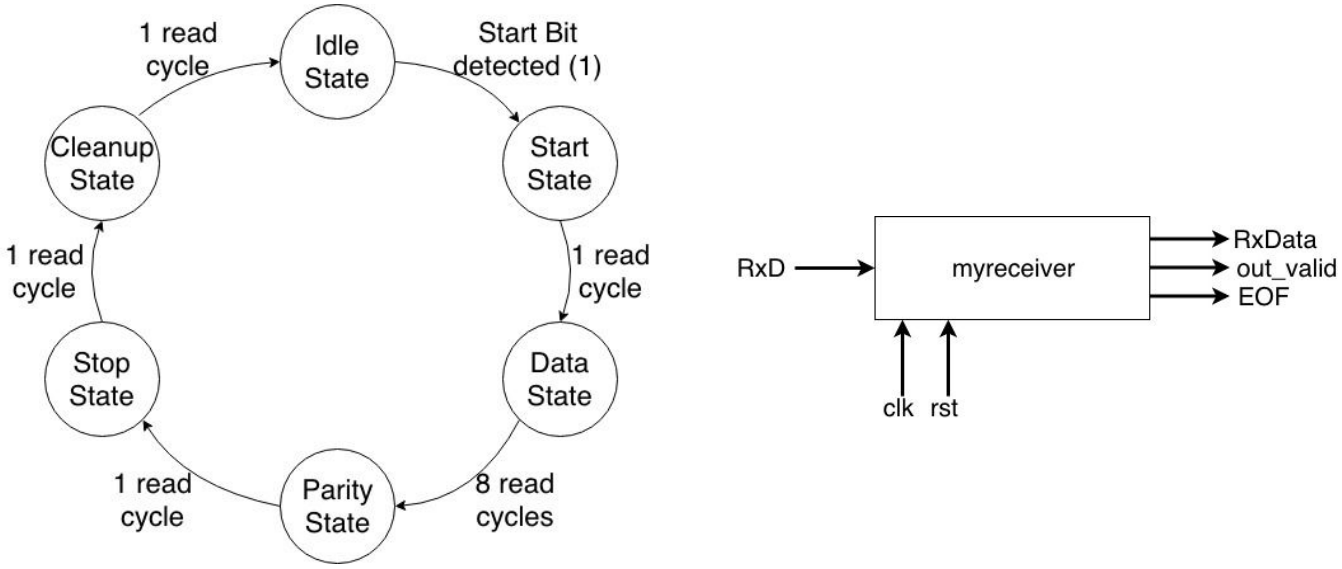

Figure 8 UART module state diagram and block diagram

1. Functionalities

The UART is the first module of our design. It collects pixels of the picture(s)/video from out of the basys 3 board and passes to the next module in our design, the row shifter. It also verifies the validity of this received pixel, outputs a valid bit for each pixel and provides a resend signal in case of invalid input. Lastly, it provides an 'end of file' signal to indicate the end of the picture(s) or the video received. The UART module resets when at the end of file or when the reset button is pressed.

Figure 9 writing/reading with UART

A serial set of 11 bits, including 1 start bit, 8 data bits, 1 parity bit and 1 stop bit are received into the receive uart port of our basys 3 board. In our case, these bits are received from matlab through the serial USB port of the computer used, so the pixel from our picture are sent from matlab via the the USB port to the basys 3 board. This USB serial port is connected to the uart receive port of our Basys 3 board. The baud rate for this connection is set to 230400 and even parity used.

The way the receive module works is that it uses a finite state machine. The different states described below;

Idle State: Note that the receive port is initially at a constant high '1' state. The idle state initializes necessary variables and waits for a start signal, '0'. Once the start bit is received, the next state, the 'Start state' is initialized.

Start State: The start state ensures that the received signal start signal was not a glitch by making sure this signal stays at '0' for at least half a period. A period is calculated by dividing the Basys' 3 clock frequency by our baud rate (100000000/230400 = 435). This gives the length of each period for a bit received. If the start bit does not stay '0' for at least half a period, we go back to our idle state. Else, we go to our next state which is the Data state.

Data State: The data state receives the 8 data bit from the uart input. To do this, we sample the incoming data at half the period as seen on the picture above. We do this by receiving a bit a period after the start bit was confirmed (distance between middle start and middle data or between the middle of 2 data bits is a period). We then wait a period and sample 7 more time. Once we have sampled the 8 data bits, we can move to the next state, the parity state.

Parity State: Similarly as above, the parity state samples the next input bit after a period, supposedly the parity bit. This bit is then compared to the even parity bit computed from the 8 input bits received. Both bits are then used to set the output valid bit (1 if equal, 0 if not). If both bits were equal, we output the 8 bit received, if not we re-ask this data by sending a transmit signal from our module to the matlab code. Lastly, if we have reached the end of the picture, we output an End of File signal.

Stop State: At this point, we have received our data. So we just sample the next bit after a period and go to the next state which is the cleanup state.

Cleanup State: Here we clear some important variables before moving back to the Idle State.

What data are stored in what format in it

2. Data-flow

The only data stored at this module are the 8 serial input bits (a pixel) received from the UART. Each bit of the pixel received in the data stage are sent to the next module in the parity stage.

3. Algorithm

- Check the UART until we detect the start bit, '1'
- When detected, start sampling the received input until the 8 bits are received, a pixel.
- The next received bit is the parity, compare with the input data bits for validity.
- If valid, save (send to next module) the pixel.
- Then Receive the stop bit, '0'.
- Repeat the process all over till we reach the end of picture.

4.2 uArch 2: Row_shifter (row_shifter.vhd)

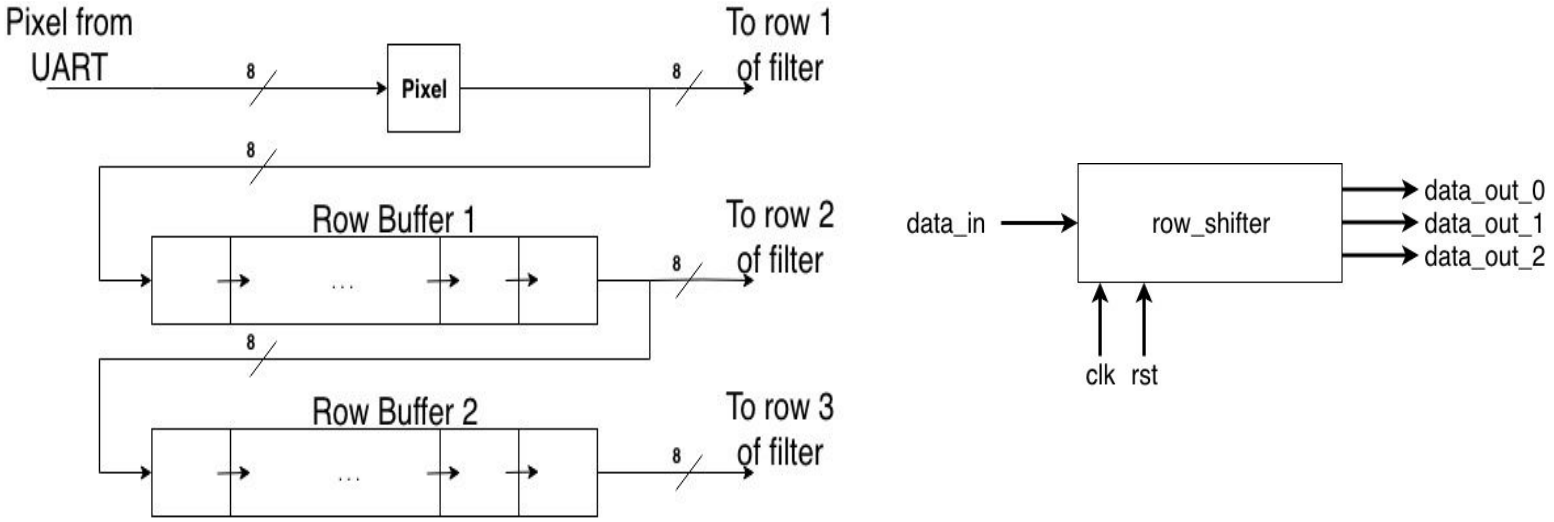

Figure 10 architecture and block diagram of row shifter

1. Functionalities

This module provides a mechanism for 3x3 window convolution over the image. Data are stored in row buffers which avoids the need for more than one lookup for each pixel. The width of each Row buffer is equal to the width of our image which separates each row from the other, while doing convolution. When calculating left and right boundary data from one Row buffer one end of image aligns with the data of the other Row buffer (other end of the image), making it seem as if image is folded for Left and Right boundary of image.

An 8 bit valid word is taken from UART, and goes into an 8 bit pixel register. Previous data from Pixel register goes into Row buffer 1 and each pixel value in both Row buffers shift to the right. The last pixel value of Row buffer 1 goes into the Row buffer 2. And also values stored in pixel, rightmost 8 bit value in Row Buffer 1 and Row Buffer 2 goes into the Sobel filter stage.

On an invalid input, input value is refused by the architecture, No shift of data occur in this case, and consequently output is marked invalid by the controller in top module of edge detector.

We are intentionally not shifting data instead of shifting data by associating a valid signal with each 8 bit value and saving computation cycles, because this will incur a huge cost for valid bits which will be equal to (2*W+1)bits .

2. Data flow

In total this architecture requires (2*W)+1 Bytes for memory, where W is width of image. We are using 256*256 image for testing and verification.

3. **Algorithm**

- The value in Pixel register, the last register of buffer 1 and that of 2 are given to sobel filter.
- The entire 2nd row buffer is right shifted to the right by 1.
- The last register of row buffer 1 is fed to first register of row buffer 2.
- The entire 1st row buffer is right shifted to the right by 1.
- Pixel is fed to first register of row buffer 1.
- Input data is stored in the pixel register

4.3 uArch 3: Sobel Filter (`sobel_filter.vhd`)

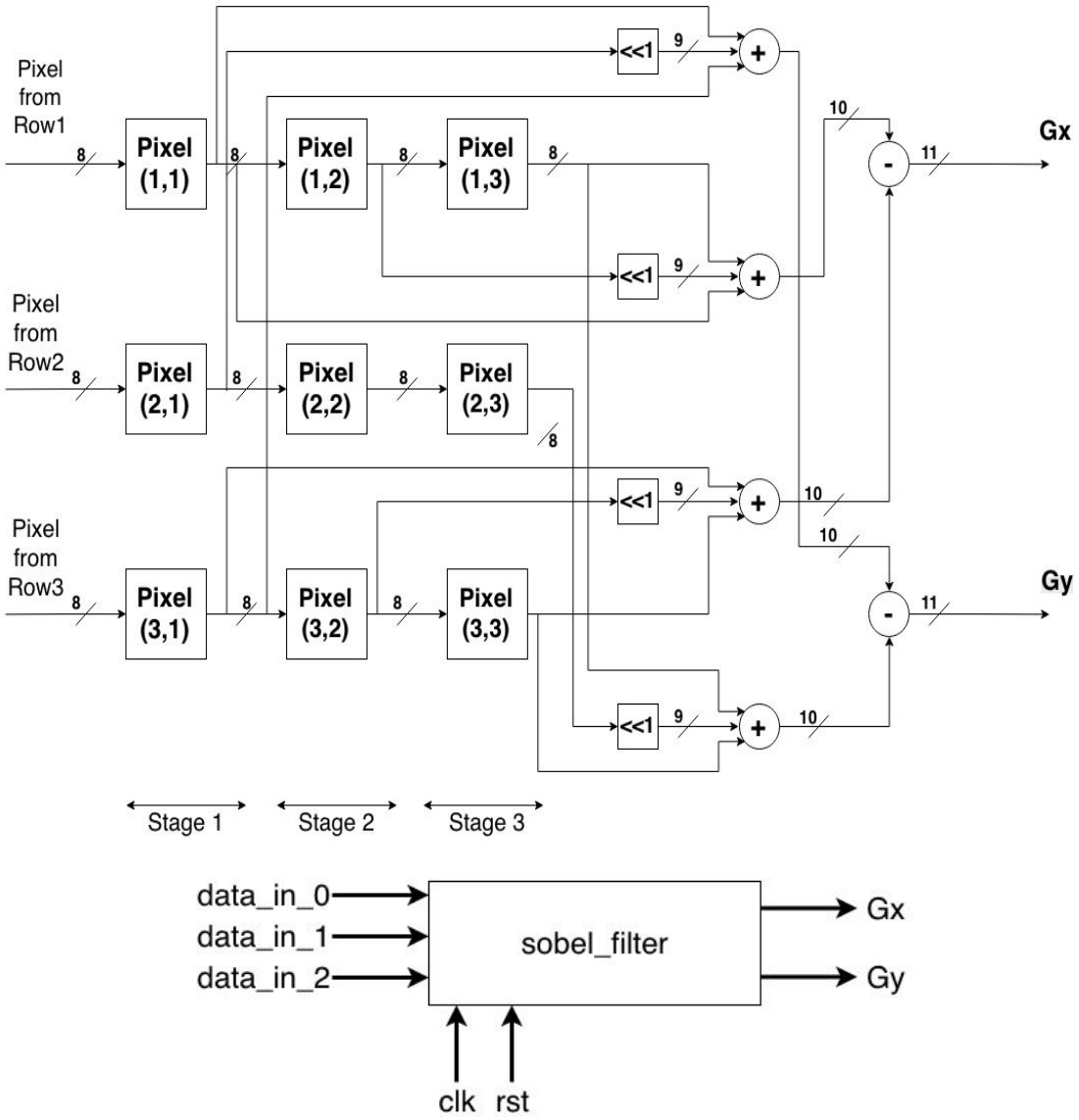

Figure 10 architecture and block diagram of sobel filter

1. Functionalities

This module efficiently performs a kernel convolution using the Gx and Gy Sobel masks shown below**.**

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Figure 11 matrices for sobel filter

The previous module, the row_shifter, provided 3 inputs to this module which stands for the coefficients for our the 3 rows. These 3 inputs are fed serially to our module, and are coefficients to a column of the Gx and Gy filters shown above. These 3 rows move serially into the Sobel masks in a pipeline fashion and the convolution is computed at each stage. These acts as if we were moving our filter into the picture matrix and computing the convolution matrix pixel for Gx and Gy.

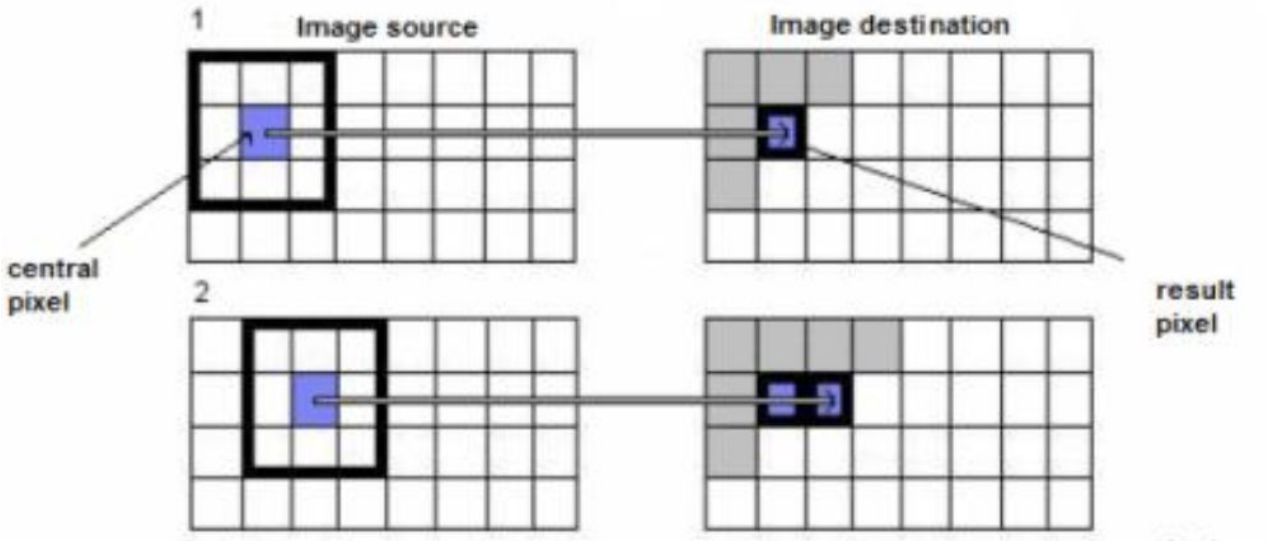

Figure 12 Image convolution with filter

The output of this module, at a particular stage, are the convolved pixel Gx and Gy for an input pixel.

Since the borders (the grey region in the picture above) does not have enough neighboring picture for the convolution mask, we treat the image as kind of a circular folded picture, with the first

column standing next to the last column. That is, for Gx and Gy in the last column of our output image, we use the pixels in the first column of the image as coefficients for the last column of our picture.

Also, since the magnitude of our coefficients are either 2 or 1, we can use the right shift operation instead of multiplication by 2 to reduce the overall area and improve performance. Therefore, our entire design won't have any multiplication operation.

2. Data flow

In this module, enough data for the 3*3 kernel multiplication is stored, that is 3*3bytes. The 3 input data received moves in a pipeline fashion from stage to stage as Gx and Gy are computed.

3. Algorithm

- The 3 inputs are stored and passed to our 1st stage of our pipeline.
- The values from our 1st stage are stored and passed to the 2nd stage.
- The values from the 2nd stage are stored and passed to the 3rd stage.
- Stored values from each stage are right shifted (multiplication by 2) and added and subtracted as required to produce Gx and Gy respectively.

4.4 uArch 4: Absolute_sum (absolute_sum.vhd)

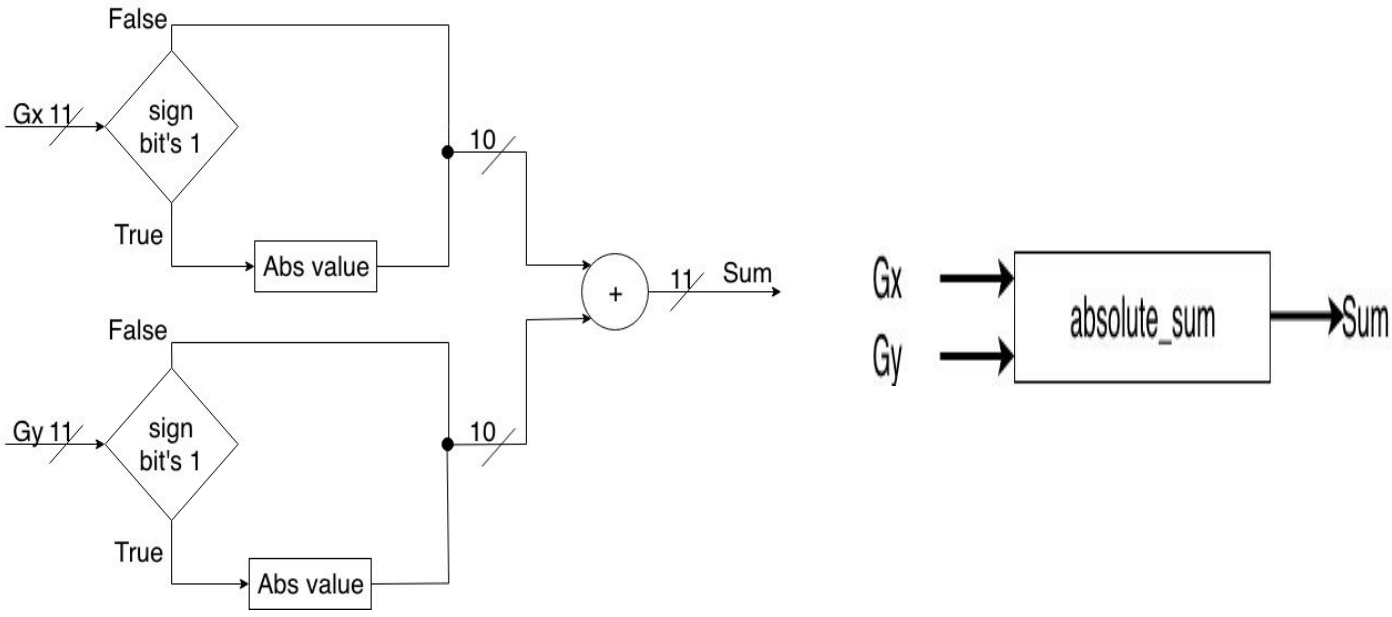

Figure 13 architecture and block diagram of absolute_sum module

This module is based on purely combinational logic, the 11 bit gradient values "Gx" and "Gy" are input to the module. This module calculates the SUM |Gx|+|Gy|, which is again an 11 bit value.
To calculate the absolute value if the sign bit is negative than |Gx|= (~Gx+1) else |Gx|=Gx.

The data widths are calculated using following logic. The value of Gx has a range from [-4*255 - 4*255] which requires minimum 11 bit to represent.The value of |Gx| has a range of [0 - 4*255] which requires minimum 10 bits. The value of |Gx|+|Gy| can have range from [0-8*255] so again requires minimum 11 bit to represent this number.

4.5 uArch 5: comparator and selection (comp_bin . vhd)

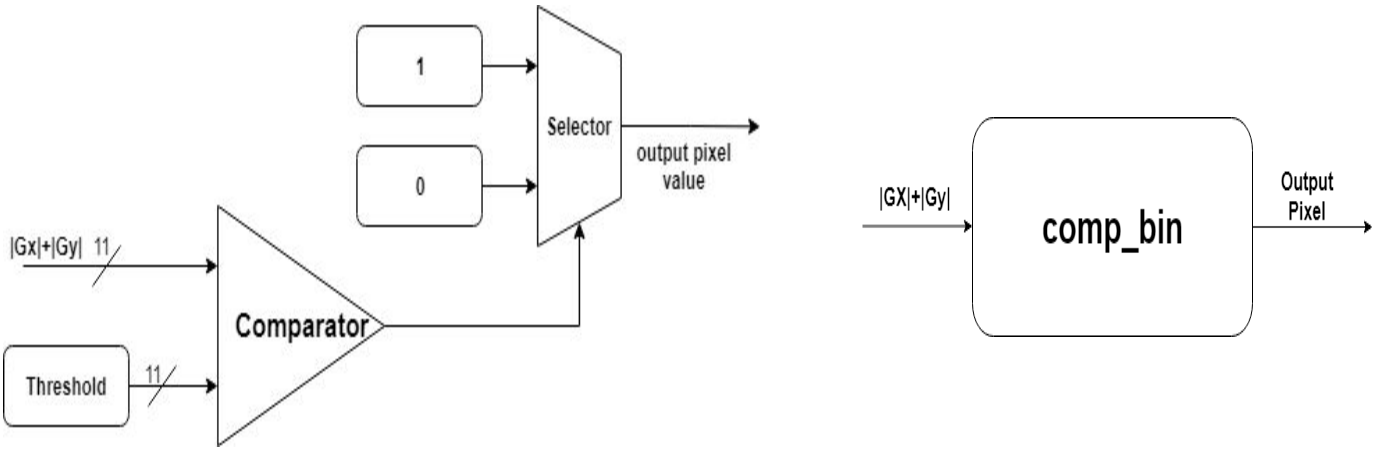

Figure 14 Architecture and block diagram for comparator and selection

This module compares the |Gx|+|Gy| with a user defined threshold value which can be changed at run time by user using switches [0-10] on the basys 3 board. Final output of this module is either a one or a zero, just a one bit answer which is written to the RAM on the appropriate address where out_valid signal is used to trigger the write. This module is purely based on combinational logic.

4.6 uArch 6: Dual Frequency RAM (RAM_dual.vhd)

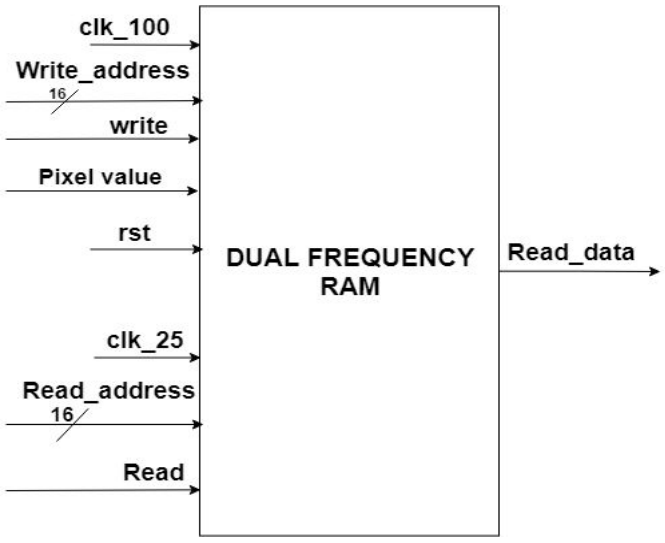

Figure 15 Dual frequency RAM

RAM used in this architecture stores one bit values and there are 256x256 such values each represent a pixel in binary image which contains edges, So size of RAM is 65536 bits. Continuous display on screen using VGA requires Image to be stored in ordered fashion. This RAM also solves the problem of frequency mismatch between Edge Detector module which produces pixel at 100 Mhz and VGA module that requires to operate on 25 Mhz to produce 60FPS display. Both Read and Write can occur simultaneously in this RAM. Every time a new image is sent to the edge detector module it overwrites the content of RAM with new binary image, the out_vallid signal of the “edge detector” module triggers the write operation of the binary pixel value in RAM. Phase Locked Loop present on the board is used to generate 25 Mhz Frequency. Data from this module is Read by the VGA controller .

4.7 uArch 7: VGA Module (vga_controller.vhd)

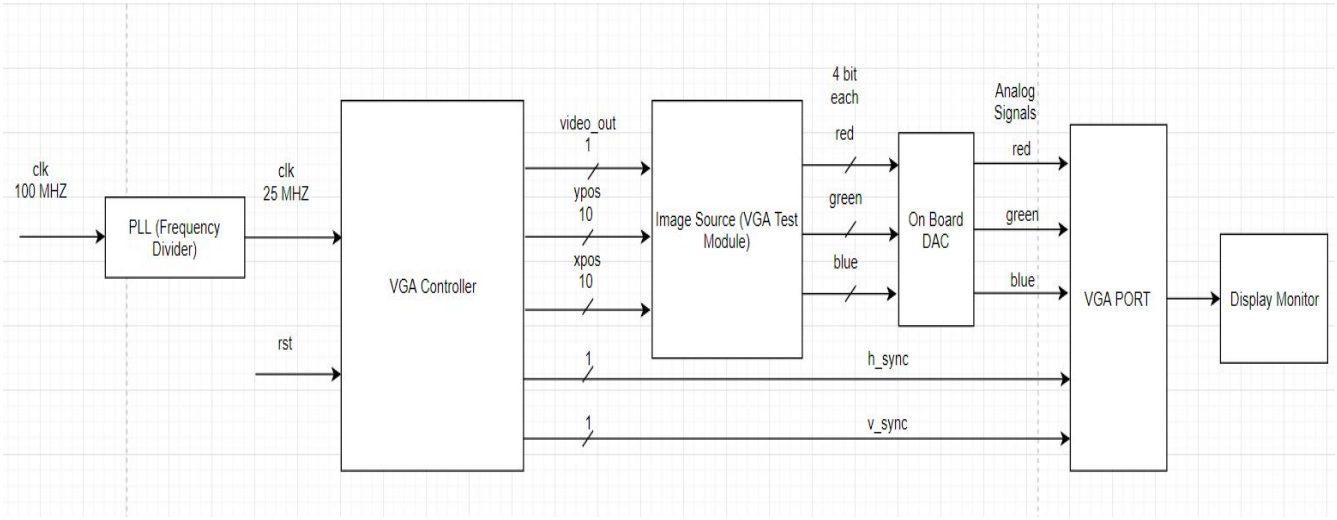

Figure 16 VGA module architecture

Video Graphics Array (VGA) is a graphics standard for video display controller first introduced with the IBM PS/2 line of computers in 1987, following CGA and EGA introduced in earlier IBM personal computers. Through widespread adoption, the term has also come to mean either an analog computer display standard, the 15-pin D-subminiature VGA connector, or the 640×480 resolution characteristic of the VGA hardware.

VGA is a standard interface for controlling analog monitors. The computing side of the interface provides the monitor with horizontal and vertical sync signals, color magnitudes, and ground references. The horizontal and vertical sync signals are 0V/5V digital waveforms that synchronize the signal timing with the monitor. Being digital, they are provided directly by the FPGA (3.3V meets the minimum threshold for a logical high, so 3.3V can be used instead of 5V). The color magnitudes are 0V-0.7V analog signals sent over the R, G, and B wires. (Alternatively, the green wire can use 0.3V-1V signals that incorporate both the horizontal and vertical sync signals, eliminating the need for those lines. This is called *sync-on-green* and is not used in our project.) To create these analog signals, the FPGA outputs each color bits to a video DAC. Through this on board DAC the digital signal are converted to analog and transmitted to the display monitor through the VGA port.

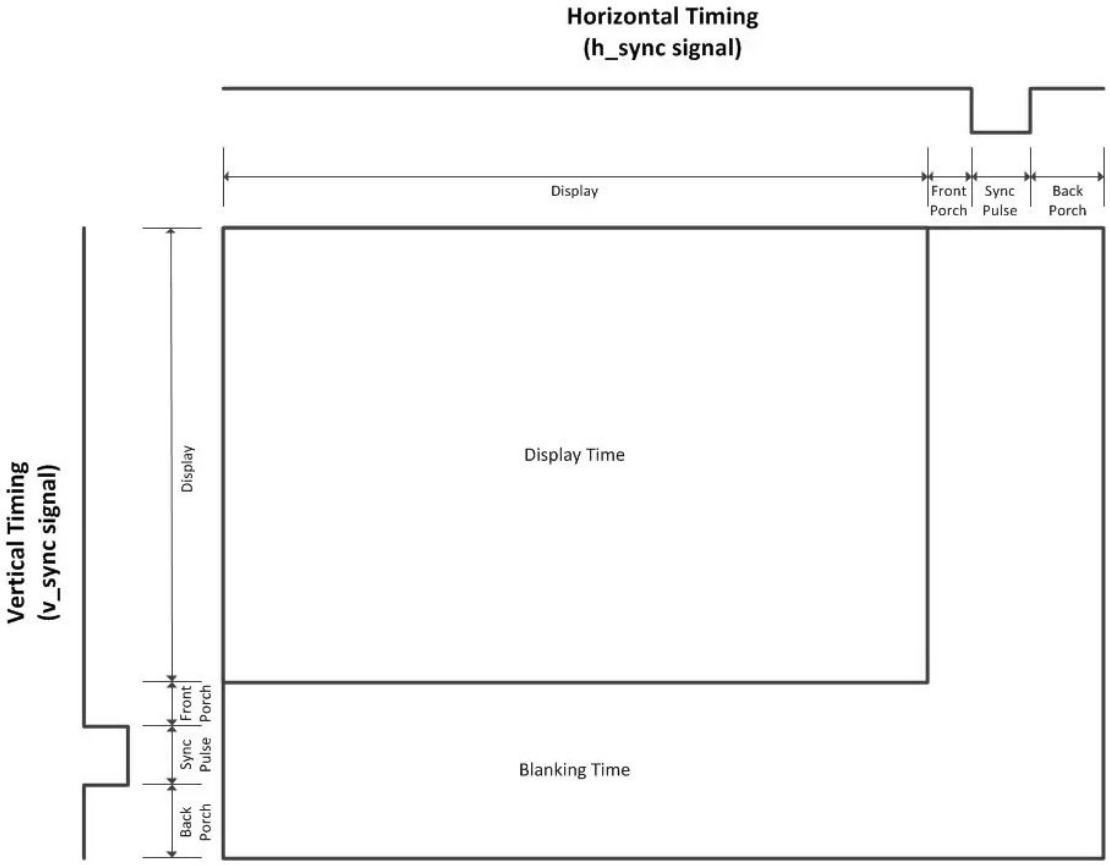

Figure 17 Timing for VGA module

For the display of 640x480 image size the following specification was used:-

Clock=25 MHz (Pixel Clock Rate)

Horizontal Timing(per pixel clock): Display: 640, Front Porch: 16, Sync Pulse: 96, Back Porch: 48

Vertical Timing(per pixel clock): Display: 480, Front Porch: 11, Sync Pulse: 2, Back Porch: 31

5 Implementation Battles and Solutions

5.1 Frequency Mismatch

Edge Detector module produces output at 100 Mhz, while the VGA controller can read data only at 25Mhz . So there was frequency mismatch between producer and consumer.

We Implemented a Dual Frequency RAM to solve this issue, this provides the edge detector module to write the RAM at 100 Mhz while allows the VGA Controller to read the data at 25Mhz simultaneously.

5.2 Setup Time violations

After integrating the overall architecture, the combinational logic of edge detector and address decoder logic for dual RAM created two paths which were not meeting the setup requirements. This happened because address decoding logic of such a large RAM(mainly this) along with combinational logic of edge detector module is difficult to meet in 100Mhz constraint.

We studied about different strategy to do synthesis and implementation, settled on following strategy.
Synthesis Strategy: “Alternate Routability” along with “resource reusability” off, because this prevents congestion on routes and decreases the net delay.
Implementation Strategy: Default Implementation strategy, These two together solved the problem of small setup violation in the design.

5.3 Invalid UART output

UART is used to ‘receive’ a byte serially as 8 bits which stands for a pixel in our picture. Though it rarely occurs, if this bit is invalid, we will have to ‘transmit’ a signal though our UART requesting the corrupted pixel should be resend. For this, we need to stall the rest of our design while waiting for the corrupted bit to be retransmitted. This added a lot more complexity to our design.

To avoid this, since most, if not all, of the input bits received are valid, we duplicated the previous pixel for invalid pixels.

5.4 Picture Border

When performing kernel convolution, every pixel should have some neighbor pixels on every side. However, border pixels do not have neighbors on all side. One way to go around this is to add zeros or duplicate the pixels the borders to fill the convolution, but this will increase storage. The way we solved this is by considering the picture as a circular folded picture. This way, the pixels computed pixels at the border maybe corrupted. However, there’s no storage or extra bit involved.

5.5 Transfer Speed with MATLAB

When transferring data from MATLAB to the board through UART, the process was extremely slow when transferring each pixel one at a time in a for loop; the overall transfer initially took about 6 minutes. Fortunately, we found a MATLAB function which was able to transfer multiple bits (up to 256) at a time through the UART and this greatly improved the transfer speed (down to less than 10 seconds).

5.6 Control signals

It was easy creating all modules individually and testing them but making them interact with each other seamlessly (as they do now) required a lot of work with the control signals (valid bits, ready bits, stall bits) between the different modules.

6 Testbench Details and Simulation Results

(Only write the system level testbenches and major testbenches) No need for UART

6.1 TB (tb_edge_detector.vhd)

This is a hybrid testbench which can test 2 features of the filter module: the single image mode and the image-after-image functionality. This testbench can take inputs of both kinds and simulates the results that would be obtained using the filter as a whole.

6.1.1 Input/Output File Format

The input for the single image mode is simply a single image converted to text. This can be done using MATLAB where all the pixels of an RGB image can be written to a file. Since the module also needs valid bits and end of file bits, these are also appropriately added in MATLAB. The test bench can then read this file and feed it into the module during simulation.

As for the image-after-image mode, the input is simply multiple images in a text file one after the other with the end of file appropriately placed (at the end of each image).

6.1.2 Patterns

There are two main “patterns” used to feed the input. The first pattern considers all the data to be valid while the second pattern has random invalid bits within the file. This is to test the control signals of the module and how it reacts to invalid input. Given that after every image the row buffers must be flushed, there is a stall at the end of each image to tell the feeder not to feed new images.

6.1.3 Simulation Results

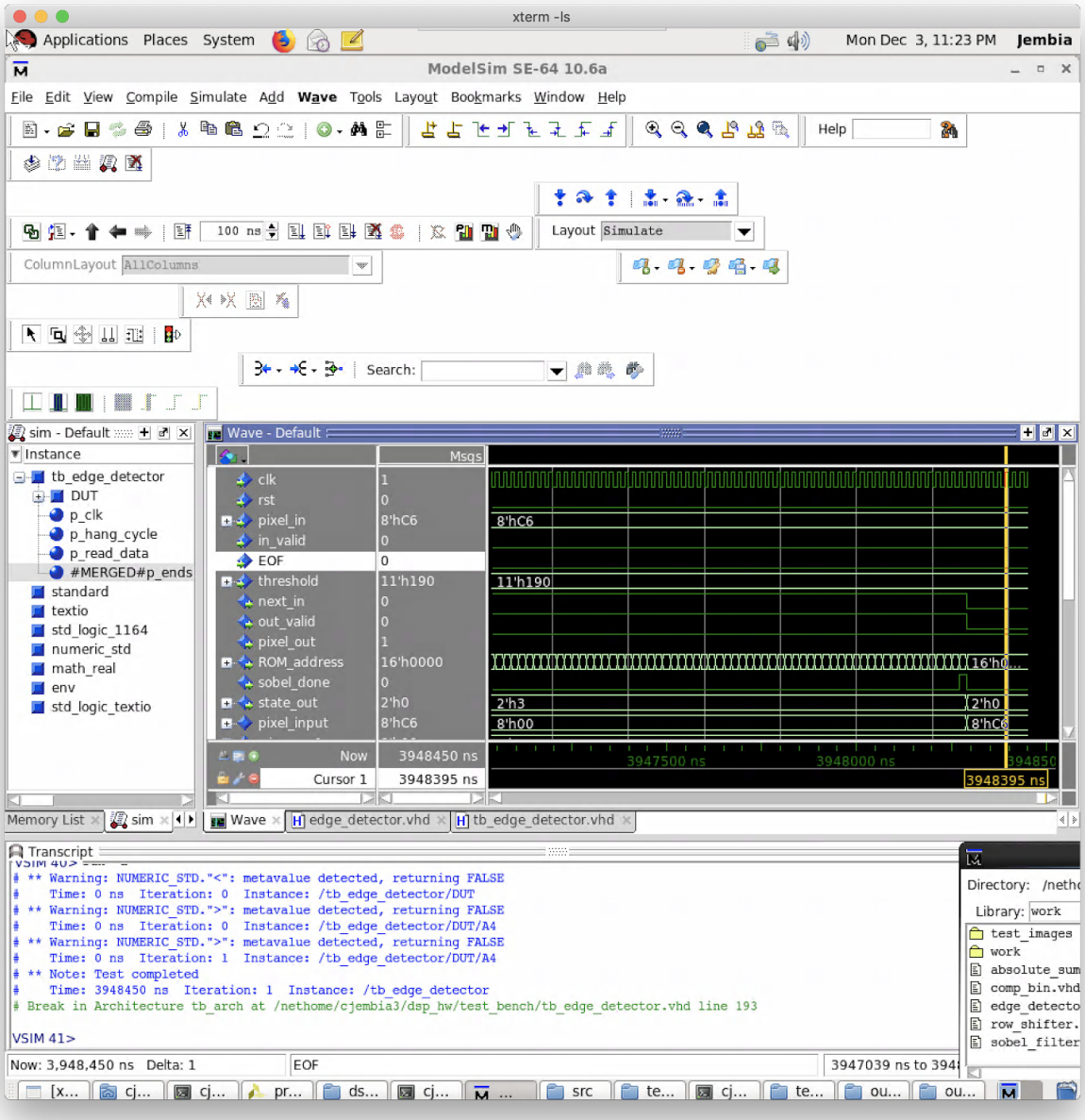

Figure 18 sample result of test bench

The diagram above shows the simulation results for the test bench of the filter module. It is difficult to show the whole functionality using just screenshots but in the segment above, it can be seen how after the end of the file, the next in goes low to indicate that data should not be sent after a whole image has been processed.

7 Implementation Results

SYSTEM LEVEL IMPLEMENTATION RESULTS

Area Utilization report :

Site Type	Used	Fixed	Available	Util%
Slice LUTs	517	0	20800	2.49
LUT as Logic	381	0	20800	1.83
LUT as Memory	136	0	9600	1.42
LUT as Distributed RAM	0	0		
LUT as Shift Register	136	0		
Slice Registers	494	0	41600	1.19
Register as Flip Flop	494	0	41600	1.19
Register as Latch	0	0	41600	0.00
F7 Muxes	0	0	16300	0.00
F8 Muxes	0	0	8150	0.00

Memory

Site Type	Used	Fixed	Available	Util%
Block RAM Tile	2	0	50	4.00
RAMB36/FIFO*	2	2	50	4.00
RAMB36E1 only	2			
RAMB18	0	0	100	0.00

Clocking :

Site Type	Used	Fixed	Available	Util%
BUFGCTRL	3	0	32	9.38
BUFIO	0	0	20	0.00
MMCME2_ADV	0	0	5	0.00
PLLE2_ADV	1	0	5	20.00
BUFMRCE	0	0	10	0.00
BUFHCE	0	0	72	0.00
BUFR	0	0	20	0.00

IO and GT specific

Site Type	Used	Fixed	Available	Util%
Bonded IOB	43	43	106	40.57
IOB Master Pads	20			
IOB Slave Pads	21			
Bonded IPADs	0	0	10	0.00
Bonded OPADs	0	0	4	0.00
PHY_CONTROL	0	0	5	0.00
PHASER_REF	0	0	5	0.00
OUT_FIFO	0	0	20	0.00
IN_FIFO	0	0	20	0.00
IDELAYCTRL	0	0	5	0.00
IBUFDS	0	0	104	0.00
GTPE2_CHANNEL	0	0	2	0.00
PHASER_OUT/PHASER_OUT_PHY	0	0	20	0.00
PHASER_IN/PHASER_IN_PHY	0	0	20	0.00
IDELAYE2/IDELAYE2_FINEDELAY	0	0	250	0.00
IBUFDS_GTE2	0	0	2	0.00
ILOGIC	0	0	106	0.00
OLOGIC	0	0	106	0.00

Primitives:

Ref Name	Used	Functional Category
FDRE	492	Flop & Latch
SRLC32E	136	Distributed Memory
LUT6	117	LUT
LUT3	91	LUT
LUT5	89	LUT
LUT2	48	LUT
LUT4	33	LUT
OBUF	26	IO
IBUF	13	IO
LUT1	5	LUT
OBUFT	4	IO
BUFG	3	Clock
RAMB36E1	2	Block Memory
PLLE2_ADV	1	Clock
FDSE	1	Flop & Latch
FDCE	1	Flop & Latch

Static and Dynamic Power :

Total On-Chip Power (W)	0.176
Dynamic (W)	0.104
Device Static (W)	0.072
Effective TJA (C/W)	5.0
Max Ambient (C)	84.1
Junction Temperature (C)	25.9
Confidence Level	Low
Setting File	---
Simulation Activity File	---
Design Nets Matched	NA

Design Time Summary:

Design Timing Summary

WNS(ns)	TNS(ns)	TNS Failing Endpoints	TNS Total Endpoints	WHS(ns)	THS(ns)	THS Failing Endpoints	THS Total Endpoints	WPWS(ns)	TPWS(ns)	TPWS Failing Endpoints	TPWS Total Endpoints
0.177	0.000	0	1574	0.052	0.000	0	1574	3.000	0.000	0	642

Row Shifter (Synthesis and Implementation results) :

1. Slice Logic

Site Type	Used	Fixed	Available	Util%
Slice LUTs	145	0	20800	0.70
LUT as Logic	9	0	20800	0.04
LUT as Memory	136	0	9600	1.42
LUT as Distributed RAM	0	0		
LUT as Shift Register	136	0		
Slice Registers	312	0	41600	0.75
Register as Flip Flop	312	0	41600	0.75
Register as Latch	0	0	41600	0.00
F7 Muxes	0	0	16300	0.00
F8 Muxes	0	0	8150	0.00

IO and GT specific :

Site Type	Used	Fixed	Available	Util%
Bonded IOB	35	1	106	33.02
IOB Master Pads	17			
IOB Slave Pads	17			
Bonded IPADs	0	0	10	0.00
Bonded OPADs	0	0	4	0.00
PHY_CONTROL	0	0	5	0.00
PHASER_REF	0	0	5	0.00
OUT_FIFO	0	0	20	0.00
IN_FIFO	0	0	20	0.00
IDELAYCTRL	0	0	5	0.00
IBUFDS	0	0	104	0.00
GTPE2_CHANNEL	0	0	2	0.00
PHASER_OUT/PHASER_OUT_PHY	0	0	20	0.00
PHASER_IN/PHASER_IN_PHY	0	0	20	0.00
IDELAYE2/IDELAYE2_FINEDELAY	0	0	250	0.00
IBUFDS_GTE2	0	0	2	0.00
ILOGIC	0	0	106	0.00
OLOGIC	0	0	106	0.00

Primitives :

Ref Name	Used	Functional Category
FDRE	312	Flop & Latch
SRLC32E	136	Distributed Memory
OBUF	24	IO
LUT2	17	LUT
IBUF	11	IO
BUFG	1	Clock

Power Summary :

1. Summary

Total On-Chip Power (W)	0.078
Dynamic (W)	0.008
Device Static (W)	0.070
Effective TJA (C/W)	5.0
Max Ambient (C)	84.6
Junction Temperature (C)	25.4
Confidence Level	Low
Setting File	---
Simulation Activity File	---
Design Nets Matched	NA

Timing Summary:

```
------------------------------------------------------------------------------------------------
| Design Timing Summary
| ---------------------
------------------------------------------------------------------------------------------------

    WNS(ns)      TNS(ns)  TNS Failing Endpoints  TNS Total Endpoints      WHS(ns)      THS(ns)  THS Failing Endpoints  THS Total
    Endpoints     WPWS(ns)     TPWS(ns)  TPWS Failing Endpoints  TPWS Total Endpoints
    -------      -------  --------------------  ------------------      -------      -------  --------------------
    ------------------     --------     --------  ----------------------  --------------------
      7.619       0.000                      0                  430       0.089       0.000
      0                 430       4.020       0.000                       0                 449
```

Sobel filter :

Slice Logic

```
1. Slice Logic
--------------

+-------------------------+------+-------+-----------+-------+
|        Site Type        | Used | Fixed | Available | Util% |
+-------------------------+------+-------+-----------+-------+
| Slice LUTs              |   61 |     0 |     20800 |  0.29 |
|   LUT as Logic          |   61 |     0 |     20800 |  0.29 |
|   LUT as Memory         |    0 |     0 |      9600 |  0.00 |
| Slice Registers         |   72 |     0 |     41600 |  0.17 |
|   Register as Flip Flop |   72 |     0 |     41600 |  0.17 |
|   Register as Latch     |    0 |     0 |     41600 |  0.00 |
| F7 Muxes                |    0 |     0 |     16300 |  0.00 |
| F8 Muxes                |    0 |     0 |      8150 |  0.00 |
+-------------------------+------+-------+-----------+-------+
```

IO and GT Specific

5. IO and GT Specific

Site Type	Used	Fixed	Available	Util%
Bonded IOB	49	1	106	46.23
IOB Master Pads	24			
IOB Slave Pads	24			
Bonded IPADs	0	0	10	0.00
Bonded OPADs	0	0	4	0.00
PHY_CONTROL	0	0	5	0.00
PHASER_REF	0	0	5	0.00
OUT_FIFO	0	0	20	0.00
IN_FIFO	0	0	20	0.00
IDELAYCTRL	0	0	5	0.00
IBUFDS	0	0	104	0.00
GTPE2_CHANNEL	0	0	2	0.00
PHASER_OUT/PHASER_OUT_PHY	0	0	20	0.00
PHASER_IN/PHASER_IN_PHY	0	0	20	0.00
IDELAYE2/IDELAYE2_FINEDELAY	0	0	250	0.00
IBUFDS_GTE2	0	0	2	0.00
ILOGIC	0	0	106	0.00
OLOGIC	0	0	106	0.00

Primitives

8. Primitives

Ref Name	Used	Functional Category
FDRE	72	Flop & Latch
LUT3	28	LUT
IBUF	27	IO
LUT2	25	LUT
LUT4	24	LUT
OBUF	22	IO
CARRY4	18	CarryLogic
LUT5	4	LUT
BUFG	1	Clock

Power Summary

1. Summary

Total On-Chip Power (W)	0.088
Dynamic (W)	0.018
Device Static (W)	0.070
Effective TJA (C/W)	5.0
Max Ambient (C)	84.6
Junction Temperature (C)	25.4
Confidence Level	Low
Setting File	---
Simulation Activity File	---
Design Nets Matched	NA

Timing summary

Design Timing Summary

WNS(ns)	TNS(ns)	TNS Failing Endpoints	TNS Total Endpoints	WHS(ns)	THS(ns)	THS Failing Endpoints
8.012	0.000	0	48	0.122	0.000	0

THS Failing Endpoints	THS Total Endpoints	WPWS(ns)	TPWS(ns)	TPWS Failing Endpoints	TPWS Total Endpoints
0	48	4.500	0.000	0	73

Absolute Sum :

Slice Logic:

Site Type	Used	Fixed	Available	Util%
Slice LUTs	24	0	20800	0.12
LUT as Logic	24	0	20800	0.12
LUT as Memory	0	0	9600	0.00
Slice Registers	0	0	41600	0.00
Register as Flip Flop	0	0	41600	0.00
Register as Latch	0	0	41600	0.00
F7 Muxes	0	0	16300	0.00
F8 Muxes	0	0	8150	0.00

IO and GT specific:

Site Type	Used	Fixed	Available	Util%
Bonded IOB	33	0	106	31.13
IOB Master Pads	16			
IOB Slave Pads	16			
Bonded IPADs	0	0	10	0.00
Bonded OPADs	0	0	4	0.00
PHY_CONTROL	0	0	5	0.00
PHASER_REF	0	0	5	0.00
OUT_FIFO	0	0	20	0.00
IN_FIFO	0	0	20	0.00
IDELAYCTRL	0	0	5	0.00
IBUFDS	0	0	104	0.00
GTPE2_CHANNEL	0	0	2	0.00
PHASER_OUT/PHASER_OUT_PHY	0	0	20	0.00
PHASER_IN/PHASER_IN_PHY	0	0	20	0.00
IDELAYE2/IDELAYE2_FINEDELAY	0	0	250	0.00
IBUFDS_GTE2	0	0	2	0.00
ILOGIC	0	0	106	0.00
OLOGIC	0	0	106	0.00

Primitives :

Ref Name	Used	Functional Category
IBUF	22	IO
OBUF	11	IO
LUT6	9	LUT
LUT5	7	LUT
LUT4	4	LUT
LUT3	4	LUT
CARRY4	3	CarryLogic
LUT2	1	LUT

Power Summary :

Total On-Chip Power (W)	9.413
Dynamic (W)	9.283
Device Static (W)	0.130
Effective TJA (C/W)	5.0
Max Ambient (C)	37.9
Junction Temperature (C)	72.1
Confidence Level	Low
Setting File	---
Simulation Activity File	---
Design Nets Matched	NA

Timing Summary : (Its purely combinational logic so timing details independently does not make sense)

Comparator and Selection: (Purely combinational Logic)

Slice Logic:

Site Type	Used	Fixed	Available	Util%
Slice LUTs	3	0	20800	0.01
LUT as Logic	3	0	20800	0.01
LUT as Memory	0	0	9600	0.00
Slice Registers	0	0	41600	0.00
Register as Flip Flop	0	0	41600	0.00
Register as Latch	0	0	41600	0.00
F7 Muxes	0	0	16300	0.00
F8 Muxes	0	0	8150	0.00

IO and GT Specific :

Site Type	Used	Fixed	Available	Util%
Bonded IOB	12	0	106	11.32
IOB Master Pads	5			
IOB Slave Pads	6			
Bonded IPADs	0	0	10	0.00
Bonded OPADs	0	0	4	0.00
PHY_CONTROL	0	0	5	0.00
PHASER_REF	0	0	5	0.00
OUT_FIFO	0	0	20	0.00
IN_FIFO	0	0	20	0.00
IDELAYCTRL	0	0	5	0.00
IBUFDS	0	0	104	0.00
GTPE2_CHANNEL	0	0	2	0.00
PHASER_OUT/PHASER_OUT_PHY	0	0	20	0.00
PHASER_IN/PHASER_IN_PHY	0	0	20	0.00
IDELAYE2/IDELAYE2_FINEDELAY	0	0	250	0.00
IBUFDS_GTE2	0	0	2	0.00
ILOGIC	0	0	106	0.00
OLOGIC	0	0	106	0.00

Primitives :

Ref Name	Used	Functional Category
IBUF	11	IO
OBUF	1	IO
LUT6	1	LUT
LUT5	1	LUT
LUT2	1	LUT

Power Summary:

Total On-Chip Power (W)	0.373
Dynamic (W)	0.302
Device Static (W)	0.071
Effective TJA (C/W)	5.0
Max Ambient (C)	83.1
Junction Temperature (C)	26.9
Confidence Level	Low
Setting File	---
Simulation Activity File	---
Design Nets Matched	NA

Timing: Again this module is based purely on combinational logic so timing does not make sense, unless integrated.

Dual Frequency RAM :

Area utilization report:

Slice Logic:

Site Type	Used	Fixed	Available	Util%
Slice LUTs	5	0	20800	0.02
LUT as Logic	5	0	20800	0.02
LUT as Memory	0	0	9600	0.00
Slice Registers	1	0	41600	<0.01
Register as Flip Flop	1	0	41600	<0.01
Register as Latch	0	0	41600	0.00
F7 Muxes	0	0	16300	0.00
F8 Muxes	0	0	8150	0.00

Memory:

Site Type	Used	Fixed	Available	Util%
Block RAM Tile	2	0	50	4.00
RAMB36/FIFO*	2	0	50	4.00
RAMB36E1 only	2			
RAMB18	0	0	100	0.00

IO and GT Specific :

Site Type	Used	Fixed	Available	Util%
Bonded IOB	38	1	106	35.85
IOB Master Pads	17			
IOB Slave Pads	21			
Bonded IPADs	0	0	10	0.00
Bonded OPADs	0	0	4	0.00
PHY_CONTROL	0	0	5	0.00
PHASER_REF	0	0	5	0.00
OUT_FIFO	0	0	20	0.00
IN_FIFO	0	0	20	0.00
IDELAYCTRL	0	0	5	0.00
IBUFDS	0	0	104	0.00
GTPE2_CHANNEL	0	0	2	0.00
PHASER_OUT/PHASER_OUT_PHY	0	0	20	0.00
PHASER_IN/PHASER_IN_PHY	0	0	20	0.00
IDELAYE2/IDELAYE2_FINEDELAY	0	0	250	0.00
IBUFDS_GTE2	0	0	2	0.00
ILOGIC	0	0	106	0.00
OLOGIC	0	0	106	0.00

Primitives :

Ref Name	Used	Functional Category
IBUF	37	IO
LUT3	3	LUT
BUFG	3	Clock
RAMB36E1	2	Block Memory
PLLE2_ADV	1	Clock
OBUF	1	IO
LUT2	1	LUT
LUT1	1	LUT
FDCE	1	Flop & Latch

Power Summary (Static and Dynamic Power):

Total On-Chip Power (W)	0.173
Dynamic (W)	0.103
Device Static (W)	0.071
Effective TJA (C/W)	5.0
Max Ambient (C)	84.1
Junction Temperature (C)	25.9
Confidence Level	Low
Setting File	---
Simulation Activity File	---
Design Nets Matched	NA

Timing Summary :

Design Timing Summary

WNS(ns)	TNS(ns)	TNS Failing Endpoints	TNS Total Endpoints	WHS(ns)	THS(ns)	THS Failing Endpoints	THS Total Endpoints	WPWS(ns)	TPWS(ns)	TPWS Failing Endpoints	TPWS Total Endpoints
37.867	0.000	0	1	0.451	0.000	0	13	1	3.000	0.000	0

VGA Module:

Slice Logic:

Site Type	Used	Fixed	Available	Util%
Slice LUTs	58	0	20800	0.28
LUT as Logic	58	0	20800	0.28
LUT as Memory	0	0	9600	0.00
Slice Registers	36	0	41600	0.09
Register as Flip Flop	36	0	41600	0.09
Register as Latch	0	0	41600	0.00
F7 Muxes	0	0	16300	0.00
F8 Muxes	0	0	8150	0.00

Memory:

Site Type	Used	Fixed	Available	Util%
Block RAM Tile	0	0	50	0.00
RAMB36/FIFO*	0	0	50	0.00
RAMB18	0	0	100	0.00

IO and GT Specifics:

Site Type	Used	Fixed	Available	Util%
Bonded IOB	35	1	106	33.02
IOB Master Pads	17			
IOB Slave Pads	17			
Bonded IPADs	0	0	10	0.00
Bonded OPADs	0	0	4	0.00
PHY_CONTROL	0	0	5	0.00
PHASER_REF	0	0	5	0.00
OUT_FIFO	0	0	20	0.00
IN_FIFO	0	0	20	0.00
IDELAYCTRL	0	0	5	0.00
IBUFDS	0	0	104	0.00
GTPE2_CHANNEL	0	0	2	0.00
PHASER_OUT/PHASER_OUT_PHY	0	0	20	0.00
PHASER_IN/PHASER_IN_PHY	0	0	20	0.00
IDELAYE2/IDELAYE2_FINEDELAY	0	0	250	0.00
IBUFDS_GTE2	0	0	2	0.00
ILOGIC	0	0	106	0.00
OLOGIC	0	0	106	0.00

Clocking:

Site Type	Used	Fixed	Available	Util%
BUFGCTRL	1	0	32	3.13
BUFIO	0	0	20	0.00
MMCME2_ADV	0	0	5	0.00
PLLE2_ADV	0	0	5	0.00
BUFMRCE	0	0	10	0.00
BUFHCE	0	0	72	0.00
BUFR	0	0	20	0.00

Primitives:

Ref Name	Used	Functional Category
FDRE	36	Flop & Latch
OBUF	31	IO
LUT5	23	LUT
LUT6	21	LUT
LUT4	13	LUT
LUT3	5	LUT
LUT2	5	LUT
IBUF	4	IO
CARRY4	4	CarryLogic
LUT1	2	LUT
BUFG	1	Clock

Design Timing Summary:

Design Timing Summary

WNS(ns)	TNS(ns)	TNS Failing Endpoints	TNS Total Endpoints	WHS(ns)	THS(ns)	THS Failing Endpoints
4.981	0.000	0	62	0.191	0.000	0

THS Total Endpoints	WPWS(ns)	TPWS(ns)	TPWS Failing Endpoints	TPWS Total Endpoints
62	4.500	0.000	0	37

Static and Dynamic Power :

Total On-Chip Power (W)	0.073
Dynamic (W)	0.003
Device Static (W)	0.070
Effective TJA (C/W)	5.0
Max Ambient (C)	84.6
Junction Temperature (C)	25.4
Confidence Level	Medium
Setting File	---
Simulation Activity File	---
Design Nets Matched	NA

8 Work Distribution

(Use "Shift+Enter" for a new line; use "Tab" key to format. Use "Enter" for a new item (bullet).)

- **Overall architecture design (Member 1)**
- **Scripts & golden references**
  input_maker.m (Input Test Vector generation | Clancy Jembia)
  output_maker.m (Output reference generation | Clancy Jembia)
  script_for_mif_gen.m (script for Memory Initialization file| Clancy Jembia)
  script_for_coe_gen.m(script for IP based Memory Initialization file| Clancy Jembia)

- **script for how our hw should behave**
  HWlikeSobel.m (Steve Medie)

- **script for preprocessing of image and sending it to FPGA board via UART**
  UART_send.m (Steve Medie)
- **comparison for different edge detection algorithms**
  Prewitt_local.m,Prewitt_edge.m,Sobel_local.m,Sobel_edge.m(Abhinav Himanshu)
- **script for preprocessing of image and sending it to FPGA board via UART**
  UART_send.m (Steve Medie)

- **Design files**

- **UART**
  myreceiver.vhd (Steve Medie, Clancy Jembia)
- **Top module of architecture**
  Our_design.vhd (Suvrat Krishna Mishra)
- **Top Module for Edge Detector**
  edge_detector.vhd (Clancy Jembia : Dataflow, Abhinav Himanshu : controls)
- **Row Shifter**
  row_shifter.vhd (Abhinav Himanshu)
- **Sobel Filter**
  soble_filter.vhd (Steve Medie)
- **Absolute_sum.vhd**
  absolute_sum.vhd(Abhinav Himanshu)
- **Comparator and Selection**
  comp_bin.vhd (Abhinav Himanshu)

- **Dual Frequency RAM**
  RAM_dual.vhd(Clancy Jembia, Abhinav Himanshu)

- **VGA Module**
  top_vga.vhd(Suvrat Krishna Mishra) vga_controller.vhd (Suvrat Krishna Mishra)
  vga_test.vhd(Suvrat Krishna Mishra)

- **Verification files**
  tb_edge_detector.vhd (Clancy Jembia)
- **Solutions for Implementation battles**
  - **Frequency Mismatch (Abhinav Himanshu)**
  - **Setup Time Violation (Abhinav Himanshu)**
  - **Invalid UART output (Steve Medie)**
  - **Picture Border (Steve Medie)**
  - **Transfer Speed with Matlab (Clancy Jembia)**
  - **Control signals (Clancy Jembia)**
- **System-level integration**
  - **Integration of UART with Edge Detector (Steve Medie)**
  - **Integration of Edge detector with RAM by writing address correctly (Abhinav Himanshu)**
  - **Integration of VGA with RAM by timing read address correctly (Clancy Jembia)**

9 References

[1]Rafael C. Gonzalez, R.E. Woods, Digital Image Processing, third edition, pp. 700-702, 2008.

[2] Ramesh Jain, Rangachar Kasturi, Brian G. Schunck ,“Machine Vision” published by McGraw-Hill, Inc., ISBN 0-07-032018-7, 1995

[3]Schiel, Jamie & Bainbridge-Smith, Andrew. (2015). Efficient Edge Detection on Low-Cost FPGAs. : https://arxiv.org/ftp/arxiv/papers/1512/1512.00504.pdf

[4]Scott Larson, VGA Controller : https://www.digikey.com/eewiki/pages/viewpage.action?pageId=15925278

[5]Girish Chap Ie and R. D. Daruwala. “Design of Sobel Operator based Image Edge Detection Algorithm on FPGA”. International Conference on Communication and Signal Processing, April 3-5, 2014, India

[6]V. Torre and T. A. Poggio. “On edge detection”. IEEE Trans. Pattern Anal. Machine Intell., vol. PAMI-8, no. 2, pp. 187-163, Mar. 1986.

[7]Code and references from Instructor and TA of course.