

**School of Electronics and Electrical Engineering**

**Faculty of Engineering and Physical Sciences**

**UNIVERSITY OF LEEDS**

**ELEC5566M – FPGA Design for System on Chip**

**Mini Project – FPGA based Edge Detection**

**ARUL PRAKASH SAMATHUVAMANI**

**201460156**

## Abstract

In this project, we designed an FPGA hardware implementation of Edge Detection using Sobel Operator. We have also demonstrated the use of FPGA as hardware accelerator to offload compute intensive tasks from General Purpose Processor. We have developed a fully parameterised IP core for edge detection using Sobel Operator. Hardware testing and demonstration on input 100x100 input image is performed on DE1-SoC board with Cyclone V FPGA. Testing methodologies for verifying correct working of the module is discussed and additional methodologies for further testing and improvements is also discussed. Increase in performance when using an FPGA is analysed.

**Keywords:** Edge detection, Sobel operator, FPGA, General Purpose Processor

## 1. Introduction

More and more compute intensive tasks are being performed in RISC based processors as more and more connected and smart devices designed and developed. One of the main challenges of embedded designers is to find trade-off between power and performance [1]. There is a growing demand for hardware accelerators to offload specific compute intensive tasks off from General Purpose CPUs. FPGAs can be great hardware accelerators as they can be designed to a gate-level and can be optimised to perform compute intensive tasks. ARM processors can be integrated with FPGA to offload compute intensive tasks and increase performance. With usage of RISC processors in data centres and machine learning embedded processors, compute intensive tasks such as Cyclic Redundancy Check (CRC) and entire TCP/IP stack can be offloaded to FPGA to increase performance [2].

Image Processing tasks require large amount of computing power. Edge detection is one of important tasks in image processing and is helpful in image segmentation and feature extraction. Real Time edge detection require large amount of computing resources and FPGA can be used as hardware accelerators to increase performance without much compromise in power consumption. In this report, we have discussed development and testing of fully parameterised IP core to perform edge detection using Sobel Operator. The module is demonstrated on DE1-SoC with Altera Cyclone V FPGA.

The first part of the report covers mathematical background behind Sobel Operator. The second part of the report covers basic information about DE1 SoC board and use of hardware components for demonstrating this module. Third, technical explanation of the design is covered along with testing of the module. Finally, performance improvements on using an FPGA are explained along with further testing and improvements that can be made to this IP core.

## 2. Edge Detection using Sobel Operator

Edges in image are areas with a contrasting jump in intensity from one pixel to next. Edges in image are essentially borders in objects present in an image. This edge detection would filter out unneeded information from an image while preserving important properties of an image.

There are variety of methods for edge detection in an image. They can be grouped into two categories.

1. Gradient
2. Laplacian

The Gradient method detects the edges by looking for maximum and minimum in first the derivative of the image [3]. There are various gradient operators such as Sobel, Roberts, Laplace operators. Sobel operator is one of the most popular edge detection operators.

Sobel edge detection involves use of two 3x3 convolution kernels. One kernel detects change in horizontal direction while the other kernel detects change in vertical direction. One kernel is 90-degree rotation of the other. These kernels are designed to respond to kernels running vertically and horizontally.

The horizontal kernel is given by,

$$\begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$$

The vertical kernel which is 90-degree rotation of horizontal kernel is given by,

$$\begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

This kernel function is convolved with input pixel P to get the approximate horizontal and vertical gradient  $G_x$  and  $G_y$ .

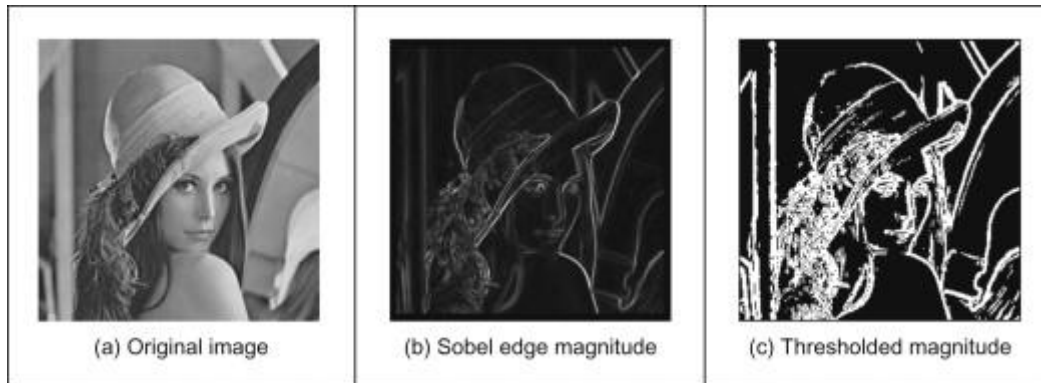


Figure1. Sobel Edge magnitude and Sobel Edge Threshold magnitude.

Consider an input pixel P for which gradient is to be calculated, the pixels surrounding the input pixel P can be denoted as,

$$\begin{array}{ccc} P1 & P2 & P3 \\ P4 & P & P6 \\ P7 & P8 & P9 \end{array}$$

The approximate horizontal and vertical gradient  $G_x$  and  $G_y$  can be calculated as [4],

$$|G_x| = | (P1 + 2 \times P2 + P3) - (P7 + 2 \times P8 + P9) | \quad (1)$$

$$|G_y| = | (P3 + 2 \times P6 + P9) - (P1 + 2 \times P4 + P7) | \quad (2)$$

The gradient magnitude is given by,

$$|G| = \sqrt{G_x^2 + G_y^2} \quad (3)$$

To reduce computational complexity, the approximate gradient magnitude can be given by

$$|G| = |G_x| + |G_y| \quad (4)$$

In Sobel edge detection with threshold magnitude, if the calculated approximate gradient magnitude is less than the given threshold value, then the value of the gradient is set to zero.

### 3. Background Discussion on Hardware Components

Terasic DE1-SoC is a powerful and resourceful board useful to implement some interesting projects. DE1 – SoC contains Hard Processor System (HPS) with ARM A9 processor coupled with Cyclone V FPGA. It is abundant with DDR3 memory, video, audio capabilities, Ethernet and USB port. It contains slide switch and push buttons to interact with the board. Simple information such as letters and numbers can be displayed using the in-built 7-segment display. Using GPIO pins in DE1-SoC, LT24, an LCD touch panel with 240 (H) x 320 (V) resolution.

The slide button in DE1 SoC board is used to switch between various operation modes. The push button is used to calculate input to be fed into LT24 display and refresh the display with the new information.

Following are the modes in Sobel Edge IP core module.

Slide Switch	Corresponding Mode	Description of Mode
No switch is ON	LOGO_STATE	Display 'hello' message on 7-segment display and Leeds Logo on LT24 display.
slide_switch0	GRAY_STATE	Display 'gray' on 7-segment display and input gray scale image on LT24 display.
slide_switch1	SOBEL_STATE Thresholding Disabled	Display 'sobel' on 7-segment display and apply sobel operation on input image and display result in LT24 display.
slide_switch2	SOBEL_STATE Thresholding Enabled	Display 'thr xxx' on 7-segment display where xxx is current threshold value used for calculation. Sobel operation with threshold approximation is applied to the input image and result is displayed on LT24 display.

Table1. Slide Switches and corresponding modes.

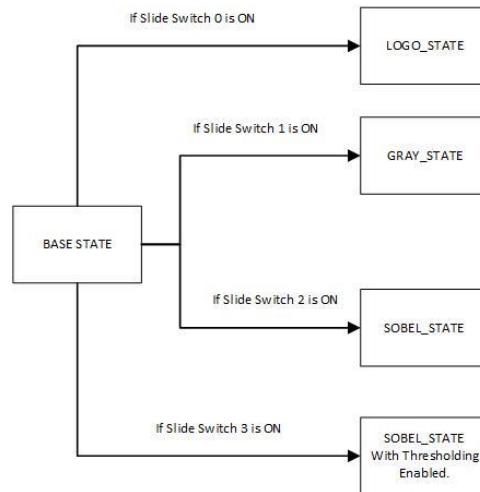


Figure2. Slide Button and corresponding modes in IP core.

When thresholding mode is ON, slide button 4 to slide button 9 is used to change the threshold values. By default, the threshold value is set to 50.

Slide Switch	Threshold Value
Slide_switch3	100
Slide_switch4	120
Slide_switch5	140
Slide_switch6	160
Slide_switch7	180
Slide_switch8	200
Slide_switch9	220

Table2. Slide Switch and corresponding Threshold Values

## 4. Technical Discussion on Sobel Edge IP Core

### 4.1 Displaying Image on LT24 Display

Cyclone V contains M10K or MLAB memory modules as single port or dual port RAM. They can be used as lookup tables or to store information for the application like input image as per application needs. M10K are dedicated memory resources and useful for large memory arrays. MLABs are enhanced memory block derived from Logic Block Arrays (LABs) [4].

Large memories can be initialised using Intel Hex (.hex) or Memory Initialisation Files (.mif). Memory contents are initialised as RAM slices in Analysis and Synthesis, and M10K memory blocks are assigned to RAM slices during Fitting.

M10K memory blocks can be either single port, simple dual port or dual port. Single port contains one reference clock with one address, one input and output data bus. Dual port memory block contains two input, two output and two address busses. M10K provides synchronous read/write logic, but we would require asynchronous read logic for our application. By bypassing output register, asynchronous read is possible in M10K memory

block. We would not be selecting if we would used MLABs or M10K memory blocks, it is done automatically by Quartus during fitting.

```
// Declaration of registers for input image, loaded using .mif Quartus file

(*ram_init_file = "data.mif"*) // Input 100x100 image -> default
reg [ 7 : 0 ] input_image [ ( ( IMAGE_WIDTH ) * ( IMAGE_HEIGHT ) - 1 ) : 0 ];

(*ram_init_file = "mod2.mif"*) // Input 100x320 Leeds Logo Image
reg [ 7:0 ] logo [ 31999 : 0 ] ;
```

Figure 3. Initialising MIF files in Quartus

We would be initialising input gray scale image with help of Memory Initialisation File (.mif). The contents to be loaded into the memory is stored in the initialisation file. The Memory Initialisation File is generated with help of MATLAB and the given MATLAB code in ELEC5566M Resources for generating MIF files.

Input JPEG image is read into MATLAB variable using imread() , an inbuilt function in MATLAB. The read variable is of size IMAGE\_WIDTH x IMAGE\_HEIGHT x 3 containing 8-bit RGB pixel information. The input image is then converted into gray scale image using function rgb2gray(). This function returns an array of size, IMAGE\_WIDTH x IMAGE\_HEIGHT containing a single 8-bit pixel information. This 8-bit pixel is average of RGB pixel values.

### RGB565 16 Bit Colour Code



Figure 4. RGB 565 Colour Code

LT24 display module requires input data to be present in RGB 565 format. 5-bits are used for Red and Blue while 6-bits are used to represent Green as our eyes are more sensitive to green. 31 is the maximum value for Red and Blue pixel in decimal format. While 63 is the maximum value for green in decimal format.

To display our gray scale image in LT24 display, we would need to convert the 8-bit image to size of 5 bits for Red and Blue and 6 bits for Green. For example, 255 is used to represent white in 8-bit size, we would need to push 31 to both Red and Blue pixels and 63 to Green pixel to display white colour in LT24 display. To perform the following size conversion, we would need to right shift the input pixel by 2 which is equivalent to dividing the input pixel by 8.

Consider an input pixel of 255 represented by 11111111 in binary format. Left shifting the input pixel by 2 results in 11111100. The bits 1-5 (size 5) is fed into Red and Blue. The bits 0-5 (size 6) is fed into green. The resultant value 11110 – 30 in decimal is fed to Red and Blue while 111100 - 60 in decimal is fed to Green which approximately produces White in LT24 display.

## 4.2 Sobel Edge Detection IP Core using Verilog HDL

The module for Sobel Edge Detection has 2 parameters, one for Image Width and another for image height. The module for interfacing LT24 display is used with this project. Though the module would work regardless of the image width and height, it is recommended to use images that are smaller than 240 x 320 resolution. Also, resources in a FPGA is limited. Hence, smaller images are recommended to be loaded into FPGA as .mif files. Alternatively, a new module can be designed to interface with Linux running on HPS. The main aim in design of this module is to design a Sobel edge detector running on minimal number of clock cycle as possible.

The Sobel edge detector has following states. The state machine jumps from one state to another during every clock cycle.

**SOBEL\_INITIAL** – Used to set x and y coordinates for fetching image from the registers as zero. This is the state the machine starts from when SOBEL\_MODE is enabled.

P is the current pixel for which approximate gradient is calculated. The other pixels used for calculation is denoted by,

$$\begin{array}{ccc} P1 & P2 & P3 \\ P4 & P & P6 \\ P7 & P8 & P9 \end{array}$$

**SOBEL\_STATE1** – Checks which pixel is currently under operation. If corner pixels, i.e pixels found in the borders is under operation, the value of the gradient is set to zero and the state machine jumps from SOBEL\_STATE1 to SOBEL\_STATE5 which feeds the pixel data to LT24 display.

Else, the pixel required to calculate approximate gradient is fetched from the memory using the following states. The state machine jumps from one state to another during every positive edge of clock.

**SOBEL\_ADDRESS1** – Used to fetch pixel P1 from memory.

**SOBEL\_ADDRESS2** – Used to fetch pixel P2 from memory.

**SOBEL\_ADDRESS3** – Used to fetch pixel P3 from memory.

**SOBEL\_ADDRESS4** – Used to fetch pixel P4 from memory.

**SOBEL\_ADDRESS5** – Used to fetch pixel P6 from memory.

**SOBEL\_ADDRESS6** – Used to fetch pixel P7 from memory.

**SOBEL\_ADDRESS7** – Used to fetch pixel P8 from memory.

**SOBEL\_ADDRESS8** – Used to fetch pixel P9 from memory.

After fetching the pixels for calculation, the state machine jumps to SOBEL\_STATE2.

**SOBEL\_STATE2** – Used to calculate the value of  $(P_1 + 2 \times P_2 + P_3)$ ,  $(P_7 + 2 \times P_8 + P_9)$ ,  $(P_3 + 2 \times P_6 + P_9)$ ,  $(P_1 + 2 \times P_4 + P_7)$ . The resultant of the values are stored in a 9-bit temporary variable. The maximum value of the operation can be 255 which is the highest value of a 8-bit

pixel. Hence if the MSB of the temporary variable is 1, it means that the value is greater than 255. The value of the temporary variable is set to 9'd255.

**SOBEL\_STATE3** – Used to calculate the value of  $G_x$  and  $G_y$  as per formulas (1) and (2). The resultant of the value cannot be negative, and zero is the minimum value of the operation as the lowest value of a 8-bit pixel is zero. Thus, the subtraction of the two values is set to a 9-bit variable. If the MSB of the temporary variable is '1', it means that the subtraction value is less than zero and should be set to 9'd0.

**SOBEL\_STATE4** – After calculating the value of  $G_x$  and  $G_y$  the value of  $G$ , the approximate gradient is calculated using formula (4). The maximum value of the operation can be 255 as maximum value of a 8-bit pixel can be 255. Hence the value of the addition is stored in a 9-bit temporary variable. If MSB of the variable is 1, it would essentially mean that the value of addition is higher than 255 and is set to 9'd255.

**SOBEL\_STATE5**- After calculating the approximate gradient, the 8-bit pixel value is converted to RGB 565 format and passed on to LT24 display. After calculation the state machine jumps to SOBEL\_STATE1.

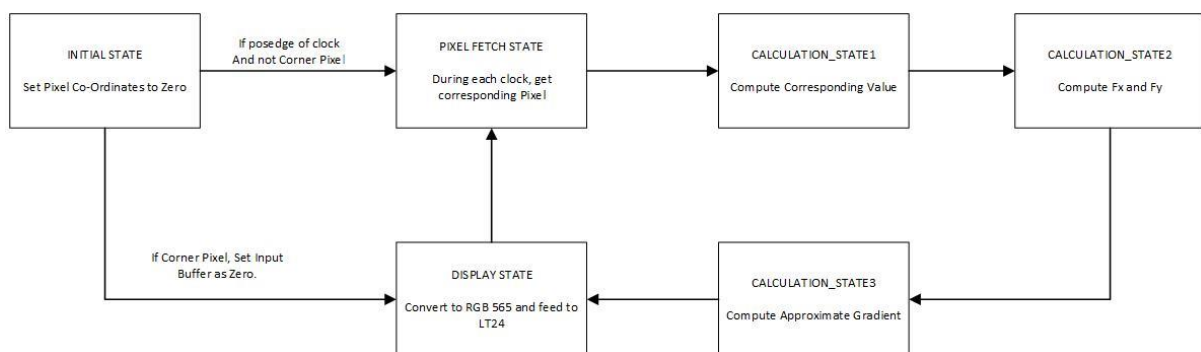


Figure 5. Sobel State Diagram – High Level

A register 'threshold\_enable' is used to check if threshold mode is ON or OFF. When slide switch 2 is turned ON, then the register 'threshold\_enable' is set to HIGH. When threshold\_enable is HIGH, SOBEL\_STATE4 checks if the approximate gradient calculated is higher or lower than threshold value. If threshold value is higher than approximate gradient, the value is retained. Else, the value of the approximate gradient is set to zero.

The threshold value can be controlled by using slide switches 3-9. By default, threshold value is set as 50.

Also, in each state we need the design to be asynchronous and sequential, as output from first step is fed as input to next. Thus, asynchronous design logic is used in Sobel Edge operation. It should be noted that asynchronous design logic can be buggy at higher clock frequencies.

### 4.3 Seven Segment Display

Seven segment display is used to display the mode of operation.

“hello” is displayed in LOGO\_STATE.

“gray” is display in GRAY\_STATE.



“EDGE” is displayed in SOBEL\_STATE when thresholding is disabled.

“THR xxx” is displayed in SOBEL\_STATE when thresholding is enabled. Xxx corresponds to current threshold value. Currently, the value to be displayed in seven segment display is fed manually, meaning the 7-bit input required by seven segment display is calculated and is fed into the display. As a result, only selected threshold values can be displayed using seven segment display. Instead, a lookup table can be used to fetch the 7-bit value required to be fed into seven segment display for corresponding threshold values. A copy of such lookup table is added to the module but requires further testing and bug fixes before release.

It should be noted that the seven-segment display has an inverted input, meaning that the value needs to be inverted before it needs to be fed into the display. If the input to the seven-segment display is 1, then seven segment display interprets the same as LOW signal and vice versa.

## **4.4 Testing of Sobel Edge IP Core**

Initially, individual components in the module are designed and tested for their functionality. The state machine to jump from one mode to another depending on the position of the slide switch. Auto verification test benches are designed to verify if the operation of the mode selection state machine is satisfactory.

Next, the module to display the image on LT24 display is designed and hardware tested. .mif files are notoriously buggy when debugged using RTL simulator, and its advisable to use hardware to test the same. Text images looks inverted when fed into LT24, which is probably due to the reason on how pixel data is stored in an JPEG image. The image to be fed into LT24 is inverted using MATLAB before it is converted into .mif format. This is a work around, and module needs to be developed to invert the input image that is fed into the memory.

During calculation, state machine needs to jump from SOBEL\_STATE1 to SOBEL\_STATE5 if the current pixels are corner pixels of the input image. But, for all other pixels, the state machine needs to jump from SOBEL\_STATE1 to SOBEL\_ADDRESS1 to fetch the pixels for calculation before it can be passed on to STATES that perform the calculation. Test bench has been developed and waveforms are used to check if the state machine works as per expectation. The waveforms from the above mentioned testing is added to Annexure.

Finally, input is manually fed into the system and verified if the resulting approximate gradient from the calculation states is equal to the value produced by Sobel Edge operation on MATLAB. The screenshot from the above-mentioned testing is added to Annexure. As mentioned above, using .mif files during RTL simulation is buggy and hence values are manually fed into the system for calculation.

Finally, individually tested modules are integrated into one final module and auto verification test bench is designed to test if the mode selecting state machine works as per expectation and if correct value is fed as input to seven segment display.

TCL script is written to connect all the input and output PINS in the module. The pins used for debug in the test bench is connected with red LEDs in the board. No input or output PINS should be left unassigned before hardware testing. Double checking is necessary to verify every

input/output PINS are assigned as unassigned input/output PINS might permanently damage the FPGA board.

Test benches, output waveforms, values printed in output console is added to Annexure for better understanding. TCL script used for PIN assignment is also added.

## 5 Performance Comparison and Further Testing

### 5.1 Performance Comparison

Consider the following calculation on SOBEL\_STATE2.

```
sobel_temp1 = ( pixel1 + ( 2 * pixel2 ) + pixel3 );  
sobel_temp2 = ( pixel7 + ( 2 * pixel8 ) + pixel9 );  
  
sobel_temp3 = ( pixel3 + ( 2 * pixel6 ) + pixel9 );  
sobel_temp4 = ( pixel1 + ( 2 * pixel4 ) + pixel7 );
```

The above operation on FPGA can be completed in a single clock cycle. But the calculation of sobel\_temp1 would take at least 7 clock cycles, and would require at least 28 clock cycles to complete in ARM.

```
ldr    r2, [fp, #-56]  
ldr    r3, [fp, #-48]  
lsl    r3, r3, #1  
add    r2, r2, r3  
ldr    r3, [fp, #-44]  
add    r3, r2, r3  
str    r3, [fp, #-8]
```

Figure 6. ARM Assembly Code to calculate sobel\_temp1.

This demonstrates the advantage of using FPGA as hardware accelerator to increase the performance of General-Purpose Processors.

### 5.2 Further Testing and Advancements

It should be noted that we are using asynchronous logic in our designs. This is not an issue when we are running the module on a low clock cycle such as 50MHz, but at higher clock frequencies, propagation delays would potentially break the system. Testing needs to be done to validate the highest clock frequency at which the module can be run.

Slide switches are used to change threshold, but instead of slide switches, push buttons can be used to control the threshold values. However, multiple button pushes are registered when using push button as control, and this needs to be fixed.

When using push buttons to control threshold values, look up table can be used to display the current selected threshold on seven segment displays. Such lookup table has been added to seven segment display module but is not implemented as it requires further testing before it can be used.

## 6 Conclusion

In this project, we have discussed the design of Edge Detection using Sobel Operator in Cyclone V FPGA. We discussed the design of fully parameterised IP core module along with LT24 display operation. We discussed about the testing methodology adopted to debug any potential bugs lingering around. Also, we discussed the performance improvements on using a FPGA as a hardware accelerator for compute intensive tasks.

The code for IP core and test benches are added in Annexure. Screenshots of waveforms and console are also added to Annexure. Further advancements and bug fixes as discussed above would be developed and pushed to Git Hub repo in later versions.

## References

- [1] Z. Guo, W. Xu and Z. Chai, "Image Edge Detection Based on FPGA," 2010 Ninth International Symposium on Distributed Computing and Applications to Business, Engineering and Science, 2010, pp. 169-171, doi: 10.1109/DCABES.2010.39.
- [2] Hardware Acceleration on SoC FPGAs – Altera - [ab07\\_soc\\_fpga.pdf \(intel.com\)](#)
- [3]. Image Processing : Edge Detection.  
<https://www.owlnet.rice.edu/~elec539/Projects97/morphjrks/moredge.html>
- [4]. Memory Blocks - [Microsoft PowerPoint - LECT06 \(ua.edu\)](#)
- [5]. LT24 Interfacing IP Core provided by University of Leeds
- [6]. UpCounter Verilog file IP Core inspired by Verilog code by Thomas Carpenter – University of Leeds
- [7]. .MIF file creation MATLAB code provided by University of Leeds.
- [8]. Edge Detector – University of Edinburgh - <https://homepages.inf.ed.ac.uk/rbf/HIPR2/sobel.htm>

## Test Bench Waveforms

Autoverification test bench to test basic verilog arithmetic for subtraction and addition if value exceeds min and max value

```

# =====
# Attempting to use alternate WLF file: ".\wlf\wlf.txt"
# ** Warning: (warn-wlf-5001) Could not open WLF file: warn.wlf
# Using alternate file: ".\wlf\wlf.txt"
# =====

Then we can configure the units for the timeline. Lets stick to
nanoseconds instead of the default of picoseconds.

#
# configure wave -timelimitsunits ns
#

Next we can start the simulation. We simply say "run".

The "run" command can be followed by a time parameter, such as:

run 100ns      # Run for 100ns
run -all      # Run until the testbench stops changing stimuli
run 10        # Run for 10 "timeunits" (typically lps per step)

===== Simulation Starts Here =====

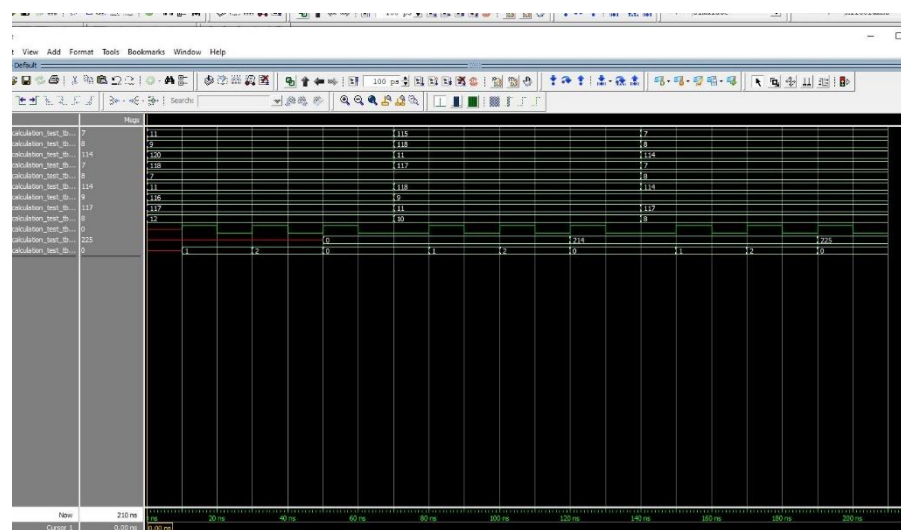
run -all
Simulation Started
Aigo Success

===== Simulation Ends Here =====

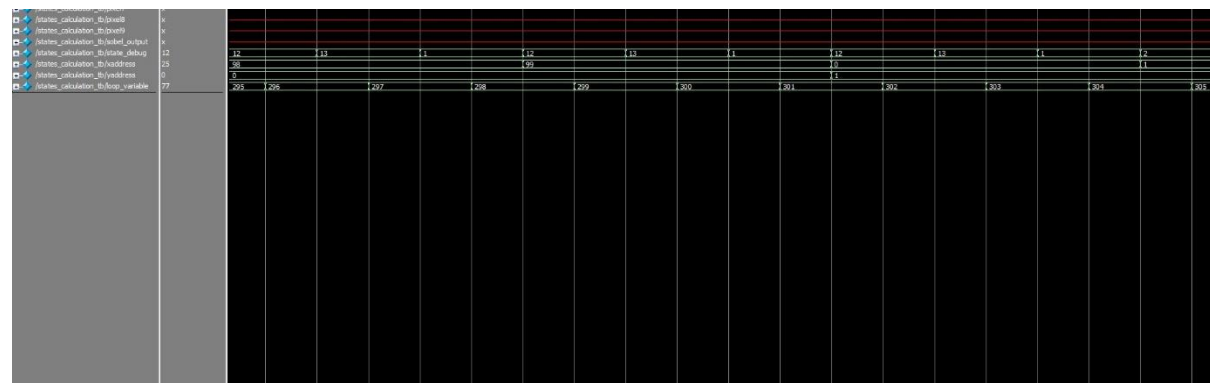
Once the simulation finishes we will reach here

```

Sobel Edge Operation performed on input pixels



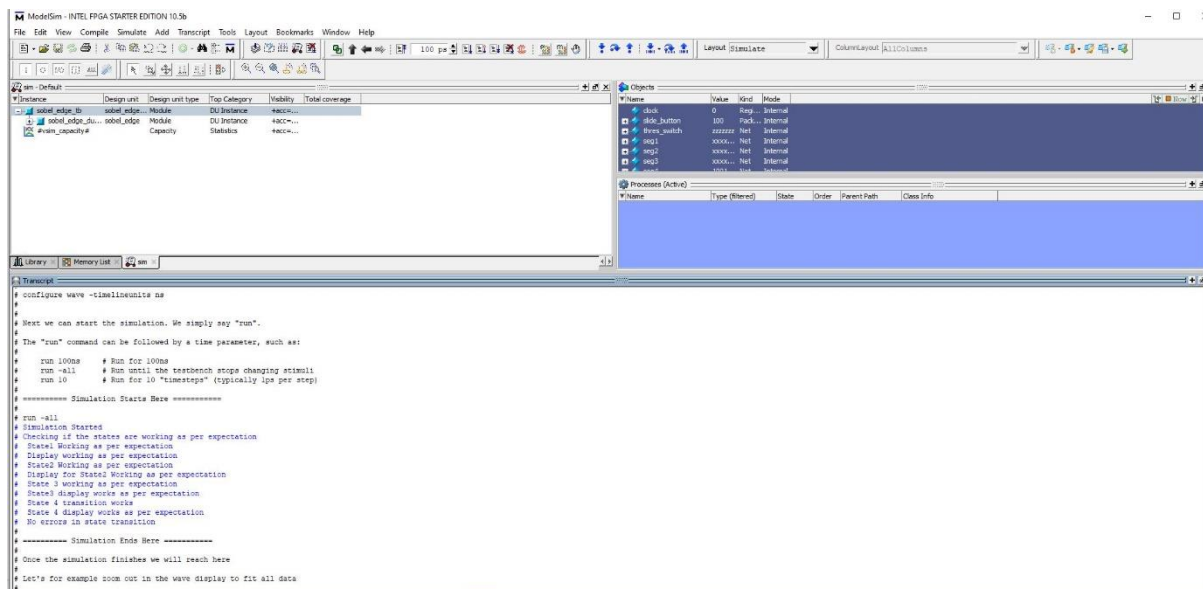
### Change in State when Corner Pixel is in calculation, and Other Pixels are in Calculation



Waveform showing change in X addr and Y Addr



Autoverification test bench to check change of state and output to seven segment display in main module.



**Sobel\_edge.v**

/\* sobel\_edge.v

Module Name : Sobel Edge Verilog Module

Module Description : Module used to calculate sobel edge and sobel threshold for a given input image.

Module Author : Arul Prakash Samathuvamani, LT24 Module provided by University of Leeds - Author - Thomas Carpenter

Date: 13/5/2021

-----

Changelog :

Threshold Lookup .mif added here

State wise counter removed and added into edge and threshold module seperately

Changed from pushbutton to slide switch - 18/5

\*/

// -----

// Declaration of module sobel\_edge

module sobel\_edge #(

    /\* Parameters for the module

    IMAGE\_WIDTH - Width of input image - Defaults 100

    IMAGE\_HEIGHT - Height of input image - Defaults to 100

    \*/

    parameter IMAGE\_WIDTH = 100,

    parameter IMAGE\_HEIGHT = 100

)(

/\* Input Pins Description:

Pin Name	Description
clock	Clock signal
globalReset	globalReset signal for LT24 Display -> Refer LT24Display module for more info
slide_button [0]	Input signal for Slide Switch 0
slide_button [1]	Input signal for Slide Switch 1
slide_button [2]	Input signal for Slide Switch 2
key_button1	Input signal for Push Button 1
key_button2	Input signal for Push Button 2
*/	

input clock,

input globalReset,

input [2:0] slide\_button, // input from slide button

input [6:0] thres\_switch,

// Output Pins Declaration

// Output Pins to seven\_segment.v

output reg [1:0] disp\_out, // Informs seven\_segment.v of which state the system is currently in

output reg [20:0] threshold, // If in edge threshold mode, passes to seven segment the input for current threshold values.

```
// LT24 Display Output Pins -> Based on verilog file written by Thomas Carpenter
```

```
output resetApp,
```

```
output LT24Wr_n,
```

```
output LT24RD_n,
```

```
output LT24CS_n,
```

```
output LT24RS,
```

```
output LT24Reset_n,
```

```
output [15:0] LT24Data,
```

```
output LT24LCDOn,
```

```
// Output Pins to 7-segment Display
```

```
output [6:0] seg1, // output to 7-segment1
```

```
output [6:0] seg2, // output to 7-segment2
```

```
output [6:0] seg3, // output to 7-segment3
```

```
output [6:0] seg4, // output to 7-segment4
```

```
output [6:0] seg5, // output to 7-segment5
```



```
output [6:0] seg6, // output to 7-segment6
```

```
// Debug used in test bench
```

```
output reg debug1,
```

```
output reg debug2,
```

```
output reg debug3,
```

```
output reg debug4
```

```
);
```

```
// -----
```

```
// Declaration of local variables used in the module
```

```
reg [7:0] xAddr; // x co-ordinate for LT24 Display
```

```
reg [8:0] yAddr; // y co-ordinate for LT24 Display
```

```
// xAddr and yAddr is also used to calculate the memory address for images.
```

```
// -----
```

```
// LT24 Display parameters -> to satisfy needs of LT24Display Verilog file by Thomas Carpenter
```

```
reg [15:0] pixelData;
```

```
wire pixelReady;
```

```
reg pixelWrite;
```

```
//-----
```

```
// Declaration of registers for input image, loaded using .mif Quartus file
```

```
(*ram_init_file = "data.mif"*) // Input 100x100 image -> default
```

```
reg [ 7 : 0 ] input_image [ ( ( IMAGE_WIDTH ) * ( IMAGE_HEIGHT ) - 1 ) : 0 ];
```

```
(*ram_init_file = "mod2.mif"*) // Input 100x320 Leeds Logo Image
```

```
reg [ 7:0 ] logo [ 31999 : 0 ];
```

```
// reg [ 7:0 ] edge_image [ ( ( IMAGE_WIDTH ) * ( IMAGE_HEIGHT ) - 1 ) : 0 ]; Used for debugging purposes
```

```
// -----
```

```
// Image buffer -> used to convert 16bit Grayscale to RGB565 Format
```

```
reg [7:0] image_buffer; // Temp variable used for conversion
```

```
//reg [31:0] address; // address register used for debugging purposes -> currently not used
```

```
//reg [7:0] pixel_buffer [299:0]; // alternate pixel fethching method -> Will not work, DO NOT USE
```

```
// -----
```

```
// Parameter values used for LCD Display
```

```
localparam LCD_WIDTH = 240;
```

```
localparam LCD_HEIGHT = 320;
```

```
// -----
```

```
// Parameter values used to denote the size of University of Leeds logo at the start
```

```
localparam LOGO_HEIGHT = 320;
```

```
localparam LOGO_WIDTH = 100;
```

```
// -----
```

```
// X Count andn Y Count values
```

```
wire [7:0] xCount;
```

```
wire [8:0] yCount;
```

```
wire yCntEnable = pixelReady && (xCount == (LCD_WIDTH-1));
```

```
// -----
```

```
// Declaration of states corresponing to the status of slide switch
```

```
localparam STATE1 = 3'b000; // STATE1 -> Display Leeds Logo
```

```
localparam STATE2 = 3'b001; // STATE2 -> Display Grayscale Image
```

```
localparam STATE3 = 3'b010; // STATE3 -> Display Sobel Edge Image
```

```
localparam STATE4 = 3'b100; // STATE4 -> Display Sobel Edge Threshold of Image
```

```
// Declaration of part to run on positive edge of clock
```

```
localparam LOGO = 2'b00; // Display Leeds Logo
```

```
localparam GRAY = 2'b01; // Display Gray Scale image
```

```
localparam EDGE = 2'b10; // Display Sobel edge image
```

```
localparam THRES = 2'b11; // Display Sobel Edge Threshold of Image
```

```
reg [2:0] STATE = 2'b00; // Declare state of the system, used to select which part to run
```

```
// Declare calculation states -> used to run a specific part during every clock
```

```
reg [3:0] SOBEL_STATE = 4'b0000;
```

```
reg [3:0] THRES_STATE = 4'b0000;
```

```
// Declaration of states in Sobel Edge Calculation
```

```
localparam SOBEL_INITIAL = 4'b0000; // Used to set xAddr and yAddr values to zero
```

```
localparam SOBEL_STATE1 = 4'b0001; // Used to check on what action to be done -> depends on coordinates
```

```
// Parameters used to fetch 8- pixel information
```

```
/* P1 P2 P3
```

```
P4 P5 P6
```

P7 P8 P8

P5 is the current pixel in calculation -> corresponding states are used to fetch corresponding pixel data

\*/

localparam SOBEL\_ADDRESS1 = 4'b0010; // Fetch Pixel1

localparam SOBEL\_ADDRESS2 = 4'b0011; // Fetch Pixel2

localparam SOBEL\_ADDRESS3 = 4'b0100; // Fetch Pixel3

localparam SOBEL\_ADDRESS4 = 4'b0101; // Fetch Pixel4

localparam SOBEL\_ADDRESS5 = 4'b0110; // Fetch pixel6

localparam SOBEL\_ADDRESS6 = 4'b0111; // Fetch Pixel7

localparam SOBEL\_ADDRESS7 = 4'b1000; // Fetch Pixel8

localparam SOBEL\_ADDRESS8 = 4'b1001; // Fetch Pixel9

// States used for Values calculation

localparam SOBEL\_STATE2 = 4'b1010;

localparam SOBEL\_STATE3 = 4'b1011;

localparam SOBEL\_STATE4 = 4'b1100;

localparam SOBEL\_STATE5 = 4'b1101;

reg threshold\_enable;

//-----

// Temporary registers used for calculations

reg [8:0] sobel\_temp1;

```

reg [8:0] sobel_temp2;

reg [8:0] sobel_temp3;

reg [8:0] sobel_temp4;


// Corresponding pixel data during each cycle of fetch is stored here

reg [7:0] pixel1;

reg [7:0] pixel2;

reg [7:0] pixel3;

reg [7:0] pixel4;

//reg [7:0] pixel5;

reg [7:0] pixel6;

reg [7:0] pixel7;

reg [7:0] pixel8;

reg [7:0] pixel9;


reg [7:0] current_threshold = 8'd50;


// Seven Segment threshold display lookup registers -> not used with slide switch method of selection- just used for presentation


(*ram_init_file = "seg_lookup.mif"*)

reg [20:0] lookup [251:0];


//-----

// Declaration of upcounter module

upcounter #(

    .WIDTH    (        9),

    .MAX_VALUE(LCD_HEIGHT-1)

) yCounter (

```

```

        .clock (clock ),

        .reset (resetApp ),

        .enable (yCntEnable),

        .countValue(yCount )

);

```

```

upcounter #(

    .WIDTH ( 8),

    .MAX_VALUE(LCD_WIDTH-1)

) xCounter (

    .clock (clock ),

    .reset (resetApp ),

    .enable (pixelReady),

    .countValue(xCount )

);

```

```

// Declaration of seven segment display module

```

```

seven_segment (

    .clock ( clock),

    .thres ( threshold ),

    .state ( disp_out ),

    .seg1 ( seg1 ),

    .seg2 ( seg2 ),

    .seg3 ( seg3 ),

    .seg4 ( seg4 ),

    .seg5 ( seg5 ),

```

```

        .seg6 ( seg6 )

        //debug1 ( debug1 ),

        //debug2 ( debug2 ),

        //debug3 ( debug3 ),

        //debug4 ( debug4 )

);

// Declaration of LT24Display module

LT24Display #(

    .WIDTH      (LCD_WIDTH ),

    .HEIGHT     (LCD_HEIGHT),

    .CLOCK_FREQ (50000000 )

) Display (

    //Clock and Reset In

    .clock      (clock      ),

    .globalReset (globalReset),

    //Reset for User Logic

    .resetApp   (resetApp   ),

    //Pixel Interface

    .xAddr      (xAddr      ),

    .yAddr      (yAddr      ),

    .pixelData   (pixelData  ),

    .pixelWrite  (pixelWrite ),

    .pixelReady  (pixelReady ),

    //Use pixel addressing mode

    .pixelRawMode(1'b0      ),

    //Unused Command Interface

    .cmdData     (8'b0      ),

```



```

.cmdWrite (1'b0 ),

.cmdDone (1'b0 ),

.cmdReady ( ),

//Display Connections

.LT24Wr_n (LT24Wr_n ),

.LT24Rd_n (LT24RD_n ),

.LT24CS_n (LT24CS_n ),

.LT24RS (LT24RS ),

.LT24Reset_n (LT24Reset_n),

.LT24Data (LT24Data ),

.LT24LCDO_n (LT24LCDO_n )

);

//-----

// Procedural block used to change the state of the system

always @ ( * ) begin

    case ( slide_button )

        STATE1 : begin

            STATE      <= 2'b00; // set to logo display state

            disp_out    <= 2'b00; // set display to logo state

            threshold_enable <= 1'b0; // Disable threshold selecting module

            // Debug Wires Declaration

            debug1 <= 1'b1;

```

```
debug2 <= 1'b0;
```

```
debug3 <= 1'b0;
```

```
debug4 <= 1'b0;
```

```
end
```

```
STATE2 : begin
```

```
STATE      <= 2'b01; // set to gray scale display state
```

```
disp_out    <= 2'b01; // set display gray
```

```
threshold_enable <= 1'b0; // Disable Threshold selecting module
```

```
// Debug Declartion
```

```
debug1 <= 1'b0;
```

```
debug2 <= 1'b1;
```

```
debug3 <= 1'b0;
```

```
debug4 <= 1'b0;
```

```
end
```

```
STATE3 : begin
```

```
STATE      <= 2'b10; // set to sobel edge display state
```

```
disp_out    <= 2'b10; // set display to edge
```

```
threshold_enable <= 1'b0; // Disable threshold selecting module
```

```
// Debug Declaration
```

```
debug1 <= 1'b0;
```

```
debug2 <= 1'b0;
```

```
debug3 <= 1'b1;
```

```
debug4 <= 1'b0;
```

```
end
```

```
STATE4 : begin
```

```
    STATE      <= 2'b10; // set to sobel edge threshold display state
```

```
    disp_out    <= 2'b11; // set display to thres
```

```
    threshold_enable <= 1'b1; // enable threshold selecting module
```

```
    // Debug Declaration
```

```
    debug1 <= 1'b0;
```

```
    debug2 <= 1'b0;
```

```
    debug3 <= 1'b0;
```

```
    debug4 <= 1'b1;
```

```
end
```

```
default : begin
```

```
    STATE      <= 2'b00; // by default, is set to display logo
```

```
    disp_out    <= 2'b00; // set to logo state
```

```
    threshold_enable <= 1'b0; // disable threshold selecting module
```

```
    // Debug Declaration
```

```
    debug1 <= 1'b1;
```

```
    debug2 <= 1'b0;
```

```
    debug3 <= 1'b0;
```

```
    debug4 <= 1'b0;
```

```
end
```

```
endcase
```

end

//-----

// Procedural block to update threshold values

/\* Has bugs and needs to be fixed -> would be added in future release

always @ ( \* ) begin

if ( threshold\_enable ) begin // Runs only if threshold select module is enabled

if ( ~button2 ) begin // If Key1 is pressed

current\_threshold = current\_threshold + 10; // Increment threshold value by 10

display threshold = lookup [ current\_threshold ]; // Use the lookup table to get input to be fed to seven segment

end

else if ( ~button3 ) begin // If key2 is pressed

current\_threshold = current\_threshold -10 ; // Increment threshold value by 10

display threshold = lookup [ current\_threshold ]; // Use the lookup table to get input to be fed into seven segment

end

end

```
end
```

```
*/
```

```
// -----
```

```
// LT24 Pixel Write
```

```
always @ ( posedge clock or posedge resetApp ) begin
```

```
    // When pixelwrite is set to 1, it means that LT24 display is ready to get the new pixel data
```

```
    if ( resetApp ) begin
```

```
        pixelWrite <= 1'b0;
```

```
    end else begin
```

```
        pixelWrite <= 1'b1;
```

```
    end
```

```
end
```

```
// -----
```

```
// Threshold Selecting Module
```

```
/* Default -> 50
```

```
Switch 3 -> Threshold of 100
```

```
Switch 4 -> Threshold of 120
```

```
Switch 5 -> Threshold of 140
```

```
Switch 6 -> Threshold of 160
```

```
Switch 7 -> Threshold of 180
```

```
Switch 8 -> Threshold of 200
```

```
Switch 9 -> Threshold of 220
```

```
*/
```

```
always @ (*) begin
```

```
case ( thres_switch )
```

```
7'b0000001 : begin // Switch 3 is on
```

```
current_threshold <= 8'd100; // Set threshold to 100
```

```
threshold      <= 21'b00001100111110111111; // set the display to 100 -> Use lookup.mif in future versions
```

```
end
```

```
7'b0000010 : begin // when switch 4 is on
```

```
    current_threshold <= 8'd120; // set threshold to 120
```

```
    threshold      <= 21'b00001101011011011111; // set the display to 120
```

```
end
```

```
7'b0000100 : begin // when switch 5 is on
```

```
    current_threshold <= 8'd140; // set threshold to 140
```

```
    threshold      <= 21'b00001101100110011111; // set the display to 140
```

```
end
```

```
7'b0001000 : begin // when switch 6 is on
```

```
    current_threshold <= 8'd160; // set threshold to 160
```

```
    threshold      <= 21'b00001101111101011111; // set the display to 160
```

```
end
```

```
7'b0010000 : begin // when switch 7 is on
```

```
    current_threshold <= 8'd180; // set threshold to 180
```

```
    threshold      <= 21'b00001101111111011111; // set the display to 180
```

```
end
```

```
7'b0100000 : begin // when switch 8 is on
```

```

        current_threshold <= 8'd200; // set the threshold to 200

        threshold      <= 21'b10110110111110111111; // set the display to 200

    end

7'b1000000 : begin // when switch 9 is on

    current_threshold <= 8'd220; // set the threshold to 220

    threshold      <= 21'b101101110110110111111; // set the display to 220

    end

default : begin

    current_threshold <= 8'd50; // set the threshold to 50

    threshold      <= 21'b000000011011010111111; // set the display to 50

    end

endcase

end

// -----

// LT24 Display Module - Displays the corresponding pixel data according to the state

/* Information regarding the states

    LOGO  - Display Leeds Logo - Initial Boot Stage

    GRAY  - Display Grayscale image on LT24

    SOBEL - Display Sobel Edge image

    THRES - Display Sobel Edge Threshold image

```





```

corresponding pixel data and convert to 5 bits
image_buffer = logo[ ( yAddr * 100 ) + xAddr ] >> 2 ; // get the

pixelData [ 15:11 ] = image_buffer [5:1]; // assing bits according
to RGB to get grayscale image., bits 1-5 contain actual pixel data, LSB contains zero

pixelData [ 4:0 ] = image_buffer [5:1];

pixelData [ 10:5 ] = image_buffer [5:0];

end

end

else begin

pixelData = 16'b1; // set the screen to black on areas other than image

end

end

GRAY : begin // This sets the display to input gray scale image

xAddr = xCount; // set counter values to value of address

yAddr = yCount;

if ( xAddr < IMAGE_WIDTH ) begin // if coordinates are less than width and height of image, continue

if ( yAddr < IMAGE_HEIGHT ) begin

image_buffer = input_image[ ( yAddr * IMAGE_HEIGHT ) + xAddr ] >> 2 ;
// get the corresponding pixel data and convert to 5 bits

pixelData [ 15 : 11 ] = image_buffer [5:1]; // assing bits according to RGB to get
grayscale image., bits 1-5 contain actual pixel data, LSB contains zero

pixelData [ 10 : 5 ] = image_buffer[5:0];

```

```
pixelData [4:0]    = image_buffer [5:1];
```

```
end
```

```
end
```

```
else begin
```

```
pixelData = 16'b1; // set the screen to black on areas other than image
```

```
end
```

```
end
```

```
EDGE : begin // sets the display input to display edge calculated image
```

```
case ( SOBEL_STATE )
```

```
SOBEL_INITIAL : begin // initial state, set all values to zero
```

```
    xAddr = 0;
```

```
    yAddr = 0;
```

```
    SOBEL_STATE = 4'b0001;
```

```
end
```

```
SOBEL_STATE1 : begin // sobel state1 -> find which data to fetch
```

```
pixel, set image_buffer as white and go to last state    if ( xAddr == 0 || xAddr == ( IMAGE_WIDTH - 1 ) ) begin // if corner
```

```
    image_buffer = 8'b1;
```

```
    SOBEL_STATE = 4'b1101;
```

```

end

corner pixel, set image_buffer as white and go to last state

else if ( yAddr == 0 || yAddr == ( IMAGE_HEIGHT - 1 ) ) begin // if

    image_buffer = 8'b1;

    SOBEL_STATE = 4'b1101;

end

// if coordinates out of the image, blacken the area

else if ( xAddr > IMAGE_WIDTH || yAddr > IMAGE_HEIGHT ) begin

    image_buffer = 8'b0;

    SOBEL_STATE = 4'b1101;

end

else begin

    SOBEL_STATE = 4'b0010; // for all other pixels, calculate

the edge using kernel

end

end

/*

Pixel1 Pixel2 Pixel3

Pixel4 Curr_P Pixel5

Pixel6 Pixel7 Pixel8

```

```

Curr_p -> current pixel

*/

SOBEL_ADDRESS1 : begin

    pixel1    = input_image [ ( ( yAddr - 1 ) * IMAGE_HEIGHT ) + (
xAddr - 1 ) ]; // get pixel 1 -> one col, one row before

    SOBEL_STATE = 4'b0011;

end

SOBEL_ADDRESS2 : begin

    pixel2    = input_image [ ( ( yAddr - 1 ) * IMAGE_HEIGHT ) + ( xAddr
) ]; // get pixel2

    SOBEL_STATE = 4'b0100;

end

SOBEL_ADDRESS3 : begin

    pixel3    = input_image [ ( ( yAddr - 1 ) * IMAGE_HEIGHT ) + (
xAddr + 1 ) ]; // get pixel3

    SOBEL_STATE = 4'b0101;

end

SOBEL_ADDRESS4 : begin

    pixel4    = input_image [ ( yAddr * IMAGE_HEIGHT ) + ( xAddr - 1 )

]; // get pixel4

    SOBEL_STATE = 4'b0110;

```

end

SOBEL\_ADDRESS5 : begin

pixel6 = input\_image [ ( yAddr \* IMAGE\_HEIGHT ) + ( xAddr + 1 )  
]; // get pixel6

SOBEL\_STATE = 4'b0111;

end

SOBEL\_ADDRESS6 : begin

pixel7 = input\_image [ ( ( yAddr + 1 ) \* IMAGE\_HEIGHT ) + (  
xAddr - 1 ) ]; // get pixel7

SOBEL\_STATE = 4'b1000;

end

SOBEL\_ADDRESS7 : begin

pixel8 = input\_image [ ( ( yAddr + 1 ) \* IMAGE\_HEIGHT ) + (  
xAddr ) ]; // get pixel8

SOBEL\_STATE = 4'b1001;

end

SOBEL\_ADDRESS8 : begin

pixel9 = input\_image [ ( ( yAddr + 1 ) \* IMAGE\_HEIGHT ) + (  
xAddr + 1 ) ]; // get pixel9

SOBEL\_STATE = 4'b1010;

end

```
SOBEL_STATE2 : begin
```

```
// Calculating pixel by multiplying with Kernel
```

```
// P1 + 2P2 + P3 , P7 + 2P8 + P9 , P3 + 2P6 + P9, P1+ 2P4+P7
```

```
/* Convolution kernel is
```

```
mx = -1 0 1 my = -1 -2 -1
```

```
-2 0 2    0 0 0
```

```
-1 0 1    1 2 1
```

```
*/
```

```
sobel_temp1 = ( pixel1 + ( 2 * pixel2 ) + pixel3 );
```

```
sobel_temp2 = ( pixel7 + ( 2 * pixel8 ) + pixel9 );
```

```
sobel_temp3 = ( pixel3 + ( 2 * pixel6 ) + pixel9 );
```

```
sobel_temp4 = ( pixel1 + ( 2 * pixel4 ) + pixel7 );
```

```
set to 255
```

```
if ( sobel_temp1 [8] ) begin // if the calculated value is greater than 255,
```

```
sobel_temp1 = 9'd255;
```

```
end
```

```
set to 255
```

```
if ( sobel_temp2 [8] ) begin // if the calculated value is greater than 255,
```

```
sobel_temp2 = 9'd255;
```

```

end

if ( sobel_temp3 [8] ) begin // if the calculated value is greater than 255,
set to 255

    sobel_temp3 = 9'd255;
end

if ( sobel_temp4 [8] ) begin // if the calculated value is greater than 255,
set to 255

    sobel_temp4 = 9'd255;

end

SOBEL_STATE = 4'b1011; // proceed to next state

end

SOBEL_STATE3 : begin

    // Calculate Fx and Fy

    sobel_temp1 = sobel_temp1 - sobel_temp2;

    sobel_temp2 = sobel_temp3 - sobel_temp4;

    if(sobel_temp1 [8] ) begin // if negative value, set to zero

        sobel_temp1 = 9'd0;

    end

```



```
if ( sobel_temp2[8] ) begin // if negative value, set to zero
```

```
    sobel_temp2 = 9'd0;
```

```
end
```

```
SOBEL_STATE = 4'b1100; // proceed to next state
```

```
end
```

```
SOBEL_STATE4 : begin
```

```
    // Edge = |fx| + |fy|
```

```
    sobel_temp1 = sobel_temp1 + sobel_temp2;
```

```
    if ( sobel_temp1[8] ) begin // if greater than 255 set to 255
```

```
        sobel_temp1 = 9'd255;
```

```
    end
```

```
    if ( threshold_enable ) begin
```

```
        if ( sobel_temp1 < current_threshold ) begin
```

```
            sobel_temp1 = 9'b0;
```

```
        end
```

```
    end
```

```
    image_buffer = sobel_temp1; // if greater than 255 set to 255
```

```
SOBEL_STATE = 4'b1101; // proceed to next state
```

```
end
```

```
SOBEL_STATE5 : begin
```

```
image_buffer = image_buffer >> 2; // get the corresponding pixel  
data and convert to 5 bits
```

```
pixelData [ 15 : 11 ] = image_buffer [5:1]; // assing bits according to  
RGB to get grayscale image., bits 1-5 contain actual pixel data, LSB contains zero
```

```
pixelData [ 10 : 5 ] = image_buffer[5:0];
```

```
pixelData [4:0] = image_buffer [5:1];
```

```
if ( xAddr < ( LCD_WIDTH - 1 ) ) begin // increment xAddr
```

```
    xAddr = xAddr +1;
```

```
end
```

```
else begin
```

```
    xAddr = 0;
```

```
    if ( yAddr < ( LCD_HEIGHT -1 ) ) begin // increment yAddr
```

```
        yAddr = yAddr + 1;
```

```
    end
```

```
    else begin
```

```
        yAddr = 0;
```

```
    end
```

end

SOBEL\_STATE = 4'b0001; // go to state1

end

endcase

end

//THRES : begin

// THRES part added into sobel edge state

//end

endcase

end

end

endmodule

## Sobel\_edge\_tb

// Verilog Test Bench Module

// Test Bench Description:

/\*

Auto verification test bench that runs all the tests necessary to determine if

the response from the module is satisfactory and as per requirement.

After running the test bench, if Success at end -> Test bench ran successfully.

If not, for check individual module errors.

Author: Arul Prakash Samathuvamani

```
*/
```

```
//-----
```

```
`timescale 1 ns/100 ps
```

```
// Module Declaration
```

```
module sobel_edge_tb;
```

```
// Test Bench input pins declaration
```

```
reg clock; // clock for the module
```

```
reg [2:0] slide_button; // input to simulate slide button
```

```
wire [6:0] thres_switch; // input to simulate slide button
```

```
// Output to seven segment displays
```

```
wire [6:0] seg1;
```

```
wire [6:0] seg2;
```

```
wire [6:0] seg3;
```

```
wire [6:0] seg4;
```

```
wire [6:0] seg5;
```

```
wire [6:0] seg6;
```

```
wire [1:0] disp_out;
```

```
// State Debug Output
```

```
wire debug1;
```

```
wire debug2;
```

```
wire debug3;
```

```
wire debug4;
```

```
wire pixelReady;
```

```
// DUT Declaration
```

```
sobel_edge sobel_edge_dut (
```

```
    .clock(clock),
```

```
    .slide_button ( slide_button),
```

```
    .thres_switch ( thres_switch),
```

```
    .seg1 ( seg1),
```

```
    .seg2( seg2),
```

```
    .seg3( seg3),
```

```
    .seg4(seg4),
```

```
    .seg5(seg5),
```

```
    .seg6( seg6),
```

```
    .disp_out (disp_out),
```

```
    .debug1( debug1),
```

```
    .debug2 ( debug2),
```

```
    .debug3 ( debug3),
```

```
    .debug4( debug4)
```

```
);
```

```
// Start Simulation
```

```
integer error = 0;
```

```
initial begin
```

```
    $display ( "Simulation Started" );
```

```
    // Set and reset clock
```

```
        clock = 1'b0;
```

```
        #10; // Delay 10 seconds
```

```
        clock = 1'b1; // set clock as high
```

```
        #10;
```

```
        clock = 1'b0;
```

```
        $display ("Checking if the states are working as per expectation");
```

```
        slide_button = 3'b000; // set slide button off
```

```
        #10;
```

```
        clock = 1'b1;
```

```
        if ( debug1 ) begin // check is state transistion works
```

```
            $display (" State1 Working as per expectation" );
```

```
        end
```

```
        else begin
```

```
            error = 1;
```

```

        $display (" Error in State1 ");

end

if ( seg6 == ~(7'b1110100) ) begin // check if display works as per expectation

        $display ( " Display working as per expectation");
end
else begin

        $display (" Error in State2 Display");
        error =1 ;
end

#10;

        // Set and reset clock

clock = 1'b0;

slide_button = 3'b001; // Turn on first slide button

#10;          // wait

//set clock

clock = 1'b1;

if ( debug2 ) begin // check if state transition is successful

        $display(" State2 Working as per expectation");

end

else begin //else

        $display( " Error in State2 ");
        error = 1;

```

```

end

if ( seg6 == ~(7'b1101111) ) begin // check if display works

    $display(" Display for State2 Working as per expectation");

end

else begin

    $display (" Display in state 2 has an error ");
    error = 1;

end

#10;

clock = 1'b0;

// Check for state 3 transition

slide_button = 3'b010; // turn on second slide button

#10;

clock = 1'b1;

if ( debug3 ) begin // check if state transition is successful

    $display (" State 3 working as per expectation" );

end

else begin

    $display ( " Error in state 3 transition");
    error = 1;

end

```



```
if ( seg6 == ~(7'b1111011) ) begin // check if display is set correctly
```

```
    $display ( " State3 display works as per expectation");
```

```
end
```

```
else begin
```

```
    $display (" State3 transition display error");
```

```
    error = 1;
```

```
end
```

```
#10;
```

```
clock = 1'b0;
```

```
// check for state 4 transition
```

```
slide_button = 3'b100; // turn on 3rd slide button
```

```
#10;
```

```
clock = 1'b1;
```

```
if (debug4 ) begin // check if state transition is successful
```

```
    $display ( " State 4 transition works");
```

```
end
```

```
else begin
```

```
    $display (" State 4 transition error");
```

```
    error = 1;
```

```
end
```

```
if ( seg6 == ~(7'b1111000) ) begin // check if display is set correctly
```

```

        $display ( " State 4 display works as per expectation ");

end

else begin

    $display (" State4 display transition works ");
    error = 1;

end

#10;

clock = 1'b0;

slide_button = 3'b110; // check if defaulting

clock = 1'b1;

#10;

if (debug1 ) begin // defaulting?

    $display ( " Abnormal state defaulting" );

end

else begin

    $display ( " Error !! " );
    error = 1;

end

#10;

clock = 1'b0;

if ( !error ) begin

```

```

        $display ( " No errors in state transition ");

    end

    else begin

        $display ( " Errors present in state transition");

    end

end

endmodule

```

Up\_counter.v

/\* upcounter.v

-----

Module Name : Upcounter Verilog Module

Module Description : Increase the value of the input by 1 at everyu positive clock untill the clock value reaches max.

If clock value reaches max, it is reset to zero.

Module Author : Arul Prakash Samathuvamani , Inspired from upcounter verilog file by Thomas Carpenter - University of Leeds

Date : 13/5/2021

Changelog:

Module is parameterized

\*/

```
// -----
```

```
// Module Declaration
```

```
module upcounter #(
```

```
    parameter WIDTH = 10, // Size of count value
```

```
    parameter INCREMENT = 1, // value to increment counter by
```

```
    parameter MAX_VALUE = ( 2**WIDTH ) -1 // maximum value to be calculated
```

```
)(
```

```
    /* Input Pins Declaration
```

```
        clock -> input clock
```

```
        reset -> reset signal
```

```
        enable -> if to enable upcounter
```

```
    */
```

```
        input clock,
```

```
        input reset,
```

```
        input enable,
```

```
        output reg [ (WIDTH-1) : 0 ] countValue
```

```
);
```

```
// Up- Counter Module
```

```
always @ ( posedge clock ) begin
```

```
    if ( reset ) begin
```

```
        countValue <= { (WIDTH){1'b0} }; // set all the bits to zero
```

```
    end else if ( enable ) begin
```

```
        if ( countValue >= MAX_VALUE [WIDTH-1:0]) begin
```

```
            countValue <= { (WIDTH){1'B0} };
```

```
        end else begin
```

```
            countValue <= countValue + INCREMENT [WIDTH-1:0];
```

```
        end
```

```
    end
```

```
end
```

```
endmodule
```

```
seven_segment.v
```

```
/* seven_segment.v
```

---

Module Name : Seven Segment Display Verilog Module

Module Description : Module to control seven segment display

Module Author : Arul Prakash Samathuvamani

Date: 16/5/2021

---

Changelog :

Threshold value lookup table removed and added to main module

```
*/
```

```
module seven_segment (
```

```
    // Declaration of input pins
```

```
    input clock,
```

```
    input [1:0] state, // says what to display
```

```
    input [20:0] thres, // says what number to display
```

```
    output [6:0] seg1, // output to 7-segment1
```

```
    output [6:0] seg2, // output to 7-segment2
```

```
    output [6:0] seg3, // output to 7-segment3
```

```
    output [6:0] seg4, // output to 7-segment4
```

```
    output [6:0] seg5, // output to 7-segment5
```

```
    output [6:0] seg6, // output to 7-segment6
```

```
    output debug1,
```

```
    output debug2,
```

```
    output debug3,
```

```
    output debug4
```

```
);
```

```
// -----
```

```
// Definition of states to the module
```

```
localparam BASE = 2'b00;  
localparam GRAY = 2'b01;  
localparam SOBEL = 2'b10;  
localparam THRES = 2'b11;
```

```
reg [6:0] data6;  
reg [6:0] data5;  
reg [6:0] data4;  
reg [6:0] data3;  
reg [6:0] data2;  
reg [6:0] data1;
```

```
reg debug_data1;  
reg debug_data2;  
reg debug_data3;  
reg debug_data4;
```

```
assign seg6 = data6;  
assign seg5 = data5;  
assign seg4 = data4;  
assign seg3 = data3;  
assign seg2 = data2;  
assign seg1 = data1;  
assign debug1 = debug_data1;  
assign debug2 = debug_data2;  
assign debug3 = debug_data3;  
assign debug4 = debug_data4;
```

```
reg [20:0] buffer;
```

```
always @ ( * ) begin
```

```
case ( state )
```

```
    BASE : begin
```

```
        data6 = ~(7'b1110100); // displays "h"
        data5 = ~(7'b1111011); // displays "e"
        data4 = ~(7'b0110000); // displays "l"
        data3 = ~(7'b0110000); // displays "l"
        data2 = ~(7'b0111111); // displays "0"
        data1 = ~(7'b0000000);
        debug_data1 = 1'b1;
        debug_data2 = 1'b0;
        debug_data3 = 1'b0;
        debug_data4 = 1'b0;
```

```
    end
```

```
    GRAY : begin
```

```
        data6 = ~(7'b1101111); // displays "g"
        data5 = ~(7'b0110001); // displays "r"
        data4 = ~(7'b1110111); // displays "a"
        data3 = ~(7'b1101110); // displays "y"
        data2 = ~(7'b0000000);
        data1 = ~(7'b0000000);
        debug_data1 = 1'b0;
        debug_data2 = 1'b1;
        debug_data3 = 1'b0;
        debug_data4 = 1'b0;
```

```
    end
```

```
    SOBEL : begin
```

```
        data6 = ~(7'b1111011); // displays "e"
        data5 = ~(7'b1011110); // displays "d"
        data4 = ~(7'b1101111); // displays "g"
        data3 = ~(7'b1111011); // displays "e"
```



```
data2 = ~(7'b00000000);  
data1 = ~(7'b00000000);  
debug_data1 = 1'b0;  
debug_data2 = 1'b0;  
debug_data3 = 1'b1;  
debug_data4 = 1'b0;
```

end

THRES : begin

```
data6 = ~(7'b1111000); // displays "t"  
data5 = ~(7'b1110100); // displays "h"  
data4 = ~(7'b0110001); // displays "r"  
debug_data1 = 1'b0;  
debug_data2 = 1'b0;  
debug_data3 = 1'b0;  
debug_data4 = 1'b1;
```

```
buffer = ~ ( thres );  
data3 = buffer [20:14] ;  
data2 = buffer [13:7];  
data1 = buffer [6:0];
```

end

default : begin

```
data6 = ~(7'b1101111); // displays "g"  
data5 = ~(7'b0110001); // displays "r"  
data4 = ~(7'b1110111); // displays "a"  
data3 = ~(7'b1101110); // displays "y"  
data2 = ~(7'b00000000);  
data1 = ~(7'b00000000);  
debug_data1 = 1'b0;  
debug_data2 = 1'b1;  
debug_data3 = 1'b0;  
debug_data4 = 1'b0;
```

end

endcase

end

endmodule

## LT24Display.v

/\*

\* LT24 Display Driver

\* -----

\* By: Thomas Carpenter

\* For: University of Leeds

\* Date: 13th March 2017

\*

\* Short Description

\* -----

\* This module is designed to interface with the LT24 Display Module

\* from Terasic. It provides functionality to initialise the display

\* and to allow individually addressed pixels to be written to the

\* internal frame buffer of the LT24.

\*

\* Interfaces

\* -----

\* The following interfaces are provided:

\*

\* clock - Input - 1bit

\*     A free-running clock. This will be used to general logic clock

\* globalReset - Input - 1bit

\*     A global reset signal to reset the entire logic herein.

\* resetApp - Output - 1bit

\*     This is the reset which you should use for your code

\*

\* xAddr - Input - XBITS (parameterised)

\*     X-Coordinate of the pixel to be updated

\* yAddr - Input - YBITS (parameterised)

\*     Y-Coordinate of the pixel to be updated

\* pixelData - Input - 16bit

\*     An RGB565 encoded colour to be written to the addressed pixel

\* pixelWrite - Input - 1bit

- \* Setting this input high will trigger a write to the addressed pixel.
- \* This should be kept high until pixelReady goes high.
- \* pixelReady - Output - 1bit
- \* This will be asserted when the LCD has accepted pixel data
- \* pixelRawMode - Input - 1bit
- \* When high this disables setting of X-Y cursor and presents a raw pixel interface.
- \*
- \* cmdData - Input - 8bit
- \* Command Data to be written to the LCD
- \* cmdWrite - Input - 1bit
- \* Setting this high will trigger a command to be written. This has priority over pixelWrite
- \* cmdDone - Input - 1bit
- \* Indicates this is the last word in the command
- \* cmdReady - Output - 1bit
- \* This will be high if LCD has accepted a command
- \*
- \* LT24\* - Outputs
- \* LT24 external display interface
- \*/

//Useful Macro - calculates ceil(log2(x))

```
`define CLOG2(x) \
    (x <= 2) ? 1 : \
    (x <= 4) ? 2 : \
    (x <= 8) ? 3 : \
    (x <= 16) ? 4 : \
    (x <= 32) ? 5 : \
    (x <= 64) ? 6 : \
    (x <= 128) ? 7 : \
    (x <= 256) ? 8 : \
    (x <= 512) ? 9 : \
    (x <= 1024) ? 10 : \
    (x <= 2048) ? 11 : \
    (x <= 4096) ? 12 : \
    (x <= 8192) ? 13 : \
    (x <= 16384) ? 14 : \
    (x <= 32768) ? 15 : \
    (x <= 65536) ? 16 : \
    (x <= 131072) ? 17 : \
```

```

(x <= 262144) ? 18 : \
(x <= 524288) ? 19 : \
(x <= 1048576) ? 20 : \
(x <= 2097152) ? 21 : \
(x <= 4194304) ? 22 : \
(x <= 8388608) ? 23 : \
(x <= 16777216) ? 24 : \

-1 //This one will trigger compiler error.

//This is to suppress a warning about a missing port on the inferred rom.
//You should not suppress warnings this way.
(* altera_attribute = "-name MESSAGE_DISABLE 10030" *)

//Main Display Module. This is the one that you should infer.
module LT24Display #(
    //Clock frequency
    parameter CLOCK_FREQ = 50000000,
    //Display Specs
    parameter WIDTH = 240,
    parameter HEIGHT = 320,
    parameter XBITS = `CLOG2(WIDTH),
    parameter YBITS = `CLOG2(HEIGHT)
)(
    //
    // Global Clock/Reset
    // - Clock
    input        clock,
    // - Global Reset
    input        globalReset,
    // - Application Reset
    output       resetApp,

    //
    // FPGA Data Interface
    // - Address
    input  [XBITS-1:0] xAddr,
    input  [YBITS-1:0] yAddr,
    // - Data
    input  [ 15:0] pixelData,

```

```

// - Write Request
input      pixelWrite,

// - Write Done
output reg pixelReady,

// - Raw Pixel Mode
input      pixelRawMode,


//

// FPGA Command Interface

// - Data
input [ 7:0] cmdData,

// - Write Request
input      cmdWrite,

// - Command Done
input      cmdDone,

// - Ready for command
output reg cmdReady,


//

// LT24 Interface

// - Write Strobe (inverted)
output     LT24Wr_n,

// - Read Strobe (inverted)
output     LT24Rd_n,

// - Chip Select (inverted)
output     LT24CS_n,

// - Register Select
output     LT24RS,

// - LCD Reset
output     LT24Reset_n,

// - LCD Data
output [ 15:0] LT24Data,

// - LCD Backlight On/Off
output     LT24LCDOn
);

assign LT24LCDOn = 1'b1; //Backlight always on.

/*

```

```

* Create power-on synchronous reset
*/

wire reset; //Synchronised Reset
ResetSynchroniser resetGen (
    .clock (clock    ), //Global clock
    .resetIn (globalReset), //Global reset
    .resetOut(reset    ) //Synchronised reset
);

/*
* LCD Initialisation Data
*/

reg [6:0] initDataRomAddr;
wire [6:0] initRomMaxAddr;
wire [8:0] initData;
LT24InitialData #(
    .WIDTH(WIDTH),
    .HEIGHT(HEIGHT)
) initDataRom (
    .clock (clock    ),
    .addr (initDataRomAddr),
    .initData(initData    ),
    .maxAddr (initRomMaxAddr )
);

/*
* State Machine for display interface
*/

reg    displayInitialised;
wire    displayReady;
reg    displayRegSelect;
reg [15:0] displayData;
reg    displayWrite;

reg [15:0] xAddrTemp; //Temp address registers are 16 bits wide so that address is correctly
reg [15:0] yAddrTemp; //padded for sending to the LT24 display.
reg [15:0] pixelDataTemp;

```

```

//State Machine
reg [3:0] stateMachine;

//General States
localparam INIT_STATE = 4'b0000; //Display Initialisation
localparam LOAD_STATE = 4'b0001; //Load Initialisation Data
localparam IDLE_STATE = 4'b1111; //Idle

//Pixel Write States
localparam CASET_STATE = 4'b1110; //Column Select
localparam XHADDR_STATE = 4'b1101; //Load X-High Address
localparam XLADDR_STATE = 4'b1100; //Load X-Low Address
localparam PASET_STATE = 4'b1011; //Row Select
localparam YHADDR_STATE = 4'b1010; //Load Y-High Address
localparam YLADDR_STATE = 4'b1001; //Load Y-Low Address
localparam WRITE_STATE = 4'b1000; //Memory Write Enable

//Command Write States
localparam CMD_STATE = 4'b0111; //Send command

//State Machine Code
always @ (posedge clock or posedge reset) begin
    if (reset) begin
        displayRegSelect    <= 1'b1;
        displayData         <= 16'b0;
        displayWrite        <= 1'b0;
        displayInitialised  <= 1'b0;
        initDataRomAddr     <= 7'b0;
        cmdReady            <= 1'b0;          //Not ready for a command
        pixelReady          <= 1'b0;          //Not ready for data
        stateMachine        <= INIT_STATE;    //Come out of reset into initial state
    end else begin
        case (stateMachine)
            INIT_STATE: begin //Power on state - initialise pins
                displayInitialised <= 1'b0;          //Display not yet initialised
                initDataRomAddr    <= 7'b0;          //Set initial ROM address one cycle early
                stateMachine       <= LOAD_STATE;    //Begin load of ROM data into LT24
            end
            LOAD_STATE: begin //Load the initialisation data
                if (displayReady) begin //Load next cycle when display is ready

```

```

//Send initialisation command/data for previous address. (ROM has one cycle latency)

displayData    <= {8'b0,initData[7:0]};    //Repackage ROM payload

displayRegSelect <= initData[8];           //Select command or data

displayWrite    <= 1'b1;                   //Issue a write

//Prepare next address and state

if (initDataRomAddr < initRomMaxAddr) begin

    //If we have not yet sent all ROM data

    initDataRomAddr <= initDataRomAddr + 7'b1;    //Increment to address

end else begin

    //Otherwise all initialisation data has been loaded

    stateMachine    <= IDLE_STATE;           //Move to Idle state to await commands.

end

end

end

IDLE_STATE: begin

displayInitialised <= 1'b1;                //Display is fully initialised once in Idle state.

if (displayReady && cmdWrite) begin

    //If command write requested (this has priority)

    cmdReady        <= 1'b1;                //Accepted command

    pixelReady      <= 1'b0;                //Not ready for data

    //Issue first write in sequence

    displayData     <= {8'b0,cmdData};       //Load command data onto display output

    displayWrite     <= 1'b1;                //Issue a write

    displayRegSelect <= 1'b0;                //Loading a command word

    stateMachine     <= CMD_STATE;           //Jump to command state for payload

end else if (displayReady && pixelWrite) begin

    //Otherwise if pixel write requested

    cmdReady        <= 1'b0;                //Not ready for a command

    pixelReady      <= 1'b1;                //Accepted data

    //Backup pixel information for later in the state machine

    xAddrTemp       <= {{(16-XBITS){1'b0}},xAddr}; //Store the current x

    yAddrTemp       <= {{(16-YBITS){1'b0}},yAddr}; //and y addresses

    pixelDataTemp    <= pixelData;           //Store the current pixel data

    if (pixelRawMode) begin

        //If in raw pixel mode, just load the data

        displayRegSelect<= 1'b1;            //Loading a pixel data word

        displayData    <= pixelData;        //Load raw pixel data

    end else begin

        //If in normal mode, go through setting of X-Y coordinates.

```



```

        displayRegSelect<= 1'b0;           //Loading a command word

        displayData    <= 16'h2A;         //Load CASET command onto display output

        stateMachine   <= CASET_STATE;    //Jump to CASET state

    end

    //Issue first write in sequence

    displayWrite       <= 1'b1;           //Issue a write
end else begin

    //Otherwise we are not ready until an access is requested

    cmdReady           <= 1'b0;           //Not ready for a command

    pixelReady         <= 1'b0;           //Not ready for data

    displayWrite       <= 1'b0;           //Don't perform write while in Idle

end

end

CMD_STATE: begin

    cmdReady           <= displayReady;    //Control flow of data into command port based on when display is ready

    if (displayReady && cmdWrite) begin

        //If additional payload, load next byte

        displayData    <= {8'b0,cmdData}; //Load command data onto display output

        displayWrite    <= 1'b1;           //Issue a write

        if (cmdDone) begin

            //If this is the last command, go to cleanup state

            stateMachine <= IDLE_STATE;

        end

    end else begin

        displayWrite    <= 1'b0;           //Write done

    end

end

end

CASET_STATE: begin //Column Select

    pixelReady         <= 1'b0;           //Not ready for next pixel.

    if (displayReady) begin

        //Once the display is ready for next transfer

        displayRegSelect <= 1'b1;           //Next display write is a payload

        displayData     <= {8'b0,xAddrTemp[15:8]}; //Load X-High payload onto display output

        displayWrite     <= 1'b1;           //Issue a write

        stateMachine     <= XHADDR_STATE;    //Next payload will be X-High

    end

end

end

XHADDR_STATE: begin //Load X-High Address

    if (displayReady) begin

```

```

//Once the display is ready for next transfer
displayRegSelect <= 1'b1;           //Next display write is a payload
displayData      <= {8'b0,xAddrTemp[7:0]}; //Load X-Low payload onto display output
displayWrite     <= 1'b1;           //Issue a write
stateMachine     <= XLADDR_STATE;    //Next payload will be X-Low
end
end
XLADDR_STATE: begin //Load X-Low Address
if (displayReady) begin
//Once the display is ready for next transfer
stateMachine     <= PASET_STATE;     //Next command will be PASET
displayRegSelect <= 1'b0;           //Next display write is a command
displayData      <= 16'h2B;         //Load PASET command onto display output
displayWrite     <= 1'b1;           //Issue a write
end
end
PASET_STATE: begin //Row Select
if (displayReady) begin
//Once the display is ready for next transfer
stateMachine     <= YHADDR_STATE;    //Next payload will be Y-High
displayRegSelect <= 1'b1;           //Next display write is a payload
displayData      <= {8'b0,yAddrTemp[15:8]}; //Load Y-High payload onto display output
displayWrite     <= 1'b1;           //Issue a write
end
end
YHADDR_STATE: begin //Load Y-High Address
if (displayReady) begin
//Once the display is ready for next transfer
stateMachine     <= YLADDR_STATE;    //Next payload will be Y-Low
displayRegSelect <= 1'b1;           //Next display write is a payload
displayData      <= {8'b0,yAddrTemp[7:0]}; //Load Y-Low payload onto display output
displayWrite     <= 1'b1;           //Issue a write
end
end
YLADDR_STATE: begin //Load Y-Low Address
if (displayReady) begin
//Once the display is ready for next transfer
stateMachine     <= WRITE_STATE;     //Next command will be Memory Write
displayRegSelect <= 1'b0;           //Next display write is a command

```

```

        displayData    <= 16'h2C;           //Load WRITE command onto display output
        displayWrite   <= 1'b1;           //Issue a write
    end
end

WRITE_STATE: begin //Memory Write Enable
    if (displayReady) begin
        //Once the display is ready for next transfer

        stateMachine    <= IDLE_STATE;       //Next payload will be Pixel Data and we are done.
        displayRegSelect <= 1'b1;           //Next display write is pixel data
        displayData     <= pixelDataTemp;    //Load pixel data onto display output
        displayWrite    <= 1'b1;           //Issue a write
    end
end

default: begin
    stateMachine    <= INIT_STATE;       //If something goes wrong, reinit the display.
end

endcase
end
end

```

//Hold application in reset until display initialised.

```
assign resetApp = reset || !displayInitialised;
```

```
/*
```

```
* Generate interface signals for LT24
```

```
*/
```

```
LT24DisplayInterface #(
```

```
    .CLOCK_FREQ(CLOCK_FREQ)
```

```
) LT24Interface (
```

```
    //Clock/Reset
```

```
    .clock    (clock        ), //Global clock
```

```
    .reset    (reset        ), //Synchronised reset
```

```
    //FPGA Display Interface
```

```
    .regSelect (displayRegSelect),
```

```
    .data      (displayData   ),
```

```
    .write     (displayWrite  ),
```

```
    .ready     (displayReady  ),
```

```

//LT24 Interface

.LT24Wr_n (LT24Wr_n    ),
.LT24Rd_n (LT24Rd_n    ),
.LT24CS_n (LT24CS_n    ),
.LT24RS   (LT24RS      ),
.LT24Reset_n(LT24Reset_n    ),
.LT24Data  (LT24Data     )
);

```

```

endmodule

```

```

/*
 * LT24 Display Interface
 * -----
 * Generates transactions on the LT24 display interface to
 * send data or commands.
 *
 * The FPGA interface signals should not change while ready is 0
 */
module LT24DisplayInterface #(
    parameter CLOCK_FREQ = 64'd50000000
)(
    input          clock,
    input          reset,

    //FPGA Interface
    input          regSelect, //1 = Pixel Data, 0 = Command
    input  [15:0] data,
    input          write,    //Assert for 1 cycle to initialise write
    output         ready,    //Indicates the LCD is ready to receive data/command

    //LCD Interface
    output         LT24Wr_n,

```

```

        output      LT24Rd_n,
        output      LT24CS_n,
        output      LT24RS,
        output      LT24Reset_n,
        output      [ 15:0] LT24Data
    );

/*
 * Reset pause timing
 */

//Determine requirements for power on reset
`ifdef MODEL_TECH //For Simulation
localparam RESETTIME = 64'd0; //ms - Skip delay during simulation.
`else //For Synthesis
localparam RESETTIME = 64'd120; //ms - 120ms is required by display
`endif //End preprocessor block

localparam RESETCOUNT = (RESETTIME * CLOCK_FREQ) / 64'd1000;

localparam RESETBITS = `CLOG2(RESETCOUNT+1); //The +1 ensures we always have room for counter to == RESETCOUNT

//These help make parameterised widths easier.
localparam ZERO = 0;
localparam ONE = 1;

//Reset delay counter
reg [RESETBITS-1:0] counter;

always @ (posedge clock or posedge reset) begin
    if (reset) begin
        counter <= ZERO[RESETBITS-1:0]; //Initially counter is zero
    end else if (counter < RESETCOUNT) begin //Once we are out of reset
        //If we haven't had a long enough reset pause
        counter <= counter + ONE[RESETBITS-1:0]; //Increment counter
    end
end

/*
 * Write control
 */

```

```

//Writes take two cycles, so track second cycle.

reg writeDly = 1'b0;

always @ (posedge clock) begin

    writeDly <= write && !writeDly;

end


//Assert ready after the reset pause and when we are not in the first write cycle
assign ready = !(counter < RESETCOUNT) && !(write && !writeDly);


/*
 * External interface
 */

assign LT24CS_n    = 1'b0;
assign LT24Rd_n    = 1'b1;
assign LT24Data     = data;
assign LT24RS       = regSelect;
assign LT24Wr_n     = writeDly;
assign LT24Reset_n  = !reset;

endmodule


/*
 * Simple Reset Synchroniser
 * -----
 * This will generate a few clock cycle reset at power on
 * or when the input reset is asserted.
 */

module ResetSynchroniser (

    input clock,

    input resetIn,

    output resetOut

);

//Reset synchroniser to avoid metastability if external push-button used

```

```
reg [3:0] resetSync = 4'hF;
```

```
always @ (posedge clock or posedge resetIn) begin
```

```
    if (resetIn) begin
```

```
        resetSync <= 4'hF; //Assert reset asynchronously
```

```
    end else begin
```

```
        resetSync <= {resetSync[2:0],1'b0}; //Deassert reset synchronously
```

```
    end
```

```
end
```

```
assign resetOut = resetSync[3];
```

```
endmodule
```

```
/*
```

```
 * Initialisation Data Lookup Table
```

```
 * -----
```

```
 * Contains initialisation data for LT24 Display
```

```
*/
```

```
module LT24InitialData #(
```

```
    parameter WIDTH = 240,
```

```
    parameter HEIGHT = 320
```

```
)(
```

```
    input      clock,
```

```
    input  [6:0] addr,
```

```
    output reg [8:0] initData,
```

```
    output  [6:0] maxAddr
```

```
);
```

```
localparam MAX_X_PIXEL = WIDTH - 1;
```

```
localparam MAX_Y_PIXEL = HEIGHT - 1;
```

```
localparam INIT_LENGTH = 102;
```

```
assign maxAddr = INIT_LENGTH[6:0]; //This can be used to determine when full ROM has been read.
```

```
localparam ROM_LENGTH = 2**(^CLOG2(INIT_LENGTH)); //Find next highest power of two that will fit the init data.
```

```
reg [8:0] ROM [ROM_LENGTH-1:0];
```

```
integer i;
```

```
initial begin
```

```
    //Note - this is ugly. A better approach is to use a .mif file.
```

```
    ROM[7'd000] <= {1'b0,8'hEF};
```

```
    ROM[7'd001] <= {1'b1,8'h03};
```

```
    ROM[7'd002] <= {1'b1,8'h80};
```

```
    ROM[7'd003] <= {1'b1,8'h02};
```

```
    ROM[7'd004] <= {1'b0,8'hCF};
```

```
    ROM[7'd005] <= {1'b1,8'h00};
```

```
    ROM[7'd006] <= {1'b1,8'h81};
```

```
    ROM[7'd007] <= {1'b1,8'hc0};
```

```
    ROM[7'd008] <= {1'b0,8'hED};
```

```
    ROM[7'd009] <= {1'b1,8'h64};
```

```
    ROM[7'd010] <= {1'b1,8'h03};
```

```
    ROM[7'd011] <= {1'b1,8'h12};
```

```
    ROM[7'd012] <= {1'b1,8'h81};
```

```
    ROM[7'd013] <= {1'b0,8'hE8};
```

```
    ROM[7'd014] <= {1'b1,8'h85};
```

```
    ROM[7'd015] <= {1'b1,8'h01};
```

```
    ROM[7'd016] <= {1'b1,8'h78};
```

```
    ROM[7'd017] <= {1'b0,8'hCB};
```

```
    ROM[7'd018] <= {1'b1,8'h39};
```

```
    ROM[7'd019] <= {1'b1,8'h2C};
```

```
    ROM[7'd020] <= {1'b1,8'h00};
```

```
    ROM[7'd021] <= {1'b1,8'h34};
```

```
    ROM[7'd022] <= {1'b1,8'h02};
```

```
    ROM[7'd023] <= {1'b0,8'hF7};
```

```
    ROM[7'd024] <= {1'b1,8'h20};
```

```
    ROM[7'd025] <= {1'b0,8'hEA};
```

```
    ROM[7'd026] <= {1'b1,8'h00};
```

```
    ROM[7'd027] <= {1'b1,8'h00};
```

```
    ROM[7'd028] <= {1'b0,8'hC0};
```

```
    ROM[7'd029] <= {1'b1,8'h23};
```

```
    ROM[7'd030] <= {1'b0,8'hC1};
```

```
    ROM[7'd031] <= {1'b1,8'h10};
```

```
    ROM[7'd032] <= {1'b0,8'hC5};
```



```
ROM[7'd033] <= {1'b1,8'h3E};
ROM[7'd034] <= {1'b1,8'h28};
ROM[7'd035] <= {1'b0,8'hC7};
ROM[7'd036] <= {1'b1,8'h86};
ROM[7'd037] <= {1'b0,8'h36};
ROM[7'd038] <= {1'b1,8'h48};
ROM[7'd039] <= {1'b0,8'h3A};
ROM[7'd040] <= {1'b1,8'h55};
ROM[7'd041] <= {1'b0,8'hB1};
ROM[7'd042] <= {1'b1,8'h00};
ROM[7'd043] <= {1'b1,8'h1b};
ROM[7'd044] <= {1'b0,8'hB6};
ROM[7'd045] <= {1'b1,8'h08};
ROM[7'd046] <= {1'b1,8'h82};
ROM[7'd047] <= {1'b1,8'h27};
ROM[7'd048] <= {1'b0,8'hF2};
ROM[7'd049] <= {1'b1,8'h00};
ROM[7'd050] <= {1'b0,8'h26};
ROM[7'd051] <= {1'b1,8'h01};
ROM[7'd052] <= {1'b0,8'hE0};
ROM[7'd053] <= {1'b1,8'h0F};
ROM[7'd054] <= {1'b1,8'h31};
ROM[7'd055] <= {1'b1,8'h2B};
ROM[7'd056] <= {1'b1,8'h0C};
ROM[7'd057] <= {1'b1,8'h0E};
ROM[7'd058] <= {1'b1,8'h08};
ROM[7'd059] <= {1'b1,8'h4E};
ROM[7'd060] <= {1'b1,8'hF1};
ROM[7'd061] <= {1'b1,8'h37};
ROM[7'd062] <= {1'b1,8'h07};
ROM[7'd063] <= {1'b1,8'h10};
ROM[7'd064] <= {1'b1,8'h03};
ROM[7'd065] <= {1'b1,8'h0E};
ROM[7'd066] <= {1'b1,8'h09};
ROM[7'd067] <= {1'b1,8'h00};
ROM[7'd068] <= {1'b0,8'hE1};
ROM[7'd069] <= {1'b1,8'h00};
ROM[7'd070] <= {1'b1,8'h0E};
ROM[7'd071] <= {1'b1,8'h14};
```

```

ROM[7'd072] <= {1'b1,8'h03};
ROM[7'd073] <= {1'b1,8'h11};
ROM[7'd074] <= {1'b1,8'h07};
ROM[7'd075] <= {1'b1,8'h31};
ROM[7'd076] <= {1'b1,8'hC1};
ROM[7'd077] <= {1'b1,8'h48};
ROM[7'd078] <= {1'b1,8'h08};
ROM[7'd079] <= {1'b1,8'h0F};
ROM[7'd080] <= {1'b1,8'h0C};
ROM[7'd081] <= {1'b1,8'h31};
ROM[7'd082] <= {1'b1,8'h36};
ROM[7'd083] <= {1'b1,8'h0f};
ROM[7'd084] <= {1'b0,8'hB1};
ROM[7'd085] <= {1'b1,8'h00};
ROM[7'd086] <= {1'b1,8'h01};
ROM[7'd087] <= {1'b0,8'hf6};
ROM[7'd088] <= {1'b1,8'h01};
ROM[7'd089] <= {1'b1,8'h10};
ROM[7'd090] <= {1'b1,8'h00};
ROM[7'd091] <= {1'b0,8'h11};
ROM[7'd092] <= {1'b0,8'h2A};
ROM[7'd093] <= {1'b1,8'h00};
ROM[7'd094] <= {1'b1,8'h00};
ROM[7'd095] <= {1'b1,MAX_X_PIXEL[15:8]};
ROM[7'd096] <= {1'b1,MAX_X_PIXEL[ 7:0]};
ROM[7'd097] <= {1'b0,8'h2B};
ROM[7'd098] <= {1'b1,8'h00};
ROM[7'd099] <= {1'b1,8'h00};
ROM[7'd100] <= {1'b1,MAX_Y_PIXEL[15:8]};
ROM[7'd101] <= {1'b1,MAX_Y_PIXEL[ 7:0]};
ROM[7'd102] <= {1'b0,8'h29};

for (i = INIT_LENGTH+1; i < ROM_LENGTH; i=i+1) begin
    ROM[i] <= {1'b0,8'h00}; //Pad others as NOP command.
end

end

always @ (posedge clock) begin
    initData <= ROM[addr];
end

```

```
endmodule
```

## Pin Assignment.tcl

```
# Require quartus project
```

```
package require ::quartus::project
```

```
# Set pin locations for LCD on GPIO 0
```

```
set_location_assignment PIN_AJ17 -to LT24Data[0]
```

```
set_location_assignment PIN_AJ19 -to LT24Data[1]
```

```
set_location_assignment PIN_AK19 -to LT24Data[2]
```

```
set_location_assignment PIN_AK18 -to LT24Data[3]
```

```
set_location_assignment PIN_AE16 -to LT24Data[4]
```

```
set_location_assignment PIN_AF16 -to LT24Data[5]
```

```
set_location_assignment PIN_AG17 -to LT24Data[6]
```

```
set_location_assignment PIN_AA18 -to LT24Data[7]
```

```
set_location_assignment PIN_AA19 -to LT24Data[8]
```

```
set_location_assignment PIN_AE17 -to LT24Data[9]
```

```
set_location_assignment PIN_AC20 -to LT24Data[10]
```

```
set_location_assignment PIN_AH19 -to LT24Data[11]
```

```
set_location_assignment PIN_AJ20 -to LT24Data[12]
```

```
set_location_assignment PIN_AH20 -to LT24Data[13]
```

```
set_location_assignment PIN_AK21 -to LT24Data[14]
```

```
set_location_assignment PIN_AD19 -to LT24Data[15]
```

```
set_location_assignment PIN_AG20 -to LT24Reset_n
```

```
set_location_assignment PIN_AG16 -to LT24RS
```

```
set_location_assignment PIN_AD20 -to LT24CS_n
```

```
set_location_assignment PIN_AH18 -to LT24Rd_n
```

```
set_location_assignment PIN_AH17 -to LT24Wr_n
```

```
set_location_assignment PIN_AJ21 -to LT24LCDOn
```

```
# Set pin location for Clock
```

```
set_location_assignment PIN_AA16 -to clock
```

```
# Set pin location for globalReset
```

```
set_location_assignment PIN_AA14 -to globalReset
```

```
# Set pin location for 7 segment displays
```

set\_location\_assignment PIN\_AE26 -to seg1[0]  
set\_location\_assignment PIN\_AE27 -to seg1[1]  
set\_location\_assignment PIN\_AE28 -to seg1[2]  
set\_location\_assignment PIN\_AG27 -to seg1[3]  
set\_location\_assignment PIN\_AF28 -to seg1[4]  
set\_location\_assignment PIN\_AG28 -to seg1[5]  
set\_location\_assignment PIN\_AH28 -to seg1[6]

set\_location\_assignment PIN\_AJ29 -to seg2[0]  
set\_location\_assignment PIN\_AH29 -to seg2[1]  
set\_location\_assignment PIN\_AH30 -to seg2[2]  
set\_location\_assignment PIN\_AG30 -to seg2[3]  
set\_location\_assignment PIN\_AF29 -to seg2[4]  
set\_location\_assignment PIN\_AF30 -to seg2[5]  
set\_location\_assignment PIN\_AD27 -to seg2[6]

set\_location\_assignment PIN\_AB23 -to seg3[0]  
set\_location\_assignment PIN\_AE29 -to seg3[1]  
set\_location\_assignment PIN\_AD29 -to seg3[2]  
set\_location\_assignment PIN\_AC28 -to seg3[3]  
set\_location\_assignment PIN\_AD30 -to seg3[4]  
set\_location\_assignment PIN\_AC29 -to seg3[5]  
set\_location\_assignment PIN\_AC30 -to seg3[6]

set\_location\_assignment PIN\_AD26 -to seg4[0]  
set\_location\_assignment PIN\_AC27 -to seg4[1]  
set\_location\_assignment PIN\_AD25 -to seg4[2]  
set\_location\_assignment PIN\_AC25 -to seg4[3]  
set\_location\_assignment PIN\_AB28 -to seg4[4]  
set\_location\_assignment PIN\_AB25 -to seg4[5]  
set\_location\_assignment PIN\_AB22 -to seg4[6]

set\_location\_assignment PIN\_AA24 -to seg5[0]  
set\_location\_assignment PIN\_Y23 -to seg5[1]  
set\_location\_assignment PIN\_Y24 -to seg5[2]  
set\_location\_assignment PIN\_W22 -to seg5[3]  
set\_location\_assignment PIN\_W24 -to seg5[4]  
set\_location\_assignment PIN\_V23 -to seg5[5]  
set\_location\_assignment PIN\_W25 -to seg5[6]

```
set_location_assignment PIN_V25 -to seg6[0]
set_location_assignment PIN_AA28 -to seg6[1]
set_location_assignment PIN_Y27 -to seg6[2]
set_location_assignment PIN_AB27 -to seg6[3]
set_location_assignment PIN_AB26 -to seg6[4]
set_location_assignment PIN_AA26 -to seg6[5]
set_location_assignment PIN_AA25 -to seg6[6]
```

# Set pin location for resetApp

```
set_location_assignment PIN_Y21 -to resetApp
```

# Set Pin Location for Slide Button

```
set_location_assignment PIN_AB12 -to slide_button[0]
set_location_assignment PIN_AC12 -to slide_button[1]
set_location_assignment PIN_AF9 -to slide_button[2]
```

# Set Pin Location for Threshold Slide Buttons

```
set_location_assignment PIN_AF10 -to thres_switch[0]
set_location_assignment PIN_AD11 -to thres_switch[1]
set_location_assignment PIN_AD12 -to thres_switch[2]
set_location_assignment PIN_AE11 -to thres_switch[3]
set_location_assignment PIN_AC9 -to thres_switch[4]
set_location_assignment PIN_AD10 -to thres_switch[5]
set_location_assignment PIN_AE12 -to thres_switch[6]
```

# Set Pin Location for Push Button

```
set_location_assignment PIN_AA15 -to button2
set_location_assignment PIN_W15 -to button3
```

#Set Pin Location for debug LEDS

```
set_location_assignment PIN_W20 -to debug1
set_location_assignment PIN_W16 -to debug2
set_location_assignment PIN_V17 -to debug3
set_location_assignment PIN_V18 -to debug4
```

```
# Commit assignments
```

```
export_assignments
```

## mif2.mif – Shortened

```
DEPTH = 32000;
```

```
WIDTH = 8;
```

```
ADDRESS_RADIX = UNS;
```

```
DATA_RADIX = HEX;
```

```
CONTENT BEGIN
```

```
[0..31999] : 0;
```

```
0 : 5;
```

## data.mif - Shortened

```
DEPTH = 30000;
```

```
WIDTH = 8;
```

```
ADDRESS_RADIX = UNS;
```

```
DATA_RADIX = HEX;
```

```
CONTENT BEGIN
```

```
[0..29999] : 0;
```

```
0 : 89;
```

## Sobel Edge Matlab

```
raw_pixels = imread ( ' test_image.gif' );
```

```
image_size = size ( raw_pixels );
```

```
row = image_size ( 1 );
```

```
col = image_size ( 2 );
```

```
thresh = 100;
```

```
for r = 1 : row
```

```
    for c = 1 : col
```

```
        if ( c == 1 || c == col)
```

```
            gradient ( r , c ) = 100;
```

```
        elseif ( r == 1 || r == row )
```

```
            gradient ( r,c ) = 100;
```

```
        else
```

```

        gradient ( r,c ) = abs ( (raw_pixels ( r-1, c-1 ) + 2*( raw_pixels ( r-1,c ) ) + raw_pixels( r-1, c+1)) - ( raw_pixels(r+1,c-1) + ( 2 *
raw_pixels( r+1, c ) ) + raw_pixels( r+1, c+1 ))) + abs ( ( raw_pixels( r-1 , c+1 ) + ( 2* raw_pixels ( r,c+1)) + raw_pixels ( r+1,c+1)) - (
raw_pixels(r-1,c-1) + ( 2* raw_pixels (r,c-1)) + raw_pixels(r+1,c-1)));

```

```

    end

```

```

end

```

```

end

```

```

edge = gradient;

```

```

imshow ( gradient );

```

### Arithmetic Test.v

```

/*

```

Module : Arithmetic\_test.v

Module Description : Used to test uint8 addition in verilog code. Max value of uint8 is 255, if any value higher than 255 occurs, then value of the output is set to 255.

Author : ARUL PRAKASH SAMATHUVAMANI - UNIVERSITY OF LEEDS

Build Date : 10 May 2021

```

*/

```

```

module Arithmetic_test(

```

```

    input [7:0]input_bits,

```

```

    input [7:0]input_bits2,

```

```

    input clock,

```

```

    output reg[8:0]output_bits

```

```

);

```

```

always @( posedge clock ) begin

```

```
output_bits = input_bits + input_bits2;
```

```
if ( output_bits[8] ) begin
```

```
    output_bits = 9'b0;
```

```
end
```

```
end
```

```
endmodule
```

### **Arithmetic Test Tb**

```
`timescale 1 ns/100 ps
```

```
module Arithmetic_test_tb;
```

```
// declaration of input bits
```

```
reg [7:0]input_bits;
```

```
reg [7:0]input_bits2;
```

```
reg clock;
```

```
// output wires
```

```
wire [8:0]output_bits;
```

```
// device under test
```

```
Arithmetic_test Arithmetic_test_dut (
```

```
    .input_bits ( input_bits ),
```

```
    .input_bits2 ( input_bits2 ),
```

```
    .clock ( clock ),
```

```
    .output_bits ( output_bits )
```

```
);
```



```
// Test Bench Logic
```

```
integer loopvariable;
```

```
integer truth = 0;
```

```
// Okay here is the logic - > for every addition, the max value should be 255. If its greater than 255, then the system should reset to zero
```

```
initial begin
```

```
    $display ( " Simulation Started " );
```

```
    // we are setting both the input bits to 240. The output then should be zero.
```

```
    input_bits = 8'd240;
```

```
    input_bits2 = 8'd240;
```

```
    clock = 1'b1; // enabling clock
```

```
    #10;
```

```
    if ( output_bits == 0 ) begin
```

```
        $display ( " Test Success " );
```

```
    end
```

```
    else begin
```

```
        $display ( " Error in design " );
```

```
    end
```

```
    clock = 1'b0;
```

```
    #10;
```

```
    // now we are setting the input to 120 and 40 respectively, then the output should be 160.
```

```
    clock = 1'b1;
```

```
    input_bits = 8'd120;
```

```
    input_bits2 = 8'd40;
```

```
    #10;
```

```

        if ( output_bits == 8'd160 ) begin

            $display ( " Test success" );

        end

        else begin

            $display ( " Error in design " );

        end

        clock = 1'b0;

    end

endmodule

state_calculation.v

/*

Module : states_calculation.v

Module Description : Module for checking Sobel Edge calculation

THIS MODULE IS FOR DEMONSTRATION ALONE, MERGED WITH FINAL MODULE

Author : ARUL PRAKASH SAMATHUVAMANI - UNIVERSITY OF LEEDS

Build Date : 10 May 2021

*/

// Module Declaration

module states_calculation #(

    // Module Parameters, Image Width and Image Height

    parameter IMAGE_WIDTH = 100,

    parameter IMAGE_HEIGHT = 100

)(

```

```
input clock, // input clock signal
```

```
// Output Pixels
```

```
output reg [7:0] pixel1,
```

```
output reg [7:0] pixel2,
```

```
output reg [7:0] pixel3,
```

```
output reg [7:0] pixel4,
```

```
output reg [7:0] pixel6,
```

```
output reg [7:0] pixel7,
```

```
output reg [7:0] pixel8,
```

```
output reg [7:0] pixel9,
```

```
// Output from Sobel Edge Detector
```

```
output reg [7:0] sobel_output,
```

```
output reg [3:0] state_debug,
```

```
// Output xAddr and yAddr
```

```
output reg [7:0] xaddress,
```

```
output reg [8:0] yaddress
```

```
);
```

```
reg [3:0] STATES;
```

```
// States Declaration
```

```
localparam STATE_INITIAL = 4'b0000; // initial state
```

```
localparam STATE1 = 4'b0001; // fetch pixel1
```

```
localparam ADDRESS1 = 4'b0010; // fetch pixel2
```

```
localparam ADDRESS2 = 4'b0011; // fetch pixel3
```

```
localparam ADDRESS3 = 4'b0100; // fetch pixel4
```

```
localparam ADDRESS4 = 4'b0101; // fetch pixel6
```

```
localparam ADDRESS5 = 4'b0110; // fetch pixel7
```

```
localparam ADDRESS6 = 4'b0111; // fetch pixel7
```

```
localparam ADDRESS7 = 4'b1000; // fetch pixel8
```

```
localparam ADDRESS8 = 4'b1001; // fetch pixel9
```

```
localparam STATE2 = 4'b1010; // calculation state1
```

```

localparam STATE3 = 4'b1011; // calculation state2,
localparam STATE4 = 4'b1100; // calculation state3
localparam STATE5 = 4'b1101; // calculation state4 -> gives output

initial begin

    STATES = 4'd0;

end

// Temp Variables used for calculation

reg [8:0] temp_variable1;
reg [8:0] temp_variable2;
reg [8:0] temp_variable3;
reg [8:0] temp_variable4;

reg [8:0] output_buffer; // output pixel

(* ram_init_file = "data.mif" *)
reg [7:0] input_image [ (( IMAGE_WIDTH ) * ( IMAGE_HEIGHT ) - 1 ) : 0 ]; // input image

reg [7:0] xAddr; // xaddress
reg [8:0] yAddr; // yaddress

reg [7:0] test_sample;

always @ ( posedge clock ) begin

    case ( STATES )

        STATE_INITIAL : begin // initial state -> set xaddr and yaddr to zero

            xAddr = 8'd0; // set xaddr to 0
            yAddr = 9'd0; // set yaddr to zero
            test_sample = input_image [ 0 ];
            pixel1 = test_sample;

```

```
STATES = 4'd1;  
state_debug = STATES;
```

```
end
```

```
STATE1 : begin // state 1 in calculation
```

```
    xaddress = xAddr;  
    yaddress = yAddr;
```

```
    if ( xAddr == 0 || xAddr == ( IMAGE_WIDTH - 1 ) ) begin // for all corner  
pixels, set the output to zero, and go to final state
```

```
        output_buffer = 8'b0; // set output to zero  
        STATES = 4'b1100; // go to final state  
        state_debug = STATES; // set current state to state debug
```

```
    end
```

```
    else if ( yAddr == 0 || yAddr == ( IMAGE_HEIGHT - 1 ) ) begin// for all  
corner pixels, set the output to zero, and go to final state
```

```
        output_buffer = 8'b0; // set output to zero  
        STATES = 4'b1100; // go to final state  
        state_debug = STATES; // set current state to state debug
```

```
    end
```

```
    else begin // else fetch the pixels
```

```
        STATES = 4'b0010;  
        state_debug = STATES;
```

```
    end
```

```
end
```

```
/*
```

```
P1 P2 P3
```

```
P4 P5 P6
```

P7 P8 P9

WHERE P5 IS CURRENT PIXEL

\*/

ADDRESS1 : begin

pixel1 = input\_image [ (( yAddr -1 ) \*IMAGE\_HEIGHT ) + ( xAddr -1 )]; // fetch P1 that is one  
row and and one col before current pixel

STATES = 4'b0011; // go to next pixel fetch

state\_debug = STATES;

end

ADDRESS2 : begin

pixel2 = input\_image [ (( yAddr -1 ) \* IMAGE\_HEIGHT ) + ( xAddr ) ]; // fetch p2 pixel

STATES = 4'b0100; // go to next state

state\_debug = STATES;

end

ADDRESS3 : begin

pixel3 = input\_image [ (( yAddr -1 ) \* IMAGE\_HEIGHT ) + ( xAddr + 1 ) ]; // fetch p3

STATES = 4'b0101; // go to next state

state\_debug = STATES;

end

ADDRESS4 : begin

pixel4 = input\_image [ ( yAddr \* IMAGE\_HEIGHT ) + ( xAddr - 1 ) ]; // fetch pixel p4

STATES = 4'b0110; // go to next pixel fetch state

state\_debug = STATES;

end

ADDRESS5 : begin

pixel6 = input\_image [ ( yAddr \* IMAGE\_HEIGHT ) + ( xAddr + 1 ) ]; // fetch pixel p6

STATES = 4'b0111; // go to next pixel fetch state

state\_debug = STATES;

end

ADDRESS6 : begin

pixel7 = input\_image [ ( ( yAddr + 1 ) \* IMAGE\_HEIGHT ) + ( xAddr + 1 ) ]; // fetch pixel p7

STATES = 4'b1000; // go to next pixel fetch state

state\_debug = STATES;

end

ADDRESS7 : begin

pixel8 = input\_image [ ( ( yAddr + 1 ) \* IMAGE\_HEIGHT ) + ( xAddr ) ]; // fetch pixel p8

STATES = 4'b1001; // go to next pixel fetch state

state\_debug = STATES;

end

ADDRESS8 : begin

pixel9 = input\_image [ ( ( yAddr + 1 ) \* IMAGE\_HEIGHT ) + ( xAddr + 1 ) ]; // fetch pixel p9

STATES = 4'b1010; // go to next pixel fetch state

state\_debug = STATES;

end

STATE2 : begin

// Vertical edge kernel calculation

temp\_variable1 = ( pixel1 + ( 2\* pixel2 ) + pixel3 );

temp\_variable2 = ( pixel7 + ( 2 \* pixel8 ) + pixel9 );

```

// Horizontal edge kernel calculation

temp_variable3 = ( pixel3 + ( 2* pixel6 ) + pixel9 );
temp_variable4 = ( pixel1 + ( 2* pixel4 ) + pixel7);

if ( temp_variable1 [8] ) begin // max value should be 255

    temp_variable1 = 9'd255;

end

if ( temp_variable2 [8] ) begin // max value should be 255
    temp_variable2 = 9'd255;

end

if ( temp_variable3 [8] ) begin // max value should be 255

    temp_variable3 = 9'd255;

end

if ( temp_variable4 [8] ) begin // max value should be 255

    temp_variable4 = 9'd255;

end

STATES = 4'b1011;
state_debug = STATES;

end

STATE3 : begin

    temp_variable1 = temp_variable1 - temp_variable2; // average value calculation
    temp_variable2 = temp_variable3 - temp_variable4; // average value calculation

    if ( temp_variable1 [8] ) begin // mininum value should be zero

```



```

        temp_variable1 = 9'd0;

    end

    if ( temp_variable2 [8] ) begin // mininum value should be zero

        temp_variable2 = 9'd0;

    end

    STATES = 4'b1100;
    state_debug = STATES;

end

STATE4 : begin

    output_buffer = temp_variable1 + temp_variable2; // find the final edge value

    if ( output_buffer [8] ) begin // max value shoule be 255

        output_buffer = 9'd255;

    end

    STATES = 4'b1101; // go to next state
    state_debug = STATES;

end

STATE5 : begin

    sobel_output = output_buffer [7:0]; // set the buffer value to output

    if ( xAddr < ( IMAGE_WIDTH -1 ) ) begin

        xAddr = xAddr + 1; // incerecent xaddr
    end
end

```

```

        end

    else begin

        xAddr = 0;

        if ( yAddr < ( IMAGE_HEIGHT -1 )) begin // set the buffer value to output

            yAddr = yAddr + 1; // increment address

        end

    else begin

        yAddr = 0;

    end

end

STATES = 4'b0001; // go to state 1
state_debug = STATES;

    end

endcase

end

endmodule

States_calculation_tb

`timescale 1 ns/100 ps

/*

```

Test Bench Verilog File

Module Name: states\_calculation\_tb

Module Desctiption: Check if the value of xAddr and yAddr moves correctly

Module Author: Arul Prakash Samathuvamani

\*/

module states\_calculation\_tb;

// declaration of input bits

reg clock;

// output pixels declaration

wire [7:0] pixel1;

wire [7:0] pixel2;

wire [7:0] pixel3;

wire [7:0] pixel4;

wire [7:0] pixel6;

wire [7:0] pixel7;

wire [7:0] pixel8;

wire [7:0] pixel9;

wire [7:0] sobel\_output;

wire [3:0] state\_debug;

wire [7:0] xaddress;

wire [8:0] yaddress;

states\_calculation states\_calculation\_dut (

.clock ( clock),

.pixel1 ( pixel1 ),

.pixel2 ( pixel2 ),

.pixel3 ( pixel3 ),

.pixel4 ( pixel4 ),

//.pixel5 ( pixel5 ),

.pixel6 ( pixel6 ),

.pixel7 ( pixel7 ),

.pixel8 ( pixel8 ),

.pixel9 ( pixel9 ),

```

        .sobel_output ( sobel_output ),
        .state_debug ( state_debug ),
        .xaddress ( xaddress ),
        .yaddress ( yaddress )

);

integer loop_variable;

(* ram_init_file = "data.mif" *)
reg [7:0] input_image [ 9999 : 0];

initial begin

    $display ( " Simulation Started ");

    // Run clock for 5000 times and check the value of xAddr and yAddr manually

    for ( loop_variable = 0 ; loop_variable < 10000 ; loop_variable = loop_variable + 1 ) begin

        clock = 1'b0;

        #10;
        clock = 1'b1;

        #10;

    end

end

endmodule

Image_Display.v

/*

```

Module Name : image\_display

Module Description: Read pixel information from RAM and display it on LT24 display

Module Author : Arul Prakash Samathuvamani

```
*/
```

```
// Module Declaration
```

```
module image_display (
```

```
    input clock, // input clock
```

```
    // LT24 PINS declaration
```

```
    input globalReset,
```

```
    output LT24Wr_n,
```

```
    output LT24Rd_n,
```

```
    output LT24CS_n,
```

```
    output LT24RS,
```

```
    output LT24Reset_n,
```

```
    output [ 15: 0 ] LT24Data,
```

```
    output LT24LCDOn
```

```
);
```

```
// Declaration of local variables
```

```
reg [ 7:0] xAddr; // X-Address
```

```
reg [ 8:0] yAddr; // Y-Address
```

```
reg [ 15:0] pixelData; // Input Pixel Data
```

```
// LT24 Control Wires
```

```
wire pixelReady;
```

```
reg pixelWrite;
```

```
// RAM initialisation with pixel data
```

```
(* ram_init_file = "ram.mif" *)
```

```
reg [ 7:0 ] imagedata [ 29999:0 ];
```

```
// Current Pixel Buffer
```

```
reg [7:0] image_buffer;
```

```
// LCD Display
```

```
localparam LCD_WIDTH = 240;
```

```
localparam LCD_HEIGHT = 320;
```

```
localparam IMAGE_HEIGHT = 100;
```

```
localparam IMAGE_WIDTH = 100;
```

```
integer image_address = 0;
```

```
// LT24 Module Call
```

```
LT24Display #(
```

```
    .WIDTH ( LCD_WIDTH ),
```

```
    .HEIGHT ( LCD_HEIGHT ),
```

```
    .CLOCK_FREQ ( 50000000 )
```

```
)Display (
```

```
    .clock ( clock ),
```

```
    .globalReset ( globalReset ),
```

```
    .resetApp ( resetApp ),
```

```
    .xAddr ( xAddr ),
```

```

        .yAddr ( yAddr ),
        .pixelData ( pixelData ),
        .pixelWrite ( pixelWrite ),
        .pixelReady ( pixelReady ),
        .pixelRawMode ( 1'b0 ),
        .cmdData ( 8'b0 ),
        .cmdWrite ( 1'b0 ),
        .cmdDone ( 1'b0 ),
        .cmdReady ( ),

        .LT24Wr_n ( LT24Wr_n ),
        .LT24Rd_n ( LT24Rd_n ),
        .LT24CS_n ( LT24CS_n ),
        .LT24RS ( LT24RS ),
        .LT24Reset_n ( LT24Reset_n ),
        .LT24Data ( LT24Data ),
        .LT24LCDOn ( LT24LCDOn )
    );

//
// X Counter
//
wire [7:0] xCount;
UpCounterNbit #(
    .WIDTH ( 8 ),
    .MAX_VALUE ( IMAGE_WIDTH - 1 )
) xCounter (
    .clock ( clock ),
    .reset ( resetApp ),
    .enable ( pixelReady ),
    .countValue ( xCount )
);

//
// Y Counter
//
wire [8:0] yCount;
wire yCntEnable = pixelReady && ( xCount == ( LCD_WIDTH - 1 ) );

```

```

UpCounterNbit #(
    .WIDTH ( 9),
    .MAX_VALUE(IMAGE_HEIGHT-1)
) yCounter (
    .clock (clock ),
    .reset (resetApp ),
    .enable (yCntEnable),
    .countValue(yCount )
);

// pixel write

always @ ( posedge clock ) begin

    pixelWrite = 1'b1;

end

always @ ( posedge clock ) begin

    xAddr= xCount;

    yAddr= yCount;

    if ( xAddr < IMAGE_WIDTH ) begin

        if ( yAddr < IMAGE_HEIGHT ) begin

            image_buffer = imagedata[ image_address ] >> 3 ; // Convert to 16 bit data

            // Copy the pixel data to display
            pixelData [ 15 : 11 ] = image_buffer [ 4: 0];
            pixelData [ 10 : 5 ] = image_buffer [ 4:0];
            pixelData [4:0] = image_buffer[4:0] + 5'b0;
            image_address = image_address + 1;

        end

    end

```



```
end
```

```
else begin
```

```
    pixelData = 16'b0; // set all other to black
```

```
end
```

```
end
```

```
endmodule
```

```
Calculation_test.v
```

```
/* Calculation_test.v
```

```
Module : Calculation Test
```

```
Module Description : Module used to test calculations for sebel edge detection
```

```
Module Author : Arul Prakash Samathuvamani for University of Leeds
```

```
Date: 10/5/2021
```

```
// Changelog:
```

```
*/
```

```
module calculation_test (
```

```
// Declaration of Inputs for the module
```

```
input clock,
```

```
output reg [7:0] gradient,
```

```
output reg [1:0] state_debug
```

```
);
```

```
(* ram_init_file = "data.mif"*)
```

```
reg [7:0] input_image [ 9999 : 0 ];
```

```
reg [7:0] pixel1;
```

```
reg [7:0] pixel2;
```

```
reg [7:0] pixel3;
```

```
reg [7:0] pixel4;
```

```
reg [7:0] pixel9;
```

```
reg [7:0] pixel6;
```

```
reg [7:0] pixel7;
```

```
reg [7:0] pixel8;
```

```
reg [7:0] pixel [ 299 : 0 ];
```

```
reg [ 31 : 0] address;
```

```
reg [1:0]STATE;
```

```
localparam i = 2;
```

```
localparam STATE1 = 2'b00;
```

```
localparam STATE2 = 2'b01;
```

```
localparam STATE3 = 2'b10;
```

```
initial begin
```

```
STATE = 2'd0;
```

```
end
```

```
reg [8:0] temp_variable1;
```

```
reg [8:0] temp_variable2;
```

```
reg [8:0] temp_variable3;
```

```
reg [8:0] temp_variable4;
```

```
reg [8:0] output_buffer;
```

```
always @ ( posedge clock ) begin
```

```
    case ( STATE )
```

```
        STATE1: begin
```

```
            //pixel1 = input_image [ address - 101 ];
```

```
            //pixel2 = input_image [ address -100 ];
```

```
            //pixel3 = input_image [ address - 99 ];
```

```
            //pixel4 = input_image [ address - 1];
```

```
            //pixel6 = input_image [ address + 1];
```

```
            //pixel7 = input_image [ address + 99 ];
```

```
            //pixel8 = input_image [ address + 100 ];
```

```
            pixel9 = input_image [ address + 101 ];
```

```
            temp_variable1 = ( pixel1 + ( 2 * pixel2 ) + pixel3 );
```

```
            temp_variable2 = ( pixel7 + ( 2* pixel8 ) + pixel9 );
```

```
            temp_variable3 = ( pixel3 + ( 2 * pixel6 ) + pixel9 );
```

```
            temp_variable4 = ( pixel1 + ( 2* pixel4 ) + pixel7 );
```

```
            if ( temp_variable1 [8] ) begin
```

```
                temp_variable1 = 9'd255;
```

```
end
```

```
if ( temp_variable2 [8] ) begin
```

```
    temp_variable2 = 9'd255;
```

```
end
```

```
if ( temp_variable3 [8] ) begin
```

```
    temp_variable3 = 9'd255 ;
```

```
end
```

```
if ( temp_variable4 [8] ) begin
```

```
    temp_variable4 = 9'd255;
```

```
end
```

```
STATE = 2'd1;
```

```
state_debug = STATE;
```

```
end
```

```
STATE2 : begin
```

```
    temp_variable1 = temp_variable1 - temp_variable2;
```

```
    temp_variable2 = temp_variable3 - temp_variable4;
```

```
if ( temp_variable1 [8] ) begin
```

```
    temp_variable1 = 9'd0;
```

```
end
```

```
if ( temp_variable2 [8] ) begin
```

```
    temp_variable2 = 9'd0;
```

```

        end

        STATE = 2'b10;

        state_debug = STATE;

    end

    STATE3 : begin

        output_buffer = temp_variable1 + temp_variable2;

        if ( output_buffer [8] ) begin

            output_buffer = 9'd255;

        end

        gradient = output_buffer [ 7: 0 ];

        STATE = 2'd0;

        state_debug = STATE;

    end

endcase

end

endmodule

```

### **Calculation\_test\_tb.v**

/\* Verilog Test Bench Module

Module Name : calculation\_test\_tb

Module Description: Check if the calculation runs correctly

Module Author : Arul Prakash Samathuvamani

```
*/
```

```
`timescale 1 ns/100 ps
```

```
// Test Bench Declaration
```

```
module calculation_test_tb;
```

```
// declaration of input bits
```

```
reg [7:0] pixel1;
```

```
reg [7:0] pixel2;
```

```
reg [7:0] pixel3;
```

```
reg [7:0] pixel4;
```

```
reg [7:0] pixel5;
```

```
reg [7:0] pixel6;
```

```
reg [7:0] pixel7;
```

```
reg [7:0] pixel8;
```

```
reg [7:0] pixel9;
```

```
reg clock;
```

```
// declaration of outputs
```

```
wire [7:0] gradient;
```

```
wire [1:0] state_debug;
```

```
// delcaration of test bench module dut
```

```
calculation_test calculation_test_dut (
```

```
    .pixel1 (pixel1),  
    .pixel2 (pixel2),  
    .pixel3 (pixel3),  
    .pixel4 (pixel4),  
    .pixel5 (pixel5),  
    .pixel6 (pixel6),  
    .pixel7 (pixel7),  
    .pixel8 (pixel8),  
    .pixel9 (pixel9),  
    .clock (clock),  
    .gradient (gradient),  
    .state_debug ( state_debug )
```

```
);
```

```
// Start the simulation
```

```
initial begin
```

```
    $display ( " Simulation Started " );
```

```
    // Set the input bits, value based on matlab
```

```
    pixel1 = 8'd11;
```

```
    pixel2 = 8'd9;
```

```
    pixel3 = 8'd120;
```

```
    pixel4 = 8'd118;
```

```
    pixel5 = 8'd7;
```

```
    pixel6 = 8'd11;
```

```
    pixel7 = 8'd116;
```

```
    pixel8 = 8'd117;
```

```
    pixel9 = 8'd12;
```

```
    // run for 3 clock cycles
```

```
    #10;
```

```
    clock = 1'b1;
```

```
#10;
```

```
clock = 1'b0;
```

```
#10;
```

```
clock = 1'b1;
```

```
#10;
```

```
clock = 1'b0;
```

```
#10;
```

```
clock = 1'b1;
```

```
#10;
```

```
clock = 1'b0;
```

```
#10;
```

```
// setting second set of bits
```

```
pixel1 = 8'd115;
```

```
pixel2 = 8'd118;
```

```
pixel3 = 8'd11;
```

```
pixel4 = 8'd117;
```

```
pixel5 = 8'd7;
```

```
pixel6 = 8'd118;
```

```
pixel7 = 8'd9;
```

```
pixel8 = 8'd11;
```

```
pixel9 = 8'd10;
```

```
// Run for three clock cycles
```

```
#10;
```



```
clock = 1'b1;

#10;

clock = 1'b0;

#10;

clock = 1'b1;

#10;

clock = 1'b0;

#10;

clock = 1'b1;

#10;

clock = 1'b0;

#10;

// setting third set of bits
pixel1 = 8'd7;
pixel2 = 8'd8;
pixel3 = 8'd114;
pixel4 = 8'd7;
pixel5 = 8'd8;
pixel6 = 8'd114;
pixel7 = 8'd9;
pixel8 = 8'd117;
pixel9 = 8'd8;

// run for three clock cycles
#10;

clock = 1'b1;

#10;
```

```
clock = 1'b0;
```

```
#10;
```

```
clock = 1'b1;
```

```
#10;
```

```
clock = 1'b0;
```

```
#10;
```

```
clock = 1'b1;
```

```
#10;
```

```
clock = 1'b0;
```

```
#10;
```

```
// End of Simulation
```

```
$display (" Simulation ended ");
```

```
end
```

```
endmodule
```