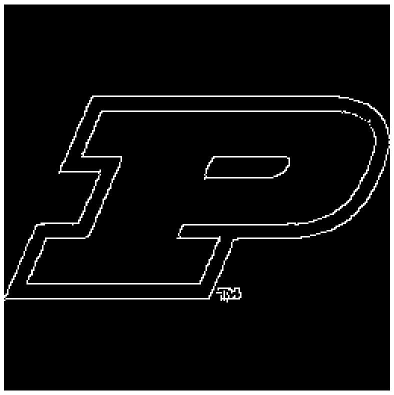


Edge Detection ASIC

Prepared By:

Suppatach Sabpisal

Hao Xiong

Sidharth Mudgal

Ryan Beasley

Akanksha Sharma

Final Report

ECE 337

May 8, 2014

Wednesday 11:30-2:20

TA : Utpal Mahanta

1 Executive Summary

We are designing a low energy application-specific integrated circuit that will operates as an auxiliary unit as part of a system on chip inside any camera - including security camera, or digital camera. The goal of this design is to give the ability of real-time still image manipulation to optical devices (computer monitors, surveillance camera, digital cameras, computer vision applications, etc.). The ASIC Image Processor has wide range of possible commercial and industrial applications. It can be used to interface over AHB bus on a system powered by ARM-based microprocessor to provide additional processing power. It can also be used to operate an image processing operations that requires processing of countless images repeatedly over time – for instance, if you want to process 2 images taken every 5 seconds for the next 5 years and feed that data into a hard drive (may require additional hardware. This is a critical point because if only one functionality is needed to be performed, ASIC architecture can provide order-of-magnitude reduction in power consumption compared to traditional methods. In addition, hardware implementation of image processing is done in more “real-time” fashion that it’s (generally) slower software counterpart. In industrial environment, it can be used as part of a system that inspect manufactured parts in the manufacturing line, in real-time. Our design of the ASIC Image Processor will be optimized for speed, because size doesn’t matter to us. It is trying to provide a “real-time” processing capability and supports wide range of image processing instruction sets and can be controlled by any ARM microprocessor over the AHB Bus.

This design is appropriate for ASIC because a microcontroller is too expensive for industrial applications, and consume too much power for commercial applications. In addition, when developing computer vision or image processing applications, a microcontrollers are too slow compared to ASIC because the system require almost instantaneous response. To perform this task it will require a SRAM interface, an ARM processor with AHB bus, and an expertise in image processing. The rest of this proposal contains extensive detail on the implementation of each block and their requirements, design specifications and image processing techniques used.

2.0 System Usage Diagram

The ASIC Image Processor interfaces with images on SRAM and the processor over the AHB bus.

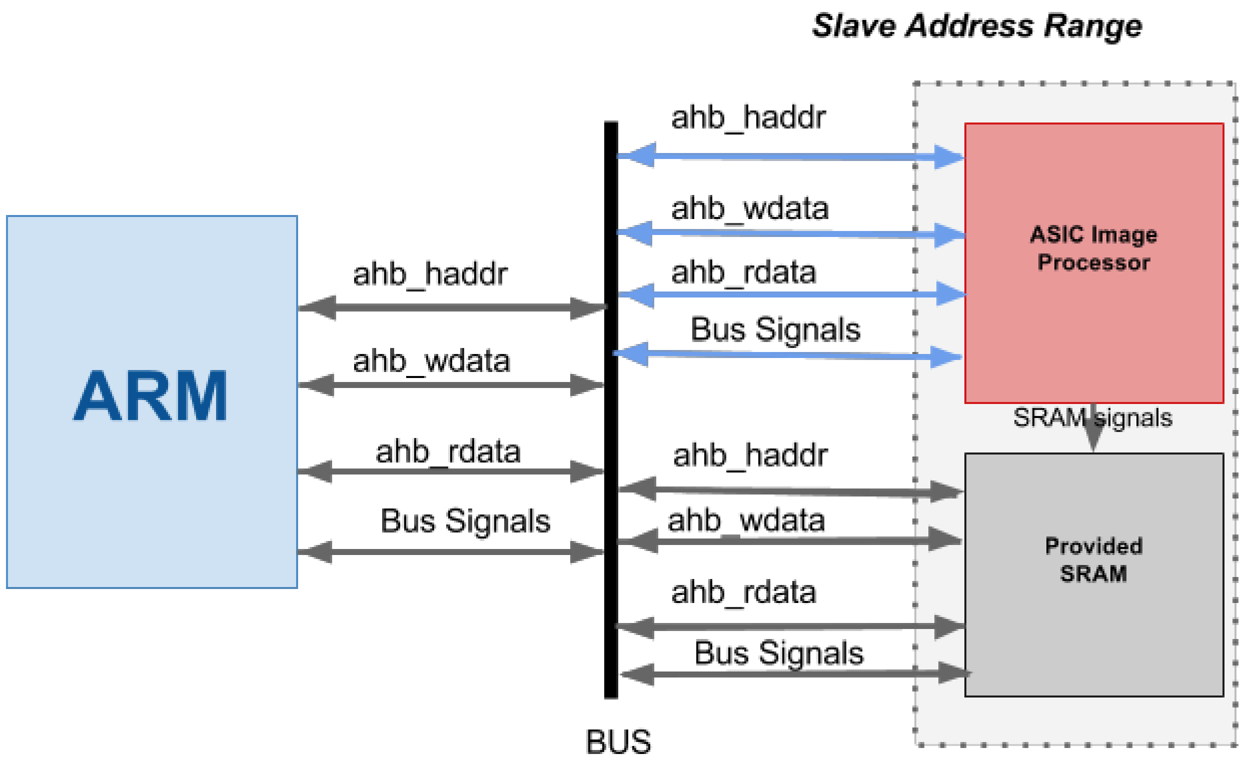


Fig 2.0.1 System Usage Diagram

2.2 Operational Characteristics

2.2.1 Introduction - Canny Edge Detection Algorithm

Canny edge detection consists of 4 major parts performed in sequence:

- Noise reduction (blur)
- Finding gradient magnitude and angle
- Non maximum suppression
- Hysteresis

These steps are explained more in depth below.

- **Noise Reduction**

Canny edge detector is susceptible to noise present in raw unprocessed image data. To overcome this Gaussian blur is applied by using convolution. For this project the mask for Gaussian blur was chosen so that the is fast and efficient.

1	4	8	4	1
4	16	32	16	4
8	32	64	32	8
4	16	32	16	4
1	4	8	4	1

Fig 2.2.1.1 Gaussian mask used

To further increase performance, this 2D gaussian mask is separated into 2 1D gaussian masks since it is separable.

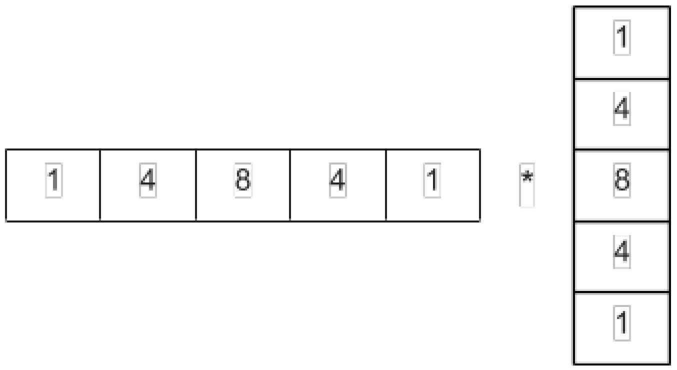
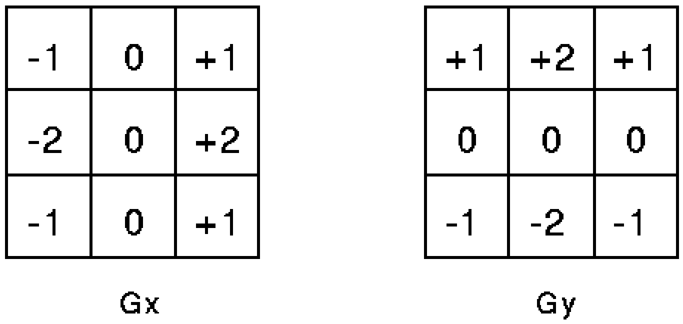


Fig 2.2.1.2 Seperation of Gaussian Mask

- **Finding Gradient Magnitude and Angle**

This is the most important step in canny edge detection. It involves finding the rate at which color intensity changes from pixel to pixel. To compute this first the gradients in the X and Y directions are computed separately. This is done by convoluting the image with sobel masks:



-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

Fig 2.2.1.3 Gradient Weighting
http://homepages.inf.ed.ac.uk/rbf/HIPR2/figs/sobmasks.gif

The magnitude is then approximated by adding their absolute values.

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

The gradient angle is computed by finding the range in which the arctan of Gy/Gx lies.

- **Non maximum suppression**

One of the features that sets canny edge detection apart from the others is its edge thinning technique. Non maximum suppression helps thin edges into one pixel wide edges resulting in clear prominent edges. This is performed by comparing the gradient magnitude of each pixel with the gradient magnitudes of the two pixels along the direction perpendicular to the direction of the edge.

- **Hysteresis**

Hysteresis is used to trace the edges in such a way that discontinuities in edges are kept to a minimum. This is achieved by using 2 thresholds as opposed to one to determine whether a pixel is an edge. If the pixel is adjacent to a pixel (in the direction of the edge angle) that has already been marked as an edge, then a low threshold is used to determine if it is an edge. Otherwise a higher threshold is used.

2.2.2 High level Sequence of steps

A pipelined architecture is used to perform canny edge detection as illustrated in the diagram below. The dotted line indicates the position of the image in the “canvas”. Pixels outside the boundaries of the image are simply copies of the pixel value at the corresponding edges. The result of each stage feeds into the next stage.

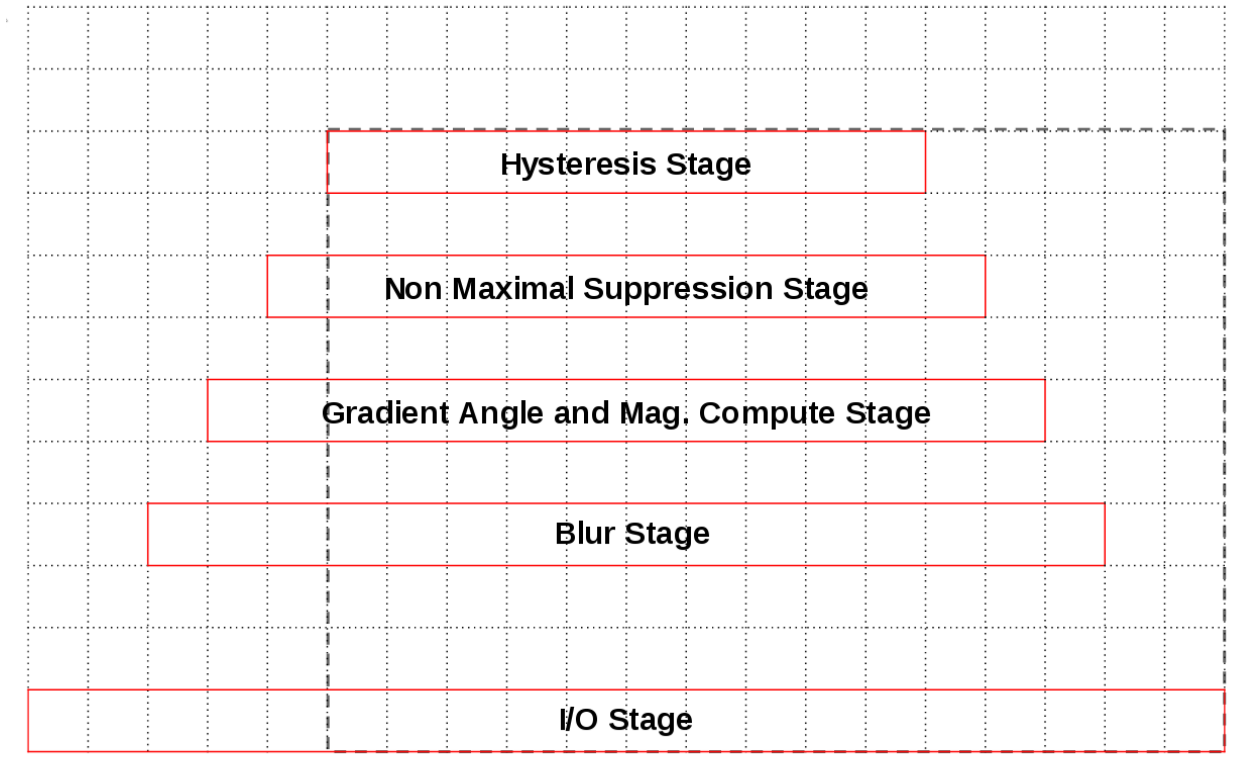

Hysteresis Stage
Non Maximal Suppression Stage
Gradient Angle and Mag. Compute Stage
Blur Stage
I/O Stage

Fig 2.2.2.1 Final Stage of Pipeline

The image is processed 10 horizontal pixels at a time. Since the algorithm requires the knowledge of “context” or knowledge of the pixels around it, 20 pixels are read in each phase. Since a blur mask of size 5 is used, it results in a loss of 2 pixels on either side. The blur phase caches the last 5 rows read to serve as context for the middle row (row being processed). The gradient stage uses a 3*3 mask and so loses only one pixel on either side. The non-maximum suppression stage also needs to compare each pixel with its adjacent pixels and so results in an output of 12 pixels. Similarly, Hysteresis the final stage relies on pixels adjacent to it to determine whether a pixel should be marked as an edge. This gives us 10 pixels of the final processed image.

2.2.3 Ports of Major Blocks

Block Name: Initializer

- **Description:** The purpose of the initializer is to initiate start edge detection request from the processor over the AHB bus. It assigns the Image Processing chip a range of address that the processor may send data to. To initiate a processing request, the processor must send information about the image for example: where it is located on SRAM, the width and the height. Once all the necessary information is received, the initializer activates the controller to kickstart the image processing pipeline.

- **Port Description:**

Signal	Direction	**Description**
n_rst	Input	Asynchronous Negative Edge Reset
ahb_hclk	Input	Bus Clock
ahb_htrans	Input	Transfer Type
ahb_hburst	Input	Toggle Burst mode on and off
ahb_hwrite	Input	Transfer Direction
ahb_hprot	Input	Protection Control
ahb_haddr	Input	Address Bus
ahb_hwdata	Input	Write Data Bus
ahb_hrdata	Input	Read Data Bus
ahb_hgrant	Input	Bus Grant
ahb_hlock	Input	Locks Transfer Request
ahb_hbusreq	Input	Bus Requests
ahb_hready	Output	Slave is ready

ahb_hresp	Output	Transfer Response
width	Output	Width of Image Size
height	Output	Height of Image Size
readStartAddress	Output	Start address of image to read
writeStartAddress	Output	Start address of where to write image
filterType	Output	What type of filter is needed
final_enable	Output	Enable’s other blocks to start processing data

Block Name: pixelcontroller

- **Description:** The purpose of the pixel controller is to manage the pixel input output as part of the image processing pipeline. As the name implies, the pixel controller has an SRAM interface and fetches or writes required number of pixels to and from SRAM continuously. In addition, since the top level design of the Image Processing chip should support color images but the pipeline only work with grayscale images, this is the point where each pixels are combinationally stepped down to grayscale. The pixel controllers take *number of pixel* wanted to read or write as input. It also takes *the offset pixel to begin reading or writing from* as input. Pixel Controller will throw out a return a series of requested pixels onto a 20 bytes wide output port which can be accessed by blocks that wish to access those pixels. Once the read operation is over, it performs a write operation. The requesting block can specify the data to be written in a 20 bytes wide input port. Each bytes correspond to a single grayscale pixel.

- **Port Description:**

Signal	**Direction**	**Description**
n_rst	Input	Asynchronous Negative Edge Reset
clk	Input	System Clock
enable	Input	Enable to start block
data_in	Input	Requested pixels to be written to SRAM
address_write_offset	Input	Starting address to write to
address_read_offset	Input	Starting address to read form
num_pix_read	Input	How many pixels to read
num_pix_write	Input	How many pixels to write
data_out	Output	Requested pixels form SRAM

read_now	Output	Flag pixel data as must be read now
end_of_operations	Output	Finish signal
address	Output	
w_data	Output	
r_data	Input	
read_enable	Output	Enable read function
write_enable	Output	Enable write function

Block Name: blur_controller

- **Description:**
  This module applies the gaussian blur mask described earlier. It tries uses 4 blur filters. First blurring in the X direction is performed, the result is added to the cache of previously X blurred rows and then Y blur is performed.

- **Port Description:**

Signal	Direction	Description
clk	Input	System Clock
n_rst	Input	Asyncronous Negative Edge Reset
anchor_moving	Input	Enabled when ancor has moved
anchor_y	Input	Vertical position of anchor
blur_in	Output	Input pixels to be blurred
blur_out	Output	Blurred pixels
blur_final	Output	Final clock cycle for blur block

Block Name: gradient_controller

- **Description:**
  This module computes the X and Y gradients using Sobel masks and then uses the results to compute the gradient magnitude and angle.

- **Port Description:**

Signal	Direction	Description
clk	Input	System Clock
n_rst	Input	Asynchronous Negative Edge Reset
anchor_moving	Input	Tells when the anchor moves to start filtering
anchor_y	Input	Vertical position of anchor
gradient_in	Input	Output from Blur Module
gradient_angle	Output	Gradient angle output
gradient_mag	Output	Gradient magnitude output
gradient_x	Output	Gradient across the x axis
gradient_y	Output	Gradient across the y axis
gradient_final	Output	Enabled on last clock cycle to tell that block is done

Block Name: nms_controller

- **Description:**
  This module performs Non Maximum Suppression. It also caches the gradient angles and finds the corresponding gradient angle for the current output.

- **Port Description:**

Signals	Direction	Description
clk	Input	System Clock
n_rst	Input	Asynchronous Negative Edge Reset
anchor_moving	Input	Tells when anchor moves to start filtering
gradient_angle	Input	Output angle from gradient block
gradient_mag	Input	Output magnitude from gradient block
nms_angle_out	Output	Non-Maximal Suppression angle output
nms_out	Output	Non-Maximal Suppression pixel output
nms_final	Output	Enabled on last clock cycle to tell that block is done

Block Name: hyst_controller

- **Description:**
  This module performs hysteresis by caching the previous outputs of this module. As mentioned earlier if an adjacent pixel along the gradient has been marked as an edge a lower threshold is used.

- **Port Description:**

Signals	Direction	Description
clk	Input	System Clock
n_rst	Input	Asynchronous Negative Edge Reset
gradient_angle	Input	The angle of the gradient
hyst_in	Input	Output from Non-Maximal Suppression
hyst_out	Output	Output from hysteresis
hyst_final	Output	Enabled on last clock cycle to tell that block is done

2.3 Requirements

The main intended application of our ASIC Image Processor is primarily to provide a fast real-time image processing capability to optical devices, which have many industrial and commercial applications such as computer vision, manufacturing, digital or surveillance cameras and more. Hence, speed is the number one priority in our design. Secondary priority would be power consumption. Since the chip has a possibility of being utilized in factory condition, it must be able to withstand high temperature swing.

Area target: 1.5mm x 1.5mm
Pin count: 14
Max throughput: 230,400 bytes per second

Since the application of this chip in commercial and industrial environment may require handling of realistic images, we estimate the input and output images to be 8 bit deep (number of bits used to represent color in a single pixel), containing 28 image (1280x720), the image will contain 921,600 pixels and it would need 7,372,800 bits (921,600 bytes) the represent the entire image. Say we operate on 400 MHz clock and we transfer 1 bit/cycle (serial). We should be able to process 1/4 portion of the image every second. So approximately, a 720p bitmap image of depth 8 will be able to be processed at around 4 to 5 seconds.

3.1 Design Architecture

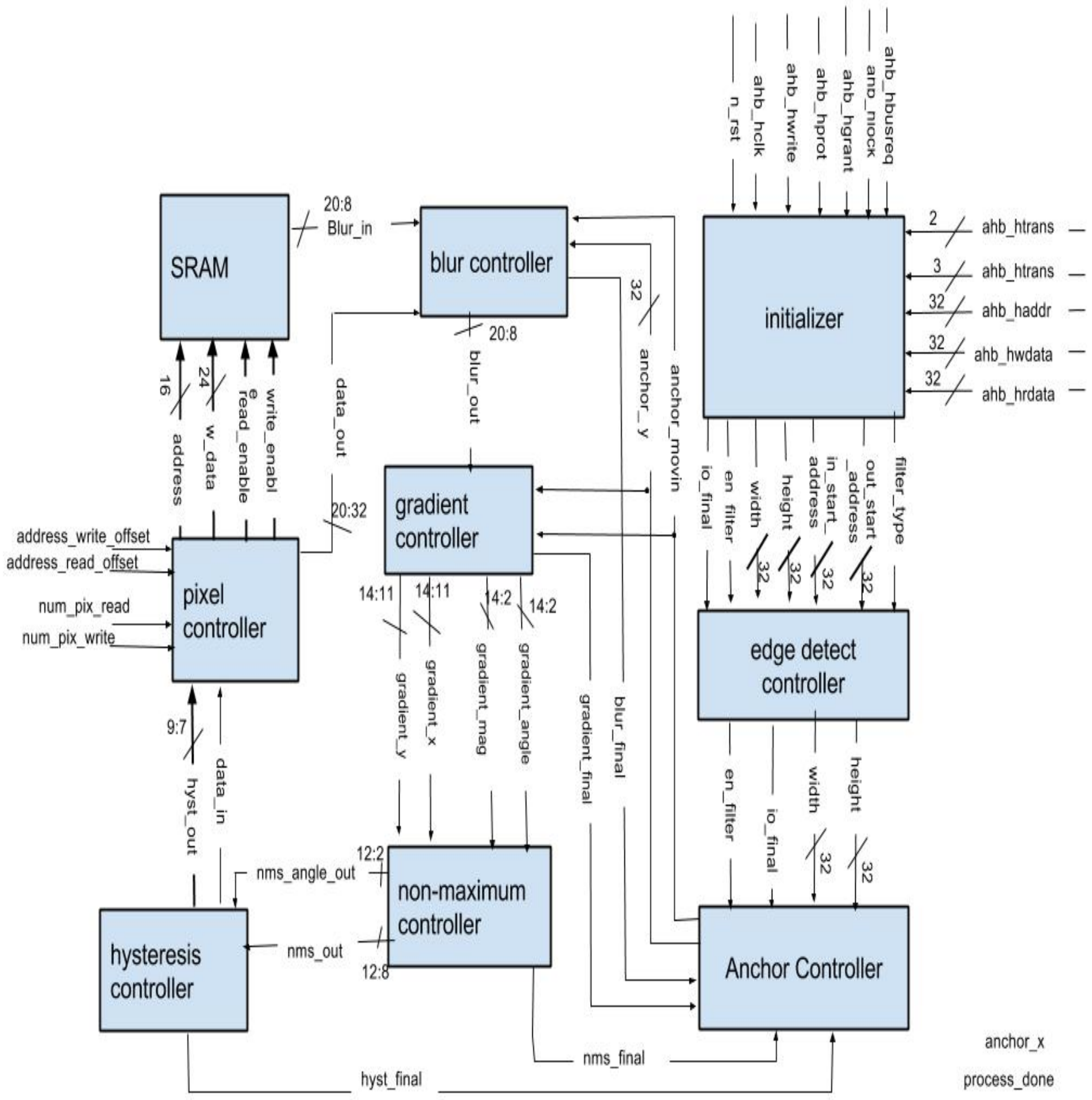
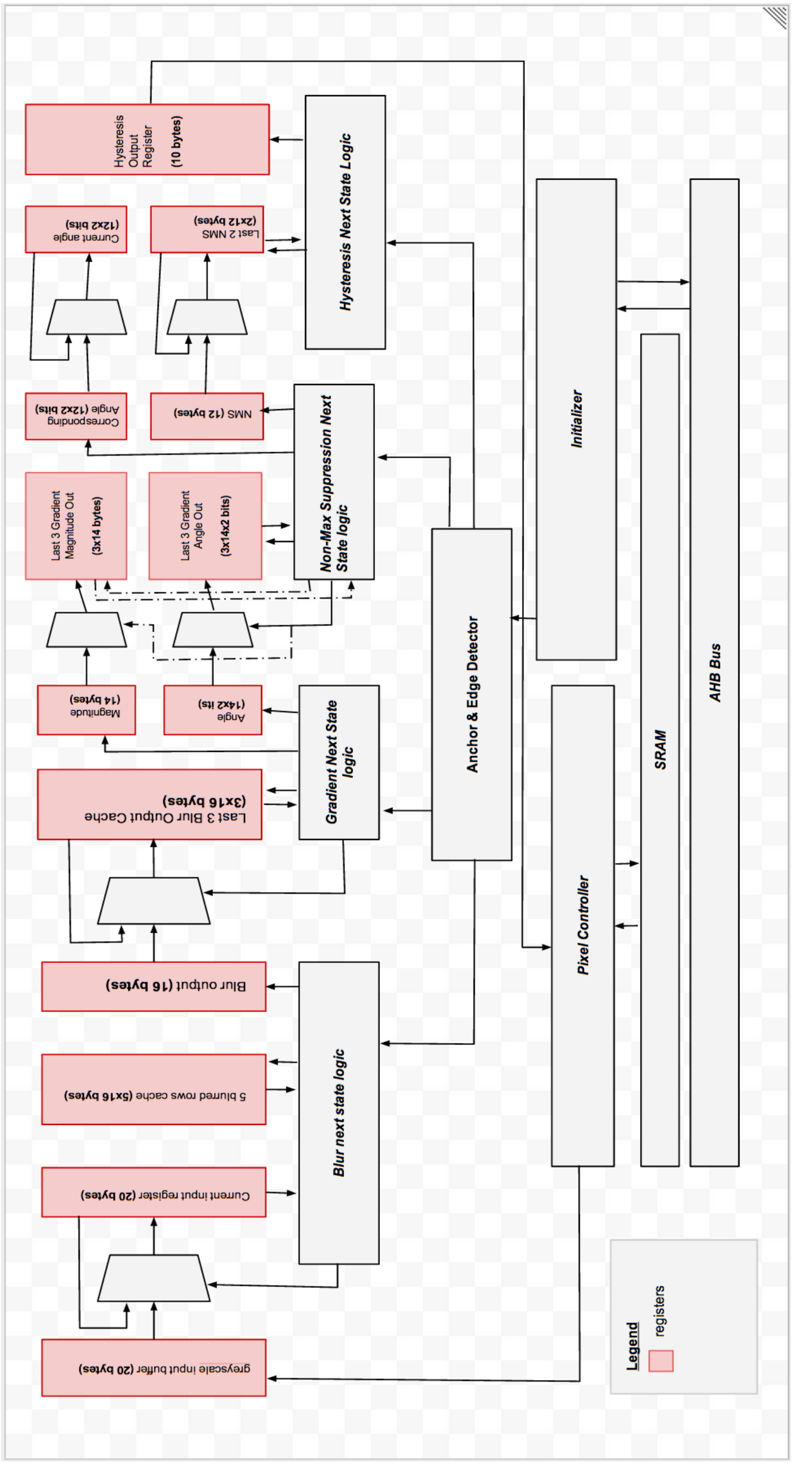


Fig 3.1.0 Design Architecture

3.2 Functional Block Diagrams

3.2.0 Top Level Block Diagram



Hysteresis Output Register (10 bytes)
Current angle (12x2 bits)
Last 2 NMS (2x12 bytes)
Hysteresis Next State Logic
Corresponding Angle (12x2 bits)
NMS (12 bytes)
Non-Max Suppression Next State logic
Last 3 Gradient Magnitude Out (3x14 bytes)
Last 3 Gradient Angle Out (3x14x2 bits)
Magnitude (14 bytes)
Angle (14x2 its)
Gradient Next State logic
Last 3 Blur Output Cache (3x16 bytes)
Anchor & Edge Detector
Initializer
SRAM
AHB Bus
Pixel Controller
Blur output (16 bytes)
5 blurred rows cache (5x16 bytes)
Blur next state logic
Current input register (20 bytes)
greyscale input buffer (20 bytes)
Legend
registers

Fig 3.2.0 Top Level Block Diagram

3.2.1 Block Diagram for Blur Controller Module

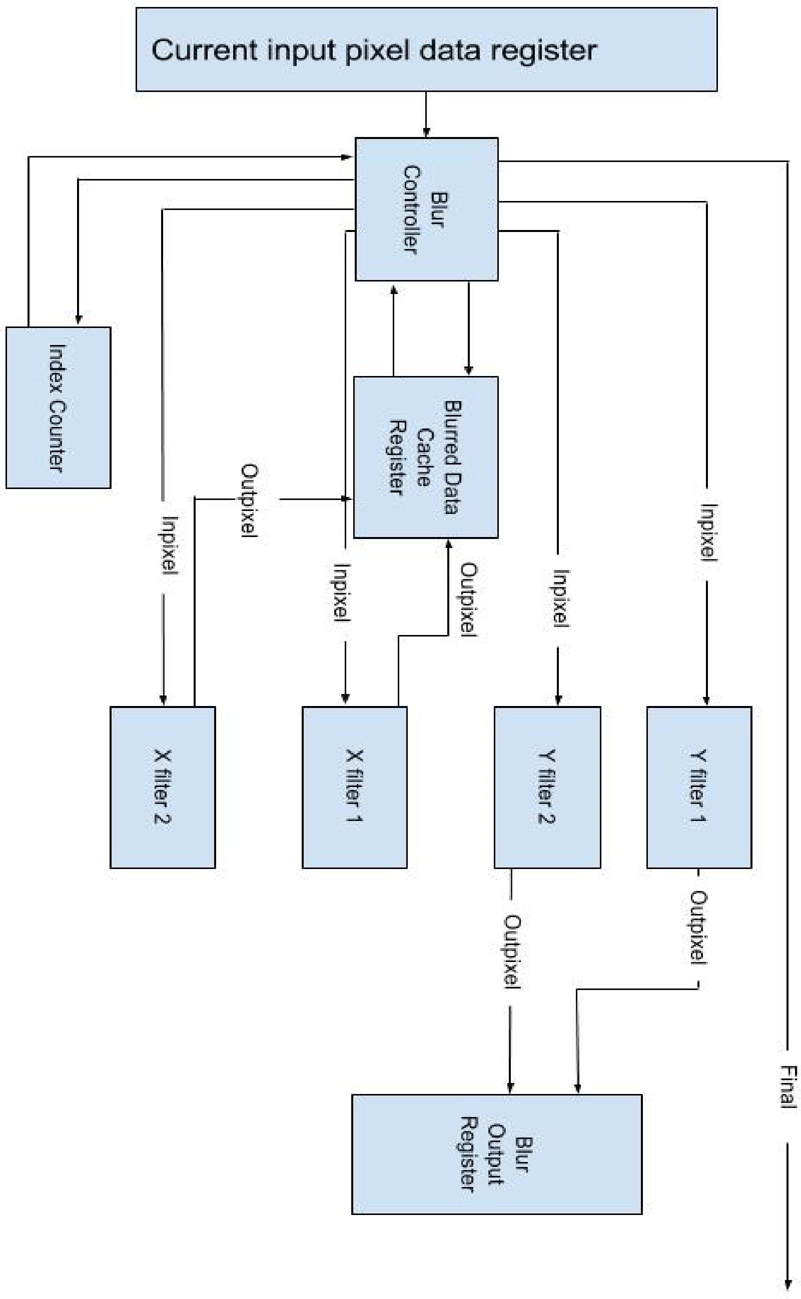


Figure 3.2.1 Blur Controller Block Diagram

3.2.2 Block Diagram for Gradient Controller Module

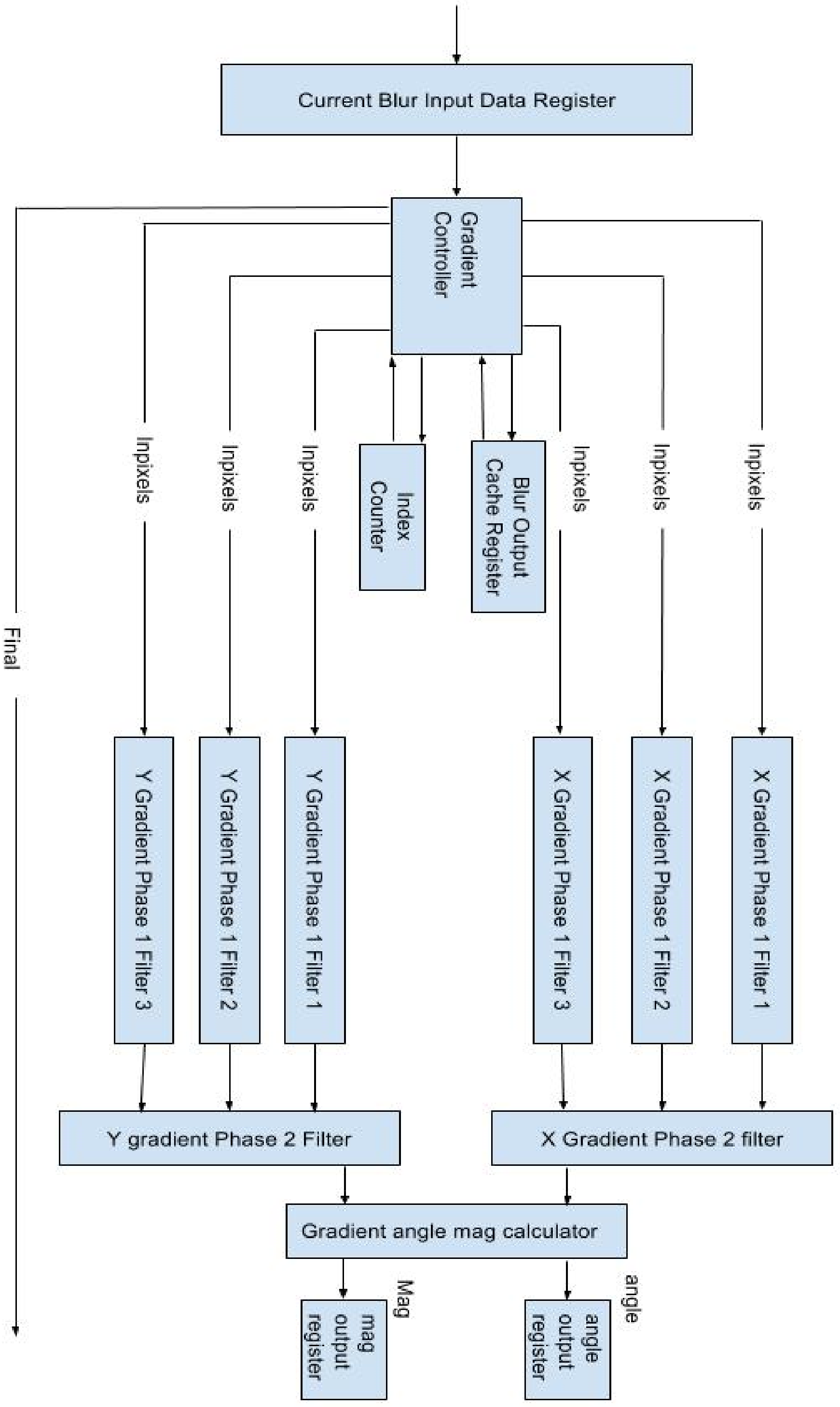

Current Blur Input Data Register
Gradient Controller
Index Counter
Blur Output Cache Register
Inpixels
Inpixels
Inpixels
Inpixels
Inpixels
Inpixels
Final
Y Gradient Phase 1 Filter 3
Y Gradient Phase 1 Filter 2
Y Gradient Phase 1 Filter 1
X Gradient Phase 1 Filter 3
X Gradient Phase 1 Filter 2
X Gradient Phase 1 Filter 1
Y gradient Phase 2 Filter
X Gradient Phase 2 filter
Gradient angle mag calculator
Mag
angle
mag output register
angle output register

Fig 3.2.2 Gradient Controller Block Diagram

3.2.3 Block Diagram for Non-Maximum Suppression (NMS) Controller Module

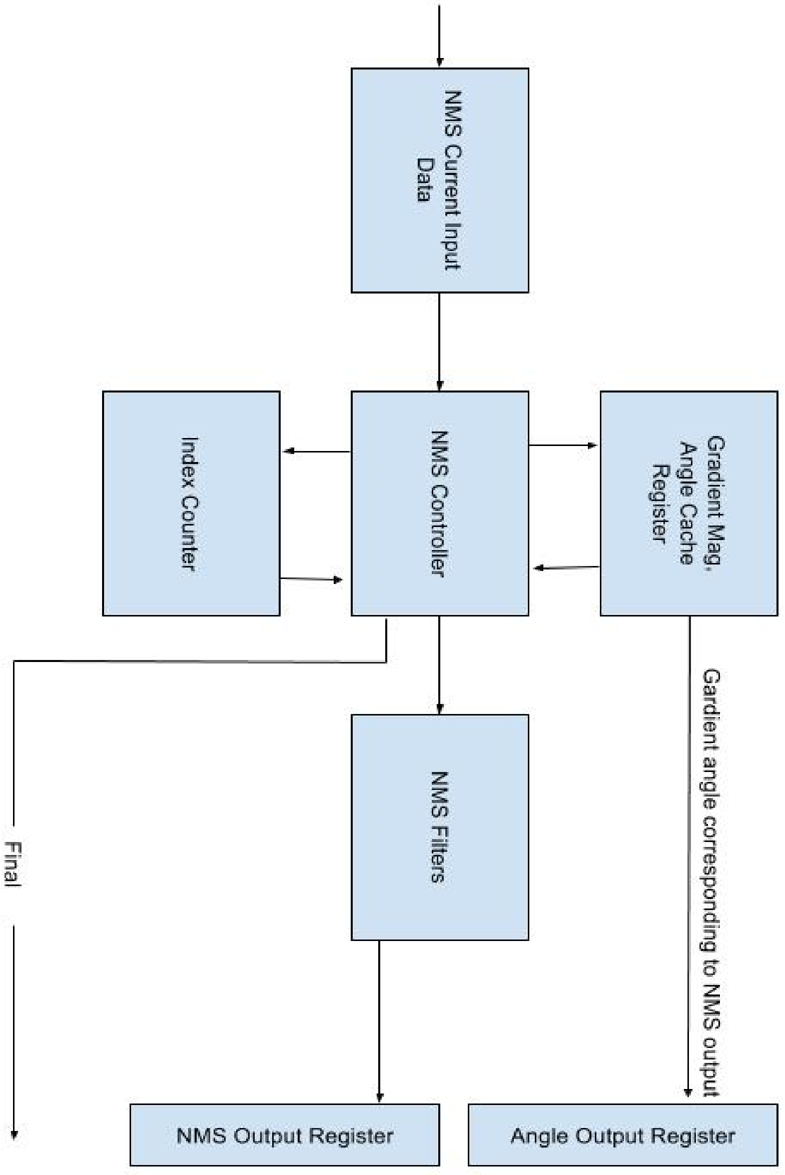


Fig 3.2.3 Non-Maximal Suppression Block Diagram

3.2.4 Block Diagram for Hysteresis Controller Module

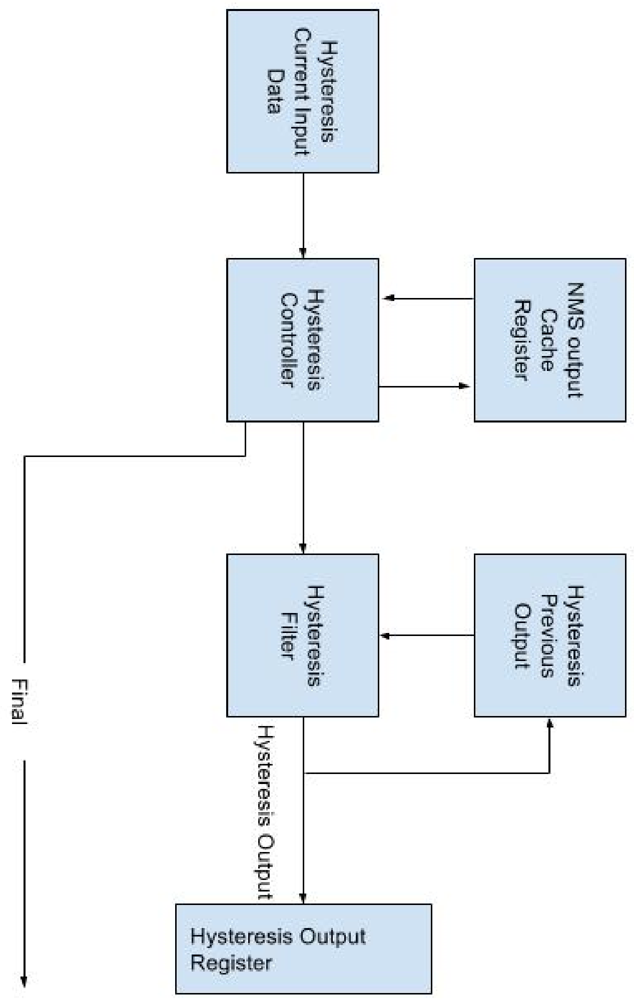

Hysteresis Current Input Data
Hysteresis Controller
NMS output Cache Register
Hysteresis Filter
Hysteresis Previous Output
Final
Hysteresis Output
Hysteresis Output Register

Fig 3.2.4 Hysteresis Controller Block Diagram

3.2.5 Block Diagram for Blur Module

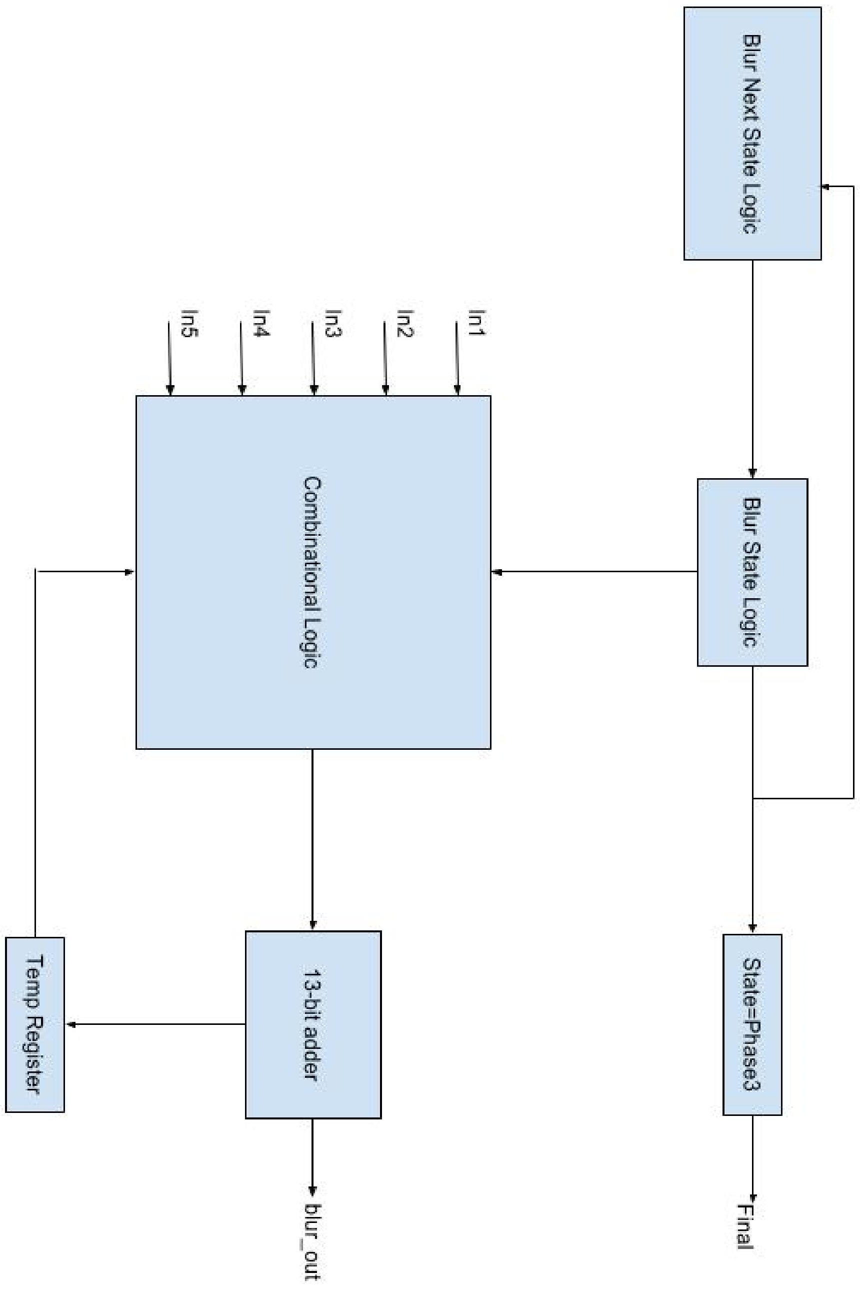


Fig 3.2.5 Blur Module Block Diagram

3.2.6 Block Diagram for Gradient Sub Module

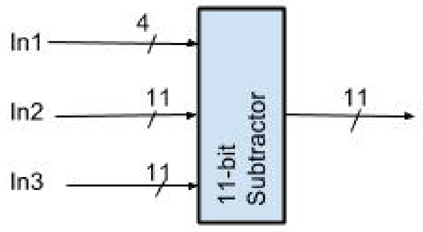


Fig 3.2.6 Gradient Sub Block Diagram

3.2.7 Block Diagram for Gradient Weight Module

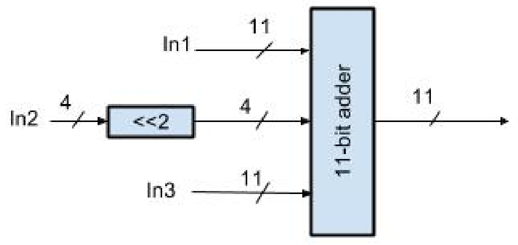


Fig 3.2.7 Gradient Weight Block Diagram

3.2.8 Block Diagram for Non-Maximum Suppression (NMS) Module

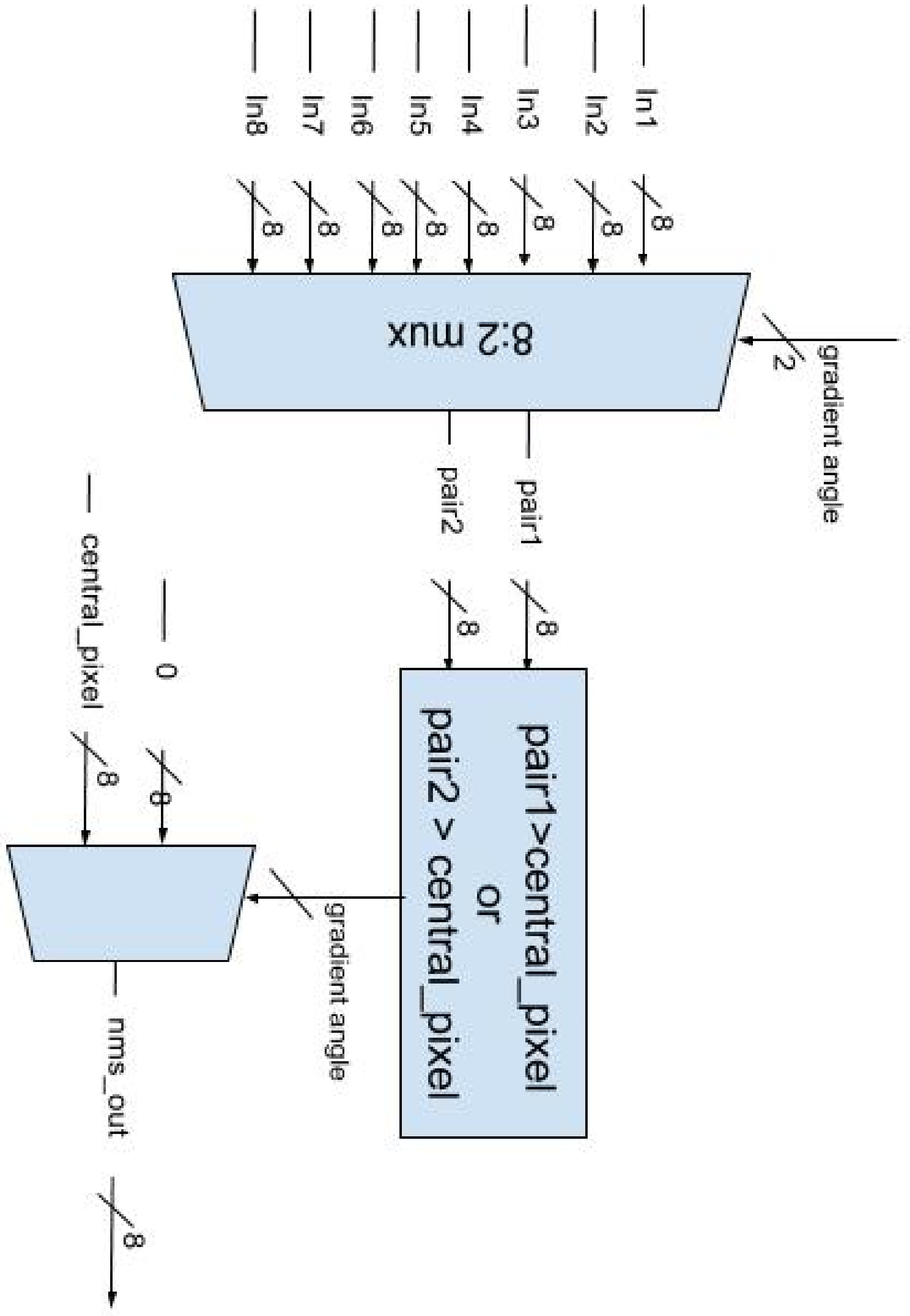


Fig 3.2.8 Non-Maximal Suppression Block Diagram

3.2.9 Block Diagram for Hysteresis Module

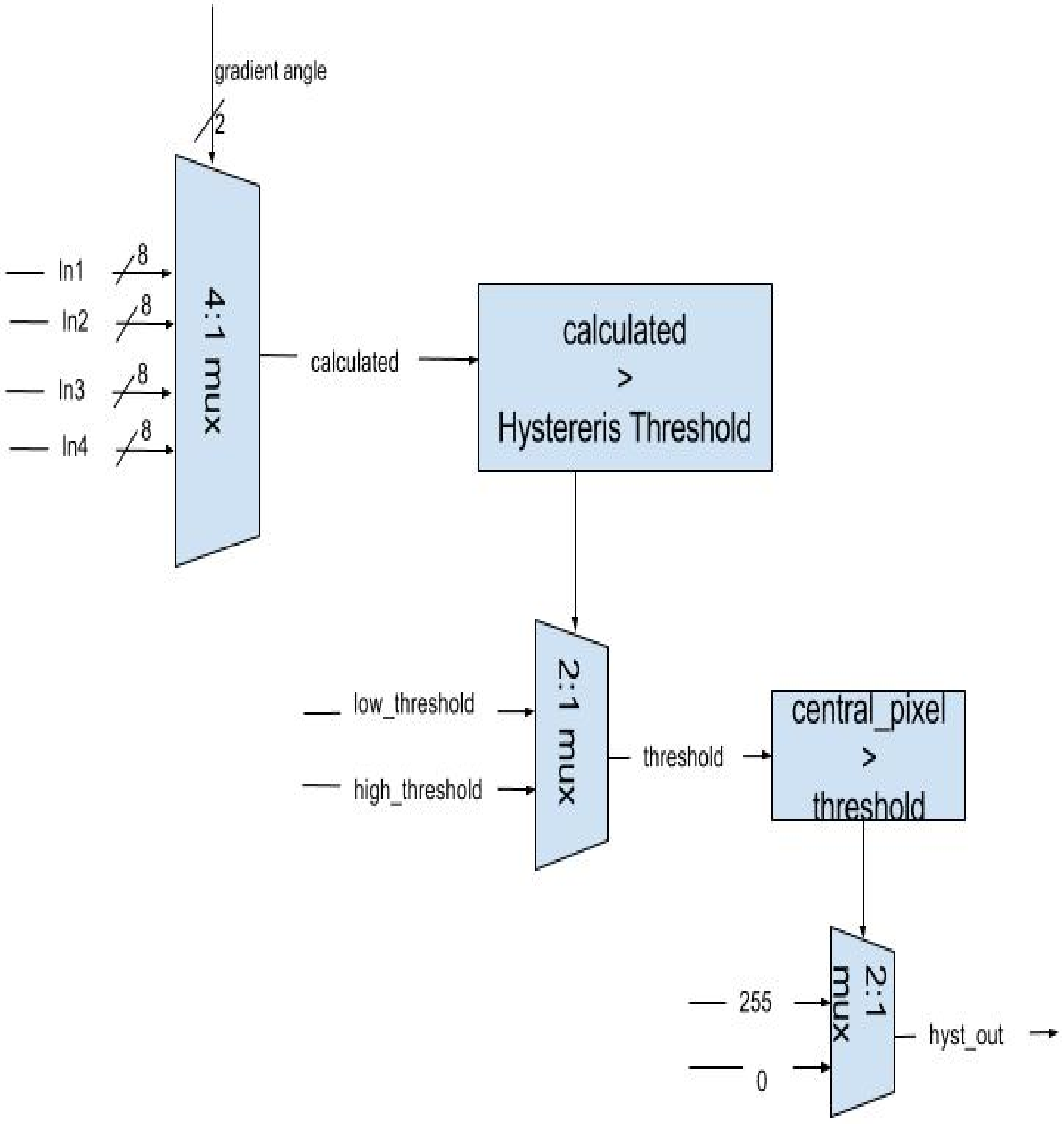


Fig 3.2.9 Hysteresis Block Diagram

3.3 Standards and Protocols

3.3.1 AMBA AHB Bus Protocol: AHB bus protocol was needed for our design to work with the mobile ARM Processor. AHB stands for Advanced High-Performance Bus. This protocol adds many features to the AMBA bus protocols already available. These additions make transferring data quicker and more seamless. The AHB protocol delivers an address and then data, but while the previous data is being transferred the next address is already on the bus. This allows the process to take half the time needed with an earlier AMBA bus protocol.

3.3.2 On-Chip SRAM: The On-chip SRAM allows for faster access to much more data. With the SRAM being on chip a custom bus interface was able to be implemented to allow the transfer of data to and from the design without tying up the AHB bus. However, due to time constraint we were not able to implement the “memory-map everything” aspect of the AHB paradigm.

3.4 Timing and Area Budget

Critical Path Constraint (Design Budget)	Area Constraint (Design Budget) (ns)	Critical Path Constraint (Final Synthesis) **(**mm^2)	Area Constraint (Final Synthesis) **(**mm^2)	Critical Path Arrival Time (Encounter) (ns)	Area (Encounter) **(**mm^2)
11.5 ns	38.71	4.91	8.715	6.689	14.39 mm^2

Since some of the blocks were not connected, but all the majors block were connected, we expect the following results for summation of the following chip areas to be an acceptable estimation of the entire chip area.

Block name	**Synthesis Area** (mm^2)	**Encounter Block Area (**mm^2)	**Critical Path Delay (**synthesis)
Initializer	0.0327	0.21911	1.73
Pixel Controller	0.3835	1.38321	3.46
Edge Detection and its sub-blocks	8.2988	12.7951	4.91

4. Verification

4.1 Initializer

The purpose of the initializer is to initiate start edge detection request from the processor over the AHB bus. The initializer also ensure that all necessary information is provided before initiating the image processing pipeline.

To verify that this is correct, we construct a testbench that emulate the ARM processor. Data is sent over the AHB address and data line, and appropriate bus signals such as HREADY and HRESP are triggered.

Finally, once all information about the image is received the initializer should be able to notify the anchor controller that the processing may begin, this is done via a single bit enable signal so testing it is trivial.

In addition, the initializer must be able to detect invalid data or irrelevant data sent form the ARM processor as part of the image's metadata. To test this, we look at the state transition of the Initializer when these unexpected events occur. *(See Appendix B)*

4.2 Pixel Controller

To verify that this blocks work as expected we construct a testbench that loads images into SRAM and does continuous read operation and capture the output from this block and rewrite them into an image. If this is correct, we should get a grayscale copy of the original image initially dumped in SRAM *(See Waveform Appendix B 1.1).* Input-output images are tabulated in *Table figure Appendix B* 2.1. As shown and discussed in Appendix B 2.1, the Pixel Controller produces grayscale image that is imperfect by misplacing last few columns of the original image. This issue was not resolved at time of writing this report.

To verify that this blocks can read and write continuously, we write a testbench to simulate a read operation and write operation and observe the SRAM dump before and after *(See Waveform Appendix B 1.1.2).* If N pixels is requested, N pixels must be fetched accordingly. This has been achieved in the result.

Finally, to verify that the calculation is sound, we use a python script to calculate the sequence of all grayscale pixels and compare with the one calculated by hardware.

4.3 Gaussian Blur

Verification of the gaussian blur module was done by generating a random stream of bytes to simulate an image stream. This was done to be sure that the module can handle all possible outputs and so that the tests covers as many possible scenarios as possible. The expected results were calculated by computing the gaussian blur in the Verilog test bench. The 2D convolution of the random image stream with the selected gaussian kernel was then compared with the results of the block. Since a maximum error of -9 was expected due to 2 passes through 1D convolution in which integer division was being performed, it was checked whether each resulting pixel was within the range of expected_value - 9 to expected_value. All the test cases passed for both the source and mapped versions of the module.

4.4 Gradient

The gradient module was verified in a manner similar to the blur module. A random stream of bytes was generated and the expected X and Y gradients were computed along with the expected total gradient magnitude. Gradient angle was not tested intensively since the X and Y gradients were being tested individually and the combinational logic computing angle was small.

The expected values *(see Section 2.2.1 for calculation detail)* were then compared with the computed X, Y gradients and the magnitudes.

All the test cases passed for both the source and mapped versions of the module.

4.5 Non-maximum suppression

Non maximum suppression was tested by running it on a specially crafted input data. The data was essentially a transition from zero gradient to max gradient and back to zero gradient in the horizontal direction. The expected result *(see Section 2.2.1 for calculation detail)* was that the module would just select the peak in the middle and zero out the rest of the image which would be an indication of correct functioning of the block. The test case passed for both source and mapped versions of the module.

4.6 Hysteresis

Hysteresis was tested using a special input image very similar to the input image used for testing non maximum suppression. The difference was that this time we expected not a single row getting selected but multiple rows as long as their gradients are above the minimum threshold.

In the test case that was used two rows were selected as expected.

4.7 Edge Detection

Edge detection block is the top level module for the edge detection component. Correct working of this module involves both Blur, Gradient, Non maximum suppression and hysteresis to work together as well as Anchor controller to work correctly. Consequently the testing of this module was done in 2 phases. First a very simple test case was created modeling an image sample. This was simply a white filled rectangle in a white background. The expected result was a hollow white rectangle in a black background. Both the source and mapped versions of the code passed this test with a success rate of more than 98% (98% of pixels correctly computed).
The second phase was to read real bmp images from the disk and feed it into the edge detection block and visually verify that the output was correct. We did this for about 6 images and the results were as expected for source.

The mapped version took a long time to complete so we focused on the source results since all the individual blocks were working correctly for both source and mapped versions.

5. Layout

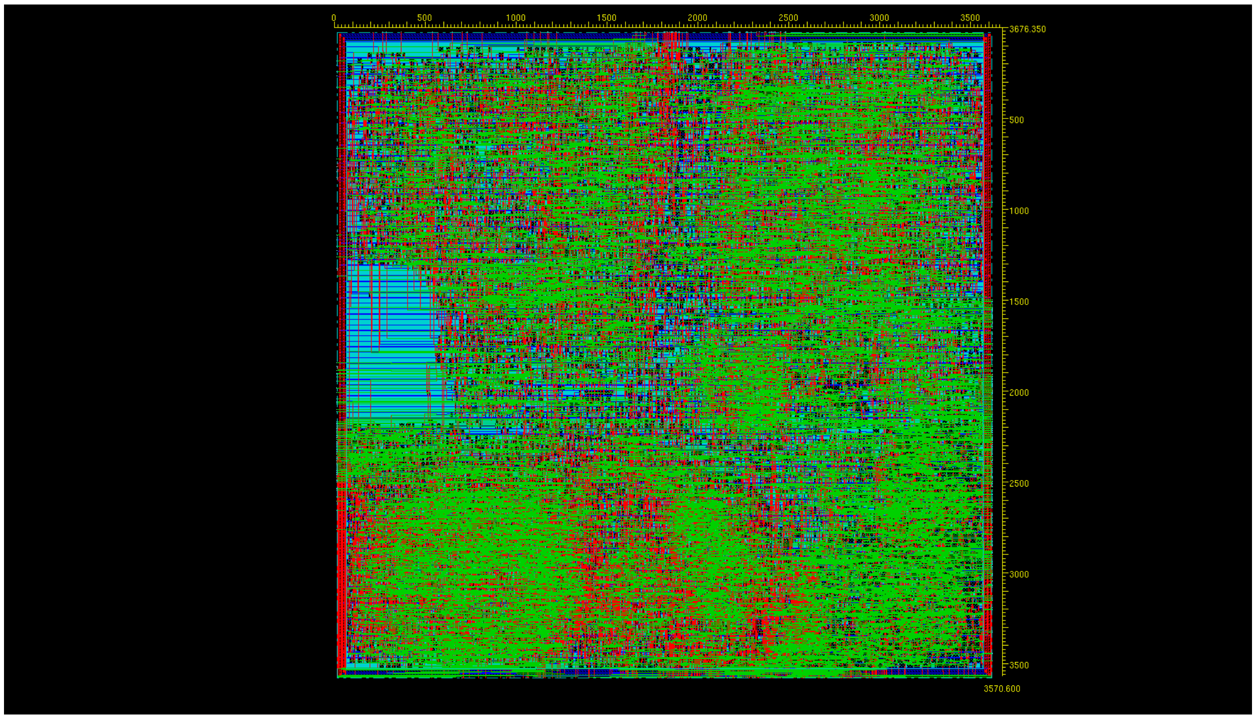
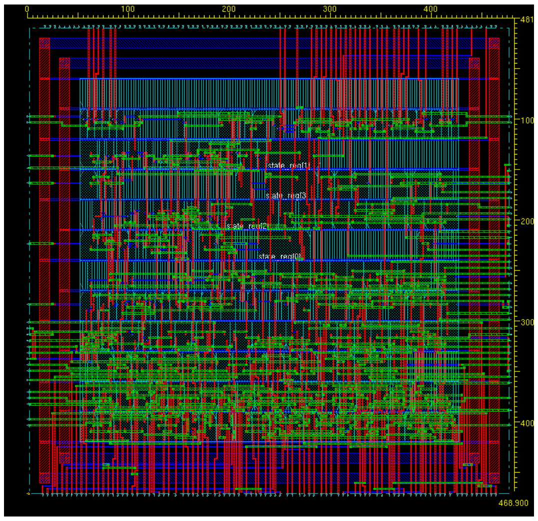


Fig 5.1 Layout for Edge Detection Block



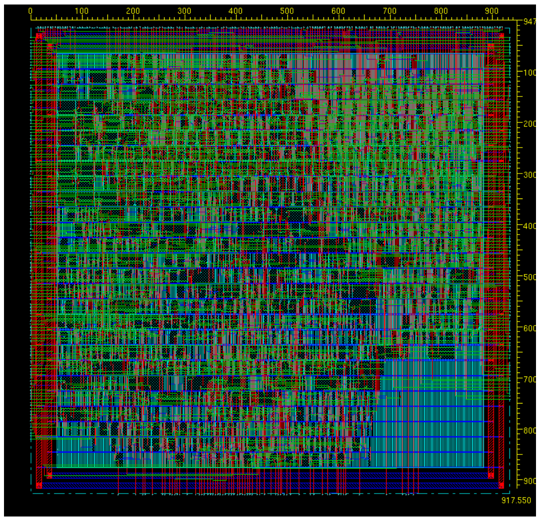


Fig 5.2 Layout for Initializer

Fig 5.3 Layout for Pixel Controller

6. Results

Success criteria that were fulfilled are as follow.

	Status	**Comments**
1. Respond to processor's image processing request.	Yes	Initializer can receive information about image and receive start request from processor.
2. Process input images according to settings.	No	Not applicable since not all blocks were connected.
3. Gaussian blur is performed correctly.	Yes	Gaussian blur output is as expected.
4. Edge detection is performed correctly.	Yes	Edge detection produces image output that appears as expected by visual verification.
5. Respond to relevant AHB bus signals.	Partially	Can communicate over the bus with the master, and can respond with appropriate response but is not memory mapped and may not respond to all combinations of bus signals.

7. Appendix A - Data Sheets

Account and directory where all of the files are located: mg81/ece337/Project

Full Project Repository: **https://github.com/sidharthms/asic-edge-detector**

Top level structural VERILOG code : N/A

Initializer Block	source/initializer.sv
Gaussian Blur Block	source/blur.sv
Gradient Block	source/gradient_controller.sv
Gradient Mag Angle	source/gradient_mag_angle.sv
Gradient Sub Block	source/gradient_sub.sv
Gradient Weight	source/gradient_weight.sv
Hysteresis Block	source/hyst_contoller.sv
Hysteresis Calculations	source/hyst.sv
Non-Maximal Suppression	source/nms_controller.sv
NMS Calculations	source/nms.sv

Test Benches:

blur controller test	source/tb_blur_controller.sv
edge detect controller test	source/tb_edge_detect.sv
gradient controller test	source/tb_gradient_controller.sv
hysteresis controller test	source/tb_hyst_controller.sv
initializer block test	source/tb_initializer.sv
non-maximum suppression controller test	source/tb_nms_controller.sv

Misc Scripts:

image unpacking	source/*generate*
image unpacking	source/generate.py
image unpack and repack	source/image_str_converter.py
memory dump to byte stream	mem2img.sh
testbench image reconstruction	reconstructX.sh

Report Files:

final project report	docs/Report.pdf
preliminary presentation	docs/PreliminaryPresentation
final presentation	docs/FinalPresentation

Table 2-1 AMBA AHB signals

Name	Source	Description
HCLK Bus clock	Clock source	This clock times all bus transfers. All signal timings are related to the rising edge of **HCLK.**
HRESETn Reset	Reset controller	The bus reset signal is active LOW and is used to reset the system and the bus. This is the only active LOW signal.
HADDR[31:0] Address bus	Master	The 32-bit system address bus.
HTRANS[1:0] Transfer type	Master	Indicates the type of the current transfer, which can be NONSEQUENTIAL, SEQUENTIAL, IDLE or BUSY.
HWRITE Transfer direction	Master	When HIGH this signal indicates a write transfer and when LOW a read transfer.
HSIZE[2:0] Transfer size	Master	Indicates the size of the transfer, which is typically byte (8-bit), halfword (16-bit) or word (32-bit). The protocol allows for larger transfer sizes up to a maximum of 1024 bits.
HBURST[2:0] Burst type	Master	Indicates if the transfer forms part of a burst. Four, eight and sixteen beat bursts are supported and the burst may be either incrementing or wrapping.
HPROT[3:0] Protection control	Master	The protection control signals provide additional information about a bus access and are primarily intended for use by any module that wishes to implement some level of protection. The signals indicate if the transfer is an opcode fetch or data access, as well as if the transfer is a privileged mode access or user mode access. For bus masters with a memory management unit these signals also indicate whether the current access is cacheable or bufferable.

Table 2-1 AMBA AHB signals (continued)

Name	Source	Description
HWDATA[31:0] Write data bus	Master	The write data bus is used to transfer data from the master to the bus slaves during write operations. A minimum data bus width of 32 bits is recommended. However, this may easily be extended to allow for higher bandwidth operation.
HSELx Slave select	Decoder	Each AHB slave has its own slave select signal and this signal indicates that the current transfer is intended for the selected slave. This signal is simply a combinatorial decode of the address bus.
HRDATA[31:0] Read data bus	Slave	The read data bus is used to transfer data from bus slaves to the bus master during read operations. A minimum data bus width of 32 bits is recommended. However, this may easily be extended to allow for higher bandwidth operation.
HREADY Transfer done	Slave	When HIGH the **HREADY** signal indicates that a transfer has finished on the bus. This signal may be driven LOW to extend a transfer. Note: Slaves on the bus require **HREADY** as both an input and an output signal.
HRESP[1:0] Transfer response	Slave	The transfer response provides additional information on the status of a transfer. Four different responses are provided, OKAY, ERROR, RETRY and SPLIT.

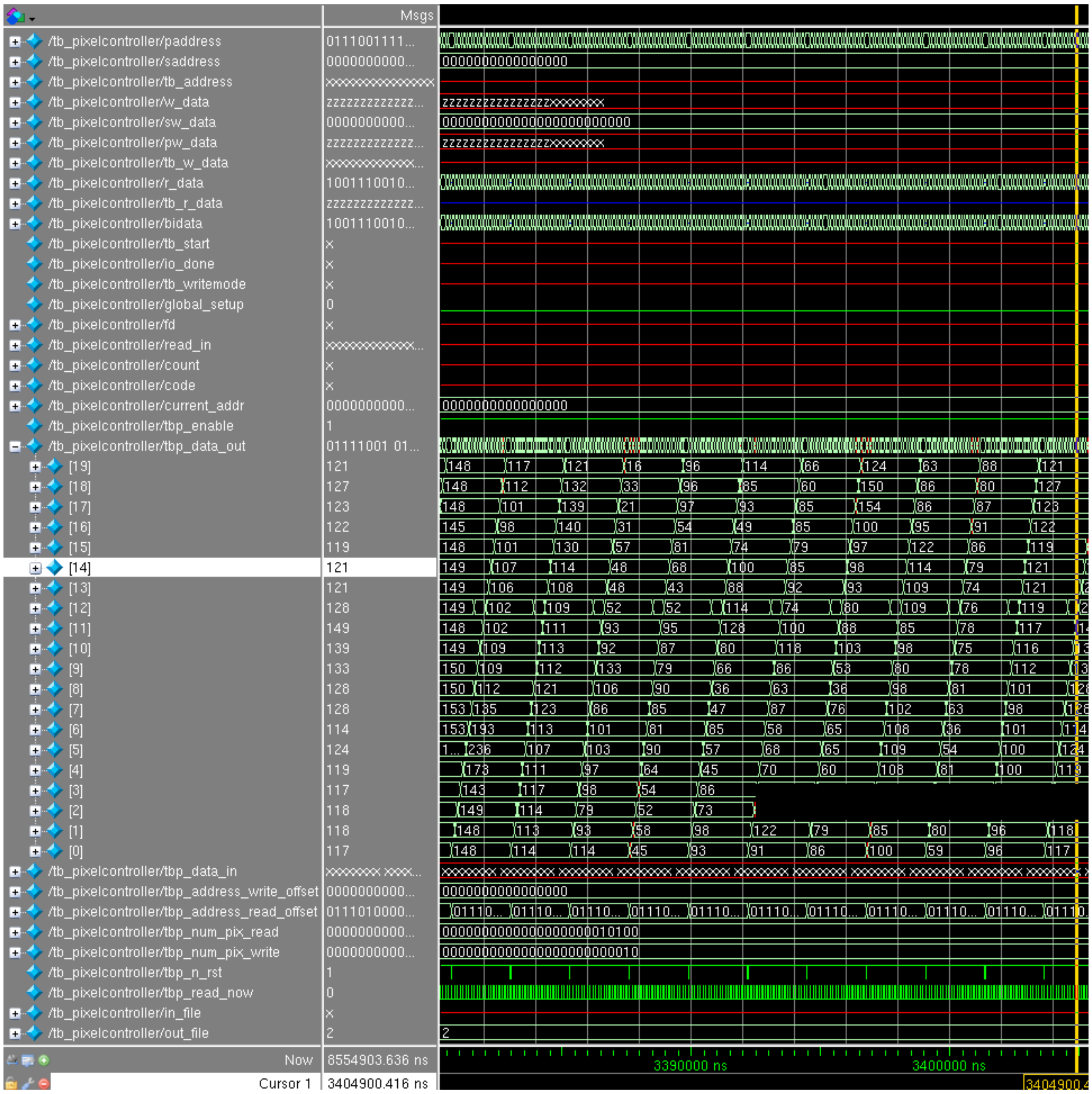
Table 3-5 Response encoding

HRESP[1]	**HRESP[0]**	**Response**	**Description**
0	0	OKAY	When **HREADY** is HIGH this shows the transfer has completed successfully. The OKAY response is also used for any additional cycles that are inserted, with **HREADY** LOW, prior to giving one of the three other responses.
0	1	ERROR	This response shows an error has occurred. The error condition should be signalled to the bus master so that it is aware the transfer has been unsuccessful. A two-cycle response is required for an error condition.
1	0	RETRY	The RETRY response shows the transfer has not yet completed, so the bus master should retry the transfer. The master should continue to retry the transfer until it completes. A two-cycle RETRY response is required.
1	1	SPLIT	The transfer has not yet completed successfully. The bus master must retry the transfer when it is next granted access to the bus. The slave will request access to the bus on behalf of the master when the transfer can complete. A two-cycle SPLIT response is required.

8. Appendix B (Simulation results)

Appendix B1 Waveforms

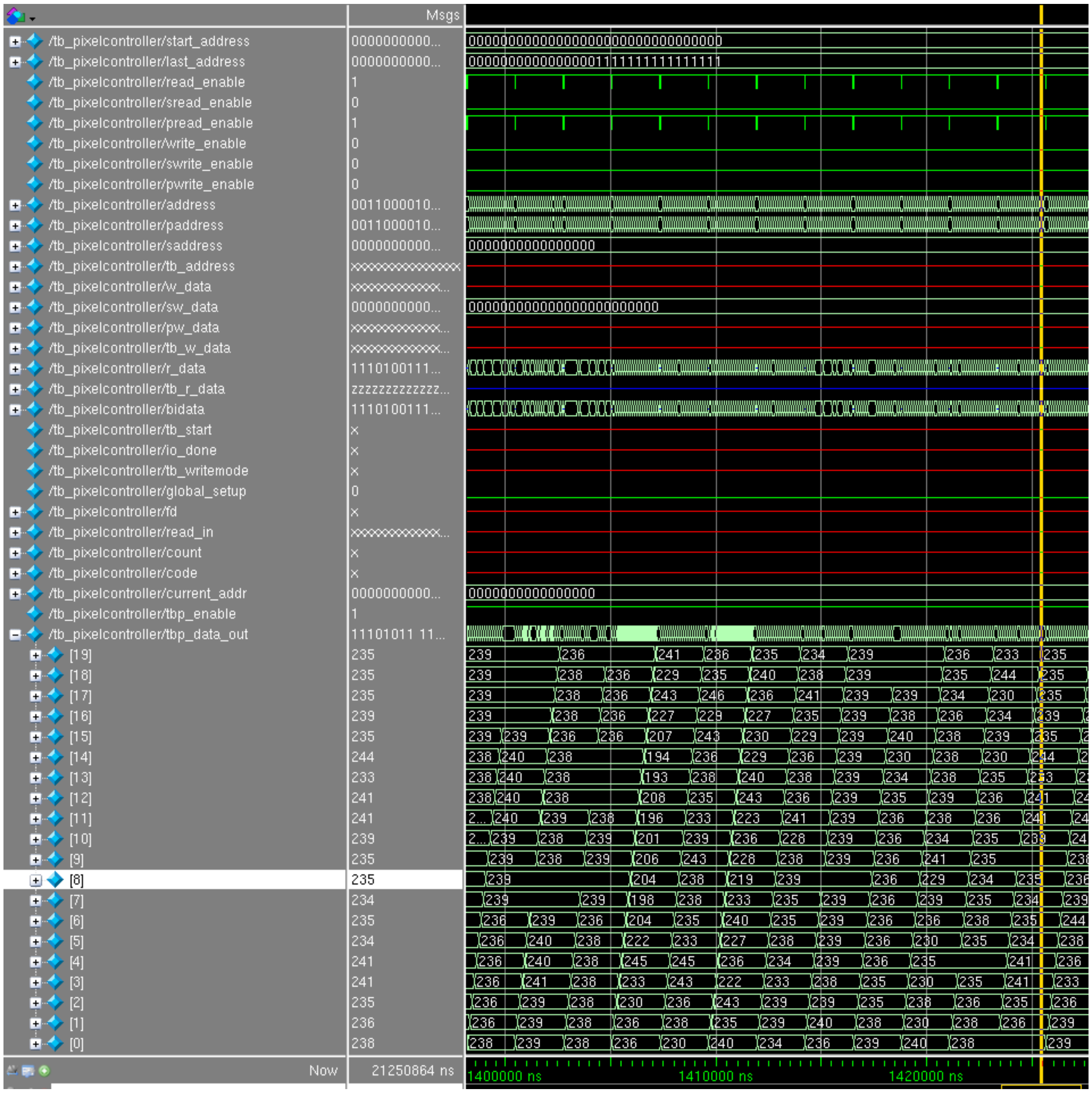
B1.1 Waveform for Pixel Controller block (Source Simulation)



	Msgs
/tb_pixelcontroller/paddress	0111001111...
/tb_pixelcontroller/saddress	0000000000...
/tb_pixelcontroller/tb_address	xxxxxxxxxxxxxxxx
/tb_pixelcontroller/w_data	zzzzzzzzzzzzz...
/tb_pixelcontroller/sw_data	0000000000...
/tb_pixelcontroller/pw_data	zzzzzzzzzzzzz...
/tb_pixelcontroller/tb_w_data	xxxxxxxxxxxxx...
/tb_pixelcontroller/r_data	1001110010...
/tb_pixelcontroller/tb_r_data	zzzzzzzzzzzzz...
/tb_pixelcontroller/bidata	1001110010...
/tb_pixelcontroller/tb_start	x
/tb_pixelcontroller/io_done	x
/tb_pixelcontroller/tb_writemode	x
/tb_pixelcontroller/global_setup	0
/tb_pixelcontroller/fd	x
/tb_pixelcontroller/read_in	xxxxxxxxxxxxx...
/tb_pixelcontroller/count	x
/tb_pixelcontroller/code	x
/tb_pixelcontroller/current_addr	0000000000...
/tb_pixelcontroller/tbp_enable	1
/tb_pixelcontroller/tbp_data_out	01111001 01...
[19]	121
[18]	127
[17]	123
[16]	122
[15]	119
[14]	121
[13]	121
[12]	128
[11]	149
[10]	139
[9]	133
[8]	128
[7]	128
[6]	114
[5]	124
[4]	119
[3]	117
[2]	118
[1]	118
[0]	117
/tb_pixelcontroller/tbp_data_in	xxxxxxxx xxxx...
/tb_pixelcontroller/tbp_address_write_offset	0000000000...
/tb_pixelcontroller/tbp_address_read_offset	0111010000...
/tb_pixelcontroller/tbp_num_pix_read	0000000000...
/tb_pixelcontroller/tbp_num_pix_write	0000000000...
/tb_pixelcontroller/tbp_n_rst	1
/tb_pixelcontroller/tbp_read_now	0
/tb_pixelcontroller/in_file	x
/tb_pixelcontroller/out_file	2
Now	8554903.636 ns
Cursor 1	3404900.416 ns

Fig B1.1 Mapped Waveform Pixel Controller

Mapped Waveform *- the data_out output is expanded and each pixels can be seen cascading. In this case the testbench requests for 20 continuous pixels from pixel 0 to* *the last pixel in the image in the SRAM. The greyscale pixel value is calculated combinationally using bit shift to achieve similar effect as averaging all RGB values. The result produced at the end of this operation is shown in Figure B2.2 Testcase 5.*



```
                                              Msgs
/tb_pixelcontroller/start_address       0000000000...
/tb_pixelcontroller/last_address        0000000000...
/tb_pixelcontroller/read_enable         1
/tb_pixelcontroller/sread_enable        0
/tb_pixelcontroller/pread_enable        1
/tb_pixelcontroller/write_enable        0
/tb_pixelcontroller/swrite_enable       0
/tb_pixelcontroller/pwrite_enable       0
/tb_pixelcontroller/address             0011000010...
/tb_pixelcontroller/paddress            0011000010...
/tb_pixelcontroller/saddress            0000000000...
/tb_pixelcontroller/tb_address          xxxxxxxxxxxxxxxx
/tb_pixelcontroller/w_data              xxxxxxxxxxxx...
/tb_pixelcontroller/sw_data             0000000000...
/tb_pixelcontroller/pw_data             xxxxxxxxxxxx...
/tb_pixelcontroller/tb_w_data           xxxxxxxxxxxx...
/tb_pixelcontroller/r_data              1110100111...
/tb_pixelcontroller/tb_r_data           zzzzzzzzzzzzz...
/tb_pixelcontroller/bidata              1110100111...
/tb_pixelcontroller/tb_start            x
/tb_pixelcontroller/io_done             x
/tb_pixelcontroller/tb_writemode        x
/tb_pixelcontroller/global_setup        0
/tb_pixelcontroller/fd                  x
/tb_pixelcontroller/read_in             xxxxxxxxxxxx...
/tb_pixelcontroller/count               x
/tb_pixelcontroller/code                x
/tb_pixelcontroller/current_addr        0000000000...
/tb_pixelcontroller/tbp_enable          1
/tb_pixelcontroller/tbp_data_out        11101011 11...
  [19]                                  235
  [18]                                  235
  [17]                                  235
  [16]                                  239
  [15]                                  235
  [14]                                  244
  [13]                                  233
  [12]                                  241
  [11]                                  241
  [10]                                  239
  [9]                                   235
  [8]                                   235
  [7]                                   234
  [6]                                   235
  [5]                                   234
  [4]                                   241
  [3]                                   241
  [2]                                   235
  [1]                                   236
  [0]                                   238
Now                                     21250864 ns
```

Fig B1.1.2 Source Waveform Pixel Controller

Source SImulation Waveform - This produces identical result as the mapped version.

B1.1.2 - Pixel Controller (*continued)*

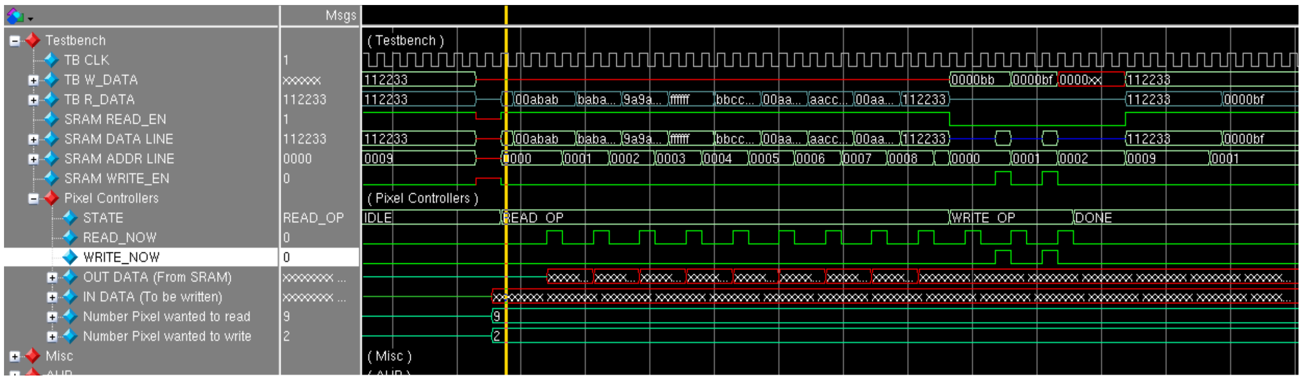
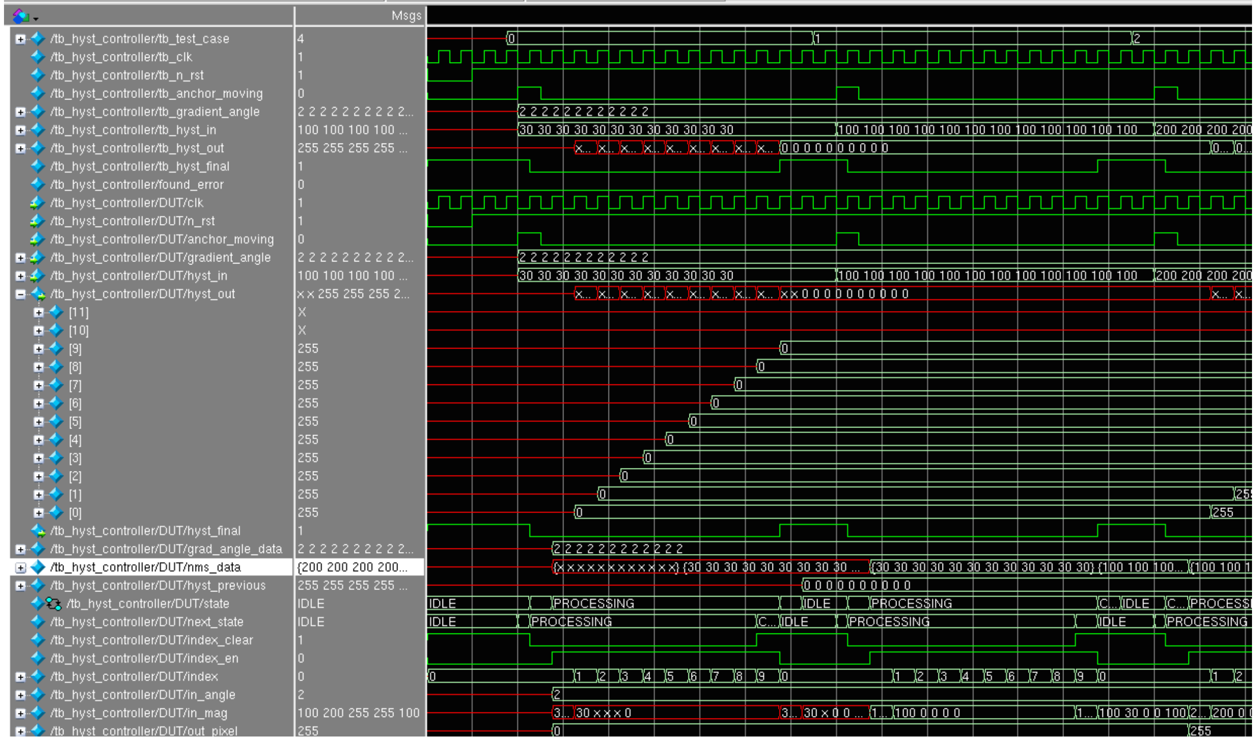


Figure B1.1.3 ***Testing Read Write -*** This simulation above is a simplified testbench that tests a single read and single write operation continuously. Below is the states of the SRAM before and after the IO operations, addresses beyond 0x08 are truncated to save space.

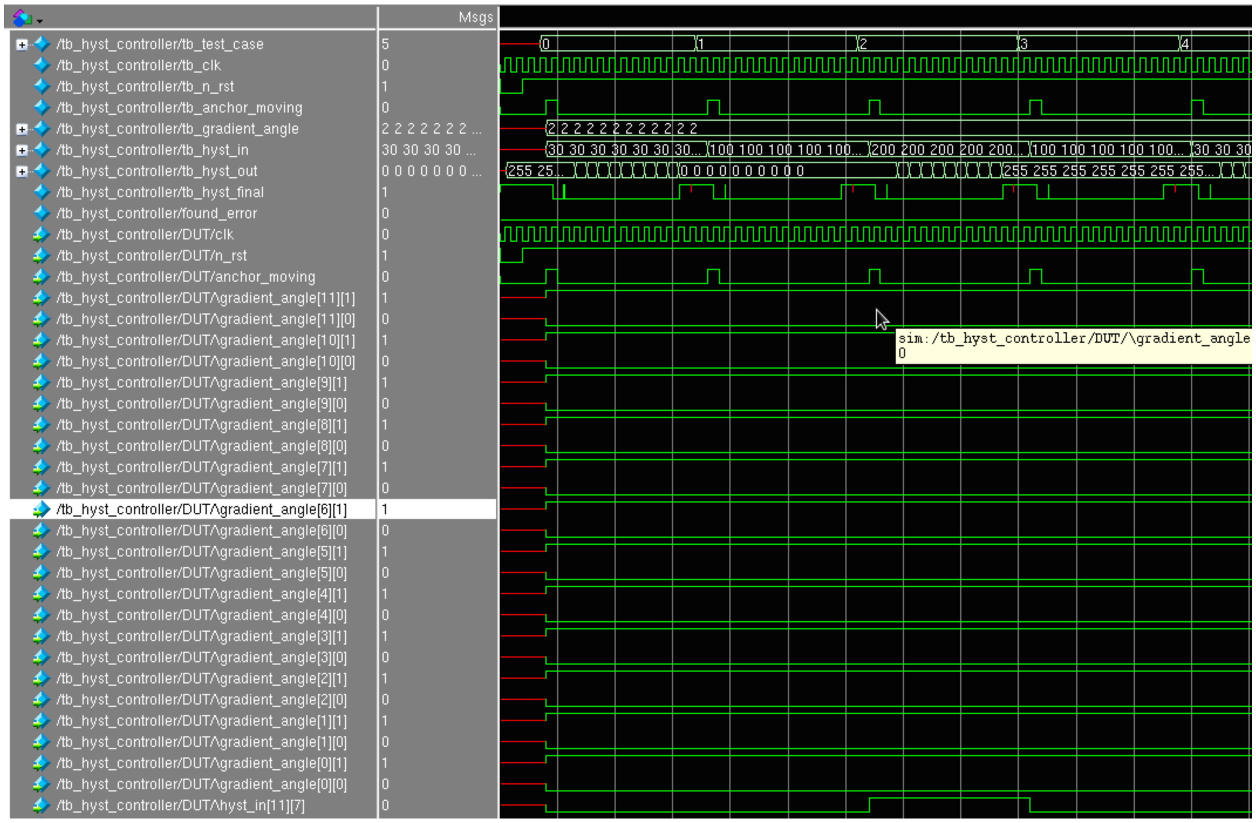
Before	After
0:00ABAB;	0:0000BB;
1:BABABA;	1:0000BF;
2:9A9A9A;	2:9A9A9A;
3:FFFFFF;	3:FFFFFF;
4:BBCCDD;	4:BBCCDD;
5:00AABB;	5:00AABB;
6:AACCDD;	6:AACCDD;
7:00AABB;	7:00AABB;
8:112233;	8:112233;

B2.1 Waveform for Hysteresis Block



```
                                                   Msgs
/tb_hyst_controller/tb_test_case                   4
/tb_hyst_controller/tb_clk                         1
/tb_hyst_controller/tb_n_rst                       1
/tb_hyst_controller/tb_anchor_moving               0
/tb_hyst_controller/tb_gradient_angle              2 2 2 2 2 2 2 2 2 2...
/tb_hyst_controller/tb_hyst_in                     100 100 100 100 ...
/tb_hyst_controller/tb_hyst_out                    255 255 255 255 ...
/tb_hyst_controller/tb_hyst_final                  1
/tb_hyst_controller/found_error                    0
/tb_hyst_controller/DUT/clk                        1
/tb_hyst_controller/DUT/n_rst                      1
/tb_hyst_controller/DUT/anchor_moving              0
/tb_hyst_controller/DUT/gradient_angle             2 2 2 2 2 2 2 2 2 2...
/tb_hyst_controller/DUT/hyst_in                    100 100 100 100 ...
/tb_hyst_controller/DUT/hyst_out                   x x 255 255 255 2...
  [11]                                             X
  [10]                                             X
  [9]                                              255
  [8]                                              255
  [7]                                              255
  [6]                                              255
  [5]                                              255
  [4]                                              255
  [3]                                              255
  [2]                                              255
  [1]                                              255
  [0]                                              255
/tb_hyst_controller/DUT/hyst_final                 1
/tb_hyst_controller/DUT/grad_angle_data            2 2 2 2 2 2 2 2 2 2...
/tb_hyst_controller/DUT/nms_data                   {200 200 200 200...
/tb_hyst_controller/DUT/hyst_previous              255 255 255 255 ...
/tb_hyst_controller/DUT/state                      IDLE
/tb_hyst_controller/DUT/next_state                 IDLE
/tb_hyst_controller/DUT/index_clear                1
/tb_hyst_controller/DUT/index_en                   0
/tb_hyst_controller/DUT/index                      0
/tb_hyst_controller/DUT/in_angle                   2
/tb_hyst_controller/DUT/in_mag                     100 200 255 255 100
/tb_hyst_controller/DUT/out_pixel                  255
```

Figure B2.1.1 Source Simulation

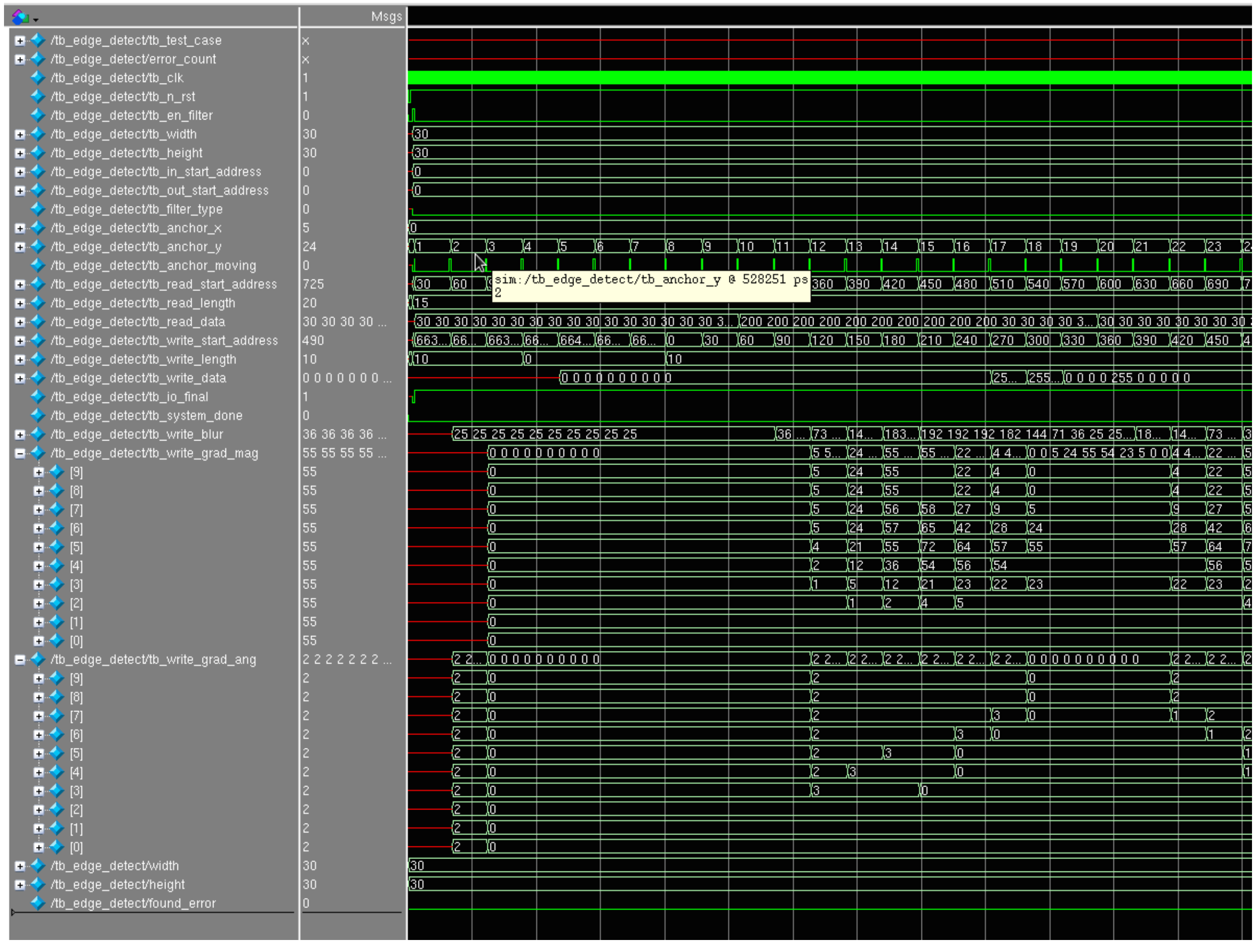


```
                                                   Msgs
/tb_hyst_controller/tb_test_case                   5
/tb_hyst_controller/tb_clk                         0
/tb_hyst_controller/tb_n_rst                       1
/tb_hyst_controller/tb_anchor_moving               0
/tb_hyst_controller/tb_gradient_angle              2 2 2 2 2 2 2 ...
/tb_hyst_controller/tb_hyst_in                     30 30 30 30 ...
/tb_hyst_controller/tb_hyst_out                    0 0 0 0 0 0 0 ...
/tb_hyst_controller/tb_hyst_final                  1
/tb_hyst_controller/found_error                    0
/tb_hyst_controller/DUT/clk                        0
/tb_hyst_controller/DUT/n_rst                      1
/tb_hyst_controller/DUT/anchor_moving              0
/tb_hyst_controller/DUT/\gradient_angle[11][1]     1
/tb_hyst_controller/DUT/\gradient_angle[11][0]     0
/tb_hyst_controller/DUT/\gradient_angle[10][1]     1
/tb_hyst_controller/DUT/\gradient_angle[10][0]     0
/tb_hyst_controller/DUT/\gradient_angle[9][1]      1
/tb_hyst_controller/DUT/\gradient_angle[9][0]      0
/tb_hyst_controller/DUT/\gradient_angle[8][1]      1
/tb_hyst_controller/DUT/\gradient_angle[8][0]      0
/tb_hyst_controller/DUT/\gradient_angle[7][1]      1
/tb_hyst_controller/DUT/\gradient_angle[7][0]      0
/tb_hyst_controller/DUT/\gradient_angle[6][1]      1
/tb_hyst_controller/DUT/\gradient_angle[6][0]      0
/tb_hyst_controller/DUT/\gradient_angle[5][1]      1
/tb_hyst_controller/DUT/\gradient_angle[5][0]      0
/tb_hyst_controller/DUT/\gradient_angle[4][1]      1
/tb_hyst_controller/DUT/\gradient_angle[4][0]      0
/tb_hyst_controller/DUT/\gradient_angle[3][1]      1
/tb_hyst_controller/DUT/\gradient_angle[3][0]      0
/tb_hyst_controller/DUT/\gradient_angle[2][1]      1
/tb_hyst_controller/DUT/\gradient_angle[2][0]      0
/tb_hyst_controller/DUT/\gradient_angle[1][1]      1
/tb_hyst_controller/DUT/\gradient_angle[1][0]      0
/tb_hyst_controller/DUT/\gradient_angle[0][1]      1
/tb_hyst_controller/DUT/\gradient_angle[0][0]      0
/tb_hyst_controller/DUT/\hyst_in[11][7]            0

sim:/tb_hyst_controller/DUT/\gradient_angle
0
```

Figure B2.1.2 Mapped Simulation

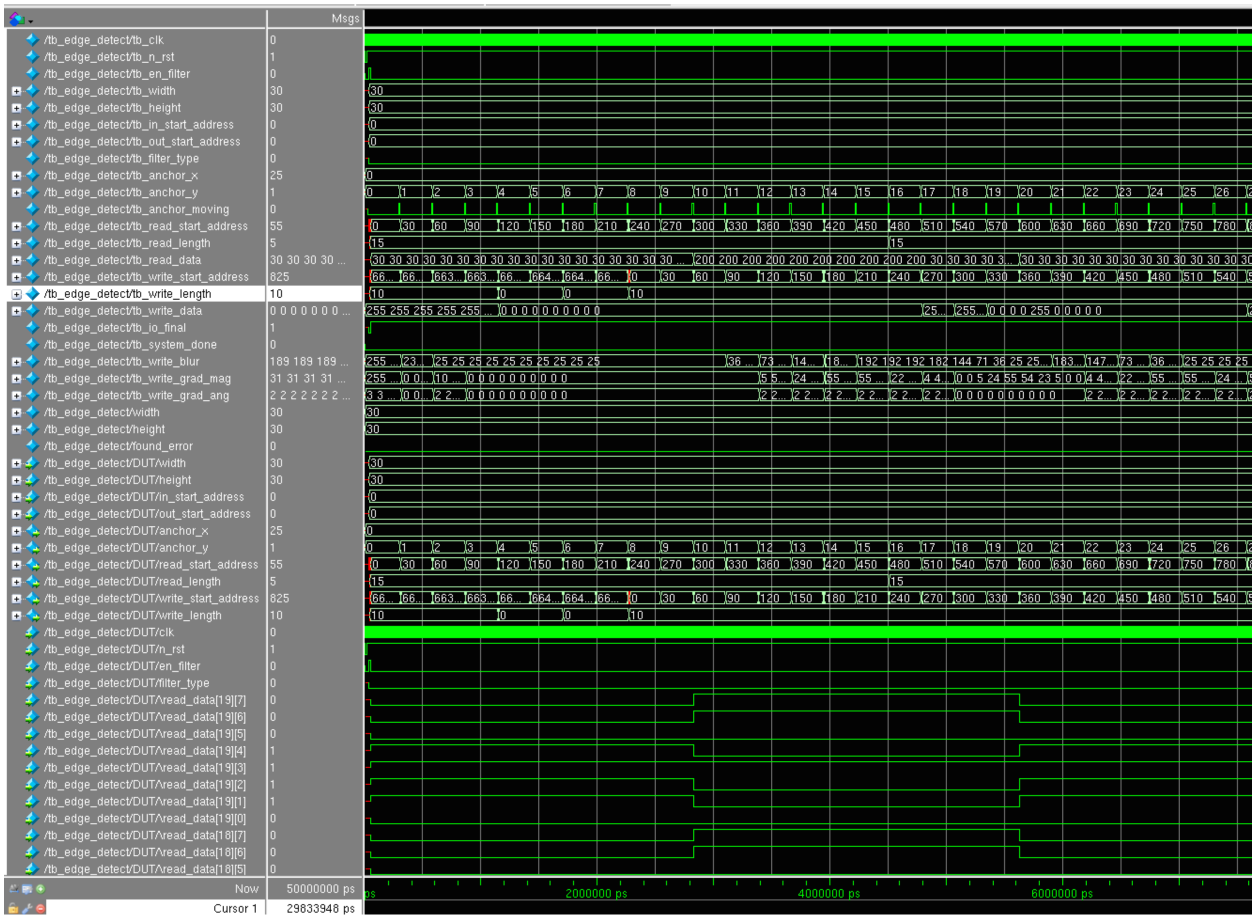
B3.1 Waveform for Edge Detection



	Msgs
/tb_edge_detect/tb_test_case	x
/tb_edge_detect/error_count	x
/tb_edge_detect/tb_clk	1
/tb_edge_detect/tb_n_rst	1
/tb_edge_detect/tb_en_filter	0
/tb_edge_detect/tb_width	30
/tb_edge_detect/tb_height	30
/tb_edge_detect/tb_in_start_address	0
/tb_edge_detect/tb_out_start_address	0
/tb_edge_detect/tb_filter_type	0
/tb_edge_detect/tb_anchor_x	5
/tb_edge_detect/tb_anchor_y	24
/tb_edge_detect/tb_anchor_moving	0
/tb_edge_detect/tb_read_start_address	725
/tb_edge_detect/tb_read_length	20
/tb_edge_detect/tb_read_data	30 30 30 30 ...
/tb_edge_detect/tb_write_start_address	490
/tb_edge_detect/tb_write_length	10
/tb_edge_detect/tb_write_data	0 0 0 0 0 0 0 ...
/tb_edge_detect/tb_io_final	1
/tb_edge_detect/tb_system_done	0
/tb_edge_detect/tb_write_blur	36 36 36 36 ...
/tb_edge_detect/tb_write_grad_mag	55 55 55 55 ...
[9]	55
[8]	55
[7]	55
[6]	55
[5]	55
[4]	55
[3]	55
[2]	55
[1]	55
[0]	55
/tb_edge_detect/tb_write_grad_ang	2 2 2 2 2 2 2 ...
[9]	2
[8]	2
[7]	2
[6]	2
[5]	2
[4]	2
[3]	2
[2]	2
[1]	2
[0]	2
/tb_edge_detect/width	30
/tb_edge_detect/height	30
/tb_edge_detect/found_error	0

```
sim:/tb_edge_detect/tb_anchor_y @ 528251 ps
2
```

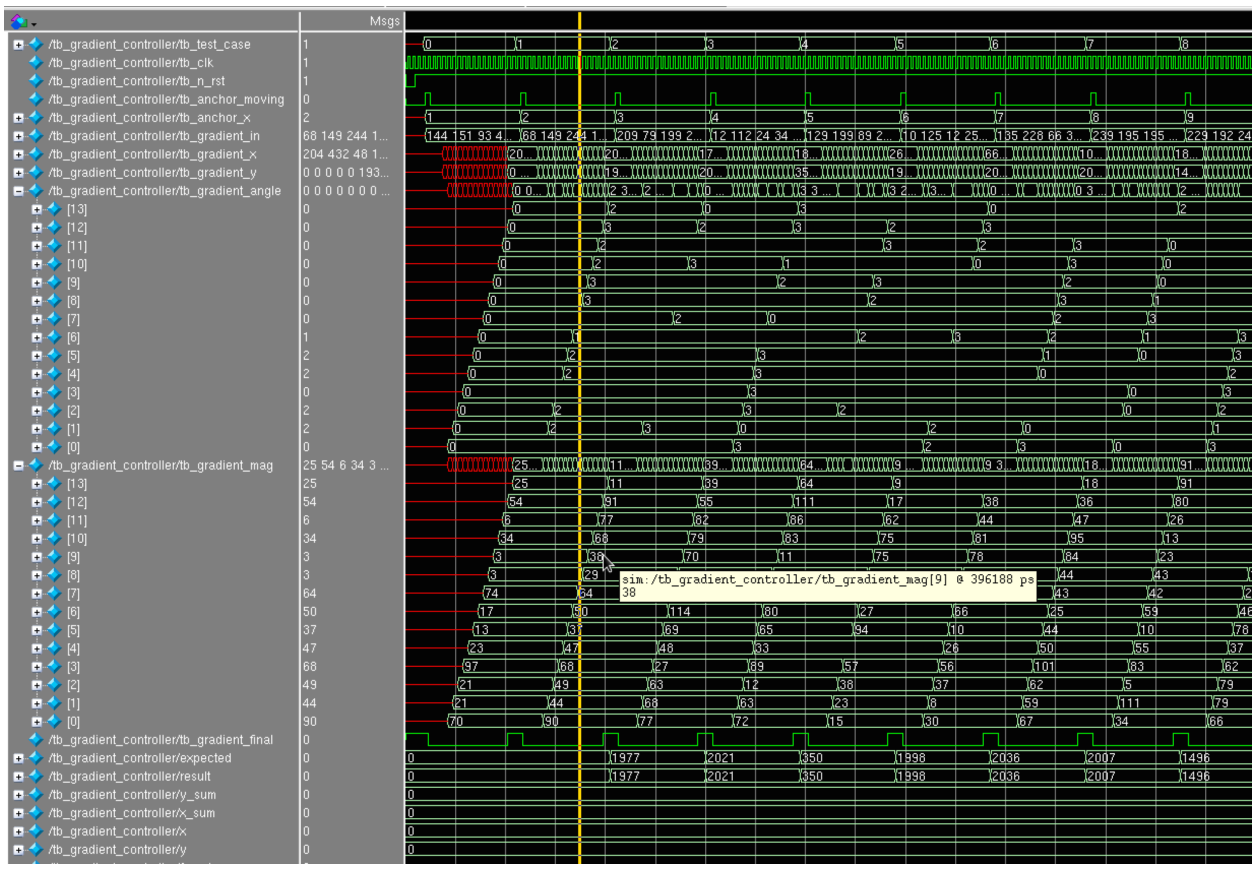
Figure B3.1.1 Source Simulation - Edge Detection



```
                                                  Msgs
/tb_edge_detect/tb_clk                            0
/tb_edge_detect/tb_n_rst                          1
/tb_edge_detect/tb_en_filter                      0
/tb_edge_detect/tb_width                          30
/tb_edge_detect/tb_height                         30
/tb_edge_detect/tb_in_start_address               0
/tb_edge_detect/tb_out_start_address              0
/tb_edge_detect/tb_filter_type                    0
/tb_edge_detect/tb_anchor_x                       25
/tb_edge_detect/tb_anchor_y                       1
/tb_edge_detect/tb_anchor_moving                  0
/tb_edge_detect/tb_read_start_address             55
/tb_edge_detect/tb_read_length                    5
/tb_edge_detect/tb_read_data                      30 30 30 30 ...
/tb_edge_detect/tb_write_start_address            825
/tb_edge_detect/tb_write_length                   10
/tb_edge_detect/tb_write_data                     0 0 0 0 0 0 0 ...
/tb_edge_detect/tb_io_final                       1
/tb_edge_detect/tb_system_done                    0
/tb_edge_detect/tb_write_blur                     189 189 189 ...
/tb_edge_detect/tb_write_grad_mag                 31 31 31 31 ...
/tb_edge_detect/tb_write_grad_ang                 2 2 2 2 2 2 2 ...
/tb_edge_detect/width                             30
/tb_edge_detect/height                            30
/tb_edge_detect/found_error                       0
/tb_edge_detect/DUT/width                         30
/tb_edge_detect/DUT/height                        30
/tb_edge_detect/DUT/in_start_address              0
/tb_edge_detect/DUT/out_start_address             0
/tb_edge_detect/DUT/anchor_x                      25
/tb_edge_detect/DUT/anchor_y                      1
/tb_edge_detect/DUT/read_start_address            55
/tb_edge_detect/DUT/read_length                   5
/tb_edge_detect/DUT/write_start_address           825
/tb_edge_detect/DUT/write_length                  10
/tb_edge_detect/DUT/clk                           0
/tb_edge_detect/DUT/n_rst                         1
/tb_edge_detect/DUT/en_filter                     0
/tb_edge_detect/DUT/filter_type                   0
/tb_edge_detect/DUT/\read_data[19][7]             0
/tb_edge_detect/DUT/\read_data[19][6]             0
/tb_edge_detect/DUT/\read_data[19][5]             0
/tb_edge_detect/DUT/\read_data[19][4]             1
/tb_edge_detect/DUT/\read_data[19][3]             1
/tb_edge_detect/DUT/\read_data[19][2]             1
/tb_edge_detect/DUT/\read_data[19][1]             1
/tb_edge_detect/DUT/\read_data[19][0]             0
/tb_edge_detect/DUT/\read_data[18][7]             0
/tb_edge_detect/DUT/\read_data[18][6]             0
/tb_edge_detect/DUT/\read_data[18][5]             0
                                     Now          50000000 ps
                                     Cursor 1     29833948 ps
```

Figure B3.1.1 Mapped Simulation - Edge Detection

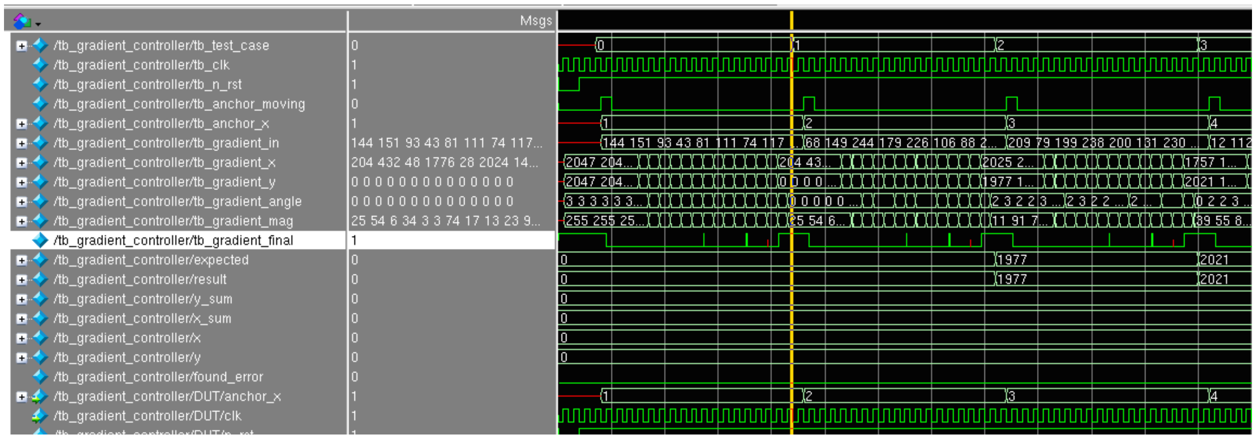
B4.1 Waveform for Gradient Controller



```
                                                Msgs
/tb_gradient_controller/tb_test_case            1
/tb_gradient_controller/tb_clk                  1
/tb_gradient_controller/tb_n_rst                1
/tb_gradient_controller/tb_anchor_moving        0
/tb_gradient_controller/tb_anchor_x             2
/tb_gradient_controller/tb_gradient_in          68 149 244 1...
/tb_gradient_controller/tb_gradient_x           204 432 48 1...
/tb_gradient_controller/tb_gradient_y           0 0 0 0 0 193...
/tb_gradient_controller/tb_gradient_angle       0 0 0 0 0 0 0 ...
  [13]                                          0
  [12]                                          0
  [11]                                          0
  [10]                                          0
  [9]                                           0
  [8]                                           0
  [7]                                           0
  [6]                                           1
  [5]                                           2
  [4]                                           2
  [3]                                           0
  [2]                                           2
  [1]                                           2
  [0]                                           0
/tb_gradient_controller/tb_gradient_mag         25 54 6 34 3 ...
  [13]                                          25
  [12]                                          54
  [11]                                          6
  [10]                                          34
  [9]                                           3
  [8]                                           3
  [7]                                           64
  [6]                                           50
  [5]                                           37
  [4]                                           47
  [3]                                           68
  [2]                                           49
  [1]                                           44
  [0]                                           90
/tb_gradient_controller/tb_gradient_final       0
/tb_gradient_controller/expected                0
/tb_gradient_controller/result                  0
/tb_gradient_controller/y_sum                   0
/tb_gradient_controller/x_sum                   0
/tb_gradient_controller/x                       0
/tb_gradient_controller/y                       0
```

```
sim:/tb_gradient_controller/tb_gradient_mag[9] @ 396188 ps
38
```

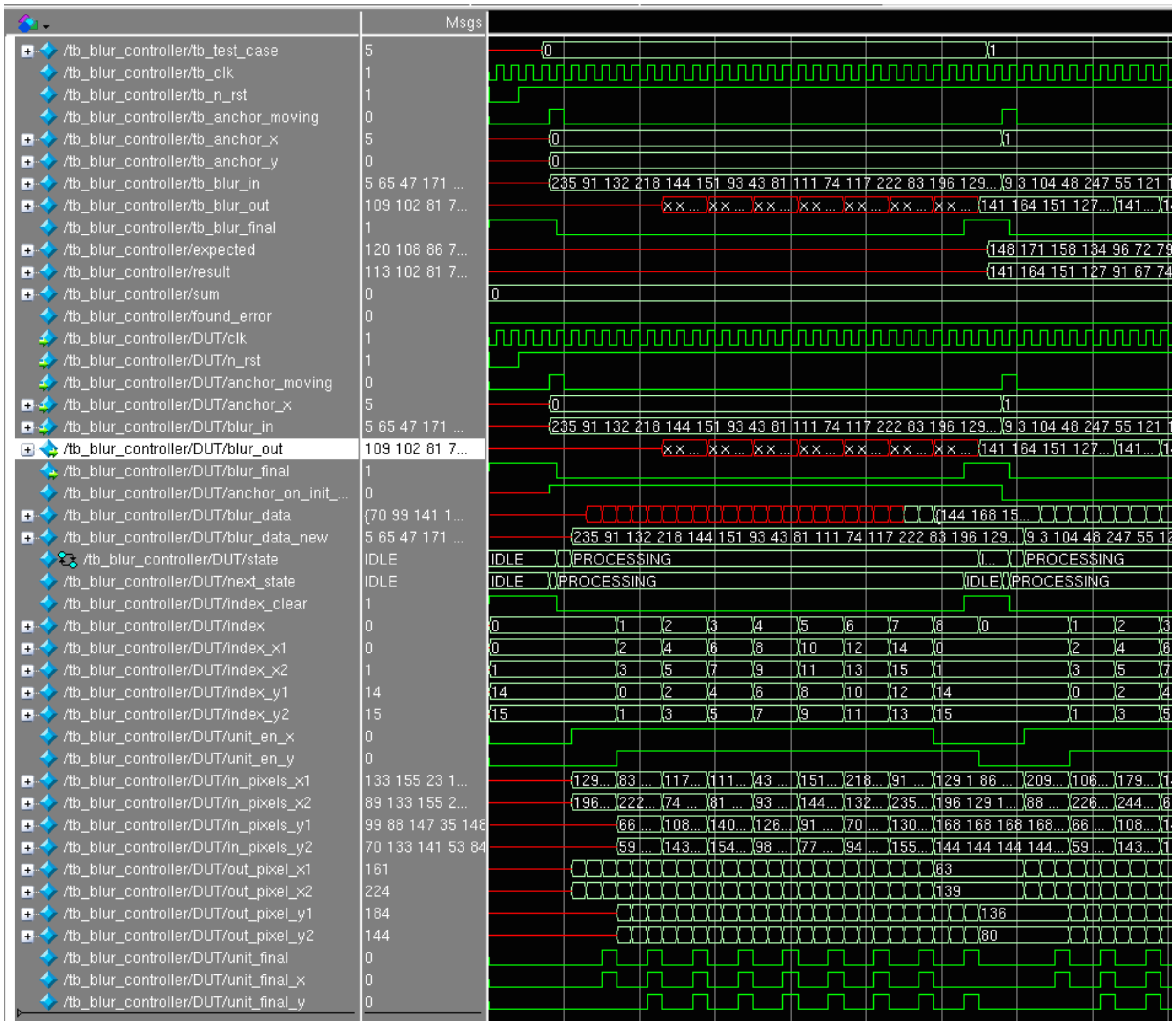
Figure B4.1.1 Source Simulation - Gradient Block



```
                                                Msgs
/tb_gradient_controller/tb_test_case            0
/tb_gradient_controller/tb_clk                  1
/tb_gradient_controller/tb_n_rst                1
/tb_gradient_controller/tb_anchor_moving        0
/tb_gradient_controller/tb_anchor_x             1
/tb_gradient_controller/tb_gradient_in          144 151 93 43 81 111 74 117...
/tb_gradient_controller/tb_gradient_x           204 432 48 1776 28 2024 14...
/tb_gradient_controller/tb_gradient_y           0 0 0 0 0 0 0 0 0 0 0 0 0 0
/tb_gradient_controller/tb_gradient_angle       0 0 0 0 0 0 0 0 0 0 0 0 0 0
/tb_gradient_controller/tb_gradient_mag         25 54 6 34 3 3 74 17 13 23 9...
/tb_gradient_controller/tb_gradient_final       1
/tb_gradient_controller/expected                0
/tb_gradient_controller/result                  0
/tb_gradient_controller/y_sum                   0
/tb_gradient_controller/x_sum                   0
/tb_gradient_controller/x                       0
/tb_gradient_controller/y                       0
/tb_gradient_controller/found_error             0
/tb_gradient_controller/DUT/anchor_x            1
/tb_gradient_controller/DUT/clk                 1
```

Figure B4.1.2 Mapped Simulation - Gradient Block

B5.1 Waveform for Blur



```
Msgs
/tb_blur_controller/tb_test_case            5
/tb_blur_controller/tb_clk                  1
/tb_blur_controller/tb_n_rst                1
/tb_blur_controller/tb_anchor_moving        0
/tb_blur_controller/tb_anchor_x             5
/tb_blur_controller/tb_anchor_y             0
/tb_blur_controller/tb_blur_in              5 65 47 171 ...
/tb_blur_controller/tb_blur_out             109 102 81 7...
/tb_blur_controller/tb_blur_final           1
/tb_blur_controller/expected                120 108 86 7...
/tb_blur_controller/result                  113 102 81 7...
/tb_blur_controller/sum                     0
/tb_blur_controller/found_error             0
/tb_blur_controller/DUT/clk                 1
/tb_blur_controller/DUT/n_rst               1
/tb_blur_controller/DUT/anchor_moving       0
/tb_blur_controller/DUT/anchor_x            5
/tb_blur_controller/DUT/blur_in             5 65 47 171 ...
/tb_blur_controller/DUT/blur_out            109 102 81 7...
/tb_blur_controller/DUT/blur_final          1
/tb_blur_controller/DUT/anchor_on_init_...  0
/tb_blur_controller/DUT/blur_data           {70 99 141 1...
/tb_blur_controller/DUT/blur_data_new       5 65 47 171 ...
/tb_blur_controller/DUT/state               IDLE
/tb_blur_controller/DUT/next_state          IDLE
/tb_blur_controller/DUT/index_clear         1
/tb_blur_controller/DUT/index               0
/tb_blur_controller/DUT/index_x1            0
/tb_blur_controller/DUT/index_x2            1
/tb_blur_controller/DUT/index_y1            14
/tb_blur_controller/DUT/index_y2            15
/tb_blur_controller/DUT/unit_en_x           0
/tb_blur_controller/DUT/unit_en_y           0
/tb_blur_controller/DUT/in_pixels_x1        133 155 23 1...
/tb_blur_controller/DUT/in_pixels_x2        89 133 155 2...
/tb_blur_controller/DUT/in_pixels_y1        99 88 147 35 148
/tb_blur_controller/DUT/in_pixels_y2        70 133 141 53 84
/tb_blur_controller/DUT/out_pixel_x1        161
/tb_blur_controller/DUT/out_pixel_x2        224
/tb_blur_controller/DUT/out_pixel_y1        184
/tb_blur_controller/DUT/out_pixel_y2        144
/tb_blur_controller/DUT/unit_final          0
/tb_blur_controller/DUT/unit_final_x        0
/tb_blur_controller/DUT/unit_final_y        0
```

Figure B5.1 Source Simulation - Blur

B6.1 Waveform for Non-Max Suppression

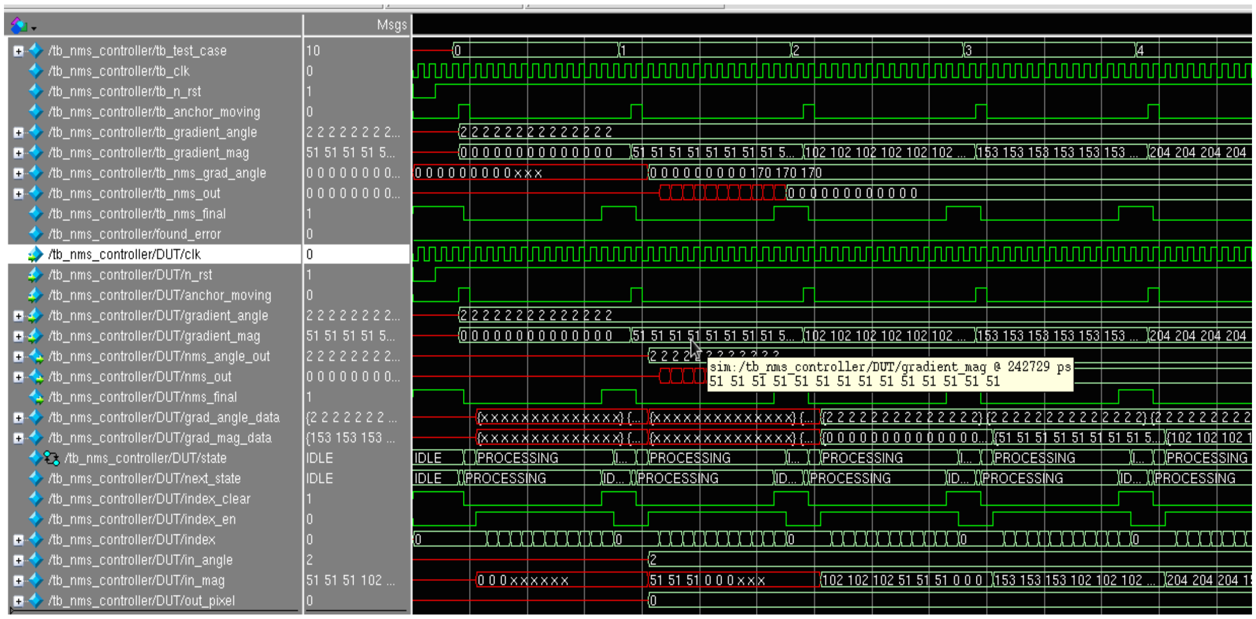

Msgs
/tb_nms_controller/tb_test_case 10
/tb_nms_controller/tb_clk 0
/tb_nms_controller/tb_n_rst 1
/tb_nms_controller/tb_anchor_moving 0
/tb_nms_controller/tb_gradient_angle 2 2 2 2 2 2 2 2...
/tb_nms_controller/tb_gradient_mag 51 51 51 51 5...
/tb_nms_controller/tb_nms_grad_angle 0 0 0 0 0 0 0 0...
/tb_nms_controller/tb_nms_out 0 0 0 0 0 0 0 0...
/tb_nms_controller/tb_nms_final 1
/tb_nms_controller/found_error 0
/tb_nms_controller/DUT/clk 0
/tb_nms_controller/DUT/n_rst 1
/tb_nms_controller/DUT/anchor_moving 0
/tb_nms_controller/DUT/gradient_angle 2 2 2 2 2 2 2 2...
/tb_nms_controller/DUT/gradient_mag 51 51 51 51 5...
/tb_nms_controller/DUT/nms_angle_out 2 2 2 2 2 2 2 2...
/tb_nms_controller/DUT/nms_out 0 0 0 0 0 0 0 0...
/tb_nms_controller/DUT/nms_final 1
/tb_nms_controller/DUT/grad_angle_data {2 2 2 2 2 2 2 ...
/tb_nms_controller/DUT/grad_mag_data {153 153 153 ...
/tb_nms_controller/DUT/state IDLE
/tb_nms_controller/DUT/next_state IDLE
/tb_nms_controller/DUT/index_clear 1
/tb_nms_controller/DUT/index_en 0
/tb_nms_controller/DUT/index 0
/tb_nms_controller/DUT/in_angle 2
/tb_nms_controller/DUT/in_mag 51 51 51 102 ...
/tb_nms_controller/DUT/out_pixel 0
sim:/tb_nms_controller/DUT/gradient_mag @ 242729 ps
51 51 51 51 51 51 51 51 51 51 51 51 51 51
PROCESSING
IDLE

Figure B6.1.1 Source Simulation - NMS

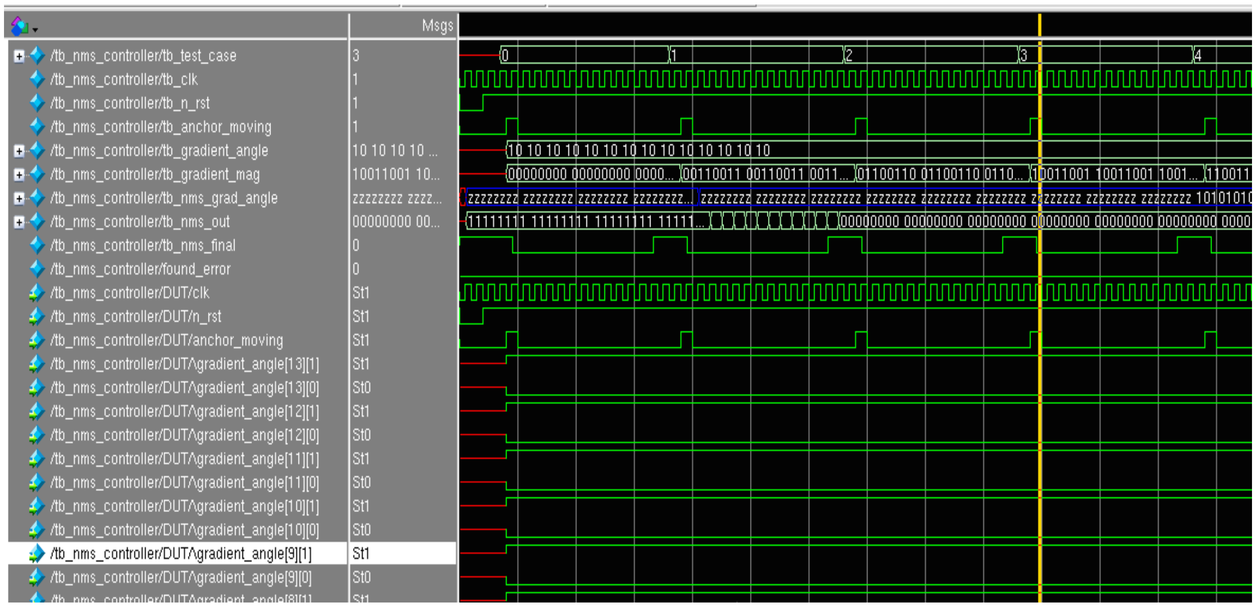
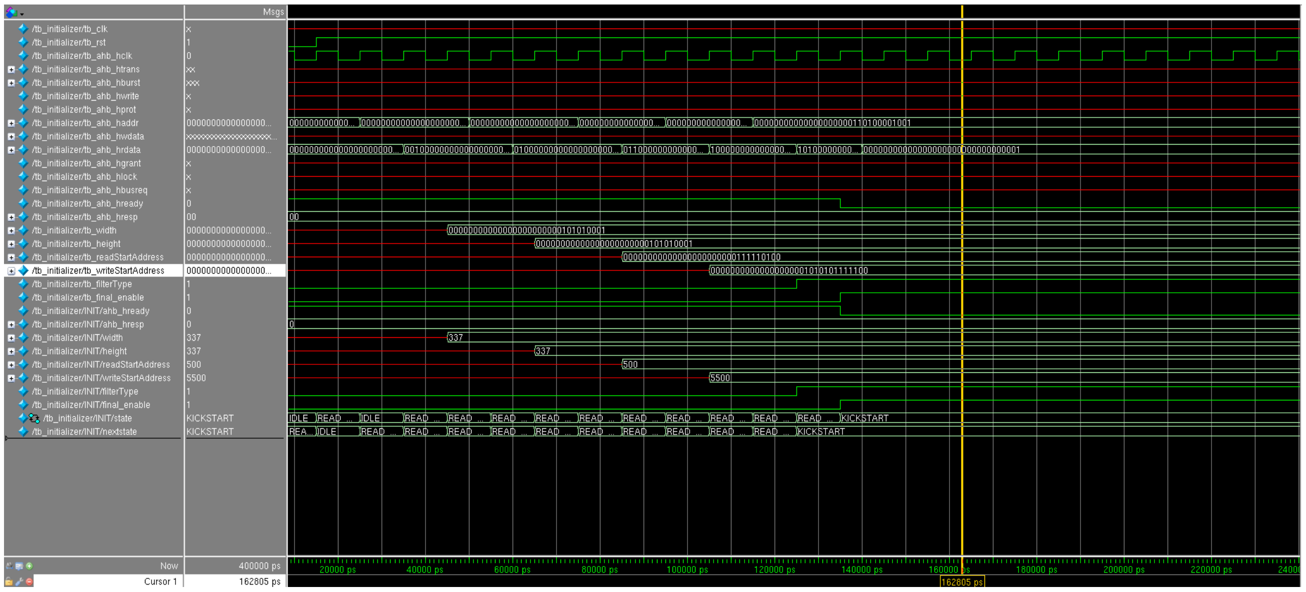

Msgs
/tb_nms_controller/tb_test_case 3
/tb_nms_controller/tb_clk 1
/tb_nms_controller/tb_n_rst 1
/tb_nms_controller/tb_anchor_moving 1
/tb_nms_controller/tb_gradient_angle 10 10 10 10 ...
/tb_nms_controller/tb_gradient_mag 10011001 10...
/tb_nms_controller/tb_nms_grad_angle zzzzzzzz zzzz...
/tb_nms_controller/tb_nms_out 00000000 00...
/tb_nms_controller/tb_nms_final 0
/tb_nms_controller/found_error 0
/tb_nms_controller/DUT/clk St1
/tb_nms_controller/DUT/n_rst St1
/tb_nms_controller/DUT/anchor_moving St1
/tb_nms_controller/DUT/\gradient_angle[13][1] St1
/tb_nms_controller/DUT/\gradient_angle[13][0] St0
/tb_nms_controller/DUT/\gradient_angle[12][1] St1
/tb_nms_controller/DUT/\gradient_angle[12][0] St0
/tb_nms_controller/DUT/\gradient_angle[11][1] St1
/tb_nms_controller/DUT/\gradient_angle[11][0] St0
/tb_nms_controller/DUT/\gradient_angle[10][1] St1
/tb_nms_controller/DUT/\gradient_angle[10][0] St0
/tb_nms_controller/DUT/\gradient_angle[9][1] St1
/tb_nms_controller/DUT/\gradient_angle[9][0] St0

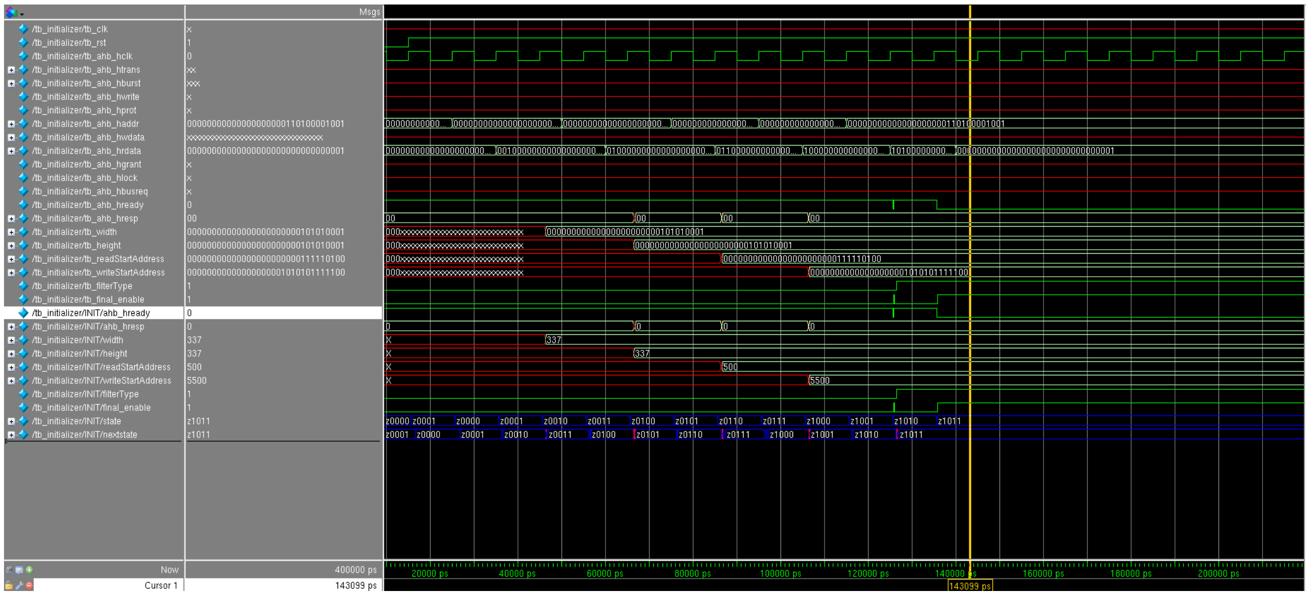
Figure B6.1.2 Mapped Simulation - NMS

B7.0 Waveform for Initializer



/tb_initializer/tb_clk
/tb_initializer/tb_rst
/tb_initializer/tb_ahb_hclk
/tb_initializer/tb_ahb_htrans
/tb_initializer/tb_ahb_hburst
/tb_initializer/tb_ahb_hwrite
/tb_initializer/tb_ahb_hprot
/tb_initializer/tb_ahb_haddr
/tb_initializer/tb_ahb_hwdata
/tb_initializer/tb_ahb_hrdata
/tb_initializer/tb_ahb_hgrant
/tb_initializer/tb_ahb_hlock
/tb_initializer/tb_ahb_hbusreq
/tb_initializer/tb_ahb_hready
/tb_initializer/tb_ahb_hresp
/tb_initializer/tb_width
/tb_initializer/tb_height
/tb_initializer/tb_readStartAddress
/tb_initializer/tb_writeStartAddress
/tb_initializer/tb_filterType
/tb_initializer/tb_final_enable
/tb_initializer/INIT/ahb_hready
/tb_initializer/INIT/ahb_hresp
/tb_initializer/INIT/width
/tb_initializer/INIT/height
/tb_initializer/INIT/readStartAddress
/tb_initializer/INIT/writeStartAddress
/tb_initializer/INIT/filterType
/tb_initializer/INIT/final_enable
/tb_initializer/INIT/state
/tb_initializer/INIT/nextstate
Msgs
337
500
5500
KICKSTART
Now 400000 ps
Cursor 1 162805 ps

Figure B7.0.1 source simulation - initializer



/tb_initializer/tb_clk
/tb_initializer/tb_rst
/tb_initializer/tb_ahb_hclk
/tb_initializer/tb_ahb_htrans
/tb_initializer/tb_ahb_hburst
/tb_initializer/tb_ahb_hwrite
/tb_initializer/tb_ahb_hprot
/tb_initializer/tb_ahb_haddr
/tb_initializer/tb_ahb_hwdata
/tb_initializer/tb_ahb_hrdata
/tb_initializer/tb_ahb_hgrant
/tb_initializer/tb_ahb_hlock
/tb_initializer/tb_ahb_hbusreq
/tb_initializer/tb_ahb_hready
/tb_initializer/tb_ahb_hresp
/tb_initializer/tb_width
/tb_initializer/tb_height
/tb_initializer/tb_readStartAddress
/tb_initializer/tb_writeStartAddress
/tb_initializer/tb_filterType
/tb_initializer/tb_final_enable
/tb_initializer/INIT/ahb_hready
/tb_initializer/INIT/ahb_hresp
/tb_initializer/INIT/width
/tb_initializer/INIT/height
/tb_initializer/INIT/readStartAddress
/tb_initializer/INIT/writeStartAddress
/tb_initializer/INIT/filterType
/tb_initializer/INIT/final_enable
/tb_initializer/INIT/state
/tb_initializer/INIT/nextstate
Msgs
337
500
5500
z1011
Now 400000 ps
Cursor 1 143099 ps

Figure B7.0.2 mapped simulation - initializes

Appendix B2.1 - Testing with Images

Edge Detection block outputs and its intermediate stage. Please note that the pixelcontroller block and the edge detection block were ***not*** directly connected in hardware to produce these outputs, but the testbench was the one that acts as a *glue* between these two processes as a proof of concept.

Also note that there is imperfection in stage 2 (*see next page*), where the Pixelcontroller block was unable to produce the image in perfect manner (offset issue, that was not resolved). Edge detection could be performed on these imperfect images with no problem (shown in Testcase 5). However, to better demo the capability of the edge detection block a software-based solution was implemented to correct these imperfections in stage 2.

(Figures on Next Page)

Testcase 1

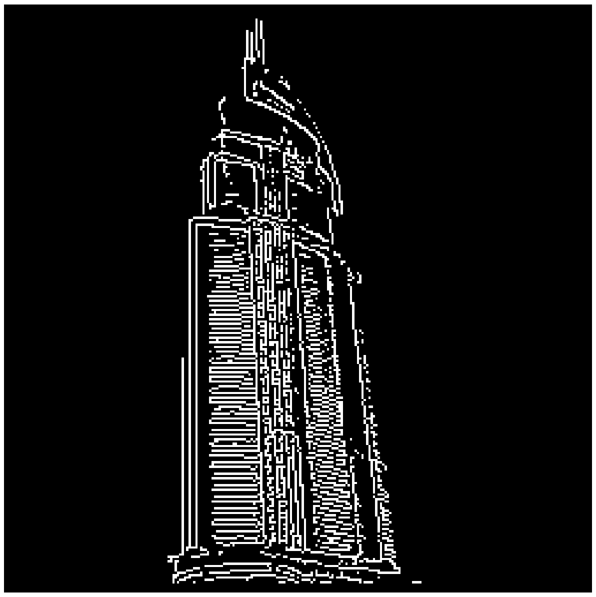
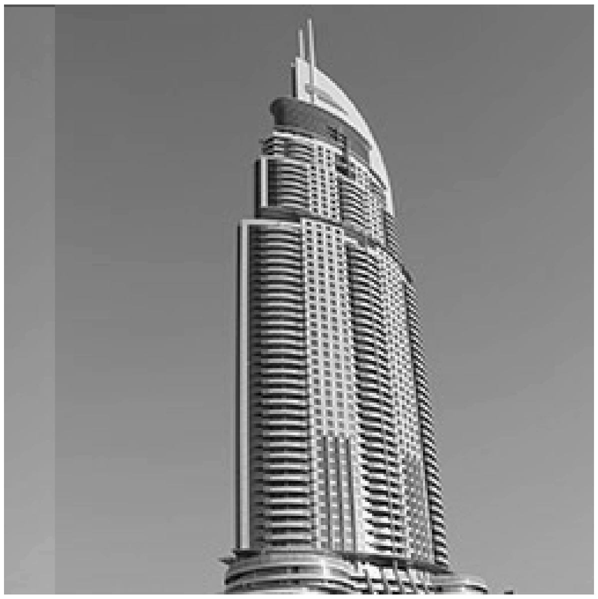
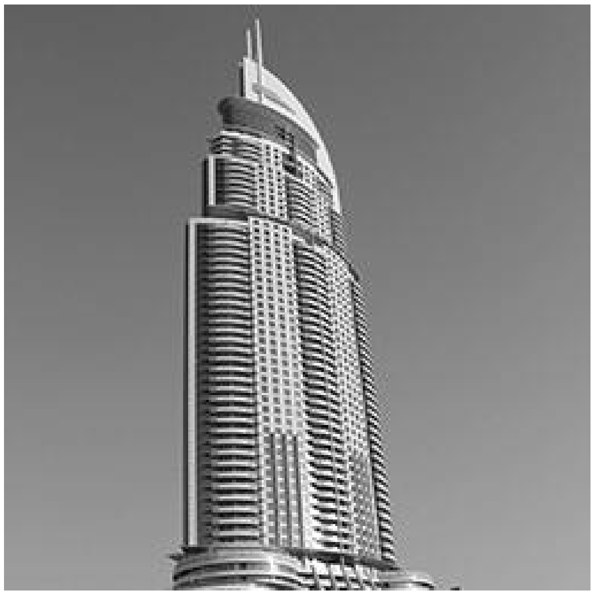
(Stage 1) Original Image	 (Stage 4) Hardware-Produce Edges
 (Stage 2) Hardware-Produced Grayscale	 (Stage 3) Software-Corrected Grayscale

Fig B2.1.1 Tower

Testcase 3

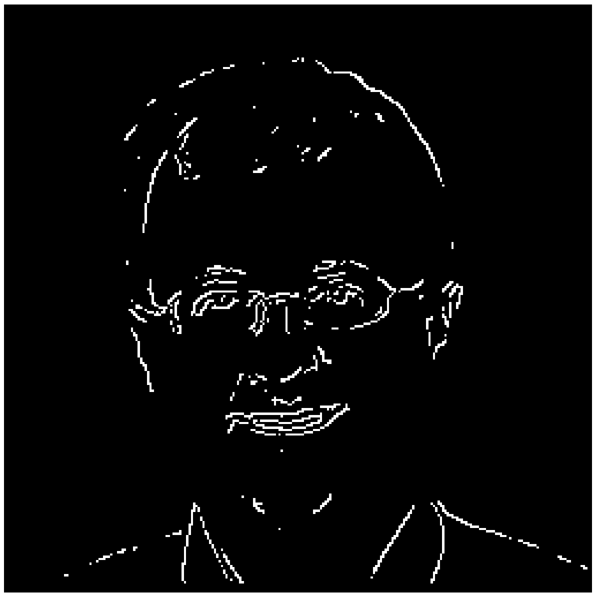
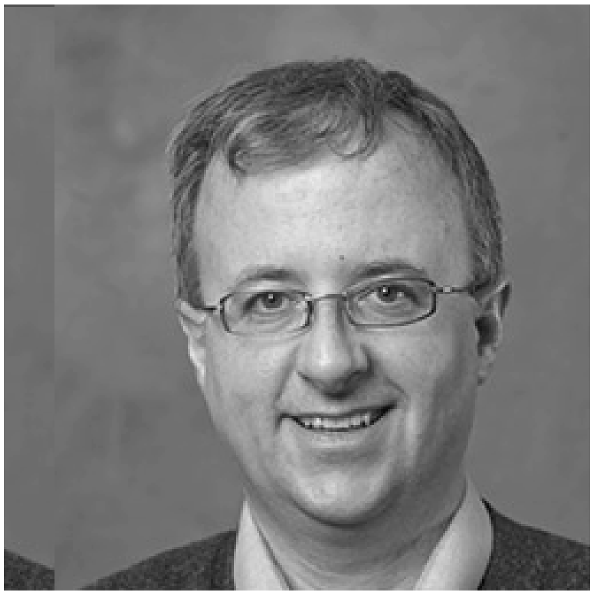
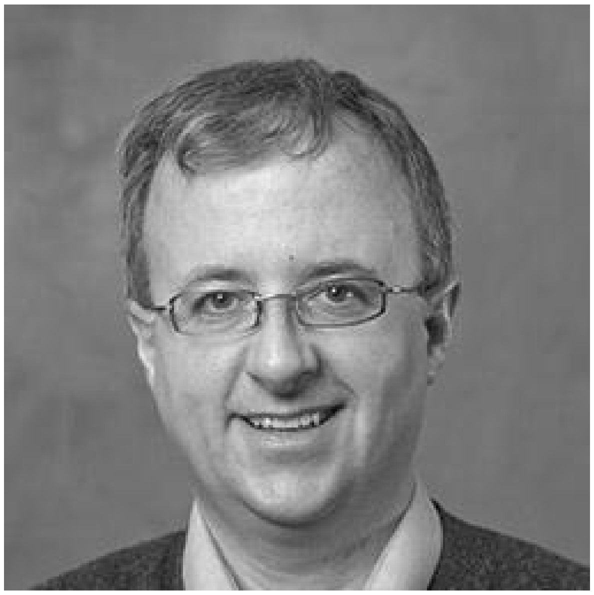
(1) Original Image	 (4) Hardware-Produce Edges
 (2) Hardware-Produced Grayscale	 (3) Software-Corrected Grayscale*

Fig B2.1.2 Dr. Johnson

Testcase 2

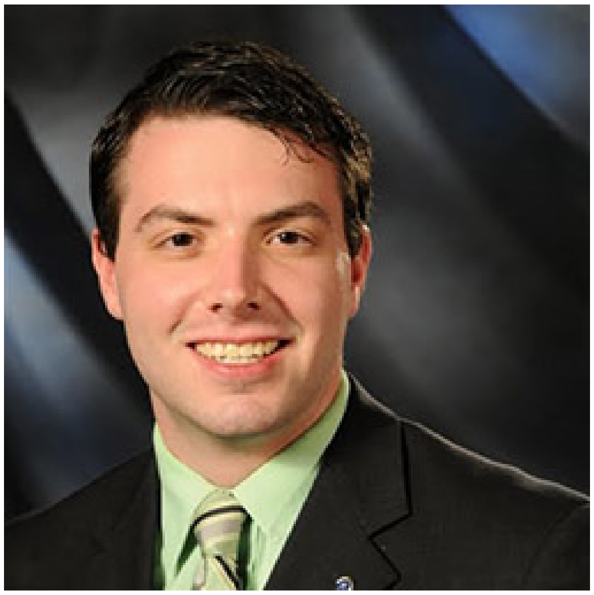
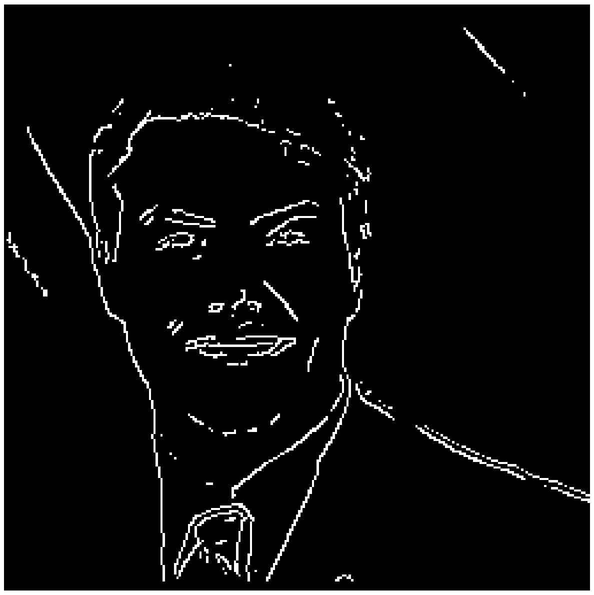
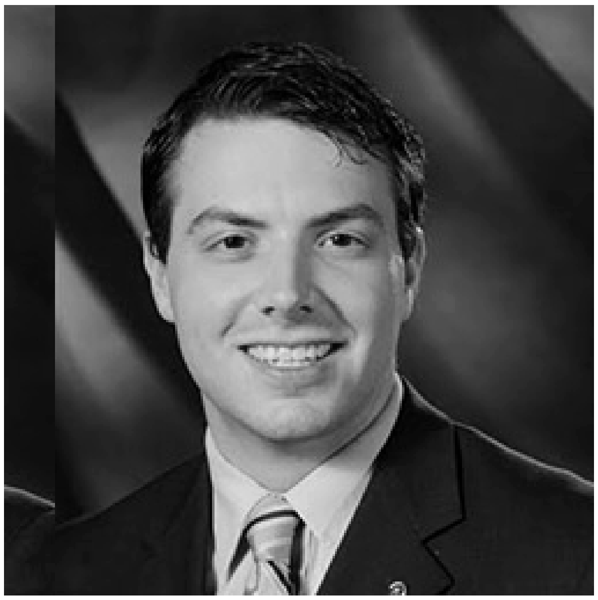
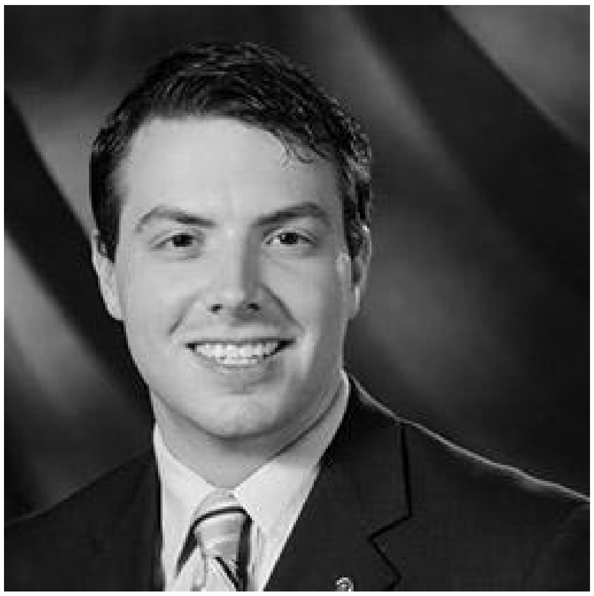
 (1) Original Image	 (4) Hardware-Produce Edges
 (2) Hardware-Produced Grayscale	 (3) Software-Corrected Grayscale

Fig 2.1.3 Tim Pritchett

Testcase 4

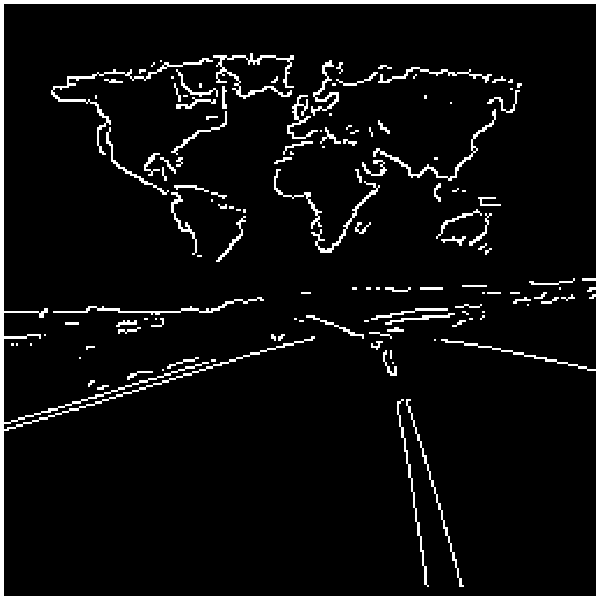
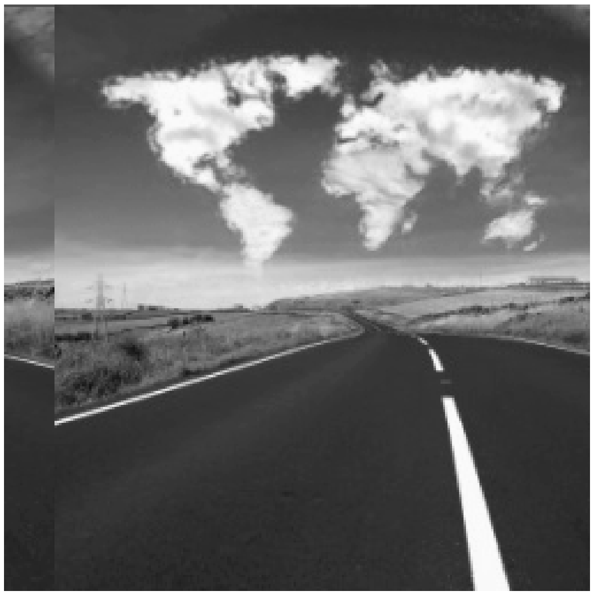
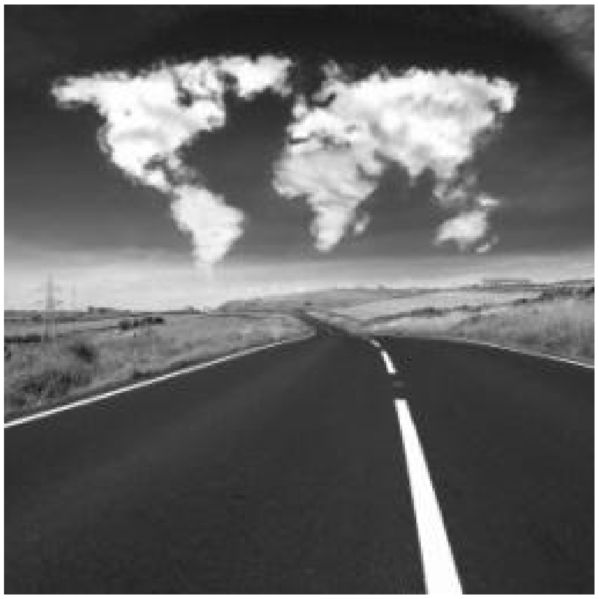
(1) Original Image	 (4) Hardware-Produce Edges
 (2) Hardware-Produced Grayscale	 (3) Software-Corrected Grayscale*

Fig B2.1.4 Road with World Clouds

Testcase 5 (As shown on cover page)

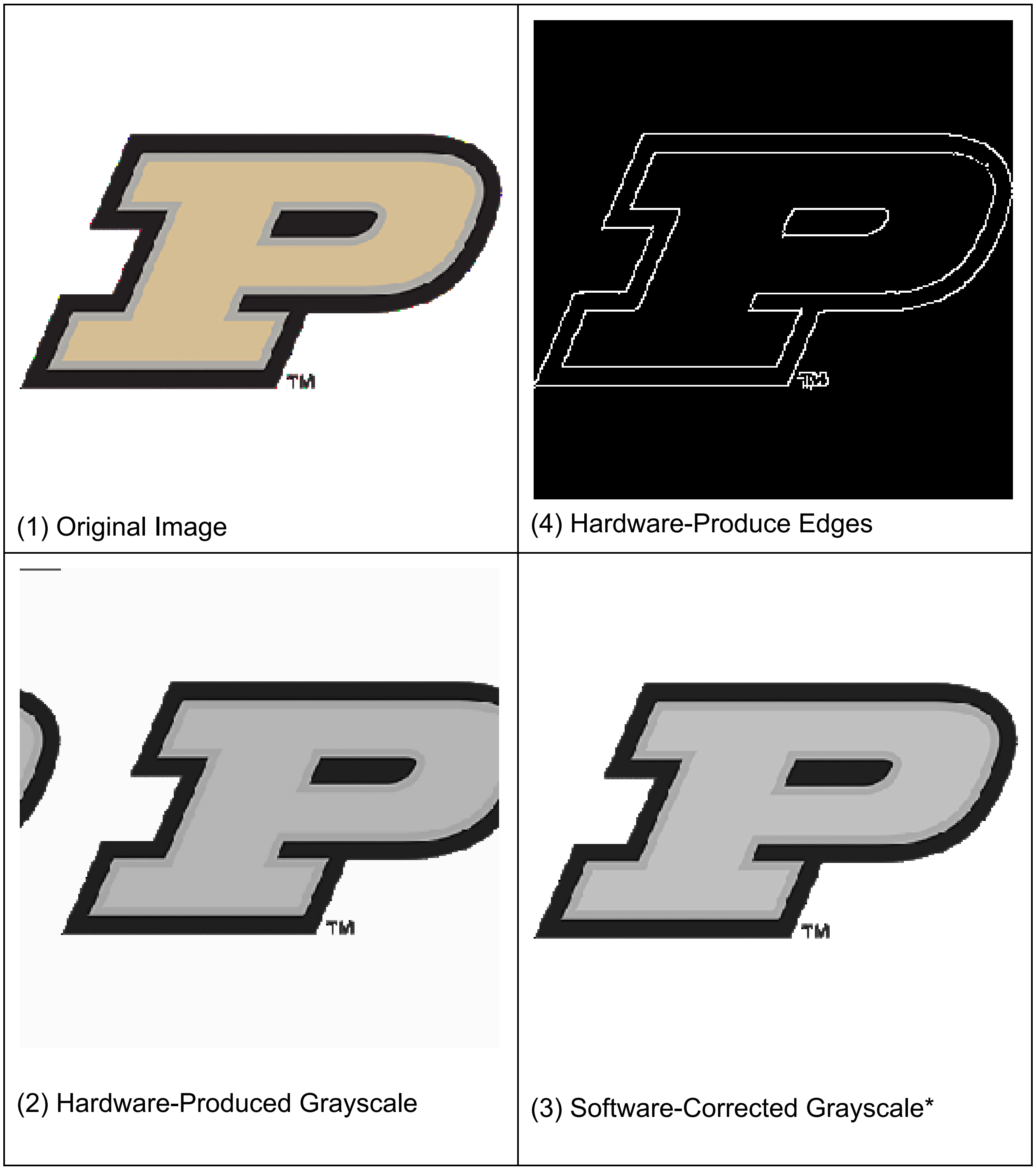

(1) Original Image
(4) Hardware-Produce Edges
(2) Hardware-Produced Grayscale
(3) Software-Corrected Grayscale*

Fig 2.1.5 Purdue Motion “P”

Testcase 6

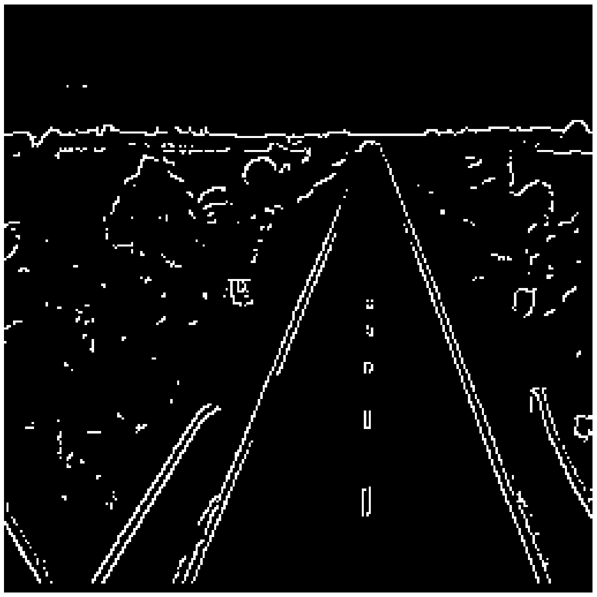
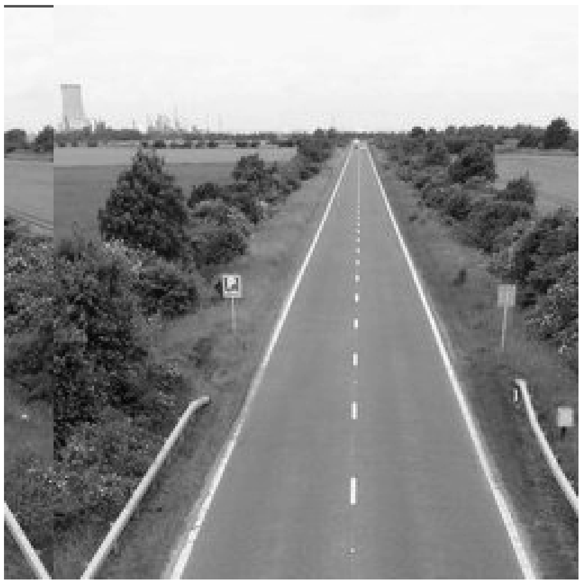
(1) Original Image	 (3) Hardware-Produce Edges
 (2) Hardware-Produced Grayscale **As seen in Appendix B1.1**	

Fig B2.1.6 Road with Trees

End of Report