

Final Report

Raw image demosaicing and filter chip


Abhishek Srikanth
Tiger Cheng
Nikhil Ghanta

ECE 337


Lab section : Tuesday 2:30 pm (#2)

TA: Nathan Conrad

17th December, 2015

Signature 1: _____ 

Signature 2: Nikhil Ghanta _____

Signature 3: _____ 

1. Executive Summary

In cameras today, image data is captured through a massive array of sensors where each sensor detects the intensity of one of the RGB (red, green blue) colors in set pattern. These sensor values gets processed through a demosaicing algorithm that then generates the digital images that are commonly seen today in standard image viewers.

Our design is a hardware implementation of Bayer demosaicing process which interpolates raw images tiled with the 2×2 , {R, G1; G2, B} array pattern to generate pixel data. In addition to converting sensor values into pixel data, our chip is also designed to apply filters such as brightness enhancement, white balancing and horizontal blurring. We will be using the DE2i-150 FPGA board to implement the design.

Having this chip on-board a camera will help convert the sensor values instantly and allow storage of the processed image. Although this process can be done on computers using special software, an on-board chip will enhance user experience by allowing immediate viewback of images taken, an element that would draw or retain customers to a specific company.

We believe that our design is unique since it allows for immediate conversion of the image along with the application of filters, something that no other chip we know off does.

The remainder of the report delves into the technicalities of our design, the interfaces, the operational characteristics, detailed block diagrams of modules, budgeting information, verification plans and results obtained.

2. Design Specifications

2.1. Interfacing Specifications

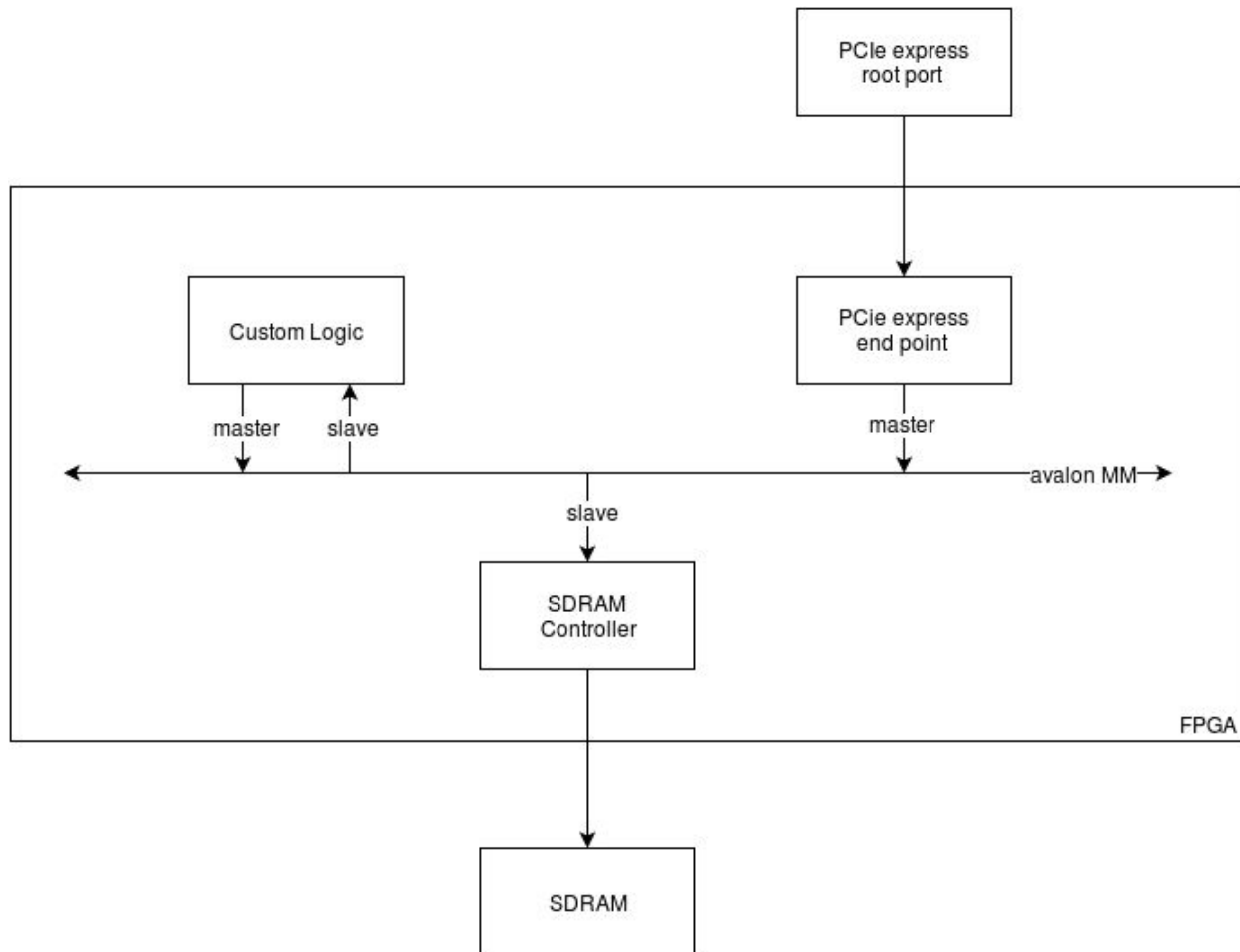


Figure 1. System usage diagram

Standards/Protocols used:

- The Avalon Master/Slave bus protocol is used
- PCIe endpoint is master to Custom Logic
 - Generation 1, transmission rate is 2.5Gbps
- SDRAM is slave to Custom Logic
- Internal M9K SRAM

Features of the design:

- It is designed to cache values read from SDRAM

- This is done to save multiple reads on the SDRAM from non-contiguous memory addresses
- It is designed to cache all output values too before writing to SDRAM
 - This is done to allow write operations to be continuous instead of having to jump between reading and writing each pixel from and to the SDRAM
- It is designed to have all filter outputs be calculated parallelly in combination blocks
 - This allows for easy and quick availability of output values that can be written into the SRAM cache
- It is designed to have a single address line that is capable of switching from one type of address to another for both the SRAM (rowCache|outputArr) and SDRAM(read|write)
 - This creates simplicity in wiring and design and allows address calculation to be hidden inside a single module

2.2. Operational Characteristics

2.2.1. Functions performed by the chip

The chip is expected to carry out the following operations:

- Wait for start control register to go high
- Read control registers for required information (eg. image height and width, filter type, address in SDRAM, etc)
- Read Image off the SDRAM
- Use SRAM to cache intensity values and processed pixel data
- Convert intensity values into pixel data based on filter type

2.2.2. Chip IO Pins

Table 1. Chip IO Pins

Signal Name	Type	No of bits	Data format	description
clk	Input	1	active high	clock to chip
n_rst	Input	1	active low	reset to chip
imageWidth	Input	13	unsigned binary number	width of image
imageHeight	Input	13	unsigned binary number	height of image
startControlRegister	Input	1	active high	register to start chip operations
start_addr_sdram	Input	26	unsigned address	address of start of RAW image
finish_addr_sdram	Input	26	unsigned address	address of start of processed image
filterMode	Input	2	2 bit binary	depicts binary mode
betaValue	Input	8	unsigned binary number	value for brightness filter
white	Input	32	ARGB data	white equivalent for white

				balancing
finish_flag	Output	1	active high	flag when complete image is processed
sdram_datareadvalid	Input	1	active high	pulses high when data is ready to be read
data_sdram	Input	32	intensity value	data read from SDRAM
sdram_read_en	Output	1	active high	signal to enable read
sdram_write_en	Output	1	active high	signal to enable write
address_sdram	Output	26	unsigned address	address for operations
writeData_sdram	Output	32	ARGB data	value to write to SDRAM
data_sram	Input	32	intensity value / ARGB data	data read from SRAM
sram_datareadvalid	Input	1	active high	pulses high when data is ready to be read
sram_dataFromSDRAM	Output	32	intensity value	value to write to SRAM (from SDRAM)
postFilterData	Output	32	ARGB data	value to write to SRAM (from window buffer)
address_sram	Output	26	unsigned address	address for operations
mode_sram	Output	1	active high	determines read/write mode
addrCalc_mode_sram	Output	1	active high	determines address set accessed
sram_en	Output	1	active high	signal to enable SRAM

2.2.3. Control registers

- Image Dimension Control Register
 - `img_width` : [28:16]
 - `img_height` : [12:0]
- Raw Image Start Address Register
 - `raw_img_start_addr` : [25:0]
- Final Image Start Address Register
 - `final_img_start_addr`
- Read Status Register
 - `start_flag` : [0]
 - `filter_type` : [2:1]
 - `brightness_value` : [31:24]
- White Balance Register
 - `red_white_balance_value` : [23:16]
 - `green_white_balance_value` : [15:8]
 - `blue_white_balance_value` : [7:0]
- Finish Register
 - `finish_flag` : [0]

2.2.4. Timing diagrams to illustrate the sequence of operations

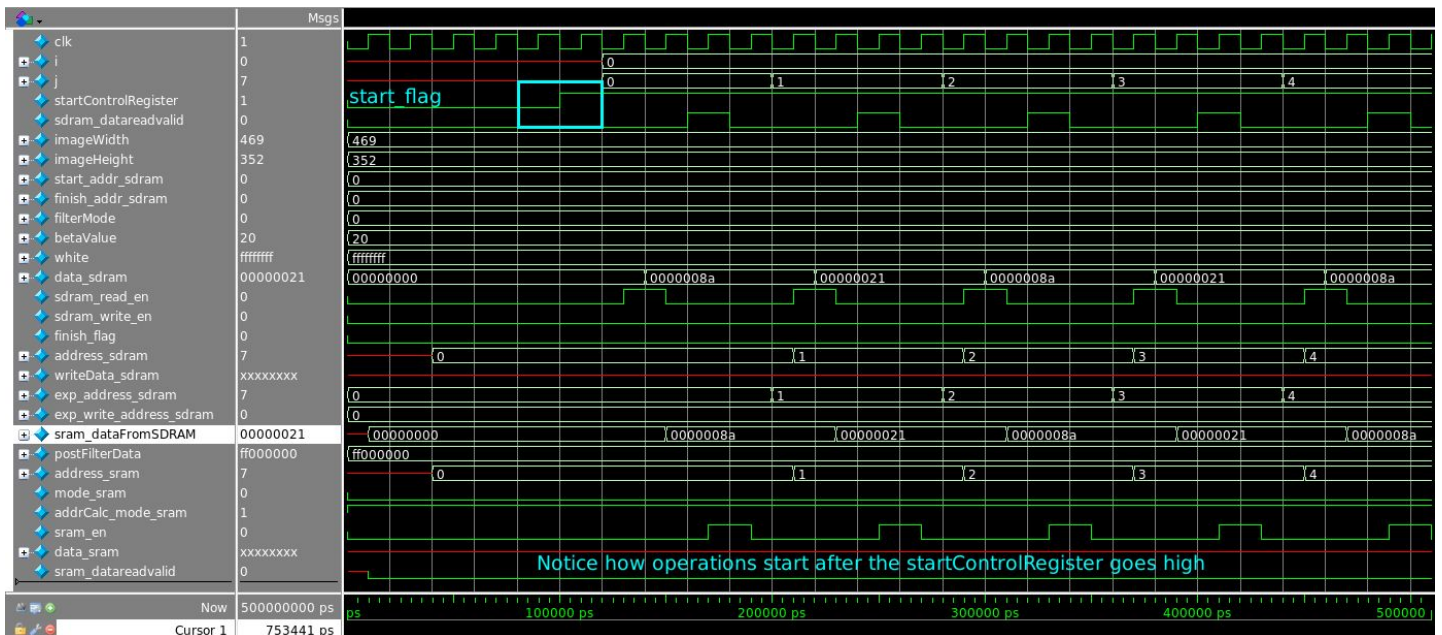


Figure 2. Start detection

The chip stays dormant until the `startControlRegister` is set high. Operations start here.

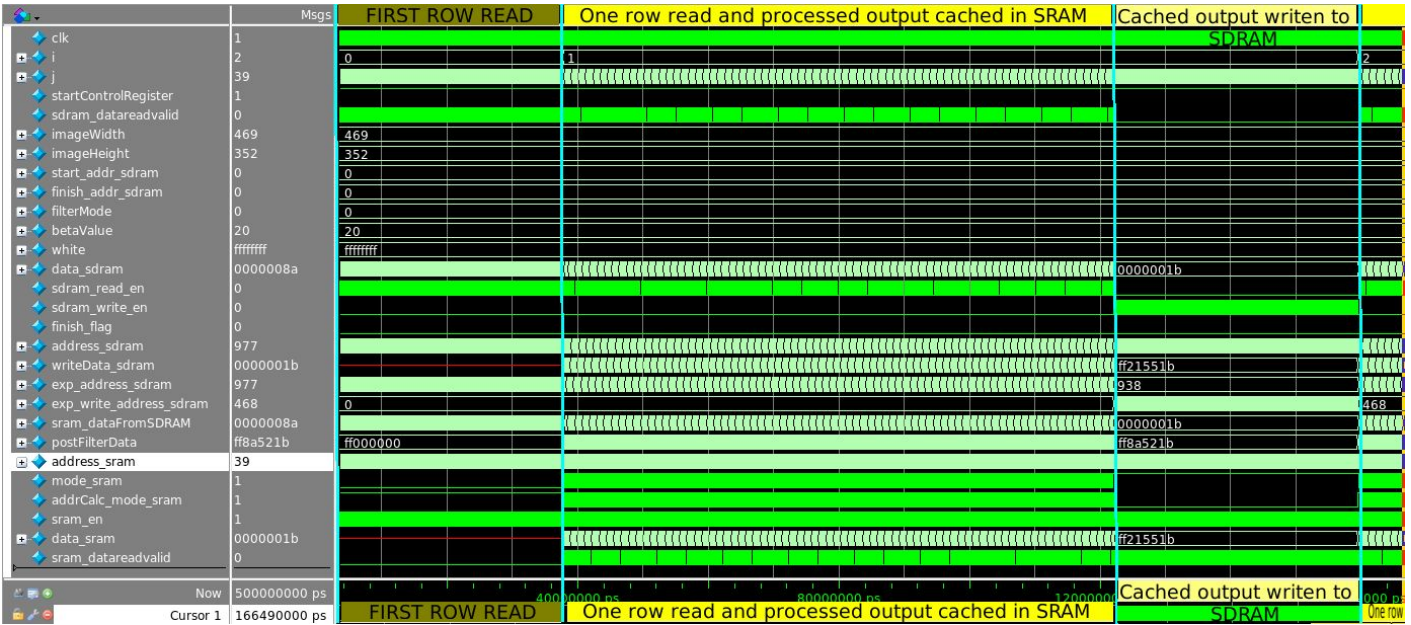


Figure 3. Single round of operations

Above image shows one row (specifically, the first round) of operations.

In the first segment, a row of values are read from the SDRAM and cached in SRAM.

This segment never repeats itself throughout the processing of the image at hand.

In the second segment, a row of data is read from sdr and sram cache to generate pixel data that is stored in another sram cache.

In the third segment, cached pixel data is written into the SDRAM.

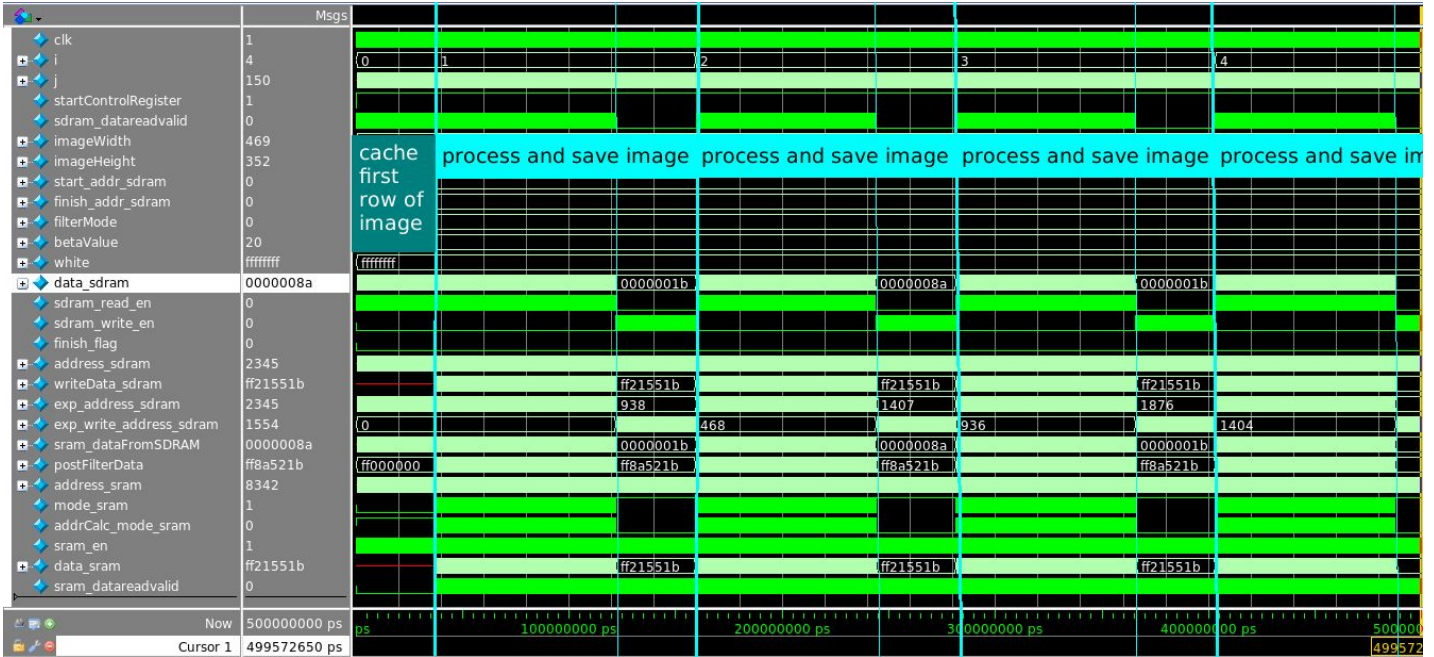


Figure 4. Snapshot of a few rounds (rows 0,1,2,3,4) of operation

The image above shows processing of multiple rows.

Note how segments keep repeating themselves after the caching of first row.

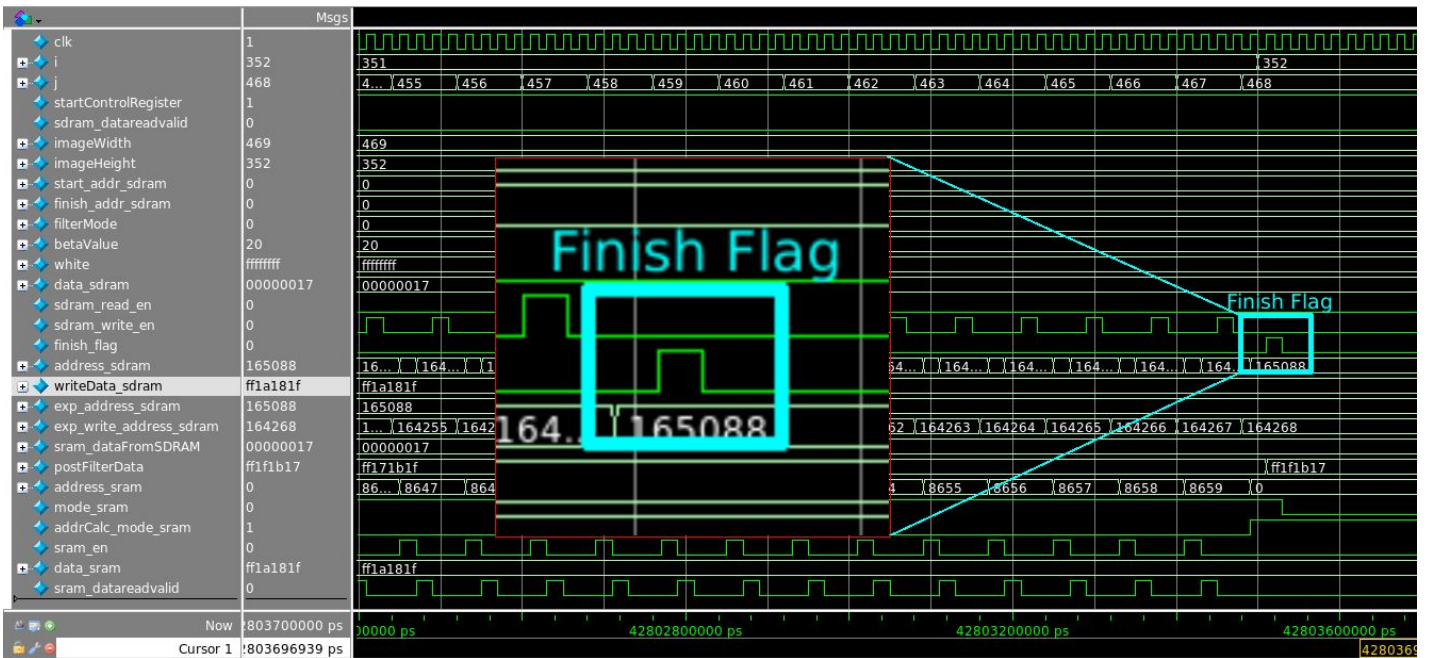


Figure 5. Finish flag at end of operations

Finally, we can see the finish flag pulsing high once all processing is complete

2.2.5. Timing waveform diagrams for operation of external devices

SDRAM

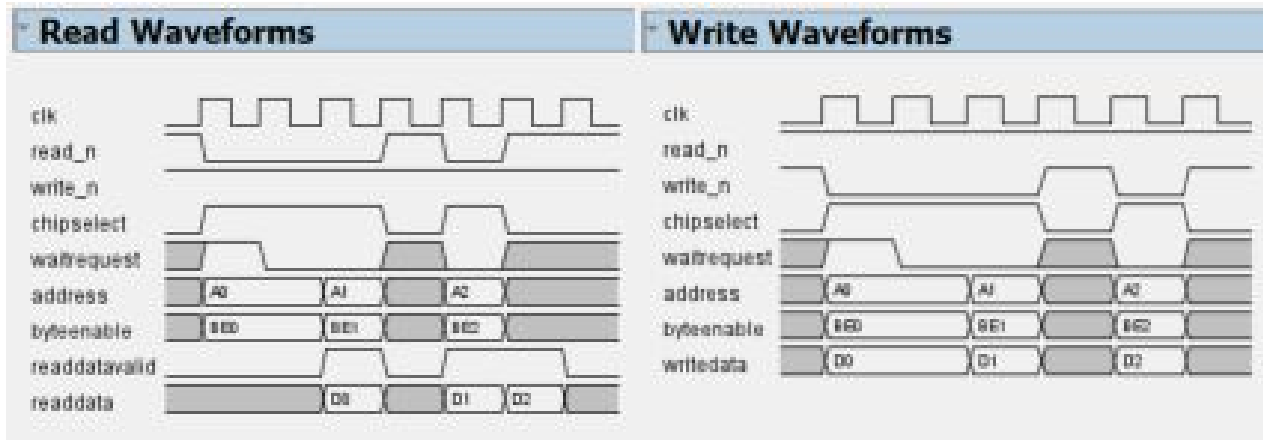


Figure 6. SDRAM Read/Write Waveforms

SRAM

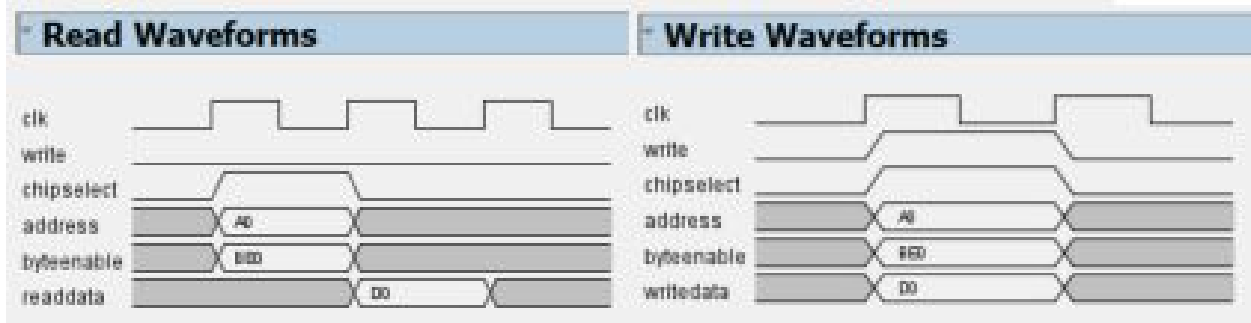


Figure 7. SRAM Read/Write Waveforms

PCIe

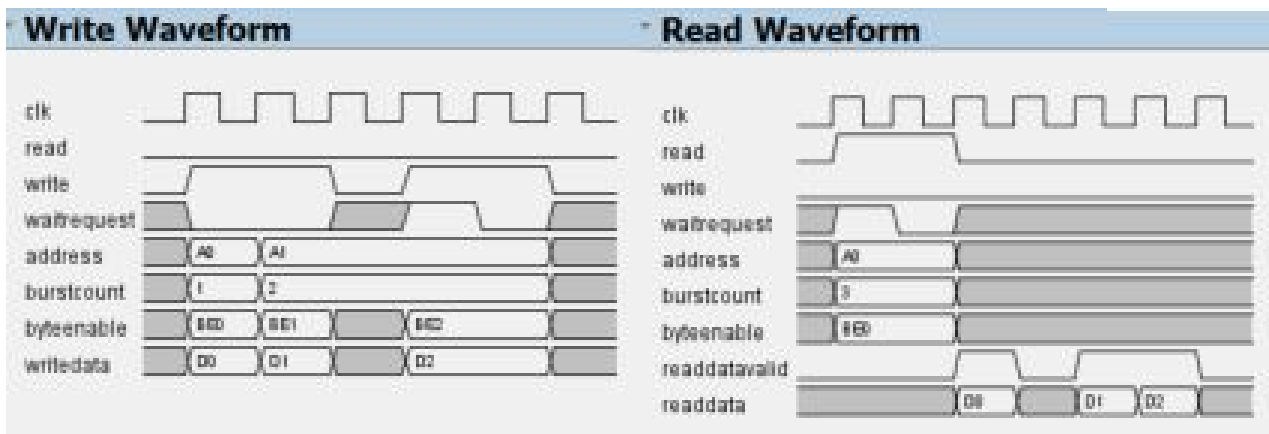


Figure 8. PCIe Write/Read Waveforms

Avalon Master

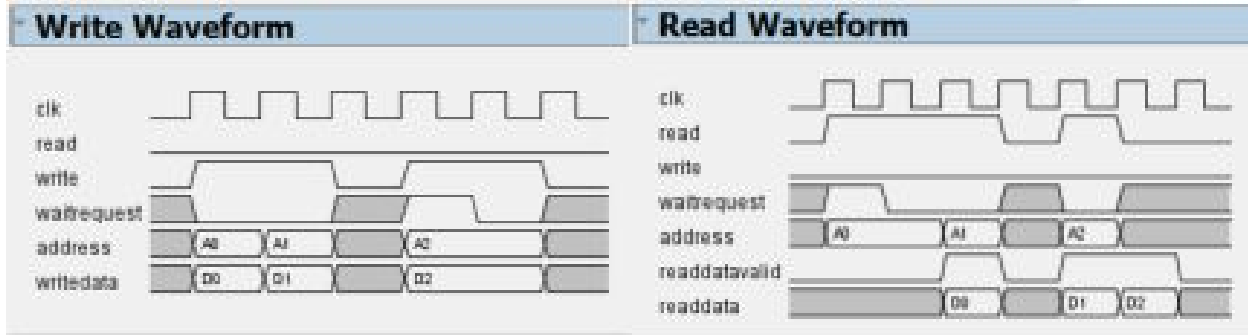


Figure 9. Avalon Master Write/Read Waveforms

Avalon Slave

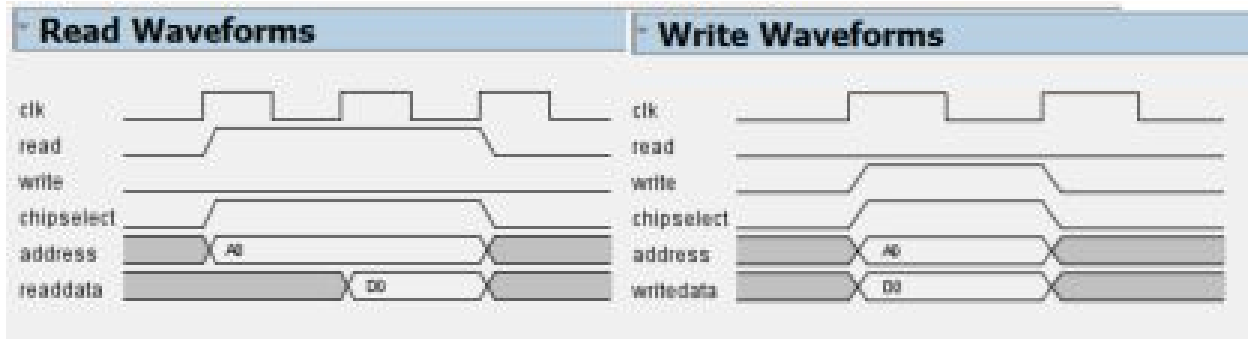
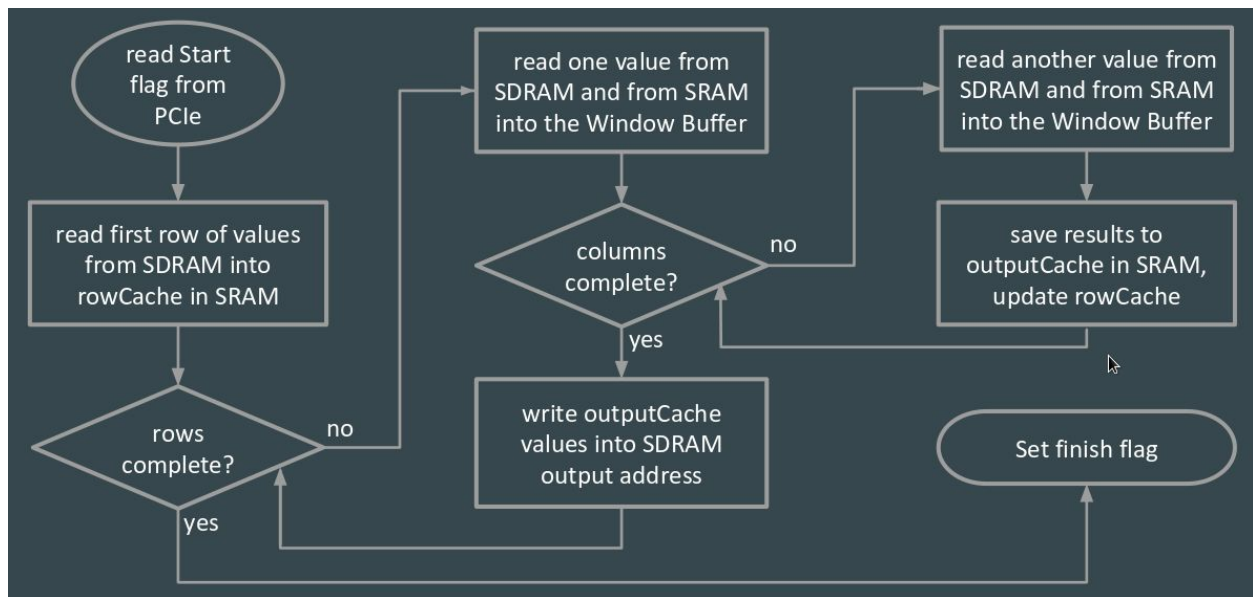


Figure 10. Avalon Slave Read/Write Waveforms

2.2.6. Flow chart of sequence of operations



2.3. Requirements

Since the chip is expected to be used in cameras, speed and data size management become key points of focus. Being placed in a camera implies that all conditions applicable to a camera become applicable to the chip as well. For instance, cameras should be operational in a wide variety of seasons and therefore perform in a wide range of temperatures. Temperature specifications will follow what is common in industry, ie. 0-40 deg Celsius.

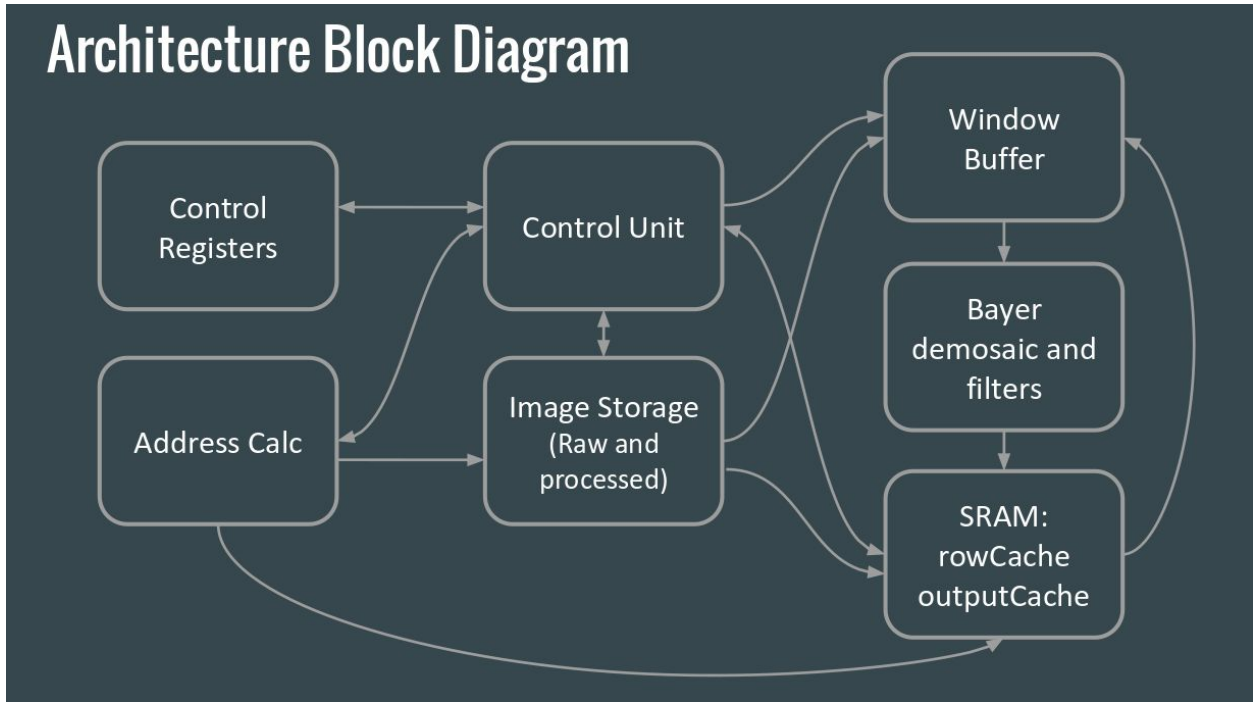
The objective of the ASIC implementation, as mentioned above, is to maximize the throughput of the chip, hence providing fast processing of the image intensity values. Since the camera could be using a large number of sensors, the chip must be able to deal with the large size of images as well.

To deal with the above requirements, a few targets have been set.

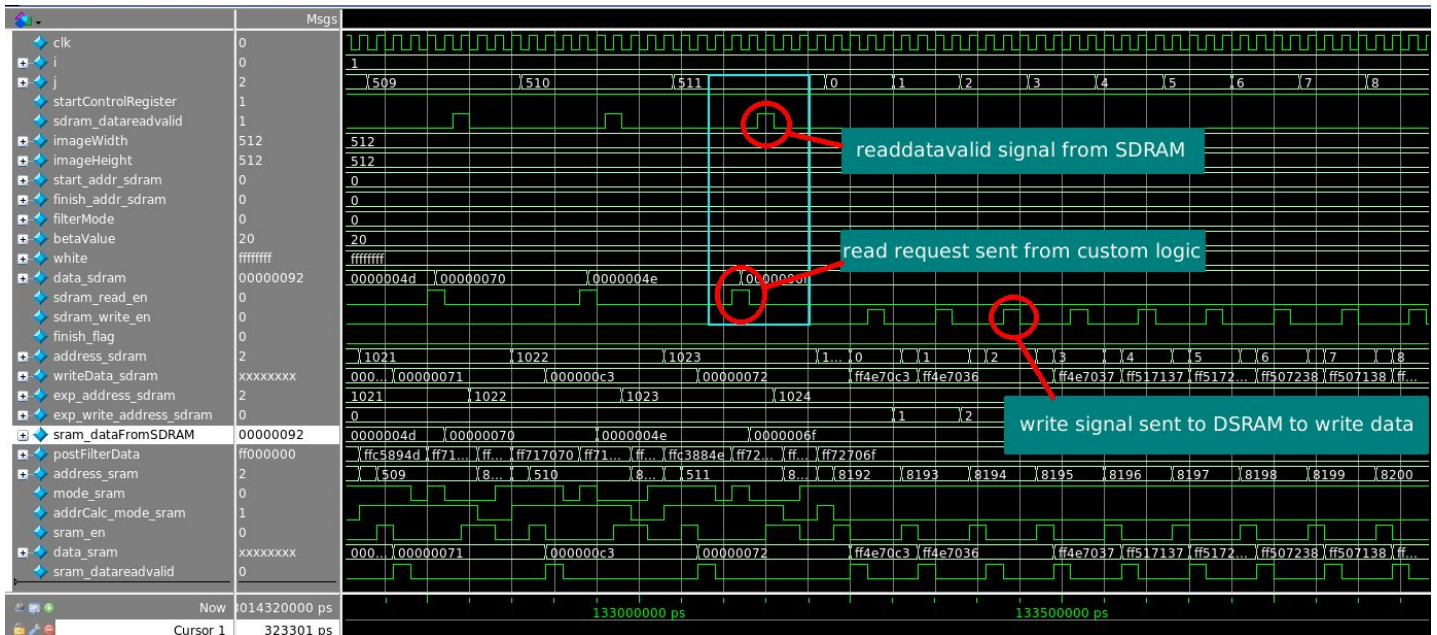
- 1.) Throughput : To process an image (~750KB, 3MB) in less that half a second.
- 2.) LookUp table count : To be able to fit on de2i-150 FPGA (less than 150,000 tables)

3. Final Design

3.1 Design Architecture

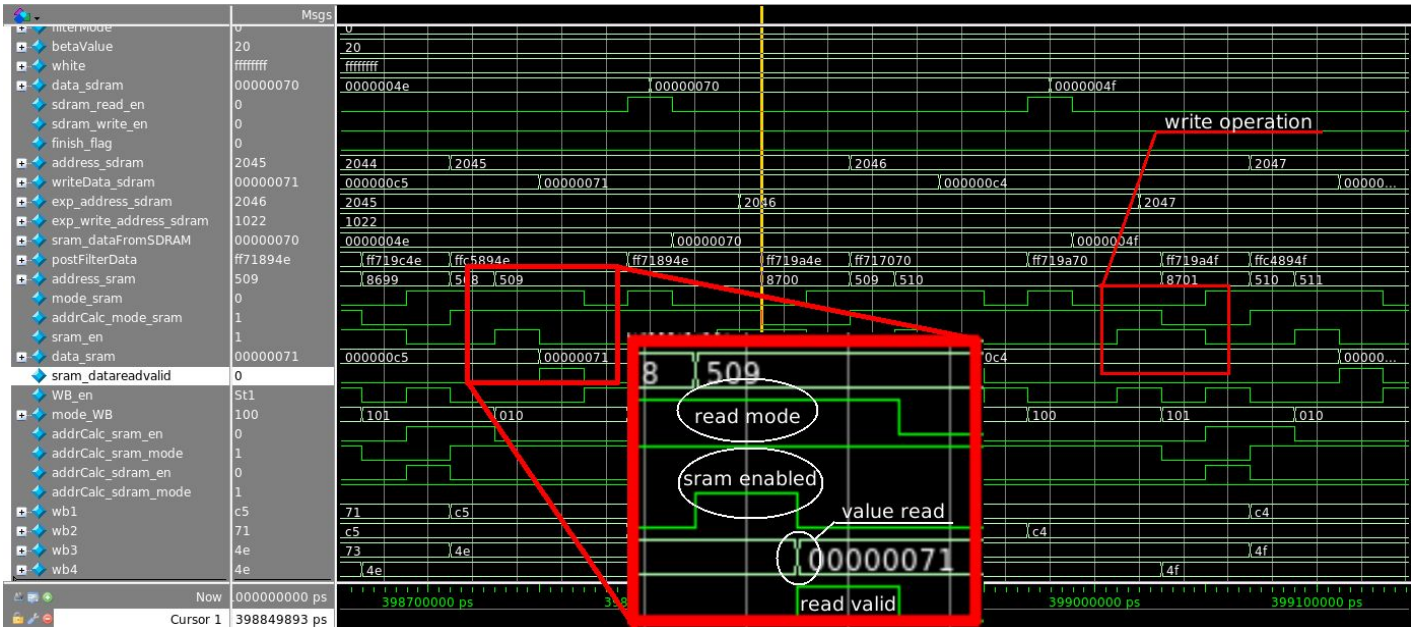


3.1.1. SDRAM read and write operations



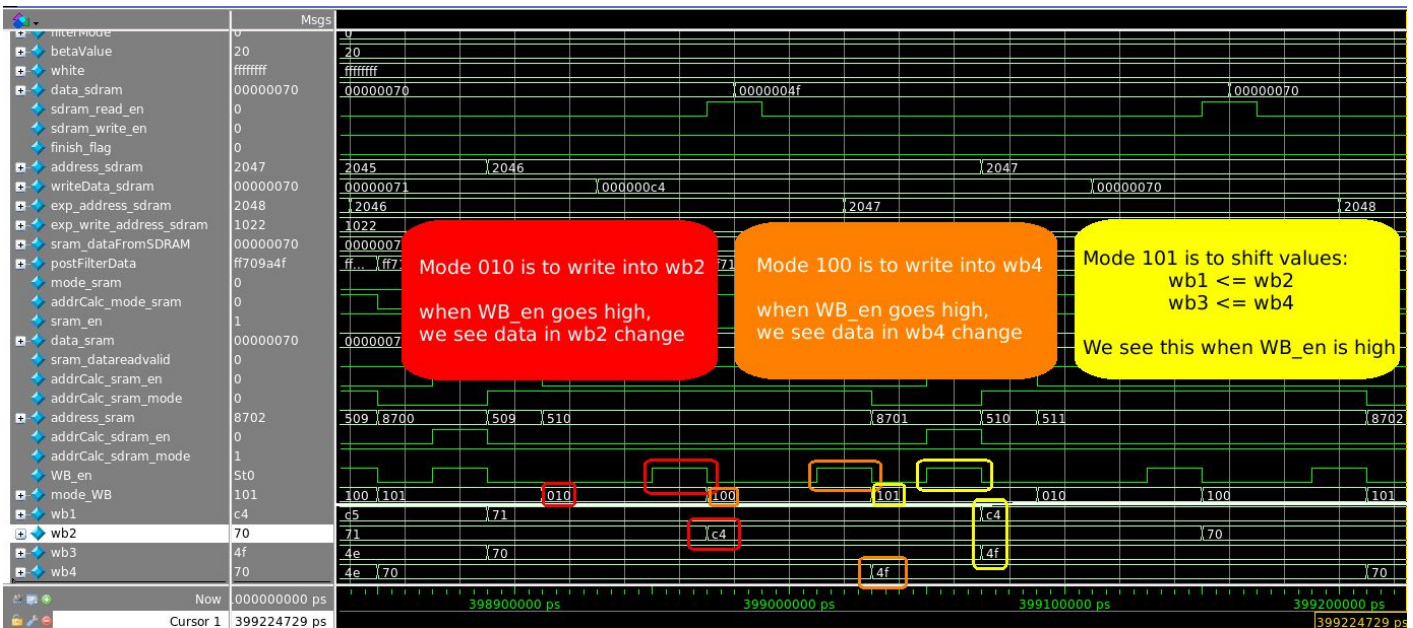
SDRAM read and write operations

3.1.2. SRAM read and write operations



SRAM read and write operations

3.1.3. Window Buffer operations



Window Buffer operations

3.1.4. Address calc update operations



Address calculator update operations

3.2. Functional Block Diagrams

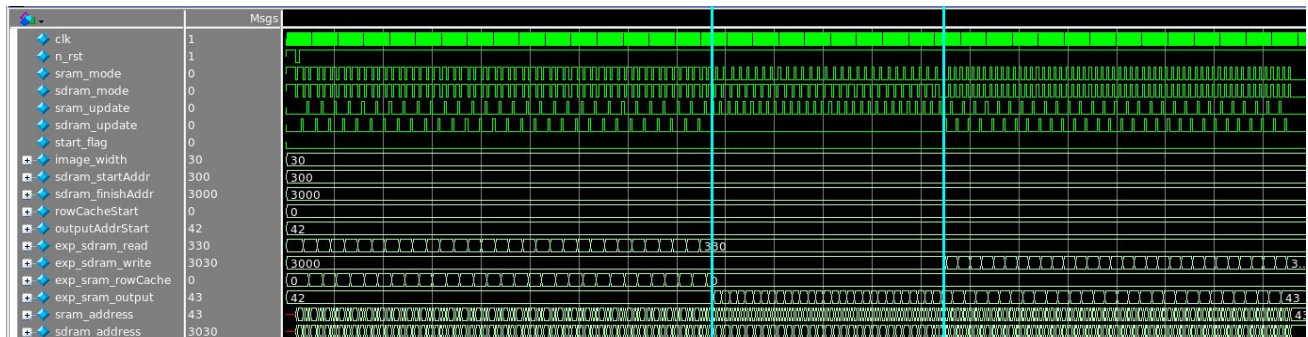
3.2.1. address_calc module

Used to keep track of sram, both rowCache and outputArr, and sdram addresses, both the address of the raw image and that of the output image.

It comprises of mappings to both the submodules.

I/O	Type	Size (bits)	Name
input	wire	1	clk
input	wire	1	n_rst
input	wire	1	sram_mode
input	wire	1	sdram_mode
input	wire	1	sram_update
input	wire	1	sdram_update
input	wire	1	start_flag
input	wire	13	image_width
input	wire	26	start_address_sdram
input	wire	26	finish_address_sdram
input	wire	26	rowCache_address_sram
input	wire	26	output_address_sram
output	reg	26	sdram_address
output	reg	26	sram_address

Relevant waveforms:



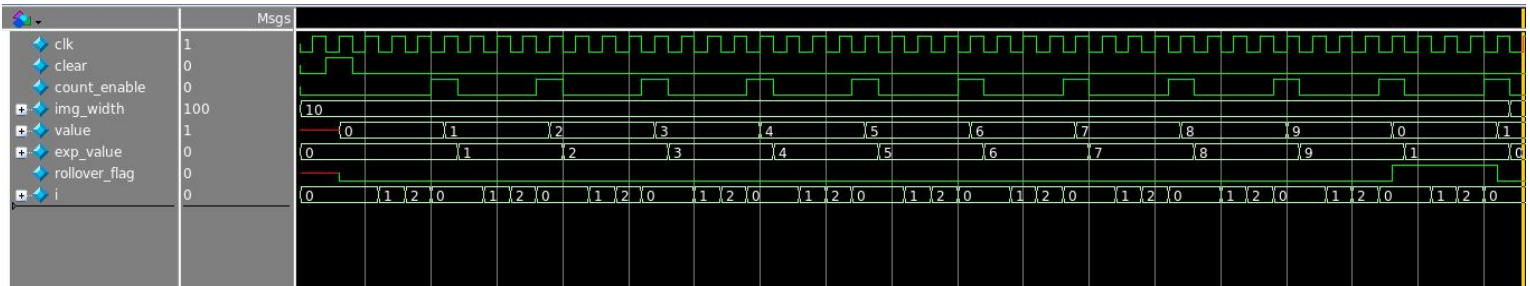
First Segment updates rowCache and output addr, second outputarr, third output sdram

3.2.1.1. i_col_counter module

Keeps track of the column index that is being accessed in the rowCache of sram.
With regards to circuitry, It is an up counter of 13 bits.

I/O	Type	Size (bits)	Name
input	wire	1	clk
input	wire	1	clear
input	wire	13	rollover_val
input	wire	1	count_enable
output	wire	1	rollover_flag
output	wire	13	value

Relevant waveforms:

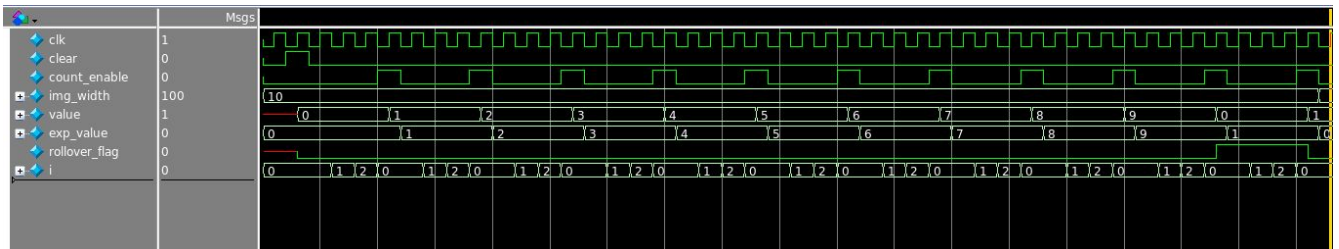


3.2.1.2. i_wr_counter module

Keeps track of the column index of output Array when writing pixel data to sdram.
With regards to circuitry, It is an up counter of 13 bits.

I/O	Type	Size (bits)	Name
input	wire	1	clk
input	wire	1	clear
input	wire	13	rollover_val
input	wire	1	count_enable
output	wire	1	rollover_flag
output	wire	13	value

Relevant waveforms:

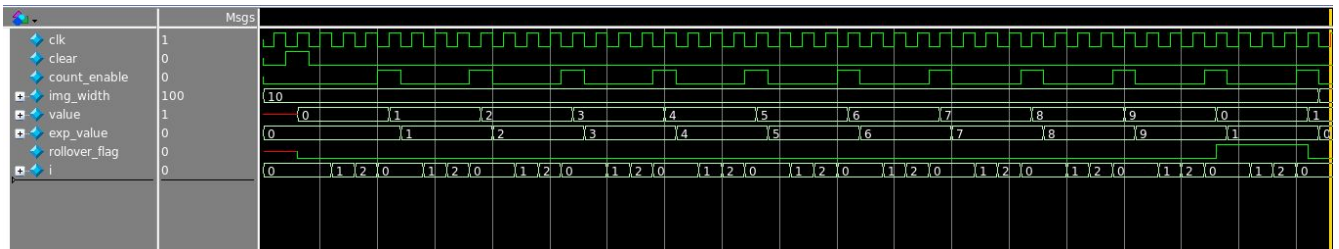


3.2.1.3. j_row_counter module

Keeps track of the row index of image. Rolls over when whole image is done being processed. With regards to circuitry, It is an up counter of 13 bits.

I/O	Type	Size (bits)	Name
input	wire	1	clk
input	wire	1	clear
input	wire	13	rollover_val
input	wire	1	count_enable
output	wire	1	rollover_flag
output	wire	13	value

Relevant waveforms:

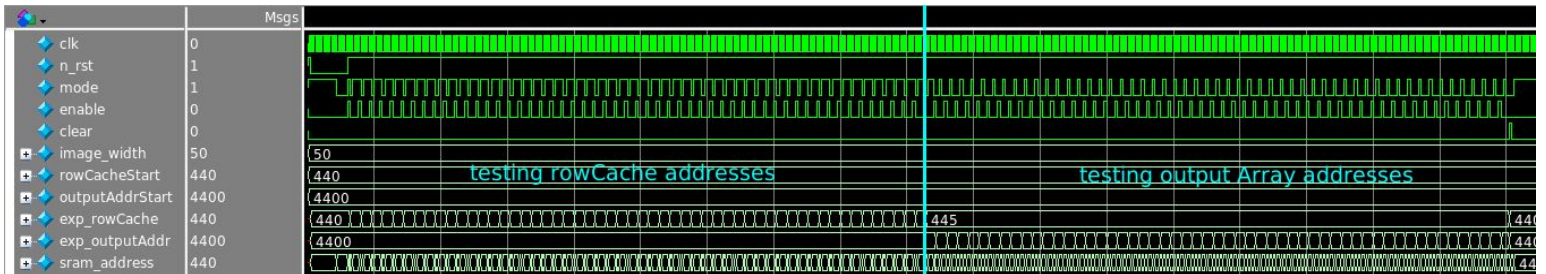


3.2.1.4. sram_address_calc module

Sub-module used to keep track of sram addresses, both the address of rowCache and the output Array. It comprises of a mux, and 2 flex counters.

I/O	Type	Size (bits)	Name
input	wire	1	clk
input	wire	1	n_rst
input	wire	1	load
input	wire	1	mode
input	wire	1	enable
input	wire	13	image_width
input	wire	26	sram_rowCacheStart
input	wire	26	sram_outputAddrStart
output	wire	26	sram_addr

Relevant waveforms:

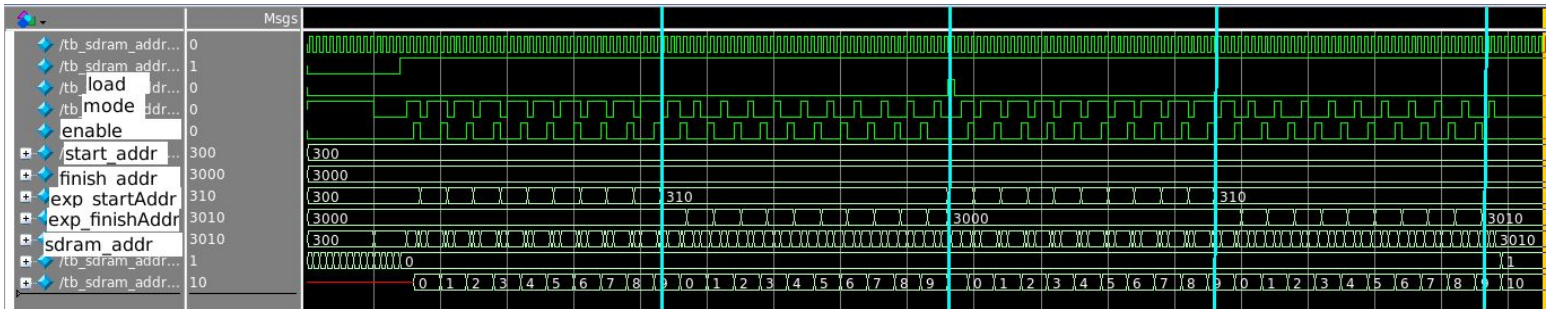


3.2.1.5. sdram_address_calc module

Sub-module used to keep track of sdram addresses, both the address of the raw image and that of the output image. It comprises of a mux, and 2 up counters that keep going up.

I/O	Type	Size (bits)	Name
input	wire	1	clk
input	wire	1	n_rst
input	wire	1	load
input	wire	1	mode
input	wire	1	enable
input	wire	26	start_address
input	wire	26	finish_address
output	wire	26	sdram_addr

Relevant waveforms:



First segment updates raw image address while the second segment updates address where image is output. This test is repeated again after a load(clear) operation.

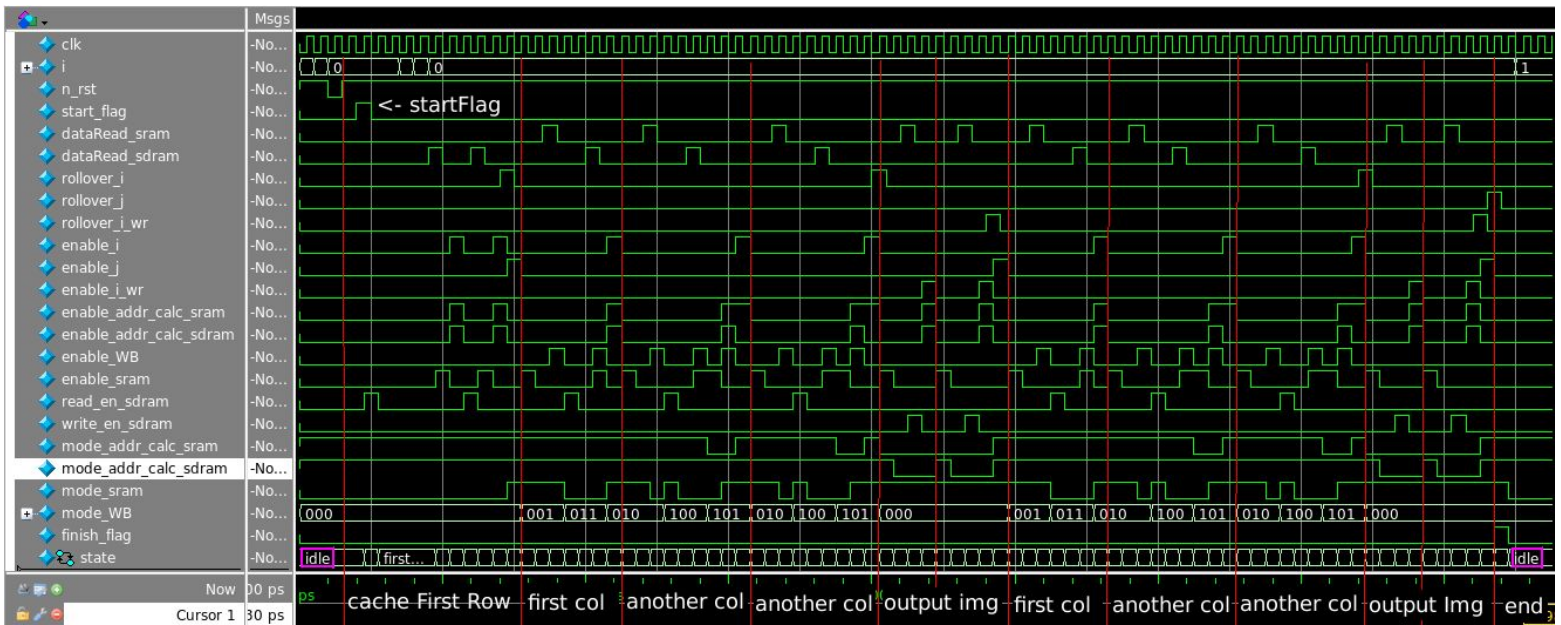
3.2.2. controlUnit module

This is the brain of the chip. It is responsible for sending read and write enables to sdr and sram, it sets window buffer mode and enables the same, it updates address calc and other counters and keeps track of number of operations complete. It is essentially a state machine.

I/O	Type	Size (bits)	Output
input	wire	1	clk
input	wire	1	n_rst
input	wire	1	start_flag
input	wire	1	dataRead_sram
input	wire	1	dataRead_sdr
input	wire	1	rollover_i
input	wire	1	rollover_j
input	wire	1	rollover_i_wr
output	reg	1	enable_i
output	reg	1	enable_j
output	reg	1	enable_i_wr
output	reg	1	enable_addr_calc_sram
output	reg	1	enable_addr_calc_sdr
output	reg	1	enable_WB
output	reg	1	enable_sram
output	reg	1	read_en_sdr
output	reg	1	write_en_sdr
output	reg	1	mode_addr_calc_sram
output	reg	1	mode_addr_calc_sdr

output	reg	3	mode_WB
output	reg	1	mode_sram
output	reg	1	finish_flag

Relevant waveforms:



The waveform is a demo of operations that would take place on an image. The image used here is a small 3x3 image since the states repeat. Each marked segment refers to a set of operations such as caching the first row, processing the first column, processing other columns and outputting the image to sdrum

3.2.3. filterTopLevel module

Top level module for all filters. It is used to pick the output of any one of the 4 filters that have been designed.

It is basically a mux with a lot of port mappings to smaller filter modules

I/O	Type	Size (Bits)	Name
input	wire	1	clk
input	wire	1	n_rst
input	wire	32	in
input	wire	2	filterMode
input	wire	8	brightnessCoeff
input	wire	3	wb_mode
input	wire	1	wb_en
input	wire	32	white
output	reg	32	result

Relevant waveforms:



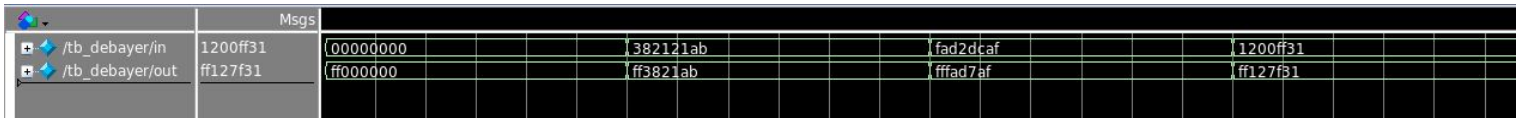
For a given input, the output can be seen for some of the opcodes like brightness addition (01) and demosaicing (00).

3.2.3.1. debayer module

SubModule to FilterTopLevel that takes in RGGGB values and returns an ARGB value. Circuit consists of an adder and a lot of rewiring (for division).

I/O	Type	Size (bits)	Name
input	wire	32	in
output	wire	32	out

Relevant Waveforms:



Msgs	
/tb_debayer/in	1200ff31
/tb_debayer/out	ff127f31

00000000	382121ab	fad2dcaf	1200ff31
ff000000	ff3821ab	fffad7af	ff127f31

Different outputs are checked for different inputs using assert statements

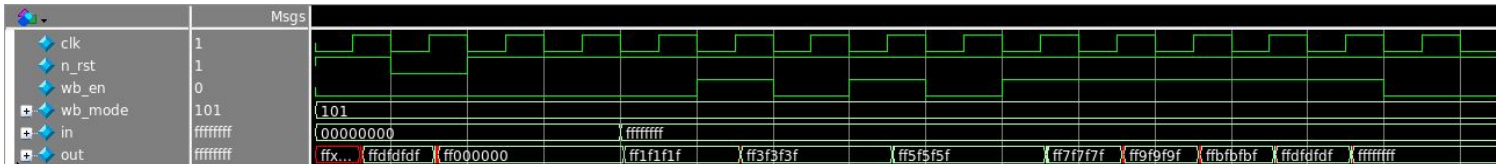
3.2.3.2. horBlur module (horizontal blur)

SubModule to FilterTopLevel that takes in ARGB values and carries out a horizontal blurring effect.

Circuit consists of 3 shift registers, adders and a lot of rewiring (for division).

I/O	Type	Size (bits)	Name
input	wire	1	clk
input	wire	1	n_rst
input	wire	1	wb_en
input	wire	3	mode_wb
input	wire	32	data
output	reg	32	blur

Relevant Waveforms:



We see that only when mode is 101 and enable is high does the shift take place. Upon shift taking place we notice the output values updating.

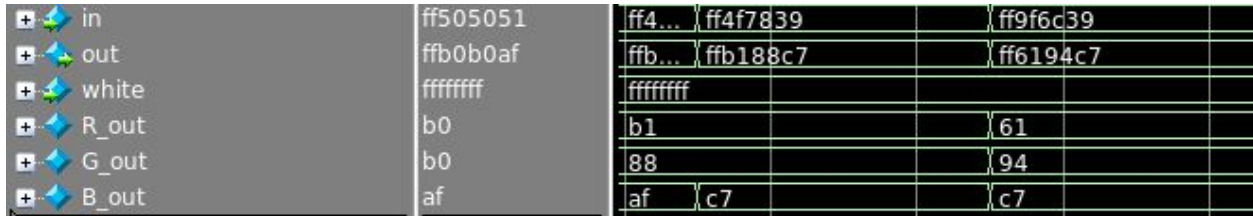
3.2.3.3. whiteBalance module

SubModule to FilterTopLevel that takes in ARGB values and scales values based on the white provided.

Circuit consists of lookup tables for GF(2⁸) division using multiplicative inverses and multipliers to carry out the said division.

I/O	Type	Size (bits)	Name
input	wire	32	in
input	wire	32	white
output	wire	32	out

Relevant waveforms:



Here, the white balancing is set to a value of 24'hFFFFFF.

We can see the input come in and the output change accordingly.

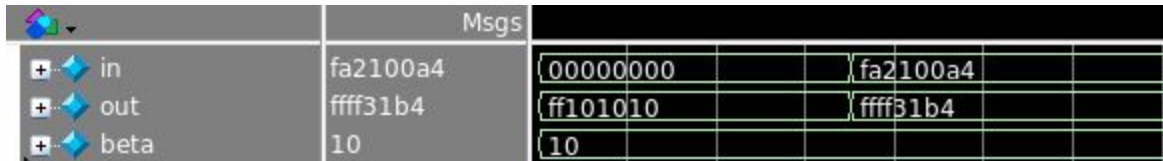
R_out, G_out, B_out are concatenated along with the alpha value to give the output.

3.2.3.4. brightnessFilter module

SubModule to FilterTopLevel that takes in beta value and adds brightness accordingly. Consists of an adder and division by rewiring.

I/O	Type	Size (bits)	Name
input	wire	32	in
input	wire	8	beta
output	wire	32	result

Relevant waveforms:



	Msgs						
+	in	fa2100a4	00000000	fa2100a4			
+	out	fff31b4	ff101010	fff31b4			
+	beta	10	10				

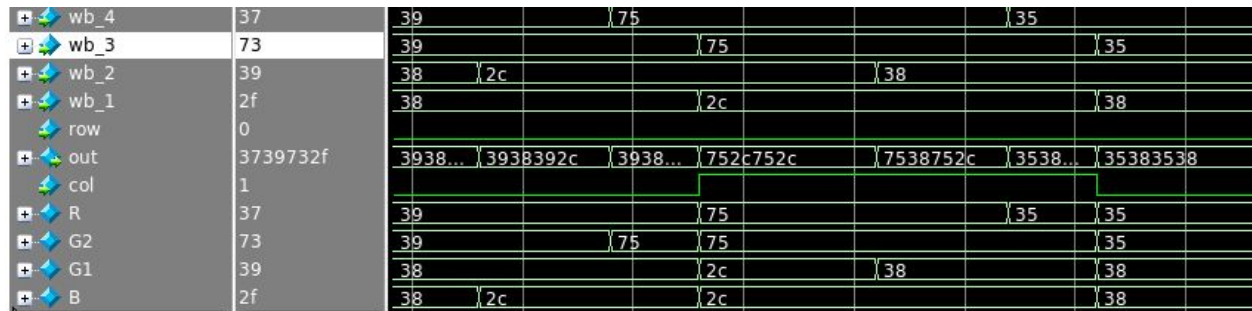
Here we see that for a given input and beta value of 10, the output is appropriately scaled up. To note that in this test bench the input is formatted as RGYB where Y is disregarded and alpha is considered FF.

3.2.4. rggb module (re-ordering module)

The rggb reordering module is essentially a multiplexer that receives 4 different arrangements of pixel intensity values, and based on the row and column index values as selection lines, it arranges them to always throw out the pixels in RRGB pattern.

I/O	Type	Size(bits)	Name
input	wire	8	wb_1
input	wire	8	wb_2
input	wire	8	wb_3
input	wire	8	wb_4
input	wire	1	row
input	wire	1	col
output	reg	32	out

Relevant waveforms:



Here, wb_1..4 and row,col are the inputs and we see how changing any of them immediately result in out changing values.

3.2.5. wBuffer module (window buffer)

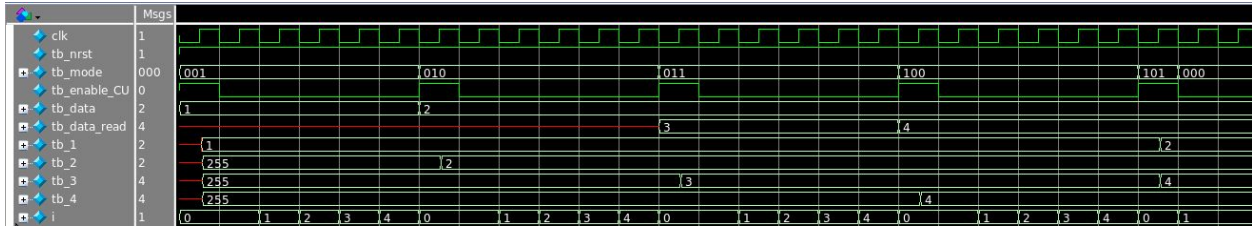
The window buffer performs reads from the SRAM and SDRAM and writes back onto the SDRAM after performing the operations of reads and writes based on the opcode sent to it by the control unit.

It contains 4 registers with next state logic that replace each register with new values provided some set conditions are met.

I/O	Type	Size(bits)	Name
input	wire		clk
input	wire	8	nrst
input	wire	8	enable_CU
input	wire	3	mode
input	wire	8	data_read
input	wire	8	data
output	reg	8	w_1
output	reg	8	w_2
output	reg	8	w_3
output	reg	8	w_4

OPCODE	OPERATION
001	READ FROM SRAM INTO WB_1
010	READ FROM SRAM INTO WB_2
011	READ FROM SDRAM INTO WB_3
100	READ FROM SDRAM INTO WB_4
101	SHIFT VALUES FROM 4->3 AND 2->1

Relevant waveforms:



We see that for each opcode (defined in the table above), the expected operation takes place. The result of each opcode can be verified by looking at tb_1..4 which contain values within each of the 4 window buffer registers.

3.3. Standards and Protocols

3.3.1. PCIE

The interfaces choices we had, to work with our custom logic were fixed due to us implementing our design on an FPGA.

The biggest benefit of using the PCI express architecture is that it facilitates or provides independent high speed serial data transfer lines with dedicated bus bandwidths.

The PCI express 1 on the FPGA is also scalable depending on the desired throughput, it has a maximum throughput of 4GB/s.

The PCIe is based on point to point topology with separate serial links between different devices and the root complex, in our project the custom logic chip is one of the endpoints to the root complex.

Since the PCIe is already integrated in the FPGA we did not have an opportunity to reduce the area by using other standards and protocols.

The PCIe 1 has a data transmission rate of 2Gbit/s per lane and allows 16 lanes(x16) to be used.

The PCIe for our design acts a master to both the custom logic chip as well as the SDRAM controller.

PCIe is used to write the raw image onto the SDRAM and sets the control registers on the custom logic.

3.3.2. SDRAM

The SDRAM is organized as a matrix of cells and interfaces with the SDRAM controller to communicate with the custom logic.

The SDRAM consists of a certain number of address bits dedicated to column addressing(CAS) and a number of address bits dedicated to row addressing(RAS) which makes it easier for us to read and write pixel values.

The memory location is referenced by placing address on address lines.

The SDRAM controller acts as a slave to both the custom logic chip and the PCIe.

The SDRAM in our design is used to store the pixel intensity values and these values are read off from the SDRAM by the custom logic to perform the necessary reordering, and filter operations. This was necessary as the FPGA wouldn't have enough registers to store data for high resolution images.

The resultant processed pixel values from debayering, filters, etc are written back onto the SDRAM.

3.3.3. SRAM

SRAM was selected cache image row data to prevent resampling pixel data each time the window buffer moves to debayer the next pixel and prevent CAS / RAS penalties.

The SRAM was a better decision than reading and writing off the SDRAM since it would save exactly 3 reads from each SDRAM address at the cost of a single read from the SRAM.

It also allowed the data to be read and written into both SDRAM and SRAM in a contiguous fashion, that was not possible with out prior design that used only the SDRAM.

3.4. Timing and Area Budgets

Due to implementation on the FPGA, timing and area budgets were set to be under system specifications of the Terasic DE2i-150 FPGA Board as referenced in Table 1, Appendix A.

4. Verification

4.1. Verification plan and corresponding results

4.1.1. Detailed Verification Test Breakouts

Correctness of whole program on FPGA

- Shown in Demo: Incomplete
- DSSC(s) Proved: 4 and 5
- Highest Level of Design Module Involved:
 - Total Design/Chip
- Test Bench Expectations/Requirements
 - To obtain valid RGB image from provided RAW image
 - To obtain well filtered images if certain filter is selected
- Compare output image of chip with same image processed on a separate script/program
- Pre/Post Processing needed
 - Create expected output using other script/program
 - Load raw image onto sdram
 - Read and check final image from sdram
- Main Verification Test Steps:
 - Write program on intel chip to interface with PCIe
 - Have raw image written onto sdram
 - Have control registers set
 - wait till control registers provide finished flag
 - read processed image from sdram
 - compare processed image with expected output

Correctness of custom logic using simulation

- Shown in Demo: **Shown**
- DSSC(s) Proved: 1 and 2
- Highest Level of Design Module Involved:
 - Total Design/Custom Logic
- Test Bench Expectations/Requirements
 - To obtain valid RGB image from provided RAW image

- To obtain well filtered images if certain filter is selected
- Compare output image of chip with same image processed on a separate script/program
- Pre/Post Processing needed
 - Create expected output using other script/program
 - Ready image to be read so that SDRAM may be simulated
 - Compare saved processed image with the expected one
- Main Verification Test Steps:
 - Set control registers to initialize custom logic to start running
 - provide image data on sdram read call
 - save to output image on sdram write call
 - while control registers provides finished flag do above
 - compare processed image with expected output

Correctness of window buffer transitions

- Shown in Demo: No
- DSSC(s) Proved: 1
- Highest Level of Design Modules(s) involved:
 - Window Buffer
- Test bench Expectations/Requirements:
 - Have temporary data registers to store the block results and compare them with expected results.
 - Perform operations and have the expected pixel values in respective positions in window buffer and check with block results.
- No external references are needed
- No pre/post processing is needed
- Main Verification Test Steps:
 - Check each window of the window buffer to see if the expected pixel value is present by comparing with test buffer.
 - Keep track of each transition and operation happening in the buffer using waveforms behavior.

Correctness of image filters

- Shown in Demo: **Yes**
- DSSC(s) Proved: 2
- Highest Level of Design Module(s) involved:
 - Image Filters
- Test bench Expectations/Requirements:
 - Check image produced after filter and compare with respective Averaged value/math algorithm results depending on selection of filter
- No pre/post processing is needed
- Main Verification Test Steps:
 - Receive the test vectors for each case and perform the math
 - Compare the calculated output pixel from the math algorithm for each filter with expected outputs
 - Check pixel by pixel values with expected output to verify result

4.1.2. Backup and Sub-Module Tests

Correctness of RGGB box type select module

- Shown in Demo: No
- DSSC(s) Proved: 1
- Highest Level of Design Module(s) involved:
 - DeBayer and filter
- Test bench Expectations/Requirements:
 - Have test vectors of sample RGGB intensity values that are reordered and compare them block outputs
- No external or premade references needed
- No pre/post processing is needed
- Main Verification Test Steps:
 - Generate 4 bytes consisting of pixels in random arrangement and provide as input to block
 - Have expected order of pixels in RGGB arrangement and compare
 - Repeat above steps for different pixel groups of 4

Correctness of Address Calculator (Address Calculator Block)

- Shown in Demo: No
- DSSC(s) Proved: None
- Highest Level of Design Module Involved:
 - Address Calculator Block
- Test Bench Expectations/Requirements
 - To generate correct read and write addresses
 - To update read and write addresses correctly
 - To check for row and column roll overs
- Manually calculate expected addresses and match with obtained results
- No pre/post processing is needed
- Main Verification Test Steps:
 - Reset all sub-blocks in the address calculator
 - Provide start address for sram and sdram
 - Provide enable to update information in the block
 - Check for correct rollover values
 - Check if addresses output are correctly updated

Correctness of Control Unit (Control Unit Block)

- Shown in Demo: No (shown in final presentation)
- DSSC(s) Proved: None
- Highest Level of Design Module Involved:
 - Control Unit Block
- Test Bench Expectations/Requirements
 - To correctly traverse through all the states
 - To ensure correct output is returned at each state
 - To ensure states change as expected when input changes
- Manually calculate expected output based on provided input
- No pre/post processing is needed
- Main Verification Test Steps:
 - Reset Control Unit
 - Provide inputs and see if output matches corresponding state

Correctness of PCIe interface (PCIe express endpoint)

- Shown in Demo: Incomplete
- DSSC(s) Proved: 3 and 4
- Highest Level of Design Module Involved:
 - FPGA interfaces
- Test Bench Expectations/Requirements
 - To write raw image data onto sdram
 - To read processed image data from sdram
 - To set and get control register data
- Manually calculate expected values and match with obtained values
- No pre/post processing is needed
- Main Verification Test Steps:
 - Program intel cpu to interface with sdram
 - Write data onto sdram
 - Read same data from sdram and compare with data written
 - Write data to control register
 - Read same data from control register to see if successfully read/written

SDRAM Protocol Interfacing Test

- Shown in Demo: Incomplete
- DSSC(s) Proved: 3
- Highest Level of Design Module Involved:
 - Image Storage Block
- Test Bench Expectations/Requirements
 - To correctly receive data from PCIe
 - To correctly receive data from SRAM
 - To correctly output data to SRAM
 - To correctly output data t
- Manually calculate expected output based on provided input
- No pre/post processing is needed
- Main Verification Test Steps:
 - Write a test image to SDRAM via PCIe

- Write a test image from SDRAM to PCIe
- Visual inspection of image / hash check

SRAM Protocol Interfacing Test

- Shown in Demo: No (On simulation)
- DSSC(s) Proved: None
- Highest Level of Design Module Involved:
 - Window Buffer, Read Buffer, Write Buffer
- Test Bench Expectations/Requirements
 - To successfully read the correct row/data from SDRAM
 - To successfully write data out to SDRAM
- Manually calculate expected output based on provided input
- No pre/post processing is needed
- Main Verification Test Steps:
 - Write a test stream to SRAM from SDRAM
 - Read the test stream from SRAM
 - Compare read stream with test stream

4.2. Steps taken to verify each block at each level of hierarchy

Almost each block had a corresponding test bench written for it (Only 2 trivial combination blocks were skipped).

To ensure correctness of modules, even underlying blocks were provided with test benches that ran successfully on both source and mapped simulations.

Top level block test benches (this includes address calculator, control unit, window buffer and customLogicTopLevel) were written such that the test bench was to simulate actual operations.

It is to be noted that some test benches might throw assert errors since minor changes were made and these changes were not reflected in their test benches due to a lack of time. The modules do work though based on top level test benches providing expected outputs.

5. FPGA Resource Usage

Minimum Usage Results:

Family	Cyclone IV GX
Device	EP4CGX150DF31C7
Timing Models	Final
Total logic elements	12,198 / 149,760 (8 %)
Total combinational functions	9,661 / 149,760 (6 %)
Dedicated logic registers	8,371 / 149,760 (6 %)
Total registers	8489
Total pins	171 / 508 (34 %)
Total virtual pins	0
Total memory bits	157,416 / 6,635,520 (2 %)
Embedded Multiplier 9-bit elements	0 / 720 (0 %)
Total GXB Receiver Channel PCS	1 / 8 (13 %)
Total GXB Receiver Channel PMA	1 / 8 (13 %)
Total GXB Transmitter Channel PCS	1 / 8 (13 %)
Total GXB Transmitter Channel PMA	1 / 8 (13 %)
Total PLLs	2 / 8 (25 %)

6. Results

1. (2 points) Test benches exist for all top level components and the entire design. The test benches for the entire design can be demonstrated or documented to cover all of the functional requirements given in the design specific success criteria. : **COMPLETE**
2. (4 points) Entire design synthesizes completely, without any inferred latches, timing arcs, and, sensitivity list warnings. **COMPLETE (sram_simulation block not part of design)**
3. (2 points) Source and mapped version of the complete design behave the same for all test cases. The mapped version simulates without timing errors except at time zero. **PARTIALLY COMPLETE (95% complete) (top level mapped simulation untested since sram_simulation takes too long to run. All submodules run as expected in source and mapped)**
4. (2 points) A complete IC layout is produced that passes all geometry and connectivity checks.
ECE337 Full Project Proposal Guidelines Fall 2015
5. (2 points) The entire design complies with targets for area, pin count, throughput (if applicable), and clock rate. The final targets for these parameters will be determined by course staff based on your design review. Failure to reach any of the targets will result a score of 1 out of 2 provided that you are within 50% on area, 10% on pin count, and 25% on throughput. Doing worse in any category will result in a score of 0 out of 2.

(2.5 points) To demonstrate the working of bayer demosaicing using system verilog simulations : **PARTIALLY COMPLETE (Image renders but color is off)**

(1.5 points) - To demonstrate the working of filters using system verilog simulations.

Filters used are:

1. Horizontal Averaging Blur : **COMPLETE**
2. Color White Balancing : **PARTIALLY COMPLETE (90%)(works but values rollover beyond a certain number to give inverted colors)**
3. Contrast and Brightness changes : **COMPLETE**

FPGA:

(1 points) - To demonstrate that the CPU can write raw image data to the SDRAM and read processed image from SDRAM using the PCIe on FPGA : **INCOMPLETE**

(1 points) - To demonstrate that the CPU can modify the custom logic control registers and take action based on their values using the PCIe on FPGA : **PARTIALLY COMPLETE (Modification to demo program is running. Data is being read and written to SDRAM)**

(2 points) - To demonstrate complete working of bayer demosaicing and filter application on FPGA : **INCOMPLETE**

7. Appendix A

Data Sheets and Guide to Design Data

mg65/ECE337/Project

/Presentations

337_designreview.ppt

337 Final Presentation.pdf

/FPGA

app.c

/Project

Contains many ppm files, including results to images

/analyzed

/createCFA

convert.cpp

/testImages

Contains original test images (not raw) in ppm format

/docs

/mapped

generated verilog files

/reports

generated reports

/schematic

/scripts

gfGen.py (Script to generate multiplicative inv on $GF(2^8)$)

/source

address_calc.sv

risingEdgeDetector.sv

tb_horblur.sv

brightnessFilter.sv

sdram_address_calc.sv

tb_i_col_counter.sv

controlUnit.sv

sram_address_calc.sv

tb_i_wr_counter.sv

customLogicTLD.sv

sram_simulation.sv

tb_j_row_counter.sv

debayer.sv

tb_address_calc.sv
tb_risingEdgeDetector.sv
delaySingleClock.sv
tb_brightnessFilter.sv
tb_sdram_address_calc.sv
filterTopLevel.sv
tb_controlUnit.sv
tb_sram_address_calc.sv
flex_counter.sv
tb_customLogicTLD.sv
tb_sram_simulation.sv
horblur.sv
tb_customLogicTLD.sv.bak
tb_wbuffer.sv
i_col_counter.sv
tb_debayer.sv
transcript
i_wr_counter.sv
tb_delaySingleClock.sv
wbuffer.sv
j_row_counter.sv
tb_filterTopLevel.sv
whiteBalance.sv
rggb.sv
tb_flex_counter.sv

Table 1



Following is more detailed information about the blocks in Figure 1-3:

FPGA device

- Cyclone IV EP4CGX150DF31 device
- 149,760 LEs
- 720 M9K memory blocks
- 6,480 Kbits embedded memory
- 8 PLLs

FPGA configuration

- JTAG and AS mode configuration
- EPCS64 serial configuration device
- On-board USB Blaster circuitry

Memory devices

- 128MB (32Mx32bit) SDRAM
- 4MB (1Mx32) SSRAM
- 64MB (4Mx16) Flash with 16-bit mode

SD Card socket

- Provides SPI and 4-bit SD mode for SD Card access

Connectors

- Ethernet 10/100/1000 Mbps ports
- High Speed Mezzanine Card (HSMC)
- 40-pin expansion port
- VGA-out connector
- VGA DAC (high speed triple DACs)
- DB9 serial connector for RS-232 port with flow control

Clock

- Three 50MHz oscillator clock inputs
- SMA connectors (external clock input/output)

Display

- 16x2 LCD module

8. Appendix B

Simulation Results

TABLE OF CONTENTS:

- 8.1. General Briefing
- 8.2. Expected throughputs
- 8.3. Output 1 (Image: Lena)
- 8.4. Output 2 (Image: Tiger)

8.1. General Briefing

Please note that all waveforms from the top level diagram and from each blocks' individual test bench has already been attached to the document in previous sections.

We kindly request you to refer to those as and when required.

With regards to simulation results:

Although great progress was made with the FPGA, no conclusive results were obtained from it.

The simulations on questaSim were much more successful and were managed to obtain results for various images.

The results from a couple of the many images we tested our custom logic on are shown in sections 8.3. and 8.4.

8.2. Expected Throughputs

We had set expectations for our chip to run in 1 second for images that are about 5MB in size.

From results of "lena" and "tiger" images we got the following:

Image	Size of Image	Time taken
lena	~750KB	0.068 sec
tiger	~2.9MB	0.266 sec

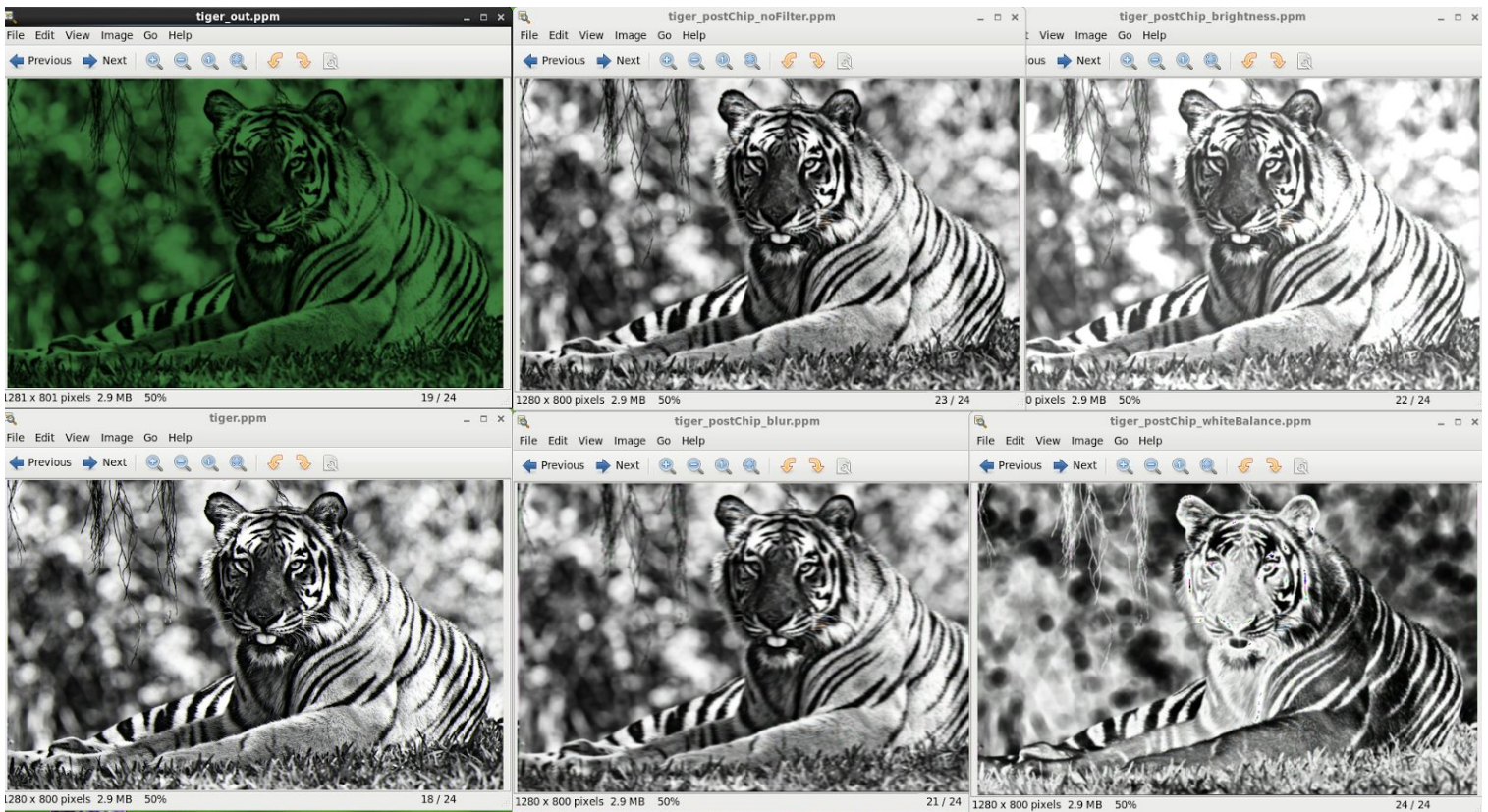
8.3. Output 1 (Image: Lena)



- (a) RAW input image that contains camera intensities
- (b) output after bayer demosaicing (no filters applied)
- (c) output after application of brightness filter
- (d) expected output from bayer demosaicing (no filters applied)
- (e) output after application of horizontal blur filter
- (f) output after application of white balancing (note: white balanced values have rolled over, hence inverted)

We notice that there is a stream of green and purple that cuts through the result images which is not a part of the expected output. After great debugging, the issue has been narrowed down to either an issue with the raw *_out_fpga file (we generated the same using a homebrew C program to contain intensity values as integers. It is not the one shown above). The other possibility is an issue with our rrgb re-ordering module. Nevertheless, we are glad to observe that besides the color, the image has not come out looking distorted or unrecognizable.

8.4. Output 2 (Image: Tiger)



- (a) {top left} RAW input image that contains camera intensities
- (b) {top mid} output after bayer demosaicing (no filters applied)
- (c) {top right} output after application of brightness filter
- (d) {btm left} expected output from bayer demosaicing (no filters applied)
- (e) {btm mid} output after application of horizontal blur filter
- (f) {btm right} output after application of white balancing (note: white balanced values have rolled over, hence inverted)

We notice that the output for a grayscale image such as the one above matches exactly what was expected. The debayered images look as expected and the filters too look like they are working correctly when applied to a grayscale image.

NOTE: The RAW input images are meant to have a green shade to them since cameras have twice the number of green sensors as they do red and blue.