Indian Institute of Technology Delhi

Department of Computer Science

COL-215
Hardware Assignment Report - 2

Dinu Goyal - 2022CS51647
Spandan Kukade - 2022CS51138

November 15, 2023

Contents

1 Introduction

The third hardware assignment for COL215 focuses on the implementation of a 3x3 image filtering operation, extending the concepts covered in the previous hardware assignment. The primary components involved in this task include memory elements (RAM, ROM, and registers), a compute unit responsible for the filter operation, and a VGA controller. The main part of this assignment focuses on a FSM that is a finite state machine using which gradient operations are performed on the 3 given input images.

2 Problem Description

The assignment requires the implementation of a 3x3 image filtering operation on a 64x64 image using a provided 3x3 image kernel. The goal is to perform the filtering operation and display the resultant image. The input image, stored in a COE file as an 8-bit binary unsigned format, is to be stored in a 1-D array format in block RAM. The kernel, also provided via the COE file, may contain negative values in 2's complement form.

3 Logic and Algorithms

The logic to apply gradient algorithms is: The code follows a sequence of operations to process data. Initially, it propagates the values of temporary variables forward. Subsequently, it retrieves data from the romaddress and assigns it to the last variable. Next, it calculates an answer and applies a filter to it.

To ensure robust operation, the code includes checks for edge cases and determines which part of the RAM is being read to avoid invalid reads. Within the filter section, it examines the calculated values to ensure they remain within the required range.

Finally, the processed data is directed to the dram component. The code also handles memory address indexing, incrementing the i and j variables to move to the next RAM address. Order of algorithm : The code implemented reads the kernel coe file once and stores all the variables, Then it reads the ROM and calculates the minimum and maximum values for further calculation, Finally it reads and processes the rom once more to assign final values after normalizing using the formula :

$$New\_I(i,j) = (I(i,j) - min) * \frac{255 - 0}{max - min} + 0$$

Figure 1

4 Optimization

For the optimization, we
1. Maintained the 9 variables to store the data read from kernel so that we may not need to read the kernel again and again.
2. Used shifting operations to propagate variables, so that variables can be reutilized and we may not need to recalculate the variables from rom for each pixel value.
3. At the time of final reading, our algorithm reads one value of rom at most thrice in the worst case.

5 Implementation

In this section, we delve into the intricacies of the VGA (Video Graphics Array) controller implementation, which forms the heart of our visual display system. A VGA controller is pivotal in converting digital data into a visual output on a screen, with the ability to control pixel colors, synchronization signals, and display timings.
The subsequent subsections will dissect the various components, signals, and processes that constitute this controller, shedding light on the underlying methodology and algorithms employed.

5.1 Components

- **clkdivider :** This component generates a 25 MHz pixel clock from a system default 100 MHz master clock.
- **hcounter and vcounter :**These components are horizontal and vertical counters, respectively. They count the horizontal and vertical sync pulses, generating horcount and vercount signals. The en signal from hcounter is used to enable the calculation of vcounter.
- **dist_mem_gen_0 and dist_mem_gen_1 :** These are components representing memory blocks. dist_mem_gen_0 is read-only, providing data based on the address (romaddress), while dist_mem_gen_1 is read-write, allowing data to be written (ramdata) or read (dram) based on the address (ramaddress).
- **ControlFSM** This is the parent component consisting of the complete logic and also integrating all the Finite states with other components. It contains primarily 4 states : ak : to assign kernel values to the variables mm : to calculate the corresponding values and then find the maxvalue and minvalue which would allow us to normalize the values. ar : to calculate final values and normalize them, so that they can be stored in the ram. d : this is the final state where it is in display mode.
- **MacUnit** This part basically multiplies and adds any 2 signals given to it, and it acts as a multiplier accumulator block for our implementation.

5.2 Signals

- **clk25 :** The 25 MHz clock is generated by clkdivider.
- **horcount and vercount :** Counters for horizontal and vertical synchronization.
- **switchdisplay :** Signal indicating whether to display data or not.
- **romaddress, ramaddress, displayaddress :** Addresses used for reading from memory (romaddress and ramaddress) and writing to the display (displayaddress)..
- **romdata :** Data read from read-only memory.
- **dram, ramdata:** Data read and written to the read-write memory.
- **r, g, b:** Output signals representing red, green, and blue pixel values.
- **mode:** Variable to control the mode of operation.

5.3 Processes

- **write_in_ram:**This process generates pixel values by performing computations on data from memory. It uses clock to control the flow, and based on the mode, it reads data from memory, processes it, and writes it back to memory.
- **calc_max_min:**This process calculates maximum and minimum values by performing computations on data from memory for normalization. It uses clock to control the flow, and based on the mode, it reads data from memory, processes it, and writes it back to memory.
- **assign_kernel:**This process assigns variables from kernel.
- **assign_sync:**This process generates horizontal (hsync) and vertical (vsync) synchronization signals based on horcount and vercount.
- **assign_display:** This process determines whether the display should be active (switchdisplay = '1') based on horcount and vercount.
- **address_assigner:** This process assigns memory addresses based on the current pixel (i and j) and the mode of operation.
- **display_on_vga :**This process generates the output pixel values (r, g, and b) based on the data from memory (ramdata). It updates the display address (ramaddress_for_display) and color values when switchdisplay is active.

5.4 Control Flow Overview

The control flow in the implemented VGA controller is a precisely choreographed series of steps. Initiated by the master clock, the system synchronizes its components, ensuring seamless operation. Horizontal and vertical counters (horcount and vercount) are driven by the pixel clock's rising edges, defining pixel positions. Concurrently, data is read from memory, processed, and mapped to red, green, and blue channels. Synchronization signals (hsync and vsync) are generated with precise timing, coordinating the display. This intricate interplay between memory, computation, and display forms the core of the VGA controller's operation, resulting in a cohesive visual output. Understanding this controlled sequence is fundamental to grasping the controller's functionality.

Figure 1: Overview of task: filtering

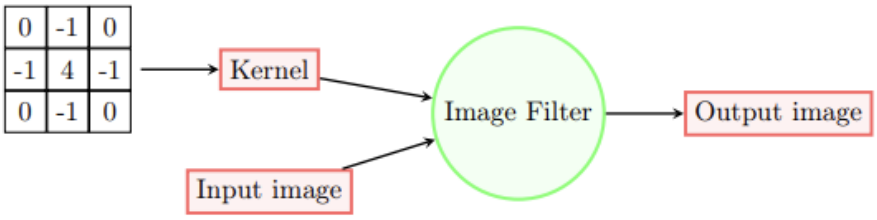

Figure 2

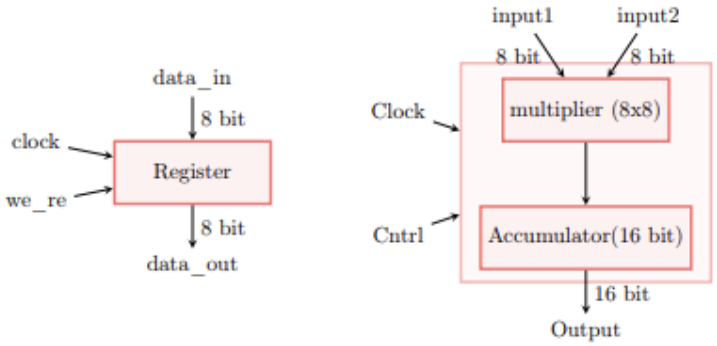

Figure 4: Register and MAC unit

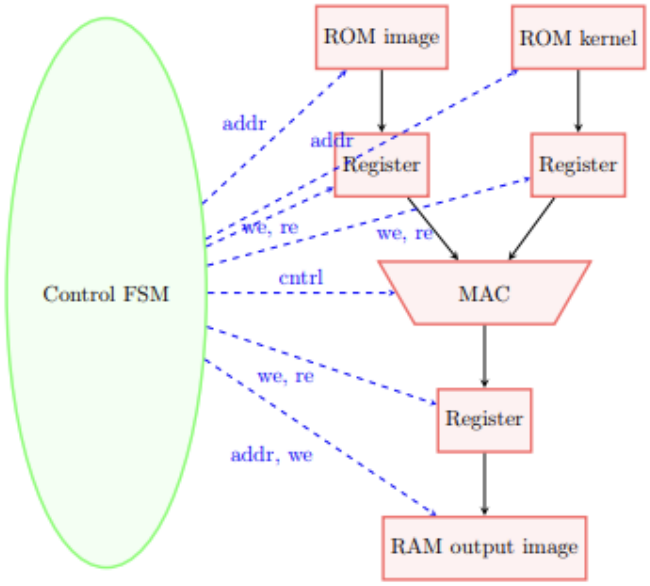

Figure 5: Overview of control path

Figure 3: Module diagram for controller

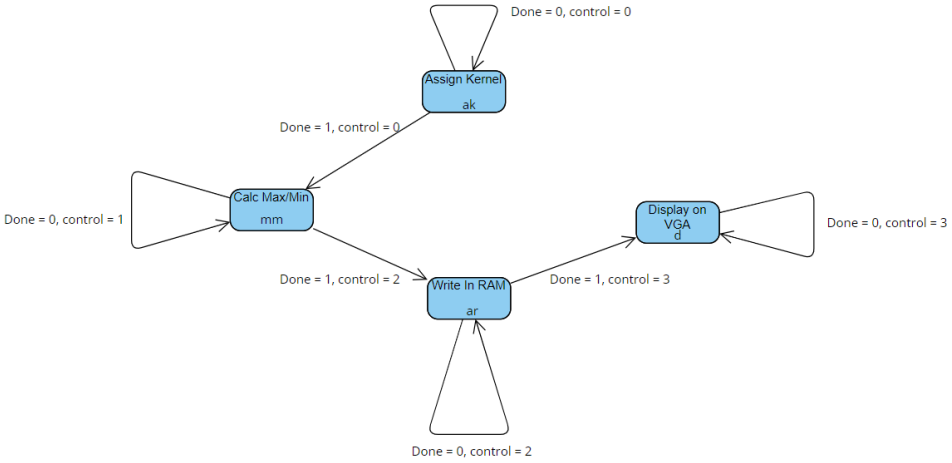

Figure 4: FSM flow

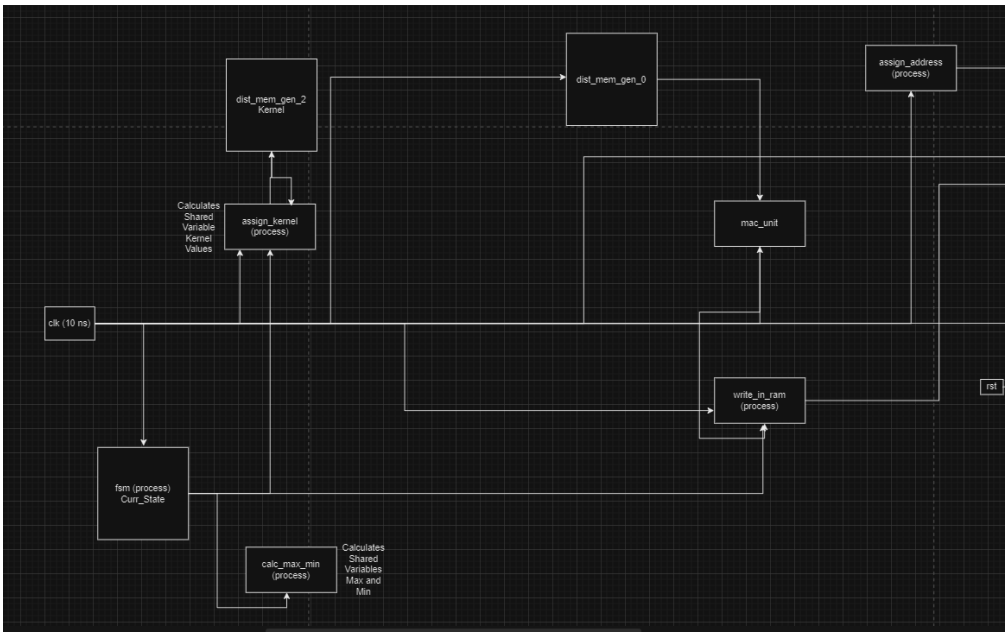

Figure 5: Module diagram for controller - 1

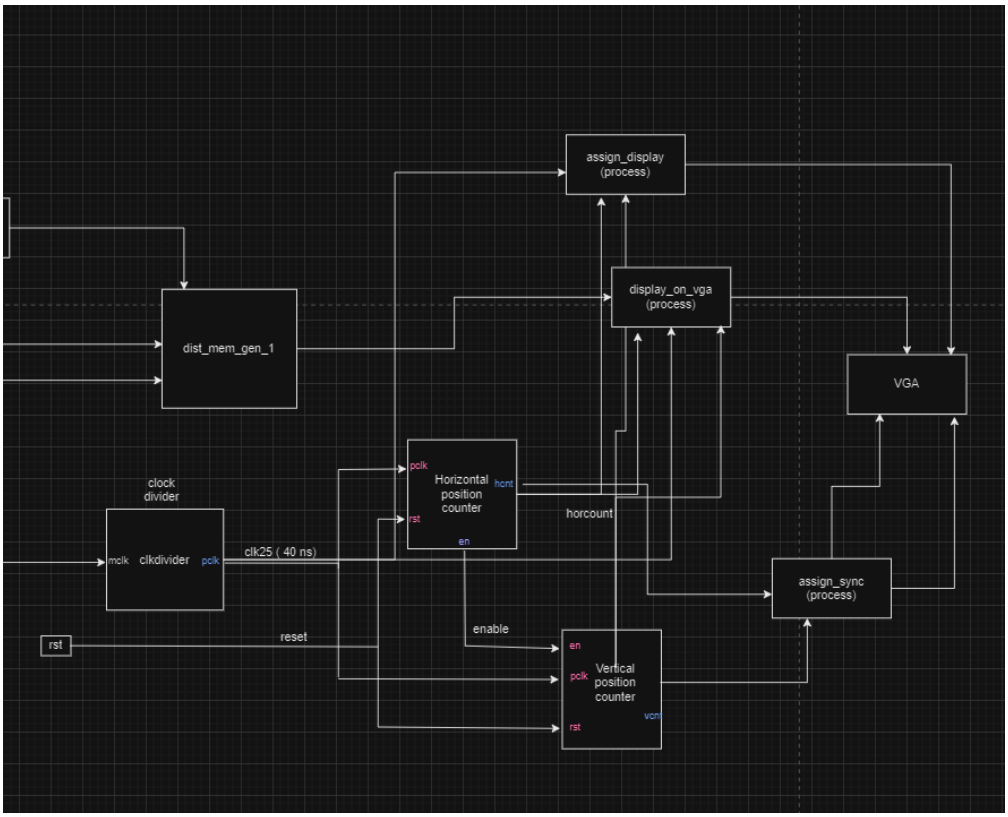

Figure 6: Module diagram for controller - 2

6 Testing Strategy and Test Cases

The project's validity and efficiency were verified through a rigorous testing approach, encompassing diverse scenarios and unique cases to assess its performance comprehensively. The testing strategy involved the following key elements:

6.1 Test Cases Generation

To rigorously evaluate the image processing system, a diverse set of 256x256 pixel 8-bit images was curated to rigorously test the image processing system. Utilizing a MATLAB script for automated formatting, the test cases included various complexities. This meticulous approach ensured a comprehensive evaluation, affirming the system's adaptability, accuracy, and resilience in handling diverse real-world inputs.

6.2 Simulations and Validation

To ensure the robustness and accuracy of the VGA controller implementation, comprehensive simulations were conducted after generating the bitstream. Each of the four VHDL programs provided in the initial prompt was rigorously tested. Through simulation, the behavior of the components, signals, and processes within the controller was scrutinized under various conditions. These simulations served as an essential testing strategy, validating the correctness of the code and its ability to generate the expected VGA signals. The successful outcomes of these simulations provided a strong validation of the controller's functionality, confirming its capability to generate synchronized video signals for display. Below, you will find the simulation images depicting the expected behavior of the VGA controller under different scenarios.

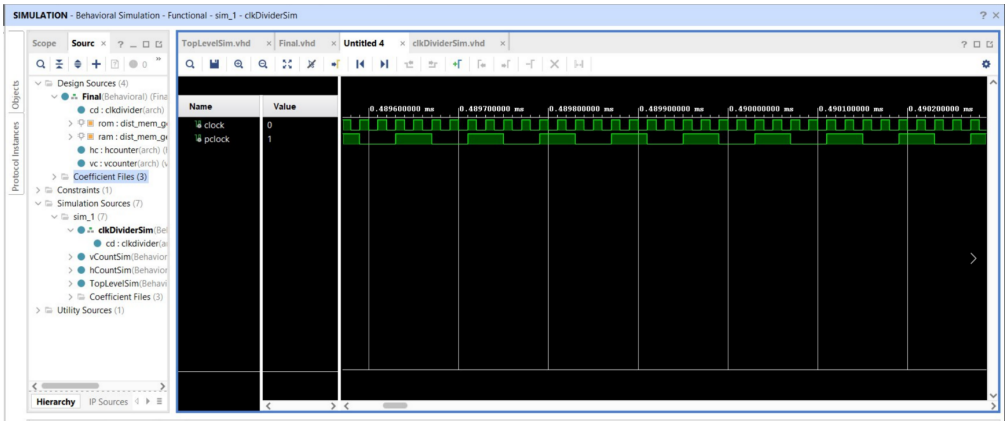

Figure 7: clkdivider Simulation

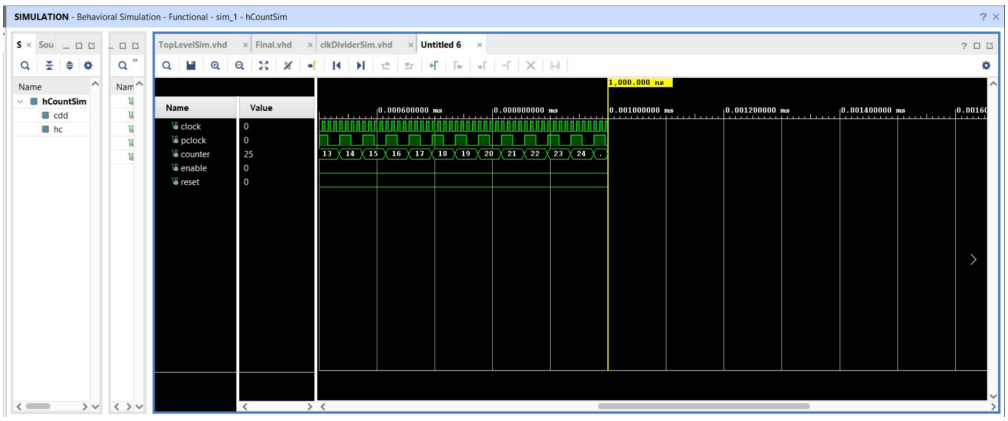

Figure 8: hcount Simulation

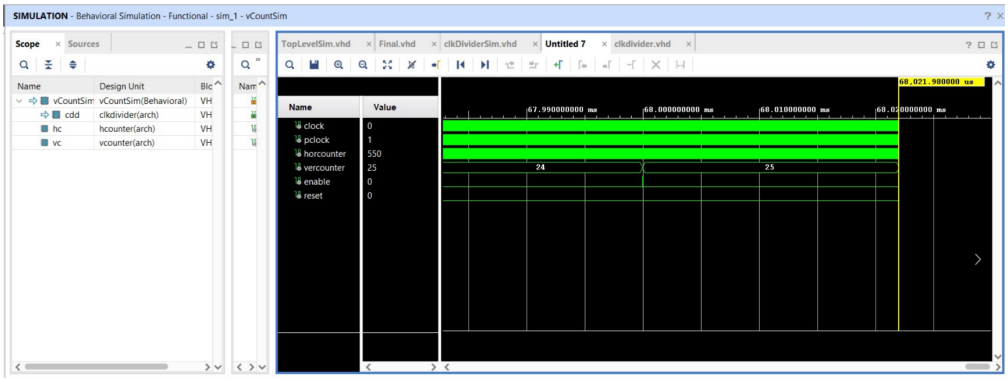

Figure 9: vcount Simulation 1

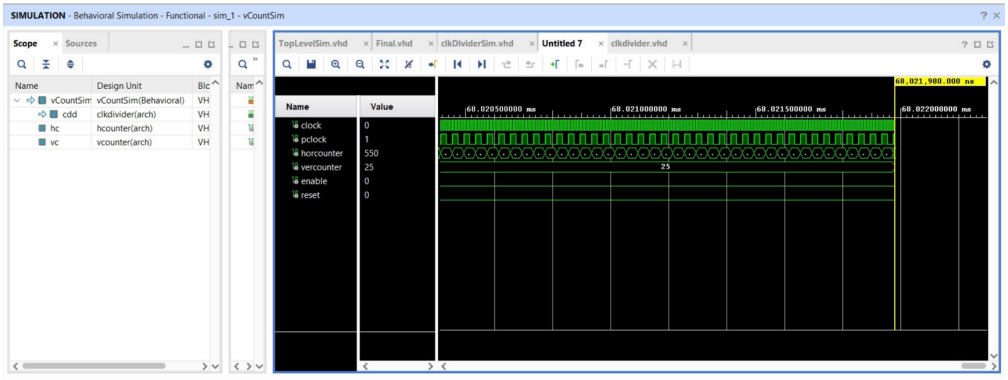

Figure 10: vcount Simulation 2

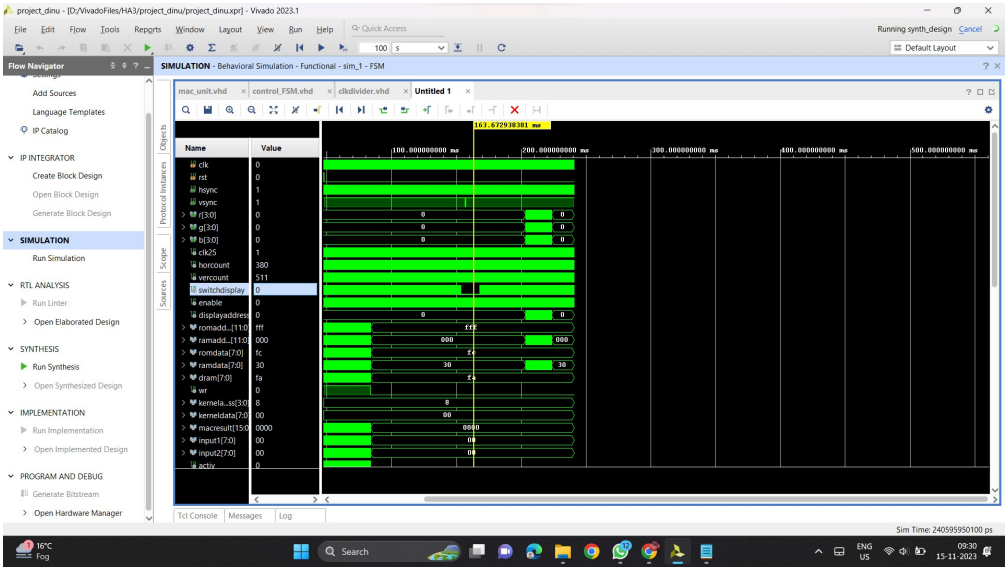

Figure 11: Complete Design Simulation 1

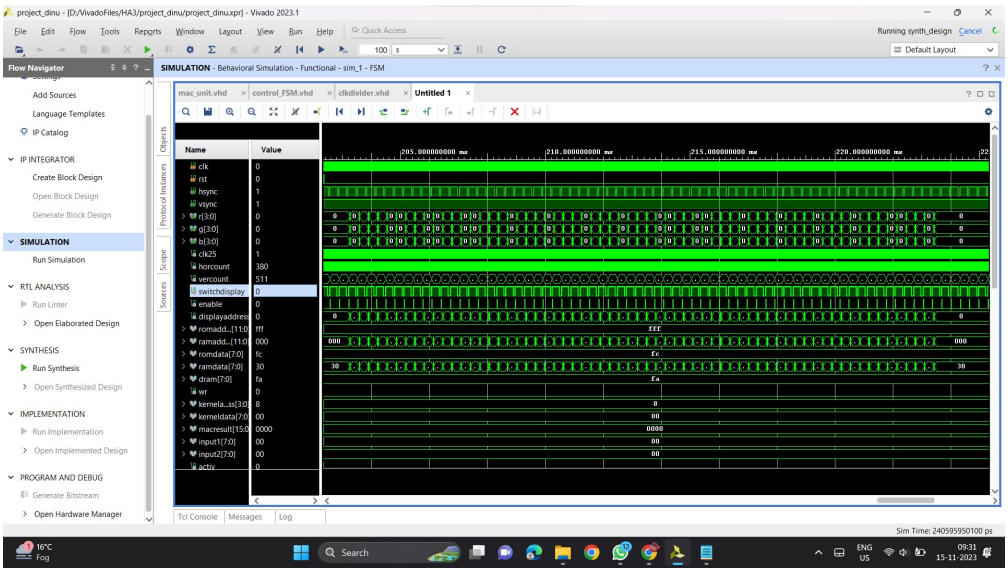

Figure 12: Complete Design Simulation 2

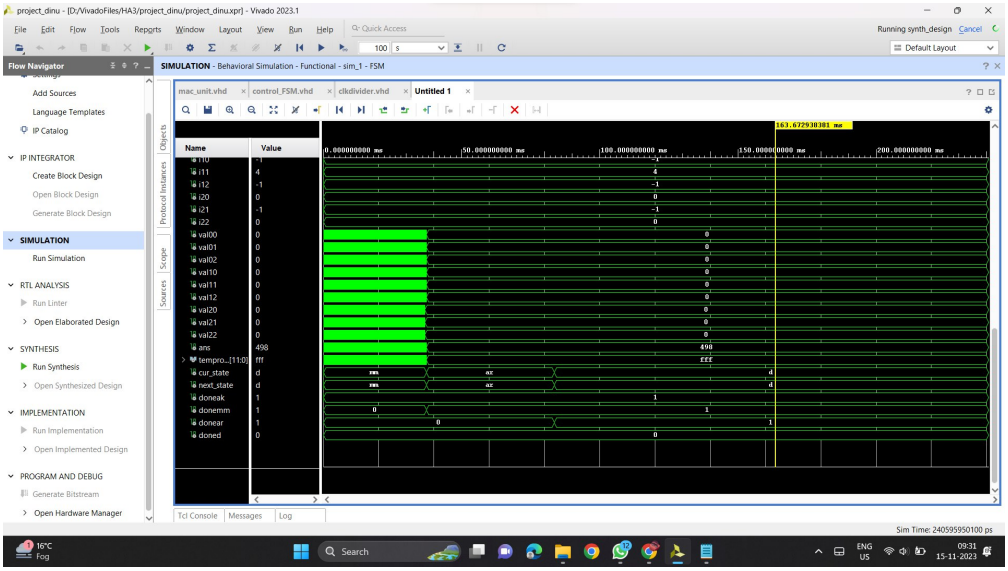

Figure 13: Complete Design Simulation 3

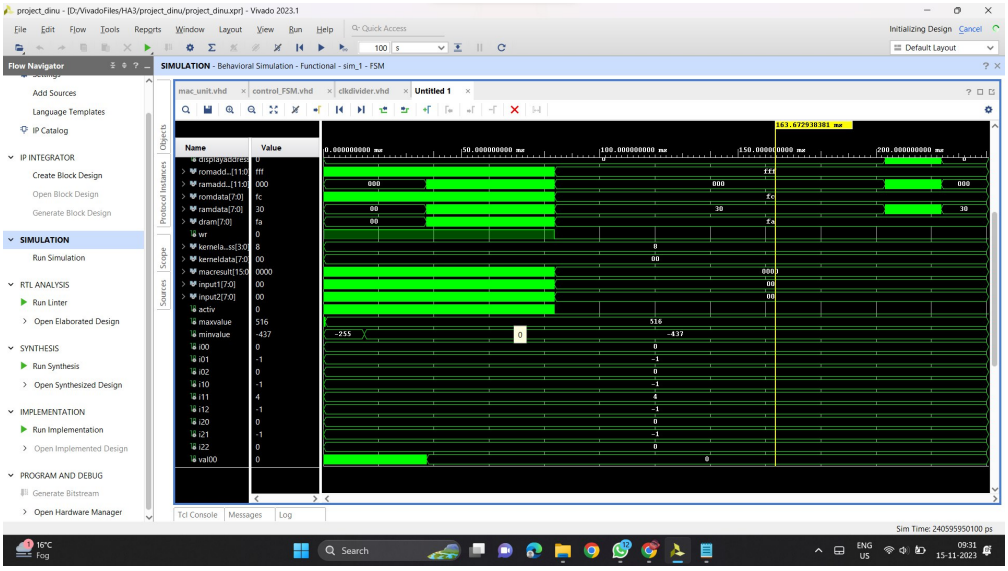

Figure 14: Complete Design Simulation 4

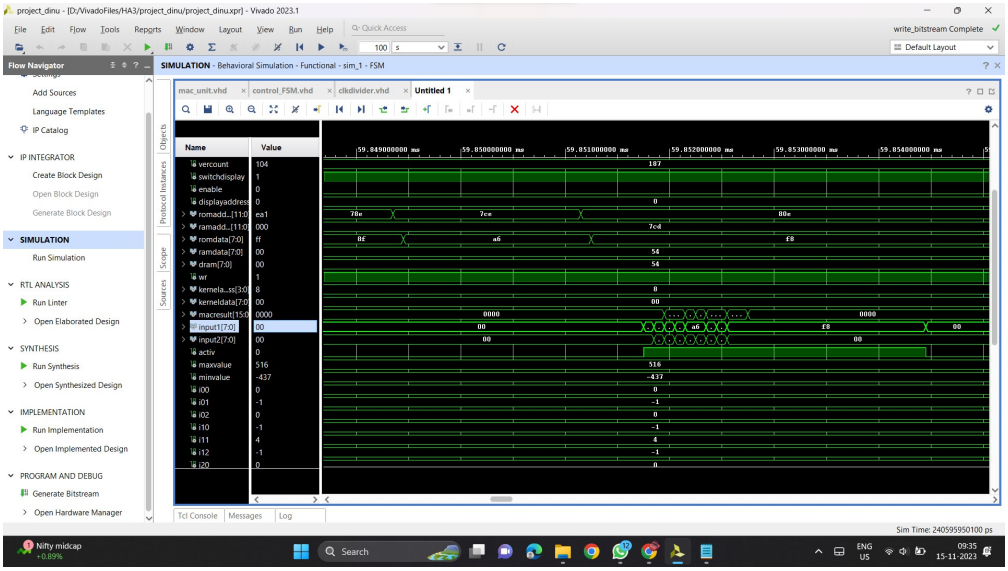

Figure 15: Complete Design Simulation 5

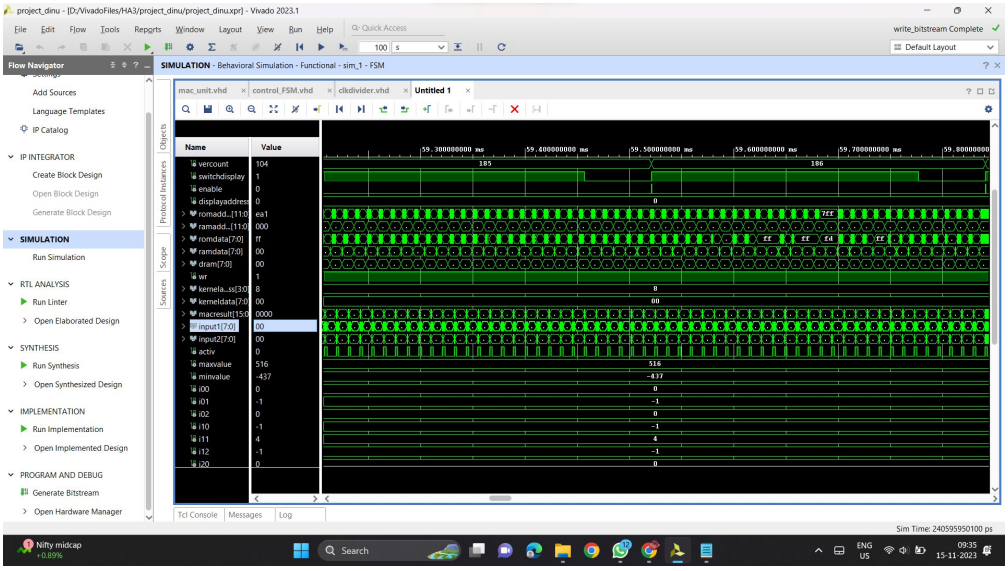

Figure 16: Complete Design Simulation 6

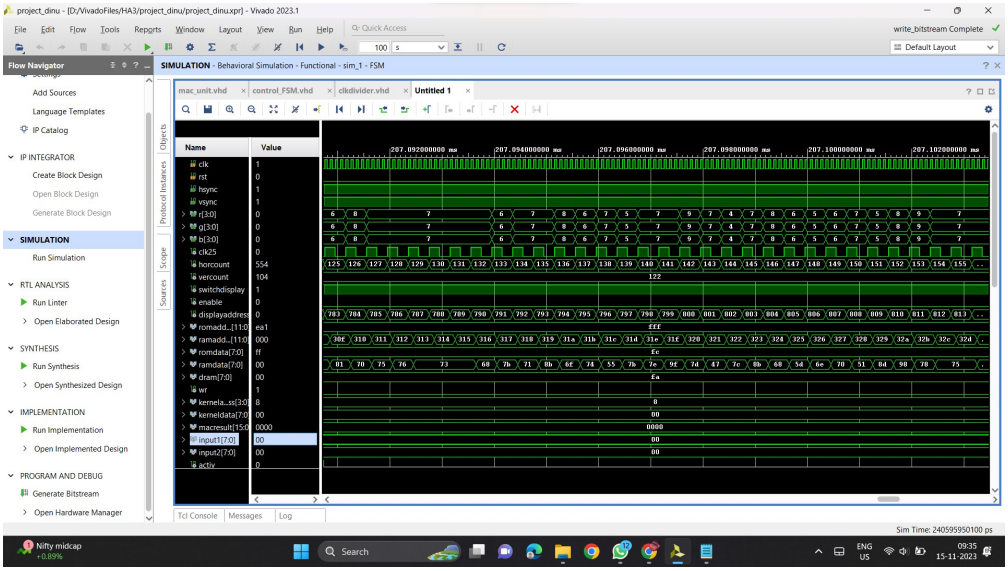

Figure 17: Complete Design Simulation 7

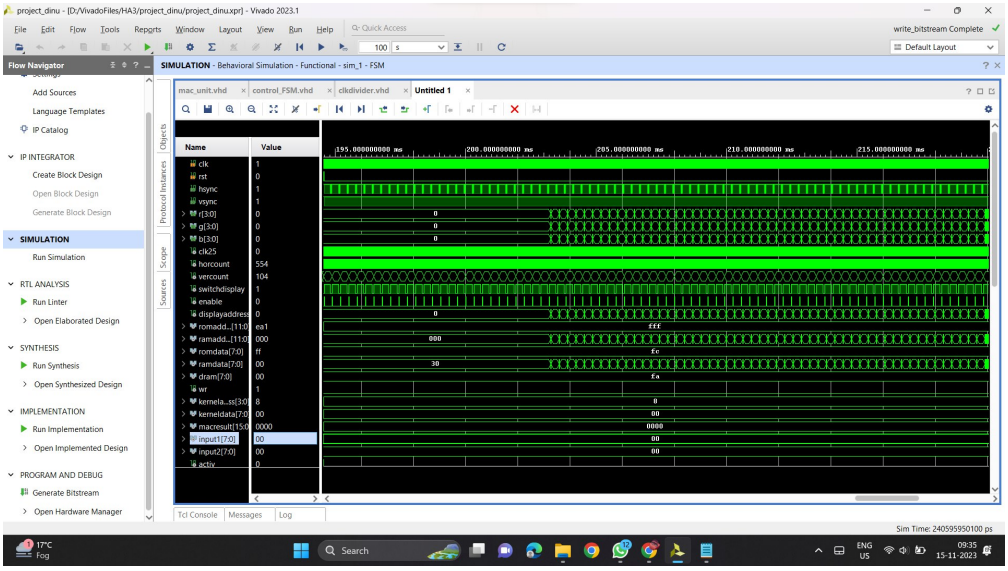

Figure 18: Complete Design Simulation 8

6.3 Testing

6.3.1 Basic Image Processing:

A set of basic images with simple structures and minimal gradients were used to verify the fundamental functionality of the program. This ensured that the system could accurately process input images and apply gradient operations to basic images.

Test Case 1:

Figure 19: Output file

6.3.2 Complex Image Structures:

More intricate images with complex gradients and varying textures were employed. This challenging test case aimed to evaluate the program's capability to handle complex image structures, ensuring that it processed intricate details accurately and produced meaningful gradient outputs.

Test Case 1:

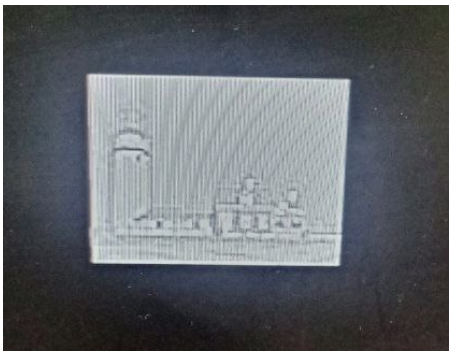

Figure 20: Output file

Test Case 2:

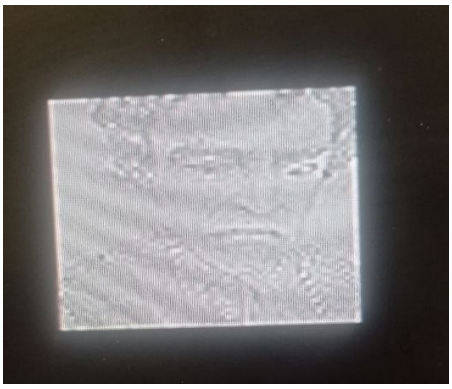

Figure 21: Output file

The successful execution and precise results of these tests instilled confidence in the project's correctness, reliability, and efficiency. This comprehensive testing approach validated the system's ability to handle various image complexities, ensuring it delivered accurate and visually appealing gradient-enhanced images in real-time scenarios.

7 Utilization Report

```
Copyright 1986-2022 Xilinx, Inc. All Rights Reserved.
------------------------------------------------------------------------------------------------------------
| Tool Version : Vivado v.2022.1 (lin64) Build 3526262 Mon Apr 18 15:47:01 MDT 2022
| Date         : Wed Nov 15 04:16:36 2023
| Host         : dhd running 64-bit Ubuntu 20.04.3 LTS
| Command      : report_utilization -file FSM_utilization_placed.rpt -pb FSM_utilization_placed.pb
| Design       : FSM
| Device       : xc7a35tcpg236-1
| Speed File   : -1
| Design State : Fully Placed
------------------------------------------------------------------------------------------------------------

Utilization Design Information
```

Table of Contents
----

1. Slice Logic

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs	3344	0	0	20800	16.08
LUT as Logic	3088	0	0	20800	14.85
LUT as Memory	256	0	0	9600	2.67
LUT as Distributed RAM	256	0			
LUT as Shift Register	0	0			
Slice Registers	835	0	0	41600	2.01
Register as Flip Flop	835	0	0	41600	2.01
Register as Latch	0	0	0	41600	0.00
F7 Muxes	439	0	0	16300	2.69
F8 Muxes	68	0	0	8150	0.83

1.1 Summary of Registers by Type

Total	Clock Enable	Synchronous	Asynchronous
0	_	-	-
0	_	-	Set
0	_	-	Reset
0	_	Set	-
0	_	Reset	-
0	Yes	-	-
0	Yes	-	Set
68	Yes	-	Reset
0	Yes	Set	-
767	Yes	Reset	-

2. Slice Logic Distribution

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice	1068	0	0	8150	13.10
SLICEL	723	0			
SLICEM	345	0			
LUT as Logic	3088	0	0	20800	14.85
using O5 output only	0				
using O6 output only	2725				
using O5 and O6	363				
LUT as Memory	256	0	0	9600	2.67
LUT as Distributed RAM	256	0			
using O5 output only	0				
using O6 output only	256				
using O5 and O6	0				
LUT as Shift Register	0	0			
Slice Registers	835	0	0	41600	2.01
Register driven from within the Slice	486				
Register driven from outside the Slice	349				
LUT in front of the register is unused	128				
LUT in front of the register is used	221				
Unique Control Sets	69		0	8150	0.85

* * Note: Available Control Sets calculated as Slice * 1, Review the Control Sets Report for more information regarding control sets.

3. Memory

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

* Note: Each Block RAM Tile only has one FIFO logic available and therefore can accommodate only one FIFO36E1 or one FIFO18E1. However, if a FIFO18E1 occupies a

4. DSP

Site Type	Used	Fixed	Prohibited	Available	Util%
DSPs	12	0	0	90	13.33
DSP48E1 only	12				

5. IO and GT Specific

Site Type	Used	Fixed	Prohibited	Available	Util%
Bonded IOB	16	16	0	106	15.09
IOB Master Pads	7				
IOB Slave Pads	8				
Bonded IPADs	0	0	0	10	0.00
Bonded OPADs	0	0	0	4	0.00
PHY_CONTROL	0	0	0	5	0.00
PHASER_REF	0	0	0	5	0.00
OUT_FIFO	0	0	0	20	0.00
IN_FIFO	0	0	0	20	0.00
IDELAYCTRL	0	0	0	5	0.00
IBUFDS	0	0	0	104	0.00
GTPE2_CHANNEL	0	0	0	2	0.00
PHASER_OUT/PHASER_OUT_PHY	0	0	0	20	0.00
PHASER_IN/PHASER_IN_PHY	0	0	0	20	0.00
IDELAYE2/IDELAYE2_FINEDELAY	0	0	0	250	0.00
IBUFDS_GTE2	0	0	0	2	0.00
ILOGIC	0	0	0	106	0.00
OLOGIC	0	0	0	106	0.00

6. Clocking

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

7. Specific Feature

Site Type	Used	Fixed	Prohibited	Available	Util%
BSCANE2	0	0	0	4	0.00
CAPTUREE2	0	0	0	1	0.00
DNA_PORT	0	0	0	1	0.00
EFUSE_USR	0	0	0	1	0.00
FRAME_ECCE2	0	0	0	1	0.00
ICAPE2	0	0	0	2	0.00
PCIE_2_1	0	0	0	1	0.00
STARTUPE2	0	0	0	1	0.00
XADC	0	0	0	1	0.00

8. Primitives

Ref Name	Used	Functional Category
LUT5	1087	LUT
LUT2	898	LUT
LUT6	804	LUT
FDRE	767	Flop & Latch
CARRY4	637	CarryLogic
MUXF7	439	MuxFx
RAMS64E	256	Distributed Memory
LUT1	246	LUT
LUT4	211	LUT
LUT3	205	LUT
MUXF8	68	MuxFx
FDCE	68	Flop & Latch
OBUF	14	IO
DSP48E1	12	Block Arithmetic
BUFG	3	Clock
IBUF	2	IO

9. Black Boxes

Ref Name	Used

10. Instantiated Netlists

Ref Name	Used
dist_mem_gen_2	1
dist_mem_gen_1	1
dist_mem_gen_0	1

Results

Part 1 (Week 1-2): Design and Test Compute Unit

- Successfully designed and tested the compute unit, implementing the MAC unit for the image filtering operation.
- The MAC unit performs element-wise multiplication and accumulation to obtain one output pixel.
- Simulated waveforms with test cases demonstrate the correct functionality of the compute unit.

Part 2 (Week 3): Implement FSM and Integrate with Hardware Design

- Implemented the Finite State Machine (FSM) to control the MAC unit and manage data transfers between ROM/RAM and local registers.
- Integrated the FSM with the hardware design, ensuring proper synchronization and interaction between memory operations and the compute unit.
- Simulated waveforms with test cases validate the correct operation of the integrated system.

Conclusions

1. **Compute Unit Design:** The MAC unit, as a part of the compute unit, effectively performs the image filtering operation, producing accurate output pixels based on the given 3x3 kernel and 64x64 input image.
2. **Finite State Machine (FSM):** The FSM successfully controls the operation of the hardware components, orchestrating the read/write operations in memory and computation in the MAC unit. It ensures a systematic and synchronized workflow for the image filtering process.
3. **Integration and Optimization:** The integration of the compute unit and FSM demonstrates a well-coordinated system for image filtering. Further optimization could be explored to enhance the execution time, taking advantage of potential overlaps in consecutive filtering operations.
4. **Simulation Verification:** Thorough simulation testing has been conducted at each stage of the design, providing confidence in the correctness and reliability of the hardware implementation. This ensures that the system operates as intended before transitioning to FPGA testing.

5. **Future Improvements:** Consideration of optimizations, such as parallelization or pipelining, could be explored to reduce the execution time further. Additionally, feedback from FPGA testing may unveil additional areas for improvement or adjustments in the design.

In summary, the implemented system fulfills the requirements of the assignment, achieving successful image filtering through the collaboration of the compute unit and FSM. Ongoing efforts will focus on potential optimizations and FPGA testing to validate real-world performance.