Hardware Assignment -3

Daksh Dhaker-2022CS51264 ,Umesh Kumar-2022CS11115
COL215:- Digital Logic & System Design
Group-3

PART-1

1 Memory Generation and Architecture

There are 4 types of memory used :-

1.1 ROM1

This is a ROM type memory .This ROM stores the values of the filter. This ROM has following 3 ports:-

- a: in std_logic_vector(3 downto 0 as it stores 9 values of the kernel);
- spo: out std_logic_vector(7 downto 0 for the image stored);
- clk: in std_logic

1.2 ROM2

This ROM stores the values of input image. This ROM has 3 ports:-

- a : Input address of the pixel (11 downto 0 in case of an image of 4096 pixels)
- clk : std logic_vector type (input clock)
- spo : This is the output pixel of the given input address (7 downto 0) (std logic_vector type)

1.3 RAM

This is a RAM memory type. This RAM stores the values of pixels after applying 3x3 filter. It has 5 ports:-

- a : Input address of the pixel (11 downto 0 in case of an image of 4096 pixels)
- clk : std logic_vector type (input clock)
- d : This is the input pixel to be stored by RAM (15 downto 0 in this case)(std logic_vector type)
- we : (when we is 1,it represents write mode, else it represents read mode) (std_logic type)
- spo : This is the output pixel obtained in read mode(15 downto 0)(std logic_vector type)

1.4 RAM2

This is also a RAM memory ; This RAM stores the values of pixels after Normalisation. It has 5 ports:-

- a : Input address of the pixel (11 downto 0 in case of an image of 4096 pixels)
- clk : std logic_vector type (input clock)
- d : This is the input pixel to be stored by RAM (11 downto 0 in this case)(std logic_vector type)
- we : (when we is 1,it represents write mode, else it represents read mode) (std logic type)
- spo : This is the output pixel obtained in read mode(15 downto 0)(std logic_vector type)

2 Signals Used

We have used the following signals for the filtering,image processing and noramlization:-

- clk_slow :-used in clock divider part;it oscillates between 0 and 1.

- registers reg00,reg02,reg01,reg10,reg11,reg12,reg20,reg21,reg22; these registers store input values of kernel
- kernel_itr;used for iterating over kernel
- kernel_comp:used as a helper to continue kernel traversal cycle.
- ker_rom_add: std_logic_vector(3 downto 0);is a vector that access the index at which we are present for the kernel.
- data_rom_ker: std_logic_vector(7 downto 0); accessing output data of kernel during register assignment.
- FSM_state: a flag which controls the process operation.

All of the above signals are used in kernel to ROM storage. Now are the signals for the image processing part:-

- pix00,pix01,pix02,pix10,pix11,pix12,pix20,pix21,pix22 :used for accessing the pixels for multiplication.
- i,j: used as iterators for ROM to RAM conversion.
- grad_cycle:this signal run the cycle for image processing.
- rom_add: std_logic_vector(11 downto 0) :this vector extracts the image address (which was stored in ROM2).
- ram_add: std_logic_vector(11 downto 0) : this vector access the RAM1 data,which includes the pixels after applying operations.
- data_rom: std_logic_vector(7 downto 0) :a vector used for accessing the output data from the ROM2,after operation.
- sum: integer :used as a helper for mathematical calculations.
- wr_enable:a signal for enabling or disabling writing.
- temp_max_val:maximum value for initial pixels.
- temp_min_val: minimum value for initial pixels.
- final_min_val: minimum value for final pixels.
- data_in: std_logic_vector(15 downto 0): a signal for giving input from RAM1 .

- final_max_val: maximum value for final pixels.

now are the signals for normalization part:-

- itr_nrml,jtr_nrml:iterators for iterating over RAM1 for image normalization.
- nrm_comp:a vector for accessing the index at which we are present.
- nrm_ram_add: std_logic_vector(11 downto 0):similar to kernel processes.
- nrm_output_pixel : std_logic_vector(15 downto 0): for accessing the output pixel after operation.
- nrm_data_in: std_logic_vector(15 downto 0):to access output data from the RAM2.
- nrm_wr_enable: same as a flag for writing.
- nrm_sum:helper integer.

3 Logic and Algorithms

3.1 For the ROM storage of kernel

The process starts with the rising edge of clk_slow,and runs for the whole time while kernel_comp<27 . There are27 cycles because on each kernel value,which processing that value in 3 cycles:

- First one is for assigning of ker_rom_add the 4-bit value of the address.
- Second,we wait for one cycle .
- Third,we assign the corresponding Register value from the kernel data.These registers are the one's which will store the kernel filter value in whole processes from now.

for implementation of above algorithm,we have used case statements,which are as follows:-

- Cases 0, 3, 6, 9, 12, 15, 18, 21, and 24: ker_rom_add is set to the value of kernel_itr (converted to a 4-bit vector). kernel_comp is incremented by 1.This step marks assignment of ker_rom_add for accessing kernel addresses. Also note that this process runs simulataneously with the FSM writing in the implementation.

- Cases 2, 5, 8, 11, 14, 17, 20, 23: kernel_itr is incremented by 1, and a variable regXY is assigned the value of data_rom_ker converted to an integer.This steps marks the register assignment .
- Case 25 (the "others" case): kernel_itr is incremented by 1, and reg22 is assigned the value of data_rom_ker converted to an integer. kernel_comp is also incremented by 1. This marks the end of whole cycle for the process.

Thus,this signifies the markings of the registers in the process ,which are assigned the corresponding RAM to be used in further processing.

3.2 Image processing (MAC)

This Process is the MAC unit which calculates the value of output pixel after applying 3x3 filter.

- In every 4 cycles, we calculate the value of each pixXY. In first cycle we assign corresponding value to the ram address. Then we wait for next cycle and in third cycle value assigned to corresponding pixXY. Then we wait for next cycle.
- This is repeated again and again for each of the pixXY register. So in 36 cycles, all of our 9 pixXY registers are assigned their corresponding values.
- In 38th cycle we calculate the value of sum,i.e., the value calculated by multiplying these pixXY with corresponding regXY and then adding all these 9 terms.
- In 39th cycle the final_max_val and final_min_val is updated.
- In the 40th cycle the value of iterators i and j is being updated.
- Also in the corner and edges cases the corresponding pixXY values are assigned zero which goes out of bounds.

3.3 The FSM process

This process controls the syncronisation between the process2,process3 and process3:

- This process decides that in which state the program is currently working and accordingly allows only one process at a time to execute.

- As soon as the conditions(values of flag signals) changes, it starts another process and halts previous one.
- Iniatially only process1 activates because both of the if statments in FSM are false.
- Then as kernel_comp >=27 then process1 halts and process2 is triggerd by this FSM.
- When value of i equals 64 then process 3 is also triggered by this FSM. Also, note that in process3 we have assigned the value to the ram_add using this process(it means at the starting of first cycle of process3 this FSM assigns a value to ram add which is further used in later 11 cycles of process 3).
- This also controls the write_enable signal of the RAMs.

3.4 For the Normalization of Image

This process starts with the rising edge of clk_slow;(Note that this process runs only for the part when i=64;which indicates the the whole image is extracted to RAM and is available for operation) This process has 12 cycles ;which are as follows:-

- in the first cycle,the value is assigned to r_add by the FSM process which is running in parallel with the current process.
- We wait for 1 cycle for output pixel and proper image processing .
- Then in the third cycle , nrm_sum is assigned the value of output_pixel .

We have implemented our idea by using case statements as follows:-

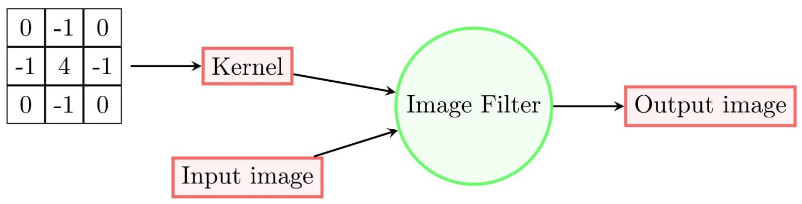
- case 0,1,3,4,8 are the cases where we just wait and increase our nrm_comp by 1;
- Case 2: nrm_comp is incremented by 1, and nrm_sum is assigned the value of output_pixel converted to an integer.
- Case 5: nrm_sum is updated by multiplying it by 255, and nrm_comp is incremented by 1.This ensures that nrm_sum gets its correct value.

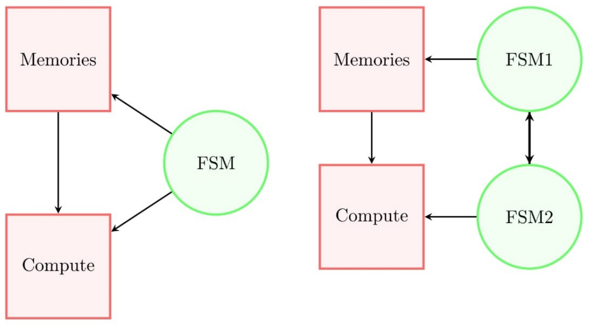
- Case 6: nrm_sum is updated by dividing it by the difference between final_max_val and final_min_val, and nrm_comp is incremented by 1.This completes our normalisation formula.
- Case 7: nrm_ram_add set to (64∗i+j) in an unsigned vector of 16 bits , and nrm_comp is incremented by 1.
- Case 9: nrm_data_in is set to the value of nrm_sum converted to a 16-bit vector,which is the output data we need to calculate , and nrm_comp is incremented by 1.
- Case 10: Checks if jtr_nrml is equal to 63. If true, it resets jtr_nrml to 0 and increments itr _nrml by 1;Thus sending the cycle to initials of the image pixels stored in RAM1. Otherwise, it increments jtr_nrml;Thus continuing the Loop. nrm_comp is incremented by 1.
- Others case: If itr_nrml is equal to 64, it resets both itr_nrml and jtr_nrml to 0. nrm_comp is set to 0.This is the formal end of our loop.

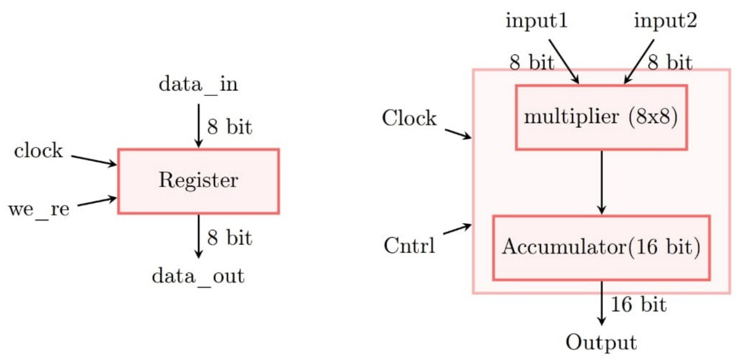
4 Formula and theory used for this part

4.1 for the image processing part:-

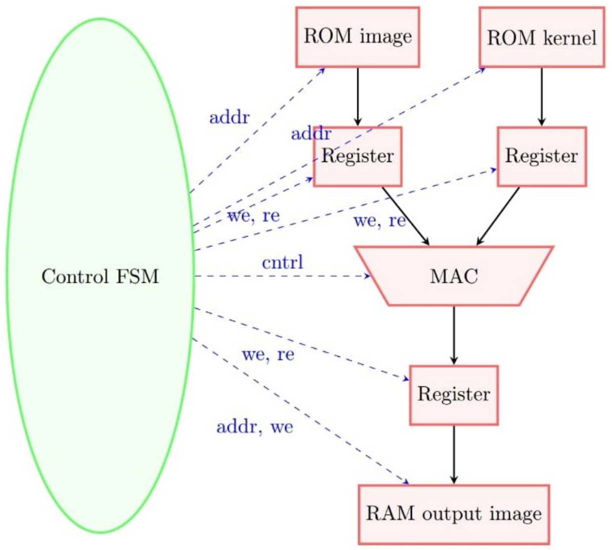
The output pixel value O(i, j) at location (i, j) is computed as the sum of element-wise multiplications between kernel value and input image pixel, as shown in the equation below. The input image pixel at location (i, j) is denoted I(i, j).
O(i, j) = a I(i 1, j 1) + b∗ I(i 1, j) + c∗I(i 1, j + 1) +d∗I(i, j 1) + e∗I(i, j) + f∗I(i, j + 1) +g∗I(i + 1, j 1) + h∗I(i + 1, j) + i∗I(i + 1, j + 1)
Below is the Block diagram/raw idea of assignment:-







(Block diagram for the Register unit and MAC unit)



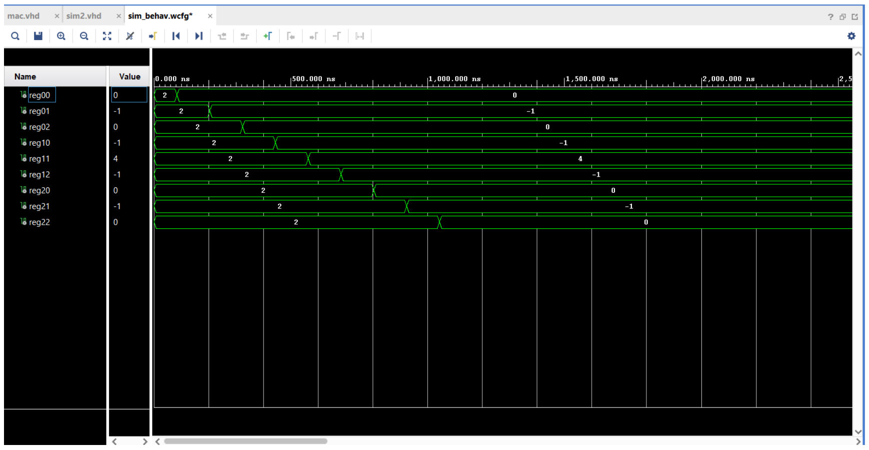
(This diagram is shown for how FSM is controlling the path of execution)

4.2 For the normalisation part

In linear normalization, for each pixel, we perform a scaling according to the equation 1, where old_max and old_min are minimum and maximum pixel values in the output image, new_max and new_min is the new range of pixel value in the final image:
New_I(i, j) = (I(i, j) old_min) ∗(new_max new_min) (old_max old_min) + new_min

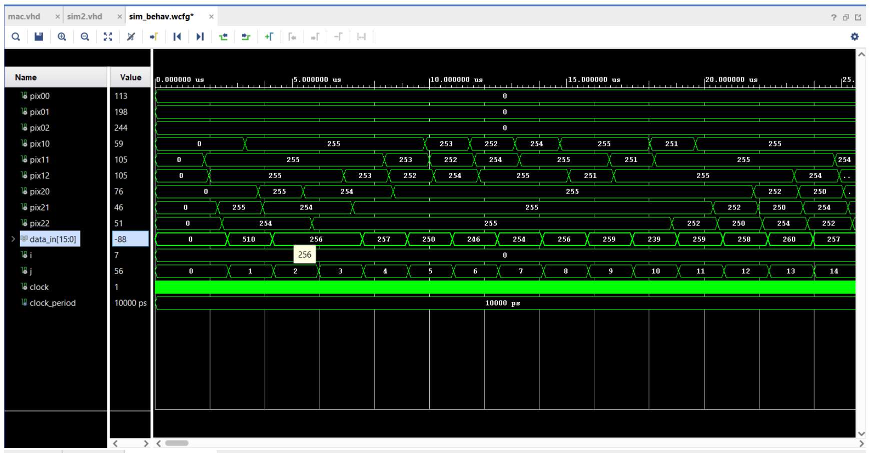
5 Snapshots and testcases

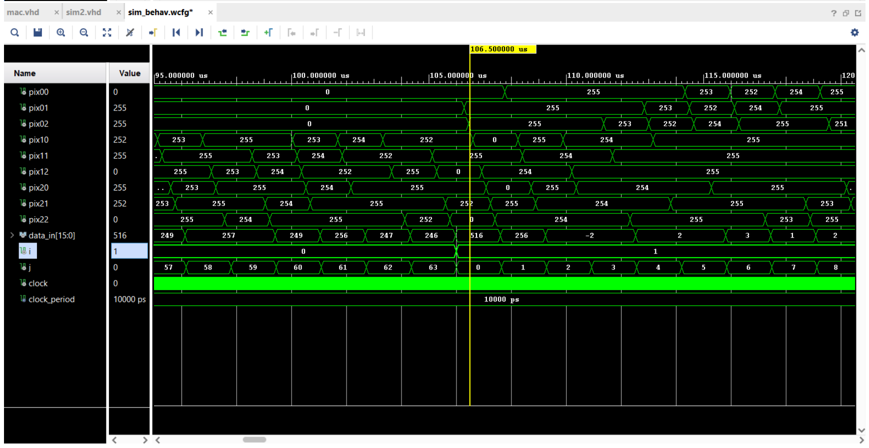
5.1 Testbench simulation for the Kernel reading part



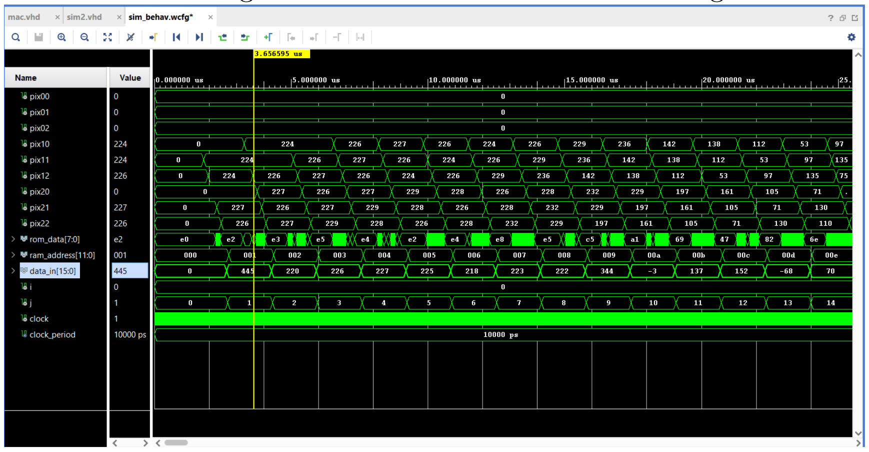
Note That the filter value is read correct here and all registers' value is initialised by 2,as in our code.

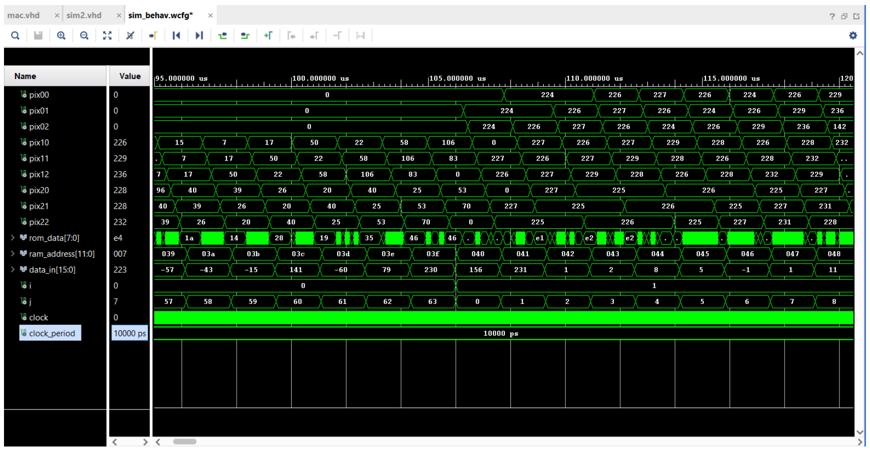
5.2 Test cases' test simulation during image processing





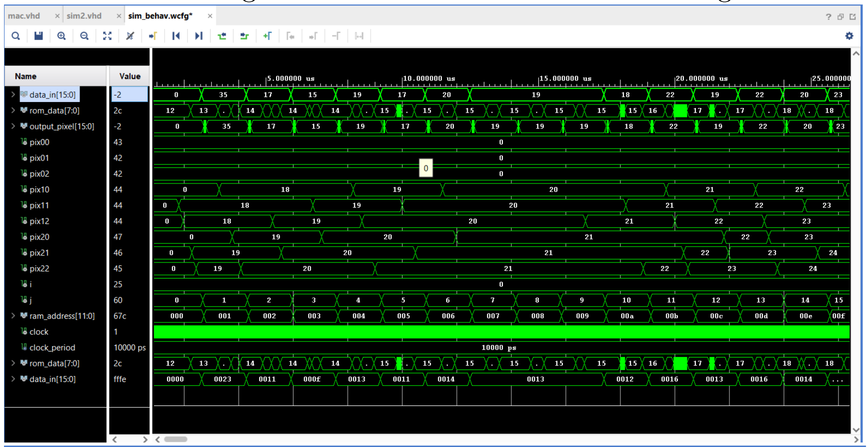
Two images of 'coin' test case while simulating .



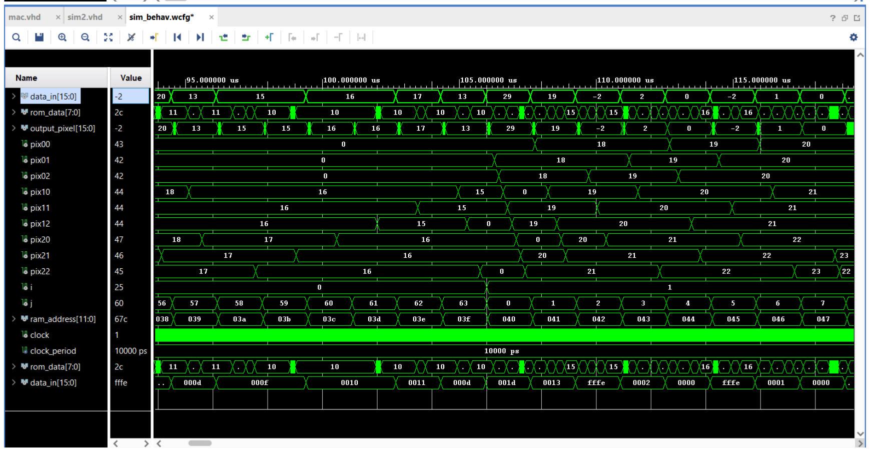


Name	Value
pix00	0
pix01	0
pix02	0
pix10	226
pix11	229
pix12	236
pix20	228
pix21	228
pix22	232
rom_data[7:0]	e4
ram_address[11:0]	007
data_in[15:0]	223
i	0
j	7
clock	0
clock_period	10000 ps

Two images of 'face' Test case while simulating.



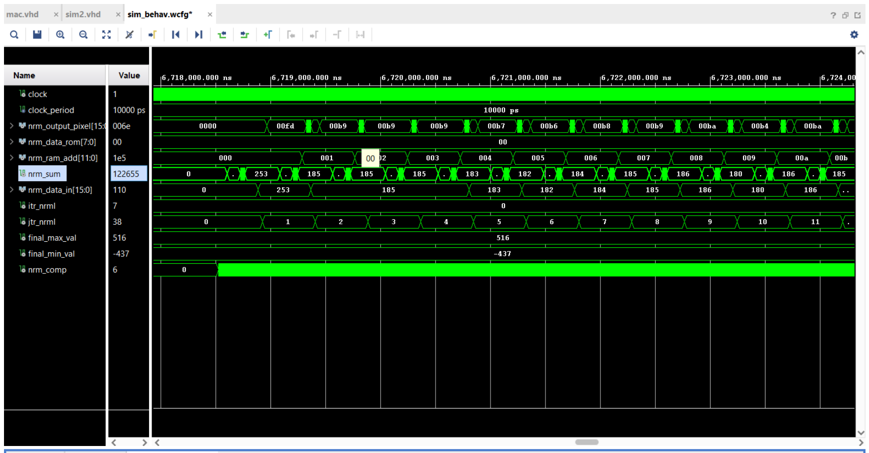
Name	Value
data_in[15:0]	-2
rom_data[7:0]	2c
output_pixel[15:0]	-2
pix00	43
pix01	42
pix02	42
pix10	44
pix11	44
pix12	44
pix20	47
pix21	46
pix22	45
i	25
j	60
ram_address[11:0]	67c
clock	1
clock_period	10000 ps
rom_data[7:0]	2c
data_in[15:0]	fffe



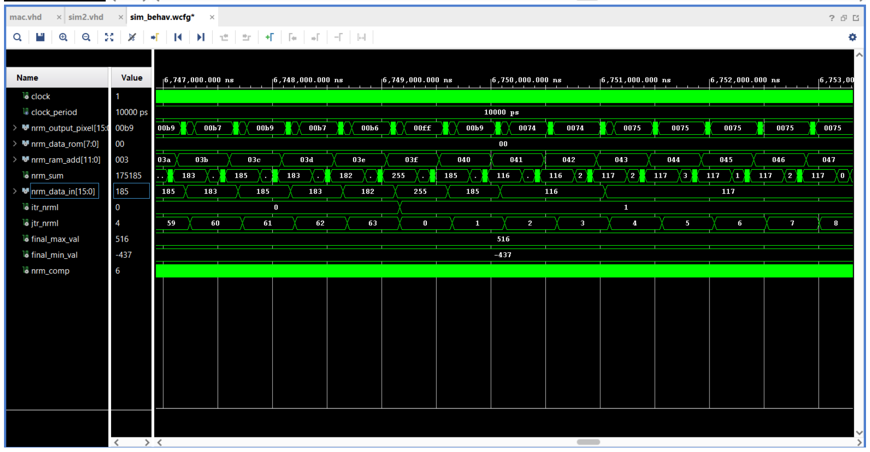
Name	Value
data_in[15:0]	-2
rom_data[7:0]	2c
output_pixel[15:0]	-2
pix00	43
pix01	42
pix02	42
pix10	44
pix11	44
pix12	44
pix20	47
pix21	46
pix22	45
i	25
j	60
ram_address[11:0]	67c
clock	1
clock_period	10000 ps
rom_data[7:0]	2c
data_in[15:0]	fffe

Two images of 'Light_house' test case while simulating.

5.3 Test cases' test simulation during Image Normalization

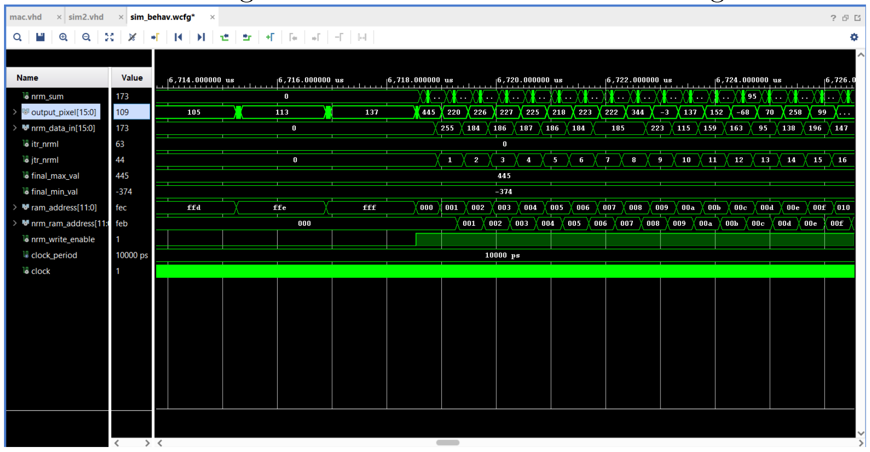


Name	Value
clock	1
clock_period	10000 ps
nrm_output_pixel[15:0	006e
nrm_data_rom[7:0]	00
nrm_ram_add[11:0]	1e5
nrm_sum	122655
nrm_data_in[15:0]	110
itr_nrml	7
jtr_nrml	38
final_max_val	516
final_min_val	-437
nrm_comp	6

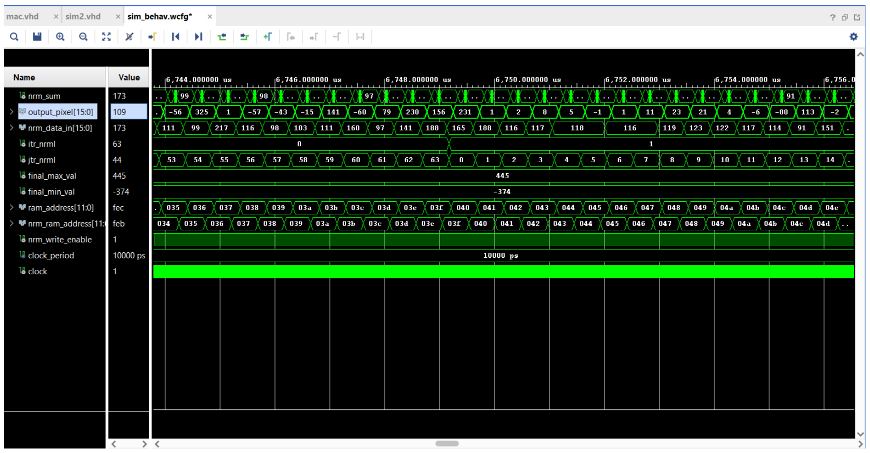


Name	Value
clock	1
clock_period	10000 ps
nrm_output_pixel[15:0	00b9
nrm_data_rom[7:0]	00
nrm_ram_add[11:0]	003
nrm_sum	175185
nrm_data_in[15:0]	185
itr_nrml	0
jtr_nrml	4
final_max_val	516
final_min_val	-437
nrm_comp	6

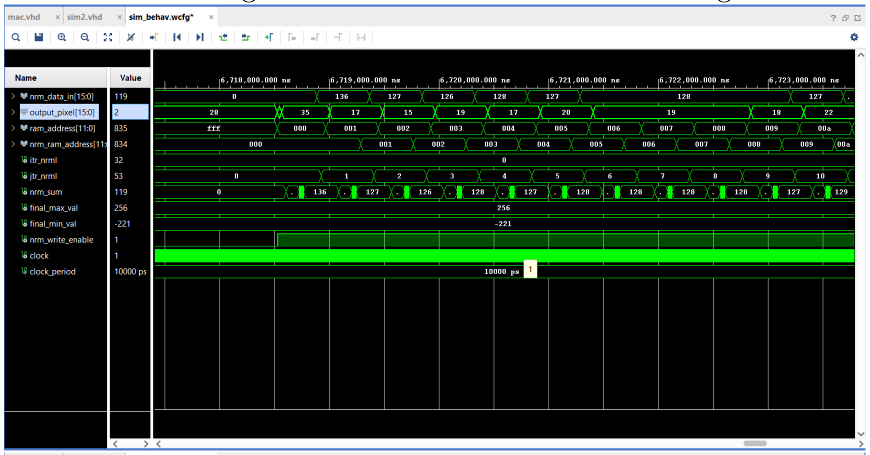
Two images of 'coin' test case while simulating .

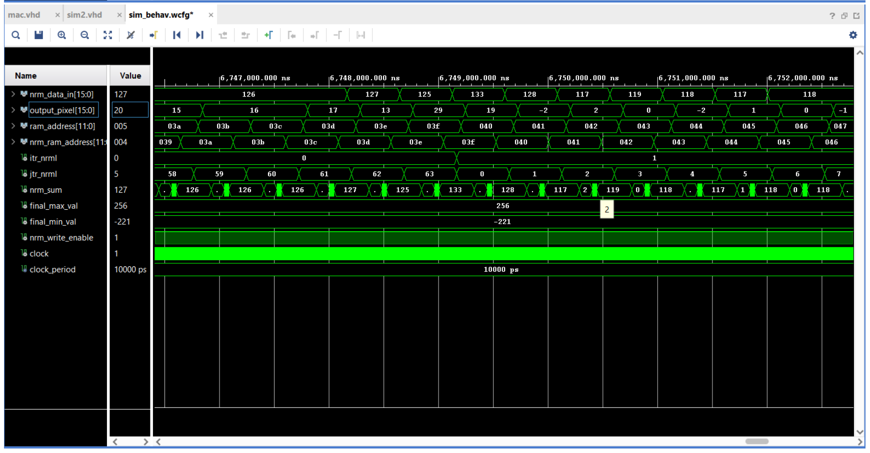


Name	Value
nrm_sum	173
output_pixel[15:0]	109
nrm_data_in[15:0]	173
itr_nrml	63
jtr_nrml	44
final_max_val	445
final_min_val	-374
ram_address[11:0]	fec
nrm_ram_address[11:0	feb
nrm_write_enable	1
clock_period	10000 ps
clock	1



Two images of ’face’ Test case while simulating.





Two images of ’Light_house’ test case while simulating.

End of Part1

Part -2

(From here onwards,We Starts the part-2,i.e. implementation through use of FSM)

Similar to the Part-1 ,we first created the kernel extraction,then image processing through the pre-defined formula ,and then normalised the image through normalization process.In part-1 ,we did this using the multiplexer or MAC unit ;but this time ,we did it using FSM. For implementation of the 3 processes using FSM,we used 2 sub-processes and with the help of counters and synchronization. The 2 processes are as follows:-

6 VGA Controller

6.1 Initialising condition

For this process,We start it on rising clock edge;because in FSM,we change the signals only at rising edges.Then we checked the condition of the **Video_On**(a new signal for controlling when to enable change in states).If Video_On is not set to '1',then we reset our signal vectors for R,G,B of the VGA controller to default '0'.Else ,we proceed to next step.

6.2 Reset and Assignment

After this,we checked the hcnt and vcnt to be in display ranges of and thus assign the respective R,G,B values for the nrm_output_pixel in 4-bit vectors. If hcnt and vcnt are not in 0 to 64 ranges,then we reset them to '0'.

Thus,our VGA controller effectively controls the reseting and memory allocation to display.

6.3 FSM States

We have 4 FSM states which decides what we are to perform:-

- State 0:-In this state ,we read from kernel and store its data in regXY.
- State 1:- In this state,we apply the multiplication and accumulation operation and stored it in RAM1(RAM) created.
- State 2:-In this state ,we apply the normalization operation and store its final pixels in RAM2.(Final state).
- State 3:-In this state,we start over VGA controller to display the image which has pixels stored in RAM2.

By these designs , for iterating over 4 states,we make our image ready to display and after it ,It remains to state 3 until a transition in state occur.

7 Transition Of States

The state transition process begins with rising edge of clock and transits only in the following conditions:-

- State 0 to State 1:-This transition occurs when Kernel comp becomes 27 which marks the completion of reading from Kernel ,and only now ,we can start the MAC operation in state 1.
- State 1 to State 2:-This transition occurs when i=64(mainly)because at this state kernel comp=27 and itr_nrml=0.So,this transition only depends on i being 64 or not. This marks completion of MAC operation and now we can perform normalization in state2.
- State 2 to State 3:- This transition occurs when itr_nrml=64 ,which means the completion of normalization and now we can display the output image through VGA controller (State 3).State 3 is the final state.

Thus ,this ends the marking for our Transitioning model.

Now we shall make a small note on how we managed the counters.
-Counters:- We used following counters/synchronizers:-

- Vsync:-Depends on Vcnt; is '1' when on rising edge,Vcnt is not in a certain range .Else,we set it to '0'.
- Vcnt:-Depends on hcnt ;is incremented by 1 when hcnt reaches a certain value and Vcnt is not the end value.
- hsync:- On rising clock edge ,when hcnt is out of a range,then it is triggered.
- hcnt :- On rising clock,increases on every edge until reaches terminal value,after which it resets to '0' again.
- Video_On:- This signal triggers on rising edge under the condition that hcnt and vcnt both are in display range;Otherwise set to '0'.

8 Test cases and Simulation

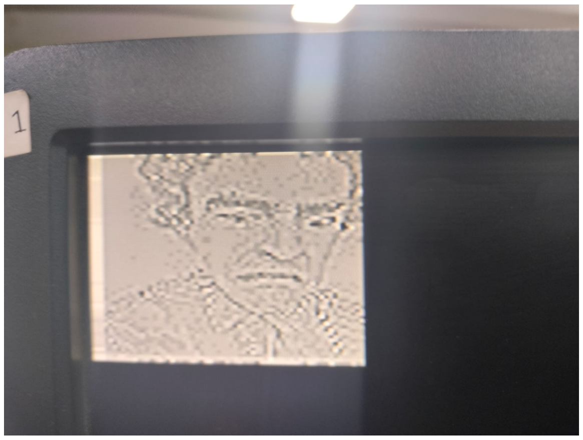
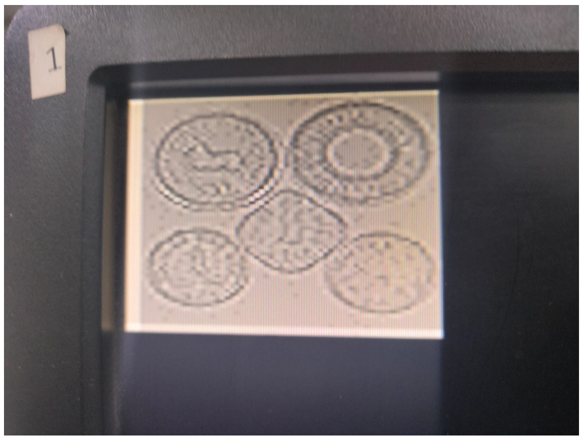
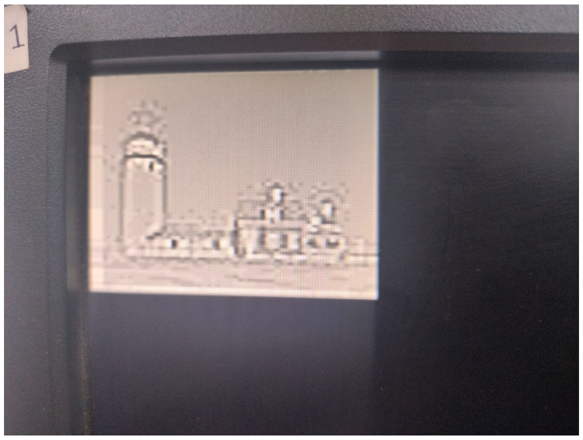


Image of face under filter

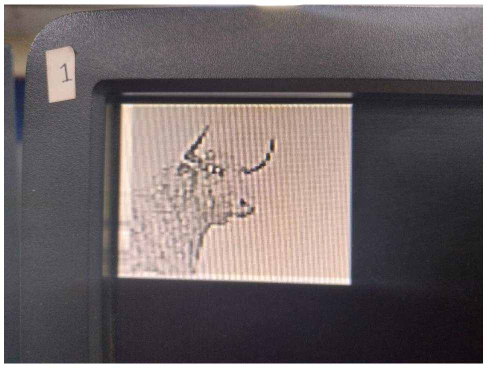


coins



light house

-Now a few of our test cases:-



Initial image of bull and image of bull after filter(sharpening)

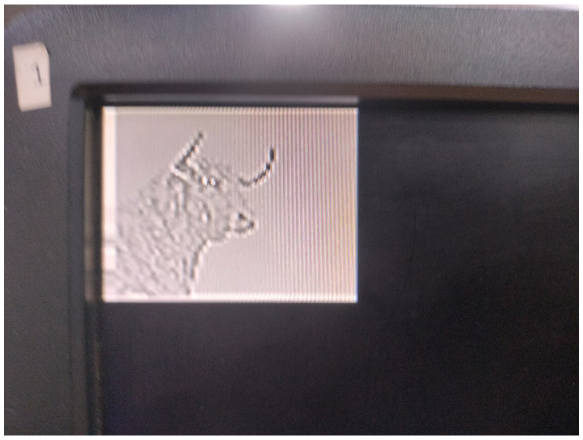
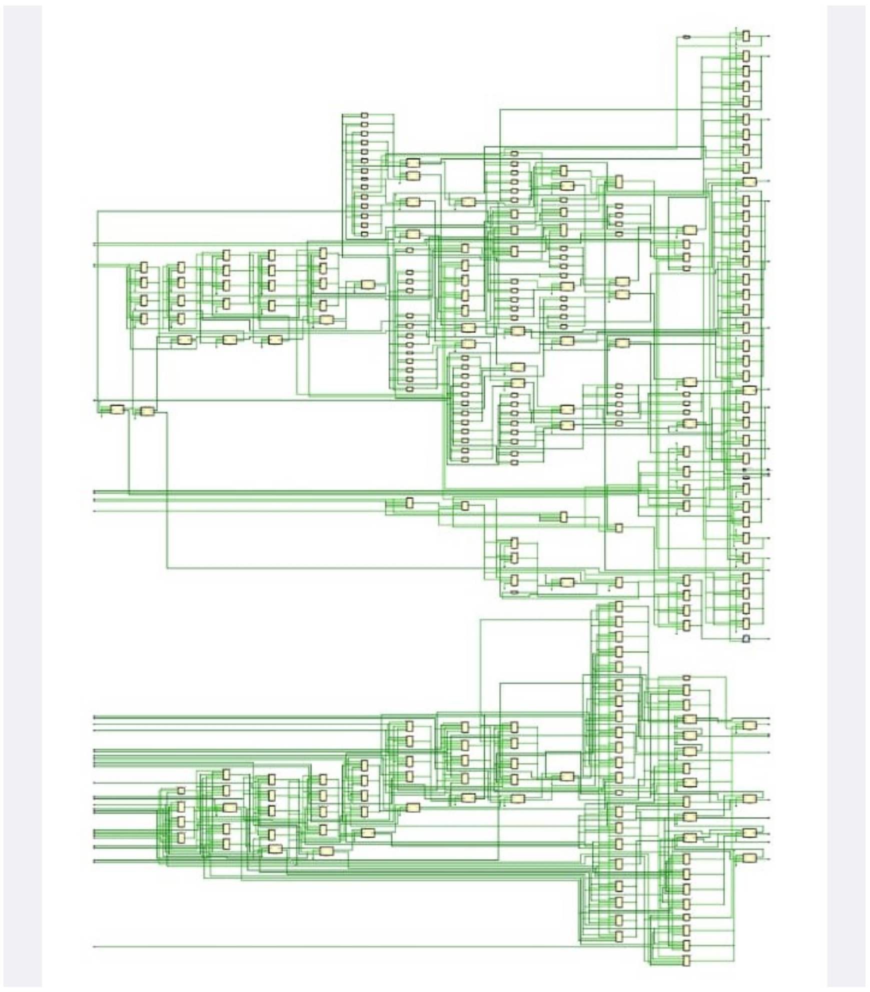


Image of bull with edge detection filter.

Image of moon initially and finally.

9 Block diagram

The block diagram for the whole FSM processing is following:-



End Of Report