Object Oriented Modeling of Electronic Circuits – Spring 1402-03
Computer Assignment 5-6

Pattern Finder Circuit

Amir Hosein Samoudi – 810100108

Introduction:

In this assignment we have designed a pattern finder circuit which can be used to find specific patterns in an image. The core of this circuit are convolutional blocks that are building block used in a convolutional neural network (CNN) for image recognition.

We have implemented this circuit using SystemC with two different methods: first we have described its bus functional model (BFM) to have a model of design and check the functionality of it and do the verifications for a wide ranges of inputs. After that we have also implemented the RT level model of this circuit with the detail design of each element.

The design approach in each part is that first we implement each one of the building blocks of the top-level design and verify it for a range of inputs so that make sure everything is working as expected before moving to the next parts.

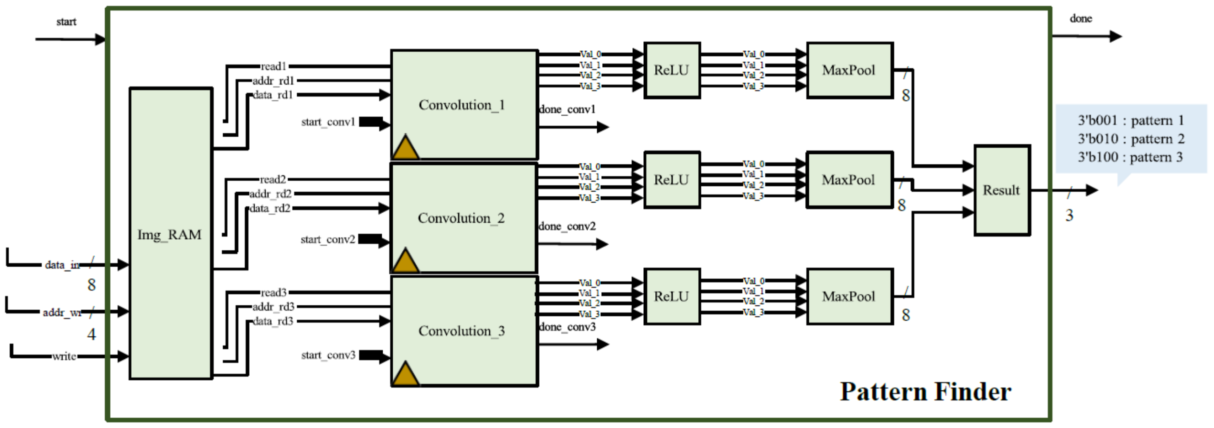
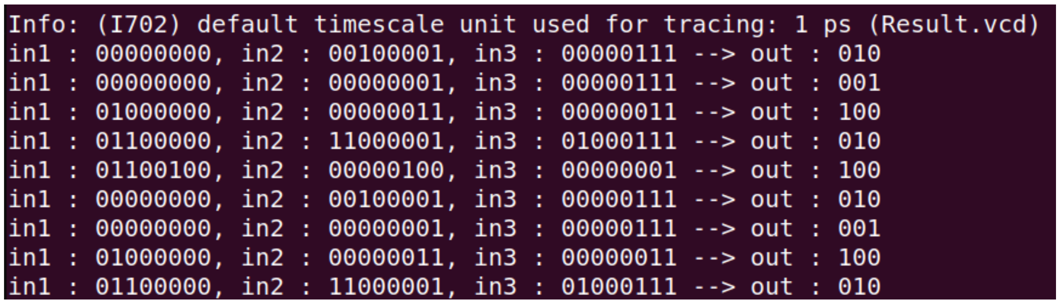


Figure 1. Schematic of the circuit

- ## Bus Functional Model (BFM)

Result block:

We start our work with describing the Result module which is a combinational module. In its output instead of the largest value, '1' is asserted. We tested the functional model of this block for different sets of inputs and results are as below:

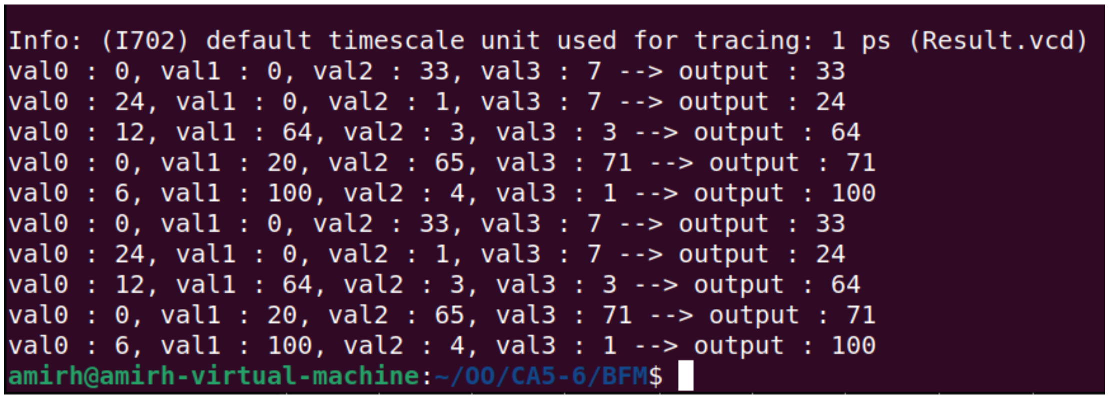


```
Info: (I702) default timescale unit used for tracing: 1 ps (Result.vcd)
in1 : 00000000, in2 : 00100001, in3 : 00000111 --> out : 010
in1 : 00000000, in2 : 00000001, in3 : 00000111 --> out : 001
in1 : 01000000, in2 : 00000011, in3 : 00000011 --> out : 100
in1 : 01100000, in2 : 11000001, in3 : 01000111 --> out : 010
in1 : 01100100, in2 : 00000100, in3 : 00000001 --> out : 100
in1 : 00000000, in2 : 00100001, in3 : 00000111 --> out : 010
in1 : 00000000, in2 : 00000001, in3 : 00000111 --> out : 001
in1 : 01000000, in2 : 00000011, in3 : 00000011 --> out : 100
in1 : 01100000, in2 : 11000001, in3 : 01000111 --> out : 010
```

Figure 2. Verification of Result module

MaxPool Block:

Next module that we have designed is MaxPool block which is also a combinational block and puts the maximum value of its inputs on its the output.



```
Info: (I702) default timescale unit used for tracing: 1 ps (Result.vcd)
val0 : 0, val1 : 0, val2 : 33, val3 : 7 --> output : 33
val0 : 24, val1 : 0, val2 : 1, val3 : 7 --> output : 24
val0 : 12, val1 : 64, val2 : 3, val3 : 3 --> output : 64
val0 : 0, val1 : 20, val2 : 65, val3 : 71 --> output : 71
val0 : 6, val1 : 100, val2 : 4, val3 : 1 --> output : 100
val0 : 0, val1 : 0, val2 : 33, val3 : 7 --> output : 33
val0 : 24, val1 : 0, val2 : 1, val3 : 7 --> output : 24
val0 : 12, val1 : 64, val2 : 3, val3 : 3 --> output : 64
val0 : 0, val1 : 20, val2 : 65, val3 : 71 --> output : 71
val0 : 6, val1 : 100, val2 : 4, val3 : 1 --> output : 100
amirh@amirh-virtual-machine:~/OO/CA5-6/BFM$
```

Figure 3. Verification of MaxPool module

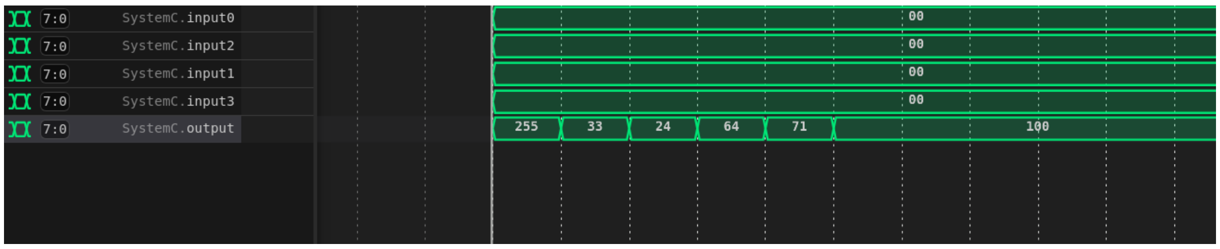
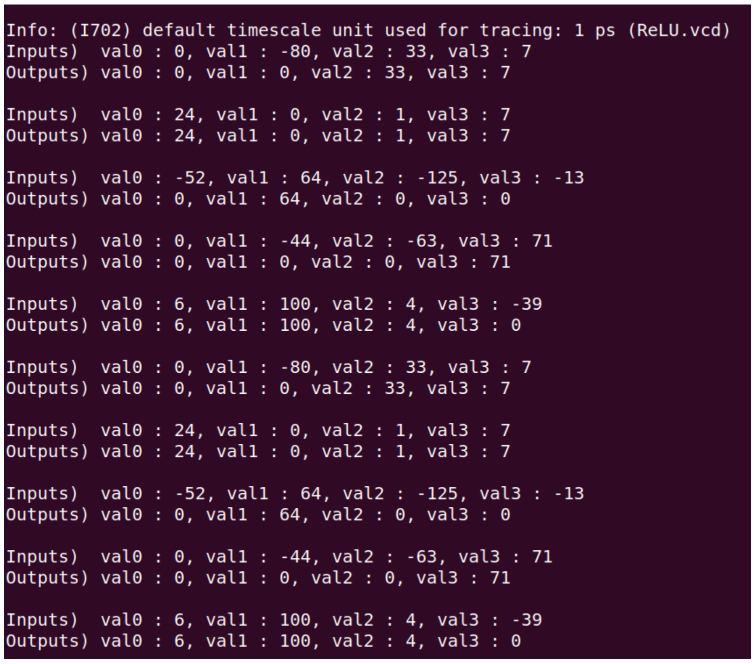


Figure 4. Verification of MaxPool module(Waveform)

As we can see the results are as expected.

ReLU Block:

The ReLU module is a combinational module with 4 inputs and 4 outputs and the output corresponded to the negative inputs are set to zero.



```
Info: (I702) default timescale unit used for tracing: 1 ps (ReLU.vcd)
Inputs)  val0 : 0, val1 : -80, val2 : 33, val3 : 7
Outputs) val0 : 0, val1 : 0, val2 : 33, val3 : 7

Inputs)  val0 : 24, val1 : 0, val2 : 1, val3 : 7
Outputs) val0 : 24, val1 : 0, val2 : 1, val3 : 7

Inputs)  val0 : -52, val1 : 64, val2 : -125, val3 : -13
Outputs) val0 : 0, val1 : 64, val2 : 0, val3 : 0

Inputs)  val0 : 0, val1 : -44, val2 : -63, val3 : 71
Outputs) val0 : 0, val1 : 0, val2 : 0, val3 : 71

Inputs)  val0 : 6, val1 : 100, val2 : 4, val3 : -39
Outputs) val0 : 6, val1 : 100, val2 : 4, val3 : 0

Inputs)  val0 : 0, val1 : -80, val2 : 33, val3 : 7
Outputs) val0 : 0, val1 : 0, val2 : 33, val3 : 7

Inputs)  val0 : 24, val1 : 0, val2 : 1, val3 : 7
Outputs) val0 : 24, val1 : 0, val2 : 1, val3 : 7

Inputs)  val0 : -52, val1 : 64, val2 : -125, val3 : -13
Outputs) val0 : 0, val1 : 64, val2 : 0, val3 : 0

Inputs)  val0 : 0, val1 : -44, val2 : -63, val3 : 71
Outputs) val0 : 0, val1 : 0, val2 : 0, val3 : 71

Inputs)  val0 : 6, val1 : 100, val2 : 4, val3 : -39
Outputs) val0 : 6, val1 : 100, val2 : 4, val3 : 0
```

Figure 5. Verification of ReLU module

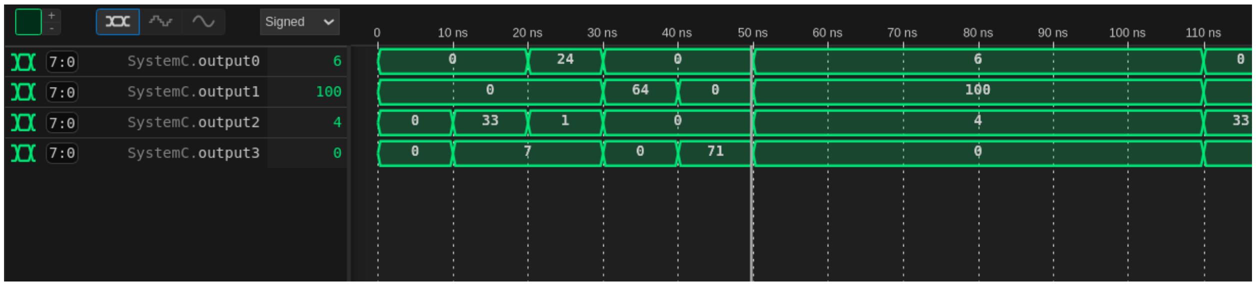


Figure 6. Verification of ReLU module (Waveform)

Convolutional Block:

This block is the heart of the design. It is a sequential module with clock and reset signals. After receiving a complete positive pulse on its start, it reads the image values and extract different features from it base on the pattern it is looking for. In this part we have only implemented the functionality of this block regardless of its hardware design.

The Kernel Values and bias of each block is determined by generic parameters.

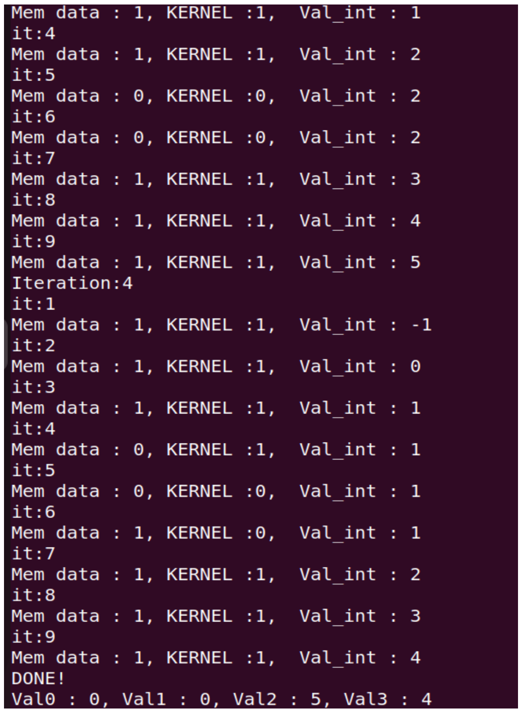

```
Mem data : 1, KERNEL :1,  Val_int : 1
it:4
Mem data : 1, KERNEL :1,  Val_int : 2
it:5
Mem data : 0, KERNEL :0,  Val_int : 2
it:6
Mem data : 0, KERNEL :0,  Val_int : 2
it:7
Mem data : 1, KERNEL :1,  Val_int : 3
it:8
Mem data : 1, KERNEL :1,  Val_int : 4
it:9
Mem data : 1, KERNEL :1,  Val_int : 5
Iteration:4
it:1
Mem data : 1, KERNEL :1,  Val_int : -1
it:2
Mem data : 1, KERNEL :1,  Val_int : 0
it:3
Mem data : 1, KERNEL :1,  Val_int : 1
it:4
Mem data : 0, KERNEL :1,  Val_int : 1
it:5
Mem data : 0, KERNEL :0,  Val_int : 1
it:6
Mem data : 1, KERNEL :0,  Val_int : 1
it:7
Mem data : 1, KERNEL :1,  Val_int : 2
it:8
Mem data : 1, KERNEL :1,  Val_int : 3
it:9
Mem data : 1, KERNEL :1,  Val_int : 4
DONE!
Val0 : 0, Val1 : 0, Val2 : 5, Val3 : 4
```

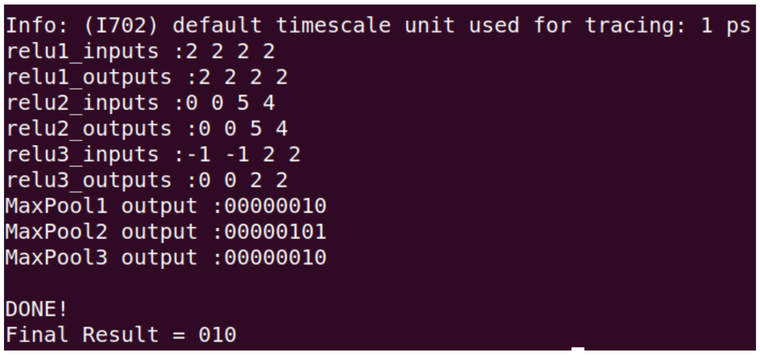
Figure 7. Verification of Convolutional module

As we can see it is correctly reading image values stored in a memory block and is performing its operation on it. In this example we have tested it on the first image data in the description of assignment and the second pattern and by performing the operation on paper, the outputs are justified.

Top-level Design:

Now that we have described all the required parts, we instantiate them in a top-module and make the Pattern Finder circuit. To instantiate the convolutional blocks, we use a generic parameter and write a parameterized code.

We have tested our design for the 6 input matrices in the description (with the values for kernels and biases in Figure.5).



```
Info: (I702) default timescale unit used for tracing: 1 ps
relu1_inputs :2 2 2 2
relu1_outputs :2 2 2 2
relu2_inputs :0 0 5 4
relu2_outputs :0 0 5 4
relu3_inputs :-1 -1 2 2
relu3_outputs :0 0 2 2
MaxPool1 output :00000010
MaxPool2 output :00000101
MaxPool3 output :00000010

DONE!
Final Result = 010
```

Figure 8. Pattern Finder result for test image 1

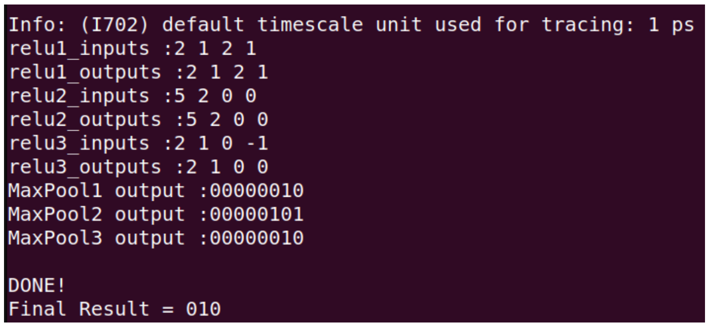
We verify the result by hand calculation.

$$\text{Conv1}: \begin{cases} -1+3=2 \\ -1+3=2 \\ -1+1+2=2 \\ -1+3=2 \end{cases} \qquad \text{Conv2}: \begin{cases} -2+2=0 \\ -2+2=0 \\ -2+3+1+3=5 \\ -2+3+3=4 \end{cases} \qquad \text{Conv3}: \begin{cases} -2+1=-1 \\ -2+1=-1 \\ -2+3+1=2 \\ -2+3+1=2 \end{cases}$$

(The calculations are for test image 1, based on the Patten matrices given in description)

As we can see the hand calculations totally justify our design outputs.

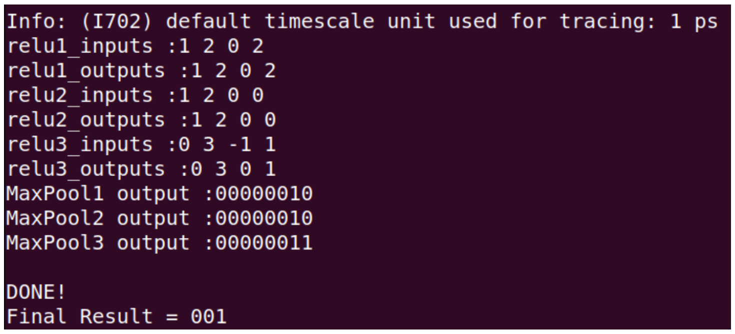
We test it similarly for the rest of test image matrices.



```
Info: (I702) default timescale unit used for tracing: 1 ps
relu1_inputs :2 1 2 1
relu1_outputs :2 1 2 1
relu2_inputs :5 2 0 0
relu2_outputs :5 2 0 0
relu3_inputs :2 1 0 -1
relu3_outputs :2 1 0 0
MaxPool1 output :00000010
MaxPool2 output :00000101
MaxPool3 output :00000010

DONE!
Final Result = 010
```

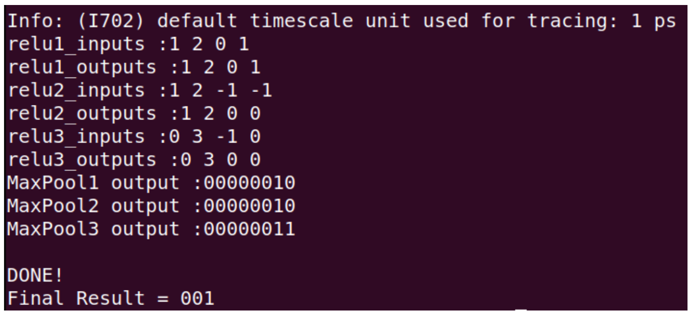
Figure 9. Pattern Finder result for test image 2



```
Info: (I702) default timescale unit used for tracing: 1 ps
relu1_inputs :1 2 0 2
relu1_outputs :1 2 0 2
relu2_inputs :1 2 0 0
relu2_outputs :1 2 0 0
relu3_inputs :0 3 -1 1
relu3_outputs :0 3 0 1
MaxPool1 output :00000010
MaxPool2 output :00000010
MaxPool3 output :00000011

DONE!
Final Result = 001
```

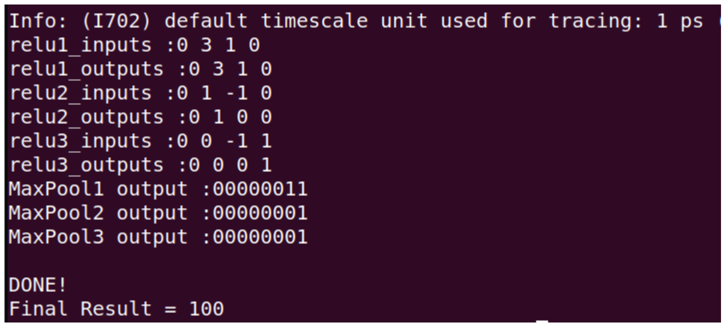
Figure 10. Pattern Finder result for test image3



```
Info: (I702) default timescale unit used for tracing: 1 ps
relu1_inputs :1 2 0 1
relu1_outputs :1 2 0 1
relu2_inputs :1 2 -1 -1
relu2_outputs :1 2 0 0
relu3_inputs :0 3 -1 0
relu3_outputs :0 3 0 0
MaxPool1 output :00000010
MaxPool2 output :00000010
MaxPool3 output :00000011

DONE!
Final Result = 001
```

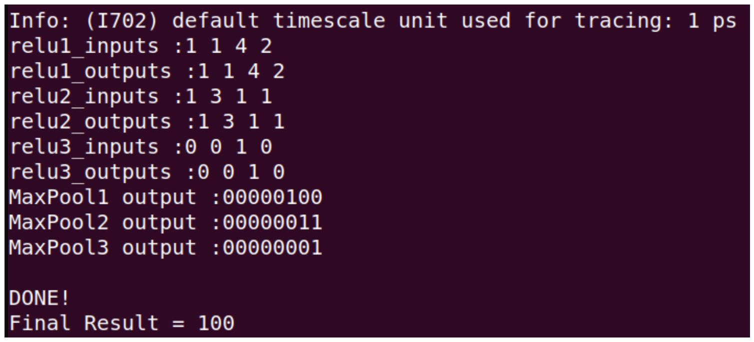
Figure 11. Pattern Finder result for test image4



```
Info: (I702) default timescale unit used for tracing: 1 ps
relu1_inputs :0 3 1 0
relu1_outputs :0 3 1 0
relu2_inputs :0 1 -1 0
relu2_outputs :0 1 0 0
relu3_inputs :0 0 -1 1
relu3_outputs :0 0 0 1
MaxPool1 output :00000011
MaxPool2 output :00000001
MaxPool3 output :00000001

DONE!
Final Result = 100
```

Figure 12. Pattern Finder result for test image5



```
Info: (I702) default timescale unit used for tracing: 1 ps
relu1_inputs :1 1 4 2
relu1_outputs :1 1 4 2
relu2_inputs :1 3 1 1
relu2_outputs :1 3 1 1
relu3_inputs :0 0 1 0
relu3_outputs :0 0 1 0
MaxPool1 output :00000100
MaxPool2 output :00000011
MaxPool3 output :00000001

DONE!
Final Result = 100
```

Figure 13. Pattern Finder result for test image6

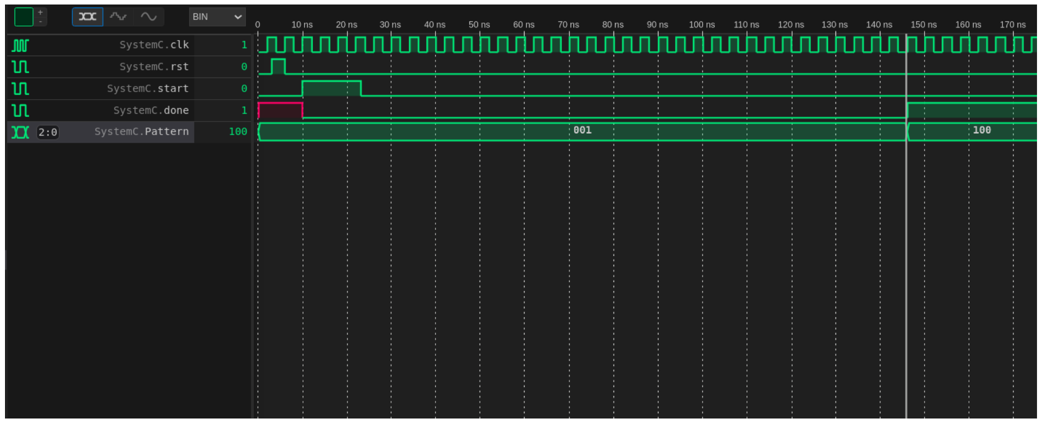


Figure 14. Pattern Finder result for test image6 (Waveform)

As we can see in the waveform the module starts its work by a complete positive pulse on clock, and when it is finished, done signal is asserted.

Also notice that the design is cycle accurate and mimic the clock cycles required for performing the operation on input image.

• RT level Model:

In this part we design the RT level model of every component (also design FSM for the sequential modules).

RLT Design:

Result Module:

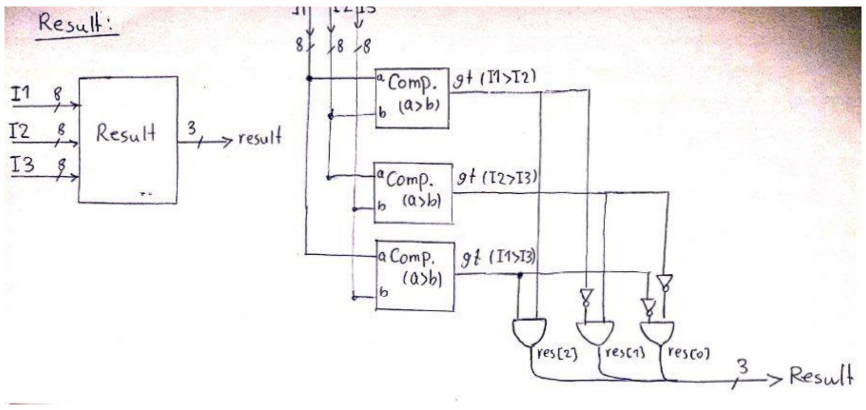


Figure 15. RTL design of Result module

As we can see we used three comparators for comparing the three inputs 2 by 2 so that we can find the largest value between them and then by using a simple logic asserted one in that output position.

We have written the SystemC description of and gate, not gate and comparator and created the top-module by instantiating them. Also note that each of the modules work concurrent to each other.

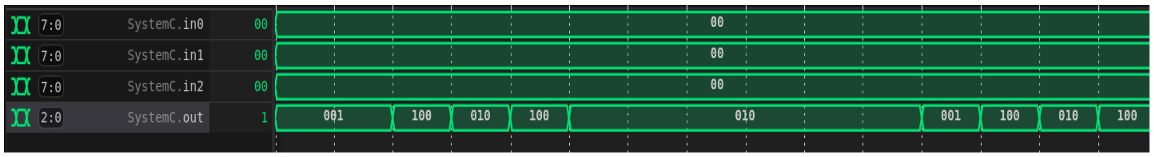


Figure 16. Verification of Result module (RTL)

We can justify it by the output of BFM model.

MaxPool module:

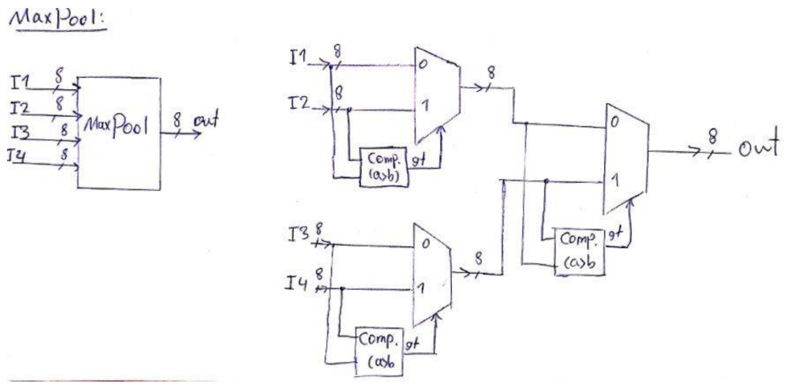
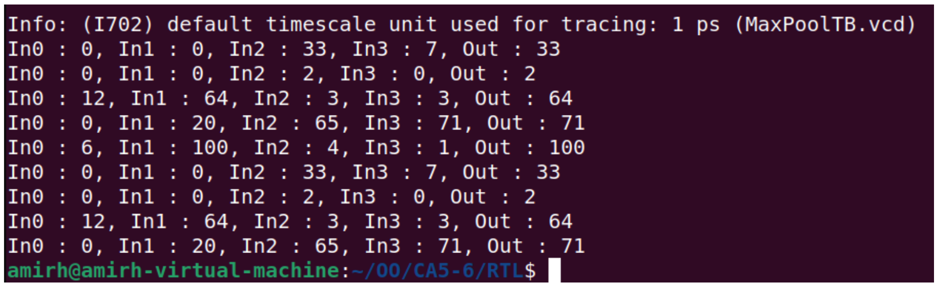


Figure 17. RTL design of MaxPool module

The RTL design of the MaxPool block is consisted of three comparators, used in 2 layer of logic to find the largest value between four input values. Here is the verification of this module.



```
Info: (I702) default timescale unit used for tracing: 1 ps (MaxPoolTB.vcd)
In0 : 0, In1 : 0, In2 : 33, In3 : 7, Out : 33
In0 : 0, In1 : 0, In2 : 2, In3 : 0, Out : 2
In0 : 12, In1 : 64, In2 : 3, In3 : 3, Out : 64
In0 : 0, In1 : 20, In2 : 65, In3 : 71, Out : 71
In0 : 6, In1 : 100, In2 : 4, In3 : 1, Out : 100
In0 : 0, In1 : 0, In2 : 33, In3 : 7, Out : 33
In0 : 0, In1 : 0, In2 : 2, In3 : 0, Out : 2
In0 : 12, In1 : 64, In2 : 3, In3 : 3, Out : 64
In0 : 0, In1 : 20, In2 : 65, In3 : 71, Out : 71
amirh@amirh-virtual-machine:~/OO/CA5-6/RTL$
```

Figure 18. Verification of MaxPool module (RTL)

ReLU Module:

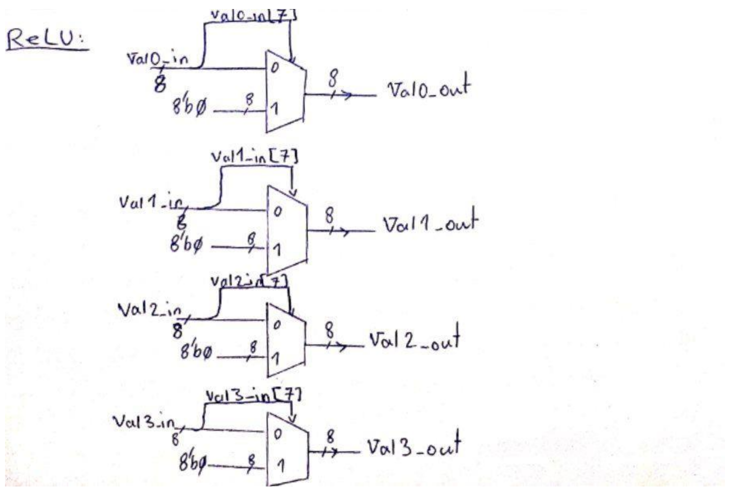
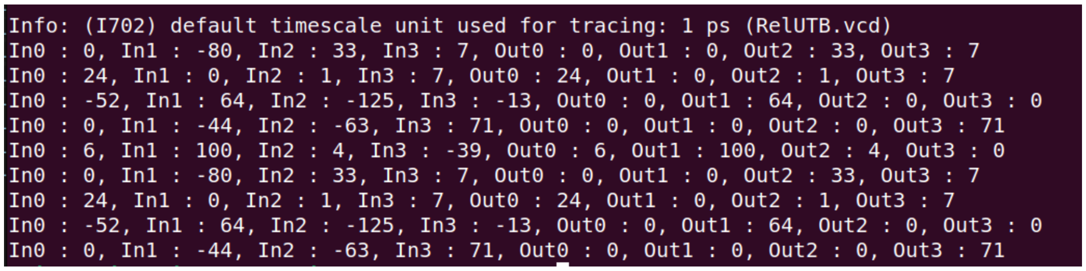


Figure 19. RTL design of ReLU module

The ReLU module should set the output to zero for the negative inputs. By using the sign-bit of each input as the multiplexer inputs we can find out whether it is negative or not.



```
Info: (I702) default timescale unit used for tracing: 1 ps (RelUTB.vcd)
In0 : 0, In1 : -80, In2 : 33, In3 : 7, Out0 : 0, Out1 : 0, Out2 : 33, Out3 : 7
In0 : 24, In1 : 0, In2 : 1, In3 : 7, Out0 : 24, Out1 : 0, Out2 : 1, Out3 : 7
In0 : -52, In1 : 64, In2 : -125, In3 : -13, Out0 : 0, Out1 : 64, Out2 : 0, Out3 : 0
In0 : 0, In1 : -44, In2 : -63, In3 : 71, Out0 : 0, Out1 : 0, Out2 : 0, Out3 : 71
In0 : 6, In1 : 100, In2 : 4, In3 : -39, Out0 : 6, Out1 : 100, Out2 : 4, Out3 : 0
In0 : 0, In1 : -80, In2 : 33, In3 : 7, Out0 : 0, Out1 : 0, Out2 : 33, Out3 : 7
In0 : 24, In1 : 0, In2 : 1, In3 : 7, Out0 : 24, Out1 : 0, Out2 : 1, Out3 : 7
In0 : -52, In1 : 64, In2 : -125, In3 : -13, Out0 : 0, Out1 : 64, Out2 : 0, Out3 : 0
In0 : 0, In1 : -44, In2 : -63, In3 : 71, Out0 : 0, Out1 : 0, Out2 : 0, Out3 : 71
```

Figure 20. Verification of ReLU module (RTL)

The functionality of the module is as expected.

Convolution Block:

Convolution module is a sequential block which is consist of a data-path and a control unit.

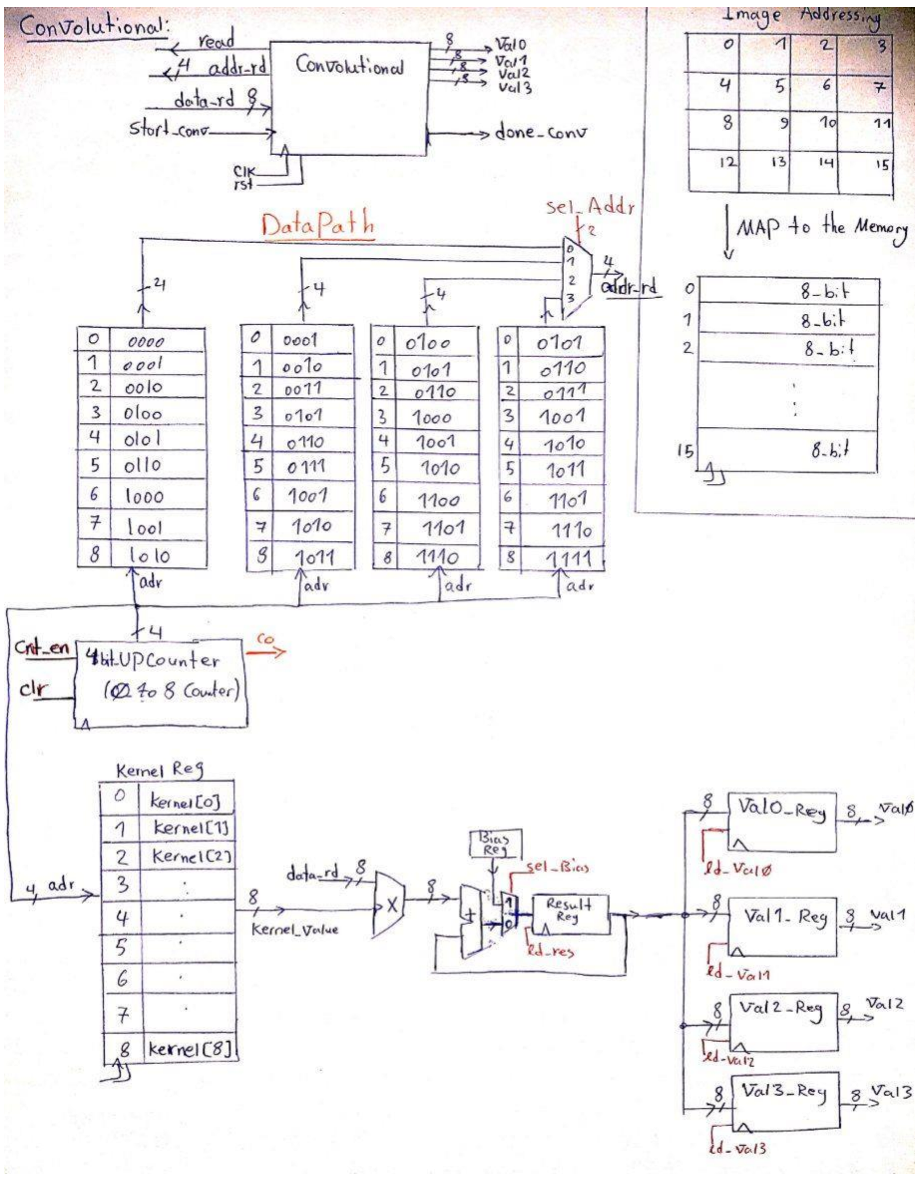


Figure 21. Data-path of Convolution module

As we can see in figure 21 we should design a logic to create the address corresponding to each element of the image matric for calculation of each one of the 4 outputs. This is done by storing addresses in 4 separate ROM (each for one of the outputs).

The Kernel values are also stored inside a ROM so we can access them by their address during the calculations.

As it was mentioned in the description we were allowed to use only one adder and one multiplier in the design, so we used a MAC block to perform the multiply operations and accumulate the partial result in a register called "Result Reg".

Also we have used a 4-bit upcounter to count from 0 to 8. This is used for the purpose of addressing ROMS to get the proper address. The carry out of the counter became '1' when it reaches 8 (1000_B).

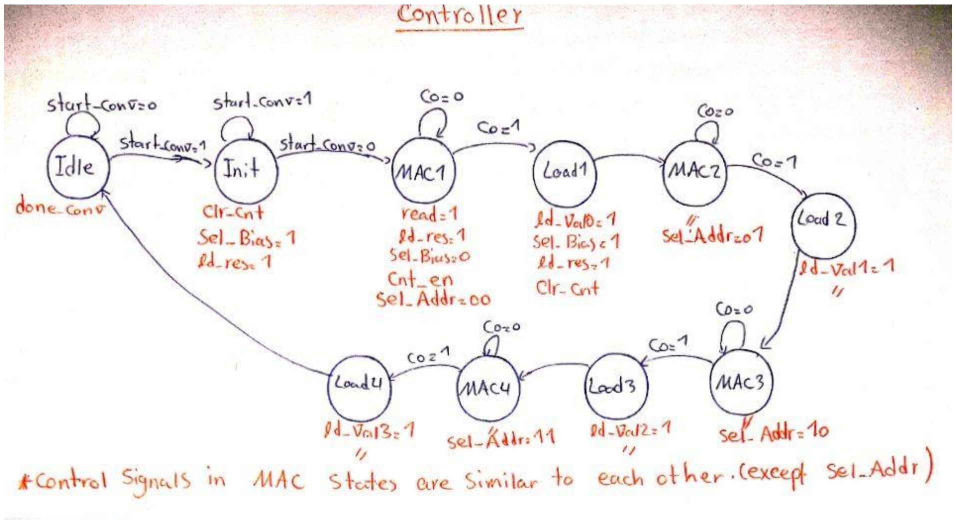


Figure 22. Control Unit of Convolution module

In the first state (Idle) "done_conv" is asserted and we are waiting for a complete positive pulse on the start. After that 4 iterations of <u>MAC</u> and <u>Load</u> states get start.

In MAC states, 'read' signal is asserted so that we read the image data corresponding to the generated address. In the first MAC, 'sel_Addr' is set to "00" so we use the addresses in the first address ROM and for the next MACs it gets increased by one. Also the 'Cnt_en' signal is asserted so the counter will count up and we stay in this state for 9 clock cycles. After that, the carry out will be asserted and we will move to the Load state of that operation.

In Load states, the proper output register load signal will be asserted (for the first operation, load signal of first register will be asserted and so on). Also 'Sel_Bias" of the multiplexer before the 'Result Reg" will be one so that the register will be loaded with bias value before starting the computations for the next operation. The 'Clr_cnt' should also be asserted so that the counter will start addressing from zero at the beginning of the next operation.

After completing all four operations, it gets back to the Idle state and the 'done_conv' will be asserted and the circuit waits for the next pulse on its start.

Now to make sure the functionality of the Convolution module is correct, we test it with the same data that we tested the bus functional model of it in figure 7.

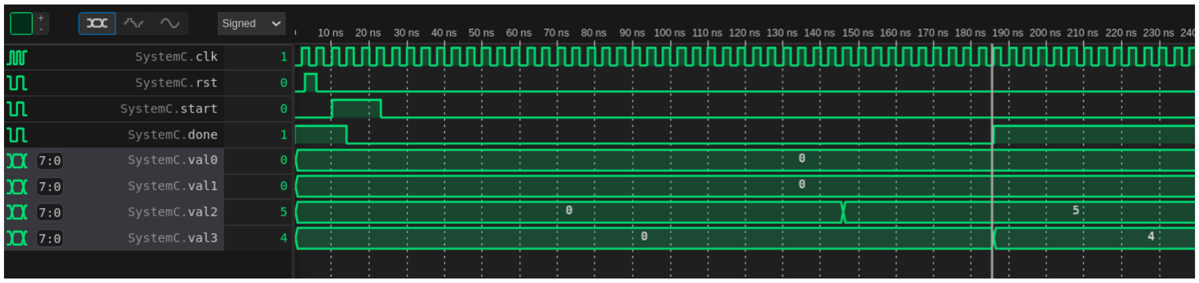


Figure 23. Verification of Convolutional module(RTL)

As we can see the outputs totally matches the result of the previous part.

Pattern Finder Circuit (Top-module):

After instantiating all the necessary blocks (memory, convolutions, ReLUs, …) the data-path of the Pattern-Finder is completed. (like figure 1)

Now we should also design a control unit for this module so that it communicates with outside (by ‘start’ and ‘done’ signal) and also assert the convolution’s blocks start signals at the right time.

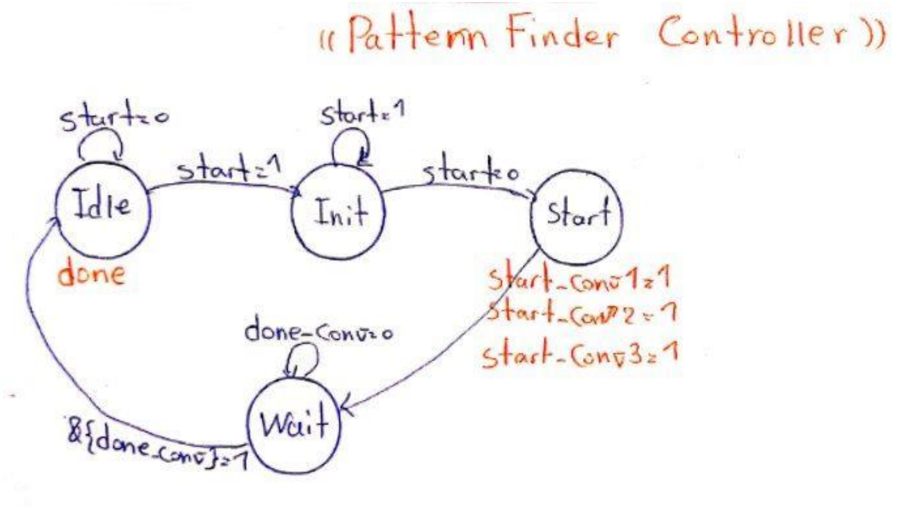


Figure 24. Control Unit of Pattern-Finder module

In the first two states, it waits to receive a complete positive pulse on its start, after that it asserts the convolutional blocks start signals to ‘1’. In the next state start signals get back to zero and it wait until all of the convolution blocks are done (all the ‘done_conv’ signals are asserted to ‘1’). Then it gets back to the Idle state and ‘done’ signal is asserted.

Now we test the final Pattern Finder module for the datasets mentioned in the description and compare the result with the BFM outputs.

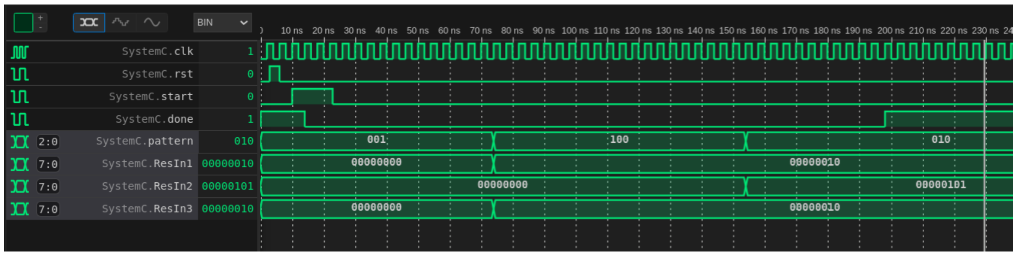


Figure 25. Pattern Finder(RTL) result for test image 1

As we can see the detected pattern is “010” which is the same as the BFM output in figure 8.

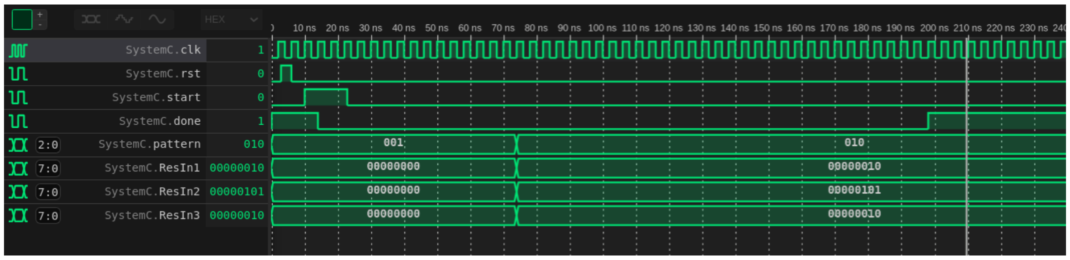


Figure 26. Pattern Finder(RTL) result for test image 2

The detected pattern is “010” (same as BFM in figure 9).

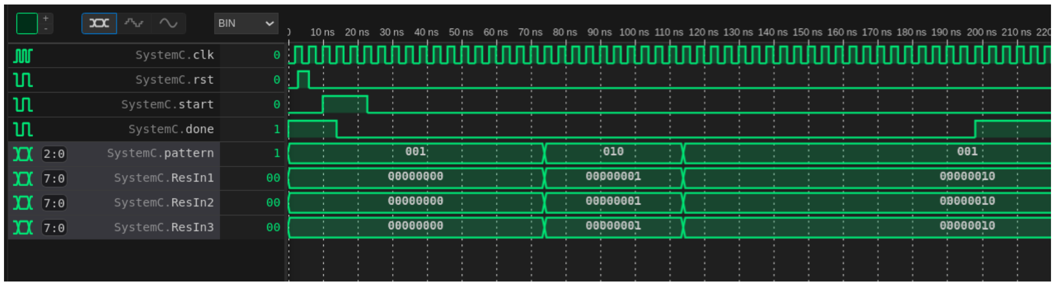


Figure 27. Pattern Finder(RTL) result for test image 3

The detected pattern is “001” (same as BFM in figure 10).

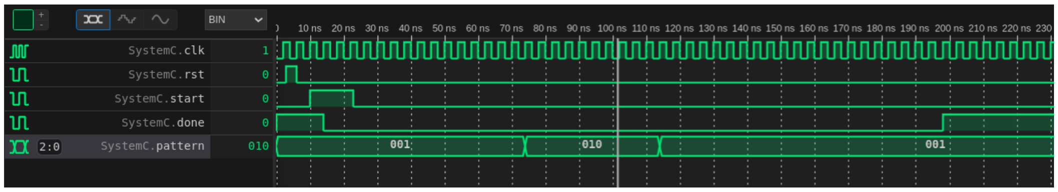


Figure 28. Pattern Finder(RTL) result for test image 4

The detected pattern is “001” (same as BFM in figure 11).

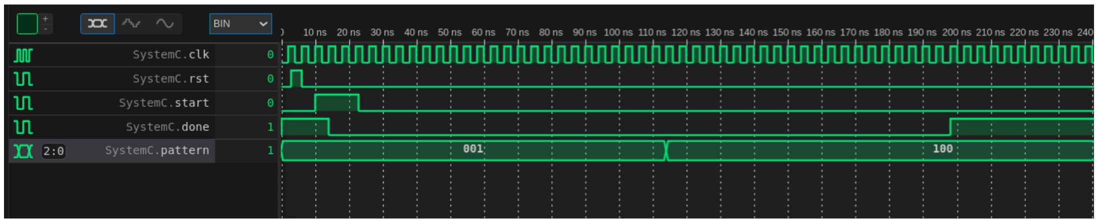


Figure 29. Pattern Finder(RTL) result for test image 5

The detected pattern is "100" (same as BFM in figure 12).

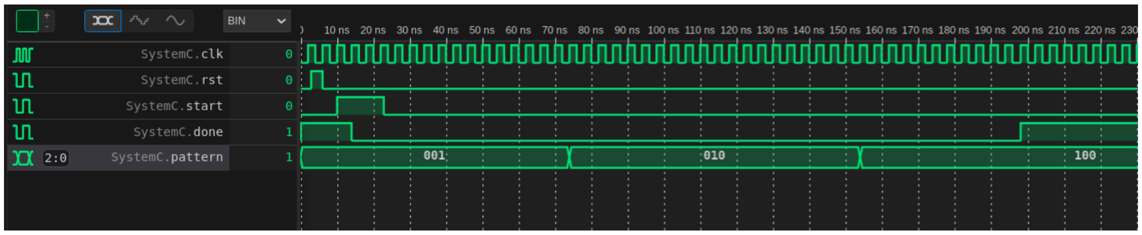


Figure 30. Pattern Finder(RTL) result for test image 6

The detected pattern is "100" (same as BFM in figure 13).

All the results are justified by the Bus Functional Model of the circuit.