University of Tehran

Electrical and Computer Engineering Department

ECE (8101) 342

Object Oriented Modeling of Electronic Circuits – Spring 1402-03

Computer Assignment 5,6

Zainalabedin Navabi

Name: Amirhesam Jafari rad

SID: 810100247

RT Level Modeling:

For implementing Pattern Finder Circuit in RT level, each component should be implemented individually as showing separate modules is requested. So I created .h files as many as needed for corresponding modules. Components which I used in my design are: Img_RAM, Convolution, Adder, Multiplier, ReLU, MaxPool, and Result which each one are tested using the testbench available in their .h file. The Adder and Multiplier is implemented inside the Convolution unit.

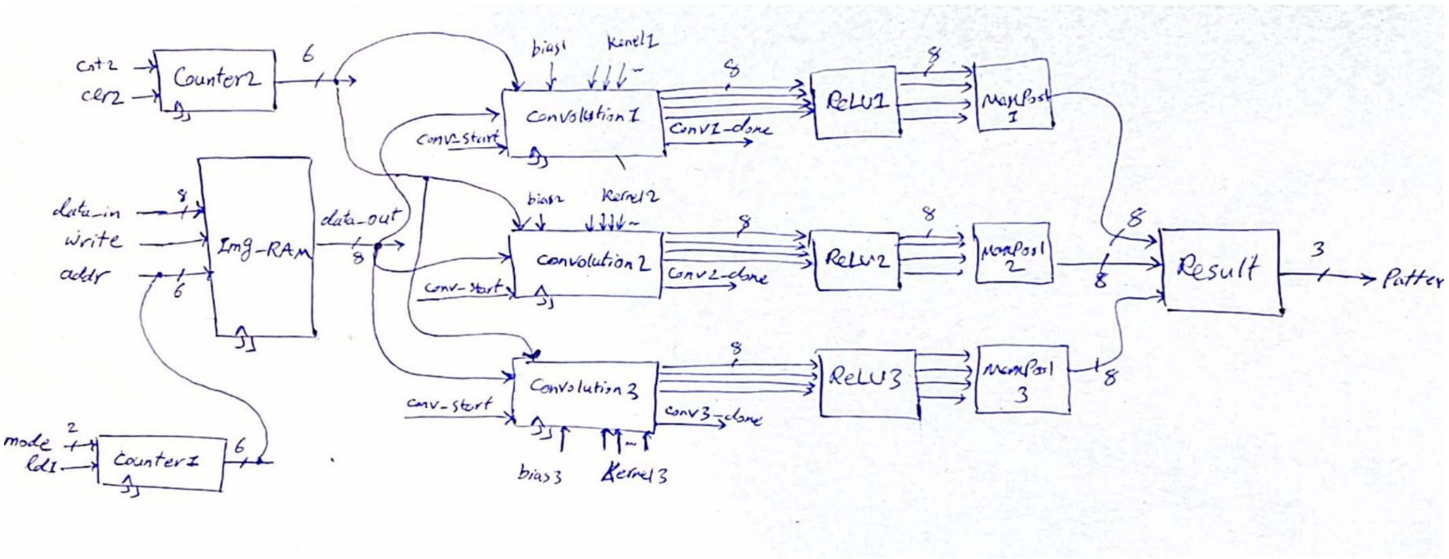
Algorithm:

I've done this phase in the order below: my memory has a 8bit output so it couldn't put all the 4*4 input to its output simultaneously, therefore it passes each 8bit element one by one to the output. The point is the addressing in which, 4 state of the 4*4 memory should be derived at the output: {0,1,2,4,5,6,8,9,10}, {1,2,3,5,6,7,9,10,11}, {4,5,6,8,9,10,12,13,14}, and {5,6,7,9,10,11,13,14,15} which are state0, state1, state2, and state3 respectively. This is done using a counter namely Counter1 which has a 2bit mode signal that determine the output should behave in which state and trend. Concurrently, there is a counter, Counter2, which count how many elements of the memory is passed. Obviously, it should count up to 36, as there are 4 states in which there are 9 element (4*9 = 36). After all elements has been passed to the output, the convolution units should start their work by issuing the conv_start signal. Note that in each convolution block there is a 36-element ram, which capture each incoming element from the memory.

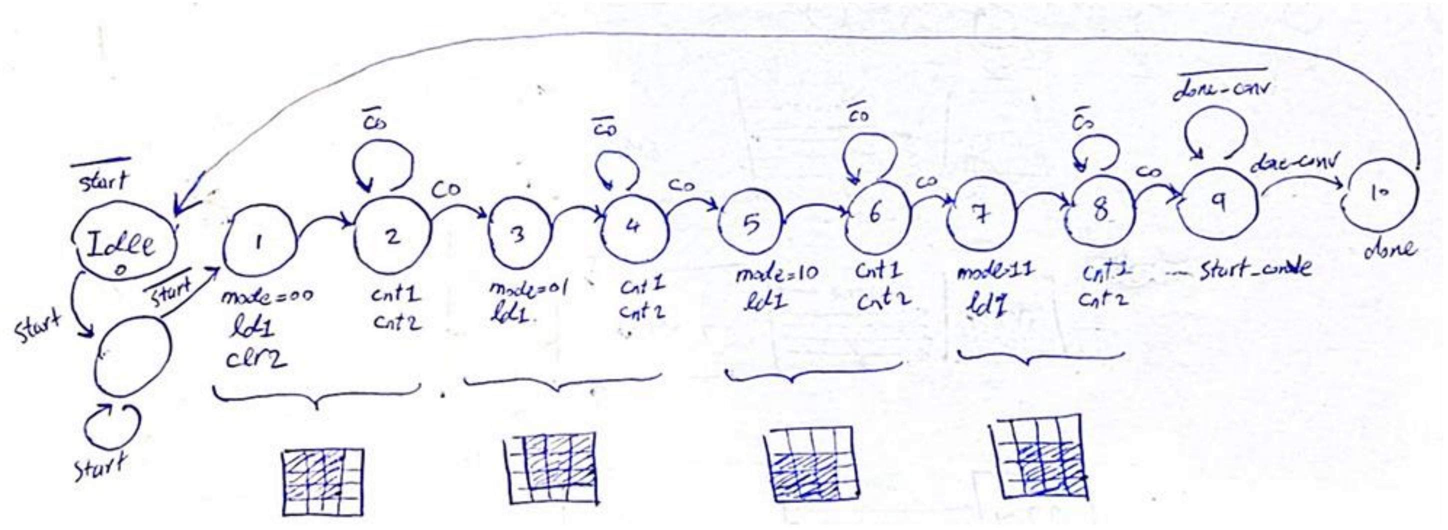
After convolution block finished their job, the conv_done signal will be issued for each convolution block and if all of them emitted a done signal, then ReLU, MaxPool, and finally Result unit comes in to detect the final pattern. Each convolution block gets its own kernel and bias value.

The data path and controller of whole design is brought below:

DataPath:



Controller:



As shown, for each 3*3 input of convolution blocks, 2 states is needed.

Also the instantiation of convolution block is done using for loops as wanted and there is a N parameter to indicate the parametric number of convolution blocks:

```
for (int i = 0; i < N; i++) {
        std::stringstream ss;
        ss << "Convolution_Inst" << i;
        Convolution_[i] = new Convolution(ss.str().c_str());
        Convolution_[i]->clk(clk);
        Convolution_[i]->start_conv(start_conv);
        Convolution_[i]->img_element(data_out);
        Convolution_[i]->kernel_index(kernel_index);
        Convolution_[i]->conv_done(conv_done[i]);
```

Tests:

Input 1

0	0	0	0
1	1	1	1
1	0	0	1
1	1	1	1

Input 2

1	1	1	0
1	0	0	0
1	1	1	0
0	0	0	1

Input 3

0	1	1	1
0	0	1	0
0	0	1	0
0	0	1	0

Input 4

0	1	1	1
0	0	1	0
0	0	1	0
0	0	0	0

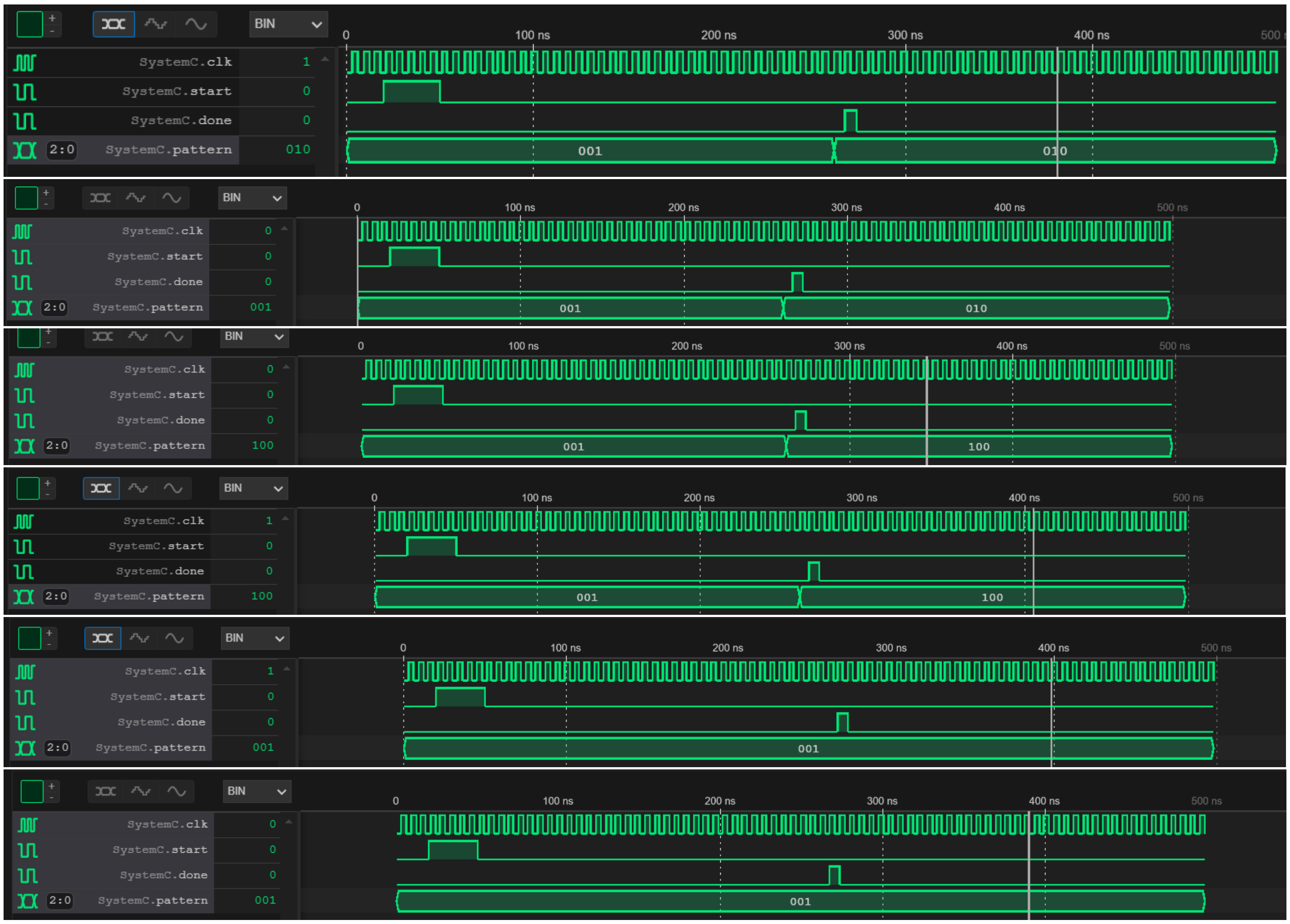
Input 5

0	0	1	0
0	1	0	1
0	0	1	0
0	0	0	0

Input 6

0	0	0	1
0	1	0	0
1	1	1	1
0	1	0	0

The corresponding output is:



```
SystemC.clk      1
SystemC.start    0
SystemC.done     0
SystemC.pattern  010
```

```
SystemC.clk      0
SystemC.start    0
SystemC.done     0
SystemC.pattern  001
```

```
SystemC.clk      0
SystemC.start    0
SystemC.done     0
SystemC.pattern  100
```

```
SystemC.clk      1
SystemC.start    0
SystemC.done     0
SystemC.pattern  100
```

```
SystemC.clk      1
SystemC.start    0
SystemC.done     0
SystemC.pattern  001
```

```
SystemC.clk      0
SystemC.start    0
SystemC.done     0
SystemC.pattern  001
```

Bus Functional Modeling:

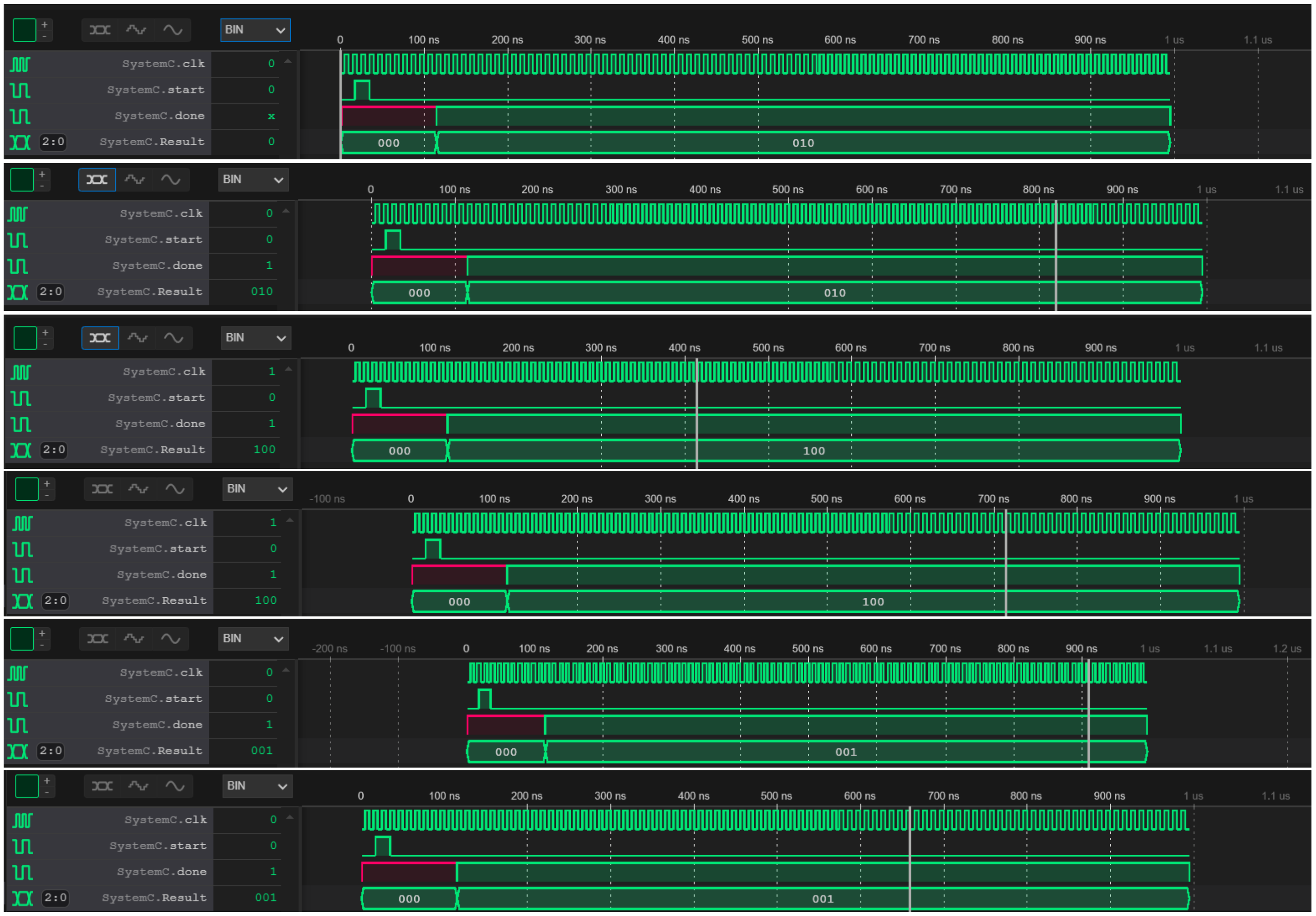
The story is by far easier in BFM as there is no need to implement individual components and every thing is just a top module named PatternFinder.h.

Algorithm:

The algorithm include the trend of reading data from memory which is done by for loops. However, the iteration of loop should follow the reading states trend. To satisfy this purpose, I found that the trends (012,345,789,…) can be done using division of iteration to 3, and plus 0, 1, 4, or 5 depending on what state we are. Other process is as it should be, without any constrains and limitations.

Tests:

The corresponding output is:



BIN
SystemC.clk 0
SystemC.start 0
SystemC.done x
2:0 SystemC.Result 0
000
010
SystemC.clk 0
SystemC.start 0
SystemC.done 1
SystemC.Result 010
000
010
SystemC.clk 1
SystemC.start 0
SystemC.done 1
SystemC.Result 100
000
100
SystemC.clk 1
SystemC.start 0
SystemC.done 1
SystemC.Result 100
000
100
SystemC.clk 0
SystemC.start 0
SystemC.done 1
SystemC.Result 001
000
001
SystemC.clk 0
SystemC.start 0
SystemC.done 1
SystemC.Result 001
000
001
0
100 ns
200 ns
300 ns
400 ns
500 ns
600 ns
700 ns
800 ns
900 ns
1 us
1.1 us