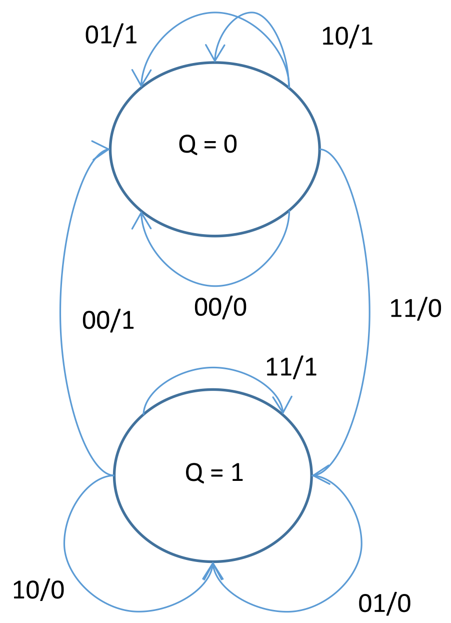
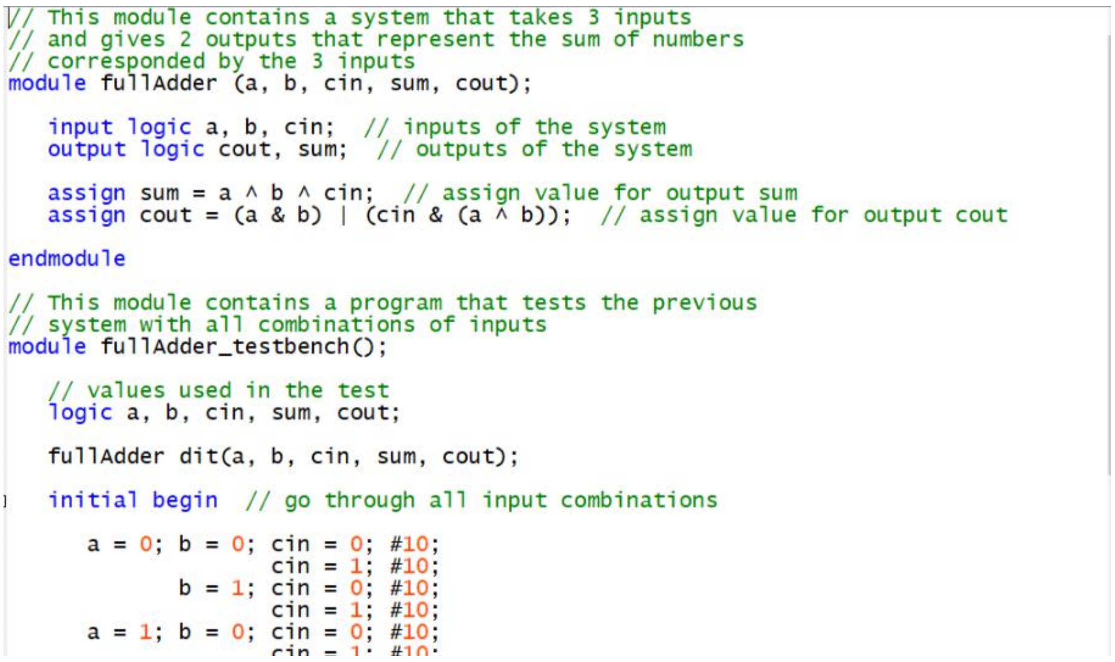
EE 371 HW1

1)

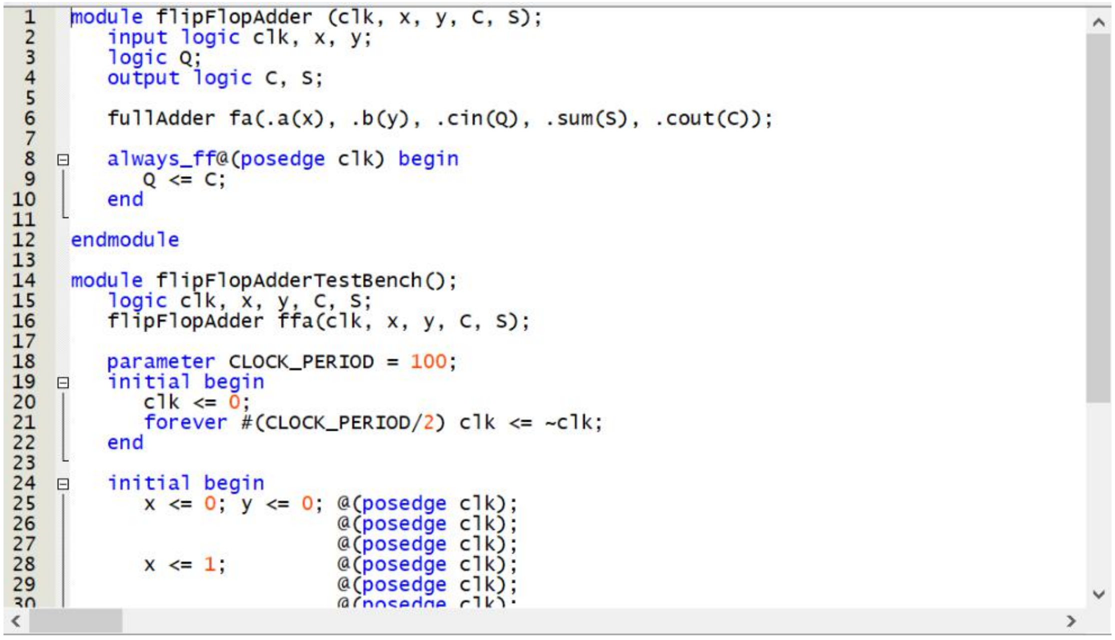


fullAdder.sv



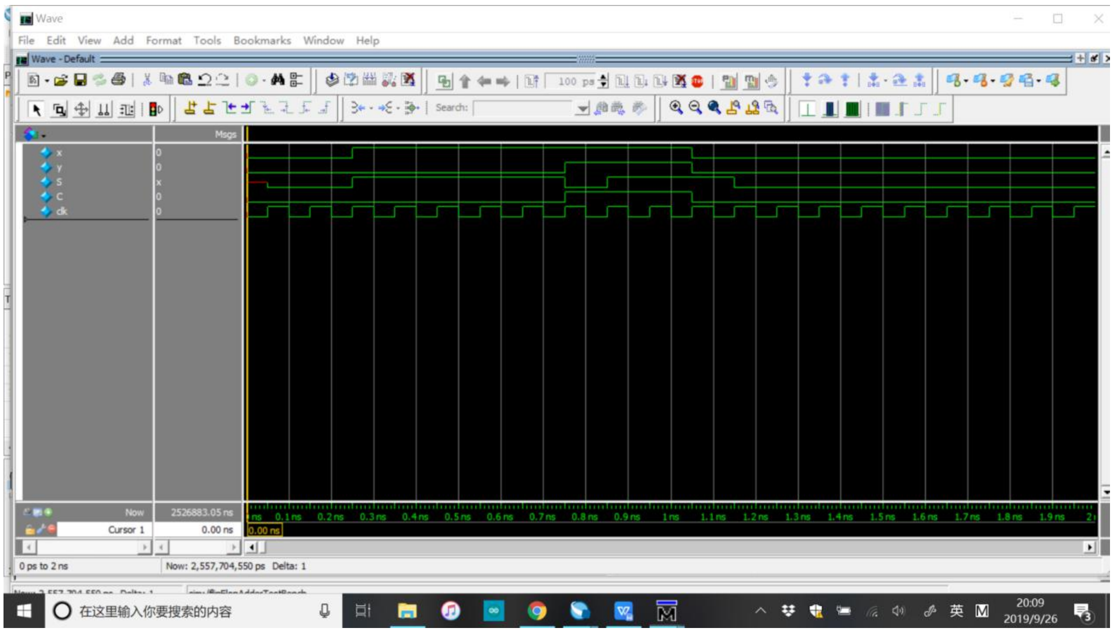
```
// This module contains a system that takes 3 inputs
// and gives 2 outputs that represent the sum of numbers
// corresponded by the 3 inputs
module fullAdder (a, b, cin, sum, cout);

   input logic a, b, cin;  // inputs of the system
   output logic cout, sum;  // outputs of the system

   assign sum = a ^ b ^ cin;  // assign value for output sum
   assign cout = (a & b) | (cin & (a ^ b));  // assign value for output cout

endmodule

// This module contains a program that tests the previous
// system with all combinations of inputs
module fullAdder_testbench();

   // values used in the test
   logic a, b, cin, sum, cout;

   fullAdder dit(a, b, cin, sum, cout);

   initial begin  // go through all input combinations

      a = 0; b = 0; cin = 0; #10;
                    cin = 1; #10;
             b = 1; cin = 0; #10;
                    cin = 1; #10;
      a = 1; b = 0; cin = 0; #10;
                    cin = 1; #10;
```

flipFlopAdder.sv



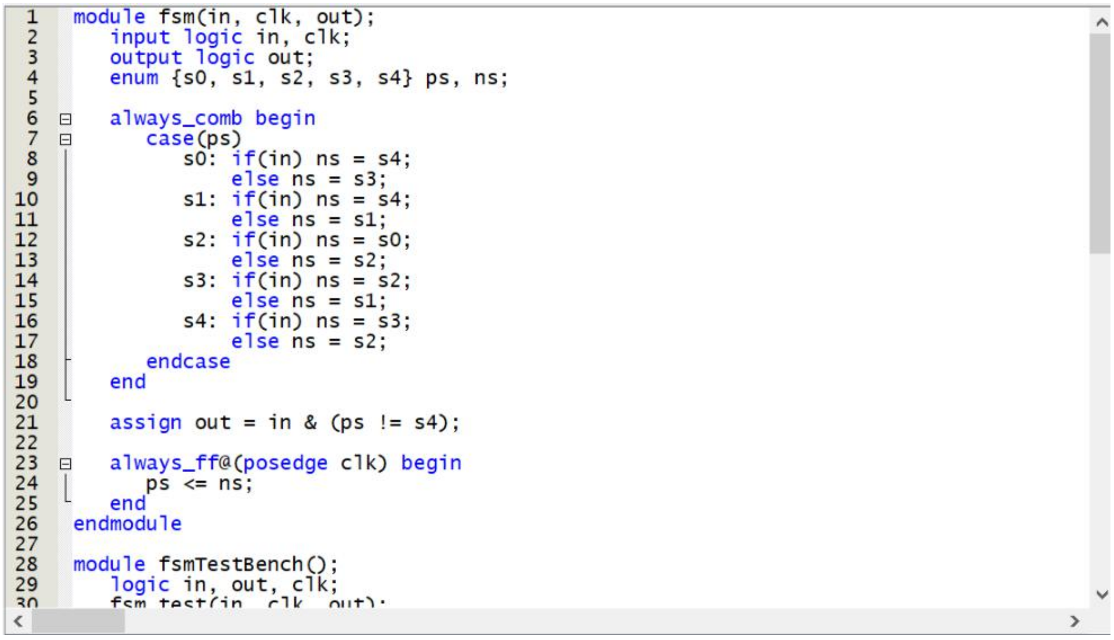
```
module flipFlopAdder (clk, x, y, C, S);
   input logic clk, x, y;
   logic Q;
   output logic C, S;

   fullAdder fa(.a(x), .b(y), .cin(Q), .sum(S), .cout(C));

   always_ff@(posedge clk) begin
      Q <= C;
   end

endmodule

module flipFlopAdderTestBench();
   logic clk, x, y, C, S;
   flipFlopAdder ffa(clk, x, y, C, S);

   parameter CLOCK_PERIOD = 100;
   initial begin
      clk <= 0;
      forever #(CLOCK_PERIOD/2) clk <= ~clk;
   end

   initial begin
      x <= 0; y <= 0; @(posedge clk);
                      @(posedge clk);
                      @(posedge clk);
      x <= 1;         @(posedge clk);
                      @(posedge clk);
                      @(posedge clk);
```

Simulation



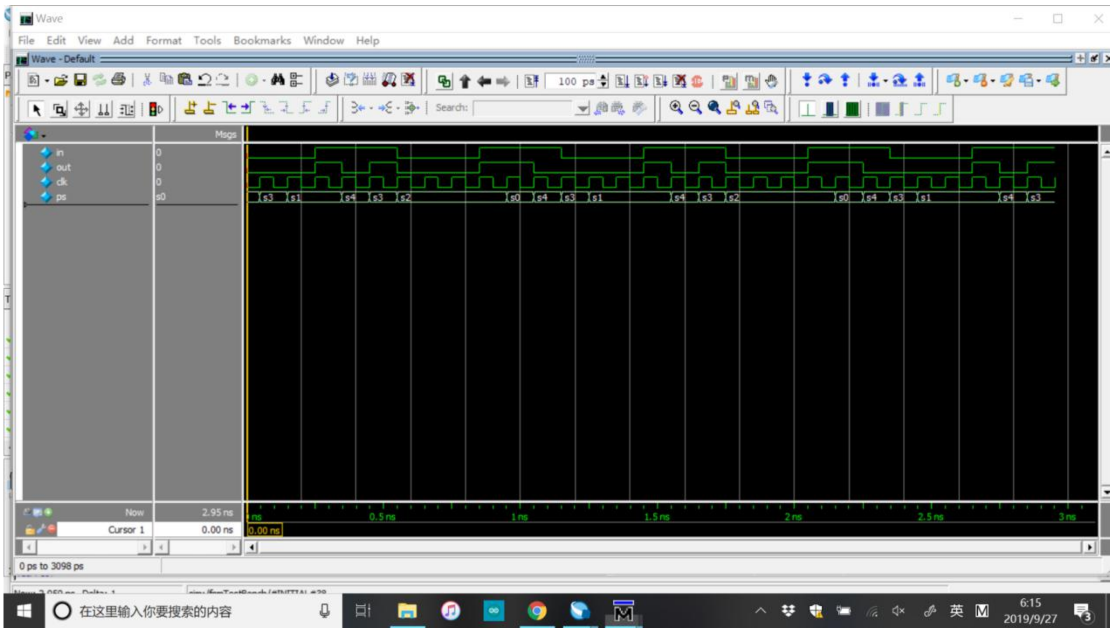
2)

fsm.sv

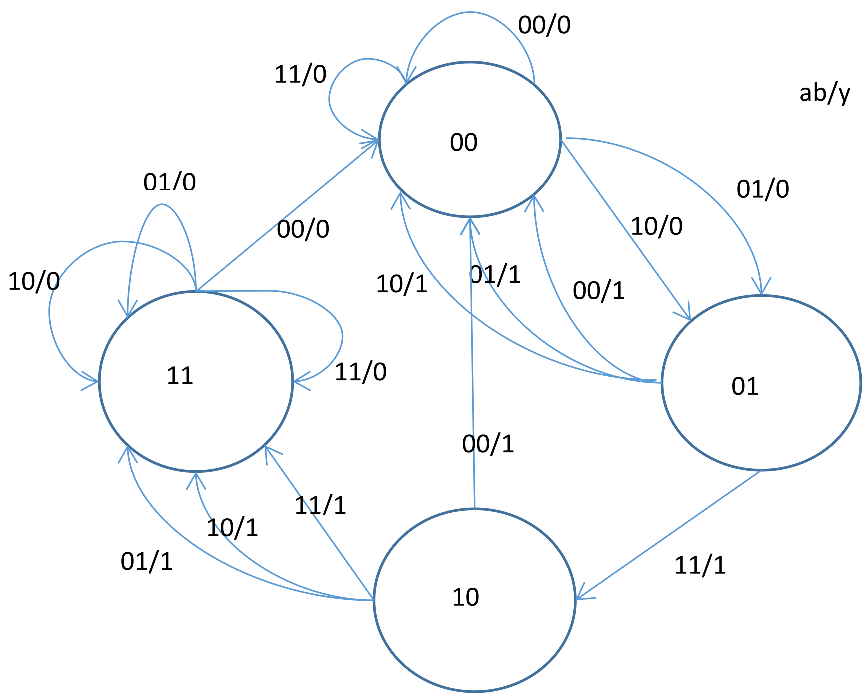


```
module fsm(in, clk, out);
   input logic in, clk;
   output logic out;
   enum {s0, s1, s2, s3, s4} ps, ns;

   always_comb begin
      case(ps)
         s0: if(in) ns = s4;
             else ns = s3;
         s1: if(in) ns = s4;
             else ns = s1;
         s2: if(in) ns = s0;
             else ns = s2;
         s3: if(in) ns = s2;
             else ns = s1;
         s4: if(in) ns = s3;
             else ns = s2;
      endcase
   end

   assign out = in & (ps != s4);

   always_ff@(posedge clk) begin
      ps <= ns;
   end
endmodule

module fsmTestBench();
   logic in, out, clk;
   fsm test(in, clk, out);
```

Simulation


Wave
File Edit View Add Format Tools Bookmarks Window Help
Wave - Default
100 ps
Search:
Msgs
in 0
out 0
clk 0
ps s0
s3 s1 s4 s3 s2 s0 s4 s3 s1 s4 s3 s2 s0 s4 s3 s1 s4 s3
Now 2.95 ns
Cursor 1 0.00 ns
0 ns 0.5 ns 1 ns 1.5 ns 2 ns 2.5 ns 3 ns
0 ps to 3098 ps
在这里输入你要搜索的内容
英
6:15
2019/9/27

3)

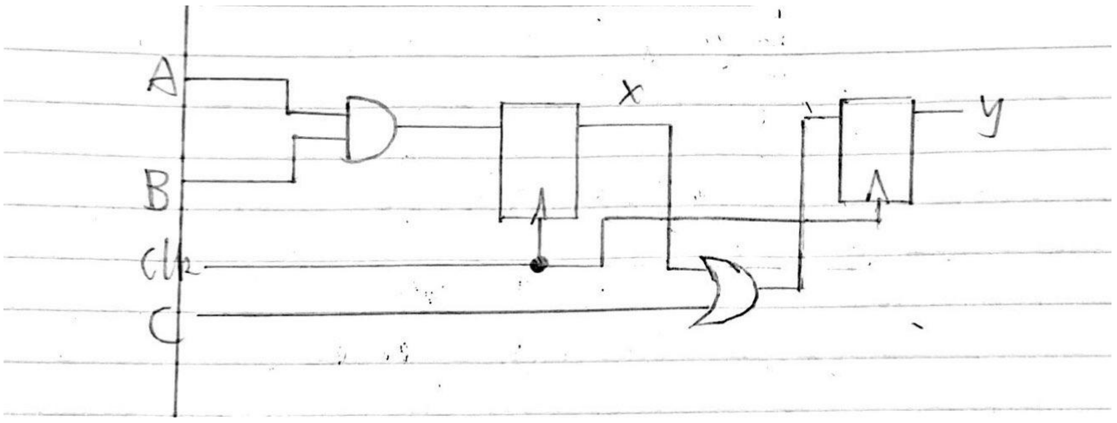


00/0
11/0
ab/y
00
01/0
01/0
10/0
00/0
10/0
10/1
01/1
00/1
11
11/0
01
00/1
11/1
10/1
01/1
11/1
10

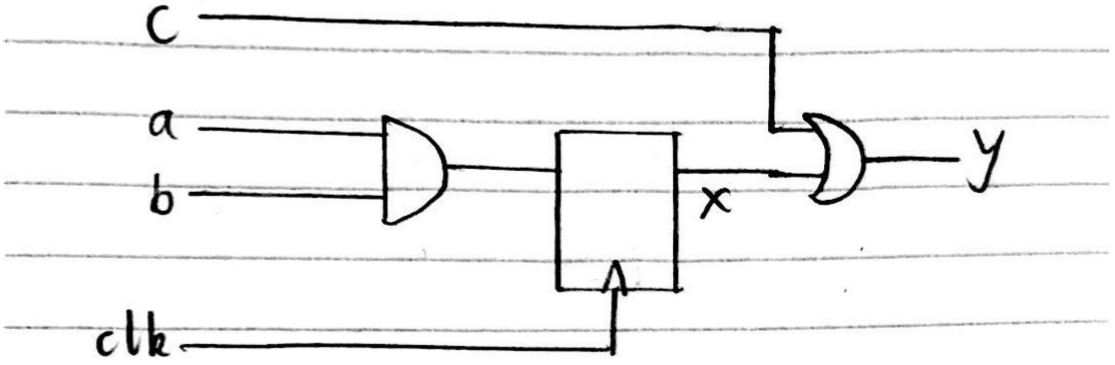
4)

Code1:

<=:

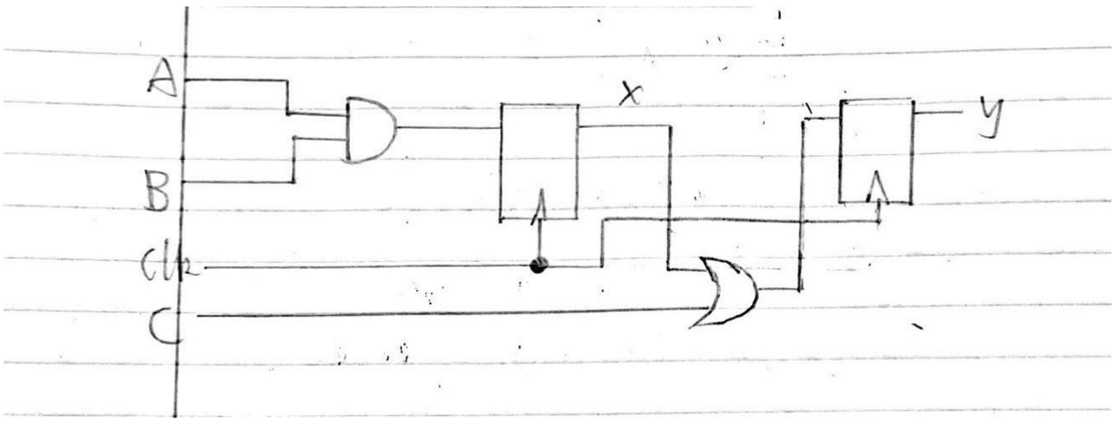


=:

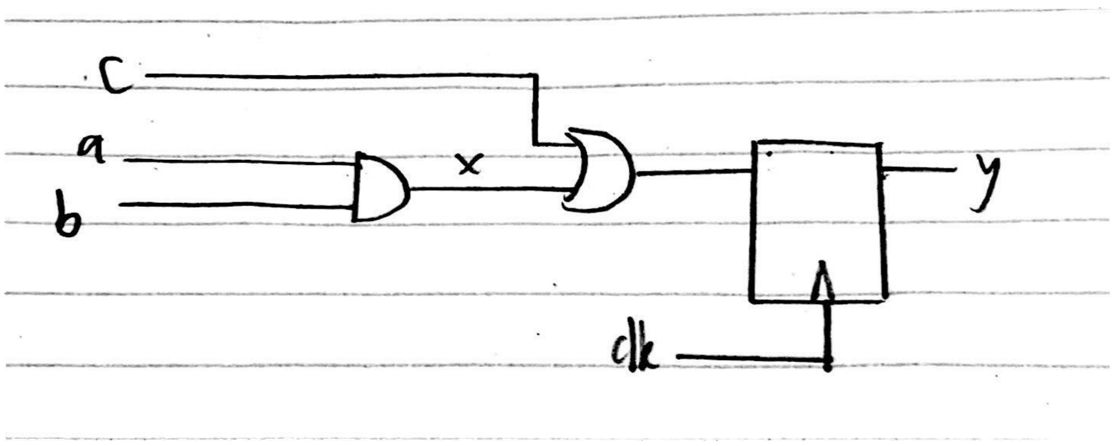


Code2:

<=:



=:



The two modules would have the same function when non-blocking assignment is used because in non-blocking assignment values are assigned simultaneously. However, they would have different functions under blocking assignment because in blocking assignment values are assigned immediately after the statement.

Feedback & Comment

I had some trouble writing the SystemVerilog codes because I kinda forget how to write them. And I also found that I can write finite state machines with logic rather than enum. I also struggled with problem 4 because I am not used to using blocking assignment with in always_ff.