```
// decoder : Q 1
module dec8x3(input1, input2, out);

input [1:0] input1 ;
input [5:0] input2 ;
output [2:0] out ;
reg [2:0] out ;

always @(input1 or input2)
begin
case (input1)
        0: out=2;
        1: out=6;
        2:
         case (input2[3:0])
                0 : out=2;
                2 : out=6;
                4 : out=0;
                5 : out=1;
                10: out=7;
                default: out=’bx;
        endcase
        default: out=’bx;
endcase
end

endmodule


// Alu 32  : Q 2
module alu32(a, b, opcode, r, v, z);
input [31:0]a;
input [31:0]b;
input [2:0]opcode;
output [31:0]r;
output v, z;
reg [31:0]r;
reg v, z;

always @ (a or b or opcode)
begin
case(opcode)
        0: begin
        // bitwise AND : no overflow possible
                r=a&b;
                v=1’b0;
        end
        1:begin
        // bitwise OR : no overflow possible
                r=a&b;
                r=a|b;
                v=1’b0;
        end
        2:begin
        // Assuming the inputs to be in 2’s complement notation:
        // can just add them irrespective of sign
        // Overflow is detected when a and b have the same sign,
        // and r has the opposite sign of a (or b)
                r=a+b;
                if ((a[31] == b[31])&&(a[31]!=r[31]))
                        v=1’b1;
```

```
                else
                        v=1'b0;
        end
        3:begin
        // As above : We can subtract normally..
        // Overflow detection : a and b have the OPPOSITE signs,
        // and r has sign OPPOSITE to a (or same as b)
                r=a-b;
                if ((a[31] != b[31])&&(a[31]!=r[31]))
                        v=1'b1;
                else
                        v=1'b0;
        end
        4:begin
        // Here again we have to check sign. If the sign is the same,
        // then r = (a < b) (NOTE that this works when a and b are positive,
        // as well as when bith are negative.
        // But if signs are opposite, then if a is negative, r is 1, else r is 0
                if (a[31]==b[31])
                // a and b are of the same sign
                        r = (a< b);
                else
                // a and b are not of the same sign
                        r = a[31];
        end
        default:begin
                r='bx;
                v='bx;
        end
endcase
// now handle the setting of z
if (r==32'd0)
        z=1'b1;
else
        z=1'b0;
end
endmodule



// satc : Q 3
module satc(in, state, clk);
input in, clk;

output [1:0] state ;

reg [1:0] state, nextstate;

// always good to explicitly name states as Parameters
parameter weak_taken=1, strong_taken=0, weak_not_taken=2, strong_not_taken=3;

initial begin
  state = weak_taken;
end

// the stuff that happens at the clock
always @(posedge clk) begin
  state <= nextstate; // NOTE the delayed assignment!
end
```

```
// the "combinatorial" logic
always @(in or state) begin
  //default
  nextstate = state;
// in = 0 means Not taken; 1 means taken
  case (state)
        strong_taken: if (in==0)
                nextstate = weak_taken;
        weak_taken: if (in==1)
                nextstate = strong_taken;
           else nextstate = strong_not_taken;
        weak_not_taken: if (in==1)
                nextstate = strong_taken;
           else nextstate = strong_not_taken;
        strong_not_taken: if (in==1)
                nextstate = weak_not_taken;
  endcase
end

endmodule
```