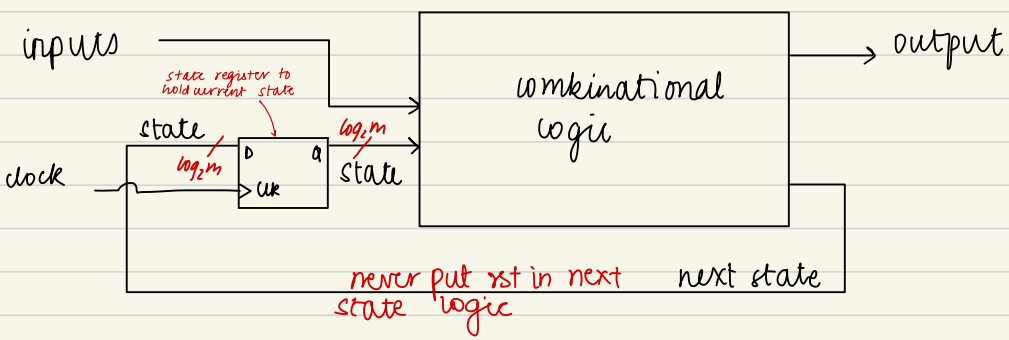
- state register
- combinational logic to determine next state
- logic to compute the output

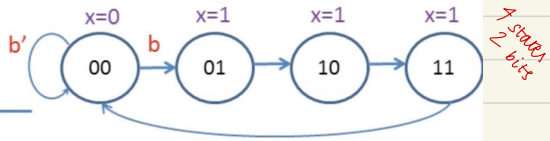
FSM → circuit requirements



- mapping FSM : m states ⇒ $(\log_2 m)$ bits to encode

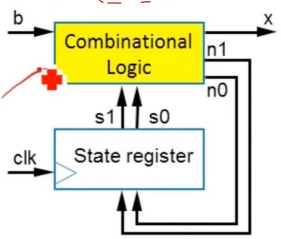
↑ point is to uniquely define each state using minimum number of bits (not necessarily in ascending order, just unique

- The transition table:



	Inputs			Outputs		
	s1	s0	b	x	n1	n0
off	0	0	0	0	0	0
	0	0	1	0	0	1
on1	0	1	0	1	1	0
	0	1	1	1	1	0
on2	1	0	0	1	1	1
	1	0	1	1	1	1
on3	1	1	0	1	0	0
	1	1	1	1	0	0

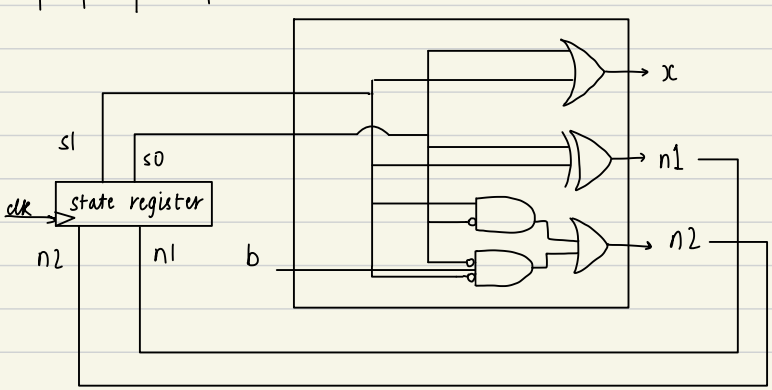
Combinational Logic
Input: s1, s0, b
Output: n1, n0, x



clearly, n1 is XOR
and n2 is $bs0's1' + s1s0'$

inputs	outputs
0 0	0
0 1	1
1 0	1
1 1	1

clearly an or operation
↑ logic for x



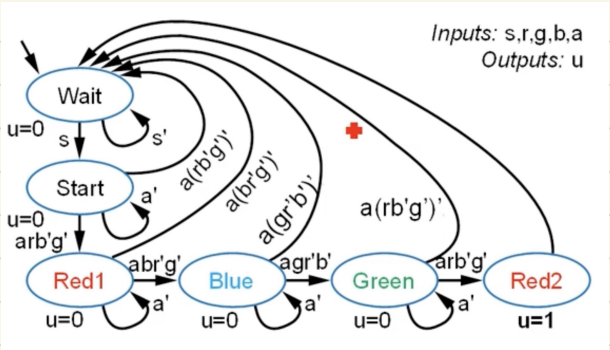
```
module pulse3 (input b,
               input clk, rst,
               output x);

 wire n1, n0;
 reg s1, s0;

assign n1 = s1^s0;
assign n0 = (~s1 & ~s0 & b) | (s1 & ~s0);
assign x = s1 | s0;

always @ (posedge clk)
begin
    if (rst) begin
          s1 <= 1'b0;
          s0 <= 1'b0;
    end
    else begin
       s1 <= n1
       s0 <= n0;
   end
end
endmodule
```

more complex circuits :



opening a locked door by adding in a certain code

① encode states :

wait = 000 red 1 = 010 green = 100
start = 001 blue = 011 red 2 = 101

and then use 'parameter' to declare named constants

eg. → parameter st1 = 2'b00, st2 = 2'b01 , st3 = 2'b10 ;

```
module pulse3 ( input s, r, g, b, a,
                input clk, rst,
                output u );

parameter waite = 3'b000, start=3'b001, red1 = 3'b010,
          blue = 3'b011, green = 3'b100, red2 = 3'b101;

reg [2:0] nst, st;

assign u = (st == red2);

always @ (posedge clk) begin
    if (rst)  st <= waite;
    else      st <= nst;

end

always @ * begin
  nst = st;                    <- default; protects self-transition
    case (st)
        waite : if (s) nst = start;
        start : if (a)
                    if (r&~b&~g) nst = red1;
                    else nst = waite;
        red1 : if (a)
                    if (b&~r&~g) nst = blue;
                    else nst = waite;
        blue : if (a)
                    if (g&~r&~b) nst = green;
                    else nst = waite;
```

wait is a reserved keyword

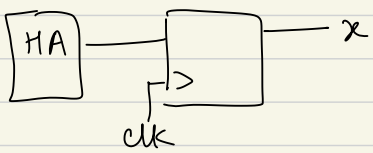
returns 0 or 1

```
        green : if(a)
                    if (r&~g&~b) nst = red2;
                    else nst = waite;

        red2  : nst = waite

        default : nst = waite;
    endcase
end
```

synchronous (pos..)	combinational (*)
· get a register	· no register (assignment is a wire)
· ⇒ output is of a register	· ⇒ output is of the adder
· x = a + b	



always