Finite State Machines

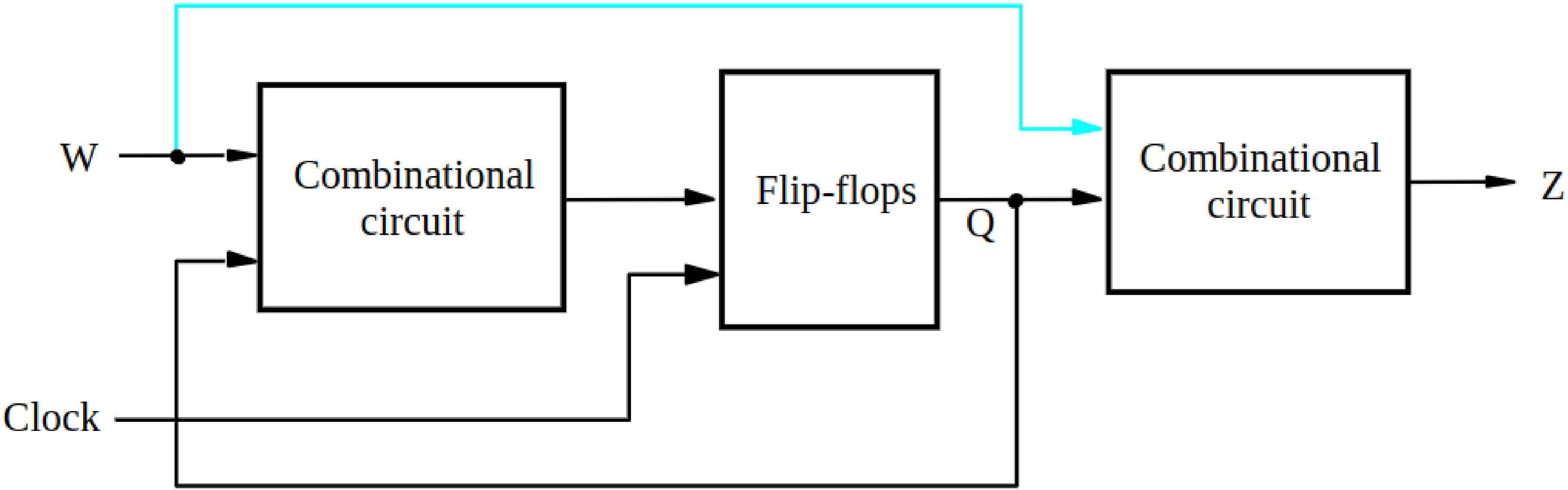
NYU

Acknowledgment for the slides: Hammond Pearce, Ramesh Karri

Sequential Circuits

- In a combinational circuit, values of outputs are determined only by inputs.
- In a sequential circuit, values of outputs depend on *history* and inputs.
- In other words, a sequential circuit has **states** which in conjunction with inputs determine behaviour.

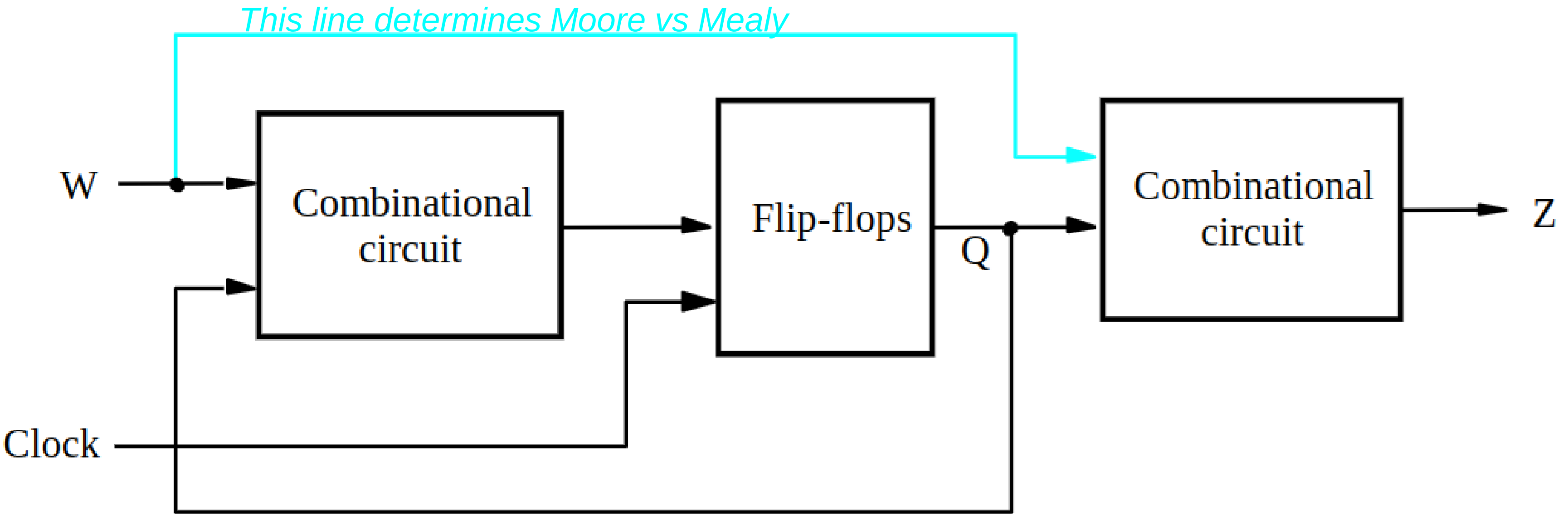
The general form of a sequential circuit



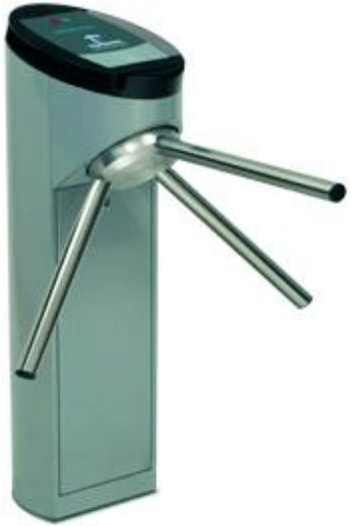
Finite State Machines (FSMs) are a useful abstraction for sequential circuits with centralized “states” of operation

Moore vs Mealy

- If the outputs depend only on the present state, it is a **Moore** FSM
- If the outputs depend on the state and the inputs, it is a **Mealy** FSM



An Example Finite State Machine (FSM)

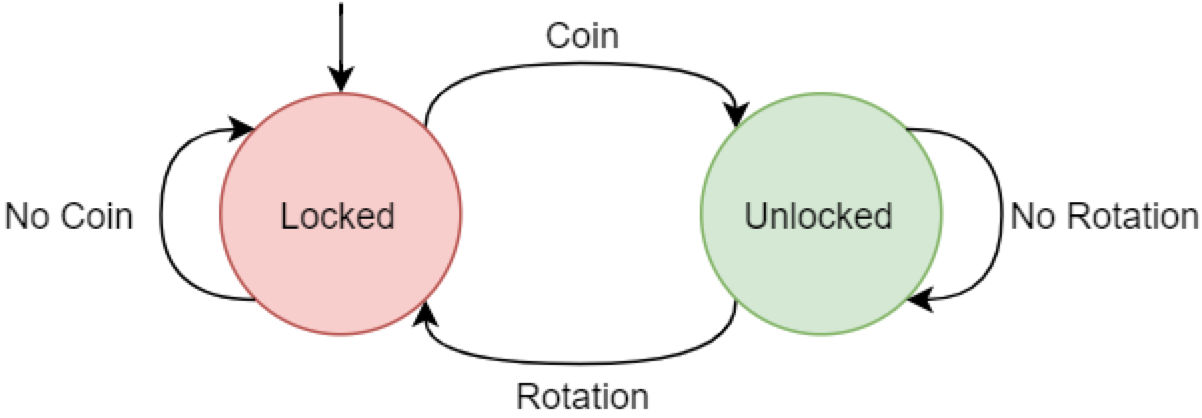


What are the possible states?

What are the possible transitions?

https://en.wikipedia.org/wiki/Finite-state_machine#/media/File:Torniqueterevolution.jpg

An Example Finite State Machine (FSM)

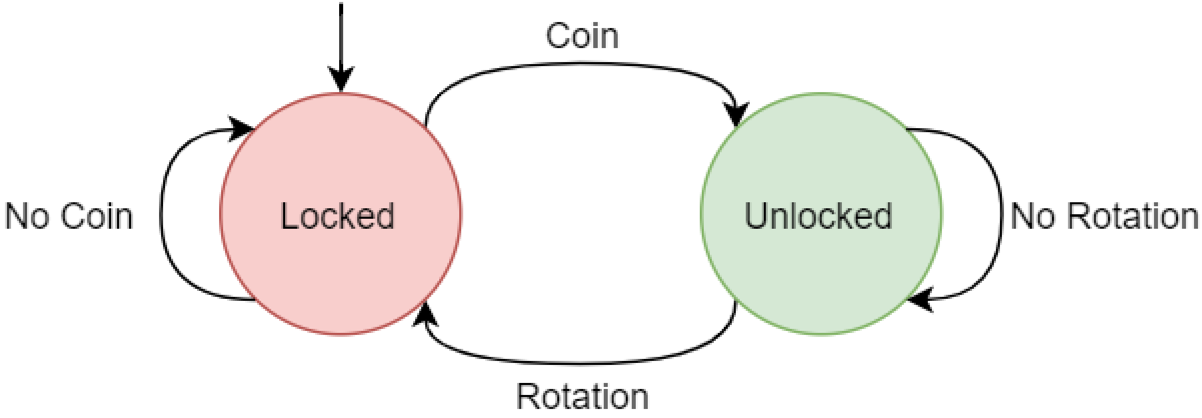


Is this Mealy or Moore?
What are the outputs?
What do the outputs depend on?

https://en.wikipedia.org/wiki/Finite-state_machine#/media/File:Torniqueterevolution.jpg

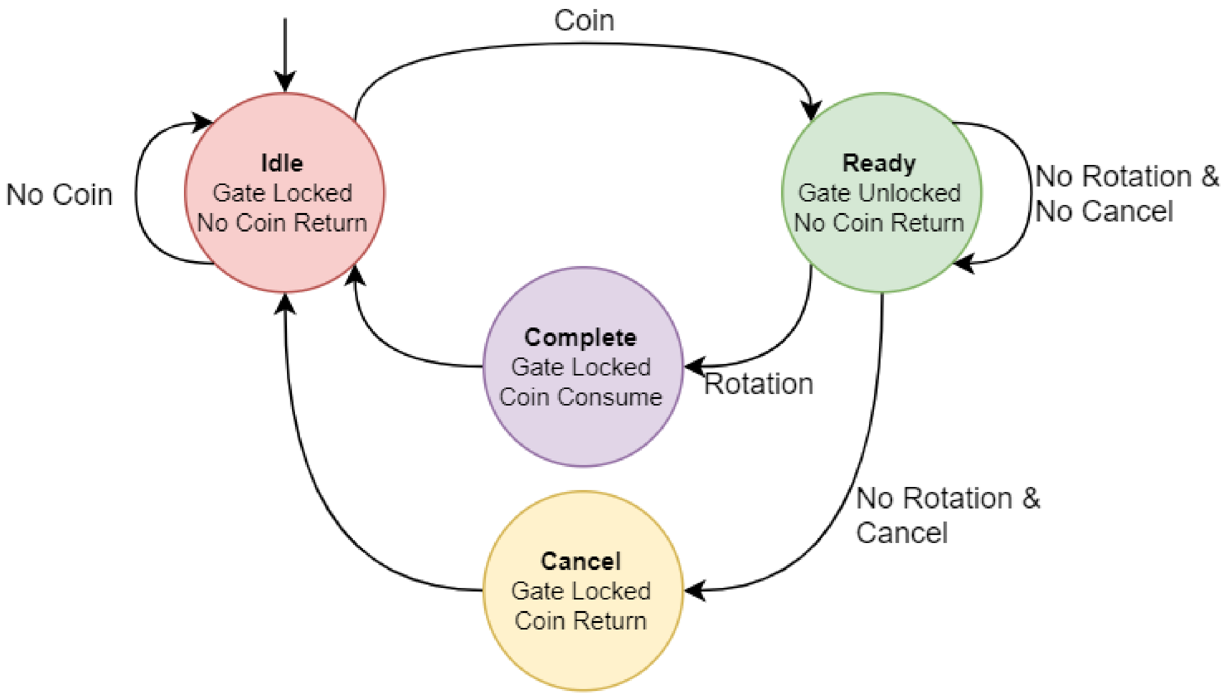
Hands-on Exercise

Expand the FSM below in include a “CANCEL” option: If after inserting a coin the user presses “CANCEL”, the coin should be returned



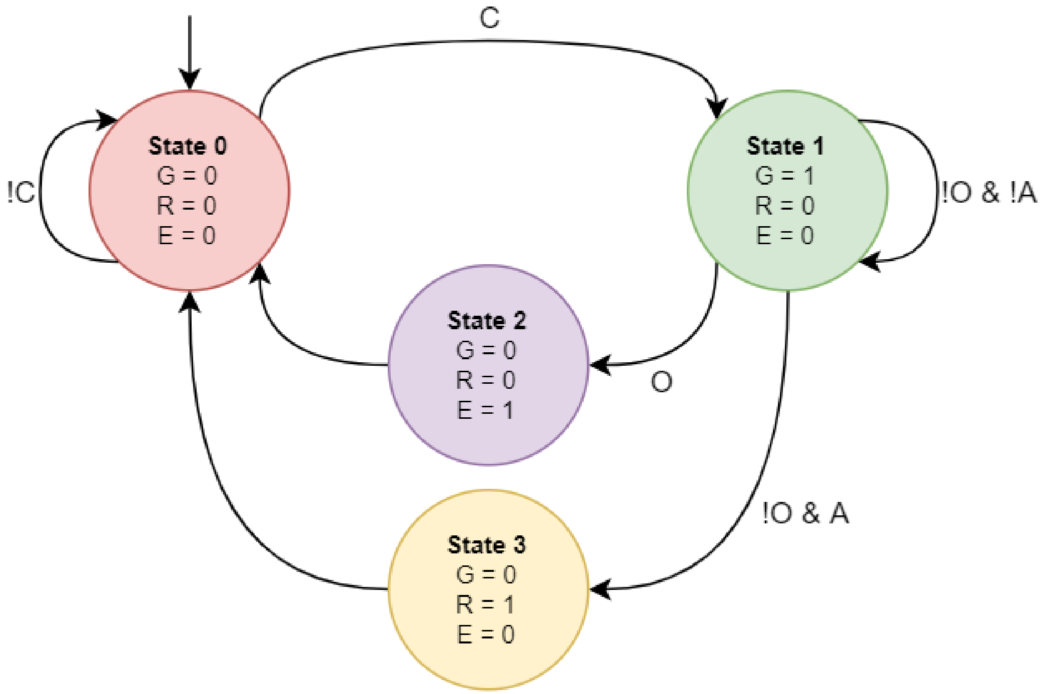
https://en.wikipedia.org/wiki/Finite-state_machine#/media/File:Torniqueterevolution.jpg

An Example FSM: adding more functionality



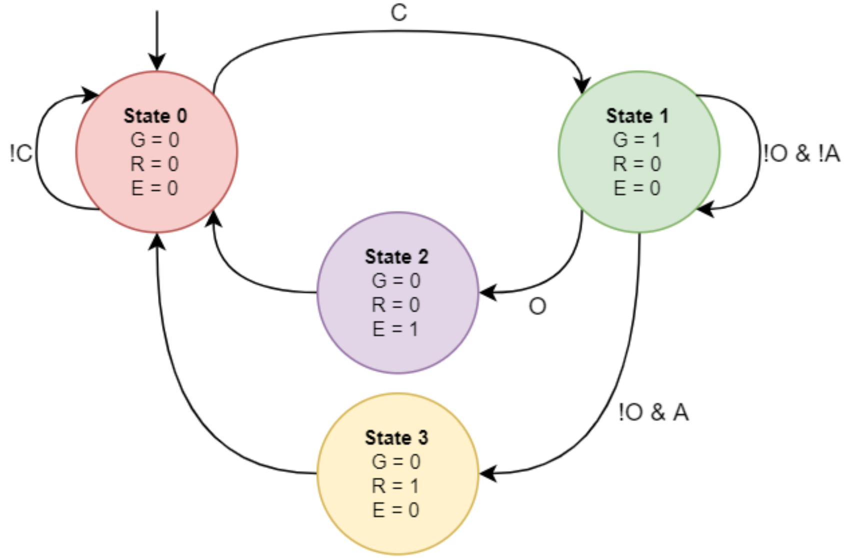
(Equivalently)

Inputs: C for Coin, O for rOtation, A for cAncel



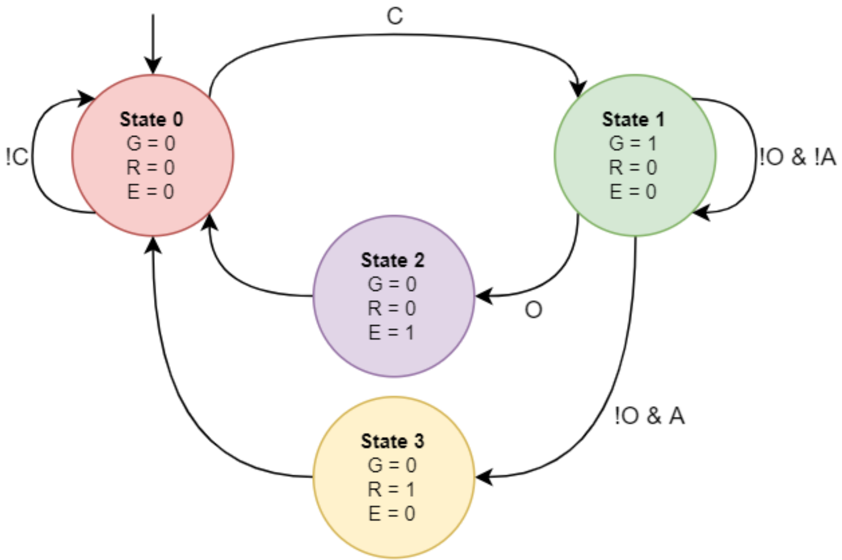
Outputs: G for Gate lock, R for coin Return, E for coin consumE

An execution *trace*



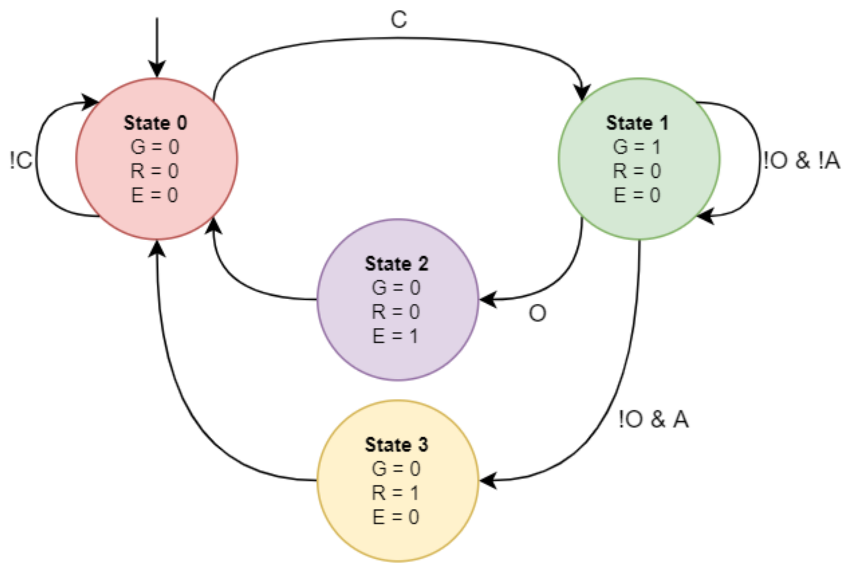
Time	0	1	2	3	4	5	6	7	8
Input	C		O		C	A	A	A	
Output		G	G	E	0	G	R	0	0

An execution *trace*



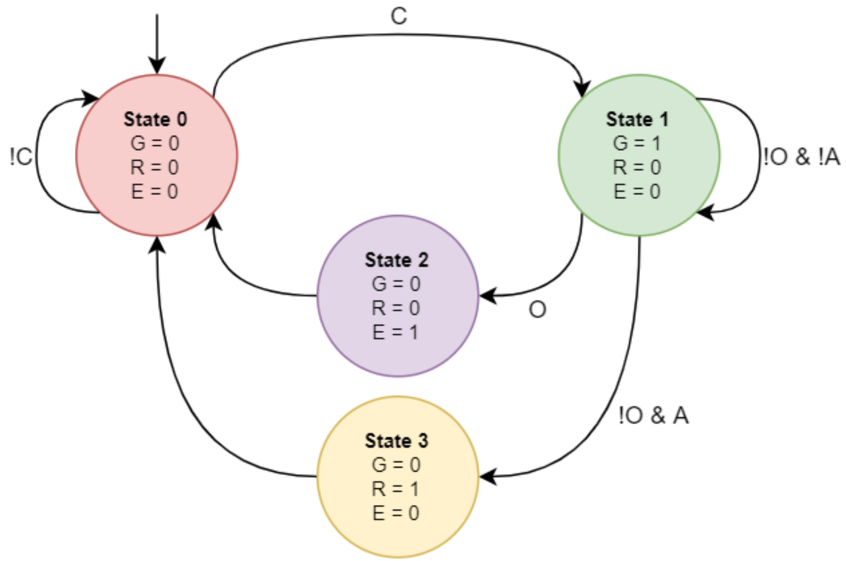
Time	0	1	2	3	4	5	6	7	8
Input	C		O		C	A	A	A	
Output		G	G	E		G	R		

Abnormal transitions?



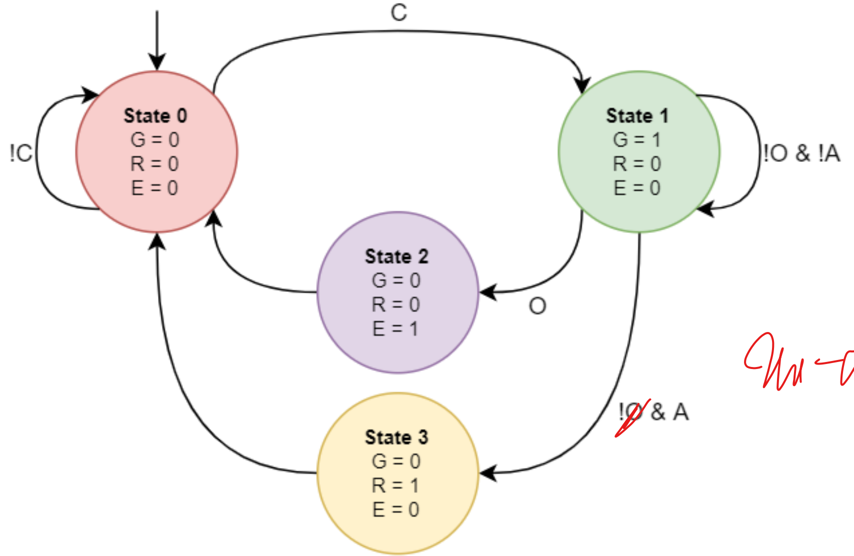
Time	0	1	2	3	4	5	6	7	8
Input									
Output									

Abnormal transitions?



Time	0	1	2	3	4	5	6	7	8
Input	C, A								
Output		??? G							

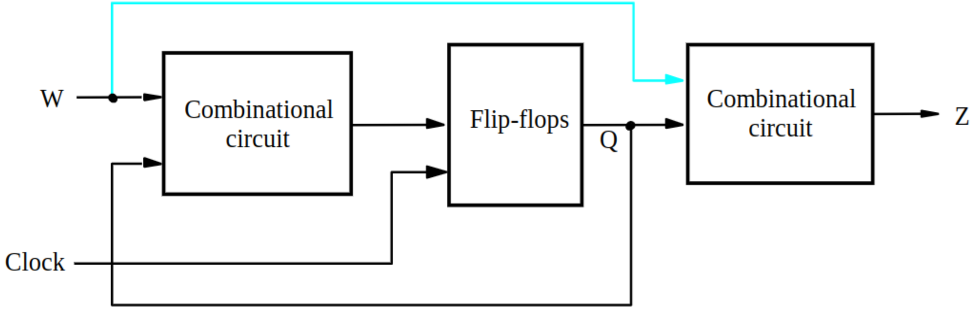
Abnormal transitions?



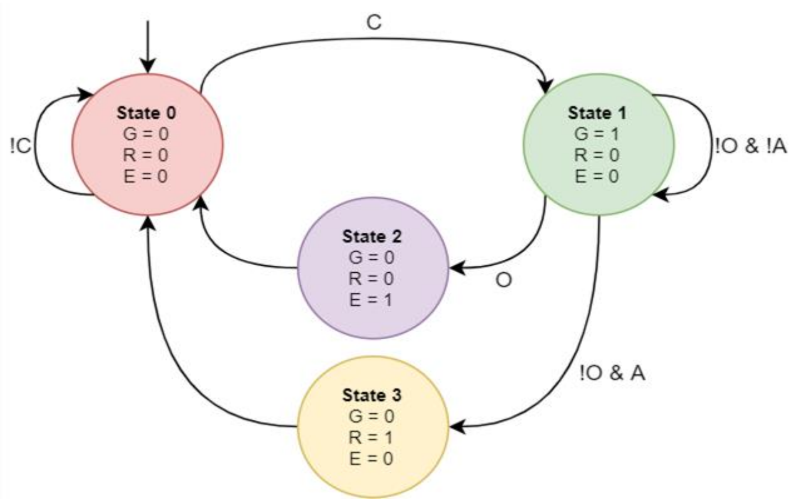
Un-thud is a mess

Time	N	1	2	3	4	5	6	7	8
Input		O, A							
Output	G	State=2	???						

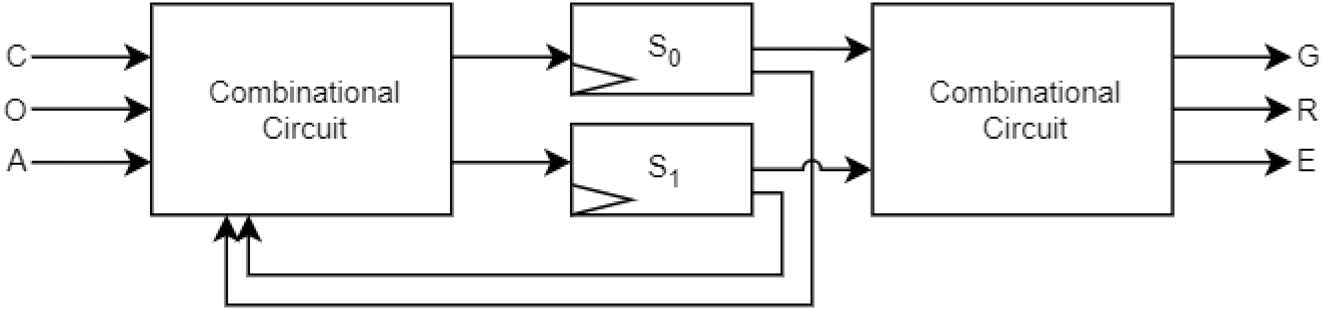
Hands-on: Circuit layout of turnstile FSM?



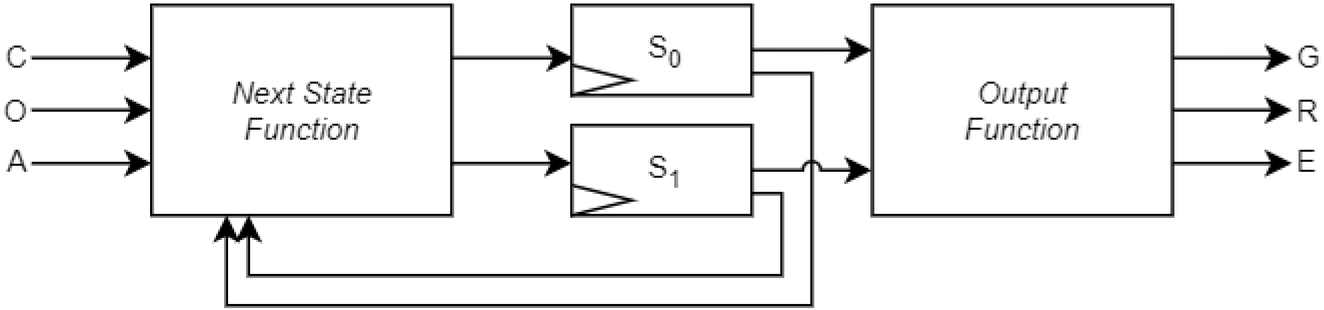
+



=



Circuit Layout



Can we express these functions as hardware?

- We sure can!
- E.g. Next state

$$s_0 = f(s_1, s_0, C, O, A)$$
$$s_1 = f(s_1, s_0, C, O, A)$$

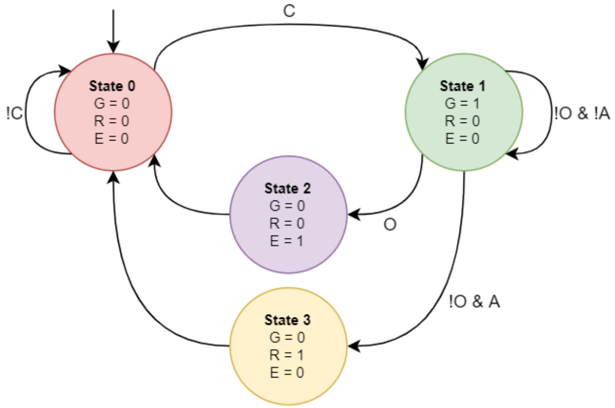
- E.g. Outputs

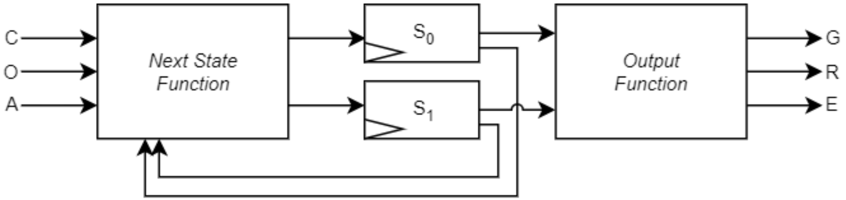
$$G = f(s_1, s_0)$$
$$R = f(s_1, s_0)$$
$$E = f(s_1, s_0)$$

(All these functions
f will be different)

- Each of these could be implemented using a sum of minterms, etc.

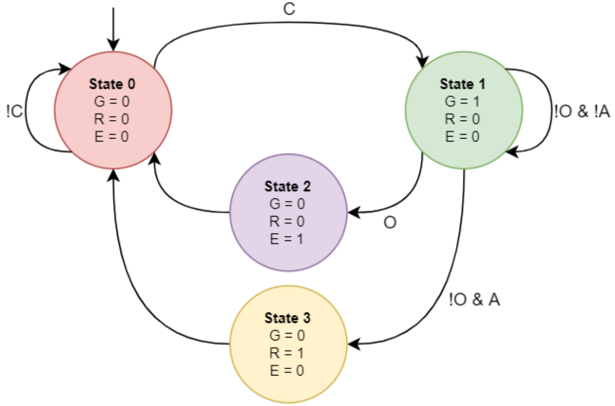
Expressing a State Machine in Verilog

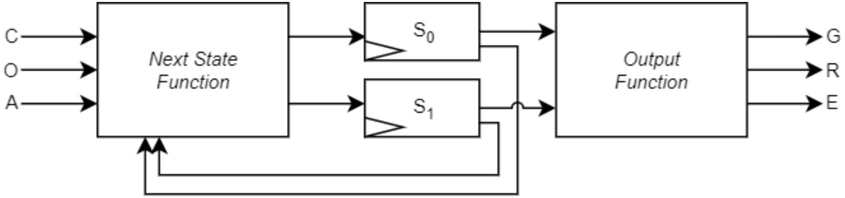




(Live coding by Hammond)

Expressing a State Machine in Verilog





```
module turnstileFsm(
        input wire clk,
        input wire C, O, A,
        output reg G, R, E
);

//state register
reg [1:0] s;

//next state function
always @(posedge clk) begin
        case(s)
                2'd0: begin
                        if(C) s <= 2'd1;
                end
                2'd1: begin
                        if(O) s <= 2'd2;
                        else if(A) s <= 2'd3;
                end
                2'd2: begin
                        s <= 2'd0;
                end
                2'd3: begin
                        s <= 2'd0;
                end
        endcase
end
```

begin end act as
( ) brackets

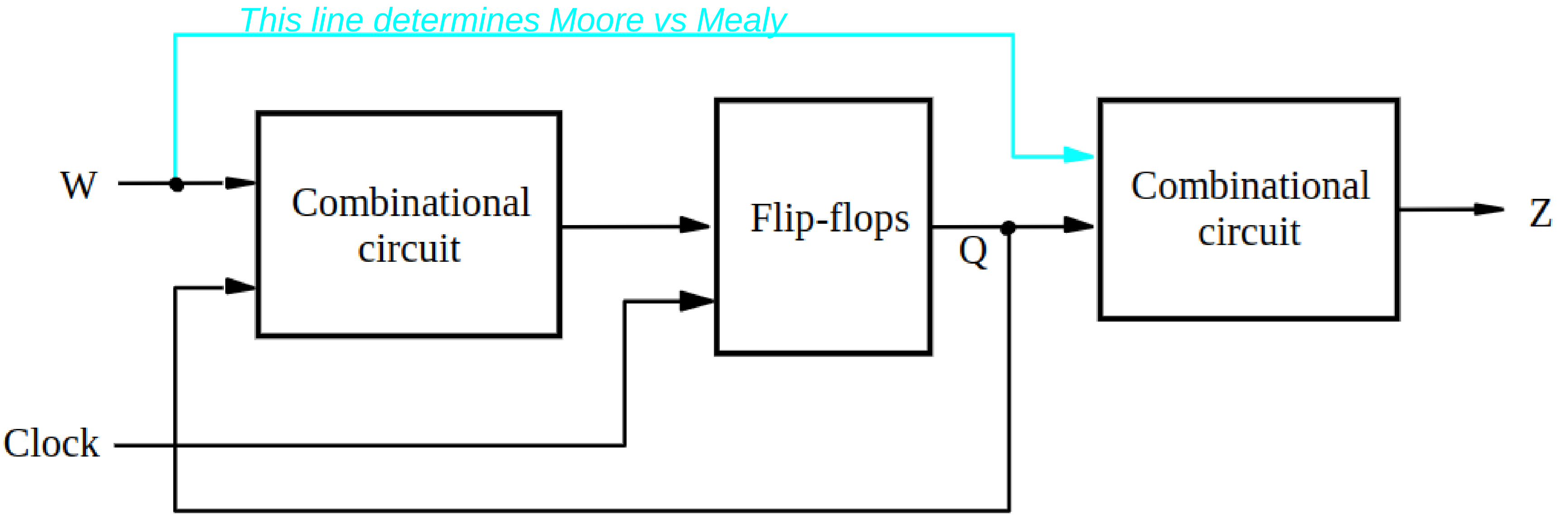
```
//output function
always @(s) begin
        G <= 1'b0;
        R <= 1'b0;
        E <= 1'b0;
        case(s)
                2'd0: begin

                end
                2'd1: begin
                        G <= 1;
                end
                2'd2: begin
                        E <= 1;
                end
                2'd3: begin
                        R <= 1;
                end
        endcase

end
endmodule
```

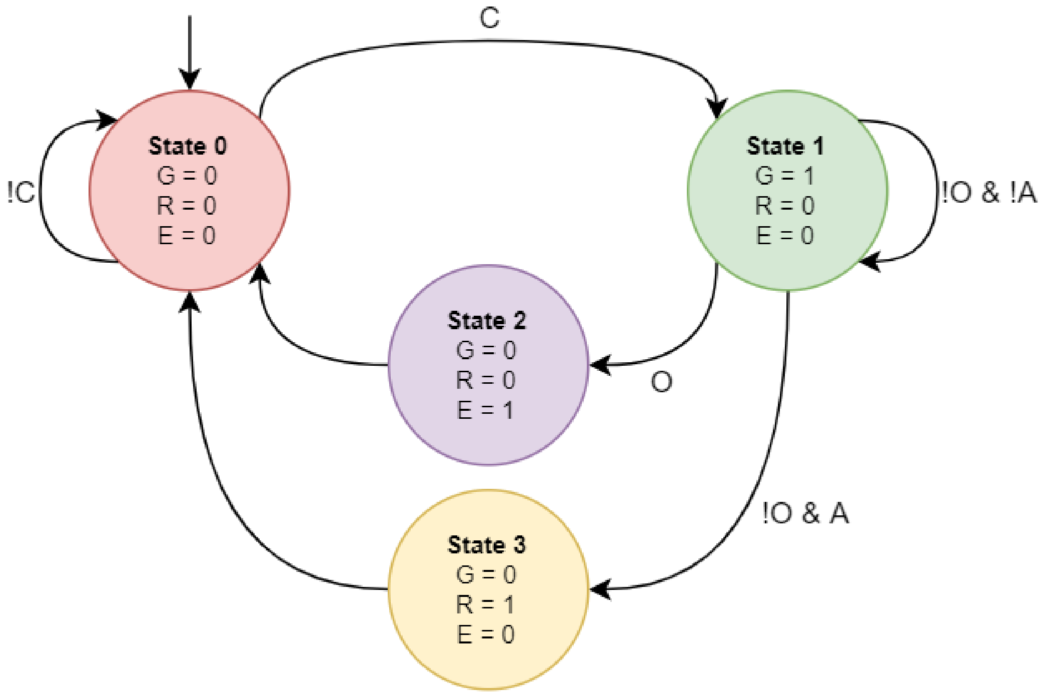
Mealy type FSMs

- Mealy machine outputs depend on *state* and *input*
- Let’s examine a similar turnstile



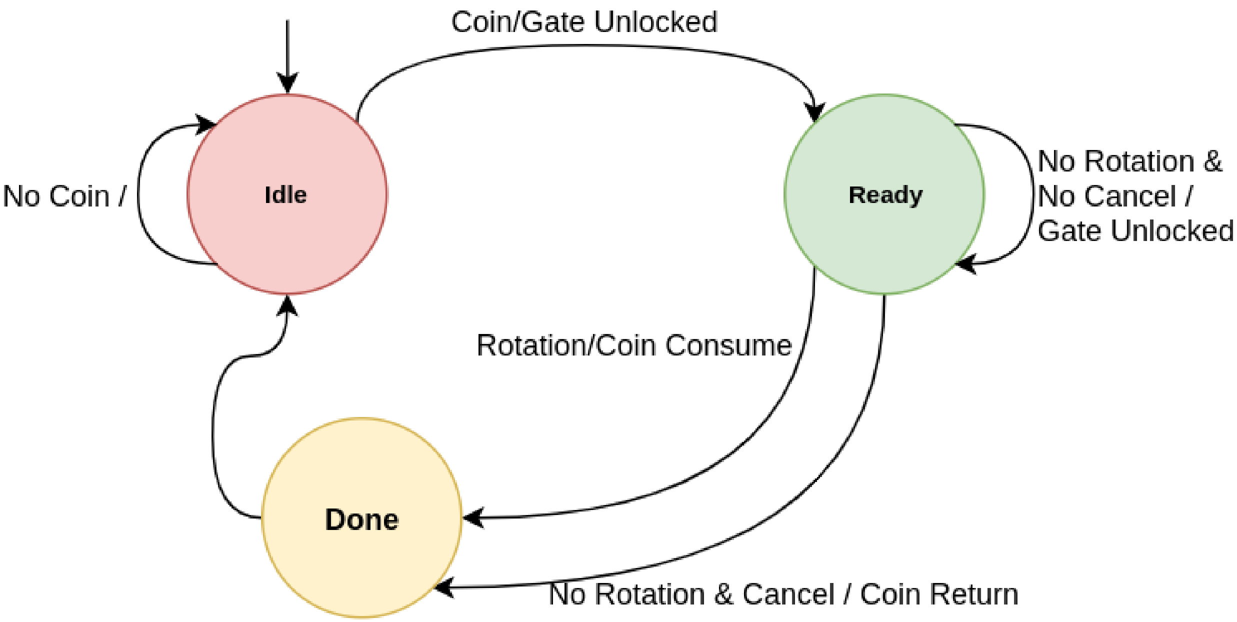
Hands-on: Convert to Mealy FSM

Inputs: C for Coin, O for rOtation, A for cAncel



Outputs: G for Gate lock, R for coin Return, E for coin consumE

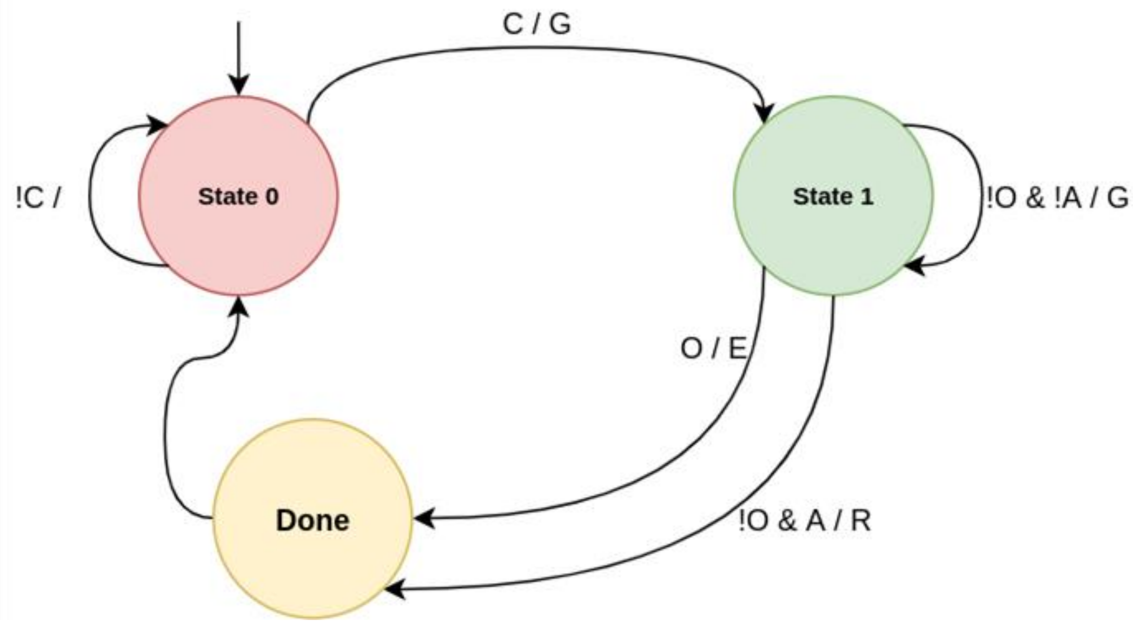
Turnstile Mealy Machine



<u>Outputs on Transitions after a "/"</u>

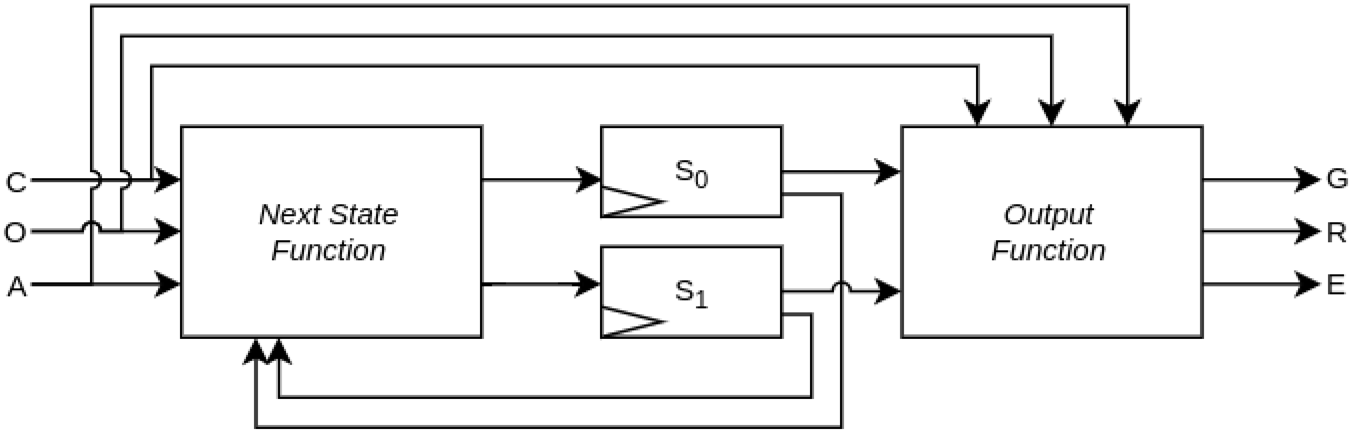
Here, omitted outputs are "0"

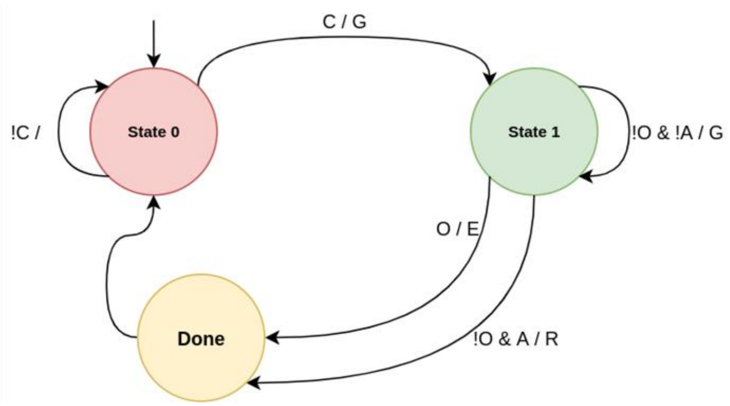
(Equivalently)



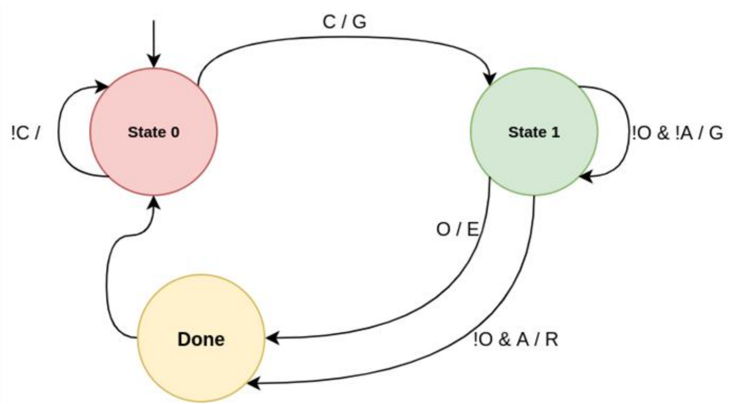
Here, omitted outputs are “0”

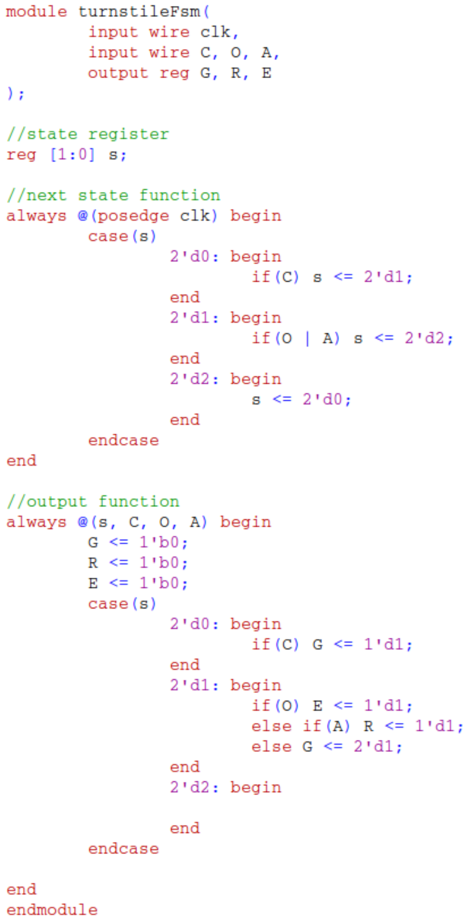
Mealy Turnstile Hardware





(Live coding by Hammond)





```
module turnstileFsm(
        input wire clk,
        input wire C, O, A,
        output reg G, R, E
);

//state register
reg [1:0] s;

//next state function
always @(posedge clk) begin
        case(s)
                2'd0: begin
                        if(C) s <= 2'd1;
                end
                2'd1: begin
                        if(O | A) s <= 2'd2;
                end
                2'd2: begin
                        s <= 2'd0;
                end
        endcase
end

//output function
always @(s, C, O, A) begin
        G <= 1'b0;
        R <= 1'b0;
        E <= 1'b0;
        case(s)
                2'd0: begin
                        if(C) G <= 1'd1;
                end
                2'd1: begin
                        if(O) E <= 1'd1;
                        else if(A) R <= 1'd1;
                        else G <= 2'd1;
                end
                2'd2: begin

                end
        endcase

end
endmodule
```

When to use each type?

- Moore machines have more states than Mealy
  - So Moore machines take more hardware
- Moore machines are “safer” than Mealy
  - Output only changes on clock edge, shorter critical paths, can mean higher clock rates
- Moore machines react “slower” than Mealy
  - Mealy output is combinatorial, FSM can react in the same clock tick as input is created
- Importantly, we can **safely convert between them**
  - For every Moore machine there is an equivalent Mealy and vice versa
  - The Mealy machine will likely have fewer states
  - The Mealy and Moore turnstile machines presented so far are *almost* equivalent (but not quite)!

State encoding

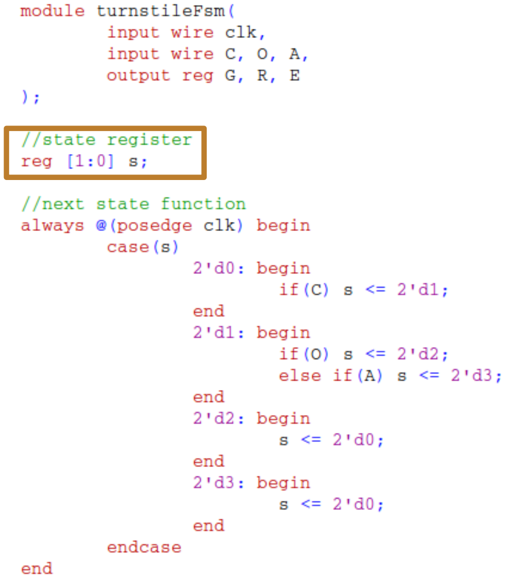
- State encoding is the transformation of the state number to a binary representation
  - Example: State 0: 00, State 1: 01, etc.
  - Various different ways to encode:

    Sequential

    Gray

    Johnson

    One-hot



```
module turnstileFsm(
        input wire clk,
        input wire C, O, A,
        output reg G, R, E
);

//state register
reg [1:0] s;

//next state function
always @(posedge clk) begin
        case(s)
                2'd0: begin
                        if(C) s <= 2'd1;
                end
                2'd1: begin
                        if(O) s <= 2'd2;
                        else if(A) s <= 2'd3;
                end
                2'd2: begin
                        s <= 2'd0;
                end
                2'd3: begin
                        s <= 2'd0;
                end
        endcase
end
```

Encoding formats

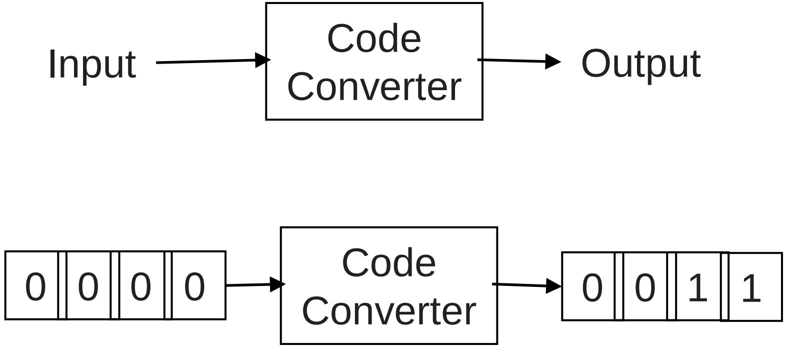
Decimal	Sequential	Gray	Johnson	One-hot
0	000	000	0000	00000001
1	001	001	0001	00000010
2	010	011	0011	00000100
3	011	010	0111	00001000
4	100	110	1111	00010000
5	101	111	1110	00100000
6	110	101	1100	01000000
7	111	100	1000	10000000

Encoding formats

- **Binary**: Good for arithmetic operations. But may have more transitions, leading to more power consumption. Also prone to error during the state transitions.
- **Gray**: Good as they reduce the transitions, and hence consume less dynamic power. Also, can be handy in detecting state transition errors.
- **Johnson**: Also there is one-bit change and can be useful in detecting errors during transitions. More bits are required, increases linearly with the number of states. There are unused states, so we require either explicit asynchronous reset or recovery from illegal states (even more hardware!)
- **One-hot**: yet another low power coding style, requires more no of bits. Useful for describing bus protocols.

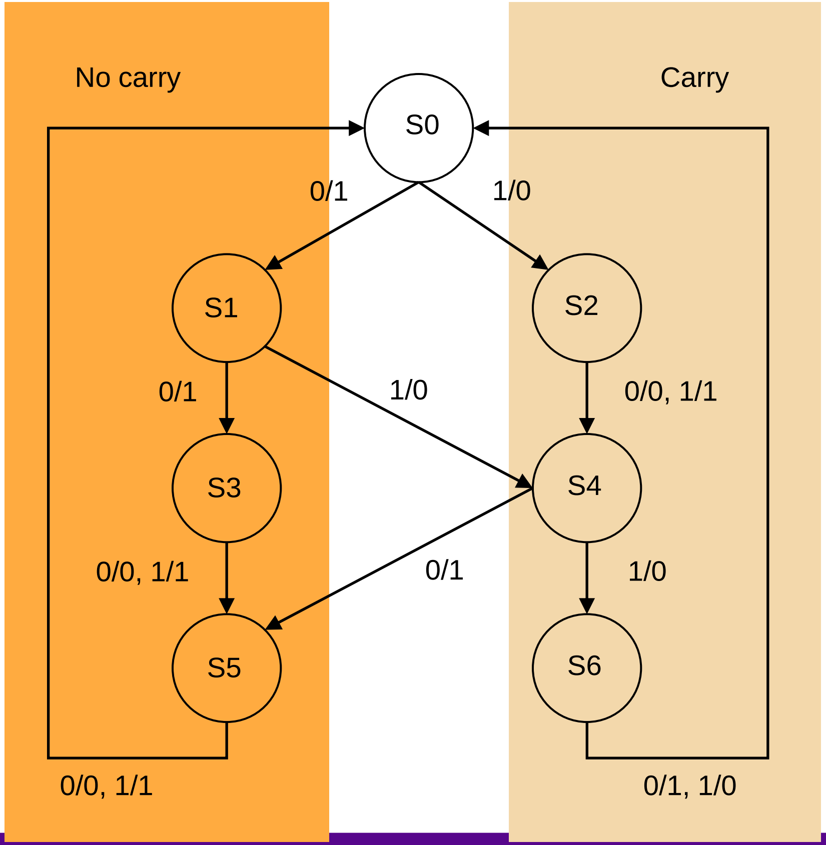
BCD to Excess-3 (BCD2X3) Code Converter

- Excess-3 code (X3): Add 3 to Binary Coded Decimal (BCD)
- Input: BCD, Least significant bit (LSB) first
- Output: Excess-3 code, LSB first



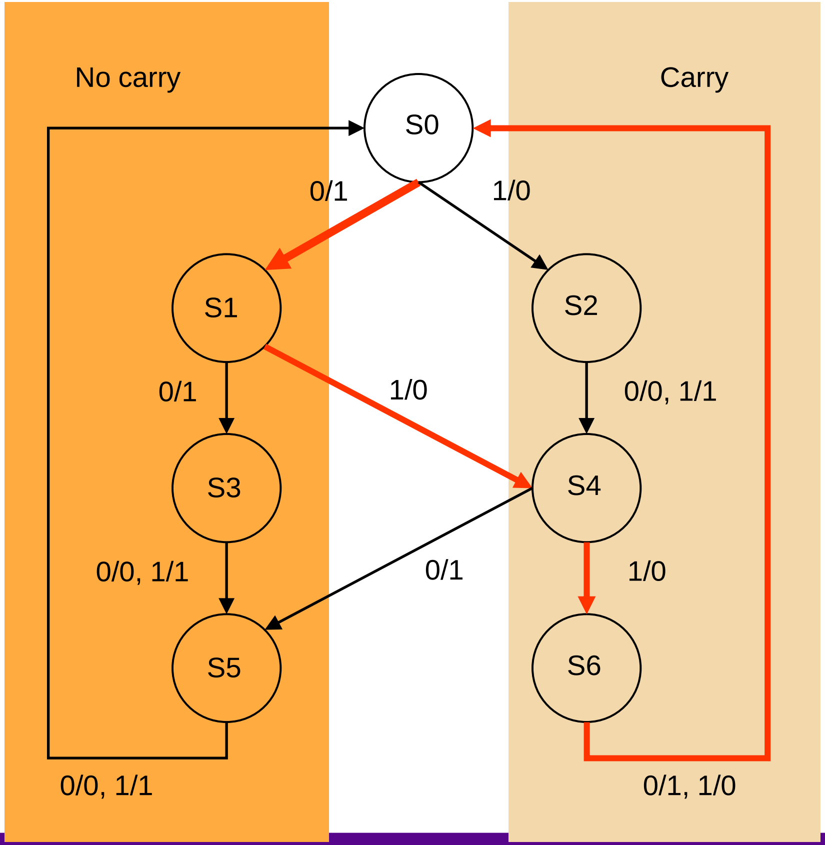
BCD	Excess-3
0000	0011
0001	0100
0010	0101
0011	0110
…	…
1111	0010

Mealy FSM



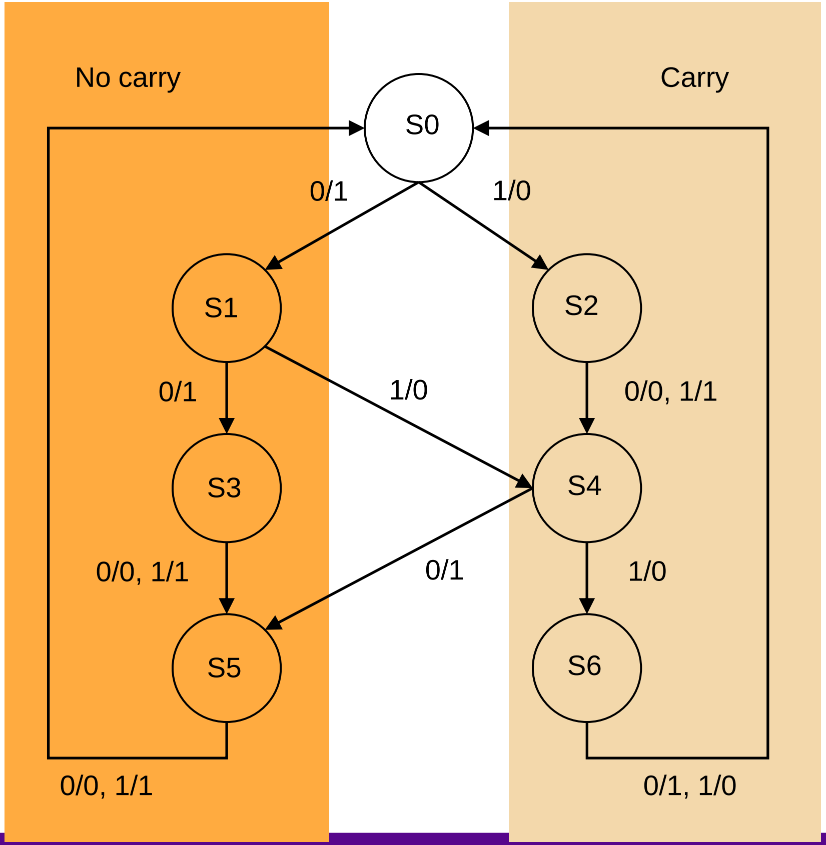
- S0: Reset state
- S1, S3, S5: No carry state
- S2, S4, S6: Carry state
- X/Y: Input/Output

A Trace on BCD-X3 Code Converter FSM



- Input: 0110
- Output: 1001
- States traversed:S0, S1, S4, S6, S0

Mealy FSM in VHDL



- S0: Reset state
- S1, S3, S5: No carry state
- S2, S4, S6: Carry state
- X/Y: Input/Output

(Live coding)

VHDL Model of BCD-X3 FSM (State Logic)

```
entity MealyFSM is port (
        x, reset, clock: in std_logic;
        z: out std_logic );
end MealyFSM;
architecture RTL of MealyFSM is
        type s_type is ( s0, s1, s2, s3, s4, s5, s6 );
        signal state, nextstate: s_type;
Begin
        stateFSM: process ( clock, reset )
        begin
          if ( reset = ‘1’ ) then state <= s0;
          elsif (clock’event and clock = ‘1’ ) then state <= nextstate;
          end if;
        end process stateFSM;
```

Defines storage element

Asynchronous reset

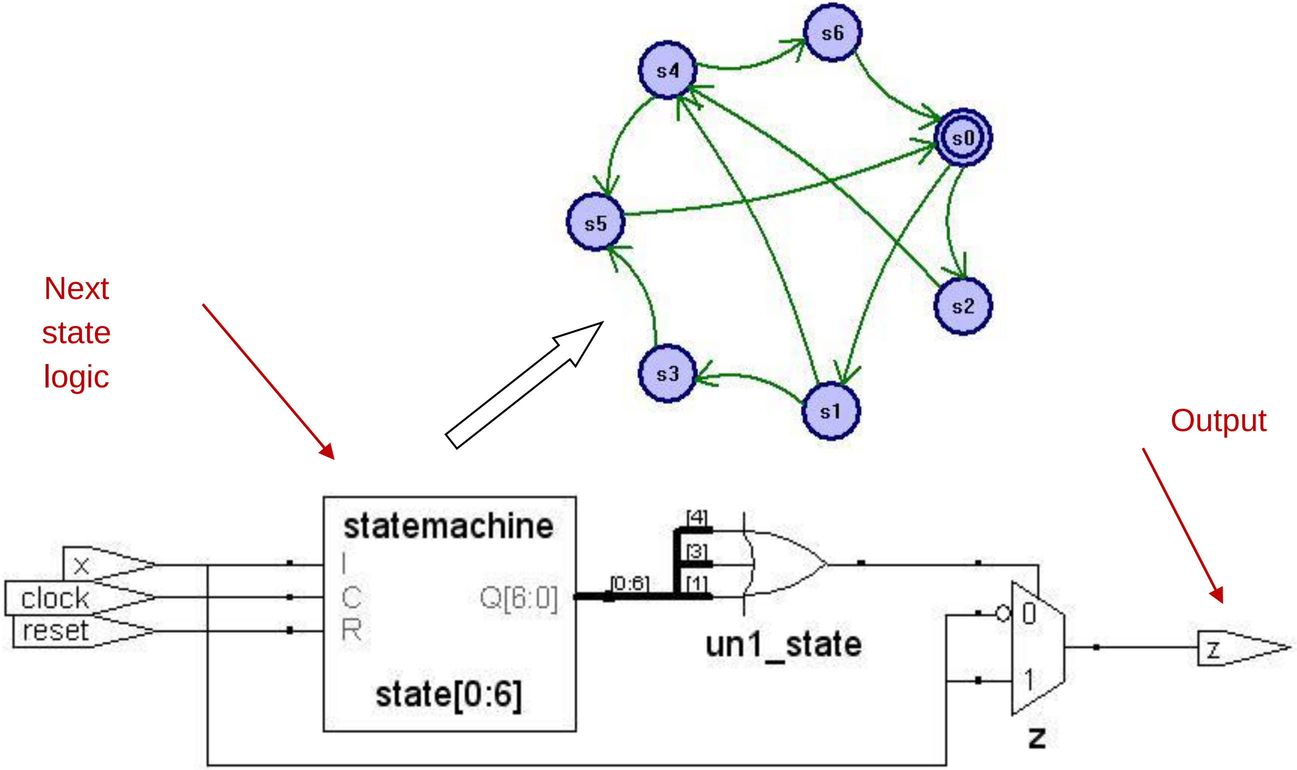
Rising edge triggered

VHDL Model of BCD-X3 FSM (Output Logic)

```
outputFSM: process ( state, x )
begin
  z <= ‘0’ ;
  nextstate <= s0;
  case state is
    when s0 => if x = ‘0’ then z <= ‘1’ ; nextstate <= s1; else z <= ‘0’ ; nextstate <= s2; end if;
    when s1 => if x = ‘0’ then z <= ‘1’ ; nextstate <= s3; else z <= ‘0’ ; nextstate <= s4; end if;
    when s2 => if x = ‘0’ then z <= ‘0’ ; nextstate <= s4; else z <= ‘1’ ; nextstate <= s4; end if;
    when s3 => if x = ‘0’ then z <= ‘0’ ; nextstate <= s5; else z <= ‘1’ ; nextstate <= s5; end if;
    when s4 => if x = ‘0’ then z <= ‘1’ ; nextstate <= s5; else z <= ‘0’ ; nextstate <= s6; end if;
    when s5 => if x = ‘0’ then z <= ‘0’ ; nextstate <= s0; else z <= ‘1’ ; nextstate <= s0; end if;
    when s6 => if x = ‘0’ then z <= ‘1’ ; nextstate <= s0; end if;
    when others => null;
  end case;
end process outputFSM;
end RTL;
```

Output is a combinational function of input and state

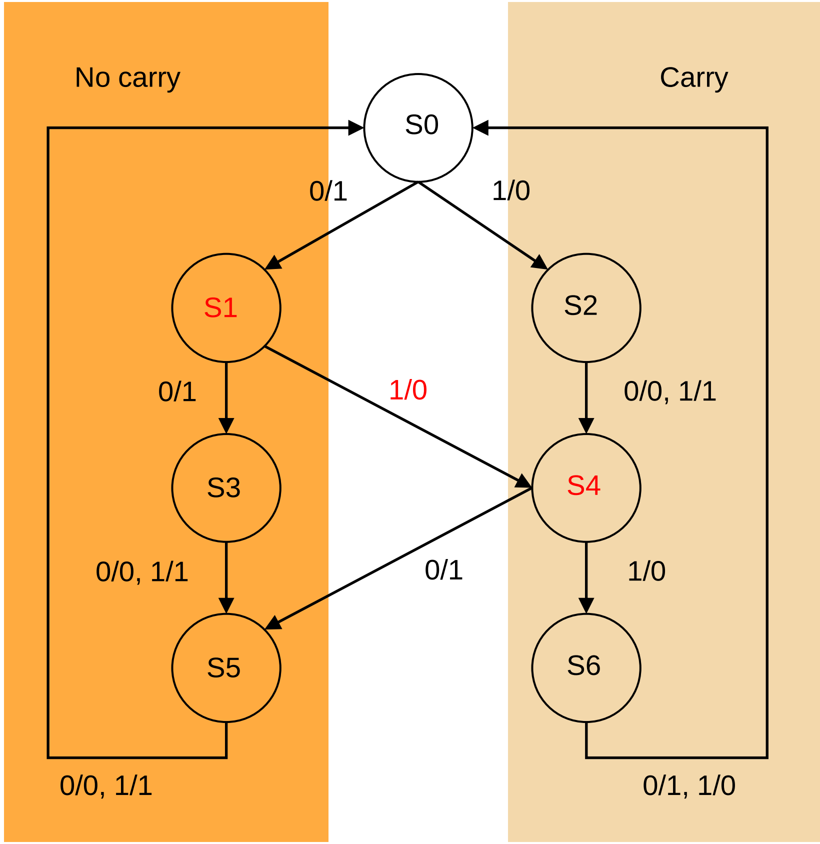
Synthesized BCD2X3 Converter



Next state logic

Output

FSM Output Glitching

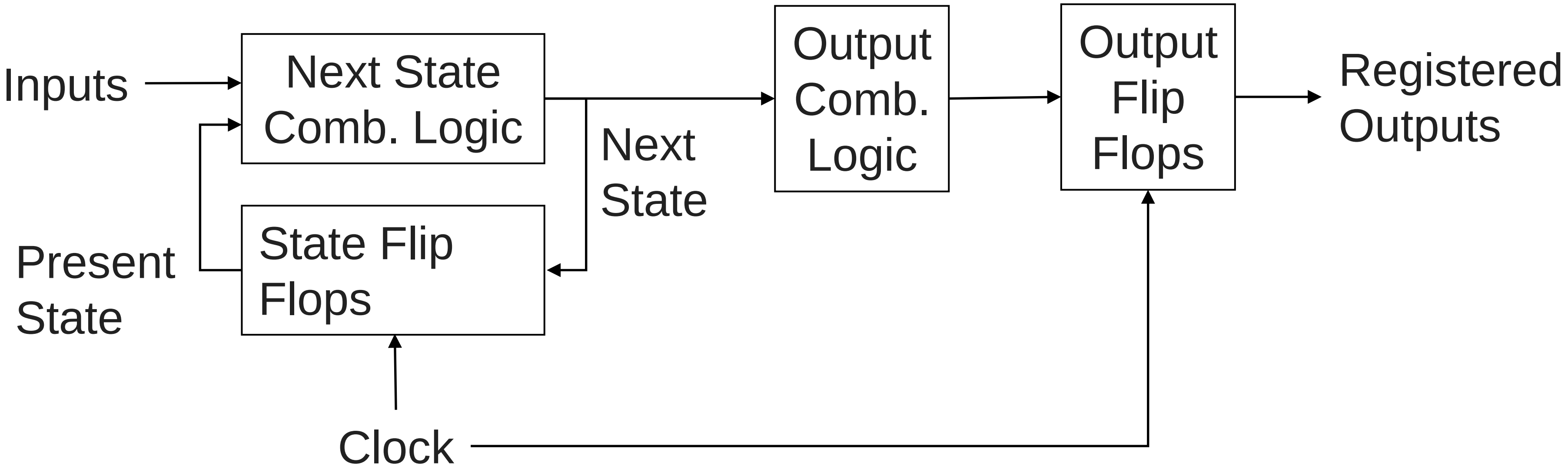


- In real hardware, state bits may not transition at precisely the same time
- Combinational logic for outputs may contain race hazards
- Output may glitch!
- Example: S1 -> S4

S1	0001
(S5)	0101
S4	0100

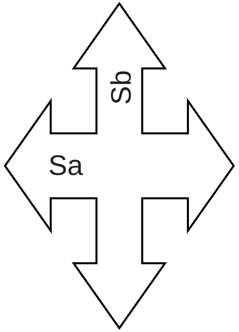
- The S5 will cause Y to glitch to 1 before becoming 0

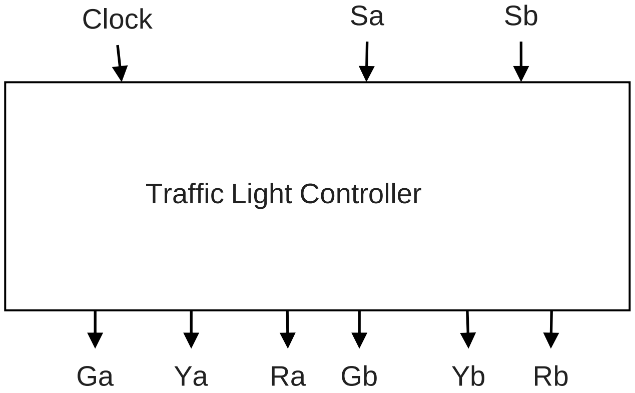
Registered output


Inputs
Next State Comb. Logic
Present State
State Flip Flops
Next State
Output Comb. Logic
Output Flip Flops
Registered Outputs
Clock

- Delays output by 1 clock cycle, which may be unacceptable in some applications

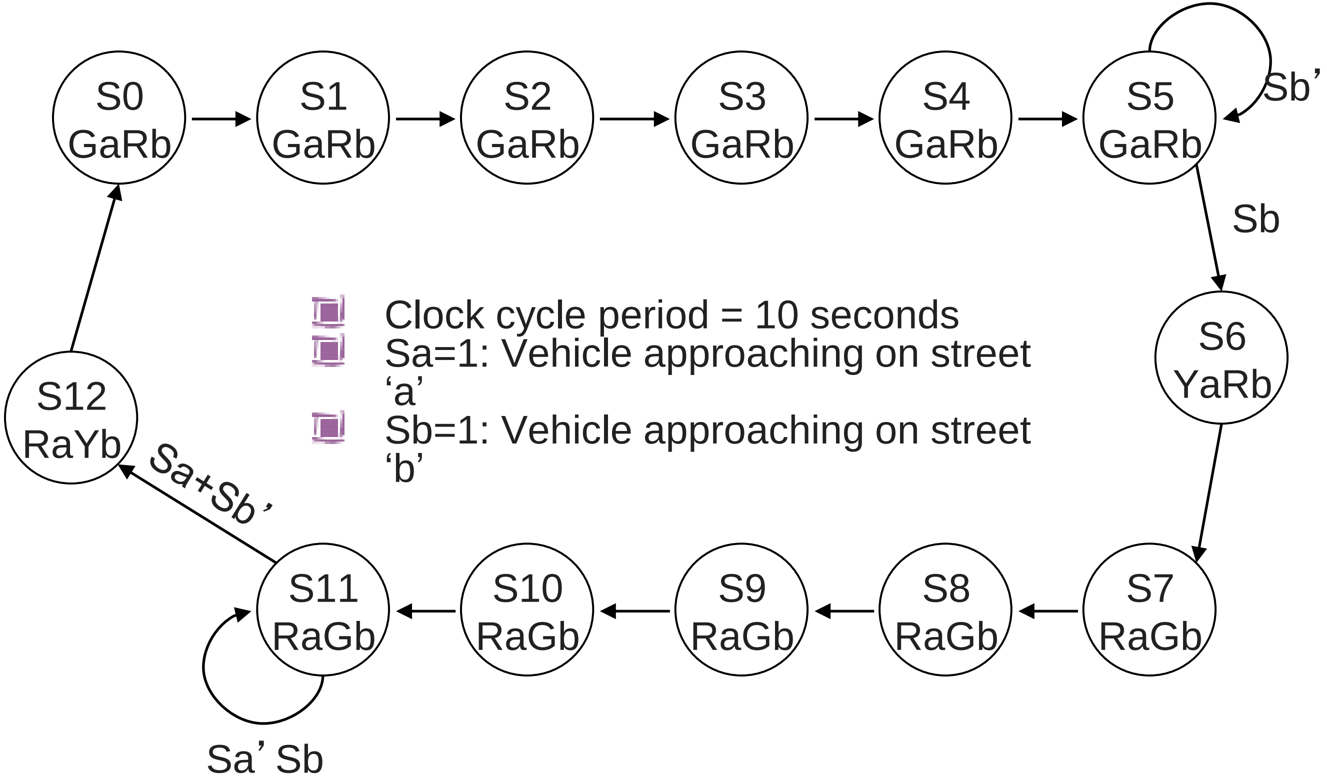
Traffic Light Controller





- Street a is the main street and street b is the side street
- Sa=1: Vehicle approaching on street ‘a’
- Sb=1: Vehicle approaching on street ‘b’
- Light on ‘a’ stays green until vehicle on ‘b’
- Then light on ‘b’ changes green and changes back after 50 seconds
- If another car on ‘b’ and no car on ‘a’ then light on ‘b’ remains green for 10 more seconds
- When ‘a’ is green remains green for at least 60 seconds

Traffic Light Controller Mealy FSM



- Clock cycle period = 10 seconds
- Sa=1: Vehicle approaching on street 'a'
- Sb=1: Vehicle approaching on street 'b'

Traffic Light Controller VHDL Model

```
entity traffic_light is port ( clk, Sa , Sb: in std_logic;
                                    Ra, Rb, Ga, Gb, Ya, Yb: out
    std_logic );
end traffic_light;

architecture behave of traffic_light is
    signal state, nextstate: integer range 0 to 12;
begin
    process( clk )
    begin
         if ( clk’event and clk = ‘1’ ) then
                    state <= nextstate;
         end if;
    end process;
```

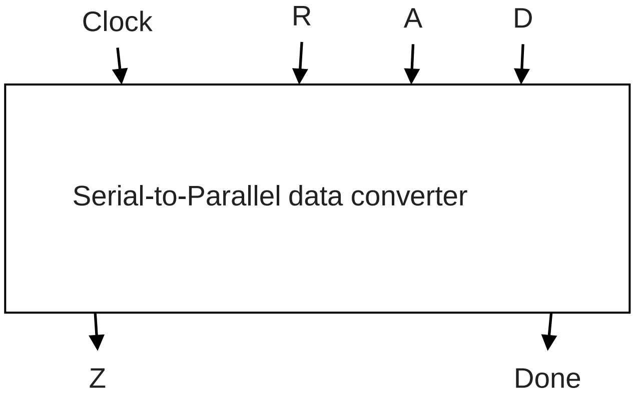
State transition on clock edge

Traffic Light Controller VHDL Model

Output and next state logic

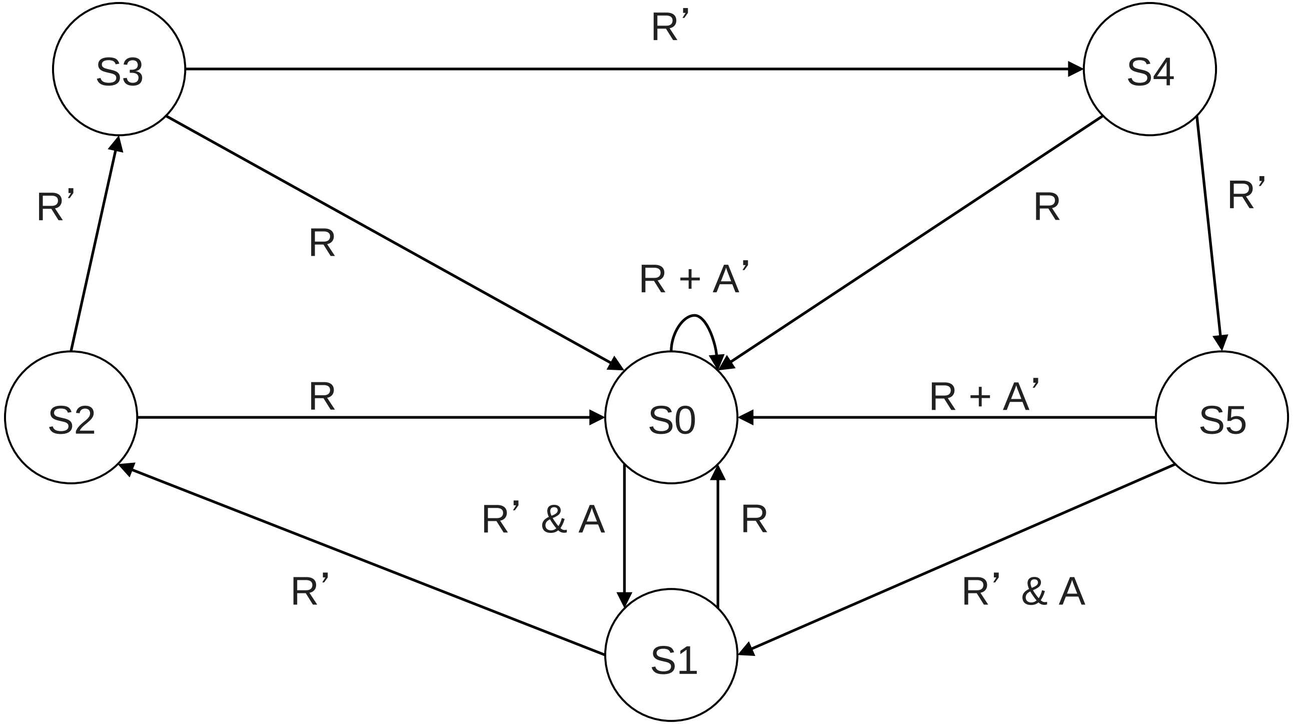
```
    process ( state, Sa, Sb )
    begin
        Ra <=‘0’; Rb<=‘0’; Ga<=‘0’;
        Gb<=‘0’; Ya<=‘0’; Yb<= ‘0’;
        case state is
            when 0 to 4 => Ga <= ‘1’; Rb <= ‘1’; nextstate <=
    state+1;
            when 5 => Ga <= ‘1’; Rb<=‘1’;
                    if Sb = ‘1’then nextstate <=6; end if;
            when 6 => Ya <= ‘1’; Rb <= ‘1’; nextstate <= 7;
            when 7 to 10 =>Ra <=‘1’; Gb <=‘1’; nextstate <= state
    +1;
            when 11=> Ra <= ‘1’; Gb <= ‘1’;
                    if (Sa =‘1’ or Sb =‘0’) then nextstate <= 12; end if;
            when 12 => Ra <= ‘1’; Yb <=‘1’; nextstate <= 0;
            when others => NULL;
        end case;
    end process;
end behave;
```

Serial-to-Parallel Data (S2PD)



- 4-bit serial-to-parallel converter
- Synchronous reset initializes the state machine from any state
- ‘A’ denotes valid data at input port ‘D’
- ‘Z’ generates parallel data

S2PD Converter Moore FSM



- R: Reset
- A: Valid data

S2PD Converter VHDL Model

```
entity spdc is port ( clk, R , A, D: in bit;
                      Z: out bit_vector ( 3 downto 0 ) ; Done: out bit );
end spdc;

architecture fsm_rtl of spdc is
    type state_type is ( S0, S1, S2, S3, S4, S5 );
    signal state: state_type;
    signal shift_reg: bit_vector ( 3 downto 0 );
begin
    process ( state )
    begin
          case state is
                      when s0 to s4 => Done <= ‘0’;
                      when s5 => Done <= ‘1’; Z <= shift_reg;
          end case;
    end process;
```

State transition logic

S2PD Converter VHDL Model (Cont.)

```
    process ( clk )
    begin
       if ( clk = ‘1’ ) then
           case state is
               when S0 => if R = ‘1’ or A = ‘0’ then state <= S0;
                       elsif R = ‘0’ and A = ‘1’ then state <= S1; end if;
               when S1 => shift_reg <= D & shift_reg ( 3 downto 1 );
                       if R = ‘0’ then state <= S2; elsif R = ‘1’ then state <= S0; end
    if;
               when S2 => shift_reg <= D & shift_reg ( 3 downto 1 );
                       if R = ‘0’ then state <= S3; elsif R = ‘1’ then state <= S0; end
    if;
               when S3 => shift_reg <= D & shift_reg ( 3 downto 1 );
                       if R = ‘0’ then state <= S4; elsif R = ‘1’ then state <= S0; end
    if;
               when S4 => shift_reg <= D & shift_reg ( 3 downto 1 );
                       if R = ‘0’ then state <= S5; elsif R = ‘1’ then state <= S0; end
    if;
               when S5 => if R = ‘0’ and A = ‘1’ then state <= S1;
                       elsif R = ‘1’ or A = ‘0’ then state <= S0; end if;
               when others => NULL;
           end case;
       end if;
    end process;
end fsm_rtl;
```

Output and next state logic

VHDL Modeling of FSMs

- Controller design should
  - Avoid asynchronous feedback, race conditions
  - Split large counters
  - Initialize design to a known state
- Minimize number of latches
  - Full specify all signals in a combinational process
  - A flow-through latch is implied when a signal or variable in a combinational process is not fully specified
- Prefer case over IF-THEN-ELSE
  - IF statement operates on a priority encoded basis