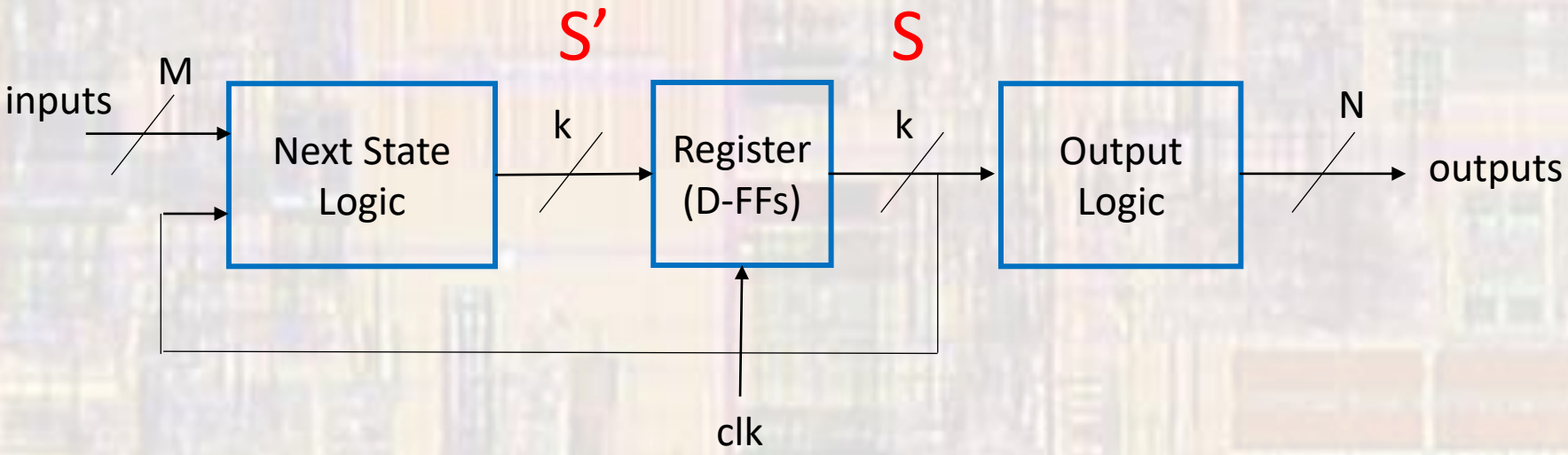
Simple Finite State Machines

Last updated 7/18/23

Simple FSMs

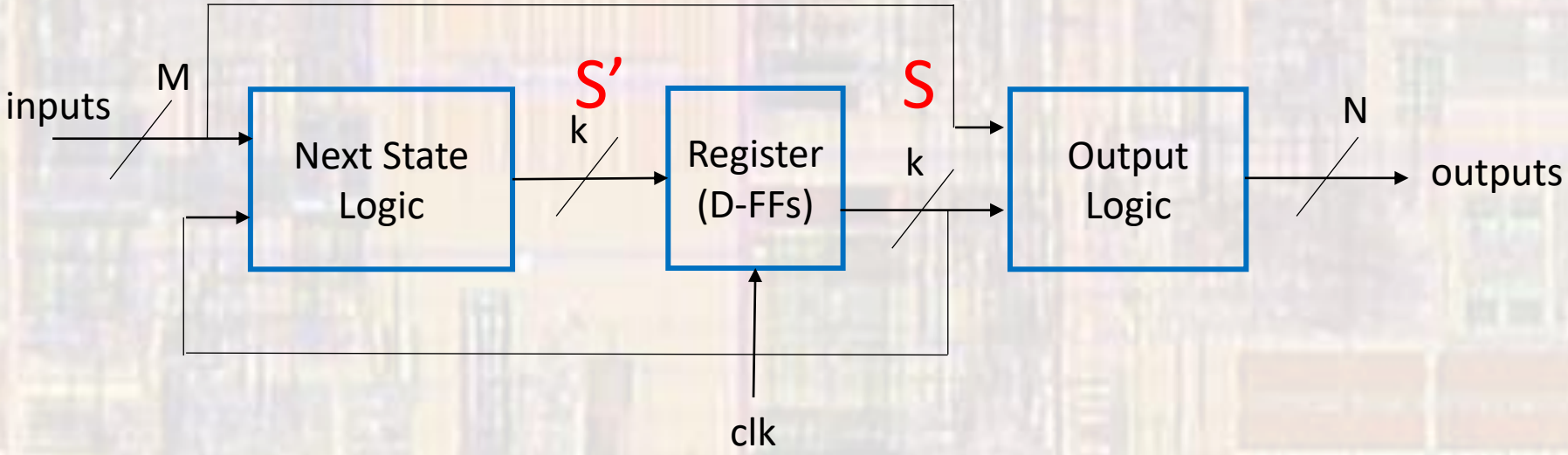
- Finite State Machine
  - Moore Machine
    - Outputs depend only on the current state(S)



© tj

Simple FSMs

- Finite State Machine
  - Mealy Machine
    - Outputs depend on the current state(S) and the inputs



© tj

Simple FSMs

- Finite State Machine
  - To create an FSM in VHDL you combine
    - Enumerated types
    - Simple register logic
    - Combinatorial logic (process)
    - Reset
  - Enumerated type
    - User defined type that only has the allowed states as values

Allowed state names

```
type STATETYPE is (rst,A,B,C,D,E,F,G,det);
signal state:       STATETYPE;
signal state_next:  STATETYPE;
```

State signals of the enumerated type
Can only have the values associated with the states

© tj

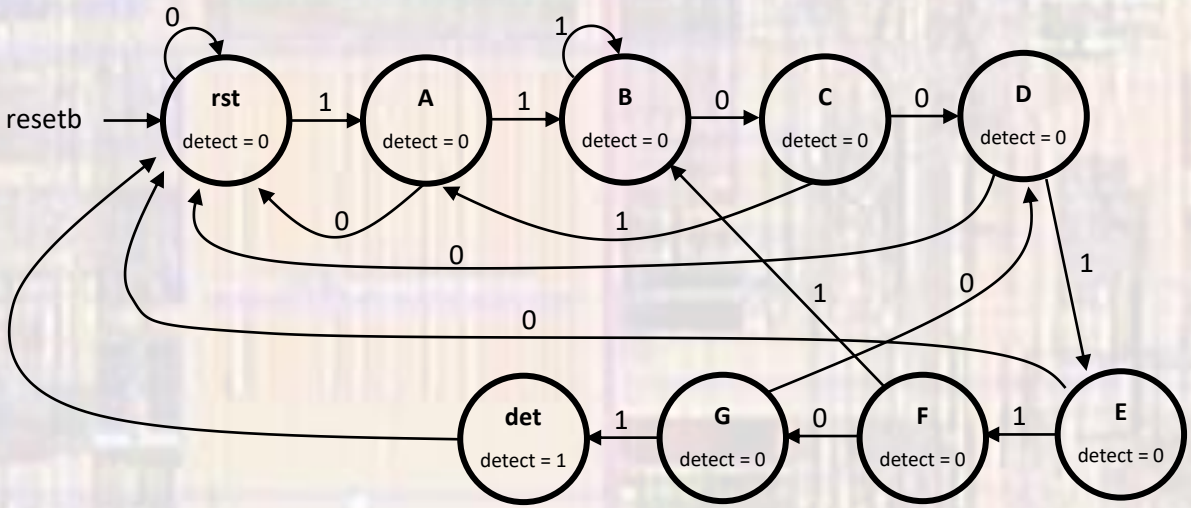
Simple FSMs

- Finite State Machine
  - Rules
    - Ensure that all states have entry and exit conditions
    - Ensure that all states are covered in the code
  - An abstraction of the FSM is used for RTL simulation
    - You cannot see the next state/register logic

© tj

Simple FSMs

- Sequence detector - CD
  - Detecting 11001101



© tj

Simple FSMs

- Sequence detector - CD

```
----------------------------------------
--
-- sequence_detector_CD_fsm.vhdl
--
-- created 4/12/2018
-- tj
--
-- rev 0
----------------------------------------
--
-- sequence detector state machine
--
----------------------------------------
--
-- Inputs: rstb, clk, Din
-- Outputs: Detect
--
----------------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity sequence_detector_CD_fsm is
   port (
      i_clk:      in std_logic;
      i_rstb:     in std_logic;
      i_D:        in std_logic;

      o_Detect:   out std_logic
    );
end entity;

architecture behavioral of sequence_detector_CD_fsm is

   --
   -- internal signals
   --
   type STATETYPE is (rst,A,B,C,D,E,F,G,det);
   signal state:       STATETYPE;
   signal state_next:  STATETYPE;

begin
```

```
   --
   -- next state logic
   --
   process(all)
   begin
      case state is
         when rst =>
            if i_D = '1' then
               state_next <= A;
            else
               state_next <= rst;
            end if;
         when A =>
            if i_D = '1' then
               state_next <= B;
            else
               state_next <= rst;
            end if;
         when B =>
            if i_D = '0' then
               state_next <= C;
            else
               state_next <= B;
            end if;
         when C =>
            if i_D = '0' then
               state_next <= D;
            else
               state_next <= A;
            end if;
         when D =>
            if i_D = '1' then
               state_next <= E;
            else
               state_next <= rst;
            end if;
         when E =>
            if i_D = '1' then
               state_next <= F;
            else
               state_next <= rst;
            end if;
         when F =>
            if i_D = '0' then
               state_next <= G;
            else
               state_next <= B;
            end if;
         when G =>
            if i_D = '1' then
               state_next <= det;
            else
               state_next <= D;
            end if;
         when others =>
            state_next <= rst;
      end case;
   end process;
```


© tj

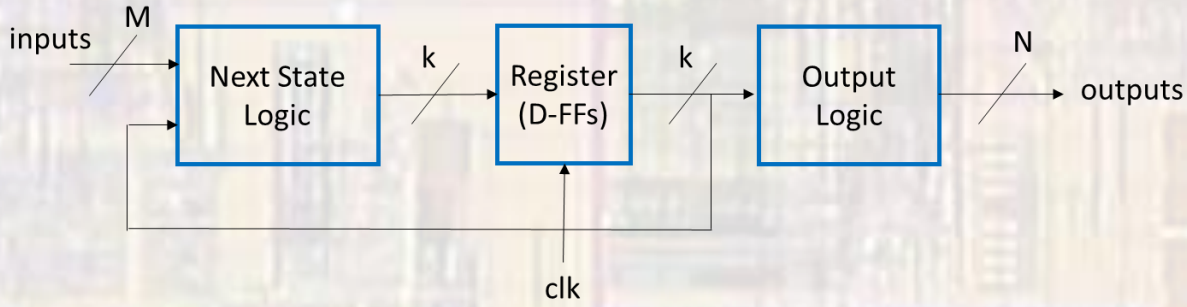
Simple FSMs

- Sequence detector - CD

```
   --
   -- Register logic
   --
   process(i_clk, i_rstb)
   begin
      -- reset
      if (i_rstb = '0') then
         state <= rst;
      -- rising clk edge
      elsif (rising_edge(i_clk)) then
         state <= state_next;
      end if;
   end process;
```

```
   --
   -- Output logic
   --
   process(state)
   begin
      case state is
         when rst => o_Detect <= '0';
         when A   => o_Detect <= '0';
         when B   => o_Detect <= '0';
         when C   => o_Detect <= '0';
         when D   => o_Detect <= '0';
         when E   => o_Detect <= '0';
         when F   => o_Detect <= '0';
         when G   => o_Detect <= '0';
         when det => o_Detect <= '1';
         when others => o_Detect <= '0';
      end case;
   end process;

end behavioral;
```

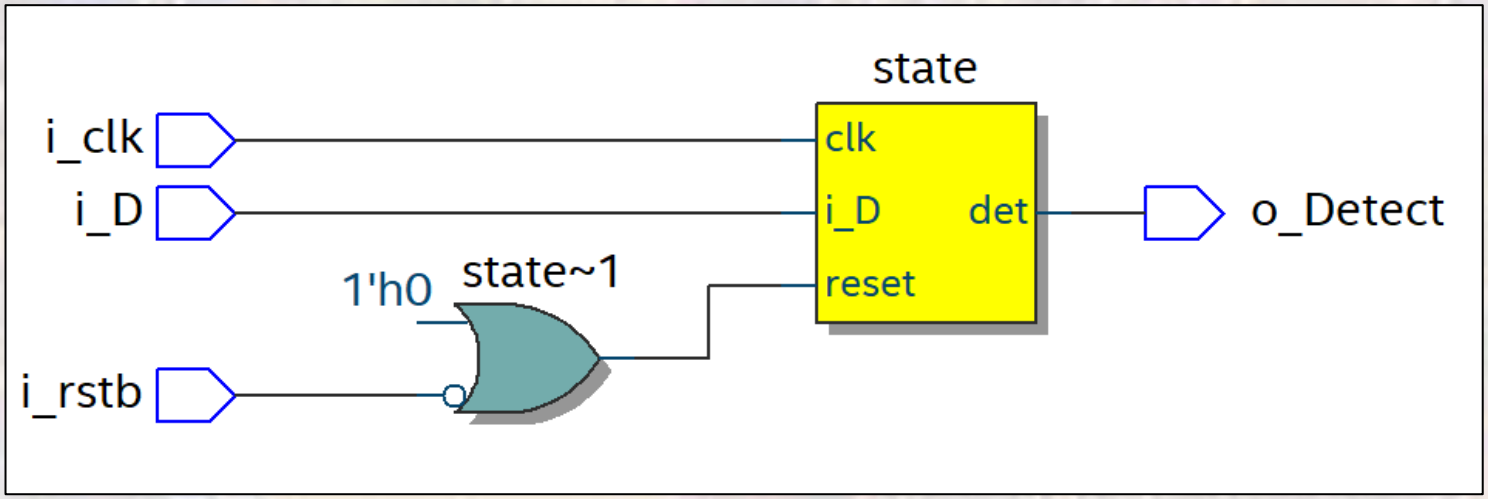
```
   --
   -- next state logic
   --
   process(all)
   begin
      case state is
         when rst =>
            if i_D = '1' then
               state_next <= A;
            else
               state_next <= rst;
            end if;
         when A =>
            if i_D = '1' then
               state_next <= B;
            else
               state_next <= rst;
            end if;
         when B =>
            if i_D = '0' then
               state_next <= C;
            else
               state_next <= B;
            end if;
         when C =>
            if i_D = '0' then
               state_next <= D;
            else
               state_next <= A;
            end if;
         when D =>
            if i_D = '1' then
               state_next <= E;
            else
               state_next <= rst;
            end if;
         when E =>
            if i_D = '1' then
               state_next <= F;
            else
               state_next <= rst;
            end if;
         when F =>
            if i_D = '0' then
               state_next <= G;
            else
               state_next <= B;
            end if;
         when G =>
            if i_D = '1' then
               state_next <= det;
            else
               state_next <= D;
            end if;
         when others =>
            state_next <= rst;
      end case;
   end process;
```



© tj

Simple FSMs

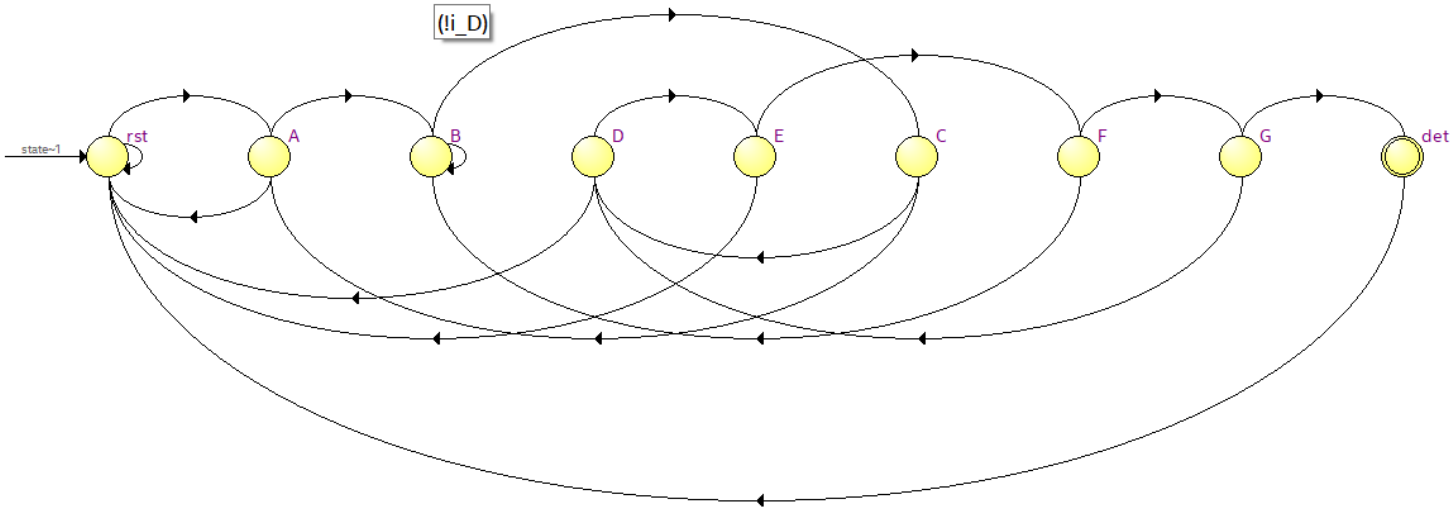
- Sequence detector - CD



© tj

Simple FSMs

- Sequence detector - CD



State Table

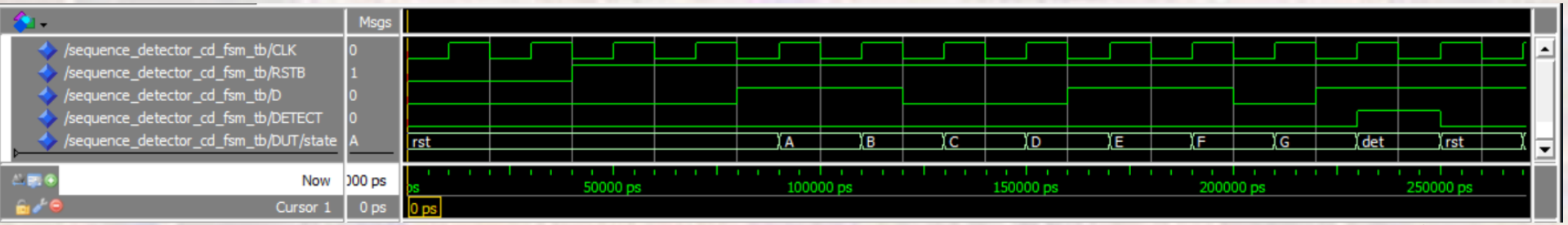
	Source State	Destination State	Condition
1	A	B	(i_D)
2	A	rst	(!i_D)
3	B	C	(!i_D)
4	B	B	(i_D)
5	C	D	(!i_D)
6	C	A	(i_D)
7	D	E	(i_D)

Transitions / Encoding

© tj

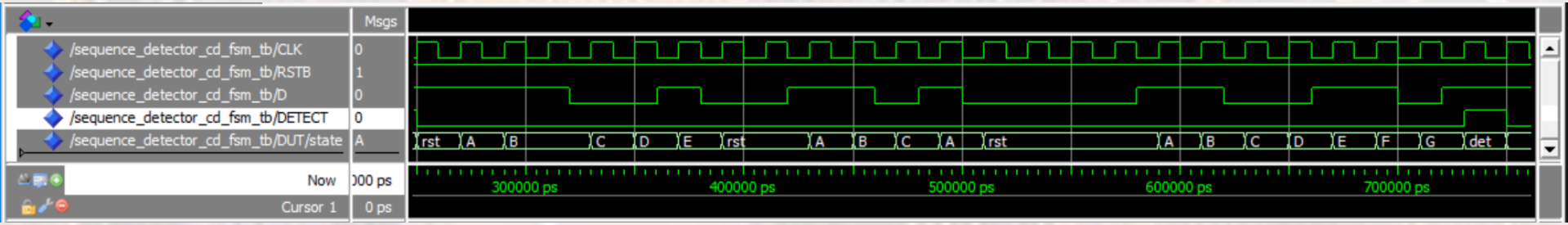
Simple FSMs

- Sequence detector - CD



full sequence

re-start



repeat B

Back to start

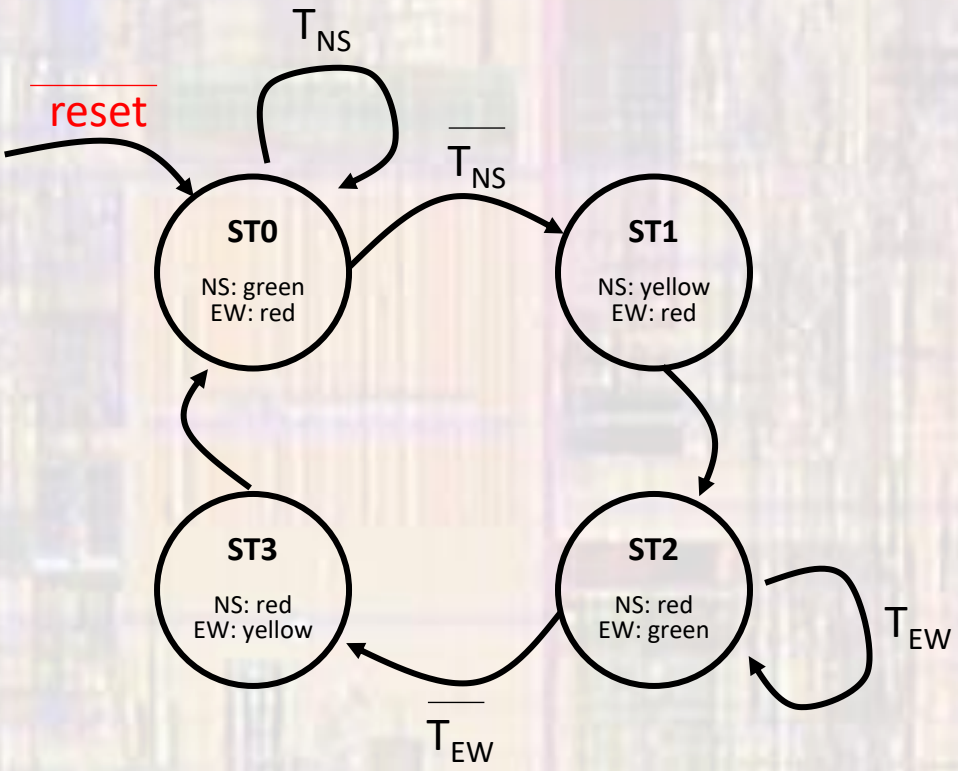
Back to A

Back to start

© tj

Simple FSMs

- Priority Stoplight
  - Stays in current mode if traffic is detected in that direction



 © tj

Simple FSMs

- Priority Stoplight

```
----------------------------------------
--
-- stoplight_nsew_fsm.vhdl
--
-- created 3/30/18
-- tj
--
-- rev 0
----------------------------------------
--
-- NS/EW Stoplight
----------------------------------------
--
-- Inputs: rstb, clk, TNS, TEW
-- Outputs: lights_ew, lights_ns
--
----------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity stoplight_nsew_fsm is
   port (
         i_clk :     in std_logic;
         i_rstb :    in std_logic;
         i_tns :     in std_logic;
         i_tew :     in std_logic;

         o_lights_ns : out std_logic_vector(1 downto 0);
         o_lights_ew : out std_logic_vector(1 downto 0)

    );
end entity;
```

```
architecture behavioral of stoplight_nsew_fsm is
    --
    -- internal signals
    --
    type STATE_TYPE is (GR, YR, RG, RY);
    signal state:        STATE_TYPE;
    signal state_next:   STATE_TYPE;

    constant R  :  std_logic_vector(1 downto 0)  := "11";
    constant Y  :  std_logic_vector(1 downto 0)  := "10";
    constant G  :  std_logic_vector(1 downto 0)  := "01";

begin
    --
    -- next state logic
    --
    process(all)
    begin
       case state is
          when GR =>
             if(i_tns = '1') then
                state_next <= GR;
             else
                state_next <= YR;
             end if;
          when YR =>
             state_next <= RG;
          when RG =>
             if(i_tew = '1') then
                state_next <= RG;
             else
                state_next <= RY;
             end if;
          when others =>
             state_next <= GR;
       end case;
    end process;
```


© tj

Simple FSMs

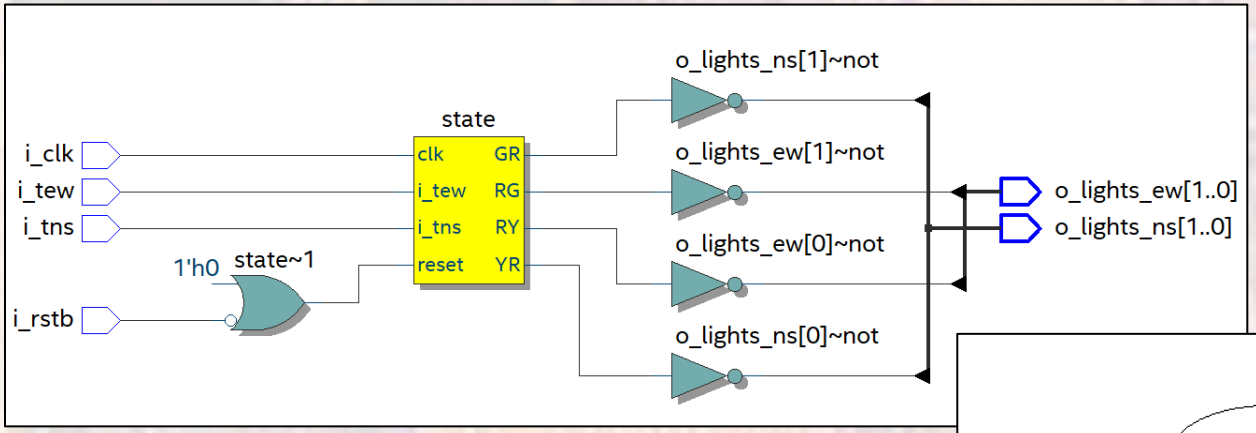
- Priority Stoplight

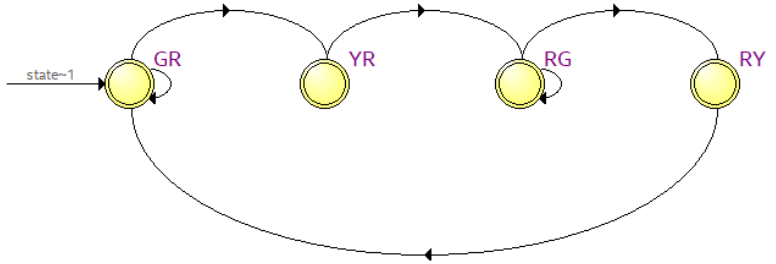
```
    --
    -- Register logic
    --
    process(i_clk, i_rstb)
    begin
        -- reset
        if (i_rstb = '0') then
            state <= GR;
        -- rising clk edge
        elsif (rising_edge(i_clk)) then
            state <= state_next;
        end if;
    end process;

    --
    -- Output logic
    --
    process(all)
    begin
        case state is
            when GR =>
                o_lights_ns <= G;
                o_lights_ew <= R;
            when YR =>
                o_lights_ns <= Y;
                o_lights_ew <= R;
            when RG =>
                o_lights_ns <= R;
                o_lights_ew <= G;
            when others =>
                o_lights_ns <= R;
                o_lights_ew <= Y;
            end case;
    end process;

end behavioral;
```


© tj

Simple FSMs

- Priority Stoplight



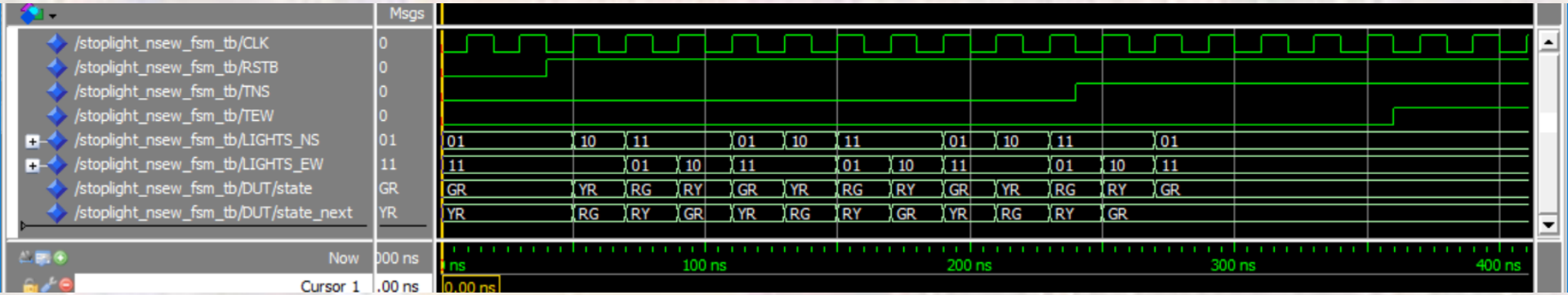


	Source State	Destination State	Condition
1	GR	YR	(!i_tns)
2	GR	GR	(i_tns)
3	RG	RY	(!i_tew)
4	RG	RG	(i_tew)
5	RY	GR	
6	YR	RG	

© tj

Simple FSMs

- Priority Stoplight



normal cycle

N/S Priority

© tj