Counter Based Finite State Machines

Last updated 5/18/20

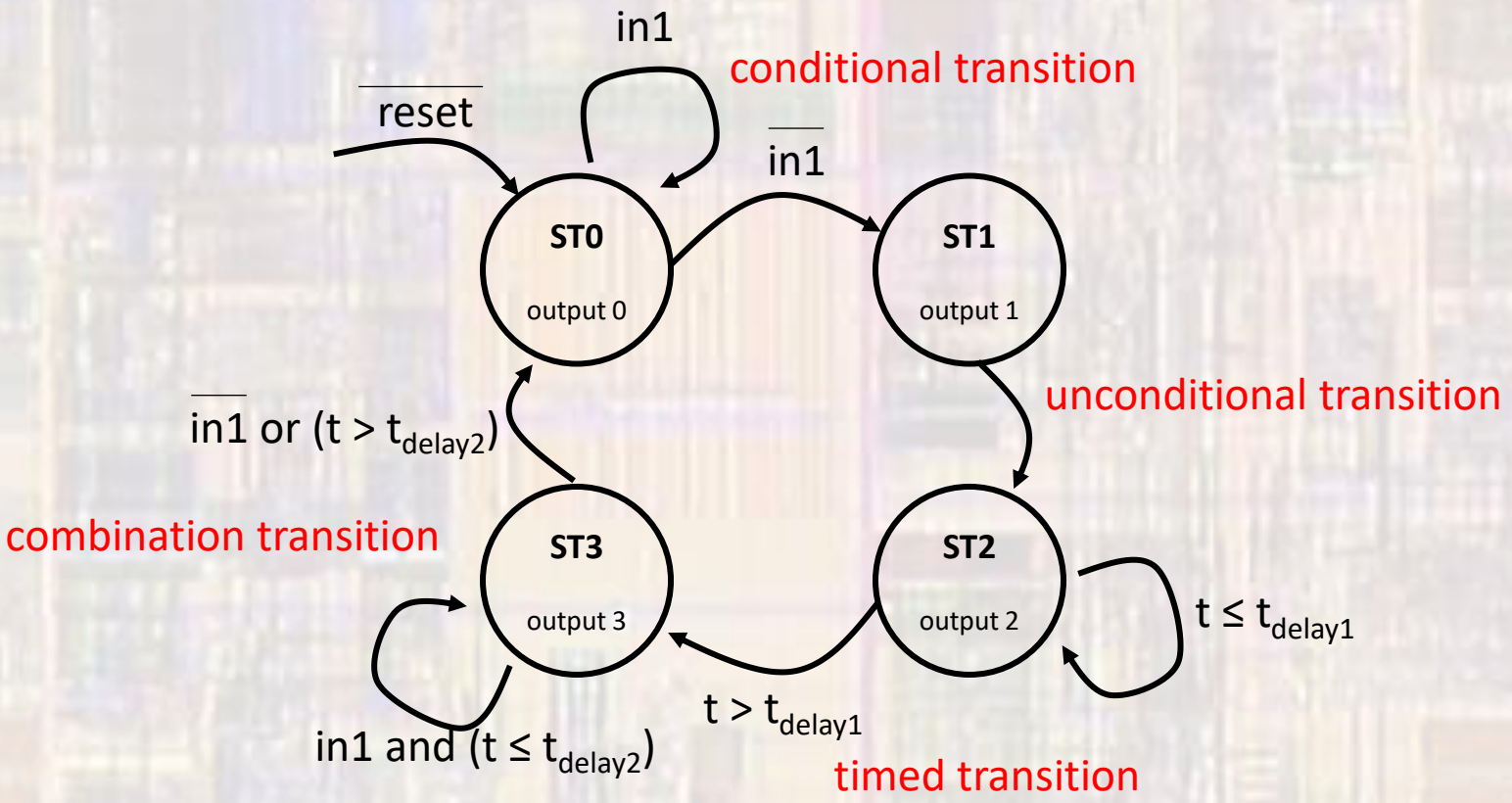
Counter Based FSMs

These slides review Counter based HDL Finite State Machines

Upon completion: You should be able to design and simulate counter based FSMs

Counter Based FSMs

- Complex FSMs

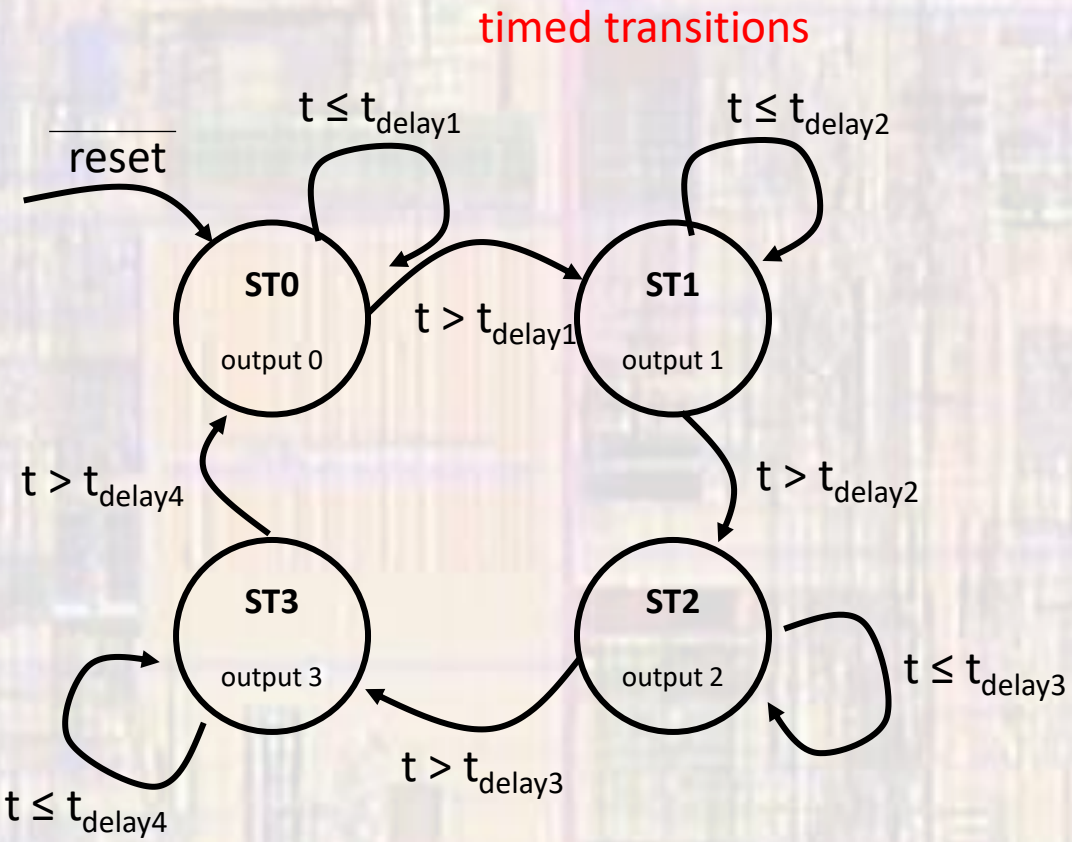


Counter Based FSMs

Timed FSMs

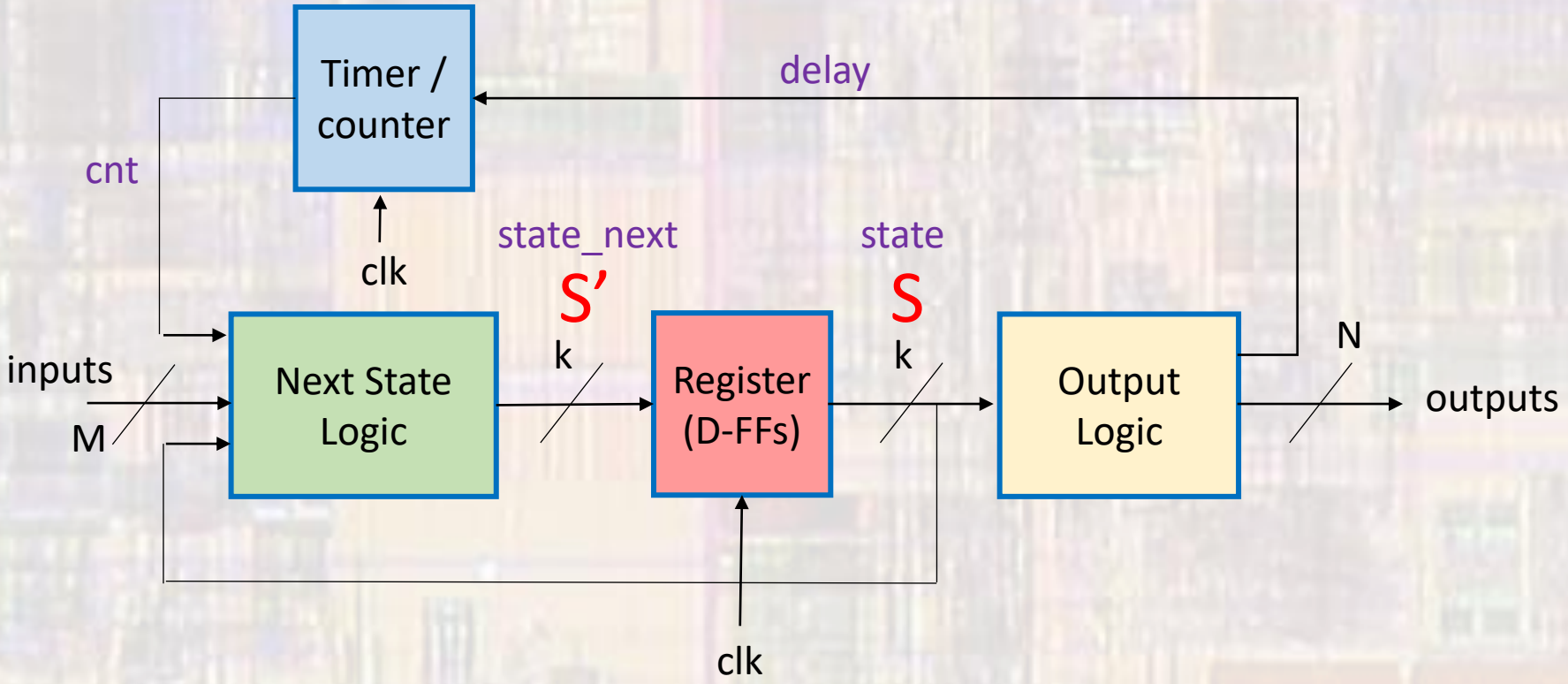
Counter Based FSMs

- Timed FSMs



Counter Based FSMs

- Timed FSMs



Counter Based FSMs

- Timed FSMs
  - Generalized approach
    - Use a timer (counter)
    - Timing based on clock cycles
    - Reset the timer when it reaches a pre-defined value
    - Check the timer before changing states

```
--
-- Timer
--
process(i_clk, i_rstb)
begin
   -- reset
   if (i_rstb = '0') then
      cnt <= (others => '0');
   -- rising clk edge
   elsif (rising_edge(i_clk)) then
      if(cnt < delay - 1) then
         cnt <= cnt + 1;
      else
         cnt <= (others => '0');
      end if;
   end if;
end process;
```

Separate from the state logic

Size of cnt and delay dependent on longest delay

delay set in the output logic
(state dependent)

Counter Based FSMs

- Timed FSMs – stoplight
  - 10sec GR
  - 1sec YR
  - 14 sec RG
  - 5 sec RY

Counter Based FSMs

- Timed FSMs – stoplight

```
--------------------------------------
--
-- stoplight_timed_fsm.vhdl
--
-- created 3/30/18
-- tj
--
-- version 3
-- rev 0
--------------------------------------
--
-- Timed Stoplight
--------------------------------------
--
-- Inputs: rstb, clk
-- Outputs: lightsA, lightsB
--          01->Green, 10->Yellow, 11->red
--
--------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity stoplight_timed_fsm is
   generic(
            N:       natural  := 5;       -- # of count bits
            T_GR:    unsigned := "01010"; -- delays
            T_YR:    unsigned := "00001";
            T_RG:    unsigned := to_unsigned(14, 5);
            T_RY:    unsigned := to_unsigned(5, 5)
   );
   port (
         i_clk :     in std_logic;
         i_rstb :    in std_logic;

         o_lights_A : out std_logic_vector(1 downto 0);
         o_lights_B : out std_logic_vector(1 downto 0)
    );
end entity;
```

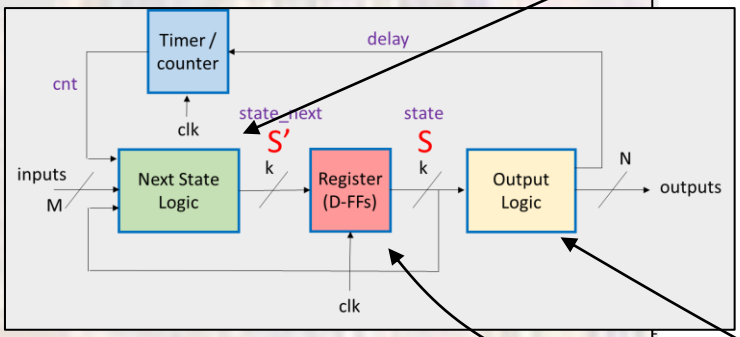
```
architecture behavioral of stoplight_timed_fsm is
    --
    -- State Types
    --
    type STATE_TYPE is (GR, YR, RG, RY);
    signal state:        STATE_TYPE;
    signal state_next:   STATE_TYPE;
    --
    -- Timer signals
    --
    signal cnt:    unsigned((N - 1) downto 0);
    signal delay:  unsigned((N - 1) downto 0);
    --
    -- Convenience Constants
    --
    constant green:   std_logic_vector  := "01";
    constant yellow:  std_logic_vector  := "10";
    constant red:     std_logic_vector  := "11";

begin
```

Counter Based FSMs

- Timed FSMs – stoplight

```
--
-- Timer
--
process(i_clk, i_rstb)
begin
   -- reset
   if (i_rstb = '0') then
      cnt <= (others => '0');
   -- rising clk edge
   elsif (rising_edge(i_clk)) then
      if(cnt < delay - 1) then
         cnt <= cnt + 1;
      else
         cnt <= (others => '0');
      end if;
   end if;
end process;
```




```
--
-- next state logic
--
process(all)
begin
   case state is
      when GR =>
         if(cnt < T_GR - 1) then
            state_next <= GR;
         else
            state_next <= YR;
         end if;
      when YR =>
         if(cnt < T_YR - 1) then
            state_next <= YR;
         else
            state_next <= RG;
         end if;
      when RG =>
         if(cnt < T_RG - 1) then
            state_next <= RG;
         else
            state_next <= RY;
         end if;
      when others =>
         if(cnt < T_RY - 1) then
            state_next <= RY;
         else
            state_next <= GR;
         end if;
   end case;
end process;
```

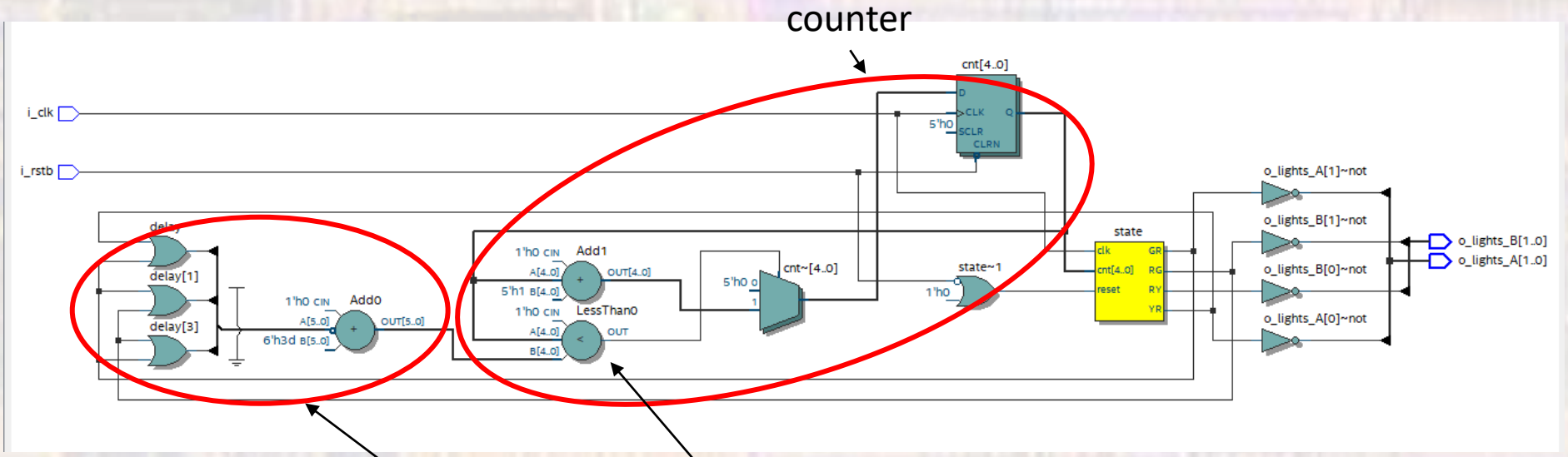
```
--
-- Register logic
--
process(i_clk, i_rstb)
begin
   -- reset
   if (i_rstb = '0') then
      state <= GR;
   -- rising clk edge
   elsif (rising_edge(i_clk)) then
      state <= state_next;
   end if;
end process;
```

```
--
-- Output logic
--
process(all)
begin
   case state is
      when GR =>
         delay <= T_GR;
         o_lights_A <= green;
         o_lights_B <= red;
      when YR =>
         delay <= T_YR;
         o_lights_A <= yellow;
         o_lights_B <= red;
      when RG =>
         delay <= T_RG;
         o_lights_A <= red;
         o_lights_B <= green;
      when others =>
         delay <= T_RY;
         o_lights_A <= red;
         o_lights_B <= yellow;
      end case;
end process;

end behavioral;
```

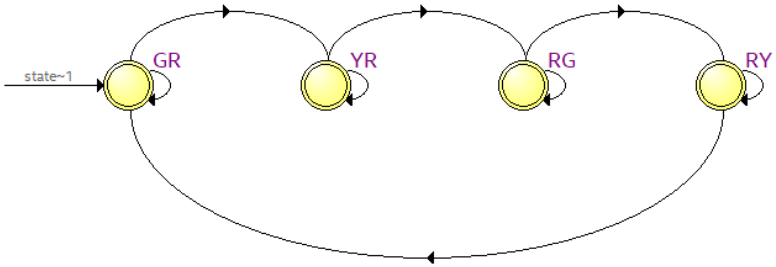
Counter Based FSMs

- Timed FSMs – stoplight



Counter Based FSMs

- Timed FSMs – stoplight

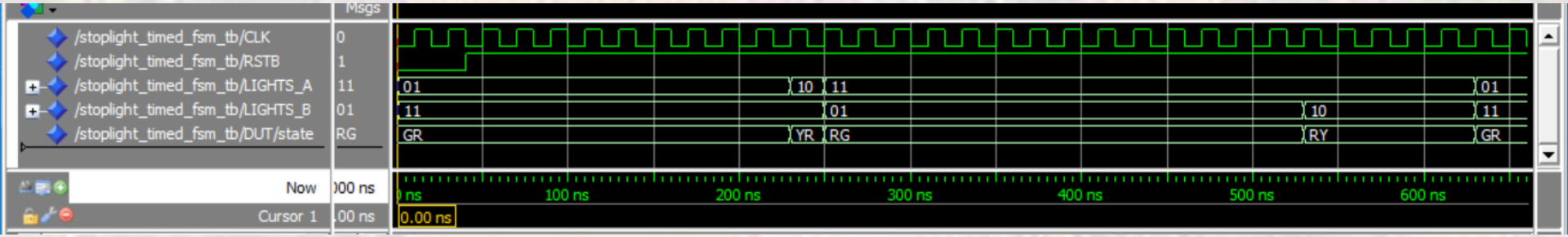


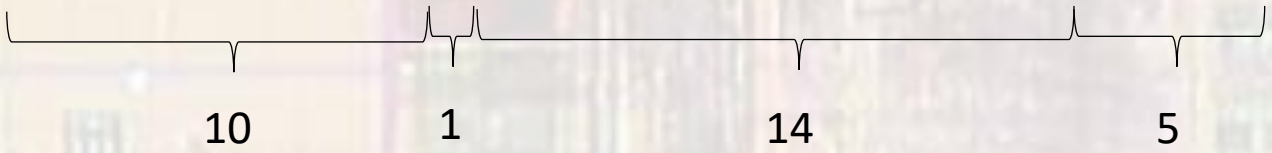
	Source State	Destination State	Condition
1	GR	YR	(!cnt[1]).(!cnt[2]).(cnt[4]) + (!cnt[1]).(cnt[2]).(!cnt[3]).(cnt[4]) + (!cnt[1]).(cnt[2]).(cnt[3]) + (cnt[1]).(!cnt[3]).(cnt[4]) + (cnt[1]).(cnt[
2	GR	GR	(!cnt[1]).(!cnt[2]).(!cnt[4]) + (!cnt[1]).(cnt[2]).(!cnt[3]).(!cnt[4]) + (cnt[1]).(!cnt[3]).(!cnt[4])
3	RG	RY	(!cnt[1]).(cnt[4]) + (cnt[1]).(!cnt[2]).(cnt[4]) + (cnt[1]).(cnt[2]).(!cnt[3]).(cnt[4]) + (cnt[1]).(cnt[2]).(cnt[3])
4	RG	RG	(!cnt[1]).(!cnt[4]) + (cnt[1]).(!cnt[2]).(!cnt[4]) + (cnt[1]).(cnt[2]).(!cnt[3]).(!cnt[4])
5	RY	RY	(!cnt[0]).(!cnt[1]).(!cnt[3]).(!cnt[4]) + (!cnt[0]).(cnt[1]).(!cnt[2]).(!cnt[3]).(!cnt[4]) + (cnt[0]).(!cnt[2]).(!cnt[3]).(!cnt[4])
6	RY	GR	(!cnt[0]).(!cnt[1]).(!cnt[3]).(cnt[4]) + (!cnt[0]).(!cnt[1]).(cnt[3]) + (!cnt[0]).(cnt[1]).(!cnt[2]).(!cnt[3]).(cnt[4]) + (!cnt[0]).(cnt[1]).(!cnt
7	YR	YR	(!cnt[0]).(!cnt[1]).(!cnt[2]).(!cnt[3]).(!cnt[4])
8	YR	RG	(!cnt[0]).(!cnt[1]).(!cnt[2]).(!cnt[3]).(cnt[4]) + (!cnt[0]).(!cnt[1]).(!cnt[2]).(cnt[3]) + (!cnt[0]).(!cnt[1]).(cnt[2]) + (!cnt[0]).(cnt[1]) + (c

Counter Based FSMs

- Timed FSMs – stoplight

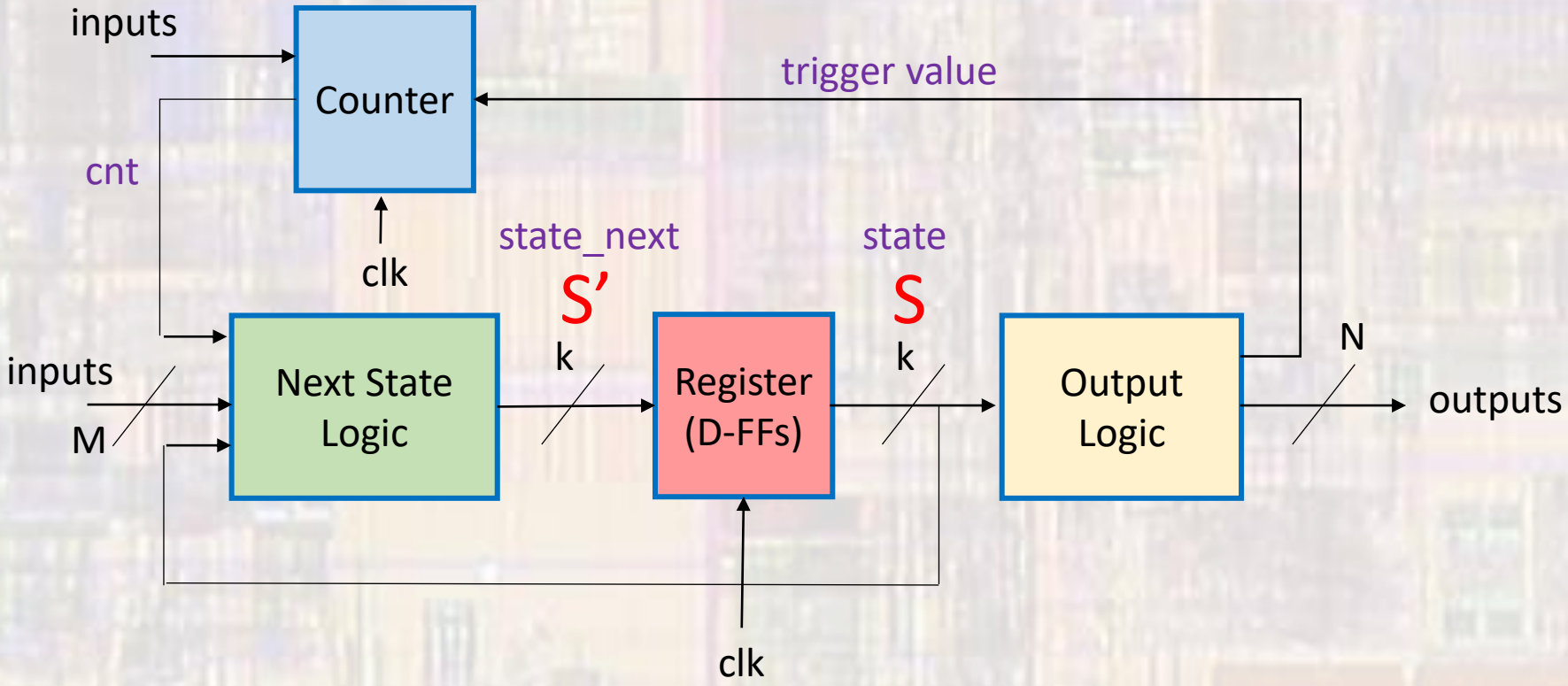
```
T_GR:    unsigned := "01010";
T_YR:    unsigned := "00001";
T_RG:    unsigned := to_unsigned(14, 5);
T_RY:    unsigned := to_unsigned(5, 5)
```





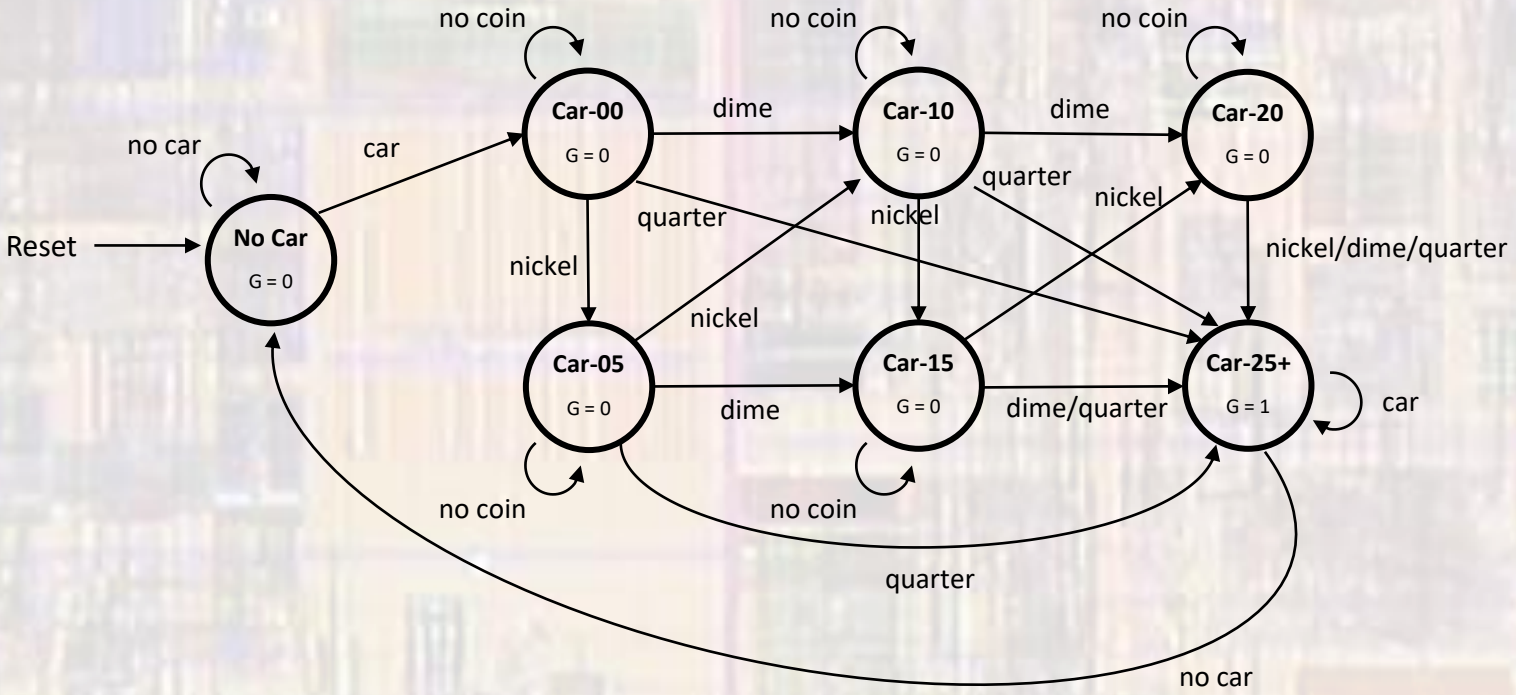
Counter Based FSMs

- Generalized counting FSMs



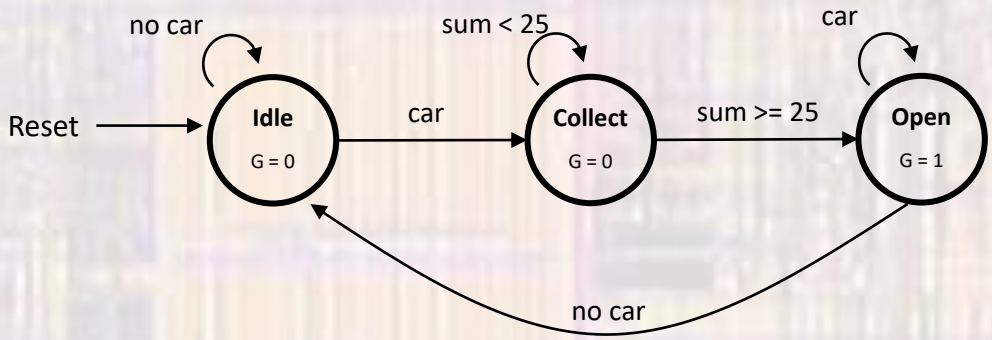
Counter Based FSMs

- Toll booth – terrible version



Counter Based FSMs

- Toll booth



Counter Based FSMs

- Toll booth

```
--------------------------------------
--
-- toll_booth_fsm.vhdl
--
-- created 4/9/18
-- tj
--
-- rev 0
--------------------------------------
--
-- toll booth FSM example
--
-- cash handled entirely in register
--------------------------------------
--
-- Inputs: rstb, clk, N, D, Q, car
-- Outputs: gate
--
--------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity tollbooth_fsm is
   port (
         i_clk    :  in std_logic;
         i_rstb   :  in std_logic;
         i_N      :  in std_logic;
         i_D      :  in std_logic;
         i_Q      :  in std_logic;
         i_car    :  in std_logic;

         o_gate   :  out std_logic
    );
end entity;
```

```
architecture behavioral of tollbooth_fsm is
   --
   -- State Types
   --
   type STATE_TYPE is (Idle, Collect, Gate_open);
   signal state:        STATE_TYPE;
   signal state_next:   STATE_TYPE;
   --
   -- Counter signals
   --
   signal cash:      unsigned(7 downto 0);

begin
```

Counter Based FSMs

- Toll booth

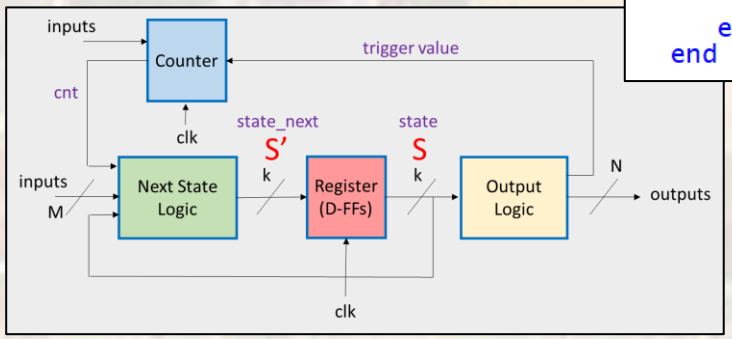
```
--
--  cash counter
--
process(i_clk, i_rstb)
begin
   -- reset
   if (i_rstb = '0') then
      cash <= (others => '0');
   -- rising clk edge
   elsif (rising_edge(i_clk)) then
      if (i_N = '1') then
         cash <= cash + 5;
      elsif (i_D = '1') then
         cash <= cash + 10;
      elsif (i_Q = '1') then
         cash <= cash + 25;
      elsif (state = Idle) then
         cash <= (others => '0');
      else
         cash <= cash;
      end if;
   end if;
end process;
```

```
--
-- next state logic
--
process(all)
begin
   case state is
      when Idle=>
         if(i_car = '1') then
            state_next <= Collect;
         else
            state_next <= state;
         end if;
      when Collect =>
         if(cash >= 25) then
            state_next <= Gate_open;
         else
            state_next <= state;
         end if;
      when Gate_open =>
         if(i_car = '0') then
            state_next <= Idle;
         else
            state_next <= state;
         end if;
      when others =>
         state_next <= Idle;
   end case;
end process;
```

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

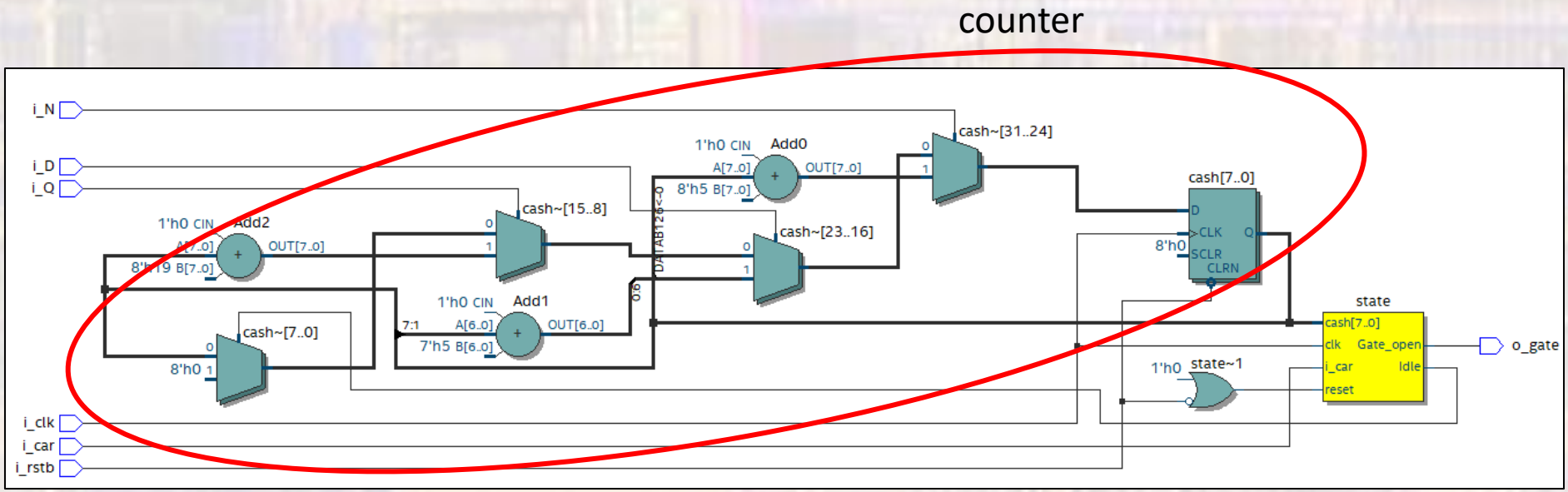
```
--
-- Output logic
--
process(all)
begin
   case state is
      when Idle =>
         o_gate <=   '0';
      when Collect =>
         o_gate <=   '0';
      when Gate_open =>
         o_gate <=   '1';
      when others =>
         o_gate <=   '0';
   end case;
end process;

end behavioral;
```



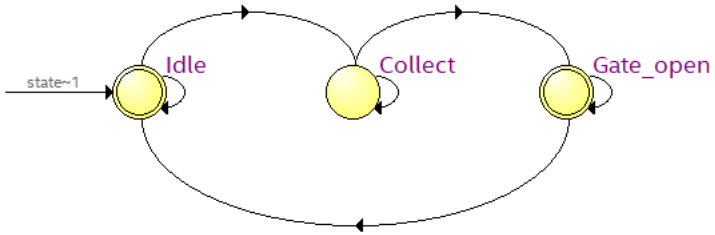
Counter Based FSMs

- Toll booth



Counter Based FSMs

- Toll booth



	Source State	Destination State	
1	Collect	Gate_open	(!cash[0]).(!cash[1]).(!cash[2]).(!cash[5]).(!cash[6]).(cash[7]) + (!cash[0]).(!cash[1]).(!cash[2]).(!cash[5]).(cash[6]) + (!cash[0]).(!cas
2	Collect	Collect	(!cash[0]).(!cash[1]).(!cash[2]).(!cash[5]).(!cash[6]).(!cash[7]) + (!cash[0]).(!cash[1]).(cash[2]).(!cash[3]).(!cash[5]).(!cash[6]).(!cash
3	Gate_open	Idle	(!i_car)
4	Gate_open	Gate_open	(i_car)
5	Idle	Idle	(!i_car)
6	Idle	Collect	(i_car)

Counter Based FSMs

- Toll Booth