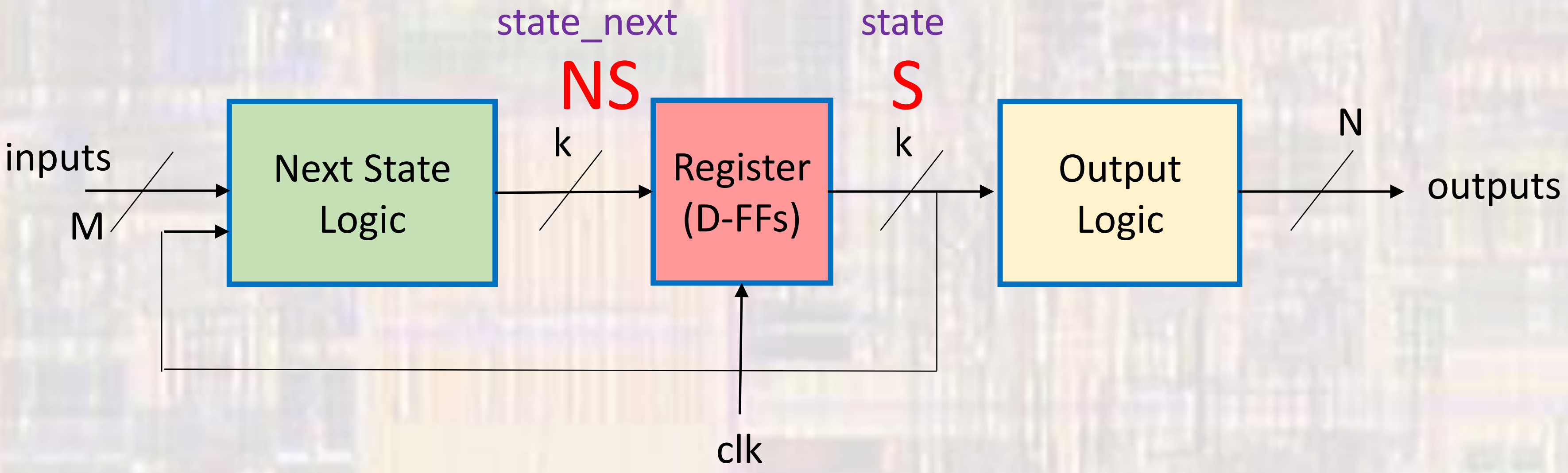
HDL FSM Timers/Counters

Last updated 1/22/21

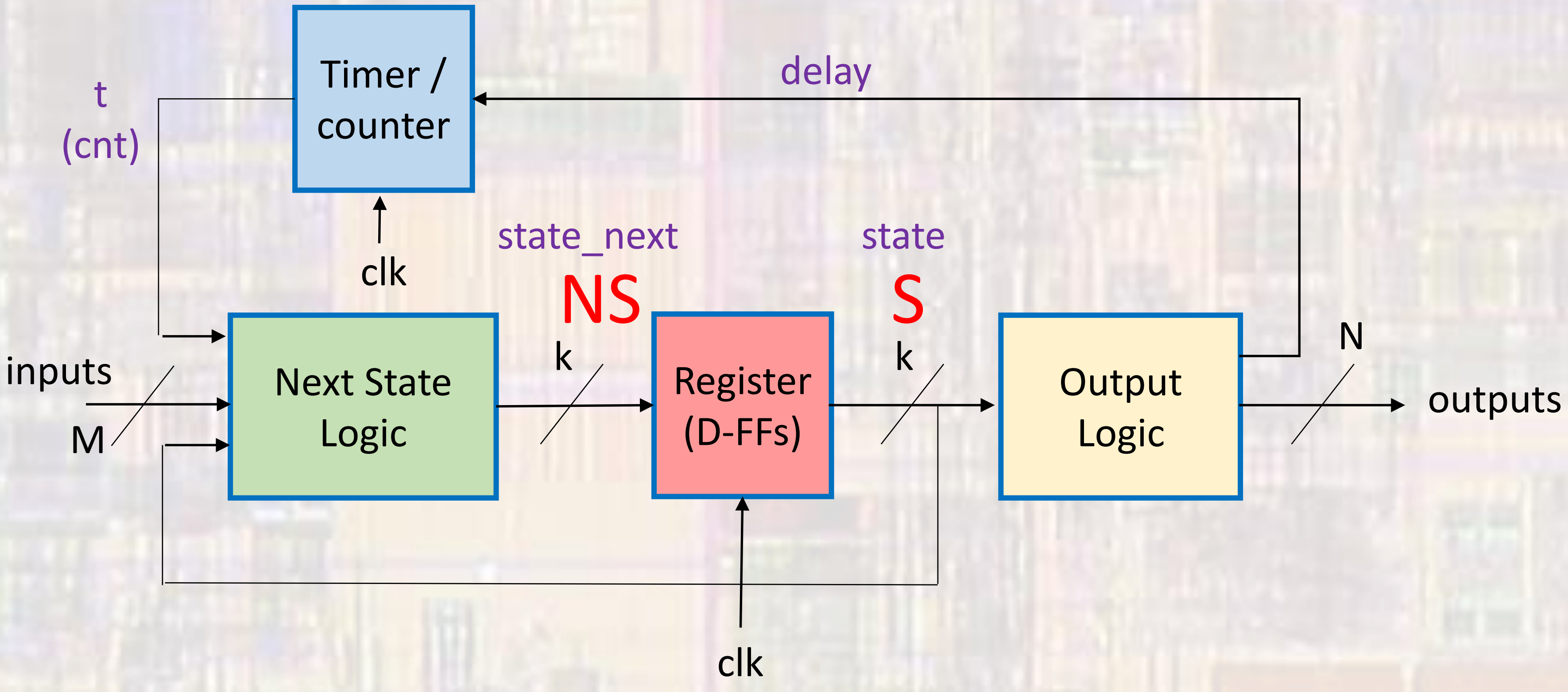
HDL FSM Timers/Counters

- FSM



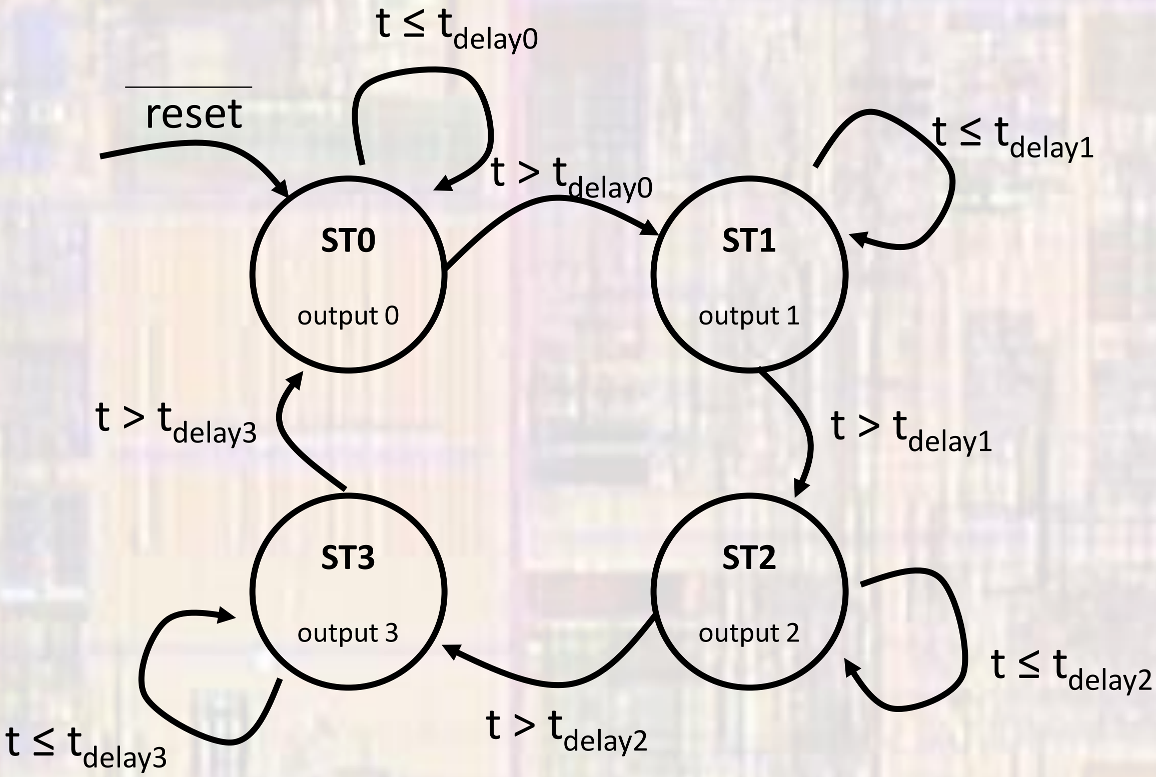
HDL FSM Timers/Counters

- Timed FSMs

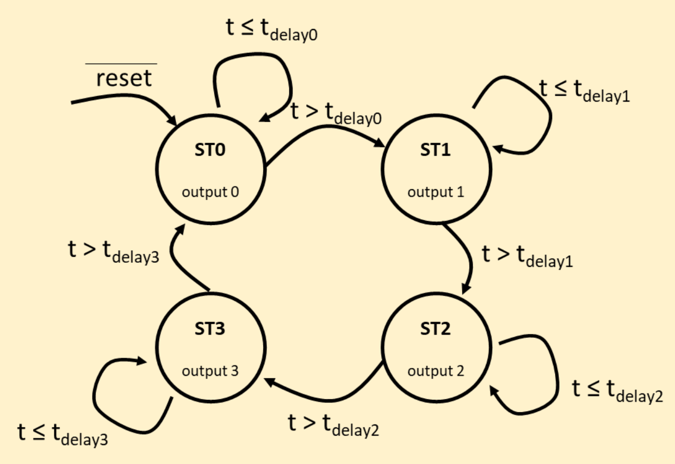


HDL FSM Timers/Counters

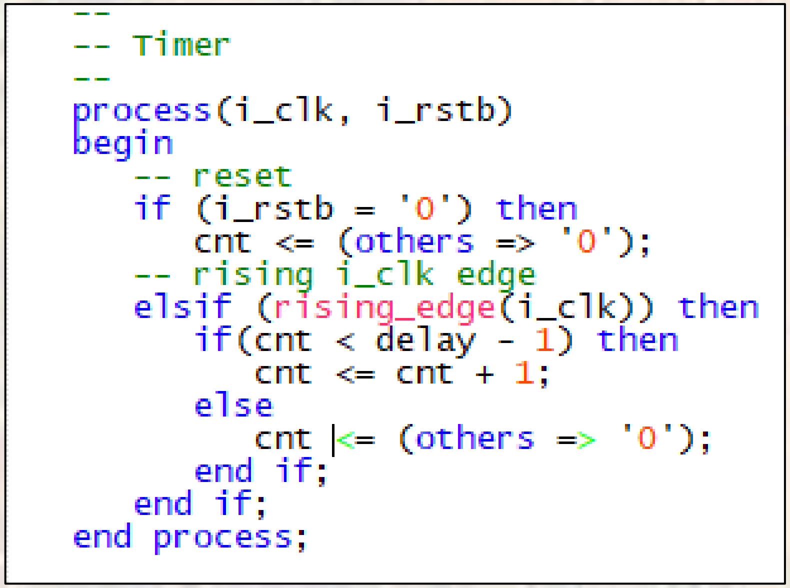
- Timed FSMs



HDL FSM Timers/Counte



- 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 i_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;
```

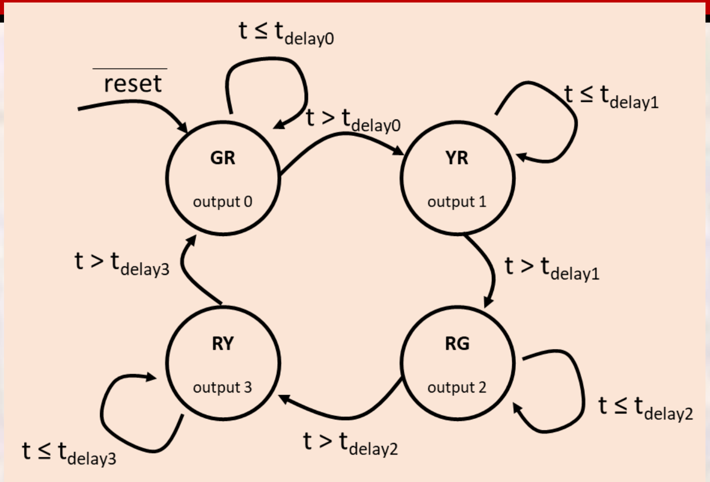
Separate from the state logic

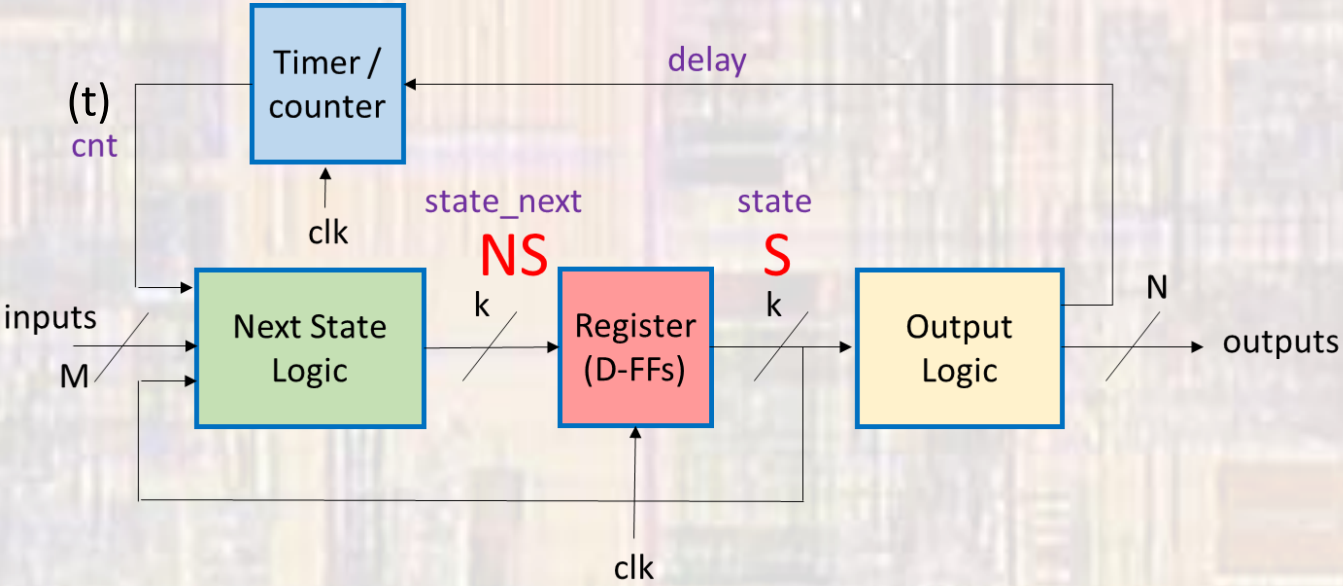
Size of cnt and delay dependent on longest delay

delay set in the output logic
(state dependent)

HDL FSM Timers/Counters

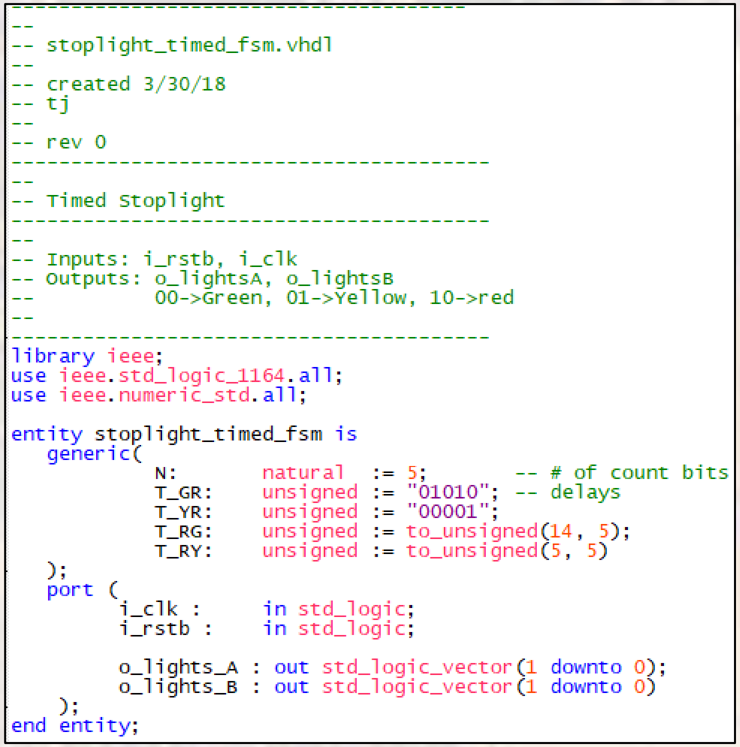
- Timed FSMs – stoplight
  - 10sec GR
  - 1sec YR
  - 14 sec RG
  - 5 sec RY



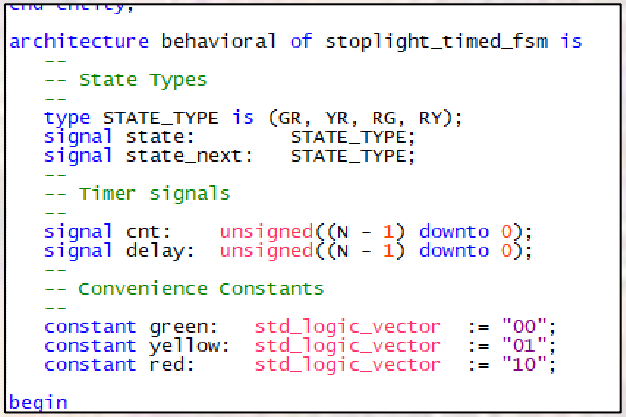


HDL FSM Timers/Counters

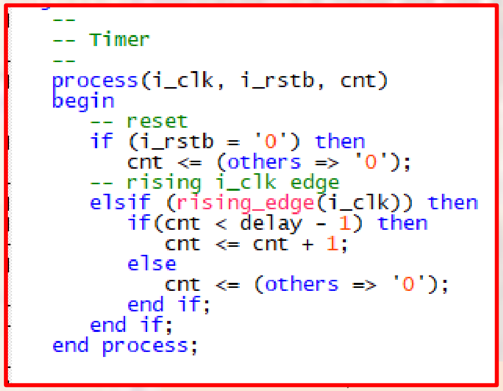
- Timed FSMs – stoplight



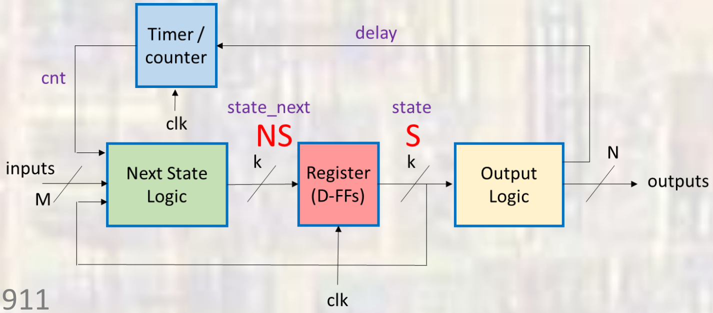
```vhdl
--
-- stoplight_timed_fsm.vhdl
--
-- created 3/30/18
-- tj
--
-- rev 0
----------------------------------------
--
-- Timed Stoplight
----------------------------------------
--
-- Inputs: i_rstb, i_clk
-- Outputs: o_lightsA, o_lightsB
--          00->Green, 01->Yellow, 10->red
--
----------------------------------------
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;
```

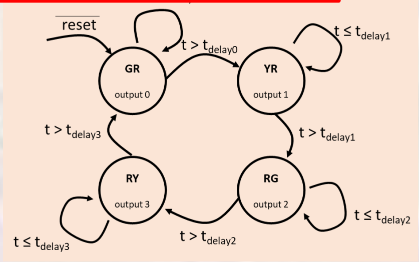


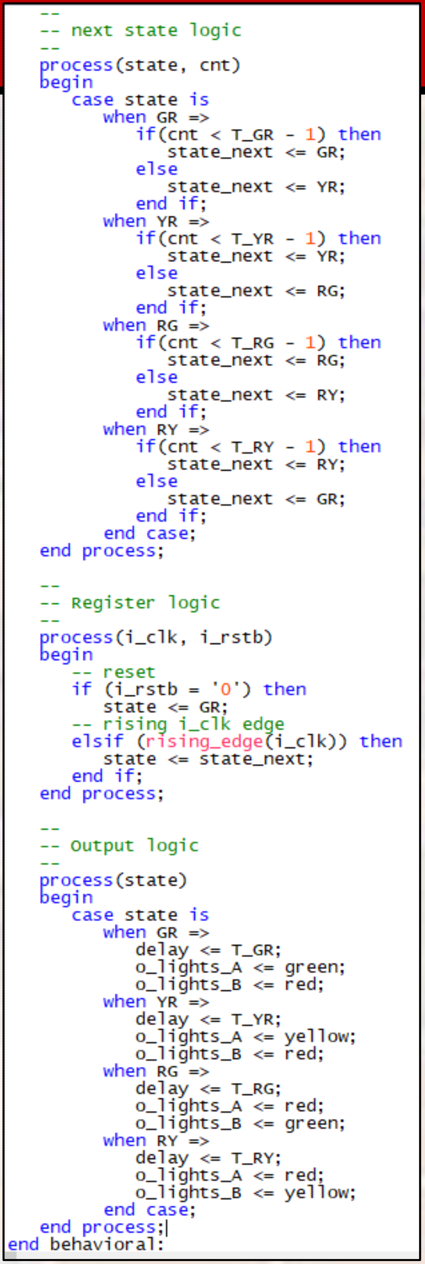
```vhdl
end entity;

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    := "00";
   constant yellow:  std_logic_vector    := "01";
   constant red:     std_logic_vector    := "10";

begin
```



```vhdl
   --
   -- Timer
   --
   process(i_clk, i_rstb, cnt)
   begin
      -- reset
      if (i_rstb = '0') then
         cnt <= (others => '0');
      -- rising i_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;
```



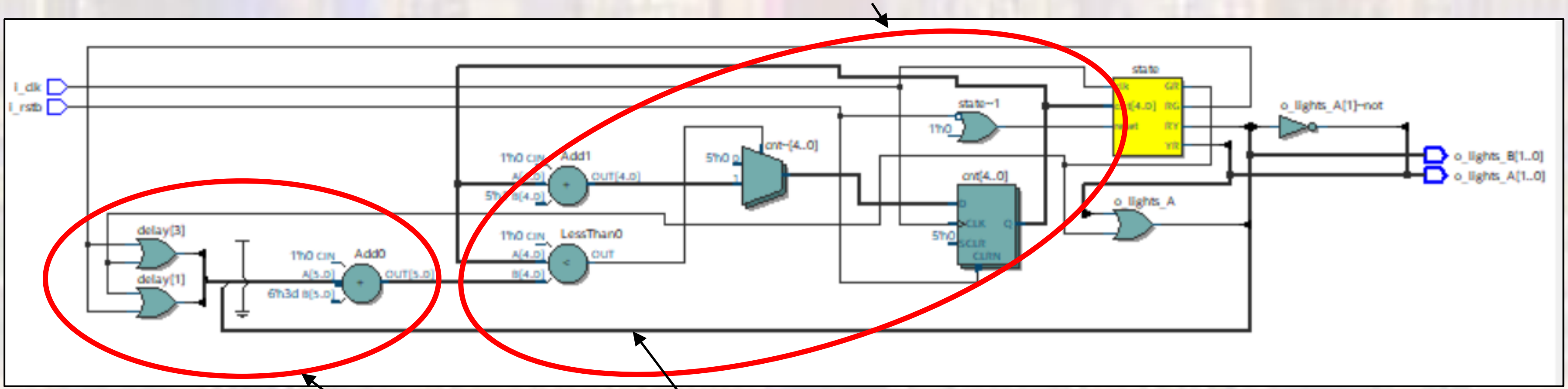




```vhdl
   --
   -- next state logic
   --
   process(state, cnt)
   begin
      case state is
         when GR =>
            if(cnt < T_GR - 1) then
               state_next <= GR;
            else
               state_next <= YR;
            end if;
         when YR =>
            if(cnt < T_YR - 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 RY =>
            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 i_clk edge
      elsif (rising_edge(i_clk)) then
         state <= state_next;
      end if;
   end process;

   --
   -- Output logic
   --
   process(state)
   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 RY =>
            delay <= T_RY;
            o_lights_A <= red;
            o_lights_B <= yellow;
         end case;
   end process;
end behavioral;
```

HDL FSM Timers/Counters

- Timed FSMs – stoplight



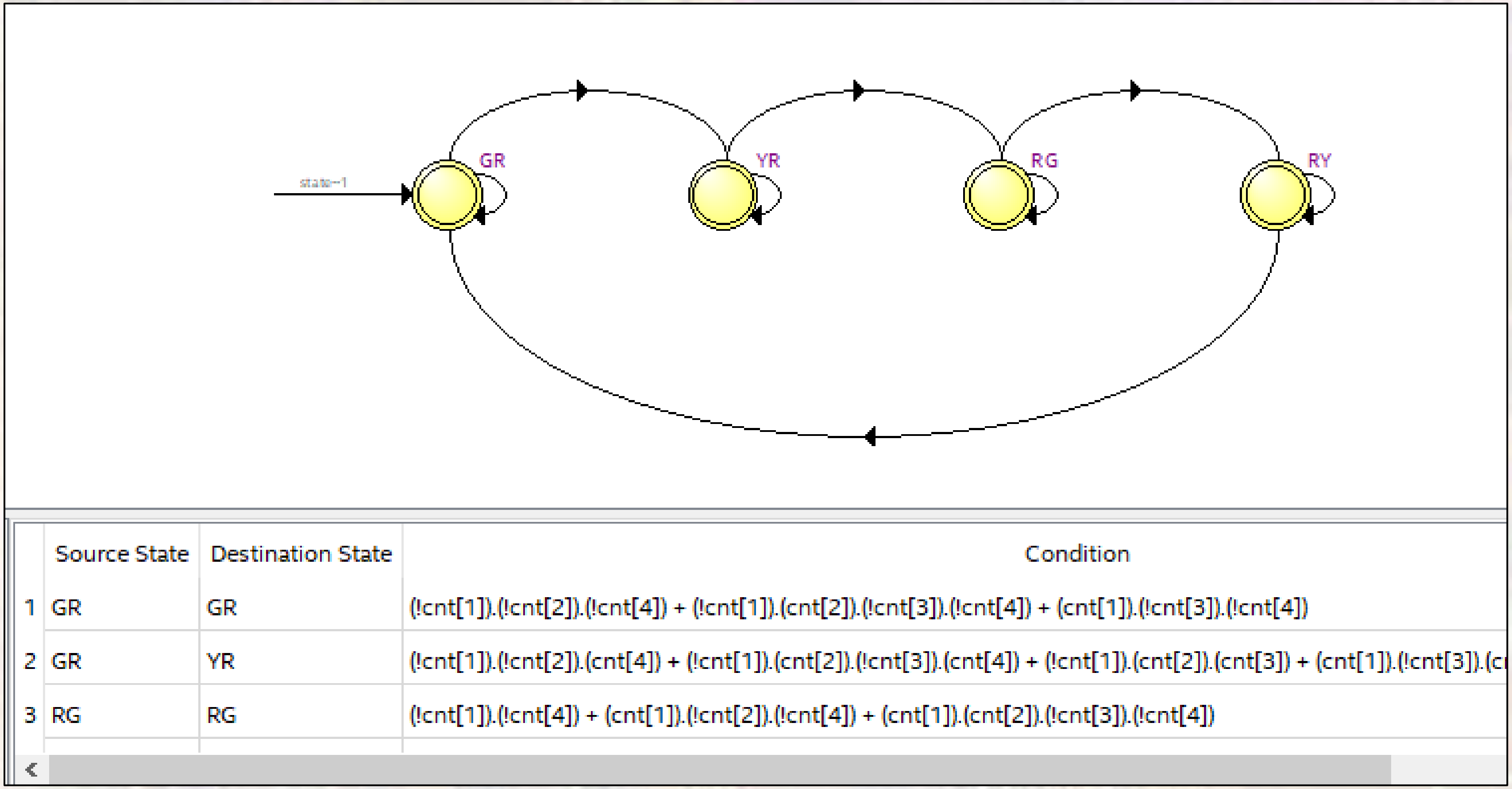
counter

"delay" creator

Roll over at "delay"

HDL FSM Timers/Counters

- Timed FSMs – stoplight

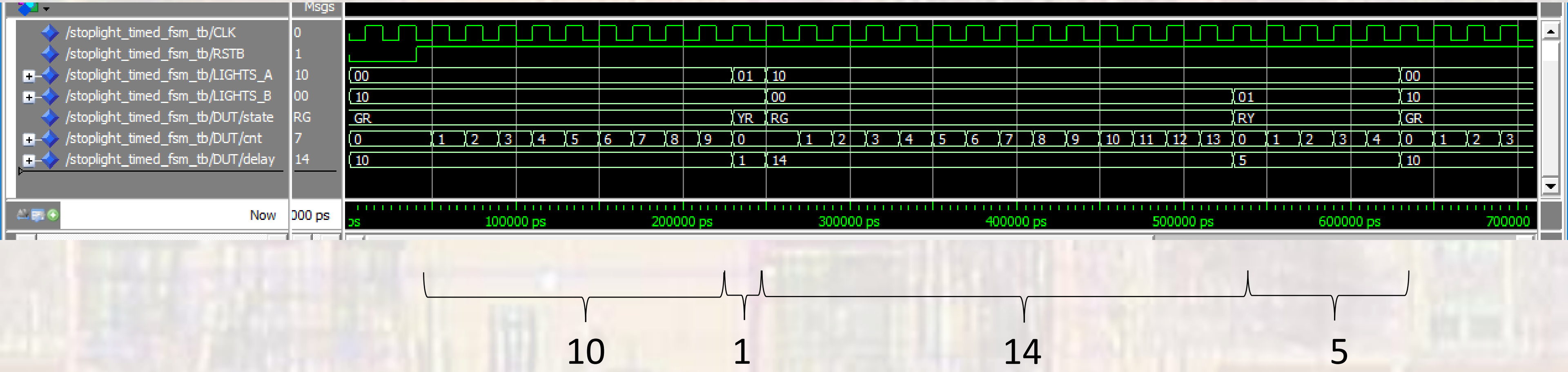


	Source State	Destination State	Condition
1	GR	GR	(!cnt[1]).(!cnt[2]).(!cnt[4]) + (!cnt[1]).(cnt[2]).(!cnt[3]).(!cnt[4]) + (cnt[1]).(!cnt[3]).(!cnt[4])
2	GR	YR	(!cnt[1]).(!cnt[2]).(cnt[4]) + (!cnt[1]).(cnt[2]).(!cnt[3]).(cnt[4]) + (!cnt[1]).(cnt[2]).(cnt[3]) + (cnt[1]).(!cnt[3]).(c
3	RG	RG	(!cnt[1]).(!cnt[4]) + (cnt[1]).(!cnt[2]).(!cnt[4]) + (cnt[1]).(cnt[2]).(!cnt[3]).(!cnt[4])

HDL FSM Timers/Counters

- Timed FSMs – stoplight

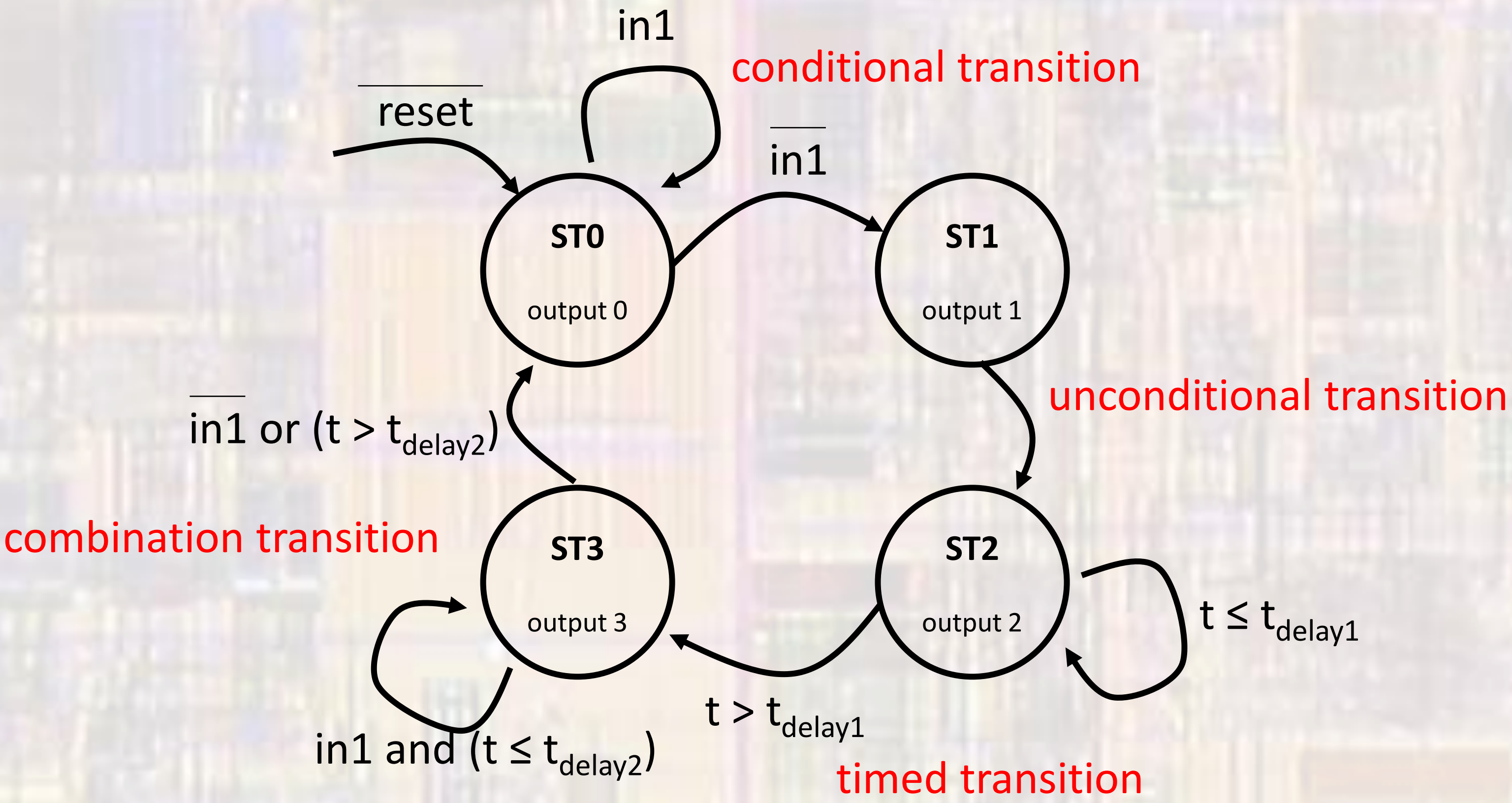
```
T_GR:    unsigned  := "01010";
T_YR:    unsigned  := "00001";
T_RG:    unsigned  := to_unsigned(14, 5);
T_RY:    unsigned  := to_unsigned(5, 5)
```



	Msgs
/stoplight_timed_fsm_tb/CLK	0
/stoplight_timed_fsm_tb/RSTB	1
/stoplight_timed_fsm_tb/LIGHTS_A	10
/stoplight_timed_fsm_tb/LIGHTS_B	00
/stoplight_timed_fsm_tb/DUT/state	RG
/stoplight_timed_fsm_tb/DUT/cnt	7
/stoplight_timed_fsm_tb/DUT/delay	14

HDL FSM Timers/Counters

- Generalized FSMs



$\overline{\text{reset}}$

in1

conditional transition

$\overline{\text{in1}}$

ST0

output 0

ST1

output 1

unconditional transition

$\overline{\text{in1}}$ or $(t > t_{delay2})$

combination transition

ST3

output 3

ST2

output 2

$t \leq t_{delay1}$

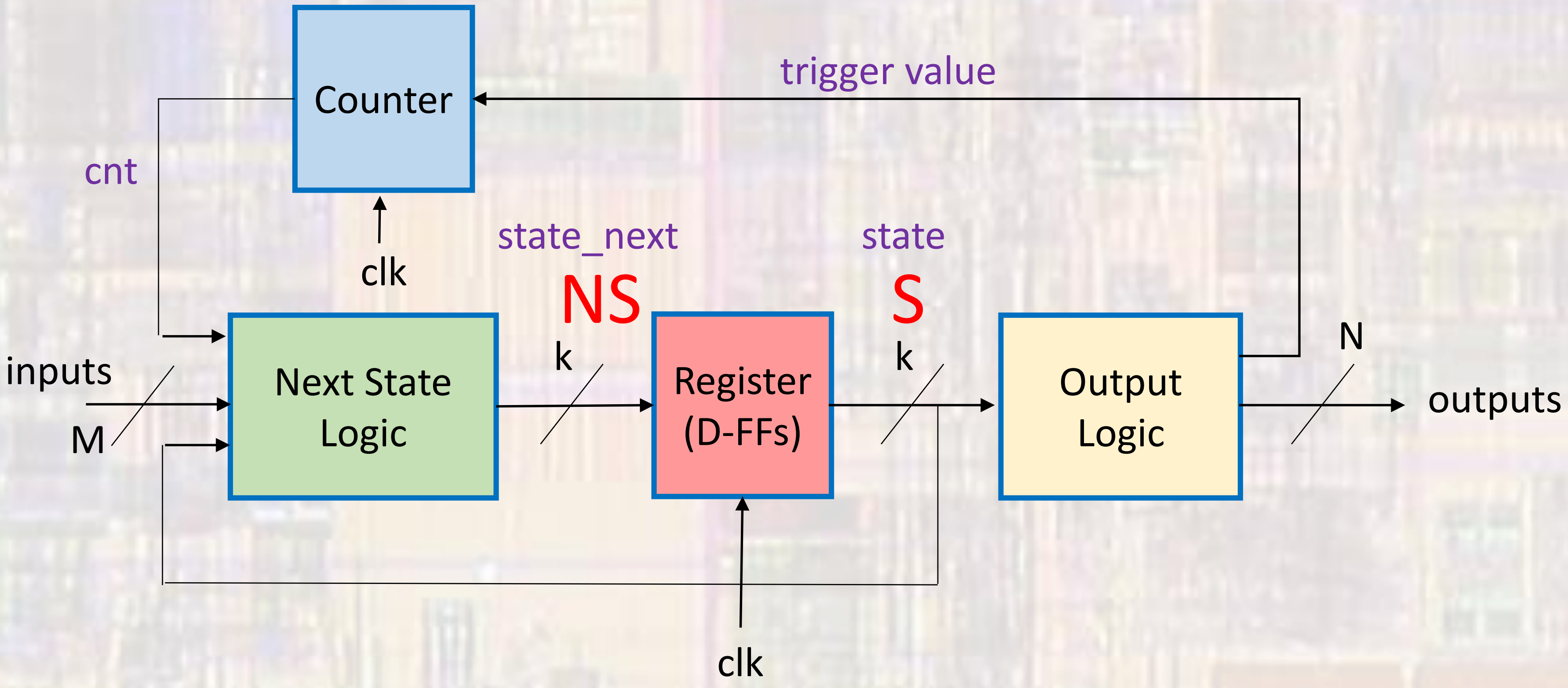
$t > t_{delay1}$

in1 and $(t \leq t_{delay2})$

timed transition

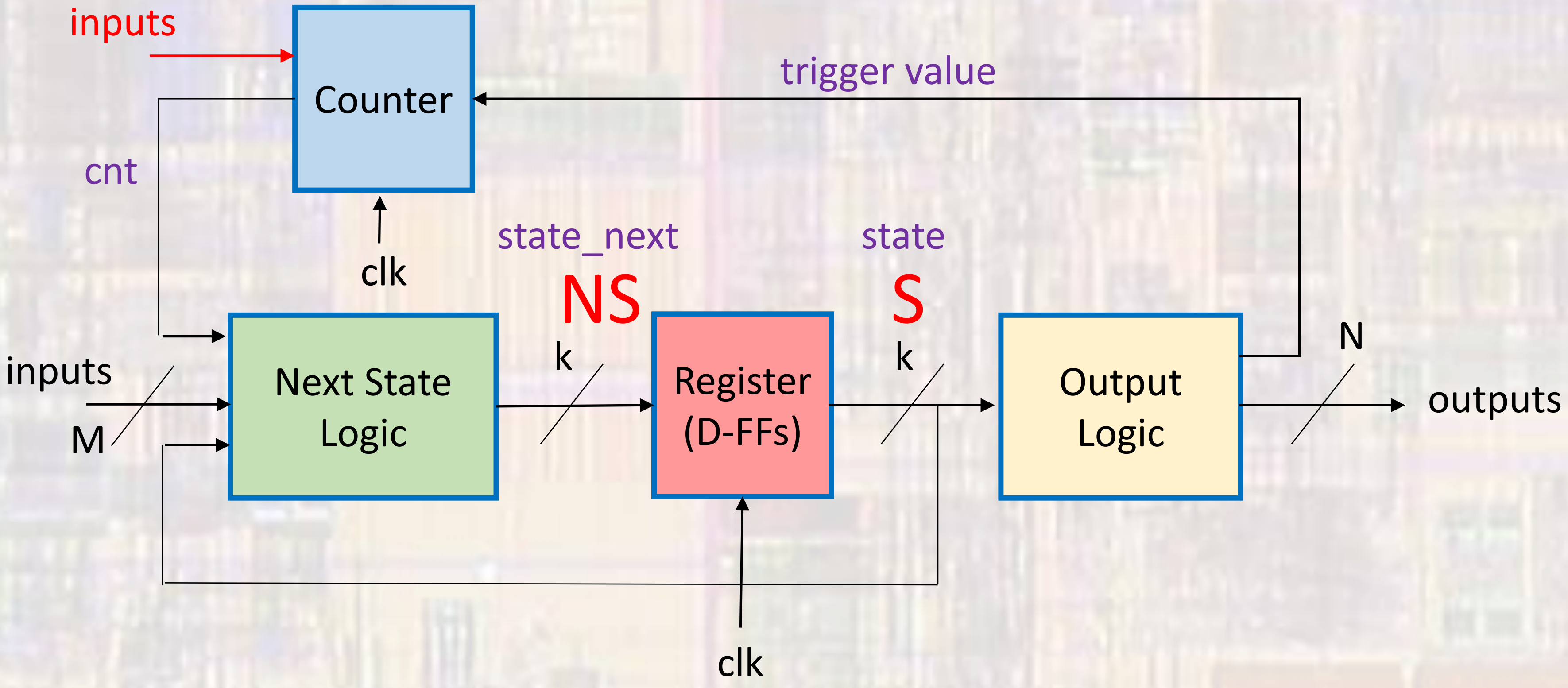
HDL FSM Timers/Counters

- Generalized counting FSMs



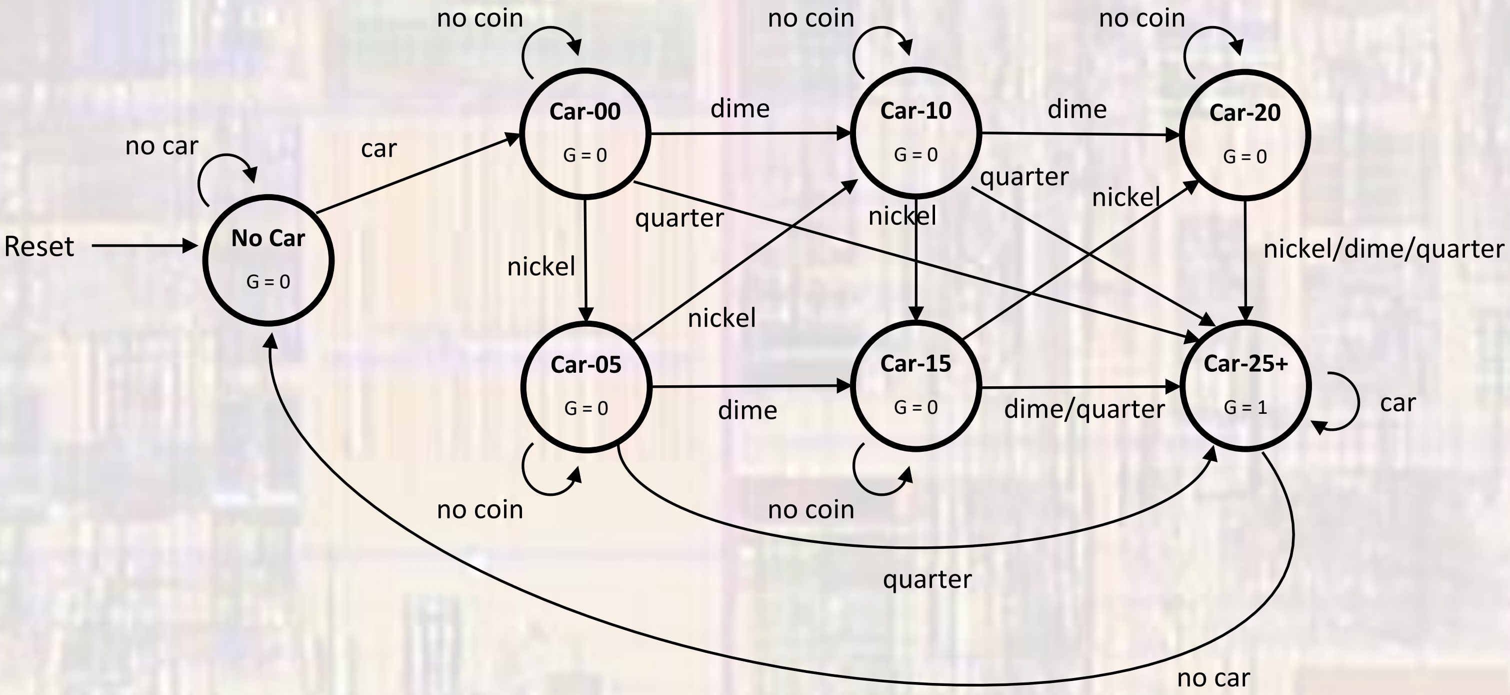
HDL FSM Timers/Counters

- Generalized counting FSMs



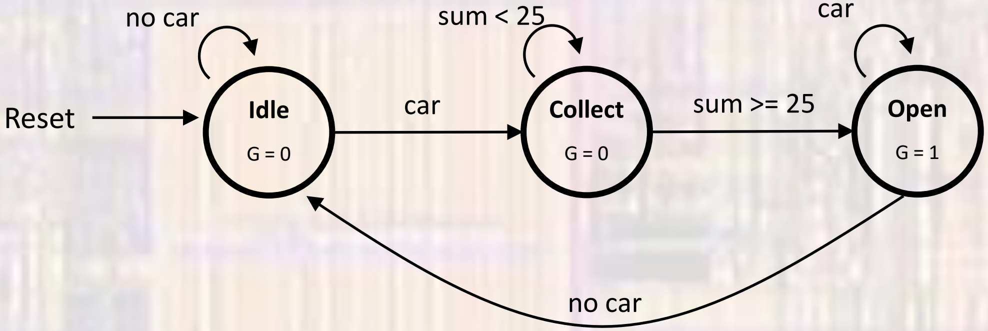
HDL FSM Timers/Counters

- Toll booth - revisited



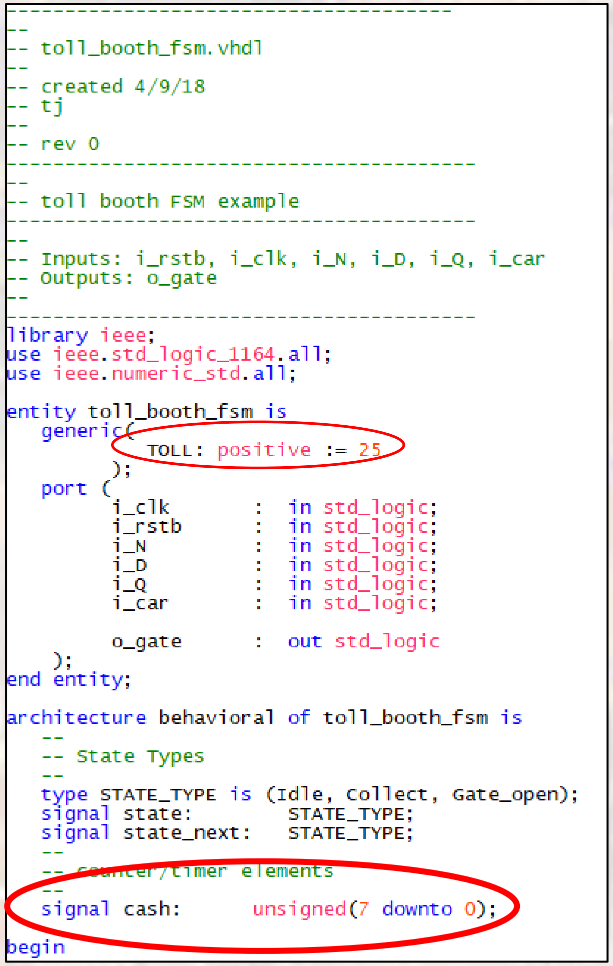
HDL FSM Timers/Counters

- Toll booth - revisited

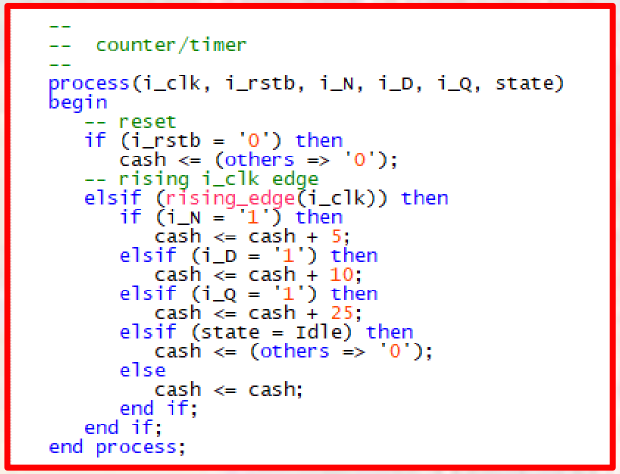


HDL FSM Timers/Counters

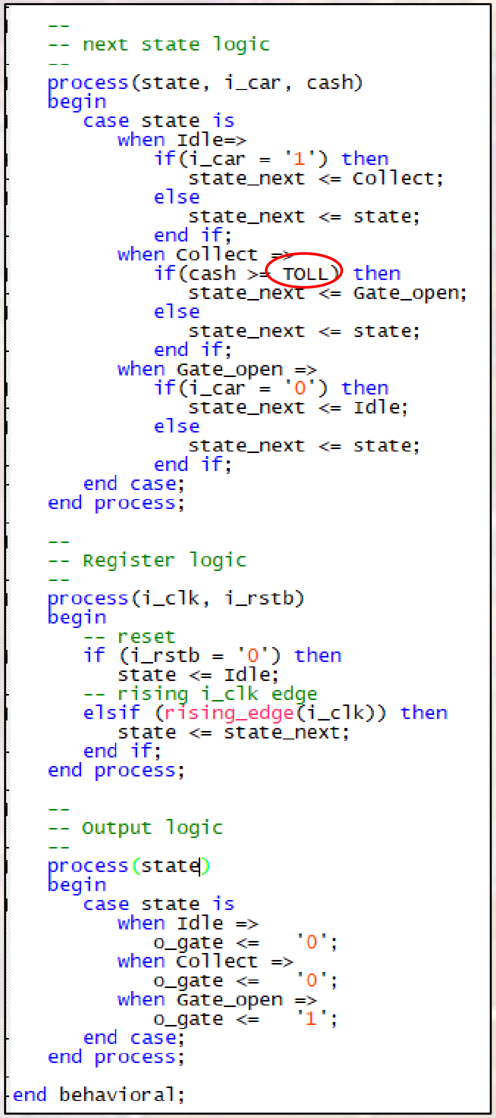
- Toll booth - revisited



```
-------------------------------------
--
-- toll_booth_fsm.vhdl
--
-- created 4/9/18
-- tj
--
-- rev 0
----------------------------------------
--
-- toll booth FSM example
----------------------------------------
--
-- Inputs: i_rstb, i_clk, i_N, i_D, i_Q, i_car
-- Outputs: o_gate
--
----------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity toll_booth_fsm is
   generic(
            TOLL: positive := 25
          );
   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 toll_booth_fsm is
   --
   -- State Types
   --
   type STATE_TYPE is (Idle, Collect, Gate_open);
   signal state:         STATE_TYPE;
   signal state_next:    STATE_TYPE;
   --
   -- counter/timer elements
   --
   signal cash:      unsigned(7 downto 0);

begin
```



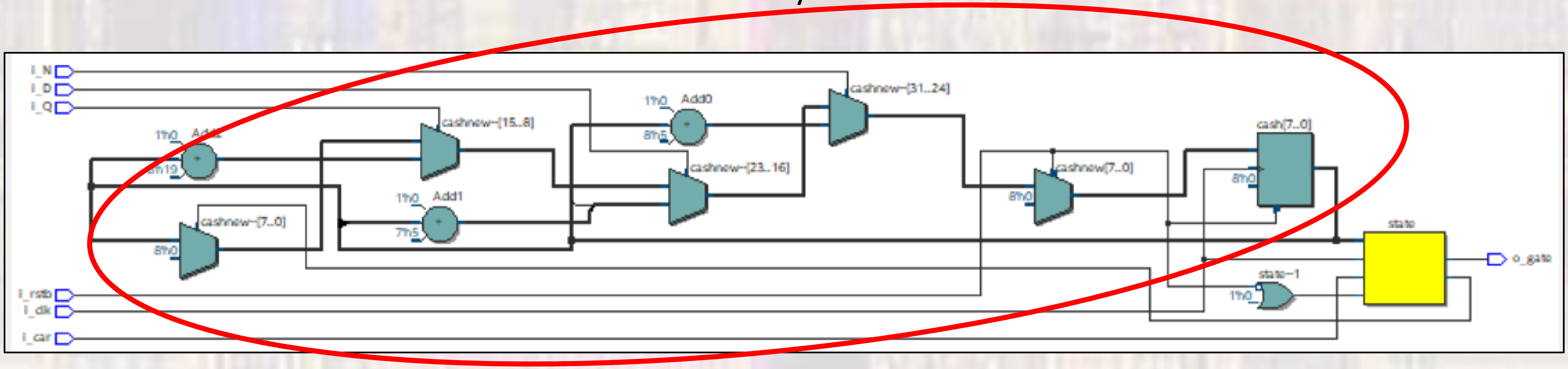
```
--
--  counter/timer
--
process(i_clk, i_rstb, i_N, i_D, i_Q, state)
begin
   -- reset
   if (i_rstb = '0') then
      cash <= (others => '0');
   -- rising i_clk edge
   elsif (rising_edge(i_clk)) then
      if (i_N = '1') then
         cash <= cash + 5;
      elsif (i_D = '1') then
         cash <= 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(state, i_car, cash)
   begin
      case state is
         when Idle=>
            if(i_car = '1') then
               state_next <= Collect;
            else
               state_next <= state;
            end if;
         when Collect =>
            if(cash >= TOLL) 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;
      end case;
   end process;

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

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

end behavioral;
```

HDL FSM Timers/Counters

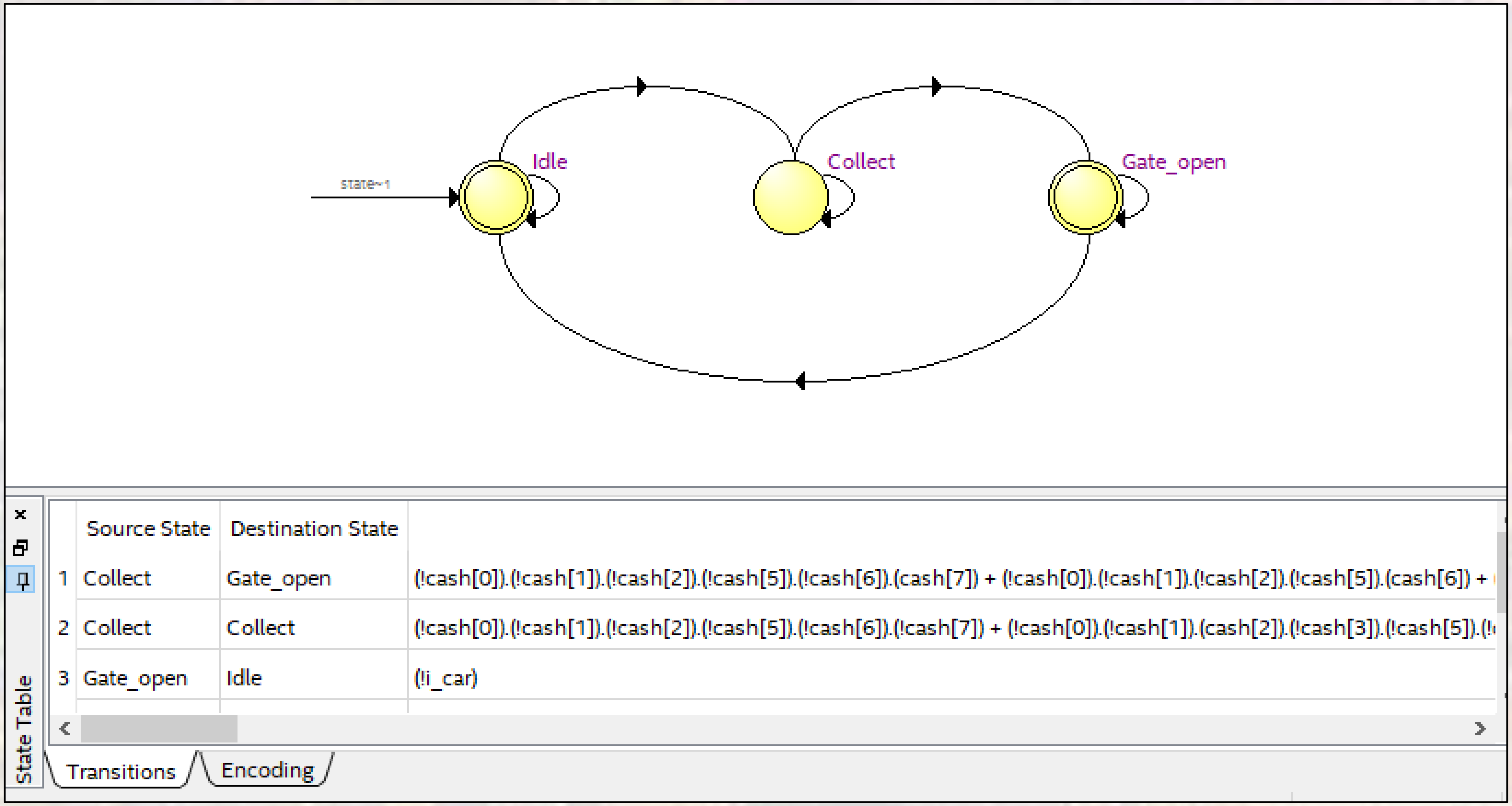
- Toll booth - revisited



counter/timer

HDL FSM Timers/Counters

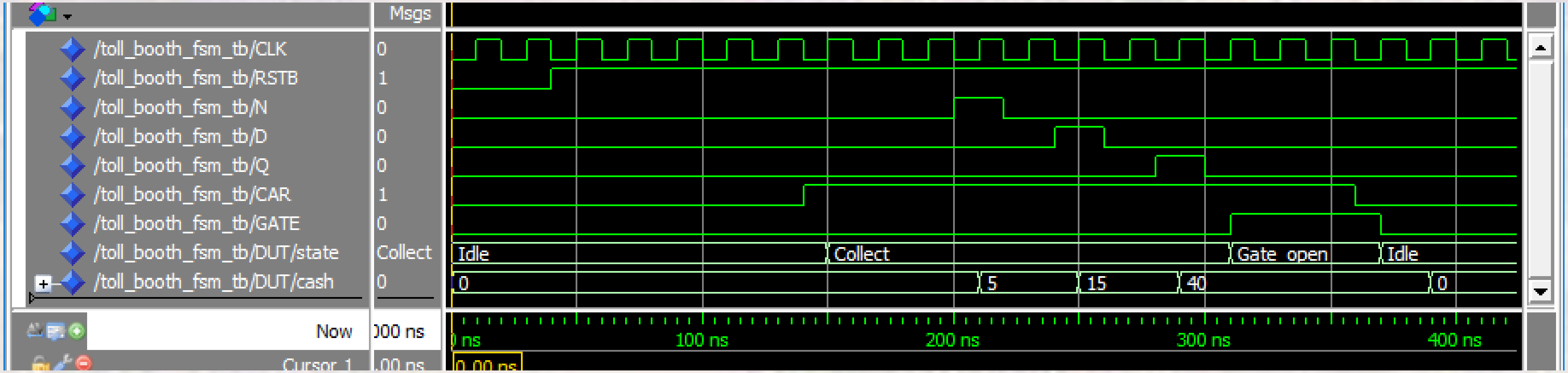
- Toll booth - revisited



	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]) + (
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]).(!
3	Gate_open	Idle	(!i_car)

HDL FSM Timers/Counters

- Toll Booth - revisited



	Msgs
/toll_booth_fsm_tb/CLK	0
/toll_booth_fsm_tb/RSTB	1
/toll_booth_fsm_tb/N	0
/toll_booth_fsm_tb/D	0
/toll_booth_fsm_tb/Q	0
/toll_booth_fsm_tb/CAR	1
/toll_booth_fsm_tb/GATE	0
/toll_booth_fsm_tb/DUT/state	Collect
/toll_booth_fsm_tb/DUT/cash	0
Now	)00 ns
Cursor 1	.00 ns

N D Q Gate stays open
until car exits