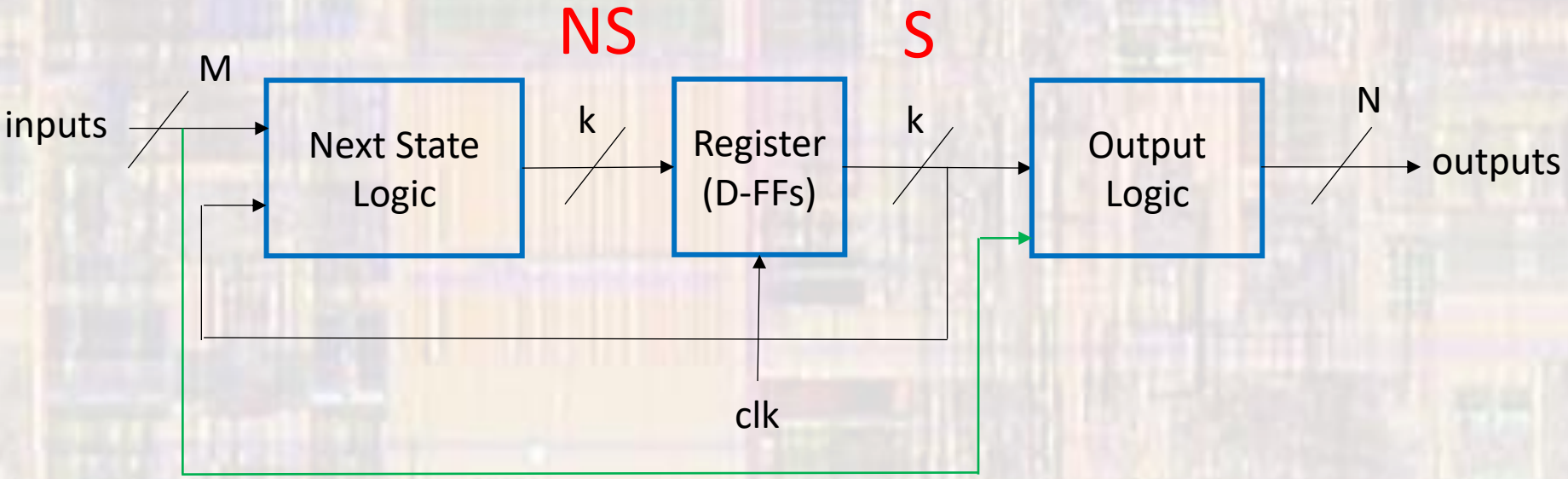
Advanced VHDL FSMs

Last updated 4/23/26

Advanced VHDL FSMs

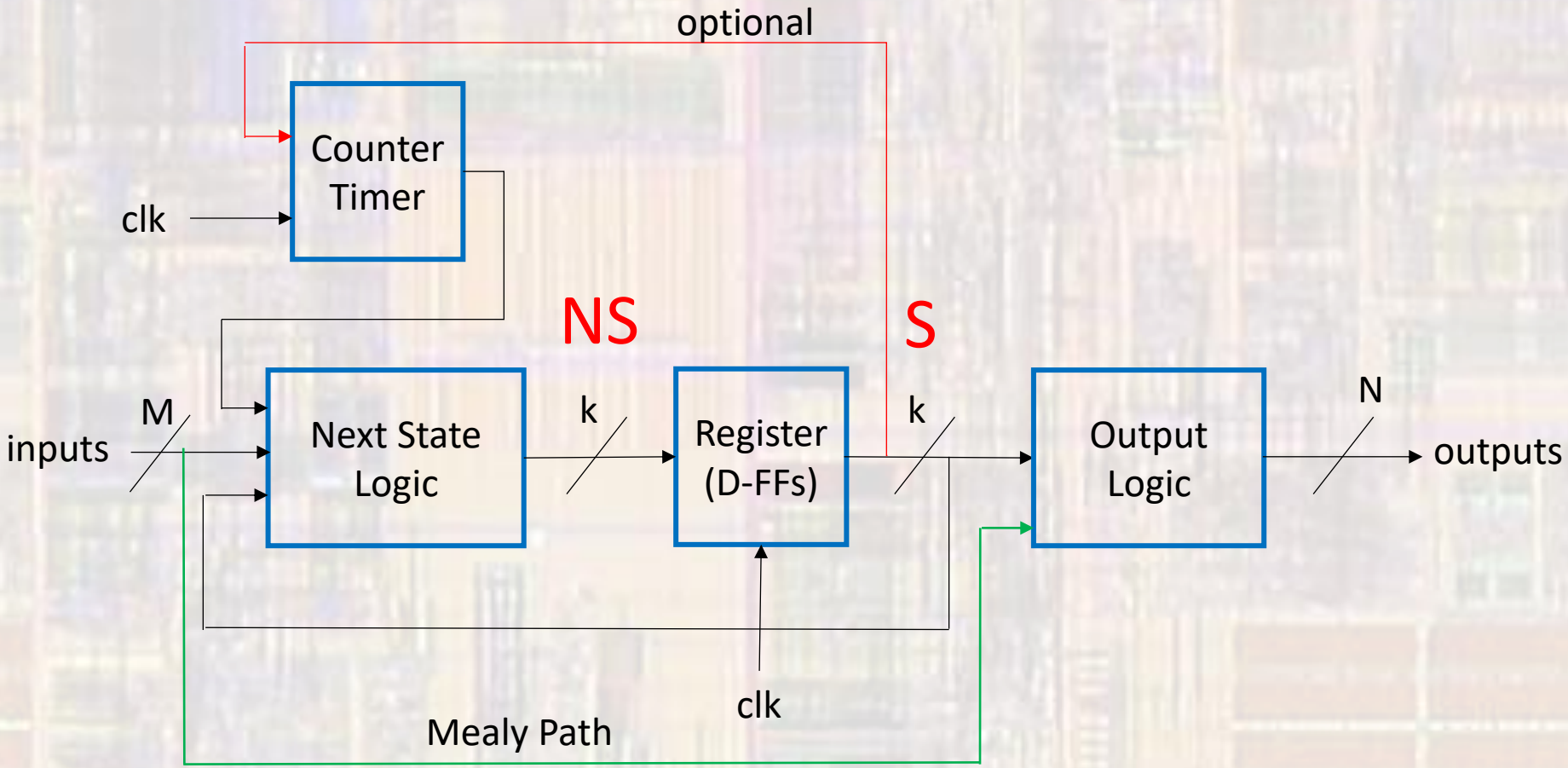
- Finite State Machine
  - Moore Machine
    - Outputs depend only on the current state(S)



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

Advanced VHDL FSMs

- Finite State Machine with Free Running Counter/Timer



Advanced VHDL FSMs

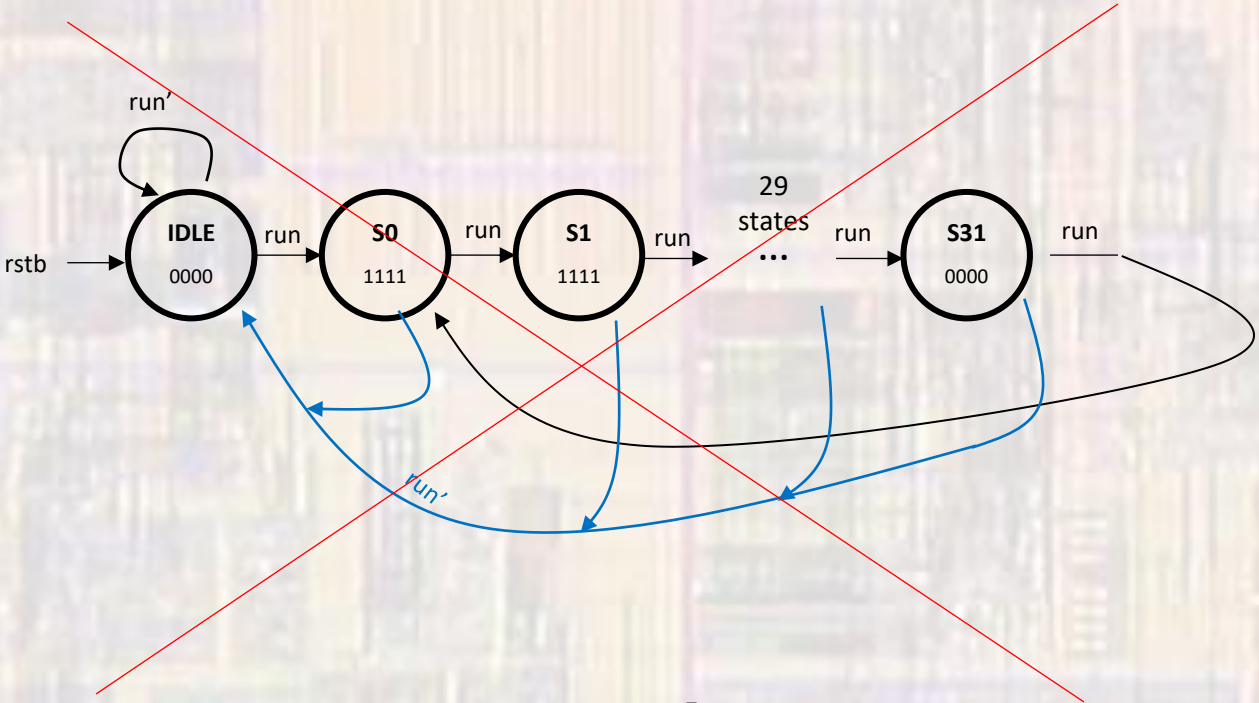
- Sequence Generator
  - Run each of 4 output values for different amounts of time
  - 1111 for 6 clks, 0110 for 7 clks, 1001 for 11 clks, 0000 for 8 clks, restart → total of 32 clks
  - run input to start/stop the sequence

  - inputs: clk, rstb, run
  - outputs: sequence signal

  - States: IDLE, S0, S1, … S31

Advanced VHDL FSMs

- Sequence Generator
  - 1111 for 6 clks, 0110 for 7 clks, 1001 for 11 clks, 0000 for 8 clks, restart → total of 32 clks
  - run input to start/stop the sequence



Advanced VHDL FSMs

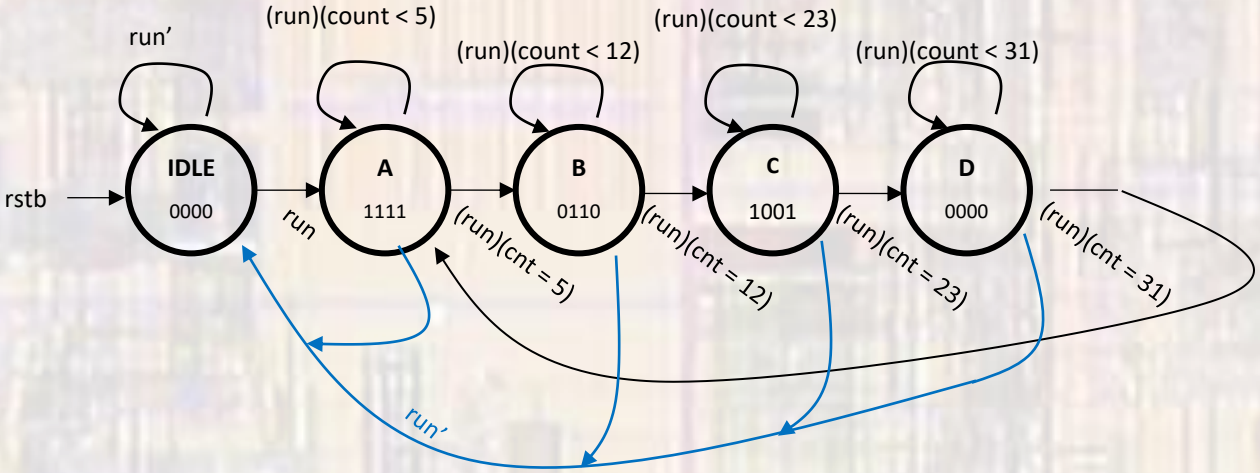
- Sequence Generator – timed
  - Run each of 4 output values for different amounts of time
  - 1111 for 6 clks, 0110 for 7 clks, 1001 for 11 clks, 0000 for 8 clks, restart → total of 32 clks
  - run input to start/stop the sequence

  - inputs: clk, rstb, run
  - outputs: sequence signal

  - States: IDLE, A, B, C, D

Advanced VHDL FSMs

- Sequence Generator – timed
  - 1111 for 6 clks, 0110 for 7 clks, 1001 for 11 clks, 0000 for 8 clks, restart → total of 32 clks
  - run input to start/stop the sequence



Advanced VHDL FSMs

- Sequence Generator – timed

```
----------------------------------------
--
-- fsm_pattern_gen_free_run_clk.vhdl
--
-- created 4/21/26
-- tj
--
-- rev 0
----------------------------------------
--
-- Example of usng a free running clock in an FSM
--    generates a timed bit val for each of 4 states
--
--    1111 for 6 clks, 0110 for 7 clks, 1001 for 11 clks, 0000 for 8 clks, restart
--    total of 32 clks
--
----------------------------------------
--
-- this is for illustrative purposes
--    we would not hardcode the values
--    we would not use a free-running clock in most situations
--
----------------------
--
-- states: IDLE, A, B, C, D
-- Inputs: i_rstb, i_clk, i_run
-- Outputs: o_val
--
----------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity fsm_pattern_gen_free_run_clk is
   port (
         i_clk :     in std_logic;
         i_rstb :    in std_logic;
         i_run :     in std_logic;

         o_val :     out std_logic_vector(3 downto 0)
    );
end entity;
```

```
architecture behavioral of fsm_pattern_gen_free_run_clk is

   --
   -- internal signals
   --
   type STATE_TYPE is (IDLE, A, B, C, D);
   signal state:        STATE_TYPE;
   signal state_next:   STATE_TYPE;

   -- unsigned cnt signal -- to allow addition and comparison
   signal cnt_sig:   unsigned(4 downto 0);

begin

   --
   -- counter logic - free running, auto wrap
   --
   process(i_clk, i_rstb, state)
   begin
      if(i_rstb = '0') then
         cnt_sig <= to_unsigned(0,5);
      elsif (state = IDLE) then
         cnt_sig <= to_unsigned(0,5);
      elsif(rising_edge(i_clk)) then
         cnt_sig <= cnt_sig + 1;
      end if;
   end process;
```

Advanced VHDL FSMs

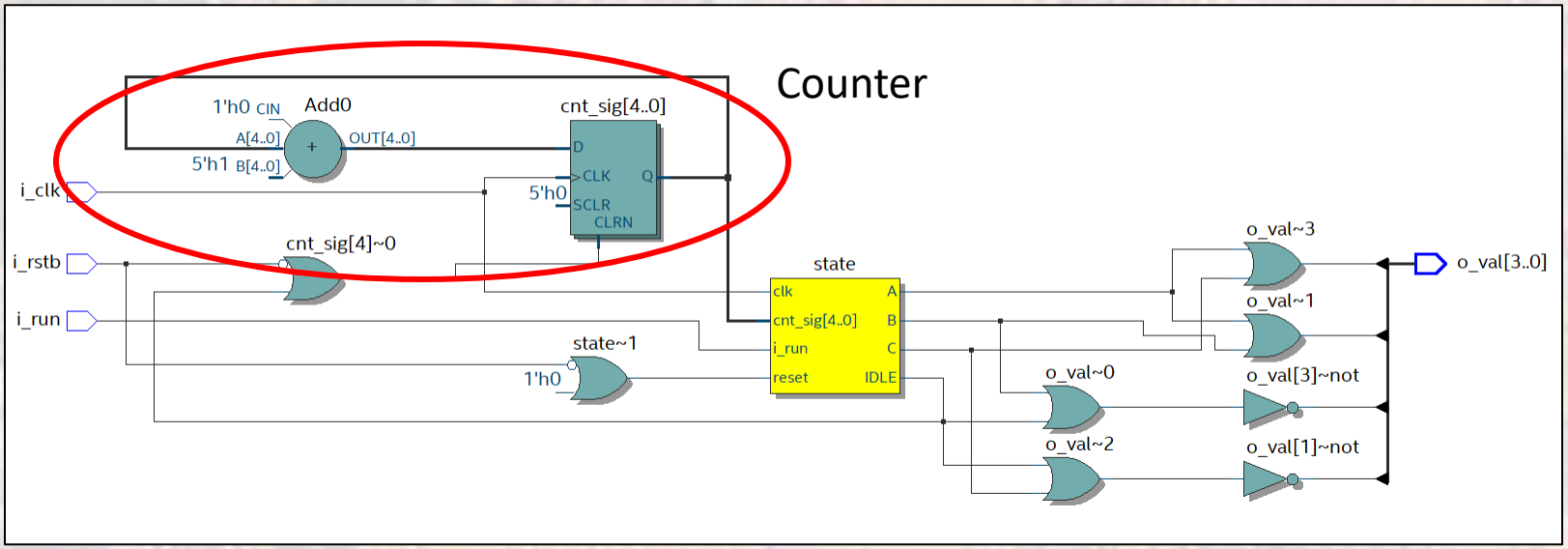
- Sequence Generator – timed

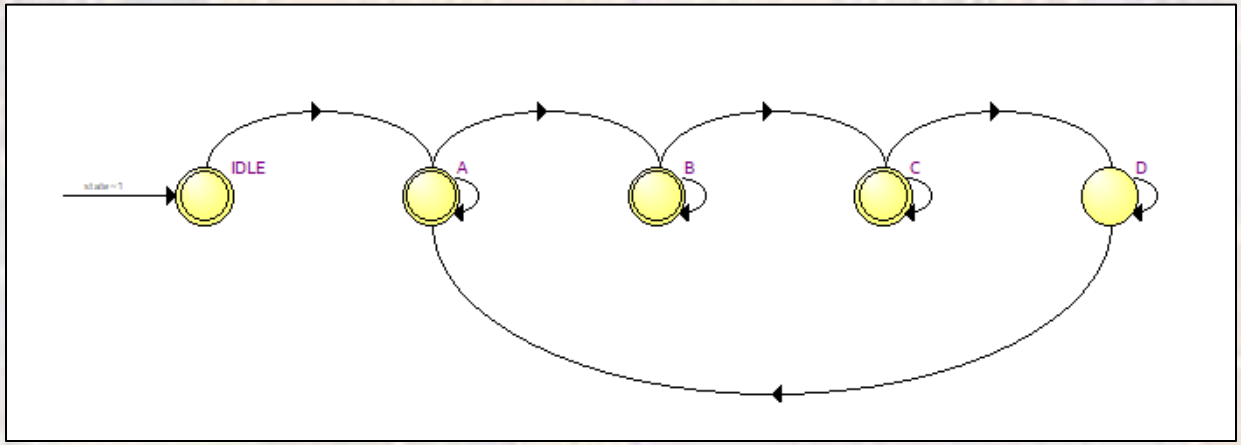
```
--
-- next state logic
--
process(state, i_run, cnt_sig)
begin
   if(i_run = '0') then
      state_next <= IDLE;

   else

      case state is
         when IDLE =>
            state_next <= A;
         when A =>
            if (cnt_sig = 5) then    -- remember 0 counts, 6 clks --> cnt = 5
               state_next <= B;
            else
               state_next <= A;
            end if;
         when B =>
            if (cnt_sig = 12) then
               state_next <= C;
            else
               state_next <= B;
            end if;
         when C =>
            if (cnt_sig = 23) then
               state_next <= D;
            else
               state_next <= C;
            end if;
         when D =>
            if (cnt_sig = 31) then
               state_next <= A;
            else
               state_next <= D;
            end if;
         when others => state_next <= IDLE;
      end case;

   end if;

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_val <= "0000";
         when A      => o_val <= "1111";
         when B      => o_val <= std_logic_vector(to_unsigned(6,4));
         when C      => o_val <= std_logic_vector(to_unsigned(9,4));
         when D      => o_val <= "1010";
         when others => o_val <= (others => '0');
      end case;
   end process;

end behavioral;
```

Advanced VHDL FSMs

- Sequence Generator – timed





Advanced VHDL FSMs

- Sequence Generator – timed

Advanced VHDL FSMs

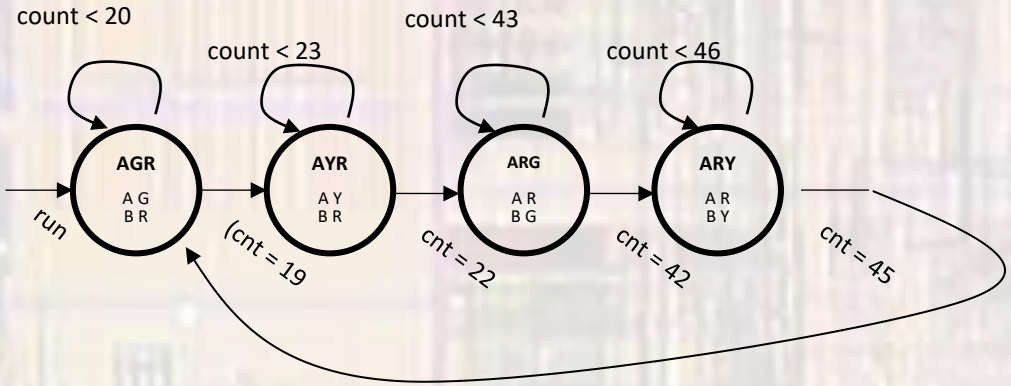
- Stoplight – timed
  - 4 way stoplight – Adir and Bdir
  - A RG for 20 clks, A RY for 3 clks, A GR for 20 clks, A YR for 3 clks, restart → total of 46 clks

  - inputs: clk, rstb
  - outputs: sequence signal

  - States: ARG, ARY, AGR, AYR

Advanced VHDL FSMs

- Stoplight – timed
  - 4 way stoplight – Adir and Bdir
  - A RG for 20 clks, A RY for 3 clks, A GR for 20 clks, A YR for 3 clks, restart → total of 46 clks



Advanced VHDL FSMs

- Stoplight - timed

```
----------------------------------------
--
-- stoplight_timed_free_running.vhdl
--
-- created 3/30/18
-- tj
--
-- rev 0
----------------------------------------
--
-- timed 4 way stoplight
----------------------------------------
--
-- Inputs: i_rstb, i_clk,
-- Outputs: o_lights_Adir, o_lights_Bdir
--
----------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity stoplight_timed_free_running is
   port (
         i_clk :     in std_logic;
         i_rstb : in std_logic;

         o_lights_Adir : out std_logic_vector(1 downto 0);
         o_lights_Bdir : out std_logic_vector(1 downto 0)

    );
end entity;
```

```
architecture behavioral of stoplight_timed_free_running is
   --
   -- internal signals
   --
   type STATE_TYPE is (AGR, AYR, ARG, ARY);
   signal state:        STATE_TYPE;
   signal state_next:   STATE_TYPE;

   signal cnt_sig:      unsigned(5 downto 0);

   --
   -- create constants to allow changing the encoding
   -- in one place
   -- could also use generics
   --
   constant green:   std_logic_vector  := "00";
   constant yellow:  std_logic_vector  := "01";
   constant red:     std_logic_vector  := "10";

begin

   --
   -- counter logic - free running, auto wrap
   -- need count from 0 - 45 - not a power of 2
   -- --> modulo counter
   --
   process(i_clk, i_rstb, state)
   begin
      if(i_rstb = '0') then
         cnt_sig <= to_unsigned(0,6);
      elsif(rising_edge(i_clk)) then
         if(cnt_sig < 45) then
            cnt_sig <= cnt_sig + 1;
         else
            cnt_sig <= to_unsigned(0,6);
         end if;
      end if;
   end process;
```

Advanced VHDL FSMs

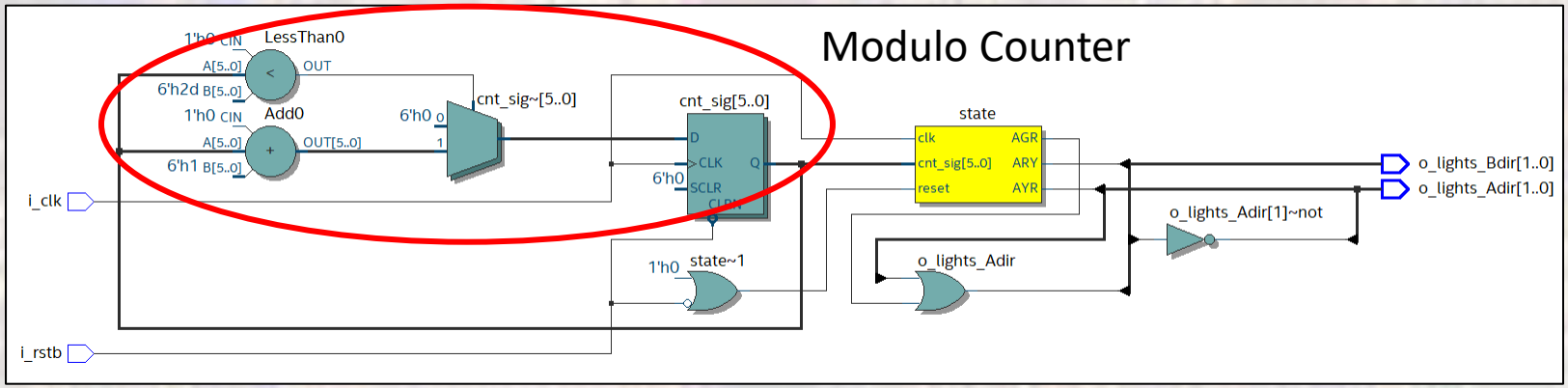
- Stoplight - timed

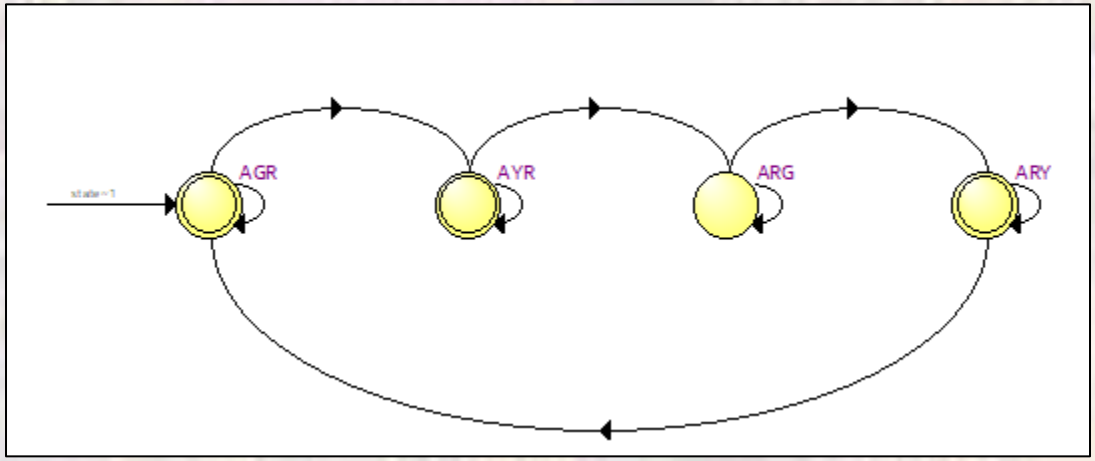
```
--
-- next state logic
--
process(state, cnt_sig)
begin
   case state is
      when AGR =>
         if(cnt_sig = 19) then
            state_next <= AYR;
         else
            state_next <= AGR;
         end if;
      when AYR =>
         if(cnt_sig = 22) then
            state_next <= ARG;
         else
            state_next <= AYR;
         end if;
      when ARG =>
         if(cnt_sig = 42) then
            state_next <= ARY;
         else
            state_next <= ARG;
         end if;
      when ARY =>
         if(cnt_sig = 45) then
            state_next <= AGR;
         else
            state_next <= ARY;
         end if;
      when others =>
         -- cant just jump to an unexpected state
         -- freeze the current state until resolved
         state_next <= state;
   end case;
end process;
```

```
   --
   -- Register logic
   --
   process(i_clk, i_rstb)
   begin
      -- reset
      if (i_rstb = '0') then
         state <= AGR;
      -- 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 AGR =>
            o_lights_Adir <= green;
            o_lights_Bdir <= red;
         when AYR =>
            o_lights_Adir <= yellow;
            o_lights_Bdir <= red;
         when ARG =>
            o_lights_Adir <= red;
            o_lights_Bdir <= green;
         when ARY =>
            o_lights_Adir <= red;
            o_lights_Bdir <= yellow;
         when others =>
            -- something went wrong - force red-red
            o_lights_Adir <= red;
            o_lights_Bdir <= red;
      end case;
   end process;

end behavioral;
```

Advanced VHDL FSMs

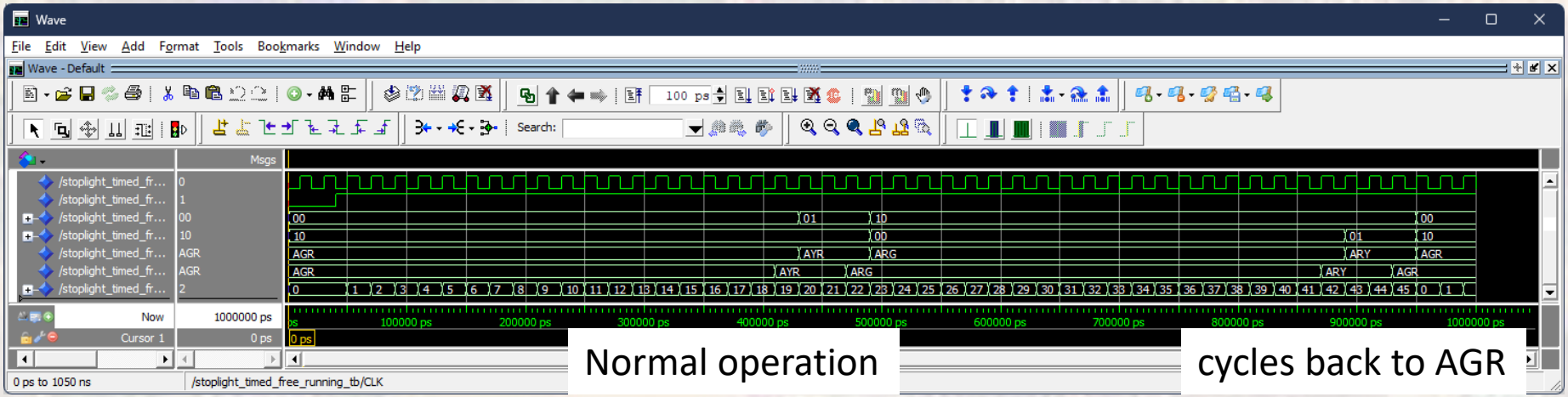
- Stoplight - timed





Advanced VHDL FSMs

- Stoplight - timed



Advanced VHDL FSMs

- Stoplight – timed
  - Improvements
    - Add a state for flashing red in both directions if something fails
    - Use generics or constants for the times
      - Can we calculate the modulo???
    - Maintenance, emergency, short RR after YR and RY

Advanced VHDL FSMs

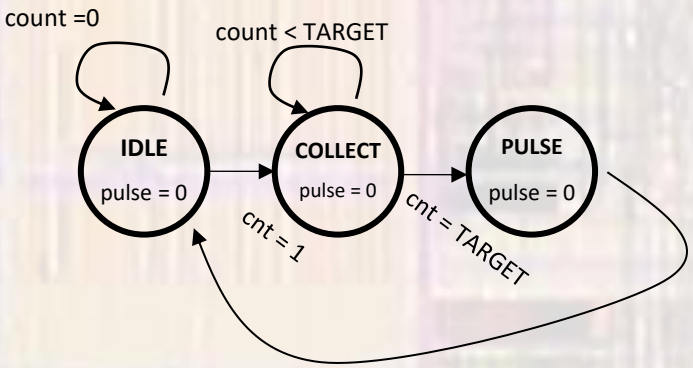
- Accumulator – counting 1's
  - Generate a 1 clk wide pulse for every N^{th} 1 entered

  - generic: target count
  - inputs: clk, rstb, din
  - outputs: pulse

  - States: IDLE, COLLECT, PULSE

Advanced VHDL FSMs

- Accumulator – counting 1's
  - Generate a 1 clk wide pulse for every N^{th} 1 entered



Advanced VHDL FSMs

- Accumulator – counting 1's

```
----------------------------------------
--
-- ones_counter.vhdl
--
-- created 3/30/18
-- tj
--
-- rev 0
----------------------------------------
--
-- Counts a defined number of 1's, then outputs a pulse
----------------------------------------
--
-- Inputs: i_rstb, i_clk, i_din
-- Outputs: o_pulse
--
----------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.math_real.all;

entity ones_counter is
   generic(
            CNT_TARGET: positive     := 5
          );
   port (
         i_clk:   in std_logic;
         i_rstb:  in std_logic;
         i_din:   in std_logic;

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

```
architecture behavioral of ones_counter is
   --
   -- internal signals
   --
   type STATE_TYPE is (IDLE, COLLECT, PULSE);
   signal state:        STATE_TYPE;
   signal state_next:   STATE_TYPE;

   constant CNT_SIZE:   positive     := integer(ceil(log2(real(CNT_TARGET))));
   signal cnt_sig:      unsigned((CNT_SIZE - 1) downto 0);


begin

   --
   -- --> modulo CNT_TARGET
   --
   process(i_clk, i_rstb, i_din)
   begin
      if(i_rstb = '0') then
         cnt_sig <= to_unsigned(0,CNT_SIZE);
      elsif(rising_edge(i_clk)) then
         if(cnt_sig < CNT_TARGET) then
            if(i_din = '1') then
               cnt_sig <= cnt_sig + 1;
            else
               cnt_sig <= cnt_sig;
            end if;
         else
            cnt_sig <= to_unsigned(0,CNT_SIZE);
         end if;
      end if;
   end process;
```

Advanced VHDL FSMs

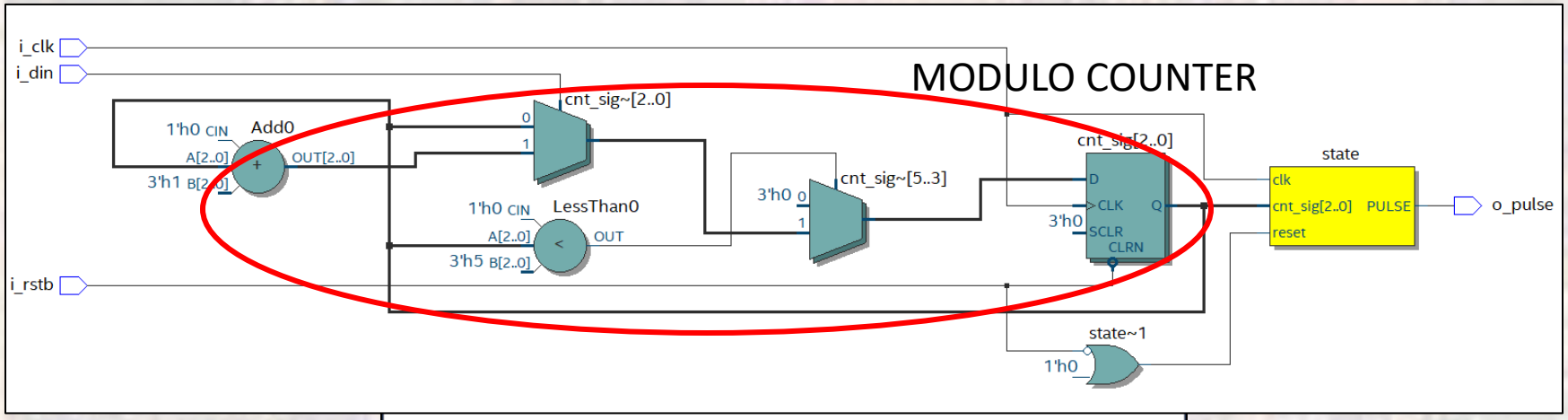
- Accumulator – counting 1's

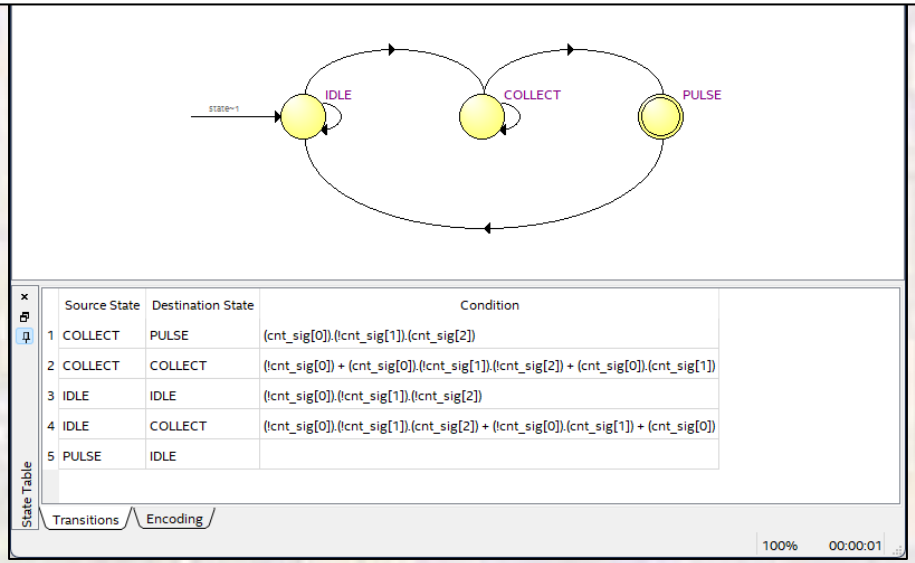
```
--
-- next state logic
--
process(state, cnt_sig)
begin
   case state is
      when IDLE =>
         if(cnt_sig = 0) then
            state_next <= IDLE;
         else
            state_next <= COLLECT;
         end if;
      when COLLECT =>
         if(cnt_sig = CNT_TARGET) then
            state_next <= PULSE;
         else
            state_next <= COLLECT;
         end if;
      when PULSE =>
         state_next <= IDLE;
      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 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_pulse <= '0';
      when COLLECT =>
         o_pulse <= '0';
      when PULSE =>
         o_pulse <= '1';
      when others =>
         o_pulse <= '0';
      end case;
end process;
```

Advanced VHDL FSMs

- Accumulator – counting 1's

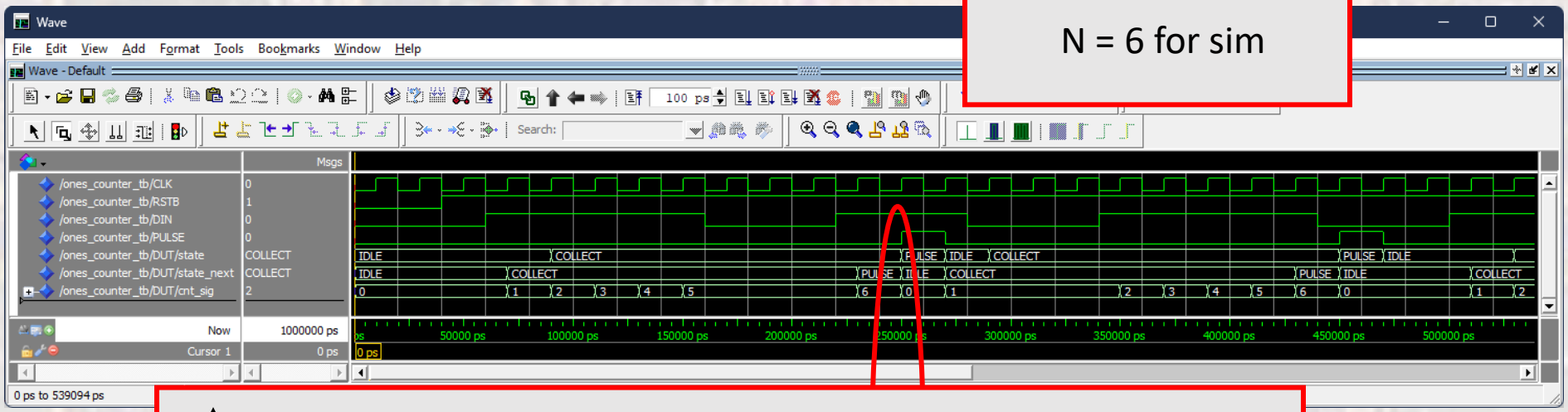




Advanced VHDL FSMs

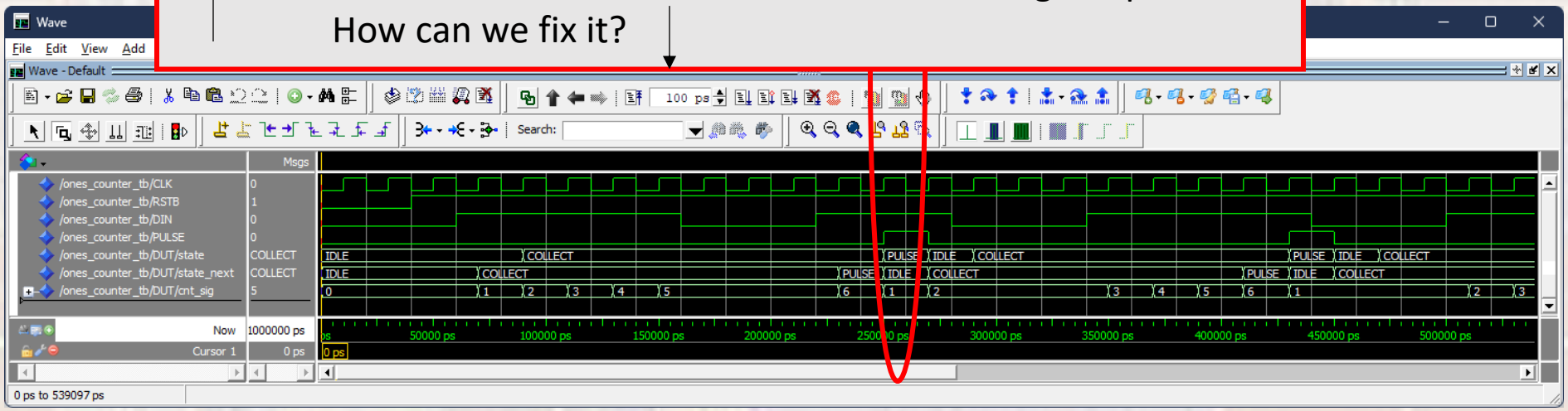
- Accumulator – counting 1's

N = 6 for sim



This solution does not count the 1 entered during the pulse

How can we fix it?

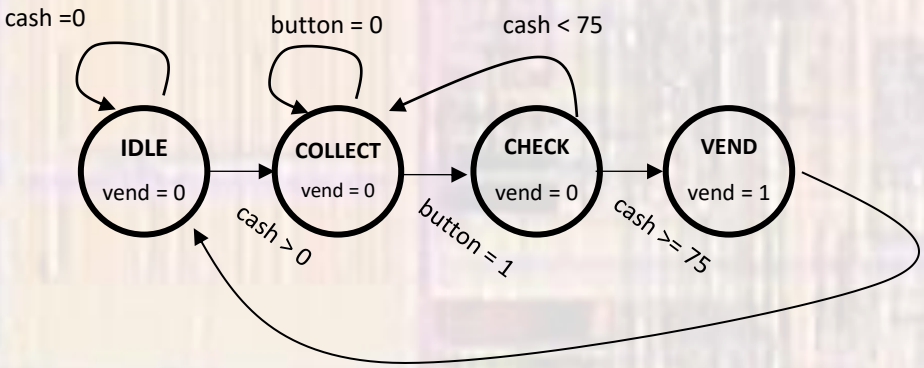


Advanced VHDL FSMs

- Accumulator – vending machine
  - Collect coins and vend when button pushed
  - inputs: clk, rstb, coin (5, 10, 25)
  - outputs: vend
  - States: IDLE, COLLECT, CHECK, VEND

Advanced VHDL FSMs

- Accumulator – vending machine
  - Collect coins and vend when button pushed



Advanced VHDL FSMs

- Accumulator – vending machine

```
----------------
--
-- vend_1_item.vhdl
--
-- created 4/25/25
-- tj
--
----------------------
--
-- 1 item vending machine - check for cash input
-- nickles, dimes, quarters
--    01      10      11
--
--------------------
--
-- inputs coin, button, clk, rstb
-- outputs item
--
----------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


entity vend_1_item is
   port(
         i_clk:      in std_logic;
         i_rstb:     in std_logic;
         i_coin:     in std_logic_vector(1 downto 0);
         i_button:   in std_logic;

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

```
architecture behavioral of vend_1_item is

   type STATE_TYPE is (IDLE, COLLECT, CHECK, VEND);
   signal state:        STATE_TYPE;
   signal state_next:   STATE_TYPE;

   -- support for $1 total, although cost is $0.75
   signal cash_sig:  unsigned(6 downto 0);

begin

   --
   -- cash counter
   process(i_clk, i_rstb, i_coin, state)
   begin
      if(i_rstb = '0') then
         cash_sig <= to_unsigned(0,7);
      elsif(rising_edge(i_clk)) then
         if (state = VEND) then
            cash_sig <= to_unsigned(0, 7);
         elsif(i_coin = "01") then
            cash_sig <= cash_sig + 5;
         elsif(i_coin = "10") then
            cash_sig <= cash_sig + 10;
         elsif(i_coin = "11") then
            cash_sig <= cash_sig + 25;
         else
            cash_sig <= cash_sig;
         end if;
      end if;
   end process;
```

Advanced VHDL FSMs

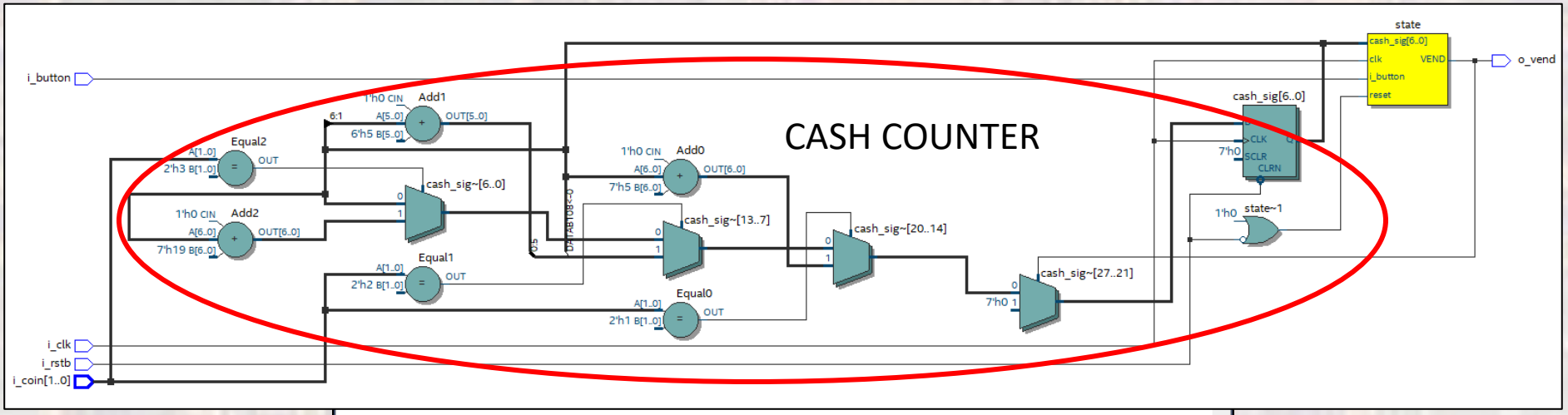
- Accumulator – vending machine

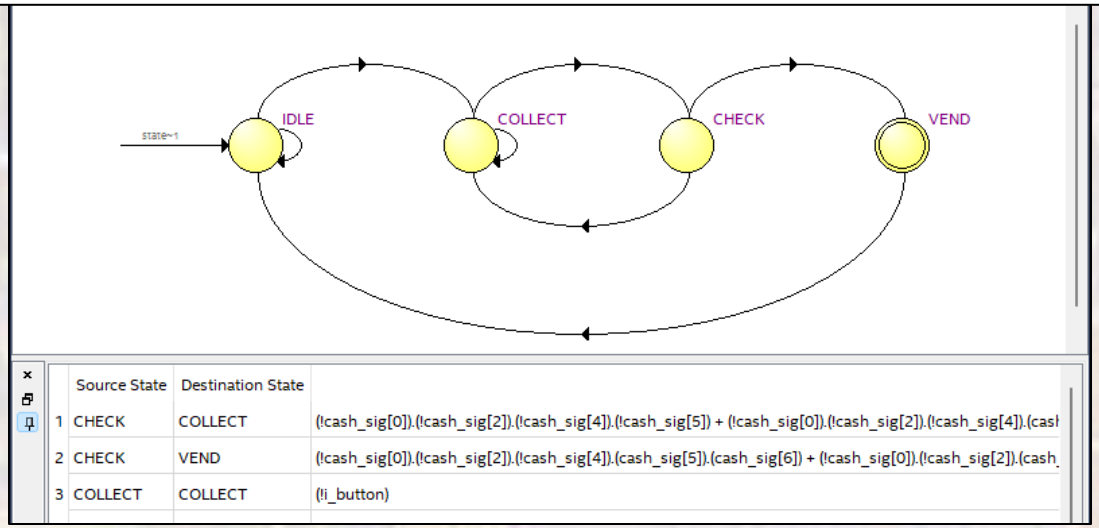
```
-- next state logic
process(state, cash_sig, i_button)
begin
   case state is
      when idle =>
         if(cash_sig = 0) then
            state_next <= IDLE;
         else
            state_next <= COLLECT;
         end if;
      when collect =>
         if(i_button = '0') then
            state_next <= COLLECT;
         else
            state_next <= CHECK;
         end if;
      when CHECK =>
         if(cash_sig >= to_unsigned(75, 7)) then
            state_next <= VEND;
         else
            state_next <= COLLECT;
         end if;
      when vend =>
         state_next <= idle;
      when others =>
         state_next <= idle;
   end case;
end process;
```

```
   -- register logic
   process(i_clk, i_rstb)
   begin
      -- reset mode
      if(i_rstb = '0') then
         state <= IDLE;
      elsif(rising_edge(i_clk)) then
         state <= state_next;
      end if;
   end process;

   -- output logic
   process(state)
   begin
      case state is
         when IDLE =>
            o_vend <= '0';
         when COLLECT =>
            o_vend <= '0';
         when CHECK =>
            o_vend <= '0';
         when VEND =>
            o_vend <= '1';
         when others =>
            o_vend <= '0';
      end case;
   end process;

end architecture;
```

Advanced VHDL FSMs

- Accumulator – vending machine





	Source State	Destination State	
1	CHECK	COLLECT	(!cash_sig[0].(!cash_sig[2]).(!cash_sig[4]).(!cash_sig[5]) + (!cash_sig[0]).(!cash_sig[2]).(!cash_sig[4]).(cash
2	CHECK	VEND	(!cash_sig[0]).(!cash_sig[2]).(!cash_sig[4]).(cash_sig[5]).(cash_sig[6]) + (!cash_sig[0]).(!cash_sig[2]).(cash_
3	COLLECT	COLLECT	(!i_button)

Advanced VHDL FSMs

- Accumulator – vending machine