ECE 5723 Homework 1

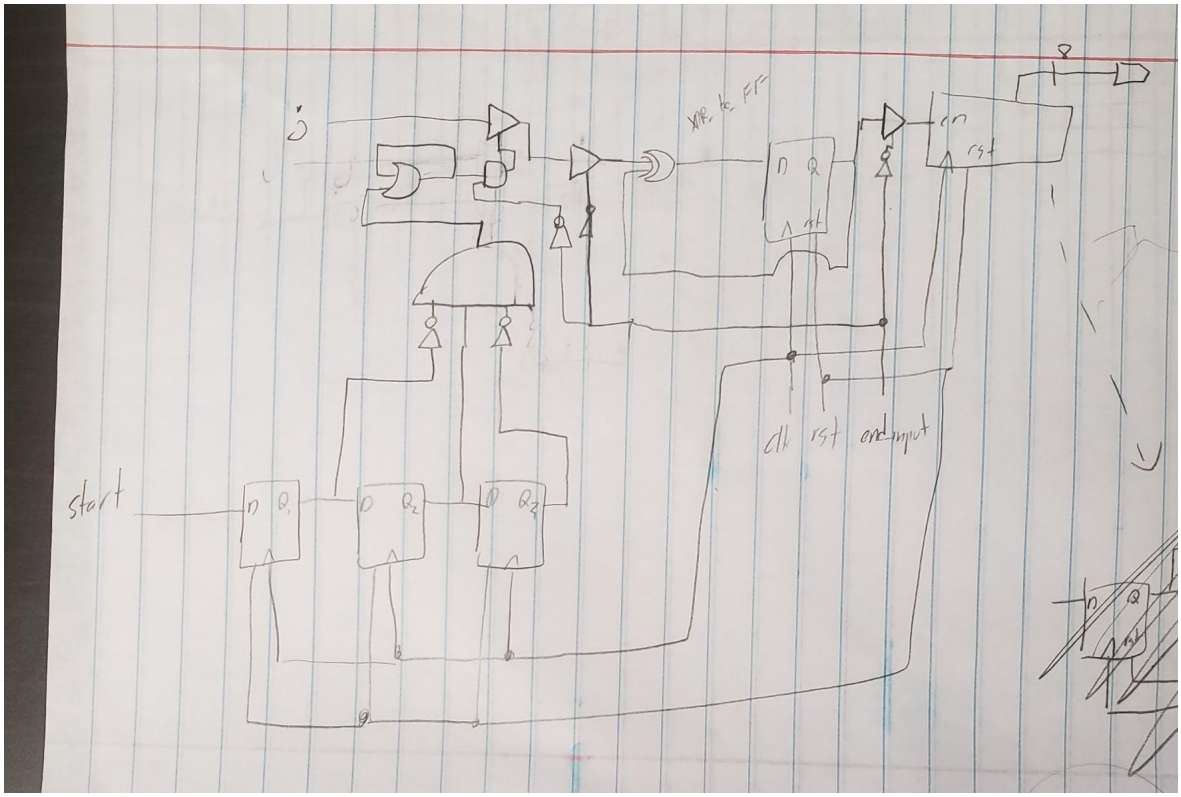
By Bruce Huynh

9/12/2021

Introduction:

In this assignment, we were tasked with building a transition counter. This counter would have a start port that when signaled with “010”, will start counting the transitions on the J port. If a signal would be detected on the end port, the counter will stop counting transitions until it receives another “010” pulse from the start port.

A.) Datapath

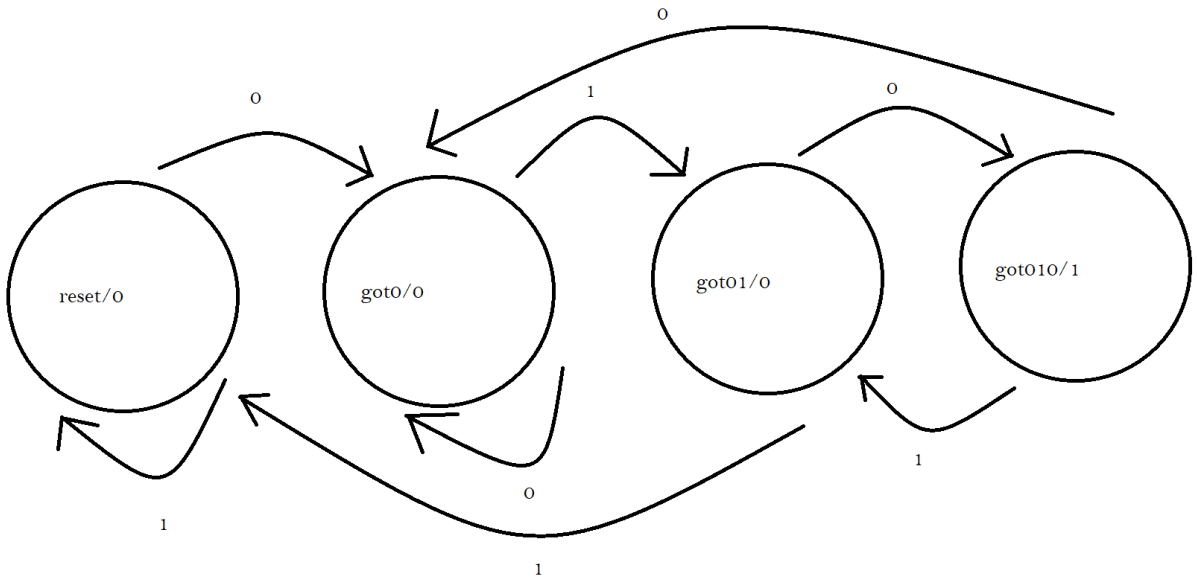


The start input goes into a shift register made of three D flip flops. Using the output data from the flip flops, an AND gate combined with NOT gates detect a “010” signal on the start port. This data gets fed into an OR gate that keeps the start signal high until it receives an input from the end input. The start signal enables the tri-state buffer and allows J to pass through. The XOR gate combined with the flip flop allows for the detection of signal transitions in J. If the end input is not detected, this signal gets moved into the counter where it will count the transitions up to

256 in which it rolls over to 0. If end input is detected at all, counting ceases we will have to wait for another start input for the circuit to reactivate.

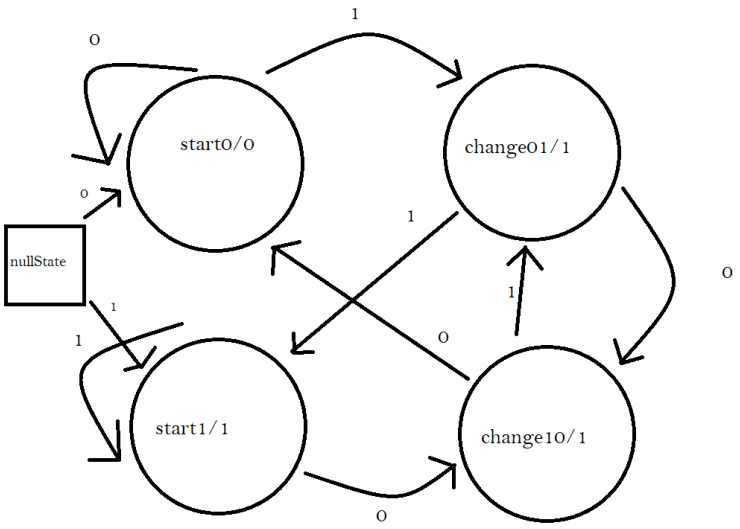
B.) State diagram

The following diagram is the state diagram for the start input:



There are four possible states for the start input and it only will start the circuit once “010” is received as seen above. The same state diagram is used for the end input.

The next diagram is the state diagram to detect transitions:



When not counting, the counter starts at a null state. When it first detects a transition, it will take the first value it receives and store it. It then compares the next signal and will add one to its current count if there is a transition.

C.) Controller Implementation:

NOTE: The shift register that controls the start input is in the datapath top module but I will list that segment here:

```
-- shift register(
FF2 : d_FF port map(D => start_input, clk => clk, rst => rst, Q =>
two_to_one);
FF1 : d_FF port map(D => two_to_one, clk => clk, rst => rst, Q =>
one_to_zero);
FF0 : d_FF port map(D => one_to_zero, clk => clk, rst => rst, Q =>
zero_out);
```

```
-- )

-- getting "010" from the registers (
is_it_starting <= (not two_to_one) AND one_to_zero AND (not zero_out)
AND not_end_input;
-- )
```

Counter Controller:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;
entity counter8bits is
port(counter_in, end_input, started, rst, clk : IN std_logic;
counter_out : OUT std_logic_vector(7 downto 0));
end entity counter8bits;

architecture counting_arc of counter8bits is
signal current_count : std_logic_vector(7 downto 0);
type jstate is (nullState, start0, start1, change01, change10);
signal present_state : jstate;
signal next_state: jstate := nullState;
signal end1 : std_logic;
signal end2 : std_logic := '0';



signal int_count: integer := 0;
begin

present_state <= next_state when clk'event and clk = '1';

current_count <= std_logic_vector(to_unsigned(int_count, 8)) when
rising_edge(clk);
counter_out <= current_count;

ending: process(clk)
```

```
begin
if started = '1' then
end2 <= '1';
elsif end_input = '1' then
end2 <= '0';
end if;
end process ending;




stacking: process(clk)
begin

next_state <= present_state;
int_count <= 0 when int_count = 255;
int_count <= 0 when rst = '1';
                     if end2 = '1' then
                          case present_state is

                               when nullState =>
                                    if (counter_in = '0') then
                                         next_state <= start0;
                                    else
                                         next_state <= start1;
                                    end if;

                               when start0 =>
                                    if (counter_in = '1') then
                                         next_state <= change01;
                                         int_count <= int_count +
1;
                                    else
                                         next_state <= start0;
                                    end if;

                               when start1 =>
                                    if (counter_in = '0') then
                                         next_state <= change10;
```

```
                                        int_count <= int_count +
1;
                                  else
                                        next_state <= start1;
                                  end if;

                            when change01 =>
                                  if (counter_in = '0') then
                                        next_state <= change10;
                                        int_count <= int_count +
1;
                                  else
                                        next_state <= start1;
                                  end if;

                            when change10 =>
                                  if (counter_in = '1') then
                                        next_state <= change01;
                                        int_count <= int_count +
1;
                                  else
                                        next_state <= start0;
                                  end if;
                            when others =>
                                  next_state <= nullState;


end case;
end if;
end process stacking;
end architecture counting_arc;
```

D/E.) Data path code and wiring code

For this assignment, the datapath and the top module were made together. Here is the code as follows:

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity trans_top is
port(j, start_input, end_input, clk, rst : IN std_logic; count : OUT
std_logic_vector(7 downto 0));
end entity trans_top;

architecture top_level of trans_top is

component d_FF port(D, clk, rst : IN std_logic; Q : OUT std_logic);
end component;
component tri_state_buffer port(inp, enable : IN std_logic; outp :
OUT std_logic);
end component;
component counter8bits port (counter_in, end_input, started, rst, clk
: IN std_logic; counter_out : OUT std_logic_vector(7 downto 0));
end component;

signal two_to_one : std_logic;
signal one_to_zero : std_logic;
signal zero_out : std_logic;
signal is_it_starting : std_logic;
signal j_tri_enable : std_logic;
signal j_buffer_output : std_logic;
signal not_end_input : std_logic;
signal NEI_to_XOR : std_logic;
signal transition_FF_out : std_logic;
signal XOR_to_FF : std_logic ;
signal j_to_counter :std_logic;
signal counter_out_to_top :std_logic_vector(7 downto 0);
begin


-- end input(
not_end_input <= (not end_input);
-- )

--data path(
```

```
-- shift register(
FF2 : d_FF port map(D => start_input, clk => clk, rst => rst, Q =>
two_to_one);
FF1 : d_FF port map(D => two_to_one, clk => clk, rst => rst, Q =>
one_to_zero);
FF0 : d_FF port map(D => one_to_zero, clk => clk, rst => rst, Q =>
zero_out);
-- )

-- getting "010" from the registers (
is_it_starting <= (not two_to_one) AND one_to_zero AND (not zero_out)
AND not_end_input;
-- )

-- monostable (
j_tri_enable <= is_it_starting OR j_tri_enable;
-- )

-- j input (
j_TS : tri_state_buffer port map(inp => j, enable => j_tri_enable,
outp => j_buffer_output);

-- )

-- end input control (
end_input_TS : tri_state_buffer port map(inp => j_buffer_output,
enable => not_end_input, outp => NEI_to_XOR);
-- )

-- transition FF (
transition_FF : d_FF port map(D => XOR_to_FF, clk => clk, rst => rst,
Q => transition_FF_out);
-- )

-- XOR (

XOR_to_FF <= NEI_to_XOR XOR transition_FF_out;
-- )
```

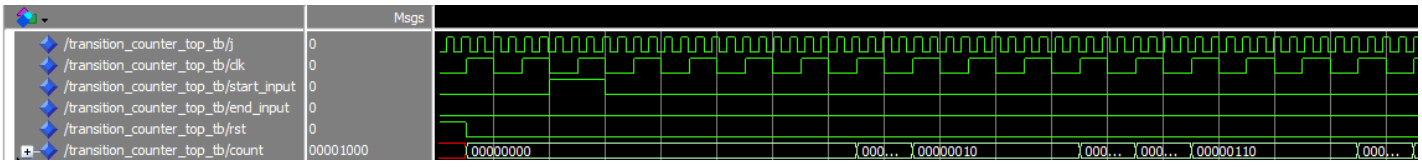
```
-- end input control for counter (
end_input_TS2 : tri_state_buffer port map (inp => transition_FF_out,
enable => not_end_input, outp => j_to_counter);
-- )

-- counter (
my_counter : counter8bits port map(counter_in => j_to_counter,
end_input => end_input, started => is_it_starting, rst => rst, clk =>
clk, counter_out => counter_out_to_top);
-- )

count <= counter_out_to_top;



end architecture top_level;
```

NOTE: The components called were made by me and can be found in the zipped folder of files.

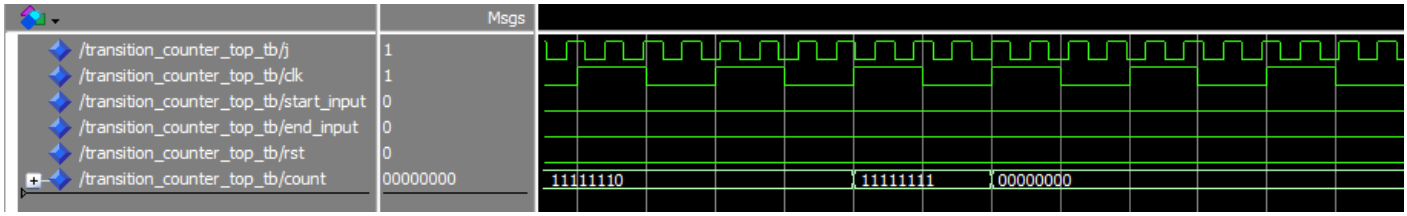
F.) Testbenches

In this section, I created four different testbenches to view the different effects of the circuit. Here are the tests and their results.

Test 1 - No end input and J alternates every 14 ns:

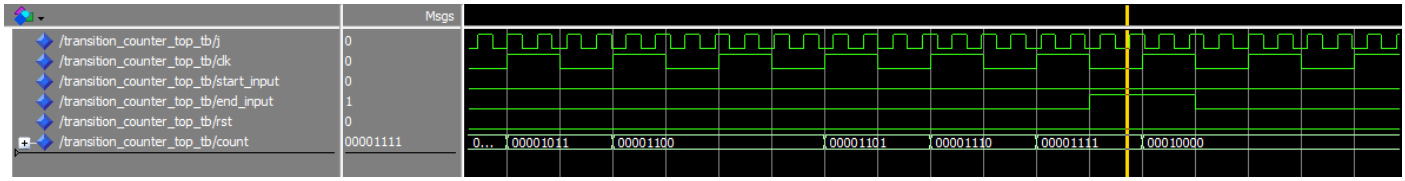


As seen above, the circuit waits for a start pulse before it can begin counting the transitions in J.



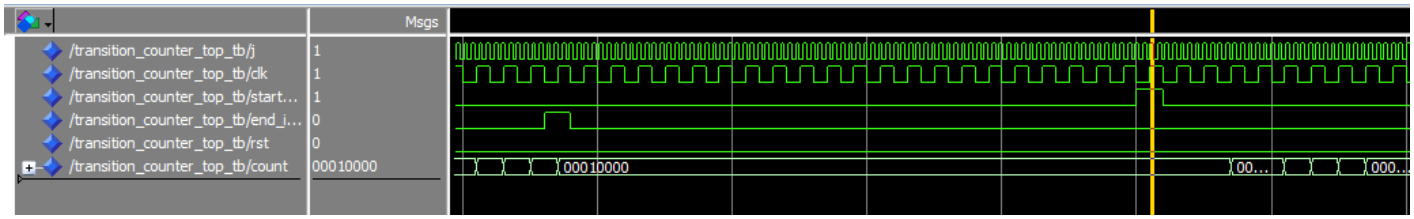
The circuit also resets when it hits 256 on the counter.

Test 2 - End input and J alternates every 14 ns:



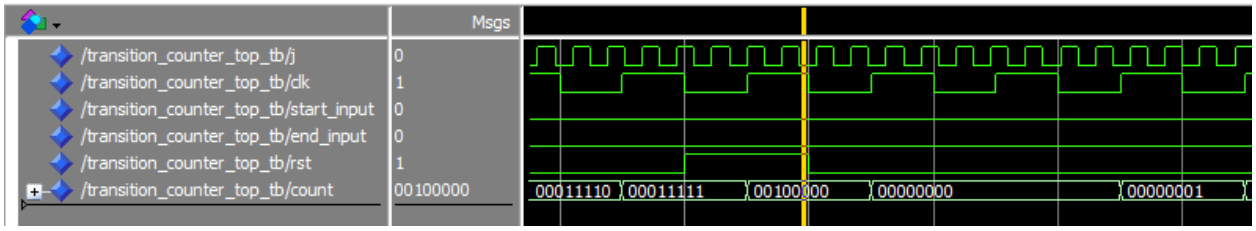
The counter stops counting once end_input becomes ‘1’.

Test 3 - Start input after end input (J alternates every 14 ns):



The counter stops once end_input becomes ‘1’ but resumes when start_input becomes ‘1’.

Test 4 - Reset after the start input after end input (J alternates every 14 ns):



The counter gets reset once rst becomes ‘1’, but the counter keeps counting.

Conclusion:

This assignment allowed us to work on many aspects of RTL design. It furthered our understanding of VHDL and it also strengthened our knowledge on data paths and controllers and how to best implement them into our design.