Lab 4. Modulus 10 Synchronous Counter

Use Quartus to design a **Mod10 Counter** (0…9) that uses a **falling** clock edge. **You must use the two segment method (see slides 30 and 31) in the Part 2 Sequential power point reader).** It should have an async clear (which is the same as reset in Chu's code), synchronous parallel load, enable input, and a carry out (o_rco). See the table below. When loading a 9 or trying a number higher, it stays 9 after a clock pulse (doesn't alternate between the higher number (or 9) and 0) as long as load is high.

It should be clocked by the *board's internal 50 MHz clock (look up the pin name in the board manual)* that has been divided down to 1 Hz. Use the internal clock source from the board to connect to the Georgia Tech 50-MHz clock divider clk_div vhdl file on Sakai to slow down the frequency. Display the count on a seven segment display. Use a top-level file to connect the clock divider, counter, and seven_seg decoder using the two signal names below. Connect the unused clock divider outputs (not shown in the figure) to "open". Connect the rco to LEDR[0]. Connect the rest of the input pins to slide switches as follows (d3 is the msb):

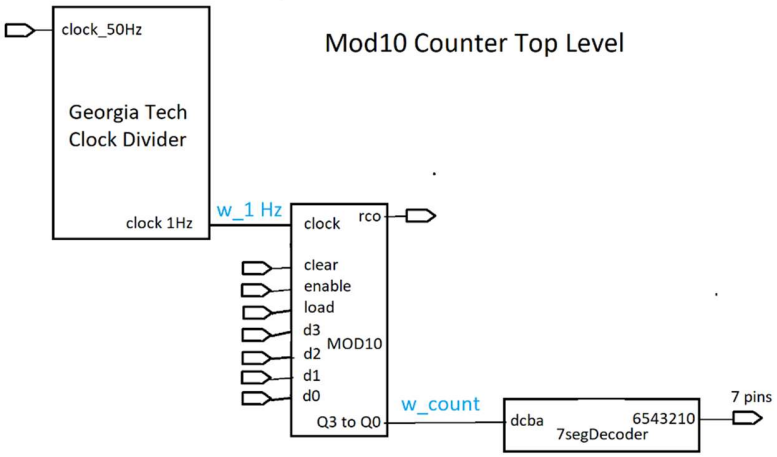
clear – SW[6]
enable – SW[5]
load – SW[4]
d3 – SW[3]
d2 – SW[2]
d1 – SW[1]
d0 – SW[0]

Use a **GENERIC** number of bits for the standard logic vectors d and q (load value and count out), as well as for the unsigned signals r_reg and r_next. Make the generic number of bits have a default value 4.

This lab has historically been harder than it may initially seem, so take extra time testing the code on the board to make sure all the requirements for the demo are met.

Design a decade counter using VHDL. The mod-10 counter has a count enable, parallel load, and asynchronous clear controls as shown in Table 14-1. The counter should also produce a ripple carry output signal (rco) that goes high during the last state in the count sequence when the counter is enabled.

clear	load	enable	clock	function
1	X	X	X	CLEAR
0	0	0	↓	HOLD
0	1	X	↓	LOAD
0	0	1	↓	COUNT UP



- For credit for this assignment, submit:

1) Demo it counting through all 10 decimal values displayed on the seven-segment display and automatically rolling over. When you demo, show that o_rco turns on in the last state (9) and o_rco turns on/off when the enable is turned on/off. Also show that loading in a 9 with the enable high has o_rco go high and turning enable off causes o_rco to go low again. Show that you can't load a value higher than 9 [when loading a 9 or trying a number higher, it stays 9 after a clock pulse (doesn't alternate between the higher number (or 9) and 0) as long as load is high].

2) Submit a working .vhd file of the Mod10 Counter.

3) Submit a pdf of the circuit in the RTL viewer.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity Mod10Counter is
   generic (N : integer := 4); -- default 4-bit width
   port (
      i_clk     : in std_logic;
      i_enable  : in std_logic;
      i_load    : in std_logic;
      i_clear   : in std_logic;
      i_dvector : in std_logic_vector(N-1 downto 0);
      o_rco     : out std_logic;
      o_q       : out std_logic_vector(N-1 downto 0)
   );
end Mod10Counter;

architecture TwoSegment of Mod10Counter is
   signal r_reg  : unsigned(N-1 downto 0) := (others => '0'); -- current state
   signal r_next : unsigned(N-1 downto 0);                    -- next state
begin

   process(i_clk, i_clear)
   begin
      if i_clear = '1' then
         r_reg <= (others => '0');
      elsif (i_clk'event and i_clk = '0') then
         r_reg <= r_next;
```

```
    end if;
  end process;

  process(i_enable, i_load, i_dvector, r_reg)
  begin
    -- Default: hold current value
    r_next <= r_reg;

    if i_load = '1' then
      if unsigned(i_dvector) > 9 then
        r_next <= (others => '0'); -- load 0 if input > 9
      else
        r_next <= unsigned(i_dvector);
      end if;
    elsif i_enable = '1' then
      if r_reg = 9 then
        r_next <= (others => '0'); -- roll over at 9
      else
        r_next <= r_reg + 1;
      end if;
    end if;
  end process;

  o_q   <= std_logic_vector(r_reg);
  o_rco <= '1' when (r_reg = 9 and i_enable = '1') else '0';

end TwoSegment;
```