1 Lab 1A:

1.1 VHDL Code for 1 Bit Adder

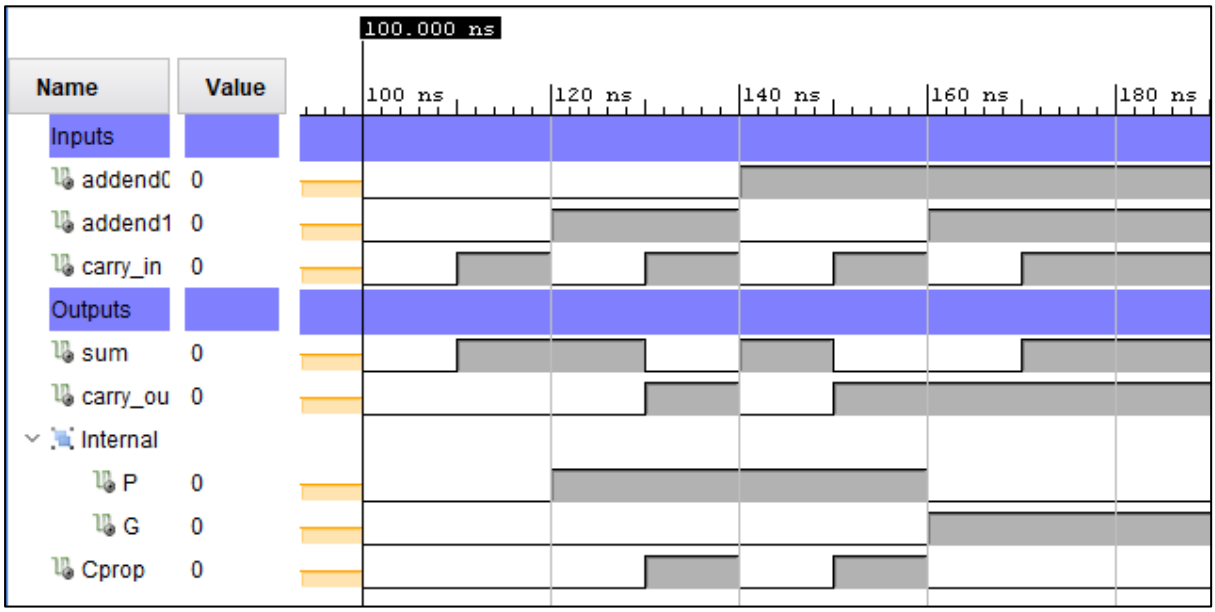
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- asynchronous two bit binary adder with output result as binary addition
-- with overflow input and output.

entity full_adder_1bit is
    Port (

           addend0   : in  STD_LOGIC;  -- 1 bit input to be added
           addend1   : in  STD_LOGIC;  -- 1 bit input to be added
           carry_in  : in  STD_LOGIC;  -- carry in 1 or 0 to be added to addend1 and
                                          addend 0

           sum       : out STD_LOGIC;  -- result of addend0, addend1 and carry in
                                          all summed
           carry_out : out STD_LOGIC); -- carry_out 1 if sum overflows
end full_adder_1bit;

architecture Behavioral_1bit of full_adder_1bit is

signal G     : STD_Logic;
signal P     : STD_Logic;
signal Cprop : STD_Logic;

begin
 G         <= addend0 and addend1;
 P         <= addend0 xor addend1;
 Cprop     <= P and carry_in;

 carry_out <= Cprop or G;
 sum       <= P xor Carry_in;

end Behavioral_1bit;
```

1.2 Testbench Code for 1 Bit Adder

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


entity full_adder_1bit_tb is
end full_adder_1bit_tb;

architecture Behavioral of full_adder_1bit_tb is

signal addend0: STD_LOGIC;
signal addend1: STD_LOGIC;
Signal carry_in: STD_LOGIC;


Signal sum:  STD_LOGIC;
Signal carry_out:  STD_LOGIC;

begin
UUT: entity work.full_adder_1bit
 PORT MAP (
   addend0   => addend0,
   addend1   => addend1,
   carry_in  => carry_in,
   sum       => sum,
   carry_out => carry_out);

Test : process
begin

-- tests each of the 8 different input variations to the entity
-- the inputs represented in binary MSB(addend0,addend1,carry_in)LSB
--begin at 000 and increment e.g. 000->001-> 010..... carrying on to targets
  input value 111


  -- wait 100 ns for global reset to finish
  wait for 100 ns;

  addend0  <= '0';
  addend1  <= '0';
  carry_in <= '0';
  wait for 10ns;

  addend0  <= '0';
  addend1  <= '0';
  carry_in <= '1';
  wait for 10ns;

  addend0  <= '0';
  addend1  <= '1';
  carry_in <= '0';
  wait for 10ns;

  addend0  <= '0';
  addend1  <= '1';
  carry_in <= '1';
  wait for 10ns;
```

```
addend0  <= '1';
addend1  <= '0';
carry_in <= '0';
wait for 10ns;

addend0  <= '1';
addend1  <= '0';
carry_in <= '1';
wait for 10ns;

addend0  <= '1';
addend1  <= '1';
carry_in <= '0';
wait for 10ns;

addend0  <= '1';
addend1  <= '1';
carry_in <= '1';
```

1.3 Simulation Results



1.4 RTL Component Statistics

```
---------------------------------------------------------------------------------
Start RTL Component Statistics
---------------------------------------------------------------------------------
Detailed RTL Component Info :
+---XORs :
       2 Input      1 Bit         XORs := 2
---------------------------------------------------------------------------------
Finished RTL Component Statistics
---------------------------------------------------------------------------------
```

1.5 Pin Assignments

```
set_property PACKAGE_PIN F22 [get_ports {carry_in}]
set_property IOSTANDARD LVCMOS18 [get_ports {carry_in}]
set_property PACKAGE_PIN G22 [get_ports {addend0}]
set_property IOSTANDARD LVCMOS18 [get_ports {addend0}]
set_property PACKAGE_PIN H22 [get_ports {addend1}]
set_property IOSTANDARD LVCMOS18 [get_ports {addend1}]
set_property PACKAGE_PIN T22 [get_ports {sum}]
set_property IOSTANDARD LVCMOS18 [get_ports {sum}]
set_property PACKAGE_PIN T21 [get_ports {carry_out}]
set_property IOSTANDARD LVCMOS18 [get_ports {carry_out}]
```

2 Lab 1B

2.1 VHDL Code for 4 Bit Adder

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

-- Four bit binary adder for the addition of 2 four bit signals and a four bit output value
--The Entity contains 4 iterations of a two input 1 bit adder with carry in and carry out
-- bits for overflow carrying over functionality.

entity full_adder_4bit is
    Port (
           First_Addend  : in  STD_LOGIC_VECTOR (3 downto 0); -- 4 bit input to be added
           Second_Addend : in  STD_LOGIC_VECTOR (3 downto 0); -- 4 bit input to be added
           carry_in      : in  STD_LOGIC;                     -- carry in 1 bit passed from previous
                                                                 carryout overflow
           Sum           : out STD_LOGIC_VECTOR (3 downto 0); -- result of addition of First_Addend,
                                                                 Second_Addend & carry in check
           carry_out     : out STD_LOGIC);                    -- over flow result from addition of
                                                                 First_Addend & Second_Addend
end full_adder_4bit;

architecture Behavioral_4bit of full_adder_4bit is
-- naming conventions : Internal_Origin_Destination
-- int: internal wire
-- Fa: full adder
-- following number refers to the full adder number
-- Co refers to Carry out
-- Ci refers to Carry in
signal int_Fa0Co_Fa1Ci : STD_Logic;  -- function is to connect the Carryout of adder 0 to the carry in 1
signal int_Fa1Co_Fa2Ci : STD_Logic;  -- function is to connect the Carryout of adder 1 to the carry in 2
signal int_Fa2Co_Fa3Ci : STD_Logic;  -- function is to connect the Carryout of adder 2 to the carry in 3

begin

-- 1 bit adder 0
full_adder_1bit_0: entity work.full_adder_1bit
port map (
addend0   => First_Addend(0),   -- input of 4 bit vector First_Addend bit '0' connects to 1 bit adder0 addend0
addend1   => Second_Addend(0),  -- input of 4 bit vector Second_Addend bit '0' connects to 1 bit adder0 addend1
sum       => sum(0),            -- output of 1 bit adder 0 sum connects to bit vector sum bit '0'
carry_in  => carry_in,          -- 1 bit carry in signal directly connect to adder 0 carry in
carry_out => int_Fa0Co_Fa1Ci);  -- carry out connects to signal "int_Fa0Co_Fa1Ci" to connect to adder 1
                                   carry in input
-- 1 bit adder 1
full_adder_1bit_1: entity work.full_adder_1bit
port map (
addend0   => First_Addend(1),   -- input of 4 bit vector First_Addend bit '1' connects to 1 bit adder1
                                   addend0
addend1   => Second_Addend(1),  -- input of 4 bit vector Second_Addend bit '1' connects to 1 bit adder1
                                   addend1
sum       => sum(1),            -- output of 1 bit adder 1 sum connects to bit vector sum bit '1'
carry_in  => int_Fa0Co_Fa1Ci,   -- 1 bit carry in connects to signal "int_Fa0Co_Fa1Ci" from adder 0 carry
                                   out
carry_out => int_Fa1Co_Fa2Ci);  -- carry out connects to signal "int_Fa1Co_Fa2Ci" to connect to adder 2
                                   carry in input

-- 1 bit adder 2
full_adder_1bit_2: entity work.full_adder_1bit
port map (
addend0   => First_Addend(2),    -- input of 4 bit vector First_Addend bit '2' connects to 1 bit adder2
                                 --addend0
addend1   => Second_Addend(2),  -- input of 4 bit vector Second_Addend bit '2' connects to 1 bit adder2
                                 -- addend1
sum       => sum(2),            -- output of 1 bit adder 2 sum connects to bit vector sum bit '2'
```

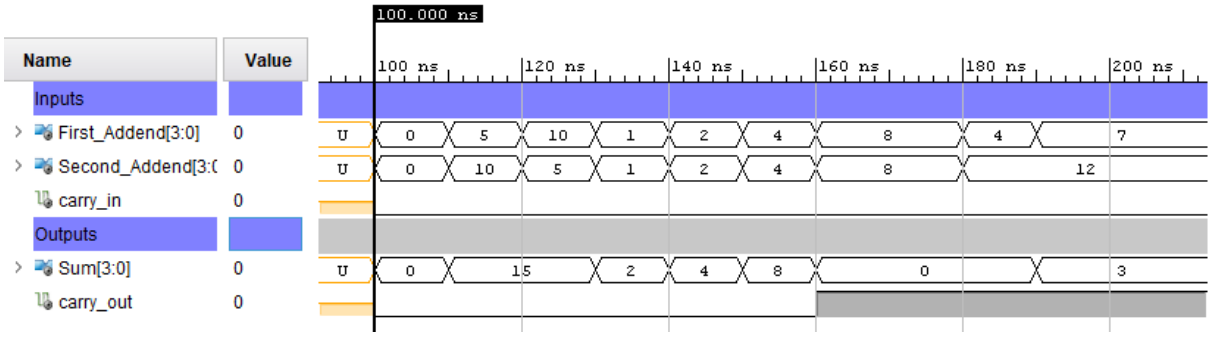
```
carry_in  => int_Fa1Co_Fa2Ci,   -- 1 bit carry in connects to signal "int_Fa1Co_Fa2Ci" from adder 1 carry
                                --  out
carry_out => int_Fa2Co_Fa3Ci);  -- carry out connects to signal "int_Fa2Co_Fa3Ci" to connect to adder 3
                                   carry in input

-- 1 bit adder 3
full_adder_1bit_3: entity work.full_adder_1bit
port map (
addend0   => First_Addend(3),  -- input of 4 bit vector First_Addend bit '3' connects to 1 bit adder3 addend0
addend1   => Second_Addend(3), -- input of 4 bit vector Second_Addend bit '3' connects to 1 bit adder3 addend1
sum       => sum(3),           -- output of 1 bit adder 3 sum connects to bit vector sum bit '3'
carry_in  => int_Fa2Co_Fa3Ci,  -- 1 bit carry in connects to signal "int_Fa2Co_Fa3Ci" from adder 2 carry out
carry_out => carry_out);       -- carry out connects directly to output port carry_out
```

2.2 Testbench Code for 4 Bit Adder

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

entity full_adder_4bit_tb is
end full_adder_4bit_tb;

architecture Behavioral_4bit of full_adder_4bit_tb is
signal First_Addend  : STD_LOGIC_VECTOR (3 downto 0);
signal Second_Addend : STD_LOGIC_VECTOR (3 downto 0);
signal carry_in      : STD_LOGIC;

signal Sum           : STD_LOGIC_VECTOR (3 downto 0);
signal carry_out     : STD_LOGIC;

begin

UUT: entity work.full_adder_4bit
    PORT MAP (First_Addend  => First_Addend,
              Second_Addend => Second_Addend,
              carry_in      => carry_in,
              Sum           => Sum,
              carry_out     => carry_out);

TEST : process
-- testing strategy:
-- test that the whole  4 bit adder works
--correcty for numbers inluding and below 15

--test the carry_in and carry_out connection
--between each 1 bit counter

--test the final carry out which makes anwser
--given the remainder of whatever number it is minus 15

begin
-- wait 100 ns for global reset tot finish
wait for 100 ns;
-- testing 4 bit adder works correctly
--for output sum"0000" carry_out'0'
First_Addend  <= "0000";
Second_Addend <= "0000";
carry_in      <= '0';
wait for 10 ns;

--testing each adders inputs and result for both
-- check individual bits add together
--without over flow for output sum"1111" carry_out'0'
First_Addend  <= "0101";
Second_Addend <= "1010";
carry_in      <= '0';
wait for 10 ns;

-- check individual bits add together
--without over flow for output sum"1111" carry_out'0'
First_Addend  <= "1010";
Second_Addend <= "0101";
carry_in      <= '0';
wait for 10 ns;
```

```
--testing each 1 bit adder carryout with the next carry in
-- Check Carry_in for full_adder_1bit_1
--for output sum"0010" carry_out'0'
First_Addend  <= "0001";
Second_Addend <= "0001";
carry_in      <= '0';
wait for 10 ns;

--  Check Carry_in for full_adder_1bit_2
--for output sum"0100" carry_out'0'
First_Addend  <= "0010";
Second_Addend <= "0010";
carry_in      <= '0';
wait for 10 ns;

--  Check Carry_in for full_adder_1bit_3
--for output sum"1000" carry_out'0'
First_Addend  <= "0100";
Second_Addend <= "0100";
carry_in      <= '0';
wait for 10 ns;

--  Check Carry_out for full_adder_1bit_3
--for output sum"0000" carry_out'1'
First_Addend  <= "1000";
Second_Addend <= "1000";
carry_in      <= '0';
wait for 10 ns;

--for output cannot be used as to big output vector
--size for  carry_out'1'
First_Addend  <= "1000";
Second_Addend <= "1000";
carry_in      <= '0';
wait for 10 ns;

--for output cannot be used as to big output vector
--size for  carry_out'1'
First_Addend  <= "0100";
Second_Addend <= "1100";
carry_in      <= '0';
wait for 10 ns;

--for output cannot be used as to big output vector
--size for  carry_out'1'
First_Addend  <= "0111";
Second_Addend <= "1100";
carry_in      <= '0';
wait for 10 ns;


wait; -- will wait forever
end process;

end Behavioral_4bit;
```

2.3 Simulation Results



2.4 RTL Component Statistics & RTL Hierarchical Component Statistics

```
---------------------------------------------------------------------------------
Start RTL Component Statistics
---------------------------------------------------------------------------------
Detailed RTL Component Info :
+---XORs :
2 Input      1 Bit         XORs := 8
---------------------------------------------------------------------------------
Finished RTL Component Statistics
---------------------------------------------------------------------------------
---------------------------------------------------------------------------------
Start RTL Hierarchical Component Statistics
---------------------------------------------------------------------------------
Hierarchical RTL Component report
Module full_adder_1bit
Detailed RTL Component Info :
+---XORs :
2 Input      1 Bit         XORs := 2
---------------------------------------------------------------------------------
Finished RTL Hierarchical Component Statistics
---------------------------------------------------------------------------------
```

3 Lab 2A

3.1 VHDL Code for 4-bit Counters

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
-- The Four_4bit_counters entity is made up of 4 individual counters that
--increment by one at the rising clock edge. They all count from 0 to 15
-- then role over back to zero again and continue incrementing.
-- the four different types of counters are:
--Synchronous reset and no enable
-- Asynchronous reset(clear) and no enable
-- Synchronous reset and enable
-- Aynchronous reset(clear) and enable
entity Four_4bit_counters is
Port (
clk : in STD_LOGIC;
rst : in STD_LOGIC;
 en : in STD_LOGIC;

Sync_Reset_No_Enable : out UNSIGNED (3 downto 0);
Asyn_Reset_No_Enable : out UNSIGNED  (3 downto 0);
Sync_Reset_Enable : out UNSIGNED  (3 downto 0);
Asyn_Reset_Enable : out UNSIGNED  (3 downto 0));
end Four_4bit_counters;

architecture Behavioral of Four_4bit_counters is
-- signals decelerations
--Each of these signals is used in its appropriate counter to
--increment the current value and connect to the corresponding 4 bit
--output port of the entity.
signal Sync_Reset_No_Enable_int : UNSIGNED(3 downto 0);
signal Asyn_Reset_No_Enable_int : UNSIGNED(3 downto 0);
signal Sync_Reset_Enable_int : UNSIGNED(3 downto 0);
signal Asyn_Reset_Enable_int : UNSIGNED(3 downto 0);
begin
--Synchronous reset and no enable
 CNT_SYN_No_EN : process(clk)
 begin
 if(rising_edge(clk)) then
    if (rst= '1') then
        Sync_Reset_No_Enable_int <=(others => '0');
    else
        Sync_Reset_No_Enable_int <= (Sync_Reset_No_Enable_int +1);
    end if;
 end if;
 end process CNT_SYN_No_EN;
 Sync_Reset_No_Enable <= Sync_Reset_No_Enable_int;

  -- Asynchronous reset(clear) and no enable
  CNT_ASYN_NO_EN : process(clk,rst)
 begin
 if (rst= '1') then
 Asyn_Reset_No_Enable_int <=(others => '0');
 else
     if(rising_edge(clk)) then
         Asyn_Reset_No_Enable_int <= (Asyn_Reset_No_Enable_int +1);
     end if;
 end if;
 end process CNT_ASYN_NO_EN;
 Asyn_Reset_No_Enable <= Asyn_Reset_No_Enable_int;
```

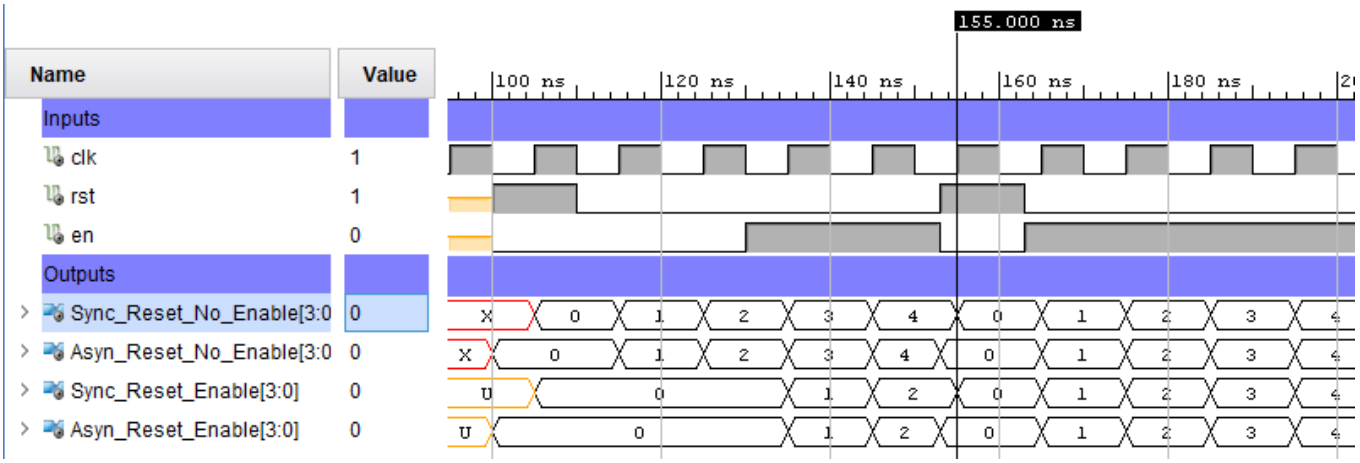
```
-- Synchronous reset and enable
 CNT_SYN_EN : process(clk)
 begin
 if(rising_edge(clk)) then
    if (rst= '1') then
        Sync_Reset_Enable_int <=(others => '0');
    else
         if(en= '1') then
            Sync_Reset_Enable_int <= (Sync_Reset_Enable_int +1);
         end if;
    end if;
 end if;
 end process CNT_SYN_EN;
 Sync_Reset_Enable <= Sync_Reset_Enable_int;

 -- Asynchronous reset(clear) and enable
  CNT_ASYN_EN : process(clk,rst)
 begin
 if (rst= '1') then
 Asyn_Reset_Enable_int <=(others => '0');
 else
     if(rising_edge(clk)) then
         if(en= '1') then
                Asyn_Reset_Enable_int <= (Asyn_Reset_Enable_int +1);
         end if;
     end if;
 end if;
 end process CNT_ASYN_EN;
 Asyn_Reset_Enable <= Asyn_Reset_Enable_int;

end Behavioral;
```

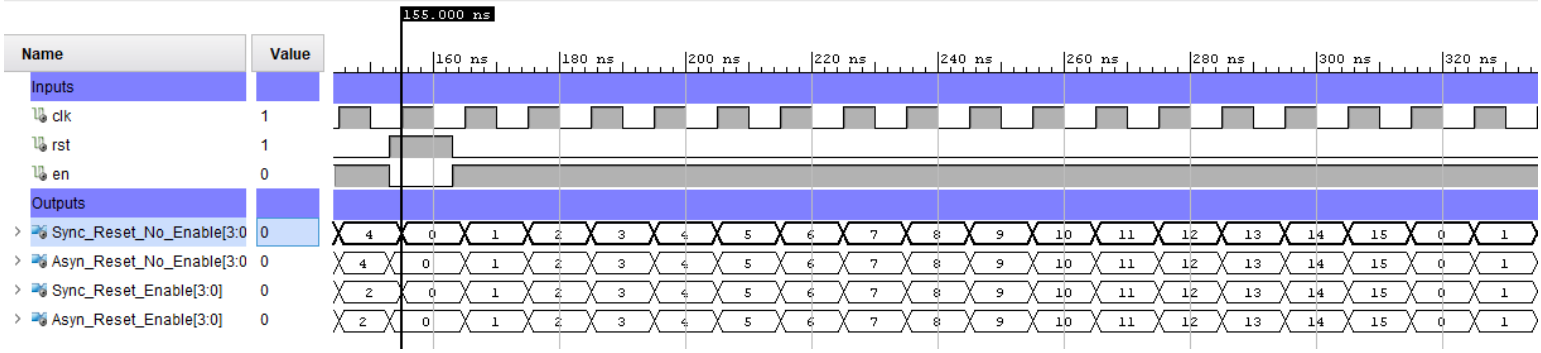
3.2 Testbench Code for 4-Bit Adder

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

entity Four_4bit_counters_tb is
end Four_4bit_counters_tb;

architecture Behavioral of Four_4bit_counters_tb is


signal clk : STD_LOGIC;
signal rst : STD_LOGIC;
signal en  : STD_LOGIC;


signal Sync_Reset_No_Enable: UNSIGNED(3 downto 0);
signal Asyn_Reset_No_Enable: UNSIGNED(3 downto 0);
signal Sync_Reset_Enable:    UNSIGNED(3 downto 0);
signal Asyn_Reset_Enable:    UNSIGNED(3 downto 0);


constant clk_period : time := 10ns;

begin

UUT: entity work.Four_4bit_counters
    PORT MAP (clk  => clk ,
              rst => rst,
              en  => en,
              Sync_Reset_No_Enable => Sync_Reset_No_Enable,
              Asyn_Reset_No_Enable => Asyn_Reset_No_Enable,
              Sync_Reset_Enable => Sync_Reset_Enable,
              Asyn_Reset_Enable => Asyn_Reset_Enable);

-- Clock process
-- clock period set to 10ns, repeats forever
clk_process :process
begin
 clk <= '0';
 wait for clk_period/2;
 clk <= '1';
 wait for clk_period/2;
end process;

TEST : process
-- testing process checks :
-- counter outputs begin from zero at reset
-- Synchronous reset counters reset output value to 0 at rising edge of clock
-- Asynchronous reset counters reset output value to 0 at point reset signal is 1
-- check that all counters count to 15 then roll over back to 0
-- check that counters with enable signal only increment when enable is 1
begin
-- wait 100 ns for global reset tot finish set by Xilinx
wait for 100 ns;
```

```
-- sets start time for changing input values on falling edge of clock
--this allows clear changes of outputs on rising edge that do not
-- interfere
wait until falling_edge(clk);

-- proves asynchronous resets work instantanously and synchronous waits for clk
--rising edge
--wait for clk_period;
 rst <= '1';
 en  <= '0';
 wait for clk_period;

-- allowing counters without enable to increment
 rst <= '0';
 en  <= '0';
 wait for clk_period*2;

 -- allow counters with enable to increment
wait until falling_edge(clk);
rst <= '0';
en  <= '1';
 wait for clk_period*2;
 -- offset reset to prove asynchronous reset not on clk edge
 wait for clk_period*0.3;

 --reset values back to zero syncronous and asynchronous
 rst <= '1';
 en  <= '0';
 wait for clk_period;

 --count increment until transiton off 15 to 0 in hex
 rst <= '0';
 en  <= '1';
 wait for clk_period*10;
 wait;-- wait forever
end process;

end Behavioral;
```

3.3 Simulation Results





3.4 RTL Component Statistics

```
---------------------------------------------------------------------------------
Start RTL Component Statistics
---------------------------------------------------------------------------------
Detailed RTL Component Info :
+---Adders :
       2 Input      4 Bit       Adders := 4
+---Registers :
                    4 Bit    Registers := 4
---------------------------------------------------------------------------------
Finished RTL Component Statistics
---------------------------------------------------------------------------------
```

4 Lab2B

4.1 VHDL Code for 0-99 Counter

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
--This Entity IS a 0-99 counter that outputs 2, 7 bit outputs to  two 7
-- segment displays. The number incremments by 1 each time the enable
--button is pressed at the clock signal and resets back to 0 when the
--reset button is pressed or the button is pressed or counter
--increments 99 by 1 all at clock signal.
-- the Entire entity uses 2 debouncers one for each buttons, enable and
reset.
-- It uses a singgle 0-99 counter and 2 4bit to 7 bit decoders.
entity Counter_0_to_99 is
    Port ( clk : in STD_LOGIC;
           rst : in STD_LOGIC;
           en : in STD_LOGIC;
           Units_7_seg : out STD_LOGIC_VECTOR (6 downto 0);
           -- output to the units 7 segment display
           Tens_7_seg : out STD_LOGIC_VECTOR (6 downto 0));
           -- output to the Tens 7 segment display
end Counter_0_to_99;

architecture Behavioral of Counter_0_to_99 is

signal int_debounced_rst_sig : std_logic;
signal int_debounced_en_sig : std_logic;

signal Unit_4bit_seg: UNSIGNED(3 downto 0);
-- 4 bit unit signal from 0-99 counter to decoder

signal Tens_4bit_seg: UNSIGNED(3 downto 0);
-- 4 bit tens signal from 0-99 counter to decoder
begin

COUNT_0_99: entity work.Count_to_99
port map (
   en => int_debounced_en_sig,   --debounced enable signal
  rst => int_debounced_rst_sig,  --debounced reset signal
  clk => clk,
 tens => Tens_4bit_seg,  --tens unit 4 bit output to decoder
 Unit => Unit_4bit_seg); -- unit 4 bit output to decoder

Enable_Debounce: entity work.Debouncer
port map (
 clk => clk,
 sig =>  en,                          -- enable input to debouncer
 deb_sig => int_debounced_en_sig  );-- debounced output enable signal

Reset_Debounce: entity work.Debouncer
port map (
clk => clk,
sig => rst ,                          -- reset input to debouncer
deb_sig => int_debounced_rst_sig ); -- debounced output reset signal

Unit_7_seg_disp: entity work.Decoder_7seg_Display
port map (
Four_bit_input => Unit_4bit_seg ,
Seven_bit_output => Units_7_seg  );
```

```
Tens_7_seg_disp: entity work.Decoder_7seg_Display
port map (
Four_bit_input   => Tens_4bit_seg ,
Seven_bit_output => Tens_7_seg  );
end Behavioral;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
-- This 0-99 counter increments by 1 at each pulse of the
--input debounced enable signal and at the clk edge.
-- It resets to zero when a debounced reset signal is
-- introduced and at the clk edge.
-- The entire counter is made up of 2 smaller counters each counting
--from 0-9 one for tens and one for units, the units increments by one
--upon the enable signal and clk rising edge. while the tens unit requires
-- the clk edge, the enable signal and the units couter value to be 9.
-- both tens and units counters have synchronous rest.
entity Count_to_99 is
    Port (

           clk : in STD_LOGIC;
           en : in STD_LOGIC;
           rst : in STD_LOGIC;


           Tens : out UNSIGNED (3 downto 0);
           Unit : out UNSIGNED (3 downto 0));
end Count_to_99;

architecture Behavioral of Count_to_99 is

signal int_CNTunit: UNSIGNED(3 downto 0 ); -- units value to be incremented, reset,
                                           checked and connected to output
signal int_CNTtens: UNSIGNED(3 downto 0 ); -- tens value to be incremented, reset,
                                           checked and connected to output
begin
cnt_unit: process(clk)
 begin
 if(rising_edge(clk)) then
    if(rst= '1') then
        int_CNTunit <=(others => '0');
    else
        if (en= '1') then
             if(int_CNTunit = 9) then -- eq. to (CNTunit = "1001")
                int_CNTunit <=(others => '0');
              else int_CNTunit <=(int_CNTunit+1);
              end if;
        end if;
     end if;
 end if;
 end process cnt_unit;


 cnt_tens: process(clk)
  begin
 if(rising_edge(clk)) then
     if(rst= '1') then
         int_CNTtens <=(others => '0');
     else
         if (en = '1') and (int_CNTunit="1001") then - checks debounced enable signal &
                                                       output of units is equal to 9
              if(int_CNTtens = 9) then -- eq. to (CNTunit = "1001")
                 int_CNTtens <=(others => '0');
               else int_CNTtens <=(int_CNTtens+1);
               end if;
         end if;
      end if;
 end if;
  end process CNT_tens;

  --Output connections
  Unit   <= int_CNTunit;
  Tens <= int_CNTtens;

  end Behavioral;
```

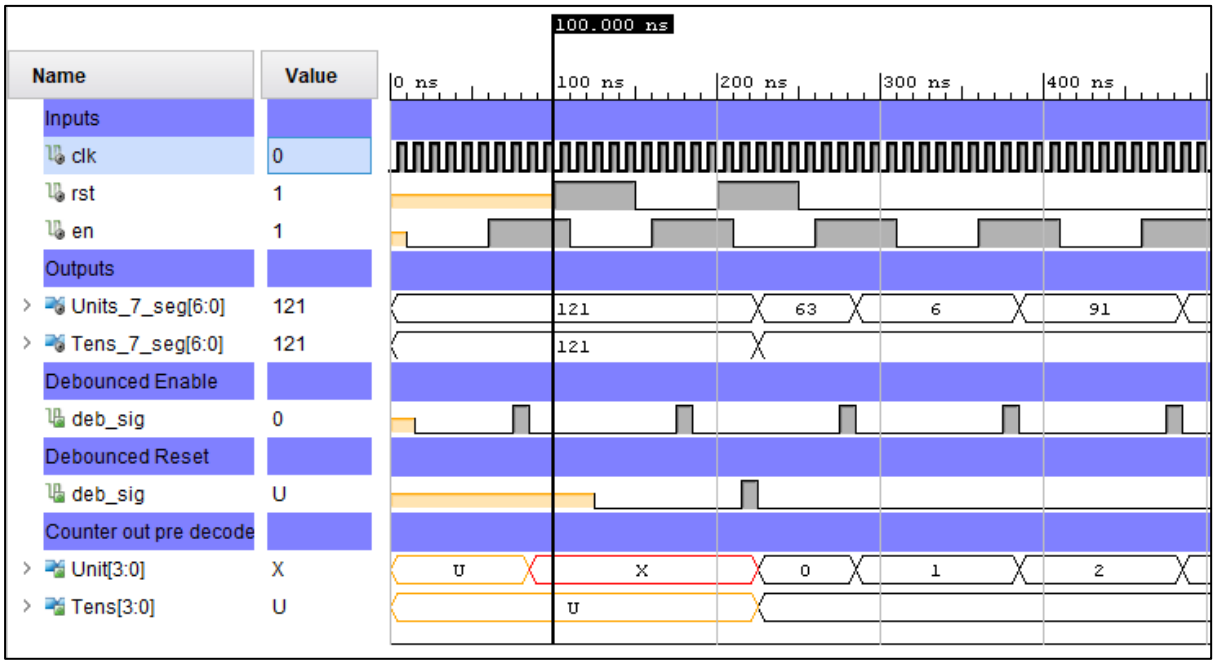
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity Decoder_7seg_Display is
    Port ( Four_bit_input : in UNSIGNED(3 downto 0);
           Seven_bit_output : out STD_LOGIC_VECTOR (6 downto 0));
end Decoder_7seg_Display;

-- 7 segment decoder bit values of the 6 down to zero output
-- displayed belowworking top middle and moving clock wise
-- around and finishing in the centre
--  the bit order is 0 top middle horizontal
--                   1 top right vertical
--                   2 bottom right vertical
--                   3 bottom middle horizontal
--                   4 left bottom vertical
--                   5 top left vertical
--                   6 middle horizontal
--
--   _
--  |_|
--  |_|

-- 0 =            _       1 =
--        --     | |             |
--        --     |_|             |

-- 2 =            _       3 =     _
--        --      _|             _|
--        --     |_             _|

-- 4 =            _       5 =     _
--        --     |_|           |_
--        --       |            _|

-- 6 =            _       7 =     _
--        --     |_             |
--        --     |_|            |

-- 8 =            _       9 =    _
--        --     |_|          |_|
--        --     |_|           _|

-- Error state =          _
--                  --   |_
--                  --   |_
architecture Behavioral of Decoder_7seg_Display is

begin
with Four_bit_input select
Seven_bit_output <= "0111111" when "0000", --0 dec=  63
                    "0000110" when "0001", --1 dec =  6
                    "1011011" when "0010", --2 dec = 91
                    "1001111" when "0011", --3 dec = 79
                    "1100110" when "0100", --4 dec =102
                    "1101101" when "0101", --5 dec =109
                    "1111101" when "0110", --6 dec =125
                    "0000111" when "0111", --7 dec =  7
                    "1111111" when "1000", --8 dec =127
                    "1101111" when "1001", --9 dec =111
                    "1111001" when others; --E dec =121  error state
end Behavioral;
```

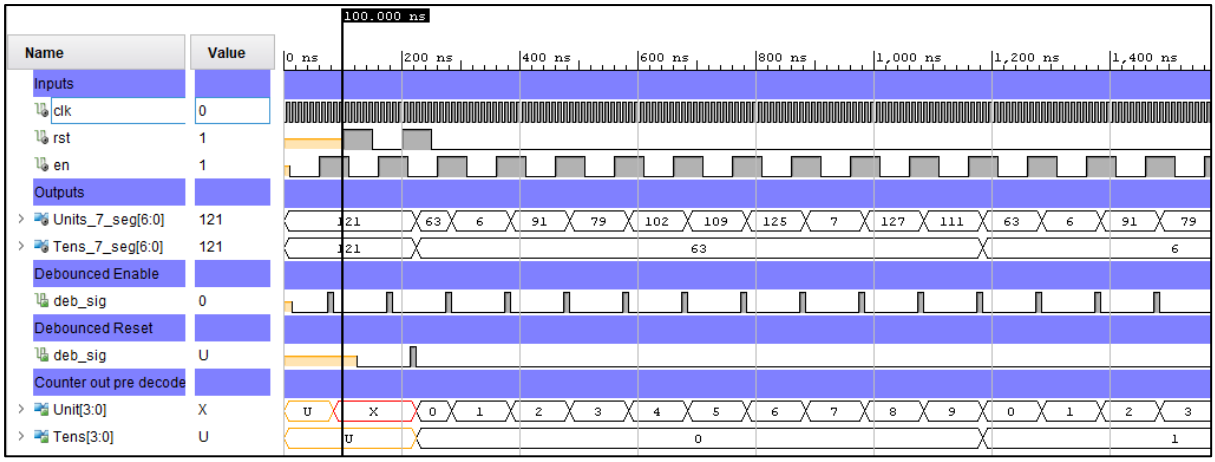
4.2 Testbench Code for 0-99 Counter

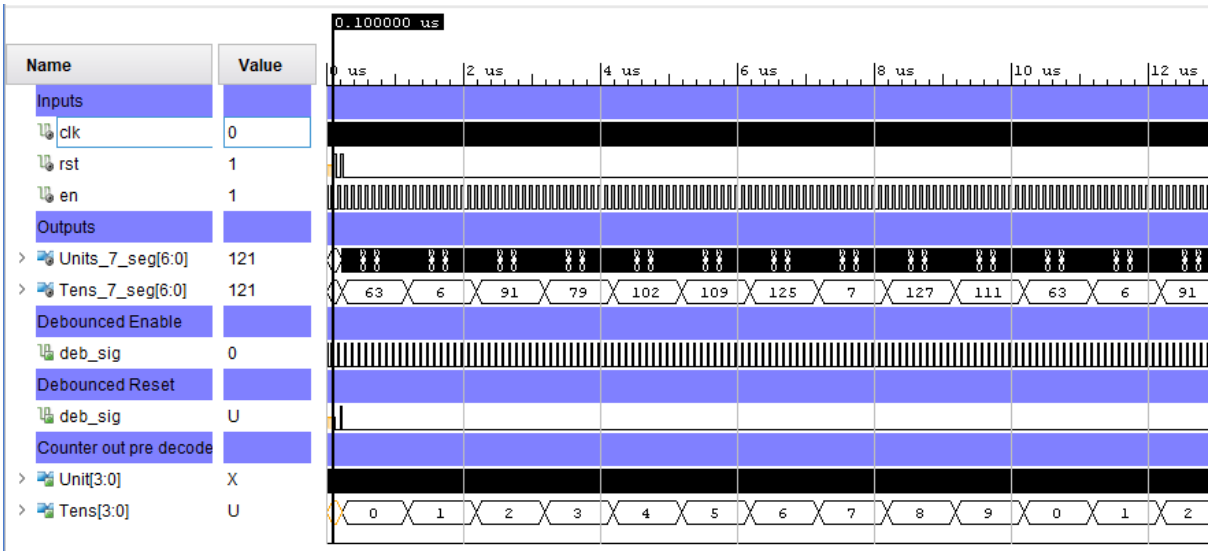
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity Counter_0_to_99_tb is

end Counter_0_to_99_tb;

architecture Behavioral of Counter_0_to_99_tb is

--input signals
signal clk : STD_LOGIC;
signal rst : STD_LOGIC;
signal en  : STD_LOGIC;

--output signals
signal Units_7_seg : STD_LOGIC_VECTOR (6 downto 0);
signal Tens_7_seg :  STD_LOGIC_VECTOR (6 downto 0);

--constants
constant clk_period : time := 10ns;

begin

UUT: entity work.Counter_0_to_99
    PORT MAP (clk  => clk ,
              rst => rst,
              en  => en,
              Units_7_seg => Units_7_seg,
              Tens_7_seg => Tens_7_seg);

-- Clock process
clk_process :process
begin
 clk <= '0';
 wait for clk_period/2;
 clk <= '1';
 wait for clk_period/2;
end process;

--enable process to repeat simulating
-- an able button pressed
en_process :process
begin
wait until falling_edge(clk);
 en <= '0';
 wait for clk_period*5;
 en <= '1';
 wait for clk_period*5;
end process;


TEST : process
-- testing process tests:
-- synchronous tens and units increments with enable anld clock

-- synchrous reset

-- reset counters values start from 0

--counters limited to 9 then back to 0
-- tens counter only increments when enable actice, clk rising
--edge and units counter  is 9
```

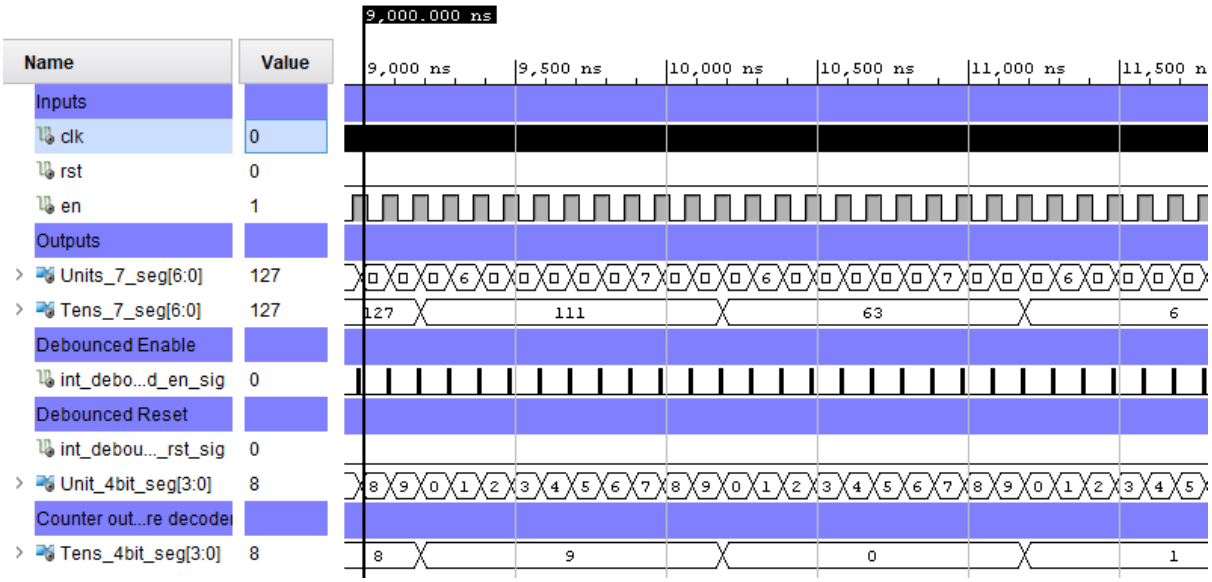
```
begin
-- wait 100 ns for global reset tot finish set by Xilinx
wait for 100 ns;

-- sets start time for changing input values on falling edge of clock
--this allows clear changes of outputs on rising edge that do not
-- interfere
wait until falling_edge(clk);
-- reset twice to clear out the unknown values within the registers
contained
-- within the debounce
rst <= '1';

-- multiples of clock period used to keep input changes on falling clk
edge
 wait for clk_period*5;

 rst <= '0';
 wait for clk_period*5;

 rst <= '1';
 wait for clk_period*5;

 rst <= '0';

 wait for clk_period*5;

 wait; --waits forever
end process;
end Behavioral;
```

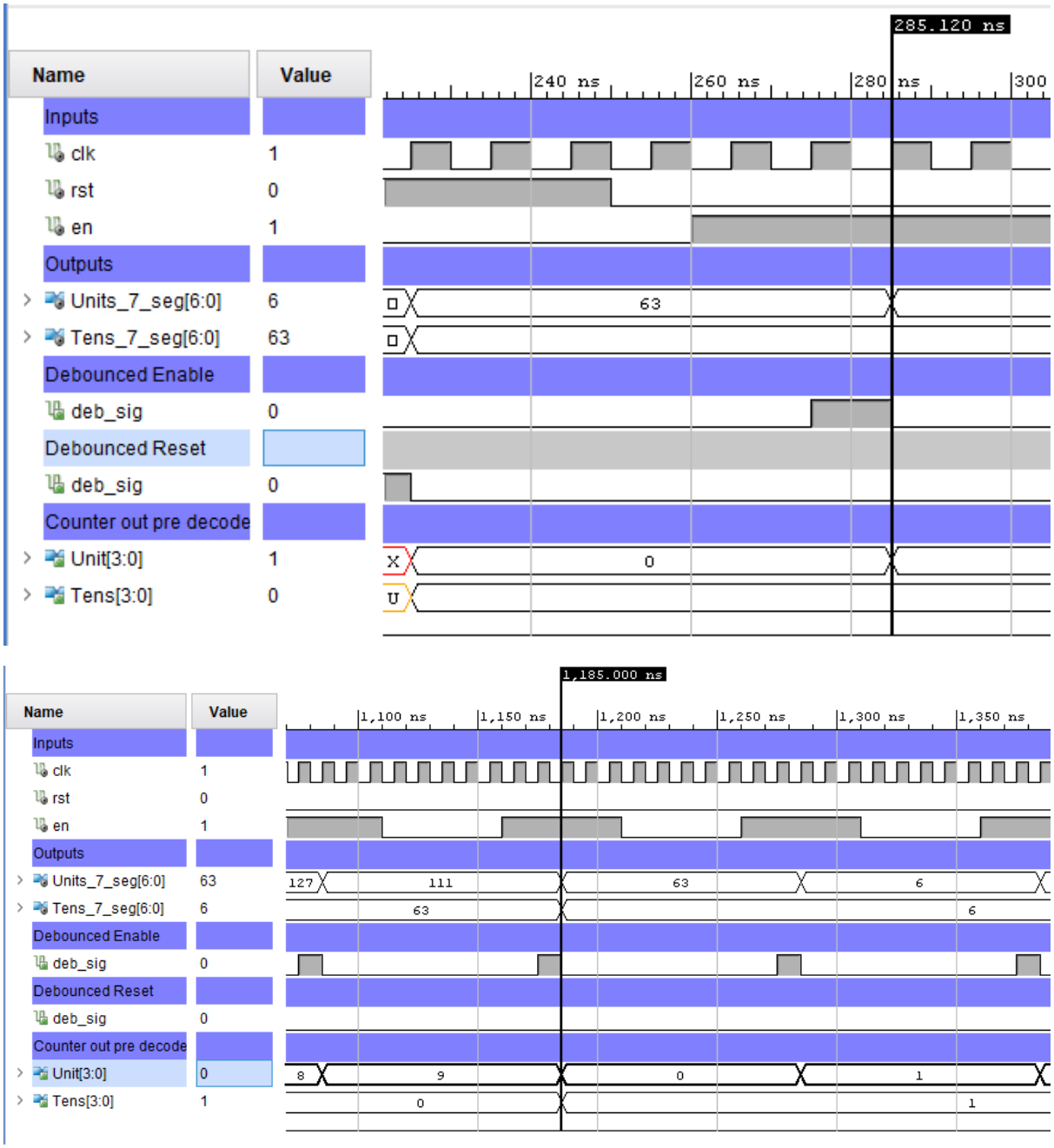
4.3 Simulation Results




100.000 ns
Name
Value
Inputs
clk
0
rst
1
en
1
Outputs
Units_7_seg[6:0]
121
Tens_7_seg[6:0]
121
Debounced Enable
deb_sig
0
Debounced Reset
deb_sig
U
Counter out pre decode
Unit[3:0]
X
Tens[3:0]
U
0 ns
200 ns
400 ns
600 ns
800 ns
1,000 ns
1,200 ns
1,400 ns


0.100000 us
Name
Value
Inputs
clk
0
rst
1
en
1
Outputs
Units_7_seg[6:0]
121
Tens_7_seg[6:0]
121
Debounced Enable
deb_sig
0
Debounced Reset
deb_sig
U
Counter out pre decode
Unit[3:0]
X
Tens[3:0]
U
0 us
2 us
4 us
6 us
8 us
10 us
12 us


9,000.000 ns
Name
Value
Inputs
clk
0
rst
0
en
1
Outputs
Units_7_seg[6:0]
127
Tens_7_seg[6:0]
127
Debounced Enable
int_debo...d_en_sig
0
Debounced Reset
int_debou..._rst_sig
0
Unit_4bit_seg[3:0]
8
Counter out...re decoder
Tens_4bit_seg[3:0]
8
9,000 ns
9,500 ns
10,000 ns
10,500 ns
11,000 ns
11,500 n


285.120 ns
Name
Value
240 ns
260 ns
280 ns
300
Inputs
clk 1
rst 0
en 1
Outputs
Units_7_seg[6:0] 6
63
Tens_7_seg[6:0] 63
Debounced Enable
deb_sig 0
Debounced Reset
deb_sig 0
Counter out pre decode
Unit[3:0] 1
0
Tens[3:0] 0
1,185.000 ns
1,100 ns
1,150 ns
1,200 ns
1,250 ns
1,300 ns
1,350 ns
Inputs
clk 1
rst 0
en 1
Outputs
Units_7_seg[6:0] 63
127
111
63
6
Tens_7_seg[6:0] 6
63
6
Debounced Enable
deb_sig 0
Debounced Reset
deb_sig 0
Counter out pre decode
Unit[3:0] 0
8
9
0
1
Tens[3:0] 1
0
1

RTL Component Statistics & RTL Hierarchical Component Statistics

```
---------------------------------------------------------------------------------
Start RTL Component Statistics
---------------------------------------------------------------------------------
Detailed RTL Component Info :
+---Adders :
       2 Input      4 Bit       Adders := 2
+---Registers :
                    4 Bit    Registers := 2
                    1 Bit    Registers := 6
+---Muxes :
       2 Input      4 Bit        Muxes := 2
---------------------------------------------------------------------------------
Finished RTL Component Statistics
---------------------------------------------------------------------------------
---------------------------------------------------------------------------------
Start RTL Hierarchical Component Statistics
---------------------------------------------------------------------------------
Hierarchical RTL Component report
Module Count_to_99
Detailed RTL Component Info :
+---Adders :
       2 Input      4 Bit       Adders := 2
+---Registers :
                    4 Bit    Registers := 2
+---Muxes :
       2 Input      4 Bit        Muxes := 2
Module Debouncer
Detailed RTL Component Info :
+---Registers :
                    1 Bit    Registers := 3
---------------------------------------------------------------------------------
Finished RTL Hierarchical Component Statistics
---------------------------------------------------------------------------------
```

The RTL Component statistics contain 2 adders which are implemented within the 0-99 adder as designed within the sequential logic. Two 4 bit registers are used to hold the current value within the counters system which makes sense as the values have to be incremented so the current value has to be remembered by the circuit. The 6 one bit registers are used within the debouncers each debouncer uses 3 registers that use 3 clock cycles worth of data with the 2 and gates and an inverted input to detect the falling edge of a button signal. The multiplexers are implemented as a comparator, if the select value equals 9 then the input changes to be equal to “0000” as defined in the if statements. It is strange that a single multiplexer per counter is used rather than a comparator as described in the sequential statement. This implementation makes sense as if a comparator was implemented instead more circuitry (probably a multiplexer) would be required to be implemented to complete the circuit as described.