Describing Sequential Circuits in VHDL
Lab Report 2

Lab Section: 003
Group 29

Zhiyi Sun 260768094
Nariman Zendehrooh 260700556
Mcgill University, Montreal, Quebec, Canada

zhiyi.sun@mail.mcgill.ca
Nariman.zendehroohKermani@mail.McGill.ca

1.0 Table of Contents

2.0 Counter

The counter is built using a sequential circuit. The 4-bits counter consists of four flip flops connecting to each other. The circuit has three inputs include reset, enable and clock and one output which is count. Firstly, the clock signal is connected to the flip flops to keep track of time. The asynchronous reset input is an active-low element. When the reset is 0, it clears the bits and resets the counter to zero. Otherwise, the circuit acts as usual. Also, there is an enable element that when it is high the counter counts up. Otherwise, the counter holds its previous values. In this lab enable is used as the start button. The counter reacts to the positive edge of the clock and adds one to the counter and remembers it. Since the clock changes between 0 and 1 in each time period, the process will be executed sequentially. The clock will not stop unless the program is terminated. Therefore, the counter is a sequential circuit. The counter will keep counting up until it gets equal to 15. After that counter resets to zero. There are two possibilities to do this action. First, we can reset the counter with the reset input when we reach 15. However, this is an abandoned action because this is a 4-bits counter so that when it reaches 16, overflow happens, and the fifth bit becomes 1 and the other bits will become equal to zero which means the counter is equal to zero again. We simulated the counter to check whether it is counting during a specific period of time. The functionality can be interpreted by checking whether the counting happens at the positive edge of the clock and whether counting happening when the enable input is equal to one and whether the counter restarting after counting up to 15.

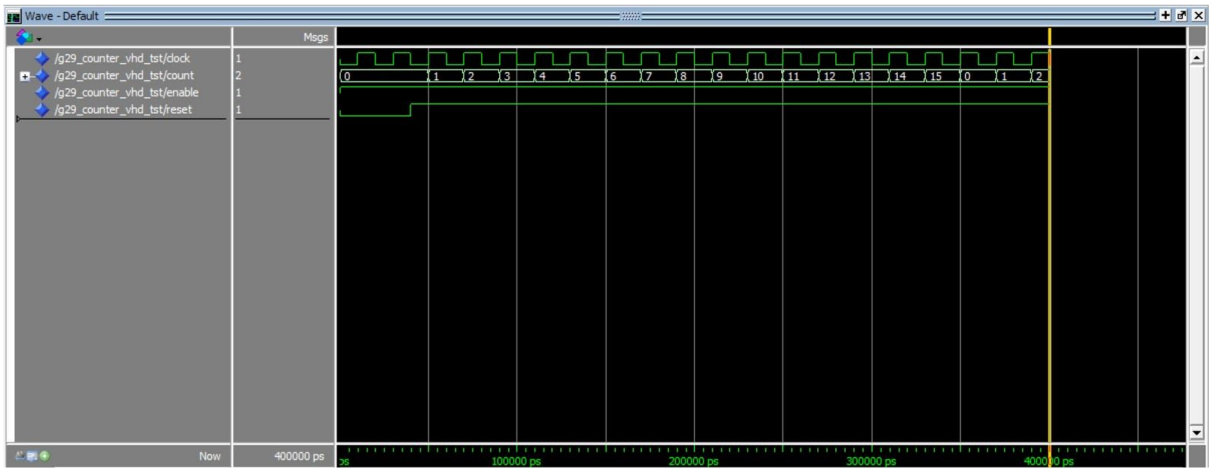


Figure 2.1: Wave of the counter testbench

3.0 Clock Divider

A clock divider is a circuit that generates a signal that is asserted once every 10 milliseconds. This signal will be used as a condition to enable the counters in the stopwatch circuit. Clock Divider divides the input clock frequency of 50MHz and produces an output clock of 1KHz. It means that the period is 1 divided by 50MHz which is 20 nanoseconds. And we want the circuit to generate an enable signal every 10 milliseconds so we divided 10 milliseconds by one period which gave us 500,000. So, we defined Cycles as a constant with the value of 499999. The circuit takes clk, reset and enable as inputs and en_out as an output. Clk is the clock variable. Enable as in counter acts as the start button. If it is 1, the counter begins. Reset also has the same functionality as in the counter which resets the counter to zero. We used the process statement to execute the code block sequentially. The process takes two inputs of clk and reset. If any of these inputs change, the process will be executed. Since clk is changing between zero and one in each time period, the process is executed sequentially. The clock will not stop unless the program is executed. Therefore, the clock divider is a sequential circuit. Inside the code block, the Count signal is set to Cycles. Count will be decremented by one whenever the enable is one. It will continue until the count becomes one. Then, en_out will become one which is the pulse signal. The count will be reassigned to Cycles when it becomes zero. We simulated the circuit and observed its behaviour in the period of 20 milliseconds right after the circuit is restarted. As we expected, we saw two pulses during that period which means the signal pulse happens after each period of 10 milliseconds. Therefore, the test is successful.

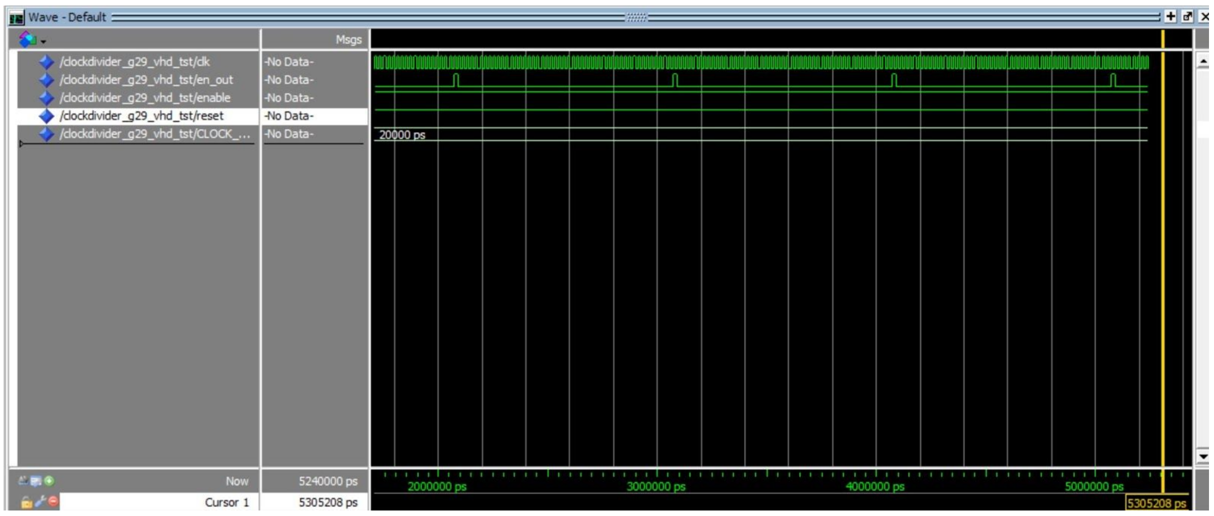

Wave - Default
Msgs
/clockdivider_g29_vhd_tst/clk
/clockdivider_g29_vhd_tst/en_out
/clockdivider_g29_vhd_tst/enable
/clockdivider_g29_vhd_tst/reset
/clockdivider_g29_vhd_tst/CLOCK_...
-No Data-
-No Data-
-No Data-
-No Data-
-No Data-
20000 ps
Now
5240000 ps
Cursor 1
5305208 ps
2000000 ps
3000000 ps
4000000 ps
5000000 ps
5305208 ps

Figure 3.1: Wave of clock divider testbench

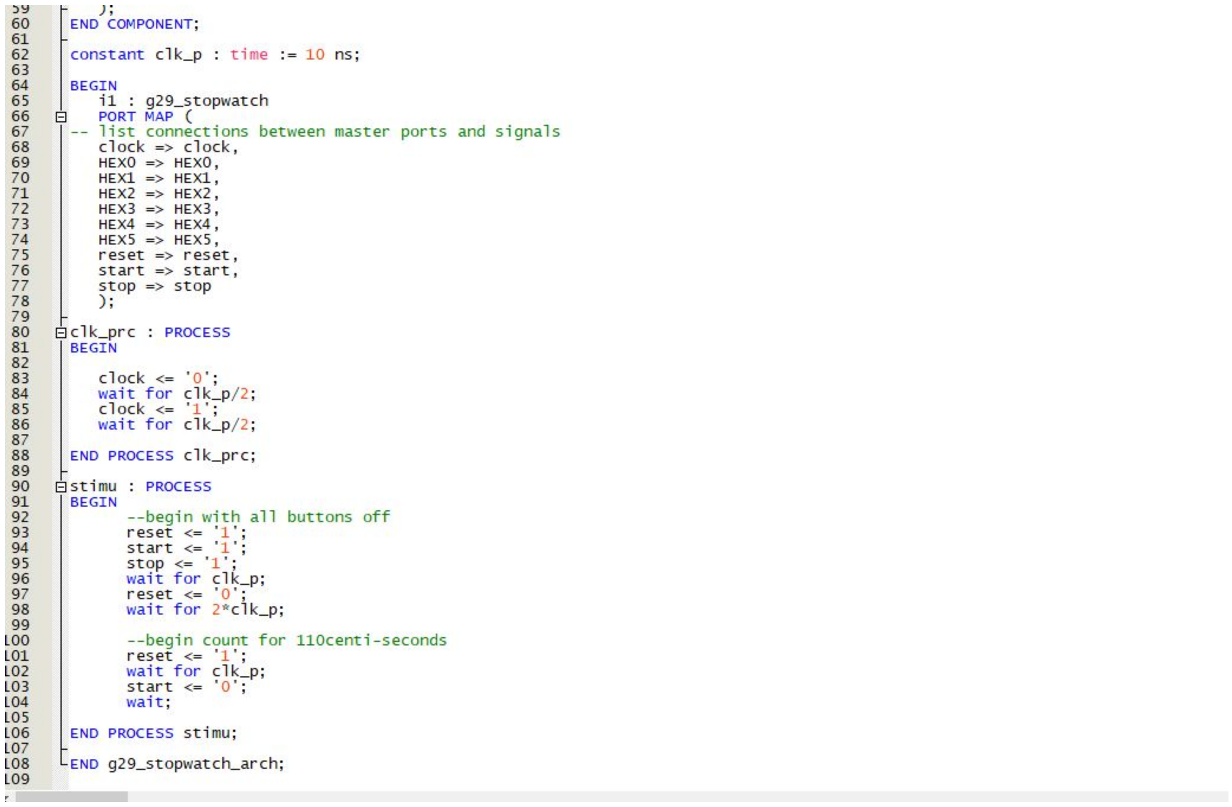
4.0 Advantage of Down-counter

Since our purpose is to count the clock period, both up-counter and down-counter will have the same result. However, if we use the down-counter, the last number of the counter is 0. When the counter reaches 0, we set the output to 1. Otherwise, we keep the output 0. We check the counter whether it reaches 0 by using a NOT gate. If we use the up-counter, the last number of the counter is 499,999. So, we set the output to 1 if the counter reaches 499,999. Therefore, we need more than a NOT gate to check whether the counter reaches this number. So, down-counter is easier to use.

5.0 A Description of the Stopwatch Circuit

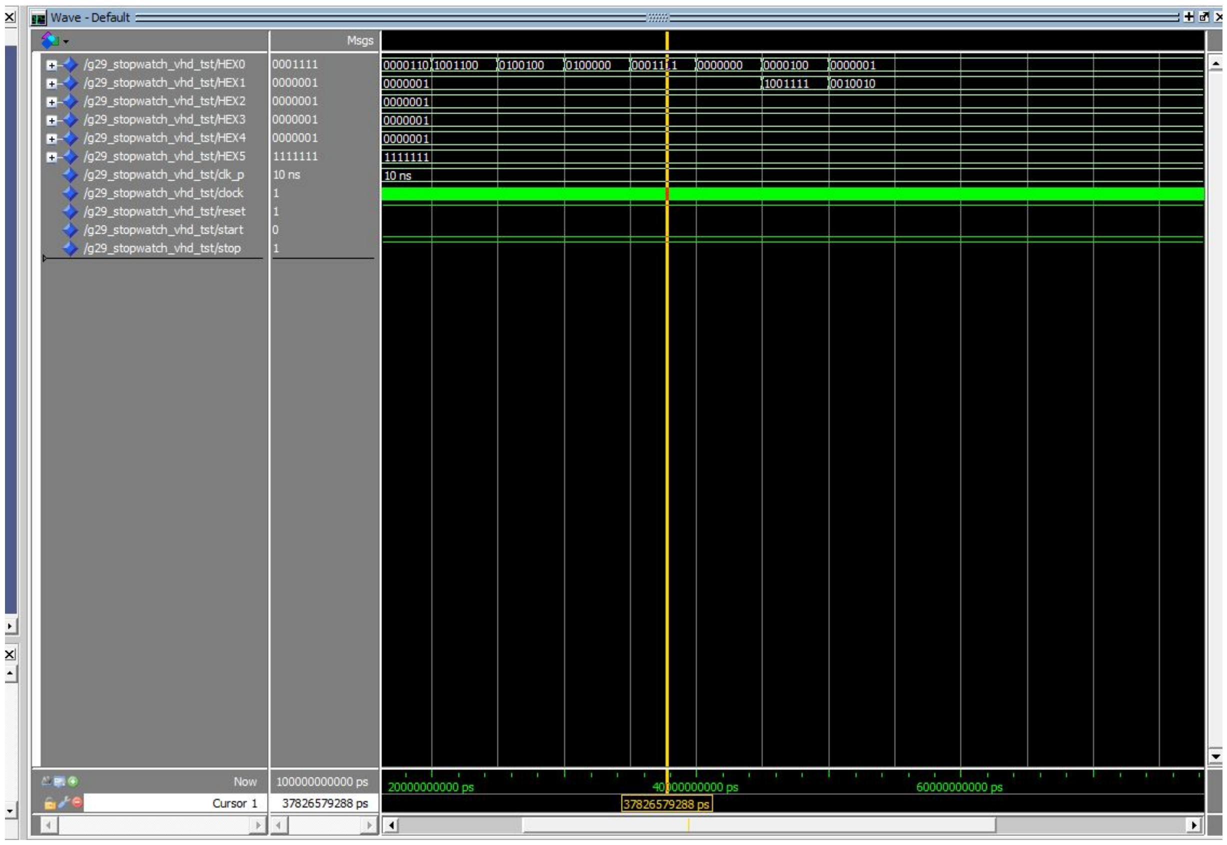
The stopwatch mainly consists of three inputs controlled by buttons (including the start, reset, and stop) and a clock with a specified frequency of 50Mhz. As the clock divider will count cycles (10 ms *50 Mhz -1 = 499999 cycles) for the stopwatch, six counters are implemented for the outputs of each 7-segments-decoders (6 in total as displayed on the board). The counters then count the corresponding numbers of cycles for their corresponding 7-segments output. The stopwatch first start with the centiseconds part (counter 0), as its output goes over 9, the value is reset for count 0 (yet keep counting) and the enable for the next number (counter 1) is on, and the next counter (counter 1) starts counting and when it goes over 9, it is reset and it enables the next counter (counter 2) which is for the seconds. The counter for the second (counter 3) then starts counting and when the number of output goes over 9, it resets and enables the next counter (counter 4) for the seconds, which will reset when the value goes over 6 and then enables the next counter (minutes part). The same process is repeated for the minutes part.

6.0 How the Stopwatch Is Tested



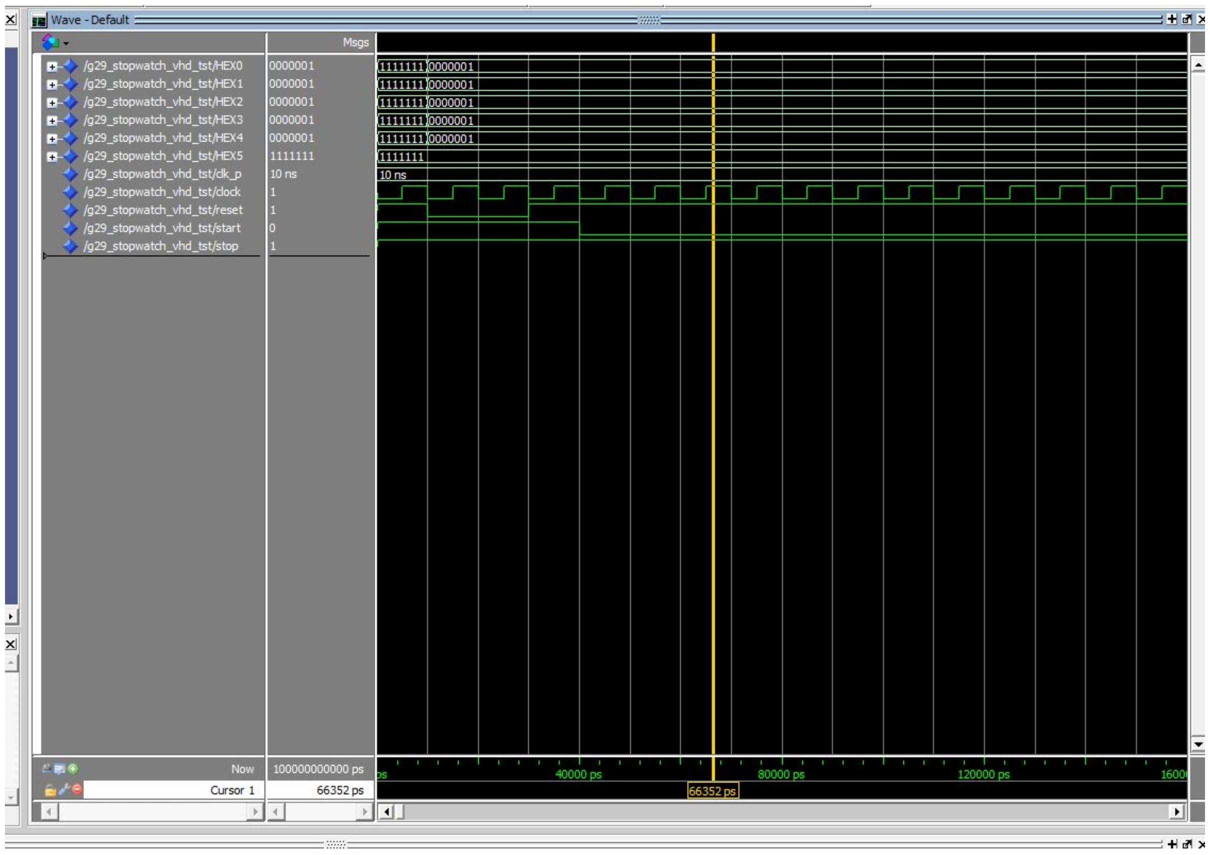
```
    );
END COMPONENT;

constant clk_p : time := 10 ns;

BEGIN
    i1 : g29_stopwatch
    PORT MAP (
-- list connections between master ports and signals
    clock => clock,
    HEX0 => HEX0,
    HEX1 => HEX1,
    HEX2 => HEX2,
    HEX3 => HEX3,
    HEX4 => HEX4,
    HEX5 => HEX5,
    reset => reset,
    start => start,
    stop => stop
    );

clk_prc : PROCESS
BEGIN

    clock <= '0';
    wait for clk_p/2;
    clock <= '1';
    wait for clk_p/2;

END PROCESS clk_prc;

stimu : PROCESS
BEGIN
        --begin with all buttons off
        reset <= '1';
        start <= '1';
        stop <= '1';
        wait for clk_p;
        reset <= '0';
        wait for 2*clk_p;

        --begin count for 110centi-seconds
        reset <= '1';
        wait for clk_p;
        start <= '0';
        wait;

END PROCESS stimu;

END g29_stopwatch_arch;
```

Fig.6.1 This figure shows the test bench for the stopwatch



```
Wave - Default
                                        Msgs
/g29_stopwatch_vhd_tst/HEX0    0001111      0000110 1001100 0100100 0100000 0001111 0000000 0000100 0000001
/g29_stopwatch_vhd_tst/HEX1    0000001      0000001                                         1001111 0010010
/g29_stopwatch_vhd_tst/HEX2    0000001      0000001
/g29_stopwatch_vhd_tst/HEX3    0000001      0000001
/g29_stopwatch_vhd_tst/HEX4    0000001      0000001
/g29_stopwatch_vhd_tst/HEX5    1111111      1111111
/g29_stopwatch_vhd_tst/clk_p   10 ns        10 ns
/g29_stopwatch_vhd_tst/clock   1
/g29_stopwatch_vhd_tst/reset   1
/g29_stopwatch_vhd_tst/start   0
/g29_stopwatch_vhd_tst/stop    1

                        Now    100000000000 ps    20000000000 ps    40000000000 ps    60000000000 ps
                   Cursor 1    37826579288 ps                37826579288 ps
```

Fig.6.2 This figure shows the stimulation for the stopwatch



```
/g29_stopwatch_vhd_tst/HEX0    0000001
/g29_stopwatch_vhd_tst/HEX1    0000001
/g29_stopwatch_vhd_tst/HEX2    0000001
/g29_stopwatch_vhd_tst/HEX3    0000001
/g29_stopwatch_vhd_tst/HEX4    0000001
/g29_stopwatch_vhd_tst/HEX5    1111111
/g29_stopwatch_vhd_tst/clk_p   10 ns
/g29_stopwatch_vhd_tst/clock   1
/g29_stopwatch_vhd_tst/reset   1
/g29_stopwatch_vhd_tst/start   0
/g29_stopwatch_vhd_tst/stop    1

Now       100000000000 ps
Cursor 1  66352 ps
```

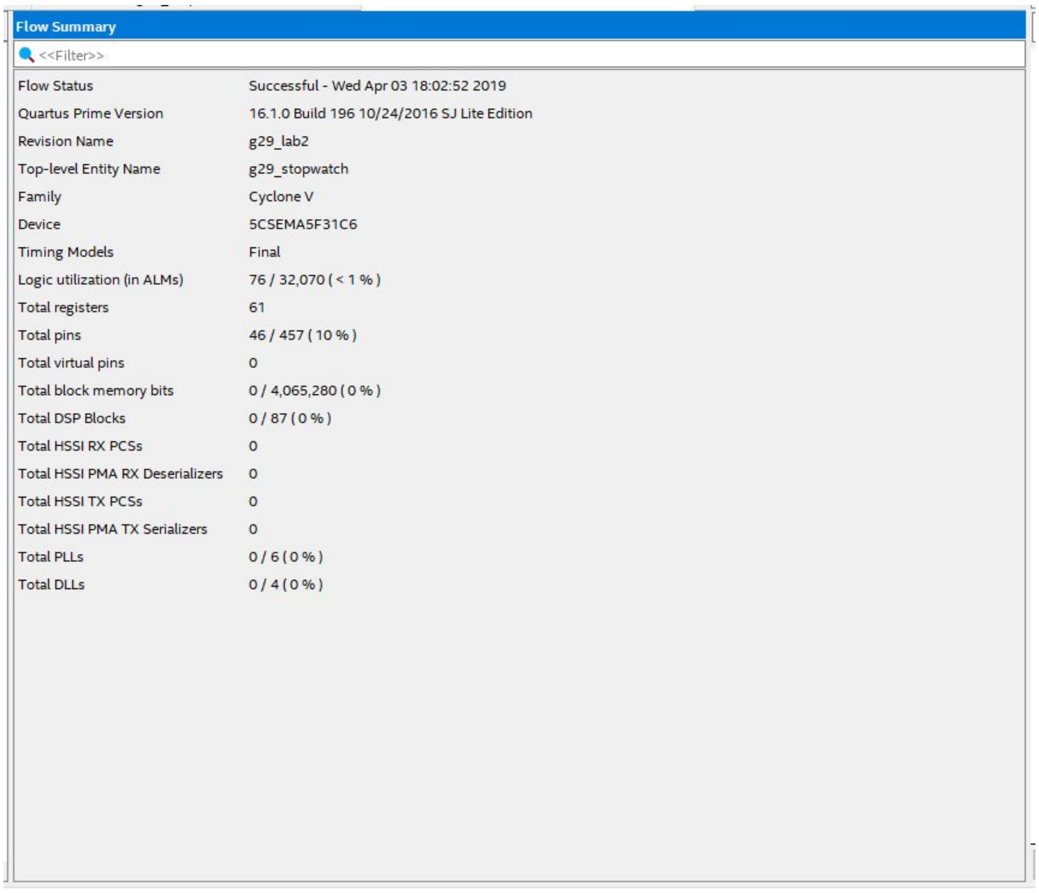
Fig.6.3 This figure shows the zoom in (magnified) stimulation for the stopwatch

For the stimulation of the stopwatch, a test bench is written by having the clock run continuously at a period of 10 ns (the clk_prc process). It also contains a process called “stimu”, which starts by closing all the buttons (including the start, the reset, and the stop). After waiting for a certain period of time, the reset button is on. After another period of time, the reset is closed again and the start button is pushed on.

After carrying out the stimulation in the ModelSim, the stopwatch code is tested using the Altera DE1-SoCboard. The code for stopwatch circuit is first compiled in the Quartus software; once it is done, we map the code to the Altera DE1-SoCboard by assigning the 7-segments declared in the code to their corresponding pin locations, the clock to the pin location for a 50 Mhz clock on the manual, and the start, stop and reset to the pin locations of the push buttons.

The board then serves as the stopwatch and displays the time using the six 7-segments LEDs. And it is proved to be working in the demo.

7.0 Summary of the FPGA Resource Utilization and the RTL Schematic Diagram



Flow Summary

<<Filter>>

Flow Status	Successful - Wed Apr 03 18:02:52 2019
Quartus Prime Version	16.1.0 Build 196 10/24/2016 SJ Lite Edition
Revision Name	g29_lab2
Top-level Entity Name	g29_stopwatch
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	76 / 32,070 (< 1 %)
Total registers	61
Total pins	46 / 457 (10 %)
Total virtual pins	0
Total block memory bits	0 / 4,065,280 (0 %)
Total DSP Blocks	0 / 87 (0 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

Fig.7.1 This figure shows the flow summary for the stopwatch

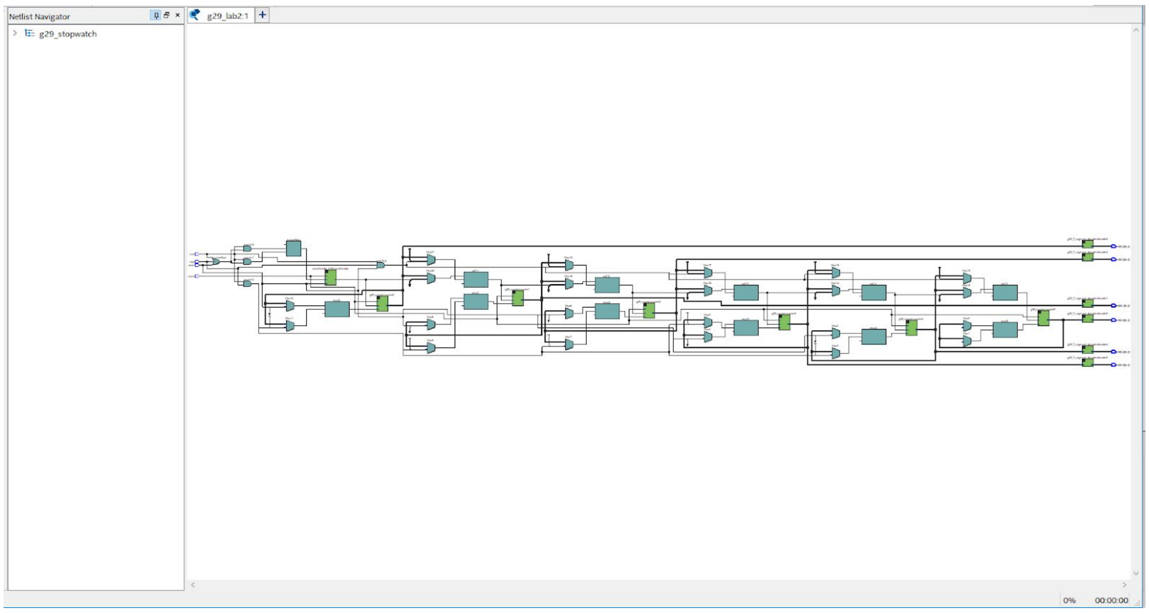


Fig.7.2 This figure shows the overview of the RTL diagram for the stopwatch

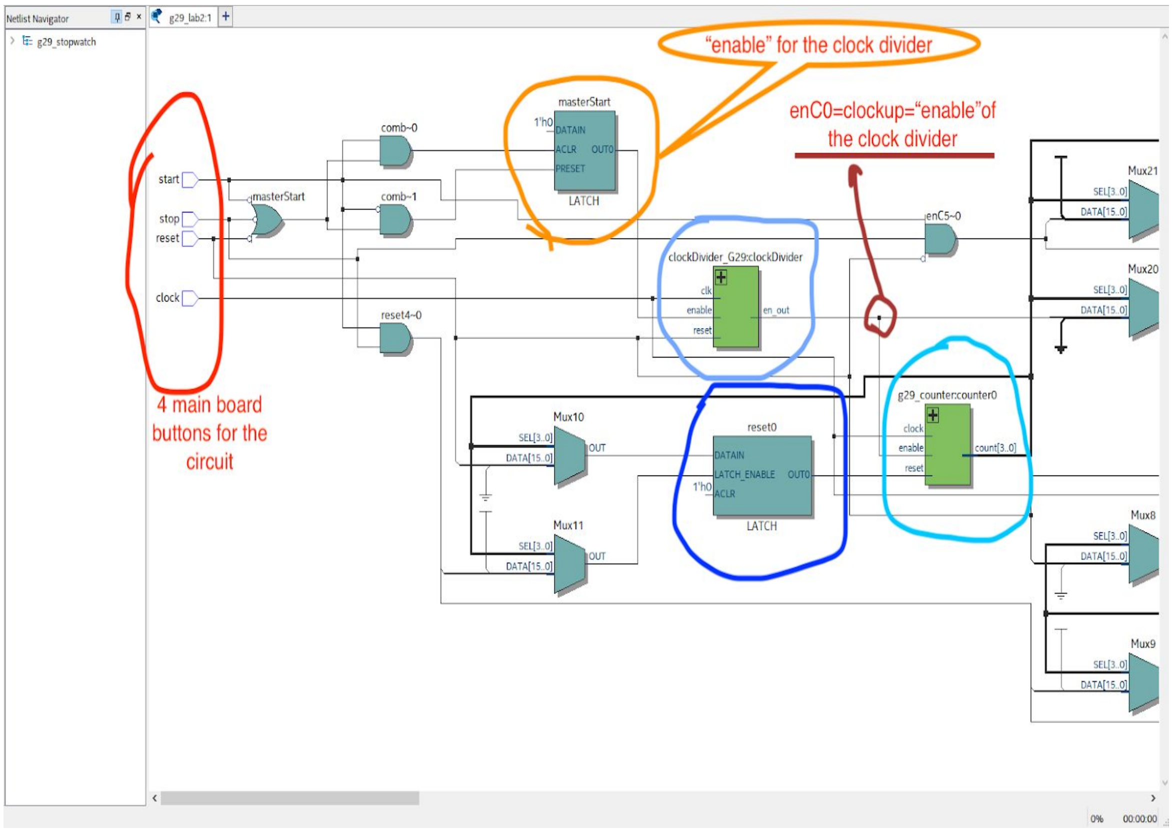


Fig.7.3 This figure shows the partial of the RTL diagram for the stopwatch (1)

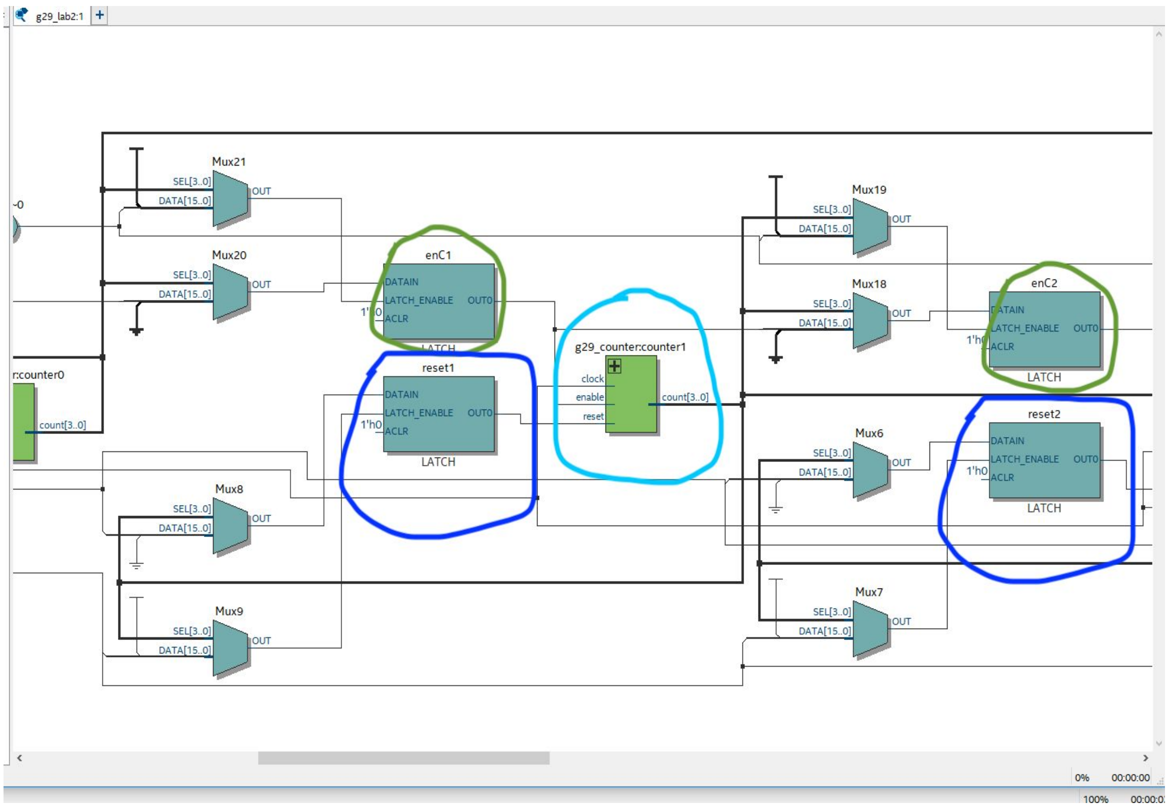


Fig.7.4 This figure shows partial of the RTL diagram for the stopwatch (2)

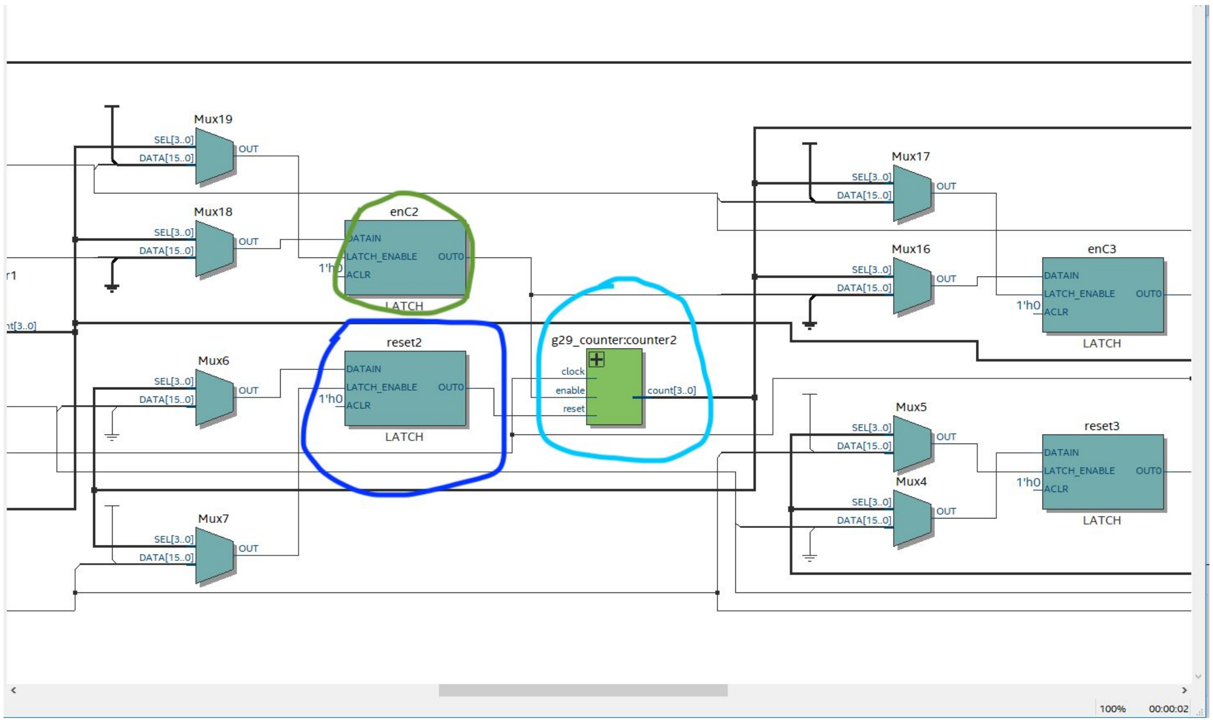


Fig.7.5 This figure shows partial of the RTL diagram for the stopwatch (3)

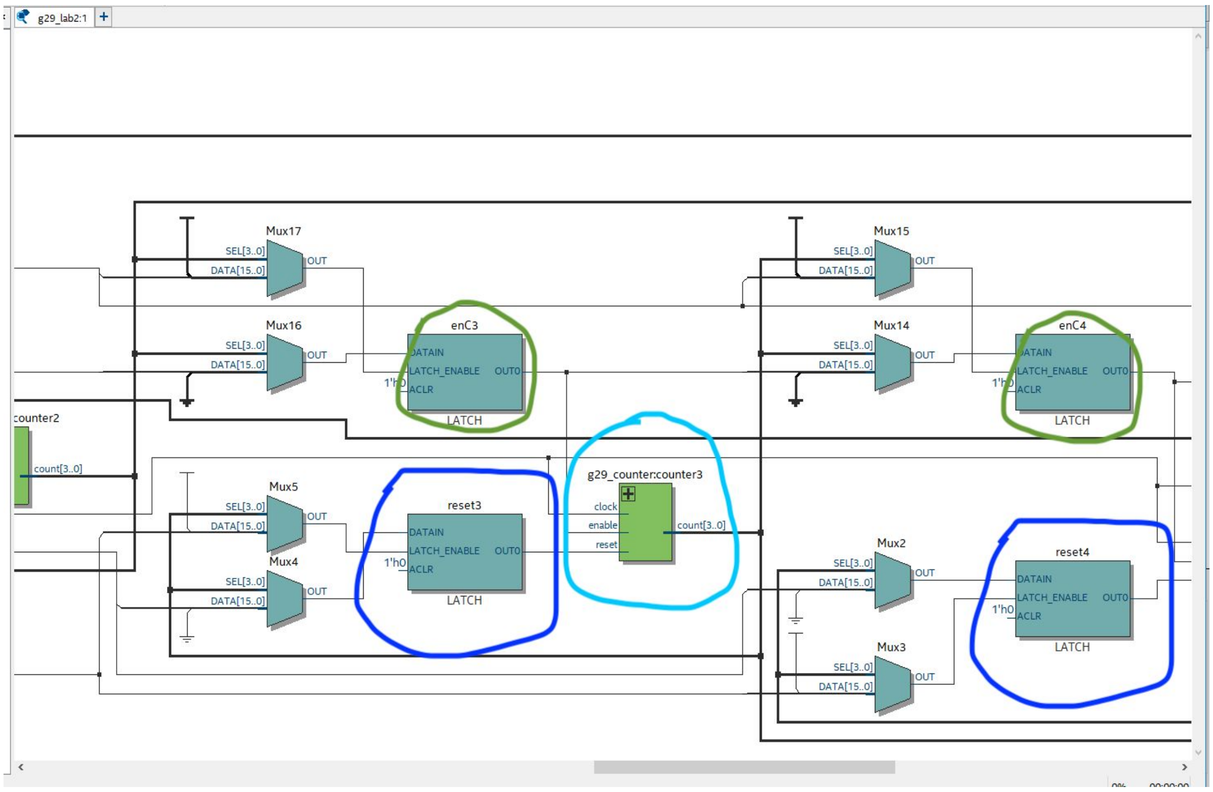


Fig.7.6 This figure shows partial of the RTL diagram for the stopwatch (4)

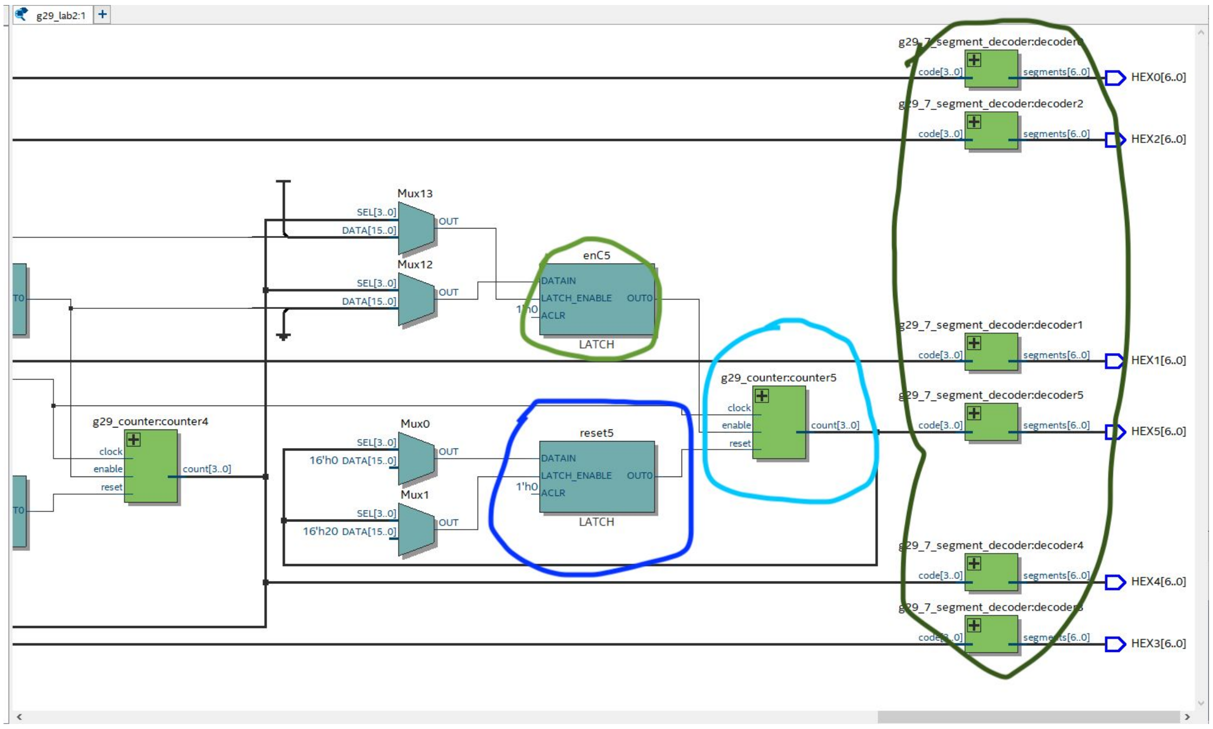
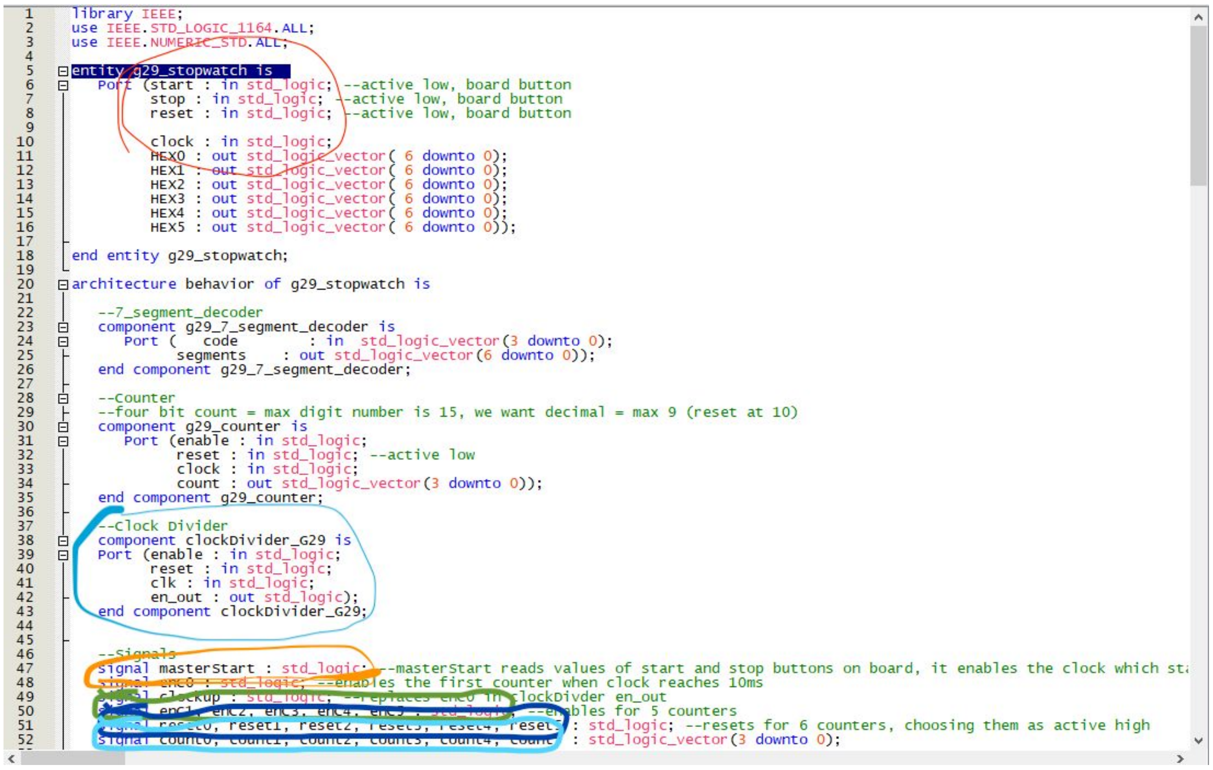
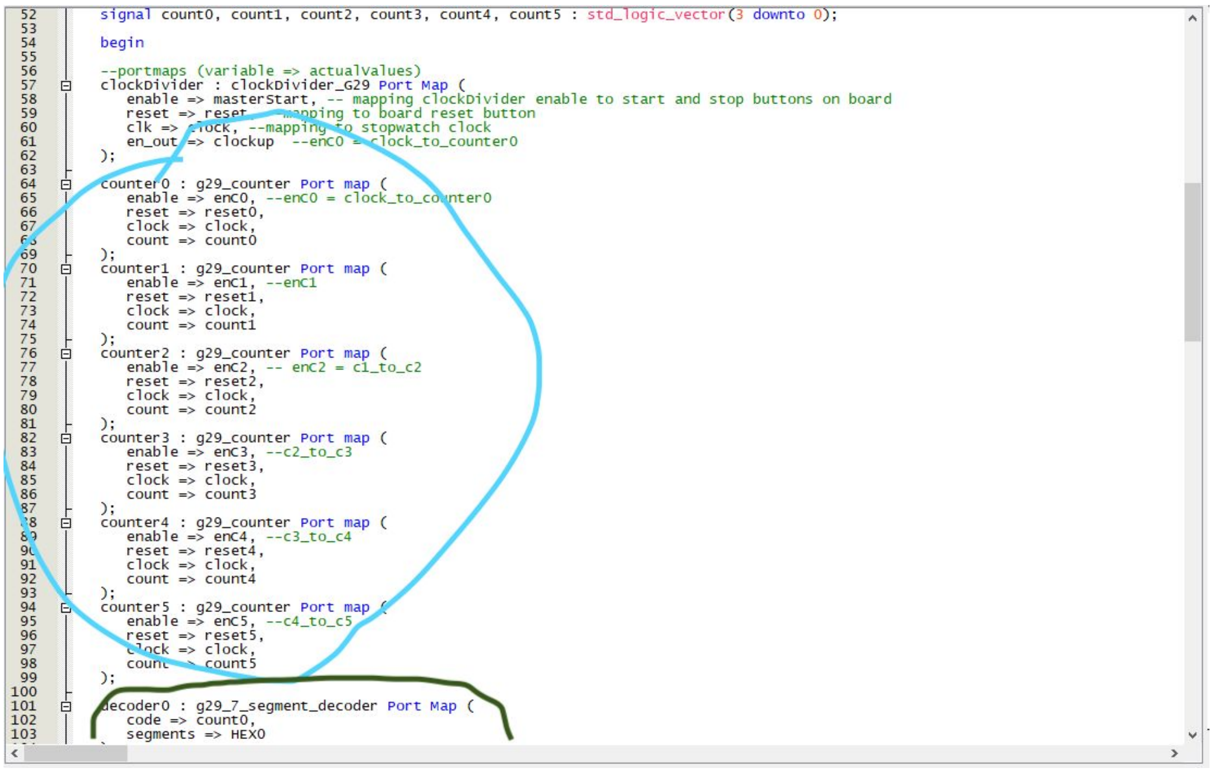


Fig.7.7 This figure shows partial of the RTL diagram for the stopwatch (5)



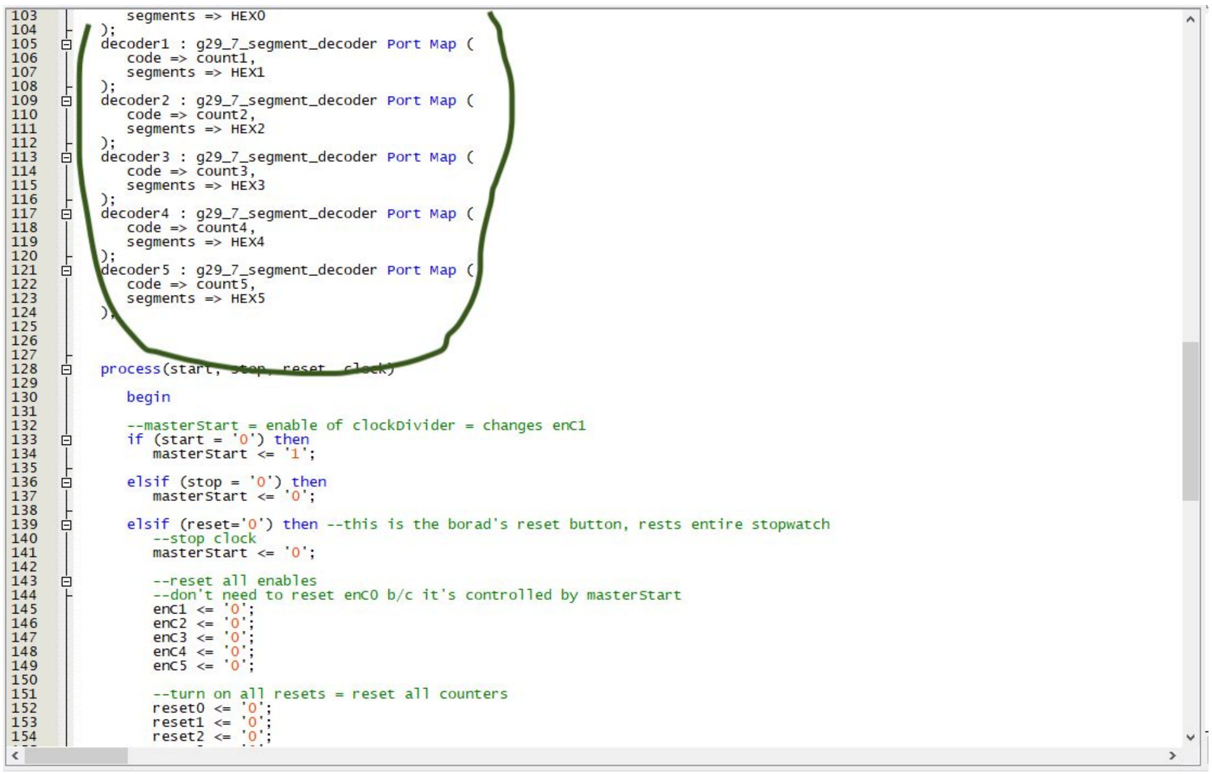
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity g29_stopwatch is
    Port (start : in std_logic; --active low, board button
          stop : in std_logic; --active low, board button
          reset : in std_logic; --active low, board button

          clock : in std_logic;
          HEX0 : out std_logic_vector( 6 downto 0);
          HEX1 : out std_logic_vector( 6 downto 0);
          HEX2 : out std_logic_vector( 6 downto 0);
          HEX3 : out std_logic_vector( 6 downto 0);
          HEX4 : out std_logic_vector( 6 downto 0);
          HEX5 : out std_logic_vector( 6 downto 0));

end entity g29_stopwatch;

architecture behavior of g29_stopwatch is

    --7_segment_decoder
    component g29_7_segment_decoder is
        Port (   code         : in  std_logic_vector(3 downto 0);
              segments     : out std_logic_vector(6 downto 0));
    end component g29_7_segment_decoder;

    --Counter
    --four bit count = max digit number is 15, we want decimal = max 9 (reset at 10)
    component g29_counter is
        Port (enable : in std_logic;
              reset : in std_logic; --active low
              clock : in std_logic;
              count : out std_logic_vector(3 downto 0));
    end component g29_counter;

    --Clock Divider
    component clockDivider_G29 is
    Port (enable : in std_logic;
          reset : in std_logic;
          clk : in std_logic;
          en_out : out std_logic);
    end component clockDivider_G29;


    --Signals
    signal masterStart : std_logic; --masterStart reads values of start and stop buttons on board, it enables the clock which st
    signal enc0 : std_logic; --enables the first counter when clock reaches 10ms
    signal clockup : std_logic; --replaces enc0 in clockDivder en_out
    signal enc1, enc2, enc3, enc4, enc5 : std_logic; --enables for 5 counters
    signal reset0, reset1, reset2, reset3, reset4, reset5 : std_logic; --resets for 6 counters, choosing them as active high
    signal count0, count1, count2, count3, count4, count5 : std_logic_vector(3 downto 0);
```

Fig.7.8 This figure shows partial of the vhdl code for the stopwatch (1)



```vhdl
signal count0, count1, count2, count3, count4, count5 : std_logic_vector(3 downto 0);

begin

--portmaps (variable => actualValues)
clockDivider : clockDivider_G29 Port Map (
   enable => masterStart, -- mapping clockDivider enable to start and stop buttons on board
   reset => reset, --mapping to board reset button
   clk => clock, --mapping to stopwatch clock
   en_out => clockup  --enC0 = clock_to_counter0
);

counter0 : g29_counter Port map (
   enable => enC0, --enC0 = clock_to_counter0
   reset => reset0,
   clock => clock,
   count => count0
);
counter1 : g29_counter Port map (
   enable => enC1, --enC1
   reset => reset1,
   clock => clock,
   count => count1
);
counter2 : g29_counter Port map (
   enable => enC2, -- enC2 = c1_to_c2
   reset => reset2,
   clock => clock,
   count => count2
);
counter3 : g29_counter Port map (
   enable => enC3, --c2_to_c3
   reset => reset3,
   clock => clock,
   count => count3
);
counter4 : g29_counter Port map (
   enable => enC4, --c3_to_c4
   reset => reset4,
   clock => clock,
   count => count4
);
counter5 : g29_counter Port map (
   enable => enC5, --c4_to_c5
   reset => reset5,
   clock => clock,
   count => count5
);

decoder0 : g29_7_segment_decoder Port Map (
   code => count0,
   segments => HEX0
```

Fig.7.9 This figure shows partial of the vhdl code for the stopwatch (2)



```vhdl
   segments => HEX0
);
decoder1 : g29_7_segment_decoder Port Map (
   code => count1,
   segments => HEX1
);
decoder2 : g29_7_segment_decoder Port Map (
   code => count2,
   segments => HEX2
);
decoder3 : g29_7_segment_decoder Port Map (
   code => count3,
   segments => HEX3
);
decoder4 : g29_7_segment_decoder Port Map (
   code => count4,
   segments => HEX4
);
decoder5 : g29_7_segment_decoder Port Map (
   code => count5,
   segments => HEX5
);


process(start, stop, reset, clock)

   begin

   --masterStart = enable of clockDivider = changes enC1
   if (start = '0') then
      masterStart <= '1';

   elsif (stop = '0') then
      masterStart <= '0';

   elsif (reset='0') then --this is the borad's reset button, rests entire stopwatch
      --stop clock
      masterStart <= '0';

      --reset all enables
      --don't need to reset enC0 b/c it's controlled by masterStart
      enC1 <= '0';
      enC2 <= '0';
      enC3 <= '0';
      enC4 <= '0';
      enC5 <= '0';

      --turn on all resets = reset all counters
      reset0 <= '0';
      reset1 <= '0';
      reset2 <= '0';
```

Fig.7.10 This figure shows partial of the vhdl code for the stopwatch (3)

Same colours indicate corresponding parts in the code and the schematic diagram.