Final Project: Digital Clock

EENG 2910 Project III: Digital System Design

Due: 04/30/2014

Team Members:
Shabuktagin Photon Khan
Youssif Mahjoub
Zhenhai Lou

University of North Texas
Department of Electrical Engineering

Contents

Abstract

The Final Project gives us the opportunity to work on Xilinx ISE, basy2board and VHDL to work on a brief report for it. The objective of the project was to make a digital clock. The project will not be focused on alarm.

Introduction

The digital system is an interconnection of digital modules designed to perform specific functions. VHDL stands for very high-speed integrated circuit (VHSIC) hardware description language. It was originally sponsored by the U.S department of Defense and later transferred to the IEEE (Institute of Electrical and Electronics Engineers). VHDL is used for describing and modeling a digital system at various levels and is a complex language.

VHDL model is a complete VHDL component description consists of an entity and an architecture. The entity describes the component's interface. Architecture defines the component's function and architectural description is the structural, behavioral. In port modes there are four default modes in VHDL in, out, inout and buffer.

We worked with the BASYS2 Spartan 3E Kit and Xilinx ISE to perform the objective. The Basy2 board is a circuit design and implementation platform that anyone can use to gain experience building real digital circuits. The FPGA on the basys2 board are programmed by the user before it can perform any functions. The Basys2 board includes several input devices, output devices, and data ports, allowing many designs to be implemented without the need for any other components.

Report

This report has been divided to different sections since to make the digital clock we used PORT Map to glue up several VHDL files to work together as one. Figure 1 shows the RTL schematic diagram of the Digital Clock VHDL file.

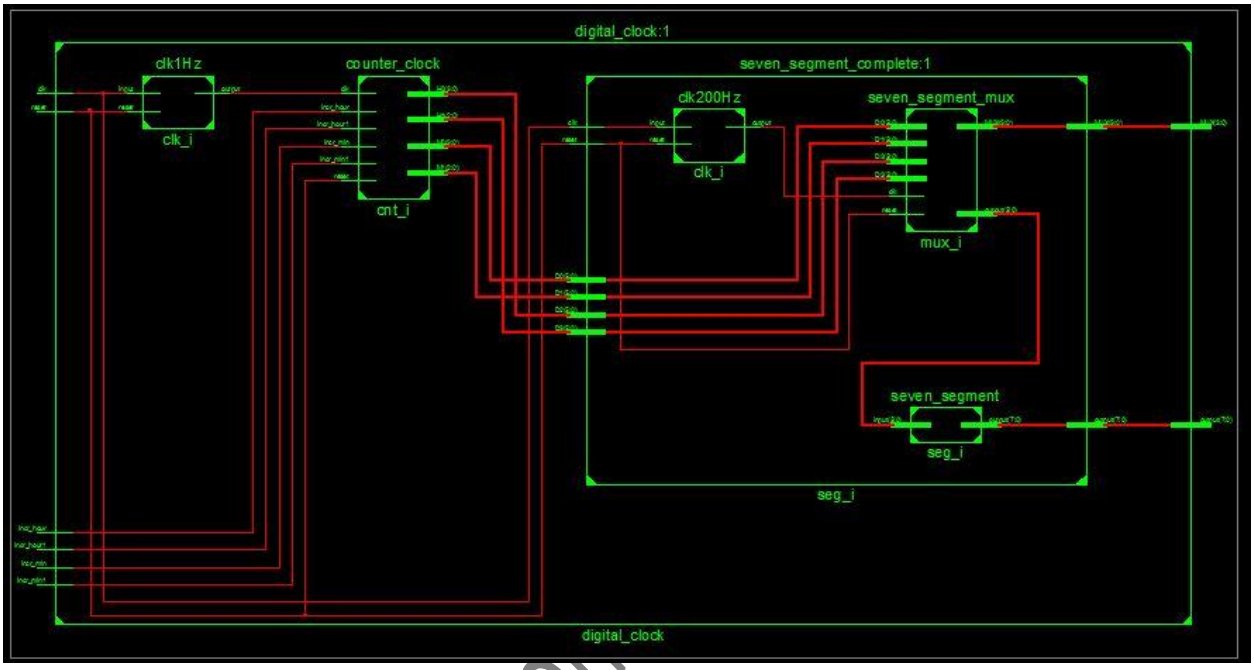

Figure 1: RTL Schematic Diagram of the Digital Clock VHDL file

The digital_clock block has 6 inputs clock clk, reset, incr_min, incr_min1, incr_hour and incr_hour1. It has 2 outputs one is "mux" and the other "output." Inside the whole block there is 6 blocks and each of them are four different functions. The clk1Hz block makes the VHDL file works according to the time. The counter_clock block is where the relationship between hours, minutes and seconds is described. In the seven segment_block_complete block there is 3 more blocks. The clk200Hz block makes the display of the segment display refresh after few milliseconds. The seven_segment_mux helps to decide whether number should display in AN3 or AN2 or AN1 or AN0 as shown in Figure 2. The seven_segment block decides what to display in 7 segment display output.

Figure 2: Mux Decision

Oscillators

The Basys2 board includes silicon oscillator which produces oscillation at 50MHz, which is the primary oscillator which is connected to global clock input pins at pin B8. Figure 3 shows the B8 pin.

Figure 3: B8 Pin

Clk1Hz

The basys2board has a silicon oscillator that produces 25 MHz, 50MHz or 100MHz based on the position of the clock select jumper at JP4. The primary and secondary pins at pin B8 and pin M6 respectively. To make the clock with respect to seconds, Frequency divider rule was applied

Formula:
Scaling Factor=Frequency of the board/ Required Frequency
As we know, time= 1/frequency. Therefore we need 1Hz.
Oscillator is 50 MHz and the required is 1Hz
50M/1=50M cycles
Only considering the rising edge. Therefore we needed 50M/2= 25M cycles
1Hz clock: is used as input signal to the counter (we will count every second).
A clock signal while maintaining the high than in low; for this particular case, 25000000 cycles
25000000 cycles high and low. Since we started counting at zero, the upper limit is 25000000-1.
The signal reset is used to reset the counter. The block diagram of the clk1Hz is shown in Figure 4.

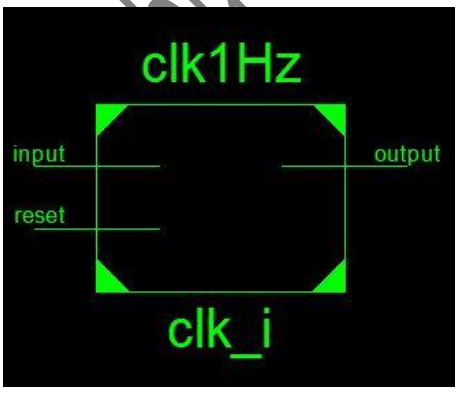

Figure 4: Clk1Hz

The VHDL code for clk1Hz is shown below

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

entity clk1Hz is
    Port (
        input: in  STD_LOGIC;
        reset  : in  STD_LOGIC;
        output : out STD_LOGIC
    );
end clk1Hz;

architecture Behavioral of clk1Hz is
    signal temporal: STD_LOGIC;
    signal counter: integer range 0 to 24999999 := 0;
begin
   frequency_divider: process (reset, input) begin
        if (reset = '1') then
            temporal <= '0';
            counter <= 0;
        elsif rising_edge(input) then
            if (counter = 24999999) then
                temporal <= NOT(temporal);
                counter <= 0;
            else
                counter <= counter+1;
            end if;
        end if;
    end process;

    output <= temporal;
end Behavioral;
```

Counter Clock

Figure 5 shows the block diagram of the counter clock. It has 6 inputs clk, incr_hour, incr_hour1, incr_min, incr_min1 and reset. It has 4 outputs H0, H1, M0 and M1.

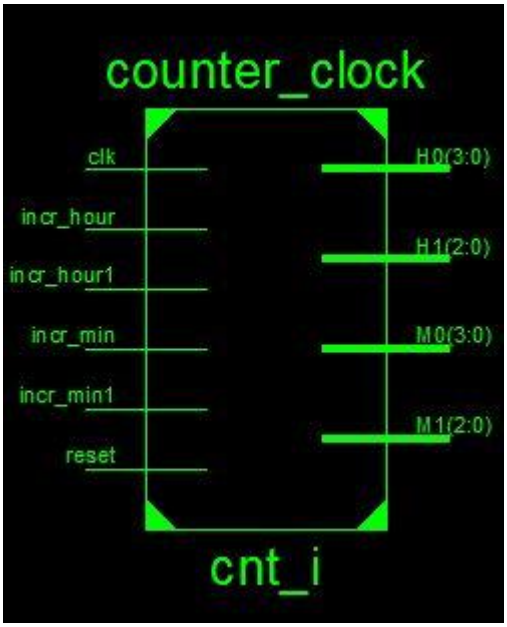

Figure 5: Counter Clock

The input of the counter_clock is the output of the clk1Hz block so that it can use the logic of 1 second. incr_hour, incr_hour1, incr_min, incr_min1 are the inputs which will be used as a switch to increment the value of the each digits. The designated digits are shown in figure 6.

Figure 6: Designated Digits

The outputs of the counter_clock are M0, M1, H0 and H1.

The VHDL code for Counter_Clock is shown below with box explanation
The words in italic are not in the code

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

```
entity counter_clock is
        PORT (
                clk  : IN  STD_LOGIC;
                reset: IN  STD_LOGIC;
                incr_hour : IN  STD_LOGIC;
                incr_min : IN  STD_LOGIC;
                incr_hour1 : IN  STD_LOGIC;
                incr_min1 : IN  STD_LOGIC;
                H1   : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
                H0   : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
                M1   : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
                M0   : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
        );
end counter_clock;
```

M0 and H0 are 3 downto 0 because in terms of binary it is 4 binary numbers starting from "0000." Therefore 2 to the power 4 is 16. Hence we know M0 and H0 are the number from 0 to 15. M1 and H1 are 2 downto 0 because in terms of binary it is 3 binary numbers starting from "000." Therefore 2 to the power 3 is 8. Hence we know M1 and H1 are the number from 0 to 7.

```
architecture Behavioral of counter_clock is
  signal mm: UNSIGNED(5 downto 0) := "000000" ;
        signal mm1: UNSIGNED(2 downto 0) := "000" ;
        signal mm0: UNSIGNED(3 downto 0) := "0000";
        signal hh1: UNSIGNED(2 downto 0) := "000" ;
        signal hh0: UNSIGNED(3 downto 0) := "0000";
```

Signal mm, mm0, mm1, hh0, hh1 are produced inside the counter_clock block. This signals uses the clock of 1Hz

```
begin
        clock: process (clk, reset) begin
                if reset = '1' then
                        hh1 <= "000" ;
                        hh0 <= "0000";
                        mm1 <= "000" ;
                        mm0 <= "0000";
                        mm<= "000000";
```

When the reset will be turned on or in other words when it will be 1 it will turn the value of the hh1, hh0, mm1, mm0 and mm to 0.

```
                elsif incr_hour1 = '1' then
                        hh1 <= hh1 +1;
                       if hh1 = 2 then
                         hh1 <= "000";
                       end if;
                elsif incr_hour = '1' then
                        hh0 <= hh0 +1;
                        if hh0 = 9 then
                           hh0<="0000";
                        end if;
                elsif incr_min1 = '1' then
                       mm1 <= mm1 + 1;
                       if mm1 = 5 then
                          mm1 <= "000" ;
                      end if;
               elsif incr_min = '1' then
                      mm0 <= mm0 + 1;
                      if mm0 = 9 then
                         mm0 <= "0000";
                     end if;
                elsif hh1 = 2 AND hh0 > 3 then
                     hh1 <= "000";
```

12:56 *<<With respect to this image, if incr_min is high then it will increment the value of 6 to 7. If incr_min1 is high it will increment the value of 5 to 6. If incr_hour is high, it will increment the value of 2 to 3. If the incr_hour1 is high, it will increment the value of 1 to 2. Incr_min will increment the 0 to 9 then it again starts from 0. Incr_min1 will increment from 0 to 5 then it again starts from 0. Incr_hour will increment from 0 to 9 then it again starts from 0. Incr_hour1*

will increment from 0 to 2 then it will again start from 0. If the value of incr_hour is more than 3 and the value of incr_hour1 is 2 then it will turn both the incr_hour as well as incr_hour1 to 0.

```
        elsif rising_edge(clk) then
            mm<= mm+1;
            if mm = 59 then
            mm0 <= mm0 + 1;
            mm<="000000";
            end if;
            if mm0 = 9 then
                mm1 <= mm1 + 1;
                mm0 <= "0000";
            end if;

            if mm1 = 5 AND mm0 = 9 then
                hh0 <= hh0 + 1;
                mm1 <= "000";
            end if;
            if hh0 = 9 then
                hh1 <= hh1 + 1;
                hh0 <= "0000";
            end if;

            if hh1 = 2 AND hh0 = 3 AND mm1 = 5 AND mm0 = 9 then
                hh1 <= "000";
                hh0 <= "0000";
            end if;
        end if;
end process;
```


When the clock has the rising edge it will make mm signal increment by 1. Therefore, the mm signal acts like seconds. When the mm signal reaches 59 seconds, the mm signal becomes 0 and it increments the value of mm0 by 1. When the mm0 reaches 9 it increments the mm1 by 1 and mm0 itself becomes zero. When the mm1 reaches 5 and mm0 reaches 9 it increments the values of hh0 by 1 and both mm1 and mm0 becomes 0. When hh0 becomes 9 it increments the value of hh1 by 1. When hh1 becomes 2, hh0 becomes 3 , mm1 becomes 5 and mm0 becomes 9, all the value resets.

```
        H1 <= STD_LOGIC_VECTOR(hh1);
        H0 <= STD_LOGIC_VECTOR(hh0);
        M1 <= STD_LOGIC_VECTOR(mm1);
        M0 <= STD_LOGIC_VECTOR(mm0);
end Behavioral;
```

The signal mm0,mm1,hh0 and hh1 are assigned to the variables M0,M1,H0,H1.
So in brief, Counter clock is the component which is responsible for counting from 0 to 23 for the hours and 0 to 59 for minutes.The reset is responsible for returning the counter to zero
The counter mm0 account 0-9, incremented by one each time it receives the clock signal.
The counter mm1 counts from 0 to 5, increases by one each time mm0 is nine.
The counter hh0 account 0-9, incremented by one each time mm1 is 5 and mm0 is 9 (hourly).
The counter hh0 account 0-2, incremented by one each time hh0 is nine.
Counters hour hh0 and hh1 , return to zero when the time is 23:59.

Clk200Hz

For each of the four digits to appear bright and continuously illuminated at 7 segment display of basys2 board, all four digits should be driven once every 1 to 16ms. For example, in a 60Hz refresh scheme, the entire display would be refreshed once every 16ms, and each digit would be illuminated for ¼ of the refresh cycle, or 4ms. To illuminate all AN1, AN2, AN3 and AN4 we took 200Hz to make digits illuminate constantly. This circuit drives the anode signals and corresponding cathode patterns of each digit in a repeating, continuous succession, at an update rate that is faster than the human eye response.

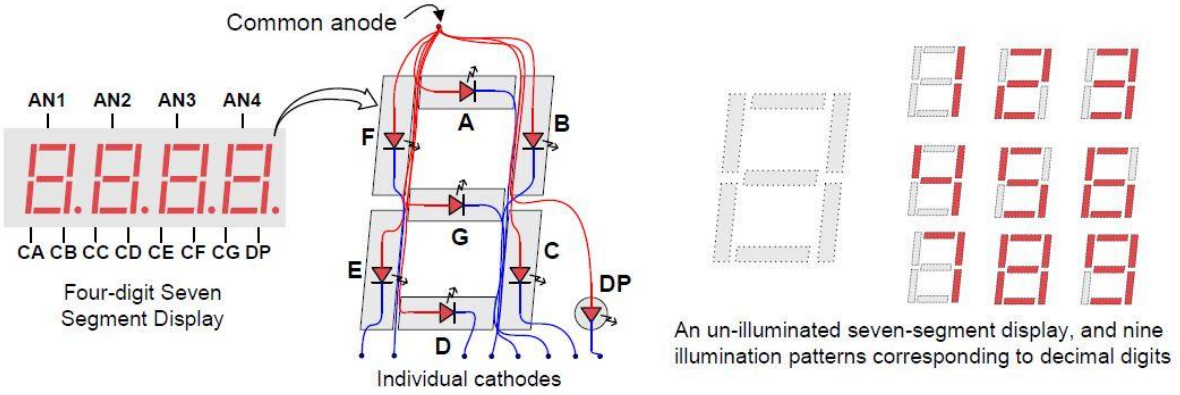

Figure 8: To illuminate all AN1, AN2, AN3 and AN4

Calculations:

The calculation process is similar to Clk1Hz using frequency divider rule.

Oscillator is 50 MHz and we need output of 200Hz

50M/200 = 250,000 cycles

Since we are considering only the rising edge

We would be needing 250,000/2= 125,000 cycles

Figure 9 shows the block diagram of the Clk200Hz

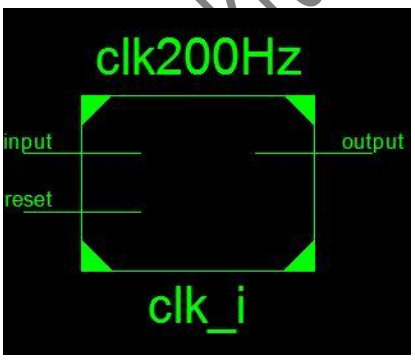

Figure 9: Clk200Hz

The VHDL code for clk200Hz is shown below

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

entity clk200Hz is
    Port (
        input: in  STD_LOGIC;
        reset  : in  STD_LOGIC;
        output : out STD_LOGIC
    );
end clk200Hz;

architecture Behavioral of clk200Hz is
    signal temporal: STD_LOGIC;
    signal counter: integer range 0 to 124999 := 0;
begin
    frequency_divider: process (reset, input) begin
        if (reset = '1') then
            temporal <= '0';
            counter <= 0;
        elsif rising_edge(input) then
            if (counter = 124999) then
                temporal <= NOT(temporal);
                counter <= 0;
            else
                counter <= counter+1;
            end if;
        end if;
    end process;

    output <= temporal;
end Behavioral;
```

Seven Segment Mux

Figure 10 shows the seven segment mux.

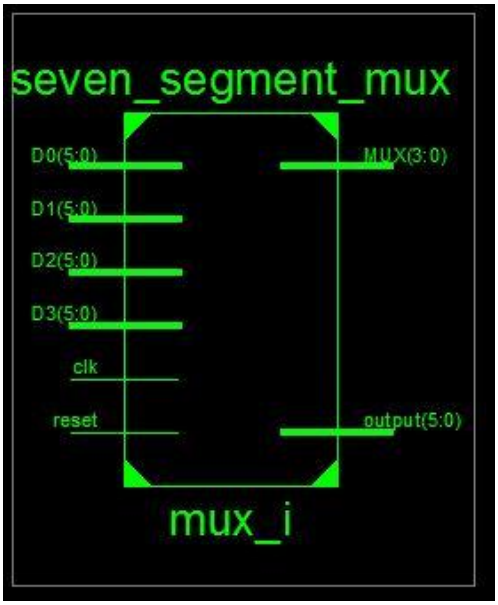

Figure 10: Seven Segment Mux

In this block, Multiplexor concept is used. A multiplexer or data selector is a logic circuit that accepts multiple inputs and only allows one to reach the exit.

The VHDL code for the Multiplexor is shown below with explanation

The writings in Italic are not in the code

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

entity seven_segment_mux is
    PORT (
        clk   : IN  STD_LOGIC;
        reset : IN  STD_LOGIC;
        D0    : IN  STD_LOGIC_VECTOR(5 downto 0);
        D1    : IN  STD_LOGIC_VECTOR(5 downto 0);
        D2    : IN  STD_LOGIC_VECTOR(5 downto 0);
        D3    : IN  STD_LOGIC_VECTOR(5 downto 0);
        output: OUT STD_LOGIC_VECTOR(5 downto 0);
        MUX   : OUT STD_LOGIC_VECTOR(3 downto 0)
    );
end seven_segment_mux;
```

D0 input uses the output of H1, D1 input uses the output of H0, D2 uses the output of M1 and D3 uses the output of M0. Its clk input uses the output of clk200Hz.

```
architecture Behavioral of seven_segment_mux is
    type states is (rst, v0, v1, v2, v3);
    signal state : states;
begin
    displays: process (reset, clk) begin
        if (reset = '1') then
            state <= rst;
            MUX <= x"F";
            output <= "111111";
        elsif rising_edge(clk) then
            case state is
                when v0 =>
                    output <= D3;
                    MUX <= "1110";
                    state <= v1;
                when v1 =>
                    output <= D2;
                    MUX <= "1101";
                    state <= v2;
                when v2 =>
                    output <= D1;
                    MUX <= "1011";
                    state <= v3;
                when others =>
                    output <= D0;
                    MUX <= "0111";
                    state <= v0;
            end case;
        end if;
    end process;
end Behavioral;
```


When the MUX <= "1110" it turns the AN0 on and the value D3 is assigned to MUX <= "1110"
When the MUX <= "1101" it turns the AN1 on and the value D2 is assigned to MUX <= "1101"
When the MUX <= "1011" it turns the AN2 on and the value D1 is assigned to MUX <= "1011"
When the MUX <= "0111" it turns the AN3 on and the value D0 is assigned to MUX <= "0111"
With respect to Figure 2.

Seven Segment Display

Figure 11 shows the block diagram of the seven segment.

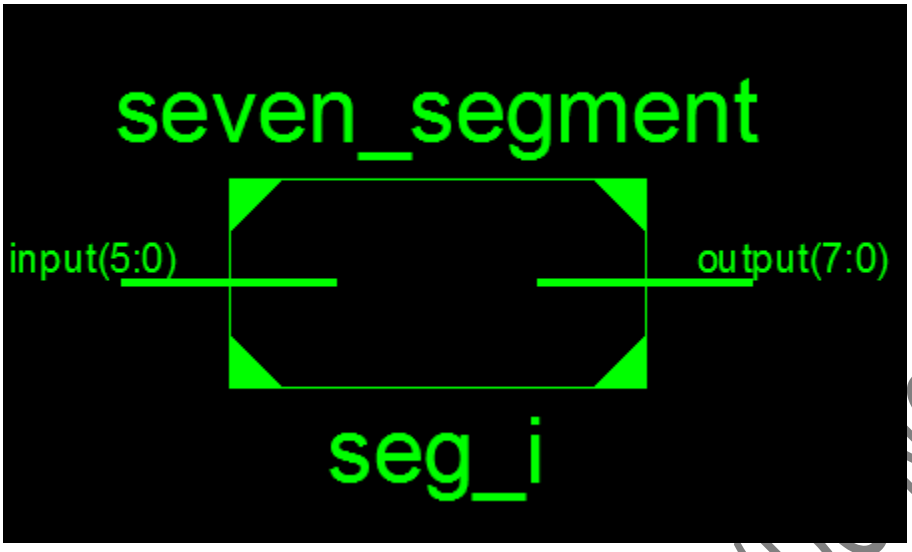

Figure 11: Seven Segment Display

The ideas behind the seven segment display are shown in Figure 12

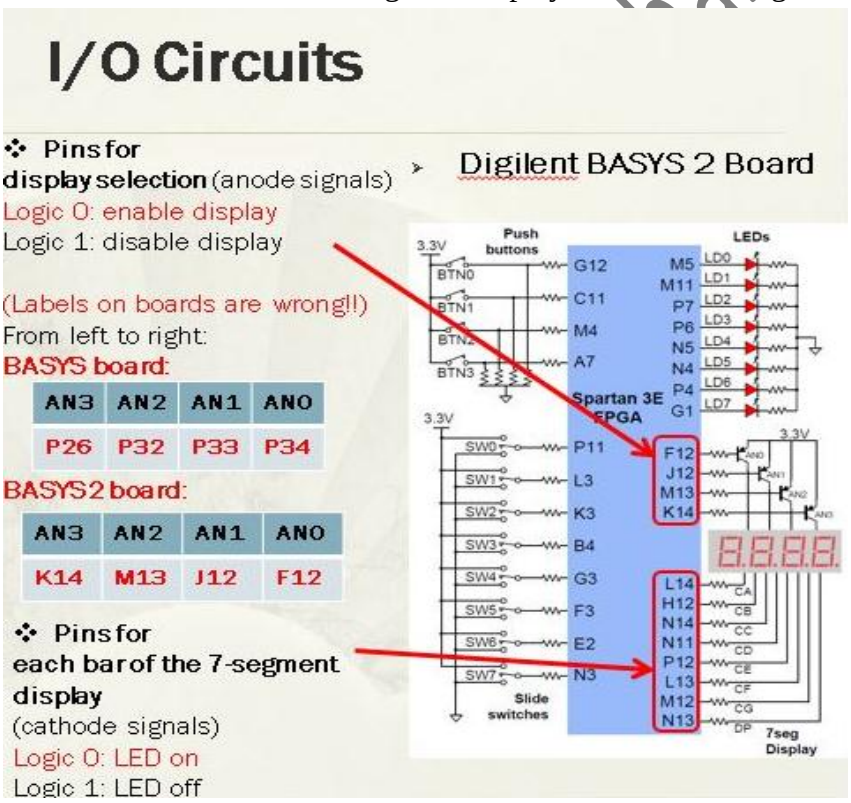

Figure 12: Seven Segment Logic

The VHDL code for the Seven Segment is shown below with explanation

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

entity seven_segment is
   PORT (
      input: IN  STD_LOGIC_VECTOR(5 downto 0);
      output : OUT STD_LOGIC_VECTOR(7 downto 0)
   );
end seven_segment;

architecture Behavioral of seven_segment is
begin
   display: process (input) begin
      case input is
         when "000000" =>  output <= x"C0"; -- 0
         when "000001" =>  output <= x"F9"; -- 1
         when "000010" =>  output <= x"A4"; -- 2
         when "000011" =>  output <= x"B0"; -- 3
         when "000100" =>  output <= x"99"; -- 4
         when "000101" =>  output <= x"92"; -- 5
         when "000110" =>  output <= x"82"; -- 6
         when "000111" =>  output <= x"F8"; -- 7
         when "001000" =>  output <= x"80"; -- 8
         when "001001" =>  output <= x"98"; -- 9
         when "001010" =>  output <= x"88"; -- A
         when "001011" =>  output <= x"83"; -- B
         when "001100" =>  output <= x"C6"; -- C
         when "001101" =>  output <= x"A1"; -- D
         when "001110" =>  output <= x"86"; -- E
         when "001111" =>  output <= x"8E"; -- F
         when "010000" =>  output <= x"90"; -- G
         when "010001" =>  output <= x"89"; -- H
         when "010010" =>  output <= x"E6"; -- I
         when "010011" =>  output <= x"E1"; -- J
         when "010100" =>  output <= x"85"; -- K
         when "010101" =>  output <= x"C7"; -- L
         when "010110" =>  output <= x"C8"; -- M
         when "010111" =>  output <= x"AB"; -- N
```

```
        when "011000" =>  output <= x"C0"; -- O
        when "011001" =>  output <= x"8C"; -- P
        when "011010" =>  output <= x"98"; -- Q
        when "011011" =>  output <= x"AF"; -- R
        when "011100" =>  output <= x"92"; -- S
        when "011101" =>  output <= x"87"; -- T
        when "011110" =>  output <= x"E3"; -- U
        when "011111" =>  output <= x"C1"; -- V
        when "100000" =>  output <= x"E2"; -- W
        when "100001" =>  output <= x"8F"; -- X
        when "100010" =>  output <= x"91"; -- Y
        when "100011" =>  output <= x"B6"; -- Z
        when "100100" =>  output <= x"BF"; -- -
        when "100101" =>  output <= x"F7"; -- _
        when "100110" =>  output <= x"7F"; -- .
        when others   =>  output <= x"FF"; -- None
      end case;
   end process;
end Behavioral;
```

Explanation: A seven-segment display consists of seven light-emitting diodes (LEDs) arranged in a pattern as shown in Figure 13, and some include an eighth LED for the decimal point. Two types of displays according to their electrical connections: common anode and common cathode . In common anode, a 0 is used for power and 1 for OFF (negative logic).
Otherwise, one uses a common cathode LEDs to turn on and zero to turn off (positive logic).

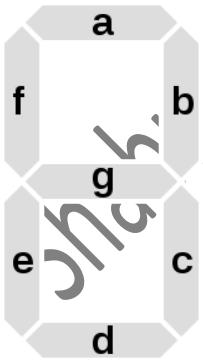

Figure 13: LEDs Arrangement

Having 7 LEDs can generate a total of 128 combinations, although not all of them make up characters. The symbol table and the following characters are encoded is 10 digits (0 through 9), 26 letters (A through Z), and characters _ , - , and . . these 37 symbols to represent 6 bits are needed (five to 32 bits can represent values only). This process of assigning a number to each character is called encoding. Finally, to display the digit on a seven-segment display is necessary to decode the numerical value according to the pattern indicated LEDs. This decoding is performed assuming that the display is common anode (active zero, one off).
In brief, initially has a character which is assigned a numerical value (coding) to deal with the digitally subsequently transforms the pattern corresponding LEDs (decoding).

Hardware description

The hardware description is responsible for determining which LEDs are lit based on the input value of six bits for each of the 37 different symbols and a case to cover any possible value. The bit order shown in Table 1.

Table 1: Bit corresponding to each segment.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
dp	g	F	and	d	c	b	to

Figure 14: Hardware Description

When "000000" => output <= x "C0"; - 0

So it assigns the hexadecimal value "C0" to exit. That translates to the following binary value: "11000000", implying that the segments corresponding to the decimal point and led g are high and the other low. For common anode connection which is handled at the Basys2 card, the LEDs *dp* and *g* are off and the other led are on.

For example:

For “0” number, the binary number is “000000” which is 11000000

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	1	0	0	0	0	0	0
dp	g	f	e	d	c	b	a

Binary to Hexadecimal conversion

1100 0000

C O

Therefore “CO”

Ports and Port Map

Basically, the steps for the declaration of a component relative to the VHDL module are:

- It replaces the keyword ENTITY by keyword COMPONENT.
- Component ports identically as they appear in the module are listed.
- It ends with the statement END COMPONENT

PORT MAP: is responsible for instantiating a component and determine its connection with other components

Tag: Chain CLK_I becomes the name of the component instance. This is necessary because each instance must have a name to be able to differentiate from each other. The label must be a unique name to avoid conflicts existential synthesis tools

Component being instantiated: It Specifies the name of the component being instantiated (in this case our frequency divider). To this point, there must be a component declaration somewhere in the main module.

Keyword PORT MAP: It is indicating that the component will instantiate and assign the input and output signals to the instance.

Port List: Here the signals that are required as input instance and the signals are assigned to receive the output value is sent.

VHDL Code for Seven Segment Complete

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

entity seven_segment_complete is
   PORT (
      clk : IN  STD_LOGIC;
      reset : IN  STD_LOGIC;
      D0    : IN  STD_LOGIC_VECTOR(5 downto 0);
      D1    : IN  STD_LOGIC_VECTOR(5 downto 0);
      D2    : IN  STD_LOGIC_VECTOR(5 downto 0);
      D3    : IN  STD_LOGIC_VECTOR(5 downto 0);
      output: OUT STD_LOGIC_VECTOR(7 downto 0);
      MUX   : OUT STD_LOGIC_VECTOR(3 downto 0)
   );
```

```
end seven_segment_complete;

architecture Behavioral of seven_segment_complete is
        COMPONENT clk200Hz IS
                PORT (
        input: IN  STD_LOGIC;
        reset  : IN  STD_LOGIC;
        output : OUT STD_LOGIC
    );
  END COMPONENT;

  COMPONENT seven_segment IS
                PORT (
        input: IN  STD_LOGIC_VECTOR(5 downto 0);
        output : OUT STD_LOGIC_VECTOR(7 downto 0)
    );
  END COMPONENT;

  COMPONENT seven_segment_mux IS
    PORT (
      clk   : IN  STD_LOGIC;
      reset : IN  STD_LOGIC;
      D0    : IN  STD_LOGIC_VECTOR(5 downto 0);
      D1    : IN  STD_LOGIC_VECTOR(5 downto 0);
      D2    : IN  STD_LOGIC_VECTOR(5 downto 0);
      D3    : IN  STD_LOGIC_VECTOR(5 downto 0);
      output: OUT STD_LOGIC_VECTOR(5 downto 0);
      MUX   : OUT STD_LOGIC_VECTOR(3 downto 0)
    );
  END COMPONENT;

  signal clk_out : STD_LOGIC := '0';
  signal digit  : STD_LOGIC_VECTOR(5 downto 0);
begin
  clk_i: clk200Hz PORT MAP(
    clk, reset, clk_out
  );

  mux_i: seven_segment_mux PORT MAP(
    clk_out, reset, D0, D1, D2, d3, digit, MUX
```

```
    );

    seg_i: seven_segment PORT MAP(
        digit, output
    );
end Behavioral;
```

VHDL Code for Digital Clock

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

entity digital_clock is
        PORT(
                clk   : IN  STD_LOGIC;
                reset : IN  STD_LOGIC;
                incr_hour : IN  STD_LOGIC;
    incr_min : IN  STD_LOGIC;
                incr_hour1 : IN  STD_LOGIC;
    incr_min1 : IN  STD_LOGIC;
                output: OUT STD_LOGIC_VECTOR(7 downto 0);
                MUX   : OUT STD_LOGIC_VECTOR(3 downto 0)
        );
end digital_clock;

architecture Behavioral of digital_clock is
        COMPONENT clk1Hz IS
                PORT (
        input: IN STD_LOGIC;
        reset  : IN  STD_LOGIC;
        output : OUT STD_LOGIC
    );
    END COMPONENT;

        COMPONENT counter_clock IS
                PORT (
                        clk  : IN  STD_LOGIC;
                        reset: IN  STD_LOGIC;
                        incr_hour : IN  STD_LOGIC;
        incr_min : IN  STD_LOGIC;
```

```
                    incr_hour1 : IN  STD_LOGIC;
        incr_min1 : IN  STD_LOGIC;
                    H1   : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
                    H0   : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
                    M1   : OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
                    M0   : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
                );
    END COMPONENT;

    COMPONENT seven_segment_complete IS
            PORT (
                    clk   : IN  STD_LOGIC;
                    reset : IN  STD_LOGIC;
                    D0    : IN  STD_LOGIC_VECTOR(5 downto 0);
                    D1    : IN  STD_LOGIC_VECTOR(5 downto 0);
                    D2    : IN  STD_LOGIC_VECTOR(5 downto 0);
                    D3    : IN  STD_LOGIC_VECTOR(5 downto 0);
                    output: OUT STD_LOGIC_VECTOR(7 downto 0);
                    MUX   : OUT STD_LOGIC_VECTOR(3 downto 0)
            );
    END COMPONENT;

    signal clk_out : STD_LOGIC := '0';
    signal HH1, MM1: STD_LOGIC_VECTOR(2 downto 0);
    signal HH0, MM0: STD_LOGIC_VECTOR(3 downto 0);
    signal pHH1, pHH0, pMM1, pMM0: STD_LOGIC_VECTOR(5 downto 0);
begin

    clk_i: clk1Hz PORT MAP(clk, reset, clk_out);
    cnt_i: counter_clock PORT MAP(clk_out,
reset,incr_hour,incr_min,incr_hour1,incr_min1,HH1, HH0, MM1, MM0);
    seg_i: seven_segment_complete PORT MAP(clk, reset, pMM0, pMM1, pHH0, pHH1,
output, MUX);

    pHH1 <= "000" & HH1;
    pHH0 <= "00"  & HH0;
    pMM1 <= "000" & MM1;
    pMM0 <= "00"  & MM0;
end Behavioral;
```

Test bench

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY timingminutestb IS
END timingminutestb;

ARCHITECTURE behavior OF timingminutestb IS


   COMPONENT digital_clock
   PORT(
       clk : IN  std_logic;
       reset : IN  std_logic;
       incr_hour : IN  std_logic;
       incr_min : IN  std_logic;
       incr_hour1 : IN  std_logic;
       incr_min1 : IN  std_logic;
       output : OUT  std_logic_vector(7 downto 0);
       MUX : OUT  std_logic_vector(3 downto 0)
      );
   END COMPONENT;


  --Inputs
  signal clk : std_logic := '0';
  signal reset : std_logic := '0';
  signal incr_hour : std_logic := '0';
  signal incr_min : std_logic := '0';
  signal incr_hour1 : std_logic := '0';
  signal incr_min1 : std_logic := '0';

     --Outputs
  signal output : std_logic_vector(7 downto 0);
  signal MUX : std_logic_vector(3 downto 0);

  -- Clock period definitions
  constant clk_period : time := 10 ns;
```

```
BEGIN

        -- Instantiate the Unit Under Test (UUT)
  uut: digital_clock PORT MAP (
       clk => clk,
       reset => reset,
       incr_hour => incr_hour,
       incr_min => incr_min,
       incr_hour1 => incr_hour1,
       incr_min1 => incr_min1,
       output => output,
       MUX => MUX
      );

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


  -- Stimulus process
  stim_proc: process
  begin
    -- hold reset state for 100 ns.
    wait for 100 ns;

    wait for clk_period*10;

    -- insert stimulus here

    wait;
  end process;

END;
```

UCF File

```
NET  "clk"        LOC = "B8";
NET  "reset"      LOC = "N3";

NET  "output<7>"   LOC = "N13"; # dp
NET  "output<6>"   LOC = "M12"; # g
NET  "output<5>"   LOC = "L13"; # f
NET  "output<4>"   LOC = "P12"; # e
NET  "output<3>"   LOC = "N11"; # d
NET  "output<2>"   LOC = "N14"; # c
NET  "output<1>"   LOC = "H12"; # b
NET  "output<0>"   LOC = "L14"; # a


NET  "MUX<3>"     LOC = "F12";
NET  "MUX<2>"     LOC = "J12";
NET  "MUX<1>"     LOC = "M13";
NET  "MUX<0>"     LOC = "K14";


NET  "incr_hour1"   LOC = "A7";
NET  "incr_hour"   LOC = "M4";
NET  "incr_min1"    LOC = "C11";
NET  "incr_min"    LOC = "G12";
```

Testing

Waveform Graphs for both clk1Hz and clk200Hz

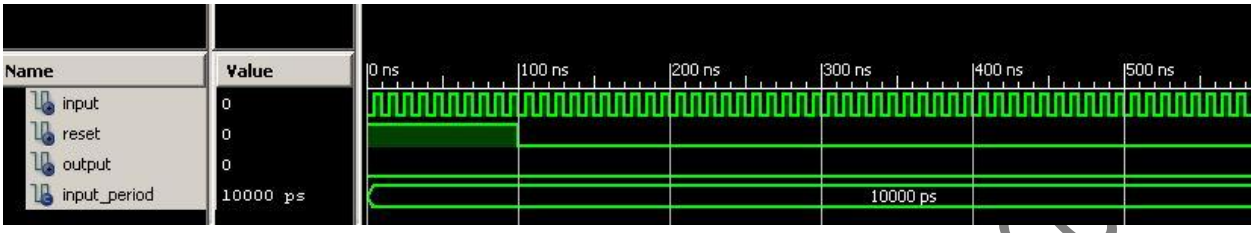

Figure 15: Waveform

The Test Bench code for 200Hz

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY clk200Hz_tb IS
END clk200Hz_tb;

ARCHITECTURE behavior OF clk200Hz_tb IS

   -- Component Declaration for the Unit Under Test (UUT)

   COMPONENT clk200Hz
   PORT(
        input : IN  std_logic;
        reset : IN  std_logic;
        output : OUT  std_logic
       );
   END COMPONENT;


  --Inputs
  signal input : std_logic := '0';
  signal reset : std_logic := '0';

          --Outputs
  signal output : std_logic;
  -- No clocks detected in port list. Replace <clock> below with
  -- appropriate port name
```

```
  constant input_period : time := 10 ns;

BEGIN

        -- Instantiate the Unit Under Test (UUT)
  uut: clk200Hz PORT MAP (
       input => input,
       reset => reset,
       output => output
      );

  -- Clock process definitions
  input_process :process
  begin
                input <= '0';
                wait for input_period/2;
                input <= '1';
                wait for input_period/2;
  end process;


  -- Stimulus process
  stim_proc: process
  begin

    reset <= '1';
      wait  for  100 ns;
      reset <= '0';
      wait;

    wait;
  end process;

END;
```

The Test Bench code for 1Hz

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY clk1Hz_tb IS
END clk1Hz_tb;

ARCHITECTURE behavior OF clk1Hz_tb IS

   -- Component Declaration for the Unit Under Test (UUT)

   COMPONENT clk1Hz
   PORT(
        input : IN  std_logic;
        reset : IN  std_logic;
        output : OUT  std_logic
       );
   END COMPONENT;


  --Inputs
  signal input : std_logic := '0';
  signal reset : std_logic := '0';

          --Outputs
  signal output : std_logic;
  -- No clocks detected in port list. Replace <clock> below with
  -- appropriate port name

  constant input_period : time := 10 ns;

BEGIN

          -- Instantiate the Unit Under Test (UUT)
  uut: clk1Hz PORT MAP (
         input => input,
         reset => reset,
         output => output
        );
```

```
  -- Clock process definitions
  input_process :process
  begin
                  input <= '0';
                  wait for input_period/2;
                  input <= '1';
                  wait for input_period/2;
  end process;


  -- Stimulus process
  stim_proc: process
  begin

    reset <= '1';
      wait  for  100 ns;
      reset <= '0';
      wait;

     wait;
  end process;

END;
```

Summary and Conclusion

With this project, we actually learned the importance of working as a team. First we decided to understand the abilities and skills of one another and how could they be applying into the project. We all knew something better at one topic than one another. With that being said we combined everything we knew at our best making the project a lot easier to work. Then we learned each other's schedule to see when meeting up was more appropriate for everyone on the team. Meeting up together was quite hard as we all had very busy schedules so we decided to work on it separately, so whenever we would meet up we could exchange ideas and experiences. We also learned that communication is a key for success, as we were in contact with other at all times by using web programs such as email and Google drive so we could share information and ideas

References

Books

FPGA Prototyping by VHDL Examples: Xilinx Spartan-3 Version by Pong P. Chu

Websites

http://www.estadofinito.com/reloj-digital/