GRID LABORATORY, OBAFEMI AWOLOWO UNIVERSITY ILE-IFE.

Digital Alarm Clock

SIWES 300 Project

Oyewale Ademola S.

Supervisor: Dr. A.O Oluwatope

Contents

Abstract

This is the technical report for the SIWES 300 final project—Digital Alarm Clock. This project is software and hardware co-design. The system was modeled in VHDL (Very-high-speed integrated circuit Hardware Description Language) and implemented the system using an Altera De2-70 FPGA board.

Introduction

Digital alarm clock is timepiece equipment which is designed to run through the day and night from a particular time when activated to when it is deactivated. My digital alarm clock has the most commonly used functionalities—Start, Pause and Reset.

We used Cyclone II FPGA board to implement the digital alarm clock. The FPGA can be programmed by the VHDL design. The board has internal clocks, on board push buttons and on board LEDs. Therefore, I am going to make the design first in VHDL and program the FPGA. I will use the internal clock to drive the entire system. Those on board LEDs will be used to display each digit. I will use push buttons for the user interaction. User can press buttons to either start, pause or reset the system.

Background and Theory

In this section, I give a background on the knowledge and theory of this project. Particularly, I will explain the operating principle of each component.

Clock Converter:

The following is the table represents each digit and its corresponding running speed:

Digit	Speed
Tenth Second	10Hz
Second	1Hz
Ten Second	0.1Hz
Minute	0.1/6 = 1/60Hz
Ten Minute	(1/60)/10 = 1/600Hz

As we can see, the fasted digit—Tenth Second, will run at the speed of 10Hz. However, our internal clock is 50MHz, which is 5000000 times faster. Therefore, the first thing we need to do is to slow down the clock. We used a clock converter (Figure 1) to accomplish this task:

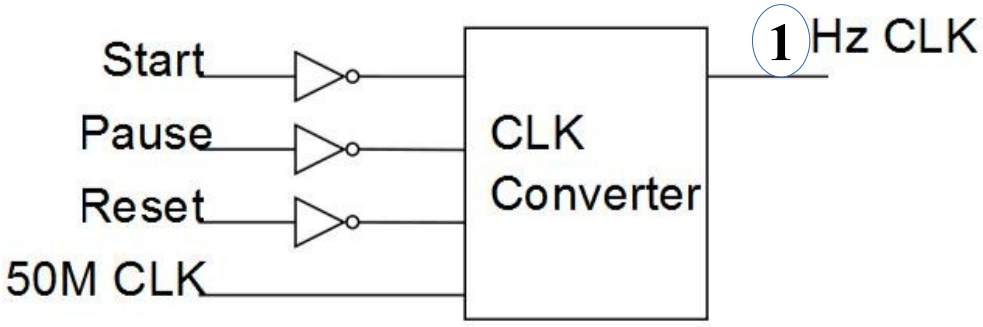

Figure 1 Clock Converter

The 50MHz clock will be fed into the convert. Inside the converter, there is a counter driven by the rising edge of the 50M clock. When it counts from 0 to 2500000, it will invert the output clock and the counter will be reset to 0 (Figure 2).

```
– 25000000 for one second (50Mhz)

IF (cnt = 25000000) THEN
        cnt := 0;
        clockTmp <= NOT clockTmp;
ELSE
        cnt := cnt + 1;

END IF;
```

Figure 2 Slows down the clock

Therefore, the output clock will be 1Hz. The same mechanism will be applies for each stage to get proper clock speed.

Because the clock is driving the entire system, so Start, Pause and Reset are also fed into the converter to control the clock. And all the push buttons on the FPGA board are low active. Therefore, the values of Start, Pause and Reset need to be inverted before connect to other components.

Mod 2, Mod 3, Mod 5 & Mod 9 Counter:

Essentially, each digit of our stop watch is implemented as a modulo counter (Figure 3). To be more specific, digits tenth second, second and minute are mod 9 counters because their values can only go from 0 to 9. Ten second, ten minute are mod 5 counters since their values can only go from 0 to 5. Digit tenth hour is mod 2 since it can only range from 0 to1 while the unit hour is mod 3 since it can only range from 0 to

3. This also indicates that the maximum range of this alarm clock is from 00:00:00 to 23:59:59.

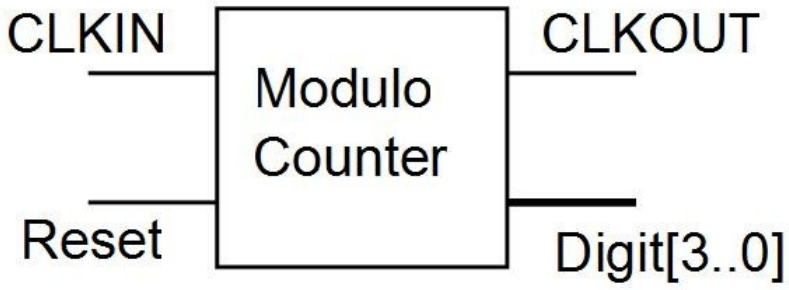

Figure 3 Modulo Counter

The modulo counter also contains the mechanism from previous section to provide proper clock speed to the next stage.

Seven-Segment Decoder:

Each modulo counter will provide its corresponding digit in binary format. In order to display them on the LEDs, we need a 4-to-7 decoder, which will decode each 4-bit digit into 7-bit display format (Figure 4).

The following is the behavior of the decoder. We implemented the decoder in algorithmic level:

```
CASE D IS
    WHEN "0000"=>O<=not "1111110";
    WHEN "0001"=>O<=not "0110000";
    WHEN "0010"=>O<=not "1101101";
    WHEN "0011"=>O<=not "1111001";
    WHEN "0100"=>O<=not "0110011";
    WHEN "0101"=>O<=not "1011011";
    WHEN "0110"=>O<=not "1011111";
    WHEN "0111"=>O<=not "1110000";
    WHEN "1000"=>O<=not "1111111";
    WHEN "1001"=>O<=not "1111011";
    WHEN OTHERS=>O<="XXXXXXX";
END CASE;
```

Figure 4 Seven-Segment Decoder algorithmic behavior

One thing need to pay attention here is that each segment of the seven-segment LED on the board is also low active. Thus, the segment will light only when its corresponding bit is 0. This is just the opposite of the regular decoder.

Design

As we mentioned earlier, the design will be implemented on the Altera DE II Board with Cyclone II FPGA. The following is the board I used for ou development:

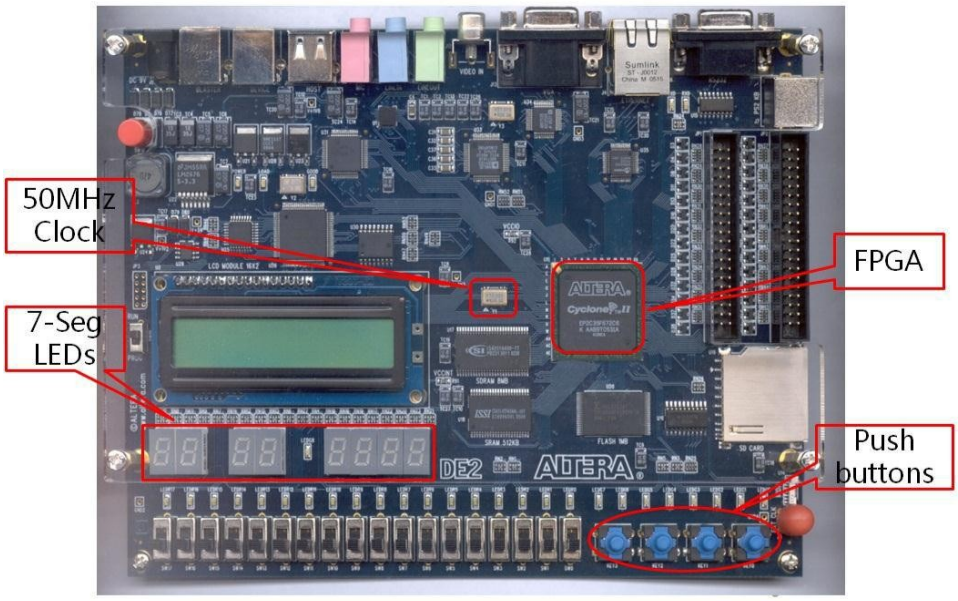

Figure 5 Altera DE2 Board

The Cyclone II FPGA will be used to implement our design. 50MHz internal clock is used to drive the entire system. Seven-Segment LEDs are used for display and push buttons will be used to interact with users.

The following is our design diagram (Figure 6):

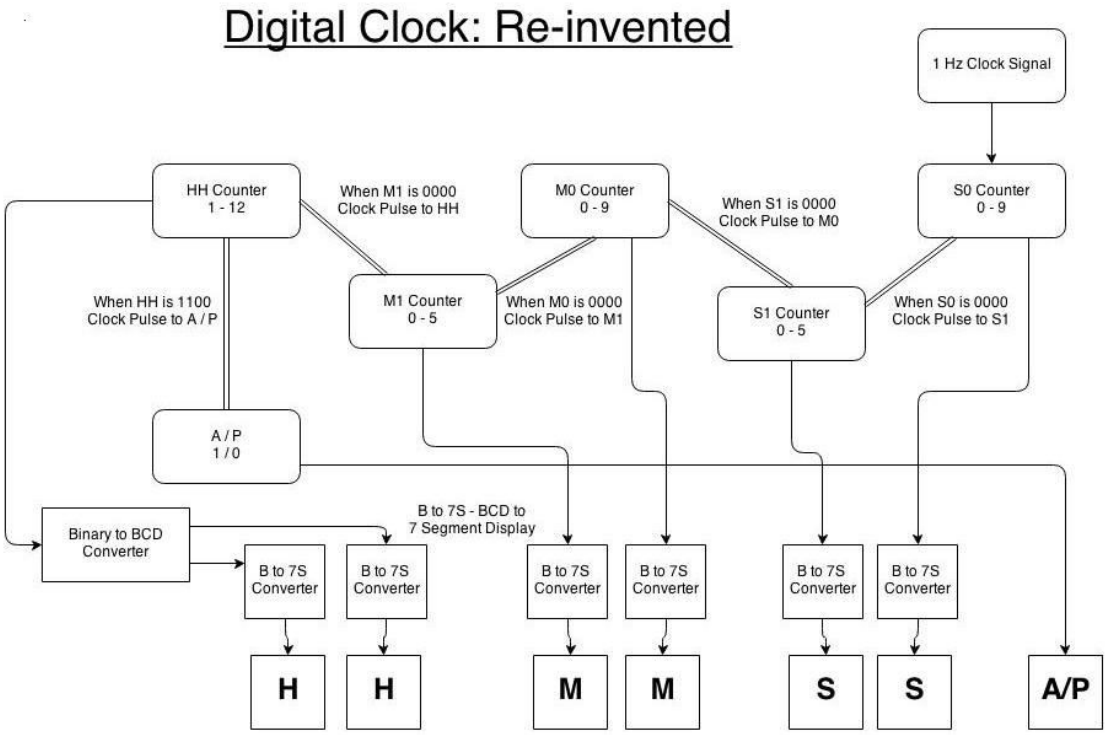

Figure 6 Digital Alarm Clock Design

This is a structural model. It has 6 stages and each stage corresponds to one digit. There are four inputs to this system. 50MHz is connected to the internal clock. Run, Pause and Reset are mapped to the push buttons.

The 50MHz internal clock will be slowed down to 1Hz, so the tenth second digit will increment at every rising edge of this clock. Since it's a modulo 10 counter, once it hits 9, it will go back to 0. In addition, this counter will provide a 10 times slower clock (0.1Hz) to the next stage (second digit). Therefore, the second digits will increment every second. Applying the same idea to connect the other three stages in cascade, eventually, second and ten minute will be ten times slower than their previous stages. Minute will be six times slower than its previous stage.

Each stage will provide one digit in 4-bit binary format. We need a 4-to-7 seven-segment decoder to decode the value into display format. The outputs from the decoders will be mapped to their corresponding segments on the board.

Procedure and Implementations

In order to accomplish this project, I thought about how to implement this digital alarm clock. I have two approaches: one is in algorithmic level and the other one is in structural level. I decided to use the structural level model.

Then we started to test the hardware, such as the push buttons and LEDs. By doing this, I figured out that both of them are low active. Therefore, I have to invert all the values coming from the push buttons and going into the LEDs. I also figured out the pin assignment of the clock, push buttons and LEDs.

I also did plenty of testing on each individual component to make sure they are correct both logically and functionally. After all the testing, I put all the components together and did more testing to the system using waveforms. Then, we downloaded our design to the board and tested on the hardware.

Implementation Images

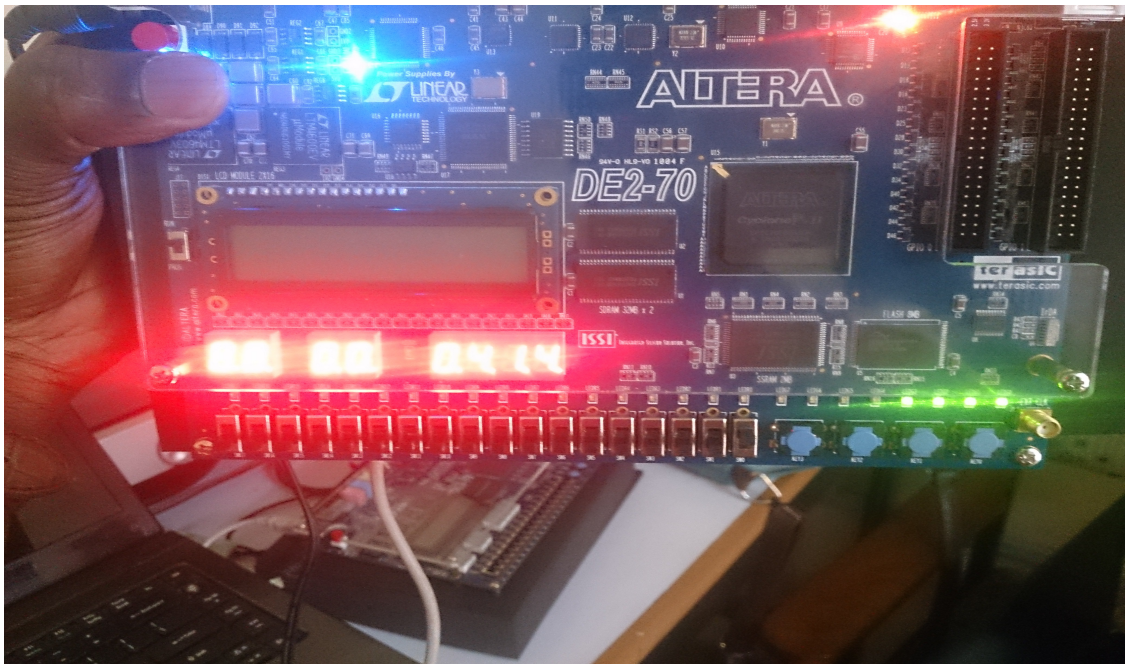

Figure 7

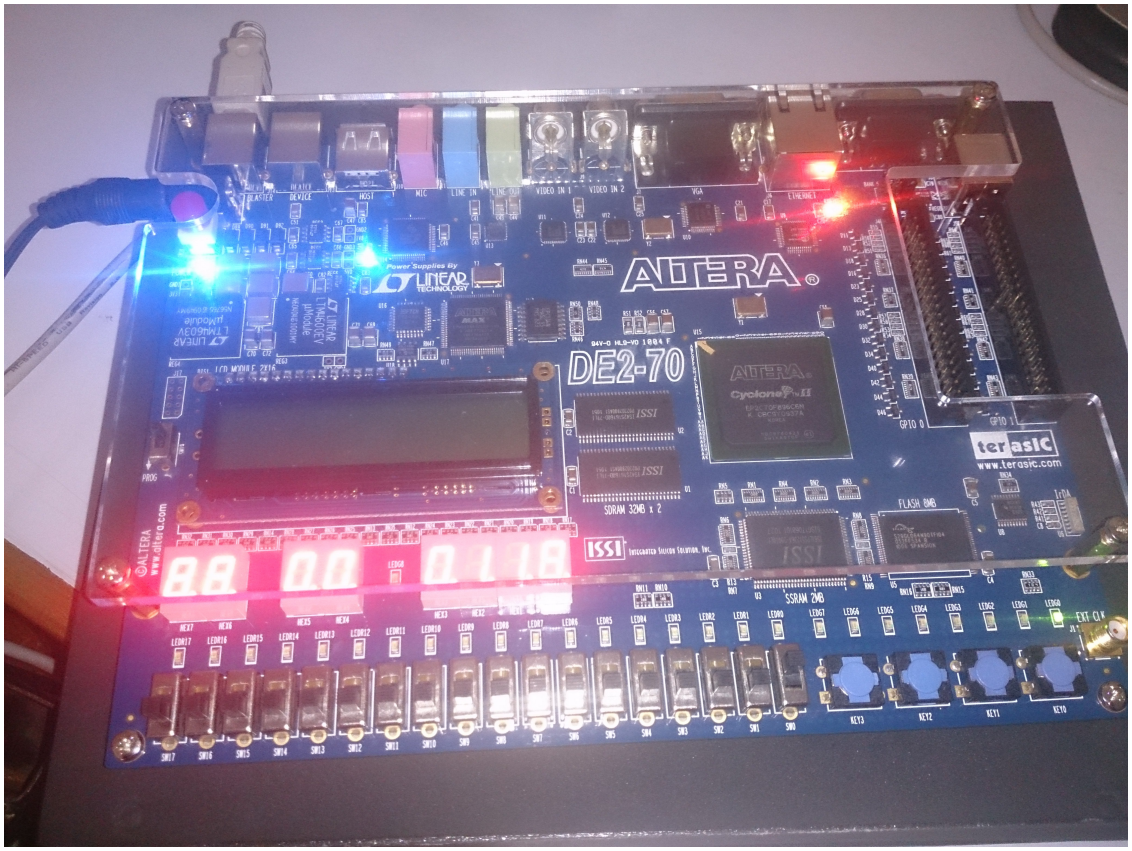

Figure 8

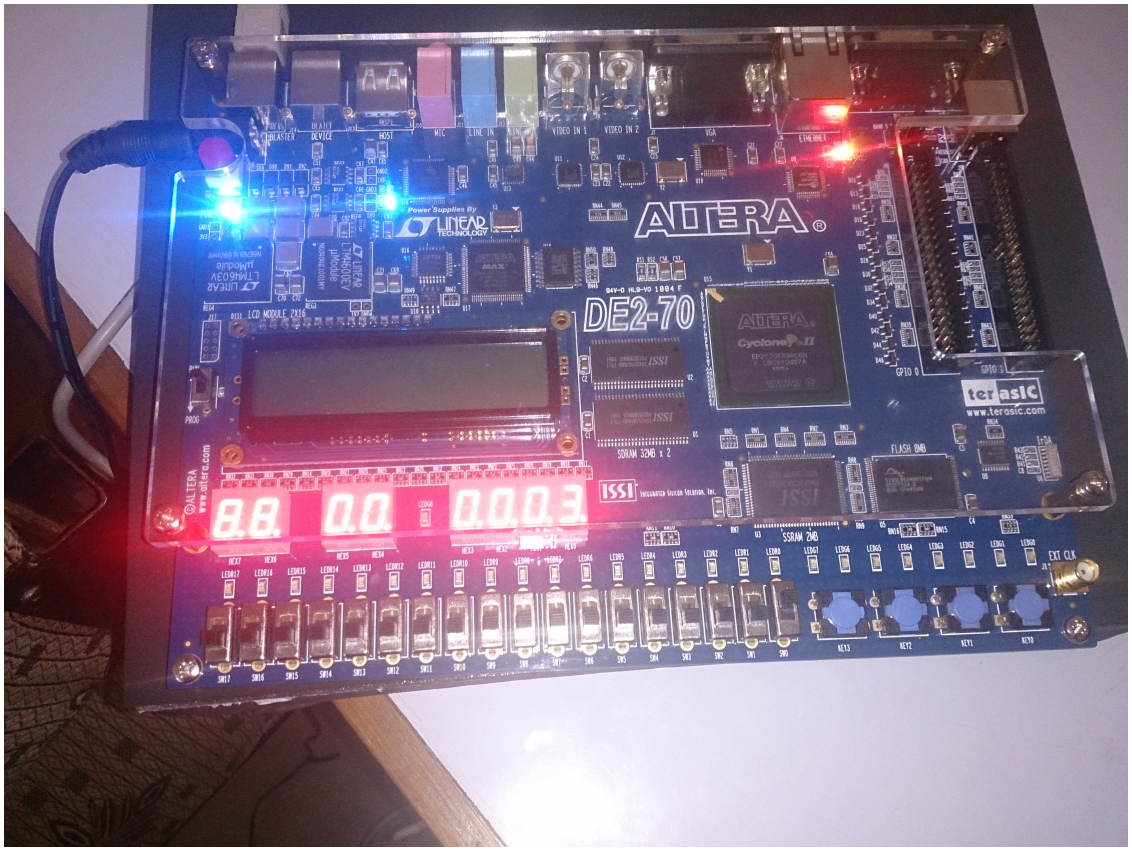

Figure 9

In the Figure 7 above, the FPGA board shows 4 green led bulbs light up when the timer shows 4minutes and 14 seconds. This confirms that the number of led bulbs should increment at each increment of the minute.

Figure 8 also shows only a green led bulb on when the timer shows 1 minute and 18 seconds.

Results and Discussion

Initially, our design was in algorithmic level. We were trying to implement an up counter. It takes the 1Hz clock from the clock converter and straight counts up until the limit. Then the count value will go through a formatter. This formatter will format the

count value into time format, i.e. 00:00:00 formats. After that, we use seven-segment decoders to decode each digit for display.

The Start, Pause and Reset functionality can be implemented and a finite state machine (Figure 7) since we want to change its state only when we press the buttons.

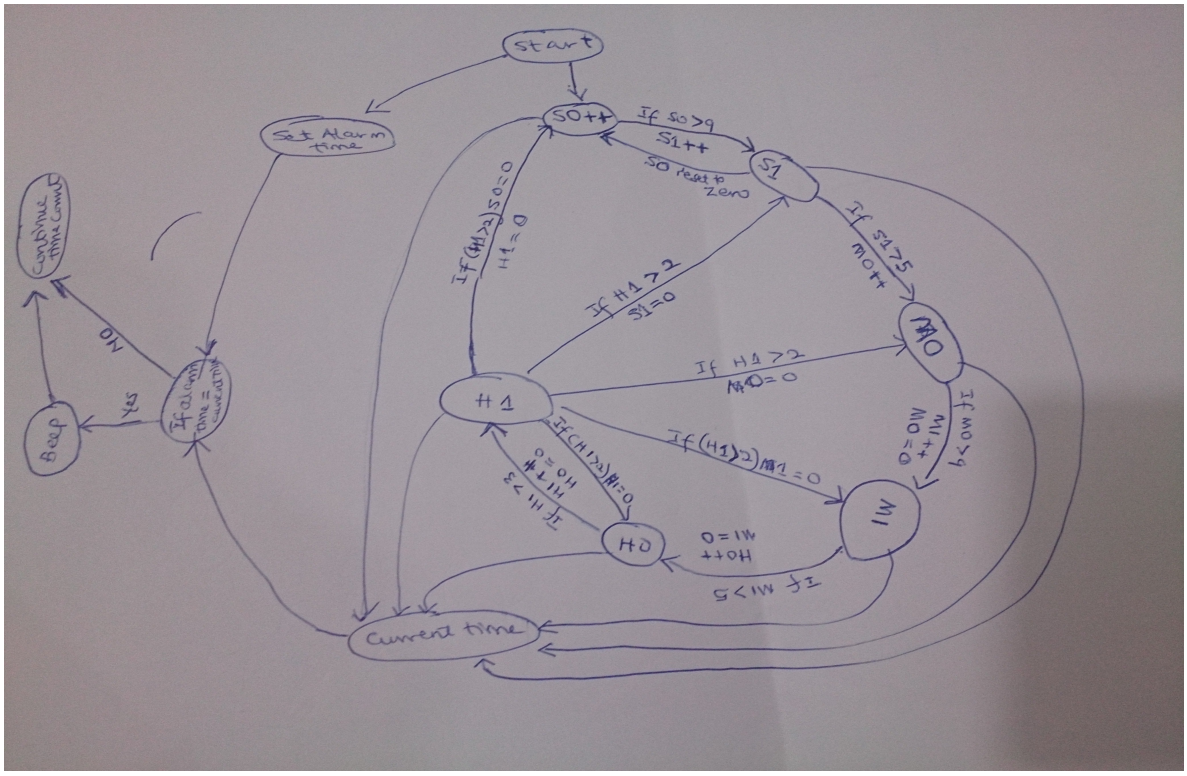

Figure 10 Digital alarm clock finite state machine implementation

This finite state will have states: S0, S1, M0, M1, H0, H1, Set Alarm time (S2) and pause (S3). The state transition will be triggered when button is pressed. Essentially, it is a Mealy Machine implementation, since the next state is determined by the current state as well as the input.

Figure 11. RTL Viewer Generated Diagram

Standards and Society Impact

Digital Alarm Clock has plenty of applications. It can be used to set notifications for various events at different times. Measure the presentation time and give necessary notification . Since this digital alarm clock is implemented on FPGA, we can add other extensions to it. We can set up a time and trigger some events when the stop watch reaches that time point.

Conclusion

In this project, I obtained more experience with Altera DE II board and how to program a FPGA. Moreover, I learned more fundamental principles of designing digital systems. In summary, this project is a good software hardware co-design practice.

VHDL Code

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;


ENTITY count IS
	PORT(
		switch0 : in STD_LOGIC;
		switch1 : in STD_LOGIC;
		ledblink : BUFFER STD_LOGIC;


		ledgrp : OUT STD_LOGIC_VECTOR (11 DOWNTO 0);
		ledgrpcount: buffer integer;
		seg1 : OUT STD_LOGIC_VECTOR(6 downto 0);
		seg2 : OUT STD_LOGIC_VECTOR(6 downto 0);
		seg3 : OUT STD_LOGIC_VECTOR(6 downto 0);
		seg4 : OUT STD_LOGIC_VECTOR(6 downto 0);
		seg5 : OUT STD_LOGIC_VECTOR(6 downto 0);
		seg6 : OUT STD_LOGIC_VECTOR(6 downto 0);
		clk1 : IN STD_LOGIC
	);

END count;


ARCHITECTURE behavioral of count is
   signal h_ms_int : integer range 0 to 2;
   signal h_ls_int : integer range 0 to 9;
   signal m_ms_int : integer range 0 to 5;
   signal m_ls_int : integer range 0 to 9;
   signal s_ms_int : integer range 0 to 5;
   signal s_ls_int : integer range 0 to 9;
   signal clk : std_logic :='0';
   signal count : integer :=1;
   signal ledclk :std_logic := '0';
   signal secCount : integer :=1;
   signal blinkCount : integer := 0;
   signal sls :   STD_LOGIC_VECTOR(3 DOWNTO 0);
   signal sms :   STD_LOGIC_VECTOR(2 DOWNTO 0);
```

```
signal mls :   STD_LOGIC_VECTOR(3 DOWNTO 0);
signal mms :   STD_LOGIC_VECTOR(2 DOWNTO 0);
signal hls :   STD_LOGIC_VECTOR(1 DOWNTO 0);
signal hms :   STD_LOGIC_VECTOR(1 DOWNTO 0);


        begin
            --clk generation.For 50 MHz clock this generates 1 Hz clock.
                process(clk1)
                        begin
                                if(clk1'event and clk1='1') then
                                        count <= count+1;
                                        if(count = .; 25000000) then
                                                clk <= not clk;
                                                count <=1;
                                        end if;
                                end if;
                process;


                --clk generation for 50MHz clock to get it blink at 1/5 of a second (as fast as possible)
                process(clk1)
                        begin
                                if(clk1'event and clk1 = '1') then
                                        secCount <= secCount +1;
                                        if(secCount = 25000000) then
                                                ledclk <= not ledclk;
                                                secCount <=1;
                                        end if;
                                end if;
                end process;


                -- Process that handles the project logic
                clk_proc : process(clk, switch0, switch1)
                        variable ledcount : integer := 1;
                        begin
                                if (clk'event and clk ='1') then
                                        if switch0 = '1' then
                                                if s_ls_int = 9 then
              if s_ms_int = 5 then
                 if m_ls_int = 9 then
                       if m_ms_int = 5 then
                             if (h_ls_int = 9 or (h_ls_int = 3 and h_ms_int = 2)) then
                                 h_ls_int <= 0;
                                if (h_ls_int=3 and h_ms_int=2) then
                                    h_ms_int <= 0;
                                    h_ls_int <= 0;


                               m_ms_int <= 0;
                                        m_ls_int <= 0;
                                        s_ms_int <= 0;
                                        s_ls_int <= 0;
                                    else
                                        h_ms_int <= h_ms_int + 1;


                                    end if;
                                        h_ls_int <= 0;
                                 else
                                     h_ls_int <= h_ls_int + 1;


                                 end if;


                m_ms_int <= 0;
                            else
                                m_ms_int <= m_ms_int + 1;
                            end if;
                                m_ls_int <= 0;
                   else
                                m_ls_int <= m_ls_int + 1;
                                blinkCount <= blinkCount + 1;
                   end if;
                      s_ms_int <= 0;
               else
                   s_ms_int <= s_ms_int + 1;


               end if;
                   s_ls_int <= 0;
            else
                s_ls_int <= s_ls_int + 1;
            end if;
```

```
        elsif switch1 = '1' then
                h_ms_int <= 0;
                h_ls_int <= 0;
                m_ms_int <= 0;
                m_ls_int <= 0;
                s_ms_int <= 0;
                s_ls_int <= 0;
                blinkCount <= 0;
        end if;
end if;
         ledgrpcount <= blinkCount
         hms <= std_logic_vector(to_unsigned(h_ms_int, hms'length));
        hls <= std_logic_vector(to_unsigned(h_ls_int, hls'length));
        mms <= std_logic_vector(to_unsigned(m_ms_int, mms'length));
        mls <= std_logic_vector(to_unsigned(m_ls_int, mls'length));
        sms <= std_logic_vector(to_unsigned(s_ms_int, sms'length));
        sls <= std_logic_vector(to_unsigned(s_ls_int, sls'length));


end process clk_proc;


PROCESS (ledgrpcount)
        BEGIN
                CASE ledgrpcount IS
                        WHEN 0 => ledgrp <= "000000000000";
                        WHEN 1 => ledgrp <= "000000000001";
                        WHEN 2 => ledgrp <= "000000000011";
                        WHEN 3 => ledgrp <= "000000000111";
                        WHEN 4 => ledgrp <= "000000001111";
                        WHEN 5 => ledgrp <= "000000011111";
                        WHEN 6 => ledgrp <= "000000111111";
                        WHEN 7 => ledgrp <= "000001111111";
                        WHEN 8 => ledgrp <= "000011111111";
                        WHEN 9 => ledgrp <= "000111111111";
                        WHEN 10 => ledgrp <= "001111111111";
                        WHEN 11 => ledgrp <= "011111111111";
                        WHEN 12 => ledgrp <= "111111111111";
                        WHEN 13 => ledgrp <= "000000000001";
                        WHEN 14 => ledgrp <= "000000000011";
                        WHEN 15 => ledgrp <= "000000000111";
                        WHEN 16 => ledgrp <= "000000001111";
                        WHEN 17 => ledgrp <= "000000011111";
                        WHEN 18 => ledgrp <= "000000111111";
                        WHEN 19 => ledgrp <= "000001111111";
                        WHEN 20 => ledgrp <= "000011111111";
                        WHEN 21 => ledgrp <= "000111111111";
                        WHEN 22 => ledgrp <= "001111111111";
                        WHEN 23 => ledgrp <= "011111111111";
                        WHEN 24 => ledgrp <= "111111111111";
                        WHEN OTHERS => ledgrp <= "000000000000";
                END CASE;
        END PROCESS;


        --- For the MS  of the Second
        --start: count_mod5 port map(sms1=> sms, seg21=> seg2);
        PROCESS (sms)
                BEGIN
                        CASE sms IS
                                WHEN "000" => seg2 <= "1000000";
                                WHEN "001" => seg2 <= "1111001";
                                WHEN "010" => seg2 <= "0100100";
                                WHEN "011" => seg2 <= "0110000";
                                WHEN "100" => seg2 <= "0011001";
                                WHEN "101" => seg2 <= "0010010";
                                WHEN OTHERS => seg2 <= "1000000";
                        END CASE;
        END PROCESS;
        -- For the LS of the second
        PROCESS (sls)
                BEGIN
                        CASE sls IS
                                WHEN "0000" => seg1  <= "1000000";
                                WHEN "0001" => seg1 <= "1111001";
                                WHEN "0010" => seg1 <= "0100100";
                                WHEN "0011" => seg1 <= "0110000";
                                WHEN "0100" => seg1 <= "0011001";
                                WHEN "0101" => seg1 <= "0010010";
                                WHEN "0110" => seg1 <= "0000010";
                                WHEN "0111" => seg1 <= "1011000";
                                WHEN "1000" => seg1 <= "0000000";
                                WHEN "1001" => seg1 <= "0011000";
                                WHEN OTHERS => seg1 <= "1000000";
```

```
                END CASE;
        END PROCESS;

        --- For the MS of the Second
        PROCESS (mms)
                BEGIN
                        CASE mms IS
                                WHEN "000" => seg4 <= "1000000";
                                WHEN "001" => seg4 <= "1111001";
                                WHEN "010" => seg4 <= "0100100";
                                WHEN "011" => seg4 <= "0110000";
                                WHEN "100" => seg4 <= "0011001";
                                WHEN "101" => seg4 <= "0010010";
                                WHEN OTHERS => seg4 <= "1000000";
                        END CASE;
        END PROCESS;


        -- For the LS of the second
        PROCESS (mls)
                BEGIN
                CASE mls IS
                        WHEN "0000" => seg3 <= "1000000";
                        WHEN "0001" => seg3 <= "1111001";
                        WHEN "0010" => seg3 <= "0100100";
                        WHEN "0011" => seg3 <= "0110000";
                        WHEN "0100" => seg3 <= "0011001";
                        WHEN "0101" => seg3 <= "0010010";
                        WHEN "0110" => seg3 <= "0000010";
                        WHEN "0111" => seg3 <= "1011000";
                        WHEN "1000" => seg3 <= "0000000";
                        WHEN "1001" => seg3 <= "0011000";
                        WHEN OTHERS => seg3 <= "1000000";
                 END CASE;
        END PROCESS;


        --For the MS of the hour hand
        PROCESS(hms)
                BEGIN
                        CASE hls IS
                                WHEN "00" => seg6 <= "1000000";
                                WHEN "01" => seg6 <= "1111001";
                                WHEN "10" => seg6 <= "0100100";
                                WHEN OTHERS => seg6 <= "1000000";
                        END CASE;
        END PROCESS;


        -- For the LS of the hour hand
        PROCESS(hls)
                BEGIN
                        CASE hls IS
                                WHEN "00" => seg5 <= "1000000";
                                WHEN "01" => seg5 <= "1111001";
                                WHEN "10" => seg5 <= "0100100";
                                WHEN "11" => seg5 <= "0110000";
                                WHEN OTHERS => seg5 <= "1000000";
                        END CASE;
        END PROCESS;
END behavioral;
```

Reference

1. http://www.codeproject.com/Tips/444385/Frequency-Divider-with-VHDL
2. https://www.pantechsolutions.net/project-kits/implementation-of-digital-clock
3. https://www.youtube.com/watch?v=Kr6AdSF8qLY
4. http://en.wikipedia.org/wiki/Alarm_clock
5. dl.ifip.org/db/conf/ifip10-5/edutech2005/Bobda05.pdf