EEE102 Term Project:

Math Problem Alarm Clock

Nizam Ercan 22302317 Section: 2

Purpose:

The aim of this project is to create a clock where the user can select an alarm time. When this alarm time comes one of the two buzzers starts to ring. If the user tries to close this buzzer with a button, the second buzzer starts to ring and a random mathematical problem is displayed on the LCD screen. If the user enters the solution of the problem in binary with switches, the alarm can be closed.

Video:

https://www.youtube.com/watch?v=zjPCFPKmuTA

Methodology:

I am trying to design two finite state machines with 2 and 3 states respectively which are the regular clock and alarm time selection and 0 buzzer ringing, only 1 buzzer ringing and two buzzers ringing with the random problem displayed on the LCD. I will use 3 modules which are for showing the time in the 7-segment-display of Basys3, selecting the alarm time and driving the LCD. After making sure that these modules work as intended, I will write the top module where I will make changing between states possible with buttons of Basys3 or whether the alarm time has come or not.

Desing Specifications:

The finite state machine in the top module has 4 inputs and 9 outputs as you can see below:

- clk : in STD_LOGIC
- but : in std_logic_vector(3 downto 0)
- data : in std_logic_vector(6 downto 0)
- button : in std_logic
- data : in std_logic_vector(6 downto 0)
- anode_active : out std_logic_vector(3 downto 0)
- display : out std_logic_vector(6 downto 0)
- LED : out std_logic_vector(3 downto 0)
- first_buzzer : out std_logic := '0'
- second_buzzer : out std_logic := '0'
- E: out std_logic
- RS: out std_logic
- RW: out std_logic
- DB: out std_logic_vector(7 downto 0)

“clk” signal is the clock input from Basys3. “but” and “button” inputs are coming from the buttons of Basys3. “data” is coming from the switches of Basys3. “LED” shows which digit is selected in alarm selection state. “display” and “anode_active” are controlling the 7-segment-display. “first_buzzer” and

“second_buzzer” are controlling the buzzers. “E”, “RS”, “RW” and “DB” are for controlling the LCD. The design hierarchy can be seen in Figure1.

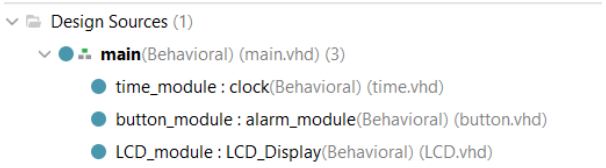

Figure 1: the hierarchy of the code

time.vhd:

This module uses the internal clock of Basys3 to count seconds, minutes and hours. It also has another counter that activates the 4 anodes of the 7-segment-display respectively. Then, depending on the “minutes” and “hours” signals, it changes the “display” output to show the appropriate number on the 7-segment-display. Finally, the module has a 28-bit std_logic_vector “alarm_time” as an output which corresponds to the displayed value at the 7-segment-display. This output will be used for determining whether the alarm time has arrived or not. The RTL schematics for this module can be seen in Figure 2.

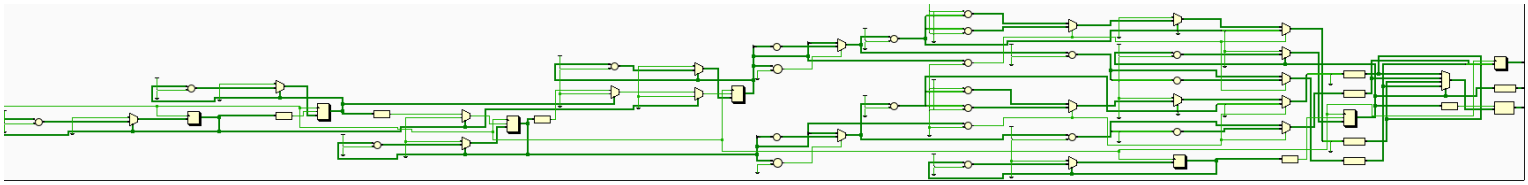

Figure 2: RTL schematics for time.vhd

button.vhd:

This module is a finite state machine with 4 states where each state indicates the digit selected to change. The user can swap between these 4 states by using the buttons of Basys3. The “LED” output shows which digit is selected since they are connected to the pins under the digits of the 7-segment-display. The user can enter the number they wanted in binary with the “data” input which will be connected to the switches of Basys3. Also, there is also a counter that activates the anodes of the 7-segment-display and a display output depending on the binary numbers entered in each state/digit. Finally, this module also has the 28-bit std_logic_vector “alarm_time” as on output which corresponds to the displayed value at the 7-segment-display. The RTL schematics for this module can be seen in Figure 3.

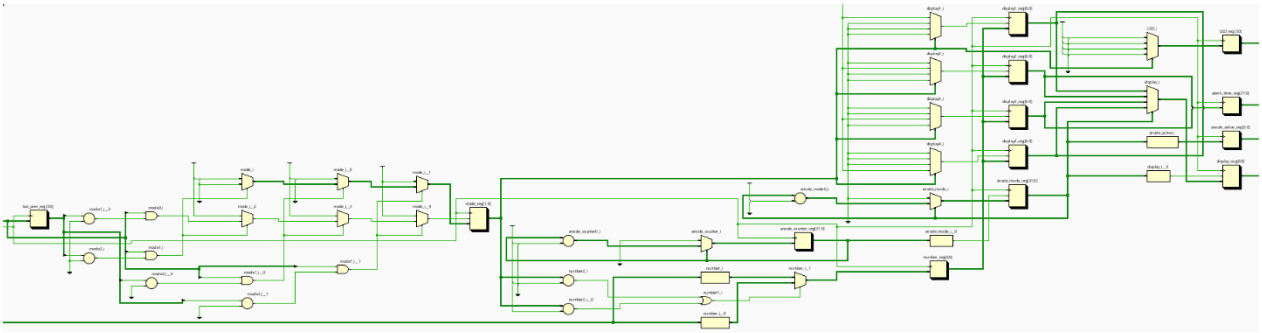

Figure 3: RTL schematics for button.vhd

LCD.vhd:

This module displays the random mathematical operation in the LCD. TC1602A is the type of the LCD I will implement these design. I don't want to read any data from the LCD, I only want to write data according to the inputs of this module. Therefore I am using the writing mode with timing diagram and table given in Figure 4.

Characteristics	Symbol	Limit			Unit	Test Condition
		Min.	Typ.	Max.		
E Cycle Time	t_C	400	-	-	ns	Pin E
E Pulse Width	t_{PW}	150	-	-	ns	Pin E
E Rise/Fall Time	t_R, t_F	-	-	25	ns	Pin E
Address Setup Time	t_{SP1}	30	-	-	ns	Pins: RS, R/W, E
Address Hold Time	t_{HD1}	10	-	-	ns	Pins: RS, R/W, E
Data Setup Time	t_{SP2}	40	-	-	ns	Pins: DB0 - DB7
Data Hold Time	t_{HD2}	10	-	-	ns	Pins: DB0 - DB7

Write Mode Timing Diagram (Writing data from MPU to SPLC780D1)

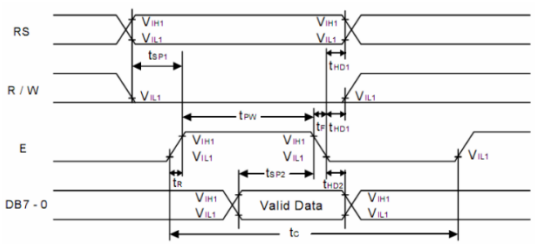

Figure 4: timing diagram for the LCD screen

For the signal "RS", "0" means that I am setting up the LCD and "1" means that I am entering data to be displayed on the LCD. Also, I need to create a wave structure with "E" signal to enter data to the LCD. I am using 8-bit mode that I am setting with "00111000", clearing the display with "00000001", returning home with "00000010" and turning the display on and curser of with "00001100" by creating enough delays for the data and the "E" signal (These values were given in the datasheet for TC1602A). Then I switch to "RS" = "1" and enter the second digit of the "operation" input, the operation corresponding to the first digit of the "operation" input and the third digit of the "operation" input by using the table provided in the datasheet as you can see in Figure 5.

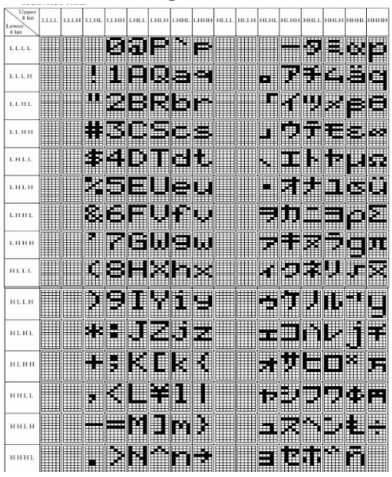

Figure 5: ASCII table for the LCD

“operation” is an integer between 0 and 199(included) that comes from the top module as an input to the LCD.vhd. This integer allows me to display a random operation in the LCD. The first digit which can be either 0 or 1 indicates whether the operation in the LCD is addition or multiplication. The second and the third digits which are between 0 and 9 shows the numbers we are using for the operation. LCD.vhd assigns all the possible values of the digits of “operation” to certain 8-bit std_logic_vectors according to Figure 5 so that they can be displayed on the LCD. Also, this module performs the operation and gives the result integer as the output “answer”. The RTL schematics for this module can be seen in Figure 6.

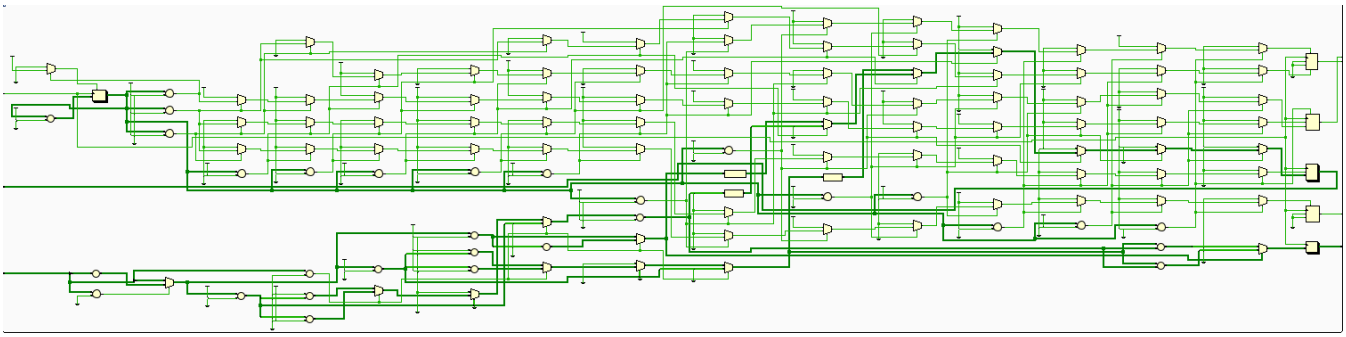

Figure 6: RTL schematics for LCD.vhd

main.vhd:

In this top module, all of the three sub-modules were port mapped. This module consists of 2 finite state machines with 2 and 3 states respectively and these states were shown with 2 signals which are “mode” (std_logic) and “alarm_mode”(integer). The “button” input toggles “mode” where “0” means the output signals of clock.vhd is connected to “anode_active” and “display” outputs of mainf.vhd and “1” means the output signals of button.vhd is connected to the “but”, “anode_active”, “display”, “data” inputs and outputs of main.vhd. When an alarm time is selected by these modes and this alarm time arrives, “alarm_mode” comes from 0 to 1 where 1 one buzzer starts to ring. When the user presses “but(0)”, “alarm_mode” comes from 1 to 2 where the second buzzer also starts to ring and the signals from the outputs of LCD.vhd are connected to the “RS”, “E”, “RW”, and “DB” outputs of the main.vhd which turns on the LED. The only way to get back to “alarm_mode” and turn of the buzzers is to match the “answer” output of LCD.vhd with the “data” input of main.vhd.

In addition, in order to display a random equation at “alarm_mode” = 2, a signal “operation_counter” is formed. This counter counts very fastly between 0 and 200 and when the user presses “but(0)” in “alarm_mode” = 1, “operation_counter”s value gets transferred as the “operation” input of LCD.vhd which achieves a pseudorandomness. The RTL schematics for this module can be seen in Figure 7.

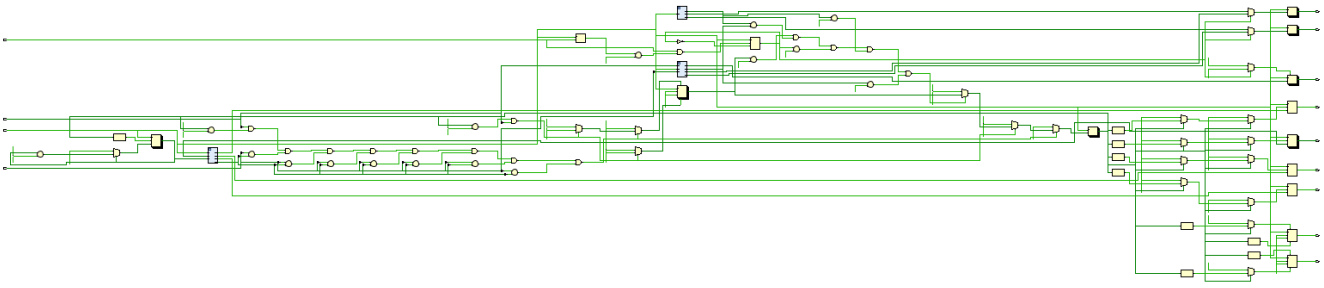

Figure 7: RTL schematics for main.vhd

After designing all the modules and connecting them in “main.vhd”, I wrote the constraints file where I connected the outputs related to the LCD and the buzzers to the PMOD ports of Basys3 and the inputs for data entry and changing modes to the buttons and the switches of Basys3.

Results:

After, finalizing the code for my design, I implemented my design on my breadboard with my Basys3, 2 buzzers and a TC1602A 16x2 LCD. I also used a voltage booster and a SN74HC08N quadruple AND gate for increasing the output voltage of 3.3 V of Basys3 to 5V as the buzzers and the LCD operate with 5V. As you can see in Figure 8, I used the voltage booster to get 5V and used it for giving power to the LCD screen and its backlight. I also connected one input of the and gate to +5V and the other end to the port controlled by “first_buzzer”. I connected the output of this and gate to the + end of the buzzer which allowed me to give it +5V and control it with the “first_buzzer” output.

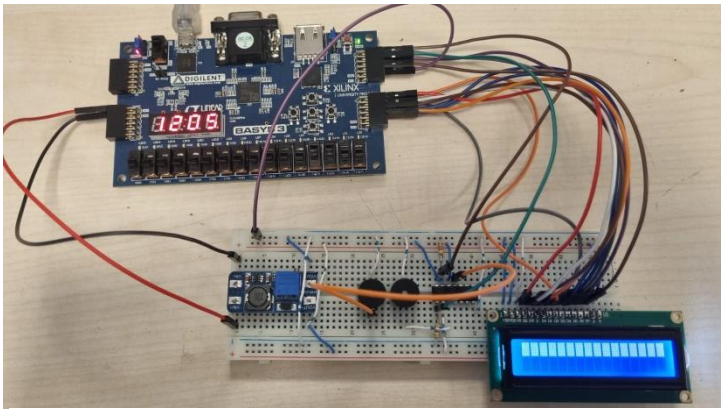

Figure 8: Working design

When I try to select an alarm time, I can use the middle button of Basys3 as you can see in Figure 9 and 10. In addition, I can change digits and change these digits as you can see in the video and the figures below.

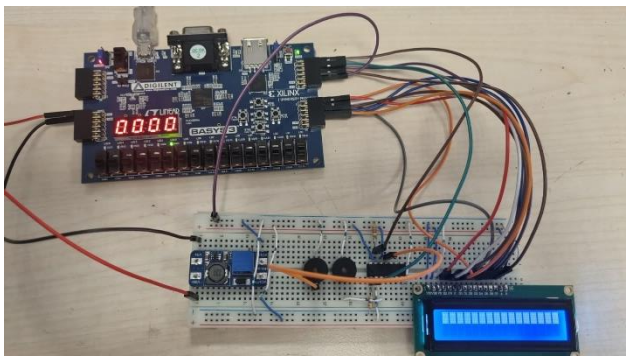

Figure 9: alarm selection mode

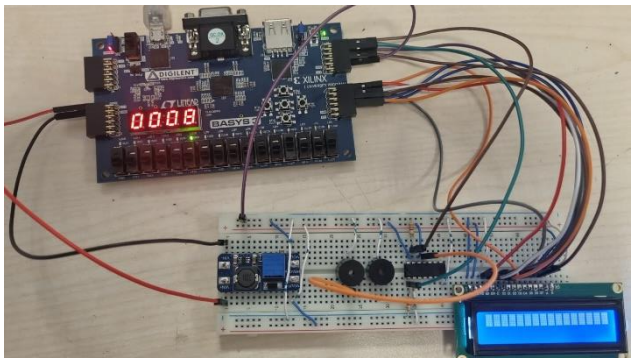

Figure 10: alarm digit selection with switches

When I enter the alarm time 12.01 and this time comes, the first buzzer starts to ring as you can seen in the video. When I press the top button to close the alarm, the second buzzer starts to ring and the random operation 8*2 comes in the LCD screen as you can see in Figure 11.

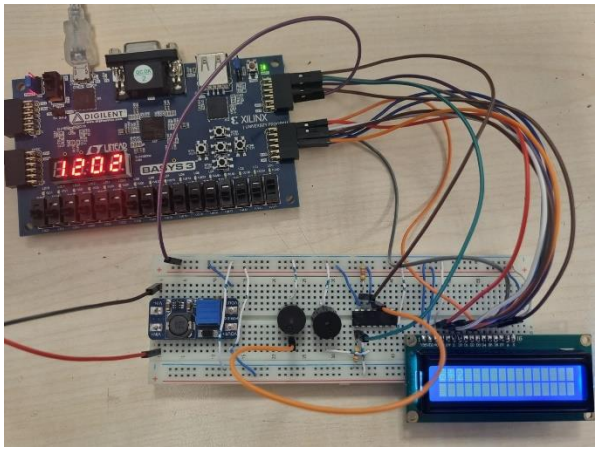

Figure 11: two buzzers are ringing and an operation is displayed

When I enter the result of the operation which is 16 with the switches and press the top button as you can see in the video and Figure 12 the buzzers stop ringing and the LCD stops displaying the operation.

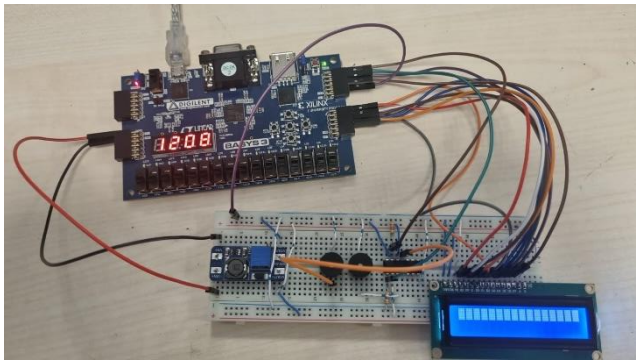

Figure 12: when 16 is entered alarm stops

From this state, the process of the alarm can be repeated with different times and with different operations.

Conclusion:

In the end, I was able to achieve what I promised in the project proposal which was an alarm clock that can be closed by solving a random math problem displayed on an LCD screen. In the process, I learned how to design a finite state machine, read datasheets and control components with an FPGA.

Appendices:

main.vhd:

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


entity main is
    Port ( clk : in STD_LOGIC;
           anode_active : out std_logic_vector(3 downto 0);
           display : out std_logic_vector(6 downto 0);
           but : in std_logic_vector(3 downto 0);
           data : in std_logic_vector(6 downto 0);
           LED : out std_logic_vector(3 downto 0);
           button : in std_logic;
           first_buzzer : out std_logic := '0';
           second_buzzer : out std_logic := '0';
           E: out std_logic;
           RS: out std_logic;
           RW: out std_logic;
           DB: out std_logic_vector(7 downto 0)
           );
end main;


architecture Behavioral of main is


    component alarm_module is
    Port (
```

```
        clk : in std_logic;
        but : in STD_LOGIC_VECTOR (3 downto 0);
        data : in std_logic_vector(3 downto 0);
        LED : out STD_LOGIC_VECTOR (3 downto 0);
        anode_active : out std_logic_vector (3 downto 0);
        alarm_time : out std_logic_vector(27 downto 0);
        display : out std_logic_vector (6 downto 0)
    );
    end component;

    component clock is
    Port ( clk : in STD_LOGIC;
           anode_active : out std_logic_vector(3 downto 0);
           alarm_time : out std_logic_vector(27 downto 0);
           display : out std_logic_vector(6 downto 0) := "0000000"
           );
    end component;

    component LCD_Display is
    Port ( operation : in integer;
           answer : out unsigned(6 downto 0);
           DB : inout STD_LOGIC_VECTOR (7 downto 0);
           RS, RW, E : out STD_LOGIC;
           CLK : in STD_LOGIC
           );
    end component;

    signal display_alarm : std_logic_vector(6 downto 0);
    signal display_time : std_logic_vector(6 downto 0);
    signal anode_active_alarm: std_logic_vector(3 downto 0);
    signal anode_active_time: std_logic_vector(3 downto 0);
    signal alarm_time_alarm: std_logic_vector(27 downto 0) :=
"0000000000000000000000000010";
    signal alarm_time_time: std_logic_vector(27 downto 0) :=
"0000000000000000000000000001";
```

```
    signal LED_enable: std_logic_vector(3 downto 0);
    signal button_prev: std_logic := '0';
    signal mode : std_logic := '0';
    signal alarm_mode: integer := 0;
    signal alarm_mode_prev: integer := 0;
    signal operation_counter: integer := 0;
    signal answer_s: unsigned(6 downto 0);
    signal DB_s: std_logic_vector(7 downto 0);
    signal RS_s,RW_s,E_s: std_logic;
    signal answer_vector : std_logic_vector(6 downto 0);
begin


    time_module: clock port map(clk => clk, anode_active => anode_active_time,
alarm_time => alarm_time_time, display => display_time);
    button_module: alarm_module port map(clk => clk, but => but, data => data(3 downto
0), LED => LED_enable, anode_active => anode_active_alarm, alarm_time =>
alarm_time_alarm, display => display_alarm);
    LCD_module: LCD_display port map(operation => operation_counter, answer =>
answer_s, DB => DB_s, RS => RS_s, RW => RW_s, E => E_s, clk => clk);


    process(clk)
    begin
        if rising_edge(clk) then


            if button = '1' and button_prev = '0' then
                mode <= not mode;
            end if;
            button_prev <= button;


            if mode = '0' then
                display <= display_time;
                anode_active <= anode_active_time;
                LED <= "0000";
            elsif mode = '1' then
                display <= display_alarm;
                anode_active <= anode_active_alarm;
```

```
            LED <= LED_enable;
        end if;

        alarm_mode <= alarm_mode_prev;
        if (alarm_mode_prev = 0) and (alarm_time_time = alarm_time_alarm) and
(mode = '0') and (alarm_time_time /= "00000000000000000000000000000000") and
alarm_time_alarm /= ("00000000000000000000000000000000") then
            alarm_mode <= 1;
        end if;

        if alarm_mode = 0 then
            first_buzzer <= '0';
            second_buzzer <= '0';
        elsif alarm_mode = 1 then
            first_buzzer <= '1';
            second_buzzer <= '0';
        elsif alarm_mode = 2 then
            first_buzzer <= '1';
            second_buzzer <= '1';
            E <= E_s;
            DB <= DB_s;
            RW <= RW_s;
            RS <= RS_s;
        end if;

        if alarm_mode = 1 then
            if operation_counter = 200 then
                operation_counter <= 0;
            else
                operation_counter <= operation_counter + 1;
            end if;
        end if;

        if alarm_mode = 1 and but(0) = '1' then
            alarm_mode <= 2;
```

```
                alarm_mode_prev <= 2;
            end if;

            answer_vector <= std_logic_vector(answer_s);
            if (but(0) = '1') and (alarm_mode = 2) and (data(0) = answer_s(0)) and
(data(1) = answer_s(1)) and (data(2) = answer_s(2)) and (data(3) = answer_s(3)) and
(data(4) = answer_s(4)) and (data(5) = answer_s(5)) and (data(6) = answer_s(6)) then
                alarm_mode <= 0;
                alarm_mode_prev <= 0;


            end if;
        end if;
    end process;


end Behavioral;
```

LCD.vhd:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;

entity LCD_Display is
    Port ( operation : in integer;
           answer : out unsigned(6 downto 0);
           DB : inout STD_LOGIC_VECTOR (7 downto 0);
           RS, RW, E : out STD_LOGIC;
           CLK : in STD_LOGIC
           );
end LCD_Display;

architecture Behavioral of LCD_Display is

    signal counter: integer := 0;
    signal answer_int : integer;
```

```
    signal answer_signal : unsigned(6 downto 0);

    constant zero: std_logic_vector(7 downto 0) := "00110000";
    constant one: std_logic_vector(7 downto 0) := "00110001";
    constant two: std_logic_vector(7 downto 0) := "00110010";
    constant three: std_logic_vector(7 downto 0) := "00110011";
    constant four: std_logic_vector(7 downto 0) := "00110100";
    constant five: std_logic_vector(7 downto 0) := "00110101";
    constant six: std_logic_vector(7 downto 0) := "00110110";
    constant seven: std_logic_vector(7 downto 0) := "00110111";
    constant eight: std_logic_vector(7 downto 0) := "00111000";
    constant nine: std_logic_vector(7 downto 0) := "00111001";
    constant plus: std_logic_vector(7 downto 0) := "00101011";
    constant multiply: std_logic_vector(7 downto 0) := "00101010";



begin
    process(clk)
    begin
        if rising_edge(clk) then
            if counter <= 10000000 then
                E <= '0';
                counter <= counter + 1;
            elsif counter <= 20000000 then
                E <= '1';
                RS <= '0';
                RW <= '0';
                DB <= "00111000"; --function set
                counter <= counter + 1;
            elsif counter <= 30000000 then
                E <= '0';
                counter <= counter + 1;
            elsif counter <= 40000000 then
```

```
    E <= '1';
    RS <= '0';
    RW <= '0';
    DB <= "00000001"; -- clear display
    counter <= counter + 1;
elsif counter <= 50000000 then
    E <= '0';
    counter <= counter + 1;
elsif counter <= 60000000 then
    E <= '1';
    RS <= '0';
    RW <= '0';
    DB <= "00000010"; --return home
    counter <= counter + 1;
elsif counter <= 70000000 then
    E <= '0';
    counter <= counter + 1;
elsif counter <= 80000000 then
    E <= '1';
    RS <= '0';
    RW <= '0';
    DB <= "00001100"; --display on no cursor
    counter <= counter + 1;
elsif counter <= 90000000 then
    E <= '0';
    counter <= counter + 1;
elsif counter <= 100000000 then
    case (operation mod 10) is
        when 0 => DB <= zero;
        when 1 => DB <= one;
        when 2 => DB <= two;
        when 3 => DB <= three;
        when 4 => DB <= four;
        when 5 => DB <= five;
```

```
            when 6 => DB <= six;
            when 7 => DB <= seven;
            when 8 => DB <= eight;
            when 9 => DB <= nine;
            when others => DB <= "01010101"; --first digit
        end case;
        E <= '1';
        RS <= '1';
        RW <= '0';
        counter <= counter + 1;
    elsif counter <= 110000000 then
        E <= '0';
        counter <= counter + 1;
    elsif counter <= 120000000 then
        case (operation / 100) is
            when 0 => DB <= plus;
            when 1 => DB <= multiply;
            when others => DB <= zero; -- operation
        end case;
        E <= '1';
        RS <= '1';
        RW <= '0';
        counter <= counter + 1;
    elsif counter <= 130000000 then
        E <= '0';
        counter <= counter + 1;
    elsif counter <= 140000000 then
        case (operation mod 100) / 10 is
            when 0 => DB <= zero;
            when 1 => DB <= one;
            when 2 => DB <= two;
            when 3 => DB <= three;
            when 4 => DB <= four;
            when 5 => DB <= five;
```

```
                    when 6 => DB <= six;
                    when 7 => DB <= seven;
                    when 8 => DB <= eight;
                    when 9 => DB <= nine;
                    when others => DB <= "01010101"; -- second digit
                end case;
                E <= '1';
                RS <= '1';
                RW <= '0';
                counter <= counter + 1;
            elsif counter <= 150000000 then
                E <= '0';
                counter <= counter + 1;
            else
                counter <= 0;
            end if;

            if ((operation/100) = 0) then
                answer_int <= (operation mod 10) + ((operation mod 100) / 10);
            else
                answer_int <= (operation mod 10) * ((operation mod 100) / 10);
            end if;

        end if;
    end process;

    answer_signal <= TO_UNSIGNED(answer_int,7);
    answer <= answer_signal;

end Behavioral;
```

button.vhd:

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

```
entity alarm_module is
    Port (
        clk : in std_logic;
        but : in STD_LOGIC_VECTOR (3 downto 0);
        data : in std_logic_vector(3 downto 0);
        LED : out STD_LOGIC_VECTOR (3 downto 0);
        anode_active : out std_logic_vector (3 downto 0);
        alarm_time : out std_logic_vector(27 downto 0);
        display : out std_logic_vector (6 downto 0)
    );
end alarm_module;

architecture Behavioral of alarm_module is

    signal anode_mode: integer := 0;
    signal anode_counter: integer := 0;
    signal but_prev : STD_LOGIC_VECTOR (3 downto 0) := (others => '0');
    signal mode : std_logic_vector (1 downto 0) := (others => '0');
    signal number : std_logic_vector (6 downto 0);
    signal display1 : std_logic_vector (6 downto 0);
    signal display2 : std_logic_vector (6 downto 0);
    signal display3 : std_logic_vector (6 downto 0);
    signal display4 : std_logic_vector (6 downto 0);

begin

    -- Anode counter and mode process
    process(clk)
    begin
        if rising_edge(clk) then
            if anode_counter = 100000 then
                anode_counter <= 0;
                if anode_mode = 4 then
                    anode_mode <= 0;
```

```
            else
                anode_mode <= anode_mode + 1;
            end if;
        else
            anode_counter <= anode_counter + 1;
        end if;
    end if;
end process;

-- Button press detection process
process(clk)
begin
    if rising_edge(clk) then
        for i in 0 to 3 loop
            if but(i) = '1' and but_prev(i) = '0' then
                case i is
                    when 0 =>
                        mode <= "00";
                    when 1 =>
                        mode <= "01";
                    when 2 =>
                        mode <= "10";
                    when 3 =>
                        mode <= "11";
                    when others =>
                        mode <= (others => '0');
                end case;
            end if;
        end loop;
        but_prev <= but;
    end if;
end process;

-- Data to 7-segment display encoding process
```

```
process(clk)
begin
    if rising_edge(clk) then
        if mode = "01" or mode = "11" then
            case data is
            when "0000" => number <= "0000001"; -- 0
            when "0001" => number <= "1001111"; -- 1
            when "0010" => number <= "0010010"; -- 2
            when "0011" => number <= "0000110"; -- 3
            when "0100" => number <= "1001100"; -- 4
            when "0101" => number <= "0100100"; -- 5
            when others => number <= "0100100"; -- Error
            end case;
        else
            case data is
            when "0000" => number <= "0000001"; -- 0
            when "0001" => number <= "1001111"; -- 1
            when "0010" => number <= "0010010"; -- 2
            when "0011" => number <= "0000110"; -- 3
            when "0100" => number <= "1001100"; -- 4
            when "0101" => number <= "0100100"; -- 5
            when "0110" => number <= "0100000"; -- 6
            when "0111" => number <= "0001111"; -- 7
            when "1000" => number <= "0000000"; -- 8
            when "1001" => number <= "0000100"; -- 9
            when others => number <= "0000100"; -- Error
            end case;
        end if;
    end if;
end process;

-- Assigning the number to the correct display based on the mode
process(clk)
begin
```

```
    if rising_edge(clk) then
        case mode is
            when "00" => display1 <= number;
            when "01" => display2 <= number;
            when "10" => display3 <= number;
            when "11" => display4 <= number;
            when others => null; -- Do nothing for invalid modes
        end case;
    end if;
end process;

process(clk)
begin
    if rising_edge(clk) then
        case mode is
            when "00" => LED <= "0001";
            when "01" => LED <= "0010";
            when "10" => LED <= "0100";
            when "11" => LED <= "1000";
            when others => null; -- Do nothing for invalid modes
        end case;
    end if;
end process;

-- Anode activation and display output process
process(clk)
begin
    if rising_edge(clk) then
        case anode_mode is
            when 0 => anode_active <= "1110";
                      display <= display1;
            when 1 => anode_active <= "1101";
                      display <= display2;
            when 2 => anode_active <= "1011";
```

```
                        display <= display3;
                when 3 => anode_active <= "0111";
                        display <= display4;
                when others => anode_active <= "1111";
            end case;
        end if;
    end process;

    process(clk)
    begin
        if rising_edge(clk) then
            alarm_time(6 downto 0) <= display1;
            alarm_time(13 downto 7) <= display2;
            alarm_time (20 downto 14) <= display3;
            alarm_time(27 downto 21) <= display4;
        end if;
    end process;
end Behavioral;
```

time.vhd:

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

entity clock is
    Port ( clk : in STD_LOGIC;
           anode_active : out std_logic_vector(3 downto 0);
           alarm_time : out std_logic_vector(27 downto 0);
           display : out std_logic_vector(6 downto 0) := "0000000"
           );
end clock;

architecture Behavioral of clock is

    signal anode_counter : integer := 0;
    signal second_counter : integer := 0;
```

```
    signal anode_mode : integer := 0;

    signal second : integer := 0;
    signal minute : integer := 00;
    signal hour : integer := 12;

    signal display1 : std_logic_vector(6 downto 0);
    signal display2 : std_logic_vector(6 downto 0);
    signal display3 : std_logic_vector(6 downto 0);
    signal display4 : std_logic_vector(6 downto 0);

    signal hour_first_digit : integer := 0;
    signal hour_second_digit : integer := 0;
    signal minute_first_digit : integer := 0;
    signal minute_second_digit : integer := 0;
begin

process(anode_mode)
begin
case anode_mode is
    when 0 => anode_active <= "1110";
              display <= display1;
    when 1 => anode_active <= "1101";
              display <= display2;
    when 2 => anode_active <= "1011";
              display <= display3;
    when 3 => anode_active <= "0111";
               display <= display4;
    when others => anode_active <= "1111";
end case;
end process;
```

```
process(clk)
begin

    if rising_edge(clk) then

        --anodes
        if anode_counter = 100000 then
            anode_counter <= 0;
            if anode_mode = 4 then
                anode_mode <= 0;
            else
                anode_mode <= anode_mode + 1;
            end if;
        else
            anode_counter <= anode_counter + 1;
        end if;



        --seconds
        if second_counter = 100000000 then
            second_counter <= 0;
            if second = 59 then
                second <= 0;
                if minute = 59 then
                    minute <= 0;
                    if hour = 23 then
                        hour <= 0;
                    else
                        hour <= hour + 1;
                    end if;
                else
                    minute <= minute + 1;
                end if;
```

```
            else
                second <= second + 1;
            end if;
        else
            second_counter <= second_counter + 1;
        end if;

    end if;

end process;

hour_first_digit <= hour mod 10;
hour_second_digit <= hour / 10;

process(second)
begin
case hour_first_digit is

    when 0 => display3 <= "0000001";
    when 1 => display3 <= "1001111";
    when 2 => display3 <= "0010010";
    when 3 => display3 <= "0000110";
    when 4 => display3 <= "1001100";
    when 5 => display3 <= "0100100";
    when 6 => display3 <= "0100000";
    when 7 => display3 <= "0001111";
    when 8 => display3 <= "0000000";
    when 9 => display3 <= "0000100";
    when others => display3 <= "0011000";

end case;

case hour_second_digit is
    when 0 => display4 <= "0000001";
```

```
    when 1 => display4 <= "1001111";
    when 2 => display4 <= "0010010";
    when 3 => display4 <= "0000110";
    when 4 => display4 <= "1001100";
    when 5 => display4 <= "0100100";
    when 6 => display4 <= "0100000";
    when 7 => display4 <= "0001111";
    when 8 => display4 <= "0000000";
    when 9 => display4 <= "0000100";
    when others => display4 <= "0011000";
end case;
end process;

minute_first_digit <= minute mod 10;
minute_second_digit <= minute / 10;

process(minute)
begin

    case minute_first_digit is

        when 0 => display1 <= "0000001";
        when 1 => display1 <= "1001111";
        when 2 => display1 <= "0010010";
        when 3 => display1 <= "0000110";
        when 4 => display1 <= "1001100";
        when 5 => display1 <= "0100100";
        when 6 => display1 <= "0100000";
        when 7 => display1 <= "0001111";
        when 8 => display1 <= "0000000";
        when 9 => display1 <= "0000100";
        when others => display1 <= "0011000";

    end case;
```

```
    case minute_second_digit is
        when 0 => display2 <= "0000001";
        when 1 => display2 <= "1001111";
        when 2 => display2 <= "0010010";
        when 3 => display2 <= "0000110";
        when 4 => display2 <= "1001100";
        when 5 => display2 <= "0100100";
        when 6 => display2 <= "0100000";
        when 7 => display2 <= "0001111";
        when 8 => display2 <= "0000000";
        when 9 => display2 <= "0000100";
        when others => display2 <= "0011000";


    end case;
end process;


process(clk)
begin
    if rising_edge(clk) then
        alarm_time(6 downto 0) <= display1;
        alarm_time(13 downto 7) <= display2;
        alarm_time (20 downto 14) <= display3;
        alarm_time(27 downto 21) <= display4;
    end if;
end process;


end Behavioral;
constraints.xdc:
set_property IOSTANDARD LVCMOS33 [get_ports {but[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {but[2]}]
```

```
set_property IOSTANDARD LVCMOS33 [get_ports {but[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {but[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[0]}]
set_property PACKAGE_PIN N3 [get_ports {LED[3]}]
set_property PACKAGE_PIN P3 [get_ports {LED[2]}]
set_property PACKAGE_PIN U3 [get_ports {LED[1]}]
set_property PACKAGE_PIN W3 [get_ports {LED[0]}]
set_property PACKAGE_PIN T18 [get_ports {but[0]}]
set_property PACKAGE_PIN T17 [get_ports {but[1]}]
set_property PACKAGE_PIN U17 [get_ports {but[2]}]
set_property PACKAGE_PIN W19 [get_ports {but[3]}]

set_property IOSTANDARD LVCMOS33 [get_ports clk]
set_property PACKAGE_PIN W5 [get_ports clk]

set_property IOSTANDARD LVCMOS33 [get_ports {anode_active[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {anode_active[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {anode_active[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {anode_active[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {data[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {data[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {data[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {data[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {display[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {display[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {display[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {display[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {display[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {display[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {display[0]}]
set_property PACKAGE_PIN W4 [get_ports {anode_active[3]}]
```

```
set_property PACKAGE_PIN V4 [get_ports {anode_active[2]}]
set_property PACKAGE_PIN U4 [get_ports {anode_active[1]}]
set_property PACKAGE_PIN U2 [get_ports {anode_active[0]}]
set_property PACKAGE_PIN W7 [get_ports {display[6]}]
set_property PACKAGE_PIN W6 [get_ports {display[5]}]
set_property PACKAGE_PIN U8 [get_ports {display[4]}]
set_property PACKAGE_PIN V8 [get_ports {display[3]}]
set_property PACKAGE_PIN U5 [get_ports {display[2]}]
set_property PACKAGE_PIN V5 [get_ports {display[1]}]
set_property PACKAGE_PIN U7 [get_ports {display[0]}]
set_property PACKAGE_PIN V17 [get_ports {data[0]}]
set_property PACKAGE_PIN V16 [get_ports {data[1]}]
set_property PACKAGE_PIN W16 [get_ports {data[2]}]
set_property PACKAGE_PIN W17 [get_ports {data[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports button]
set_property PACKAGE_PIN U18 [get_ports button]
set_property IOSTANDARD LVCMOS33 [get_ports {DB[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {DB[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {DB[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {DB[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {DB[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {DB[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {DB[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {DB[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports E]
set_property IOSTANDARD LVCMOS33 [get_ports first_buzzer]
set_property IOSTANDARD LVCMOS33 [get_ports RS]
set_property IOSTANDARD LVCMOS33 [get_ports RW]
set_property IOSTANDARD LVCMOS33 [get_ports second_buzzer]
set_property PACKAGE_PIN C15 [get_ports {DB[7]}]
set_property PACKAGE_PIN A17 [get_ports {DB[6]}]
set_property PACKAGE_PIN A15 [get_ports {DB[5]}]
set_property PACKAGE_PIN R18 [get_ports {DB[4]}]
set_property PACKAGE_PIN P17 [get_ports {DB[3]}]
```

```
set_property PACKAGE_PIN M19 [get_ports {DB[2]}]
set_property PACKAGE_PIN L17 [get_ports {DB[1]}]
set_property PACKAGE_PIN P18 [get_ports {DB[0]}]
set_property PACKAGE_PIN N17 [get_ports E]
set_property PACKAGE_PIN K17 [get_ports RS]
set_property PACKAGE_PIN M18 [get_ports RW]
set_property PACKAGE_PIN A14 [get_ports first_buzzer]
set_property PACKAGE_PIN A16 [get_ports second_buzzer]
set_property IOSTANDARD LVCMOS33 [get_ports {data[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {data[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {data[4]}]
set_property PACKAGE_PIN W15 [get_ports {data[4]}]
set_property PACKAGE_PIN V15 [get_ports {data[5]}]
set_property PACKAGE_PIN W14 [get_ports {data[6]}]
```

References:

https://cdn-shop.adafruit.com/datasheets/TC1602A-01T.pdf

https://www.ti.com/lit/ds/symlink/sn74hc08.pdf?ts=1715287856844&ref_url=https%253A%252F%252Fwww.google.com%252F