EEE 102 Term Project

Base Converter

Mehmet Emre Uncu

22003884

Section 02

Youtube:

https://youtu.be/i22pxs_1PQ4

Purpose:

This project aimed to make a base converter that converts four number systems (decimal, hexadecimal, octal, and binary) to each other using BASYS3.

Methodology:

The first step was to determine the project topic. As a result of the research done on the internet, some options were specified. The base converter has been chosen among these options because it has not been done before. After this stage, which number systems would be converted between them was considered. It was decided to make twelve different conversions using the four main number systems (decimal, hexadecimal, octal, and binary). Finally, a component was searched for number and letter inputs, and it was decided to use the 4x4 keypad since it contains all 16 numbers and letters (0 to F). Among the three different keypads found, the membrane keypad was found to provide the most consistent use, and it was decided to use breadboards and jumpers for less contact failure.

In the coding part, firstly, the working logic of the keypad was investigated, and a VHDL code was created so that the entered input would appear on the seven-segment display. Then, if more than one character is entered, a shift register is designed so that the old input can pass to the left and the new input can come to the right so that the desired number from one digit to four digits (0 to FFFF) can be displayed on the FPGA. Lastly, the necessary VHDL codes were written to make the conversion in the determined mode. After the design process was completed, the project was simulated on BASYS3.

Design Specifications:

The design includes one main module, three top modules, and eight different submodules under them (Figure 1).

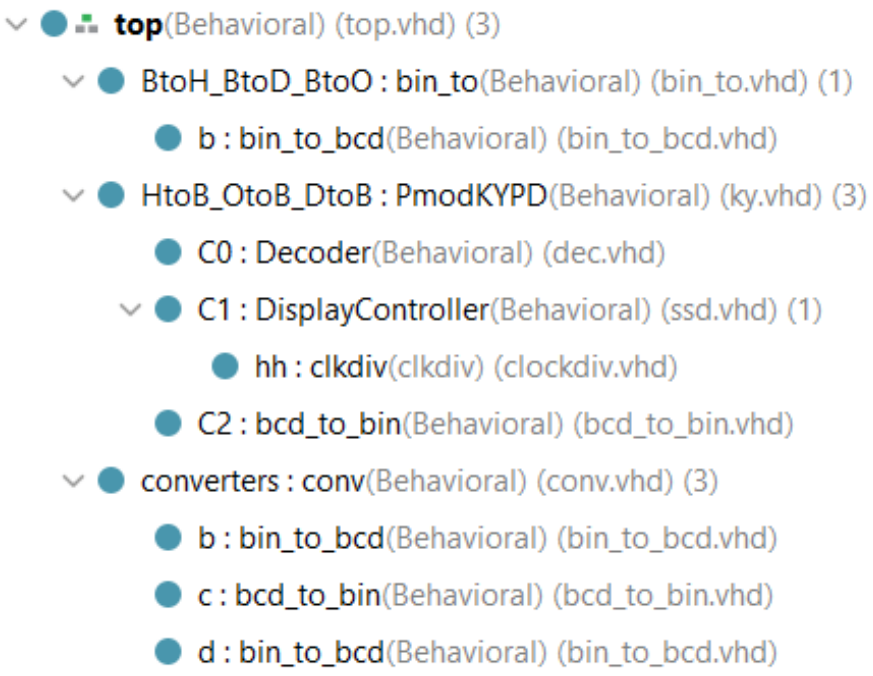

Figure 1: The Hierarchy

There are five inputs, two buffers, and two outputs in the main module as follows:

- Clk : in std_logic; clock
- Shift : in std_logic; enable button for shift register
- Reset : in std_logic; reset button for clock
- Selection : in std_logic_vector(3 downto 0); 4 switches to determine conversion mode
- Sw : in std_logic_vector(11 downto 0); 12 switches for binary inputs
- JA : buffer std_logic_vector (7 downto 0); pin buffers for keypad
- led : out std_logic_vector (15 downto 0); 16 LEDs on FPGA for binary outputs
- an : out std_logic_vector (3 downto 0); seven segment display anode selection
- seg : buffer std_logic_vector (6 downto 0)); seven segment display outputs

The “top.vhd” module (Figure 2) takes the necessary inputs according to the selected mode and distributes them to the top module to be used. Then it assigns the outputs from the top modules to the main outputs.

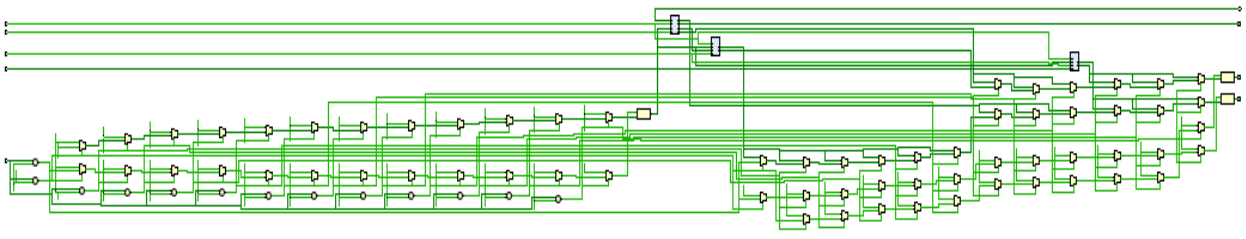

Figure 2: The RTL Schematics of "top.vhd"

The "bin_to.vhd" top module (Figure 3) is activated in the first three conversion modes, takes the binary number as input with the switches, and then shows the conversion on the seven-segment display in hexadecimal, octal, or decimal base by the selected mode.

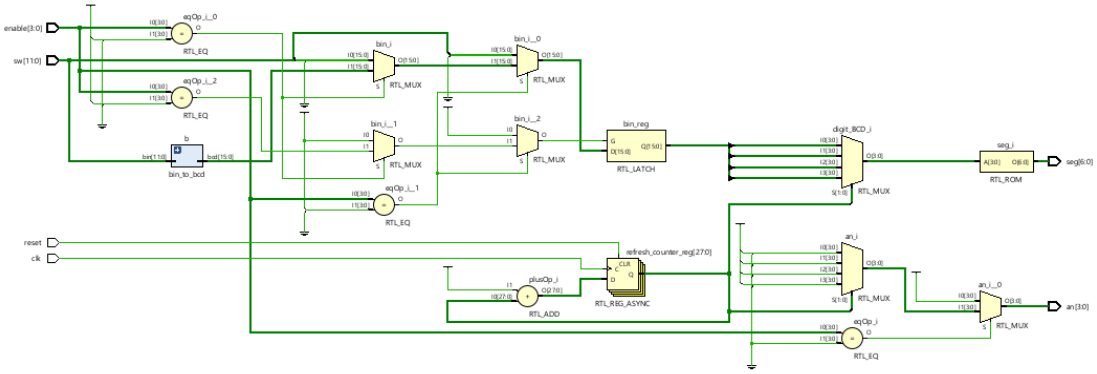

Figure 3: The RTL Schematics of "bin_to.vhd"

The "ky.vhd" top module (Figure 4) contains the “dec.vhd” and “ssd.vhd” submodules and is activated when converting a hexadecimal, octal, or decimal number to a binary base. It shows the number entered with the keypad on the seven-segment display, and when the mode is selected, it outputs the binary number on the LEDs.

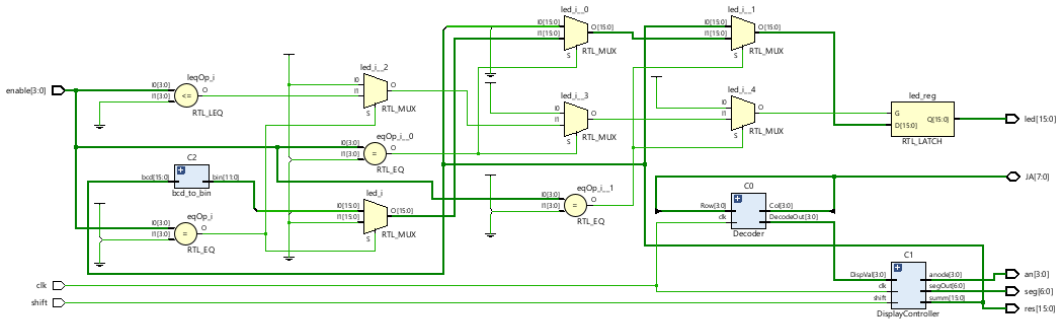

Figure 4: The RTL Schematics of "ky.vhd"

The "dec.vhd" submodule (Figure 5) is a decoder that determines which character (0 to F) the key pressed on the keypad will be assigned to.

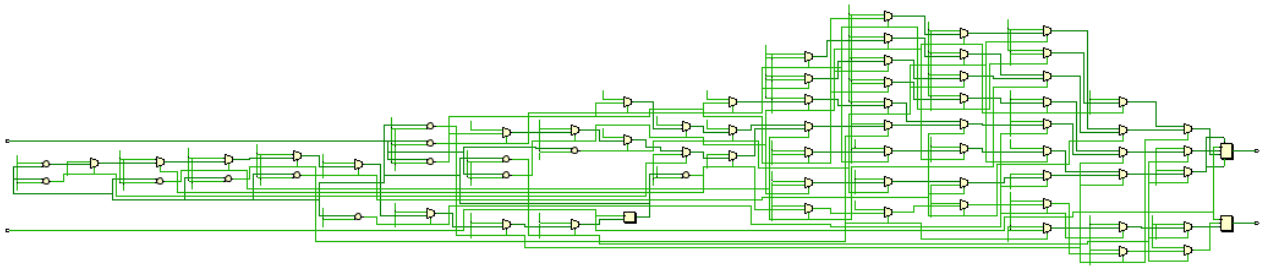

Figure 5: The RTL Schematics of "dec.vhd"

The "ssd.vhd" submodule (Figure 6) takes the character from the decoder as input and outputs it on the seven-segment display. A shift register has been added to this module so that the number to be entered can be two, three, or four digits. After a character is entered, this shift register shifts that character to a left digit and allows a new character to be entered into the right digit when the shift button is pressed.

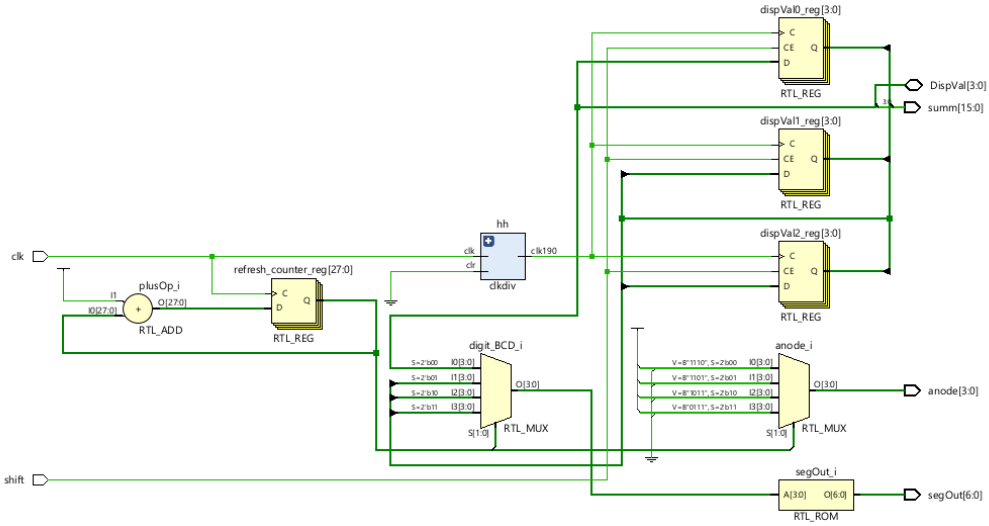

Figure 6: The RTL Schematics of "ssd.vhd"

When the shift button is pressed, BASYS3's clock is too fast for shift register operation because it does this on every rising edge of the clock. To solve this problem, a clock divider with the "clockdiv.vhd" submodule (Figure 7) is used as a clock divider, so the shift operation is set to be performed every half a second.

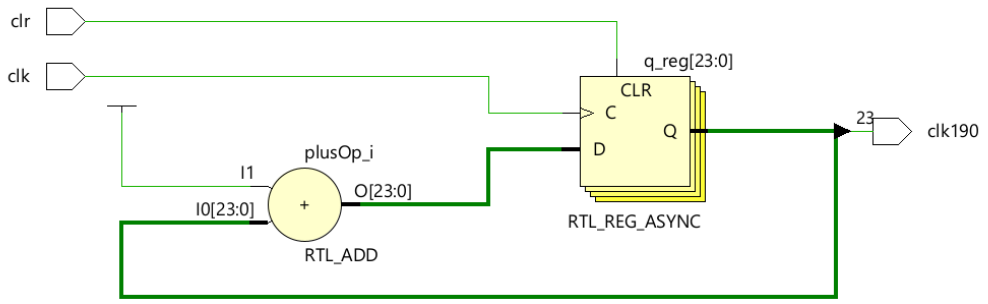

Figure 7: The RTL Schematics of "clockdiv.vhd"

The "conv.vhd" top module (Figure 8) is activated for the last six modes. It displays the number entered with the keypad on the seven-segment display. When the desired mode is selected, the number formed at the end of the necessary operations is output on the seven-segment display again. In other words, it provides the conversion of hexadecimal, octal or decimal base numbers between each other.

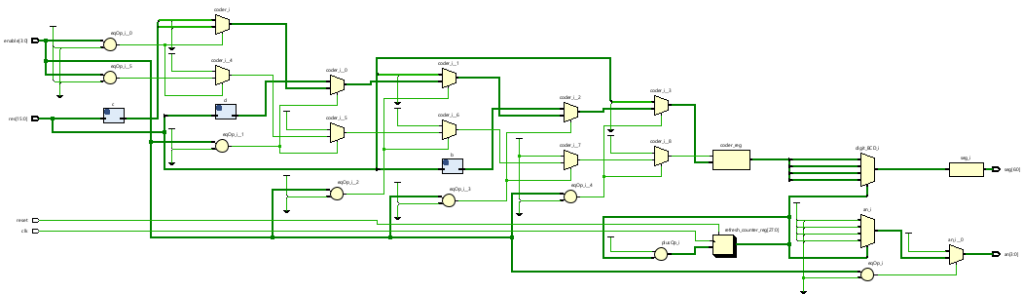

Figure 8: The RTL Schematics of "conv.vhd"

Finally, the “bin_to_bcd.vhd” (Figure 9) and “bcd_to_bin.vhd” (Figure 10) submodules are used to convert binary and decimal base numbers to each other when necessary. Instead of using numbers as decimals in this project, it is preferred to use them as binary-coded decimal (BCD). Because the number in the base of decimal will be displayed on the seven-segment display, it is more appropriate to have it divided into digits as in the base of BCD. These submodules containing the necessary mathematical operations were created to make this conversion.

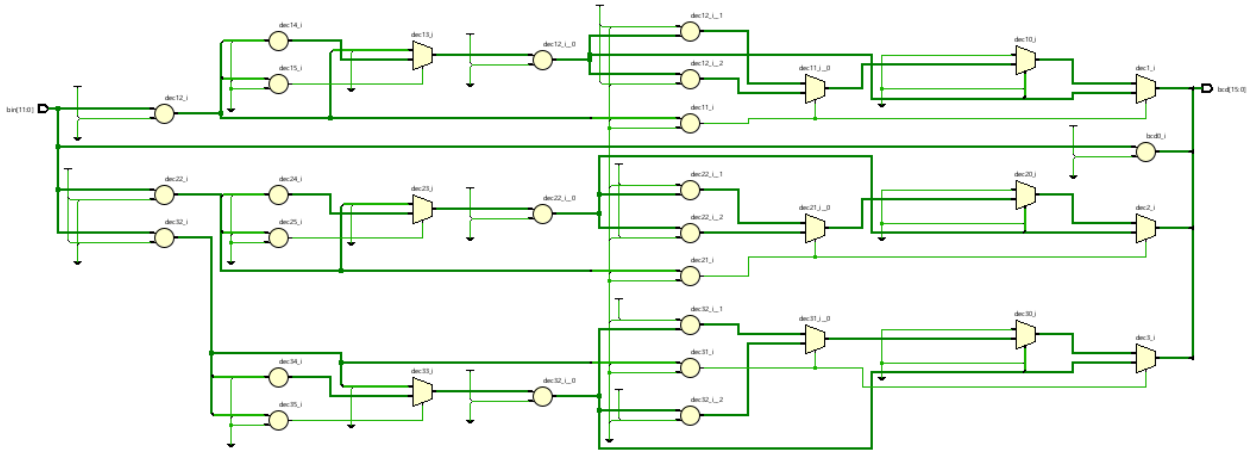

Figure 9: The RTL Schematics of "bin_to_bcd.vhd"

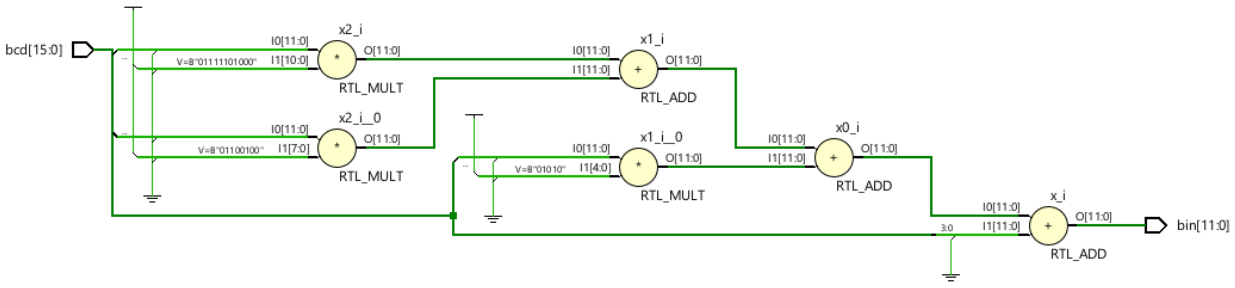

Figure 10: The RTL Schematics of "bcd_to_bin.vhd"

The general view of the created design is as follows (Figure 11):

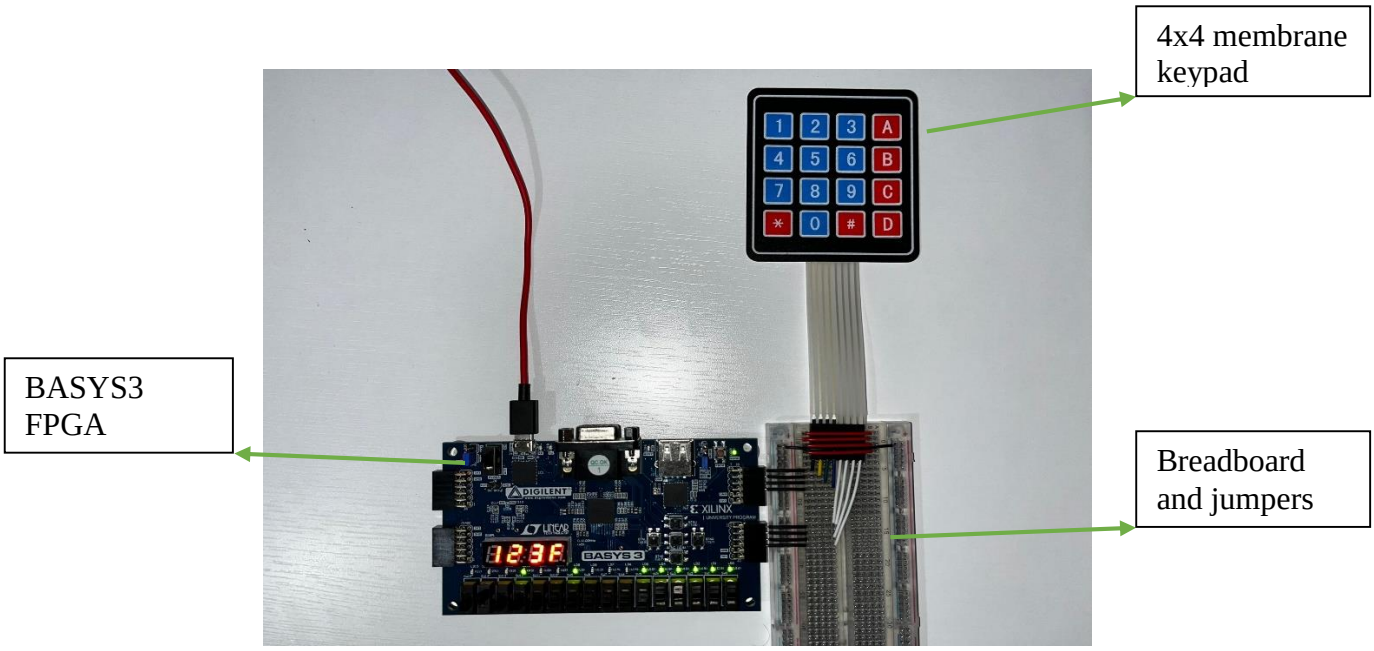

Figure 11: The general view of the design

BASYS3 FPGA was programmed via a constraints file for implementation according to the following assignments (Figure 12):

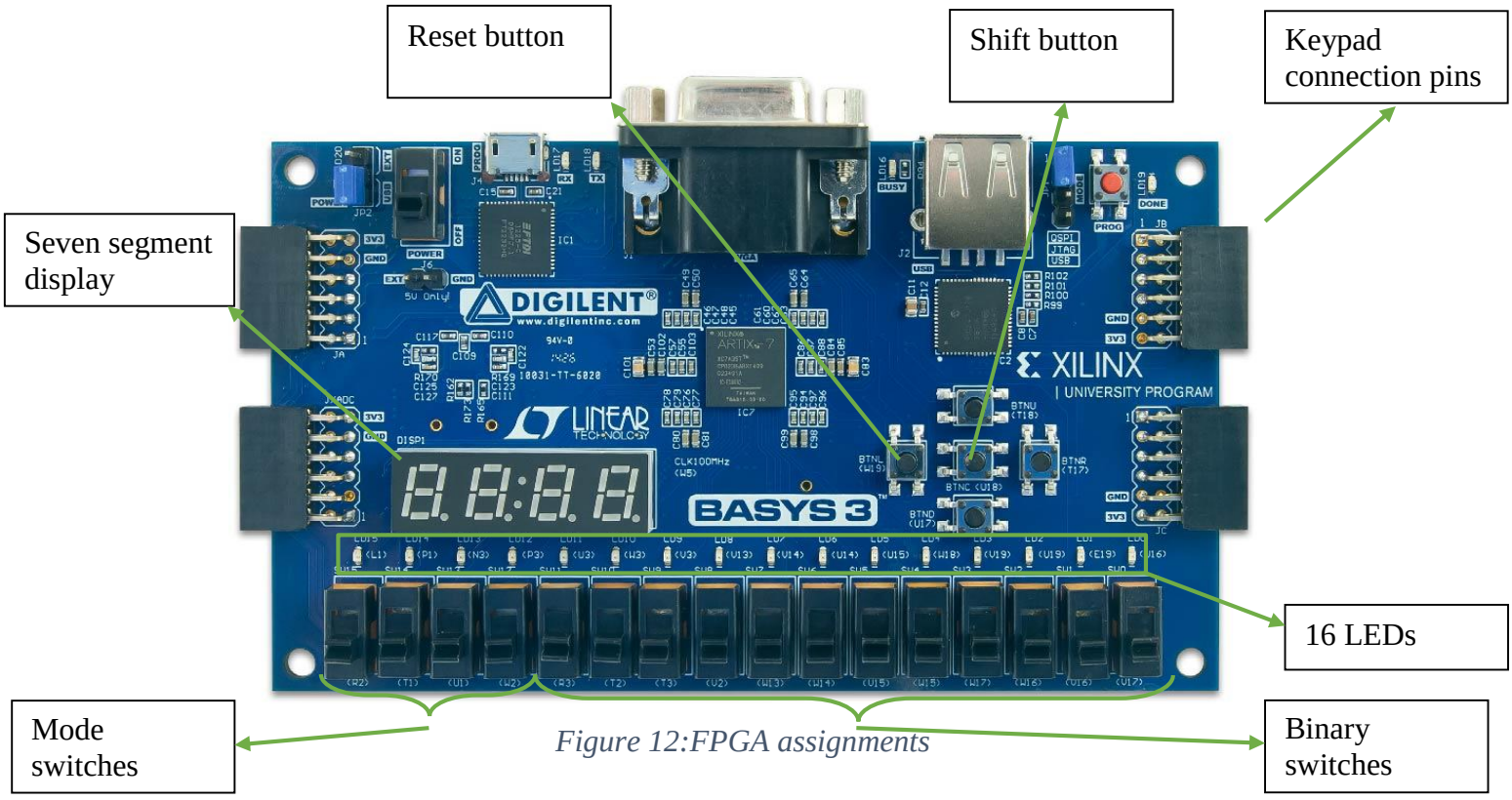

Figure 12:FPGA assignments

Results:

There are four number systems, so there are twelve different combinations of conversions between them. An example for each mode is as follows:

Mode 1: Binary to Hexadecimal

Input = ‘000000011111’

Selection = ‘0001’

Output = ‘1F’

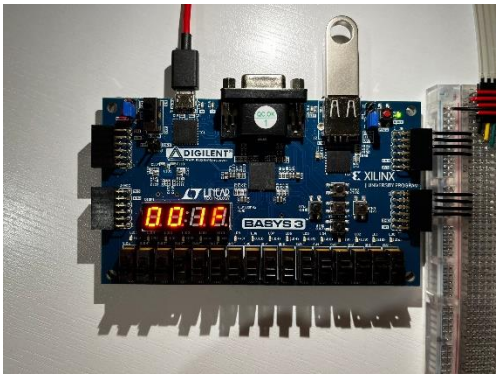

Figure 13

Mode 2: Binary to Octal

Input = ‘000000011111’

Selection = ‘0010’

Output = ‘37’

Figure 14

Mode 3: Binary to Decimal

Input = ‘000000011111’

Selection = ‘0011’

Output = ‘31’

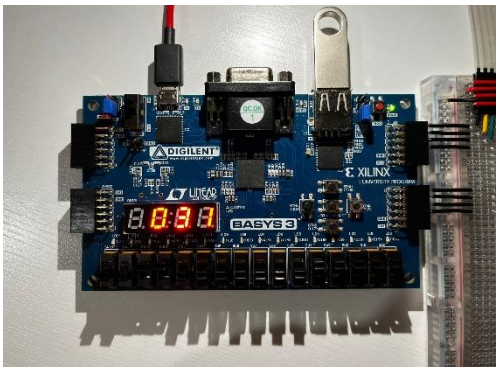

Figure 15

Mode 4: Hexadecimal to Binary

Input = ‘12A’

Selection = ‘0100’

Output = ‘000100101010’

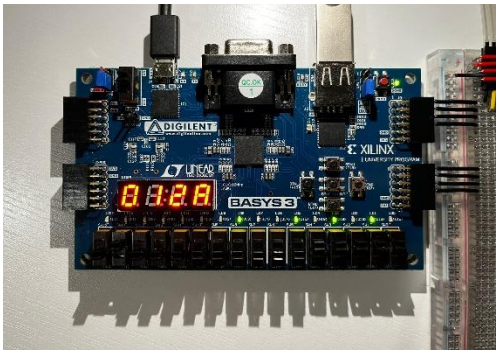

Figure 16

Mode 5: Octal to Binary

Input = ‘4765’

Selection = ‘0101’

Output = ‘100111110101’

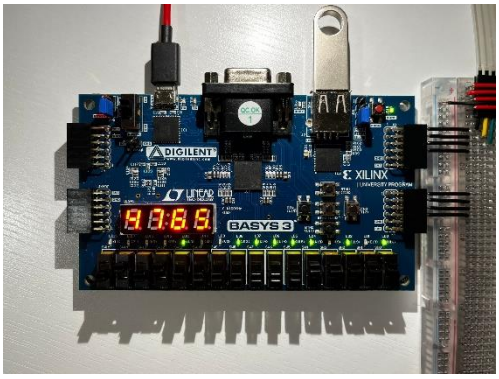

Figure 17

Mode 6: Decimal to Binary

Input = ‘3290’

Selection = ‘0110’

Output = ‘110011011010’

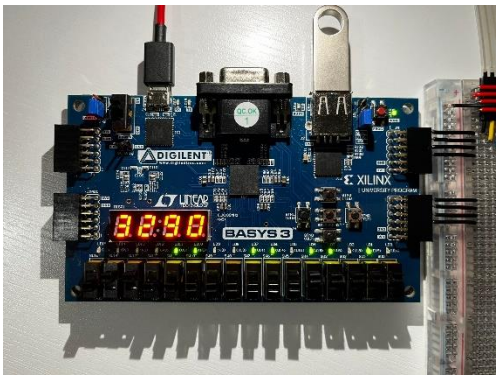

Figure 18

Mode 7: Hexadecimal to Octal

Input = ‘3CD’

Selection = ‘0111’

Output = ‘1715’

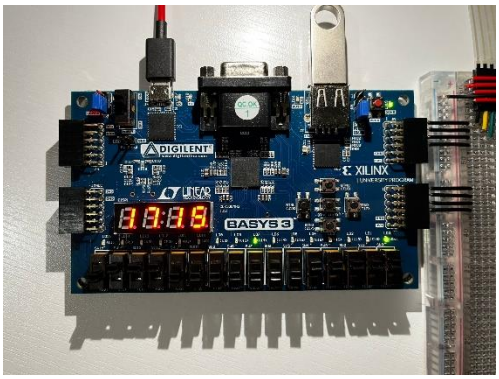

Figure 19

Mode 8: Hexadecimal to Decimal

Input = ‘1EE’

Selection = ‘1000’

Output = ‘494’

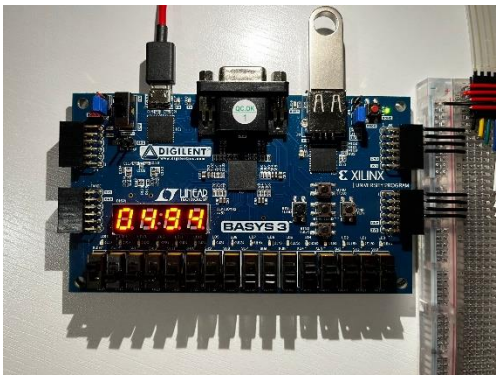

Figure 20

Mode 9: Octal to Hexadecimal

Input = ‘2735’

Selection = ‘1001’

Output = ‘5DD’

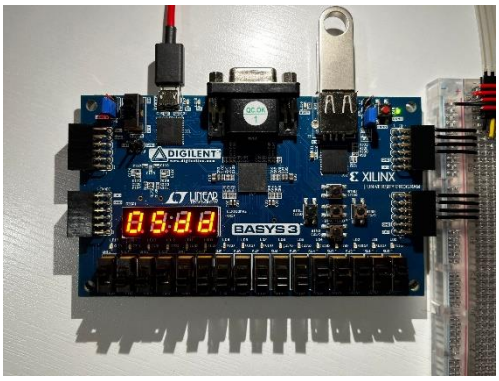

Figure 21

Mode 10: Octal to Decimal

Input = ‘6420’

Selection = ‘1010’

Output = ‘3344’

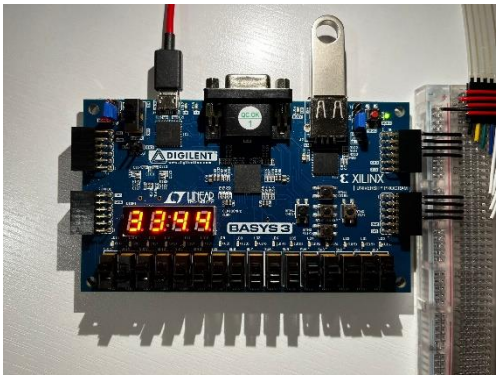

Figure 22

Mode 11: Decimal to Hexadecimal

Input = ‘3884’

Selection = ‘1011’

Output = ‘2FC’

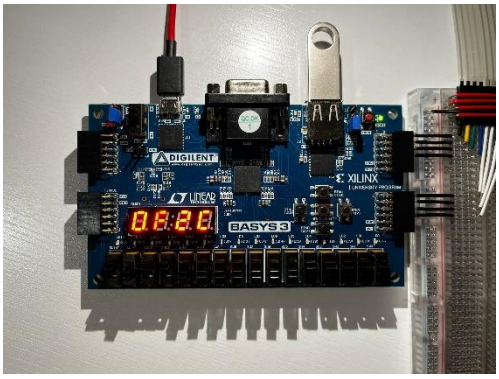

Figure 23

Mode 12: Decimal to Octal

Input = ‘1694’

Selection = ‘1100’

Output = ‘3236’

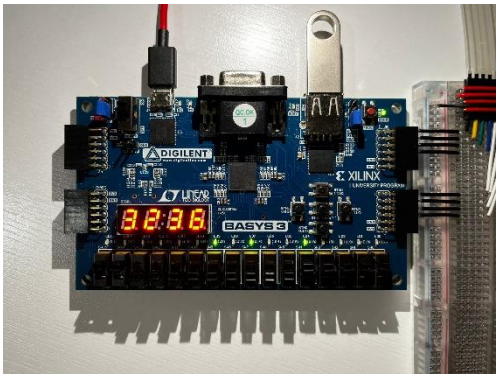

Figure 24

Conclusion:

This project aimed to make a base converter that converts four number systems to each other using BASYS3. Only VHDL was used as a software language in this project. 4x4 Keypad, Breadboard, and Jumper cables were used as components. The biggest problem in the project design was the combination of submodules under the main module. Because in the selected mode, only one submodule had to work and converse. To fix this problem, a "enable" signal was added so that the submodules' activation is specified with a 4-bit number.

With this project, it was learned how to make the desired project with BASYS3 step by step, from design to implementation. In addition, the logic of using the keypad in the project was learned, and the logic of the shift register and clock divider was developed.

After the design was completed, the constraints file was created, and the project was implemented on BASYS3. It was verified that base conversion could be made between the desired numbers in the selected mode. The results were as expected, and no mistakes were observed during the project. Only cable contact problems occurred when the keypad was connected. These troubles were resolved with the use of a breadboard.

References:

Digilent - 4x4 keypad examples.
https://digilent.com/reference/pmod/pmodkypd/start?redirect=1

Digilent – Basys3 Manual.
https://digilent.com/reference/_media/basys3:basys3_rm.pdf.

Appendix:

top.vhd:

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

entity top is
    Port (  clk: in std_logic;
            shift: in std_logic;
            selection: in std_logic_vector(3 downto 0);
            reset: in std_logic;
            sw: in std_logic_vector(11 downto 0);
            JA : inout  STD_LOGIC_VECTOR (7 downto 0);
            led : out STD_LOGIC_VECTOR (15 downto 0);
            an: out std_logic_vector (3 downto 0);
            seg: buffer std_logic_vector (6 downto 0));
end top;

architecture Behavioral of top is
```

```
signal an1, an2, an3: std_logic_vector (3 downto 0);
signal seg1, seg2, seg3: std_logic_vector (6 downto 0);
signal enable: std_logic_vector (3 downto 0);
signal res: std_logic_vector(15 downto 0);
begin

   BtoH_BtoD_BtoO: entity work.bin_to port map (clk, enable, reset, sw, an1, seg1);
   HtoB_OtoB_DtoB: entity work.PmodKYPD port map (clk, shift, enable, res, led, JA, an2, seg2);
   converters:     entity work.conv port map(clk, enable, reset, res, an3, seg3);
process begin
if selection = "0001" then     -- B to H
   enable <= "0001";
   an <= an1;
   seg <= seg1;
elsif selection = "0010" then  -- B to O
   enable <= "0010";
   an <= an1;
   seg <= seg1;
elsif selection = "0011" then -- B to D
   enable <= "0011";
   an <= an1;
   seg <= seg1;
elsif selection = "0100" then -- H to B
   enable <= "0100";
   an <= an2;
   seg <= seg2;
elsif selection = "0101" then -- O to B
   enable <= "0101";
   an <= an2;
   seg <= seg2;
```

```
elsif selection = "0110" then -- D to B
    enable <= "0110";
    an <= an2;
    seg <= seg2;
elsif selection = "0111" then -- H to O
    enable <= "0111";
    an <= an3;
    seg <= seg3;
elsif selection = "1000" then -- H to D
    enable <= "1000";
    an <= an3;
    seg <= seg3;
elsif selection = "1001" then -- O to H
    enable <= "1001";
    an <= an3;
    seg <= seg3;
elsif selection = "1010" then -- O to D
    enable <= "1010";
    an <= an3;
    seg <= seg3;
elsif selection = "1011" then -- D to H
    enable <= "1011";
    an <= an3;
    seg <= seg3;
elsif selection = "1100" then -- D to O
    enable <= "1100";
    an <= an3;
    seg <= seg3;
elsif selection = "0000" then -- NONE
    enable <= "0000";
    an <= "1111";
```

```
end if;
end process;
end Behavioral;
```

bin_to.vhd:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity bin_to is
 Port ( clk: in std_logic;
        enable: in std_logic_vector(3 downto 0);
        reset: in std_logic;
        sw: in std_logic_vector(11 downto 0);
        an: out std_logic_vector (3 downto 0);--which digit lights up
        seg: buffer std_logic_vector (6 downto 0));--which LEDs light up on the selected digit
end bin_to;
architecture Behavioral of bin_to is
signal digit_BCD: std_logic_vector (3 downto 0);
signal digit_counter: std_logic_vector (1 downto 0);
signal refresh_counter: std_logic_vector(27 downto 0);
signal bin: std_logic_vector(15 downto 0);
signal bcd: std_logic_vector(15 downto 0);
begin
    process(clk,reset)
    begin
        if(reset='1') then
            refresh_counter <= (others => '0');
        elsif (rising_edge(clk)) then
            refresh_counter <= refresh_counter + 1;
```

```
        end if;
    end process;
digit_counter <= refresh_counter(19 downto 18);
b: entity work.bin_to_bcd port map(sw(11 downto 0),bcd);
    process
    begin
        if enable = "0001" then
            bin <= "0000" & sw(11 downto 0);
        elsif enable = "0010" then
            bin <= "0" & sw(11 downto 9) & "0" & sw(8 downto 6) & "0" & sw(5 downto 3) & "0"
& sw(2 downto 0);
        elsif enable = "0011" then
            bin <= bcd;
        end if;
    end process;
    process(digit_counter)
    begin
        case digit_counter is
            when "00" => an <= "1110";
 -- activate LED1 and Deactivate LED2, LED3, LED4
            digit_BCD <= bin(3 downto 0);
            when "01" => an <= "1101";
 -- activate LED2 and Deactivate LED1, LED3, LED4
            digit_BCD <= bin(7 downto 4);
            when "10" => an <= "1011";
 -- activate LED3 and Deactivate LED2, LED1, LED4
            digit_BCD <= bin(11 downto 8);
            when "11" => an <= "0111";
 -- activate LED4 and Deactivate LED2, LED3, LED1
            digit_BCD <= bin(15 downto 12);
        end case;
```

```
        if enable = "0000" then
           an <= "1111";
        end if;
    end process;
digit_BCD <= digit_BCD;
         with digit_BCD select
                seg <=      "1000000" when "0000", --0
                                    "1111001" when "0001", --1
                                    "0100100" when "0010", --2
                                    "0110000" when "0011", --3
                                    "0011001" when "0100", --4
                                    "0010010" when "0101", --5
                                    "0000010" when "0110", --6
                                    "1111000" when "0111", --7
                                    "0000000" when "1000", --8
                                    "0010000" when "1001", --9
                                    "0001000" when "1010", --A
                                    "0000011" when "1011", --B
                                    "1000110" when "1100", --C
                                    "0100001" when "1101", --D
                                    "0000110" when "1110", --E
                                    "0001110" when "1111", --F
                                    "0111111" when others;
end Behavioral;
```

ky.vhd:

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

```
entity PmodKYPD is
   Port ( clk : in  STD_LOGIC;
          shift : in std_logic;
          enable : in STD_LOGIC_VECTOR (3 downto 0);
          res : out std_logic_vector (15 downto 0);
          led : out STD_LOGIC_VECTOR (15 downto 0);
                  JA : inout  STD_LOGIC_VECTOR (7 downto 0); -- PmodKYPD is designed to
be connected to JA
          an : out  STD_LOGIC_VECTOR (3 downto 0);   -- Controls which position of the seven
segment display to display
          seg : out  STD_LOGIC_VECTOR (6 downto 0)); -- digit to display on the seven segment
display
end PmodKYPD;

architecture Behavioral of PmodKYPD is
signal Decode, dispVal, dispVal0, dispVal1, dispVal2: STD_LOGIC_VECTOR (3 downto 0);
signal seg_sum, summ: std_logic_vector (15 downto 0);
signal bin: std_logic_vector (11 downto 0);
begin
          C0: entity work.Decoder port map (clk=>clk, Row =>JA(7 downto 4), Col=>JA(3
downto 0), DecodeOut=> Decode);
          C1: entity work.DisplayController port map (clk=>clk, shift=>shift, summ=>summ,
DispVal=>Decode, anode=>an, segOut=>seg);
   C2: entity work.bcd_to_bin port map (summ, bin);
res <= summ;
   process
   begin
      if enable = "0100" then
         led <= summ;
      elsif enable = "0101" then
```

```
        led <= "0000" & summ(14 downto 12) & summ(10 downto 8) & summ(6 downto 4) &
summ(2 downto 0);
    elsif enable = "0110" then
        led <= "0000" & bin;
    elsif enable <= "0000" then
        led <= "0000000000000000";
    end if;
  end process;
end Behavioral;
```

conv.vhd:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity conv is
 Port ( clk: in std_logic;
    enable: in std_logic_vector(3 downto 0);
    reset: in std_logic;
    res: in std_logic_vector(15 downto 0);
    an: out std_logic_vector (3 downto 0);--which digit lights up
    seg: buffer std_logic_vector (6 downto 0));--which LEDs light up on the selected digit
end conv;
architecture Behavioral of conv is
signal digit_BCD: std_logic_vector (3 downto 0);
signal digit_counter: std_logic_vector (1 downto 0);
signal refresh_counter: std_logic_vector(27 downto 0);
signal coder: std_logic_vector(15 downto 0);
signal bcd, bin, bcd1: std_logic_vector(15 downto 0);
```

```
begin
   process(clk,reset)
   begin
      if(reset='1') then
         refresh_counter <= (others => '0');
      elsif (rising_edge(clk)) then
         refresh_counter <= refresh_counter + 1;
      end if;
   end process;
digit_counter <= refresh_counter(19 downto 18);
b: entity work.bin_to_bcd port map(res(11 downto 0),bcd);
c: entity work.bcd_to_bin port map(res, bin(11 downto 0));
d: entity work.bin_to_bcd port map(res(14 downto 12) & res(10 downto 8) & res(6 downto 4) &
res(2 downto 0),bcd1);
   process
   begin
      if enable = "0111" then     -- H to O
         coder <= "0" & res(11 downto 9) & "0" & res(8 downto 6) & "0" & res(5 downto 3) &
"0" & res(2 downto 0);
      elsif enable = "1000" then  -- H to D
         coder <= bcd;
      elsif enable = "1001" then  -- O to H
         coder <= "0000" & res(14 downto 12) & res(10 downto 8) & res(6 downto 4) & res(2
downto 0);
      elsif enable = "1010" then  -- O to D
         coder <= bcd1;
      elsif enable = "1011" then  -- D to H
         coder <= "0000" & bin(11 downto 0);
      elsif enable = "1100" then  -- D to O
         coder <= "0" & bin(11 downto 9) & "0" & bin(8 downto 6) & "0" & bin(5 downto 3) &
"0" & bin(2 downto 0);
```

```
        end if;
    end process;
    process(digit_counter)
    begin
        case digit_counter is
            when "00" => an <= "1110";
 -- activate LED1 and Deactivate LED2, LED3, LED4
            digit_BCD <= coder(3 downto 0);
            when "01" => an <= "1101";
 -- activate LED2 and Deactivate LED1, LED3, LED4
            digit_BCD <= coder(7 downto 4);
            when "10" => an <= "1011";
 -- activate LED3 and Deactivate LED2, LED1, LED4
            digit_BCD <= coder(11 downto 8);
            when "11" => an <= "0111";
 -- activate LED4 and Deactivate LED2, LED3, LED1
            digit_BCD <= coder(15 downto 12);
        end case;
            if enable = "0000" then
               an <= "1111";
            end if;
    end process;
digit_BCD <= digit_BCD;
            with digit_BCD select
                    seg <=      "1000000" when "0000", --0
                                          "1111001" when "0001", --1
                                          "0100100" when "0010", --2
                                          "0110000" when "0011", --3
                                          "0011001" when "0100", --4
                                          "0010010" when "0101", --5
                                          "0000010" when "0110", --6
```

```
                                        "1111000" when "0111", --7
                                        "0000000" when "1000", --8
                                        "0010000" when "1001", --9
                                        "0001000" when "1010", --A
                                        "0000011" when "1011", --B
                                        "1000110" when "1100", --C
                                        "0100001" when "1101", --D
                                        "0000110" when "1110", --E
                                        "0001110" when "1111", --F
                                        "0111111" when others;
end Behavioral;
```

dec.vhd:

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

entity Decoder is
    Port ( clk : in  STD_LOGIC;
           Row : in  STD_LOGIC_VECTOR (3 downto 0);
                     Col : out  STD_LOGIC_VECTOR (3 downto 0);
           DecodeOut : out  STD_LOGIC_VECTOR (3 downto 0));
end Decoder;

architecture Behavioral of Decoder is

signal sclk :STD_LOGIC_VECTOR(19 downto 0);
begin
            process(clk)
```

```
begin
if clk'event and clk = '1' then
        -- 1ms
        if sclk = "00011000011010100000" then
                --C1
                Col<= "0111";
                sclk <= sclk+1;
        -- check row pins
        elsif sclk = "00011000011010101000" then
                --R1
                if Row = "0111" then
                        DecodeOut <= "0001";          --1
                --R2
                elsif Row = "1011" then
                        DecodeOut <= "0100"; --4
                --R3
                elsif Row = "1101" then
                        DecodeOut <= "0111"; --7
                --R4
                elsif Row = "1110" then
                        DecodeOut <= "0000"; --0
                end if;
                sclk <= sclk+1;
        -- 2ms
        elsif sclk = "00110000110101000000" then
                --C2
                Col<= "1011";
                sclk <= sclk+1;
        -- check row pins
        elsif sclk = "00110000110101001000" then
                --R1
```

```
            if Row = "0111" then
                  DecodeOut <= "0010"; --2
            --R2
            elsif Row = "1011" then
                  DecodeOut <= "0101"; --5
            --R3
            elsif Row = "1101" then
                  DecodeOut <= "1000"; --8
            --R4
            elsif Row = "1110" then
                  DecodeOut <= "1111"; --F
            end if;
            sclk <= sclk+1;
--3ms
elsif sclk = "01001001001111100000" then
            --C3
            Col<= "1101";
            sclk <= sclk+1;
-- check row pins
elsif sclk = "01001001001111101000" then
            --R1
            if Row = "0111" then
                  DecodeOut <= "0011"; --3
            --R2
            elsif Row = "1011" then
                  DecodeOut <= "0110"; --6
            --R3
            elsif Row = "1101" then
                  DecodeOut <= "1001"; --9
            --R4
            elsif Row = "1110" then
```

```
                        DecodeOut <= "1110"; --E
                    end if;
                    sclk <= sclk+1;
                --4ms
                elsif sclk = "01100001101010000000" then
                    --C4
                    Col<= "1110";
                    sclk <= sclk+1;
                -- check row pins
                elsif sclk = "01100001101010001000" then
                    --R1
                    if Row = "0111" then
                        DecodeOut <= "1010"; --A
                    --R2
                    elsif Row = "1011" then
                        DecodeOut <= "1011"; --B
                    --R3
                    elsif Row = "1101" then
                        DecodeOut <= "1100"; --C
                    --R4
                    elsif Row = "1110" then
                        DecodeOut <= "1101"; --D
                    end if;
                    sclk <= "00000000000000000000";
                else
                    sclk <= sclk+1;
                end if;
            end if;
        end process;
end Behavioral;
```

ssd.vhd:

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


entity DisplayController is
    Port ( clk: in std_logic;
           shift: in std_logic;
           summ: out std_logic_vector(15 downto 0);
           DispVal : inout  STD_LOGIC_VECTOR (3 downto 0);--output from the Decoder
                     anode: out std_logic_vector(3 downto 0);--controls the display digits
                     segOut : out  STD_LOGIC_VECTOR (6 downto 0));--controls which digit to
display
end DisplayController;

architecture Behavioral of DisplayController is
--begin
           -- only display the leftmost digit
           --anode<="1110";
signal digit_BCD: std_logic_vector (3 downto 0);
signal digit_counter: std_logic_vector (1 downto 0);
signal refresh_counter: std_logic_vector(27 downto 0);
signal Dispsum: std_logic_vector(15 downto 0);
signal dispVal0, dispVal1, dispVal2, dispVal3: std_logic_vector(3 downto 0);
signal clk190: std_logic;
begin
    process(clk)
```

```
    begin
        if(rising_edge(clk)) then
            refresh_counter <= refresh_counter + 1;
        end if;
    end process;
digit_counter <= refresh_counter(19 downto 18);
hh: entity work.clkdiv port map (clk,'0',clk190);
    process
    begin
        if rising_edge(clk190) and shift = '1' then
            dispVal0 <= DispVal;
            dispVal1 <= dispVal0;
            dispVal2 <= dispVal1;
        end if;
        end process;
    process(digit_counter)
    begin
        case digit_counter is
            when "00" => anode <= "1110";
 -- activate LED1 and Deactivate LED2, LED3, LED4
            digit_BCD <= DispVal;
            when "01" => anode <= "1101";
 -- activate LED2 and Deactivate LED1, LED3, LED4
            digit_BCD <= DispVal0;
            when "10" => anode <= "1011";
 -- activate LED3 and Deactivate LED2, LED1, LED4
            digit_BCD <= DispVal1;
            when "11" => anode <= "0111";
 -- activate LED4 and Deactivate LED2, LED3, LED1
            digit_BCD <= DispVal2;
        end case;
```

```
    end process;
digit_BCD <= digit_BCD;
summ <= dispVal2 & dispVal1 & dispVal0 & DispVal;
        with digit_BCD select
                segOut <=     "1000000" when "0000", --0
                                        "1111001" when "0001", --1
                                        "0100100" when "0010", --2
                                        "0110000" when "0011", --3
                                        "0011001" when "0100", --4
                                        "0010010" when "0101", --5
                                        "0000010" when "0110", --6
                                        "1111000" when "0111", --7
                                        "0000000" when "1000", --8
                                        "0010000" when "1001", --9
                                        "0001000" when "1010", --A
                                        "0000011" when "1011", --B
                                        "1000110" when "1100", --C
                                        "0100001" when "1101", --D
                                        "0000110" when "1110", --E
                                        "0001110" when "1111", --F
                                        "0111111" when others;

end Behavioral;
```

clockdiv.vhd:

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

```
entity clkdiv is
Port ( clk: in std_logic ;
       clr: in std_logic ;
       clk190: out std_logic);
       --clk178: out std_logic_vector(1 downto 0 ));
end clkdiv;

architecture clkdiv of clkdiv is
signal q : std_logic_vector (23 downto 0);

begin

process(clk,clr)
begin

if clr = '1' then
   q <= x"000000";
elsif clk'event and clk = '1' then
   q <= q+1;
end if;
end process;
clk190 <= q(23);
--clk178 <= q(18 downto 17);
end clkdiv;
```

bin_to_bcd.vhd:

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

```
use IEEE.numeric_std.all;
use IEEE.std_logic_arith;

entity bin_to_bcd is
    Port ( bin: in std_logic_vector(11 downto 0);
           bcd: out std_logic_vector(15 downto 0));
           --dec : inout integer;
           --dec0: inout integer;
           --dec1: inout integer;
           --dec2: inout integer;
           --dec3: inout integer;
           --dec0v: inout std_logic_vector(3 downto 0);
           --dec1v: inout std_logic_vector(3 downto 0);
           --dec2v: inout std_logic_vector(3 downto 0);
           --dec3v: inout std_logic_vector(3 downto 0);

end bin_to_bcd;

architecture Behavioral of bin_to_bcd is
signal dec, dec0, dec1, dec2, dec3: integer;
signal dec0v, dec1v, dec2v, dec3v: std_logic_vector(3 downto 0);
begin
dec <= to_integer(unsigned(bin));
dec0 <= dec mod 10;
dec1 <= (dec / 10) mod 10;
dec2 <= (dec / 100) mod 10;
dec3 <= (dec / 1000) mod 10;
dec0v <= std_logic_vector(to_unsigned(dec0, 4));
dec1v <= std_logic_vector(to_unsigned(dec1, 4));
dec2v <= std_logic_vector(to_unsigned(dec2, 4));
dec3v <= std_logic_vector(to_unsigned(dec3, 4));
```

```
bcd <= dec3v & dec2v & dec1v & dec0v;
end Behavioral;
```

bcd_to_bin.vhd:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_arith;

entity bcd_to_bin is
    Port ( bcd: in std_logic_vector(15 downto 0);
           bin: out std_logic_vector(11 downto 0));
           --dec : inout integer;
           --bin0: inout integer;
           --bin1: inout integer;
           --bin2: inout integer;
           --bin3: inout integer);
           --dec0v: inout std_logic_vector(3 downto 0);
           --dec1v: inout std_logic_vector(3 downto 0);
           --dec2v: inout std_logic_vector(3 downto 0);
           --dec3v: inout std_logic_vector(3 downto 0);

end bcd_to_bin;

architecture Behavioral of bcd_to_bin is
signal x, bin0, bin1, bin2, bin3: integer;

begin
bin0 <= to_integer(unsigned(bcd(3 downto 0)));
```

```
bin1 <= to_integer(unsigned(bcd(7 downto 4)));
bin2 <= to_integer(unsigned(bcd(11 downto 8)));
bin3 <= to_integer(unsigned(bcd(15 downto 12)));
x <= bin3*1000 + bin2*100 + bin1*10 + bin0;
bin <= std_logic_vector(to_unsigned(x,12));
end Behavioral;
```