KOÇ ÜNİVERSİTESİ

ELEC 204 Digital System Design Laboratory

LAB PROJECT REPORT

ARDA DUMAN

Computer Engineering

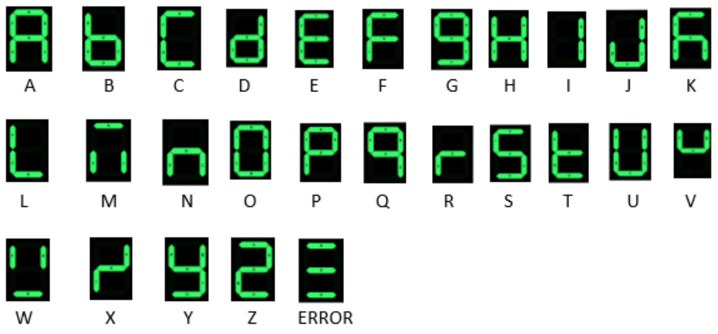
60609

ELEC 204 LAB B

17.05.2019

1. Introduction:

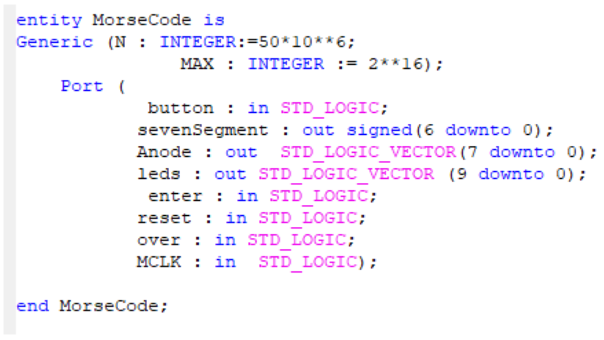
The aim of this project is to design and implement a Morse Code decoder. In this project the user will give the input as Morse Code and the implementation will give the corresponding letter on the seven-segment display.



A	•–	J	•–––	S	•••
B	–•••	K	–•–	T	–
C	–•–•	L	•–••	U	••–
D	–••	M	––	V	•••–
E	•	N	–•	W	•––
F	••–•	O	–––	X	–••–
G	––•	P	•––•	Y	–•––
H	••••	Q	––•–	Z	––••
I	••	R	•–•		

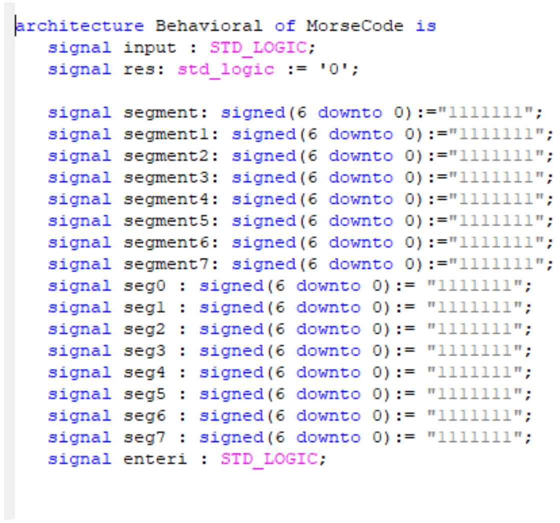
2. Methodology:

This design has 5 inputs and 3 outputs. The input "button" is the button for the Morse Code input. The user is able to make short and long pushes in order to give the dot and dash inputs. Input "over" is for indicating that the Morse Code input is over. When the user pushes this button, the letter will be displayed on the seven-segment display. Input "enter" is implemented in order to shift the displayed letters one anode left. When the user pushes this button the letter sequence will shift a digit left and when it reaches the leftmost anode, the sequence will reset itself. Intput "reset" is implemented in order to reset the sequence anytime and input "MCLK" is the clock. Outputs "SevenSegment" and "Anode" work together in order to display the score on the 7-segmented display. The output "leds" turns on the LEDs when the button for the Morse Code is pushed.



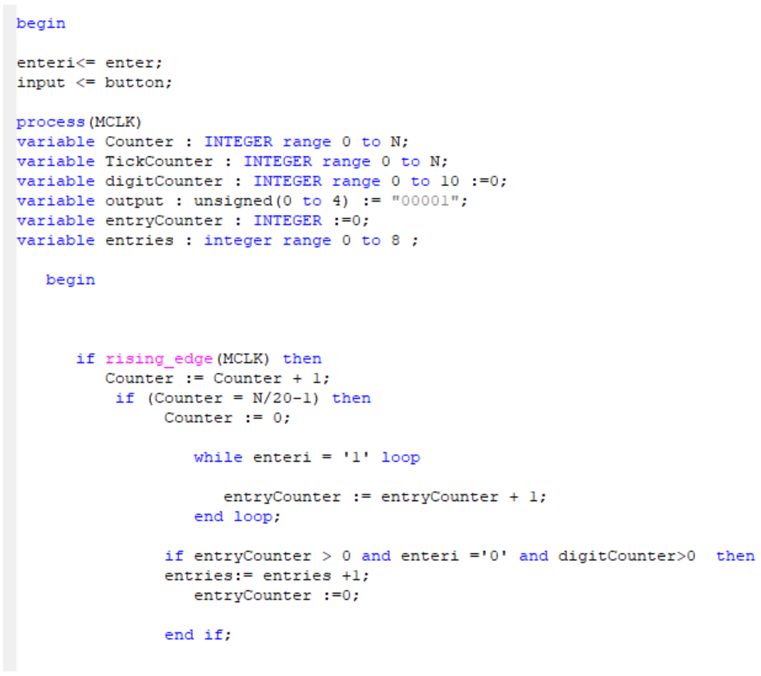
```
entity MorseCode is
Generic (N : INTEGER:=50*10**6;
               MAX : INTEGER := 2**16);
    Port (
            button : in STD_LOGIC;
           sevenSegment : out signed(6 downto 0);
           Anode : out  STD_LOGIC_VECTOR(7 downto 0);
           leds : out STD_LOGIC_VECTOR (9 downto 0);
            enter : in STD_LOGIC;
           reset : in STD_LOGIC;
           over : in STD_LOGIC;
           MCLK : in  STD_LOGIC);

end MorseCode;
```

The intermediate signals. These segment signals are used for the switch operations.



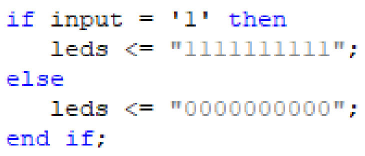
```
architecture Behavioral of MorseCode is
   signal input : STD_LOGIC;
   signal res: std_logic := '0';

   signal segment: signed(6 downto 0):="1111111";
   signal segment1: signed(6 downto 0):="1111111";
   signal segment2: signed(6 downto 0):="1111111";
   signal segment3: signed(6 downto 0):="1111111";
   signal segment4: signed(6 downto 0):="1111111";
   signal segment5: signed(6 downto 0):="1111111";
   signal segment6: signed(6 downto 0):="1111111";
   signal segment7: signed(6 downto 0):="1111111";
   signal seg0 : signed(6 downto 0):= "1111111";
   signal seg1 : signed(6 downto 0):= "1111111";
   signal seg2 : signed(6 downto 0):= "1111111";
   signal seg3 : signed(6 downto 0):= "1111111";
   signal seg4 : signed(6 downto 0):= "1111111";
   signal seg5 : signed(6 downto 0):= "1111111";
   signal seg6 : signed(6 downto 0):= "1111111";
   signal seg7 : signed(6 downto 0):= "1111111";
   signal enteri : STD_LOGIC;
```

The clock is implemented by the help of an input named MCLK which changes its binary value extremely fast. Every time it changes, an integer variable increases, when the variable reaches a certain amount, the variable is set back to 0 and the signal CLK_DIV is inverted, which calls another concurrent process. Here I used an entryCounter in order to know whether the enter button is pushed or not. DigitCounter is implemented in order to keep the length of the Morse Code.



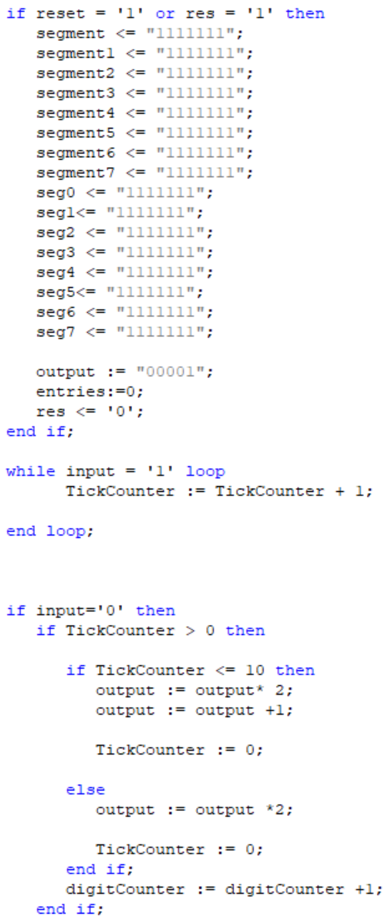
```
begin

enteri<= enter;
input <= button;

process(MCLK)
variable Counter : INTEGER range 0 to N;
variable TickCounter : INTEGER range 0 to N;
variable digitCounter : INTEGER range 0 to 10 :=0;
variable output : unsigned(0 to 4) := "00001";
variable entryCounter : INTEGER :=0;
variable entries : integer range 0 to 8 ;

   begin


      if rising_edge(MCLK) then
         Counter := Counter + 1;
          if (Counter = N/20-1) then
               Counter := 0;

                  while enteri = '1' loop

                     entryCounter := entryCounter + 1;
                  end loop;

               if entryCounter > 0 and enteri ='0' and digitCounter>0  then
               entries:= entries +1;
                  entryCounter :=0;

               end if;
```

This part turns the LEDs on when the Morse Code button is pushed.



```
if input = '1' then
   leds <= "1111111111";
else
   leds <= "0000000000";
end if;
```

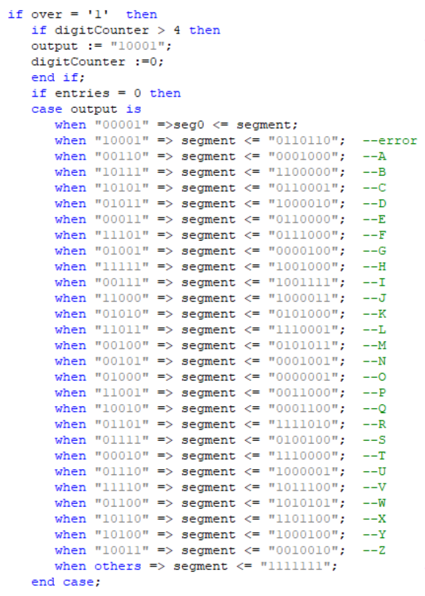
Here is the reset and Morse Code buttons implementations.



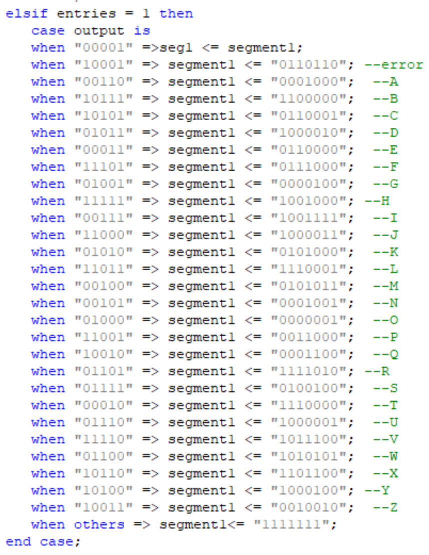
```
if reset = '1' or res = '1' then
   segment <= "1111111";
   segment1 <= "1111111";
   segment2 <= "1111111";
   segment3 <= "1111111";
   segment4 <= "1111111";
   segment5 <= "1111111";
   segment6 <= "1111111";
   segment7 <= "1111111";
   seg0 <= "1111111";
   seg1<= "1111111";
   seg2 <= "1111111";
   seg3 <= "1111111";
   seg4 <= "1111111";
   seg5<= "1111111";
   seg6 <= "1111111";
   seg7 <= "1111111";

   output := "00001";
   entries:=0;
   res <= '0';
end if;

while input = '1' loop
      TickCounter := TickCounter + 1;

end loop;


if input='0' then
   if TickCounter > 0 then

      if TickCounter <= 10 then
         output := output* 2;
         output := output +1;

         TickCounter := 0;

      else
         output := output *2;

         TickCounter := 0;
      end if;
      digitCounter := digitCounter +1;
   end if;
```

In order to know whether it is a dash or dot, I keep another counter (TickCounter) which increases while the Morse Code button is pushed. Then, if the TickCounter is smaller or equal to 10 (it is approximately 0.5 seconds of pushing the button), it represents a dot, otherwise a dash. If it is a dot the “output” is multiplied by two and incremented by 1. This means it shift all the bits of “output” to one bit left and add a 1 behind it. Otherwise just multiplies by 2. That means all the bits are shifted by 1 and a zero is added after the sequence. In conclusion, a 1 represents a dot and a 0 represents a dash. In order to keep track of the leading nonsense zeros, I used a 1. For this reason, the “output” is initially “00001”. So, after the first 1 the sequence will be meaningful.

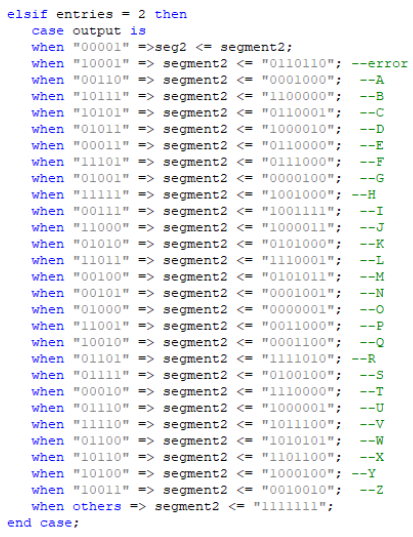
Then the displaying and the shift operations.



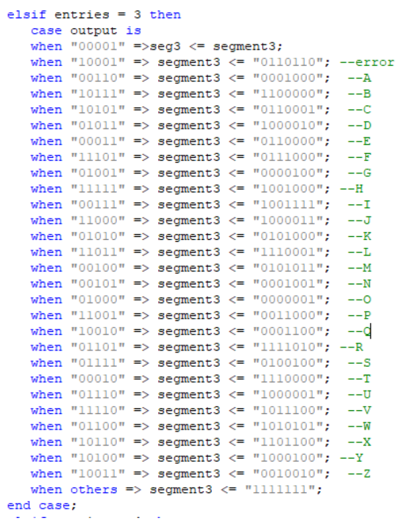
```vhdl
if over = '1'  then
   if digitCounter > 4 then
   output := "10001";
   digitCounter :=0;
   end if;
   if entries = 0 then
   case output is
      when "00001" =>seg0 <= segment;
      when "10001" => segment <= "0110110";  --error
      when "00110" => segment <= "0001000";  --A
      when "10111" => segment <= "1100000";  --B
      when "10101" => segment <= "0110001";  --C
      when "01011" => segment <= "1000010";  --D
      when "00011" => segment <= "0110000";  --E
      when "11101" => segment <= "0111000";  --F
      when "01001" => segment <= "0000100";  --G
      when "11111" => segment <= "1001000";  --H
      when "00111" => segment <= "1001111";  --I
      when "11000" => segment <= "1000011";  --J
      when "01010" => segment <= "0101000";  --K
      when "11011" => segment <= "1110001";  --L
      when "00100" => segment <= "0101011";  --M
      when "00101" => segment <= "0001001";  --N
      when "01000" => segment <= "0000001";  --O
      when "11001" => segment <= "0011000";  --P
      when "10010" => segment <= "0001100";  --Q
      when "01101" => segment <= "1111010";  --R
      when "01111" => segment <= "0100100";  --S
      when "00010" => segment <= "1110000";  --T
      when "01110" => segment <= "1000001";  --U
      when "11110" => segment <= "1011100";  --V
      when "01100" => segment <= "1010101";  --W
      when "10110" => segment <= "1101100";  --X
      when "10100" => segment <= "1000100";  --Y
      when "10011" => segment <= "0010010";  --Z
      when others => segment <= "1111111";
   end case;
```



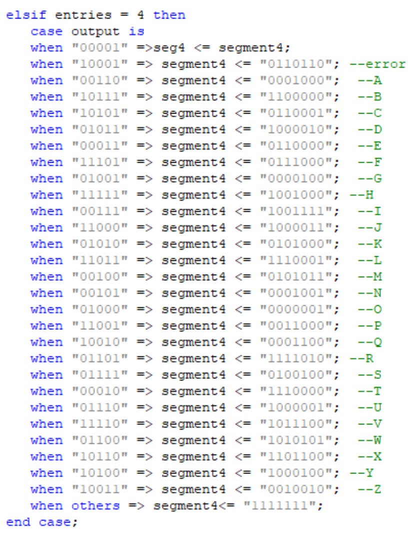
```vhdl
elsif entries = 1 then
   case output is
   when "00001" =>seg1 <= segment1;
   when "10001" => segment1 <= "0110110"; --error
   when "00110" => segment1 <= "0001000";  --A
   when "10111" => segment1 <= "1100000";  --B
   when "10101" => segment1 <= "0110001";  --C
   when "01011" => segment1 <= "1000010";  --D
   when "00011" => segment1 <= "0110000";  --E
   when "11101" => segment1 <= "0111000";  --F
   when "01001" => segment1 <= "0000100";  --G
   when "11111" => segment1 <= "1001000"; --H
   when "00111" => segment1 <= "1001111";  --I
   when "11000" => segment1 <= "1000011";  --J
   when "01010" => segment1 <= "0101000";  --K
   when "11011" => segment1 <= "1110001";  --L
   when "00100" => segment1 <= "0101011";  --M
   when "00101" => segment1 <= "0001001";  --N
   when "01000" => segment1 <= "0000001";  --O
   when "11001" => segment1 <= "0011000";  --P
   when "10010" => segment1 <= "0001100";  --Q
   when "01101" => segment1 <= "1111010"; --R
   when "01111" => segment1 <= "0100100";  --S
   when "00010" => segment1 <= "1110000";  --T
   when "01110" => segment1 <= "1000001";  --U
   when "11110" => segment1 <= "1011100";  --V
   when "01100" => segment1 <= "1010101";  --W
   when "10110" => segment1 <= "1101100";  --X
   when "10100" => segment1 <= "1000100"; --Y
   when "10011" => segment1 <= "0010010";  --Z
   when others => segment1<= "1111111";
end case;
```



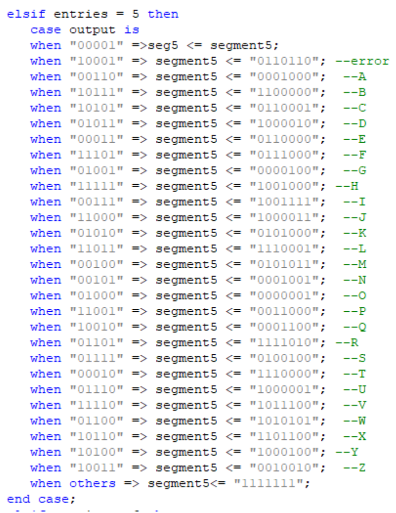
```vhdl
elsif entries = 2 then
   case output is
   when "00001" =>seg2 <= segment2;
   when "10001" => segment2 <= "0110110"; --error
   when "00110" => segment2 <= "0001000";  --A
   when "10111" => segment2 <= "1100000";  --B
   when "10101" => segment2 <= "0110001";  --C
   when "01011" => segment2 <= "1000010";  --D
   when "00011" => segment2 <= "0110000";  --E
   when "11101" => segment2 <= "0111000";  --F
   when "01001" => segment2 <= "0000100";  --G
   when "11111" => segment2 <= "1001000"; --H
   when "00111" => segment2 <= "1001111";  --I
   when "11000" => segment2 <= "1000011";  --J
   when "01010" => segment2 <= "0101000";  --K
   when "11011" => segment2 <= "1110001";  --L
   when "00100" => segment2 <= "0101011";  --M
   when "00101" => segment2 <= "0001001";  --N
   when "01000" => segment2 <= "0000001";  --O
   when "11001" => segment2 <= "0011000";  --P
   when "10010" => segment2 <= "0001100";  --Q
   when "01101" => segment2 <= "1111010"; --R
   when "01111" => segment2 <= "0100100";  --S
   when "00010" => segment2 <= "1110000";  --T
   when "01110" => segment2 <= "1000001";  --U
   when "11110" => segment2 <= "1011100";  --V
   when "01100" => segment2 <= "1010101";  --W
   when "10110" => segment2 <= "1101100";  --X
   when "10100" => segment2 <= "1000100"; --Y
   when "10011" => segment2 <= "0010010";  --Z
   when others => segment2 <= "1111111";
end case;
```



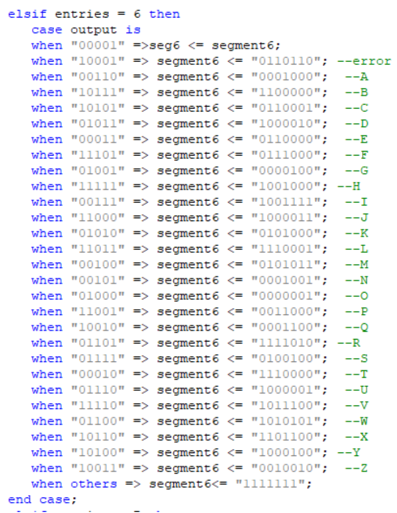
```vhdl
elsif entries = 3 then
   case output is
   when "00001" =>seg3 <= segment3;
   when "10001" => segment3 <= "0110110"; --error
   when "00110" => segment3 <= "0001000";  --A
   when "10111" => segment3 <= "1100000";  --B
   when "10101" => segment3 <= "0110001";  --C
   when "01011" => segment3 <= "1000010";  --D
   when "00011" => segment3 <= "0110000";  --E
   when "11101" => segment3 <= "0111000";  --F
   when "01001" => segment3 <= "0000100";  --G
   when "11111" => segment3 <= "1001000"; --H
   when "00111" => segment3 <= "1001111";  --I
   when "11000" => segment3 <= "1000011";  --J
   when "01010" => segment3 <= "0101000";  --K
   when "11011" => segment3 <= "1110001";  --L
   when "00100" => segment3 <= "0101011";  --M
   when "00101" => segment3 <= "0001001";  --N
   when "01000" => segment3 <= "0000001";  --O
   when "11001" => segment3 <= "0011000";  --P
   when "10010" => segment3 <= "0001100";  --Q
   when "01101" => segment3 <= "1111010"; --R
   when "01111" => segment3 <= "0100100";  --S
   when "00010" => segment3 <= "1110000";  --T
   when "01110" => segment3 <= "1000001";  --U
   when "11110" => segment3 <= "1011100";  --V
   when "01100" => segment3 <= "1010101";  --W
   when "10110" => segment3 <= "1101100";  --X
   when "10100" => segment3 <= "1000100"; --Y
   when "10011" => segment3 <= "0010010";  --Z
   when others => segment3 <= "1111111";
end case;
```



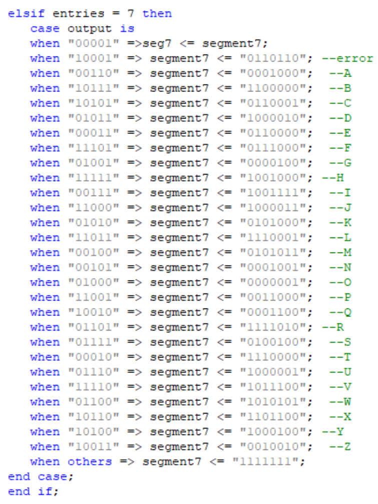
```vhdl
elsif entries = 4 then
   case output is
   when "00001" =>seg4 <= segment4;
   when "10001" => segment4 <= "0110110"; --error
   when "00110" => segment4 <= "0001000";  --A
   when "10111" => segment4 <= "1100000";  --B
   when "10101" => segment4 <= "0110001";  --C
   when "01011" => segment4 <= "1000010";  --D
   when "00011" => segment4 <= "0110000";  --E
   when "11101" => segment4 <= "0111000";  --F
   when "01001" => segment4 <= "0000100";  --G
   when "11111" => segment4 <= "1001000"; --H
   when "00111" => segment4 <= "1001111";  --I
   when "11000" => segment4 <= "1000011";  --J
   when "01010" => segment4 <= "0101000";  --K
   when "11011" => segment4 <= "1110001";  --L
   when "00100" => segment4 <= "0101011";  --M
   when "00101" => segment4 <= "0001001";  --N
   when "01000" => segment4 <= "0000001";  --O
   when "11001" => segment4 <= "0011000";  --P
   when "10010" => segment4 <= "0001100";  --Q
   when "01101" => segment4 <= "1111010"; --R
   when "01111" => segment4 <= "0100100";  --S
   when "00010" => segment4 <= "1110000";  --T
   when "01110" => segment4 <= "1000001";  --U
   when "11110" => segment4 <= "1011100";  --V
   when "01100" => segment4 <= "1010101";  --W
   when "10110" => segment4 <= "1101100";  --X
   when "10100" => segment4 <= "1000100"; --Y
   when "10011" => segment4 <= "0010010";  --Z
   when others => segment4<= "1111111";
end case;
```



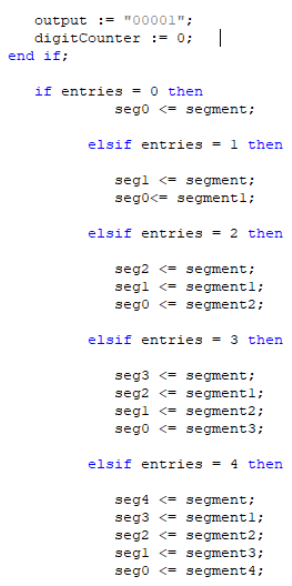
```vhdl
elsif entries = 5 then
   case output is
   when "00001" =>seg5 <= segment5;
   when "10001" => segment5 <= "0110110"; --error
   when "00110" => segment5 <= "0001000";  --A
   when "10111" => segment5 <= "1100000";  --B
   when "10101" => segment5 <= "0110001";  --C
   when "01011" => segment5 <= "1000010";  --D
   when "00011" => segment5 <= "0110000";  --E
   when "11101" => segment5 <= "0111000";  --F
   when "01001" => segment5 <= "0000100";  --G
   when "11111" => segment5 <= "1001000"; --H
   when "00111" => segment5 <= "1001111";  --I
   when "11000" => segment5 <= "1000011";  --J
   when "01010" => segment5 <= "0101000";  --K
   when "11011" => segment5 <= "1110001";  --L
   when "00100" => segment5 <= "0101011";  --M
   when "00101" => segment5 <= "0001001";  --N
   when "01000" => segment5 <= "0000001";  --O
   when "11001" => segment5 <= "0011000";  --P
   when "10010" => segment5 <= "0001100";  --Q
   when "01101" => segment5 <= "1111010"; --R
   when "01111" => segment5 <= "0100100";  --S
   when "00010" => segment5 <= "1110000";  --T
   when "01110" => segment5 <= "1000001";  --U
   when "11110" => segment5 <= "1011100";  --V
   when "01100" => segment5 <= "1010101";  --W
   when "10110" => segment5 <= "1101100";  --X
   when "10100" => segment5 <= "1000100"; --Y
   when "10011" => segment5 <= "0010010";  --Z
   when others => segment5<= "1111111";
end case;
```



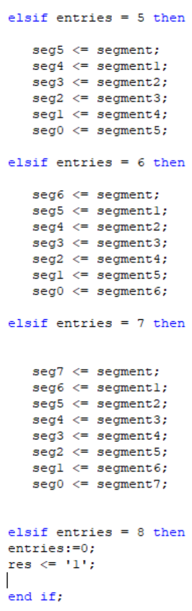
```vhdl
elsif entries = 6 then
   case output is
   when "00001" =>seg6 <= segment6;
   when "10001" => segment6 <= "0110110"; --error
   when "00110" => segment6 <= "0001000";  --A
   when "10111" => segment6 <= "1100000";  --B
   when "10101" => segment6 <= "0110001";  --C
   when "01011" => segment6 <= "1000010";  --D
   when "00011" => segment6 <= "0110000";  --E
   when "11101" => segment6 <= "0111000";  --F
   when "01001" => segment6 <= "0000100";  --G
   when "11111" => segment6 <= "1001000"; --H
   when "00111" => segment6 <= "1001111";  --I
   when "11000" => segment6 <= "1000011";  --J
   when "01010" => segment6 <= "0101000";  --K
   when "11011" => segment6 <= "1110001";  --L
   when "00100" => segment6 <= "0101011";  --M
   when "00101" => segment6 <= "0001001";  --N
   when "01000" => segment6 <= "0000001";  --O
   when "11001" => segment6 <= "0011000";  --P
   when "10010" => segment6 <= "0001100";  --Q
   when "01101" => segment6 <= "1111010"; --R
   when "01111" => segment6 <= "0100100";  --S
   when "00010" => segment6 <= "1110000";  --T
   when "01110" => segment6 <= "1000001";  --U
   when "11110" => segment6 <= "1011100";  --V
   when "01100" => segment6 <= "1010101";  --W
   when "10110" => segment6 <= "1101100";  --X
   when "10100" => segment6 <= "1000100"; --Y
   when "10011" => segment6 <= "0010010";  --Z
   when others => segment6<= "1111111";
end case;
```



```vhdl
elsif entries = 7 then
   case output is
   when "00001" =>seg7 <= segment7;
   when "10001" => segment7 <= "0110110"; --error
   when "00110" => segment7 <= "0001000";  --A
   when "10111" => segment7 <= "1100000";  --B
   when "10101" => segment7 <= "0110001";  --C
   when "01011" => segment7 <= "1000010";  --D
   when "00011" => segment7 <= "0110000";  --E
   when "11101" => segment7 <= "0111000";  --F
   when "01001" => segment7 <= "0000100";  --G
   when "11111" => segment7 <= "1001000"; --H
   when "00111" => segment7 <= "1001111";  --I
   when "11000" => segment7 <= "1000011";  --J
   when "01010" => segment7 <= "0101000";  --K
   when "11011" => segment7 <= "1110001";  --L
   when "00100" => segment7 <= "0101011";  --M
   when "00101" => segment7 <= "0001001";  --N
   when "01000" => segment7 <= "0000001";  --O
   when "11001" => segment7 <= "0011000";  --P
   when "10010" => segment7 <= "0001100";  --Q
   when "01101" => segment7 <= "1111010"; --R
   when "01111" => segment7 <= "0100100";  --S
   when "00010" => segment7 <= "1110000";  --T
   when "01110" => segment7 <= "1000001";  --U
   when "11110" => segment7 <= "1011100";  --V
   when "01100" => segment7 <= "1010101";  --W
   when "10110" => segment7 <= "1101100";  --X
   when "10100" => segment7 <= "1000100"; --Y
   when "10011" => segment7 <= "0010010";  --Z
   when others => segment7 <= "1111111";
end case;
end if;
```

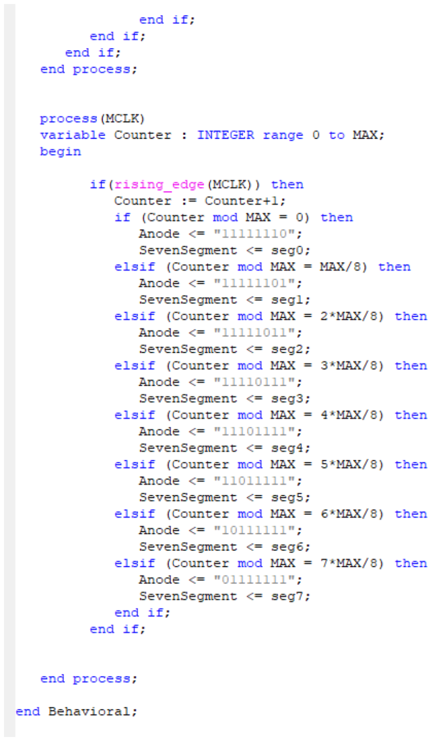


```
    output := "00001";
    digitCounter := 0;
  end if;

    if entries = 0 then
            seg0 <= segment;

        elsif entries = 1 then

            seg1 <= segment;
            seg0<= segment1;

        elsif entries = 2 then

            seg2 <= segment;
            seg1 <= segment1;
            seg0 <= segment2;

        elsif entries = 3 then

            seg3 <= segment;
            seg2 <= segment1;
            seg1 <= segment2;
            seg0 <= segment3;

        elsif entries = 4 then

            seg4 <= segment;
            seg3 <= segment1;
            seg2 <= segment2;
            seg1 <= segment3;
            seg0 <= segment4;
```



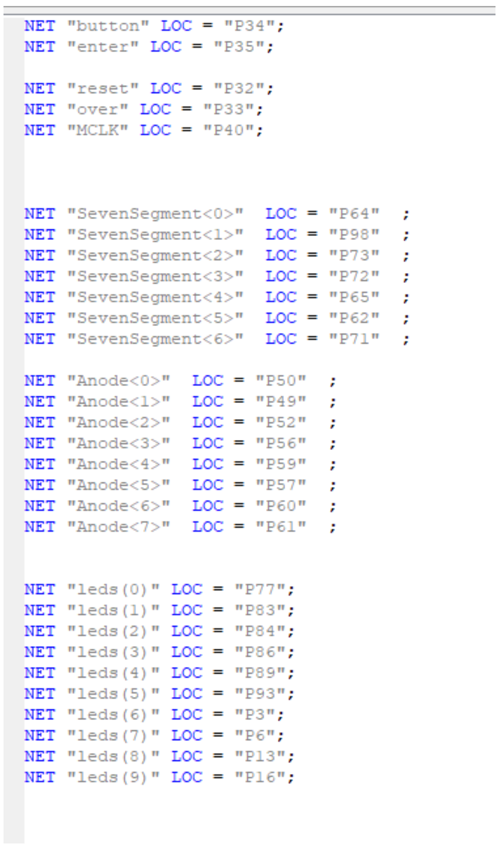
```
elsif entries = 5 then

    seg5 <= segment;
    seg4 <= segment1;
    seg3 <= segment2;
    seg2 <= segment3;
    seg1 <= segment4;
    seg0 <= segment5;

elsif entries = 6 then

    seg6 <= segment;
    seg5 <= segment1;
    seg4 <= segment2;
    seg3 <= segment3;
    seg2 <= segment4;
    seg1 <= segment5;
    seg0 <= segment6;

elsif entries = 7 then


    seg7 <= segment;
    seg6 <= segment1;
    seg5 <= segment2;
    seg4 <= segment3;
    seg3 <= segment4;
    seg2 <= segment5;
    seg1 <= segment6;
    seg0 <= segment7;


elsif entries = 8 then
entries:=0;
res <= '1';

end if;
```

And I used another clock in order to display every anode separately.



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


    process(MCLK)
    variable Counter : INTEGER range 0 to MAX;
    begin

        if(rising_edge(MCLK)) then
            Counter := Counter+1;
            if (Counter mod MAX = 0) then
                Anode <= "11111110";
                SevenSegment <= seg0;
            elsif (Counter mod MAX = MAX/8) then
                Anode <= "11111101";
                SevenSegment <= seg1;
            elsif (Counter mod MAX = 2*MAX/8) then
                Anode <= "11111011";
                SevenSegment <= seg2;
            elsif (Counter mod MAX = 3*MAX/8) then
                Anode <= "11110111";
                SevenSegment <= seg3;
            elsif (Counter mod MAX = 4*MAX/8) then
                Anode <= "11101111";
                SevenSegment <= seg4;
            elsif (Counter mod MAX = 5*MAX/8) then
                Anode <= "11011111";
                SevenSegment <= seg5;
            elsif (Counter mod MAX = 6*MAX/8) then
                Anode <= "10111111";
                SevenSegment <= seg6;
            elsif (Counter mod MAX = 7*MAX/8) then
                Anode <= "01111111";
                SevenSegment <= seg7;
            end if;
        end if;


    end process;

end Behavioral;
```

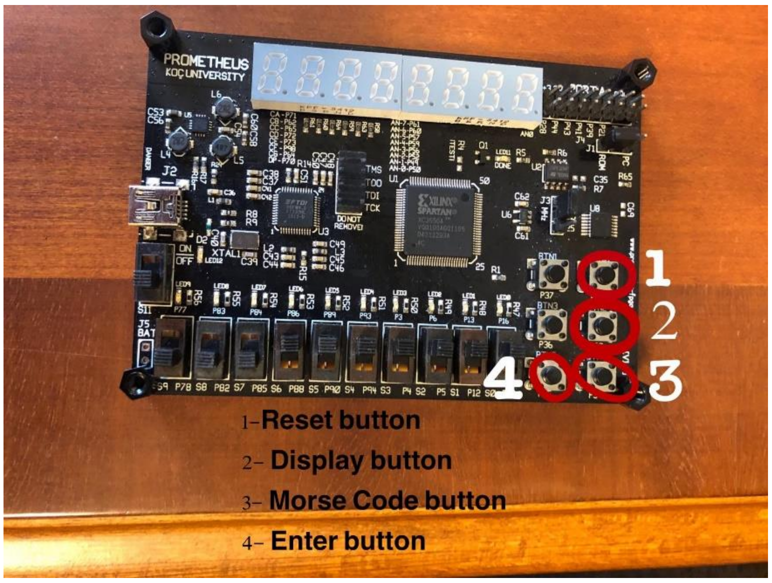
Lastly locations are set.



```
NET "button" LOC = "P34";
NET "enter" LOC = "P35";

NET "reset" LOC = "P32";
NET "over" LOC = "P33";
NET "MCLK" LOC = "P40";


NET "SevenSegment<0>"  LOC = "P64"  ;
NET "SevenSegment<1>"  LOC = "P98"  ;
NET "SevenSegment<2>"  LOC = "P73"  ;
NET "SevenSegment<3>"  LOC = "P72"  ;
NET "SevenSegment<4>"  LOC = "P65"  ;
NET "SevenSegment<5>"  LOC = "P62"  ;
NET "SevenSegment<6>"  LOC = "P71"  ;

NET "Anode<0>"  LOC = "P50"  ;
NET "Anode<1>"  LOC = "P49"  ;
NET "Anode<2>"  LOC = "P52"  ;
NET "Anode<3>"  LOC = "P56"  ;
NET "Anode<4>"  LOC = "P59"  ;
NET "Anode<5>"  LOC = "P57"  ;
NET "Anode<6>"  LOC = "P60"  ;
NET "Anode<7>"  LOC = "P61"  ;


NET "leds(0)" LOC = "P77";
NET "leds(1)" LOC = "P83";
NET "leds(2)" LOC = "P84";
NET "leds(3)" LOC = "P86";
NET "leds(4)" LOC = "P89";
NET "leds(5)" LOC = "P93";
NET "leds(6)" LOC = "P3";
NET "leds(7)" LOC = "P6";
NET "leds(8)" LOC = "P13";
NET "leds(9)" LOC = "P16";
```

3. Experimental Results:

For this experiment, it is not useful to create a simulation. Because we can observe the experiment on the FPGA board easier.



4. Discussion and Conclusion:

In conclusion, with this project I believe I practiced a lot of topics of Elec 204. While doing this project, I think I learned to manipulate the clock input completely since I implemented a code that can differentiate the long and short pushes. At first, in order to use clock, I used a copy of what is given for us on BlackBoard for the lab implementations. But now, for this project I implemented it quite different. I modified it on the way that it can understand the time of a given input. While doing this project, I encountered some problems. For example, it was challenging to find the perfect timing of the dots and dashes. After some calculations and tries I found a good timing.