Hardware Description Languages: Interfacing Peripherals to an FPGA

Kim Camilleri – 345098m

Davis Puku Amankwa – 1606018

MNE3002

2019/2020

Problem Description:

The problem we were assigned was to design a system which would give the distance between an ultrasonic sensor and an obstacle and display that distance on a seven-segment display. If the obstacle was too close, then a buzzer would notify the user by means of an audible alarm.

VHDL Code:

Top Level

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

entity TopDesign is
    Port (
        Anode0 : OUT std_logic;
        Anode1 : OUT std_logic;
        Anode2 : OUT std_logic;
        Anode3 : OUT std_logic;
        seg : OUT std_logic_vector(6 downto 0);
        rst : IN std_logic;
        buzz : OUT STD_LOGIC;
        trig   : OUT std_Logic;
        echo : IN std_logic;
        clk : IN std_logic);
end TopDesign;

architecture Behavioral of TopDesign is

COMPONENT segment_driver
PORT(
        display_A : IN std_logic_vector(3 downto 0);
        display_B : IN std_logic_vector(3 downto 0);
        display_C : IN std_logic_vector(3 downto 0);
        display_D : IN std_logic_vector(3 downto 0);
        Anode3 : OUT std_logic;
        Anode2 : OUT std_logic;
        Anode1 : OUT std_logic;
        Anode0 : OUT std_logic;

        seg : OUT std_logic_vector(6 downto 0);
        clk,rst : IN std_logic);

    END COMPONENT;

COMPONENT sensor
        PORT(

        Trig   : OUT std_logic;
        echo   : IN std_logic;
        distance : OUT std_logic_vector(31 downto 0);
        rst : IN std_logic;
        clk : IN std_logic);
   END COMPONENT;
```

```vhdl
COMPONENT BCD_converter
       PORT (
             distance_input : in  STD_LOGIC_VECTOR (31 downto 0);
             thousands: out  STD_LOGIC_VECTOR (3 downto 0);
             hundreds : out  STD_LOGIC_VECTOR (3 downto 0);
             tens : out  STD_LOGIC_VECTOR (3 downto 0);
             unit : out  STD_LOGIC_VECTOR (3 downto 0));


END COMPONENT;

COMPONENT buzzer

    PORT(
            thousand: IN STD_LOGIC_VECTOR (3 downto 0);
            hundred : IN  STD_LOGIC_VECTOR (3 downto 0);
            ten     : IN  STD_LOGIC_VECTOR (3 downto 0);
            rst     : IN std_logic;
            clk     : IN std_logic;
            buzz    : OUT STD_LOGIC
     );

END COMPONENT;

signal Ai : std_logic_vector(3 downto 0);
signal Bi : std_logic_vector(3 downto 0);
signal Ci : std_logic_vector(3 downto 0);
signal Di : std_logic_vector(3 downto 0);
--signal digitsBinary : std_logic_vector(15 downto 0);
signal distanceBinary  : std_logic_vector(31 downto 0);

begin

uut3: segment_driver PORT MAP(
        display_A => Di,
        display_B => Ci,
        display_C => Bi,
        display_D => Ai,
        seg => seg,

        Anode0 => Anode0,
        Anode1 => Anode1,
        Anode2 => Anode2,
        Anode3 => Anode3,
      rst => rst,
        clk => clk
    );

uut4: sensor PORT MAP(

        Trig => trig,
      echo => echo,
      distance => distanceBinary,
      rst => rst,
        clk => clk
);

uut5: BCD_CONVERTER PORT MAP(

      distance_input => distanceBinary,
        thousands => Ai,
        hundreds =>  Bi,
        tens => Ci,
        unit => Di

);

uut6: buzzer PORT MAP(

    thousand => Ai,
     hundred  =>Bi,
     ten      =>Ci,
      rst =>rst,
      clk => clk ,
      buzz => buzz

);

--Ai <= distanceBinary(15 downto 12);
--Bi <= digitsBinary(11 downto 8);
--Ci <= digitsBinary(7 downto 4);
--Di <= digitsBinary(3 downto 0);
--Ai <= "0101";
--Bi <= "0110";
--Ci <= "0111";
--Di <= "1000";
end Behavioral;
```

BCD Converter Code

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

entity BCD_converter is
    Port ( distance_input : in  STD_LOGIC_VECTOR (31 downto 0);
           thousands: out  STD_LOGIC_VECTOR (3 downto 0);
           hundreds : out  STD_LOGIC_VECTOR (3 downto 0);
           tens : out  STD_LOGIC_VECTOR (3 downto 0);
           unit : out  STD_LOGIC_VECTOR (3 downto 0)
            );
end BCD_converter;

architecture Behavioral of BCD_converter is

begin
process(distance_input)

variable i : integer := 0;
variable bcd : std_logic_vector(67 downto 0);

begin

bcd := (others => '0');
bcd( 31 downto 0) := distance_input;

for i in 0 to 31 loop

bcd(67 downto 0) := bcd(66 downto 0) & '0';

if(i<31 and bcd(67 downto 64) > "0100") then
    bcd(67 downto 64) := bcd(67 downto 64) + "0011";
end if;

if(i<31 and bcd(63 downto 60) > "0100") then
    bcd(63 downto 60) := bcd(63 downto 60) + "0011";
end if;



if(i<31 and bcd(59 downto 56) > "0100") then
    bcd(59 downto 56) := bcd(59 downto 56) + "0011";
end if;



if(i<31 and bcd(55 downto 52)> "0100") then
    bcd(55 downto 52) := bcd(55 downto 52) + "0011";
end if;



if(i<31 and bcd(51 downto 48)> "0100") then
    bcd(51 downto 48) := bcd(51 downto 48) + "0011";
end if;


if(i<31 and bcd(47 downto 44)> "0100") then
    bcd(47 downto 44) := bcd(47 downto 44) + "0011";
```

```
end if;



if(i<31 and bcd(43 downto 40)> "0100") then
    bcd(43 downto 40) := bcd(43 downto 40) + "0011";
end if;



if(i<31 and bcd(39 downto 36)> "0100") then
    bcd(39 downto 36) := bcd(39 downto 36) + "0011";
end if;



if(i<31 and bcd(35 downto 32)> "0100") then
    bcd(35 downto 32) := bcd(35 downto 32) + "0011";

end if;
end loop;

thousands <= bcd(67 downto 64);
hundreds  <=  bcd(63 downto 60);
tens      <= bcd(59 downto 56);
unit      <= bcd(55 downto 52);

end process;
end Behavioral;
```

Sensor Code

```
library IEEE;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_misc.all;
use IEEE.STD_LOGIC_unsigned.ALL;

entity sensor is
        PORT(

        Trig   : OUT std_logic;
        echo   : IN std_logic;
        distance : OUT std_logic_vector(31 downto 0);
        rst : IN std_logic;
        clk : IN std_logic);
end sensor;

architecture Behavioral of sensor is

begin
   sensorproc:process(clk,rst)
        variable counter: std_logic_vector(20 downto 0);
        variable  tempdistance :  std_logic_vector(31 downto 0);
        variable  bcdoutput : std_logic_vector(31 downto 0);
        variable  bcd_counter: std_logic_vector(5 downto 0);
```

```
 begin
        if rst = '1' then
             counter := "000000000000000000000";
             distance <= "00000000000000000000000000000000";
             Trig <= '0';
        elsif clk'event and clk = '1' then
        if counter = "000000000000000000000" then -- counter  = 0
             Trig <= '0';
             counter  := counter + "000000000000000000001";
        elsif counter = "000000000000000000001" then -- counter  = 1
             Trig <= '1';
             counter  := counter + "000000000000000000001";

        elsif counter > "000000000000000000001" and counter <
"000000000000111110100" then  -- counter greater than 0 but less than 500

              counter  := counter + "000000000000000000001";
        elsif counter = "000000000000111110100" then --counter = 500
              Trig <= '0';
              counter  := counter + "000000000000000000001";
        elsif counter > "000000000000111110100" and counter <
"000000111010100110000" then -- counter is greater than 500 but less than
10000
              counter  := counter + "000000000000000000001";
        elsif counter >= "00000010011100010000" and counter <=
"0100110001001011010000" then -- counter is greater than 10000 but less
than 1250000
        --computing range if echo is received
        if echo = '1' then
              tempdistance :=  counter * "100010";
              distance <= tempdistance;
        elsif echo = '0' then
              counter  := counter + "000000000000000000001";
        end if;
        elsif counter > "0100110001001011010000" and counter <
"111001111110111100000" then
              counter  := counter + "000000000000000000001";
        elsif counter = "111001111110111100000" then
              Trig <= '1';
              counter := "000000000000000000000";
              distance <= "00000000000000000000000000000000";
        end if;
    end if;

end process;
end Behavioral;
```

Buzzer Code

```vhdl
library IEEE;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_misc.all;
use IEEE.STD_LOGIC_unsigned.ALL;

entity buzzer is
      PORT(
            thousand: IN STD_LOGIC_VECTOR (3 downto 0);
            hundred : IN  STD_LOGIC_VECTOR (3 downto 0);
            ten : IN  STD_LOGIC_VECTOR (3 downto 0);
            rst : IN std_logic;
            buzz : OUT STD_LOGIC;
            clk: IN STD_LOGIC
     );
end buzzer;

architecture Behavioral of buzzer is


type state_type is (s0,s1,s2,s3); --state0 reset, state1 1sec, state2 2
seconds ,state 3 3 seconds
signal next_state, state : state_type;
begin
    clock:process(clk,rst)
         begin
              if rst = '1' then
                state <= s0;
                elsif clk'event and clk = '1' then
                     state <= next_state;
              end if;
          end process;

    inprocess:process(state,thousand,hundred,ten)
         variable buzz_counter :std_logic_vector(15 downto 0);
         begin

         case state is

               when s0 =>
                    buzz <= '0';
                    buzz_counter := "0000000000000000";
               if thousand >= "0001" then
                    next_state <= s3;
               elsif thousand = "0000" and hundred >= "0001" then
                    next_state <= s2;

               elsif thousand = "0000" and hundred = "0000" and ten >=
               "0001" then
                    next_state <= s1;
               elsif ten < "0001" then
                    next_state <= s0;
                    end if;

               when s1 =>
                    buzz <= '1';

               when s2 =>
                    buzz <= '1';
               when s3 =>
                    buzz <= '1';


               when others =>
                    NULL;

              end case;
            end process;

end Behavioral;
```

Seven-Segment Display

```
library IEEE;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_misc.all;
use IEEE.STD_LOGIC_unsigned.ALL;

entity segment_driver is

PORT(
        display_A : IN std_logic_vector(3 downto 0);
        display_B : IN std_logic_vector(3 downto 0);
        display_C : IN std_logic_vector(3 downto 0);
        display_D : IN std_logic_vector(3 downto 0);
        Anode0  : OUT std_logic;
        Anode1  : OUT std_logic;
        Anode2  : OUT std_logic;
        Anode3  : OUT std_logic;
        rst  : IN std_logic;
        seg  : OUT std_logic_vector(6 downto 0);
        clk  : IN std_logic);
end segment_driver;

architecture Behavioral of segment_driver is

begin

sevenseg:process(clk,rst)
       variable counter: std_logic_vector(19 downto 0);

    begin

         if rst = '1' then

             counter := "00000000000000000000";
             Anode0 <= '1';
             Anode1 <= '1';
             Anode2 <= '1';
             Anode3 <= '1';

         elsif clk'event and clk = '1' then

             counter := counter + "00000000000000000001";

          if counter < "1100001101010000000" then --displaying first digit

                    Anode0 <= '0';
                    Anode1 <= '1';
                    Anode2 <= '1';
                    Anode3 <= '1';

            case display_A IS
                  when "0000" =>
                     seg <= "1000000";

                  when "0001" =>
                     seg <= "1111001";

                  when "0010" =>
                     seg <= "0100100";
                   when "0011" =>
                      seg <= "0110000";

                   when "0100" =>
                      seg <= "0011001";

                   when "0101" =>
                      seg <= "0010010";

                   when "0110" =>
                      seg <= "0000010";

                   when "0111" =>
                      seg <= "1111000";

                   when "1000" =>
                      seg <= "0000000";

                   when "1001" =>
                      seg <= "0011000";

                   when others => NULL;

            end case;

          elsif "110000110101000000" < counter and counter <
"1100001101010000000" then    --displaying second digit

                    Anode0 <= '1';
                    Anode1 <= '0';
                    Anode2 <= '1';
                    Anode3 <= '1';

            case display_B IS
                   when "0000" =>
                      seg <= "1000000";
                   when "0001" =>
                      seg <= "1111001";

                   when "0010" =>
                      seg <= "0100100";

                   when "0011" =>
                      seg <= "0110000";

                   when "0100" =>
                      seg <= "0011001";

                   when "0101" =>
                      seg <= "0010010";

                   when "0110" =>
                      seg <= "0000010";

                   when "0111" =>
                      seg <= "1111000";

                   when "1000" =>
                      seg <= "0000000";

                   when "1001" =>
```

```vhdl
                        seg <= "0011000";
                     when others => NULL;

               end case;

             elsif  "1100001101010000000" < counter  and counter <
"1001001001111000000" then    --displaying third digit

                      Anode0 <= '1';
                      Anode1 <= '1';
                      Anode2 <= '0';
                      Anode3 <= '1';

               case display_C IS
                     when "0000" =>
                        seg <= "1000000";

                     when "0001" =>
                        seg <= "1111001";

                     when "0010" =>
                        seg <= "0100100";

                     when "0011" =>
                        seg <= "0110000";

                     when "0100" =>
                        seg <= "0011001";

                     when "0101" =>
                        seg <= "0010010";

                     when "0110" =>
                        seg <= "0000010";

                     when "0111" =>
                        seg <= "1111000";

                     when "1000" =>
                        seg <= "0000000";

                     when "1001" =>
                        seg <= "0011000";

                     when others => NULL;

               end case;

             elsif "1001001001111000000"< counter  and counter <
"1100001101010000000" then    --displaying fourth digit

                      Anode0 <= '1';
                      Anode1 <= '1';
                      Anode2 <= '1';
                      Anode3 <= '0';

               case display_D IS
                     when "0000" =>
                        seg <= "1000000";

                     when "0001" =>
```

```
                        seg <= "1111001";

                     when "0010" =>
                        seg <= "0100100";

                     when "0011" =>
                        seg <= "0110000";

                     when "0100" =>
                        seg <= "0011001";

                     when "0101" =>
                        seg <= "0010010";

                     when "0110" =>
                        seg <= "0000010";

                     when "0111" =>
                        seg <= "1111000";

                     when "1000" =>
                        seg <= "0000000";

                     when "1001" =>
                        seg <= "0011000";

                     when others => NULL;

                  end case;

            end if;

          end if;

  end process;

  end Behavioral;
```

Oscilloscope Screen Capture:

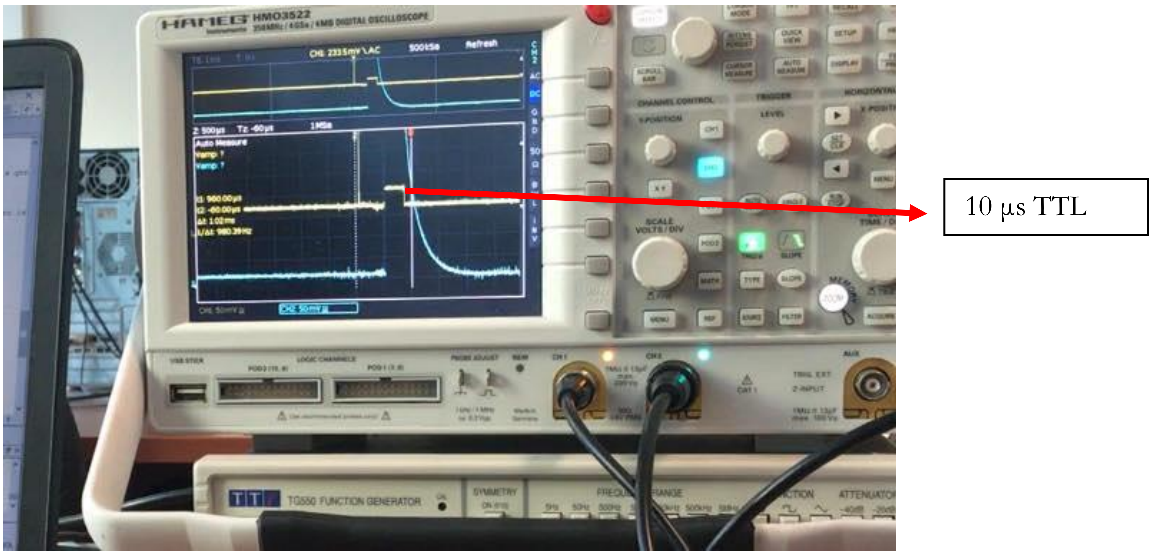

Figure 1: Oscilloscope image of trigger and echo

Assembled circuit:

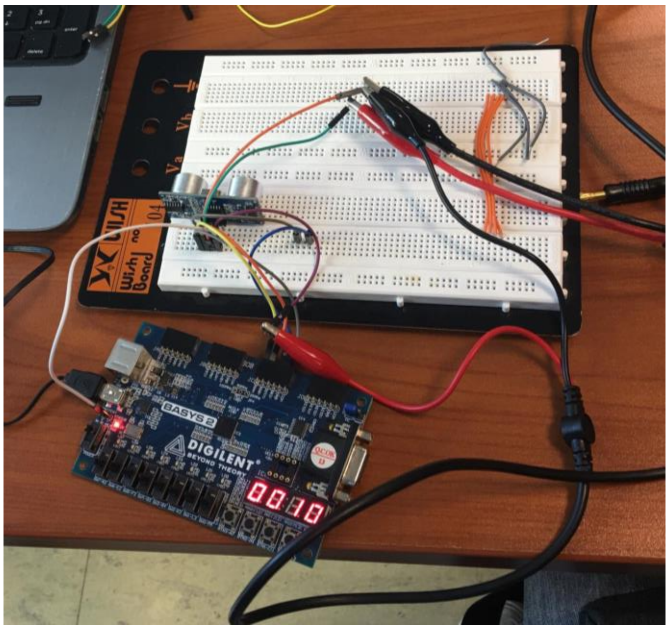

Figure 2: Sensor connected to fpga and oscilloscope