Seven Segment Display – Binary to Unsigned/Signed Decimal

Name: Jonathan J. Rodriguez

CSC343 – Computer Organization Lab, Spring 2020

Instructor: Izidor Gertner

The City College of New York

Table of Contents

Objective

The objective of this laboratory exercise is to use both the DE2 and the newer DE1-SOC FPGA boards in order to display 8-bit binary numbers onto seven segment displays. The use of Quartus 13.0 (for DE2) and Quartus 16.1 (for DE1-SOC) software will allow us to design electronic circuits that utilizes the switches on the boards to create the output that is desired.

Board Specifications & Software Used

1. Altera DE2 Board (Quartus II 13.0 sp1)
2. Altera DE1-SOC Board (Quartus Prime 16.0)

Design Specifications & Functionality

Unsigned/Signed Circuit (Two's Complement)

RODR_UnsignedSigned.vhd

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
USE IEEE.numeric_std.all;

   ENTITY RODR_UnsignedSigned IS
   PORT(RODR_SIGBIT : IN STD_LOGIC;                        -- Switch for Unsigned/Signed
        RODR_IN   : IN STD_LOGIC_VECTOR (7 downto 0) ;     -- Switch Inputs
        RODR_OUT  : OUT STD_LOGIC_VECTOR (7 downto 0);     -- Signed or Unsigned Output
        RODR_LEDS : OUT STD_LOGIC_VECTOR (8 downto 0));    -- LED On/Off
END RODR_UnsignedSigned ;

ARCHITECTURE LogicFunction OF RODR_UnsignedSigned IS

   signal IN_FLIP : STD_LOGIC_VECTOR (7 downto 0);                 --Functionality to Flip all Bits from switch
   signal ADD_ONE : STD_LOGIC_VECTOR (7 downto 0) := "00000001";   -- Add one to Complete "Two's Complement"
   signal SIGNED_OUT : STD_LOGIC_VECTOR (7 downto 0);              -- Temporary signal to store end result

   BEGIN
         IN_FLIP <= NOT RODR_IN;


   SIGNED_OR_UNSIGNED: process(RODR_IN, IN_FLIP, ADD_ONE, SIGNED_OUT, RODR_SIGBIT)  -- Create a process for sequential logic reading, not concurrent

   variable carry : STD_LOGIC;                  -- Initialized to one to add 1 bit to Bit flipped RODR_IN
   BEGIN
      for LEDNUM in 0 to 7 loop                 -- Assignment for switches based on
         RODR_LEDS(LEDNUM) <= RODR_IN(LEDNUM);
      end loop;
      RODR_LEDS(8) <= RODR_SIGBIT;              -- Assigned RODR_SIGBIT to the 8th LED (from the right)

      case RODR_SIGBIT is                       -- Checks if Unsigned/Signed Switch is On/Off

      when '0' =>                               --CASE 1: UNSIGNED
         RODR_OUT <= RODR_IN;

      when '1' =>                            --CASE 2: SIGNED
         if((RODR_IN(7) = '0')) then         -- 0 to 127
            RODR_OUT <= RODR_IN;
         else
            if (IN_FLIP(0) = '0') then             -- Checks to see if IN_FLIP's first bit 0 or 1 and add accordingly
               SIGNED_OUT <= IN_FLIP XOR ADD_ONE;
               carry := '0';
            elsif (IN_FLIP(0) = '1') Then
               SIGNED_OUT <= IN_FLIP XOR ADD_ONE;
               carry := '1';
            end if;
            for i in 1 to 7 loop            --CHECK THROUGH REST OF IN_FLIP ARRAY checking each bit using the carry
               if (IN_FLIP(i) = '0') then
                  SIGNED_OUT(i) <= IN_FLIP(i) XOR carry;
                  carry := '0';
               elsif ((carry = '1') AND (IN_FLIP(i) xor carry) = '0') Then
                  SIGNED_OUT(i) <= IN_FLIP(i) XOR carry;
                  carry := '1';
               end if ;
            end loop ;
```

```
54              end if ;
55            end loop ;
56          RODR_OUT <= SIGNED_OUT;          -- Put temporary array of bits into RODR_OUT
57          end if;
58
59        end case;
60
61    END process Signed_OR_Unsigned ;
62
63  END LogicFunction ;
```

Figure 1: Circuit Decoding the inputs from the switches and determining if two's complement is needed

The first thing that the board is faced with is the Unsigned/Signed circuit. It takes in the inputs from the switches on either the DE2 or DE1-SOC boards and determines if two's complement is needed in order to be displayed correctly on the seven segment displays.

```
1   LIBRARY ieee ;
2   USE ieee.std_logic_1164.all ;
3   USE IEEE.numeric_std.all;
4
```

Libraries

Figure 2: Libraries used in RODR_UnsignedSigned VHDL file

To determine if we need to do such a conversion, we first need to set up our libraries so that we can take in logical inputs from the switches, as ***Figure 2*** shows. We are using *IEEE.std_logic_1164.all* to define our logical inputs. We are also using *IEEE.numeric_std.all* to provide for arithmetic in logical vectors.

```
5   ENTITY RODR_UnsignedSigned IS
6   PORT(RODR_SIGBIT : IN STD_LOGIC;                          -- Switch for Unsigned/Signed
7        RODR_IN   : IN STD_LOGIC_VECTOR (7 downto 0) ;       -- Switch Inputs
8        RODR_OUT  : OUT STD_LOGIC_VECTOR (7 downto 0);       -- Signed or Unsigned Output
9        RODR_LEDS : OUT STD_LOGIC_VECTOR (8 downto 0));      -- LED On/Off
10  END RODR_UnsignedSigned ;
```

Inputs & Outputs

Figure 3: Inputs and Outputs defined

Since the DE2 has 18 switches and the DE1-SOC has 10, we will only be using 9 switches to accommodate for the limitation. **RODR_SIGBIT** is set as the 9th input switch that the user will define as to whether they want the input to be signed or unsigned. **RODR_IN** is set as the first 8 switches on the board (from the right) and will be used to input binary numbers that will eventually be displayed on the seven segment displays. **RODR_OUT** will be the output from the Unsigned/Signed circuit which will be either signed or unsigned. **RODR_LEDS** is the output of the LED's that will be displayed on the board.

```
12  ARCHITECTURE LogicFunction OF RODR_UnsignedSigned IS
13
14      signal IN_FLIP : STD_LOGIC_VECTOR (7 downto 0);              --Functionality to Flip all Bits from switch
15      signal ADD_ONE : STD_LOGIC_VECTOR (7 downto 0) := "00000001"; -- Add one to Complete "Two's Complement"
16      signal SIGNED_OUT : STD_LOGIC_VECTOR (7 downto 0);           -- Temporary signal to store end result
17
18  BEGIN
19          IN_FLIP <= NOT RODR_IN;
20
```

Temp. Variables

Figure 4: Defining temporary variables to store values needed for two's complement and outputs

There are also some assigned variables that will also be used in the case that we need two's complement. Signal **IN_FLIP** is used for automatically inverting **RODR_IN** to start the two's complement process. Signal **ADD_ONE** is used to "add one" to the inversion of the bits

taken in from the board to complete two's complement. Signal **SIGNED_OUT** is a temporary variable that will store the resulting output.

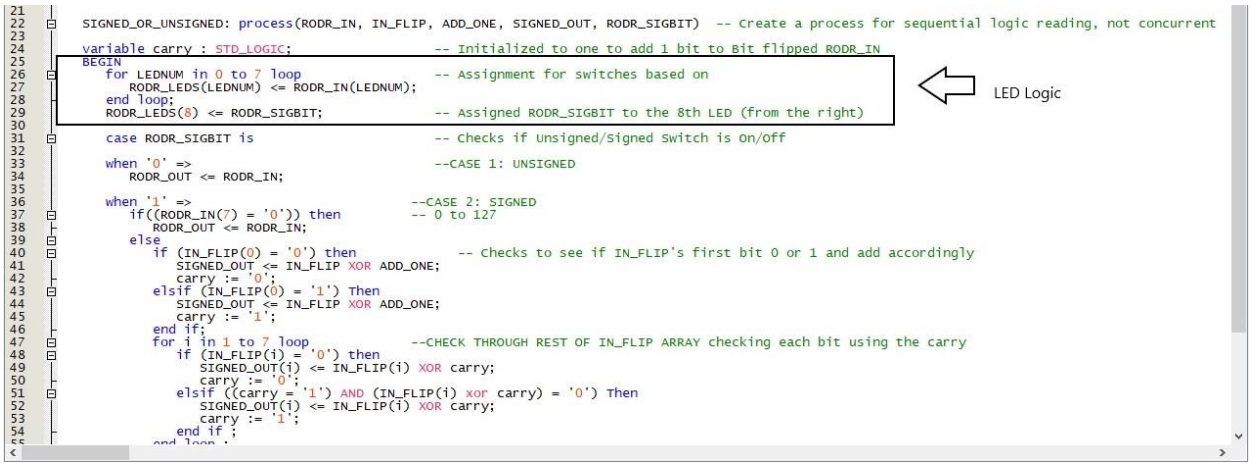


Figure 5: For loop that turns on or off LED's on board

There is a designation of **LEDNUM** that will loop through the LED's and automatically set it to the inputs from the switches (including the signed/unsigned switch).

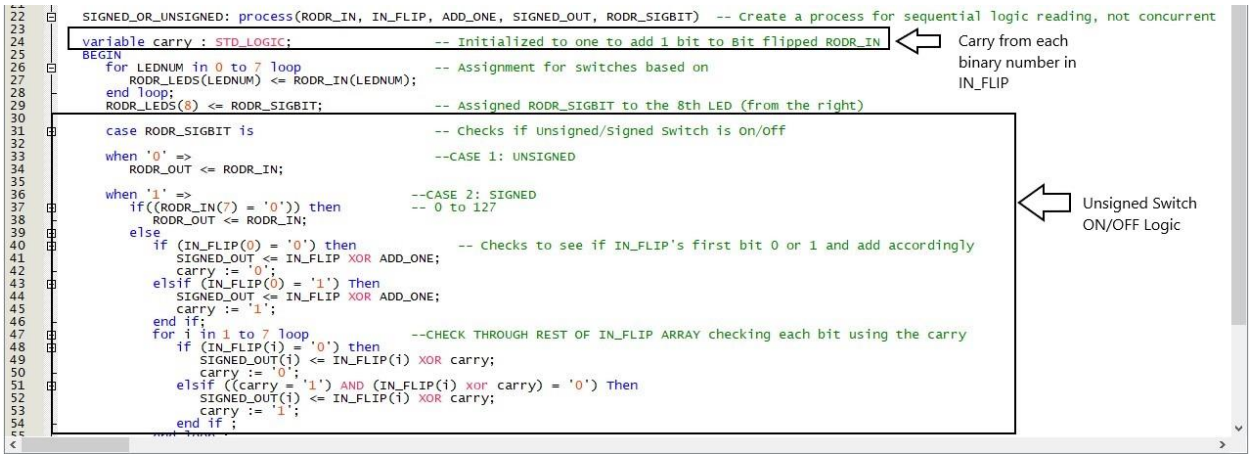


Figure 6: Creating two's complement using an XOR and variable "Carry" system

In order to create two's complement, we have created a variable called **carry**, which will determine whether or not the addition of the first bit with two's complement will be either lead to a logical '0' carry or a logical '1' carry. The use of **RODR_SIGBIT** will also be carried into a process that will provide for the use of switch statements on whether we need to check for two's complement. If logical '0' is passed in from the switch (off on the board), then simply pass **RODR_IN** through to **RODR_OUT** for the Binary to BCD Decoder. If logical '1' is passed in

from the switch (flipped upward on the board), then we must check if the binary input needs to use two's complement to show a negative number. If the most significant bit in **RODR_IN** is '0', then we also just simply pass through **RODR_IN** to **RODR_OUT**. If the most significant bit is '1', then we must do two's complement.

We used an initial condition for the first bit of the binary number from **IN_FLIP** and check whether if it is a logical '0' or '1'. If it is a logical '0', then we use an XOR gate to add both **IN_FLIP** and **ADD_ONE** to **SIGNED_OUT** and start without any carry. If **IN_FLIP** starts with a logical '1', then we will also add **IN_FLIP** and **ADD_ONE** using an XOR gate, but provide for a carry of '1'. The for loop right after will allow for the use of the variable **carry** to go through each bit in **IN_FLIP** to complete two's complement.

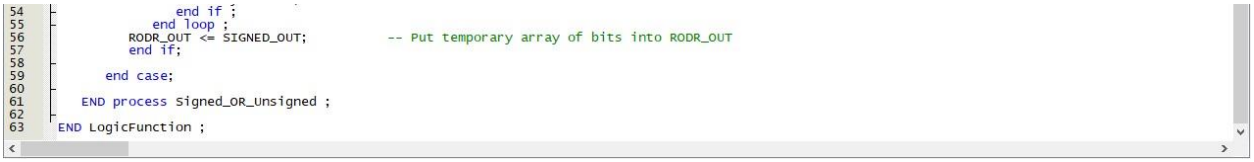


Figure 7: End of code for RODR_UnsignedSigned

At the very end we pass in the two's complement version of the input **RODR_IN** and pass it to the output of the circuit.

Binary to BCD Decoder

RODR_BINARYBCD.vhd

```
1   LIBRARY ieee ;
2   USE ieee.std_logic_1164.all ;
3   USE ieee.std_logic_unsigned.all;
4
5      ENTITY RODR_BINARYBCD IS
6      PORT(RODR_IN: IN STD_LOGIC_VECTOR (7 downto 0) ;          -- Input from Unsigned/Signed Decoder
7           RODR_OUT : OUT STD_LOGIC_VECTOR (11 downto 0));      -- Output as 12 bits (for three 7 Segment Display)
8   END RODR_BINARYBCD ;
9
10  ARCHITECTURE LogicFunction OF RODR_BINARYBCD IS
11  BEGIN
12      shift: process(RODR_IN)
13      variable shiftedbits: STD_LOGIC_VECTOR (17 downto 0);   -- Needed to Make 255
14
15      BEGIN
16        for BITNUM in 0 to 17 loop                             -- Initializes all bits to 0
17           shiftedbits(BITNUM) := '0';
18        end loop;
19        shiftedbits(10 downto 3) := RODR_IN;                   -- Initial shift of 3 bits from RODR_IN
20
21        for i in 0 to 4 loop                                   -- Need to shift 4 times to get numbers for the HUNDREDS place
22
23           if shiftedbits(11 downto 8) > 4 then                --  Checks if binary within the ONES place is > 4
24              shiftedbits(11 downto 8) := shiftedbits(11 downto 8) + 3;
25           end if;
26
27           if shiftedbits(15 downto 12) > 4 then               --  Checks if binary within the TENS place is > 4
28              shiftedbits(15 downto 12) := shiftedbits(15 downto 12) + 3;
29           end if;
30
31        shiftedbits(17 downto 1) := shiftedbits(16 downto 0); -- Shift all bits to the left by 1
32        end loop;
33
34        RODR_OUT(9 downto 0) <= shiftedbits(17 downto 8);
35        RODR_OUT(10) <= '0';                                   -- Set Bit 10 to 0
36        RODR_OUT(11) <= '0';                                   -- Set Bit 11 to 0
37
38     END process shift;
39  END LogicFunction ;
```

Figure 8: VHDL code for the Binary to BCD Decoder

In this section, we will be converting the output from the Unsigned/Signed circuit and importing it into the Binary to BCD Decoder that was created. It will convert 8-bit binary into 12-bit binary that will then be parsed every 4 consecutive bits to represent the ones, tens, and hundreds place and passed to the BCD to Seven Segment Decoder to convert it into decimal.

```
1   LIBRARY ieee ;
2   USE ieee.std_logic_1164.all ;
3   USE ieee.std_logic_unsigned.all;
4
```

Libraries

Figure 9: Libraries used in RODR_BinaryBCD

To start, we used *ieee.std_logic_1164.all* to use logical variables and vectors that will be in use. We also used *ieee.std_logic_unsigned.all* to allow for the addition of numbers through integers than adding binary.

```
ENTITY RODR_BINARYBCD IS
PORT(RODR_IN: IN STD_LOGIC_VECTOR (7 downto 0) ;          -- Input from Unsigned/Signed Decoder
     RODR_OUT : OUT STD_LOGIC_VECTOR (11 downto 0));      -- Output as 12 bits (for three 7 Segment Display)
END RODR_BINARYBCD ;
```

Inputs & Outputs

Figure 10: Inputs and outputs defined for the Binary to BCD Decoder

In this case, **RODR_IN** is defined as the input from the Unsigned/Signed circuit. **RODR_OUT** will be the 12-bit output that will be needed to display a three-digit decimal number to the seven segment displays.

```
ARCHITECTURE LogicFunction OF RODR_BINARYBCD IS
BEGIN
    shift: process(RODR_IN)
    variable shiftedbits: STD_LOGIC_VECTOR (17 downto 0);   -- Needed to Make 255

    BEGIN
      for BITNUM in 0 to 17 loop                             -- Initializes all bits to 0
         shiftedbits(BITNUM) := '0';
      end loop;
```

18 Bit end result to make decimal '255'

Figure 11: Temporary variable declaration that will store the results of the Binary to BCD Decoder

We begin by declaring a process that will allow for the functionality of loop statements. The declaration of a variable called **shiftedbits** will be used to hold the total result of the algorithm we will be using and initially set all of its bits to logical ‘0’ to use it for comparison.

```
    shiftedbits(10 downto 3) := RODR_IN;                  -- Initial shift of 3 bits from RODR_IN

    for i in 0 to 4 loop                                  -- Need to shift 4 times to get numbers for the HUNDREDS place

       if shiftedbits(11 downto 8) > 4 then               --  Checks if binary within the ONES place is > 4
          shiftedbits(11 downto 8) := shiftedbits(11 downto 8) + 3;
       end if;

       if shiftedbits(15 downto 12) > 4 then              --  Checks if binary within the TENS place is > 4
          shiftedbits(15 downto 12) := shiftedbits(15 downto 12) + 3;
       end if;

    shiftedbits(17 downto 1) := shiftedbits(16 downto 0); -- Shift all bits to the left by 1
    end loop;
```

Bit shifting & adding 3 if needed

Figure 12: Shift & Add 3 Algorithm

In this segment we will see the “Shift & Add 3” algorithm being used. This algorithm enables the conversion from 8 bit binary to 12-bit binary. This is done by first shifting **RODR_IN** by three bits to start checking in the ones and tens place if a conversion needs to be made. If any of those 4 consecutive bits that resides in the ones and/or tens place is greater than the decimal number 4, add 3 to it. If not, continue. Then at the end shift all bits to the left by one. We are doing this process 4 times to account for the extra bits that are being added that will not be used in the result.

```
    RODR_OUT(9 downto 0) <= shiftedbits(17 downto 8);
    RODR_OUT(10) <= '0';                                  -- Set Bit 10 to 0
    RODR_OUT(11) <= '0';                                  -- Set Bit 11 to 0

  END process shift;
END LogicFunction ;
```

Placing temporary variable into output of decoder

Figure 13: Storing of converted numbers

After the algorithm has completed, store bits 18 – 9 into **RODR_OUT**. In the end, **RODR_OUT** will be segmented into 3 parts. Bits 0-3 will be sent to one BCD to Seven Segment Decoder signifying the ONES place. Bits 4-7 will be sent to another BCD to Seven Segment Decoder signifying the TENS place. Bits 8-11 will be sent to a third BCD to Seven Segment Decoder signifying the HUNDREDS place. We add logical '0' to bits 10 and 11 to complete the hundreds place decoding as it won't go beyond "0010".

BCD to Seven Segment Display Decoder

RODR_BCDDecoder.vhd

```
1   LIBRARY ieee ;
2   USE ieee.std_logic_1164.all;
3
4      ENTITY RODR_BCDDecoder IS
5      PORT(RODR_IN   : IN STD_LOGIC_VECTOR (3 downto 0) ;   -- Input from Binary to BCD Decoder (For Ones, Tens, and Hundreds)
6           RODR_OUT  : OUT STD_LOGIC_VECTOR (6 downto 0);   -- Output as segments from 7 Segment Display
7   END RODR_BCDDecoder ;
8
9   ARCHITECTURE LogicFunction OF RODR_BCDDecoder IS
10  BEGIN
11
12     LogicO_To_9: process(RODR_IN)    -- Create a process for sequential logic reading, not concurrent
13     BEGIN
14        case RODR_IN is               -- Switch statement checking 4-bit RODR_IN
15
16        when "0000" =>                -- Number 0
17           RODR_OUT <= "0000001";
18        when "0001" =>                -- Number 1
19           RODR_OUT <= "1001111";
20        when "0010" =>                -- Number 2
21           RODR_OUT <= "0010010";
22        when "0011" =>                -- Number 3
23           RODR_OUT <= "0000110";
24        when "0100" =>                -- Number 4
25           RODR_OUT <= "1001100";
26        when "0101" =>                -- Number 5
27           RODR_OUT <= "0100100";
28        when "0110" =>                -- Number 6
29           RODR_OUT <= "1100000";
30        when "0111" =>                -- Number 7
31           RODR_OUT <= "0001111";
32        when "1000" =>                -- Number 8
33           RODR_OUT <= "0000000";
34        when "1001" =>                -- Number 9
35           RODR_OUT <= "0001100";
36
37        when others =>
38           RODR_OUT <= "1111111";     -- Segment Display is off (Error Checking)
39
40        END case;
41
42     END process LOGICO_TO_9 ;
43
44  END LogicFunction ;
```

Figure 14: VHDL code for BCD to Seven Segment Decoder

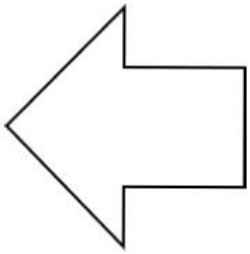
The BCD to Seven Segment Decoder will be used to take the ONES, TENS, and HUNDREDS place sections from the 12-bit Binary to BCD Decoder under separate decoders to then be displayed on the boards.

```
1   LIBRARY ieee ;
2   USE ieee.std_logic_1164.all;
3
4      ENTITY RODR_BCDDecoder IS
5      PORT(RODR_IN   : IN STD_LOGIC_VECTOR (3 downto 0) ;   -- Input from Binary to BCD Decoder (For Ones, Tens, and Hundreds)
6           RODR_OUT  : OUT STD_LOGIC_VECTOR (6 downto 0);   -- Output as segments from 7 Segment Display
7   END RODR_BCDDecoder ;
8
```

Figure 15: Libraries and Inputs/Outputs defined in RODR_BCDDecoder

In this decoder, we are only using the *ieee.std_logic_1164.all* library to define our logical variables and vectors. **RODR_IN** will be used to take in the 4 bits provided by a selection from the Binary to BCD Decoder. **RODR_OUT** will be the code that enable and disables parts of the seven segment displays to display numbers 0 to 9.

```
ARCHITECTURE LogicFunction OF RODR_BCDDecoder IS
BEGIN

   Logic0_To_9: process(RODR_IN)     -- Create a process for sequential logic reading, not concurrent
   BEGIN
      case RODR_IN is                -- Switch statement checking 4-bit RODR_IN

      when "0000" =>                 -- Number 0
         RODR_OUT <= "0000001";
      when "0001" =>                 -- Number 1
         RODR_OUT <= "1001111";
      when "0010" =>                 -- Number 2
         RODR_OUT <= "0010010";
      when "0011" =>                 -- Number 3
         RODR_OUT <= "0000110";
      when "0100" =>                 -- Number 4
         RODR_OUT <= "1001100";
      when "0101" =>                 -- Number 5
         RODR_OUT <= "0100100";
      when "0110" =>                 -- Number 6
         RODR_OUT <= "1100000";
      when "0111" =>                 -- Number 7
         RODR_OUT <= "0001111";
      when "1000" =>                 -- Number 8
         RODR_OUT <= "0000000";
      when "1001" =>                 -- Number 9
         RODR_OUT <= "0001100";

      when others =>
         RODR_OUT <= "1111111";      -- Segment Display is off (Error Checking)

      END case;

   END process LOGIC0_TO_9 ;

END LogicFunction ;
```



Lookup table for 4-bit binary input

Figure 16: Switch case that will be used to decode number 0 to 9

In this architecture, we will be using a switch case to determine the number sequence being taken in to each decoder. The input that will be accepted will be the binary numbers “0000” to “1001” signifying their decimal counterparts 0 to 9. Based on both the DE2 & DE1-SOC manuals, **RODR_OUT** will be determined based on the bit sequences from those manuals to display numbers 0 through 9. If there are other inputs that are taken in through **RODR_IN**, it will then display nothing. This is used to do some error checking to signify that there is something wrong with the mapping of all these decoders and circuits together.

Plus/Minus Decoder

RODR_PlusMinus.vhd

```
1   LIBRARY ieee ;
2   USE ieee.std_logic_1164.all ;
3
4   ENTITY RODR_PlusMinus IS
5      PORT(RODR_SIGBIT : IN STD_LOGIC;                    -- Input from Signed/Unsigned Switch
6           RODR_IN: IN STD_LOGIC_VECTOR (7 downto 0);     -- Input from Bit Switches
7           RODR_OUT : OUT STD_LOGIC_VECTOR (6 downto 0));   -- Output for 7 Segment Display
8   END RODR_PlusMinus ;
9
10  ARCHITECTURE LogicFunction OF RODR_PlusMinus IS
11  BEGIN
12     plus_or_minus : process(RODR_SIGBIT, RODR_IN)
13     BEGIN
14
15        if   ((RODR_SIGBIT = '0') AND (RODR_IN(7) = '0')) then      -- Display Nothing (Unsigned)
16           RODR_OUT <= "1111111";
17        elsif((RODR_SIGBIT = '0') AND (RODR_IN(7) = '1')) then      -- Display Nothing (Unsigned)
18           RODR_OUT <= "1111111";
19        elsif((RODR_SIGBIT = '1') AND (RODR_IN(7) = '1')) then      -- Display Minus (Signed)
20           RODR_OUT <= "1111110";
21        elsif((RODR_SIGBIT = '1') AND (RODR_IN(7) = '0')) then      -- Display Plus (Signed)
22           RODR_OUT <= "1001110";
23        end if;
24     end process plus_or_minus;
25
26  END LogicFunction ;
```

Figure 17: VHDL Code showcasing the Plus/Minus decoder

In this decoder, we are using the 4th seven segment display to display either plus, minus or nothing. This is decided based on whether the unsigned/signed switch is turned off or on, respectively.

```
1   LIBRARY ieee ;
2   USE ieee.std_logic_1164.all ;
3
4   ENTITY RODR_PlusMinus IS
5      PORT(RODR_SIGBIT : IN STD_LOGIC;                    -- Input from Signed/Unsigned Switch
6           RODR_IN: IN STD_LOGIC_VECTOR (7 downto 0);     -- Input from Bit Switches
7           RODR_OUT : OUT STD_LOGIC_VECTOR (6 downto 0));   -- Output for 7 Segment Display
8   END RODR_PlusMinus ;
```

Figure 18: Libraries and Inputs/Outputs defined in RODR_PlusMinus

We are using the *ieee.std_logic_1164.all* library to help define our logical variables and vectors. In this decoder, we are taking in **RODR_SIGBIT**, which allows us to use the unsigned/signed switch to know if we need a positive or negative sign displayed. **RODR_IN** will take the entire input vector from the switches in order to show a plus or minus. **RODR_OUT** will be the decoded version for the display to show either a “+”, “-“, or turned off.

```
10  ARCHITECTURE LogicFunction OF RODR_PlusMinus IS
11  BEGIN
12      plus_or_minus : process(RODR_SIGBIT, RODR_IN)
13      BEGIN
14
15          if   ((RODR_SIGBIT = '0') AND (RODR_IN(7) = '0')) then      -- Display Nothing (Unsigned)
16              RODR_OUT <= "1111111";
17          elsif((RODR_SIGBIT = '0') AND (RODR_IN(7) = '1')) then      -- Display Nothing (Unsigned)
18              RODR_OUT <= "1111111";
19          elsif((RODR_SIGBIT = '1') AND (RODR_IN(7) = '1')) then      -- Display Minus (Signed)
20              RODR_OUT <= "1111110";
21          elsif((RODR_SIGBIT = '1') AND (RODR_IN(7) = '0')) then      -- Display Plus (Signed)
22              RODR_OUT <= "1001110";
23          end if;
24      end process plus_or_minus;
25
26   END LogicFunction ;
```

Figure 19: Logic behind Plus/Minus Decoder

In order to use this decoder properly, we have to check **RODR_SIGBIT** and the most significant bit in **RODR_IN** to know when to display either a positive, negative, or nothing is displayed at all. These if statements will use an AND gate to determine whether we need any of those as an output.

To ensure that nothing is displayed, **RODR_SIGBIT** bit MUST be a logical ‘0’. This means that the bits being displayed to the output is unsigned and thus does not need an indication of a plus or minus as it is already implied that the decimal number displayed is positive. However, if **RODR_SIGBIT** is a logical ‘1’ (switch flipped), then we need to display positive or negative. To display a negative, the most significant bit from **RODR_IN** must be a logical ‘1’. This means that two’s complement was done in the Unsigned/Signed Circuit to account for it being a negative number between [-1 to -128]. Otherwise, display a plus as the binary number is between decimal converted 0 to 127.

No Output Decoder (Use in DE2 Board Only)

RODR_NOOUTPUT.vhd

```
1   LIBRARY ieee ;
2   USE ieee.std_logic_1164.all ;
3   USE ieee.std_logic_unsigned.all;
4
5      ENTITY RODR_NOOUTPUT IS
6      PORT(RODR_OUT : OUT STD_LOGIC_VECTOR (6 downto 0));
7   END RODR_NOOUTPUT ;
8
9   ARCHITECTURE LogicFunction OF RODR_NOOUTPUT IS
10  BEGIN
11        RODR_OUT <= "1111111";
12  END LogicFunction ;
```

Figure 20: VHDL code for RODR_NOOUTPUT

This “No Output” decoder is only used on the DE2 board. This is because the Seven Segment displays that are not in use automatically turn all its lights on. So this decoder is used to turn them all off instead. The DE1-SOC defaults all unused displays off. We don’t need any input as it will be turned off on its own. In order to do this, **RODR_OUT** will be set to hold 7 bits and be assigned all logical ‘1’s. This ensure that all segments on a display are off when not in use.

Master File

RODR_MASTERMAP.vhd

```
1   LIBRARY ieee ;
2   USE ieee.std_logic_1164.all ;
3
4   Entity RODR_MASTERMAP IS
5       PORT(RODR_SIGBIT      : IN STD_LOGIC;                     -- Signed or Unsigned Switch
6            RODR_IN          : IN STD_LOGIC_VECTOR (7 downto 0) ;  -- Input from Switches
7            RODR_ONES        : OUT STD_LOGIC_VECTOR (6 downto 0);  -- Output to Rightmost 7 Seg Display
8            RODR_TENS        : OUT STD_LOGIC_VECTOR (6 downto 0);  -- Output to 2nd 7 Seg Display
9            RODR_HUNDREDS    : OUT STD_LOGIC_VECTOR (6 downto 0);  -- Output to 3rd 7 Seg Display
10           RODR_SIGN        : OUT STD_LOGIC_VECTOR (6 downto 0);  -- Output to 4th 7 Seg Display
11           RODR_LEDONOFF    : OUT STD_LOGIC_VECTOR (8 downto 0)); -- LED ON/OFF determined by Switches
12
13  END RODR_MASTERMAP;
14
15  ARCHITECTURE CONNECTIONS of RODR_MASTERMAP IS
16      Component RODR_UnsignedSigned IS                          --Signed Decoder & LED Control
17          PORT(RODR_SIGBIT : IN STD_LOGIC;
18               RODR_IN   : IN STD_LOGIC_VECTOR (7 downto 0) ;
19               RODR_OUT  : OUT STD_LOGIC_VECTOR (7 downto 0);
20               RODR_LEDS : OUT STD_LOGIC_VECTOR (8 downto 0));
21      END Component;
22
23      Component RODR_BINARYBCD                          -- Binary (from signed Decoder) to BCD Decoder
24          PORT(RODR_IN: IN STD_LOGIC_VECTOR (7 downto 0) ;
25               RODR_OUT : OUT STD_LOGIC_VECTOR (11 downto 0));
26      END Component ;
27
28      Component RODR_BCDDecoder                           -- BCD (from BCD Decoder) to 7 Seg Display
29          PORT(RODR_IN: IN STD_LOGIC_VECTOR (3 downto 0) ;
30               RODR_OUT : OUT STD_LOGIC_VECTOR (6 downto 0));
31      END Component ;
32
33      Component RODR_PlusMinus                            -- Displays + or - in Signed Mode
34      PORT(RODR_SIGBIT : IN STD_LOGIC;
35           RODR_IN: IN STD_LOGIC_VECTOR (7 downto 0);
36           RODR_OUT : OUT STD_LOGIC_VECTOR (6 downto 0));
37      END Component;
38
39  signal RODR_TCO : STD_LOGIC_VECTOR (7 downto 0);      --TCO = TWo's Complement Output
40  signal RODR_BCD : STD_LOGIC_VECTOR (11 downto 0);     -- Takes in Output from BCD
41  signal RODR_BCD_ONE : STD_LOGIC_VECTOR (3 downto 0);
42  signal RODR_BCD_TEN : STD_LOGIC_VECTOR (3 downto 0);
43  signal RODR_BCD_HUNDRED : STD_LOGIC_VECTOR (3 downto 0);
44
45  BEGIN
46  -- CONNECTIONS BETWEEN THE VHDL FILES AND DECODERS
47
48      RODR_TWOSCOMPLEMENT   : RODR_UnsignedSigned  PORT MAP(RODR_SIGBIT, RODR_IN, RODR_TCO, RODR_LEDONOFF);
49      RODR_BINARYTOBCD      : RODR_BINARYBCD       PORT MAP(RODR_TCO, RODR_BCD);
50
51      RODR_BCD_ONE <= STD_LOGIC_VECTOR (RODR_BCD(3 downto 0));       -- Parsing of RODR_BCD Output to three 7 Seg Displays
52      RODR_BCD_TEN <= STD_LOGIC_VECTOR (RODR_BCD(7 downto 4));
53      RODR_BCD_HUNDRED <= STD_LOGIC_VECTOR (RODR_BCD(11 downto 8));
54
55
56      RODR_DecoderONES      : RODR_BCDDecoder      PORT MAP(RODR_BCD_ONE, RODR_ONES);
57      RODR_DecoderTENS      : RODR_BCDDecoder      PORT MAP(RODR_BCD_TEN, RODR_TENS);
58      RODR_DecoderHUNDREDS  : RODR_BCDDecoder      PORT MAP(RODR_BCD_HUNDRED, RODR_HUNDREDS);
59      RODR_POSNEG           : RODR_PlusMinus       PORT MAP (RODR_SIGBIT, RODR_IN, RODR_SIGN);
60
61  END CONNECTIONS;
```

Figure 21: VHDL code for connections between each electronics device created for DE1-SOC board

This VHDL file **RODR_MASTERMAP** describes all inputs and outputs that are going in and out of each created device. All devices are set as components and then set to port maps that will connect the I/O's to their correct locations. We are also creating temporary signals to take in outputs from other electronics and parse them later to convert them to be placed into the next electronic.

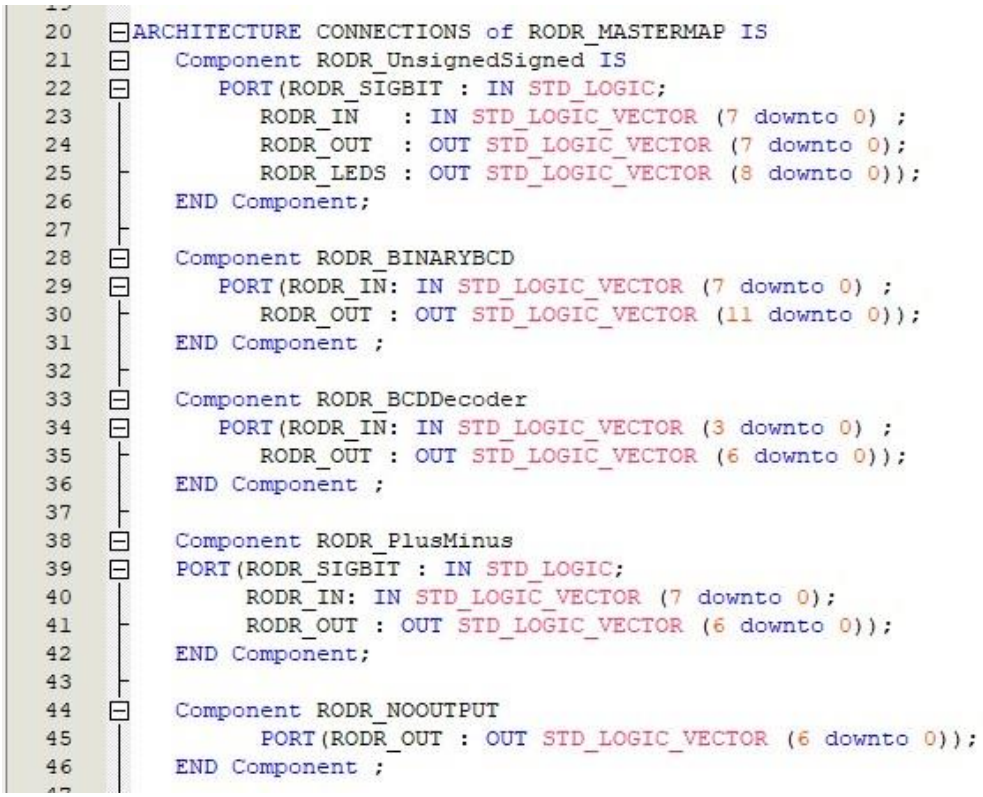
From *Figure 21*, we are setting up inputs and outputs that will have pin assignments imported through a .txt file for both the DE2 and DE1-SOC boards. Inputs **RODR_SIGBIT** an **RODR_IN** will be inputs directly from the board. The first one defines if the binary is either positive, negative, or won't be displayed at all. The other is user defined by the switches on the board in binary to be then converted into a decimal representation on the seven segment displays. **RODR_ONE**S, **RODR_TENS**, and **RODR_HUNDREDS** are outputs taken from the BCD to Seven Segment Decoder that are to be displayed in their respective displays on the board. **RODR_SIGN** is a version of the seven segment decoder that determines if there will be a sign

displayed for the corresponding decimal number. **RODR_LEDONOFF** will determine whether the red LED lights on top of the switches will be on or off. They are based on if the binary switches and the signed/unsigned switch is toggled or not.

```
1   LIBRARY ieee ;
2   USE ieee.std_logic_1164.all ;
3
4   Entity RODR_MASTERMAP IS
5      PORT(RODR_SIGBIT      : IN STD_LOGIC;
6           RODR_IN          : IN STD_LOGIC_VECTOR (7 downto 0) ;
7           RODR_ONES        : OUT STD_LOGIC_VECTOR (6 downto 0);
8           RODR_TENS        : OUT STD_LOGIC_VECTOR (6 downto 0);
9           RODR_HUNDREDS    : OUT STD_LOGIC_VECTOR (6 downto 0);
10          RODR_SIGN        : OUT STD_LOGIC_VECTOR (6 downto 0);
11          RODR_LEDONOFF    : OUT STD_LOGIC_VECTOR (8 downto 0);
12          RODR_DISP4       : OUT STD_LOGIC_VECTOR (6 downto 0);
13          RODR_DISP5       : OUT STD_LOGIC_VECTOR (6 downto 0);
14          RODR_DISP6       : OUT STD_LOGIC_VECTOR (6 downto 0);
15          RODR_DISP7       : OUT STD_LOGIC_VECTOR (6 downto 0));
16
17
18  END RODR_MASTERMAP;
19
20  ARCHITECTURE CONNECTIONS of RODR_MASTERMAP IS
21     Component RODR_UnsignedSigned IS
22        PORT(RODR_SIGBIT : IN STD_LOGIC;
23             RODR_IN     : IN STD_LOGIC_VECTOR (7 downto 0) ;
24             RODR_OUT    : OUT STD_LOGIC_VECTOR (7 downto 0);
25             RODR_LEDS : OUT STD_LOGIC_VECTOR (8 downto 0));
26     END Component;
```

Figure 22: Inputs and outputs from Quartus 13.0 for DE2 board

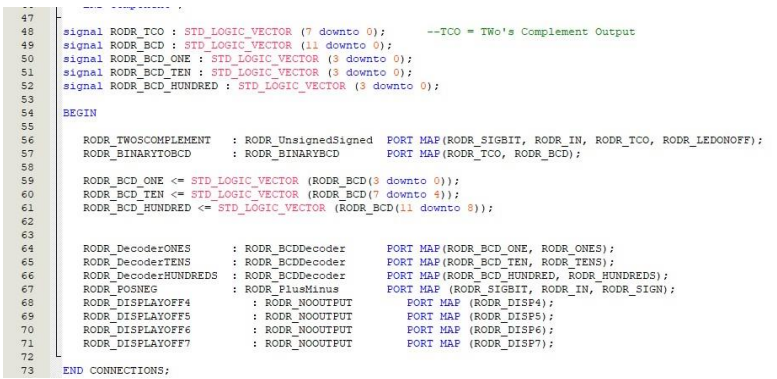
The difference between the DE1-SOC and DE2 is that the DE2 does not turn off the segment displays that aren't in use. So **RODR_DISP4** through **RODR_DISP7** will allow for the DE2 board segment displays to be turned off.



```
ARCHITECTURE CONNECTIONS of RODR_MASTERMAP IS
    Component RODR_UnsignedSigned IS
        PORT(RODR_SIGBIT : IN STD_LOGIC;
             RODR_IN    : IN STD_LOGIC_VECTOR (7 downto 0) ;
             RODR_OUT   : OUT STD_LOGIC_VECTOR (7 downto 0);
             RODR_LEDS : OUT STD_LOGIC_VECTOR (8 downto 0));
    END Component;

    Component RODR_BINARYBCD
        PORT(RODR_IN: IN STD_LOGIC_VECTOR (7 downto 0) ;
             RODR_OUT : OUT STD_LOGIC_VECTOR (11 downto 0));
    END Component ;

    Component RODR_BCDDecoder
        PORT(RODR_IN: IN STD_LOGIC_VECTOR (3 downto 0) ;
             RODR_OUT : OUT STD_LOGIC_VECTOR (6 downto 0));
    END Component ;

    Component RODR_PlusMinus
    PORT(RODR_SIGBIT : IN STD_LOGIC;
          RODR_IN: IN STD_LOGIC_VECTOR (7 downto 0);
          RODR_OUT : OUT STD_LOGIC_VECTOR (6 downto 0));
    END Component;

    Component RODR_NOOUTPUT
          PORT(RODR_OUT : OUT STD_LOGIC_VECTOR (6 downto 0));
    END Component ;
```

Figure 23: Components created to output unsigned/signed numbers to the seven segment display

Defined here will show the components that were already created using their respective VHDL files. Each of their functionalities are explained in their respective subsections. What the component keyword does is define a file that has already been placed inside the project folder and use it to be linked in one main file. All the ports defined in each component are the same as the components that were assigned in their respective files.



```
signal RODR_TCO : STD_LOGIC_VECTOR (7 downto 0);      --TCO = TWo's Complement Output
signal RODR_BCD : STD_LOGIC_VECTOR (11 downto 0);
signal RODR_BCD_ONE : STD_LOGIC_VECTOR (3 downto 0);
signal RODR_BCD_TEN : STD_LOGIC_VECTOR (3 downto 0);
signal RODR_BCD_HUNDRED : STD_LOGIC_VECTOR (3 downto 0);

BEGIN

    RODR_TWOSCOMPLEMENT   : RODR_UnsignedSigned  PORT MAP(RODR_SIGBIT, RODR_IN, RODR_TCO, RODR_LEDONOFF);
    RODR_BINARYTOBCD      : RODR_BINARYBCD       PORT MAP(RODR_TCO, RODR_BCD);

    RODR_BCD_ONE <= STD_LOGIC_VECTOR (RODR_BCD(3 downto 0));
    RODR_BCD_TEN <= STD_LOGIC_VECTOR (RODR_BCD(7 downto 4));
    RODR_BCD_HUNDRED <= STD_LOGIC_VECTOR (RODR_BCD(11 downto 8));


    RODR_DecoderONES      : RODR_BCDDecoder      PORT MAP(RODR_BCD_ONE, RODR_ONES);
    RODR_DecoderTENS      : RODR_BCDDecoder      PORT MAP(RODR_BCD_TEN, RODR_TENS);
    RODR_DecoderHUNDREDS  : RODR_BCDDecoder      PORT MAP(RODR_BCD_HUNDRED, RODR_HUNDREDS);
    RODR_POSNEG           : RODR_PlusMinus       PORT MAP (RODR_SIGBIT, RODR_IN, RODR_SIGN);
    RODR_DISPLAYOFF4         : RODR_NOOUTPUT        PORT MAP (RODR_DISP4);
    RODR_DISPLAYOFF5         : RODR_NOOUTPUT        PORT MAP (RODR_DISP5);
    RODR_DISPLAYOFF6         : RODR_NOOUTPUT        PORT MAP (RODR_DISP6);
    RODR_DISPLAYOFF7         : RODR_NOOUTPUT        PORT MAP (RODR_DISP7);

END CONNECTIONS;
```

Figure 24: Connections between the inputs/outputs using the PORT MAP function

In order to ensure that the connections work properly, we must use the PORT MAP function to ensure that every input and output connect to get the right output on the seven-segment display.

RODR_TWOSCOMPLEMENT is defined as an instance of the **RODR_UnsignedSigned** VHDL file that converts binary to either signed or unsigned. It is mapped to *RODR_SIGBIT, RODR_IN (from the board), RODR_TCO (temporary variable storing the output), and RODR_LEDONOFF (turns on or off led based on switch inputs).*

RODR_BINARYTOBCD is an instance of **RODR_BINARYBCD** that converts the binary from the unsigned/signed circuit into 12 bits. It takes in *RODR_TCO* and *RODR_BCD (temporary variable that stores 12 bit binary output).*

We then parse the output 12 bit binary file from *RODR_BCD* into 4 consecutive bits (from right to left) that signify the ones, tens, and hundreds place under temporary variables *RODR_BCD_ONE, RODR_BCD_TEN,* and *RODR_BCD_Hundred*, respectively.

There needs to be three decoders to display them onto the seven segment displays, so we pass those inputs to their respecitive decoders, namely **RODR_DecoderONES, RODR_DecoderTENS,** and **RODR_DecoderHundreds** and pass in those as the input to their respective decoders. Their output will be the outputs *RODR_ONES, RODR_TENS***,** and *RODR_Hundreds* that were defined at the top of the file.

RODR_POSNEG is defined as an instance of **RODR_PlusMinus** that changed the 4th seven segment display to either a plus/minus sign or nothing at all. Thus, we pass in *RODR_SIGBIT, RODR_IN, and RODR_SIGN (output defined at top of file for pin assignment)*.

The rest are the other seven segment displays that are programmed to be shut off as for the DE2 board, the seven segment displays are automatically set to being all on. So displays 4 to 7 (from right to left on the board), are shut off.

```
15  ARCHITECTURE CONNECTIONS of RODR_MASTERMAP IS
16     Component RODR_UnsignedSigned IS                          --Signed Decoder & LED Control
17        PORT(RODR_SIGBIT : IN STD_LOGIC;
18             RODR_IN   : IN STD_LOGIC_VECTOR (7 downto 0) ;
19             RODR_OUT  : OUT STD_LOGIC_VECTOR (7 downto 0);
20             RODR_LEDS : OUT STD_LOGIC_VECTOR (8 downto 0));
21     END Component;
22
23     Component RODR_BINARYBCD                          -- Binary (from signed Decoder) to BCD Decoder
24        PORT(RODR_IN: IN STD_LOGIC_VECTOR (7 downto 0) ;
25             RODR_OUT : OUT STD_LOGIC_VECTOR (11 downto 0));
26     END Component ;
27
28     Component RODR_BCDDecoder                         -- BCD (from BCD Decoder) to 7 Seg Display
29        PORT(RODR_IN: IN STD_LOGIC_VECTOR (3 downto 0) ;
30             RODR_OUT : OUT STD_LOGIC_VECTOR (6 downto 0));
31     END Component ;
32
33     Component RODR_PlusMinus                          -- Displays + or - in Signed Mode
34     PORT(RODR_SIGBIT : IN STD_LOGIC;
35          RODR_IN: IN STD_LOGIC_VECTOR (7 downto 0);
36          RODR_OUT : OUT STD_LOGIC_VECTOR (6 downto 0));
37     END Component;
38
```

Figure 25: Components used to link components together from the different VHDL files

```
signal RODR_TCO : STD_LOGIC_VECTOR (7 downto 0);       --TCO = Two's Complement Output
signal RODR_BCD : STD_LOGIC_VECTOR (11 downto 0);      -- Takes in Output from BCD
signal RODR_BCD_ONE : STD_LOGIC_VECTOR (3 downto 0);
signal RODR_BCD_TEN : STD_LOGIC_VECTOR (3 downto 0);
signal RODR_BCD_HUNDRED : STD_LOGIC_VECTOR (3 downto 0);

BEGIN
-- CONNECTIONS BETWEEN THE VHDL FILES AND DECODERS

   RODR_TWOSCOMPLEMENT   : RODR_UnsignedSigned  PORT MAP(RODR_SIGBIT, RODR_IN, RODR_TCO, RODR_LEDONOFF);
   RODR_BINARYTOBCD      : RODR_BINARYBCD       PORT MAP(RODR_TCO, RODR_BCD);

   RODR_BCD_ONE <= STD_LOGIC_VECTOR (RODR_BCD(3 downto 0));        -- Parsing of RODR_BCD Output to three 7 Seg Displays
   RODR_BCD_TEN <= STD_LOGIC_VECTOR (RODR_BCD(7 downto 4));
   RODR_BCD_HUNDRED <= STD_LOGIC_VECTOR (RODR_BCD(11 downto 8));


   RODR_DecoderONES      : RODR_BCDDecoder      PORT MAP(RODR_BCD_ONE, RODR_ONES);
   RODR_DecoderTENS      : RODR_BCDDecoder      PORT MAP(RODR_BCD_TEN, RODR_TENS);
   RODR_DecoderHUNDREDS  : RODR_BCDDecoder      PORT MAP(RODR_BCD_HUNDRED, RODR_HUNDREDS);
   RODR_POSNEG           : RODR_PlusMinus       PORT MAP (RODR_SIGBIT, RODR_IN, RODR_SIGN);

END CONNECTIONS;
```

Figure 26

Both *Figure 25* & *26* are the same code ported from Quartus 13.0 into Quartus 16.1 for use on the DE1-SOC. However, the main difference is the absence of the ***RODR_NOOUTPUT*** component. This removes the need to turn them off as the board automatically defaults the outputs from the segment displays to all be turned off.

Waveforms

Master File

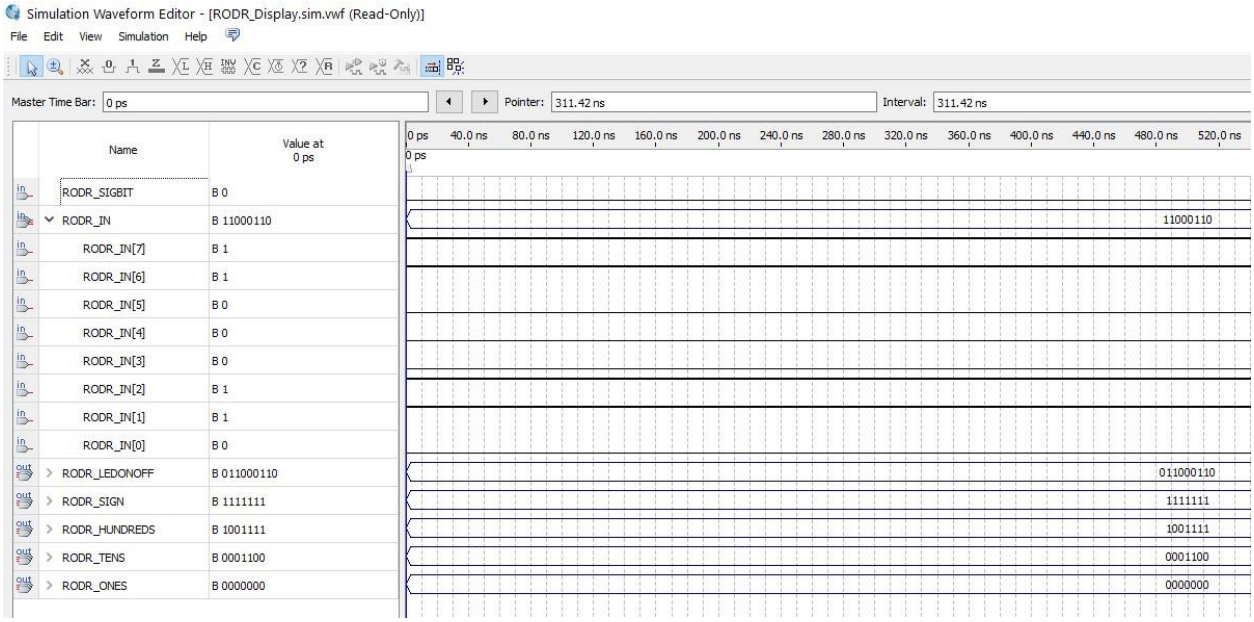


Figure 27: Output waveforms for all inputs and outputs that have been given pin assignments to both DE2 and DE1-SOC

This waveform verifies the input and output assignments that are used on the board. Both **RODR_SIGBIT** and **RODR_IN** are used as inputs. As shown here, **RODR_LEDONOFF** is working properly as it will display the exact same bit sequence passed in from the two inputs. **RODR_SIGN** is also showing all logical '1's, since **RODR_SIGBIT** is in unsigned mode, so no sign needs to be shown as the decimal number is implied to be positive. **RODR_HUNDREDS, RODR_TENS, and RODR_ONES** are showing the code for their respective displays that will display the number 198. The conversion codes are show in the manuals for both the DE2 & DE1-SOC boards.

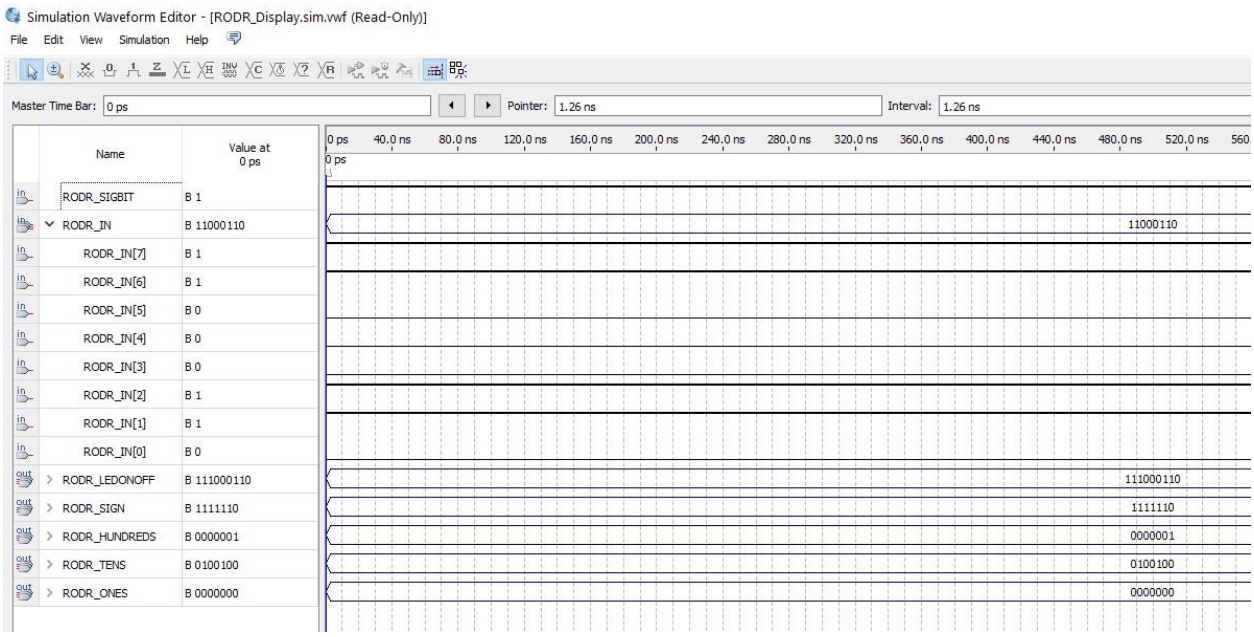


Figure 28: Output waveforms for all inputs and outputs that have been given pin assignments to both DE2 and DE1-SOC but with RODR_SIGBIT converting the binary into signed mode.

This waveform does the same as that of *Figure 28*, but utilizes **RODR_SIGBIT** to convert binary into signed mode. **RODR_LED** will still display the exact same as the inputs and outputs and displays them to the LED's in front of the binary switches plus the unsigned/signed switch. The main differences are the sign and binary for the ones, tens, and hundreds place. **RODR_SIGN** is now displaying a negative on the 4th seven segment display, meaning that the current number is negative, and two's complement has been done. The other three are outputs that are of the result of two's complement, which is -58.

Unsigned /Signed Circuit + LED Functionality

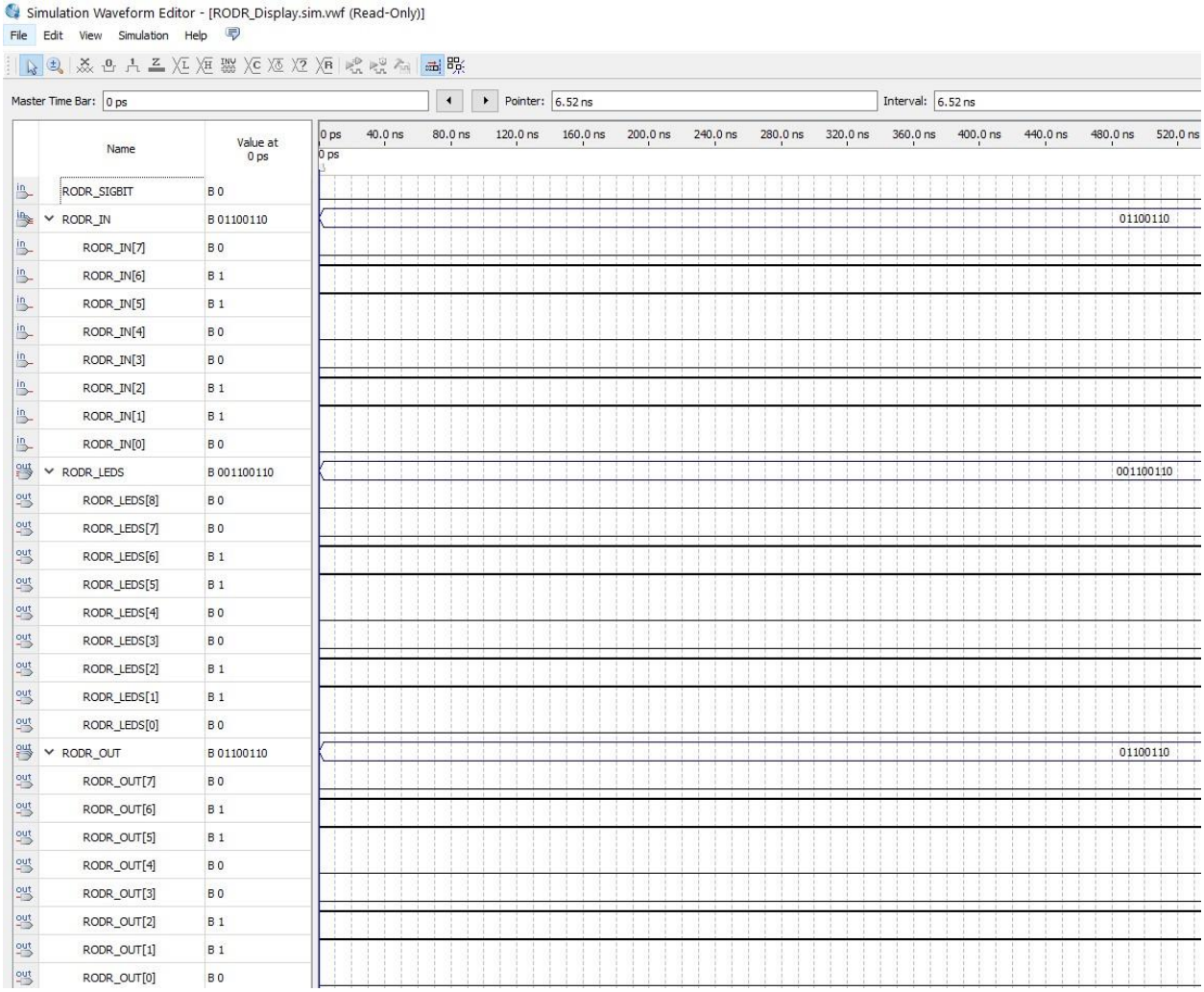


Figure 28: Unsigned + LED Functionality

This waveform shows what occurs within the Unsigned/Signed circuit. It takes in inputs from the board using **RODR_SIGBIT and RODR_IN** and then uses it for LED functionality and it's output. **RODR_OUT**. The LED's will never be changed from the inputs as the sequence "001100110" is displayed. The output also doesn't change as the most significant bit is not on along with the signed/unsigned bit being at a logical '0'.

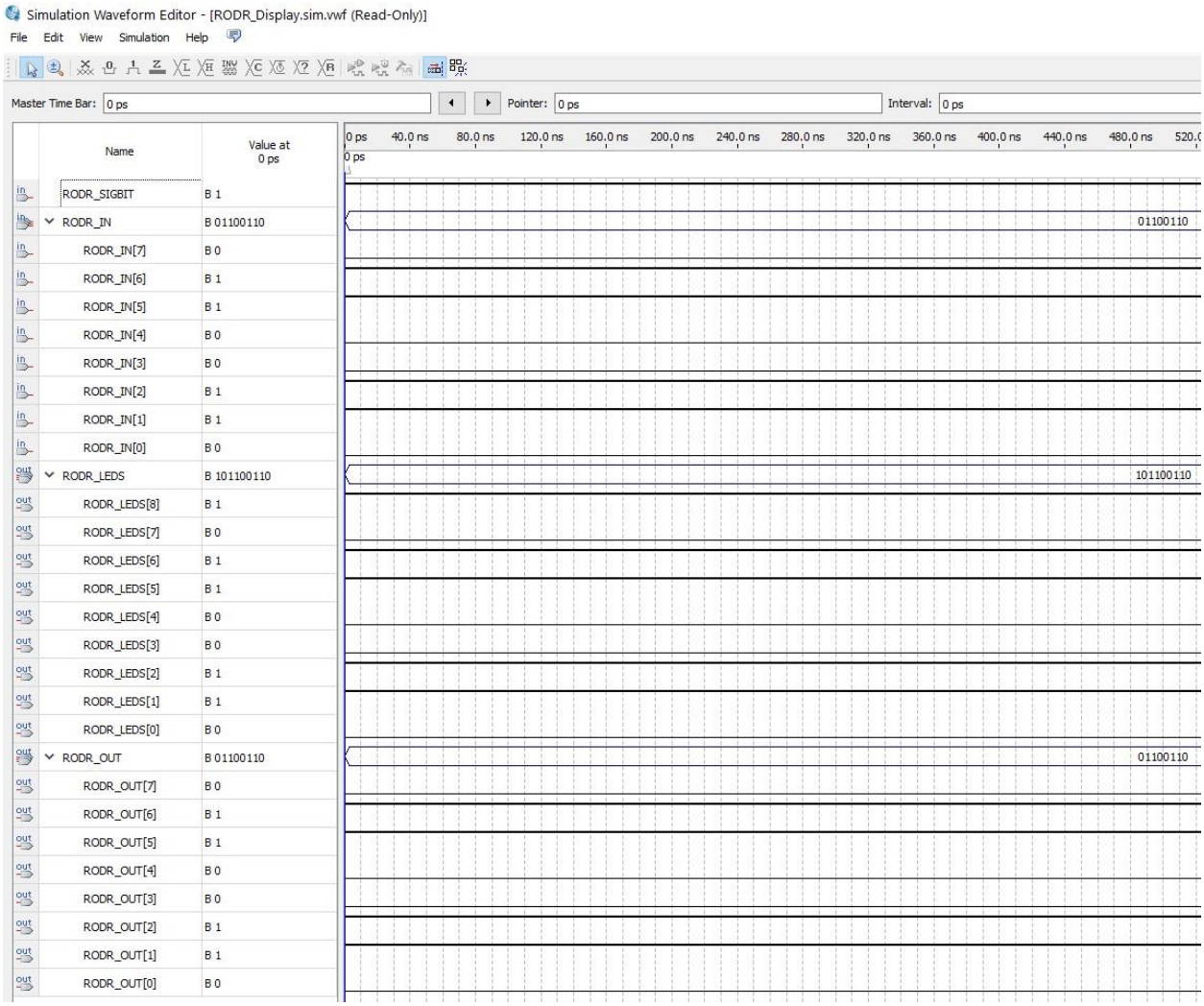


Figure 29: Signed + LED Functionality

In this case, the waveform now includes the unsigned/signed bit being turned on. The LED's will remain the same as they simply take in the inputs and turns their respective LEDs on or off. Despite the signed bit being turned on, the output remains the same. This is because the number converted into decimal is 102. It has yet to reach the maximum allowed for a positive binary number, which is "01111111" or 127 in decimal. So the output will remain the same without two's complement being done.

Figure 30

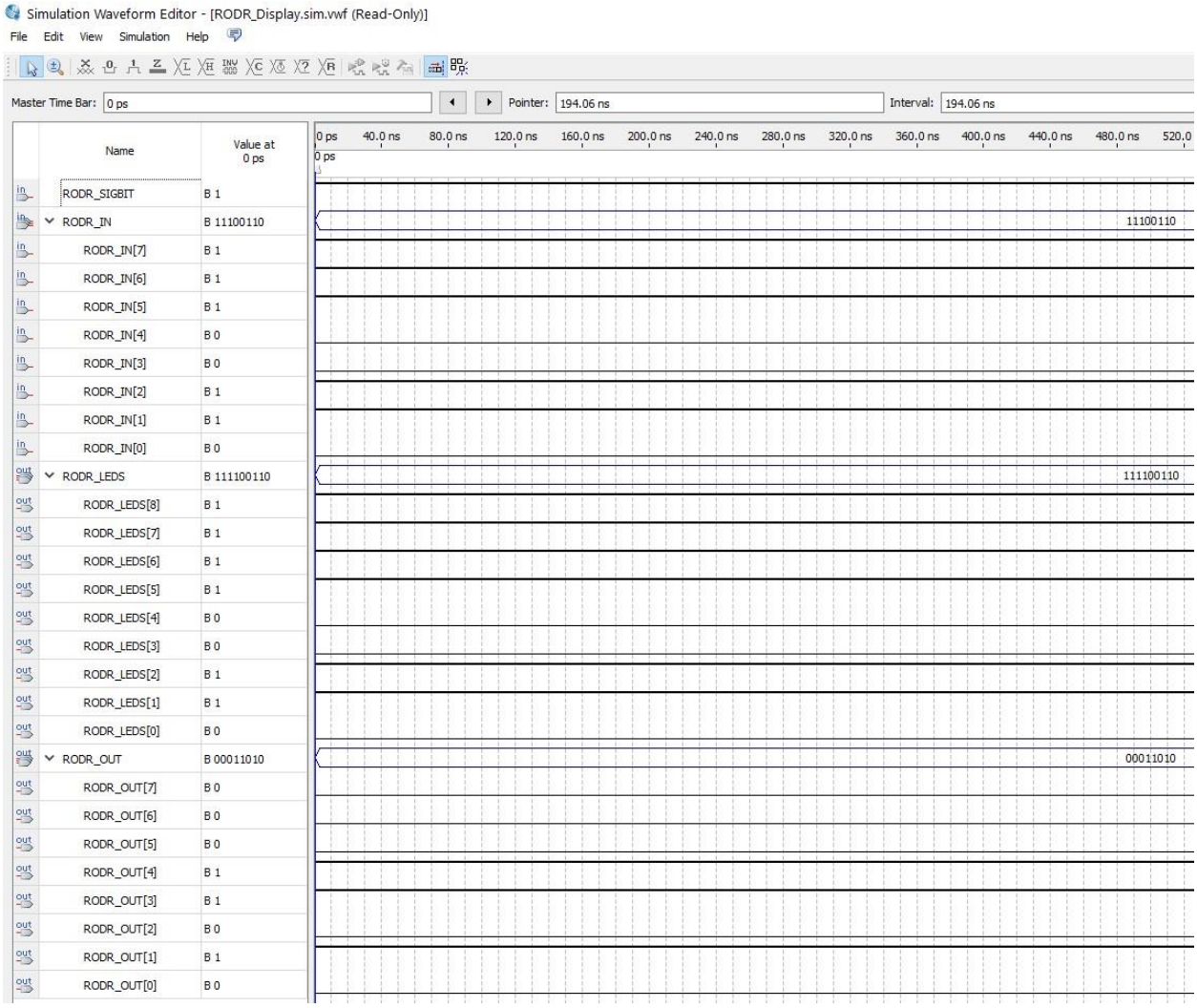


Figure 30: Signed + LED Functionality with Two's Complement

In this case, we have the signed bit turned on and the binary number coming in has the most significant bit being a logical '1', so we must do two's complement. The LEDs will still simply take the inputs and display them accordingly. However, the output will have completed two's complement using XOR gates for they act similar to addition when used, along with a carry variable that defines what happens when a logical '1' plus another logical '1' is added together. If so, the carry is a logical '1' and the carry must be added to the next bit.

Binary to BCD

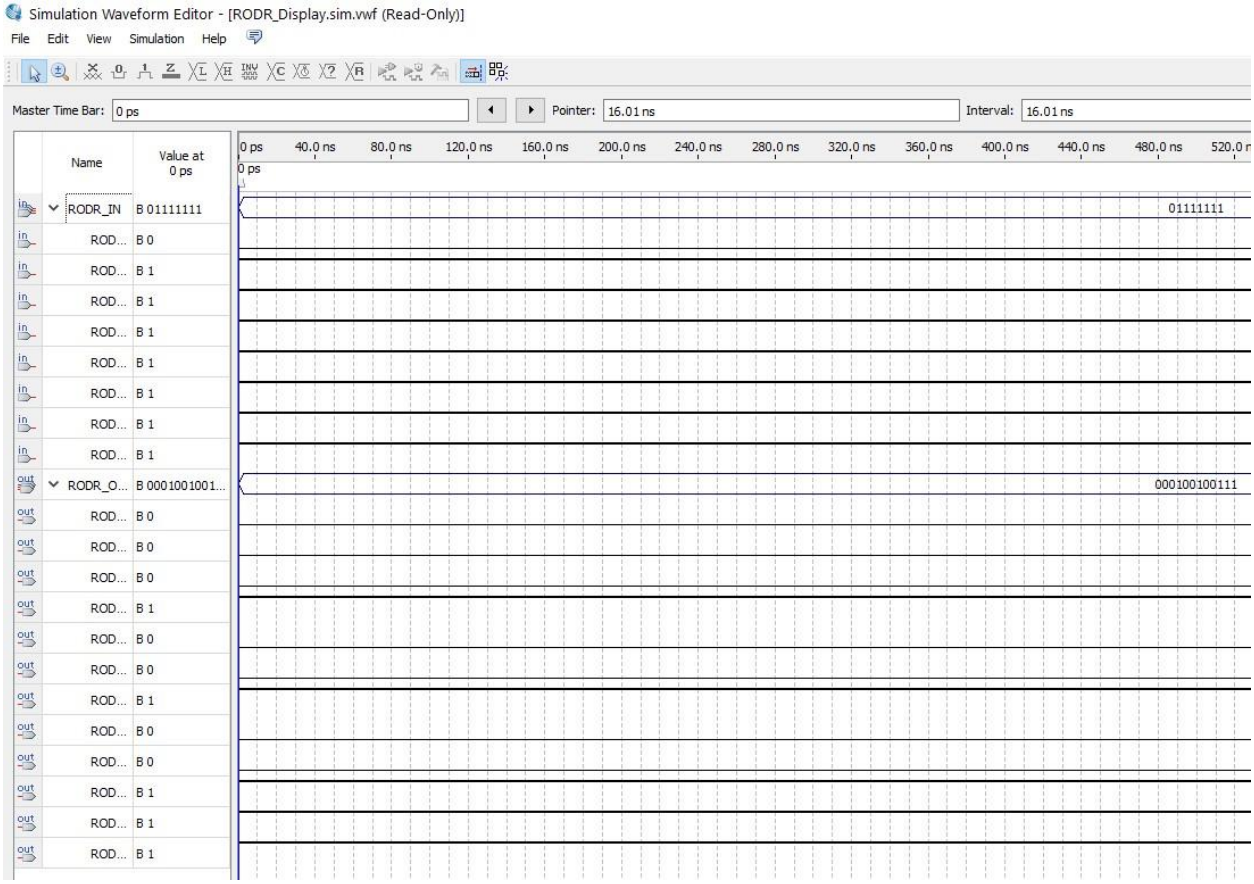


Figure 31: Binary to BCD Decoder output waveform

This waveform showcases the Binary to BCD decoder. It takes in a hypothetical decimal 127 from the Signed/Unsigned circuit and then converts it into a 12-bit binary number. The logic behind this is shown in the previous section. The 12-bits will then be pared into 4 bits signifying the ones, tens, and hundreds place for the BCD to Seven Segment decoder.

BCD to Seven Segment Decoder + Sign Decoder

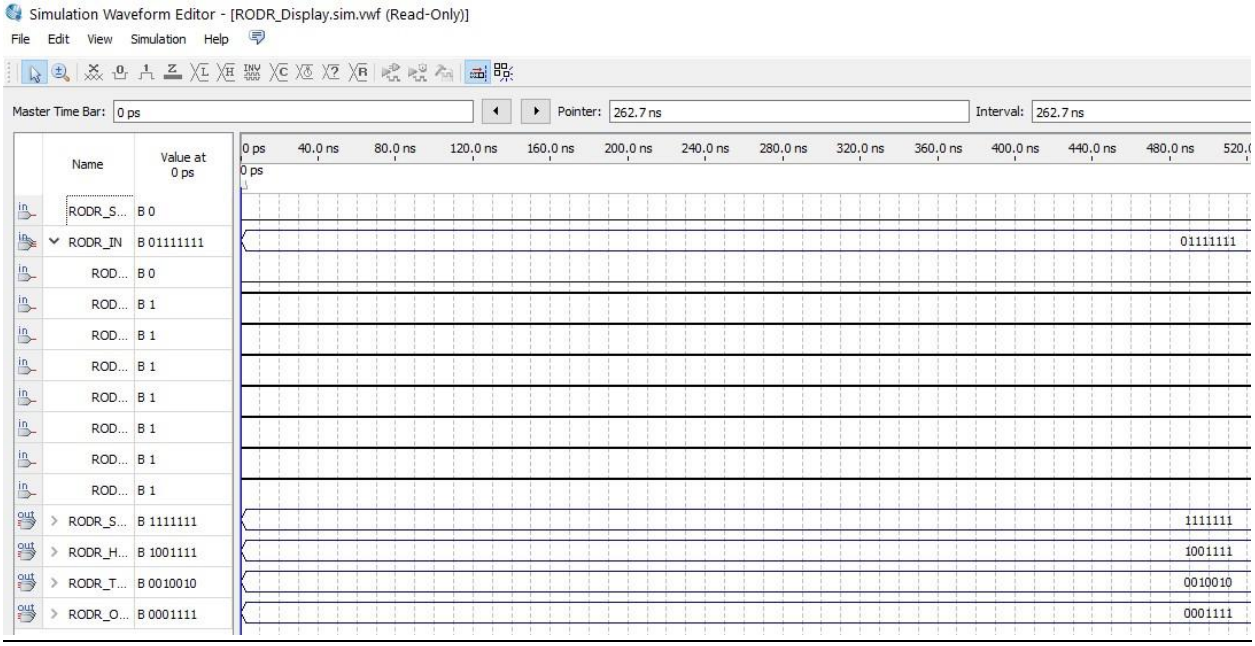


Figure 32: BCD Decoder without sign

This waveform showcases how the BCD to Seven Segment Decoder works along with the Sign Decoder. The BCD to Seven Segment Decoder takes in **RODR_IN** and converts it using the Binary to BCD decoder. It then parses it into **RODR_HUNDREDS, RODR_TENS, and RODR_ONES** that will then use the BCD to Seven Segment decoder to translate it and be sent to their respective displays. Because **RODR_SIGBIT** is a logical ‘0’, **RODR_SIGN** that specifies the sign of the decimal number is turned off.

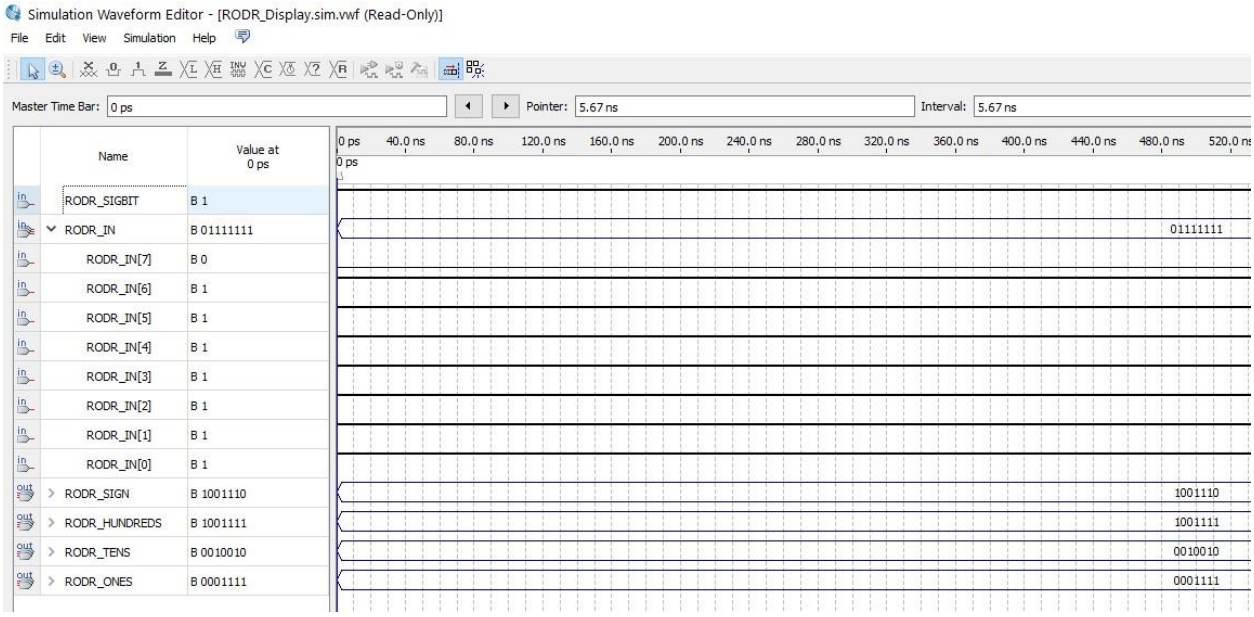


Figure 33: BCD Decoder with positive sign

In this case the only difference is the sign that is being displayed through the Plus/Minus Decoder. Due to the sign being changed, the binary output to the 4th segment display will show a "plus" sign.

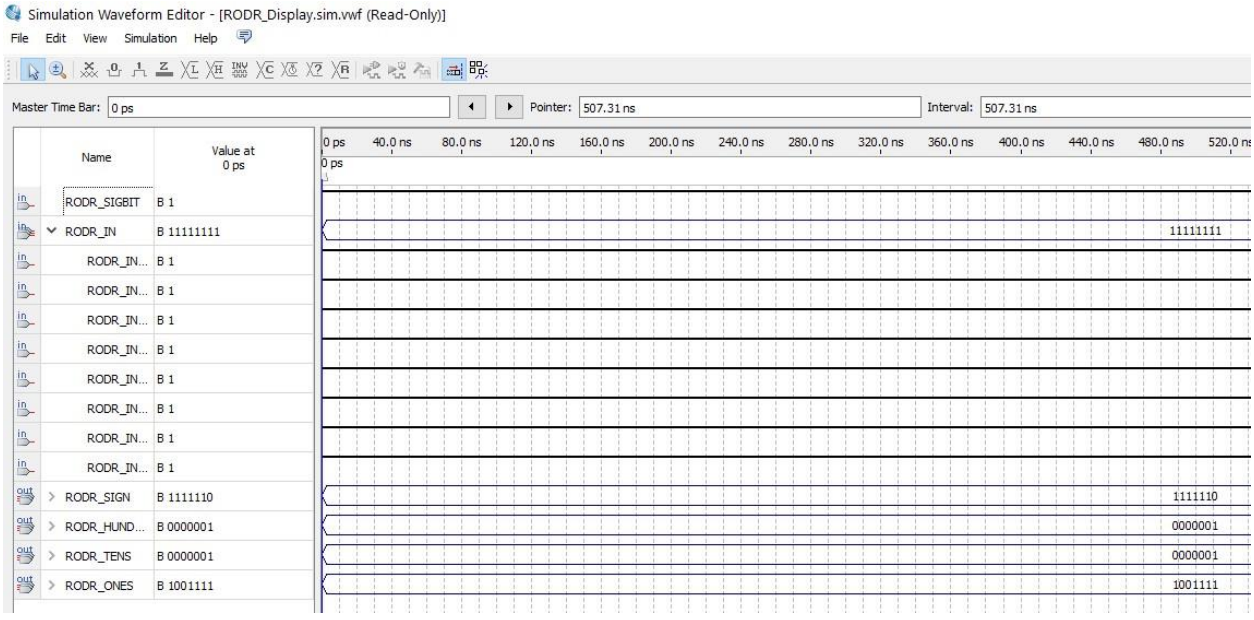


Figure 34: BCD Decoder with negative sign

In this case the Plus/Minus Decoder will display a negative sign as the **RODR_SIGBIT** is a logical ‘1’ and the most significant bit from **RODR_IN** is a logical ‘1’.

Note: The “No Output Decoder” takes in no inputs, so a waveform cannot be generated. It simply acts as a BCD to Seven Segment Decoder that has output of all logical ‘1’.

Board Operation (DE2 & DE1-SOC)

DE2

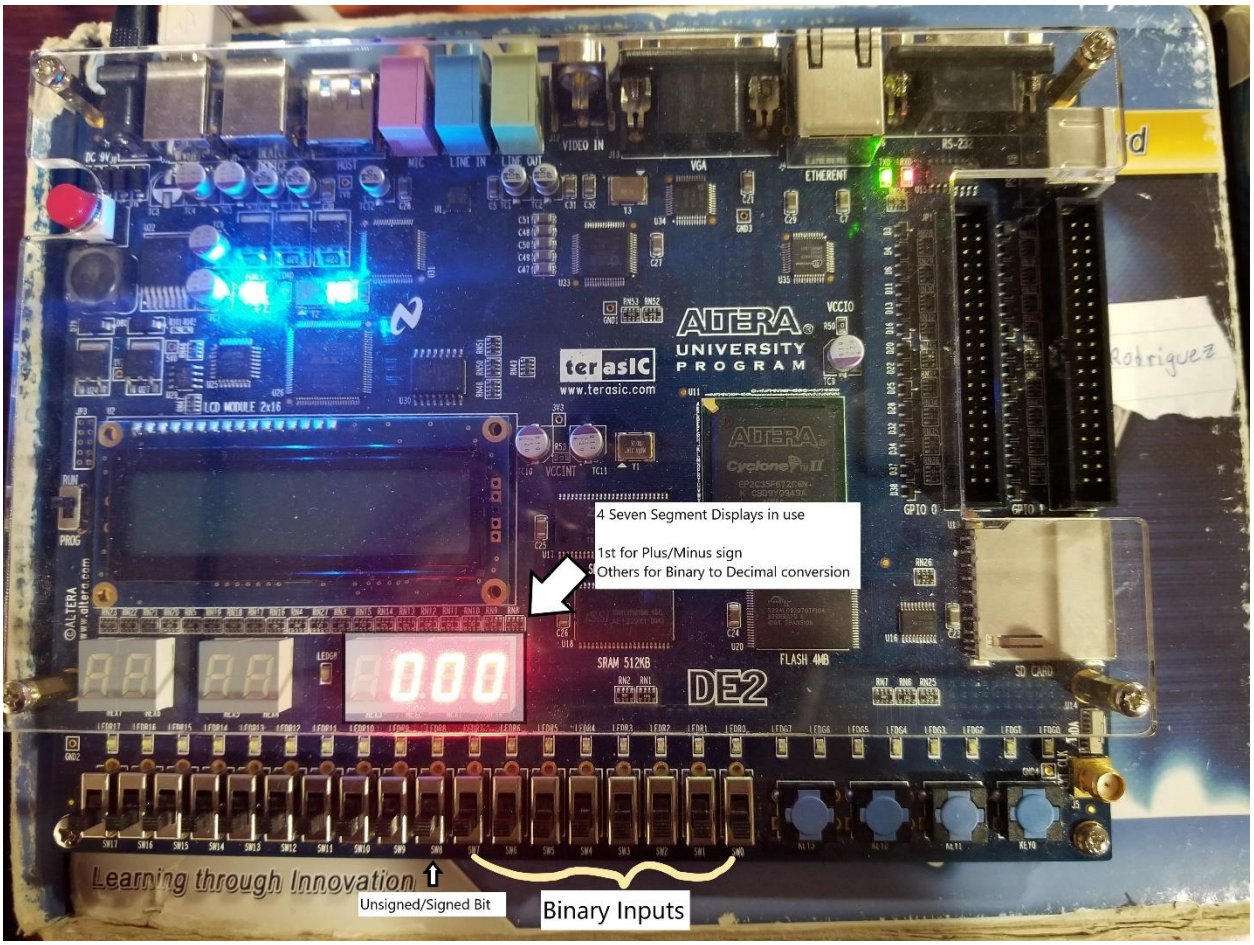


Figure 35: Layout of Switches on DE2 Board

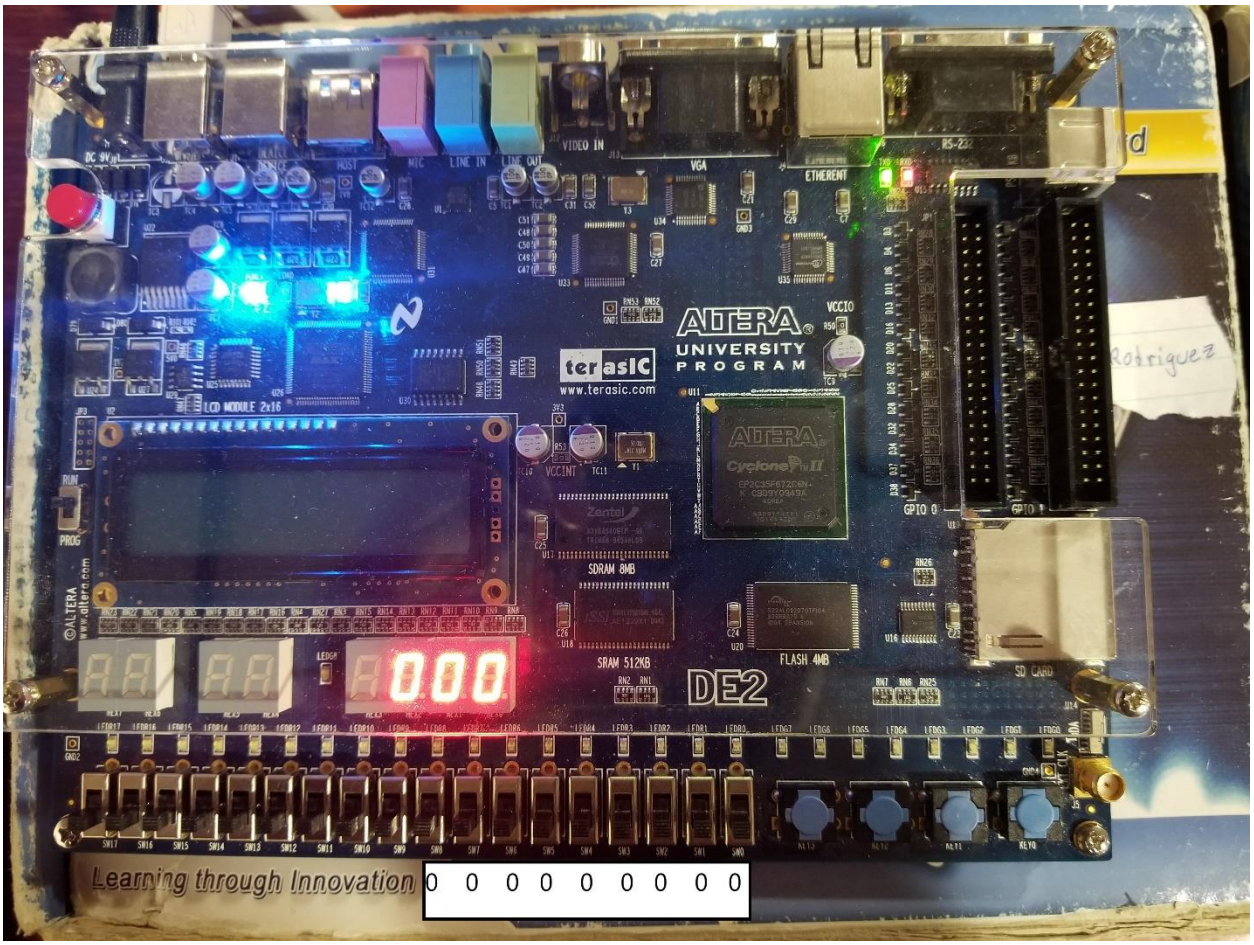


Figure 36: DE2 board showing the number 0 with all switches turned off

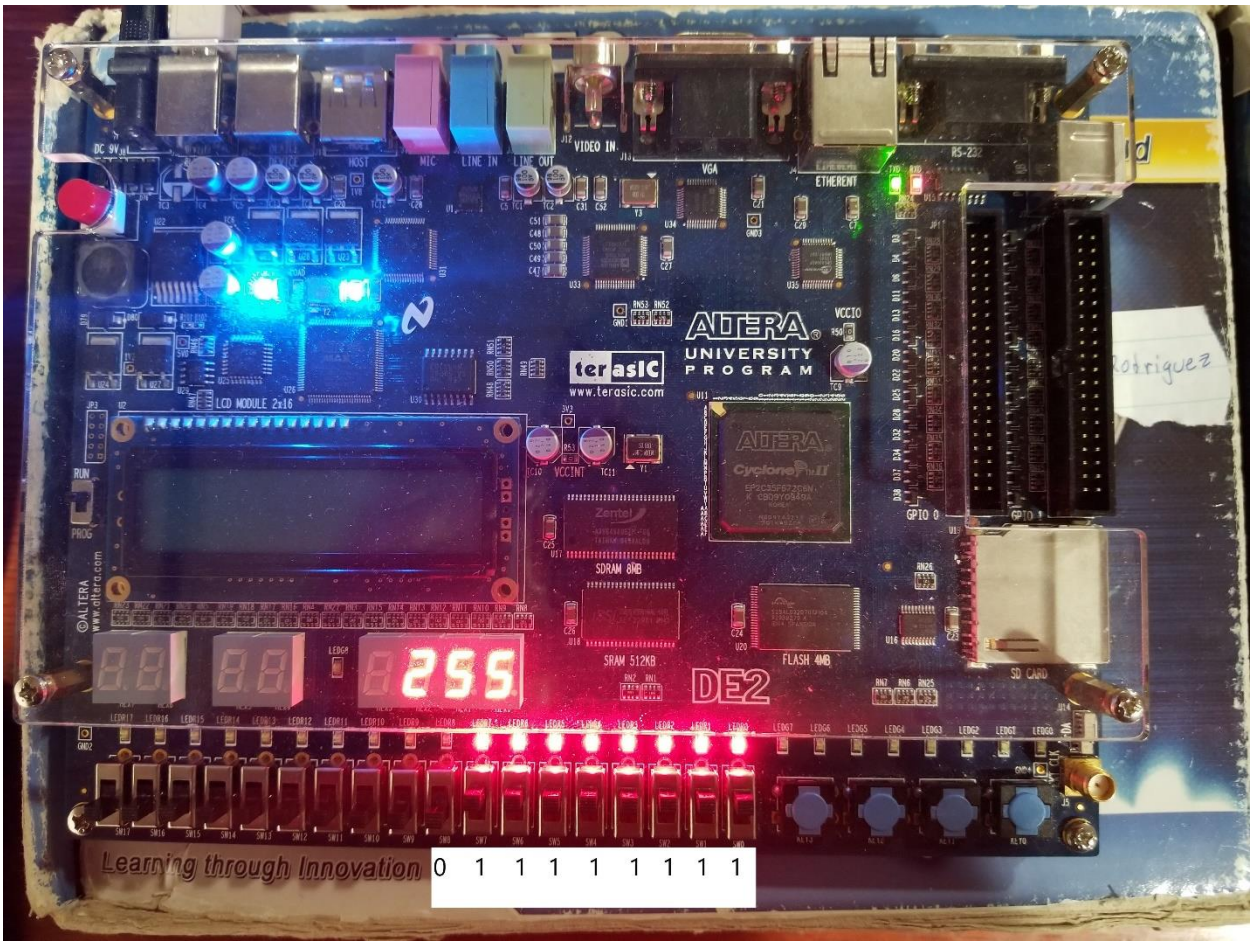


Figure 37: DE2 Board showing the number 255 with the unsigned/signed bit off and all 1's in binary.

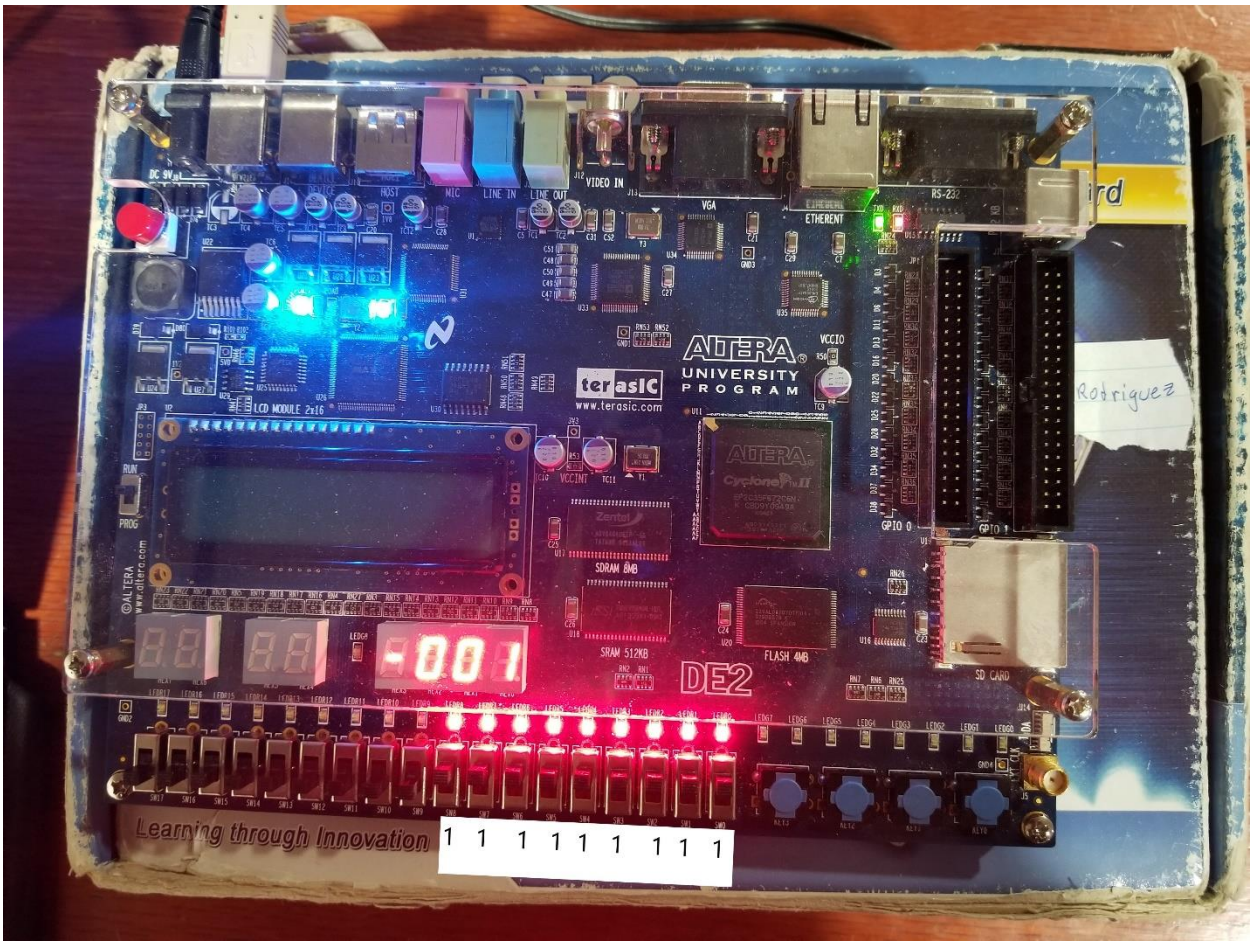


Figure 38: DE2 Board showing the number -1 The unsigned/signed bit is '1' along with all 1s for the binary input. Two's complement is applied.

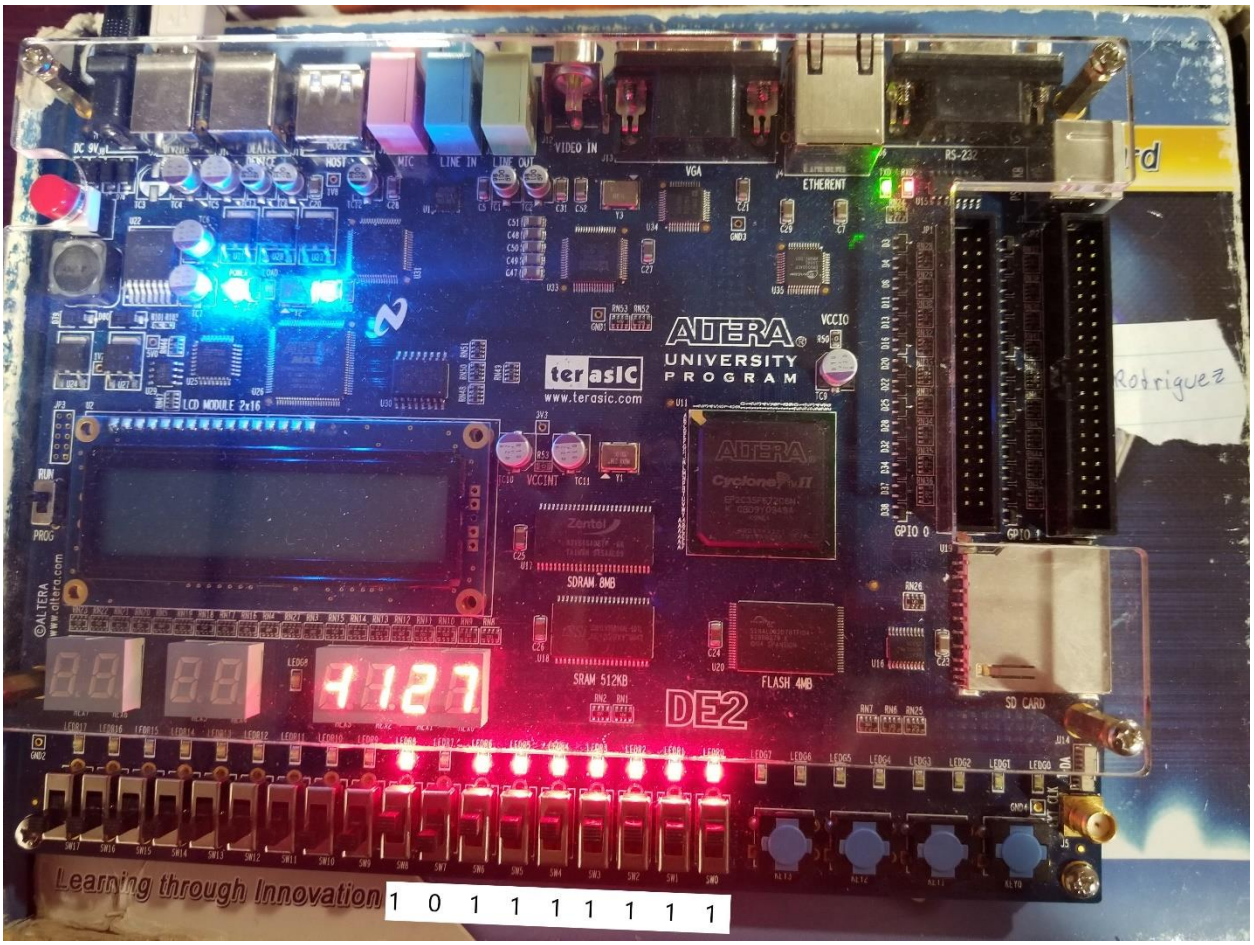


Figure 39: DE2 Board showing +127. The unsigned/signed bit is on followed by "01111111". Two's complement is not implemented as this is the maximum allowable positive number.

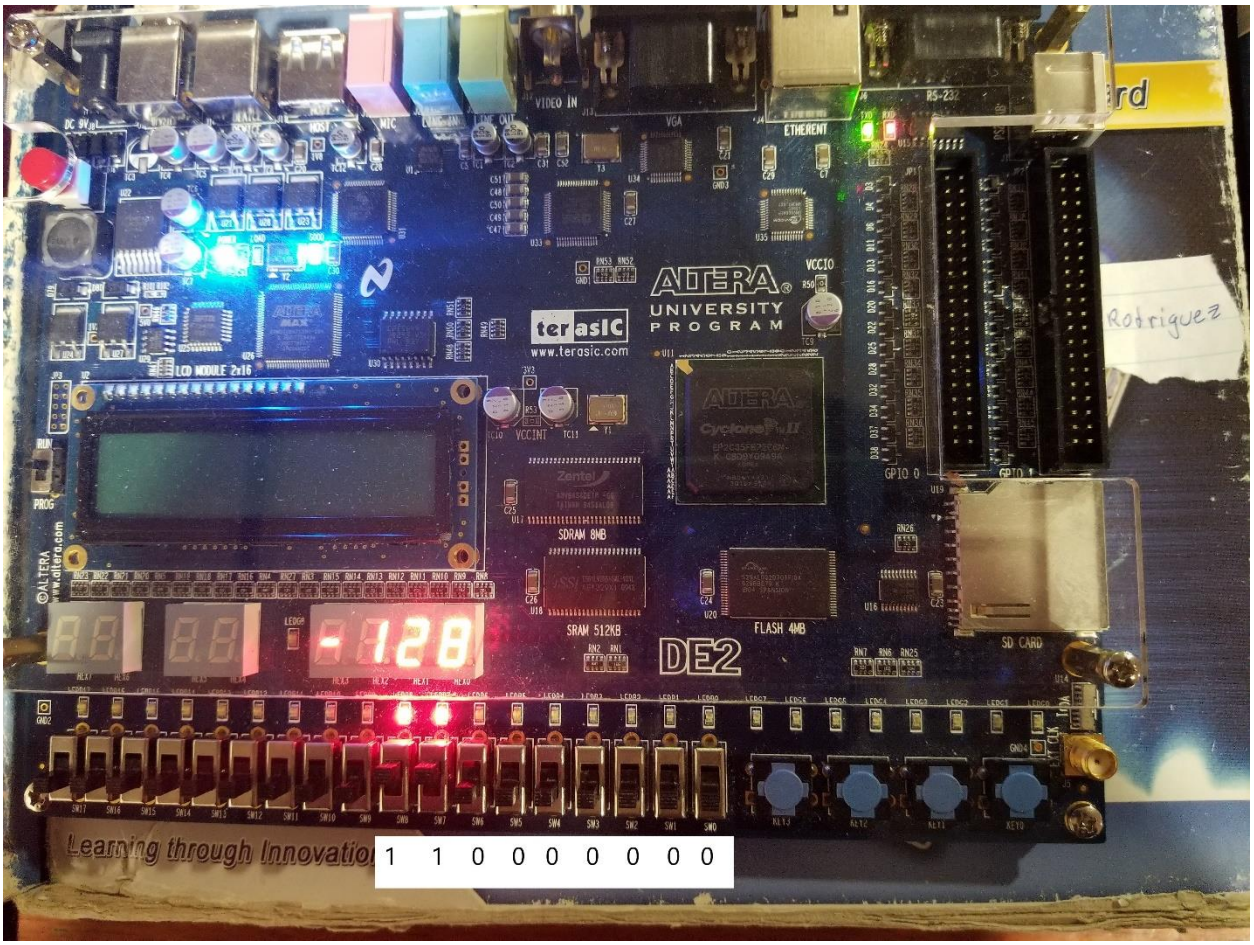


Figure 40: DE2 board showing the lowest possible number in binary -128. The unsigned/signed switch is on along with the most significant bit. Two's complement is applied.

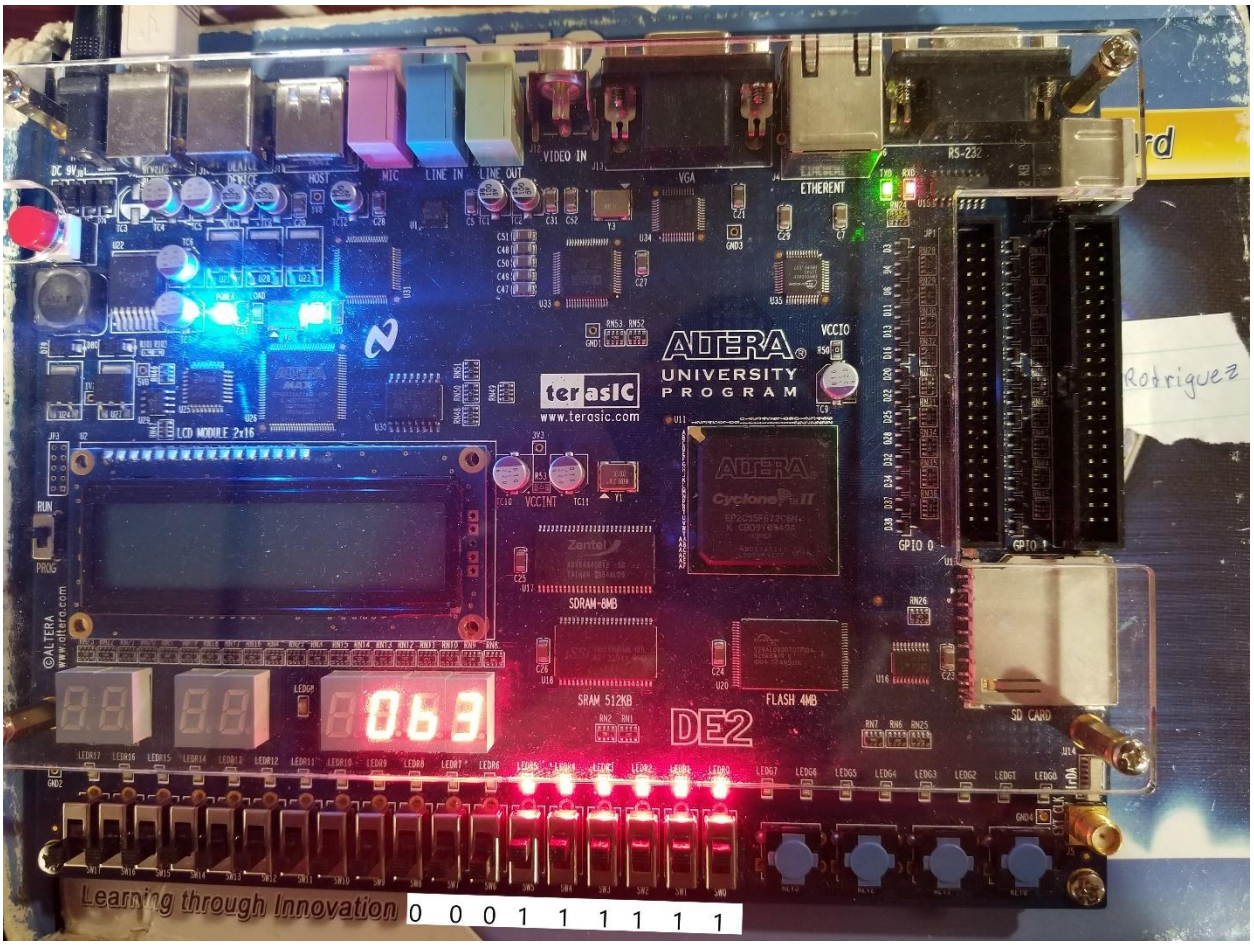


Figure 41: DE2 Board showing the number 63 unsigned. The unsigned/signed bit is off followed by the binary "00111111" which is converted to decimal number 63.

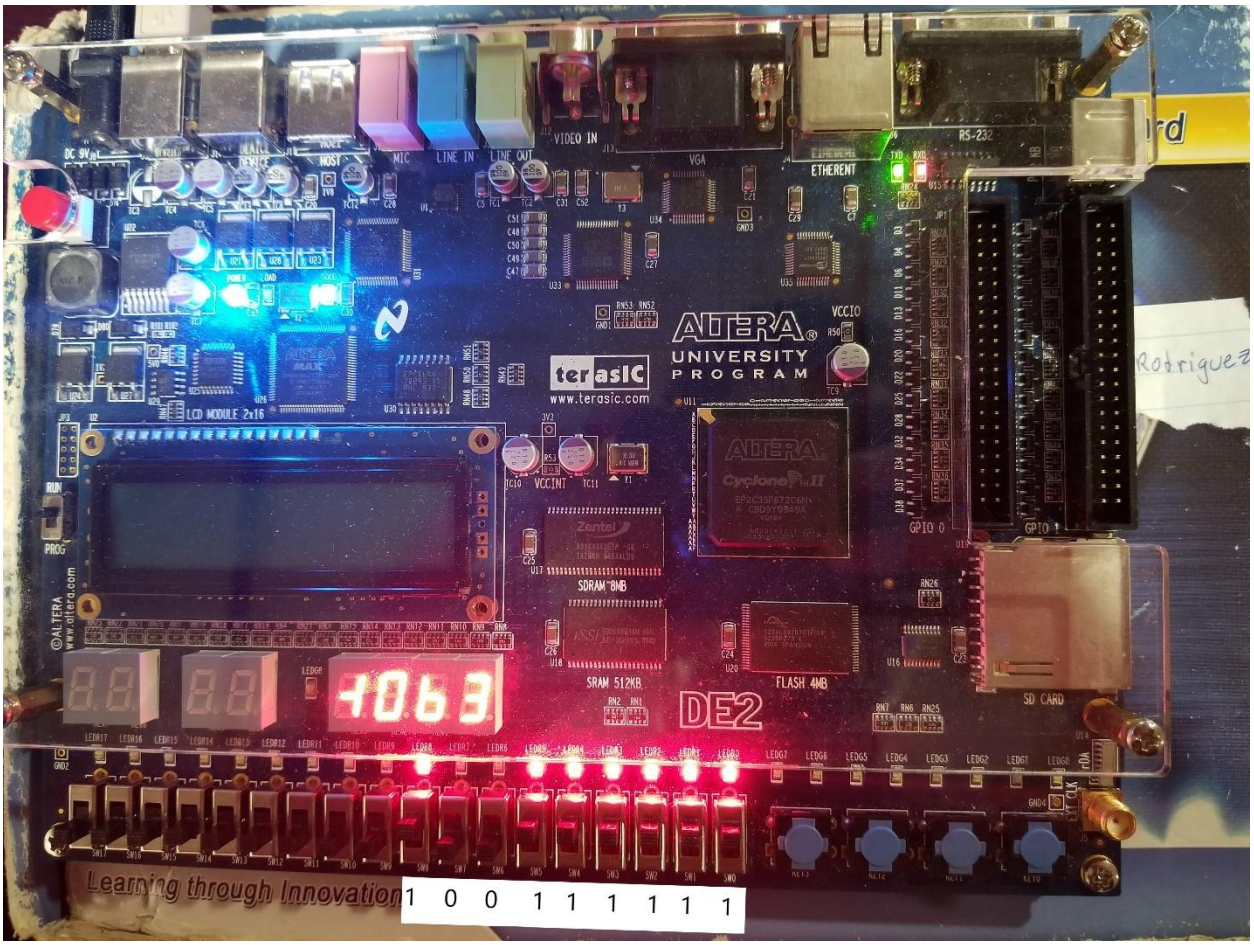


Figure 42: DE2 board showing +63. The unsigned/signed bit is up followed by the binary of "00111111". There is no use of two's complement for it is within the range of +127.

DE1-SOC

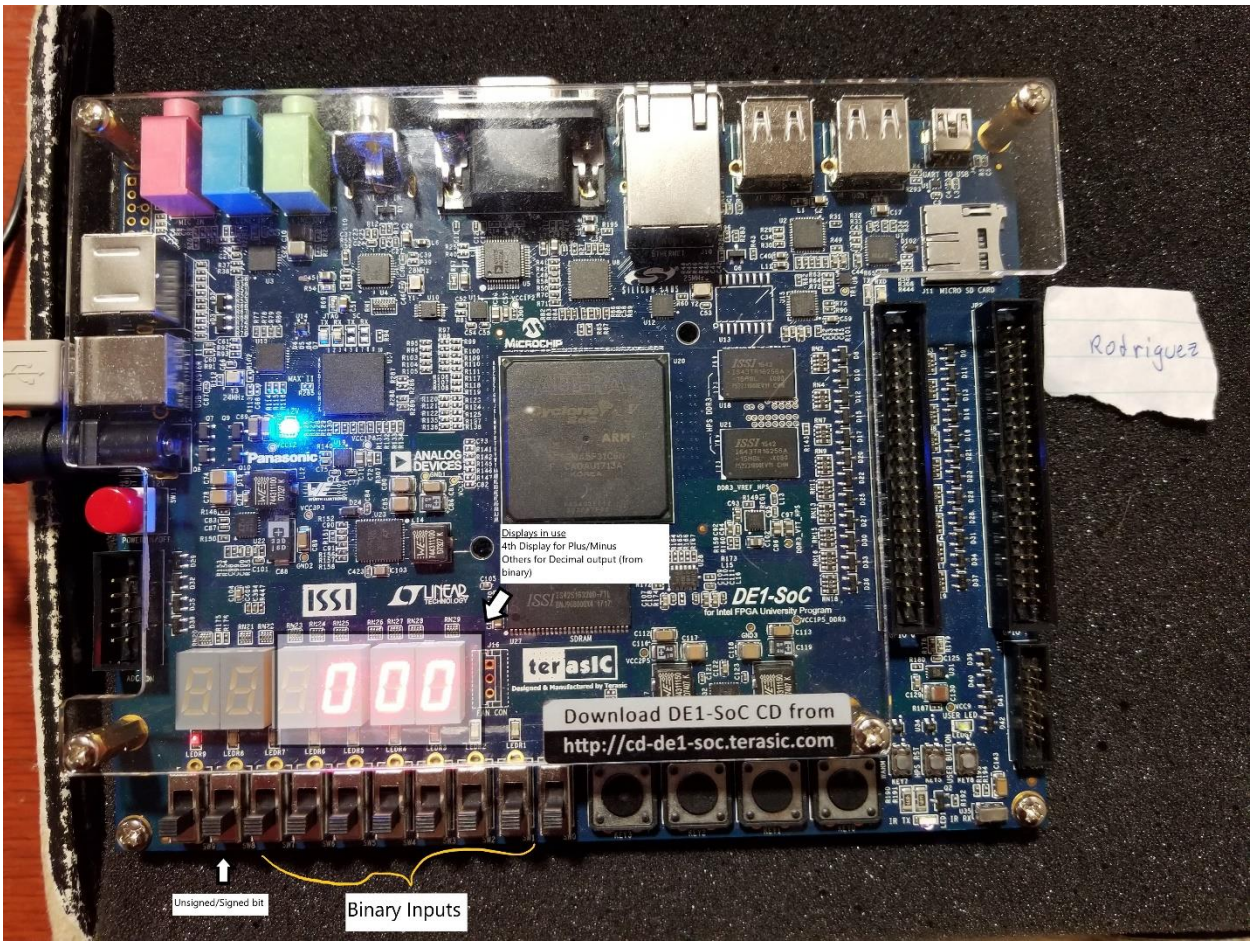


Figure 43: Layout used for DE1-SOC board

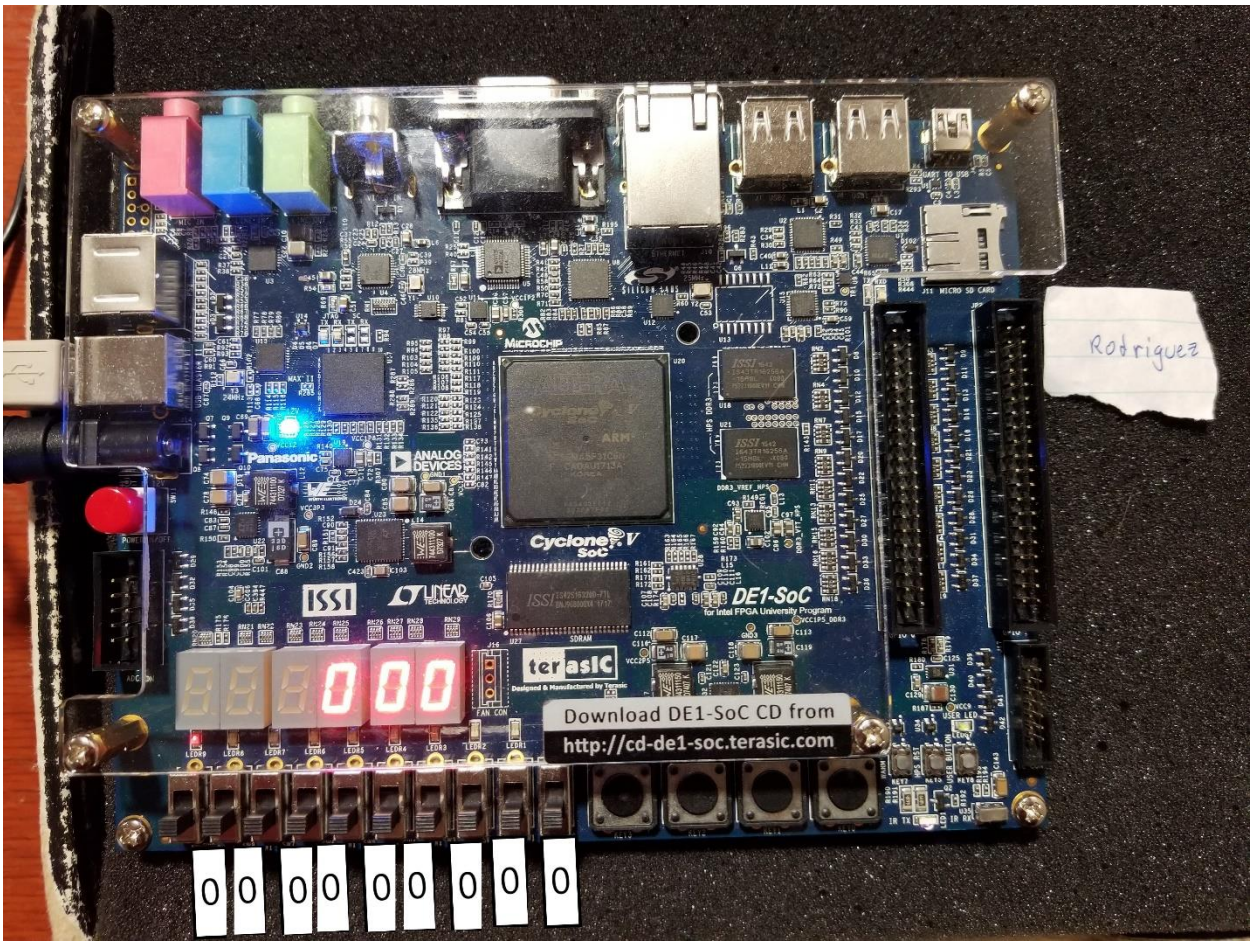


Figure 44: DE1-SOC board having all switches off, showing the number 0 on the segment displays

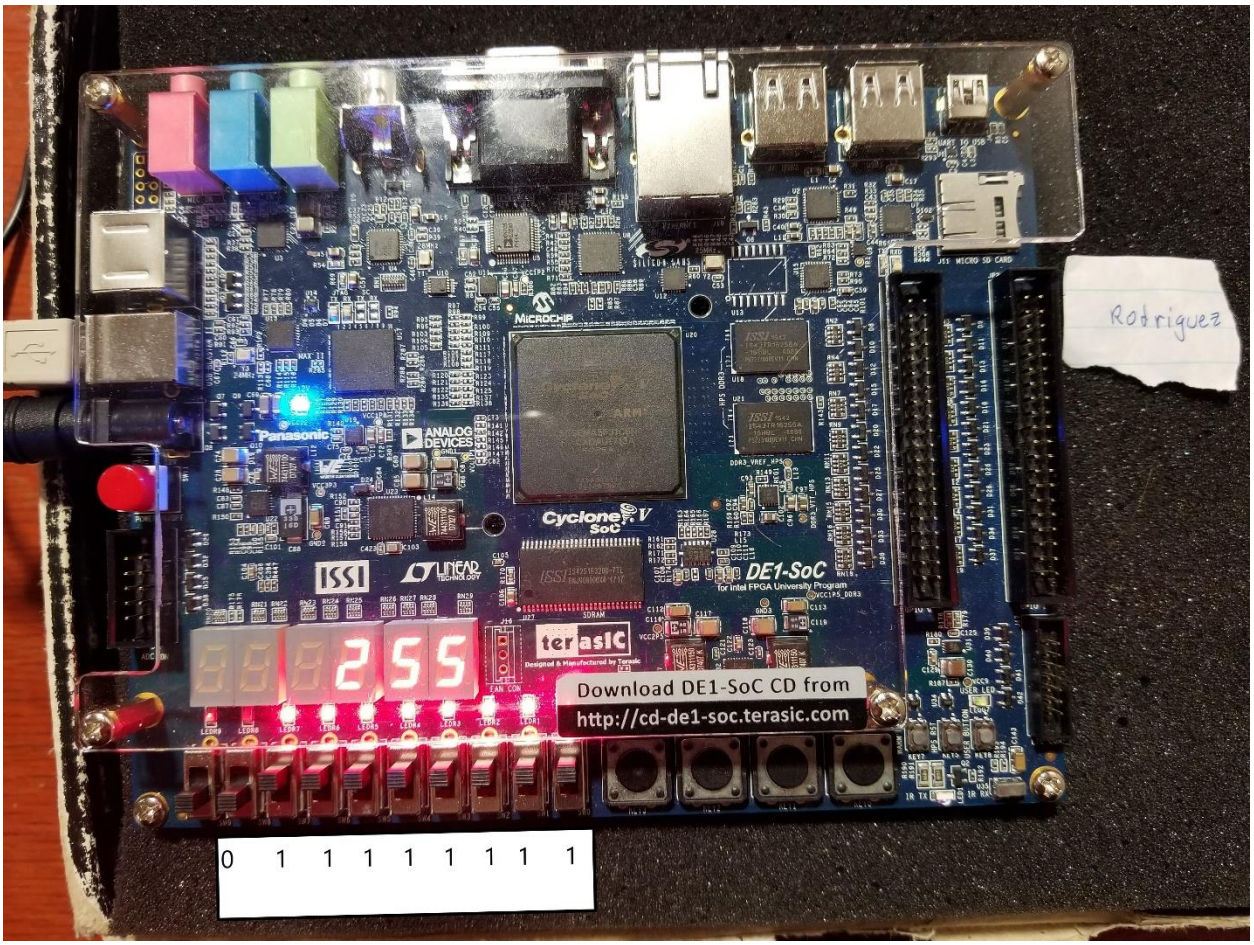


Figure 45: DE1-SOC board showing the number 255 by turning the unsigned/signed bit off followed by all 1s.

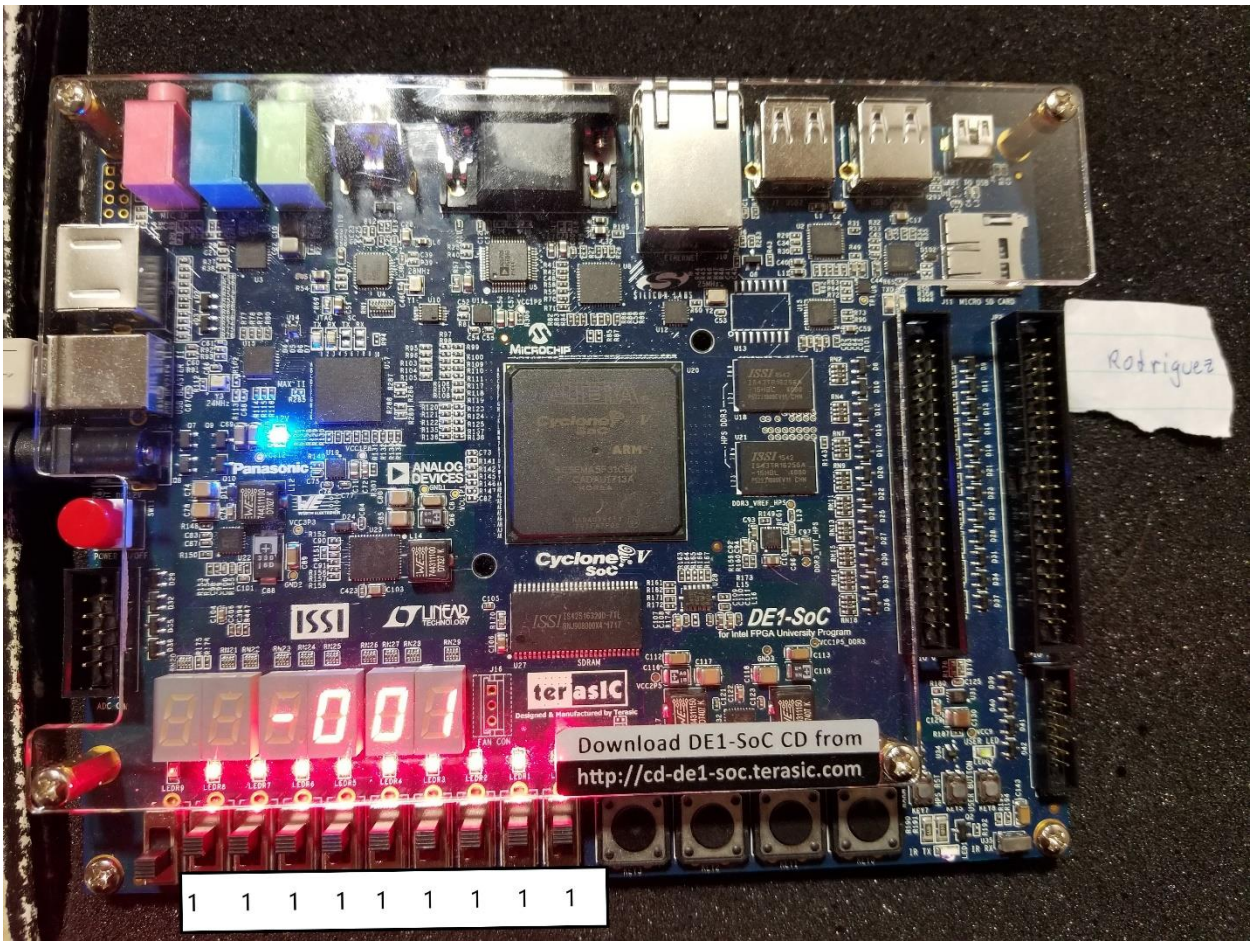


Figure 46:DE1-SOC board showing -1. This is done by having all bits turned on. Two's complement is applied to convert from 255 to -1.

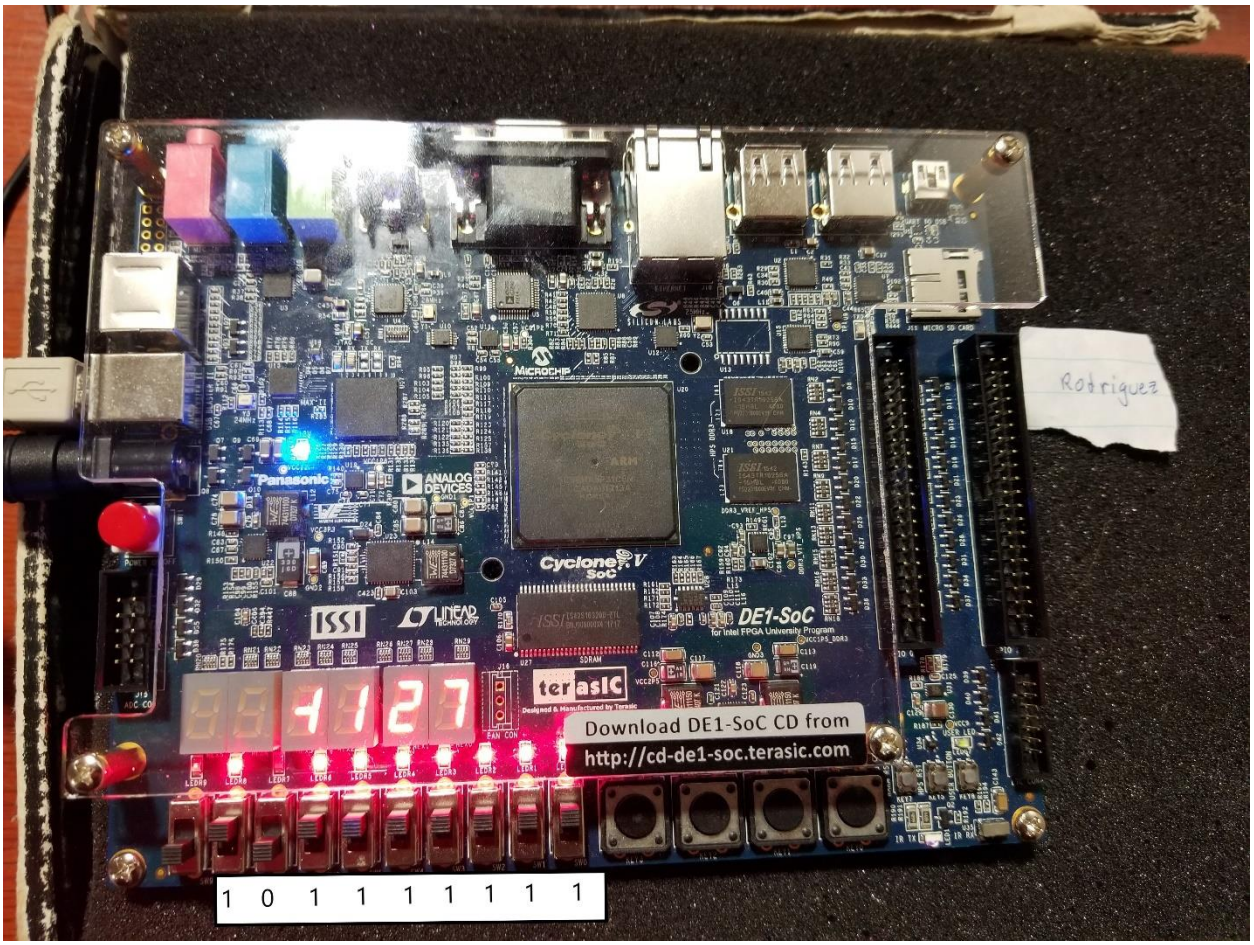


Figure 47: DE1-SOC board showing +127. Done by turning the unsigned/signed switch on followed by the binary for 127 (“01111111”). This is allowed for 127 is the highest positive number allowed in signed mode.

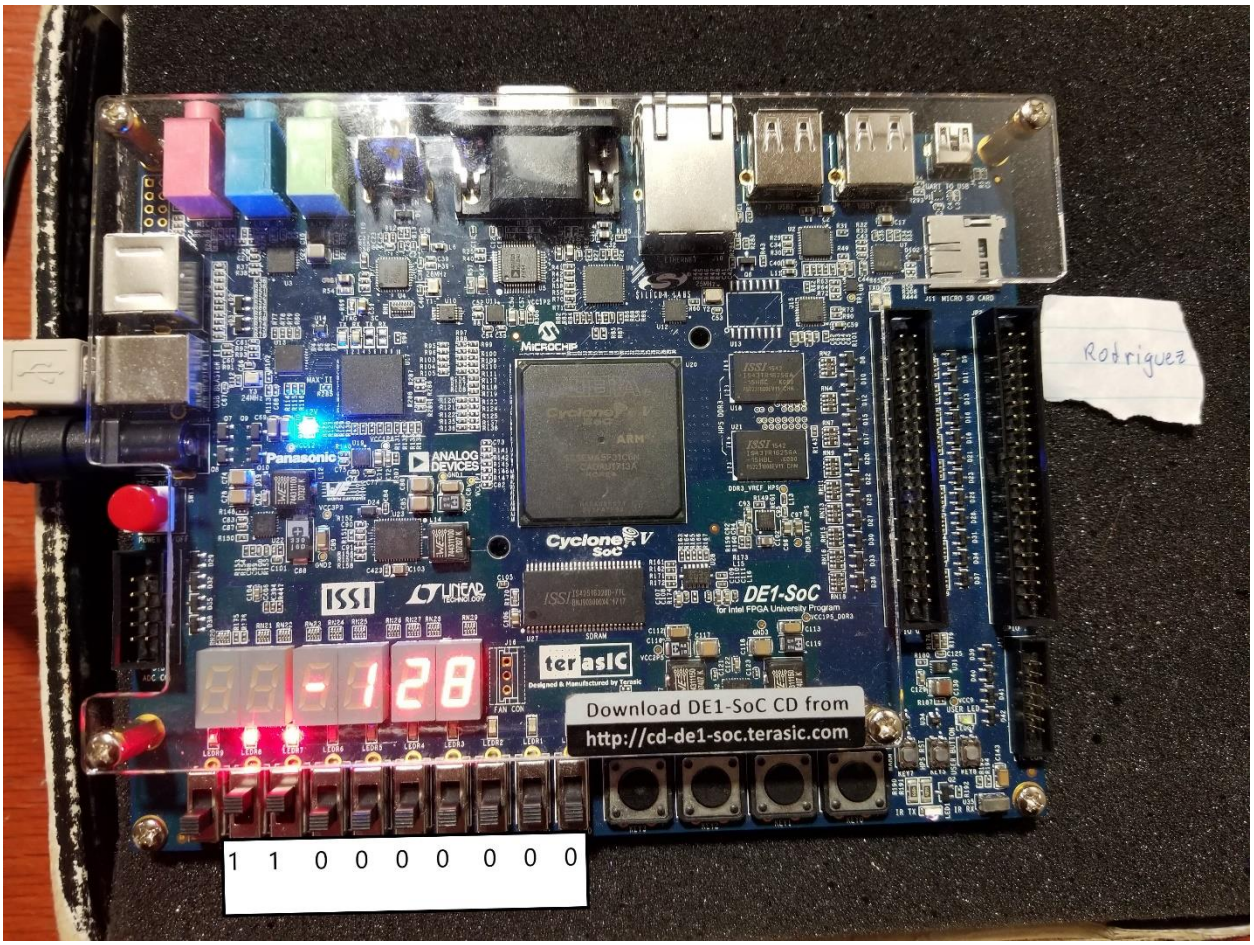


Figure 48: DE1-SOC board showing -128. This is done by having the unsigned/signed bit and most significant bit turned on. Two's complement is applied to display the number.

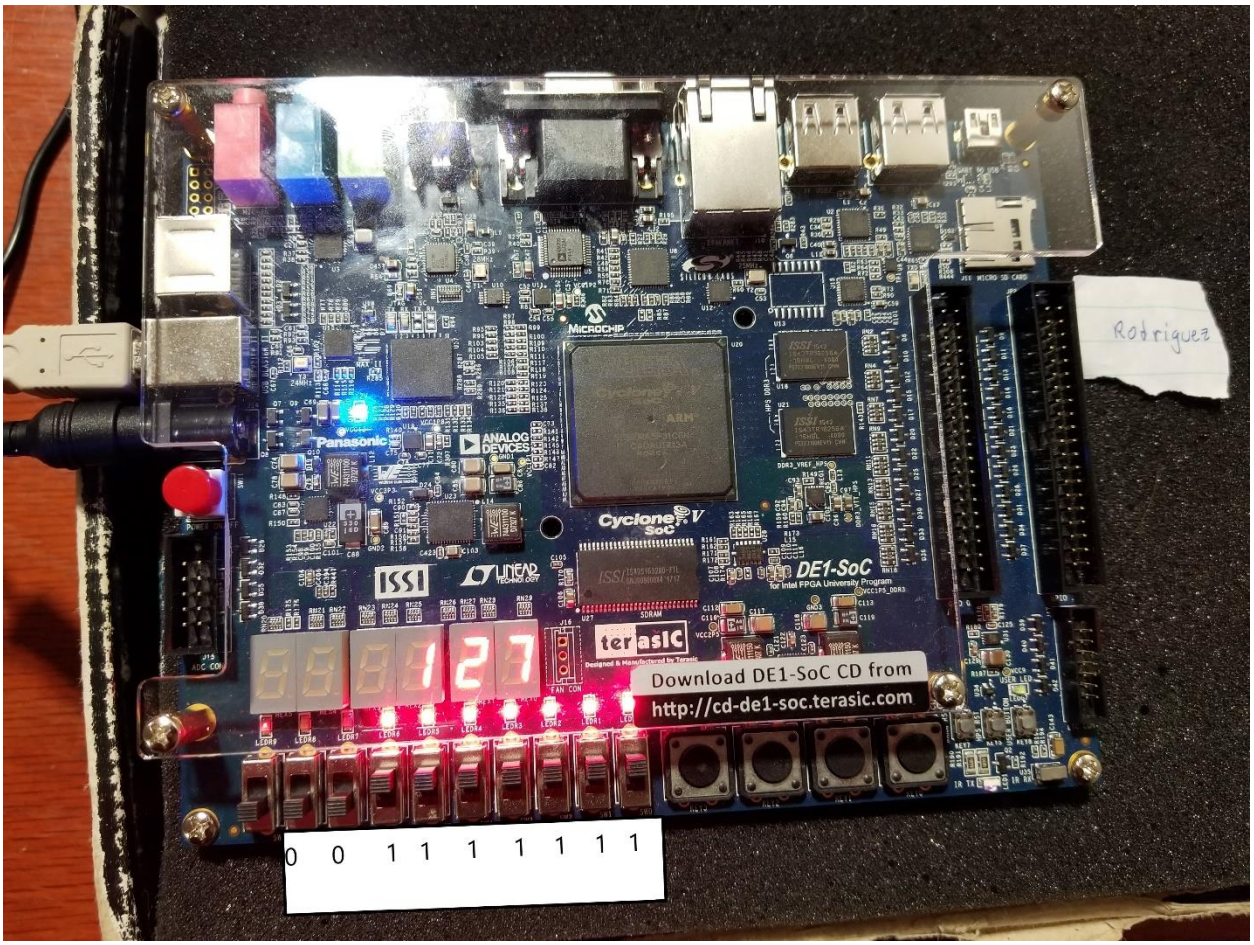


Figure 49: DE1-SOC board showing 127. The unsigned/signed bit is off followed by the binary for 127 ("01111111"). No sign is shown as it is implied that 127 is a positive number.

Conclusion

In this laboratory exercise, I have learned how to use Quartus software to create projects using VHDL how to utilize the DE2 and DE1-SOC boards. By creating the Unsigned/Signed circuit, it taught me how two's complement works at the gate level and use AND gates to add the carry whenever necessary if the input bits are to be in signed mode. By creating the Binary to BCD Decoder, it taught me the "Shift & Add 3" algorithm that converts 8-bit binary to 12-bits with every 4 sequential bits (from right to left) converting into the appropriate decimal number to the segment displays. Creating the BCD to seven segment decoder (including the plus/minus decoder and No Output Decoder) taught me how to light up parts of the display with the board manuals and setting pin assignments accordingly. The master file taught me how to combine all my VHDL files into one and ensure that the inputs and outputs correspond to the right circuits.