Lab 8

EE214: Digital Circuits Laboratory

B. Siddharth Prabhu
Roll No. 200010003

18 February 2022

1 Aim

Design and execution of VHDL code for the given logic systems, viz., 3-bit Adder/Subtractor, 3-bit ALU, and 4:2 Priority Encoder.

2 Summary of the experiment

Design and realization of 3-bit Adder/Subtractor, 3-bit ALU (Arithmetic and Logic Unit), and 4:2 priority encoder (with active high enable pin), then implementation using structural, and behavioural modelling styles respectively.

3 Components Used

Helium v1.1 (MAX 3000 architecture), PC with Altera Quartus Simulator Software, USB Type-B cable, conducting wires.

4 Problem Statements

4.1 3-bit Adder/Subtractor

Write VHDL code for a 3-bit Adder/Subtractor in structural modelling style. The Adder/Subtractor operation is controlled by signal 'm'. VHDL code for 1-bit full adder in dataflow must be written in the same VHDL file.

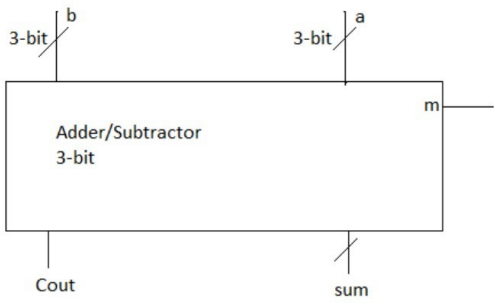


Figure 1: Representation of Adder-Subtractor Circuit

4.2 3-bit Arithmetic and Logic Unit (ALU)

Write VHDL code for ALU which performs the following operations on two 3-bit inputs A and B, depending on selection lines. Use behavioural modelling style.

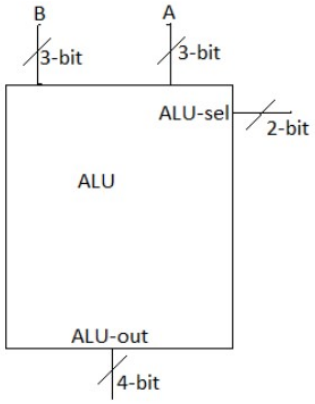


Figure 2: Representation of ALU based on 2-bit selector line

ALU selector line 1	ALU selector line 2	Operation selected
0	0	$A + B$
0	1	$A - B$
1	0	A bitwise AND B
1	1	A bitwise XOR B

Table 1: ALU operations based on select lines

4.3 4:2 Priority Encoder

Design and implement VHDL code for a 4:2 priority encoder (with active high enable pin), using behavioural modelling style.

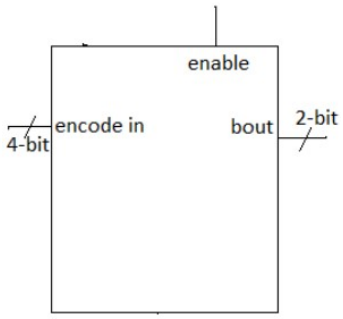


Figure 3: Representation of 4:2 Priority Encoder

5 Design Procedure: Controls and Truth Table

5.1 3-bit Adder/Subtractor

If m is 0, then the XOR gate within it will pass on value of B, and $A + B$ will be given as output. If m is 1, then the XOR gate within it will complement the bits of B to give its one's complement, and 1 will be added as initial carry to obtain two's complement of B. Adding A to 2's complement of B results in $A - B$.

m	Operation
0	$A + B$
1	$A - B$ (2's Complement form)

Table 2: Role of Control 'm' in 3-bit Adder/Subtractor

5.2 3-bit Arithmetic and Logic Unit

If we reuse the 3-bit Adder/Subtractor described and implemented in the first problem statement, then we can account for the ALU Selects 00 and 01. (Selector line 1 is more significant bit.) Otherwise, we can write lines of code that describe the process of the full adder/subtractor for each bit. For bitwise operations, we can directly use $A\ AND\ B$ and $A\ XOR\ B$.

As illustrated in the code of section 6.2, we have considered "behavioral" to mean that the architecture of the main entity is defined with no sub-components.

5.3 4:2 Priority Encoder

A 4-to-2 priority encoder has the following truth table:

enable	in_0	in_1	in_2	in_3	out_0	out_1
0	X	X	X	X	0	0
1	1	X	X	X	0	0
1	0	1	X	X	0	1
1	0	0	1	X	1	0
1	0	0	0	1	1	1

Table 3: Truth table for Priority Encoder (4:2)

This can directly be implemented in behavioral modeling style, as shown in section 6.3.

In a 4:2 Priority encoder, if a more significant bit is high, then the less significant bits are not considered at all. The priority to decide the outputs is with the bits in their order of significance. Here, the order of significance/priority is in_0 $>$ in_1 $>$ in_2 $>$ in_3 .

6 VHDL Code

6.1 3-bit Adder/Subtractor

```
-----------------------------------------------
---------- CODE FOR 1-BIT FULL ADDER ----------
----------------- (DATAFLOW) ------------------
library ieee;
use ieee.std_logic_1164.all;

entity fulladder is
port(
    i1, i2, i3: in std_logic;
    o1, o2: out std_logic);
end fulladder;

architecture fa_dat of fulladder is
begin
    o1 <= i1 xor i2 xor i3;
    o2 <= (i1 and i2) or ((i1 xor i2) and i3);
end fa_dat;

-----------------------------------------------
------- CODE FOR 3-BIT ADDER/SUBTRACTOR -------
---------------- (STRUCTURAL) -----------------
library ieee;
use ieee.std_logic_1164.all;

entity adsub3 is
port(
    a: in std_logic_vector (2 downto 0);
    b: in std_logic_vector (2 downto 0);
    m: in std_logic;
    sum: out std_logic_vector (2 downto 0);
    cout: out std_logic);
end adsub3;


architecture struct of adsub3 is

    ------- COMPONENT DEFINITION -------
    component fulladder is
    port(
        i1, i2, i3: in std_logic;
        o1, o2: out std_logic);
    end component;

    ------- INTERMEDIATE SIGNALS -------
    signal c1: std_logic;
    signal c2: std_logic;
    signal bb: std_logic_vector (2 downto 0);
```

```
------- STRUCTURAL MODELING CODE -------
begin
    bb(0) <= b(0) xor m;
    bb(1) <= b(1) xor m;
    bb(2) <= b(2) xor m;

    u1: fulladder port map ( a(0), bb(0), m, sum(0), c1 );
    u2: fulladder port map ( a(1), bb(1), c1, sum(1), c2 );
    u3: fulladder port map ( a(2), bb(2), c2, sum(2), cout );

end struct;
```

6.2 3-bit Arithmetic and Logic Unit (ALU)

```
----------------------------------------------------
------- 3-bit ALU using behavioral modelling --------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;

entity ALU is
port(
    A, B: in STD_LOGIC_VECTOR(2 downto 0);      -- 2 inputs of 3 bits each
    ALU_Sel: in STD_LOGIC_VECTOR(1 downto 0);
    ALU_Out: out STD_LOGIC_VECTOR(3 downto 0) );
end ALU;

architecture alu_beh of ALU is

    ------- INTERMEDIATE SIGNALS -------
    signal temp1: std_logic;
    signal temp2: std_logic;

begin
    c1: process(A, B, ALU_Sel)
    begin
        case ALU_Sel is
            when "00" =>
                ALU_Out(0) <= A(0) XOR B(0);
                temp1 <= A(0) AND B(0);
                ALU_Out(1) <= (A(1) XOR B(1)) XOR temp1;
                temp2 <= ((A(1) AND B(1)) OR ((A(1) AND temp1) OR (B(1) AND
                ↪  temp1));
                ALU_Out(2) <= (A(2) XOR B(2)) XOR temp2;
                ALU_Out(3) <= ((A(2) AND B(2)) OR ((A(2) AND temp2) OR (B(2) AND
                ↪  temp2));
```

```
            when "01" =>
                ALU_Out(0) <= (A(0) XOR (NOT B(0))) XOR '1';
                temp1 <= ((A(1) AND (NOT B(1))) OR ((A(1) AND '1') OR ((NOT B(1))
                ↪  AND '1'));
                ALU_Out(1) <= (A(1) XOR (NOT B(1))) XOR temp1;
                temp2 <= ((A(1) AND (NOT B(1))) OR ((A(1) AND temp1) OR ((NOT
                ↪  B(1)) AND temp1));
                ALU_Out(2) <= (A(2) XOR (NOT B(2))) XOR temp2;
                ALU_Out(3) <= ((A(2) AND (NOT B(2))) OR ((A(2) AND temp2) OR ((NOT
                ↪  B(2)) AND temp2));
            when "10" =>
                ALU_Out <= '0' & (A AND B);
            when "11" =>
                ALU_Out <= '0' & (A XOR B);
        end case;
    end process c1;
end alu_beh;
```

6.3 4:2 Priority Encoder

```
----------------------------------------------------
--------------- 4:2 PRIORITY ENCODER ---------------
library ieee;
use ieee.std_logic_1164.all;

entity encoder is
port(
    encoder_in : in std_logic_vector(3 downto 0);
    bin_out : out std_logic_vector(1 downto 0);
    enable : in std_logic);
end encoder;

architecture behavioral of encoder is
begin
    c1 : process(enable, encoder_in)
    begin
    if enable = '1' then
        if encoder_in(0) = '1' then
            bin_out(0) <= '0';
            bin_out(1) <= '0';
        elsif encoder_in(1) = '1' then
            bin_out(0) <= '0';
            bin_out(1) <= '1';
        elsif encoder_in(2) = '1' then
            bin_out(0) <= '1';
            bin_out(1) <= '0';
        elsif encoder_in(3) = '1' then
            bin_out(0) <= '1';
            bin_out(1) <= '1';
        end if;
```

```
    else
       bin_out(0) <= '0';
        bin_out(1) <= '0';
    end if;
    end process c1;
end behavioral;
```

7 Circuit and Simulation Snapshots

(RTL and Circuit snapshots are included in the later pages of this report.)

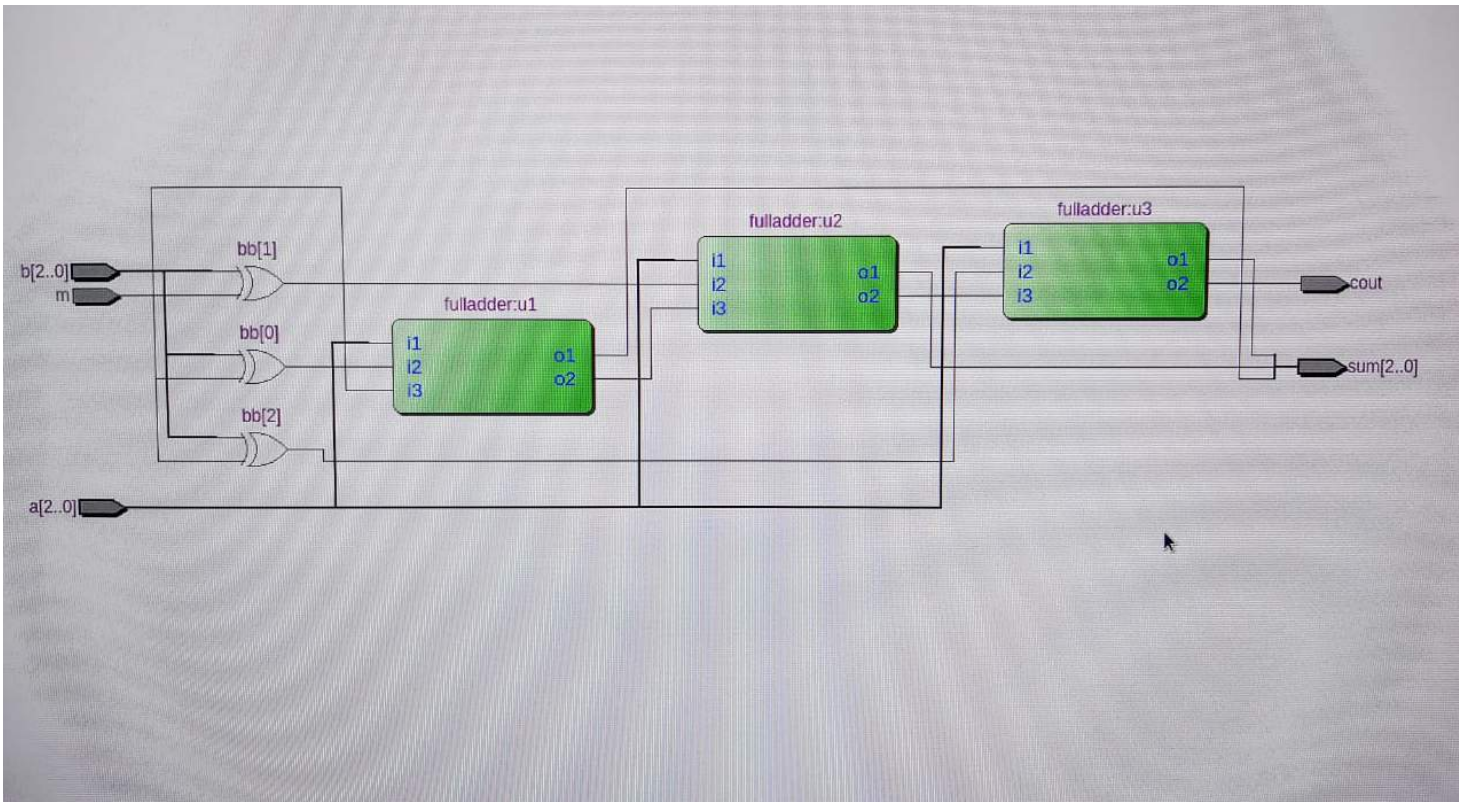
8 Results and Discussions

- We designed and implemented different logic circuits with the use of VHDL, via the Altera Quartus Software.
- The outputs obtained could be verified via truth tables and various examples by hand (for the adder/subtractor).
- CPLD greatly reduces time and effort required in designing and simulating logic circuits, like the ones we demonstrated.
- It is important to stick to proper syntax and good coding practices, so that execution and debugging goes smoothly.
- In real scenarios where we don't have constraints on modeling styles, we must opt for the one that is most time and cost effective; we may reuse code. For example, we could reuse the code from section 6.1 in simplifying the code of section 6.2, if the constraint of behavioral modeling was not present.

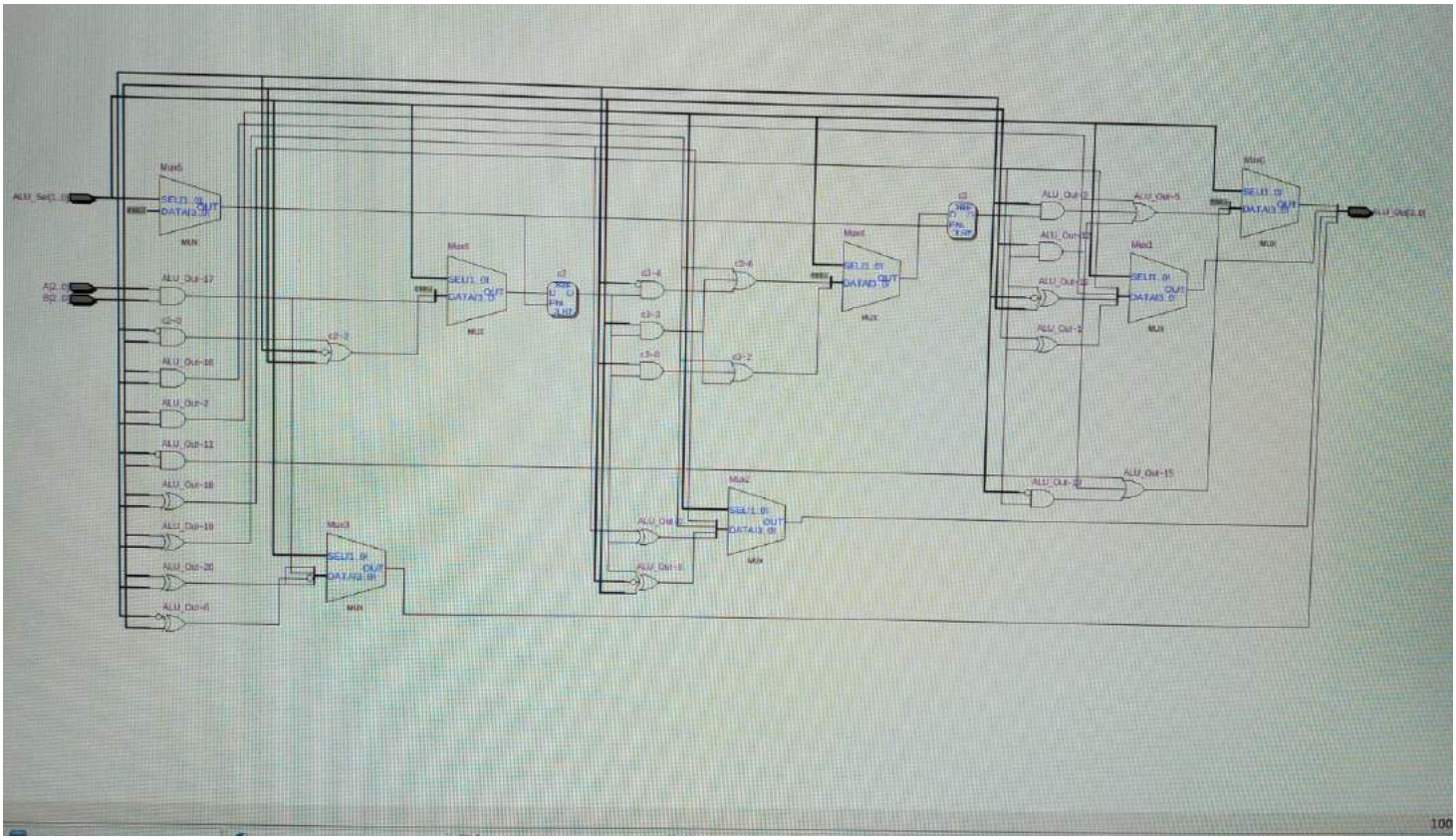
9 Conclusion

We utilized VHDL to simulate and implement 3-bit Adder/Subtractor, 3-bit ALU, and 4:2 Priority Encoder in a systematic and simplified manner.

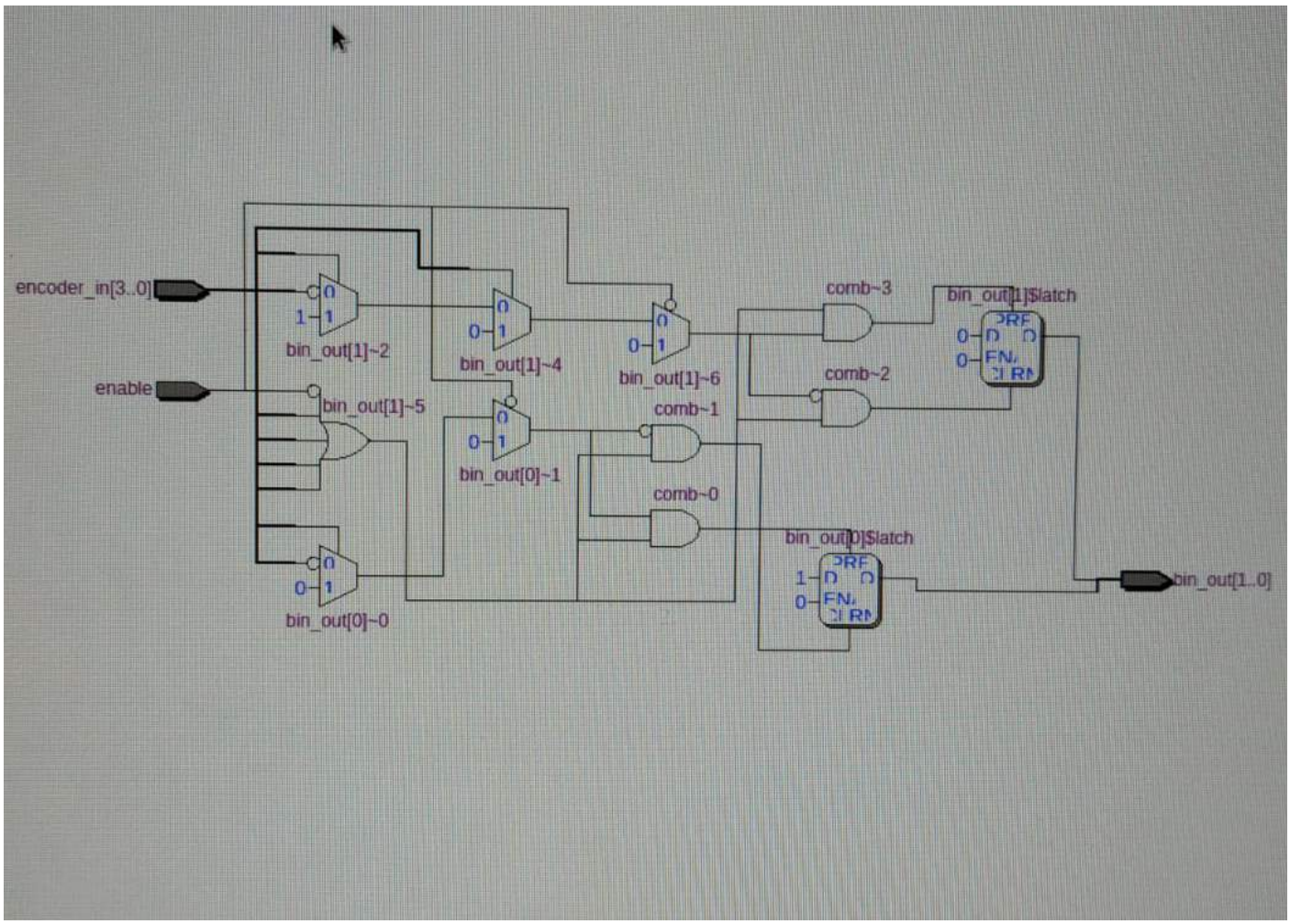
RTL Images:



Problem 1



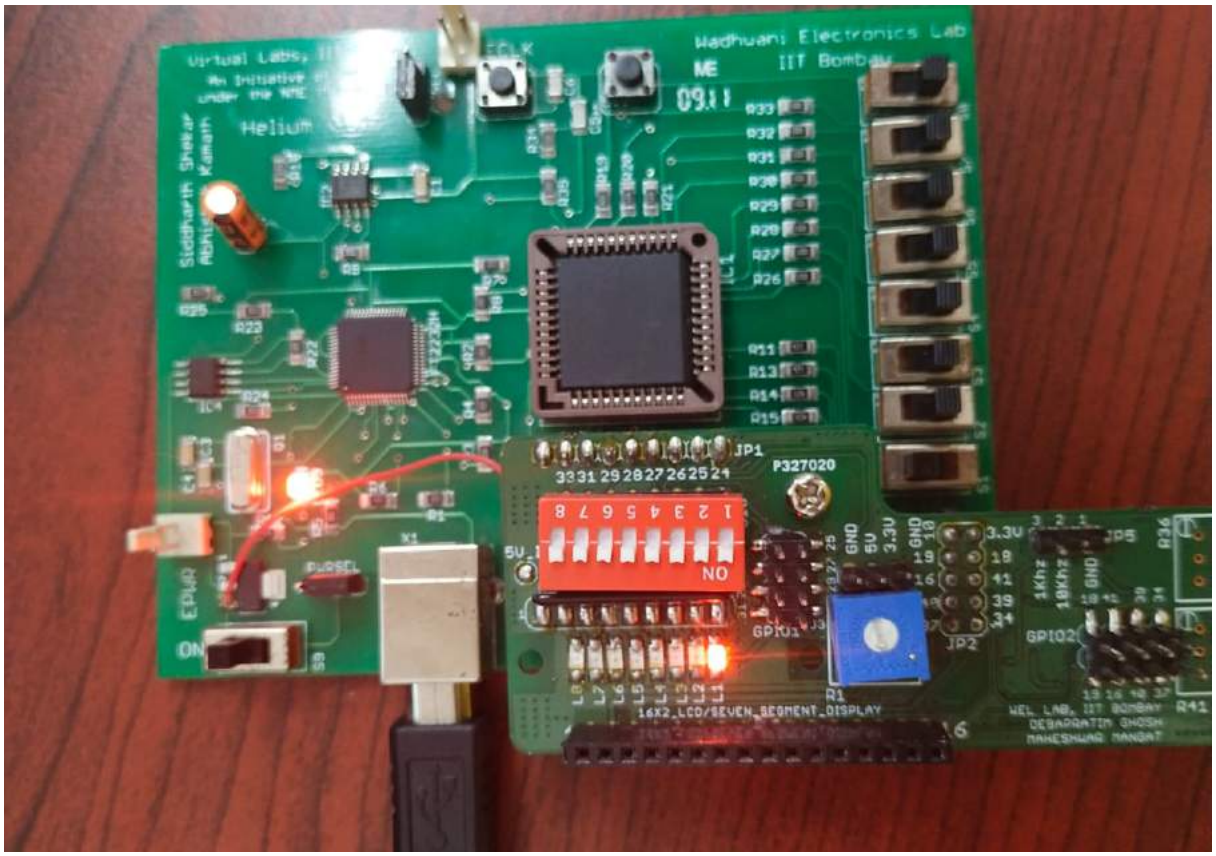
Problem 2

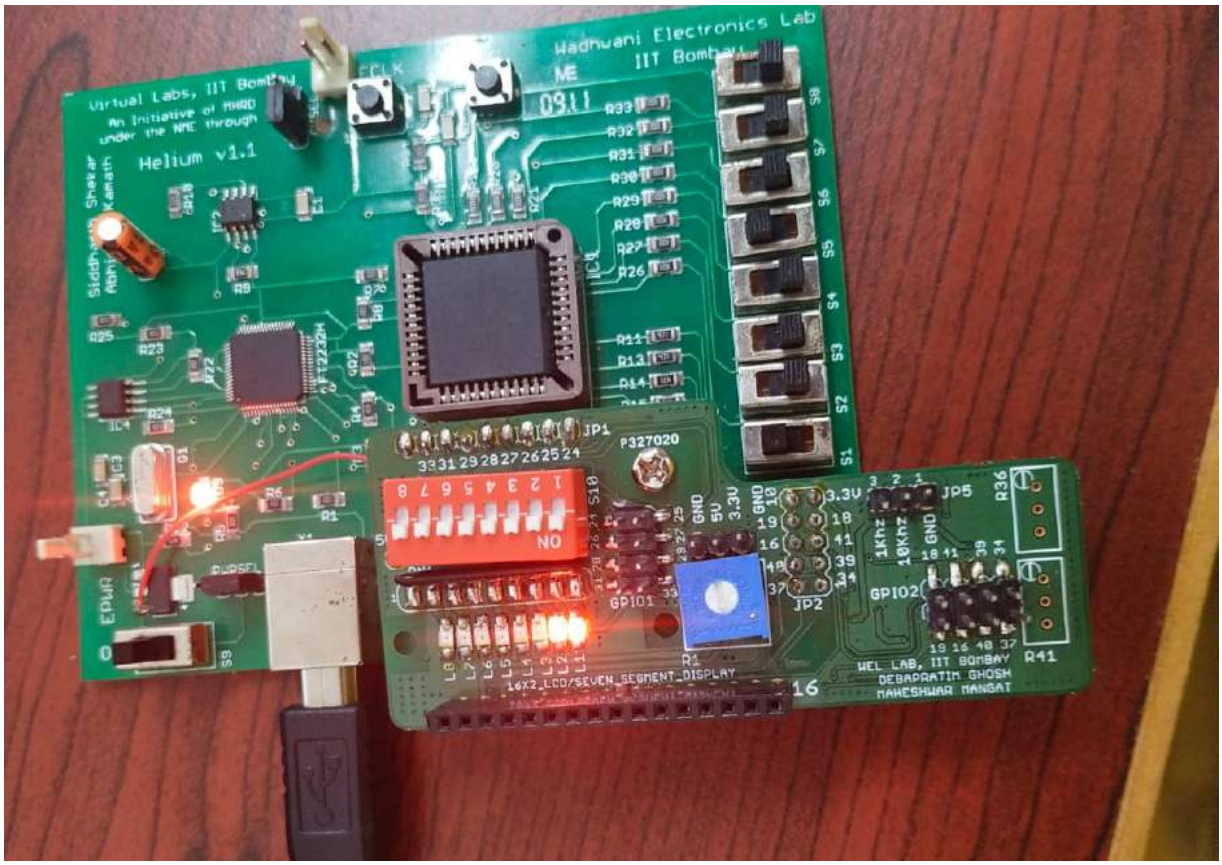

encoder_in[3..0]
enable
bin_out[1]~2
bin_out[1]~4
bin_out[1]~6
bin_out[1]~5
bin_out[0]~1
bin_out[0]~0
comb~3
comb~2
comb~1
comb~0
bin_out[1]$latch
bin_out[0]$latch
bin_out[1..0]

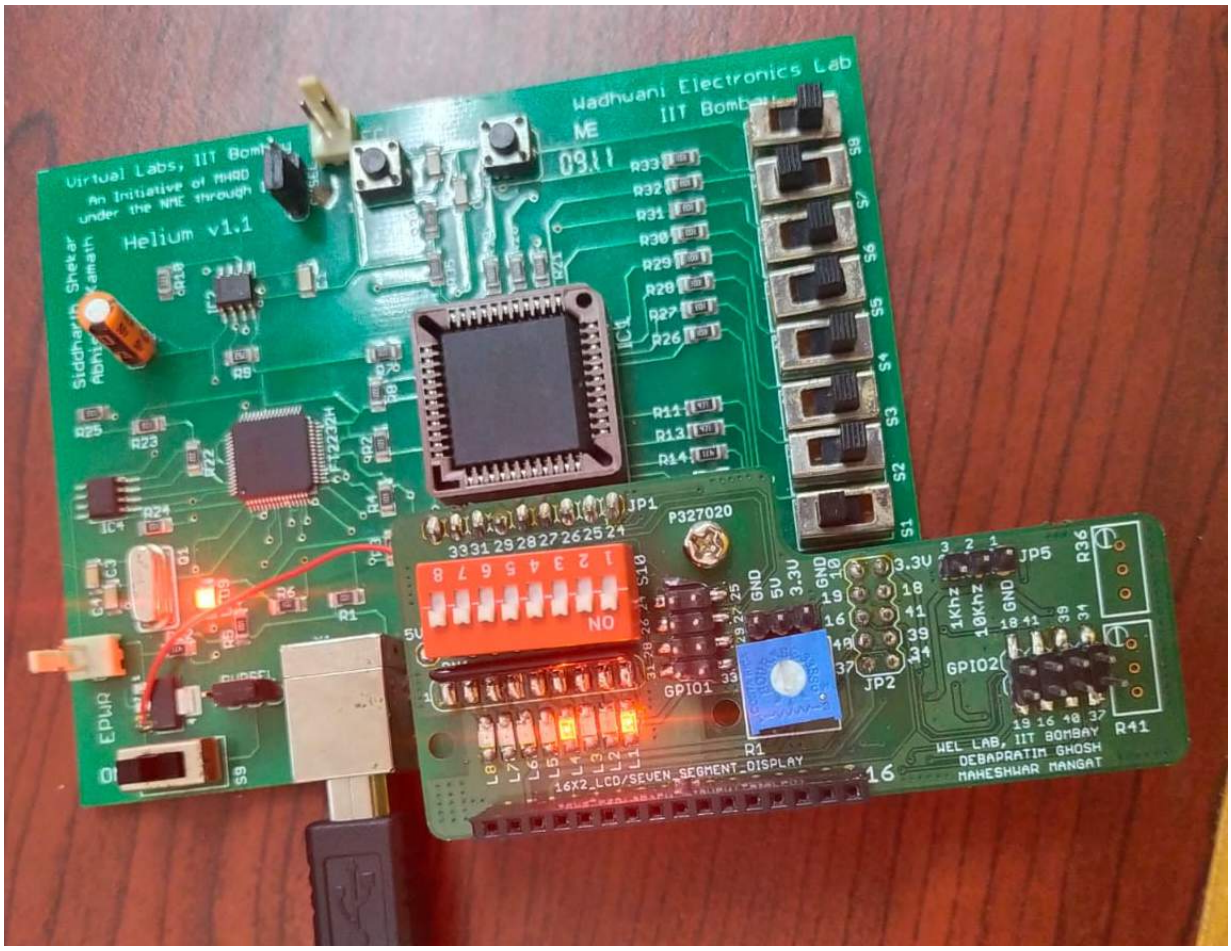
Problem 3

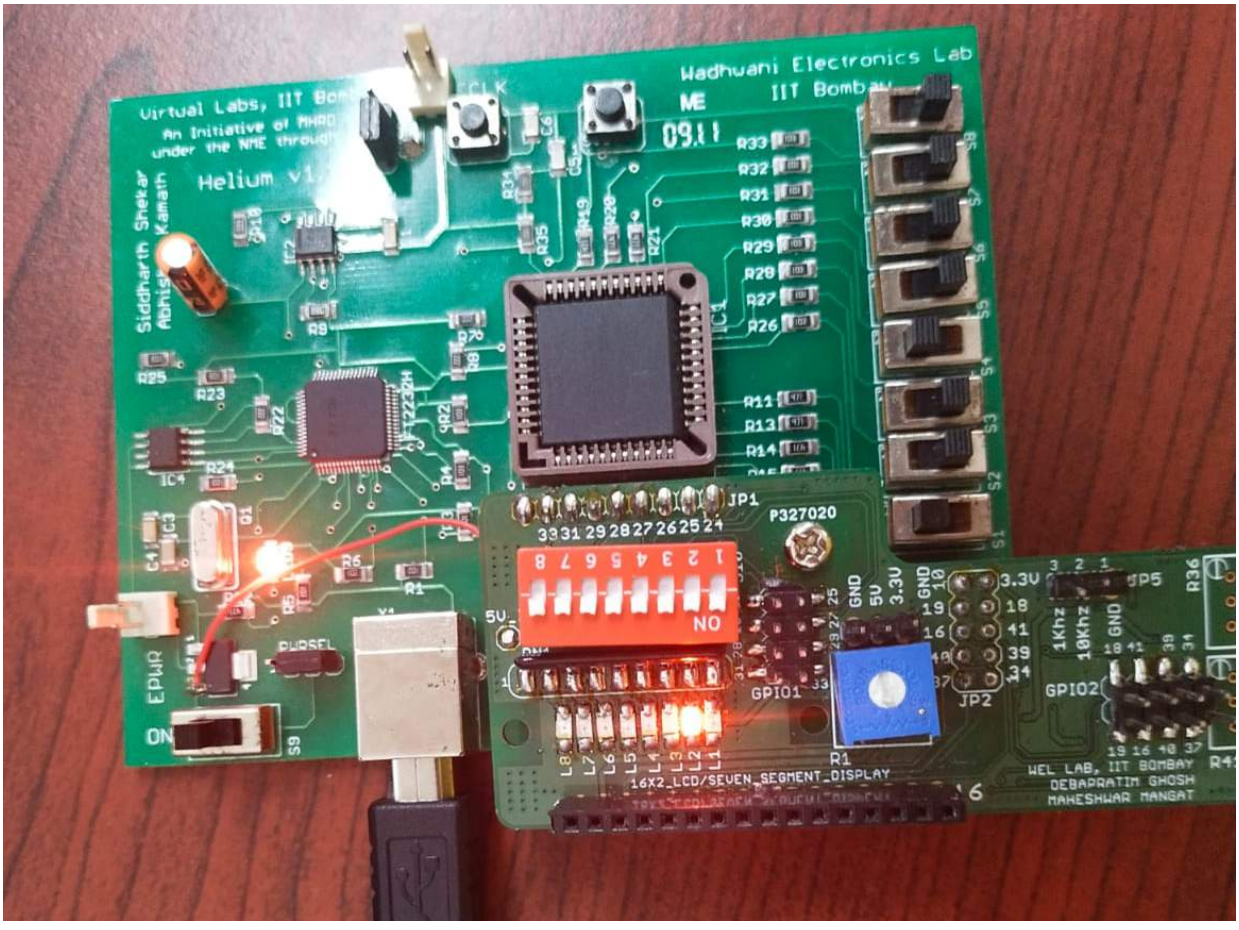
Circuit and Simulation Snapshots :

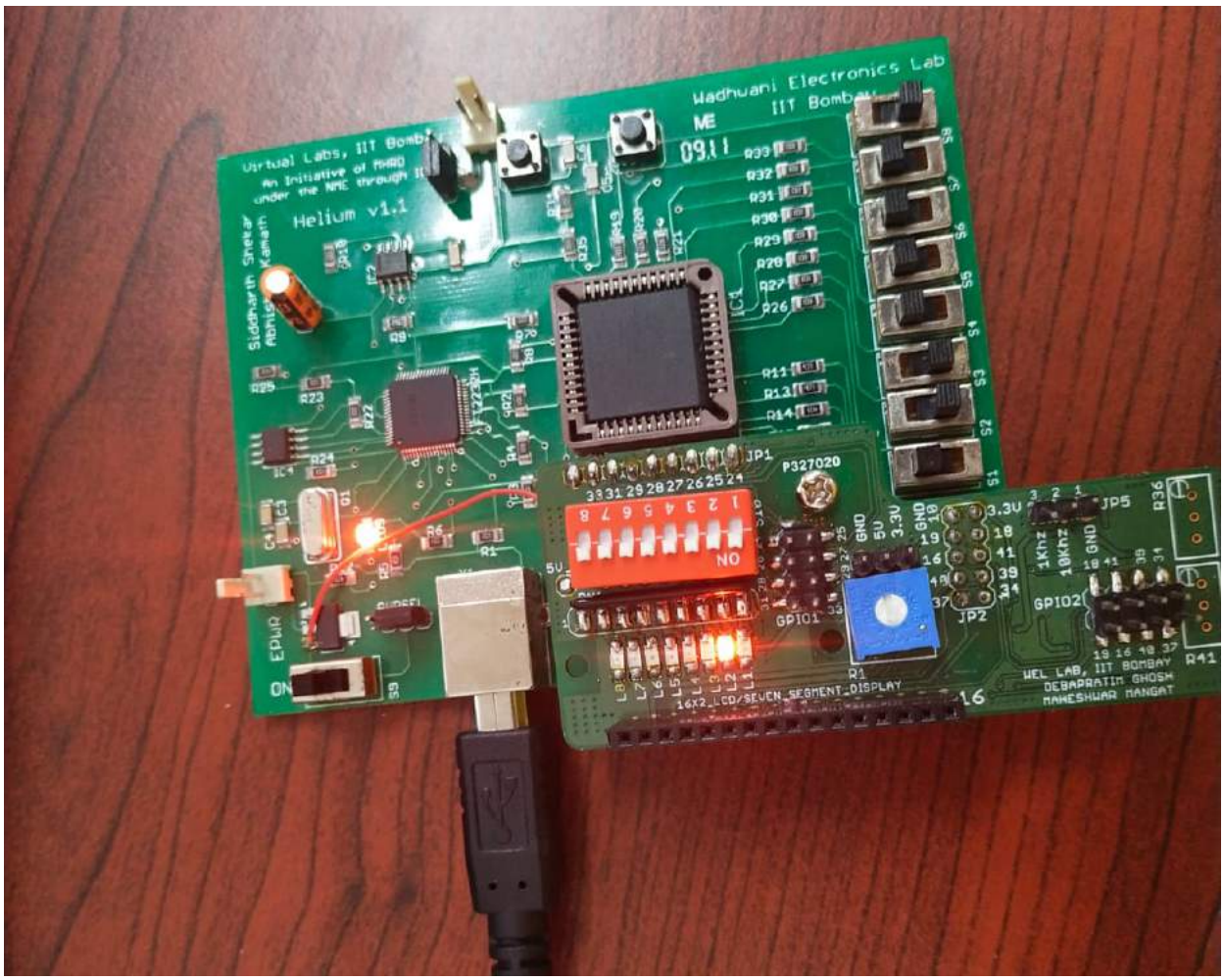
1) 3-bit Adder/subtractor:



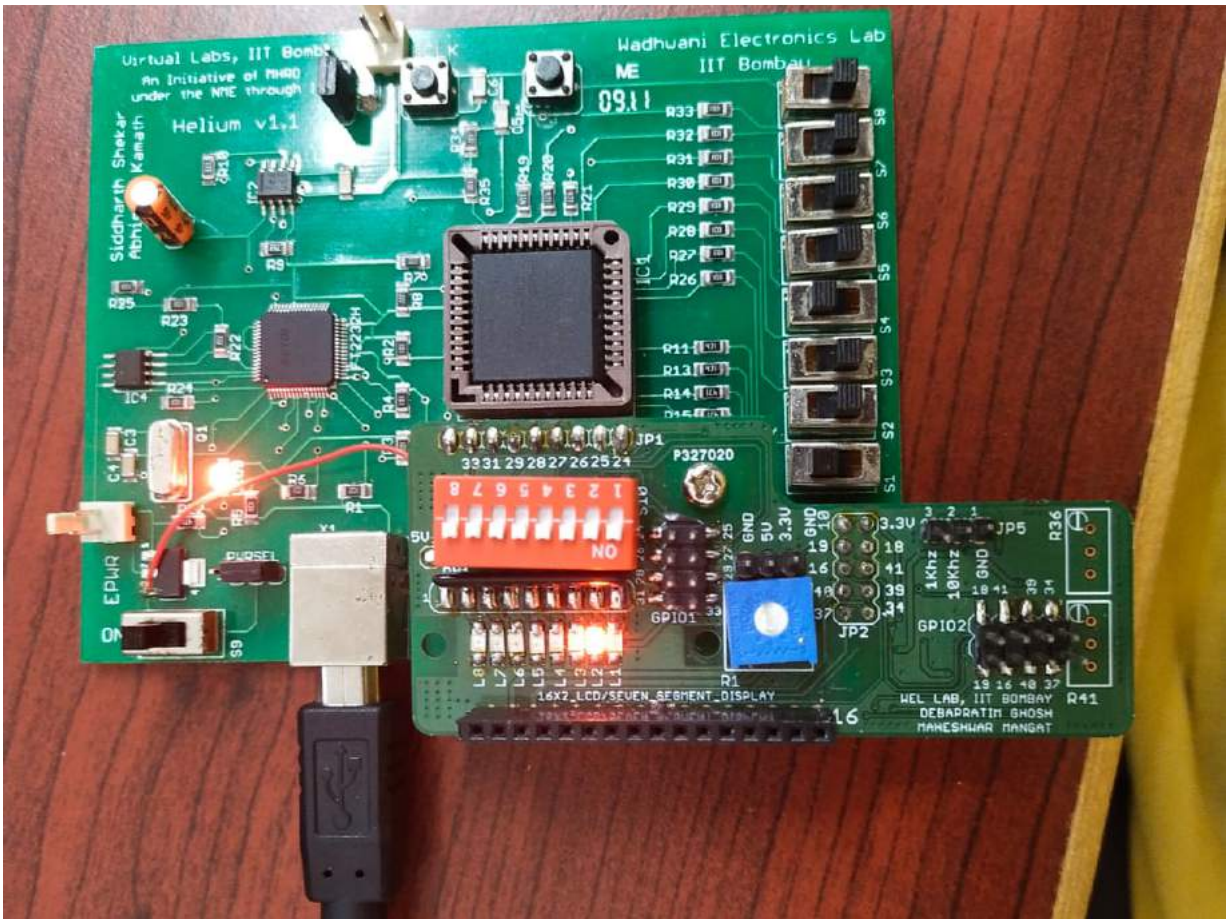


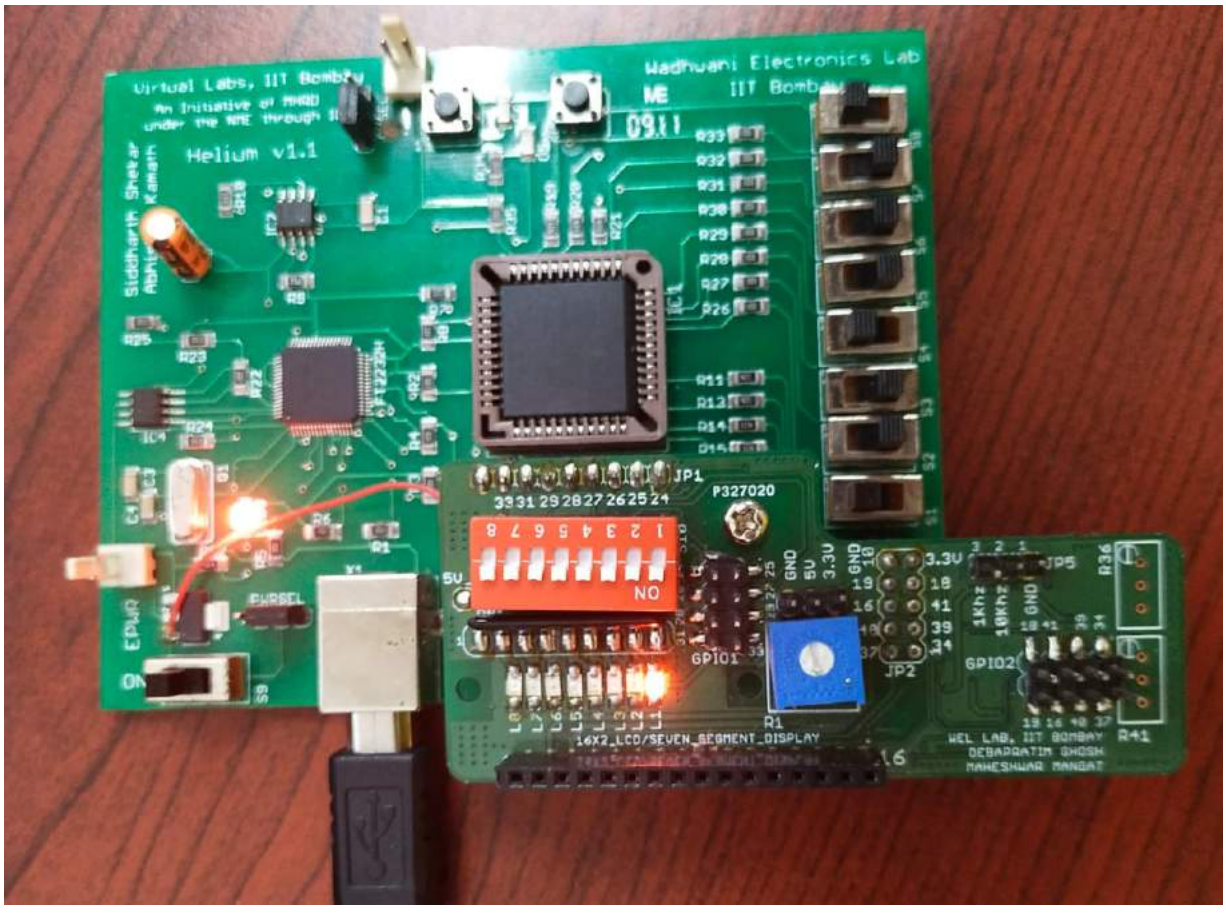

Wadhwani Electronics Lab
IIT Bombay
ME
Virtual Labs, IIT Bombay
An Initiative of MHRD
under the NME through
Helium v1.1
Siddharth Shekar
Abhishek Kamath
P327020
JP1
JP2
JP5
GPIO1
GPIO2
EPWR
PWRSEL
ON
16X2_LCD/SEVEN_SEGMENT_DISPLAY
WEL LAB, IIT BOMBAY
DEBAPRATIM GHOSH
MAHESHWAR MANGAT


Wadhwani Electronics Lab
IIT Bombay
ME
Virtual Labs, IIT Bombay
An Initiative of MHRD
under the NME through
Helium v1.1
Siddharth Shekar
Abhishek Kamath
P327020
JP1
JP2
JP5
GPIO1
GPIO2
EPWR
PWRSEL
ON
16X2_LCD/SEVEN_SEGMENT_DISPLAY
WEL LAB, IIT BOMBAY
DEBAPRATIM GHOSH
MAHESHWAR MANGAT

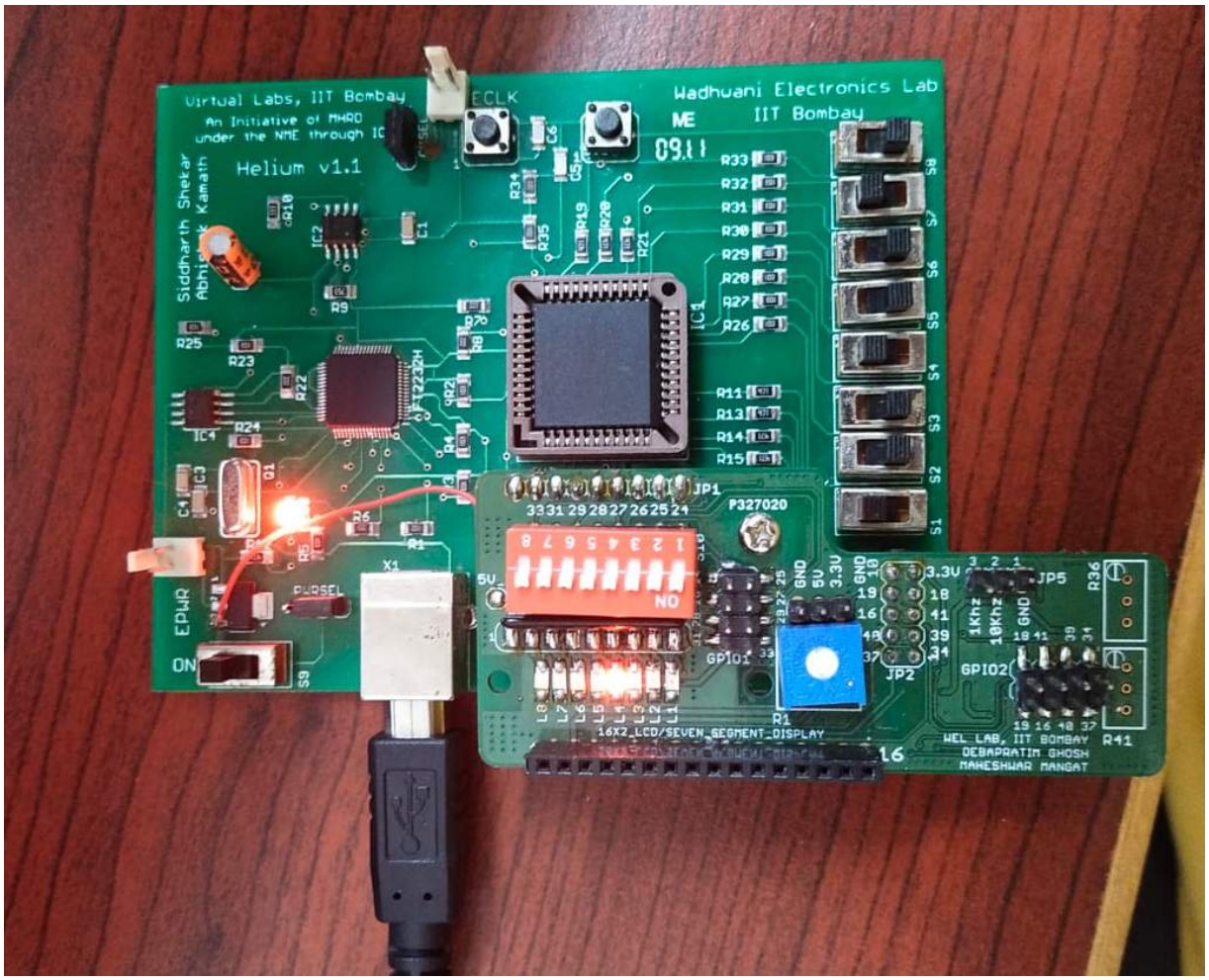

Wadhwani Electronics Lab
IIT Bombay
Virtual Labs, IIT Bombay
An Initiative of MHRD
under the NME through ICT
Helium v1.1
Siddharth Shreyar
Abhijeet Kamath
ME
09.11
JP1
P327020
JP2
JP5
GPIO1
GPIO2
3.3V
5V
GND
1Khz
10Khz
EPWR
ON
16X2_LCD/SEVEN_SEGMENT_DISPLAY
WEL LAB, IIT BOMBAY
DEBAPRATIM GHOSH
MAHESHWAR MANGAT

2) 3-bit ALU:

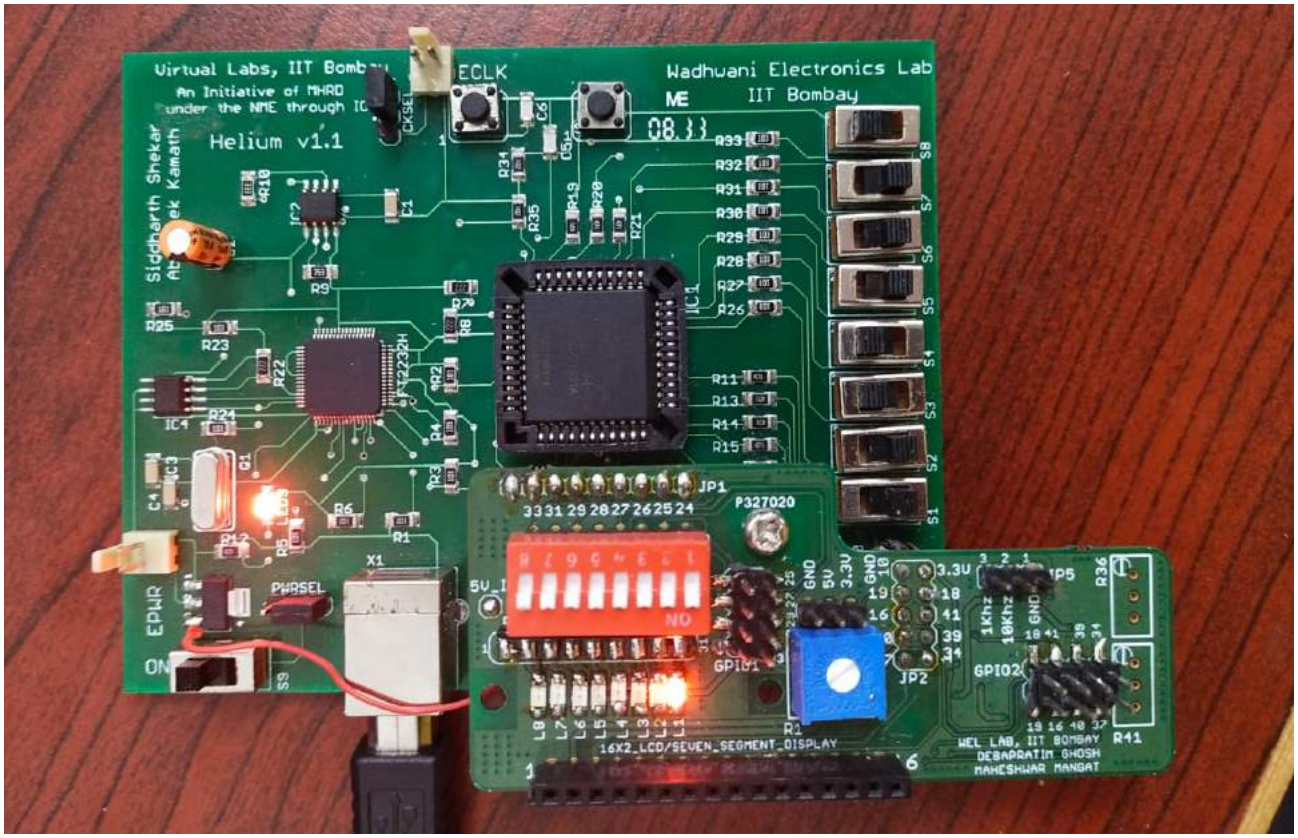

Virtual Labs, IIT Bombay
An Initiative of MHRD under the NME through
Helium v1.1
Siddharth Shekar
Abhi Kamath
Wadhwani Electronics Lab
ME
IIT Bombay
0911
JP1
P327020
PWRSEL
EPWR
GPIO1
GPIO2
JP2
JP5
1Khz
10Khz
GND
3.3V
5V
R1
16X2_LCD/SEVEN_SEGMENT_DISPLAY
WEL LAB, IIT BOMBAY
DEBAPRATIM GHOSH
MAHESHWAR MANGAT
R41
R36

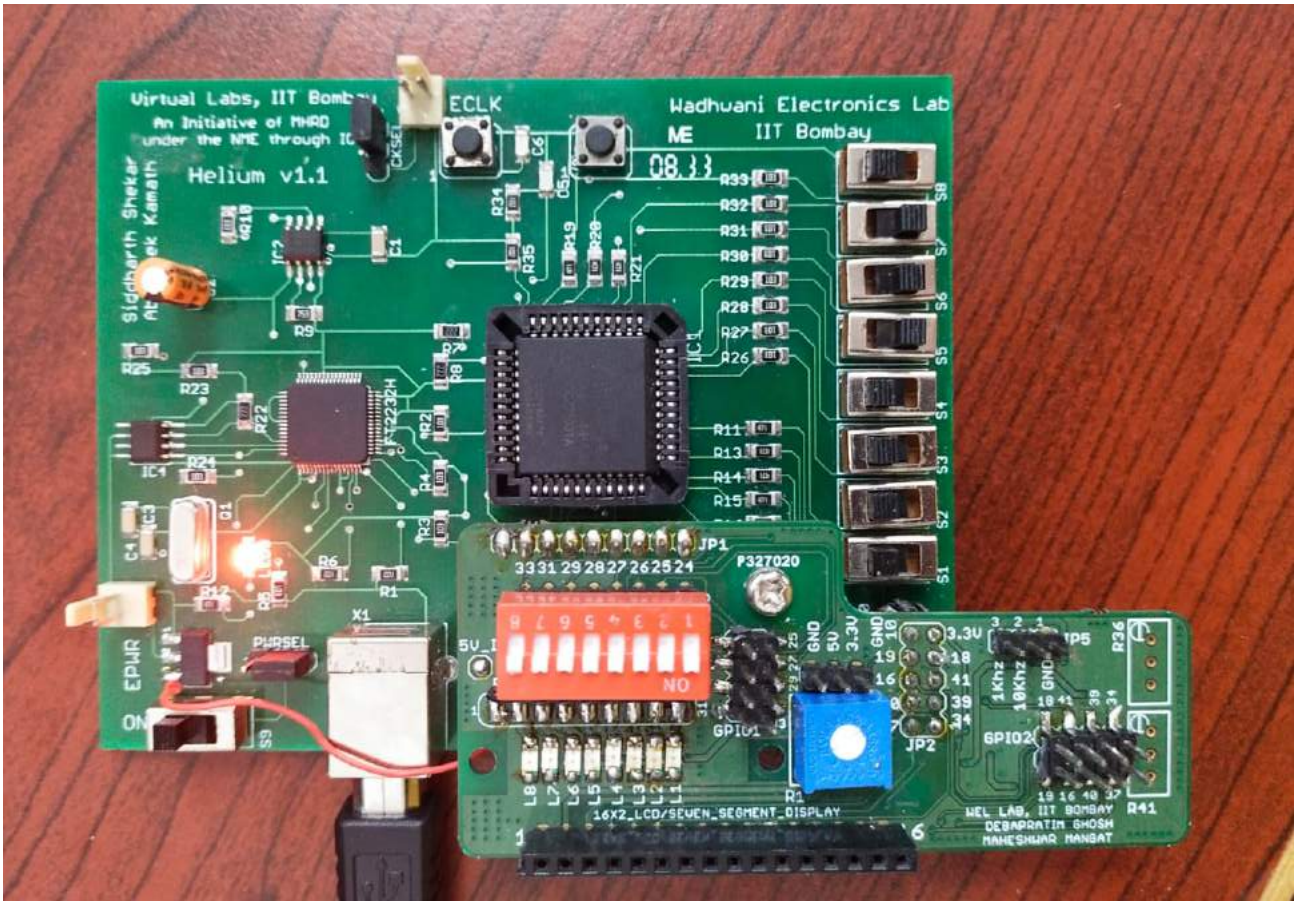

Virtual Labs, IIT Bombay
Helium v1.1
Wadhwani Electronics Lab
ME
IIT Bombay
0911
JP1
P327020
PWRSEL
EPWR
GPIO1
GPIO2
JP2
JP5
1Khz
10Khz
GND
3.3V
5V
R1
16X2_LCD/SEVEN_SEGMENT_DISPLAY
WEL LAB, IIT BOMBAY
DEBAPRATIM GHOSH
MAHESHWAR MANGAT
R41
R36

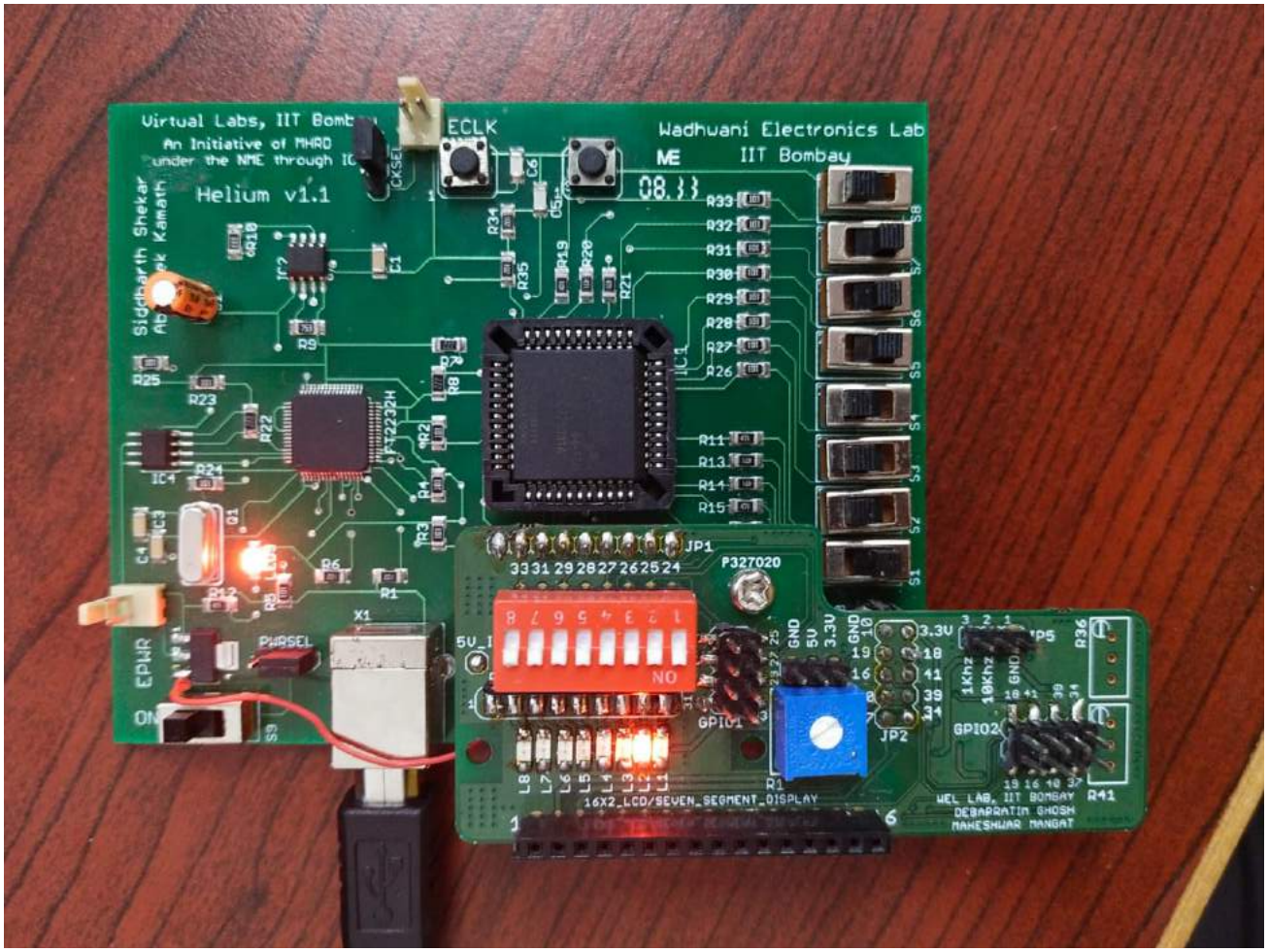

Virtual Labs, IIT Bombay
Wadhwani Electronics Lab
IIT Bombay
Helium v1.1
Siddharth Shekar
Abhi
Kamath
09.11
P327020
JP1
JP2
JP5
R36
R41
GPIO1
GPIO2
PWRSEL
EPWR
ON
NO
1Khz
10Khz
16X2_LCD/SEVEN_SEGMENT_DISPLAY
WEL LAB, IIT BOMBAY
DEBAPRATIM GHOSH
MAHESHWAR MANGAT

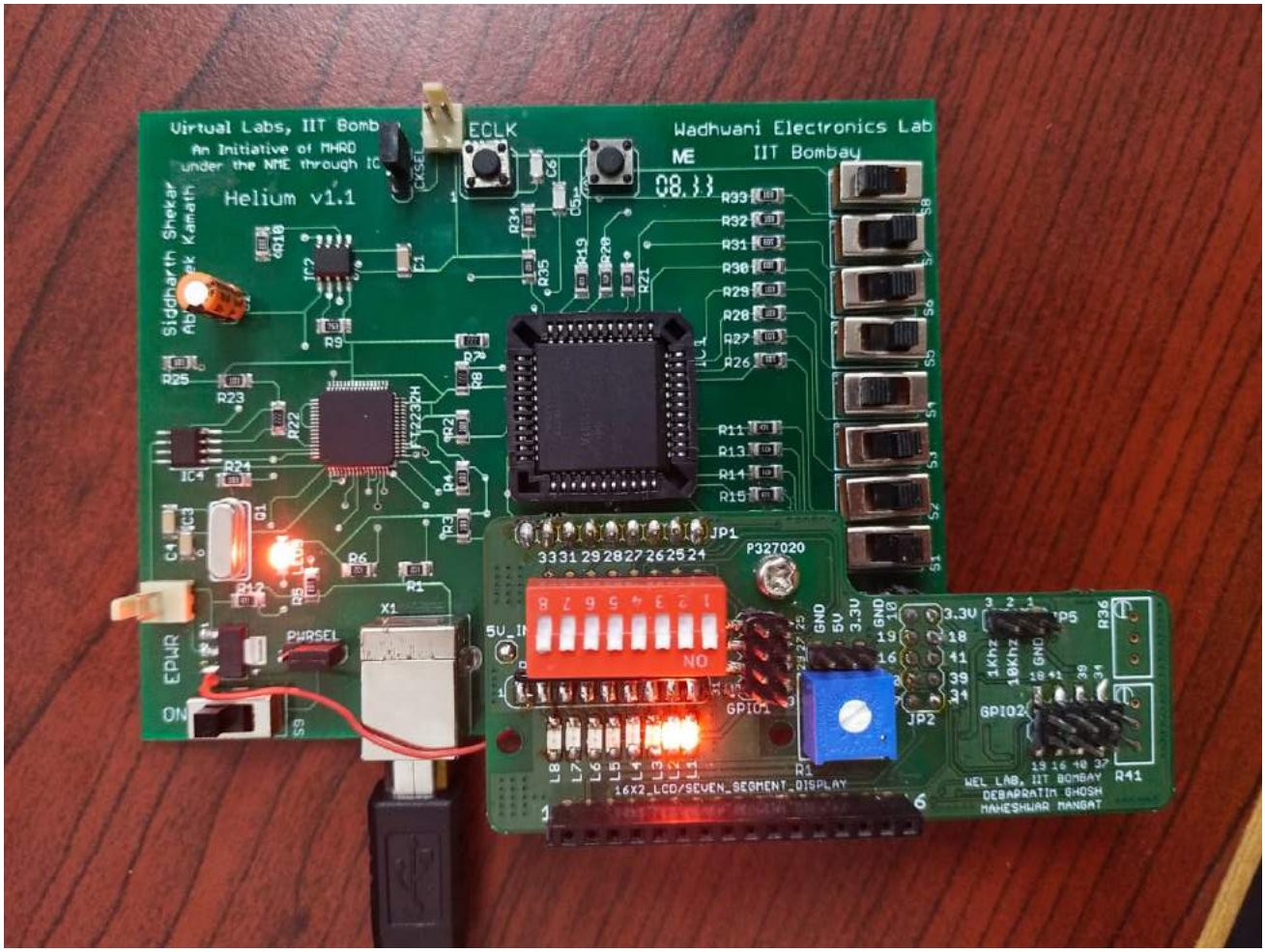

Virtual Labs, IIT Bombay
ECLK
Wadhwani Electronics Lab
IIT Bombay
Helium v1.1
Siddharth Shekar
Kamath
09.11
P327020
JP1
JP2
JP5
R36
R41
GPIO1
GPIO2
PWRSEL
EPWR
ON
NO
1Khz
10Khz
16X2_LCD/SEVEN_SEGMENT_DISPLAY
WEL LAB, IIT BOMBAY
DEBAPRATIM GHOSH
MAHESHWAR MANGAT

3) 4:2 Priority Encoder:






Virtual Labs, IIT Bomb
An Initiative of MHRD
under the NME through IC
Helium v1.1
Siddharth Shekar
Wadhwani Electronics Lab
ME
IIT Bombay
ECLK
CKSEL
08.33
R33
R32
R31
R30
R29
R28
R27
R26
R11
R13
R14
R15
IC1
IC2
IC3
IC4
FT232RH
JP1
P327020
3331 29 28 27 26 25 24
NO
GPIO1
GPIO2
JP2
JP5
3.3V
5V
GND
1Khz
10Khz
R36
R41
PWRSEL
EPWR
ON
X1
16X2_LCD/SEVEN_SEGMENT_DISPLAY
WEL LAB, IIT BOMBAY
DEBAPRATIM GHOSH
MAHESHWAR MANGAT


Virtual Labs, IIT Bomb
An Initiative of MHRD
under the NME through IC
Helium v1.1
Siddharth Shekar
Wadhwani Electronics Lab
ME
IIT Bombay
ECLK
CKSEL
08.33
R33
R32
R31
R30
R29
R28
R27
R26
R11
R13
R14
R15
IC1
IC2
IC3
IC4
FT232RH
JP1
P327020
3331 29 28 27 26 25 24
NO
GPIO1
GPIO2
JP2
JP5
3.3V
5V
GND
1Khz
10Khz
R36
R41
PWRSEL
EPWR
ON
X1
16X2_LCD/SEVEN_SEGMENT_DISPLAY
WEL LAB, IIT BOMBAY
DEBAPRATIM GHOSH
MAHESHWAR MANGAT