**DIGITAL CIRCUITS LABORATORY**

**Experiment-7**

**Full Adder using VHDL and CPLD Board**

Tanish H Talapaneni

200020050

**Aim:**To implement the Full Adder by writing the code using VHDL and implementing it on the CPLD Board

**Summary:** Implementation of the Full Adder on CPLD board by wiring up the circuit and executing the 3 types of VHDL code

**Components used:**CPLD Board, PC

**Procedure:**

* Configure the Quartus Software and open a new project
* Writing the code for Full Adder in VHDL
* Compiling the code and correcting all errors
* Viewing the RTL schematic of the written VHDL Code
* Creating .svf file and planning the pins for outputs and inputs, in order to execute on the CPLD Board
* Running the code on the CPLD Board by executing it via the Terminal Window on the PC

**VHDL Code:**

i)VHDL Code for Full-Adder in Dataflow modeling style:

-- Data flow modeling:

library ieee;

use ieee.std\_logic\_1164.all;

-- Data Flow Modelling Style

entity full\_adder\_dataflow is

port(a, b, cin: in bit;

sum, cout: out bit);

end full\_adder;

architecture dataflow of full\_adder\_dataflow is

begin sum <= a xor b xor cin;

cout<= (a and b) or ((a xor b) and cin);

end dataflow;

ii)VHDL Code for Full-Adder in Behavioral Modeling Style:

-- Behavioral modeling Style:

library ieee;

use ieee.std\_logic\_1164.all;

entity full\_adder\_behaviroal is

port (a, b, cin: in std\_logic; s, cout: out std\_logic);

end full\_adderb;

architecture behavior of full\_adder\_behaviroal is

Begin

c1: process (a,b,cin) Begin

if a = ‘0’ and b = ’0’ then

s <= cin;

cout <= 0;

if a = ‘0’ and b = ‘1’ then

s <= not cin;

c <= cin;

if a = ‘1’ and b = ‘0’ then

s <= not cin;

cout <= cin;

if a = ‘1’ and b = ‘1’ then

s <= cin;

cout <= 1;

end if;

end process c1;

end behavior;

iii)VHDL Code for Full-Adder in Structural Modeling Style:

-- structural Full Adder

library ieee;

use ieee.std\_logic\_1164.all;

entity FAS is

port (A, B,Cin: in std\_logic;

S, Cout: out std\_logic);

end FAS;

architecture structure of FAS is

signal o1, o2, o3, o4, i1, i2, i3, i4, i5,i6,i7,i8,i9,i10,i11: std\_logic;

-- gates description

component xor\_gate\_3

port (i1, i2,i3: in std\_logic;

o1: out std\_logic);

end component;

component xor\_gate\_2

port (i4,i5: in std\_logic; C

o2: out std\_logic);

end component;

component and\_gate\_1

port (i6,i7: in std\_logic;

o3: out std\_logic);

end component;C

component and\_gate\_2

port (i8, i9: in std\_logic;

o4: out std\_logic);

end component;

component or\_gate

port (i10, i11: in std\_logic;

o5: out std\_logic);

end component;

begin

u1: xor\_gate\_3 port map (i1 => A, i2 => B, i3 => Cin, o1 => i4);

u2: xor\_gate\_2 port map (i4 => o1, i5 => Cin, o2 => S);

u3: and\_gate\_1 port map (i6 => o1, i7 => Cin, o3 => i10);

u4: and\_gate\_2 port map (i8 => A, i9 => B, o4 => i11);

u5: or\_gate port map (i10 => o3, i11 => o4, o5 => Cout);

end structure;

library ieee;C

use ieee.std\_logic\_1164.all;

entity xor\_gate\_2 is

port (i4, i5: in std\_logic;

o2: out std\_logic);

end xor\_gate\_2;

architecture dataflow of xor\_gate\_2 is

BeginC

o2 <= i4 xor i5;

end dataflow;

library ieee;

use ieee.std\_logic\_1164.all;

entity xor\_gate\_3 is

port (i1, i2,i3: in std\_logic;

o1: out std\_logic);

end xor\_gate\_3;

architecture dataflow of xor\_gate\_3 is

Begin

o1 <= ((i1 xor i2) xor i3);

end dataflow;

library ieee;

use ieee.std\_logic\_1164.all;

entity and\_gate\_1 is

port (i6, i7: in std\_logic;

o3: out std\_logic);

end and\_gate\_1;

architecture dataflow of and\_gate\_1 is

Begin

o3 <= i6 and i7;

end dataflow;

library ieee;

use ieee.std\_logic\_1164.all;

entity and\_gate\_2 is

port (i8, i9: in std\_logic;

o4: out std\_logic);

end and\_gate\_2;

architecture dataflow of and\_gate\_2 is

Begin

o4 <= i8 and i9;

end dataflow;

library ieee;

use ieee.std\_logic\_1164.all;

entity or\_gate is

port (i10, i11: in std\_logic;

o5: out std\_logic);

end or\_gate;

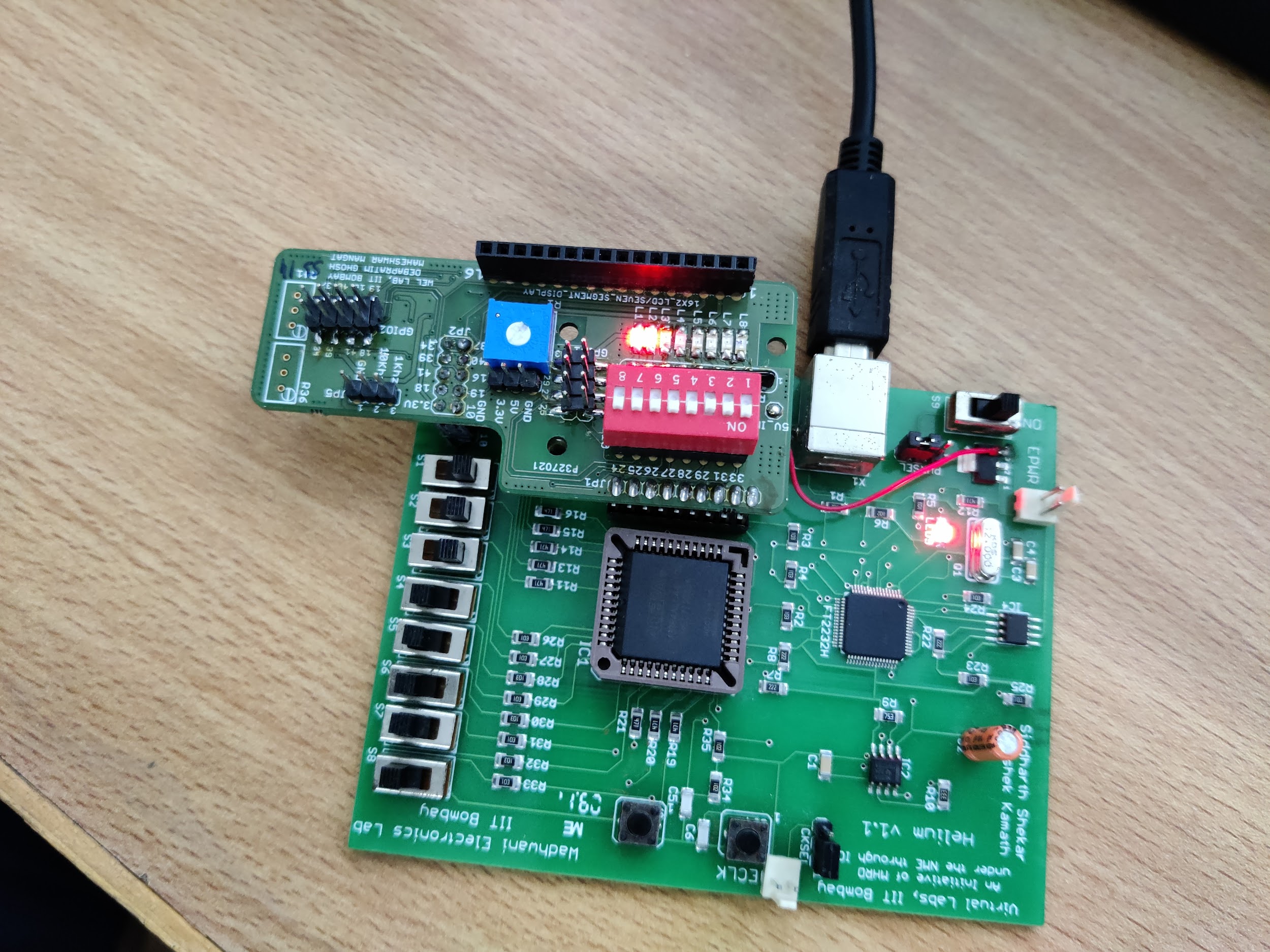
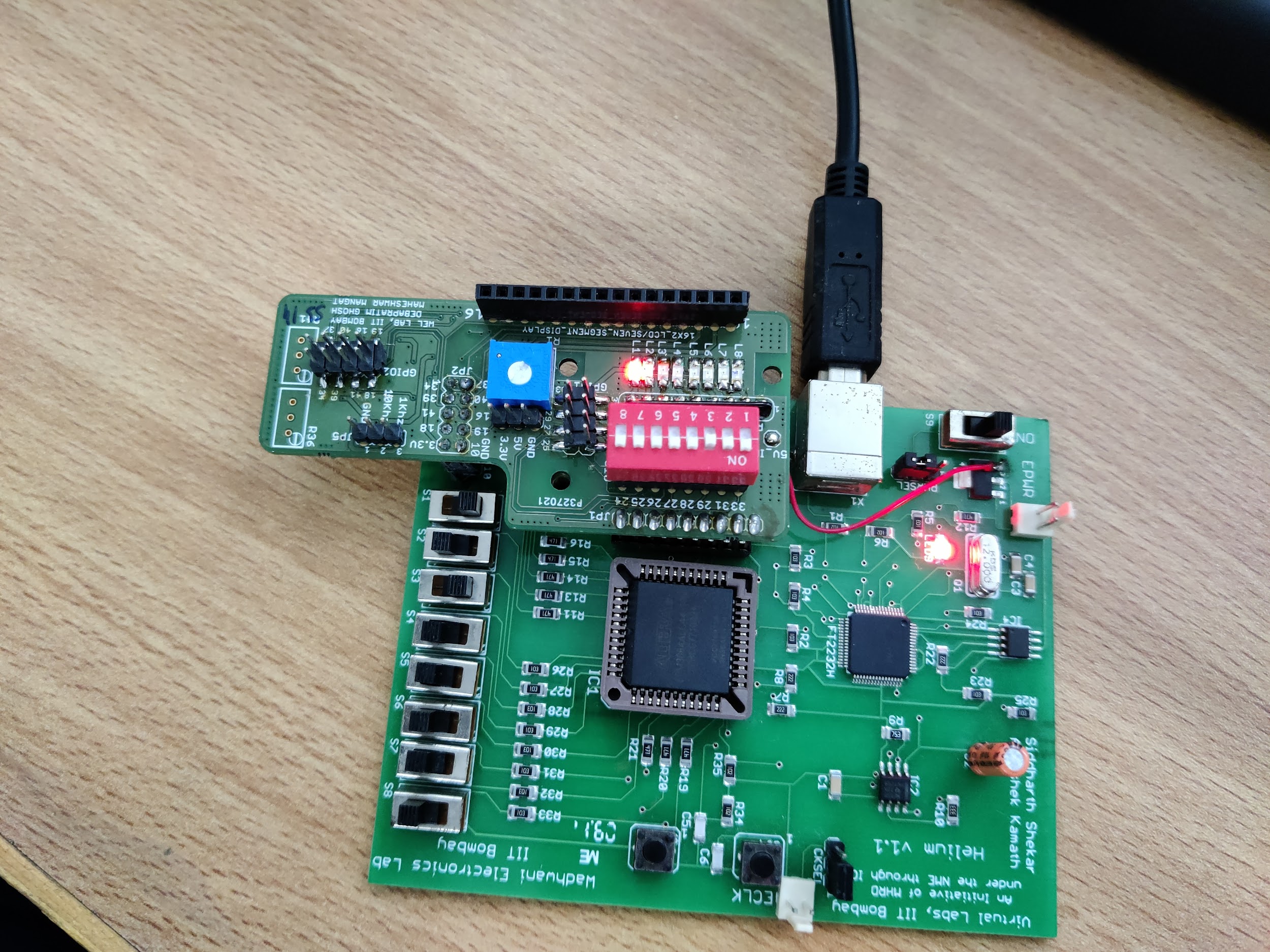
architecture dataflow of or\_gate is

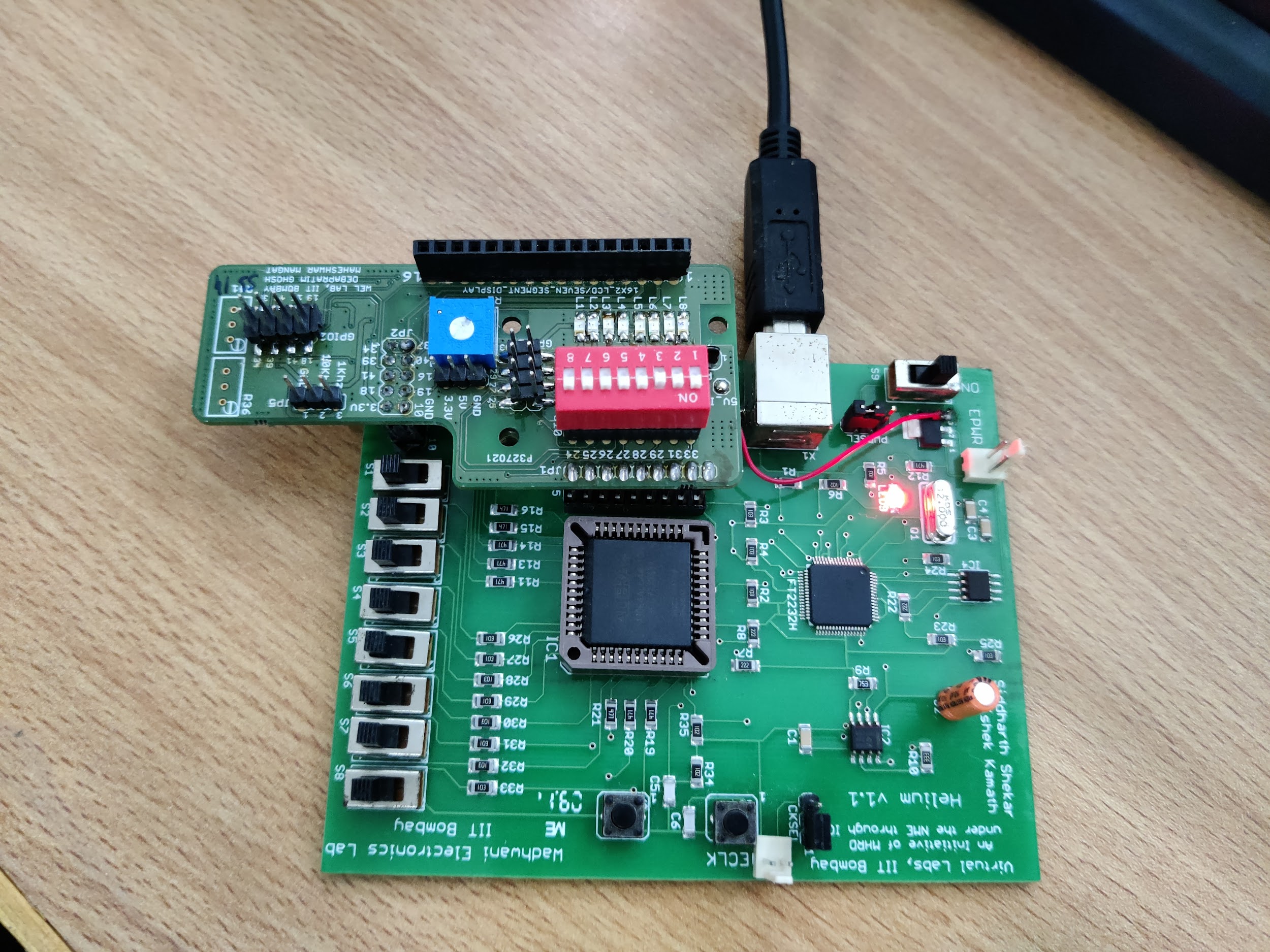
Begin

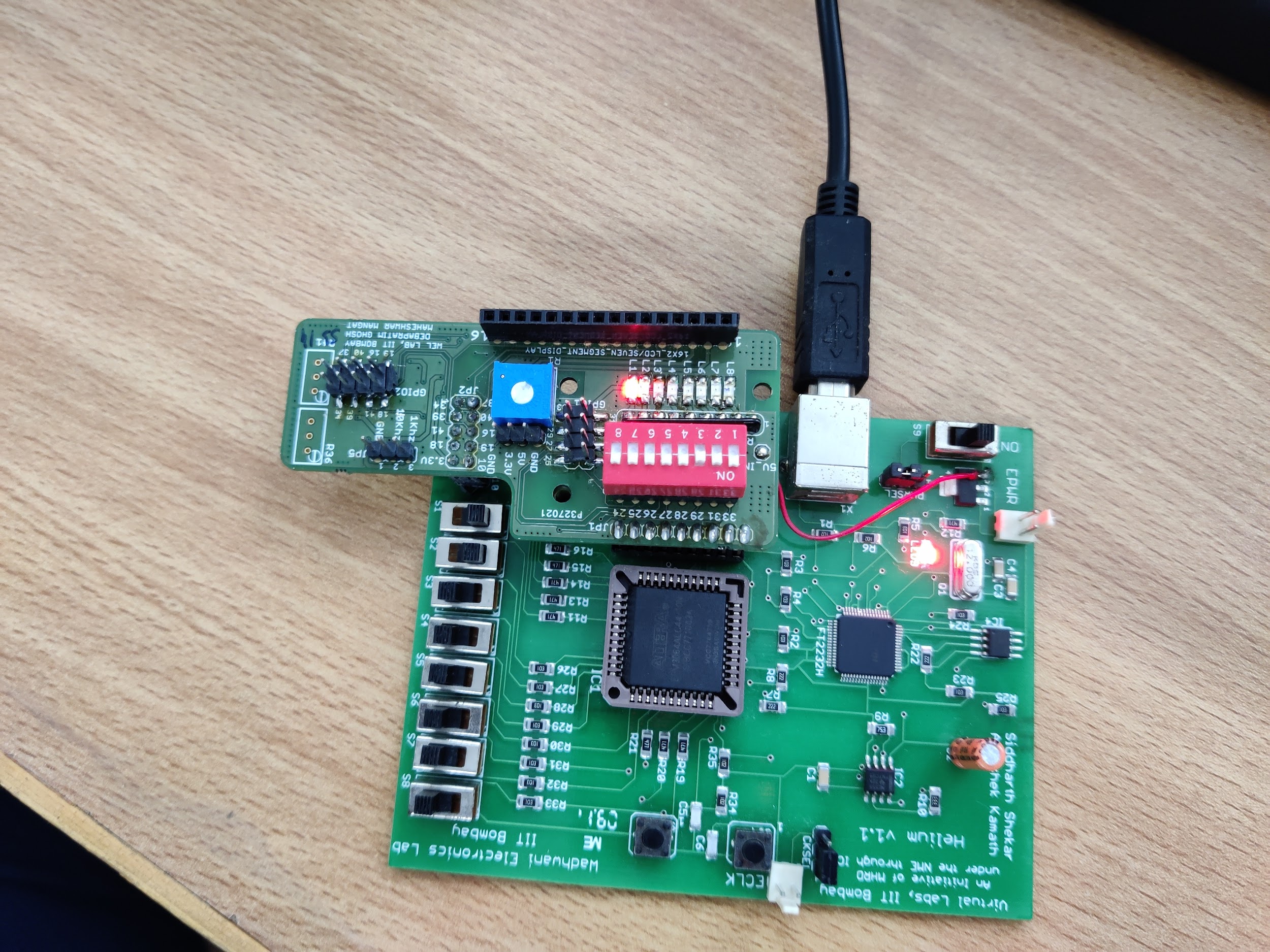
o5 <= i10 or i11;

end dataflow;

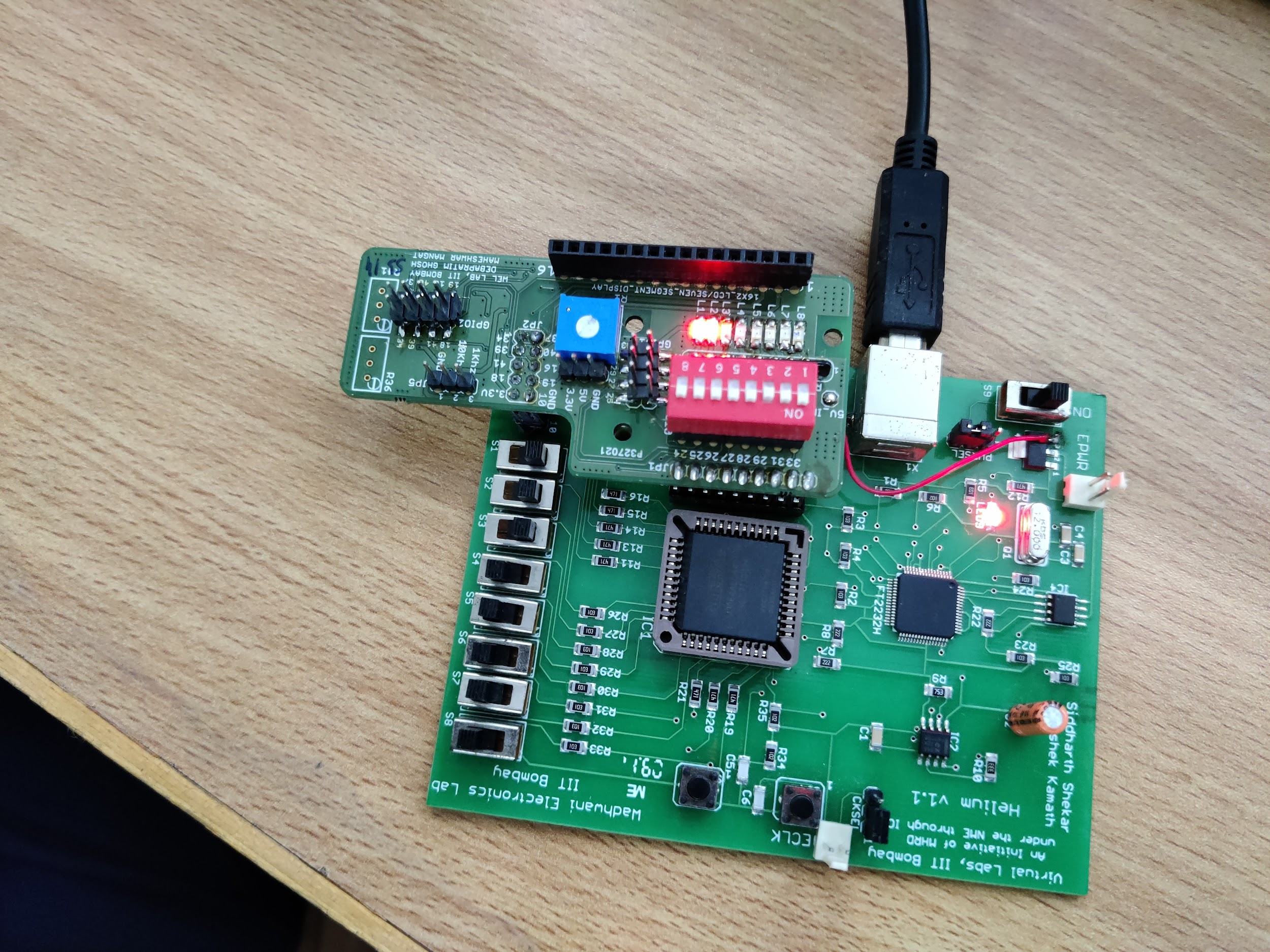
**Hardware:**











**Results and Discussions:** We finally understood how to program in VHDL and how to use the CPLD Board and execute the working of many devices which we will learn in the future. Initially, it was tricky to configure the software and write the codes, as it was our first time, but slowly we learnt and corrected all of the errors, and we were finally able to see the output of the full adder on the CPLD Board. We coded the working of the Full Adder in 3 modes: Data Flow, Behavioral and Structural, and it was nice to analyze the differences observed in the working of the three codes. Overall, it was a great experience working with VHDL and CPLD.

**Conclusion:**

Execution of Full-Adder Logic on the CPLD Board using VHDL Programming Language. Three types of logic were used: Data Flow, Behavioral and Structural. Data Flow Logic is based on the type of input(0 or 1). Behavioral Logic is based on if-else statements applied on the inputs. Structural Logic is based on designing the structure of each logic gate involved. The output observed is the same for all three types of design, as all the designs encode the same logic of the Full Adder.