COL-215

Hardware Assignment 1

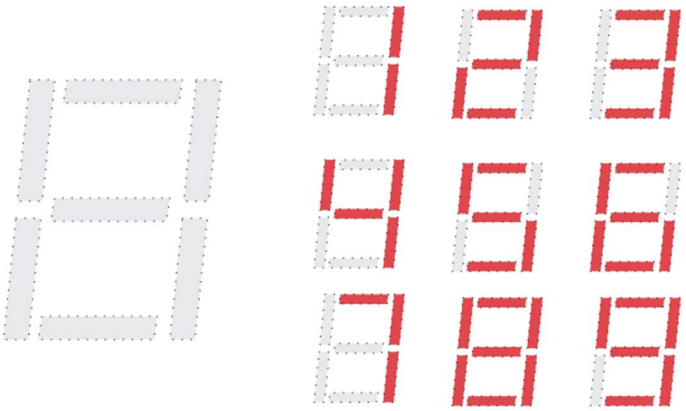
Poojan Shah 2022CS11594 *Hemanshu Garg* 2022CS11090

OBJECTIVE

To design and implement a combinational circuit that takes a 4-digit decimal/hexadecimal number from switches in the Basys3 board and displays it on the seven-segment displays on the board.

THE SEVEN SEGMENT DISPLAY

The display on the basys3 board is implemented using a common anode configuration. The illumination pattern depends upon the biasing of the transistors used. To illuminate a segment, the anode should be driven high while the cathode is driven low. The anodes are labelled **AN00** *to* **AN03** *and the cathodes are labelled* **CA** *to* **CG**. *However, since the basys3 uses pnp transistors to drive enough current into the common anode point, the anode enables are inverted. Therefore, both the* **AN0..3** *and the* **CA..G** *signals are driven low when active. The figure below shows the various illumination patterns.*

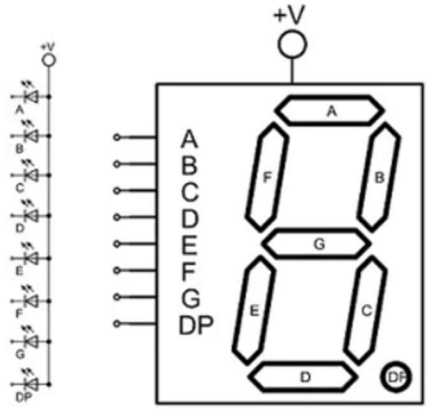


FORMULATING THE SOLUTION

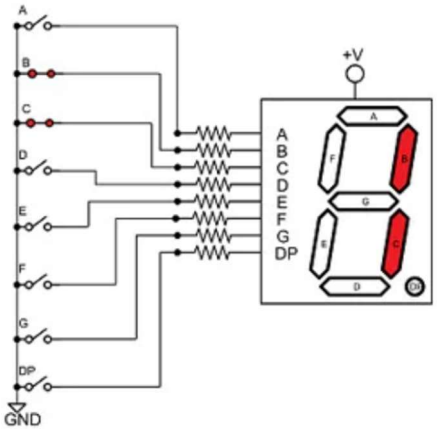
The main task is to get an input of a number through the switches and get an output on the display. The input is taken as a 4-bit binary number to cover digits from 0 to 15 in hexadecimal. The number is represented as:

$$(abcd)_2 = 8a + 4b + 2c + d$$

Here, a,b,c and d are Boolean variables. The output is a set of seven functions of the four input bits as shown in the figure below:



As an example, let us say that we want to display one, whose binary representation leads to the input a=b=c=0 and d=1. The corresponding output variables are **CB** = **CC** = *0 , and the rest equal to 1, as shown in the following figure:85*



Following similar lines of reasoning, we can develop a truth table as shown:

Input				***Output***							***Display***
A	**b**	**c**	**d**	**ca**	**cb**	**cc**	**cd**	**ce**	**cf**	**cg**	
0	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*1*	***0***
0	*0*	*0*	*1*	*1*	*0*	*0*	*1*	*1*	*1*	*1*	***1***
0	*0*	*1*	*0*	*0*	*0*	*1*	*0*	*0*	*1*	*0*	***2***
0	*0*	*1*	*1*	*0*	*0*	*0*	*0*	*1*	*1*	*0*	***3***
0	*1*	*0*	*0*	*1*	*0*	*0*	*1*	*1*	*0*	*0*	***4***
0	*1*	*0*	*1*	*0*	*1*	*0*	*0*	*1*	*0*	*0*	***5***
0	*1*	*1*	*0*	*0*	*1*	*0*	*0*	*0*	*0*	*0*	***6***
0	*1*	*1*	*1*	*0*	*0*	*0*	*1*	*1*	*1*	*1*	***7***
1	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	***8***
1	*0*	*0*	*1*	*0*	*0*	*0*	*0*	*1*	*0*	*0*	***9***
1	*0*	*1*	*0*	*0*	*0*	*0*	*1*	*0*	*0*	*0*	***A***
1	*0*	*1*	*1*	*1*	*1*	*0*	*0*	*0*	*0*	*0*	***B***
1	*1*	*0*	*0*	*0*	*1*	*1*	*0*	*0*	*0*	*1*	***C***
1	*1*	*0*	*1*	*1*	*0*	*0*	*0*	*0*	*1*	*0*	***d***
1	*1*	*1*	*0*	*0*	*1*	*1*	*0*	*0*	*0*	*0*	***E***
1	*1*	*1*	*1*	*0*	*1*	*1*	*1*	*0*	*0*	*0*	***F***

OPTIMIZATION USING KARNAUGH MAPS

Now that we have the truth tables ready, we use Karnaugh maps to minimize the output functions.

ca

cd \ ab	00	01	11	10
00	0	1	0	0
01	1	0	1	0
11	0	0	0	1
10	0	0	0	0

cb

cd \ ab	00	01	11	10
00	0	0	1	0
01	0	1	0	0
11	0	0	1	1
10	0	1	1	0

cc

cd \ ab	00	01	11	10
00	0	0	1	0
01	0	0	0	0
11	0	0	1	0
10	1	0	1	0

cd

cd \ ab	00	01	11	10
00	0	1	0	0
01	1	0	0	0
11	0	1	1	0
10	0	0	0	1

ce

cd \ ab	00	01	11	10
00	0	1	0	0
01	1	1	0	1
11	1	1	0	0
10	0	0	0	0

cf

cd \ ab	00	01	11	10
00	0	0	0	0
01	1	0	1	0
11	1	1	0	0
10	1	0	0	0

cg

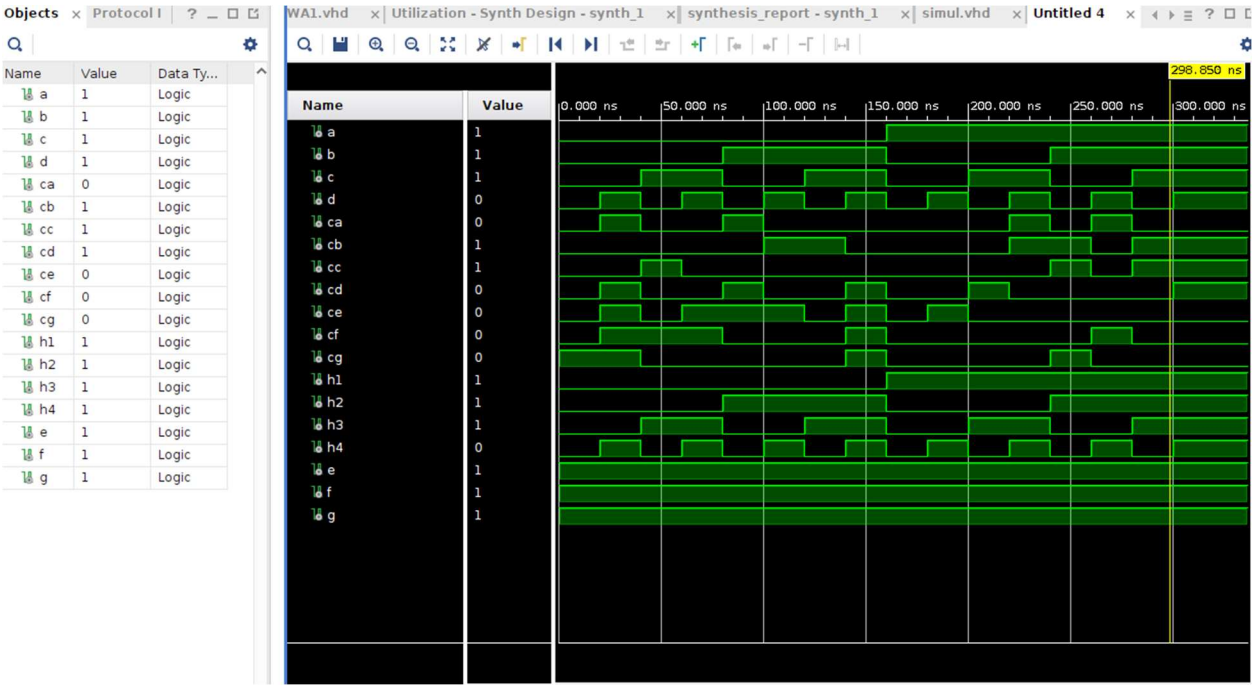
cd \ ab	00	01	11	10
00	1	0	1	0
01	1	0	0	0
11	0	1	0	0
10	0	0	0	0

Using these, we arrive at the following code snippet in VHDL:

```
ca <=  (not a and not b and not c and d) or (not a and b and not c and not d)
or (a and b and not c and d) or ( a and not b and c and d);

cb <= (b and c and not d) or (not a and b and  not c and d) or (a and b and
not d) or (a and c and d);

cc <= (a and b and not c and not d) or (not a and not b and c and not d) or
(a and b and c);

cd <= (not a and b and not c and not d) or (not a and not b and not c and d)
or (b and c and d) or (a and not b and c and not d);

ce <= (not a and d) or (not a and b and not c) or (not b and not c and d);

cf <= ( not a and not b and d ) or ( not a and not b and c ) or ( not a and c
and d) or ( a and b and not c and d );

cg <=  (not a and not b and not c) or (a and b and not c and not d) or (not a
and b and c and d);
```

SIMULATION

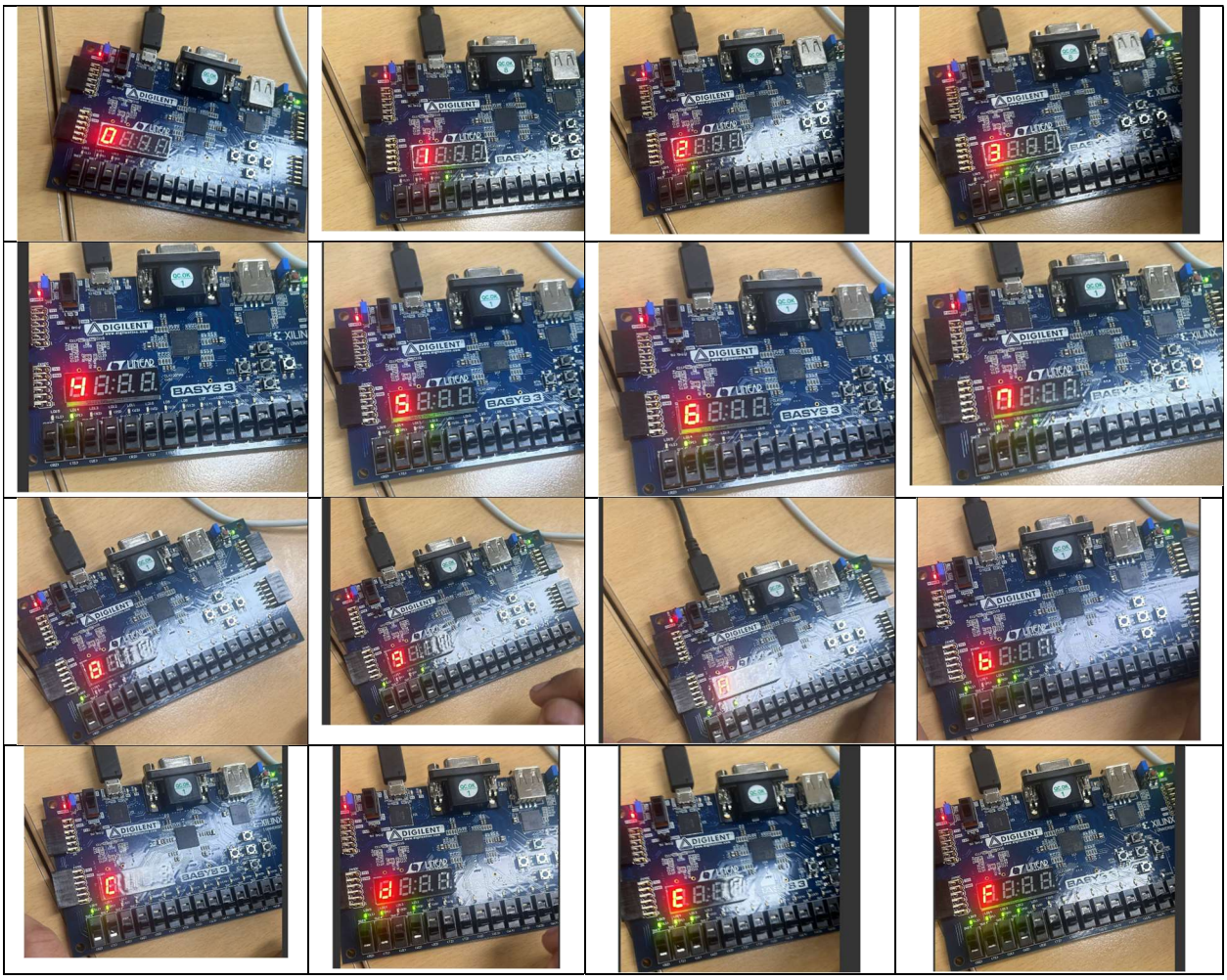
We performed a testbench simulation to check whether our implementation was correct or not prior to making a synthesis. Here, h1,h2,h3,h4 are signals related to led display corresponding to the "on" bits



IMPLEMENTATION AND SYNTHESIS REPORT

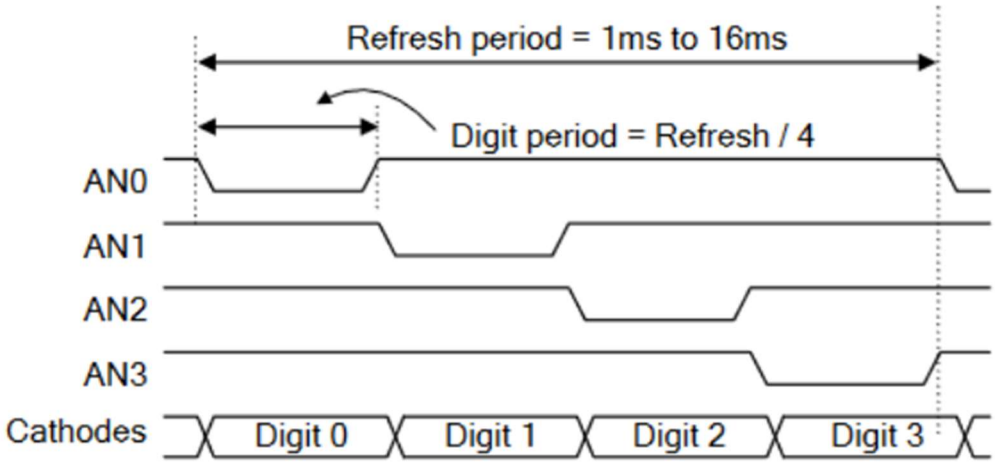
The resource utilization was obtained from the synthesis report and the snapshots of the working implementation.

```
Report Cell Usage:
+------+-----+------+
|      |Cell |Count |
+------+-----+------+
|1     |LUT4 |     7|S
|2     |IBUF |     4|
|3     |OBUF |    14|
+------+-----+------+
```



FROM 4 BITS TO 16: TACKLING THE COMMON ANODE PROBLEM

Now we want to extend our design so that we can use all four 7-segment displays of the basys3 board. But since the cathodes are tied to each anode, we cannot simply just provide 16 inputs. Rather, we need to use the inbuilt clock of the basys3 board to switch extremely fast between the four anodes so that we can avoid flickering of the display.



Schematic showing how to tackle the following

We use the "clk" variable as a timing reference and a counter which counts up to a specified number and then repeats again. We also maintain a variable flag which takes on values 0,1,2,3 to specify which anode is currently being used. Each time the counter hits the threshold, we update flag by using flag <= (flag+1)mod4. Which anodes to be used are kept track of by variables an0,an1,an2 and an3. The VHDL code on the following page exemplifies this.

```
entity HWA1 is
  port(a1,b1,c1,d1,a2,b2,c2,d2,a3,b3,c3,d3,a4,b4,c4,d4,clk:in std_logic;
ca,cb,cc,cd,ce,cf,cg,an0,an1,an2,an3,l1,l2,l3,l4,l5,l6,l7,l8,l9,l10,l11,l12,l
13,l14,l15,l16: out std_logic);
end HWA1;

architecture Behavioral of HWA1 is
signal counter : INTEGER := 0;
signal mux_select : STD_LOGIC_VECTOR(2 downto 0);
constant ANODE_DELAY : INTEGER := 1001;
signal flag: integer := 0;
begin
process(clk)
    begin
        if rising_edge(clk) then
            if counter = ANODE_DELAY then
                counter <= 0;
                if (flag mod 4) = 0 then

                    an0 <= '1';
                    an1 <= '1';
                    an2 <= '1';
                    an3 <= '0';

                    ca <=  (not a1 and not b1 and not c1 and d1) or (not a1
and b1 and not c1 and not d1) or (a1 and b1 and not c1 and d1) or ( a1 and
not b1 and c1 and d1);
                    cb <= (b1 and c1 and not d1) or (not a1 and b1 and  not
c1 and d1) or (a1 and b1 and not d1) or (a1 and c1 and d1);
                    cc <= (a1 and b1 and not c1 and not d1) or (not a1 and
not b1 and c1 and not d1) or (a1 and b1 and c1);
                    cd <= (not a1 and b1 and not c1 and not d1) or (not a1
and not b1 and not c1 and d1) or (b1 and c1 and d1) or (a1 and not b1 and c1
and not d1);
                    ce <= (not a1 and d1) or (not a1 and b1 and not c1) or
(not b1 and not c1 and d1);
                    cf <= ( not a1 and not b1 and d1 ) or ( not a1 and not b1
and c1 ) or ( not a1 and c1 and d1) or ( a1 and b1 and not c1 and d1 );
                    cg <=  (not a1 and not b1 and not c1) or (a1 and b1 and
not c1 and not d1) or (not a1 and b1 and c1 and d1);



                elsif (flag mod 4) = 1 then

                    an0 <= '1';
                    an1 <= '1';
                    an2 <= '0';
                    an3 <= '1';

                    ca <=  (not a2 and not b2 and not c2 and d2) or (not a2
and b2 and not c2 and not d2) or (a2 and b2 and not c2 and d2) or ( a2 and
not b2 and c2 and d2);
                    cb <= (b2 and c2 and not d2) or (not a2 and b2 and  not
c2 and d2) or (a2 and b2 and not d2) or (a2 and c2 and d2);
                    cc <= (a2 and b2 and not c2 and not d2) or (not a2 and
not b2 and c2 and not d2) or (a2 and b2 and c2);
```

```
                    cd <= (not a2 and b2 and not c2 and not d2) or (not a2
and not b2 and not c2 and d2) or (b2 and c2 and d2) or (a2 and not b2 and c2
and not d2);
                    ce <= (not a2 and d2) or (not a2 and b2 and not c2) or
(not b2 and not c2 and d2);
                    cf <= ( not a2 and not b2 and d2 ) or ( not a2 and not b2
and c2 ) or ( not a2 and c2 and d2) or ( a2 and b2 and not c2 and d2 );
                    cg <=  (not a2 and not b2 and not c2) or (a2 and b2 and
not c2 and not d2) or (not a2 and b2 and c2 and d2);



                elsif (flag mod 4) = 2 then

                    an0 <= '1';
                    an1 <= '0';
                    an2 <= '1';
                    an3 <= '1';

                     ca <=  (not a3 and not b3 and not c3 and d3) or (not a3
and b3 and not c3 and not d3) or (a3 and b3 and not c3 and d3) or ( a3 and
not b3 and c3 and d3);
                    cb <= (b3 and c3 and not d3) or (not a3 and b3 and  not
c3 and d3) or (a3 and b3 and not d3) or (a3 and c3 and d3);
                    cc <= (a3 and b3 and not c3 and not d3) or (not a3 and
not b3 and c3 and not d3) or (a3 and b3 and c3);
                    cd <= (not a3 and b3 and not c3 and not d3) or (not a3
and not b3 and not c3 and d3) or (b3 and c3 and d3) or (a3 and not b3 and c3
and not d3);
                    ce <= (not a3 and d3) or (not a3 and b3 and not c3) or
(not b3 and not c3 and d3);
                    cf <= ( not a3 and not b3 and d3 ) or ( not a3 and not b3
and c3 ) or ( not a3 and c3 and d3) or ( a3 and b3 and not c3 and d3 );
                    cg <=  (not a3 and not b3 and not c3) or (a3 and b3 and
not c3 and not d3) or (not a3 and b3 and c3 and d3);



                else

                    an0 <= '0';
                    an1 <= '1';
                    an2 <= '1';
                    an3 <= '1';

                    ca <=  (not a4 and not b4 and not c4 and d4) or (not a4
and b4 and not c4 and not d4) or (a4 and b4 and not c4 and d4) or ( a4 and
not b4 and c4 and d4);
                    cb <= (b4 and c4 and not d4) or (not a4 and b4 and  not
c4 and d4) or (a4 and b4 and not d4) or (a4 and c4 and d4);
                    cc <= (a4 and b4 and not c4 and not d4) or (not a4 and
not b4 and c4 and not d4) or (a4 and b4 and c4);
                    cd <= (not a4 and b4 and not c4 and not d4) or (not a4
and not b4 and not c4 and d4) or (b4 and c4 and d4) or (a4 and not b4 and c4
and not d4);
                    ce <= (not a4 and d4) or (not a4 and b4 and not c4) or
(not b4 and not c4 and d4);
```

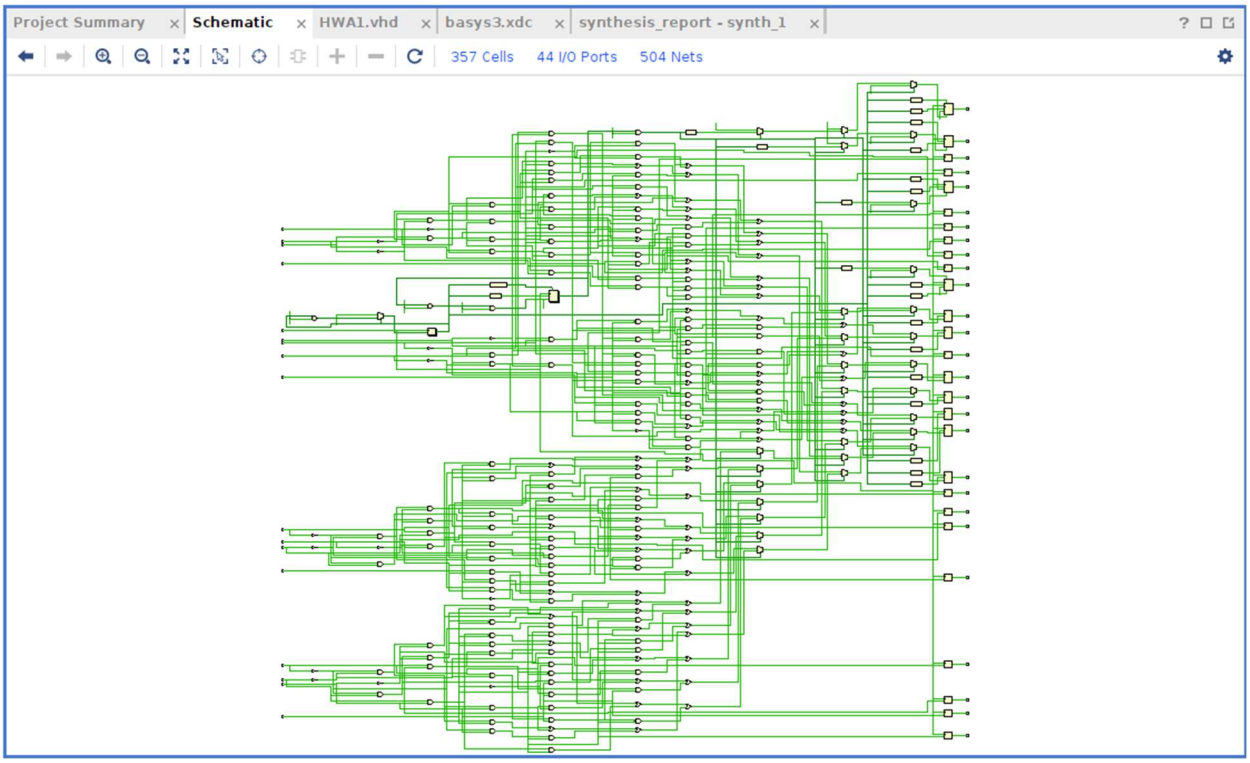
```
                    cf <= ( not a4 and not b4 and d4 ) or ( not a4 and not b4
and c4 ) or ( not a4 and c4 and d4) or ( a4 and b4 and not c4 and d4 );
                    cg <=  (not a4 and not b4 and not c4) or (a4 and b4 and
not c4 and not d4) or (not a4 and b4 and c4 and d4);

            end if;
            else
                counter <= counter + 1;
                flag <= ((flag+1) mod 4);
            end if;

--ca <=  (not a and not b and not c and d) or (not a and b and not c and not
d) or (a and b and not c and d) or ( a and not b and c and d);
--cb <= (b and c and not d) or (not a and b and  not c and d) or (a and b and
not d) or (a and c and d);
--cc <= (a and b and not c and not d) or (not a and not b and c and not d) or
(a and b and c);
--cd <= (not a and b and not c and not d) or (not a and not b and not c and
d) or (b and c and d) or (a and not b and c and not d);
--ce <= (not a and d) or (not a and b and not c) or (not b and not c and d);
--cf <= ( not a and not b and d ) or ( not a and not b and c ) or ( not a and
c and d) or ( a and b and not c and d );
--cg <=  (not a and not b and not c) or (a and b and not c and not d) or (not
a and b and c and d);

l1 <= a1;
l2 <= b1;
l3 <= c1;
l4 <= d1;
l5 <= a2;
l6 <= b2;
l7 <= c2;
l8 <= d2;
l9 <= a3;
l10 <= b3;
l11 <= c3;
l12 <= d3;
l13 <= a4;
l14 <= b4;
l15 <= c4;
l16 <= d4;


end if;
end process;

end Behavioral;
```

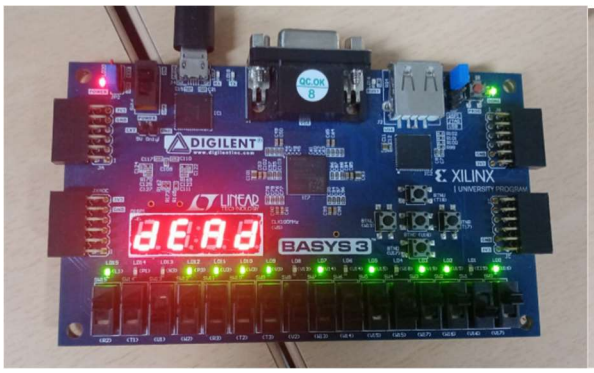
SYNTHESIS REPORT AND IMPLEMENTATION

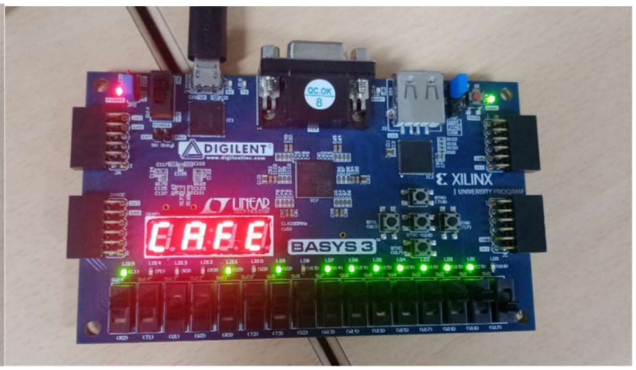
The resource utilization was obtained from the synthesis report. Moreover, the schematic diagram was obtained and is presented here.

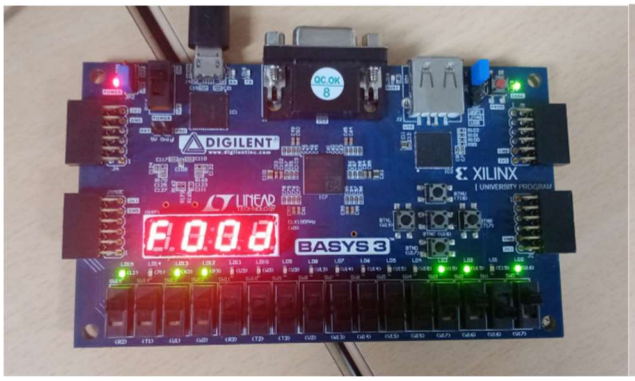
```
Report Cell Usage:
+------+-----+------+
|      |Cell |Count |
+------+-----+------+
|1     |LUT4 |     7|
|2     |IBUF |     4|
|3     |OBUF |    14|
+------+-----+------+
```

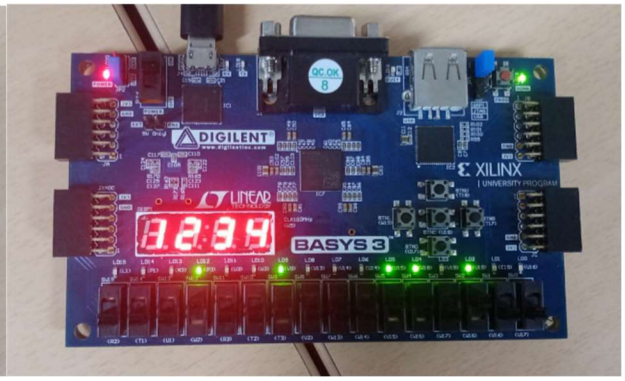
```
---------------------------------------------------
Detailed RTL Component Info :
+---Adders :
           2 Input   32 Bit       Adders := 1
           2 Input    3 Bit       Adders := 1
+---Registers :
                     32 Bit    Registers := 2
                      1 Bit    Registers := 27
+---Muxes :
           2 Input   32 Bit        Muxes := 1
           2 Input    4 Bit        Muxes := 1
           4 Input    1 Bit        Muxes := 8
           2 Input    1 Bit        Muxes := 6
```


Project Summary
Schematic
HWA1.vhd
basys3.xdc
synthesis_report - synth_1
357 Cells
44 I/O Ports
504 Nets


DIGILENT
XILINX
BASYS 3


DIGILENT
XILINX
BASYS 3


DIGILENT
XILINX
BASYS 3


DIGILENT
XILINX
BASYS 3

CONCLUSIONS AND LEARNINGS

We explored the vivado software and learned how to control the seven segment display and the internal clock of the basys3 board. Simulations to check proper working of the circuit design were performed and exhaustive testing for proper outcomes was done,