Embedded Electronics Lab

Ragesh Ramachandran
Vivekanandan Pazhanirajan

Question 1: To display number 5 on digit 0 of the 7-segment display of the Xilinx cool runner board.

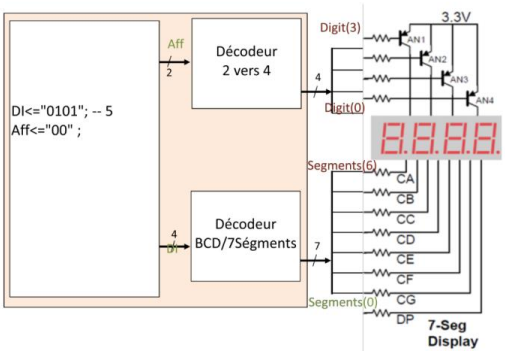


Fig 1: Circuit for the program 1

The program was done using the concept of components so we used three entity and their architecture. The first entity was for decoder which is necessary for selecting the corresponding digit on the seven-segment display. There are 4 digits in the display and each of these can be individually selected. Only 2-bits are necessary for the selection of 4 digits. So, the entity '*decoder*' has input '*aff* ' which is a '*standard_logic_vector* ' of 2 bits and an output digit which is also a '*standard_logic_vector* ' of 4 bits. The architecture part of the decoder decides the behavior of the decoder. The '*aff*' is used to select a particular *'digit'* in the display

```
1
2   library IEEE;
3   use IEEE.STD_LOGIC_1164.ALL;
4   use ieee.std_logic_arith.all;
5   use ieee.std_logic_unsigned.all;
6
7
8   entity decoder is port
9       (aff: in std_logic_vector (1 downto 0) ;
10      digit: out std_logic_vector (3 downto 0));
11  end decoder;
12
13  architecture archi_decoder of decoder is
14
15  begin
16      with aff select
17           digit <= "1110" when "00",
18            "1101" when "01",
19            "1011" when "10",
20            "0111" when "11",
21            "1111" when others;
22
23  end archi_decoder;
24
```

This part of the program is used to display digits in the seven-segment display using the 4-bit inputs. The seven-segment display can display 10 digit from 0 to 9.

The entity ‘bcdTo7seg’ part has ‘di’ as the input which is a *‘std_logic_vector’* of 4 bits and segments as the output which is a *‘std_logic_vector’* of 7 bits. The architecture *‘arCbcdToseg’* part decides the numbers (0 to 9) based on the input *'di'*

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity bcdTo7seg is port
(di: in std_logic_vector (3 downto 0) ;
segments: out std_logic_vector (6 downto 0));
end bcdTo7seg;

architecture arCbcdTo7seg of bcdTo7seg is

begin
    with di select
         segments <= "0000001" when "0000",
                 "1001111" when "0001",
             "0010010" when "0010",
             "0000110" when "0011",
             "1001100" when "0100",
             "0100100" when "0101",
             "0100000" when "0110",
             "0001111" when "0111",
             "0000000" when "1000",
             "0000100" when "1001",
             "1111111" when others;

end arCbcdTo7seg;

```

The entity ‘program1’ part has ‘digit’ as the output with *‘std_logic_vector’* of 4-bits’ and ‘segments’ as the output with *‘std_logic_vector’* of 7-bits. The architecture ‘Behavioural’ part gets the signal ‘aff’ and ‘di’. Now we use the decoder and 7-segment display written before as components in this program so we can reuse it. The components ‘decoder’ and ‘bcdTo7seg’ are defined in the architecture *‘Behavioural’* part. Now we use a command named *"portmap"* and using this *‘aff’* and *‘digit’* are port mapped and also the *‘di’* and *‘segments’* are port mapped. Now the output of the display is 5 on the digit at unit's place

```
50
51    end arCbcdTo7seg;
52
53    entity program1 is port
54          ( digit: out std_logic_vector (3 downto 0);
55        segments: out std_logic_vector (6 downto 0));
56    end program4;
57
58    architecture Behavioral of program1 is
59
60        signal aff: std_logic_vector (1 downto 0) ;
61        signal di: std_logic_vector (3 downto 0) ;
62
63        component decoder port
64            (aff: in std_logic_vector (1 downto 0) ;
65            digit: out std_logic_vector (3 downto 0));
66        end component;
67
68        component bcdTo7seg port
69            (di: in std_logic_vector (3 downto 0) ;
70            segments: out std_logic_vector (6 downto 0));
71        end component;
72
73    begin
74        aff = "00";
75        di = "0101"
76        digit: decoder port map ( aff,digit);
77        segment: bcdTo7seg port map (di,segments);
78
79    end Behavioral;
80
```

Question 2: To display value 0 on digit 0 of the Xilinx cool runner board.

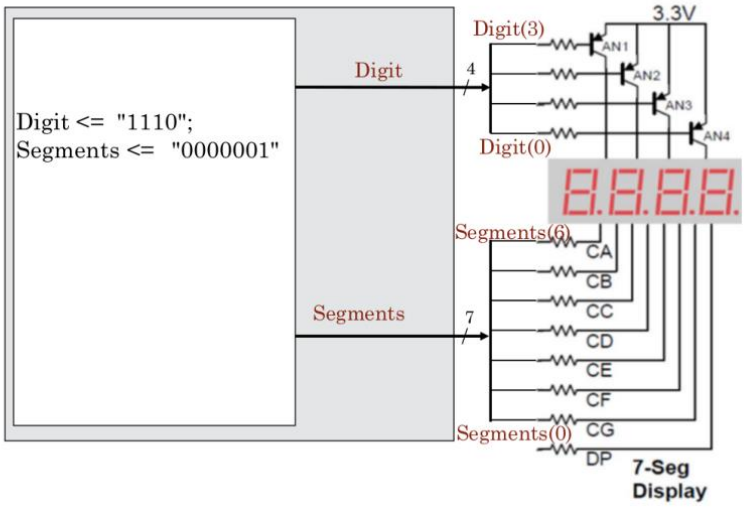


Fig 2: Circuit for the program 2

The same program was used for this question with only a small modification in the architecture *'Behavioural'* part. This is the advantage of using the components it helps in using the code again with minor changes. So, in this program the *'aff'* is assigned "00" which select the digit in unit's place and the digit is assigned *"0000"* which display the number *'0'* in the 7-segment display.

```
50
51  end arCbcdTo7seg;
52
53  entity program1 is port
54        ( digit: out std_logic_vector (3 downto 0);
55      segments: out std_logic_vector (6 downto 0));
56  end program4;
57
58  architecture Behavioral of program1 is
59
60      signal aff: std_logic_vector (1 downto 0) ;
61      signal di: std_logic_vector (3 downto 0) ;
62
63      component decoder port
64          (aff: in std_logic_vector (1 downto 0) ;
65          digit: out std_logic_vector (3 downto 0));
66      end component;
67
68      component bcdTo7seg port
69          (di: in std_logic_vector (3 downto 0) ;
70          segments: out std_logic_vector (6 downto 0));
71      end component;
72
73  begin
74      aff = "00";
75      di = "0000"
76      digit: decoder port map ( aff,digit);
77      segment: bcdTo7seg port map (di,segments);
78
79  end Behavioral;
80
```

Question 3: perform a scan of the display using counter to display the number 2018

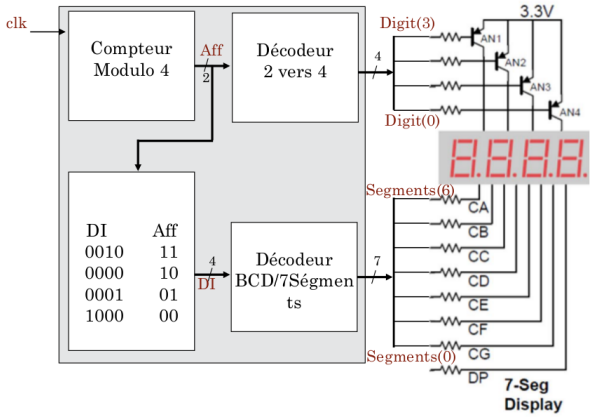


Fig 3: Circuit for the program 3

This program was done using 4 entities and they are:

- '*Decoder*' for selecting the digit place to be displayed.
- *'BcdTo7seg'* to display the digit in the 7-segment display.
- '*Counter*' to do a modulo 4 counter and also uses this to display *"2008"* in the display.
- '*Prog4*' to combine all these entity as components and display the output *"2018"*

The entity decoder has '*aff*' as input of type *'std_logic_vector'* of 2-bits and '*digit*' as output of type *'std_logic_vector'* of 4-bits. The architecture part of the decoder decides the behavior of the decoder. The '*aff*' is used to select a particular *'digit'* in the display.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
----------------------------------------------
entity decoder is port
    (aff: in std_logic_vector (1 downto 0) ;
    digit: out std_logic_vector (3 downto 0));
end decoder;
----------------------------------------------
architecture archi_decoder of decoder is

begin
    with aff select
         digit <= "1110" when "00",
          "1101" when "01",
          "1011" when "10",
          "0111" when "11",
          "1111" when others;

end archi_decoder;
----------------------------------------------
```

Entity *'Decoder'*

The entity ‘bcdTo7seg’ part has ‘di’ as the input which is a *‘std_logic_vector’* of 4 bits and segments as the output which is a *‘std_logic_vector’* of 7 bits. The architecture *‘arCbcdToseg’* part decides the numbers (0 to 9) based on the input *'di'*

```
24  ----------------------------------------------
25  library IEEE;
26  use IEEE.STD_LOGIC_1164.ALL;
27  use ieee.std_logic_arith.all;
28  use ieee.std_logic_unsigned.all;
29  ----------------------------------------------
30  entity bcdTo7seg is port
31  (di: in std_logic_vector (3 downto 0) ;
32  segments: out std_logic_vector (6 downto 0));
33  end bcdTo7seg;
34  ----------------------------------------------
35  architecture arCbcdTo7seg of bcdTo7seg is
36
37  begin
38      with di select
39           segments <= "0000001" when "0000",
40                   "1001111" when "0001",
41               "0010010" when "0010",
42               "0000110" when "0011",
43               "1001100" when "0100",
44               "0100100" when "0101",
45               "0100000" when "0110",
46               "0001111" when "0111",
47               "0000000" when "1000",
48               "0000100" when "1001",
49               "1111111" when others;
50
51  end arCbcdTo7seg;
52  ----------------------------------------------
```

Entity *BcdTo7seg'*

- The entity *'counter'* has *'clk'* as input of type 'std_logic' which is used for the counter to start counting at every rising edge of the clock.
- 'di' as output of type *‘std_logic_vector’* of 4 bits which is used for selecting the number to be displayed from 0 to 9 in the 7-segement display.

- 'aff' as output of type *'std_logic_vector'* of 2 bits which is used for selecting the digit place to display the number

```
53  ----------------------------------------------------
54  library IEEE;
55  use IEEE.STD_LOGIC_1164.ALL;
56  use ieee.std_logic_arith.all;
57  use ieee.std_logic_unsigned.all;
58  ----------------------------------------------------
59  entity counter is port
60      (clk: in std_logic;
61      di: out std_logic_vector (3 downto 0);
62      aff: out std_logic_vector (1 downto 0)) ;
63  end counter;
64  ----------------------------------------------------
65  architecture archi_counter of counter is
66
67      signal count: integer :=0 ;
68      signal divclk: std_logic :='0';
69      signal affc:std_logic_vector (1 downto 0);
70
```

Entity '*Counter*'

In the architecture we made two processes. First process is used for division of the clock frequency from 8 Mhz to a lower value. This done by using a signal count which is incremented every rising edge of the clock and once the value of the count reaches 400000 then the new clock is generated and this repeats as long as the clock is available.

The second process uses this newly generated clock to initialize the process and, in this process, for every rising edge of the new clock the digits as well as the digit place is selected.

```
70
71    begin
72        process(clk)
73        begin
74        if rising_edge(clk) then
75            count <= count + 1;
76            if cn = 4000000 then
77            divclk <= not(divclk);
78            count <= 0;
79            end if;
80        end if;
81        end process;
82
83        process(divclk)
84        begin
85            if rising_edge(divclk) then
86
87            case affc is
88                when "00" => di <= "1000";
89                when "01" => di <= "0001";
90                when "10" => di <= "0000";
91                when "11" => di <= "0010";
92                when others => di <= "0000";
93            end case;
94                affc<=affc+1;
95                aff<=affc;
96            end if;
97        end process;
98
99    end archi_counter;
```

The entity *'pro4'* part has *'digit'* as the output with *'std_logic_vector'* of 4-bits' which is used to select the digit place and *'segments'* as the output with *'std_logic_vector'* of 7-bits which displays the digit and *'clk'* as input of type *'std_logic'* which is used for the counter to start counting at every rising edge of the clock.

The architecture ‘archi_prog4’ part uses the signal ‘aff’ and ‘di’. Now we use the decoder and 7-segment display and counter written before as components in this program. The components ‘decoder’, ‘bcdTo7seg’ and '*Counter*' are defined in the architecture ‘archi_prog4’ part before the program begins.

```
100    ----------------------------------------------------------------
101    ----------------------------------------------------------------
102    entity prog4 is port
103          ( digit: out std_logic_vector (3 downto 0);
104        segments: out std_logic_vector (6 downto 0);
105        clk: in std_logic;
106        bp0,bp1: in std_logic);
107    end prog4;
108    ----------------------------------------------------------------
109    architecture archi_prog4 of prog4 is
110
111        signal aff: std_logic_vector (1 downto 0) ;
112        signal di: std_logic_vector (3 downto 0) ;
113
114        component decoder port
115            (aff: in std_logic_vector (1 downto 0) ;
116            digit: out std_logic_vector (3 downto 0));
117        end component;
118
119        component bcdTo7seg port
120            (di: in std_logic_vector (3 downto 0) ;
121            segments: out std_logic_vector (6 downto 0));
122        end component;
123
124        component counter port
125            (clk: in std_logic;
126            di: out std_logic_vector (3 downto 0);
127            aff: out std_logic_vector (1 downto 0) :="00");
128        end component;
129    ----------------------------------------------------------------
130
```

Now we use a command named *"portmap"* and using this *‘aff’* and *‘digit’* are port mapped, *‘di’* and *‘segments’* are port mapped and *‘aff’, 'clk'* and *‘di’* are port mapped. Now the output of the display is *"2008"*

```
129    ------------------------------------
130
131    begin
132
133        digit: decoder
134            port map ( aff, digit);
135        segment: bcdTo7seg
136            port map (di, segments);
137        counter:counter
138            port map(clk, di, aff );
139
140    end archi_prog4;
141    ------------------------------------
142    ------------------------------------
```

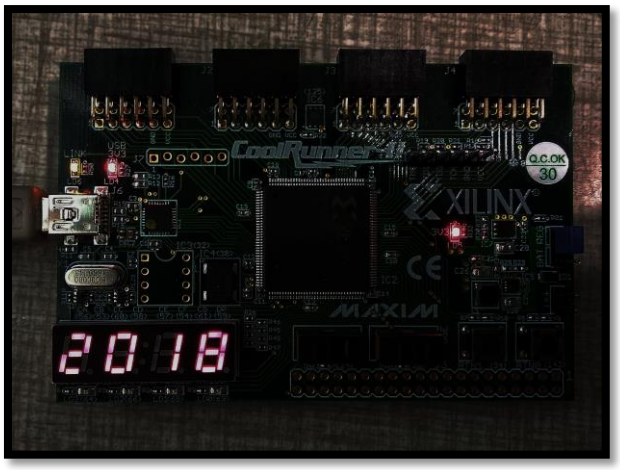


Fig 4: Results obtained

Question 4: To implement a calculator using Xilinx cool runner II

The goal of this lab is to implement a calculator in Xilinx cool runner with the following requirements:

z = x ~ y with

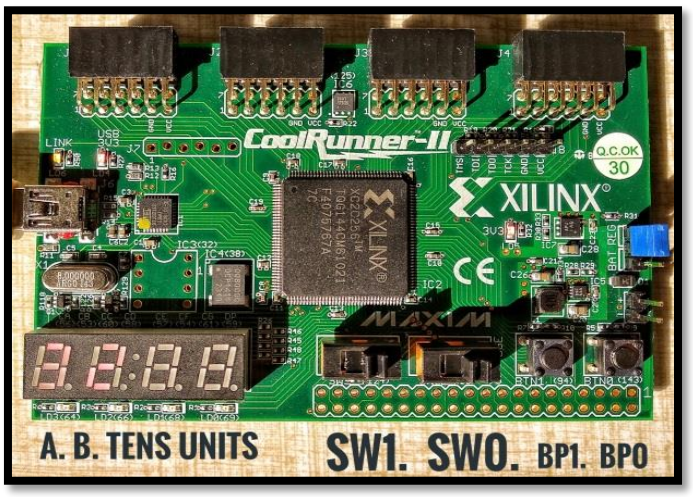
- ~ operator is either+ or –or *
- x on the left digit (Digit 3)
- y on Digit 2
- z on the 2 right digits (Digit 1 Digit 0)

Choose x and y:

- x increments (modulo 10) each time youpressBP0
- y increments (modulo 10) each time youpressBP1

Operator is fixed by switch SW0 and SW1:

- + if SW0 = 0 and SW1 = 0
- - if SW0 = 1 and SW1 = 0
- * if SW1 = 1



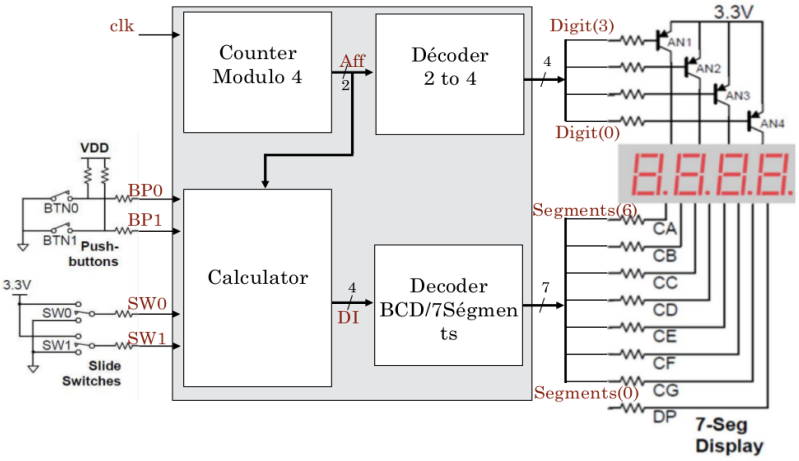


Fig 5: Circuit for the calculator

This program was done using 5 entities and they are:

- '*Decoder'* for selecting the digit place to be displayed.
- *'BcdTo7seg'* to display the digit in the 7-segment display.
- '*Counter'* to do a modulo 4 counter operation.
- '*Calculator'* to do the operation on the inputs and also to input the numbers and operands
- *'Program5* to combine all these entity as components and then display the output on the 7-segment display as per the specifications

Entity '*Decoder*'

The entity decoder has '*aff*' as input of type *'std_logic_vector'* of 2-bits and '*digit'* as output of type *'std_logic_vector'* of 4-bits. The architecture part of the decoder decides the behavior of the decoder. The '*aff*' is used to select a particular *'digit'* in the display.

```
1
2   ----------------------------------------------------------
3   library IEEE;
4   use IEEE.STD_LOGIC_1164.ALL;
5   use ieee.std_logic_arith.all;
6   use ieee.std_logic_unsigned.all;
7
8
9   entity decoder is port
10  (aff: in std_logic_vector (1 downto 0) ;
11  digit: out std_logic_vector (3 downto 0));
12  end decoder;
13
14  architecture archi_decoder of decoder is
15
16  begin
17      with aff select
18              digit <= "1110" when "00",
19                       "1101" when "01",
20                       "1011" when "10",
21                       "0111" when "11",
22                       "1111" when others;
23
24  end archi_decoder;
25  ----------------------------------------------------------
```

Entity *BcdTo7seg'*

The entity ‘bcdTo7seg’ part has ‘di’ as the input which is a *‘std_logic_vector’* of 4 bits and segments as the output which is a *‘std_logic_vector’* of 7 bits. The architecture *‘arCbcdToseg’* part decides the numbers (0 to 9) based on the input *'di'*

```
25  ----------------------------------------------------------
26  library IEEE;
27  use IEEE.STD_LOGIC_1164.ALL;
28  use ieee.std_logic_arith.all;
29  use ieee.std_logic_unsigned.all;
30
31  entity bcdTo7seg is port
32  (di: in std_logic_vector (3 downto 0) ;
33  segments: out std_logic_vector (6 downto 0));
34  end bcdTo7seg;
35
36  architecture archi_7seg of bcdTo7seg is
37
38  begin
39      with di select
40           segments <= "0000001" when "0000",
41                       "1001111" when "0001",
42                       "0010010" when "0010",
43                       "0000110" when "0011",
44                       "1001100" when "0100",
45                       "0100100" when "0101",
46                       "0100000" when "0110",
47                       "0001111" when "0111",
48                       "0000000" when "1000",
49                       "0000100" when "1001",
50                       "1111111" when others;
51
52  end archi_7seg;
53  ----------------------------------------------------------
```

Entity '*Counter*'

\

The entity *'counter*' has *'clk'* as input of type 'std_logic' which is used for the counter to start counting at every rising edge of the clock.

'di' as output of type *‘std_logic_vector’* of 4 bits which is used for selecting the number to be displayed from 0 to 9 in the 7-segement display.

'aff' as output of type *‘std_logic_vector’* of 2 bits which is used for selecting the digit place to display the number

```
53  ----------------------------------------------------------
54  library IEEE;
55  use IEEE.STD_LOGIC_1164.ALL;
56  use ieee.std_logic_arith.all;
57  use ieee.std_logic_unsigned.all;
58
59  entity counter is port
60  (clk: in std_logic;
61  aff: out std_logic_vector (1 downto 0)) ;
62  end counter;
63
64  architecture archi_counter of counter is
65  signal cn: integer :=0 ;
66  signal divclk: std_logic :='0';
67  signal affc:std_logic_vector (1 downto 0);
68  begin
69
70  Clock_div: process(clk)
71  begin
72      if clk'event and clk='1' then
73                  cn <= cn + 1;
74                  if cn= 400000 then
75                  divclk <= not(divclk);
76                  cn<=0;
77                  end if;
78      end if;
79  end process;
80  pfs1: process(divclk)
81  begin
82          if divclk'event and divclk='1' then
83
84                      affc<=affc-1;
85                  aff<=affc;
86          end if;
87  end process;
88
89  end archi_counter;
90  ----------------------------------------------------------
```

Entity *'Calculator'*

The entity takes '*bp0', 'bp1',' sw0',' sw1'* as inputs of *'std_logic'* in order to get inputs form the switches , *'aff'* as input of 2-bits of type '*std_logic_vector'* and '*di'* of 4-bits as output of type '*std_logic_vector'*.

In the architecture of the calculator we need 5 signals of type '*std_logic_vector'* and 4-bits each. This is for performing the operations as well as selecting the units and tens place in the display.

```
90    ----------------------------------------------------
91    library IEEE;
92    use IEEE.STD_LOGIC_1164.ALL;
93    use ieee.std_logic_arith.all;
94    use ieee.std_logic_unsigned.all;
95
96    entity calculator is port
97    (
98    bp0,bp1 : in std_logic;
99    sw0,sw1 : in std_logic;
100   aff: in std_logic_vector(1 downto 0);
101   di: out std_logic_vector(3 downto 0));
102   end calculator;
103
104   architecture archi_calculator of calculator is
105   signal x: std_logic_vector (3 downto 0)
106   signal y: std_logic_vector (3 downto 0)
107   signal z: std_logic_vector (7 downto 0)
108
109   signal units: std_logic_vector (3 downto 0)
110   signal tens: std_logic_vector (3 downto 0)
111
112   ----------------------------------------------------
```

The process *'increment_XY'* is used for incrementing the values of *'x'* and *'y'* and this process is triggered using the outputs of the button switches *'bp0', 'bp1'*. Modulo 10 operation is done because we want only the numbers from 0 to 9.

- x increments (modulo 10) each time youpressBP0
- y increments (modulo 10) each time youpressBP1

The process *'operation'* is used for selecting the operands. The process is triggered when it receives signal from *'sw0',' sw1'* .

- \+ if SW0 = 0 and SW1 = 0
- \- if SW0 = 1 and SW1 = 0
- * if SW1 = 1

```
112     ------------------------------------------------
113     begin
114     increment_XY: process(bp0,bp1)
115     begin
116             if bp0'event and bp0 = '0' then
117                 y<=y+1;
118
119                 end if;
120             if bp1'event and bp1 = '0' then
121                 x<=x+1;
122
123             end if;
124
125             if y="1010" then
126                         y<="0000";
127             end if;
128             if x="1010" then
129                         x<="0000";
130             end if;
131
132     end process;
133
134     operation: process(sw0,sw1)
135     variable N: integer:=1;
136     ------------------------------------------------
```

```
136     ------------------------------------------------
137     begin
138         if  sw1='0' then
139                 if sw0='0' then
140                 z<= ("000" & x) + ("000" & y);
141
142                 elsif sw0='1' then
143                     if x>y then
144                         z<= x-y;
145                     else
146                         z<= y-x;
147                     end if;
148                 end if;
149
150         elsif  sw1='1' then
151                 z <= x*y;
152         end if;
153     ------------------------------------------------
```

This part of the code is used to separate the units and the tens place of the output *'z'*. We tried using MOD operator but it did not produce the results as expected so we discussed with other groups and came to this solution. This performs the same operation as that of MOD operator.

The output is processed to determine its units and tens place. This is then used to select the corresponding digits in the 7-segment display.

```
153    ----------------------------------------------------------
154            for N in 1 to 9 loop
155                if (10*N) = z then
156                    units <="0000";
157                    tens <= conv_std_logic_vector(N,4);
158                    exit;
159                elsif (10*N) > z then
160                    units <= z - 10*(N-1);
161                    tens <= conv_std_logic_vector(N,4)-1;
162                    exit;
163                end if;
164            end loop;
165    end process;
166
167    with aff select
168            di<= y when "10",
169                 x when "11",
170                  units when "00",
171                  tens when "01",
172                  "1000" when others;
173
174    end archi_calculator;
175    ----------------------------------------------------------
```

The entity *'program5'* is the final entity and it has *'clk','bp0', 'bp1',' sw0',' sw1'* as inputs of *'std_logic* , *'digit* as output of 4 -bits of type *'std_logic_vector' to select the digits place* and 'segments' of 7-bits as output of type *'std_logic_vector'* for displaying the numbers on 7-segment display.

In the architecture of program5 we used the concepts of components to make the programming easier. We used the entities *'deoder', 'bcdTo7Seg', 'counter', 'calculator'* inside the architecure and now we just need to portmap the inputs and outputs.

```
176   library IEEE;
177   use IEEE.STD_LOGIC_1164.ALL;
178   use ieee.std_logic_arith.all;
179   use ieee.std_logic_unsigned.all;
180
181   entity program5 is port
182   (
183   digit: out std_logic_vector (3 downto 0);
184   segments: out std_logic_vector (6 downto 0);
185   clk: in std_logic;
186   bp0,bp1: in std_logic;
187   sw0,sw1: in std_logic);
188   end program5;
189
190   architecture Behavioral of program5 is
191
192   signal aff: std_logic_vector (1 downto 0) ;
193   signal di: std_logic_vector (3 downto 0) ;
194
195   component decoder port
196       (aff: in std_logic_vector (1 downto 0) ;
197       digit: out std_logic_vector (3 downto 0));
198   end component;
199
200   component bcdTo7seg port
201       (di: in std_logic_vector (3 downto 0) ;
202       segments: out std_logic_vector (6 downto 0));
203   end component;
204
205   component counter port
206       (clk: in std_logic;
207       aff: out std_logic_vector (1 downto 0));
208   end component;
209
210   component calculator port
211       (bp0,bp1 : in std_logic;
212       sw0,sw1 : in std_logic;
213       aff: in std_logic_vector(1 downto 0);
214       di: out std_logic_vector(3 downto 0));
215   end component;
```

- portmap the input '*aff*' and output *'digit'* of the counter
- portmap the input '*clk*' and output *'aff'* of the decoder.
- portmap the input '*di*' and output *'segments'* of the decoder.
- portmap the inputs *'bp0', 'bp1',' sw0',' sw1'* and outputs *aff' and 'di'* of the decoder.

```
216
217     begin
218     decoder
219         port map ( aff,digit);
220     bcdTo7seg
221         port map (di,segments);
222     counter
223         port map(clk,aff);
224     calculator
225         port map(bp0,bp1,sw0,sw1,aff,di);
226
227     end Behavioral;
228
```

Results obtained

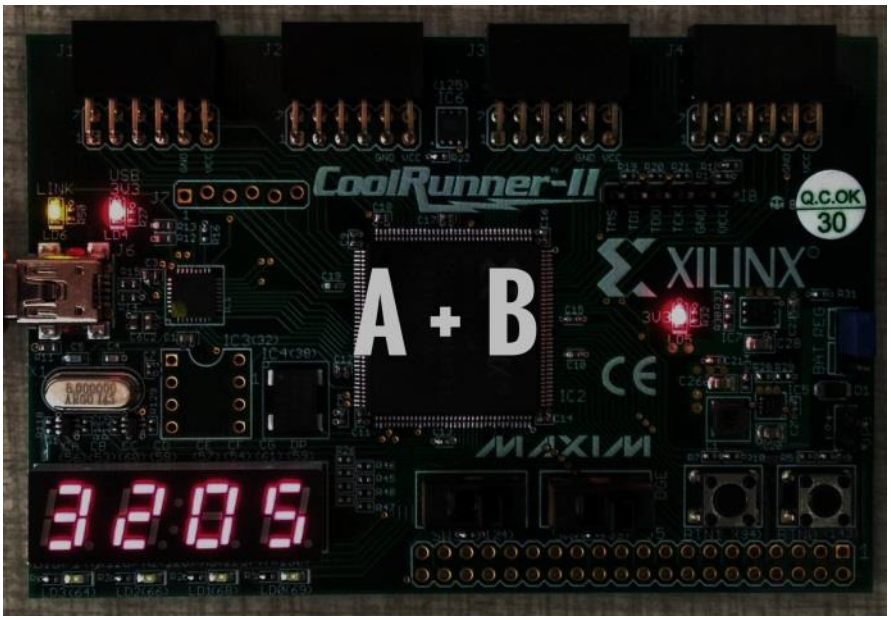


Fig 6: Addition operation

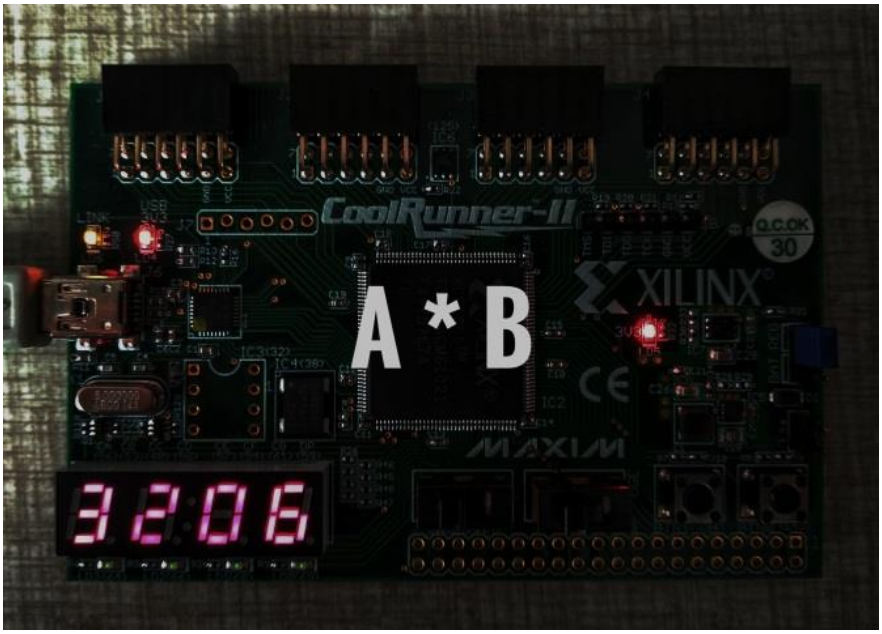


Fig 7: Multiplication operation

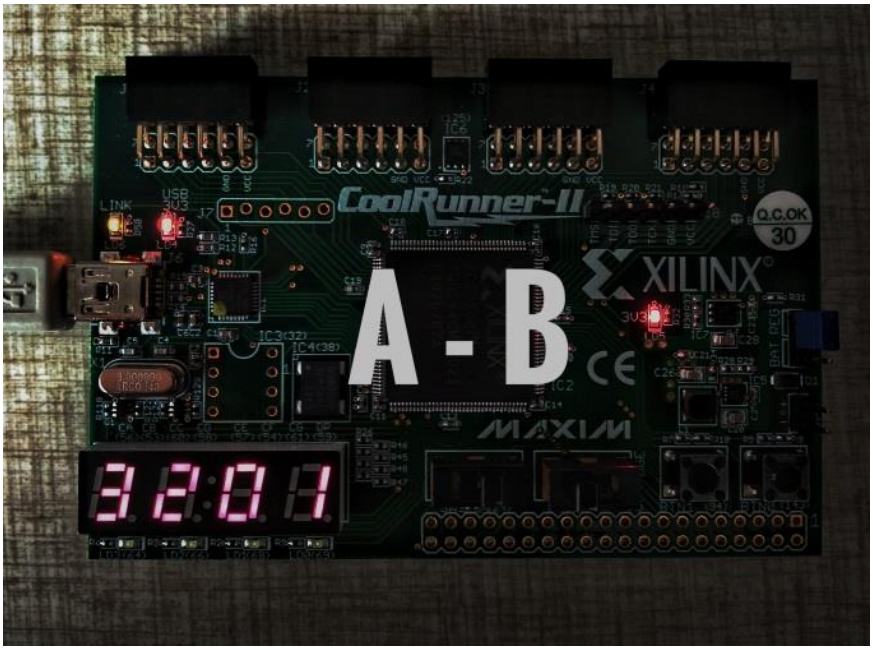


Fig 8: Subtraction operation

Video link: https://drive.google.com/open?id=1c7B8S07kvOYUtDbuzeuENgBPgMNBJ9Am