دانشگاه صنعتی امیرکبیر
(پلی‌تکنیک تهران)
دانشکده برق

گزارشکار آزمایشگاه مدار منطقی

پروژه درس

نگارش
پارسا محمّدی ۹۹۲۳۱۲۱

استاد درس
مهندس مهرابادی

تیر ۱۴۰۲

فهرست

ساختار پروژه

در این پروژه یک ALU طبق خواسته صورت پروژه ساخته شده است و در فایل ALU قرار دارد. و همچنین یک تست بنچ برای بررسی درستی عملکرد آن طراحی شده است.

در این پروژه نیاز به استفاده از چندین کد دستور مختلف برای مشخص کردن ورودی و خروجی می‌باشد. جدول کد ها به صورت زیر می باشند.

Op code	Operation
0000	SUB
0001	AND
0010	ADD
0011	OR
0100	MUL
0101	XOR
0110	Shift Right
0111	Shift Left
1000	Write
1001	Read

با توجه به اینکه این واحد پردازش باید ۱۰ دستور مختلف را پردازش کند به همین منظور از دستورات ۴ بیتی برای مپ کردن اپ کد و عملیات استفاده شده است. و از ۰ شروع شده و تا ۹ می روند.

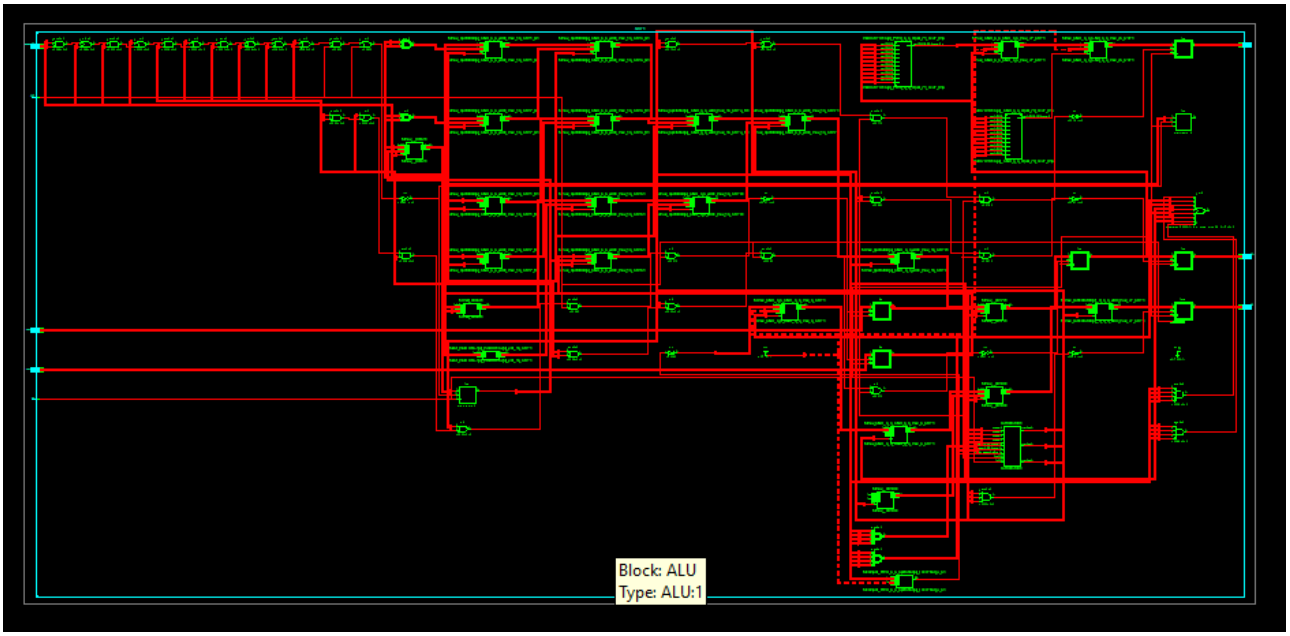
این کد ها به ورودی ALU داده می‌شوند تا ALU بدلند در ادامه چه کاری را بر روی ورودی هایش انجام دهد.

code	State
000	Ready
001	In Prosses
010	Writing Process
011	Reading Process
100	In output

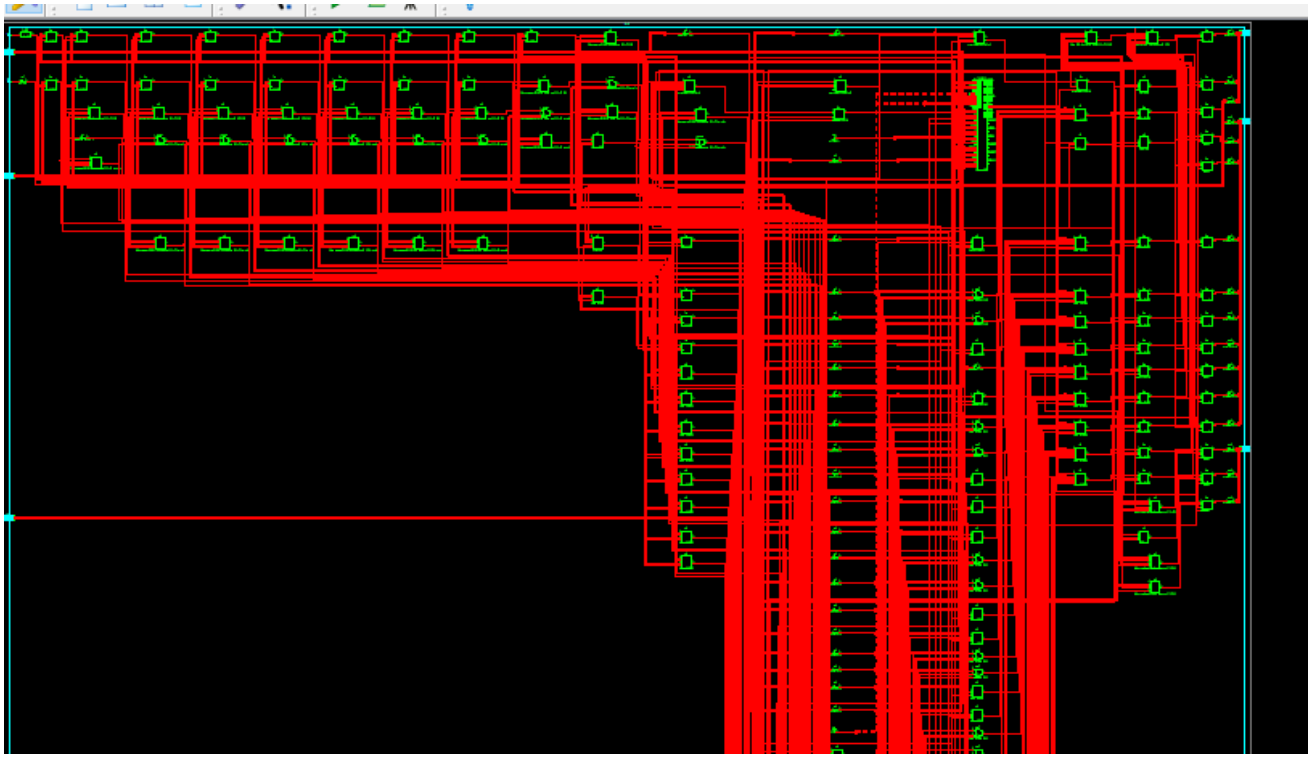
جدول بالا کد های متناظر با هر کدام از استیت های قطعه را مشخص می‌کند چون در صورت پروژه آمده است که این قطعه باید استیت خود را مشخص کند این قطعه نیز در هر استیتی که باشد آن را خروجی می‌دهد و کاربر را از استیتی که در آن قرار دارد آگاه می‌کند. خروجی های ۳ بیتی برای مشخص کردن استیت استفاده می‌شود.

code	Flag
000	None
001	Z(zero)
010	C(carry)
011	N(negative)
100	V(overflow)

جدول بالا فلگ ها را مشخص می‌کند. با توجه به اینکه این قطعه باید چند فلگ را بعد از انجام محاسبات مشخص کند به همین دلیل جدول بالا طراحی شده است و این قطعه در صورت وقوع هرکدام خروجی متناظر با آن را می دهد. حالت None برای زمانی است که هیچ کدام از فلگ ها فعال نمی‌باشند.



تصویر RTL ماژول



بخشی از شمای تکنولوژی ماژول

تشریح کد ALU

```
 8  library IEEE;
 9  use IEEE.STD_LOGIC_1164.ALL;
10  use IEEE.NUMERIC_STD.ALL;
11  use IEEE.STD_LOGIC_UNSIGNED.ALL;
12
13
14  entity ALU is
15  generic(DataWidth : integer := 10); -- Generic allows user to input with any length
16      Port (
17             input1 : in  STD_LOGIC_VECTOR (DataWidth -1 downto 0);
18             input2 : in  STD_LOGIC_VECTOR (DataWidth -1 downto 0);
19             operation : in  STD_LOGIC_VECTOR (3 downto 0);
20             clk : in STD_LOGIC;
21             RST : IN STD_LOGIC;
22             output : out  STD_LOGIC_VECTOR (DataWidth -1 downto 0);
23             scode : out STD_LOGIC_VECTOR (2 downto 0);
24             flag : out std_logic_vector(2 downto 0)
25             );
26  end ALU;
27
28  architecture Behavioral of ALU is
```

در ابتدا کد بخش اینتیتی کد می‌باشد و با توجه به خواست پروژه مبنی بر اینکه این قطعه بتواند ورودی با هر تعداد بیت را پردازش کند از generic استفاده شده است که به کاربر اجازه می دهد ورودی را با هر مقدار دلخواه به ماژول بدهد.

در ادامه هم ورودی ها و خروجی های ماژول مشخص شده اند.

```
30  -- Define States
31  type State is (ready, decode, in_process, writing, reading , in_output);
32  signal currentState : State;
33
34
```

در این قسمت همانطور که در صورت پروژه آمده است چندین استیت تعریف شده است که هرکدام وضعیت ماژول را مشخص میکند

- Ready

در این استیت ماشین آماده دریافت ورودی های جدید می‌باشد و ورودی ها را بافر میکند

- Decode

در این قسمت آپ کد آنالیز می‌شود و مشخص می‌شود که نوع دستور جیست و در ادامه به کدام استیت برود.

- In_process

در این بخش محاسبات منطقی و ریاضی انجام می‌شوند

- Writing

در این بخش در ورودی در حافظه نوشته می‌شود

- Reading

در این بخش حافظه خوانده و به خروجی داده می‌شود

- In_OUTPUT

در این استیت حافظه به خروجی داده می‌شود.

```
38
39  main : process(clk, RST)
40
41  -- Define additional variables
42  variable REG1,REG2 : std_logic_vector(DataWidth -1  downto 0); -- In this line I have definded two signal as two input registe
43  variable OUTREG : std_logic_vector(DataWidth -1  downto 0); -- In this line I have defined one signal as output register
44  variable temp : std_logic_vector(DataWidth -1  downto 0) := (others => '0');
45  variable temp2 : std_logic_vector(((2*DataWidth) -1)  downto 0) := (others => '0');
46
47  begin
```

با توجه به اینکه این ماژول حساس به کلاک می‌باشد یک پروسس برای آن تعرف شده است که به هر بار تغییر کلاک تغییرات اعمال شود این استیت به تغییرات ریست نیز حساس است.

در ادامه چندین متغییر تعرف کرده ام. دو خط اول ریجیستر های ورودی و خروجی را مشخص میکنند و برای مشخص کردن آنها از متغییر استفاده شده است زیرا این متغییر ها میتوانند مقدار ورودی و خروجی ها را در خود ذخیره کنند و از آنها به عنوان بافر استفاده شده است.

```
48
49      if (RST = '0') then
50         if rising_edge(clk) then
51               case currentState is
52
53                     when ready => -- in ready state machine is ready for new inputs ans commands
54                        REG1 := input1;
55                        REG2 := input2;
56                        currentState <= decode;
57
58                     when decode => -- in this state machine will figure out what to do
59                           if (operation = "1000") then -- checks if is write moode or not
60                                 scode <= "010";
61                                 currentState <= writing;
62
63                           elsif (operation = "1001") then -- checks if is read moode or not
64                                 scode <= "011";
65                                 currentState <= reading;
66
67                           elsif (operation > "1001") then -- if none turn back and resive new comm
68                                 scode <= "000";
69                                 currentState <= ready;
70
71                           else  -- logical and computional operations
72                                 scode <= "001";
73                                 currentState <= in_process;
74                           end if ;
75
```

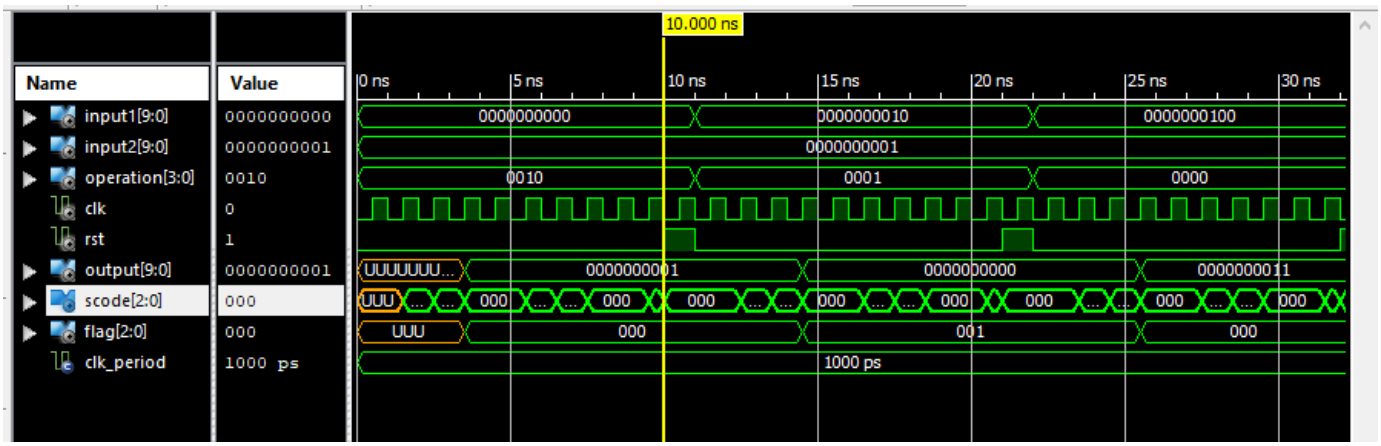
در ادامه یک ریســت آســکرون برای ماژول طراحی شــده اســت و همچنین این ماژول به لبه بالا رونده حساس می باشد.

ادامه توضیحات عملکرد کد در خود کد کامنت شده است.

تست بنچ

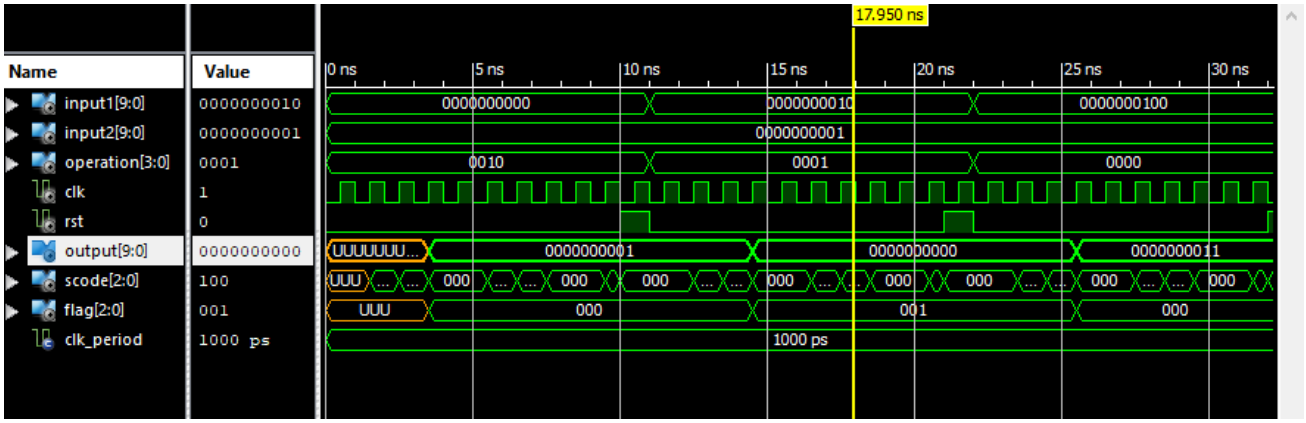
برای این ماژول یک تست بنچ طراحی شده است که بخش های مختلف آن را تست می‌کند.

```
82      -- ADD
83      input1 <= std_logic_vector(to_unsigned(0,10));
84      input2 <= std_logic_vector(to_unsigned(1,10));
85      operation <= "0010";
86      RST <= '0';
87      -- OUT = "0000000001"
88
89      wait for 10 ns;
90      RST <= '1';
91      wait for 1 ns;
```



همطور که مشخص است خروجی مجموع دو ورودی می باشد و درست است. همین طور که مشخص است زمانی که ریست آسکرون فعال می‌شود در همان زمان دستگاه ریست شده و از استیت اول شروع به کار می‌کند.

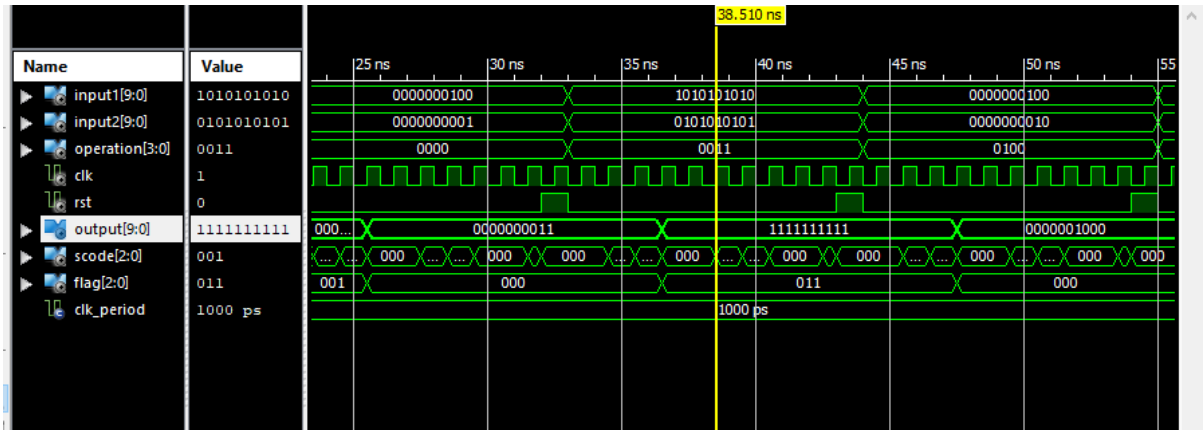
```
-- AND
input1 <= std_logic_vector(to_unsigned(2,10));
input2 <= std_logic_vector(to_unsigned(1,10));
operation <= "0001";
RST <= '0';
-- OUT = "0000000000"
```



همان طور ملاحظه می شود حاصل And ۱ و ۴ صفر شده است.

```
-- OR
input1 <= "1010101010";
input2 <= "0101010101";
operation <= "0011";
RST <= '0';
-- OUT = "1111111111"

wait for 10 ns;
RST <= '1';
wait for 1 ns;
```



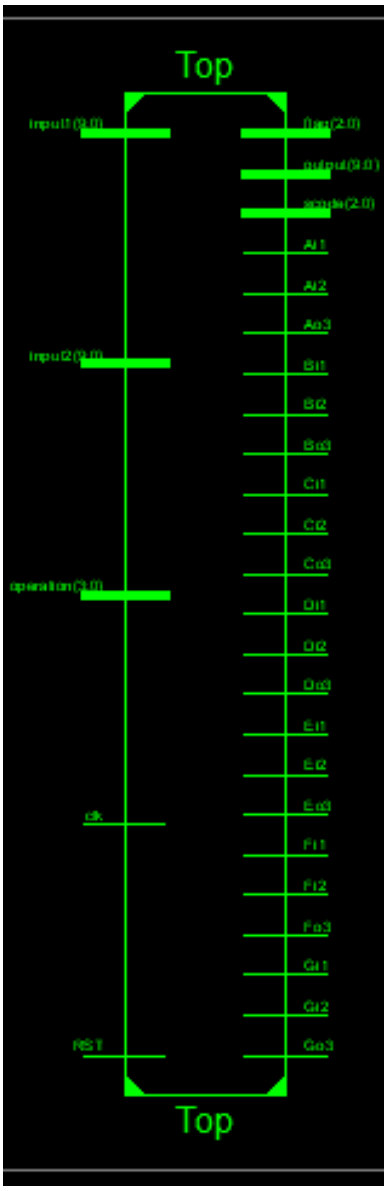
همان طور که مشخص است حاصل or این دو عدد باهم همه بیت هایشش ۱ می شود.

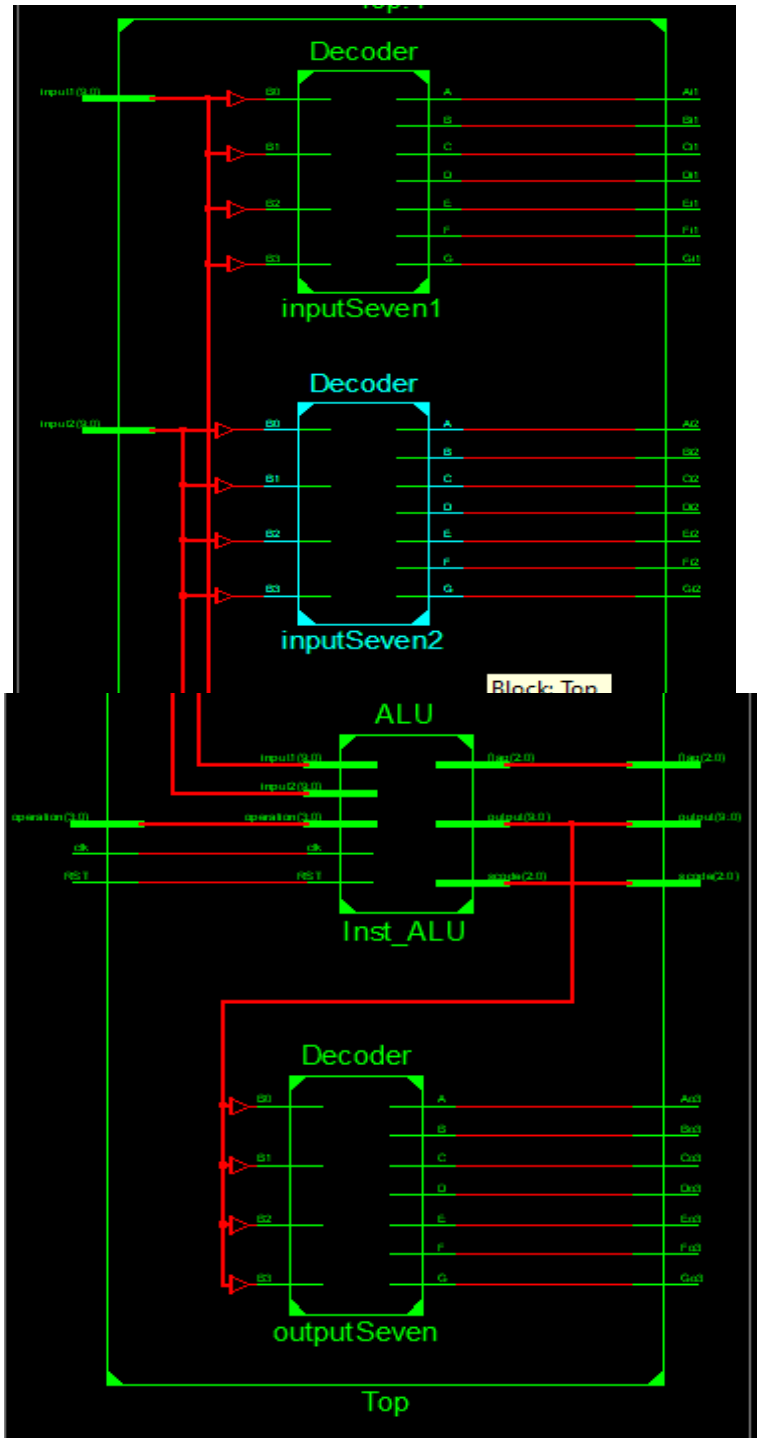
بقیه تست ها هم به همین ترتیب درست اند.

اضافه کردن بخش امتیازی

در بخش امتیازی یک فایل Top تعریف شـــده اســـت که هم شـــامل alu و هم شـــامل 7segment ها می‌باشد. و آنها را طبق صورت پروژه به هم وصل می‌کنیم.

RTLکد تاپ





نقشه RTL مدار هرکدام از ماژول ها در بخش های قبل نمایش داده شده است.

از دوتا دیکدر BCD به 7Segment برای ورودی و یکی هم برای خروجی در نظر گرفته شده است طبق خواست صورت پروژه.

```
generic(DataWidth : integer := 10); -- Generic allows user to input with any
    Port (
            -- For ALU
            input1 : in  STD_LOGIC_VECTOR (DataWidth -1 downto 0);
            input2 : in  STD_LOGIC_VECTOR (DataWidth -1 downto 0);
            operation : in  STD_LOGIC_VECTOR (3 downto 0);
            clk : in STD_LOGIC;
            RST : IN STD_LOGIC;
            output : out  STD_LOGIC_VECTOR (DataWidth -1 downto 0);
            scode : out STD_LOGIC_VECTOR (2 downto 0);
            flag : out std_logic_vector(2 downto 0);
            -- For seven segments:
            Ai1, Ai2, Ao3 : out std_logic:= '0';
            Bi1, Bi2, Bo3 : out std_logic:= '0';
            Ci1, Ci2, Co3 : out std_logic:= '0';
            Di1, Di2, Do3 : out std_logic:= '0';
            Ei1, Ei2, Eo3 : out std_logic:= '0';
            Fi1, Fi2, Fo3 : out std_logic:= '0';
            Gi1, Gi2, Go3 : out std_logic:= '0'
            );
end Top;
```

در این پروژه تمام ورودی های مورد نیاز ALU به علاوه خروجی های جدید که برای 7segment اســت نیز نوشته شده است.

هم کدام از A تا G را برای هرکدام از خروجی ها مینویسیم تا خروجی های دیکدر ها نمایش داده شود.

ابتدا دیکدر و ALU اضافه مکنیم به صورت کامپوننت

```
30  COMPONENT ALU
31     PORT(
32        input1 : IN std_logic_vector(9 downto 0);
33        input2 : IN std_logic_vector(9 downto 0);
34        operation : IN std_logic_vector(3 downto 0);
35        clk : IN std_logic;
36        RST : IN std_logic;
37        output : OUT std_logic_vector(9 downto 0);
38        scode : OUT std_logic_vector(2 downto 0);
39        flag : OUT std_logic_vector(2 downto 0)
40        );
41     END COMPONENT;
42
43  COMPONENT Decoder
44     PORT(
45        B0 : IN std_logic;
46        B1 : IN std_logic;
47        B2 : IN std_logic;
48        B3 : IN std_logic;
49        A : OUT std_logic;
50        B : OUT std_logic;
51        C : OUT std_logic;
52        D : OUT std_logic;
53        E : OUT std_logic;
54        F : OUT std_logic;
55        G : OUT std_logic
56        );
57     END COMPONENT;
58
```

```
59  signal REG1 : std_logic_vector(DataWidth -1  downto 0);
60  signal REG2 : std_logic_vector(DataWidth -1  downto 0); -- In this line I have definded two signal as two inpu
61  signal OUTREG : std_logic_vector(DataWidth -1  downto 0); -- In this line I have defined one signal as output
62
63  begin
64
65  REG1 <= input1;
66  REG2 <= input2;
67  output <= OUTREG;
68
69  Inst_ALU: ALU PORT MAP(
70        input1 => input1,
71        input2 => input2,
72        operation => operation,
73        clk => clk,
74        RST => RST,
75        output => OUTREG,
76        scode => scode,
77        flag => flag
78     );
79
80  inputSeven1: Decoder PORT MAP(
81        B0 => REG1(3),
82        B1 => REG1(2),
83        B2 => REG1(1),
84        B3 => REG1(0),
85        A => Ai1,
86        B => Bi1,
```

سیگنال های تعریف شده کارایی رجیستر ها را دارد و مقدار ورودی ور خروجی را در خود بافرر می‌کند.

درادامه ماژول ها پورت مپ می‌شوند.

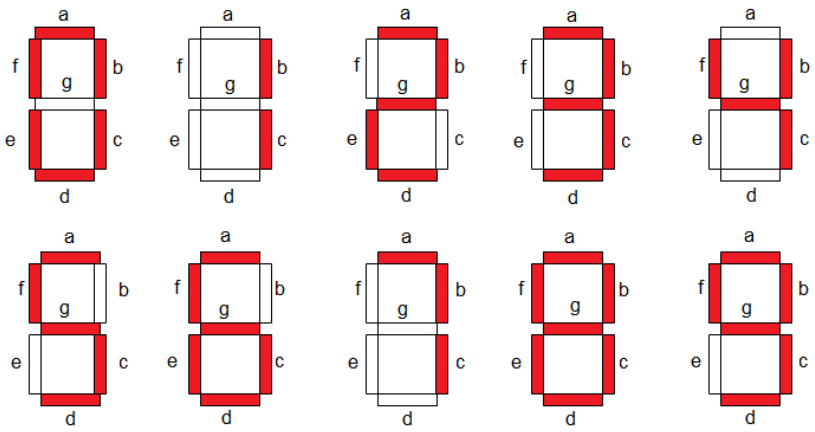
تست نچ بخش امتیازی و بررسی 7segment ها

تمام بخش های فایل مانند بقیه پروژه می باشد در این بخش ورودی مورد نظر داده می‌شود.

ورودی اول و صفر ۰ و ورودی دوم ۱ می‌باشد انتظار می‌رود که دیکدر های ورودی ۰ و ۱ را نشان بدهند.

و خروجی هم با توجه به اینکه از درستور جمع استفاده شده است باید ۱ باشد. بعد از شبیه سازی خواهیم داشت:

```
124       wait for clk_period/2;
125       clk <= '1';
126       wait for clk_period/2;
127    end process;
128
129
130    -- Stimulus process
131    stim_proc: process
132    begin
133       -- ADD
134       input1 <= std_logic_vector(to_unsigned(0,10));
135       input2 <= std_logic_vector(to_unsigned(1,10));
136       operation <= "0010";
137       RST <= '0';
138       -- OUT = "0000000001"
139
140
141       wait;
142    end process;
143
144 END;
```



همین طور که مشخص است خروجی های 7segment ها درست می‌باشند.