

به نام خدا

گزارش پروژه ی (ALU)

استاد: دکتر فخر احمد

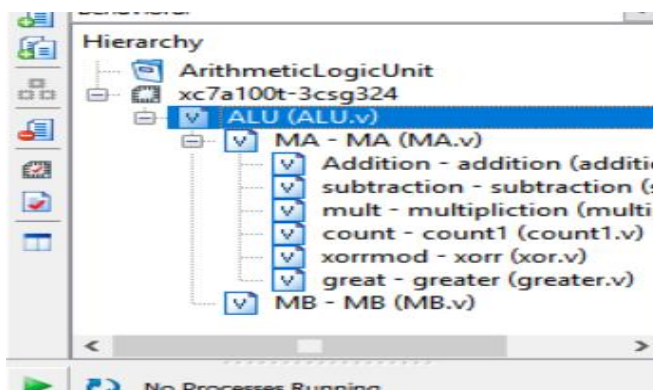
دستیاران: علیرضا خالقی – سینا وفادار

علی اثنی عشری-9732510

دومین پروژه شامل یک ماژول اصلی به نام (ALU.V) با دو زیرماژول (MA) و (MB) می باشد که زیر ماژول (MA) خود شامل 6 زیر ماژول با نام های زیر است:

- Addition (1)
- Subtraction (2)
- Multiplication (3)
- Count1 (4)
- Xorr (5)
- Greater (6)

مطابق شکل زیر:



ماژول اصلی (ALU) شامل موارد زیر است:

ورودی ها:

- Control bit (one bit) (1)
- Number1(5 bit) (2)
- Number2(5 bit) (3)
- Printout(6bit) (4)
- Clock (5)
- Reset (6)

خروجی ها:(register)

Balancebit (1
equalityBit (2
conclusion(32bit) (3

سیم (wire):

balance_w1 (1
equal_w1 (2
wire1(32bit) (3
balance_w2 (4
equal_w2 (5
wire2(32bit) (6

نکته :

از سیم‌ها (wire) برای instance MA or MB استفاده میشود.

توجه:

سیم‌هایی که شامل عدد یک در انتهای آن است برای MA و سیم‌هایی که شامل عدد دو در انتهای آن است برای MB مورد استفاده قرار میگیرد.

سپس MA و MB را شبیه‌سازی کرده (instance) و در top module (ALU.V) قرار میدهیم و متناسب با ورودی‌ها و خروجی‌های هر ماژول شبیه‌سازی شده، سیم‌های تعریف شده‌ی فوق را به آن‌ها متصل میکنیم.

مطابق شکل زیر:

```
// Instantiate the module
MA MA (
    .Number1(Number1),
    .Number2(Number2),
    .printout(printout),
    .balancebit(balance_w1),
    .equalityBit(equal_w1),
    .conclusion(wire1)
);

// Instantiate the module
MB MB (
    .allfunc(Number1),
    .decfunc(Number2),
    .balancebit(balance_w2),
    .equalityBit(equal_w2),
    .conclusion(wire2),
);
```

در مرحله ی بعد , در یک **always block** حساس به لبه ی مثبت **clock** ابتدا **reset** بررسی میشود که اگر برابر **یک** باشد همه ی خروجی های تعریف شده در بالای صفحه برابر **صفر** قرار میگیرد. مطابق شکل زیر :

```
er\ArithmeticLogicUnit\ArithmeticLogicUnit.xise - [ALU.v*]
Is Window Layout Help
[Icons]
65
66     always @( posedge clock)
67         begin
68
69             // reset check
70             if ( reset == 1 )
71                 begin
72
73                     conclusion = 0;
74                     balancebit = 0;
75                     equalityBit = 0;
76                 end
77
```

در مرحله ی بعد , **controlbit** مورد بررسی قرار میگیرد که دو حالت اتفاق میافتد:

1. در صورت **صفر** بودن : سیم های **MA** به رجیستر های خروجی تعریف شده در بالا وصل میشوند.
2. در صورت **یک** بودن : سیم های **MB** به رجیستر های خروجی تعریف شده در بالا وصل میشوند.

مطابق شکل زیر:

```
//controlbit checked
|
// MA
else if ( controlBit == 0 )
    begin

        conclusion= wire1;
        balancebit = balance_w1;
        equalityBit = equal_w1;

    end

//MB
else if ( controlBit == 1 )
    begin
        conclusion = wire2;
        balancebit = balance_w2;
        equalityBit = equal_w2;
    end
end
```

پایان ماژول ALU

در زیر ماژول MA داریم :

ورودی ها:

Number1(5 bit) .1

Number2(5 bit) .2

Printout(6bit) .3

خروجی ها (register):

Balancebit (1

equalityBit (2

conclusion(32bit) (3

سیم (wire) :

مطابق شکل زیر:

```
35
36 // temp wires for each module
37 wire balance_w1;
38 wire [31:0] wire1;//thirty-two bit
39
40 wire balance_w2;
41 wire [31:0] wire2;//thirty-two bit
42
43 wire balance_w3;
44 wire [31:0] wire3;//thirty-two bit
45
46 wire balance_w4;
47 wire [31:0] wire4;//thirty-two bit
48
49 wire balance_w5;
50 wire [31:0] wire5;//thirty-two bit
51
52 wire balance_w6;
53 wire [31:0] wire6;//thirty-two bit
54
```

که wire ها عددی 32 bit و balance_w ها عددی تک بیتی میباشند.(در شکل بالا نیز به شکل کامل در آمده است)

که هرکدام از سیم ها برای زیرماژول های MA میباشد.

متغیر ها:

.1 k

.2 i

ابتدا در یک بلاک initial ($i = 0$) قرار میدهیم و سپس شبیه ساز زیر ماژول هرکدام از 6 موارد را قرار داده و سیم های را متناسب با ورودی و خروجی ماژول ها به ان ها متصل میکنیم.

در مرحله ی بعد در یک بلاک **always** با حساس به **تمام موارد** (*) ابتدا بیت تساوی را بررسی میکنیم:
روش:

اگر عدد اول و دوم با هم **برابر** باشند بیت تساوی برابر **یک** و در غیر این صورت بیت تساوی **صفر** خواهد بود.
مطابق شکل زیر:

```
always @( * )
begin

    //checking the equality

    if ( Number1 == Number2 )//condition for equal two numbers
    begin

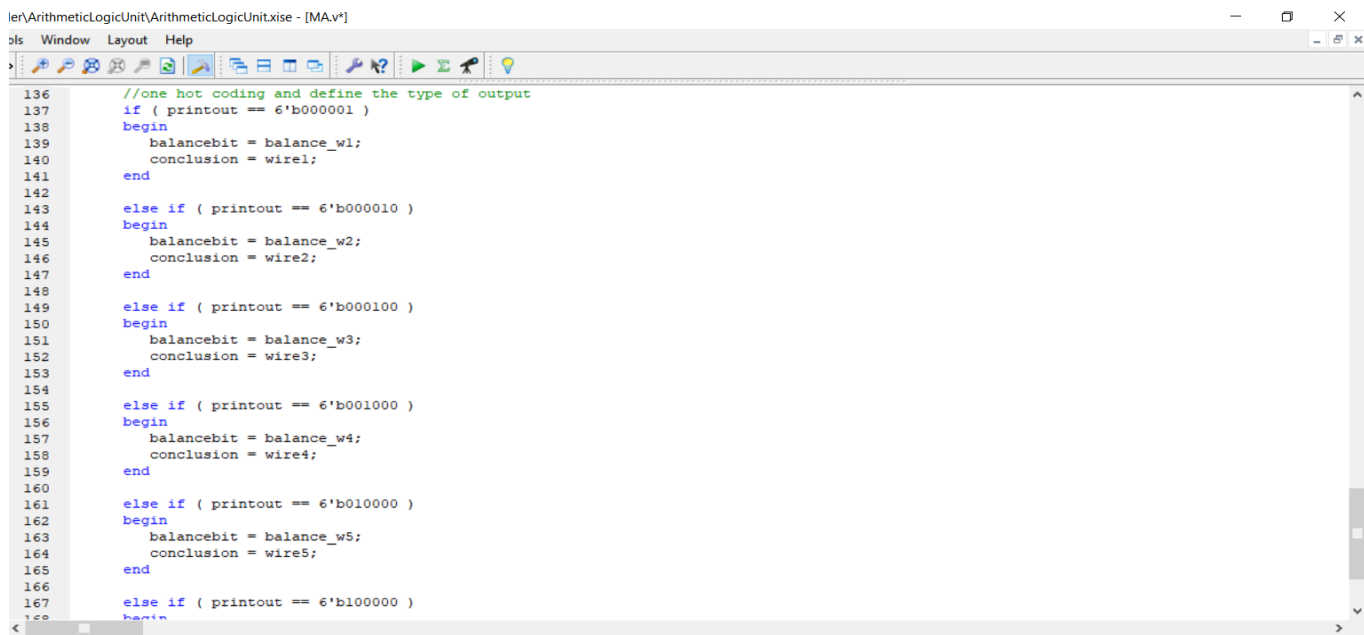
        equalitybit = 1;

    end
else
    begin

        equalitybit = 0;

    end
end
```

برای انجام هر کدام از 6 محاسبات ذکر شده در صورت سوال با توجه به **printout** که نوع محاسبات مورد نظر را مشخص میکند
سیم های مورد نیاز را به **conclusion** و **balancebit** وصل میکنیم. مطابق شکل زیر:



```
136 //one hot coding and define the type of output
137 if ( printout == 6'b000001 )
138 begin
139     balancebit = balance_w1;
140     conclusion = wire1;
141 end
142
143 else if ( printout == 6'b000010 )
144 begin
145     balancebit = balance_w2;
146     conclusion = wire2;
147 end
148
149 else if ( printout == 6'b000100 )
150 begin
151     balancebit = balance_w3;
152     conclusion = wire3;
153 end
154
155 else if ( printout == 6'b001000 )
156 begin
157     balancebit = balance_w4;
158     conclusion = wire4;
159 end
160
161 else if ( printout == 6'b010000 )
162 begin
163     balancebit = balance_w5;
164     conclusion = wire5;
165 end
166
167 else if ( printout == 6'b100000 )
168 begin
```

پایان مازول MA

توجه :

ذره‌رکدام از زیرماژول های ماژول MA با نام های :

- 1. addition
- 2. subtraction
- 3. multiplication
- 4. count1
- 5. xorr
- 6. greater

ورودی ها:

- 1. Number1(5 bit)
- 2. Number2(5 bit)
- 3. Printout(6bit)

خروجی ها (register):

- 1 Balancebit
- 2 conclusion(32bit)

و تقریباً هرکدام از آن ها شامل دو شمارنده برای حلقه ها (k) و (i) و یک متغیر موقت (temp) برای ذخیره محاسبات قبل از چاپ خروجی (conclusion) می باشد.

به عنوان مثال نحوه ی کارکرد ماژول جمع:

ماژول جمع (addition):

1. یک always block حساس به ورودی های داده شده داریم که ابتدا onehotcoding بررسی میشود که در اینجا با printout نامگذاری شده است.
 2. اگر شرط درست بود وارد if شده و دو عدد ورودی باهم جمع شده و در متغیر موقتی که در اینجا temp نام دارد ذخیره میکنیم.
 3. توجه : در قسمت دوم چون بیشترین تعداد بیت ممکن برای جمع دو عدد 5 بیتی , 6 بیت است در نتیجه متغیر موقتی ما 6 بیت حافظه دارد.
 4. در مرحله ی بعد با یک forloop برای بررسی بیت توازن , تعداد یک ها را شمرده که دو حالت زیر برقرار میشود:
(*) اگر تعداد یک ها زوج باشد بیت توازن که در اینجا balancebit نام دارد برابر یک و در غیر این صورت صفر خواهد شد.
 5. sign extend : چون conclusion 32 بیت است پس پر ارزش ترین بیت متغیر موقتی را تا آخر conclusion تکرار میکنیم.
- شکل زیر بخشی از عملکرد ماژول addition است :

```
neticLogicUnit\ArithmeticLogicUnit.xise - [addition.v*]  
dow Layout Help  
always @ ( Number1 or Number2 or printout)  
begin  
    //onehotcoding checked  
    if ( printout == 6'b000001 )  
    begin  
        temp = Number1 + Number2;  
        //count numbers of 1  
        for ( k = 0 ; k < 6 ; k = k + 1 )  
        begin  
            if ( temp[k] == 1'b1 )  
            begin  
                i = i + 1;  
            end  
        end//end for  
    if ( i % 2 == 0 )  
    begin  
        balancebit = 1'b1;//even  
    end  
    else  
    begin  
        balancebit = 1'b0;//odd  
    end//end else  
end
```

پایان مازول addition

در زیرماژول MB داریم:

هدف :

تبدیل یک عدد decimal به floating point تحت استاندارد IEEE754 صورت میگیرد.

ورودی ها :

Allfunc(5bit) .1

Decfunc(5bit) .2

خروجی ها (register):

Balancebit (1

equalityBit (2

conclusion(32bit) (3

متغير ها:

counter1 .1

counter2 .2

```
sc(shiftcount) .3
```

count .4

رجیستر و متغیرهای موقتی:

mantissa(9 bit) .1

temporary(10bit) .2

neticLogicUnit\ArithmeticLogicUnit.xise - [MB.v]

Window Layout Help

```
// Tool versions:
```

```
// Description:
```

//

```
// Dependencies:
```

```
//
// Derivation:
```

```
// Revision:
// Revision 0.01 - File Created
```

```
// Revision 0.01 - File
// Additional Comments:
```

```
// Additional comments:
//
```

////////////////////

```
module MB(allfunc, decfunc, balancebit, equalityBit, conclusion );
```

```
//input
```

```
input [4:0] allfunc;
```

```
input [4:0] decfunc;//decimal function
```

```
//output
```

```
output reg balancebit;
```

```
output reg equalityBit;
```

```
output reg [31:0] conclusion;
```

```
//counter and variable
```

```
integer count;
```

```
integer counter1;
```

```
integer counter2;
```

integer

```
//reg
reg [8:0] mantissa;//9 bit
reg [9:0] temporary;//10 b
```

```
reg [9:0] temporary; // 10 bit
```

initial

ابتدا در یک بلاک initial متغیرهای موقتی را برابر صفر قرار میدهیم.

سپس در یک بلاک always حساس به متغیرهای ورودی بررسی میکنیم که به چه تعداد باید در بخش صحیح که در اینجا

Allfunc نام دارد باید جابه‌جا (shift) شویم تا...

سپس بیت اول conclusion که نشان دهنده ی علامت است را اگر مثبت باشد برابر صفر و اگر منفی باشد برابر یک قرار میدهیم.

```
// sign when we have positive(0)
```

```
conclusion[31] = 1'b0;
```

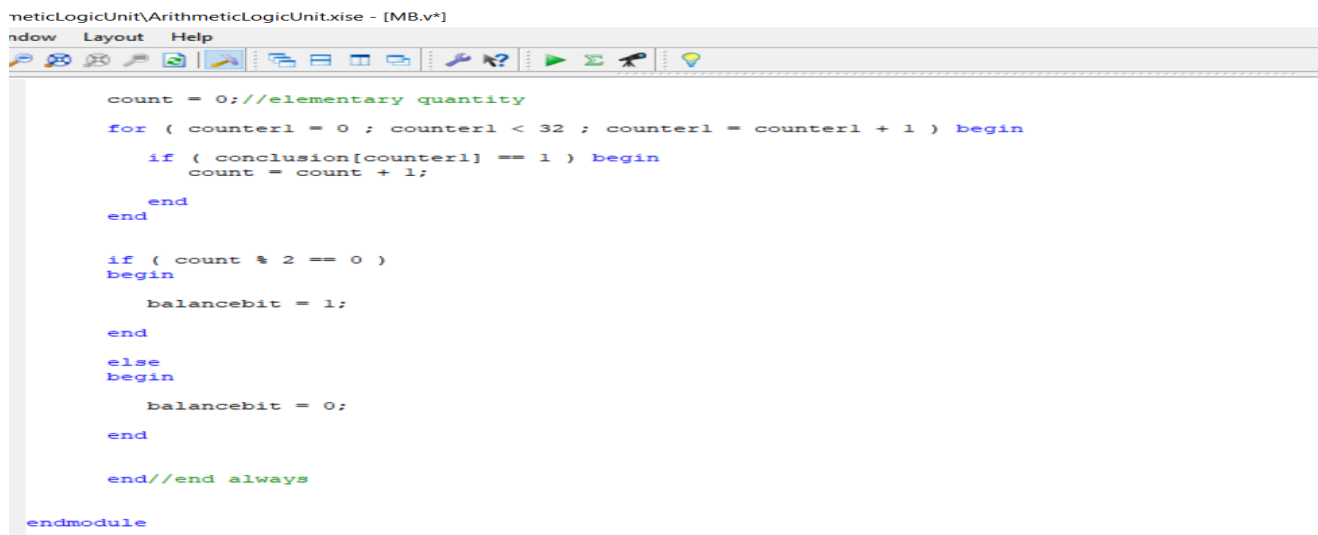
در شکل فوق علامت مثبت انتصاب شده است (بیت صفر)

می دانیم 8 بیت بعدی conclusion توان است پس تعداد ارقامی که به اندازه ی آن ها شیفت داده ایم را با 127 جمع کرده و در 8 بیت بعدی conclusion قرار میدهیم.

سپس بخش صحیح (allfunc) و اعشاری (decfunc) را ترکیب کرده و در متغیر موقتی قرار میدهیم و با یک while از قسمت اعشار تا آخر متغیر در mantissa که 9 بیت است قرار میدهیم و با یک حلقه ی دیگر بقیه ی بیت های mantissa را صفر میکنیم.

سپس با استفاده از یک if بیت توازن فرد و بیت تساوی را بررسی میکنیم.

شکل زیر نحوه ی عملکرد بیت توازن فرد است:



```
neticLogicUnit\ArithmeticLogicUnit.xise - [MB.v*]  
ndow  Layout  Help  
count = 0;//elementary quantity  
for ( counter1 = 0 ; counter1 < 32 ; counter1 = counter1 + 1 ) begin  
    if ( conclusion[counter1] == 1 ) begin  
        count = count + 1;  
    end  
end  
  
if ( count % 2 == 0 )  
begin  
    balancebit = 1;  
end  
else  
begin  
    balancebit = 0;  
end  
  
end//end always  
endmodule
```

پایان MB