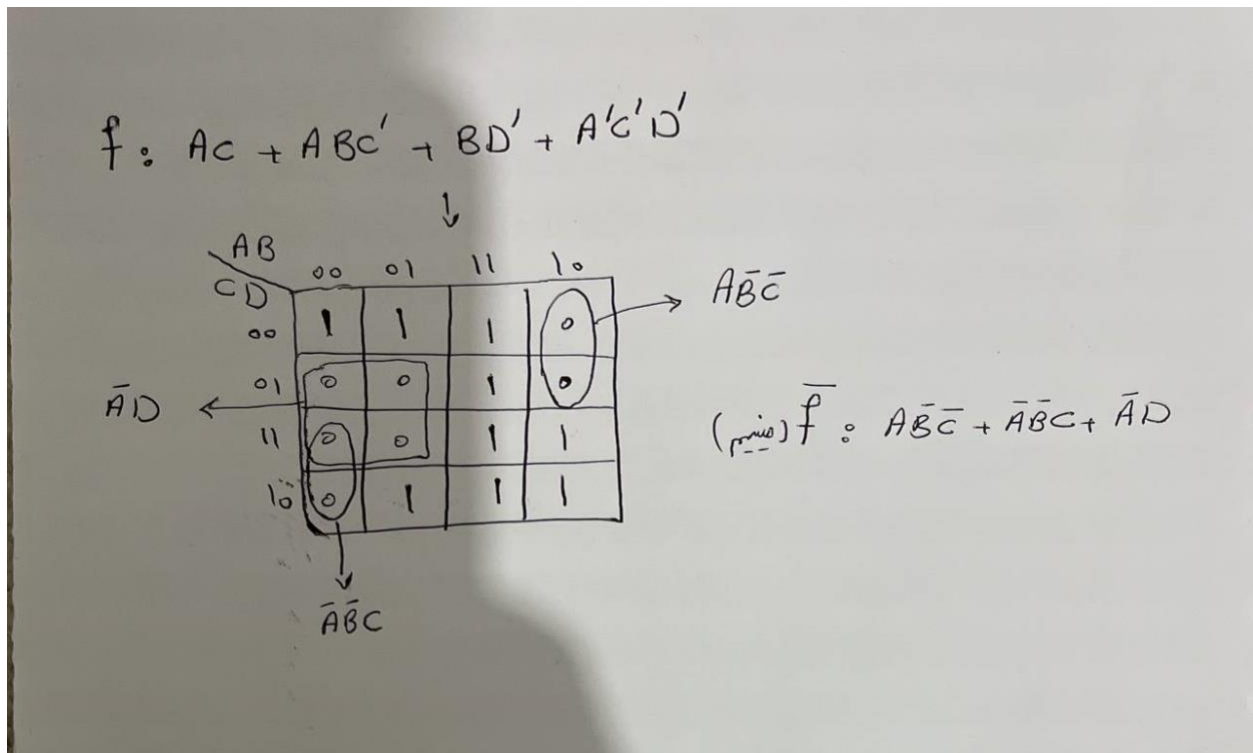


به نام خدا

آراس ولی زاده - ۴۰۱۲۴۳۰۹۵

سوال ۱ : برای پیاده سازی تابع داخل cmos نیاز به نقیض تابع و دوگان نقیض داریم که برای رسیدن به این هدف میتونیم اول بریم خود تابع رو با کارنومپ بهینه کنیم و بعد از اون نقیض کنیم تابع بهینه شده رو و ادامه ماجرا ، یا اینکه مستقیم با خود کارنومپ بهینه نقیض تابع رو بدست بیاریم تا بتونیم نقیض رو وارد n-network و دوگان نقیض رو وارد p - network کنیم که در ابتدا نحوه بهینه کردن تابع آورده شده



در ادامه کد و تست بنچ آورده شده که خروجی برنامه هم به شکل واضح در ترمینال آورده شده

کد برنامه که ترانزیستور ها پیاده سازی شده :

```
Valizadeh.Aras.401243095.Problem1.Testbench.v Valizadeh.Aras.401243095.Problem1.Module.v
Users > arasvalizadeh > Desktop > verilog > soal1 > problem 1 > Valizadeh.Aras.401243095.Problem1.Module.v
1 module modulesoale1(output out , input A , input not_A , input B , input not_B , input C , input not_C , input D , input
2 wire o1 , o2 ;
3 wire p1 , p2 , p3 , p4 , p5;
4 supply1 sup1;
5 supply0 sup0;
6 pmos p12(o1 , sup1 , not_A);
7 pmos p22(o1 , sup1 , not_B);
8 pmos p32(o1,sup1 , C);
9 pmos p42(o2,o1,not_A);
10 pmos p52(o2,o1,D);
11 pmos p62(out , o2 , not_C);
12 pmos p72(out , o2 , A);
13 pmos p82 (out , o2 , not_B);
14 nmos n12 (p1,sup0,A);
15 nmos n22 (p2,p1,not_C);
16 nmos n32(out , p2 , not_B);
17 nmos n42(p3,sup0 , D);
18 nmos n52(out,p3,not_A);
19 nmos n62(p4,sup0,C);
20 nmos n72(p5,p4,not_A);
21 nmos n82(out , p5 , not_B);
22 endmodule
```

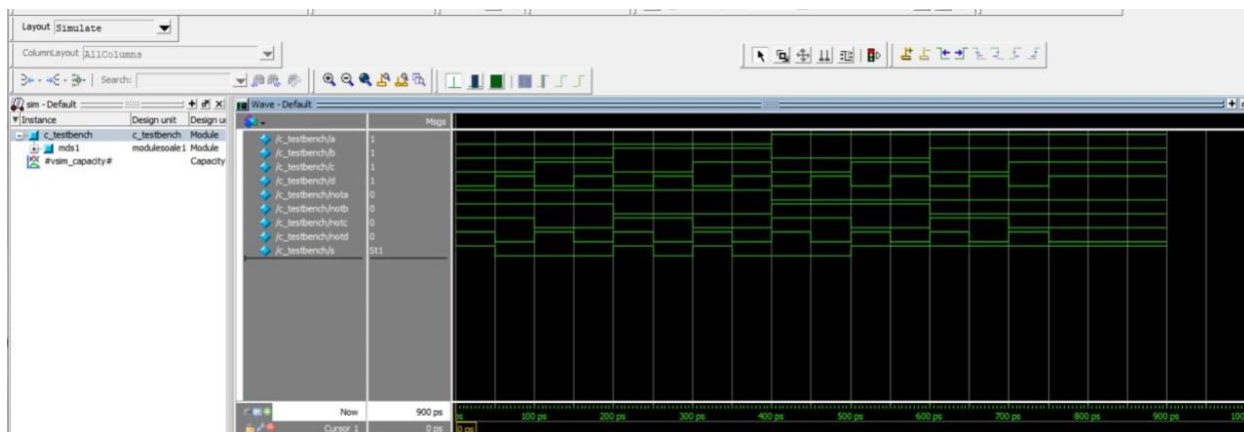
تست بنچ برنامه که تمام حالت های ممکن رو پوشش میده :

```
Get Started Valizadeh.Aras.401243095.Problem1.Testbench.v
Users > arasvalizadeh > Desktop > verilog > problem 1 > Valizadeh.Aras.401243095.Problem1.Testbench.v
1 module testbench ;
2 reg a , b , c , d , nota , notb , notc , notd;
3 integer i , j , k , m;
4 wire s ;
5 modulesoale1 mds1(s,a , nota ,b ,notb ,c , notc ,d , notd) ;
6 initial begin
7 #500;
8 a = 0 ;
9 nota = ~a;
10 b = 0 ;
11 notb = ~b ;
12 c = 0 ;
13 notc = ~c ;
14 d = 0 ;
15 notd = ~d ;
16 #500;
17 $display("A : %d , B : %d , C : %d , D : %d --> %d",a,b,c,d,s);
18 #500;
19 a = 0 ;
20 nota = ~a;
21 b = 0 ;
22 notb = ~b ;
23 c = 0 ;
24 notc = ~c ;
25 d = 1 ;
26 notd = ~d ;
27 #500;
28 $display("A : %d , B : %d , C : %d , D : %d --> %d",a,b,c,d,s);
29 #500;
30 a = 0 ;
31 nota = ~a;
32 b = 0 ;
33 notb = ~b ;
34 c = 1 ;
```

خروجی برنامه در ترمینال برای اطمینان از صحت عملکرد کد :

```
Last login: Sun Jun 18 00:23:57 on ttys000
arasvalizadeh@Arass-MacBook-Pro problem 1 % iverilog Valizadeh.Aras.401243095.Problem1.Mo
arasvalizadeh@Arass-MacBook-Pro problem 1 % ./a.out
A : 0 , B : 0 , C : 0 , D : 0 --> 1
A : 0 , B : 0 , C : 0 , D : 1 --> 0
A : 0 , B : 0 , C : 1 , D : 0 --> 0
A : 0 , B : 0 , C : 1 , D : 1 --> 0
A : 0 , B : 1 , C : 0 , D : 0 --> 1
A : 0 , B : 1 , C : 0 , D : 1 --> 0
A : 0 , B : 1 , C : 1 , D : 0 --> 1
A : 0 , B : 1 , C : 1 , D : 1 --> 0
A : 1 , B : 0 , C : 0 , D : 0 --> 0
A : 1 , B : 0 , C : 0 , D : 1 --> 0
A : 1 , B : 0 , C : 1 , D : 0 --> 1
A : 1 , B : 0 , C : 1 , D : 1 --> 1
A : 1 , B : 1 , C : 0 , D : 0 --> 1
A : 1 , B : 1 , C : 0 , D : 1 --> 1
A : 1 , B : 1 , C : 1 , D : 0 --> 1
A : 1 , B : 1 , C : 1 , D : 1 --> 1
arasvalizadeh@Arass-MacBook-Pro problem 1 %
```

شکل موج :



سوال ۲: برای این سوال به دلیل استفاده از سیستم مکمل ۲ی برای نمایش اعداد و انجام عملیات جمع بر روی آنها طبعاً کار ساده تر از باقی سیستم ها هستش . برای پیاده سازی سوال ۲ عدد ۸ بیتی ورودی داریم که ۸ بیت خروجی و یک خروجی تشخیص دهنده اورفلو داریم . برای بدست آوردن هر بیت و carry تولید شده باید ۳ مولفه ۲ بیت ورودی و carry که از بیت قبلی تولید شده رو در نظر بگیریم و برای هر کدام یک کارنومپ ۳ ورودی داریم ، فقط برای بیت اول چون carry از بیت های قبلی نداریم این کارنومپ ۲ ورودی میشود که در ادامه آورده شده :

|       |   |   |
|-------|---|---|
| $a_0$ | 0 | 1 |
| $b_0$ | 0 | 1 |
|       | 1 | 0 |

 $\rightarrow S_0 = a_0 \bar{b}_0 + \bar{a}_0 b_0$

|       |   |   |
|-------|---|---|
| $a_0$ | 0 | 1 |
| $b_0$ | 0 | 0 |
|       | 0 | 1 |

 $\rightarrow C_0 = a_0 b_0$

|           |    |    |    |    |
|-----------|----|----|----|----|
| $a_n b_n$ | 00 | 01 | 11 | 10 |
| $C_{n-1}$ | 0  | 1  | 0  | 1  |
|           | 1  | 0  | 1  | 0  |

 $\rightarrow S_n = \bar{a}_n \bar{b}_n C_{n-1} + \bar{a}_n b_n \bar{C}_{n-1} + a_n b_n C_{n-1} + a_n \bar{b}_n \bar{C}_{n-1}$

|           |    |    |    |    |
|-----------|----|----|----|----|
| $a_n b_n$ | 00 | 01 | 11 | 10 |
| $C_{n-1}$ | 0  | 0  | 1  | 0  |
|           | 1  | 1  | 1  | 1  |

 $\rightarrow C_n = a_n b_n + a_n C_{n-1} + b_n C_{n-1}$

توضیح کد :

در ابتدا ورودی ها و خروجی ها مشخص میشه و در همون ابتدا not تمامی عبارت ها ساخته میشه ، تا در ادامه کد هر جا نیاز باشه استفاده بشه از شون :

```
Valizadeh.Aras.401243095.Problem2.Module.v x
Users > arasvalizadeh > Desktop > verilog > soal2 > problem2 > Valizadeh.Aras.401243095.Problem2.Module.v
1 module soal2(input[7:0] A , input[7:0] B , output[7:0] sum , output overflow);
2   wire not_a0,not_a1,not_a2,not_a3,not_a4,not_a5,not_a6,not_a7;
3   wire not_b0,not_b1,not_b2,not_b3,not_b4,not_b5,not_b6,not_b7;
4   not not0(not_a0,A[0]);
5   not not1(not_b0,B[0]);
6   not not2(not_b1,B[1]);
7   not not3(not_a1,A[1]);
8   not not4(not_a2,A[2]);
9   not not5(not_b2,B[2]);
10  not not6(not_b3,B[3]);
11  not not7(not_a3,A[3]);
12  not not8(not_a4,A[4]);
13  not not9(not_b4,B[4]);
14  not not10(not_b5,B[5]);
15  not not11(not_a5,A[5]);
16  not not12(not_a6,A[6]);
17  not not13(not_b6,B[6]);
18  not not14(not_b7,B[7]);
19  not not15(not_a7,A[7]);
```

در ادامه بیت اول رو پیاده سازی کردم با استفاده از توابع کارنومپ صفحه قبل :

```
21 wire ex1,ex2;
22 and and1(ex1 , A[0] , not_b0);
23 and and2(ex2 , not_a0, B[0]);
24 or or1(sum[0],ex1,ex2);
25 wire carry0;
26 and and3(carry0 , A[0] , B[0]);
27 wire not_carry0;
28 not not16(not_carry0,carry0);
29
30
31 wire w1,w3,w5,w7;
32 and and4(w1,not_a1,not_b1,carry0);
33 and and5(w3,not_a1,B[1],not_carry0);
34 and and6(w5,A[1],B[1],carry0);
35 and and7(w7,A[1],not_b1,not_carry0);
36 or or4(sum[1],w1,w3,w5,w7);
37 wire carry1;
38 wire e1,e2,e3;
39 and and8(e1,A[1],B[1]);
40 and and9(e2,A[1],carry0);
41 and and10(e3,B[1],carry0);
42 or or5(carry1,e1,e2,e3);
43 wire not_carry1;
44 not not17(not_carry1,carry1);
```

و در ادامه بیت های بعدی (که با توجه به یکسان بودن تابع و صرفا متفاوت بودن عدد های بیت ورودی یک نمونه از بیت ها رو برای مثال قرار میدم)

```
30
31 wire w1,w3,w5,w7;
32 and and4(w1,not_a1,not_b1,carry0);
33 and and5(w3,not_a1,B[1],not_carry0);
34 and and6(w5,A[1],B[1],carry0);
35 and and7(w7,A[1],not_b1,not_carry0);
36 or or4(sum[1],w1,w3,w5,w7);
37 wire carry1;
38 wire e1,e2,e3;
39 and and8(e1,A[1],B[1]);
40 and and9(e2,A[1],carry0);
41 and and10(e3,B[1],carry0);
42 or or5(carry1,e1,e2,e3);
43 wire not_carry1;
44 not not17(not_carry1,carry1);
45
```

و برای تشخیص اورفلو تابع زیر را پیاده سازی شده در کد :

$$\text{overflow} \quad f(a,b,s) = a \cdot b \cdot \bar{s} + \bar{a} \cdot \bar{b} \cdot s$$

```
wire w71,w72;
and and66(w71,A[7],B[7],not_s7);
and and67(w72,not_a7,not_b7,sum[7]);
or or17(overflow,w71,w72);
```

در قسمت تست بنچ همانطور که در صورت سوال ذکر شده بود که ۴ عمل جمع متفاوت با خاصیت هایی که ذکر شده بود صورت بگیرد و در تست بنچ زیر هر ۴ تست گفته شده به علاوه تست های بیشتر ، تست میشود و حاصل با تشخیص اتفاق اورفلو و حاصل جمع اعداد در ترمینال قرار دادم :



```

1 module testbench ;
2   reg [7:0]a;
3   reg [7:0]b;
4   wire [7:0]s;
5   wire over;
6   soale2 soale2_instance(a,b,s,over);
7   initial begin
8     a = 8'b10110010;
9     b = 8'b11101111;
10    #1000;
11    $display("%b +\n%b\n%b / over = %b\n",a,b,s,over);
12    a = 8'b10010010;
13    b = 8'b10000111;
14    #1000;
15    $display("%b +\n%b\n%b / over = %b\n",a,b,s,over);
16    a = 8'b00110010;
17    b = 8'b01101111;
18    #1000;
19    $display("%b +\n%b\n%b / over = %b\n",a,b,s,over);
20    a = 8'b00010110;
21    b = 8'b00101111;
22    #1000;
23    $display("%b +\n%b\n%b / over = %b\n",a,b,s,over);
24    a = 8'b11011010;
25    b = 8'b11001111;
26    #1000;
27    $display("%b +\n%b\n%b / over = %b\n",a,b,s,over);
28    a = 8'b10110010;
29    b = 8'b01000111;
30    #1000;
31    $display("%b +\n%b\n%b / over = %b\n",a,b,s,over);
32    a = 8'b00110111;
33    b = 8'b01101100;
34    #1000;

```

جمع اول < ۷۸- + ۱۷- میشه که حاصل -۹۵ میشه و به درستی صورت میگیره  
 جمع دوم < ۱۱۰- + ۱۲۱- میشه که حاصل و اورفلو به درستی تشخیص داده میشه  
 جمع سوم < ۵۰ + ۱۱۱ میشه که حاصل و اورفلو به درستی تشخیص داده میشه  
 جمع چهارم < ۲۲ + ۴۷ که حاصل ۶۹ میشه و به درستی صورت میگیره :

```

Last login: Sun Jun 18 03:18:30 on ttys000
arasvalizadeh@Arass-MacBook-Pro: problem2 % iverilog Valizadeh.Aras.401243095.Pr
arasvalizadeh@Arass-MacBook-Pro: problem2 % ./a.out
10110010 +
11101111
10100001 / over = 0

10010010 +
10000111
00011001 / over = 1

00110010 +
01101111
10100001 / over = 1

00010110 +
00101111
01000101 / over = 0

11011010 +
11001111
10101001 / over = 0

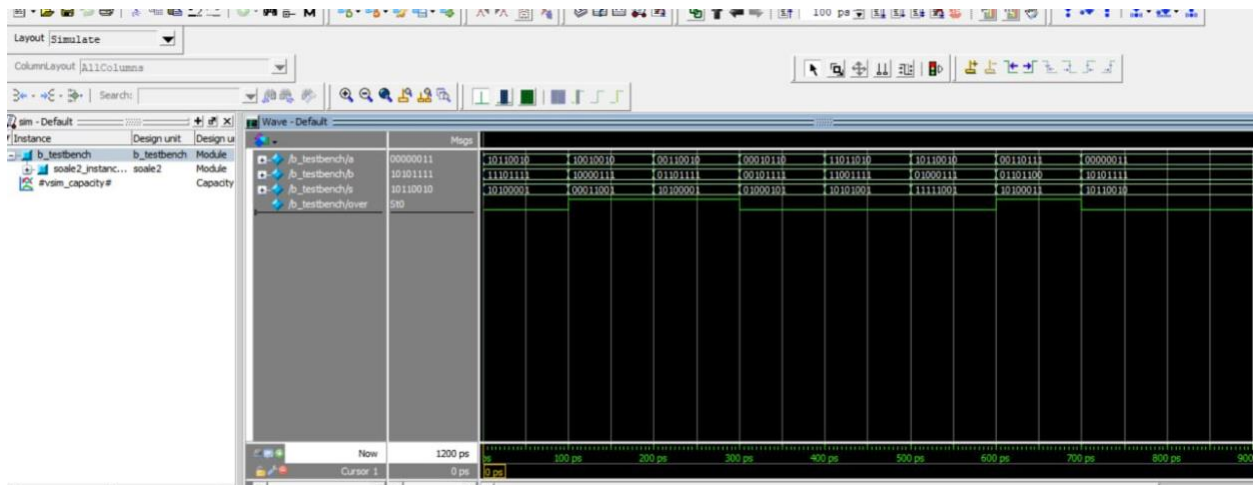
10110010 +
01000111
11111001 / over = 0

00110111 +
01101100
10100011 / over = 1

00000011 +
10101111
10110010 / over = 0
arasvalizadeh@Arass-MacBook-Pro: problem2 %

```

شكل موج :





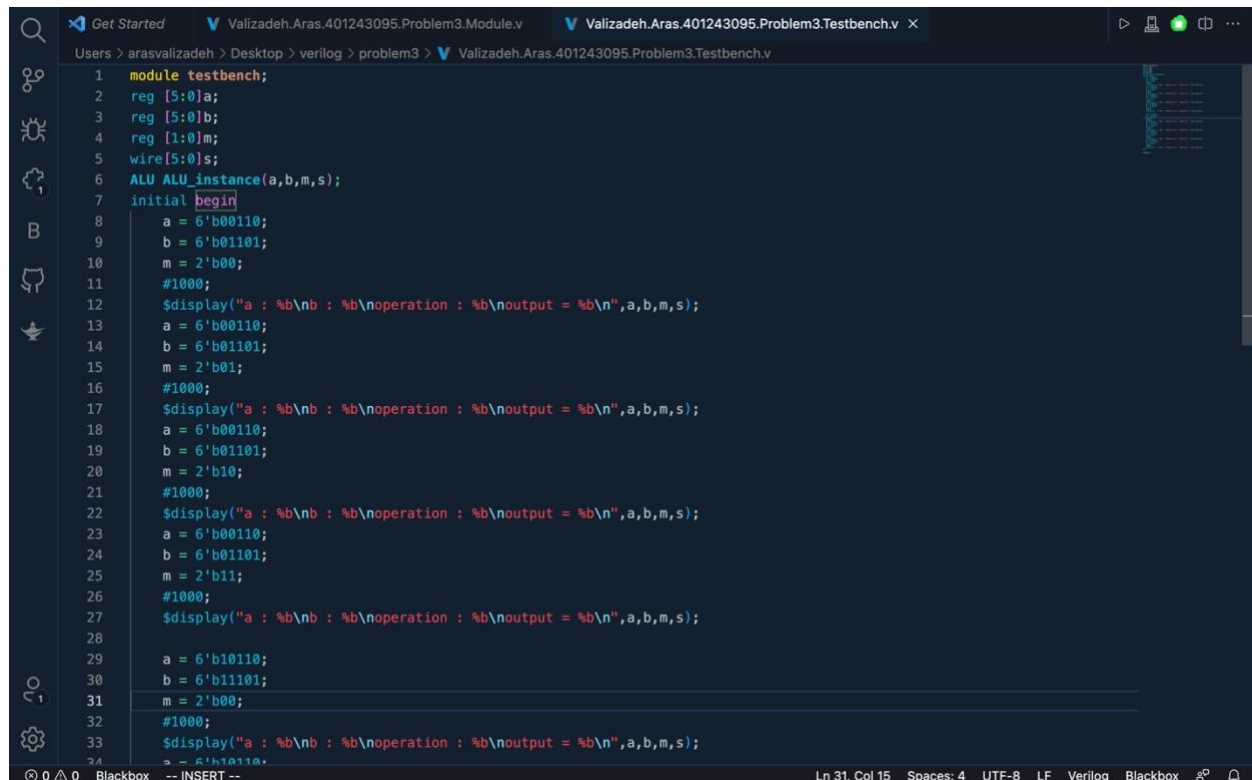
سوال سوم :

طبق خواسته سوال که هر بخش alu در یک ماژول جداگانه و با ساختار مشخص خواسته شده پیاده سازی بشه این کار صورت گرفت و در قسمت کد ماژول های ساخته شده در ابتدای alu یک نمونه ساخته و مورد استفاده قرار میگیرن تا به هدف نهایی طراحی alu برسیم  
ساختار کد به شکل زیر است :

```
Get Started Valizadeh.Aras.401243095.Problem3.Module.v X
Users > arasvalizadeh > Desktop > verilog > problem3 > Valizadeh.Aras.401243095.Problem3.Module.v
1 module ALU (input signed [5:0] A, input signed [5:0] B, input [1:0] Mode, output signed [5:0] Out);
2   wire [5:0] operation1, operation2, operation3, operation4;
3
4   ShiftAdd s1(.A(A), .B(B), .Out(operation1));
5   AddMultiply ad1(.A(A), .B(B), .Out(operation2));
6   Negative n1(.B(B), .Out(operation3));
7   Absolute ab1(.A(A), .B(B), .Out(operation4));
8
9   assign Out = (Mode == 2'b00) ? operation1 :
10               (Mode == 2'b01) ? operation2 :
11               (Mode == 2'b10) ? operation3 : operation4;
12 endmodule
13
14 module Negative ( input signed [5:0] B, output signed [5:0] Out);
15   assign Out = -B;
16 endmodule
17
18 module ShiftAdd ( input signed [5:0] A, input signed [5:0] B, output signed [5:0] Out);
19   assign Out = (A << 2) + (B >> 1);
20 endmodule
21
22 module AddMultiply ( input signed [5:0] A, input signed [5:0] B, output signed [5:0] Out);
23   assign Out = A + (B + B + B) ;
24 endmodule
25
26 module Absolute ( input signed [5:0] A, input signed [5:0] B, output signed [5:0] Out);
27   assign Out = (A + A - B) > 0 ? (A + A - B) : -(A + A - B);
28 endmodule
29
```

که ۳ ورودی شامل ۲ عدد علامت دار ۶ بیتی به همراه ۲ بیت ورودی انتخاب عملگر و ۶ بیت خروجی که در نهایت با ساختار dataflow خروجی تمامی ماژول ها مشخص می شود .

در قسمت تست بنچ شامل ۸ تست با انتخاب عملگر ها و عدد های مثبت و منفی  
مختلف تا حالات متفاوت را پوشش دهد :

A screenshot of a Verilog testbench in an IDE. The code defines a module 'testbench' with three 5-bit registers 'a', 'b', and 'm'. It instantiates an 'ALU' block and uses an 'initial' block to set initial values and perform four operations: AND, OR, XOR, and NOT. Each operation is followed by a 1000ns delay and a display statement showing the inputs, operation, and output. The IDE interface includes a file explorer on the left, a toolbar at the top, and a status bar at the bottom showing 'Ln 31, Col 15' and 'Verilog Blackbox'.

```

Last login: Sun Jun 10 02:33:45 on tty001
arasvalizadeh@Aras-MacBook-Pro: problem3 % lverilog Valizadeh.Aras.401243095.Problem3.Module.v Valizadeh.Aras.401243095.Problem3.Testbench.v
arasvalizadeh@Aras-MacBook-Pro: problem3 % ./a.out
a : 000110
b : 001101
operation : 00
output = 011110

a : 000110
b : 001101
operation : 01
output = 101101

a : 000110
b : 001101
operation : 10
output = 110011

a : 000110
b : 001101
operation : 11
output = 000001

a : 010110
b : 011101
operation : 00
output = 100110

a : 010110
b : 011101
operation : 01
output = 101101

a : 000110
b : 011101
operation : 10
output = 100011

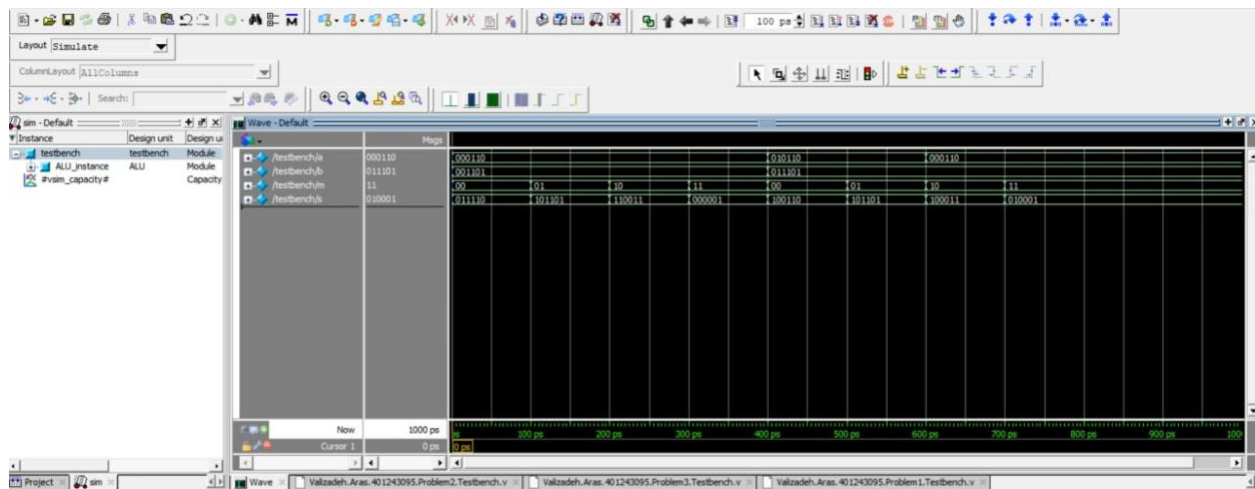
a : 000110
b : 011101
operation : 11
output = 010001

arasvalizadeh@Aras-MacBook-Pro: problem3 %

```

خروجی ها در ترمینال آورده شدن تا درستی برنامه تایید شود .

شکل موج :



پایان