HACETTEPE UNİVERSİTY COMPUTER SCİENCE DEPARTMENT

BBM 233 – LOGİCAL DESİGN LAB.

VERİLOG ASSİGNMENT 2

Name: Mert
Surname: Çökelek
ID: 21727082
Date: 9.11.2018
Subject: Implementing 1-bit, 4,bits Full Adder and Half Adder

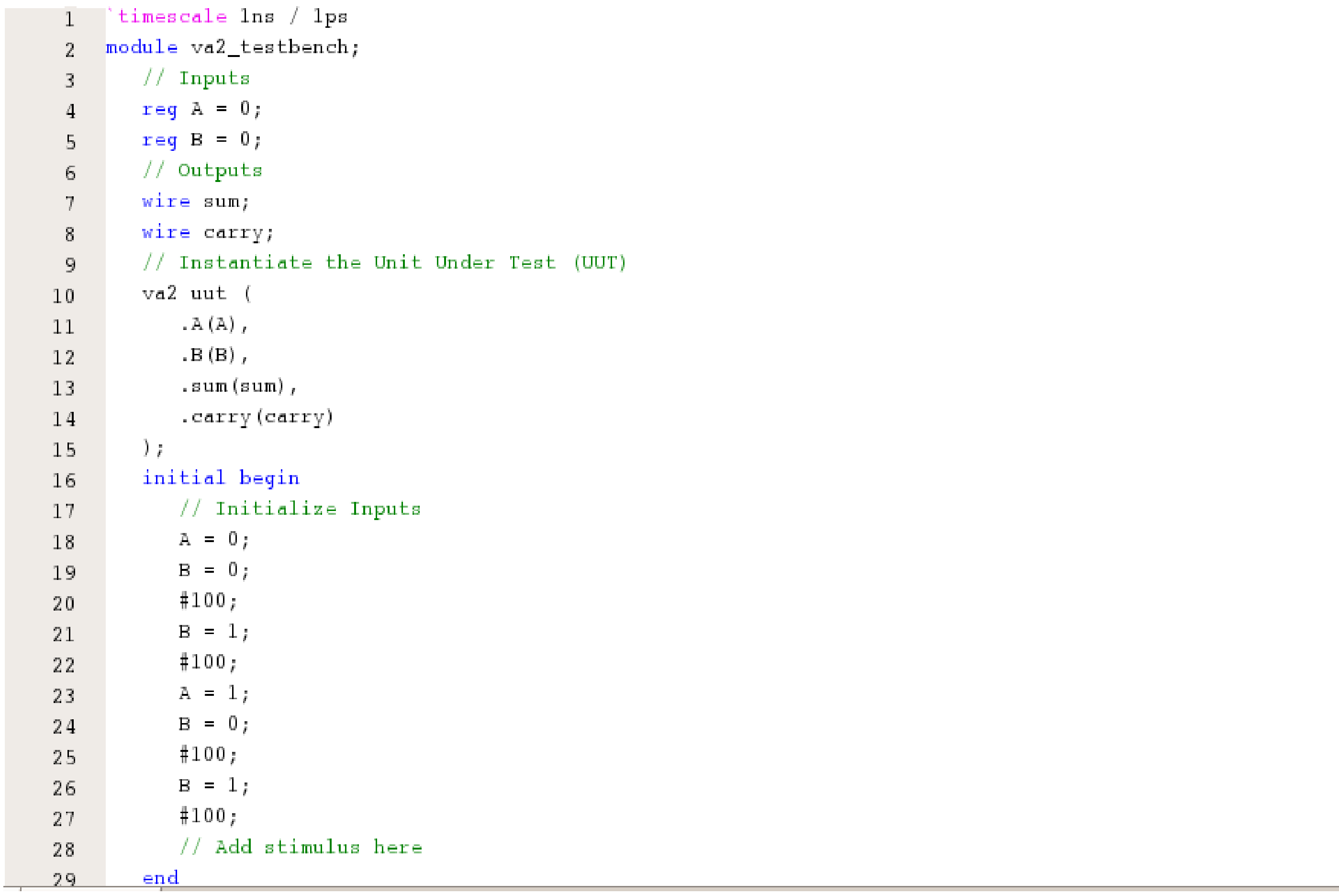
Part 1: Implementing a Half Adder:

Here is my verilog code and the output as waveform.

Halfadder.v

```
`timescale 1ns / 1ps

module half_adder (
  A,
  B,
  Sum,
  Carry);

 input  A;
 input  B;
 output Sum;
 output Carry;

 assign Sum   = A  ^  B  // xor
 assign o_carry = A & B;  // and
endmodule
```

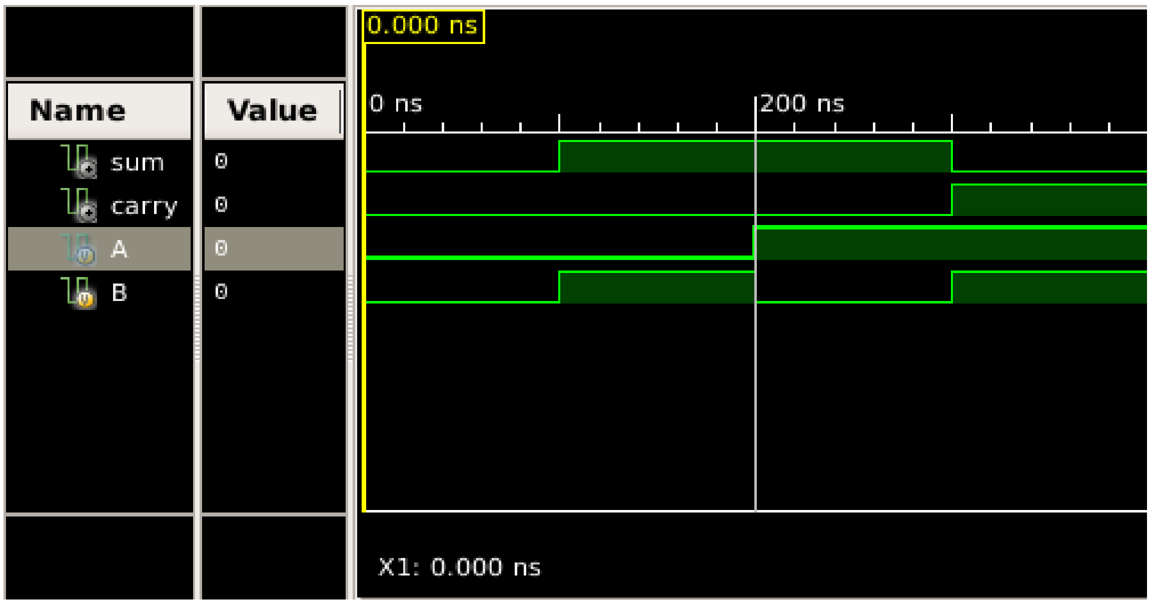
HalfAdderTestBench:



```verilog
`timescale 1ns / 1ps
module va2_testbench;
   // Inputs
   reg A = 0;
   reg B = 0;
   // Outputs
   wire sum;
   wire carry;
   // Instantiate the Unit Under Test (UUT)
   va2 uut (
      .A(A),
      .B(B),
      .sum(sum),
      .carry(carry)
   );
   initial begin
      // Initialize Inputs
      A = 0;
      B = 0;
      #100;
      B = 1;
      #100;
      A = 1;
      B = 0;
      #100;
      B = 1;
      #100;
      // Add stimulus here
   end
```

endmodule

And the waveform of halfAdder is:



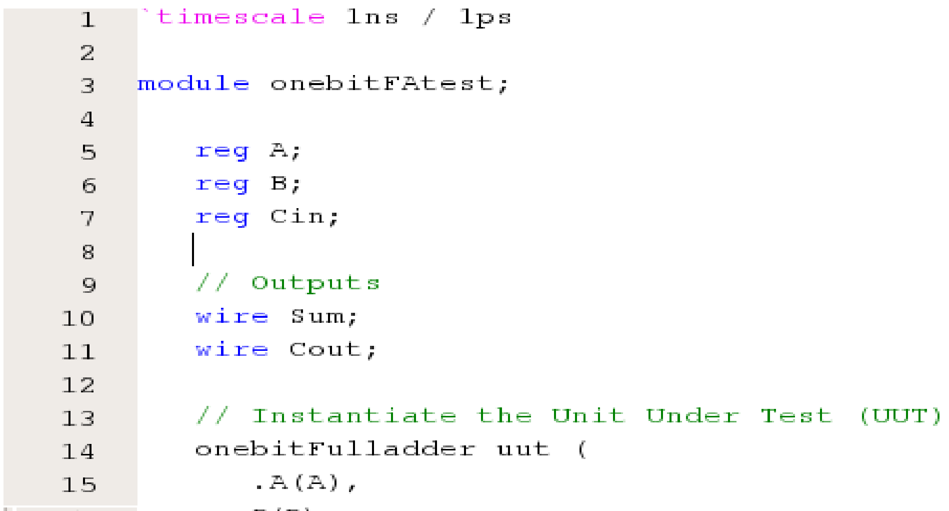
Part 2: Implementing a 1-Bit Full Adder:

Here is my verilog code for this part:

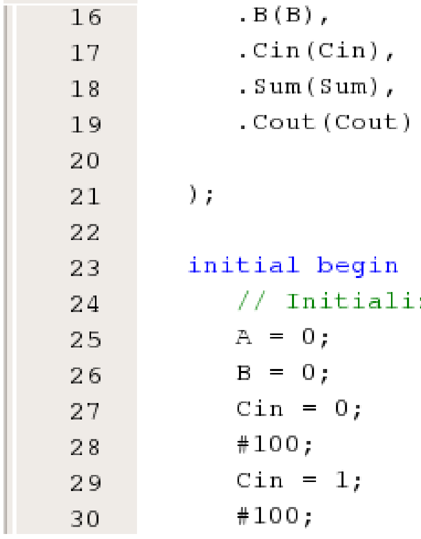
1- onebitFullAdder.v:

```
`timescale 1ns / 1ps

module onebitFulladder
(
     input A,
     input B,
     input Cin,
     output Sum,
     output Cout
    );
    assign {Cout, Sum} = Cin + A + B;
endmodule

```

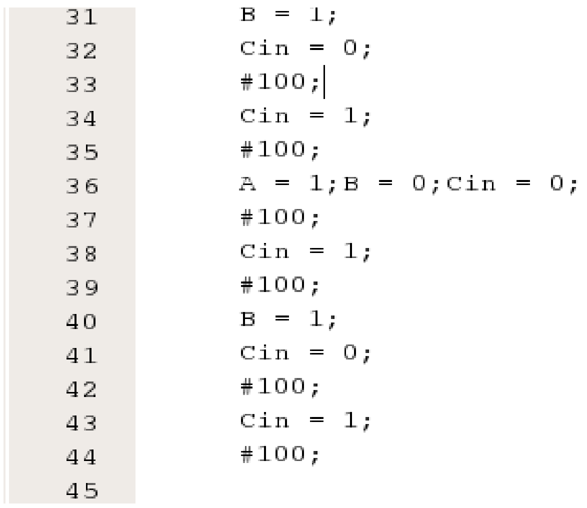
2- oneBitFAtest.v:



```verilog
`timescale 1ns / 1ps

module onebitFAtest;

   reg A;
   reg B;
   reg Cin;

   // Outputs
   wire Sum;
   wire Cout;

   // Instantiate the Unit Under Test (UUT)
   onebitFulladder uut (
      .A(A),
```



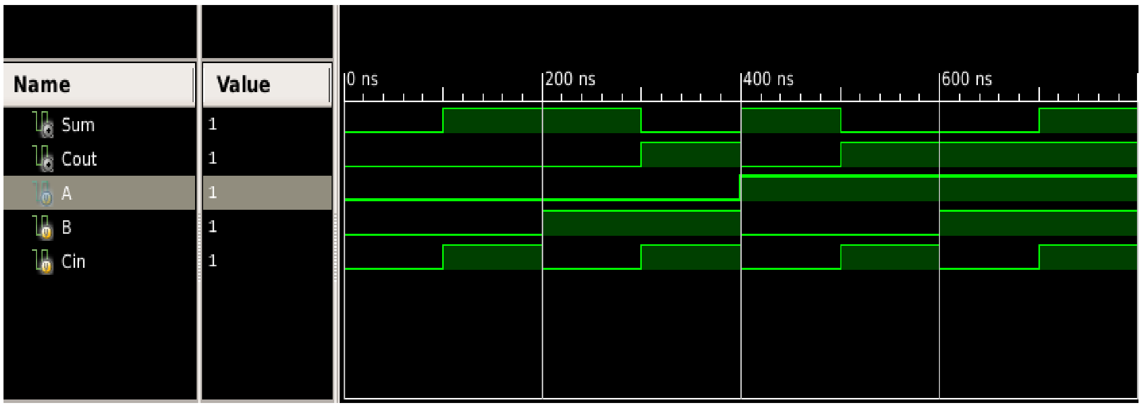
```verilog
     .B(B),
     .Cin(Cin),
     .Sum(Sum),
     .Cout(Cout)

  );

  initial begin
     // Initialize Inputs
     A = 0;
     B = 0;
     Cin = 0;
     #100;
     Cin = 1;
     #100;
```



```verilog
     B = 1;
     Cin = 0;
     #100;
     Cin = 1;
     #100;
     A = 1;B = 0;Cin = 0;
     #100;
     Cin = 1;
     #100;
     B = 1;
     Cin = 0;
     #100;
     Cin = 1;
     #100;

```

```verilog
        end
endmodule
```

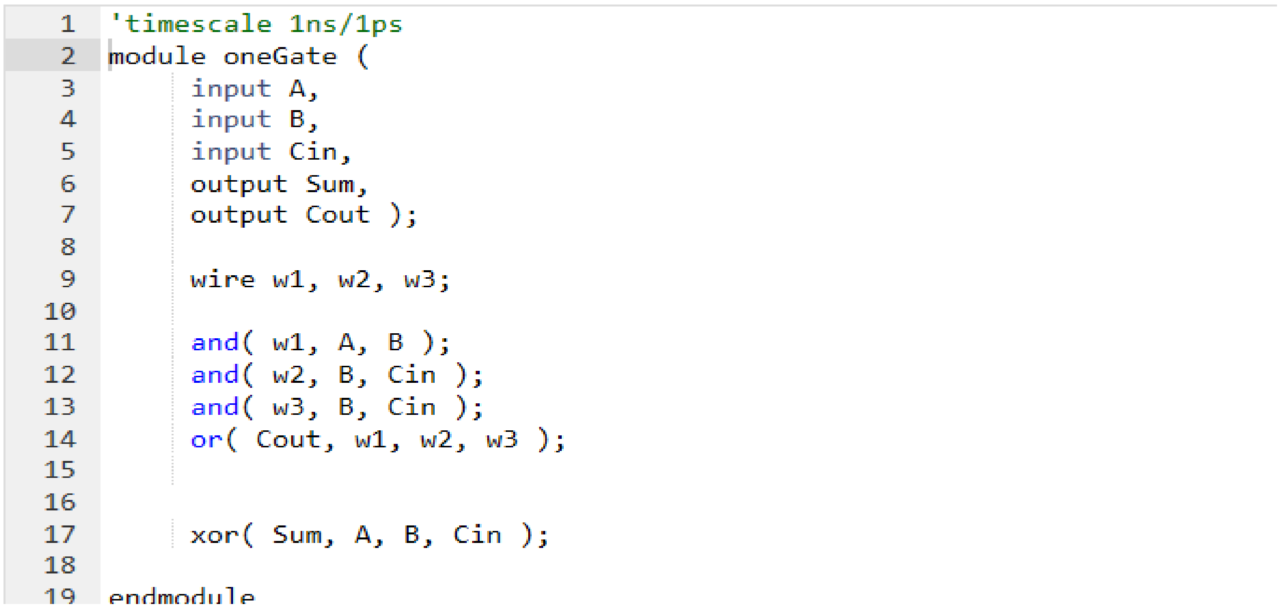
And the waveform for this code is:



Part 3: 4-Bits Full Adder:

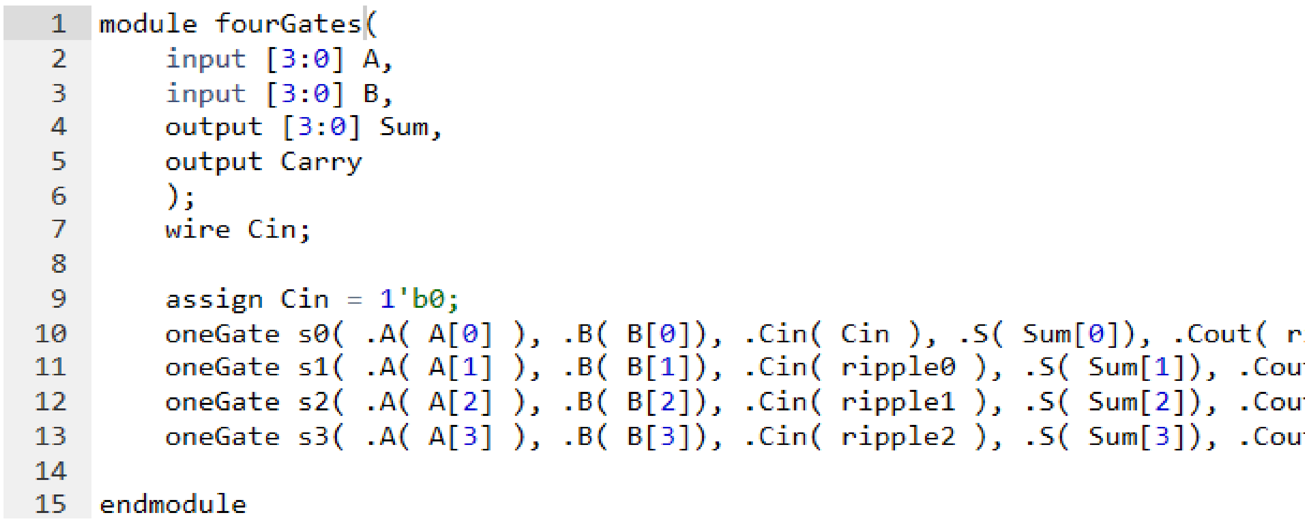
Here is my code and the output for this part:

For one Gate:



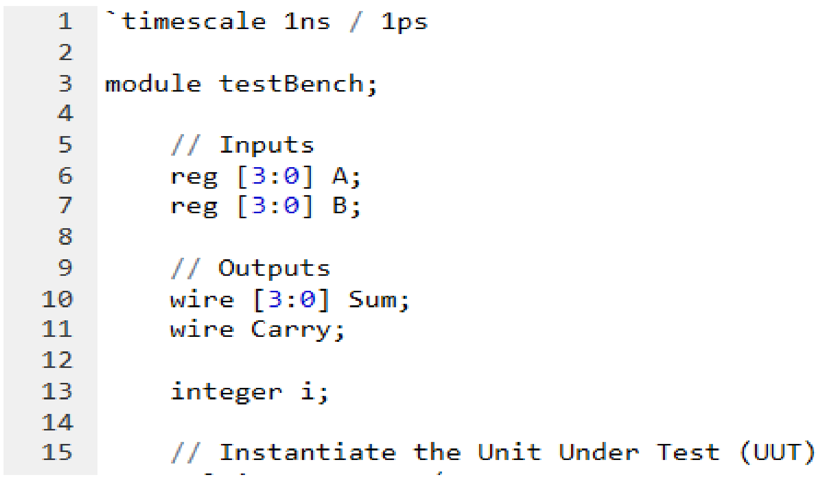
```
'timescale 1ns/1ps
module oneGate (
        input A,
        input B,
        input Cin,
        output Sum,
        output Cout );

        wire w1, w2, w3;

        and( w1, A, B );
        and( w2, B, Cin );
        and( w3, B, Cin );
        or( Cout, w1, w2, w3 );


        xor( Sum, A, B, Cin );

endmodule
```

for four gates:

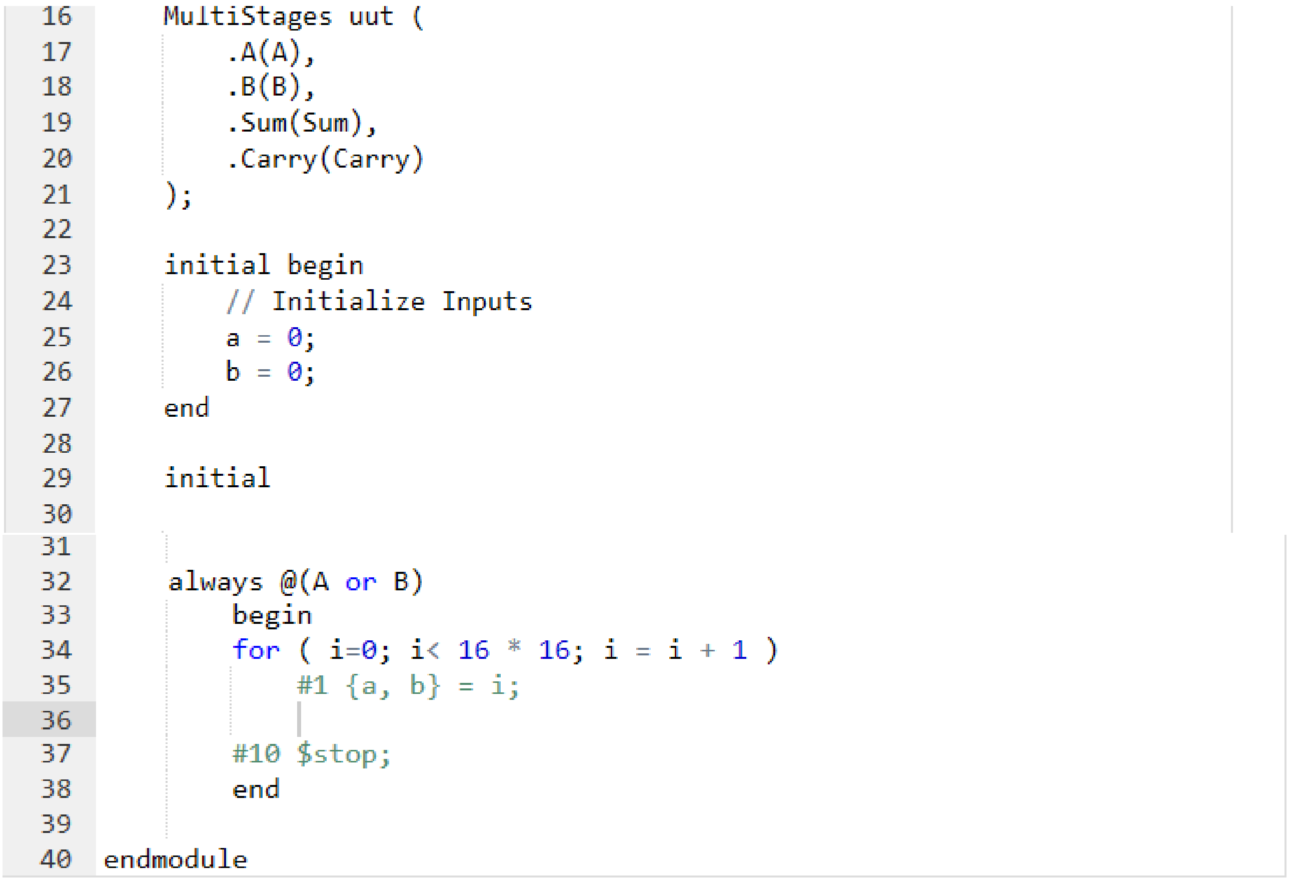


```
module fourGates(
    input [3:0] A,
    input [3:0] B,
    output [3:0] Sum,
    output Carry
    );
    wire Cin;

    assign Cin = 1'b0;
    oneGate s0( .A( A[0] ), .B( B[0]), .Cin( Cin ), .S( Sum[0]), .Cout( r
    oneGate s1( .A( A[1] ), .B( B[1]), .Cin( ripple0 ), .S( Sum[1]), .Cou
    oneGate s2( .A( A[2] ), .B( B[2]), .Cin( ripple1 ), .S( Sum[2]), .Cou
    oneGate s3( .A( A[3] ), .B( B[3]), .Cin( ripple2 ), .S( Sum[3]), .Cou

endmodule
```

For TestBench:



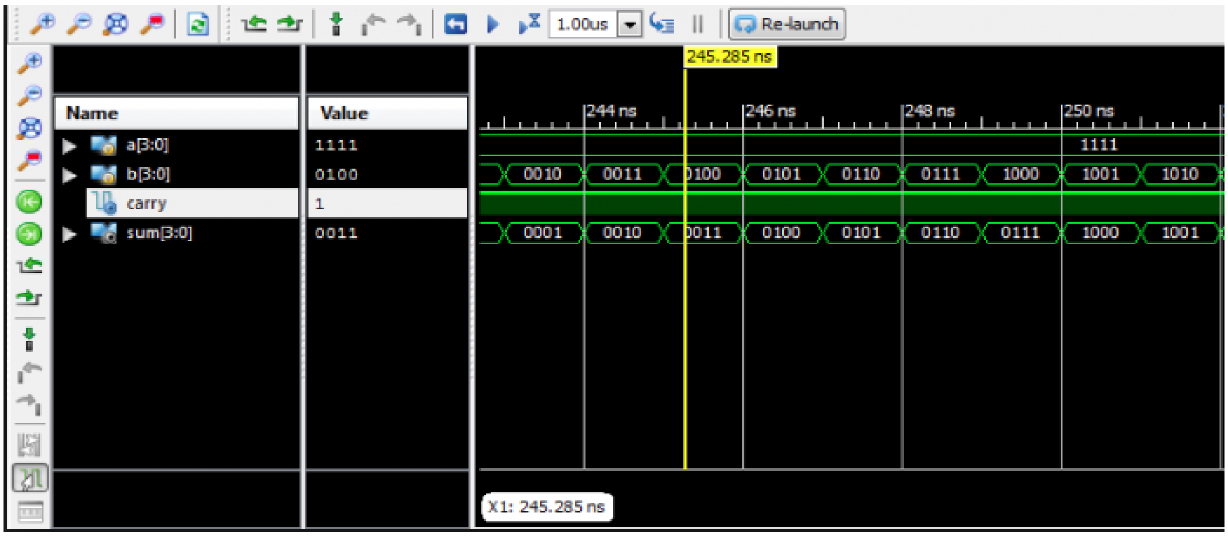
```
`timescale 1ns / 1ps

module testBench;

    // Inputs
    reg [3:0] A;
    reg [3:0] B;

    // Outputs
    wire [3:0] Sum;
    wire Carry;

    integer i;

    // Instantiate the Unit Under Test (UUT)
```



```verilog
	MultiStages uut (
		.A(A),
		.B(B),
		.Sum(Sum),
		.Carry(Carry)
	);

	initial begin
		// Initialize Inputs
		a = 0;
		b = 0;
	end

	initial

	
	always @(A or B)
		begin
		for ( i=0; i< 16 * 16; i = i + 1 )
			#1 {a, b} = i;
			|
		#10 $stop;
		end

endmodule
```

IMPORTANT NOTE: Despite I've compiled my code successfully, I could not get the waveform from ISE. Because it was appearing for milliseconds and disappearing. For hours of googling, I could not find a solution, so (unfortunately) I decided to put the waveform which I found on internet. But I think my code would give the same output.

I can delete this from my report, if I will have problem..



Name	Value
a[3:0]	1111
b[3:0]	0100
carry	1
sum[3:0]	0011

```
a[3:0]:   1111
b[3:0]:   0010 0011 0100 0101 0110 0111 1000 1001 1010
sum[3:0]: 0001 0010 0011 0100 0101 0110 0111 1000 1001
```