ASSIGNMENT 1

31.01.2022

—

Twisted
Srujana Vanka - 2020102005
Shreeya Singh - 2020102011

ADD MODULE

The below code is a basic unit of an n-bit adder.

a[i] and b[i] are the inputs given by the user, c[i] is the carry bit obtained from the previous sum, ie, a[i-1] and b[i-1].

We made use of 4 if-else statements to build the module.

The following are the cases to determine the bitwise sum of two numbers.

1- If all the input bits are 1, ie, a[i], b[i] and c[i] are 1, sum output and carry is also 1.

2- If any 2 of the input bits is 1, then the sum output = 0 and the carry bit is 1.

3- If only 1 of the input bits is 1 i.e a[i], b[i], or c[i] is 1, then the sum output bit will be 1, while the carry bit is 0.

4- If there are no 1s in the input bit, then both the output bit and the carry is 0.

Using this module 64 times in sequence from 0th bit to 63rd bit gives us the sum for two 64 bit numbers.

```
`timescale 1ns / 1ps
module adder(a,b,c,y,s);
input a,b,c;
output reg y,s;
always @ (a or b or c) begin
   if (a == 1'b1) begin
         if(b == 1'b1) begin
             if(c == 1'b1)begin
                 y=1'b1;
                 s=1'b1;
             end
             else begin
                 y=1'b0;
```

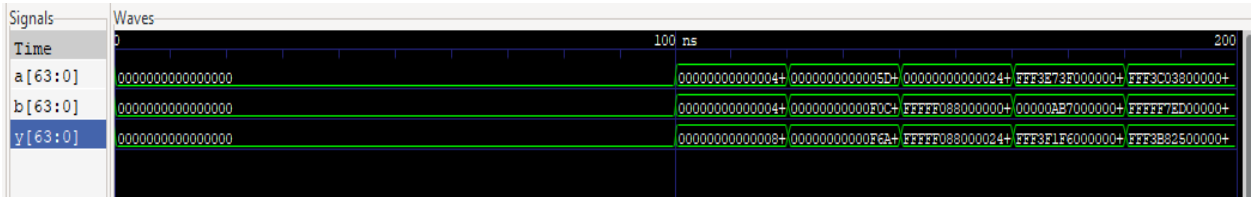
```
            s=1'b1;
        end
    end
    else begin
        if(c == 1'b1)begin
            y=1'b0;
            s=1'b1;
        end
        else begin
            y=1'b1;
            s=1'b0;
        end
    end

 end
else begin
    if(b == 1'b1) begin
        if(c == 1'b1)begin
            y=1'b0;
            s=1'b1;
        end
        else begin
            y=1'b1;
            s=1'b0;
        end
    end
    else begin
```

```
            if(c == 1'b1)begin
                y=1'b1;
                s=1'b0;
            end
            else begin
                y=1'b0;
                s=1'b0;
            end
        end


    end


end
endmodule
```

RESULTS:

```
D:\Shreeya\iverilog\bin>vvp.exe add
VCD info: dumpfile bitadder.vcd opened for output.
a=                   0 b=                   0 y=                   0
a=                1029 b=                1027 y=                2056
a=               23967 b=              986290 y=             1010257
a=                9269 b=     -17008070492160 y=     -17008070482891
a=   -3404916928282624 b=      11781095292928 y=   -3393135832989696
a=   -3447827946536960 b=      -8877697400832 y=   -3456705643937792
```

WAVEFORM:



SUB MODULE

Subtraction is the sum of the first number with the negative of the second number.

We will be using the 2s complement notation to obtain the negative number.

Now for taking 2s complement of an n-bit number, we first invert all its bits.

COMPLIMENT MODULE

This is the code for inverting the bits of the input 64-bit number

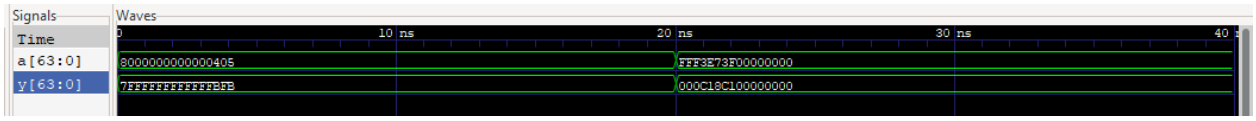
```
module compliment(a,y);

input a;

output reg y;


always @ (a) begin

   if(a==1'b1)

      y = 1'b0;

   else

      y= 1'b1;

end


endmodule
```

We then add nb'0...(n-1 times)1 to it. We used the previously made adder module, to do this addition and generate 2's complement of the given number.

RESULTS:

```
D:\Shreeya\iverilog\bin>vvp comp
VCD info: dumpfile bitcompliment.vcd opened for output.
a=-9223372036854774779 y= 9223372036854774779

a=    -3404916928282624 y=     3404916928282624
```

WAVEFORM:

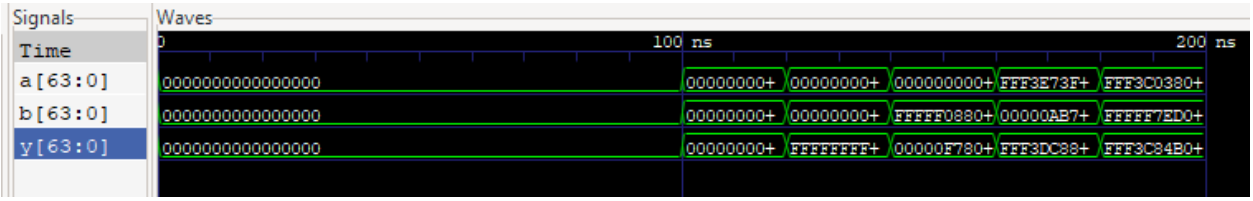


Once the complements are generated, we've used the code to add the 64 bit numbers and generated the output for subtraction.

RESULTS:

```
D:\Shreeya\iverilog\bin>iverilog -o sub bitsub.v sub_TB.v

D:\Shreeya\iverilog\bin>vvp sub
VCD info: dumpfile bitsub.vcd opened for output.
a=                    0 b=                    0 y=                    0

a=                 1029 b=                 1027 y=                    2

a=                23967 b=               986290 y=              -962323

a=                 9269 b=     -17008070492160 y=      17008070501429

a=    -3404916928282624 b=      11781095292928 y=    -3416698023575552

a=    -3447827946536960 b=      -8877697400832 y=    -3438950249136128
```

WAVEFORM:



AND MODULE

The below code is for a 1-bit AND module.

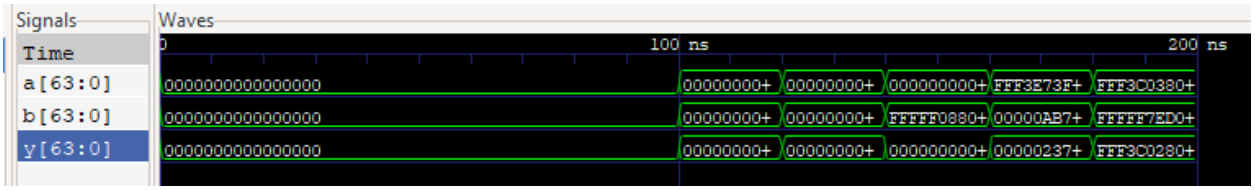
We made use of 4 if-else statements to build the module. The output is 1 only if both the input bits are 1, otherwise, the output is always 0.

```
`timescale 1ns / 1ps

module and(a,b,y);

input a,b;

output reg y;

always @ (a or b) begin

   if (a == 1'b1) begin

        if(b == 1'b1)

            y = 1'b1;

        else

            y = 1'b0;

   end

   else

       y = 1'b0;

end

endmodule
```

RESULTS:

```
D:\Shreeya\iverilog\bin>vvp.exe and
VCD info: dumpfile bitand.vcd opened for output.
a=                    0 b=                    0 y=                    0

a=                 1029 b=                 1027 y=                 1025

a=                23967 b=               986290 y=                 3218

a=                 9269 b=       -17008070492160 y=                    0

a=    -3404916928282624 b=        11781095292928 y=         2435246456832

a=    -3447827946536960 b=        -8877697400832 y=    -3447896666013696
```

WAVEFORM:



XOR MODULE

The below code is for a 1-bit XOR module

We made use of 4 if-else statements to build the module. The output is 1 only when the input bits are opposite, otherwise, the output is always 0.

```
`timescale 1ns / 1ps

module xor(a,b,y);

input a,b;

output reg y;

always @ (a or b) begin

   if (a == 1'b1) begin

         if(b == 1'b1)
```

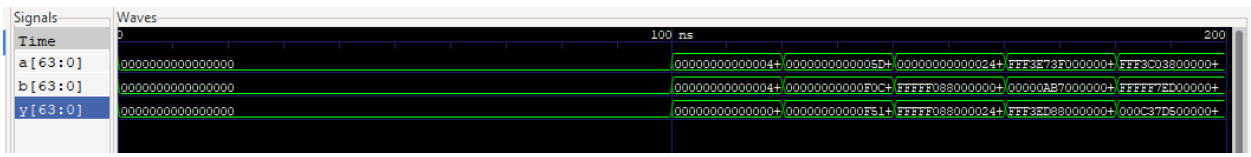
```
            y = 1'b0;
        else
            y = 1'b1;
    end
    else begin
        if(b == 1'b1)
            y = 1'b1;
        else
            y = 1'b0;
    end
end
endmodule
```

If the input bits were unequal, the output is 1 else the output is always zero.

RESULTS:

```
D:\Shreeya\iverilog\bin>vvp.exe xor
VCD info: dumpfile bitxor.vcd opened for output.
a=                   0 b=                   0 y=                   0
a=                1029 b=                1027 y=                   6
a=               23967 b=              986290 y=             1003821
a=                9269 b=     -17008070492160 y=     -17008070482891
a=   -3404916928282624 b=      11781095292928 y=   -3398006325903360
a=   -3447827946536960 b=      -8877697400832 y=    3439087688089600
```

WAVEFORM:



ALU MODULE

The function of ALU is to call the modules above based on the control input. The ALU unit takes as input the control signal, and two 64-bit inputs, and returns the 64-bit output corresponding to the control signal chosen.

The Control performs 4 functions,

0 performs addition,

1 performs subtraction,

2 performs AND operation and,

3 performs XOR operation

Using the above modules, we obtain the desired output as per the given control.

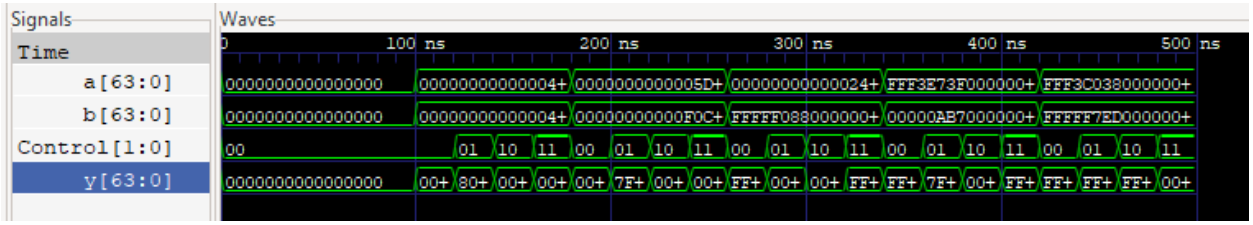
```
540  module ALU(Control,a,b,y);
541      input [63:0] a,b;
542      input [1:0] Control;
543      output reg [63:0] y;
544      wire [63:0] y1,y2,y3,y4;
545      reg [63:0] r1,r2,r3,r4;
546      bitadder c1(a,b,y1);
547      bitsub c2(a,b,y2);
548      bitand c3 (a,b,y3);
549      bitxor c4(a,b,y4);
550
551      always @(*)
552       begin
553          case(Control)
554 >            2'b00: begin    ···
620 >            2'b01:  begin···
686 >            2'b10:  begin···
752 >            2'b11:  begin···
819          endcase
820
821       end
822
823  endmodule
```

RESULTS:

Command Prompt

```
D:\Shreeya\iverilog\bin>vvp alu
VCD info: dumpfile ALU.vcd opened for output.
a=                   0 b=                   0 Control=0 y=                   0

a=                1029 b=                1027 Control=0 y=                2056

a=                1029 b=                1027 Control=1 y=-9223372036854775806

a=                1029 b=                1027 Control=2 y=                1025

a=                1029 b=                1027 Control=3 y=                   6

a=               23967 b=              986290 Control=0 y=             1010257

a=               23967 b=              986290 Control=1 y= 9223372036853813485

a=               23967 b=              986290 Control=2 y=                3218

a=               23967 b=              986290 Control=3 y=             1003821

a=                9269 b=     -17008070492160 Control=0 y=     -17008070482891

a=                9269 b=     -17008070492160 Control=1 y=      17008070501429

a=                9269 b=     -17008070492160 Control=2 y=                   0

a=                9269 b=     -17008070492160 Control=3 y=     -17008070482891

a=   -3404916928282624 b=      11781095292928 Control=0 y=   -3393135832989696

a=   -3404916928282624 b=      11781095292928 Control=1 y= 9219955338831200256

a=   -3404916928282624 b=      11781095292928 Control=2 y=       2435246456832

a=   -3404916928282624 b=      11781095292928 Control=3 y=   -3398006325903360

a=   -3447827946536960 b=      -8877697400832 Control=0 y=   -3456705643937792

a=   -3447827946536960 b=      -8877697400832 Control=1 y=   -3438950249136128

a=   -3447827946536960 b=      -8877697400832 Control=2 y=   -3447896666013696

a=   -3447827946536960 b=      -8877697400832 Control=3 y=    3439087688089600
```

WAVEFORM:



TESTBENCH

We used the following 64-bit numbers as inputs for the testbench

```
a = 64'b0000000000000000000000000000000000000000000000000000000000000000;
b = 64'b0000000000000000000000000000000000000000000000000000000000000000;

a = 64'b0000000000000000000000000000000000000000000000000000010000000101;
b = 64'b0000000000000000000000000000000000000000000000000000010000000011;

a = 64'b0000000000000000000000000000000000000000000000001011101100011111;
b = 64'b0000000000000000000000000000000000000000000011110000110010110010;

a = 64'b0000000000000000000000000000000000000000000000010010000110101;
b = 64'b1111111111111111111100001000100000000000000000000000000000000000;

a = 64'b1111111111110011111001100111111000000000000000000000000000000000;
b = 64'b0000000000000000000010101011011100000000000000000000000000000000;

a = 64'b1111111111110011110000000111000000000000000000000000000000000000;
b = 64'b1111111111111111111101111110110100000000000000000000000000000000;
```