64-Bit ALU

Tanish Taneja

2021112011

Overview

- We have to build an ALU with the following functionalities :

  i) AND

  ii) XOR

  iii) ADD

  iv) SUB (using 2's complement)

- All operations are on signed 64 bit input numbers `x` and `y` and return a signed 64 bit output `ans` and an `overflow` bit

- A control bit is taken as input which indicates the operation to be performed :

  i) Control 0 - ADD x and y

  ii) Control 1 – SUB y from x

  iii) Control 2 – AND x and y

  iv) Control 3 – XOR x and y

AND

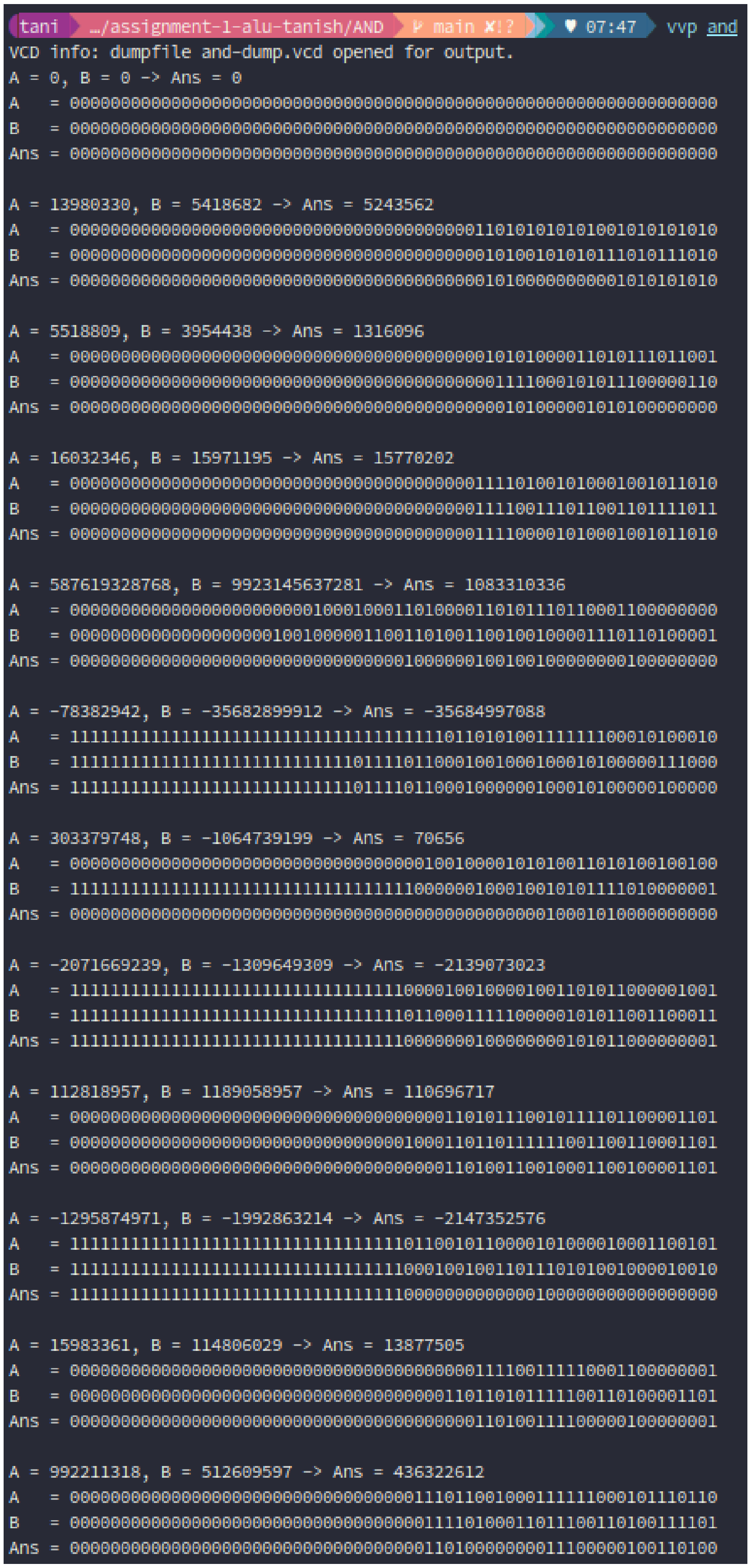
- All the required verilog files are found within the AND directory.

- The implementation is using a simple `for loop` which does the bitwise `AND` operation.

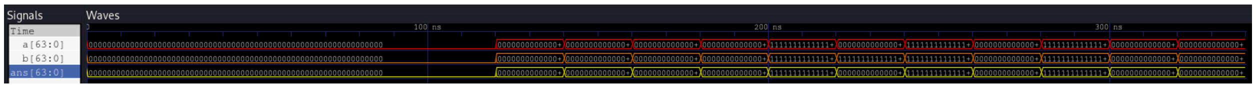
- `and-64.v` defines the module and `and-testbench.v` is the testbench for the same.

- To compile and see results, run

```
iverilog -o and and-64.v and-testbench.v
vvp and
gtkwave and-dump.vcd
```



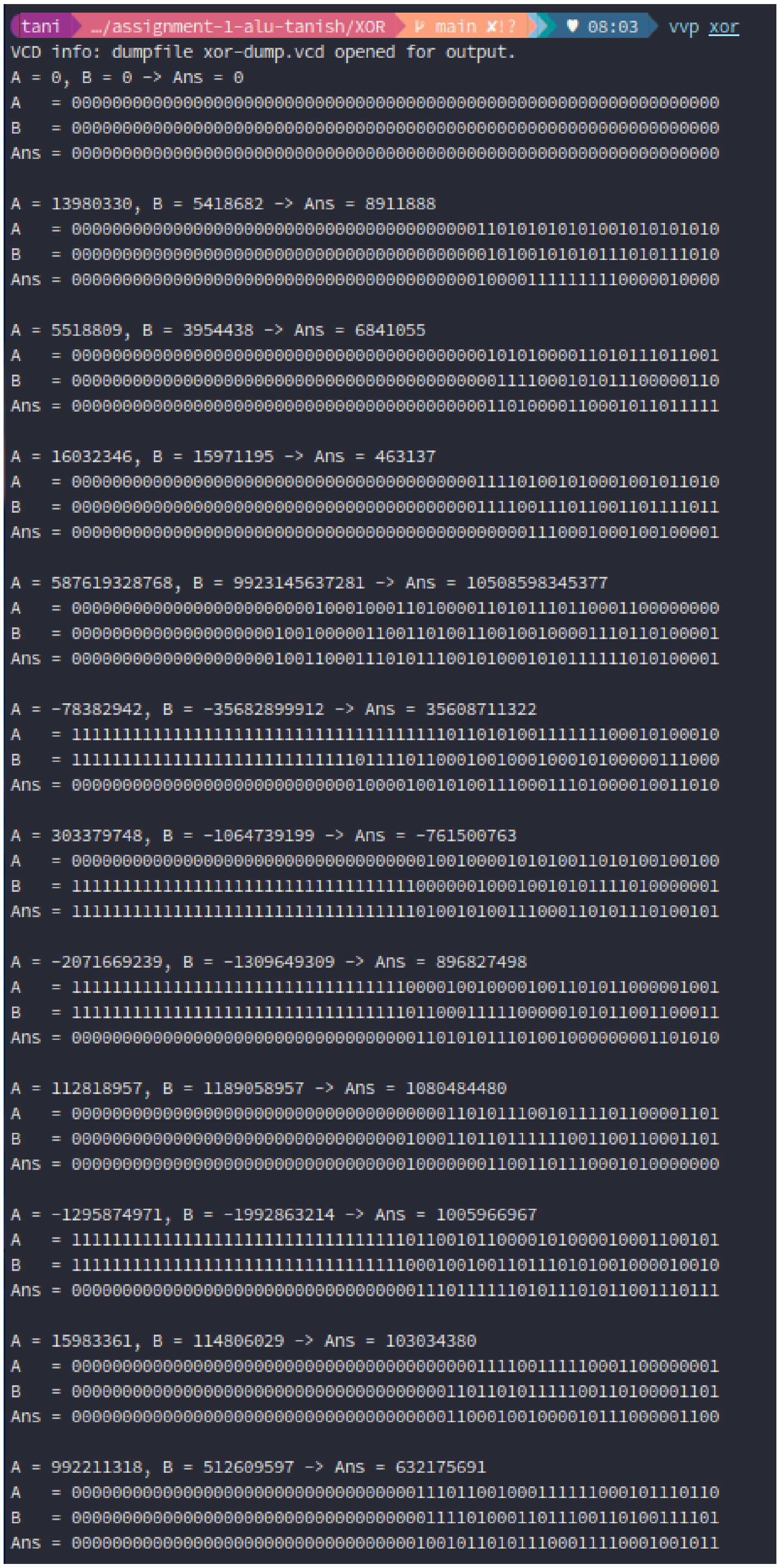
```
tani  …/assignment-1-alu-tanish/AND   main ✘!?   ♥ 07:47  vvp and
VCD info: dumpfile and-dump.vcd opened for output.
A = 0, B = 0 -> Ans = 0
A   = 0000000000000000000000000000000000000000000000000000000000000000
B   = 0000000000000000000000000000000000000000000000000000000000000000
Ans = 0000000000000000000000000000000000000000000000000000000000000000

A = 13980330, B = 5418682 -> Ans = 5243562
A   = 0000000000000000000000000000000000000000110101010101001010101010
B   = 0000000000000000000000000000000000000000010100101010111010111010
Ans = 0000000000000000000000000000000000000000010100000000001010101010

A = 5518809, B = 3954438 -> Ans = 1316096
A   = 0000000000000000000000000000000000000000010101000011010111011001
B   = 0000000000000000000000000000000000000000001111000101011100000110
Ans = 0000000000000000000000000000000000000000000101000001010100000000

A = 16032346, B = 15971195 -> Ans = 15770202
A   = 0000000000000000000000000000000000000000111101001010001001011010
B   = 0000000000000000000000000000000000000000111100111011001101111011
Ans = 0000000000000000000000000000000000000000111100001010001001011010

A = 587619328768, B = 9923145637281 -> Ans = 1083310336
A   = 0000000000000000000000001000100011010000110101110110001100000000
B   = 0000000000000000000010010000011001101001100100100001110110100001
Ans = 0000000000000000000000000000000001000000100100100000000100000000

A = -78382942, B = -35682899912 -> Ans = -35684997088
A   = 1111111111111111111111111111111111111011010100111111100010100010
B   = 1111111111111111111111111111011110110001001000100010100000111000
Ans = 1111111111111111111111111111011110110001000000100010100000100000

A = 303379748, B = -1064739199 -> Ans = 70656
A   = 0000000000000000000000000000000000010010000101010011010100100100
B   = 1111111111111111111111111111111111000000100010010101111010000001
Ans = 0000000000000000000000000000000000000000000000010001010000000000

A = -2071669239, B = -1309649309 -> Ans = -2139073023
A   = 1111111111111111111111111111111110000100100001001101011000001001
B   = 1111111111111111111111111111111110110001111100000101011001100011
Ans = 1111111111111111111111111111111110000000100000000101011000000001

A = 112818957, B = 1189058957 -> Ans = 110696717
A   = 0000000000000000000000000000000000000110101110010111101100001101
B   = 0000000000000000000000000000000001000110110111111001100110001101
Ans = 0000000000000000000000000000000000000110100110010001100100001101

A = -1295874971, B = -1992863214 -> Ans = -2147352576
A   = 1111111111111111111111111111111110110010110000101000010001100101
B   = 1111111111111111111111111111111110001001001101110101001000010010
Ans = 1111111111111111111111111111111110000000000000100000000000000000

A = 15983361, B = 114806029 -> Ans = 13877505
A   = 0000000000000000000000000000000000000000111100111110001100000001
B   = 0000000000000000000000000000000000000110110101111100110100001101
Ans = 0000000000000000000000000000000000000000110100111100000100000001

A = 992211318, B = 512609597 -> Ans = 436322612
A   = 0000000000000000000000000000000000111011001000111111000101110110
B   = 0000000000000000000000000000000000011110100011011100110100111101
Ans = 0000000000000000000000000000000000011010000000011100000100110100
```



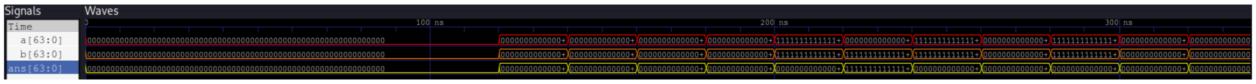
XOR

- All the required verilog files are found within the XOR directory.
- The implementation is using a simple `for loop` which does the bitwise `XOR` operation.
- `xor-64.v` defines the module and `xor-testbench.v` is the testbench for the same.
- To compile and see results, run

```
iverilog -o xor xor-64.v xor-testbench.v
vvp xor
gtkwave xor-dump.vcd
```



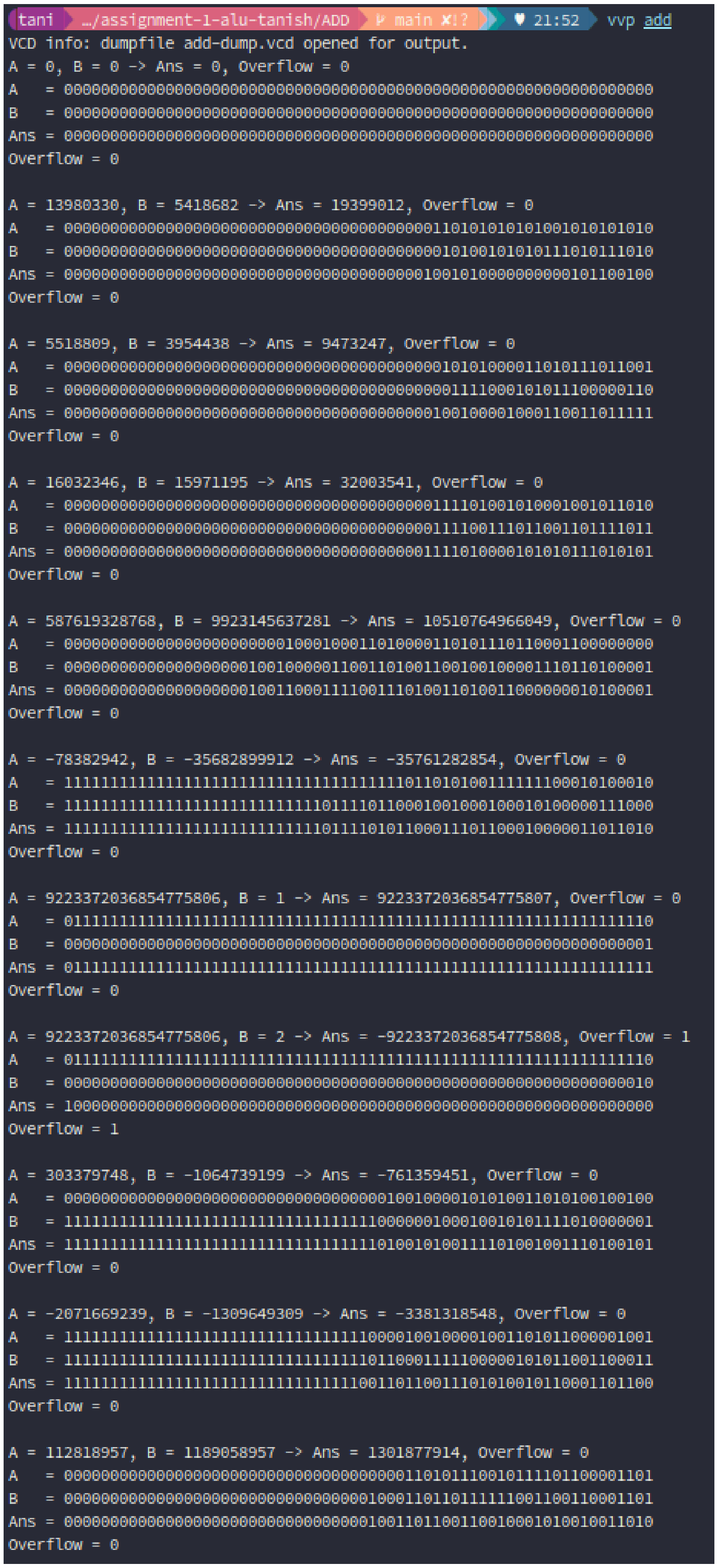
```
tani  …/assignment-1-alu-tanish/XOR   main ✘!?   08:03  vvp xor
VCD info: dumpfile xor-dump.vcd opened for output.
A = 0, B = 0 -> Ans = 0
A   = 0000000000000000000000000000000000000000000000000000000000000000
B   = 0000000000000000000000000000000000000000000000000000000000000000
Ans = 0000000000000000000000000000000000000000000000000000000000000000

A = 13980330, B = 5418682 -> Ans = 8911888
A   = 0000000000000000000000000000000000000000110101010101001010101010
B   = 0000000000000000000000000000000000000000010100101010111010111010
Ans = 0000000000000000000000000000000000000000100001111111110000010000

A = 5518809, B = 3954438 -> Ans = 6841055
A   = 0000000000000000000000000000000000000000010101000011010111011001
B   = 0000000000000000000000000000000000000000001111000101011100000110
Ans = 0000000000000000000000000000000000000000011010000110001011011111

A = 16032346, B = 15971195 -> Ans = 463137
A   = 0000000000000000000000000000000000000000111101001010001001011010
B   = 0000000000000000000000000000000000000000111100111011001101111011
Ans = 0000000000000000000000000000000000000000000001110001000100100001

A = 587619328768, B = 9923145637281 -> Ans = 10508598345377
A   = 0000000000000000000000001000100011010000110101110110001100000000
B   = 0000000000000000000010010000011001101001100100100001110110100001
Ans = 0000000000000000000010011000111010111001010001010111111010100001

A = -78382942, B = -35682899912 -> Ans = 35608711322
A   = 1111111111111111111111111111111111111011010100111111100010100010
B   = 1111111111111111111111111111011110110001001000100010100000111000
Ans = 0000000000000000000000000000100001001010011100011101000010011010

A = 303379748, B = -1064739199 -> Ans = -761500763
A   = 0000000000000000000000000000000000010010000101010011010100100100
B   = 1111111111111111111111111111111111000000100010010101111010000001
Ans = 1111111111111111111111111111111111010010100111000110101110100101

A = -2071669239, B = -1309649309 -> Ans = 896827498
A   = 1111111111111111111111111111111110000100100001001101011000001001
B   = 1111111111111111111111111111111110110001111100000101011001100011
Ans = 0000000000000000000000000000000000110101011101001000000001101010

A = 112818957, B = 1189058957 -> Ans = 1080484480
A   = 0000000000000000000000000000000000000110101110010111101100001101
B   = 0000000000000000000000000000000001000110110111111001100110001101
Ans = 0000000000000000000000000000000001000000011001101110001010000000

A = -1295874971, B = -1992863214 -> Ans = 1005966967
A   = 1111111111111111111111111111111110110010110000101000010001100101
B   = 1111111111111111111111111111111110001001001101110101001000010010
Ans = 0000000000000000000000000000000000111011111101011101011001110111

A = 15983361, B = 114806029 -> Ans = 103034380
A   = 0000000000000000000000000000000000000000111100111110001100000001
B   = 0000000000000000000000000000000000000110110101111100110100001101
Ans = 0000000000000000000000000000000000000110001001000010111000001100

A = 992211318, B = 512609597 -> Ans = 632175691
A   = 0000000000000000000000000000000000111011001000111111000101110110
B   = 0000000000000000000000000000000000011110100011011100110100111101
Ans = 0000000000000000000000000000000000100101101011100011110001001011
```



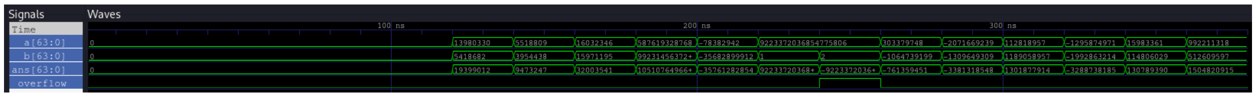
ADD

- All the required verilog files are found within the ADD directory.
- The implementation is using 64 `1 bit` full adders (which have a sum and carry). We also have an overflow which is used to check whether out output fits within the size of 64 bits or not.
- `add-1.v, add-` `64.v` define the module and `add-testbench.v` is the testbench for the same.
- To compile and see results, run

```
iverilog -o add add-1.v add-64.v add-testbench.v
vvp add
gtkwave add-dump.vcd
```



```
tani  …/assignment-1-alu-tanish/ADD  ᚠ main ✘!?  ♥ 21:52  vvp add
VCD info: dumpfile add-dump.vcd opened for output.
A = 0, B = 0 -> Ans = 0, Overflow = 0
A   = 0000000000000000000000000000000000000000000000000000000000000000
B   = 0000000000000000000000000000000000000000000000000000000000000000
Ans = 0000000000000000000000000000000000000000000000000000000000000000
Overflow = 0

A = 13980330, B = 5418682 -> Ans = 19399012, Overflow = 0
A   = 0000000000000000000000000000000000000000110101010101001010101010
B   = 0000000000000000000000000000000000000000010100101010111010111010
Ans = 0000000000000000000000000000000000000001001010000000000101100100
Overflow = 0

A = 5518809, B = 3954438 -> Ans = 9473247, Overflow = 0
A   = 0000000000000000000000000000000000000000010101000011010111011001
B   = 0000000000000000000000000000000000000000001111000101011100000110
Ans = 0000000000000000000000000000000000000000100100001000110011011111
Overflow = 0

A = 16032346, B = 15971195 -> Ans = 32003541, Overflow = 0
A   = 0000000000000000000000000000000000000000111101001010001001011010
B   = 0000000000000000000000000000000000000000111100111011001101111011
Ans = 0000000000000000000000000000000000000001111010000101010111010101
Overflow = 0

A = 587619328768, B = 9923145637281 -> Ans = 10510764966049, Overflow = 0
A   = 0000000000000000000000001000100011010000110101110110001100000000
B   = 0000000000000000000010010000011001101001100100100001110110100001
Ans = 0000000000000000000010011000111100111010011010011000000010100001
Overflow = 0

A = -78382942, B = -35682899912 -> Ans = -35761282854, Overflow = 0
A   = 1111111111111111111111111111111111111011010100111111100010100010
B   = 1111111111111111111111111111011110110001001000100010100000111000
Ans = 1111111111111111111111111111011110101100011101100010000011011010
Overflow = 0

A = 9223372036854775806, B = 1 -> Ans = 9223372036854775807, Overflow = 0
A   = 0111111111111111111111111111111111111111111111111111111111111110
B   = 0000000000000000000000000000000000000000000000000000000000000001
Ans = 0111111111111111111111111111111111111111111111111111111111111111
Overflow = 0

A = 9223372036854775806, B = 2 -> Ans = -9223372036854775808, Overflow = 1
A   = 0111111111111111111111111111111111111111111111111111111111111110
B   = 0000000000000000000000000000000000000000000000000000000000000010
Ans = 1000000000000000000000000000000000000000000000000000000000000000
Overflow = 1

A = 303379748, B = -1064739199 -> Ans = -761359451, Overflow = 0
A   = 0000000000000000000000000000000000010010000101010011010100100100
B   = 1111111111111111111111111111111111000000100010010101111010000001
Ans = 1111111111111111111111111111111111010010100111101001001110100101
Overflow = 0

A = -2071669239, B = -1309649309 -> Ans = -3381318548, Overflow = 0
A   = 1111111111111111111111111111111110000100100001001101011000001001
B   = 1111111111111111111111111111111110110001111100000101011001100011
Ans = 1111111111111111111111111111111100110110011101010010110001101100
Overflow = 0

A = 112818957, B = 1189058957 -> Ans = 1301877914, Overflow = 0
A   = 0000000000000000000000000000000000000110101110010111101100001101
B   = 0000000000000000000000000000000001000110110111111001100110001101
Ans = 0000000000000000000000000000000001001101100110010001010010011010
Overflow = 0
```

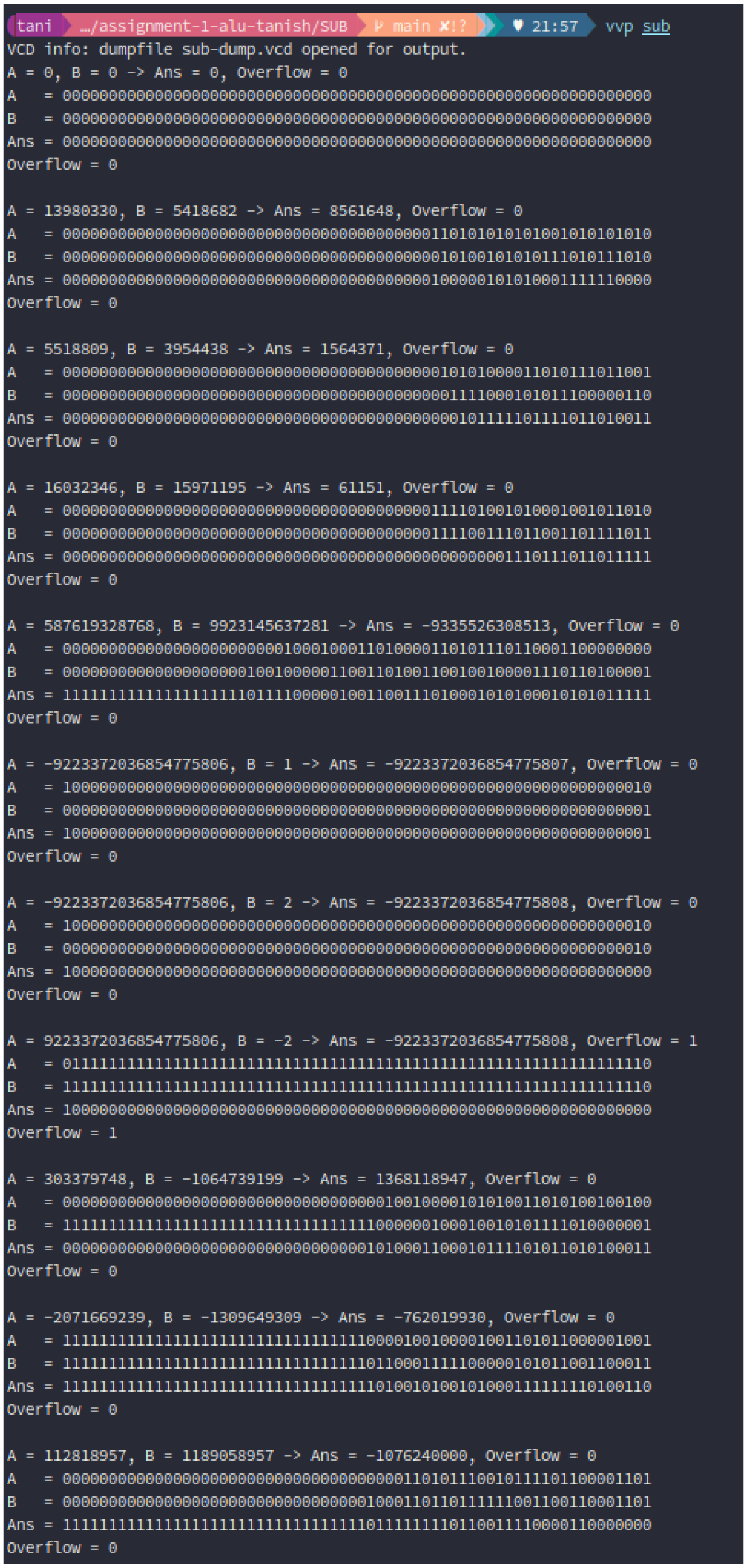


```
Time                  100 ns                                        200 ns                                                        300 ns
a[63:0]    0   13980330 5518809 16032346 587619328768   -78382942     9223372036854775806           303379748   -2071669239 112818957  -1295874971 15983361  992211318
b[63:0]    0   5418682  3954438 15971195 99231456372+   -35682899912  1             2               -1064739199 -1309649309 1189058957 -1992863214 114806029 512609597
ans[63:0]  0   19399012 9473247 32003541 10510764966+   -35761282854  92233720368+  -9223372036+    -761359451  -3381318548 1301877914 -3288738185 130789390 1504820915
overflow
```

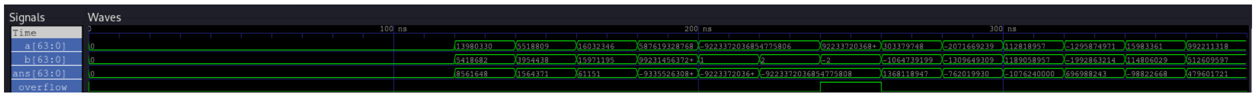
SUB

- All the required verilog files are found within the SUB directory.
- The implementation is using the adders with the only difference being we first find the `2s complement` of the number to be subtracted and add that.
- `not-64.v, sub-64.v` define the module (the `add-64.v` module is also used) and `sub-testbench.v` is the testbench for the same.
- To compile and see results, run

```
iverilog -o sub sub-64.v not-64.v helper.v sub-testbench.v
vvp sub
gtkwave sub-dump.vcd
```



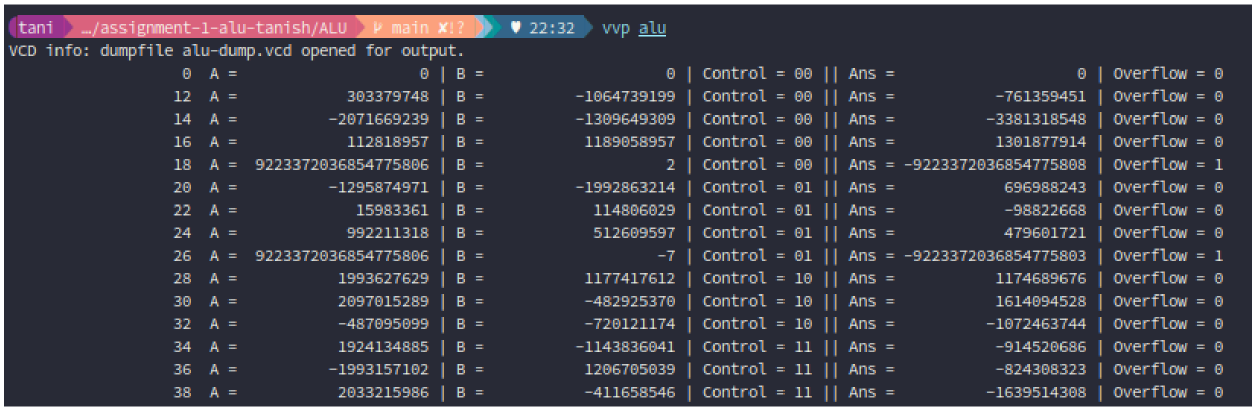
```
tani  …/assignment-1-alu-tanish/SUB  ᚠ main ✘!?  ♥ 21:57  vvp sub
VCD info: dumpfile sub-dump.vcd opened for output.
A = 0, B = 0 -> Ans = 0, Overflow = 0
A   = 0000000000000000000000000000000000000000000000000000000000000000
B   = 0000000000000000000000000000000000000000000000000000000000000000
Ans = 0000000000000000000000000000000000000000000000000000000000000000
Overflow = 0

A = 13980330, B = 5418682 -> Ans = 8561648, Overflow = 0
A   = 0000000000000000000000000000000000000000110101010101001010101010
B   = 0000000000000000000000000000000000000000010100101010111010111010
Ans = 0000000000000000000000000000000000000000100000101010001111110000
Overflow = 0

A = 5518809, B = 3954438 -> Ans = 1564371, Overflow = 0
A   = 0000000000000000000000000000000000000000010101000011010111011001
B   = 0000000000000000000000000000000000000000001111000101011100000110
Ans = 0000000000000000000000000000000000000000000101111101111011010011
Overflow = 0

A = 16032346, B = 15971195 -> Ans = 61151, Overflow = 0
A   = 0000000000000000000000000000000000000000111101001010001001011010
B   = 0000000000000000000000000000000000000000111100111011001101111011
Ans = 0000000000000000000000000000000000000000000000001110111011011111
Overflow = 0

A = 587619328768, B = 9923145637281 -> Ans = -9335526308513, Overflow = 0
A   = 0000000000000000000000001000100011010000110101110110001100000000
B   = 0000000000000000000010010000011001101001100100100001110110100001
Ans = 1111111111111111111101111000001001100111010001010100010101011111
Overflow = 0

A = -9223372036854775806, B = 1 -> Ans = -9223372036854775807, Overflow = 0
A   = 1000000000000000000000000000000000000000000000000000000000000010
B   = 0000000000000000000000000000000000000000000000000000000000000001
Ans = 1000000000000000000000000000000000000000000000000000000000000001
Overflow = 0

A = -9223372036854775806, B = 2 -> Ans = -9223372036854775808, Overflow = 0
A   = 1000000000000000000000000000000000000000000000000000000000000010
B   = 0000000000000000000000000000000000000000000000000000000000000010
Ans = 1000000000000000000000000000000000000000000000000000000000000000
Overflow = 0

A = 9223372036854775806, B = -2 -> Ans = -9223372036854775808, Overflow = 1
A   = 0111111111111111111111111111111111111111111111111111111111111110
B   = 1111111111111111111111111111111111111111111111111111111111111110
Ans = 1000000000000000000000000000000000000000000000000000000000000000
Overflow = 1

A = 303379748, B = -1064739199 -> Ans = 1368118947, Overflow = 0
A   = 0000000000000000000000000000000000010010000101010011010100100100
B   = 1111111111111111111111111111111111000000100010010101111010000001
Ans = 0000000000000000000000000000000001010001100010111101011010100011
Overflow = 0

A = -2071669239, B = -1309649309 -> Ans = -762019930, Overflow = 0
A   = 1111111111111111111111111111111110000100100001001101011000001001
B   = 1111111111111111111111111111111110110001111100000101011001100011
Ans = 1111111111111111111111111111111111010010100101000111111110100110
Overflow = 0

A = 112818957, B = 1189058957 -> Ans = -1076240000, Overflow = 0
A   = 0000000000000000000000000000000000000110101110010111101100001101
B   = 0000000000000000000000000000000001000110110111111001100110001101
Ans = 1111111111111111111111111111111110111111110110011110000110000000
Overflow = 0
```



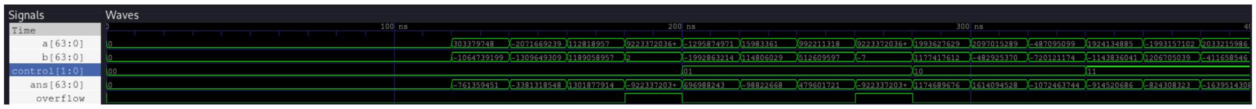
ALU

- All the required verilog files are found within the ALU directory.
- The implementation is using all the modules (contained in `helper.v`) we have defined earlier and using a switch case to decide based on the control bit.
- `alu.v` define the module and *`alu-testbench.v`* is the testbench for the same.
- To compile and see results, run

```
iverilog -o alu helper.v alu.v alu-testbench.vcd
vvp alu
gtkwave alu-dump.vcd
```



```
tani  …/assignment-1-alu-tanish/ALU  main ✘!?  22:32  vvp alu
VCD info: dumpfile alu-dump.vcd opened for output.
                  0  A =                    0 | B =                    0 | Control = 00 || Ans =                    0 | Overflow = 0
                 12  A =            303379748 | B =          -1064739199 | Control = 00 || Ans =           -761359451 | Overflow = 0
                 14  A =          -2071669239 | B =          -1309649309 | Control = 00 || Ans =          -3381318548 | Overflow = 0
                 16  A =            112818957 | B =           1189058957 | Control = 00 || Ans =           1301877914 | Overflow = 0
                 18  A =  9223372036854775806 | B =                    2 | Control = 00 || Ans = -9223372036854775808 | Overflow = 1
                 20  A =          -1295874971 | B =          -1992863214 | Control = 01 || Ans =            696988243 | Overflow = 0
                 22  A =             15983361 | B =            114806029 | Control = 01 || Ans =            -98822668 | Overflow = 0
                 24  A =            992211318 | B =            512609597 | Control = 01 || Ans =            479601721 | Overflow = 0
                 26  A =  9223372036854775806 | B =                   -7 | Control = 01 || Ans = -9223372036854775803 | Overflow = 1
                 28  A =           1993627629 | B =           1177417612 | Control = 10 || Ans =           1174689676 | Overflow = 0
                 30  A =           2097015289 | B =           -482925370 | Control = 10 || Ans =           1614094528 | Overflow = 0
                 32  A =           -487095099 | B =           -720121174 | Control = 10 || Ans =          -1072463744 | Overflow = 0
                 34  A =           1924134885 | B =          -1143836041 | Control = 11 || Ans =           -914520686 | Overflow = 0
                 36  A =          -1993157102 | B =           1206705039 | Control = 11 || Ans =           -824308323 | Overflow = 0
                 38  A =           2033215986 | B =           -411658546 | Control = 11 || Ans =          -1639514308 | Overflow = 0
```


Signals
Waves
Time
a[63:0]
b[63:0]
control[1:0]
ans[63:0]
overflow