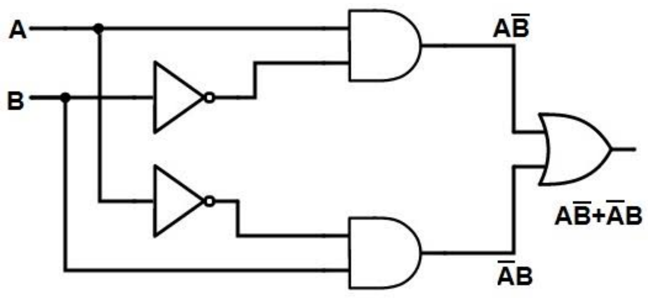
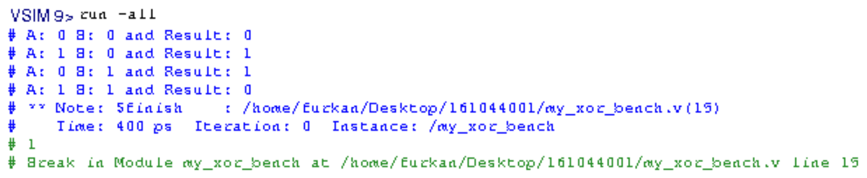
CSE 331 HW2 REPORT

Firstly, since using xor gate is prohibited, created a xor module.

By using this diagram I created my xor module.I used 2 NOT gate and 2 AND gate and also to fetch result I “OR”ed results of AND.



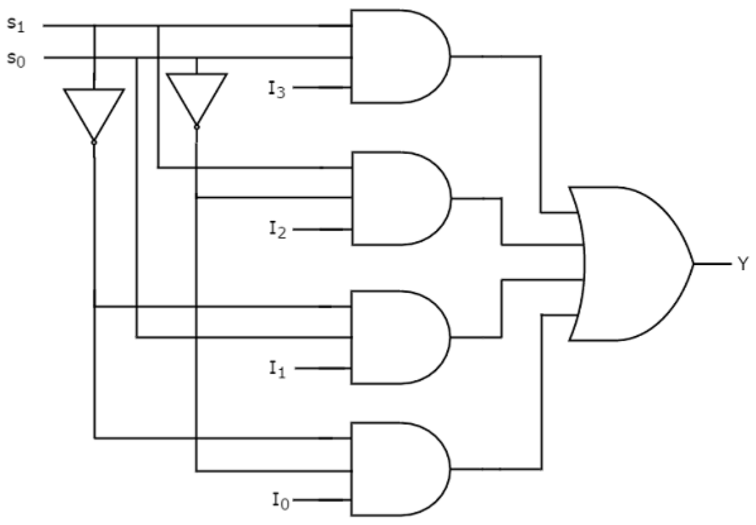
This is testbench results of XOR.



```
VSIM 9> run -all
# A: 0 B: 0 and Result: 0
# A: 1 B: 0 and Result: 1
# A: 0 B: 1 and Result: 1
# A: 1 B: 1 and Result: 0
# ** Note: $finish    : /home/furkan/Desktop/161044001/my_xor_bench.v(15)
#    Time: 400 ps  Iteration: 0  Instance: /my_xor_bench
# 1
# Break in Module my_xor_bench at /home/furkan/Desktop/161044001/my_xor_bench.v line 15
```

After creating XOR module, Project ready to go next phases.

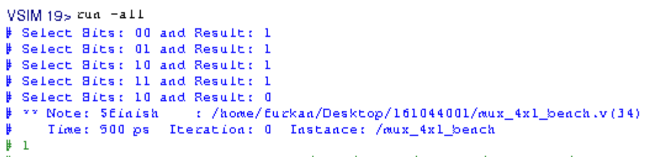
To create 4x1 MUX I used block diagram.



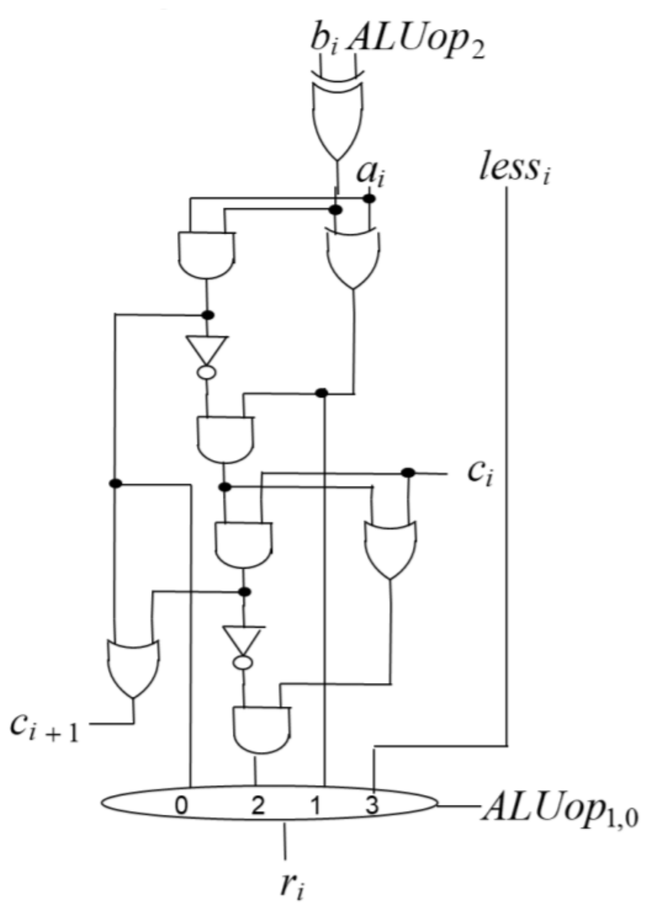
As shown in diagram, I used 2 NOT gate to invert select bits. After that to obtain selected values, I used 4 AND gate and finally to get final result I used and or gate.

At testbench, I changed test bench step by step and I also changed Inputs to select always 1. As shown in output, mux selected always 1 but at last case I changed select bit to any one of the 0’s and as a result, I get 0 as expected.

This is my testbench result for 4x1 MUX.



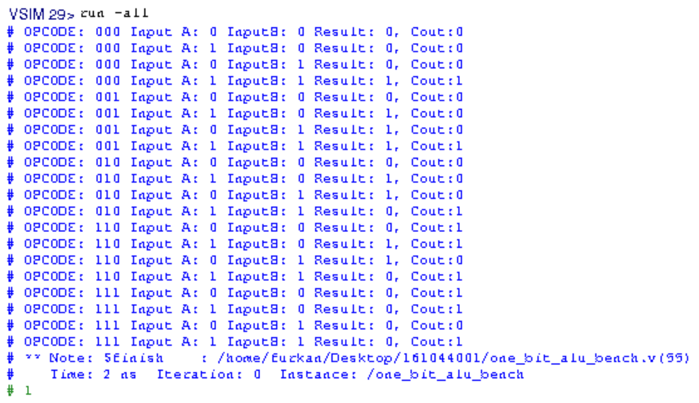
```
VSIM 19> run -all
# Select Bits: 00 and Result: 1
# Select Bits: 01 and Result: 1
# Select Bits: 10 and Result: 1
# Select Bits: 11 and Result: 1
# Select Bits: 10 and Result: 0
# ** Note: $finish    : /home/furkan/Desktop/161044001/mux_4x1_bench.v(34)
#    Time: 500 ps  Iteration: 0  Instance: /mux_4x1_bench
# 1
```



When MUX is ready, next step is creating 1-bit ALU. In this process, we have xor module and MUX that created above and other fundamental gates. Firstly, xor operation performed and first sum performed. From this sum we have also OR and AND result of operands. After that sum is performed with Carry In. Also as a result of first and second sum we determine there is a carry or not with and additional OR gate. In 1 bit ALU less does not have meaning, since that we give that input zero at testbench. With that we all results and we can finally select with help of MUX. Operation of giving ALUop2 to xor, determines there is a substraction or adding operation. Xor is used to invert bit but also this must be remembered. Cin at substraction must be 1 and at summation must be 0 for one bit ALU. If we think 2's complement after inverting a number we must add 1 to that number. This result comes from here.
I used 1 XOR gate which is consist of 2 NOT 2 AND and 1 OR gate. Inside ALU, we have 4 AND gate, 3 OR gate and 2 NOT gate. Also we have 4x1 MUX to select desired result. We determine desired result by looking OP1 and OP2. With all of that we have 1 bit ALU that performs anding, oring, addition, substraction and comparison with set-on-less-than.

As sum with all mux and xor module we have 10 AND gate, 5 OR gate, 6 NOT gate.

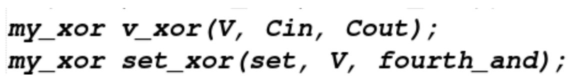
This is the testbench results of 1'bit ALU.



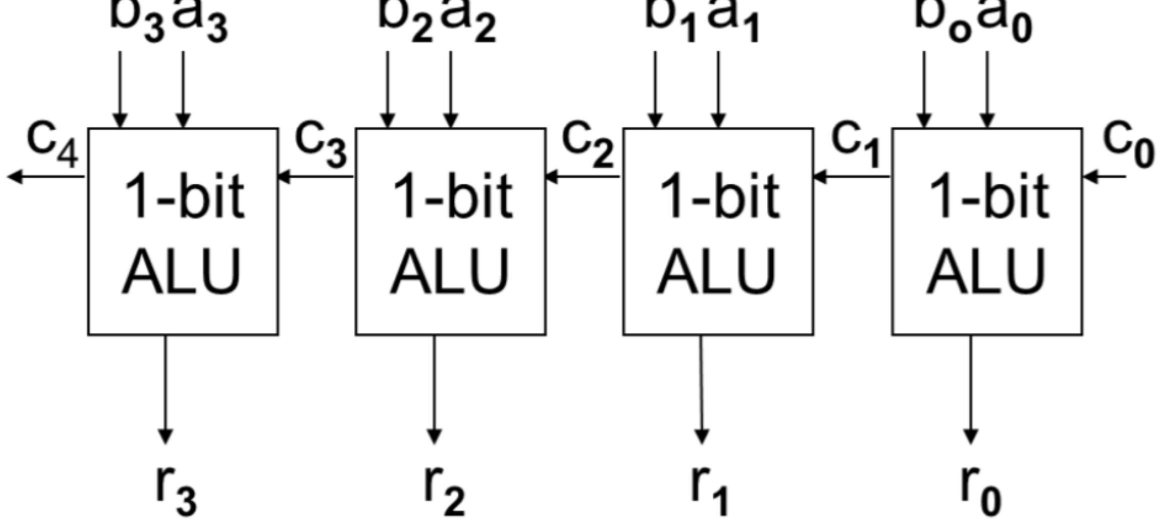
```
VSIM 29> run -all
# OPCODE: 000 Input A: 0 InputB: 0 Result: 0, Cout:0
# OPCODE: 000 Input A: 1 InputB: 0 Result: 0, Cout:0
# OPCODE: 000 Input A: 0 InputB: 1 Result: 0, Cout:0
# OPCODE: 000 Input A: 1 InputB: 1 Result: 1, Cout:1
# OPCODE: 001 Input A: 0 InputB: 0 Result: 0, Cout:0
# OPCODE: 001 Input A: 1 InputB: 0 Result: 1, Cout:0
# OPCODE: 001 Input A: 0 InputB: 1 Result: 1, Cout:0
# OPCODE: 001 Input A: 1 InputB: 1 Result: 1, Cout:1
# OPCODE: 010 Input A: 0 InputB: 0 Result: 0, Cout:0
# OPCODE: 010 Input A: 1 InputB: 0 Result: 1, Cout:0
# OPCODE: 010 Input A: 0 InputB: 1 Result: 1, Cout:0
# OPCODE: 010 Input A: 1 InputB: 1 Result: 0, Cout:1
# OPCODE: 110 Input A: 0 InputB: 0 Result: 0, Cout:1
# OPCODE: 110 Input A: 1 InputB: 0 Result: 1, Cout:1
# OPCODE: 110 Input A: 0 InputB: 1 Result: 1, Cout:0
# OPCODE: 110 Input A: 1 InputB: 1 Result: 0, Cout:1
# OPCODE: 111 Input A: 0 InputB: 0 Result: 0, Cout:1
# OPCODE: 111 Input A: 1 InputB: 0 Result: 0, Cout:1
# OPCODE: 111 Input A: 0 InputB: 1 Result: 0, Cout:0
# OPCODE: 111 Input A: 1 InputB: 1 Result: 0, Cout:1
# ** Note: $finish    : /home/furkan/Desktop/161044001/one_bit_alu_bench.v(55)
#    Time: 2 ns  Iteration: 0  Instance: /one_bit_alu_bench
# 1
```

Before creating 32-bit ALU we must consider that 0 to 31 bit, 31 bit ALU must be different than others. This is a special case for MSB bit and determines some values of overall result. This ALU performs all operations that performed on 1-bit ALU but at end we must determine that is there any overflow, and also according to substraction and overflow result, we must determine less input of 0th bit of 32-bit ALU. This can be done for overflow by performing XOR between last Carry Input and last Carry Output. For set-on-less-than input of 0th bit, we again perform XOR between overflown and substraction result. Result of last XOR connected to 0th bit less input.

For MSB ALU adding these two line to normal 1-bit ALU is sufficient.



```
my_xor v_xor(V, Cin, Cout);
my_xor set_xor(set, V, fourth_and);
```

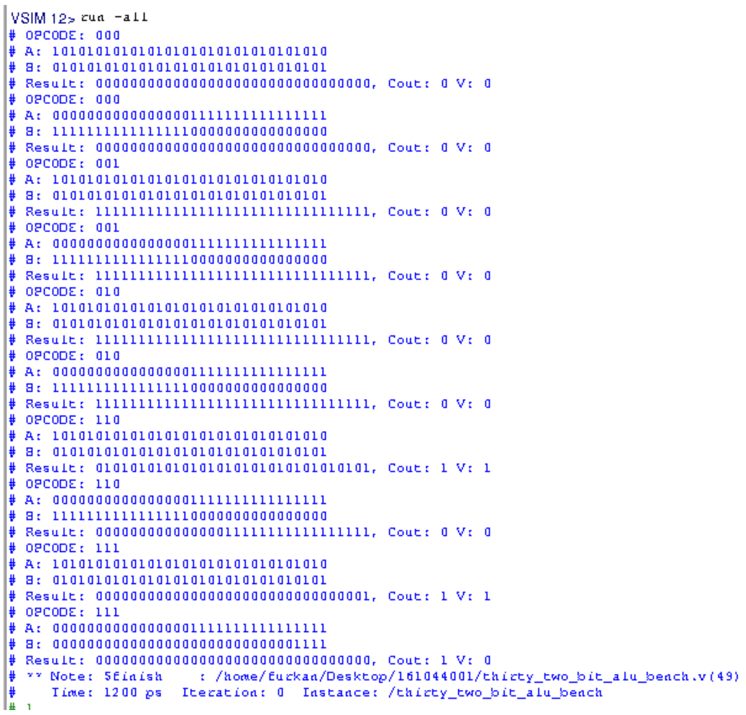


For 32-bit ALU we use 31 1-bit ALU and 1-bit MSB ALU. We give Carry Output of one ALU to next ALU Carry Input. We do operations one by one, step by step and calculation result from LSB to MSB. At the end we determine less value and also is there any overflow. At first all less inputs will be 0 but at the end we determine less result is 0 or 1 by looking its set value. At last bit we must use MSB ALU for finding these results. For 30 1-bit ALU we have 30 XOR module, which has 60 NOT gate 60 AND gate and 30 OR gate. And 30x3 OR gate for 1-bit ALU, 30x4 AND gate and also 30x2 NOT gate. For MSB-bit ALU we have 3 OR gate, 4 AND gate, 2 NOT gate and additionaly one xor module for Overflow flag and one xor module for set value. Also to determine which operation occuring currently, we have 32 4x1MUX.

At the end for 32-bit we have 324 AND gate, 162 OR gate, 196 NOT gate for 32-bit ALU.

TEST RESULTS IN NEXT PAGE.

This is test result of 32-bit ALU desing.



```
VSIM 12> run -all
# OPCODE: 000
# A: 10101010101010101010101010101010
# B: 01010101010101010101010101010101
# Result: 00000000000000000000000000000000, Cout: 0 V: 0
# OPCODE: 000
# A: 00000000000000001111111111111111
# B: 11111111111111110000000000000000
# Result: 00000000000000000000000000000000, Cout: 0 V: 0
# OPCODE: 001
# A: 10101010101010101010101010101010
# B: 01010101010101010101010101010101
# Result: 11111111111111111111111111111111, Cout: 0 V: 0
# OPCODE: 001
# A: 00000000000000001111111111111111
# B: 11111111111111110000000000000000
# Result: 11111111111111111111111111111111, Cout: 0 V: 0
# OPCODE: 010
# A: 10101010101010101010101010101010
# B: 01010101010101010101010101010101
# Result: 11111111111111111111111111111111, Cout: 0 V: 0
# OPCODE: 010
# A: 00000000000000001111111111111111
# B: 11111111111111110000000000000000
# Result: 11111111111111111111111111111111, Cout: 0 V: 0
# OPCODE: 110
# A: 10101010101010101010101010101010
# B: 01010101010101010101010101010101
# Result: 01010101010101010101010101010101, Cout: 1 V: 1
# OPCODE: 110
# A: 00000000000000001111111111111111
# B: 11111111111111110000000000000000
# Result: 00000000000000001111111111111111, Cout: 0 V: 0
# OPCODE: 111
# A: 10101010101010101010101010101010
# B: 01010101010101010101010101010101
# Result: 00000000000000000000000000000001, Cout: 1 V: 1
# OPCODE: 111
# A: 00000000000000001111111111111111
# B: 00000000000000000000000000001111
# Result: 00000000000000000000000000000000, Cout: 1 V: 0
# ** Note: $finish    : /home/furkan/Desktop/161044001/thirty_two_bit_alu_bench.v(49)
#    Time: 1200 ps  Iteration: 0  Instance: /thirty_two_bit_alu_bench
# 1
```