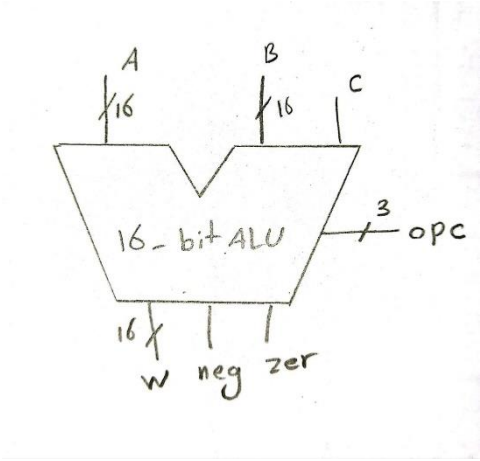
Parisa yahyapour fatideh 810101551

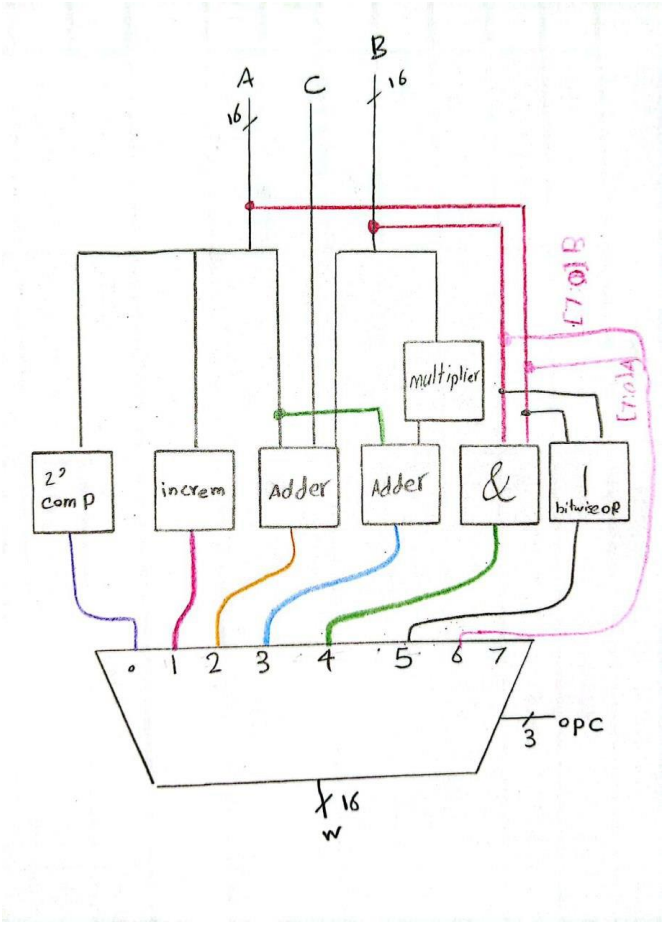
Computer assignment 3 / digital logic design / 367

Question 1:

Based on information that we have from description; we are going to design an ALU same as diagram below:



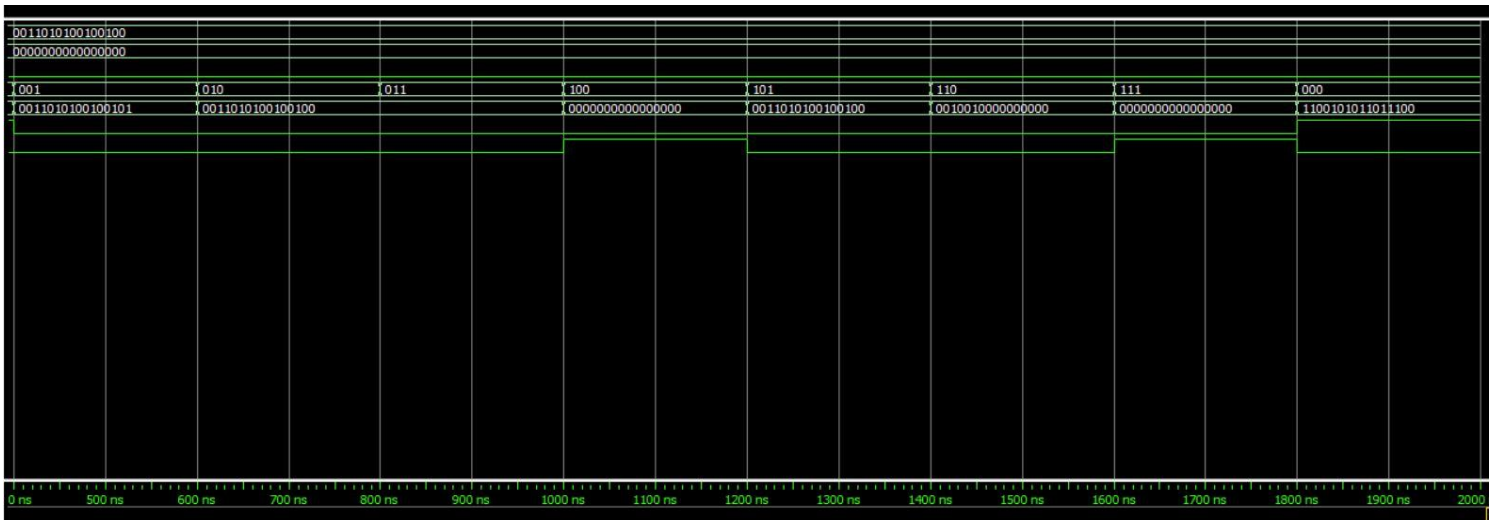
In this part, we are going to design this ALU in behavioral way, which means that we are going to use “always” statement in system Verilog description. So, we are not going to use any special technology or idea for having minimal number of cells or using shared components. ALU will be as below:



Part a) by using a testbench (TB1.sv), we will see the functionality of the ALU. In this testbench, first of all, all of inputs are initialized with zero. Then we will change the amount of input A with a random 16-bit number which will be different if we run the simulation more than once. In this case the random number is 0011010100100100 = aa. A is one of the inputs that our circuit is sensitive of it. So, after changing aa with a random number, “always” statement will be simulated, because opcode is equal to 0, the chosen function is 2’s complement of A. so ww = 1100101011011100. Zer flag will be 0 and neg flag will be 1.

After that we have a “repeat” part in testbench which will change the opcode. So, we can see different functionality of ALU. When opcode changes to 1, ww is incremented A so ww = 0011010100100101. Zer and neg are 0. When opcode changes to 2 or 3, because bb = cc = 0 output will be aa. Neg and zer are still 0. When opcode = 4, ww = 0000000000000000 because bb = 0, zer flag will be 1 and neg still is 0. When opcode = 5, we will see bitwise or on output which is ww = 0011010100100100 which is aa. Opcode = 6 will be a 16-bit binary number that first 8 digits are related to aa and second 8 digits are related to bb. So, the output will be ww = 0010010000000000. Opcode = 7 has no operation, but as “always” statement is sensitive of opcode, it will be simulated. At the beginning of “always” statement, we make our outputs inactive. So, ww = 0000000000000000 and based on this, zer and neg will change to 1 and 0. The last repeat will result in situation that we start the simulation.

Wave form of this testbench is as below:



Part b) yosys results are as below:

```
=== behavioral_ALU ===

   Number of wires:                446
   Number of wire bits:            493
   Number of public wires:           7
   Number of public wire bits:      54
   Number of memories:               0
   Number of memory bits:            0
   Number of processes:              0
   Number of cells:                456
     $_AND_                         58
     $_AOI3_                        47
     $_AOI4_                         4
     $_MUX_                         16
     $_NAND_                        26
     $_NOR_                         64
     $_NOT_                         70
     $_OAI3_                        48
     $_OAI4_                        14
     $_OR_                          18
     $_XNOR_                        76
     $_XOR_                         15
```

```
=== behavioral_ALU ===

   Number of wires:                1186
   Number of wire bits:            1233
   Number of public wires:            7
   Number of public wire bits:       54
   Number of memories:                0
   Number of memory bits:             0
   Number of processes:               0
   Number of cells:                 704
     NAND                           187
     NOR                            371
     NOT                            146
```

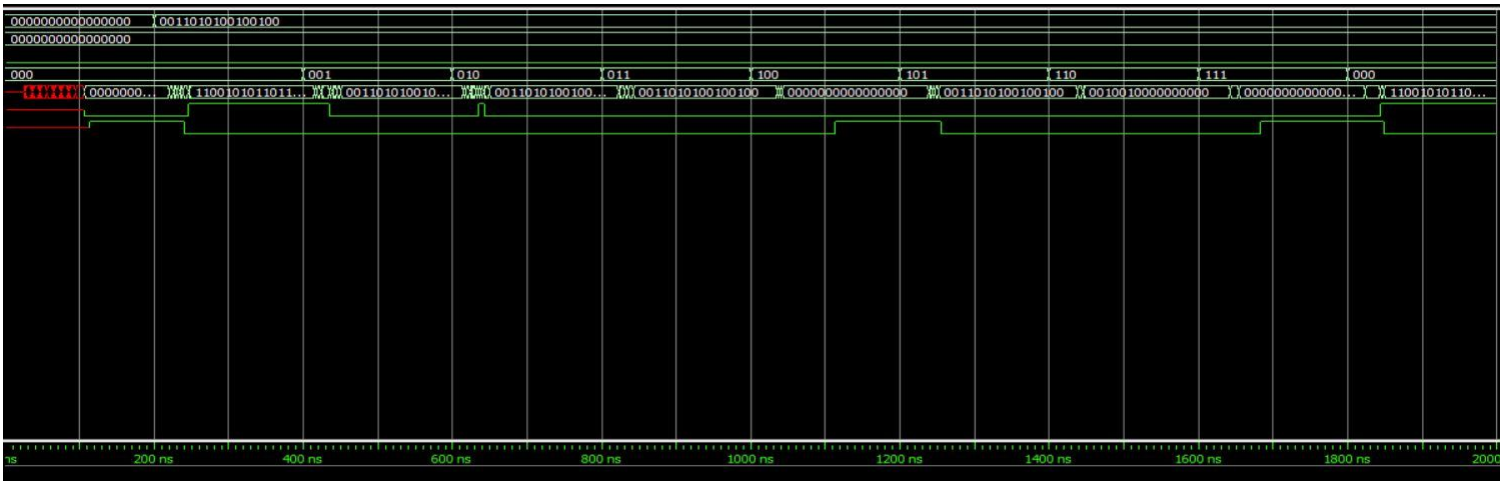
After using mycells.lib (fig.1)

Because we synthesize behavioral description of ALU, number of cells are a lot because there is no shared component. As in our library we just have NOT, NAND and NOR gates, after adding library to yosys, circuit is synthesized with these gates (fig.1). because yosys is forced to use special gates, number of cells will increase after using library. here is a part of netlist:

```
2086    );
2087    NOR _1368_ (
2088      .A(_0596_),
2089      .B(_0592_),
2090      .Y(_0597_)
2091    );
2092    NOT _1369_ (
2093      .A(_0557_),
2094      .Y(_0598_)
2095    );
2096    NAND _1370_ (
2097      .A(_0560_),
2098      .B(_0598_),
2099      .Y(_0599_)
2100    );
2101    NOT _1371_ (
2102      .A(_0464_),
2103      .Y(_0600_)
2104    );
2105    NOR _1372_ (
2106      .A( 0600 ),
```

Fig.2

Part c) when yosys output is simulated, wave forms are as below:



Part d) wave form of part a doesn't have X, because that is same as assign statement, but here as we synthesize the circuit in gate level with yosys, at the beginning of simulation, we don't know what data is on outputs. So, we see X in outputs till the first change occur and the outputs will change. The other parts' outputs are the same as we haven't changed the functionality of ALU (if the random numbers are the same). The main difference is when output changes. In mycells.lib, primitive gates have delay. So, because of them, outputs won't change immediately after changing inputs. For example, when aa sets to a random number, in previous wave form, zer changed immediately but here it will change after 40ns which is related to delay of gates.

Bonus: time of simulation is based on various factors such as number of gates, complexity of designed circuits and etc. for calculating the speed of simulation, there are some commands that can be used in modelsim's transcript. This sequence of commands will actually get the time of starting simulation and end of it, then by subtracting them, time of simulation will be shown. Commands are as below:

set start [clock microseconds];

run -all

set finish [clock microseconds];

puts [expr {$finish - $start}]

for behavioral design that was written with "always" statement, testbench will run in 52936 micro seconds. When the yosys output is on testbench, time of simulation is 89389 micro seconds. Part a is simulating faster than part c.

```
VSIM 21> set start [clock microseconds];
# 1700923356551864
VSIM 22> run -all
VSIM 23> set finish [clock microseconds];
# 1700923356604800
VSIM 24> puts [expr {$finish - $start}]
# 52936
```

Part a's simulation time

```
VSIM 34> set start [clock microseconds];
# 1700923839158568
VSIM 35> run -all
VSIM 36> set finish [clock microseconds];
# 1700923839247957
VSIM 37> puts [expr {$finish - $start}]
# 89389
```

Part c's simulation time

Question 2:

For decreasing number of cells, shared components should be used. It means, we try to do similar mathematic operation with one component. By using a signal and changing inputs with this signal, the component will do different operations.

One of the things we can do, is to use one component for opcode 0 and 1 which are 2'complement and incrementing. S1 is defined for choosing the function (signal), new_a is where the input changes, based on signal. So, n1 will be new data that will be calculated with one complement.With this change number of cells will decrease significantly:

```
   Number of wires:                404
   Number of wire bits:            451
   Number of public wires:           7
   Number of public wire bits:      54
   Number of memories:               0
   Number of memory bits:            0
   Number of processes:              0
   Number of cells:                414
     $_AND_                         41
     $_AOI3_                        34
     $_AOI4_                        15
     $_MUX_                         16
     $_NAND_                        36
     $_NOR_                         46
     $_NOT_                         62
     $_OAI3_                        46
     $_OR_                          25
     $_XNOR_                        64
     $_XOR_                         29

2.24. Executing CHECK pass (checking for obvious problems).
checking module structural_ALU..
found and reported 0 problems.
```

Fig.3

Here 414 cells are used, in pre-synthesis it was 456. So, 42 cells less than before.

The other change that can be used is to use one adder for opcode of 2 and 3. This operations have one same operand which is A and based on opcode the second operand differs. So, an assign statement can be used for choosing the second operand based on opcode which is s2. Another assign statement is required for the result of these two opcodes which is n2 which is result of adding A to s2. The “always” statement should be sensitive of n2 too. By giving this design to yosys number of cells will be as below:

```
=== structural_ALU_1 ===

   Number of wires:                376
   Number of wire bits:            423
   Number of public wires:           7
   Number of public wire bits:      54
   Number of memories:               0
   Number of memory bits:            0
   Number of processes:              0
   Number of cells:                386
     $_AND_                         50
     $_AOI3_                        29
     $_MUX_                         30
     $_NAND_                        36
     $_NOR_                         48
     $_NOT_                         50
     $_OAI3_                        42
     $_OR_                          19
     $_XNOR_                        78
     $_XOR_                          4

2.24. Executing CHECK pass (checking for obvious problems).
checking module structural_ALU_1..
found and reported 0 problems.
```

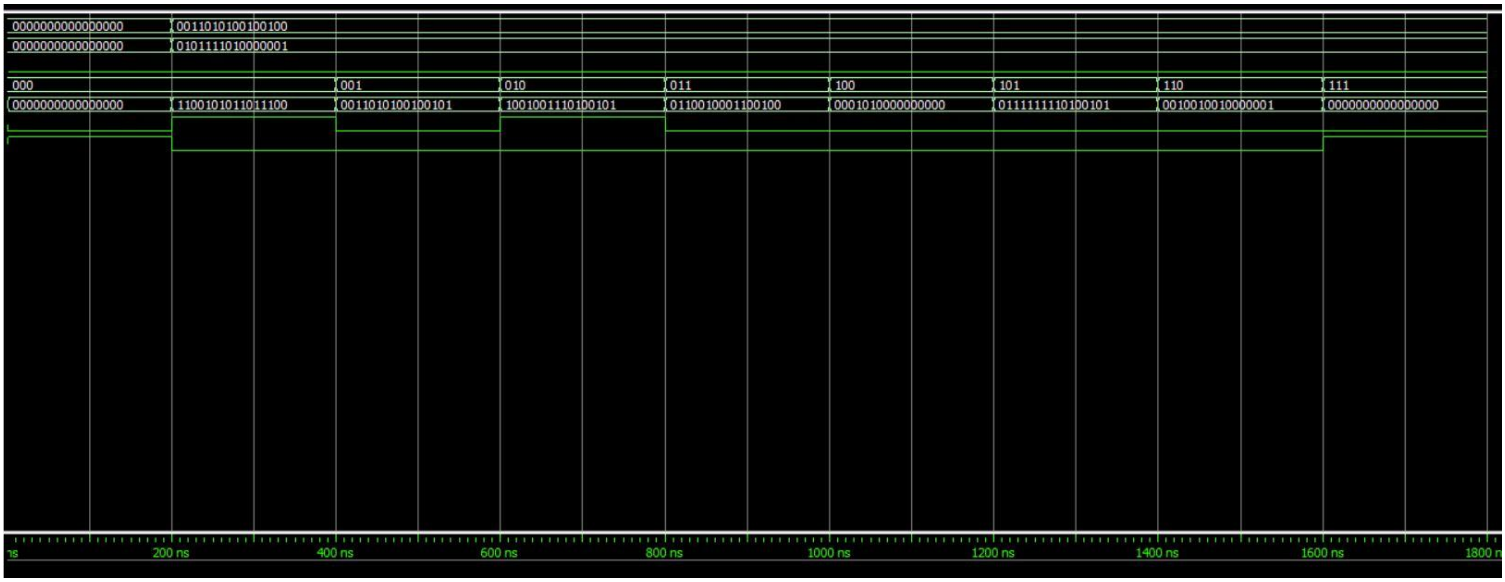
```
=== structural_ALU_1 ===

   Number of wires:                1066
   Number of wire bits:            1113
   Number of public wires:            7
   Number of public wire bits:       54
   Number of memories:                0
   Number of memory bits:             0
   Number of processes:               0
   Number of cells:                 654
     NAND                           179
     NOR                            354
     NOT                            121
```

Fig.4

Number of cells will decrease because one adder is deleted from design and two opcodes will calculate output with just one shared component. Before number of cells was 414 and now it’s 386 that is 28 less than before. Totally by using shared components 70 cells are saved. If we want to consider library, behavioral design has 704 cells but structural design has 654 cells.

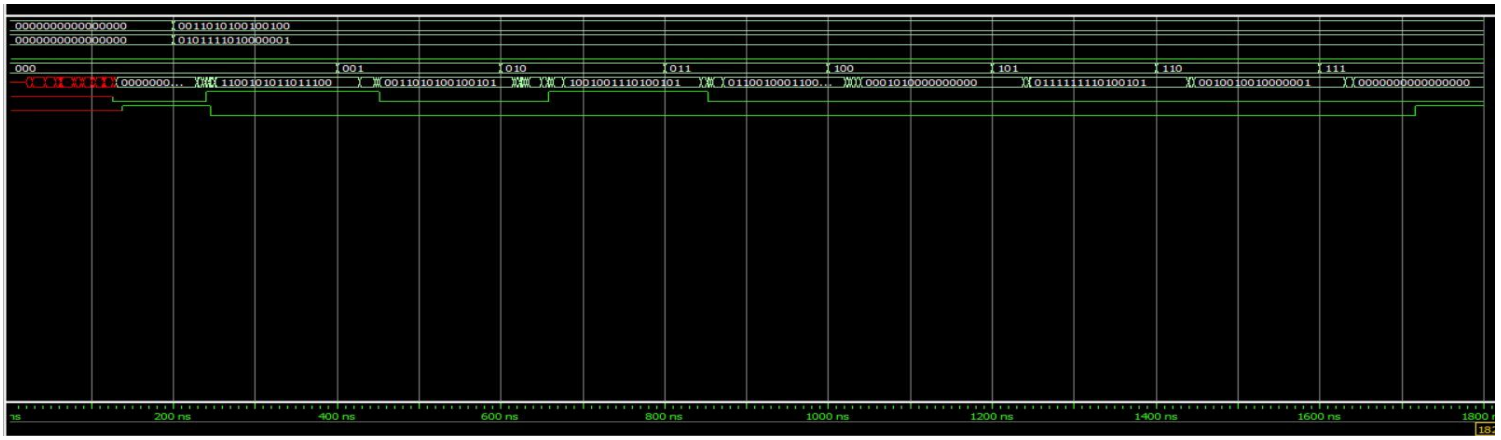
Part a) in this testbench by setting aa and bb to a random number, functionality of circuit will be tested. As it’s shown in wave forms, all opcodes are tested and the results are true. Because there is no delay, all outputs will change immediately after changing inputs. Wave forms are as below:



Part b) as it was described, by changing design and using shared components, number of cells decreased to 386. Because of using mycells.lib which provides available gates, in net list there are NAND, NOR and NOT gates. A part of netlist is as below:

```
1917    );
1918    NOT _1236_ (
1919      .A(_0514_),
1920      .Y(_0515_)
1921    );
1922    NOR _1237_ (
1923      .A(_0515_),
1924      .B(_0506_),
1925      .Y(_0516_)
1926    );
1927    NOR _1238_ (
1928      .A(_0514_),
1929      .B(_0374_),
1930      .Y(_0517_)
1931    );
1932    NOR _1239_ (
1933      .A(_0517_),
1934      .B(_0516_),
1935      .Y(_0518_)
1936    );
1937    NAND _1240_ (
1938      .A(_0441_),
1939      .B(_0422_),
1940      .Y(_0519_)
1941    );
1942    NAND _1241_ (
1943      .A(_0519_),
1944      .B(_0432_),
1945      .Y(_0520_)
1946    );
```

Part c) wave form of testing yosys's output is as below:



It's clear that because in used library, primitive gates have delay, outputs, at the beginning, are X till the first outputs occurred based on initialized number for inputs. The outputs are same as previous waveform and the difference is just the time of changing outputs which is related to gates' delay.

Part d) **bonus:** same as question 1, time of simulation is calculated with described commands. The result is as below:

```
VSIM 47> set start [clock microseconds];
# 1700924913033262
VSIM 48> run -all
VSIM 49> set finish [clock microseconds];
# 1700924913099475
VSIM 50> puts [expr {$finish - $start}]
# 66213
```

```
VSIM 60> set start [clock microseconds];
# 1700926099718466
VSIM 61> run -all
VSIM 62> set finish [clock microseconds];
# 1700926099830556
VSIM 63> puts [expr {$finish - $start}]
# 112090
```

Part a will be simulated in 66213 micro seconds and yosys's output will be simulated in 112090 micro seconds.

Question 3:

Until here, two different designs of the ALU are shown. The diagram of components in two designs are as below:

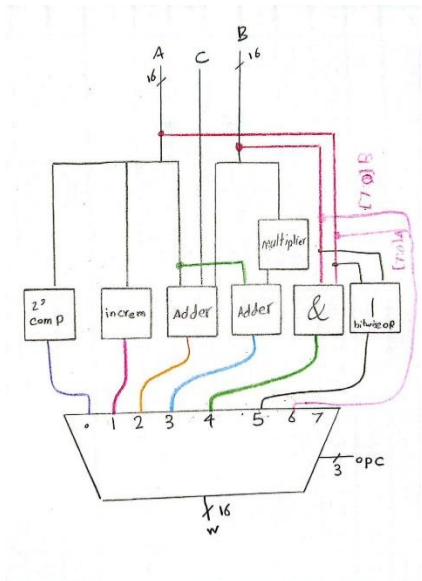


Fig.5 Behavioral design

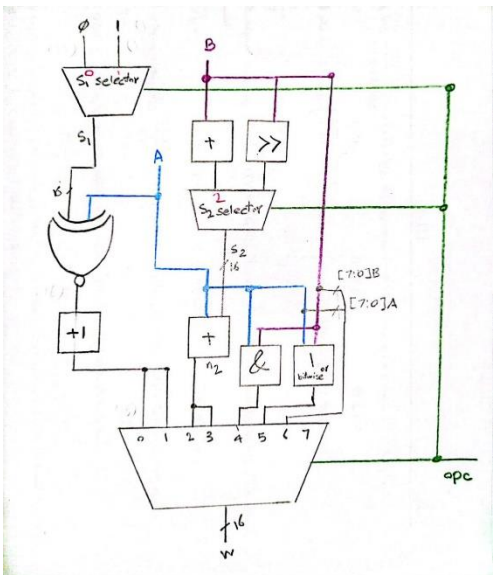
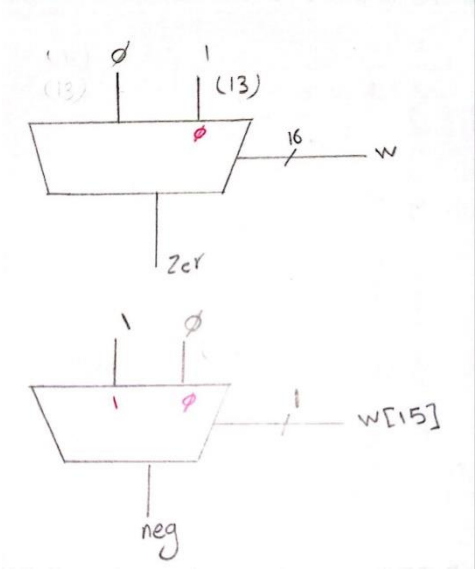


Fig.6 Structural design

For zer and neg flags both ALUs are connected to two components same as below:




```
=== behavioral_ALU ===

   Number of wires:                 1186
   Number of wire bits:             1233
   Number of public wires:             7
   Number of public wire bits:        54
   Number of memories:                 0
   Number of memory bits:              0
   Number of processes:                0
   Number of cells:                  704
     NAND                            187
     NOR                             371
     NOT                             146
```

Fig.7 behavioral design

```
=== structural_ALU_1 ===

   Number of wires:                 1066
   Number of wire bits:             1113
   Number of public wires:             7
   Number of public wire bits:        54
   Number of memories:                 0
   Number of memory bits:              0
   Number of processes:                0
   Number of cells:                  654
     NAND                            179
     NOR                             354
     NOT                             121
```

Fig.8 structural design

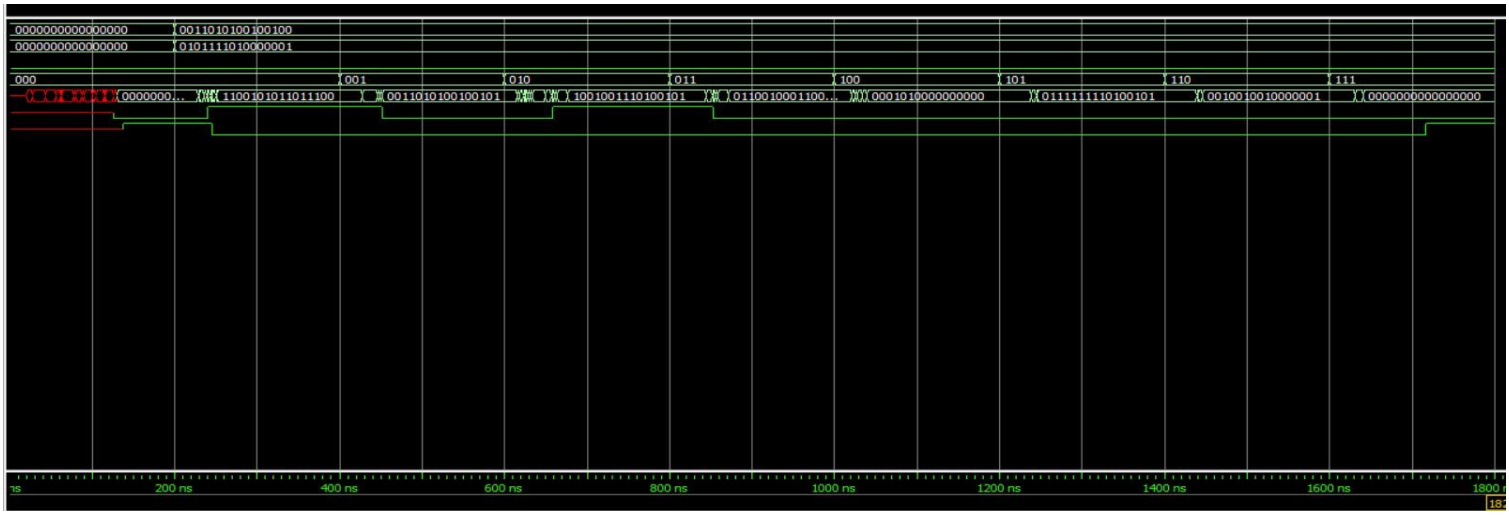


Fig.9 behavioral design

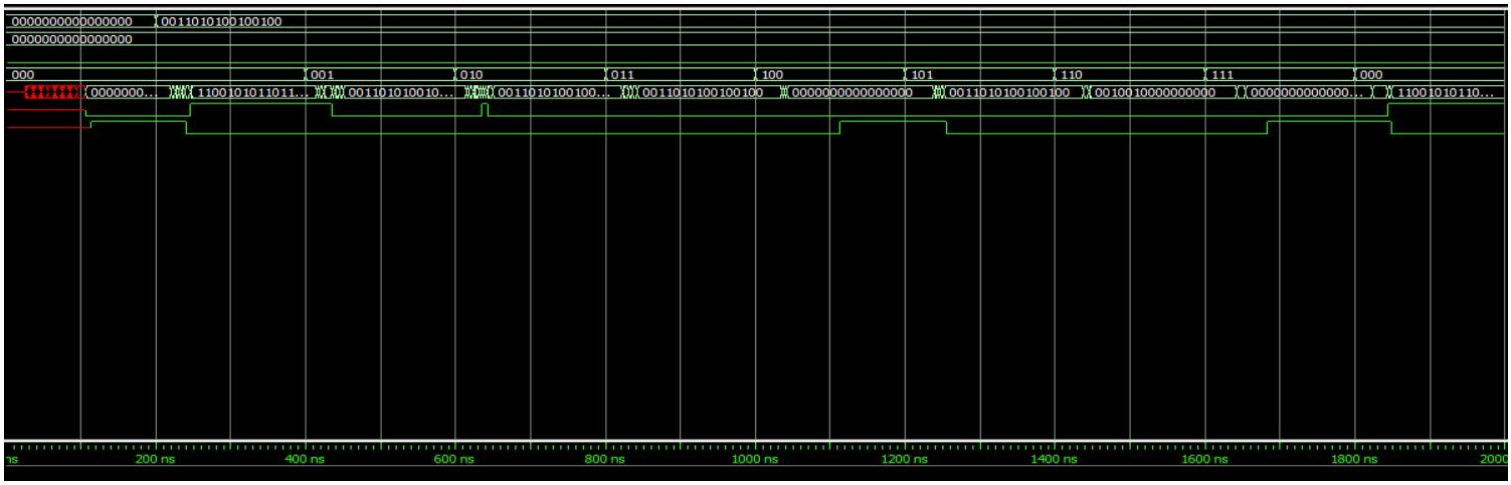


Fig.10 structural design

By using technology of shared components, number of cells and components have been decreased as it can be recognized from pictures. Speed of simulation in structural design is more because the used components are more complex. Number of cells are significantly different. In behavioral design number of cells are 704 but in structural design is 654. So, based on hardware usage, the structural design is better. Testbench of structural and behavioral design are different so, waveforms are different too.