CA3-Report

Digital Systems I

Spring 1402-03
Digital Systems I – ECE894

Amirmortaza Rezaee
810101429

Problem 1

a. Pre-synthesize:

Figures 1.1 and 1.2 illustrates the Verilog code and testbench for the behavioral description of this ALU:

```
module myALU(input signed [15:0] inM , inN , input inC , input [2:0] opc ,
             output logic [15:0] outF , output logic zer , neg);

    always @ (inM , inN , inC , opc) begin
        outF = 16'b0;
        case (opc)
            3'd0: {outF} = inM + inN + inC;
            3'd1: {outF} = inM + (inN>>>1);
            3'd2: outF = (inM>inN) ? inM : inN;
            3'd3: {outF} = inM + (inM<<<1);
            3'd4: outF = inM & inN;
            3'd5: outF = inM | inN;
            3'd6: outF = ~inM;
            3'd7: outF = 16'bZ;
           default outF = 16'b0;
        endcase
    end
    assign zer = ~|outF ;
    assign neg = outF[15];
endmodule
```

Figure 1.a.1

```
`timescale 1ns/1ns
module Q1aTB ();
    logic [15:0] MM = 16'b0;
    logic [15:0] NN = 16'b0;
    logic CC = 1'b0;
    logic [2:0] OO =3'b111;
    wire [15:0] FF;
    wire ZZ , NG;
    myALU CUT1( MM , NN , CC , OO , FF , ZZ , NG);
    initial begin
        repeat(8) begin
            #100 OO=OO+1;
                 MM=$random;
                 NN=$random;
                 CC=$random;
        end
        #100 $stop;
    end
endmodule
```

Figure 1.a.2

Let's analyze the simulation's results:

1) *inM= (0011010100100100)$_B$ = (13604) $_D$*
   inN= (0101111010000001) $_B$ = (24193) $_D$
   inC= 1
   opc= 000 -> inM + inN + inC
   So, the result should be like:
   outF= (1001001110100110), neg= 1, zer= 0, here overflow occurs! But the unsigned summation done correctly!

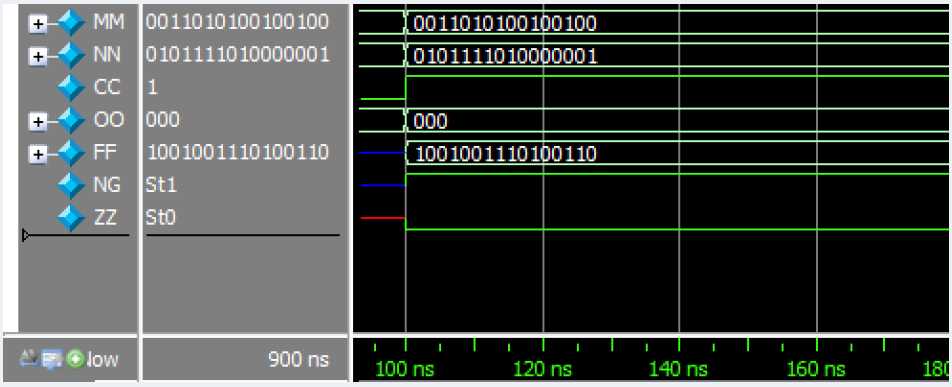

Figure 1.a.3

2) in *M= (0101011001100011)$_B$ = (22115) $_D$*
   inN= (0111101100001101) $_B$ = (31501) $_D$
   inC= 1
   opc= 001 -> inM + inN ÷ 2
   So, the result should be like:
   outF= (1001001111101001), neg= 1, zer= 0, here overflow occurs! But the unsigned summation done correctly!

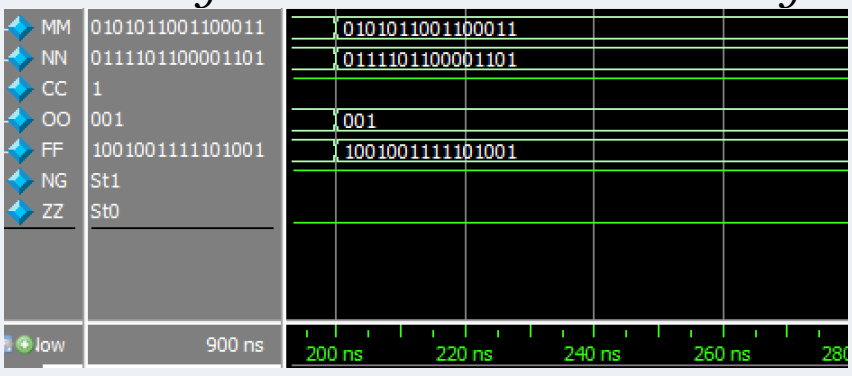

Figure 1.a.4

3) in *M= (1000010001100101)$_B$ = (-31643)$_D$*
inN= (0101001000010010)$_B$ = (21010)$_D$
inC= 1
opc= 010 -> MAX (inM, inN)
So, the result should be like:
outF= (0101001000010010)$_B$ = (21010)$_D$, neg= 0, zer= 0

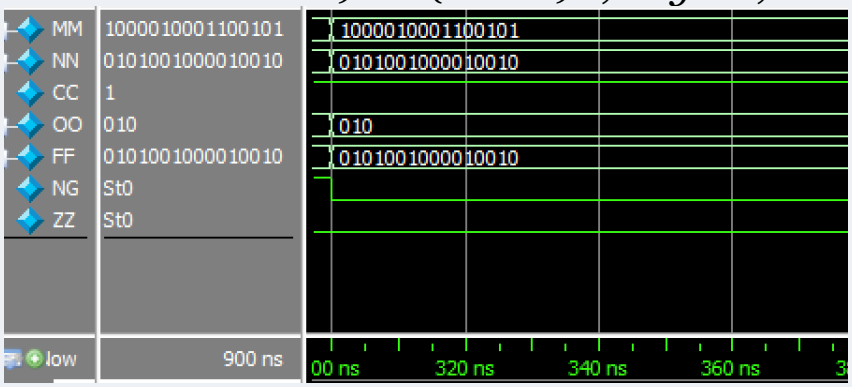

Figure 1.a.5

4) in *M= (1100110100001101)$_B$ = (-13043)$_D$*
inN= (1111000101110110)$_B$ = (-3722)$_D$
inC= 1
opc=011 -> inM×3
So, the result should be like:
outF= (0110011100100111)$_B$, neg= 0, zer= 0, here overflow occurs! But the unsigned summation done correctly!

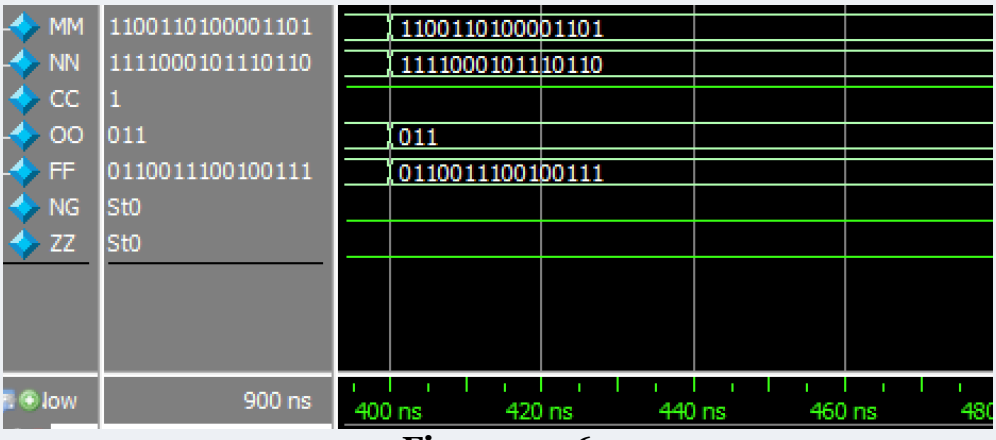

Figure 1.a.6

5) in *M= (0101011111101101)$_B$ = (22509)$_D$*
inN= (1111011110001100)$_B$ = (-2164)$_D$
inC= 1
opc= 100 -> Bitwise ANDing
So, the result should be like:
outF= (0101011110001100)$_B$, neg= 0, zer= 0

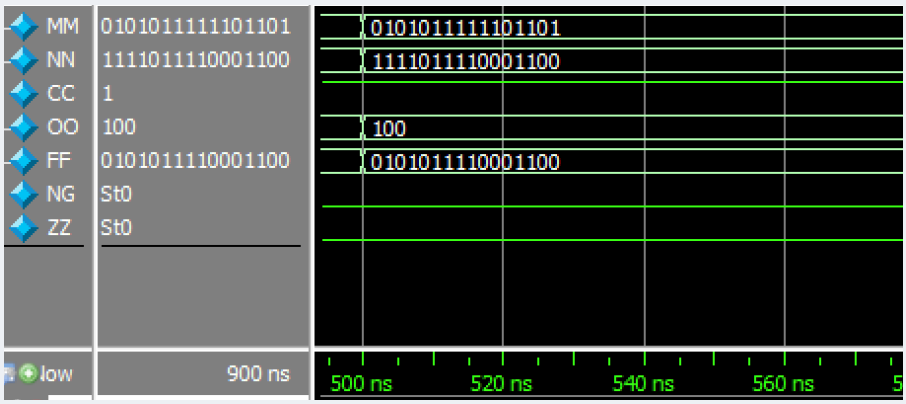

Figure 1.a.7

6) in *M= (0010010011000110)$_B$ = (9414)$_D$*
inN= (1000010011000101)$_B$ = (-31547)$_D$
inC= 0
opc= 101 -> Bitwise ORing
So, the result should be like:
outF= (1010010011000111)$_B$, neg= 1, zer= 0

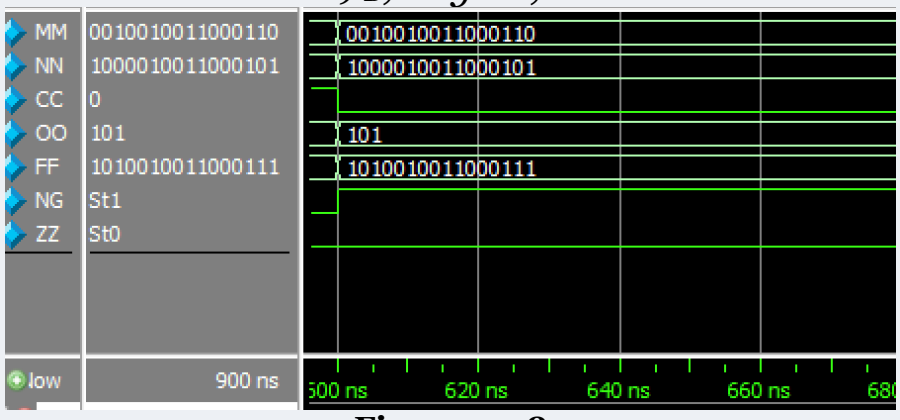

Figure 1.a.8

7) in *M= (1111011111100101)$_B$ = (-2075)$_D$*
inN= (0111001001110111)$_B$ = (29303)$_D$
inC= 0
opc= 110 -> Bitwise Inverting inM
So, the result should be like:
outF= (0000100000011010)$_B$, neg= 0, zer= 0

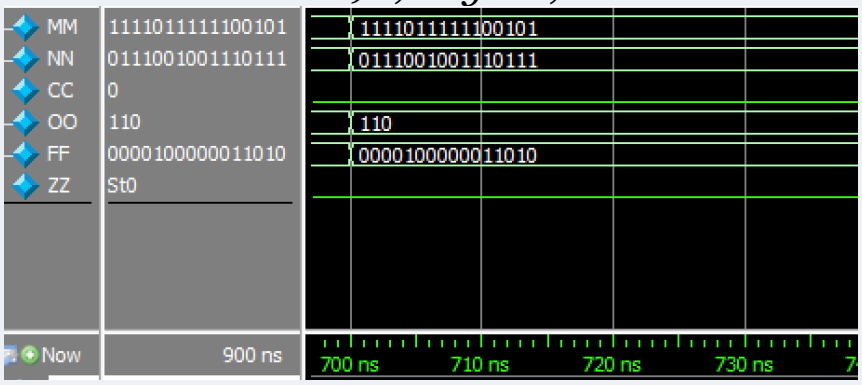

Figure 1.a.9

8) in *M= (1101101110001111)$_B$ = (-9329)$_D$*
inN= (0110100111110010)$_B$ = (27122)$_D$
inC= 0
opc= 111 -> No Operation
So, the result should be like:
outF= Z, neg= Z, zer= X

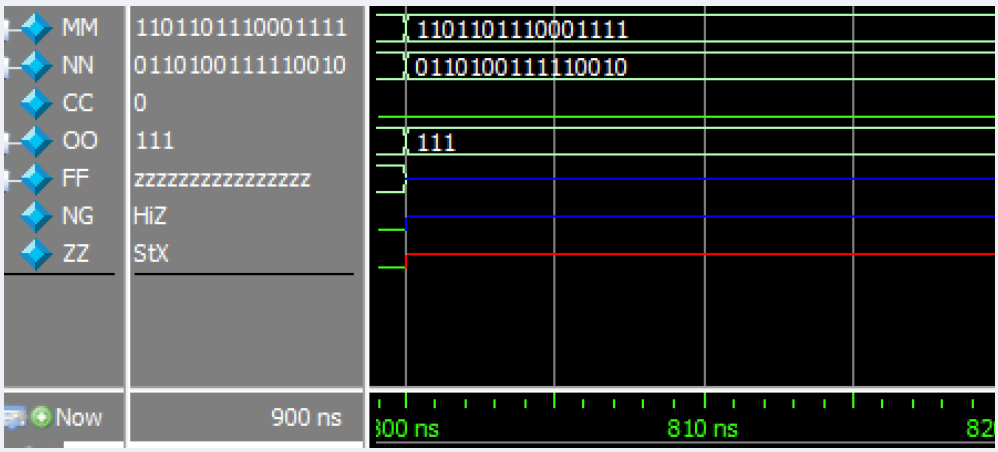

Figure 1.a.10

b. synthesize:

synthesizing progress shown in the figures below:

```
yosys> read_verilog -sv CA3Q1.sv
1. Executing Verilog-2005 frontend.
Parsing SystemVerilog input from `CA3Q1.sv' to AST representation.
Warning: Yosys has only limited support for tri-state logic at the moment. (CA3Q1.sv:14)
Generating RTLIL representation for module `\myALU'.
Note: Assuming pure combinatorial block at CA3Q1.sv:4 in
compliance with IEC 62142(E):2005 / IEEE Std. 1364.1(E):2002. Recommending
use of @* instead of @(...) for better match of synthesis and simulation.
Successfully finished Verilog frontend.
```

Figure 1.b.1

```
=== myALU ===

   Number of wires:                522
   Number of wire bits:            569
   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:                532
     $_AND_                         47
     $_AOI3_                        71
     $_AOI4_                         3
     $_MUX_                         33
     $_NAND_                        62
     $_NOR_                         67
     $_NOT_                         66
     $_OAI3_                        61
     $_OAI4_                        17
     $_OR_                          14
     $_XNOR_                        61
     $_XOR_                         30

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

Figure 1.b.2

```
yosys> dfflibmap -liberty mycells.lib

3. Executing DFFLIBMAP pass (mapping DFF cells to sequential cells from liberty file).
  cell DFF (noninv, pins=3, area=18.00) is a direct match for cell type $_DFF_P_.
  cell DFF_PP0 (noninv, pins=4, area=0.00) is a direct match for cell type $_DFF_PP0_.
  cell DFF_PP1 (noninv, pins=4, area=0.00) is a direct match for cell type $_DFF_PP1_.
  create mapping for $_DFF_PN0_ from mapping for $_DFF_PP0_.
  create mapping for $_DFF_PN1_ from mapping for $_DFF_PP1_.
  create mapping for $_DFF_N_ from mapping for $_DFF_P_.
  create mapping for $_DFF_NN0_ from mapping for $_DFF_PN0_.
  create mapping for $_DFF_NP0_ from mapping for $_DFF_NN0_.
  create mapping for $_DFF_NN1_ from mapping for $_DFF_NN0_.
  create mapping for $_DFF_NP1_ from mapping for $_DFF_NN1_.
  final dff cell mappings:
    DFF _DFF_N_ (.C(~C), .D( D), .Q( Q));
    DFF _DFF_P_ (.C( C), .D( D), .Q( Q));
    DFF_PP0 _DFF_NN0_ (.C(~C), .D( D), .Q( Q), .R(~R));
    DFF_PP0 _DFF_NN1_ (.C(~C), .D(~D), .Q(~Q), .R(~R));
    DFF_PP0 _DFF_NP0_ (.C(~C), .D( D), .Q( Q), .R( R));
    DFF_PP0 _DFF_NP1_ (.C(~C), .D(~D), .Q(~Q), .R( R));
    DFF_PP0 _DFF_PN0_ (.C( C), .D( D), .Q( Q), .R(~R));
    DFF_PP1 _DFF_PN1_ (.C( C), .D( D), .Q( Q), .R(~R));
    DFF_PP0 _DFF_PP0_ (.C( C), .D( D), .Q( Q), .R( R));
    DFF_PP1 _DFF_PP1_ (.C( C), .D( D), .Q( Q), .R( R));
    unmapped dff cell: $_DFFSR_NNN_
    unmapped dff cell: $_DFFSR_NNP_
    unmapped dff cell: $_DFFSR_NPN_
    unmapped dff cell: $_DFFSR_NPP_
    unmapped dff cell: $_DFFSR_PNN_
    unmapped dff cell: $_DFFSR_PNP_
    unmapped dff cell: $_DFFSR_PPN_
    unmapped dff cell: $_DFFSR_PPP_
Mapping DFF cells in module `\myALU':
```

Figure 1.b.3

```
4.1.2. Re-integrating ABC results.
ABC RESULTS:               NAND cells:      172
ABC RESULTS:                NOR cells:      550
ABC RESULTS:                NOT cells:      180
ABC RESULTS:        internal signals:      515
ABC RESULTS:           input signals:       36
ABC RESULTS:          output signals:       17
Removing temp directory.

yosys> clean
Removed 0 unused cells and 568 unused wires.

yosys> write_verilog myALUsynth.v

5. Executing Verilog backend.
Dumping module `\myALU'.
```

Figure 1.b.4

Actually, Yosys synthesizes a circuit like:

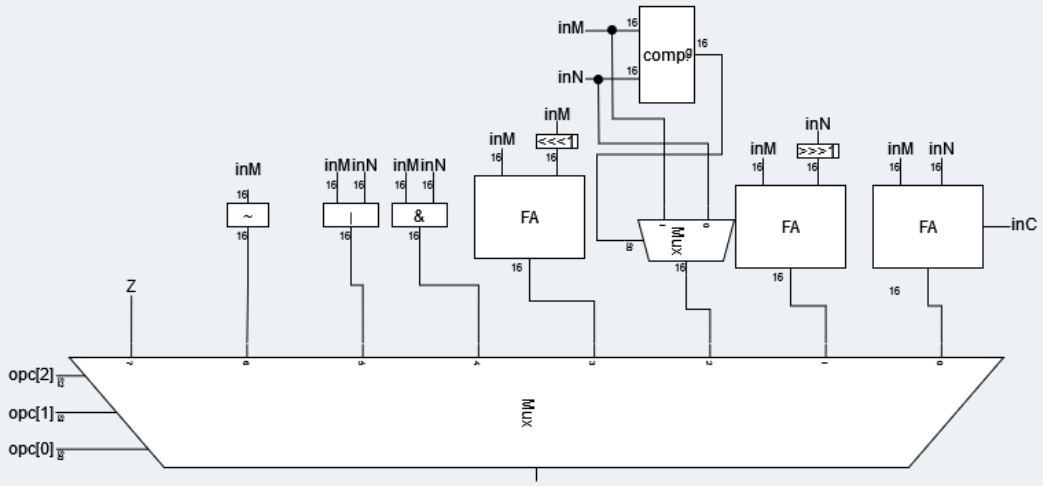

Figure 1.b.5

Now let's count the approximate number of gates for this structure:

1- *Each 16-bit full adder formed by 16 slices of 1-bit FAs in which there are 2 XOR gates (that can be represented by 4 2-input NANDs) and 2 2-input ANDs and a 2-input OR that can be replaced with 3 2-input NANDs.*
   So, for each FA: $16 \times (8 + 3) = 224 \rightarrow for\ 3\ FAs = 528$
2- *A 16-bit comparator consists of 4 NANDs which form a XOR and a NORs and an inverter. (* $= 16 \times 6 = 96$*)*
3- *A 16-bit 2-to-1 MUX built up by 16 layers of 1-bit MUXs, which is formed by 3 NANDs and an inverter. (* $=16 \times 4 = 64$*)*
4- *For 16-bit Bitwise ANDing (and ORing), we need 16 slices of 2-input NANDs (NORs) followed by an inverter. (*$=16 \times 2 \times 2 = 64$*)*
5- *For 16-bit inverting there will be 16 inverters.*
6- *A 16-bit 8-to-1 MUX, consists of 16 layers in which there are about 20 NOR, NAND and inverters. (*$= 13 \times 20 = 260$*)*

So, this structure, contains about 1000 gates. As it's obvious, this number of gates is ***approximately equal*** *to the number of gates that Yosys had synthesized.*

c. Post-synthesize:

Figure 1.c.1 illustrates the testbench that had been written for the Yosys's output:

```
`timescale 1ns/1ns
module Q1cTB ();
    logic [15:0] MM = 16'b0;
    logic [15:0] NN = 16'b0;
    logic CC = 1'b0;
    logic [2:0] OO =3'b111;
    wire [15:0] FF;
    wire ZZ , NG;
    myALUsynth CUT2( MM , NN , CC , OO , FF , ZZ , NG);
    initial begin
        repeat(8) begin
            #500 OO=OO+1;
                    MM=$random;
                    NN=$random;
                    CC=$random;
        end
        #500 $stop;
    end
endmodule
```

Figure 1.c.1

Regarding the fact that the result of each function for each bit, produced parallelly and not sequentially, and by assuming a 5 NS delay for each gate, lets derive some of the delay values we expect to see:

For the adder each bit's path contains about 5 gates -> 25 NS

For the last big MUX each bit's path contains about 5 gates -> 25 NS

So there will be a 50 NS delay for the functions that use an adder!

d.

The simulation time for part a (200000 reptations):

```
VSIM 239> simstats
# Memory Statistics
#     mem: size after elab (VSZ)                141.87 Mb
#     mem: size during sim (VSZ)                173.78 Mb
# Elaboration Time
#    elab: wall time                              0.19 s
#    elab: cpu time                               0.03 s
# Simulation Time
#     sim: wall time                              1.37 s
#     sim: cpu time                               0.33 s
# Tcl Command Time
#     cmd: wall time                             15.16 s
#     cmd: cpu time                               0.08 s
# Total Time
#   total: wall time                             16.72 s
#   total: cpu time                              0.44 s
#
```

Figure 1.d.1

The simulation time for part c (200000 reptations):

```
VSIM 250> simstats
# Memory Statistics
#     mem: size after elab (VSZ)                141.89 Mb
#     mem: size during sim (VSZ)                173.56 Mb
# Elaboration Time
#    elab: wall time                              0.24 s
#    elab: cpu time                               0.11 s
# Simulation Time
#     sim: wall time                             19.00 s
#     sim: cpu time                               5.53 s
# Tcl Command Time
#     cmd: wall time                             19.91 s
#     cmd: cpu time                               0.13 s
# Total Time
#   total: wall time                             39.14 s
#   total: cpu time                              5.77 s
#
```

Figure 1.d.2

Problem 2

a. Pre-synthesize:

The manually designed ALU:

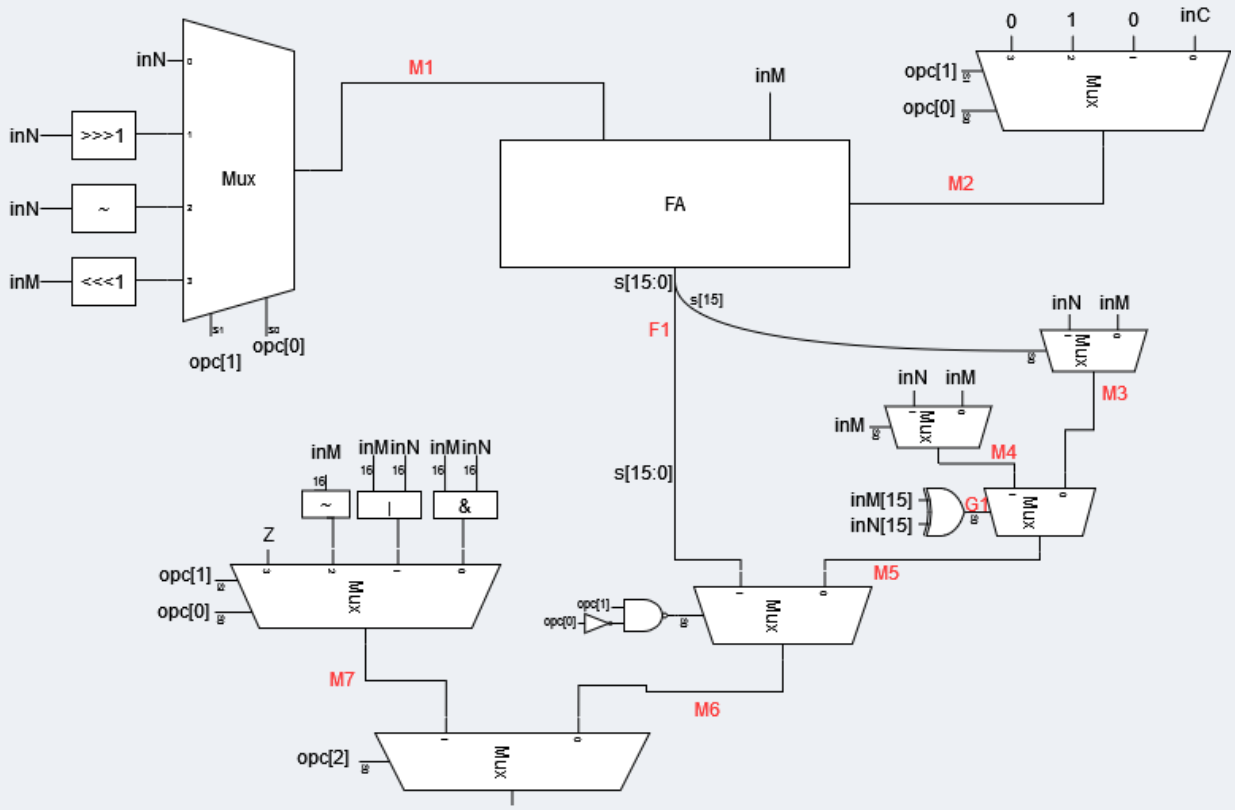

Figure 2.a.1

Figures 2.a.2 and 2.a.3 illustrates the Verilog code and testbench for the hardware-oriented description of this ALU:

```
module   myALUopt(input signed [15:0] inM , inN , input inC , input [2:0] opc ,
                  output logic [15:0] outF , output logic zer , neg);

    logic [15:0] M1 , M5 , M3 , M4, M6 , M7 , F1;
    logic M2 , G1 , G2;
    assign M1 = ({opc[1],opc[0]} == 2'd0) ? inN:
                ({opc[1],opc[0]} == 2'd1) ? (inN>>>1):
                ({opc[1],opc[0]} == 2'd2) ? (~inN):
                (inM<<<1);

    assign M2 = ({opc[1],opc[0]} == 2'd0) ? inC:
                ({opc[1],opc[0]} == 2'd1) ? 1'b0:
                ({opc[1],opc[0]} == 2'd2) ? 1'b1:
                1'b0;

    assign F1 = M1 + inM + M2;

    assign M3 = (F1[15]==1'b1) ? inN : inM;

    assign M4 = (inN[15]==1'b1) ? inM : inN;

    assign G1 = ((inM[15] & ~inN[15]) | (~inM[15] & inN[15]));

    assign M5 = (G1==1'b1) ? M4 : M3;

    assign G2 = ~((~opc[0]) & opc[1]);

    assign M6 = (G2==1'b1) ? F1 : M5;
    assign M7 = ({opc[1],opc[0]} == 2'd0) ? (inM&inN):
                ({opc[1],opc[0]} == 2'd1) ? (inM|inN):
                ({opc[1],opc[0]} == 2'd2) ? ~inM : 16'bz;

    assign outF = (opc[2]) ? M7 : M6;
    assign zer = ~|outF ;
    assign neg = outF[15];
endmodule
```

Figure 2.a.2

```
`timescale 1ns/1ns
module Q2aTB ();
    logic [15:0] MM = 16'b0;
    logic [15:0] NN = 16'b0;
    logic CC = 1'b0;
    logic [2:0] OO =3'b111;
    wire [15:0] FF;
    wire ZZ , NG;
    myALUopt CUT4( MM , NN , CC , OO , FF , ZZ , NG);
    initial begin
        repeat(8) begin
            #100 OO=OO+1;
                 MM=$random;
                 NN=$random;
                 CC=$random;
        end
        #100 $stop;
    end
endmodule
```

Figure 2.a.3

- *Simulation results are just as like as the behavioral structure!*

b. synthesize:

synthesizing progress shown in the figures below:

```
yosys> read_verilog -sv CA3Q2a.sv
1. Executing Verilog-2005 frontend.
Parsing SystemVerilog input from `CA3Q2a.sv' to AST representation.
Warning: Yosys has only limited support for tri-state logic at the moment. (CA3Q2a.sv:31)
Generating RTLIL representation for module `\myALUopt'.
Successfully finished Verilog frontend.
```

Figure 2.b.1

```
=== myALUopt ===

   Number of wires:                357
   Number of wire bits:            404
   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:                367
     $_AND_                         19
     $_AOI3_                        12
     $_MUX_                        142
     $_NAND_                        11
     $_NOR_                         36
     $_NOT_                         60
     $_OAI3_                        22
     $_OR_                          31
     $_XNOR_                        27
     $_XOR_                          7

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

Figure 2.b.2

```
4.1.2. Re-integrating ABC results.
ABC RESULTS:               NAND cells:      144
ABC RESULTS:                NOR cells:      473
ABC RESULTS:                NOT cells:      142
ABC RESULTS:        internal signals:      350
ABC RESULTS:           input signals:       36
ABC RESULTS:          output signals:       17
Removing temp directory.

yosys> clean
Removed 0 unused cells and 403 unused wires.

yosys> write_verilog Q2synth.v

5. Executing Verilog backend.
Dumping module `\myALUopt'.
```

Figure 2.b.3

- *In this structure, Yosys synthesized the ALU with 759 gates, it means about 143 gates less than the previous part!*
- *This is because the number of adders has been decreased to 1 and the comparator replaced with multiple 2-to-1 MUXs!*

c. Post-synthesize:

Figure 2.c.1 illustrates the testbench that had been written for the Yosys's output:

```
`timescale 1ns/1ns
module Q2cTB ();
    logic [15:0] MM = 16'b0;
    logic [15:0] NN = 16'b0;
    logic CC = 1'b0;
    logic [2:0] OO =3'b111;
    wire [15:0] FF;
    wire ZZ , NG;
    myALUoptSynth CUT8( MM , NN , CC , OO , FF , ZZ , NG);
    initial begin
        repeat(8) begin
            #500 OO=OO+1;
                 MM=$random;
                 NN=$random;
                 CC=$random;
        end
        #500 $stop;
    end
endmodule
```

Figure 2.c.1

Figures below, illustrate the simulation results:

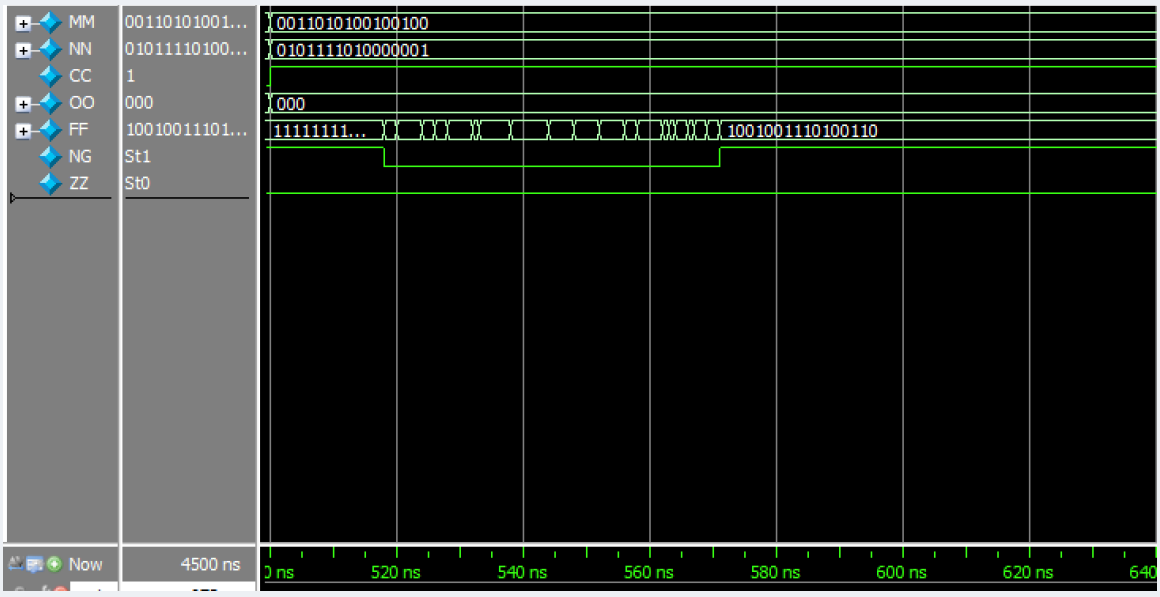

Figure 2.c.2

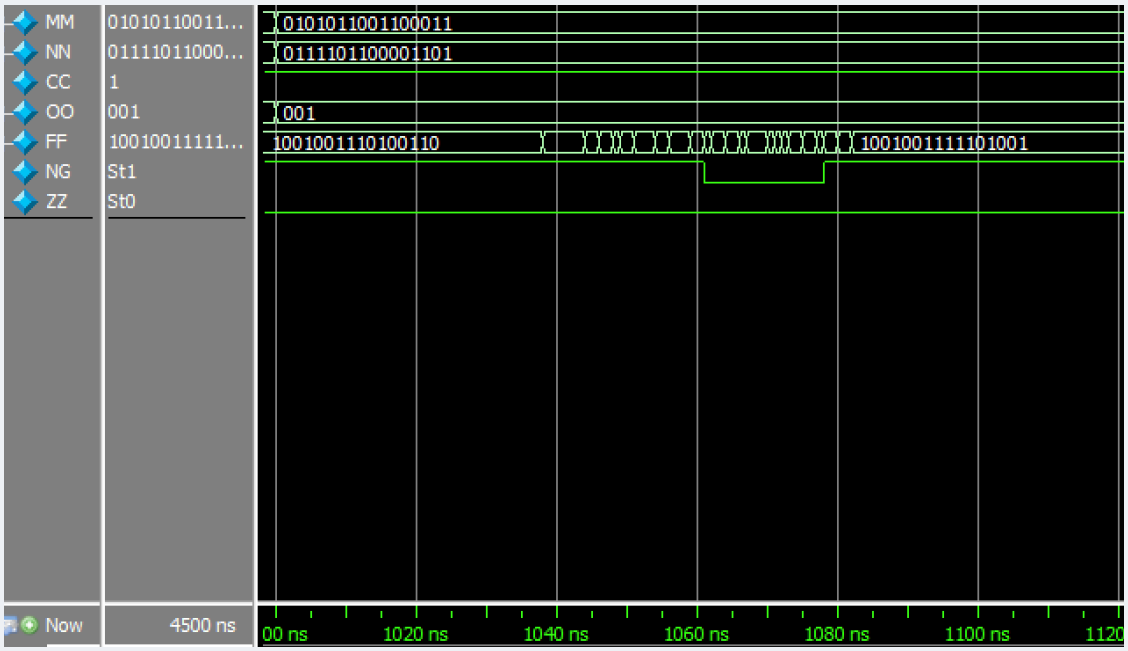

Figure 2.c.3

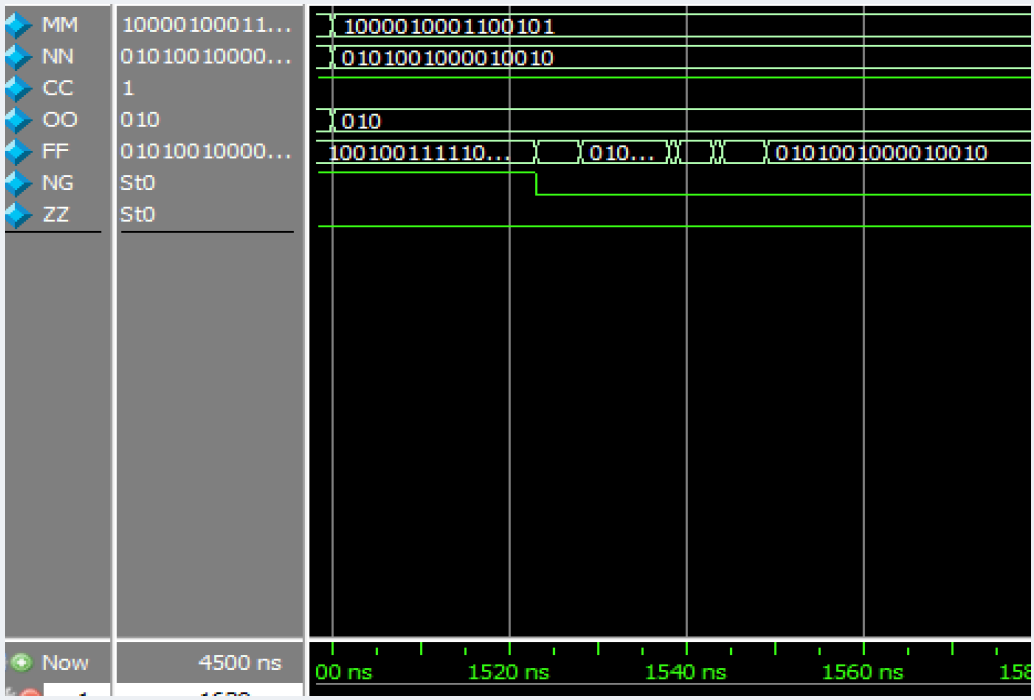

Figure 2.c.4

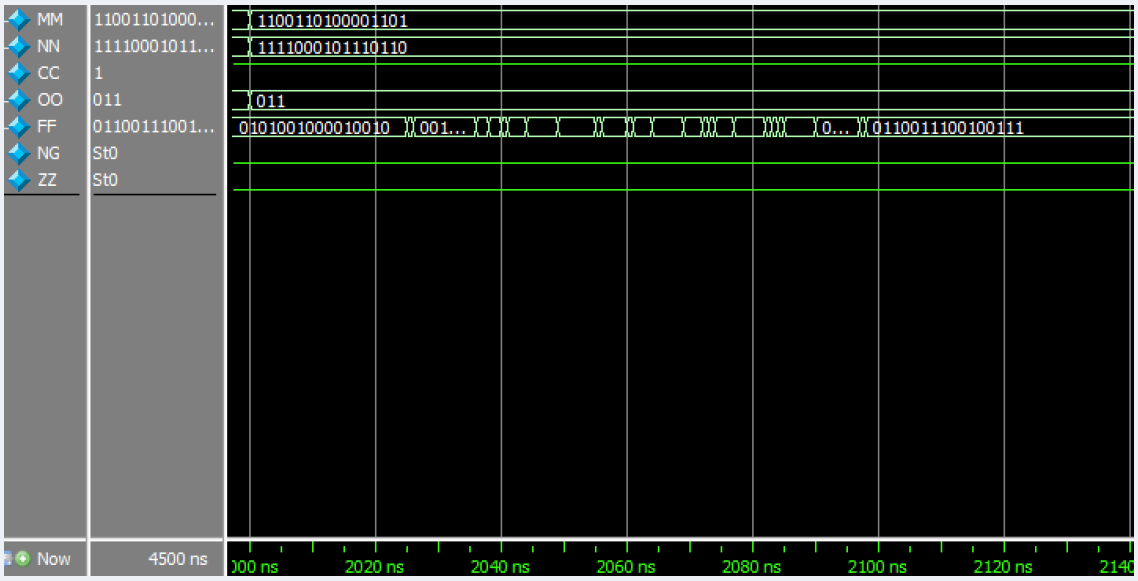

Figure 2.c.5

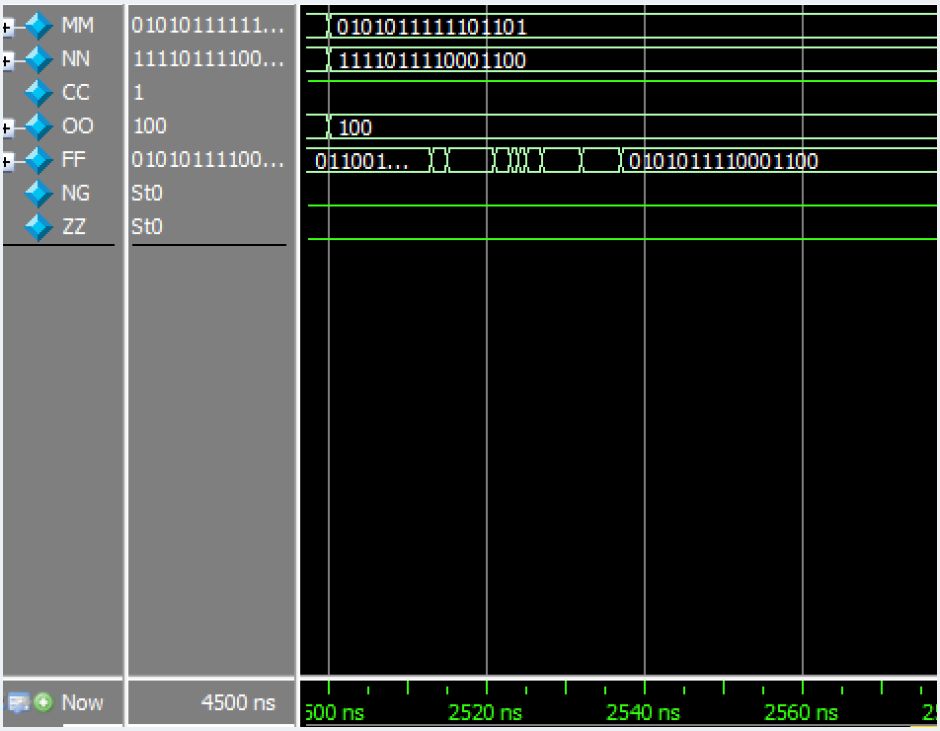

Figure 2.c.6

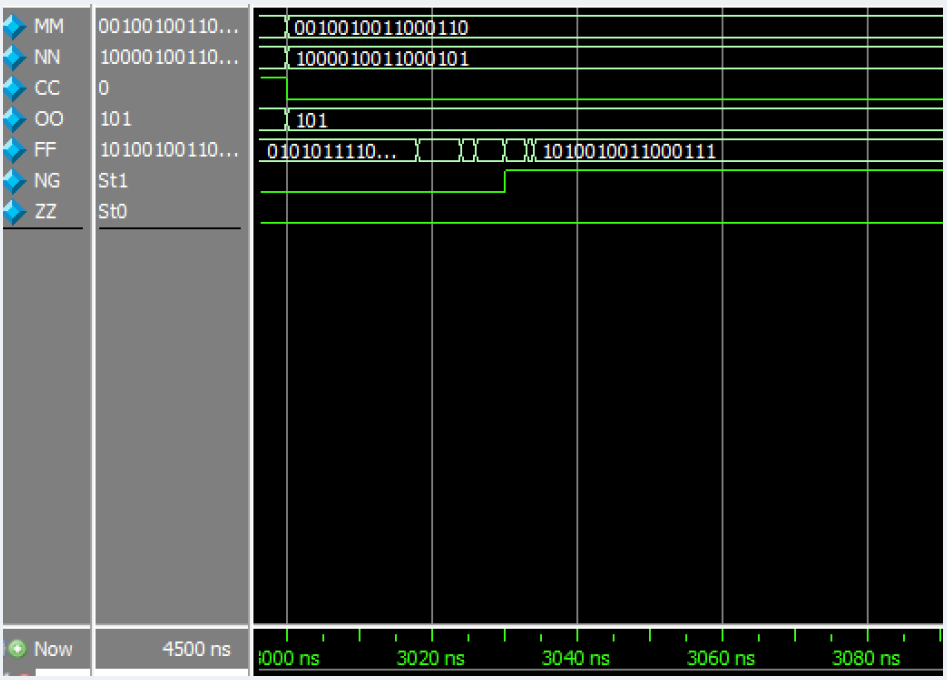

Figure 2.c.7

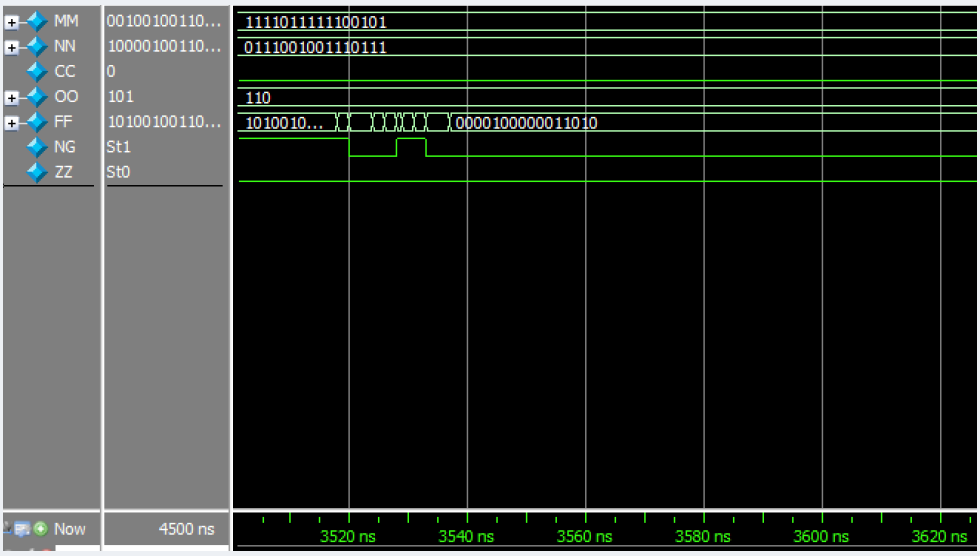

Figure 2.c.8

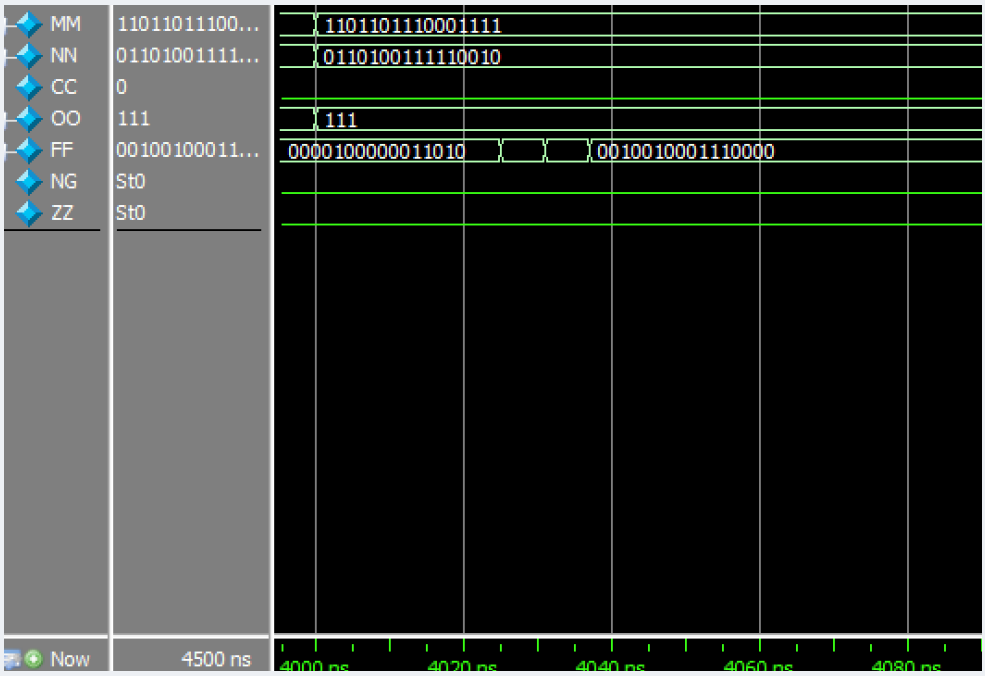
MM
NN
CC
OO
FF
NG
ZZ
11011011100...
01101001111...
0
111
00100100011...
St0
St0
1101101110001111
0110100111110010
111
0000100000011010
0010010001110000
Now
4500 ns
4000 ns
4020 ns
4040 ns
4060 ns
4080 ns

Figure 2.c.9

d.

The simulation time for part a (200000 reptations):

```
VSIM 273> simstats
# Memory Statistics
#     mem: size after elab (VSZ)                141.88 Mb
#     mem: size during sim (VSZ)                173.54 Mb
# Elaboration Time
#    elab: wall time                              0.18 s
#    elab: cpu time                               0.06 s
# Simulation Time
#     sim: wall time                              2.04 s
#     sim: cpu time                               0.61 s
# Tcl Command Time
#     cmd: wall time                             21.99 s
#     cmd: cpu time                               0.08 s
# Total Time
#   total: wall time                             24.21 s
#   total: cpu time                              0.75 s
#
```

Figure 2.d.1

The simulation time for part c (200000 reptations):

```
VSIM 284> simstats
# Memory Statistics
#     mem: size after elab (VSZ)                141.91 Mb
#     mem: size during sim (VSZ)                173.55 Mb
# Elaboration Time
#    elab: wall time                              0.25 s
#    elab: cpu time                               0.05 s
# Simulation Time
#     sim: wall time                             22.30 s
#     sim: cpu time                               8.11 s
# Tcl Command Time
#     cmd: wall time                             24.28 s
#     cmd: cpu time                               0.16 s
# Total Time
#   total: wall time                             46.83 s
#   total: cpu time                              8.31 s
#
```

Figure 2.d.2

Problem 3

As described in past two questions, synthesizing the behavioral structure, contains more gates then the other. In other words, Yosys is just a synthesizing tool and we can use our intelligence in order to redesign the structure so that the result becomes more accurate. Designs, number of gates and simulation timings and result described in previous pages!