Homework 3

COMPUTER ORGANIZATION – VERILOG FPGA

171044095 – Djuro Radusinovic

December 2021

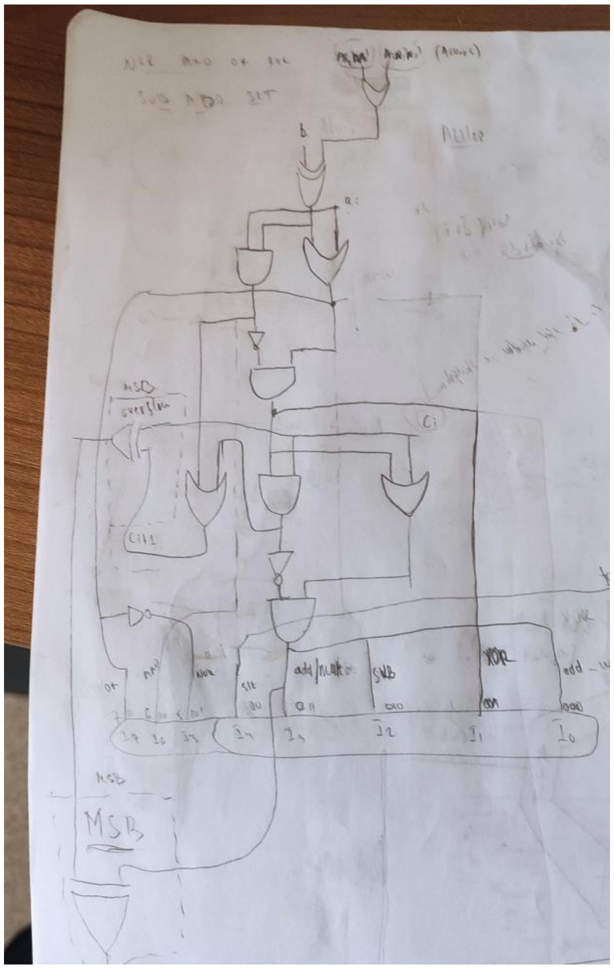
In this homework we were supposed to make an ALU in verilog. In order to achieve that I used an extended 1-bit ALU from the slides. This way I covered most of the funcitonality needed.

Also I used a 8x1 multiplexer in order to choose the opreation that will be performed.

In order to perform SLT for the MSB alu I XOR'ed MSB of the substitution with the overflow (xor'ing previous two carry outs).

After this I had all the operations I needed. After the operation was done I just used a multiplexer in order to choose between ALU's result and SLT's extended result (because SLT would use ALU for substraction and that wouldn't be result of slt).

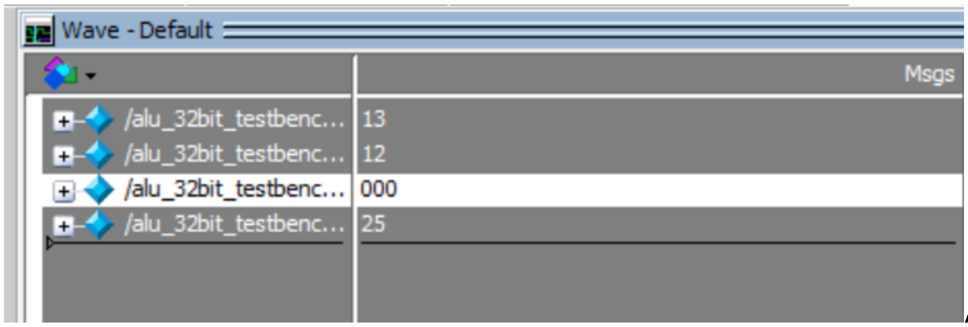
Layout of my ALU:



This is for the most significat bit. The normal 1-bit ALU is the same just without overflow and slt detection, two xors we can see here.

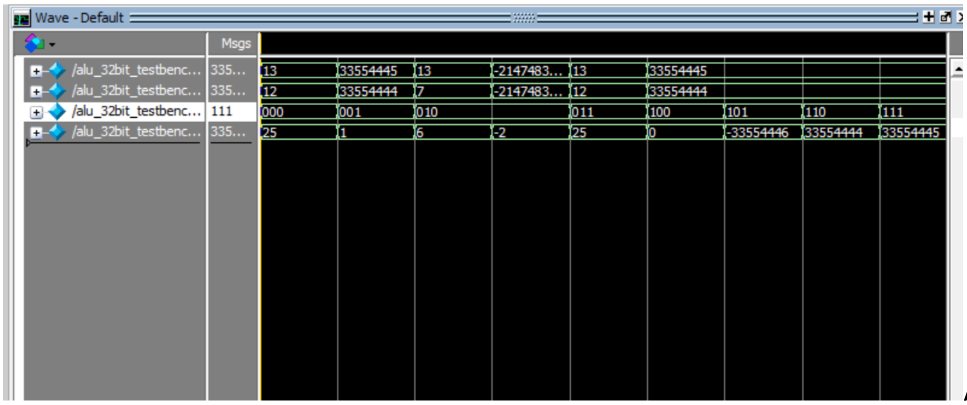
Also in order to figure out if b will be complemented I used: A2’A1A0’ + A2A1’A0’ which is for slt and sub. This was also used for first carry in of the ALU I built.

Here are the tests:



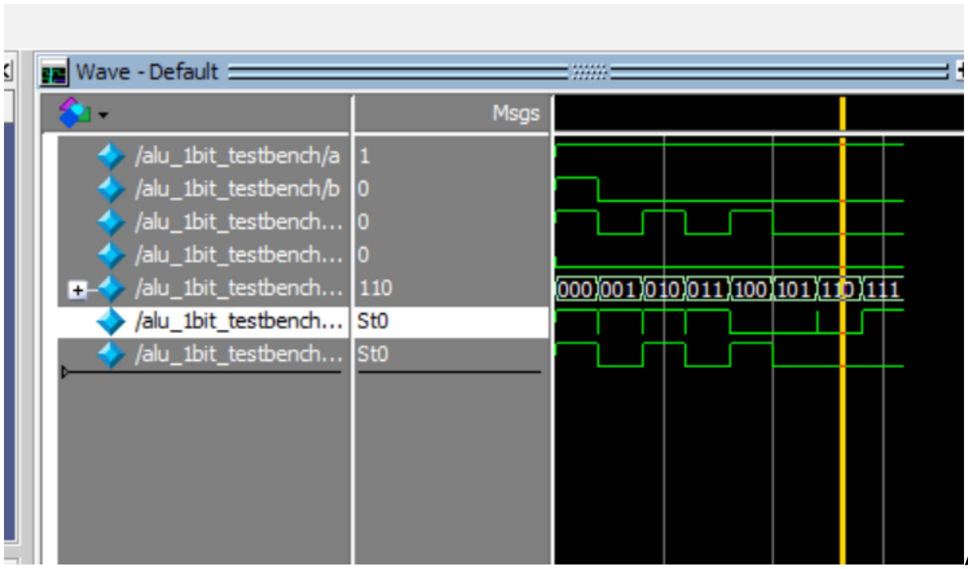
```
Wave - Default
                              Msgs
/alu_32bit_testbenc...   13
/alu_32bit_testbenc...   12
/alu_32bit_testbenc...   000
/alu_32bit_testbenc...   25
```

Addition



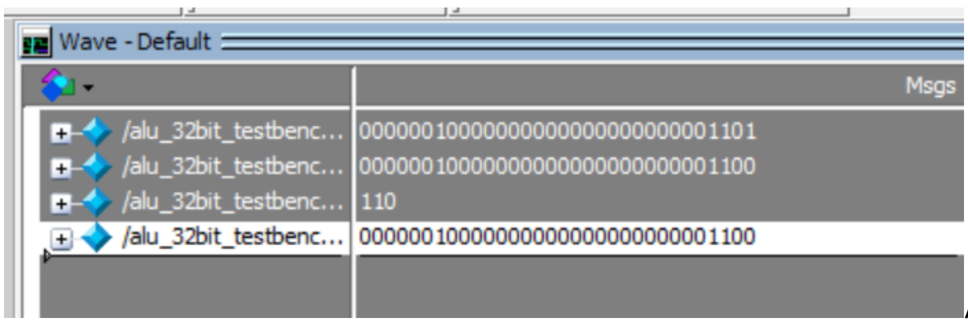
```
Wave - Default
                            Msgs
/alu_32bit_testbenc...  335...  13   33554445  13   -2147483...  13   33554445
/alu_32bit_testbenc...  335...  12   33554444  7    -2147483...  12   33554444
/alu_32bit_testbenc...  111     000  001       010               011  100       101        110       111
/alu_32bit_testbenc...  335...  25   1         6    -2           25   0         -33554446  33554444  33554445
```

ALU's overall results in decimal



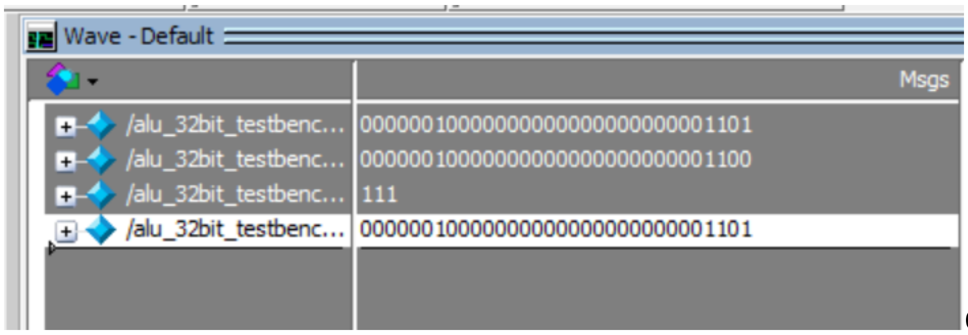
```
Wave - Default
                              Msgs
/alu_1bit_testbench/a      1
/alu_1bit_testbench/b      0
/alu_1bit_testbench...     0
/alu_1bit_testbench...     0
/alu_1bit_testbench...     110     000 001 010 011 100 101 110 111
/alu_1bit_testbench...     St0
/alu_1bit_testbench...     St0
```

ALU 1-bit test


Wave - Default

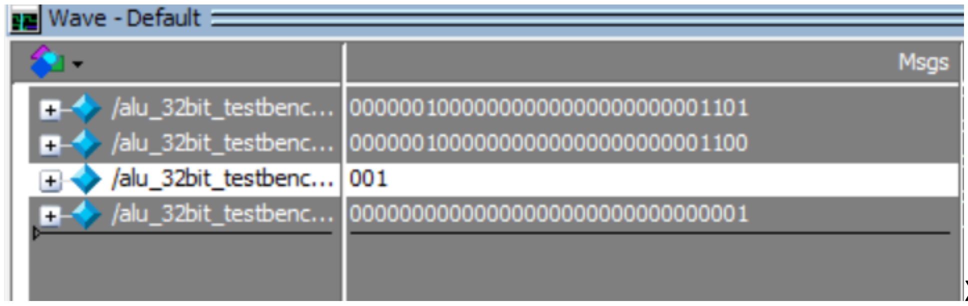
```
/alu_32bit_testbenc...   00000010000000000000000000001101
/alu_32bit_testbenc...   00000010000000000000000000001100
/alu_32bit_testbenc...   110
/alu_32bit_testbenc...   00000010000000000000000000001100
```

AND result


Wave - Default

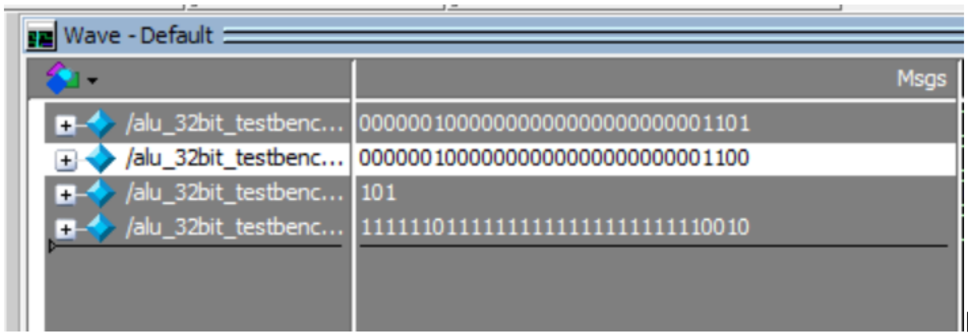
```
/alu_32bit_testbenc...   00000010000000000000000000001101
/alu_32bit_testbenc...   00000010000000000000000000001100
/alu_32bit_testbenc...   111
/alu_32bit_testbenc...   00000010000000000000000000001101
```

OR result


Wave - Default

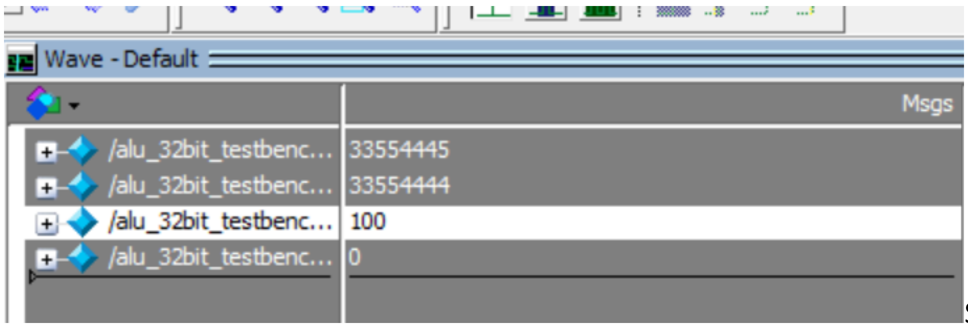
```
/alu_32bit_testbenc...   00000010000000000000000000001101
/alu_32bit_testbenc...   00000010000000000000000000001100
/alu_32bit_testbenc...   001
/alu_32bit_testbenc...   00000000000000000000000000000001
```

XOR result


Wave - Default

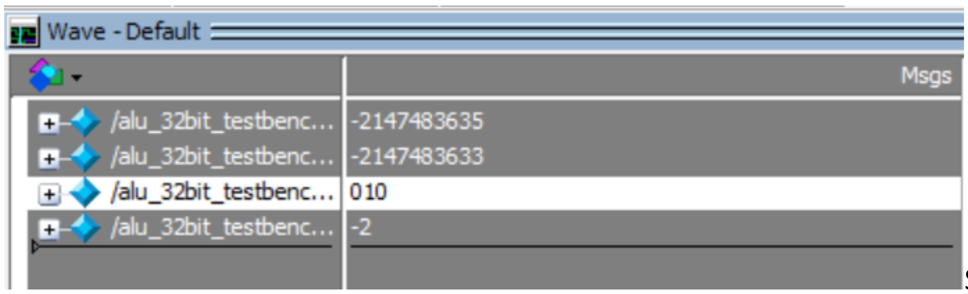
```
/alu_32bit_testbenc...   00000010000000000000000000001101
/alu_32bit_testbenc...   00000010000000000000000000001100
/alu_32bit_testbenc...   101
/alu_32bit_testbenc...   11111101111111111111111111110010
```

NOR result


Wave - Default

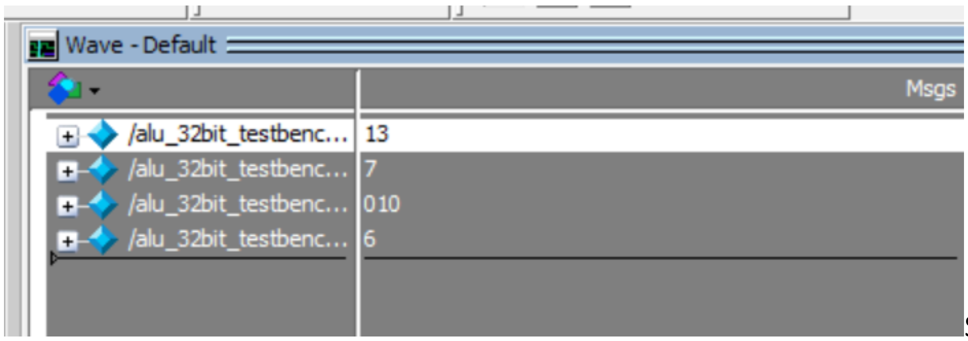
```
/alu_32bit_testbenc...   33554445
/alu_32bit_testbenc...   33554444
/alu_32bit_testbenc...   100
/alu_32bit_testbenc...   0
```

SLT result



```
Wave - Default
                                  Msgs
/alu_32bit_testbenc...   -2147483635
/alu_32bit_testbenc...   -2147483633
/alu_32bit_testbenc...   010
/alu_32bit_testbenc...   -2
```

SUB result

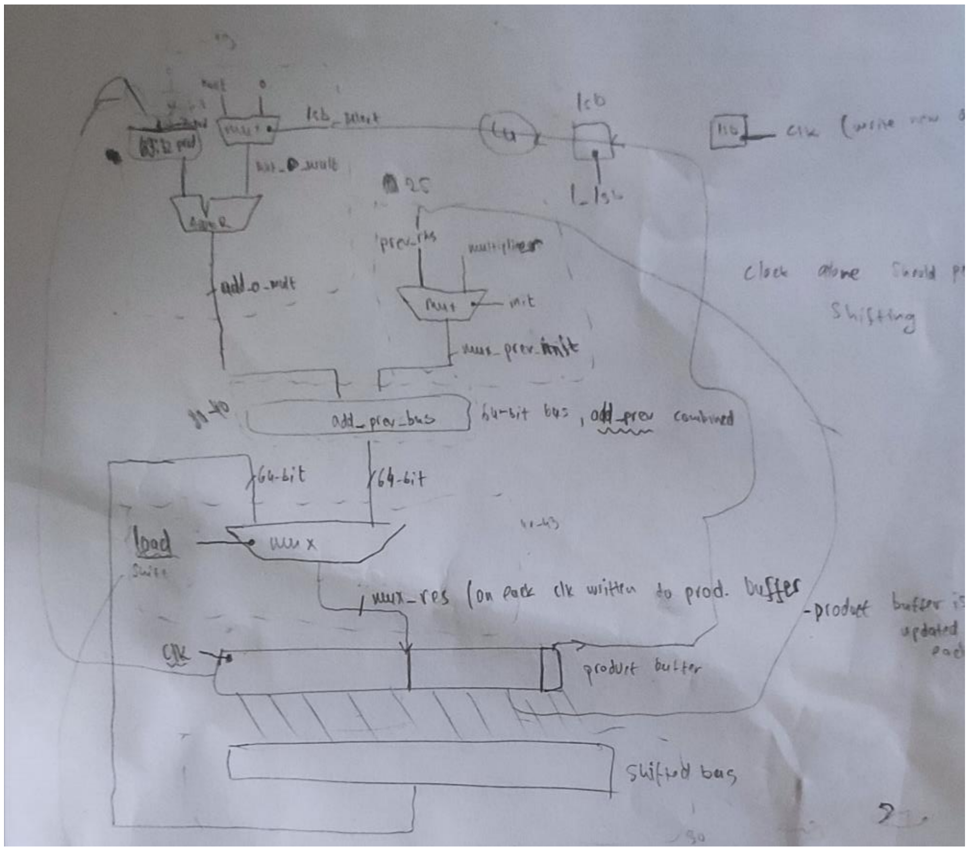


```
Wave - Default
                                  Msgs
/alu_32bit_testbenc...   13
/alu_32bit_testbenc...   7
/alu_32bit_testbenc...   010
/alu_32bit_testbenc...   6
```

SUB result 2

PART 2 – MULTIPLIER

For this I used the following datapath:

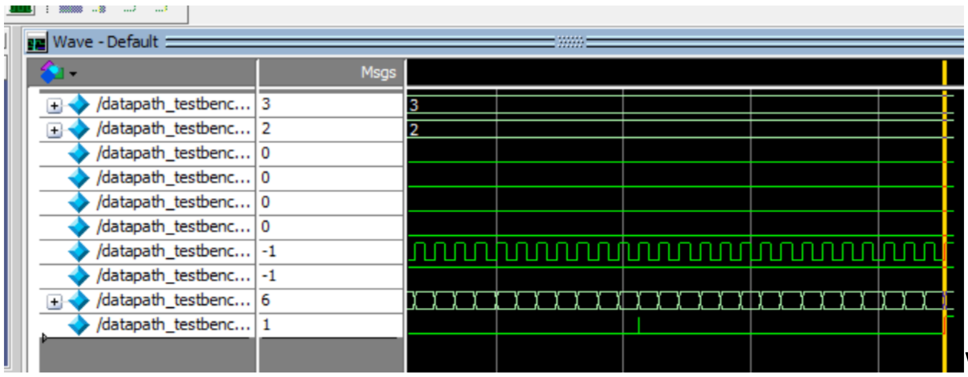


Here, there are many signals fed by Control unit. Signals coming from the control unit are:

- Shift_load (64bit product buffer is always being fed and on each clock is being changed by either its shifted value, if shift_load is 0 the product buffer will be shifted on positive clock edge, otherwise it will take the addition value.
- Lsb_select (this signal is used to select least signifiact bit of the product buffer before it was shifted, it is stored in LSB register and its value is taken in the third state. This Lsb_select has an AND gate formed with value of lsb's register). This way value is added only in a stage when lsb can selected which we will se in the finite state machine I designed. Otherwise it will perform addition with 0 which doesn't change the value of our product buffer which is desired behaviour.
- L_lsb (this signal is used to tell the ALU when to load the least significant bit into LSB register. This will always be performed one clock before the product buffer is updated/shifted.)
- Init (This signal is used for reseting the circuit. Even though I didn't draw it here there is an I_count register which stores number of iterations which are 32 to be exact here. When this counter reaches 0 its signal is sent to the FSM and FSM than sets init to 1 in the next clock's positive edge. This way reset can be performed.
- Finished_cycle – this signal is not shown here but this signal is used in order to decrease our I_count register/counter. One cycle doesn't take one clock of course. In order to work, counter is updated each time before shift is performed. This way we can keep

track of the number of shifts/additions that have been performed and reset them in appropriate time.

Datapath’s test:

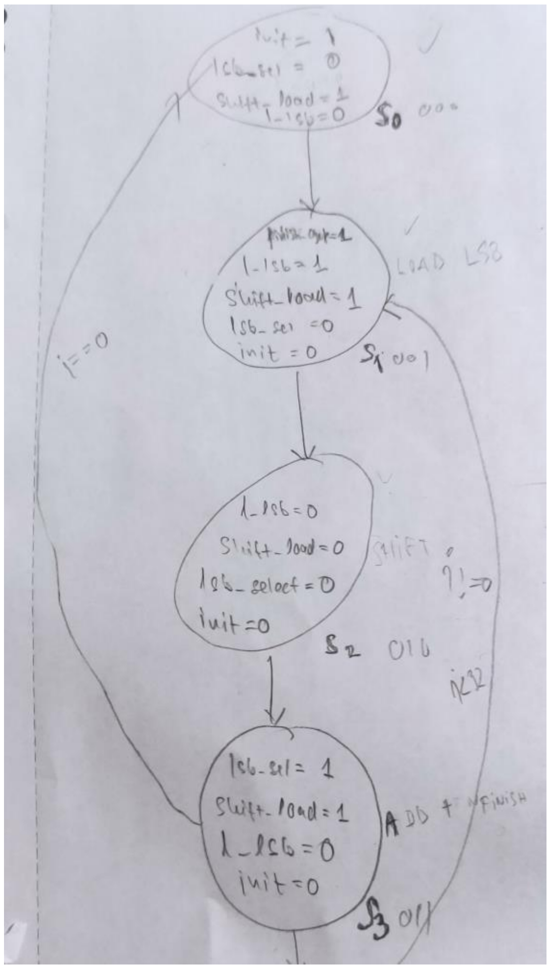

Wave - Default

	Msgs
/datapath_testbenc…	3
/datapath_testbenc…	2
/datapath_testbenc…	0
/datapath_testbenc…	0
/datapath_testbenc…	0
/datapath_testbenc…	0
/datapath_testbenc…	-1
/datapath_testbenc…	-1
/datapath_testbenc…	6
/datapath_testbenc…	1

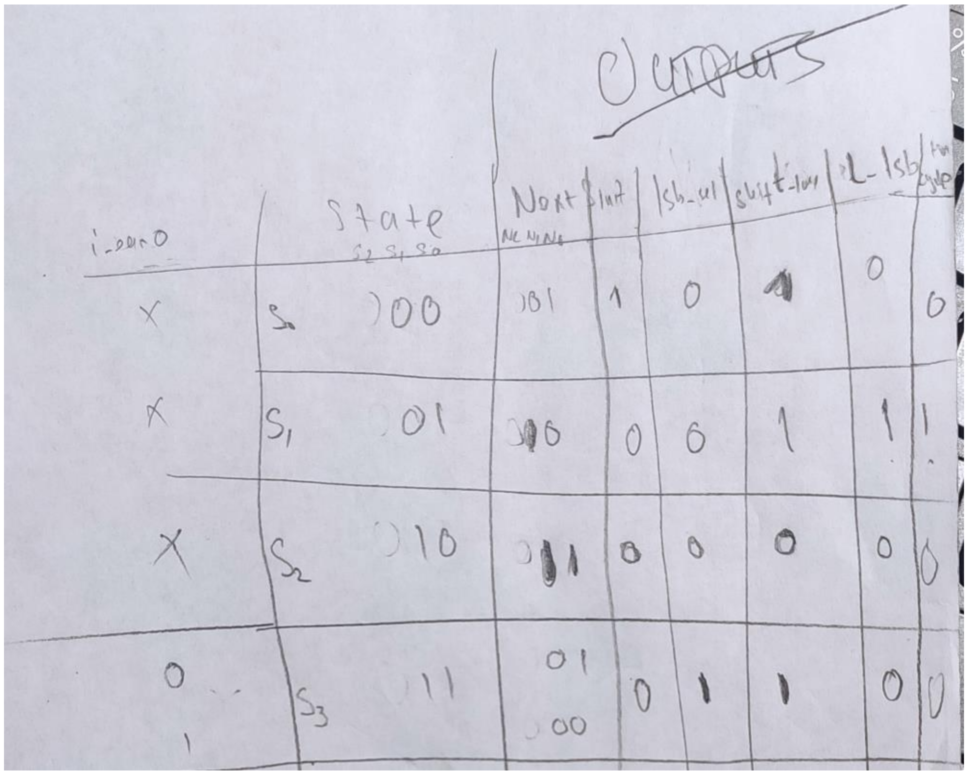
We can see here that for inputs 3 and 2, 6 is our result. Later on when I assembled everything I used larger numbers for testing.

For this datapath I implemented the following FSM.

Here is my FSM diagram:



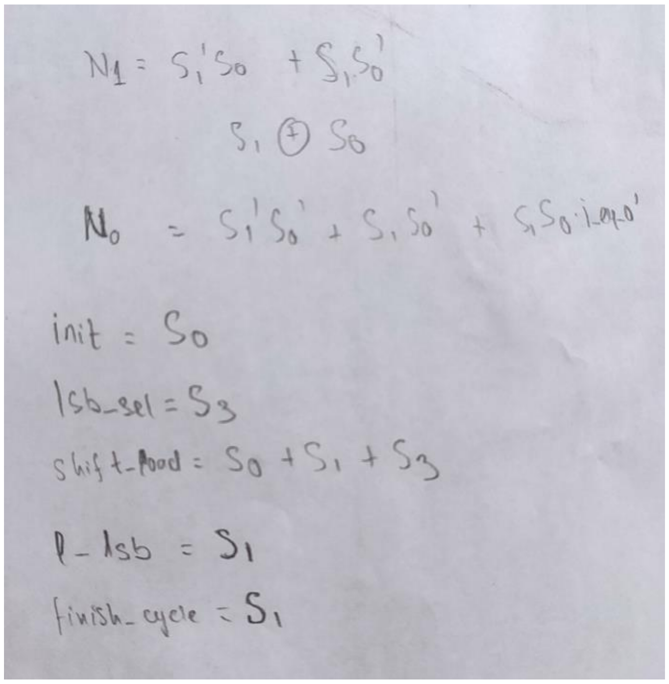
Here is my FSM table:



~~Outputs~~

i_zero	State	$S_2 S_1 S_0$	Next state $N_2 N_1 N_0$	init	lsb_sel	shift_load	L-lsb	fin_cycle
X	S_0	00	001	1	0	1	0	0
X	S_1	01	010	0	0	1	1	1
X	S_2	10	011	0	0	0	0	0
0 / 1	S_3	11	01 / 00	0	1	1	0	0

Here are the boolean equations I extracted from the table:



$N_1 = S_1'S_0 + S_1S_0'$

$S_1 \oplus S_0$

$N_0 = S_1'S_0' + S_1S_0' + S_1S_0 \cdot i\_zero'$

init $= S_0$

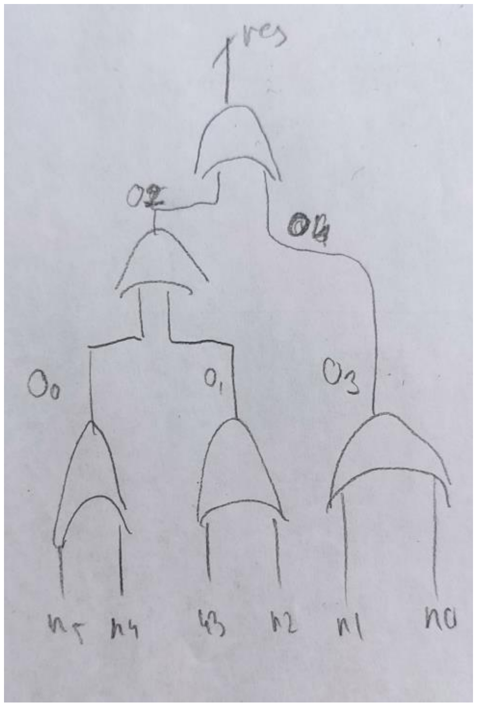
lsb_sel $= S_3$

shift_load $= S_0 + S_1 + S_3$

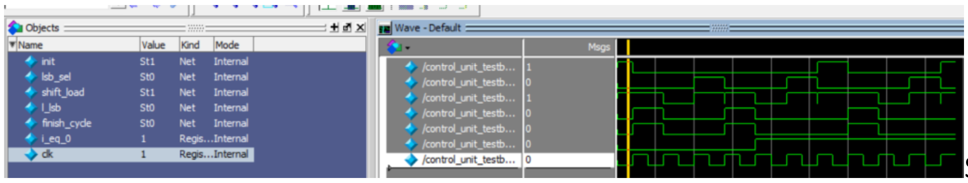
L-lsb $= S_1$

finish_cycle $= S_1$

Also in order to keep track of our counter I implemented a simple circuit that checks if its value is zero. Since 32 needs 6bits I used only 6 bits in order to implement it. It is just a bunch of OR gates and I use an INVERTER to get the result after that which will tell if the I_count is equal to zero or not. This is an output of datapath used in FSM.



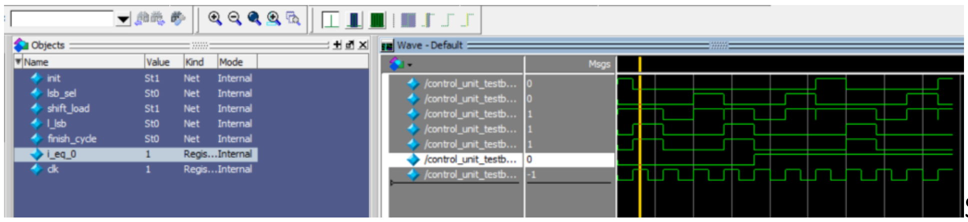
Here are the results of the Control unit that I wrote:



Name	Value	Kind	Mode
init	St1	Net	Internal
lsb_sel	St0	Net	Internal
shift_load	St1	Net	Internal
l_lsb	St0	Net	Internal
finish_cycle	St0	Net	Internal
i_eq_0	1	Regis...	Internal
clk	1	Regis...	Internal

```
/control_unit_testb...  1
/control_unit_testb...  0
/control_unit_testb...  1
/control_unit_testb...  0
/control_unit_testb...  0
/control_unit_testb...  0
/control_unit_testb...  0
```

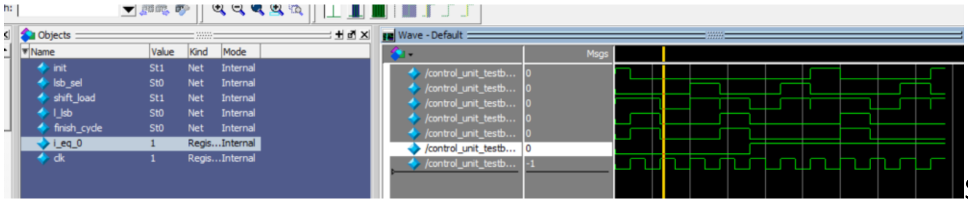
State 0



Name	Value	Kind	Mode
init	St1	Net	Internal
lsb_sel	St0	Net	Internal
shift_load	St1	Net	Internal
l_lsb	St0	Net	Internal
finish_cycle	St0	Net	Internal
i_eq_0	1	Regis...	Internal
clk	1	Regis...	Internal

```
/control_unit_testb...  0
/control_unit_testb...  0
/control_unit_testb...  1
/control_unit_testb...  1
/control_unit_testb...  1
/control_unit_testb...  0
/control_unit_testb...  -1
```

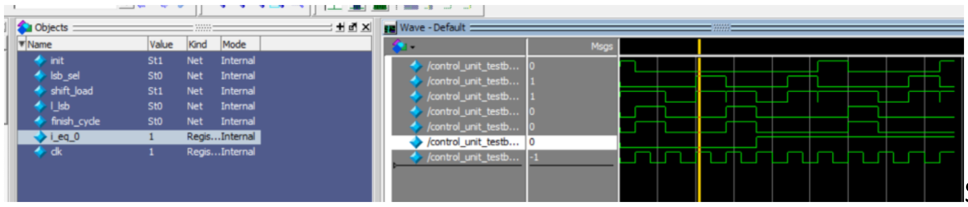
State 1



Name	Value	Kind	Mode
init	St1	Net	Internal
lsb_sel	St0	Net	Internal
shift_load	St1	Net	Internal
l_lsb	St0	Net	Internal
finish_cycle	St0	Net	Internal
i_eq_0	1	Regis...	Internal
clk	1	Regis...	Internal

```
/control_unit_testb...  0
/control_unit_testb...  0
/control_unit_testb...  0
/control_unit_testb...  0
/control_unit_testb...  0
/control_unit_testb...  0
/control_unit_testb...  -1
```

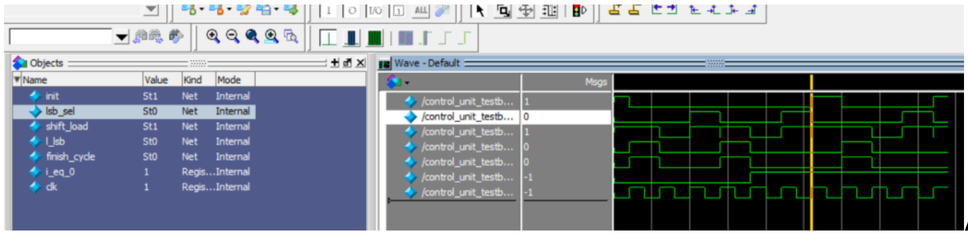
State 2



Name	Value	Kind	Mode
init	St1	Net	Internal
lsb_sel	St0	Net	Internal
shift_load	St1	Net	Internal
l_lsb	St0	Net	Internal
finish_cycle	St0	Net	Internal
i_eq_0	1	Regis...	Internal
clk	1	Regis...	Internal

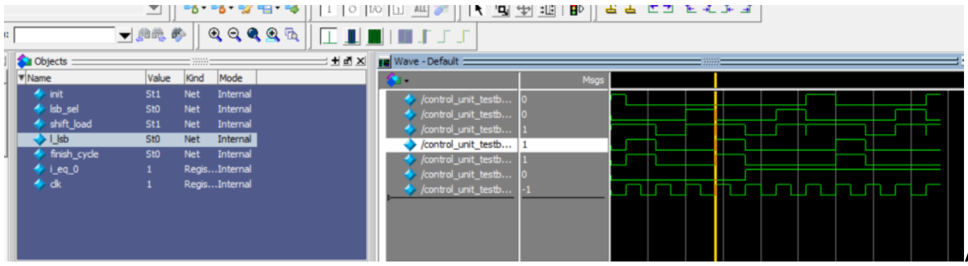
```
/control_unit_testb...  0
/control_unit_testb...  1
/control_unit_testb...  1
/control_unit_testb...  0
/control_unit_testb...  0
/control_unit_testb...  0
/control_unit_testb...  -1
```

State 3



```
Objects
Name          Value  Kind      Mode
init          St1    Net       Internal
lsb_sel       St0    Net       Internal
shift_load    St1    Net       Internal
l_lsb         St0    Net       Internal
finish_cycle  St0    Net       Internal
i_eq_0        1      Regis...  Internal
clk           1      Regis...  Internal

Wave - Default                Msgs
/control_unit_testb...        1
/control_unit_testb...        0
/control_unit_testb...        1
/control_unit_testb...        0
/control_unit_testb...        0
/control_unit_testb...        -1
/control_unit_testb...        -1
```

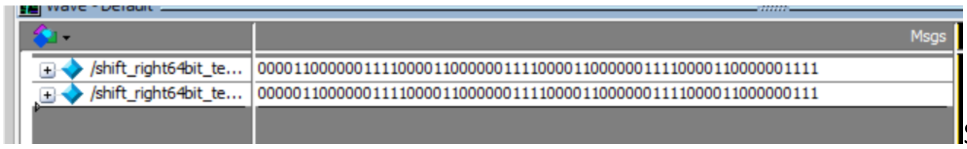
After state 3 when i_eq_0 = 1 {should reset after this}



```
Objects
Name          Value  Kind      Mode
init          St1    Net       Internal
lsb_sel       St0    Net       Internal
shift_load    St1    Net       Internal
l_lsb         St0    Net       Internal
finish_cycle  St0    Net       Internal
i_eq_0        1      Regis...  Internal
clk           1      Regis...  Internal

Wave - Default                Msgs
/control_unit_testb...        0
/control_unit_testb...        0
/control_unit_testb...        1
/control_unit_testb...        1
/control_unit_testb...        1
/control_unit_testb...        0
/control_unit_testb...        -1
```

After state 3 when i_eq_0 = 0 {continues with updating product buffer}

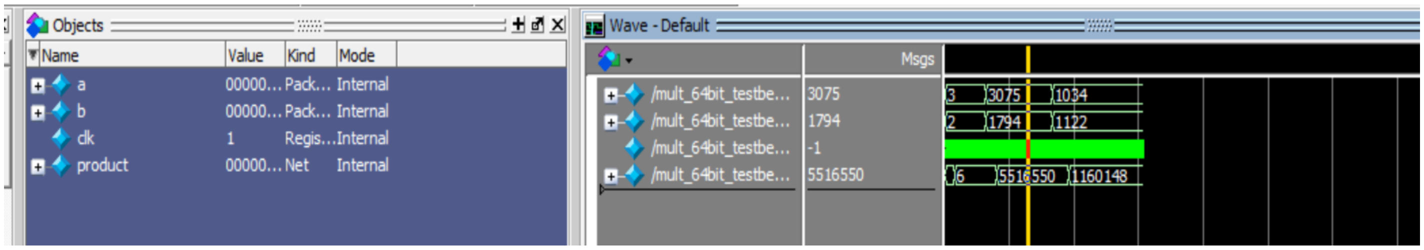
I built a shifter in order for this to work and connected it to the product buffer which is constantly being updated. This Product buffer is in a for a SHIFT REGISTER being able to either store its previous value or shift itself to the right.



```
Wave - Default                                                                         Msgs
/shift_right64bit_te...  0000110000000111100001100000001111000011000000011110000110000001111
/shift_right64bit_te...  0000011000000011110000110000000111100001100000001111000011000000111
```

Shifter result

Now here are the results of the MULTIPLIER circuit:



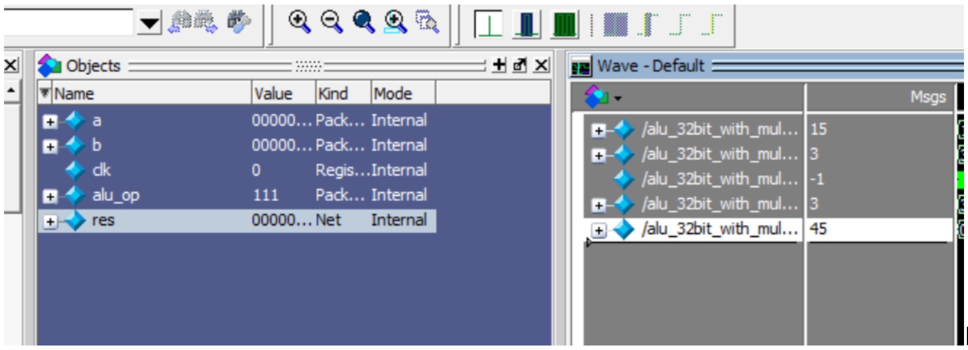
```
Objects
Name       Value    Kind      Mode
a          00000... Pack...   Internal
b          00000... Pack...   Internal
clk        1        Regis...  Internal
product    00000... Net       Internal

Wave - Default           Msgs
/mult_64bit_testbe...    3075      3  3075  1034
/mult_64bit_testbe...    1794      2  1794  1122
/mult_64bit_testbe...    -1
/mult_64bit_testbe...    5516550   6  5516550  1160148
```

MULTIPLIER64 results

After finishing my multiplier I made another ALU. This ALU combines previously mentioned ALU with the newly built multiplier. In order to achieve that functionality I used a 32bit multiplexer.

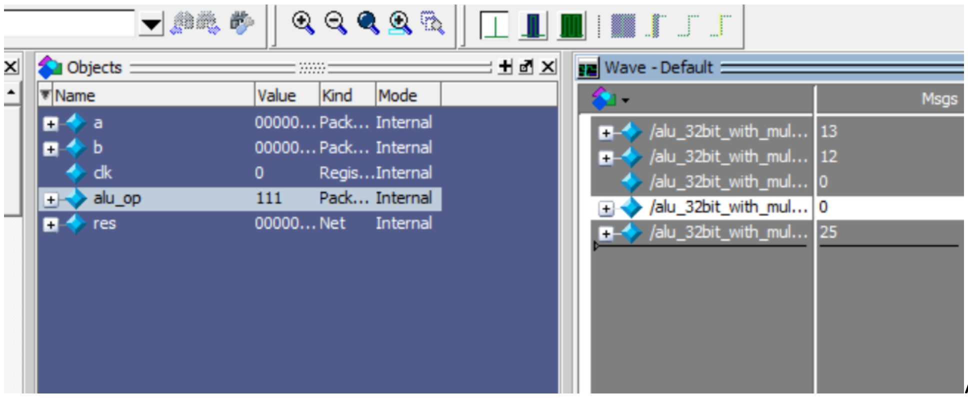
Select of this multiplexer is dependent on ALU_OP. When code corresponds to multiplication operation -> A2'.A1.A0 {select of mux} than multiplication result is chosen and otherwise other operation is chosen as a result. Note that since MULT64 is sequential the result will be seen at the end with a small delay. In the beginning the value will be 0 until calculation is finished after which value is shown and updated.

Here are the tests of ALU with 32bit Multiplier:



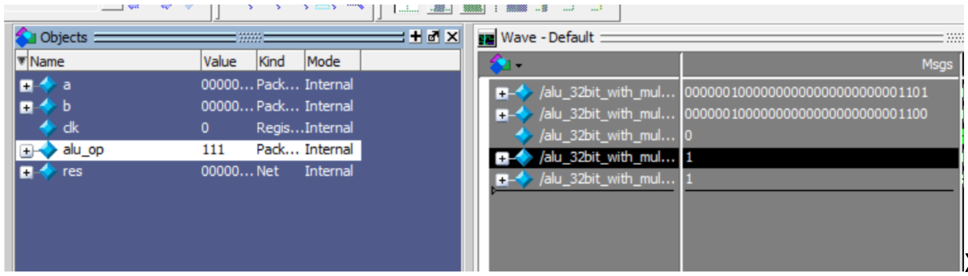
```
Objects
Name     Value    Kind     Mode
a        00000... Pack...  Internal
b        00000... Pack...  Internal
clk      0        Regis... Internal
alu_op   111      Pack...  Internal
res      00000... Net      Internal

Wave - Default
                              Msgs
/alu_32bit_with_mul...   15
/alu_32bit_with_mul...   3
/alu_32bit_with_mul...   -1
/alu_32bit_with_mul...   3
/alu_32bit_with_mul...   45
```

Multiplication



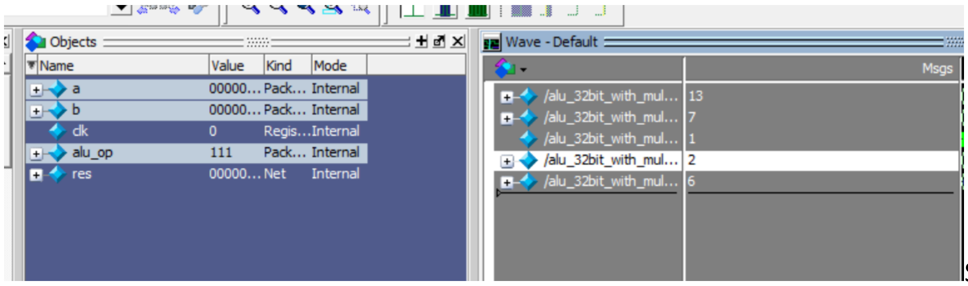
```
Objects
Name     Value    Kind     Mode
a        00000... Pack...  Internal
b        00000... Pack...  Internal
clk      0        Regis... Internal
alu_op   111      Pack...  Internal
res      00000... Net      Internal

Wave - Default
                              Msgs
/alu_32bit_with_mul...   13
/alu_32bit_with_mul...   12
/alu_32bit_with_mul...   0
/alu_32bit_with_mul...   0
/alu_32bit_with_mul...   25
```

Addition



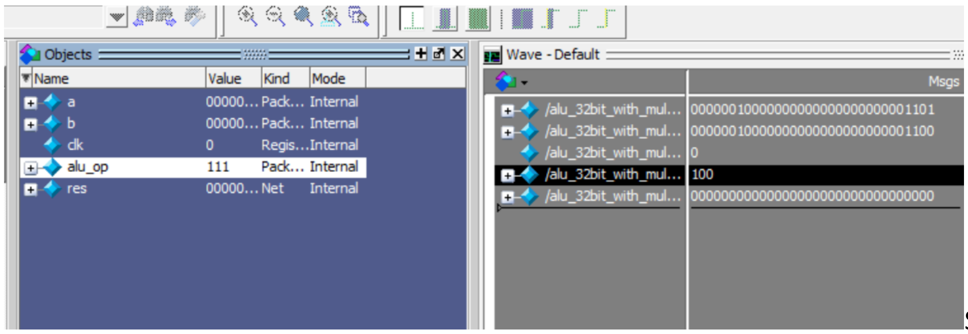
```
Objects
Name     Value    Kind     Mode
a        00000... Pack...  Internal
b        00000... Pack...  Internal
clk      0        Regis... Internal
alu_op   111      Pack...  Internal
res      00000... Net      Internal

Wave - Default
                              Msgs
/alu_32bit_with_mul...   00000010000000000000000000001101
/alu_32bit_with_mul...   00000010000000000000000000001100
/alu_32bit_with_mul...   0
/alu_32bit_with_mul...   1
/alu_32bit_with_mul...   1
```

Xor



```
Objects
Name     Value    Kind     Mode
a        00000... Pack...  Internal
b        00000... Pack...  Internal
clk      0        Regis... Internal
alu_op   111      Pack...  Internal
res      00000... Net      Internal

Wave - Default
                              Msgs
/alu_32bit_with_mul...   13
/alu_32bit_with_mul...   7
/alu_32bit_with_mul...   1
/alu_32bit_with_mul...   2
/alu_32bit_with_mul...   6
```

Substraction



Name	Value	Kind	Mode
a	00000...	Pack...	Internal
b	00000...	Pack...	Internal
clk	0	Regis...	Internal
alu_op	111	Pack...	Internal
res	00000...	Net	Internal

```
Wave - Default                     Msgs
/alu_32bit_with_mul...  00000010000000000000000000001101
/alu_32bit_with_mul...  00000010000000000000000000001100
/alu_32bit_with_mul...  0
/alu_32bit_with_mul...  100
/alu_32bit_with_mul...  00000000000000000000000000000000
```

SLT

NOTE:

I wasn't able to access transcript and see Monitor's values though I checked everything from the wave form in order to make sure everything works properly and didn't find any need to redo it using Transcript which for some reason doesn't appear on my laptop.

There are some additional circuits as part of the project even though I didn't use them. Example is: or_32bit, and_32bit. In the beginning I though it was better to do it like that but later on changed my mind and decided to make an ALU design since it uses less gates and has more functionality in itself alone. Also all the tests are performed for the additional circuits I used and of course you can check them using testbench.v documents I uploaded with this assignment.