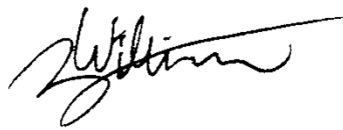


Constructing an Arithmetic Logic Unit based on Verilog HDL

VLSI 4054

Zakary Williams

1106374

A handwritten signature in black ink, appearing to read 'Zakary Williams', with a stylized, cursive script.

Zakary Williams

12/07/2020

Abstract:

This project required the creation of a 4-function arithmetic logic unit (ALU), with two 3-bit data inputs and a 2-bit control, which would then output to a dual 7-segment display module. This was achieved using Verilog HDL which is a hardware description language that allows for a program such as Quartus prime, which synthesizes the processes required to create the instructions needed for an FPGA (Field programmable gate array) to create its CMOS structure. A series of modules were written which completed the XNOR (Exclusive nor), shift register left, add, and multiply functions as well as a module which displays the outputs of those functions on six 7-segment displays. A top level was also constructed to create the wiring paths and registers required to accommodate each module in a single structure. A series of testbenches were also required to test each of the modules individually.

An ALU was successfully realized and the final result was tested and proved able to output the desired values on the six 7-segments. Each module functioned as intended individually and the multiplication function was constructed using the addition and shift left method. Many obstacles needed to be overcome for this project to function. An issue that was encountered was that the top-level wiring could not come from multiple outputs. This meant that each module must use a unique signifier. The display module proved difficult at first but was implemented successfully using case statements and lookup tables. This also proved to be the fastest method. Finally, the shift register function required a reset register to be created to allow the reset to resample the input after the reset is lifted.

Contents

Design specifications:.....	5
Theory:	5
Operation theory,	5
Design:	7
Verification:.....	10
Summary:	13
Citations:	14
Appendices:.....	15
XNOR code and testbench	15
Shift code and testbench	17
Add code and testbench	19
Multiplier code and testbench.....	21
Display code and testbench	23
Top level code and testbench	30
Output waves:.....	34
<i>Blocks</i>	37

Figures and Tables

Figure 1: 7-Segment display illumination table [2]	7
Figure 2: ALU block [3]	7
Figure 3: Top level block diagram of ALU and display unit	8
Figure 4: ALU timing diagram.....	9
Figure 5: Output wave of the XNOR module	10
Figure 6: Output wave of the shift registers left module	10
Figure 7: Output wave of the addition module	11
Figure 8: Output wave of the multiplication module	11
Figure 9: Output wave of the display module	12
Figure 10: Output wave of the overall ALU unit showing the function of each module working	12
Table 1: ALU operations	5
Table 2: Xnor truth table	5

Design specifications:

This project required the design of an ALU (Arithmetic Logic Unit) with the following requirements:

Inputs:

- Two 3-bit register inputs
- 2-bit control signal input
- Enable signal
- Reset signal
- Clock signal

Outputs:

- Six 7-segment displays

Operations:

Operations	OP (control signal)	Operation performed
XNOR (Bitwise)	00	Doutxnor = (A Xnor B)
Shift left	01	Shift left [A,B] by 1-bit
Add	10	Doutadd = A + B
Multiply	11	Doutmult = A * B

Table 1: ALU operations

Theory:

Operation theory,

XNOR

The exclusive nor gate is useful as a comparator. Given two logic “1” signals, it outputs a logic “1”, likewise for two “0” inputs. The following 3-input Xnor truth table describes that. This function was implemented using the Verilog primitive (^~).

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

Table 2: Xnor truth table

Shift

The shift function implemented takes the two 3-bit inputs and concatenates them to form a 6-bit binary number. Then when the clock signal is applied it shifts each bit to the left and the vacant bit is replaced with a "0". When the reset signal is applied during operation the register is reset to all low and once the reset is lifted the inputs are polled and the register is again filled and shifted.

Add

The add function adds the two 3-bit inputs. This is implemented using the primitive (+) found in Verilog. The addition outputs to a 6-bit register and a pull off register is created for the carry bit. An example of the add function is shown in equation one.

$$\begin{array}{r} 101 \\ + 010 \\ \hline 0111 \end{array} \quad (eq, 1)$$

Multiply

The multiply function was implemented using the shift add approach. Where a case statement samples B input and shifts and adds A depending on that input. For example,

B = 010, A = 100

B is the multiplicand and A the multiplier in this case. The multiplier is shifted until it's last bit lines up with the first "1" of the multiplicand then add the multiplier in that position to the result. This is shown in equation two.

$$\begin{array}{r} 010 \\ * 100 \\ \hline 001000 \end{array} \quad (eq, 2)$$

Display

Multiple 7-segment displays needed to be used to display the output registers bits as either binary or decimal numbers. 7-segments display using LED (light emitting diodes) and we can display integers on them by selecting individual segments which are regarded as such.

Segments (✓ = ON)							Display	Segments (✓ = ON)							Display
a	b	c	d	e	f	g		a	b	c	d	e	f	g	
✓	✓	✓	✓	✓	✓		0	✓	✓	✓	✓	✓	✓	✓	8
	✓	✓					1	✓	✓	✓			✓	✓	9
✓	✓		✓	✓		✓	2	✓	✓	✓		✓	✓	✓	A
✓	✓	✓	✓			✓	3			✓	✓	✓	✓	✓	b
	✓	✓			✓	✓	4	✓			✓	✓	✓		c
✓		✓	✓		✓	✓	5		✓	✓	✓	✓		✓	d
✓		✓	✓	✓	✓	✓	6	✓			✓	✓	✓	✓	E
✓	✓	✓					7	✓				✓	✓	✓	F

Figure 1: 7-Segment display illumination table [2]

Design:

To implement this design multiple individual modules had to be constructed. Each module corresponds to a function that takes place within the ALU. If this project was in person, the functionality of this project would have been tested and implemented onto an FPGA (Field programmable gate array) board but due to circumstances this was not an option.

Each module shares an input clock, an enable signal, the operation selection signal, and two 3-bit inputs. Each module is synched to the clock by an “always” command which triggers the modules function on the positive edge of that clock. If the reset signal turns negative, as it is a normally closed signal, each module responds by clearing its registers and halting output. Once the reset is lifted the modules will continue regular operation. Finally, the enable signal surrounds the entire module and if that signal is not logic “1” the unit will not register the clock signal.

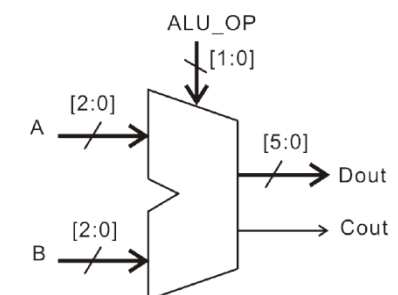


Figure 2: ALU block [3]

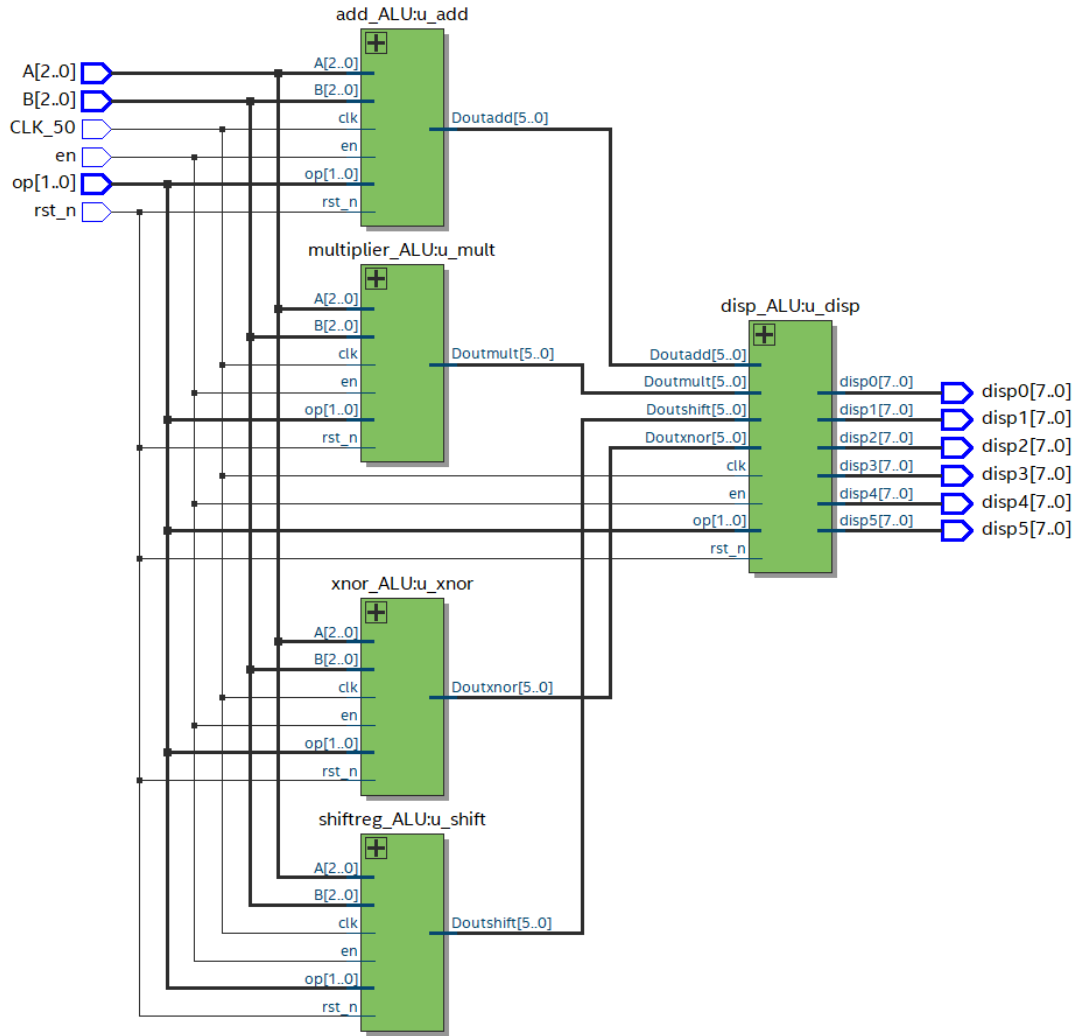


Figure 3: Top level block diagram of ALU and display unit

This block diagram is a representation of the communication paths used to allow for inter module communication. This image was rendered using Quartus prime lite edition. Each module has its own sub block diagram which shows the logic gates used to implement their functions. The module block diagram can be found in the appendix on page # of this document. Twelve Verilog files were created to achieve this. The “Top level” is the highest level of structural code, each other module is executed from within it. The code of each file can be found in the appendix on page # of this document.

Module	Module file	Test bench file
Xnor	ALU_Xnor.v	ALU_Xnor_TB.v
Shift left	ALU_Shiftreg.v	ALU_Shiftreg_TB.v
Add	ALU_Add.v	ALU_Add_TB.v
Mult	ALU_Mult.v	ALU_Mult_TB.v
Display	ALU_Dis.v	ALU_Dis_TB.v
Top Level	ALU_Toplevel.v	ALU_Toplevel_TB.v

Table 3: Project files

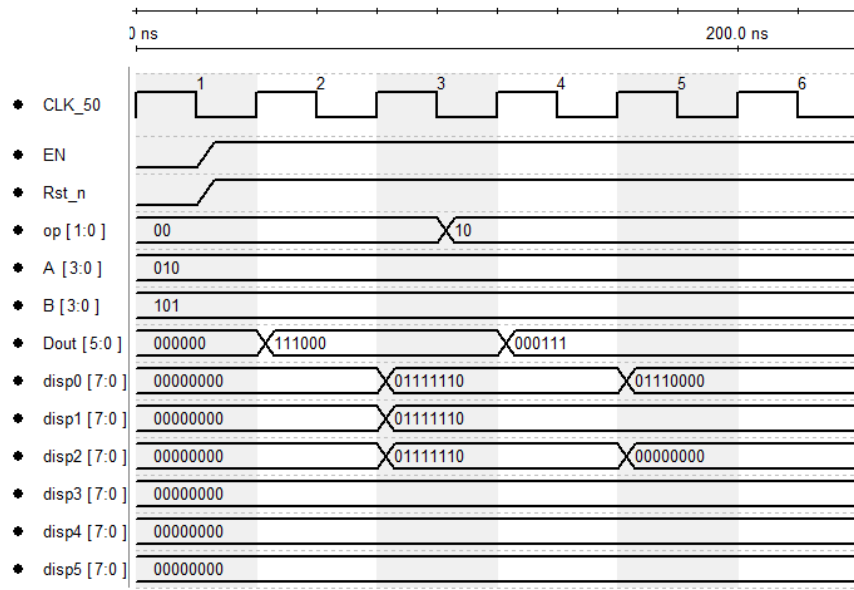


Figure 4: ALU timing diagram

This is the timing diagram of the overall ALU unit. The operations are triggered by the positive edge of the 50MHz clock signal regardless of input timing. The Dout register is adjusted to reflect the output of whichever module's function has been utilised at that clock signal. Then once that register has changed, at the next clock cycle the display module will update and display the values on the 7-segments.

Verification:

XNOR

This module compares the bits of each input register, A and B, to the other registers bit at the same location. If the two bits are the same, both “1” or both “0” then a logic “1” will be output in Y. The function of this module was confirmed with the following output wave.

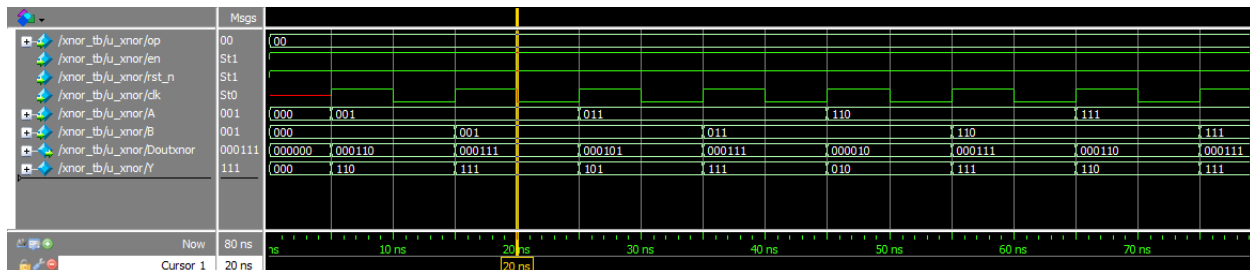


Figure 5: Output wave of the XNOR module

@ 10ns

A = 001

B = 000

Dout = 000110

^

Since the final bits of A and B do not match.

@ 20ns

A = 001

B = 001

Dout = 000111

^

Since the final bits of A and B match.

Shift

The shift module takes A and B inputs, concatenates them and then shifts them to the left by 1 bit each clock signal. We see from this output wave below that the inputs are successfully sampled and shifted with a functioning register reset.

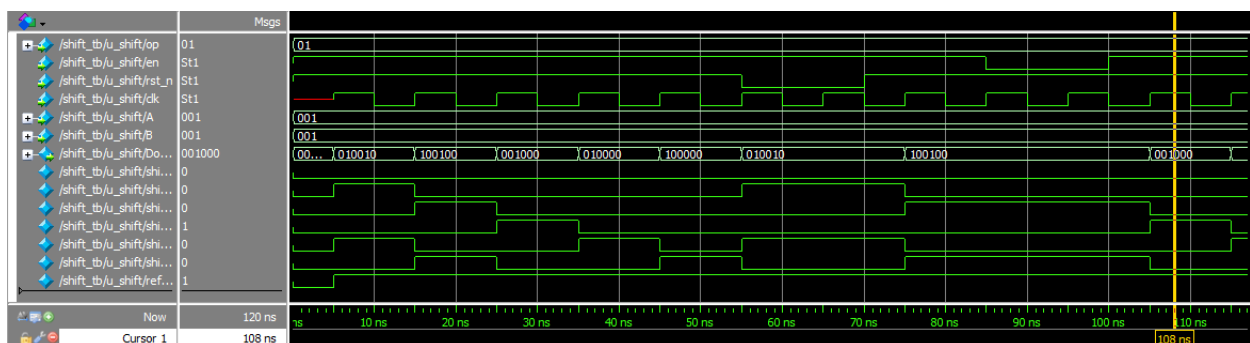


Figure 6: Output wave of the shift registers left module

Add

The add module simply adds the two input registers and outputs the result as an integer as well as outputting a carry bit. From the following output waveform we can see the functions successfully adds and creates the proper carry bit.

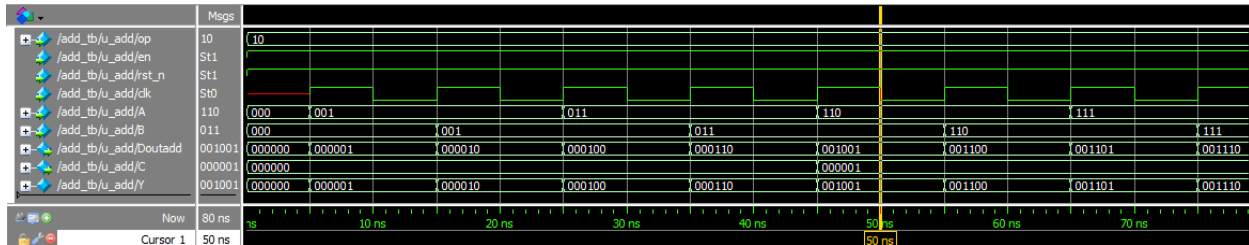


Figure 7: Output wave of the addition module

@ 50 ns (Using eq, 1)

$$\begin{array}{r} 110 \\ + 011 \\ \hline 1001 \end{array} \text{ equivalent to } 6 + 3 = 9, 9 \text{ in binary is } 1001, \text{ function is confirmed.}$$

Multiply

The multiply function as stated above uses the shift add method to accomplish binary multiplication. The following output waveform demonstrates the module functioning.

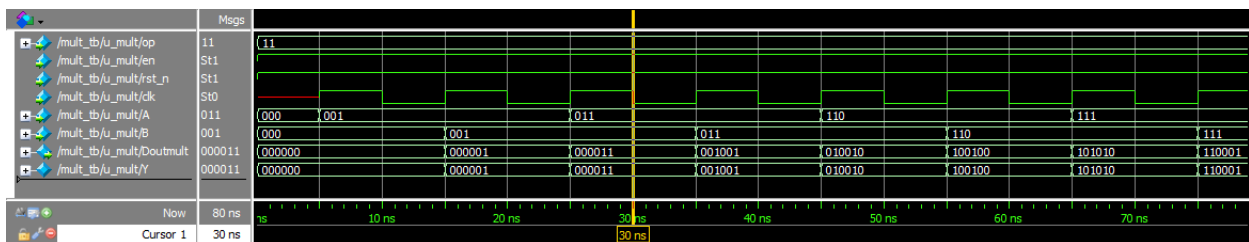


Figure 8: Output wave of the multiplication module

@ 30 ns

$$\begin{array}{r} B \ 001 \\ A \ 011 \\ \hline 000011 \end{array} = \text{Dout} = 000011$$

@ 50 ns

$$\begin{array}{r} B \ 011 \\ A \ 110 \\ \hline 010010 \end{array} = \text{Dout} = 010010$$

Function is confirmed.

The timing diagram displays the following signals and their values over time:

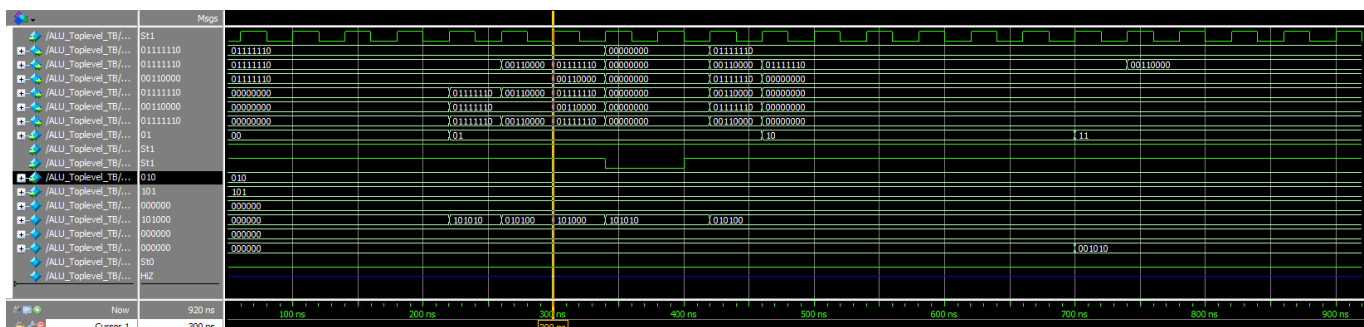
Signal	Value
/dsp_tb/u_dsp/op	11
/dsp_tb/u_dsp/en	St1
/dsp_tb/u_dsp/rst_n	St1
/dsp_tb/u_dsp/clk	S10
/dsp_tb/u_dsp/Doutxnor	010101
/dsp_tb/u_dsp/Doutshift	010101
/dsp_tb/u_dsp/Doutadd	010101
/dsp_tb/u_dsp/Doutmult	010101
/dsp_tb/u_dsp/dsp0	00110000
/dsp_tb/u_dsp/dsp1	01101101
/dsp_tb/u_dsp/dsp2	00000000
/dsp_tb/u_dsp/dsp3	00000000
/dsp_tb/u_dsp/dsp4	00000000
/dsp_tb/u_dsp/dsp5	00000000
/dsp_tb/u_dsp/Hex0	00110000
/dsp_tb/u_dsp/Hex1	01101101
/dsp_tb/u_dsp/Hex2	00000000
/dsp_tb/u_dsp/Hex3	00000000
/dsp_tb/u_dsp/Hex4	00000000
/dsp_tb/u_dsp/Hex5	00000000

The time scale ranges from 0 ns to 260 ns, with major ticks every 20 ns. The cursor is positioned at 0 ns.

The ALU was configured to select the XNOR, shift, add and multiply functions based off a 2-bit register as follows. Given Dout = 010101.

Table 4: Display module operation confirmation

Finally, the full ALU was tested and successfully wired each of the modules together. The reset function was tested and resulted in the display being blanked and the shift register function halting its progress. Each of the modules were properly selected when the appropriate operation control signal was present. A larger view of this wave output is shown in the Appendix, page #



12

Summary:

In this project we were tasked with constructing an ALU that had the XNOR, shift, add and multiply functions. It must take two 3-bit inputs and based off a synched clock, enable and reset signal and an operation select signal will perform arithmetic operations on those inputs. The output of that arithmetic operation is to be displayed on multiple 7-segment displays as either binary outputs or integer values. Multiple Verilog files were created to complete the creation of the ALU and after verification the system proved successful.

Many aspects of this project needed research and further understanding to accomplish. Little information on top level functionality was known before the completion of this project. However, it proved to be quite intuitive. Another issue faced was communication between modules. Wires can only run to individual outputs and this means that each module must have its own respective output. Optimization of the display module took some time. A few methods of implementation including splitting the output register into two integer values which would be displayed separately or running a for loop which would count the register and create two integers based off were attempted but a simple lookup table made in Excel proved to be not only the fastest but easiest to implement method. Furthermore, the display module is delayed by one clock signal from the output of the arithmetic units. This seems to not have an easy solution since the register must change before the display module can update. However, this is not an issue since the clock signal is quite high frequency and if tested on an FPGA the delay is negligible.

This project successfully taught the proper methods of creating and testing a simple multi-module logic unit. Considerations of a dedicated synch module could prove useful but no issues arose without one. Finally testing with a true FPGA would more greatly solidify the knowledge garnered from this project as a real-world application is always useful since theory only translates so well.

Citations:

- [1] Y. Zhou, "Digital VLSI Circuit Design Course Project," 2020. .
- [2] R. Marston, "Using Seven-Segment Displays - Part 1," *Nuts and Volts Magazine*, May-2001. [Online]. Available: <https://www.nutsvolts.com/magazine/article/using-seven-segment-displays-part-1>. [Accessed: 08-Dec-2020].

Appendices:

XNOR code and testbench

```
1 // EELE 4054: Digital VLSI Design
2 // Author: Zakary Williams
3 // Date: Dec 7, 2020
4 //
5 // ALU xnor module
6
7 module xnor_ALU(input [1:0]op, input en, input rst_n, input clk, input [2:0]A, input [2:0]B, output [5:0]Doutxnor);
8
9     reg [2:0] Y; //Temp register for output
10
11     initial Y = 3'b0;
12     assign Doutxnor = Y;
13
14     always @ (posedge clk) begin
15         if (en) begin //ALU must be enabled to receive inputs
16             if (~rst_n) //Global reset
17                 Y = 1'b0;
18             else begin
19                 if (op == 2'b00) //Selecting which ALU function is used
20                     Y = A^~B; //Bitwise XNOR function
21             end
22         end
23     end
24 endmodule
```

```

1  `timescale 1ns / 1ns
2
3  module xnor_tb;
4
5  reg [2:0] A;
6  reg [2:0] B;
7  reg clk;
8  reg rst_n;
9  reg en;
10 reg [1:0] op;
11
12 wire [5:0] Doutxnor;
13
14 initial begin
15     A = 3'b000;
16     B = 3'b000;
17     rst_n = 1;
18     en = 1;
19     op = 2'b00;
20
21     #5 clk = 1; A = 3'b001;
22     #5 clk = 0;
23     #5 clk = 1; B = 3'b001;
24     #5 clk = 0;
25     #5 clk = 1; A = 3'b011;
26     #5 clk = 0;
27     #5 clk = 1; B = 3'b011;
28     #5 clk = 0;
29     #5 clk = 1; A = 3'b110;
30     #5 clk = 0;
31     #5 clk = 1; B = 3'b110;
32     #5 clk = 0;
33     #5 clk = 1; A = 3'b111;
34     #5 clk = 0;
35     #5 clk = 1; B = 3'b111;
36     #5 clk = 0;
37
38     end
39
40 xnor_ALU u_xnor(
41     .op(op),
42     .en(en),
43     .rst_n(rst_n),
44     .clk(clk),
45     .A(A),
46     .B(B),
47     .Doutxnor(Doutxnor)
48 );
49 endmodule

```


Shift code and testbench

```

1 // EELE 4054: Digital VLSI Design
2 // Author: Zakary Williams
3 // Date: Dec 7, 2020
4 //
5 // ALU shift reg module
6
7 module shiftreg_ALU(input [1:0]op, input en, input rst_n, input clk, input [2:0]A, input [2:0]B, output [5:0]Doutshift);
8
9     reg shift0; //Declaration of shift registers
10    reg shift1;
11    reg shift2;
12    reg shift3;
13    reg shift4;
14    reg shift5;
15
16    reg refresh;
17
18
19    assign Doutshift[0] = shift0; //Assign outputs
20    assign Doutshift[1] = shift1;
21    assign Doutshift[2] = shift2;
22    assign Doutshift[3] = shift3;
23    assign Doutshift[4] = shift4;
24    assign Doutshift[5] = shift5;
25
26
27    initial begin //Initialising each registers
28        shift5 = 1'b0;
29        shift4 = 1'b0;
30        shift3 = 1'b0;
31        shift2 = 1'b0;
32        shift1 = 1'b0;
33        shift0 = 1'b0;
34        refresh = 1'b0;
35    end
36
37    always @ (posedge clk) begin
38        if (en) begin //ALU must be enabled to receive inputs
39            if (~rst_n) begin //Global reset
40                shift5 = 1'b0;
41                shift4 = 1'b0;
42                shift3 = 1'b0;
43                shift2 = 1'b0;
44                shift1 = 1'b0;
45                shift0 = 1'b0;
46                refresh = 1'b0;
47            end
48            if (op == 2'b01) begin //Selecting which ALU function is used
49                if (refresh == 1'b0) begin //refreshing the shift registers with the given inputs and shifting 1 bit
50                    shift0 = B[0];
51                    shift1 = B[1];
52                    shift2 = B[2];
53                    shift3 = A[0];
54                    shift4 = A[1];
55                    shift5 = A[2];
56                    shift5 = shift4;
57                    shift4 = shift3;
58                    shift3 = shift2;
59                    shift2 = shift1;
60                    shift1 = shift0;
61                    shift0 = 1'b0;
62                    refresh = 1'b1;
63                end
64                else begin //Shifting all registers 1 bit
65                    shift5 = shift4;
66                    shift4 = shift3;
67                    shift3 = shift2;
68                    shift2 = shift1;
69                    shift1 = shift0;
70                    shift0 = 1'b0;
71                end
72            end
73        end
74    end
75 endmodule

```

```

1  `timescale 1ns/ 1ns
2
3  module shift_tb;
4
5  reg [2:0] A;
6  reg [2:0] B;
7  reg clk;
8  reg rst_n;
9  reg en;
10 reg [1:0] op;
11
12 wire [5:0] Doutshift;
13
14 initial begin
15     A = 3'b001;
16     B = 3'b001;
17     rst_n = 1;
18     en = 1;
19     op = 2'b01;
20
21     #5 clk = 1;
22     #5 clk = 0;
23     #5 clk = 1;
24     #5 clk = 0;
25     #5 clk = 1;
26     #5 clk = 0;
27     #5 clk = 1;
28     #5 clk = 0;
29     #5 clk = 1;
30     #5 clk = 0;
31     #5 clk = 1; rst_n = 0;
32     #5 clk = 0;
33     #5 clk = 1;
34     #5 clk = 0; rst_n = 1;
35     #5 clk = 1;
36     #5 clk = 0;
37     #5 clk = 1; en = 0;
38     #5 clk = 0;
39     #5 clk = 1;
40     #5 clk = 0; en = 1;
41     #5 clk = 1;
42     #5 clk = 0;
43     #5 clk = 1;
44     #5 clk = 0;
45
46 end
47 shiftreg_ALU u_shift(
48     .op(op),
49     .en(en),
50     .rst_n(rst_n),
51     .clk(clk),
52     .A(A),
53     .B(B),
54     .Doutshift(Doutshift)
55 );
56 endmodule

```

Add code and testbench

```
1 // EELE 4054: Digital VLSI Design
2 // Author: Zakary Williams
3 // Date: Dec 7, 2020
4 //
5 // ALU Add module
6
7 module add_ALU(input [1:0]op, input en, input rst_n, input clk, input [2:0]A, input [2:0]B, output [5:0]Doutadd, C);
8
9     reg [5:0] Y; //Temp register for output
10
11
12     initial Y = 6'b0;
13     assign Doutadd = Y;
14     assign C = Y[3];
15
16     always @ (posedge clk) begin
17         if (en) begin //ALU must be enabled to receive inputs
18             if (~rst_n) //Global reset
19                 Y = 1'b0;
20             else begin
21                 if(op == 2'b10) //Selecting which ALU function is used
22                     Y = A+B; //Addition function
23             end
24         end
25     end
26 endmodule
```

```

1  `timescale 1ns/ 1ns
2
3  module add_tb;
4
5      reg [2:0] A;
6      reg [2:0] B;
7      reg clk;
8      reg rst_n;
9      reg en;
10     reg [1:0]op;
11
12     wire [5:0] Doutadd;
13     wire C;
14
15     initial begin
16         A = 3'b000;
17         B = 3'b000;
18         rst_n = 1;
19         en = 1;
20         op = 2'b10;
21
22         #5 clk = 1; A = 3'b001;
23         #5 clk = 0;
24         #5 clk = 1; B = 3'b001;
25         #5 clk = 0;
26         #5 clk = 1; A = 3'b011;
27         #5 clk = 0;
28         #5 clk = 1; B = 3'b011;
29         #5 clk = 0;
30         #5 clk = 1; A = 3'b110;
31         #5 clk = 0;
32         #5 clk = 1; B = 3'b110;
33         #5 clk = 0;
34         #5 clk = 1; A = 3'b111;
35         #5 clk = 0;
36         #5 clk = 1; B = 3'b111;
37         #5 clk = 0;
38
39     end
40
41     add_ALU u_add(
42         .op(op) ,
43         .en(en) ,
44         .rst_n(rst_n) ,
45         .clk(clk) ,
46         .A(A) ,
47         .B(B) ,
48         .Doutadd(Doutadd) ,
49         .C(C)
50     );
51 endmodule

```

Multiplier code and testbench

```
1 // EELE 4054: Digital VLSI Design
2 // Author: Zakary Williams
3 // Date: Dec 7, 2020
4 //
5 // ALU multiplier module
6
7 module multiplier_ALU(input [1:0]op, input en, input rst_n, input clk, input [2:0]A, input [2:0]B, output [5:0]Doutmult);
8
9     reg [5:0] Y; //Temp register for output
10
11     initial Y = 6'b0;
12     assign Doutmult = Y;
13
14     always @ (posedge clk) begin
15         if (en) begin //ALU must be enabled to receive inputs
16             if (~rst_n) //Global reset
17                 Y = 1'b0;
18             else begin
19                 if (op == 2'b11) begin //Selecting which ALU function is used
20                     Y = 6'b0;
21                     case(B) // Addition and shift implementation of multiplier
22                         3'b000: Y = 6'b0;
23                         3'b001: Y = {3'b0, A};
24                         3'b010: Y = {2'b0, A, 1'b0};
25                         3'b011: Y = {2'b0, A, 1'b0}+{3'b0, A};
26                         3'b100: Y = {1'b0, A, 2'b0};
27                         3'b101: Y = {1'b0, A, 2'b0}+{3'b0, A};
28                         3'b110: Y = {1'b0, A, 2'b0}+{2'b0, A, 1'b0};
29                         3'b111: Y = {1'b0, A, 2'b0}+{2'b0, A, 1'b0}+{3'b0, A};
30                         default Y = 6'b0;
31                     endcase
32                 end
33             end
34         end
35     end
36 endmodule
```

```

1  `timescale 1ns/ 1ns
2
3  module mult_tb;
4
5  reg [2:0] A;
6  reg [2:0] B;
7  reg clk;
8  reg rst_n;
9  reg en;
10 reg [1:0]op;
11
12 wire [5:0] Doutmult;
13
14 initial begin
15     A = 3'b0;
16     B = 3'b0;
17     rst_n = 1;
18     en = 1;
19     op = 2'b11;
20
21     #5 clk = 1; A = 3'b001;
22     #5 clk = 0;
23     #5 clk = 1; B = 3'b001;
24     #5 clk = 0;
25     #5 clk = 1; A = 3'b011;
26     #5 clk = 0;
27     #5 clk = 1; B = 3'b011;
28     #5 clk = 0;
29     #5 clk = 1; A = 3'b110;
30     #5 clk = 0;
31     #5 clk = 1; B = 3'b110;
32     #5 clk = 0;
33     #5 clk = 1; A = 3'b111;
34     #5 clk = 0;
35     #5 clk = 1; B = 3'b111;
36     #5 clk = 0;
37
38     end
39
40 multiplier_ALU u_mult(
41     .op(op),
42     .en(en),
43     .rst_n(rst_n),
44     .clk(clk),
45     .A(A),
46     .B(B),
47     .Doutmult(Doutmult)
48 );
49 endmodule

```

Display code and testbench

```

1 // EELE 4054: Digital VLSI Design
2 // Author: Sakshy Williams
3 // Date: Dec 7, 2020
4 //
5 // ALU display module
6
7 module disp_ALU(input [1:0]op, input en, input rst_n, input clk, input [5:0]Doutxnor, input [5:0]Doutshift, input [5:0]Doutadd, input [5:0]Doutmult, output [7:0]disp0, output [7:0]disp1, output [7:0]disp2, output [7:0]disp3, output [7:0]disp4, output [7:0]disp5);
8
9     reg [7:0] Hex0; //Initializing output registers
10    reg [7:0] Hex1;
11    reg [7:0] Hex2;
12    reg [7:0] Hex3;
13    reg [7:0] Hex4;
14    reg [7:0] Hex5;
15
16    assign disp0 = Hex0; //Assigning displays to those registers
17    assign disp1 = Hex1;
18    assign disp2 = Hex2;
19    assign disp3 = Hex3;
20    assign disp4 = Hex4;
21    assign disp5 = Hex5;
22
23
24    always @ (posedge clk) begin //Synched clock
25        if (~rst_n) begin //Global reset
26            Hex0 = 7'b0;
27            Hex1 = 7'b0;
28            Hex2 = 7'b0;
29            Hex3 = 7'b0;
30            Hex4 = 7'b0;
31            Hex5 = 7'b0;
32        end
33        else begin
34            Hex0 = 7'b0; //Clear
35            Hex1 = 7'b0;
36            Hex2 = 7'b0;
37            Hex3 = 7'b0;
38            Hex4 = 7'b0;
39            Hex5 = 7'b0;
40            case (op)
41                2'b00: begin //3 bit output of Xnor displayed (binary)
42                    case (Doutxnor[0])
43                        1'b0: Hex0=7'b11111110;
44                        1'b1: Hex0=7'b01100000;
45                        default: Hex0 = 7'b0;
46                    endcase
47                    case (Doutxnor[1])
48                        1'b0: Hex1=7'b11111110;
49                        1'b1: Hex1=7'b01100000;
50                        default: Hex1 = 7'b0;
51                    endcase
52                    case (Doutxnor[2])
53                        1'b0: Hex2=7'b11111110;
54                        1'b1: Hex2=7'b01100000;
55                        default: Hex2 = 7'b0;
56                    endcase
57                end

```

```

58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114

2'b01: begin // Shift register output displayed (binary)
    case (Doutshift[0])
        1'b0:Hex0=7'b1111110;
        1'b1:Hex0=7'b0110000;
        default: Hex0 = 7'b0;
    endcase
    case (Doutshift[1])
        1'b0:Hex1=7'b1111110;
        1'b1:Hex1=7'b0110000;
        default: Hex1 = 7'b0;
    endcase
    case (Doutshift[2])
        1'b0:Hex2=7'b1111110;
        1'b1:Hex2=7'b0110000;
        default: Hex2 = 7'b0;
    endcase
    case (Doutshift[3])
        1'b0:Hex3=7'b1111110;
        1'b1:Hex3=7'b0110000;
        default: Hex3 = 7'b0;
    endcase
    case (Doutshift[4])
        1'b0:Hex4=7'b1111110;
        1'b1:Hex4=7'b0110000;
        default: Hex4 = 7'b0;
    endcase
    case (Doutshift[5])
        1'b0:Hex5=7'b1111110;
        1'b1:Hex5=7'b0110000;
        default: Hex5 = 7'b0;
    endcase
end
2'b10: begin //Addition output display (integer)
    case (Doutadd) //dual 7 segment display lookup table
        //0-9
        6'b000000:begin Hex1=7'b1111110;Hex0=7'b1111110; end
        6'b000001:begin Hex1=7'b1111110;Hex0=7'b0110000; end
        6'b000010:begin Hex1=7'b1111110;Hex0=7'b1101101; end
        6'b000011:begin Hex1=7'b1111110;Hex0=7'b1111001; end
        6'b000100:begin Hex1=7'b1111110;Hex0=7'b0110011; end
        6'b000101:begin Hex1=7'b1111110;Hex0=7'b1011011; end
        6'b000110:begin Hex1=7'b1111110;Hex0=7'b1011111; end
        6'b000111:begin Hex1=7'b1111110;Hex0=7'b1110000; end
        6'b001000:begin Hex1=7'b1111110;Hex0=7'b1111111; end
        6'b001001:begin Hex1=7'b1111110;Hex0=7'b1110011; end
        //10-19
        6'b001010:begin Hex1=7'b0110000;Hex0=7'b1111110; end
        6'b001011:begin Hex1=7'b0110000;Hex0=7'b0110000; end
        6'b001100:begin Hex1=7'b0110000;Hex0=7'b1101101; end
        6'b001101:begin Hex1=7'b0110000;Hex0=7'b1111001; end
        6'b001110:begin Hex1=7'b0110000;Hex0=7'b0110011; end
        6'b001111:begin Hex1=7'b0110000;Hex0=7'b1011011; end
        6'b010000:begin Hex1=7'b0110000;Hex0=7'b1011111; end
        6'b010001:begin Hex1=7'b0110000;Hex0=7'b1110000; end
        6'b010010:begin Hex1=7'b0110000;Hex0=7'b1111111; end
        6'b010011:begin Hex1=7'b0110000;Hex0=7'b1110011; end

```



```

115 //20-29
116 6'b010100:begin Hex1=7'b1101101;Hex0=7'b1111110; end
117 6'b010101:begin Hex1=7'b1101101;Hex0=7'b0110000; end
118 6'b010110:begin Hex1=7'b1101101;Hex0=7'b1101101; end
119 6'b010111:begin Hex1=7'b1101101;Hex0=7'b1111001; end
120 6'b011000:begin Hex1=7'b1101101;Hex0=7'b0110011; end
121 6'b011001:begin Hex1=7'b1101101;Hex0=7'b1011011; end
122 6'b011010:begin Hex1=7'b1101101;Hex0=7'b1011111; end
123 6'b011011:begin Hex1=7'b1101101;Hex0=7'b1110000; end
124 6'b011100:begin Hex1=7'b1101101;Hex0=7'b1111111; end
125 6'b011101:begin Hex1=7'b1101101;Hex0=7'b1110011; end
126 //30-39
127 6'b011111:begin Hex1=7'b1111001;Hex0=7'b1111110; end
128 6'b100000:begin Hex1=7'b1111001;Hex0=7'b0110000; end
129 6'b100001:begin Hex1=7'b1111001;Hex0=7'b1101101; end
130 6'b100010:begin Hex1=7'b1111001;Hex0=7'b1111001; end
131 6'b100011:begin Hex1=7'b1111001;Hex0=7'b0110011; end
132 6'b100100:begin Hex1=7'b1111001;Hex0=7'b1011011; end
133 6'b100101:begin Hex1=7'b1111001;Hex0=7'b1011111; end
134 6'b100110:begin Hex1=7'b1111001;Hex0=7'b1110000; end
135 6'b100111:begin Hex1=7'b1111001;Hex0=7'b1111111; end
136 6'b101000:begin Hex1=7'b1111001;Hex0=7'b1110011; end
137 //40-49
138 6'b101001:begin Hex1=7'b0110011;Hex0=7'b1111110; end
139 6'b101010:begin Hex1=7'b0110011;Hex0=7'b0110000; end
140 6'b101011:begin Hex1=7'b0110011;Hex0=7'b1101101; end
141 6'b101100:begin Hex1=7'b0110011;Hex0=7'b1111001; end
142 6'b101101:begin Hex1=7'b0110011;Hex0=7'b0110011; end
143 6'b101110:begin Hex1=7'b0110011;Hex0=7'b1011011; end
144 6'b101111:begin Hex1=7'b0110011;Hex0=7'b1011111; end
145 6'b110000:begin Hex1=7'b0110011;Hex0=7'b1110000; end
146 6'b110001:begin Hex1=7'b0110011;Hex0=7'b1111111; end
147 6'b110010:begin Hex1=7'b0110011;Hex0=7'b1110011; end
148 //50-59
149 6'b110011:begin Hex1=7'b1011011;Hex0=7'b1111110; end
150 6'b110100:begin Hex1=7'b1011011;Hex0=7'b0110000; end
151 6'b110101:begin Hex1=7'b1011011;Hex0=7'b1101101; end
152 6'b110110:begin Hex1=7'b1011011;Hex0=7'b1111001; end
153 6'b110111:begin Hex1=7'b1011011;Hex0=7'b0110011; end
154 6'b111000:begin Hex1=7'b1011011;Hex0=7'b1011011; end
155 6'b111001:begin Hex1=7'b1011011;Hex0=7'b1011111; end
156 6'b111010:begin Hex1=7'b1011011;Hex0=7'b1110000; end
157 6'b111011:begin Hex1=7'b1011011;Hex0=7'b1111111; end
158 6'b111100:begin Hex1=7'b1011011;Hex0=7'b1110011; end
159 //60-63
160 6'b111101:begin Hex1=7'b1011111;Hex0=7'b1111110; end
161 6'b111110:begin Hex1=7'b1011111;Hex0=7'b0110000; end
162 6'b111111:begin Hex1=7'b1011111;Hex0=7'b1101101; end
163 default: Hex0 = 7'b0;
164 endcase
165 end
166 2'b11: begin //Multiplication output display (integer)
167     case (Doutmult) //dual 7 segment Hexlay lookup table
168         //0-9
169         6'b000000:begin Hex1=7'b1111110;Hex0=7'b1111110; end
170         6'b000001:begin Hex1=7'b1111110;Hex0=7'b0110000; end
171         6'b000010:begin Hex1=7'b1111110;Hex0=7'b1101101; end

```

```

172 6'b000011:begin Hex1=7'b1111110;Hex0=7'b1111001; end
173 6'b000100:begin Hex1=7'b1111110;Hex0=7'b0110011; end
174 6'b000101:begin Hex1=7'b1111110;Hex0=7'b1011011; end
175 6'b000110:begin Hex1=7'b1111110;Hex0=7'b1011111; end
176 6'b000111:begin Hex1=7'b1111110;Hex0=7'b1110000; end
177 6'b001000:begin Hex1=7'b1111110;Hex0=7'b1111111; end
178 6'b001001:begin Hex1=7'b1111110;Hex0=7'b1110011; end
179 //10-19
180 6'b001010:begin Hex1=7'b0110000;Hex0=7'b1111110; end
181 6'b001011:begin Hex1=7'b0110000;Hex0=7'b0110000; end
182 6'b001100:begin Hex1=7'b0110000;Hex0=7'b1101101; end
183 6'b001101:begin Hex1=7'b0110000;Hex0=7'b1111001; end
184 6'b001110:begin Hex1=7'b0110000;Hex0=7'b0110011; end
185 6'b001111:begin Hex1=7'b0110000;Hex0=7'b1011011; end
186 6'b010000:begin Hex1=7'b0110000;Hex0=7'b1011111; end
187 6'b010001:begin Hex1=7'b0110000;Hex0=7'b1110000; end
188 6'b010010:begin Hex1=7'b0110000;Hex0=7'b1111111; end
189 6'b010011:begin Hex1=7'b0110000;Hex0=7'b1110011; end
190 //20-29
191 6'b010100:begin Hex1=7'b1101101;Hex0=7'b1111110; end
192 6'b010101:begin Hex1=7'b1101101;Hex0=7'b0110000; end
193 6'b010110:begin Hex1=7'b1101101;Hex0=7'b1101101; end
194 6'b010111:begin Hex1=7'b1101101;Hex0=7'b1111001; end
195 6'b011000:begin Hex1=7'b1101101;Hex0=7'b0110011; end
196 6'b011001:begin Hex1=7'b1101101;Hex0=7'b1011011; end
197 6'b011010:begin Hex1=7'b1101101;Hex0=7'b1011111; end
198 6'b011011:begin Hex1=7'b1101101;Hex0=7'b1110000; end
199 6'b011100:begin Hex1=7'b1101101;Hex0=7'b1111111; end
200 6'b011101:begin Hex1=7'b1101101;Hex0=7'b1110011; end
201 //30-39
202 6'b011111:begin Hex1=7'b1111001;Hex0=7'b1111110; end
203 6'b100000:begin Hex1=7'b1111001;Hex0=7'b0110000; end
204 6'b100001:begin Hex1=7'b1111001;Hex0=7'b1101101; end
205 6'b100010:begin Hex1=7'b1111001;Hex0=7'b1111001; end
206 6'b100011:begin Hex1=7'b1111001;Hex0=7'b0110011; end
207 6'b100100:begin Hex1=7'b1111001;Hex0=7'b1011011; end
208 6'b100101:begin Hex1=7'b1111001;Hex0=7'b1011111; end
209 6'b100110:begin Hex1=7'b1111001;Hex0=7'b1110000; end
210 6'b100111:begin Hex1=7'b1111001;Hex0=7'b1111111; end
211 6'b101000:begin Hex1=7'b1111001;Hex0=7'b1110011; end
212 //40-49
213 6'b101001:begin Hex1=7'b0110011;Hex0=7'b1111110; end
214 6'b101010:begin Hex1=7'b0110011;Hex0=7'b0110000; end
215 6'b101011:begin Hex1=7'b0110011;Hex0=7'b1101101; end
216 6'b101100:begin Hex1=7'b0110011;Hex0=7'b1111001; end
217 6'b101101:begin Hex1=7'b0110011;Hex0=7'b0110011; end
218 6'b101110:begin Hex1=7'b0110011;Hex0=7'b1011011; end
219 6'b101111:begin Hex1=7'b0110011;Hex0=7'b1011111; end
220 6'b110000:begin Hex1=7'b0110011;Hex0=7'b1110000; end
221 6'b110001:begin Hex1=7'b0110011;Hex0=7'b1111111; end
222 6'b110010:begin Hex1=7'b0110011;Hex0=7'b1110011; end
223 //50-59
224 6'b110011:begin Hex1=7'b1011011;Hex0=7'b1111110; end
225 6'b110100:begin Hex1=7'b1011011;Hex0=7'b0110000; end
226 6'b110101:begin Hex1=7'b1011011;Hex0=7'b1101101; end
227 6'b110110:begin Hex1=7'b1011011;Hex0=7'b1111001; end
228 6'b110111:begin Hex1=7'b1011011;Hex0=7'b0110011; end

```

```

227         6'b110110:begin Hex1=7'b1011011;Hex0=7'b1111001; end
228         6'b110111:begin Hex1=7'b1011011;Hex0=7'b0110011; end
229         6'b111000:begin Hex1=7'b1011011;Hex0=7'b1011011; end
230         6'b111001:begin Hex1=7'b1011011;Hex0=7'b1011111; end
231         6'b111010:begin Hex1=7'b1011011;Hex0=7'b1110000; end
232         6'b111011:begin Hex1=7'b1011011;Hex0=7'b1111111; end
233         6'b111100:begin Hex1=7'b1011011;Hex0=7'b1110011; end
234         //60-63
235         6'b111101:begin Hex1=7'b1011111;Hex0=7'b1111110; end
236         6'b111110:begin Hex1=7'b1011111;Hex0=7'b0110000; end
237         6'b111111:begin Hex1=7'b1011111;Hex0=7'b1101101; end
238         default: Hex0 = 7'b0;
239     endcase
240     end
241     default: Hex0 = 7'b0;
242     endcase
243     end
244     end
245 endmodule

```

```

1  `timescale 1ns/ 1ns
2
3  module disp_tb;
4
5  reg [2:0] A;
6  reg [2:0] B;
7  reg clk;
8  reg rst_n;
9  reg en;
10 reg [1:0]op;
11 reg [5:0]Doutxnor;
12 reg [5:0]Doutshift;
13 reg [5:0]Doutadd;
14 reg [5:0]Doutmult;
15
16
17 wire [7:0]disp0; wire [7:0]disp1; wire [7:0]disp2; wire [7:0]disp3; wire [7:0]disp4; wire [7:0]disp5;
18
19 initial begin
20     A = 3'b0;
21     B = 3'b0;
22     rst_n = 1;
23     en = 1;
24     Doutxnor = 6'b010101;
25     Doutshift = 6'b010101;
26     Doutadd = 6'b010101;
27     Doutmult = 6'b010101;
28     op = 2'b00;
29
30     #5 clk = 1;
31     #5 clk = 0;
32     #5 clk = 1;
33     #5 clk = 0;
34     #5 clk = 1; op = 2'b00;
35     #5 clk = 0;
36     #5 clk = 1;
37     #5 clk = 0;
38     #5 clk = 1;
39     #5 clk = 0;
40     #5 clk = 1; op = 2'b01;
41     #5 clk = 0;
42     #5 clk = 1;
43     #5 clk = 0;
44     #5 clk = 1;
45     #5 clk = 0;
46     #5 clk = 1; op = 2'b10;
47     #5 clk = 0;
48     #5 clk = 1;
49     #5 clk = 0;
50     #5 clk = 1;
51     #5 clk = 0;
52     #5 clk = 1; op = 2'b11;
53     #5 clk = 0;
54     #5 clk = 1;
55     #5 clk = 0;
56     #5 clk = 1;
57     #5 clk = 0;
58     #5 clk = 1; op = 2'b00;

```

```

45     #5 clk = 0;
46     #5 clk = 1; op = 2'b10;
47     #5 clk = 0;
48     #5 clk = 1;
49     #5 clk = 0;
50     #5 clk = 1;
51     #5 clk = 0;
52     #5 clk = 1; op = 2'b11;
53     #5 clk = 0;
54     #5 clk = 1;
55     #5 clk = 0;
56     #5 clk = 1;
57     #5 clk = 0;
58     #5 clk = 1; op = 2'b00;
59     #5 clk = 0;
60     #5 clk = 1;
61     #5 clk = 0;
62     #5 clk = 1;
63     #5 clk = 0;
64     #5 clk = 1; op = 2'b01;
65     #5 clk = 0;
66     #5 clk = 1;
67     #5 clk = 0;
68     #5 clk = 1;
69     #5 clk = 0;
70     #5 clk = 1; op = 2'b10;
71     #5 clk = 0;
72     #5 clk = 1;
73     #5 clk = 0;
74     #5 clk = 1;
75     #5 clk = 0;
76     #5 clk = 1; op = 2'b11;
77     #5 clk = 0;
78     #5 clk = 1;
79     #5 clk = 0;
80     #5 clk = 1;
81     #5 clk = 0;
82
83     end
84
85     disp_ALU u_disp(
86         .op(op),
87         .en(en),
88         .rst_n(rst_n),
89         .clk(clk),
90         .Doutxnor(Doutxnor),
91         .Doutshift(Doutshift),
92         .Doutadd(Doutadd),
93         .Doutmult(Doutmult),
94         .disp0(disp0),
95         .disp1(disp1),
96         .disp2(disp2),
97         .disp3(disp3),
98         .disp4(disp4),
99         .disp5(disp5)
100     );
101 endmodule

```

Top level code and testbench

```
1 // EELE 4054: Digital VLSI Design
2 // Author: Zakary Williams
3 // Date: Dec 7, 2020
4 //
5 // ALU wrapper module
6
7 module ALU_Toplevel(
8
9     //Clock
10    input          CLK_50,
11
12    //7 Seg
13    output         [7:0]    disp0,
14    output         [7:0]    disp1,
15    output         [7:0]    disp2,
16    output         [7:0]    disp3,
17    output         [7:0]    disp4,
18    output         [7:0]    disp5,
19
20    //Operation select
21    input          [1:0]    op,
22
23    //En/Rst_n
24    input          en, rst_n,
25
26    //Inputs
27    input          [2:0]A,
28    input          [2:0]B
29 );
30
31 //Wire
32 wire            [5:0]    Doutxnor;
33 wire            [5:0]    Doutshift;
34 wire            [5:0]    Doutadd;
35 wire            [5:0]    Doutmult;
36 wire            [5:0]    C;
37
38 // Structural coding
39
40 xnor_ALU u_xnor(
41     .op(op),
42     .en(en),
43     .rst_n(rst_n),
44     .clk(CLK_50),
45     .A(A),
46     .B(B),
47     .Doutxnor(Doutxnor)
48 );
49
50 shiftreg_ALU u_shift(
51     .op(op),
52     .en(en),
53     .rst_n(rst_n),
54     .clk(CLK_50),
55     .A(A),
56     .B(B),
57     .Doutshift(Doutshift)
58 );
```

```

58     );
59
60     add_ALU u_add(
61         .op(op),
62         .en(en),
63         .rst_n(rst_n),
64         .clk(clk),
65         .A(A),
66         .B(B),
67         .Doutadd(Doutadd),
68         .C(C)
69     );
70     multiplier_ALU u_mult(
71         .op(op),
72         .en(en),
73         .rst_n(rst_n),
74         .clk(CLK_50),
75         .A(A),
76         .B(B),
77         .Doutmult(Doutmult)
78     );
79
80     disp_ALU u_disp(
81         .op(op),
82         .en(en),
83         .rst_n(rst_n),
84         .clk(CLK_50),
85         .Doutxnor(Doutxnor),
86         .Doutshift(Doutshift),
87         .Doutadd(Doutadd),
88         .Doutmult(Doutmult),
89         .disp0(disp0),
90         .disp1(disp1),
91         .disp2(disp2),
92         .disp3(disp3),
93         .disp4(disp4),
94         .disp5(disp5)
95     );
96
97     endmodule

```

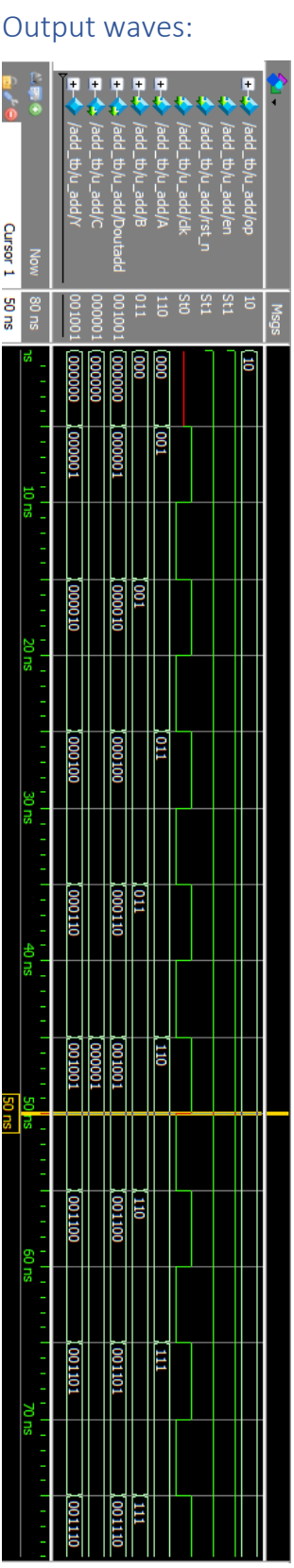
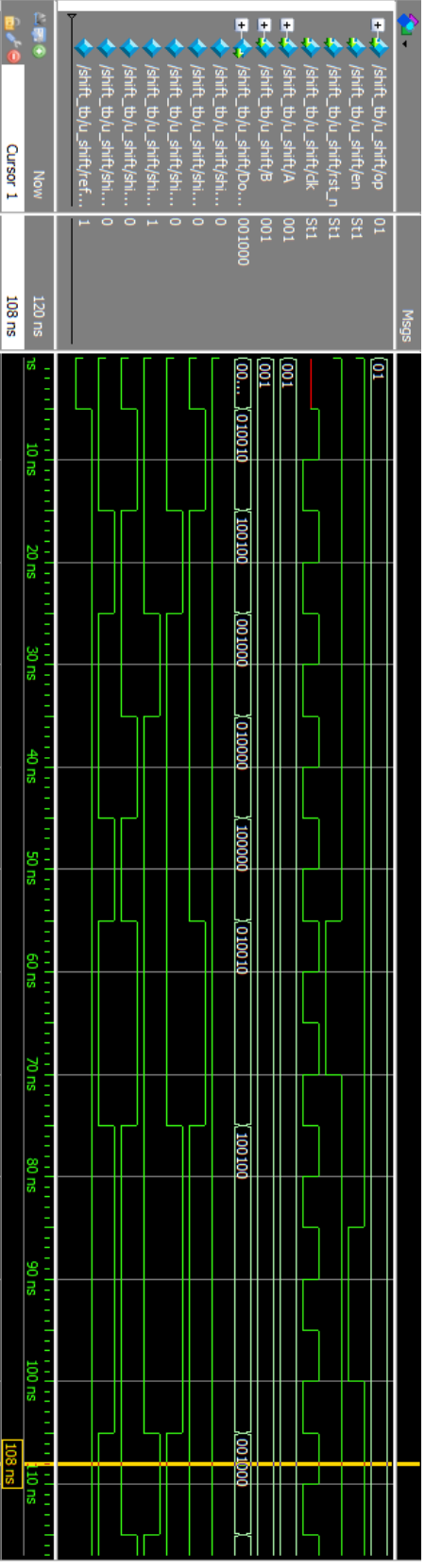
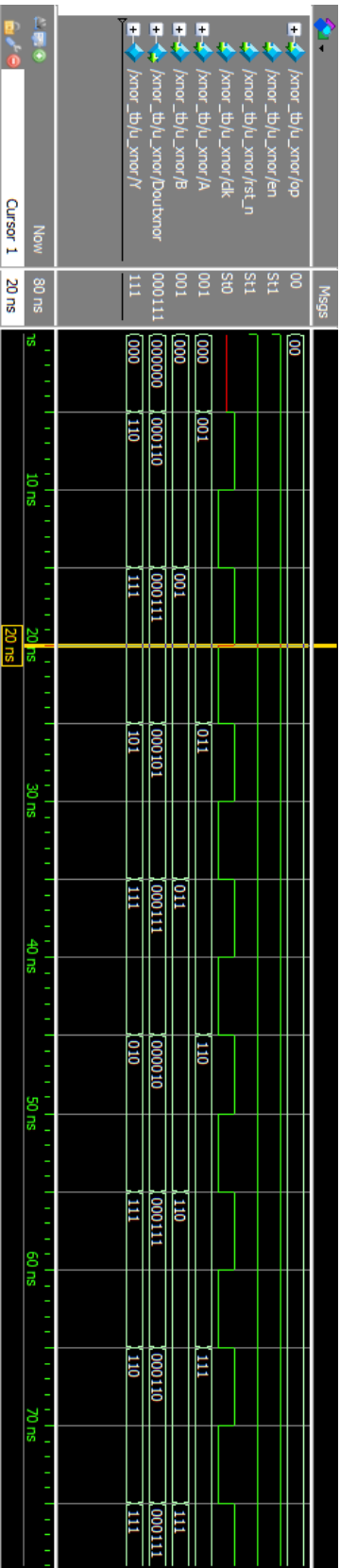
```

1  `timescale 10ns/ 10ns
2
3  module ALU_Toplevel_TB;
4
5  reg [2:0] A;
6  reg [2:0] B;
7  reg clk;
8  reg rst_n;
9  reg en;
10 reg [1:0]op;
11
12 wire [7:0]disp0; wire [7:0]disp1; wire [7:0]disp2; wire [7:0]disp3; wire [7:0]disp4; wire [7:0]disp5;
13
14 initial begin
15     A = 3'b010;
16     B = 3'b101;
17     rst_n = 1;
18     en = 1;
19     op = 2'b00;
20
21     #2 clk = 1;
22     #2 clk = 0;
23     #2 clk = 1;
24     #2 clk = 0;
25     #2 clk = 1; op = 2'b00;
26     #2 clk = 0;
27     #2 clk = 1;
28     #2 clk = 0;
29     #2 clk = 1;
30     #2 clk = 0;
31     #2 clk = 1; op = 2'b01;
32     #2 clk = 0;
33     #2 clk = 1;
34     #2 clk = 0;
35     #2 clk = 1;
36     #2 clk = 0;
37     #2 clk = 1; rst_n = 1'b0;
38     #2 clk = 0;
39     #2 clk = 1;
40     #2 clk = 0; rst_n = 1'b1;
41     #2 clk = 1;
42     #2 clk = 0;
43     #2 clk = 1; op = 2'b10;
44     #2 clk = 0;
45     #2 clk = 1;
46     #2 clk = 0;
47     #2 clk = 1;
48     #2 clk = 0;
49     #2 clk = 1;
50     #2 clk = 0;
51     #2 clk = 1;
52     #2 clk = 0;
53     #2 clk = 1;
54     #2 clk = 0;
55     #2 clk = 1; op = 2'b11;
56     #2 clk = 0;
57     #2 clk = 1;
58     #2 clk = 0;

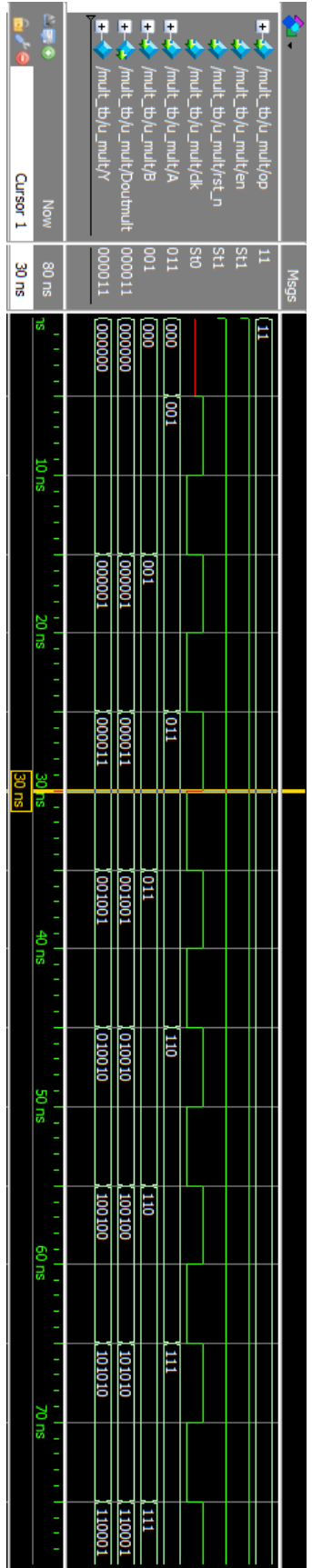
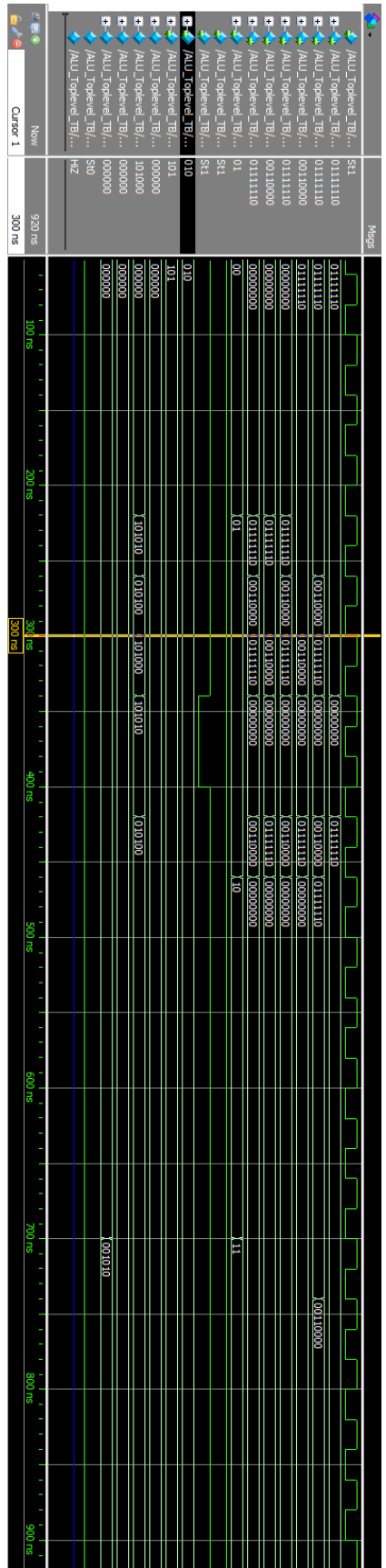
```



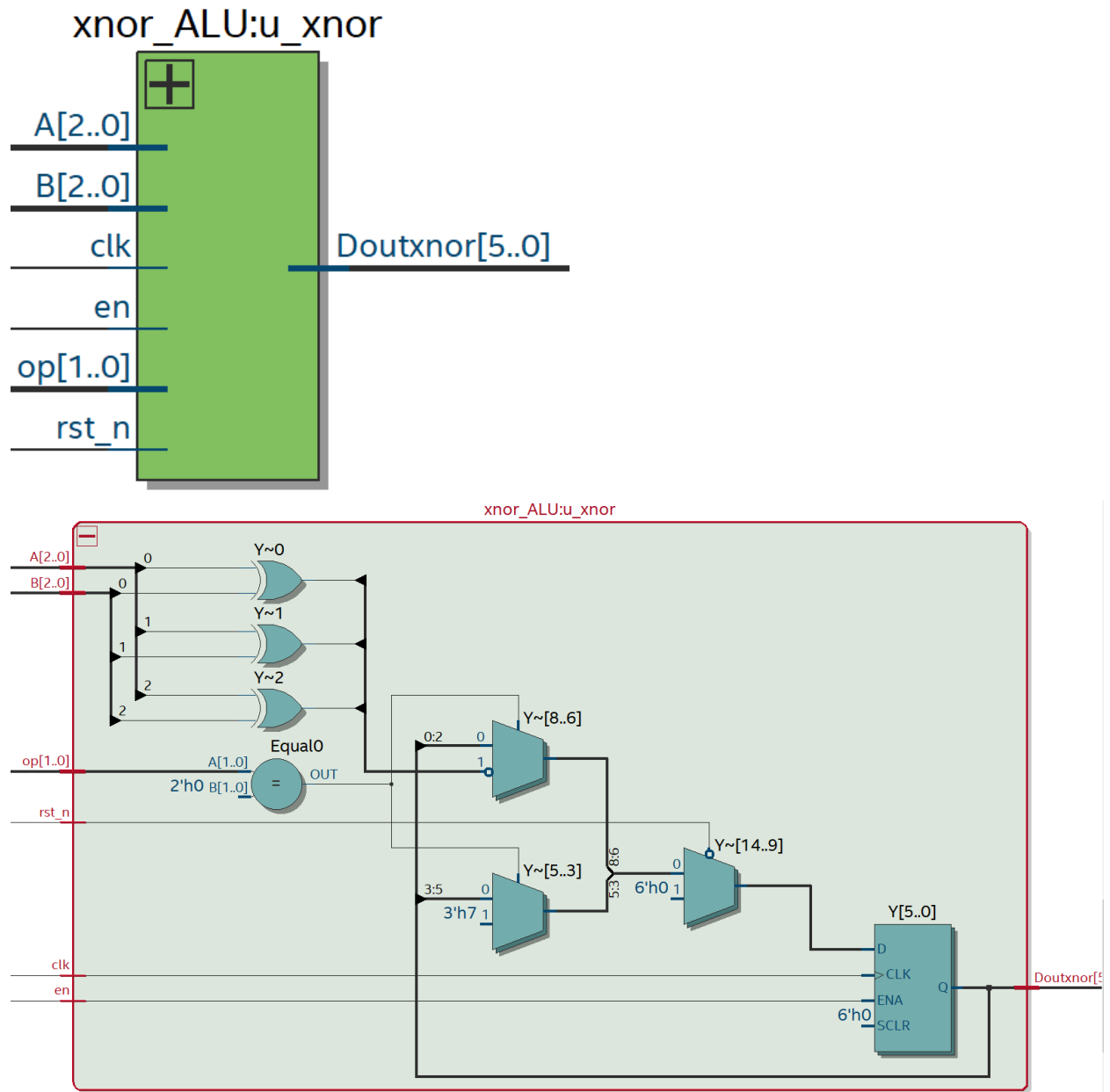
```
59         #2 clk = 1;
60         #2 clk = 0;
61         #2 clk = 1;
62         #2 clk = 0;
63         #2 clk = 1;
64         #2 clk = 0;
65         #2 clk = 1;
66         #2 clk = 0;
67
68
69
70     end
71
72     ALU_Toplevel u_ALU(
73     .op(op) ,
74     .en(en) ,
75     .rst_n(rst_n) ,
76     .CLK_50(clk) ,
77     .A(A) ,
78     .B(B) ,
79     .disp0(disp0) ,
80     .disp1(disp1) ,
81     .disp2(disp2) ,
82     .disp3(disp3) ,
83     .disp4(disp4) ,
84     .disp5(disp5));
85 endmodule
```



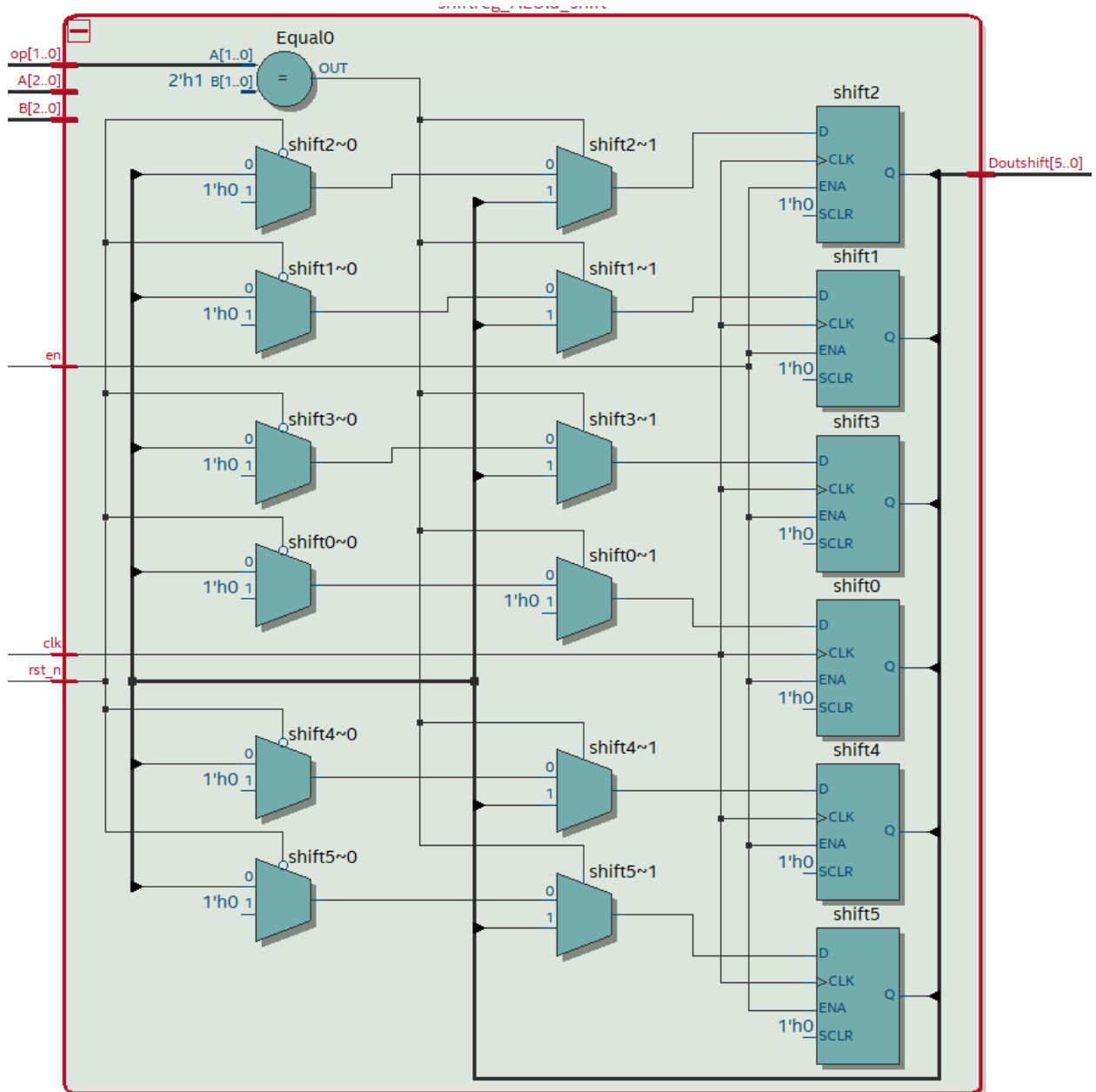
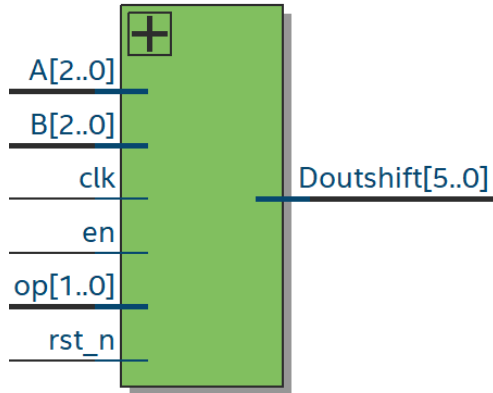
Output waves:

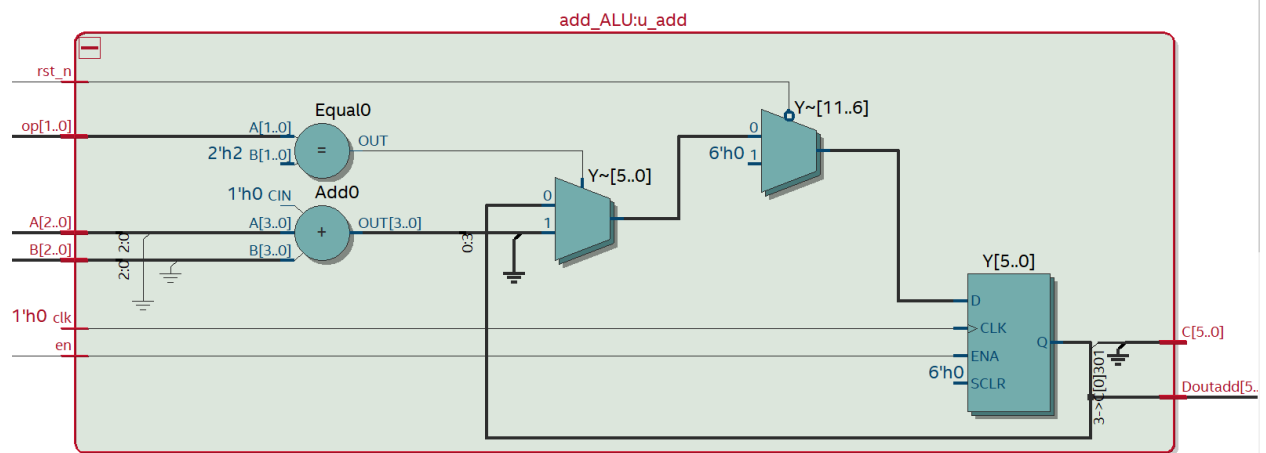
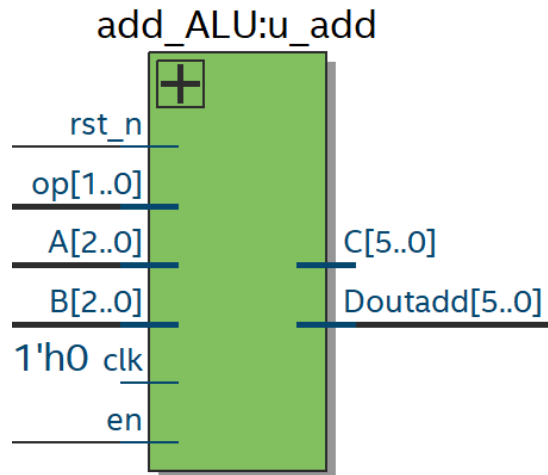


Blocks



shiftreg_ALU:u_shift





multiplier_ALU:u_mult

