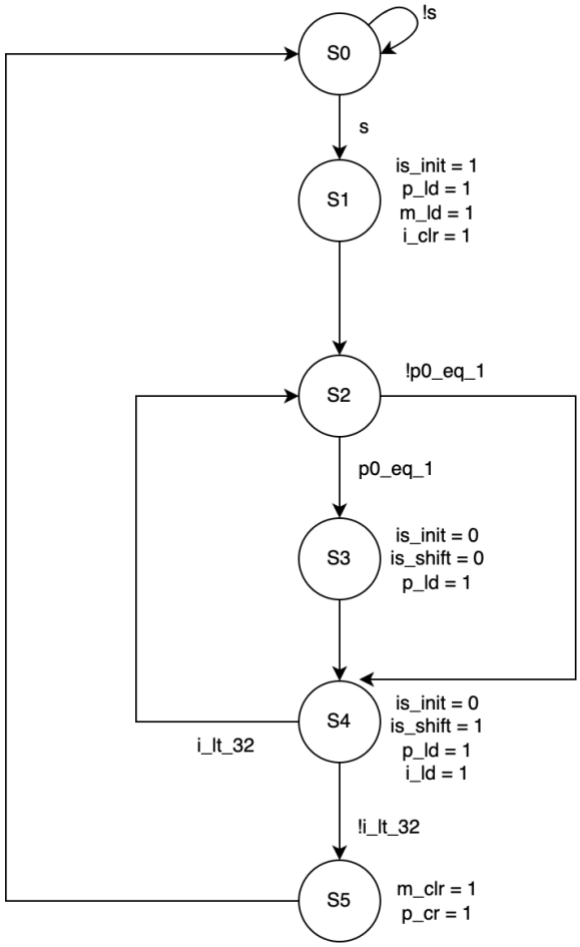
Mult State Diagram



Mult Truth Table

inputs						outputs										
s_2	s_1	s_0	start	p0_eq_1	i_lt_32	n_2	n_1	n_0	p_ld	m_ld	i_clr	is_init	is_shift	i_ld	p_clr	m_clr
0	0	0	0	X	X	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	X	X	0	0	1	0	0	0	0	0	0	0	0
0	0	1	X	X	X	0	1	0	1	1	1	1	0	0	0	0
0	1	0	X	0	X	1	0	0	0	0	0	0	0	0	0	0
0	1	0	X	1	X	0	1	1	0	0	0	0	0	0	0	0
0	1	1	X	X	X	1	0	0	1	0	0	0	0	0	0	0
1	0	0	X	X	0	1	0	1	1	0	0	0	1	1	0	0
1	0	0	X	X	1	0	1	0	1	0	0	0	1	1	0	0
1	0	1	X	X	X	0	0	0	0	0	0	0	0	0	1	1
1	1	0	X	X	X	Don't care condition										
1	1	1	X	X	X	Don't care condition										

n0 = s2's1's0start + s2's1s0'p0_eq_1 + s2s1's0'i_lt_32'
n1 = s2's1's0 + s2's1s0'p0_eq_1 + s2s1's0'i_lt_32
n2 = s2's1s0'p0_eq_1' + s2's1s0 + s2s1's0'i_lt_32'
p_ld = s2's1's0 + s2's1s0 + s2s1's0'
m_ld = s2's1's0
m_clr = s2s1's0
i_clr = s2's1's0
is_init = s2's1's0
is_shift = s2s1's0'
i_ld = s2s1's0'
p_clr = s2s1's0

half_adder

This module performs an addition operation on 2 1-bit inputs without a carry in.

testbench

```
`define DELAY 20
module half_adder_testbench();
reg a, b;
wire sum, carry_out;

half_adder hatb (sum, carry_out, a, b);

initial begin
a = 1'b0; b = 1'b0;
#`DELAY;
a = 1'b1; b = 1'b0;
#`DELAY;
a = 1'b0; b = 1'b1;
#`DELAY;
a = 1'b1; b = 1'b1;
end


initial
begin
$monitor("time = %2d, a =%1b, b=%1b, sum=%1b, carry_out=%1b", $time, a, b, sum, carry_out);
end

endmodule
```

testbench result

```
# time =  0, a =0, b=0, sum=0, carry_out=0
# time = 20, a =1, b=0, sum=1, carry_out=0
# time = 40, a =0, b=1, sum=1, carry_out=0
# time = 60, a =1, b=1, sum=0, carry_out=1
```

full_adder

This module performs an addition operation on 2 1-bit inputs with a carry in.

testbench

```
`define DELAY 20
module full_adder_testbench();
reg a, b, carry_in;
wire sum, carry_out;

full_adder fatb (sum, carry_out, a, b, carry_in);

initial begin
a = 1'b0; b = 1'b0; carry_in = 1'b0;
#`DELAY;
a = 1'b0; b = 1'b0; carry_in = 1'b1;
#`DELAY;
a = 1'b0; b = 1'b1; carry_in = 1'b0;
#`DELAY;
a = 1'b0; b = 1'b1; carry_in = 1'b1;
#`DELAY;
a = 1'b1; b = 1'b0; carry_in = 1'b0;
#`DELAY;
a = 1'b1; b = 1'b0; carry_in = 1'b1;
#`DELAY;
a = 1'b1; b = 1'b1; carry_in = 1'b0;
#`DELAY;
a = 1'b1; b = 1'b1; carry_in = 1'b1;
end


initial
begin
$monitor("time = %2d, a =%1b, b=%1b, carry_in=%1b, sum=%1b, carry_out=%1b", $time, a, b, carry_in, sum, carry_out);
end

endmodule
```

testbench result

```
# time =  0, a =0, b=0, carry_in=0, sum=0, carry_out=0
# time = 20, a =0, b=0, carry_in=1, sum=1, carry_out=0
# time = 40, a =0, b=1, carry_in=0, sum=1, carry_out=0
# time = 60, a =0, b=1, carry_in=1, sum=0, carry_out=1
# time = 80, a =1, b=0, carry_in=0, sum=1, carry_out=0
# time = 100, a =1, b=0, carry_in=1, sum=0, carry_out=1
# time = 120, a =1, b=1, carry_in=0, sum=0, carry_out=1
# time = 140, a =1, b=1, carry_in=1, sum=1, carry_out=1
```

_4bit_adder

This module performs an addition operation on 2 4-bit inputs with a carry in.

testbench

```
`define DELAY 20
module _4bit_adder_testbench();
reg [3:0] a, b;
reg carry_in;
wire carry_out;
wire [3:0] sum;

_4bit_adder _4ba (sum, carry_out, a, b, carry_in);

initial begin
a = 4'b0000; b = 4'b0000; carry_in = 1'b0;
#`DELAY;
a = 4'b0000; b = 4'b0000; carry_in = 1'b1;
#`DELAY;
a = 4'b0100; b = 4'b0001; carry_in = 1'b0;
#`DELAY;
a = 4'b0100; b = 4'b001; carry_in = 1'b1;
#`DELAY;
a = 4'b0011; b = 4'b0011; carry_in = 1'b0;
#`DELAY;
a = 4'b0011; b = 4'b0011; carry_in = 1'b1;
#`DELAY;
a = 4'b0001; b = 4'b0001; carry_in = 1'b0;
#`DELAY;
a = 4'b0001; b = 4'b0001; carry_in = 1'b1;
end


initial
begin
$monitor("time = %2d, a = %4b, b = %4b, carry_in = %1b, sum = %4b, carry_out = %1b", $time, a, b, carry_in, sum, carry_out);
end

endmodule
```

testbench result

```
# time =  0, a = 0000, b = 0000, carry_in = 0, sum = 0000, carry_out = 0
# time = 20, a = 0000, b = 0000, carry_in = 1, sum = 0001, carry_out = 0
# time = 40, a = 0100, b = 0001, carry_in = 0, sum = 0101, carry_out = 0
# time = 60, a = 0100, b = 0001, carry_in = 1, sum = 0110, carry_out = 0
# time = 80, a = 0011, b = 0011, carry_in = 0, sum = 0110, carry_out = 0
# time = 100, a = 0011, b = 0011, carry_in = 1, sum = 0111, carry_out = 0
# time = 120, a = 0001, b = 0001, carry_in = 0, sum = 0010, carry_out = 0
# time = 140, a = 0001, b = 0001, carry_in = 1, sum = 0011, carry_out = 0
```

_32bit_adder

This module performs an addition operation on 2 4-bit inputs with a carry in and an op code. Op code decides whether the adder perform an addition operation or a subtraction operation. This module also outputs set-less-than and overflow values.

testbench

```
initial begin
a = 32'd25;
b = 32'd81;
carry_in = 1'b0;
op = 0;
#`DELAY;
carry_in = 1'b1;
op = 0;
#`DELAY;
carry_in = 1'b1;
op = 1;
#`DELAY;
a = 32'd90;
b = 32'd10;
carry_in = 1'b0;
op = 0;
#`DELAY;
carry_in = 1'b1;
op = 0;
#`DELAY;
carry_in = 1'b1;
op = 1;
#`DELAY;
a = 32'h7FFF_FFFF;
b = 32'd0;
carry_in = 1'b1;
op = 0;
#`DELAY;
a = 32'h8000_0000;
b = 32'd1;
carry_in = 1'b0;
op = 1;
end
```

testbench result

```
# time =  0
# a = 10101010101010101010101010101010
# b = 01010101010101010101010101010101
# r = 00000000000000000000000000000000
#
# time = 20
# a = 11111111111111111111111111111111
# b = 11111111111111111111111111111111
# r = 11111111111111111111111111111111
#
# time = 40
# a = 00000000000000000000000000000000
# b = 00000000000000000000000000000000
# r = 00000000000000000000000000000000
#
# time = 60
# a = 11111111111111111111111111111111
# b = 00000000000000000000000000000000
# r = 00000000000000000000000000000000
```

_4bit_and

This module performs and operation on 2 4-bit inputs.

testbench

```
`define DELAY 20
module _4bit_and_testbench();
reg [3:0] a, b;
wire [3:0] r;

_4bit_and _4ba (r, a, b);

initial begin
a = 4'b1010; b = 4'b0101;
#`DELAY;
a = 4'b1111; b = 4'b1111;
#`DELAY;
a = 4'b0000; b = 4'b0000;
#`DELAY;
a = 4'b1111; b = 4'b0000;
end


initial
begin
$monitor("time = %2d, a = %4b, b = %4b, r = %4b", $time, a, b, r);
end

endmodule
```

testbench result

```
# time =  0, a = 1010, b = 0101, r = 0000
# time = 20, a = 1111, b = 1111, r = 1111
# time = 40, a = 0000, b = 0000, r = 0000
# time = 60, a = 1111, b = 0000, r = 0000
```

_32bit_and

This module performs and operation on 2 32-bit inputs.

testbench

```
`define DELAY 20
module _32bit_and_testbench();
reg [31:0] a, b;
wire [31:0] r;

_32bit_and _32ba (r, a, b);

initial begin
a = 32'hAAAA_AAAA; b = 32'h5555_5555;
#`DELAY;
a = 32'hFFFF_FFFF; b = 32'hFFFF_FFFF;
#`DELAY;
a = 32'h0000_0000; b = 32'h0000_0000;
#`DELAY;
a = 32'hFFFF_FFFF; b = 32'h0000_0000;
end


initial
begin
$monitor("time = %2d\na = %32b\nb = %32b\nr = %32b\n", $time, a, b, r);
end

endmodule
```

testbench result

```
# time =  0
# a = 10101010101010101010101010101010
# b = 01010101010101010101010101010101
# r = 00000000000000000000000000000000
#
# time = 20
# a = 11111111111111111111111111111111
# b = 11111111111111111111111111111111
# r = 11111111111111111111111111111111
#
# time = 40
# a = 00000000000000000000000000000000
# b = 00000000000000000000000000000000
# r = 00000000000000000000000000000000
#
# time = 60
# a = 11111111111111111111111111111111
# b = 00000000000000000000000000000000
# r = 00000000000000000000000000000000
```

_4bit_not

This module negates the 4-bit input.

testbench

```
`define DELAY 20
module _4bit_not_testbench();
reg [3:0] a;
wire [3:0] r;

_4bit_not _4bn (r, a);

initial begin
a = 4'b1010;
#`DELAY;
a = 4'b0101;
#`DELAY;
a = 4'b1111;
#`DELAY;
a = 4'b0000;
end


initial
begin
$monitor("time = %2d, a = %4b, r = %4b", $time, a, r);
end

endmodule
```

testbench result

```
# time =  0, a = 1010, r = 0101
# time = 20, a = 0101, r = 1010
# time = 40, a = 1111, r = 0000
# time = 60, a = 0000, r = 1111
```

_32bit_not

This module negates the 32-bit input.

testbench

```
`define DELAY 20
module _32bit_not_testbench();
reg [31:0] a;
wire [31:0] r;

_32bit_not _32bn (r, a);

initial begin
a = 32'hAAAA_AAAA;
#`DELAY;
a = 32'h5555_5555;
#`DELAY;
a = 32'hFFFF_FFFF;
#`DELAY;
a = 32'h0000_0000;
end


initial
begin
$monitor("time = %2d\na = %32b\nr = %32b", $time, a, r);
end

endmodule
```

testbench result

```
# time =  0
# a = 10101010101010101010101010101010
# r = 01010101010101010101010101010101
# time = 20
# a = 01010101010101010101010101010101
# r = 10101010101010101010101010101010
# time = 40
# a = 11111111111111111111111111111111
# r = 00000000000000000000000000000000
# time = 60
# a = 00000000000000000000000000000000
# r = 11111111111111111111111111111111
```

_4bit_or

This module performs or operation on 2 4-bit inputs.

testbench

```
`define DELAY 20
module _4bit_or_testbench();
reg [3:0] a, b;
wire [3:0] r;

_4bit_or _4bo (r, a, b);

initial begin
a = 4'b1010; b = 4'b0101;
#`DELAY;
a = 4'b1111; b = 4'b1111;
#`DELAY;
a = 4'b0000; b = 4'b0000;
#`DELAY;
a = 4'b1111; b = 4'b0000;
end


initial
begin
$monitor("time = %2d, a = %4b, b = %4b, r = %4b", $time, a, b, r);
end

endmodule
```

testbench result

```
# time =  0, a = 1010, b = 0101, r = 1111
# time = 20, a = 1111, b = 1111, r = 1111
# time = 40, a = 0000, b = 0000, r = 0000
# time = 60, a = 1111, b = 0000, r = 1111
```

_32bit_or

This module performs or operation on 2 32-bit inputs.

testbench

```
`define DELAY 20
module _32bit_or_testbench();
reg [31:0] a, b;
wire [31:0] r;

_32bit_or _32bo (r, a, b);

initial begin
a = 32'hAAAA_AAAA; b = 32'h5555_5555;
#`DELAY;
a = 32'hFFFF_FFFF; b = 32'hFFFF_FFFF;
#`DELAY;
a = 32'h0000_0000; b = 32'h0000_0000;
#`DELAY;
a = 32'hFFFF_FFFF; b = 32'h0000_0000;
end


initial
begin
$monitor("time = %2d\na = %32b\nb = %32b\nr = %32b\n", $time, a, b, r);
end

endmodule
```

testbench result

```
# time =  0
# a = 10101010101010101010101010101010
# b = 01010101010101010101010101010101
# r = 11111111111111111111111111111111
#
# time = 20
# a = 11111111111111111111111111111111
# b = 11111111111111111111111111111111
# r = 11111111111111111111111111111111
#
# time = 40
# a = 00000000000000000000000000000000
# b = 00000000000000000000000000000000
# r = 00000000000000000000000000000000
#
# time = 60
# a = 11111111111111111111111111111111
# b = 00000000000000000000000000000000
# r = 11111111111111111111111111111111
#
```

_4bit_xor

This module performs exclusive or operation on 2 4-bit inputs.

testbench

```
`define DELAY 20
module _4bit_xor_testbench();
reg [3:0] a, b;
wire [3:0] r;

_4bit_xor _4bx (r, a, b);

initial begin
a = 4'b1010; b = 4'b0101;
#`DELAY;
a = 4'b1111; b = 4'b1111;
#`DELAY;
a = 4'b0000; b = 4'b0000;
#`DELAY;
a = 4'b1111; b = 4'b0000;
end


initial
begin
$monitor("time = %2d, a = %4b, b = %4b, r = %4b", $time, a, b, r);
end

endmodule
```

testbench result

```
# time =  0, a = 1010, b = 0101, r = 1111
# time = 20, a = 1111, b = 1111, r = 0000
# time = 40, a = 0000, b = 0000, r = 0000
# time = 60, a = 1111, b = 0000, r = 1111
```

_32bit_xor

This module performs exclusive or operation on 2 32-bit inputs.

testbench

```
`define DELAY 20
module _32bit_xor_testbench();
reg [31:0] a, b;
wire [31:0] r;

_32bit_xor _32bx (r, a, b);

initial begin
a = 32'hAAAA_AAAA; b = 32'h5555_5555;
#`DELAY;
a = 32'hFFFF_FFFF; b = 32'hFFFF_FFFF;
#`DELAY;
a = 32'h0000_0000; b = 32'h0000_0000;
#`DELAY;
a = 32'hFFFF_FFFF; b = 32'h0000_0000;
end


initial
begin
$monitor("time = %2d\na = %32b\nb = %32b\nr = %32b\n", $time, a, b, r);
end

endmodule
```

testbench result

```
# time =  0
# a = 10101010101010101010101010101010
# b = 01010101010101010101010101010101
# r = 11111111111111111111111111111111
#
# time = 20
# a = 11111111111111111111111111111111
# b = 11111111111111111111111111111111
# r = 00000000000000000000000000000000
#
# time = 40
# a = 00000000000000000000000000000000
# b = 00000000000000000000000000000000
# r = 00000000000000000000000000000000
#
# time = 60
# a = 11111111111111111111111111111111
# b = 00000000000000000000000000000000
# r = 11111111111111111111111111111111
```

_32bit_nor

This module performs nor operation on 2 32-bit inputs.

testbench

```
`define DELAY 20
module _32bit_nor_testbench();
reg [31:0] a, b;
wire [31:0] r;

_32bit_nor _32bnor (r, a, b);

initial begin
a = 32'hAAAA_AAAA; b = 32'h5555_5555;
#`DELAY;
a = 32'hFFFF_FFFF; b = 32'hFFFF_FFFF;
#`DELAY;
a = 32'h0000_0000; b = 32'h0000_0000;
#`DELAY;
a = 32'hFFFF_FFFF; b = 32'h0000_0000;
end


initial
begin
$monitor("time = %2d\na = %32b\nb = %32b\nr = %32b\n", $time, a, b, r);
end

endmodule
```

testbench result

```
# time =  0
# a = 10101010101010101010101010101010
# b = 01010101010101010101010101010101
# r = 00000000000000000000000000000000
#
# time = 20
# a = 11111111111111111111111111111111
# b = 11111111111111111111111111111111
# r = 00000000000000000000000000000000
#
# time = 40
# a = 00000000000000000000000000000000
# b = 00000000000000000000000000000000
# r = 11111111111111111111111111111111
#
# time = 60
# a = 11111111111111111111111111111111
# b = 00000000000000000000000000000000
# r = 00000000000000000000000000000000
```

_32bit_mux_2x1

This module selects one of the 2 32-bit inputs as an output according to the select bit.

testbench

```
`define DELAY 20
module _32bit_mux_2x1_testbench();
reg [31:0] a, b;
reg s;
wire [31:0] r;

_32bit_mux_2x1 mux0(r, s, a, b);

initial begin
a = 32'h0000_0000; b = 32'h0000_0001; s = 1'b0;
#`DELAY;
a = 32'h0000_0000; b = 32'h0000_0001; s = 1'b1;
end

initial
begin
$monitor("time = %2d\na = %32b\nb = %32b\ns = %1b\nr = %32b", $time, a, b, s, r);
end
endmodule
```

testbench result

```
# time =  0
# a = 00000000000000000000000000000000
# b = 00000000000000000000000000000001
# s = 0
# r = 00000000000000000000000000000000
# time = 20
# a = 00000000000000000000000000000000
# b = 00000000000000000000000000000001
# s = 1
# r = 00000000000000000000000000000001
```

_32bit_mux_8x1

This module selects one of the 8 32-bit inputs as an output according to the select bits.

testbench

```
i0 = 32'h0000_0000;
i1 = 32'h0000_0001;
i2 = 32'h0000_0002;
i3 = 32'h0000_0003;
i4 = 32'h0000_0004;
i5 = 32'h0000_0005;
i6 = 32'h0000_0006;
i7 = 32'h0000_0007;
s2 = 1'b0;
s1 = 1'b0;
s0 = 1'b0;
#`DELAY;
s2 = 1'b0;
s1 = 1'b0;
s0 = 1'b1;
#`DELAY;
s2 = 1'b0;
s1 = 1'b1;
s0 = 1'b0;
#`DELAY;
s2 = 1'b0;
s1 = 1'b1;
s0 = 1'b1;
#`DELAY;
s2 = 1'b1;
s1 = 1'b0;
s0 = 1'b0;
#`DELAY;
s2 = 1'b1;
s1 = 1'b0;
s0 = 1'b1;
#`DELAY;
s2 = 1'b1;
s1 = 1'b1;
s0 = 1'b0;
```

testbench result

```
# time = 80
# i0 = 00000000000000000000000000000000
# i1 = 00000000000000000000000000000001
# i2 = 00000000000000000000000000000010
# i3 = 00000000000000000000000000000011
# i4 = 00000000000000000000000000000100
# i5 = 00000000000000000000000000000101
# i6 = 00000000000000000000000000000110
# i7 = 00000000000000000000000000000111
# s2 = 1
# s1 = 0
# s0 = 0
# r = 00000000000000000000000000000100
#
# time = 100
# i0 = 00000000000000000000000000000000
# i1 = 00000000000000000000000000000001
# i2 = 00000000000000000000000000000010
# i3 = 00000000000000000000000000000011
# i4 = 00000000000000000000000000000100
# i5 = 00000000000000000000000000000101
# i6 = 00000000000000000000000000000110
# i7 = 00000000000000000000000000000111
# s2 = 1
# s1 = 0
# s0 = 1
# r = 00000000000000000000000000000101
```

alu

This module performs alu operations according to the 3-bit op code. 000 = add, 001 = xor, 010 = sub, 011 = mult, 100 = slt, 101 = nor, 110 = and, 111 = or. I couldn't do the mult part, so when op code is 100 (mult), result is 0.

testbench

```
alu _alu(r, a, b, op);

initial begin
a = 32'd25; b = 32'd15; op = 3'b000;
#`DELAY;
op = 3'b001;
#`DELAY;
op = 3'b010;
#`DELAY;
op = 3'b011;
#`DELAY;
op = 3'b100;
#`DELAY;
op = 3'b101;
#`DELAY;
op = 3'b110;
#`DELAY;
op = 3'b111;
#`DELAY;
a = 32'd30; b = 32'd45; op = 3'b000;
#`DELAY;
op = 3'b001;
#`DELAY;
op = 3'b010;
#`DELAY;
op = 3'b011;
#`DELAY;
op = 3'b100;
#`DELAY;
op = 3'b101;
#`DELAY;
op = 3'b110;
#`DELAY;
op = 3'b111;
#`DELAY;
end
```

testbench result

```
# time =  0
# a = 00000000000000000000000000011001
# b = 00000000000000000000000000001111
# op code = 000
# result = 00000000000000000000000000101000
#
# time = 20
# a = 00000000000000000000000000011001
# b = 00000000000000000000000000001111
# op code = 001
# result = 00000000000000000000000000010110
#
# time = 40
# a = 00000000000000000000000000011001
# b = 00000000000000000000000000001111
# op code = 010
# result = 00000000000000000000000000001010
#
# time = 60
# a = 00000000000000000000000000011001
# b = 00000000000000000000000000001111
# op code = 011
# result = 00000000000000000000000000000000
#
# time = 80
# a = 00000000000000000000000000011001
# b = 00000000000000000000000000001111
# op code = 100
# result = 00000000000000000000000000000000
#
# time = 100
# a = 00000000000000000000000000011001
# b = 00000000000000000000000000001111
# op code = 101
# result = 11111111111111111111111111100000
#
# time = 120
# a = 00000000000000000000000000011001
# b = 00000000000000000000000000001111
# op code = 110
# result = 00000000000000000000000000001001
#
# time = 140
# a = 00000000000000000000000000011001
# b = 00000000000000000000000000001111
# op code = 111
# result = 00000000000000000000000000011111
```

```
# time = 160
# a = 00000000000000000000000000011110
# b = 00000000000000000000000000101101
# op code = 000
# result = 00000000000000000000000001001011
#
# time = 180
# a = 00000000000000000000000000011110
# b = 00000000000000000000000000101101
# op code = 001
# result = 00000000000000000000000000110011
#
# time = 200
# a = 00000000000000000000000000011110
# b = 00000000000000000000000000101101
# op code = 010
# result = 11111111111111111111111111110001
#
# time = 220
# a = 00000000000000000000000000011110
# b = 00000000000000000000000000101101
# op code = 011
# result = 00000000000000000000000000000000
#
# time = 240
# a = 00000000000000000000000000011110
# b = 00000000000000000000000000101101
# op code = 100
# result = 00000000000000000000000000000001
#
# time = 260
# a = 00000000000000000000000000011110
# b = 00000000000000000000000000101101
# op code = 101
# result = 11111111111111111111111111000000
#
# time = 280
# a = 00000000000000000000000000011110
# b = 00000000000000000000000000101101
# op code = 110
# result = 00000000000000000000000000001100
#
# time = 300
# a = 00000000000000000000000000011110
# b = 00000000000000000000000000101101
# op code = 111
# result = 00000000000000000000000000111111
```

P.S. Project's Verilog files are under workspace/CSE331 Hw2. I wrote the wrong hw number in directory name but quartus didn't let me change it, so I stuck with CSE331 Hw2.

Burak Yıldırım
1901042609