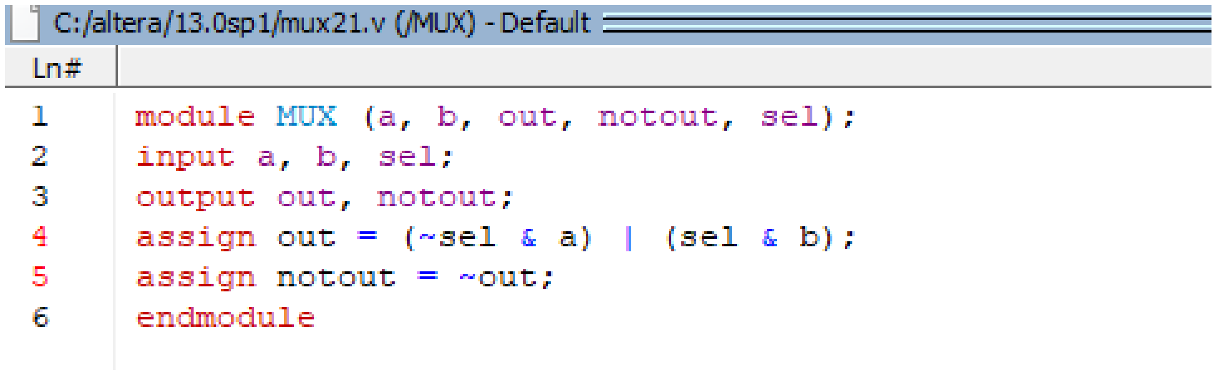
Class:	**CPE300L Digital Systems Architecture and Design 1001**	Semester:	**Fall 2024**
Points		Document author:	**Narek Kalikian**
		Author's email:	**kalikn1@unlv.nevada.edu**
		Document topic:	**Postlab 1**
Instructor's comments:			

1. Introduction / Theory of Operation

A hardware description language is a lower-level computer language that is used primarily in direct accordance and for implementation on hardware devices such as fully programmable gate arrays and integrated circuits. Two common HDLs are VHDL and Verilog.

2. Description of Experiments

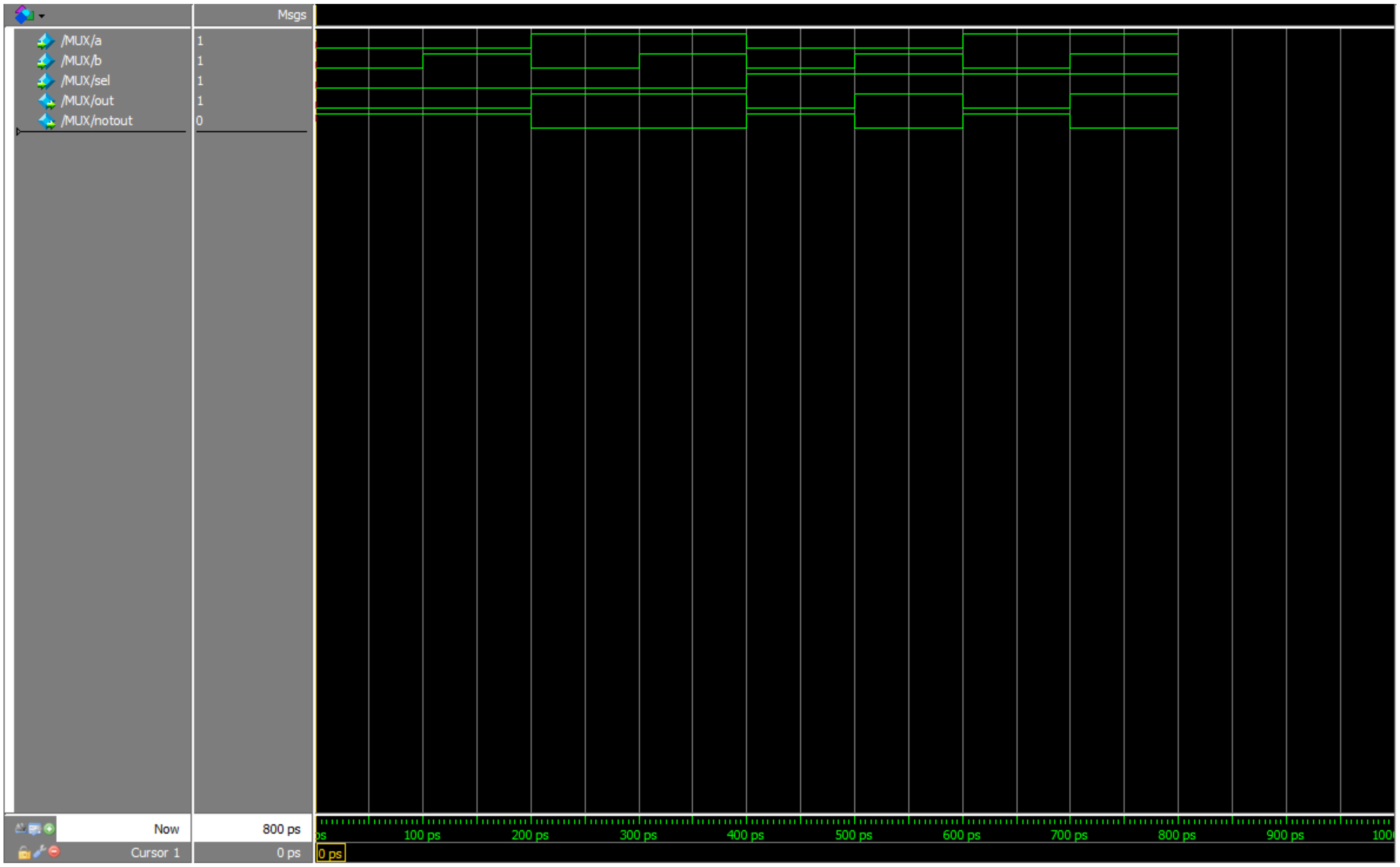
Experiment 1A - Continuous Assignments Code:



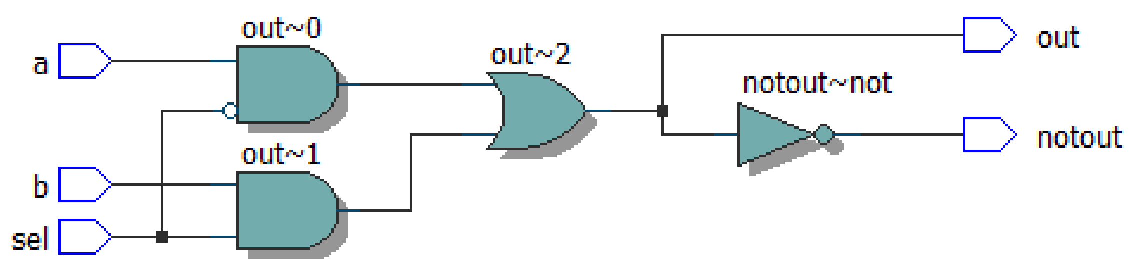
C:/altera/13.0sp1/mux21.v (/MUX) - Default

```
module MUX (a, b, out, notout, sel);
input a, b, sel;
output out, notout;
assign out = (~sel & a) | (sel & b);
assign notout = ~out;
endmodule
```

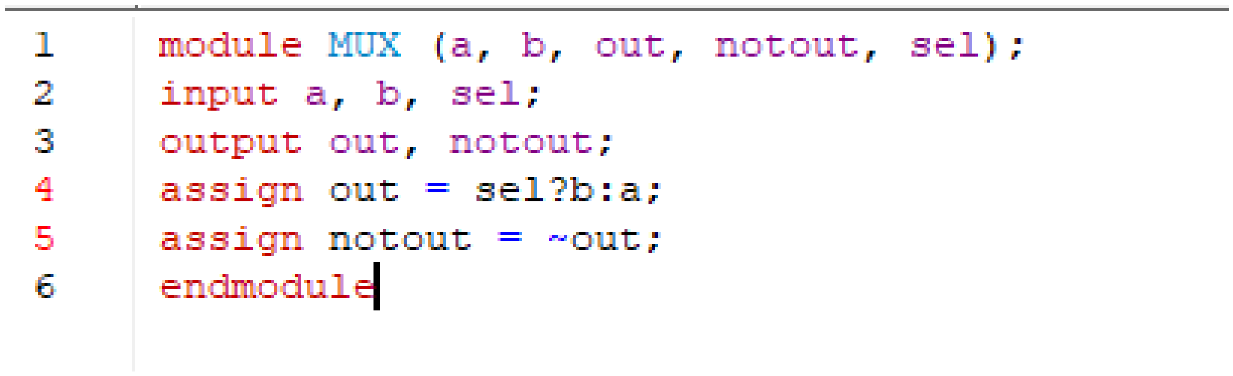
Experiment 1A - Continuous Assignments Waveforms:



Experiment 1A - Continuous Assignments Schematic:

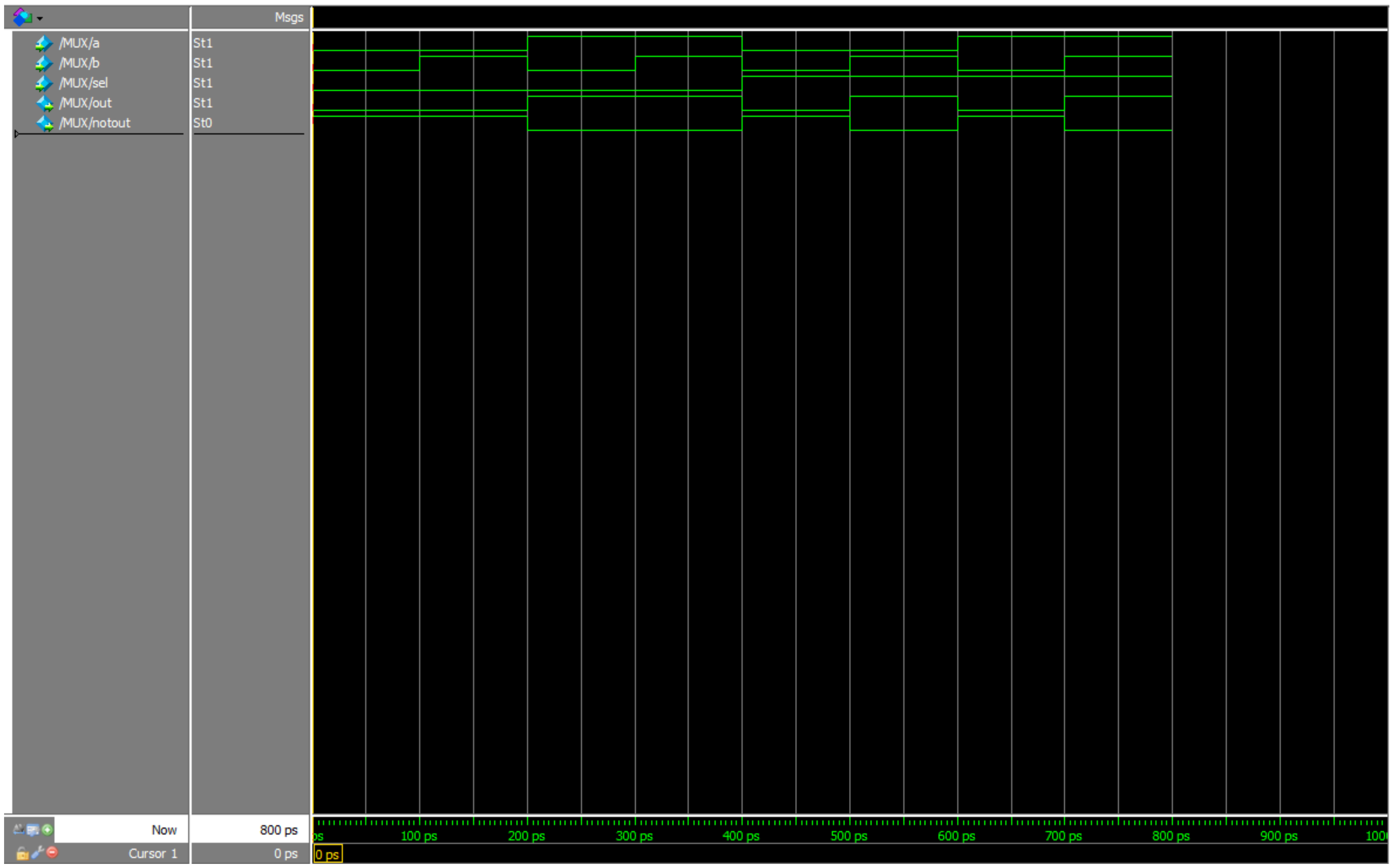


Experiment 1B - Conditional Operator Code:



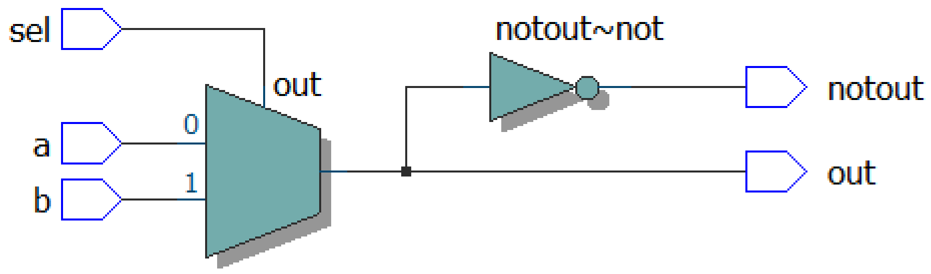
```
module MUX (a, b, out, notout, sel);
input a, b, sel;
output out, notout;
assign out = sel?b:a;
assign notout = ~out;
endmodule
```

Experiment 1B - Conditional Operator Waveforms:

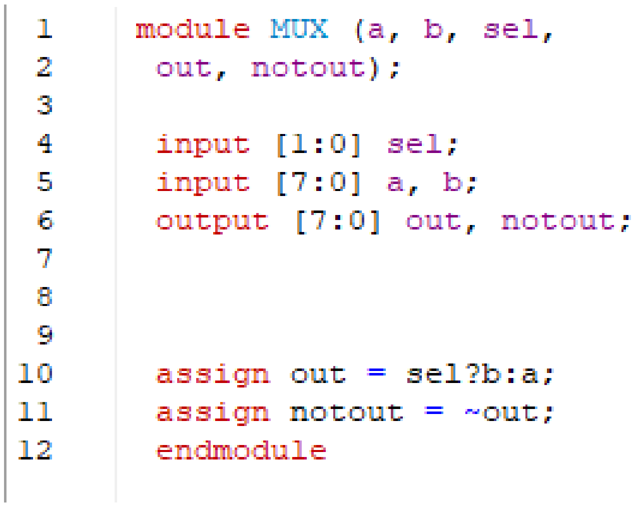


```
                Msgs
/MUX/a          St1
/MUX/b          St1
/MUX/sel        St1
/MUX/out        St1
/MUX/notout     St0

Now       800 ps
Cursor 1    0 ps
```

Experiment 1B - Conditional Operator Schematic:

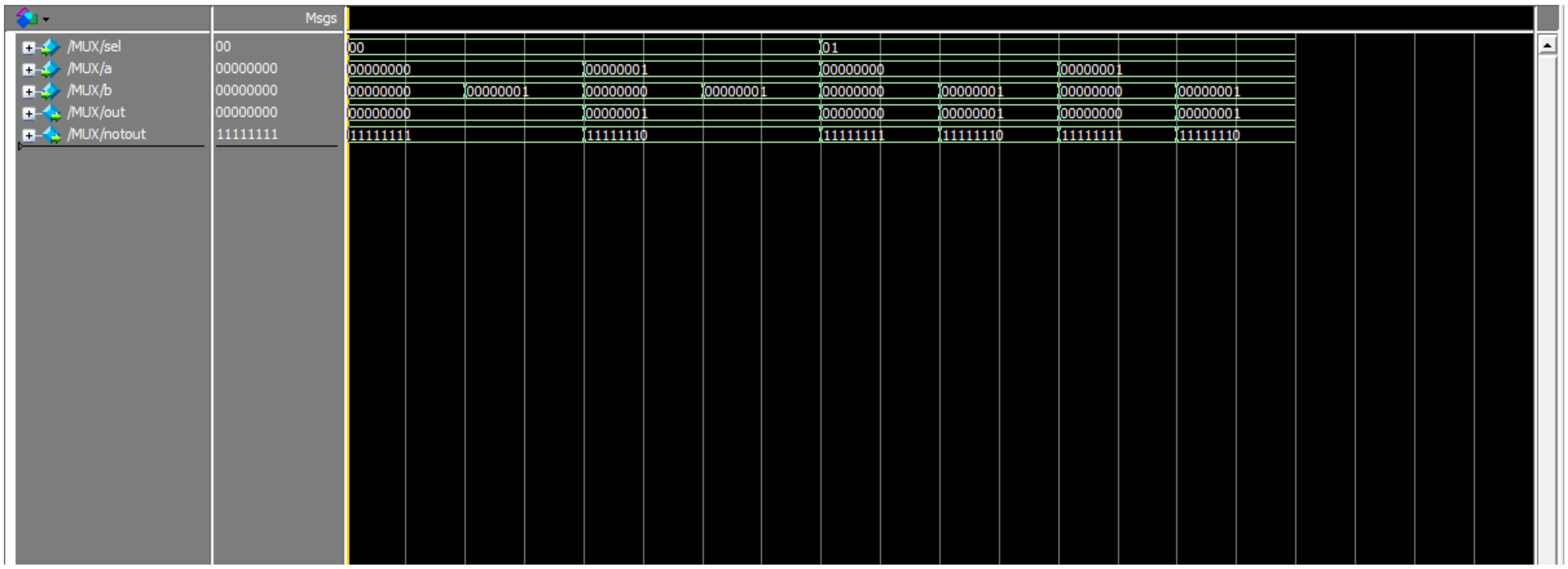


Experiment 1C - 8-bit Code:



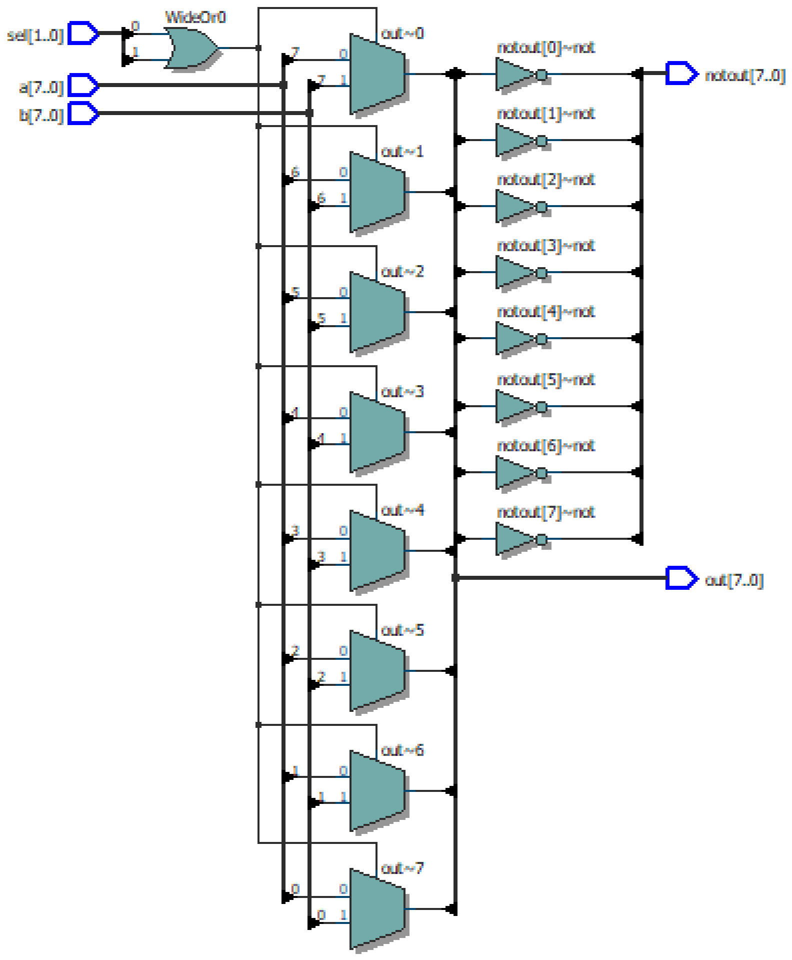
```
module MUX (a, b, sel,
 out, notout);

 input [1:0] sel;
 input [7:0] a, b;
 output [7:0] out, notout;



 assign out = sel?b:a;
 assign notout = ~out;
 endmodule
```

Experiment 1C - 8-bit Waveforms:

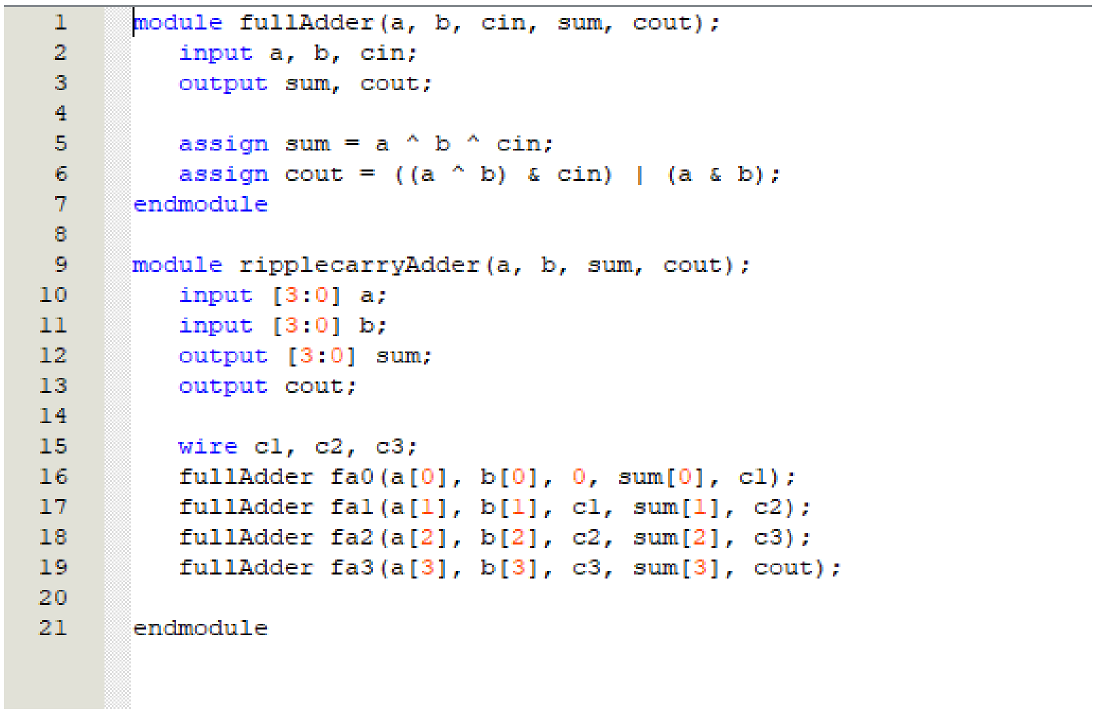


```
                Msgs
/MUX/sel        00         00                                                     01
/MUX/a          00000000   00000000                     00000001                  00000000                    00000001
/MUX/b          00000000   00000000      00000001       00000000      00000001    00000000      00000001      00000000      00000001
/MUX/out        00000000   00000000                     00000001                  00000000      00000001      00000000      00000001
/MUX/notout     11111111   11111111                     11111110                  11111111      11111110      11111111      11111110
```

Experiment 1C - 8-bit Schematic:

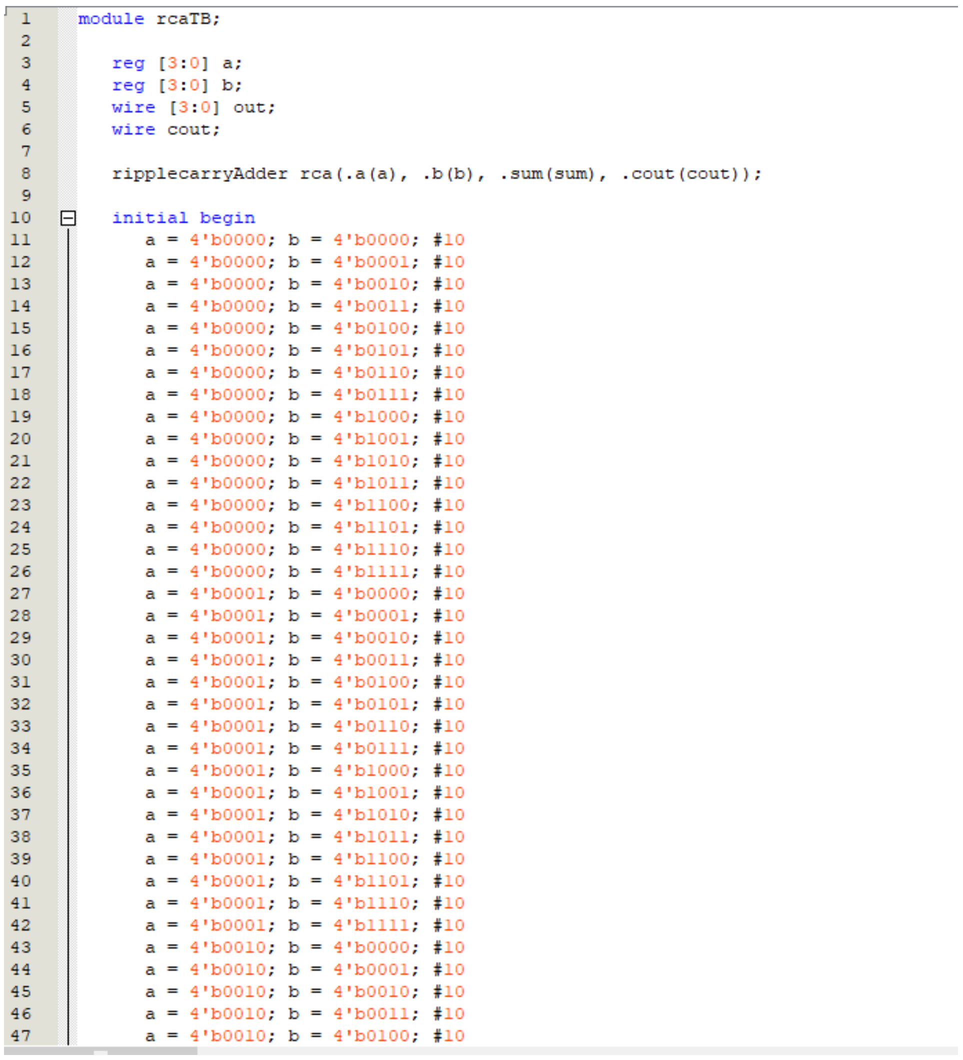


Experiment 2 - RCA Logic Code:

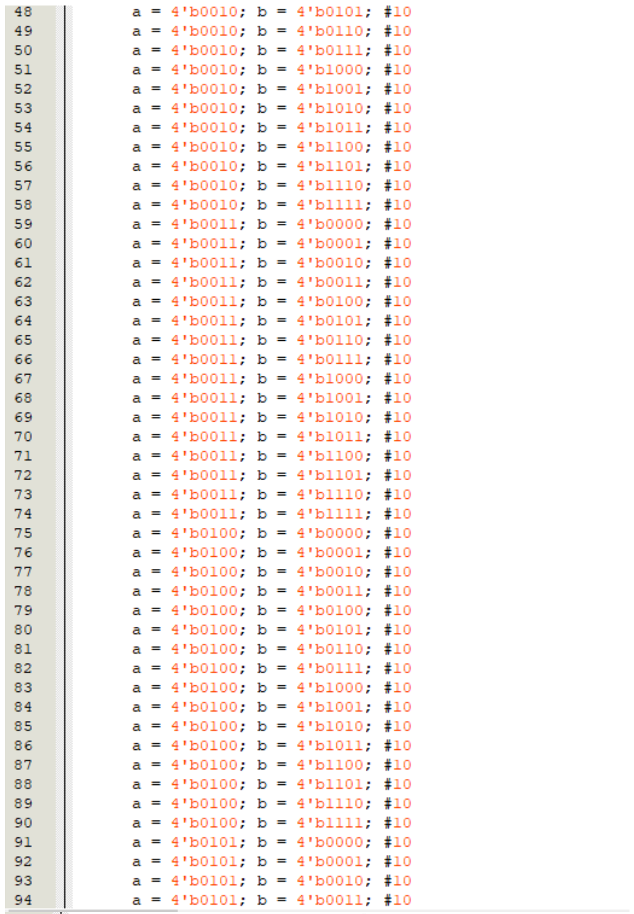


```verilog
module fullAdder(a, b, cin, sum, cout);
   input a, b, cin;
   output sum, cout;

   assign sum = a ^ b ^ cin;
   assign cout = ((a ^ b) & cin) | (a & b);
endmodule

module ripplecarryAdder(a, b, sum, cout);
   input [3:0] a;
   input [3:0] b;
   output [3:0] sum;
   output cout;

   wire c1, c2, c3;
   fullAdder fa0(a[0], b[0], 0, sum[0], c1);
   fullAdder fa1(a[1], b[1], c1, sum[1], c2);
   fullAdder fa2(a[2], b[2], c2, sum[2], c3);
   fullAdder fa3(a[3], b[3], c3, sum[3], cout);

endmodule
```

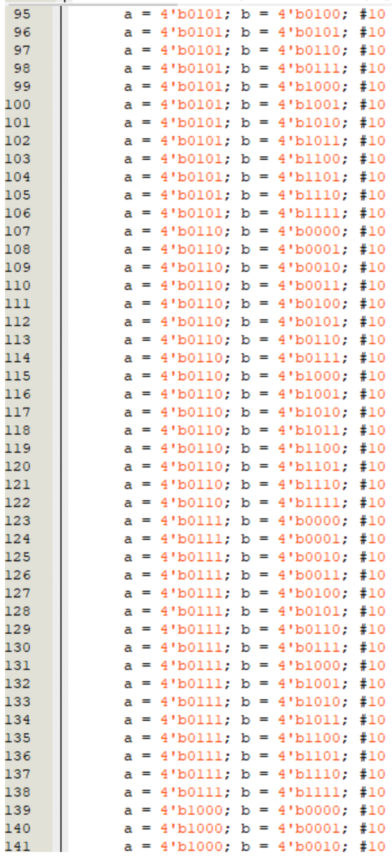
Experiment 2 - RCA Testbench Code:



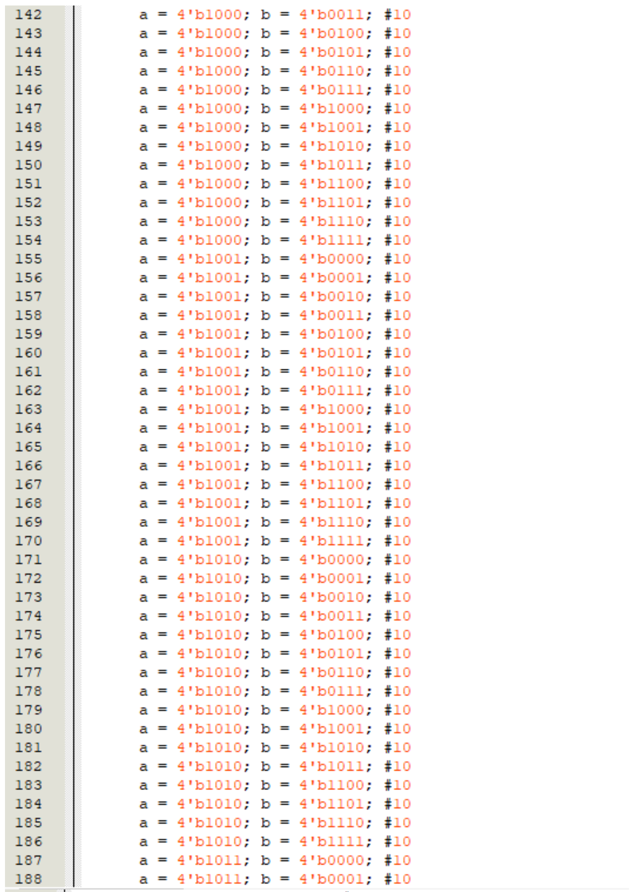
```verilog
module rcaTB;

   reg [3:0] a;
   reg [3:0] b;
   wire [3:0] out;
   wire cout;

   ripplecarryAdder rca(.a(a), .b(b), .sum(sum), .cout(cout));

   initial begin
      a = 4'b0000; b = 4'b0000; #10
      a = 4'b0000; b = 4'b0001; #10
      a = 4'b0000; b = 4'b0010; #10
      a = 4'b0000; b = 4'b0011; #10
      a = 4'b0000; b = 4'b0100; #10
      a = 4'b0000; b = 4'b0101; #10
      a = 4'b0000; b = 4'b0110; #10
      a = 4'b0000; b = 4'b0111; #10
      a = 4'b0000; b = 4'b1000; #10
      a = 4'b0000; b = 4'b1001; #10
      a = 4'b0000; b = 4'b1010; #10
      a = 4'b0000; b = 4'b1011; #10
      a = 4'b0000; b = 4'b1100; #10
      a = 4'b0000; b = 4'b1101; #10
      a = 4'b0000; b = 4'b1110; #10
      a = 4'b0000; b = 4'b1111; #10
      a = 4'b0001; b = 4'b0000; #10
      a = 4'b0001; b = 4'b0001; #10
      a = 4'b0001; b = 4'b0010; #10
      a = 4'b0001; b = 4'b0011; #10
      a = 4'b0001; b = 4'b0100; #10
      a = 4'b0001; b = 4'b0101; #10
      a = 4'b0001; b = 4'b0110; #10
      a = 4'b0001; b = 4'b0111; #10
      a = 4'b0001; b = 4'b1000; #10
      a = 4'b0001; b = 4'b1001; #10
      a = 4'b0001; b = 4'b1010; #10
      a = 4'b0001; b = 4'b1011; #10
      a = 4'b0001; b = 4'b1100; #10
      a = 4'b0001; b = 4'b1101; #10
      a = 4'b0001; b = 4'b1110; #10
      a = 4'b0001; b = 4'b1111; #10
      a = 4'b0010; b = 4'b0000; #10
      a = 4'b0010; b = 4'b0001; #10
      a = 4'b0010; b = 4'b0010; #10
      a = 4'b0010; b = 4'b0011; #10
      a = 4'b0010; b = 4'b0100; #10
```



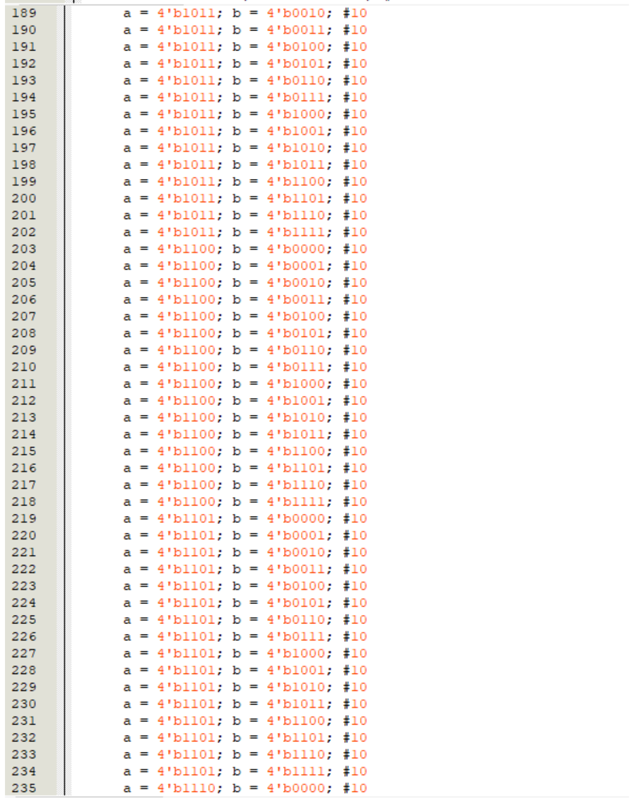
```
        a = 4'b0010; b = 4'b0101; #10
        a = 4'b0010; b = 4'b0110; #10
        a = 4'b0010; b = 4'b0111; #10
        a = 4'b0010; b = 4'b1000; #10
        a = 4'b0010; b = 4'b1001; #10
        a = 4'b0010; b = 4'b1010; #10
        a = 4'b0010; b = 4'b1011; #10
        a = 4'b0010; b = 4'b1100; #10
        a = 4'b0010; b = 4'b1101; #10
        a = 4'b0010; b = 4'b1110; #10
        a = 4'b0010; b = 4'b1111; #10
        a = 4'b0011; b = 4'b0000; #10
        a = 4'b0011; b = 4'b0001; #10
        a = 4'b0011; b = 4'b0010; #10
        a = 4'b0011; b = 4'b0011; #10
        a = 4'b0011; b = 4'b0100; #10
        a = 4'b0011; b = 4'b0101; #10
        a = 4'b0011; b = 4'b0110; #10
        a = 4'b0011; b = 4'b0111; #10
        a = 4'b0011; b = 4'b1000; #10
        a = 4'b0011; b = 4'b1001; #10
        a = 4'b0011; b = 4'b1010; #10
        a = 4'b0011; b = 4'b1011; #10
        a = 4'b0011; b = 4'b1100; #10
        a = 4'b0011; b = 4'b1101; #10
        a = 4'b0011; b = 4'b1110; #10
        a = 4'b0011; b = 4'b1111; #10
        a = 4'b0100; b = 4'b0000; #10
        a = 4'b0100; b = 4'b0001; #10
        a = 4'b0100; b = 4'b0010; #10
        a = 4'b0100; b = 4'b0011; #10
        a = 4'b0100; b = 4'b0100; #10
        a = 4'b0100; b = 4'b0101; #10
        a = 4'b0100; b = 4'b0110; #10
        a = 4'b0100; b = 4'b0111; #10
        a = 4'b0100; b = 4'b1000; #10
        a = 4'b0100; b = 4'b1001; #10
        a = 4'b0100; b = 4'b1010; #10
        a = 4'b0100; b = 4'b1011; #10
        a = 4'b0100; b = 4'b1100; #10
        a = 4'b0100; b = 4'b1101; #10
        a = 4'b0100; b = 4'b1110; #10
        a = 4'b0100; b = 4'b1111; #10
        a = 4'b0101; b = 4'b0000; #10
        a = 4'b0101; b = 4'b0001; #10
        a = 4'b0101; b = 4'b0010; #10
        a = 4'b0101; b = 4'b0011; #10
```



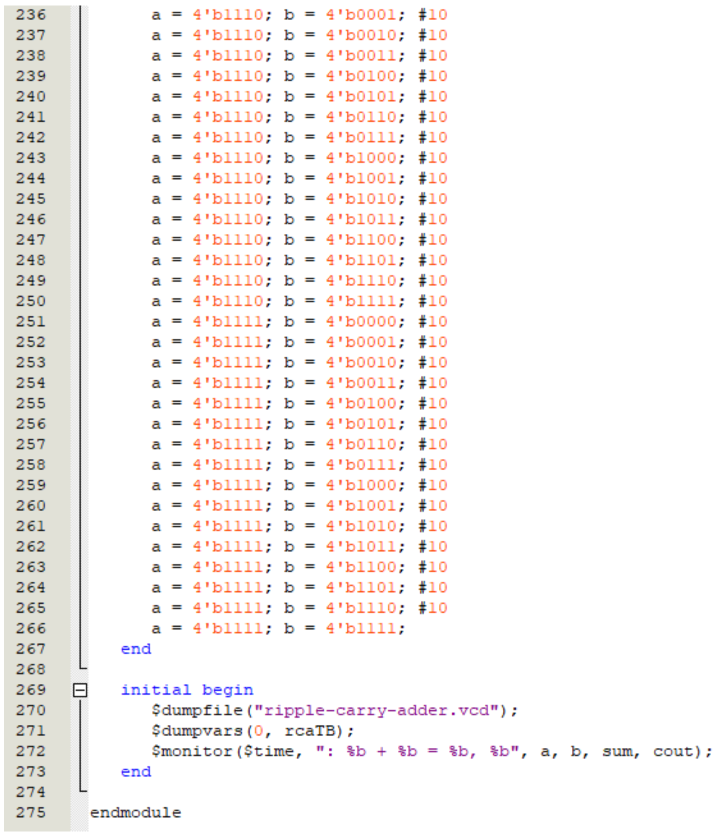
```
        a = 4'b0101; b = 4'b0100; #10
        a = 4'b0101; b = 4'b0101; #10
        a = 4'b0101; b = 4'b0110; #10
        a = 4'b0101; b = 4'b0111; #10
        a = 4'b0101; b = 4'b1000; #10
        a = 4'b0101; b = 4'b1001; #10
        a = 4'b0101; b = 4'b1010; #10
        a = 4'b0101; b = 4'b1011; #10
        a = 4'b0101; b = 4'b1100; #10
        a = 4'b0101; b = 4'b1101; #10
        a = 4'b0101; b = 4'b1110; #10
        a = 4'b0101; b = 4'b1111; #10
        a = 4'b0110; b = 4'b0000; #10
        a = 4'b0110; b = 4'b0001; #10
        a = 4'b0110; b = 4'b0010; #10
        a = 4'b0110; b = 4'b0011; #10
        a = 4'b0110; b = 4'b0100; #10
        a = 4'b0110; b = 4'b0101; #10
        a = 4'b0110; b = 4'b0110; #10
        a = 4'b0110; b = 4'b0111; #10
        a = 4'b0110; b = 4'b1000; #10
        a = 4'b0110; b = 4'b1001; #10
        a = 4'b0110; b = 4'b1010; #10
        a = 4'b0110; b = 4'b1011; #10
        a = 4'b0110; b = 4'b1100; #10
        a = 4'b0110; b = 4'b1101; #10
        a = 4'b0110; b = 4'b1110; #10
        a = 4'b0110; b = 4'b1111; #10
        a = 4'b0111; b = 4'b0000; #10
        a = 4'b0111; b = 4'b0001; #10
        a = 4'b0111; b = 4'b0010; #10
        a = 4'b0111; b = 4'b0011; #10
        a = 4'b0111; b = 4'b0100; #10
        a = 4'b0111; b = 4'b0101; #10
        a = 4'b0111; b = 4'b0110; #10
        a = 4'b0111; b = 4'b0111; #10
        a = 4'b0111; b = 4'b1000; #10
        a = 4'b0111; b = 4'b1001; #10
        a = 4'b0111; b = 4'b1010; #10
        a = 4'b0111; b = 4'b1011; #10
        a = 4'b0111; b = 4'b1100; #10
        a = 4'b0111; b = 4'b1101; #10
        a = 4'b0111; b = 4'b1110; #10
        a = 4'b0111; b = 4'b1111; #10
        a = 4'b1000; b = 4'b0000; #10
        a = 4'b1000; b = 4'b0001; #10
        a = 4'b1000; b = 4'b0010; #10
```



```
        a = 4'b1000; b = 4'b0011; #10
        a = 4'b1000; b = 4'b0100; #10
        a = 4'b1000; b = 4'b0101; #10
        a = 4'b1000; b = 4'b0110; #10
        a = 4'b1000; b = 4'b0111; #10
        a = 4'b1000; b = 4'b1000; #10
        a = 4'b1000; b = 4'b1001; #10
        a = 4'b1000; b = 4'b1010; #10
        a = 4'b1000; b = 4'b1011; #10
        a = 4'b1000; b = 4'b1100; #10
        a = 4'b1000; b = 4'b1101; #10
        a = 4'b1000; b = 4'b1110; #10
        a = 4'b1000; b = 4'b1111; #10
        a = 4'b1001; b = 4'b0000; #10
        a = 4'b1001; b = 4'b0001; #10
        a = 4'b1001; b = 4'b0010; #10
        a = 4'b1001; b = 4'b0011; #10
        a = 4'b1001; b = 4'b0100; #10
        a = 4'b1001; b = 4'b0101; #10
        a = 4'b1001; b = 4'b0110; #10
        a = 4'b1001; b = 4'b0111; #10
        a = 4'b1001; b = 4'b1000; #10
        a = 4'b1001; b = 4'b1001; #10
        a = 4'b1001; b = 4'b1010; #10
        a = 4'b1001; b = 4'b1011; #10
        a = 4'b1001; b = 4'b1100; #10
        a = 4'b1001; b = 4'b1101; #10
        a = 4'b1001; b = 4'b1110; #10
        a = 4'b1001; b = 4'b1111; #10
        a = 4'b1010; b = 4'b0000; #10
        a = 4'b1010; b = 4'b0001; #10
        a = 4'b1010; b = 4'b0010; #10
        a = 4'b1010; b = 4'b0011; #10
        a = 4'b1010; b = 4'b0100; #10
        a = 4'b1010; b = 4'b0101; #10
        a = 4'b1010; b = 4'b0110; #10
        a = 4'b1010; b = 4'b0111; #10
        a = 4'b1010; b = 4'b1000; #10
        a = 4'b1010; b = 4'b1001; #10
        a = 4'b1010; b = 4'b1010; #10
        a = 4'b1010; b = 4'b1011; #10
        a = 4'b1010; b = 4'b1100; #10
        a = 4'b1010; b = 4'b1101; #10
        a = 4'b1010; b = 4'b1110; #10
        a = 4'b1010; b = 4'b1111; #10
        a = 4'b1011; b = 4'b0000; #10
        a = 4'b1011; b = 4'b0001; #10
```

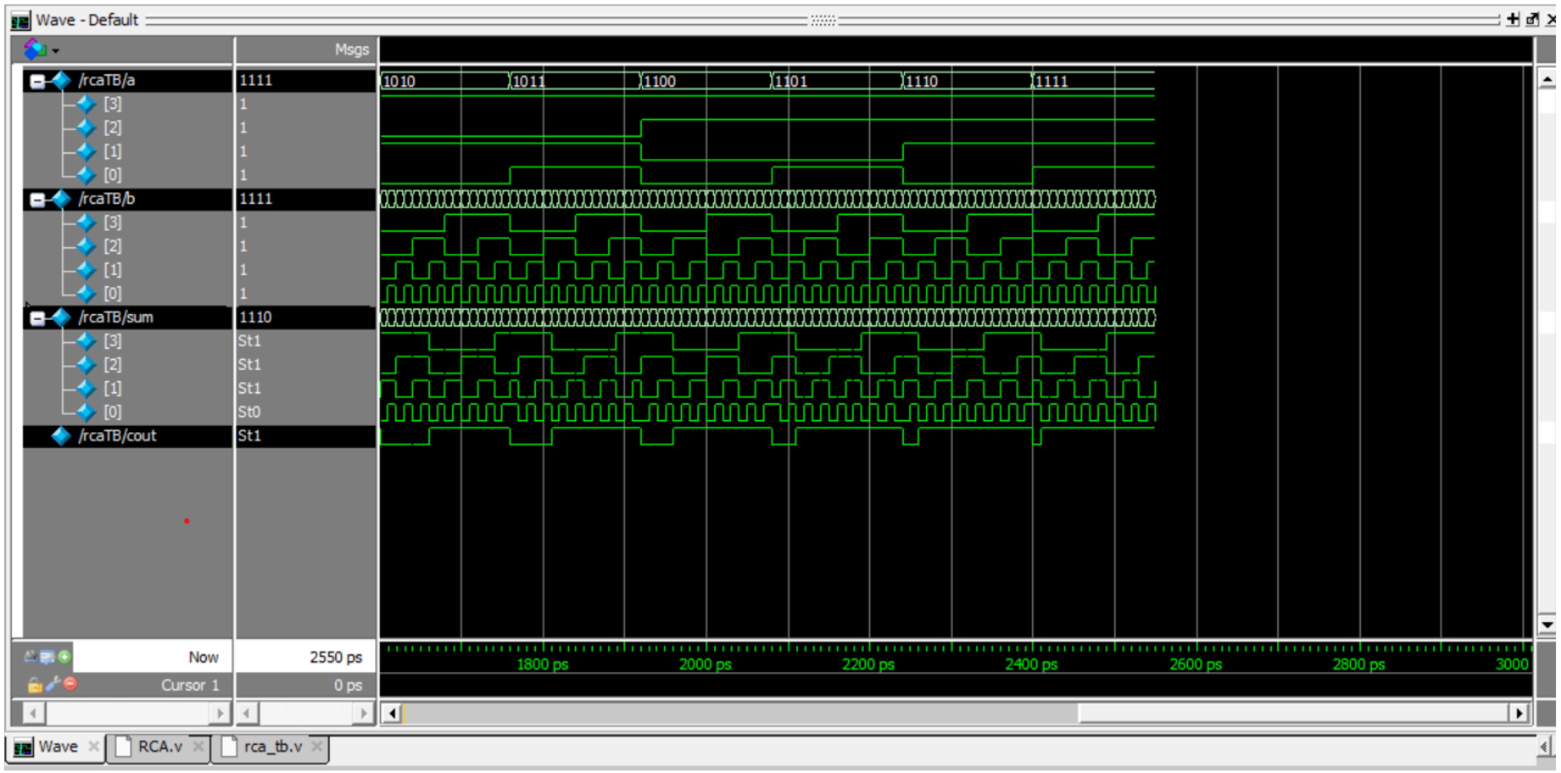


```
        a = 4'b1011; b = 4'b0010; #10
        a = 4'b1011; b = 4'b0011; #10
        a = 4'b1011; b = 4'b0100; #10
        a = 4'b1011; b = 4'b0101; #10
        a = 4'b1011; b = 4'b0110; #10
        a = 4'b1011; b = 4'b0111; #10
        a = 4'b1011; b = 4'b1000; #10
        a = 4'b1011; b = 4'b1001; #10
        a = 4'b1011; b = 4'b1010; #10
        a = 4'b1011; b = 4'b1011; #10
        a = 4'b1011; b = 4'b1100; #10
        a = 4'b1011; b = 4'b1101; #10
        a = 4'b1011; b = 4'b1110; #10
        a = 4'b1011; b = 4'b1111; #10
        a = 4'b1100; b = 4'b0000; #10
        a = 4'b1100; b = 4'b0001; #10
        a = 4'b1100; b = 4'b0010; #10
        a = 4'b1100; b = 4'b0011; #10
        a = 4'b1100; b = 4'b0100; #10
        a = 4'b1100; b = 4'b0101; #10
        a = 4'b1100; b = 4'b0110; #10
        a = 4'b1100; b = 4'b0111; #10
        a = 4'b1100; b = 4'b1000; #10
        a = 4'b1100; b = 4'b1001; #10
        a = 4'b1100; b = 4'b1010; #10
        a = 4'b1100; b = 4'b1011; #10
        a = 4'b1100; b = 4'b1100; #10
        a = 4'b1100; b = 4'b1101; #10
        a = 4'b1100; b = 4'b1110; #10
        a = 4'b1100; b = 4'b1111; #10
        a = 4'b1101; b = 4'b0000; #10
        a = 4'b1101; b = 4'b0001; #10
        a = 4'b1101; b = 4'b0010; #10
        a = 4'b1101; b = 4'b0011; #10
        a = 4'b1101; b = 4'b0100; #10
        a = 4'b1101; b = 4'b0101; #10
        a = 4'b1101; b = 4'b0110; #10
        a = 4'b1101; b = 4'b0111; #10
        a = 4'b1101; b = 4'b1000; #10
        a = 4'b1101; b = 4'b1001; #10
        a = 4'b1101; b = 4'b1010; #10
        a = 4'b1101; b = 4'b1011; #10
        a = 4'b1101; b = 4'b1100; #10
        a = 4'b1101; b = 4'b1101; #10
        a = 4'b1101; b = 4'b1110; #10
        a = 4'b1101; b = 4'b1111; #10
        a = 4'b1110; b = 4'b0000; #10
```



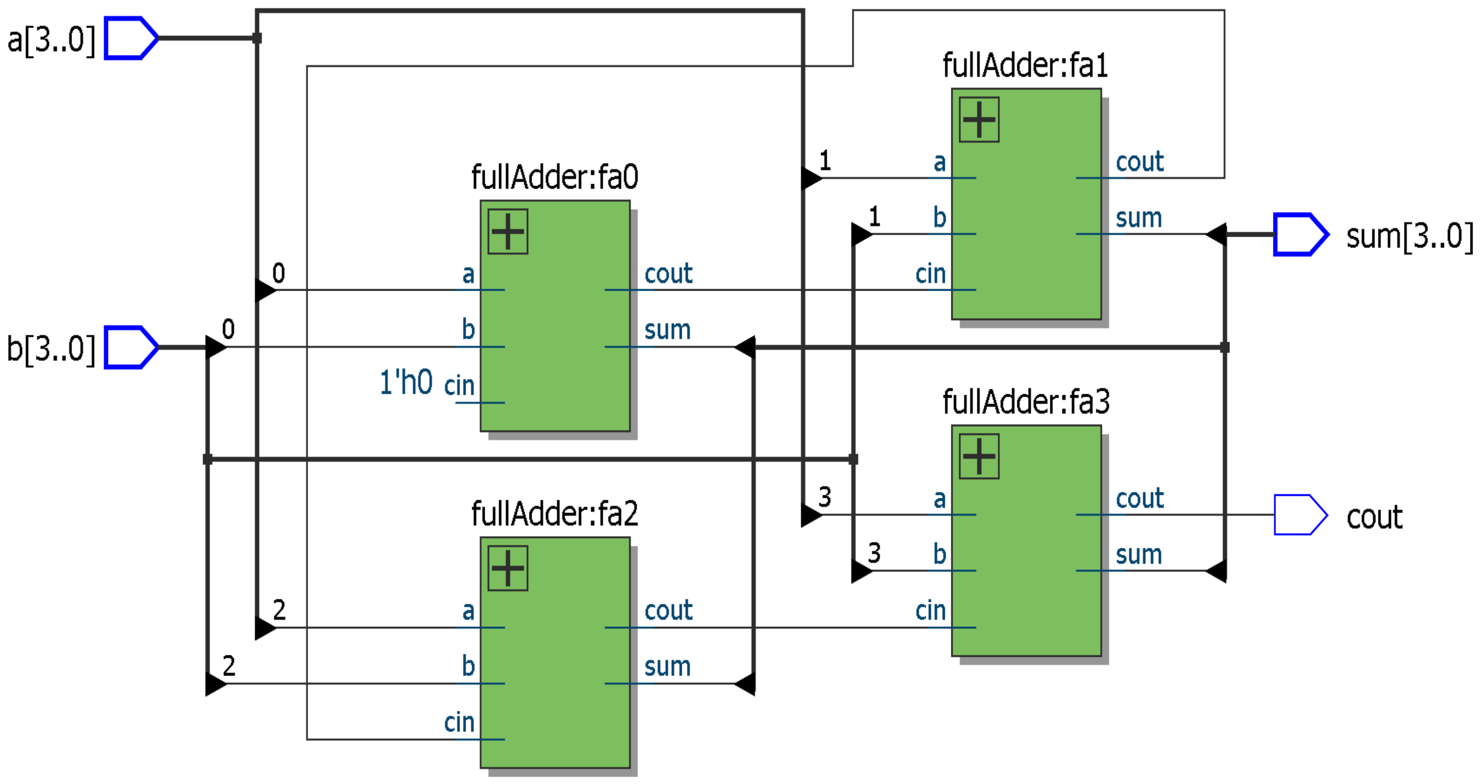
```
      a = 4'b1110; b = 4'b0001; #10
      a = 4'b1110; b = 4'b0010; #10
      a = 4'b1110; b = 4'b0011; #10
      a = 4'b1110; b = 4'b0100; #10
      a = 4'b1110; b = 4'b0101; #10
      a = 4'b1110; b = 4'b0110; #10
      a = 4'b1110; b = 4'b0111; #10
      a = 4'b1110; b = 4'b1000; #10
      a = 4'b1110; b = 4'b1001; #10
      a = 4'b1110; b = 4'b1010; #10
      a = 4'b1110; b = 4'b1011; #10
      a = 4'b1110; b = 4'b1100; #10
      a = 4'b1110; b = 4'b1101; #10
      a = 4'b1110; b = 4'b1110; #10
      a = 4'b1110; b = 4'b1111; #10
      a = 4'b1111; b = 4'b0000; #10
      a = 4'b1111; b = 4'b0001; #10
      a = 4'b1111; b = 4'b0010; #10
      a = 4'b1111; b = 4'b0011; #10
      a = 4'b1111; b = 4'b0100; #10
      a = 4'b1111; b = 4'b0101; #10
      a = 4'b1111; b = 4'b0110; #10
      a = 4'b1111; b = 4'b0111; #10
      a = 4'b1111; b = 4'b1000; #10
      a = 4'b1111; b = 4'b1001; #10
      a = 4'b1111; b = 4'b1010; #10
      a = 4'b1111; b = 4'b1011; #10
      a = 4'b1111; b = 4'b1100; #10
      a = 4'b1111; b = 4'b1101; #10
      a = 4'b1111; b = 4'b1110; #10
      a = 4'b1111; b = 4'b1111;
   end

   initial begin
      $dumpfile("ripple-carry-adder.vcd");
      $dumpvars(0, rcaTB);
      $monitor($time, ": %b + %b = %b, %b", a, b, sum, cout);
   end

endmodule
```

Experiment 2 - RCA Waveform:

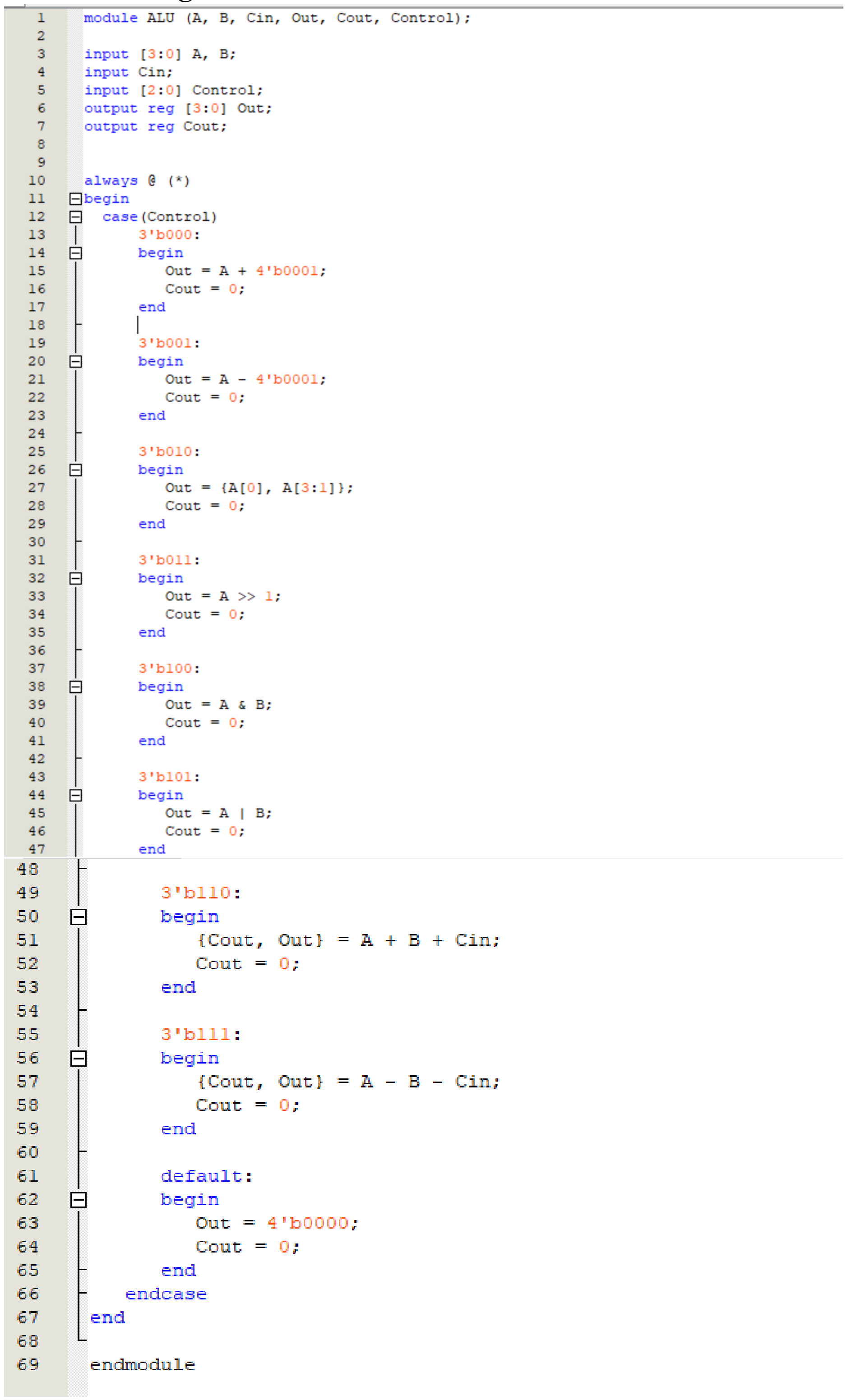


```
/rcaTB/a      1111
  [3]         1
  [2]         1
  [1]         1
  [0]         1
/rcaTB/b      1111
  [3]         1
  [2]         1
  [1]         1
  [0]         1
/rcaTB/sum    1110
  [3]         St1
  [2]         St1
  [1]         St1
  [0]         St0
/rcaTB/cout   St1

Now           2550 ps
Cursor 1      0 ps
```

Experiment 2 - RCA Schematic:

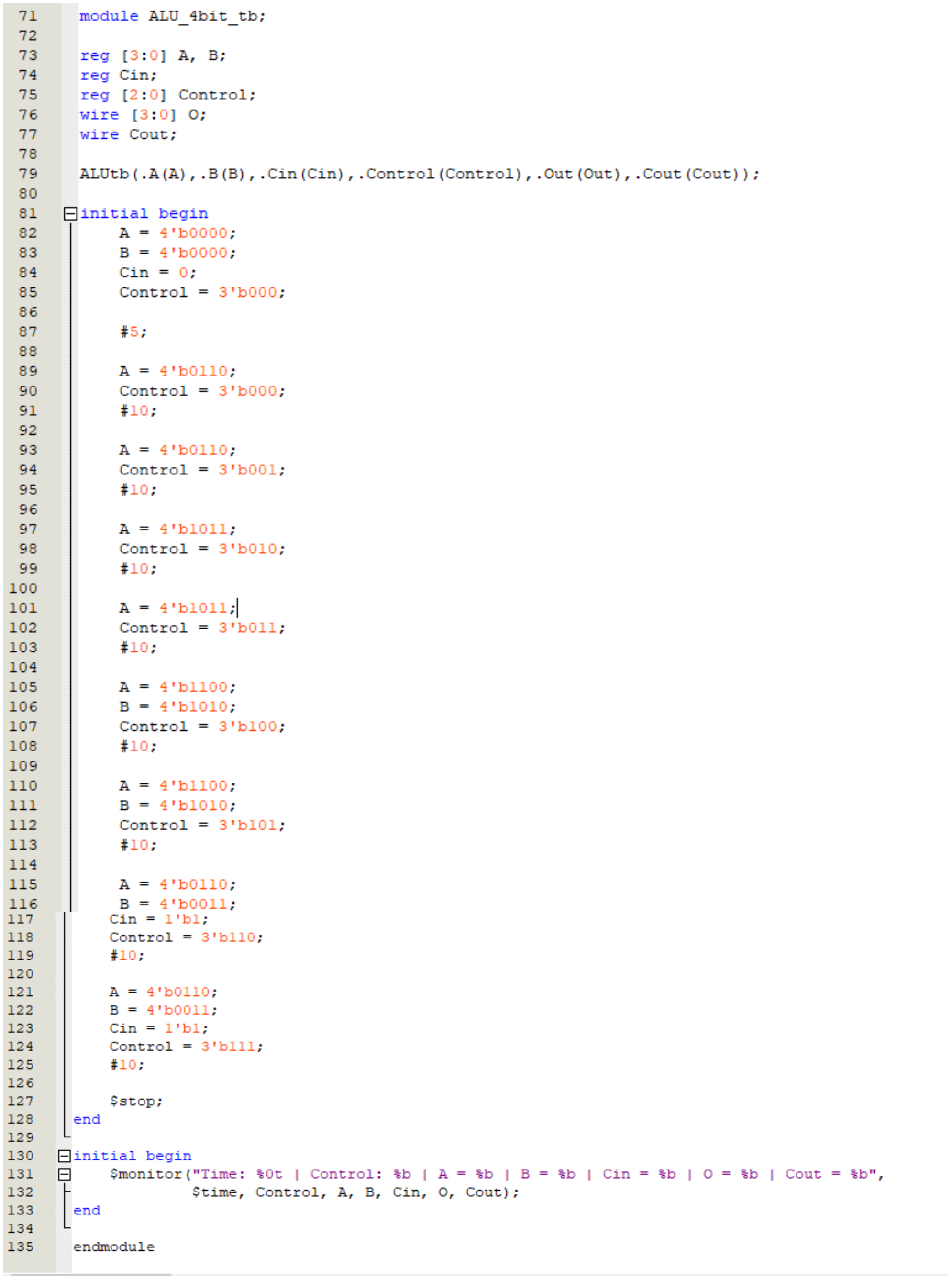


Experiment 3 - ALU Logic Code:



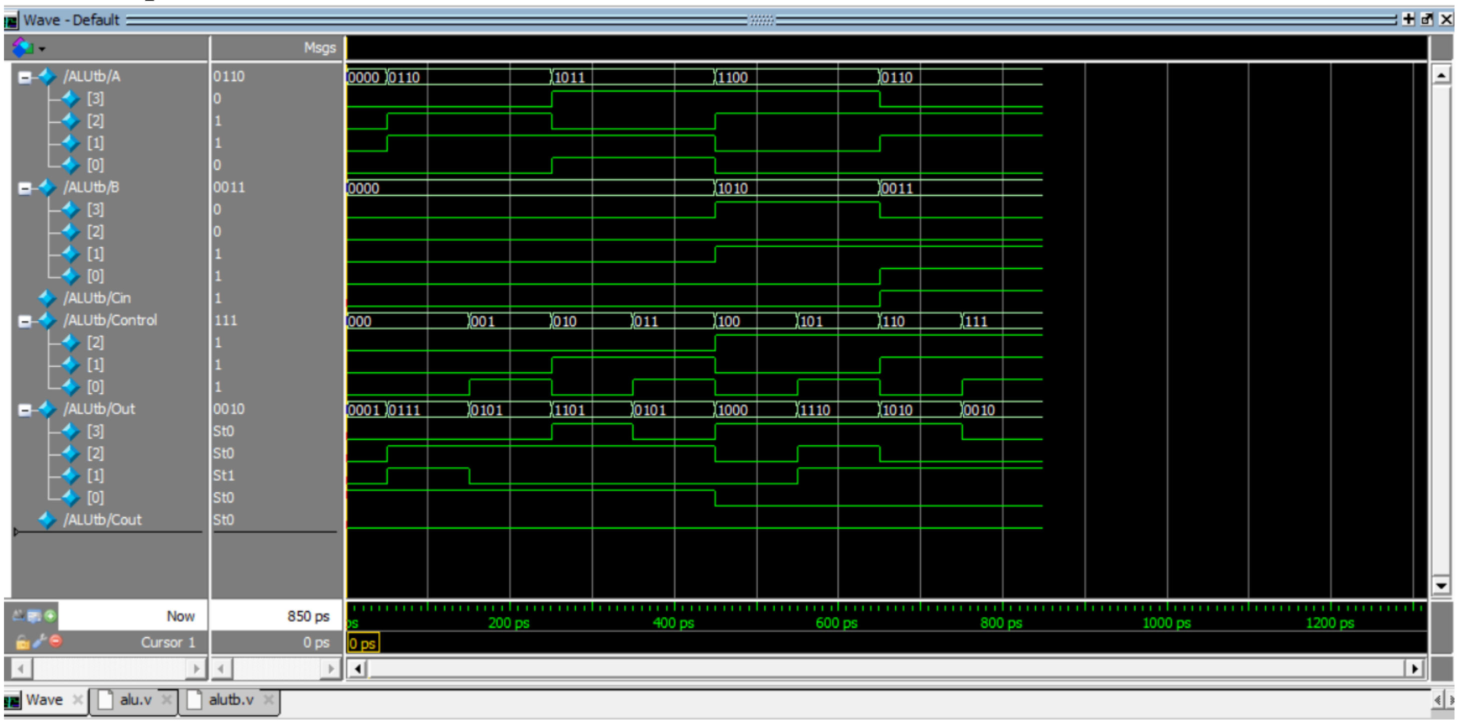
```
module ALU (A, B, Cin, Out, Cout, Control);

input [3:0] A, B;
input Cin;
input [2:0] Control;
output reg [3:0] Out;
output reg Cout;


always @ (*)
begin
  case(Control)
      3'b000:
      begin
         Out = A + 4'b0001;
         Cout = 0;
      end

      3'b001:
      begin
         Out = A - 4'b0001;
         Cout = 0;
      end

      3'b010:
      begin
         Out = {A[0], A[3:1]};
         Cout = 0;
      end

      3'b011:
      begin
         Out = A >> 1;
         Cout = 0;
      end

      3'b100:
      begin
         Out = A & B;
         Cout = 0;
      end

      3'b101:
      begin
         Out = A | B;
         Cout = 0;
      end

      3'b110:
      begin
         {Cout, Out} = A + B + Cin;
         Cout = 0;
      end

      3'b111:
      begin
         {Cout, Out} = A - B - Cin;
         Cout = 0;
      end

      default:
      begin
         Out = 4'b0000;
         Cout = 0;
      end
  endcase
end

endmodule
```

Experiment 3 - ALU Testbench Code:

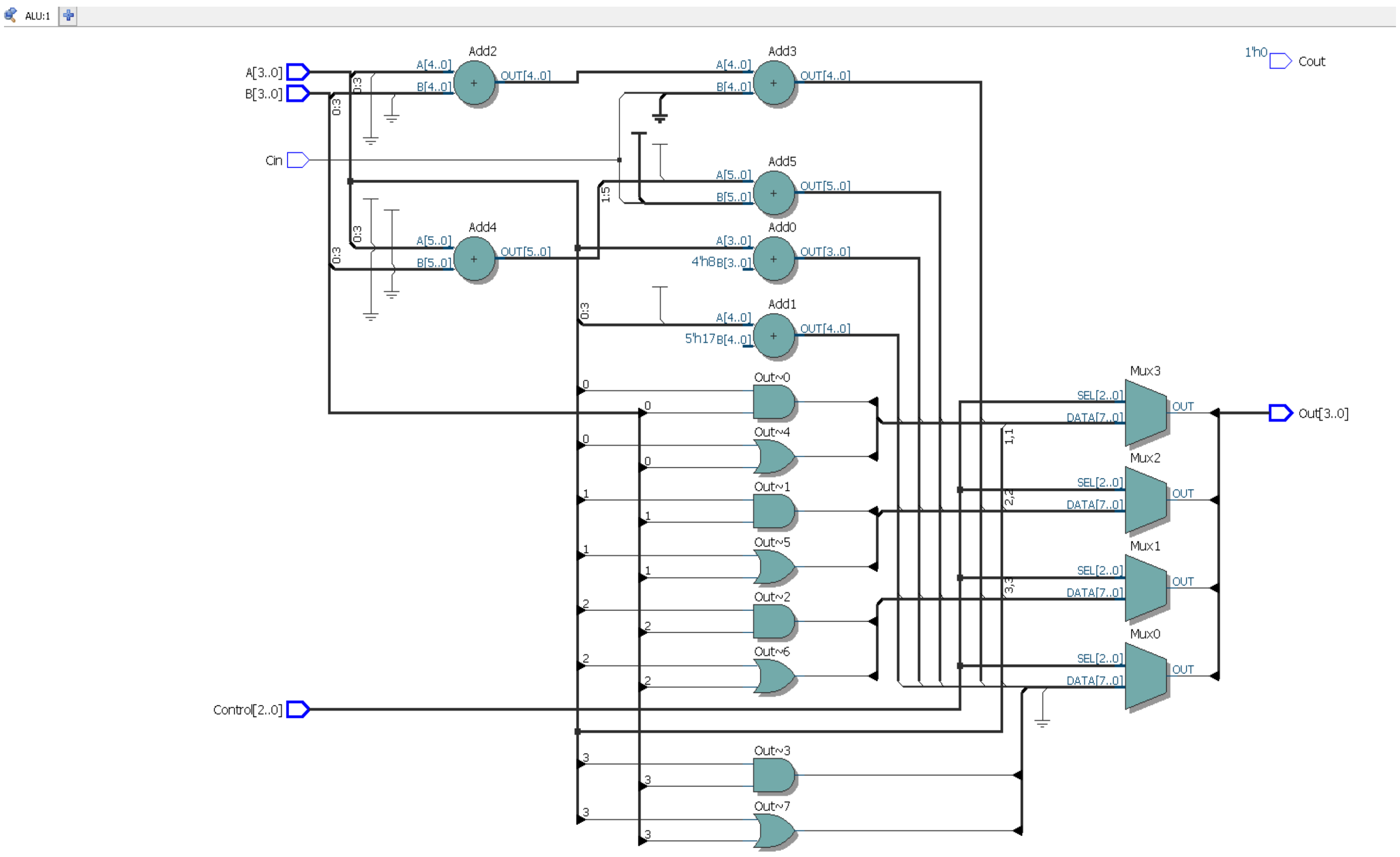


```verilog
module ALU_4bit_tb;

reg [3:0] A, B;
reg Cin;
reg [2:0] Control;
wire [3:0] O;
wire Cout;

ALUtb(.A(A),.B(B),.Cin(Cin),.Control(Control),.Out(Out),.Cout(Cout));

initial begin
    A = 4'b0000;
    B = 4'b0000;
    Cin = 0;
    Control = 3'b000;

    #5;

    A = 4'b0110;
    Control = 3'b000;
    #10;

    A = 4'b0110;
    Control = 3'b001;
    #10;

    A = 4'b1011;
    Control = 3'b010;
    #10;

    A = 4'b1011;
    Control = 3'b011;
    #10;

    A = 4'b1100;
    B = 4'b1010;
    Control = 3'b100;
    #10;

    A = 4'b1100;
    B = 4'b1010;
    Control = 3'b101;
    #10;

    A = 4'b0110;
    B = 4'b0011;
    Cin = 1'b1;
    Control = 3'b110;
    #10;

    A = 4'b0110;
    B = 4'b0011;
    Cin = 1'b1;
    Control = 3'b111;
    #10;

    $stop;
end

initial begin
    $monitor("Time: %0t | Control: %b | A = %b | B = %b | Cin = %b | O = %b | Cout = %b",
             $time, Control, A, B, Cin, O, Cout);
end

endmodule
```

Experiment 3 - ALU Waveform:



Experiment 3 - ALU Schematic:



3. Questions and Answers

a.) Register Transfer Level: A register transfer level is a digital logic design abstraction that shows how data flows and operations are performed between registers. In Quartus, the RTL viewer allowed us to generate schematics for our experiments to ensure they were accurate.

b.) HDL V.S. C: Hardware description languages (HDLs) are primarily used to create logic for implementation on hardware devices such as fully programmable gate arrays. They are used to write the behavior of digital logic circuits. General-purpose languages such as C and C++ are high-level programming languages that are primarily used to create software to be run on a processor.

c.) Verilog Modules: In Verilog, a module is essentially a building block for hardware implementation. Each module has its own unique functionality and can be as simple or as complex as the user wants it to be. Modules work together and you can even instantiate modules within one another to accomplish more complex tasks.

4. Conclusions

Through this lab, we were able to refresh and refine our ability to write Verilog code. Each lab experiment worked on a different digital logic component. We used Modelsim and Quartus to write, compile, synthesize, and ensure our modules correctly modeled the expected behavior of these components.