

UNIVERSITY OF NEVADA LAS VEGAS, DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING LABORATORIES.

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

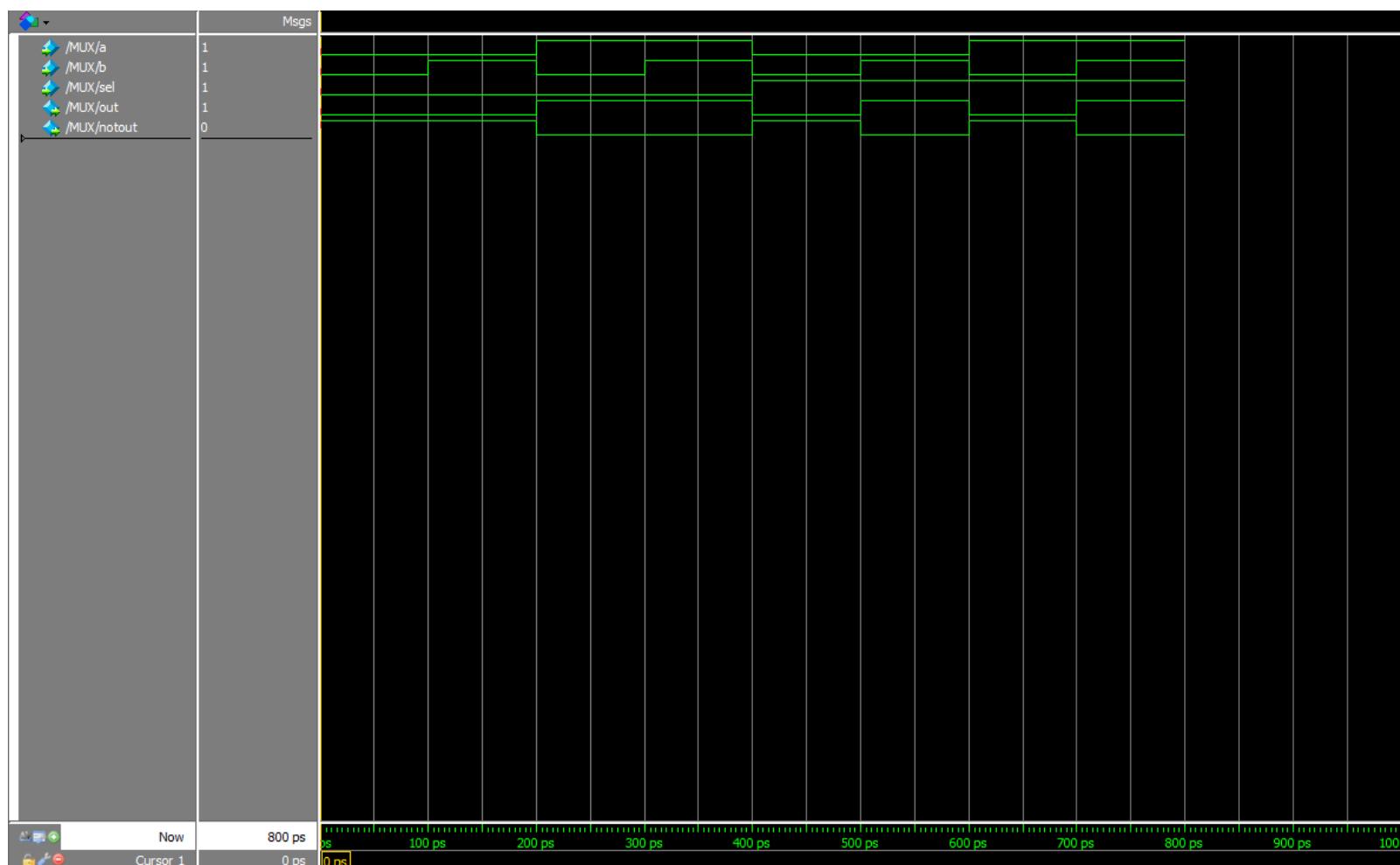
A hardware description language is a lower-level computer language that is used primarily in direct accordance and for implementation on hardware devices such as fully programmable gate arrays and integrated circuits. Two common HDLs are VHDL and Verilog.

2. Description of Experiments

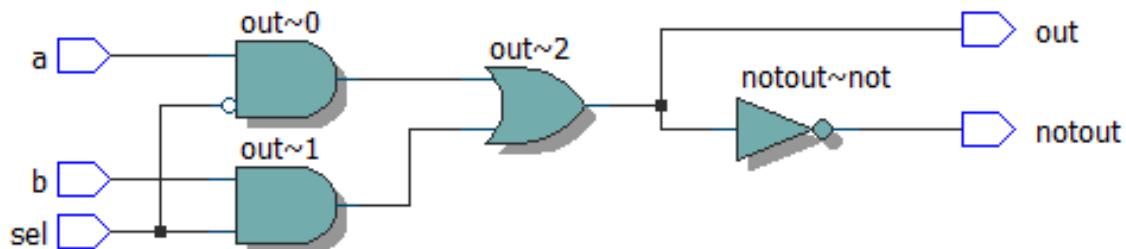
Experiment 1A - Continuous Assignments Code:

```
C:/altera/13.0sp1/mux21.v (/MUX) - Default
Ln# | 
1   module MUX (a, b, out, notout, sel);
2     input a, b, sel;
3     output out, notout;
4     assign out = (~sel & a) | (sel & b);
5     assign notout = ~out;
6   endmodule
```

Experiment 1A - Continuous Assignments Waveforms:



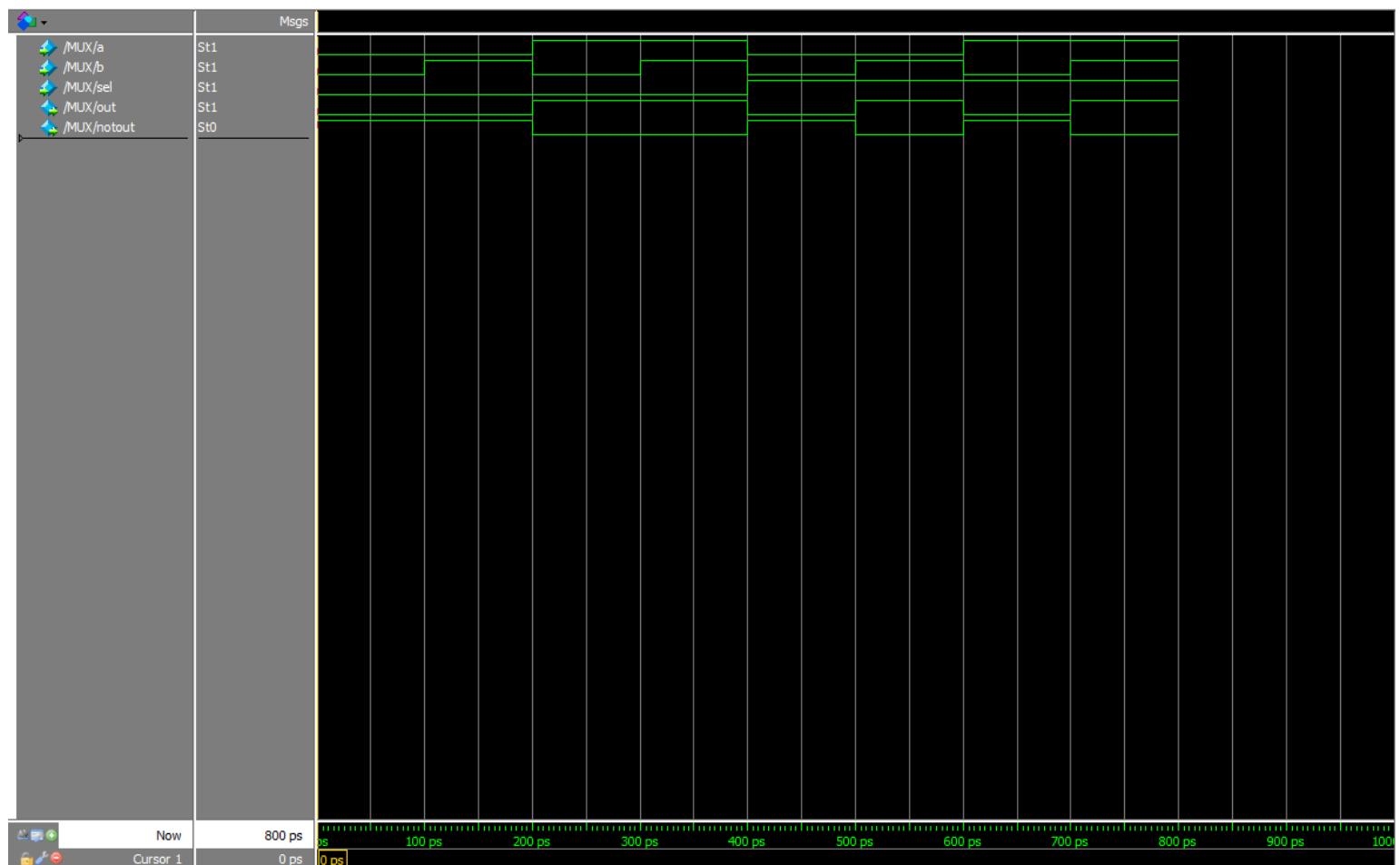
Experiment 1A - Continuous Assignments Schematic:



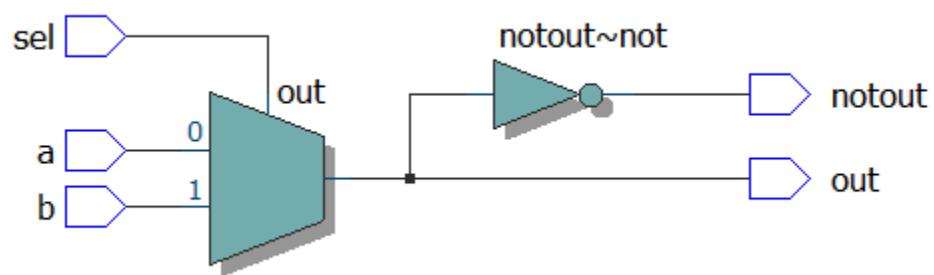
Experiment 1B - Conditional Operator Code:

```
1 module MUX (a, b, out, notout, sel);
2   input a, b, sel;
3   output out, notout;
4   assign out = sel?b:a;
5   assign notout = ~out;
6 endmodule
```

Experiment 1B - Conditional Operator Waveforms:



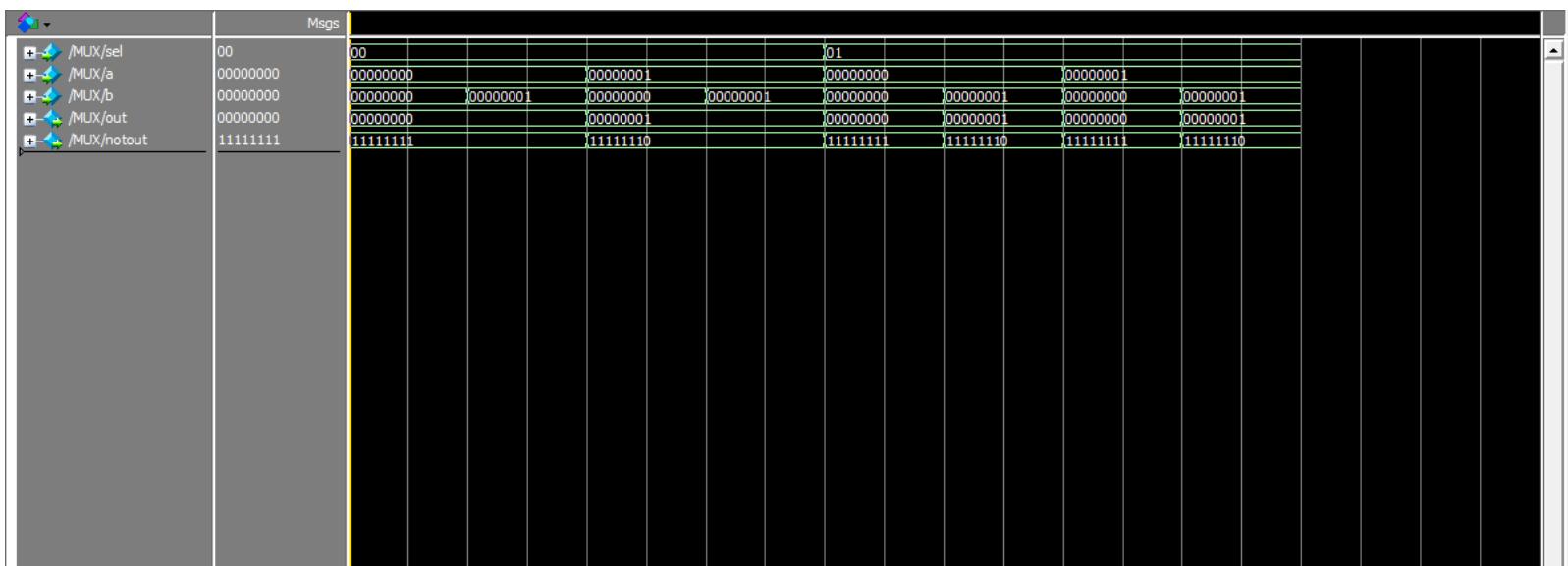
Experiment 1B - Conditional Operator Schematic:



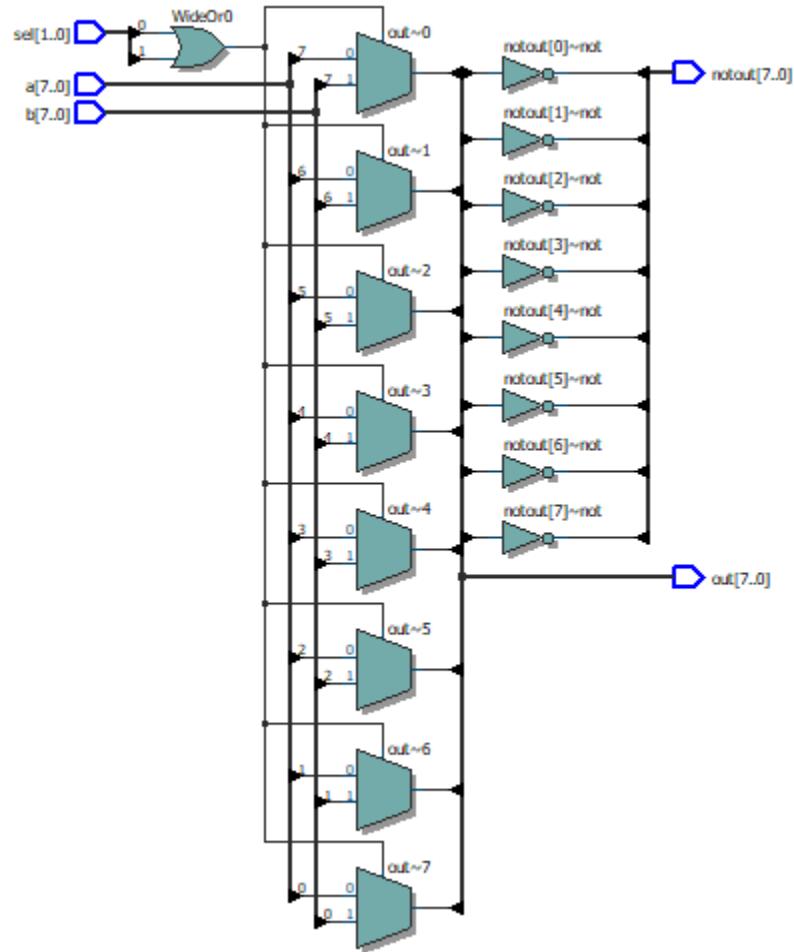
Experiment 1C - 8-bit Code:

```
1  module MUX (a, b, sel,
2    out, notout);
3
4    input [1:0] sel;
5    input [7:0] a, b;
6    output [7:0] out, notout;
7
8
9
10   assign out = sel?b:a;
11   assign notout = ~out;
12 endmodule
```

Experiment 1C - 8-bit Waveforms:



Experiment 1C - 8-bit Schematic:



Experiment 2 - RCA Logic Code:

```
1  module fullAdder(a, b, cin, sum, cout);
2      input a, b, cin;
3      output sum, cout;
4
5      assign sum = a ^ b ^ cin;
6      assign cout = ((a ^ b) & cin) | (a & b);
7  endmodule
8
9  module ripplecarryAdder(a, b, sum, cout);
10     input [3:0] a;
11     input [3:0] b;
12     output [3:0] sum;
13     output cout;
14
15     wire c1, c2, c3;
16     fullAdder fa0(a[0], b[0], 0, sum[0], c1);
17     fullAdder fa1(a[1], b[1], c1, sum[1], c2);
18     fullAdder fa2(a[2], b[2], c2, sum[2], c3);
19     fullAdder fa3(a[3], b[3], c3, sum[3], cout);
20
21 endmodule
```

Experiment 2 - RCA Testbench Code:

```
1  module rcaTB;
2
3  reg [3:0] a;
4  reg [3:0] b;
5  wire [3:0] out;
6  wire cout;
7
8  ripplecarryAdder rca(.a(a), .b(b), .sum(sum), .cout(cout));
9
10 initial begin
11     a = 4'b0000; b = 4'b0000; #10
12     a = 4'b0000; b = 4'b0001; #10
13     a = 4'b0000; b = 4'b0010; #10
14     a = 4'b0000; b = 4'b0011; #10
15     a = 4'b0000; b = 4'b0100; #10
16     a = 4'b0000; b = 4'b0101; #10
17     a = 4'b0000; b = 4'b0110; #10
18     a = 4'b0000; b = 4'b0111; #10
19     a = 4'b0000; b = 4'b1000; #10
20     a = 4'b0000; b = 4'b1001; #10
21     a = 4'b0000; b = 4'b1010; #10
22     a = 4'b0000; b = 4'b1011; #10
23     a = 4'b0000; b = 4'b1100; #10
24     a = 4'b0000; b = 4'b1101; #10
25     a = 4'b0000; b = 4'b1110; #10
26     a = 4'b0000; b = 4'b1111; #10
27     a = 4'b0001; b = 4'b0000; #10
28     a = 4'b0001; b = 4'b0001; #10
29     a = 4'b0001; b = 4'b0010; #10
30     a = 4'b0001; b = 4'b0011; #10
31     a = 4'b0001; b = 4'b0100; #10
32     a = 4'b0001; b = 4'b0101; #10
33     a = 4'b0001; b = 4'b0110; #10
34     a = 4'b0001; b = 4'b0111; #10
35     a = 4'b0001; b = 4'b1000; #10
36     a = 4'b0001; b = 4'b1001; #10
37     a = 4'b0001; b = 4'b1010; #10
38     a = 4'b0001; b = 4'b1011; #10
39     a = 4'b0001; b = 4'b1100; #10
40     a = 4'b0001; b = 4'b1101; #10
41     a = 4'b0001; b = 4'b1110; #10
42     a = 4'b0001; b = 4'b1111; #10
43     a = 4'b0010; b = 4'b0000; #10
44     a = 4'b0010; b = 4'b0001; #10
45     a = 4'b0010; b = 4'b0010; #10
46     a = 4'b0010; b = 4'b0011; #10
47     a = 4'b0010; b = 4'b0100; #10
```

```

48      a = 4'b0010; b = 4'b0101; #10
49      a = 4'b0010; b = 4'b0110; #10
50      a = 4'b0010; b = 4'b0111; #10
51      a = 4'b0010; b = 4'b1000; #10
52      a = 4'b0010; b = 4'b1001; #10
53      a = 4'b0010; b = 4'b1010; #10
54      a = 4'b0010; b = 4'b1011; #10
55      a = 4'b0010; b = 4'b1100; #10
56      a = 4'b0010; b = 4'b1101; #10
57      a = 4'b0010; b = 4'b1110; #10
58      a = 4'b0010; b = 4'b1111; #10
59      a = 4'b0011; b = 4'b0000; #10
60      a = 4'b0011; b = 4'b0001; #10
61      a = 4'b0011; b = 4'b0010; #10
62      a = 4'b0011; b = 4'b0011; #10
63      a = 4'b0011; b = 4'b0100; #10
64      a = 4'b0011; b = 4'b0101; #10
65      a = 4'b0011; b = 4'b0110; #10
66      a = 4'b0011; b = 4'b0111; #10
67      a = 4'b0011; b = 4'b1000; #10
68      a = 4'b0011; b = 4'b1001; #10
69      a = 4'b0011; b = 4'b1010; #10
70      a = 4'b0011; b = 4'b1011; #10
71      a = 4'b0011; b = 4'b1100; #10
72      a = 4'b0011; b = 4'b1101; #10
73      a = 4'b0011; b = 4'b1110; #10
74      a = 4'b0011; b = 4'b1111; #10
75      a = 4'b0100; b = 4'b0000; #10
76      a = 4'b0100; b = 4'b0001; #10
77      a = 4'b0100; b = 4'b0010; #10
78      a = 4'b0100; b = 4'b0011; #10
79      a = 4'b0100; b = 4'b0100; #10
80      a = 4'b0100; b = 4'b0101; #10
81      a = 4'b0100; b = 4'b0110; #10
82      a = 4'b0100; b = 4'b0111; #10
83      a = 4'b0100; b = 4'b1000; #10
84      a = 4'b0100; b = 4'b1001; #10
85      a = 4'b0100; b = 4'b1010; #10
86      a = 4'b0100; b = 4'b1011; #10
87      a = 4'b0100; b = 4'b1100; #10
88      a = 4'b0100; b = 4'b1101; #10
89      a = 4'b0100; b = 4'b1110; #10
90      a = 4'b0100; b = 4'b1111; #10
91      a = 4'b0101; b = 4'b0000; #10
92      a = 4'b0101; b = 4'b0001; #10
93      a = 4'b0101; b = 4'b0010; #10
94      a = 4'b0101; b = 4'b0011; #10
95      a = 4'b0101; b = 4'b0100; #10
96      a = 4'b0101; b = 4'b0101; #10
97      a = 4'b0101; b = 4'b0110; #10
98      a = 4'b0101; b = 4'b0111; #10
99      a = 4'b0101; b = 4'b1000; #10
100     a = 4'b0101; b = 4'b1001; #10
101     a = 4'b0101; b = 4'b1010; #10
102     a = 4'b0101; b = 4'b1011; #10
103     a = 4'b0101; b = 4'b1100; #10
104     a = 4'b0101; b = 4'b1101; #10
105     a = 4'b0101; b = 4'b1110; #10
106     a = 4'b0101; b = 4'b1111; #10
107     a = 4'b0110; b = 4'b0000; #10
108     a = 4'b0110; b = 4'b0001; #10
109     a = 4'b0110; b = 4'b0010; #10
110     a = 4'b0110; b = 4'b0011; #10
111     a = 4'b0110; b = 4'b0100; #10
112     a = 4'b0110; b = 4'b0101; #10
113     a = 4'b0110; b = 4'b0110; #10
114     a = 4'b0110; b = 4'b0111; #10
115     a = 4'b0110; b = 4'b1000; #10
116     a = 4'b0110; b = 4'b1001; #10
117     a = 4'b0110; b = 4'b1010; #10
118     a = 4'b0110; b = 4'b1011; #10
119     a = 4'b0110; b = 4'b1100; #10
120     a = 4'b0110; b = 4'b1101; #10
121     a = 4'b0110; b = 4'b1110; #10
122     a = 4'b0110; b = 4'b1111; #10
123     a = 4'b0111; b = 4'b0000; #10
124     a = 4'b0111; b = 4'b0001; #10
125     a = 4'b0111; b = 4'b0010; #10
126     a = 4'b0111; b = 4'b0011; #10
127     a = 4'b0111; b = 4'b0100; #10
128     a = 4'b0111; b = 4'b0101; #10
129     a = 4'b0111; b = 4'b0110; #10
130     a = 4'b0111; b = 4'b0111; #10
131     a = 4'b0111; b = 4'b1000; #10
132     a = 4'b0111; b = 4'b1001; #10
133     a = 4'b0111; b = 4'b1010; #10
134     a = 4'b0111; b = 4'b1011; #10
135     a = 4'b0111; b = 4'b1100; #10
136     a = 4'b0111; b = 4'b1101; #10
137     a = 4'b0111; b = 4'b1110; #10
138     a = 4'b0111; b = 4'b1111; #10
139     a = 4'b1000; b = 4'b0000; #10
140     a = 4'b1000; b = 4'b0001; #10
141     a = 4'b1000; b = 4'b0010; #10

```

```

142 a = 4'b1000; b = 4'b0011; #10
143 a = 4'b1000; b = 4'b0100; #10
144 a = 4'b1000; b = 4'b0101; #10
145 a = 4'b1000; b = 4'b0110; #10
146 a = 4'b1000; b = 4'b0111; #10
147 a = 4'b1000; b = 4'b1000; #10
148 a = 4'b1000; b = 4'b1001; #10
149 a = 4'b1000; b = 4'b1010; #10
150 a = 4'b1000; b = 4'b1011; #10
151 a = 4'b1000; b = 4'b1100; #10
152 a = 4'b1000; b = 4'b1101; #10
153 a = 4'b1000; b = 4'b1110; #10
154 a = 4'b1000; b = 4'b1111; #10
155 a = 4'b1001; b = 4'b0000; #10
156 a = 4'b1001; b = 4'b0001; #10
157 a = 4'b1001; b = 4'b0010; #10
158 a = 4'b1001; b = 4'b0011; #10
159 a = 4'b1001; b = 4'b0100; #10
160 a = 4'b1001; b = 4'b0101; #10
161 a = 4'b1001; b = 4'b0110; #10
162 a = 4'b1001; b = 4'b0111; #10
163 a = 4'b1001; b = 4'b1000; #10
164 a = 4'b1001; b = 4'b1001; #10
165 a = 4'b1001; b = 4'b1010; #10
166 a = 4'b1001; b = 4'b1011; #10
167 a = 4'b1001; b = 4'b1100; #10
168 a = 4'b1001; b = 4'b1101; #10
169 a = 4'b1001; b = 4'b1110; #10
170 a = 4'b1001; b = 4'b1111; #10
171 a = 4'b1010; b = 4'b0000; #10
172 a = 4'b1010; b = 4'b0001; #10
173 a = 4'b1010; b = 4'b0010; #10
174 a = 4'b1010; b = 4'b0011; #10
175 a = 4'b1010; b = 4'b0100; #10
176 a = 4'b1010; b = 4'b0101; #10
177 a = 4'b1010; b = 4'b0110; #10
178 a = 4'b1010; b = 4'b0111; #10
179 a = 4'b1010; b = 4'b1000; #10
180 a = 4'b1010; b = 4'b1001; #10
181 a = 4'b1010; b = 4'b1010; #10
182 a = 4'b1010; b = 4'b1011; #10
183 a = 4'b1010; b = 4'b1100; #10
184 a = 4'b1010; b = 4'b1101; #10
185 a = 4'b1010; b = 4'b1110; #10
186 a = 4'b1011; b = 4'b0000; #10
187 a = 4'b1011; b = 4'b0001; #10
188 a = 4'b1011; b = 4'b0010; #10
189 a = 4'b1011; b = 4'b0011; #10
190 a = 4'b1011; b = 4'b0100; #10
191 a = 4'b1011; b = 4'b0101; #10
192 a = 4'b1011; b = 4'b0110; #10
193 a = 4'b1011; b = 4'b0111; #10
194 a = 4'b1011; b = 4'b1000; #10
195 a = 4'b1011; b = 4'b1001; #10
196 a = 4'b1011; b = 4'b1010; #10
197 a = 4'b1011; b = 4'b1011; #10
198 a = 4'b1011; b = 4'b1100; #10
199 a = 4'b1011; b = 4'b1101; #10
200 a = 4'b1011; b = 4'b1110; #10
201 a = 4'b1011; b = 4'b1111; #10
202 a = 4'b1010; b = 4'b0111; #10
203 a = 4'b1100; b = 4'b0000; #10
204 a = 4'b1100; b = 4'b0001; #10
205 a = 4'b1100; b = 4'b0010; #10
206 a = 4'b1100; b = 4'b0011; #10
207 a = 4'b1100; b = 4'b0100; #10
208 a = 4'b1100; b = 4'b0101; #10
209 a = 4'b1100; b = 4'b0110; #10
210 a = 4'b1100; b = 4'b0111; #10
211 a = 4'b1100; b = 4'b1000; #10
212 a = 4'b1100; b = 4'b1001; #10
213 a = 4'b1100; b = 4'b1010; #10
214 a = 4'b1100; b = 4'b1011; #10
215 a = 4'b1100; b = 4'b1100; #10
216 a = 4'b1100; b = 4'b1101; #10
217 a = 4'b1100; b = 4'b1110; #10
218 a = 4'b1100; b = 4'b1111; #10
219 a = 4'b1101; b = 4'b0000; #10
220 a = 4'b1101; b = 4'b0001; #10
221 a = 4'b1101; b = 4'b0010; #10
222 a = 4'b1101; b = 4'b0011; #10
223 a = 4'b1101; b = 4'b0100; #10
224 a = 4'b1101; b = 4'b0101; #10
225 a = 4'b1101; b = 4'b0110; #10
226 a = 4'b1101; b = 4'b0111; #10
227 a = 4'b1101; b = 4'b1000; #10
228 a = 4'b1101; b = 4'b1001; #10
229 a = 4'b1101; b = 4'b1010; #10
230 a = 4'b1101; b = 4'b1011; #10
231 a = 4'b1101; b = 4'b1100; #10
232 a = 4'b1101; b = 4'b1101; #10
233 a = 4'b1101; b = 4'b1110; #10
234 a = 4'b1101; b = 4'b1111; #10
235 a = 4'b1110; b = 4'b0000; #10

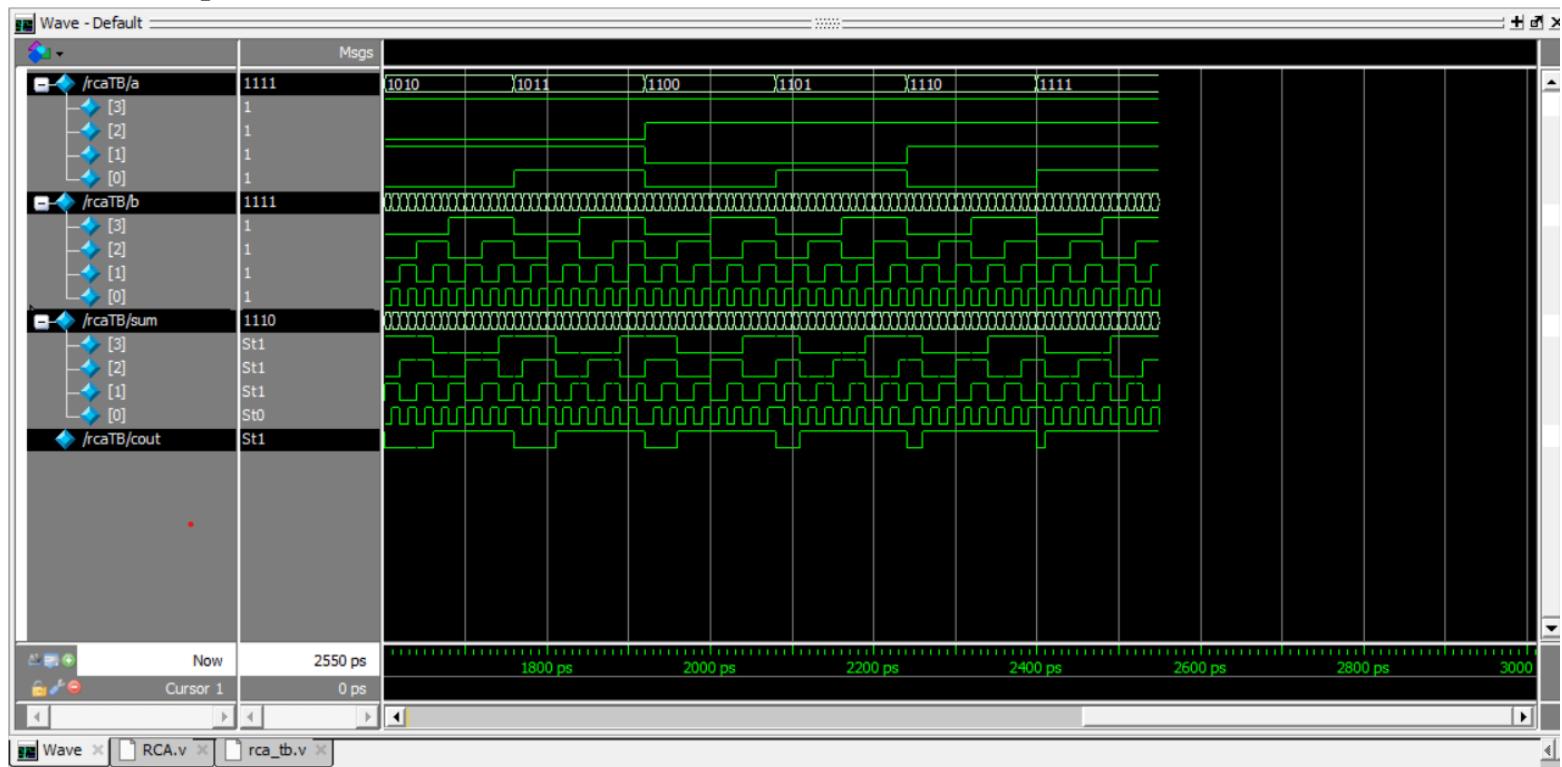
```

```

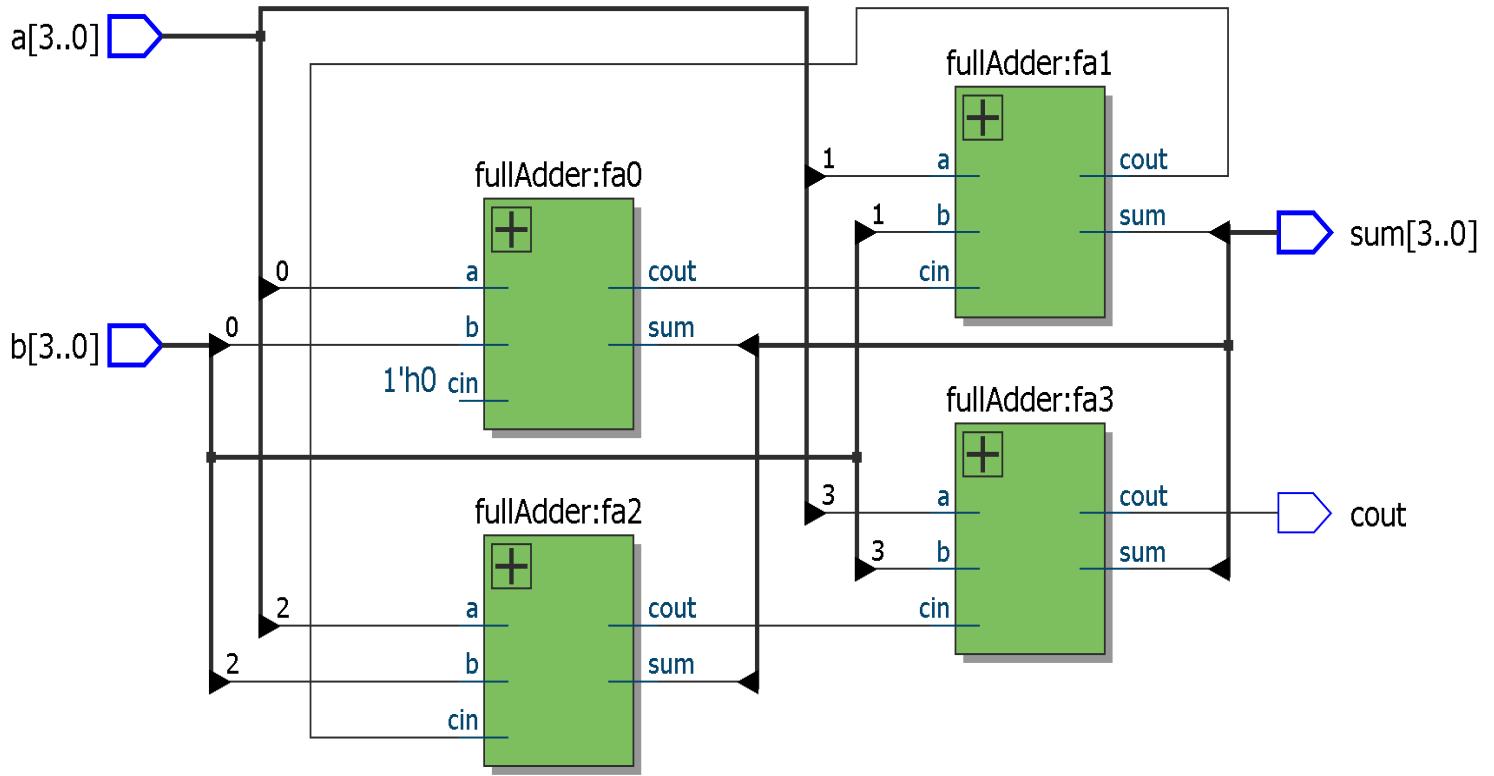
236      a = 4'b1110; b = 4'b0001; #10
237      a = 4'b1110; b = 4'b0010; #10
238      a = 4'b1110; b = 4'b0011; #10
239      a = 4'b1110; b = 4'b0100; #10
240      a = 4'b1110; b = 4'b0101; #10
241      a = 4'b1110; b = 4'b0110; #10
242      a = 4'b1110; b = 4'b0111; #10
243      a = 4'b1110; b = 4'b1000; #10
244      a = 4'b1110; b = 4'b1001; #10
245      a = 4'b1110; b = 4'b1010; #10
246      a = 4'b1110; b = 4'b1011; #10
247      a = 4'b1110; b = 4'b1100; #10
248      a = 4'b1110; b = 4'b1101; #10
249      a = 4'b1110; b = 4'b1110; #10
250      a = 4'b1110; b = 4'b1111; #10
251      a = 4'b1111; b = 4'b0000; #10
252      a = 4'b1111; b = 4'b0001; #10
253      a = 4'b1111; b = 4'b0010; #10
254      a = 4'b1111; b = 4'b0011; #10
255      a = 4'b1111; b = 4'b0100; #10
256      a = 4'b1111; b = 4'b0101; #10
257      a = 4'b1111; b = 4'b0110; #10
258      a = 4'b1111; b = 4'b0111; #10
259      a = 4'b1111; b = 4'b1000; #10
260      a = 4'b1111; b = 4'b1001; #10
261      a = 4'b1111; b = 4'b1010; #10
262      a = 4'b1111; b = 4'b1011; #10
263      a = 4'b1111; b = 4'b1100; #10
264      a = 4'b1111; b = 4'b1101; #10
265      a = 4'b1111; b = 4'b1110; #10
266      a = 4'b1111; b = 4'b1111;
267
268
269      initial begin
270          $dumpfile("ripple-carry-adder.vcd");
271          $dumpvars(0, rcaTB);
272          $monitor($time, ": %b + %b = %b, %b", a, b, sum, cout);
273      end
274
275  endmodule

```

Experiment 2 - RCA Waveform:



Experiment 2 - RCA Schematic:



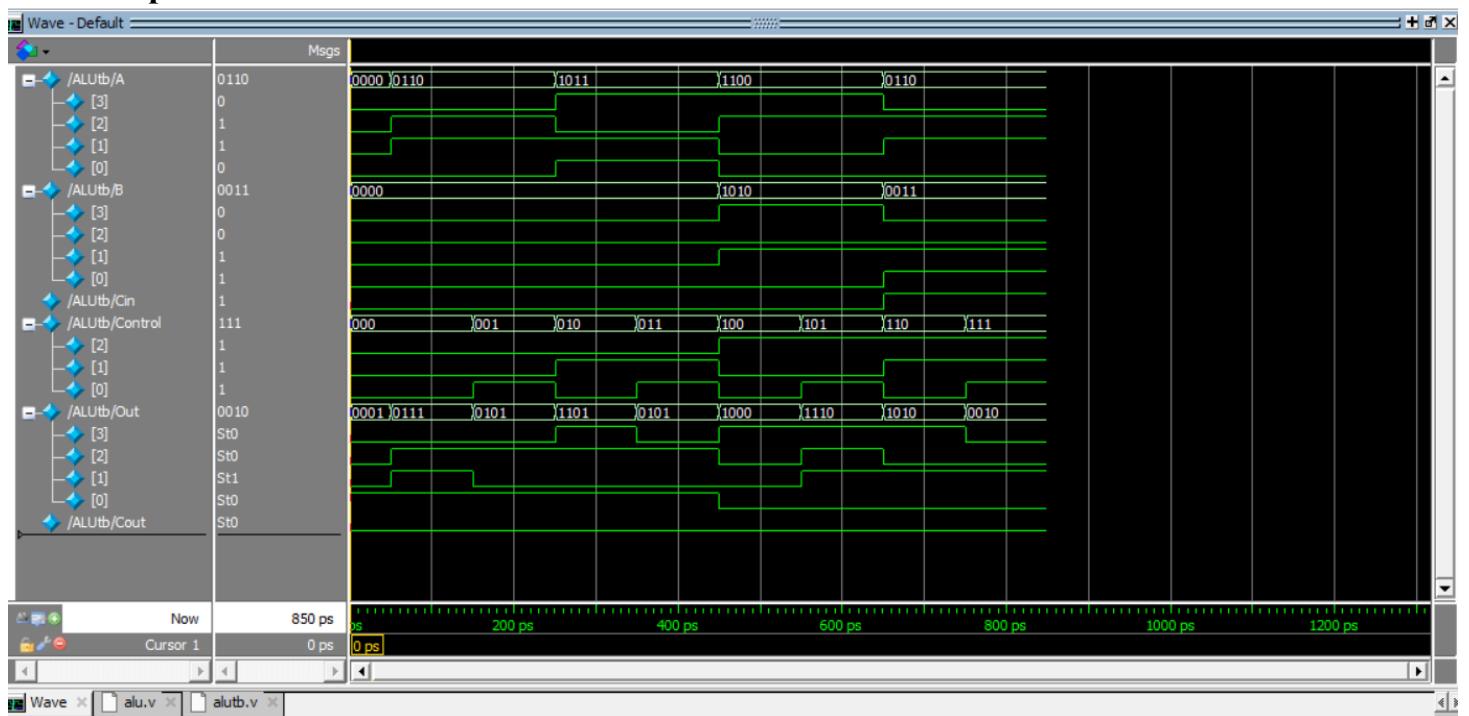
Experiment 3 - ALU Logic Code:

```
1  module ALU (A, B, Cin, Out, Cout, Control);
2
3  input [3:0] A, B;
4  input Cin;
5  input [2:0] Control;
6  output reg [3:0] Out;
7  output reg Cout;
8
9
10 always @ (*) begin
11   case(Control)
12     3'b000:
13       begin
14         Out = A + 4'b0001;
15         Cout = 0;
16       end
17
18     3'b001:
19       begin
20         Out = A - 4'b0001;
21         Cout = 0;
22       end
23
24
25     3'b010:
26       begin
27         Out = {A[0], A[3:1]};
28         Cout = 0;
29       end
30
31     3'b011:
32       begin
33         Out = A >> 1;
34         Cout = 0;
35       end
36
37     3'b100:
38       begin
39         Out = A & B;
40         Cout = 0;
41       end
42
43     3'b101:
44       begin
45         Out = A | B;
46         Cout = 0;
47       end
48
49     3'b110:
50       begin
51         {Cout, Out} = A + B + Cin;
52         Cout = 0;
53       end
54
55     3'b111:
56       begin
57         {Cout, Out} = A - B - Cin;
58         Cout = 0;
59       end
60
61     default:
62       begin
63         Out = 4'b0000;
64         Cout = 0;
65       end
66     endcase
67   end
68
69 endmodule
```

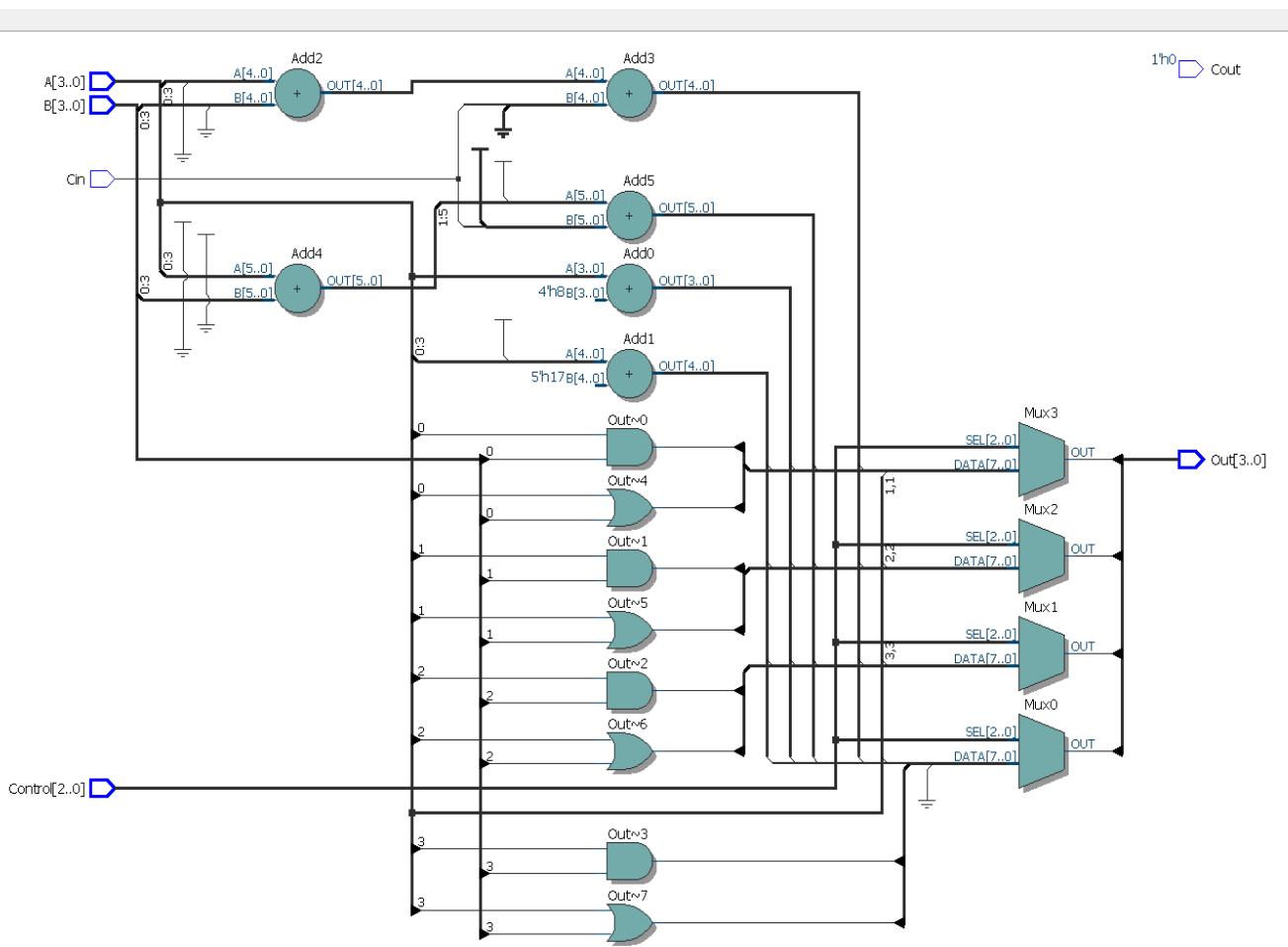
Experiment 3 - ALU Testbench Code:

```
71  module ALU_4bit_tb;
72
73  reg [3:0] A, B;
74  reg Cin;
75  reg [2:0] Control;
76  wire [3:0] O;
77  wire Cout;
78
79  ALUtb(.A(A), .B(B), .Cin(Cin), .Control(Control), .Out(O), .Cout(Cout));
80
81  initial begin
82      A = 4'b0000;
83      B = 4'b0000;
84      Cin = 0;
85      Control = 3'b000;
86
87      #5;
88
89      A = 4'b0110;
90      Control = 3'b000;
91      #10;
92
93      A = 4'b0110;
94      Control = 3'b001;
95      #10;
96
97      A = 4'b1011;
98      Control = 3'b010;
99      #10;
100
101     A = 4'b1011;
102     Control = 3'b011;
103     #10;
104
105     A = 4'b1100;
106     B = 4'b1010;
107     Control = 3'b100;
108     #10;
109
110     A = 4'b1100;
111     B = 4'b1010;
112     Control = 3'b101;
113     #10;
114
115     A = 4'b0110;
116     B = 4'b0011;
117     Cin = 1'b1;
118     Control = 3'b110;
119     #10;
120
121     A = 4'b0110;
122     B = 4'b0011;
123     Cin = 1'b1;
124     Control = 3'b111;
125     #10;
126
127     $stop;
128 end
129
130 initial begin
131     $monitor("Time: %0t | Control: %b | A = %b | B = %b | Cin = %b | O = %b | Cout = %b",
132             $time, Control, A, B, Cin, O, Cout);
133 end
134
135 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.