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

Class:	СР	E300L	Semester:	Fall 2021							
					-						
Points		Document author:	Jerrod Batu								
		Author's email:	batuj1@unlv.nevada.edu								
			-								
		Document topic:	Postlab 3	Postlab 3							
Instructor's	s com	iments:									

1. Introduction / Theory of Operation

Throughout this lab, I will learn about Quartus, VCS, and continue my knowledge of Verilog HDL. Specifically, I will have a better understanding of VCS for testbenches and combinational circuit designs. This lab will include a 7-segment in Verilog, magnitude comparators, ripple adder, and simple ALUS.

- 1. **Megafunction** in Quartus are ready-made, pre-tested practical blocks of code that expand existing design procedures. They decrease design assignments, extremely shorten tasks, and allow designers to optimize their time and energy on improving their system-level items and existing intellectual properties. Altera provides a library of megafunctions necessary to design more efficient logic synthesis and device applications.
- 2. The MegaWizard Plug-In Manager, provides several **applications** of custom megafunction variations to include in a design file. The main goal of these applications is to lessen the amount of instructions when instantiating specific functions. Some of these applications include: altfp_abs, altfp_add_sub, lpm_divide, lpm_and, lpm_or, etc. altfp_abs: floating-point absolute value megafunction altfp_add_sub: floating-point adder/subtractor megafunction lpm_divide: parameterized divider megafunction lpm_and: parameterized AND gate megafunction lpm_or: parameterized OR gate megafunction

2. Prelab

https://docs.google.com/document/d/10xN_bbupBwU2bnhr079n2DhvzONp04eo/edit?us p=sharing&ouid=102808507017671072128&rtpof=true&sd=true

This is the link to my prelab 3.

3. Results of Experiments <u>Experiment 1</u>



```
module seg7 (data, segments);
 1
         input [3:0] data;
 2
 3
         output reg [6:0] segments;
 4
    \Box
         always @ (data) begin
 5
     Ξ
             case (data)
 6
                0 : segments = 7'b0000001;
 7
                1 : segments = 7'bl001111;
 8
                2 : segments = 7'b0010010;
 9
                3 : segments = 7'b0000110;
10
                4 : segments = 7'b1001100;
11
                5 : segments = 7'b0100100;
12
                6 : segments = 7'b0100000;
13
                  : segments = 7'b0001111;
                7
14
                8 : segments = 7'b0000000;
15
                9 : segments = 7'b0000100;
16
                10: segments = 7'b0001000;
17
                11: segments = 7'b1100000;
18
                12: segments = 7'b0110001;
19
                13: segments = 7'b1000010;
20
                14: segments = 7'b0110000;
21
                15: segments = 7'b0111000;
22
                default: segments = 7'blllllll;
23
             endcase
24
          end
25
      endmodule
26
```

This is my Verilog code for a 7-segment display.



This is my DE2 board showing the 7-segment display of "0" when the data input is 0000.



This is my DE2 board showing the 7-segment display of "7" when the data input is 0111.



This is my DE2 board showing the 7-segment display of "C" when the data input is 1100.



This is my DE2 board showing the 7-segment display of "F" when the data input is 1111.

Experiment 2

```
a.
      1
           module magCompare (a,b,gt,eq,lt);
      2
               input a,b;
      3
               output reg gt,eq,lt;
      4
      5
               always @ (a,b) begin
      6
                    gt = (a > b) ? 1 : 0; // greater than
                    eq = (a == b) ? 1 : 0; // equal to
      7
      8
                    lt = (a < b) ? 1 : 0; // less than
      9
                end
     10
           endmodule
     11
     12
           //magCompareTB testbench
     13
           module magCompareTB;
     14
               parameter N = 4;
     15
               reg a tb;
     16
               reg b tb;
     17
               wire gt tb;
     18
               wire eq tb;
     19
               wire lt tb;
     20
               reg [0:N] a tb array;
     21
               reg [0:N] b_tb_array;
     22
               reg [0:N] gt_tb_array;
     23
               reg [0:N] eq_tb_array;
     24
               reg [0:N] lt_tb_array;
     25
     26
         Ξ
               magCompare U0 (
                   .a (a_tb),
     27
     28
                    .b (b_tb),
     29
                   .gt (gt_tb),
     30
                    .eq (eq_tb),
     31
                    .lt (lt tb)
     32
               );
     33
          34
               initial begin
                    $display("\t\ttime\ta tb,\tb tb,\tgt tb,\teq tb,\tlt tb");
     35
                    $monitor("%d,\t%b,\t%b,\t%b,\t%b",$time,a_tb,b_tb,gt_tb,eq_tb,lt_tb);
     36
     37
                end
     38
     39
          initial begin
     40
                   //initialization of input and output arrays
     41
                    a_tb_array [0] = 0;
     42
                   b tb array [0] = 0;
     43
                   gt tb array [0] = 0;
     44
                   eq tb array [0] = 1;
     45
                   lt tb array [0] = 0;
     46
                    #10
     47
                    a_tb_array [1] = 0;
     48
                   b_{tb_array} [1] = 1;
     49
                    gt_tb_array [1] = 0;
                    eq_tb_array [1] = 0;
     50
     51
                    lt tb array [1] = 1;
     52
                    #10
     53
                    a_tb_array [2] = 1;
     54
                   b_tb_array [2] = 0;
     55
                    gt_tb_array [2] = 1;
     56
                    eq tb array [2] = 0;
     57
                    lt_tb_array [2] = 0;
```

58	#10
59	$a_{tb}array [3] = 1;$
60	$b_{tb_{array}}[3] = 1;$
61	$gt_tb_array[3] = 0;$
62	<pre>eq_tb_array [3] = 1;</pre>
63	$lt_tb_array[3] = 0;$
64	#10
65	$a_{tb}array [4] = 0;$
66	$b_{tb_{array}}[4] = 0;$
67	$gt_tb_array[4] = 0;$
68	$eq_{tb}array [4] = 1;$
69	$lt_tb_array [4] = 0;$
70	end
71	
72	integer i;
73	📮 always begin
74	for $(i = 0; i \le N; i = i + 1)$ begin
75	<pre>a_tb <= a_tb_array [i];</pre>
76	<pre>b_tb <= b_tb_array [i];</pre>
77	#100;
78	- end
79	<pre>\$display("\t\t\Test Finished");</pre>
80	\$stop;
81	L end
82	endmodule
83	

This is my Verilog code for the magnitude comparator testbench from prelab 3.

dis.	DVE - TopLevel.2 - [Wave.1] /home/batuj1/	/Fall2021/lab3/simv@csimcluster.ee.unlv.edu								
😾 E	ile <u>E</u> dit <u>V</u> iew Si <u>m</u> ulator Signal <u>S</u> co	pe <u>T</u> race <u>W</u> indow <u>H</u> elp								
	0 ×1s 🕶 🛛 🞇 🆓	▼ _ @ @ D D D D D D D D D D		■ 🗊 🐎 第 🕶 🖬 🕶 66'	• ?\: @ RP •	Any Edge 🗾 1		2 2 0		
1	· 유 유 · · · · · · · · · · · · · · · · ·	: ;) @ @ (} ●] ▼	🕞 🎝 📗 🎬 📰							
×		_ ₩ ₩ ₩	C1:0 REF					Sim:500 (500)		
Nan	ne	Value	0		200	300	400 	500		
þ. I	Group1									
	🛙 a_tb	1'b0								
	🛙 b_tb	1'b0								
	gt_tb	1'b0								
	… r- eq_tb	1'b1								
		1'b0								
	a_tb_array[0:4]	5'b0xxxx			00110					
		5'b0xxxx			01010					
		5'b0xxxx			00100					
	⊕- 🛛 eq_tb_array[0:4]	5'b1xxxx		10011						
		5'b0xxxx			01000					
Lı	New Group									

These are my VCS waveforms for the magnitude comparator testbench. The red waveforms represent the beginning delays at the start of the code.

Chronologic VCS simulator copyright 1991-2017 Contains Synopsys proprietary information. Compiler version N-2017.12-SP2-14; Runtime version N-2017.12-SP2-14; Sep 14 13:49 2021 VCD+ Writer N-2017.12-SP2-14 Copyright (c) 1991-2017 by Synopsys Inc. The file '/home/batuj1/Fall2021/lab3/inter.vpd' was opened successfully. time a_tb, b_tb, gt_tb, eq_tb, lt_tb 0. 0. 0. 0. 1. 0

υ,	Ο,	Ο,	υ,	1,0	0						
100,	Ο,	1,	Ο,	Ο,	1						
200,	1,	Ο,	1,	Ο,	0						
300,	1,	1,	Ο,	1,	0						
400,	Ο,	Ο,	0,	1,	0						
Test Finished											

This is the console output from VCS including the time, inputs, and results of the magnitude comparator testbench.



This is the DE2 board implementation of the magnitude comparator when a > b as a = 1 and b = 0.

c.



This is the DE2 board implementation of the magnitude comparator when a < b as a = 0 and b = 1.



This is the DE2 board implementation of the magnitude comparator when a = b as a = 1 and b = 1.



This is the DE2 board implementation of the magnitude comparator when a = b as a = 0 and b = 0.

Experiment 3

a.

```
1
      //4-bit Ripple Carry Adder
 2
      module rippleAdd (A, B, Cinra, Coutra, Sumra, Segments);
 3
          input [3:0] A, B;
 4
          input Cinra;
 5
          output [3:0] Sumra;
 6
          output Coutra;
 7
          output reg [6:0] Segments;
 8
          wire [3:1] carry;
9
10
          //instantiate four copies of the FullAdder
11
          FullAdd FA0 (A[0],B[0],Cinra,Sumra[0],carry[1]);
12
          FullAdd FA1 (A[1],B[1],carry[1],Sumra[1],carry[2]);
          FullAdd FA2 (A[2],B[2],carry[2],Sumra[2],carry[3]);
13
14
          FullAdd FA3 (A[3],B[3],carry[3],Sumra[3],Coutra);
15
16
          //7segment display
17
    Ę
          always @ (Sumra) begin
18
              case (Sumra)
    Ē
19
                  0 : Segments = 7'b0000001;
20
                  1 : Segments = 7'b1001111;
21
                  2 : Segments = 7'b0010010;
22
                  3 : Segments = 7'b0000110;
23
                  4 : Segments = 7'bl001100;
24
                  5 : Segments = 7'b0100100;
                  6 : Segments = 7'b0100000;
25
                  7 : Segments = 7'b0001111;
26
27
                  8 : Segments = 7'b0000000;
28
                  9 : Segments = 7'b0000100;
29
                  10: Segments = 7'b0001000;
                  11: Segments = 7'b1100000;
30
31
                  12: Segments = 7'b0110001;
                  13: Segments = 7'b1000010;
32
                  14: Segments = 7'b0110000;
33
                   15: Segments = 7'b0111000;
34
35
                  default: Segments = 7'blllllll;
36
              endcase
37
          end
38
      endmodule
39
40
41
      //l-bit Full Adder
42
      module FullAdd (X,Y,Cinfa,Sumfa,Coutfa);
43
          input X, Y, Cinfa;
44
          output Sumfa, Coutfa;
45
46
          assign Sumfa = X ^ Y ^ Cinfa;
47
          assign Coutfa = (X && Y) || ((X^Y) && Cinfa);
48
      endmodule
49
```

This is my Verilog code for implementing a Full Adder module into a 4-bit Ripple Carry Adder including the 7-segment output display.

b. _

```
`timescale 1 ns / 100 ps
 2
 3
      //4-bit Ripple Carry Adder
 4
      module rippleAdd (A, B, Cinra, Coutra, Sumra, Segments);
 5
          input [3:0] A, B;
 6
          input Cinra;
 7
          output [3:0] Sumra;
 8
          output Coutra;
9
          output reg [6:0] Segments;
10
          wire [3:1] carry;
11
12
          //instantiate four copies of the FullAdder
          FullAdd FA0 (A[0],B[0],Cinra,Sumra[0],carry[1]);
13
14
          FullAdd FA1 (A[1],B[1],carry[1],Sumra[1],carry[2]);
15
          FullAdd FA2 (A[2],B[2],carry[2],Sumra[2],carry[3]);
16
          FullAdd FA3 (A[3],B[3],carry[3],Sumra[3],Coutra);
17
18
          //7segment display
   E
19
          always @ (Sumra) begin
20
    Ė
              case (Sumra)
21
                  0 : Segments = 7'b0000001;
22
                  1 : Segments = 7'b1001111;
23
                  2 : Segments = 7'b0010010;
                  3 : Segments = 7'b0000110;
24
25
                  4 : Segments = 7'b1001100;
                  5 : Segments = 7'b0100100;
26
                  6 : Segments = 7'b0100000;
27
28
                  7 : Segments = 7'b0001111;
                  8 : Segments = 7'b0000000;
29
30
                  9 : Segments = 7'b0000100;
                  10: Segments = 7'b0001000;
31
32
                  11: Segments = 7'b1100000;
33
                  12: Segments = 7'b0110001;
34
                  13: Segments = 7'b1000010;
                  14: Segments = 7'b0110000;
35
36
                  15: Segments = 7'b0111000;
37
                  default: Segments = 7'blllllll;
38
              endcase
39
          end
40
     endmodule
41
42
43
      //1-bit Full Adder
44
      module FullAdd (X,Y,Cinfa,Sumfa,Coutfa);
45
          input X, Y, Cinfa;
46
          output Sumfa, Coutfa;
47
          assign Sumfa = X ^ Y ^ Cinfa;
48
          assign Coutfa = (X && Y) || ((X^Y) && Cinfa);
49
50
      endmodule
51
52
53
      //4-bit Ripple Carry Adder TestBench
54
      module rippleAddTB;
          parameter N = 32; //16 tests
55
56
          reg [3:0] A tb;
57
          reg [3:0] B tb;
```

```
58
                    Cin tb;
          reg
59
          wire [3:0] Sum_tb;
60
          wire
                     Cout tb;
61
          wire [6:0] Seg_tb;
62
   Ę
          rippleAdd U0 (
63
64
              .A (A_tb),
65
              .B (B_tb),
66
              .Cinra (Cin tb),
67
              .Sumra (Sum_tb),
68
              .Coutra (Cout_tb),
69
70
               .Segments (Seg_tb)
     L
          );
71
72
    Ę
          initial begin
73
           $display("\t\ttime\tA tb,\tB tb,\tCin tb,\tSum tb,\tCout tb,\tSeg tb");
74
              $monitor("%d,\t%b,\t%b,\t%b,\t%b,\t%b,\t%b,\t%b",$time,A tb,B tb,Cin tb,Sum tb,Cout tb,Seg tb);
75
          end
76
77
          integer i;
78
    always begin
79
              for (i = 0; i \le N; i = i + 1) begin
80
                  #10;
81
                if (i >= 16) begin
82
                      A tb = i;
83
                      B_{tb} = i + 1;
84
                      Cin_{tb} = i - 1;
85
                end
    þ
86
                else begin
87
                  A_{tb} = i;
                  B_{tb} = i;
88
89
                  Cin_tb = i;
90
                end
91
92
              end
93
                   $display("\t\Test Finished");
94
                   Sstop;
95
          end
96
      endmodule
97
```

This is my Verilog code for the 4-bit Ripple Carry Adder testbench including the 7-segment output display.

	с.																																	
ø	g DVE - TopLevel.2 - [Wave.1] /home/batuj1/Fall2021/lab3/simv@csimcluster.ee.unlv.edu — 🗆 🗙																																	
53	Elie Edit View Simulator Signal Scope Trace Window Help																																	
	3300 ×100ps - 🛛 👯 🏘		3 🛛 🖿 🛛 🖛	2			<u>ا</u> ر (- <u>111</u> -	5 - 6	6° • °	10	FPP 🔻	I 🗲 -	S An	y Edge		•	1	₿ ®.	©, (∥		2 3		n C	0									
	ف ال) @ (f) 🕹 🛛 🏦 🗸	- 00	£¥ [چ 🍯																													
×		▼ 343 ▼ 158 ▼																															C1: RE	3300 F
Na	lame Value			1.1			500		. 1			1000		. I			1500					2000					2500				. [3000		
÷	- Group1																																	
	. ■ A_tb[3:0]	15->0	0)(1	χ 2	3	4	5	6	7	8	9	10	(11	12	13	14	15	0	(1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		0->1	0	χ 1	X 2	3	4	5	6	7	8	9	10	11	12	13	14	15	(1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0
	🛙 Cin_tb	St0->St1]]						1]				1								
		15	0	χ 3	4	χ 7	8	11	12	15	0	3	4	7	8	X 11	12	15	2	3	6	7	10	11	14	15	2	3	6	7	10	11	14	15
		Sto																																
		56	1	χ 6	76	15	χo	96	49	56	1	6	76	15	χ o	96	49	56	18	6	32	15	8	96	48	56	18	6	32	15	8)	96	48	56
	New Group																																	

These are my waveforms from VCS that display the inputs and outputs of the 4-bit Ripple Carry Adder testbench including the 7-segment output display. The blocks of red lines indicate delay.

<u>×</u>	I <u>T</u> ype: <mark>I II ▼</mark> <u>S</u> everity:		ode: All			•	8 ()	
Ш	Contains Synopsys pr	oprietary	informati	on.				
Ш	Compiler version N-2	017,12-SP2	-14; Runt	ime vers	ion N-20	17,12-9	6P2-14; Se	ep 18 11:24 2021
Ш	VCD+ Writer N-2017.1	.2-SP2-14 C	opyright	(c) 1991	-2017 by	y Synops	sys Inc.	
Ш	The file '/home/batu	ij1/Fall202	1/lab3/in	ter.vpd'	was ope	ened suc	cessfully	•
Ш	time	e A_tb,	B_tb,	Cin_tb,	Sum_tb,	Cout_t	b,	Seg_tb
Ш	0	, xxxx,	XXXX,	×,	XXXX,	×,	×××××××	
Ш	10	, 0000,	0000,	0,	0000,	0,	0000001	
Ш	20	, 0001,	0001,	1,	0011,	0,	0000110	
Ш	30	, 0010,	0010,	0,	0100,	0,	1001100	
Ш	40	, 0011,	0011,	1,	0111,	0,	0001111	
Ш	50	, 0100,	0100,	0,	1000,	0,	0000000	
Ш	60	, 0101,	0101,	1,	1011,	0,	1100000	
Ш	70	, 0110,	0110,	0,	1100,	0,	0110001	
Ш	80	0, 0111,	0111,	1,	1111,	0,	0111000	
Ш	90	, 1000,	1000,	0,	0000,	1,	0000001	
Ш	100	, 1001,	1001,	1,	0011,	1,	0000110	
Ш	110	, 1010, 4044	1010,	0,	0100,	1,	1001100	
Ш	120	, 1011,	1011,	1,	0111,	1,	0001111	
Ш	130	1100,	1100,	0,- 1	1000,	1,	1100000	
Ш	140	, 1101, 1110	1101,	1,	11000	1,-	01100000	
Ш	150	· 1110, 11110,	1110,	4	1100,	1,	0110001	
Ш	180	0000	0001	1,-	0010	1, 0	0010010	
Ш	190	0000	0001,	0	0010,	0,	0010010	
Ш	190	0010	0010,	1	0110	ŏ,	0100000	
Ш	200	0011	0100	n,	0111	Ň,	0001111	
Ш	210	0100	0101	1.	1010	Ŏ.	0001000	
Ш	220	0101.	0110.	ō.	1011.	ő.	1100000	
Ш	230	. 0110.	0111.	1.	1110.	ó.	0110000	
Ш	240	. 0111.	1000.	ō.	1111.	0.	0111000	
Ш	250	, 1000,	1001,	1,	0010,	1,	0010010	
Ш	260	, 1001,	1010,	0,	0011,	1,	0000110	
Ш	270	, 1010,	1011,	1,	0110,	1,	0100000	
	280	, 1011,	1100,	0,	0111,	1,	0001111	
	290	, 1100,	1101,	1,	1010,	1,	0001000	
	300), 1101,	1110,	0,	1011,	1,	1100000	
	310), 1110,	1111,	1,	1110,	1,	0110000	
	320	, 1111,	0000,	0,	1111,	0,	0111000	
	Test	Finished						

This is my console output from VCS that displays the time for the test to finish, the inputs, and the outputs for the 4-bit Ripple Carry Adder testbench including the 7-segment output display.

d.



This is my DE2 board with the implementation of the 4-bit RCA displaying the output of 3 + 5 (0011 + 0101 = 1000 and no Cout). The 7-segment display shows the addition to be "8", and the LEDs will follow the output of 1000.



This is my DE2 board with the implementation of the 4-bit RCA displaying the output of 8 + 7 (1000 + 0111 = 1111 and no Cout). The 7-segment display shows the addition to be "F", and the LEDs will follow the output of 1111.



This is my DE2 board with the implementation of the 4-bit RCA displaying the output of 15 + 1 (1111 + 0001 = 0000 and 1 Cout). The 7-segment display shows the addition to be "0" because the addition goes out of the bit range, and the LEDs will follow the output of 0000 with LEDR5 as the Cout.



This is my DE2 board with the implementation of the 4-bit RCA displaying the output of 11 + 11 (1011 + 1011 = 0110 and 1 Cout). The 7-segment display shows the addition to be "6" because the addition goes out of the bit range, and the LEDs will follow the output of 0110 with LEDR5 as the Cout.

Experiment 4



This is my Verilog code for implementing a 4-bit ALU module including the 7-segment output display.

b.

```
1
       `timescale 1 ns / 100 ps
2
3
      //simpleALU
      module simpleALU (A, B, Cin, Cntrl, O, Cout, segments);
4
5
         input [3:0] A,B;
6
          input [2:0] Cntrl;
 7
          input Cin;
8
          output reg [3:0] O;
9
          output reg Cout;
10
          output reg [6:0] segments;
11
12
          reg [3:0] Carry, Borrow;
13
          always 0 (*)
    14
          begin
15
               case(Cntrl)
16
                   3'b000: begin
                                                       //INC
17
                                   0 = A + 1;
18
                                   if (A == 4'b1111)
19
     ¢
                                       begin
20
                                         Cout = 1;
21
                                       end
22
                                   else
23
     Þ
                                       begin
24
                                       Cout = 0;
25
                                       end
26
                                 end
     ╘
27
                   3'b001: begin
                                                       //DEC
28
                                   0 = A - 1;
29
                                  Cout = 0;
30
                                 end
     31
                   3'b010: begin
                                                       //ROR
32
                                   O = \{A[0], A[3], A[2], A[1]\};
33
                                   Cout = 0;
34
                                 end
    þ
35
                   3'b011: begin
                                                       //SHR
36
                                   0 = A >> 1;
37
                                  Cout = 0;
38
                                 end
     ╘
                   3'bl00: begin
39
                                                       //AND
40
                                   O = A \& B;
41
                                  Cout = 0;
42
                                 end
                                                       //OR
43
     Þ
                   3'bl01: begin
44
                                   O = A \mid B;
45
                                   Cout = 0;
46
                                 end
     ╘
47
                   3'bll0: begin
                                                       //ADD
48
                                   0 = A + B + Cin;
49
                                   Carry = (A && B) || ((A && Cin) || (B && Cin));
50
                                   Cout = Carry;
51
                                end
52
                   3'blll: begin
                                                       //SUB
53
                                   0 = A - B - Cin;
                                   Borrow = (!A && (B ^ Cin)) || (B && Cin);
54
55
                                   Cout = Borrow;
56
                                 end
57
               endcase
```

```
L
58
           end
 59
 60
           //7-Segment Display
     F
 61
           always 0 (0) begin
 62
               case (0)
 63
                  0 : segments = 7'b0000001;
                  1 : segments = 7'bl001111;
 64
                  2 : segments = 7'b0010010;
 65
                  3 : segments = 7'b0000110;
 66
                  4 : segments = 7'b1001100;
 67
 68
                  5 : segments = 7'b0100100;
                  6 : segments = 7'b0100000;
 69
 70
                  7 : segments = 7'b0001111;
                  8 : segments = 7'b0000000;
 71
 72
                  9 : segments = 7'b0000100;
                  10: segments = 7'b0001000;
 73
 74
                  11: segments = 7'b1100000;
 75
                  12: segments = 7'b0110001;
                  13: segments = 7'b1000010;
 76
 77
                  14: segments = 7'b0110000;
                  15: segments = 7'b0111000;
 78
 79
                  default: segments = 7'blllllll;
 80
               endcase
 81
           end
 82
       endmodule
 83
 84
       //simple ALU testbench
 85
       module simpleALU_tb;
 86
           parameter N = 32; //32 tests
 87
           reg [3:0] Atb,Btb;
 88
           reg [2:0] Sel;
 89
           reg Cintb;
 90
           wire [3:0] Otb;
 91
           wire Couttb;
 92
           wire [6:0] Segtb;
 93
           simpleALU U0 (
 94
     Ę
 95
               .A (Atb),
               .B (Btb),
 96
 97
               .Cin (Cintb),
 98
               .Cntrl (Sel),
 99
               .0 (Otb),
              .Cout (Couttb),
101
               .segments (Segtb)
      );
103
104
           initial begin
105
              $display("\t\ttime\tAtb,\tBtb,\tCintb,\tSel,\tOtb,\tCouttb,\tSegtb");
106
               L
107
           end
108
109
           integer i;
     F
110
           always begin
111
              for (i = 0; i \le N; i = i + 1) begin
112
                  #10;
113
                  Atb = $random;
114
                  Btb = $random;
115
                   Cintb = $random;
116
                   Sel = $random;
117
               end
118
                   $display ("\t\tTest Finished");
119
                   $stop;
           end
       endmodule
```

This is my Verilog code for the 4-bit ALU module testbench including the 7-segment output display.

С.

🖕 DVE - TopLevel.2 - [Wave.1] /home/batuji/fall2021/lab3/simv@csimcluster.ee.unlv.edu — 🗆 >											
Elle Edit Wew Simulator Signal Scope Irace Window Help											
3 ×100ps ▼ 3 # 4 0 € # #	종] 🗈 🕐 한 🖓 다 🗍 🖻 🎝 10 월 - 67 + 19, 47 Hr - 🕽 🦨 Hr - 🕽 🦨 Hr 모 Edge 🔄 🚺 🛯 전 이 🔍 🔍 Q,										
↓ 0 B B B B B B B B B B B B B B B B B B											
	C13 REF	Sim:3300 (3297)									
lame Value	0	3000									
- Group1											
		15 7 9 5									
■ Btb[3:0] ×	1 X 13 X 12 X 10 X 15 X 5 X 0 X 3 X 5 X 15 X 12 X 8 X 6 X 10 X 15 X 5 X 12 X 7 X 9 X 1 X 8 X 14 X 3 X 9 X 6 X	3 11 13 15									
∎ Sel[2:0] ×	3 \ 2 \ 5 \ 6 \ 7 \ 6 \ 5 \ 2 \ 5 \ 3 \ 6 \ 2 \ 1 \ 6 \ 1 \ 2 \ 1 \ 7 \ 4 \ 0 \ 4 \ 5 \ 1 \ 0 \ 2 \ 4 \ 3 \	4 2 5 4									
🛙 Cintb Stx											
		3 11 13 5									
r∟Couttb St×											
	18 \ 48 \ 66 \ 8 \ 79 \ 66 \ 48 \ 1 \ 6 \ 1 \ 48 \ 36 \ 1 \ 79 \ 96 \ 8 \ 66 \ 36 \ 1 \ 49 \ 79 \ 8 \ 79 \ 1 \ 66 \ 1 \ 36 \	6 96 66 36									
- New Group											

These are my waveforms from VCS that display the inputs and outputs of the 4-bit ALU testbench including the 7-segment output display. The blocks of red lines indicate delay.

×	<u>T</u> ype: <mark>⊰ ⊰ ▼</mark> <u>S</u> e	everity: [3 3	; <u> </u>	de: All			- @		
	Chronologic VCS	3 simulat	or copyr	ight 199	91-2017				
	Contains Synops	sys propr	ietary i	.nformati	lon.				
	Compiler version	on N-2017	.12-SP2-	14; Runt	ime vers	ion N-20	17.12-SF	2-14; 9	ep 18 11:53 2021
	VCD+ Writer N-2	2017.12-S	P2-14 Cc	pyright	(c) 1991	-2017 by	i Synopsy	js Inc.	
	The file '/home	e/batuj1/	Fall2021	./lab3/in	nter,vpd'	was ope	ned succ	essfully	J+
		time	Atb,	Btb,	Cintb,	Sel,	Otb,	Couttb,	. Segtb
		0,	xxxx,	XXXX,	×.	XXX,	XXXX,	×.	XXXXXXXX
		10,	0100,	0001,	1,	011,	0010,	0,	0010010
		20,	1101,	1101,	1,	010,	1110,	0,	0110000
		30,	0001,	1101,	0,	101,	1101,	0,	1000010
		40,	1101,	1100,	1,	110,	1010,	1,	0001000
		50,	0101,	1010,	1,	111,	1010,	1,	0001000
		60,	0010,	1111,	0,	110,	0001,	1,	1001111
		70,	1000,	0101,	0,	101,	1101,	0,	1000010
		80,	1101,	0101,	1,	010,	1110,	Ο,	0110000
		90,	0000,	0000,	0,	101,	0000,	υ,	0000001
		100,	0110,	0011,	1,	011,	0011,	0,	0000110
		110,	1011,	0101,	0,	110,	0000,	1,	0000001
		120,	1101,	1111,	1,	010,	1110,	υ,	0110000
		130,	1010,	1100,	0,	010,	0101,	υ,	0100100
		140,	4044	1000,	0,	001,	0000,	0,	0000001
		150,	1011,	0110,	0,	110,	4044	1,	1001111
		160,	1100,	1010,	1,-	001,	1011,	0,	1100000
		170,	4440	0101	1,- 1	010,	1010,	0,	10001000
		180,	0010	1100	1,-	444	0101,	0, 4	0100010
		190,	10010,	0111	1,-	100	0101,	1,	0100100
		200,	1011	1001	1,	100,	1100	<i>0,</i>	0000001
		210,	0111,	0001,	1, 0	100,	0001	<i>0,</i>	1001111
		220,	0010	10001,	1	100,	1010	0, 0	0001000
		230,	0010,	1110	1	001	0001	<i>°</i> ,	1001000
		240,	1111	0011	1	001,	0001,	1	0000001
		200,	1011	10011,	1	010	1101	0	1000001
		280,	1000	0110	0	100	0000	0,	0000001
		280	1010	0110,	1	011	0101	0	0100100
		200,	1111	0011	1	100	0011	Ň,	0000110
		300	0111	1011	<u> </u>	010	1011	ŏ.	1100000
		310	1001.	1101	ŏ.	101	1101.	ŏ.	1000010
		320.	0101.	1111	1_	100_	0101.	ŏ.	0100100
		Test, Fi	nished	,	-,	2009	VIVI,	~,	0100100
		103011	niisneu						

This is my console output from VCS that displays the time for the test to finish, the inputs, and the outputs for the 4-bit ALU testbench including the 7-segment output display.

d.

https://drive.google.com/file/d/1ukPazOBCfBhTPD_-QSV45y_pNUsIS62B/view? usp=sharing

This is a link to my video displaying the opcode from the table for the 4-bit ALU. The video will also be included within the submission zip file.

4. Answers to questions

Question 1: What is an unintentional latch in a Verilog Design? Is it a good or bad design practice?

An unintentional latch in a Verilog Design is when the user inserts a latch in place of where combinational logic can be found. An example of this is when the user takes out the default case within a case statement. Instead of a multiplexer being formed, a latch would take its place because the net is not assigned to any known values. These unintentional latches may also occur from any missing signals within the sensitivity list. Unintentional latches are also called unintended latches or inferred latches. These unintentional latches are considered as bad design practice as the user is unintentionally creating these latches. By accidentally making these latches, the original design is altered; therefore, the user would need to go back into the Verilog code and rewrite it to where the combinational logic would form multiplexers.

Question 2: Why do we need a testbench? Is the waveform simulation not good enough?

We need a testbench to verify the functionality of a design and to report the inputs and outputs in a readable format within the console. The waveform simulation is only good enough in specific circumstances. By using a testbench, the user can check a large number of signals rather than by manually inputting them into the waveform simulation. In addition, the testbench, itself, can generate a periodic clock signal, acquire signal waveforms, and create a simulation report. Forcing inputs within a waveform simulation may be time consuming when there are multiple signals; hence, the testbench would make verification of functionality much quicker.

Question 3: Explain the differences between \$monitor and \$display.

\$monitor and \$display have several differences. \$monitor is a computing program for editing and viewing, whereas \$display is a screen that only shows graphics or text. \$display and \$write both display arguments in the order that they are laid out within the argument list. The key difference between the two is that \$display is used when values are to be printed to the console, and \$monitor is to be called only one time to print the value of a variable whenever it is to be altered. \$monitor is utilized to monitor signals when values change throughout the compilation of the Verilog code, and \$display is utilized every time to print values or display the immediate values of signals.

5. Conclusions & Summary

This lab was pretty straightforward as the lab instructions were somewhat clear. The only parts I did have confusion was for experiment 2 where the instructions said to create a Magnitude Comparator from prelab 2; however, this was a typo instead of prelab 2 it should've been prelab 3. The ripple adder was a great review because I remember doing a similar lab in CPE 200 L.

Experiment 1 seemed to have been carried over from CPE 200 L as the 7-segment display module is somewhat identical to the one in the previous class. Experiment 4 would probably be the most challenging as the 4-bit ALU implementation involved if statements within cases to include the cout within the outputs. I enjoyed reading upon testbenches as they were my struggle in the CPE 200 L class, and I am more comfortable with writing testbenches now than I was before. In addition, the hardest part of this lab was filming the video for the 4-bit ALU operations because the video took up a lot of data. Overall, I enjoyed this lab, and my knowledge of testbenches has increased as well as the implementation of our Verilog code into the DE2 board.