CSCI 355 - Lab 5 Report

Student: Luka Karanovic
Student #: 665778833
Instructor: Ajay Shrestha

1 - Pre-Lab Answers

4.1 Why is a multi-bit adder processed from the least-significant bit to the most significant bit while a multi-bit comparator is processed from the most-significant bit to the least-significant bit?

When you add numbers, you want to start from the least-significant bit because the carry-out of one bit becomes the carry-in for the next.

For example, you can't add 37 + 95 by starting with 9 + 3, getting 2 and carrying over the 1 for 7 + 5 and getting 3 with another carry of 1. We know 37 + 95 doesn't equal 231, and that carrying like this doesn't work because of how we set up our number system and digits to work.

When comparing numbers or strings, you want to start from the most-significant bit because if one number of string is greater at bit i, it will always be greater regardless of any bits that come after it.

- This way, we can achieve the result of the comparison as quickly as possible (don't have to compare less-significant bits if we know one is greater than other already)

Think of numbers again, if we compare 2 3-digit numbers, say 100 and 099. We start at the most-significant digit and see 1 > 0. No matter what happens after that digit, the number starting with 0 can't be greater than the one starting with 1, as we can see 100 > 099. This logic applies for strings too.

- Mathematically: r^n > r^(n-1) + r^(n-2) + … + r^0

4.2 Write the decimal number 7129 in different representations in Verilog HDL:

Binary = 13'b1101111011001
Octal = 13'o15731
Hexadecimal = 13'h1BD9
Decimal = 13'd7129

2 - Lab Procedures

6.1 N-bit adder

6.1.1 Write the structural Verilog HDL code for a full adder with the module interface as shown below

Verilog Code:

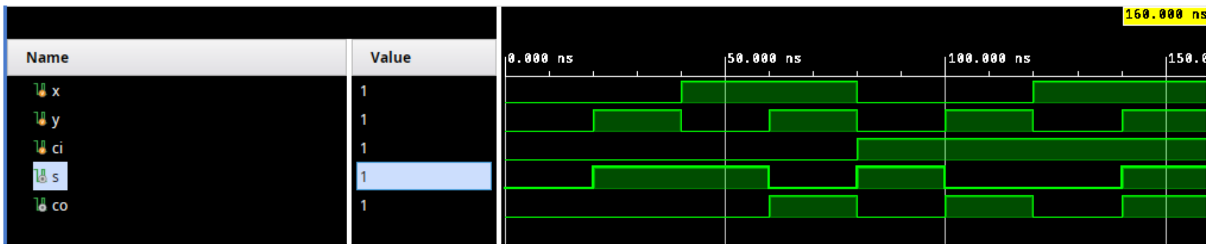
<u>FIlename: fullAdder.v</u>

```
module fullAdder(ci, xi, yi, so, co);
        input ci, xi, yi;
        output so, co;

        wire xy, xoy, xyc;
        // carry out
        and(xy, xi, yi);
        and(xyc, xoy, ci);
        or(co, xy, xyc);
        // sum
        xor(xoy, xi, yi);
        xor(so, xoy, ci);

endmodule
```

Testbench:
FIlename: fullAdder_tb.v

```
module fullAdder_tb();

        reg x, y, ci;
        wire s, co;
        fullAdder ut(ci, x, y, s, co);

        initial begin
        {ci, x, y} = 3'd0; # 20;
        {ci, x, y} = 3'd1; # 20;
        {ci, x, y} = 3'd2; # 20;
        {ci, x, y} = 3'd3; # 20;
        {ci, x, y} = 3'd4; # 20;
        {ci, x, y} = 3'd5; # 20;
        {ci, x, y} = 3'd6; # 20;
        {ci, x, y} = 3'd7; # 20;

        $display("Test completed");
        $finish;
        end

endmodule
```

Timing Diagram:



6.1.2 Write the structural Verilog code for an eight-bit adder that uses eight of the full adders from 6.1.1.

Verilog Code:
FIlename: eightBitAdder.v

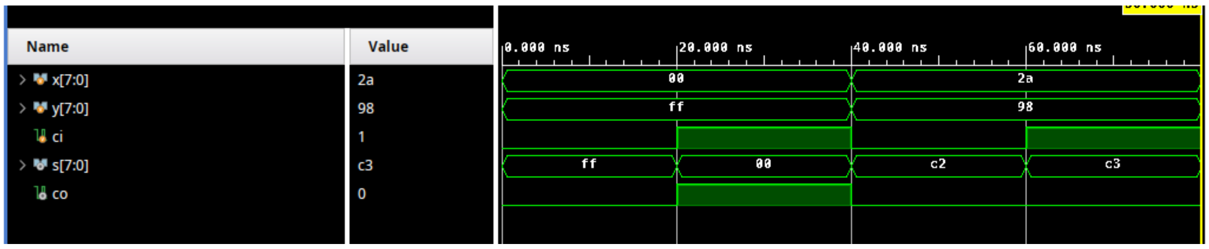
```
module eightBitAdder(carryin, x, y, s, carryout);
        input [7:0] x, y;
        input carryin;
        output [7:0] s;
        output carryout;

        wire c[7:1];

        fullAdder stage0 (carryin, x[0], y[0], s[0], c[1]);
        fullAdder stage1 (c[1], x[1], y[1], s[1], c[2]);
        fullAdder stage2 (c[2], x[2], y[2], s[2], c[3]);
        fullAdder stage3 (c[3], x[3], y[3], s[3], c[4]);
        fullAdder stage4 (c[4], x[4], y[4], s[4], c[5]);
        fullAdder stage5 (c[5], x[5], y[5], s[5], c[6]);
        fullAdder stage6 (c[6], x[6], y[6], s[6], c[7]);
        fullAdder stage7 (c[7], x[7], y[7], s[7], carryout);

endmodule
```

Testbench:

<u>FIlename: eightBitAdder_tb.v</u>

```
module eightBitAdder_tb();
        reg [7:0] x, y;
        reg ci;

        wire [7:1] c;
        wire [7:0] s;
        wire co;
        eightBitAdder ut(ci, x, y, s, co);


        initial begin
        // Test case 1: 00000000 + 11111111 - carryin = 0
        ci = 1'd0;
        x = 8'h00;
        y = 8'hFF;
        #20;

        // Test case 2: 00000000 + 11111111 - carryin = 1
        ci = 1'd1;
        x = 8'h00;
        y = 8'hFF;
        #20;

        // Test case 3: 00101010 + 10011000 - carryin = 0
        ci = 1'd0;
        x = 8'h2A;
        y = 8'h98;
        #20;
        // Test case 4: 00101010 + 10011000 - carryin = 1
        ci = 1'd1;
        x = 8'h2A;
        y = 8'h98;
        #20;

        $display("Test completed");
        $finish;
        end
endmodule
```

Timing Diagram:



Name	Value
x[7:0]	2a
y[7:0]	98
ci	1
s[7:0]	c3
co	0

6.1.3 Write the <u>behavioral</u> Verilog code for a generic adder (use parameter value 32 for a case of 32-bit adder) which uses the full adder module from 6.1.1.

Verilog Code:

<u>FIlename: 32bitAdder.v</u>

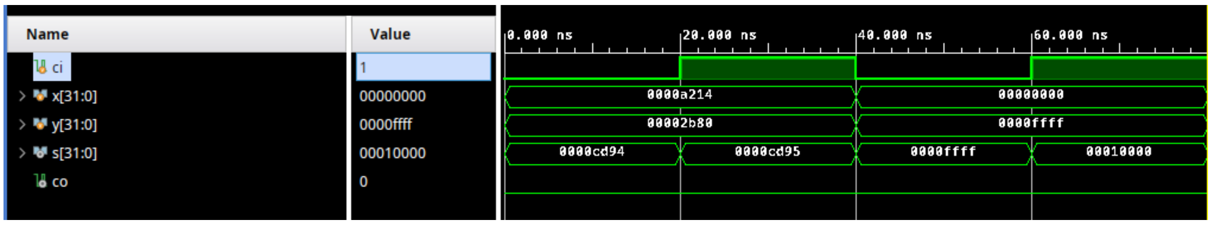
```
module nBitAdder(carryin, x, y, s, carryout);
        // Define parameter, input and outputs.
        parameter n = 32;
        input carryin;
        input [n-1:0] x, y;

        output [n-1:0] s;
        output carryout;

        wire[n:0] c;
        genvar i;

        // Set up variables for generate for loop
        assign c[0] = carryin;
        assign carryout = c[n];

        generate
        for(i = 0; i <= n-1; i = i + 1)
        begin: addbit
        fullAdder stage(c[i], x[i], y[i], s[i], c[i+1]);
        end
        endgenerate
```

Endmodule

Testbench:
FIlename: 32bitAdder_tb.v

```
module nBitAdder_tb();
        reg ci;
        reg [31:0] x, y;

        wire [31:0] s;
        wire co;
        nBitAdder ut(ci, x, y, s, co);

        initial begin
        // Test case 1: carryin = 0
        ci = 1'd0;
        x = 32'hA214;
        y = 32'h2B80;
        #20;

        // Test case 2: carryin = 1
        ci = 1'd1;
        x = 32'hA214;
        y = 32'h2B80;
        #20;

        // Test case 3: carryin = 0
        ci = 1'd0;
        x = 32'h0000;
        y = 32'hFFFF;
        #20;

        // Test case 4: carryin = 1
        ci = 1'd1;
        x = 32'h0000;
        y = 32'hFFFF;
        #20;

        $display("Test completed");
        $finish;
        end

endmodule
```

Timing Diagram:



6.2 Four-Bit Comparator

6.2.1 Write the continuous behavioural assignment (i.e., using "assign") Verilog HDL code for the one-bit comparator with the module interface as shown below

1-bit Comparator Truth Table:

lt_in	eq_in	gt_in	a	b	lt	eq	gt
1	0	0	0	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	0	1	0	0
1	0	0	1	1	1	0	0
0	**1**	**0**	**0**	**0**	**0**	**1**	**0**
0	**1**	**0**	**0**	**1**	**1**	**0**	**0**
0	**1**	**0**	**1**	**0**	**0**	**0**	**1**
0	**1**	**0**	**1**	**1**	**0**	**1**	**0**
0	0	1	0	0	0	0	1
0	0	1	0	1	0	0	1
0	0	1	1	0	0	0	1
0	0	1	1	1	0	0	1

For 1 bit comparator with eq_in = 1 (bold rows:
- if a xnor b, then eq = 1 (and gt = 0, lt = 0)
- If a and ~b, then gt = 1 (and eq =0, lt = 0)
- If ~a and b, then lt = 1 (and gt = 0, eq = 0)

So for eq_in = 1:
- eq = a XNOR b
- gt = a AND ~b
- lt = ~a AND b

Adding gt_in, eq_in and lt_in:

- eq = a XNOR b AND eq_in
- gt = (eq_in AND a AND ~b) OR gt_in
- lt = (eq_in AND ~a AND b) OR lt_in

Verilog Code:

FIlename: comparator.v

```
module comparator(
	input wire eq_in,
	input wire gt_in,
	input wire lt_in,
	input wire a,
	input wire b,
	output wire eq,
	output wire gt,
	output wire lt
   );

   // could use < > = but I wanted to do it with and or not
   assign eq = eq_in & ~(a ^ b);
   assign gt = gt_in | (eq_in & a & ~b);
   assign lt = lt_in | (eq_in & ~a & b);

endmodule
```

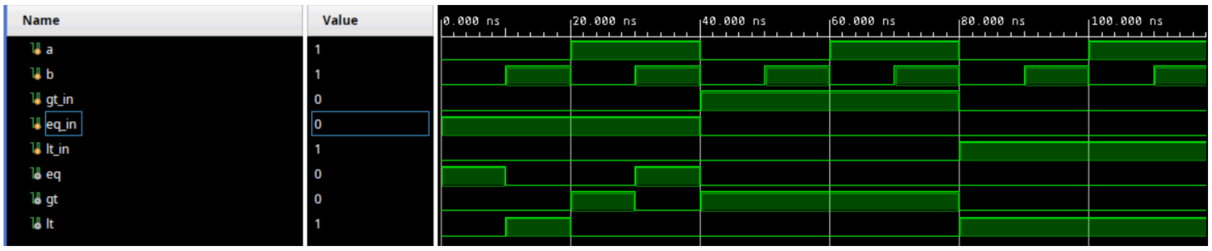
Testbench:

FIlename: comparator_tb.v

```
module comparator_tb();
   reg a, b, gt_in, eq_in, lt_in;
   wire eq, gt, lt;

   comparator ut(eq_in, gt_in, lt_in, a, b, eq, gt, lt);

   initial begin
      eq_in = 1;
      lt_in = 0;
      gt_in = 0;

      a = 0;
      b = 0;
      #10;
      a = 0;
      b = 1;
      #10;
      a = 1;
      b = 0;
      #10;
      a = 1;
      b = 1;
```

```
        #10;

        eq_in = 0;
        gt_in = 1;

        a = 0;
        b = 0;
        #10;
        a = 0;
        b = 1;
        #10;
        a = 1;
        b = 0;
        #10;
        a = 1;
        b = 1;
        #10;

        gt_in = 0;
        lt_in = 1;

        a = 0;
        b = 0;
        #10;
        a = 0;
        b = 1;
        #10;
        a = 1;
        b = 0;
        #10;
        a = 1;
        b = 1;
        #10;

        $display("Done.");
        $finish;
    end

endmodule
```

Timing Diagram:



6.2.2 Write the structural Verilog HDL code for a four-bit comparator with the help of the one-bit comparator module from section 6.2.1

Verilog Code:

FIlename: fourBitComparator.v

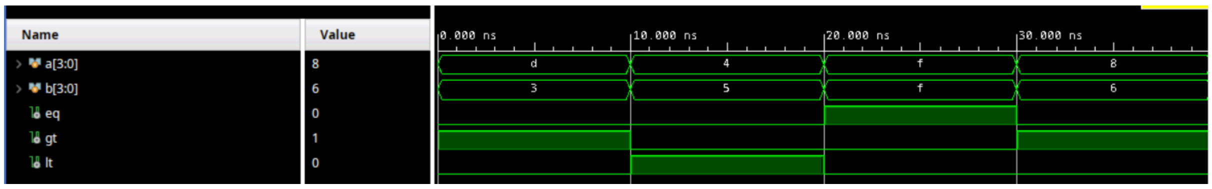
```
module fourBitComparator(
        input wire [ 3:0 ] a,
        input wire [ 3:0 ] b,
        output wire equal,
        output wire greaterThan,
        output wire lessThan
    );

    wire [2:0] eq, gt, lt;

    comparator stage0(1, 0, 0, a[3], b[3], eq[0], gt[0], lt[0]);
    comparator stage1(eq[0], gt[0], lt[0], a[2], b[2], eq[1], gt[1], lt[1]);
    comparator stage2(eq[1], gt[1], lt[1], a[1], b[1], eq[2], gt[2], lt[2]);
    comparator stage3(eq[2], gt[2], lt[2], a[0], b[0], equal, greaterThan, lessThan);
endmodule
```

Testbench:

FIlename: fourBitComparator_tb.v

```
module fourBitComparator_tb();

    reg [3:0] a, b;
    wire eq, gt, lt;

    fourBitComparator ut(a, b, eq, gt, lt);

    initial begin

        a = 4'b1101; // 13
        b = 4'b0011; // 3
        #10;
        a = 4'b0100; // 4
        b = 4'b0101; // 5
        #10;
        a = 4'b1111; // 15
        b = 4'b1111; // 15
        #10;
        a = 4'b1000; // 8
        b = 4'b0110; //6
        #10;

        $display("Done.");
        $finish;
    end

endmodule
```

Timing Diagram:



Name
Value
a[3:0]
8
b[3:0]
6
eq
0
gt
1
lt
0
0.000 ns
10.000 ns
20.000 ns
30.000 ns
d
4
f
8
3
5
f
6