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

Class:	CPE300L	Semester:	Fall 2021

Points		Document author:	Jerrod Batu
		Author's email:	batuj1@unlv.nevada.edu
		Document topic:	Postlab 6

Instructor's comments:

1. Introduction / Theory of Operation

Throughout this lab I will continue my knowledge about general data paths and their implementation with each component.

a. The **digital circuit design** can be tested through either a testbench or through waveforms. Testbenches compare Verilog code to its expected output by comparing a large number of inputs and outputs. This allows the user to efficiently check the correctness of a circuit. Waveforms can also test a digital circuit design by comparing inputs to the outputs in a "form of a wave." Here, the inputs would be clocked at a specific time period where the outputs are based on the combinational logic of the Verilog code. Waveforms would not be ideal for a large number of inputs as this would become a long and tedious process; therefore, testbenches would be more efficient for testing a digital circuit design.

b. The **datapath** supports how algorithms may be implemented through condition checking logic circuits. The general datapath includes a register file, ALU, shifter, and a buffer. To identify the values of the control signals, functional tables would be utilized such as the ALU functional table and the shift functional table. An **example** of a datapath structure would be for an algorithm that takes a positive number (N) from input and checks if it is greater than 5. When the N input is greater than 5, the output is 1, otherwise it is 0. To implement this example onto the datapath, the condition logic would include an or gate connected to the last 5 bits of an 8 bit N input and an and gate would connect to the 2nd and 3rd bits. Then a state diagram would show how the input would be determined rather than using a loop, and the control words would determine where the data is being written into, read, shifted, outputted, inputted, and possibly using the ALU. The buffer would determine the final output of the algorithm.

2. Prelab

https://docs.google.com/document/d/1DzrYP_1NJ9M1d7LmcRpmQx2F1VEeUCJU/edit?usp=sharing&ouid=102808507017671072128&rtpof=true&sd=true

This is the link to my prelab 6.

3. Results of Experiments

EXPERIMENT 1

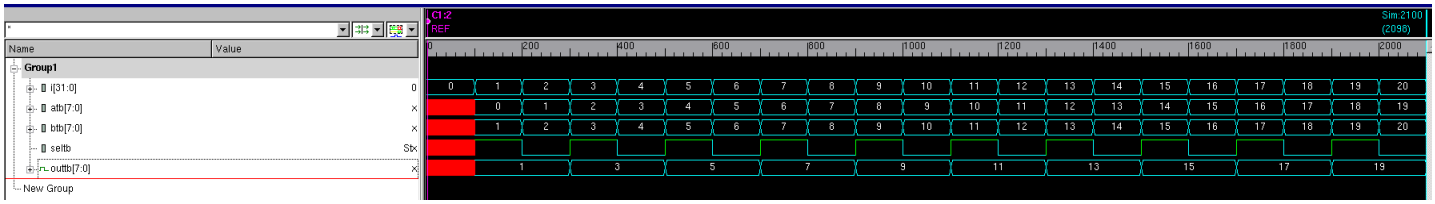
MUX

```
`timescale 1ns / 100ps

// 2-to-1 mux
module gdpMux (a, b, sel, out);
    input[7:0] a;
    input[7:0] b;
    input sel;
    output reg [7:0] out;

    always @(*)
        if(sel == 0)
            out = a;
        else
            out = b;
endmodule
```

This is the Verilog code for the GDP Mux.

```
`timescale 1ns / 100ps

module gdpMuxTB;
  reg [7:0] atb, btb;
    reg seltb;
    wire [7:0] outtb;

    gdpMux U0 (
        .a (atb),
        .b (btb),
        .sel (seltb),
        .out (outtb)
    );

    initial begin
      $display("\t\ttime\tatb\t        btb\t        seltb\touttb");
        $monitor("%d\t%b\t%b\t%b\t%b",$time,atb,btb,seltb,outtb);
    end

    integer i;

    always begin
      for (i = 0; i <= 20; i = i + 1) begin
            #10;
            atb = i;
            btb = i + 1;
            seltb = i + 1;
        end
            $display("\t\t\Test Finished");
            $stop;
    end
endmodule
```

This is the Verilog testbench code for the GDP Mux.



These are the VCS waveforms for the GDP Mux. The red blocks represent delay.

Type: Severity: Code: All

```
Chronologic VCS simulator copyright 1991-2017
Contains Synopsys proprietary information.
Compiler version N-2017.12-SP2-14; Runtime version N-2017.12-SP2-14;  Oct  5 18:31 2021
VCD+ Writer N-2017.12-SP2-14 Copyright (c) 1991-2017 by Synopsys Inc.
The file '/home/batuj1/Fall2021/lab6/inter.vpd' was opened successfully.
                time    atb             btb             seltb   outtb
                   0    xxxxxxxx        xxxxxxxx        x       xxxxxxxx
                  10    00000000        00000001        1       00000001
                  20    00000001        00000010        0       00000001
                  30    00000010        00000011        1       00000011
                  40    00000011        00000100        0       00000011
                  50    00000100        00000101        1       00000101
                  60    00000101        00000110        0       00000101
                  70    00000110        00000111        1       00000111
                  80    00000111        00001000        0       00000111
                  90    00001000        00001001        1       00001001
                 100    00001001        00001010        0       00001001
                 110    00001010        00001011        1       00001011
                 120    00001011        00001100        0       00001011
                 130    00001100        00001101        1       00001101
                 140    00001101        00001110        0       00001101
                 150    00001110        00001111        1       00001111
                 160    00001111        00010000        0       00001111
                 170    00010000        00010001        1       00010001
                 180    00010001        00010010        0       00010001
                 190    00010010        00010011        1       00010011
                 200    00010011        00010100        0       00010011
                Test Finished
```

This is the VCS console for the GDP Mux.

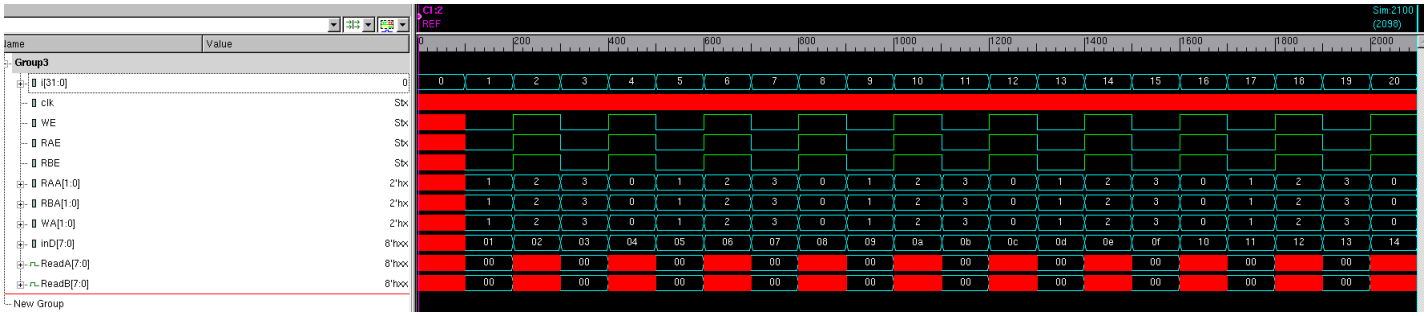
REGISTER FILE

```
`timescale 1ns / 100ps

// register file
module gdpRF(clk, WE, RAE, RBE, RAA, RBA, WA, inD, ReadA, ReadB);
    input clk, WE, RAE, RBE;
    input [1:0] RAA, RBA, WA;
    input [7:0] inD;
    output [7:0] ReadA, ReadB;
    reg [7:0] regF [0:3];

    // Write when WE asserted
    always @(posedge clk)
    if (WE == 1)  regF[WA] <= inD;

    //reading to Port A and B, combinational
    assign ReadA = (RAE)? regF [RAA]:0;
    assign ReadB = (RBE)? regF [RBA]:0;
endmodule

```

This is the Verilog code for the GDP Register File.

```
`timescale 1ns / 100ps

module gdpRFTB;
    reg clk, WE, RAE, RBE;
    reg [1:0] RAA, RBA, WA;
    reg [7:0] inD;
    wire [7:0] ReadA, ReadB;

    `define PERIOD 10

    always
        #(`PERIOD/2) clk = ~clk;

    gdpRF U0 (
      .clk (clk),
      .WE (WE),
      .RAE (RAE),
      .RBE (RBE),
      .RAA (RAA),
      .RBA (RBA),
      .WA (WA),
      .inD (inD),
      .ReadA (ReadA),
      .ReadB (ReadB)
    );

    initial begin
       $timeformat (-9, 1, "ns", 9 );
        $monitor ( "time=%t     WE = %b     RAE = %b     RBE = %b     RAA = %b     RBA = %b     WA = %b     inD = %b     ReadA = %b     ReadB = %b",
                   $time,   WE,   RAE,   RBE,   RAA,   RBA,   WA,   inD,   ReadA,   ReadB);
      #(`PERIOD * 100)
      $display ( "TESTING TIMEOUT" );
      $finish;
    end

    integer i;

    always begin
      for (i = 0; i <= 20; i = i + 1) begin
            #10;
            WE  = i;
            RAE = i;
            RBE = i;
            RAA = i + 1;
            RBA = i + 1;
            WA  = i + 1;
            inD = i + 1;
        end
            $display ( "*****Test PASSED*****" );
            $stop;
    end
endmodule

```

This is the Verilog testbench code for the GDP Register File.



These are the VCS waveforms for the GDP Register File. The red blocks to the left represent delay; however, the red blocks within the output represent don't cares as any outputs would be acceptable here.

```
Type:  Severity:  Code:  All
Chronologic VCS simulator copyright 1991-2017
Contains Synopsys proprietary information.
Compiler version N-2017.12-SP2-14; Runtime version N-2017.12-SP2-14;  Oct  5 18:36 2021
VCD+ Writer N-2017.12-SP2-14 Copyright (c) 1991-2017 by Synopsys Inc.
The file '/home/batuj1/Fall2021/lab6/inter.vpd' was opened successfully.
time=    0.0ns    WE = x    RAE = x    RBE = x    RAA = xx    RBA = xx    WA = xx    inD = xxxxxxxx    ReadA = xxxxxxxx    ReadB = xxxxxxxx
time=   10.0ns    WE = 0    RAE = 0    RBE = 0    RAA = 01    RBA = 01    WA = 01    inD = 00000001    ReadA = 00000000    ReadB = 00000000
time=   20.0ns    WE = 1    RAE = 1    RBE = 1    RAA = 10    RBA = 10    WA = 10    inD = 00000010    ReadA = xxxxxxxx    ReadB = xxxxxxxx
time=   30.0ns    WE = 0    RAE = 0    RBE = 0    RAA = 11    RBA = 11    WA = 11    inD = 00000011    ReadA = 00000000    ReadB = 00000000
time=   40.0ns    WE = 1    RAE = 1    RBE = 1    RAA = 00    RBA = 00    WA = 00    inD = 00000100    ReadA = xxxxxxxx    ReadB = xxxxxxxx
time=   50.0ns    WE = 0    RAE = 0    RBE = 0    RAA = 01    RBA = 01    WA = 01    inD = 00000101    ReadA = 00000000    ReadB = 00000000
time=   60.0ns    WE = 1    RAE = 1    RBE = 1    RAA = 10    RBA = 10    WA = 10    inD = 00000110    ReadA = xxxxxxxx    ReadB = xxxxxxxx
time=   70.0ns    WE = 0    RAE = 0    RBE = 0    RAA = 11    RBA = 11    WA = 11    inD = 00000111    ReadA = 00000000    ReadB = 00000000
time=   80.0ns    WE = 1    RAE = 1    RBE = 1    RAA = 00    RBA = 00    WA = 00    inD = 00001000    ReadA = xxxxxxxx    ReadB = xxxxxxxx
time=   90.0ns    WE = 0    RAE = 0    RBE = 0    RAA = 01    RBA = 01    WA = 01    inD = 00001001    ReadA = 00000000    ReadB = 00000000
time=  100.0ns    WE = 1    RAE = 1    RBE = 1    RAA = 10    RBA = 10    WA = 10    inD = 00001010    ReadA = xxxxxxxx    ReadB = xxxxxxxx
time=  110.0ns    WE = 0    RAE = 0    RBE = 0    RAA = 11    RBA = 11    WA = 11    inD = 00001011    ReadA = 00000000    ReadB = 00000000
time=  120.0ns    WE = 1    RAE = 1    RBE = 1    RAA = 00    RBA = 00    WA = 00    inD = 00001100    ReadA = xxxxxxxx    ReadB = xxxxxxxx
time=  130.0ns    WE = 0    RAE = 0    RBE = 0    RAA = 01    RBA = 01    WA = 01    inD = 00001101    ReadA = 00000000    ReadB = 00000000
time=  140.0ns    WE = 1    RAE = 1    RBE = 1    RAA = 10    RBA = 10    WA = 10    inD = 00001110    ReadA = xxxxxxxx    ReadB = xxxxxxxx
time=  150.0ns    WE = 0    RAE = 0    RBE = 0    RAA = 11    RBA = 11    WA = 11    inD = 00001111    ReadA = 00000000    ReadB = 00000000
time=  160.0ns    WE = 1    RAE = 1    RBE = 1    RAA = 00    RBA = 00    WA = 00    inD = 00010000    ReadA = xxxxxxxx    ReadB = xxxxxxxx
time=  170.0ns    WE = 0    RAE = 0    RBE = 0    RAA = 01    RBA = 01    WA = 01    inD = 00010001    ReadA = 00000000    ReadB = 00000000
time=  180.0ns    WE = 1    RAE = 1    RBE = 1    RAA = 10    RBA = 10    WA = 10    inD = 00010010    ReadA = xxxxxxxx    ReadB = xxxxxxxx
time=  190.0ns    WE = 0    RAE = 0    RBE = 0    RAA = 11    RBA = 11    WA = 11    inD = 00010011    ReadA = 00000000    ReadB = 00000000
time=  200.0ns    WE = 1    RAE = 1    RBE = 1    RAA = 00    RBA = 00    WA = 00    inD = 00010100    ReadA = xxxxxxxx    ReadB = xxxxxxxx
*****Test PASSED*****
```

This is the VCS console for the Register File.

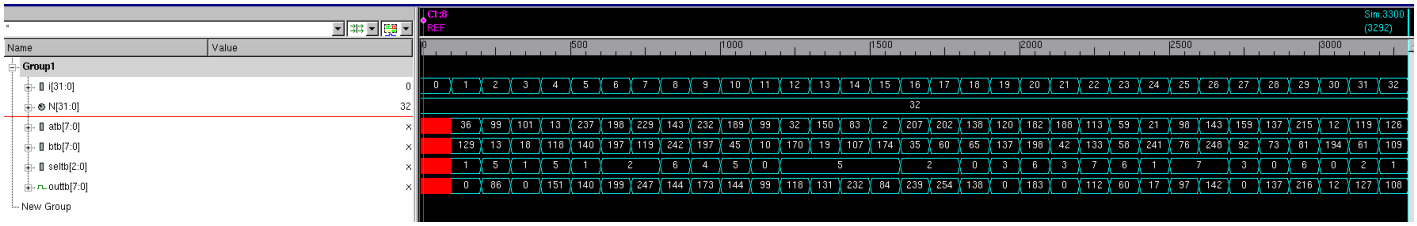
ALU

```
`timescale 1ns / 100ps

// arithmetic logic unit
module gdpALU (a,b,sel, out);
    input [7:0] a,b;
    input [2:0] sel;
    output reg [7:0] out;

    always @ (*)
    begin
        case (sel)
            3'b000: out = a;
            3'b001: out = a & b;
            3'b010: out = a | b;
            3'b011: out = !a;
            3'b100: out = a + b;
            3'b101: out = a - b;
            3'b110: out = a + 1;
            3'b111: out = a - 1;
        endcase
    end
endmodule
```

This is the Verilog code for the GDP ALU.

```
`timescale 1ns / 100ps

module gdpALUTB;
    parameter N = 32; //32 tests
    reg [7:0] atb,btb;
    reg [2:0] seltb;
    wire [7:0] outtb;

    gdpALU U0 (
      .a (atb),
      .b (btb),
      .sel (seltb),
      .out (outtb)
    );

    initial begin
      $display("\t\ttime\tatb\t        btb\t        seltb\touttb");
      $monitor("%d\t%b\t%b\t%b\t%b",$time,atb,btb,seltb,outtb);
    end

    integer i;
    always begin
        for (i = 0; i <= N; i = i + 1) begin
            #10;
            atb = $random;
            btb = $random;
            seltb = $random;
        end
            $display ("\t\t\Test Finished");
            $stop;
    end
endmodule
```

This is the Verilog testbench code for the GDP ALU.



These are the VCS waveforms for the GDP ALU. The red blocks represent delay.

Type: Severity: Code: All

```
Chronologic VCS simulator copyright 1991-2017
Contains Synopsys proprietary information.
Compiler version N-2017.12-SP2-14; Runtime version N-2017.12-SP2-14;  Oct  5 18:43 2021
VCD+ Writer N-2017.12-SP2-14 Copyright (c) 1991-2017 by Synopsys Inc.
The file '/home/batuj1/Fall2021/lab6/inter.vpd' was opened successfully.
                time    atb              btb              seltb   outtb
                   0    xxxxxxxx         xxxxxxxx         xxx     xxxxxxxx
                  10    00100100         10000001         001     00000000
                  20    01100011         00001101         101     01010110
                  30    01100101         00010010         001     00000000
                  40    00001101         01110110         101     10010111
                  50    11101101         10001100         001     10001100
                  60    11000110         11000101         010     11000111
                  70    11100101         01110111         010     11110111
                  80    10001111         11110010         110     10010000
                  90    11101000         11000101         100     10101101
                 100    10111101         00101101         101     10010000
                 110    01100011         00001010         000     01100011
                 120    00100000         10101010         101     01110110
                 130    10010110         00010011         101     10000011
                 140    01010011         01101011         101     11101000
                 150    00000010         10101110         101     01010100
                 160    11001111         00100011         010     11101111
                 170    11001010         00111100         010     11111110
                 180    10001010         01000001         000     10001010
                 190    01111000         10001001         011     00000000
                 200    10110110         11000110         110     10110111
                 210    10111100         00101010         011     00000000
                 220    01110001         10000101         111     01110000
                 230    00111011         00111010         110     00111100
                 240    00010101         11110001         001     00010001
                 250    01100010         01001100         111     01100001
                 260    10001111         11111000         111     10001110
                 270    10011111         01011100         011     00000000
                 280    10001001         01001001         000     10001001
                 290    11010111         01010001         110     11011000
                 300    00001100         11000010         000     00001100
                 310    01110111         00111101         010     01111111
                 320    01111110         01101101         001     01101100
                Test Finished
```

This is the VCS console for the GDP ALU.

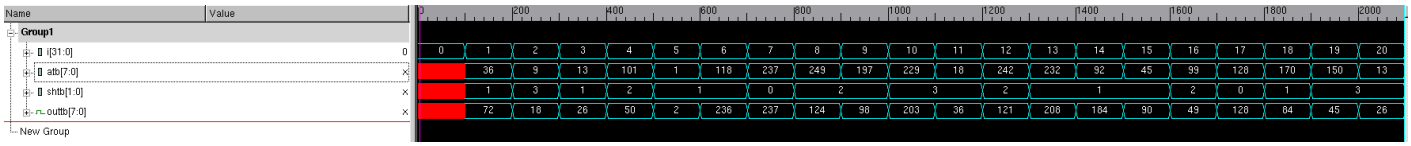
SHIFTER

```
`timescale 1ns / 100ps

// Shifter
module gdpShift (a,sh,out);
    input [7:0] a;
    input [1:0] sh;
    output reg [7:0] out;

    always @ (*)
    begin
        case(sh)
            3'b00: out = a;
            3'b01: out = a << 1;
            3'b10: out = a >> 1;
            3'b11: out= { a[6],a[5],a[4],a[3],a[2],a[1],a[0], a[7] } ;
        endcase
    end
endmodule
```

This is the Verilog code for the GDP Shifter.

```
`timescale 1ns / 100ps

module gdpShiftTB;
    reg [7:0] atb;
    reg [1:0] shtb;
    wire [7:0] outtb;

    gdpShift U0 (
      .a (atb),
      .sh (shtb),
      .out (outtb)
    );

    initial begin
      $display("\t\ttime\tatb\t        shtb\touttb");
      $monitor("%d\t%b\t%b\t%b",$time,atb,shtb,outtb);
    end

    integer i;
    always begin
        for (i = 0; i <= 20; i = i + 1) begin
            #10;
            atb = $random;
            shtb = $random;
        end
            $display ("\t\t\Test Finished");
            $stop;
    end
endmodule
```

This is the Verilog testbench code for the GDP Shifter.



These are the VCS waveforms for the GDP Shifter. The red blocks represent delay.

```
Type:  Severity:  Code:  All
Chronologic VCS simulator copyright 1991-2017
Contains Synopsys proprietary information.
Compiler version N-2017.12-SP2-14; Runtime version N-2017.12-SP2-14;  Oct  5 18:49 2021
VCD+ Writer N-2017.12-SP2-14 Copyright (c) 1991-2017 by Synopsys Inc.
The file '/home/batuj1/Fall2021/lab6/inter.vpd' was opened successfully.
                time    atb             shtb    outtb
                   0    xxxxxxxx        xx      xxxxxxxx
                  10    00100100        01      01001000
                  20    00001001        11      00010010
                  30    00001101        01      00011010
                  40    01100101        10      00110010
                  50    00000001        01      00000010
                  60    01110110        01      11101100
                  70    11101101        00      11101101
                  80    11111001        10      01111100
                  90    11000101        10      01100010
                 100    11100101        11      11001011
                 110    00010010        11      00100100
                 120    11110010        10      01111001
                 130    11101000        01      11010000
                 140    01011100        01      10111000
                 150    00101101        01      01011010
                 160    01100011        10      00110001
                 170    10000000        00      10000000
                 180    10101010        01      01010100
                 190    10010110        11      00101101
                 200    00001101        11      00011010
                Test Finished
```

This is the VCS console for the GDP Shifter.

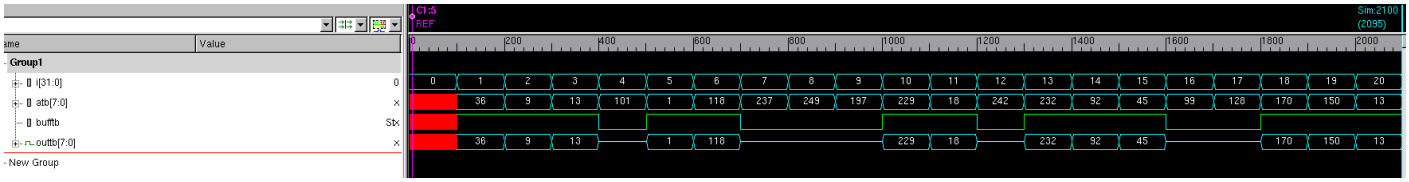
BUFFER

```
1   `timescale 1ns / 100ps
2
3   //buffer
4   module gdpBuffer (a, buff, out);
5       input [7:0] a;
6       input buff;
7       output reg [7:0] out;
8
9       always @(*)
10          if(buff == 1)
11              out = a;
12          else
13              out = 8'bzzzz_zzzz;
14  endmodule
15
```

This is the Verilog code for the GDP Buffer.

```
1   `timescale 1ns / 100ps
2
3   module gdpBufferTB;
4       reg [7:0] atb;
5       reg bufftb;
6       wire [7:0] outtb;
7
8       gdpBuffer U0 (
9         .a (atb),
10        .buff (bufftb),
11        .out (outtb)
12      );
13
14      initial begin
15        $display("\t\ttime\tatb\t        bufftb\touttb");
16        $monitor("%d\t%b\t%b\t%b",$time,atb,bufftb,outtb);
17      end
18
19      integer i;
20      always begin
21          for (i = 0; i <= 20; i = i + 1) begin
22              #10;
23              atb = $random;
24              bufftb = $random;
25          end
26              $display ("\t\t\Test Finished");
27              $stop;
28      end
29  endmodule
30
```

This is the Verilog testbench code for the GDP Buffer.



These are the VCS waveforms for the GDP Buffer. The red blocks represent delay, and the yellow lines represent Z.

Type: Severity: Code: All

```
Chronologic VCS simulator copyright 1991-2017
Contains Synopsys proprietary information.
Compiler version N-2017.12-SP2-14; Runtime version N-2017.12-SP2-14;  Oct  5 18:53 2021
VCD+ Writer N-2017.12-SP2-14 Copyright (c) 1991-2017 by Synopsys Inc.
The file '/home/batuj1/Fall2021/lab6/inter.vpd' was opened successfully.
                time    atb             bufftb  outtb
                   0    xxxxxxxx        x       xxxxxxxx
                  10    00100100        1       00100100
                  20    00001001        1       00001001
                  30    00001101        1       00001101
                  40    01100101        0       zzzzzzzz
                  50    00000001        1       00000001
                  60    01110110        1       01110110
                  70    11101101        0       zzzzzzzz
                  80    11111001        0       zzzzzzzz
                  90    11000101        0       zzzzzzzz
                 100    11100101        1       11100101
                 110    00010010        1       00010010
                 120    11110010        0       zzzzzzzz
                 130    11101000        1       11101000
                 140    01011100        1       01011100
                 150    00101101        1       00101101
                 160    01100011        0       zzzzzzzz
                 170    10000000        0       zzzzzzzz
                 180    10101010        1       10101010
                 190    10010110        1       10010110
                 200    00001101        1       00001101
               Test Finished
```

This is the VCS console for the GDP Buffer.

EXPERIMENT 2

```
1   `timescale 1ns / 100ps
2
3   //Data Path
4   module DP (nIn, clk, IE, WE, WA, RAE, RAA, RBE, RBA, ALU, SH, OE, ne0, out);
5       input clk, IE, WE, RAE, RBE, OE;
6       input [1:0] WA, RAA, RBA, SH;
7       input [2:0] ALU;
8       input [7:0] nIn;
9       output ne0;
10      output wire [7:0] out;
11
12      reg [7:0] rfIn;
13      wire [7:0] rfA, rfB, aluOut, shOut, n;
14
15      initial
16      rfIn = 0;
17
18      always @ (*)
19      rfIn = n;
20
21      gdpMux mux (shOut, nIn, IE, n);
22      gdpRF RF (clk, WE, RAE, RBE, RAA, RBA, WA, rfIn, rfA, rfB);
23      gdpALU theALU (rfA, rfB, ALU, aluOut);
24      gdpShift SHIFT (aluOut, SH, shOut);
25      gdpBuffer buff (shOut, OE, out);
26
27      assign ne0 = n != 0;  //note: checks the false //condition
28
29  endmodule
30
31  // 2-to-1 mux
32  module gdpMux (a, b, sel, out);
33      input[7:0] a;
34      input[7:0] b;
35      input sel;
36      output reg [7:0] out;
37
38      always @(*)
39          if(sel == 0)
40              out = a;
41          else
42              out = b;
43  endmodule
44
45  // register file
46  module gdpRF(clk, WE, RAE, RBE, RAA, RBA, WA, inD, ReadA, ReadB);
47      input clk, WE, RAE, RBE;
48      input [1:0] RAA, RBA, WA;
49      input [7:0] inD;
50      output [7:0] ReadA, ReadB;
51      reg [7:0] regF [0:3];
52
53      // Write when WE asserted
54      always @(posedge clk)
55      if (WE == 1)  regF[WA] <= inD;
56
57      //reading to Port A and B, combinational
```

```
58      assign ReadA = (RAE)? regF [RAA]:0;
59      assign ReadB = (RBE)? regF [RBA]:0;
60  endmodule
61
62  // arithmetic logic unit
63  module gdpALU (a,b,sel, out);
64      input [7:0] a,b;
65      input [2:0] sel;
66      output reg [7:0] out;
67
68      always @ (*)
69      begin
70          case (sel)
71              3'b000: out = a;
72              3'b001: out = a & b;
73              3'b010: out = a | b;
74              3'b011: out = !a;
75              3'b100: out = a + b;
76              3'b101: out = a - b;
77              3'b110: out = a + 1;
78              3'b111: out = a - 1;
79          endcase
80      end
81  endmodule
82
83  // Shifter
84  module gdpShift (a,sh,out);
85      input [7:0] a;
86      input [1:0] sh;
87      output reg [7:0] out;
88
89      always @ (*)
90      begin
91          case(sh)
92              3'b00: out = a;
93              3'b01: out = a << 1;
94              3'b10: out = a >> 1;
95              3'b11: out= { a[6],a[5],a[4],a[3],a[2],a[1],a[0], a[7] } ;
96          endcase
97      end
98  endmodule
99
100 //buffer
101 module gdpBuffer (a, buff, out);
102     input [7:0] a;
103     input buff;
104     output reg [7:0] out;
105
106     always @(*)
107         if(buff == 1)
108             out = a;
109         else
110             out = 8'bzzzz_zzzz;
111 endmodule
112
```

This is the Verilog code for my DP (DataPath) module that would instantiate all of the components verified in experiment 1.

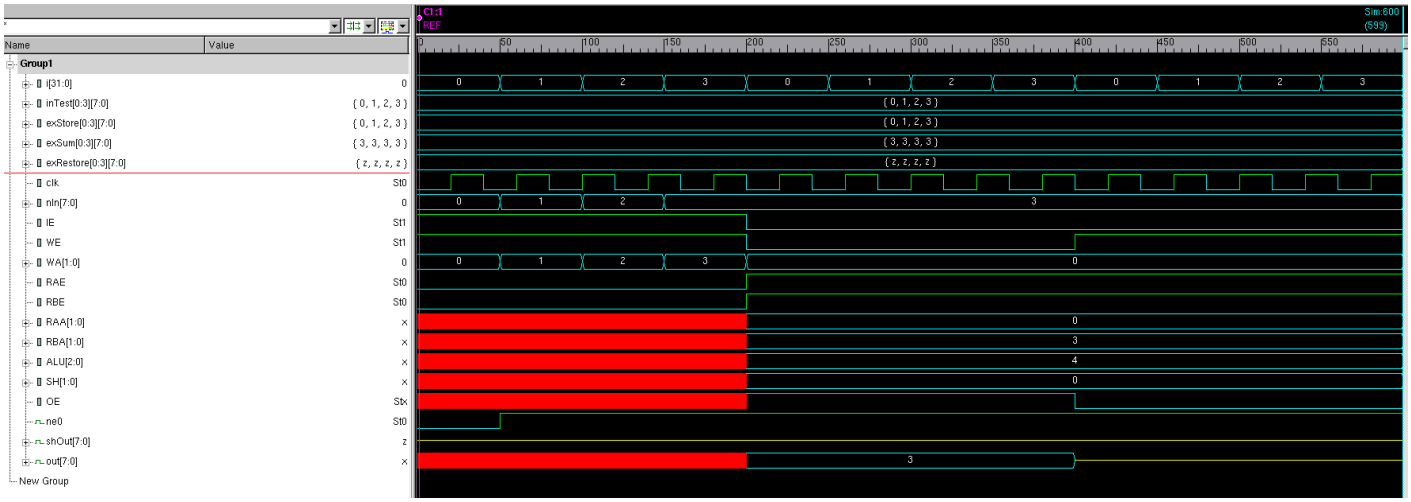
EXPERIMENT 3

```
1   `timescale 1ns / 100ps
2
3   module gdpDPTB;
4       reg clk, IE, WE, RAE, RBE, OE;
5       reg [1:0] WA, RAA, RBA, SH;
6       reg [2:0] ALU;
7       reg [7:0] nIn;
8       wire ne0;
9       wire [7:0] out;
10
11      wire [7:0] shOut;
12      reg [7:0] inTest [0:3];
13      reg [7:0] exStore [0:3];
14      reg [7:0] exSum [0:3];
15      reg [7:0] exRestore [0:3];
16
17      integer i;
18
19      initial
20          begin
21              // initialize test inputs and expected outputs
22              inTest [0] = 8'b00000000;
23              inTest [1] = 8'b00000001;
24              inTest [2] = 8'b00000010;
25              inTest [3] = 8'b00000011;
26              exStore [0] = 8'b00000000;
27              exStore [1] = 8'b00000001;
28              exStore [2] = 8'b00000010;
29              exStore [3] = 8'b00000011;
30              exSum [0] = 8'b00000011;
31              exSum [1] = 8'b00000011;
32              exSum [2] = 8'b00000011;
33              exSum [3] = 8'b00000011;
34              exRestore [0] = 8'bzzzzzzzz;
35              exRestore [1] = 8'bzzzzzzzz;
36              exRestore [2] = 8'bzzzzzzzz;
37              exRestore [3] = 8'bzzzzzzzz;
38
39              // generate clock
40              clk = 0;
41              forever
42                  #2 clk = ~clk;
43          end
44
45      always
46          begin
47              for (i = 0; i <= 3; i = i + 1)          // store into Register File
48                  begin
49                      nIn <= inTest [i];
50                      IE = 1;
51                      WE = 1;
52                      WA = i;
53                      RAE = 0;
54                      RBE = 0;
55                      #5;
56                      if (nIn != exStore[i])
57                          begin
```

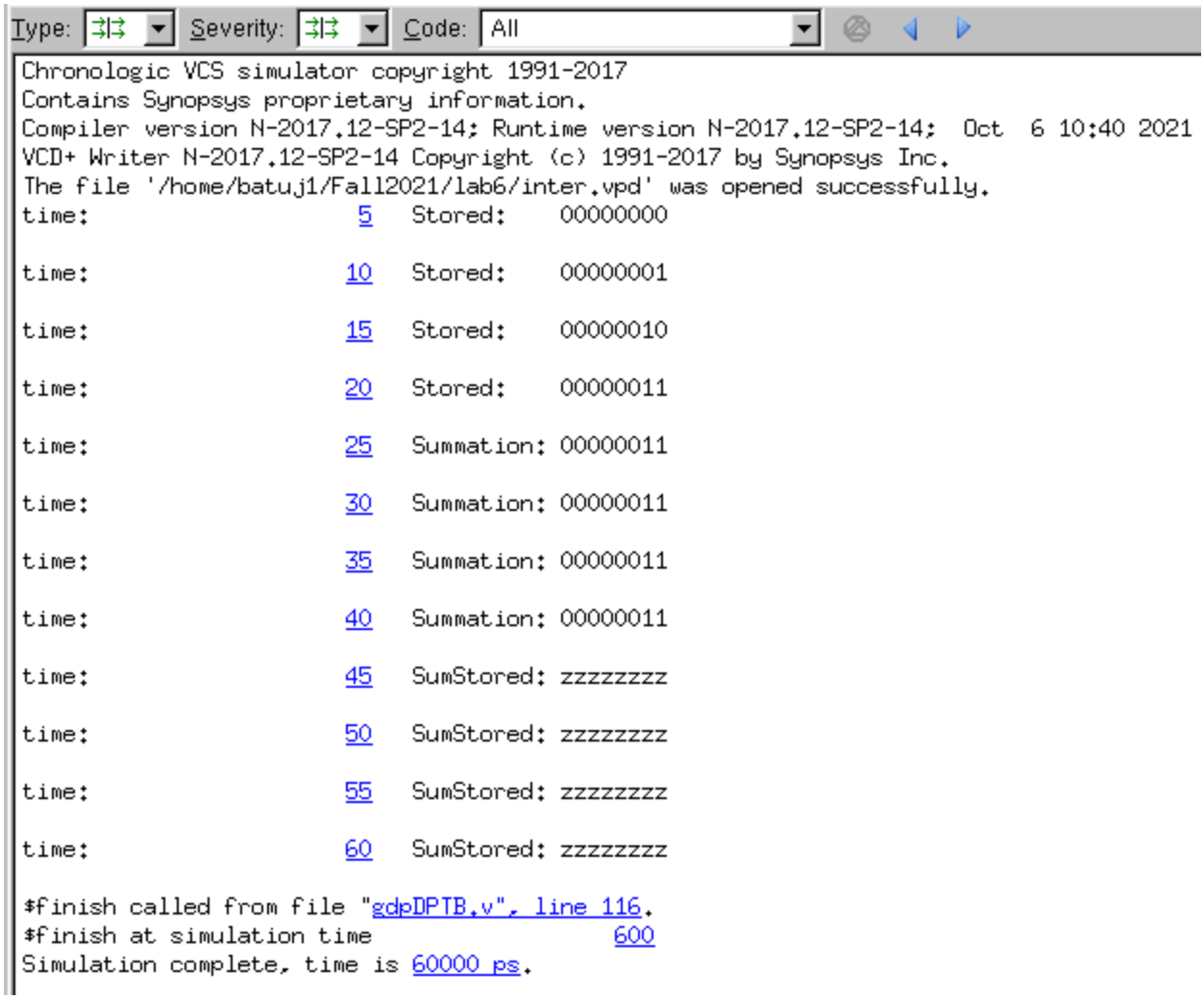
```
                    $write ("Stored incorrectly!");
                    $stop;
                end
            else
                begin
                    $write("time: %d   Stored:    %b", $time, nIn);
                    $display ("");
                end

        end
    for (i = 0; i <= 3; i = i + 1)            // summation
        begin
            IE = 0;
            WE = 0;
            RAE = 1;
            RBE = 1;
            RAA = 00;
            RBA = 11;
            WA = 0;
            ALU = 3'b100;
            SH = 2'b00;
            OE = 1;
            #5;
            if (out != exSum[i])
                begin
                    $write ("Wrong Sum!");
                    $stop;
                end
            else
                begin
                    $write("time: %d   Summation: %b", $time, out);
                    $display ("");
                end
        end
    for (i = 0; i <= 3; i = i + 1)            // store summation into 00
        begin
            IE = 0;
            WE = 1;
            WA = 00;
            RAE = 1;
            RAA = 00;
            RBE = 1;
            RBA = 11;
            ALU = 3'b100;
            SH = 2'b00;
            OE = 0;
            #5;
            if (out != exRestore[i])
                begin
                    $write ("Inproper Storing!");
                    $stop;
                end
            else
                begin
                    $write("time: %d   SumStored: %b", $time, out);
                    $display ("");
                end
```

```
115                 end
116                 $finish;
117             end
118
119         //Instantiate DataPath
120         DP U0 (
121             .nIn (nIn),
122             .clk (clk),
123             .IE (IE),
124             .WE (WE),
125             .WA (WA),
126             .RAE (RAE),
127             .RAA (RAA),
128             .RBE (RBE),
129             .RBA (RBA),
130             .ALU (ALU),
131             .SH (SH),
132             .OE (OE),
133             .ne0 (ne0),
134             .out (out)
135         );
136     endmodule
137
```

This is my Verilog testbench code for the GDP DataPath that would store inputs "00,01,10,11" into the Register File, add "00" and "11", and finally restore back into "00".



These are the VCS waveforms for my DataPath testbench. The red blocks represent don't care as they are not needed to store information within the Register File. The yellow lines for shift represent pass through, and the yellow lines for out represent that the value has been restored into the Register File.



This is the VCS console for my DataPath testbench. From time 5-20, the tested values are being stored within the Register File. From time 25-40, the value in "00" and the value in "11" are being added together from the ALU. From time 45-60, the ALU output is being restored back into the Register File.

4. Answers to questions

Question 1: Control words are from the control unit. The control unit will utilize the control word. Control words indicate the certain cycle number that would utilize the components of the general data path. The control unit is implemented by a finite state machine, and the control words are the inputs to the general data path. By utilizing the control words, the data path would be clocked; therefore, each state would determine at what certain time the control word would be inputted into the data path.

Question 2: To design a control unit for a general data path, a finite state machine would be used; therefore, the finite state machine would indicate when to output into the general data path inputs. Each state would determine where the control words are going into the data path. The initialization of inputs would take place within the first few states, then the next states would

determine if the inputs are being written or read into the register file, then possibly using a shifter or alu, and finally going to the buffer to determine the final output of the algorithm. To determine how each state would go from one to the next, the output of the control signal from the datapath would be implemented as an input in the control unit.

Question 3: A Clock Domain Crossing (CDC), within a digital design, is the process of passing the clock signal from one clock domain to another. This occurs when the data is transferred between flip-flops as one flip-flop would be driven by one clock, and the other flip-flop would be driven by another clock. More than one clock is utilized for Clock Domain Crossing. The main issue caused by Clock Domain Crossing is metastability. Metastability happens when the output of a flip-flop will not reach its expected output and may lead to oscillations between 0's and 1's if one clock domain is clocked relatively close to another clock domain. To prevent metastability from happening, it would be best to use one clock that drives into multiple flip-flops rather than multiple clocks that can cause an error.

5. Conclusions & Summary

Overall, this lab was very similar to the homework provided from the CPE 300 lecture. It was fairly easy, but sometimes it was tedious due to the fact of creating testbenches for every single component of the Data Path. The main issues I had were with experiment 3 when properly storing and restoring the values; however, I just rewatched the video in module 6 again to fully understand how to utilize the Register File. I expect the next lab to implement the control unit for this datapath.