Class:	CPE-300L	Semester:	Spring 2020

Points		Document author:	Brysen Kokubun
		Author's email:	Brysenkokubun96@gmail.com
		Document topic:	Postlab 6

Instructor's comments:

1. Introduction / Theory of Operation

- For this lab experiment we learned about the components of the general data path. The components of the data path that we designed and tested were, a multiplexer, a Register file, an ALU, a shifter and a tristate buffer.

a. Describe how the digital circuit design can be tested

- You are able to test the digital data path circuit, by first testing each component to make sure it is functioning properly. After verifying that the components are correct, then one may write a test bench to test different functions that the data path can perform. Such as reading, writing, outputting values of registers, doing arithmetic operations on values within the registers, etc.

b. What is the data path in general? Provide an example.

- Central processing unit, joining together various components to work together to perform various data operations. An example of a data path would be a processor of some sort that is created to perform any time of data manipulations & operations.

2. Prelab report

- Attached below

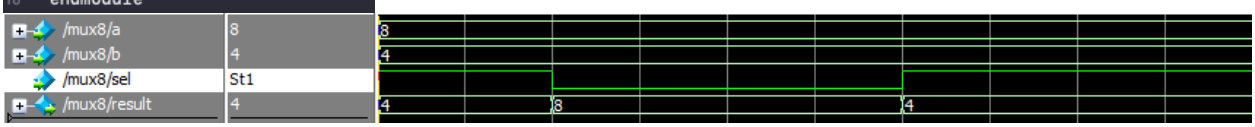
3. Experiment Results

Experiment #1: Implementation of the Data Path I

- Multiplexer Code & Waveform

```
3   module mux8(result, a, b, sel);
4
5       output reg[7:0] result;
6       input[7:0] a;
7       input[7:0] b;
8       input sel;
9
10      always @(*)
11          if(sel == 0)
12              result = a;
13          else
14              result = b;
15
16  endmodule
```

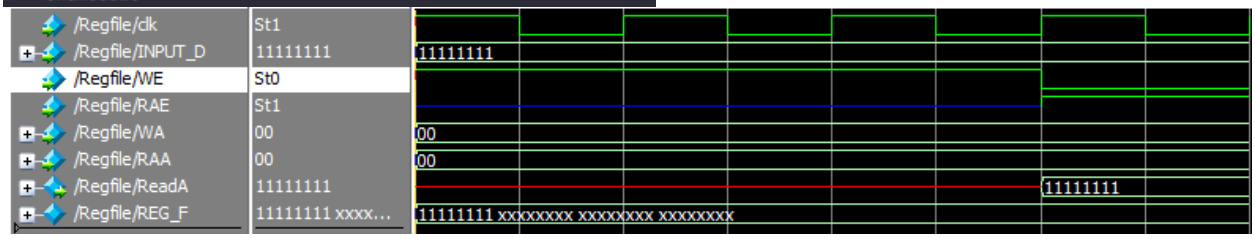
Waveform verifies the selection of either a or b



- Register File Code & Waveform

```
module Regfile(
        input clk,
        input   [1:0]  RAA, // Port A Read address
        output  [7:0] ReadA,   // Port A
        input   [1:0] RBA,    // Port B Read address
        output  [7:0] ReadB,   // Port B
        input   WE,    // Write enable
        input   [1:0] WA,  // Write port register address
        input   [7:0] INPUT_D,  // Write data port
        input   RAE,  // Port A decoder enable
        input   RBE // Port B decoder enable

);
    // width          depth
    reg [7:0]  REG_F [0:3];

// Write only when WE is asserted

always @(posedge clk)
if (WE == 1)  REG_F[WA] <= INPUT_D;

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

endmodule
```

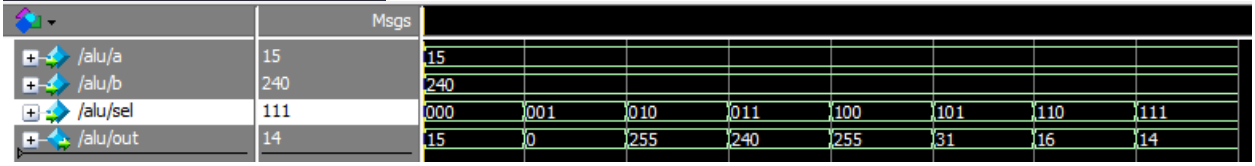
Waveform verifies the input of D, Write Enable, writing to Register A (00) and then Read Enable, then reading value from Register A (00)



- ALU Code & Waveform

```
module alu (out,a,b,sel);
    input [7:0] a,b;
    input [2:0] sel;
    output [7:0] out;
    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
```

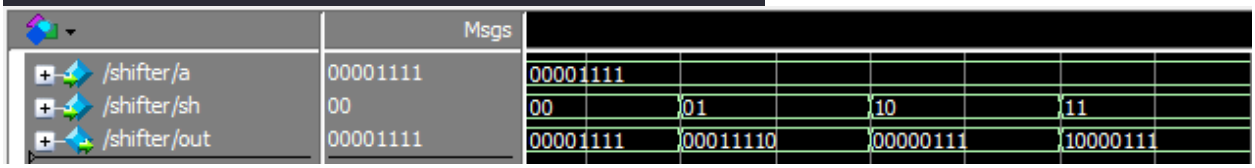
Waveform verifies all arithmetic & logical operations of the ALU



- Shifter Code & Waveform

```
module shifter (out,a,sh);
    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[0],a[7],a[6],a[5],a[4],a[3],a[2], a[1]} ;
        endcase
    end
endmodule
```

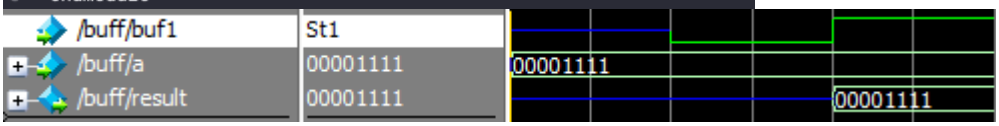
Waveform verifies operations of the shifter



- Tristate Buffer Code & Waveform

```
module buff(output reg [7:0] result, input[7:0] a, input buf1);
    always @(*)
        if(buf1 == 1)
            result = a;
        else
            result = 8'bzzzz_zzzz;
endmodule
```

Waveform verifies operations of tristate buffer

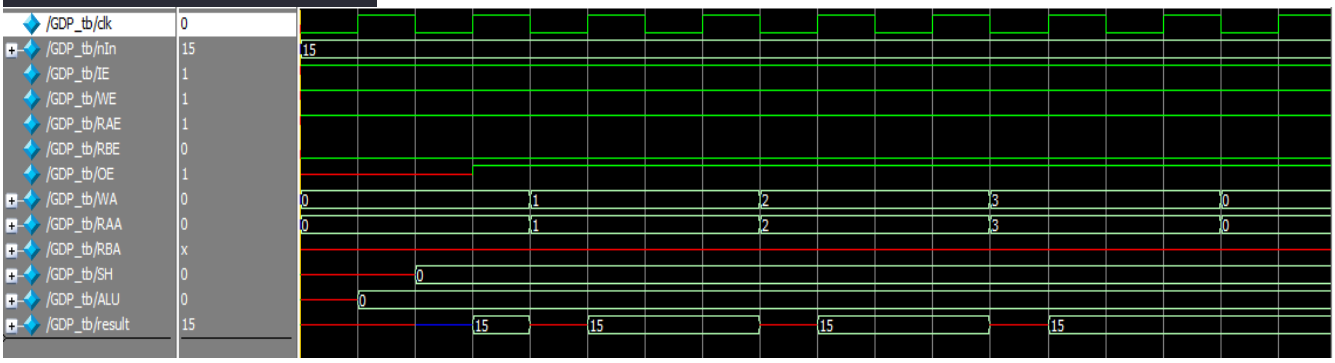


- Test Bench Code & Waveform for DP module with all Components

```
module GDP_tb;

  wire[7:0] result;
  reg [7:0] nIn;
  reg IE, WE, RAE, RBE, OE, clk;

  reg [1:0] WA, RAA, RBA, SH;

  reg [2:0] ALU;

  DP u0(result, nIn, clk, IE, WE, WA, RAE, RAA, RBE, RBA, ALU, SH, OE);

  initial begin  // begin initial
    nIn = 8'b00001111;
    clk = 0;
    forever
    #2 clk = ~clk;
  end


  always begin
    IE = 1;
    WE = 1;
    WA = 2'b00;
    RAE = 1;
    RAA = 2'b00;
    RBE = 0;
    #2;
    ALU = 3'b000;
    #2;
    SH = 2'b00;
    #2;
    OE = 1;
    #2;
```

```
36        IE = 1;
37        WE = 1;
38        WA = 2'b01;
39        RAE = 1;
40        RAA = 2'b01;
41        RBE = 0;
42        #2;
43        ALU = 3'b000;
44        #2;
45        SH = 2'b00;
46        #2;
47        OE = 1;
48        #2;
49
50        IE = 1;
51        WE = 1;
52        WA = 2'b10;
53        RAE = 1;
54        RAA = 2'b10;
55        RBE = 0;
56        #2;
57        ALU = 3'b000;
58        #2;
59        SH = 2'b00;
60        #2;
61        OE = 1;
62        #2;
63
64        IE = 1;
65        WE = 1;
66        WA = 2'b11;
67        RAE = 1;
68        RAA = 2'b11;
69        RBE = 0;
70        #2;
71        ALU = 3'b000;
72        #2;
73        SH = 2'b00;
74        #2;
75        OE = 1;
76        #2;
77
78      end
79    endmodule
```

Waveform verifies the input of 15, Write Enable to Register A (00, 01, 10, 11), Read enable, ALU passing through, Shifter passing through, and output enable



Experiment #2: Implementation of Data Path II

- Data Path Verilog

```
module DP(Result, nIn, clk, IE, WE, WA, RAE, RAA, RBE, RBA, ALU, SH, OE);

    input clk, IE, WE, RAE, RBE, OE;
    input [1:0] WA, RAA, RBA, SH;
    input [2:0] ALU;
    input [7:0] nIn;

    output wire [7:0] Result;

    reg [7:0] rfIn;
    wire [7:0] RFa, RFb, aluOut, shOut, n;

     initial
     rfIn = 0;

    always @ (*)
     rfIn = n;

    mux8 mux (n, shOut, nIn, IE);
    Regfile RF (clk, RAA, RFa, RBA, RFb, WE, WA, rfIn, RAE, RBE);
    alu theALU (aluOut, RFa, RFb, ALU);
    shifter SHIFT (shOut, aluOut, SH);
    buff buffer1 (Result, shOut, OE);

endmodule
```

4. Question

1. What are the control words for the data path?

- The control words for a data path are what cause the data path to perform different operations. For example, the control word for taking an input, writing to a register, performing multiplication and shifting, will have a different control word from taking a feedback value from the shifter and reading from a register and subtracting from another register. The control word is a set of bits which will manipulate every bit of the data path to perform different operations.

5. Conclusions

- To conclude this lab, we were able to create the components of the Data path. The components of our data path included a multiplexer, register file, ALU, shifter and tristate buffer. We verified each component with a wave form. After verifying the components, we combined them all together into a single module. We verified our single module with all the components with a testbench and waveform. The only encountered problem for us would have to have been creating the testbench for the module with all components. After taking some time to see what components needed what signals it was very straightforward. Overall this lab gave us a better understanding of what a Data Path is and how all the components come together to perform various operations.

Class:	CPE-300L	Semester:	Spring 2020

Points		Document author:	Brysen Kokubun
		Author's email:	Brysenkokubun96@gmail.com
		Document topic:	Prelab 6

Instructor's comments:

Introduction / Theory of operation

- For this lab experiment we will be learning how to implement the general datapath and its components. The components for our general datapath is a multiplexer, a Register files, an ALU, a shifter, and a tristate buffer.

Prelab main content

1. Understand the objective of the summation algorithm and explain how the algorithm works.

- The summation algorithm starts with the sum = 0 and takes in some input n. If the value of n is != 0 then add the current sum value and the current n value. Next decrement the n value and check if it is != 0. Keep decrementing the n value until the condition is met. Once the condition is met then output the value of the sum.

2. Identify the components in the general datapath from Fig 2 and explain the use of each component with respect to the summation algorithm?

- The MUX decides if it is a new input n or feedback data from the GDP, the feedback data will not stop until the condition of n = 0 is met.
- The OR gate gives a signal to the control unit and verifies if the n = 0 or n != 0.
- The Reg File is composed of 4 8-bit registers that can write/store, and read data, the data within the Reg File is the accumulated sum.
- The ALU is where the arithmetic logic/operations on the data are performed. The ALU will decrement n by 1 and sums the current sum value with the current n value.
- The shifter will pass through the data.
- The tristate buffer prevents the output to be enabled until n = 0. Once n = 0 then the tristate buffer will allow the data to be outputted.

3. What are all the input and output signals shown in Fig2 associated with each block, explain all of them.

- IE: Input Enable for the MUX indicates if there is new data or if it is feedback data from the GDP
- WE: Write Enable for the Reg File allows the data to be written to a register
- WA: Write Register for the Reg File has 2 signals 1 and 0 each combination allows to select which of the 4 registers to write to
- RAE: Read Enable Register for register A, this allows data in register A to be read into the ALU
- RAA: Read Register for A, has 2 signals 1 and 0 to indicate which register of A to read from
- RBE: Read Enable Register for register B, this allows data in register B to be read into the ALU
- RBA: Read Register for B, has 2 signals 1 and 0 to indicate which register of B to read from
- ALU: Arithmetic Logic Unit, where logic/computation is made, 3 selection signals justify the ALU's operation.
- SH: Shifter, 2 signals that will justify the shifters operation
- OE: Output Enable, 1 signal that will either cause the data to be outputted or feedback through GDP

4. Write a Verilog code for the ALU described in Fig 2b and verify using the Modelsim.

```
module alu(A, B, select, Y, out);
input [7:0] A,B;
input [2:0] select;
output reg [7:0] Y;
output out;
wire [8:0] tmp; //For necessary carry
assign tmp = {1'b0,A} + {1'b0,B};
assign out = tmp[8];
always @(*)
  begin
    case(select)
      3'b000:
        Y = A;
      3'b001:
        Y = A & B;
      3'b010:
        Y = A | B;
      3'b011:
        Y = ~A;
      3'b100:
        Y = A + B;
      3'b101:
        Y = A - B;
      3'b110:
        Y = A + 1;
      3'b111:
        Y = A - 1;
      default:
        Y = A + B;
    endcase
  end
endmodule
```