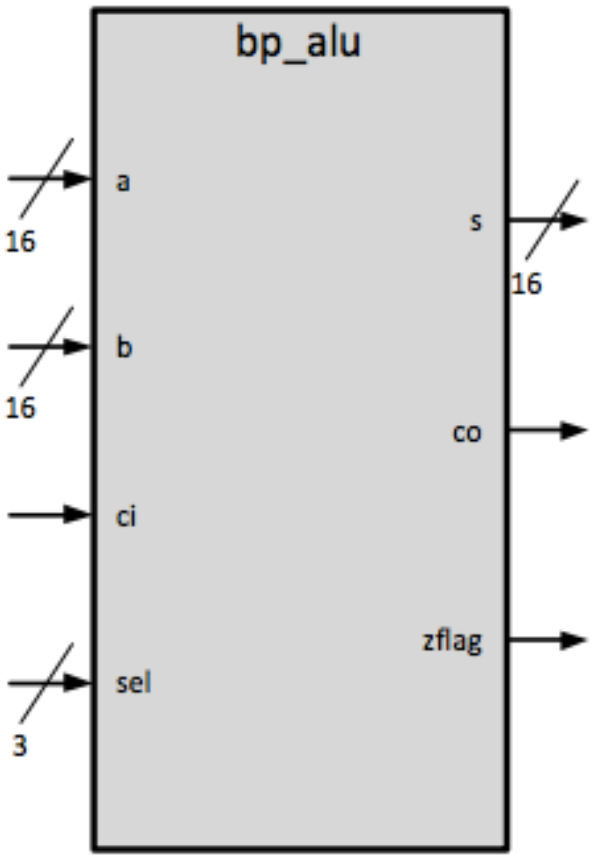
Basic Processor Datapath Design

Vincent Martin
TUID: 913012274
Lab #11 (11/14/2012)

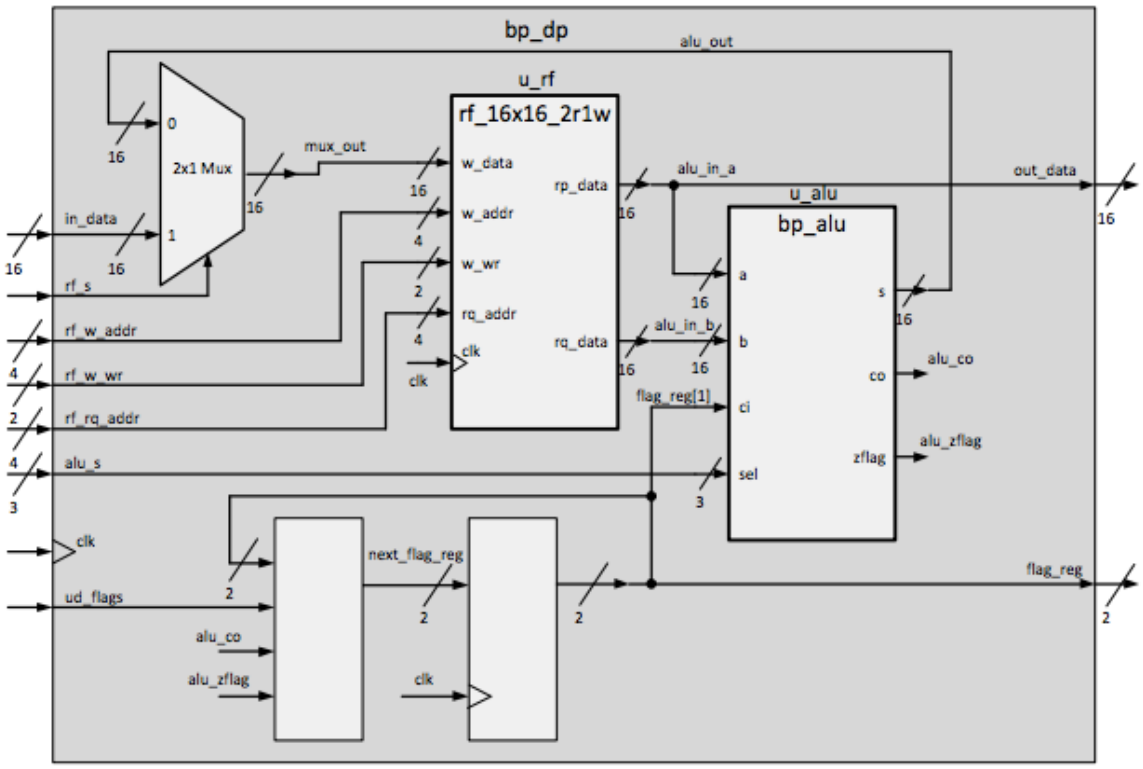
Introduction:
The objective of this lab is to design a datapath component for a general-purpose processor unit. This lab will consist of two major parts, the first of which will be designing the Arithmetic Logic Unit (ALU) and then secondary, implementing that ALU into a datapath block.

This lab will be completed in simulation mode only and will not involve physical hardware, so all testing will be completed via simulation and not on our hardware board.

Applying the Theory to Block Designs
To best understand the modules that are involved in our design it is good to look at block diagrams. Included below are block diagrams for our ALU and the datapath block that it will be implemented into.



ALU Module



Datapath Block Module

Procedures:

Completing this lab will be broken into two sets of procedures. One for creating the ALU and the second, implementing the ALU into the Datapath block.

Creating the ALU:

1. Connect to the development server
2. Open XISE
3. Open the lab11 project file in ~/xilinx/lab11/ folder
4. Use the new source wizard to create the bp_alu module with the following input and output settings
    a. 16 bit input a
    b. 16 bit input b
    c. 1 bit input ci
    d. 3 bit input sel
    e. 16 bit output reg s
    f. 1 bit output reg co
    g. 1 bit output reg zflag
5. Design the internal operations based on the chart below.

Description	Select code	Function
Addition with carry	3'b000	s = a + b + ci co = carry out bit of result zflag = 1 if s == 0
Subtraction with carry	3'b 001	s = a – b – ci co = borrow (carry out) bit of result
Increment a	3'b 010	s = a + 1 co = carry out bit of result
Select b with no operation	3'b 011	s = b co = 1
Bitwise AND of a, b	3'b 100	s = a AND b co = 0
Bitwise OR of a, b	3'b 101	s = a OR b co = 0
Bitwise XOR of a, b	3'b 110	s = a XOR b co = 0
Bitwise NOT of a	3'b 111	s = NOT a co = 0
For all cases above: zflag = 0 if result [**s**] is not 0, zflag = 1 if result [**s**] is 0.		

6.
7. Test the design using iSim and fix any errors that are found.

Integrate the ALU into the Datapath Block

1. Navigate to the bp_dp.v module.
2. Instantiate the bp_alu module into the bp_dp module as u_alu with the following connections as seen in the block diagram for the Datapath block.
    a. .a(alu_in_a),
    b. .b(alu_in_b),
    c. .ci(flag_reg[1]),
    d. .sel(alu_s),
    e. .s(alu_out),
    f. .co(alu_co),
    g. .zflag(alu_zflag));
3. Simulate the design with the tb_bp_dp test bench in iSim and fix any errors found.

Result:

Testing the modules inside of iSim caused no mismatch errors. In addition other things requested from the lab report can be found below.

Truth Table Describing bp_dp

ud_flags	alu_co	alu_zflag	ck	flag_reg[1]	flag_reg[0]
0	X	X	⇧	Hold	Hold
1	0	0	⇧	0	0
1	0	1	⇧	0	1
1	1	0	⇧	1	0
1	1	1	⇧	1	1

Discussion:
I felt that this lab was not so difficult except for the concatenation needed to handle the carry out bit in some of the ALU operations. I found the proper information on how to do this from a website hosted by Berkley at http://www-inst.eecs.berkeley.edu/~cs61c/resources/verilog.pdf. This alteration to the code can be seen in the section of this report that includes the Verilog source.

Source for Verilog Code:

bp_alu.v:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    14:41:34 11/17/2012
// Design Name:
// Module Name:    bp_alu
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module bp_alu(
    input [15:0] a,
    input [15:0] b,
```

```
    input ci,
    input [2:0] sel,
    output reg [15:0] s,
    output reg co,
    output reg zflag
    );



//Let's handle the select codes
always @(a,b,ci,sel) begin

        //Make the default value for the carry out to be 0
        co = 0;

        //Check all of the select codes
        case (sel)

                //Addition with carry
                3'b000: begin
                        {co,s} = a + b + ci;
                end

                //Subtraction with carry
                3'b001: begin
                        {co,s} = a - b - ci;
                end

                //Increment a
                3'b010: begin
                        {co,s} = a + 1;
                end

                //Select b with no operation
                3'b011: begin
                        s = b;
                        co = 1;
                end

                //Bitwise AND of a,b
                3'b100: begin
                        s = a & b;
                        co = 0;
```

```
                end

                //Bitwise OR of a,b
                3'b101: begin
                        s = a | b;
                        co = 0;
                end

                //Bitwise XOR of a,b
                3'b110: begin
                        s = a ^ b;
                        co = 0;
                end

                //Bitwise NOT of a
                3'b111: begin
                        s = ~a;
                        co = 0;
                end

        endcase

        //Highest priority stuff
        //case(s)
        //              1: zflag = 0;
        //              default: zflag = 1;
        //      endcase


//      Having some problems with the case statement, lets use a few ifs
//      instead.
        if (s == 0) zflag = 1;
        if (s != 0) zflag = 0;

end


endmodule
```

bp_dp.v:

```
//
// lab11 : version 11/08/2012
//
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
module bp_dp(
	input clk,
	input rf_s,
	input [3:0] rf_w_addr,
	input [1:0] rf_w_wr,
	input [3:0] rf_rq_addr,
	input [2:0] alu_s,
	input ud_flags,
	input [15:0] in_data,
	output reg [1:0] flag_reg,
	output [15:0] out_data
	);

	reg [15:0] mux_out;
	wire [15:0] alu_in_a, alu_in_b, alu_out;
	wire alu_co, alu_zflag;
	reg [1:0] next_flag_reg;

	// create mux (combinational)
	always @(rf_s, alu_out, in_data) begin
		if (rf_s == 1'b1) mux_out = in_data;
		else mux_out = alu_out;
	end

	// flag register - synchronous block
	always @(posedge clk) begin
		flag_reg <= next_flag_reg;
	end

	// combinational part of flag register
	always @(flag_reg, ud_flags, alu_co, alu_zflag) begin
		// default
		next_flag_reg = flag_reg;   // hold
		if (ud_flags == 1'b1) begin
			next_flag_reg = {alu_co, alu_zflag};
		end
	end    // end of always
```

```
        // instantiate register file
        rf_16x16_2r1w u_rf (
                .clk(clk),
                .w_data(mux_out),
                .w_addr(rf_w_addr),
                .w_wr(rf_w_wr),
                .rq_addr(rf_rq_addr),
                .rp_data(alu_in_a),
                .rq_data(alu_in_b)
        );

        // instantiate alu

        assign out_data = alu_in_a;

        bp_alu u_alu(
                .a(alu_in_a),
                .b(alu_in_b),
                .ci(flag_reg[1]),
                .sel(alu_s),
                .s(alu_out),
                .co(alu_co),
                .zflag(alu_zflag));
endmodule
```