**Basic Processor Datapath Design**

Vincent Martin

TUID: 913012274

Lab #11 (11/14/2012)

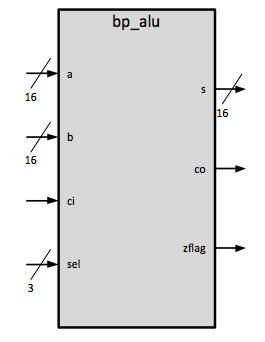
**Introduction:**

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.

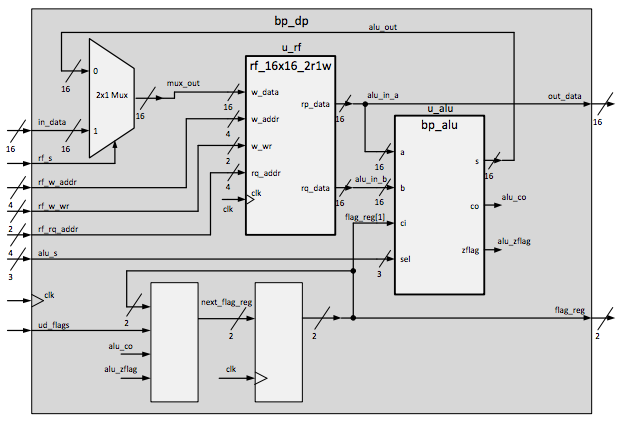
This lab will be completed in simulation mode only and will not involve physical hardware, so all testing will be completed via simulation and not on our hardware board.

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**

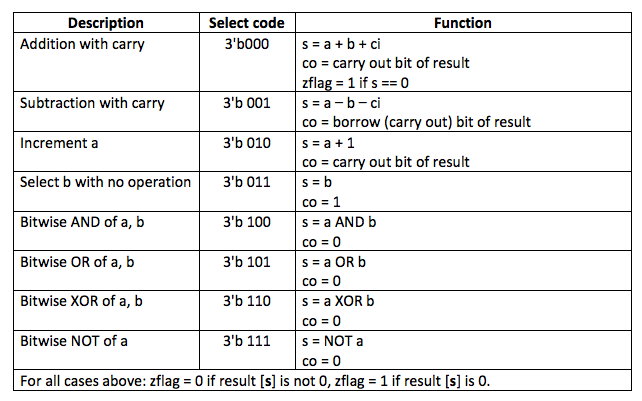
****

**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
   1. 16 bit input a
   2. 16 bit input b
   3. 1 bit input ci
   4. 3 bit input sel
   5. 16 bit output reg s
   6. 1 bit output reg co
   7. 1 bit output reg zflag
5. Design the internal operations based on the chart below.
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.
   1. .a(alu\_in\_a),
   2. .b(alu\_in\_b),
   3. .ci(flag\_reg[1]),
   4. .sel(alu\_s),
   5. .s(alu\_out),
   6. .co(alu\_co),
   7. .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*