**Basic Control Processor Control Unit Design**

Vincent Martin

TUID: 913012274

Lab #11 (11/27/2012)

**Introduction:**

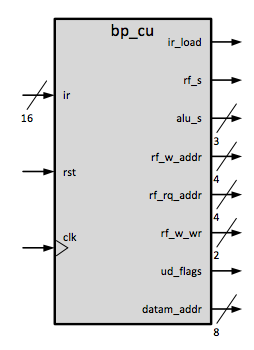
The objective of this lab is to design a control unit for a basic processor. Essentially we will be creating the control unit that will control the work that we completed in our previous lab 11.

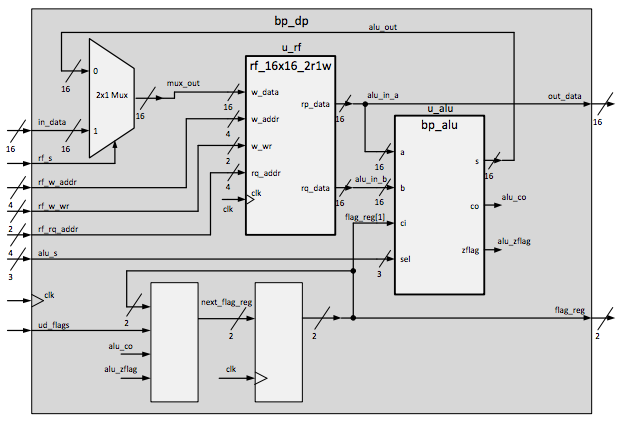
The control unit will take in a IR code and then decode that code in order to do control our register and ALU combination.

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 control unit and an example of a data path block that this control unit could feed with inputs to.

**Control Unit 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 Control Unit:

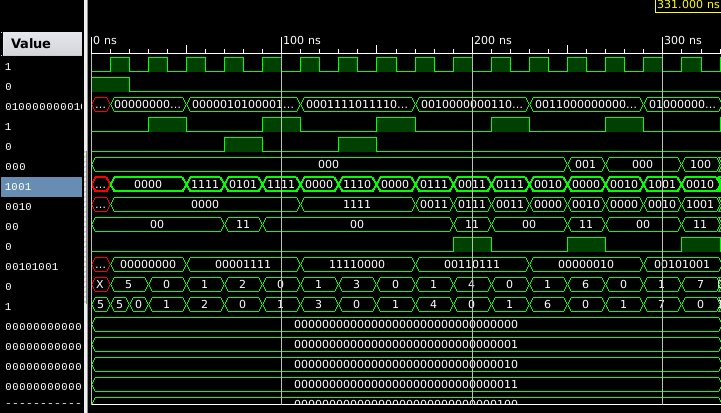
1. Connect to the development server
2. Open XISE
3. Open the lab12 project file in ~/xilinx/lab12/ folder
4. Open bp\_cu module
5. Alter the code to handle the following instructions
   1.  load data from memory – {4’b0000,r[3:0],d[7:0]}
   2.  store data to memory – {4’b0001,r[3:0],d[7:0]}
   3.  two register add (rm = rn + rm) – {8’b00100000,n[3:0],m[3:0]}
   4.  increment single register – {12’b001100000000,m[3:0]}
   5.  register move (copy) (rm = rn) – {8’b01000000,n[3:0],m[3:0]}
6. Check the syntax of your design for any errors.

Testing the Control Unit:

1. Create test vectors inside of the tb\_bp\_cu.txt file included in the lab 12 project directory.
2. Run iSim and go into the wave viewer.
3. Verify that the output of the simulation matches what you expect based on your alterations of the testing text file.

**Result:**

Testing the module inside of iSim gave me the results that I expected.



**Discussion:**

This lab was extremely difficult for me because I was not quite sure what was expected. The lack of having actual memory inside of our code confused me greatly and I did not quite understand where my data was going to be stored on certain instances of instructions.

This caused me to overcomplicate everything and made me spend more hours than needed. Stepping back and approaching it more simply was beneficial for me as it lead to completing the lab quite quickly.

The most interesting part of this lab is how I can use this as input for lab 11. This puts everything into order for me and gives me appreciation of how to design such things.

**Source for Verilog Code and Testing File:**

**Bp\_cu.v**

*//*

*// lab12 : version 11/15/2012*

*//*

*`timescale 1ns / 1ps*

*//////////////////////////////////////////////////////////////////////////////////*

*//////////////////////////////////////////////////////////////////////////////////*

*module bp\_cu(*

*input clk,*

*input rst,*

*input [15:0] ir,*

*output reg ir\_load,*

*output reg rf\_s,*

*output reg [2:0] alu\_s,*

*output reg [3:0] rf\_w\_addr,*

*output reg [3:0] rf\_rq\_addr,*

*output reg [1:0] rf\_w\_wr,*

*output reg ud\_flags,*

*output reg [7:0] datam\_addr*

*);*

*// define states*

*parameter FETCH=0, DECODE=1, LOAD=2, STORE=3, ADDR=4, IDLE=5, INCR=6;*

*parameter MOVR=7;*

*// define instruction codes for decoding*

*parameter CODE\_LOAD=4'b0000, CODE\_STORE=4'b0001, CODE\_ADDR=4'b0010;*

*parameter CODE\_INCR=4'b0011, CODE\_MOVR=4'b0100;*

*// registers for holding the state*

*reg [2:0] state, next\_state;*

*// synchronous logic*

*always @(posedge clk) begin*

*state <= next\_state;*

*end*

*// combinational logic*

*always @(rst, ir, state) begin*

*// define defaults*

*ir\_load = 0; // load the instruction register*

*rf\_s = 0; // input mux selector*

*alu\_s = 3'b000; // alu function selector*

*rf\_w\_addr = ir[3:0]; // register file write address*

*rf\_rq\_addr = ir[7:4]; // register file read port q address*

*rf\_w\_wr = 2'b00; // register file write enable (2 bytes)*

*ud\_flags = 0; // enable update flag register*

*datam\_addr = ir[7:0]; // data memory address bus*

*next\_state = state;*

*// main logic*

*case(state)*

*FETCH: begin*

*ir\_load = 1; // load the instruction register*

*next\_state = DECODE;*

*end*

*DECODE: begin*

*// decoder*

*case(ir[15:12])*

*CODE\_LOAD: next\_state = LOAD;*

*CODE\_STORE: next\_state = STORE;*

*CODE\_ADDR: next\_state = ADDR;*

*CODE\_INCR: next\_state = INCR;*

*CODE\_MOVR: next\_state = MOVR;*

*default: next\_state = IDLE;*

*endcase // end of decoding case*

*end // end of decode state*

*LOAD: begin*

*rf\_s = 1; // input mux selector*

*rf\_w\_addr = ir[11:8]; // register file write address*

*rf\_w\_wr = 2'b11; // register file write enable (2 bytes)*

*ud\_flags = 0; // enable update flag register*

*datam\_addr = ir[7:0]; // data memory address bus*

*next\_state = FETCH; // get ready for next one*

*end*

*STORE: begin*

*rf\_s = 1; // input mux selector*

*rf\_w\_addr = ir[11:8]; // register file read address*

*rf\_w\_wr = 2'b00; // register file read disable (2 bytes)*

*ud\_flags = 0; // enable update flag register*

*datam\_addr = ir[7:0]; // data memory address bus*

*next\_state = FETCH; // get ready for next one*

*end*

*ADDR: begin // add register*

*rf\_s = 0;*

*alu\_s = 3'b000;*

*rf\_w\_addr = ir[7:4];*

*rf\_rq\_addr = ir[3:0];*

*rf\_w\_wr = 2'b11;*

*ud\_flags = 1;*

*next\_state = FETCH;*

*end*

*INCR: begin // increment register*

*rf\_s = 0;*

*alu\_s = 3'b001;*

*rf\_w\_addr = ir[7:4];*

*rf\_rq\_addr = ir[3:0];*

*rf\_w\_wr = 2'b11;*

*ud\_flags = 1;*

*next\_state = FETCH;*

*end*

*MOVR: begin // register to register move*

*rf\_s = 0;*

*alu\_s = 3'b100;*

*rf\_w\_addr = ir[7:4];*

*rf\_rq\_addr = ir[3:0];*

*rf\_w\_wr = 2'b11;*

*ud\_flags = 1;*

*next\_state = FETCH;*

*end*

*IDLE: begin // no-op state - do nothing - set outputs to the defaults*

*next\_state = FETCH;*

*end*

*default: begin // no-op state - do nothing - set outputs to the*

*defaults*

*next\_state = FETCH;*

*end*

*endcase*

*// priority logic*

*if (rst == 1) begin*

*next\_state = IDLE;*

*rf\_w\_wr = 2'b00;*

*ud\_flags = 0;*

*end*

*end // end of always loop*

*endmodule*

**tb\_bp\_cu.txt**

*//*

*// lab12 : version 11/15/2012*

*//*

*// This file contains the test vectors for the basic*

*// processor control unit.*

*//*

*// This is NOT a self checking testbench. It only displays*

*// the outputs to the terminal.*

*//*

*// Columns: (0 is rightmost column)*

*// ------inputs------*

*// [16] - reset signal: rst*

*// [15:0] - instruction register : ir*

*//*

*// There is one clock cycle per test vector.*

*// The inputs are set up 1/2 cycle before ^clk.*

*// The inputs and outputs are displayed 1/2 cycle after ^clk.*

*//*

*// reset for one clock cycle*

*1\_xxxxxxxxxxxxxxxx*

*// Load data instruction - pick register 5*

*// Need to hold the IR for 3 clocks to get*

*// through the state machine.*

*0\_0000010100001111*

*0\_0000010100001111*

*0\_0000010100001111*

*// Store data instruction - pick register 14*

*0\_0001111011110000*

*0\_0001111011110000*

*0\_0001111011110000*

*// Add registers - pick 3 and 7*

*0\_0010000000110111*

*0\_0010000000110111*

*0\_0010000000110111*

*// Increment register - pick 2*

*0\_0011000000000010*

*0\_0011000000000010*

*0\_0011000000000010*

*// Move register 2 to register 9*

*0\_0100000000101001*

*0\_0100000000101001*

*0\_0100000000101001*