ENGR 304
Hardware Design Project: Week Three

Objectives:

After completing this design project, you should be able to:

- write VHDL for multiple-level state machines,
- work as a team to complete a project,
- better understand how a CPU operates and is designed, and
- combine digital elements to create more complex logic to meet system requirements.

Overview of the Design:

In this final part of your design project, you will complete the controller implementation by adding support for the remaining instructions (LDW and STW). Once the hardware is completed, your team will execute a provided final test program (the longer of the two RAM contents files provided in Moodle) to verify that your CPU works correctly.

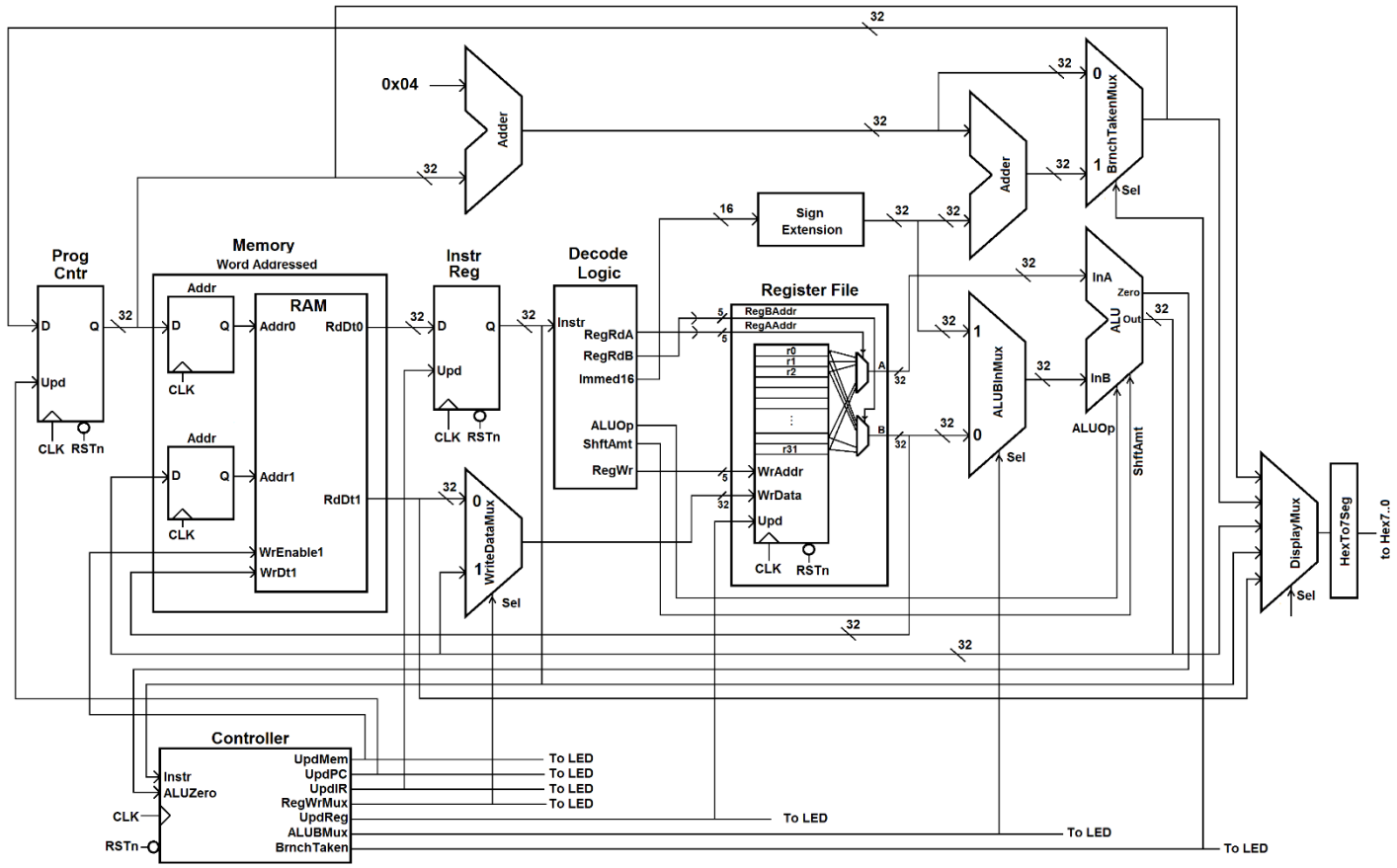

Figure 1. Processor Block Diagram

Testing for Week Three:

<u>ModelSim Implementation</u>

1. Compile your design using Quartus to ensure that there are no latch or sensitivity warnings. As described in the beginning of this project, the Quartus compiler will flag any sensitivity list and inferred latch problems in your

code that the ModelSim compiler does not. Capture ALL warnings so they can be included in your final report.

2. Using the Ram_Contents.mem file, compile your ModelSim project.

3. Using your previous do files as templates, create a *.do file that accomplishes the following for your system:
    a. Starts vsim for your VHDL system's behavior.
    b. Adds these signal waveforms: WrAddr, WrData, UpdReg, PC-Q, IR-Q, the state of your state machine, and the registers in your genRegisters component. (You may limit the registers traced to those used in the test code.)
    c. Forces the clock to cycle at a period of 20 ns and forces the reset to be asserted only for the first 15 ns.
    d. Executes the simulation for enough time so that your design will completely execute the provided test code program through executing the final branch instruction at least once. The time can be estimated based on the number of instructions executed and your state machine's average cycles per instruction.

4. Run the simulation and locate the execution of the final branch instruction in the test code program in the timing diagram. The program (specifically the instruction at location 0x0E) should write the value 0x3012_7400 to r5, and it should be followed by the program branching indefinitely to itself. Verify this value is written and that the other final register values match what is shown here. Capture that portion of the wave window and include it in your writeup.

r2	0x4567_89AB
r3	0x1357_9BDF
r4	0x0246_8ACE
r5	0x3012_7400
r6	0x0246_12B8
r12	0x3012_7400
r16	0x0000_4567

5. After completing the simulation, assess the execution time (in terms of the number of clock cycles needed) for each of the five types of instructions (R-type, branching, I-type arithmetic, LDW, and STW). Create a table that summarizes this information.

<u>Quartus Implementation</u>

While the design you used for ModelSim will *compile* in Quartus, some changes are necessary to *run* the project on the DE2 board.

- Map your RSTn signal input to the Key0 input. This will allow you to reset your system with a press of Key0.
- Map your CLK signal input to the Key3 input. A 20 ns clock period (50 MHz) is far too fast for us to observe the operation of the CPU. Mapping the clock to Key3 will let you control each clock pulse and observe the behavior of the CPU on each.
- In Week One, you were instructed to map the hex displays to the Instruction Register (IR). Implement a display multiplexer that lets you select one of the signals / registers listed below based on the inputs of SW(2..0). This will provide you with easy access to various points in the design for verification and debugging. The signals your design should support are:
    - Program Counter
    - Branch Adder output
    - ALU output
    - Instruction Register
    - Memory Output 0
    - Memory Output 1
    - Register A Output
    - Register B Output

  (Note that the "DisplayMux" in Figure 1 shows only five signals; the diagram is complex enough with these!)

- Map the outputs of your state machine to the green LEDs on the DE2 board, as shown in Figure 1. This will allow you to observe which control signals are active at various points in the program execution.
- The RAM contents files provided in Week One (.mem format) contain the code for initializing memory in ModelSim. Rather than the .mem format, however, Quartus uses the .mif format. The file Ram_Contents.mif is included in the reference files for part III of the lab; use that for your Quartus project by including it in the list of Quartus files just as you would a VHDL file.
- The VHDL file we've been using for memory so far works with ModelSim, but for Quartus we need to use a slightly different memory implementation with a different initialization file. Details are provided below. There is, however, no difference in functionality.

6. Change your clock and reset inputs from simulated signals to KEY3 and KEY0 respectively. Edit the pin assignment template file from moodle and import it prior to compiling your design.

7. Implement the display multiplexer and LEDG outputs as described earlier. Hint: use a case statement for the display multiplexer.

8. Replace the MemoryModule.vhd file in your project with the MemoryBlockQuartus.vhd file. Note these differences in the ports and memory file.
    a. The signal "CLK" is replaced by "clock."
    b. "WrDt0" and "WrEnable0" ("Write Data 0" and "Write Enable 0") signals are added in addition to the existing "WrDt1" and "WrEnable1" signals. We will not use these, but you must define signals for them in your top-level project file and tie them to zeroes. ("MemoryBlockQuartus" defines a memory module with two read and two write ports. Although we want two read ports and only one write port, Quartus does not allow us to define a memory module in this way.)
    c. Include the .mif file in your design, not the .mem file.

9. Compile your Quartus design and use the programmer tool to load it onto the DE2 Board's FPGA.

10. Use the keys and switches to execute the entire test program. Take a video that shows (on the HEX displays) the final values of the eight signals. Include a link to the video in your submission. For reference, a cycle-by-cycle list of instructions and outputs is provided. If your design doesn't work properly, you will typically see issues in the first few clock cycles. If it does work properly, you could easily cycle through all the instruction steps while paying little attention to the details. I encourage you, however, to trace through at least one or two instructions of each type step by step and understand the details of the register, bus, and control values. It will improve your understanding of this project and be valuable if/when you continue in ENGR 325 next semester.

Hand In (for Week 3):

As a **team**, submit to moodle:

A) A zip file including:
   1) Cover page with the names of the team members,
   2) Complete state diagram for the controller,
   3) Link to video demonstrating the system working,
   4) All VHDL code (properly formatted) that has been authored by your team,
   5) Table of the number of clock cycles needed for executing each of the five instruction types,
   6) Screen capture of all Quartus warnings from the Quartus compilation,
   7) Screen capture of the simulation of your design showing the values found in the registers while executing the final branch instruction, and

B) A zipped folder that contains a copy of your final design (all needed VHD files, the *.mif file, and the final .do file) so that your design can be tested independently. Please name that folder "FinalDesign" and ensure that the report PDF and final files are easy to find; you may wish to omit any autogenerated or work-in-progress files.

Individually, each team member should email your instructor with the following information: (due at the same time as the writeup of the project).

1) Names of team members, including yourself, and
2) The level of contribution each team member, including yourself, gave to the success of the project.
3) Any additional comments regarding the project, your team, or the lab overall.