Brown University
School of Engineering
EN1640 Design of Computing Systems
Professor Sherief Reda
LAB 05 (150 points) – Teamwork (groups of max 2) is allowed.
Final report due on March 25th (Milestone on March 18)

In this lab you are required to design and boot a single-cycle ARM processor. To make this lab manageable, it will be split into smaller components, where you implement an increasing sets of ARM instructions by certain due dates. You will need to submit only one lab report on the due date of March 25. However, you will need to demonstrate progress to the TA by an intermediate date of March 18.

- To verify that your processor works correctly, you need to store its execution results in the RAM data memory and then read the memory contents with the Quartus II tool after you run and are done with the program. Guideline is at the end of this assignment.
- It is probably a very good investment of time to learn how SignalTap works. Guideline at the end of the assignment and tutorial is at the class webpage.
- You probably want to create your CPU module with a clock input and address/data/control busses to the external instruction ROM or variables that hold your instructions. That way you can easily instantiate a testbench and connect it to your Verilog code for easy simulation in ModelSim. You might also do the same with the data memory though it is trickier; it is up to you!
- Make sure to go through all the guidelines at the end of this assignment before working on it.
- There are 20 points allocated based on your actual processor speed on the chip (20 ≥ 100 MHz, 19 ≥ 95, 18 ≥ 90, 17 ≥ 85, 16 ≥ 80, 15 ≥ 75, 13 ≥ 70, 11 ≥ 65, 9 ≥ 60, 7 ≥ 55, 5 ≤ 50.

(a) Implement processor with the following instructions: **`MOV, STR, ADD, SUB, AND, ORR, MUL, LDR.`** Validate the design by booting the processor and running the following two programs. Note that you can use the first program to test your processor after implementing the first two instructions (MOV, STR). Make sure to create ModelSim testbenches and verify your code with it first before you verify it on the board. You can then continue expanding the instruction set and eventually test with the second program. Read the memory contents after execution is finished. Contrast the memory contents with the results you would get from the ARM simulator. Demonstrate the experiment to the TA by Friday 21th of March. (Let's the opcode 0xFFFFFFFF for HALT and you can get the machine code of these programs using the ARMSim# tool)

Program 1:

```
MOV R1, #0
MOV R2, #16
MOV R3, #100
MOV R4, #8
```

```
ADD R5, R1, R2
ADD R6, R3, R4
STR R5, [R1]
STR R6, [R1, R2]
HALT
```

Program 2:

```
MOV    R0, #8
MOV    R1, #15
STR    R1, [R0]
MOV    R2, #0
LDR    R2, [R2, #8]
ADD    R2, R1, R0
SUB    R3. R1, R0
MUL    R4, R2, R3
ADD    R0, R0, #4
LDR    R2, [R0, #-4]
SUB    R2, R1, R2
STR    R2, [R0]
HALT
```

(b) Implement processor with the following instructions together with the ability to **execute all instructions conditionally:** MOV, STR, ADD, SUB, AND, ORR, MUL, LDR, **B, BL, CMP.**

Validate the design by booting the processor and running the following code which calculates the factorial code. Make sure to initialize the SP to the end of your memory. Make sure to create ModelSim testbenches and verify your code with it first before you verify it on the board. Read the memory contents after execution is finished. Contrast the memory contents with the results you get from the ARM simulator. Demonstrate the experiment to the TA by Friday March 25th.

```
      MOV          R0, #6
      BL           FACT
      MOV          R1, #0
      STR          R0,[R1]
      HALT
FACT: CMP          R0, #1
      MOVEQ        PC, LR
      SUB          SP, SP, #8
      STR          LR, [SP, #4]
      STR          R0, [SP]
      SUB          R0, R0, #1
      BL           FACT
      LDR          R1, [SP]
      LDR          LR, [SP, #4]
      ADD          SP, SP, #8
      MUL          R0, R1, R0
      MOV          PC, LR
```

In addition to the factorial program, please validate your design using another meaningful piece of code from the your Lab03 programs that include the direct use of a conditional branch (e.g., BNE or BEQ).

Use part (b) as the reference for the following requirements of your report:

Note: For your final working version of your design, please make sure to remove debugging circuitry (e.g., for memory-content editor or signal tap) to reduce your footprint and improve your timing. You might also like to explore the Quartus II tool settings, which can adjust the strength of circuit optimization and trade-off design area with timing.

1. Include the assembly and machine code of the factorial program and your program of choice in the documentation.

2. Include your Verilog testbenches in ModelSim and show representative outputs from ModelSim (e.g., using $display outputs or waveforms) that verify functionality.

3. Print screenshots that shows the memory contents after executing the factorial program and your program of choice.

4. Use the TimingQuest tool to analyze the timing in your design. Report the critical path delay in your design and predicted Fmax. Use the tool to (1) locate the critical path in your floorplan and print the annotated critical in the floorplan view, and (2) estimate the delay breakdown among the different stages of execution (e.g., fetch, decode, execution, memory and write back). Please read the tutorial on TimingQuest analysis tool to understand the operation of this tool. A tutorial is distributed in class and also available on the class web page.

5. Using the actual board, find the actual maximum frequency that your processor can sustain without producing incorrect results. You need to keep on incrementing the frequency of the design, and re-running your experiments. Once the processors fails in booting, report the lowest frequency in which such failure occurs. Contrast the actual maximum frequency to the estimated Fmax from the TimingQuest tool. If there are differences between the predicted and actual maximum frequency, explain potential reasons for these discrepancies.

6. Print and annotate the floorplan of your design. Report the resources being used by your processor: LEs (combinational and dedicated registers), PLL, embedded multipliers, memory blocks, and routing resources. Make sure to remove

Guidelines for Creation of Instruction and Data Memories:

Ideally the instruction memory should be built out of the ROM component. Unfortunately, the ROM component in the FPGA cannot be read combinationally; i.e., the output will not update its value until the positive edge of the clock comes in. Because the ROM address is

supplied directly from the PC register, it will not be possible to update the PC and fetch the instruction in the same cycle. To avoid this problem, I suggest creating the ROM directly using an array of 32-bit registers in your code and initialize the registers within your code using the `initial` statement.

For the data memory, you should initialize the RAM blocks using the M4K blocks using the MegaWizard manager. Make sure the output port is not registered; however, there is no way to avoid that the inputs are not registered (same problem as in ROM). To fix the situation in this case, I suggest clocking the RAM with the inverse of the clock signal. This will give the processor half a cycle to fetch and execute the instruction, and another half a cycle to access the memory and loading its contents into register. It is not an ideal situation. If you have other suggestions, feel free to suggest them.

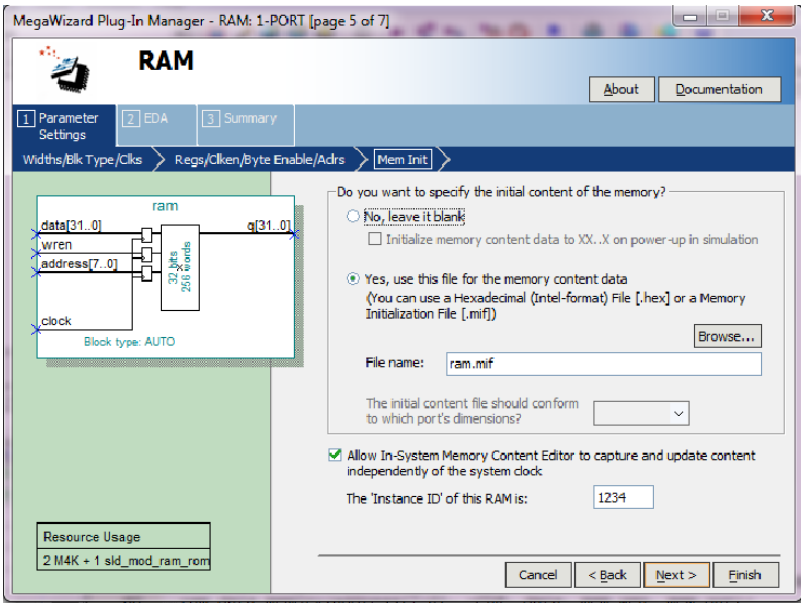
Guidelines for Debugging:

You are likely to encounter many bugs in your design. To help in debugging your code during runtime, you should learn how to use the Signal Tap tool. The Signal Tap tool inserts additional circuitries in your design to allow you monitor the activities of various wires during runtime through the Quartus II tool. The tool is very powerful for debugging. I have distributed a tutorial in class and the tutorial is also available on the class web page. The Signal Tap tool is very powerful and sufficient for debugging your code.

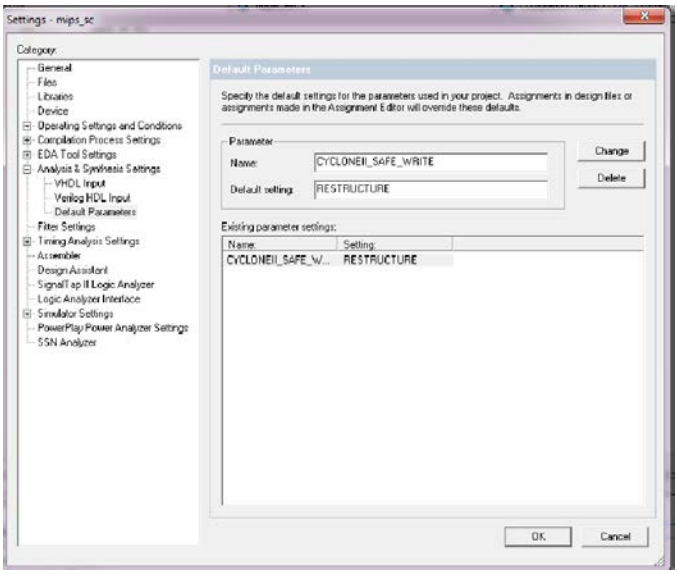
Guidelines for reading back RAM memory:

To test the correctness of your code, you will need to read back the contents of the RAM blocks after you are done with executing your code. The Quartus II tool enables you to read back the contents of memory at any time after programming and during operation using the In-System Memory Content Editor tool. You need to carry out the following steps to enable such reading.

1. When you instantiate RAM, you will need to enable “Allow In-System Memory Content Editor to capture and update content independently of the system clock” as indicated in the next figure



2. Before you can use the In-System Memory Content Editor tool, one additional setting has to be made. In the Quartus II software select Assignments > Settings to open the window, and then open the item called Default Parameters under Analysis



and Synthesis Settings. As shown in the figure, type the parameter name CYCLONEII_SAFE_WRITE and assign the value RESTRUCTURE. This parameter allows the Quartus II synthesis tools to modify the single-port RAM as needed to allow reading and writing of the memory by the In-System Memory Content Editor tool. Click OK to exit from the Settings window.

3. After your programming your design, you need to launch the “In-System Memory Content Editor” which can be access from the Tools menu. Note that the memory content editor can be used to program your design. Select the memory in Instance manager and click the read data from system memory button (one with a red box) as shown in next figure. Now you can see the contents in the memory.