CSC 244: Digital Logic
Project 2: 10-bit Processor with Immediate Instructions

1 Introduction

ARM-TI has hired your company, Super State machines, LLC., to design their next computer processor. Your manager was so impressed with your vending machine design that they are tasking you to design, describe, and build a 10-bit processor using ARM-TI's specifications. The project requirements are outlined below.

1.1 Additional Course Requirements

Your 10-bit processor will be designed using SystemVerilog (SV) and demonstrated on a DE10-Lite board. Your top-level SV module should only be structural SV, connecting together the DE10-Lite inputs and outputs with your submodules (e.g., see Fig. 1). You must work with a single partner (**teams of exactly two**). Choose your partner wisely and split the work evenly (for example, it is probably not a good idea for one person to design and build the project, and the other to write the report). You must e-mail Mr. Galipeau your group before Monday, October 30th. You will submit your deliverables to a D2L dropbox, one set per group.

2 Deliverables and Deadlines

Item	**Due Date**
E-mail Mr. Galipeau your group	Monday, Oct 30th
System Verilog Completed, Tested, and Submitted	11:59 pm Thursday Nov 30th
Project Demonstration	Schedule with Shadrack, Logan, Cole, or Amir On or Before Thursday Nov 30th
Final Design Report	11:59 pm on Tuesday, Dec 5th

3 Project Description

3.1 Processor Overview

You will build the processor shown in Fig. 1 with the set of instructions in Section 3.2. Your data bus will be 10-bits wide, and is used to pass data between elements of your processor (shown in green in Fig. 1). A general description of the processor will be provided here, with each of the top-level modules and inputs/outputs described in more detail below.

Your 10-bit processor takes in data and instructions via an external source (labeled "data"). Depending on the current instruction (saved in the instruction register) and the current timestep (saved in a 2-bit counter), your `controller` drives the control signals of your multi-stage processor circuit. All data is saved in one of four 10-bit registers (R0–R3) within the register file. To perform arithmetic and logic operations on the stored data, a multi-stage arithmetic logic unit (ALU) is used that calculates an output $G = A$ FN OP, where "FN" is the arithmetic or logic operation to be performed, A and G are intermediate input and output registers, respectively, and OP and RES are connected to the shared data bus as the operand input and result output, respectively.

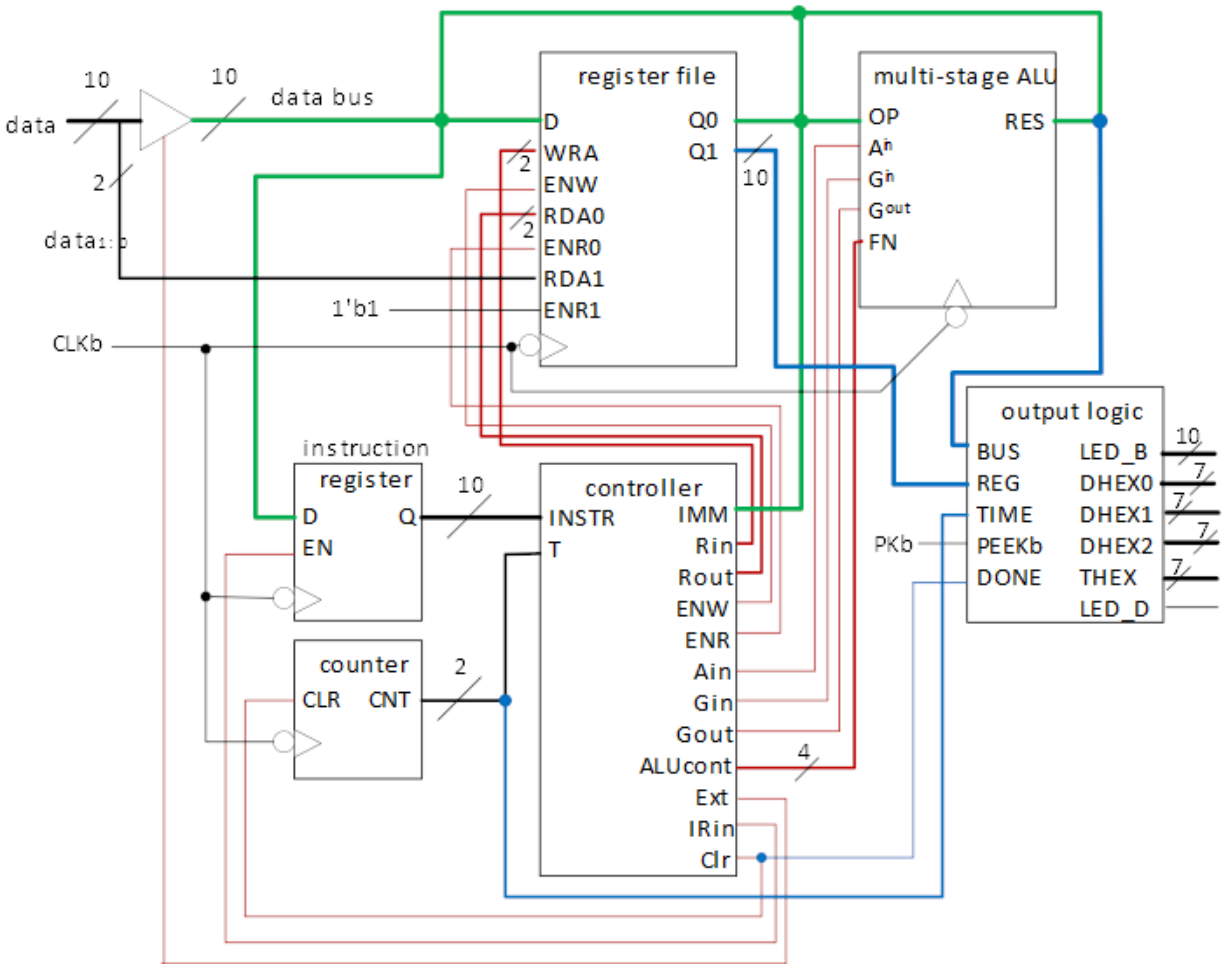

Figure 1: Required top-level structural view of the 10-bit processor with shared data bus. **Do not directly use this figure in your report!** The shared data bus is highlighted in green, control signals in red, and output signals in blue.

The 10-bit processor is *similar to* that discussed in the supplementary PDF written by Brown and Vranesic (old textbook) and that we discussed in class. The PDF contains many hints and SystemVerilog samples beyond those discussed in class.

3.2 Instruction Set

You will implement, and your processor will support, the following functions using the exact instruction formats in Table 1. Some instructions use the ALU, some need just one operand, some are just for data movement, and some use *immediate* values as operands. ARM-TI is leaving you the design decision to determine how best to decode each instruction, but they expect the minimum number of timesteps required per instruction type (**hint:** if you only need one operand, you *might* (read: **must**) be able to *skip* an ALU step [wink, wink]).

Table 1: Processor instruction set

Opcode	Mnemonic	Instruction
00_XX_UUU_0000	ld Rx	Load data into Rx from the slide switches (external data input): Rx ← Data
00_XX_YY_0001	cp Rx, Ry	Copy the value from Ry and store to Rx: Rx ← [Ry]
00_XX_YY_0010	add Rx, Ry	Add the values in Rx and Ry and store the result in Rx: Rx ← [Rx] + [Ry]
00_XX_YY_0011	sub Rx, Ry	Subtract the value in Ry from Rx and store the result in Rx: Rx ← [Rx]−[Ry]
00_XX_YY_0100	inv Rx, Ry	Take the twos-complement of the value in Ry and store to Rx: Rx ← −[Ry]
00_XX_YY_0101	flp Rx, Ry	Flip the bits of the value in Ry and store to Rx: Rx ←∼ [*Ry*]
00_XX_YY_0110	and Rx, Ry	Bit-wise AND the values in Rx and Ry and store the result in Rx: Rx ← [Rx] & [Ry]
00_XX_YY_0111	or Rx, Ry	Bit-wise OR the values in Rx and Ry and store the result in Rx: Rx ← [Rx] | [Ry]
00_XX_YY_1000	xor Rx, Ry	Bit-wise XOR the values in Rx and Ry and stroe the result in Rx: ← [Rx] ∧ [Ry]
00_XX_YY_1001	lsl Rx, Ry	Logical shift left the value in Rx by Ry and store the result in Rx: Rx ← [Rx] <<[Ry]
00_XX_YY_1010	lsr Rx, Ry	Logical shift right the value in Rx by Ry and store the result in Rx: Rx ← [Rx] >>[Ry]
00_XX_YY_1011	asr Rx, Ry	Arithmatic shift right the value of Rx by Ry and store the result in Rx: ← [Rx] >>>[Ry]
10_XX_IIIIII	addi Rx, 6'bIIIIII	Add the 6-bit *immediate* value 10'b0000IIIIII (left-padded with zeros) to the value in Rx and store in Rx: Rx ← [Rx] + `10'b0000IIIIII`
11_XX_IIIIII	subi Rx, 6'bIIIIII	Subtract the 6-bit *immediate* value 10'b0000IIIIII (left-padded with zeros) from the value in Rx and store in Rx: Rx ← [Rx] - `10'b0000IIIIII`

3.3 Hardware Modules

3.3.1 Register File

```
module registerFile(
        input logic [9:0] D,
        input logic ENW, ENR0, ENR1, CLKb,
        input logic [1:0] WRA, RDA0, RDA1,
        output logic [9:0] Q0, Q1
);
```

Similar to the in-class example, the register for this processor will contain eight 10-bit registers. Each register shares a common 10-bit input (D), and share two common 10-bit data outputs (Q0 and Q1). To determine which register (if any) should be saving the input data and driving the output signals, there are three 3-bit address inputs and three enable inputs. The operation of the

register file is as follows:

Write-operation: On the active-edge of the *debounced* clock signal CLKb, if ENW is active, then the register associated with the write address WRA saves the value on the D input.

Read-operation: The read operation of the register file is combinational and **does not depend on the active-edge of the clock**. For each read signal (Q0 or Q1), if ENR is active, the output Q is equal to the value stored in the register associated with read address RDA. If not enabled, the register file should disconnect its outputs to avoid contention (i.e., high-impedance 'Z'). **Warning:** you should ***NOT*** have an additional 8 registers to implement the second read port; this will cost you points. Use good engineering design to be able to add a second read port to your design.

3.3.2 Instruction Register

```
module reg10(
        input logic [9:0] D,
        input logic EN, CLKb,
        output logic [9:0] Q
);
```

The 10-bit instruction register is negative-edge triggered with synchronous active-high enable. This register is used to save the instruction at timestep 0, and maintain the instruction throughout the multiple clock cycles required to complete a given instruction.

3.3.3 Timestep Counter

```
module upcount2(
        input logic CLR, CLKb,
        output logic [1:0] CNT
);
```

A negative-edge triggered 2-bit up-counter with active-high synchronous clear is used to keep track of the timestep of the current instruction.

3.3.4 Multi-stage ALU

```
module ALU(
        input logic [9:0] OP,
        input logic [3:0] FN,
        input logic Ain, Gin, Gout, CLKb,
        output logic [9:0] Q
);
```

The multi-stage ALU shown in Fig. 2, which performs the useful arithmetic operations on data held in the registers, is very similar to Lab 7. The ALU takes one common operand through the OP input. To perform an arithmetic or logic operation on two inputs, an 'A' register is used to stage the inputs. The ALU always outputs (combinational logic) A FN OP, where FN is one of the operations in the instruction set specified by ALUcont. Two additional control signals determine when the result should be saved (Gin) and when the result should output (Gout). Both of the registers operate on a synchronized negative-edge triggered clock signal.

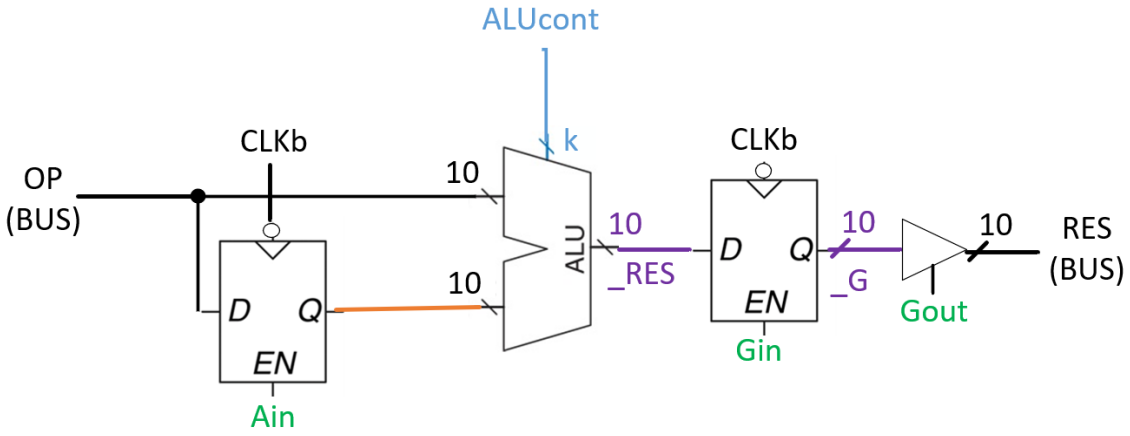

Figure 2: Multi-stage ALU with staged A input and G output.

3.3.5 Processor Controller

The 'brains' of the processor, and perhaps the most challenging aspect of the design process, is the controller. The controller needs to determine the output of each of the control signals for the processor (shown as red in Fig. 1). The list of outputs, given in Table 2, are combinational logic that depend on the current timestep T and the current instruction INSTR (from the list in Table 1). It is suggested that a time-instruction table similar to that in class is created, and behavioral or procedural logic is derived.

Table 2: Processor Control Signal Outputs

Signal	**Bit-width**	**Description**
IMM	10	*Immediate* value to be used for the two immediate instructions
Rin	2	Address for the register to be written to, from the shared data bus
Rout	2	Address for the register to be read from, to the shared data bus
ENW	1	Enable signal to write data to the register file
ENR	1	Enable signal to read data from the register file
Ain	1	Enable signal to save data to the intermediate ALU input "A"
Gin	1	Enable signal to save data to the intermediate ALU output "G"
Gout	1	Enable signal to write data from the ALU intermediate output "G" to the shared data bus
ALUcont	4	Signal to control which arithmetic or logic operation the ALU should perform
Ext	1	Enable signal to drive the shared data bus from the external "data" signal
IRin	1	Enable signal to save data to the instruction register
Clr	1	Clear signal for the timestep counter

3.3.6 Output Logic

In a normal processor, the logic signals remain invisible to the user. For testing and demonstration purposes, ARM-TI has requested a special combinational output logic block that can probe the data within the processor (shown as the blue signals in Fig. 1). The output logic takes as inputs the current data on the shared bus (BUS), the output of the second read port on the register file (REG), and the current timestep of the processor (TIME). Based on these inputs, **the following outputs are always shown:** LED_B shows the current values on the data bus, and THEX shows the current timestep decoded to a 7-segment display.

The DHEX2:0 outputs change depending on the value of PEEKb, which can be used to "peek" into a register. If PEEKb is a logic-1, DHEX2:0 shows the current 10-bit value on the data bus decoded to three 7-segment displays. If PEEKb is a logic-0, DHEX2:0 shows the 10-bit output of the second read port of the register file. This can be used to "peek" into each of the eight registers using the three least significant bits of the "data" signal (RDA1).

One final input/output of the output logic block takes the "Clr" signal from the controller. If "Clr" is a logic-1, an LED_DONE output should be active to indicate the current instruction has completed.

3.4 DE10-Lite Implementation

Your 10-bit processor will be implemented on your DE10-Lite board using SystemVerilog HDL to create a modular design. You will implement your project using whichever SystemVerilog you wish, except your top-level module **must only include structural SV**, connecting the DE10-Lite I/O to your submodules (you must at minimum have those listed above). The required input and output assignments for your DE10-Lite board are given in Tables 3 and 4, respectively. **You must debounce the signals from the KEY1:0 inputs (not shown in the figure, add these to your report figure).**

Table 3: Required DE10-Lite Inputs

Signal	**DE10-Lite I/O**
Data Inputs	SW[9:0]
CLKb	KEY0
PKb	KEY1

Table 4: Required DE10-Lite Outputs

Signal	**DE10-Lite I/O**
Data Bus (LED_B)	LEDR[9:0]
Data Bus/Register Peek (DHEX2:0)	HEX2:0
Current Timestep (THEX)	HEX5
Done Signal	DP on HEX5

4 Example Project Timeline

Although the processor does not need to be demonstrated until the end of the semester, it is a good idea to get started early and make weekly progress. An example of a project management plan is provided in Table 5. **I highly suggest you and your partner make and follow a similar plan.** You should test each module *individually* to determine that it works before you connect them all together (i.e., system integration, similar to the testing plan from Project 1). It is significantly harder to debug a system if you have not tested individual components.

4.1 Project Demonstration

You shall implement your project using SystemVerilog on your DE-10 Lite FPGA board using the I/O listed above. Mr. Galipeau will give you a list of *mnemonic* instructions that you will need to demonstrate and prove to Mr. Galipeau your processor is working. You are expected to use the minimum timesteps required for each instruction (e.g., load and move should only take two timesteps, other instructions may need more). An example mnemonic program (i.e., assembly language) is provided below.

```
ld R0, 10'h01F
ld R1, 10'h3E0
mv R3, R1
or R3, R0
andi R0, 5'b01001
```

You are to demonstrate your project working and tested *before* 5:00 PM on Thursday, Nov 30th. Your tested and fully functional SystemVerilog code should be submitted via D2L Dropbox prior to that time, one per group, one .txt file per module, **not .zip or .sv**. Any projects that are not demonstrated on or before Thursday, Nov 30th, are considered late. If you are unable to demo during lab time that week, *you* will need to schedule a time to demonstrate your project to Amir, Cole, Shadrack, Logan, or Mr. Galipeau. All demos must be completed by 5:00 PM on Friday, Dec. 1st, to receive credit. **Do not assume that you will be able to demonstrate to Mr. Galipeau at 5:00 PM on Friday, Dec. 1st.**

Table 5: Example Project Timeline

Date Completed and/or Tested	**Processor Item(s) Completed**
Nov. 2	Find team and send Mr. Galipeau e-mail
Nov. 10	Control Table for Instruction/Timestep; Register File; Output Logic
Nov. 17	ALU; instruction register, counter, controller
Nov. 21	System Integration, Debugging, and Testing
Nov. 29	Project Demonstration
Dec 6	Project Report

5 Final Report

Each **team** will submit a final video report by 11:59 pm on Tuesday, Dec 5th via D2L Dropbox. This report should include everything I might want to know about your project and the steps you took in designing and building it (according to the EECS department guidelines and the project rubric). Include all information pertaining to the design, and more — whatever you need to complete your design or need to *fully document* your **circuit** using your chosen approach. Your diagram should represent **your** circuit, including naming, not **Mr. Galipeau's** generic processor circuit. Circuit diagrams included in the lab report should be created using a software tool, such as Visio or powerpoint. Screenshots of the RTL schematics provided by Altera are *not* adequate for this report. Camera pictures of hand drawn figures are **not acceptable** in a technical report. Again, your report should discuss the **hardware** (that is, the *circuit* you designed, not the SystemVerilog HDL you wrote).

You are required to compare and contrast your project with several others. At the minimum, this should include the size of the project (as the number of logic elements) and the maximum clock rate. Please share this info concerning your project with other teams that ask. Include a section in the conclusions on how you could improve your project if you had another week to work on it (e.g., can you make it faster or more efficient?). This is your chance to sell me on your project, so mention any extra features, or anything you feel is extraordinary about your project. Make sure that you **justify any and all assumptions that you made**. Also make sure that you use this opportunity to **cite any and all sources that you used, including the text book and the web links Mr. Galipeau has provided.** If you used code from somewhere else, (such as the code Mr. Galipeau wrote in class), properly document it! Remember that you need to **describe the hardware** you used; **Do not write a report that only discusses the HDL constructs.**

Project 2 Rubric – 10-bit Processor

Student Names:

score	possible	area
	45	Working Project Demo (basic functionality required for any further credit, points may be deducted for sub-optimal or out of spec implementations), **On-time** team selection, SystemVerilog, and demonstration. **Followed instructions.**
	5	Abstract and Introduction
	25	Theory, Procedure, Results, and Analysis
	5	Conclusions
	5	Comparison to another group (size (space), timing, design decisions) and discussion of these comparisons
	15	Overall Quality of Report, Format, Spelling, Grammar, and quality of figures, equations, etc. **Note: a screenshot of the RTL viewer is NOT a quality diagram.**
	100	**Total**