EE 231L Lab 2

Design and Implementation of Combinational Circuits

Part 1. The Majority Circuit

1. Build the majority circuit you designed in the pre-lab using HCMOS logic chips.
2. Test your circuit with your logic probe, and confirm that it functions for all possible input combinations. Have your lab instructor or TA verify the circuit works.

Part 2. Majority Circuit in Altera

1. Program the majority circuit in Altera using a Graphics Design File.
2. Program the majority circuit in Altera using a Text Design File.
3. Simulate the circuit with Altera's waveform editor.
4. Test your circuit with your logic probe, and confirm that it functions for all possible input combinations. Have your lab instructor or TA verify the circuit works.

Part 3. Arithmetic Logic Unit

The heart of every computer is an Arithmetic Logic Unit (ALU). This is the part of the computer which performs arithmetic operations on numbers, e.g. addition, subtraction, etc. Here you will use the Altera language to implement an ALU having 10 functions.

ALU Operations

Your ALU will perform 10 functions on two 8-bit inputs. Later on this ALU will be one component of the computer you build in the final lab. At that time the ALU inputs will be from the `DATA` line and from `ACCA` (Accumulator A). To help make the transition to the computer, you should call the inputs `DATA[7..0]` and `ACCA[7..0]`. These inputs could represent either unsigned numbers, two's complement numbers, or simply non-numeric bit patterns. The ALU will generate an 8-bit result (`A`) and a one bit carry (`C`). To select which of the 10 functions to implement you will use `ALU_CTL` as selection lines. You will decide which combination of bits in the selection lines `ALU_CTL` correspond to for each instruction. The 10 functions are described in Table 1.

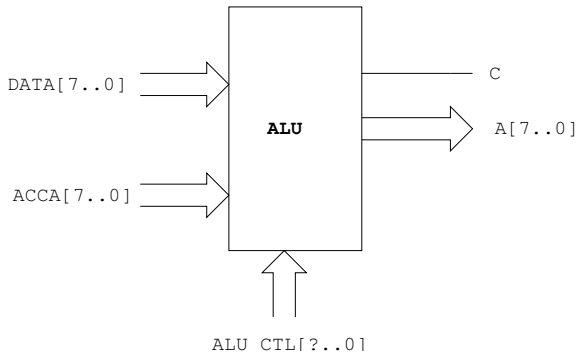

Figure 1. ALU block diagram.

It is up to you to determine how many control lines are necessary to select the ten different functions.

Table 1. ALU Functions.

ALU_CTL	Mnemonic	Description
	Load (load into A)	DATA => A: Output = DATA input C is a don't care
	ADDA (add and store into A)	ACCA+DATA => A: Add DATA and ACCA C is carry from addition
	SUBA (subtract and store into A)	ACCA-DATA => A: Subtract DATA from ACCA C is borrow from subtraction
	ANDA (logical AND)	ACCA & DATA => A: Logical AND C is a don't care
	ORAA (logical OR)	ACCA # DATA => A: Logical OR C is a don't care
	COMA (complement)	$\overline{\text{ACCA}}$ => A: One's complement of ACCA 1 => C
	INCA (increment)	ACCA + 1 => A: Add one to the value in ACCA C is a don't care
	LSRA (logical shift right)	Shift all bits of ACCA one place to the right 0 => A[7], ACCA[0] => C
	LSLA (logical shift left)	Shift all bits of ACCA one place to the left 0 => A[0], ACCA[7] => C
	ASRA (arithmetic shift right)	Shift all bits of ACCA one place to the right ACCA[7] => A[7], ACCA[0] => C

1. Design your ALU using Altera. Use a Text Design File. Be certain to deal with any unused bit combinations of the ALU_CTL lines in your Altera program. If for any reason ALU_CTL should have an undefined bit pattern on its lines during operation you should know what output will be produced.

2. Simulate the ALU using the Altera simulator. Test multiple combinations of DATA and ACCA. Choose test values that will test all possibilities for the carry bit.

3. Program your ALU code into you EPM7064. Verify that it works, using the test data from your simulation.

4. Make your code into an Altera function called ALU. Verify that you can call this function from another Altera TDF program.