EE 231L Lab 2

Design and Implementation of Combinational Circuits

Part 1. The Decoder Circuit

1. Build the decoder 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. Decoder Circuit in Altera

1. Program the decoder circuit in Altera using a Graphics Design File.
2. Program the decoder 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 11 functions.

ALU Operations

Your ALU will perform 11 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` bus, `ACCA` (Accumulator A) and `X` (X register). To help make the transition to the computer, you should call the inputs `DATA[7..0]`, `ACCA[7..0]`, and `X[7..0]`. These inputs could represent either unsigned numbers, two's complement numbers, or non-numeric bit patterns. The ALU will generate an 8-bit result (`result`), a one bit carry (`C`) and a one-bit zero bit (`Z`). To select which of the 11 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 11 functions are described in Table 1.

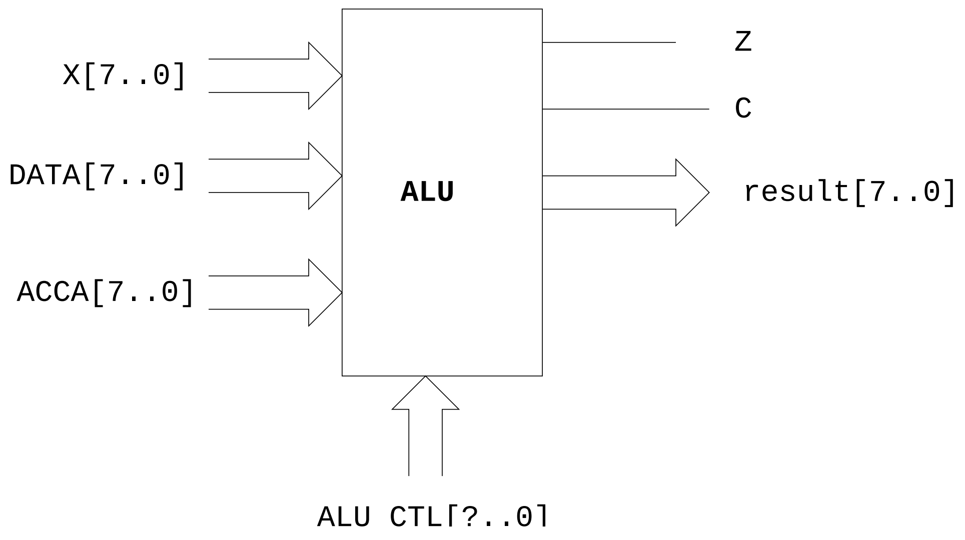

Figure 1. ALU block diagram.

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 DATA into result)	DATA => result: Output = DATA input C is a don't care 1 -> Z if result === 0, 0 -> Z otherwise
	ADDA (add)	ACCA+DATA => result: Add DATA and ACCA C is carry from addition 1 -> Z if result === 0, 0 -> Z otherwise
	SUBA (subtract)	ACCA-DATA => result: Subtract DATA from ACCA C is borrow from subtraction 1 -> Z if result === 0, 0 -> Z otherwise
	ANDA (logical AND)	ACCA & DATA => result: Logical AND C is a don't care 1 -> Z if result === 0, 0 -> Z otherwise
	ORAA (logical OR)	ACCA # DATA => result: Logical OR C is a don't care 1 -> Z if result === 0, 0 -> Z otherwise
	COMA (complement)	$\overline{\text{ACCA}}$ => result: One's complement of ACCA 1 => C 1 -> Z if result === 0, 0 -> Z otherwise
	INCA (increment)	ACCA + 1 => result: Add one to the value in ACCA C is a don't care 1 -> Z if result === 0, 0 -> Z otherwise
	LSRA (logical shift right)	Shift all bits of ACCA one place to the right 0 => result[7], ACCA[7..1] -> result[6..0], ACCA[0] => C 1 -> Z if result === 0, 0 -> Z otherwise
	LSLA (logical shift left)	Shift all bits of ACCA one place to the left 0 => result[0], ACCA[6..0] -> result[7..1], ACCA[7] => C 1 -> Z if result === 0, 0 -> Z otherwise

	ASRA
	(arithmetic shift right)
		Shift all bits of ACCA one place to the right
		ACCA[0] => result[7], ACCA[7..1] -> result[6..0],
		ACCA[0] => C
		1 -> Z if result === 0, 0 -> Z otherwise

	CPX (compare X to DATA)	X-DATA => result: Subtract DATA from X C is borrow from subtraction 1 -> Z if result === 0, 0 -> Z otherwise

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, ACCA and X. Choose test values that will test all possibilities for the carry and zero bits.

3. Program your ALU code into you EPF6016. 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.