CSCE 613
Lab 6
Course Project: Design, Verification, and Layout of Accumulator-Based ALU
Due Date: Finals Week

Introduction

Lab 4 is the course project. For this project, you will work in teams (for undergrads) or individually (for grads) to build your first large-scale VLSI design. To complete this project, you may use the cell library developed by any team member in order to achieve synthesis runs with the best performance. The goal of this project is to design a 16-bit accumulator-based (DSP-like) ALU.

Your ALU must have a top-level interface as shown in Figure 1 and as described in Table 1.

Figure 1: Interface

Pin Name	Direction	Width	Description
rst	in	1	active-high synchronous reset
clk	in	1	rising-edge clock signal
valid	in	1	active-high signal that will cause **in** and **op** inputs to be internally latched on the rising-edge of the clock whenever the **busy** output signal is not asserted
in	in	16	16-bit input
op	in	4	operation code (see Table 3)
acc	out	32	current value of accumulator – valid on falling-edge of **busy**
busy	out	1	active-high busy signal, indicates that ALU is busy performing an operation

Table 1: Interface Description

Your ALU must implement the 13 operations described in Table 2.

Name	Operation	Description
NOT	ACC[15 downto 0]=NOT(ACC[15 downto 0])	bit-wise NOT (compliment)
AND	ACC[15 downto 0]=ACC[15 downto 0] AND IN	bit-wise AND
OR	ACC[15 downto 0]=ACC[15 downto 0] OR IN	bit-wise OR
XOR	ACC[15 downto 0]=ACC[15 downto 0] XOR IN	bit-wise XOR
ADD	ACC[15 downto 0]=ACC[15 downto 0] + IN	signed addition
SUB	ACC[15 downto 0]=ACC[15 downto 0] – IN	signed substraction
MULT	ACC[31 downto 0]=ACC[15 downto 0] * IN	unsigned multiplication
DIV	ACC[31 downto 16]=ACC MOD IN ACC[15 downto 0]=ACC / IN	unsigned division
SLL	ACC=ACC SLL IN	shift left logical
SRL	ACC=ACC SRL IN	shift right logical
SRA	ACC=ACC SRA IN	shift right arithmetic
ROL	ACC=ACC ROL IN	rotate left
ROR	ACC=ACC ROR IN	rotate right

Table 2: List of Operations

The operation encoding is shown in Table 3.

Name	Op(3 downto 0)
NOT	0000
AND	0001
OR	0010
XOR	0011
ADD	0100
SUB	0101
MULT	0110
DIV	0111
SLL	1000
SRL	1001
SRA	1010
ROL	1011
ROR	1100

Table 3: Operation Code Encoding

Implementation Requirements

The ALU must be capable of operating at a clock speed of 20 MHz (50 ns clock period). However, you are not required to meet any requirements for number of cycles per operation. The layout must fit within the core area of a MOSIS "tiny chip," which is a 900 μm x 900 μm square.

Subtraction Implementation

The tutorial guided you through the design of a 32-bit carry-lookahead adder. You may use a similar design for the 16-bit adder required for this design. However, in order to implement subtraction, you will need to add additional logic around the adder.

First, recall that when using two's complement integer representation, the sign of a value may be changed by taking the complement of a value and adding one.

$$-A = \overline{A} + 1$$

Also recall that subtraction is equivalent to adding one value to the negated form of another value.

$$A - B = A + (-B)$$

Finally, recall that addition is an associative operation.

$$A + (\overline{B} + 1) = (A + \overline{B}) + 1$$

Therefore, you can easily turn an adder into a subtractor by inverting the B input then adding one to the sum. This may easily be performed using a single adder by taking advantage of the "carry in" input to the adder.

Multiplier Implementation

For this project, you may implement a multiple-cycle multiplier and divider. The easiest way to do this is to design an add-then-shift multiplier and a subtract-then-shift divider.

The multiplier design is control logic is shown in Figure 2. Your design will multiply a 16-bit *multiplicand*, provided by the input, by a 16-bit *multiplier*, provided by the low-order 16 bits of the accumulator register. This operation produces a 32-bit result in the accumulator register.

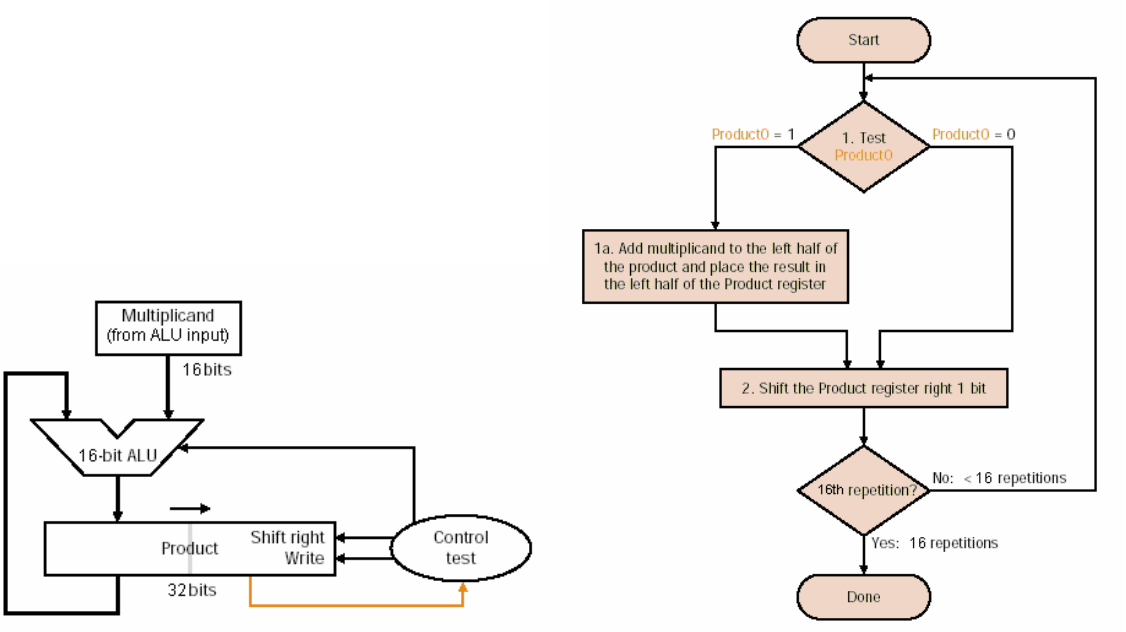

Figure 2: Multiplier Design

Note that the "ALU" shown in Figure 2 is not an entire ALU, but an adder/subtractor.

Example

Assume a 4-bit multiplicand register and a 8-bit product register. If the multiplicand is 9 and the multiplier is 5, Table 4 illustrates the operation of the multiplier.

multiplicand register (MR)	product register (PR)	next action	it
1001	0000 0101	LSB of PR is 1, so PR[7:4]=PR[7:4]+MR	1
1001	1001 0101	shift PR	1
1001	0100 1010	LSB of PR is 0, so shift	2
1001	0010 0101	LSB of PR is 1, so PR[7:4]=PR[7:4]+MR	3
1001	1011 0101	shift PR	3
1001	0101 1010	LSB of PR is 0, so shift	4
1001	0010 1101	PR is 45, done!	X

Table 4: Multiplier Example

Divider Implementation

Fortunately, the same hardware (aside from differences in the control logic) may be used for the divider. First, let's cover some terminology. If you divide A by B, A is referred to as the *dividend* and B is referred to as the *divisor*.

A divider design is shown in Figure 3.

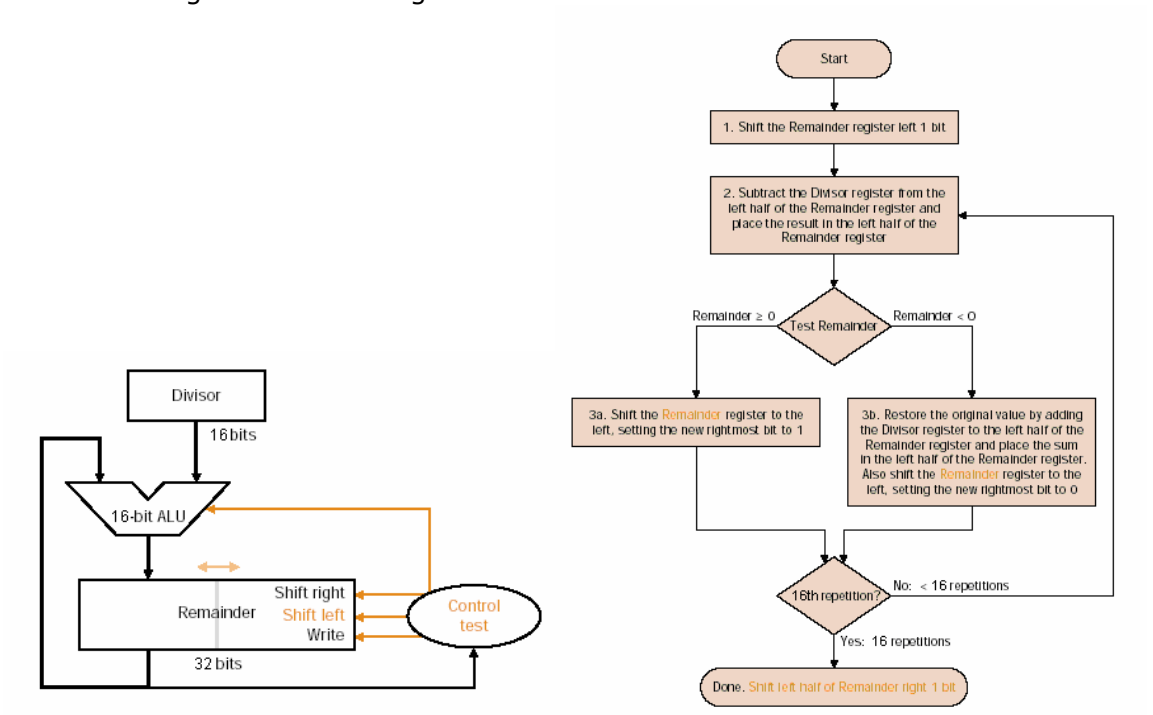

Figure 3: Divider Design

Example
Assume a 4-bit divisor register and a 8-bit remainder register. If the divisor is 4 and the dividend is 11, Table 5 illustrates the operation of the divider.

divisor register (DR)	remainder register (RR)	next action	it
0100	0000 1011	shift RR left 1 bit	0
0100	0001 0110	RR[7:4]=RR[7:4]-DR	1
0100	1101 0110	RR<0, so RR[7:4]=RR[7:4]+DR	1
0100	0001 0110	shift RR to left, shift in 0	1
0100	0010 1100	RR[7:4]=RR[7:4]-DR	2
0100	1110 1100	RR<0, so RR[7:4]=RR[7:4]+DR	2
0100	0010 1100	shift RR to left, shift in 0	2
0100	0101 1000	RR[7:4]=RR[7:4]-DR	3
0100	0001 1000	RR>=0, so shift RR to left, shift in 1	3
0100	0011 0001	RR[7:4]=RR[7:4]-DR	4
0100	1111 0001	RR<0, so RR[7:4]=RR[7:4]+DR	4
0100	0011 0001	shift RR to left, shift in 0	4
0100	0110 0010	shift RR[7:4] to right	
0100	0011 0010	done, quotient=2, remainder=3	

Table 5: Divider Example

What to Submit
The project report must consist of the following components.

- *All high-level designs*
  Submit a plot for each behavioral design. This includes any graphical designs or pure HDL. Feel free to add comments/notes where appropriate to explain any details concerning your designs. You are not required to submit any HDL that was generated from graphical designs.

- *Testbench design*
  Submit the design of your testbench.

- *Behavioral simulation*
  Submit the waveform for the behavioral simulation of your testbench.

- *Synthesized designs*
  Submit plots revealing the entire hierarchy of your entire synthesized (gate-level) design.

- *Synthesis reports*
  Submit synthesis reports for timing and cell usage.

- *Layout*
  Submit a plot of your final layout from the Cadence IC-Tools. Add rulers to show the dimensions of your final layout.

- *Timing simulation*
  Submit the waveform of the testbench simulation that includes the cell and interconnect delay models.

- Place-and-route timing reports
  Submit a place-and-route report for the timing of your design. Also submit a report for dynamic power analysis over your testbench simulation.