

**CSCE 611**  
**Lab 4**  
**Design of an Unsigned Integer Multiplier/Divider**  
**Due Date: 10/18**

**Design Requirements**

The goal of this lab is to design a multicycle unsigned integer multiplier/divider. You will consist of the following building blocks:

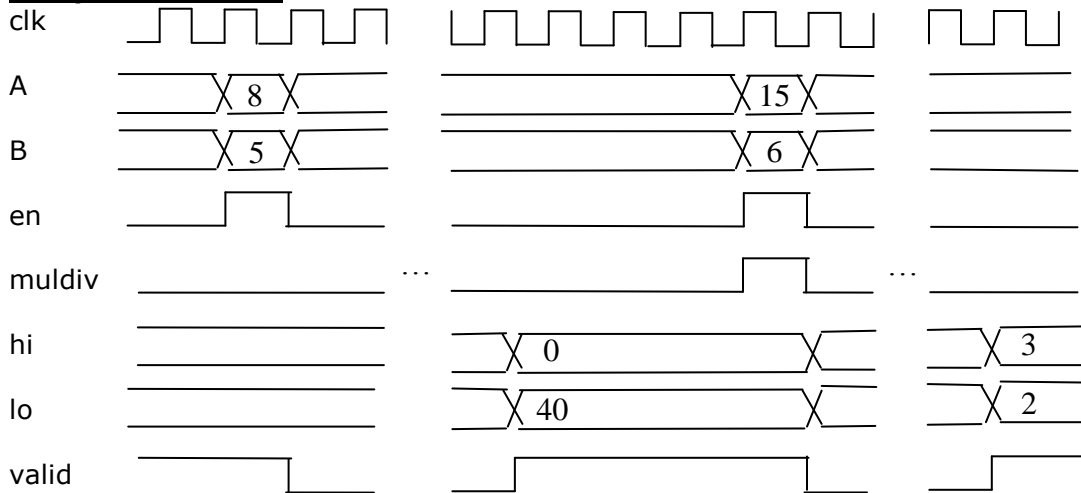
- Your existing ALU, which will be used for its ability to act as an adder/subtractor.
- A component called "**hilo\_shifter**" that is provided in the CSELib library. This component is a 64-bit register with 32-bit enables and the ability to shift left and right.
- A component called "**count6**" that is provided in the CSELib library. This component is a 6-bit counter.
- A state machine based controller that you must design.

**Top-Level Interface**

Your component must have the following top-level interface:

<b>Name</b>	<b>Type</b>	<b>I/O</b>	<b>Description</b>
clk	std_logic	input	clock signal, rising edge active
rst	std_logic	input	asynchronous active high reset
A	std_logic_vector (31 downto 0)	input	Multiplier input (for multiply) Dividend input (for divide)
B	std_logic_vector (31 downto 0)	input	Multiplicand (for multiply) Divisor (for divide)
muldiv	std_logic	input	Mode select (0 for multiply, 1 for divide)
en	std_logic	input	Input enable (active high) Assert for one cycle to register inputs and mode select
hi	std_logic_vector (31 downto 0)	output	High-order 32 bits of product (for multiply) Remainder (for divide)
lo	std_logic_vector (31 downto 0)	output	Low-order 32 bits of product (for multiply) Quotient (for divide)
valid	std_logic	output	Active-high output valid Indicates that multiply or divide operation has completed

### Sample Waveform



### CSE Components

#### *hilo\_shifter*

Name	Type	I/O	Description
clk	std_logic	input	clock signal, rising edge active
rst	std_logic	input	asynchronous active high reset
hi_data_in	std_logic_vector (31 downto 0)	input	input for upper 32 bits (used for initialization)
lo_data_in	std_logic_vector (31 downto 0)	input	input for lower 32 bits (used for initialization)
shift_left	std_logic	input	For each cycle this signal is asserted, all 64 bits are shifted one position to the left (active-high)
shift_right	std_logic	input	For each cycle this signal is asserted, all 64 bits are shifted one position to the right (active-high)
en_hi	std_logic	input	Input enable (active high) Assert for one cycle to write the upper 32 bits
en_lo	std_logic	input	Input enable (active high) Assert for one cycle to write the lower 32 bits
shift_right_hi	std_logic	input	For each cycle this signal is asserted, the upper 32 bits are shifted one position to the right (active-high)
shift_bit_in	std_logic	input	Defines what bit is shifted in during a left-shift or right-shift
hi	std_logic_vector (31 downto 0)	output	The current value of the upper 32 bits
lo	std_logic_vector (31 downto 0)	output	The current value of the lower 32 bits

**counter6**

Name	Type	I/O	Description
clk	std_logic	input	clock signal, rising edge active
rst	std_logic	input	asynchronous active high reset
en	std_logic	input	While asserted the counter increments by one per cycle (active-high)
count	std_logic_vector (5 downto 0)	output	The current value of the count

**Notes**

Consider a 4-bit multiplier and assume you want to multiply 15 by 15 (1111 x 1111).  
The

multiplicand register (MR)	product register (PR)	next action	it
1111	0000 1111	LSB of PR is 1, so PR[7:4]=PR[7:4]+MR	1
1111	1111 1111	shift PR	1
1111	0111 1111	LSB of PR is 1, so PR[7:4]=PR[7:4]+MR	2
1111	1 0110 1111	Note that 1111 + 0111 = 1 0110 (carryout of 1!) when shifting PR, need to shift in a 1 instead of zero when shifting right	2
1111	1011 0111	LSB of PR is 1, so PR[7:4]=PR[7:4]+MR	3
1111	1 1010 0111	Again carryout from add is 1, so shift in a 1	3
1111	1101 0011	LSB of PR is 1, so PR[7:4]=PR[7:4]+MR	4
1111	1 1100 0011	Again carryout from add is 1, so shift in a 1	4
1111	1110 0001	sum = 225	

Notice how you sometimes need to shift in a 1-bit when shifting to the right. This happens whenever you have a carryout during the add.

You can implement this using the following technique:

1. add an output to your ALU which passes the CARRYOUT output from your arithmetic unit out of the ALU, then
2. in the controller design, connect the ALU's CARRYOUT output to the shift\_in input on the hilo\_shifter when performing a multiply.

Also, make sure you set the "type" of all default outputs to "Comb" instead of "Clocked":

	B	C	D	E	F	G	H	I	J	K	L
op	Name	Mode	Type	Bounds	Initial	Category	Assign In	Expression	Scheme	Default	Reset
	clk	IN	std_logic			Clock (Rising)		clk'EVENT AND clk = '1			
	rst	IN	std_logic			reset (Async High)		rst = '1'			
	A	IN	std_logic_vector	(31:0)		Data					
	B	IN	std_logic_vector	(31:0)		Data					
	muldív	IN	std_logic			Data					
	en	IN	std_logic			Data					
	hi	OUT	std_logic_vector	(31:0)		Data	<auto>		Comb		
	lo	OUT	std_logic_vector	(31:0)		Data	<auto>		Comb		
	valid	OUT	std_logic			Data	<auto>		Comb	'0'	
	shift_left	LOCAL	std_logic			Data	<auto>		Comb	'0'	
	shift_right	LOCAL	std_logic			Data	<auto>		Comb	'0'	
	en_hi	LOCAL	std_logic			Data	<auto>		Comb	'0'	
	shift_right_h	LOCAL	std_logic			Data	<auto>		Comb	'0'	
	shift_bit_in	LOCAL	std_logic			Data	<auto>		Comb	'0'	
	B_reg	LOCAL	std_logic_vector	(31:0)		Data	<auto>		Clocked		
	ALUop	LOCAL	std_logic_vector	(1:0)		Data	<auto>		Comb	vers => '	
	hi_internal	LOCAL	std_logic_vector	(31:0)		Data	<auto>		Clocked		
	lo_internal	LOCAL	std_logic_vector	(31:0)		Data	<auto>		Clocked		
	count	LOCAL	std_logic_vector	(5:0)		Data	<auto>		Clocked		
	en_out	LOCAL	std_logic			Data	<auto>		Comb	'0'	
	rst_out	LOCAL	std_logic			Data	<auto>		Comb	'0'	
	alu_out	LOCAL	std_logic_vector	(31:0)		Data	<auto>		Clocked		

## **Project Submission**

Each group must submit:

1. An archive snapshot of your design library  
(e.g. tar cvf lastname1\_lastname2.tar ~/HDS/my\_611\_project/ALU\_lib)

Submit your projects through the course Moodle site (<http://dropbox.cse.sc.edu>)