Columbia University
Department of Electrical Engineering
EECS E4340. Problem Set #2.
Digital system design and VHDL modelling.
Due: February 18, 2004

Before you begin this problem set, you will want to go through the ”VHDL design entry and simulation” tutorial (handed out in class) to become familiar with the design environment and the tools which will be required to complete this assignment. The tutorial will probably take 6-8 hours to complete so please plan accordingly.

In this problem set, you are asked to design a sequential add-shift multiplier that multiplies two 16-bit (unsigned) numbers returning a 32-bit product. The datapath for the multiplier is shown in Figure 2. The algorithm for this multiplier is illustrated in Figure 1 for two 5-bit numbers. The initial partial product (stored in register A in the dataflow) is 0, as is the carry out register C. The multiplier is initially loaded into register Q and the multiplicand in register B. Each time the multiplier bit being processed is a 1, an addition of the multiplicand, followed by a right shift (and registers C, A and Q, together) is performed. Each time that the multiplier bit is 0, only a right shift is performed. In order to count, the number of add-shift (or just shift) operations that occur, the counter P is used. This is initialized to 15 and counts down to 0, giving the sixteen operations required to multiply two 16-bit numbers. When the P counter gets to zero, registers A and Q (together) contain the 32-bit product.

The controller you will design for this multiplier datapath has five inputs. go triggers the multiplication, presumably after the multiplier and multiplicand have been loaded into registers Q and A, respectively. $loadb$ loads the B register from $data\_in$, when the state machine is in its “idle” state. $loadq$ loads the Q register from $data\_in$, when the state machine is in its “idle” state. You may assume that $loadb$ and $loadq$ are synchronized and “pulsed” so that they are only high for one cycle to perform a load, and that when they are high, the appropriate data is valid on $data\_in$. is_zero and $q(0)$ come from the datapath as status. $q(0)$ is the LSB of the Q register. The outputs of the controller are the appropriate control lines for the multiplexers of the datapath. Please create and name these as you wish as you design your controller.

1. Draw an ASM chart for the controls of your multiplier design.

23	**10111**	**Multiplicand**
19	**10011**	**Multiplier**
	00000	**Initial partial product**
	10111	**Add multiplicand, since multiplier bit is 1**
	10111	**Partial product after add and before shift**
	010111	**Partial product after shift**
	10111	**Add multiplicand, since multiplier bit is 1**
	1000101	**Partial product after add and before shift**
	1000101	**Partial product after shift**
	01000101	**Partial product after shift, since multiplier bit is 0**
	001000101	**Partial product after shift, since multiplier bit is 0**
	10111	**Add multiplicand, since multiplier bit is 1**
	110110101	**Partial product after add and before shift**
437	**0110110101**	**Product after final shift**

Figure 1: Binary multiplication algorithm.

2. Write a complete VHDL description of your multiplier, both controls and datapath. It is probably easiest to do this as a single entity-architecture for the datapath and one for the controls, connected schematically in Composer. You may only use VHDL $process$ statements to represent latches and the combinational logic of your ASM. All other VHDL constructs should be concurrent and should clearly represent the underlying hardware. You are to run extensive simulations on your design with a VHDL testbench to verify that it functions correctly.

Please turn in your complete VHDL, including your testbench (this can be easily exported from the Cadence Composer interface). Please also hand-in the ASM charts for your controller design. You should also turn in a description of the functional testing you did for your design, and any waveform plots or other results to substantiate this testing.

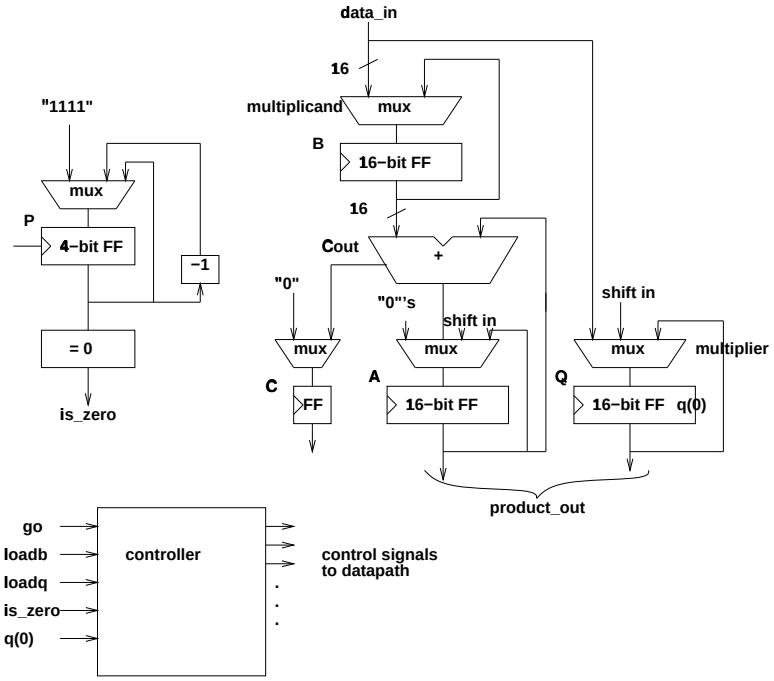

Figure 2: Binary multiplier datapath.