University of Pennsylvania
Department of Electrical and Systems Engineering

Digital Design Laboratory

Lab 4: Combinational Multiplier with Binary-to-BCD Converter

Purpose
In this lab, you will learn about:

- The operation and design of a combinational multiplier for non-signed numbers
- The use of timing constraints
- The use of multiple 7-segment displays at one time
- The conversion of numbers from binary to BCD using a combinational circuit
- Reading Xilinx reports

This lab is somewhat longer and more involved than in previous labs. For this reason, it will be a two week lab. For the same reason, make sure you are careful to simulate and verify the correct operation of each component as you go. Otherwise, you could easily run into problems later.

Pre-lab and Background Information

Multiplication is a very common operation in digital systems. For that reason, custom designed multipliers are often used. The goal of this lab is to make you familiar with the design and implementation of such a multiplier. Multiplication of a 3-bit number B(2:0) by a 4-bit number A(3:0) can be illustrated as follows.

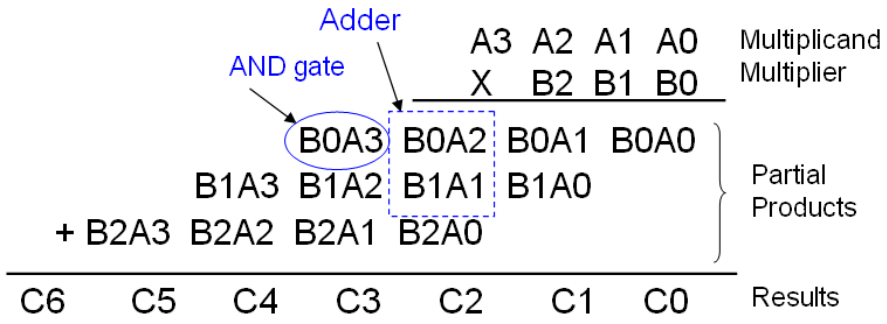
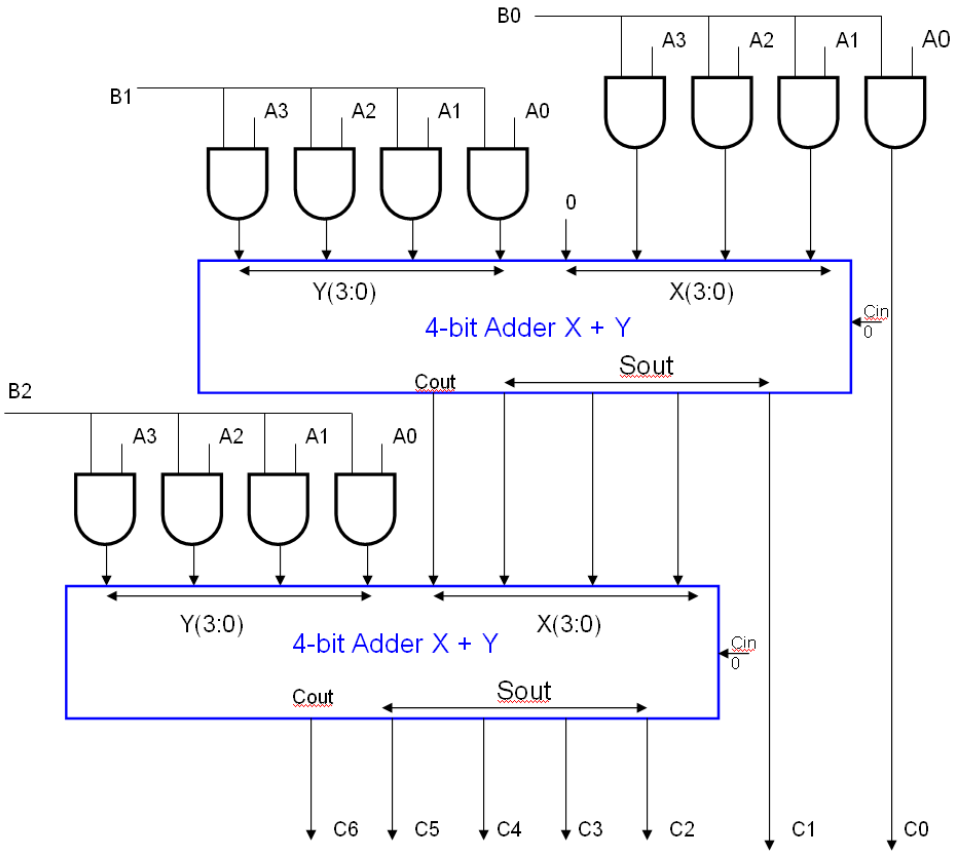


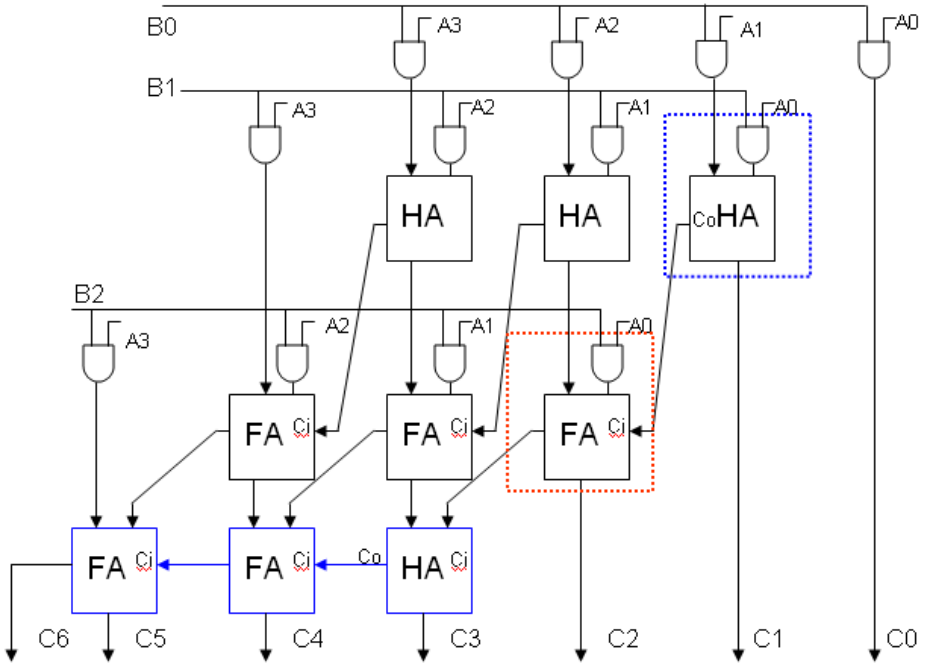
Figure 1. Multiplication of two binary numbers

The implementation follows a one-to-one topological correspondence with the manual multiplication shown above. This gives rise to a very regular structure that is easily implemented. Figure 2 shows two alternative implementations for a 4x3bits multiplier.

Figure 2a shows one such implementation using two 4-bit adders. Figure 2b shows a slightly different implementation that makes use of half adders, full adders and AND gates. The adder sums a locally-computed partial product (AND gate) with an input S that is the sum passed from the row above and a carry from the block diagonally



a.



b.

Figure 2. Combinational 4-bit by 3-bit multiplier implementation

above. It generates a sum and a carry out that are passed to the row below. This multiplier is also called a carry-save multiplier. The main difference with the implementation of Fig. 2a above is that the output carry bits are passed diagonally downwards instead of to the right (as is the case with the circuit of Fig. 2a). The carry-save implantation is usually faster since you can use a fast carry look-ahead adder for the bottom adders.

Another important part of this lab is **binary to BCD conversion**. We want to display the final output on 7-segment displays, but since the output of the multiplier is in binary, we have to convert it before we can display it. Although it is possible to build a decoder by the same method as in previous labs, this time the code would be pages and pages long. We would rather have a more efficient method. Once such implementation is called the "*shift-add-3*" algorithm. Here is the concept:

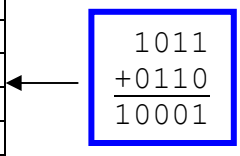
Suppose we have a four-bit binary number that we want to convert to BCD. If it is less than 10, we don't need to change anything. If it is larger than 10, however, we need to do a conversion. We can subtract 10 from the original number to get the ones digit. Then, putting a 1 in the tens digit in BCD can be thought of as adding $0001\ 0000_2=16_{10}$ to the original number. For example,

Start with $12=1100_2$	$0000\ 1100_2$
Subtract $10=1010_2$	$0000\ 0010$
Add $16=10000_2$	$0001\ 0010_{BCD}$

If we combine these operations into one, we see that to convert a four digit number from binary to BCD, just add 6 or 0110_2 if the number is greater than or equal to 10. This works well for four digit numbers, but what if we have more than that? In this case, we can shift the input, one bit at a time, through sets of four bits, and at each step add 0110_2 to a digit if the number is 10 or greater. Here is $22=10110_2$ as an example:

Table I: Illustration of converting a 5-bit binary number into a BCD number

Operation	Tens	Ones	Input
Binary Input			10110
Shift Left (and check if ≥10?)		1	0110
Shift Left (and check):		10	110
Shift Left (and check):		101	10
Shift Left ("Ones" is >10:add 6)		**1011**	0
After adding $6=0110_2$	**1**	**0001**	0
Shift Left	10	0010	
BCD Output	2	2	

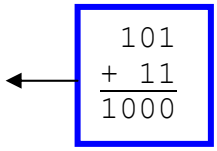


After each shift, one needs to check if the number in the "Ones" column is equal or greater than 10_{10}. If it is, one add 6 to it before shifting again. Notice that when we added 0110_2, the carry bit was carried into the next digit. This means we would need five output bits at each step rather than four. However, there is a shortcut we can use to avoid this. What does shifting a binary number one spot to the left do to the number mathematically? It has the effect of doubling a number (try it out!). If we switch the order of the operations, we only need four bits of output. Specifically, we can add 3 and double instead of doubling and adding 6, because both give the same result in the end. This is where the algorithm gets its name. The new rule then is to shift the input, one bit at a time, through sets of four bits, and if a digit is

five or greater, then we add 3 before shifting again. Here is the same example again, using the "shift-add-3" algorithm:

Table II: Conversion of a 5-bit number into a BCD number using the shift-add-3 algorithm.

Operation	Tens	Ones	Input
Binary Input			10110
Shift Left (and check if ≥ 5?)		1	0110
Shift Left		10	110
Shift Left (Ones >5: add 3)		**101**	10
Add 3=0011_2		**1000**	10
Shift Left	1	0001	0
Shift Left	10	0010	
BCD Output	2	2	



The same procedure can be applied for any number of bits. You will keep shifting (and conditionally adding 3) until all input bits have been shifted (e.g. for an 8 bit you will need to shift 8 times).

Now, the question is how to implement this in your circuit. First of all, we need a macro which has four bit input and output and which performs the operation "add 3 if the input is 5 or greater". This is up to you to design however you want. You can use schematics, K-maps, VHDL, or anything else. Then, the converter can be arranged in the following manner:

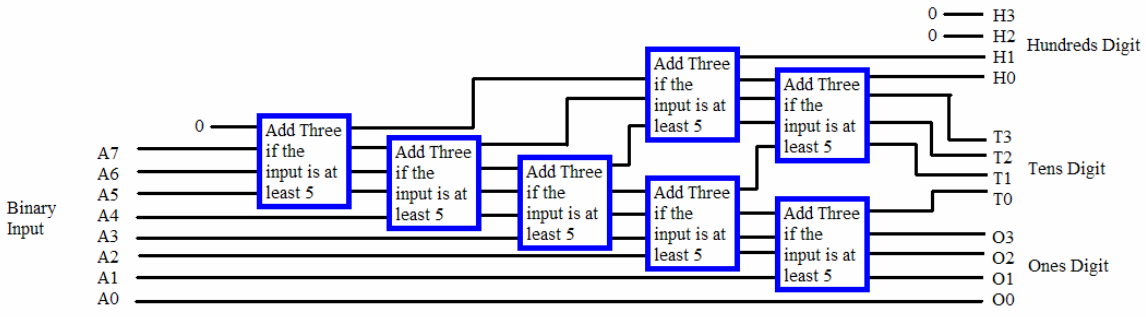


Figure 3: Binary to BCD Converter

This diagonal arrangement is what causes the bits to "shift" through the algorithm as described above.

Now that we have the individual digits, we can easily **convert them into 7-segment display** signals as we did before in order to display them. However, as you noticed in previous labs, the 7-segment displays all share the same cathodes (see description of the Peripheral DIO4 board, pp. 1-2). Therefore, we will need to come up with a way to show all of the digits. The solution, as you may imagine, is to switch between the digits, displaying one at a time. However, if you do this at just the right speed (just like in the movies), the switching is imperceptible and the display looks correct. We will need a switcher, a component which takes a number of inputs and passes just one to the output based on a separate clock signal.

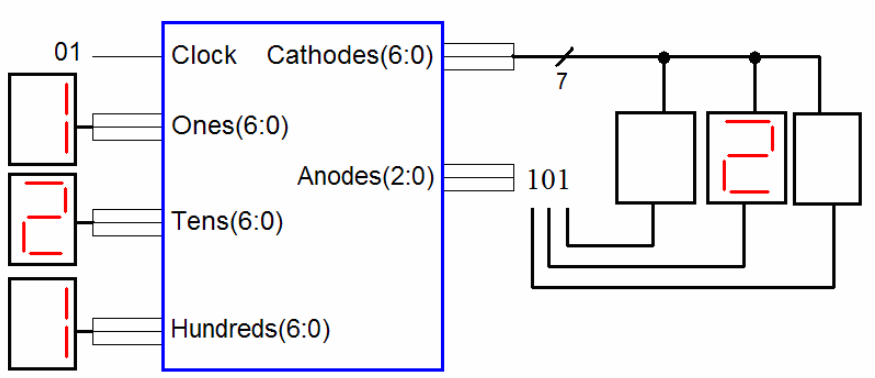


Figure 4: Block diagram of the switching circuit for the 7-segment displays.

Based on the clock input, it should choose the ones, the tens, or the hundreds digit input to pass to the cathodes. The switch circuit also has to alternate between the anodes of the 7-segment displays in order to make sure only the correct display is on at any time.

Unfortunately, there is one more small issue: The 100MHz system clock on the board is too fast for this operation. We need to divide this signal down to be able to use it. Luckily, Xilinx has a built-in divider that we can use.

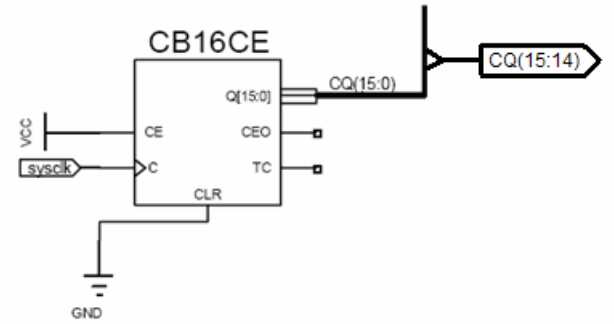


Figure 5: CB16CE counter circuit used to convert the high speed clock to a lower frequency.

This CB16CE is a counter, but if we take just the last two bits of the output, it acts as a divider that gives a clock signal we can use. In the end, then, the top level schematic should look something like this:

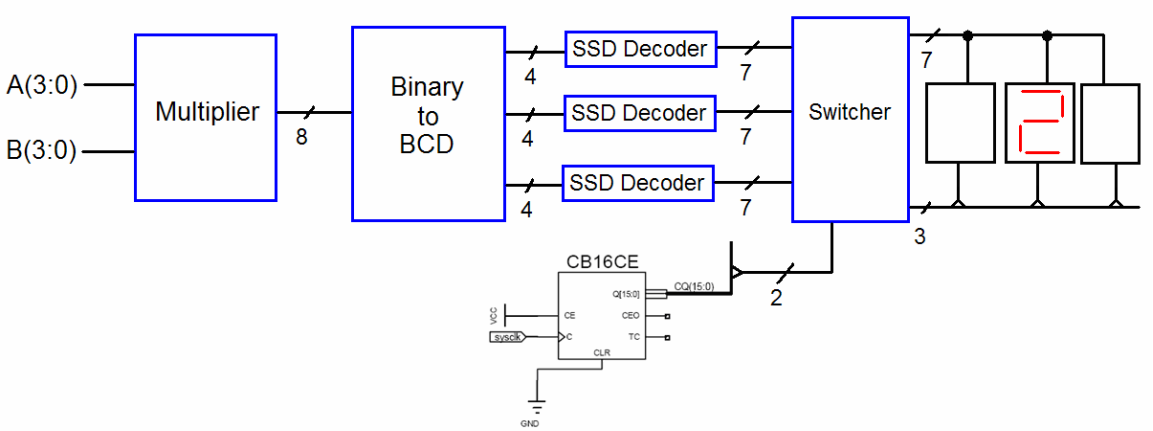


Figure 6: Overall block diagram of the circuit consisting of the 4x4 multiplier, converter, 7-segment display decoders, switcher and display.

Finally, we have one more concept which is important to this lab. In many cases, you want to specify **timing constraints**. These are specification related to delays that the circuit needs to achieve. A typical spec would be the maximum delay between input and output for a combinational circuit (for sequential, clocked circuits, other constraints related to the set up times, clock period, etc. can be added). This limits the maximum frequency at which the circuit can run, so it is important to make sure that whatever circuit we build meets its specifications.

Pre-Lab Reading and assignment

1. Select one of the two implementations and draw the corresponding schematic for a **4-bit by 4-bit** multiplier.

Pre-Lab Questions

Before coming to the lab answer these questions. The pre-lab needs to be handed in at **the start of the lab**.

1. For a MxN binary multiplier implemented with the schematic of Fig. 2a, how many AND gates and M-bit adders are needed?
2. The same question as above for the implementation of Fig.2b: how many AND gates, FA and HA are needed.
3. For the circuit of Fig. 2a write down the values of the input and output of each block on the schematic for a multiplication of (1101)x(111). Verify that the result is correct. Put is in your lab notebook and hand a copy in at the start of the lab.
4. For Fig. 2b do the same as for question 3: write on the schematic at each input and output of each block the corresponding bit value for the multiplication of (1101)x(111) and verify that the circuit works properly. Hand it in at the start of the lab.
5. In Figure 3, write down the values of the input and output of each block on the schematic for a conversion of 10111010_2. Verify the output by converting 10111010_2 to BCD by hand, and make sure the outputs correspond. Hand in a copy at the start of the lab.

In-Lab Assignment
F. Ketterer Lab, 204 Moore

Parts and Equipment

1. PC with Xilinx 8.2i software
2. Virtex-II Development FPGA Board
3. Peripheral DIO4 board

Procedure

During the **first week**, you will design and simulate a 4bit by 4bit multiplier, binary-to-BCD converter and 7-segment decoder:

1. Create a new project and give it the name CombMultipl. Place the project in the C:\users\your_name folder.
2. Create the schematic of the **4x4** bit combinational multiplier. Make use of previously designed circuits such as the 4-bit adder or the Full adder. In case you decide the implement the circuit of Fig. 2b, make use of macros (modules) such as the one shown in the blue and red rectangles. You do not need to redesign the full adder; use the one of the previous labs. Use of buses and net names to simplify the routing when appropriate.
3. Do a behavioral simulation. As input give several values and verify that that the output is correct: e.g. multiply the decimal equivalents: 5x3; 9x7; 13x7; 15x7; 14x15. Once you are sure the multiplier works correctly, make it into a macro.
4. Create the "shift-add-3" module. It should have a four bit input and a four bit output. If the input is four or smaller, it should pass through unchanged. If the input is five or larger, the output should be the same as the input plus 3. You can realize this module as a schematic or VHDL code. To design it you can start with the truth table and K-maps. When you are done, do a simulation, testing various values.
5. With the previous module, create the binary to BCD converter, following the schematic in Figure 3. Again, run a simulation of the converter to make sure it works.

During the second week, you will build the switching circuit and complete the overall system.

6. Build the switching circuit (see Fig. 4). It should have as inputs the 7-segment display cathode signals for the ones, tens, and hundreds digits, as well as a two-bit clock signal. It should also have two outputs: one 7-bit signal for the cathodes and one 3-bit signal for the anodes. As the clock changes, the switch should alternate which signal is passed to the cathodes. At the same time, it should turn on the correct anode and turn off the rest. Do a simulation when you are done.
7. Create the top level schematic, similar to Fig. 6. It should have two four bit inputs, which we will control on the board with the switches. There should also be an input from the system clock. The outputs should be a 7-segment signal to the cathodes and a 3-bit signal to the anodes of the 7-segment display. You should also add an extra output bit to turn off the anode of the unused fourth display rather than leaving it hanging.
8. Set the timing constraints for the circuit. We will specify that the max delay between pads should be maximum 20ns. To specify the Timing constraint, click on the Create Timing Constraint in the User Constraint Processes. This will open the Constraint Editor. Select the "Global" tab and click on the Pad to

Pad button. This will allow you to enter a global constraint for the pad to pad delay, shown below. When done, save and close the Editor.

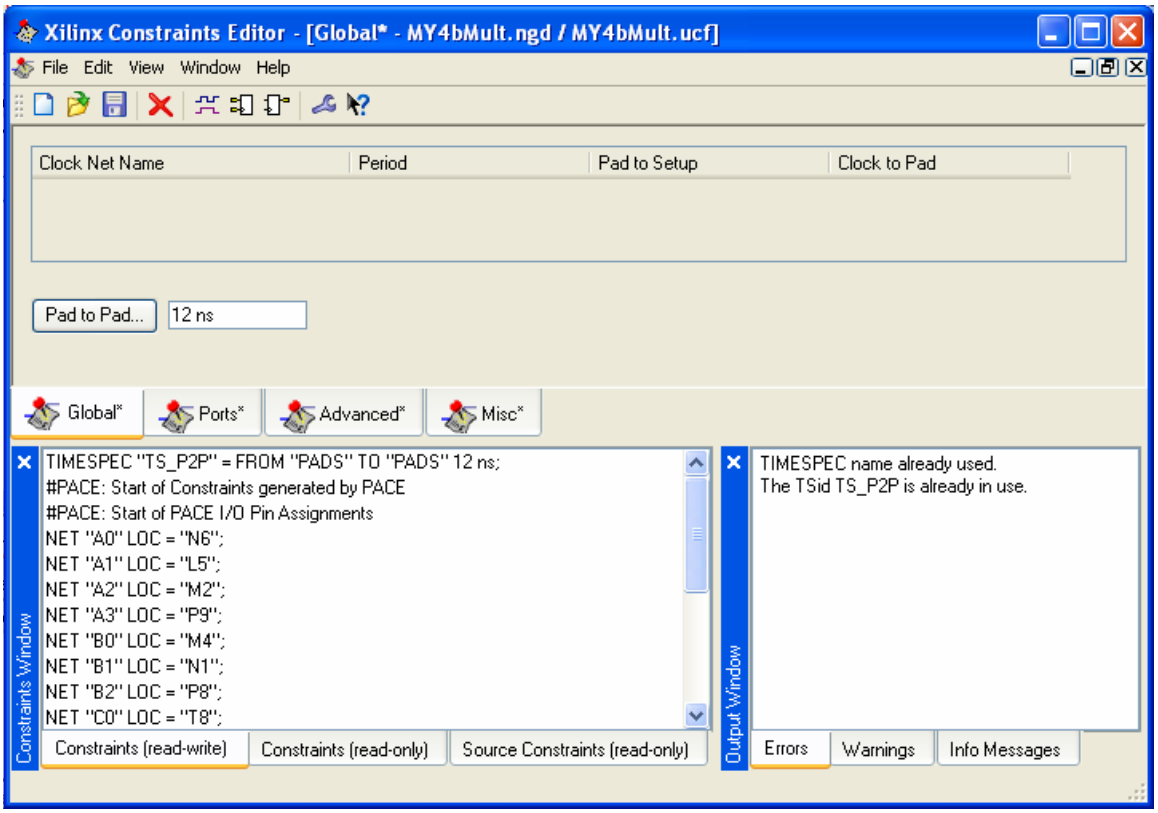


Figure 7: Constrain Editor window to specify the Pad-to-Pad delay (in this example we specified 12ns)

9. Since you will download the circuit on the FPGA board, you can also specify the pin locations. The outputs should appear on the 7-segment displays and the inputs are set with the sliding switches.
10. Implement the design.
11. Now you can check various reports. In the Design Summary, click on the Map Report to see a detailed report on the device implementation.
    a. Find the number and percentage of LUTS used;
    b. number and percentage of slices used;
    c. How many IOB are used?
    d. Are there any errors or warnings?
    e. Was any logic removed?
12. Do a timing simulation (Post Place & Route simulation).
13. Determine the maximum delay that the output is valid after the inputs 15x15 are applied. Zoom in the waveform to see the delay. Do you notice anything unusual in the outputs?
14. Next, look at the performance summary in the Summary Window and see if all constraints are met. Also, check the static timing report (see the Device Summary Window in the Project Navigator: Detailed reports). Verify the maximum delays from input to any of the pads. Look at the data sheet report that lists all the delays between inputs and output. Compare with what you have measured. What is the maximum path delay between any node? If the

constraints are not met, the report will give details about what pad has violated the timing specification. If the timing constraints are not met, there are a couple of things that can be done. You can redesign the circuit, using a more efficient design or increase the Place and Route Effort level when doing the implementation. The latter one can be done by right clicking on the Implement Design Process and selecting Properties.

15. Download the circuit to the board and test the multiplier; use the 7-segment displays to show the output. Set the input bits using the switches.
16. Give a demo to the TA.

Hand-In (At the start of the next lab)

You have to hand in a report that contains the following (See guidelines for reports):

1. Course title, Lab number, Lab title, Your names, and date
2. Answers to the pre-lab questions. Answers submitted online do not have to be repeated here.
3. Brief description of the experiment including the goals and theory
4. Circuit schematics (screenshots, include your name)
5. Waveform simulations (screen shots)
6. Discussion of the results indicating proper function.
7. Conclusion

References:

1. M. Mano and C. Kime, Logic and Computer Design Fundamentals, Prentice Hall, Upper Saddle River, NJ, 2004.
2. J. Rabaey, A. Chandrakasan and B. Nikolic, "Digital Integrated Circuits," Prentice Hall, Upper Saddle River, NJ, 2003.
3. R. Katz, "Contemporary Logic Design," Benjamin Cummings Publishers, New York.
4. "Combinational Design: Binary-to-BCD conversion", Dept. of Electrical and Computer Engineering, Mississippi State University

Updated September 26, 2007