Introduction to Computer Engineering I (ECSE-221)
Assignment 2
Available: Oct 6, 2006
Due Date: Oct 20, 2006

Please submit this assignment by 5pm, Oct 20, 2006 on WebCT. Your assignment must consist of all of the Microsoft Word, text, circuit, and timing files as requested for the questions making sure to carefully name the files *exactly* as specified, since marks will be deducted for any incorrectly named files. You must submit 13 files for this assignment, as follows:

Q1.doc	Q4.tim
Q1.tim	Q4a.cct
Q1.cct	Q4b.cct
Assign2.clf	Q4c.cct
Q2b.cct	Q4d.cct
Q2c.cct	Q4e.cct
Q3.doc	

This assignment will be marked out of 100 points.

Assignments received up to 24 hours late will be penalized by 10%; assignments received up to 48 hours late will be penalized by 20%, and assignments received more than 48 hours late will not be marked.

Q1. (25 points) Although rarely implemented, binary subtraction can be defined in a manner analogous to addition, replacing the carry input and carry output by a borrow input and borrow output. The truth table corresponding to binary subtraction can be defined in terms of the following inputs: borrow in, *Bin*, minuend, *A*, and subtrahend, *B*, and outputs: difference, *D* and borrow out, *Bout*. It is simple to determine that the truth table is as follows:

Bin	**A**	**B**	**D**	**Bout**
0	0	0	0	0
0	0	1	1	1
0	1	0	1	0
0	1	1	0	0
1	0	0	1	1
1	0	1	0	1
1	1	0	0	0
1	1	1	1	1

(a) Using your choice of minimization method, determine both the minimal forms, Sum of Products and Product of Sums, which correspond to *D* and *Bout* respectively.

Write up your answers to question 1 (a) as a Word file, named "Q1.doc". If you use Karnaugh maps, you may scan in your hand-written maps, or create them using a graphics program. All other text must be typed.

(b) Design the implementation for the two functions using whatever form leads to the simplest overall circuit (fewest number of switching elements). In doing so, the forms must be converted to its NAND-NAND or NOR-NOR form. For the case of a tie, you may implement either form. Implement your functions in LogicWorks using the gates in the standard library.

Save the circuit resulting from question 1 (b) in LogicWorks, naming it "Q1.cct".

(c) Devise an appropriate simulation to verify that each circuit is implemented correctly (see appendix for more instructions on how to create a timing file for the simulation).

Save your timing file from question 1 (c) as a tab-delimited timing file, named "Q1.tim".

Q2. (25 points) In this question you will use the subtractor building block you designed in Q1 to construct a 4-bit system. To make things more interesting, you will also design and implement the complete logic from the gate level upwards.

(a) Start off by designing your own version of each of the gates you used from the standard library in Q1, using X-Gates (switches). Since your design is comprised entirely of NAND or NOR gates, only 1 gate type will be required. However you will need at least three varieties to account for different numbers of inputs. As well, you can design a gate with a single input as an inverter. Your design must not contain any passive elements.

Note: Since LogicWorks does not have an active(0) type switch, you may fabricate one using an X-Gate (active(1) switch) and inverter from the standard library, then encapsulate the resulting circuit and use it as a building block in constructing your family of logic gates.

Save your gates in a library called "Assign2.clf".

(b) Repeat Question 1(c), substituting the gates you designed above in part (a). If all is correct, you should get near identical simulation outputs. Explain any discrepancies between the two outputs, using the text tool in Logicworks to write your explanation below your circuit. (Hint: consider changes in gate delay time).

Save your circuit in a file called "Q2b.cct".

(c) Encapsulate your circuit from part (b) to create a 1-bit subtractor module with three inputs, *Bin*, *A*, and *B*, and two outputs, *D*, and *Bout*. Construct a 4-bit subtractor by assembling four of these encapsulated modules together. Use 2 LogicWorks "Hex Keyboards" to input the four bits for the minuend and subtrahend, and a "Hex Display" to show the result. You can also use a binary probe to indicate overflow (i.e. when the most significant bit requires a borrow). Make sure to test your circuit thoroughly.

Save your encapsulated circuit in the library you created for part (a), "Assign2.clf".
Save your circuit in a file called "Q2c.cct".

Q3. (25 points) A convenient way of representing Boolean functions is by specifying either minterms for which the function is true, or maxterms for which it is false. Consider a function $F(A,B,C,D)$ which is true for 0011, 0100, 0101, 0110, 1010, 1011, 1100, 1110 and 1111. The shorthand notation for this function is:

$$F(A,B,C,D) = \Sigma\{3, 4, 5, 6, 10, 11, 12, 14, 15\}.$$

Similarly, we can also specify F in terms of its maxterms as follows:

$$F(A,B,C,D) = \Pi\{0, 1, 2, 7, 8, 9, 13\}.$$

These representations are referred to as $\Sigma\Pi$ and $\Pi\Sigma$ forms respectively. Now do the following:

(a) Determine the minimal sum-of-products ($\Sigma\Pi$) form for F first using the Karnaugh map method, and then using algebraic minimization.

(b) Repeat Part (a), but this time for the minimal product-of-sums form ($\Pi\Sigma$). (Hint: the dual form will be useful here).

(c) Prove algebraically that the minimal forms obtained for Parts (a) and (b) correspond to the same function.

Write up your answers to question 3 as a Word file, named "Q3.doc". You may scan in your hand-written Karnaugh maps, or create them using a graphics program. All other text must be typed.

Q4. (25 points) In this question, you will implement the function from Q3 in various different ways using LogicWorks, using the gates from the LogicWorks libraries (you don't need to create your own gates).

(a) Using the appropriate minimal form from Q3, create a circuit corresponding to F using NAND-NAND logic. Create a LogicWorks timing file (see appendix) to demonstrate that the truth table is correctly implemented.

Save your circuit in a file called "Q4a.cct".
Save your timing file as a tab-delimited timing file, named "Q4.tim".

(b) Using the appropriate minimal form from Q3, create a circuit corresponding to F using NOR-NOR logic. Use the timing file from Q4(a) to test that the truth table is correctly implemented.

Save your circuit in a file called "Q4b.cct".

(c) Implement a circuit corresponding to F using a PROM. Use the timing file from Q4(a) to test that the truth table is correctly implemented.

Save your PROM in the library you created for part (a), "Assign2.clf".
Save your circuit in a file called "Q4c.cct".

(d) Use two 8-bit multiplexers to create a 16-bit multiplexer. Implement a circuit corresponding to *F* using this 16-bit multiplexer. Use the timing file from Q4(a) to test that the truth table is correctly implemented.

Save your circuit in a file called "Q4d.cct".

(e) Although it is straightforward to implement any Boolean function using a multiplexer with 2^n inputs (where *n* is the number of variables in the function), the same can actually be accomplished using half the inputs, i.e., 2^{n-1}. The trick is to use one of the variables (along with its complement) as inputs to the multiplexer (in addition to constant 1's and 0's). Repeat Part (d) using a single 8-input multiplexer. Use the same simulation to show that identical outputs are obtained.

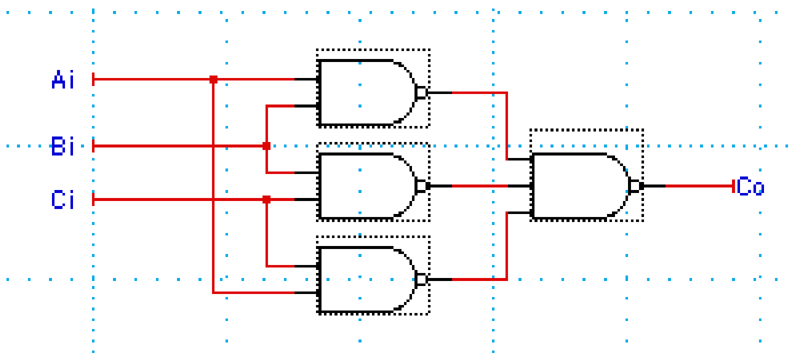
Save your circuit in a file called "Q4e.cct".

Assignment 2
Appendix
Using LogicWorks to test Combinational Logic.

Consider the following truth table which corresponds to the carry output (Co) of a 1-bit full adder:

Ci	**Ai**	**Bi**	**Co**
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Using the methods discussed in Module 2, the following circuit is obtained:



One easy way to test this circuit would be to sequentially apply each of the truth table inputs and verify if the output of the circuit matches that of the table. LogicWorks has such a provision called a *Timing File*. The easiest way to explain it is to show what the corresponding file is for the above example.

Timing file for the Full Adder Carry

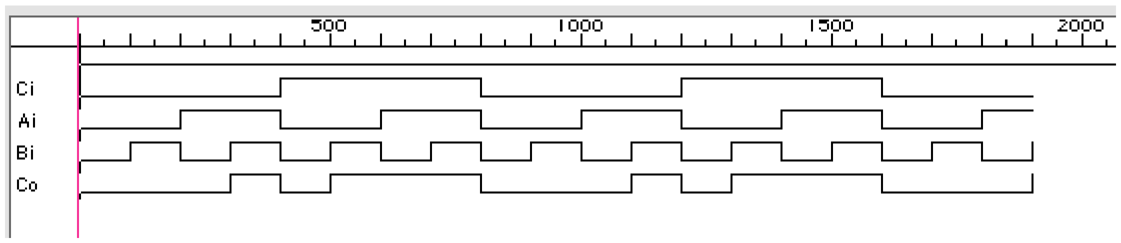
$T	$D	$I Ci	$I Ai	$I Bi
0	100	0	0	0
100	100	0	0	1
200	100	0	1	0
300	100	0	1	1
400	100	1	0	0
500	100	1	0	1
600	100	1	1	0
700	100	1	1	1
800	100	0	0	0
900	100	0	0	1
1000	100	0	1	0
1100	100	0	1	1
1200	100	1	0	0
1300	100	1	0	1
1400	100	1	1	0
1500	100	1	1	1
1600	100	0	0	0
1700	100	0	0	1
1800	100	0	1	0
1900	100	0	1	1

The table looks just like a spreadsheet (in fact it was created using Excel). The leftmost column, with the header $T, represents units of time in the LogicWorks simulation. At each time unit, the simulator updates the values of all nodes in the circuit. The timing file specifies the time values when the inputs change and their respective values. An input is defined with $I followed by the name of the signal, e.g., $I Ci, $I Ai, and $I Bi, for this example. Reading the first two rows of the table, at T=0 inputs Ci, Ai, and Bi are set to 0, 0, and 0 respectively. At T=100, inputs Ci and Ai are held steady while input Bi is changed from 0 to 1, Notice how the timing table has a direct correspondence to the truth table for the circuit.

The above example specifies event changes in absolute time units, e.g, at T=1300 input Bi changes from 0 to 1, while inputs Ci and Ai are held steady at 1 and 0 respectively. Another way of specifying when things happen is to use the $D column to specify the interval between the last time step and the current time (i.e. the Delay). In the example above, each time step corresponds to a delay of 100 time units. It is not necessary to specify both $T and $D, they may be used independently or in combination.

A good way to create timing files is to use a spreadsheet program such as Quattro or Excel. Unfortunately LogicWorks cannot read spreadsheets directly, but it does know how to read tab delimited text. Most spreadsheets can output their data in this form. In general, a circuit can have any number of inputs, so the number of columns will correspond to the number of truth table inputs + 2.

The procedure for testing is as follows. Once your circuit is created in LogicWorks, label each input with a name corresponding to the column headings in your timing file. Make sure that you label the output of your circuit as well so that it is correctly recorded by the simulator (this will all be discussed in detail in the tutorial). All you need to do is use the file dialog to open the timing file. The simulator will then read in the file, and determine the outputs corresponding to your inputs as shown below.



Notice the correspondence to the truth table. Moving from left to right, one cycles through the truth table twice. It's fairly easy to check the output Co against the truth table, at least for small numbers of variables (which will be the case in this course). For circuits with many variables, e.g., the 4-bit subtractor in Q2(c), it becomes difficult to cycle through all possible input combinations. In such cases one normally chooses a subset of the full truth table, one that is sufficient to exercise the functionality of the circuit.

LogicWorks is installed in all of the Engineering computer facilities.

FPF/September 23, 2002 (Modified by JWH, Oct 2, 2006)