INTRODUCTION TO COMPUTER ENGINEERING

WINTER 2014

ASSIGNMENT 2: COMBINATIONAL LOGIC
DUE DATE: FRIDAY FEBRUARY 21ST, 2014 AT 5:00PM

Please submit this assignment by 5:00 pm on the due date via *MyCourses*. Your assignment must be packaged as follows: one compressed file (.zip) containing the following files:

- Answers to all questions as well as discussions (including schematic circuit diagrams and timing printouts) in **one single** Microsoft Word (doc/docx) or portable document format (pdf) file. Note that scanned handwritten solutions are accepted.
- All files related to your Logicworks design. This includes: circuit files (`.cct`), timing files (`.tim`) and library files (`.clf`). All files should be named according to the following convention:

  `LAST_FIRST_QX_DESC.xxx`

  where "`LAST`" and "`FIRST`" are you last and first name respectively (no spaces), "`QX`" should be replaced by Q1, Q2 or Q3 depending on the question in the assignment to which your file belongs, "`DESC`" is an optional short description and "`xxx`" is the file extension.

A penalty of 10% will be applied if you do not follow these guidelines.

This assignment will be marked out of 100 points. **Late submissions will NOT be accepted.**

QUESTION 1 (30 POINTS) – LOGIC DESIGN

The following truth table describes a logic function that you are asked to implement, with A, B, C and D the input Boolean variables and X the function value or output.

row#	A	B	C	D	X
0	0	0	0	0	1
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	1
9	1	0	0	1	1
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	0
15	1	1	1	1	0

(a) Write explicitly the canonical forms for this logic function (both ΣΠ and ΠΣ)
(b) Using the Karnaugh map technique, minimize both canonical forms found under (a)
(c) Implement you original ΣΠ form and its minimized version in Logicworks, using only basic AND, OR, and NOT gates. Develop a testing mechanism (through an imported timing file, see Appendix for instructions) to show that both of these implementations correspond to the truth table for every possible input combination.
(d) Draw a NAND-NAND implementation of the minimized ΣΠ form, and implement it in Logicworks using **only** NAND gates (no other gates). Verify your implementation using the same technique as in (c).
(e) Show algebraically that both minimized functions from (b) are equivalent.

QUESTION 2 (20 POINTS) – COMPARATOR

In this problem, you will design and implement a 2-bit comparator. The 2-bit comparator has a two 2-bit inputs, $X = X_1X_0$ and $Y = Y_1Y_0$, and a one-bit output that is set to 1 when $X = Y$, and set to 0 in any other case.

(a) Write a 4-input, 1-output truth table to represent the function of this comparator.
(b) Derive a Boolean equation representing the ΣΠ canonical form from the truth table in (a).
(c) Using a Karnaugh map, minimize the logic function obtained in (b).
(d) Implement the function obtained in (c) in Logicworks using AND, OR and NOT gates. Verify your implementation by using the procedure outlined in the Appendix.
(e) A simpler implementation of the comparator could be derived using two XNOR gates and one AND gate. Draw this implementation, implement it in Logicworks, and verify that it gives the same results as (d).
(f) How would the implementation in (e) be generalized for a 4-bit comparator?

QUESTION 3 (50 POINTS) – 4-BIT SUBTRACTOR

In this problem, you will design a 4-bit 2's complement subtractor, implement it in Logicworks, and test it.

The 4-bit subtractor works as follows: given two numbers X and Y in 2's complement binary representation on 4 bits, it outputs a 4-bit value representing $X - Y$ in 2's complement.

To obtain full marks, the following requirements must be met:

- You are only allowed to use basic gates, including NOT, AND, OR, NAND, NOR, XOR, XNOR. (You can of course include connections to the ground or to the supply voltage, and additional IO elements such hex display, hex keyboard, etc.)
- You should, in the course of your design, implement a 1-bit adder. Encapsulate this 1-bit full adder as a new reusable subcircuit with its own device symbol, which will be saved in a local library file (see Section 3 of the Logicworks 5 reference, under "Creating a Subcircuit—Bottom–Up" for details)
- The final circuit should be exhaustively tested through an imported timing file (see Appendix).
- The testing mechanism for your design should include a comparator circuit to test whether the output from the 4-bit subtractor you have implemented corresponds to the expected result (which can be loaded from the imported timing file).

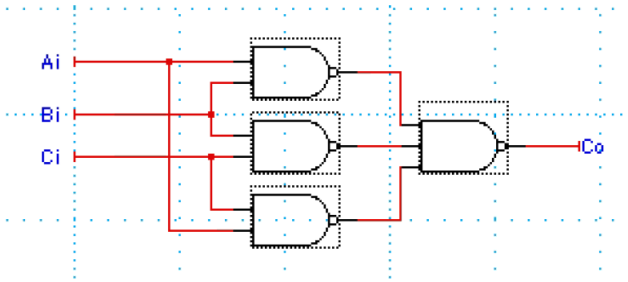
Explain and document all your steps and design choices; truth tables, Karnaugh maps, equations, all designed circuit diagrams and corresponding Logicworks timing outputs should be included in your DOC or PDF submission, and all corresponding Logicworks files should be included in your assignment submission to allow testing.

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 that the corresponding file is for the above example.

Timing file for the Full Adder Carry:

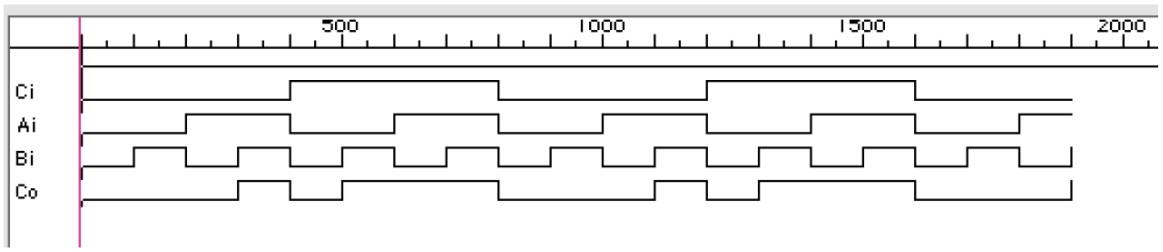
$T	$D	$I Ci	$I Ai	$I Bi
0	100	0	0	0
100	100	0	0	1
200	100	0	1	0
300	100	0	1	1
400	100	1	0	0
500	100	1	0	1
600	100	1	1	0
700	100	1	1	1
800	100	0	0	0
900	100	0	0	1
1000	100	0	1	0
1100	100	0	1	1
1200	100	1	0	0
1300	100	1	0	1
1400	100	1	1	0
1500	100	1	1	1
1600	100	0	0	0
1700	100	0	0	1
1800	100	0	1	0
1900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

The table looks just like a spreadsheet (in fact this one 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 Excel. However, LogicWorks cannot read spreadsheets directly, but it does know how to read tab delimited text. Most spreadsheet programs can output their data in this form. Note also that for very long timing files, creating them through a script might be the best idea (e.g. a C program that writes to a timing file).

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. You need to use the “Simulation-> Import timing…” dialog to open the timing file. The simulator will then read in the file, and determine the outputs corresponding to your inputs as shown in the figure below.



Notice the correspondence to the truth table. Moving from left to right, there are two cycles of the truth table. You can check the output Co against the truth table. For circuits with many variables, it becomes difficult to cycle through all possible input combinations and verify the output by hand. In such cases, the best solution is to:

- First include more inputs to your circuit (and consequently more columns in your timing file) that represent the expected output (i.e. the outputs of your truth table); and
- Develop a testing sub-circuit within your design, which compares the output obtained from your circuit with the expected one (read from the timing file); the result of the comparison should be summarized in a one-bit signal which is set to 1 when expected and actual outputs are the same and 0 otherwise.

For a working circuit, a display of this one-bit signal in the timing windows should remain at 1 throughout the simulation (except for some very short glitches around input transitions due to gate delays).

For complete details, see “Appendix D—Timing Text Data Format” in the Logicworks 5 Reference manual.