Programmable Logic Design Techniques II

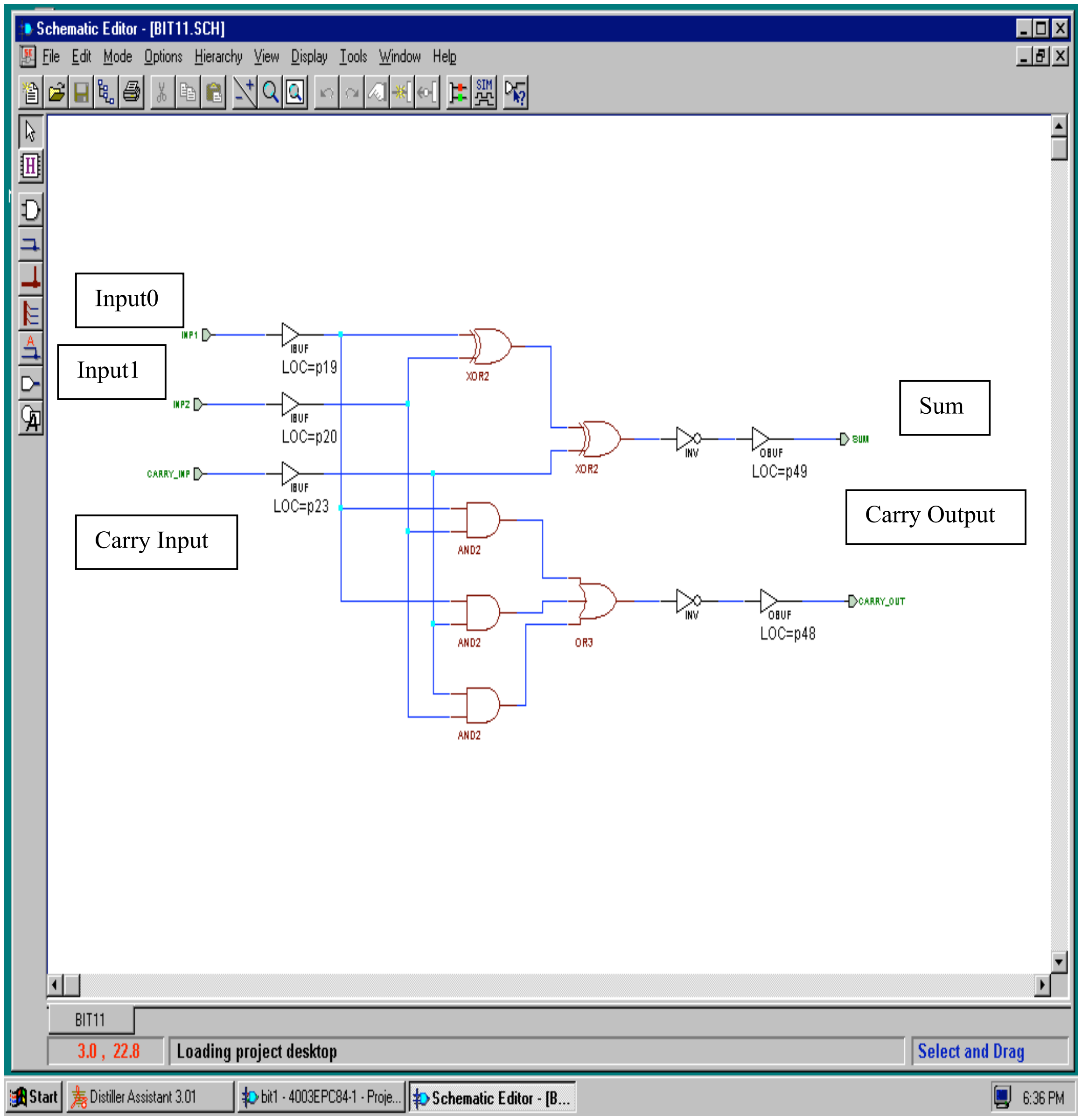
Almost all digital signal processing requires that information be recorded, possibly manipulated and then stored in some way. Thus it makes sense to design small modules that perform basic signal processing operations and then combine them into more complex designs when more complex operations are needed. The resulting design is hierarchical with a number layers with increasing complexity. Building a circuit in this manner is much easier than attempting to cobble together a huge design using only circuit primitives.
In this Lab you will design a module that performs a frequently used operation such as "1-bit" addition. Then you will use it to create a “4-bit” adder. Also, you will use other already designed modules to create a relatively complex circuit for recording the number of events – an "8-bit" up counter. *To be successful in this Lab you should review DH p.270-p.284 and the Guides to the Xilinx software and hardware posted on the PHY440 WWW site.*

Problem 1. Create a digital circuit that accepts two "1-bit" binary numbers and outputs a "1-bit sum", i.e. a “1-bit adder”. Simulate the design, download it to the FPGA demo board and test it.
As specified, the circuit has to have at least two inputs (one for each 1-bit number) and one output (for the 1-bit sum). But it is possible that the sum will not fit into1 bit. For example, adding two 1-bit numbers like 1 and 1, results in a sum 10(2), which requires 2 bits to represent it. For this reason, an extra *carry* output has to be added to indicate when this *overflow* occurs. And if the adder can have a *carry output*, then it makes sense that it should also have a *carry input*. The truth table for such a “1-bit” adder with *carry* input and output is as follows:

Input 1	Input 2	Carry Input	Sum Output	Carry Output
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Transform the truth table into a gate-level logic design using appropriate gate primitives from the Schematic Library of Xilinx Foundation Series Software.

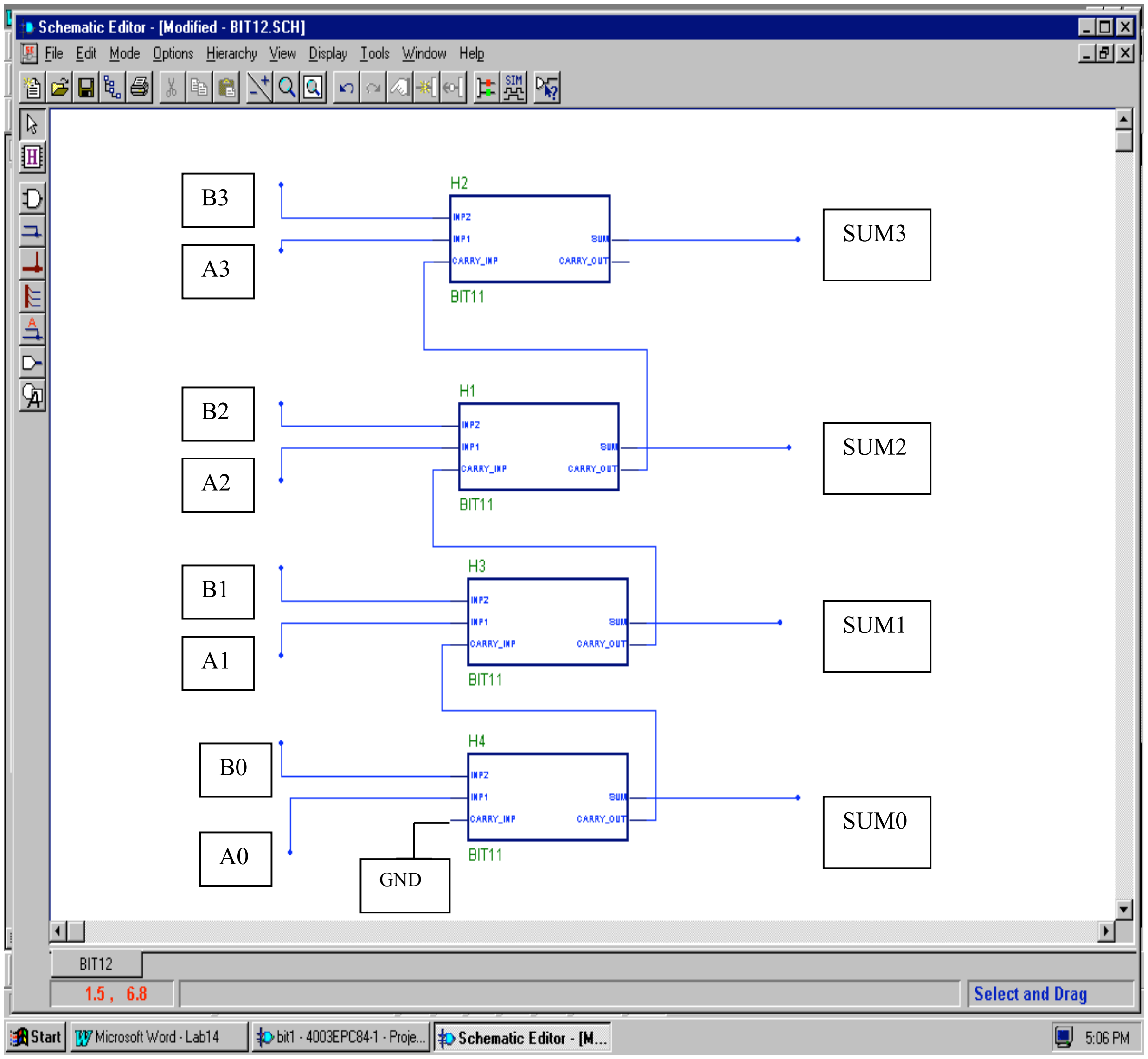
For those of you having difficulties in creating this design on your own, a “hint” is given below:



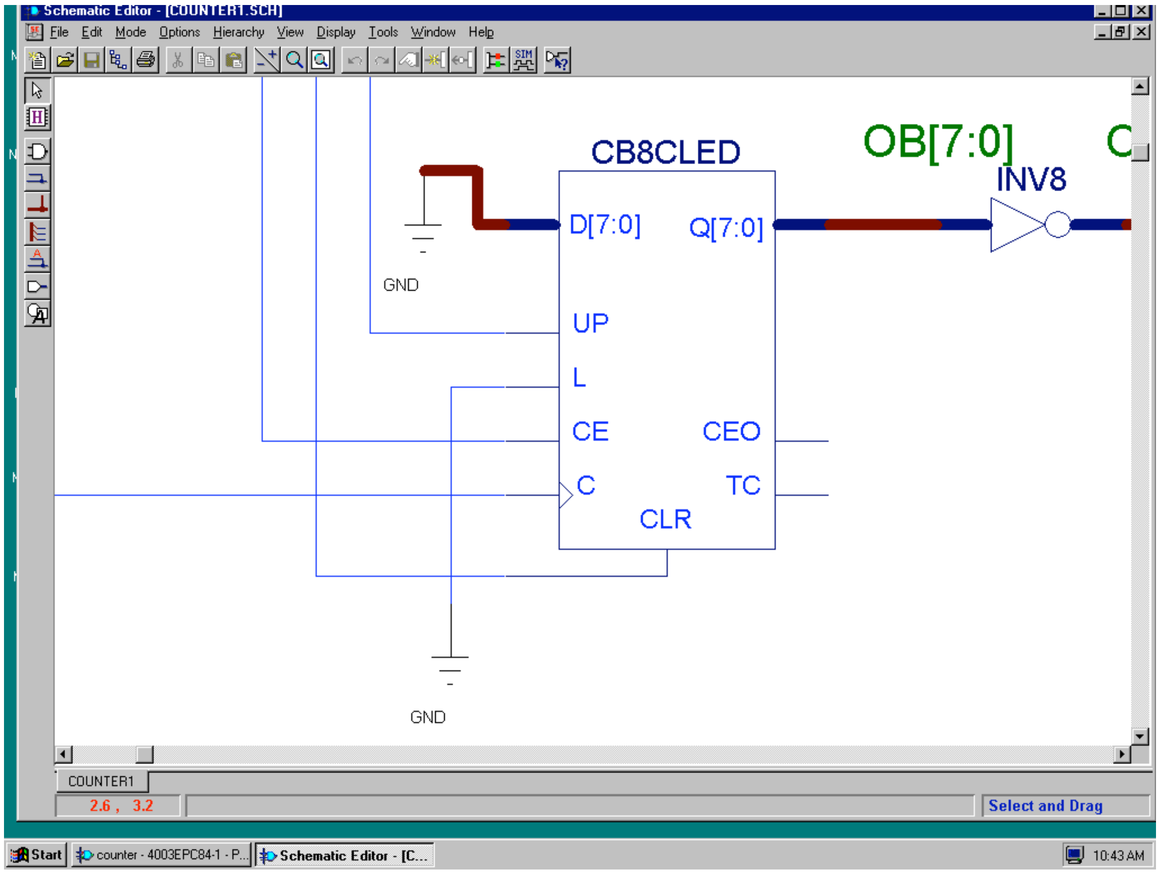
Suppose you have designed and simulated the “1-bit” adder successfully. Now you have to assign the three inputs of the circuit to three pins of the XC4003 FPGA which can be driven by the SW3-1 to SW3-8 switch(es) on the board. Pins # 19, 20, 23, 24, 25, 26, 27 and 28are all at your disposal. The outputs may be assigned to any of the pins # 49, 48, 47, 46, 45, 50 and 51which drive segments of a single 7-segment LED on the board. Since each LED segment is turned “on” by driving the corresponding pin to logic “0” you may invert the output signal (as done in the circuit above). Then you will have a LED segment “on” when the output of the summer is “1” and not “0”.
Now you may wish to create a module that can be saved and used in other designs.
First, get rid of the inverters, the input (IBUF) and output buffers (OBUF) and the associated pins.

Next select **Hierarchy** -> **Create Macro Symbol from Current Sheet** from the menu. Select a name of your choice for your adder and put in relevant comments. Then save it. It will be appended to the **SC** library and available for use in future designs.

Problem 2. Create a design that accepts two 4-bits numbers (A3....A0 and B3….B0) and outputs a 4-bit sum (Sum3….Sum0). Make use of your "1-bit" adder module. Simulate the design, download it to the Xilinx demo board and test it. An example of such a "4-bit" adder is given below. This time you may associate the outputs with pins # 61, 62, 65, 66, 57, 58, 59 and 60, which drive 8 LED bars (D9-D16 row) on the Xilinx board. Please, note that the pin ordering is important. Pins # 61 and # 60 should be associated with the most and least significant bits of the 8-bit number, respectively. You may invert the output signal to have the LED segments "on" when the output of the summer is "1". Also, you may associate pins 19, 20, 23 and 24 with the bits A3.0, and pins 25, 26, 27 and 28 with the bits B3.0.



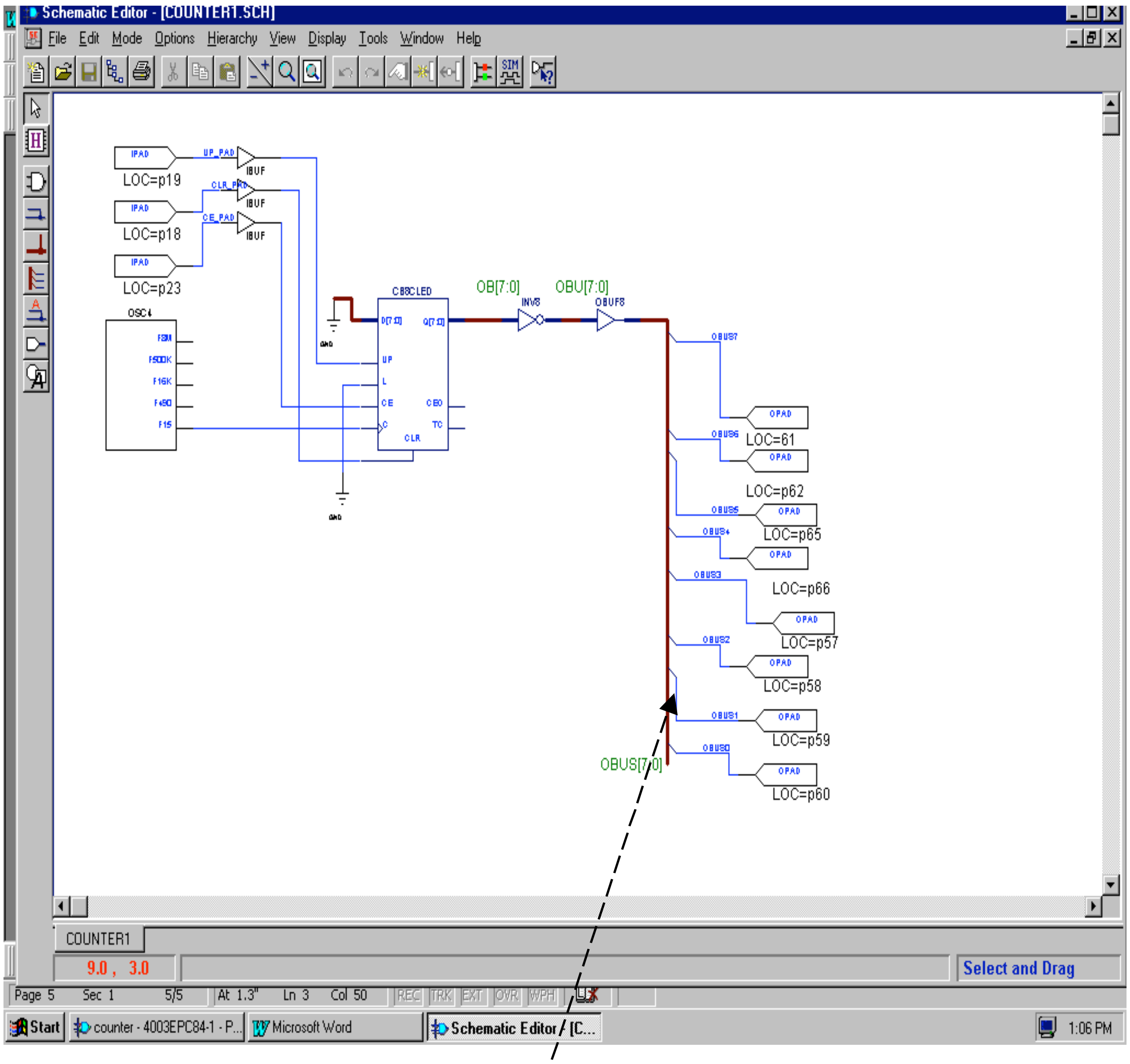
Problem 3. Design an 8-bit counter triggered by an oscillating signal. Use the internal 5-frequency signal generator module, **OSC4**, to trigger (clock) the counter. Use the 8-bit loadable cascadable bidirectional binary up/down counter with clock enable and clear, **CB8LED**, to count the events. Direct the output to pins # 61, 62, 65, 66, 57, 58, 59 and 60, which drive 8 LED bars (D9-D16 row) on the Xilinx board. Download and test the design. Provide external control over the reset/clear, enable/disable and up/down inputs of the counter using switch SW3 (pins # 19, 20, 23, 24, 25, 26, 27 and 28) and the SW5/SPARE push button (associated with pin #18) on the Xilinx demo board.
For reference, the CB8CLED module, given below, has the following inputs and outputs:


Schematic Editor - [COUNTER1.SCH]
File Edit Mode Options Hierarchy View Display Tools Window Help
CB8CLED
OB[7:0]
INV8
D[7:0]
Q[7:0]
GND
UP
L
CE
CEO
C
TC
CLR
GND
COUNTER1
2.6 , 3.2
Select and Drag
Start
counter - 4003EPC84-1 - P...
Schematic Editor - [C...
10:43 AM

Inputs: **UP** = “1”/”0” = up/down;
D[7:0] – an 8-bit number that can be used to initialize the counter;
L = “1”/”0” = enables/disables the usage/download of **D[7:0]**
CE = “1”/”0” = clock enable/disable;
C = signal from internal or external clock;
CLR = “1” = counter reset (clear);

Outputs: **TC** = “1” indicates that the **T**erminal (1111 1111 for an up counter) Count has been reached;
CEO – to be connected to the **CE** input of the next counter in a cascade configuration;
Q[7:0] – the current number of counts accumulated (an 8-bit number);

The **OSC4** module has four outputs: F8M, F500K, F16K, F490 and F15 providing clock pulses (square-wave signal) with frequency 8 MHz, 500 KHz, 16 KHz, 490 Hz and 15 Hz, respectively. A possible design is given below.



Please, note that the design relies on the usage of both busses and wires. For reference, a bus is a named set of discrete signals (wires), BUS_NAME[X:Y]. The constituent signals are called bus members and are ordered from the left (X) to the right (Y) with the leftmost bit/signal treated as the most significant. In order to connect a single wire to a bus, you need to draw a wire that **starts or ends** at the bus line. A wire connected to a bus should bear the name of a signal that is a member of the bus.
Also, you may find useful the primitives GND = "0" and VCC="1". For more information, see the Xilinx Foundation Help Topics.