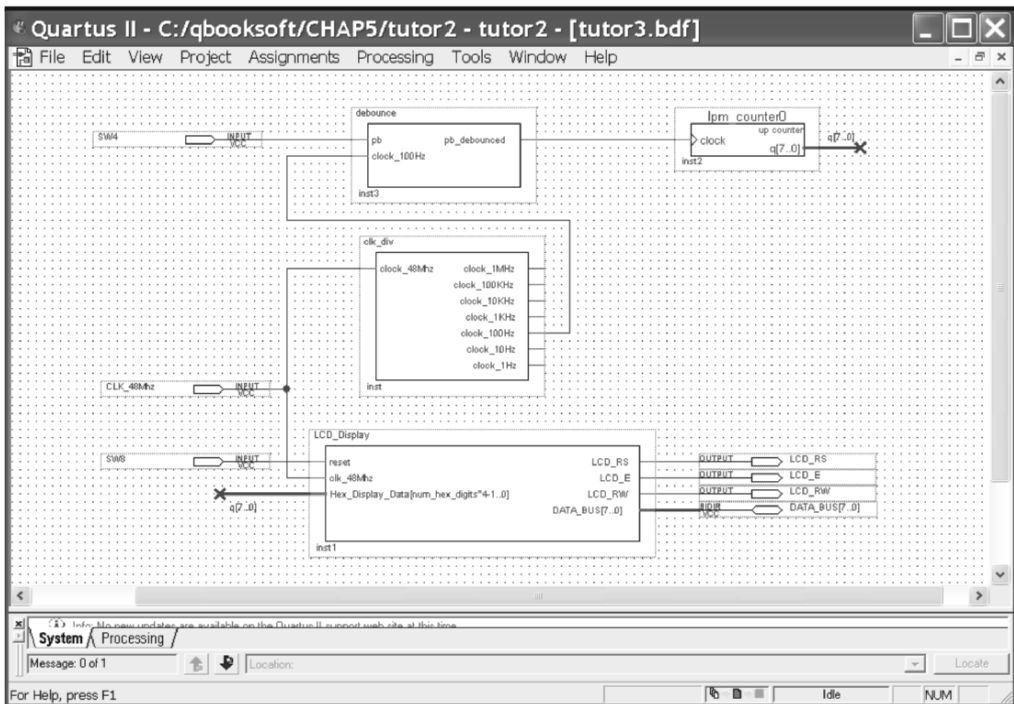


CHAPTER 4

Tutorial II: Sequential Design and Hierarchy



4 Tutorial II: Sequential Design and Hierarchy

The second tutorial contains a more complex design containing sequential logic and hierarchy with a counter and a Hex display. To save time, much of the design has already been entered. The existing design will require some modifications. Once again, any of the Altera educational FPGA boards can be used.

Once completed, you will:

- Understand the fundamentals of hierarchical design tools,
- Complete an example of a sequential logic design,
- Use the FPGAcore library designed for the FPGA boards,
- Use a hex display, pushbuttons, and the onboard clock,
- Use buses in a schematic, and
- Be able to perform automatic timing analysis of sequential circuits.

4.1 Install the Tutorial Files and FPGAcore Library for your board

Depending on your FPGA board type, slightly different I/O features are used in this tutorial as seen in Table 4.1. DE2 and UP3 boards will use their LCD display module to display a counter value in hexadecimal. On the DE1, UP2, and UP1 boards, two seven-segment LEDs will be used to display to counter value in hex. A slide switch, SW0, is used on the DE2 and DE1 for the count up operation, and pushbuttons on the other boards.

Table 4.1 FPGA I/O Devices used in the tutorial on the various FPGA boards.

I/O Device	DE1	DE2	UP3	UP2 & 1
SW4 – count	SW0	SW0	SW4	Flex PB1
SW8 - reset	KEY3	KEY3	SW8	Flex PB2
Hex Display	7-Segment LEDs HEX0 & HEX1	LCD module	LCD module	Flex 7-Segment LEDs
Clock	50Mhz	50Mhz	48Mhz	25Mhz

Locate the **\BOARD\chap4** directory on the DVD that came with the book. Each board type has a subdirectory with the required project files for that board. Copy all of the Chapter 4 tutorial files in this directory to your *drive:\mydesigns* directory or another subdirectory.

In the UP3 directory, A special version of the files for the larger 1C12 UP3 board is in the subdirectory **\UP3\1C12\chap4**. If you are using the UP2, a version of the files for the UP2 board is in the subdirectory, **\UP2\chap4**.

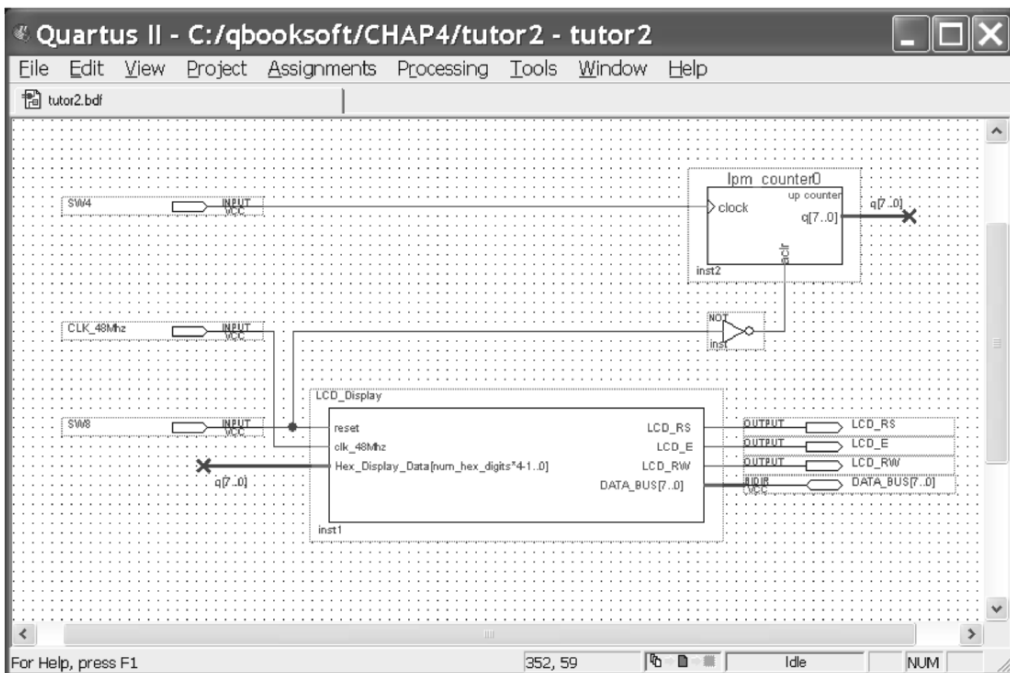


Figure 4.1 The tutor2.gdf schematic uses the LCD module on the DE2 and UP3 boards to display an 8-bit counter value in hex. DE1 and UP2 boards use two seven-segment LED displays instead.

4.2 Open the tutor2 Schematic

After setting up the files in your directory, select **File ⇒ Open Project ⇒ drive:\mydesigns\tutor2.qpf**. Open the top-level schematic by selecting **File ⇒ Open ⇒ drive:\mydesigns\tutor2.bdf** (not tutor2.gdf) and a schematic similar to Figure 4.1 should be displayed. This design has been partially entered to save time. This is an 8-bit counter design that outputs the counter value to a two digit hexadecimal display.

On the DE1 and UP2 version of the tutorial, you will see the counter value in two seven-segment LED displays and the DE2 and UP3 boards will use their LCD module.

Click on the lpm_counter0 symbol to activate the MegaWizard Plug-In Manager. The MegaWizard seen in Figure 4.2 can be used to create and edit megafunctions. In this case, you can see that lpm_counter0 is an 8-bit binary counter that counts up. You can click on the documentation button and then generate sample waveforms to view more details about the counter's operation. You can create new functions with the MegaWizard using **Tools ⇒ MegaWizard Plug-In Manager**. Close the MegaWizard window to continue.

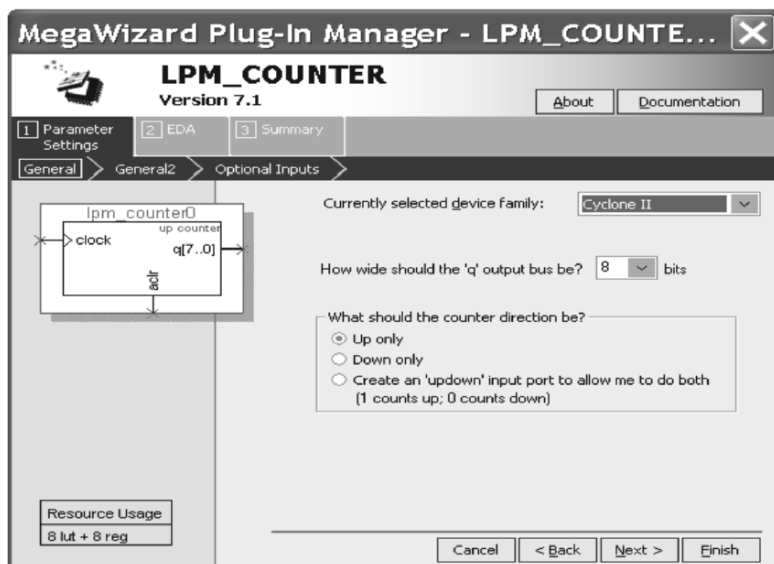


Figure 4.2 Lpm_counter0 MegaWizard edit window.

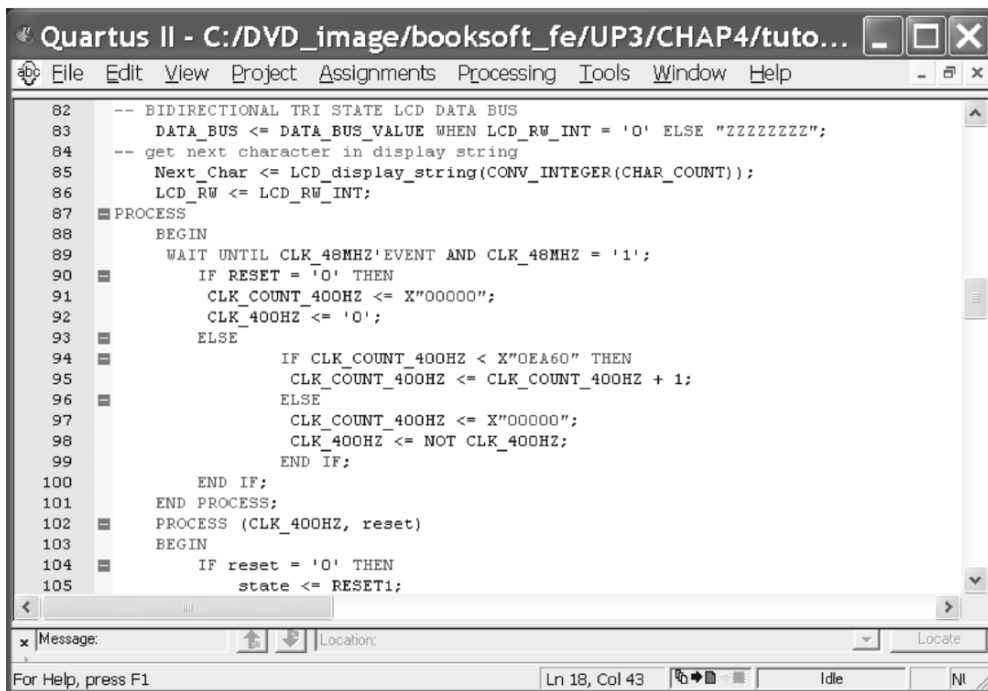
Special hardware blocks have been designed to support the easy use of the advanced I/O features found on the FPGA board. They include pushbuttons, LCD displays, keyboard, mouse, and video output. More details on all of these FPGAcore functions are provided in Chapter 5. The FPGAcore functions needed for this tutorial have already been placed in this project's directory. Several symbols from the FPGAcore library appear in the project library and are available to be entered in a design.

As an alternative to copying these files to each project's directory, under **Project ⇒ Add/Remove Files in Project**, in the left column you can click on **Libraries** and enter another path to an external library such as the FPGAcore library.

4.3 Browse the Hierarchy

In engineering, the principle of functional decomposition is normally used in large designs. Complex designs are typically broken into smaller design units. The smaller design units are then more easily understood and implemented. The smaller designs are interconnected to form the complex system. The overall design is a hierarchy of interconnected smaller design units. This also promotes the re-use of portions of the design.

The current schematic is a view of the top level of the design. In this design, the problem was decomposed into a module (design unit) or symbol with logic for a counter and another design unit to display the count. Each symbol also has an internal design that can be any combination of another schematic, megafunction, VHDL, or Verilog file.



```

82  -- BIDIRECTIONAL TRI STATE LCD DATA BUS
83  DATA_BUS <= DATA_BUS_VALUE WHEN LCD_RW_INT = '0' ELSE "zzzzzzzz";
84  -- get next character in display string
85  Next_Char <= LCD_display_string(CONV_INTEGER(CHAR_COUNT));
86  LCD_RW <= LCD_RW_INT;
87  PROCESS
88  BEGIN
89      WAIT UNTIL CLK_48MHZ'EVENT AND CLK_48MHZ = '1';
90      IF RESET = '0' THEN
91          CLK_COUNT_400HZ <= X"000000";
92          CLK_400HZ <= '0';
93      ELSE
94          IF CLK_COUNT_400HZ < X"0EA60" THEN
95              CLK_COUNT_400HZ <= CLK_COUNT_400HZ + 1;
96          ELSE
97              CLK_COUNT_400HZ <= X"000000";
98              CLK_400HZ <= NOT CLK_400HZ;
99          END IF;
100     END IF;
101     END PROCESS;
102     PROCESS (CLK_400HZ, reset)
103     BEGIN
104         IF reset = '0' THEN
105             state <= RESET1;

```

Figure 4.3 Internal VHDL code for LCD_Display function.

On schematic for the DE2 or UP3, double click on the **LCD_Display** symbol to see the underlying VHDL code that describes the internal operation of the LCD_Display block. As shown in Figure 4.3, it contains a complex state machine that sends commands and ASCII character data to the LCD controller.

DE1 and UP2 users should click on the **DEC_7SEG** symbol. It contains VHDL code with a case statement to implement the seven-segment LED decoder hardware for the count display.

As an alternative, the module could be designed in Verilog or even at the gate level using basic logic symbols (if you had infinite time and patience to work at that low of a level!). Close the VHDL text editor and return to the graphic editor.

To see the overall hierarchy of the design, select **View ⇒ Utility Windows Project Navigator** and make sure the **Hierarchy** tab is selected. After expanding this window as seen in Figure 4.4, note that the tutor2 schematic is comprised of two symbols.

For the DE2 and UP3, the LCD_Display symbol is used in the design to output the count to the LCD display. The lpm_counter0 symbol contains the 8-bit binary counter. If you click on the “+” block on the lpm_counter symbol, you will see that it contains an lpm_counter megafunction. In this case, the design hierarchy is three levels deep. After examining the hierarchy display window, close it and return to the graphic editor window that contains the tutor2 schematic.

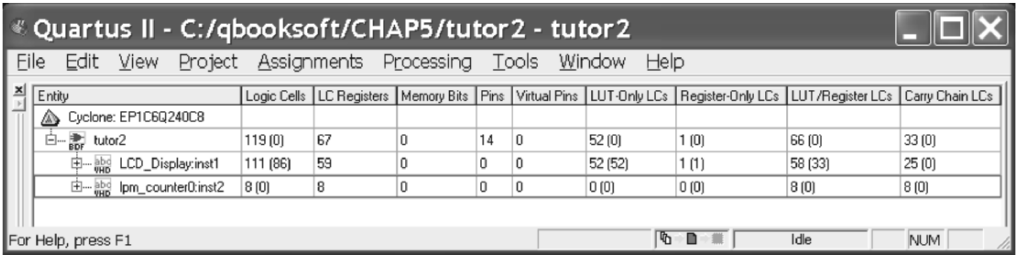


Figure 4.4 Hierarchy display window for the tutor2 design.

4.4 Using Buses in a Schematic

In Figure 4.5, find the heavy purple lines flowing out of the lpm_counter0 symbol in the upper right corner and into the LCD_Display (DE2 and UP3) or DEC_7SEG (DE1 and UP2) symbol in the lower left corner. This is an example of a bus. A bus is just a parallel collection of numbered bits. The bus is labeled q[7..0] indicating the bus has eight signals (bits) named q[7], q[6] ... q[0]. q[3..0] would be the low four bits of the q[7..0] bus and q[7..4] would be the high four bits. The q[7..0] bus sends the counter's eight output bits to the appropriate display function for each FPGA board.

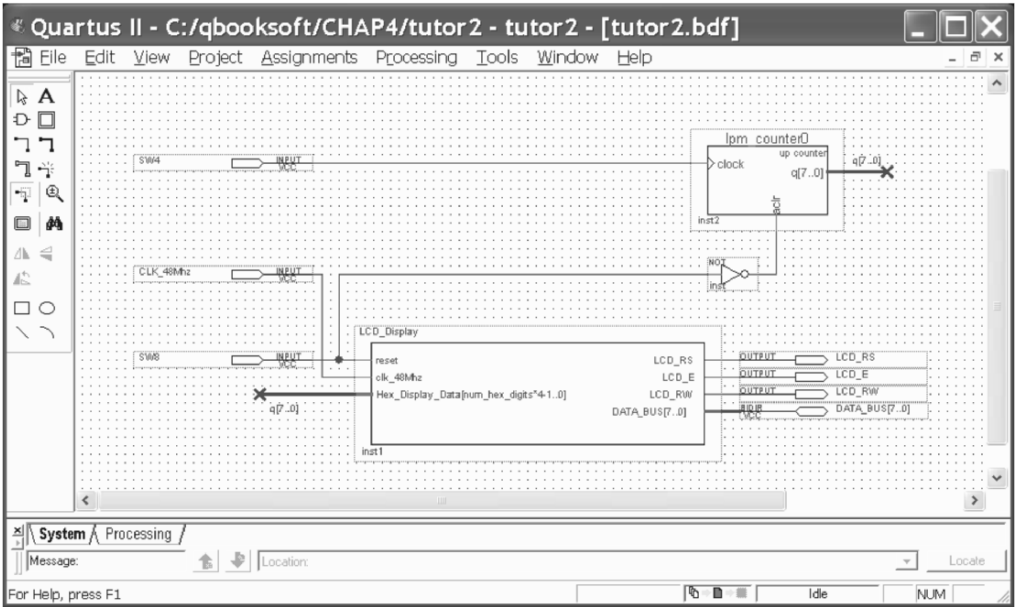


Figure 4.5 Enlarged view of tutor2 design showing q[7..0] bus connections.

To connect single node lines to a bus, it is first necessary to assign a name such as q[7..0] to the bus. Then the node line that needs to connect to a bus line is

given the name of one or more of the bus elements. As an example, the counter output MSB signal line is labeled $q[7]$. To label a bus or node, right click on the node or bus line and select **Properties**. You can then type in or edit the name. When signal lines have the same name, they are automatically connected in the graphic editor. A physical node line connecting a node and a bus with the same name is optional. Leaving it out often times makes a complex schematic easier to follow since there will be fewer lines crossing on the schematic. Node and bus names must be assigned first when connecting a node to a bus.

4.5 Testing the Pushbutton Counter and Displays

Compile the design with **Processing** \Rightarrow **Start Compilation**. Wait a few seconds for the “Full Compilation was successful” message to appear. Select **Processing** \Rightarrow **Classic Timing Analyzer Tool**. This counter circuit is a sequential design. The primary timing issue in sequential circuits is the maximum clock rate. Whenever you compile, a timing analysis tool automatically runs that will determine the maximum clock frequency of the logic circuit.

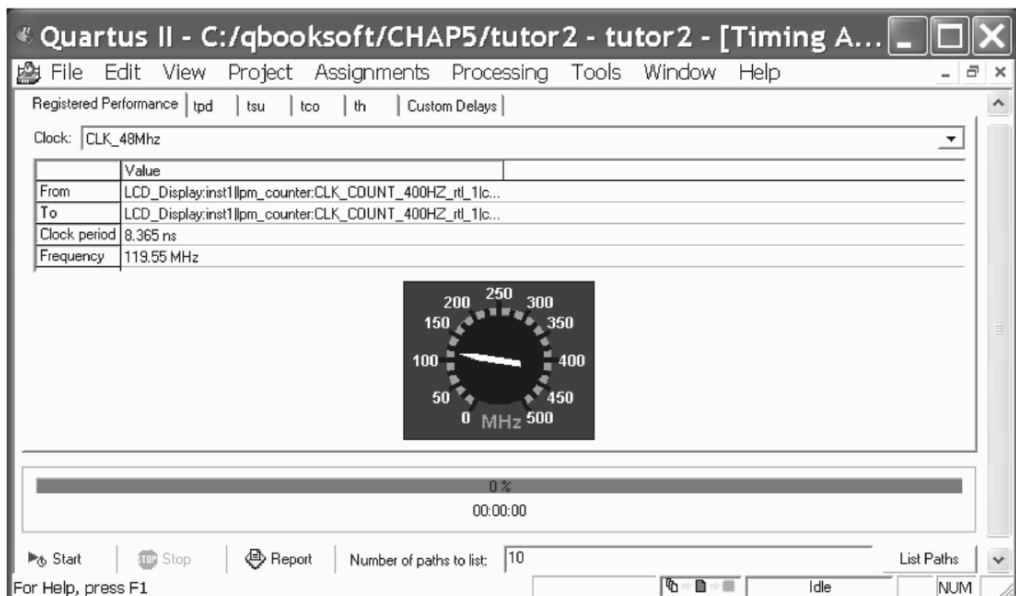


Figure 4.6 Timing analysis of a Sequential Circuit

The Timing Analyzer shows the maximum clock frequency of this logic circuit to be approximately 120 MHz. Clock rates you will obtain will vary depending on the FPGA device type, the complexity and size of the logic circuits, the speed grade of the chip, and the CAD tool version and settings. In this design, the clock is supplied by a manual switch input so a clock input of only a few hertz will be used for the counter.

Since the FPGA's clock input on the DE2 and UP3 is only 50 or 48 MHz for the LCD_Display core, this simple counter cannot be overclocked. Close and exit the timing analyzer.

4.6 Testing the Initial Design on the Board

Download the design to the FPGA board. If you need help downloading to the board, refer back to Sections 1.5 to 1.8 depending on board type.

The name of the specific switch assigned for each FPGA board is shown in Table 4.2. Hit or turn on the count switch several times to clock the counter and watch the count display as it counts up. When the switch is hit, it will occasionally count up by more than one. This is a product of mechanical bounce in the switch. A switch contains a metal spring that actually forces contact and bounces several times before stabilizing. The high-speed logic circuits will react to the switch contact bounce just as if several clock signals have occurred. This makes the counter count up by more than one randomly.

Table 4.2 Location of Switches on each FPGA board.

I/O Device	DE1	DE2	UP3	UP2 & 1
SW4 – count up	SW0	SW0	SW4	Flex PB1
SW8 - reset	KEY3	KEY3	SW8	Flex PB2

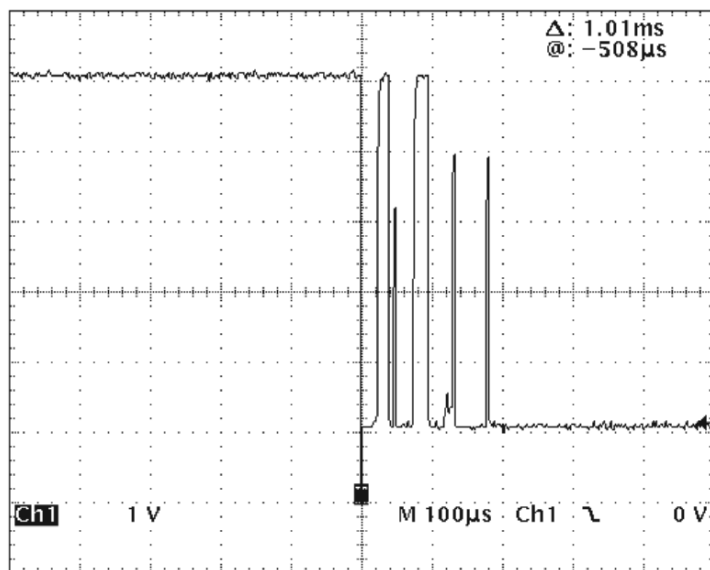


Figure 4.7 Oscilloscope display of switch contact bounce.

The actual output of the switch as it appears on a digital oscilloscope is shown in Figure 4.7. When the switch is hit, a random number of pulses appear as the switches mechanical metal contacts bounce several times and then finally stabilize. Several of the pulses will have a voltage and duration long enough to generate extra clock pulses to the counter. An FPGA will respond to pulses in the nanosecond (ns) range, and these pulses are in the microsecond (μ s) range.

This problem occurs with all slide switches or pushbuttons in digital designs. If the pushbutton is a double-pole double-throw (DPDT; i.e., has both an ON and an OFF contact), an SR latch is commonly used to remove the contact bounce. The pushbutton on the FPGA boards are single pole single throw (SPST), so a time averaging filter is used. This example demonstrates why designs must be tested on actual hardware after simulation. This problem would not have shown up in a simulation. Verify that the reset switch resets the display and the counter. DE1 and DE2 boards already have a hardware debounce circuit on the four pushbuttons and that is why the undebounced slide switch (SW0) was used instead. The other boards do not have this built-in hardware debounce circuit on their pushbuttons.

4.7 Fixing the Switch Contact Bounce Problem

For the hardware implementation to work correctly, the switch contact bounce must be removed. A logic circuit that filters the pushbutton output using a small shift register can be added to filter the output. This process is called switch debouncing. Using the right click, insert the symbol **debounce** from the project library into the schematic.

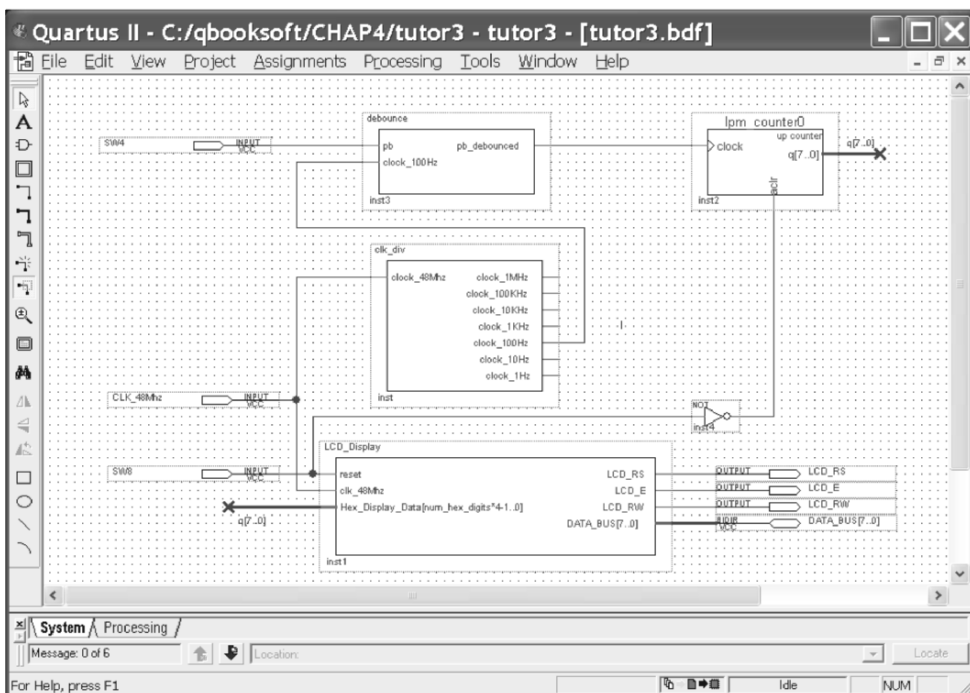


Figure 4.8 Modified tutor2 design schematic.

Disconnect the pushbutton from the `lpm_counter0`'s clock pin and connect it to the pushbutton input pin, `PB`, on the debounce symbol. Now connect the `PB_DEBOUNCED` pin to the `lpm_counter0`'s clock pin. The debounce circuit needs a 100Hz clock signal for the time averaging filter. The clock needed is much slower than a 25-50MHz system clock, so a clock prescaler is needed. A clock prescaler is a logic circuit that divides a clock signal.

Add the `clk_div` symbol from the project library to the schematic. Connect the 100Hz input pin on the debounce symbol to the 100Hz output pin on the clock prescaler. Connect the 48MHz clock input on the `clk_div` symbol to the `clk_48MHz` (or `clk_50Mhz` on DE1 and DE2) input pin.

The internal VHDL design in the debounce module generates the switch debounce circuit. The debounce circuit contains a 4-bit shift register that is clocked at 100Hz. The shift register shifts in the inverted pushbutton output. When any of the four bits of the shift register (i.e., four 10 ms time-spaced samples of the pushbutton's output) are High the output of the debounce circuit changes to High. When all four bits of the shift register are Low the output goes Low. This delays the High to Low change until after the switch contact bounce stops.

If the external input signal being counted or used for a reset was a fast clock that was not synchronized to the internal FPGA clock, another problem with the simple counter circuit could appear that would cause problems. If the external input signal changes right at a clock edge, it could violate the flip-flop setup and hold times and cause unpredictable results (metastability). The traditional solution for this problem is to feed any external input signals through two cascaded D flip-flops that are clocked by the system clock. This reduces the probability of a such an error to only once in several years even for a rapidly changing clock input signal. This circuit is sometimes called a synchronizer. A very slowly changing signal, such as a user input switch, has a very low probability of such an event occurring even without a synchronizer circuit, but it could still occur and would require a synchronizer in mission critical applications where such a failure could be catastrophic.

4.8 Testing the Modified Design on the FPGA Board

Verify that your schematic has the same connections to the new debounce and clock divide symbols as seen in Figure 4.8. Compile the design and download the design to the board again. Hit the count switch several times to clock the counter and watch the LCD display as it counts up. It should now count up reliably by one whenever the pushbutton is hit. Hit the reset switch and verify that the count resets to zero.

The FPGAcore functions `LCD_Display` or `DEC_7SEG`, `clk_div`, and `debounce` will be useful in future design projects using the FPGA board. They can be used in any VHDL, Verilog, or schematic designs by using the graphical editor and FPGAcore symbols or by using an HDL component instantiation statement.



ALL SOURCE CODE FOR THE BOOK'S DESIGNS IS AVAILABLE ON THE DVD.
MATERIALS CAN BE FOUND IN THE `\BOOKSOFT_FE\BOARD\CHAPX`
DIRECTORIES FOR EACH SPECIFIC *BOARD* (I.E., DE1, DE2, UP3, UP2)

4.9 Laboratory Exercises

1. Simulate the initial design without the switch debounce circuit by setting up an initial reset pulse and a periodic 200 ns clock input in the simulator. In sequential simulations, turn on the **setup and hold time violation detection** simulator setting option before running the simulator. This will check for flip-flop timing problems that would otherwise go undetected in the simulation. Adjust the reset pulse so that it changes right before the clock edge and run another simulation to see if you can produce a setup or hold violation.
2. Modify the counter circuit so that it counts down or up depending on the state of a switch input. See Table 2.4 for the pin assignment for the new switch input.
3. Modify the counter circuit so that it parallel loads a count value from the four switches on the FPGA board when PB2 is pushed. Zero out the low four counter bits during a load. Since the switch inputs are only used when PB2 is hit, they do not need to be debounced. See Table 2.4 for the pin assignments for the new switch inputs.
4. Build a stopwatch with the following modifications to the design. Disconnect the counter clk line and connect it to the clock_10hz pin on the clock_div symbol. Clock a toggle flip-flop with the pb_debounced output. A toggle flip-flop, tff, can be found in the prim symbol library. A toggle flip-flop's output changes state every time it is clocked. Connect the output of the toggle flip-flop to a new count enable input added to the counter with the megawizard. The count should start and stop when PB1 is hit. Elapsed time in tenths of seconds should be displayed in hexadecimal. Pushing PB2 should reset the stopwatch.
5. The elapsed time in the stopwatch from problem 3 is displayed in hexadecimal. Replace the counter with two cascaded binary-coded-decimal (BCD) counters so that it displays the elapsed time as two decimal digits.

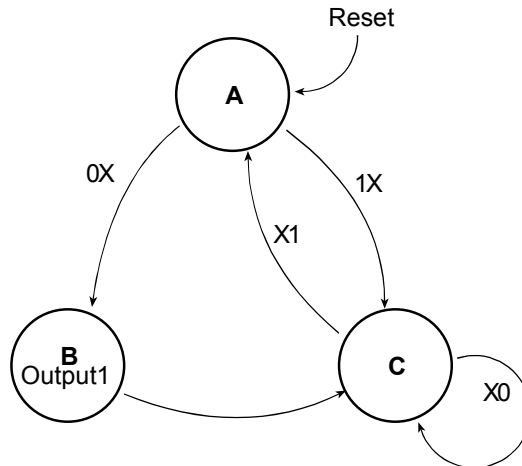
6. Build a watch by expanding the counter circuit to count seconds, hours, and minutes. The two pushbuttons reset and start the watch.
7. Replace the `lpm_counter0` logic with a VHDL or Verilog counter design, simulate the design, and verify operation on the FPGA board. Read Chapter 5 and note the example counter design in section 6.10.
8. Draw a schematic, develop a simulation, and download a design to the FPGA board that uses the LCD displays for outputs and the DIP switch for input, to test the 74161 4-bit TTL counter function found in the `/others/maxplus2` symbol library. Use the DIP or slide switch to provide four inputs for a parallel load of the count. Use a debounced pushbutton input for the clock. Use the second pushbutton for the load input.
9. Draw a schematic, develop a simulation, and download a design to the FPGA board to test the following functions that can be created with the MegaWizard:

LPM_ADD_SUB:	a 2-bit adder/subtractor; test the add operation
LPM_ADD_SUB:	a 2-bit adder/subtractor; test the subtract operation
LPM_COMPARE:	compare two 2-bit unsigned numbers
LPM_DECODE:	a 4 to 16-bit decoder
LPM_CLSHIFT:	a 4-bit shift register
LPM_MULT:	a 2-bit unsigned multiply

The LPM megafunctions require several parameters to specify bus size and other various options. For this problem, do not use pipelining and use the unregistered input options. Refer to the online help files for each LPM function for additional information. In the **enter symbol** window, use the megawizard button to help configure LPM symbols. Use the FPGA boards switches for four inputs as needed and display the output in hex on the two seven-segment displays. Use a debounced pushbutton input for the clock, if one is required. Use the second pushbutton for a Clear or Reset input. Use the timing analyzer to determine the worst-case delay time for each function.

10. Draw a schematic and develop a simulation to test the `LPM_ROM` megafunction. Create a sixteen word ROM with eight data bits per word. Specify initial values in hex for the ROM in a memory initialization file (*.mif) file. The contents of each memory location should be initialized to four times its address. See MIF in the online help for details on the syntax of a MIF file. Enter the address in four switches and display the data from the ROM in the two seven-segment LEDs or the LCD. Determine the access time of the ROM.

11. Using gates and the DFF part from the primitives/storage library, design a circuit that implements the state machine shown below. Use two D flip-flops with an encoded state.



For the encoded states use A = "00", B = "01", and C = "10". Ensure that the undefined "11" state enters a known state. Enter the design using the graphical editor. Develop a simulation that tests the state machine for correct operation. The simulation should test all states and arcs in the state diagram and the "11" state. Use the **Processing** ➡ **Classic Timing Analyzer Tool** option to determine the maximum clock frequency on the Cyclone device. Use an asynchronous reset.

12. Repeat the previous problem using one-hot encoding. Recall that one-hot encoding uses one flip-flop per state, and only one flip-flop is ever active at any given time in valid states. The state encoding for the one-hot state machine would be A = "100", B = "010", and C = "001". Start with a reset in the simulation. It is not necessary to test illegal states in the one-hot simulation. One-hot state machine encoding is recommended by many FPGA device manufacturers.