

ADVANCED QUARTUS II TUTORIAL

OVERVIEW

This tutorial illustrates several advanced topics that were not covered in Lab 2. These topics include creating symbols for use in a hierarchical design, using Altera parameterized modules, such as a ROM, and initializing ROM contents.

USING PARAMETERIZED MODULES

Altera's Library of Parameterized Modules (LPM) contains many high-level components that can be configured to meet specific design goals. You will probably find the LPM components very convenient and easy to use.

In this tutorial, you will design a circuit that computes the Fibonacci sequence using an adder and registers. (The Fibonacci sequence is 0, 1, 1, 2, 3, 5, 8, ...) Recall that a register is simply a group of flip-flops with a common clock signal. The data path will be 8-bits wide (therefore the data is 8-bits wide as well) so that the circuit can be easily tested using the Altera Education Board. If you thought a sequence of numbers could never be very interesting or have any applications, read up on the Fibonacci sequence—google it or check out <http://www.branta.connectfree.co.uk/fibonacci.htm>

- Open a new schematic and enter the symbol **lpm_add_sub** from the mega_lpm library. A dialog box should come up to allow you to configure the parameters and ports. Click on each parameter from the list and use the pull-down arrow in the dialog box to select the desired value for the parameter. Enter the following values for the parameters:

```
LPM_DIRECTION = "ADD"  
LPM_REPRESENTATION = "UNSIGNED"  
LPM_WIDTH = 8
```

Any other parameters, such as **LPM_PIPELINE** or **ONE_INPUT_IS_CONSTANT**, should be left unassigned. For an explanation on how the **lpm_add_sub** component works and how each of the parameters can be used, select the **Help on LPM_ADD_SUB** button in the dialog box.

Configure the ports so that **cin**, **cout**, **dataa[]**, **datab[]** and **result[]** are *Used* and all other ports are *Unused*. When you have configured all ports and parameters, press the OK button.

If you need to change any parameters or port declarations after you have closed the dialog box, click on the **lpm_add_sub** component on your schematic and select **Edit Ports/Parameters...** (Symbol Menu). This will re-open the dialog box so that you can make corrections. Another way to re-open the dialog box is to double-click the left-mouse button (LMB) on the parameters box.

The adder will be used to add the last two numbers in the sequence to produce the next number. We will also need two 8-bit registers in order to store the last two numbers in the sequence. We could use a standard D flip-flop and let the Altera software create a "primitive array". However, the standard D flip-flop only has an asynchronous reset (resets can happen at any time) and in our design we would like to synchronously clear the registers (synchronous reset—the reset signal can affect the output only during the active clock transition edge) when a carry-out is generated by the adder. We can use another LPM component to design a D flip-flop with both asynchronous and synchronous clear inputs.

- Enter the symbol **lpm_dff** from the mega_lpm library. Set the parameters such that **LPM_WIDTH = 8** and the other parameters are unassigned. Configure ports **aclr**, **clock**, **data[]**, **q[]** and **sclr** as *Used* and all other ports as *Unused*.

- Make a copy of the **lpm_dff** that you just configured and place the components as shown in Figure 1. Also, add the components **not**, **dff**, **input** and **output** and complete the schematic as shown in Figure 1. Note that buses are shown with thick lines and are labeled as NAME[MSB..LSB].
- Label the output bus with your first name. I.e. FRANK[7..0]. Also, label the output of the lpm_add_sub component with your initials followed by _SUM[7..0]. Note that wires which have the same name on the same level of hierarchy are connected. *Remember to keep the wire labels close to the wires that they label!* Otherwise, the Altera Compiler may not interpret the text as a valid net label.
- Save & Check your file to make sure that there are no errors. Then compile your circuit with the following options selected: **Functional SNF Extractor** and **Preserve All Node Name Synonyms** (Processing menu). (You must turn *off* the **Timing SNF Extractor** before you can select the **Functional SNF Extractor**.) Once the options are selected, click on the **Start** button to compile your circuit. You must fix any compile errors before proceeding to the next step.
- Open the Waveform Editor and select the Nodes to watch during simulation. You can use **Enter Nodes from SNF...** (Node menu) to select nodes such as RESET, CLK and the output signal with your name. Some other nodes are difficult to find in the SNF list. Instead, you can select **Insert Node** (Node menu) and type the name of a node. Try this for entering nodes A[7..0], CIN, COUT and the output of the lpm_add_sub that has your initials in the name.
- Set the **Grid Size** (Options menu) to 20 ns.
- Select the CLK node with the left-mouse button so that it is highlighted. Then select **Overwrite>Clock** from the Edit pull-down menu. Specify a clock period of 40 ns.
- Assert the RESET node high for the first clock cycle. (Note that the RESET is active-high in this design.)
- Save your waveform file as an .scf file with the same prefix as your schematic and project. Open the Simulator (QUARTUS II menu) and click on the Start button to run a functional simulation. Verify your waveforms with the results shown in Figure 2.

CREATING A SYMBOL

You will now create a symbol for the circuit you have just designed so that it can be used as a component in a more complex design.

- Open the graphic design file (.gdf) for your circuit. Select **Create Default Symbol** (File menu). This will create a symbol file with the same name as your design and the .sym extension.

CREATING A HIERARCHICAL DESIGN

You will now design a top-level schematic that uses the component you just created.

- Close any open windows such as the Graphics Editor, Simulator, Waveform Editor, etc. Create a new schematic file (.gdf) for your top-level design. Name the file and save it in your working directory. Change the project to this new name.
- Place an instance of your component in your top-level schematic. Your symbol should have RESET and CLK input ports and an output port named after you.

USING A ROM

In this design, we will use a ROM to implement a combinational logic circuit that converts a 4-bit hex number to the corresponding 7-segment display driver signals. We will illustrate how the ROM can be initialized using a Memory Initialization File (.mif) which can be created using a simple text editor.

- Select an **lpm_rom** component from the mega_lpm library. Select **address[]** and **q[]** as the only ports to be *Used*. Also, configure the parameters as shown below.

```
LPM_ADDRESS_CONTROL = "UNREGISTERED"  
LPM_OUTDATA = "UNREGISTERED"  
LPM_FILE = "hex7seg.mif"  
LPM_WIDTH = 8  
LPM_WIDTHAD = 4
```

Note that you must type the quotes around the file name and the case of the name must match the file that you create. Once the ports and parameters are configured, press OK.

- Using Copy and Paste, place a second, identical ROM component on your schematic.

Next we will create the .mif file. The format of the MIF file is quite simple and is explained in Altera's on-line HELP documentation. The default radix is hexadecimal. Our ROM will contain 16 words (also called "16 words deep") and each word will be 8 bits wide. The data represents active-low 7-segment display drivers where the MSB corresponds to signal *a*, MSB-1 corresponds to *b*, etc. and the LSB corresponds to the decimal point which should always be off. For example, when the value 0 is the address value, we would like to display the symbol 0 on the 7-segment display. This requires all segments to be on (low) except for *g* and the decimal point, giving a hex value of 03.

```

----- hex7seg.mif -----
DEPTH = 16;
WIDTH = 8;
CONTENT
BEGIN
0 : 03;
1 : 9F;
2 : 25;
3 : 0D;
4 : 99;
5 : 49;
6 : 41;
7 : 1F;
8 : 01;
9 : 19;
A : 11;
B : C1;
C : 63;
D : 85;
E : 61;
F : 71;
END;
-----

```

CLOCK GENERATION

The Altera UP1 Education Board has a 25.175 MHz crystal oscillator that drives global clock input **pin 83** of the EPM7128S device and global clock input **pin 91** of the EPF10K20 device. Since the EPM7128S device does not support memory devices such as ROM we will use the EPF10K20 for this design. We will need to divide the clock signal down to a frequency on the order of 1 Hz so that we can observe the Fibonacci sequence on the 7-segment displays. We can use a counter from the LPM library to accomplish this.

- Select **lpm_counter** (mega_lpm library). Configure two ports, **clk** and **q[]**, as *Used* and all other ports as *Unused*. Set the LPM_DIRECTION parameter to "UP" and the LPM_WIDTH to 24. Then press the OK button.

You will use the most significant bit of the counter as the clock signal for the Fibonacci subcircuit. The frequency should be approximately $(25 \text{ MHz} / 2^{24}) = 1.5 \text{ Hz}$. After the design has been compiled with the **Timing SNF Extractor**, we will use the Timing Analysis tool to verify that the counter can operate with a clock frequency of 25 MHz.

TRI-STATES

Based on the state of a push-button switch input, this design will display on the 7-segment displays either the value of eight input switches or the Fibonacci sequence. We will use tri-state buffers in order to multiplex these different output functions.

- Select **tri** from the primitive library. Although this primitive represents a single tri-state buffer, it can be connected to a bus and the Altera compiler will automatically generate a "primitive array" of the same width as the bus. Place a tri buffer at the output of the Fibonacci subcircuit and another connected to an **input** primitive, as shown in Figure 3.

There are some very important guidelines for designing with tri-state buffers using Altera tools. **Altera devices do not have tri-state buffers for driving internal logic.** When tri-state buffers are used to multiplex signals, as in our current design, QUARTUS II will substitute a multiplexer made up of combinational logic for the tri-state gates. Bidirectional signals, however, **cannot** be implemented in the internal Altera logic due to the lack of internal tri-state buffers. Altera devices *do* have tri-state buffers in the I/O Elements so that bidirectional I/O pins with tri-state output capability can be implemented. For more information on using tri-state buses in Altera devices, check the documentation available on the Altera Web page:

http://www.altera.com/support/examples/ged/tri_state.html

PIN AND DEVICE ASSIGNMENTS

- Add the remaining components and connect the circuit as shown in Figure 3. Select **Line Style** (Options menu) to draw a bus between the input switches, SW[7..0], and the tri component.
- Label the output of the Fibonacci circuit with *your* name and the output of the tri-state multiplexer with *your* initials.

We will now assign the device type and pin locations that correspond to the Altera Education Board.

- From the Graphics Editor, select **Device...** (Assign menu). Choose FLEX10K as the Device Family. Make sure that the option "Show Only Fastest Speed Grades" is not selected and select device **EPF10K20RC240-4.**

The pushbutton switches are active low and are each pulled-up through a 10 K ohm resistor. The pushbutton switches have the following pin assignments.

<u>Pushbutton switch</u>	<u>Pin number</u>
PB1	28
PB2	29

The pin assignments for the eight dipswitch inputs are:

<u>Switch</u>	<u>Pin number</u>
SW-1	41
SW-2	40
SW-3	39
SW-4	38
SW-5	36
SW-6	35
SW-7	34
SW-8	33

The pin assignments for the 7-segment display driver signals are shown below. **Note that these are different from the pin assignments for the EPM7128S.**

<u>Display segment</u>	<u>MS Digit pin number</u>	<u>LS Digit pin number</u>
a	6	17
b	7	18
c	8	19
d	9	20
e	11	21
f	12	23
g	13	24
decimal point	14	25

- Select **Pin/Location/Chip** (Assign menu) and, using the tables given above, assign pin numbers to all input and output ports in your design. For example,

```

OSC = 91
/RESET = 28
PB = 29
SW7 = 41 ... SW0 = 33
MSB7 = 6 ... MSB0 = 14
LSB7 = 17 ... LSB0 = 25

```

- Save and Check your design. (The project should be set to your top-level design.) When it checks without errors, compile the design using Timing SNF Extraction. If you receive warnings or errors, you can select the message and press the Help on Message button if you aren't sure how to fix the problem. You can also select the Locate button to locate the problem in your schematic.

VERIFYING TIMING

We should check that the 24-bit counter can actually operate with a clock frequency of 25 MHz.

- Select **Timing Analyzer** (QUARTUS II menu) and choose **Registered Performance** (Analysis menu). When the dialog box comes up, press the Start button. Since this design uses 2 clocks, OSC and Q23, the Timing Analyzer tools will calculate the maximum clock frequency for each clock. We only need to verify that the OSC signal can be ≥ 25 MHz. After the Timing Analyzer has run, choose OSC from the Clock pull-down box. You should see a maximum clock frequency in the range of 69 MHz.

VERIFYING THE FINAL DESIGN

It is not necessary to do a complete simulation on the top-level design. The 24-bit counter would take excessive computer resources to simulate completely. If you desire to simulate your design, you should test the 24-bit counter and the remainder of the circuit separately.

Your TA will configure the Altera Education Board and help you download your circuit to the EPF10K20 device. The SRAM Object File (.sof) is used to program the FLEX10K family of devices rather than the Programmer Object File (.pof) that is used to program the MAX 7000S family of devices. Thus, you will need to transfer the .sof file to the lab PC for downloading.

DOWNLOADING TO THE ALTERA BOARD

The procedure for downloading the programming file to the Altera board is as follows:

1. Set up the Altera board. Connect the ribbon-cable connector to the primary parallel port of the PC. Set the power supply to 7-8 Volts since the Altera board contains a +5V voltage regulator. Turn on the power supply.
2. Log-in to the PC so that you will have access to your workstation directory. Otherwise, you will need to use ftp or perhaps a diskette to transfer the programming file to the PC.
3. Run the Max+PlusII program by double-clicking on the icon or using the Start Menu. Open the Programmer tool by selecting it from the Max+PlusII window.
4. If the Hardware Setup Dialog Box pops up, set Hardware Type to **ByteBlaster**. Click OK.
5. From the JTAG menu, choose Multi-Device JTAG Chain Setup. In the dialog box, set Device Name: **EPF10K20**. Then choose the Select Programming File button. Select the Programming File type as: SRAM Object Files (*.sof). Then locate the programming file *.sof. If you have logged-in to the PC, you can set the drive to h: and browse your directories. Once you have located the file, double-click on it and then choose the Add button in the dialog box. You can then close the dialog box. Answer Yes to the question about whether it is ok to turn on the multi-device JTAG Chain Mode.
6. Choose Configure in the Programmer dialog box. The software should download your file to the Altera board. You can then test the circuit on the Altera board.
7. Remember to log out when you are done so that others don't gain access to the files in your workstation account. Also, delete the programming file from the PC if you stored it on the hard drive.

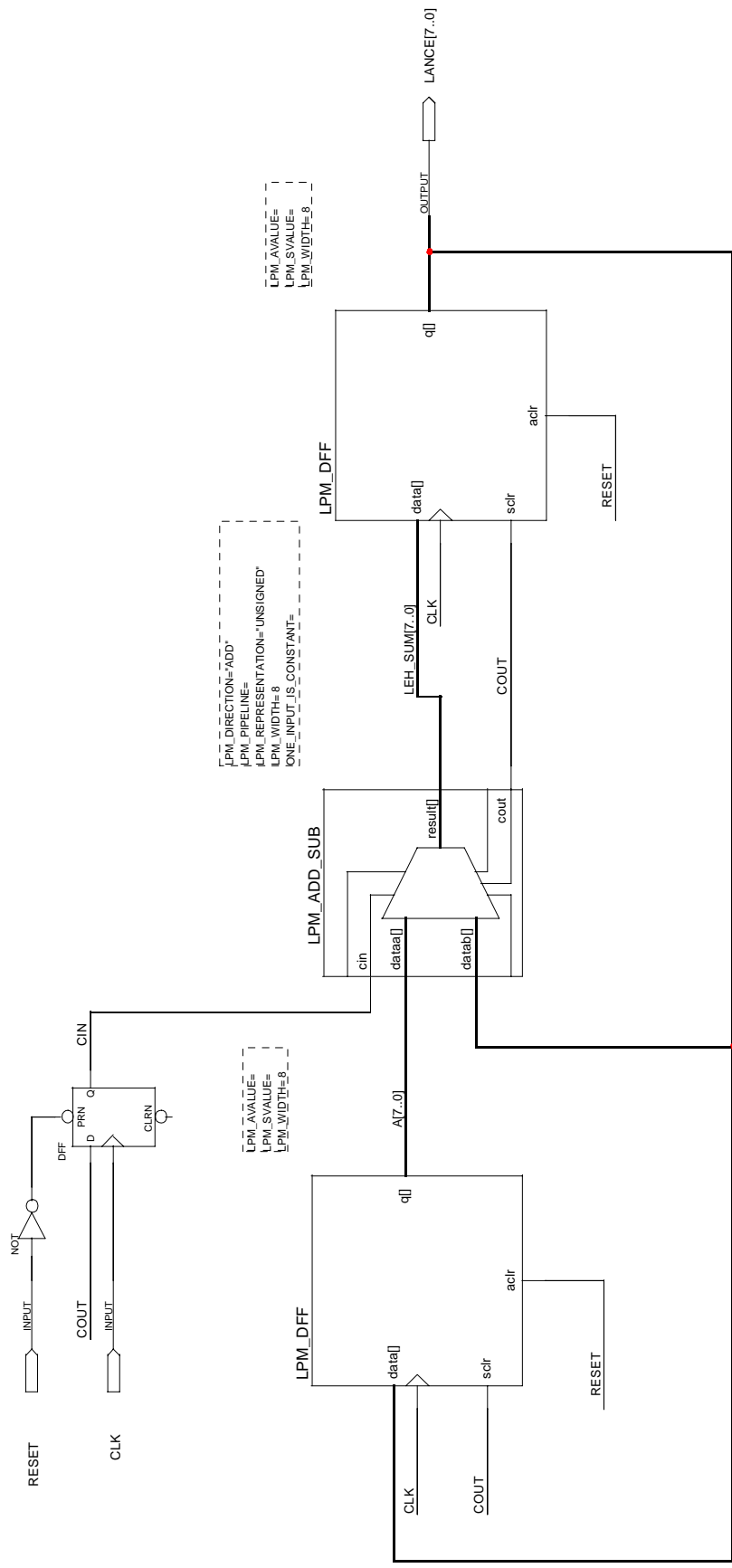


FIGURE 1 - FIBONACCI SUBCIRCUIT

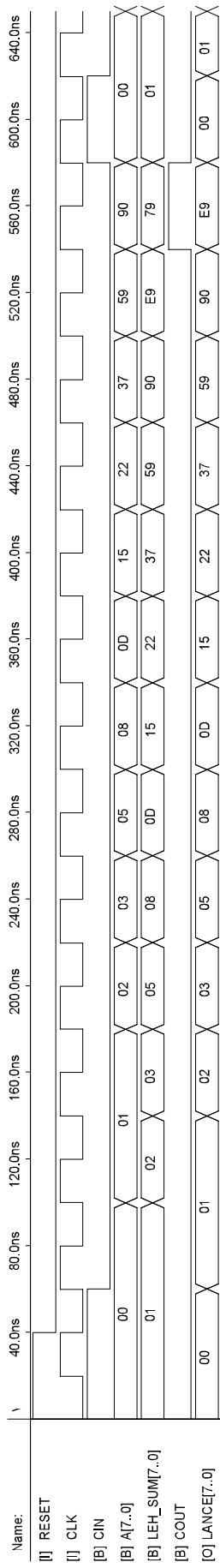


FIGURE 2 - FIBONACCI SUBCIRCUIT FUNCTIONAL SIMULATION

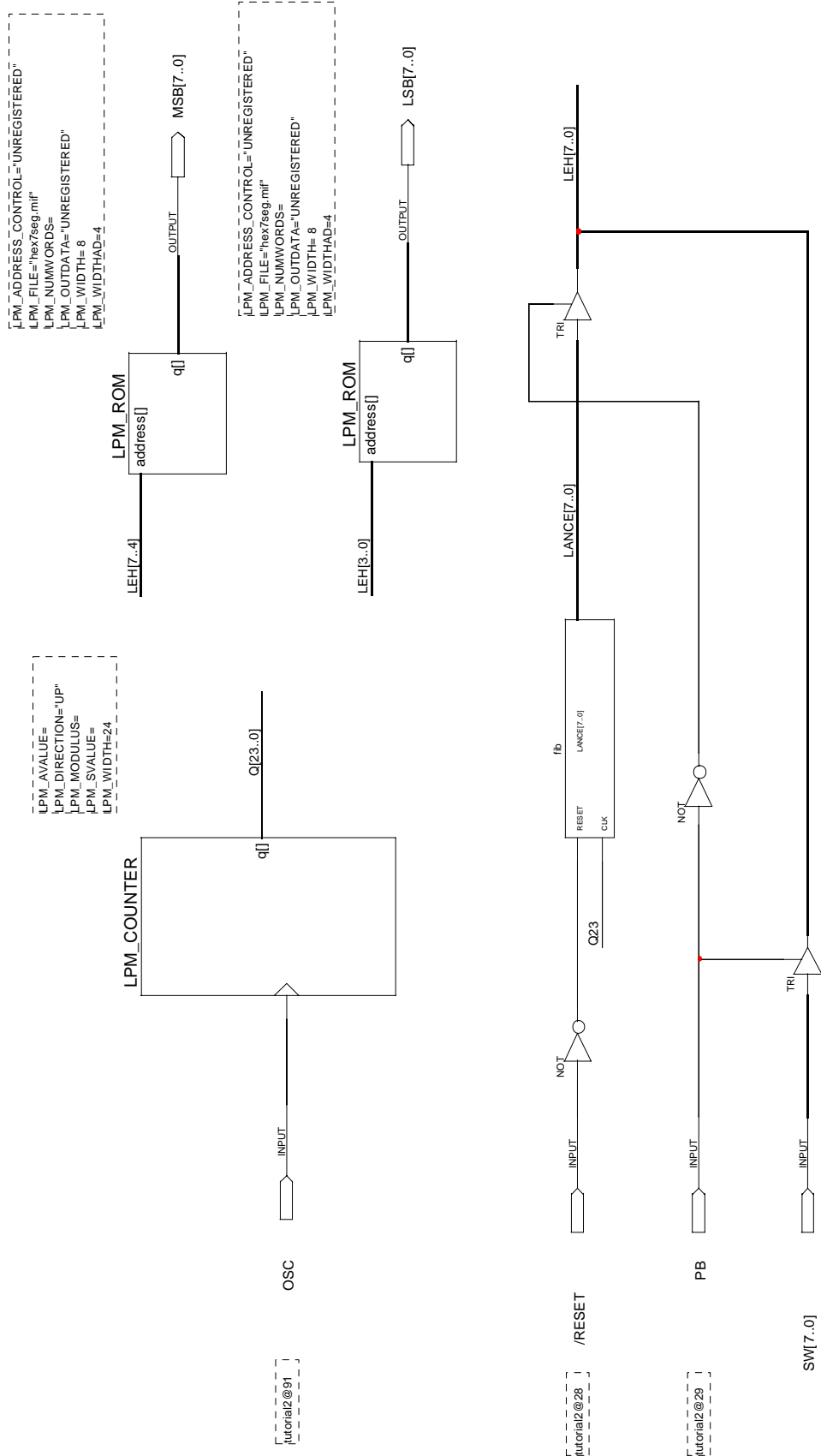


FIGURE 3 - TUTORIAL2 TOP-LEVEL CIRCUIT