

(Optional) Instructions for Schematic Design Entry for the Bit-Serial Logic Processor

These instructions are a tutorial for creating your top-level entity as a schematic diagram, rather than as a SystemVerilog module. Quartus II can generate Verilog code automatically based on your schematic. (SystemVerilog is not yet supported in the current version.) This is useful for small designs, but discouraged for larger designs as text designs are more maintainable and legible.

These instructions are optional; if you would rather enter your port maps in text form, you are free to do so. Note that the schematic design introduced here isn't directly compatible with the Testbench simulation described in IQT. 33-40. Conversion to Verilog is needed (you may follow the steps provided in IQT. 28 for conversion between schematic entry and HDL code, or simply use the provided 'processor.sv' module. However, we recommend reading through these instructions, since this procedure can save you time and frustration on the later labs and on the final project.

Preliminary Work:

Build the bit-serial processor project and include all the provided code, **except** for the "Processor (Main Module)" code.

Add New Source – Schematic:

In the **File** menu, select **New....** Select **Block Diagram/Schematic File** in the window that pops up and click **OK**. A blank schematic will be opened for editing.

Prepare Schematic Symbols:

In the **Project Navigator** window, click the **Files** tab. Expand the **Device Design Files** folder if it isn't already expanded; this will show all the design files currently in the project. Open the module from the Project Navigator, then click **File -> Create/Update -> Create Symbol Files for Current File** (See Figure 1). This creates a symbol for the Register Unit which you will be able to place within your schematic. Repeat this sequence for all the other SystemVerilog modules in the project.

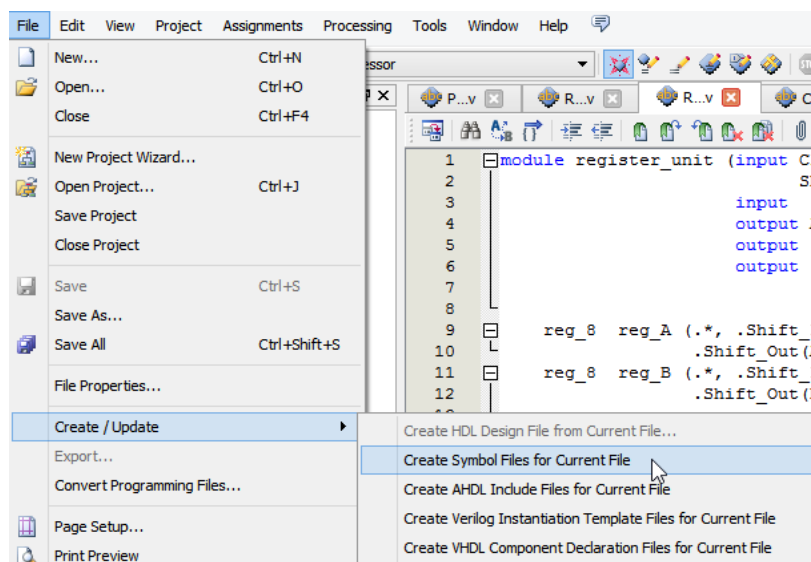



Figure 1

Add Symbols to Schematic:

If your blank schematic is not currently displayed, bring it to the top by clicking on its tab. Click the **Symbol Tool** () button. In the **Libraries** selection box, expand the **Project** folder. You should see a list of all the entities for which you have created a symbol. Click on the Register Unit, uncheck **Repeat-insert mode**, and click **OK** (see Figure 2). Move the mouse out over the schematic. A preview of the symbol will follow the cursor. Click to lay down an instance of the Register Unit. Repeat this for each unit in the top-level entity. For the HexDrivers, leave **Repeat-insert mode** checked to allow you to place multiple instances of the unit at a time; hit escape to exit repeat-insert mode when you're done. Save your design now, naming it **Processor**.

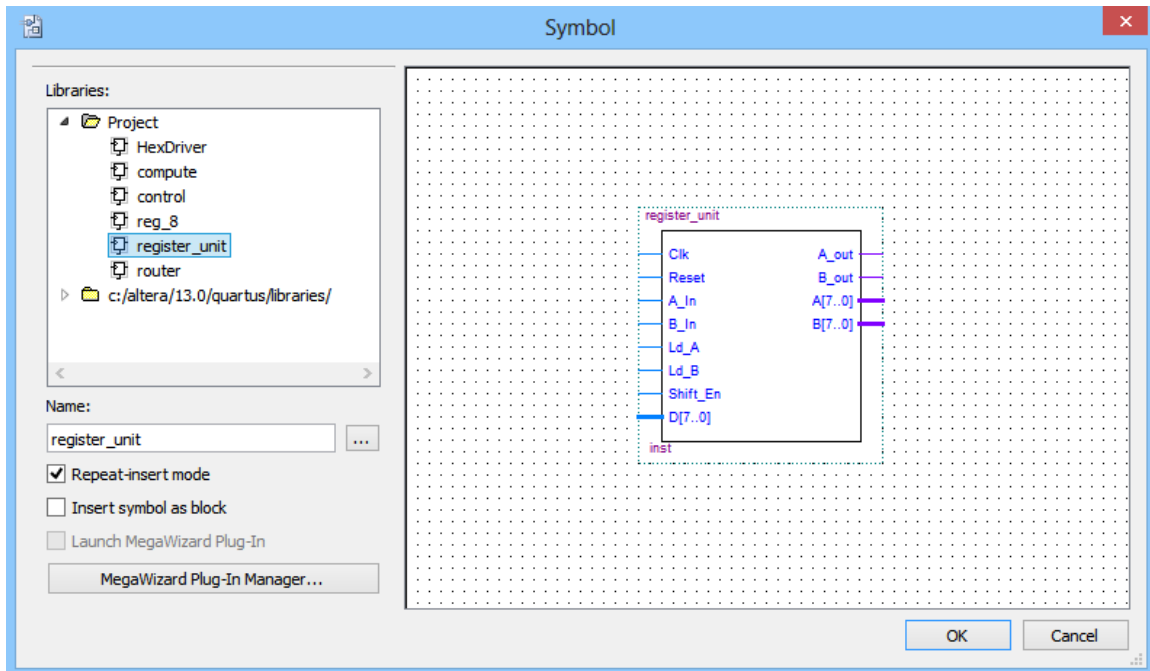




Figure 2

Navigating & Altering Your Design:

Click the **Zoom Tool** () button. Left-clicking with the Zoom Tool will zoom in on the spot you clicked; right clicking will zoom out. You can also draw a box with the tool to zoom the window on the box. Draw a box around one of your instanced entities to take a closer look at the symbol. By default, inputs are arranged on the left, with outputs on the right. Note the visual distinction between single-bit signals and multi-bit buses. Zoom back out. If you are unhappy with the placement of any of your symbols, you can click and drag it to another location with the **Selection Tool** (). The symbol will automatically snap to the grid defined in the schematic. You should lay out the register unit, compute unit, and router units so that they form a left-to-right data path, as in the block diagram in the lab manual.

Editing Symbols:


The visual layout of the default symbols is tolerable, but in complex designs, it is often inconvenient to have all inputs on the left side of a block and all outputs on the right. With the selection tool, right-click on the register unit symbol and select **Edit Selected Symbol**. A view of the register unit's symbol will open. With the selection tool, you can drag and drop pins around, resize the drawn box that forms the body of the symbol, and resize the overall dimensions of the symbol by dragging the


edges of the outer hashed box. Rearrange the symbol to better resemble the configuration in the lab manual's block diagram: put the A and B output buses on the top, the Ld_A, Ld_B, and Shift_En pins on the bottom, and move the Clk and Reset pins on the lower left instead of the upper left. Save the view and close the window to return to the top-level schematic view. Notice that the symbol hasn't changed; we need to force it to update. Right-click on the symbol and select *Update Symbol or Block*, and click *OK* in the box that comes up. The old symbol will be replaced by your edited version. Repeat this for the other symbols in the diagram: move the control signals for the function and routing units to the bottom, the control signal outputs of the control unit to the top, and shorten the HexDriver symbol vertically to save some space.

Other notes on symbol editing:

- To shorten a symbol vertically, you may have to move the “inst” text box up. Be aware that everything must fit inside of the dashed symbol boundary.
- Pins will always snap to the outer perimeter of the symbol. To move a pin to the top or bottom of the symbol, it can help to move the mouse to just below the outer boundary.
- Make sure to resize your symbols so that all the pin labels are readable.
- When laying out pins for closely-linked symbols, you can save yourself some time by putting the adjoining pins in the same configuration in both symbols. The utility of this will be demonstrated in the next section.

Wire it Up:

If you've set up the pins on adjoining units as stated above, you can quickly connect the pins by dragging one unit next to the other so that the pins overlap. Provided that the ***Use Rubberbanding*** option () is selected, you can pull the units apart again and the pins will remain connected; the program will automatically draw wires to keep them so. You should be able to do this with the register unit to the function unit, the function unit to the routing unit, and the control unit to the register unit.

Other connections aren't so easily made, however. Click the ***Orthogonal Node Tool*** () button. You can draw wires by clicking and dragging the cursor from one point to another. Connect the A_out and B_out ports of the router unit to the A_in and B_in ports of the register unit; you will have to do this in several stages to properly route the wires around the placed units. If you make a mistake, you can use the selection tool to correct it, by clicking on a wire segment to select it, and dragging it to the correct location. Save your design now; it should look something like Figure 3.

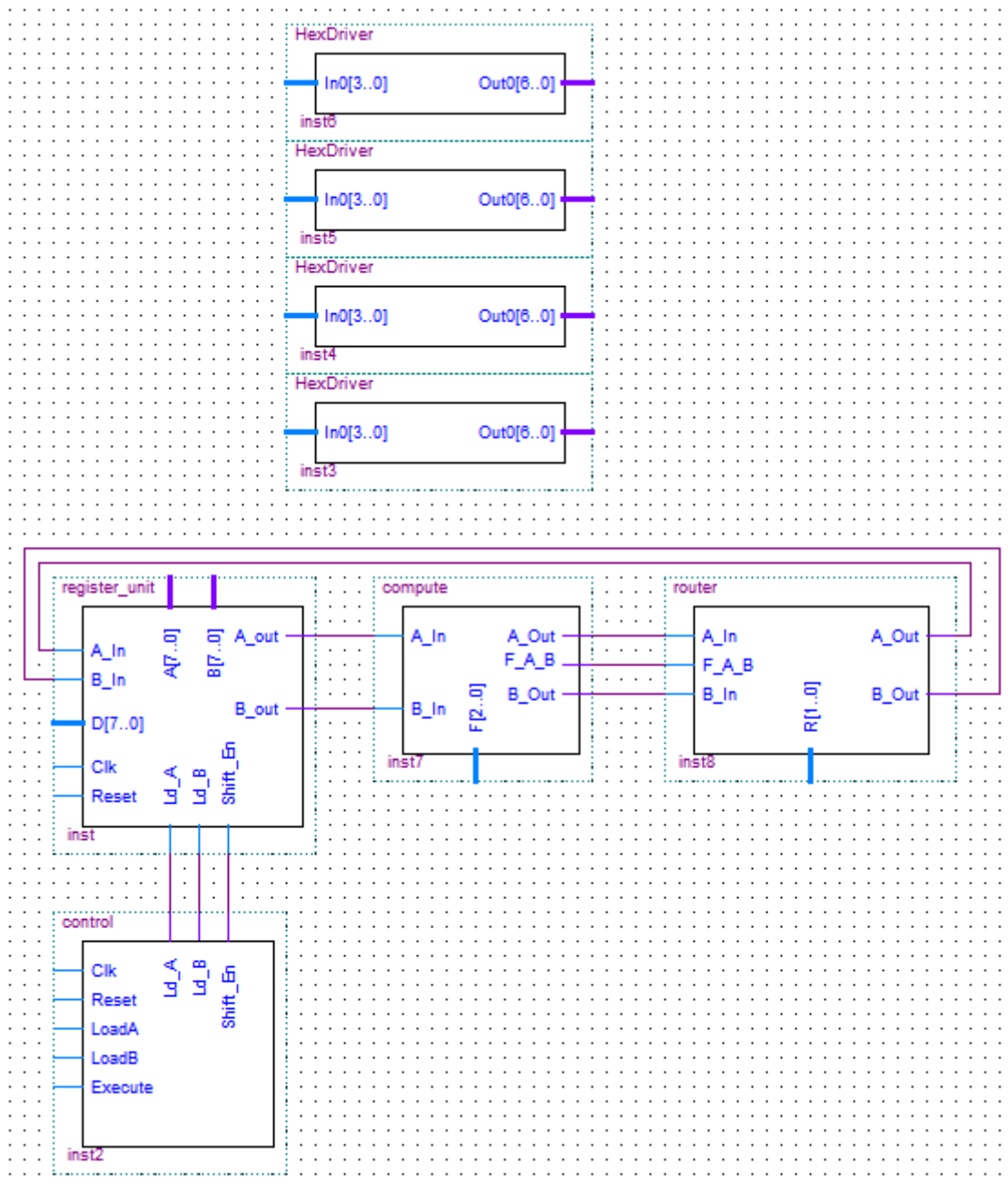



Figure 3

Ripping Buses:

You may have noticed that the outputs of the register unit that are to be displayed on the seven segment displays are eight bits wide, but the inputs to the HexDrivers are only four bits wide. We need to “rip” four-bit slices from the eight-bit signals so that they can be connected to the HexDrivers. First, use the **Orthogonal Bus Tool** () to draw a bus from the A[7..0] output of the register unit to the inputs of both of the upper HexDrivers. Similarly, connect the B[7..0] output to the other two HexDrivers. Notice that these wires are thicker than the ones you drew before; this indicates that these are multi-bit buses.

In order to rip the slices we want, we need to give these buses recognizable names. Right-click on the wire segment attached to the A output and select **Properties**. Enter “A[7..0]” in the **Name** field, and click **OK**. A text box will appear near the wire showing its name. Repeat this for bus B, using the name “B[7..0]”. (The syntax [7..0] is Altera-specific, and means the same thing as <7 downto 0> in SystemVerilog.)

To choose the slice that will be connected to the top HexDriver, right-click on the segment leading into it and select **Properties**. Here, enter the name “A[7..4]”. Repeat this for each of the other three HexDrivers, with the names “A[3..0]”, “B[7..4]”, and “B[3..0]”, respectively. Names for all these signals will appear nearby, letting us visually confirm which bits are going where. You can use the selection tool to click and drag these labels to another location if other things get in the way of reading them. Overall, your HexDriver connections should look like Figure 4. Save your design again.

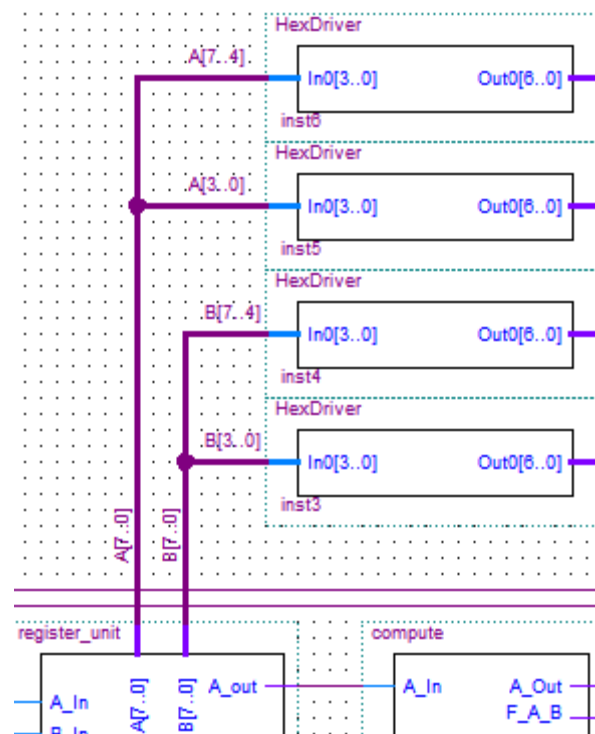


Figure 4

Add Input Pins:

We need to create 8 input pins and 7 output pins for our circuit. Click the symbol tool, and this time, expand the other folder in the **Libraries** window. (I will call this the **Libraries** folder, since it points to the location of the design libraries within the Quartus II install directory). Under that, expand **Primitives**, and then **Pin**. Select **input**, and click **OK**. (See **Figure 5**.) Place an input pin in front of the D input of the register unit, one in front of each input to the control unit, and two pointing between the register and control units. Name these pins, from top to bottom, “D[7..0]”, “F[2..0]”, “R[1..0]”, “Clk”, “Reset”, “Load_A”, “Load_B”, and “Execute” (or, for the last five, use whatever order appears on the control unit). You can name pins individually by double-clicking on them with the select tool to access the pin properties dialogue, or en-mass by first selecting the top pin with a single-click, and then double-clicking on the name text to edit it directly; when you press enter, it

will cycle forward to the next pin. Once this is done, connect the Clk, D, F, and R inputs to the correct pins on the register, function, routing, and control units. Hold off on the Reset, Load_A, Load_B, and Execute inputs for the time being.

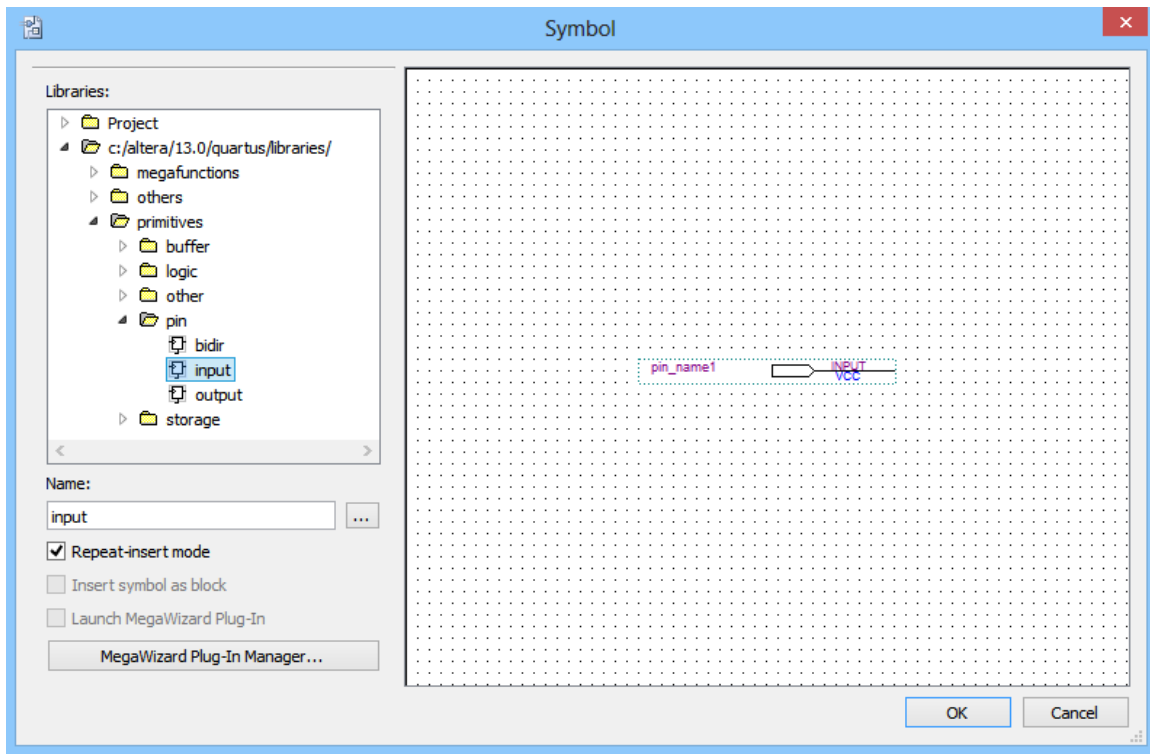


Figure 5

Add Inverters:

There's a subtlety regarding the pushbuttons on the DE2 board that is mentioned in the given code in the bit-serial processor for the top-level entity: the pushbuttons are active-low! All the given code assumes active-high logic, for consistency and simplicity. We therefore need to add inverters to the Reset, Load_A, Load_B, and Execute inputs. Click the symbol tool, and expand the **Libraries** folder, then **Primitives**, and finally **Logic**. Select **not** near the bottom of this list. An image of an inverter will appear. Click **OK**, instance four of these near the four pushbutton input pins, connect these pins to the inverter inputs, and connect the inverter outputs to the appropriate ports on the register and control units.

Add Output Pins:

We're almost finished! For the outputs, we use a similar procedure. Use the symbols tool to place seven output pins – one next to each HexDriver, two beneath these (between the HexDrivers and the datapath), and one beneath the control unit. Name these “AhexU[6..0]”, “AhexL[6..0]”, “BhexU[6..0]”, “BhexL[6..0]”, “Aval[7..0]”, “Bval[7..0]”, and “LED[3..0]”, from top to bottom. The hex outputs can be connected directly to the outputs of the HexDrivers; the Aval and Bval outputs should be connected to the A and B signals. For the LED output, we need to merge the Execute, Load_A, Load_B, and Reset signals. Draw a bus out from the LED output to the left, under the control unit and just under the input pins, then draw a wire from each of the four signals we need to this bus. Label the bus “LED[3..0]” and the wires individually, as before. The one connected to Execute should be “LED[3]”, Load_A “LED[2]”, Load_B “LED[1]”, and Reset “LED[0]”. At this point, your design should look something like **Figure 6**. Save your design now.

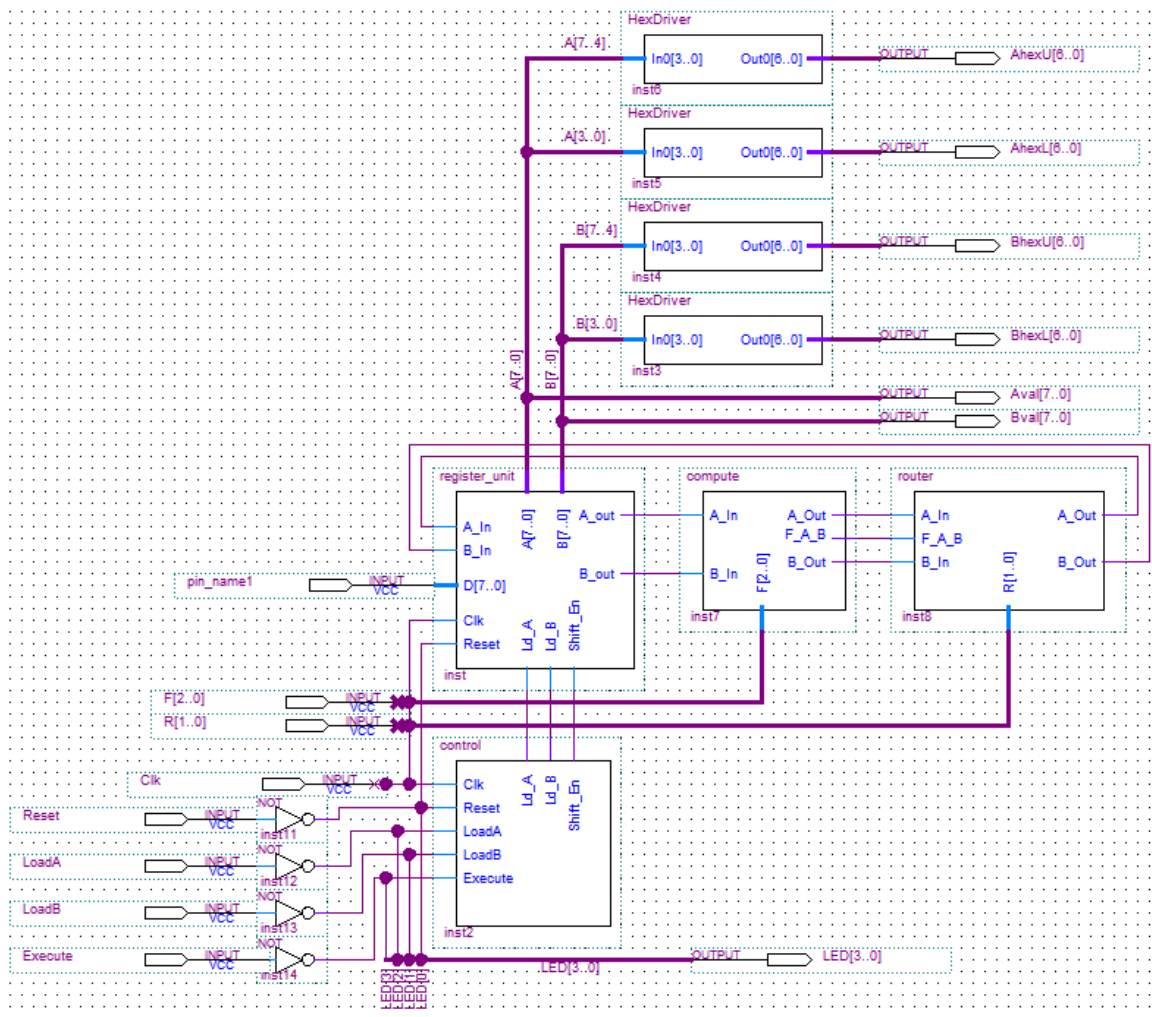



Figure 6

Checking Design Rules and Synthesizing:

Click the **Start Analysis & Synthesis** button () . If you've followed these instructions carefully, analysis & synthesis should complete to 100% with 0 warnings and errors. If you do get warnings or errors, view them in the appropriate tab and correct them. Quartus will build the hierarchy in the **Project Navigator** window.

Schematic to HDL Design File Conversion and vice versa:

Sometimes it is necessary to convert the schematic files into HDL design files for portability, since the schematic file format in Quartus is not supported in other compilers or simulators, not even for ModelSim-Altera. For other times, it is desirable to view the RTL synthesis schematics of the HDL design files to get a sense of what the codes are synthesized into. Quartus has the capability to perform the conversion automatically. The procedures are described below:

- **Schematic to HDL**

Select the desired schematic file, and in the **File** menu, select **Create/Update -> Create HDL design for current file**. In the window that opens, make sure **Verilog HDL** is selected, and click **OK**. This will convert your schematic design into a Verilog file (currently conversion to SystemVerilog is not yet supported!) for you to hand in. Compare this code to the code for the Processor unit in the lab manual. You should find that they are functionally equivalent.

- **HDL to Schematic**

Select the desired SystemVerilog file, and in the **Tools** menu, select **Netlist Viewers -> RTL Viewer**. This will display the design file in synthesized schematic view. If the conversion HDL file is a top-level entity with instantiated modules, then the modules will be shown in RTL blocks. Double-clicking on a module block will bring you to the lower-level schematic of the module. You will be able repeat this until you have reached the lowest gate-level schematics. See [1] for detailed explanation.

The remaining steps are the same whether you've used these instructions or entered the port map directly. You should pick up from "Analyzing, Synthesizing, and Implementing the Design" in the Instructions for Performing Experiments Involving SystemVerilog in the lab manual, page IQT.12.

Final Notes:

If you've done your design well, it's probably high-enough quality to hand in as your block diagram in your lab report. Make sure any signals that need labels are labeled (by giving them names), and print it out. This not only saves you the trouble of entering the port map in text, but you don't have to draw a block diagram by hand either. You can add a title block if you like by using the symbol tool and looking in **Libraries -> Primitives -> Other -> title**. One point to note is that after you add pin assignments, these will appear in small tables next to each pin; you should move these tables so that they don't obscure any part of your circuit. Also note that block diagrams may be needed for subcomponents of your circuit. (For example, you might find it easier to group several block diagrams in a larger upper-level block diagram for readability and tidiness. In this case you will have to also provide the sub-block diagram to show a complete view of your circuit.) When you hand in your code, be sure to include the Verilog file you generated from your schematic.

References:

- [1] “Analyzing Designs with quartus II Netlist viewers” in Quartus II handbook v13.0.0, Altera, 2012. Available at: http://www.altera.com/literature/hb/qts/qts_qii51013.pdf