COMP 231
Introduction to Computer Organization
Using Blocks in Quartus

Rhodes College
—1848—

This document will describe an effective strategy for dealing with complicated logic circuits in Quartus. Similar to using functions in a computer program to abstract ideas and eliminate redundancy, we can use blocks in Quartus to the same end. First we define a small circuit as a block, then we can use it as a building block of a larger circuit. You may find this technique useful in the adder lab, as well as later projects.

An XOR Block

A Quartus project has a *top-level entity*, which is like your `main()` routine. This is the file you create when you first start your project. In this example we are going to create an XOR circuit in a block, then use that block in the top-level function, similar to the first lab.

Let's assume that we have created a new Quartus project with a top-level entity, `blocks`. For now, I've just added two input pins and a single output pin:

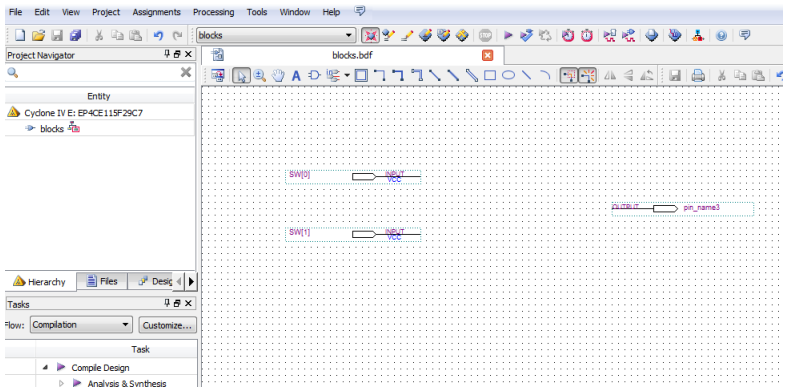

Figure 1: Initial top level circuit: *blocks*

Now, let's create a block for our XOR circuit. In the File menu, select *New...* and select a new Block Diagram / Schematic File:

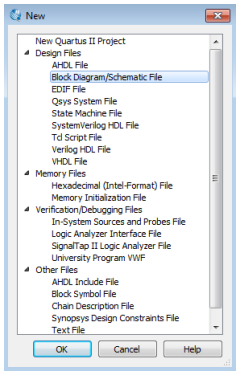

Figure 2: Creating a new block file

We should end up with a new tab on the top and a blank schematic canvas. This is the place we will implement the XOR circuit as a block (like a function) which will be used by the top-level. Start by creating two input pins and one output pin. These will define the inputs/outputs of the block and are like parameters and a return value from a function:

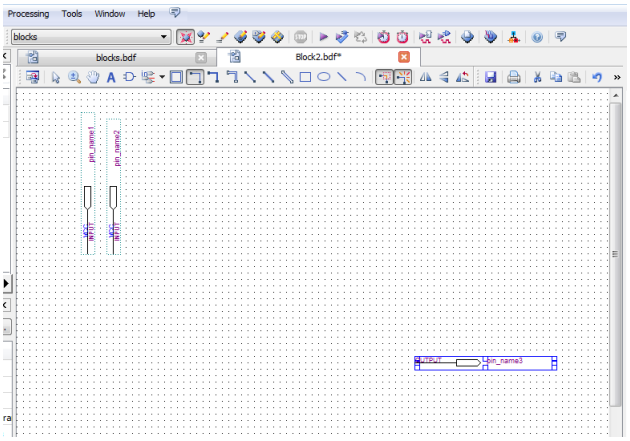

Figure 3: Creating the input and outputs of the block

Next, we know that XOR requires two inputs, an x and a y. Internal to the circuit, we know we will need terms for $\bar{x}y$ and $x\bar{y}$. If we are careful about how we run the wires in this circuit, it will be easy to organize our circuit. This structure can be replicated whenever you are implementing a complicated circuit in MSOP form.

First, we run both inputs using vertical wires on the left-hand side of the circuit. Next, we run taps into each input and connect each to its own inverter. Lastly, we run the inverted signals down with vertical wires parallel to the un-inverted inputs.

Now we have a set of wires for each of the input parameters and their complements, which we can tap into to create our circuit. Since XOR has two and gates and an or gate, we can create these gates and connect the appropriate inputs. The final circuit looks like this (note labels on input wires):

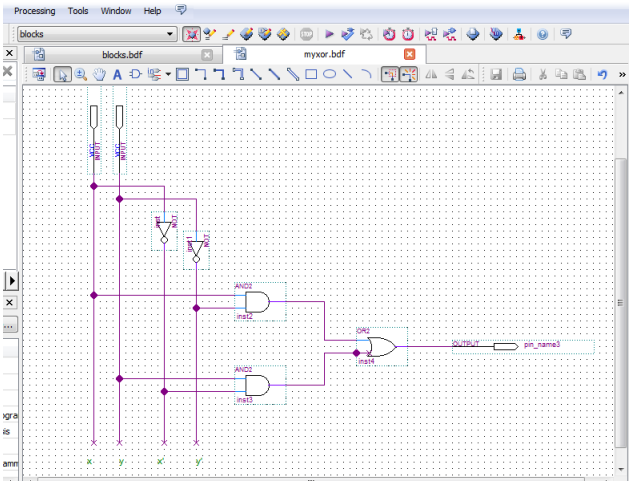

Figure 4: The final XOR circuit block

This structure makes it easy to see the layout of the circuit and allows for simple changes without much effort.

To use this block, we need to save the BDF file with a meaningful name, I chose `myxor.bdf`. Next, under the Project

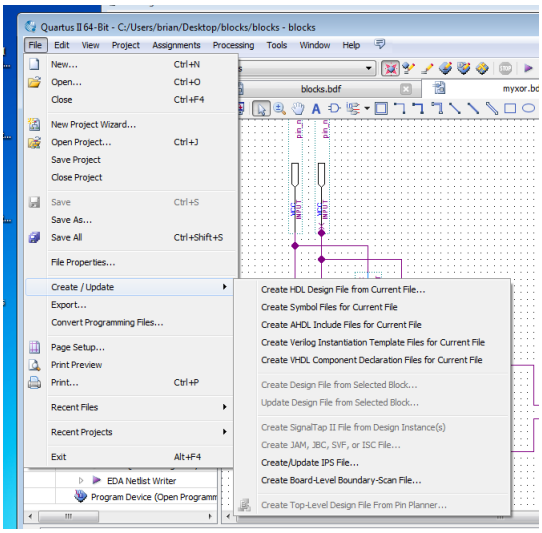

Figure 5: Creating symbols for the block

menu, select *Add Current File to Project*. You can select the *Add/Delete Files from Project...* dialog to check if it was added properly.

Next, we need to export the symbols from the BDF file and make them visible to the top-level circuit. Again, under the File menu, select *Create / Update* → *Create Symbol Files for Current File*, while still in the `myxor` tab:

Now we are ready to import our XOR circuit block into our top-level. Switch over to the top-level schematic, `blocks`. Select the tool to add a logic gate as you would add a new AND, OR, or NOT gate. You should now see a *Project* folder, with our `myxor` block inside:

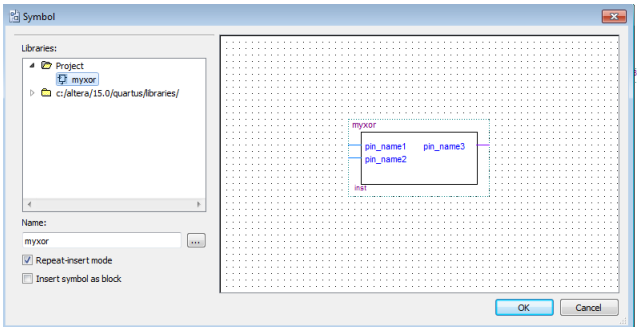

Figure 6: Importing our XOR block

Now we should see our block imported into the top-level. We can connect wires to the inputs and outputs like any other gate. Here is the final circuit:

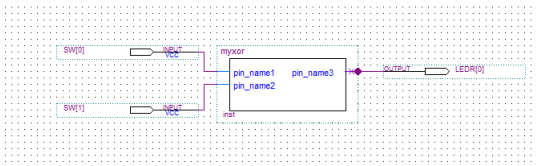

Figure 7: Imported block

If we change the `myxor` block later, we would need to update the symbols in the file. To do this, select the block and right click. Next select the *Update Symbol or Block...* entry:

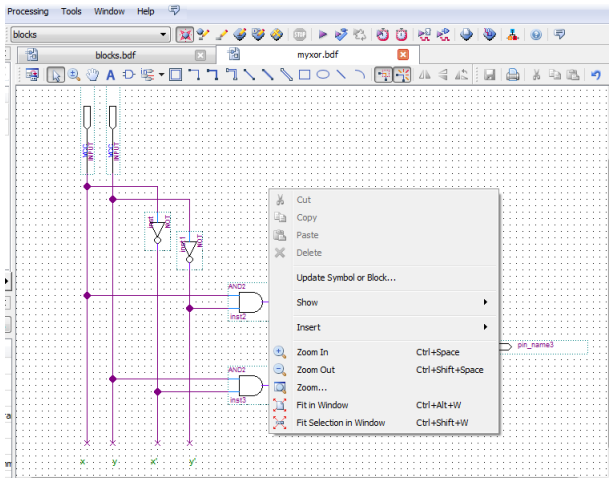

Figure 8: Updating the block symbols