

Tutorial: ISE 11.1 and the Spartan3e Board

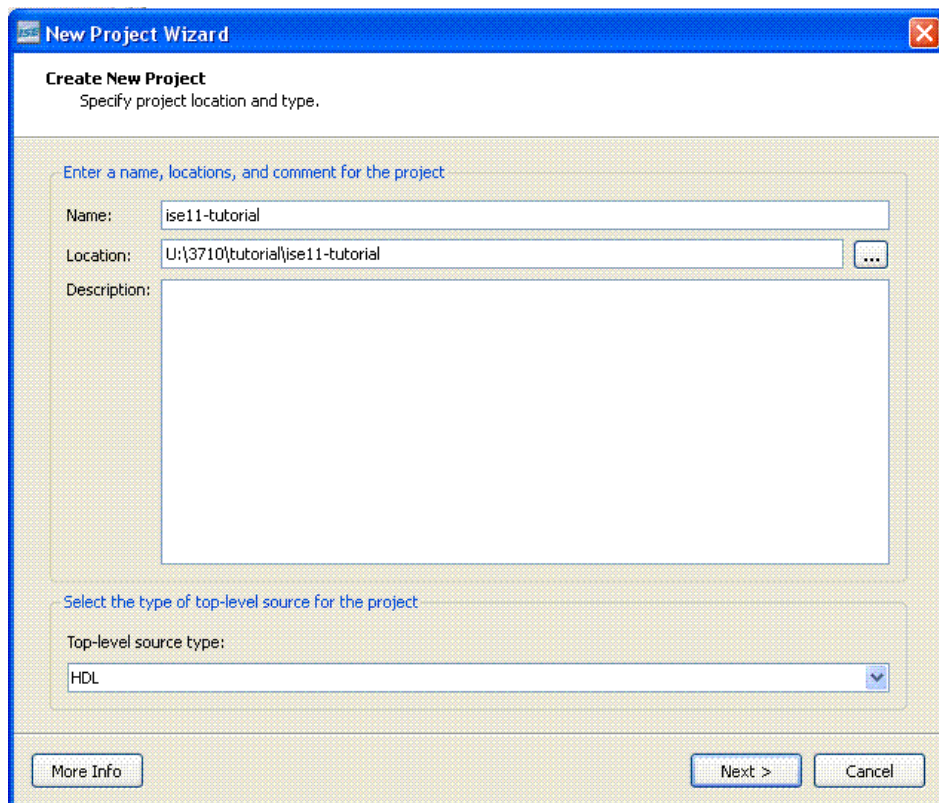
This tutorial will show you how to:

- Use a combination of schematics and Verilog to specify a design
- Simulate that design
- Define pin constraints for the FPGA (.ucf file)
- Synthesize the design for the FPGA board
- Generate a bit file
- Load that bit file onto the Spartan3e board in your lab kit

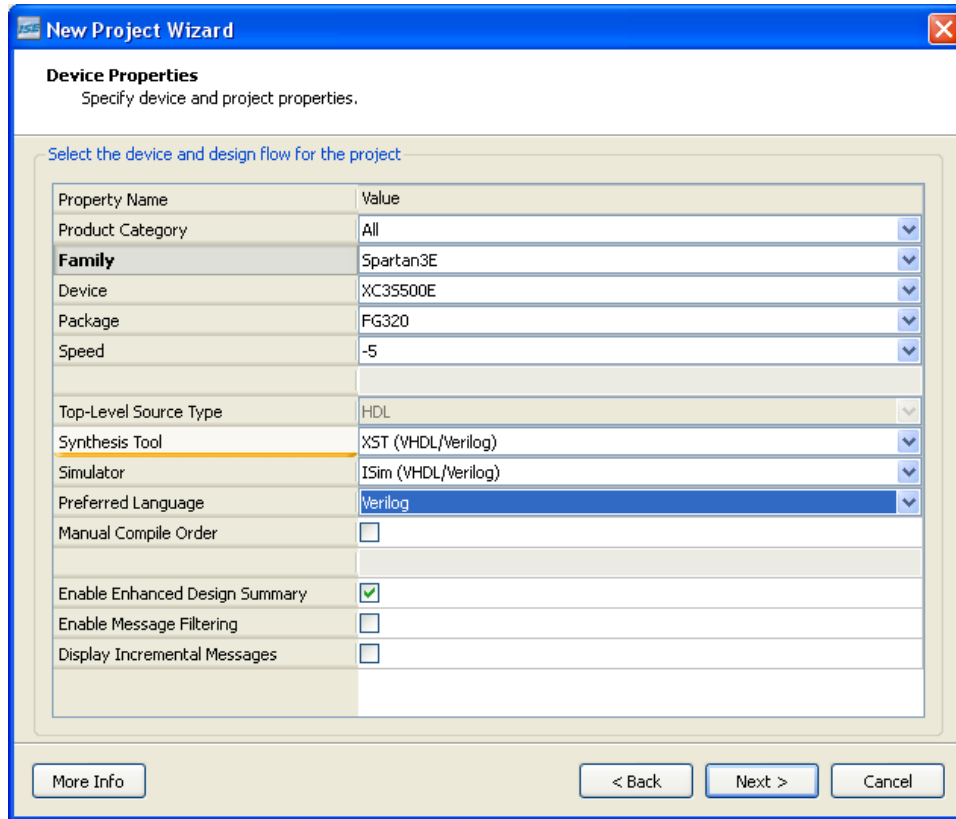
This assumes that you're using a DSL lab machine, or that you've installed Xilinx ISE 11.1 on your own machine. This tutorial is specifically for the Spartan3e board.

1 New Project Setup and Verilog Circuit Specification

1. Start the Xilinx ISE 11.1 tool. It should be on the desk top.
2. Create a new project. The Create New Project wizard will prompt you for a location for your project. Note that by default this will be in the documents and Settings folder the first time you start up. You'll probably want to change this to something in your own folder tree (usually the U:\ directory).

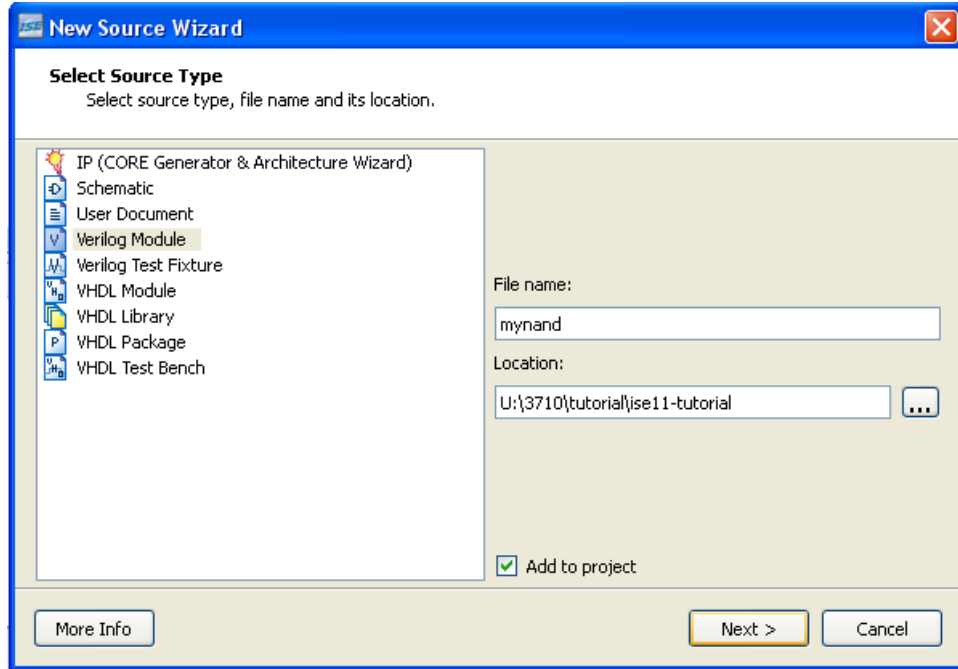


3. On the second page of the Create New Project dialog, make *sure* that you use the **Spartan3E** Device Family, **XC3S500E** Device, **FG320** Package, and **-5** Speed Grade. You can also specify **HDL** as the Top-Level Source Type with **XST** as the Synthesis Tool, **ISim** as the Simulator, and **Verilog** as the language. These aren't critical, but they do save time later.

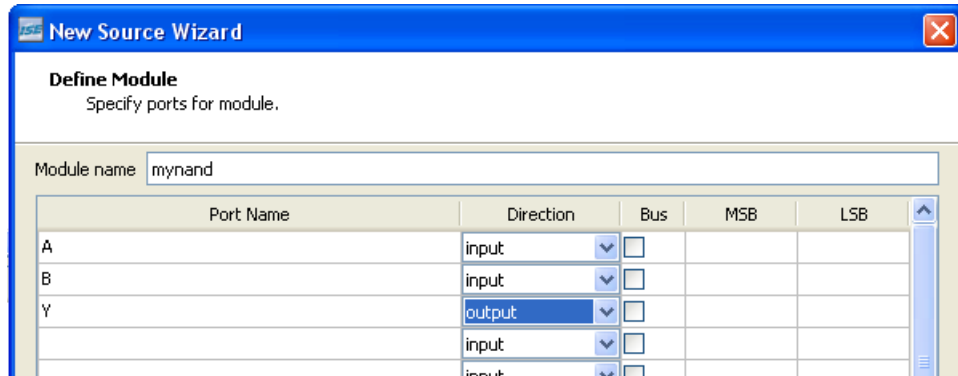


4. You can skip the other wizard dialogs, or you can use them to create new Verilog file templates for your project. I usually just skip them and create my own files later.

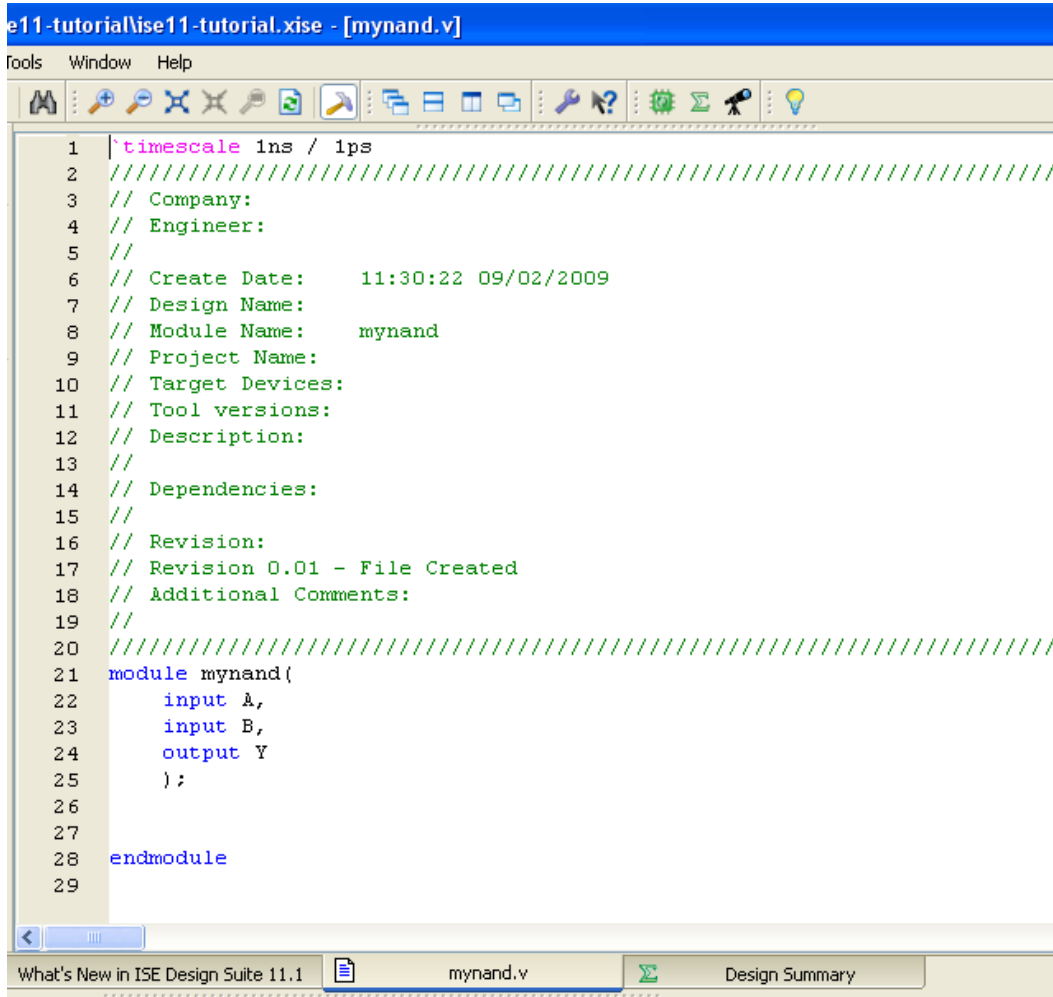
- Now you want to open a new source file. Use the **Project | NewSource** menu choice. This first one will be a Verilog file so make sure you've selected **Verilog Module** as the type and give it a name. This tutorial calls the example **myrand**.



- When you press **Next** you'll get a dialog box that lets you define the inputs and outputs of your new module. Add two inputs (A and B), and one output named Y. Remember that Verilog is case sensitive!



7. When you **Finish**, you'll have a template for a Verilog module that you can fill in with your Verilog code. Click on the `myrand.v` tab. It looks like this (note that you can also fill in the spots in the comment header with more information):



The screenshot shows the Xilinx ISE Design Suite 11.1 interface. The title bar reads "e11-tutorialise11-tutorial.xise - [myrand.v]". The menu bar includes "Tools", "Window", and "Help". The toolbar contains various icons for file operations, editing, and simulation. The main editor window displays the following Verilog code:

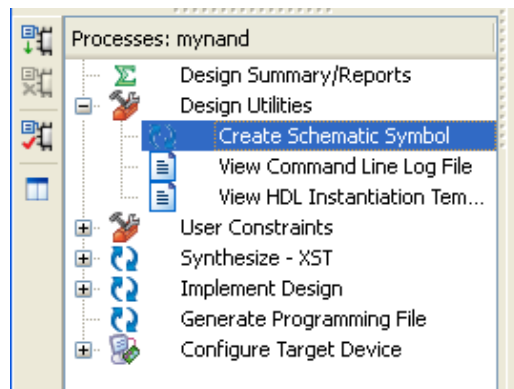
```
1 |`timescale 1ns / 1ps
2 |////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
3 | // Company:
4 | // Engineer:
5 | //
6 | // Create Date:      11:30:22 09/02/2009
7 | // Design Name:
8 | // Module Name:     myrand
9 | // Project Name:
10 | // Target Devices:
11 | // Tool versions:
12 | // Description:
13 | //
14 | // Dependencies:
15 | //
16 | // Revision:
17 | // Revision 0.01 - File Created
18 | // Additional Comments:
19 | //
20 |////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
21 | module myrand(
22 |     input A,
23 |     input B,
24 |     output Y
25 | );
26 |
27 |
28 | endmodule
29 |
```

The status bar at the bottom shows "What's New in ISE Design Suite 11.1", "myrand.v", and "Design Summary".

8. Now you can fill in the rest of the Verilog module to implement some Boolean function. Implement a NAND for this example. You can use any of the Verilog techniques that you know about. (see the Brown & Vranesic text from 3700, for example, or any number of Verilog tutorials on the web.) Note that ISE 10.1 uses Verilog 2001 syntax where the inputs and outputs are defined right in the argument definition line. This example uses a continuous assignment statement: `assign Y = ~(A & B);` as shown below. Save the file.

```
20 ////////////////////////////////////////////////////
21 module mynand(
22     input A,
23     input B,
24     output Y
25 );
26
27     assign Y = ~(A & B);
28
29 endmodule
30
```

9. In order to use this Verilog code in a schematic, you'll need to create a schematic symbol. Select the `mynand.v` file in the Sources For: | Hierarchy window. Then in the Processes: mynand window expand the Design Utilities option and double click Create Schematic Symbol.

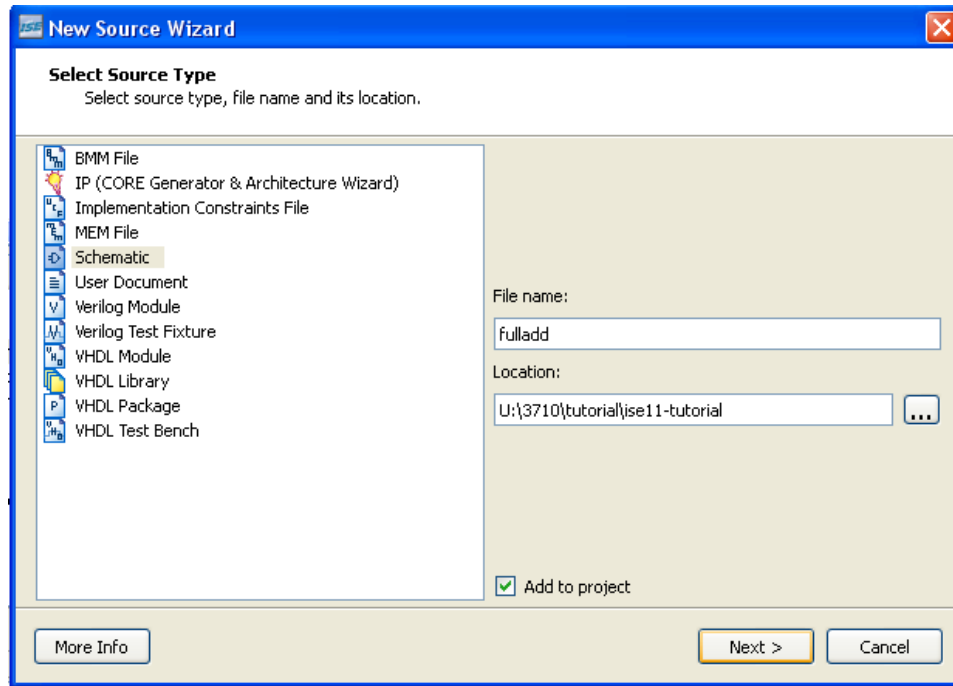


You now have a piece of Verilog that you can simulate and synthesize as is, or you can also use it in a schematic as a component.

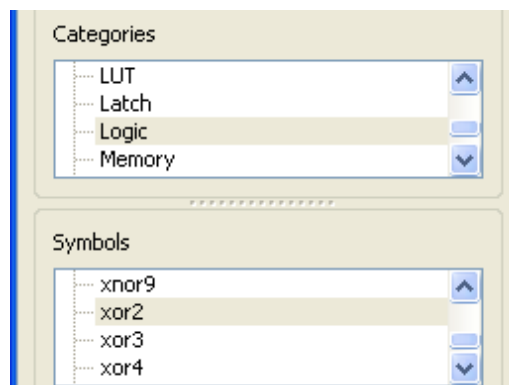
2 Creating a Schematic

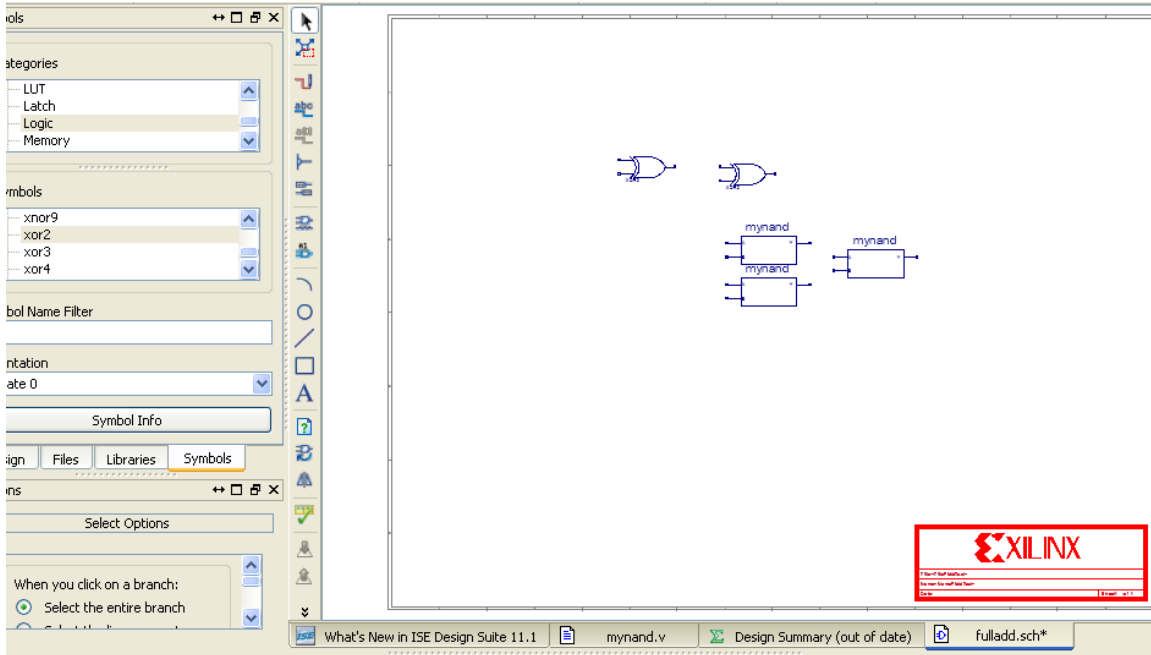
Your project can be totally Verilog, or totally schematics, or a mixture of the two. This example will feature a mix, just to show you how it can be done.

1. Start by going to Project | NewSource and this time choosing schematic as the type. I'm calling this **fulladd**. Can you guess where this is going?

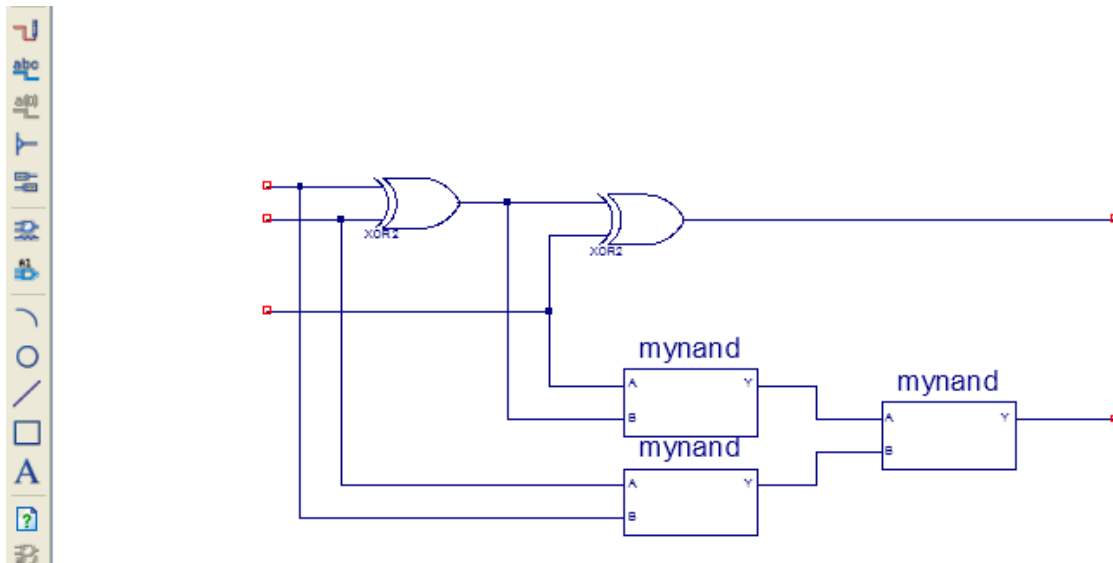


2. In the schematic window you'll see a frame in which you can put your schematic components. You can select components by selecting the Symbols tab in the Sources pane. You should always add titles to your schematics. You do this under the General category selecting the Title symbol. This is normally placed in the bottom right hand corner. You can fill in the fields of the title by double clicking on it. Add three copies of **myand** from the < -- All Symbols -- > category, and two copies of the **xor2** component from the Logic Category.

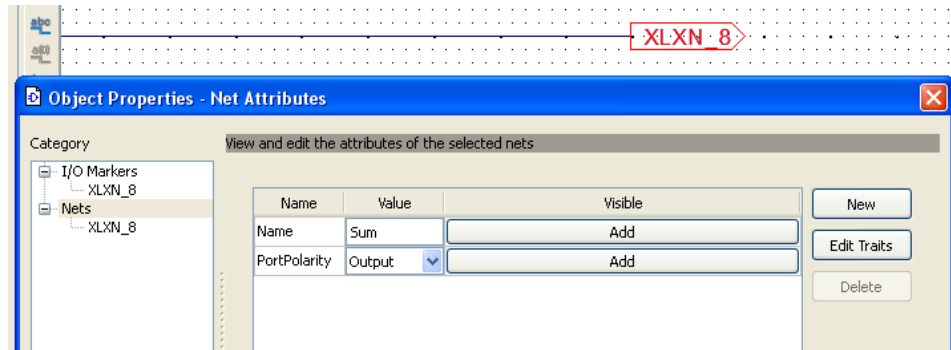




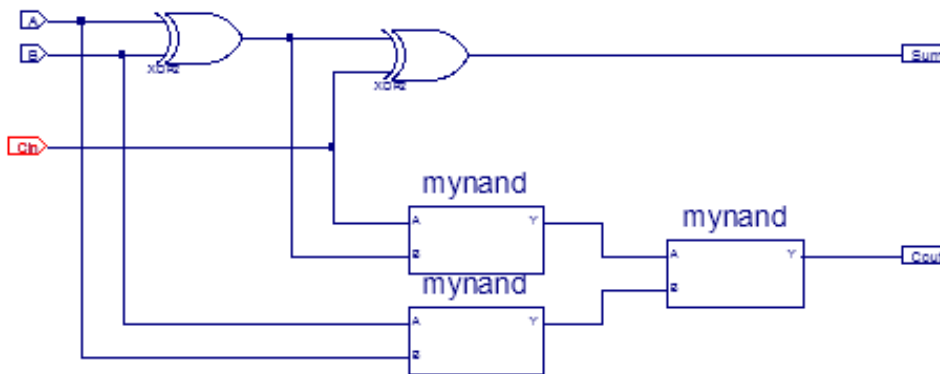
- Now use the wiring tool (red line with pencil) to connect up the components to make a Full Adder. Right click on a wire or component to delete it.



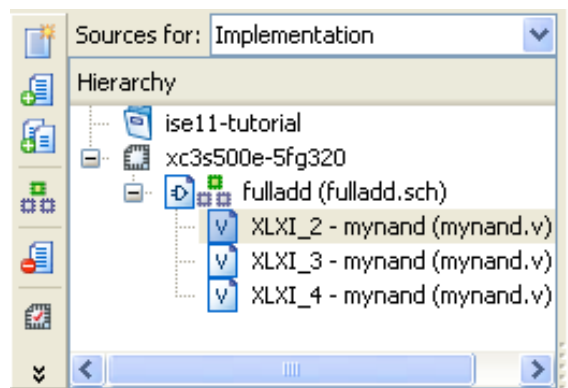
- Use the Add I/O Marker symbol to add pins to export from this circuit. Click on the red boxes to add them to the nets. Double click on the I/O markers to change their names, and click on Nets.



The completed schematic with the I/O pins defined looks like this:



- Save the schematic. You are now ready to simulate the circuit which consists of part schematics (using xor2 from the Xilinx library), and part Verilog (your mynand.v code). If you go back to the Design tab and look into the Hierarchy pane you will see that the fulladd schematic includes three copies of mynand.v.

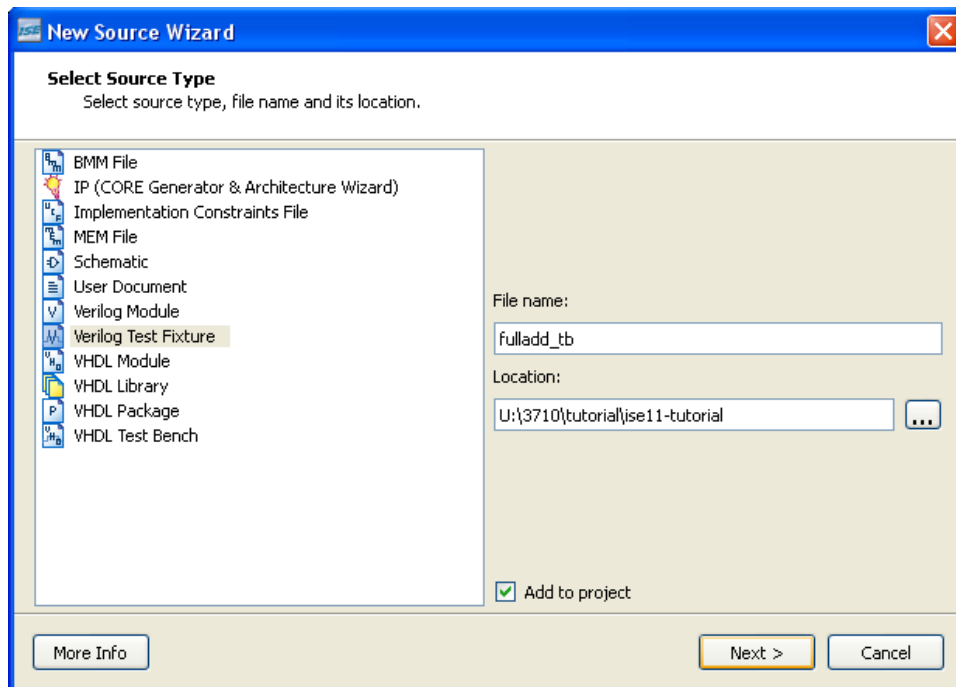


3 Simulating your Circuit

Now that you have a saved source file (fulladd is the top file in this case), you can simulate its behavior. We'll use the ISE simulator with a testbench to drive the simulation. Note that the testbench files that drive the simulations are also Verilog files.

To simulate the fulladd circuit:

1. Go to the top left pane and in the **Design** tab change the **Sources For:** field to be **Behavioral Simulation**. This changes the view to include sources that are interesting for simulation, and also changes the options in the bottom pane to show the simulation options.
2. You can go to the **Project | NewSource** menu again or you can select **New Source** icon (the page with an orange star). This will bring up the **New Source Wizard**. In that dialog select **Verilog Test Fixture** and type in the name of your testbench file. I will name my testbench `fulladd_tb` (where the `tb` stands for testbench). The box looks like:



3. The Next dialog asks you which source you want the testbench constructed from. Choose **fulladd**, of course. The code that gets generated includes an instance of the fulladd schematic named **UUT** (for Unit Under Test).

```
1 // Verilog test fixture created from sc
2
3 `timescale 1ns / 1ps
4
5 module fulladd_fulladd_sch_tb();
6
7 // Inputs
8     reg Cin;
9     reg A;
10    reg B;
11
12 // Output
13    wire Sum;
14    wire Cout;
15
16 // Bidirs
17
18 // Instantiate the UUT
19    fulladd UUT (
20        .Cin(Cin),
21        .A(A),
22        .Sum(Sum),
23        .Cout(Cout),
24        .B(B)
25    );
26 // Initialize Inputs
27 `ifdef auto_init
28     initial begin
29         Cin = 0;
30         A = 0;
```

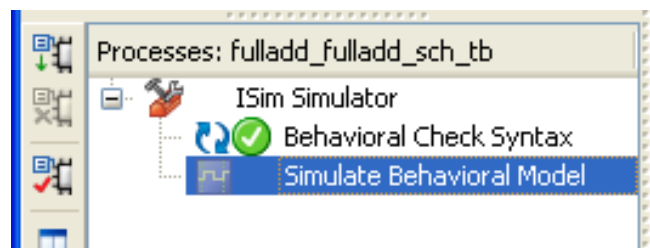
- Note that the generated template has some code with an **'ifdef** for initializing things. I don't use the **'ifdef** code. Instead I write my own **initial** block and driving code for testing the circuit. *Remember that good testbenches ALWAYS use \$display statements and "if" checks so that the testbench is self-checking!* You could enumerate all eight possibilities of the inputs and check the outputs. I'm going to get a little tricky with a concatenation and a loop.

```

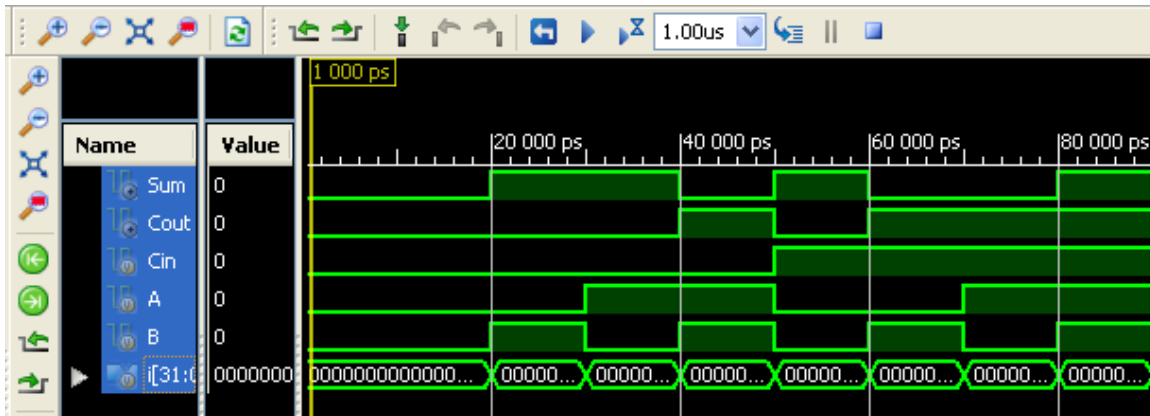
26 // Initialize Inputs
27 integer i=0;
28 initial begin
29     Cin = 0;
30     B = 0;
31     A = 0;
32     #10 $display("starting test");
33     for (i=0; i<8; i=i+1)
34     begin
35         Cin = i[2];
36         A = i[1];
37         B = i[0];
38         #10 $display("%d: Cin,A,B = %b%b%b Cout,Sum = %b%b", i, Cin, A, B, Cout, Sum);
39         if (Cout != ((A&B)|(A&Cin)|(B&Cin)))
40         begin
41             $display("Carry fault: Cout=%b for Cin,A,B = %b%b%b", Cout, Cin, A, B);
42         end
43         if (Sum != (A+B+Cin))
44         begin
45             $display("Sum Fault: Sum=%b for Cin,A,B = %b%b%b", Sum, Cin, A, B);
46         end
47     end
48 end

```

- Once you fill in the testbench with Verilog code to drive the simulation, you can simulate the design. Select the **fulladd_tb.v** in the Hierarchy pane, and check the syntax and run the simulation from the **Processes** pane.



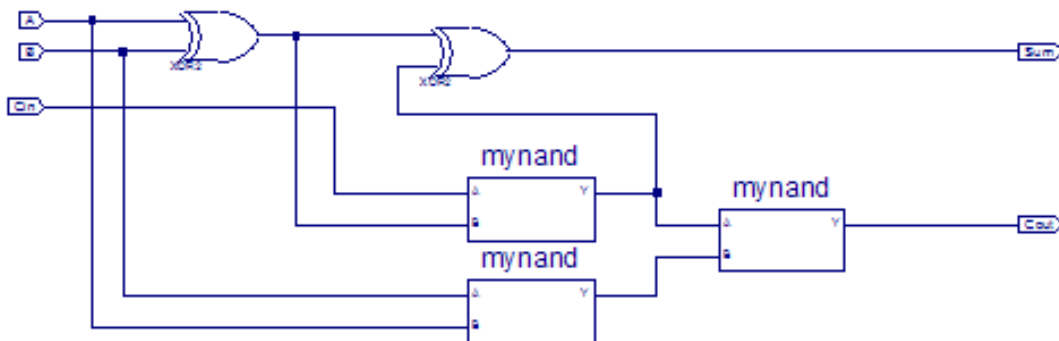
The output will be displayed as waveforms, and the \$display data will show up in the console as shown (after scrolling all the way to the left and zooming out to see all the waveforms). You can see that not only do the waveforms show the results of the simulation, but the \$display statements have printed data, and because the circuit is correctly functioning, no error statements were printed.



```

Console
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
starting test
0: Cin,A,B = 000 Cout,Sum = 00
1: Cin,A,B = 001 Cout,Sum = 01
2: Cin,A,B = 010 Cout,Sum = 01
3: Cin,A,B = 011 Cout,Sum = 10
4: Cin,A,B = 100 Cout,Sum = 01
5: Cin,A,B = 101 Cout,Sum = 10
6: Cin,A,B = 110 Cout,Sum = 10
7: Cin,A,B = 111 Cout,Sum = 11
ISim>
  
```

6. You can change the circuit to have a mistake to show off the self-checking test-bench...



```
Console
Finished circuit initialization process.
starting test
  0: Cin,A,B = 000 Cout,Sum = 01
Sum Fault: Sum=1 for Cin,A,B = 000
  1: Cin,A,B = 001 Cout,Sum = 00
Sum Fault: Sum=0 for Cin,A,B = 001
  2: Cin,A,B = 010 Cout,Sum = 00
Sum Fault: Sum=0 for Cin,A,B = 010
  3: Cin,A,B = 011 Cout,Sum = 11
Sum Fault: Sum=1 for Cin,A,B = 011
  4: Cin,A,B = 100 Cout,Sum = 01
  5: Cin,A,B = 101 Cout,Sum = 11
Sum Fault: Sum=1 for Cin,A,B = 101
Console Breakpoints Search Results
```

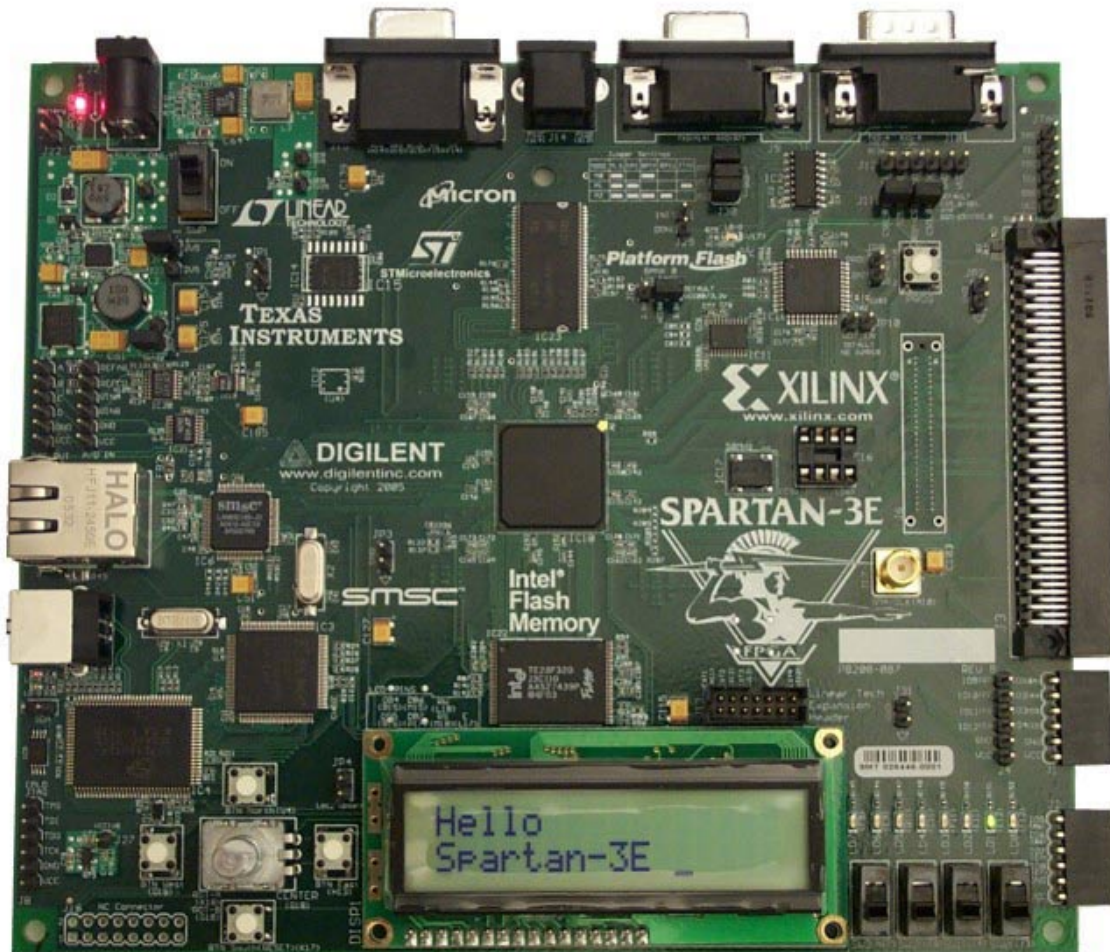
It's easy to tell that something is wrong!

4 Synthesizing your circuit to the Xilinx FPGA

Now that you have a correctly simulating Verilog module, you will have the ISE (web-PAK) tool synthesize your Verilog to something that can be mapped to the Xilinx FPGA. That is, the Verilog code will be converted by ISE to some gates that are on the FPGA. To be even more specific, ISE will convert the schematic/Verilog project description into a set of configuration bits that are used to program the Xilinx part. Those configuration bits are in a **.bit** file and are downloaded to the Xilinx part in this section of the tutorial.

You will use your Spartan-3E board for this part of the tutorial. This is known as the "Spartan 3E Starter Kit" and is a board produced by Xilinx. It is a very feature-laden board with a Spartan 3e XC3S500E FPGA, 64Mbytes of SDRAM, 128Mbits of flash EPROM, A/D and D/A converters, RS232 drivers, VGA, PS/2, USB, and Ethernet connectors, a 16 character two-line LCD, and a lot more. You can get more info from Xilinx at

<http://www.xilinx.com/products/devkits/HW-SPAR3E-SK-US-G.htm>

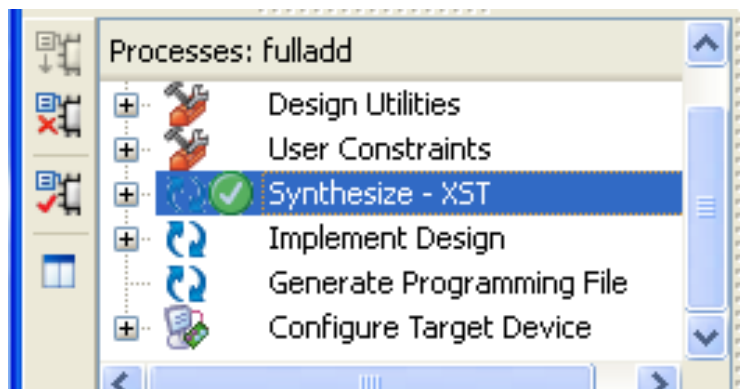


Specifically we need to:

- Synthesize the Verilog code into FPGA configuration
 - Assign A, B, and Y to the correct pins on the FPGA that connect to the switches and LEDs on the S3E board
 - Generate a programming file with all this information (.bit file)
 - Use the **impact** tools from Xilinx (part of WebPACK) to configure the FPGA through the USB connection.
1. Back in the Sources pane, return to the **Implementation** view and select your **fulladd** schematic. Now in the bottom (Processes) pane you will see some options including **Synthesize - XST**. Double click on this to synthesize your circuit. After a while you will (hopefully) get the "Process 'Synthesize' completed successfully" message in the console. If you've already simulated your circuit and found it to do what you want, there's every chance that this will synthesize correctly without problems.

In any case, there is lots of interesting information in the synthesis report (the data in the console window). It's worth looking at, although for this amazingly simple example there isn't anything that fascinating.

Make sure that you end the process with a green check for this process. If you get something else, especially a red X, you'll need to fix errors and re-synthesize.



```
Console

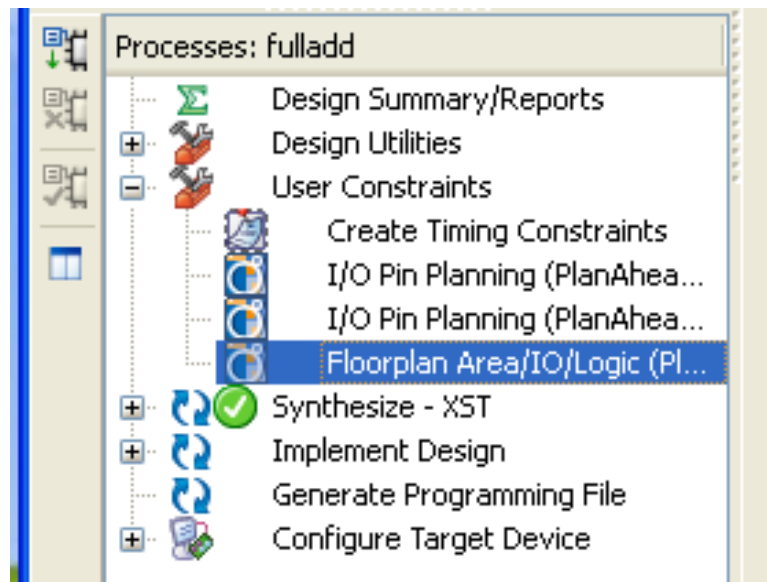
Timing Summary:
-----
Speed Grade: -5

Minimum period: No path found
Minimum input arrival time before clock: No path found
Maximum output required time after clock: No path found
Maximum combinational path delay: 6.685ns

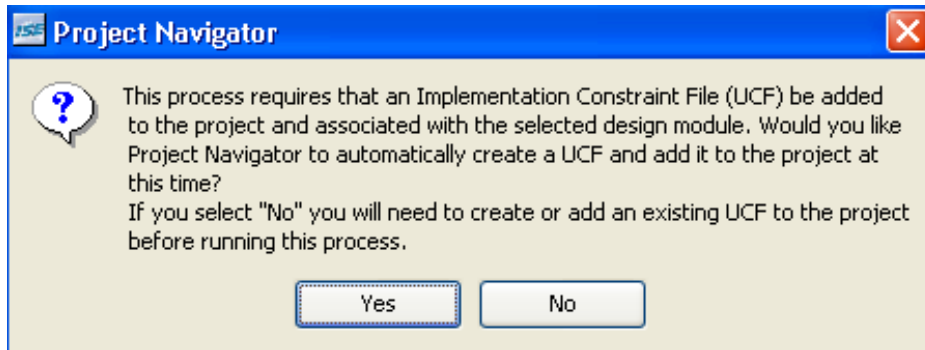
=====

Process "Synthesis" completed successfully
```

2. Now, because we're headed towards putting this on the Xilinx FPGA on the Spartan-3E board, we need to set some constraints. In particular, we need to tell ISE which pins on the Xilinx chip we want **A, B, Cin** assigned to so that we can access those from switches, and where we want **Cout** and **Sum** so we can see those on the LEDs on the Spartan-3E board. Go to the Processes pane and expand **User Constraints** so you can select **Floorplan Area/IO/Logic**.

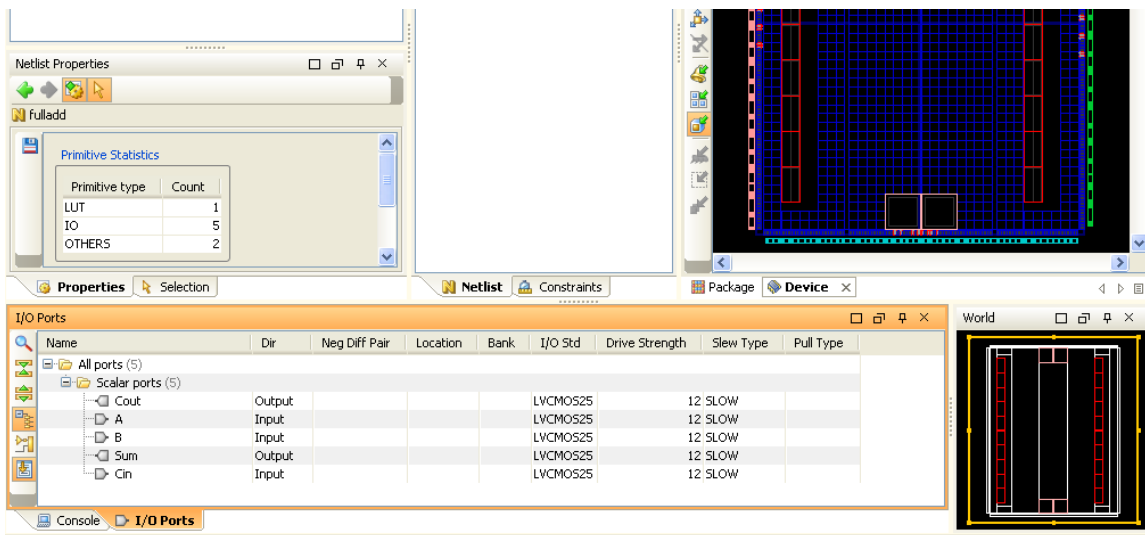


This will open a whole new tool called **PlanAhead** (replacing the old PACE tool) which you can use to set your pin constraints. You may have to agree to add a **UCF** (Universal Constraints File) file to your project. You should agree to this.



If you are asked for a software update, decline.

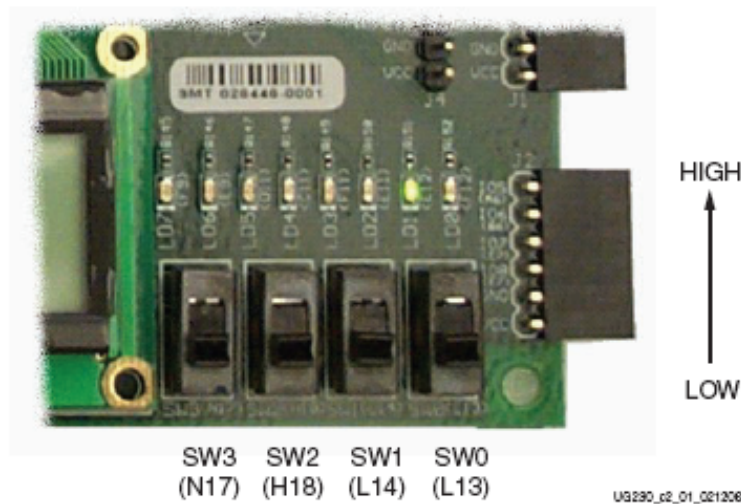
3. The PlanAhead tools lets you set a number of different types of constraints on how the circuit is mapped to the Xilinx part. For now we'll just use the pin constraints in the UCF file.



In the bottom part of the screen you can find your pins in the I/O Ports pane. You can set their location in the **Location** field. How do you know which pins to assign the signals to in order to use the switches and LEDs on the Spartan-3E board? You look in the Spartan-3E Starter Kit Users Manual which is linked to the class web site, and also available from Xilinx at

http://www.xilinx.com/support/documentation/boards_and_kits/ug230.pdf

For now I'll just tell you that the four sliding switches on the Spartan-3E board are, from left to right as you're looking at the board with the LCD at the bottom, are on pins **N17**, **H18**, **L14**, and **L13**. Here's the diagram from the User Guide:



SW3 SW2 SW1 SW0
(N17) (H18) (L14) (L13)

UG290_c2_01_021208

Figure 2-1: Four Slide Switches

and the UCF info is:

```
NET "SW<0>" LOC = "L13" | IOSTANDARD = LVTTTL | PULLUP ;
NET "SW<1>" LOC = "L14" | IOSTANDARD = LVTTTL | PULLUP ;
NET "SW<2>" LOC = "H18" | IOSTANDARD = LVTTTL | PULLUP ;
NET "SW<3>" LOC = "N17" | IOSTANDARD = LVTTTL | PULLUP ;
```

Figure 2-2: UCF Constraints for Slide Switches

This tells you how to fill out the information in **PlanAhead** for the switches. I'll put **Cin**, **A**, and **B** on **Sw3**, **Sw2**, and **Sw1**. You do this in the I/O Port Properties pane above the left part of the I/O Ports pane after selecting the pin you want to constrain.

4. The LEDs are also described in the User Guide:

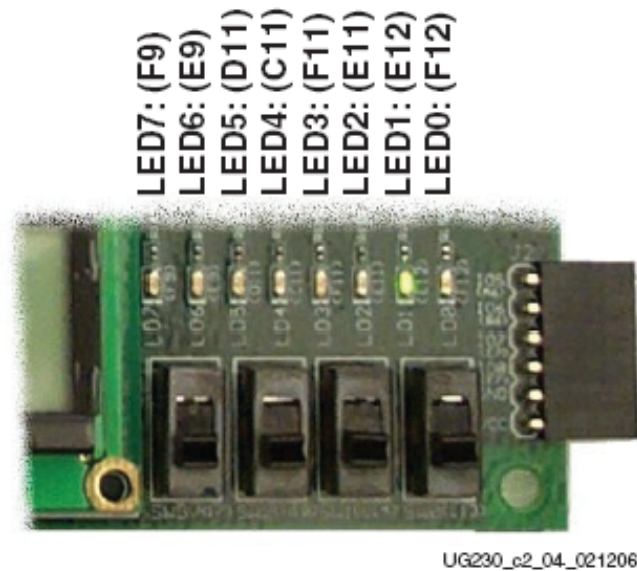


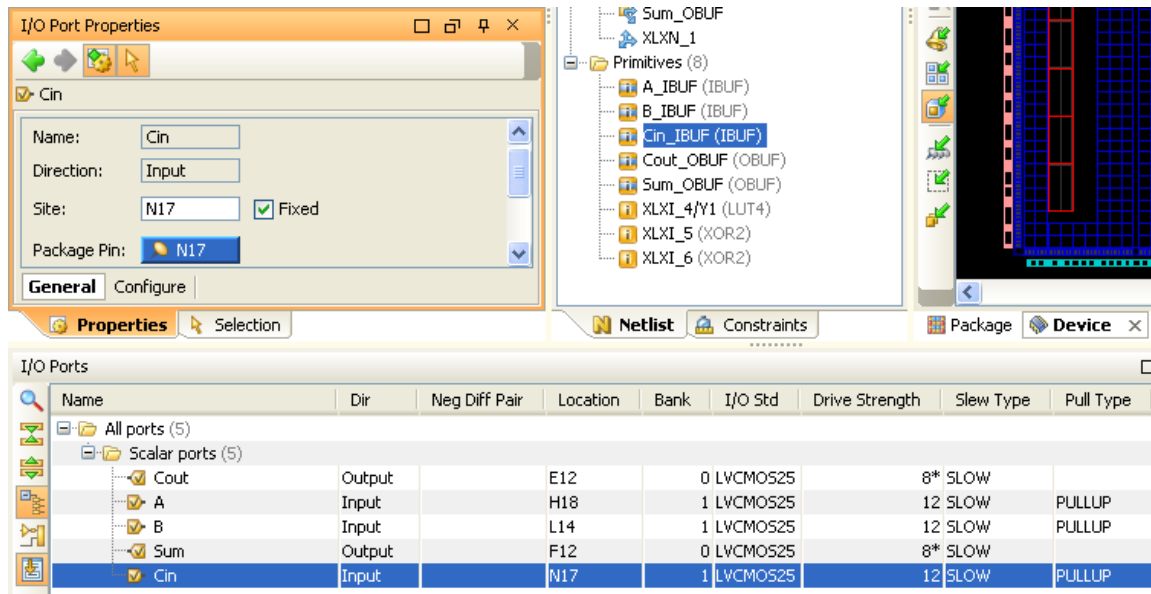
Figure 2-10: Eight Discrete LEDs

```
NET "LED<7>" LOC = "F9" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 8 ;  
NET "LED<6>" LOC = "E9" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 8 ;  
NET "LED<5>" LOC = "D11" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 8 ;  
NET "LED<4>" LOC = "C11" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 8 ;  
NET "LED<3>" LOC = "F11" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 8 ;  
NET "LED<2>" LOC = "E11" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 8 ;  
NET "LED<1>" LOC = "E12" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 8 ;  
NET "LED<0>" LOC = "F12" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 8 ;
```

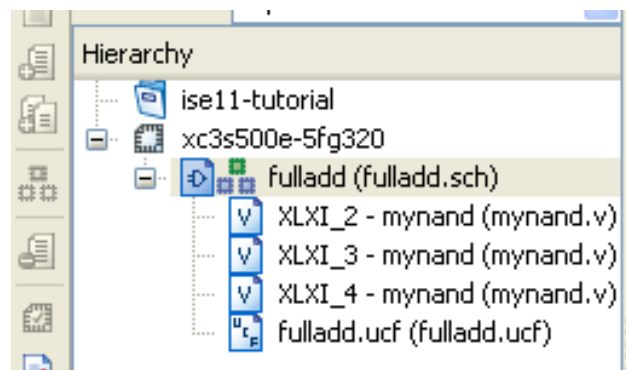
Figure 2-11: UCF Constraints for Eight Discrete LEDs

I'll use **LED1** and **LED0** as **Cout** and **Sum**.

Note that it is important to get all the details of the pins correct as they're described in the manual! The switches won't function properly without the pullup, for example, and the LEDs really need to have the drive strength set. This is done in the **Configure** section of the I/O Port Properties pane.



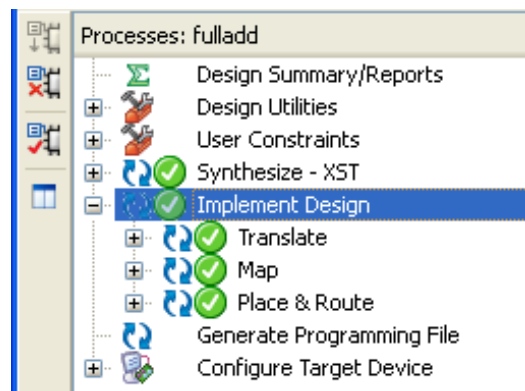
5. Now **Save** the **PlanAhead** settings.
6. When you exit you'll see that a **fulladd.ucf** file has been added to the project.



You can also edit **fulladd.ucf** by opening it in a text editor. It's just a text file with constraints formatted as shown in the User Guide.

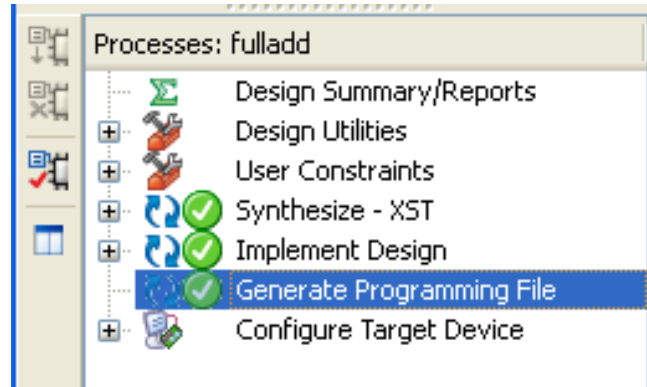
7. With your source file selected (**fulladder** in this case), double click the **Implement Design** process in the **Processes** pane. This will translate the design to something that can physically be mapped to the particular FPGA that's on our board (the xc3s500e-5fg320 in this case). You should see a green check mark if this step finishes without issues. If there are issues, you need to read them for clues about what went wrong and what you should look at to fix things.

8. If you expand the **Implement Design** tab (which is not necessary) you will see that the **Implement Design** process actually consists of three parts:
- (a) **Translate:** Translate is the first step in the implementation process. The Translate process merges all of the input netlists and design constraint information and outputs a Xilinx NGD (Native Generic Database) file. The output NGD file can then be mapped to the targeted FPGA device.
 - (b) **Map:** Mapping is the process of assigning a design's logic elements to the specific physical elements that actually implement logic functions in a device. The Map process creates an NCD (Native Circuit Description) file. The NCD file will be used by the Place and Route process.
 - (c) **Place and Route:** Place and Route uses the NCD file created by the Map process to place and route your design. Place and Route outputs an NCD file that is used by the bitstream generator (BitGen) to create a (.bit) file. The Bit file (see the next step) is what's used to actually program the FPGA.



9. At this point you can look at the **Design Summary** (tab selecting the view in the top right quadrant of the tool) to find out all sorts of things about your circuit. One thing that you might want to check is to click on the **Pinout Report** and check that your signals were correctly assigned to the pins to which you wanted them assigned.

10. Now double click the process: **Generate Programming File**. This will generate the actual configuration bits into a .bit file that you can use to program your Spartan-3E board to behave like your circuit (in this case a full adder).



11. Now that you have the programming file, you can program the Spartan-3E board using the **iMPACT** tool and the USB cable on your PC/laptop. First, make sure that the jumpers on your Spartan-3E board are installed correctly. In particular, check that the configuration options are correctly set. The configuration options are at the top of the board near the RS232 interfaces.

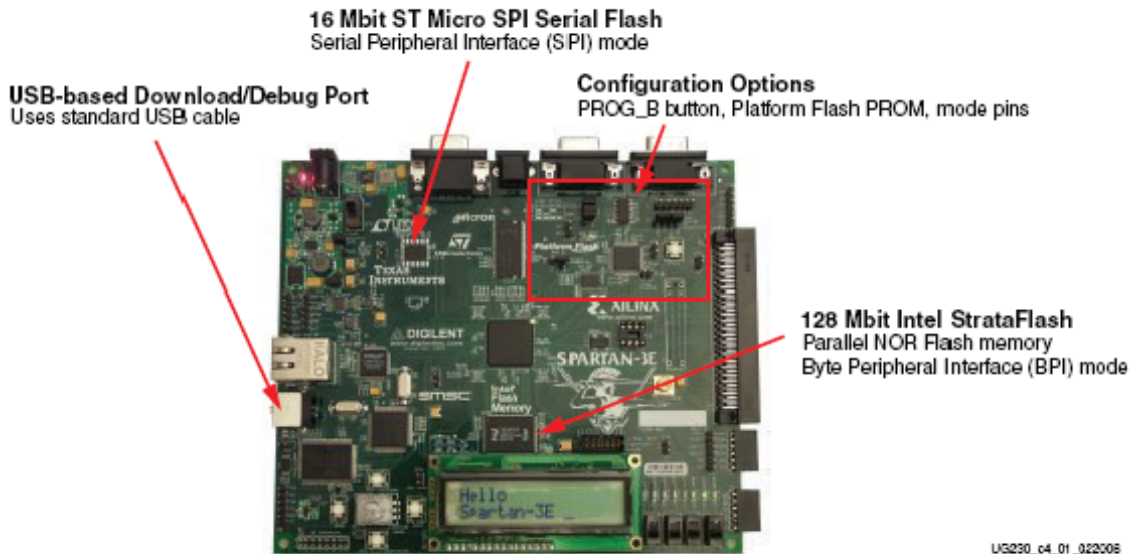


Figure 4-1: Spartan-3E Starter Kit FPGA Configuration Options

The jumpers on the J30 headers must be set for **JTAG** programming. This means that only the middle pins of the header should have a jumper on them. See the following illustration from the User Guide. Your board should look like this!

Configuration Mode Jumper Settings (Header J30)

Select between three on-board configuration sources

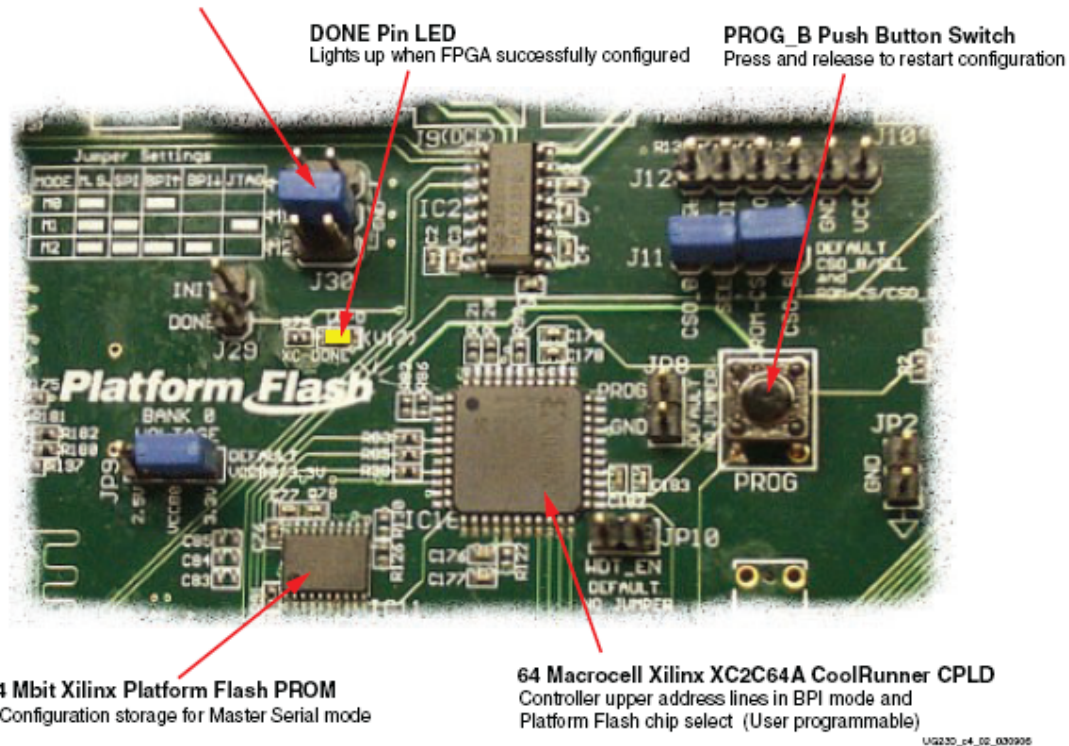
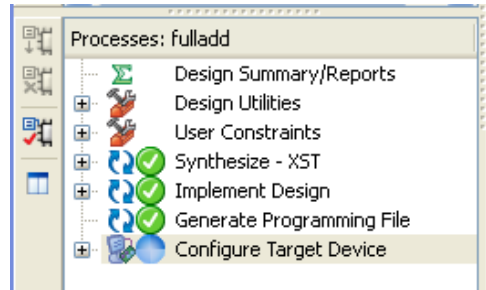


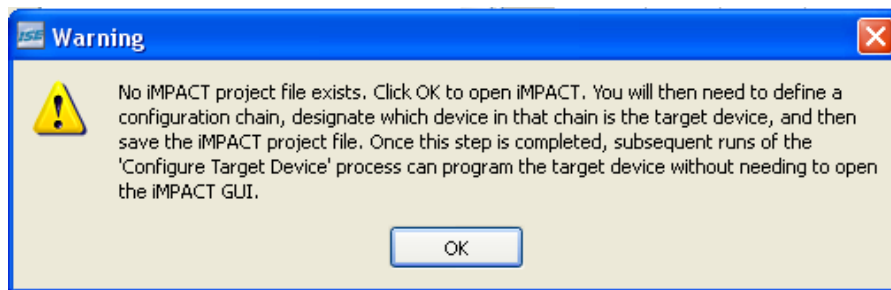
Figure 4-2: Detailed Configuration Options

- Now that you have the jumpers set correctly, you can plug in the power to your Spartan-3E board and connect the USB cable between the Spartan-3E and your PC. The on/off switch is on the top left of the board. Then when you turn on the power, the PC should recognize the Xilinx cable/board and install the drivers. (You may go through a few iterations of installations.)

13. Once the PC has recognized the USB connection to the Spartan-3E board, you can use the Process **Configure Target Device** to start up the **iMPACT** tool to program the FPGA.

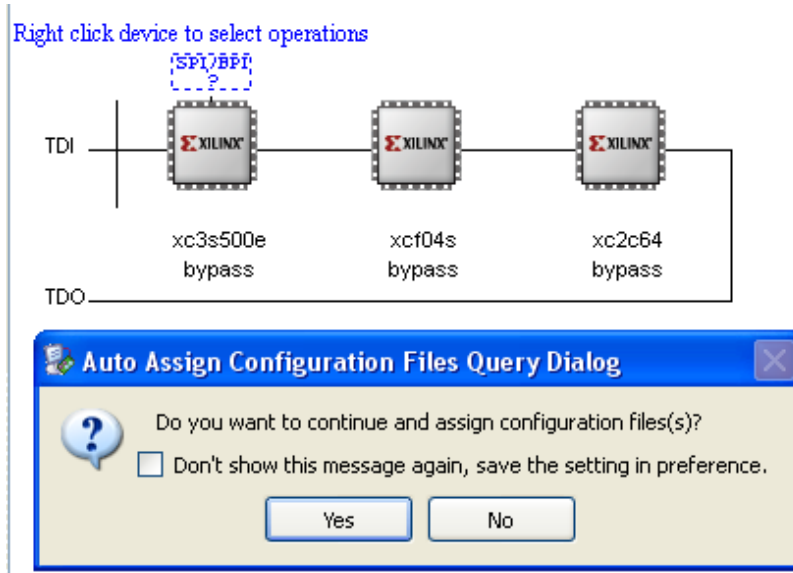


14. The first time you **Configure Target Device** for a new project, you'll get the following message about setting up an **iMPACT** file. You can click OK here and start up the **iMPACT** tool.



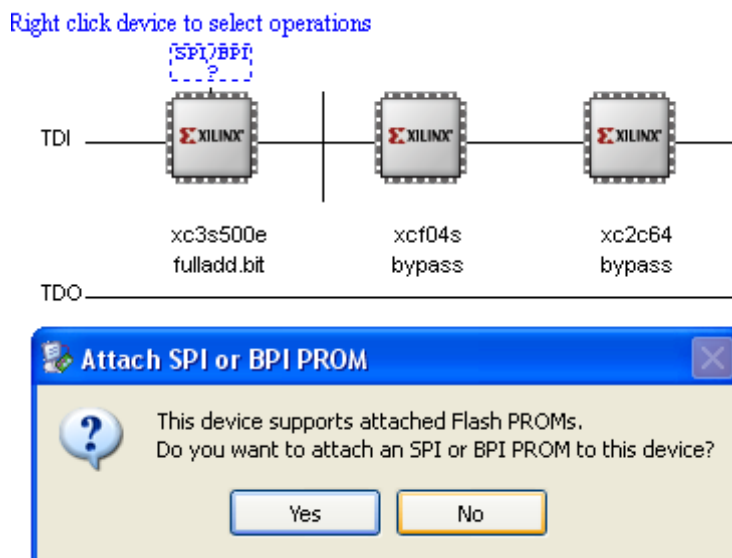
15. Again, the first time you do this for a given project, you'll get the **iMPACT** tool.

16. Double click on **Boundary Scan** in the iMPACT Flows pane. Then right click in the right pane to add a device or initialize **JTAG** chain and select **Initialize Chan...** You'll then get the following menu:



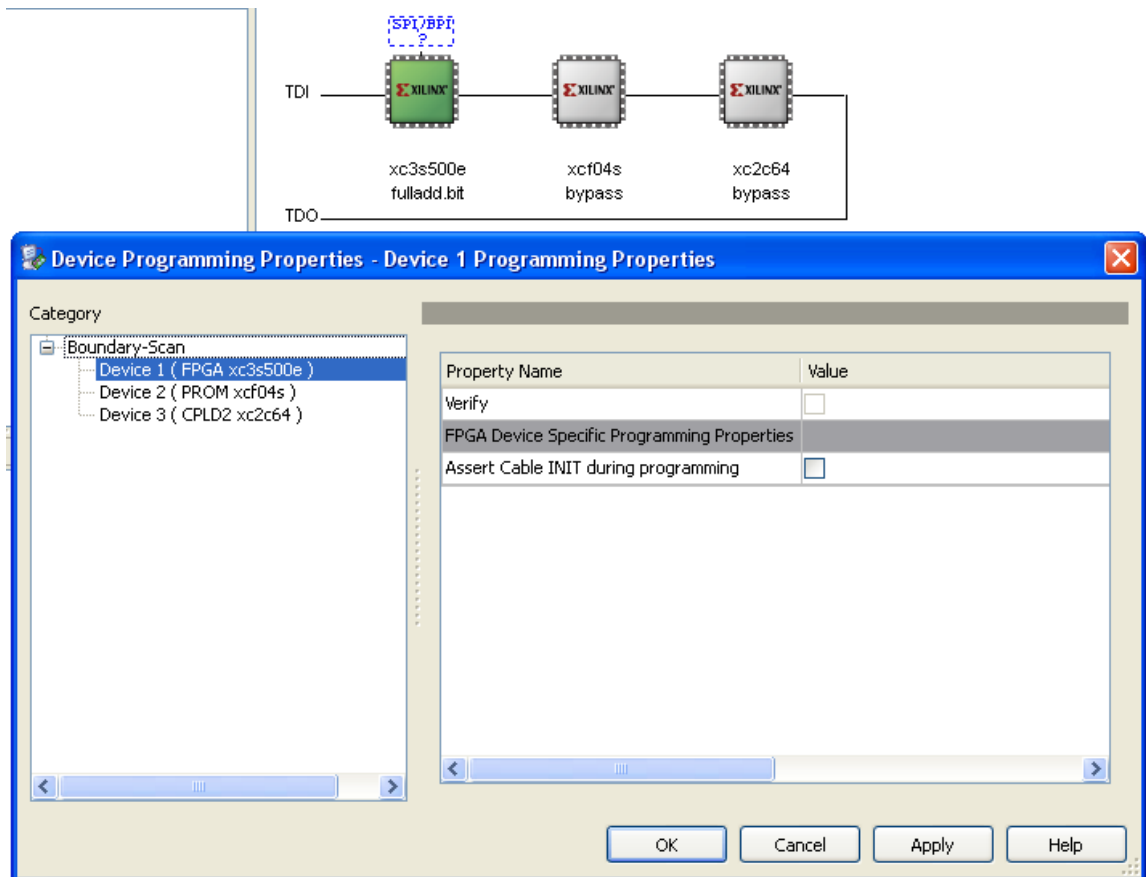
Click **Yes**, browse to your project, and select **fulladd.bit** for the **xc3s500e** device (your FPGA).

You'll then be able to select to including configuration files for the **SPI** ports or **BPI PROM**. Select **No**.

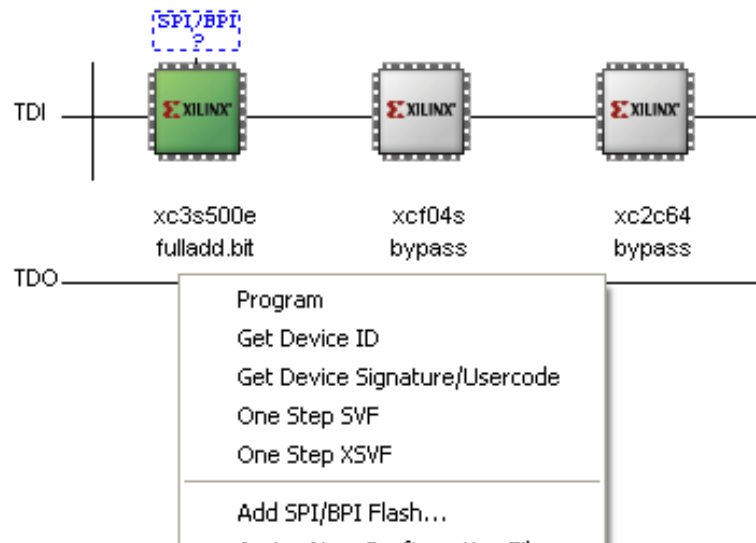


You'll then be asked to assign a configuration files for the other parts. Select **Bypass** for both the **xcf04s** and **xc2c64**.

You'll then see the following screen. You can click **OK** without changing the default settings.



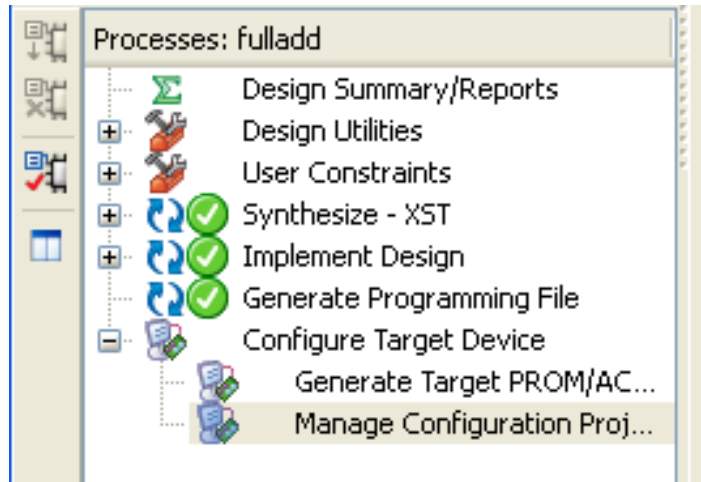
17. Now you can select the Spartan-3E (the **xc3s500e**) and right click to get a dialog. Select **Program** in this dialog to program the FPGA.



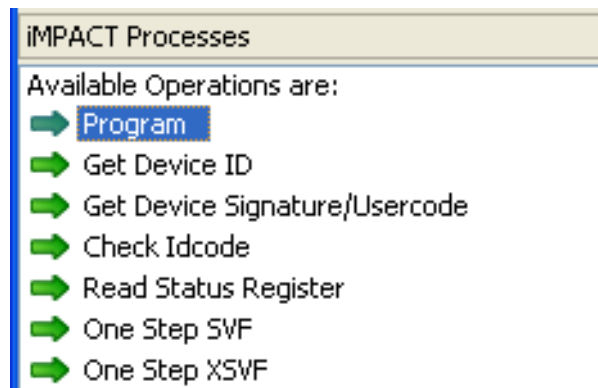
You should see the following indication that the programming has succeeded. You should also see the **xc-done** LED (a little yellow LED underneath the J30 jumper on the board) light up if the programming is successful, and the following should show up in **iMPACT**:



18. Your circuit should now be running on the Spartan-3E board. If you've followed this tutorial you should now be able to set the sw3, sw2, and sw1 switches and look for the full adder output on LED1 and LED0.
19. You can now save your changes as file **fulladd.ipf**.
20. If you make changes and want to reload the bit file to the FPGA (after making changes, for example), you can restart the **iMPACT** tool using the Manage Configuration Project (iMPACT) option under **Configure Target Device** in ISE.



At this point you can right click on the **xc3s500e** as before to program, or double click on **Program** in the iMPACT Processes pane.



That's it! You're done!

5 Overview of the Procedure

1. **Design** the circuit that you would like to map to the Xilinx part on the FPGA. You can use schematics, or Verilog, or a mixture of both.
2. **Simulate** your circuit using the ISE Simulator and a Verilog testbench to provide inputs to the circuit. Use "if" statements in your testbench to make it self-checking.
3. **Generate a UCF file** to hold constraints such as pin assignments (later we'll use the UCF file for other constraints like timing and speed).
4. **Assign the I/O pins** in your design to the pins on the FPGA that you want them connected to.
5. **Synthesize** the design for the FPGA using the XST synthesis tool.
6. **Implement** the design to map it to the specific FPGA on the Spartan-3E board.
7. **Generate the programming .bit file** that has the bitstream that configures the FPGA.
8. **Program** the FPGA using the bitstream with the iMPACT tool.