

# Getting Started with Xilinx Design Tools and the Xilinx Spartan-3 Starter Kit -- a User's Guide

by

Sin Ming Loo, Version 1.02, Boise State University, 2005

Sin Ming Loo, Version 1.01, Boise State University, 2004

B. Earl Wells, Sin Ming Loo, Version 1.0, University of Alabama in Huntsville, 2000

## Introduction

The purpose of this manual is to provide additional support to students in EE/EE 497/597, Digital Systems Rapid Prototyping, who will be using the Xilinx ISE software to rapidly prototype digital design on the Xilinx Spartan-3 Starter Kit (S3Kit). This manual supplements the existing documentation on the S3Kit [1] and the material on the Xilinx ISE 6.3i [2] computer aided design tool by introducing a coordinated set of examples which will take the user through the most common steps necessary for design entry, functional simulation, logic synthesis, and the actual configuration of the Xilinx Spartan-series FPGA on the S3Kit.

The guide is organized into five chapters which cover the following topics:

**Chapter 1:** General information is presented about the S3Kit and the Xilinx ISE 6.3i Design tools.

**Chapter 2:** A simple four bit binary counter design example is introduced and used to show the common steps associated with schematic capture design entry, functional simulation, design implementation, and the Spartan 3 XC3S200 FPGA configuration.

**Chapter 3:** A four bit binary to seven segment LED design example is introduced to illustrate the common steps associated with hardware description language design entry in VHDL, logic synthesis, functional simulation, and XC3S200 implementation.

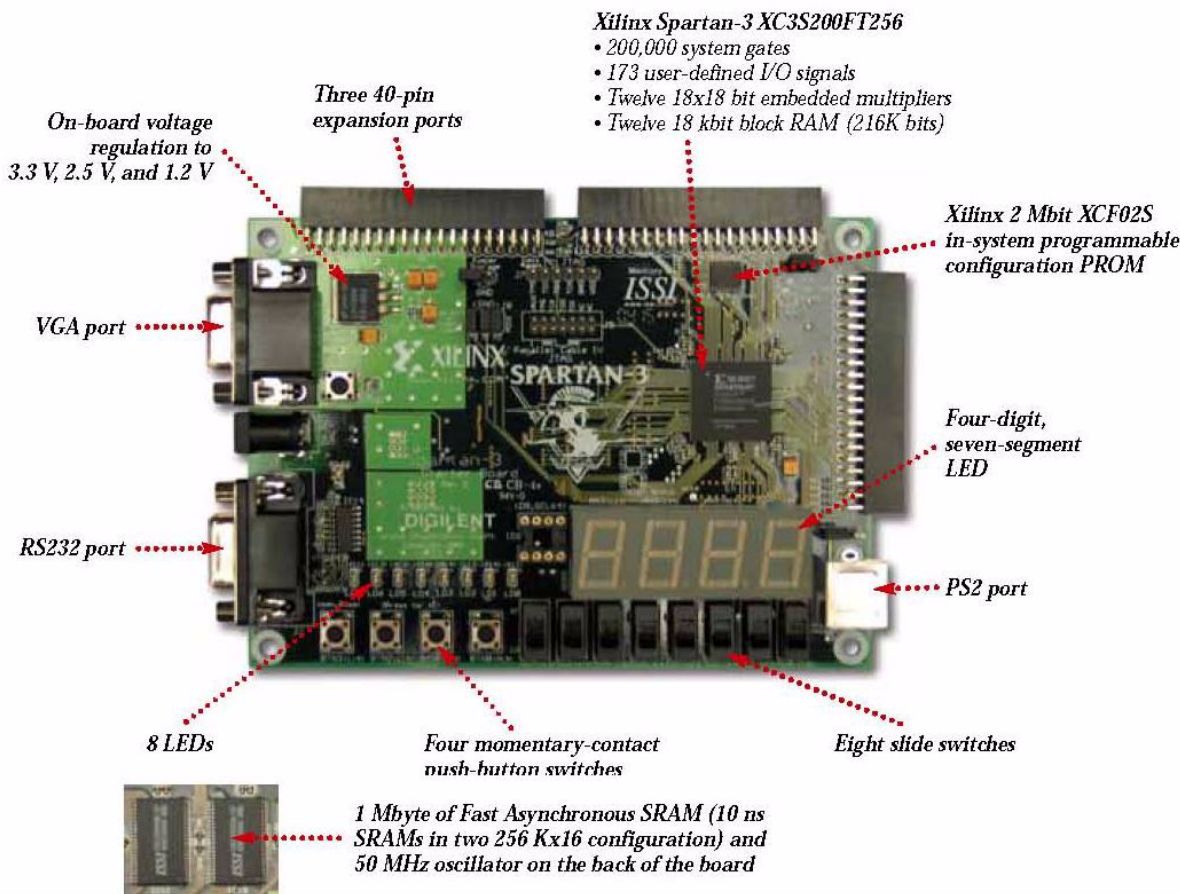
**Chapter 4:** The four bit binary counter and the binary to seven segment LED examples presented in Chapters 2 and 3 are combined to illustrate how designs can be created using hybrid schematic capture techniques and hardware description languages.

**Chapter 5:** References.

## Chapter 1: The S3Kit and Xilinx 6.3i Computer Aided Design Tool

### The S3kit Board

The S3Kit has many features that facilitate experimentation in reconfigurable logic design as well as the general rapid prototyping of digital logic. The S3Kit is shown below in Figure 1.1.



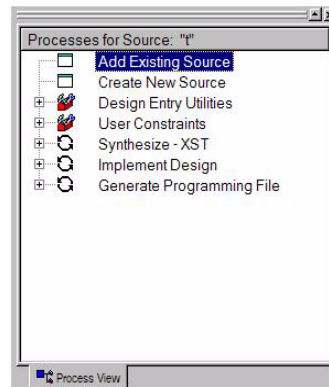
**Figure 1.1: The Xilinx Spartan-3 Starter Kit.**

The S3Kit is a stand-alone board for experimenting and developing prototypes with FPGAs using Xilinx FPGA architecture. The S3Kit has one Xilinx Spartan XC3S200 FPGA. An on-board 50 MHz oscillator drives the FPGA. More information about this S3Kit board can be found in [1].

### The Xilinx ISE Foundation Series Software

Designs that are entered on the S3Kit require the use of special Computer Aided Design, CAD, software to configure the Xilinx Spartan-series FPGA. The software used by students is the Xilinx ISE Foundation Series tool suite. This software runs on a PC under Windows/Unix and it

supports design entry (via schematic capture, state diagram entry, and hardware description language), logic synthesis, logic simulation, timing analysis, and device configuration.



**Figure 1.2: The Design Cycle Supported by Xilinx ISE Foundations Series Software.**

The general engineering design cycle which is supported by the Xilinx Foundation Series CAD software is highlighted in Figure 1.2. It includes the design entry, synthesis, simulation, implementation, verification, and programming phases, the function of which, will now be briefly described.

**Design Entry:** In this stage of the design cycle the design is specified in a form that is recognizable by the Xilinx design automation tools. Xilinx tools support design entry using schematic capture, hardware description languages (Verilog/VHDL, state diagram specification or a combination of these techniques.

**Synthesis:** Whenever a design is entered using a high-level language or state diagram specification the design automation tool must synthesize the relatively abstract representation into its low-level logical representation. This step is not necessary for components of the design that have been entered through schematic capture. The Xilinx tools support this option for several HDL's (e.g. ABEL, VHDL, and Verilog).

**Simulation:** This phase of the design process allows for the logical correctness of the design to be validated before it is implemented. As design errors are exposed corrections will often be made by repeating the design entry portion of the design cycle.

**Implementation:** This phase is concerned with converting the design information into a form that can be used to configure the targeted FPGA/CPLD device so that it will behave in the manner that is intended.

**Verification:** This phase of the design is used to verify that the resulting implementation meets timing and other constraints. This is very important for high speed designs. The examples in this document do not utilize this phase of the design cycle.

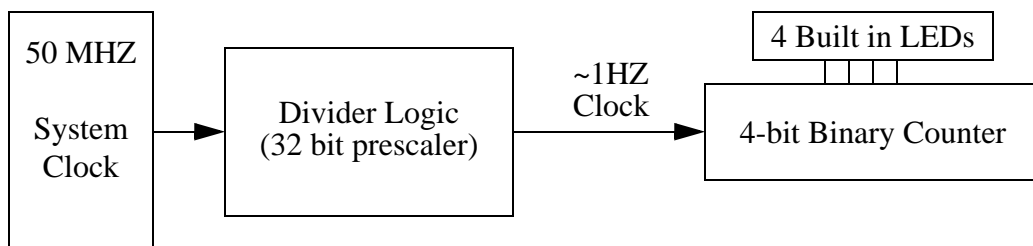
*Programming (Device Configuring):* The resulting bit stream that was produced during the implementation phase to represent the design is downloaded directly (or indirectly through flash memory, etc.) to the targeted device. There are several programming standards supported by the X3Kit (such as JTAG, Xchecker and MultiLINX). The design examples discussed in this manual will utilize the standard parallel port which utilizes a parallel cable supplied by S3Kit which connects to the parallel port of a typical PC or unix workstation.

**Note:** In the following Chapters, the XC3S200FT256 is selected for logic/design implementation.

## Chapter 2: Schematic Capture Design Example


### Example

In this chapter, a simple four-bit binary counter will be used to illustrate the common steps needed to enter a digital design using schematic capture techniques, perform functional simulations, implement the design on the S3Kit (locking the Xilinx XC3S200 I/O pins to specific signals specified in the design), and configure (program) the XC3S200. The binary counter design which will serve as a simple example is shown in Figure 2.1. In this design, the four outputs from the binary counter will each be connected to a built-in LED on the S3Kit. The counter will be clocked by a relatively slow clock ( $\sim 1\text{HZ}$ ) to allow the operation to be viewed on the S3Kit by the user. The clocking signal will originate from an internal 50 MHz oscillator which will be directed through a 24 bit prescaler network. Both the 4-bit counter and the 24 bit prescaler circuit will be implemented using Xilinx's counter macros.

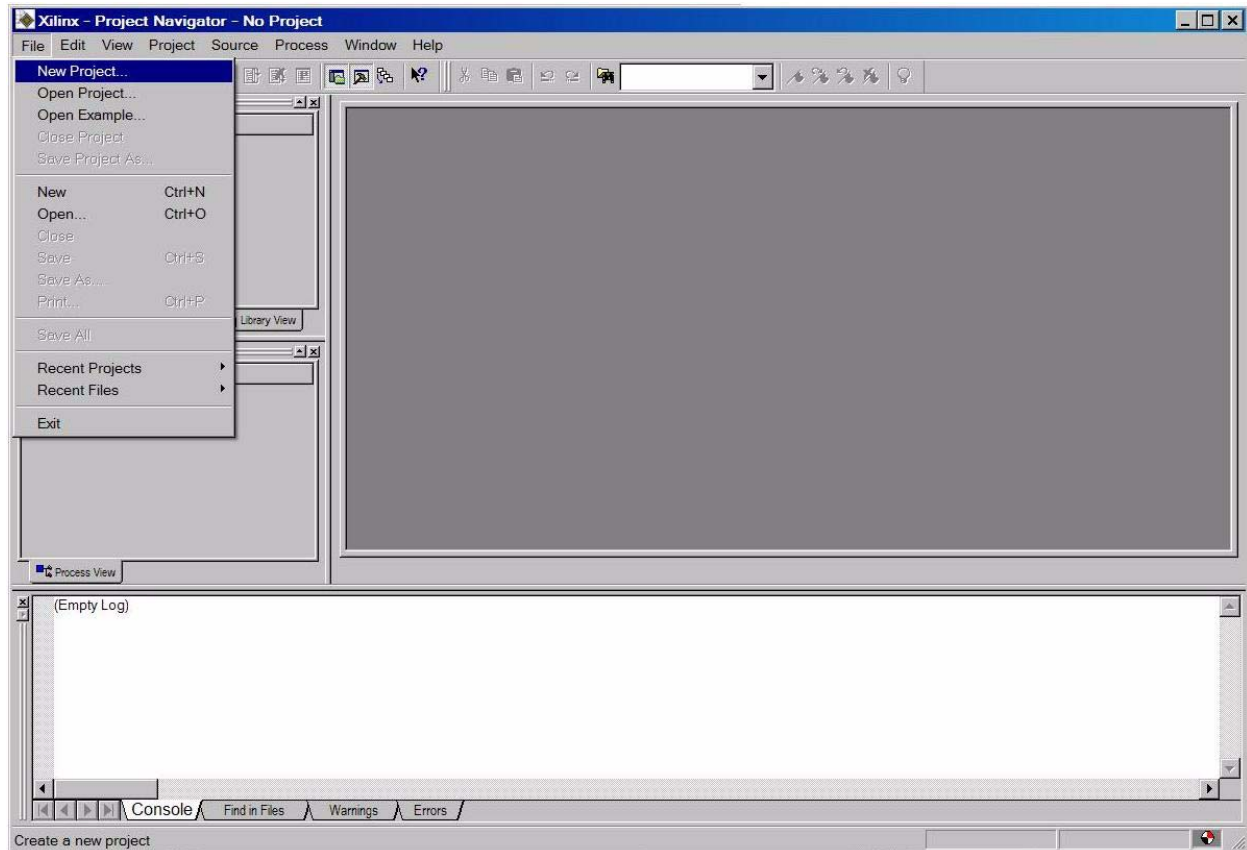


**Figure 2.1: Binary Counter Example**

### Setting up a Project

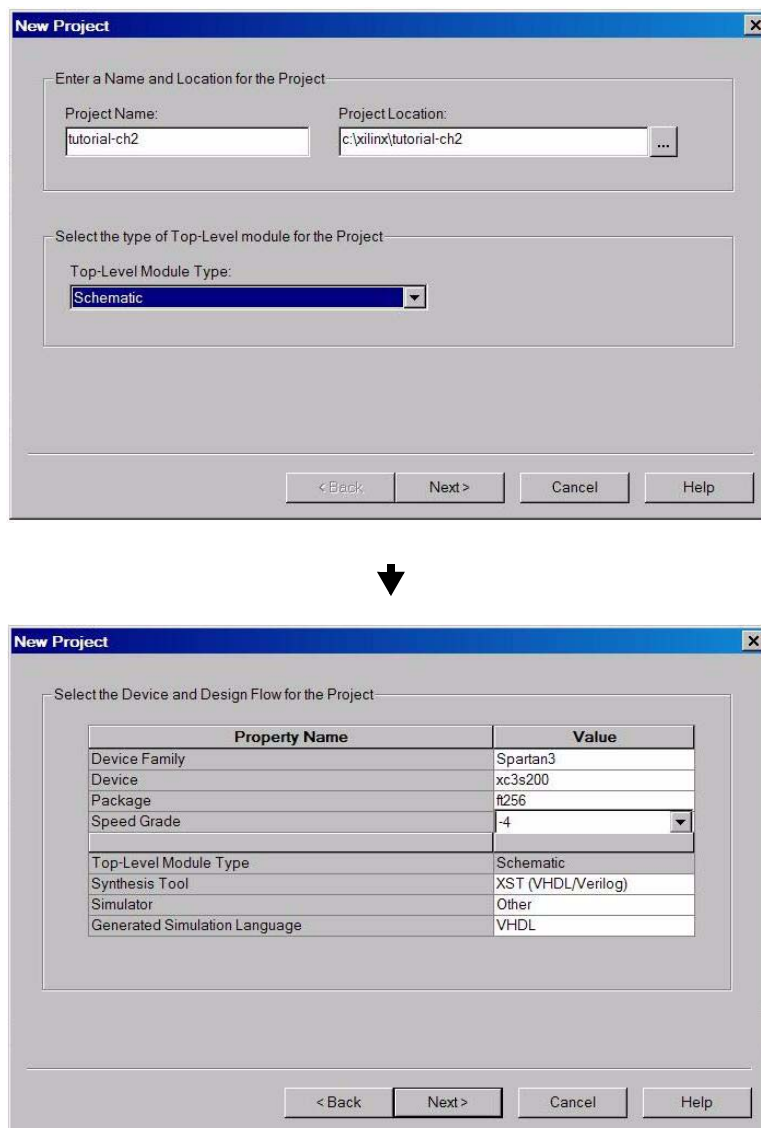
To enter this design, first start the **Project Navigator** of Xilinx ISE 6.3i by double clicking on the **Project Navigator** Icon,  of ISE 6.3i from the Window's Desktop.

The Project Navigator will then display the last project the tool worked on. If the previous project is accessible, it displays the project's flow. Otherwise, the tool will show a warning (This warning can be ignored!). For a new project, select the **File** menu and select **New Project** as shown in Figure 2.2. This window allows the user create a New Project. For the binary counter design example, we would like to create a **New Project**.



**Figure 2.2: Xilinx ISE Series Software New Project Menu.**

A **New Project** window will then appear. It will have the following fields: **Project Name** (Name of project), **Project Location** (location to store files), and **Project Device Options**. Name the project and place it as your desire. Since we are demonstrating schematic capture design capture techniques in this chapter, the input source is schematic diagram and the **Design Flow** for this design can be set to XST VHDL or XST Verilog as shown in Figure 2.3. The targeted FPGA device is a Xilinx Spartan 3 XC3S200 family device, specifically a XC3S200FT256 FPGA (it is written directly on the device).

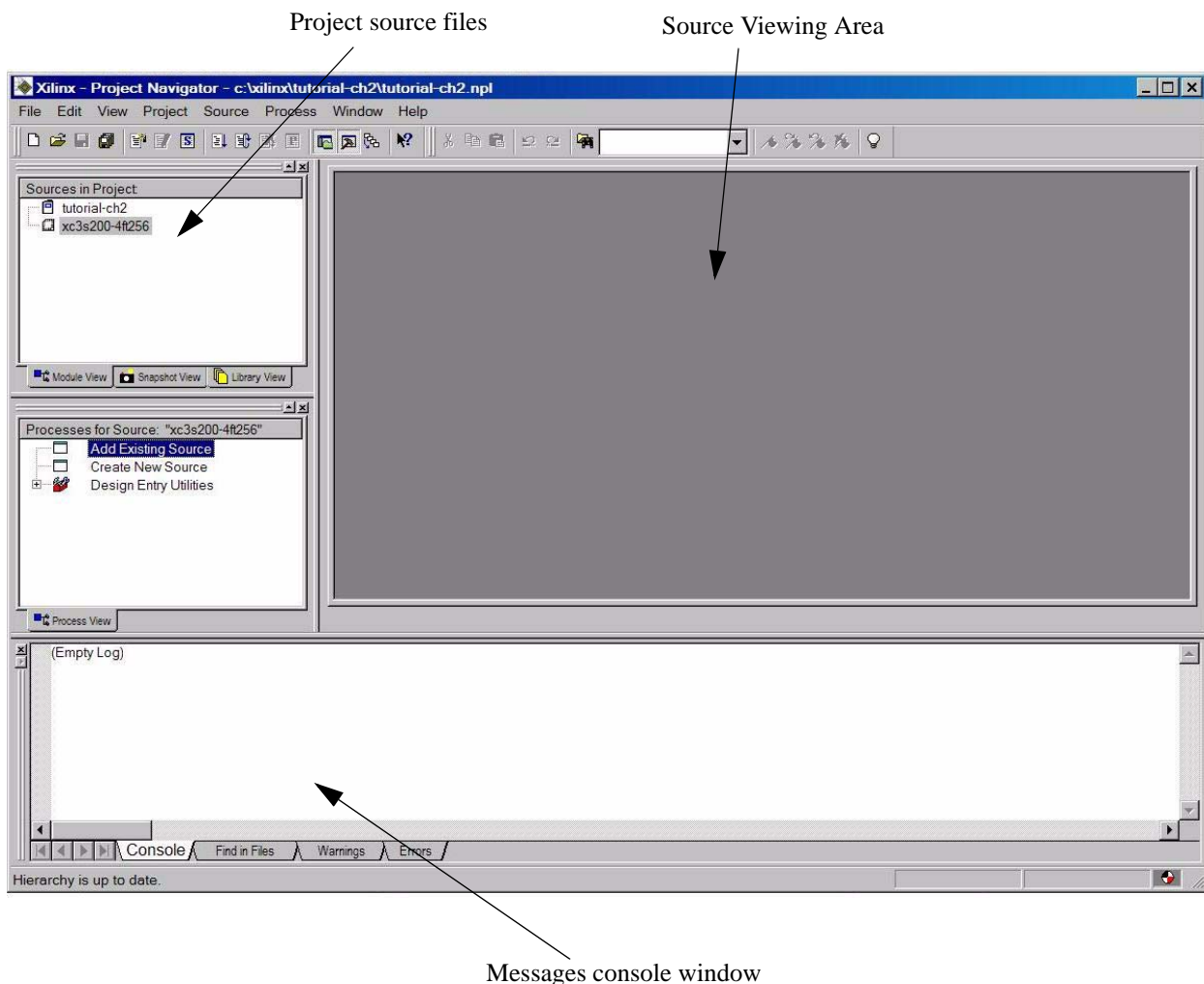


**Figure 2.3: New Project Window**

When all the fields are completed as shown in Figure 2.3 then single click on the **Next** button THREE times (including the add source windows, we will add new source next) and **Finish**

Button at the third window. The Xilinx Foundation Series **Project Navigator** window will now appear as shown in Figure 2.4.

On the left side of the **Project Manager** window under the **Files** tab, the project source files are displayed in a hierarchical (tree structured) manner. To add files to the project (you will be shown how to add a schematic file as source), point the cursor to the top level (xc3s200-ft256) of the source tree, then press and hold the right mouse button and select the **New Source** option (or you can add new source from **Project** pull-up menu). This will cause the file that was added to be included as part of the design and be utilized by the Xilinx CAD software during the project compilation process. To delete a file from the project, used the same method but this time select the **Remove** option. (One should be aware that when a file is removed from the project, it is not actually deleted from the hard disk. It is just not included in this project)

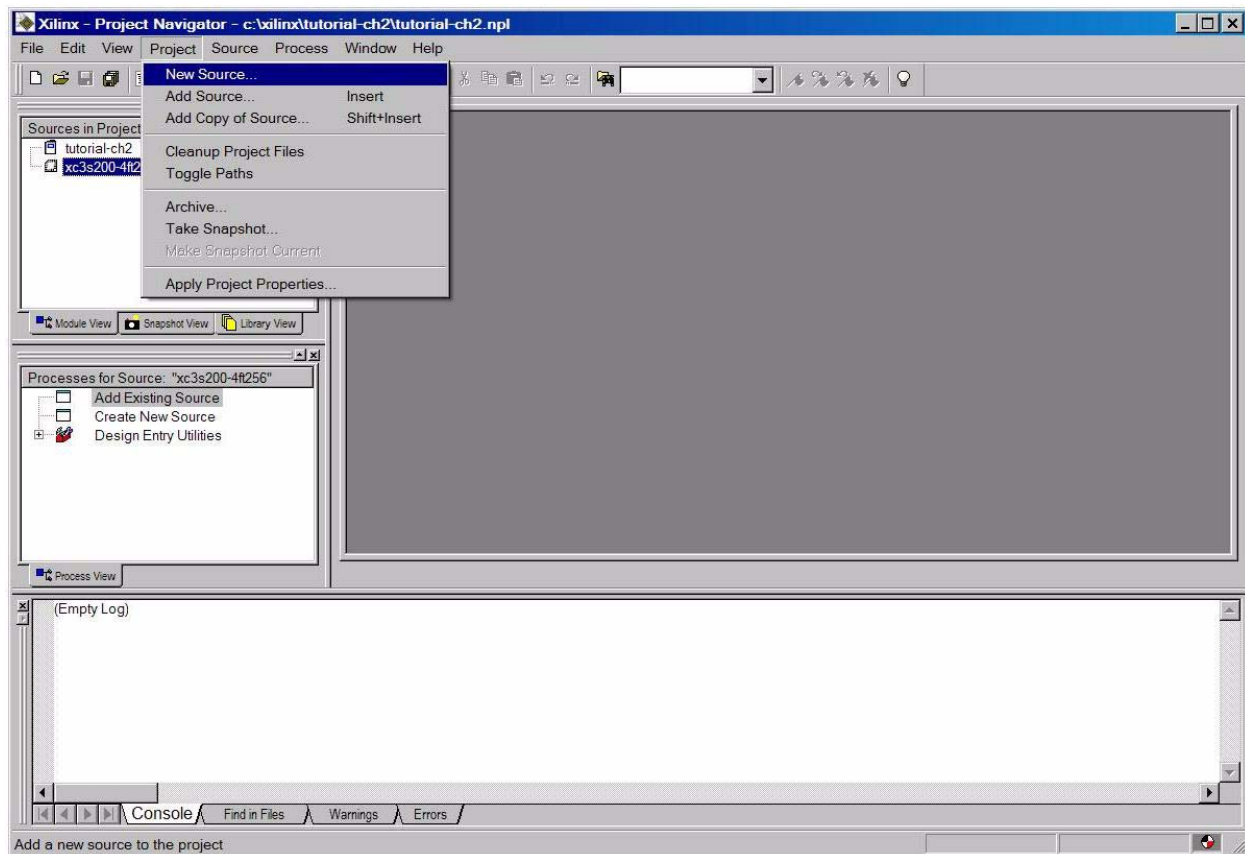


**Figure 2.4: Xilinx ISE Project Manager Window**

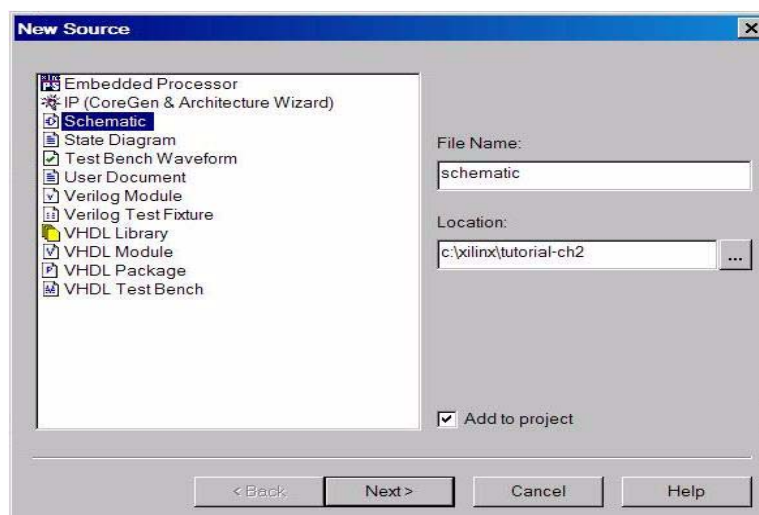


## Design Entry

Design entry using schematic capture techniques involves employing a CAD tool (such as the Xilinx ISE software) to enter a complete logic level schematic. This requires that the user use the CAD tool to graphically enter the symbols that represent each of the components that make up the design and then use the tool to ‘wire’ the components to one another to form the complete schematic diagram. To enter this mode in the Xilinx Foundation software select the **Project** menu on **Xilinx Project Navigator** (Figure 2.5a). At the **New** window, select *Schematic* and enter a **File Name**. This is shown in Figure 2.5b. An informational window will appear with the schematic setup once you click **Next**. To enter the schematic workspace, click **Finish** at the informational window. You should be at what is shown in Figure 2.6.

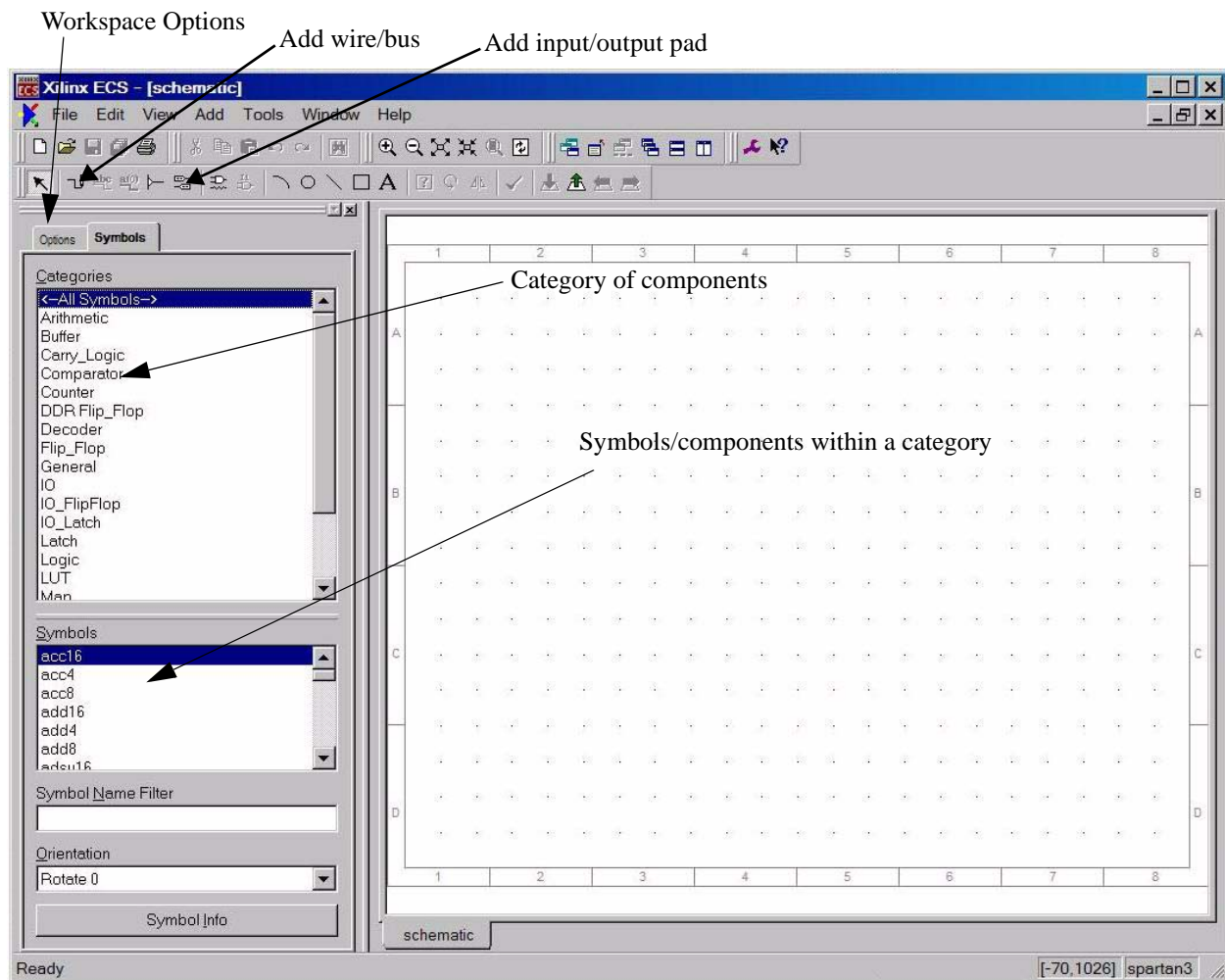


(a) Project Selection Menu



(b) Source Selections

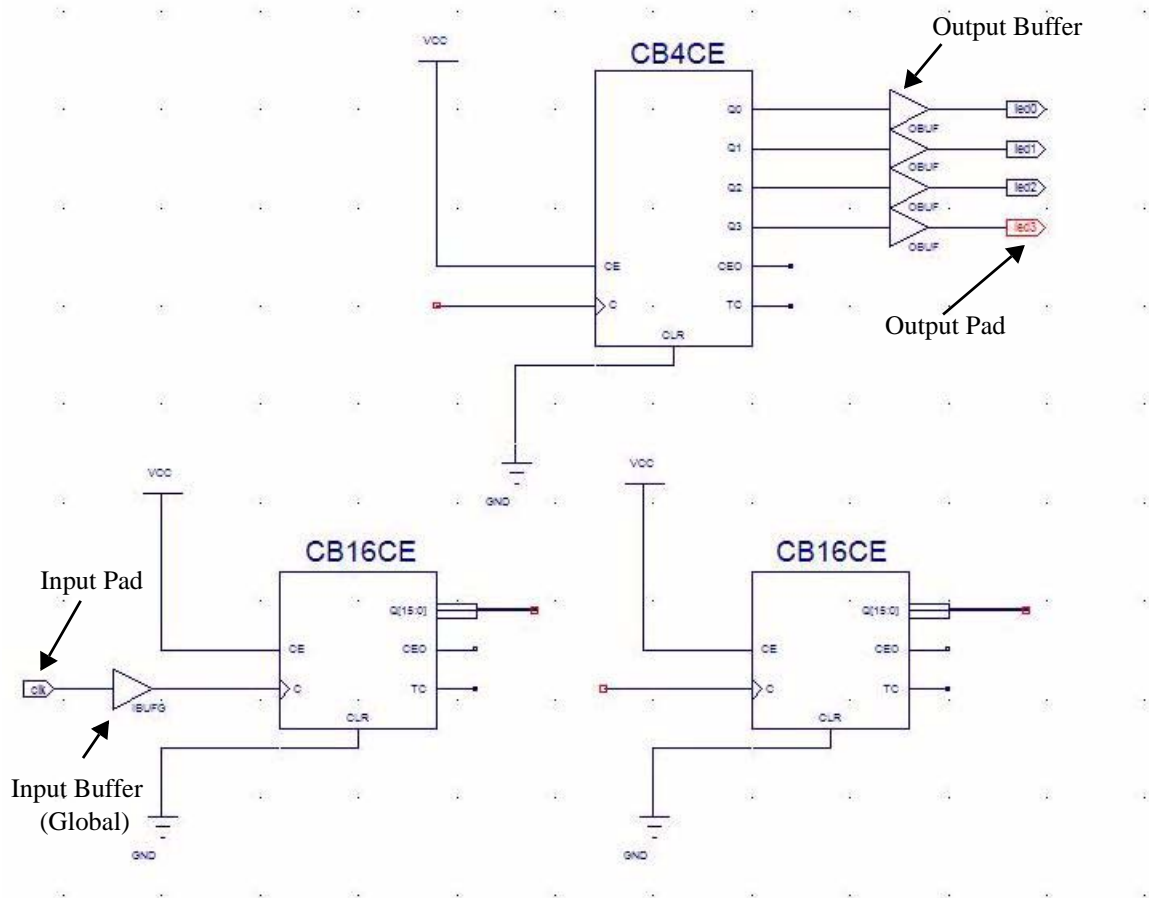
**Figure 2.5: Create Schematic Source**



**Figure 2.6: Schematic Editor Window**

In the case of the four-bit binary counter example, components are needed for the internal logic for the prescaler and counter, and components for the I/O pads and buffers. We will use the high level macros which are provided by Xilinx (these macros are themselves created in a hierarchical manner using low level gate and flip-flop primitives). To implement the design we have chosen to use two type of binary counter macros from *Counter Category*, the **CB4CE** to form the basic four bit counter, and two instances of the **CB16CE** macro to form the prescaler logic. (Note: When you have completed placing a symbol, press once on the Escape key is a quick way to exit the selection before you move on to the next selection!) This design requires one input, a clock signal, and four outputs, the outputs of the 4 bit binary counter. Interfacing with the outside world like this requires that appropriate I/O pads and I/O buffering components be used. In this case will need one instance of the components, **I/O Maker** or **I/O Pad**(To add a pad to the input buffer or

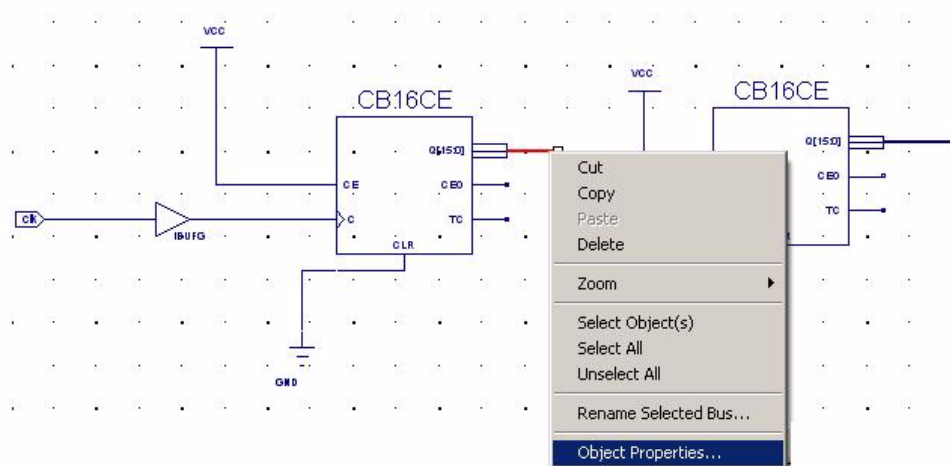
output buffer, add a small segment of wire to the buffer and click on the pad selection button as shown in Figure 2.6, select input or output pad. Then, click at the location of where the pad is to be added.), and **IBUFG**, and four instances of the components **I/O** (direction on adding output maker coming up!), and **OBUF**. The **IBUFG** and **OBUF** symbols are from **I/O Category**. From **General Category**, you can find symbols for **Vcc** and **Gnd**. It is assumed that the output of this design will drive four of the external discrete LEDs. These LEDs are wired up in a manner where a logic high will light them up and a logic low will deactivate them (common cathode configuration).



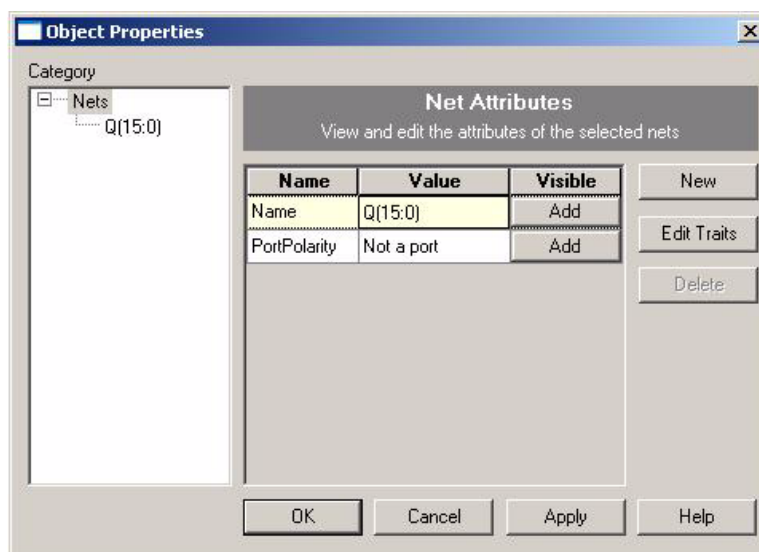
**Figure 2.7: 4-bit Binary Counter Example**

Figure 2.7 shows the completed four-bit binary counter schematic. In this design, the components are all wired together using a combination of nets (wires) and simple busses (collection of nets). This was accomplished in the case of individual nets by first invoking the net/bus drawing tool by pressing **Ctrl-W** or the second button on quick-access toolbar (as shown in Figure 2.7).

Then the cursor was placed at the point where each net begins (such as a pin on a component) after which the left mouse button was pressed once (single clicked). The cursor was then moved to the desired termination point (such as another component pin in the design) and the left mouse button was then pressed (single clicked). Busses were created in a similar manner. Now, place a short segment of bus as shown in Figure 2.7 to both of the **CB16CE** symbols. Select the bus added to the left **CB16CE** by a left mouse click and click the right mouse button for a working menu. You need to select **Object Properties** (Figure 2.8a). Once **Object Properties** has been selected, edit the value field as shown in Figure 2.8b. The value field has been re-named to **Q(15:0)**.



(a)




(b)

Figure 2.8: Object Properties

You should continue the similar process for the *right* **CB16CE** symbol and named that bus **QQ(15:0)**. The on-board clock (50 MHz) is connected to the *left* **CB16CE**. The most significant bit (**Q15**) of the output of the *left* **CB16CE** is used as clock for the *right* **CB16CE**. A connection between output signal **Q(15)** and clock input at the *right* **CB16CE** can be accomplished with name association. In order to do this, place a short segment of wire at the *right* **CB16CE** clock input, with steps shown in Figure 2.8, change the value field **Q(15)**. For the clock input of the 4-bit counter (**CB4CE**), **QQ(8)** from the *right* **CB16CE** is to be used as input.

It should be noted, that the counter macros used in the design shown in Figure 2.7 all have the inputs **CE** (chip enable), and **CLR** (clear) which need to be tied to logic high and low, respectively. This is accomplished by incorporating the components **Vcc** and **Gnd** and wiring them up to the appropriate pins using the net command.

Internal nets and busses do not always have to have user specified names (for example consider the nets which connect the inverters to the output buffers in the binary counter design). As a general rule-of-thumb, however, one should always explicitly name the I/O nets (nets which connect to I/O PADS), internal nets which connect to busses, and any internal net which the user would like to appear in a simulation.

A couple of other items which should be pointed out include the fact that after a design is entered, component positioning can be easily changed simply by entering the select and drag mode (by pressing the  button) and clicking and holding the left mouse button after the cursor is placed on the object to be moved. Also entire sections of the design can be accessed as a single entity by using standard windows based techniques to select multiple components, components in a rectangular area, etc. These sets of objects can then be placed in the Windows clip board for inclusion in other designs. Discussion of these techniques is beyond the scope of this manual but such techniques can greatly speed up design entry as one gains hands-on experience.

After the design has been entered, **Save** the file (It doesn't matter if you don't close the **Schematic Editor**, just remember to save the file. You can check your schematics diagram for error by using the **Check Schematics** (Under **Tool** pull-down menu) feature provided by the tool. Then return to the **Project Navigator** window.

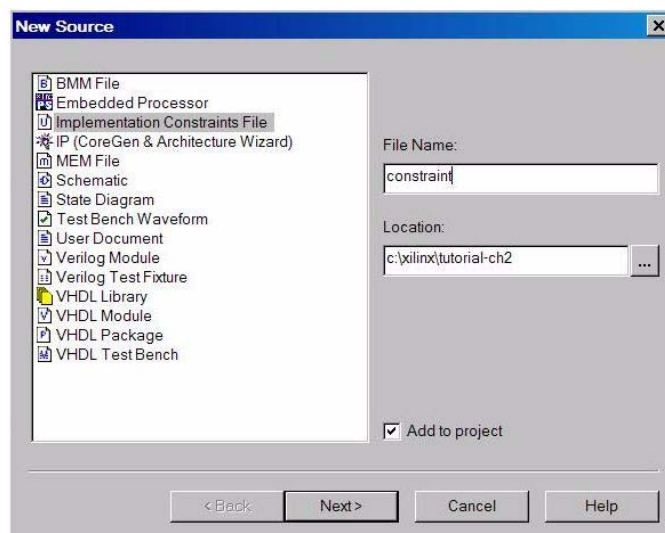
## User Constraints Entry Phase

After the design is entered and saved, it is the job of the Xilinx Foundation software to convert the user entered schematic into a form that can be used to configure the targeted FPGA/CPLD device so that it will behave in the manner that is intended. This conversion process results in the mapping of the desired the logical configuration into a form that can be implemented within the internal FPGA architecture. It is at this point that users have the option of specifying certain constraints which the complex mapping algorithms (place and route) [3,4] must adhere to when making the implementation files. There are a number of such constraints which can be specified (worst case timing, etc.), but the one that is of most importance in the binary counter design is which pins on the Xilinx XC3S200 chip are going to be assigned to the input/output nets of the design. In the binary counter design, there is only one input pin that runs from the S3Kit clock circuit into the Xilinx XC3S20E chip and there are only four output pins that are to connect the external set of discrete LEDs.

Fortunately, the Xilinx libraries which come with each of the supported Xilinx FPGAs and CPLDs have a set of symbolic names which correspond to each I/O pin on the chip. A cross reference can be created which specifies to the Xilinx software which symbolic pin name on the FPGA/CPLD is to be assigned to the logical I/O net name used in the design. This cross reference operation is contained within the **User Constraint File** which is associated with each project. If such information is not in the **User Constraint File**, the Xilinx software will arbitrarily assign FPGA pins to the design's I/O nodes. (Note: there is a general rule -- the more constraints entered by the user the more inefficient the implementation processes is in terms of processing time and complexity of the design that can be implemented in a given FPGA/CPLD. In tightly packed/high speed cases, it may be desirable to have the Xilinx software make its own I/O pin assignments and external PC board circuitry designed to adhere to this placement [5].)

To lock a specified pin name with the logical I/O net name specified in the schematic (or HDL for that matter), from the **Project Navigator** window, move the cursor to point to the top hierarchy source tree located under the **Sources in Project** on the left-hand side of the window. Click on **Project** pull-down menu and select **New Source** as shown in Figure 2.9. The user constraint file has extension **ucf**. Click next and observe the setup of the following two windows. No changes are required for the next two windows. There are at least two methods to add constraints

to the constraints file. The first method is using the GUI. The second method is by entering the text. We will utilize the second method.



**Figure 2.9: Entering User Constraints from the Project Navigator**

For this design the desired schematic name to Xilinx XC3S200 Pin name cross reference (i.e. Pin Locks) is shown in Table 2.1.

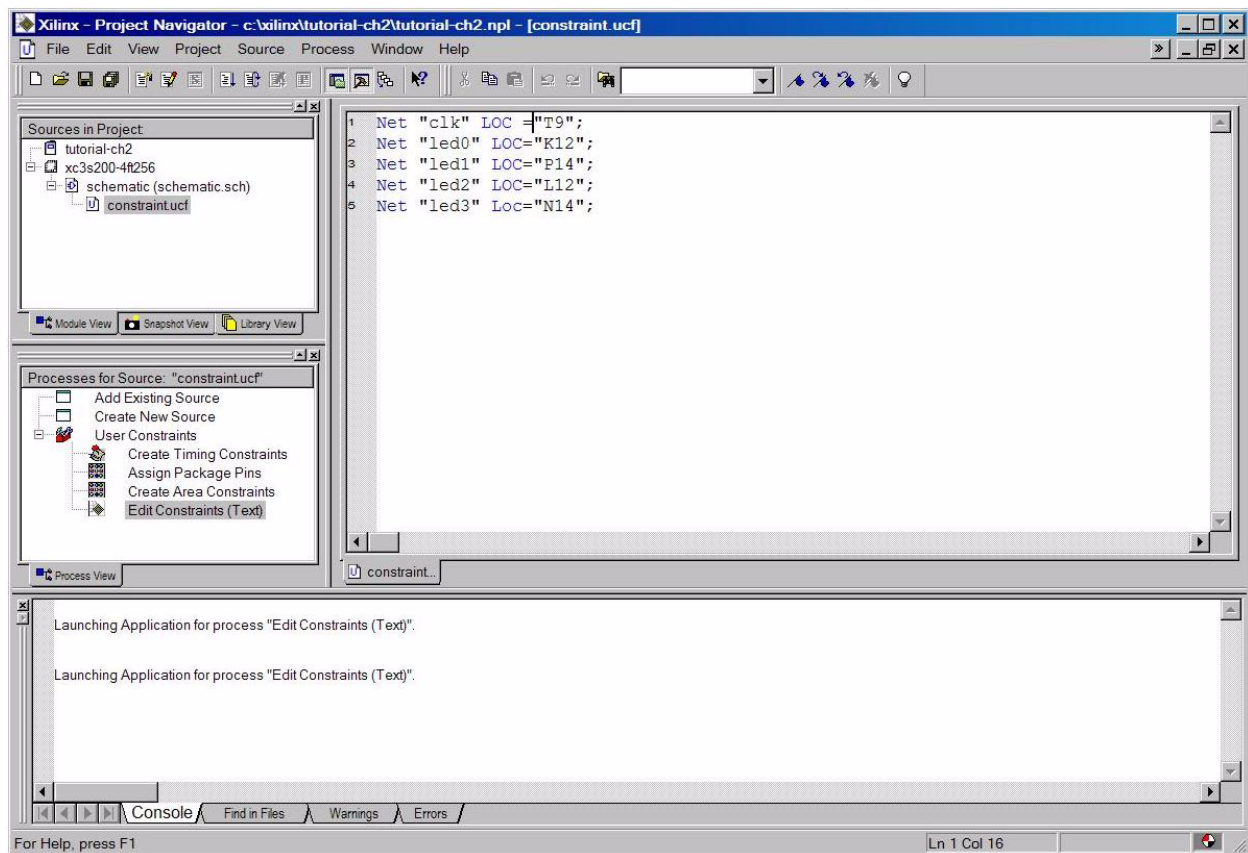
**Table 2.1: Desired Pin Locking (Cross Reference) Configuration**

I/O Pin Description	Schematic Net Name	Xilinx XC4010XL Pin Name
50 MHZ Clock	CLK	T9
Discrete LED #1 (Active High)	BIT0	K12
Discrete LED #2 (Active High)	BIT1	P14
Discrete LED #3 (Active High)	BIT2	L12
Discrete LED #4 (Active High)	BIT3	N14

Now open the constraint.ucf by going to the **File** pull-down menu and use the **Open** function. The resulting additions to the **User Constraints File** for the binary counter example are shown in Figure 2.10 where the cross reference described in Table 2.1 was implemented using the **NET <schematic net name> LOC=<symbolic Xilinx pin name>**



construct. After these additions have been made to the **User Constraints File** one should **Save** the work, **Exit** the Report Browser, and return to the **Project Navigator** window. You can double-click on the *constraint.ucf* under **Sources in Project** to view the changes you have made.



**Figure 2.10: Adding Pin Locking Information to the Project Constraint File**

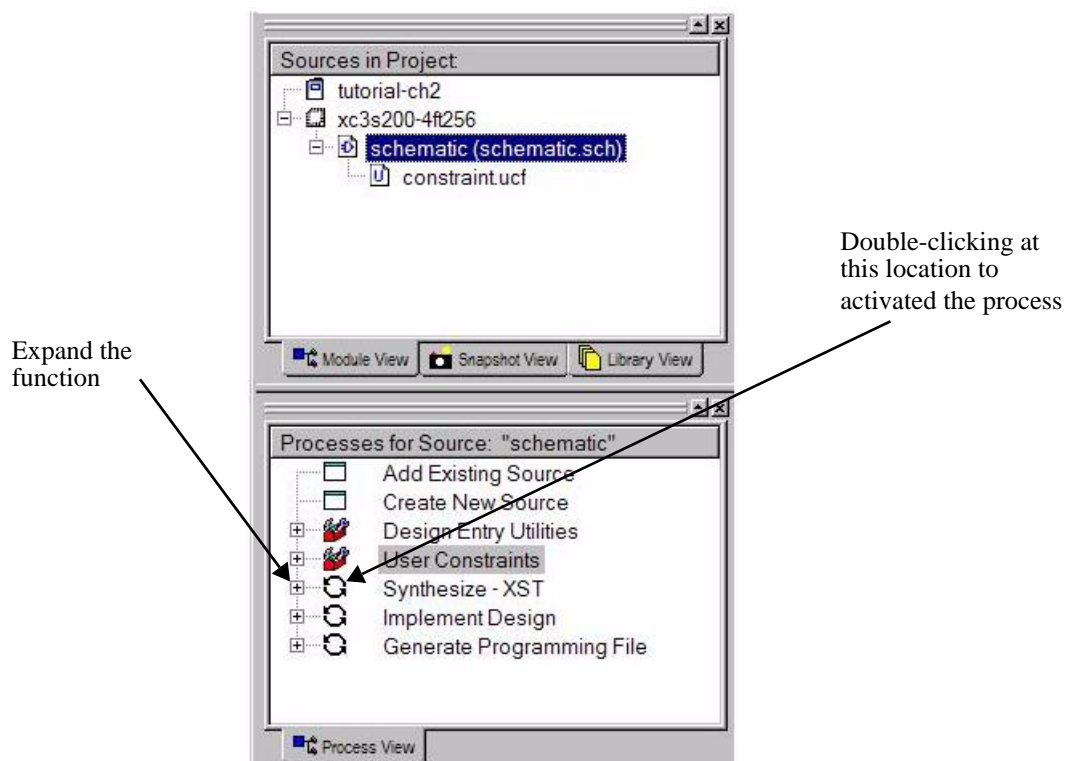
## Functional Simulation Phase

This is not a good example to show the working of functional simulation. Another lab will be presented with ModelSim as the simulation tool.

## Synthesis/Implementation/Generate Programming File Phases

Now it is time to take the design through the compilation (consists of synthesis, implementation, and generate programming file) process.

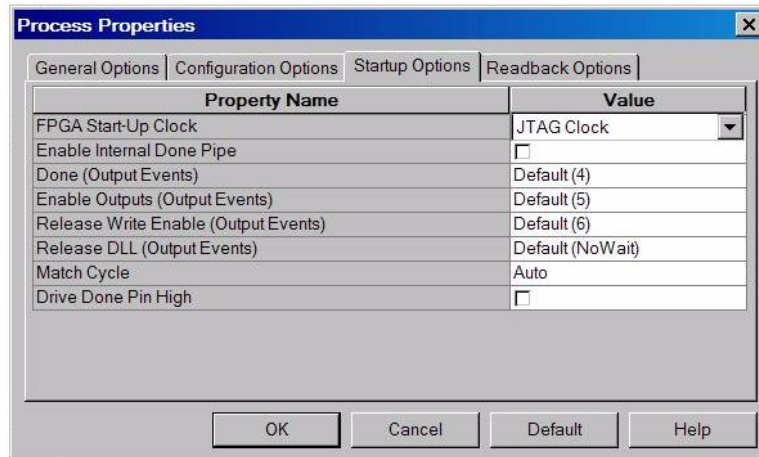
Each sub-process can be activated by double-clicking at the location as shown in Figure 2.11. At the completion of each sub-process, the report can be found by clicking at the '+' next the the name of the sub-process. The information recorded in these files are the similar to those information shown in the bottom window of the **Project Navigator** during the compilation process. However, the information has been saved in files for easy viewing. You are required to observed the report carefully and making sure that the synthesis process is carried out correctly. There are some warning that can be ignored. However, there are some that absolutely needed your attention. One such example is that you want a flip-flop for your design, but latch was synthesized!



**Figure 2.11: Compilation Process**

Assuming that the Synthesis and Implementation phases have been carried out with any error. Before you started the last sub-process, a property change is required. *Left* click on **Generate**

**Programming File** then right click to select the **Property** menu. Under the **Startup Options** menu, change the **CCLK** to **JTAG Clock**. Then, click OK.

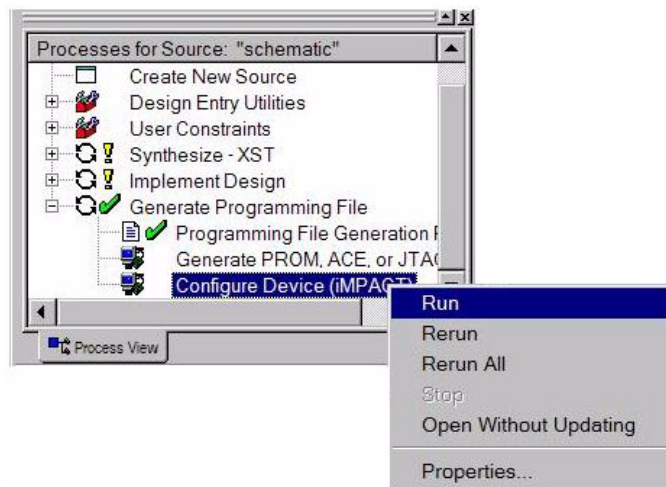


**Figure 2.12: Generate Programming File Properties**

Now, double-click  to start the **Generate Programming File** process.

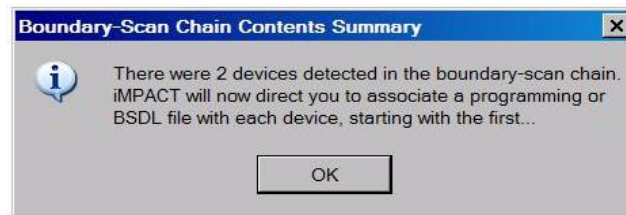
## Programming (Device Configuration) Phase

To download a design to the targeted reconfigurable hardware (in this case the S3Kit) first from the **Processes for Source** expands the Generate Programming File function, right click on **Configure Device (IMPACT)** and select **Run**.

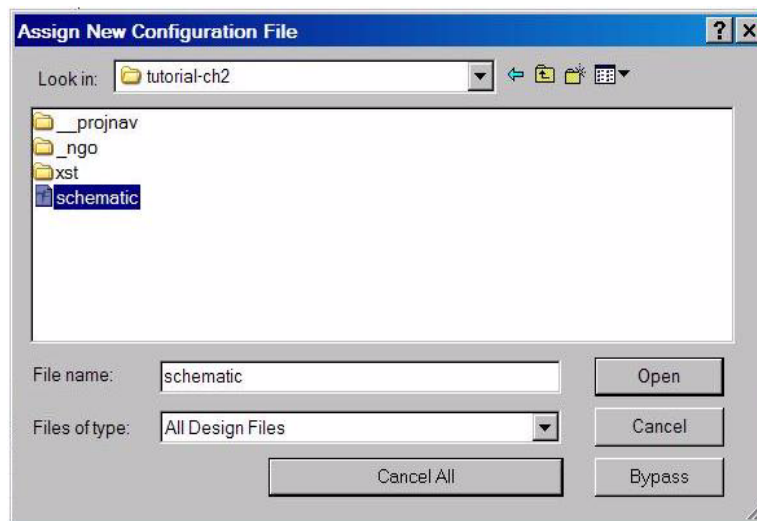


**Figure 2.13: Configure Device**

The default setup should be just fine. The flow is as the following: *Boundary-Scan Mode*  
→ *Automatically connect to cable and identify Boundary-Scan chain.* Once the chain has been identified, the following message appears.

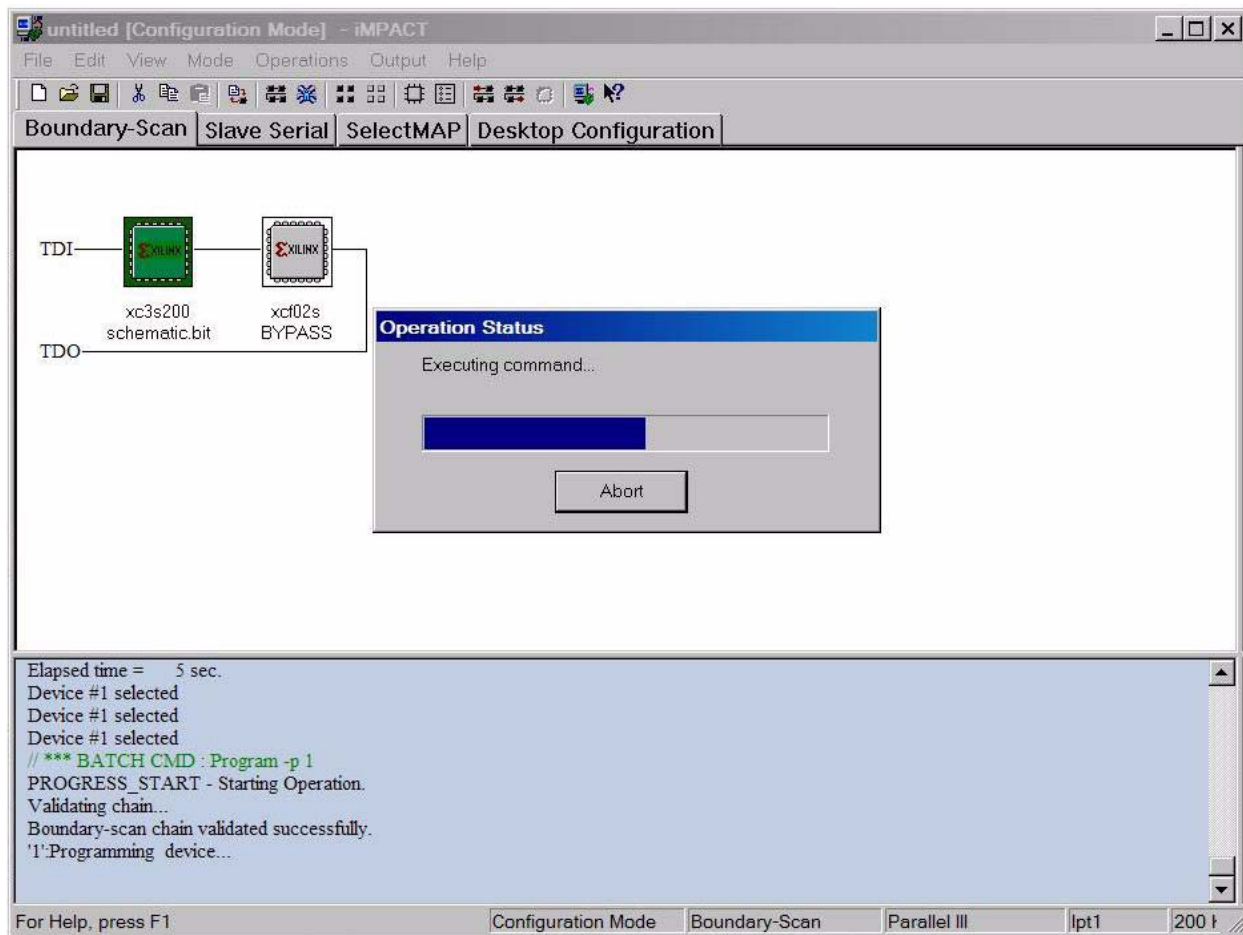


Click OK. Now, a programming file is to be selected.



Click Open. The following message appears. Click **Bypass** on the second selection window.

To download the design into the S3Kit simply right click on *chip* (xc3s200) and select **Program**. No changes are required under Program Options. Click **OK** to program the FPGA.



**Figure 2.14: Programming the FPGA**

The design should be configured and functioning as intended. Thus for the four bit counter example one should see the four lower order S3Kit LEDs counting through the binary sequence.

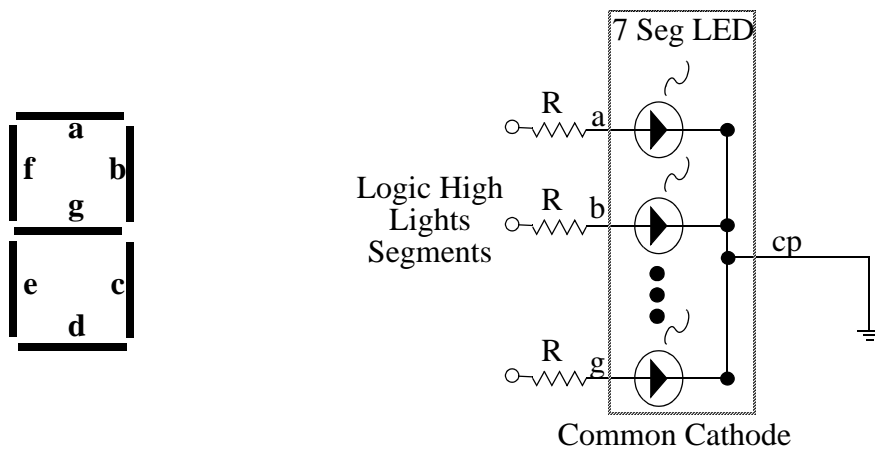
## Chapter 3: Hardware Description Language Design Example

### Example

In this chapter, a binary to seven segment display converter example will be used to illustrate how a design can be entered using VHDL [6], synthesized, simulated, and implemented on S3Kit. The design will is to drive an external seven-segment common anode LED as shown in Figure 1.1. In a similar manner, the inputs to this design will be driven directly by the first four DIP switches (labeled SW0 -- SW3).

### Background

A single seven-segment indicator can be used to display the digits '0' through '9' and the hexadecimal symbols 'A' through 'F' (with symbols 'b', and 'd' being displayed in lower case) by lighting up the appropriate set of segments. For example, the number '8' can be displayed by illuminating all seven segments and the lower case letter 'b' can be displayed by illuminating the segments labeled c,d,e,f, and g for the seven segment display element that is shown in Figure 3.1.:

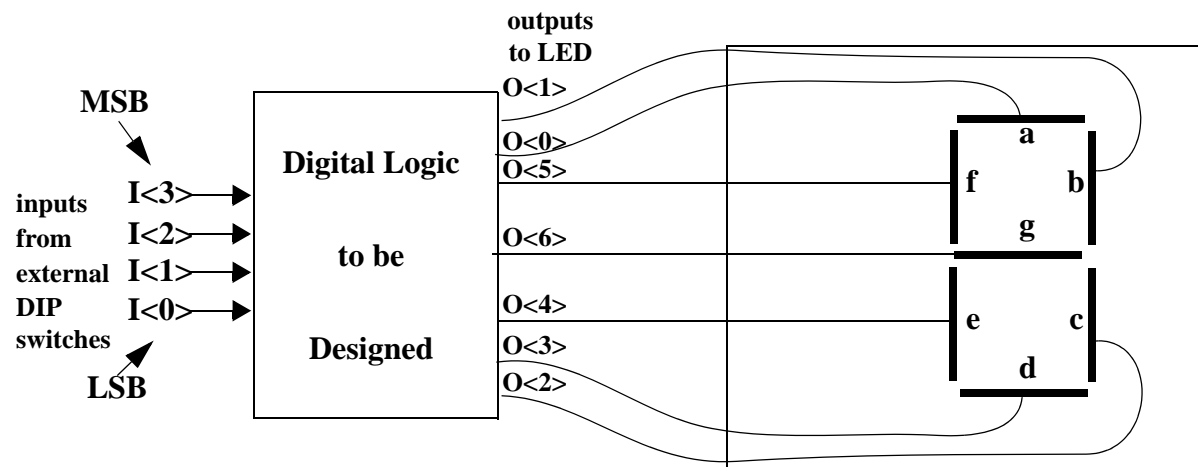


**Figure 3.1: Seven Segment Display Unit**

One common type of seven segment display unit utilizes Light Emitting Diodes, LEDs, as the display elements. In this arrangement, each segment that makes up the seven segment display unit is a separate light emitting diode (LED) that will light up when it is forward biased. Often commercially available seven-segment LED display units minimize the number of external pins needed by internally connecting together one node of each of the seven individual LEDs. In one arrangement, the *common cathode*, the cathodes of the diodes have a common connection point. If

this common point is connected to ground and a set of current limiting resistors are connected in series with the individual segments then each segment of the display can be independently illuminated by placing a logic high on the corresponding segment lead (assuming the logic device is capable of sourcing enough current). The binary to seven segment display example assumes that an external common cathode seven segment LED will be used as the targeted display element with the common cathode point being connected to ground. Thus a logic high will be required to light up each segment.

Figure 3.2 shows a block diagram of the display converter circuit that is to be designed. The display converter circuit is to contain the logic necessary to drive the seven segment display in a manner in which the hexadecimal symbol associated with the four bit input is displayed. Thus the symbol 0 would be displayed if all of the input bits were logic low, and the symbol 8 would be displayed if bit I<3> was high and the rest low. Table 3.1 shows the desired display configuration for each of the 16 possible input scenarios.



**Figure 3.2: Display Converter Seven Segment Display System Overview**

**Table 3.1: Desired LED Display Configurations**

Inputs				Display Configuration	Inputs				Display Configuration
I3	I2	I1	I0		I3	I2	I1	I0	
0	0	0	0	0	1	0	0	0	8
0	0	0	1	1	1	0	0	1	9
0	0	1	0	2	1	0	1	0	A
0	0	1	1	3	1	0	1	1	b
0	1	0	0	4	1	1	0	0	c
0	1	0	1	5	1	1	0	1	d
0	1	1	0	6	1	1	1	0	E
0	1	1	1	7	1	1	1	1	F

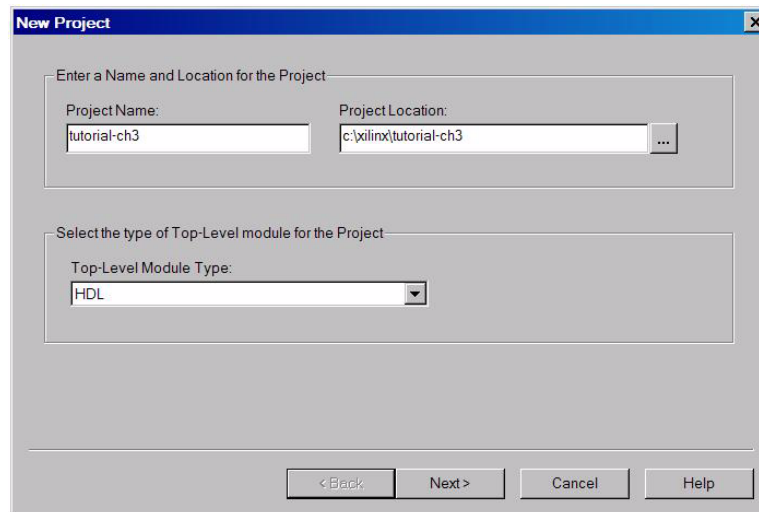
## Design Entry

HDLs are textural representations that are used to model the structure and/or behavior of the system hardware. They are analogous in some ways to high-level software languages such as C, FORTRAN or C++ with the important distinction that HDLs have special constructs specifically designed to model the characteristics of digital hardware. The major difference in HDLs and high-level software languages are that HDL's can easily model the timing attributes and the highly concurrent aspects of digital hardware (i.e. in physical hardware many events often happen at the same time). In addition HDL's have the power to fully describe a logic system using both behavioral and structural design techniques.

To enter the binary to hexadecimal converter example using an HDL, first invoke the **Xilinx Project Navigator** window in the same manner as was done previously, by double clicking on the **Project Navigator** icon.

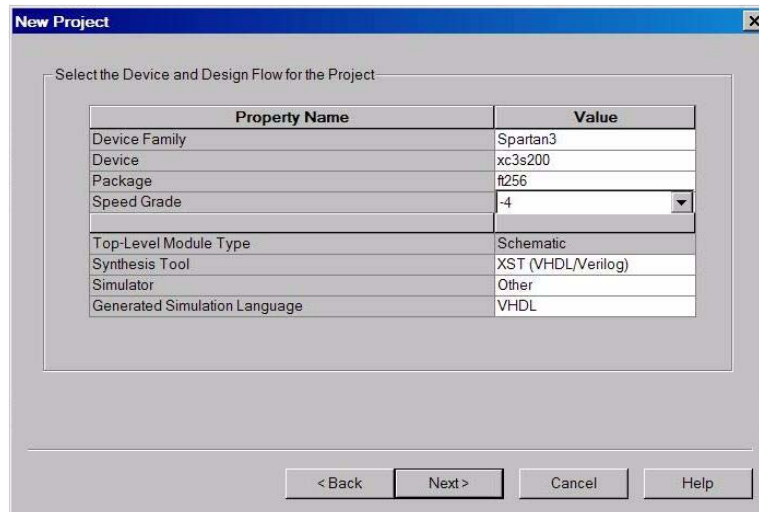


As with the previous example that was presented in Chapter 2, the **Project Navigator** will try to display the previously selected project. Under **File** pull-down menu, select **New Project**. Select this option and enter according to Figure as shown in Figure 3.3. Then click the **Next** button.



**Figure 3.3: New Project Window**

At this point a **New Project** window will appear as shown in Figure 3.4. Make sure that the **Flow** field for this example is set to HDL, since we are demonstrating in this example design entry made through the use of the hardware description language, VHDL. Also, make sure that the device selection and parameters are correct. Then click on the **Next** button. We will not be adding source at this time, so click on the **Next** button on the following two windows.



The 'New Project' dialog box contains a table for selecting device and design flow properties. The table has two columns: 'Property Name' and 'Value'.

Property Name	Value
Device Family	Spartan3
Device	xc3s200
Package	ft256
Speed Grade	-4
Top-Level Module Type	Schematic
Synthesis Tool	XST (VHDL/Verilog)
Simulator	Other
Generated Simulation Language	VHDL

At the bottom of the dialog are four buttons: '< Back', 'Next >', 'Cancel', and 'Help'.

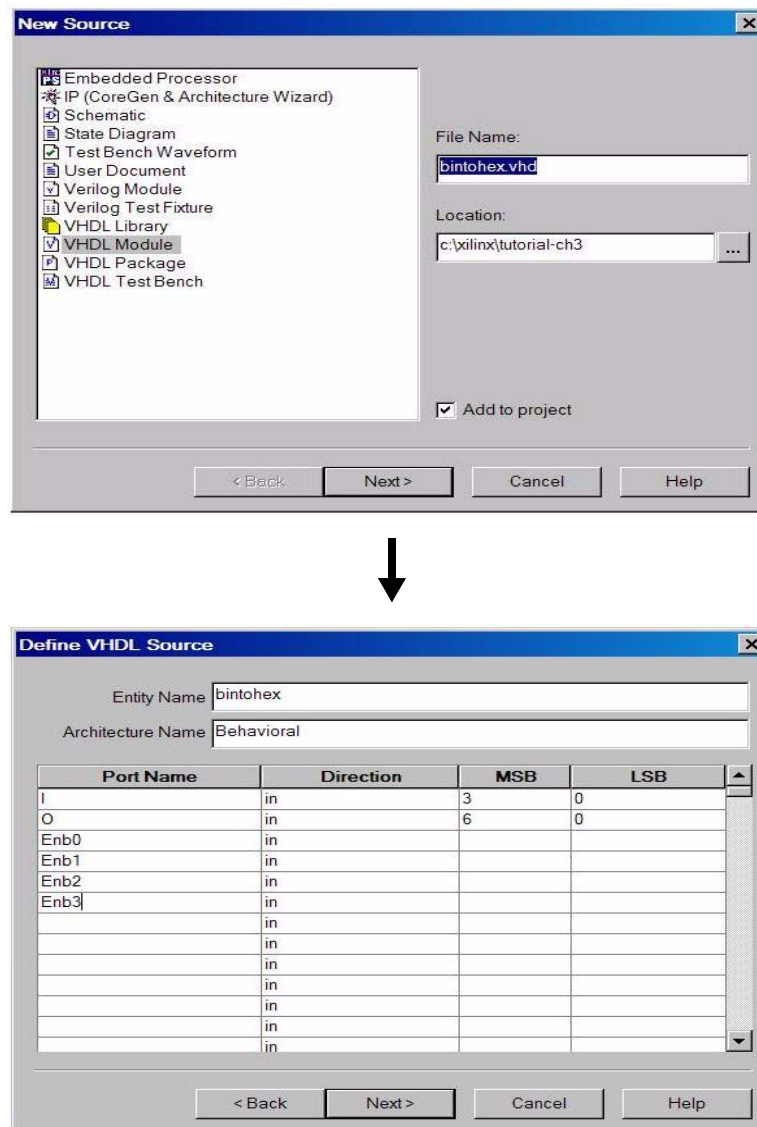
**Figure 3.4: New Project Window**

Once the project has been successfully created, we will utilize the VHDL basic module creation tool to provide us with a VHDL template. Use the **Project** pull-down menu and select **New Source**. Figure 3.5 shows the steps.

For this example, we want to have a four bit input port which we will call **I**. The individual signals of the input port are to be named  $I<3>$ ,  $I<2>$ ,  $I<1>$ ,  $I<0>$ , respectively with  $I<3>$  acting as the most significant bit. To do this, first enter **I** in the **Port Name** field. Then set the range of the **Bus** field (using the up and down arrows) to be 3:0. After making sure the **Direction** of the bus is an **Input** (its default). The output bus of the hexadecimal converter example should be seven bits wide (one for each segment of the LED). We will call this bus **O**, with the individual signals which make up the bus being named  $O<6>$ ,  $O<5>$  ...  $O<1>$ ,  $O<0>$ , respectively (with  $O<6>$  acting as the most significant bit, i.e. the signal to be connected to segment 'g' of the LED and  $O<0>$  acting as the least significant bit, i.e. the signal which is to be connected to segment 'a' of the LED). This can be accomplished by entering **O** in the **Port Name** field, setting the range to 6:0 in the **Bus** field, selecting **Output** in the **Direction** field and then pressing the **Next** button.

After completing the interfacing details, the **VHDL template** window (Figure 3.6) will appear and will contain the skeleton code for the VHDL model. VHDL files contain two main sections, the *Entity* and the *Architecture*. The *Entity* section was directly created by the wizard where it describes the interface between the design being modeled and the outside world. The wizard also automatically created a portion of the *Architecture* section which describes how the

design is to function. It is now up to the user to supply the additional VHDL statements needed to fully describe the implementation. The VHDL template file which was created for the binary to hexadecimal example is shown in Figure 3.7.



**Figure 3.5: New VHDL Source**

```

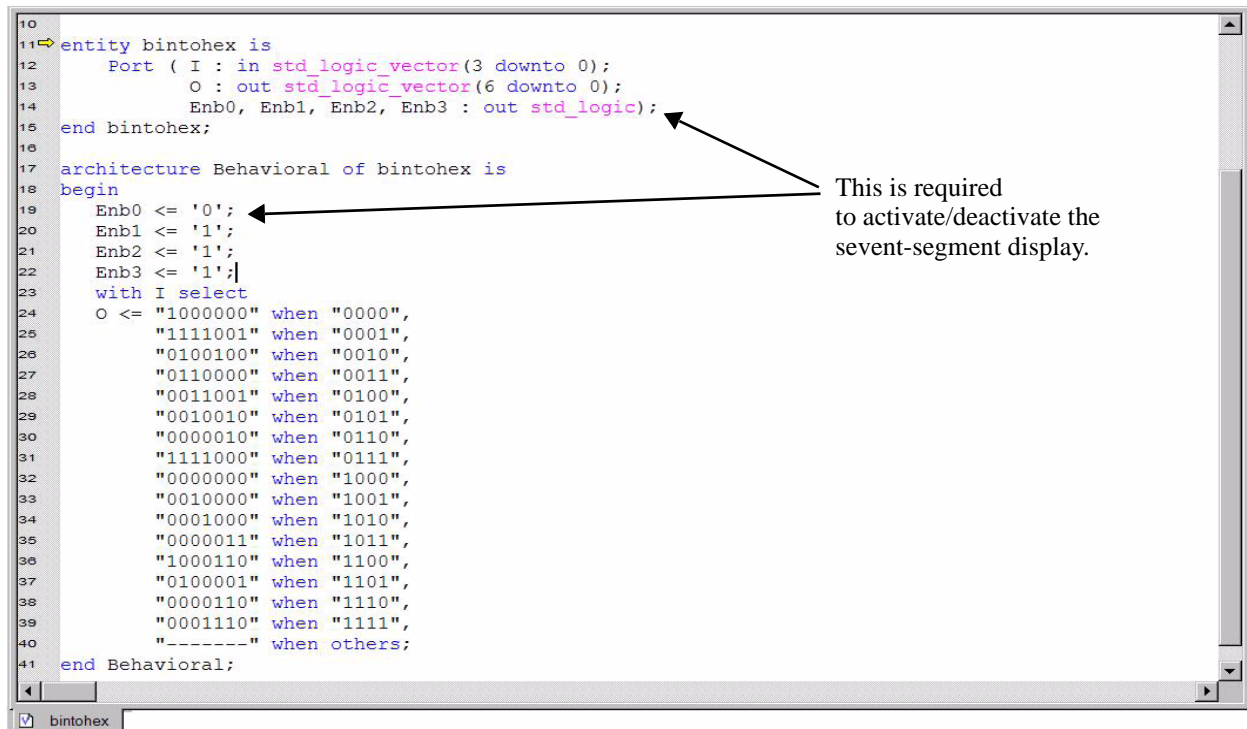
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  -- Uncomment the following lines to use the declarations that are
7  -- provided for instantiating Xilinx primitive components.
8  --library UNISIM;
9  --use UNISIM.VComponents.all;
10
11 entity bintoheX is
12     Port ( I : in std_logic_vector(3 downto 0);
13           O : in std_logic_vector(6 downto 0);
14           Enb0 : in std_logic;
15           Enb1 : in std_logic;
16           Enb2 : in std_logic;
17           Enb3 : in std_logic);
18 end bintoheX;
19
20 architecture Behavioral of bintoheX is
21
22 begin
23
24
25 end Behavioral;
26

```

**Figure 3.6: VHDL Template**

The structure of the VHDL file can be described as follows. Double dashes, “--”, are used to introduce comments and semicolons are used to terminate the statements. The *Entity* section that was produced by the wizard is incomplete, refer to Figure 3.7 for modifications. It begins with the VHDL keyword **entity** followed by the user defined name of the logic design (in this case *bintoheX*). The entity name is then followed by the keywords **is port** that designates that a list of input/output ports is to follow. In this case, we have two bus signals that are defined. One is named *I* (for input) which is a four member input vector (direction specified by the *in* keyword),  $I<3> \text{ -- } I<0>$ , with  $I<3>$  acting as the most significant bit (this is due to the *downto* keyword). The other is a seven member output vector (direction specified by the *out* keyword) which is named *O*, which has seven members  $O<6> \text{ -- } O<0>$ . [Note: The `STD_LOGIC_VECTOR` defines the vector type. It is declared in the library `IEEE.std_logic_1164.all`. It is possible for the user to declare arbitrary data types in VHDL. The data types defined in this library are standardized allowing for

increased portability among VHDL simulators and synthesizers.] The entity section ends with the **end** statement followed by the design name.



**Figure 3.7: VHDL Template for binto hex Example**

The desired behavior of the *Entity* section is modeled in the *Architecture* section. This section begins with the keyword **architecture** which is followed by an arbitrarily defined name for the architecture (in this case the wizard choose to use, *binto hex\_arch*) which is paired with the identity of the *Entity* using the **of** keyword followed by the *Entity* name (i.e. in this case *of binto hex*). The **begin** keyword is used to separate the *declarative* part of the model from the *statement* part of the *Architecture* section. The *architecture declarative* part is used to make declarations for such items type, signals, and components. For the model presented in Figure 3.6, no declarations are needed to represent the design so this part is not entered. The VHDL modeling statements are to be placed in the *architecture statement* area of the model which is located between the **begin** keyword and the **end** keyword.

Figure 3.7 shows the final VHDL model which was created for the binary to hexadecimal converter example by adding additional comments and an additional architecture statement to the original template file that was created by the wizard. Comments were added as necessary throughout the model and a **with... select...when** statement was added to the *architecture* section. The

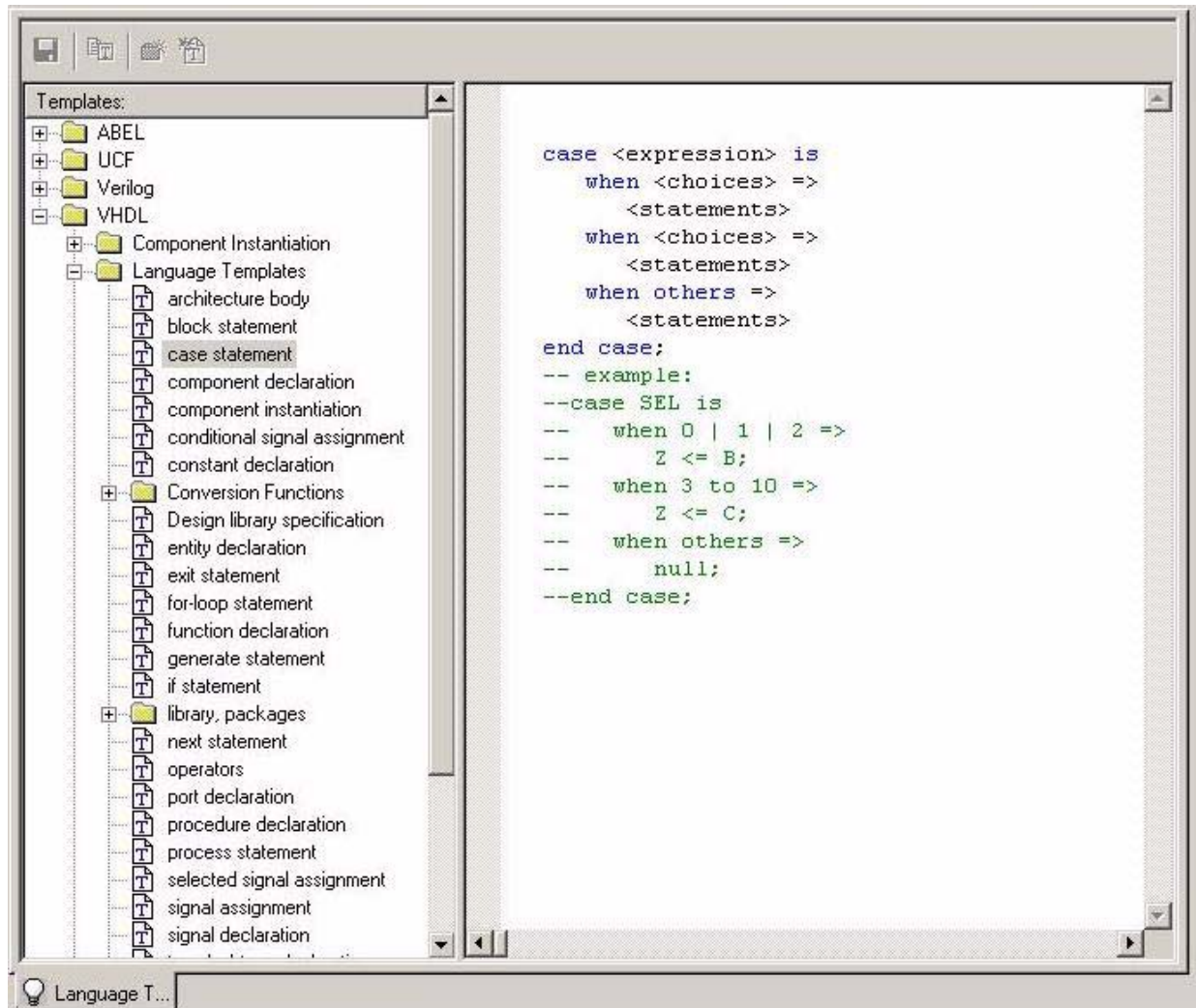
**with... select...when** statement is analogous in many ways to a **case** statement in the C programming language in that provides selective signal assignments. It has the following structure:

```
with input_signal select
output_signal <= value_a when value_1,
                    value_b when value_2,
                    .
                    .
                    .
                    value_x when last value,
                    value_z when others;
```


When the input signal equals *value\_1* then the *output\_signal* is set to *value\_a*. When the input\_signal equals *value\_2* the *output\_signal* will be set to *value\_b* and so on. In this example, this construct is being used to implement the truth table that describes the desired binary to hexadecimal converter operation (but unlike most truth tables the inputs appear on the right side). The input\_signal and output\_signal are both vectors that are defined within the library package IEEE.std\_logic\_1164.all. In VHDL, the logic values that are support include '0' for logic low (forcing 0), '1' for logic high (forcing 1), and '-' for don't care.

[Note: This illustrates manual editing of the VHDL file without any outside assistance. For more complex designs one might want to utilize the built-in **Language Template Tool**. To use the **Language Template Tool**, from the pull-down menu, select **Edit**, then click on the **Language Template** option (second from bottom of the list). The **Language Template** window as shown in Figure 3.8 will appear. At the touch of the mouse button additional information will be presented

and constructs for large sections of the design can automatically be entered into the design. This is a very powerful tool, the detailed description of which is beyond the scope of this manual.]]



**Figure 3.8: Language Template Window (VHDL)**

After the VHDL model has been entered, one needs to check the syntactical correctness of the modeling code. This can be done from the **Project Navigator** by double clicking on  **Synthesis - XST**. Assuming that no syntax errors exist in the above HDL code, we will move on to the next step. Once again, ModelSim will be utilized as functional simulation too and this will be presented in another tutorial.

## User Constraints Entry Phase

In this design example, it will be necessary to assign (lock) the four logical inputs of the binary to hexadecimal design to the set of DIP switches (SW1, SW2, SW3, and SW4) located on S3Kit and assign the seven outputs to the jumper locations which will be used to drive the individual segments of the externally connected common anode seven segment LED display.

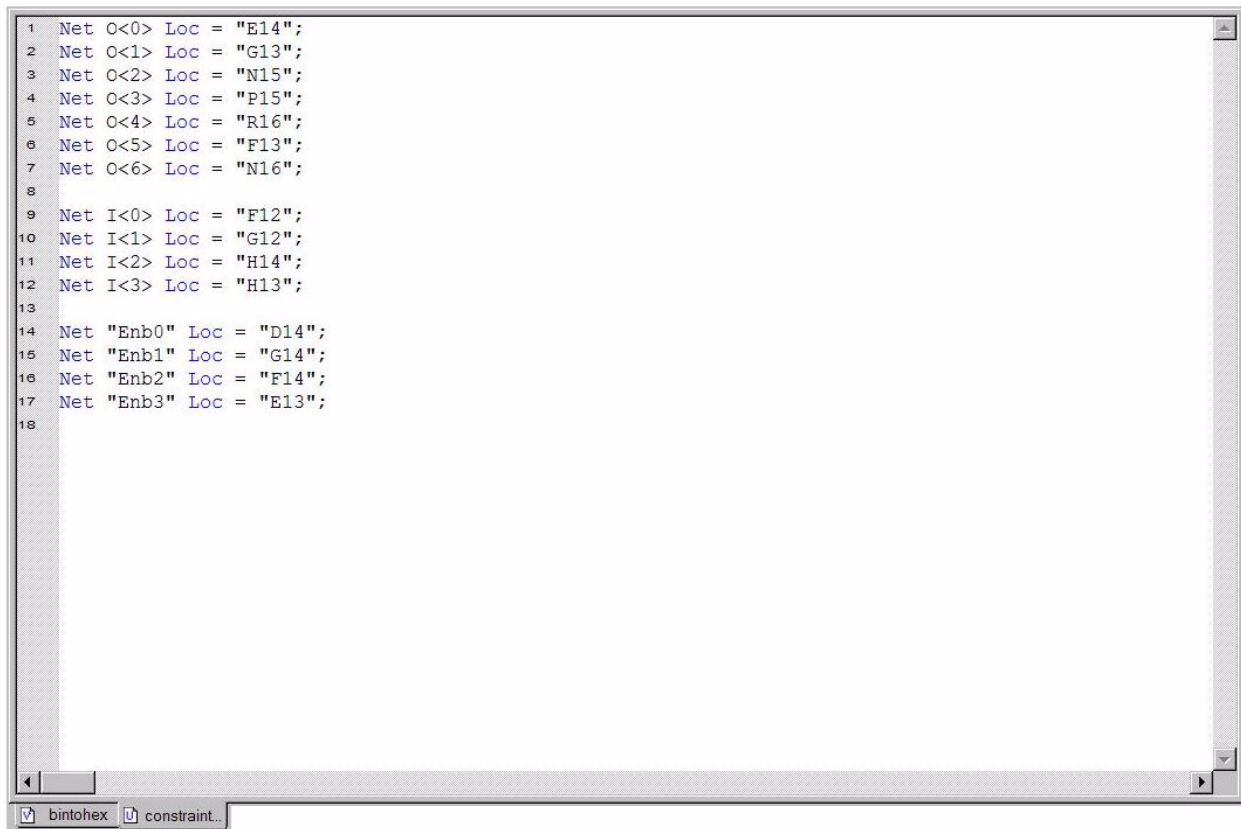
For this design the desired schematic name to Xilinx XC3S200 Pin name cross reference (i.e. Pin Locks) is shown in Table 3.2.

**Table 3.2: Desired Pin Locking (Cross Reference) Configuration**

I/O Pin Description	VHDL “Net” Name	Xilinx XC3S200 Pin Name
DIP Switch SW0	I<0>	F12
DIP Switch SW1	I<1>	G12
DIP Switch SW2	I<2>	H14
DIP Switch SW3	I<3>	H14
Segment ‘a’ of Digit #0	O<0>	E14
Segment ‘b’ of Digit #0	O<1>	G13
Segment ‘c’ of Digit #0	O<2>	N15
Segment ‘d’ of Digit #0	O<3>	P15
Segment ‘e’ of Digit #0	O<4>	R16
Segment ‘f’ of Digit #0	O<5>	F13
Segment ‘g’ of Digit #0	O<6>	N16
Enable/Disable LED #0	Enb0	D14
Enable/Disable LED #1	Enb1	G14
Enable/Disable LED #2	Enb2	F14
Enable/Disable LED #3	Enb3	E13

The resulting additions to the **User Constraints File** are shown in Figure 3.9 where the cross reference described in Table 3.2. After these additions have been made to the **User Constraints File** one should **Save** the work, and return to the **Project Manager** window.





The image shows a screenshot of a text editor window. The window has a title bar with two tabs: 'binto hex' and 'constraint...'. The 'constraint...' tab is active. The editor contains a list of pin locking constraints, each on a new line. The lines are numbered 1 through 18 on the left margin. The constraints are as follows:

```
1 Net O<0> Loc = "E14";
2 Net O<1> Loc = "G13";
3 Net O<2> Loc = "N15";
4 Net O<3> Loc = "P15";
5 Net O<4> Loc = "R16";
6 Net O<5> Loc = "F13";
7 Net O<6> Loc = "N16";
8
9 Net I<0> Loc = "F12";
10 Net I<1> Loc = "G12";
11 Net I<2> Loc = "H14";
12 Net I<3> Loc = "H13";
13
14 Net "Enb0" Loc = "D14";
15 Net "Enb1" Loc = "G14";
16 Net "Enb2" Loc = "F14";
17 Net "Enb3" Loc = "E13";
18
```

**Figure 3.9: Adding Pin Locking Information to the Project Constraint File**

## **Synthesis/Implementation/Generate Programming File Phases**

This section is similar to the one presented in Chapter 2, readers are referred to Chapter 2 for Synthesis/Implementation/Generate Programming File Phases.

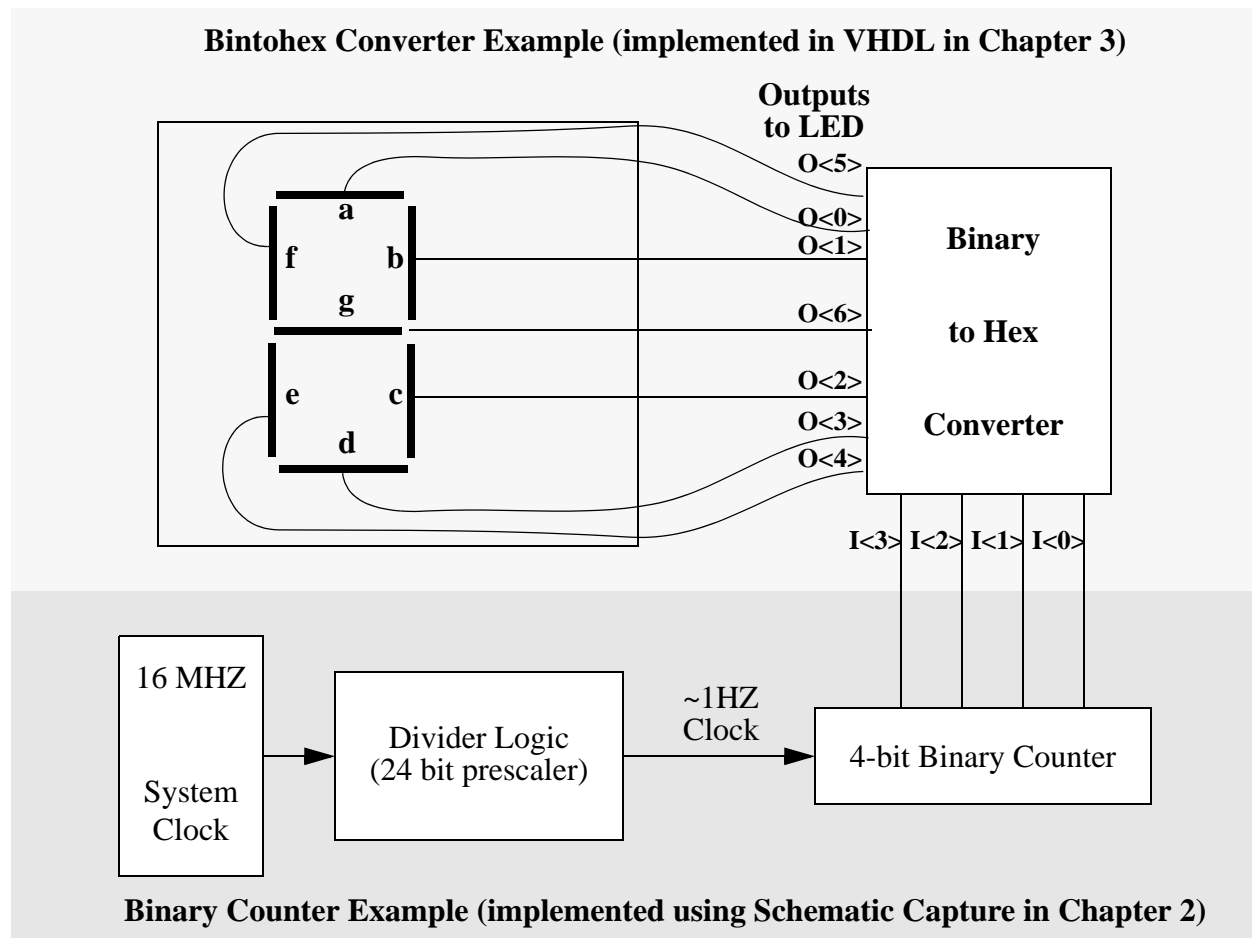
### **Programming (Configuration) Phase**

This phase is identical to that described in Chapter 2. Please refer to this material to configure the S3Kit for the binary to hexadecimal converter design. When the design has been configured the external LED display should display the hexadecimal symbol which represents the current state of the four external DIP switches.

## Chapter 4: Combined Schematic Capture/HDL Design Example

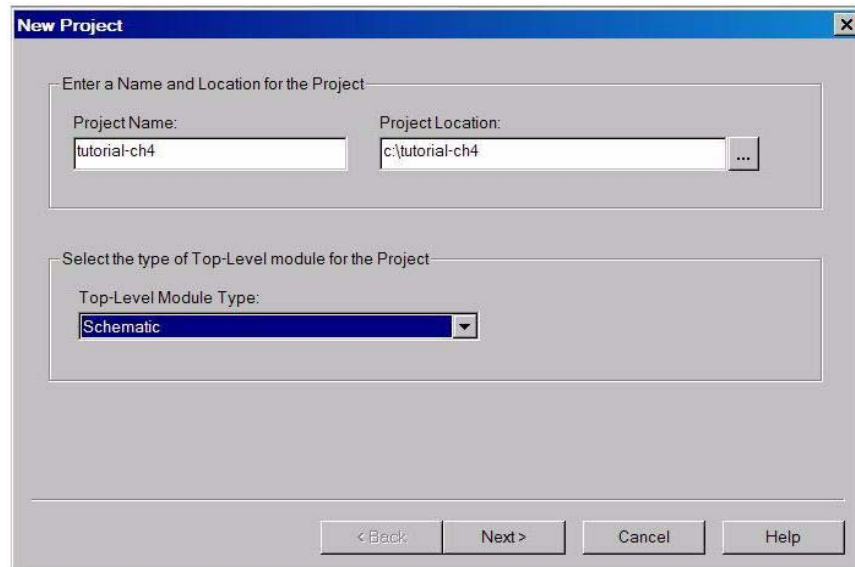
### Example

In this chapter, the two previous design examples from Chapter 2 (the binary counter) and Chapter 3 (the binary to seven segment hexadecimal converter) will be combined to form a complete hexadecimal counter design. The final design will result in a new hexadecimal symbol being displayed on the external seven segment LED after each master clock pulse of ~1HZ. The hexadecimal counter will be constructed by connecting the outputs of the binary counter design directly to the inputs to the binary to seven segment hexadecimal converter as shown in Figure 4.1. As in the binary counter example of Chapter 2, the clocking signal will originate from an internal 50 MHZ oscillator which will be directed through a 24 bit prescaler network. To minimize the design effort and illustrate the mechanics of hybrid design methodology, this design is to be entered using both schematic capture and HDL methodology and constructs.



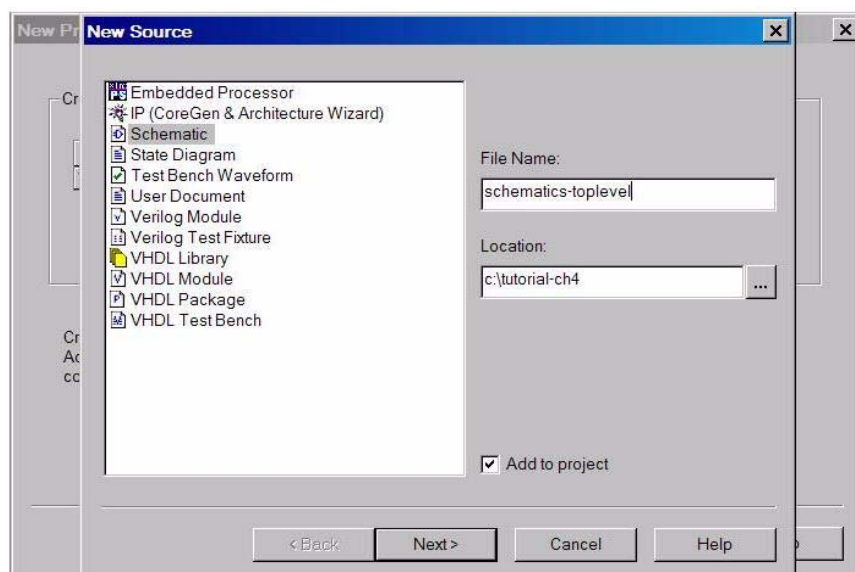
## Setting up the Project

As in the previous cases, to enter this design, first start the **Project Navigator** of Xilinx Foundation Series 6.3i by double clicking on the **Project Navigator**. For this case, we would to Create a New Project with **Schematic** as the **Top-Level Module Type**. Select this option as shown in Figure 4.2. Then click on the **Next** button.



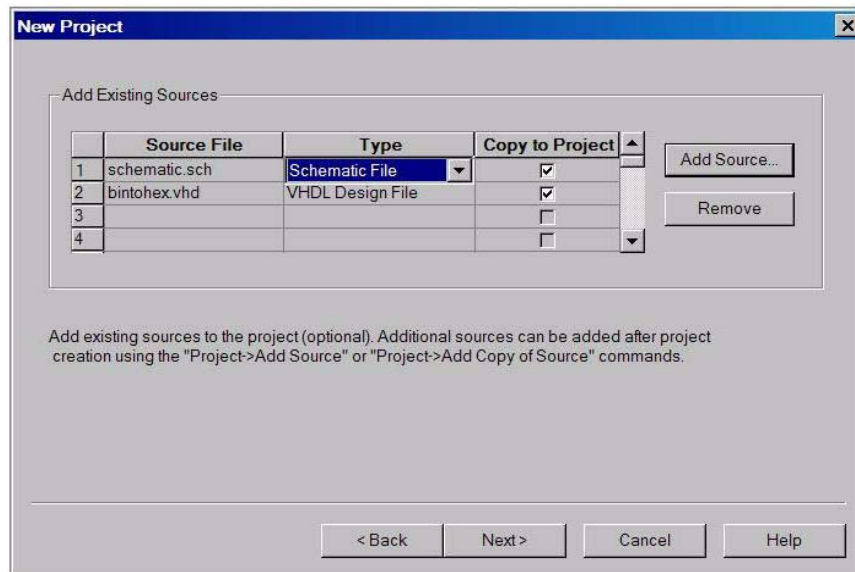
**Figure 4.2: New Project Setup**

A blank schematic file will be added to be the top-level for this hybrid project (Figure 4.3).



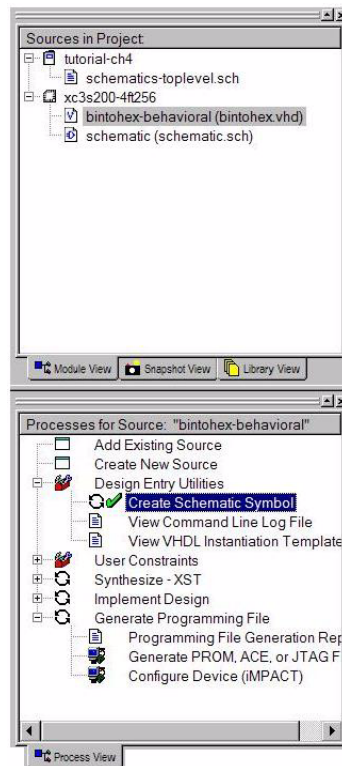
**Figure 4.3: New Schematic File As Top Level**

The first step in entering the hybrid hexadecimal counter example will be to enter or copy the logical schematic elements used to create the bcount (binary counter) example of Chapter 2 into the schematic associated with this design. Then re-enter the binary counter design (but without the input/output buffers). The schematic file and the binto hex VHDL module created in Chapter 2 and Chapter 3, respectively, are added as shown in Figure 4.4.



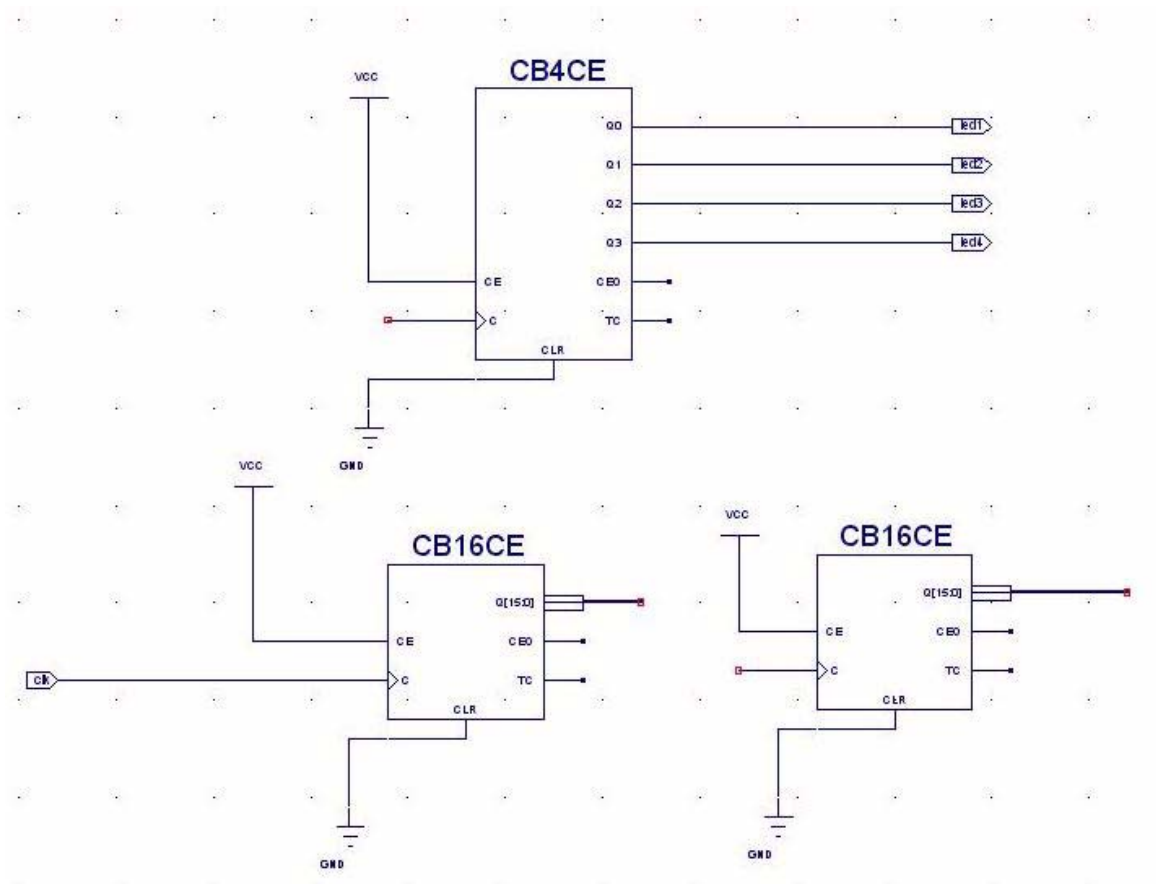
**Figure 4.4: Add previously created sources to this project**

The next step is to convert the binary to hexadecimal converter VHDL model developed in Chapter 3 into a macro which will appear as a symbol in the schematic. To do this, return to the **Project Navigator** window, select the binto hex-behavior and click on the '+' next to the **Design Entry Utilities** within **Processes for Source**. Then, double-click on **Create Schematic Symbol**. These steps are shown in Figure 4.5.



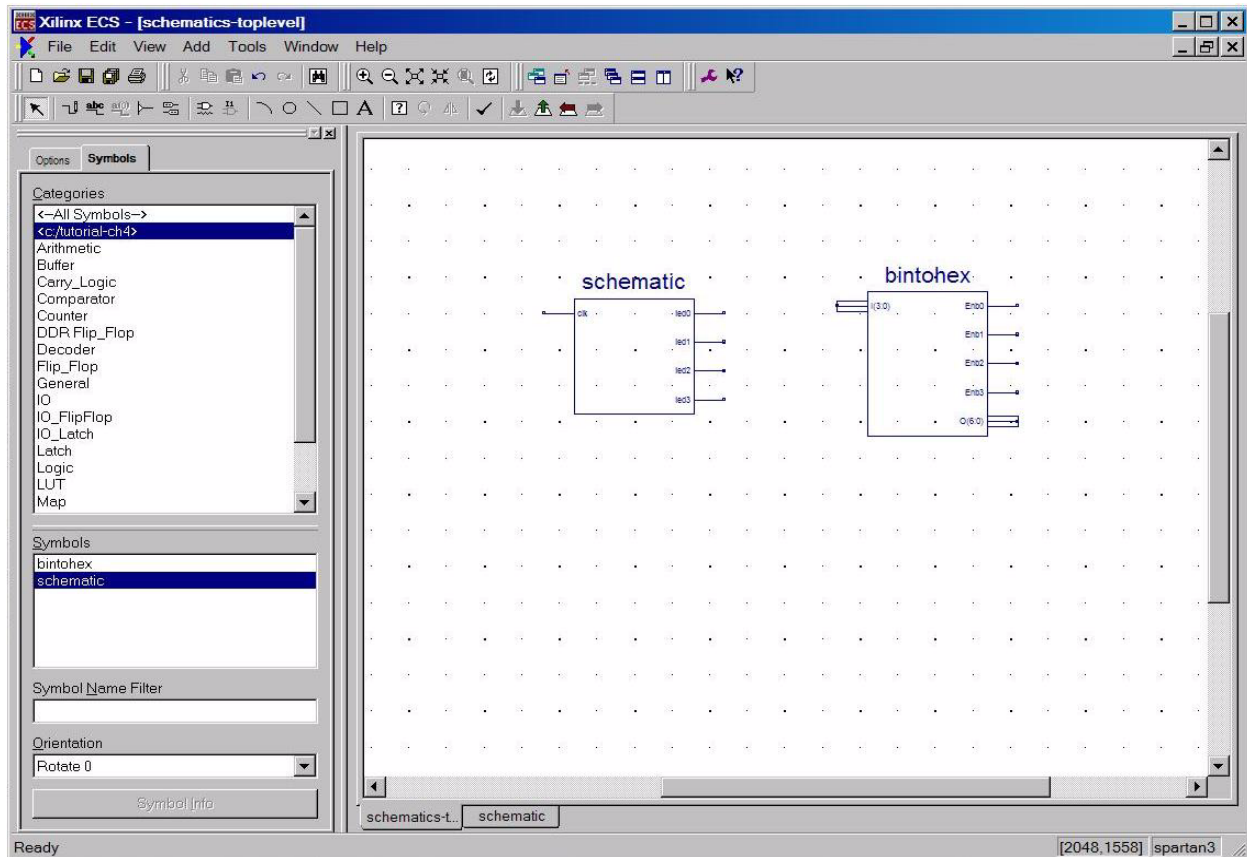
**Figure 4.5: Create a binto-hex symbol**

Once the symbol for the VHDL module has been created, the symbol for the 4-bit counter will also need to be created. In order to do this, we need to modified the schematics.sch from Chapter 2 which has been added to this project. As shown in Figure 4.6, the buffers for sch.sch have been removed. When the modifications have been made, save the schematic file and create a symbol with the steps shown for the HDL module.



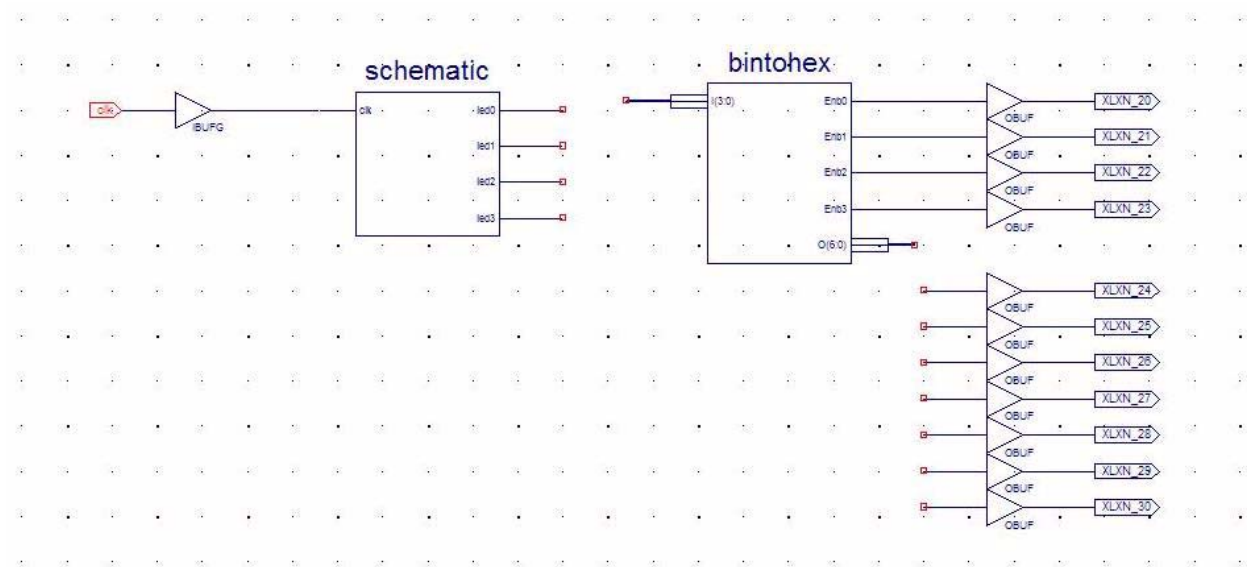
**Figure 4.6: Modify the 4-bit counter (with all the buffers removed)**

With both symbols successfully created, return to **Project Navigator** and enter schematics.sch to add the symbols to schematics.sch. This is shown in Figure 4.7.



**Figure 4.7: Add the created symbols in schematics.sch**

The complete connections for Figure 4.7 is shown in Figure 4.8.



**Figure 4.8: Connections in schematics.sch**



Now the design can be completed using the schematic capture techniques which have already been presented in Chapter 2. The completed schematic with its associated components and net/bus names are shown in Figure 4.8. After the schematic is successfully entered, the design should be **Saved** and the user should return to the **Project Navigator** window.

## User Constraints Entry Phase

In this hybrid HDL/schematic capture hexadecimal counter design example, it will be necessary to assign the **CLK** signal to the S3Kit 50MHz internal clock as was done with the design of Chapter 2 and assign the seven outputs to the jumper locations which will be used to drive the individual seven segments of the LED display as was done with the binary to hexadecimal converter example of Chapter 3. In this case, the binary to hexadecimal converter design is used as a component macro so the schematic capture net names will be used to specify which signals one would like to lock to specific XC3S200 pins.

For this hybrid design, the desired schematic name to Xilinx XC3S200 Pin name cross reference (i.e. Pin Locks) will be as shown in Table 4.1. (Note: in this hybrid design, the outputs of the binary counter design now internally drive the inputs to the binary to hexadecimal converter so only the input to the first design and outputs to the second need to be routed to the Xilinx's XC3S200 I/O pins.)

**Table 4.1: Desired Pin Locking (cross reference) Configuration**

I/O Pin Description	Schematic Net Name	Xilinx XC4010XL Pin Name
50MHz Clock	CLK	T9
Segment 'a' of Digit #0	O<0>	E14
Segment 'b' of Digit #0	O<1>	G13
Segment 'c' of Digit #0	O<2>	N15
Segment 'd' of Digit #0	O<3>	P15
Segment 'e' of Digit #0	O<4>	R16
Segment 'f' of Digit #0	O<5>	F13
Segment 'g' of Digit #0	O<6>	N16
Enable/Disable LED #0	Enb0	D14

**Table 4.1: Desired Pin Locking (cross reference) Configuration**

I/O Pin Description	Schematic Net Name	Xilinx XC4010XL Pin Name
Enable/Disable LED #1	Enb1	G14
Enable/Disable LED #2	Enb2	F14
Enable/Disable LED #3	Enb3	E13

The resulting additions to the **User Constraints File** are shown in Figure 4.9.

```

1 Net "clk" LOC = "T9";
2
3 Net "a" Loc = "E14";
4 Net "b" Loc = "G13";
5 Net "c" Loc = "N15";
6 Net "d" Loc = "P15";
7 Net "e" Loc = "R16";
8 Net "f" Loc = "F13";
9 Net "g" Loc = "N16";|
10
11 Net "Enb0" Loc = "D14";
12 Net "Enb1" Loc = "G14";
13 Net "Enb2" Loc = "F14";
14 Net "Enb3" Loc = "E13";
15

```

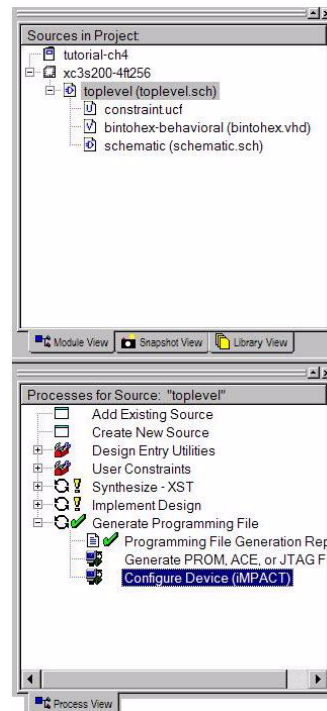
**Figure 4.9: Constraints**

**NET** <schematic net name> LOC=<symbolic Xilinx pin name>

construct. After these additions have been made to the **User Constraints File** one should **Save** the work, **Exit** the **Report Browser**, and return to the **Project Manager** window.

## Synthesis/Implementation/Generate Programming File Phases

This section is similar to the one presented in Chapter 2/3, readers are referred to Chapter 2 or 3 for Synthesis/Implementation/Generate Programming File Phases. The process is once again shown in Figure 4.10.



**Figure 4.10: Top-level compilation**

## Programming (Configuration) Phase

This phase is identical to that described in Chapter 2. Please refer to this material to configure the S3Kit. When the design has been configured the external LED display should successfully and repetitively display the hexadecimal pattern '0' to 'F' in sequence (lowest to highest with wrap around) without requiring any interaction from the user.

## Chapter 5: References

- [1] Spartan-3 Starter Kit Board User Guide, Xilinx, 2004.
- [2] *ISE Quick Start Guide*, Xilinx, 2004.
- [3] *Development System Reference Guide*, Xilinx, 2004.
- [4] *Integrated Software Environment (ISE) Guide*, 2004.
- [5] *Constraints Guide*, Xilinx, 2004.
- [6] *XST User Guide*, Xilinx, 2004.