Getting Started with Xilinx Design Tools and the XS 40 Prototyping Platform-- a User's Guide

by

B. Earl Wells   Sin Ming Loo

Electrical and Computer Engineering Department

University of Alabama in Huntsville

Introduction

The purpose of this manual is to provide additional support to students in CPE 427, Computer Engineering Senior Design I, who will be using the Xilinx Foundation series software to rapidly prototype digital designs on the XS 40 reconfigurable prototyping platform. This manual supplements the existing documentation on the XS 40 [1] and the material on the Xilinx's Foundation 3.1i [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 XC4010XL FPGA on the XS 40.

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

Chapter 1: General information is presented about the XS 40 platform and the Xilinx Foundation 3.1i 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 XS 40 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 XC4010XL 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.

This material is based upon work supported by the National Science Foundation under Grant No. 9751482.
Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the view of the National Science Foundation.

Chapter 1: The XS 40 and Xilinx Foundation 3.1i Computer Aided Design Tool

The XS 40

The XS 40 has many features that facilitate experimentation in hardware/software codesign as well as the general rapid prototyping of digital logic. The rapid prototyping environment that will be used at UAH in CPE 427 is shown below in Figure 1.1.

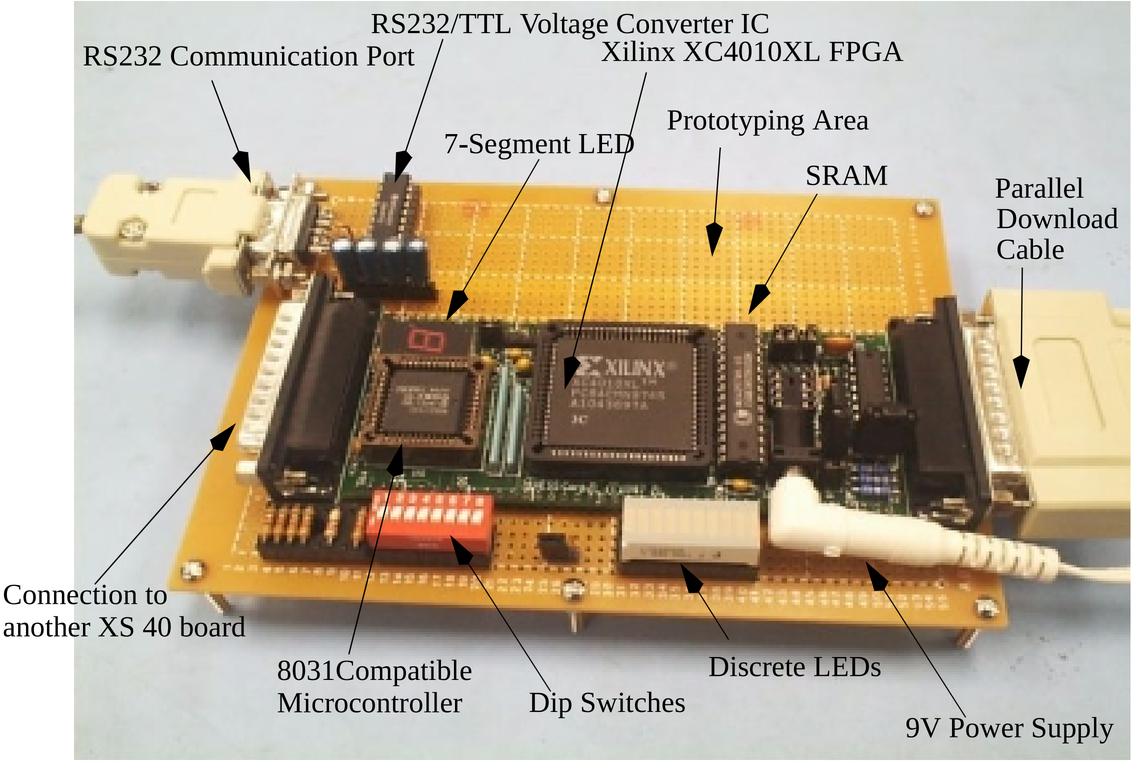


Figure 1.1: The XS 40 Reconfigurable Prototyping Platform.

The major component of this rapid prototyping environment is the XS 40 board (version 1.1). This board is designed and produce by X Engineering Software Systems Corporation (XESS). The XS 40 is a small rapid prototyping board containing both a Xilinx XC4010XL FPGA and an 8031 type microcontroller (a ROM-less version of 8051). The Xilinx XC4010XL FPGA has 400 Configurable Logic Blocks (CLBs, Xilinx's logic cell name) which is equivalent to 1,120 flip-flops. The 8031 type microcontroller is a derivative of Intel's 8051 series of microcontrollers. The board also contains a 12MHz oscillator for use with the microcontroller and user supplied logic which resides within the Xilinx FPGA. Most of the I/O pins are directly connected with the XC4010XL FPGA -- these connections are detailed in Appendix A. There is also 32K of SRAM on board XS 40 for storing 8031's code/data (and other uses).

The XS 40 board communicates with the PC via the parallel port. There are two parallel ports on the XS 40 board. The parallel port on the power supply side of the board is used to download both the bit-stream information to the FPGA and Intel hex format information to the 8031 microcontroller. The parallel port connector on the other end of the XS 40 board is there to allow multiple XS 40 boards to be daisy-chained together.

Figure 1.2 shows the block diagram of XS 40 prototyping environment that will be used in CPE 427 this term. To facilitate the work done in this class external discrete LEDs, a bidirectional RS 232 communication port, and a set of dip switches have been added to the XS 40 using wire wrap technqiues.

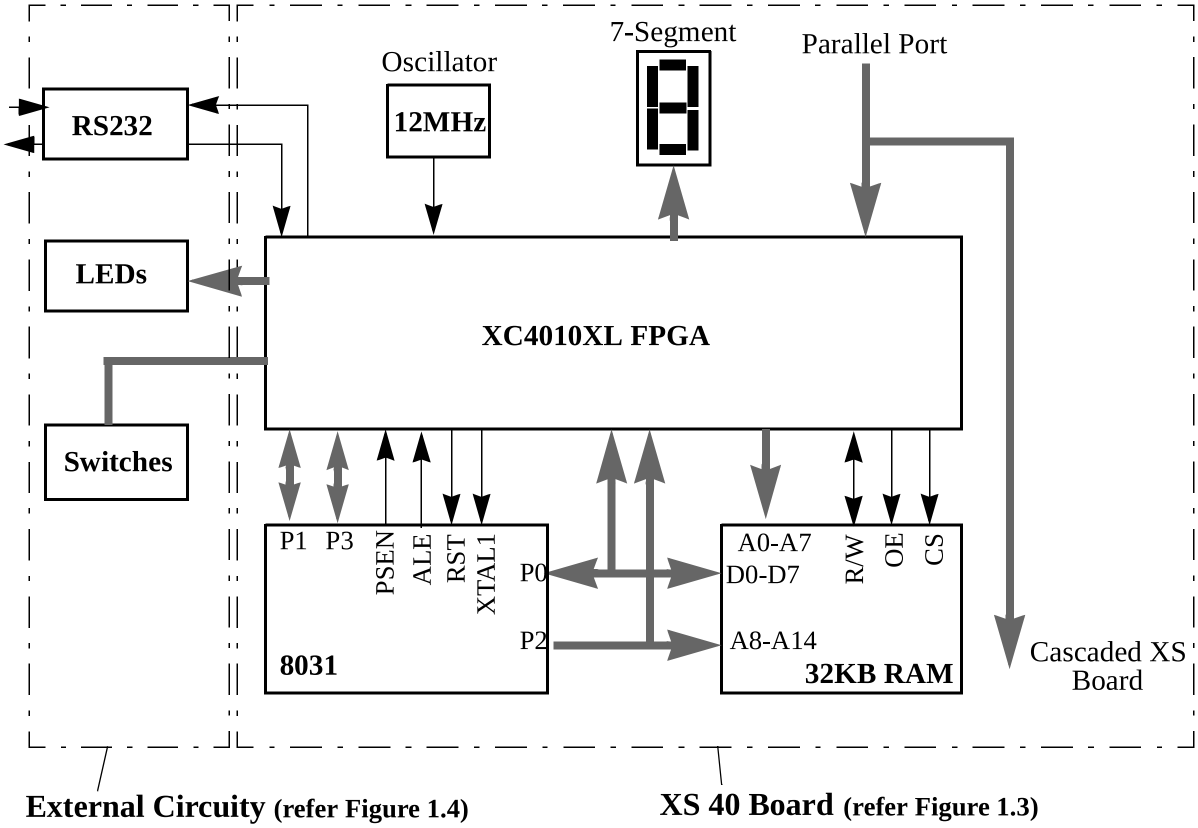


Figure 1.2: Connections among XS 40 Board (8031, RAM, and XC4010XL FPGA), Serial Communication Port (RS232), LED, and Switches.

A complete Schematic of this board and the UAH added external circuity are shown in Figures 1.3 and 1.4, repectively.

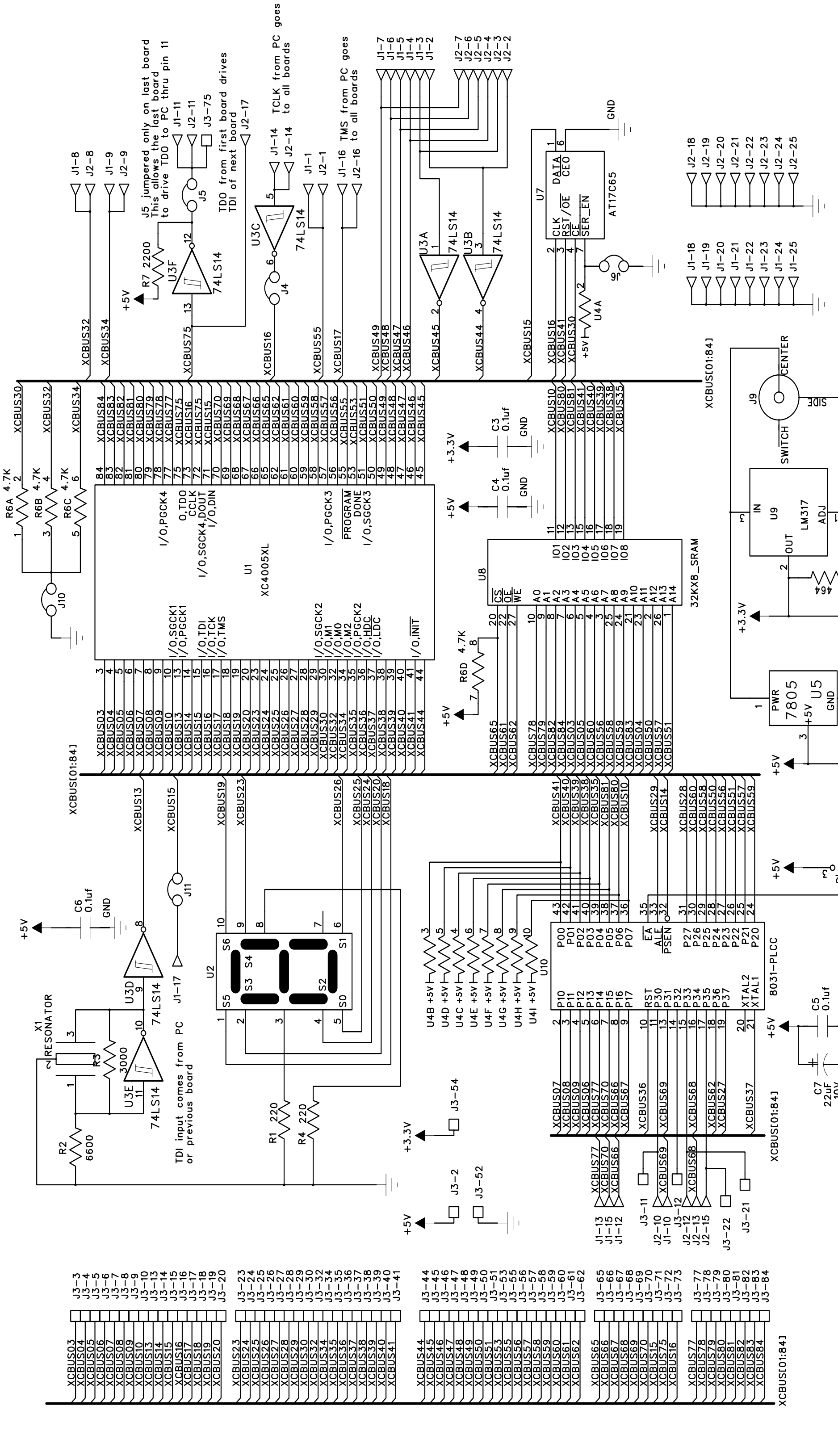
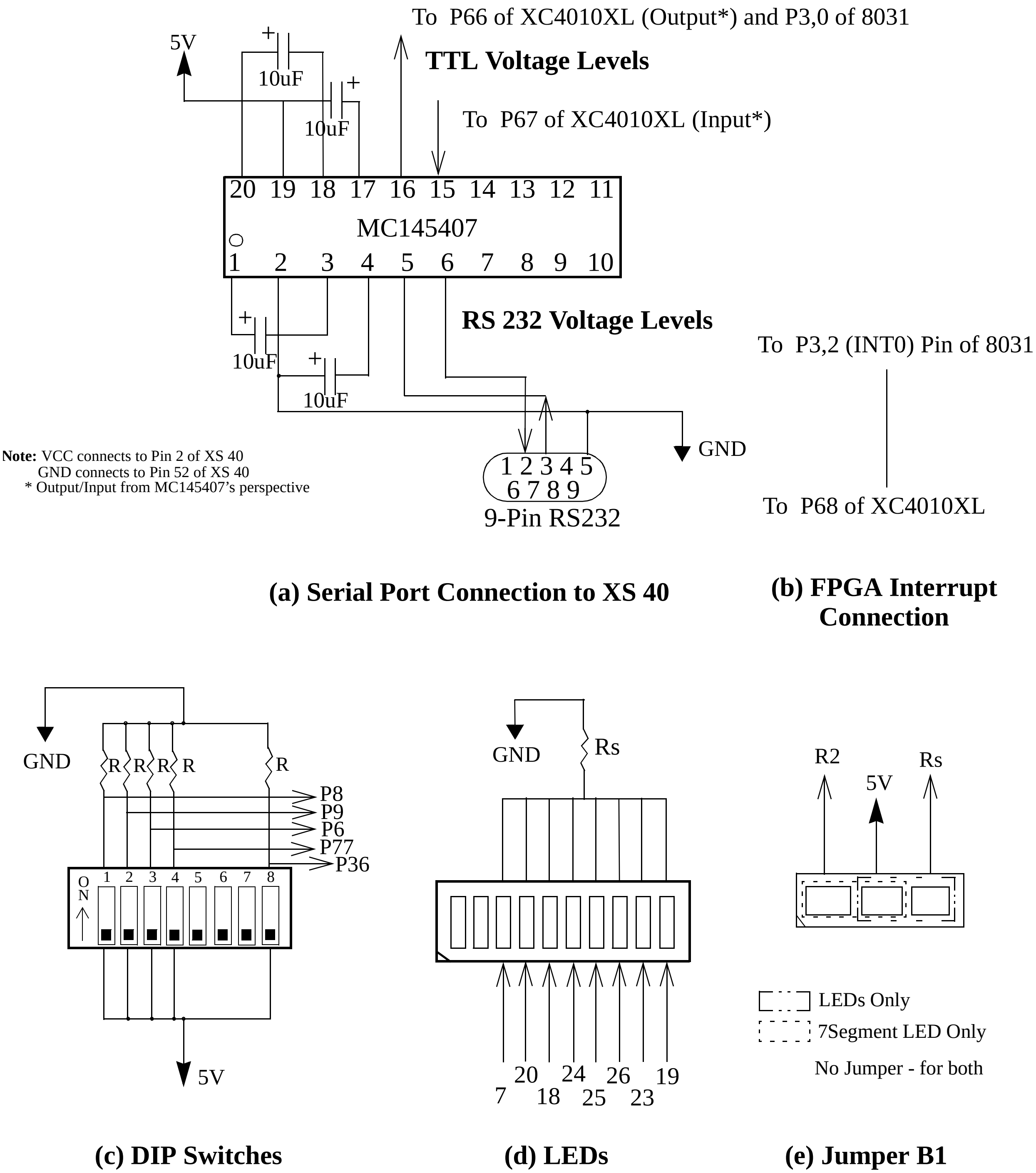


Figure 1.3: Schematic of the XS 40 Board



(a) Serial Port Connection to XS 40

(b) FPGA Interrupt Connection

(c) DIP Switches

(d) LEDs

(e) Jumper B1

R = 620 ohms
Rs = 220 ohms

Figure 1.4: External Connections Present on the UAH XS 40 Prototyping Environment

The Microcontroller (8031)

The 8031 series of microcontrollers are highly integrated single chip microcontrollers with an 8-bit CPU, memory, interrupt controller, timers, serial I/O and digital I/O on a single piece of silicon [1]. The actual microcontroller used in this board is an OKI 80C154S which is fully compatible with Intel’s 80C31 microcontroller. This microcontroller is the ROM-less version of the popular Intel 8051 family of microcontrollers.

The Xilinx Foundation Series Software

Designs that are entered on the XS 40 require the use of special Computer Aided Design, CAD, software to configure the Xilinx 4000-series FPGA. The software used by students is the Xilinx Foundation series tool suite. This software runs on a PC under Windows NT and it supports design entry (via schematic capture, state diagram entry, and hardware description language), logic synthesis, logic simulation, timing analysis, and device configuration.

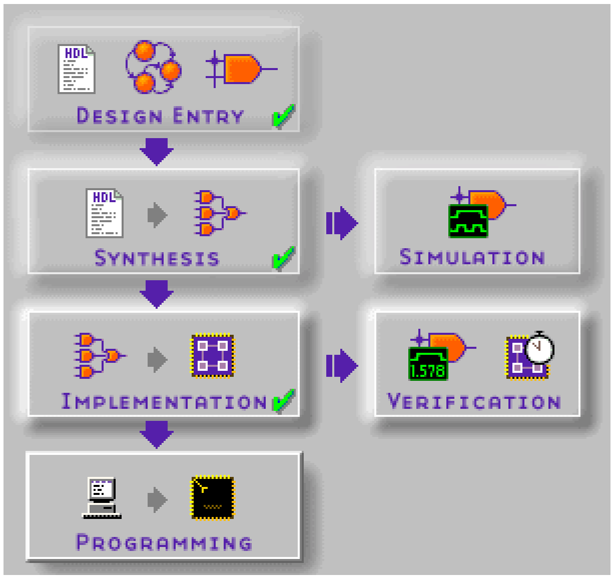


Figure 1.5: The General Design Cycle supported by the Xilinx Foundation Series Software

The general engineering design cycle which is supported by the Xilinx Foundation Series CAD software is highlighted in Figure 1.3. 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, 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 constrains. 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. The design examples discussed in this manual will utilize the GXLoad downloading facilities provided by XS corporation which utilizes the parallel port of a typical PC or unix workstation.

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 XS 40 (locking the Xilinx XC4010XL I/O pins to specific signals specified in the design), and configure (program) the XS 40. 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 XS 40. The counter will be clocked by a relatively slow clock (~1HZ) to allow the operation to be viewed on the XS 40 by the user. The clocking signal will originate from an internal 12 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.

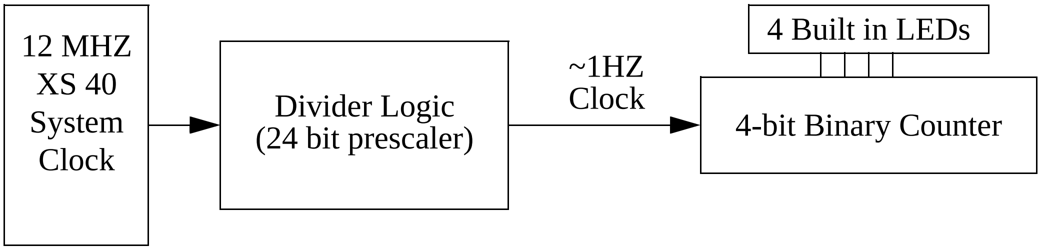
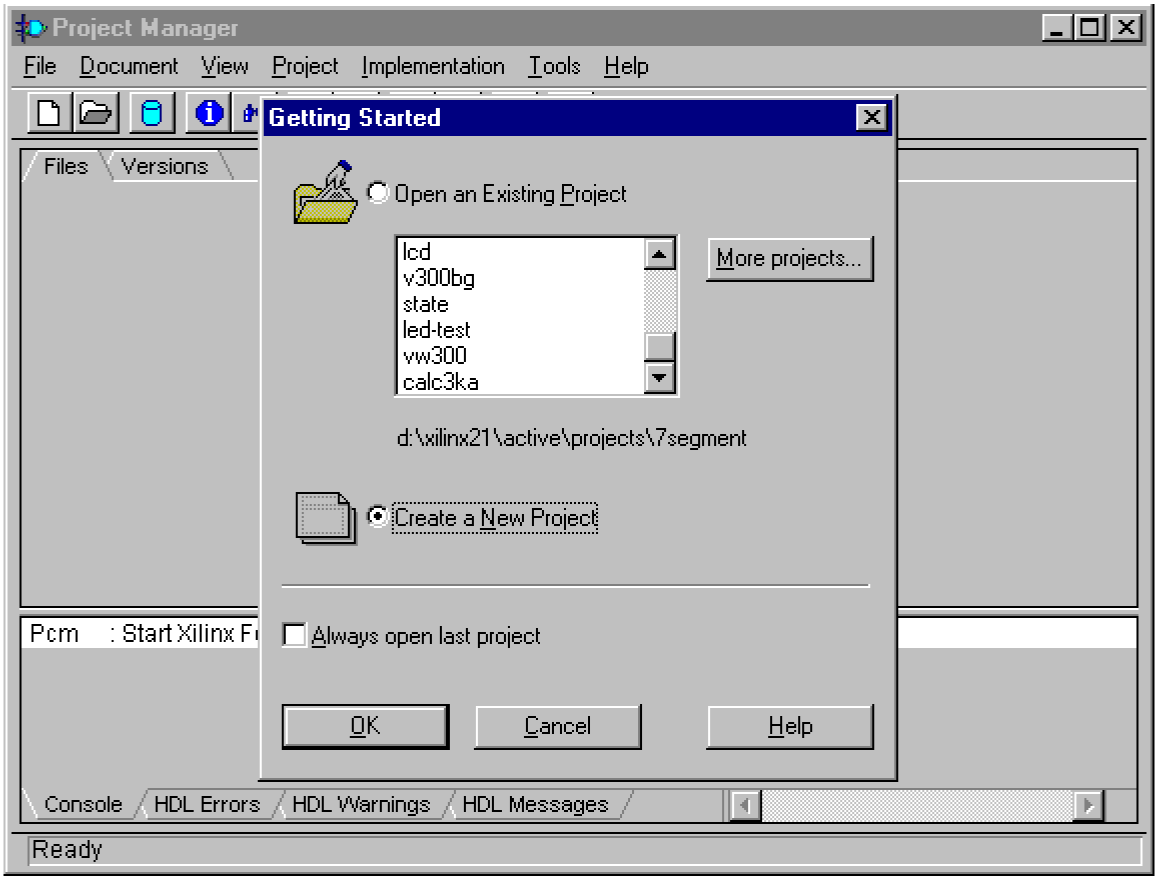


Figure 2.1: Binary Counter Example

Setting up a Project

To enter this design, first start the **Project Manager** of Xilinx Foundation Series 3.1i by double clicking on the **Project Manager** Icon, of Foundation 3.1i from the Window's Desktop.

The Project Manager will then display the **Getting Started** selection window. This window allows the user to **Open** an Existing Project or **Create** a New Project. For the binary counter design example, we would like to **Create a New Project**. Select this option as shown in Figure 2.2. Then click on the **OK** button.



○ Open an Existing Project

lcd
v300bg
state
led-test
vw300
calc3ka

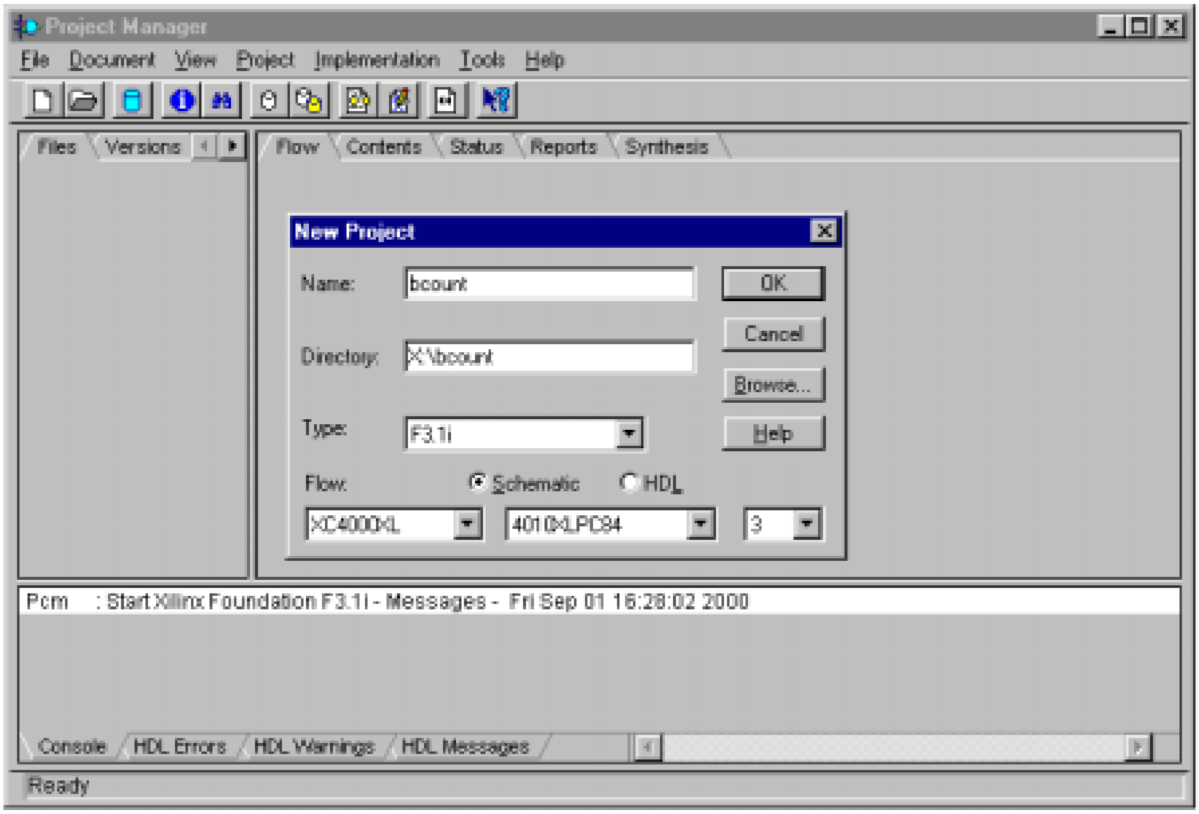
d:\xilinx21\active\projects\7segment

● Create a New Project

□ Always open last project

Figure 2.2: Xilinx Foundation Series Software Getting Started Window

A **New Project** window will then appear. It will have the following fields: **Name** of project, **Directory** (location to store files), and **Type** (i.e. version) of Xilinx Software. Since we are demonstrating schematic capture design capture techniques in this chapter, the **Flow** for this design should be set to **Schematic** as shown in Figure 2.3. (In Chapter 3 we will illustrate how to enter a hardware description language design and in Chapter 4 we will illustrate how to create a hybrid schematic and HDL design.) On the XC4000XL, the targeted FPGA device is a Xilinx 4000 family device, specifically a 4010XLPC84 FPGA (it is written directly on the device).



Name: bcount

Directory: X:\bcount

Type: F3.1i

Flow: ● Schematic ○ HDL

XC4000XL 4010XLPC84 3

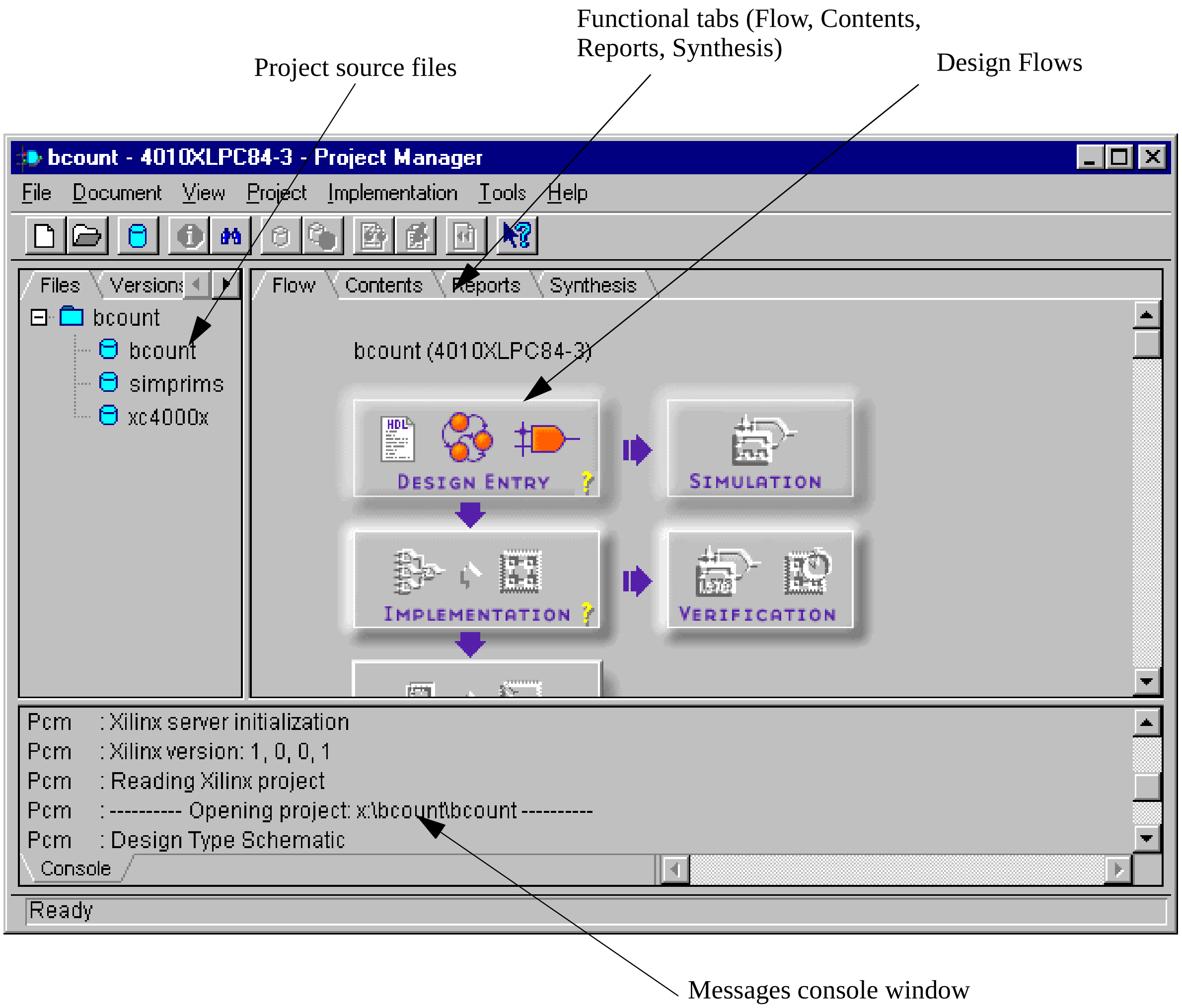
Pcm : Start Xilinx Foundation F3.1i - Messages - Fri Sep 01 16:28:02 2000

Ready

Figure 2.3: New Project Window

When all the fields are completed as shown in Figure 2.3 then single click on the **OK** button. The Xilinx Foundation Series **Project Manager** 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, point the cursor to the top level of the source tree, then press and hold the right mouse button and select the **Add** option. 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)



```
Pcm    : Xilinx server initialization
Pcm    : Xilinx version: 1, 0, 0, 1
Pcm    : Reading Xilinx project
Pcm    : --------- Opening project: x:\bcount\bcount ---------
Pcm    : Design Type Schematic
```

Figure 2.4: Xilinx Foundation Project Manager Window

On the right hand side of the **Project Manager** window are the functional tabs, **Flow**, **Contents**, **Reports**, and **Synthesis**. The function of each of these tabs is described below:

Flow - Provides quick access to the tools used in each stage of the design process

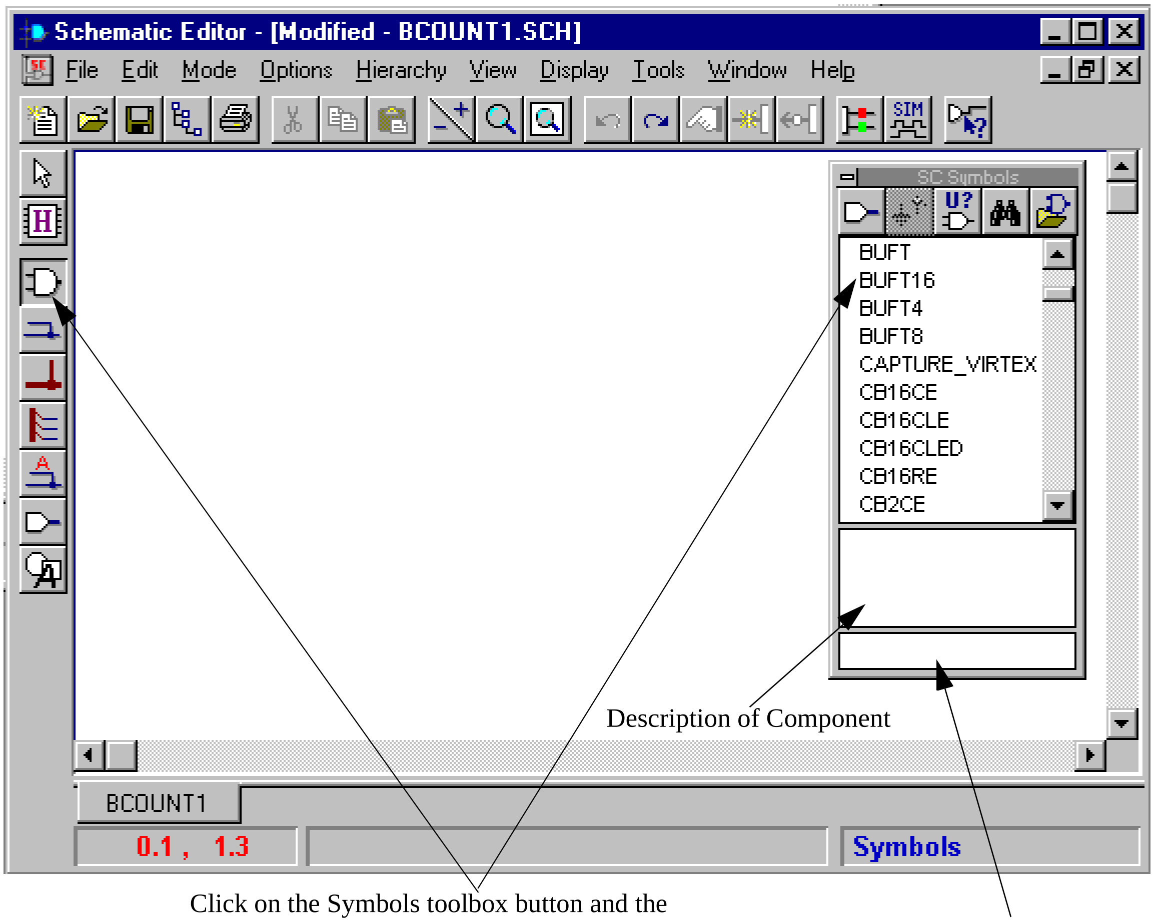
Contents - Lists contents and dates the design files were last modified

Reports - Contains the design constraints flow reports (Pin locking, Area, etc.)

Synthesis - Displays information about the HDL logic synthesis process

Design Entry

Design entry using schematic capture techniques involves employing a CAD tool (such as the Xilinx Foundation Series 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 click on the **Schematic Editor** Icon, , located on the **Design Entry** button under the **Flow** tag of the **Project Manager** window (see Figure 2.4).


Schematic Editor - [Modified - BCOUNT1.SCH]
File Edit Mode Options Hierarchy View Display Tools Window Help
SIM
SC Symbols
U?
BUFT
BUFT16
BUFT4
BUFT8
CAPTURE_VIRTEX
CB16CE
CB16CLE
CB16CLED
CB16RE
CB2CE
BCOUNT1
0.1 , 1.3
Symbols

Description of Component

Click on the Symbols toolbox button and the **SC Symbols Window** will appear which allows you to select components for your implementations.

Component name

Figure 2.5: Schematic Editor Window

To begin entering a graphical symbols, i.e. a components, first press the **Symbols** toolbox button . The **SC Symbols** window will appear which has a listing of components present in the libraries. Components can then be selected by scrolling down and clicking on the name of the component or by typing the name of the component in the dialog box which is at the bottom of the **SC Symbols** menu.

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, the **CB4CE** to form the basic four bit counter, and two instances of the **CC16CE** macro to form the prescaler logic. 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, **IPAD**, and **IBUF**, and four instances of the components **OPAD**, and **OBUF**. 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).

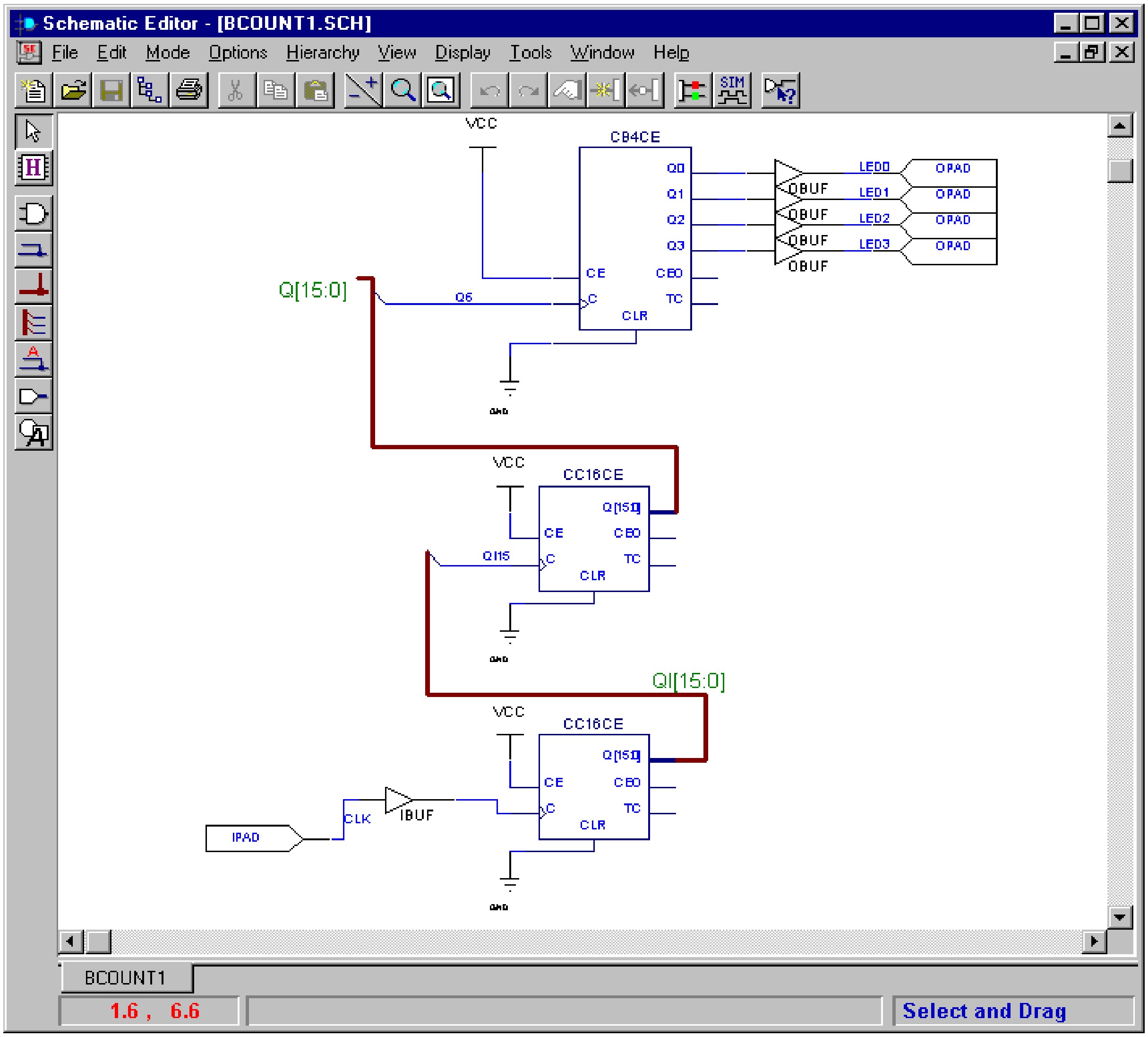


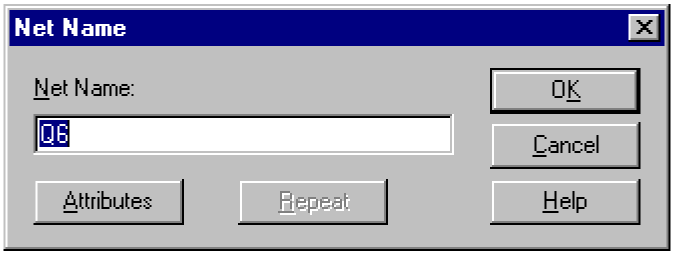
Figure 2.6: 4-bit Binary Counter Example

Figure 2.6 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 drawing tool by pressing the **F4** key on the keyboard. 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). Nets were created in a similar manner except that the bus drawing tool was invoked by pressing the **F5** key. (Bends in the nets and busses were made by single clicking before the net or bus was terminated on a component. Some busses were not terminated to a component. This was accomplished by moving the cursor to the desired bus termination point and double clicking the left mouse button.)

It should be noted, that the counter macros used in the design shown in Figure 2.6 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 (**F4**).

Individual nets (i.e. wires) were brought out from busses simply by connecting each net to the bus and giving the net the same **name** as the desired component of the bus. For example, in this design there is a simple bus named **Q** which has a range of **15** to **0**. This implies that there are 16 nets contained within the bus that are individually named **Q15**, **Q14**, ..., **Q1**, **Q0**, respectively. To logically connect a given net, such as **Q6** as shown in Figure 2.6, to the bus **Q**, required that the bus and net were first created using the appropriate drawing tools (by press **F5** and **F4** as described previously) then the bus was named **Q** and the net **Q6**.

To name a net or a bus, one first double clicks on the net or bus using the **Select and Drag** tool which is activated by pressing the ↖ icon, which is located on the upper left hand corner of the **Schematic Editor** window. When a net is selected the **Net Name** window will appear as shown in Figure 2.7. Simply enter the net name in the dialog box and press **OK**. For the binary counter design example first double click on the **Q6** net, then enter **Q6** in the dialog box, and press the **OK** button.

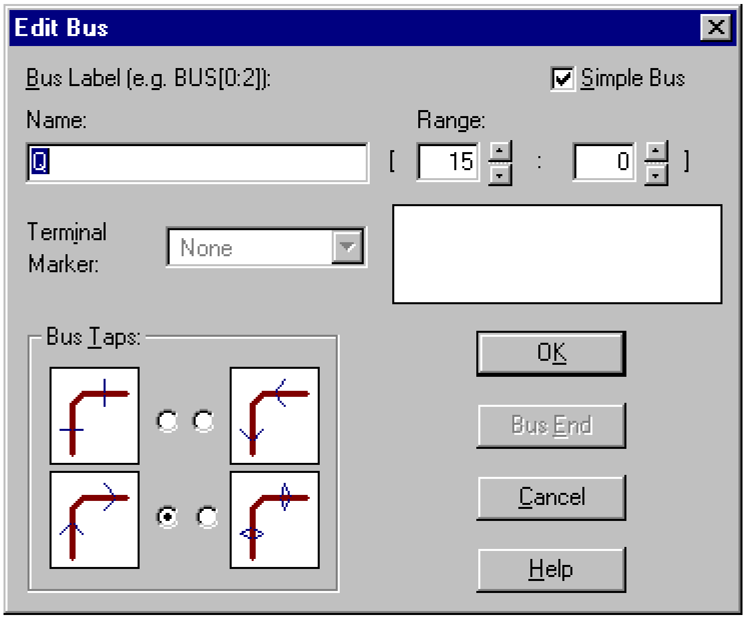


Net Name:

Q6

Figure 2.7: Net Name Window

When a bus is selected in this way, the **Edit Bus** window is displayed. The bus **Name** and the **Range** (i.e. starting number and ending number of the nets which are contained in the bus) can now be entered. To name the bus **Q[15:0]** in the binary counter design, one would double click on the bus, enter **Q** in the **Name** field, enter **15** and **0** in the **Range** fields, and then press the **OK**. button.



Bus Label (e.g. BUS[0:2]):

☑ Simple Bus

Name: Q

Range: [15 : 0]

Terminal Marker: None

Bus Taps: ○ ○ ● ○

Figure 2.8: Edit Bus Window

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 place 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.

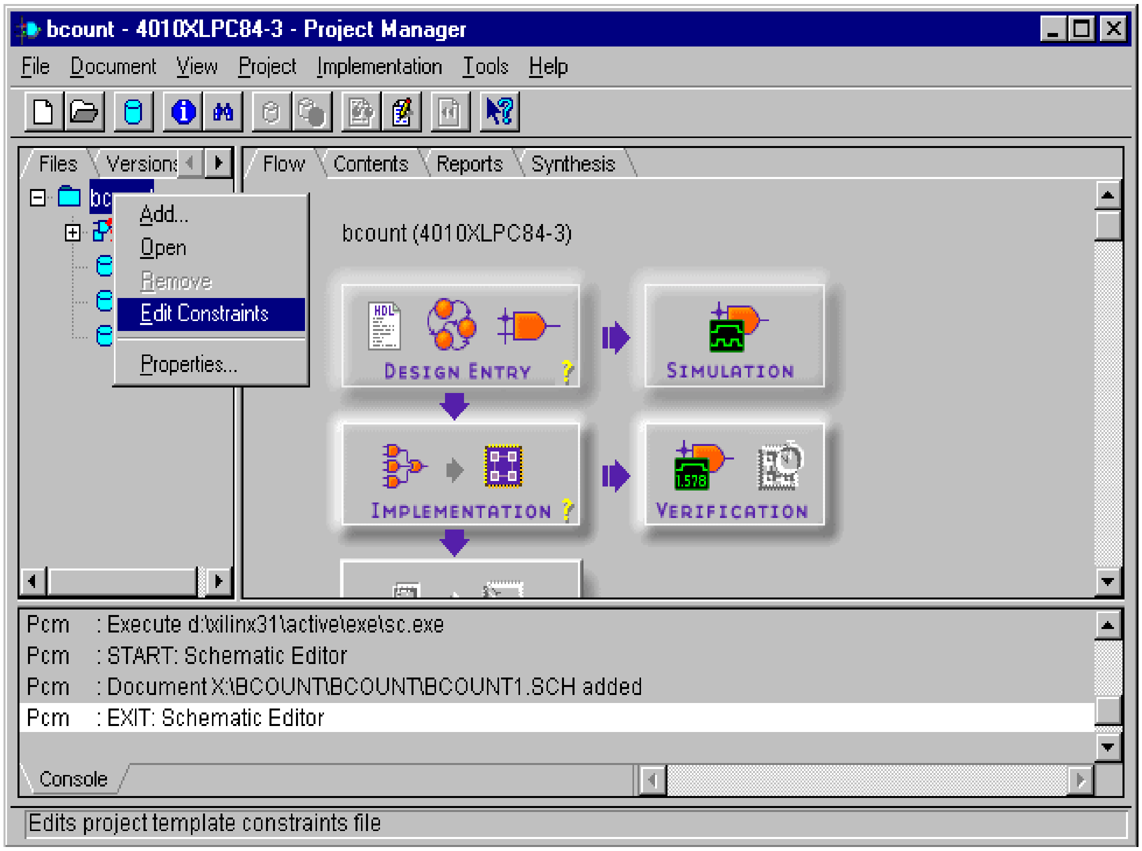
After the design has been entered, **Save** the file (It doesn't matter if you don't close the **Schematic Editor**, just remember to save the file. The **Project Manager** automatically saves the Schematic diagram by using your project name and concatenating a number to the end). Then return to the **Project Manager** 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 XC4010XL 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 XS 40 clock circuit into the Xilinx XC4010XL 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 Manager** window, move the cursor to point to the top hierarchy source tree located under the **Files** tab on the left-hand side of the window. Click on right most button and a pull down window will be displayed as shown in Figure 2.9. Then select the **Edit Constraints** option with the left mouse button.



```
Pcm   : Execute d:\xilinx31\active\exe\sc.exe
Pcm   : START: Schematic Editor
Pcm   : Document X:\BCOUNT\BCOUNT\BCOUNT1.SCH added
Pcm   : EXIT: Schematic Editor
```

Edits project template constraints file

Figure 2.9: Entering User Constraints from the Project Manager

For this design the desired schematic name to Xilinx XC4010X Pin name cross reference (i.e. Pin Locks) is shown in Table 2.1. The symbolic names for the XS 40 environment are shown in Appendix I of this manual.

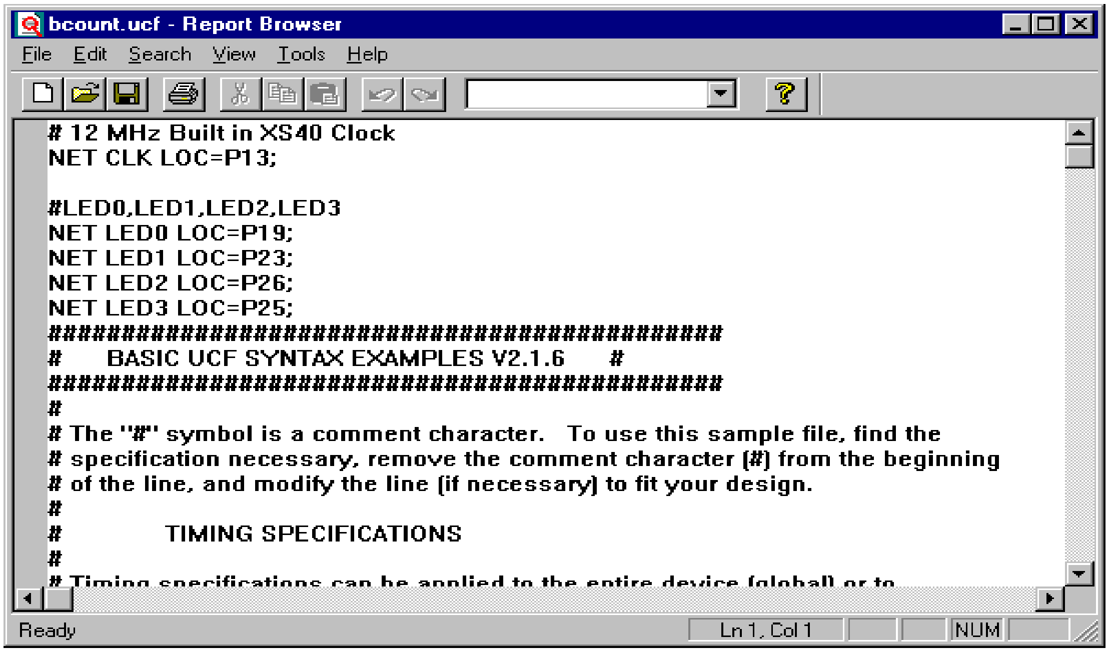
Table 2.1: Desired Pin Locking (Cross Reference) Configuration

I/O Pin Description	Schematic Net Name	Xilinx XC4010XL Pin Name
XS 40 12 MHZ Clock	CLK	13
XS 40 Discrete LED #0	BIT0	19
XS 40 Discrete LED #1	BIT1	23
XS 40 Discrete LED #2	BIT2	26
XS 40 Discrete LED #3	BIT3	25

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 Manager** window.



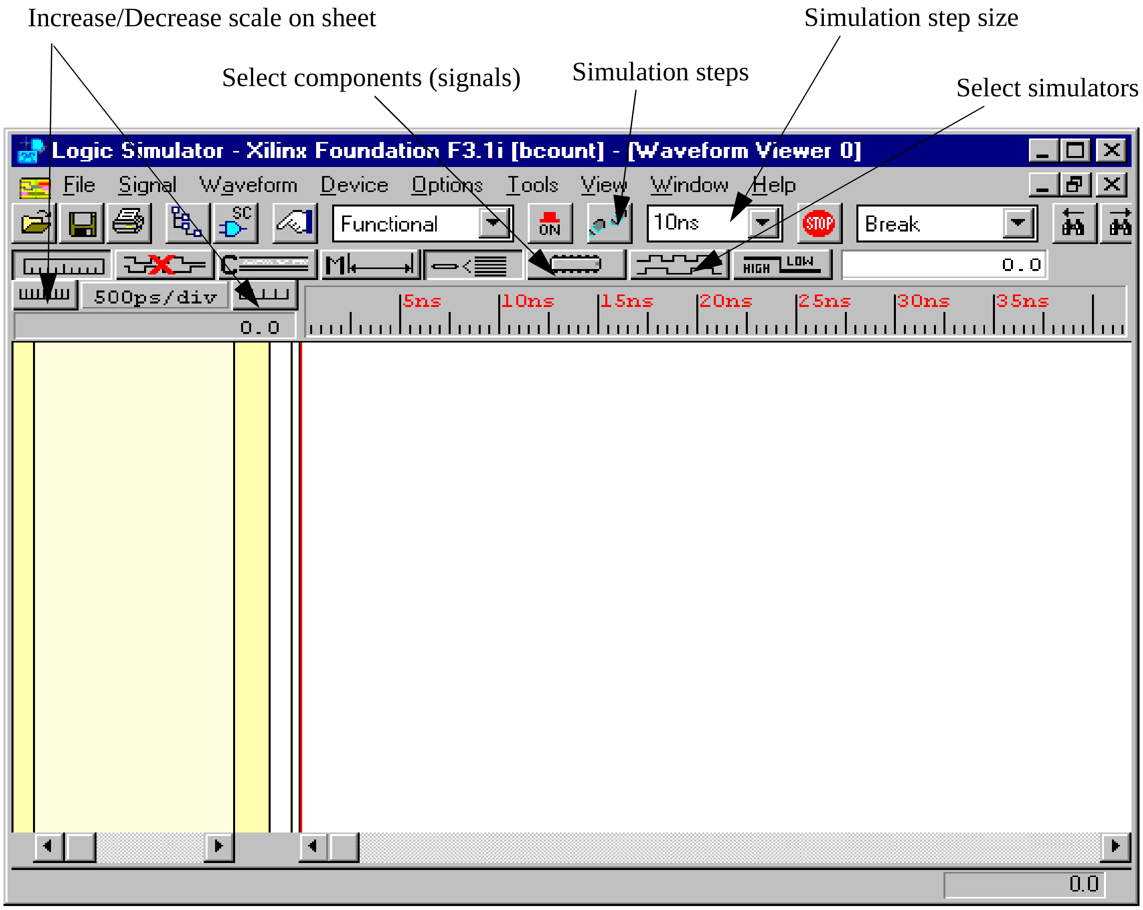
```
# 12 MHz Built in XS40 Clock
NET CLK LOC=P13;

#LED0,LED1,LED2,LED3
NET LED0 LOC=P19;
NET LED1 LOC=P23;
NET LED2 LOC=P26;
NET LED3 LOC=P25;
##################################################
#     BASIC UCF SYNTAX EXAMPLES V2.1.6      #
##################################################
#
# The "#" symbol is a comment character.   To use this sample file, find the
# specification necessary, remove the comment character (#) from the beginning
# of the line, and modify the line (if necessary) to fit your design.
#
#            TIMING SPECIFICATIONS
#
# Timing specifications can be applied to the entire device (global) or to
```

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

Functional Simulation Phase

We are now at the point in the design cycle where the design has been entered and the pin locking constraints have been specified. The next most common step is the **Functional Simulation** phase, in which one is able to check the logical correctness of a design before it is actually implemented on the targeted system (which is in the case the XS 40). Design errors found in this phase can then be corrected in an iterative manner by re-entering the design entry phase. While this phase can be bypassed with the behavior of the design being observed after the device is configured, this is not recommended since simulation often gives a much more detailed view of the operation of the design than is usually observable directly from the implementation. There are also some design errors that can be identified through simulation which might be destructive if implemented in hardware. Simulation is a powerful tool. The following is a brief overview of how simulation can be used to verify the functionality of the binary counter example. A much more detailed discussion of this material is presented in the Xilinx literature.

To perform a simulation of the four-bit binary counter example, first click on the **Functional Simulation** button, on **Project Manager** window. This should launch the **Logic Simulator** window shown in Figure 2.11.


Increase/Decrease scale on sheet

Select components (signals)

Simulation steps

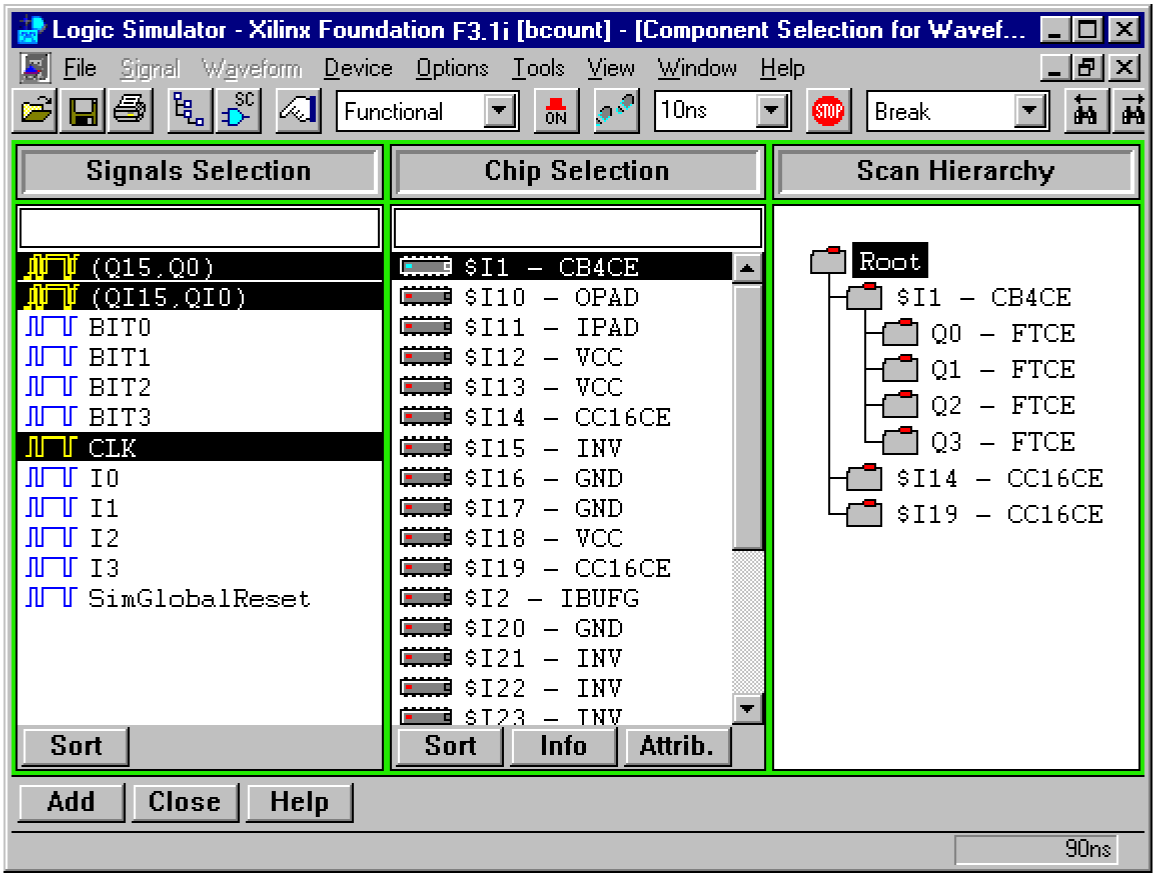
Simulation step size

Select simulators

Figure 2.11: Logic Simulator Window

One of the first steps in the simulation process is to choose the set of signals which one wants to observe during the simulation. Obvious candidates are the global input and output nets/busses of the design as well as certain internal signals that are present. For the binary counter example, we will choose for demonstration purposes to observe the global **CLK** input and a set of internal busses, **QI15:0** and **Q15:0**. [Note: This is not really a good set of signals to observe for this design, but complete simulation runs which would exercise the main counter output lines, **BIT0-BIT3**, would take several hours to complete. The large amount of computer simulation time required to simulate sequential circuits is one of the major drawback of simulation. One could get an idea of how well the circuit works though by running the following simulation and then temporarily modifying the design to remove the prescaler.] To select these nets/busses click on the

Select Components button, . Then at the **Select Signals** windows, choose the desired signals to be included in the simulation. Do this by clicking on each signal with the left mouse button. Then to select more than one signal, hold on the **<ctrl>**-key and click on the signals with left mouse button. Figure 2.12 shows the case where the **CLK**, **QI15:0** and **Q15:0** signals have all been highlighted using this method. To complete the process, press the **Add** button and then the **Close** button.


Logic Simulator - Xilinx Foundation F3.1i [bcount] - [Component Selection for Wavef...
File Signal Waveform Device Options Tools View Window Help
Functional
10ns
Break
Signals Selection
Chip Selection
Scan Hierarchy
Sort
Sort
Info
Attrib.
Add
Close
Help
90ns

Signals Selection:

```
(Q15.Q0)
(QI15.QI0)
BIT0
BIT1
BIT2
BIT3
CLK
I0
I1
I2
I3
SimGlobalReset
```

Chip Selection:

```
$I1  - CB4CE
$I10 - OPAD
$I11 - IPAD
$I12 - VCC
$I13 - VCC
$I14 - CC16CE
$I15 - INV
$I16 - GND
$I17 - GND
$I18 - VCC
$I19 - CC16CE
$I2  - IBUFG
$I20 - GND
$I21 - INV
$I22 - INV
$I23 - INV
```

Scan Hierarchy:

```
Root
  $I1 - CB4CE
    Q0 - FTCE
    Q1 - FTCE
    Q2 - FTCE
    Q3 - FTCE
  $I14 - CC16CE
  $I19 - CC16CE
```

Figure 2.12: Signal Selection Window

Figure 2.13 shows the state of the **Logic Simulator** window after the three sets of signals have been added to the observation list. They appear on the left hand side of the window. To start the simulation, we must first define the stimulus pattern for all of the external inputs to the design. In the 4-bit binary counter example, the only input to the design is the **CLK** signal which needs to be driven in a periodic manner. The Xilinx simulation software allows the user to define the waveform pattern associated with each of four stimulus clocks which can be assigned to selected inputs of the design. For this example, we will define the input pattern associated with one of these stimulus clocks and then assign it to drive the **CLK** input.

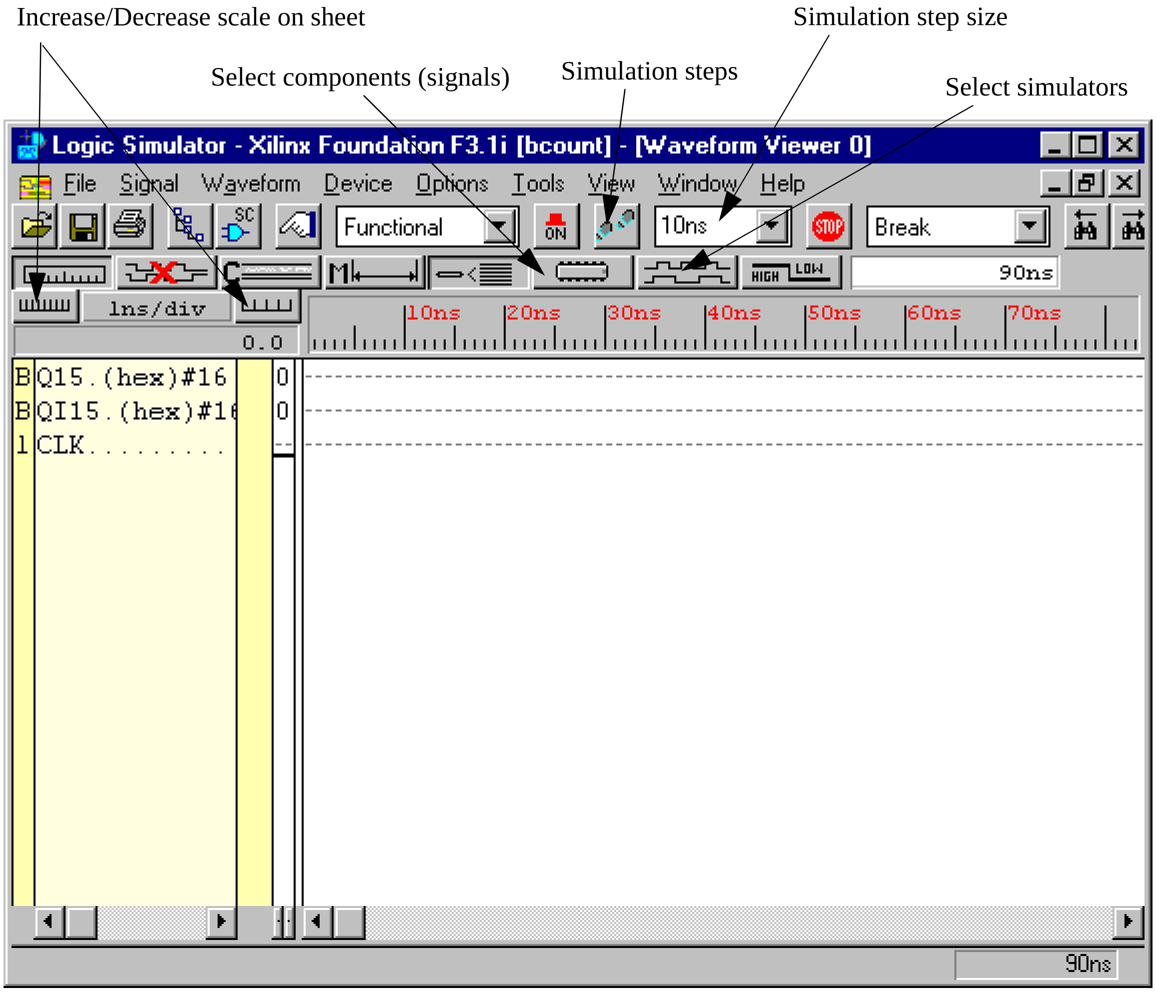


Figure 2.13: Logic Simulator Window State after Selecting Signals

To define the desired clock stimulus waveform, click on the **Select Stimulators** button, . The **Stimulator Selection** window will appear as shown in Figure 2.14. On this window there are several **Clocks** labeled **C1-C4** which can be defined. Then click on the **Formula** button.

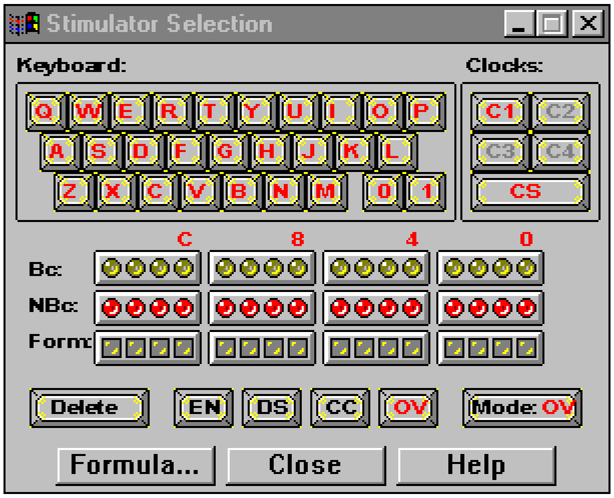
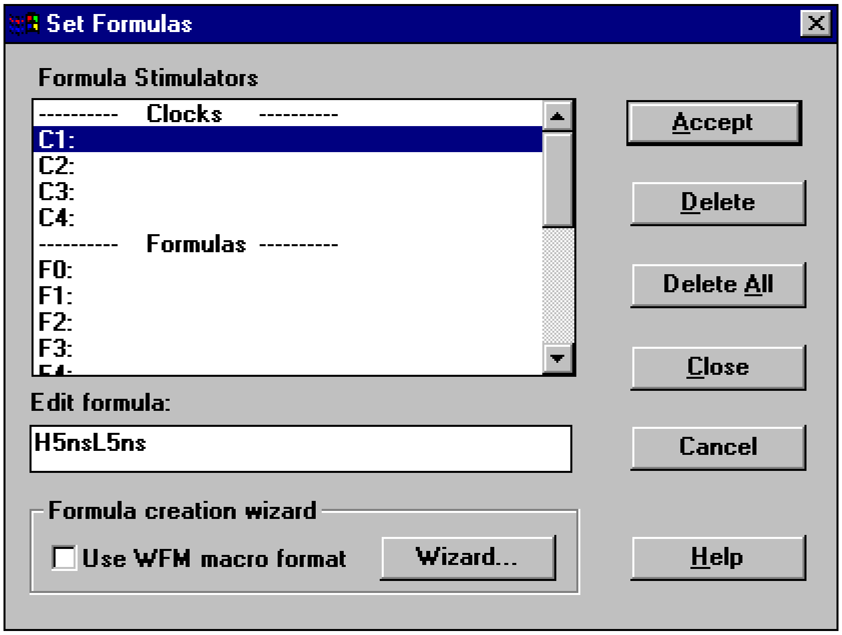


Figure 2.14: Stimulator Selection Window

A **Select Formula** window will appear as shown in Figure 2.15. In the **Formula Stimulators** area of the window double click on **C1**. Then in the **Edit Formula** dialog box, type in the pattern H5nsL5ns. This defines a simple clock cycle that is logic high 5ns and then is logic low 5ns (i.e. a clock cycle of 10ns). Then click on the **Accept** and **Close** buttons to exit the window.



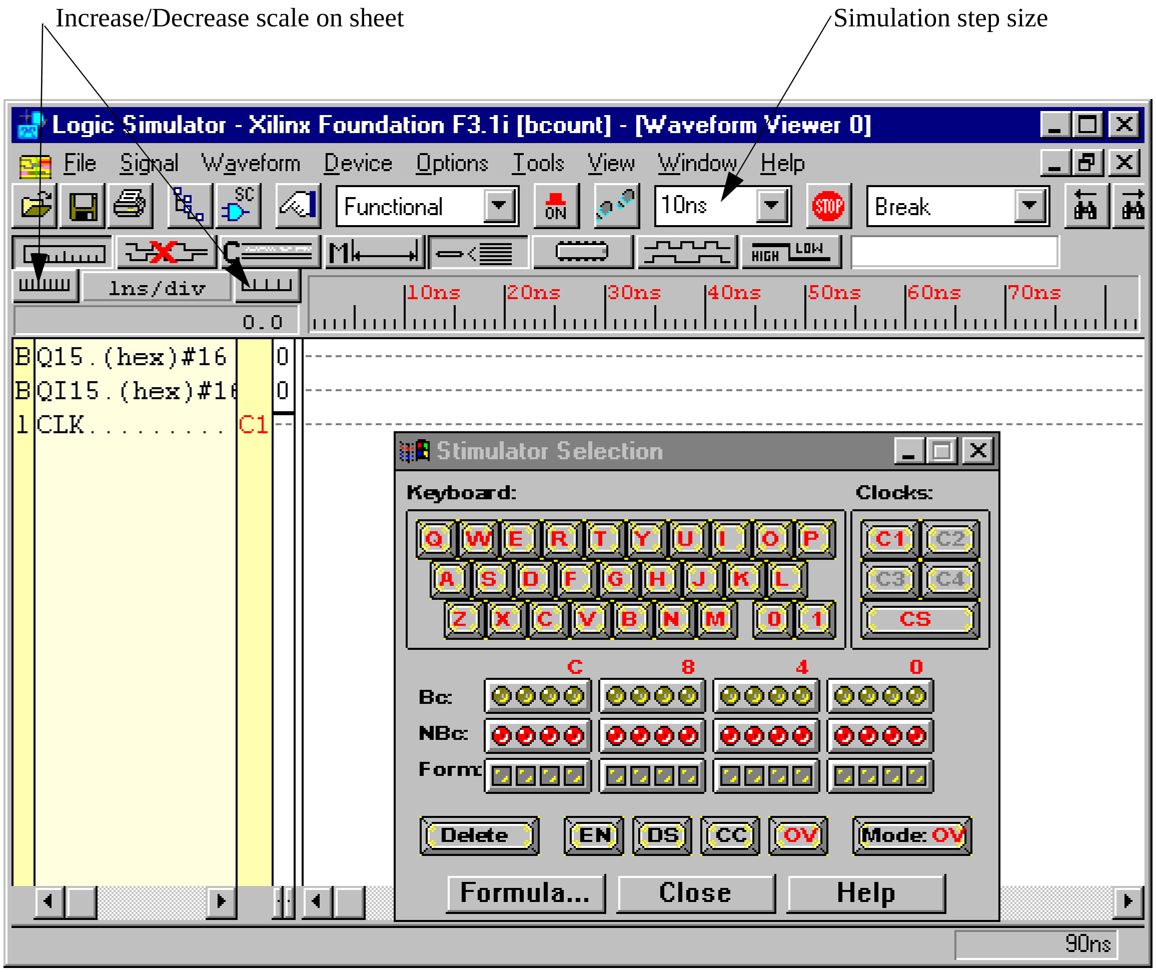
---------- Clocks ----------
C1:
C2:
C3:
C4:
---------- Formulas ----------
F0:
F1:
F2:
F3:

Edit formula: H5nsL5ns

☐ Use WFM macro format

Figure 2.15: Set Formulas Window

After the **Set Formulas** window is closed, the **C1** button in the **Stimulator Selection** window will change from gray to red color. This means that **C1** is now a valid **Clock** stimulator. This clock stimulator will now be available to drive input signals during the simulation. To assign this stimulus to the **CLK** signal in the binary counter design first make sure both the **Logic Simulator** and **Stimulator Selection** windows appear on the screen at the same time. Then click once on the **CLK** signal on **Logic Simulator** window, and move the cursor over to the **Stimulator Selection** window and click once on the clock **C1** button. This will cause the **C1** value to appear along side the clock signal name in the **Logic Simulator** window. Since all of the stimulus needed to drive the binary counter example has been defined and the appropriate signals to observe have been selected the design is almost ready to be simulated.


Increase/Decrease scale on sheet

Simulation step size

Logic Simulator - Xilinx Foundation F3.1i [bcount] - [Waveform Viewer 0]

Functional

10ns

Break

1ns/div

0.0

B Q15.(hex)#16 0

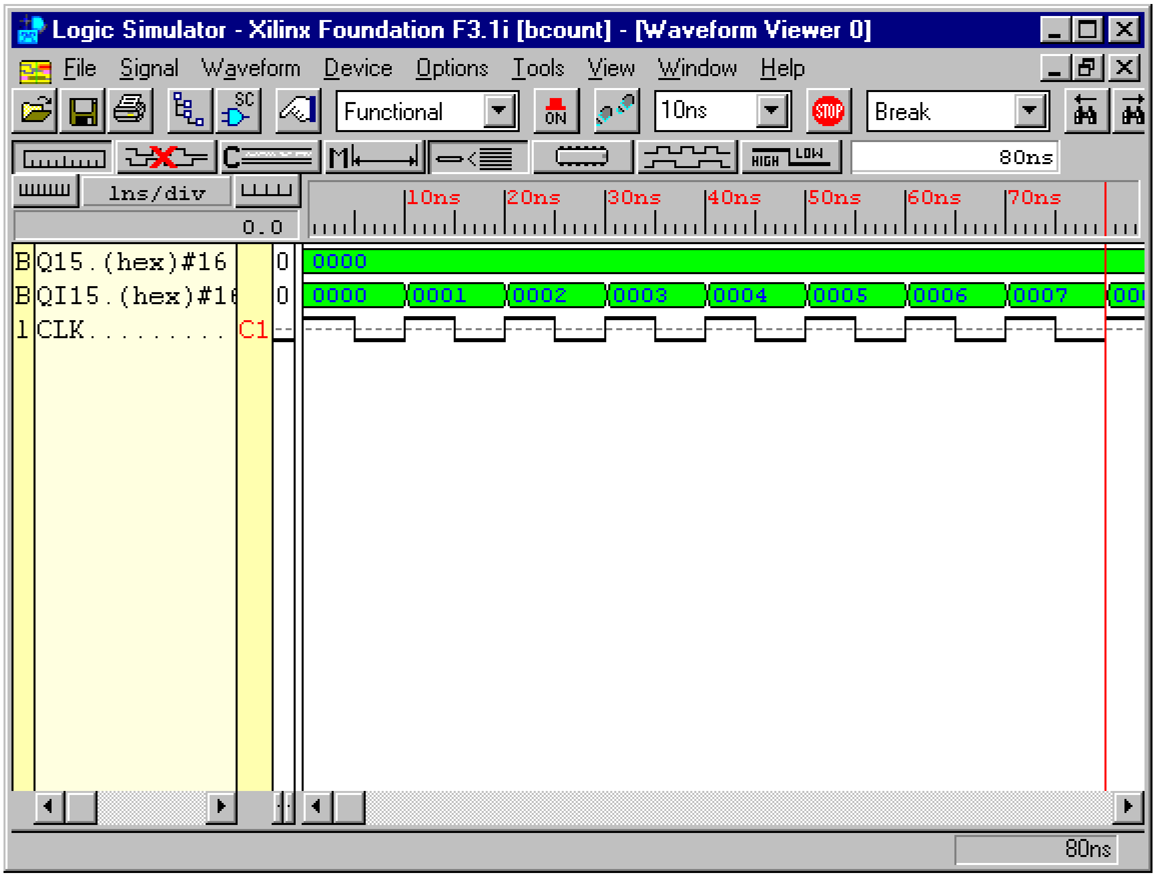
B QI15.(hex)#16 0

1 CLK......... C1

90ns

Figure 2.16: Assigning Clock Stimulus to CLK Input

Before simulating the four-bit binary counter make sure that the display sheet time scale and the simulation step size are set to appropriate values to accommodate the 10ns clock cycle of the stimulus signal. This is accomplished by clicking on the increase time scale, , and decrease time scale buttons, , on the **Logic Simulation** window and entering the simulation step size in the simulation step size dialog box. To perform the simulation in a step by step manner click on the **Simulation Step** button, , as many times as is necessary to observe the selected signals. Figure 2.17 shows the waveforms that occur during the first 80nS of the binary counter simulation.


Logic Simulator - Xilinx Foundation F3.1i [bcount] - [Waveform Viewer 0]
File Signal Waveform Device Options Tools View Window Help
ON
STOP
1ns/div
0.0
10ns 20ns 30ns 40ns 50ns 60ns 70ns
80ns

Functional

10ns

Break

B Q15.(hex)#16 0 0000

B QI15.(hex)#16 0 0000 0001 0002 0003 0004 0005 0006 0007 000

1 CLK.......... C1

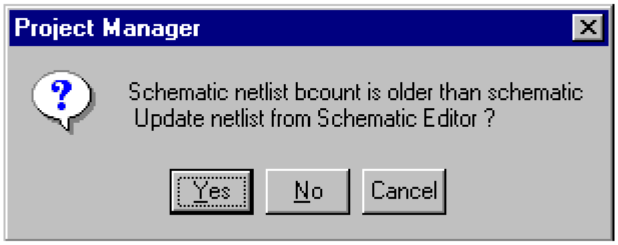
80ns

Figure 2.17: Example Simulation

Observe and study the waveforms carefully. When the simulation has been determined to be correct, **Exit** the **Logic Simulator** window and return to **Project Manager** window.

Implementation Phase

Now it is time to take the design through the compilation process. To do this first click on the **Implementation** button, , on the **Project Manager** window. A netlist generation confirmation message similar to the one shown in Figure 2.18 will appear. Select **Yes**, since it is desirable to use the latest netlist of the design.

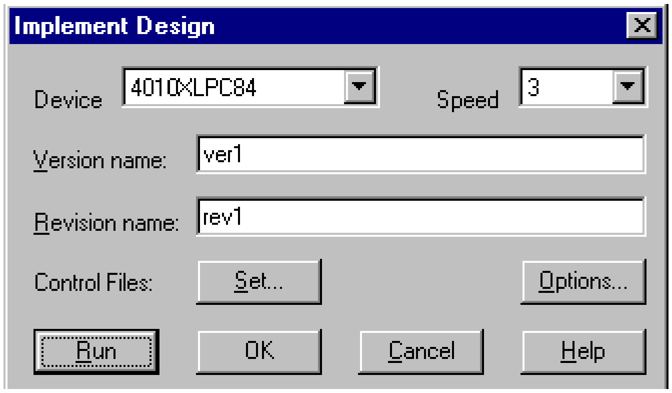


Project Manager

Schematic netlist bcount is older than schematic
Update netlist from Schematic Editor ?

Figure 2.18: Netlist Generation Confirmation

Then an **Implement Design** window will appear. All the parameters in the dialog boxes should be as shown in Figure 2.19. Click on the **Run** button to continue.

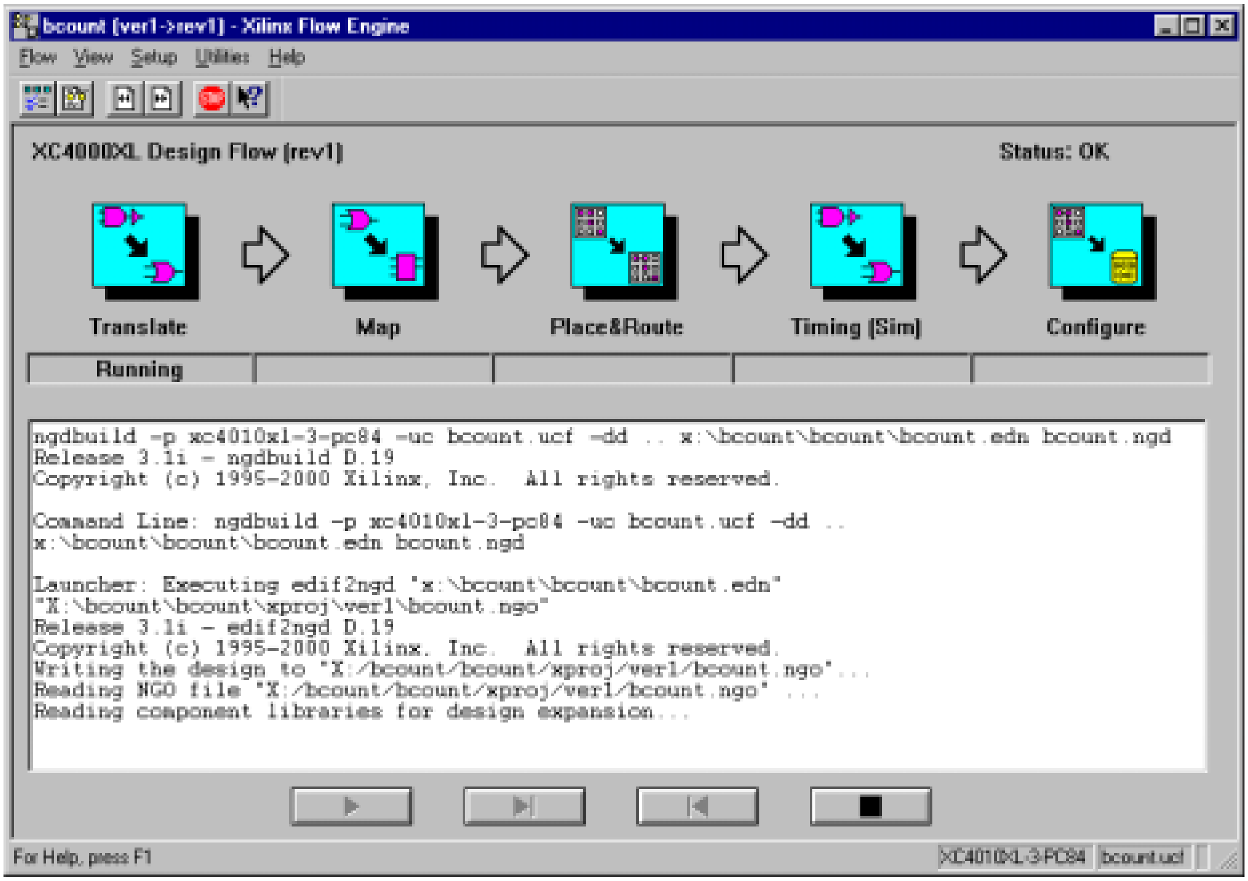


Implement Design

Device: 4010XLPC84
Speed: 3
Version name: ver1
Revision name: rev1
Control Files:

Figure 2.19: Implement Design Window

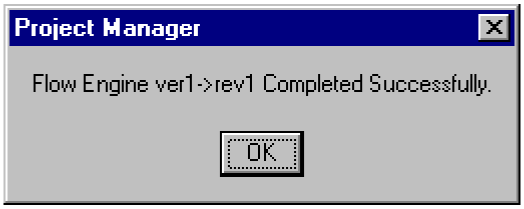
After this, a **Compilation Flow Engine** window will appear with appropriate implementation information being sent to the status area of this window as the design compilation process progresses.



```
ngdbuild -p xc4010xl-3-pc84 -uc bcount.ucf -dd .. x:\bcount\bcount\bcount.edn bcount.ngd
Release 3.1i - ngdbuild D.19
Copyright (c) 1995-2000 Xilinx, Inc.  All rights reserved.

Command Line: ngdbuild -p xc4010xl-3-pc84 -uc bcount.ucf -dd ..
x:\bcount\bcount\bcount.edn bcount.ngd

Launcher: Executing edif2ngd "x:\bcount\bcount\bcount.edn"
"X:\bcount\bcount\xproj\ver1\bcount.ngo"
Release 3.1i - edif2ngd D.19
Copyright (c) 1995-2000 Xilinx, Inc.  All rights reserved.
Writing the design to "X:/bcount/bcount/xproj/ver1/bcount.ngo"...
Reading NGO file "X:/bcount/bcount/xproj/ver1/bcount.ngo" ...
Reading component libraries for design expansion...
```

Figure 2.20: Compilation Flow Engine Window

This engine will take the design through the Translate, Map, Place&Route, Timing, and Configure steps. At the end of process, many files will be generated which can be observed by clicking on the **Report** tab on the right hand side of the **Project Manager** window. Errors in this process will require modifications to the original design. When the implementation portion of the design is successfully completed the following message will appear

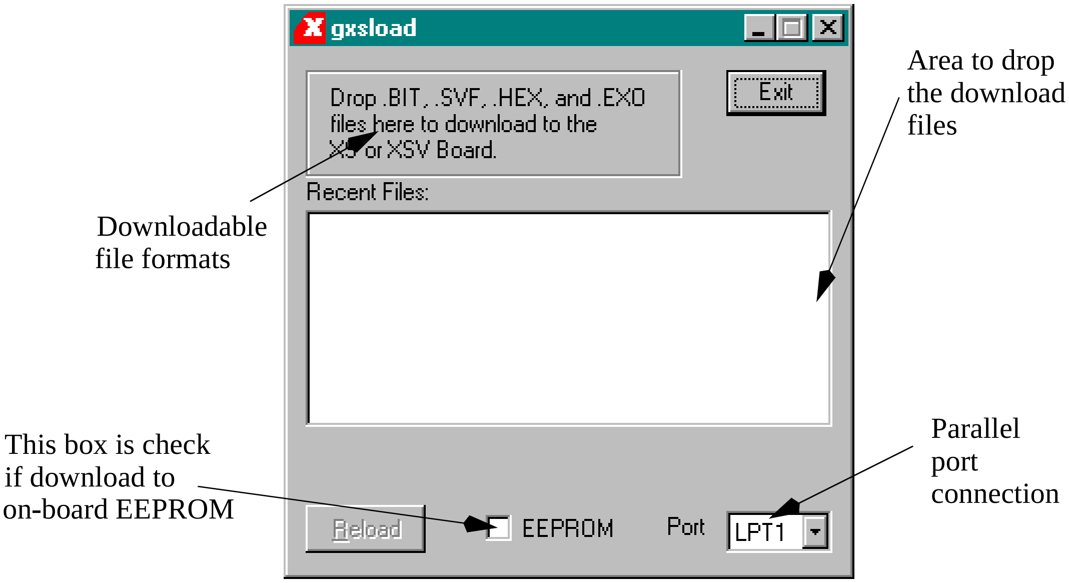


Flow Engine ver1->rev1 Completed Successfully.

Click on the **OK** button. Then return to the **Project Manager** window to prepare to actually download the design to configure the targeted hardware.

Programming (Device Configuration) Phase

To configure the XS 40 (i.e. download the bit map file that was created by the Xilinx Foundation CAD software), simply follow the following procedure. First, connect the XS 40 to the AC adapter and make sure the parallel cable is connected to the power supply side of the XS 40. Connect the other end of the cable to the host PC's parallel port. Then from the Windows desktop, **click** on the **gxsload** icon to bring up the **gxsload** window as shown in Figure 2.21. Then drag the corresponding bit map file (**bcount.bit** in this case) and drop it into the **gxsload** window. The bit map file should always be under the student's project directory on the **X:** drive.



Drop .BIT, .SVF, .HEX, and .EXO files here to download to the XS or XSV Board.

Area to drop the download files

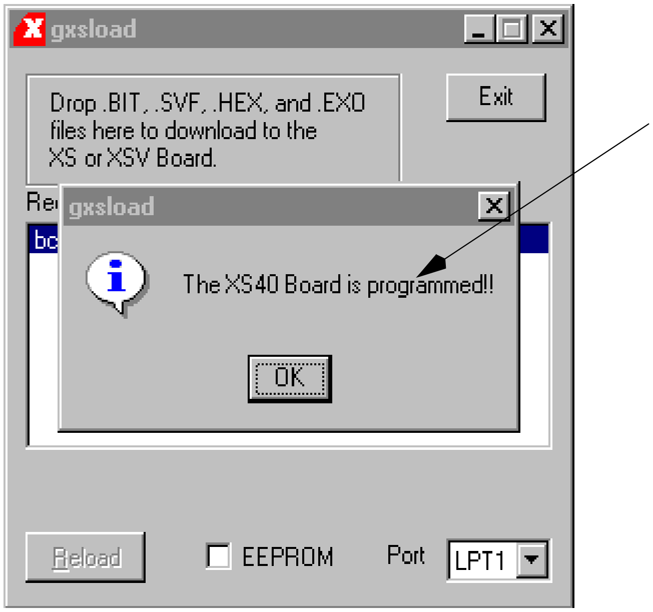
Downloadable file formats

This box is check if download to on-board EEPROM

Parallel port connection

Figure 2.21: The Downloading Program, GXLoad.

After this is done the device should begin the configuration process which will take a few seconds. A status window similar to the one shown in Figure 2.22 will appear upon successful completion of the download process.



Drop .BIT, .SVF, .HEX, and .EXO files here to download to the XS or XSV Board.

The XS40 Board is programmed!!

Success
download
message

Figure 2.22: The Downloading Program, GXLoad.

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 XS 40. The design will is to drive an external seven-segment common anode LED that is to be connected to pins 8, 9, 6, and 77 of XS 40 (which comes directly from the Xilinx XC4010XL FPGA) 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 DIP 1 -- DIP4).

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.:

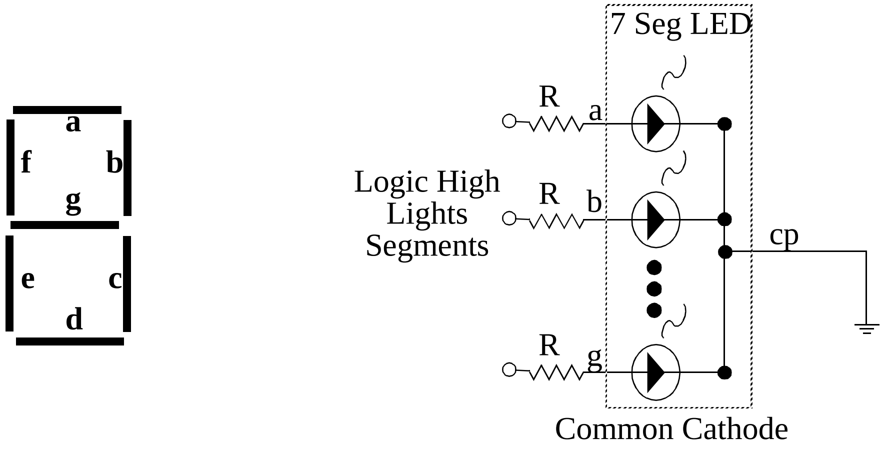


Figure 3.1: Seven Segment Display Unit

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.

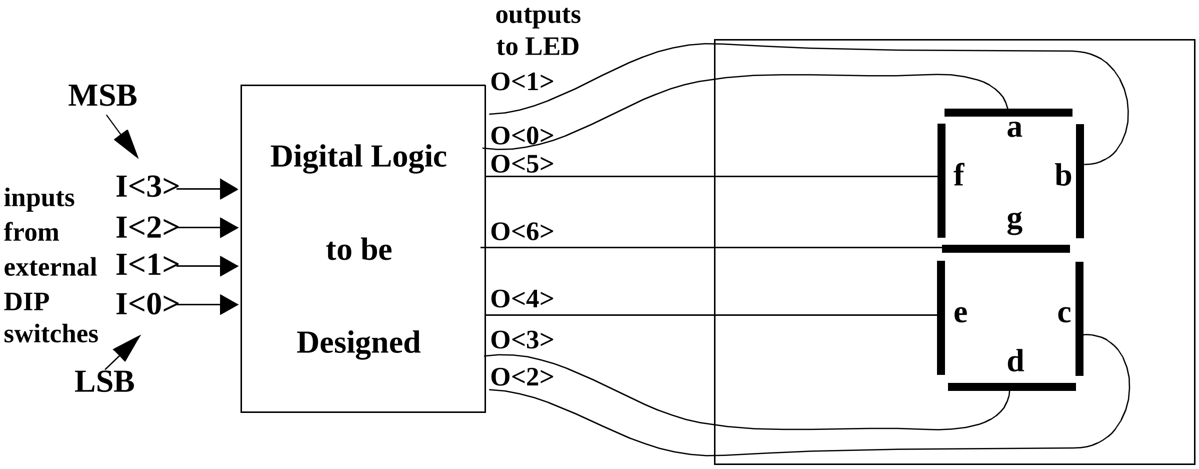


Figure 3.2: Display Converter Seven Segment Display System Overview

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 **Project Manager** window in the same manner as was done previously, by double clicking on the **Project Manager** icon, from the Window's Desktop.

As with the previous example that was presented in Chapter 2, the **Project Manager** will display the **Getting Started** selection window. This window allows one to **Open** an existing project or **Create** a new project. For the binary to hexadecimal converter example, we would like to **Create a New Project**. Select this option as shown in Figure 2.2. Then click the **OK** button.

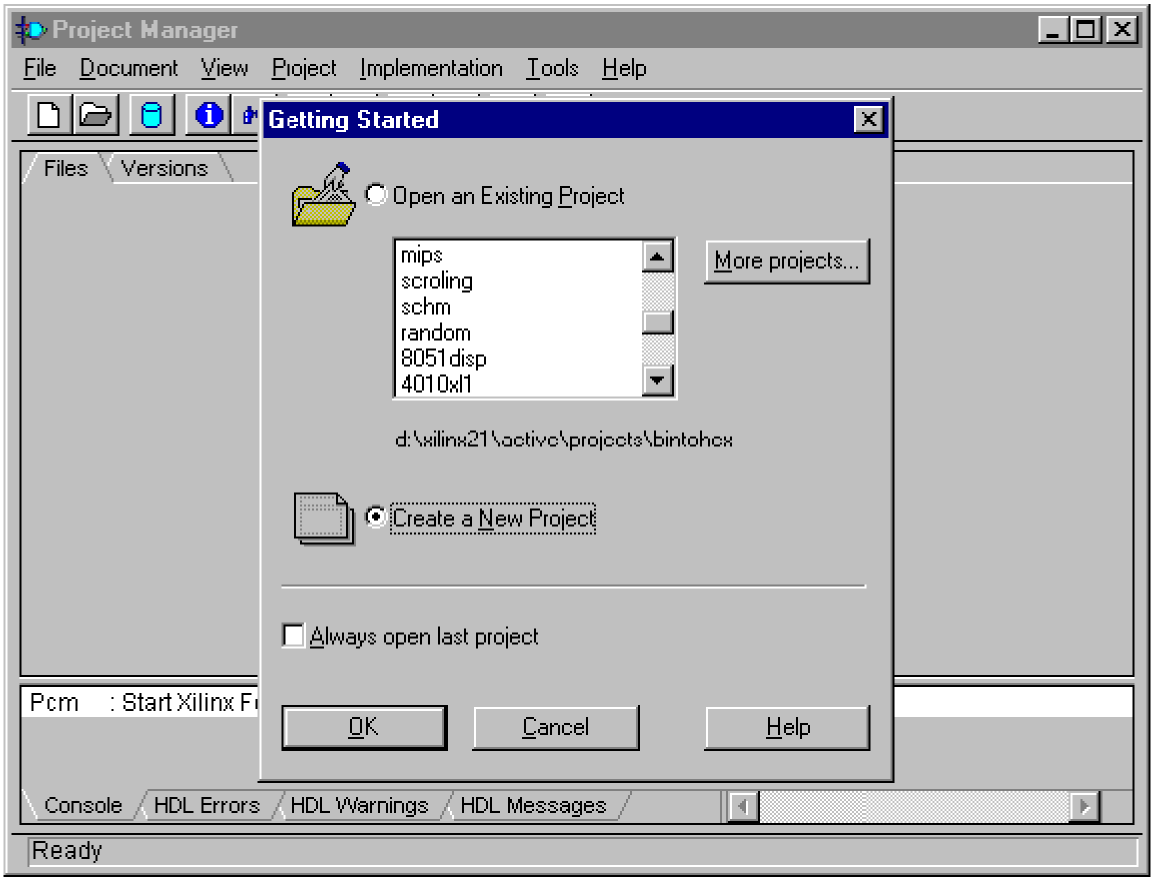
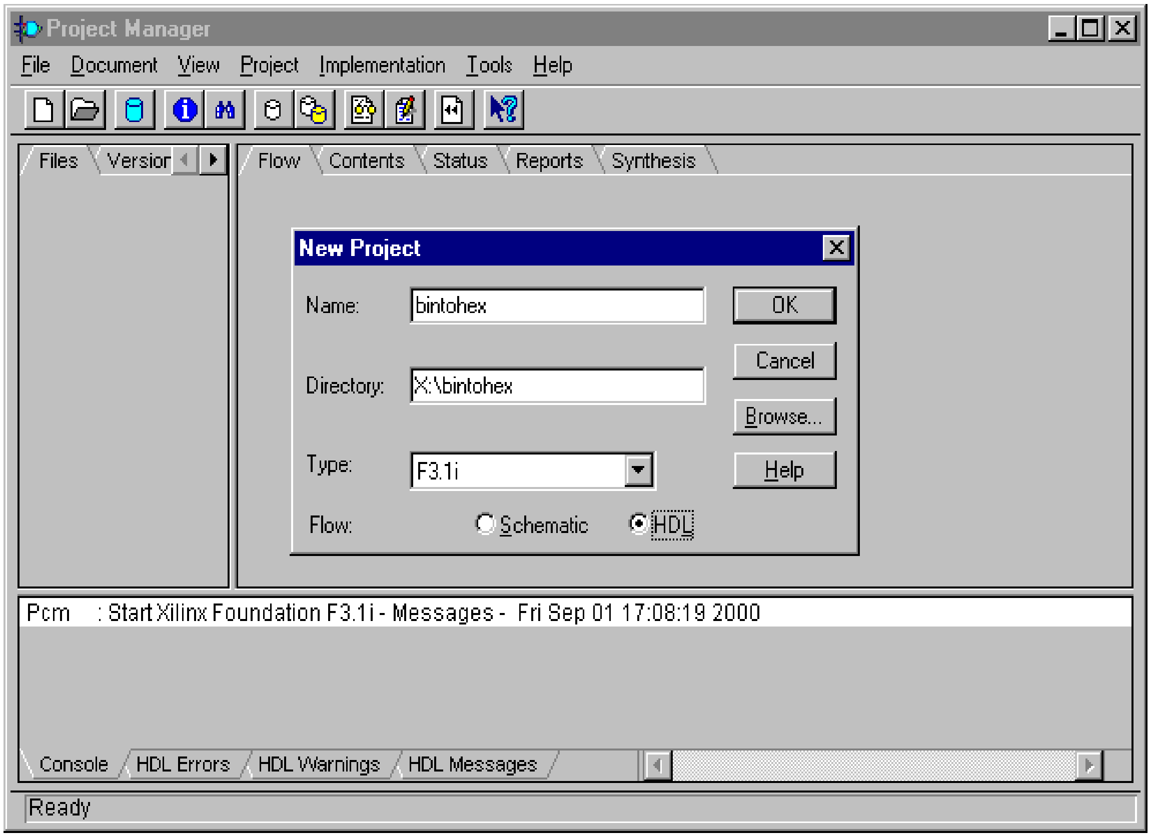

Project Manager
File Document View Project Implementation Tools Help
Getting Started
Files Versions
○ Open an Existing Project
mips
scroling
schm
random
8051disp
4010xl1
More projects...
d:\xilinx21\active\projects\bintohex
● Create a New Project
□ Always open last project
OK
Cancel
Help
Pcm : Start Xilinx F
Console HDL Errors HDL Warnings HDL Messages
Ready

Figure 3.3: Getting Started Window

At this point a **New Project** window will appear as shown in Figure 3.4. Let us call the binary to hexadecimal converter design, bintohex. Enter this name in the **Name** field. Make sure that the **Directory** field is set to the project directory and the **Type** field is set to the appropriate version of the Xilinx Foundation tool set (in this case, Foundation 3.1i). Also 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. Then click on the **OK** button.


Project Manager
File Document View Project Implementation Tools Help
Files Versior
Flow Contents Status Reports Synthesis
New Project
OK
Cancel
Browse...
Help
Console HDL Errors HDL Warnings HDL Messages

Name: bintohex

Directory: X:\bintohex

Type: F3.1i

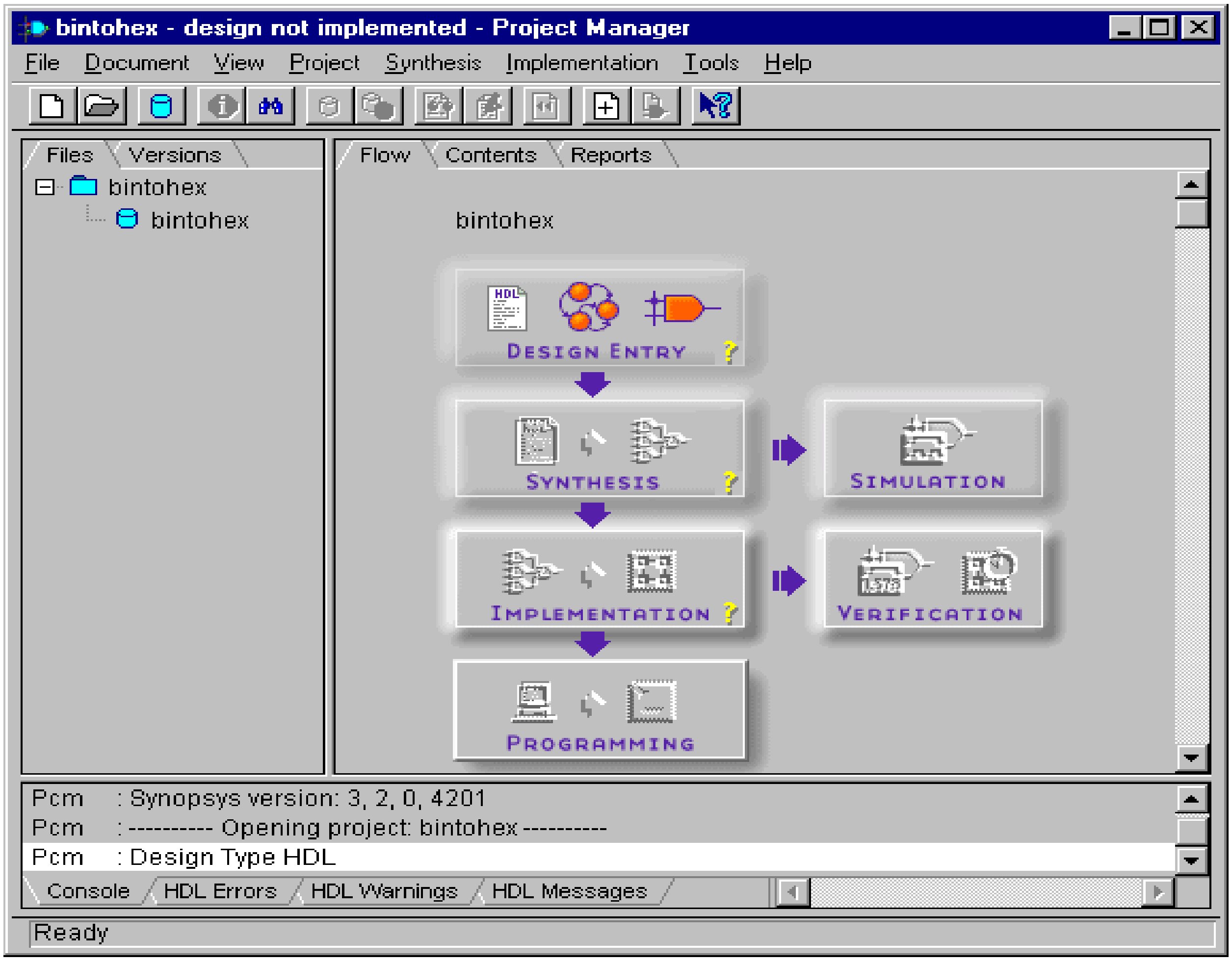
Flow: ○ Schematic ● HDL

```
Pcm    : Start Xilinx Foundation F3.1i - Messages -  Fri Sep 01 17:08:19 2000
```

Ready

Figure 3.4: New Project Window

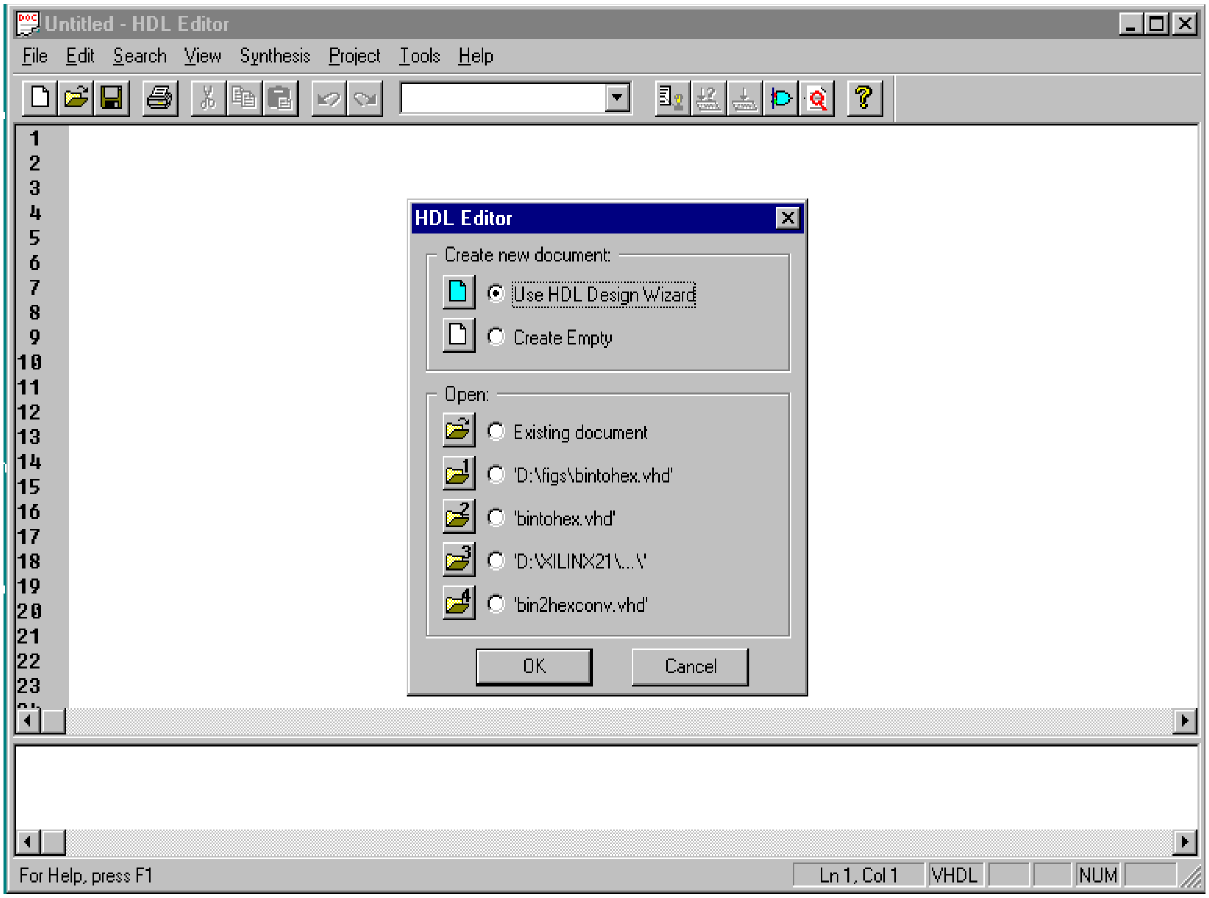
Figure 3.5 shows the **Project Manager** window with its associated HDL design cycle. It is almost the same as the schematic capture design cycle the difference being that the synthesis phase has been added. We will now create a VHDL file by using the HDL Editor. After creating the HDL file we will then add it to the project after which we will check its syntax.



```
Pcm   : Synopsys version: 3, 2, 0, 4201
Pcm   : ---------- Opening project: bintohex ----------
Pcm   : Design Type HDL
```

Figure 3.5: Project Manager Window for HDL Entered Designs

To enter the VHDL modeling information first click on the **HDL Editor** icon, on the **Design Entry** button under the **Flow** tab of the **Project Manager** window. This will invoke the **HDL Editor** and a window which will allow one to create a new HDL file or open an existing HDL file (see Figure 3.6). In this example, we will want to create a new HDL file. In this case, there are two options, **Create Empty** which will open an new text file for HDL source entering or launching the HDL **Design Wizard** which will aid in creating the VHDL code by producing templates which can be filled in by the user. To illustrate the utility of the **Design Wizard** we will choose this option as shown in Figure 3.6 for the binary to hexadecimal converter example. To do this we will make sure the **Design Wizard** option is checked and then press the **OK** button.



HDL Editor

Create new document:

● Use HDL Design Wizard

○ Create Empty

Open:

○ Existing document

○ 'D:\figs\bintohex.vhd'

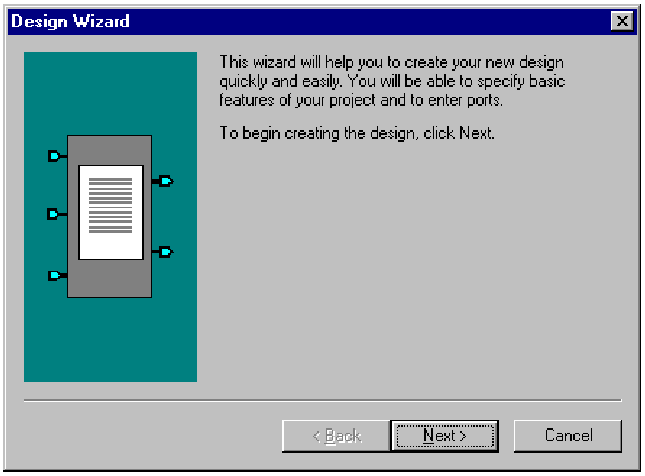
○ 'bintohex.vhd'

○ 'D:\XILINX21\...\'

○ 'bin2hexconv.vhd'

Figure 3.6: HDL Editor Window

The next window that will appear is shown in Figure 3.7. It simply describes the purpose of the **Design Wizard**. Press the **Next** button to continue.

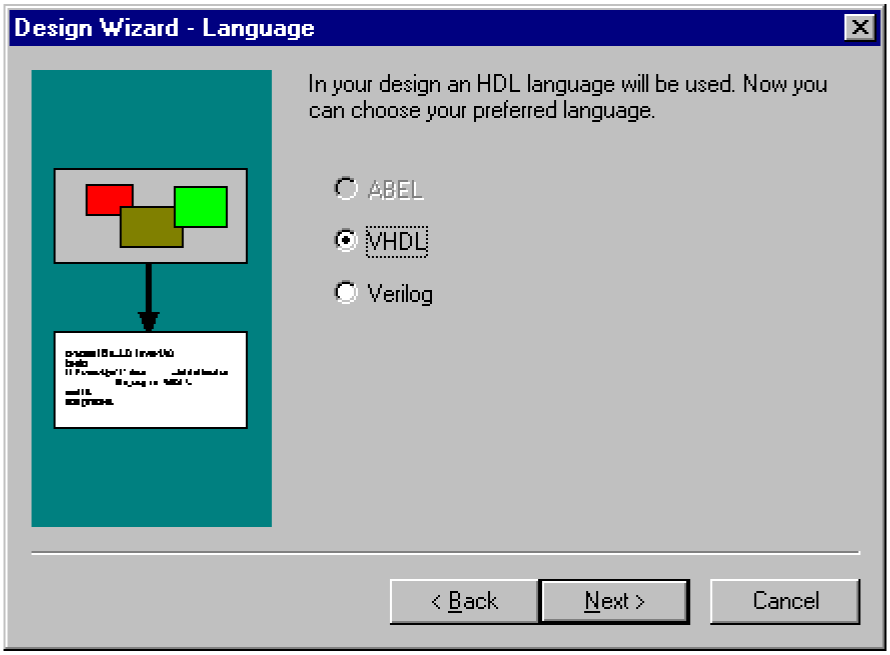

Design Wizard

This wizard will help you to create your new design quickly and easily. You will be able to specify basic features of your project and to enter ports.

To begin creating the design, click Next.

Figure 3.7: Design Wizard Getting Started Window

The HDL that is being used in this example is VHDL (Very High Speed Integrated Circuit Hardware Description Language [6]). It is one of the major HDLs currently being used by practicing engineers. Check the **VHDL** option an press the **Next** button.



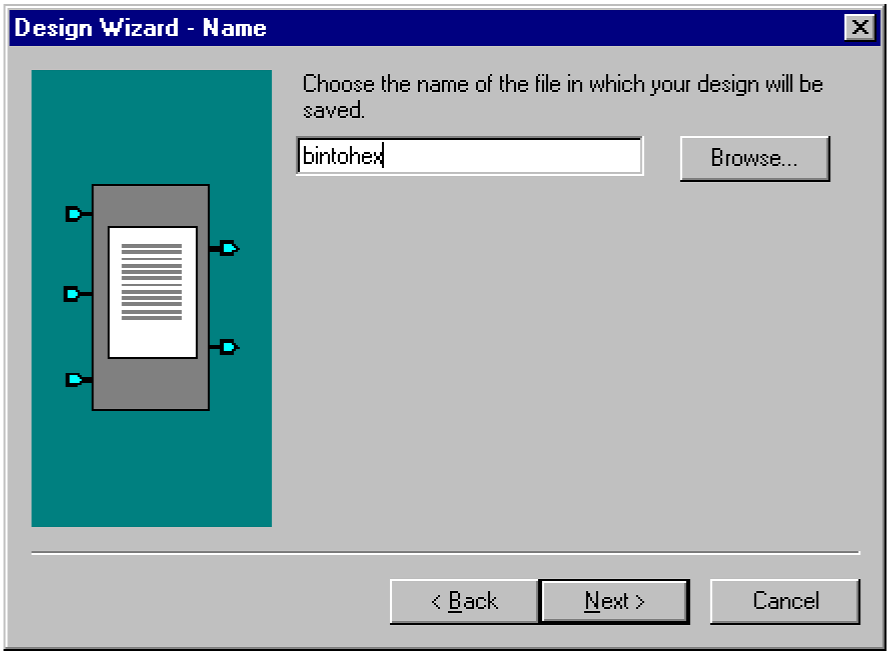
Design Wizard - Language

In your design an HDL language will be used. Now you can choose your preferred language.

○ ABEL
● VHDL
○ Verilog

Figure 3.8: HDL Selection Window

Then enter the name of the VHDL model file. For this example, enter <u>bintohex</u> and press the **Next** button.



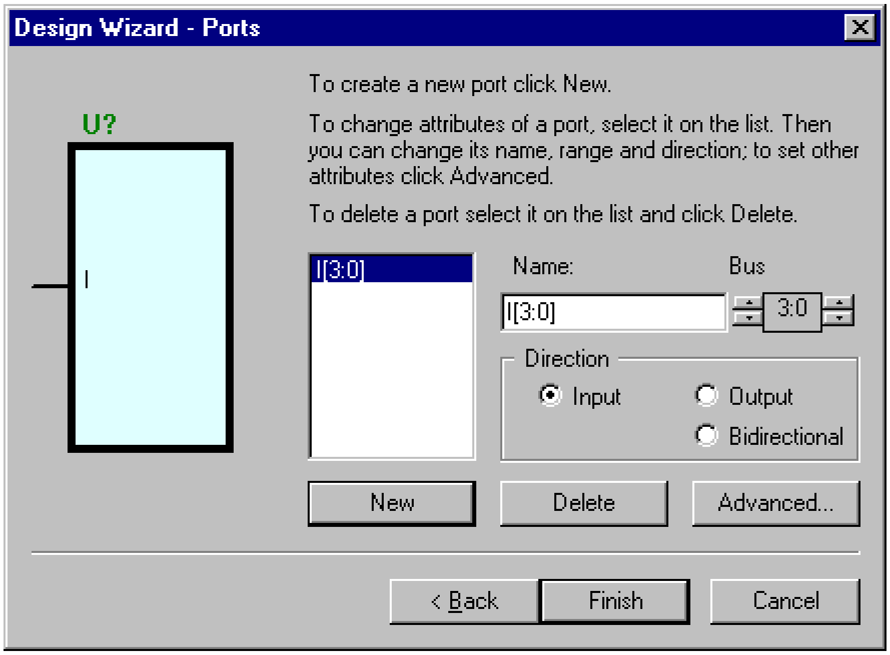
Design Wizard - Name

Choose the name of the file in which your design will be saved.

bintohex

Figure 3.9: Design Wizard Name Entry Window

The **Design Wizard -- Ports** window will now appear as shown in Figure 3.10. As directed, press the **New** button to create a new input port. 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 **Name:** field. Then set the range of the **Bus** field (using the up and down arrows) to be 3:0 as shown in Figure 3.10. After making sure the **Direction** of the bus is an **Input** (its default) press **New** again to go to the next step which is to enter the output bus.


Design Wizard - Ports

To create a new port click New.

To change attributes of a port, select it on the list. Then you can change its name, range and direction; to set other attributes click Advanced.

To delete a port select it on the list and click Delete.

I[3:0]

Name: I[3:0]

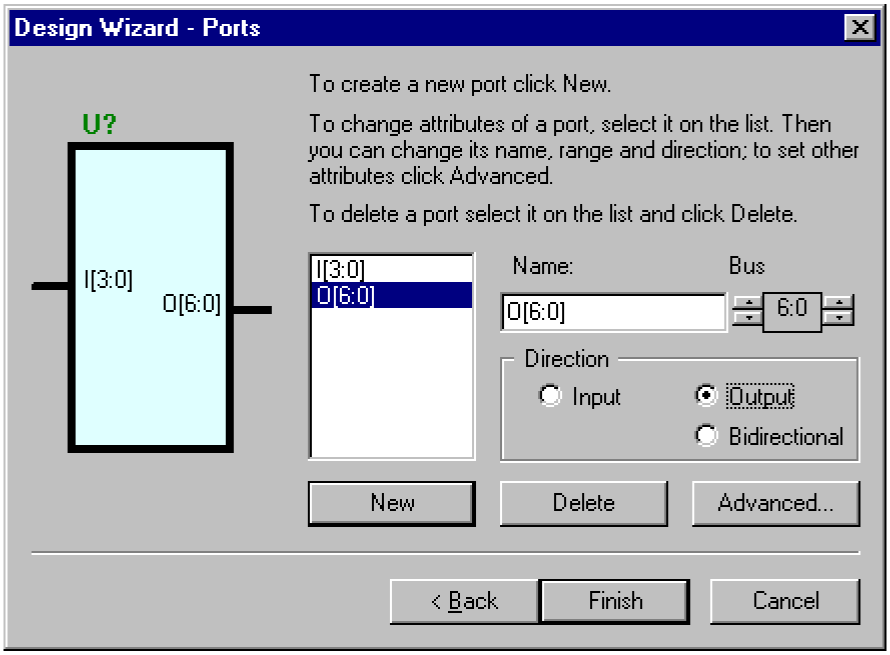
Bus: 3:0

Direction

● Input ○ Output ○ Bidirectional

Figure 3.10: Design Wizard I/O Port Entry Window-- (Input Port Entry)

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 **Name** field, setting the range to 6:0 in the **Bus** field, selecting **Output** in the **Direction** field and then pressing the **Finish** button.



Design Wizard - Ports

To create a new port click New.

To change attributes of a port, select it on the list. Then you can change its name, range and direction; to set other attributes click Advanced.

To delete a port select it on the list and click Delete.

I[3:0]
O[6:0]

Name: O[6:0]

Bus 6:0

Direction

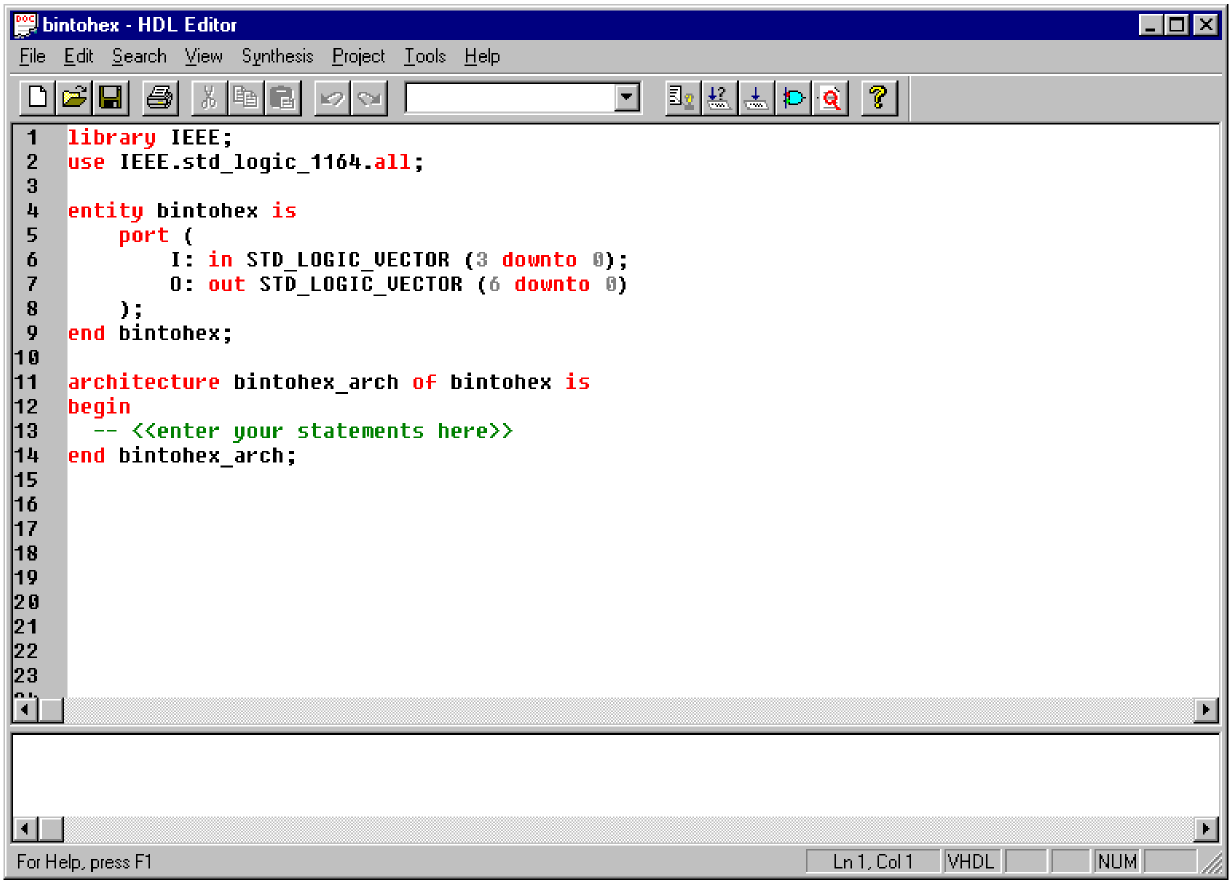
○ Input ● Output
○ Bidirectional

Figure 3.11: Design Wizard I/O Port Entry Window-- (Output Port Entry)

After completing the interfacing details, the **HDL Editor** window (Figure 3.12) 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 **Design Wizard** where it describes the interface between the design being modeled and the outside world. The **Design 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. Most of these statements can be entered into the VHDL skeleton file at the point where the comment has been inserted which states:

```
-- <<enter your statements here >>
```

It is at that point that the user is to enter the behavioral or structural VHDL code which describes the logical operation of the design. The Xilinx CAD tool allows the user to utilize a built-in Language Assistant to aid in this process. The VHDL template file which was created for the binary to hexadecimal example is shown in Figure 3.12.



```
library IEEE;
use IEEE.std_logic_1164.all;

entity bintohex is
    port (
        I: in STD_LOGIC_VECTOR (3 downto 0);
        O: out STD_LOGIC_VECTOR (6 downto 0)
    );
end bintohex;

architecture bintohex_arch of bintohex is
begin
  -- <<enter your statements here>>
end bintohex_arch;
```

Figure 3.12: Design Wizard Created HDL Template for bintohex Example

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 **Design Wizard** is complete. 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>* -- *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>* -- *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.

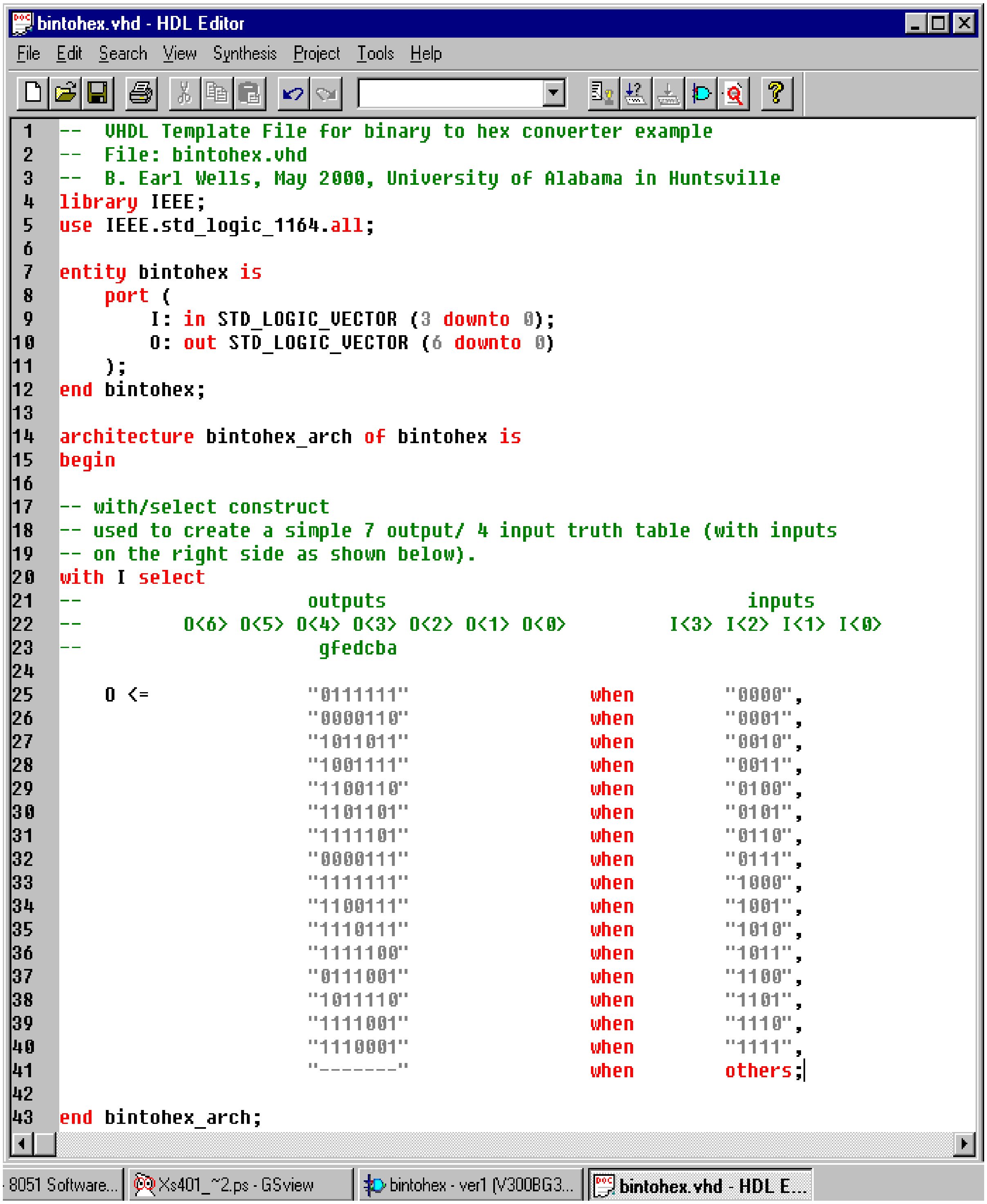
The desired behavior of the *Entity* section is modeled in the *Architecture* section. This section begins with the keyword **architecture** which is followed by an arbitrarily defined name for the architecture (in this case the **Design Wizard** choose to use, *bintohex_arch*) which is paired with the identity of the *Entity* using the **of** keyword followed by the *Entity* name (i.e. in this case *of bintohex*). 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 3.12, 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 at the point indicated by the comment --*<<enter your statements here >>* which was inserted by the **Design Wizard**. VHDL is very rich in constructs that allow one to model digital logic using a wide range of styles.

Figure 3.13 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 **Design 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.



```
--  VHDL Template File for binary to hex converter example
--  File: bintohex.vhd
--  B. Earl Wells, May 2000, University of Alabama in Huntsville
library IEEE;
use IEEE.std_logic_1164.all;

entity bintohex is
    port (
        I: in STD_LOGIC_VECTOR (3 downto 0);
        O: out STD_LOGIC_VECTOR (6 downto 0)
    );
end bintohex;

architecture bintohex_arch of bintohex is
begin

-- with/select construct
-- used to create a simple 7 output/ 4 input truth table (with inputs
-- on the right side as shown below).
with I select
--                    outputs                                inputs
--         O<6> O<5> O<4> O<3> O<2> O<1> O<0>        I<3> I<2> I<1> I<0>
--                     gfedcba

    O <=              "0111111"                when        "0000",
                      "0000110"                when        "0001",
                      "1011011"                when        "0010",
                      "1001111"                when        "0011",
                      "1100110"                when        "0100",
                      "1101101"                when        "0101",
                      "1111101"                when        "0110",
                      "0000111"                when        "0111",
                      "1111111"                when        "1000",
                      "1100111"                when        "1001",
                      "1110111"                when        "1010",
                      "1111100"                when        "1011",
                      "0111001"                when        "1100",
                      "1011110"                when        "1101",
                      "1111001"                when        "1110",
                      "1110001"                when        "1111",
                      "-------"                when        others;

end bintohex_arch;
```

Figure 3.13: Final HDL Model for bintohex Example

[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 Assistant Tool**. To use the **Language Assistant**, place the cursor at "**-- <<enter your statements here>>** "in the VHDL template file for the design. Then, from the pull-down menu, select **Tools**, then click on the **Language Assistant** option. The **Language Assistant** window as shown in Figure 3.14 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.]]

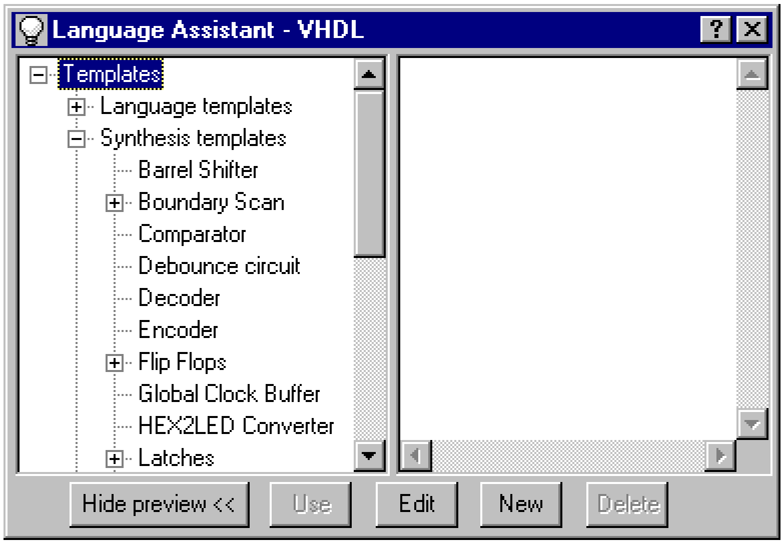
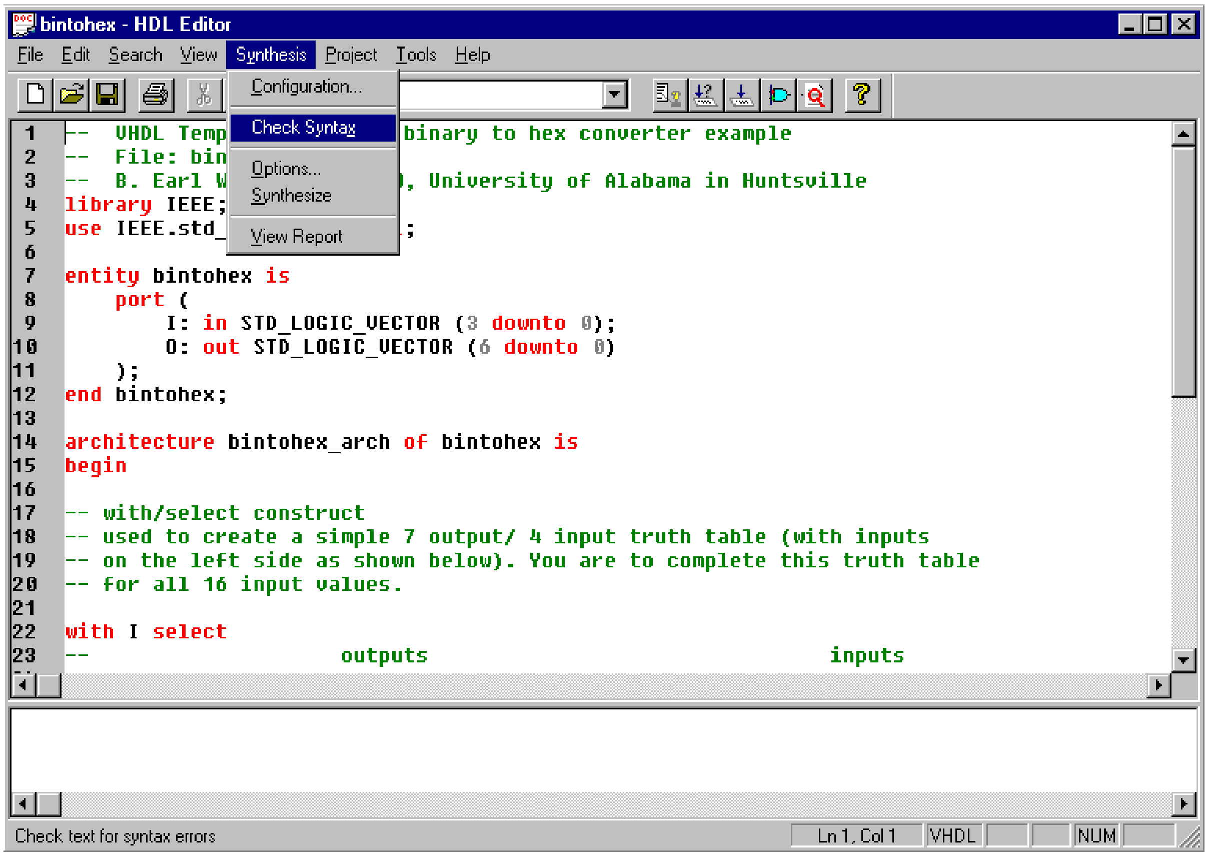


Figure 3.14: Language Assistant Window (VHDL)

After the VHDL model has been entered, one needs to check the syntactical correctness of the modeling code. This can be done from the **HDL Editor** by using the pull-down window obtained by first selecting the **Synthesis** option on the HDL toolbar and then clicking on the **Check Syntax** option as shown in Figure 3.15. the same analysis can be done from within the **Project Manager** window by first selecting the **Synthesis** option, and then selecting the **Analyze All Sources** option.

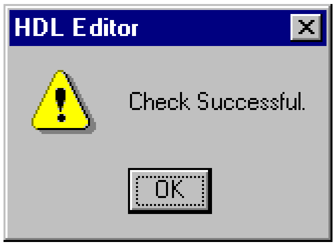


```
--  VHDL Temp            binary to hex converter example
--  File: bin
--  B. Earl W            ), University of Alabama in Huntsville
library IEEE;
use IEEE.std_             ;

entity bintohex is
    port (
        I: in STD_LOGIC_VECTOR (3 downto 0);
        O: out STD_LOGIC_VECTOR (6 downto 0)
    );
end bintohex;

architecture bintohex_arch of bintohex is
begin

-- with/select construct
-- used to create a simple 7 output/ 4 input truth table (with inputs
-- on the left side as shown below). You are to complete this truth table
-- for all 16 input values.

with I select
--                     outputs                                inputs
```

Check text for syntax errors

Figure 3.15: Checking the Syntax of the VHDL Model

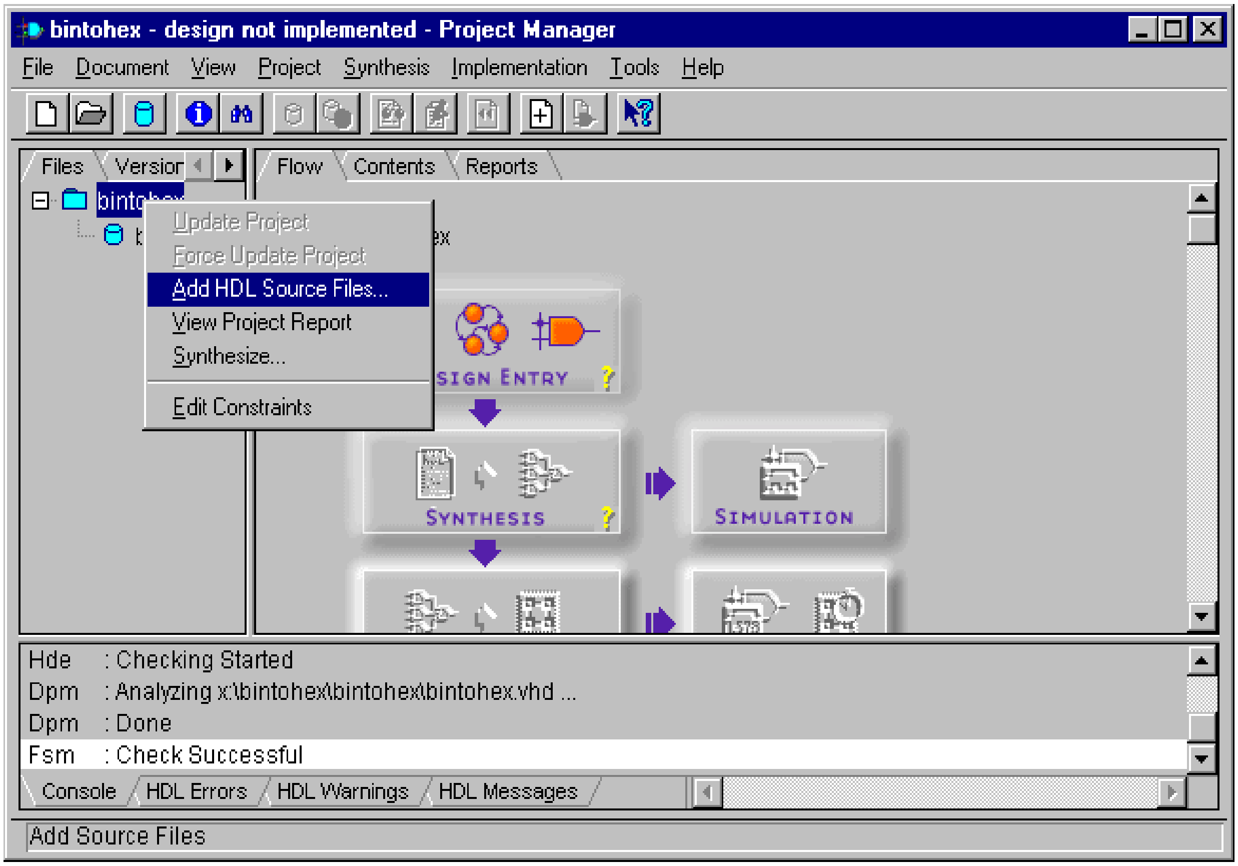
If there are no coding errors then the following status message should appear. To continue press the **Ok** button and return to the **Project Manager** window.



Check Successful.

Figure 3.16: Syntax Checking Status Message

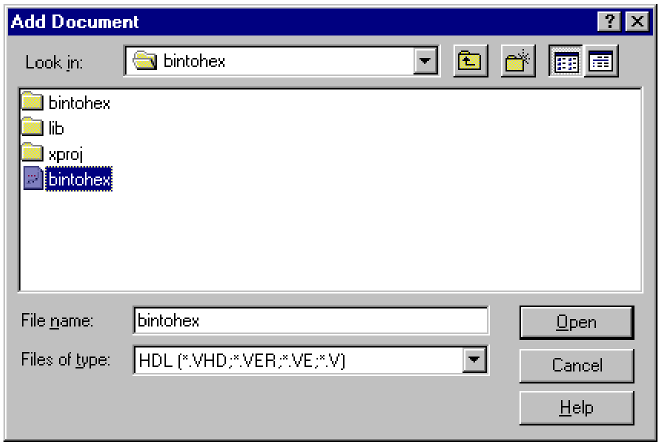
The next step after creating the VHDL file is to add it to the project. To do this for the binary to hexadecimal converter example from the **Project Manager** window, first move the cursor to point to the top hierarchy source tree located under the **Files** tab on the left-hand side of the window as shown in Figure 3.17. Click on the right most button and a pull down window will be displayed as shown in Figure 3.17. Then select the **Add HDL Source Files** option with the left mouse button.



```
Hde   : Checking Started
Dpm   : Analyzing x:\bintohex\bintohex\bintohex.vhd ...
Dpm   : Done
Fsm   : Check Successful
```

Figure 3.17: Project Manger Window: Adding HDL files to the Project

An **Add Document** window will appear as shown in Figure 3.18. For this example enter, **bintohex** in the **File name:** field, then press the **Open** button.

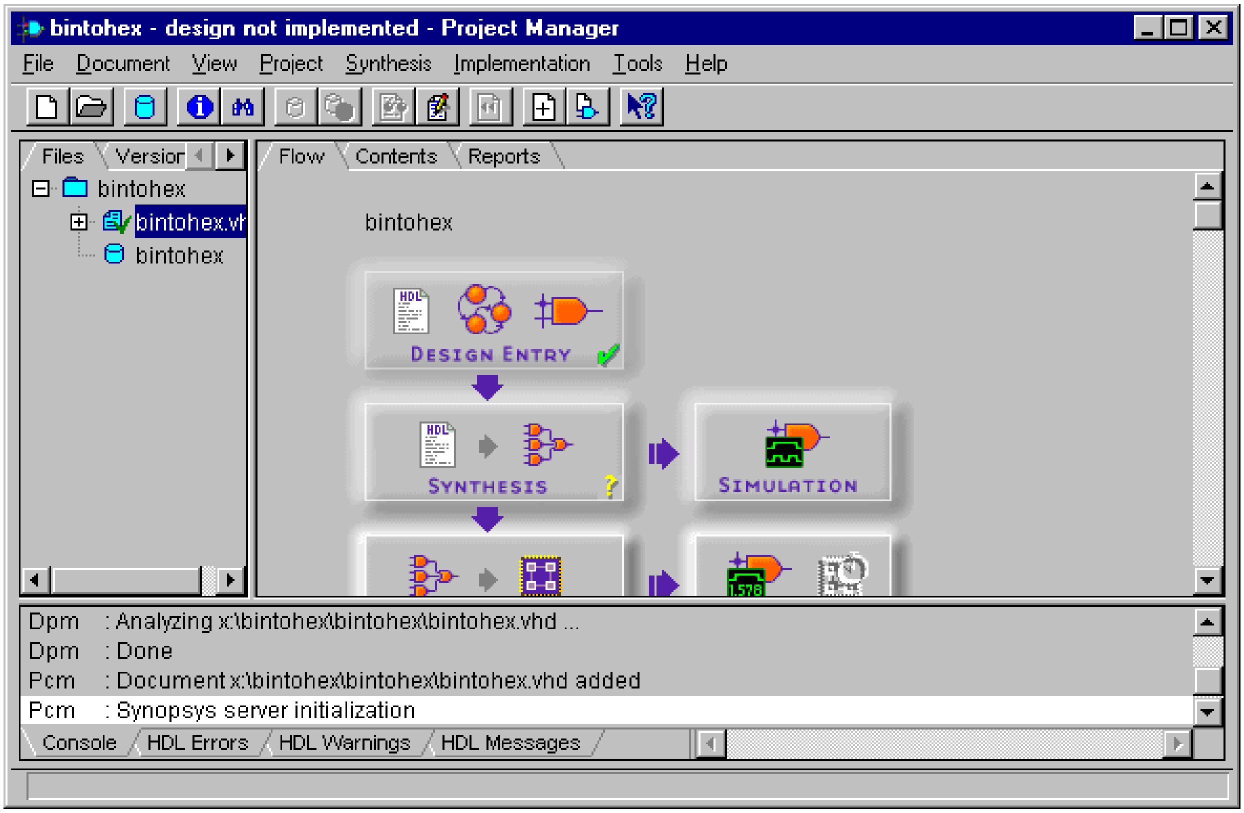


File name: bintohex

Files of type: HDL (*.VHD;*.VER;*.VE;*.V)

Figure 3.18: Add Document Window

The file ‘bintohex.vhd’ will appear under the **File** tab of the **Project Manager** window and a message will appear in the text area of this window indicating that this file has been successfully added to the project as shown in Figure 3.19.



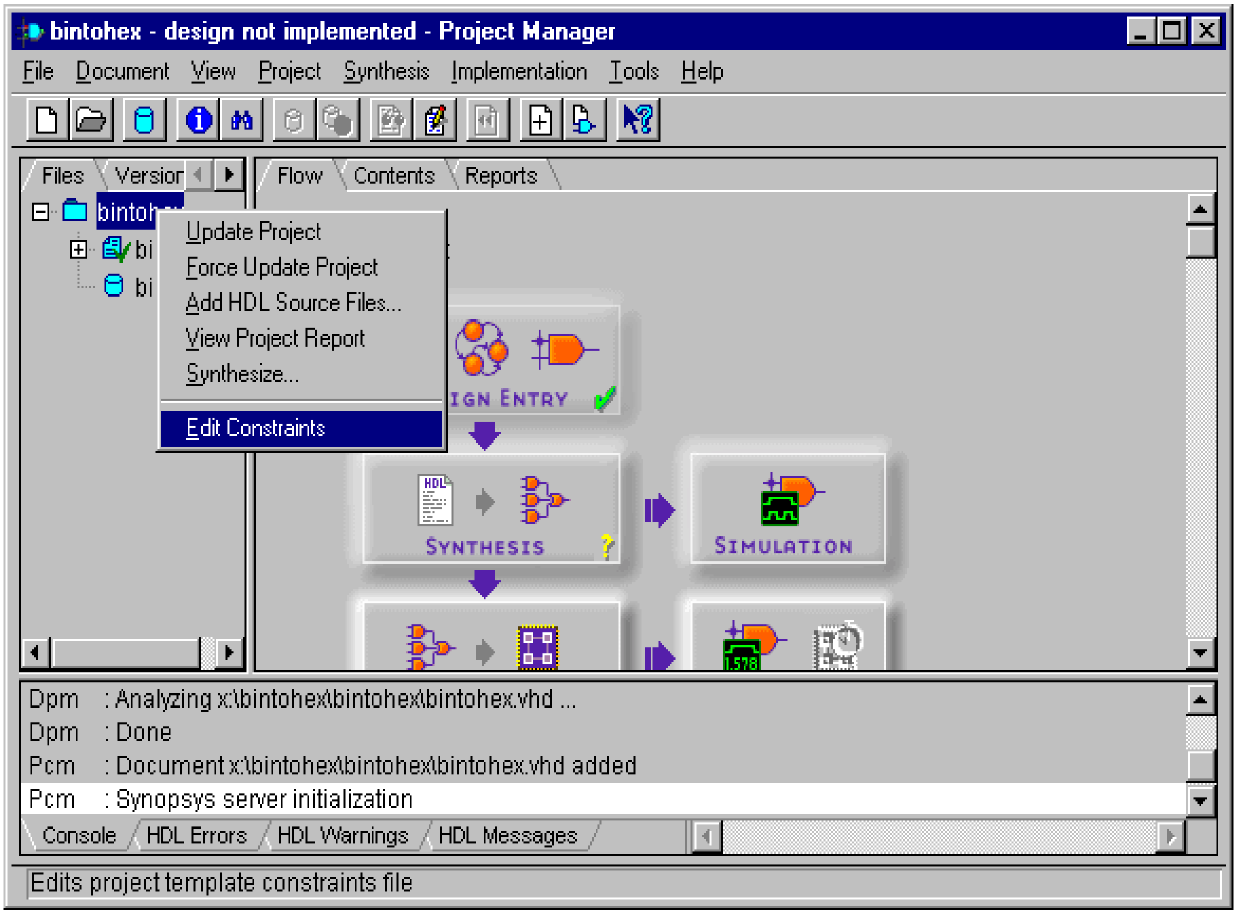
```
Dpm   : Analyzing x:\bintohex\bintohex\bintohex.vhd ...
Dpm   : Done
Pcm   : Document x:\bintohex\bintohex\bintohex.vhd added
Pcm   : Synopsys server initialization
```

Figure 3.19: Completed VHDL File Addition to

The next step will be to “lock” the VHDL signals to the user specified XC4010XL pins.

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 located external to XS 40 and assign the seven outputs to the jumper locations which will be used to drive the individual segments of the externally connected common cathode seven segment LED display. To lock a specified pin name on the XC4010XL FPGA with the logical I/O net name specified in the VHDL model do the following. From the **Project Manager** window, move the cursor to point to the top hierarchy source tree located under the **Files** tab on the left-hand side of the window. Click on right most button and a pull down window will appear. Then select the **Edit Constraints** option with the left mouse button.



```
Dpm   : Analyzing x:\bintohex\bintohex\bintohex.vhd ...
Dpm   : Done
Pcm   : Document x:\bintohex\bintohex\bintohex.vhd added
Pcm   : Synopsys server initialization
```

Edits project template constraints file

Figure 3.20: Entering User Constraints from the Project Manager

For this design the desired schematic name to Xilinx XC4010XL Pin name cross reference (i.e. Pin Locks) is shown in Table 3.2.

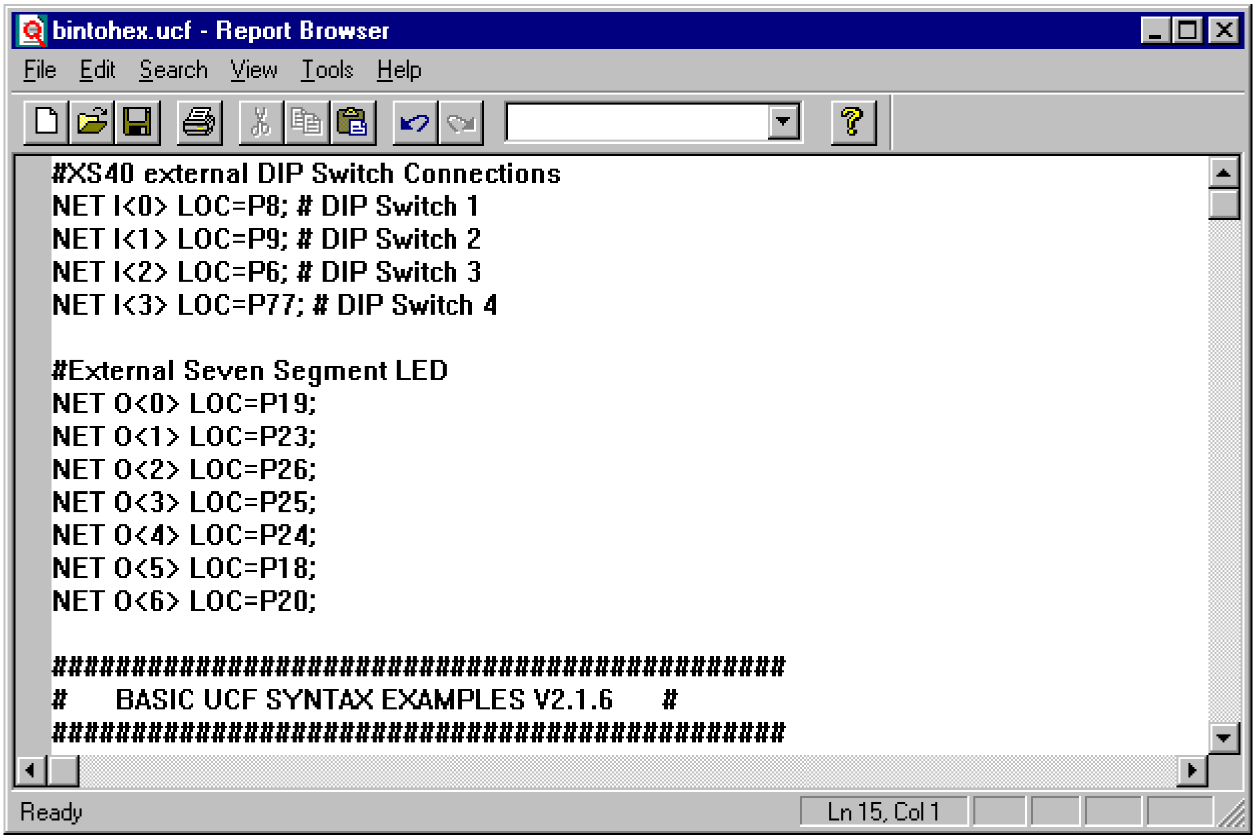
Table 3.2: Desired Pin Locking (Cross Reference) Configuration

I/O Pin Description	VHDL “Net” Name	Xilinx XC4010XL Pin Name
XS 40 DIP Switch #1	I<0>	8
XS 40 DIP Switch #2	I<1>	9
XS 40 DIP Switch #3	I<2>	6
XS 40 DIP Switch #4	I<3>	77
XC4010XL Pin 19, (segment ‘a’ of LED)	O<0>	19
XC4010XL Pin 23, (segment ‘b’ of LED)	O<1>	23
XC4010XL Pin 26, (segment ‘c’ of LED)	O<2>	26
XC4010XL Pin 25, (segment ‘d’ of LED)	O<3>	25
XC4010XL Pin 24, (segment ‘e’ of LED)	O<4>	24
XC4010XL Pin 18, (segment ‘f’ of LED)	O<5>	18
XC4010XL Pin 20, (segment ‘g’ of LED)	O<6>	20

The resulting additions to the **User Constraints File** are shown in Figure 3.21 where the cross reference described in Table 3.2 was implemented using the

NET <vhdl bus element name> LOC=<symbolic Xilinx pin name>

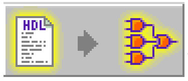
construct. After these additions have been made to the **User Constraints File** one should **Save** the work, **Exit** the Report Browser, and return to the **Project Manager** window

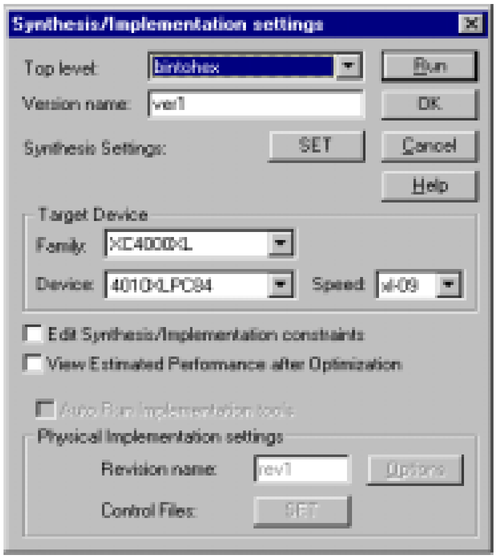

bintohex.ucf - Report Browser
File Edit Search View Tools Help
Ready
Ln 15, Col 1

```
#XS40 external DIP Switch Connections
NET I<0> LOC=P8; # DIP Switch 1
NET I<1> LOC=P9; # DIP Switch 2
NET I<2> LOC=P6; # DIP Switch 3
NET I<3> LOC=P77; # DIP Switch 4

#External Seven Segment LED
NET O<0> LOC=P19;
NET O<1> LOC=P23;
NET O<2> LOC=P26;
NET O<3> LOC=P25;
NET O<4> LOC=P24;
NET O<5> LOC=P18;
NET O<6> LOC=P20;

################################################
#     BASIC UCF SYNTAX EXAMPLES V2.1.6      #
################################################
```

Figure 3.21: Adding Pin Locking Information to the Project Constraint File

Logic Synthesis Phase

When all the sources are successfully analyzed it is now time to synthesize the design. Synthesis is an automatic method of converting a higher level of abstraction (VHDL code in this case) to a lower level of abstraction (primitive-level netlist). To do this, click on the **Synthesis** button, , under the **Flow** tab of the **Project Manager** window. A **Synthesis/Implementation settings** window will appear as shown in Figure 3.22. All default settings on this window should be as indicated on the figure. Click on the **Run** button and the synthesis process will start.



Top level: bintohex

Version name: ver1

Synthesis Settings:

Target Device

Family: XC4000XL

Device: 4010XLPC84

Speed: xl-09

□ Edit Synthesis/Implementation constraints

□ View Estimated Performance after Optimization

□ Auto Run Implementation tools

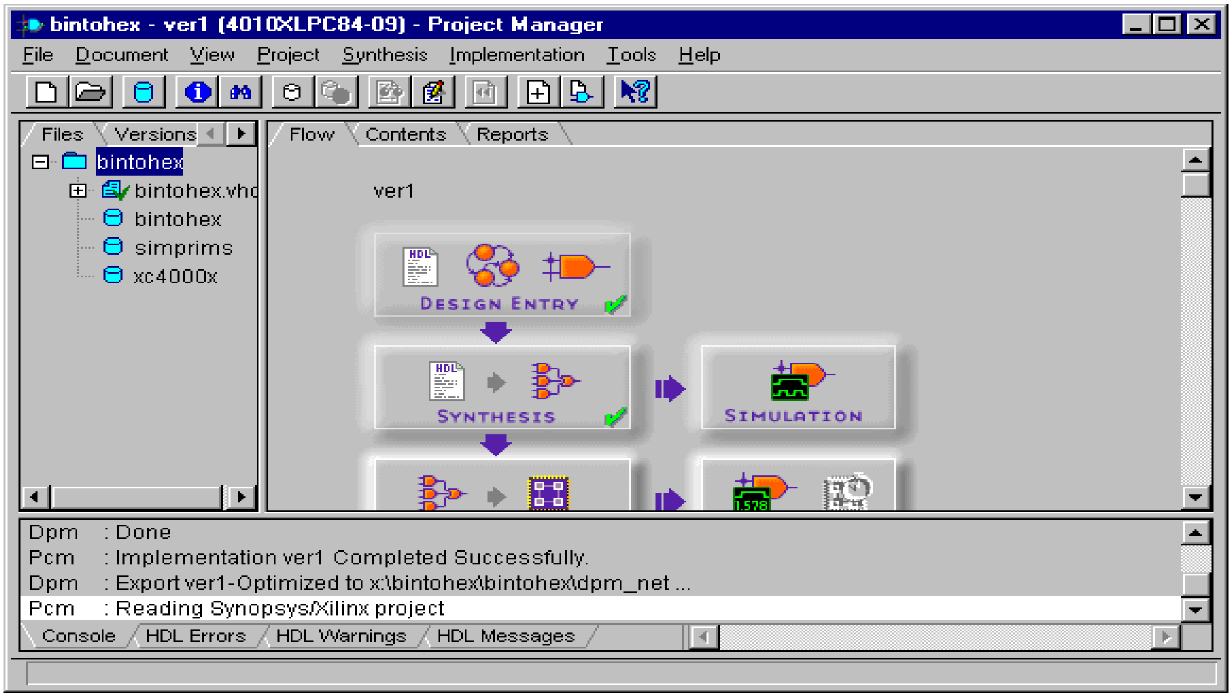
Physical Implementation settings

Revision name: rev1

Control Files:

Figure 3.22: Synthesis/Implementation Settings Window

When the project is successfully synthesized a small check mark will appear on the **Synthesis** button of the **Project Manager** window as shown in Figure 3.23 and the Pre-Synthesis and Post-Synthesis reports will be produced. They can be found at the **Report Tab** at **Project Manager** window.



```
Dpm   : Done
Pcm   : Implementation ver1 Completed Successfully.
Dpm   : Export ver1-Optimized to x:\bintohex\bintohex\dpm_net ...
Pcm   : Reading Synopsys/Xilinx project
```

Figure 3.23: Project Manager Window after Synthesis is Completed

Functional Simulation Phase

To perform a functional simulation of the binary to hexadecimal converter example, proceed in the manner described in the previous chapter. First, click on the **Functional Simulation** button, on the **Project Manager** window. This should launch the **Logic Simulator** window. Then choose the set of signals which one wants to observe during the simulation. For the bintohex design these signals include the input bus **I** and the output bus **O**. To select these nets/ busses click on **Select Components** button, . Then at the **Select Signals** windows, choose the desired signals to be included in the simulation. Do this by clicking each signal with the left mouse button. Then to select more than one signal, hold on the **<ctrl>**-key and click on the signals with left mouse button. Figure 3.24 shows the case where the **I3:0**, and the **O6:0** signals have all been highlighted using this method. To complete the process, press the **Add** button followed by the **Close** button.

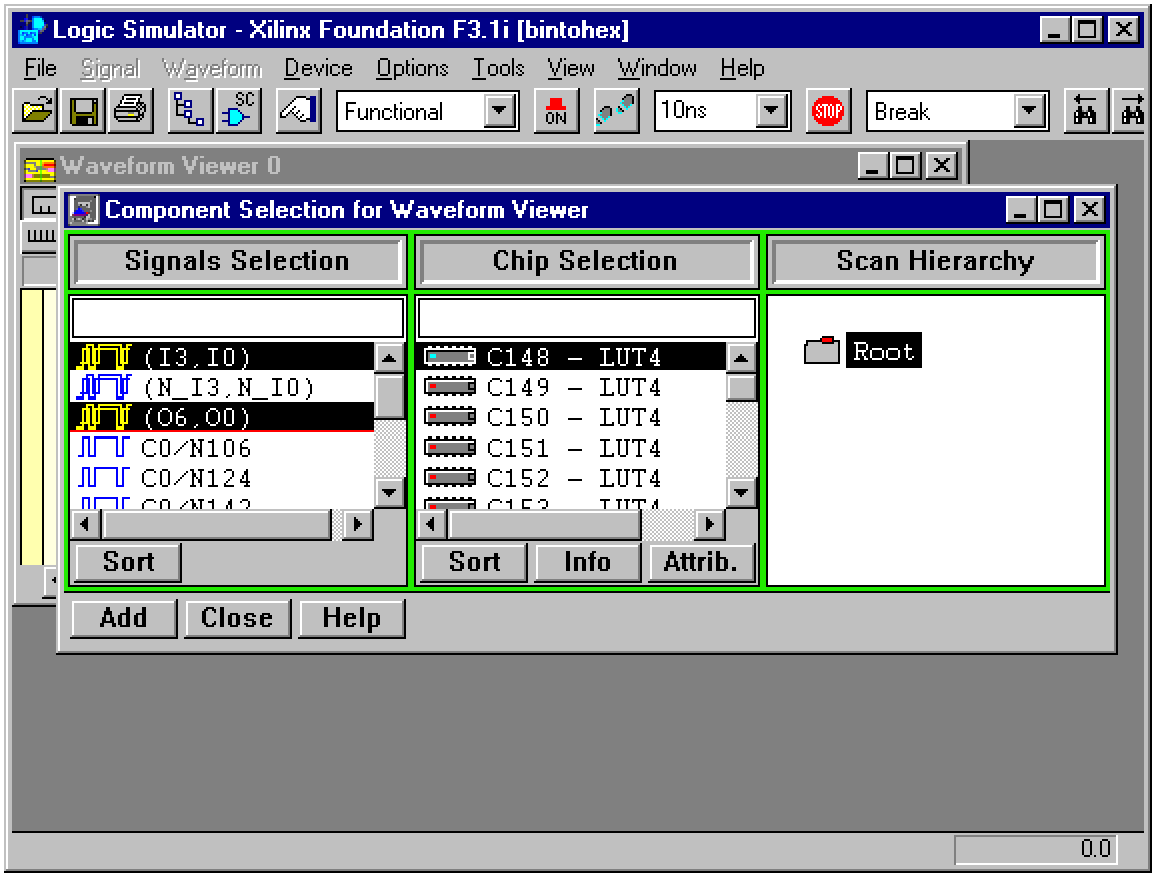


Figure 3.24: Signal Selection Window

Figure 3.24 shows the state of the **Logic Simulator** window after the input bus, **I3:0** and the output bus **O6:0** have been added to the observation list. They appear on the left hand side of the window. To start the simulation, we must of course define the stimulus pattern for all of the external inputs to the design. In the binary to hexadecimal converter design we would like to have a four bit stimulus pattern that exhaustively goes through all 16 possible input combinations. Fortunately, the Xilinx simulation software has a predefined binary counter stimulus generator which can be used for this purpose. To prepare the input bus so that this stimulus generator can be applied, one should first “flatten” the bus so that each of the four binary signals which make up the bus can be accessed separately. To do this one should first highlight the **I(3:0)** bus, on the right hand side of the **Logic Simulator** window, then click on **Signal** on the tool bar, followed by the **Bus** option, and finally the **Flatten** option, as shown in Figure 3.25.

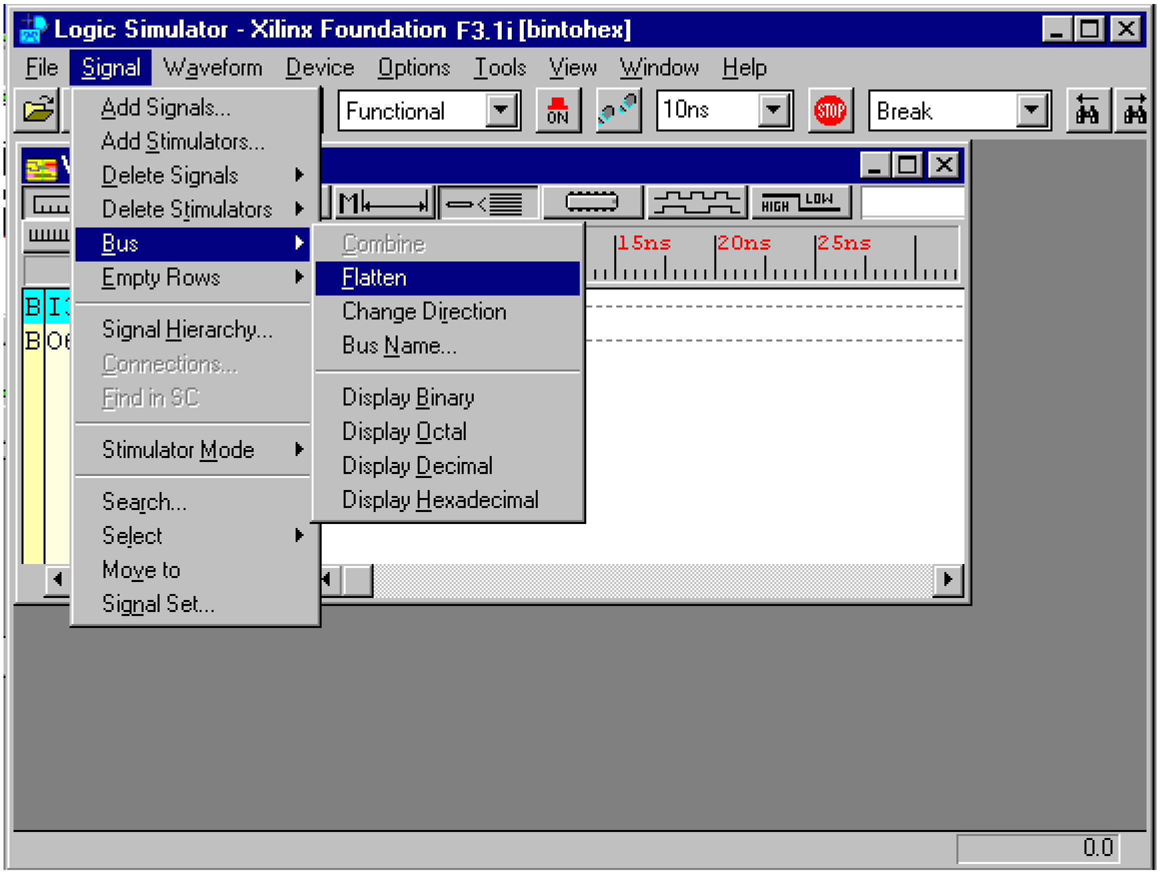


Figure 3.25: Flattening out the Input Bus in the Logic Simulator Window

This will separate the bus into the individual signals, I3, I2, I1, and I0 as shown in Figure 3.26. The next step is to assign the lower four bits of the built-in binary counter stimulus generator to each of the four individual signals.

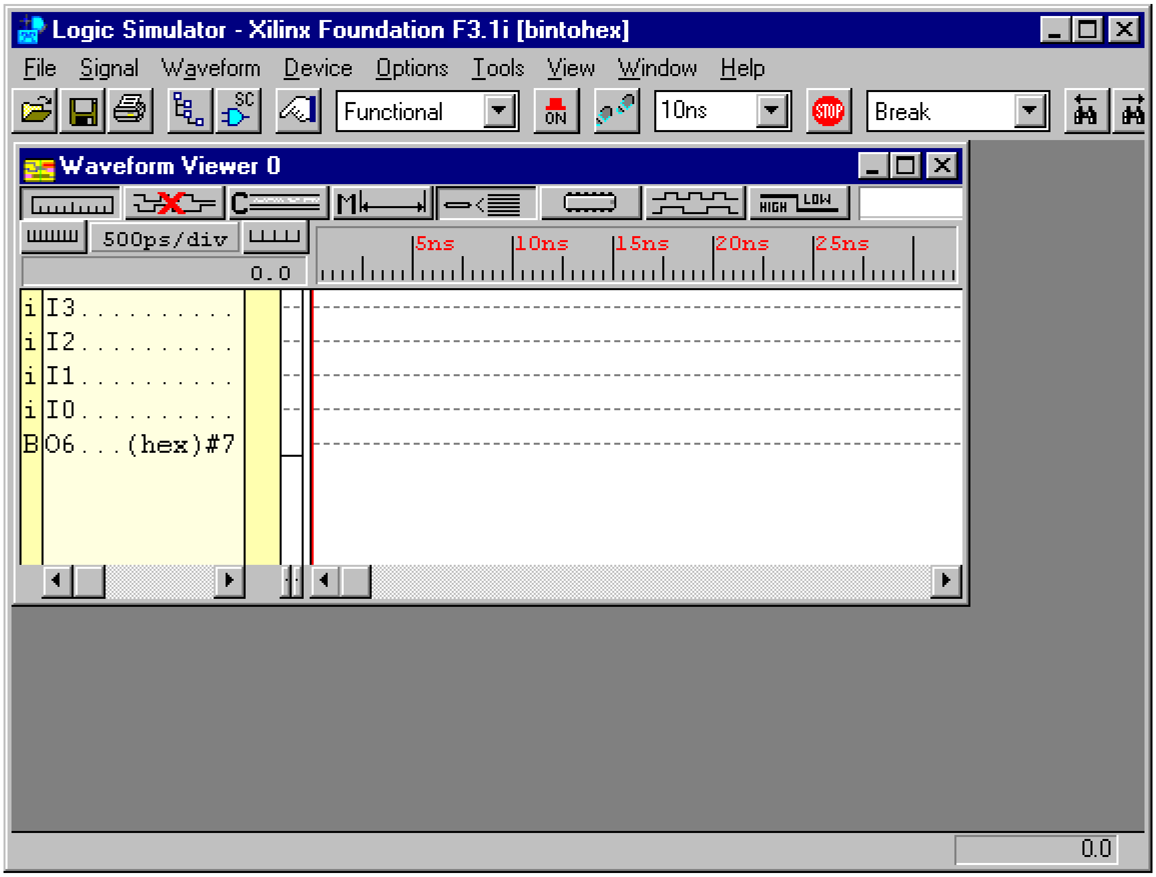


Figure 3.26: Flattened I Bus Representation

To define the desired clock stimulus waveform for the I3, I2, I1, and I0 signals, click on the **Select Stimulators** button, ⎍⎍⎍. The **Stimulator Selection** window will appear as shown in Figure 3.27. This is the same window in the previous chapter where we defined the periodic clock signal. This time we are going to assign the individual stimulus binary counter bits to the input signals and then define the period of the counter. The row of sixteen circles labeled **BC:** on the **Stimulator Selection** window represents in a bit-by-bit manner each bit of the built-in binary counter stimulus generator (most significant to least significant bit, being assigned from left to right as labeled in the window). Now we are ready to assign the lower four bits of the built-in binary counter stimulus generator to the four input signals of the binary to hexadecimal converter design. We will do this on a bit-by-bit basis starting with the **I0** signal and continuing on through the **I3** signal. To assign the IO signal to the least significant bit of the stimulus counter, it is important to first make sure that both the **Logic Simulator** and **Stimulator Selection** windows appear on the screen at the same time. Then single click on the **I0** signal on **Logic Simulator** window, move the cursor over to the **Stimulator Selection** window and click on the right most circle in the **BC**: row which is labeled **0**. This will cause the **B0** value (binary counter bit 0) to appear along side the **I0** signal name in the **Logic Simulator** window as shown in Figure 3.27. Next apply the same procedure to signal **I1** but this time select the circle which one position to the left of the one previously selected. Continue this process until all four signals have been selected as shown if Figure 3.27. To exit the **Stimulator Selection** window press the **Close** button. The source of the stimulus has now been defined, but the frequency of the binary stimulus counter needs to be defined.

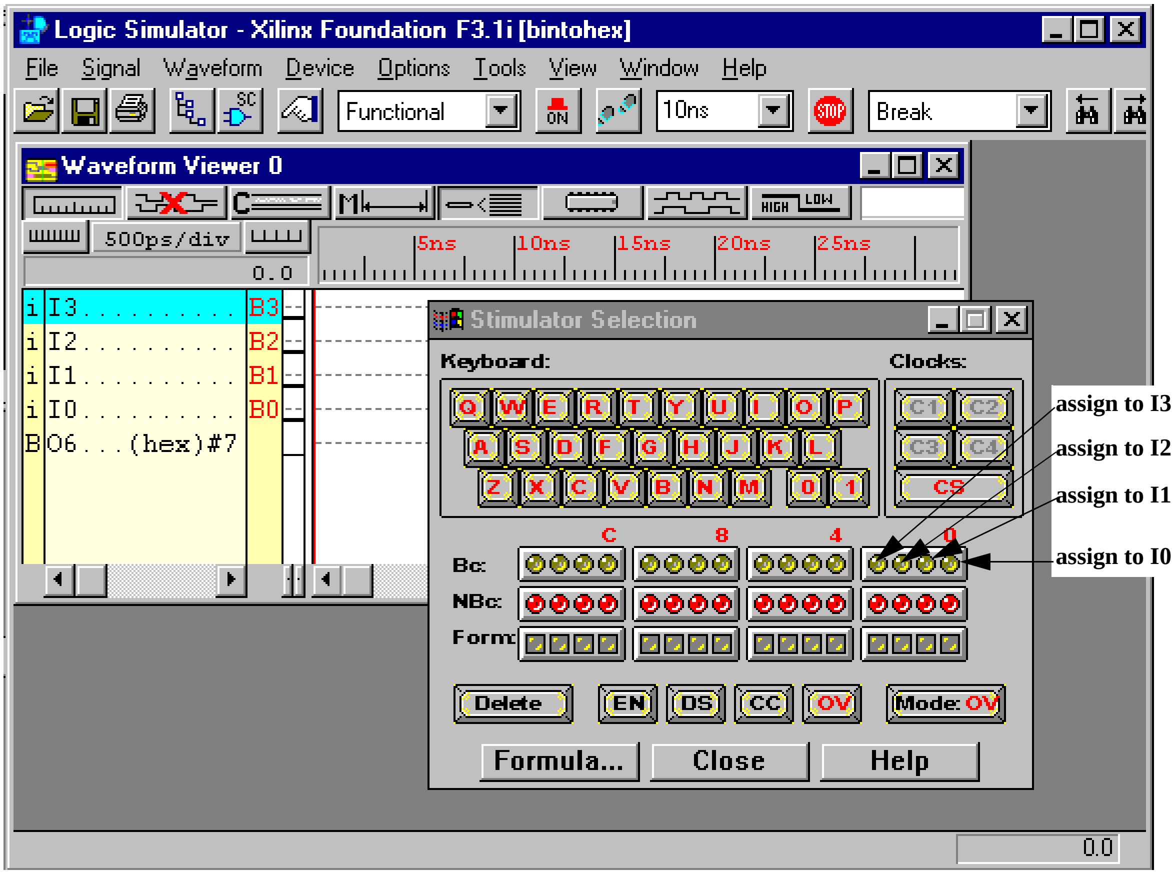
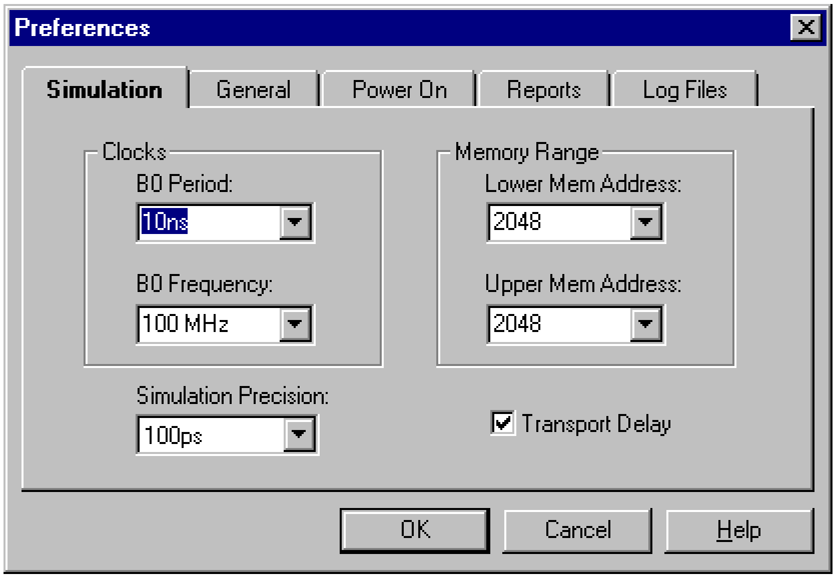


Figure 3.27: Assigning the Built-in Binary Counter to Provide Stimulus to the I bus

To assign a period or frequency to the stimulus binary counter first click on **Options** on the **Logic Simulator** tool bar, and then select **Preferences**. A window labeled **Preferences** will appear as shown in Figure 3.28. Select the **Simulation** tab from this window and in the **Clocks** area go to the **B0 Period:** field. This is where the binary counter's period or frequency can be defined. Then enter a value of <u>10ns</u> in the B) Period: field and press the **OK** button.



Clocks

B0 Period: 10ns

B0 Frequency: 100 MHz

Memory Range

Lower Mem Address: 2048

Upper Mem Address: 2048

Simulation Precision: 100ps

☑ Transport Delay

Figure 3.28: Setting the Period of the Binary Counter

Now that the stimulus is defined for the "bintohex" example we can start the simulation. It may be more convenient though to display the four bit **I** bus as a single four-bit vector. This can be accomplished by recombining the four elements into a single bus. To do this, first select the four individual signals **I3**, **I2**, **I1**, **I0**. Then select the **Signal** option from the **Logic Simulator** window and select **Bus**, followed by **Combine** as shown in Figure 3.29. This should result in the **I** bus appearing in the Logic Simulator window as it did before we flattened the bus to add the binary stimulus counter. The bus direction, however, may be displayed in a reverse manner from what was intended (i.e. **I0** may be displayed as the most significant bit and **I3** the least significant bit, etc.). If this is the case the direction of the bus should be changed.

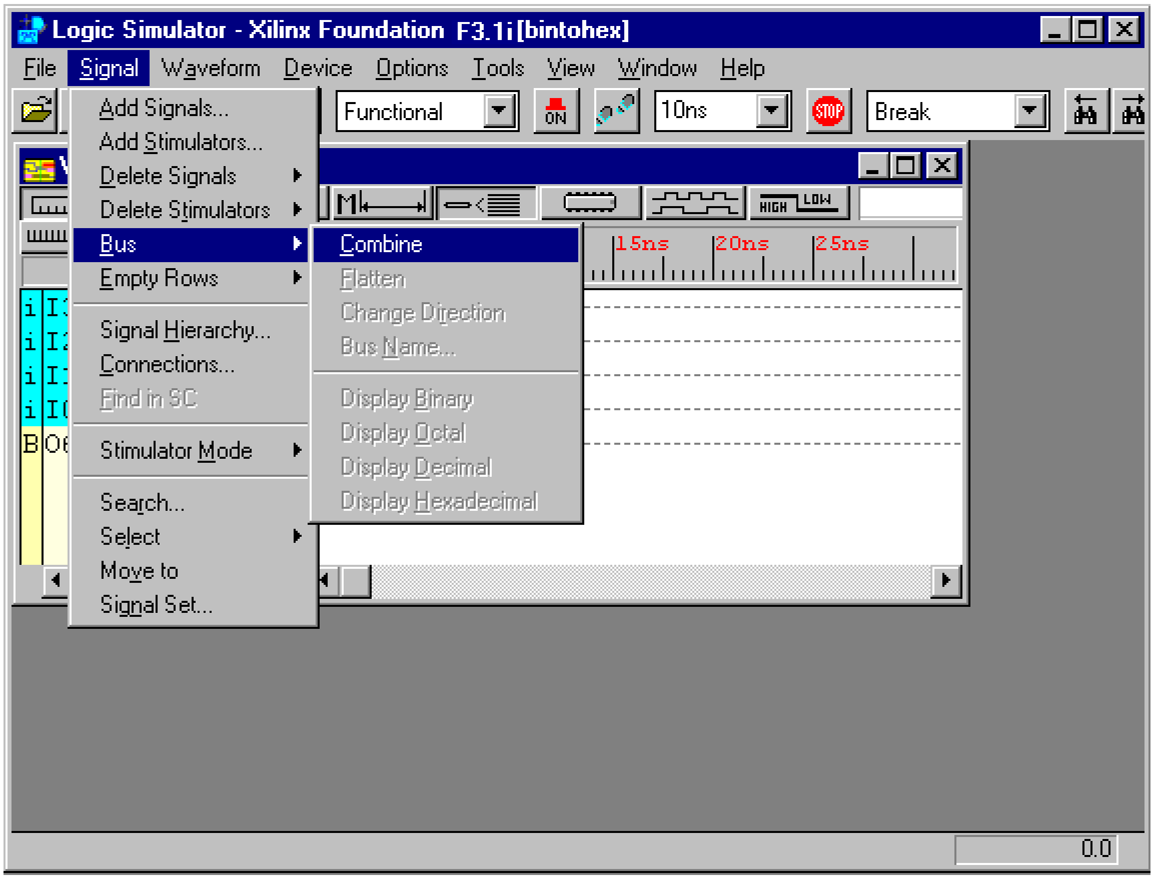


Figure 3.29: Returning the Display of the I Bus signals back to a Bus Representation

To change the direction of the bus first highlight the **I** bus from within the **Logic Simulator** window then select the **Signal** option from the tool bar area, then select **Bus**, and choose the **Change Direction** option as shown in Figure 3.30.

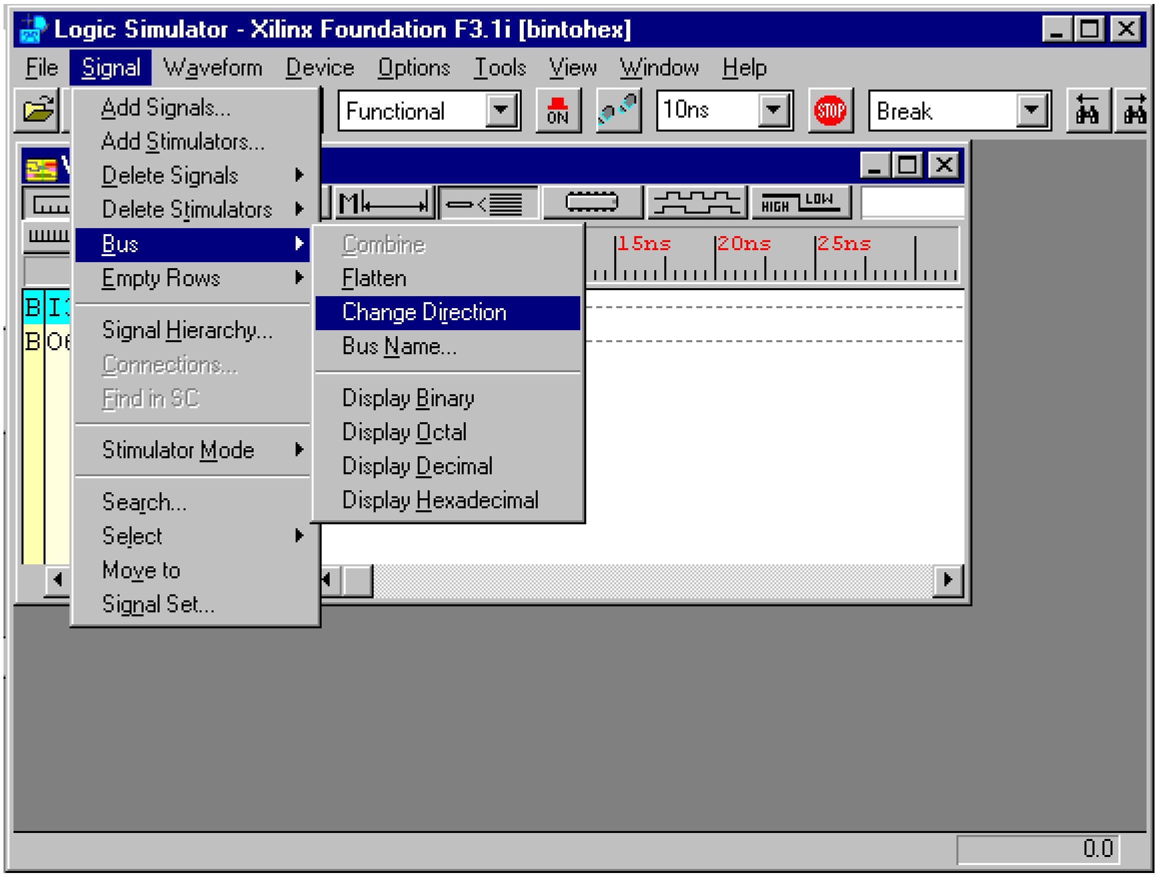


Figure 3.30: Changing back the direction of the I Bus Representation

Before simulating the binary to hexadecimal converter example one should first make sure that the display sheet time scale and the simulation step size are set to appropriate values to accommodate the 10ns clock cycle of the binary counter stimulus. This is accomplished by clicking on the increase time scale, , and decrease time scale buttons, , on the **Logic Simulation** window and then entering the simulation step size in the simulation step size dialog box. To perform the simulation in a step by step manner, click on the **Simulation Step** button, , as many times as is necessary to observe the selected signals. Figure 3.31 shows the waveforms that occur during the first 40nS of an example simulation.

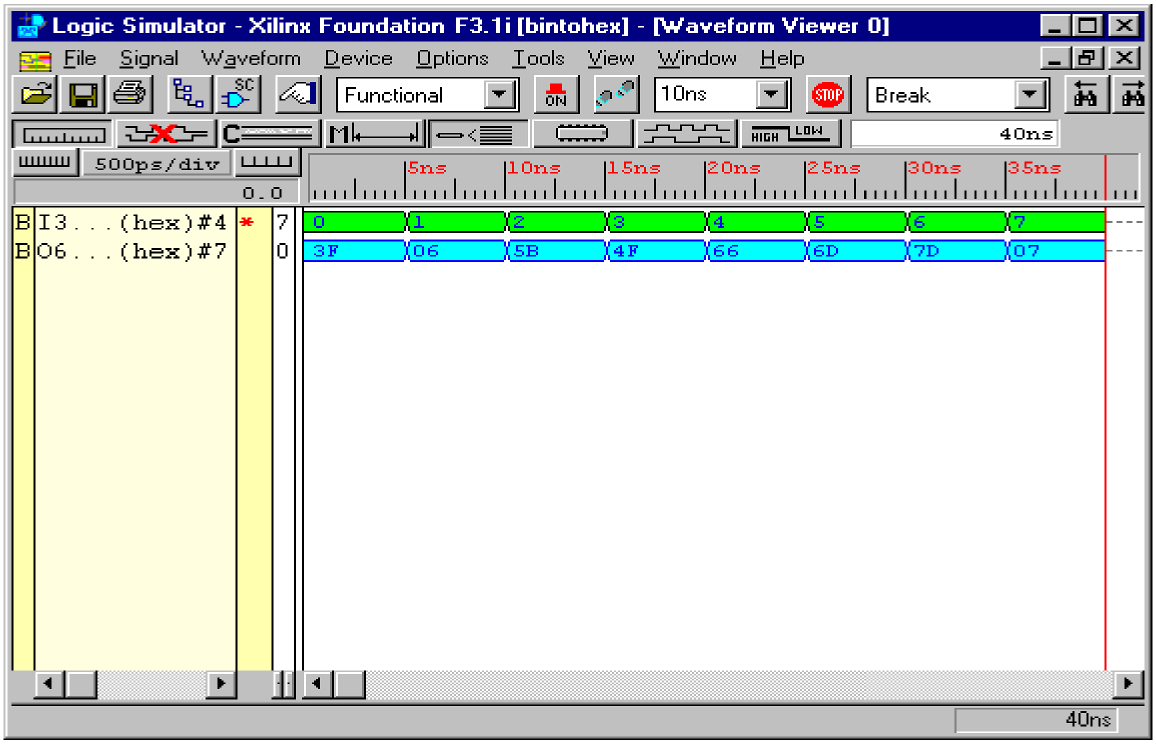
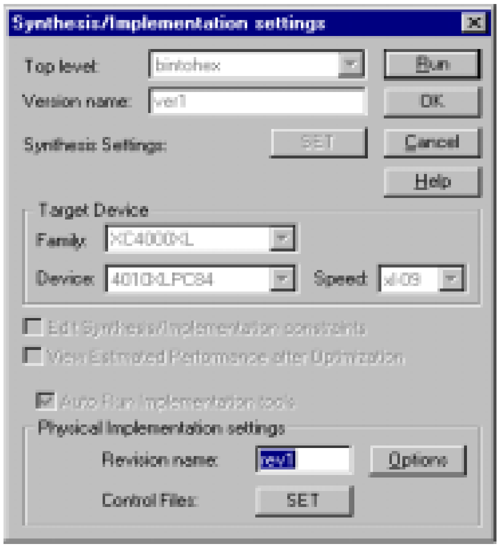


Figure 3.31: Example Simulation

Implementation Phase

To compile the binary to hexadecimal converter example from the **Flow** tab of the **Project Manager** window, click on the **Implementation** button, . The **Synthesis/Implementation Settings** window should appear as shown in Figure 3.32. All the parameters in the dialog boxes should be correct, so click on the **Run** button.



Synthesis/Implementation settings

Top level: bintohex

Version name: ver1

Synthesis Settings: SET

Target Device

Family: XC4000XL

Device: 4010XLPC84

Speed: xl-09

□ Edit Synthesis/Implementation constraints

□ View Estimated Performance after Optimization

☑ Auto Run Implementation tools

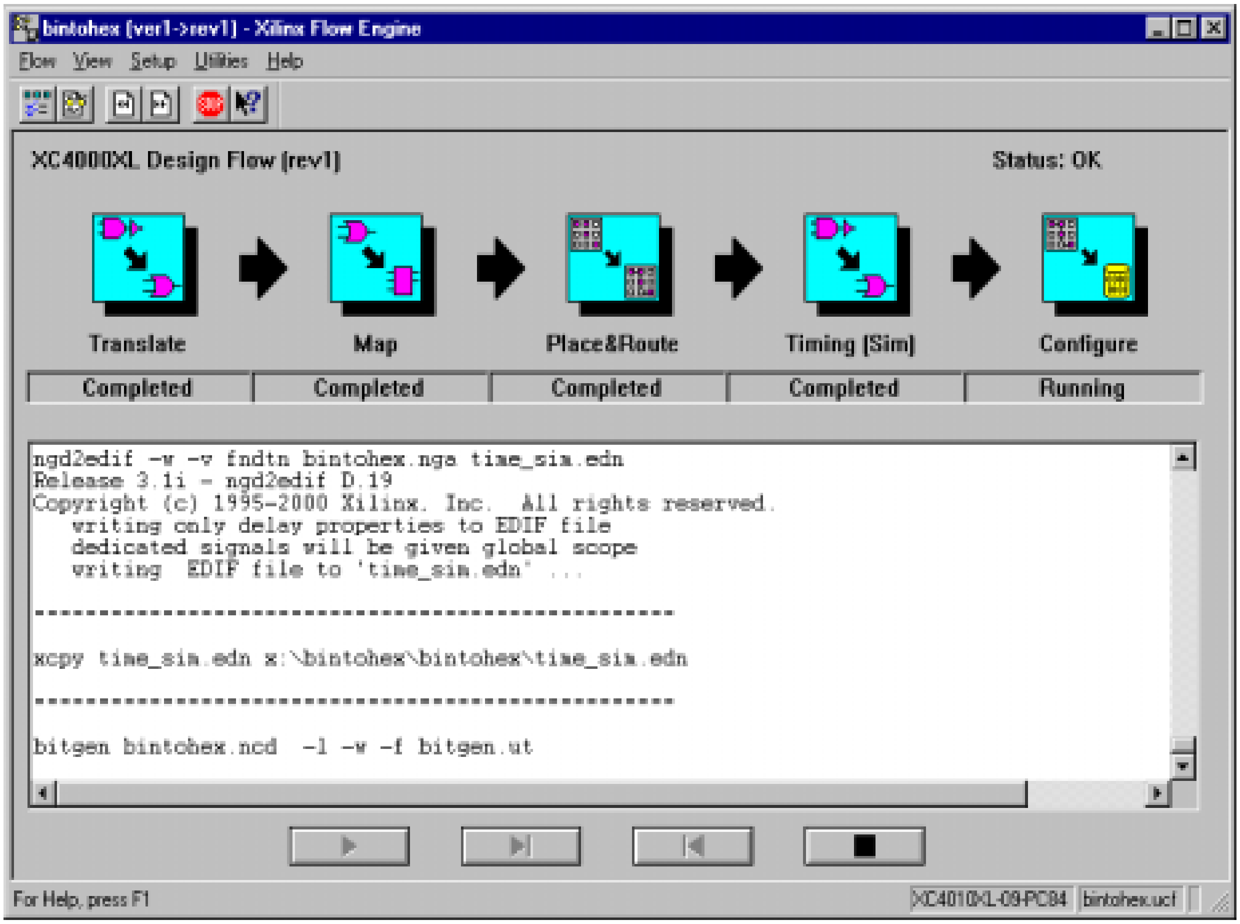
Physical Implementation settings

Revision name: rev1

Control Files: SET

Figure 3.32: Synthesis/Implementation Setting Window

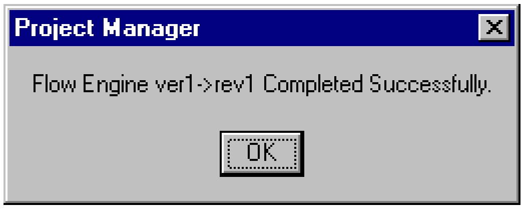
After this, a **Compilation Flow Engine** window will appear as shown in Figure 3.33 with the appropriate implementation information being sent to the status area of this window as the compilation process progresses.



```
ngd2edif -w -v fndtn bintohex.ngo time_sim.edn
Release 3.1i - ngd2edif D.19
Copyright (c) 1995-2000 Xilinx, Inc.  All rights reserved.
   writing only delay properties to EDIF file
   dedicated signals will be given global scope
   writing  EDIF file to 'time_sim.edn' ...

--------------------------------------------------

xcpy time_sim.edn x:\bintohex\bintohex\time_sim.edn

--------------------------------------------------

bitgen bintohex.ncd  -l -w -f bitgen.ut
```

Figure 3.33: Compilation Flow Engine Window

This engine will take the design through the Translate, Map, Place&Route, Timing, and Configure steps. At the end of process, many files will be generated which can be observed by clicking on the **Report** tab of the **Project Manager**. Errors in this process will require modifications to the original design. When the implementation portion of the design is successfully completed the following message will appear



Flow Engine ver1->rev1 Completed Successfully.

Click on **OK**. Then return to the **Project Manager** window to prepare to actually download the design to configure the targeted hardware.

Programming (Configuration) Phase

This phase is identical to that described in Chapter 2. Please refer to this material to configure the XS 40 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 12 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.

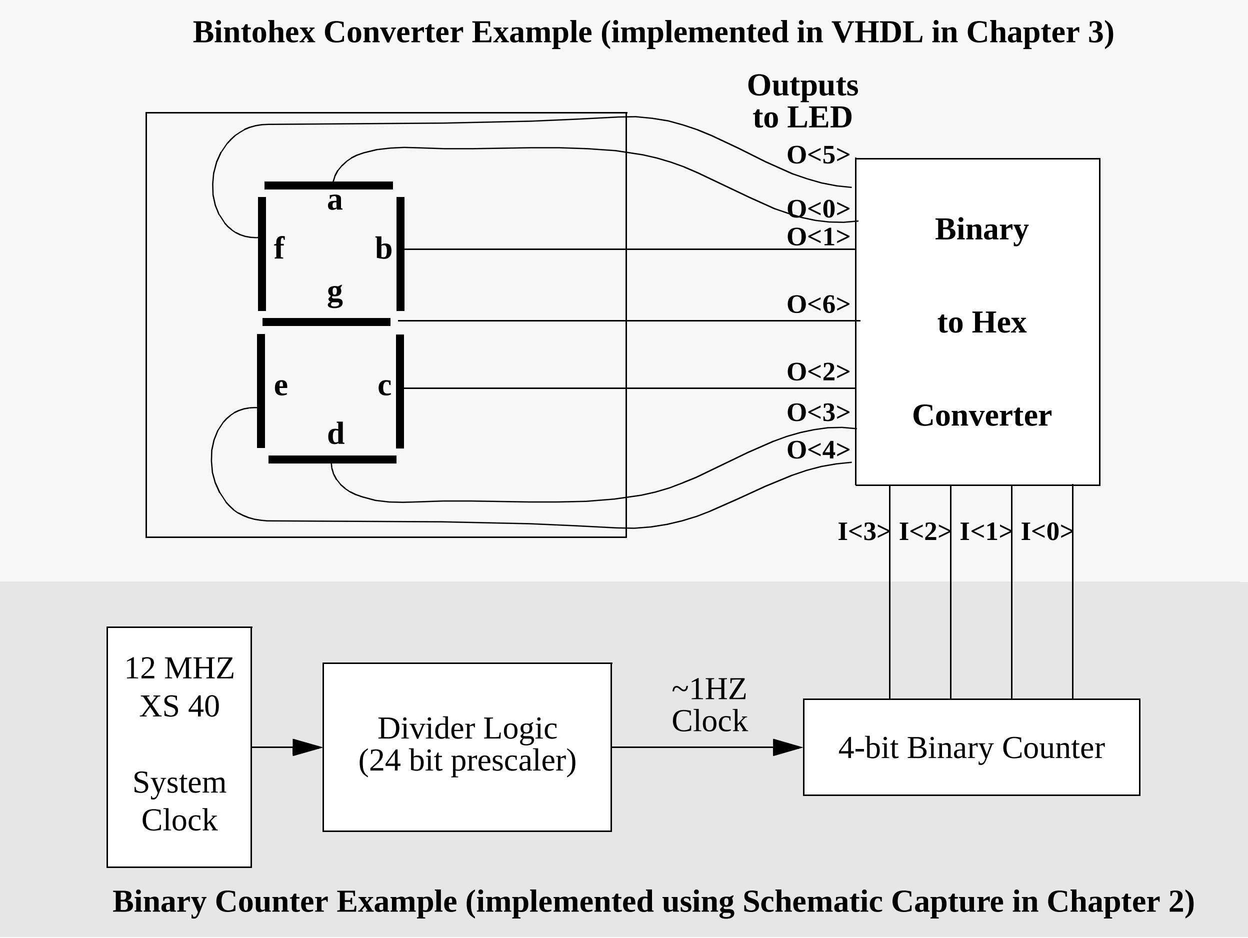
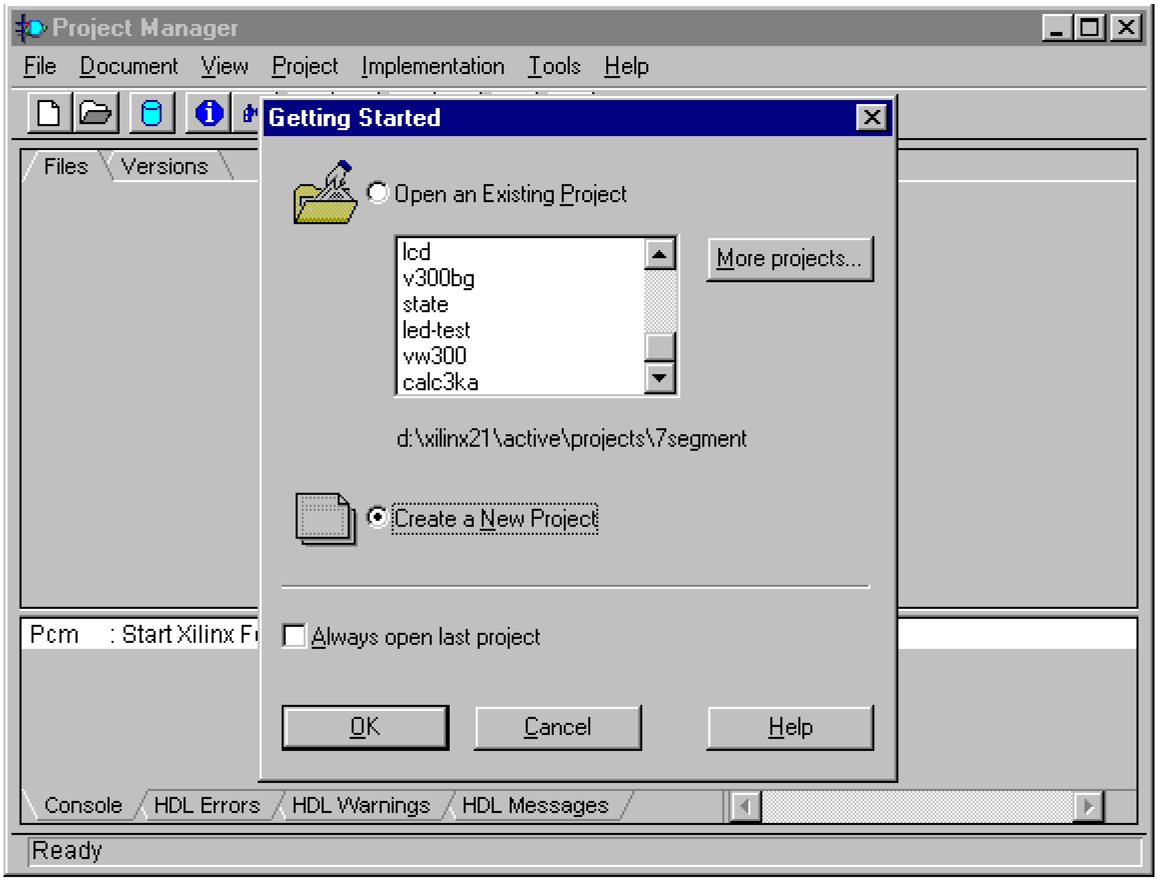

Bintohex Converter Example (implemented in VHDL in Chapter 3)
Outputs to LED
O<5>
O<0>
O<1>
O<6>
O<2>
O<3>
O<4>
a
f
b
g
e
c
d
Binary to Hex Converter
I<3>
I<2>
I<1>
I<0>
12 MHZ XS 40 System Clock
Divider Logic (24 bit prescaler)
~1HZ Clock
4-bit Binary Counter
Binary Counter Example (implemented using Schematic Capture in Chapter 2)

Figure 4.1: Hexadecimal Counter Example

Setting up the Project

As in the previous cases, to enter this design, first start the **Project Manager** of Xilinx Foundation Series 3.1i by double clicking on the **Project Manager** Icon, of Foundation 3.1i from the Window's Desktop.

The Project Manager will then display the **Getting Started** selection window. This window directs the user either to **Open** an Existing Project or **Create** a New Project. For this case, we would to **Create a New Project**. Select this option as shown in Figure 4.2. Then click on the **OK** button.



○ Open an Existing Project

lcd
v300bg
state
led-test
vw300
calc3ka

d:\xilinx21\active\projects\7segment

● Create a New Project

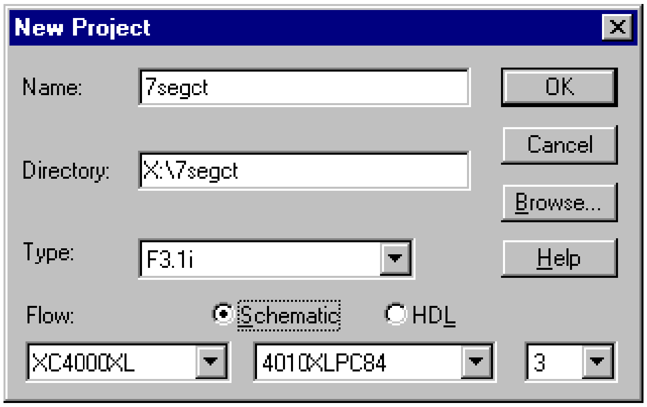
□ Always open last project

Pcm : Start Xilinx F

Ready

Figure 4.2: Getting Started Window

A **New Project** window will appear as shown in Figure 4.3. We will call this hybrid schematic capture/VHDL design <u>7segct</u>. Enter this name in the **Name** field. Make sure that the **Directory** field is set to the desired directory and the **Type** field is set to the appropriate version of the Xilinx Foundation tool set (Foundation 3.1i in this case). Also make sure that the **Flow** field for this example is set to Schematic, since we will be using a schematic for our base design and will be incorporating an HDL macro as a component of the schematic. On the XS 40, the targeted device is Xilinx XC4000 series family device, specifically a XC4010XL FPGA (it is written directly on the device).Then click on the **OK** button.



Name: 7segct

Directory: X:\7segct

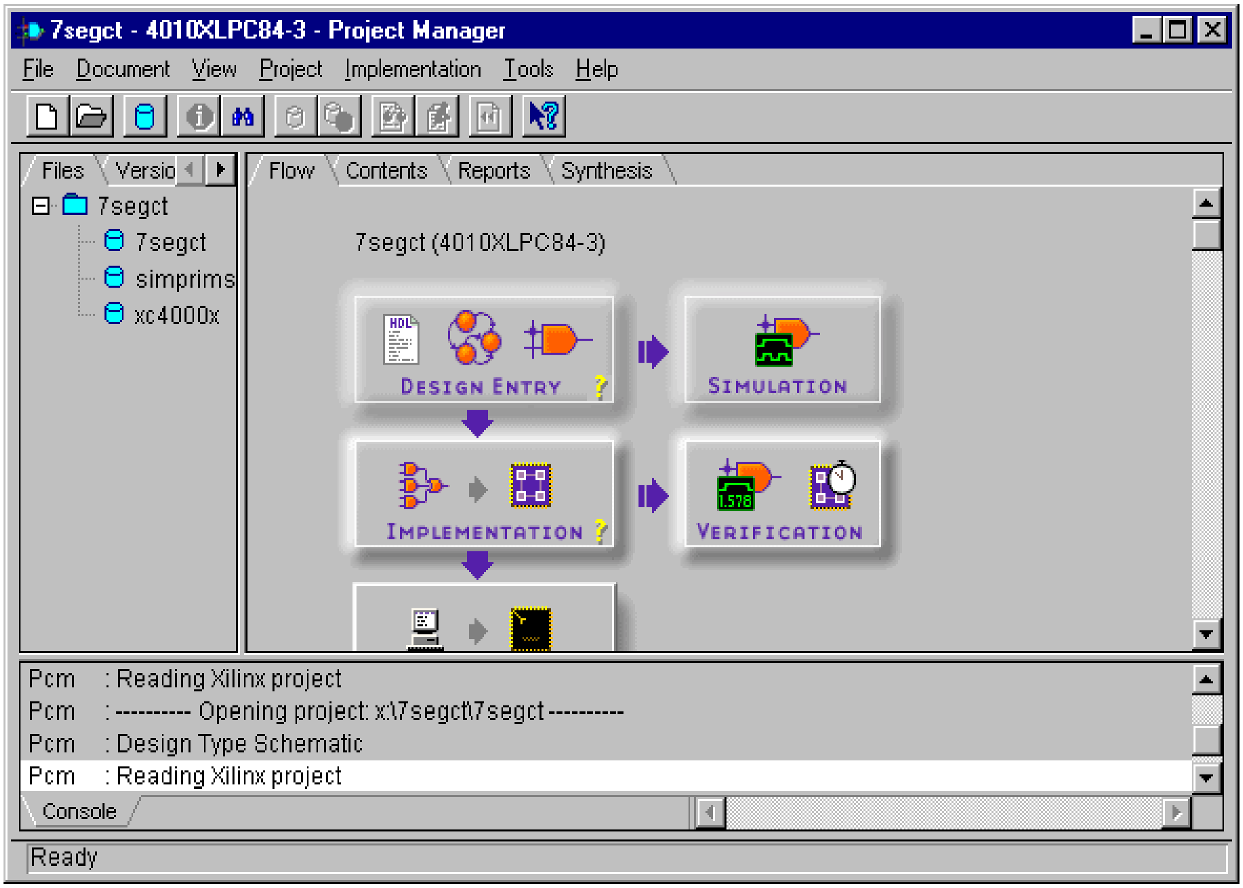
Type: F3.1i

Flow: ● Schematic ○ HDL

XC4000XL 4010XLPC84 3

Figure 4.3: New Project Window

The Xilinx Foundation Project Manager window will appear as shown in Figure 4.4.



```
Pcm    : Reading Xilinx project
Pcm    : --------- Opening project: x:\7segct\7segct ---------
Pcm    : Design Type Schematic
Pcm    : Reading Xilinx project
```

Figure 4.4: Project Manager Window -- Preparing to Enter Schematic Capture Mode

Design Entry

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. To do this first enter the Xilinx Foundation software **Schematic Editor** by pressing the appropriate icon, , located on the **Design Entry** button of the Project Manager Window (see Figure 4.4). Then reenter the binary counter design (but do not enter the inverters, output buffers and output pads) or copy the appropriate sections from the previous design using the Windows NT clip board. The portion of the design to be replicated is shown in Figure 4.5.

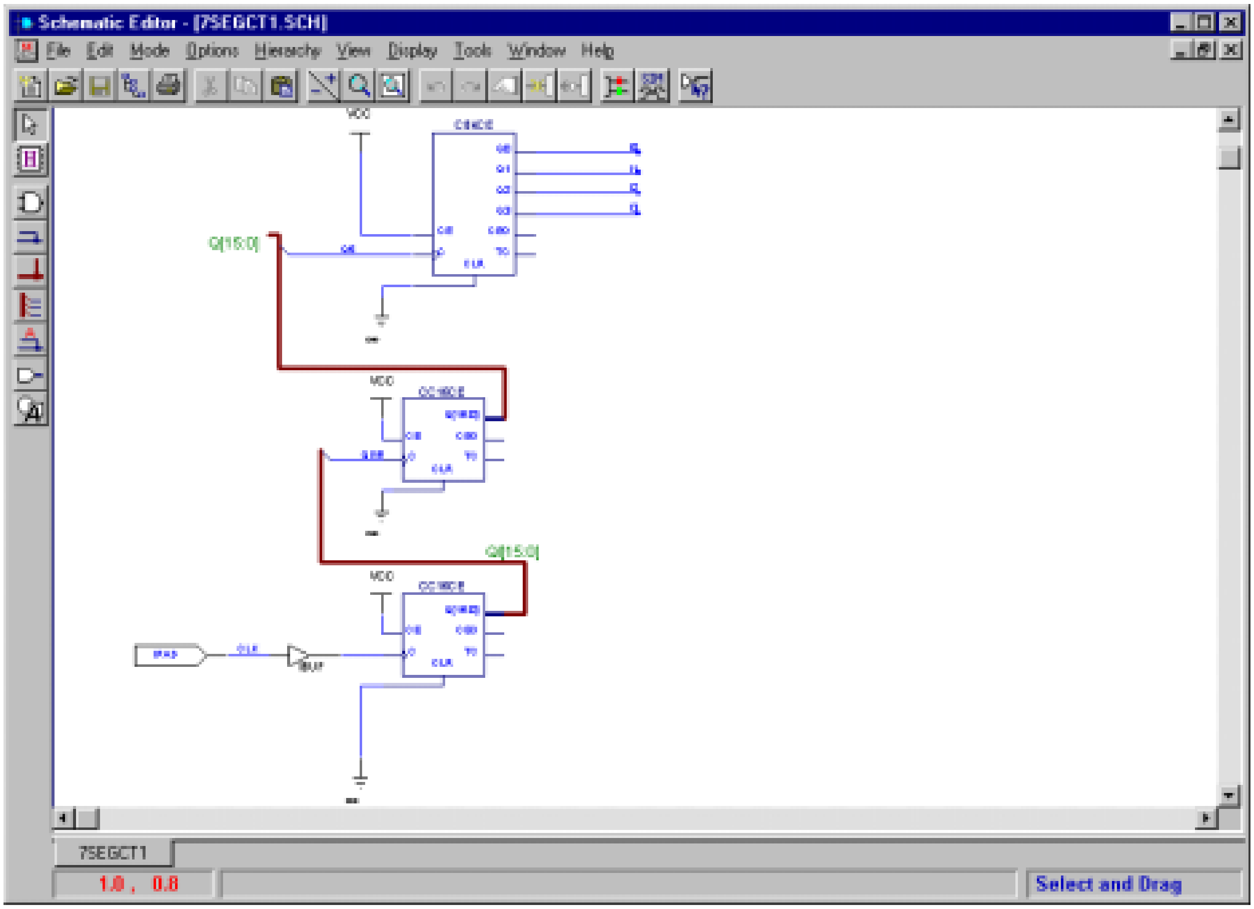
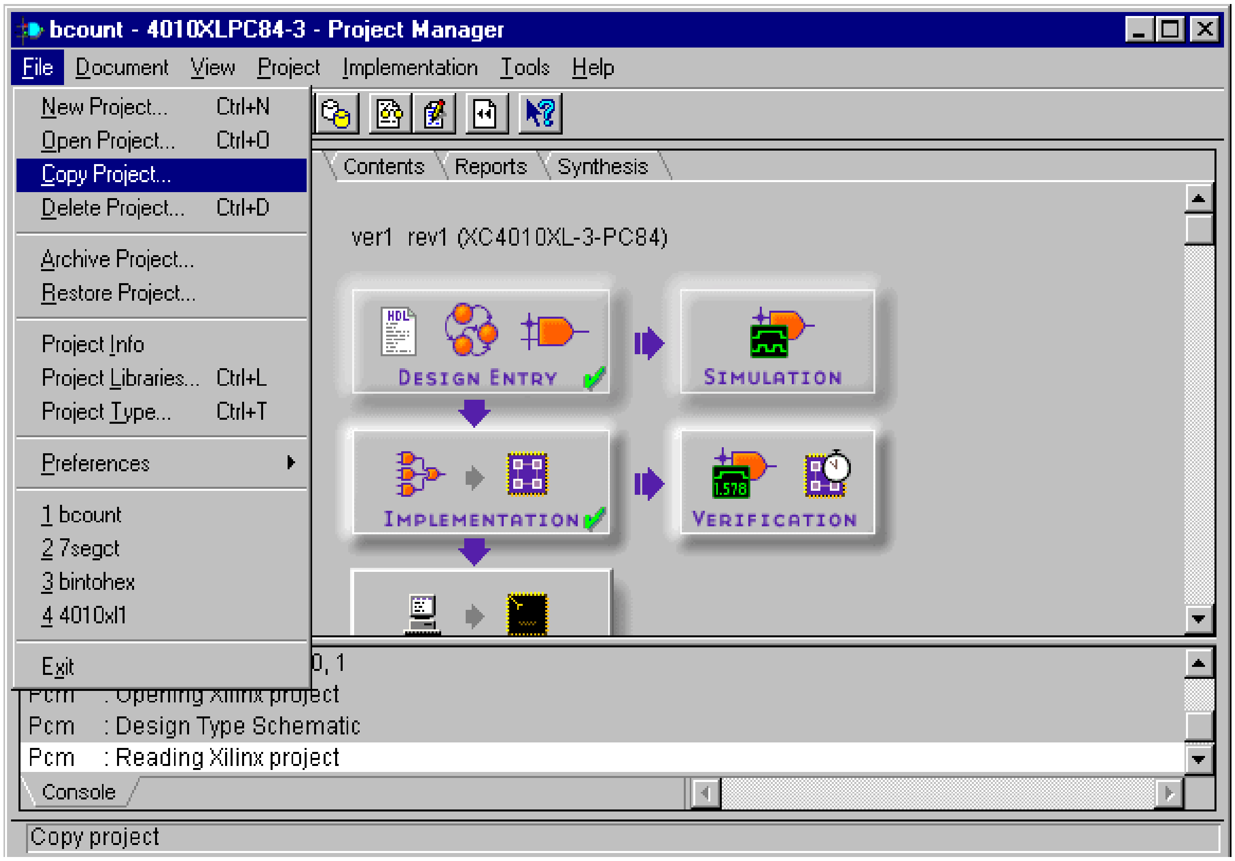

Schematic Editor - [7SEGCT1.SCH]
File Edit Mode Options Hierarchy View Display Tools Window Help
VCC
CB4CE
Q[15:0]
VCC
CC16CE
Q[15:0]
VCC
CC16CE
IPAD
CLK
BUFG
7SEGCT1
1.0 , 0.8
Select and Drag

Figure 4.5: Schematic Editor -- Coping Relevant Parts of Four Bit Binary Counter Design

An alternative means to reuse the logical schematic of the bcount (binary counter) example of Chapter 2 is to first to open the bcount project from the **Project Manager's File** menu. Then select the **Copy Project** option as shown in Figure 4.6.



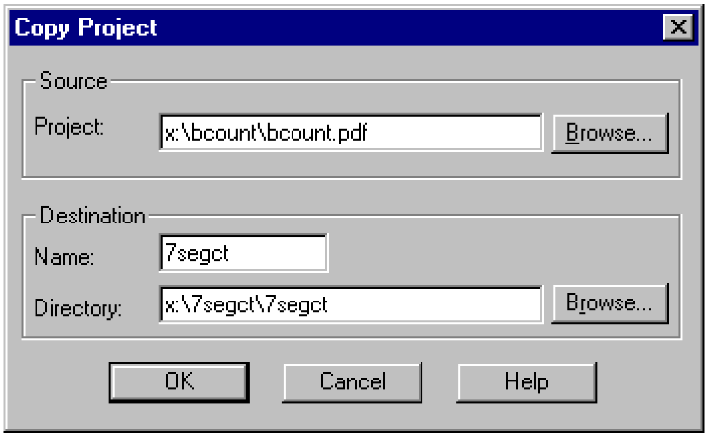
ver1 rev1 (XC4010XL-3-PC84)

```
0, 1
Pcm   : Opening Xilinx project
Pcm   : Design Type Schematic
Pcm   : Reading Xilinx project
```

Copy project

Figure 4.6: Coping the bcount Project into a new Project Directory

Then simply enter the new project name, 7segct, into the **Name**: field of the **Destination** area of the **Copy Project** dialog box as shown in Figure 4.7. Once the project is copied, select the **Open Project** option from the **File** menu of the **Project Manager** window and enter the newly created 7segct design. Then enter the **Schematic Editor** and remove the inverters, output buffers and output pads to make the design appear as shown in Figure 4.5.



Copy Project

Source

Project: x:\bcount\bcount.pdf

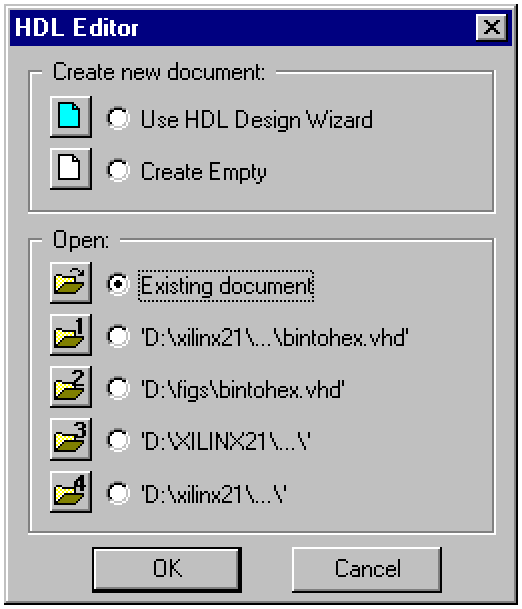
Destination

Name: 7segct

Directory: x:\7segct\7segct

Figure 4.7: Coping Project Dialog Box

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 Manager** window and click on the **HDL Editor** icon, on the **Design Entry** button of the **Project Manager** flow window (under the **Flow** tab). This will launch the **HDL Editor** and a window which will allow one to either create a new HDL file or open an existing HDL file (see Figure 4.8). In this example, we will want to use an existing document (i.e. the bintohex.vhd file presented in Chapter 3). To do this make sure the **Existing document** option is checked and then press the **OK** button.



HDL Editor

Create new document:

○ Use HDL Design Wizard

○ Create Empty

Open:

● Existing document

○ 'D:\xilinx21\...\bintohex.vhd'

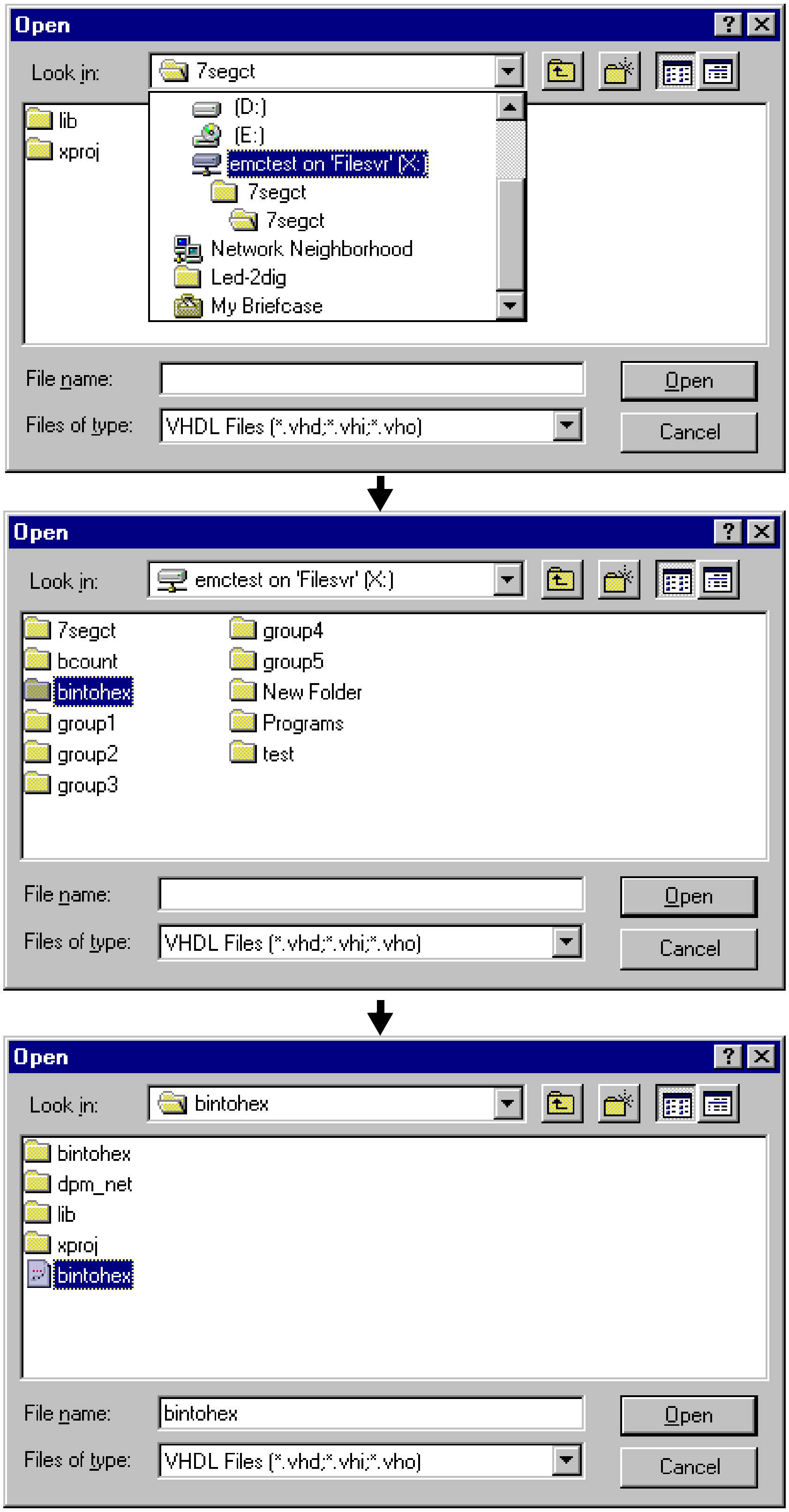
○ 'D:\figs\bintohex.vhd'

○ 'D:\XILINX21\...\'

○ 'D:\xilinx21\...\'

Figure 4.8: HDL Editor Window -- Using an Existing Design (i.e. bintohex.vhd)

At this point an **Open** window will appear. We want to select the binary to hexadecimal converter model file, 'bintohex.vhd' from the previous design so we need to go back to the previous project. This is accomplished as outlined in Figure 4.9.



Look in: 7segct

File name:

Files of type: VHDL Files (*.vhd;*.vhi;*.vho)

Look in: emctest on 'Filesvr' (X:)

File name:

Files of type: VHDL Files (*.vhd;*.vhi;*.vho)

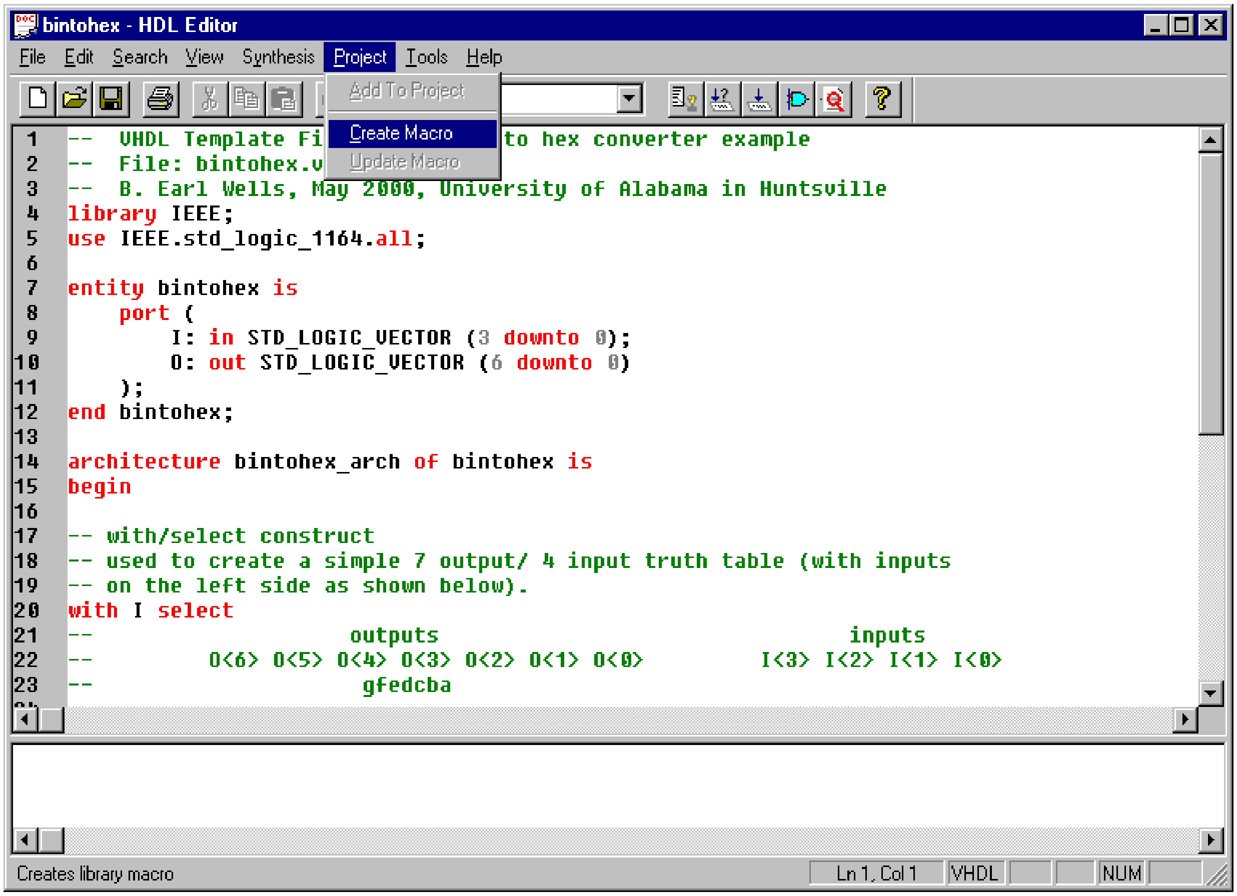
Look in: bintohex

File name: bintohex

Files of type: VHDL Files (*.vhd;*.vhi;*.vho)

Figure 4.9: Opening and Exporting the Previous bintohex.vhd Model

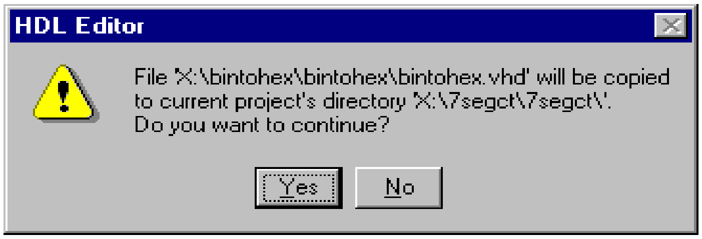
The next step is to create a macro from this file which can be selected as a component from within the **Schematic Editor**. To do this, select the **Project** option from the tool box area of the **HDL Editor**. The select the **Create Macro** option as shown in Figure 4.10.



```
-- VHDL Template Fi                    to hex converter example
-- File: bintohex.v
-- B. Earl Wells, May 2000, University of Alabama in Huntsville
library IEEE;
use IEEE.std_logic_1164.all;

entity bintohex is
    port (
        I: in STD_LOGIC_VECTOR (3 downto 0);
        O: out STD_LOGIC_VECTOR (6 downto 0)
    );
end bintohex;

architecture bintohex_arch of bintohex is
begin

-- with/select construct
-- used to create a simple 7 output/ 4 input truth table (with inputs
-- on the left side as shown below).
with I select
--                    outputs                                inputs
--          O<6> O<5> O<4> O<3> O<2> O<1> O<0>         I<3> I<2> I<1> I<0>
--                     gfedcba
```

Figure 4.10: HDL Editor Window -- Creating a Macro for the bintohex

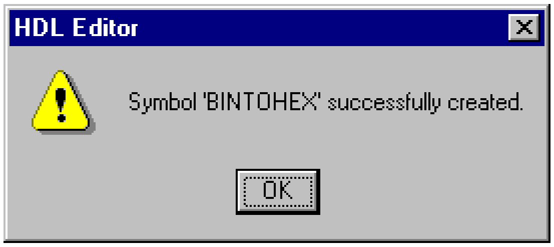
A file copy warning message similar to the one in Figure 4.11 should appear. Click the **Yes** button to continue.



File 'X:\bintohex\bintohex\bintohex.vhd' will be copied
to current project's directory 'X:\7segct\7segct\'.
Do you want to continue?

Figure 4.11: File Copy Warning Message

The macro should now be successfully created. A status message to this effect will appear. Press the **OK** button to continue.

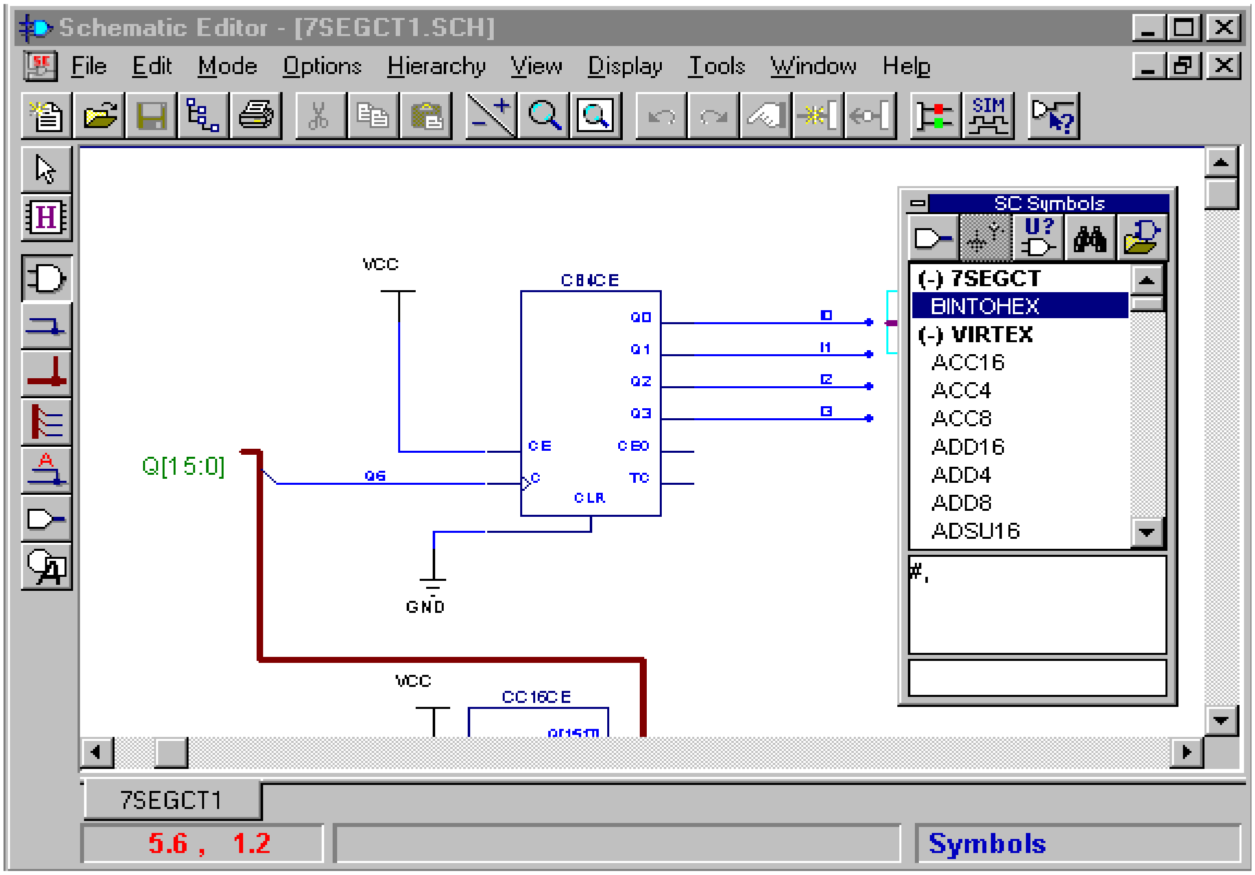


HDL Editor

Symbol 'BINTOHEX' successfully created.

Figure 4.12: Macro Symbol Creation Status Message

The BINTOHEX macro that was created can now be selected in the same manner as any other component. It will appear on the **SC Symbols** window as shown in Figure 4.13.



Schematic Editor - [7SEGCT1.SCH]

Figure 4.13: Schematic Editor Window -- BINTOHEX Macro Selection

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 4.14. After the schematic is successfully entered, the design should be **Saved** and the user should return to the **Project Manager** window.

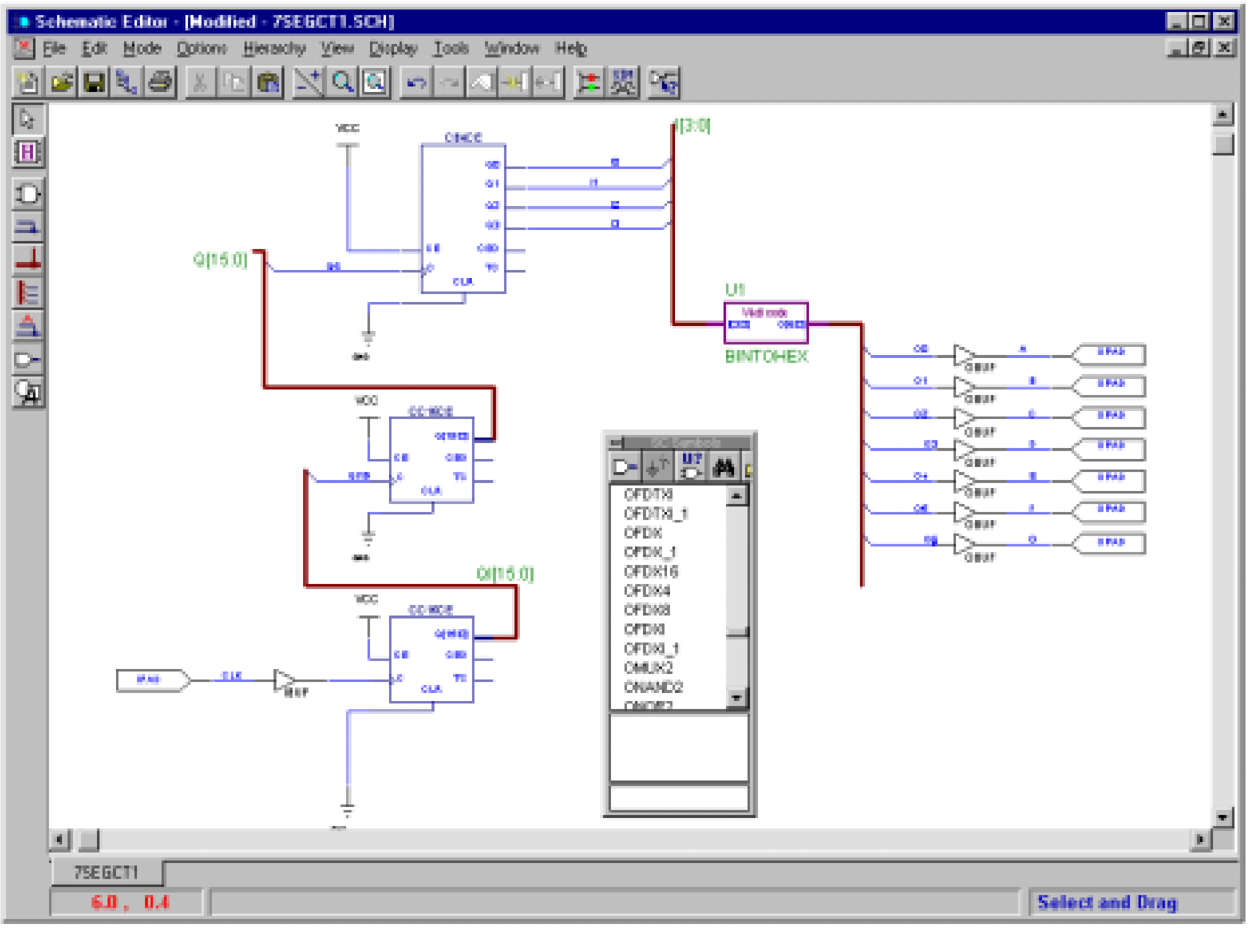
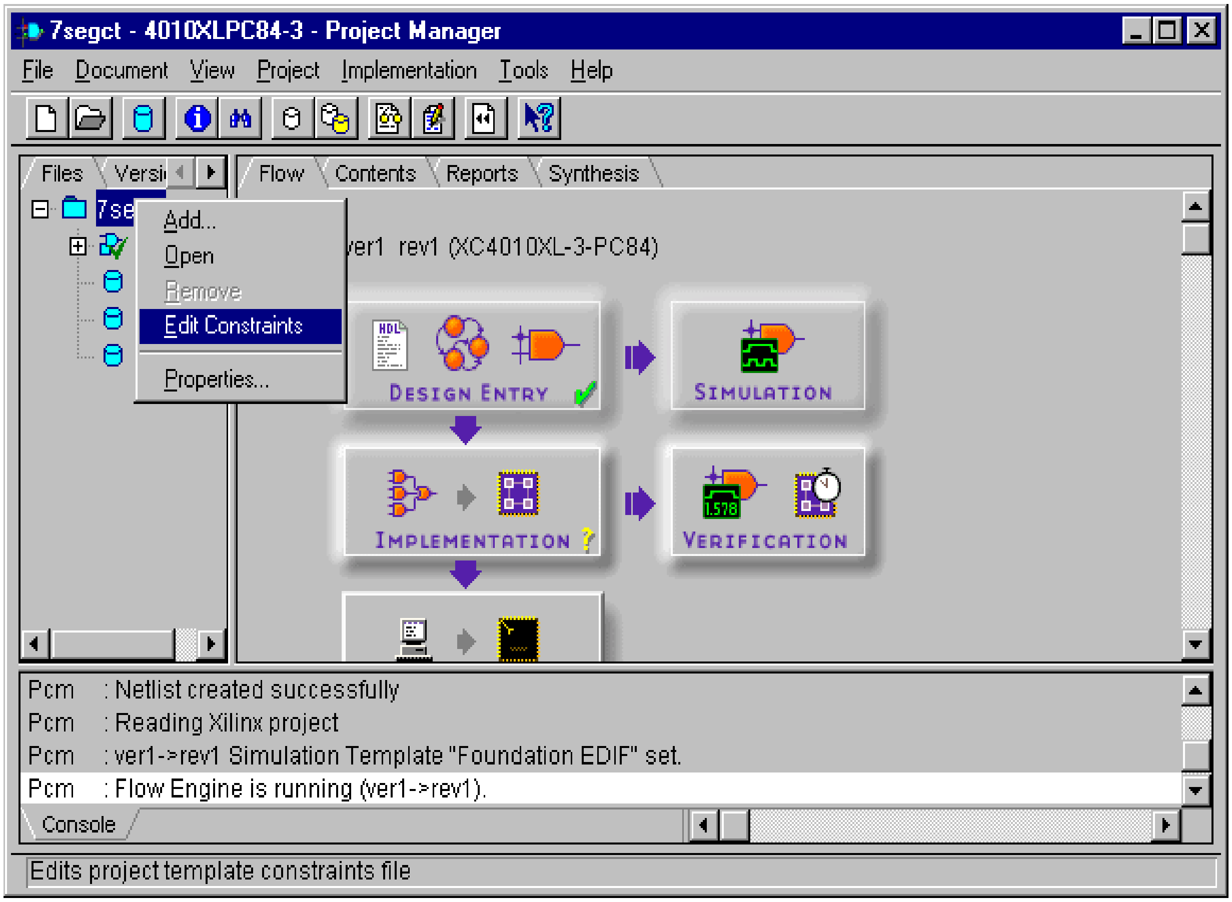


Figure 4.14: Schematic Editor Window: Final Design for the Hexadecimal Counter

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 XS 40 12MHz 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 XS 40 pins. To lock a specified pin name with the net name specified in the schematic from the **Project Manager** window, again move the cursor to point to the top hierarchy source tree located under the **Files** tab on the left-hand side of the window. Then click on right most button and a pull down window should appear as shown in Figure 4.15. Select the **Edit Constraints** option with the left mouse button.



```
Pcm    : Netlist created successfully
Pcm    : Reading Xilinx project
Pcm    : ver1->rev1 Simulation Template "Foundation EDIF" set.
Pcm    : Flow Engine is running (ver1->rev1).
```

Edits project template constraints file

Figure 4.15: HDL Editor Window -- Creating a Macro for the bintohex model

For this hybrid design, the desired schematic name to Xilinx XC4010XL 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 XC4010XL's I/O pins.)

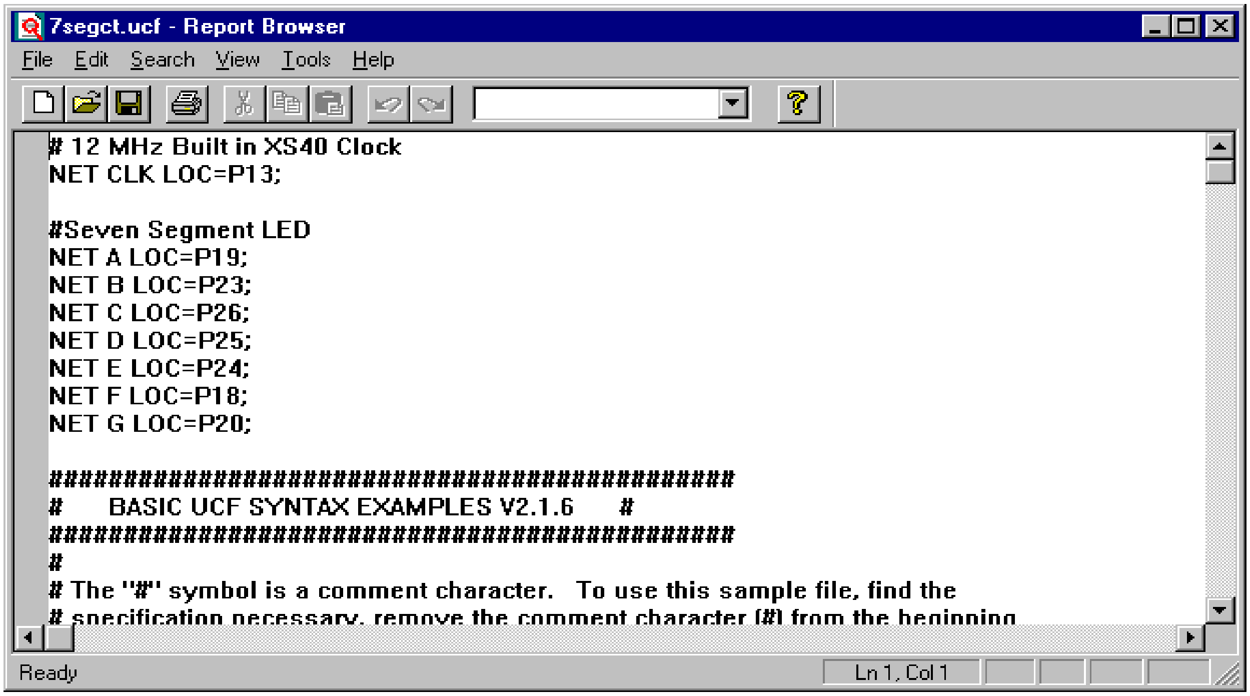
Table 4.1: Desired Pin Locking (cross reference) Configuration

I/O Pin Description	Schematic Net Name	Xilinx XC4010XL Pin Name
XS 40 12MHz Clock	CLK	13
XC4010XL Pin 19, (segment ‘a’ of LED)	A	19
XC4010XL Pin 23, (segment ‘b’ of LED)	B	23
XC4010XL Pin 26, (segment ‘c’ of LED)	C	26
XC4010XL Pin 25, (segment ‘d’ of LED)	D	25
XC4010XL Pin 24, (segment ‘e’ of LED)	E	24
XC4010XL Pin 18, (segment ‘f of LED)	F	18
XC4010XL Pin 20, (segment ‘g’ of LED)	G	20

The resulting additions to the **User Constraints File** are shown in Figure 4.16 where the cross reference described in Table 4.1 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 Manager** window.

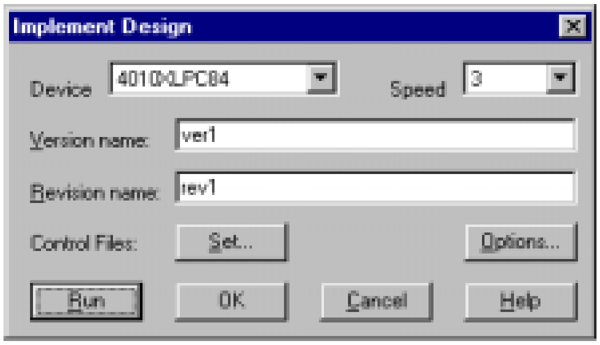


```
# 12 MHz Built in XS40 Clock
NET CLK LOC=P13;

#Seven Segment LED
NET A LOC=P19;
NET B LOC=P23;
NET C LOC=P26;
NET D LOC=P25;
NET E LOC=P24;
NET F LOC=P18;
NET G LOC=P20;

####################################################
#      BASIC UCF SYNTAX EXAMPLES V2.1.6      #
####################################################
#
# The "#" symbol is a comment character.   To use this sample file, find the
# specification necessary, remove the comment character (#) from the beginning
```

Figure 4.16: Adding Pin Locking information to the User Constraint File

Implementation Phase

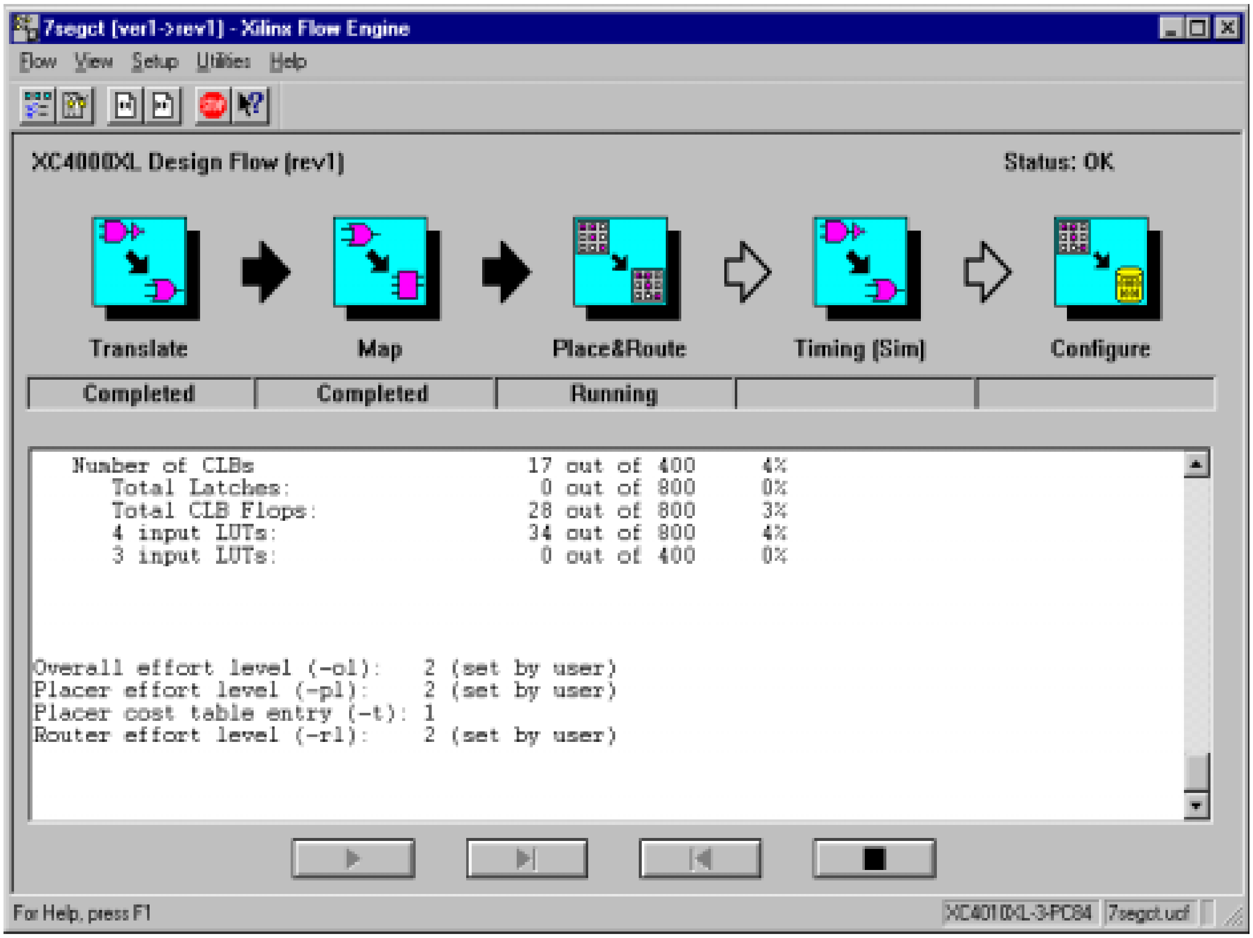
This phase is virtually the same as in the previous two examples. To compile the hexadecimal counter design from the **Project Manager** window click on the **Implementation** button, which is under the **Flow** tab. The **Implement Design** window should appear as shown in Figure 4.17. All the parameters in the dialog boxes should be correct, so click on the **Run** button.



Device: 4010XLPC84
Speed: 3
Version name: ver1
Revision name: rev1

Figure 4.17: Implement Design Window

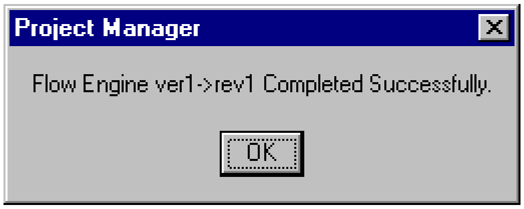
After this, a **Compilation Flow Engine** window will appear as shown in Figure 4.18 with appropriate implementation information being sent to the status area of this window as the design compilation process progresses.



```
   Number of CLBs                          17 out of  400    4%
      Total Latches:                        0 out of  800    0%
      Total CLB Flops:                     28 out of  800    3%
      4 input LUTs:                        34 out of  800    4%
      3 input LUTs:                         0 out of  400    0%



Overall effort level (-ol):   2 (set by user)
Placer effort level (-pl):    2 (set by user)
Placer cost table entry (-t): 1
Router effort level (-rl):    2 (set by user)
```

Figure 4.18: Compilation Flow Engine Window

This engine will take the design through the Translate, Map, Place&Route, Timing, and Configure steps. At the end of process, many files will be generated which can be observed by clicking on the **Report** tab of the **Project Manager**. Errors in this process will require modifications to the original design. When the implementation portion of the design is successfully completed the following message will appear



Flow Engine ver1->rev1 Completed Successfully.

Click on **OK**. Then return to the **Project Manager** window to prepare to actually download the design to configure the targeted hardware.

Programming (Configuration) Phase

This phase is identical to that described in Chapter 2. Please refer to this material to configure the XS 40. When the design has been configured the external LED display should successively and repetitively display the hexadecimal pattern ‘0’ to ‘F’ in sequence (lowest to highest with wrap around) with out requiring any interaction from the user.

Chapter 5: References

[1] *XS 40, XSP, and XS95 Board Manual*, Version 1.1, Xess Corporation, 1998.

[2] *Foundation Series 3.1i Quick Start Guide*, Xilinx, 2000.

[3] *Foundation Series 3.1i Floorplanner Guide*, Xilinx, 2000.

[4] *Foundation Series 3.1i Timing Analyzer Guide*, Xilinx, 2000.

[5] *Foundation Series 3.1i Constraints Editor Guide*, Xilinx, 2000.

[6] *Foundation Series 3.1i VHDL Reference Guide*, Xilinx, 2000.

Appendix A

Table 1: XS 40 I/O Pins Function

Pin # XS 40	***Pin # 8031***	***Function***	***Description***
1		***GND***	
2		***VCC***	
3		A<0>	Address bus
4		A<1>	Address bus
5		A<2>	Address bus
6	P1.3(43)	Switch 3	
7	P1.0(40)	DLED7	Bit7 of DLED
8	P1.1(41)	Switch 1	
9	P1.2(42)	Switch 2	
10	P0.7(30)	AD<7>	Address/Data bus
11	P3.0(5)	_Receiver_	
12	P3.2(8)	_Interrupt 0_	
13		12MHz	Clock from external oscillator
14	PSEN(26)	PSEN_	Active-Low Program-Store Enable
_15_			_TDI*_
_16_			_TCK*_
_17_			_TMS*_
18		LED<5> and DLED5	Segment 5 of 7Segment and Bit5 of DLED
19		LED<0> and DLED0	Segment 0 of 7Segment and Bit0 of DLED
20		LED<6> and DLED6	Segment 6 of 7Segment and Bit6 of DLED
21	P3.3(9)	_Interrupt 1_	Only connected to 8031
22	P3.5(11)	_Timer 1_	Only connected to 8031
23		LED<1> and DLED1	Segment 1 of 7Segment and Bit1 of DLED
24		LED<4> and DLED4	Segment 4 of 7Segment and Bit4 of DLED
25		LED<3> and DLED3	Segment 3 of 7Segment and Bit3 of DLED
26		LED<2> and DLED2	Segment 2 of 7Segment and Bit2 of DLED
27	P3.7(13)	RD_	Active-Low Read
28	P2.5(25)	A<15>	Address bus
29	ALE(27)	ALE_	Active-Low Address Latch Enable
_30_			_M1*_
31		***GND***	
_32_			_M0*_
33		***VCC***	
_34_			_M2*_
35	P0.4(33)	AD<4>	Address/Data bus

Table 1: XS 40 I/O Pins Function

Pin # XS 40	***Pin # 8031***	***Function***	***Description***
36	RST(4)	RST, Switch 8 to reset 8031	Active-High Reset
37	XTAL1(15)	XTAL	Input Clock to 8031
38	P0.3(34)	AD<3>	Address/Data bus
39	P0.2(35)	AD<2>	Address/Data bus
40	P0.1(36)	AD<1>	Address/Data bus
41	P0.0(37)	AD<0>	Address/Data bus
42		***VCC***	
43		***GND***	
44		Parallel<0>	Parallel Port Input
45		Parallel<1>	Parallel Port Input
46		Parallel<2>	Parallel Port Input
47		Parallel<3>	Parallel Port Input
48		Parallel<4>	Parallel Port Input
49		Parallel<5>	Parallel Port Input
50	P2.4(22)	A<12>	Address bus
51	P2.2(20)	A<10>	Address bus
52		***GND***	
53			DONE*
54		***VCC***	
55			PROGRAM*
56	P2.3(21)	A<11>	Address bus
57	P2.1(19)	A<9>	Address bus
58	P2.5(23)	A<13>	Address bus
59	P2.0(18)	A<8>	Address bus
60	P2.6(24)	A<14>	Address bus
61		OE_	Active-Low Output Enable (RAM)
62	P3.6(12)	WR_	Active-Low Write, Also control RAM
63		***VCC***	
64		***GND***	
65		CE_	Active-Low Chip Enable (RAM)
66	P1.6(2)	Serial Receiver	XS 4010XL’s perspective
67	P1.7(3)	Serial Transmitter	XC4010XL’s perspective
68	P3.4(10)	Interrupt 0	
69	P3.1(7)	Transmitter	
70	P1.5(1)		General I/O
_71_			_D0*, DIN*_
_72_			_DOUT*_

Table 1: XS 40 I/O Pins Function

Pin # XS 40	***Pin # 8031***	***Function***	***Description***
73			*CCLK**
74		***VCC***	
75			*TDO**
76		***GND***	
77	P1.4(44)	Switch 4	
78		A<3>	Address bus
79		A<4>	Address bus
80	P0.6(31)	AD<6>	Address/Data bus
81	P0.5(32)	AD<5>	Address/Data bus
82		A<5>	Address bus
83		A<6>	Address bus
84		A<7>	Address bus