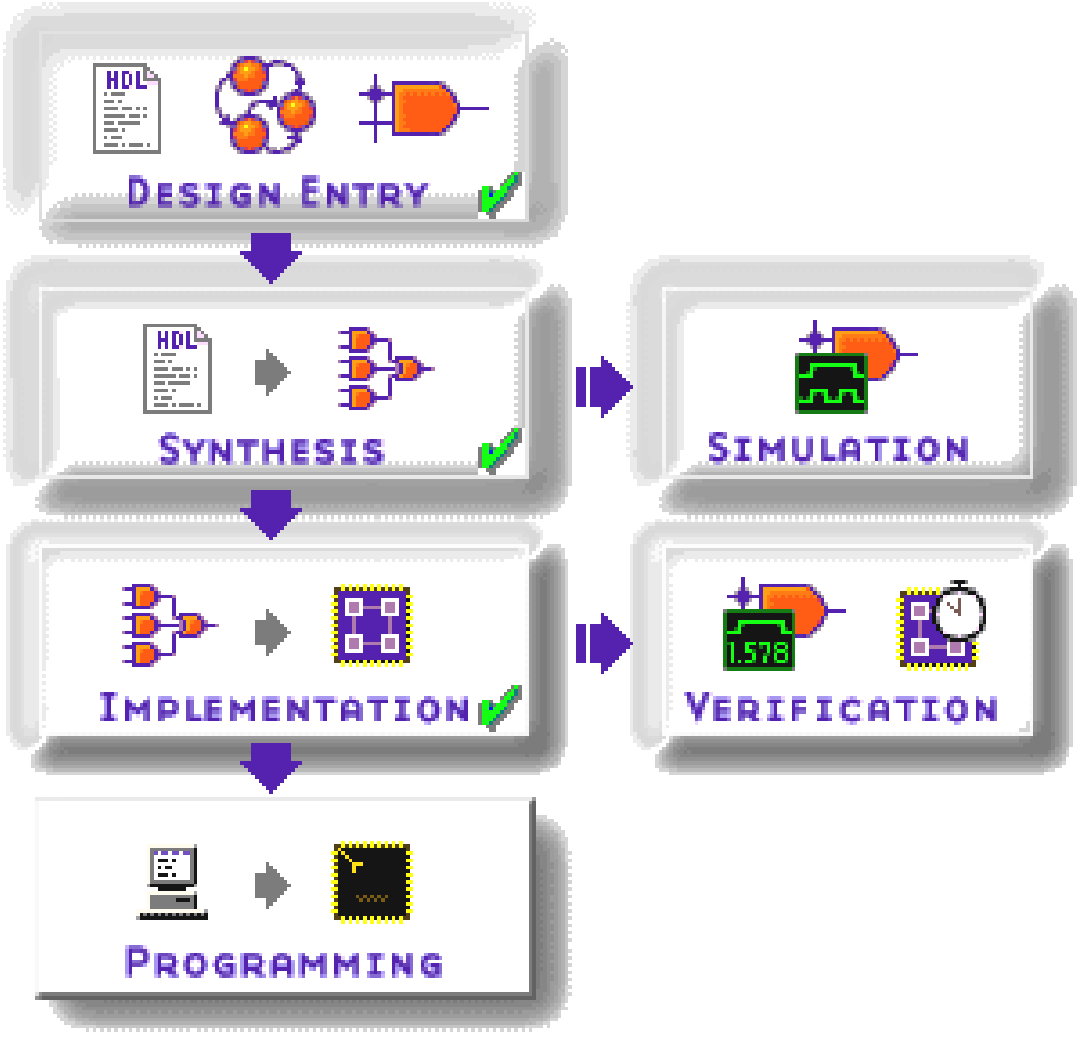
Pragmatic Logic Design

With XILINX Foundation 2.1i



David E. Vanden Bout

XESS Corp

© 2001 by X Engineering Software Systems Corp., Apex, North Carolina 27502

All rights reserved. No part of this text may be reproduced, in any form or by any means, without permission in writing from the publisher.

The author and publisher of this text have used their best efforts in preparing this text. These efforts include the development, research, and testing of the theories and programs to determine their effectiveness. The author and publisher make no warranty of any kind, expressed or implied, with regard to these programs or the documentation contained in this text. The author and publisher shall not be liable in any event for incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of these programs.

XESS, XS40, and XS95 are trademarks of X Engineering Software Systems Corp. XILINX, Foundation, XC4000, and XC9500 are trademarks of XILINX Corporation. Other product and company names mentioned are trademarks or trade names of their respective companies.

The software described in this text is furnished under a license agreement. The software may be used or copied under terms of the license agreement.

3

Hierarchical Design

In this chapter you will learn how to:

- Create VHDL designs composed of a hierarchy of VHDL modules.
- Create hierarchical VHDL designs which also include schematics in the hierarchy.
- Create hierarchical schematics that include VHDL modules.
- Use LogiBlox to create modules for incorporation into hierarchical designs.

Hierarchy

Most complex systems possess a hierarchical structure. Hierarchy arises in man-made systems because people are good at decomposing problems into simpler problems whose solutions can be combined into a complete solution. In an eight-bit adder, for example, you might decompose the operation into a set of one-bit additions. Then you could design a logic circuit that adds binary bits using AND, OR, and NOT gates. Finally, you could combine the one-bit adder modules into a single eight-bit adder. But it doesn't have to stop there because you could use the eight-bit adder to build a 32-bit adder as part of a microprocessor chip that resides on a circuit board in a computer attached to a network…

Some of the advantages of building circuits from a hierarchy of modules are:

Design re-use: A module can be re-used across multiple designs so you do less work overall.

Information hiding: Encapsulating a circuit into a module lets you ignore its internal operational details while allowing you to concentrate on the interaction of the module's inputs and outputs with the rest of the system.

Replication: Building a large circuit by duplicating a small group of modules is much easier than building a large circuit by stitching together a large number of primitive gates.

In this chapter we will see how the Foundation software supports hierarchical design techniques. The example design I will use consists of a 28-bit binary counter whose four upper bits are displayed as a hexadecimal digit on a seven-segment LED (see Figure 7). When driven by a 50 MHz clock, the displayed digit will change every 2^{24} / 50,000,000 =

© 2001 by XESS Corp.

0.34 seconds. Foundation lets you describe the root of the design using an HDL (either VHDL or Verilog) or as a schematic, and you have this same flexibility with each module in the lower levels of the hierarchy. So you can change your design style to match the type of circuit you are building. We will build the design in three different ways:

1. The root and all lower level-modules are described using VHDL;
2. The root is described using VHDL, but the lower-level modules are designed using both VHLD and schematics.
3. The root is designed as a schematic that contains VHDL and schematic-based lower-level modules.

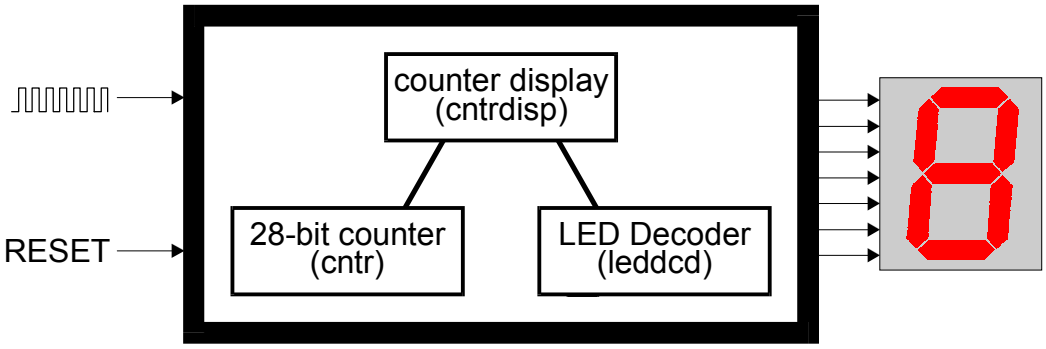
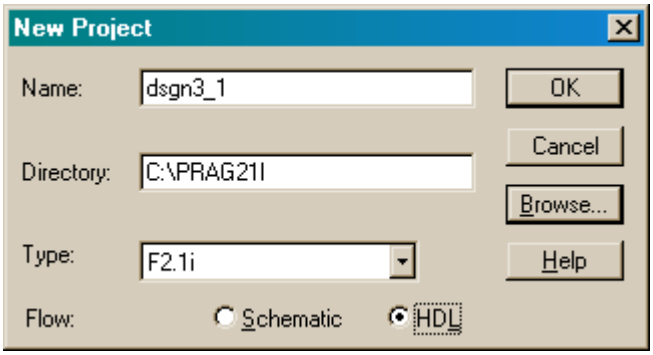


Figure 7: Design hierarchy for a counter display.

Hierarchical VHDL-Based Design

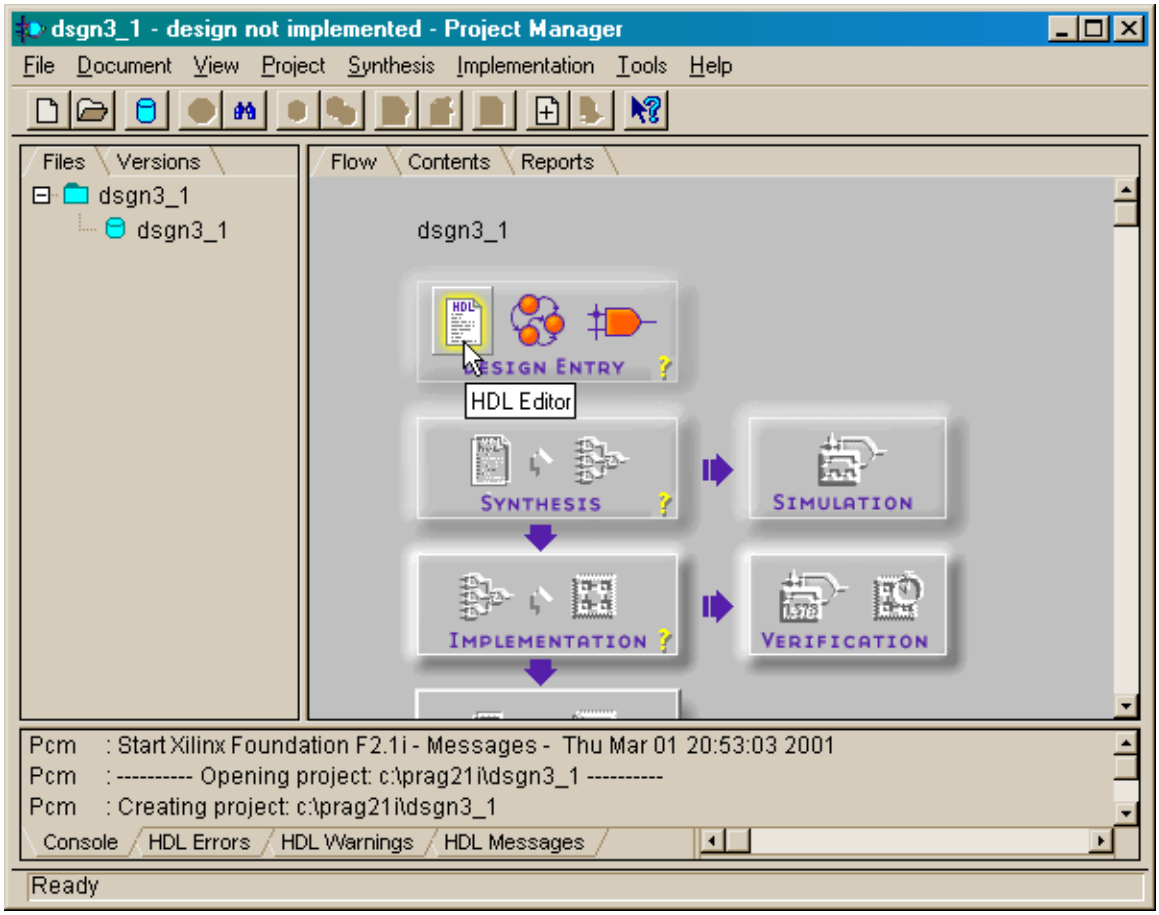
The first design (***dsgn3_1***) is started as an HDL-based project.



© 2001 by XESS Corp.

Creating the VHDL Files for the Lower-Level Modules

We start by using the HDL Editor to design the lower-level modules for the 28-bit counter and the seven-segment LED decoder.



The VHDL code for the LED decoder is shown in Listing 1. The code looks very similar to what we saw in Chapter 1 with the following differences:

Line 3: A new package from the `IEEE` library is used. The `numeric_std` package gives us access to some new VHDL types such as `SIGNED` and `UNSIGNED` bit vectors and the arithmetic operations that act on them.

Lines 5–12: A new package (`leddcd_pckg`) is defined that contains a single component declaration for the LED decoder. The component declaration tells other VHDL modules about the types of inputs and outputs used to interface to the LED decoder.

Line 12: The four-bit `d` input to the LED decoder has been declared as an `UNSIGNED` bit vector rather than as a `STD_LOGIC_VECTOR`. This will simplify the interface with the 28-bit counter module which also outputs `UNSIGNED` values.

Listing 1: VHDL code for the seven-segment LED decoder module.

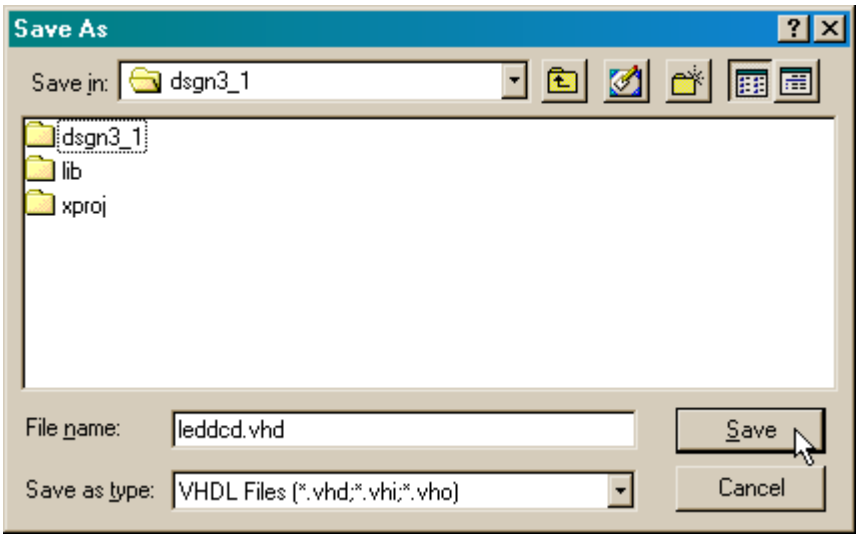
```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.numeric_std.all;
```

© 2001 by XESS Corp.

```
 1
 2  package leddcd_pckg is
 3
 4  component leddcd
 5      port (
 6          d: in UNSIGNED (3 downto 0);
 7          s: out STD_LOGIC_VECTOR (6 downto 0)
 8      );
 9  end component;
10
11  end leddcd_pckg;
12
13
14  library IEEE;
15  use IEEE.std_logic_1164.all;
16  use IEEE.numeric_std.all;
17
18  entity leddcd is
19      port (
20          d: in UNSIGNED (3 downto 0);
21          s: out STD_LOGIC_VECTOR (6 downto 0)
22      );
23  end leddcd;
24
25  architecture leddcd_arch of leddcd is
26  begin
27      with d select
28          s <= "1110111" when "0000", -- 0
29               "0010010" when "0001", -- 1
30               "1011101" when "0010", -- 2
31               "1011011" when "0011", -- 3
32               "0111010" when "0100", -- 4
33               "1101011" when "0101", -- 5
34               "1101111" when "0110", -- 6
35               "1010010" when "0111", -- 7
36               "1111111" when "1000", -- 8
37               "1111011" when "1001", -- 9
38               "1111110" when "1010", -- A
39               "0101111" when "1011", -- b
40               "1100101" when "1100", -- C
41               "0011111" when "1101", -- d
42               "1101101" when "1110", -- E
43               "1101100" when others; -- F
44  end leddcd_arch;
```

© 2001 by XESS Corp.

In the **HDL Editor** window, select the File➔Save As menu item and save the LED decoder VHDL into the leddcd.vhd file in the ***dsgn3_1*** project directory.



The VHDL code for the counter is shown in Listing 1. The important parts of the code are as follows:

Line 3: Once again we access the `numeric_std` package. We will need the arithmetic addition operator to increment the counter value.

Lines 5–12: Another new package (`cntr_pckg`) is defined that contains a single component declaration for the counter.

Lines 25–34: The counter interface consists of an input to reset the counter to zero, a clock input that increments the counter value on each rising edge, and an `UNSIGNED` bit vector that outputs the current value of the counter. Note that the number of counter output bits is determied by the generic parameter `LENGTH` declared on lines 26–28. The `cntr` module is customized when it is instantiated for a particular application by specifying the value of `LENGTH`.

Line 37: An internal `UNSIGNED` bit vector is declared to hold the value of the counter.

Lines 39–49: The actual counting operation is specified in the `COUNT` process. The process is triggered by changes in the `clk` input (line 39) and the counter changes value on the rising edge of `clk` (line 42). If the `rst` input is high on a rising clock edge, then the counter value is cleared to zero (lines 43–44). (The `TO_UNSIGNED` function from the `numeric_std` package converts an integer into an `UNSIGNED` bit vector.) If the `rst` input is not high, then the value in the counter register is incremented by one (lines 45–46).

Line 51: The value in the counter register is placed on the outputs.

Listing 2: VHDL code for the counter module.

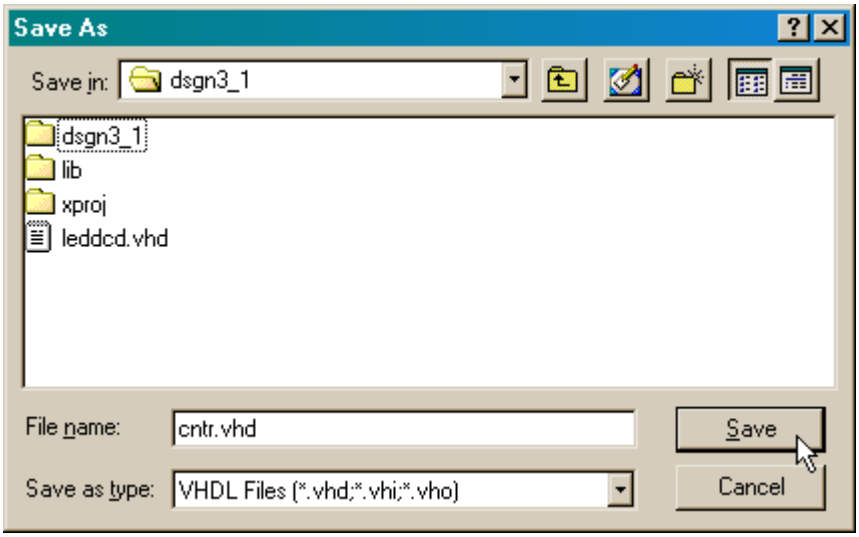
```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
```

© 2001 by XESS Corp.

```
3   use IEEE.numeric_std.all;
4
5   package cntr_pckg is
6
7   component cntr
8         generic (
9               LENGTH: natural         -- number of bits in counter
10        );
11      port (
12        rst: in STD_LOGIC;      -- synchronous reset
13        clk: in STD_LOGIC;      -- counter clock
14        cnt: out UNSIGNED(LENGTH-1 downto 0)      -- counter output
15      );
16  end component;
17
18  end cntr_pckg;
19
20
21  library IEEE;
22  use IEEE.std_logic_1164.all;
23  use IEEE.numeric_std.all;
24
25  entity cntr is
26        generic (
27              LENGTH: natural         -- number of bits in counter
28        );
29      port (
30        rst: in STD_LOGIC;      -- synchronous reset
31        clk: in STD_LOGIC;      -- counter clock
32        cnt: out UNSIGNED(LENGTH-1 downto 0)      -- counter output
33      );
34  end cntr;
35
36  architecture cntr_arch of cntr is
37  signal cnt_r: UNSIGNED(LENGTH-1 downto 0);      -- counter register
38  begin
39        COUNT: process(clk)
40        begin
41              -- change counter only on rising clock edges
42              if (clk'event and clk='1') then
43                    if rst='1' then         -- synchronous reset to 0
44                          cnt_r <= TO_UNSIGNED(0,LENGTH);
45                    else              -- otherwise, increment counter
46                          cnt_r <= cnt_r + 1;
47                    end if;
48              end if;
49        end process COUNT;
50
51        cnt <= cnt_r;     -- output register contents
52  end cntr_arch;
```

© 2001 by XESS Corp.

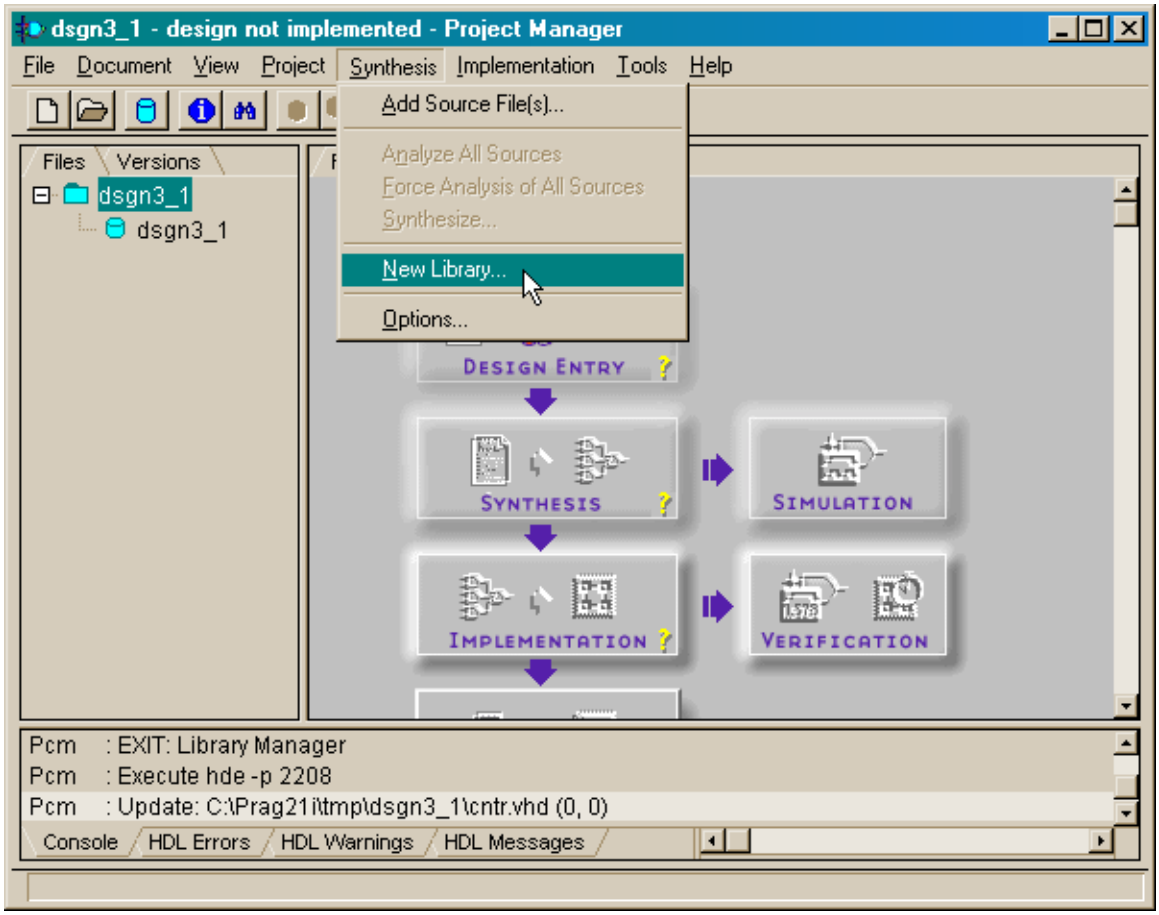
In the **HDL Editor** window, select the File➔Save As menu item and save the counter VHDL into the cntr.vhd file in the ***dsgn3_1*** project directory.



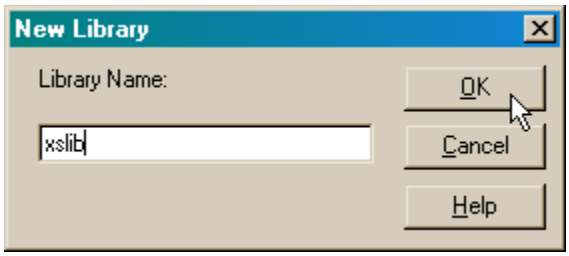
© 2001 by XESS Corp.

Adding a New Library to the Project

Now we have the lower-level modules defined, but we still have to add them to the project so they can be accessed by the root module. To do this, we will create a new library and then add the modules to the library. Select the Synthesis➔New Library... menu item to start this step.

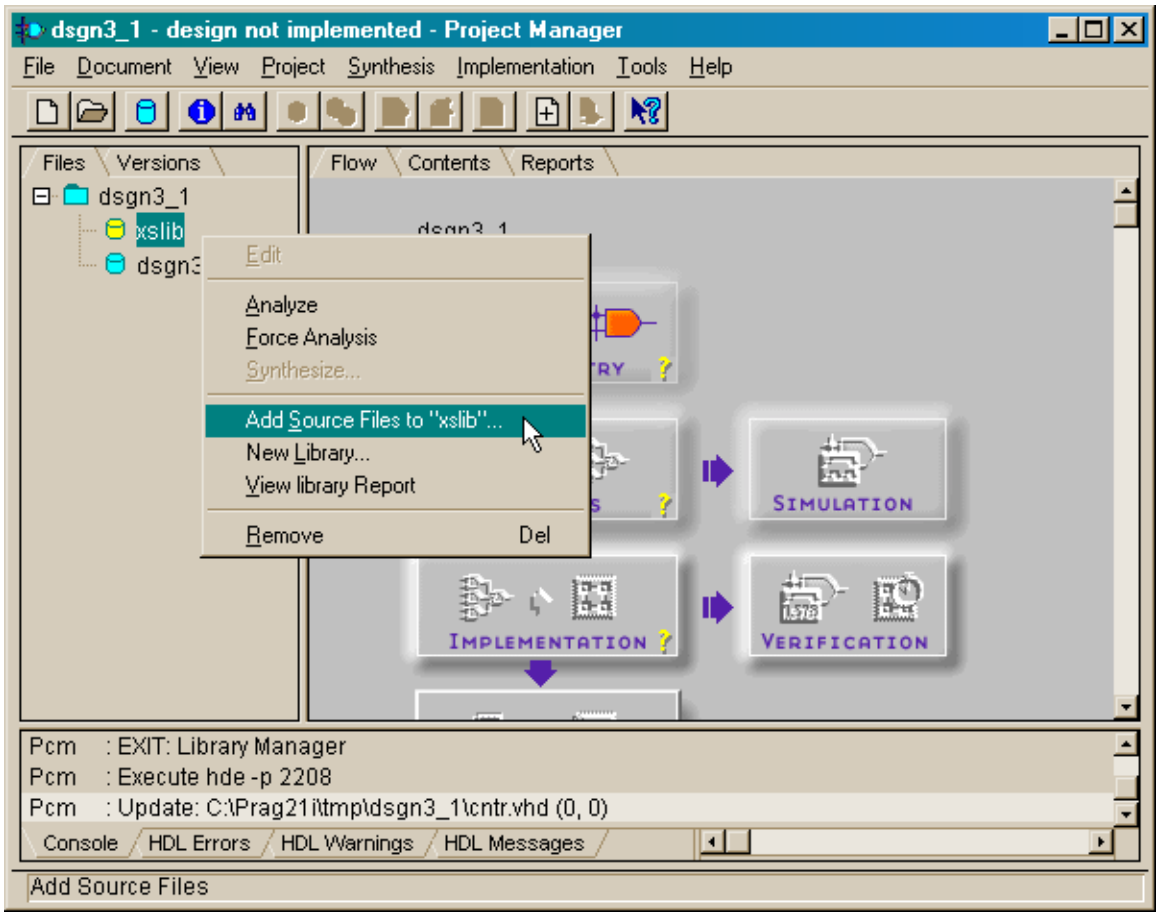


Specify the library name as xslib in the **New Library** window and click on the OK button.

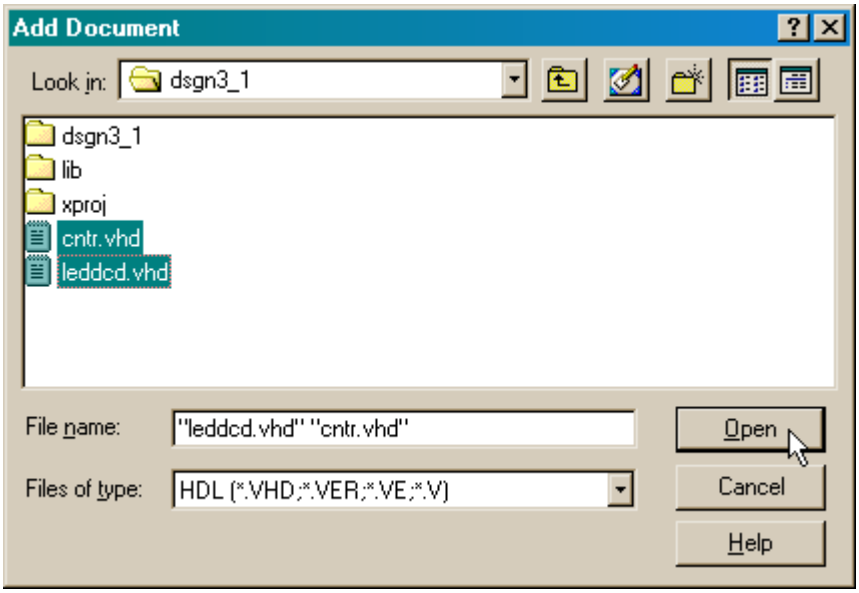


Now you will see the xslib library has been added to the left-hand **Hierarchy** pane of the **Project Manager** window. There isn't anything in this new library yet, but we will fix that by right-clicking on the xslib icon and selecting the Add Source Files to "xslib"... menu item.

© 2001 by XESS Corp.

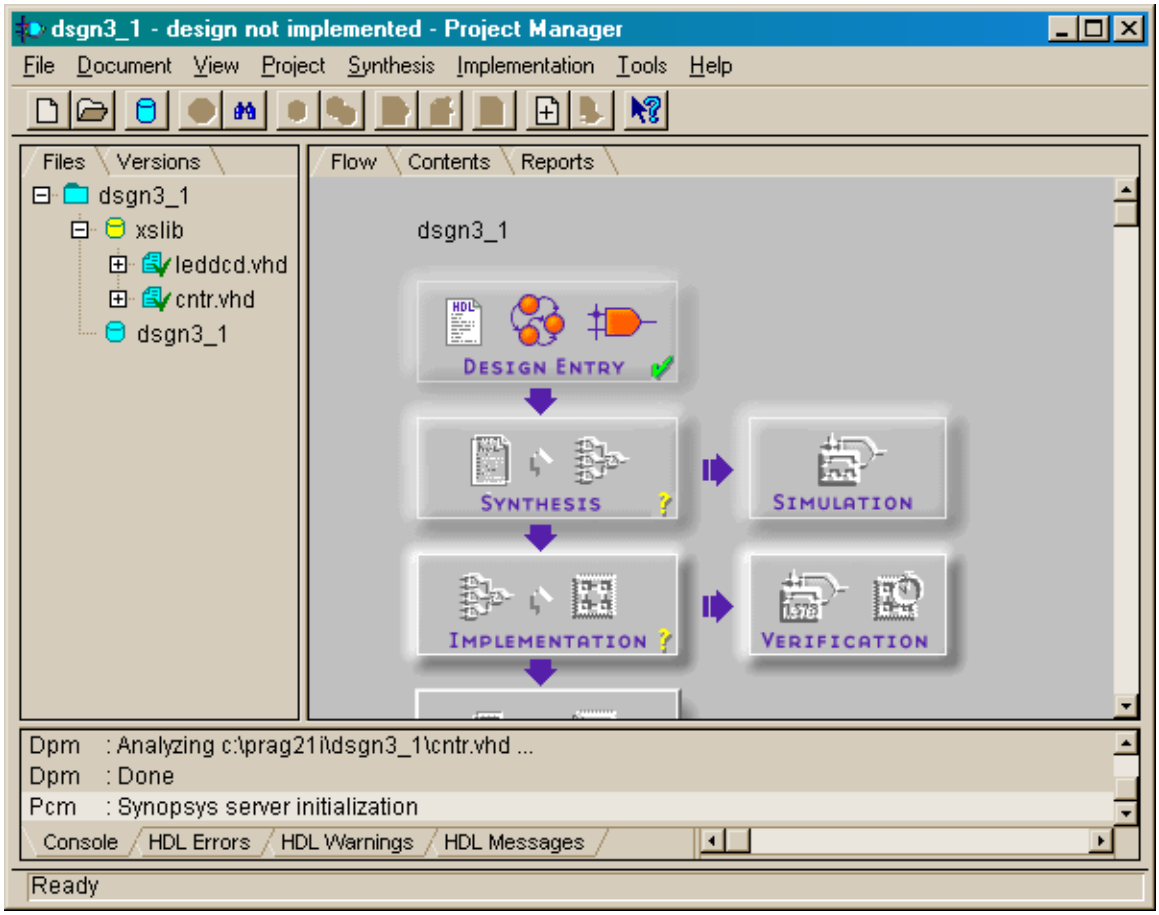


Highlight the cntr.vhd and leddcd.vhd files in the **Add Document** window that appears and then click on the Open button.



© 2001 by XESS Corp.

The modules for the LED decoder and counter will be added to the xslib library. You can click on the **+** icon to the left of the xslib icon to view what modules are included in the library.



Creating the Root Module

Now we have to build the root module that combines the counter and LED decoder modules to create the complete circuit. The VHDL code for the cntdisp module has the following features:

Line 1: XSLIB is now included in the list of libraries used for this design.

Lines 4–5: Each lower-level module in the library declared its own package, so we have to explicitly declare that we are going to use all the components in each package.

Line 16: A constant for the number of bits in the counter is declared and set to 28.

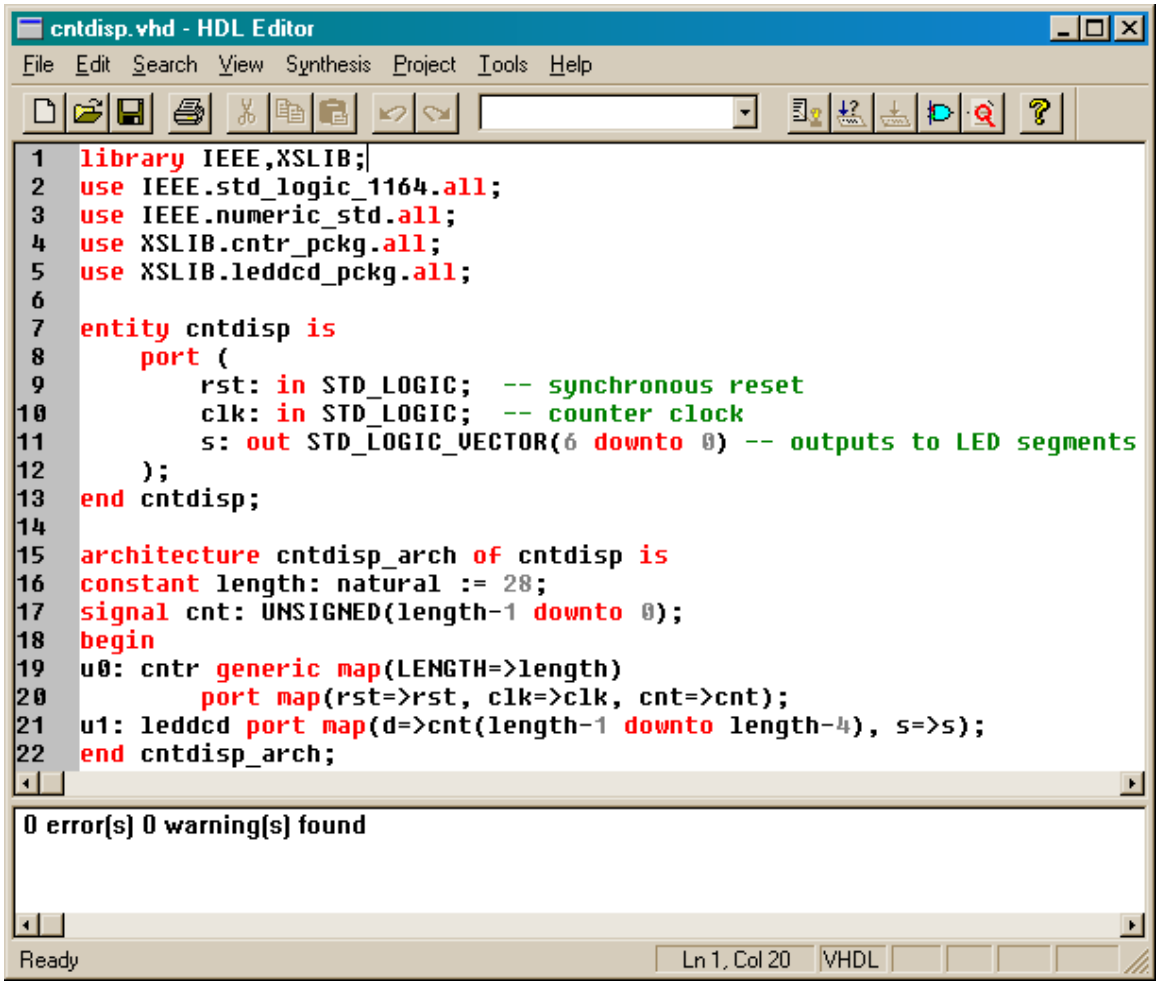
Line 17: An internal 28-bit bit vector is declared. The upper four bits of this vector will be used to transfer the upper counter bits to the LED decoder module.

Lines 19–20: The counter module is instantiated. The generic length parameter is set to 28 and the reset and clock inputs of the counter module are attached to the

© 2001 by XESS Corp.

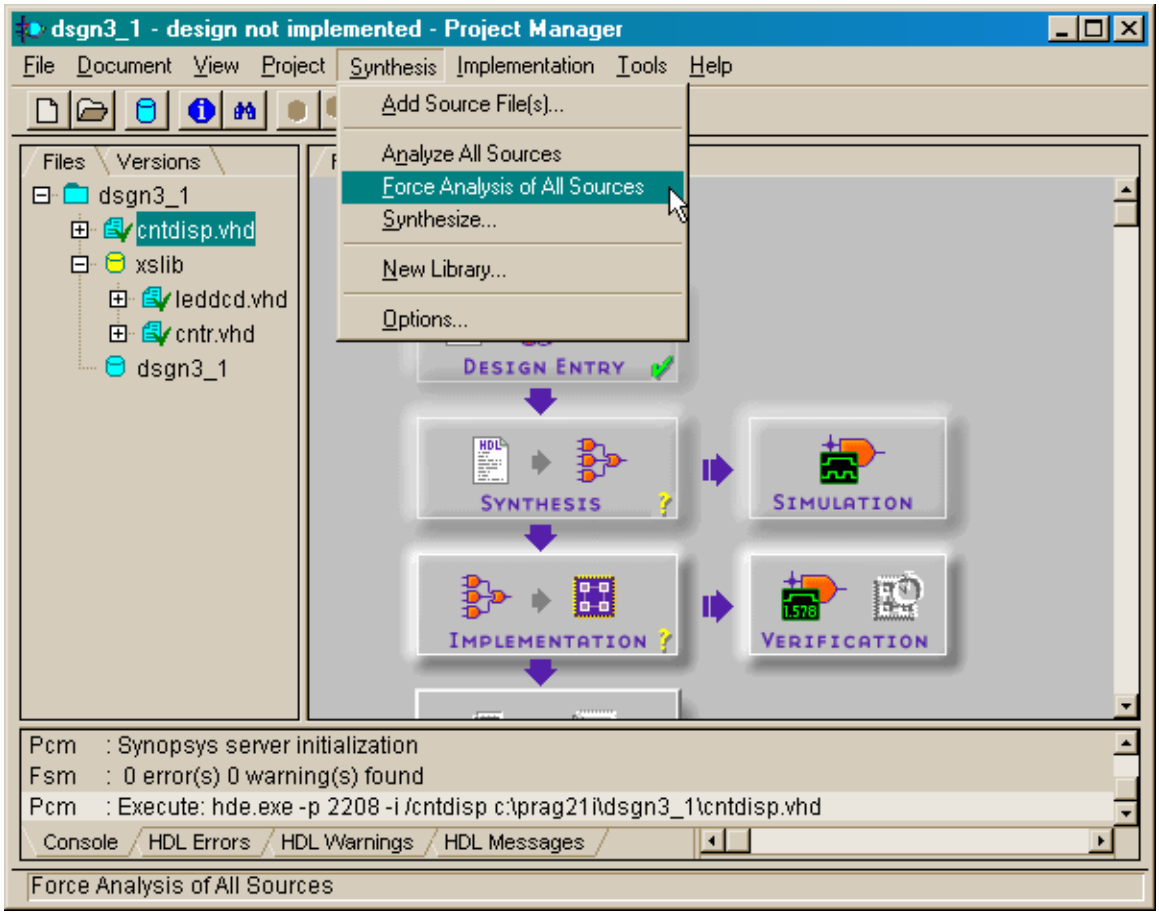
reset and clock inputs to the root module. The counter outputs are attached to the internal bit vector in the root module.

Line 21: The LED decoder module is instantiated. The upper four bits of the counter value are passed into the LED decoder and the outputs of the decoder are connected to the root-level outputs.



© 2001 by XESS Corp.

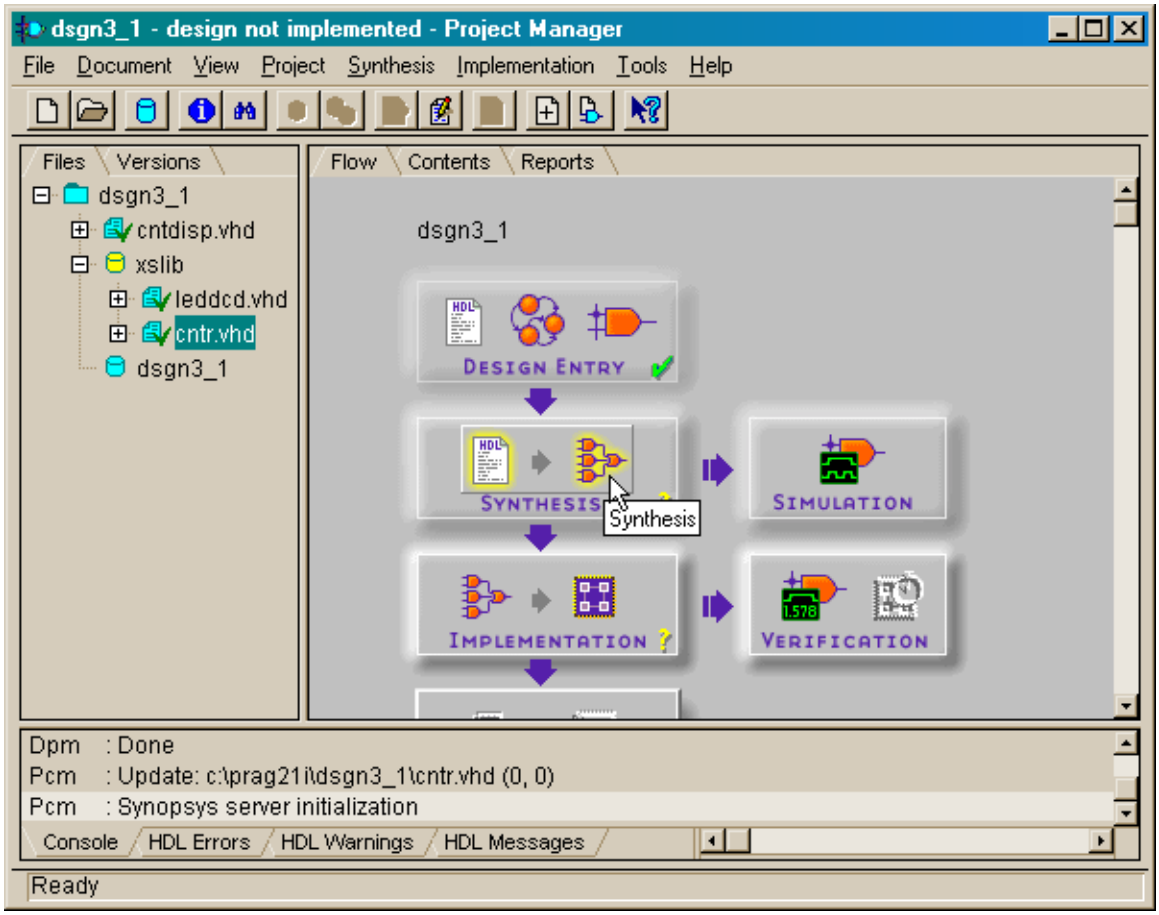
The root module is stored in the cntdisp.vhd file and then that file is added to the project hierarchy. With all the source files in place, we can check for any VHDL syntax errors by selecting the Synthesis➔Force Analysis of All Sources menu item. In this case there are no errors (as indicated by the green checkmarks by each source file name in the **Project Hierarchy** pane.) If errors were found, you could double-click the marked files to open them with the HDL Editor and make the necessary fixes.



© 2001 by XESS Corp.

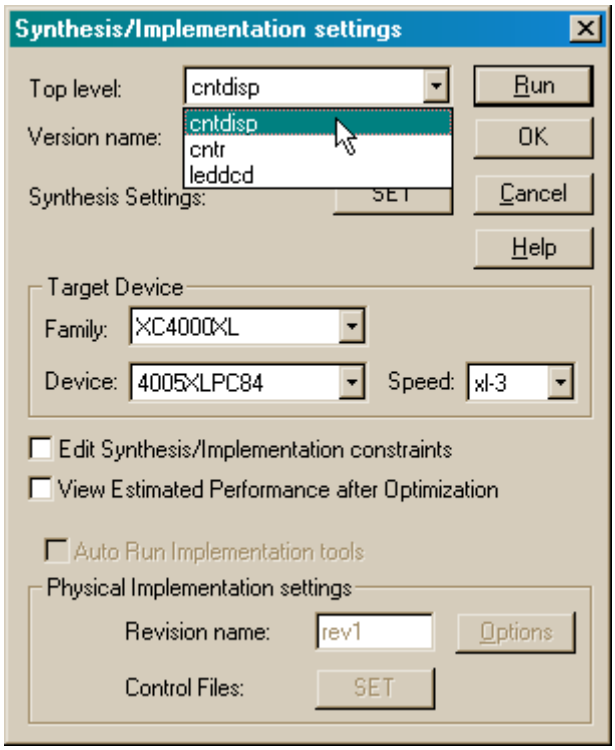
Synthesizing the Netlist

Once we know there are no syntax errors, we can run the synthesis tools to extract a netlist for the circuit.



© 2001 by XESS Corp.

We are going to target this design to the XS40-005XL Board so set the target device as shown below in the **Synthesis/Implementation settings** window. Then pull-down the list of modules in the Top level field and highlight the cntdisp entry. This tells the synthesizer tools that the **cntdisp** module is the root of the design.



After clicking on the Run button, the synthesis tools will process the VHDL in the three source files to create a netlist.

© 2001 by XESS Corp.

Assigning the I/O Signals to the FPGA Pins

Before mapping the synthesized netlist to the FPGA, we need to specify the pin assignments for the inputs and outputs of the circuit. The pins on the FPGA of the XS40 Board that are connected to the clock oscillator and the seven-segment LED are shown in Figure 8. We will also control the reset input of the circuit using the **D0** pin of the parallel port. That way we can reset the circuit using the PC attached to the XS40 Board.

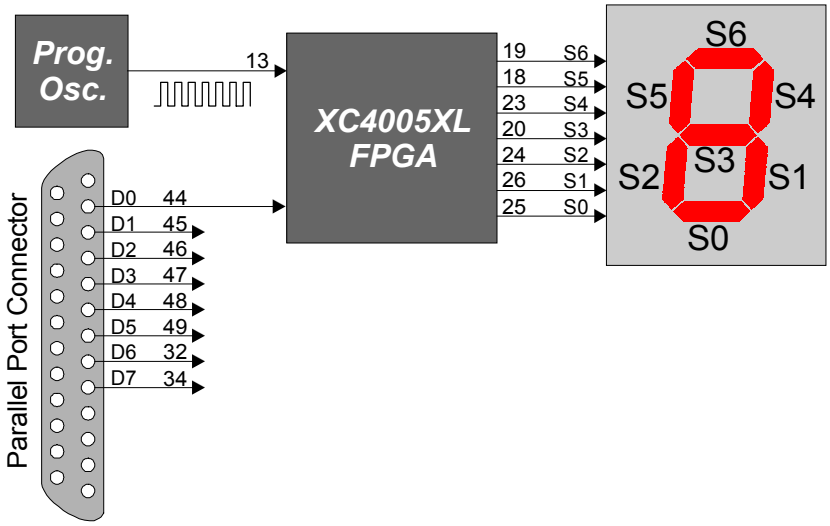
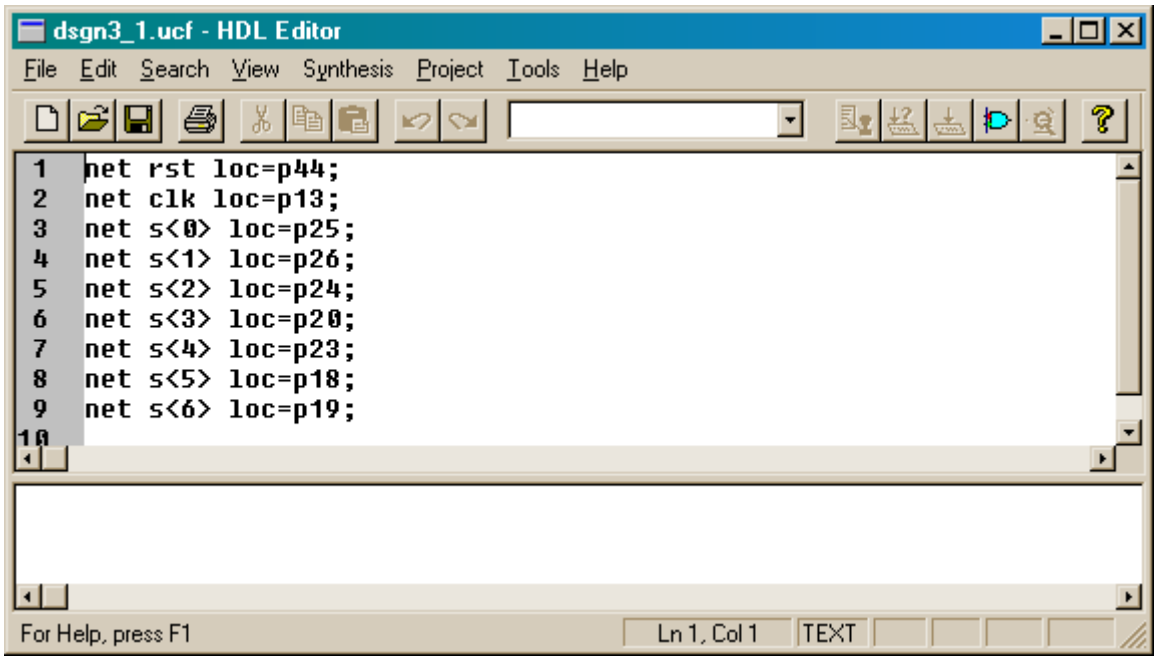


Figure 8: Connection of the programmable oscillator, parallel port, and LED digit to the pins of the FPGA on the XS40 Board.

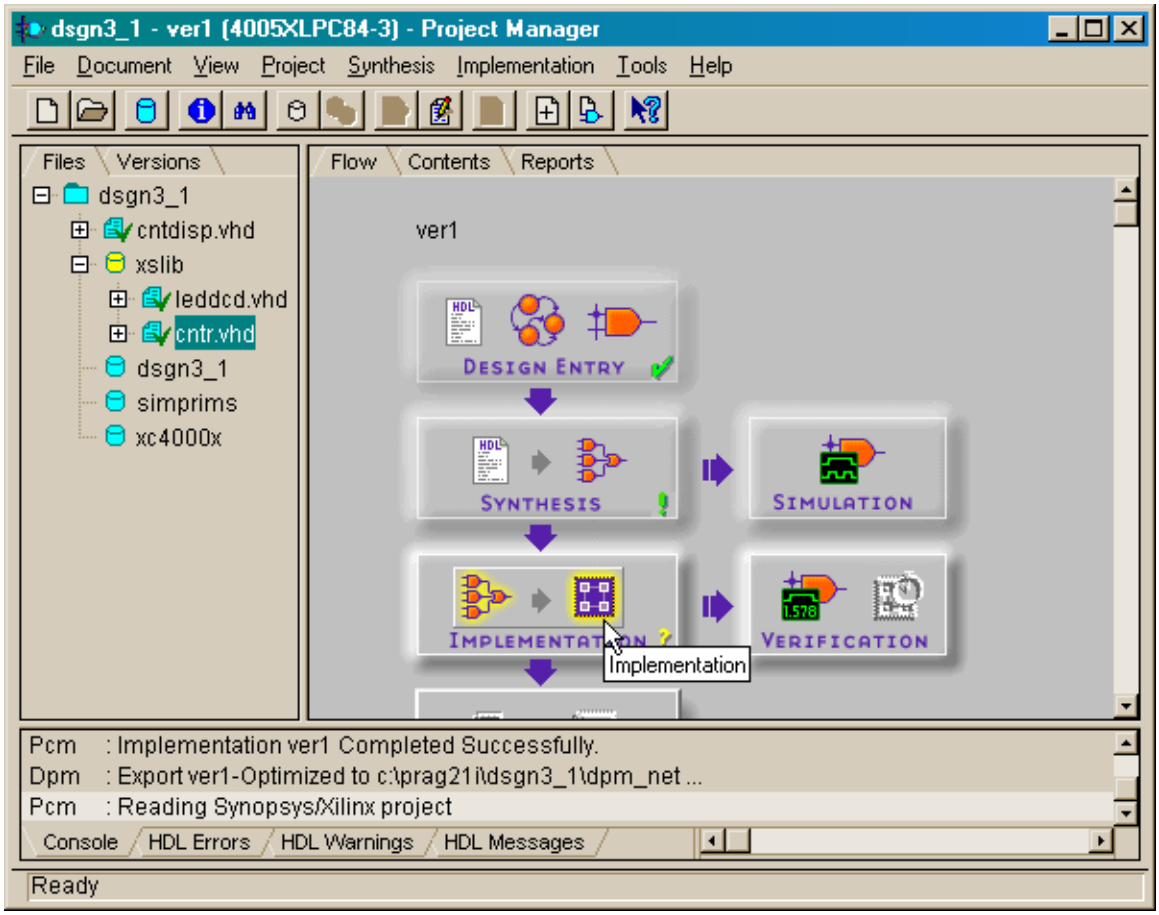
The pin assignments corresponding to Figure 8 are stored in the dsgn3_1.ucf file.



© 2001 by XESS Corp.

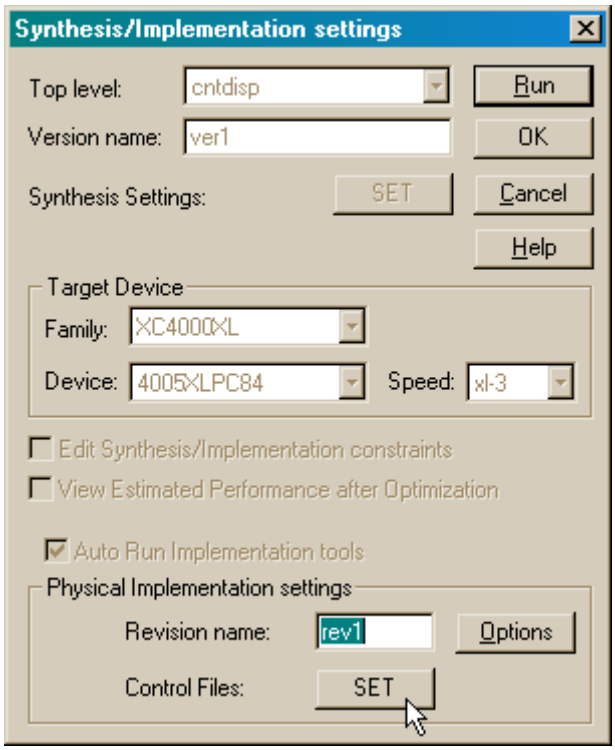
Implementing the Design

Once the pin assignments are in place, we can start the implementation tools.

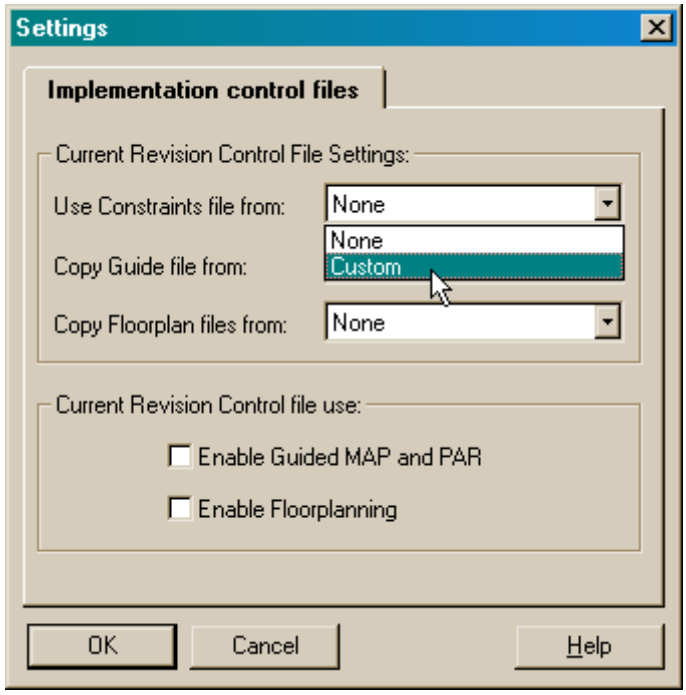


© 2001 by XESS Corp.

Press the SET button in the **Synthesis/Implementation settings** window so the location of the pin assignments can be specified.

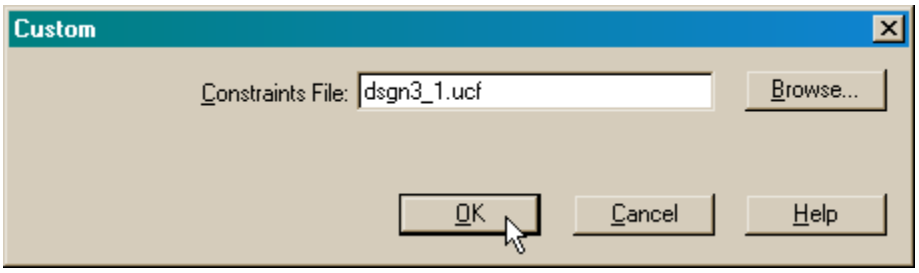


Select Custom from the Constraints file field of the **Settings** window.

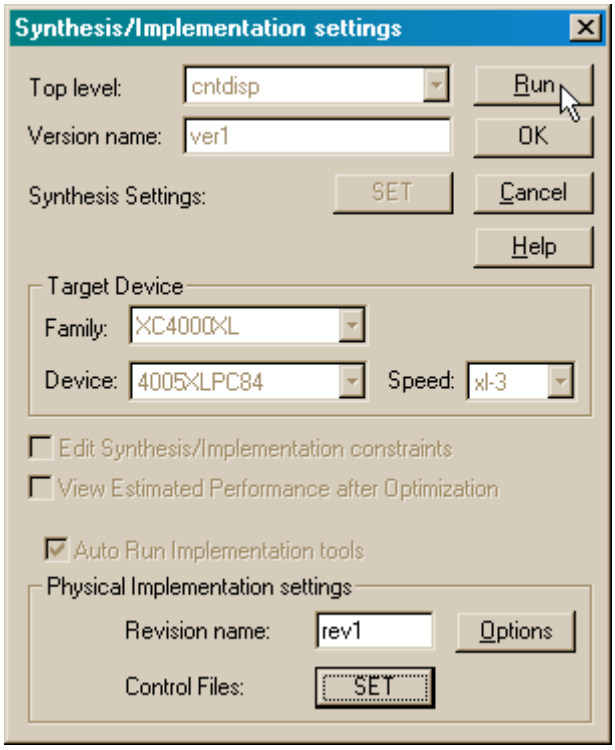


© 2001 by XESS Corp.

The name of the dsgn3_1.ucf file will already be listed in the Constraints File field of the **Custom** window that appears, so just click on the OK button.

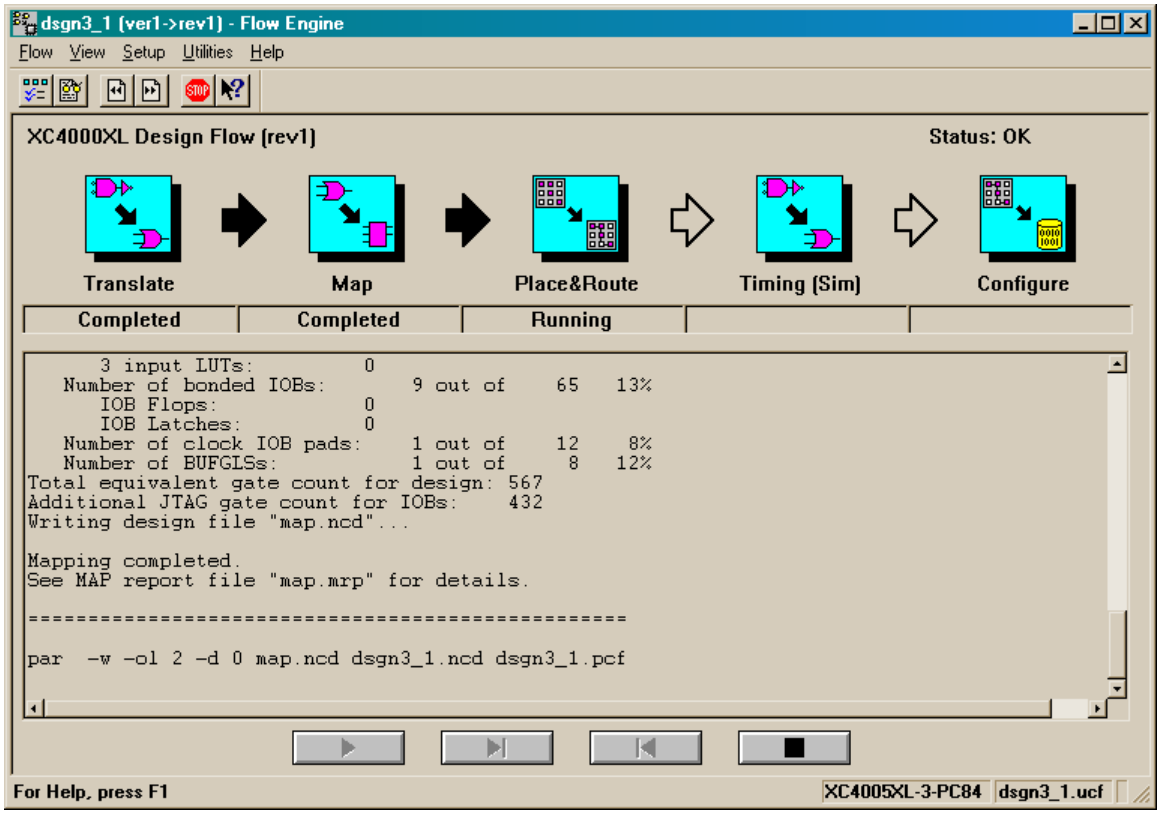


After returning to the **Synthesis/Implementation settings** window, click on Run to initiate the implementation process.



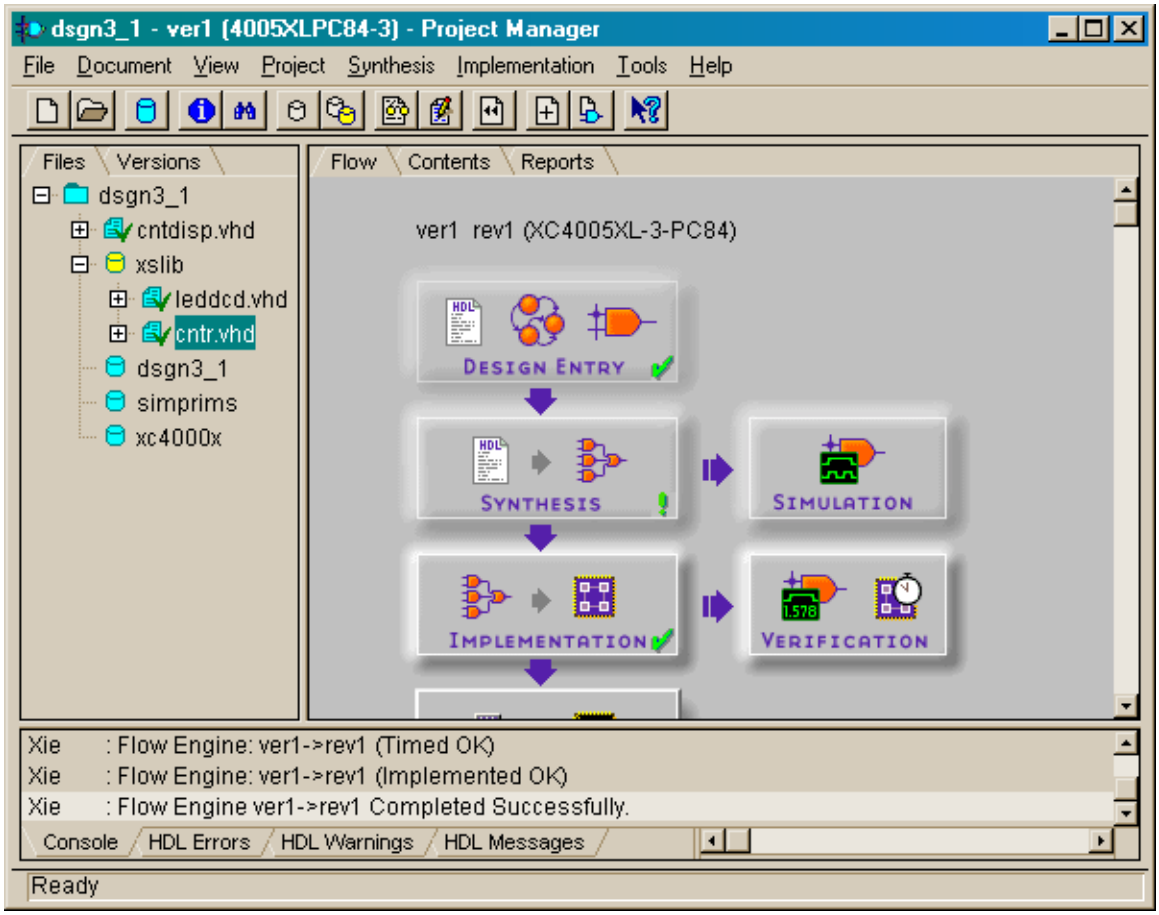
© 2001 by XESS Corp.

All the steps in the implementation process should complete without errors.



© 2001 by XESS Corp.

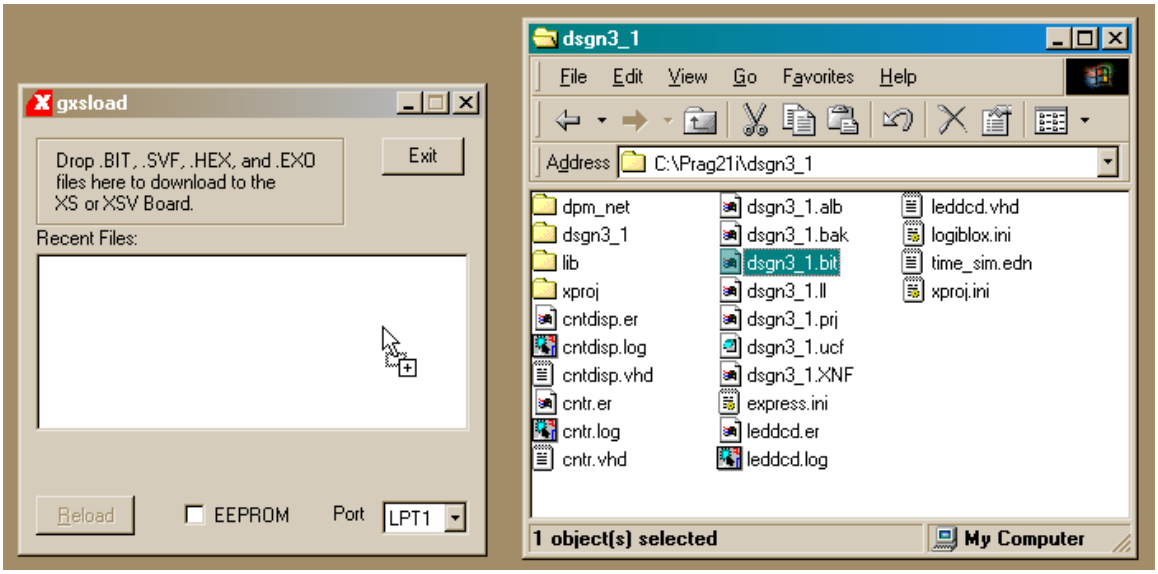
The Project Manager window should look as follows after the implementation tools have terminated successfully.



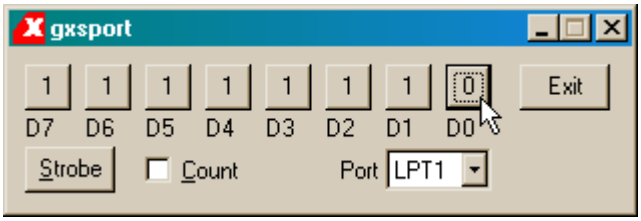
© 2001 by XESS Corp.

Downloading and Testing the Design

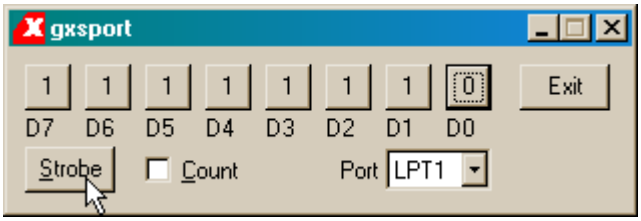
At this point, the final bitstream for downloading into the XS40-005XL Board is available. Open the directory containing the ***dsgn3_1*** project files and drag-and-drop the dsgn3_1.bit file into the **gxsload** window. The bitstream will download into the XS40 Board attached to the parallel port.



If pin D0 of the parallel port is at logic 1 after the downloading completes, the counter will be held in the reset state so only a static 0 is displayed. To release the reset, open the gxsport window and click on the D0 button until it displays a 0.



Then click on the Strobe button so the logic 0 value is output on the D0 pin of the parallel port.

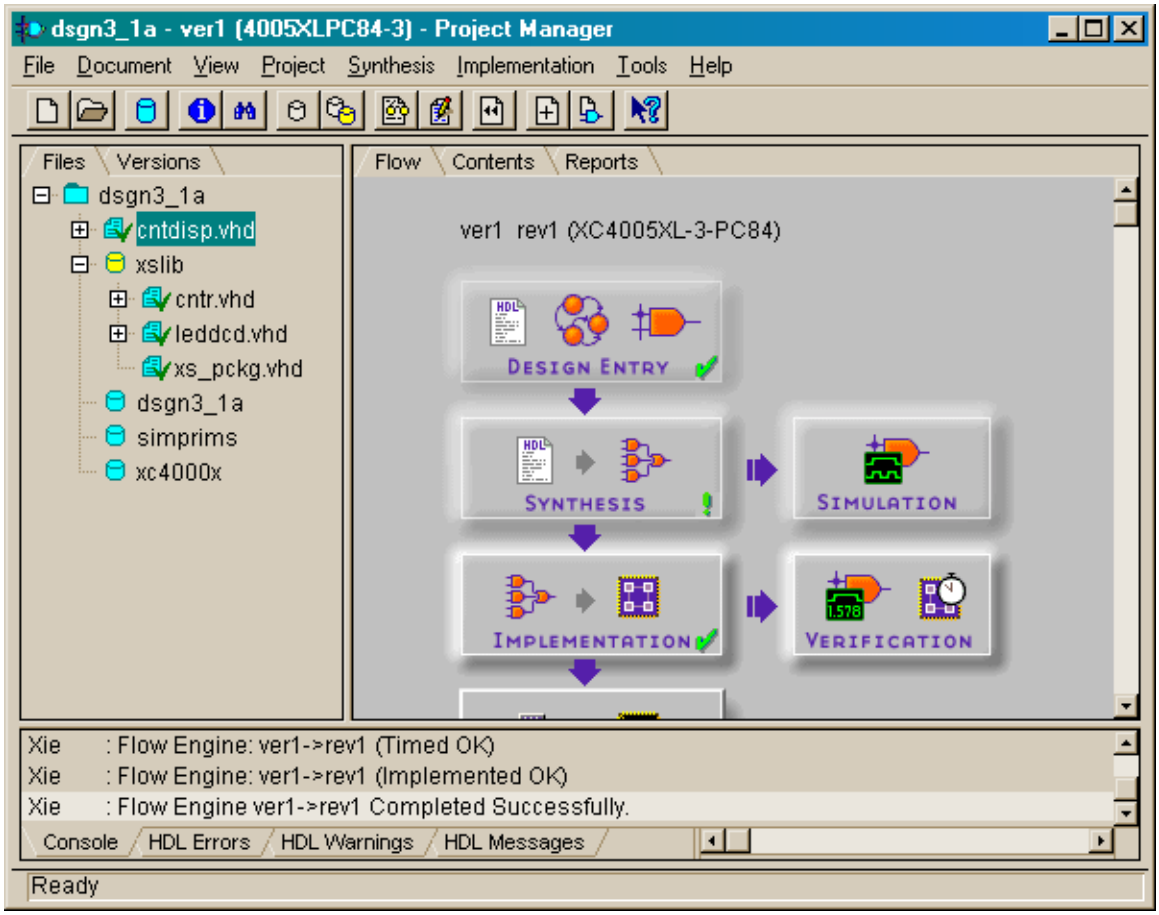


Now you should observe the seven-segment LED running through the sequence: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, b, C, d, E, F, … with each digit being displayed for roughly 1/3 seconds.

© 2001 by XESS Corp.

Consolidating the Packages

It can be inconvenient to place each module in a package and then have to explicitly include each package in the root module. Instead, you can create a single VHDL file that contains the package declarations from each of the other modules, and then just include this single module in the root. This was done for project ***dsgn3_1a*** as shown below.



The package declarations were removed from cntr.vhd and leddcd.vhd and the component declarations from each file were incorporated into a single package in the xs_pckg.vhd file (Listing 2).

Listing 2: Consolidated package declaration for the counter and LED decoder modules.

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.numeric_std.all;
4
5  package xs_pckg is
6
7  component leddcd
8      port (
9          d: in UNSIGNED (3 downto 0);
10         s: out STD_LOGIC_VECTOR (6 downto 0)
11     );
12 end component;
```

© 2001 by XESS Corp.

```
13
14  component cntr
15        generic (
16              LENGTH: natural         -- number of bits in counter
17        );
18      port (
19        rst: in STD_LOGIC;      -- synchronous reset
20          clk: in STD_LOGIC;    -- counter clock
21          cnt: out UNSIGNED(LENGTH-1 downto 0)    -- counter output
22      );
23  end component;
24
25  end xs_pckg;
```

Then the single `xs_pckg` package is included on line 4 in the root module (Listing 3).

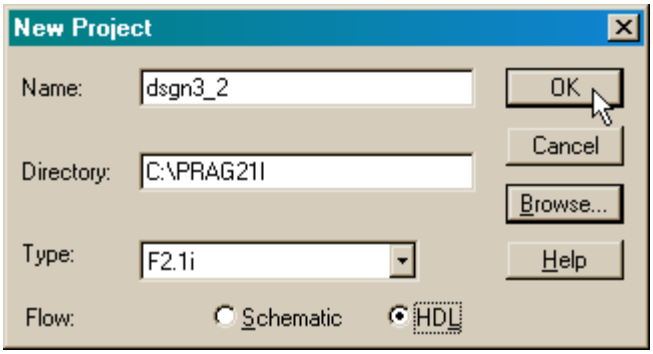
Listing 3: VHDL source for the root module of dsgn3_1a.

```
1   library IEEE,XSLIB;
2   use IEEE.std_logic_1164.all;
3   use IEEE.numeric_std.all;
4   use XSLIB.xs_pckg.all;
5
6   entity cntdisp is
7       port (
8         rst: in STD_LOGIC;      -- synchronous reset
9           clk: in STD_LOGIC;    -- counter clock
10          s: out STD_LOGIC_VECTOR(6 downto 0)     -- outputs to LED
11  segments
12      );
13  end cntdisp;
14
15  architecture cntdisp_arch of cntdisp is
16  constant length: natural := 28;
17  signal cnt: UNSIGNED(length-1 downto 0);
18  begin
19  u0: cntr generic map(LENGTH=>length)
20              port map(rst=>rst, clk=>clk, cnt=>cnt);
21  u1: leddcd port map(d=>cnt(length-1 downto length-4), s=>s);
22  end cntdisp_arch;
```

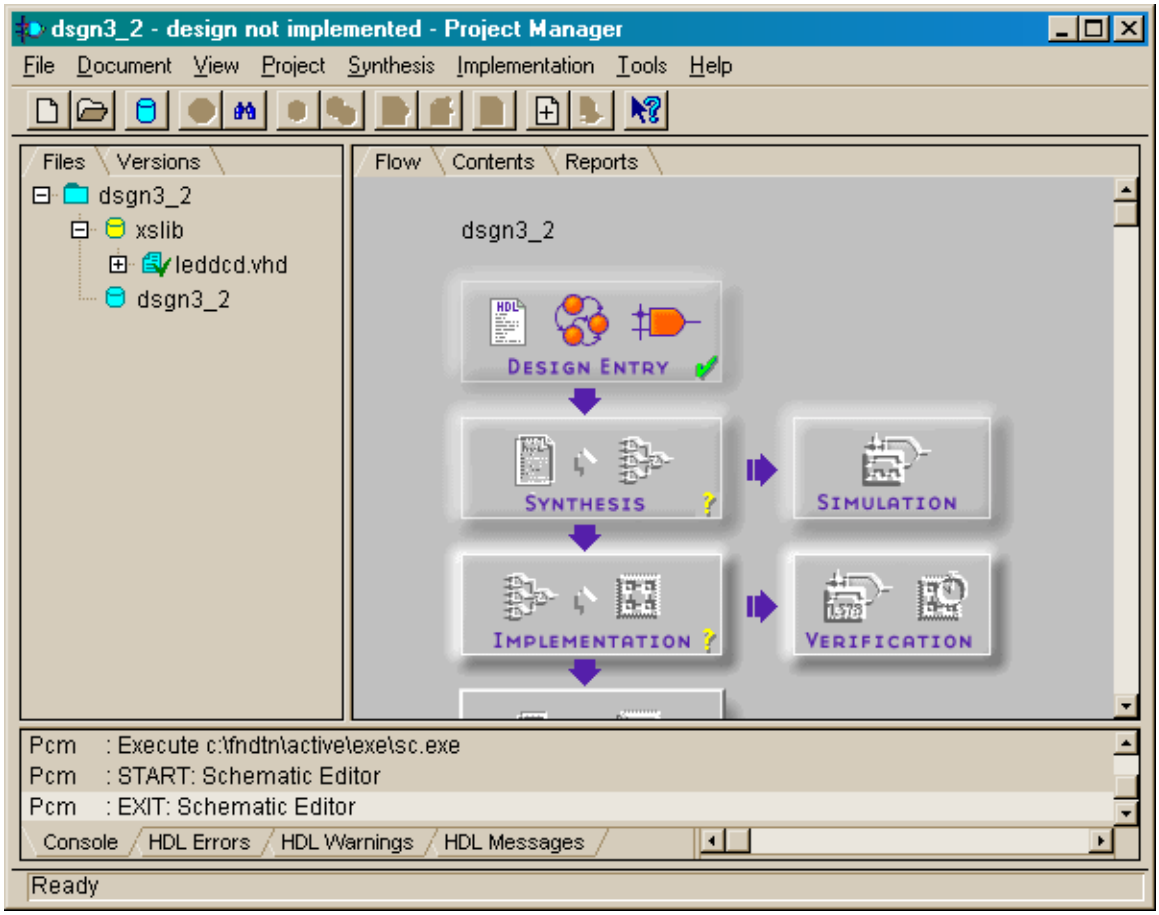
© 2001 by XESS Corp.

Hierachical VHDL Design with Schematic-Based Modules

In the ***dsgn3_2*** project, we will replace the cntr.vhd module with a counter desscribed by schematics. Once again, the design is initiated as an HDL-based project.



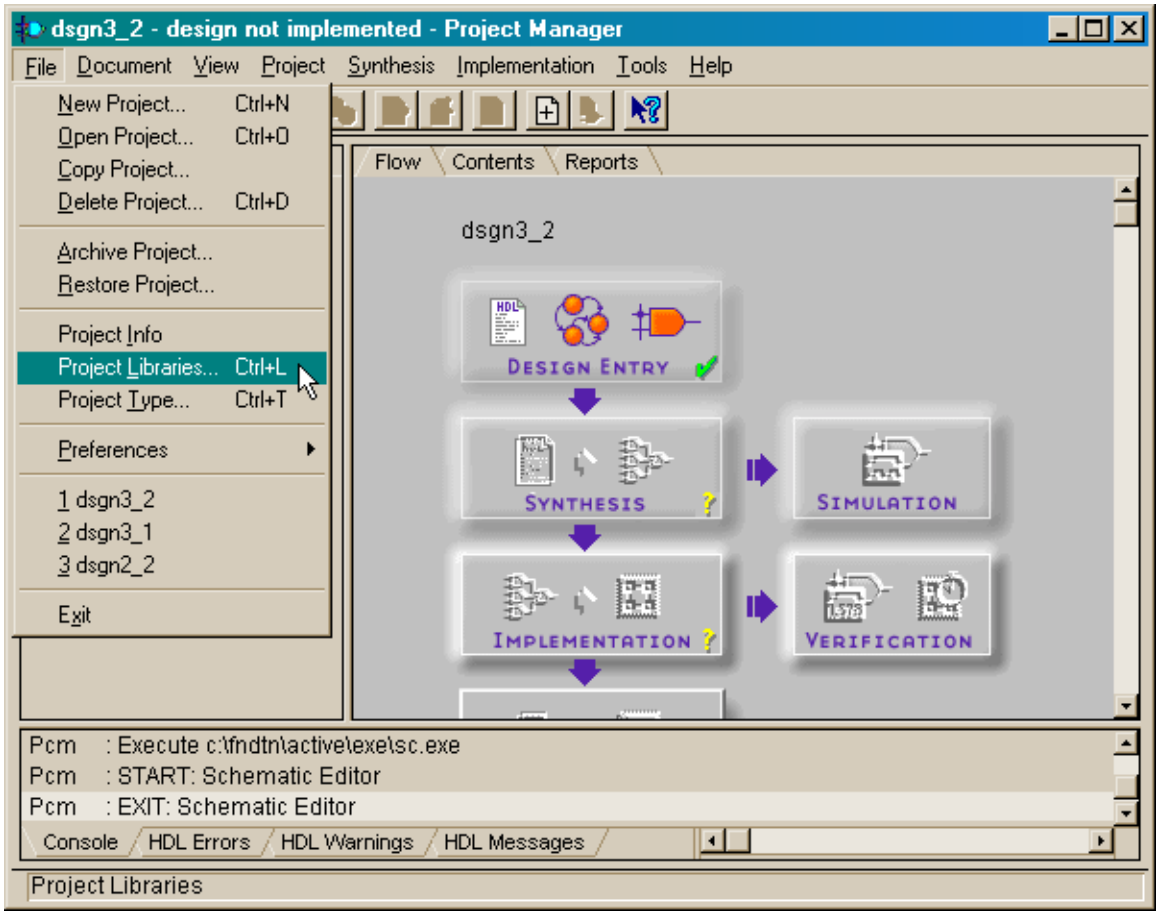
Then we can add the xslib library and add the leddcd.vhd file to it as we did in the ***dsgn3_1*** project.



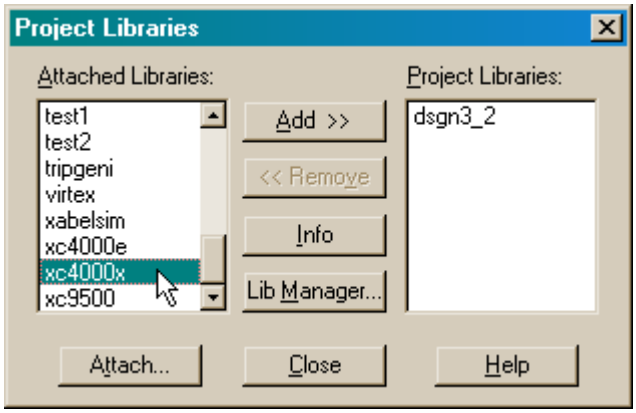
© 2001 by XESS Corp.

Adding a Predefined Library to the Project

Next we need to draw the schematic for the counter. But we need a library of parts with which to build the counter. Schematic part libraries are tied specific device families, so we need to add the appropriate part library to our project. Once again we will target the XS40-005XL Board with the XC4005XL FPGA. To add the library for this device, select the File➔Project Libraries... menu item.

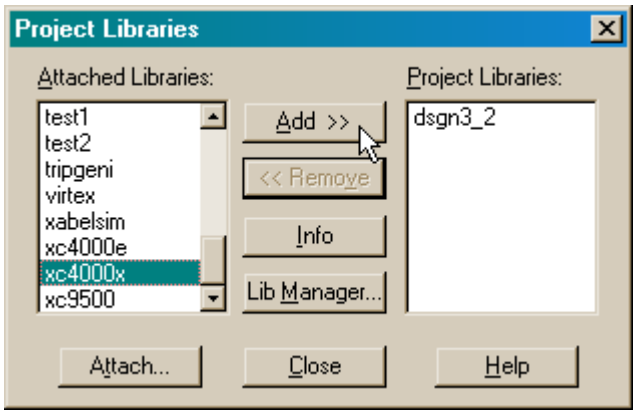


Scroll down in the list of libraries in the **Project Libraries** window and highlight the xc4000x entry.

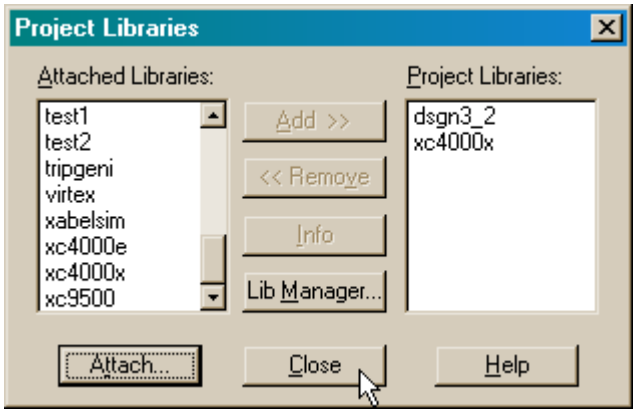


© 2001 by XESS Corp.

Then click the Add>> button to copy the xc4000x library of parts to the list of project libraries.

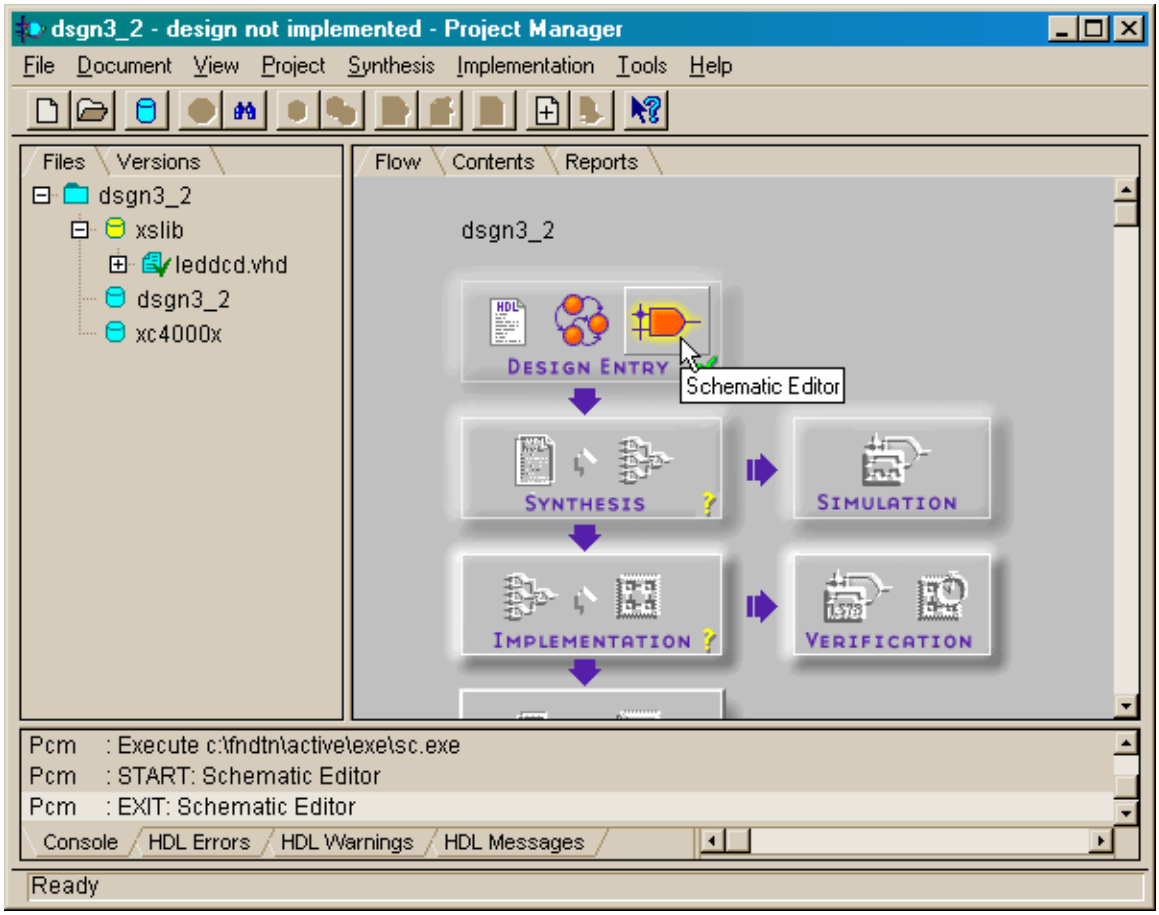


This is the only library we need for this project, so click on the Close Button.



© 2001 by XESS Corp.

Notice that now the xc4000x library icon now appears in the **Hierarchy** pane of the **Project Manager** window. We can now open the **Schematic Editor** window and begin to design the counter module.

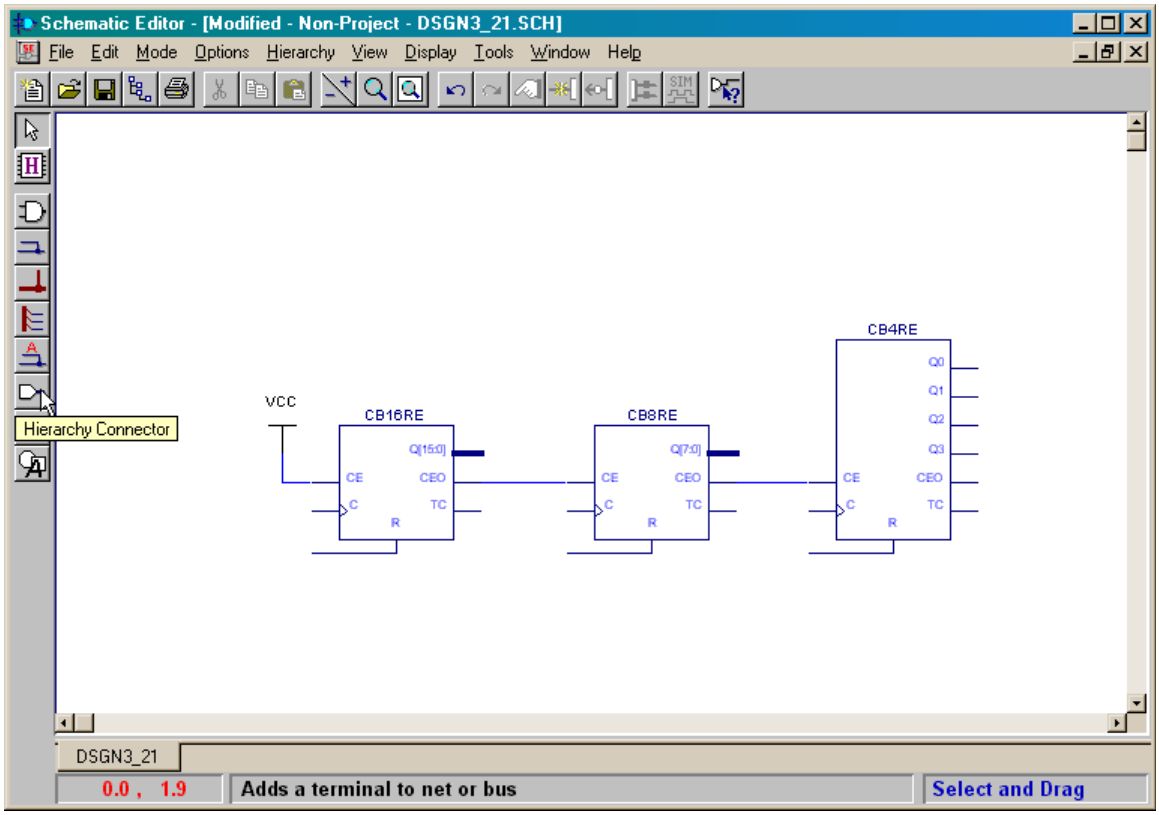


© 2001 by XESS Corp.

Drawing the Lower-Level Counter Schematic

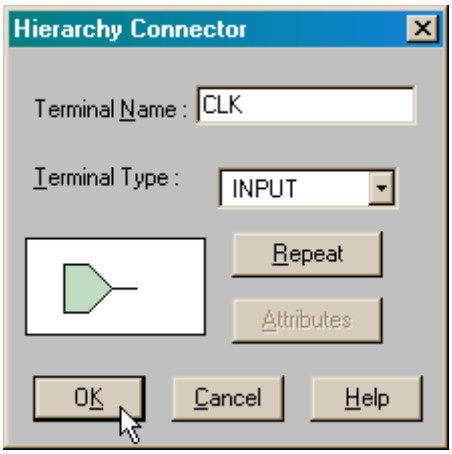
In the **Schematic Editor** window, we begin by adding a sixteen-bit counter (CB16RE), an eight-bit counter (CB8RE), and a four-bit counter (CB4RE) to get a total length of 28 bits. Then the clock-enables of the counters are connected as we did in the example in Chapter 2. All we have left to do is add the inputs and outputs to the counter. We will not use IPADs or OPADs for this since these correspond to pins on the actual FPGA device. It is better to use hierarchical connectors or terminals for I/O into or out of a lower-level module and then place all pin connections in the root module. That way we can defer the decision as to which signals enter and exit the chip when we design the top-level module.

To begin placing I/O terminals, click on the Hierarchy Connector button.

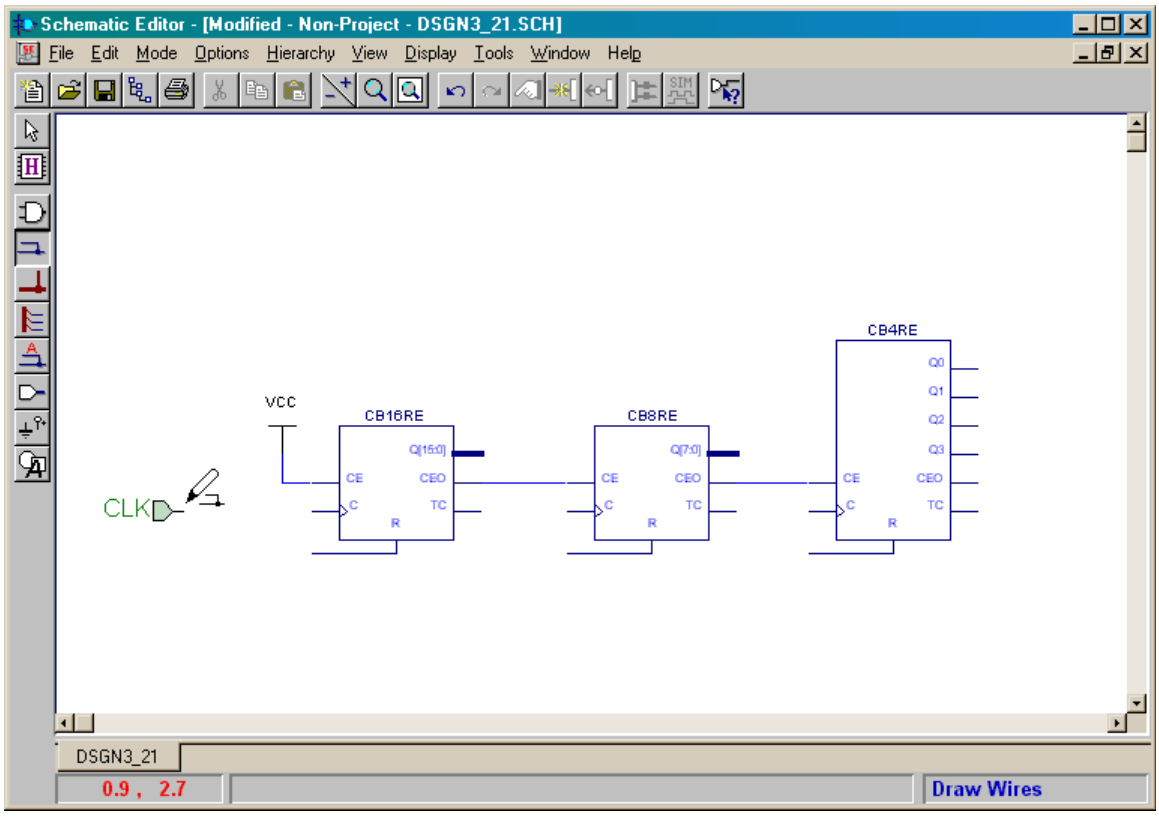


© 2001 by XESS Corp.

The **Hierarchy Connector** window will appear. The first input we will add is for the clock. Type the name of the input (CLK) into the Terminal Name field and click on the OK button.

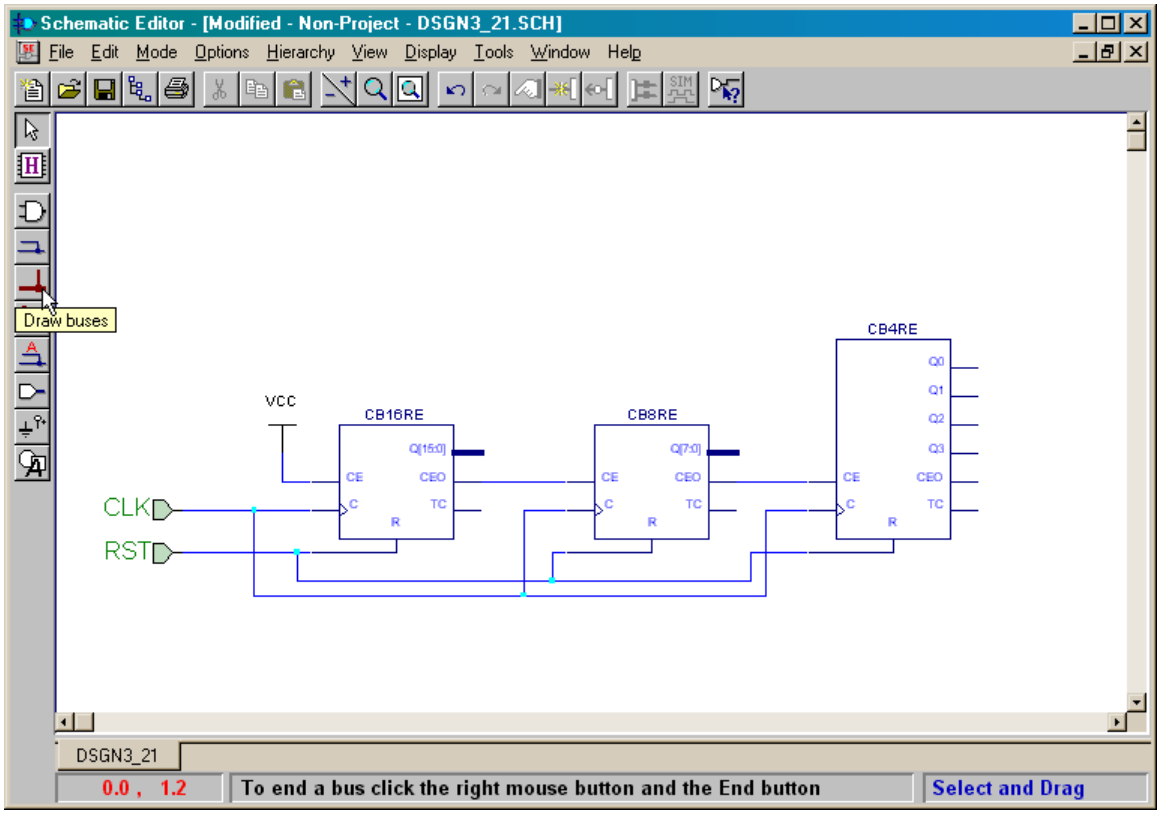


Then click in the drawing area of the **Schematic Editor** window and the input terminal will appear.



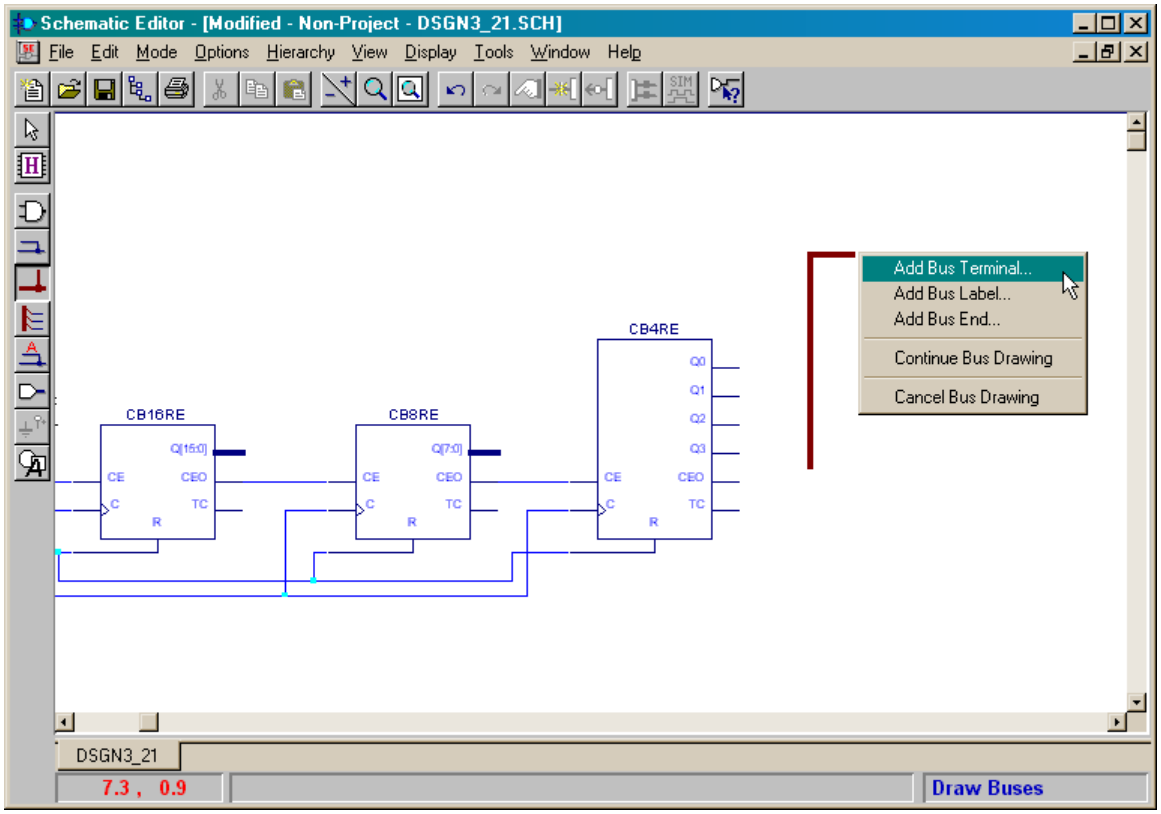
© 2001 by XESS Corp.

Repeat this procedure to add the reset input terminal (RST). Then wire these terminals to the clock and reset inputs of the counters as shown below. This takes care of the inputs to the 28-bit counter. Now we will use a bus to get the outputs from the counter.

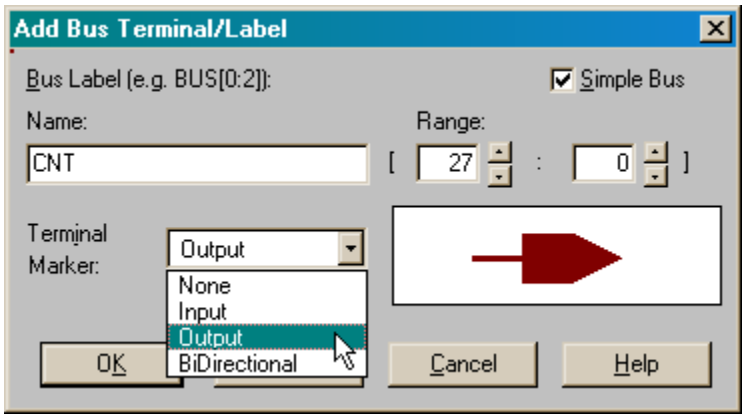


© 2001 by XESS Corp.

Start by drawing a bus at the far right of the schematic. When the last point of the bus is drawn, right-click the mouse and select Add Bus Terminal... from the pop-up menu that appears.

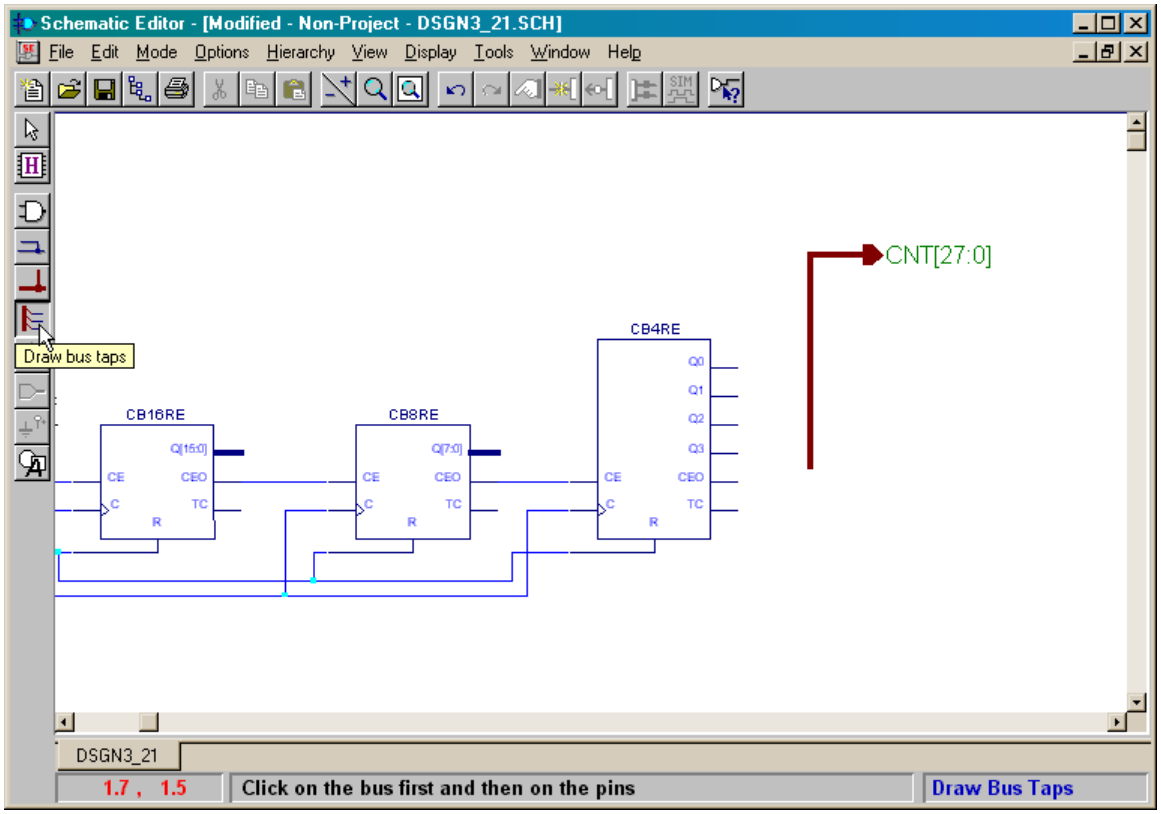


Type the bus name (CNT) into the Name field of the **Add Bus Terminal/Label** window that appears. Also set the upper and lower indices of the bus range to 27 and 0, respectively, to set the bus width to 28 bits. Finally, specify that this is an output bus by selecting Output from the Terminal Marker drop-down list. Then click on the OK button.



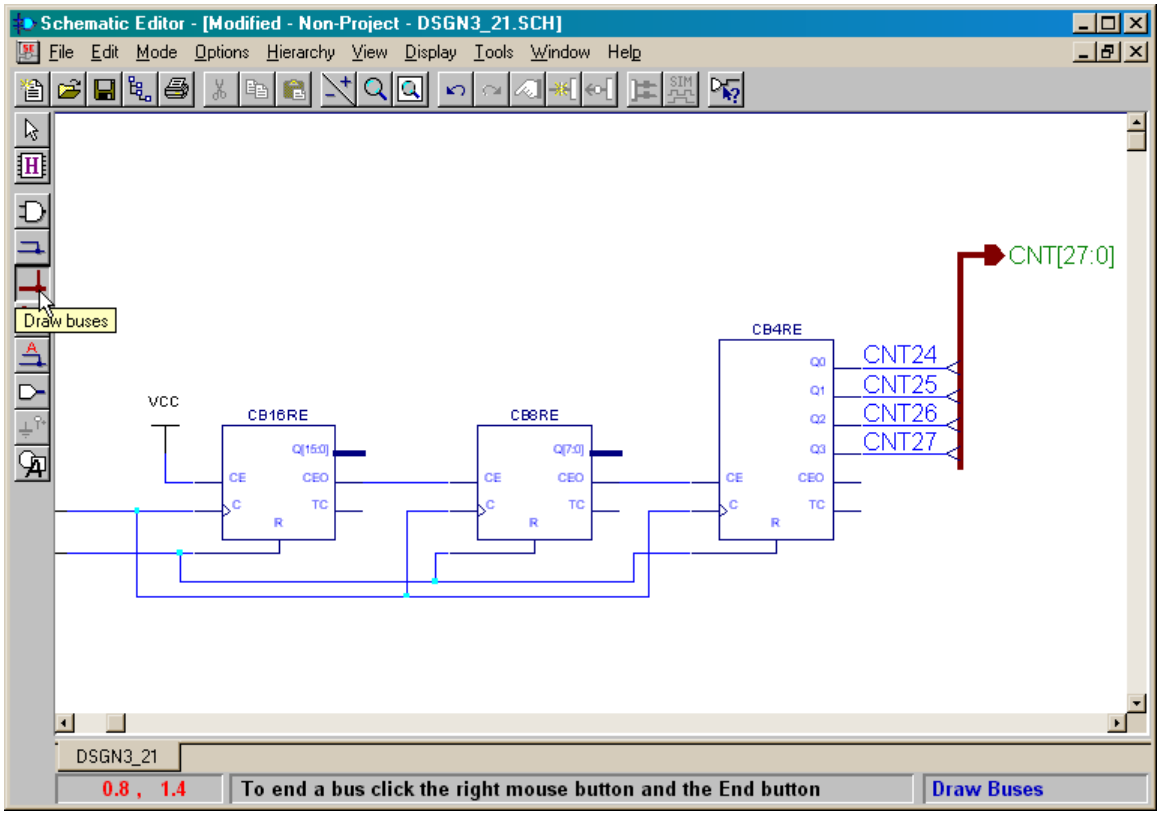
© 2001 by XESS Corp.

The name of the bus and its upper and lower indices will appear in the schematic. Now we need to tap the upper four bits of this bus and connect them to the four-bit counter. Click on the Draw bus taps button to start this operation.



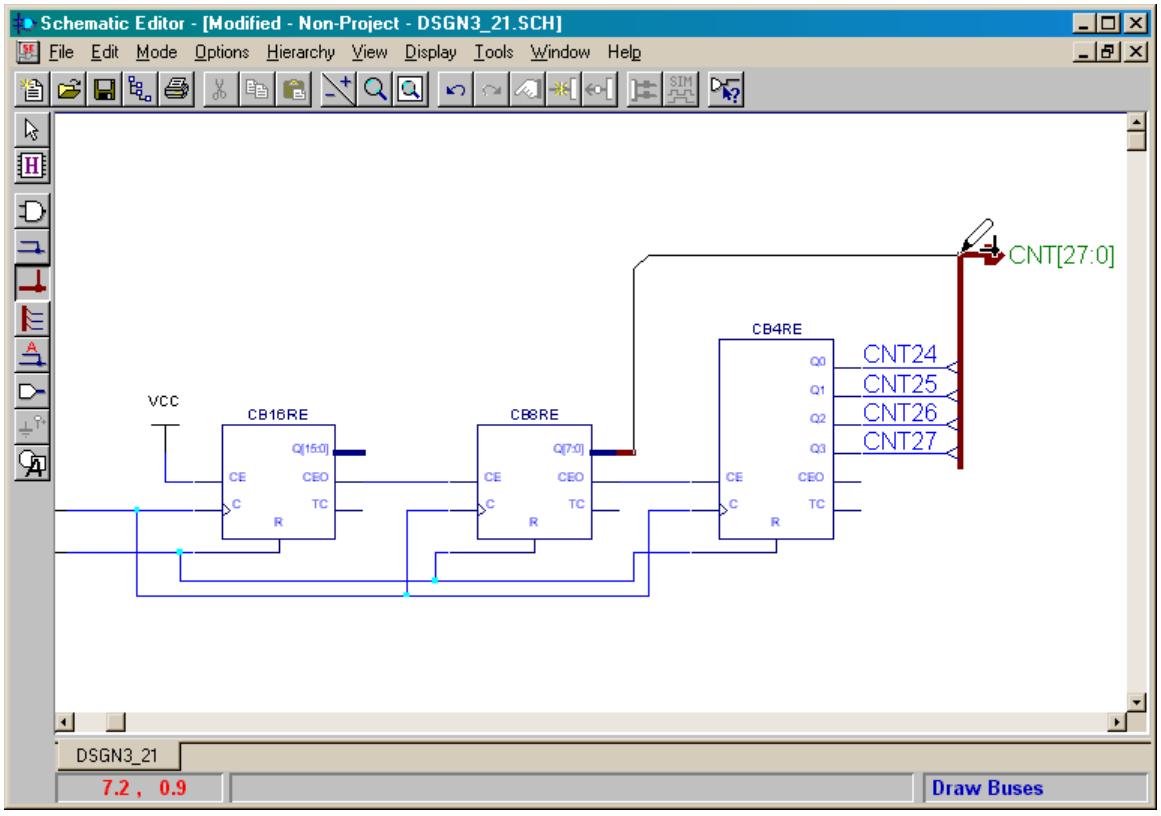
Then click on the CNT bus to select the bus that will be tapped. Create the individual taps by clicking on the Q3, Q2, Q1, and Q0 outputs (in that order) of the CB4RE counter.

The eight and sixteen-bit counters will be connected to the output terminals using buses, so click on the Draw buses button.



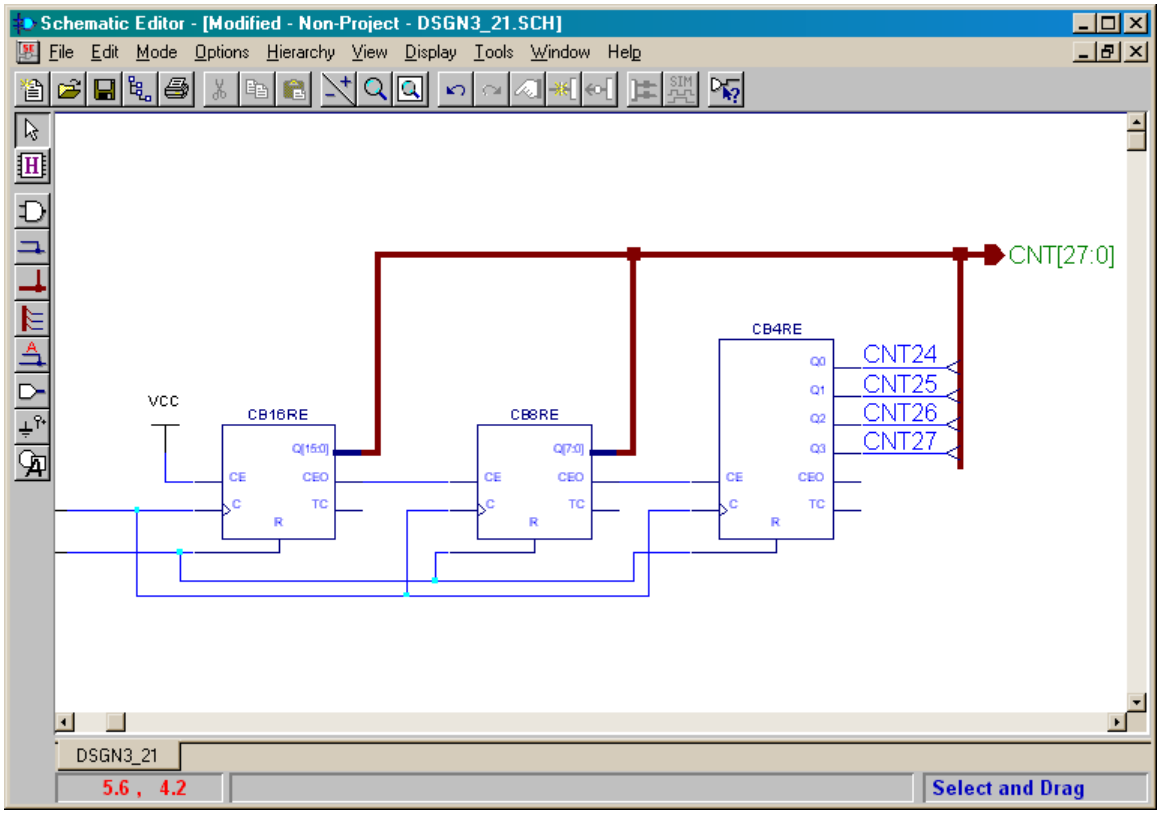
© 2001 by XESS Corp.

Draw a bus from the output bus of the eight-bit counter to the CNT bus as shown below.

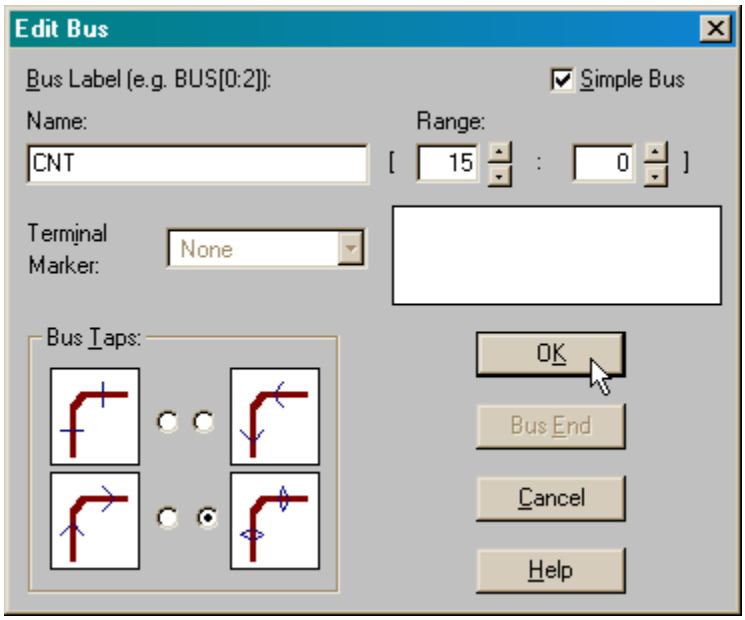


© 2001 by XESS Corp.

Repeat this operation to draw a bus from the output of the sixteen-bit counter to the CNT bus. Now the question arises: "Which of the 28 bus lines are the sixteen-bit and eight-bit counter outputs connected to?" There are some implicit rules that govern this, but it is clearer if we explicitly specify the bus connections. To do this, double-click on the bus connected to the CB16RE counter.

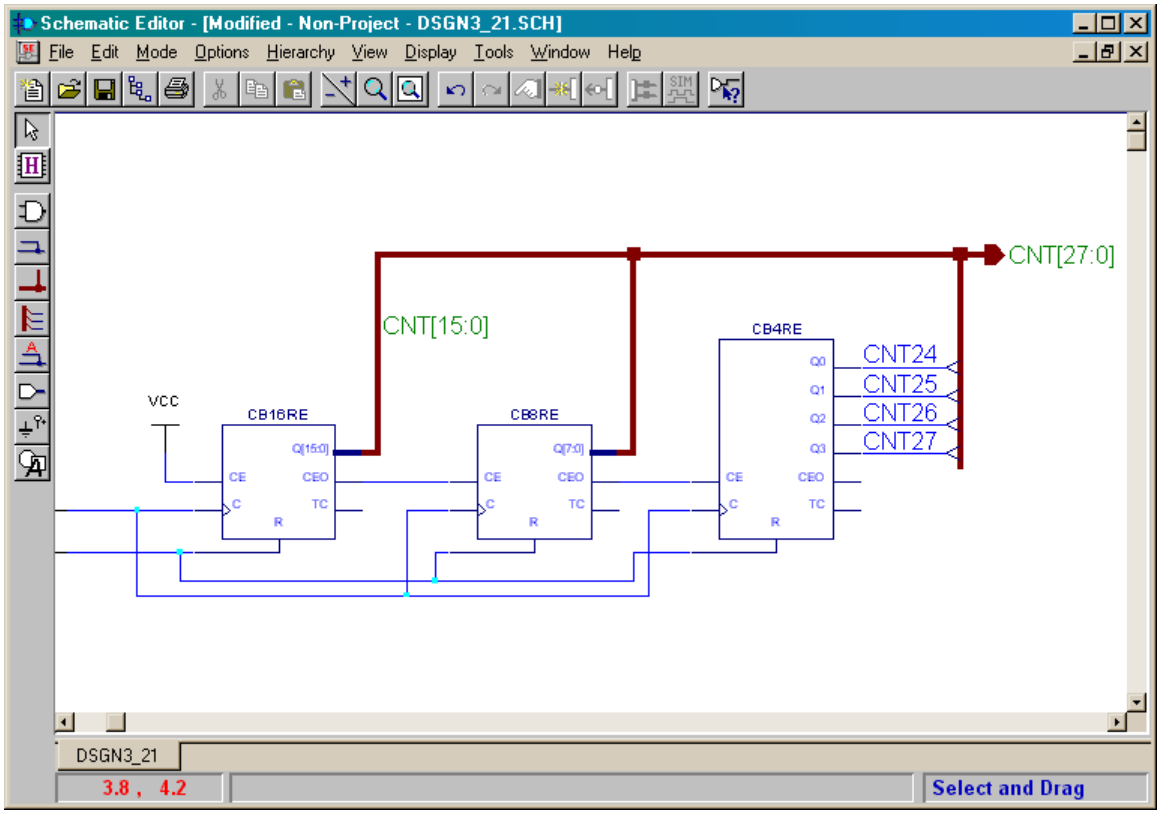


The **Edit Bus** window will appear. Specify the upper and lower index of the bus as 15 and 0, respectively. Then click on the OK button. This will connect the sixteen outputs of the CB16RE counter to the lower sixteen bits of the 28-bit CNT bus.



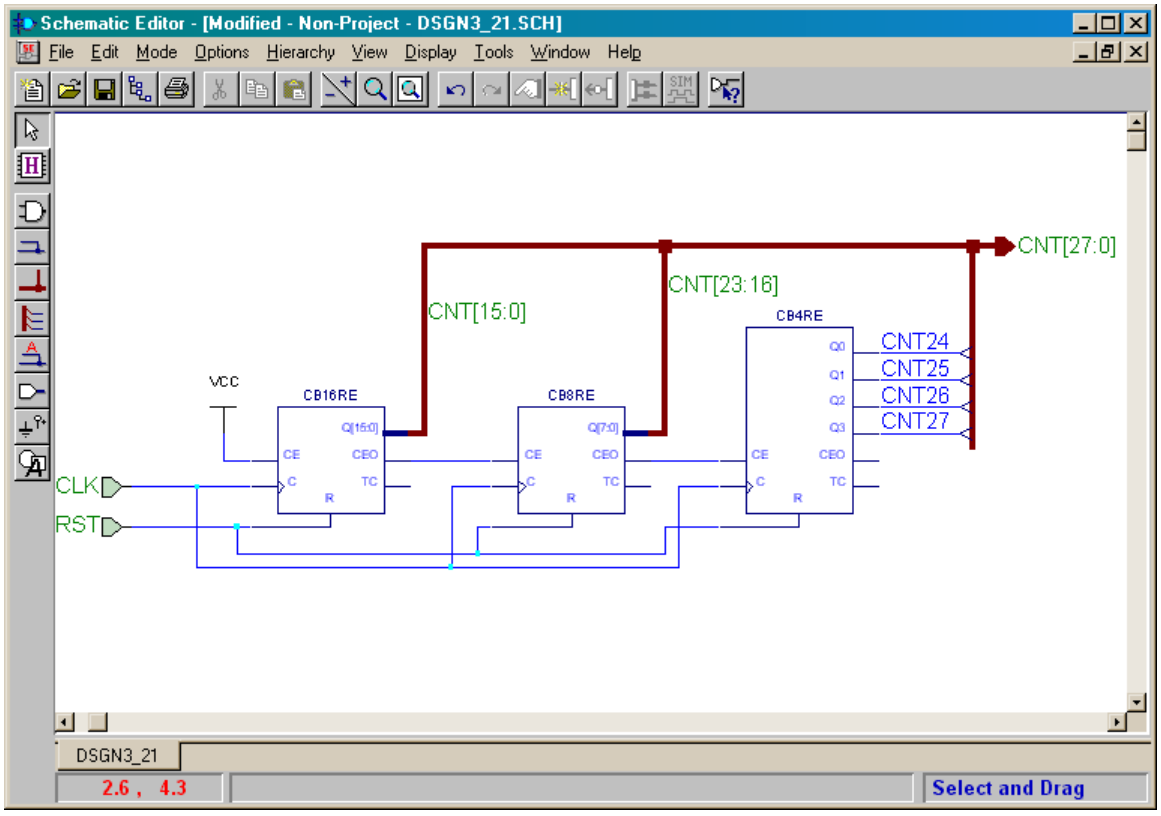
© 2001 by XESS Corp.

The index range of the bus connected to the sixteen-bit counter will now appear in the schematic.



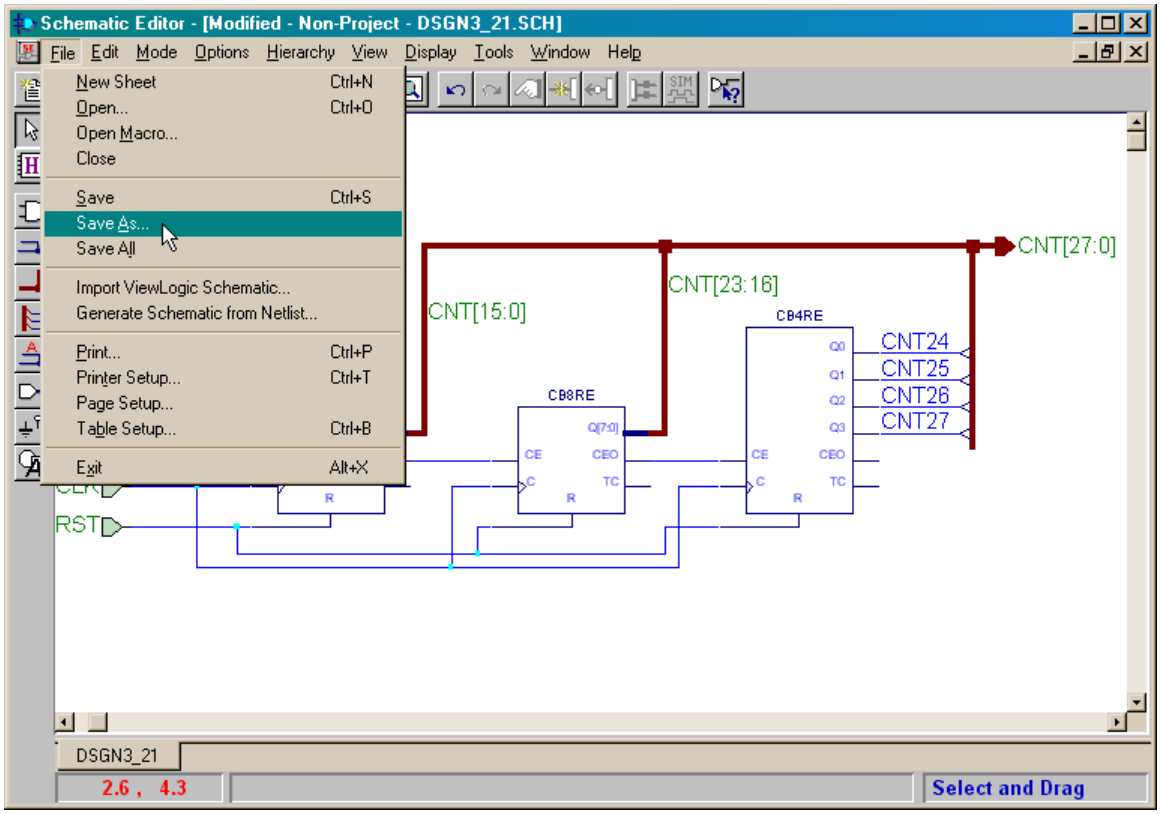
© 2001 by XESS Corp.

We can repeat this procedure to specify the bus connections for the eight-bit counter as indicated below. Now all the bits in the output bus are connected to the counters.

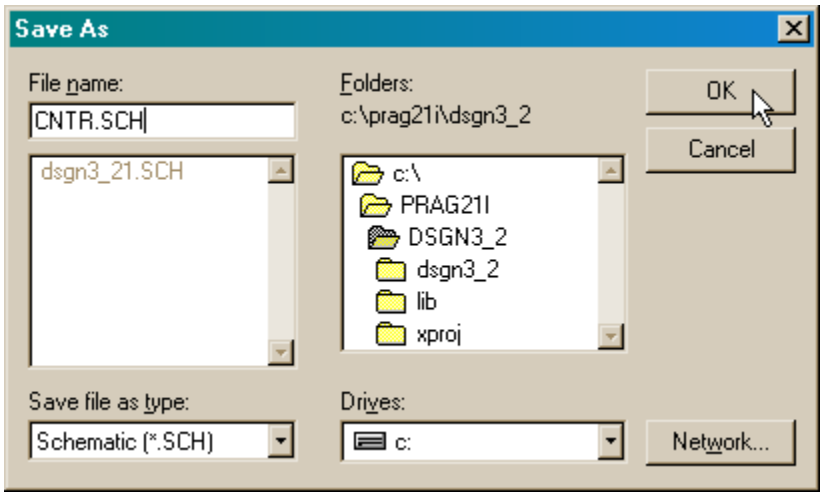


© 2001 by XESS Corp.

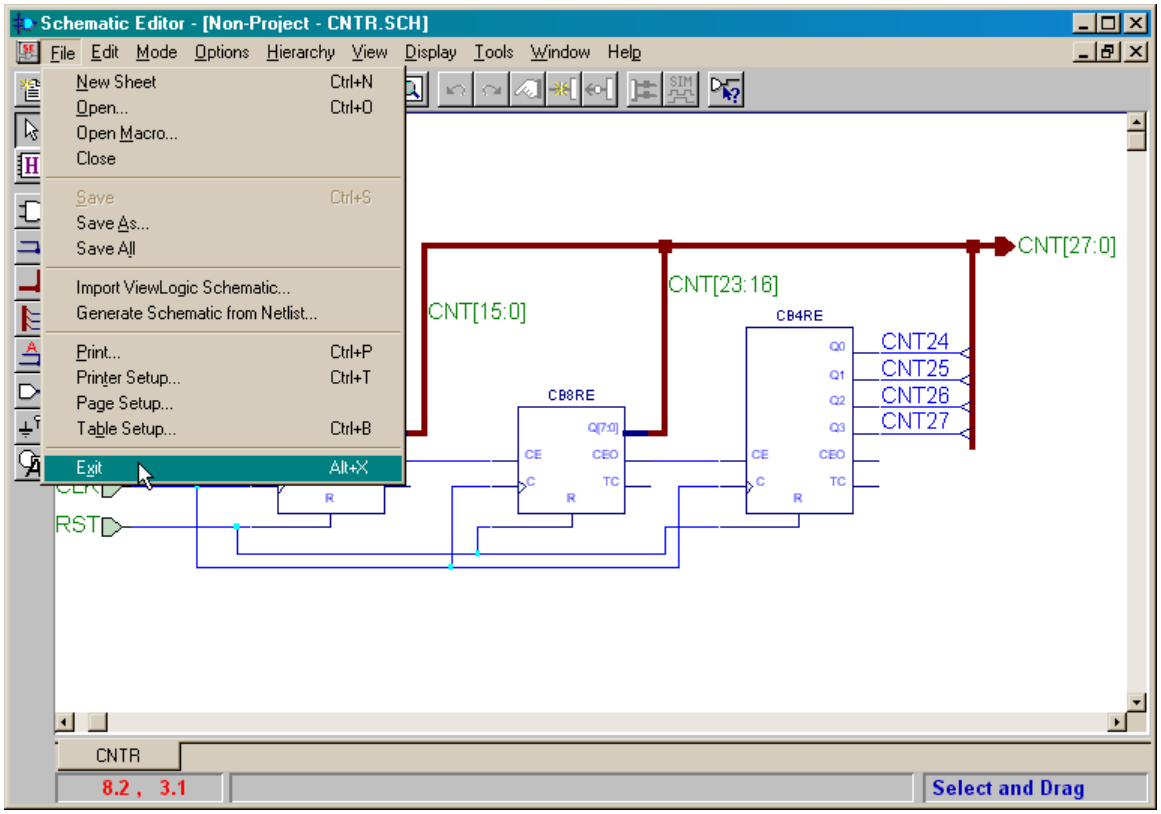
At this point we should save the schematic.



Set the name of this module to CNTR and then click on the OK button.



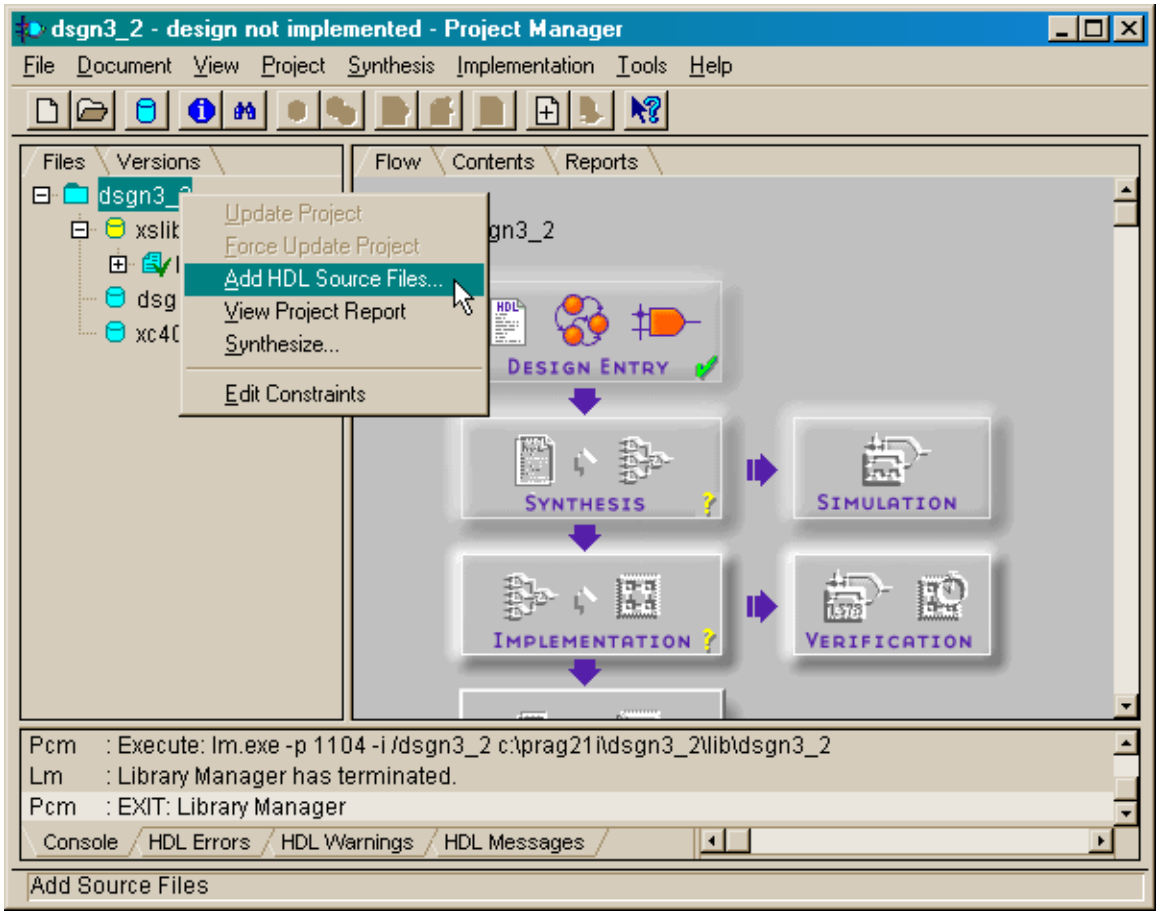
Finally, we can exit the **Schematic Editor** window.



© 2001 by XESS Corp.

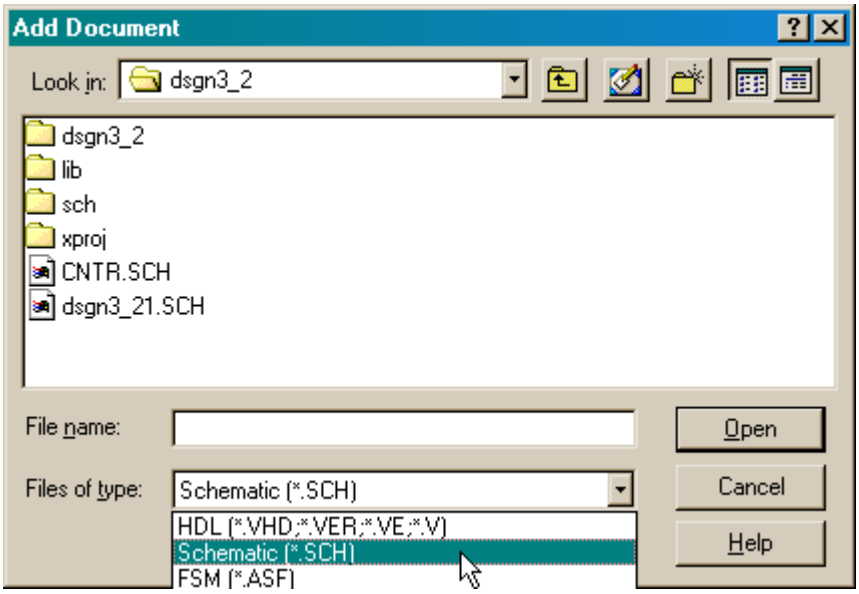
Adding the Lower-Level Counter Module to the Project

Once we are back in the **Project Manager** window, we can add the counter schematic to the project by clicking on the dsgn3_2 icon in the **Hierarchy** pane and selecting Add HDL Source Files... from the pop-up menu. (I know the counter is a schematic and not an HDL file, but this works anyway.)



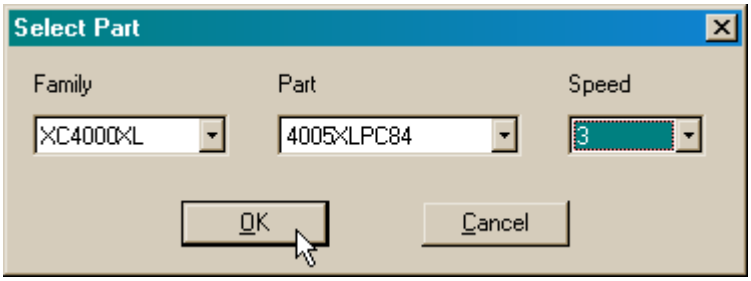
© 2001 by XESS Corp.

In the **Add Document** window, select Schematic (*.SCH) in the drop-down list attached to the Files of type field. Then highlight the CNTR.SCH file and click on the Open button.

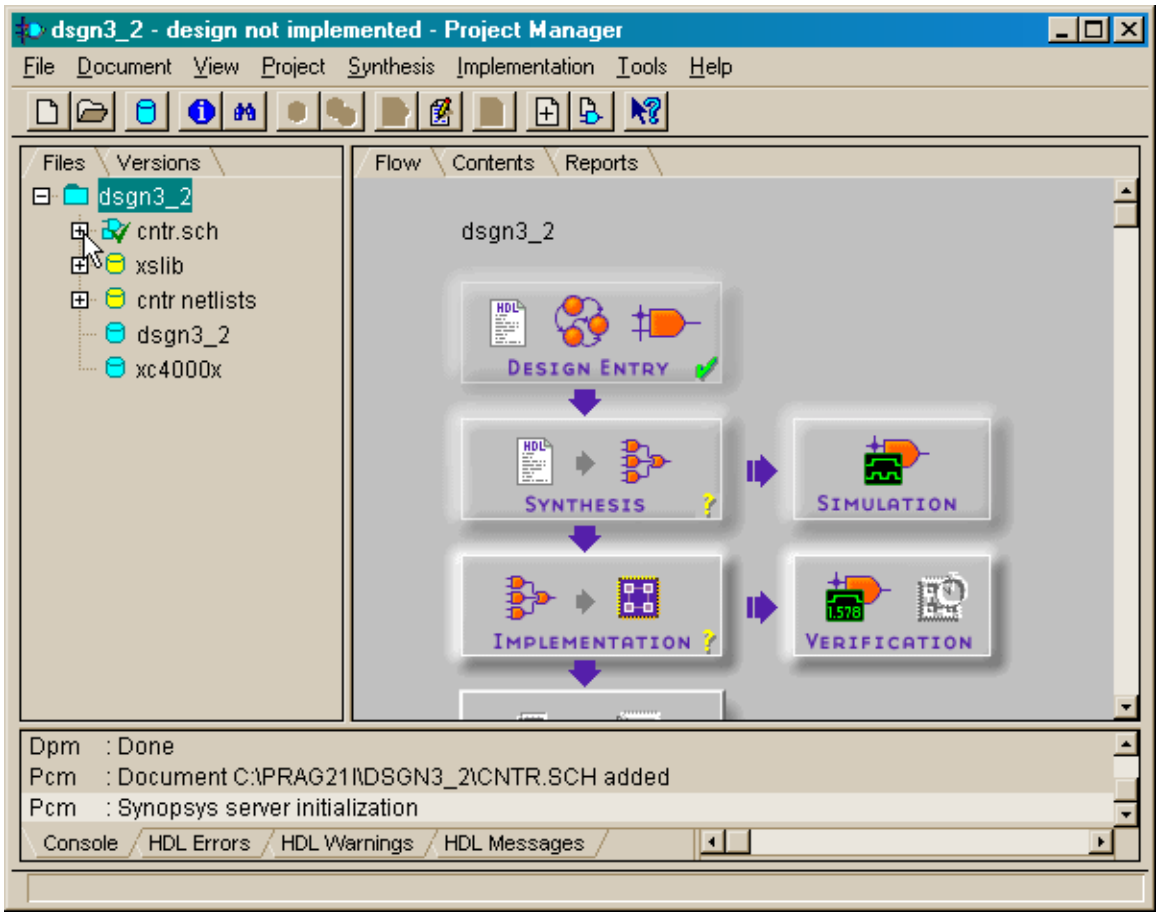


© 2001 by XESS Corp.

Before adding the CNTR.SCH schematic to the project, Foundation will ask you to specify the target device for the schematic. As we stated before, we are targeting the XS40-005XL Board so set the device information as shown below.

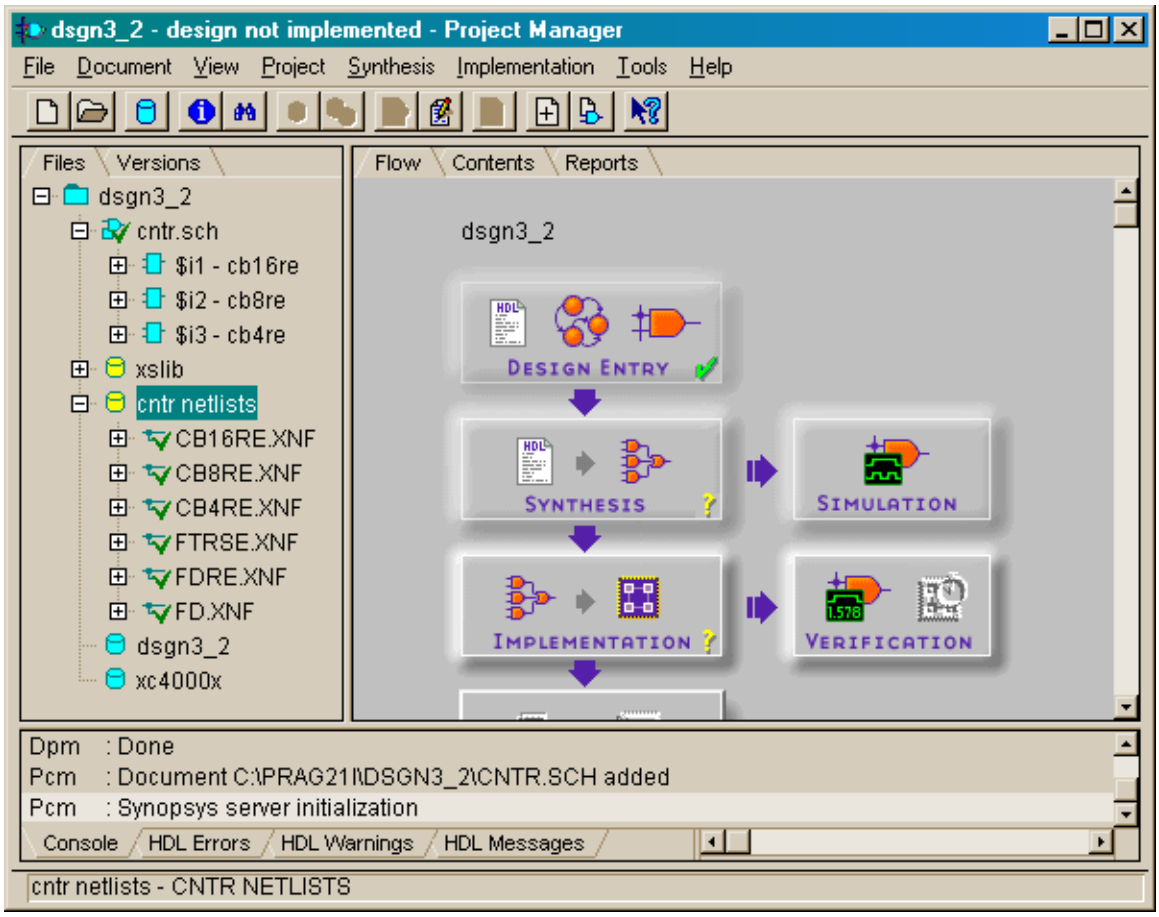


After clicking on OK in the **Select Part** window, the Foundation software will extract the netlist from the schematic and add the cntr.sch schematic and cntr netlists library to the project. Clicking the **+** signs to the left of these elements will expand them so we can see their contents.



© 2001 by XESS Corp.

We note that the cntr.sch element lists the names of the four, eight, and sixteen-bit counters as its subcomponents as we would expect. The cntr netlists element also lists the Xilinx netlist files (XNF) for these counters as subcomponents as well as the XNF files for the toggle and D flip-flops that make up the counters.



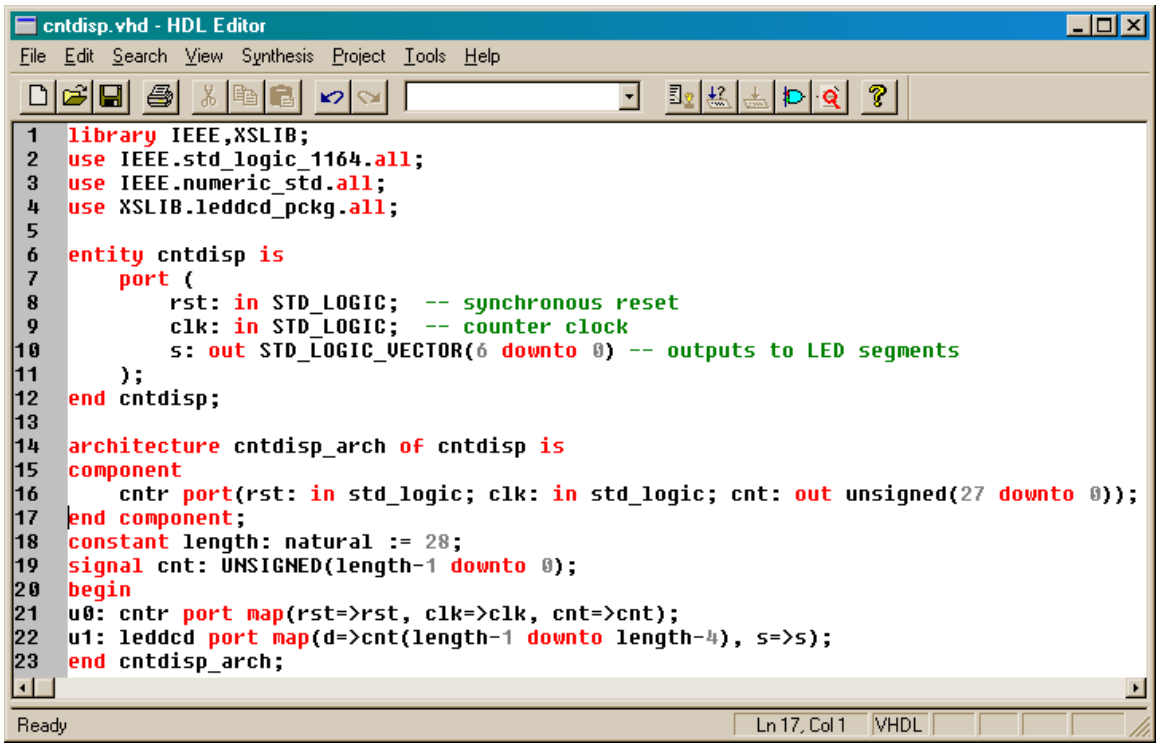
Modifications to the VHDL Code of the Root Module

The VHDL for the top-level root module is entered in the **HDL Editor** window as shown below. The main differences between this root module and the one from ***dsgn3_1*** project are:

Line 4: Only the `leddcd_pckg` is included in this module because no VHDL package was created for the counter schematic.

Lines 15–17: The component declaration for the 28-bit counter is directly incorporated into the architecture section of the root module. This is the simplest way to do it since the counter is only used in one place. For more complex designs, you could package the component declaration in a file that could be included anywhere the counter was needed.

© 2001 by XESS Corp.

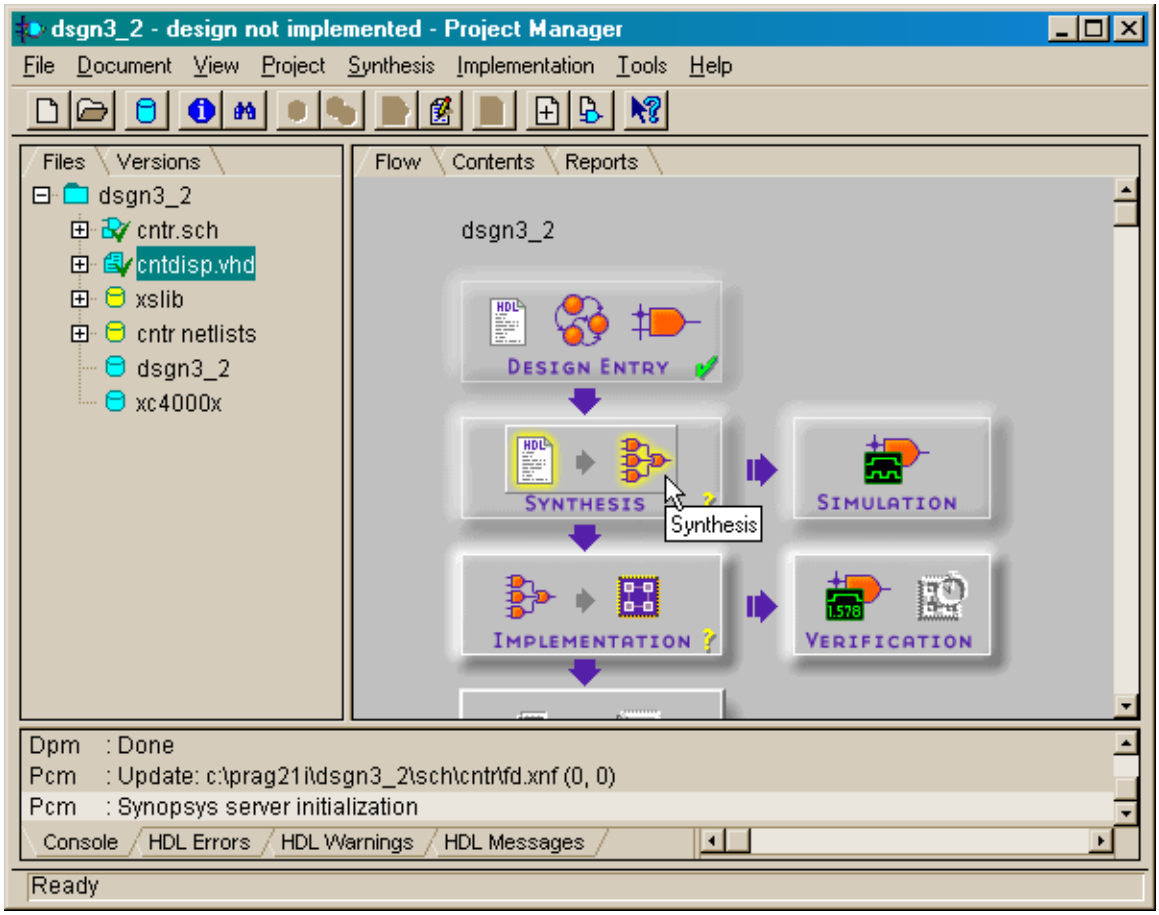


The root module is stored in the cntdisp.vhd file, and this file is added to the ***dsgn3_2*** project.

© 2001 by XESS Corp.

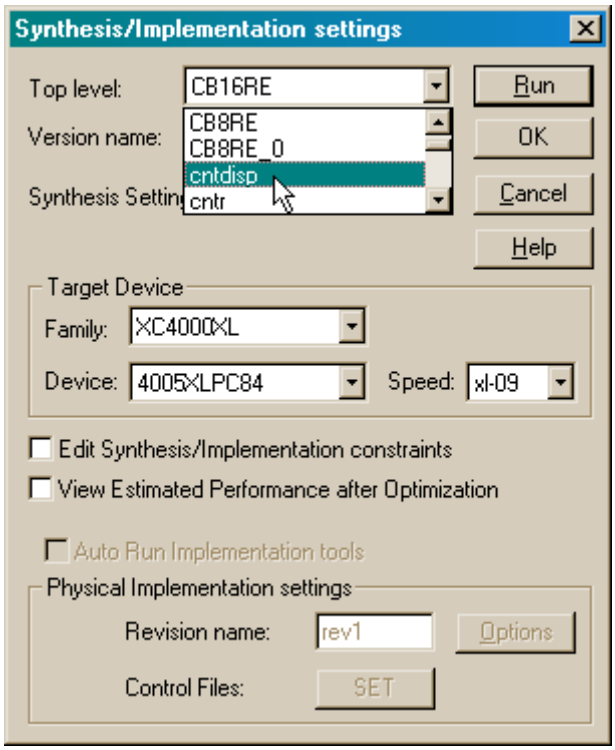
Synthesizing the Netlist

Now the synthesis tool can be run on the project files to extract the netlist.



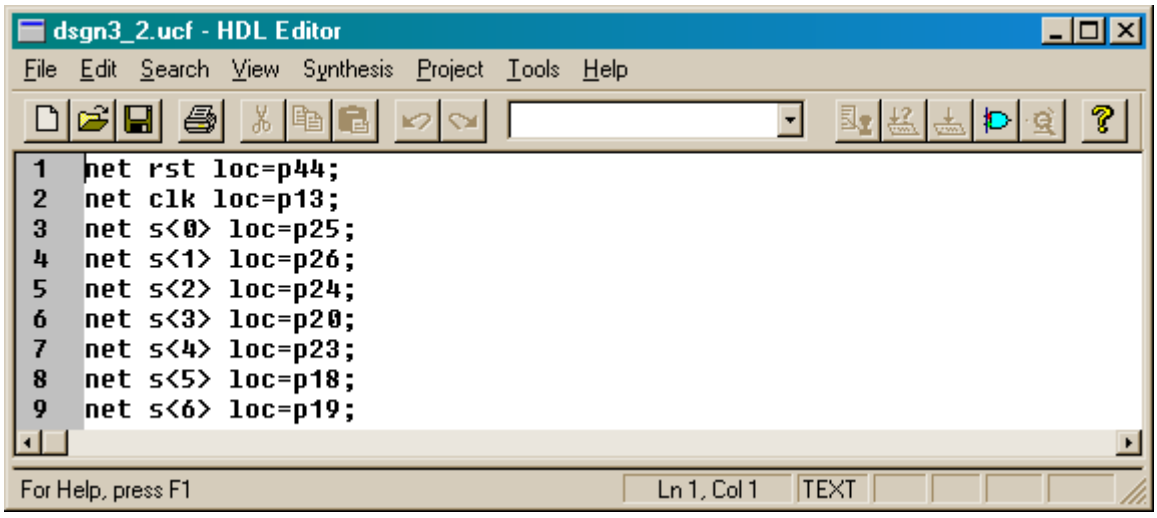
© 2001 by XESS Corp.

As before, set the target device appropriately and select the cntdisp entry as the top-level module for the synthesizer. Then click on the Run button and the synthesizer will do its job.



Assigning the I/O Pins and Implementing the Design

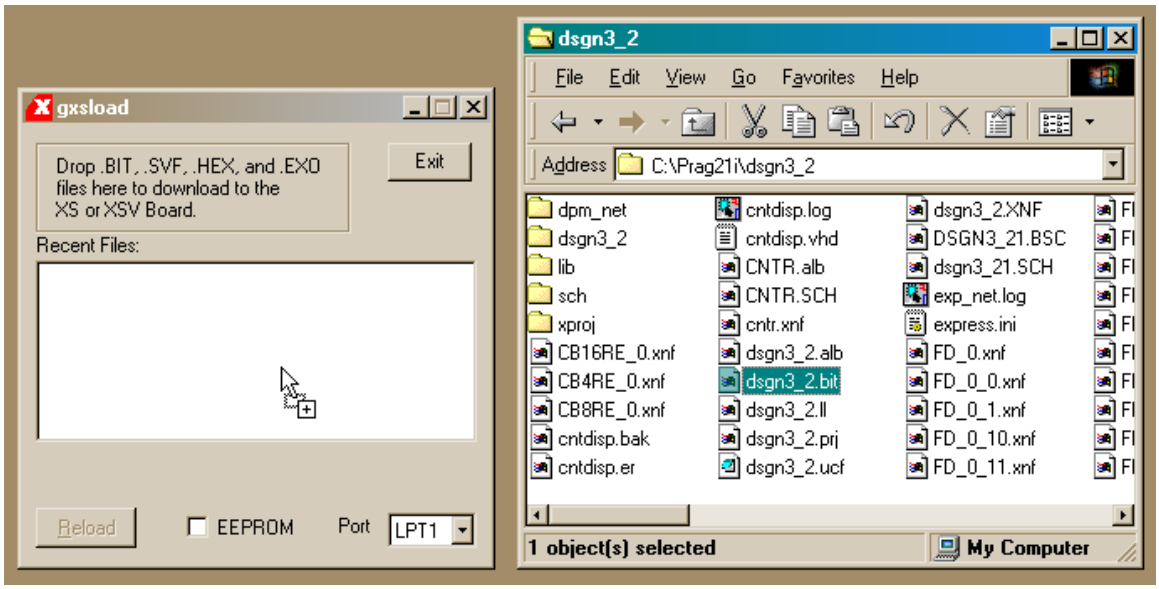
After the netlist is synthesized, place the pin assignments for the XS40 Board into the dsgn3_2.ucf file as shown below. Then specify this file as the constraints file when the implementation tools are run.



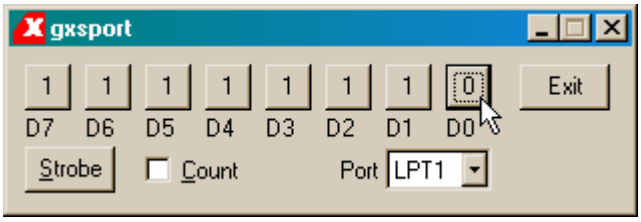
© 2001 by XESS Corp.

Downloading and Testing the Design

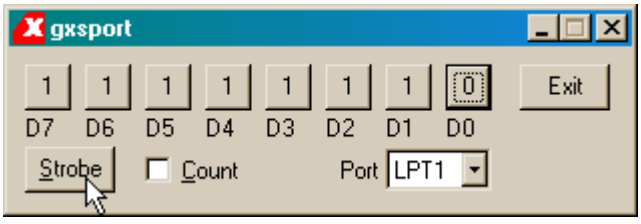
At this point, the final bitstream for downloading into the XS40-005XL Board is available. Open the directory containing the ***dsgn3_2*** project files and drag-and-drop the dsgn3_2.bit file into the **gxsload** window. The bitstream will download into the XS40 Board attached to the parallel port.



If pin D0 of the parallel port is at logic 1 after the downloading completes, the counter will be held in the reset state so only a static 0 is displayed. To release the reset, open the **gxsport** window and click on the D0 button until it displays a 0.



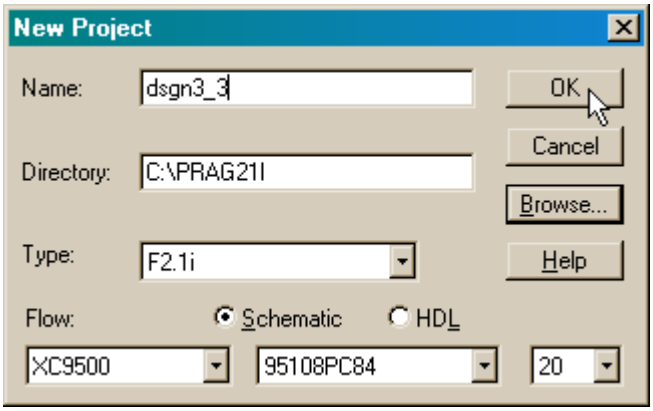
Then click on the Strobe button so the logic 0 value is output on the D0 pin of the parallel port.



Now you should observe the seven-segment LED running through the sequence: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, b, C, d, E, F, … with each digit being displayed for roughly 1/3 seconds.

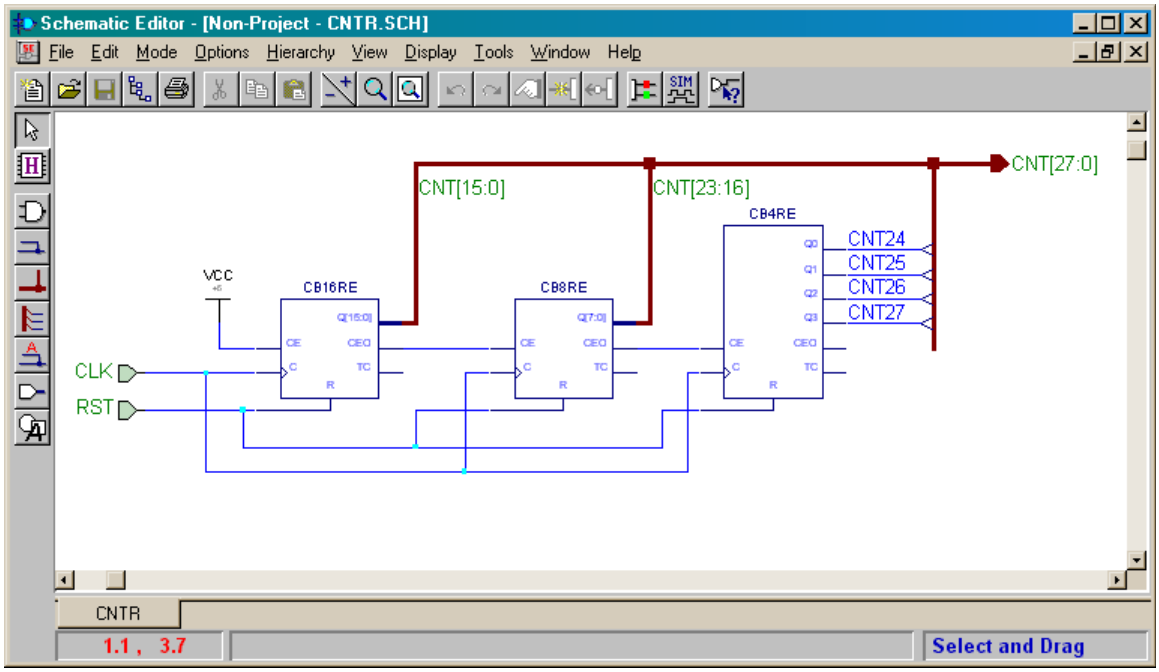
© 2001 by XESS Corp.

In the ***dsgn3_3*** project, we will replace the root VHDL module with a schematic. This design will be tested with an XS95-108 Board, so start a schematic-based project targeted at an XC95108 CPLD.



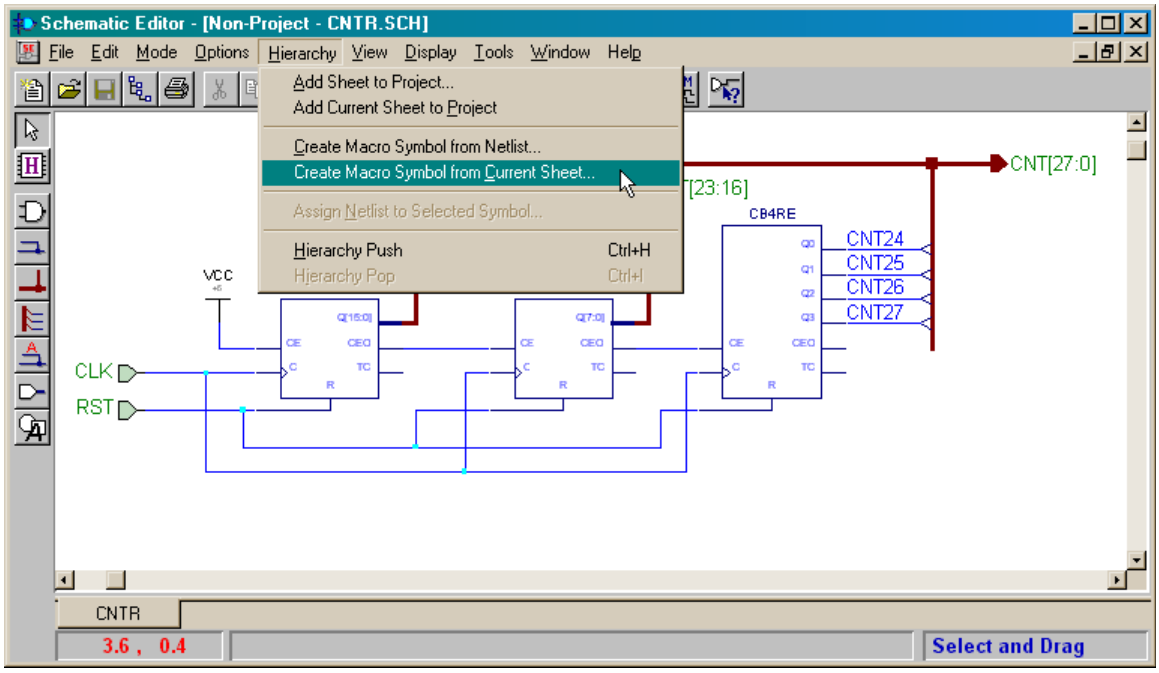
Creating the Schematic-Based Counter Macro

Open a **Schematic Editor** window and create a 28-bit counter as we did in the previous section and save it in the cntr.sch file.

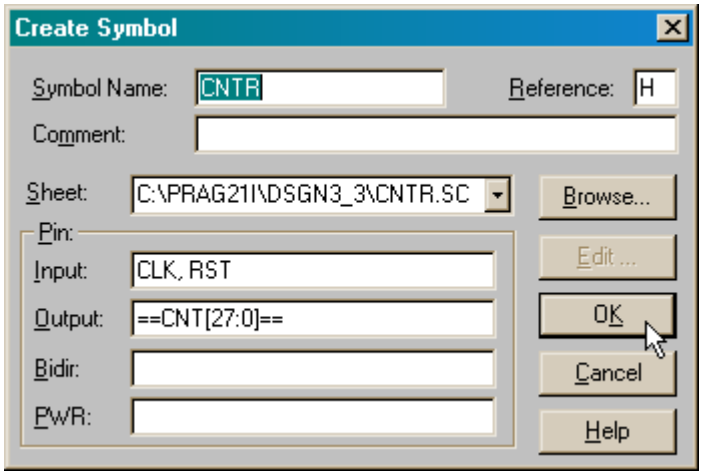


© 2001 by XESS Corp.

Since we will want to include this 28-bit counter in the top-level schematic, we need to create a macro symbol to represent the counter in the list of parts. This is done using the Hierarchy➔Create Macro Symbol from Current Sheet... command.

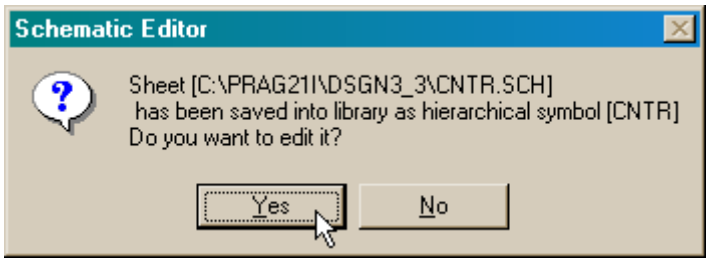


Type the name for the macro (CNTR) in the Symbol Name field of the **Create Symbol** window and click on the OK button.



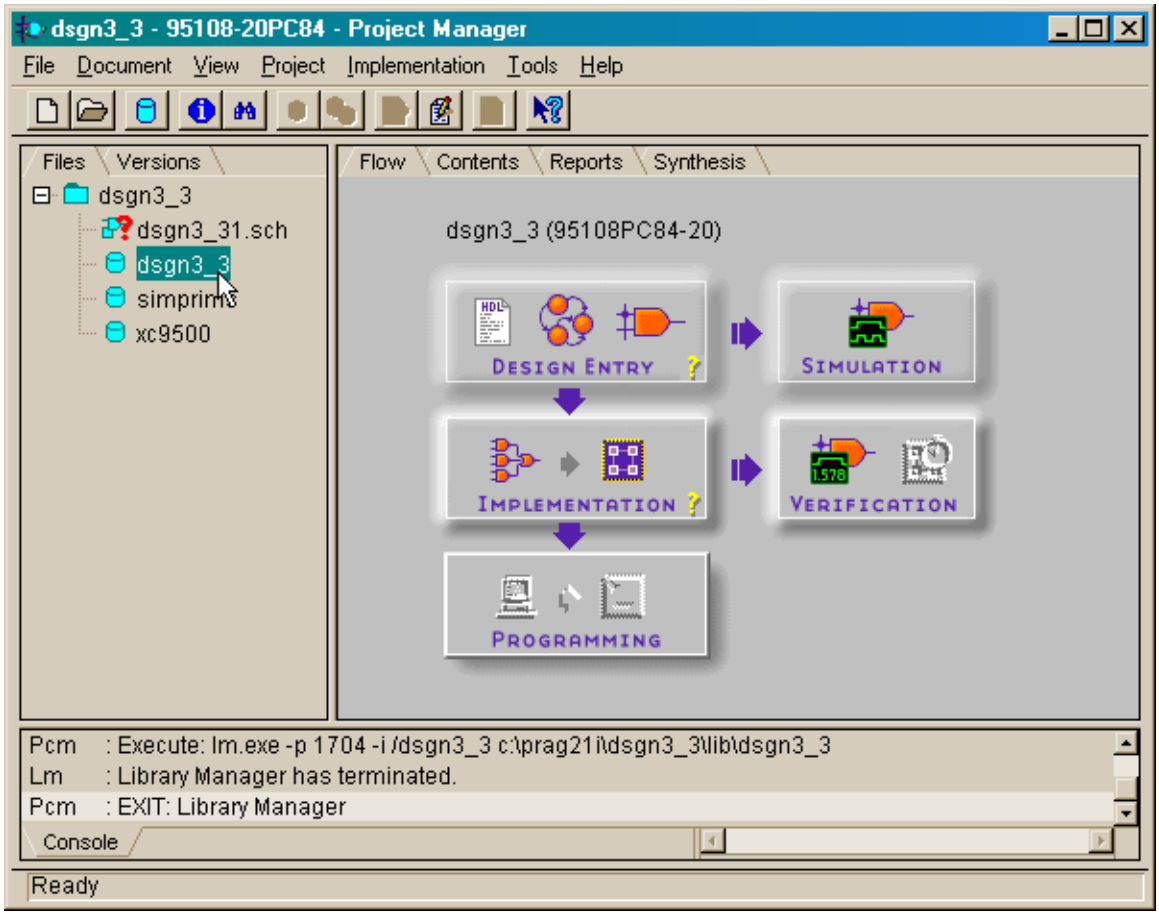
© 2001 by XESS Corp.

A symbol for the counter will be added to the ***dsgn3_3*** project library and you will be asked if you want to edit it. Click the No button and return to the **Project Manager** window.



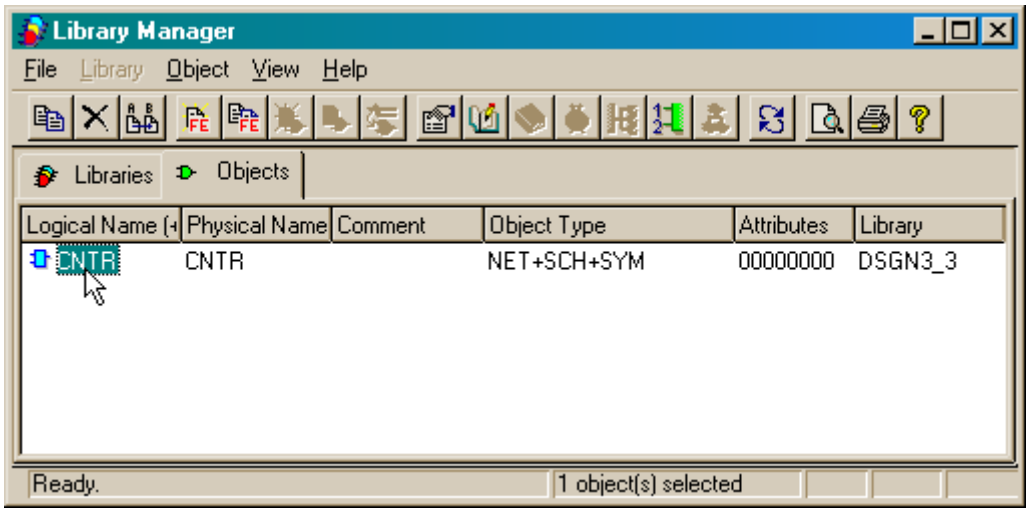
Examining the Project Library

Now we can view what has been added to the project library by double-clicking the dsgn3_3 library symbol.

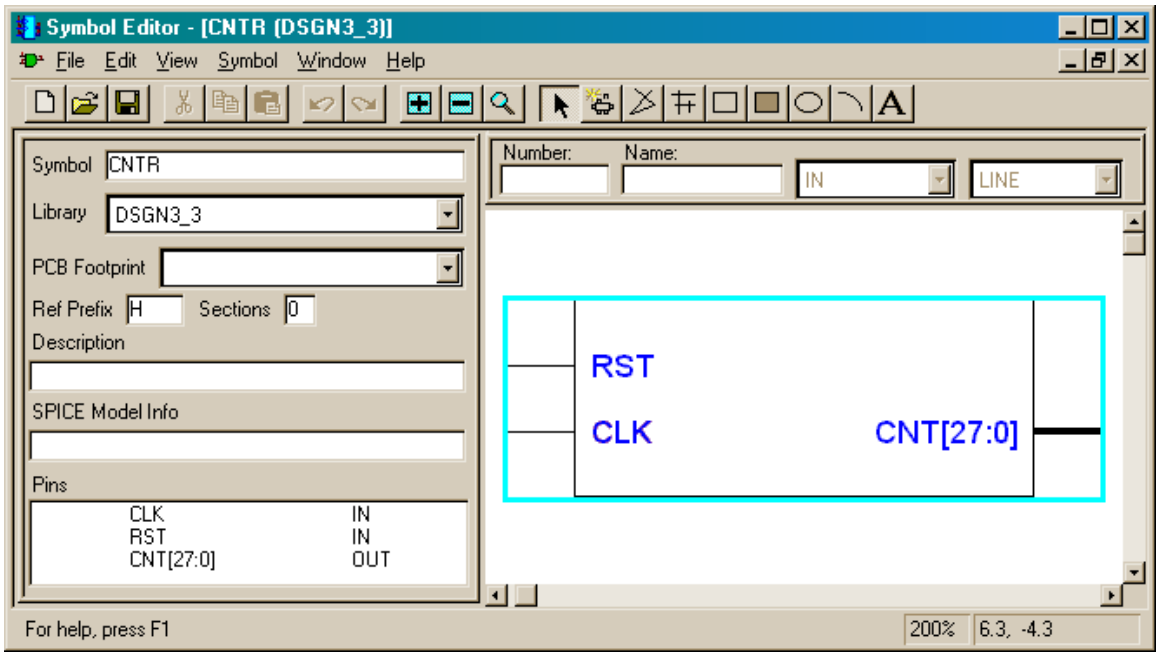


© 2001 by XESS Corp.

The **Library Manager** window will appear and show that the dsgn3_3 library contains the single CNTR object. We can see the symbol for this component by double-clicking the CNTR entry in the window.



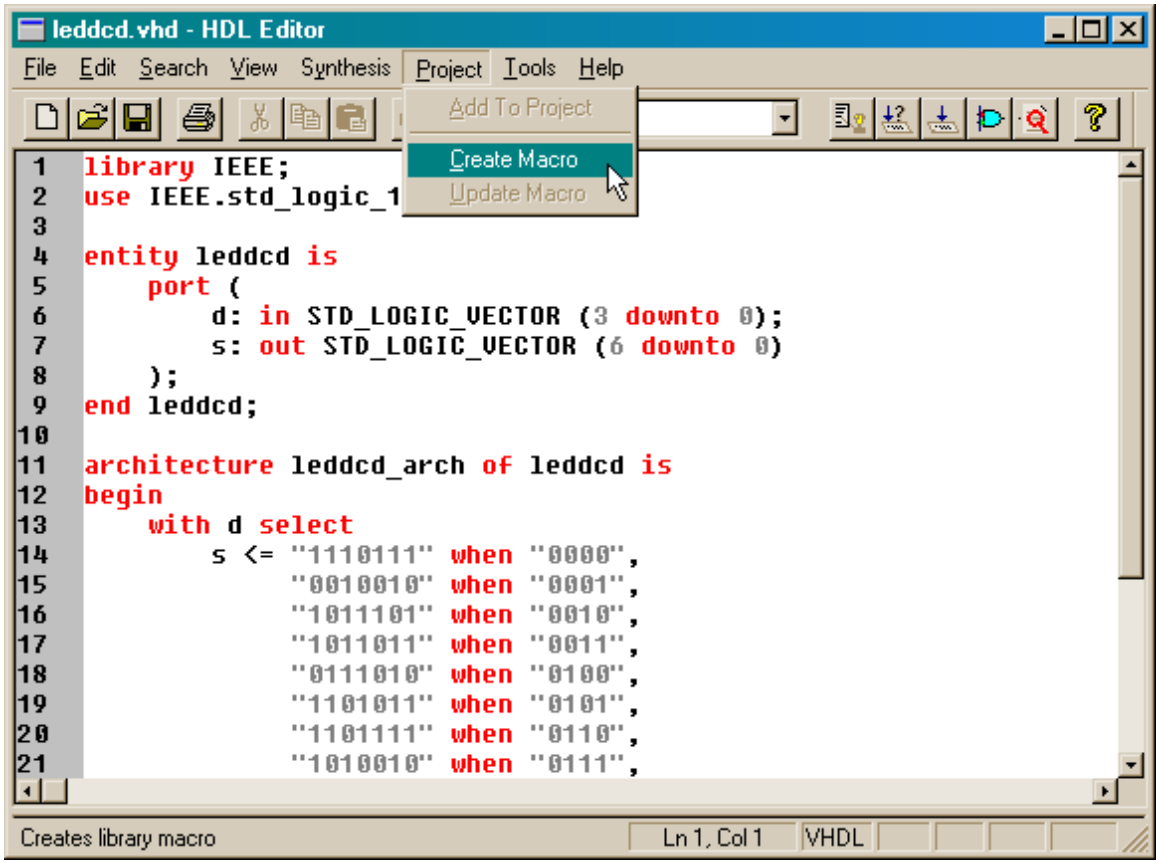
The Symbol Editor window shows the 28-bit counter with the reset and clock inputs arranged along the left-hand side and the 28-bit counter output bus on the right-hand side. This completes the process of adding the 28-bit counter macro to the project so close the **Symbol Editor** window.



© 2001 by XESS Corp.

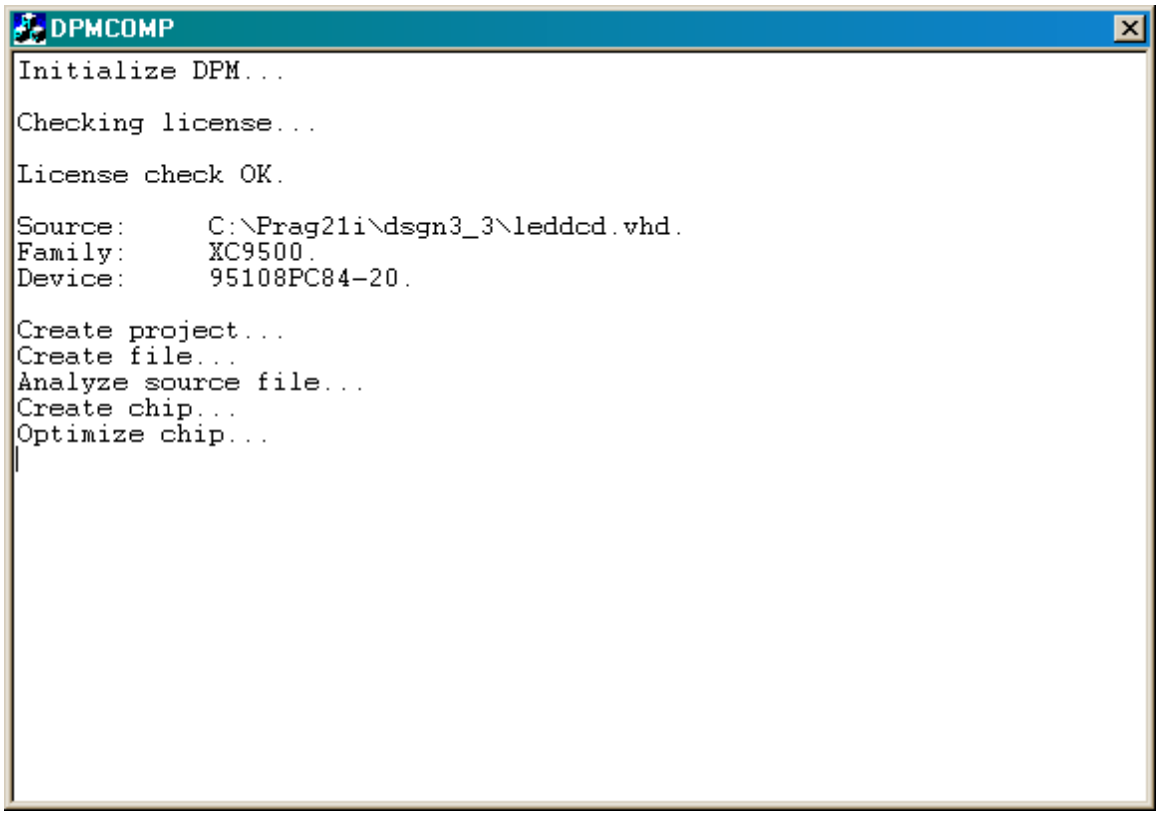
Adding a VHDL-Based Macro to the Project Library

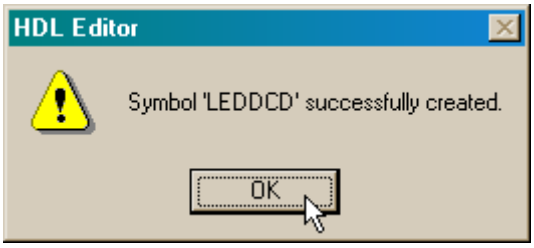
Next we need to turn the VHDL code for the seven-segment LED decoder into a macro. Enter the source code as shown below and then execute the Project➔Create Macro command.



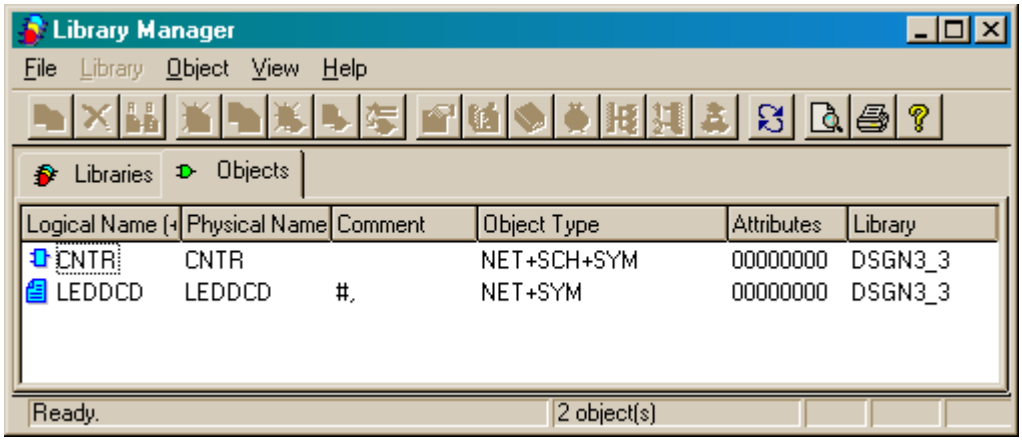
© 2001 by XESS Corp.

The **DPMCOMP** window shows the progress as the synthesizer tool processes the VHDL and deposits the netlist into the project library.



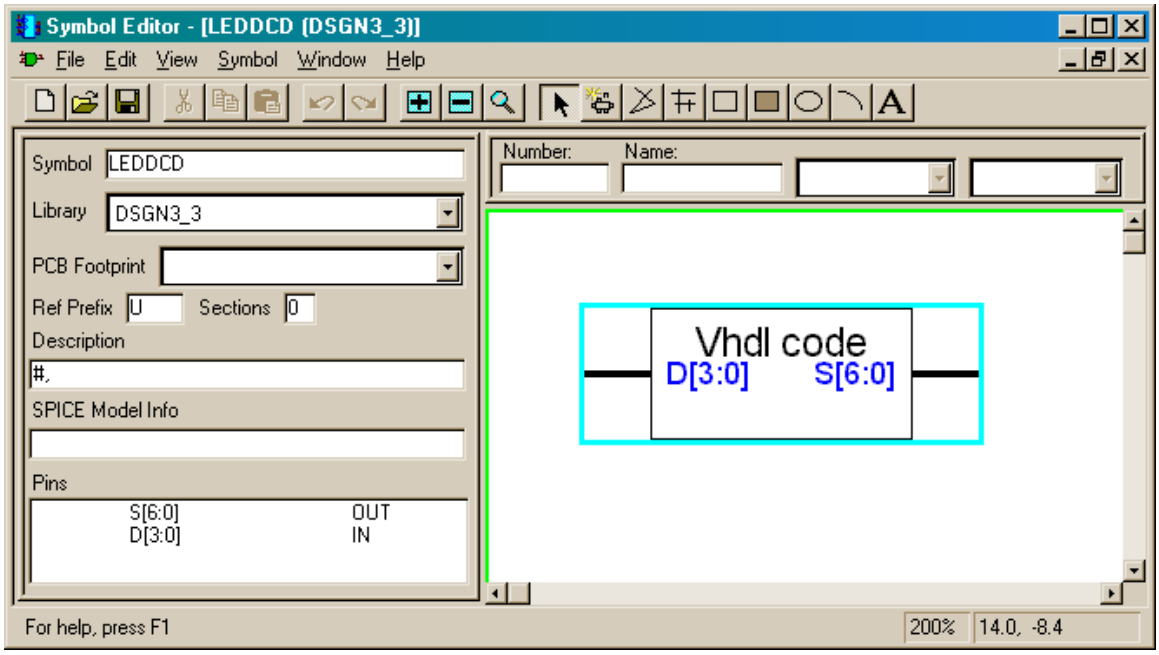


After the synthesis completes, we can double-click the dsgn3_3 library icon and see that the LED decoder has now been added to the library.



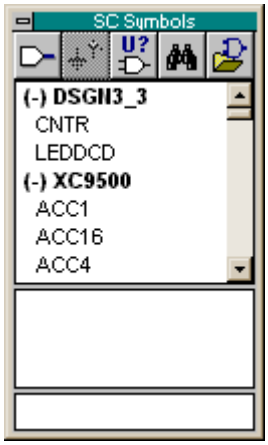
© 2001 by XESS Corp.

Double-clicking the LEDDCD entry in the list of library objects shows the symbol for the LED decoder.



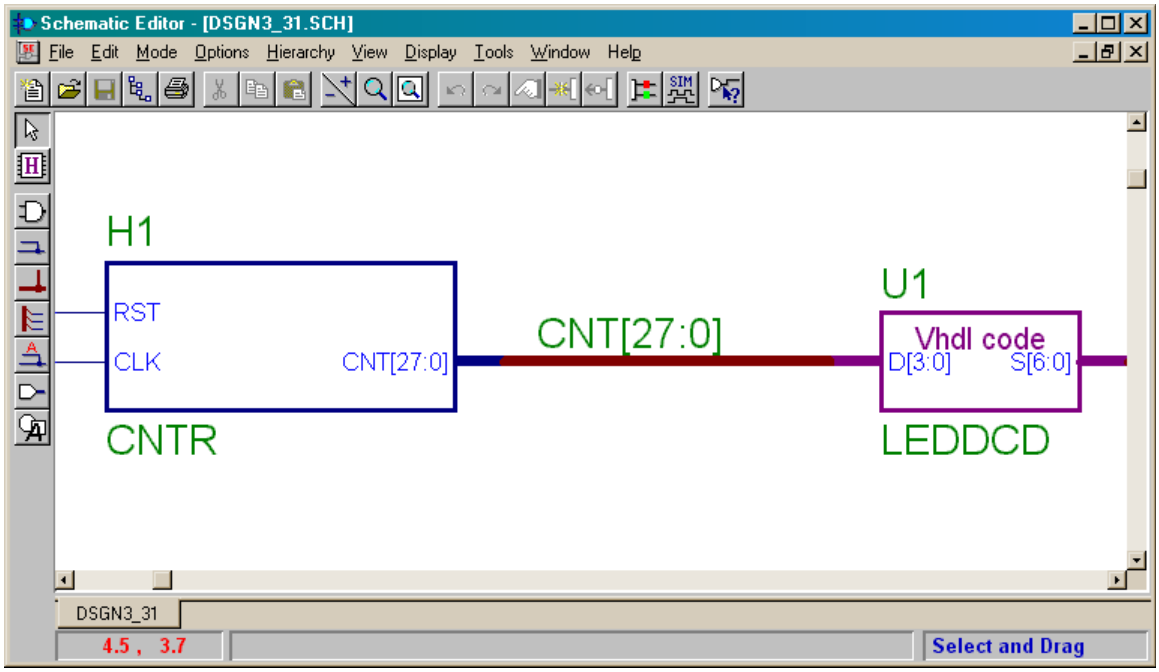
Placing the Lower-Level Macros in the Root Schematic

Now that the lower-level modules are designed, we can open a schematic for the top-level module. Notice that the list of parts available for use in creating the root module now contains the 28-bit counter and LED decoder macros.



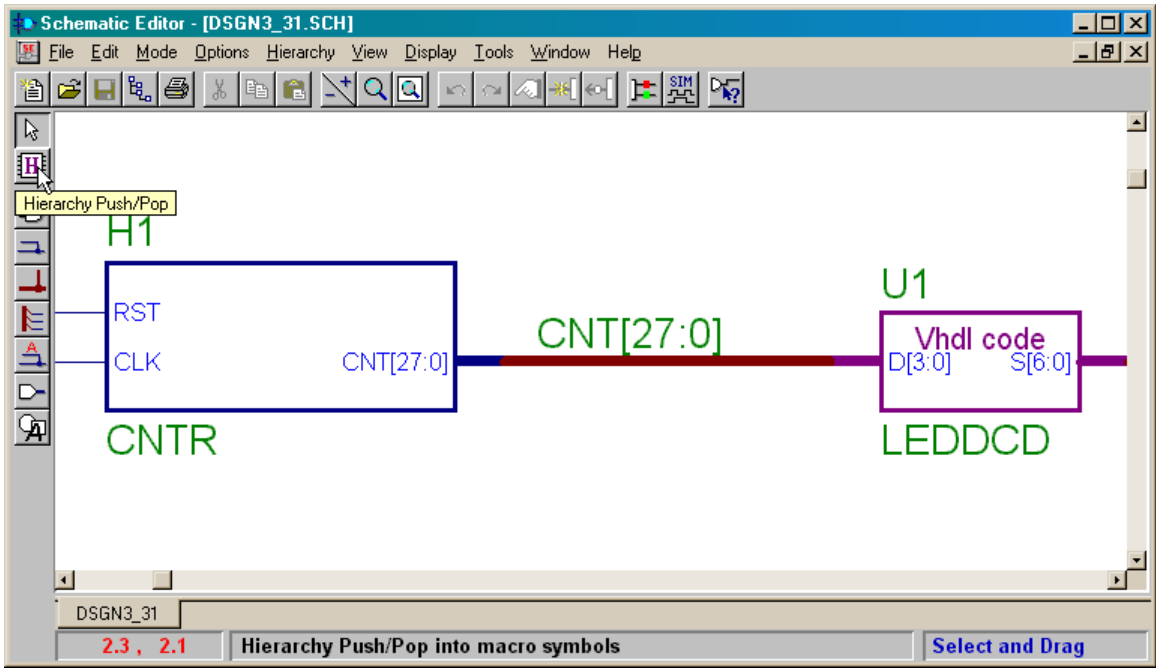
© 2001 by XESS Corp.

We can drag-and-drop the CNTR and LEDDCD macros into the drawing area of the **Schematic Editor** window and connect them with a 28-bit bus. But which of the 28 outputs from the counter macro are connected to the four inputs of the LED decoder? The rule is that the pins are connected starting at the left-most index and proceeding to the right. So CNT27 connects to D3, CNT26 connects to D2, CNT25 connects to D1, and CNT24 connects to D0.



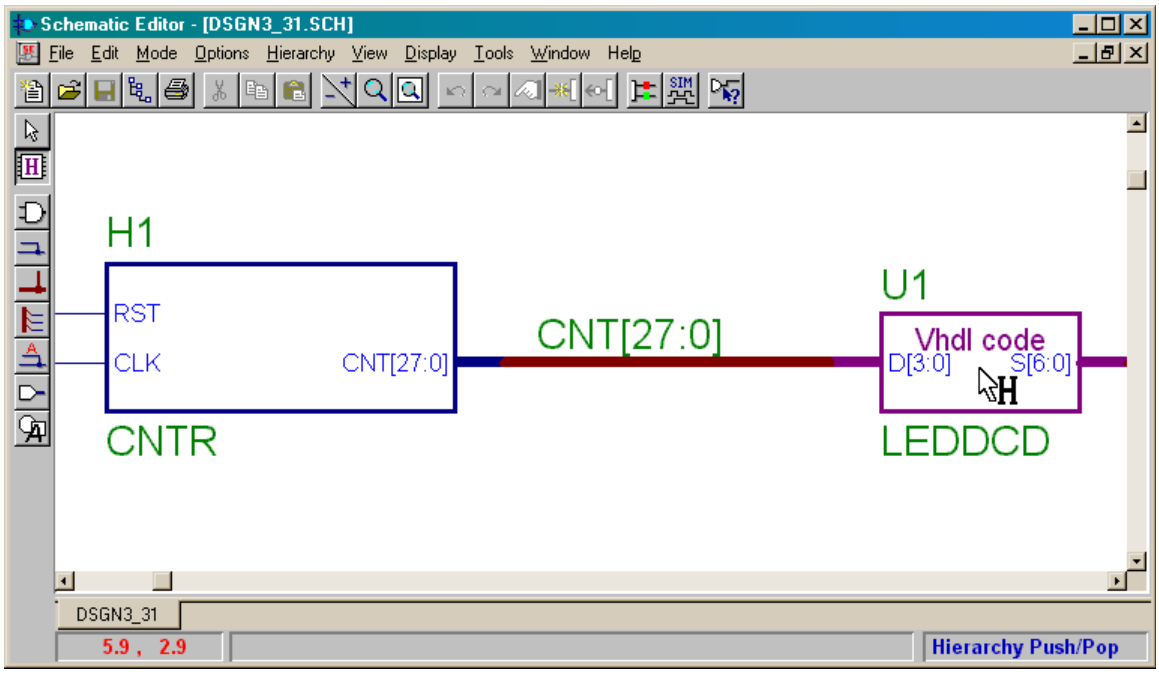
Examining Lower-Level Macros in the Hierarchy

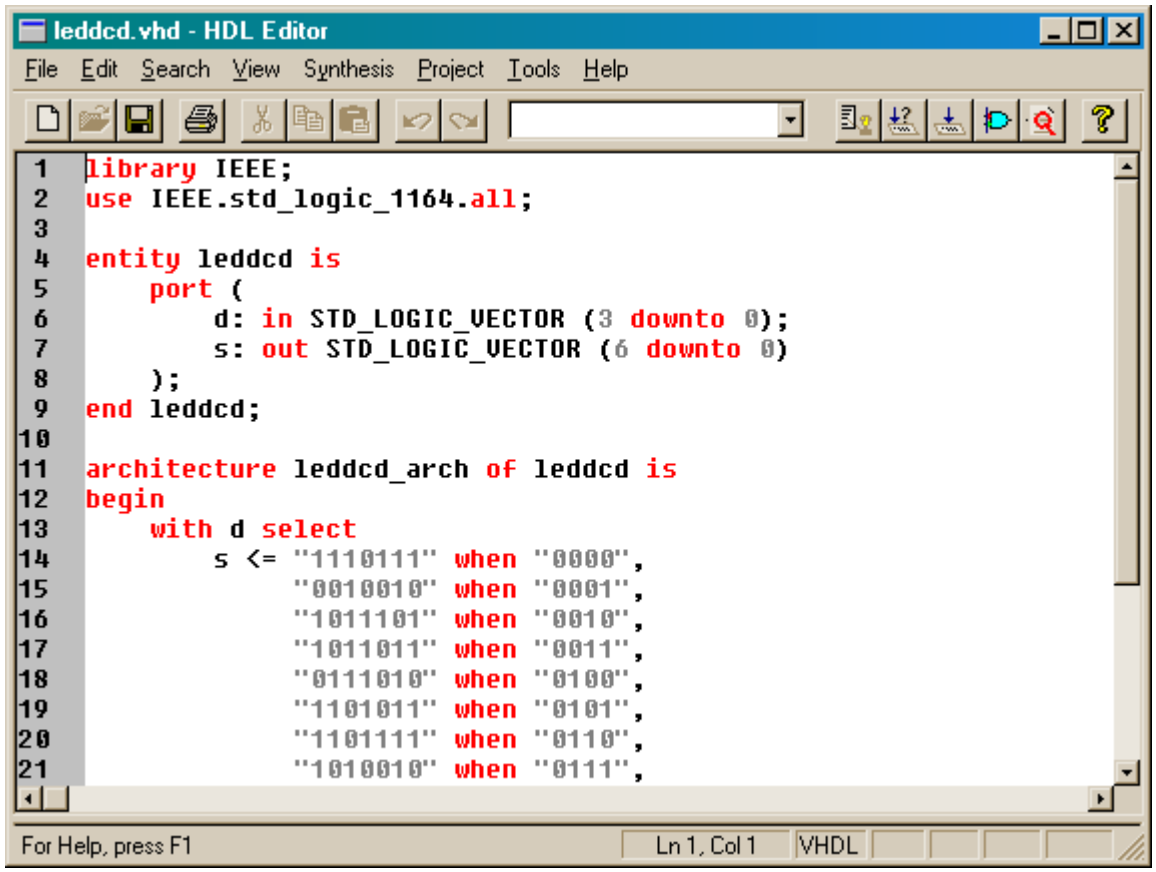
We can actually view what is inside these macros using the Hierachy Push/Pop button.



© 2001 by XESS Corp.

An **H** will be attached to the cursor indicating that it can be used to descend through the project hierarchy. Double-clicking the LEDDCD symbol loads the VHDL code describing this macro into an **HDL Editor** window. You can modify and update the macro symbol if needed.





© 2001 by XESS Corp.

Assigning the I/O Ports to the CPLD Pins and Exporting the Netlist

Since the design is being targeted to an XS95-108 Board, the inputs and outputs of the top-level module have to be connected to the CPLD pins as shown in Figure 9.

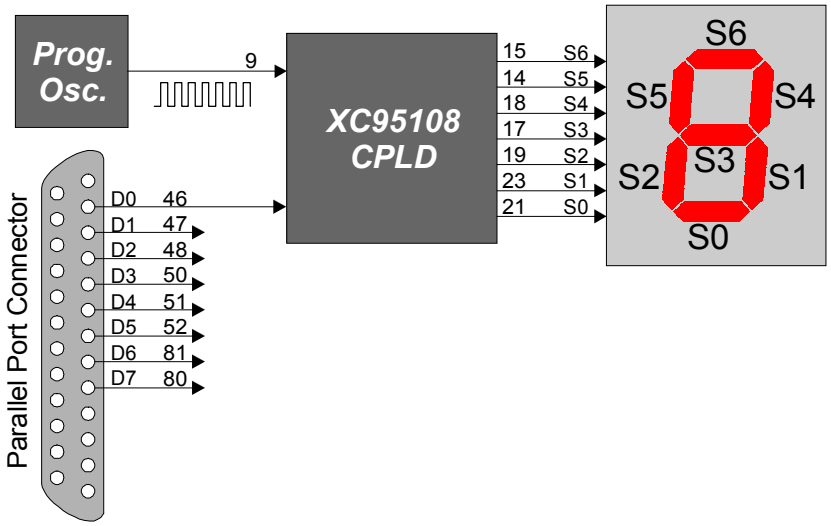
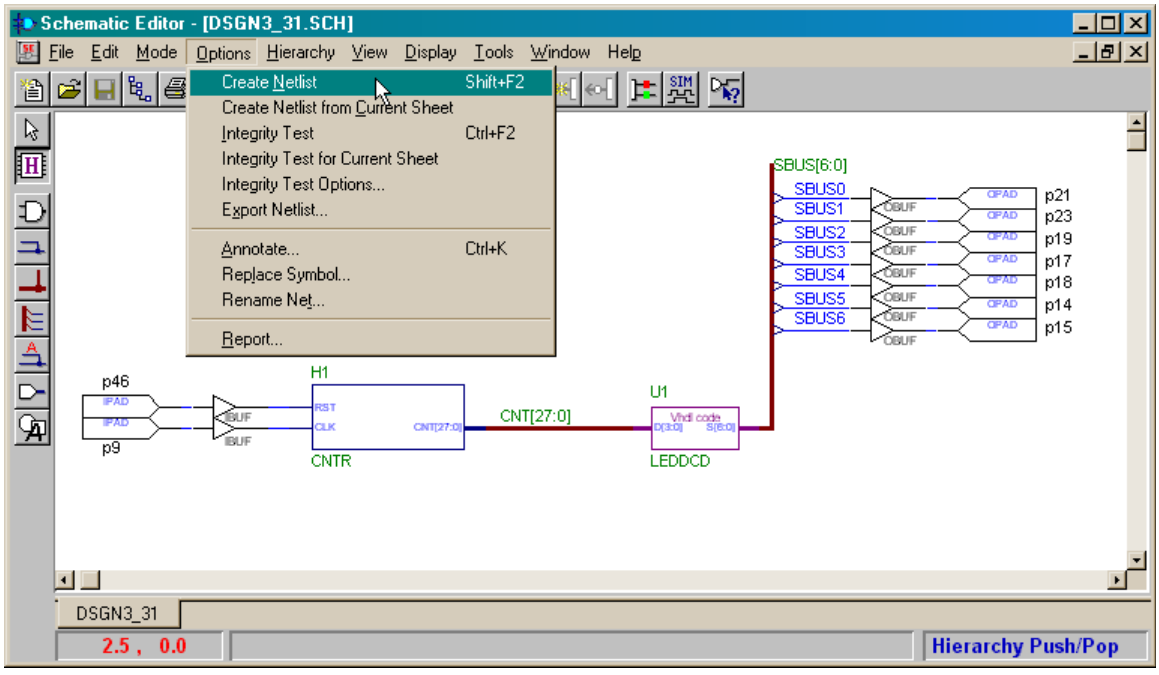


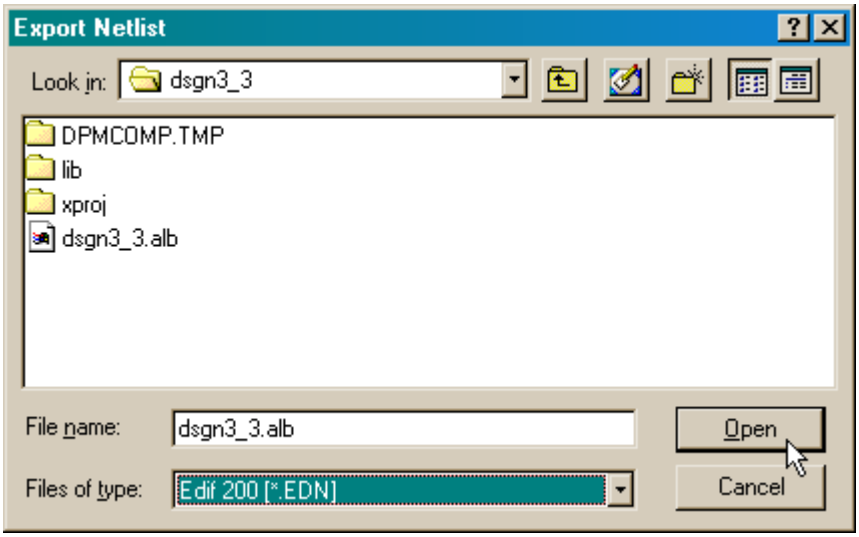
Figure 9: Connection of the programmable oscillator, parallel port, and LED digit to the pins of the CPLD on the XS95 Board.

The completed schematic with all I/O pads and their pin assignments is shown below. The netlist for the entire design is created using the Options➔Create Netlist command.



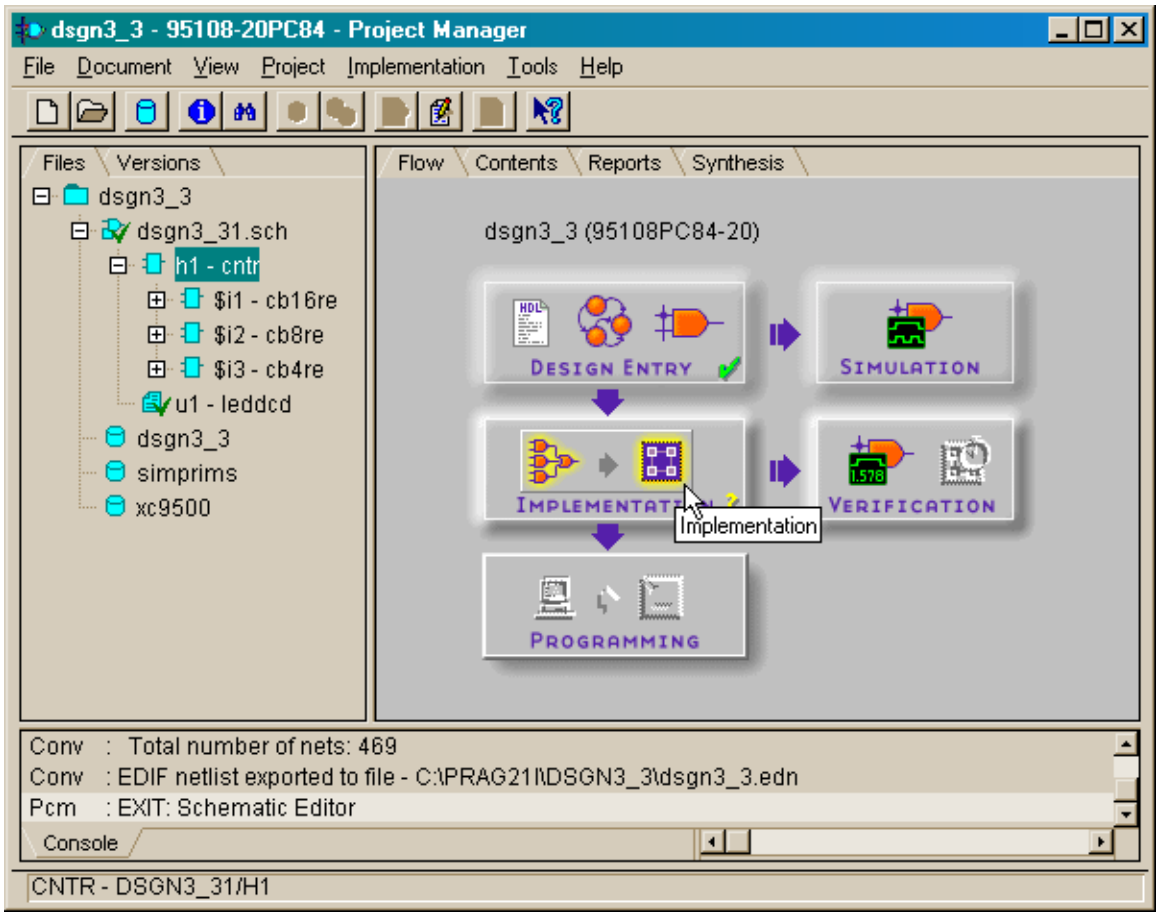
© 2001 by XESS Corp.

Then the netlist is exported into the dsgn3_3.alb file in EDIF 2.0 format using the Options➔Export Netlist... command. Then the **Schematic Editor** window can be closed.



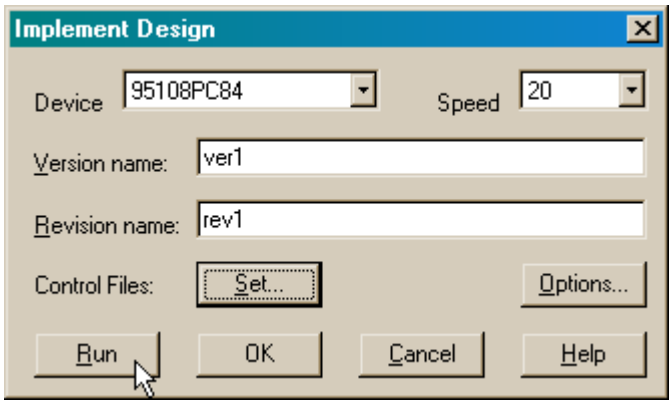
Implementing the Design

Now the implementation tools are run to map the netlist to the XC95108 CPLD chip.



© 2001 by XESS Corp.

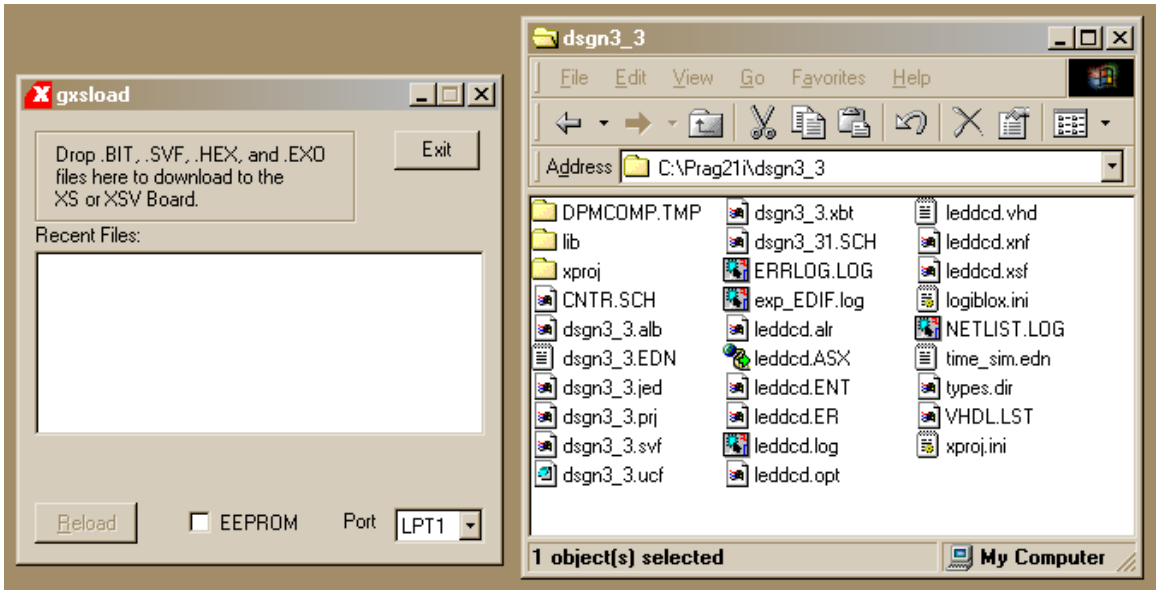
The appropriate CPLD part must be specified in the Device and Speed fields of the **Implement Design** window that apears. There is no need to specify a constraint file with the pin assignments since these have already been added to the top-level schematic.



After this is done, the programming tools are used to generate an SVF file that can be downloaded into the XS95-108 Board.

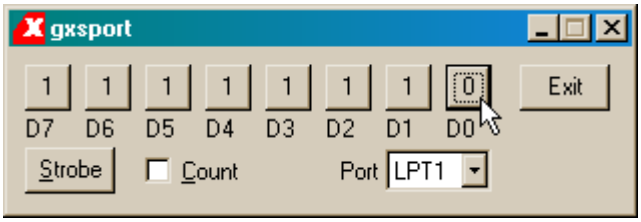
Downloading and Testing the Design

Open the directory containing the ***dsgn3_3*** project files and drag-and-drop the dsgn3_3.svf file into the **gxsload** window. The bitstream will download into the XS95 Board attached to the parallel port.

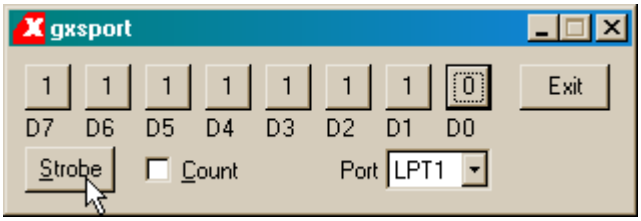


© 2001 by XESS Corp.

If pin D0 of the parallel port is at logic 1 after the downloading completes, the counter will be held in the reset state so only a static 0 is displayed. To release the reset, open the **gxsport** window and click on the D0 button until it displays a 0.



Then click on the Strobe button so the logic 0 value is output on the D0 pin of the parallel port.

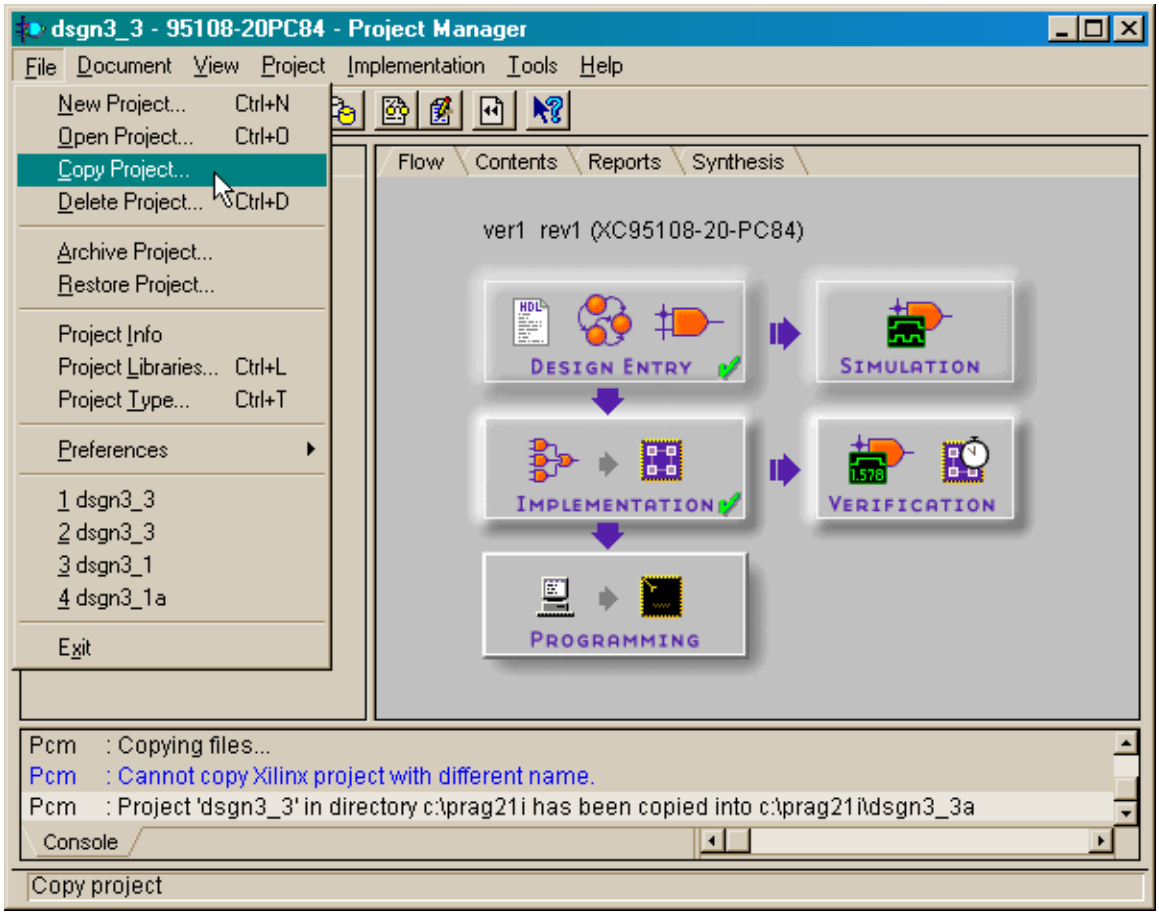


Now you should observe the seven-segment LED running through the sequence: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, b, C, d, E, F, … with each digit being displayed for roughly 1/3 seconds.

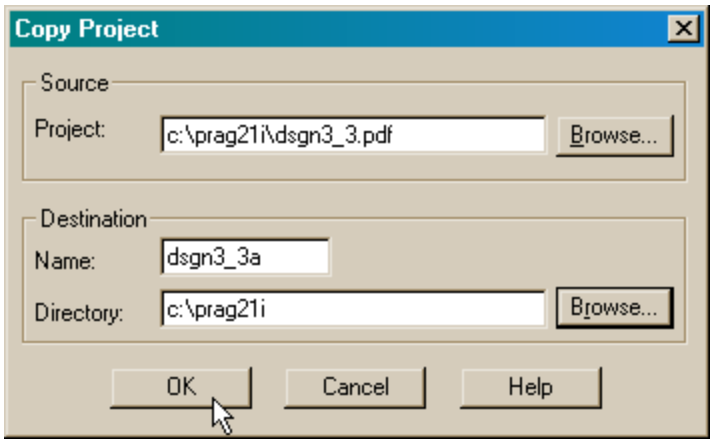
© 2001 by XESS Corp.

Creating a Macro Using LogiBLOX

Xilinx Foundation contains the LogiBLOX tool that lets you create many types of commonly-used logic functions. Rather than having to create an entire schematic to design a 28-bit counter, LogiBLOX lets you do it with a few mouse clicks. We will modify the ***dsgn3_3*** project to use a counter created with LogiBLOX. To start, make a copy of the dsgn3_3 project using the File➔Copy Project menu item.

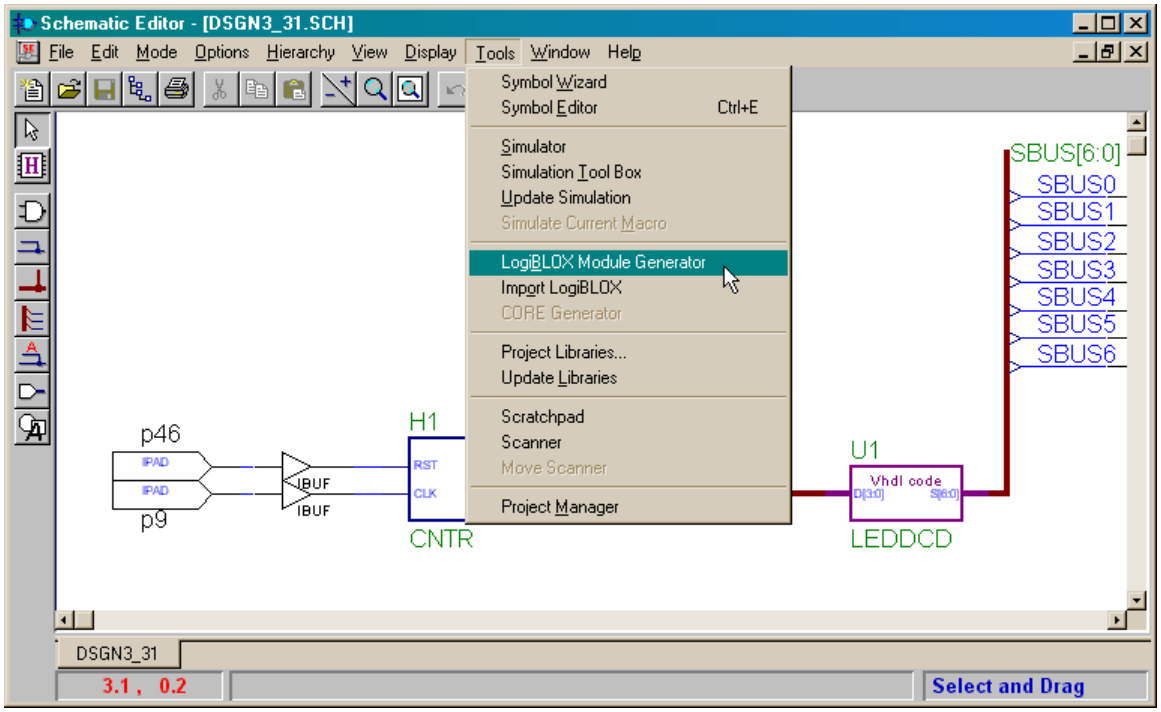


In the **Copy Project** window, name the new project dsgn3_3a.



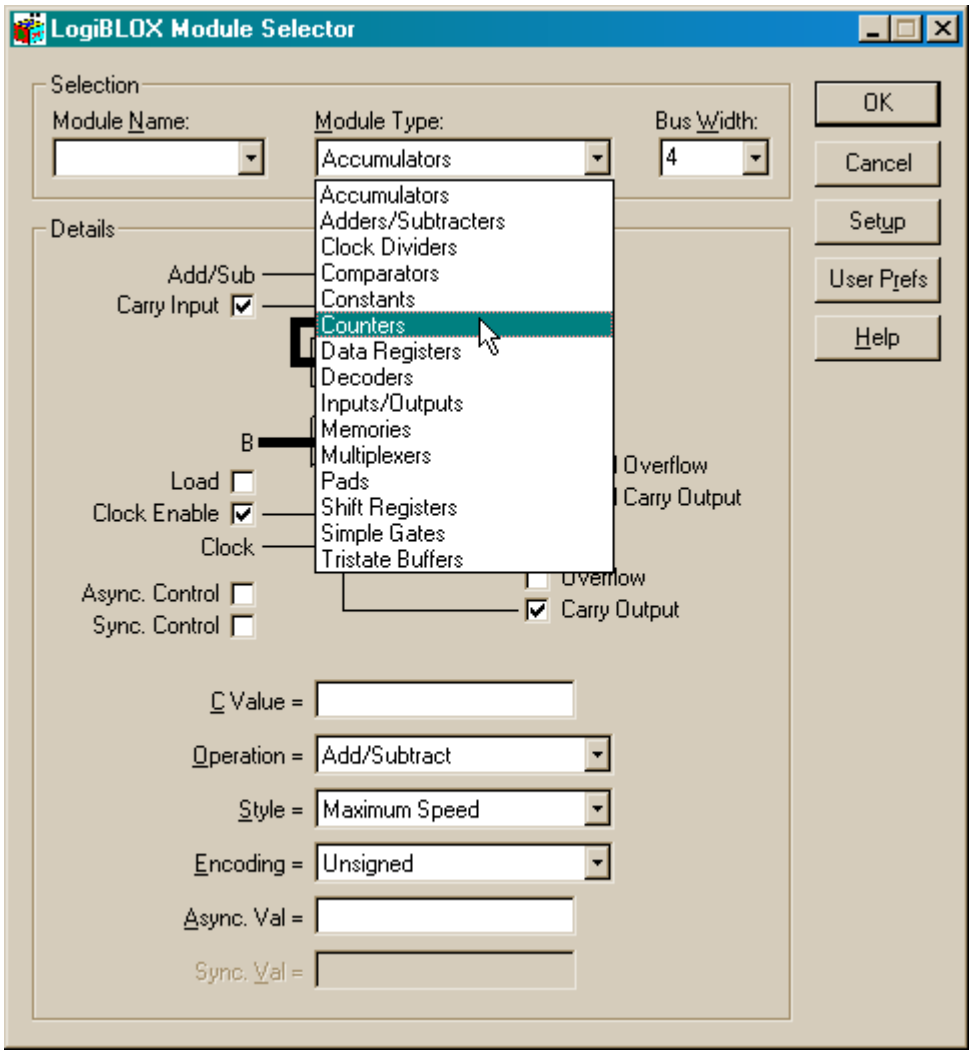
© 2001 by XESS Corp.

Open the ***dsgn3_3a*** project and then open the root module schematic (which still has the name dsgn3_31.sch). In the **Schematic Editor** window, issue the Tools➔LogiBLOX Module Generator command to begin creating a new counter macro.



© 2001 by XESS Corp.

The **LogiBLOX Module Selector** window will appear. A pull-down list attached to the Module Type field shows the range of functions that LogiBLOX can generate. Highlight the Counters entry since that is what we wish to build.



© 2001 by XESS Corp.

Now set the following fields in the window:

1. Enter CNTR2 in the Module Name field (since we already have a macro named CNTR).

2. Set the Bus Width field to 28.

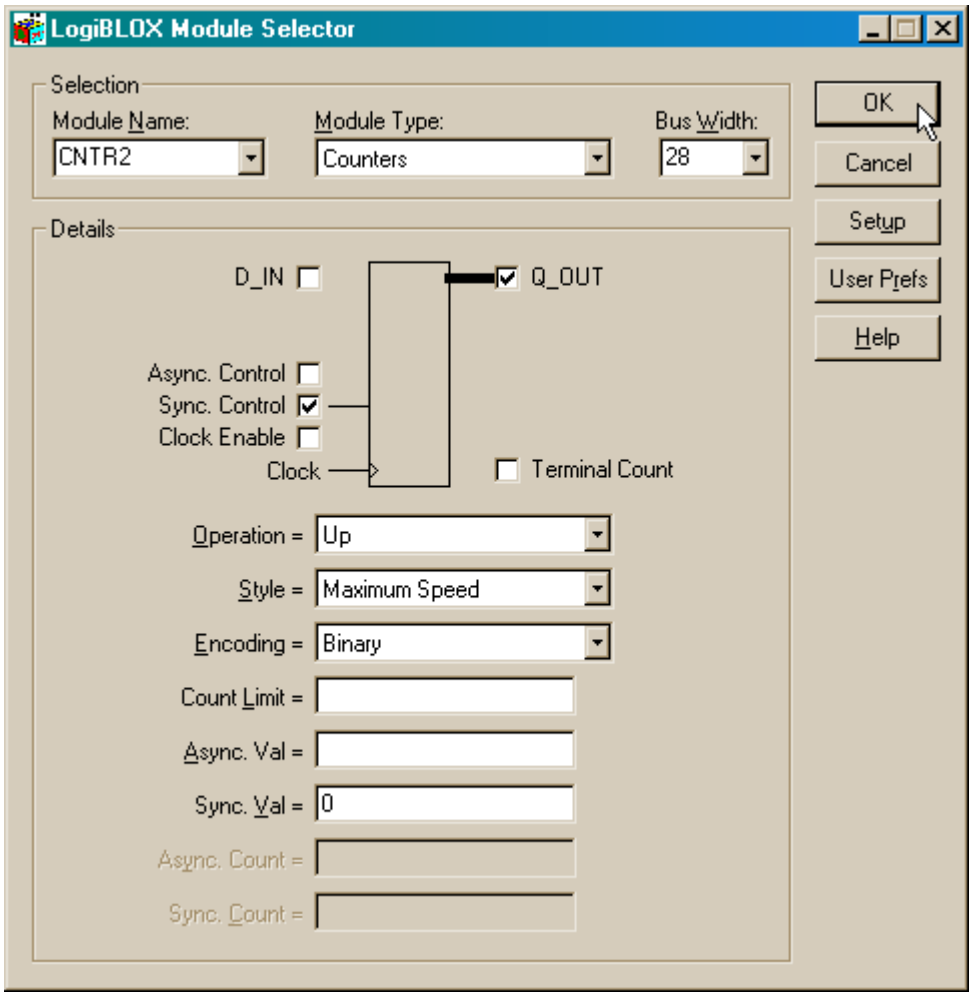
3. Remove the checkmark in the D_IN box because we do not need to load arbitrary values into the counter.

4. Remove the checkmark in the Clock Enable box because we do not need to disable the incrementing of the counter.

5. Place a checkmark in the Sync. Control box that will be used as a synchronous reset.

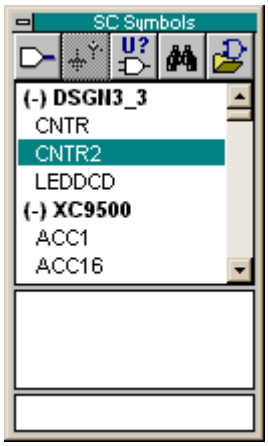
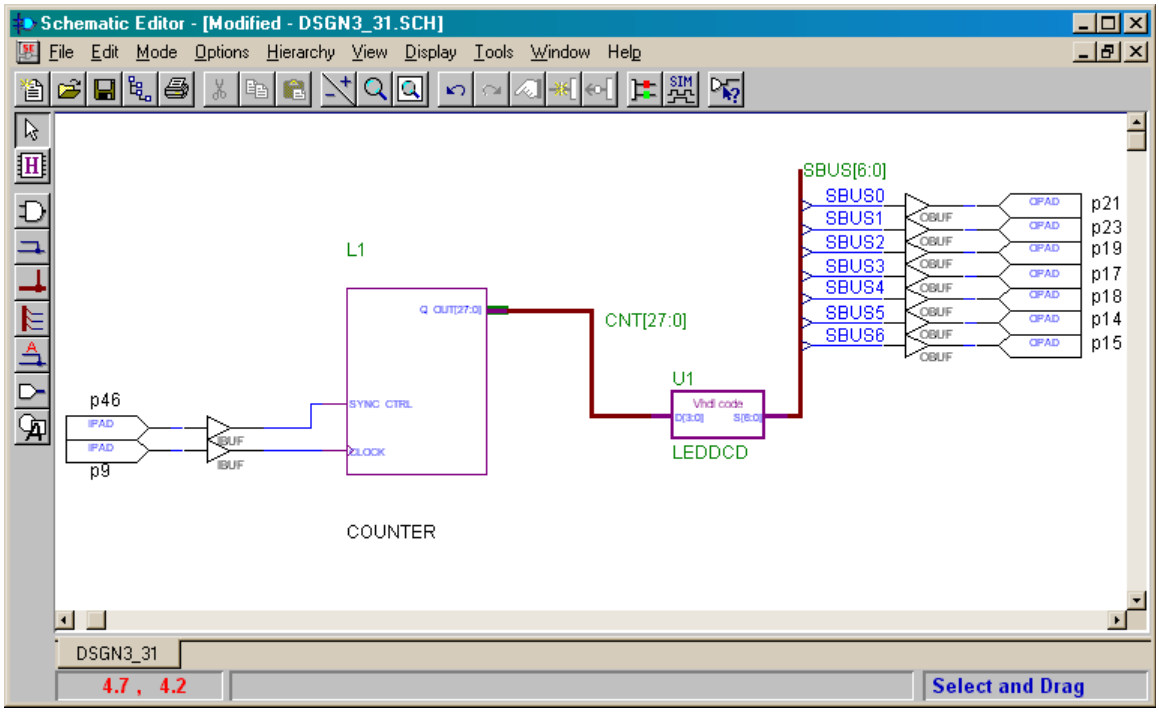
6. Enter 0 into the Sync. Val field. This value is loaded into the counter whenever a rising clock edge occurs and the Sync. Control input is a logic 1. (Thus, the Sync. Control acts as a synchronous reset input.)

7. Set the Operation field to Up since the counter only needs to be count in one direction.



© 2001 by XESS Corp.

After clicking OK in the **LogiBLOX Module Selector** window, the CNTR2 macro will appear in the part list in the **Schematic Editor** window.



The CNTR2 macro can be dropped into the schematic drawing area and attached as shown below. Then the netlist for the schematic can be extracted, exported, implemented, downloaded, and tested using the XS95-108 Board as was done with the ***dsgn3_3*** project.