

Xilinx Academy - Foundation Series training, part II

This tutorial demonstrates the HDL-based flow through the Foundation Series tools. It also shows the integrated synthesis and implementation features, new as of the Foundation Series release 1.5. This tutorial has been extracted from the F1.5 Quick-Start guide, and also includes timing simulation.

Before working on this tutorial, make sure you have properly installed the Foundation Series tools, including the FPGA Express license and any recent patches. Instructions for installation have already been provided to you, in the document called “Foundation Series training preparation.”

The HDL-Based Design Flow

This tutorial guides you through a typical HDL-based design procedure using a design of a runner’s stopwatch(called *Watch*). This design targets an XC4000E device; however, all of the principles and flows taught are applicable to any Xilinx device family.

In the first part of the tutorial, you use the Foundation Series design entry tools to complete the design. The design is composed of HDL elements and a LogiBLOX macro; you will synthesize the design using the FPGA Express compiler, integrated within the Foundation Series environment. Next, you will implement the design using the Xilinx Implementation Tools. Finally, you will verify the design through timing simulation, using the new Script Wizard.

Tutorial Project Directories and Files

During the software installation, the WTUT_VHD and WTUT_VER directories are created within c:\fndtn\Active\projects, and the tutorial files are copied into these directories. These directories contain incomplete versions of the design, done in VHDL and Verilog, respectively. You will complete the design in the tutorial. Solutions projects, that contain all completed input and output files, are also provided. The following table lists the associated project.

Directory	Description
WTUT_VHD	Incomplete Watch Tutorial - VHDL
WTUT_VER	Incomplete Watch Tutorial - Verilog
WATCHVHD	Solution for Watch - VHDL
WATCHVER	Solution for Watch - Verilog

You may copy the tutorial project to a new project if you wish to work on this tutorial more than once. The WATCHVHD and WATCHVER solution projects contain the design files for the completed tutorials, including HDL files and the bitstream file.

VHDL or Verilog?

This tutorial has been prepared for both VHDL and Verilog designs. This document applies to both designs simultaneously, noting differences where applicable. You will need to decide which HDL language you would like to work through the tutorial when you open the project.

Starting the Project Manager

1. Double click the Foundation Series Project Manager icon on your desktop or select **Programs** → **Xilinx Foundation Series** → **Xilinx Foundation Project Manager** from the Start menu.



2. A Getting Started dialog box opens. You can select a recently opened project from this box. If you have not opened this tutorial project already, click the **More Projects...** button.

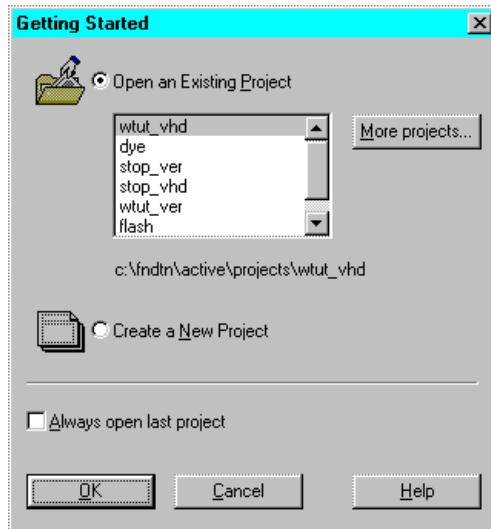


Figure 1 Getting Started Dialog Box

3. In the Directories list, browse to `c:\fndtn\Active\projects`. In the Projects list, open WTUT_VHD or WTUT_VER by double clicking.

Design Description

The design used in this tutorial is a top-level HDL design file that references several other lower-level macros. The lower-level macros are either HDL modules or LogiBLOX modules. The design begins as an unfinished design. Throughout the tutorial, you complete the design by generating some of the modules as new modules, and by completing some others from existing files.

This is the same design that you worked on in the schematic-based tutorial. The only difference is: this design is done in HDL rather than a schematic flow. Watch is a simple runner's stopwatch. There are two external inputs, and three external output buses in the completed design. The system clock is an internally generated signal produced by the OSC4, the internal oscillator in the XC4000 devices. The following list summarizes the input lines and output buses.

Inputs:

- **STRTSTOP** - Starts and stops the stopwatch. This is an active-low signal which acts like the start/stop button on a runner's stopwatch.
- **RESET** - Resets the stopwatch to 00.0 after it has been stopped.

and three additional inputs, used by the hardware debug circuit (not used in this lab):

- GSRT
- CLK_SELECT
- EXT_CLK

Outputs:

- TENSOUT[6:0]—7-bit bus which represents the Ten's digit of the stopwatch value. This bus is in 7-segment display format viewable on the 7-segment LED display on the Xilinx demonstration board.
- ONESOUT[6:0]—Similar to TENSOUT bus above, but represents the One's digit of the stopwatch value.
- TENTHSOUT[9:0]—10-bit bus which represents the Tenths' digit of the stopwatch value. This bus is one-hot encoded.

and an output, generated by the debug circuit (not used in this lab):

- CLK_OUT_15HZ

The completed design consists of the following functional blocks.

- OSC4 - Xilinx Unified Library component which represents the XC4000 on-chip oscillator.
- STATMACH - State Machine module.
- CNT60 - HDL-based module which counts from 0 to 59, decimal. This macro has 2 4-bit outputs, which represent the ones and tens digits of the decimal values, respectively.
- TENTHS - Logiblox 10-bit, one-hot encoded counter. This macro outputs the tenths digit of the watch value as a 10-bit one-hot encoded value.
- HEX2LED - HDL-based macro. This macro decodes the ones and tens digit values from hexadecimal to 7-segment display format for viewing on the FPGA Demonstration Board.
- SMALLCNTR - A simple Counter.
- DEBUG_CKT - Contains STARTUP and READBACK for use with the demonstration board. (This module is required for the Hardware tutorial and is not used in this training session.)

The Project Manager

The Project Manager controls all aspects of the design flow. You will notice in the HDL flow, the project manager appears a little different than it did for the schematic flow. A Synthesis phase button is now included in the Flow tab. Additional HDL tabs (for errors, warnings and messages) appear in the message console.

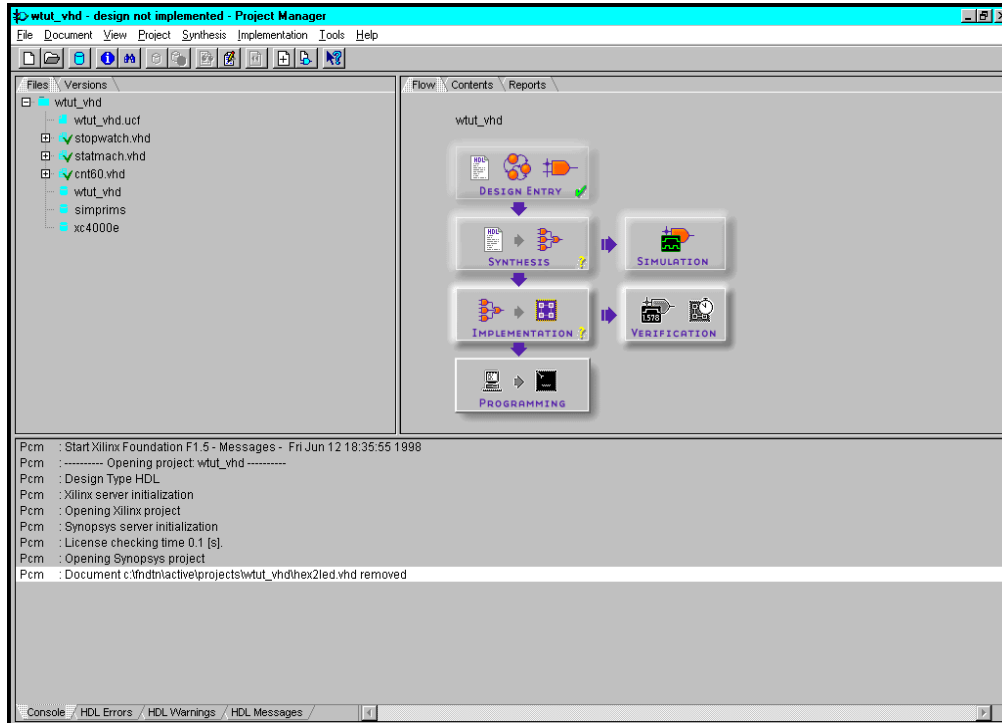


Figure 2 Project Manager

Design Entry

In this hierarchical design, you examine HDL files, correct syntax errors, create an HDL macro, and add a LogiBLOX module.

Adding Source Files

4. You must add HDL files to the project before they can be synthesized. Four HDL files have already been added to this project, but have not yet been analyzed. Use **Synthesis** → **Analyze All HDL Source Files** to update these files.
5. Add the remaining HDL file to the project. Select **Synthesis** → **Add HDL Source Files** and select SMALLCNTR.VHD or SMALLCNTR.V from the project directory.

This file will be analyzed when it is added to the project. HDL files that have been added to the project always have one of four status indicators associated with the file. Examples of these indicators are:

- A red question mark means the file has been modified and needs to be re-analyzed. Right-click the file and select Analyze.



- A red X means errors have been found. You will correct errors in an HDL file in the next section.



- A red exclamation point means warnings have been issued. Select the file and examine the warnings under the HDL Warnings tab. Many warnings can be safely ignored.



- A green check means that the file is up-to-date with no errors or warnings.



Correcting HDL errors

The SMALLCNTR design contains a syntax error that must be corrected. The red “x” next to the filename indicates an error was found during analysis. The Project Manager reports errors in red and warnings in blue in the console.

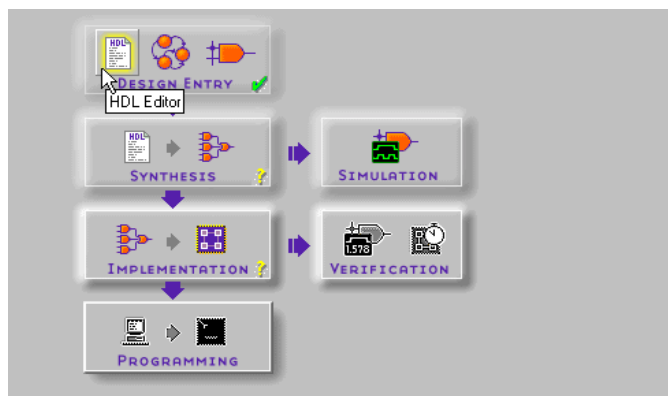
Note: To open extended help on Express errors or warnings, select the error or message in the HDL Error or Warning tab, then press the F1 key. Try this on the errors found in your HDL source during this lab.

6. Open SMALLCNTR.VHD or SMALLCNTR.V in the HDL Editor

Starting the HDL Editor

There are three different ways to open the HDL Editor tool.

- From the Flow tab, click the HDL icon in the Design Entry box. This will open the HDL editor and allow you to select the file you wish to edit.



- or, Double click an HDL file name in the Files tab.
 - or, Right-click an HDL file in the Files tab and select **Edit** from the resulting menu.
7. Correct any errors in the HDL source file. The comments next to the error explain this simple fix.
 8. Select **File** → **Save** to save the file.
 9. Re-analyze the file by selecting **Synthesis** → **Check Syntax** in the HDL Editor or by right-clicking the HDL file in the Project Manager and selecting Analyze.

Creating an HDL-Based Module

With Foundation Series, you can easily create modules from HDL code. The HDL code is connected to your top-level HDL design through instantiation and compiled with the rest of the design.

You will now create a new HDL module. This macro serves to convert the two 4-bit outputs of the CNT60 module into a 7-segment LED display format.

Using the HDL Design Wizard and HDL Editor

When you enter the name and ports of the component in the HDL Wizard, the Wizard creates a “skeleton” HDL file which you complete by inserting the remainder of your code.

10. From the Flow tab in the Project Manager, click the HDL Editor button.
11. In the HDL Editor dialog, click the radio button next to **Use HDL Design Wizard** and click **OK**.
12. Follow the instructions from the Wizard. When you are prompted for a preferred HDL language, choose whichever one you want, VHDL or Verilog. You may create files in either language, since Express allows mixed language projects. As long as the ports between your VHDL and Verilog modules match, mixed language projects are supported.
13. When you are prompted for a file name, type **HEX2LED**.
14. The HEX2LED component has a 4-bit input port named **HEX** and a 7-bit output port named **LED**. To enter these ports, first click the **New** button in the Ports dialog box. Select **Input** as the direction and type **HEX** in the Name field. Then, click the arrow next to the **Bus** field to select **3:0**, which is the width of the bus. In the **Name** field, you should now see **HEX[3:0]**, and a corresponding pin should appear on the symbol diagram on the left.

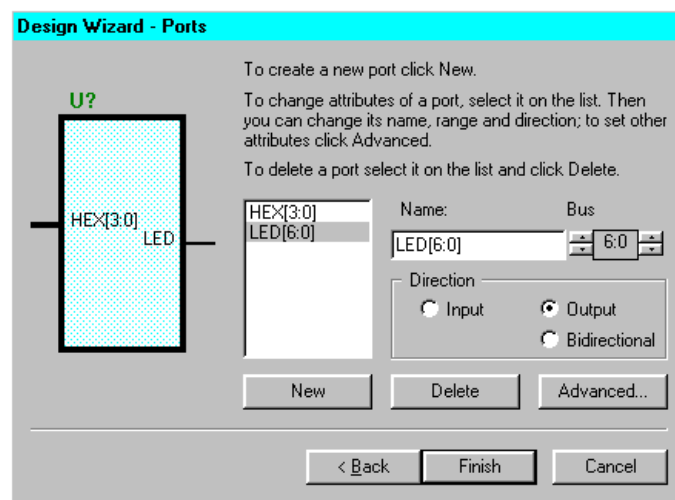
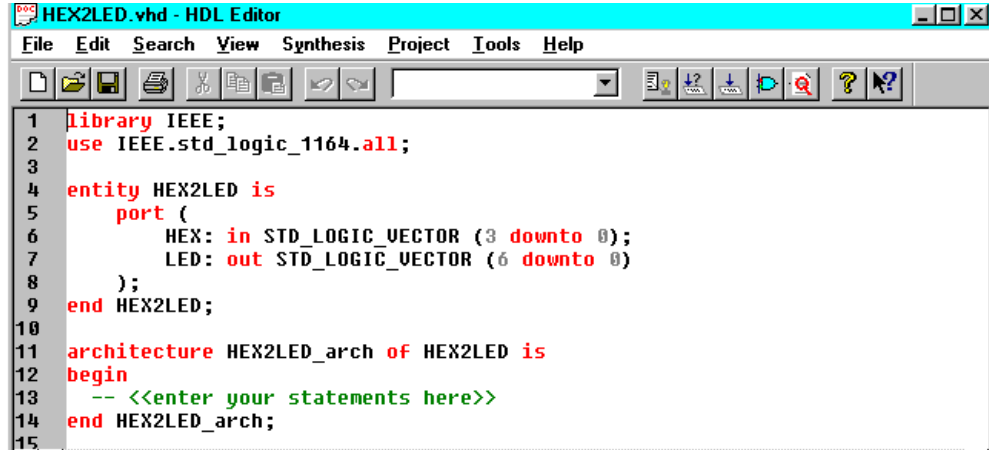


Figure 3 HDL Wizard

15. Repeat the previous step for the LED[6:0] output bus. Be sure that the direction is set to **Output**.
16. Click **Finish** to complete the Wizard session. A “skeleton” HDL file now displays in the HDL Editor.

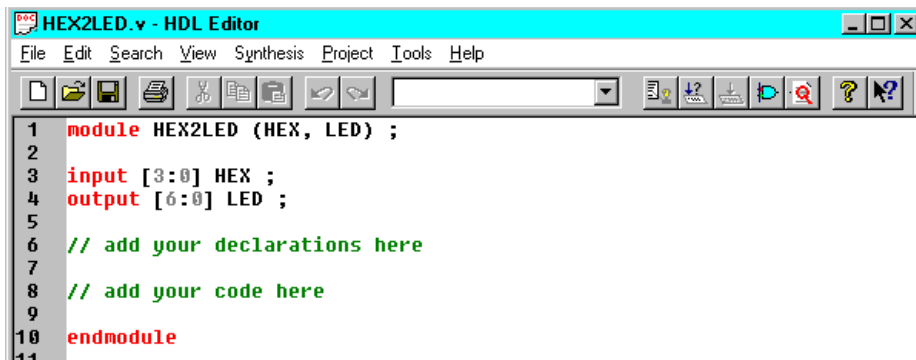


```

1 library IEEE;
2 use IEEE.std_logic_1164.all;
3
4 entity HEX2LED is
5     port (
6         HEX: in STD_LOGIC_VECTOR (3 downto 0);
7         LED: out STD_LOGIC_VECTOR (6 downto 0)
8     );
9 end HEX2LED;
10
11 architecture HEX2LED_arch of HEX2LED is
12 begin
13     -- <<enter your statements here>>
14 end HEX2LED_arch;
15

```

Figure 4 Skeleton VHDL File



```

1 module HEX2LED (HEX, LED) ;
2
3 input [3:0] HEX ;
4 output [6:0] LED ;
5
6 // add your declarations here
7
8 // add your code here
9
10 endmodule
11

```

Figure 5 Skeleton Verilog File

Using the Language Assistant

- Open the Language Assistant (select **Tools** → **Language Assistant** from the HDL Editor pull-downs) and use the template called HEX2LED Converter located under the Synthesis Templates heading. This template provides source code to convert a 4-bit value to 7-segment LED display format.

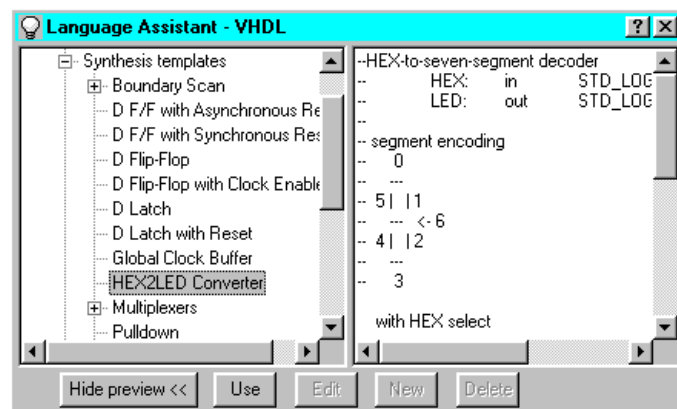


Figure 6 Language Assistant

- Before adding this template to your HDL file, be sure that the cursor in the HDL Editor is positioned below the line with the comments “<<enter your statements here>>” for VHDL. For Verilog, enter code after the “// Add your code here” line.

19. Add the HEX2LED Converter template code by clicking the **Use** button in the Language Assistant while the HEX2LED Converter template is selected.
20. Close the Language Assistant by clicking the X in the upper right corner of the window.
21. *Verilog only:* After the “//add your declarations here” statement and before the HEX2LED converter that you just added, add the following line of code to the VerilogHDL file to allow an assignment.

```
reg[6:0] LED;
```

22. You now have complete and functional HDL code. Now, check the syntax using **Synthesis** → **Check Syntax**.
23. After you successfully complete the syntax check, save the file by selecting **File** → **Save** from the HDL Editor.
24. Add this HDL file to your current project by selecting **Project** → **Add to Project**.
25. Exit the HDL Editor.

Examining the Top-Level HDL (reference)

Open STOPWATCH.VHD or STOPWATCH.V in the HDL Editor. This is the top level of the design and consists mainly of the top level ports and connections to the lower hierarchical blocks. Two Xilinx library components have been instantiated in this HDL file: OSC4 and the BUFG.

OSC4: The XC4000 devices contain an on-chip oscillator used to generate internal clock signals. To access the internal oscillator, you must instantiate the OSC4 component. In the Watch design, the 15Hz clock output of the OSC4 component is used as the system clock in the design.

BUFG: All Xilinx devices contain a set of Global Buffers that provide low-skew distribution of high fanout signals. In the Watch design, a BUFG component drives the clock signal from the OSC4. The signal on the output of the BUFG is the buffered clock signal which drives all the clocks in the system. Express infers global clock buffers, but since this clock signal is generated by the instantiated OSC4 component, the BUFG must also be instantiated.

Creating a LogiBLOX Module

In this section, you create a LogiBLOX module called Tenths. Tenths is a 10-bit one-hot encoded counter. It counts the tenths digit of the stopwatch's time value.

The LogiBLOX Module Generator GUI can be invoked from either the Project Manager, the HDL Editor, or the Schematic Editor. The operation of the tool is the same regardless of where it is invoked from. In the next steps, we will open LogiBLOX from the HDL Editor.

26. If you have closed the HDL Editor, open STOPWATCH.VHD or STOPWATCH.V.
27. From within the HDL Editor, select **Synthesis** → **LogiBLOX**.
28. The Setup window opens if this is your first call to the LogiBLOX module generator. If the Setup window does not open, click the **Setup** button. Enter the following items.
 - a) Under the Device Family tab, use the pulldown to select xc4000e.
 - b) Under the Options tab, select VHDL Template or Verilog Template. This template will be inserted into your top-level STOPWATCH file.
 - c) If you plan to simulate an HDL design, select Behavioral VHDL Netlist or Structural Verilog netlist, depending on the HDL simulator you want to use.
29. Click **OK** when you have defined all of the options.

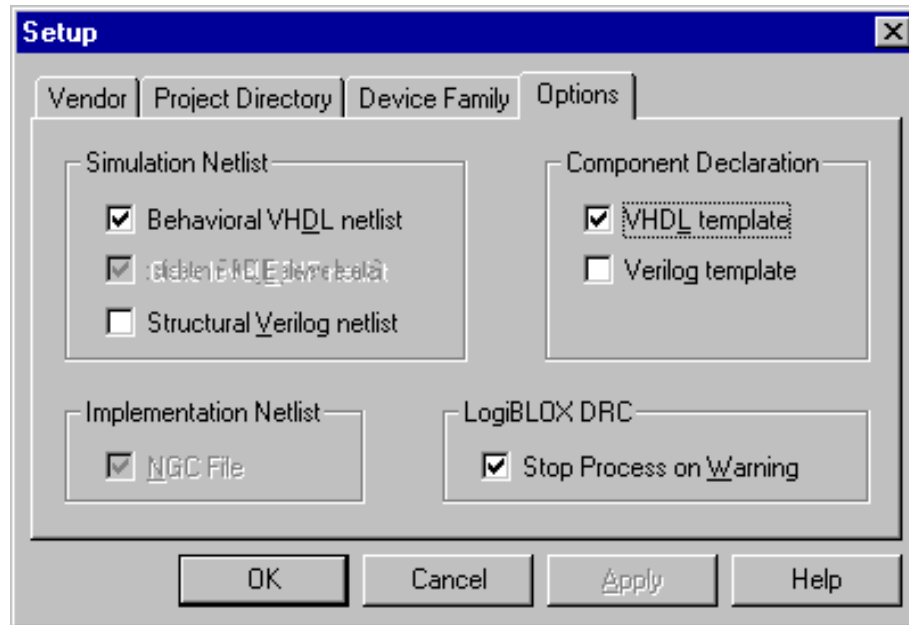


Figure 7 LogiBLOX Setup for VHDL Designs

30. Fill in the LogiBLOX Module Selector with the following settings.

- Module Type: Counters
- Module Name: Tenths
- Bus Width: 10
- Operation: Up
- Style: Maximum Speed
- Encoding: One Hot
- Async Val: 000000001

31. Check or uncheck the appropriate boxes on the module diagram so that **only** the following four pins are used.

- Async. Control
- Clock Enable
- Q_OUT
- Terminal Count

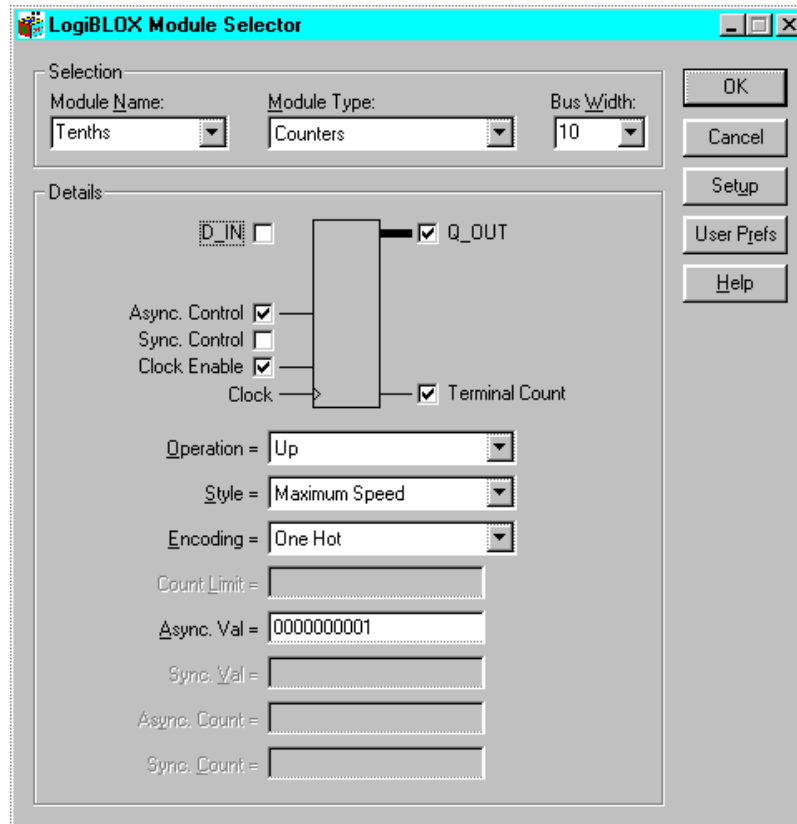


Figure 8 LogiBLOX Module Selector

32. Click **OK**. The module is created and automatically added to the project library.

A number of files are added to the project directory. These files follow:

- TENTHS.NGC - This file is the netlist that is used during the Translate phase of implementation.
- TENTHS.VHI or TENTHS.VEI - This is the instantiation template that is used to incorporate the LogiBLOX module in your source HDL.
- TENTHS.VHD or TENTHS.V - This is an HDL file to be used only for functional simulation. Do not attempt to synthesize this file. Also do not add this file to the Foundation Series project.
- TENTHS.MOD - This file stores the configuration information for the Tenths module.
- LOGIBLOX.INI - This file stores the LogiBLOX configuration for the project.

Instantiating the LogiBLOX Module in the HDL Code

You now have the LogiBLOX counter available to your project. Your HDL code must call it, therefore you must insert some declarations and instantiations into your code. You will insert the LogiBLOX-generated code fragments: TENTHS.VHI into your VHDL code or TENTHS.VEI into your Verilog.

VHDL Flow (see Verilog flow instructions in the next section)

33. If you have closed the HDL Editor, open STOPWATCH.VHD.

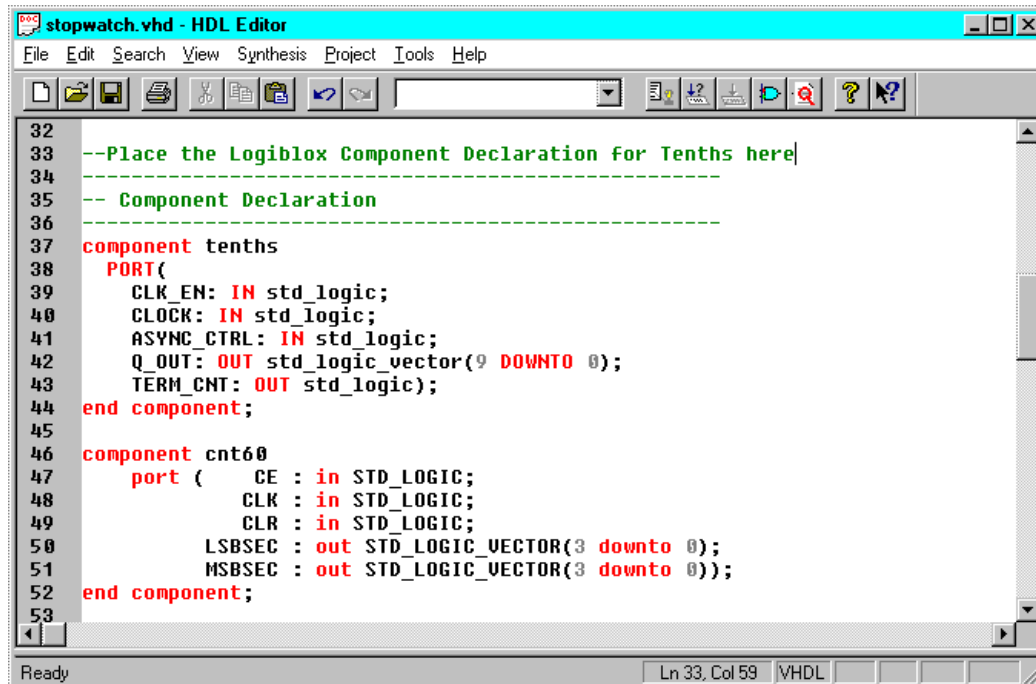
34. Place your cursor after the line that states:

“-- Place the LogiBLOX Component Declaration for Tenths here”

Select **Edit** → **Insert File** and choose Tenths.vhi. The VHDL template file for the LogiBLOX instantiation is inserted.

The Component Declaration does not need to be modified. Now you will move the instantiation portion of the code fragment.

35. Highlight the inserted code from “--Component Instantiation” to “TERM_CNT=>)”. Select **Edit** → **Cut**.



```

32
33 --Place the Logiblox Component Declaration for Tenths here|
34 -----
35 -- Component Declaration
36 -----
37 component tenths
38   port (
39     CLK_EN: IN std_logic;
40     CLOCK: IN std_logic;
41     ASYNC_CTRL: IN std_logic;
42     Q_OUT: OUT std_logic_vector(9 DOWNTO 0);
43     TERM_CNT: OUT std_logic;
44   end component;
45
46 component cnt60
47   port (
48     CE : in STD_LOGIC;
49     CLK : in STD_LOGIC;
50     CLR : in STD_LOGIC;
51     LSBSEC : out STD_LOGIC_VECTOR(3 downto 0);
52     MSBSEC : out STD_LOGIC_VECTOR(3 downto 0));
53   end component;

```

Figure 9 VHDL Component Declaration of LogiBLOX Module

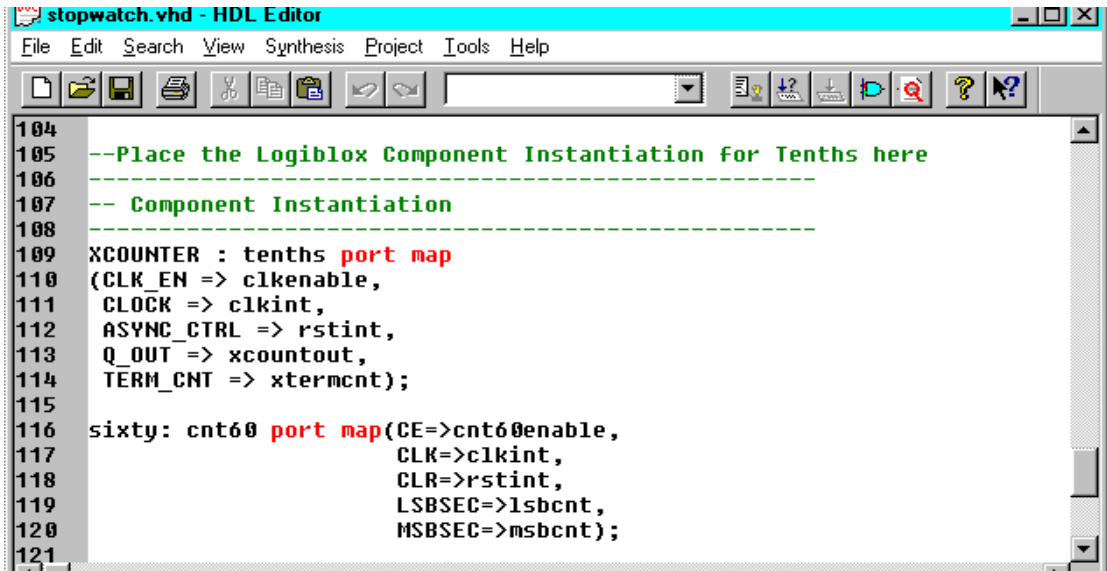
36. Place the cursor after the line that states:

“--Place the LogiBLOX Component Instantiation for Tenths here.”

Select **Edit** → **Paste** to place the instantiation here.

Change “instance_name” to “XCOUNTER”

37. Edit this instantiated code to connect the signals in the Stopwatch design to the ports of the LogiBLOX module. The completed code looks like the following.



```

104
105 --Place the Logiblox Component Instantiation for Tenths here
106 -----
107 -- Component Instantiation
108 -----
109 XCOUNTER : tenths port map
110 (CLK_EN => clkenable,
111  CLOCK => clkint,
112  ASYNC_CTRL => rstint,
113  Q_OUT => xcountout,
114  TERM_CNT => xtermcnt);
115
116 sixty: cnt60 port map(CE=>cnt60enable,
117                      CLK=>clkint,
118                      CLR=>rstint,
119                      LSBSEC=>lsbcnt,
120                      MSBSEC=>msbcnt);
121

```

Figure 10 VHDL Component Instantiation of LogiBLOX Module

38. Save the design and close the HDL Editor. Now, skip the Verilog Flow section and go to “Simulating the HDL.”

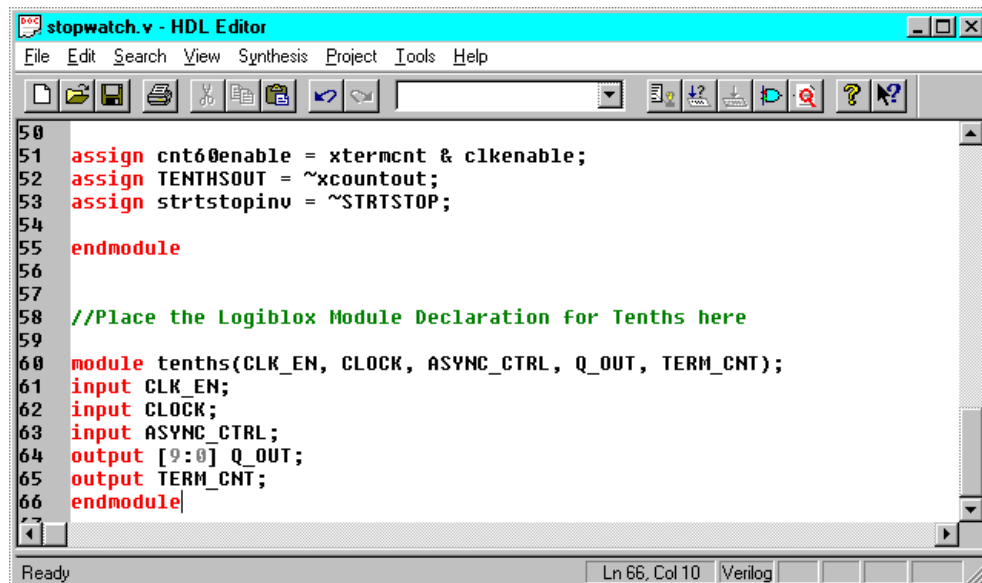
Verilog Flow

39. If you have closed the HDL Editor, open STOPWATCH.V.
 40. Place your cursor after the line that states:

“//Place the LogiBLOX Module Declaration for Tenths here” (This line is at the end of the file.)

Select **Edit** → **Insert File** and choose Tenths.vei. The Verilog template file for the LogiBLOX instantiation is inserted.

The Component Declaration does not need to be modified.



```

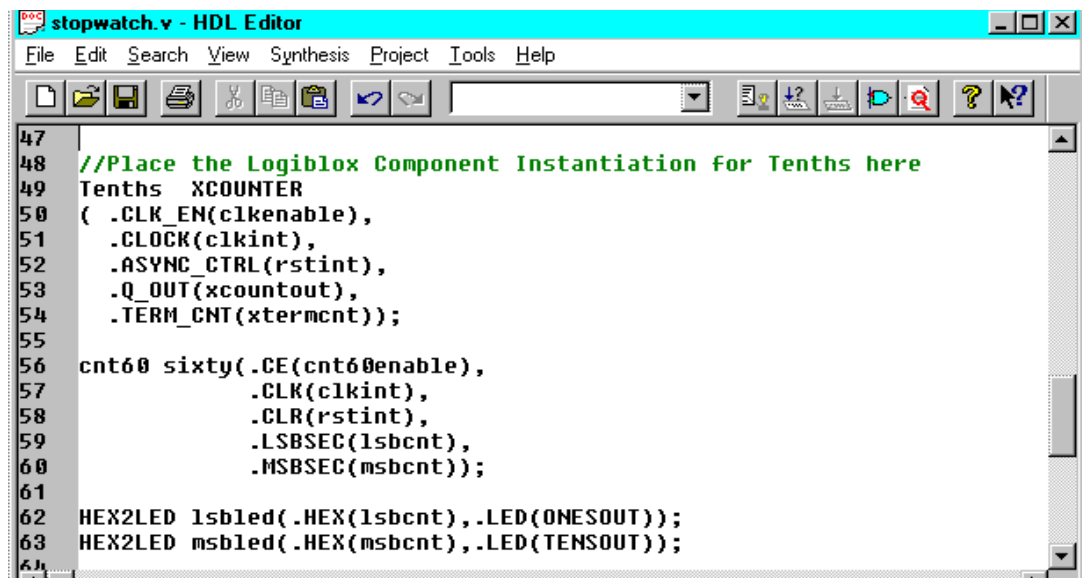
50
51 assign cnt60enable = xtermcnt & clkenable;
52 assign TENTHSOUT = ~xcountout;
53 assign strtstopinv = ~STRTSTOP;
54
55 endmodule
56
57
58 //Place the Logiblox Module Declaration for Tenths here
59
60 module tenths(CLK_EN, CLOCK, ASYNC_CTRL, Q_OUT, TERM_CNT);
61 input CLK_EN;
62 input CLOCK;
63 input ASYNC_CTRL;
64 output [9:0] Q_OUT;
65 output TERM_CNT;
66 endmodule

```

Figure 11 Verilog Module Declaration of LogiBLOX Module

Note: Alternately, the remaining module declaration can be placed in a new Verilog file (name it TENTHS.V) and added to the project. Be careful not to overwrite the Verilog simulation model, also named TENTHS.V, if one has been created. This module declaration is required to define the port directions of the ports of the LogiBLOX module.

41. Highlight the inserted code from “Tenths instance_name” to “.TERM_CNT());”. Select **Edit** → **Cut**.
42. Place the cursor after the line that states:
“//Place the LogiBLOX Component Instantiation for Tenths here.”
Select **Edit** → **Paste** to place the instantiation here.
Change “instance_name” to “XCOUNTER”.
43. Edit this code to connect the signals in the Stopwatch design to the ports of the LogiBLOX module. The completed code is shown in the following figure.



```

47
48 //Place the Logiblox Component Instantiation for Tenths here
49 Tenths XCOUNTER
50 ( .CLK_EN(clkenable),
51   .CLOCK(clkint),
52   .ASYNC_CTRL(rstint),
53   .Q_OUT(xcountout),
54   .TERM_CNT(xtermcnt));
55
56 cnt60 sixty(.CE(cnt60enable),
57             .CLK(clkint),
58             .CLR(rstint),
59             .LSBSEC(1sbcnt),
60             .MSBSEC(msbcnt));
61
62 HEX2LED 1sbled(.HEX(1sbcnt),.LED(ONESOUT));
63 HEX2LED msbled(.HEX(msbcnt),.LED(TENSOUT));
64

```

Figure 12 Verilog Component Instantiation of LogiBLOX Module

44. Save the design and close the HDL Editor.

Simulating the HDL

At this point, you could use a behavioral simulator to simulate your HDL. (This is not a part of this lab, however. This information is provided for your later use.) Application notes are available from two companies, describing the use of their simulators with Foundation Series.

- Active-VHDL, from Aldec. Download the application note (in zip format) from the Aldec web site: <http://www.aldec.com/Support/xilinx.zip>
- ModelSim, from Model Technology. Download the application note (in pdf format) from the MTI web site: <http://www.model.com/pdf/108xilinxfnd.pdf>. Or visit the Model Technology tech-notes page: <http://www.model.com/support/technote.html>, and click on the [ModelSim and Xilinx Foundation](#) link.

If you have Active-VHDL loaded, it is accessible via the **Tools** → **Simulation/Verification** pulldown menu from the Foundation Series Project Manager.

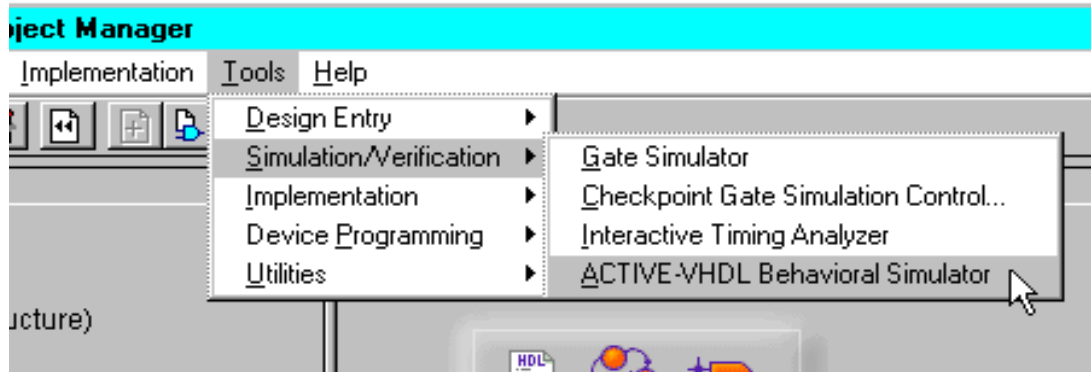


Figure 13 Active-VHDL from Project Manager menu

Synthesis and Implementation

Project Management (Versions tab - reference)

Project management controls design versions and revisions. A version represents an input design netlist. For HDL, versions represent synthesis runs. Each time a change is made to the source design, such as logic being added to or removed from the schematic or the HDL source being modified, a new version is created. A revision represents an implementation on a given version, usually with new implementation options, such as different placement or router effort level.

Foundation Series maintains revision control, meaning that the resulting files from each implementation revision are archived in the project directory.

Note: The project management tools currently support revision control for implementation data only. To save your source code changes for future use, you should archive the source HDL code and/or synthesized netlists(s) manually.

Foundation Series manages and displays your design versions and revisions graphically in the Versions tab of the Project Manager. Since you have not yet implemented the design, the Versions tab is currently empty.

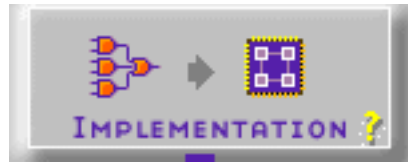
Preparing Synthesis

Now that the design has been entered and analyzed, the next steps are to synthesize and implement the design. In this lab, we will “pull” the design completely through synthesis and implementation. However, it is also possible to synthesize and implement the design in separate steps, if you want to.

Note: you must have the PCM.EXE patch installed before doing this section. See the handout: “*Foundation Series training preparation*” for details about the PCM.EXE patch.

45. Set the global synthesis options by selecting **Synthesis** → **Options**. Set the Default Frequency to 20, and check the Export Timing Constraints box. Click **OK** to accept these values.
46. Click the + next to STOPWATCH.VHD (or STOPWATCH.V). This shows the entities (or modules) within the HDL file. Some files may have multiple entities (or modules).

47. Select the entity named “stopwatch” and click the Implementation button from the Flow tab.



This step will cause the flow to proceed first through Synthesis, then Implementation. Synthesis may also be done separately, by clicking the Synthesis button under the flow tab, or by right-clicking on the entity name, and choosing Synthesize from the resulting menu.

48. In the Synthesis/Implementation window, complete the Target Device fields with this information:

- Family: XC4000E
- Device: 4003EPC84
- Speed Grade: -3

49. Check the boxes labeled Edit Synthesis/Implementation Constraints and View Estimated Performance after Optimization.

Note: Selecting the Edit Synthesis/Implementation Constraints box automatically opens the Express Constraints Editor after synthesis is complete.

Note: Selecting the View Estimated Performance after Optimization box automatically opens the Optimized dialog box which displays the results of the synthesis and optimization.

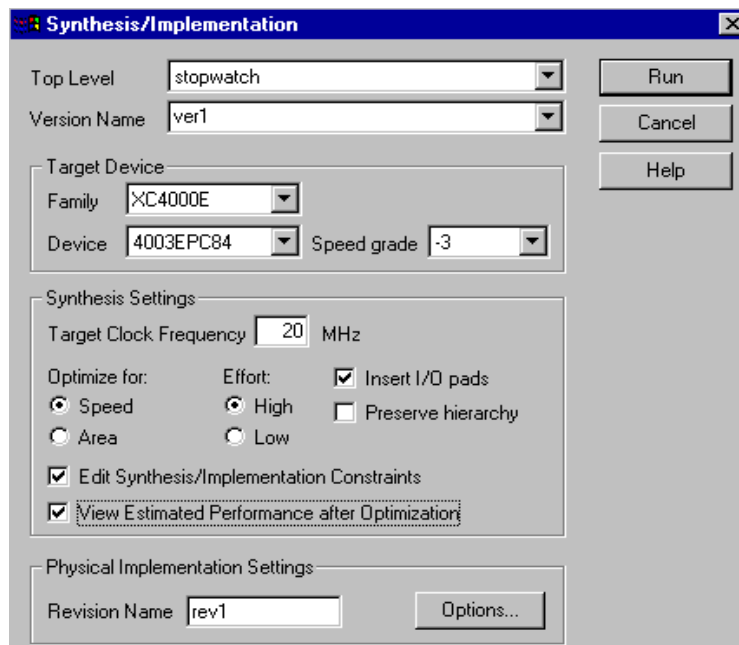


Figure 14 Synthesis/Implementation Window

50. Next you will set Implementation options. Do not run the implementation yet.

Preparing Implementation

Design Implementation is the process of translating, mapping, placing, routing, and generating a BIT file for your design. The Design Implementation tools are now embedded into the Foundation Series Project Manager for easy access and project management.

51. In the Physical Implementation Settings portion of the Synthesis/Implementation dialog box, notice the Revision Name field is automatically filled in. If you want to use another name, enter it in the box.

Implementation Options

52. Click the **Options** button. The Options dialog box opens. A summary of the options provided in this box follows.

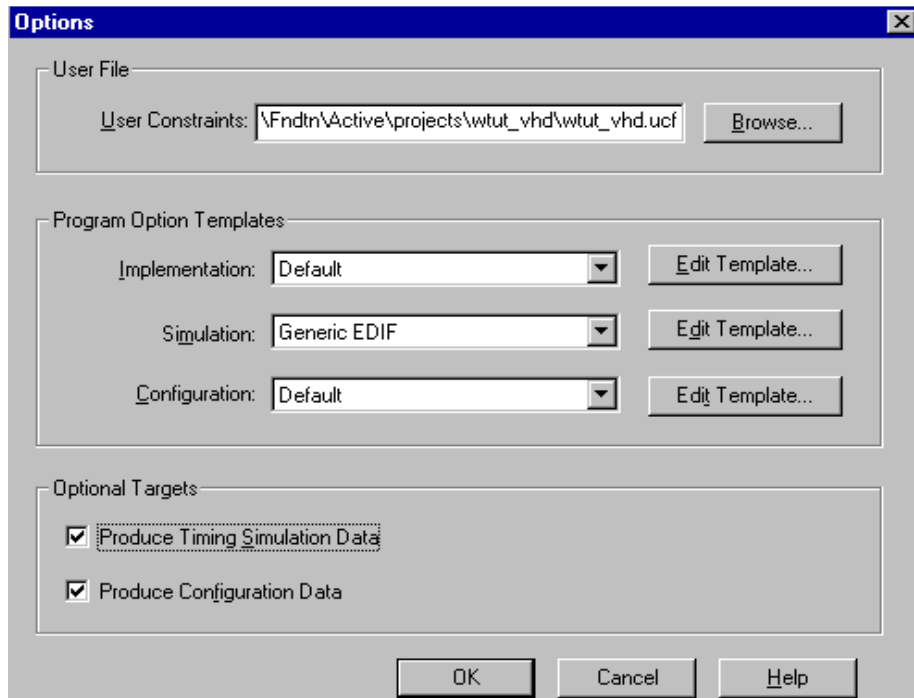


Figure 15 Implementation Options Dialog Box

- User Constraints File (.ucf). By default, Foundation Series creates a blank UCF file in the project directory. You can edit this UCF file from the Files view in the Project Manager. Because the name of this UCF file is the same as the project name, it may already be loaded in the Options dialog box, above. If you have other UCF files that you want to use instead, browse to find and select them.
- Program Option Templates. If you wish, enter and modify implementation options by using the Program Option templates.
- Optional Targets. You can specify whether you want to generate a Timing Simulation netlist for back-annotated timing simulation, as well as Configuration Data, which is the design's .bit file suitable for device programming. For this lab, you must make sure to check "Produce Timing Simulation Data."

53. Click **OK** to close the Options dialog.

54. Launch Synthesis/Implementation by clicking **RUN**.

In the following process of Synthesis and Implementation, Express first synthesizes the design and opens the Express Constraints Editor (since you clicked the Edit Synthesis/Implementation Constraints option earlier).

The Express Constraints Editor (information for reference)

You control optimization options and pass timing specifications to the Place and Route software through a GUI in the Express Synthesis software. This editor is only available with the Foundation Express product, not with Base Express. All timing specifications are passed in the netlist directly to the place and route engine and are used in the synthesis process for timing estimation purposes.

Constraints tabs are:

- Clocks

The Default Frequency set in **Synthesis** → **Options** is applied to all clocks in the design. To change the specification of a clock, click inside the box to the right of the clock and select Define. Enter the clock period or give the rise and fall times.

- Paths

All types of paths that can be covered by timing specifications are listed here, with unique specifications given for each clock in the design. To modify these specifications, enter a new delay in the Req. Delay column.

To create a subpath within a path, right click the source or destination and select New Subpath. Give the subpath a new name and delay value, then select sources and destinations by double clicking the instances. You can also use wildcards in the selection filters to choose a group of elements.

- Ports

With the Ports tab, you set input and out delay requirements, assign clock buffers, insert pullup or pulldown resistors in the I/O, set delay properties for input registers, set slew rate, disable the use of I/O registers, and assign pin locations. For all but the pin locations, click in the box to use the pulldown menu. For pin locations, type the pin number in the box.

- Modules

With the Modules tab, you choose to keep or eliminate hierarchy and disable resource sharing. You can also override the default settings for effort and area versus speed at the module level.

- Xilinx Options

The Ignore unlinked cells during GSR mapping option directs Express to infer a global reset signal (and, therefore, insert the STARTUP module), even if black boxes have been instantiated. Express cannot know the reset characteristics of any logic in black boxes, so it will not insert STARTUP unless you check this option.

Using the Express Constraints Editor (tutorial instructions)

Xilinx recommends that you let the automatic placement and routing program, PAR, define the pinout of your design. Pre-assigning locations to the pins can sometimes degrade the performance of the place-and-route tools. However, it is usually necessary, at some point, to lock the pinout of a design so that it can be integrated into a PCB (printed circuit board).

Define the initial pinout by running the place-and-route tools without pin assignments, then locking down the pin placement so that it reflects the locations chosen by the tools. Assign locations to the pins in the Watch design so that the design can function in a Xilinx demonstration board. Because the design is simple and timing is not critical, these pin assignments do not adversely affect the ability of PAR to place-and-route the design. For HDL-based designs, these pin assignments can be done in a User Constraints File

(UCF) or with the Express Constraints Editor. Although .UCF files are provided for this tutorial, you will assign the pin location constraints in the Express Constraints Editor.

55. In the Express Constraint Editor, click the Import Constraints button. Select WATCHVHD.EXC or WATCHVER.EXC, depending on the language you are using. These files are located in the project directory.

This file has been created for you during a previous synthesis run. The only difference you should see between your initial constraints and the ones saved in the .EXC file is the set of pin locations under the Ports tab.

You can save Constraint Editor settings for a design by clicking the **Export Constraints** button. When this .EXC file is read in for a later synthesis run, all constraints are re-established in the GUI, as long as they can be matched to instances in the current version.

56. Under the Paths tab, click in the box directly below the Req. Delay header. Change the delay to 35. Under the Ports tab, the Input Delays for RESET and STRTSTOP have changed to 35, as these represent all the Pad to Setup delays.

You can change the values of individual Input or Output Delays by clicking the value in the Ports tab and either editing the value there or using the pulldown tab to select a value or define a new one. Change the values on one of the output signals using one of these methods.

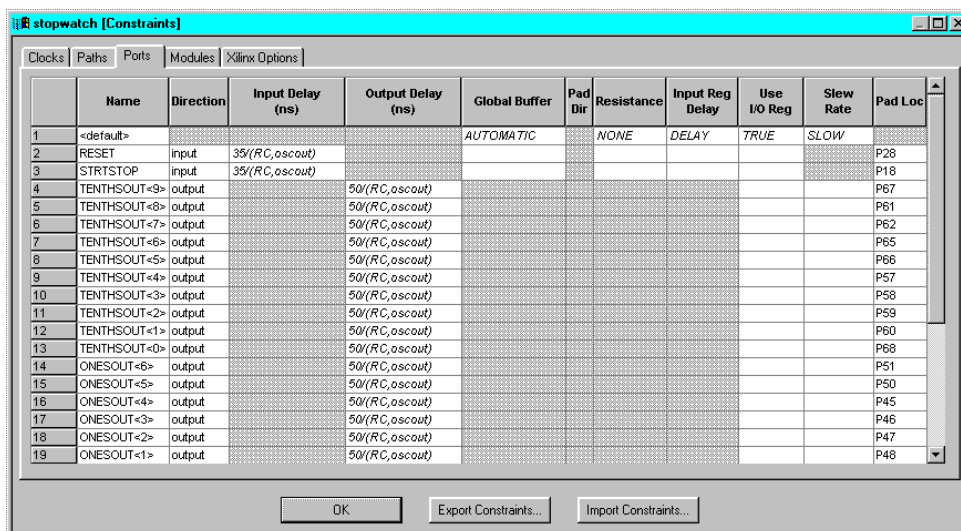


Figure 16 Ports Tab Display

57. Under the Paths tab, right click on the RC-oscout to All Output Ports row and select **New Subpath**. The Create/Edit Timing Subpath window opens.

Give this new subpath a name, Sub_flops_to_out, and a Delay value, 30. On the left hand side, double click all four flip flops that contain the name /ver1/sixty/lbcount/qout*, to determine the sources of this subpath. On the lower right hand side, use the filter to select the destinations. Type **ONE*** in the field and click the **select** button. All the ports beginning with ONESOUT will be highlighted. Click **OK** to see your new subpath.

You have now created a new subpath between the LSBSEC port (defined in your design source file "cnt60") and its outputs.

Note: Base Express users cannot access the Express Constraints Editor. Pin location constraints must therefore be defined in a UCF file, which Xilinx has provided. Select **Implementation → Implementation Options**. Click the Browse button next to User Constraints and select **BASE.UCF**.

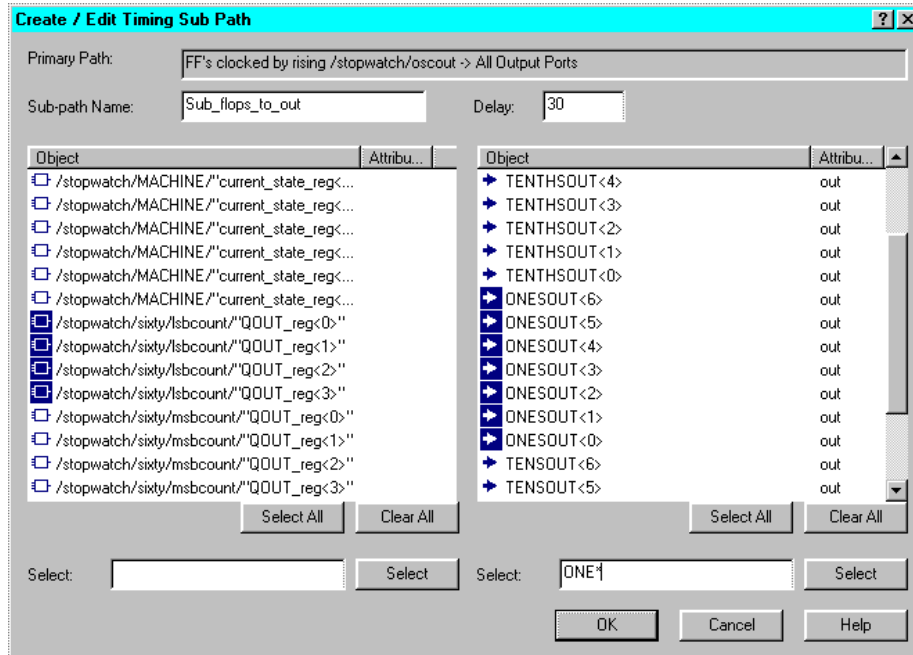


Figure 17 Editing Subpath in the Express Constraints Editor

58. Under the Ports tab, add the two final pin locations: RESET must be assigned to P28, and STRTSTOP must be assigned to P18. To reassign, click the box and enter the pin number (including the P).

Note: The remaining I/Os have pin assignments. This information is contained in the .exc file, which you imported in Step 1.

59. Click **OK** to continue synthesis. Express now optimizes the design.

Viewing Synthesis Results

Since you checked the View Estimated Performance after Optimization box earlier, the Express Constraints Editor opens after the optimization phase of synthesis with preliminary performance results. The delay values are based on wireload models and, therefore, must be considered preliminary. Consult the post-route timing reports for the most accurate delay information.

60. Under the Clocks tab, examine the estimated delay value of the clock. Delays greater than the specification appear in red.
61. Under the Paths tab, examine the estimated delays for the paths and subpath. Click the source or destination of a path to see the members of the path, and click a specific path to see the individual segments of that path.

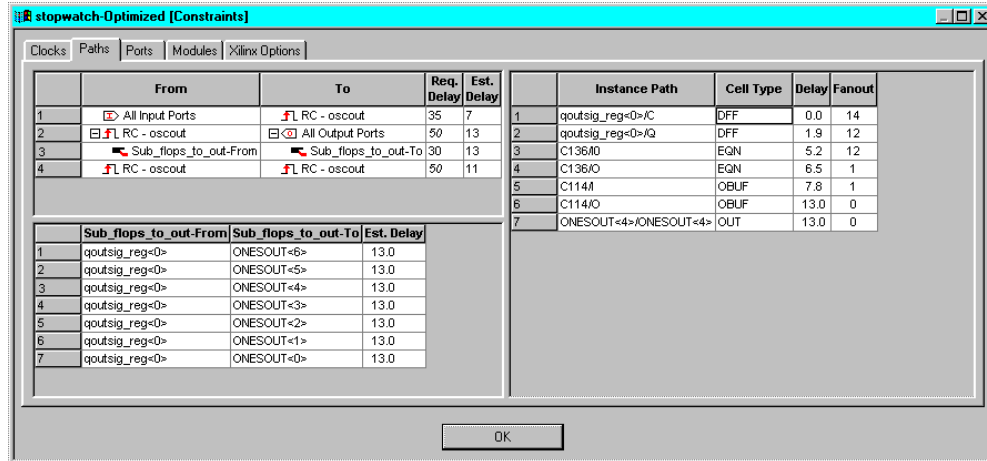


Figure 18 Estimated Timing Data Under Paths Tab

62. Examine the Ports tab to see that all of the settings and delays have been assigned and met.
63. Under the Modules tab, you can examine the elements used to synthesize this design. Click the box in the second row under Area and select **Details**. This section summarizes all the design elements used in the Stopwatch design that Express knows about. Since the Tenths module is a LogiBLOX component and has not been synthesized by Express, it is UNLINKED and no summary information is available.

Note: Black boxes (modules not read into the Express design environment) are always noted as UNLINKED in the Express reports. As long as the underlying netlist (.xnf, .ngo, .ngc or EDIF) for a black box exists in the project directory, the Implementation tools merge the netlist during the Translate phase. Since the Tenths module was built using LogiBLOX called from the project, the tenths NGC file will be found.

64. Click **OK** to complete the Synthesis phase.

At this point, an XNF file exists for the Stopwatch design. If desired, you may perform a post-synthesis simulation of this design using the Foundation Series functional simulator. The simulator used for functional simulation is the same one used for timing simulation, which you will use in the last section of this tutorial. The only difference is that the design which is loaded into the simulator for timing simulation contains worst-case routing delays based on the actual placed and routed design.

Running Implementation — The Flow Engine

65. Now that the Synthesis phase has completed, the Flow Engine displays and implementation begins.

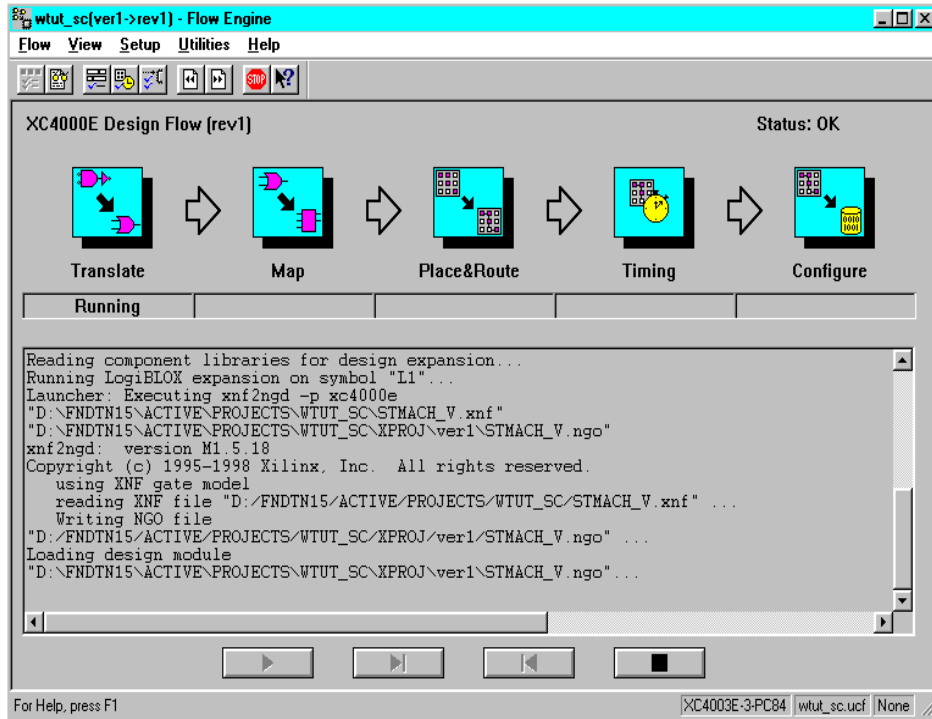


Figure 19 Flow Engine

When the implementation is complete, the Flow Engine closes automatically, and the Foundation Series Project Manager is fully visible again.

- 66. The status of the implementation is displayed in the console window at the bottom of the Project Manager. You should see (OK Implemented) and Completed Successfully for the version and revision. If you encountered any errors in the implementation, refer to the Implementation Log file for details on the error.

Viewing Implementation Results

The Foundation Series Project Manager maintains control over all of your design implementation versions and revisions. You can directly view and analyze these implementations from the Project Manager.

- 67. Click the Versions tab in the left-hand pane of the Project Manager. You should see a hierarchical display of the implementation you just ran. The revision that is most current is displayed in bold.

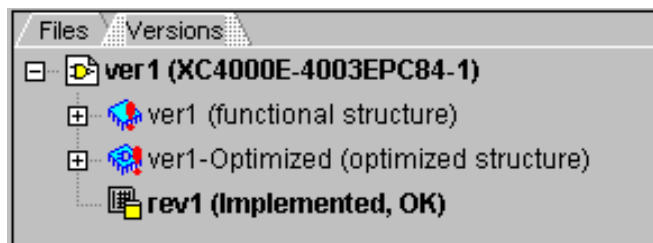


Figure 20 Versions Tab

- 68. With the current revision selected, click the Reports tab in the right-hand side of the Project Manager. The Reports tab displays reports and logs for the selected revision of the design.

69. Double click the report entitled Implementation Report Files. This displays the Xilinx Report Browser, which contains all of the implementation reports. You have the option to browse through any of these reports at this time.

Other Implementation Tools (reference)

The Foundation Series Project Manager also gives you access to the other implementation tools, including the Timing Analyzer, EPIC Design Editor, Floorplanner, JTAG Programmer, Prom File Formatter and Hardware Debugger. These tools can be invoked from the **Tools** → **Implementation and Tools** → **Device Programming** menus. The Timing Analyzer and Device Programming tools are also available from the Flow diagram.

These implementation tools are sensitive to the implementation revision. That is, depending on which Revision you have selected in the Versions tab when you invoked the tool, the tools will be loaded with data from that implementation revision.

Now you can invoke any of these tools to see what they look like. For more information, refer to the appropriate online documentation for each tool.

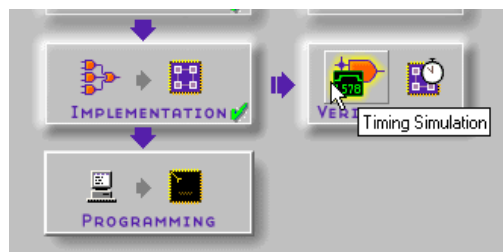
Timing Simulation

Timing simulation uses the block and routing delay information from the routed design to give a more accurate assessment of the behavior of the circuit under worst-case conditions. For this reason, timing simulation is performed after the design has been placed and routed.

The following section of the lab includes the Script Editor, Script Wizard, and Simulator. These tools, used for timing simulation, are the same ones used for functional simulation (which may be run following design entry). The only difference is that the design which is loaded into the simulator for timing simulation contains worst-case routing delays based on the actual placed and routed design.

Invoking Timing Simulation

1. To invoke the timing simulator, click the Timing Simulation icon in the Verification phase button in the Project Manager Flow diagram.



The simulator is now loaded and ready to simulate. For this simulation, you will use script files.

Simulating with Script Files - Script Wizard and Script Editor

The Foundation Series functional simulator provides simulation by applying various types of stimulus including keyboard stimulus, formulae, and by using the internal binary counter. In this section, you use a script file to simulate the design, and will use the new Script Wizard feature from the Script Editor.

2. To invoke the Script Editor, select **Tools** → **Script Editor** from the pulldown menus within the Simulator. A dialog box prompts you to select a script file.

3. Choose **Use Script Wizard** to invoke the Script Wizard.
4. Follow the instructions in the Wizard to advance to the Initialization page.
5. On the Initialization page, select the following options.
 - **Delete Existing Signals** — clears all the waveforms at the start of each simulation.
 - **Restart (Power On)** — forces the simulator to perform a global reset at the start of the simulation to initialize all of the registers.
 - **Simulation Mode: Timing**
 - **Step Size: 30 ns** — determines the size of the simulation step.
 - **Generate additional comments** — inserts comments into the script file to aid you in further editing of the script file.
 - **Script File Description** — type “Simulation Script File for Watch Tutorial.” Whatever you type here will be placed as a comment at the top of the script file.

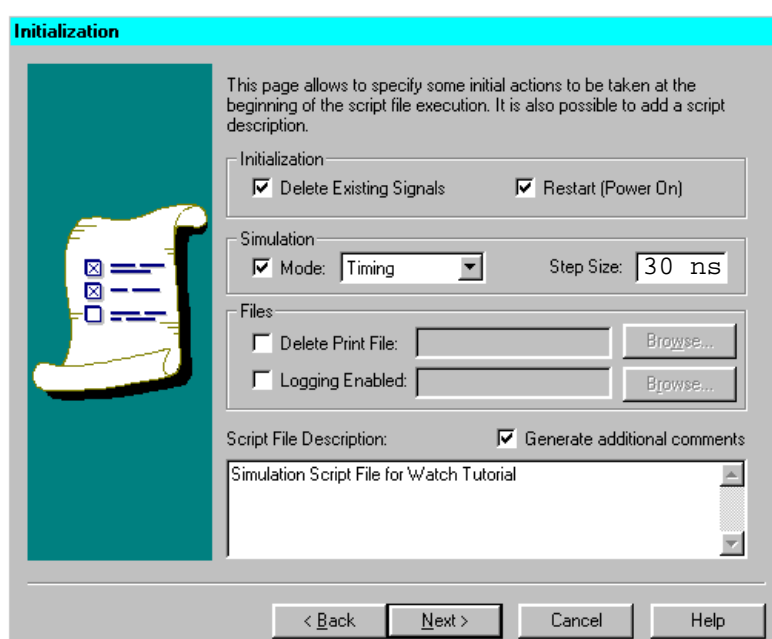


Figure 21 Script Wizard -- Initialization

6. Click **Next** to advance to the Vectors page.

Vectors provide a more convenient way to use buses in the script file. By defining vectors, you can more easily refer to these buses in the rest of the script file. You can also create vectors out of any group of signals, regardless of whether they are a bus in the original design.

In this step, you define vectors for the three output buses, ONESOUT[6:0], TENSOUT[6:0], and TENTHSOUT[9:0]. For simplicity, name these vectors ONES, TENS and TENTHS, respectively.
7. Click the **New** button. This adds a new vector to the vector list entitled Vector_Name_1 by default. Type **TENS** in the place of Vector_Name_1 to rename it.
8. Click the **Browse . . .** button. This displays a Component Selection window which contains all of the signals in the design. On the right-hand side, scroll down to find the TENSOUTS6 [7-bit] bus. Select this bus, and then click **OK**. By doing this, you have assigned the seven bits of the TENSOUT bus to the newly created TENS vector.

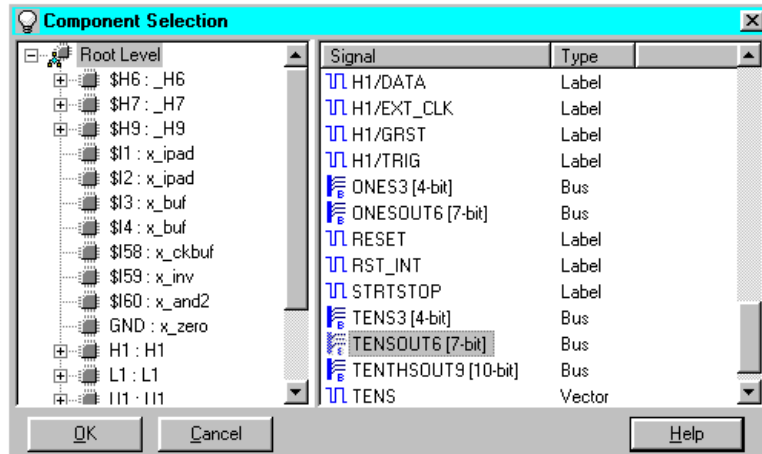


Figure 22 Script Wizard Component Selection

The seven bits of the TENSOUT bus are listed as components in the newly created TENS vector.

9. With the TENS vector selected, click the **Radix** pulldown menu to change the radix of the vector to Binary. This determines how the vector is displayed in the simulator.
10. Repeat Steps 6 through 8 to create vectors called ONES and TENTHS for both the ONESOUT[6:0] and TENTHSOUT[9:0] buses, respectively.

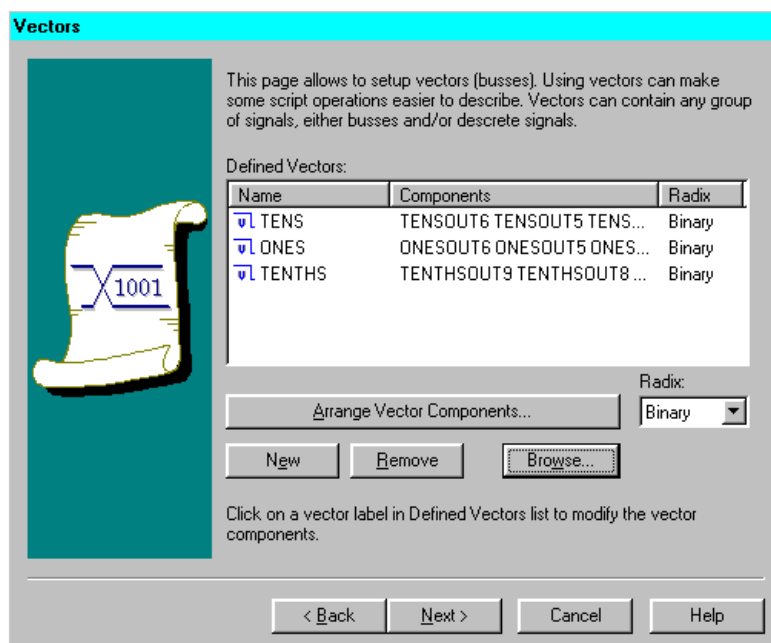


Figure 23 Script Wizard Vectors

11. Click the **Next** button to advance to the Stimulators page.

Stimulators define the action of the inputs in the design. There are several different commands that can be used to define input stimulus. You will use three different methods in this tutorial. For a complete description of all available commands, refer to the online help.

12. To select the first signal to stimulate, click the **Browse . . .** button.
13. In the Component Selection window, scroll down the signal list on the right-hand side, and locate the CLKINT signal. Select it and click **OK**.

14. See the CLKINT signal listed in the Stimulators and Watched Signals list. Click the CLKINT signal and the Stimulator Type field now becomes active. Use the pulldown menu in the Stimulator Type field to select Clock.
15. In the Value field, set the pattern of the clock. By typing 0 1 (with a space between them) in the value field, you define the clock as having a pattern of low for one simulation step (previously defined as 10ns), then high for one simulation step. This pattern repeats indefinitely to produce the clock signal.

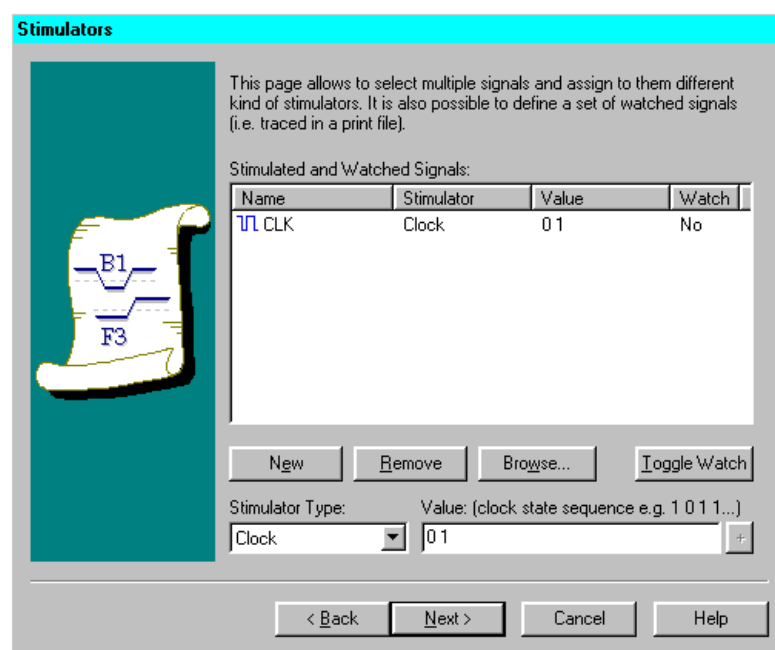


Figure 24 Clock Stimulus

16. Repeat Steps 12 and 13 to add the STRTSTOP signal to the Stimulated signals list.
17. With the STRTSTOP signal selected in the Stimulated Signals list, set the Stimulator Type to Aldec Waveform.
18. In the Value field, type the following:

H200L30H500L30H500L30H3000

Similar to the Custom Formula you can create for functional simulation, this waveform means high for 200ns, then low for 30ns, then high for 500ns, and so on. This waveform will define a stimulus pattern for the STRTSTOP input signal.

19. Repeat Steps 12 through 13 to add the RESET signal and the GSRT signal to the Stimulated Signals list.
20. With the RESET signal selected in the Stimulated Signals list, set the Stimulator Type to be Waveform.
21. In the value field type the following.

@0=0 9500=1 400=0

This means “at 0ns the signal is 0, 950ns later the signal is high, 40ns later the signal is low.” Note that the units of this measurement are tenths of nanoseconds. This waveform provides a reset pulse to reset the stopwatch during the simulation.

In the same manner, set the GSRT Stimulator Type to be Waveform. In the value field, type:

@0=0 400=1

22. The Stimulators page also allows you to select signals which you wish to “watch” in a printed output file. Since you will be setting a printed output file in the next section of the Wizard, you will add more signals to this list so that they may be watched.

Repeat Steps 12 and 13 to add the TENS, ONES, and TENTHS vectors to the Stimulated and Watched Signals list. Be sure that you add the *vectors* and not the *buses*.

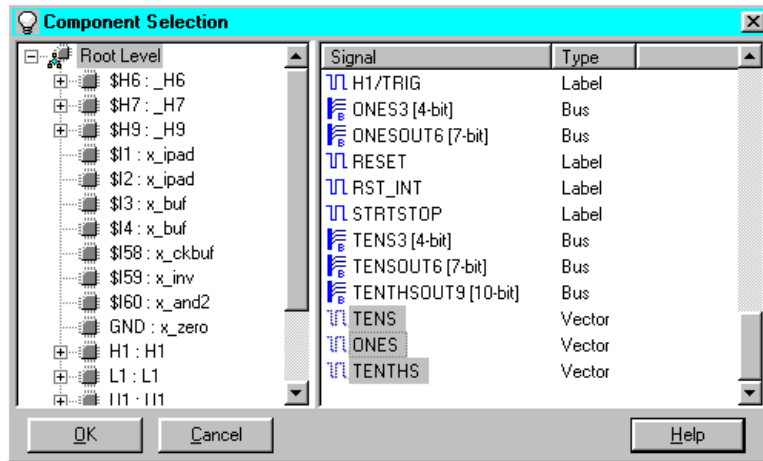


Figure 25 Selecting Vectors to Watch

23. Because the TENS, ONES, and TENTHS vectors are outputs, they should not have stimulus assigned to them. Select each of these vectors individually and set the Stimulator Type to be None.
24. You should now see six signals listed in the window. Select them all by performing a Shift-Click and then click the **Toggle Watch** button. This changes the watch status of all of them from No to Yes.

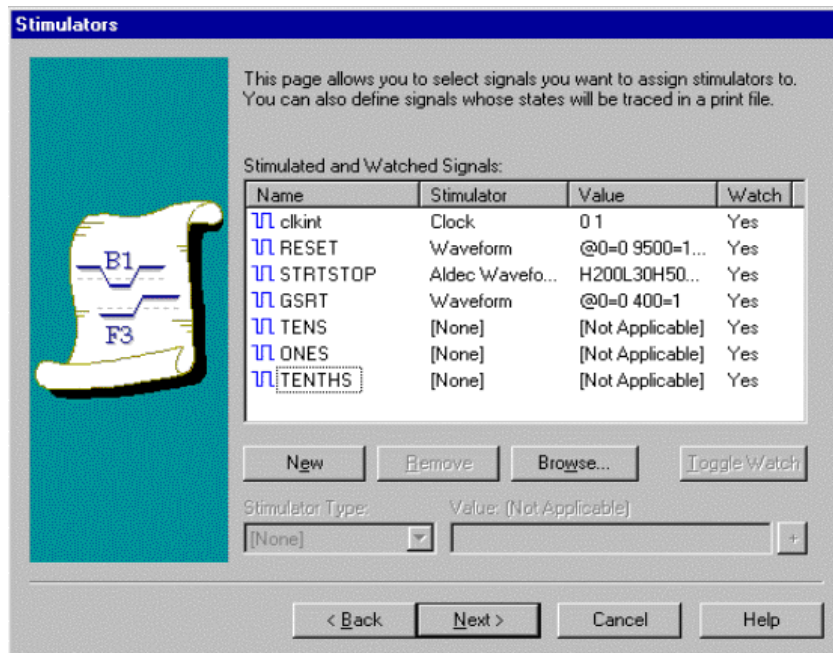


Figure 26 Signals' Stimulus

25. Click **Next** to advance to the Breakpoints and Simulation page.

Breakpoints allow you to monitor the simulation for some output response. You can specify how the simulator will notify you when the output response is detected.

26. On the Breakpoints and Simulation page, click the **Browse . . .** button to choose the first signal to set a breakpoint on.
27. In the Component Selection window, choose the ONES vector from the signal list and click **OK**.
28. You should now see the ONES vector listed in the Defined Breakpoints list. Highlight ONES, and then from the Condition pulldown menu, select **Low state**. This defines the condition which must be present on the ONES vector for the breakpoint to occur.
29. In the Action field, type the following:

print > tim_out.txt

This tells the simulator to write out an output report called tim_out.txt whenever the breakpoint condition is met.

30. Set the Simulation Command to Cycle, and the Simulation Value to 400. This tells the simulator to run for 400 clock cycles.

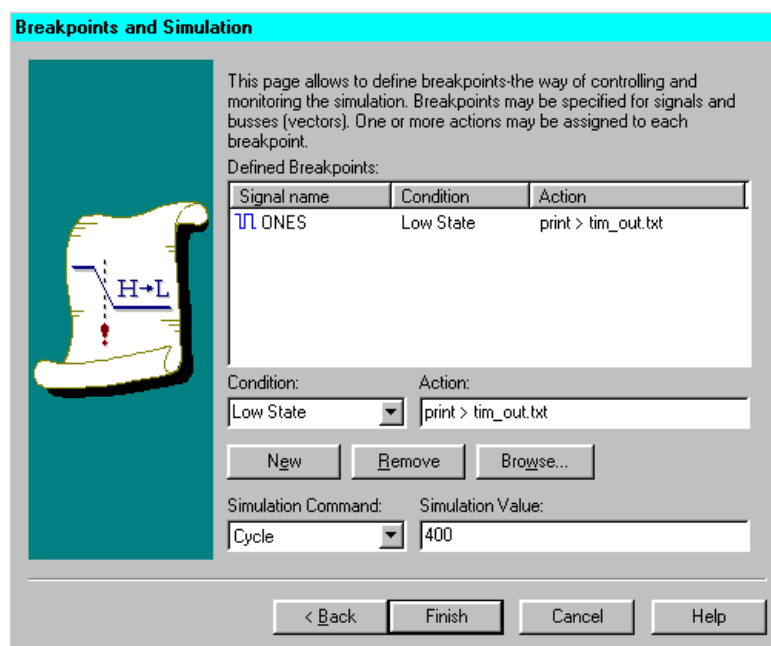


Figure 27 Breakpoints and Simulation

31. Click **Finish**. You can now view your completed script file in the Script Editor.

Viewing the Script File with the Script Editor

32. Save the script file that was created by the Script Wizard by selecting **File** → **Save**. Be sure that the file is being saved into the current Foundation Series project directory (that is, C:\Fndtn\Active\projects\watch_proj_name). Name the script file watchtim.cmd.
33. Look through the script file to see what the Script Wizard created.

Running the Simulation from the Script Editor

34. You can execute the simulation directly from the Script Editor. To do this, select **Execute** → **Go**.

A log of the executed commands appears at the bottom of the Script Editor, including messages indicating when breakpoints were encountered.

35. To view the simulation results in the Waveform Viewer, move the Script Editor window and bring the Waveform Viewer window to the front of your view. Inspect the simulation results to make sure they are accurate. Note that the TENTHS output is active-low and you are viewing the inverted value in the simulator. Also, you are viewing the HEX2LED result of ONES and TENS.

You should now see that this is indeed performing a timing simulation based on actual delays in the placed and routed design. If you zoom in to get a closer view of the waveforms, you will see that there is a delay from the rising edge of the clock to the transitions or the counter outputs.

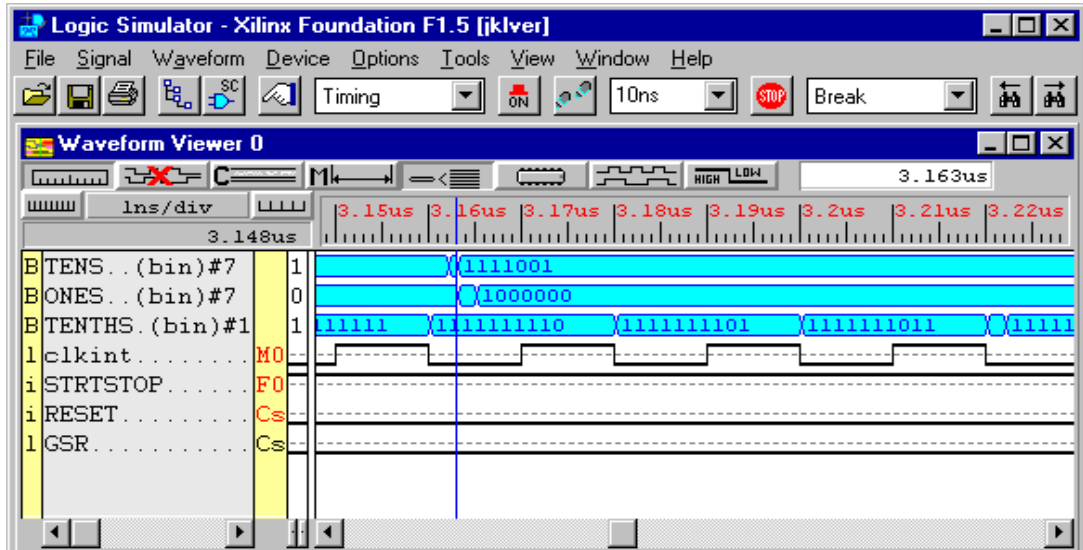


Figure 28 Timing Simulation Waveforms

For more detailed information related to actual path delays and system performance requirements, you can use the Xilinx Timing Analyzer to do Static Timing Analysis.

Viewing the Printed Output File

As previously mentioned, you set a breakpoint action to write to a printed output file called tim_out.txt. This file is a text file that is viewable in any text editor. You can use the Script Editor or any other text editor to view this file.

To view this file from the Script Editor, select **File** → **Open** from the Script Editor and set the File Type filter to *. *. Locate the file tim_out.txt, and click **Open**.

This file is a printed output file in the form of a state table, showing the states of all the “watched” signals at the times at which breakpoints were encountered. The times at the five breakpoints should match the times listed in the log console area of the Script Editor when the simulation was originally run. You should still be able to see the console messages to verify this.

Closing the Simulator

36. When you are satisfied with the results of the simulation, you may close the Script Editor and the Simulator.

You have now completed the Foundation Series 1.5i lab, showing the new HDL flow, plus new features of the Script Wizard and Design pull-through. More new features are included with Foundation Series 1.5i. Please experiment with other parts of the software and ask the lab instructor if you have questions!