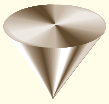
XILINX®

ONLINE

XEPLD VIEWSYNTHESIS DESIGN GUIDE

TABLE OF CONTENTS

INDEX



GO TO OTHER BOOKS

XACT™ STEP

0401419

Contents

Appendix C Library Component Specifications

Chapter 1

System Configuration

This chapter describes the capabilities of the Xilinx® EPLD Viewlogic Synthesis Library and Interface and shows you how to configure your system. For installation instructions, refer to the Release Notes.

Software Capabilities

The Xilinx EPLD Viewlogic Synthesis Interface has the following features:

- Supports all XC7200 and XC7300 devices.
- Functional simulation of synthesized gate-level designs (including all XC7000-specific library components).
- Full-timing simulation (after fitting).
- Static Timing Report created by the XEPLD™ Fitter (after fitting).
- Attributes for allocating logic to EPLD Fast Function Blocks and Fast Inputs.
- Attributes for controlling XEPLD optimization of clocks, input pad registers, output enable signals, and UIM™-AND functions.

Unsupported Features

The Xilinx EPLD Viewlogic Synthesis Interface currently has the following limitations:

- No technology-specific optimization (for speed or density) is performed by the Viewlogic synthesizer; all optimization is performed by the Xilinx EPLD Translator Core Tool (XEPLD).
- No timing or area information is contained in the XC7000 library. Therefore, no timing or area estimation is available from

Viewlogic Synthesis. Timing and resource utilization results are available from XEPLD after completion of fitting.

- The XEPLD fitter (v5.1) currently does not support timing-constraint-driven optimization; Viewlogic Synthesis timing constraints have no effect on EPLD design processing. Instead, use the "F" attribute to designate EPLD Fast Function Block and Fast Input resources.

System Requirements

You will need the following software installed on your system to develop EPLD designs:

- Viewlogic PRO series or Powerview with Viewlogic Synthesis v2.3.1 or later.
- Xilinx Viewlogic Interface and libraries version 5.1 PRO (all of the Xilinx Viewlogic system products contain this interface).
- Xilinx EPLD Implementation software (DS550) version 5.1 or later (all of the Xilinx Viewlogic system products contain this interface).

Configuring the Viewlogic Library Search Order

After you have installed the Xilinx software, you must configure the viewdraw.ini file (originally in your Viewlogic STANDARD directory) for accessing the XC7000 library. Each design directory where XC7000 designs are processed must contain a properly configured viewdraw.ini file.

The viewdraw.ini File

Copy the viewdraw.ini file from the Viewlogic standard directory into your project directory, or use the Viewlogic project management utility to create a project directory containing a copy of viewdraw.ini. Edit your local viewdraw.ini file so that it contains the following library directory pointers in the following order:

```
DIR [p] .                    (primary)
DIR [m] Xilinx_library_path\unified\xc7000 (xc7000)
DIR [m] Xilinx_library_path\unified\builtin (builtin)
DIR [m] Xilinx_library_path\unified\xbuiltin (xbuiltin)
```

On the workstation, use [r] instead of [m] in the library paths.

Verifying Your File Structure

Figure 1-1 shows the relative locations of the essential files included with the Xilinx Viewlogic Synthesis Interface and Libraries:

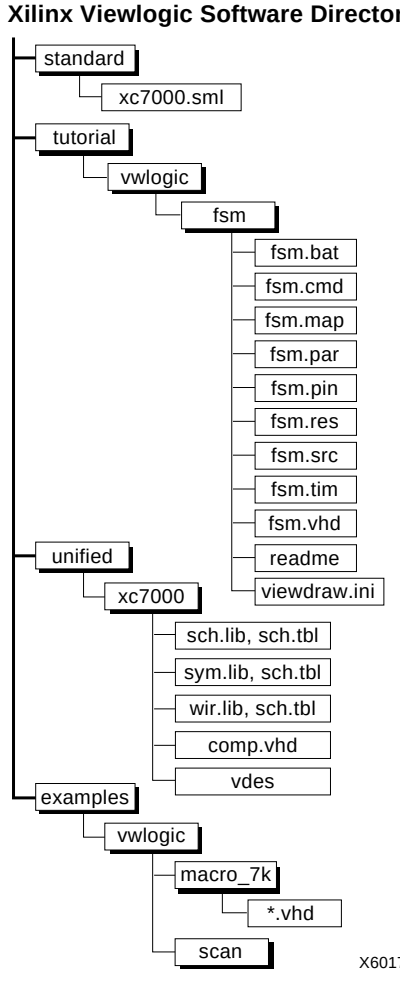


Figure 1-1 XEPLD Viewlogic Synthesis Interface (v5.1) Files

Chapter 2

Getting Started With Xilinx EPLDs

This chapter provides an overview of the basic steps for implementing Xilinx EPLD designs using PROsynthesis. The remaining chapters in this manual provide additional detailed information on these steps.

This overview assumes that you have installed and configured the Xilinx software and libraries. See the System Configuration chapter for instructions on how to verify your installation.

The examples in this chapter show the PRO series environment on a PC. You can also process EPLD designs using ViewSynthesis (for example, under Powerview on a workstation) using a similar sequence of steps.

Xilinx EPLD Design Flow

The basic steps for creating Xilinx EPLDs using Viewlogic Synthesis are described as follows:

1. Enter your VHDL design using a text editor.
2. Synthesize your design using PROsynthesis and generate a schematic for fitting.
3. Perform functional simulation using PROsim (optional).
4. Run the Xilinx fitter on the schematic and create a timing report and device programming file.
5. Perform timing simulation using PROsim (optional).

Figure 2-1 shows the basic design flow for creating EPLD designs. Each step is described in the following design example.

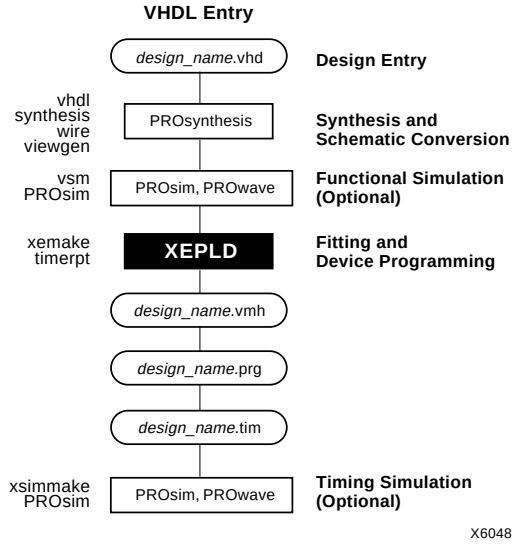


Figure 2-1 Basic EPLD Design Flow

Design Example

The following design example, fsm.vhd, is used to demonstrate the basic EPLD design flow. The design is implemented in a Xilinx XC7336-5PC44 device.

This design is a finite state machine that outputs a 1 only when the bitstream 11100 is present, ignoring all other 5-bit combinations. Figure 2-2 illustrates how this state machine works, and Figure 2-3 is a state diagram for the fsm.vhd design.

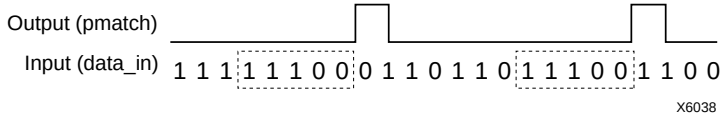


Figure 2-2 FSM Output vs. Input Relationship

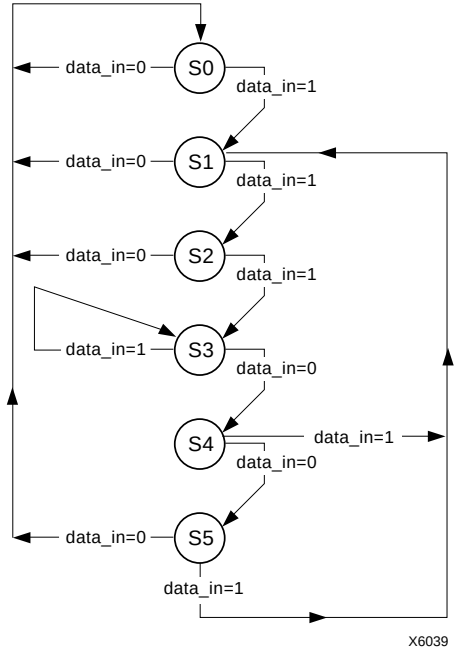


Figure 2-3 FSM State Diagram

Design Entry

The VHDL source file for the example design is shown in below. Note that this file contains IBUF and OBUF instances, which are necessary to identify all device input and output pins in all EPLD designs.

```
library synth;          -- Used for simulation
use synth.stdsynth.all;
library xc7000;         -- Used for EPLD I/O buffer components
-- use xc7000.components.all;

-- Entity: Input and Output Declarations.
-------------------------------------------------

entity fsm is
   port (
     signal reset, clk, datain       : in vlbit;
     signal pmatch                   : out vlbit
   );
end fsm;
```

```
-------------------------------------------------
-- The architectural body - behavioral description.

architecture first of fsm is

signal reset_in, clk_in, datain_in : vlbit;
signal pmatch_out : vlbit;

constant s0 : vlbit_1d(2 downto 0) := "000";
constant s1 : vlbit_1d(2 downto 0) := "001";
constant s2 : vlbit_1d(2 downto 0) := "010";
constant s3 : vlbit_1d(2 downto 0) := "011";
constant s4 : vlbit_1d(2 downto 0) := "100";
constant s5 : vlbit_1d(2 downto 0) := "101";

signal next_state : vlbit_1d(2 downto 0);
signal state      : vlbit_1d(2 downto 0);

begin

  i1: ibuf port map (reset_in, reset);
  i2: ibuf port map (clk_in, clk);
  i3: ibuf port map (datain_in, datain);
  o1: obuf port map (pmatch, pmatch_out);
  cntl: process(datain_in, state)
  begin
    pmatch_out <= '0';
    next_state <= s0;
-------------------------------------------------
-- Begin CASE statement

  CASE state(2) IS

   WHEN '1' =>
      IF state(1) = '1' THEN
       next_state <= s0; -- error condition
      ELSE
       IF state(0) = '0' THEN
         IF datain_in = '1' THEN
           next_state <= s1; --transition from "100" to "001"
         ELSE
           next_state <= s5; --transition from "100" to "101"
         END IF;
       ELSE
         pmatch_out <= '1';
         IF datain_in = '1' THEN
           next_state <= s1; --transition from "101" to "001"
         ELSE
           next_state <= s0; --transition from "101" to "000"
```

```
        END IF;
       END IF;
      END IF;

   WHEN '0' =>
      IF state(1) = '1' THEN
       IF state(0) = '0' THEN
         IF datain_in = '1' THEN
           next_state <= s3; --transition from "010" to "011"
         ELSE
           next_state <= s0; --transition from "010" to "000"
         END IF;
       ELSE
         IF datain_in = '1' THEN
           next_state <= s3; --transition from "011" to "011"
         ELSE
           next_state <= s4; --transition from "011" to "100"
         END IF;
       END IF;
      ELSE
       IF state(0) = '0' THEN
         IF datain_in = '1' THEN
           next_state <= s1; --transition from "000" to "001"

         ELSE
           next_state <= s0; --transition from "000" to "000"
         END IF;
       ELSE
         IF datain_in = '1' THEN
           next_state <= s2; --transition from "001" to "010"
         ELSE
           next_state <= s0; --transition from "001" to "000"
         END IF;
       END IF;
      END IF;

    WHEN OTHERS => next_state <= s0; -- error condition

   END CASE;

   end process cntl;
-------------------------------------------------
-- Seperate clocked process
-- state<=next_state
```

```
-------------------------------------------------
  cntl_clk: process
  begin
   wait until (prising(clk_in) or (reset_in = '1'));
        if (reset_in = '1') then
          state <= s0;
        else
          state <= next_state;
        end if;
        end process cntl_clk;

end first;
```

The example file is already created for you and is located in the directory: tutorial\vwlogic\fsm.

Copying the Tutorial Files

Before you start the tutorial, you should copy the files installed in tutorial\vwlogic\fsm to your own project directory. You can use DOS commands or the Windows File Manager to copy the files. In the examples that follow, the tutorial files are copied into c:\designs\fsm.

This will place the following four files, plus a few report files, into your new project directory:

- fsm.vhd — VHDL source design file
- fsm.cmd — PROsim simulation command file
- fsm.bat — batch file to automatically run the tutorial from DOS
- fsm.src — VHDLDES command file invoked by fsm.bat

Project Creation

To establish the FSM design as a Viewlogic project, follow these steps:

1. Start Windows by typing **`win`** at the DOS prompt.
2. Open the PRO Series window.
3. Open the Xilinx flow manager, PROflow, by double clicking on the **Psfm** icon.

The Xilinx PROflow window appears as in Figure 2-5:

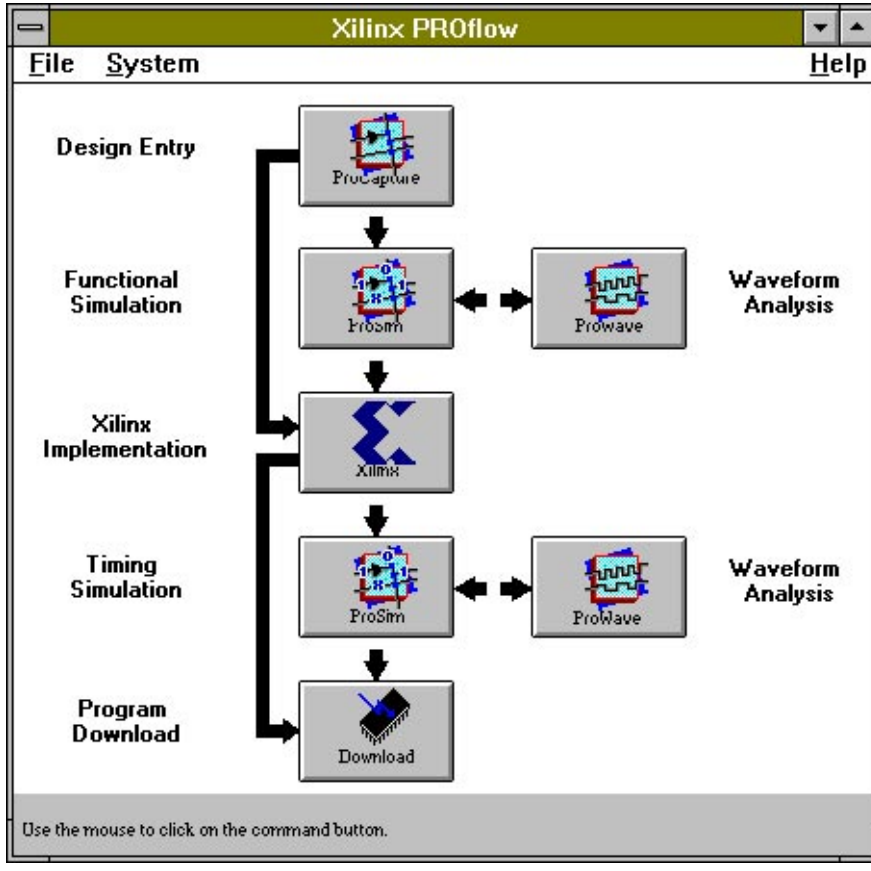


Figure 2-4 Xilinx PROflow Window

4. Select the **PROcapture** button in PROflow. The Design Entry dialog box appears as in Figure 2-6:

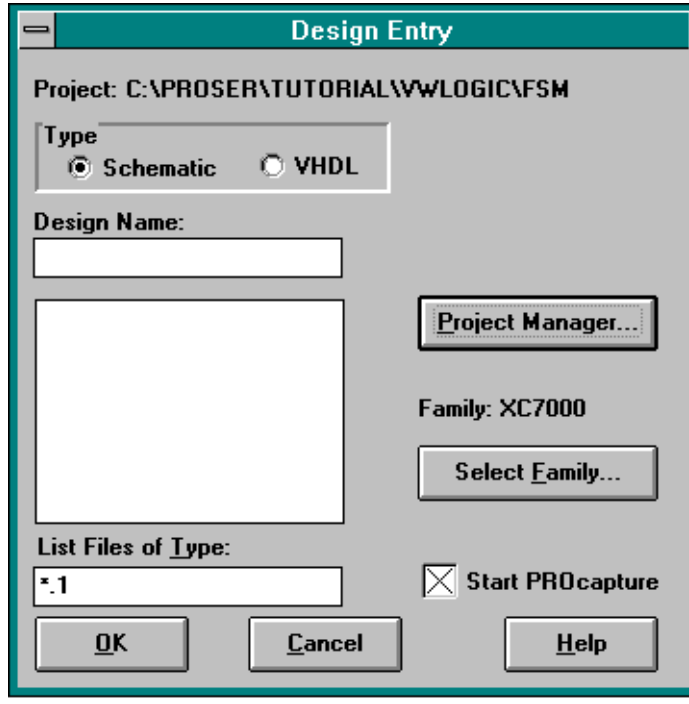


Figure 2-5 Design Entry Dialog Box

5. Click on the **Project Manager** button. The dialog box shown in Figure 2-7 appears:

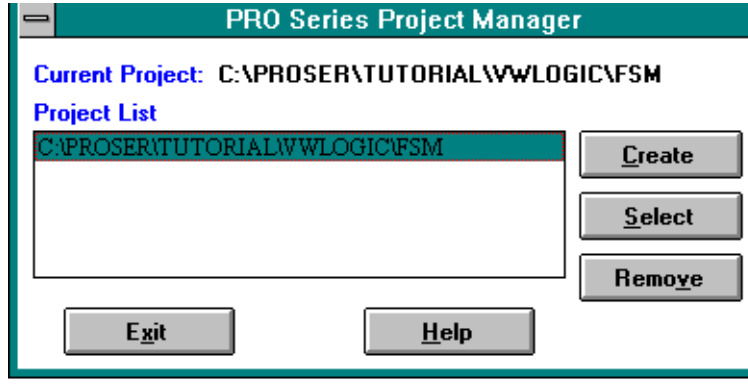


Figure 2-6 Project Manager Dialog Box

6. Select the **`Create`** button. This opens the Create Project dialog box as in Figure 2-8:

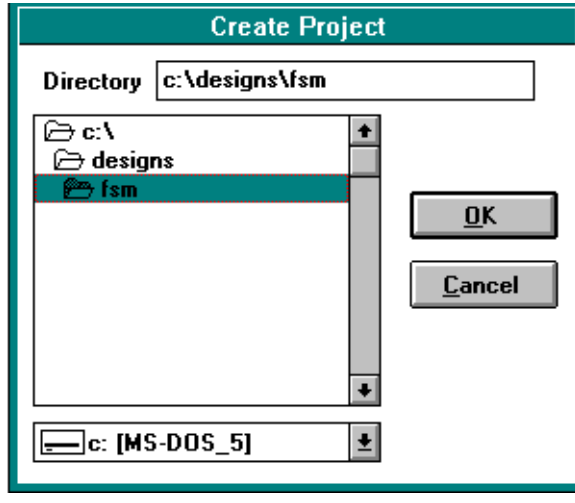


Figure 2-7 Create Project Dialog Box

7. In the Create Project dialog box, select or type in the path of the directory to which you copied the FSM tutorial files, for example:

   ```
   C:\DESIGNS\FSM
   ```

8. Select **OK**. The second dialog box closes.
9. Select the project you just created if it is not selected already, then click on the **Select** button.
10. Select the **Exit** button to close the Project Manager.
11. After you close the project manager, the dialog box shown in Figure 2-9 prompts you to select a device family:

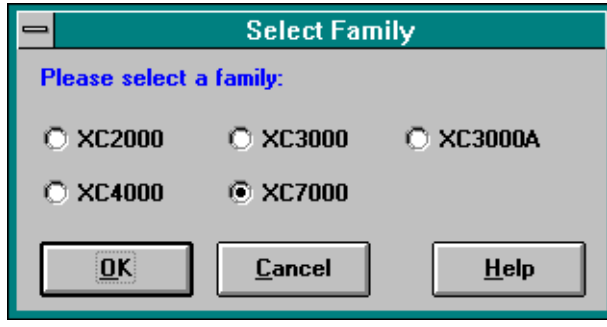


Figure 2-8 Select Family Dialog Box

12. Select **XC7000**, then select **OK**.
13. In the Design Entry dialog box, select **VHDL** as the Type of design.

14. Select the **fsm.vhd** file from the list. The file name appears in the Design Name field. Make sure the **Start Notepad** box is NOT checked. The Design Entry dialog box should now look like Figure 2-10:

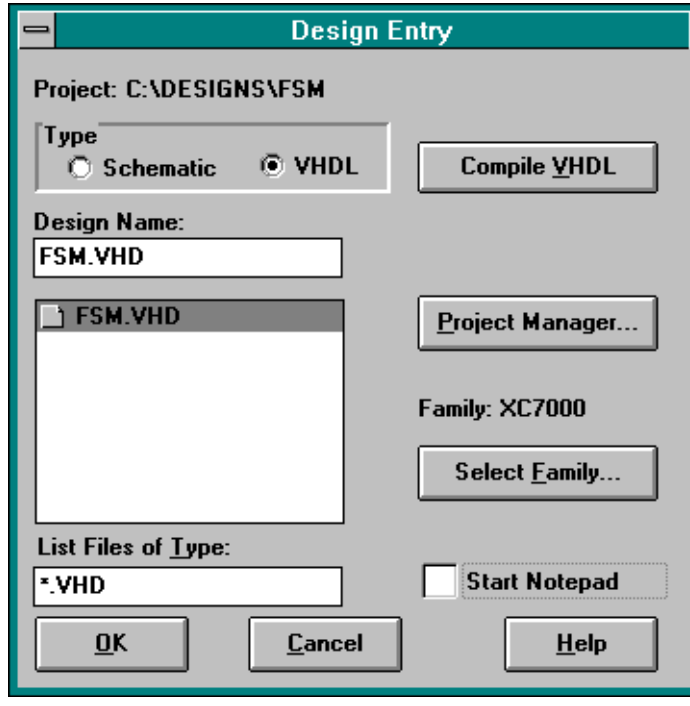


Figure 2-9 Design Entry Dialog Box After Changes

Note: If this were a multi-file design, for each lower-level file you would select the file from the list and then select the Compile VHDL button. You would then select the top-level file and select the OK button, just as in steps 14 and 15 here.

15. Select the **OK** button.

Design Synthesis and Schematic Generation

To create a schematic representation of your design, follow these steps:

1. Select the **Xilinx** button in PROflow. The Xilinx Implementation dialog box appears as in Figure 2-11:

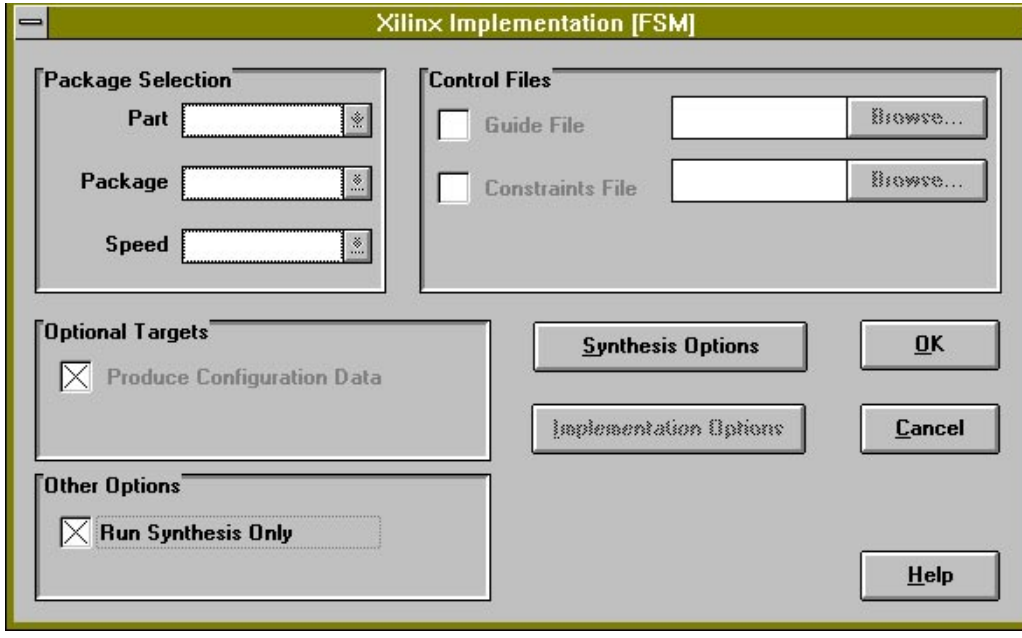


Figure 2-10 Xilinx Implementation Dialog Box

2. Check the **Run Synthesis Only** box. This synthesizes your design without running the fitter, so you can functionally simulate your design.

3. Click on the **Synthesis Options** button. The dialog box shown in Figure 2-12 appears:

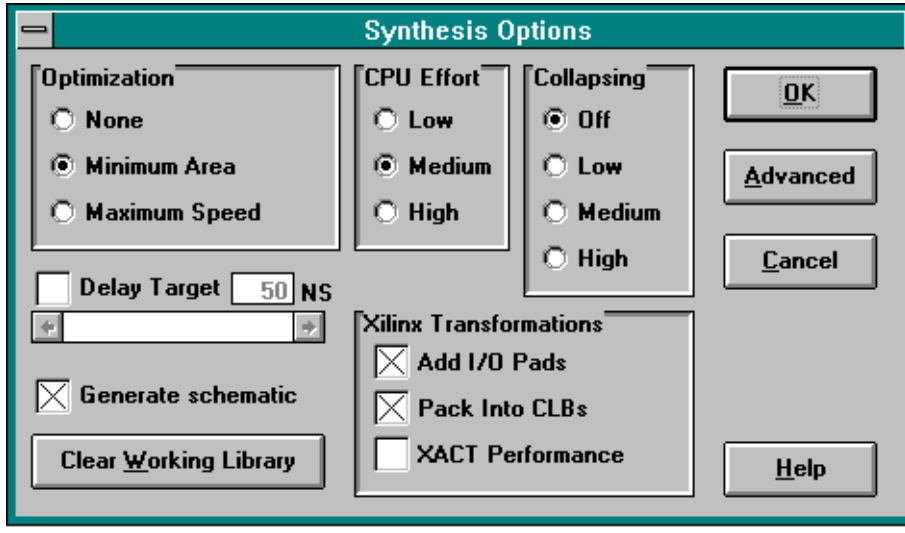


Figure 2-11 Synthesis Options Dialog Box

4. Make sure the **Generate Schematic** box is checked. This will cause a schematic representation of the VHDL design to be created.
5. Select **OK** to close the Synthesis Options dialog box.
6. Select **OK** to close the Xilinx Implementation dialog box and run the synthesis. When synthesis is complete, you see the following message in a prompt box:

```
Synthesis complete. View report?
```

7. You can select Yes or No. If you select **Yes**, you must close the report window after you have finished reading the report.

Functional Simulation

After you have synthesized your design and created a schematic, you can perform functional simulation.

1. Select the **PROcapture** button in PROflow. The Design Entry dialog box appears.

2. Select **Schematic** as the Type of design. You should see the fsm.1 file in the Design Name field. Make sure the Start PROcapture box is NOT checked. The dialog box should look like Figure 2-13 after these changes:

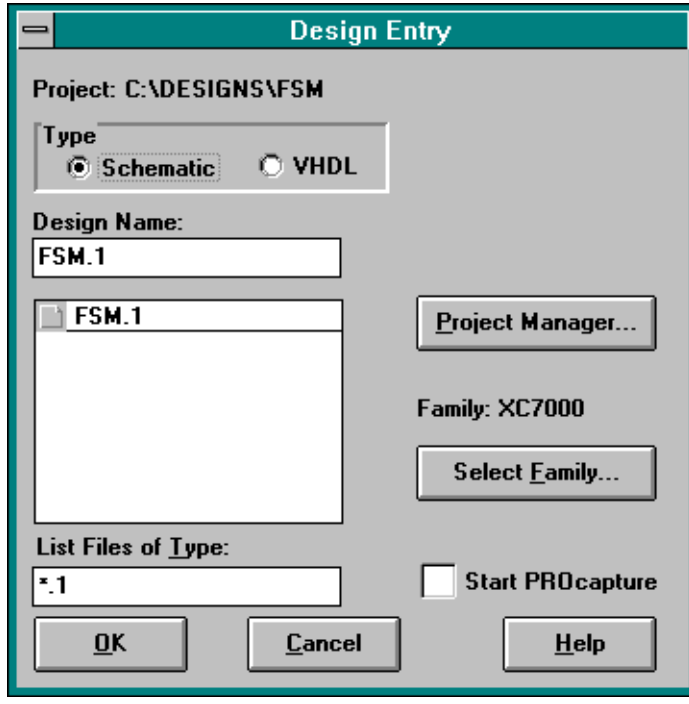


Figure 2-12 Design Entry Dialog Box After Schematic Changes

3. Select the **OK** button.

4. Select the **PROsim** button in PROflow. The Functional Simulation dialog box appears as in Figure 2-14:

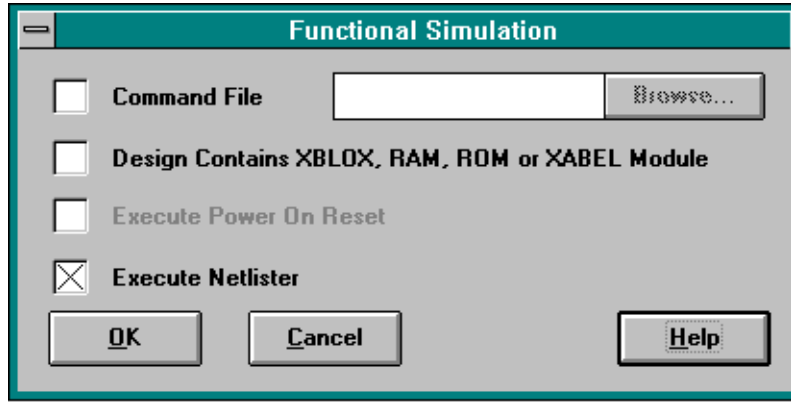


Figure 2-13 Functional Simulation Dialog Box

5. Select the **Command File** box and the **Browse** button. The Command Files browser appears as in Figure 2-15:

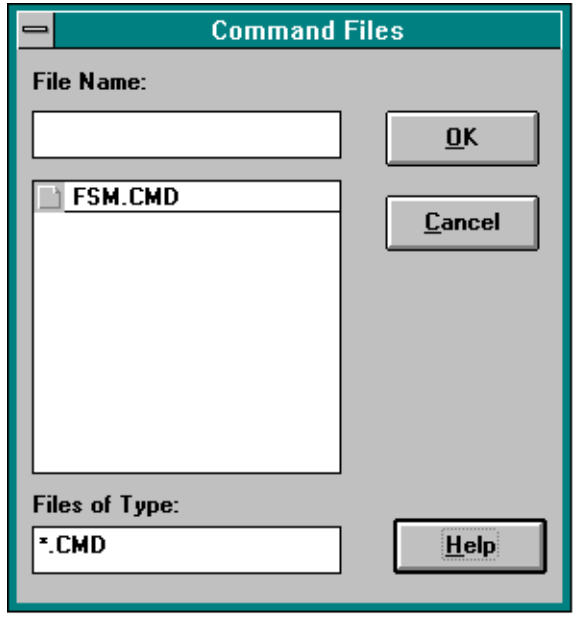


Figure 2-14 Command Files Browser

6. Select **fsm.cmd** from the list, then select **OK**. The browser closes.

   The fsm.cmd file tells PROsim how to simulate the fsm design.The file's contents are shown here:

```
wave fsm.wfm PRLD CLK RESET DATAIN PMATCH
vector fsm PRLD CLK RESET DATAIN PMATCH
watch PRLD CLK RESET DATAIN PMATCH
break fsm ? do (print > fsm.out)

stepsize 250
clock CLK 0 1

wfm PRLD @0=1 +
@0500=0

wfm RESET @0=0 +
@0500=1 +
@1000=0
```

```
wfm DATAIN @0=1 +
@4000=0 +
@5500=1 +
@6500=0 +
@7000=1 +
@8000=0 +
@8500=1 +
@10000=0 +
@11000=1

sim 13000
```

7. Select **OK** in the Functional Simulation dialog box. After the simulation netlist is processed, the vsm.log file is displayed. Close this file window.
8. The PROsim window appears, and the netlist is read. Close the PROsim window when processing is complete (simulation is stopped at 1401 ns).
9. Select the **PROwave** button in PROflow. The PROwave Setup dialog box appears as in Figure 2-16:

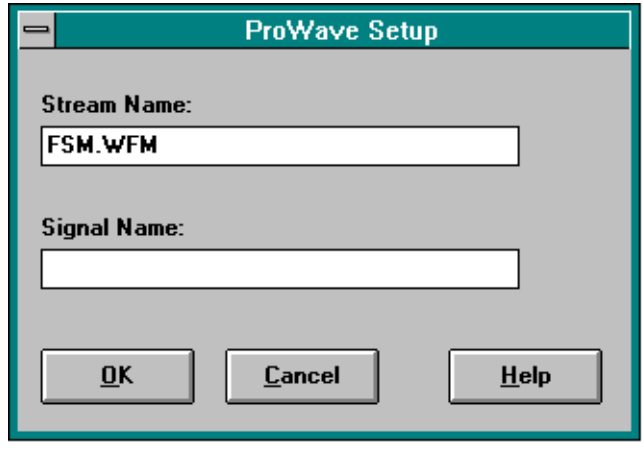


Figure 2-15 PROwave Setup Dialog Box

10. Select **OK** in the PROwave Setup dialog box.
11. The PROwave window appears, along with the **File → Open** browser. Select **fsm.wfm** from the list and select **OK**.

The waveforms appear as in Figure 2-17:

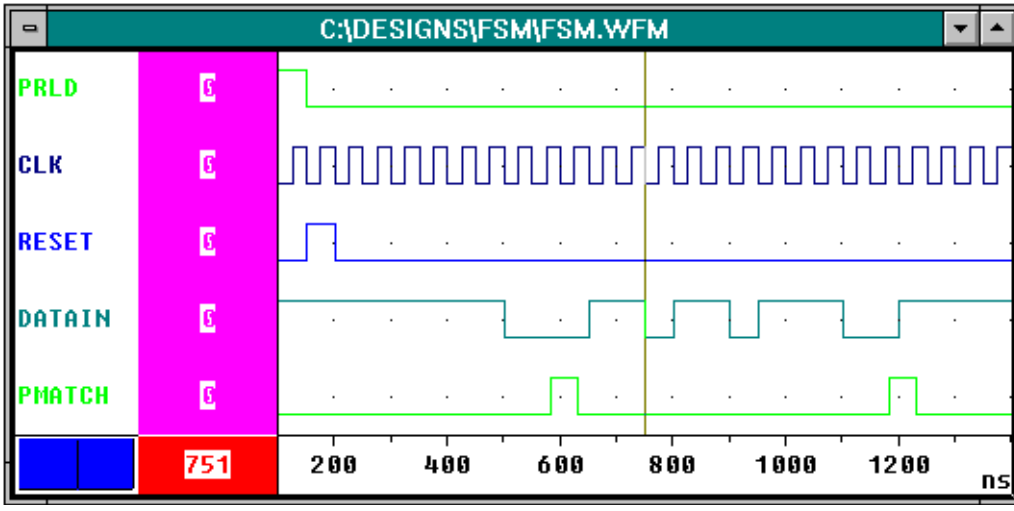


Figure 2-16 Simulation Waveforms for the FSM Design

12. When you are finished viewing, close the PROwave window.

Design Fitting and Device Programming

The XEPLD fitter translates the schematic representation of your FSM design into a physical device layout. To invoke the fitter, follow these steps:

1. Click on the **Xilinx Implementation** button in PROflow. The dialog box shown in Figure 2-18 appears:

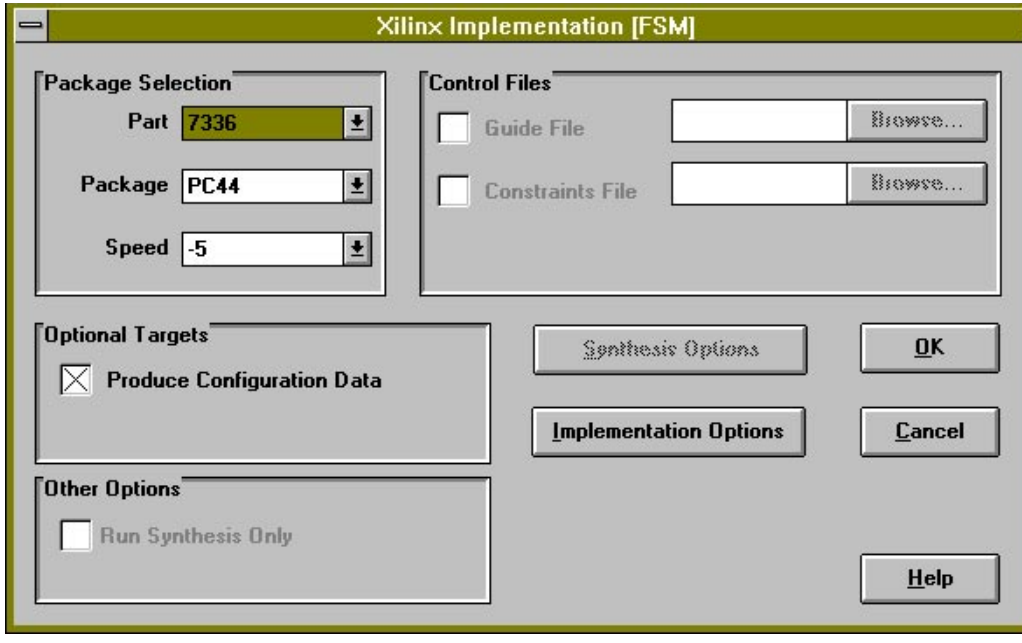


Figure 2-17 Xilinx Implementation Dialog Box

2. Select **7336** in the Part field. The default package is PC44, and the default speed is 5.
3. Make sure the Run Synthesis Only box is NOT checked.

4. Click on the **Implementation Options** button. The dialog box shown in Figure 2-19 appears:

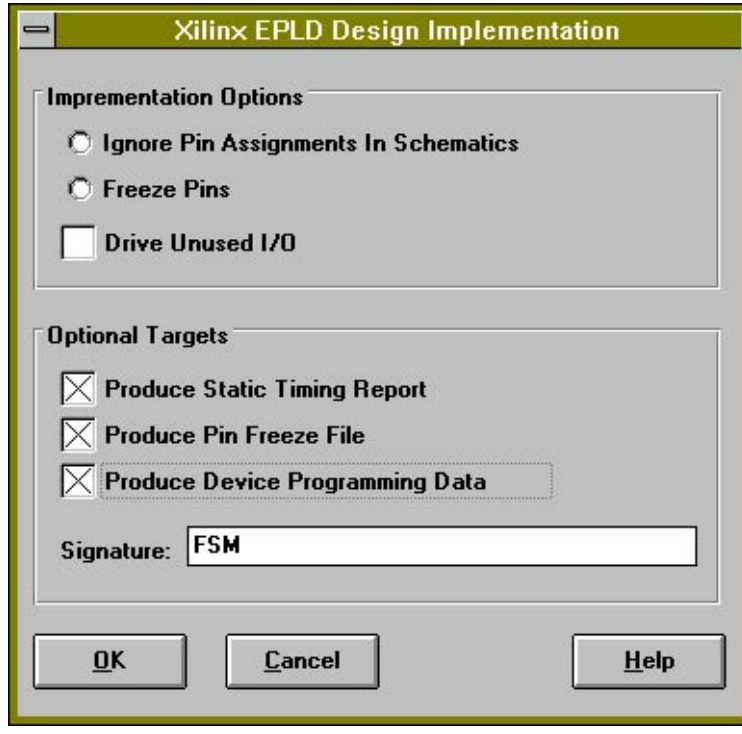


Figure 2-18 Implementation Options Dialog Box

5. This dialog box offers the following implementation options. Do not check any of these first three options.
   - Ignore Pin Assignments in Schematics — Does not apply to PROsynthesis designs.
   - Freeze Pins — If checked, tells the fitter to use the pin freeze file created the last time the fitter was run on the design, thereby preserving the pinout (default is unchecked).
   - Drive Unused I/O — If checked, drives unused I/O pins to VCC or GND to prevent noise in the chip (default is unchecked).

6. In addition to creating a design database, fsm.vmh, which the fitter does automatically, you can also tell the fitter to perform the following functions. Check all of these last three boxes.

   - Produce Static Timing Report — When checked, creates a report showing the calculated worst case timing for your design, fsm.tim (default is unchecked).

     See the Fitter Reports appendix for more information on how to interpret the Static Timing Report.

   - Produce Pin Freeze File — When checked, creates a file containing the pin assignments, fsm.vmf (default is unchecked).

   - Produce Device Programming Data — When checked, creates an Intel HEX file for programming the device, fsm.prg (default is unchecked). The device programming file contains all necessary information for programming EPLDs using a device programmer.

     A Signature field also appears when you check this box. Use the default signature, which is the design name.

7. Select the **OK** button to close the Implementation Options window, then select **OK** to run the fitter.

   You will see a series of windows pop up and display messages from the fitter as your design is processed. Each window stays up after processing is complete so you can read the final messages. The window names are as follows: XEMAKE, TIMERPT, PINSAVE, and MAKEPRG.

8. After each window shows that processing is complete, close the window so the next process can begin. The Xilinx Implementation window closes when all processing is complete.

Timing Simulation

1. Select the second **PROsim** button in PROflow. The Timing Simulation dialog box appears as in Figure 2-20:

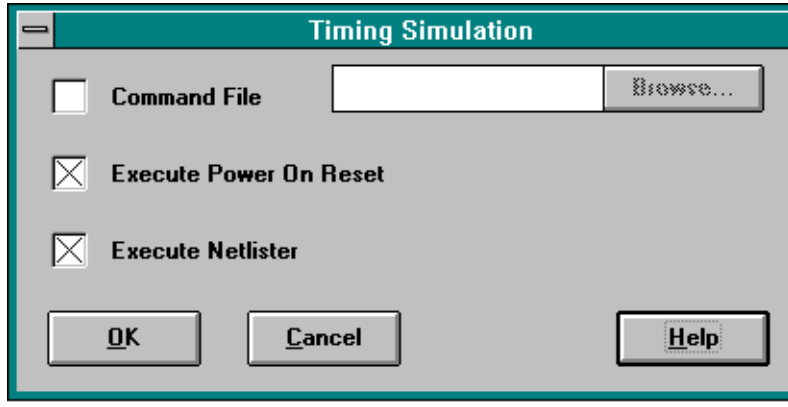


Figure 2-19 Timing Simulation Dialog Box

2. Select the **Command File** box and the **Browse** button. The Command Files browser appears as in Figure 2-21:

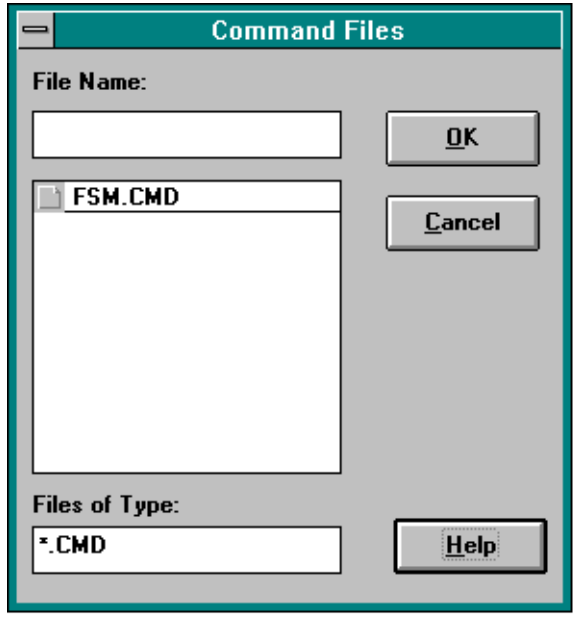


Figure 2-20 Command Files Browser

3. Select **fsm.cmd** from the list, then select **OK**. The browser closes. The fsm.cmd file tells PROsim how to simulate the fsm design.
4. Select **OK** in the Timing Simulation dialog box. After the simulation netlist is created, the xsimmake.out file is displayed. Close this file window.
5. The PROsim window appears, and the netlist is read. Close the PROsim window when processing is complete (simulation is stopped at 1401 ns).

6. Select the **PROwave** button in PROflow. The PROwave Setup dialog box appears as in Figure 2-22:

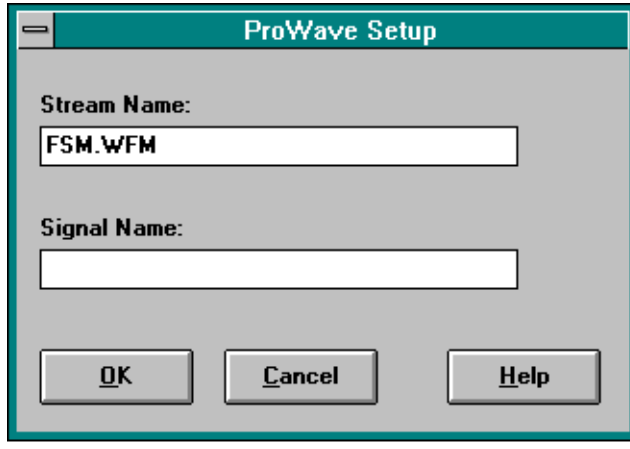


Figure 2-21 PROwave Setup Dialog Box

7. Select **OK** in the PROwave Setup dialog box.
8. The PROwave window appears, along with the **File → Open** browser. Select **fsm.wfm** from the list and select **OK**.

The waveforms appear as in Figure 2-23:

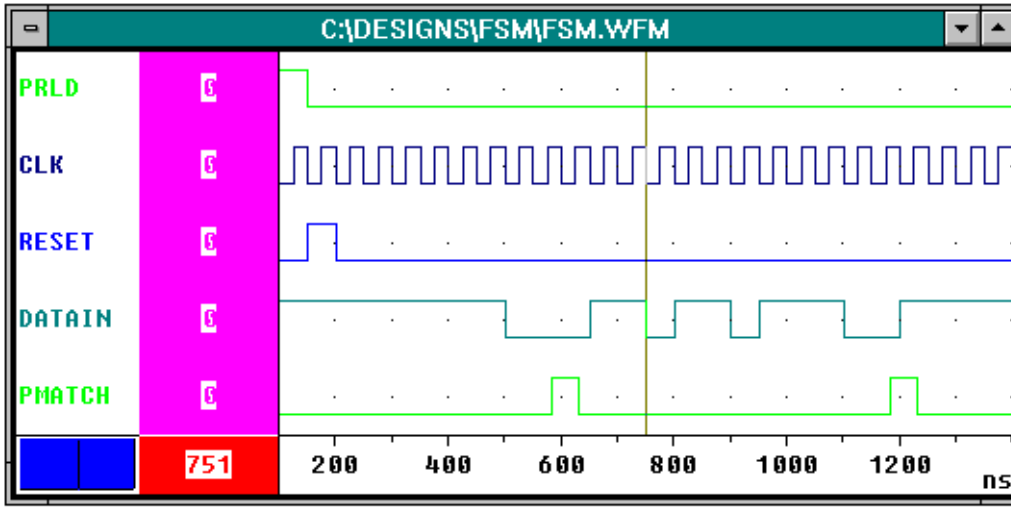


Figure 2-22 Simulation Waveforms for the FSM Design

9. When you are finished viewing, close the PROwave window, then close the PROflow window.

Chapter 3

Designing With EPLDs

This chapter discusses how to use design techniques, library components, and attributes to get the best performance from Xilinx EPLDs. For more information on library components, see the Library Component Specifications appendix. For more information on attributes, see the Attributes appendix.

VHDL Design File General Requirements

Design files for EPLD designs must reference the XC7000 library and must use IBUF and OBUF input and output buffers.

XC7000 Components Package

If you plan to instantiate any components from the XC7000 library, you will need to declare the Xilinx XC7000.components package in your design source file. It is generally a good idea to always declare this package in all EPLD designs.

To declare the XC7000.components package, insert the following two lines at the top of your VHDL source file:

```
library xc7000;
use xc7000.components.all;
```

I/O Buffers

All input and output ports in your design must pass through IBUF and OBUF type buffer components. EPLD device pins will only be applied where IBUF and OBUF type buffers have been specified.

You can specify I/O buffers either in the main design file or in a separate file, whichever is more convenient.

Defining I/Os in the Main Design

For each input or output port in your main design, instantiate an IBUF or OBUF component from the XC7000 library and pass the signal through it on the way into or out of your design. For bus ports, you'll need to pass each signal through an individual IBUF or OBUF component. For 3-state output ports, you can either instantiate an OBUFE component and present the output enable signal to it, or express the 3-state assignment behaviorally and pass the resulting signal through an OBUF component. For bidirectional I/O ports, instantiate both an IBUF and either an OBUF or OBUFE, connecting both to the same inout-type port in your design.

Defining I/Os Using a Top-Level File

Create your basic design using ViewLogic VHDL. In your design entity, define your input-only ("in") and output-only ("out" or "buffer") port signals as usual. However, do not define any bidirectional ("inout") ports in the basic design entity. Instead, define a separate input port signal ("in") and output port signal ("out") to represent the signals received from and transmitted to each physical I/O port. Each pair of input and output signals will later be connected to a pair of IBUF and OBUF components instantiated in the top-level design file to implement the desired bidirectional I/O port on the device.

Three-state output port signals, including those transmitted out to a bidirectional I/O, can be expressed in the basic design file as usual. Three-state outputs can be expressed either behaviorally or by instantiating a BUFE cell.

Creating a Top-Level Design

Your top-level design file defines the actual I/O ports to be implemented on the target device. The top-level design entity should define all the I/O port signals of your design as "in", "out" or "inout" (for bidirectional I/Os). Except for the inout ports, the port definitions in your top-level file should look similar to the port definition in your basic design file entity. Output ports defined as "buffer" in your basic design file can be declared as "out" in your top-level file because they are not re-used as inputs at this level of the hierarchy.

Within the architecture header of the top-level design file, you must declare the basic design entity as a component, and a set of intermediate signals used to connect the design instance to the I/O cell instances. In the architecture body, instantiate the basic design and connect the intermediate signals to it.

For each input signal, instantiate an IBUF cell (unless a special EPLD-specific input buffer is desired instead). For each output signal, instantiate an OBUF cell (unless a special EPLD-specific output buffer is desired instead). For each inout signal, instantiate both an IBUF and OBUF cell, and map their external ports to the same inout port signal.

The interconnections of port signals in the top-level design is illustrated in Figure 3-1.

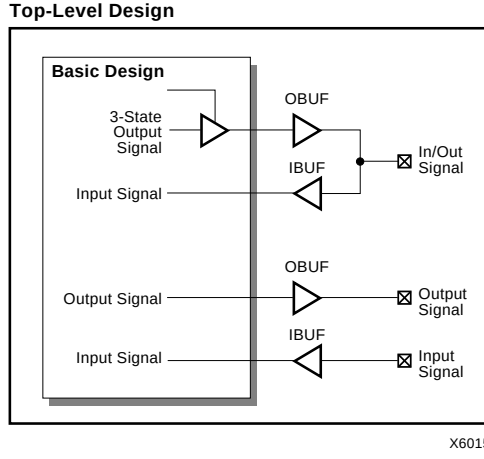


Figure 3-1 Defining Device I/O Cells in the Top-Level Design

Special-Purpose I/O Ports

As a minimum, you must instantiate IBUF and OBUF cells for all top-level input, output, and I/O ports. However, the Xilinx component library also includes special-purpose I/O buffer cells that you can use instead of IBUF and OBUF to allow you to explicitly instantiate specific I/O functions. You will want to explicitly assign special I/O buffer cells for the following reasons.

- *There are more clocks or OE signals in your design than there are FastClock or FOE pins available on the device.* The Xilinx fitter automatically assigns the most frequently used clock signals to FastClock pins and the most frequently used 3-state control inputs to FOE pins. You can force specific clocks onto the global FastClock pins by instantiating the BUFG cell. You can force specific output enable signals onto the global FOE pins by instantiating the BUFFOE cell.
- *You do not want some clocks, output enable signals, or registers to be optimized automatically.* You can globally inhibit optimization of these resources by instantiating the NO_FCLK, NO_FOE, and NO_IFD attribute cells. In this case you can manually assign selected clock or OE inputs to the global FastClock or FOE inputs by instantiating the BUFG or BUFFOE component. Instantiate the IFDX1 or ILD components to explicitly implement registers and latches in input pads.
- *You are generating global clock or FOE signals from within your design.* If you want to drive the global FastClock or FOE inputs from signals within your design, you must first drive those signals onto the corresponding device I/O pad through an output buffer (OBUF) and then back into the chip through either the BUFG component (for FastClocks) or through the BUFFOE component (for FOE inputs).

Selecting 3-State Control Sources

Xilinx EPLDs have dedicated high-speed routing that can be used for fast output enable signals (FOE). Any unused FOE routing is automatically assigned by the fitter to the most used output enable signals in your design (unless you turn off optimization by using the NO_FOE attribute).

To be eligible for optimization, an output enable signal must come directly from an input or I/O port and not be used for any other logic function.

Assigning Specific Fast Output Enable Signals

If you want to assign a specific output enable signal in your design to an FOE net, instantiate the BUFFOE input buffer to drive the Enable input of an OBUFEX1 component. The signal produced by the

BUFFOE component cannot be used by any other logic, including the OE input of ordinary OBUFE components.

Using Xilinx-Supplied Macros

The Xilinx library contains several VHDL macros of variable width (such as incrementors and comparators) that you can instantiate in your design.

To use these macros, do the following:

- Copy the macro from the examples\vwlogic\macro_7k library to your design directory.
- Edit the macro and change the default value of the generic parameter WIDTH to your required width.
- Edit the macro and change its entity name to a new unique name (for example, append the width value to the original macro name). There are three places in the macro definition where this change needs to be made.
- Declare and instantiate the macro in your basic design.

Using Registers And Latches

The Xilinx EPLD architecture allows you to implement both registers and latches within function block macrocells and within input pads. This section shows you how to assign logic to specific registers and latches, and how to control their initial states after power is applied.

The Xilinx fitter uses input pad registers and latches to implement functions whenever possible to reduce the device macrocell resource requirements. Input pad registers also have a shorter setup time requirement than macrocell registers. Register functions using any control inputs, such as clear, preset, or clock enable, will only be implemented in macrocell registers; only simple D-type flip-flops can be optimized into input pads.

To be eligible for optimization into an input pad, a register's D and C inputs must come directly from input or I/O ports. The C (clock) input signal must be used only for register clocking, and the D (data) input signal must not be used for any other input.

Note: You can prevent the fitter from automatically assigning any registered or latched functions to the input pads by using the NO_IFD global attribute cell in your source design, as described in the Attributes appendix.

Using Input Pad Registers

If you want to assign a specific register in your design to an input pad, instantiate the IFDX1 component. The Clock input must be driven by a BUFG component (global FastClk™), and the Clock Enable input (if used) must be driven by a BUFCE component (Global Clock Enable). Except for signals declared as FastInputs, the D input signal must not be used for any other input.

Using Macrocell Registers

Inferred registered functions will be placed either into macrocells or input pads at the discretion of the fitter (unless register optimization is turned off).

The techniques used to infer registers in EPLD designs are no different than for any other ViewSynthesis design. For example, the following behavioral VHDL process implements a D-type flip-flop with asynchronous clear and clock-enable:

```
process begin
     wait until (prising (C) or (clr = '1'));
     if (clr = '1') then
        Q <= '0';
     else
        if (CE = '1') then
           Q <= D;
        end if;
     end if;
end process;
```

You can also instantiate the FDCP, FDPC, or FDCPE register components. If none of the control inputs are used, the software will attempt to optimize these registers into input pads, provided optimization is enabled.

Using Input Pad Latches

If you want to assign a specific latch in your design to an input pad, instantiate the ILD component. The G input must be driven by a BUFG component (global FastClk). Except for signals declared as FastInputs, the D input signal must not be used for any other input.

Using Macrocell Latches

The EPLD architecture can implement simple transparent latches using the clear and preset product terms of macrocell flip-flops. However, any other logic functions adjacent to such a latch typically cannot be implemented in the same macrocell.

Using Special Logic Functions

Some types of logic functions can be made faster and more efficient if they are constructed in a way that takes advantage of EPLD architectural features.

Binary Counters

You should instantiate all counters, incrementers and decrementers from the XC7000 library. Do not attempt to express counters larger than 4 bits behaviorally using the "+1" or "-1" operations. If you do, the resulting implementation will tend to consume extra resources and run more slowly.

State Machines

When you initially compile a state machine, use the binary encoding option (the default). If the logic complexity of a binary encoded state machine results in poor device resource utilization, you can try less fully encoded state assignments explicitly in your VHDL design. In general you can use a few more registers to represent state vectors to reduce the amount of combinational logic required for each state flip-flop.

In some cases, one-hot-encoding may produce satisfactory results, and it is convenient to specify to the synthesizer. Other schemes such as Gray coding do not help in EPLD designs because the EPLD architecture is primarily composed of D-type flip-flops.

Arithmetic Functions

All arithmetic components must be instantiated. Do not attempt to express adders or subtracters behaviorally using the "+" or "-" operators.

When creating registered arithmetic functions, instantiate the ACC component (Adder/Subtracter/Accumulator) or the ADSUR component (Adder/Subtracter with registered output) for best results. These components are scalable for any width and they are optimized for the Xilinx EPLD architecture.

Comparators

Magnitude comparators are expressed by instantiating the LT or LE components. They are implemented essentially the same as a subtracter, with the carry-out serving as the comparator output. 3-bit look-ahead logic at the low-order end of the comparator saves about 2 macrocells in the EPLD over the straight subtracter solution. The EPLD high-speed arithmetic carry chain is used for all magnitude comparators larger than 4 bits.

Equality comparators are implemented combinationally using XOR gates for each operand bit. The EPLD HDFB can accommodate up to an 8-bit equality compare in a single macrocell. You should use the EQ or NE components from the XC7000 library for all equality comparators. Comparators larger than 8 bits are implemented using multiple macrocells, each producing an 8-bit intermediate result. By using the EQ or NE components, the gate logic combining the macrocells' intermediate results is implemented in the UIM if possible (without extra delay).

Comparator outputs in high-speed applications are often pipelined before driving other logic or passing off-chip. By breaking larger comparators into 8-bit slices and pipelining each slice, gate logic combining the slices can still be implemented in the UIM (for on-chip logic).

In this example, a pipelined 16-bit comparator (with Boolean-type output Q) cannot be run at the maximum frequency of the EPLD because the logic preceding the register cannot fit a single macrocell:

```
U1: EQ_16 port map (AEQB, A(0 to 15), B(0 to 15));
process begin
```

```
    wait (prising (clock));
     Q0 <= AEQB;
end process;
```

In the following example, the 16-bit comparator is broken into two 8-bit registered comparators, joined by a UIM-based AND-gate. This solution can be clocked at the maximum frequency of the EPLD if it drives on-chip logic.

```
U1: EQ_8 port map (AEQB_LOW, A(0 to 7), B(0 to 7));
U1: EQ_8 port map (AEQB_HI, A(8 to 15), B(8 to 15));
process begin
    wait (prising (clock));
     Q_LOW <= AEQB_LOW;
     Q_HI <= AEQB_HI;
end process;
Q <= '1' when (Q_LOW = '1' and Q_HI = '1') else '0';
```

Targeting a Specific Device

Before fitting your design you must select a target device. You have three key questions to consider when selecting an EPLD:

- How many signal pins are required?
- How much Logic resources are required?
- How much performance (speed) is required?

The answers to these questions determine which device you will choose to contain your design.

Device selection can be an iterative process, as shown in the following steps:

1. Use the Xilinx EPLD data book to make a preliminary choice. This choice is usually based on the number of required signal pins because this is often the easiest question to answer. It is easiest to begin with the largest device (XC73144); this gives you the best chance for a successful fit. Otherwise, you can get a very rough estimate of the number of required macrocells as follows:

   [(the number of output ports)
   + (the number of flip-flops not directly driving output ports)]
   + [20%]

2. Run the fitter on your design using the selected device. After fitting, the Resource Report indicates how much device resources were required. This will help you determine the best device size. If your design does not fit you will need to choose a larger device or partition your design among multiple devices. If you have unused logic resources, you may want to try a smaller device.
3. Once an optimal device size has been determined, you can create a Static Timing Report that will indicate the calculated timing of your design based on the device layout. You can also simulate the timing of your design using the simulator. This timing information will help you select the optimal target device speed.

The EPLD Architecture appendix shows you a device selection chart. The Library Component Specifications appendix shows you which library components can be used with specific target devices. See the device data sheets for more information.

Controlling Design Performance

Devices in the Xilinx EPLD family include Fast Function Blocks (FFBs) and/or High Density Function Blocks (HDFBs). Fast Function Blocks provide the shortest delay paths while High Density Function Blocks provide the most logic resources. EPLDs also contain special high speed routing for clocks, output enable signals, clock enable signals, and logic inputs to FFBs.

You can control your design performance by using attributes to assign specific signals in your design to the appropriate physical EPLD resources.

Using High-Speed Clocks

Xilinx EPLDs have dedicated high-speed (FastCLK) routing that can be used for global clock signals. Any unused FastCLK routing is automatically assigned by the fitter to the most used clock signals (if eligible) in your design (unless you turn off optimization). To be eligible for FastCLK optimization, an input port signal must be used only for register clocking using the positive clock edge.

Note: EPLD Fast Function Blocks, input pad registers, and input pad latches must use FastCLK routing; they cannot use normal signal routing for clocks.

Assigning Specific High-Speed Clocks

If you want to assign a specific clock in your design to a FastCLK net, instantiate the BUFG buffer cell in your design.

Selecting EPLD Function Block Types

By assigning logic signals to specific EPLD Function Block resources, you can control the performance of logic paths in your design.

Specifying High Speed Paths

To assign a logical signal to a Fast Function Block (shortest delay paths), instantiate the F attribute cell in your source design and pass the intended FFB output signal through it as follows:

```
U1: F port map (signal_out, signal_in);
```

This causes the logic function which produces *signal_in* to be implemented in a FFB.

Specifying High Density Paths

To assign a logic signal to a High Density Function Block (normal delay paths), instantiate the H attribute cell in your source design and pass the intended HDFB output signal through it as follows:

```
U1: H port map (signal_out, signal_in);
```

This causes the logic function which produces *signal_in* to be implemented in a HDFB.

Using EPLD FastInputs

Some of the inputs to FFBs can be taken directly from input pins using a high-speed FastInput path which bypasses the Universal Interconnection Matrix. To assign input port signals to the EPLD FastInputs, instantiate the F attribute cell in your source design and pass the intended FastInput signal (from an IBUF buffer) through it as follows:

```
U1: F port map (signal_out, signal_in);
```

where *signal_in* comes from an IBUF component. This causes any logic functions using *signal_out* to take the input signal via the Fast

Input path, provided the function is implemented in a FFB.

Note: An input signal declared as a FastInput (applied to an F component) can also be used as the D input to an input pad register or latch (IFDX1 or ILD).

The Design Rule Checker

The Design Rule Checker (DRC) reads the design from the database and checks to see if any of the design rules have been violated. The following is a partial list of rules that are checked.

General Design Rule Violations

The DRC displays an error or warning if:

- Open (hanging) inputs to an instantiated XC7000 component are found. Unless otherwise specified, all unused inputs of a library component must be connected or tied to VCC or GND.
- Some library components can only be used for a particular target EPLD. The DRC will generate an error if you attempt to use these components for other EPLDs. Restrictions on the use of components can be found in the library data sheets.

Pad Component Design Rule Violations

The DRC displays an error if:

- A signal driving an output port (input to an OBUF) is driven by more than one source.
- The same signal drives more than one output port (OBUF).
- An input port signal (from an IBUF) is connected directly to an output port signal (to an OBUF).
- An output port (OBUF) is driven by VCC or GND.
- Multiple input buffers (IBUFs) are connected to the same port (the exception is when an IBUF and F component are used with an IFD, IFDX1, or ILD to receive a FastInput signal).

FastCLK, Clock Enable, Fast Output Enable Violations

The DRC displays an error if:

- There are more FastCLK (BUFG), CE (BUFCE), or FOE (BUFFOE) ports in the design than the target EPLD can support.
- A FastCLK, CE, or FOE signal drives a component pin that is not a clock, CE, or FOE input.

Using PROflow

This chapter shows you how to create a project directory, synthesize your design, fit your design, verify design timing, create a device programming file, and save your pinouts for later design iterations, all using the PROflow software.

You must have a complete VHDL design before compiling and fitting. You can run the compiler and fitter automatically one after the other, or you can run each separately.

Fitter Overview

XEPLD is the Xilinx EPLD implementation software (fitter). XEPLD uses the schematic design produced by PROsynthesis (after compiling) to create a physical layout for a target EPLD. XEPLD performs the following functions:

- Converts a Viewlogic schematic into netlist form (XNF).
- Reads the netlist file (sch*design_name*.1) and reports any rule violations to the error log file (*design_name*.err).
- Minimizes the combinational logic of your design so that it requires the least number of product term resources.
- Optimizes, partitions, and maps your design to fit within the architecture of the target device.
- Creates a pin-save file (optional) that is used to lock signal names to device pins, allowing you to keep the device pinouts during subsequent design iterations.
- Creates a Static Timing Report that shows the calculated worst-case timing for all signal paths in your design.

- Creates a timing simulation file that can be used by the PROsim simulator (*design_name*.vsm).
- Creates a device programming file (*design_name*.prg).
- Creates detailed reports that show you information such as the type and quantity of device resources used and device pinouts.

Creating a Project Directory

To create and configure a project directory, follow these steps:

1. Open the Xilinx flow manager, PROflow, by double clicking on the Psfm icon. The Xilinx PROflow window appears.
2. Select the **PROcapture** button in PROflow. The Design Entry dialog box appears.
3. Click on the **Project Manager** button. The PRO Series Project Manager dialog box appears.
4. Select the **Create** button to open the Create Project dialog box.
5. In the Create Project dialog box, select or type in the path of the project directory you wish to create. If the directory does not exist, it is created automatically.
6. Select **OK**. The Create Project dialog box closes.
7. Select the project you just created if it is not selected already, then click on the **Select** button.
8. Select the **Exit** button to close the Project Manager.
9. After you close the project manager, the Select Family dialog box prompts you to select a device family. Select **XC7000** if it is not selected already, then select **OK**.
10. In the Design Entry dialog box, select **VHDL** as the Type of design.
11. Select the design file from the list. The file name appears in the Design Name field.
12. You can perform design entry at this point by checking the **Open Notepad** box.
13. Select the **OK** button to close the Design Entry window.

Compiler Operation

To synthesize and create a schematic representation of your design, follow these steps.

1. Select the **PROcapture** button in PROflow. The Design Entry dialog box appears.
2. If your design has multiple files, for each lower-level file select the file name from the list and then select the **Quick Compile** button.
3. Select the top-level design file you want to process, then select **OK**. This closes the Design Entry window.

Note: If you have a purely behavioral VHDL design, you can perform functional simulation at this point by selecting the PROsim button in PROflow. For XC7000 designs, functional simulation is normally performed after you synthesize your design.

4. Select the **Xilinx** button in PROflow. The Xilinx Implementation dialog box appears.
5. Click on the **Synthesis Options** button. The Synthesis Options dialog box appears.
6. Make sure the **Generate Schematic** box is checked. This causes a schematic representation of the VHDL design to be created.
7. Select **OK** to close the Synthesis Options dialog box.
8. If you want to functionally simulate your design after synthesis, check the Run Synthesis Only box. This synthesizes your design without running the fitter. If you do not check this box, the fitter is automatically run after synthesis is complete.

   If you are running synthesis and the fitter, DO NOT check Run Synthesis Only, skip the rest of this section, and go on to the next section, "Fitter Operation."
9. If you are running synthesis only, select **OK** to close the Xilinx Implementation dialog box and run the synthesis. When synthesis is complete, you see the following message in a prompt box:

   ```
   Synthesis complete. View report?
   ```

   You can select Yes or No. If you select **Yes**, you must close the report window after you have finished reading the report.

10. You are now ready to perform functional simulation. The steps for functional simulation are the same as for timing simulation, which is described later in this chapter; the only exception is step 5, in which the functional simulation flow displays the vsm.log file instead of the xsimmake.out file.
11. Click on the **Xilinx Implementation** button in PROflow to reopen the Xilinx Implementation dialog box.

Fitter Operation

The following steps show you how to fit your schematic design into a target device using the Xilinx XEPLD fitter.

Fitting Your Design

The XEPLD fitter translates the schematic representation of your design into a physical device layout. You can run the fitter automatically after synthesis or separately from synthesis.

At this point the Xilinx Implementation dialog box should be open. To invoke the fitter, follow these steps:

1. Select the part, package, and speed grade. The valid Xilinx EPLD part numbers are listed in the Part field. When you choose a part, valid packages and speed grades for that part are listed in the Package and Speed fields, respectively.
2. Make sure the Run Synthesis Only box is NOT checked.
3. Click on the **Implementation Options** button. The Implementation Options dialog box appears.
4. This dialog box offers the following implementation options.
   - Ignore Pin Assignments in Schematics — Does not apply to PROsynthesis designs.
   - Freeze Pins — If checked, tells the fitter to use the pin freeze file created the last time the fitter was run on the design, thereby preserving the pinout (default is unchecked).
   - Drive Unused I/O — If checked, drives unused I/O pins to VCC or GND to prevent noise in the chip (default is unchecked).

5. In addition to creating a design database, *design_name*.vmh, which the fitter does automatically, you can also tell the fitter to perform the following functions.

   - Produce Static Timing Report — If checked, creates a report showing the calculated worst case timing for your design, *design_name*.tim (default is unchecked).

     See the Fitter Reports appendix for more information on how to interpret the Static Timing Report.

   - Produce Pin Freeze File — If checked, creates a file containing the pin assignments, *design_name*.vmf (default is unchecked). During any subsequent invocation of the fitter, you can select the Freeze Pins option to restore pinouts saved in the .vmh file.

     Making major changes to a fixed-pinout design may prevent the fitter from achieving a successful mapping if you use the Freeze Pins option. If the fitter fails, try running without this option to see if a fit is still possible. To fit a modified design into the selected device, you may need to delete some of the pin assignments in the .vmh file, allowing those pins to move to new locations.

     You cannot use a previously saved pinout if you change the size or package type of the target device. See the EPLD Data Book to determine which devices in the EPLD family have compatible pinouts across similar packages.

   - Produce Device Programming Data — If checked, creates an Intel HEX file for programming the device, *design_name*.prg (default is unchecked). The device programming file contains all necessary information for programming EPLDs using a device programmer.

     A Signature field also appears when you check this box. By default, the design name is used as the signature string. Only alphanumeric characters are allowed in the signature string; edit the string if necessary.

     EPLD programmers are available from Xilinx and from third-party developers. See your device programmer documentation for instructions on how to download the programming file.

6. Select the **OK** button to close the Implementation Options window, then select **OK** to run the fitter.

You will see a series of windows pop up and display messages from the fitter as your design is processed. Each window stays up after processing is complete so you can read the final messages. The window names are as follows: XEMAKE, TIMERPT, PINSAVE, and MAKEPRG.

7. After each window shows that processing is complete, close the window so the next process can begin. The Xilinx Implementation Options window closes when all processing is complete.

Fitter Reports

The fitter produces various reports:

- The Resource Report (*design_name*.res) indicates how well your design fits in the target device. This report shows the utilization of macrocells, Function Blocks and each type of device pin, and indicates the amount of remaining logic and I/O resources in the device. The Resource summary is also displayed near the end of the fitnet process.
- The Pinlist Report (*design_name*.pin) shows the signals assigned to each pin of the target device.
- The Partitioner Report (*design_name*.par) and the Mapping Report (*design_name*.map) show the detailed physical layout of your design within the EPLD.
- The Equation File (*design_name*.eqn) contains boolean equations representing the final implementation of your design after minimization and optimization, and is expressed in Xilinx PLUSASM syntax.
- The Static Timing Report (*design_name*.tim) shows the worst-case timing based on the physical implementation of your design.

Note: See the Fitter Reports appendix or the *Xilinx XEPLD Reference Guide* for more information on reports.

Design Timing Verification

There are two ways you can verify your design timing:

- *The Xilinx Static Timing Report* provides the calculated worst-case timing for all signal paths in your design.
- *PROsim* provides timing simulation based on the physical layout of the target device.

Creating The Xilinx Static Timing Report

The Static Timing Report is generated if you checked the Produce Static Timing Report box in the Implementation Options window.

The Static Timing Report is created and saved as *design_name*.tim. See the Fitter Reports appendix for a complete description of the Static Timing report.

Performing Timing Simulation

To perform timing simulation on the design, follow these steps:

1. Select the **PROcapture** button in PROflow. The Design Entry dialog box appears.
2. Select **Schematic** as the Type of design. You should see the schematic file in the Design Name field. Make sure the Start PROcapture box is NOT checked. Select the **OK** button.
3. Select the second **PROsim** button in PROflow. The Timing Simulation dialog box appears.
4. If you are using a simulation command file (.cmd), select the **Command File** box and the **Browse** button. The Command Files browser appears. Select the .cmd file from the list, then select **OK**.
5. Select **OK** in the Timing Simulation dialog box. After the design is processed, the xsimmake.out file is displayed. Close this window.
6. The PROsim window appears, and the design is processed further. Close the PROsim window when processing is complete.
7. If your simulation command file created a waveform file, you can use PROwave to view it. (If you know PROwave commands, you can create a waveform file after simulation is complete.) Select the **PROwave** button in PROflow.

8. The PROwave Setup dialog box appears. Select **OK** in the PROwave Setup dialog box.
9. The PROwave window appears, along with the **File → Open** browser. Select the waveform file (.wfm) from the list (if you have one) and select **OK**.
10. Close the PROwave window.

Appendix A

Using the Command Line

You can run synthesis and the XEPLD fitter software from the DOS command line. This appendix shows you how to run the tutorial design, FSM, from the DOS command line. You can either run all the commands in a batch file or run each command individually.

Using a Batch File

The files fsm.bat and fsm.src, which are supplied with the EPLD Viewlogic Synthesis tutorial, run all the DOS commands for the tutorial automatically and echo each command back to the screen. The fsm.bat file contains the Xilinx commands, and the fsm.src file contains the Viewlogic VHDL shell commands. The fsm.bat file invokes the fsm.src file, so all you need to do is type **`fsm`** at the DOS prompt.

Using Individual Commands

Table A-1 shows, in the proper order, the specific commands needed to run the FSM design. This table also includes generic command formats and an explanation of each command.

Table A-1 Commands to Run the Tutorial from DOS

Command to Run Tutorial	Generic Format	Command Explanation
`vhdldes`	**`vhdldes`**	Enters the Viewlogic VHDL design command shell, where you can invoke the VHDLDES commands that follow.

Command to Run Tutorial	Generic Format	Command Explanation
technology xc7000 (VHDLDES shell)	**technology** *family* (VHDLDES shell)	Specifies the device family.
clear_work_library (VHDLDES shell)	**clear_work_library** (VHDLDES shell)	Clears the working library so that previously run designs do not interfere with the processing of this design.
vhdl fsm.vhd (VHDLDES shell)	**vhdl** *filename* (VHDLDES shell)	Compiles the .vhd source file(s). If your design has more than one file, run this command on each file, the lowest-level files first and the top-level file last.
synthesize (VHDLDES shell)	**synthesize** (VHDLDES shell)	Produces a gate-level representation of the design.
wire (VHDLDES shell)	**wire** (VHDLDES shell)	Produces a netlist of the design in Viewlogic WIR format.
exit (VHDLDES shell)	**exit** (VHDLDES shell)	Exits the VHDLDES command shell.
wir2xnf fsm -p 7336-5PC44	**wir2xnf** *design* **-p** *device*	Creates an XNF formatted netlist file based on the WIR file.
xnfmerge fsm	**xnfmerge** *design*	Flattens macros so the netlist is ready for fitting.
fitnet fsm	**fitnet** *design* [**-p** *device*] [**-f** | **-i**] [**-m**]	Maps the design to the device you specified when using the wir2xnf command. The -p option can override the previous device specification.
timerpt fsm	**timerpt** *design*	Creates a static timing report (.tim).

Command to Run Tutorial	Generic Format	Command Explanation
`xsimmake -f vet fsm`	**`xsimmake -f vet`** *design*	Creates a Viewsim netlist (.vsm) file for timing simulation. The -f vet parameter stands for "Viewlogic EPLD Timing."
`pinsave fsm`	**`pinsave`** *design*	Creates a pin freeze (.vmf) file, which you can use to preserve the pinout in the next iteration of the design (use the fitnet -f option).
`makeprg fsm -s fsm`	**`makeprg`** *design* **`-s`** *signature*	Creates an Intel HEX file, which you use to program the device.

EPLD Architecture

The Xilinx EPLD family uses a simple PAL-like architecture to provide both high speed and high density in a variety of packages and configurations. Through a unique Dual-Block architecture, High Density Function Blocks (FBs) provide high speed and maximum logic density for implementing complex functions while Fast Function Blocks (FFBs) provide even higher speed for critical decoding and ultra-fast state machine applications. For more information see *The Programmable Logic Data Book*.

The EPLD architecture consists of multiple Function Blocks and I/O blocks interconnected by the UIM as shown in Figure B-1.

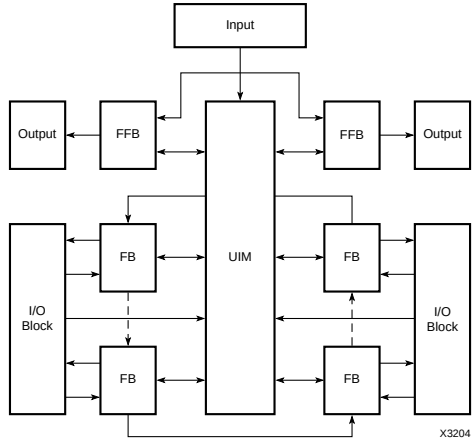


Figure B-1 EPLD Architecture Block Diagram

Device Selection

Table B-1 shows the Xilinx EPLD family, grouped by user application. Use this table to select the best target device for your design.

Package options and speed grades are always being updated. Please check the latest device data sheets for the most up to date information.

Table B-1 Device Selection Guide

Features	Fast Functions		Dual Block™ Arch. Fast and High Density			
	7318	7336	7354	7372	73108	73144
22VI0 Equiv.	2	4	6	8	12	16
Macrocells	18	36	54	72	108	144
FFBs	2	4	2	2	2	4
FBs	0	0	4	6	10	12
Flip-Flops	18	36	108	126	198	276
Fast Inputs Supported	12	12	12	12	12	12
Fast Clock Inputs	2	2	3	3	3	3
Fast Output Enab.	2	2	2	2	2	2
Fast Clock Enab.	0	0	2	2	2	2
Pin-to-Pin delay (ns.)	5	5	7	7	7	7
Clock Freq. (Mhz)	167	167	100	100	80	100
Signal Pins (max)	38	38	58	74	120	156
Speed Grades	-5 -7	-5 -7 -10 -12 -15	-7 -10 -12 -15	-7 -10 -12 -15	-7 -10 -12 -15	-7 -10 -12 -15

Device Packaging Options	Features	Fast Functions		Dual Block™ Arch. Fast and High Density			
		7318	7336	7354	7372	73108	73144
	44 Pin PLCC	X	X	X			
	44 Pin CLCC		X	X			
	44 Pin PQFP	X	X				
	68 Pin PLCC			X	X		
	68 Pin CLCC			X	X		
	84 Pin PLCC				X	X	
	84 Pin CLCC				X	X	
	100 Pin PQFP					X	
	144 Pin PGA					X	
	160 Pin PQFP					X	X
	225 Pin BGA					X	X

The Universal Interconnect Matrix

The Universal Interconnect Matrix™ (UIM) functions as an unrestricted crossbar switch.It guarantees complete interconnection of all internal functions and provides constant, short interconnect delays. It receives inputs from Macrocells, bidirectional I/O pins, and dedicated input pins and provides 21 outputs to each High-Density Function Block and 24 outputs to each Fast Function Block. Any UIM input can drive one or more UIM outputs with the interconnect delay remaining constant.

When multiple inputs are connected to the same output, this output produces the logical AND of the input signals. By choosing the appropriate signal inversions, this AND logic can also implement wide NAND, OR or NOR functions. This provides an additional level of logic with no additional delay.

A Macrocell feedback signal that is disabled by the output enable product term represents a High input to the UIM. Programming several such Macrocell outputs onto the same UIM output thus emulates a 3-state bus line. When one of the Macrocell outputs is enabled, the UIM output assumes its level.

High-Density Function Blocks

Each High Density Function Block contains nine Macrocells which can be configured for either registered or combinatorial logic. A detailed Function Block diagram is shown in Figure 4-2.

Each FB receives 21 signals and their complements from the UIM and an additional three inputs from the FastInput (FI) pins.

Note: The XC7272A FB architecture, including the ALU, is slightly different. See the data sheet for details.

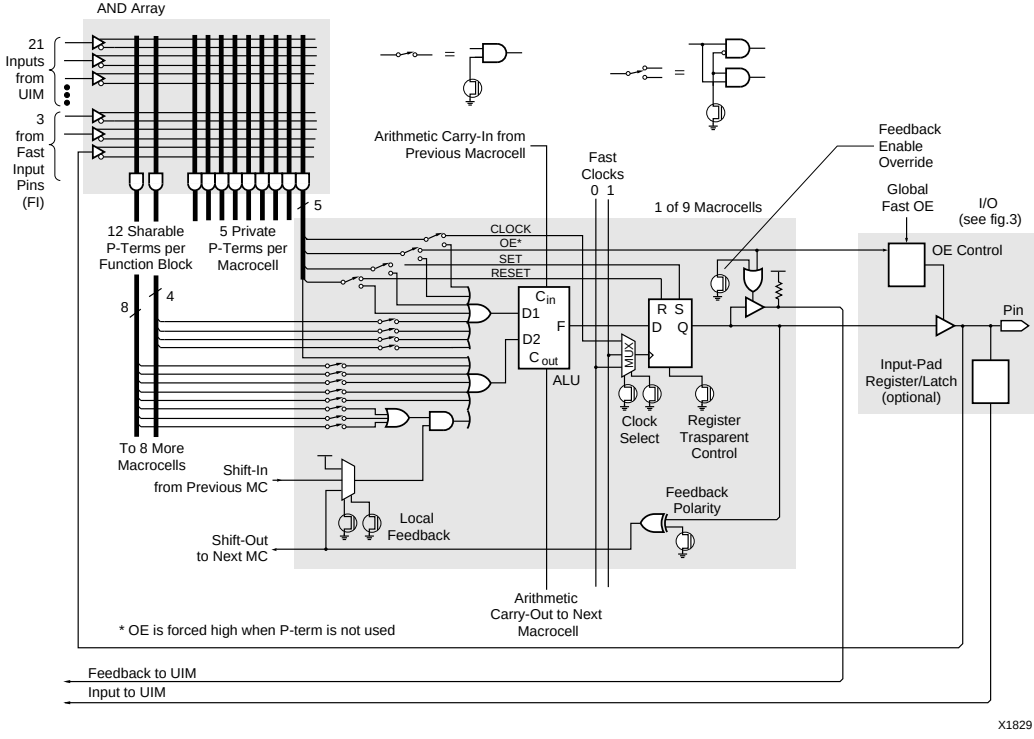


Figure B-2 High Density Function Block Schematic

Shared and Private Product Terms

Each Macrocell contains five private product terms that can be used as the primary inputs for combinatorial functions implemented in the Arithmetic Logic Unit (ALU), or as individual Reset, Set, Output-Enable, and Clock logic functions for the flip-flop. Each Function Block also provides an additional 12 shared product terms, which are uncommitted product terms available for any of the nine Macrocells within the FB.

Four private product terms can be ORed together with up to four shared product terms to drive the D1 input to the ALU. The D2 input is driven by the OR of the fifth private product term and up to eight of the remaining shared product terms. The shared product terms add no logic delay and each shared product term can be connected to one or all nine Macrocells in the Function Block.

Arithmetic Logic Unit

The versatility of each Macrocell is enhanced through additional gating and control functions available in the ALU. A detailed block diagram of the XC7300 and XC7236A™ ALU is shown in Figure 4-3.

The ALU has a logic mode and an arithmetic mode. In logic mode, the ALU functions as a 2-input function generator using a 4-bit look-up table that can be programmed to generate any Boolean function from the D1 and D2 inputs.

The function generator can OR its inputs, widening the OR function to a maximum of 17 inputs. It can AND them, which means that one sum-of-products can be used to mask the other. It can also XOR them, toggling the flip-flop or comparing the two sums of products. Either or both of the sum-of-product inputs to the ALU can be inverted, and either or both can be ignored. Therefore, the ALU can implement one additional layer of logic with no speed penalty.

In arithmetic mode, the ALU can generate the arithmetic sum or difference of the D1 and D2 inputs. Combined with the carry input from the next lower Macrocell, the ALU operates as a 1-bit full adder generating a carry output to the next higher Macrocell. The dedicated carry chain propagates between adjacent Macrocells and crosses the boundaries between Function Blocks providing very fast arithmetic operation with no additional resource requirements.

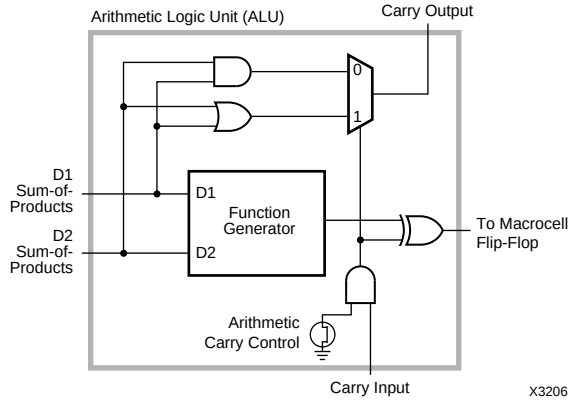


Figure B-3 ALU Schematic

Carry Lookahead (7300 Family Only)

Each Function Block provides a carry lookahead generator capable of anticipating the carry across all nine Macrocells. This reduces the ripple-carry delay of wide arithmetic functions such as add, subtract, and magnitude compare to that of the first nine bits, plus the carry lookahead delay of the higher-order Function Blocks.

Macrocell Flip-Flop

The ALU output drives the input of a programmable D-type flip-flop. The flip-flop is triggered by the rising edge of the clock input, and it can be configured as transparent, making the Q output identical to the D input, independent of the clock.

The Macrocell clock source is programmable and can be one of the private product terms or one of the global FastCLK™ signals. The FastCLK signals are distributed to every Macrocell flip-flop with short delay and minimal skew. The asynchronous Set and Reset product terms override the clocked operation. If both asynchronous inputs are active simultaneously, Reset overrides Set.

In addition to driving the chip output buffer, the Macrocell output is routed back to the UIM. One private product term can be configured to control the Output Enable of the output buffer and the feedback to the UIM. If it is configured to control UIM feedback, the Output

Enable product term forces the UIM feedback control input High when the Macrocell output is disabled.

Fast Function Blocks

Each Fast Function Block receives 24 signals and their complements from the UIM. The 24 inputs can be individually selected from the UIM, the 12 FastInput pins, or the nine Macrocell feedbacks from the FFB. The programmable AND array in each FFB generates 45 product terms to drive the nine Macrocells, which can be configured for registered or combinatorial logic. The FFB logic is shown in Figure 4-4.

Five product terms from the programmable AND array are allocated to each Macrocell. Four of these product terms are ORed together and drive the input of a programmable D-type flip-flop. The fifth product term drives the asynchronous active-high Set Input to the Macrocell flip-flop. The flip-flop can be configured as transparent to produce a combinatorial output.

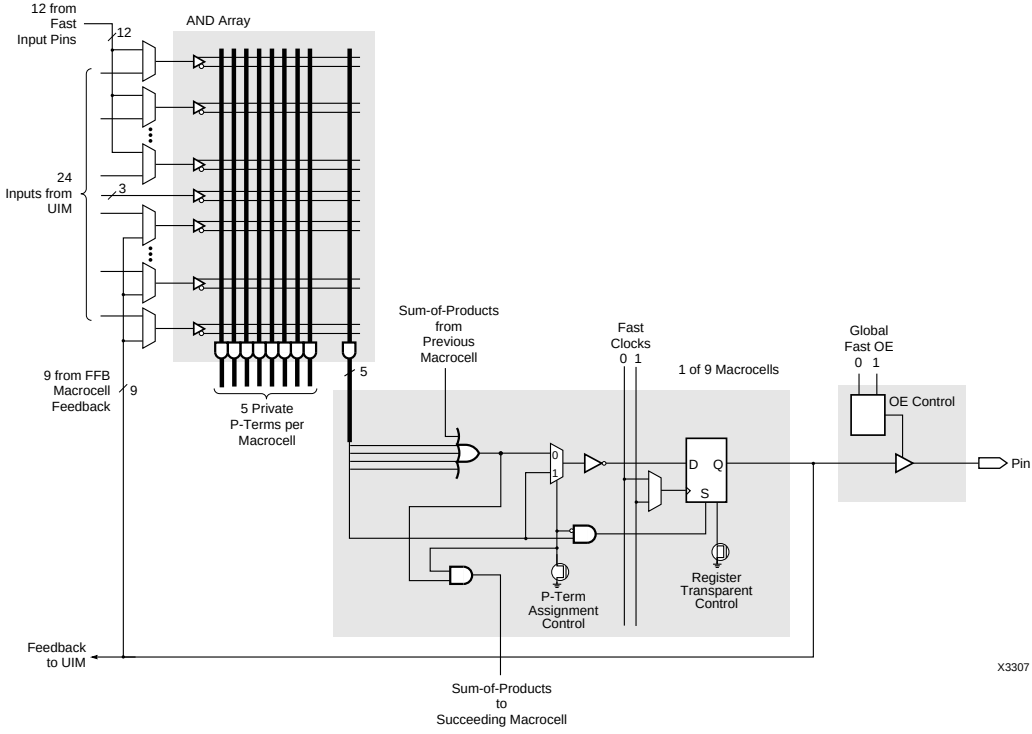


Figure B-4 Fast Function Block Schematic (for 7354, 7372, 73108, 73144)

The programmable clock source is one of two global FastCLK signals (FCLK0 or FCLK1) that are distributed with short delay and minimal skew over the entire chip.

The FFB Macrocells drive chip outputs directly through 3-state buffers. Each output buffer can be permanently enabled, permanently disabled, or controlled by one of two dedicated Fast Output Enable inputs. The Macrocell output is also routed back to the FFB and to the UIM.The XC7300 family provides a product term expansion feature that increases product-term flexibility without disabling Macrocell outputs. Product term expansion transfers product terms in increments of four product terms from one Macrocell to the next.

Product Term Expansion

Complex logic functions requiring up to 36 product terms can be implemented using this method. When product terms are assigned to adjacent Macrocells, the product term normally dedicated to the Set function becomes the D-input to the Macrocell register. Thus, the Macrocell is still usable while product terms are transferred to adjacent Macrocells. Figure B-5 illustrates product term expansion.

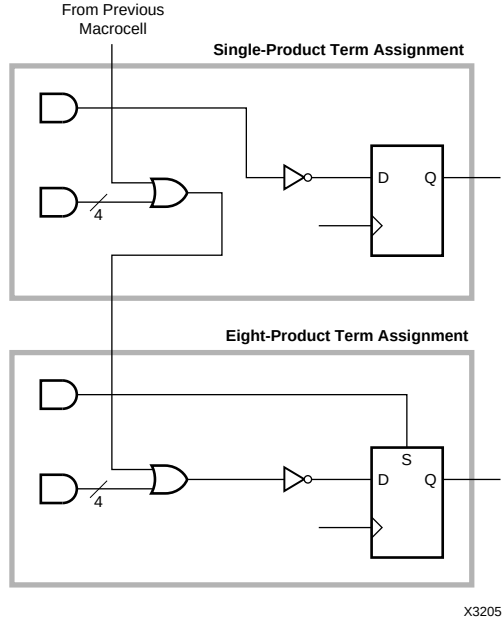


Figure B-5 FFB Product Term Expansion

XC7336 and XC7318 Fast Function Blocks

The Fast Function Blocks within the XC7318 and XC7336 are slightly different from those in the rest of the Xilinx EPLD family as shown in Figure 4-6.

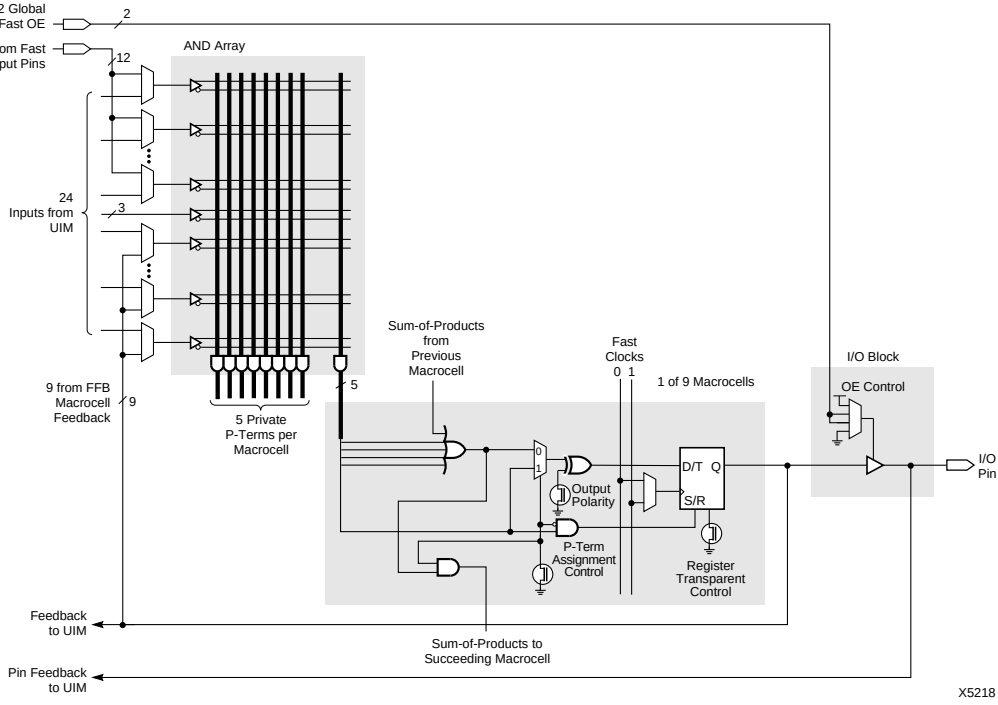


Figure B-6 Fast Function Block Schematic (for 7318, 7336)

Input/Output Blocks

I/O blocks provide 3-state outputs and registered, latched, or direct inputs. The I/O block registers can also implement logic equations and therefore decrease macrocell resource requirements.

Macrocells drive chip outputs directly through 3-state output buffers, each individually controlled by the Output Enable product term. An additional configuration option allows the output to be disabled permanently. Two dedicated Fast Output Enable inputs can also be configured to control any of the chip outputs instead of, or in conjunction with, the individual Output Enable product term. See Figure 4-7 for the I/O block schematic diagram.

Each signal input to the chip is connected to a programmable input structure that can be configured as direct, latched, or registered. The latch and flip-flop can use the FastCLK signals as latch enable or clock. Latches are transparent when FastCLK is High, and flip-flops clock on the rising edge of FastCLK.

The flip-flop includes an active-low clock enable, which when High, holds the present state of the flip-flop and inhibits response to the input signal. The clock enable source is one of two global Clock Enable signals ($\overline{\text{CKEN0}}$ and $\overline{\text{CKEN1}}$). An additional configuration option is polarity inversion for each input signal.

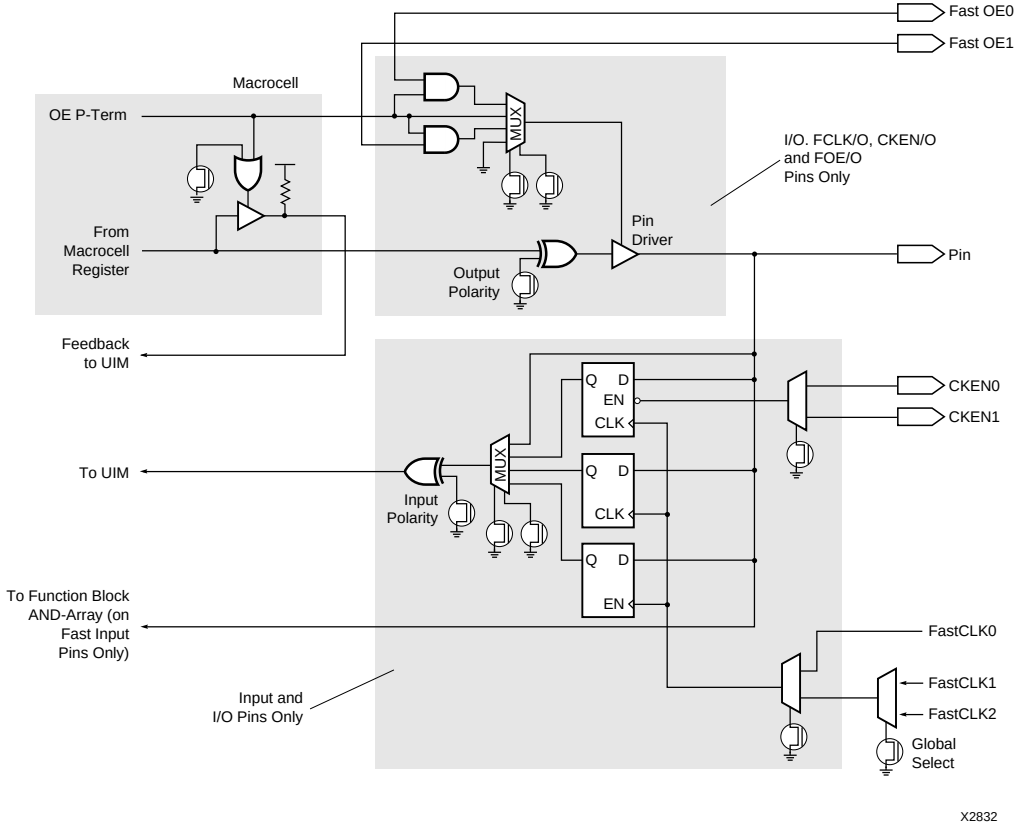


Figure B-7 Input/Output Block Schematic

The $\overline{\text{CKEN0}}$ and $\overline{\text{CKEN1}}$ inputs are only available in XC7300 family devices. Also, the programmable input polarity feature is not available in the XC7272A.

Library Component Specifications

This appendix describes each of the Xilinx library components, which are summarized in Table C-1.

Table C-1 Library Component Summary

			Used with These Devices				
Component Name	**Component Description**	**VHDL Macro**	**7272**	**7236**	**7318 7336**	**7354 7372 73108**	**73144**
ACC	Adder/Subtracter/Accumulator	X		X		X	X
ADD	Adder	X	X	X		X	X
ADSU	Adder/Subtracter	X	X	X		X	X
ADSUR	Adder/Subtracter with Registered Outputs	X		X		X	X
AND2-AND8	AND Gates		X	X	X	X	X
BUF	Buffer		X	X	X	X	X
BUFCE	Clock Enable Inp. Buff. for Input Pad Reg.					X	X
BUFE	3-State Buffer		X	X	X	X	X
BUFFOE	Fast Output Enable Input Buffer			X	X	X	X
BUFG	FastCLK Input Buffer		X	X	X	X	X
CBX1	Up/Down Counter with Asynchronous Clear	X	X	X	X	X	X
CBX2	Up/Down Counter with Asynchronous Reset	X	X	X	X	X	X
DEC	Decrementor	X	X	X	X	X	X
EQ	Equal-To Comparator	X	X	X	X	X	X
FDCP	Edge-Triggered D-Type Flip-Flop with Asynchronous Clear and Preset		X	X	X	X	X
FDCPE	Edge-Triggered D-Type Flip-Flop with Clock Enable, Async. Clear and Preset		X	X		X	X
FDPC	Edge-Triggered D-Type Flip-Flop with Asynchronous Clear and Preset		X	X	X	X	X
IBUF	Input Buffer		X	X	X	X	X
IFD	Input Pad Register		X	X		X	X
IFDX1	Input Pad Register with Clock Enable					X	X
ILD	Input Pad Latch		X	X		X	X
INC	Incrementer	X	X	X	X	X	X
INV	Inverter		X	X	X	X	X

			Used with These Devices				
Component Name	**Component Description**	**VHDL Macro**	**7272**	**7236**	**7318 7336**	**7354 7372 73108**	**73144**
LD	D-Type Latch		X	X		X	X
LE_TC	Less-Than-Or-Equal Comparator, 2's Comp.	X	X	X		X	X
LE_US	Less-Than-Or-Equal Comparator, Unsigned	X	X	X		X	X
LT_TC	Less-Than Comparator, 2's Complement	X	X	X		X	X
LT_US	Less-Than Comparator, Unsigned	X	X	X		X	X
NE	Not-Equal Comparator	X	X	X	X	X	X
OBUF	Output Buffer (Slow Slew Rate)		X	X	X	X	X
OBUF_F	Output Buffer (Fast Slew Rate)						X
OBUF_S	Output Buffer (Slow Slew Rate)						X
OBUFE	3-State Output Buffer (Slow Slew Rate)		X	X	X	X	X
OBUFE_F	3-State Output Buffer (Fast Slew Rate)						X
OBUFE_S	3-State Output Buffer (Slow Slew Rate)						X
OBUFEX1	3-State Output Buffer with FOE Enable (Slow Slew Rate)			X	X	X	X
OBUFEX1F	3-State Output Buffer with FOE Enable (Fast Slew Rate)						X
OBUFEX1S	3-State Output Buffer with FOE Enable (Slow Slew Rate)						X
OR2-OR8	OR Gates		X	X	X	X	X
SUBT	Subtracter	X	X	X		X	X
XOR2-XOR8	XOR Gates		X	X	X	X	X

ACC

ACC is an adder/subtracter/accumulator macro. The .vhd file for this macro, located in the *ds391_path*\examples\vwlogic\macro_7k directory, can be copied and edited to implement a specific number of bits by changing the default value of the generic "width" parameter.

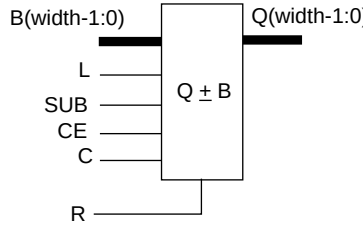
Component Instantiation

```
U1: ACC  port map (Q=>output, B=>in_operand,
    C=>clock, CE=>clock_en, R=>sync_reset,
    L=>load_en, SUB=>add_sub_ctl);
```

Truth Table and Logic Symbol

R	L	CE	C	SUB	Q*
1	X	X		X	0
0	0	0	X	X	Q
0	0	1		0	Q+B
0	0	1		1	Q-B
0	1	X		X	B

* The initial state is "0".



ADD

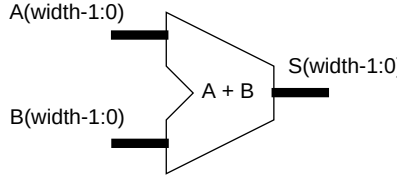
ADD is an adder macro. The .vhd file for this macro, located in the *ds391_path*\examples\vwlogic\macro_7k directory, can be copied and edited to implement a specific number of bits by changing the default value of the generic "width" parameter.

Component Instantiation

```
U1: ADD port map (S=>sum, A=>in1, B=>in2);
```

Truth Table and Logic Symbol

A	B	S
A	B	A+B



ADSU

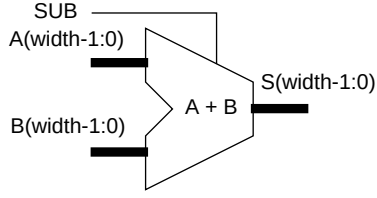
ADSU is an adder/subtracter macro. The .vhd file for this macro, located in the *ds391_path*\examples\vwlogic\macro_7k directory, can be copied and edited to implement a specific number of bits by changing the default value of the generic "width" parameter.

Component Instantiation

```
U1: ADSU port map (S=>output, A=>in1, B=>in2,
    SUB=>sub_ctl);
```

Truth Table and Logic Symbol

SUB	S
0	A+B
1	A-B



ADSUR

ADSUR is a registered adder/subtracter macro. The .vhd file for this macro, located in the *ds391_path*\examples\vwlogic\macro_7k directory, can be copied and edited to implement a specific number of bits by changing the default value of the generic "width" parameter.

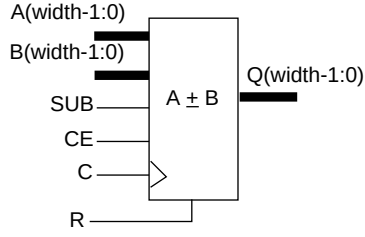
Component Instantiation

```
U1: ADSUR port map (Q=>output, A=>in1, B=>in2,
    C=>clock, CE=>clock_en, R=>sync_reset,
    SUB=>add_sub_ctl);
```

Truth Table and Logic Symbol

R	CE	C	SUB	Q*
1	X		X	0
0	0	X	X	Q
0	1		0	A+B
0	1		1	A-B

* The initial state is "0".



AND2 — AND8

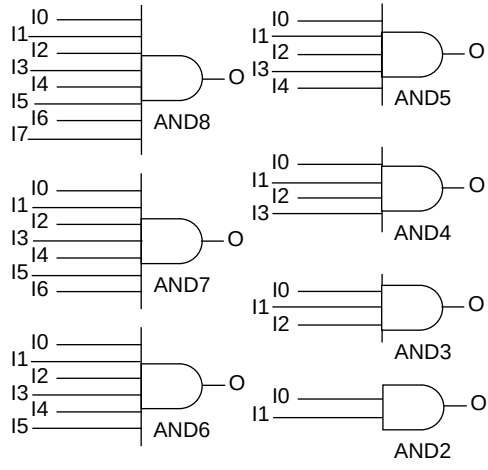
AND2 through AND8 are AND gates with 2 to 8 inputs.

Component Instantiation

```
U1: AND2 port map (O=>out,I1=>in2,I0=>in1);
```

Truth Table and Logic Symbol

I0	I1	O
0	0	0
0	1	0
1	0	0
1	1	1



BUF

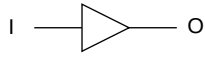
BUF is a buffer.

Component Instantiation

```
U1: BUF port map (O=>out_port, I=>in_port);
```

Truth Table and Logic Symbol

I	O
0	0
1	1



BUFCE

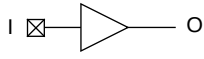
BUFCE is an input buffer used to drive the global CE signal (Chip Enable) for EPLD input pad registers. BUFCE may only be used to drive the CE input of IFDX1 components.

Component Instantiation

```
U1: BUFCE port map (O=>global_ce, I=>in_port);
```

Truth Table and Logic Symbol

I	O
0	0
1	1



BUFE

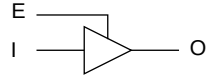
BUFE is a non-inverting 3-state buffer.

Component Instantiation

```
U1: BUFE port map (O=>ts_out, I=>inp, E=>enable);
```

Truth Table and Logic Symbol

I	E	O
X	0	Z
0	1	0
1	1	1



BUFFOE

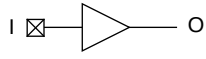
BUFFOE is a an input buffer used to drive the global FOE signal (Fast Output Enable). BUFFOE may only be used to drive the E input of OBUFEX1 components.

Component Instantiation

```
U1: BUFFOE port map (O=>global_foe, I=>in_port);
```

Truth Table and Logic Symbol

I	O
0	0
1	1



BUFG

BUFG is an input buffer used to drive the Global FastCLK signal.

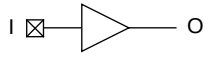
Note: BUFG can only drive register clock inputs (including IFDX1) and the G input of ILD components. It cannot drive the LD component.

Component Instantiation

```
U1: BUFG port map (O=>global_clk, I=>in_port);
```

Truth Table and Logic Symbol

I	O
0	0
1	1



CBX1

CBX1 is a loadable up/down counter macro with asynchronous clear. The .vhd file for this macro, located in the *ds391_path*\examples\vwlogic\macro_7k directory, can be copied and edited to implement a specific number of bits by changing the default value of the generic "width" parameter.

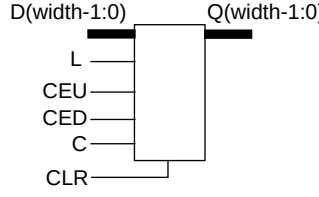
Component Instantiation

```
U1: CBX1 port map (Q=>output, TCU => all_ones,
    TCD => all_zeros, D=>load_data, C=>clock,
    CLR=>async_clr, L=>load_ctl,
    CEU=>count_up_ctl, CED=>count_down_ctl);
```

Truth Table and Logic Symbol

CLR	L	CEU	CED	C	TCU	TCD	Q*
1	X	X	X	X	0	1	0
0	1	X	X		D=111...	D=000...	D
0	0	0	0	X	Q=111...	Q=000...	Q
0	0	1	0		Q=111...	Q=000...	Q+1
0	0	0	1		Q=111...	Q=000...	Q-1
0	0	1	1		ILLEGAL CONDITION		

* The initial state is "0".



CBX2

CBX2 is a loadable up/down counter macro with synchronous reset. The .vhd file for this macro, located in the *ds391_path*\examples\vwlogic\macro_7k directory, can be copied and edited to implement a specific number of bits by changing the default value of the generic "width" parameter.

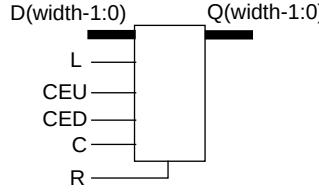
Component Instantiation

```
U1: CBX2 generic map (WIDTH => wordlength)
    port map (Q=>output, TCU => all_ones, TCD =>
    all_zeros, D=>load_data, C=>clock,
    R=>sync_reset, L=>load_ctl,
    CEU=>count_up_ctl, CED=>count_down_ctl);
```

Truth Table and Logic Symbol

R	L	CEU	CED	C	TCU	TCD	Q*
1	X	X	X		0	1	0
0	1	X	X		D=11...	D=00...	D
0	0	0	0	X	Q=11...	Q=00...	Q
0	0	1	0		Q=11...	Q=00...	Q+1
0	0	0	1		Q=11...	Q=00...	Q-1
0	0	1	1		ILLEGAL CONDITION		

* The initial state is "0".



DEC

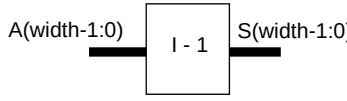
DEC is an decrementor macro. The .vhd file for this macro, located in the *ds391_path*\examples\vwlogic\macro_7k directory, can be copied and edited to implement a specific number of bits by changing the default value of the generic "width" parameter.

Component Instantiation

```
U1: DEC  port map (S=>sum, A=>in);
```

Truth Table and Logic Symbol

A	S
A	A-1



EQ

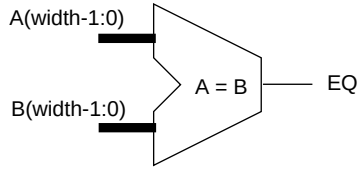
EQ is an equal-to comparator macro. The .vhd file for this macro, located in the *ds391_path*\examples\vwlogic\macro_7k directory, can be copied and edited to implement a specific number of bits by changing the default value of the generic "width" parameter.

Component Instantiation

```
U1: EQ port map (EQ=>comparison, A=>in1, B=>in2);
```

Truth Table and Logic Symbol

Condition	EQ
A<B	0
A=B	1
A>B	0



FDCP

FDPC is an edge-triggered D-type flip-flop with preset and clear.

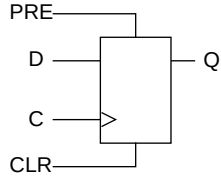
Component Instantiation

```
U1: FDCP port map (Q=>out, D=>data, C=>clock,
    CLR=>async_clr, PRE=>async_set);
```

Truth Table and Logic Symbol

CLR	PRE	C	Q*
1	X	X	0
0	1	X	1
0	0		D

* The initial state is “0”.



FDCPE

FDCPE is an edge-triggered D-type flip-flop with preset, clear, and clock enable.

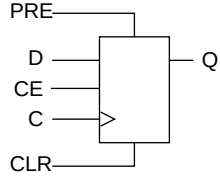
Component Instantiation

```
U1: FDCPE port map (Q=>out, D=>in, C=>clock,
    CE=>clock_enab, CLR=>async_clr,
    PRE=>async_set);
```

Truth Table and Logic Symbol

CLR	PRE	CE	C	Q*
1	X	X	X	0
0	1	X	X	1
0	0	0	X	Q
0	0	1		D

* The initial state is “0”.



FDPC

FDPC is an edge-triggered D-type flip-flop with preset and clear.

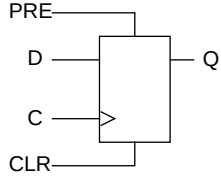
Component Instantiation

```
U1: FDPC port map (Q=>out, D=>data, C=>clock,
    CLR=>async_clr, PRE=>async_set);
```

Truth Table and Logic Symbol

CLR	PRE	C	Q*
X	1	X	1
1	0	X	0
0	0		D

* The initial state is “0”.



IBUF

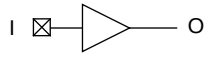
IBUF is an input buffer.

Component Instantiation

```
U1: IBUF port map (O=>received_signal,
    I=>in_port);
```

Truth Table and Logic Symbol

I	O
0	0
1	1



IFD

IFD is an edge-triggered D-type flip-flop. The C input must be driven by a BUFG component. IFD is only available for use in EPLD Input Blocks.

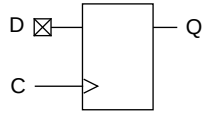
Component Instantiation

```
U1: IFD port map (Q=>output, D=>in_port,
    C=>global_clock);
```

Truth Table and Logic Symbol

C	Q*
X	Q
	D

* The initial state is “0”.



IFDX1

IFDX1 is an edge-triggered D-type flip-flop with active-low clock enable. The C input must be driven by a BUFG component. The CE input, if used, must be driven by a BUFCE component. IFDX1 is only available for use in EPLD Input Blocks.

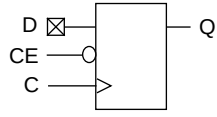
Component Instantiation

```
U1: IFDX1 port map (Q=>output, D=>in_port,
    C=>global_clock, CE=>global_ce);
```

Truth Table and Logic Symbol

CE	C	Q*
1	X	Q
0		D

* The initial state is “0”.



ILD

ILD is a D-type flip-flop available in the EPLD Input Block. The G input must be driven by a BUFG buffer.

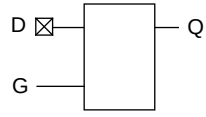
Component Instantiation

```
U1: ILD port map (Q=>output, D=>in_port,
    G=>global_clock);
```

Truth Table and Logic Symbol

G	Q*
0	Q
1	D

* The initial state is “0”.



INC

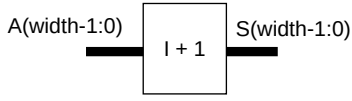
INC is an Incrementer macro. The .vhd file for this macro, located in the *ds391_path*\examples\vwlogic\macro_7k directory, can be copied and edited to implement a specific number of bits by changing the default value of the generic "width" parameter.

Component Instantiation

```
U1: ILD port map (Q=>output, D=>in_port,
    G=>global_clock);
```

Truth Table and Logic Symbol

A	S
A	A+1



INV

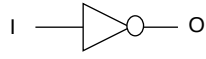
INV is an inverter.

Component Instantiation

```
U1: INV port map (O=>not_in1, I=>in1);
```

Truth Table and Logic Symbol

I	O
0	1
1	0



LD

LD is a D-type latch. The G input of LD cannot be driven by a BUFG buffer.

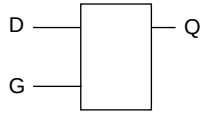
Component Instantiation

```
U1: LD port map (Q=>out, D=>data,
    G=>latch_enable);
```

Truth Table and Logic Symbol

G	Q*
0	Q
1	D

* The initial state is “0”.



LE_TC, LE_US

LE_US is an unsigned binary less-than-or-equal-to comparator macro. LE_TC is a two's complement less-than-or-equal-to comparator macro. The .vhd files for these macros, located in the *ds391_path*\examples\vwlogic\macro_7k directory, can be copied and edited to implement a specific number of bits by changing the default value of the generic "width" parameter.

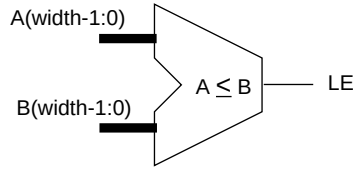
Component Instantiation

```
U1: LE_US port map (LE=>comparison, A=>in1,
    B=>in2);

U1: LE_TC port map (LE=>comparison, A=>in1,
    B=>in2);
```

Truth Table and Logic Symbol

Condition	LE
A<B	1
A=B	1
A>B	0



LT_TC, LT_US

LT_US is an unsigned binary less-than comparator macro. LT_TC is a two's complement less-than comparator macro. The .vhd files for these macros, located in the *ds391_path*\examples\vwlogic\macro_7k directory, can be copied and edited to implement a specific number of bits by changing the default value of the generic "width" parameter.

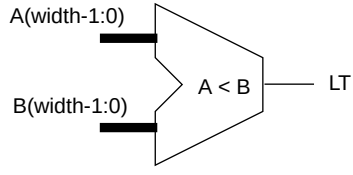
Component Instantiation

```
U1: LT_US port map (LT=>comparison, A=>in1,
    B=>in2);

U1: LT_TC port map (LT=>comparison, A=>in1,
    B=>in2);
```

Truth Table and Logic Symbol

Condition	LT
A<B	1
A=B	0
A>B	0



NE

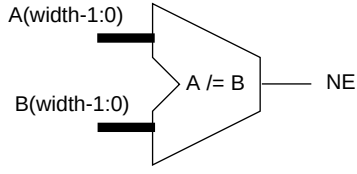
NE is an not-equal-to comparator macro. The .vhd file for this macro, located in the *ds391_path*\examples\vwlogic\macro_7k directory, can be copied and edited to implement a specific number of bits by changing the default value of the generic "width" parameter.

Component Instantiation

```
U1: NE port map (NE=>comparison, A=>in1, B=>in2);
```

Truth Table and Logic Symbol

Condition	NE
A<B	1
A=B	0
A>B	1



OBUF, OBUF_F, OBUF_S

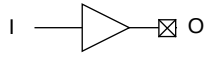
OBUF is an output buffer. OBUF and OBUF_S have slow output slew rate. OBUF_F has fast output slew rate.

Component Instantiation

```
U1: OBUF port map (O=>out_port,
    I=>driving_signal);
```

Truth Table and Logic Symbol

I	O
0	0
1	1
Z	Z



OBUFE, OBUFE_F, OBUFE_S

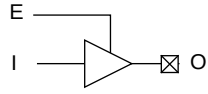
OBUFE is a 3-state output buffer. OBUFE and OBUFE_S have slow output slew rate. OBUFE_F has fast output slew rate.

Component Instantiation

```
U1: OBUF port map (O=>out_port,
     I=>driving_signal, E=enable);
```

Truth Table and Logic Symbol

I	E	O
X	0	Z
0	1	0
1	1	1



OBUFEX1, OBUFEX1F, OBUFEX1S

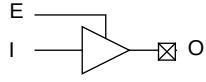
OBUFEX1 is a 3-state output buffer that uses the EPLD FOE enable signal. The E input must be driven by a BUFFOE buffer. OBUFEX1 and OBUFEX1S have slow output slew rate. OBUFEX1F has fast output slew rate.

Component Instantiation

```
U1: OBUFEX1 port map (O=>out_port,
    I=>driving_signal, E=>global_foe);
```

Truth Table and Logic Symbol

I	E	O
X	0	Z
0	1	0
1	1	1



OR2 — OR8

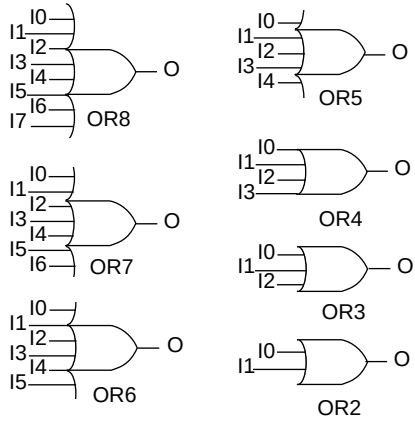
OR2 through OR8 are OR gates with 2 to 8 inputs.

Component Instantiation

```
U1: OR2 port map (O=>out, I1=>in2, I0=>in1);
```

Truth Table and Logic Symbol

I0	I1	O
0	0	0
0	1	1
1	0	1
1	1	1



SUBT

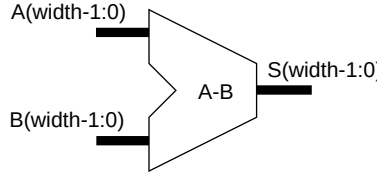
SUBT is a subtracter macro. The .vhd file for this macro, located in the *ds391_path*\examples\vwlogic\macro_7k directory, can be copied and edited to implement a specific number of bits by changing the default value of the generic "width" parameter.

Component Instantiation

```
U1: SUBT port map (S=>diff, A=>in1, B=>in2);
```

Truth Table and Logic Symbol

A	B	S
A	B	A-B



XOR2 — XOR8

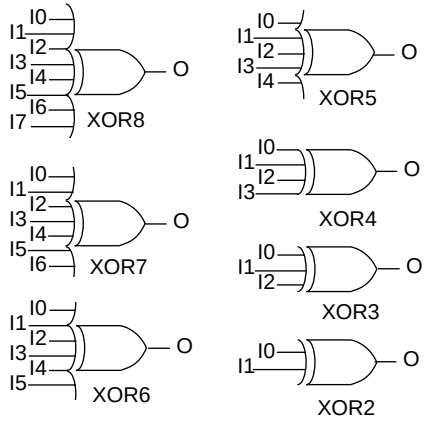
XOR2 through XOR8 are XOR gates with 2 to 8 inputs.

Component Instantiation

```
U1: XOR2 port map (O=>out, I1=>in2, I0=>in1);
```

Truth Table and Logic Symbol

I0	I1	O
0	0	0
0	1	1
1	0	1
1	1	0



Appendix D

Attributes

Attributes are used to control how the software uses the architecture specific features of the XC7000 EPLDs. See the device data sheets for more information about these device features.

Global Attributes

To apply a global attribute, place one instance of the desired attribute cell in your design and pass any internal signal through it.

LOWPWR

This attribute controls the macrocell power usage. If the LOWPWR attribute is specified it indicates that all macrocells have low power operation. If LOWPWR is not specified, the macrocells have standard power operation.

To specify low power operation for all macrocells, instantiate the LOWPWR component as follows:

```
U1: LOWPWR port map (signal_out, signal_in);
```

MRINPUT

This attribute controls the use of the Master Reset pin on the XC7354 and XC7336 devices. If the MRINPUT attribute is specified it indicates that the pin is used as a logic input. If MRINPUT is not specified, the pin is used as the Master Reset input. To specify that the Master Reset pin is used as a logic input, instantiate the MRINPUT component as follows:

```
U1: MRINPUT port map (signal_out, signal_in);
```

NO_FOE

This attribute controls the automatic use of global Fast Output Enable inputs. If the NO_FOE attribute is specified, it indicates that the software will *not* automatically assign 3-state control inputs to the global FOE inputs. If NO_FOE is not specified, the software will assign the
3-state control signals in your design to the global FOE inputs of the device, if possible.

To specify that no 3-state control signals will automatically be assigned to the global FOE inputs, instantiate the NO_FOE component as follows:

```
U1: NO_FOE port map (signal_out, signal_in);
```

NO_FCLK

This attribute controls the automatic use of global FastClock inputs. If the NO_FCLK attribute is specified, the software will not automatically assign clock signals to the global FastClock inputs. If NO_FCLK is not specified, the software will assign clock signals in your design to the dedicated FastCLK inputs of the device, if possible.

To specify that no clock signals will automatically be assigned to the FastClock nets, instantiate the NO_FCLK component as follows:

```
U1: NO_FCLK port map (signal_out, signal_in);
```

Note: Signals driven by the BUFG buffer always use FastCLK routing independent of the NO_FCLK attribute.

NO_IFD

This attribute controls the automatic usage of input pad registers. If the NO_IFD attribute is specified, it indicates that the software will not automatically use the registers in the input pads. If NO_IFD is not specified, the software will assign registers in your design to the input pads whenever possible, to reduce macrocell resource requirements.

To specify that registers will not be automatically placed into the input pads, instantiate the NO_IFD component as follows:

```
U1: NO_IFD port map (signal_out, signal_in);
```

Note: The NO_IFD attribute does not prevent you from instantiating specific input pad register and latch components.

PRELOAD

This attribute controls the use of default initial state values for registers in your design. If PRELOAD is specified, the software will use the default initial states for each register as shown in the library specifications. If the PRELOAD attribute is not specified, it indicates that the software may change the initial states of registers (unless explicitly specified) if the change allows a more efficient implementation.

To prevent the software from changing the initial state of registers in your design, instantiate the PRELOAD component as follows:

```
U1: PRELOAD port map (signal_out, signal_in);
```

Signal Attributes

Signal attributes are applied by instantiating one of the following components and passing the affected signal through it.

F

This attribute indicates either a Fast Function Block output signal or a Fast Input signal. Use this attribute to assign specific functions to EPLD Fast Function Blocks, which provide the highest performance, by passing the function's output signal through the F cell.

Use this attribute to designate a Fast Input by passing an input port signal (after an IBUF component) through the F cell. For example:

```
U1: F port map (signal_out, signal_in);
```

H

This attribute indicates a High Density Function Block output signal. Use this attribute to assign specific functions to High Density Function Blocks, which provide the most macrocell resources.

To specify that a signal is driven from a High Density Function Block, use:

```
U1: H port map (signal_out, signal_in);
```

OPT_OFF

This attribute inhibits the software from optimizing the cell that drives the connected signal.

To specify that a signal is to remain as a macrocell output, use:

```
U1: OPT_OFF port map (signal_out, signal_in);
```

OPT_UIM

This attribute can only be connected to a signal that originates from an AND gate. It forces the software to place the AND gate into the UIM.

To specify that a specific AND gate is to be implemented in the UIM, instantiate the OPT_UIM component as follows:

```
U1: UIM_OPT port map (signal_out, signal_in);
```

Note: The optimization of AND gates into the UIM is done automatically by the fitter whenever possible.

Appendix E

Fitter Reports

The primary fitter reports that you will use are:

- *design_name*.res — Resource Report showing the amount of EPLD macrocell and pin resources remaining.
- *design_name*.tim — Static Timing Report showing the calculated worst-case timing for the logic paths in your design.
- *design_name*.pin — Pin-List Report showing the final pinout of your design.

Examples of these three reports are provided in the following sections.

Resource Report

Use this report to determine the amount of EPLD resources used by your design, and the amount of remaining resources.

```
XEPLD, Version 5.0.0                                        Xilinx Inc.
                                Resource Report
 Circuit name: SCAN_TOP
Target Device: XC7354-10PC44                 Integrated: 10-17-94, 12:13PM


LOGIC RESOURCES

                  Required  Used      Remaining
Function Blocks     6         6         0
Macrocells         26        26        28


PIN RESOURCES:

Type    Req   --------Used--------------  --------Remaining---------
              I  O  I/O Fclk Foe Cen Tot  I  O  I/O Fclk Foe Cen Tot
Inputs   12   8       4                12  0      15                15
Outputs   9      0    6    1   1   1   9     0   15    0   0   0  15
I/Os      0           0                0         15                15
Fclks     1                1           1               0           0
Foes      0                    0       0                   0       0
Cens      0                        0   0                       0   0
         --- --- --  ---  --- --- --- ---
          22   8  0   10    2   1   1  22


Note:The design requires 0 pins with Fast Input capability.
     This device has 11 pins with Fast Input capability.
     The design requires 0 pins with Fast Output capability.
     This device has 0 FO and 1 I/FO remaining from original 0 FO and 16
I/FO.

 End of Resource Report
```

The Static Timing Report

Use this report to verify your design timing. This report shows the calculated worst-case timing based on the physical implementation of your design. The types of reported timing parameters are described in the following sections.

Creating the Timing Report

The timing analyzer uses the .vmh or .vmd file which is created after a successful fit of your design. The timing report has three options; a space is required in front of each option, and multiple options can be listed in any order. The options are as summarized in Table E-1:

Option	Description
-f	Creates additional timing information. This information shows the clock and data input timing used to calculate the Setup-To-Clock time. Usually this timing information is not needed.
-w	Creates 132 column reports (default is 80 columns).
-o *file_name*	Creates a *file_name*.XNF file instead of a *design_name*.XNF file. If you do not use this option, the timing analyzer overwrites any existing *design_name*.XNF file.

Table E-1 Timing Report Options

To create the Timing Report, type this at the DOS or UNIX prompt:

`timerpt [`*options***`]`** *design_name*

For example, to create a 132 column report which includes clock and data input timing (overwriting the *design_name*.XNF file), enter:

`timerpt -w -f` *design_name*

Combinational Pad-to-Pad Delays

A combinational pad-to-pad delay is calculated from an input pad through any combinational logic to an output pad. Combinational paths include any asynchronous Set and Reset inputs to registers as shown in Figure 4-8.

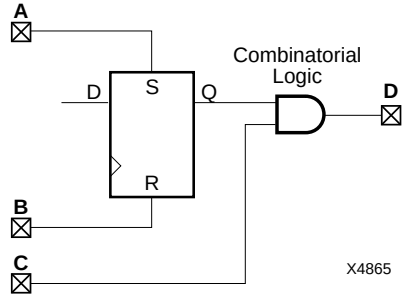


Figure E-1 Combinational Pad-to-Pad Delays

```
...
Summary of Combinational Pad-to-Pad Delays (In
Best to Worst Order)
From            To              Delay(nsec)
C               D               7.5
A               D               15.0
B               D               15.0
...
```

Setup-to-Clock Time

The setup time is calculated using the fastest clock path and the slowest data path. The timing analyzer checks all registers and reports the worst-case setup time for each pair of clock and data signals. Figure E-2 shows an example setup-to-clock path.

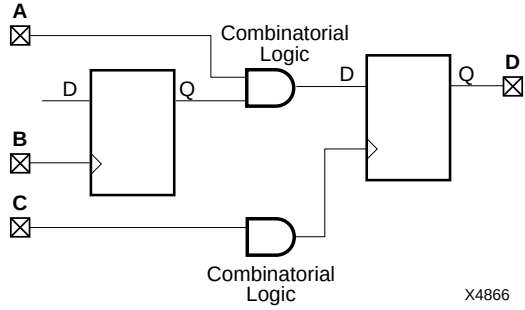


Figure E-2 Setup-to-Clock Time

```
...
Summary of Setup-to-Clock at the Pads (In Best to
Worst Order)
Data            Clock          Delay(nsec)
A               C              4.0
B               C              8.0
...
```

Clock-to-Output Delays

Clock-to-output delays are calculated from the input pad of a clock signal to the output pad. Figure E-3 shows an example.

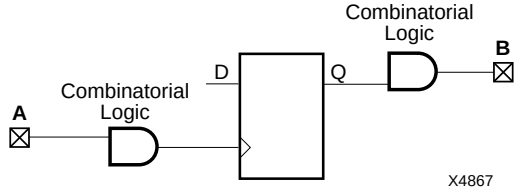


Figure E-3 Clock-to-Output Delays

```
...
Summary of Clock Pad-to-Output Pad Delays (In
Best to Worst Order)
Clock          Output         Delay(nsec)
A              B              5.5
...
```

Cycle Time

The cycle time is calculated between two registers that share the same clock. The timing report does not show cycle times for circuits that do not have a register-to-register path. Figure E-4 shows an example.

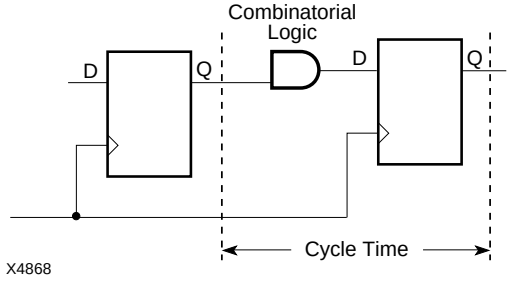


Figure E-4 Cycle Time

Note: You should also consider the setup and clock-to-output delays when determining the maximum device speed in your system.

```
...
Summary of Cycle Time Delays (In Best to Worst
Order)
(See .map file for signal names)
From            To              Delay(nsec)
FB3_9           FB3_4           8.0
FB3_6           FB3_7           8.0
...
```

Example Timing Report

The following timing report is for a Black Jack game example. This report was generated using the -f option and therefore it includes additional information at the end.

```
XEPLD, Version 5.0F                                  Xilinx Inc.
                          Timing Report
 Circuit name: bjack
Target Device: 7354-10PC68      Report Date:  5-5-94, 22:19:44

Slowest Combinational Pad-to-Pad        22.0 nsec  (Worst Case)
Slowest Setup-to-Clock at the pads      13.0 nsec  (Worst Case)
Slowest Clock-to-Output(Pad-to-Pad)     22.0 nsec  (Worst Case)
Maximum Clock Frequency CLK               76 Mhz   (Worst Case)

Summary of Combinational Pad-to-Pad Delays (In Best to Worst Order)
From            To              Delay(nsec)
ENA             SUB10           20.0
ENA             Q2              20.0
ENA             Q1              20.0
ENA             Q0              20.0
ENA             ADD10           20.0
ENA             ACE             20.0
CLKIN           ADDCLK          22.0

Summary of Setup-to-Clock at the Pads (In Best to Worst Order)
Data            Clock           Delay(nsec)
RESTART         CLK             13.0
LT22            CLK             13.0
IS_ACE          CLK             13.0
GT16            CLK             13.0
CARDOUT         CLK             13.0
CARDIN          CLK             13.0

Summary of Clock Pad-to-Output Pad Delays (In Best to Worst Order)
Clock           Output          Delay(nsec)
CLK             SUB10           10.0
CLK             Q2              10.0
CLK             Q1              10.0
CLK             Q0              10.0
CLK             ADD10           10.0
CLK             ACE             10.0
CLK             ADDCLK          22.0
```

```
Summary of Cycle Time Delays (In Best to Worst Order)
(See .map file for signal names)
From            To              Delay(nsec)
FB3_9           FB3_4           13.0
FB3_6           FB3_7           13.0
FB3_8           FB3_4           13.0
FB3_7           FB3_9           13.0
FB3_7           FB3_4           13.0
...             ...             ...
FB3_6           FB3_6           13.0
FB3_8           FB3_9           13.0
FB3_5           FB3_6           13.0
FB3_4           FB3_9           13.0
FB3_8           FB3_7           13.0

**************************************************************************

                         Combinational Pad-to-Pad Delays(nsec)

   \From             C     E
    \                L     N
     \               K     A
      \              I
       \             N
  To    \------------------

ACE                     20.0
ADD10                   20.0
ADDCLK           22.0
Q0                      20.0
Q1                      20.0
Q2                      20.0
SUB10                   20.0
**************************************************************************
 Clock Pad-to-Output Pad Delays(nsec)

   \Clock            C
    \                L
     \               K
Output\--------------

ACE              10.0
ADD10            10.0
ADDCLK           22.0
Q0               10.0
Q1               10.0
Q2               10.0
SUB10            10.0
**************************************************************************
```

```
                        Register-to-Register Delays(nsec)

   \From             F     F     F     F     F     F
    \                B     B     B     B     B     B
     \               3     3     3     3     3     3
      \              _     _     _     _     _     _
       \             4     5     6     7     8     9
   To   \-----------------------------------------

FB3_4             13.0        13.0  13.0  13.0  13.0
FB3_5                         13.0  13.0  13.0  13.0
FB3_6             13.0  13.0  13.0  13.0  13.0  13.0
FB3_7                   13.0  13.0  13.0  13.0
FB3_8             13.0  13.0  13.0  13.0  13.0  13.0
FB3_9             13.0  13.0  13.0  13.0  13.0  13.0
********************************************************************************
                        Setup Delays(nsec)

   \Clock            C
    \                L
     \               K
  Data\--------------

CARDIN            13.0
CARDOUT           13.0
GT16              13.0
IS_ACE            13.0
LT22              13.0
RESTART           13.0
********************************************************************************
                        Dpath Delays(nsec)

   \From             C     C     C     G     I     L     R
    \                A     A     L     T     S     T     E
     \               R     R     K     1     _     2     S
      \              D     D           6     A     2     T
       \             I     O                 C           A
        \            N     U                 E           R
         \                 T                             T
   To     \-----------------------------------------------

FB3_4                       15.5        15.5
FB3_5                       15.5              15.5
FB3_6             15.5      15.5  15.5        15.5
FB3_7             15.5  15.5  15.5                    15.5
FB3_8                   15.5  15.5  15.5  15.5  15.5  15.5
FB3_9                       15.5
********************************************************************************
```

```
                        ClkPath Delays(nsec)

   \From            C
    \               L
     \              K
   To \--------------

FB3_4             2.5
FB3_5             2.5
FB3_6             2.5
FB3_7             2.5
FB3_8             2.5
FB3_9             2.5

 End of Timing Report
```

Pin-List Report

Use this report to see the final EPLD pin assignments.

```
XEPLD, Version 5.0.0
Xilinx Inc.
                                        Pin-List Report

 Circuit name: SCAN_TOP
Target Device: XC7354-10PC44                    Integrated:
10-17-94, 12:13PM

Pkg  Pin   Pin    Pin
Pin  Type  Use    Name
---  ----  ---    ----
1    MR
2     I     I     DATA6
3     I     I     DATA5
4     I     I     DATA4
5    CLK    I     CLOCK
6    CLK    O     COUNT2
7     I     I     DATA3
8    I/O    I     DATA2
9    I/O    I     DATA1
10   VSS
11   I/O    I     DATA0
12   I/O    I     CLEAR
13   I/O   tie    (unused)
14   I/O   tie    (unused)
15   I/O   tie    (unused)
16   I/O   tie    (unused)
17   I/O    O     COUNT7
```

```
18   I/O   tie    (unused)
19   I/O   tie    (unused)
20   I/O   tie    (unused)
21   VCC
22   I/O    O     COUNT6
23   VSS
24   I/O    O     COUNT4
25   I/O    O     DONE
26   I/O    O     COUNT1
27   I/O    O     COUNT0
28    I     I     WRITE_START
29   I/O   tie    (unused)
30   I/O   tie    (unused)
31   VSS
32   VCC
33   I/O   tie    (unused)
34   I/O   tie    (unused)
35   I/O   tie    (unused)
36   I/O   tie    (unused)
37   I/O   tie    (unused)
38   I/O   tie    (unused)
39   CEN    O     COUNT5
40   FOE    O     COUNT3
41   VCC
42    I     I     WRITE_END
43    I     I     START
44    I     I     DATA7

Pin Use Legend:

I     - input
O     - output
I/O   - input/output
I-L   - input uses latch
I-R   - input uses register
I/O-L - input/output uses latch
I/O-R - input/output uses register
NC    - not connected/not available
tie   - unused pin must be tied to VCC or GND
(O)   - unused pin attached to used macrocell

 End of Pin-List Report
```

Index

Trademark Information

XILINX®, XACT, XC2064, XC3090, XC4005, and XC-DS501 are registered trademarks of Xilinx. All XC-prefix product designations, XACT-Floorplanner, XACT-Performance, XAPP, XAM, X-BLOX, X-BLOX plus, XChecker, XDM, XDS, XEPLD, XPP, XSI, BITA, Configurable Logic Cell, CLC, Dual Block, FastCLK, HardWire, LCA, Logic Cell, LogicProfessor, MicroVia, PLUSASM, SMARTswitch, UIM, VectorMaze, VersaBlock, VersaRing, and ZERO+ are trademarks of Xilinx. The Programmable Logic Company and The Programmable Gate Array Company are service marks of Xilinx.

IBM is a registered trademark and PC/AT, PC/XT, PS/2 and Micro Channel are trademarks of International Business Machines Corporation. DASH, Data I/O and FutureNet are registered trademarks and ABEL, ABEL-HDL and ABEL-PLA are trademarks of Data I/O Corporation. SimuCad and Silos are registered trademarks and P-Silos and P/C-Silos are trademarks of SimuCad Corporation. Microsoft is a registered trademark and MS-DOS is a trademark of Microsoft Corporation. Centronics is a registered trademark of Centronics Data Computer Corporation. PAL and PALASM are registered trademarks of Advanced Micro Devices, Inc. UNIX is a trademark of AT&T Technologies, Inc. CUPL, PROLINK, and MAKEPRG are trademarks of Logical Devices, Inc. Apollo and AEGIS are registered trademarks of Hewlett-Packard Corporation. Mentor and IDEA are registered trademarks and NETED, Design Architect, QuickSim, QuickSim II, and EXPAND are trademarks of Mentor Graphics, Inc. Sun is a registered trademark of Sun Microsystems, Inc. SCHEMA II+ and SCHEMA III are trademarks of Omation Corporation. OrCAD is a registered trademark of OrCAD Systems Corporation. Viewlogic, Viewsim, and Viewdraw are registered trademarks of Viewlogic Systems, Inc. CASE Technology is a trademark of CASE Technology, a division of the Teradyne Electronic Design Automation Group. DECstation is a trademark of Digital Equipment Corporation. Synopsys is a registered trademark of Synopsys, Inc. Verilog is a registered trademark of Cadence Design Systems, Inc.

Xilinx does not assume any liability arising out of the application or use of any product described or shown herein; nor does it convey any license under its patents, copyrights, or maskwork rights or any rights of others. Xilinx reserves the right to make changes, at any time, in order to improve reliability, function or design and to supply the best product possible. Xilinx will not assume responsibility for the use of any circuitry described herein other than circuitry entirely embodied in its products. Xilinx devices and products are protected under one or more of the following U.S. Patents: 4,642,487; 4,695,740; 4,706,216; 4,713,557; 4,746,822; 4,750,155; 4,758,985; 4,820,937; 4,821,233; 4,835,418; 4,853,626;

4,855,619; 4,855,669; 4,902,910; 4,940,909; 4,967,107; 5,012,135; 5,023,606; 5,028,821; 5,047,710; 5,068,603; 5,140,193; 5,148,390; 5,155,432; 5,166,858; 5,224,056; 5,243,238; 5,245,277; 5,267,187; 5,291,079; 5,295,090; 5,302,866; 5,319,252; 5,319,254; 5,321,704; 5,329,174; 5,329,181; 5,331,220; 5,331,226; 5,332,929; 5,337,255; 5,343,406; 5,349,248; 5,349,249; 5,349,250; 5,349,691; 5,357,153; 5,360,747; 5,361,229; 5,362,999; 5,365,125; 5,367,207; 5,386,154; 5,394,104; 5,399,924; 5,399,925; 5,410,189; 5,410,194; 5,414,377; RE 34,363, RE 34,444, and RE 34,808. Other U.S. and foreign patents pending. Xilinx, Inc. does not represent that devices shown or products described herein are free from patent infringement or from any other third party right. Xilinx assumes no obligation to correct any errors contained herein or to advise any user of this text of any correction if such be made. Xilinx will not assume any liability for the accuracy or correctness of any engineering or software support or assistance provided to a user.

Xilinx products are not intended for use in life support appliances, devices, or systems. Use of a Xilinx product in such applications without the written consent of the appropriate Xilinx officer is prohibited.