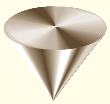
XILINX®

ONLINE

XEPLD SCHEMATIC DESIGN GUIDE

TABLE OF CONTENTS

INDEX



GO TO OTHER BOOKS

XACT™ STEP

0401417

Contents

Getting Started with Schematic Design

This chapter will help you quickly understand how to develop a schematic design using XEPLD. Brief schematic design examples are included, illustrating the device-independent schematic library and the automatic PAL conversion process.

An Overview of Schematic Design Methods

A schematic design defines the functionality of a logic circuit using one or more schematic files, each of which contains components whose functions are already defined (74xx TTL or similar functions) and components for which you define the function using behavioral modules. Figure 1-1 summarizes the design flow.

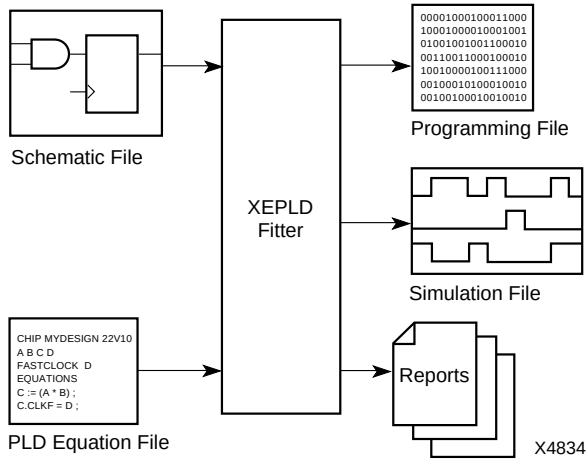


Figure 1-1 Basic Schematic Design Flow

The Viewlogic, OrCAD, Mentor, and Cadence software packages are supported.

The following sections provide an overview of methods for creating schematics and behavioral modules for schematic designs.

Using the Unified Library

The Unified Library allows you to create a device-independent design to test how the design works in different devices, or to create a device-specific design to take full advantage of a device's unique architectural features.

The Unified Library contains all the component symbols for all the available device families. As illustrated in Figure 1-2, most of the symbols can be used in designs targeted for any Xilinx device, but some of the symbols are specific to one or more device families.

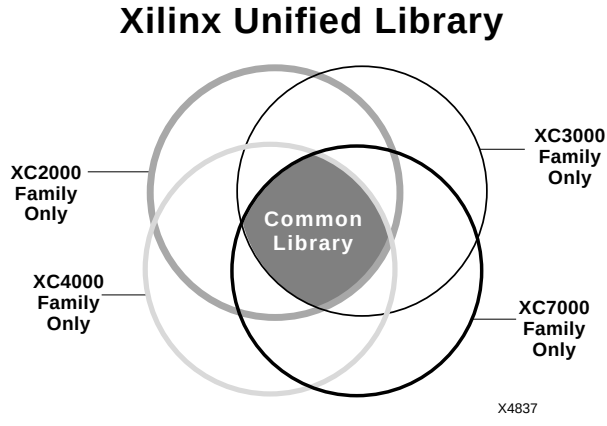


Figure 1-2 Device Families and the Unified Library

The common symbols are automatically mapped to the chosen target device. The same common symbol may be mapped differently to target devices with different architectures.

The "Device-Independent Design" chapter describes the library components and how to retarget an existing FPGA design to an EPLD device.

For more information about the Unified Library, refer to the *XACT Libraries Guide*.

Behavioral Modules and PAL Conversion

There are two ways to include a behavioral module in a schematic design. Both are described in the chapter entitled "Using Behavioral Modules in Schematics."

- Use a PLD symbol from the XC7000 library. This method is best if you have a module that is already targeted to a standard PAL such as a 22V10.
- Create a custom component, and use SymGen to create the custom symbol for it. This method is best if your behavioral module is not already targeted to a standard PAL, for example if it is a state machine expressed in a high-level language.

The PLD symbols are a special group of EPLD-specific symbols in the Unified Library. You define the logic behind these symbols in a behavioral design file. To create a custom symbol, you create the behavioral design file, then use SymGen to create a symbol based on the file.

The original design file can be a new file created using the PLUSASM™ language or an existing PAL file converted to PLUSASM. You can convert the following types of files to PLUSASM files and use them with PLD symbols:

- ABEL
- PALASM
- JEDEC

Figure 1-3 shows the design flow for including a behavioral module.

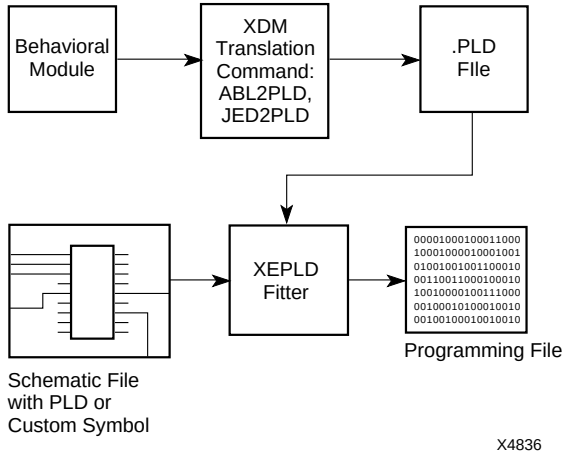


Figure 1-3 Schematic PAL Conversion Flow

Schematic Design Flow Example

This section runs through the entire schematic design process, from creating a design to programming and simulating the design. The following device-independent design, a 4-bit Johnson counter, is used as an example:

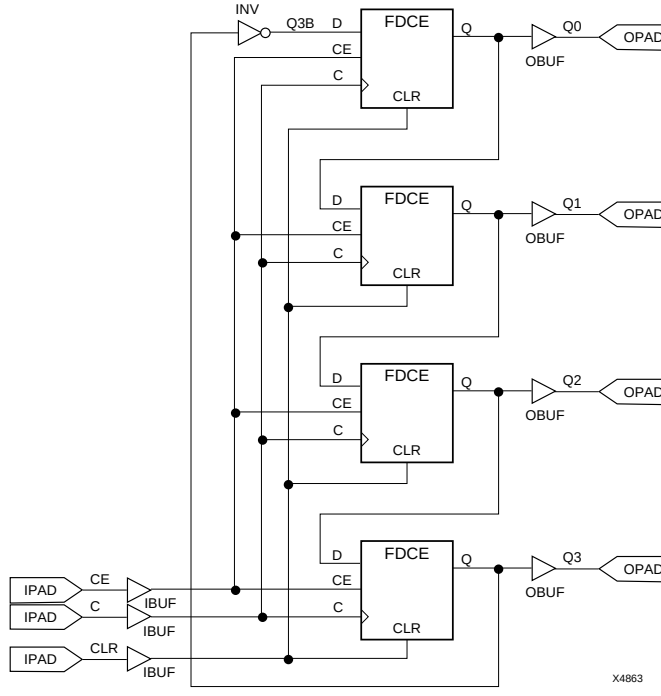


Figure 1-4 Example 4-Bit Johnson Counter Design

Simulation results for this design are shown in Figure 1-5.

This design contains no behavioral modules. For an example of a design that includes a behavioral module, see the chapter entitled "Using Behavioral Modules in Schematics."

The steps are summarized for Viewlogic and OrCAD.

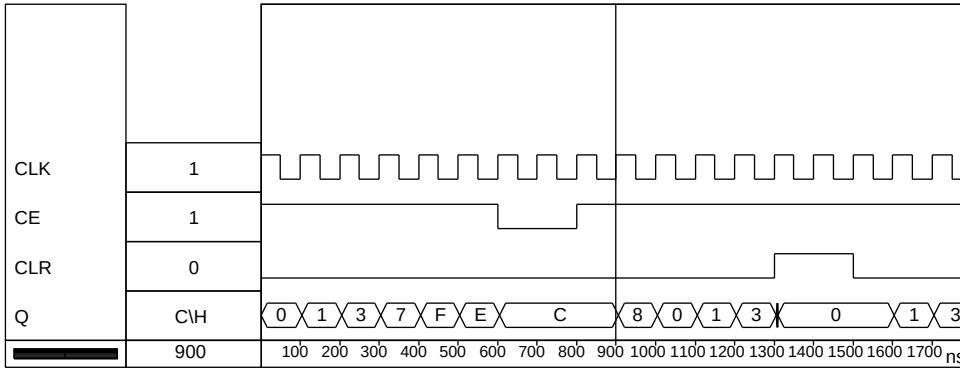


Figure 1-5 Example Viewlogic Functional Simulation Results

Viewlogic Procedure

Step 1 — Configure ViewDraw

Create a directory for your design. Copy the viewdraw.ini file from your workview\standard directory into that directory, and edit the end of it to include the following lines:

```
DIR [pw] . (primary)
DIR [m] \correct_path\unified\xc7000 (xc7000)
DIR [m] \correct_path\unified\builtin (builtin)
```

Note: Use [r] instead of [m] if you are using a UNIX workstation.

You could also use ViewFile to configure your library directory specifications.

Step 2 — Enter XDM and Select the Device

Enter XDM by typing the following at the operating system prompt:

```
XDM
```

Select the device family and part by clicking on the **Family** and **Part** fields in the lower left corner of the screen. For the family, select "XC7300". For the part, select "XC7318-PC44". For the speed grade, select "–5".

Step 3 — Enter Workview and Draw the Design

Enter Workview by selecting **DesignEntry → Workview** from the XDM menu.

When the Workview screen appears, type the following to create a new schematic:

```
sch jcount
```

Draw the design as shown in Figure 1-5. For more information about using ViewDraw, see the *Viewlogic Interface User Guide*.

If you do not wish to draw the design, you can copy the jcount.1 schematic file in the \xact\tutorial\vwlogic\jcount\sch directory ($XACT/tutorial/vwlogic/jcount/sch on the workstation).

Note: It is important that you label the nets between the IPADs and IBUFs and between the OPADs and OBUFs, because these names will appear in reports and on simulation traces.

Save your design using the **File → Write** command.

Step 4 — Perform Functional Simulation (Optional)

If you wish to perform functional simulation, first use the **Export → Wirelist → ViewSim** command to create a simulation file.

Use the **Window → Open → ViewSim S/D** command to enter ViewSim. Type ↵ in response to the network file name prompt.

Type **jcount** to run the following command file, jcount.cmd, which simulates the design:

```
restart
vector Q Q[3:0]
wave jcount.wfm CLK CE CLR Q
clock c 1 0
step 50ns
h prld
h CE
l CLR
cycle
l prld
cycle 5
l CE
```

```
cycle 2
h CE
cycle 5
h CLR
cycle 2
l CLR
cycle 3
```

To view the waveform file, type **wv**, then type ↵ when prompted for the file name.

Exit Workview by typing **quit**. If asked `Are you sure?` type **y**. You are back in XDM.

Step 5 — Fit the Design and Create a Programming File

Select **Translate** → **XEMake**. Select **Done** in response to the options window. Select the **jcount.1** file from the list. Finally, select **Make Intel HEX File** as the target. Type **jcnt.a** as the signature.

The XEMake command creates a netlist, fits the design, and creates a programming file in one step.

Step 6 — Examine the Reports

Examine the reports to verify that the design was implemented as you expected. The following reports are most useful for schematic designs:

- *design*.ERR — This is the error log. Use it to correct mistakes in your design.
- *design*.RES — The resource report lists the device resources used by the design and the resources remaining.
- *design*.PIN — The pinlist report shows how the external nets in your design were mapped to the device pins.
- *design*.MAP — The mapping report tells you how the logic in the design was mapped to the device.

Step 7 — Prepare for Timing Simulation

To prepare for timing simulation, select **Verify** → **XSimMake**. Select **Viewlogic_EPLD_Timing** as the flow name. Select **jcount.vmh** as the file name.

Step 8 — Perform Timing Simulation

When XSimMake is finished running, enter Workview again and follow the same instructions as for functional simulation.

OrCAD Procedure

Step 1 — Enter XDM and Select the Device

Enter XDM by typing the following at the operating system prompt:

```
XDM
```

Select the device family and part by clicking on the **Family** and **Part** fields in the lower left corner of the screen. For the family, select "XC7300". For the part, select "XC7318-PC44". For the speed grade, select "–5".

Step 2 — Enter and Configure OrCAD

Enter OrCAD by selecting **DesignEntry** → **OrCAD** from the menu.

Double click on **Design Management Tools**. Click on **Create Design**.

In the prompt box that appears, type **jcount**↵, then click on **OK**.

The JCOUNT design appears, highlighted, in the list. Click on **Suspend to System**. Type the following at the DOS prompt:

```
xdraft 7
```

(wait for the command to complete)

```
exit
```

You are back in the Design Management Tools window. Click on **OK** to exit.

Step 3 — Enter Draft and Draw the Design

Double click on **Schematic Design Tools**. Double Click on **Draft**. A blank schematic window appears.

Draw the design as shown in Figure 1-5. For more information about using Draft, see the *OrCAD Interface User Guide*.

If you do not wish to draw the design, you can copy the jcount.sch schematic file in the xact\tutorial\orcad\jcount directory.

Note: It is important that you label the nets between the IPADs and IBUFs and between the OPADs and OBUFs, because these names will appear in reports and on simulation traces.

Step 4 — Add Simulation Information

To add trace information, follow these steps for each net that is between an IPAD and an IBUF or between an OBUF and an OPAD:

1. Place the mouse cursor over the net and select the **Place** → **Trace Name** command.
2. In response to the `Trace Name?` prompt, type the name of the net, for example **CE**.
3. Double click to place the trace marker, then click with the right button to exit the command.

To add stimulus information, follow these steps for each net that is between an IPAD and an IBUF:

1. Place the mouse cursor over the net and select the **Place** → **Stimulus** command.
2. In response to the `Stimulus?` prompt, type the following:

   For CE:

   ```
   0:1 6000:0 8000:1
   ```

   For CLK:

   ```
   0:0 500:T 1000:G:500
   ```

   For CLR:

   ```
   0:0 13000:1 15000:0
   ```

3. Double click to place the stimulus marker, then click with the right button to exit the command.

Save your design using the **Quit** → **Update File** command. Exit Draft using **Quit** → **Abandon Edits**.

To exit OrCAD, double click on **To Main**, then on **Exit ESP**. You are back in XDM.

Step 5 — Prepare for Functional Simulation (Optional)

To prepare for functional simulation, select **Verify** → **XSimMake**. Select **Orcad_EPLD_Func** as the flow name. Select **jcount.sch** as the file name.

XSimMake produces ASCII versions of OrCAD stimulus and trace files, which you must convert to binary. The conversion command is memory-intensive, so it is best to execute it in DOS.

Quit XDM using the **Quit** command. Type the following at the DOS prompt:

```
asctovst jcount.ast
```

(wait for the command to complete)

```
asctovst jcount.atr
```

(wait for the command to complete)

```
xdm
```

You are back in XDM.

Step 6 — Perform Functional Simulation (Optional)

Enter OrCAD by selecting **DesignEntry** → **OrCAD** from the menu. Double click on **Design Management Tools**, select **JCOUNT** from the list, and click on **OK**.

Double Click on **Digital Simulation Tools**.

Click once on **Simulate**. Select **Local Configuration** and then **Configure Simulate**. Change jcount.inf to **jcount.vst**, then select **OK**.

Double click on **Simulate**. A blank simulation waveform window appears with the R, Q3, Q2, Q1, Q0, CE, and CLK nets listed.

Select **Edit Stimulus** → **Yes**. The Stimulus Editor window appears.

Select **Add**. Press the down arrow key to highlight the **Signal Name** field, then select **Browse**. Type a **P** to scroll down to the net names beginning with P. Press the down arrow key to highlight **PRLD**, then press ↵.

Press the down arrow key to highlight the **Initial Value** field. Type **1**.

Press the down arrow key again. Select **Add**. In response to `Time of Function?` type **10**, then press ↵ to accept a 0 (zero) value.

Select **Return** to return to the main stimulus editor, select **Write** → **Yes** to add PRLD to the stimulus file, and select **Use** to return to the waveform window.

Select **Trace** → **Change View** and enter **150**. Select **Run Simulation** and enter **18000**. The simulation waveforms appear.

Exit Simulate using **Quit** → **Abandon Simulation**.

To exit OrCAD, double click on **To Main**, then on **Exit ESP**. You are back in XDM.

Step 7 — Fit the Design and Create a Programming File

Select **Translate** → **XEMake**. Select **Done** in response to the options window. Select the **jcount.sch** file from the list. Finally, select **Make Intel HEX File** as the target. Type **jcnt.a** as the signature.

The XEMake command creates a netlist, fits the design, and creates a programming file in one step.

Step 8 — Examine the Reports

Examine the reports to verify that the design was implemented as you expected. The following reports are most useful for schematic designs:

- *design*.ERR — This is the error log. Use it to correct mistakes in your design.
- *design*.RES — The resource report lists the device resources used by the design and the resources remaining.
- *design*.PIN — The pinlist report shows how the external nets in your design were mapped to the device pins.
- *design*.MAP — The mapping report tells you how the logic in the design was mapped to the device.

Step 9 — Prepare for Timing Simulation

To prepare for timing simulation, select **Verify** → **XSimMake**. Select **Orcad_EPLD_Timing** as the flow name. Select **jcount.vmh** as the file name.

Step 10 — Perform Timing Simulation

If you skipped functional simulation, follow the preceding instructions for functional simulation. If you did perform functional simulation, follow the instructions in this section.

Enter OrCAD by selecting **DesignEntry** → **OrCAD** from the menu. Double click on **Design Management Tools**, select **JCOUNT** from the list, and click on **OK**.

Double Click on **Digital Simulation Tools**. Double click on **Simulate**. A blank simulation waveform window appears with the R, Q3, Q2, Q1, Q0, CE, and CLK nets listed.

Select **Trace** → **Change View** and enter **125**. Select **Run Simulation** and enter **15000**. The simulation waveforms appear.

Exit Simulate using **Quit** → **Abandon Simulation**.

To exit OrCAD, double click on **To Main**, then on **Exit ESP**. You are back in XDM.

Chapter 2

Device-Independent Design

This chapter discusses how and why to create a device-independent or device-specific design. It also explains how to take an existing FPGA design and retarget it to an EPLD device.

Choosing Components

The Unified Library contains all Xilinx component symbols, some of which are specific to one device family, some of which are common to two or more families, and some of which are common to all families.

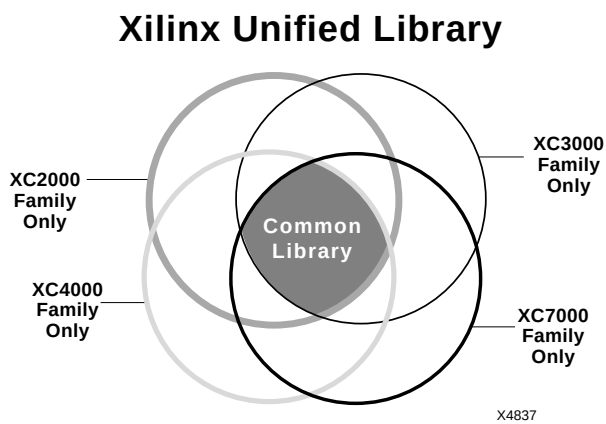


Figure 2-1 Common Symbols in the Unified Library

When a component of the same name is present in multiple families' libraries, it has the same functionality and graphic symbol body, and similarly named pins. However, the component's implementation, including whether the symbol is a primitive (behavioral module) or macro (schematic module), may vary between families.

When to Use EPLD-Specific Components

In general, common library components work well for EPLD designs. You should use XC7000-specific components only under these special conditions:

- If you are cascading arithmetic components, you should use EPLD-specific arithmetic components, because the carry lines that go between components are mapped to the carry chain.
- If you are cascading up/down counters, you should use EPLD-specific counters, because the separate up and down terminal counts can be placed in the UIM for greater speed and density.
- If you need input pad registers with clock enable, you must use an EPLD-specific component such as IFDX1.

For example, suppose you were working on a design that needed an 8-bit full adder. In most designs, you could use a device-independent adder, such as ADD8. If this component is used in an EPLD design, the internal logic is mapped onto the special EPLD arithmetic carry lines; if used in an FPGA design, the logic is mapped in the way that is most efficient for the FPGA's architecture.

However, if you needed to cascade two 8-bit adders, it would be most efficient to use EPLD-specific adders, because the carry lines that go between components would be mapped to the carry chain.

When to Use Common Components

To make your design device-independent, use only the symbols common to all device families — the shaded area in Figure 2-1. The XACT software automatically maps the symbols in your design onto the chosen target device. Creating a device-independent design allows you to easily test your design with different Xilinx devices.

For a complete list of Unified Library components and their compatibility with the different Xilinx device families, refer to the *XACT Libraries Guide*.

Attributes and Device Independence

If you want your design to be completely device-independent, do not use schematic attributes. The only attributes common to FPGA and EPLD devices are as follows:

- PART=*device_name* (or PARTTYPE in OrCAD)
- LOC=*device_pin*

Even these attributes are not truly device independent, because you must change the values when you change devices. All other EPLD-compatible attributes are EPLD-specific. For more information about these attributes, refer to the "Attributes" appendix. These attributes are also described in the context of how they are used in the "EPLD Architecture and Design Tradeoffs" and "Design Applications" chapters of this manual.

General Conversion Procedure: FPGA to EPLD

The basic steps for retargeting an FPGA design to an EPLD device are as follows. Examples in this section assume you are converting from the XC3000 family.

1. Change (cd) to your working directory and reconfigure the libraries in your CAE tool for the XC7000 family.

   - If you are using Viewlogic software, edit your viewdraw.ini file so it looks like this:

     ```
     DIR [pw] . (primary)
     DIR [r] /correct_path/unified/xc3000 (xc3000)
     DIR [r] /correct_path/unified/xc7000 (xc7000)
     DIR [r] /correct_path/unified/xblox (xblox)
     DIR [r] /correct_path/unified/builtin (builtin)
     ```

     Substitute the path to your libraries for *correct_path*. Leave the XC3000 library reference in the file, because the Altran command, which you will use in the next step, requires it.

   - If you are an OrCAD user, type the following at the DOS prompt while in your working directory:

     ```
     xdraft 7
     ```

2. Use any necessary conversion program to make the symbols in your schematic reference the reconfigured libraries. If you are using OrCAD software, skip this step.
   - If you are using Viewlogic software, use the Altran command, where *N* is 2, 3, or 4:

     `altran -l primary xc`*N*`000=xc7000`**

3. Enter your CAE tool and open your design.
4. Find all the library components in the schematic that are not EPLD-compatible, and use the *XACT Libraries Guide* to find EPLD-compatible equivalents.
   - If you have run Altran on a Viewlogic schematic, incompatible components appear as white squares in the schematic.
   - If you have run XDraft 7 on an OrCAD schematic, the incompatible components generate error messages in OrCAD and do not appear in the schematic.
5. If necessary, use EPLD-specific components to obtain a more efficient design. In most cases, device-independent components are mapped onto the EPLD architecture efficiently, but there are exceptions, which are described in the "When to Use EPLD-Specific Components" section earlier in this chapter.
6. Remove all attributes except PART=*device_name* and LOC=*pin_name*. You can change the device name to the new target device if you wish. If you wish to reassign pins, see the "EPLD Architecture and Design Tradeoffs" chapter for information about EPLD pin assignment. If you do not wish to reassign pins, remove the LOC=*pin_name* attributes as well.
7. You may want to look over the list of EPLD attributes, especially the optimization attributes, and use them to fine-tune your design. The software optimizes everything it can by default; you may want to prevent optimization in some cases, for example to ensure that an internal node is visible during timing simulation.
8. Enter XDM. Run the **`XEMake`** command to fit your design. Run **`XSimMake`** if you also want to perform functional or timing simulation.

9. When you perform either functional or timing simulation, you must set the PRLD (preload) signal High then Low instead of setting GR (global reset) Low then High.
10. If you wish to perform timing simulation, you may have to change the nodes you drive and monitor. The EPLD fitter optimizes the logic, which makes many of the internal nodes in the design disappear. However, all external signals are always visible.

Note: If your FPGA design has RAM, ROM, or other elements that do not have EPLD equivalents, you cannot retarget your design unless you redesign those parts.

Converting Behavioral Modules

If your design contains Xilinx-ABEL or XBLOX modules, you must perform these additional steps before running XEMAKE:

1. Change the encoding of state machine modules. You do not have to rewrite the logic, just the state assignment. For FPGAs, which are rich in registers, one-hot encoding (a symbolic state machine) is most efficient. For EPLDs, which are rich in product terms, maximal encoding (an encoded state machine) is most efficient. Conversion may be unnecessary for very simple state machines.

   For more information about symbolic and encoded state machines, see the "Simulating an ABEL-HDL Design" and "Converting Encoded State Machine to Symbolic State Machine" sections in the "Design Examples" chapter of the *Xilinx ABEL User Guide*. You should follow the steps of the latter section in reverse.
2. Convert the files behind all Xilinx-ABEL modules to PLUSASM using the **ABL2PLD** command. Use the existing Xilinx-ABEL symbols, but remove the DEF=XABEL and FILE=*file_name* attributes, replacing them with PLD=*file_name*. To ensure that the software does not process old files, delete the *file_name*.xnf files in the xnf directory.
3. Convert the XBLOX modules. There is no straightforward conversion—you can rewrite the logic using a behavioral entry tool (PLUSASM, Xilinx-ABEL, ABEL) or create an EPLD-compatible lower-level schematic.

FPGA to EPLD Conversion Example: CALC Design

The CALC design is extensively documented in the tutorial chapters of the Xilinx Interface User Guide for your CAE tool. It is used as an example of an FPGA design. This section describes the steps necessary to convert the XC3000 version of this design to an EPLD design.

The steps are different for each CAE tool. The procedure for Viewlogic users is described first, followed by OrCAD.

Procedure for Viewlogic Users

Reconfiguring the Libraries and Schematic Symbols

To reconfigure the libraries for the XC7000 family and update all the symbols so their properties are compatible with XC7000 devices, follow these steps:

1. Copy the CALC tutorial files to a directory under your home directory as described in the *Xilinx Viewlogic Interface User Guide*. Change (cd) to the calc/soln_3k directory.
2. Edit your viewdraw.ini file so it looks like this:

   ```
   DIR [pw] . (primary)
   DIR [r] /correct_path/unified/xc3000 (xc3000)
   DIR [r] /correct_path/unified/xc7000 (xc7000)
   DIR [r] /correct_path/unified/xblox (xblox)
   DIR [r] /correct_path/unified/builtin (builtin)
   ```

   Substitute the path to your libraries for *correct_path*. Leave the XC3000 library reference in the file, because the Altran command, which you will use in the next step, requires it.
3. While in your working directory, type this command at the operating system prompt:

   ```
   altran -l primary xc3000=xc7000
   ```

Editing the Schematic

To edit the schematic so all its symbols are compatible with XC7000 devices, follow these steps:

1. Start up XDM. Select the **`Workview`** command from the **`Design-Entry`** menu.
2. Open the CALC schematic. Select the OSC_3K symbol and select the **`Level`** → **`Push`** → **`Schematic`** command.
3. Edit the schematic so it looks like Figure 2-2:

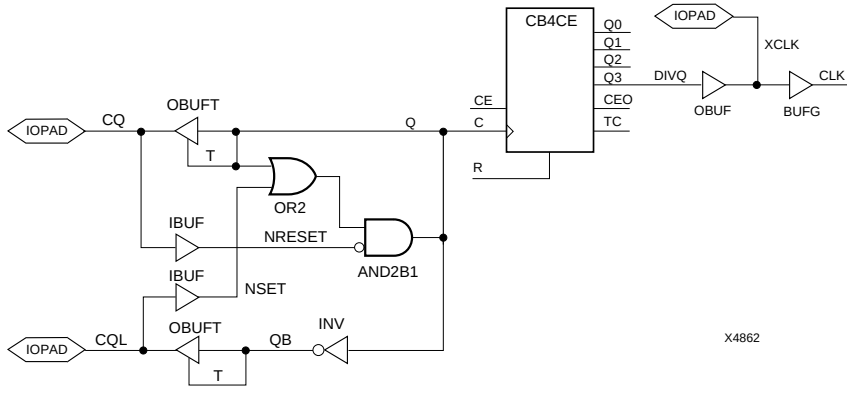


Figure 2-2 New Viewlogic Oscillator Schematic for EPLD CALC

4. Be sure to label the new IOPAD net "XCLK".
5. Select the **`File`** → **`Write`** command. Select the **`Level`** → **`Pop`** command. (If you wish, you can use the **`File`** → **`Write to`** command to save this schematic as OSC_7K. If you save this schematic to a new name, be sure to save the corresponding symbol file to the same name. In addition, make sure the new symbol is added to and saved in the top-level CALC schematic.)
6. Search the top-level schematic and every sub-schematic for attributes, including LOC= pin assignments. Delete all except the PART= attribute. Attributes are displayed in yellow to distinguish them from labels, which are white. Save each schematic that you change using the **`File`** → **`Write`** command.

7. Use the **Change → Text** command to change the part name as follows:

   PART=7372-10PC68

8. Save the top-level schematic using the **File → Write** command.

Performing Functional Simulation

To perform functional simulation, follow these steps:

1. Select the **Export → Wirelist → ViewSim** command.
2. Run simulation as described in the ViewSim tutorial in the *Viewlogic Interface User Guide*, except change these lines in the calc_3k.cmd file (which you can rename calc_7k.cmd if you wish):

   ```
   l gr
   ...
   h gr
   ```

   to the following:

   ```
   h prld
   ...
   l prld
   ```

Running the Fitter Commands

To map the design onto a target XC7000 device, follow these steps:

1. Exit Workview by typing the following:

   quit

2. Select the **XEMake** command from the **Translate** menu in XDM.
3. Select **Done** to select all the default options.
4. Select the **calc.1** file from the list.
5. Select **Make design database** as the target.

Performing Timing Simulation

To perform timing simulation, follow these steps:

1. Select the **XSimMake** command and **Viewlogic_EPLD_Timing** as the flow name. Select the **calc.vmh** file from the list.

2. Run simulation as described in the ViewSim tutorial in the *Viewlogic Interface User Guide*, except change the lines in the Vector Definition, Simulation Output Definition, Clock Definition, and Global Reset & Initial Input Values to those shown in the following file. The ... indicates comments that have been removed.

   You must make these changes because the EPLD fitter optimizes the logic, removing many internal nodes. This makes the design more efficient, but harder to simulate.

```
|--------------------VECTOR DEFINITION-------------------
...
vector SW sw7\sw6_p sw7\sw5_p sw7\sw4_p sw7\sw3_p +
sw7\sw2_p sw7\sw1_p sw7\sw0_p
| You can also use bus syntax when defining vectors
vector ALU alu[3:0]
vector LED_P led\led[3:0]_p
| Set radices for vectors
|   The default radix is binary for input, hex for output
radix hex SW ALU
radix bin LED_P
|---------------SIMULATION OUTPUT DEFINITION--------------
...
wave calc_7k.wfm osc_7k\xclk SW exc_p ALU LED_P we rst
| Save simulation values for these nodes
watch osc_7k\xclk SW exc_p ALU LED_P we rst
| Output the values of all watched signals each time
|   "xclk" goes high.  Create tabular output.
break osc_7k\xclk 1 do (print > calc_7k.tab)
| Output node and vector transitions and simulation time
|   whenever any of the nodes or vectors changes state
trace osc_7k\xclk SW exc_p ALU LED_P we rst > calc_7k.trc
|-------------------CLOCK DEFINITION----------------------
clock osc_7k\xclk 1 0
| Use a clock period of 100ns.  Set stepsize=50ns
step 50ns
|------------GLOBAL RESET & INITIAL INPUT VALUES----------
| Set initial values for all inputs using the "H" and "L"
|   commands for nets and "assign" for vectors
h exc_p
assign SW 00\h
| Initialize all flip-flops (preload- is active high
|   for 7k designs, you can abbreviate to prld)
h prld
| Viewsim uses units of 0.1 ns, so this statement
|   simulates for 100 ns.
cycle
l prld
cycle
```

Procedure for OrCAD Users

Reconfiguring the Libraries and Schematic Symbols

To reconfigure the libraries for the XC7000 family and update all the symbols so their properties are compatible with XC7000 devices, follow these steps:

1. Copy the CALC tutorial files to a directory under your home directory as described in the *OrCAD Interface User Guide*. Change (cd) to the calc\soln_3k directory.
2. While in your working directory, type the following at the DOS prompt:

   ```
   xdraft 7
   ```

Editing the Schematic

To edit the schematic so all its symbols are compatible with XC7000 devices, follow these steps:

1. Start up XDM. Select the **OrCAD** command from the **DesignEntry** menu.
2. Double click on the **Design Management Tools** button. Select **CALC** from the list on the left, then select the **OK** button.
3. Enter **Schematic Design Tools** and then **Draft**. The top-level schematic of the CALC design appears. If you see a message about an X being deleted, do not worry; this is an FPGA-specific property that you would have to delete anyway.
4. Change the PART= text to **7372-10PC68**.
5. Enter the OSC_3K sheet symbol using the **Quit** → **Enter Sheet** → **Enter** command.

   The OSC_3K schematic is displayed. There may be messages telling you that two components, ACLK and GCLK, are unavailable. The spaces where these two components were on the original schematic are left open, with unconnected nets.
6. Edit the schematic so it looks like Figure 2-3. The part of the schematic that is not visible is unchanged except for LOC= properties being removed.

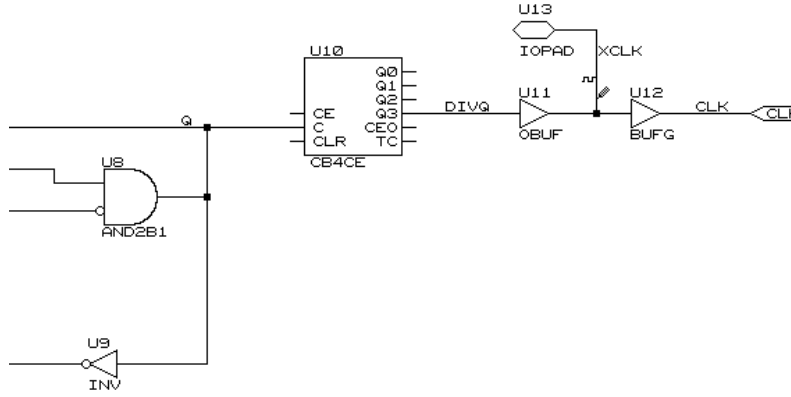


Figure 2-3 New OrCAD Oscillator Schematic for EPLD CALC

7. Be sure to label the new IOPAD net "XCLK".
8. Be sure to delete the LOC=... properties from the two IOPADs at the left end of the schematic. (Use the **Edit → Edit → 1st Part Field → Name** command.)
9. Select **Quit → Update File → Leave Sheet**.
10. Use the **Quit → Enter Sheet**, **Edit → Edit → 1st Part Field → Name**, and **Quit → Update File → Leave Sheet** commands to delete all the LOC= and FAST statements in the CALC schematic and all its sub-schematics.
11. When you are finished editing all the schematics, select **Quit → Update File → Abandon Edits** from the top-level schematic, double click on the **To Main** button, double click on the **Exit ESP** button, and press any key to return to XDM.

Performing Functional Simulation

To perform functional simulation, follow the instructions in the "VST Tutorial" chapter of the *OrCAD Interface User Guide*, with the exceptions in the following steps.

1. If you plan to perform timing simulation later, place stimulus and trace information on the XCLK signal in the OSC_3K schematic

instead of on the CLK signal, and on the LED_P signals in the LED schematic instead of on the STACK signals (pp. 12-6 to 12-8).

2. In addition, you will have to move the stimulus information in the SW7 schematic from the SW signals to the SW_P signals and subtract one clock cycle. Follow these steps:

   a) Place the mouse cursor on the stimulus symbol on the SW4 signal and select the **Inquire** command. The following line is displayed in the top left corner of the screen:

      ```
      Stimulus: 0:0 6000:1
      ```

   b) Subtract 500 time units, which is equal to one clock cycle, from the times at which the signal changes value. Replace all time values less than or equal to 500 with 0. In this case, 0 remains 0 and 6000 becomes 5500.

   c) Use the **Place → Stimulus** command to add the following stimulus to the SW4_P signal:

      0:0 5500:1

   d) Delete the stimulus indicator on the SW4 signal.

   e) Repeat steps a through d to delete all stimulus information from each SW signal and add it to each SW_P signal.

   f) Use **Quit → Update File** to save your edits.

   The EPLD fitter software optimizes away many internal nodes such as the CLK, STACK, and SW7 signals, but cannot optimize external signals such as the XCLK, LED_P, and SW_P signals. This does not affect functional simulation, but it does affect timing simulation.

3. Substitute “EPLD” wherever the VST Tutorial says “FPGA”. For example, select **Orcad_Epld_Func** as the XSIMMAKE flow name instead of Orcad_Fpga_Func.

4. Instead of adding a GR or GSR stimulus (p. 12-21), add a PRLD stimulus. Like GSR, the PRLD stimulus has an initial value of 1 and is brought down to 0 at 1 nanosecond.

Running the Fitter Commands

To map the design onto a target XC7000 device, follow these steps:

1. Select the **XEMake** command from the **Translate** menu in XDM.
2. Select **Done** to select all the default options.
3. Select **CALC.SCH** from the list of files.
4. Select **Make design database** as the target.

Performing Timing Simulation

To perform timing simulation, follow the instructions in the "VST Tutorial" chapter of the *OrCAD Interface User Guide*, with the exceptions in the following steps.

1. Place stimulus and trace information on the XCLK signal in the OSC_3K schematic instead of on the CLK signal, and on the LED_P signals in the LED schematic instead of on the STACK signals (pp. 12-6 to 12-8).
2. In addition, you will have to move the stimulus information in the SW7 schematic from the SW signals to the SW_P signals and subtract one clock cycle. Follow these steps:

   a) Place the mouse cursor on the stimulus symbol on the SW4 signal and select the **Inquire** command. The following line is displayed in the top left corner of the screen:

   ```
   Stimulus: 0:0 6000:1
   ```

   b) Subtract 500 time units, which is equal to one clock cycle, from the times at which the signal changes value. Replace all time values less than or equal to 500 with 0. In this case, 0 remains 0 and 6000 becomes 5500.

   c) Use the **Place** → **Stimulus** command to add the following stimulus to the SW4_P signal:

   ```
   0:0 5500:1
   ```

   d) Delete the stimulus indicator on the SW4 signal.

   e) Repeat steps a through d to delete all stimulus information from each SW signal and add it to each SW_P signal.

   f) Use **Quit** → **Update File** to save your edits.

The EPLD fitter software optimizes away many internal nodes such as the CLK, STACK, and SW7 signals, but cannot optimize external signals such as the XCLK, LED_P, and SW_P signals.

3. Substitute "EPLD" wherever the VST Tutorial says "FPGA". For example, select **`Orcad_Epld_Timing`** as the XSIMMAKE flow name instead of Orcad_Fpga_Timing.
4. Instead of adding a GR or GSR stimulus (p. 12-21), add a PRLD stimulus. Like GSR, the PRLD stimulus has an initial value of 1 and is brought down to 0 at 1 nanosecond.

Converting a Xilinx-ABEL Module (Optional)

You can use a Xilinx-ABEL module instead of the STATMACH schematic in the CALC design. The stat_abl.abl file specifies the logic. Substituting this Xilinx-ABEL module is described in the Interface User Guide for your CAE tool. This section describes how to make this module EPLD-compatible after you have substituted it.

1. Select the **`ABL2PLD`** command in the XDM menu. Select the **`stat_abl.abl`** file from the list. The ABL2PLD command creates a file named stat_abl.pld.
2. Start up your CAE tool.
3. Open the schematic named CONTROL. Delete the DEF=XABEL attribute from the STATMACH symbol, which referenced the Xilinx-ABEL module, and change FILE=stat_abl to PLD=stat_abl. For specific instructions on changing attributes, see the "Attributes" appendix.

Note: If you wish, you can convert the stat_abl.abl file from symbolic (one-hot) encoding to maximal encoding. It is not really necessary, however, because this state machine has only three states, and therefore the conversion spares only one register.

Chapter 3

EPLD Architecture and Design Tradeoffs

This chapter discusses EPLD architecture and tradeoffs in fitting your design to the EPLD architecture: designing for speed, density, or pinout preservation; controlling power consumption; and controlling preload values.

The tips and techniques in this chapter are guidelines only. They are general principles that work in most cases. They may or may not be applicable to a particular design.

EPLD Architecture

EPLD devices have special architectural features that can make your design faster and more efficient. The XEPLD fitter software automatically analyzes your design, optimizes the logic, and maps functions into the appropriate device resources. However, an understanding of the EPLD architecture can help you exercise complete control of design optimization.

For more detailed information about EPLD architecture, refer to the EPLD device data sheets.

Figure 3-1 is a simplified diagram of the XC7354 device that shows the main architectural features of EPLD devices.

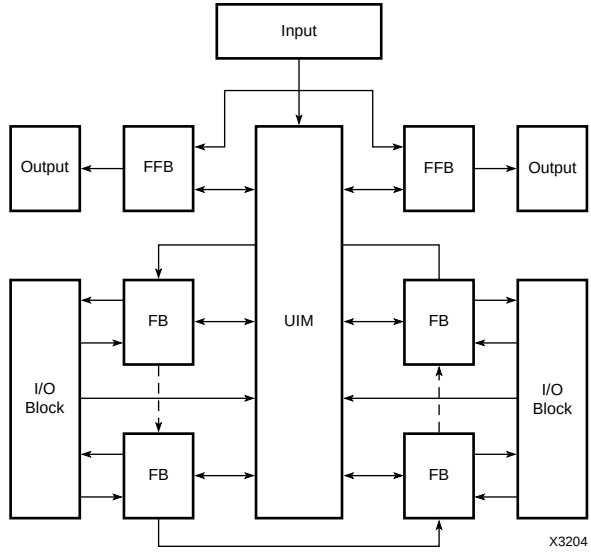


Figure 3-1 EPLD Device Structure

The five basic architectural features in an EPLD device are as follows:

- Input Pads
- Output Pads
- High Density Function Blocks (HDFBs)
- Fast Function Blocks (FFBs)
- The Universal Interconnection Matrix (UIM)

This section describes these features and how designs are mapped onto them for best results.

Note: For a complete explanation of the XC7000 architectural features, see the EPLD device data sheets.

Input Pad Structures

The XC7000 devices have two types of input pads: Fast Inputs and standard inputs.

Fast Input pins have two paths through the device. One path drives directly into the Fast Function Blocks, bypassing the UIM, and is used for signals that require the fastest propagation delays and shortest macrocell register setup times.The second path drives all function blocks (both FFBs and HDFBs) through the UIM.

Standard inputs and UIM paths of Fast Inputs can be configured as follows:

- Registered
- Registered with clock enable
- Latched
- Combinatorial

Registering and/or latching signals at the input pad shortens register setup times and is used most often to pipeline data on-chip or synchronize asynchronous inputs. The input pad registers can also store data, making more macrocells available for implementing logic.

Output Pad Structures

The XC7000 devices have two types of output pads: those driven by HDFBs, which have standard drive capability; and those driven by FFBs, which have higher drive capability. These outputs can be configured as follows:

- 3-state with individual p-term control (HDFB only)
- 3-state with FOE and individual p-term control (HDFB only)
- 3-state with FOE control
- Direct (always on)

Each output pad driven by a HDFB can be 3-stated by its own macrocell product term for maximum flexibility. The global FOE net offers maximum speed.

Bidirectional pins have both input pad structures and output pad structures. See your device data sheet for details.

High-Density Function Blocks

High Density Function Blocks provide the maximum amount of logic resources for use in your design. They are well-suited for arithmetic functions, counters, and other kinds of complex logic.

High Density Function Blocks contain special fast carry lines for arithmetic logic. These lines extend between High-Density Function Blocks, allowing fast carry for very large arithmetic functions.

Fast Function Blocks

The XC7300-series devices have a combination of High Density Function Blocks and Fast Function Blocks; this is called "Dual Block Architecture."

Logic placed in Fast Function Blocks performs faster than logic in High Density Function Blocks. Fast Function Blocks are well-suited for critical decoding and ultra-fast state machine applications.

The Universal Interconnect Matrix (UIM)

The Universal Interconnect Matrix, or UIM™, provides a 100% interconnection matrix allowing any function block output to drive any function block input in the device; routing is never blocked. All function block inputs (except for the FastInputs) come from the UIM.

The UIM can perform wired-AND functions, which the software uses automatically when possible to improve resource utilization.

Designing for Speed

To optimize for speed (faster pin-to-pin and register setup times), follow these guidelines:

- Use Fast Function Blocks for the functions in which speed is most critical.
- Use input pad registers.
- Use EPLD-specific arithmetic functions when cascading.
- Use EPLD-specific bidirectional counters when cascading.
- Reduce levels of logic.

Assigning Logic to Fast Function Blocks

Fast Function Blocks (FFBs), which are available in XC7300 devices, have Fast Inputs and Fast Outputs, which bypass the UIM and thus do not incur a UIM delay. Their logic is simpler than that of High-Density Function Blocks (HDFBs), which also means less delay. Use FFBs for critical signals and functions in which speed is important.

To place logic in an FFB, use the F attribute on the outputs. Use the F attribute on inputs (the outputs of IBUF components) to make them Fast Inputs. For more about the F attribute, see the "Attributes" appendix.

Turning off preload optimization affects whether logic can be moved into Fast Function Blocks. For more information, see the "Controlling Preload Values" section in this chapter.

Logic Requirements for Fast Function Blocks

The XEPLD software automatically maps into the Fast Function Blocks any function that meets these requirements, even if you do not use the F attribute:

- All clocks use the global FastCLK™ signals, which means that the global clocks must be available.
- All 3-state controls use the global FOE signals.
- All registers may only be asynchronously set except in 7336 and 7318 devices (unless you use reset emulation, described in the "Design Applications" chapter).
- All registers may only be preloaded to a logic High state or have unspecified preload values except in 7336 and 7318 devices (unless you use reset emulation, described in the "Design Applications" chapter).
- All logic must use four or less p-terms when implemented as active-Low except in 7336 and 7318 devices. You can override this requirement by using the F attribute — the software exports product terms if the output uses more than four product terms.
- All components used must be allowable in a Fast Function Block.

Use the F attribute in these cases:

- If you want a function to have priority for being placed in an FFB.

- If you have a function with more than four product terms that you would like to place in a Fast Function Block.

If your logic output signals must use an internal p-term clock, the software drives the p-term clock off-chip through a FastCLK pin and back into the global FastCLK net through the I/O buffer on the FastCLK pin.

If your registers require asynchronous reset inputs or if the preload state must be a logic Low, your design will need modification in order to fit into an FFB. See the "Reset and Preload Control in FFB and Input Pad Registers" section in the "Design Applications" chapter.

To place component outputs that do not drive anything in FFBs, add dangling nets to the outputs and apply the F attribute to the dangling nets as shown in Figure 3-2.

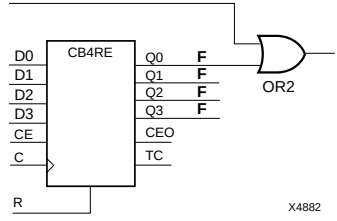


Figure 3-2 Assigning Dangling Outputs to Fast Function Blocks

Note: When placing functions into Fast Function Blocks, it is best to choose functions that require the least number of product terms.

Components Not Allowed in Fast Function Blocks

The following components are not allowed in a Fast Function Block because they require special features, such as arithmetic carry lines, that are not present in Fast Function Blocks.

- PLFB9
- ADD symbols
- ADSU symbols
- ACC symbols
- BUFCE

- IFD
- IFDX1
- ILD
- COMPM
- LD
- FDCP, FDCPE (If you use a flip-flop with both CLR and PRE, you must tie either CLR or PRE to GND)
- OBUFT
- OFDT
- XOR7, XOR8, XOR9 (cascade the smaller XORs instead)

Note: The BUFT and BUFE symbols are allowed for external outputs only (not nodes) and must allow FOE optimization.

Using Input Pad Registers

Input pad registers and latches offer these advantages:

- Faster setup-to-clock time than macrocell registers
- Additional storage for register-intensive designs

Figure 3-3 shows the components you use to specify a regular macrocell register, an input pad register, an input pad latch, and an input pad register with global clock enable.

Note: The XEPLD software automatically maps FDs and FD variations to input pad registers when possible. You need only use IFDs and variations for more direct control of the mapping, for latching (use ILD or a variation), or for global clock enable (use IFDX1).

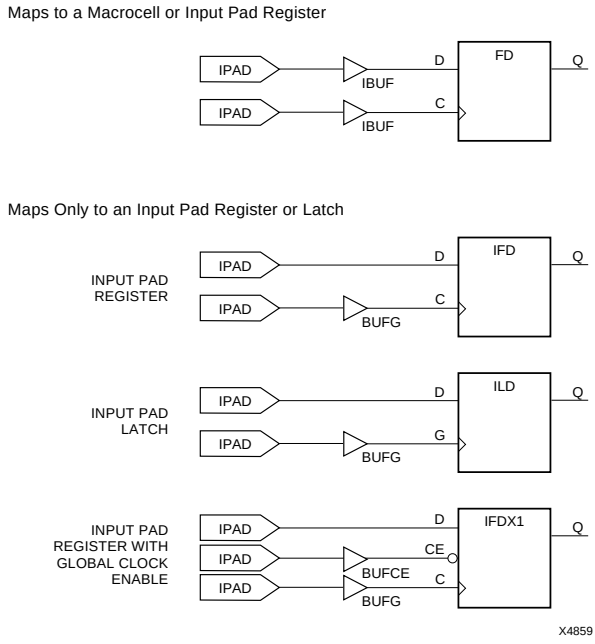


Figure 3-3 Input Pad Registers

Placing Clock Enable Signals in Input Pad Registers

If you want to use the global clock enable net and an input pad register, but your clock enable signal must be controlled by internal logic, use the BUFCE and IFDX1 components as shown in Figure 3-4. The FDCE shown in Figure 3-5 is valid, but it does not use the global clock enable net or an input register.

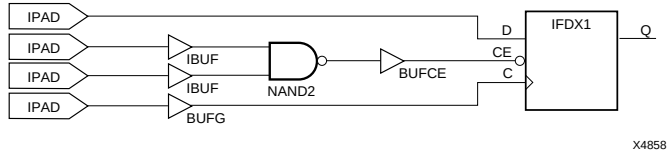


Figure 3-4 Can Be Placed in an Input Pad Register

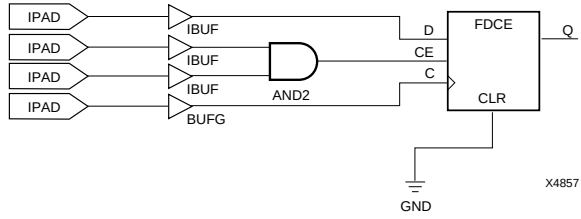


Figure 3-5 Cannot Be Placed in an Input Pad Register

Using EPLD-Specific Arithmetic Functions

XC7000-specific arithmetic components use the fast carry chain for their CI and CO pins. Equivalent common components do not use the fast carry chain for their CI and CO pins and therefore slower and use more device resources when cascaded. If you are not cascading, however, the common arithmetic components work well.

Cascading Counters

If you are cascading bidirectional or down counters, you should use XC7000-specific counters. These counters have separate up and down terminal counts (CEOU and CEOD) that can be cascaded in the UIM. The up terminal count is generated by ANDing all of the counter bits in the UIM. The down terminal count lookahead is generated in a macrocell. These terminal counts are then ANDed with the count enable inputs (CEU and CED) to produce the component's up and down terminal count outputs as shown in Figure 3-6.

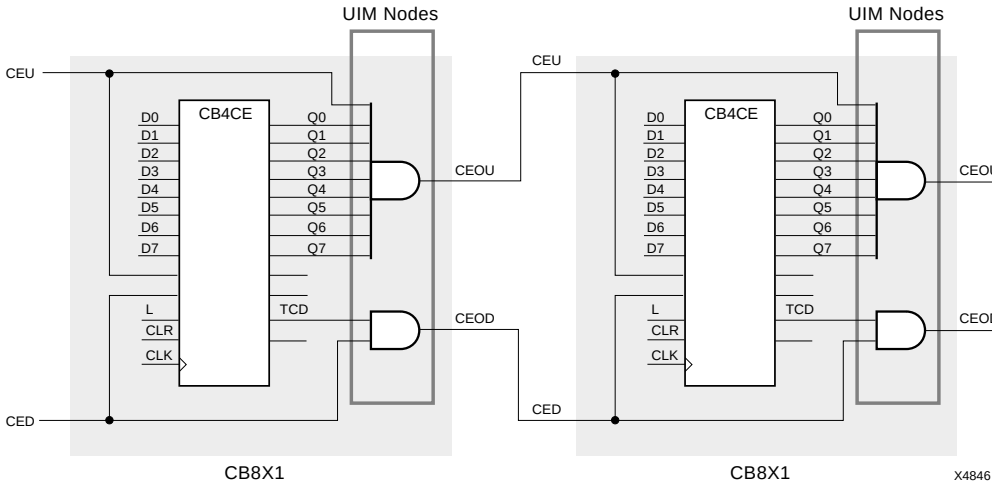


Figure 3-6 Cascading EPLD-Specific Up/Down Counters

Because the XEPLD optimization software collapses the cascaded UIM nodes into a single level of logic, the speed of the cascaded counter is constant, no matter how many bits it has. However, in common library bidirectional counters, the up and down terminal counts are combined into a single terminal count. This terminal count uses both the true and the complement of the counter bits, which makes the terminal count impossible to place in the UIM.

Reducing Levels of Logic

Each EPLD macrocell has several levels of logic followed by a register. If you put the logic first, the XEPLD fitter software maps the logic and register into the same macrocell. If you put logic after the registers, however, the XEPLD fitter software may use additional macrocells for the logic that follows the registers, decreasing both the speed and density of your design. Figure 3-7 shows two equivalent circuits, one that is efficient and one that is inefficient.

Note: There are exceptions to this guideline. For an example, see the following “Splitting Wide Functions” section.

Another way to minimize levels of logic is to make sure you do not exceed 21 inputs or 17 product terms for a function. This makes it likely that the function will fit in one HDFB macrocell.

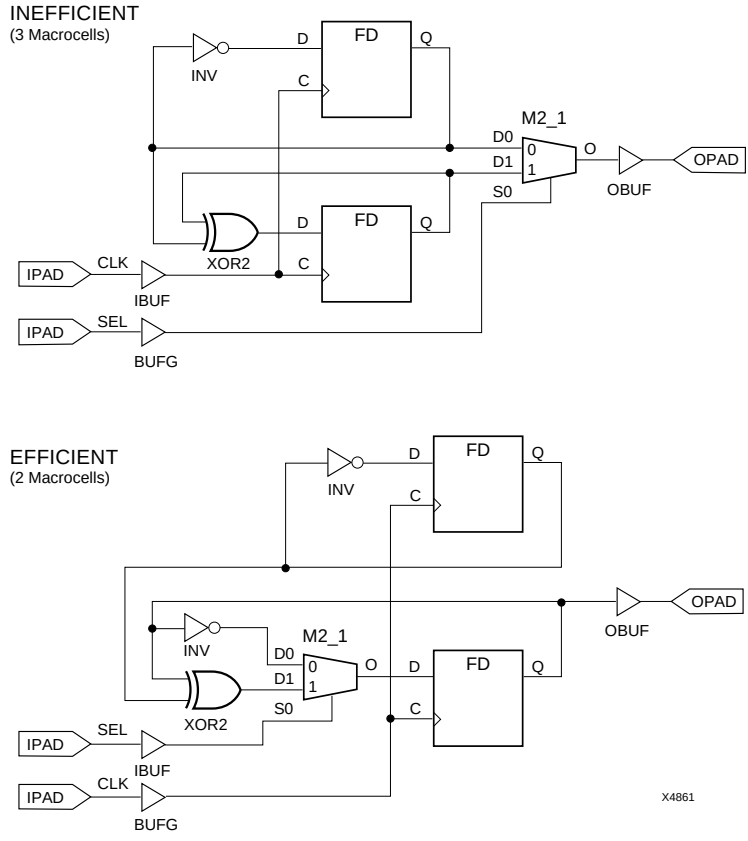


Figure 3-7 Reducing Levels of Logic

Splitting Wide Functions

The XEPLD software can handle a function with up to 17 product terms and 21 inputs as a single level of logic in a High-Density Function Block. If the function exceeds these limits, it is implemented as two levels of logic and an extra delay occurs. This typically happens with wide compare functions, since two product terms are required for each pair of bits being compared.

You can keep your wide function implemented as a single level of logic if you manually split the function.

Figure 3-8 shows a design with a wide compare function that is twice as fast after splitting.

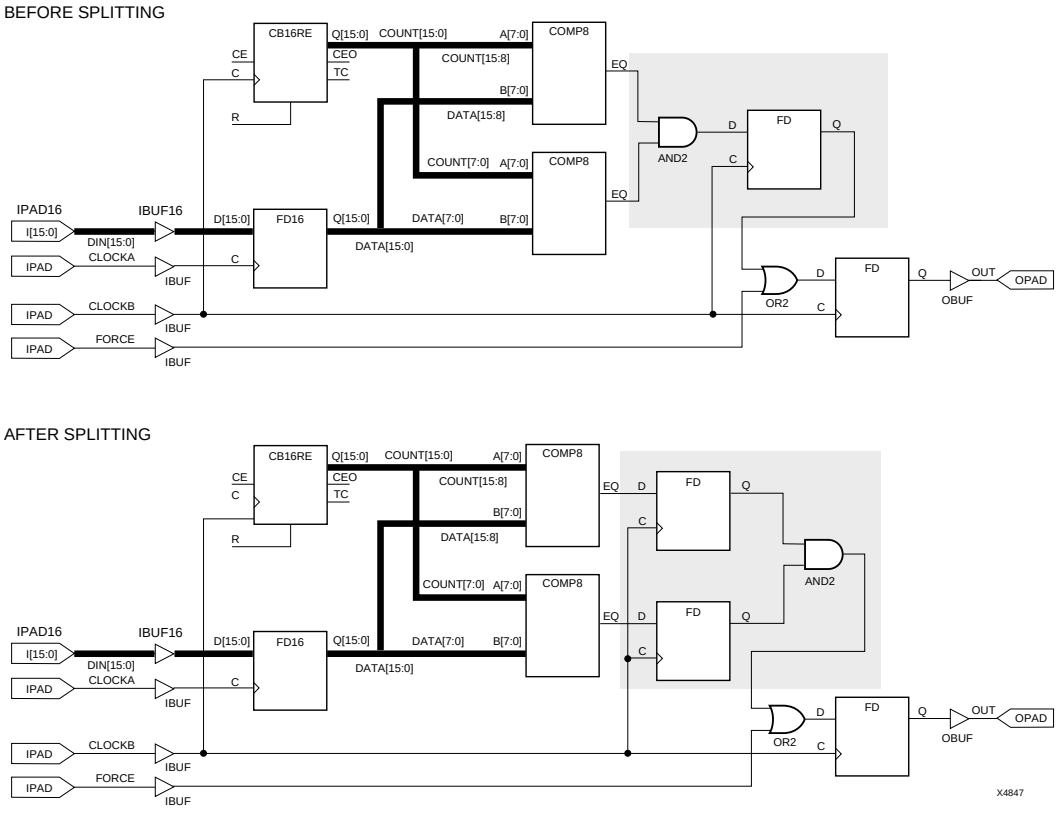


Figure 3-8 Splitting a Wide Function

Random Logic

If your design contains custom gate-level logic, follow these guidelines to ensure that the logic will optimize well:

- Try not to exceed 21 inputs or 17 product terms for a single output or registered function. Logic must meet these requirements to fit within a single High-Density Function Block macrocell.

- For a Fast Function Block macrocell, the limit is 24 inputs. If a function exceeds 4 product terms, the product terms are exported from adjacent macrocells, but this means that a wide function uses up many macrocells.
- In general, use library components instead of gate-level logic wherever possible.
- If random logic is represented in many layers of combinatorial logic before it reaches pads or registers, it may be better to use a behavioral module.

The EPLD software will attempt to reduce the number of inputs to 21 or less so that the logic can be implemented in a single pass through the UIM. However, this may be more difficult if the design is implemented with large numbers of random gates.

Designing for Density

The XEPLD software optimizes for density by default, but there are some additional things you can do to improve density optimization:

- Maximally encode all state machines.
- Specify active-High output enables and rising edge clocks so these signals can be mapped to global nets.
- Specify UIM nodes for wide input AND functions.
- Use input pad registers whenever possible to make more macrocells available for logic.
- Use the UIM paths of Fast Inputs.
- Turn off logic optimization on selected combinatorial nodes.
- Use the Master Reset pin if your design requires device reinitialization, or use this pin as a regular input if your design requires an additional I/O pin.

Maximally Encoding State Machines

If your design contains behavioral modules written as state machines, be sure that the functions are maximally encoded. This works best for EPLDs, which are rich in product terms. (For FPGAs, it is best to use one-hot encoding, because FPGAs are rich in registers.)

Using Global Nets

Clock and output enable signals mapped to global nets do not consume function block resources. Optimization software automatically maps the most used rising edge clock inputs to the FastCLK nets and the active-High output enable inputs to the FOE nets.

Moving Logic into the Universal Interconnect Matrix

In addition to offering 100% routability, the UIM can function as a very wide AND gate. The UIM can also implement DeMorgan equivalent functions. The following figure illustrates this process:

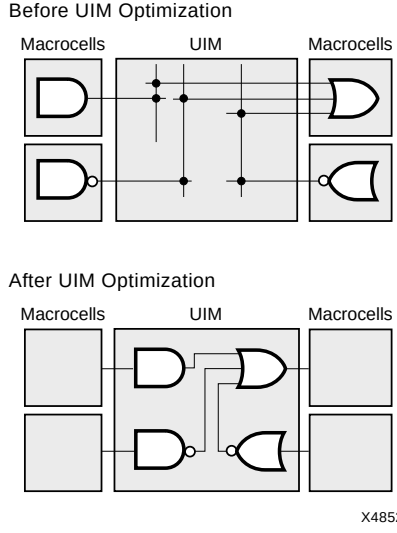


Figure 3-9 Moving Functions into the UIM

Sometimes specifying a UIM node for a wide ANDing function can result in improved optimization. Moving a node into the UIM can free up additional function block inputs for use by other logic functions. This should only be tried after reviewing the results of the automatic optimization. See the description of the OPT=UIM attribute in the “Attributes” appendix for more information.

Using Input Pad Registers

Registering or latching signals at the input pad shortens register setup times and is often used to pipeline data on-chip or synchronize asynchronous inputs. The input pad registers can also store data, making more macrocells available for implementing logic. You can even emulate asynchronous set/reset control using a design technique outlined in the "Reset and Preload Control in Input Pad and FFB Registers" section in the "Design Applications" chapter.

Macrocell Register vs. I/O Pin Tradeoff

If your design is register-intensive but requires few I/O pins, you may be able to trade a few macrocell registers for I/O pins. A register having only a buffer or inverter in front of its D input can be converted to an input register. Use the method shown in Figure 3-10.

For clarity, IPAD, IBUF, and BUFG symbols are omitted from the inputs on the left, and OBUF and OPAD from the output on the right.

Note: Because IFDs preload to 1, you may see 1s propagate through your design at the beginning of simulation. To prevent this, you can use the technique described in "Reset and Preload Control in FFB and Input Pad Registers" in the "Design Applications" chapter.

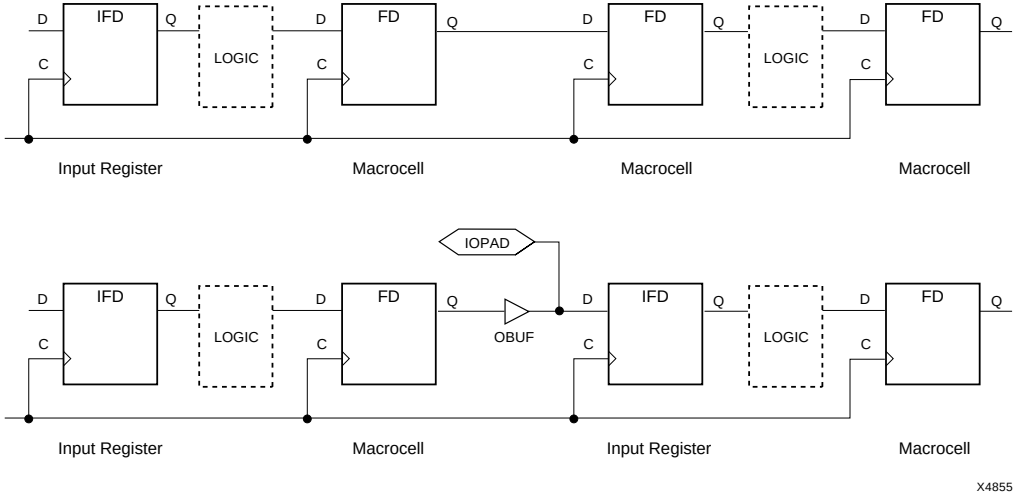


Figure 3-10 Using a Macrocell vs. Using an I/O Pin

UIM Versus Fast Input Paths

A second path into the Fast Function Block in available through the UIM. If the Fast Function Block becomes input limited, moving a signal from the direct Fast Input path to the UIM path may allow the optimization software to free up an extra FB input.

To move a signal from the direct Fast Input path to the UIM path, remove the F attribute from the input signal. Do not remove the F attribute from the output signal, however, because this is what places the logic in a Fast Function Block.

Controlling Logic Optimization

The XEPLD software attempts to reduce the number of logic levels for all signal paths. This means that the remaining logic functions have a wider signal fan-in and possibly require more product terms than if some combinatorial nodes were retained.

By turning off logic optimization on selected nodes, you may free up additional function block inputs and/or macrocell product terms, at the expense of the macrocells used to implement the nodes. Try this only after reviewing the results of the automatic optimization. See the "Attributes" appendix for details about LOGIC_OPT and OPT.

Master Reset Pin Tradeoffs

This discussion describes tradeoffs of using the XC7000 MR pin as a global reset and 3-state control. Consult your device data sheet for specific requirements of the MR pin during power-up. See the "Controlling Preload Values" section at the end of this chapter for the default preload values of device registers.

The XC7000 devices feature a master reset pin that can be used to reinitialize the device. When the device is reinitialized, all device pins are 3-stated and registers are preloaded. When initialization is complete, the register preload is released and the outputs become operational.

The master reset pin can completely 3-state the device during board testing. It can also force state machines and registers to a known state if the reinitialization delay is not critical in your design (see the device datasheet for details). The advantage is that no product terms or Function Block inputs are required to preload the registers or

3-state the device pins. If the reinitialization delay is critical, use a logic input to force state machines and registers to a known state.

On the 7354, 7336, and 7318 devices, the master reset pin can also be programmed as a logic pin by assigning the MRINPUT=ON global attribute to a TBLOCK symbol. If this attribute is specified, the device is initialized only on power-up.

Designing to Preserve the Pinout

In the XC7000 devices, logic capacity and device pinout are determined only by the resources available in the Function Blocks; the logic mapped into the Function Blocks is always guaranteed to route in the UIM. The factors that determine the logic capacity of the Function Block are as follows:

- Number of Function Block inputs
- Number of product terms available to each macrocell in the Function Block
- Number of macrocells in the Function Block
- Number of device pins driven by the Function Block

You can reserve High Density Function Block resources by adding a "filler" circuit to your design as shown in Figure 3-11. This circuit consumes two UIM inputs, one shared product term, and one macrocell for each output used in the design. It can be used to reserve resources in up to 10 Function Blocks. Each output is assigned to a specific Function Block by assigning it to a specific device pin with the LOC attribute. The equation file, filler.pld, is in the xact\examples\behavior\epld directory.

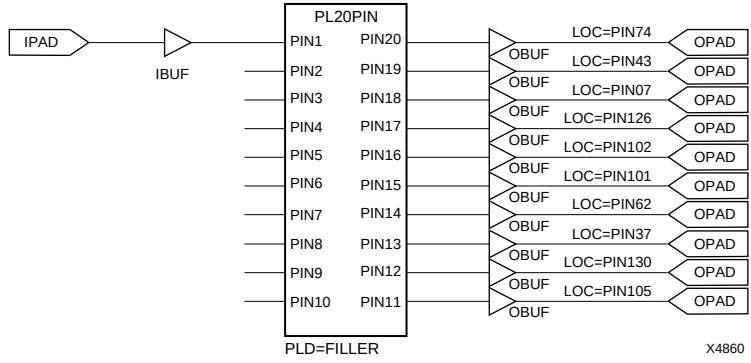


Figure 3-11 Filler Circuit

Note: If more than one output is assigned to the same Function Block, the first output consumes two UIM inputs. Each additional output consumes one additional UIM input.

You should evaluate the product term requirements of your logic assigned to pins that are driven by Fast Function Blocks. Check the design.eqn file to determine how many product terms (including the implied exported product terms) are required. Plan ahead for design iterations which may create functions that require the exported product terms from an adjacent macrocell.

To preserve the pinout from an earlier iteration of the design, use the **`Translate` → `PinSave`** command to generate the .VMF pin freeze file. Then use the **`Profile` → `Options` → `FitNet` → `-f`** command to turn pin freezing on for the design. Save the profile with the **`Profile` → `Save Profile`** command. Now, each time you recompile the design, the old pinout will be reused.

Manual Pin Assignment

Note: Manual pin assignment can restrict the layout capability of the software. It is usually best to allow XEPLD to automatically assign pins based on the most efficient placement of logic in the device.

XEPLD automatically assigns device pins for you, based on the most efficient usage of device resources. This is usually the best method for pin assignment if you do not have specific pinout requirements.

Automatic pin assignment is performed only for those pins that have not been assigned through some other method. After a successful design compilation, you can use the PinSave command to maintain the pin assignments during design iteration.

If you have specific pinout requirements you can use the LOC=*pin_number* attribute to assign the signal to a specific pin.

Note: LOC attributes override the pin assignments in the pin-save file. This allows you to make changes to your fixed pin specifications. However, if you override the pin-save file with LOC attributes, the software will issue a warning.

Manual Pin Assignment Precautions

When you manually assign output and I/O pins, you force the software to place logic functions into specific function blocks. If the logic does not exceed the function block resources (macrocells, product terms, and UIM inputs) and the function block has the correct external pin resources to meet the logic I/O requirements, the logic is mapped into the function block and the design will route in the UIM.

Try to place product term intensive logic onto pins that are driven by High Density Function Blocks. Be sure that the Function Block's shared product term resources and UIM inputs will not be exhausted. You may also wish to leave additional room in the Function Block for design iterations.

Assign your external rising-edge clocks and active-High output enable signals to the FastCLK and FOE pins on the device. To create global on-chip clocks, assign them to the FastCLK nets. To create global output enable control signals, assign them to the FOE nets. These signals will use the I/O buffer on the pin to route the macrocell output onto the global net.

Evaluate the requirements of your logic assigned to pins that are driven by the Fast Function Blocks. Functions mapped to an FFB can be clocked only by global clocks, 3-stated only by FOEs, and, for the 7354, 7372, and 73108 devices, only asynchronously set. Plan ahead for design iterations which may create functions that require the exported product terms from an adjacent macrocell.

The LOC Attribute

Use the LOC=*pin_name* attribute on a PAD symbol to assign the signal to a specific pin. The pin name is P*nn* for PC packages; the *nn* is a pin number. The pin name is *rc* (rowcolumn) for PG packages. Examples are LOC=P24 and LOC=G2.

You can apply the LOC attribute to as many PADs in your design as you like. However, each pin assignment further constrains the software as it automatically allocates logic and I/O resources to internal nodes and I/O pins with no LOC attributes.

Note: Pin assignment using the LOC attribute is not supported for bus components such as OBUF8.

Controlling Power Consumption

You control power consumption for specific macrocell outputs using the LOWPWR attribute. This attribute is valid only for XC7300 designs. This attribute is either a global or component attribute.

The default is LOWPWR=OFF (high speed) for all macrocells used in the design unless otherwise specified.

To make low power the global default power setting, place the global attribute LOWPWR=ALL in the schematic. (See the Global Attributes section of the "Attributes" appendix for instructions.)

To control the power setting of the macrocells used by an individual symbol, use LOWPWR=ON or LOWPWR=OFF (if the global LOWPWR=ALL was used). This attribute is ignored if assigned to a symbol that uses no macrocells, such as an inverter.

Note: Low-power outputs are slower than regular-power outputs. If you have a mixture of low- and regular-power outputs, pay close attention to simulation results or the timing report to see how the power settings affect timing interactions.

Controlling Preload Values

The preload values used in the implementation of your design depend on these factors:

- The register resources of the target device

- The preload values of the library components used in the design. Every registered component in the Xilinx library has a default preload value defined; for most components, it is 0. You can look up preload values for any component in the *XACT Libraries Guide*.
- The .PRLD equations in the behavioral modules
- Whether preload optimization is on. By default, the XEPLD fitter performs preload optimization, ignoring the library defaults and .PRLD equations, to produce the most efficient mapping of components to available device resources. You can control preload optimization using the PRELOAD_OPT and INIT attributes.

Physical Resources of EPLDs

Registers in EPLD devices can physically support the following initial preload states:

- Input registers and latches in 7272 devices have no predetermined preload value.
- Input registers and latches in non-7272 devices always have a preload value of 1.
- Macrocell registers in 7336/7318 Fast Function Blocks have a preload value that depends on the use of the asynchronous Set/Reset product term. When this product term is defined as Reset, the register's preload value is 0; when Set, the value is 1. If no Set/Reset product term is specified, the default is 1.
- Macrocell registers in non-7336/7318 Fast Function Blocks always have a preload value of 1, but if the internal logic in these macrocells is implemented as negative logic, the apparent preload value will be 0.
- Macrocell registers in High-Density Function Blocks can support preload values of 0 or 1. If the preload value is not specified, the default is 0.

Attributes for Controlling Preload Values

If your design is not sensitive to preload values, it is best to allow preload optimization, because this produces efficient results. However, if you want to control register preload states, you can prevent preload optimization in these ways:

- Use the PRELOAD_OPT=OFF global attribute. This turns off preload optimization for all registers in the design. The fitter is forced to obey all library defaults and .PRLD equations.
- Use the INIT=S (preload=1) and INIT=R (preload=0) attributes to specify the preload values of individual registers. The INIT= attributes are always obeyed by the fitter, regardless of the PRELOAD_OPT attribute, but are ignored in functional simulation.

Note: You cannot change the preload value of an input register to 0 using the INIT=R attribute because input registers physically do not support preload to the 0 state. Also, if you specify PRELOAD_OPT=OFF or INIT=R to control preload values, it prevents registers from mapping into FFBs, and attempting to force such a register into an FFB (using the F attribute or through pin assignment) results in an error.

Preload Values for Functional and Timing Simulation

The only functional differences expected between functional and timing simulation involve the initial states of registers and latches in the design. Functional simulation assumes that preload values are as defined in the library components and the .PRLD equations in behavioral modules. Timing simulation uses the actual preload values implemented by the fitter.

When functional and timing simulation yield different results, it is probably because the XEPLD fitter did not use the library default and .PRLD equation values due to the preload optimization feature.

For example, if an FDR component is mapped to a Fast Function Block, the FDR will appear to preload to 0 during functional simulation, because that is how the library component is defined. However, during timing simulation, this FDR will actually preload to 1, because that is physical preload state of an FFB macrocell register where the FD component was mapped.

Chapter 4

Design Applications

This chapter describes some of the most useful techniques for making your EPLD design more efficient. These examples are suggestions and guidelines only, and may not apply to your particular design.

Reset and Preload Control in FFB and Input Pad Registers

Use the following reset emulation technique to do these things:

- Emulate reset or clear when using Fast Function Block registers
- Emulate reset or clear when using input pad registers (sparing macrocell registers)
- Change preload values in input pad or FFB registers to 0 (the preload value for these registers is normally 1)

Figure 4-1 shows how to set up reset emulation. An additional macrocell register from a High-Density Function Block provides the reset input. You AND the output of this HDFB register with the outputs of the FFB or input pad registers. These AND gates end up in the UIM, so there is no additional delay.

The XEPLD software automatically tries to map FD-type registers into input pad registers or FFB macrocell registers before it maps to HDFB macrocell registers. However, if you want to explicitly specify input pad registers, use IFD-type registers. To explicitly specify FFB registers, use the F attribute on the register outputs (see the "Attributes" appendix for more about the F attribute).

Note: If you are changing preload values, you do not need an HDFB macrocell register with a reset; a simple FD will work.

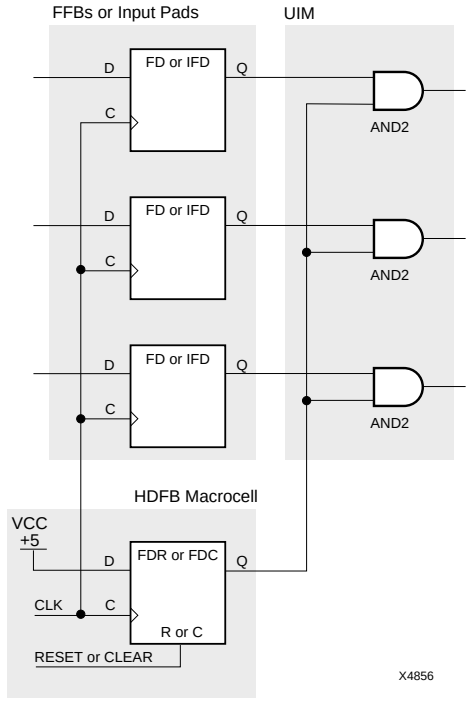


Figure 4-1 Reset and Clear Emulation for FFB or Input Registers

Read-Back Registers

Figure 4-4 shows a simple read-back register. Data is written from the IOPAD to the register on the rising edge of the clock if READ_ENABLE is inactive and WRITE_ENABLE is active. Data is read from the IOPAD when READ_ENABLE is active.

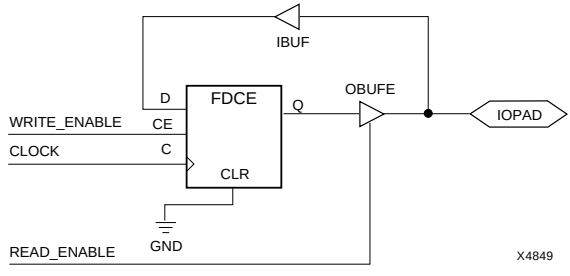


Figure 4-2 Read-Back Register Example

Bidirectional Signals and Buses

Figure 4-2A shows how to specify a bidirectional pin. Figure 4-2B shows that you can have a bidirectional signal passing through the chip. To make a bidirectional bus, use bus components as shown in Figure 4-2C.

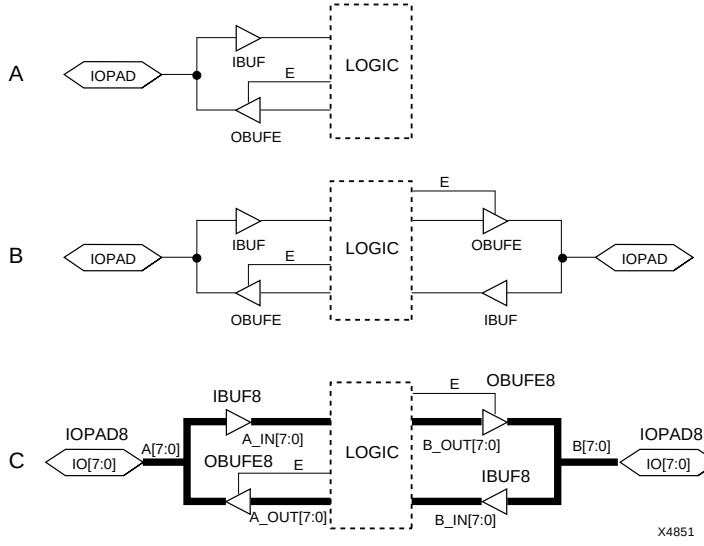


Figure 4-3 Bidirectional Signals and Buses

Bidirectional Signals in PLDs

If you want to use a PLD output with a TRST equation to control a bidirectional I/O pin of the EPLD, connect the OBUF output to an IOPAD and IBUF (or IFD/ILD). If the same PLD symbol that generates the output is also to receive the I/O pin input, you must use a separate pin of the PLD symbol to receive the signal from the IBUF. Do not tie the signal received from an IBUF to the net driving the OBUF of the same IOPAD as shown in Figure 4-3A; these input and output nets must remain separate as shown in Figure 4-3B.

Rules for connecting PLD symbols also apply to any custom symbols defined by equation files or macro schematics.

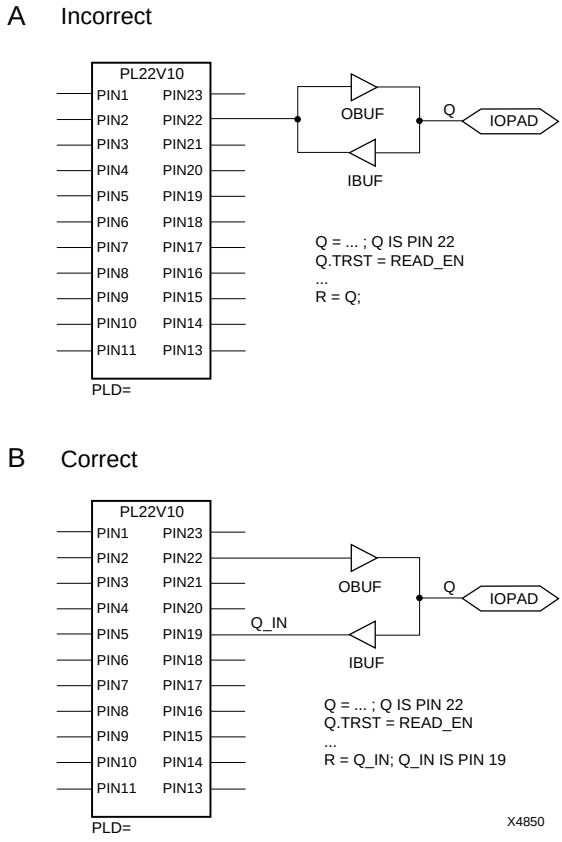


Figure 4-4 How to Control a Bidirectional PLD Pin

Multiplexing 3-State Signals

Three methods of multiplexing 3-state signals are shown in Figure 4-5 on the next page. Which method you choose depends on your application, resources, and speed requirements, although method C, which uses a multiplexer, is usually best for EPLD designs.

Method A, shown in Figure 4-5A, uses 3-state buffers instead of a multiplexer. The advantage of method A over method C is that method A uses only one Function Block input in the macrocell that sends the signal off-chip. The disadvantage of method A is that macrocell feedback is lost because the outputs are 3-stated; therefore counters will not work with Method A, but will work with Method C.

Method B, shown in Figure 4-5B, requires that you tie the signals together off-chip. This method results in a short clock-to-out delay and uses fewer macrocells than methods A and C. However, it uses more pins than method A or C.

Method C, shown in Figure 4-5C, uses a multiplexer instead of 3-state buffers. This method results in a longer clock-to-out delay than method B, although you can shorten this delay to that of method B by registering the output of the multiplexer and asserting the select signals one clock cycle in advance. This method uses more macrocells than method B, but uses fewer pins.

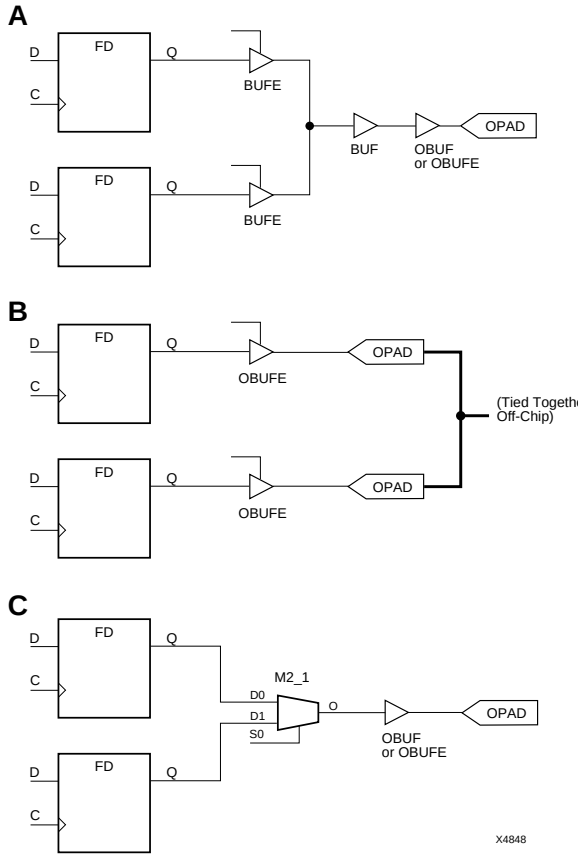


Figure 4-5 Methods of Multiplexing 3-State Signals

Optimizing Registered Arithmetic Performance

The XEPLD software optimizes adders and subtractors into FD, FDC, and FDP registers. If your arithmetic component drives any other register type, the arithmetic and register functions are implemented in separate macrocells, impacting both speed and density.

The example in Figure 4-6 shows an adder/subtractor driving a register with clock enable and synchronous clear. When the logic in these components is broken down, each bitslice is represented as shown in Figure 4-7.

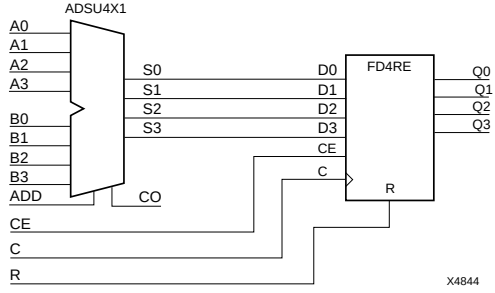


Figure 4-6 Using ADSU4X1 and FD4RE Library Components

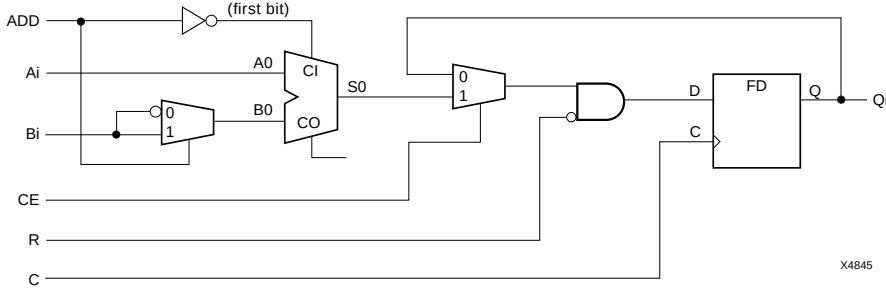


Figure 4-7 ADSU4X1 and FD4RE Equivalent Logic

In EPLD High-Density Function Blocks, the arithmetic logic physically occurs just before the register, as shown in Figure 4-8. This means that, because of the reset and clock enable on the register, the ADSU4X1 and FD4RE are implemented as two levels of logic in an EPLD device.

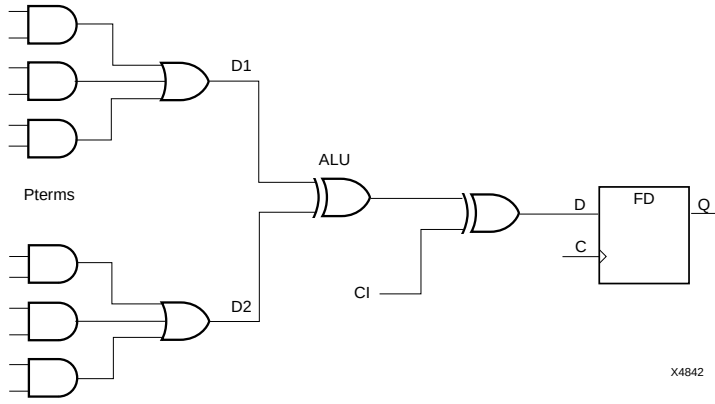


Figure 4-8 EPLD High-Density Function Block Architecture

Because the logic in the adder must be performed in the ALU block of the macrocell, the fitter cannot collapse the logic in Figure 4-7 into the same macrocell. As a result, the logic formation requires two macrocells and two macrocell delays.

You can achieve more efficient results by placing the register's reset and clock enable logic in front of the arithmetic logic as shown in Figure 4-9.

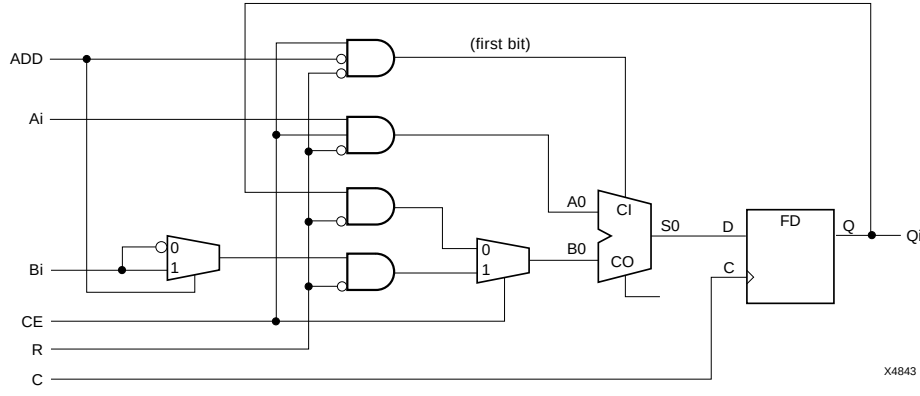


Figure 4-9 ADSUR4 Custom Symbol Logic Implementation

You can optimize the speed and density of the design in this way by modifying an existing arithmetic component equation file, then linking it to a custom symbol in the schematic.

Implement the clock enable circuit using the XC7000 macrocell's local feedback path. Recirculating the register output through this path saves UIM inputs. Use a .FBK equation for each sum bit in the equation file. Whenever the .FBK equation is TRUE, the register feedback is ORed into the macrocell's D2 sum of products. (See the HDFB macrocell schematic for details.)

To implement the synchronous clear, mask each operand and .FBK equation with the complement of R. When R is TRUE, all inputs to the ALU are zeroed and the registers are cleared on the rising edge of the clock.

To modify the ADSU4X1 equation file, follow these steps:

1. Copy the equation file ADSU4X1.PLD from the xact\examples \behavior\library directory to your design directory.
2. Rename the file to ADSU4X1R.PLD, and change the symbol name in the CHIP statement of the file to ADSU4X1R.
3. Add the C, R, and CE signals to the pinlist.
4. Change the equations for the CIN signal:

Original Equations	Modified Equations
cin.d1 = /add	cin.d1 = /add * ce * /r
cin.d2 = /add	cin.d2 = /add * ce * /r
cin = cin.d1 gnd cin.d2	cin = cin.d1 gnd cin.d2

5. Change the equations for each sum bit of the adder:

Original Equations	Modified Equations
s0.d1 = b0 * add	s0.d1 = b0 * add * ce * /r
+ /b0 * /add	+ /b0 * /add * ce * /r
s0.d2 = a0	s0.d2 = a0 * ce * /r
s0 = s0.d1 xor s0.d2	s0 := s0.d1 xor s0.d2
s0.add = vcc	s0.add = vcc
	s0.fbk = /ce * /r
	s0.clkf = c

The entire equation file follows. For how to create a custom component, see the "Using Behavioral Modules in Schematics" chapter of this manual.

```
TITLE           Registered Add/sub: 4-bit, clock-enable, synch reset
CHIP            ADSU4X1R COMPONENT

;Inputs
a0 a1 a2 a3                 ; adder A-operand
b0 b1 b2 b3                 ; adder B-operand
add                         ; function select: 1=add, 0=subtract(A-
B)
c                           ; clock (rising edge)
ce                          ; clock enable (1=write, 0=hold)
r                           ; synch reset

;Nodes
cin                         ; generates carry-in for subtract

;Outputs
s0 s1 s2 s3                 ; adder register outputs

PARTITION s3_0 cin s0 s1 s2 s3

EQUATIONS

cin.D1  = /add*ce*/r        ; cin generates carry into s0 when subtr.
cin.D2  = /add*ce*/r
cin     = cin.D1 gnd cin.D2 ; cin macrocell output not used

s0.D1   = b0*add*ce*/r      ; when adding, use positive B-operand
        + /b0*/add*ce*/r    ; when subtracting, use negated B-operand
s0.D2   = a0*ce*/r          ; positive A-operand
s0.fbk  = /ce*/r            ; when ce disabled, recirculate Q-output
s0     := s0.D1 xor s0.D2   ; macrocell output is A xor B xor c_in
s0.add  = vcc               ; enable carry_in to macrocell
s0.clkf = c

s1.D1   = b1*add*ce*/r
        + /b1*/add*ce*/r
s1.D2   = a1*ce*/r
s1.fbk  = /ce*/r
s1     := s1.D1 xor s1.D2
s1.add  = vcc
s1.clkf = c

s2.D1   = b2*add*ce*/r
        + /b2*/add*ce*/r
s2.D2   = a2*ce*/r
s2.fbk  = /ce*/r
s2     := s2.D1 xor s2.D2
s2.add  = vcc
s2.clkf = c

s3.D1   = b3*add*ce*/r
        + /b3*/add*ce*/r
s3.D2   = a3*ce*/r
s3.fbk  = /ce*/r
s3     := s3.D1 xor s3.D2
s3.add  = vcc
s3.clkf = c
```

For more information about writing arithmetic equations for XC7000 devices, see “Design Rules for Arithmetic Design” in the “Advanced Behavioral Design Techniques” chapter of the *XEPLD Design Guide*.

Hierarchical Design

You can create symbols with schematics under them and place these symbols in your top-level schematic. This can make your design more modular and easier to understand.

User-created symbols are termed **custom components**. Custom components with schematics under them are termed **macros**, as opposed to **primitives**, which are custom components with behavioral modules under them. For information about creating primitives, see the "Using Behavioral Modules in Schematics" chapter.

The procedure for creating a symbol with an underlying schematic is the same for EPLD and FPGA except for the library you use (XC7000 instead of XC3000 or XC4000):

1. Create a lower-level schematic using XC7000 library symbols. To make a device-independent custom macro, use only device-independent symbols.
2. Create a symbol for the schematic.
3. Add attributes that the symbol needs to work in your CAE tool.

Notes for Viewlogic users:

- Label the nets in your lower-level schematic with the same names as the pins on the symbol.
- The block type of the symbol must be Composite (not Module). Use the **`Change`** → **`Block Type`** command to change this.
- If you copy a Xilinx-supplied library symbol to use as the basis for your custom macro, make sure you delete the invisible symbol attribute LEVEL=XILINX, as this marks the symbol as a primitive. Use the **`Change`** → **`Attr`** → **`Dialog`** → **`All`** command to view and delete this attribute.

Notes for OrCAD users:

- Label the module ports in your lower-level schematic with the same names as the pins on the symbol.
- Use the Edit Library (or LIBEDIT) utility to create your symbol.

Note: For information about storing custom components, see the "Using Behavioral Modules in Schematics" chapter. Custom

components with underlying schematics are stored in the same way as custom components with underlying behavioral modules.

Schematic Custom Component Example

This next example shows you how to create a custom symbol with an underlying schematic. The steps for Viewlogic users are shown, with notes at the end for OrCAD users. Follow these steps:

1. Create the schematic using common symbols from the XC7000 library. It should look something like this:

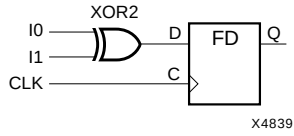


Figure 4-10 The REGXOR Schematic

2. Create a symbol with pin names that match the inputs and outputs of the schematic.

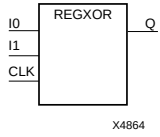


Figure 4-11 The REGXOR Symbol

3. Use the **Change → Block Type** command to change the symbol's block type to composite.

Notes for OrCAD users:

- Label the module ports in your lower-level schematic with the same names as the pins on the symbol.
- Use the Edit Library (or LIBEDIT) utility to create your symbol.

Chapter 5

Using Behavioral Modules in Schematics

This chapter discusses how to include behavioral (equation-based) modules in schematic designs. There are two reasons why you may want to use behavioral modules in your schematic:

- If portions of your design are already implemented using conventional programmable logic devices (PLDs), you can re-use your existing PLD equations without having to redraw the same logic schematically.
- You may wish to create a "custom primitive" symbol using equations (instead of a schematic-based "macro" symbol) because of the efficient sum-of-products logic equations. You can often achieve better logic density and performance for custom logic functions in an EPLD by using equation-based modules due to the inherent sum-of-products logic structures comprising the EPLD Function Block architecture. Custom primitives are easy to create, and can be used just like regular library components.

This chapter shows you how to use PLD symbols, create new components, edit library components, store custom components, and adapt behavioral logic to schematic designs.

This chapter includes design examples that use PALs in a schematic.

Preparing a Component

To prepare a PLD or custom primitive for use in a schematic design, follow these steps. After you have prepared your component, you can use it in a design just as you can any library component. However, unlike schematic-based macro components, behavioral components do not support functional simulation.

1. Create a PLUSASM equation file or a file that can be converted to PLUSASM; see "Choosing a Behavioral Design Method" later in this chapter for details. Name the file *symbolname*.pld.

   The CHIP statement in this file must specify the symbol name:

   ```
   CHIP symbolname COMPONENT
   ```

   For a custom symbol, use the COMPONENT keyword. If you decide to use the standard PL20V8 or PL22V10 symbol from the library, use "20V8" or "22V10" as the PLD type instead:

   ```
   CHIP symbolname 22V10
   ```

2. Run TRANSLATE → PLUSASM on the file to perform a syntax check and create the database file, *symbolname*.vmh, for the custom component. This file is automatically placed in the custom library subdirectory (\clib) of your design directory. If you use XEMAKE to process your design, PLUSASM is run automatically.

3. Use a PL20V8 or PL22V10 symbol, or use the SymGen automatic symbol generation utility to create a symbol. See "Choosing a Symbol" later in this chapter for details.

4. Add attributes that the symbol needs to work in your CAE tool.

   For Viewlogic symbols, SymGen automatically adds the LEVEL=XILINX symbol attribute to mark the symbol as a primitive.

   When you use the symbol in a schematic, add the PLD=*symbolname* attribute to the symbol instance (do not add it using the symbol editor). It ensures that if you use XEMAKE to process your design, the equation file is automatically assembled by PLUSASM whenever you modify your equation file. If you do not use the PLD=*symbolname* attribute, you must assemble the equation file as a separate step every time you change it.

Behavioral Module Example

This first simple example shows you how to create a custom symbol with an underlying equation file. The steps for Viewlogic and OrCAD users are shown. Follow these steps:

1. Create the PLUSASM file, regxor.pld. The CHIP statement specifies the symbol name and the COMPONENT keyword.

```
TITLE    Registered XOR gate
AUTHOR   John Q. Engineer
COMPANY  Xilinx
DATE     July 29
CHIP regxor COMPONENT

;PINLIST
I0 I1 CLK Q

EQUATIONS

Q:= I0 :+: I1
Q.CLKF = CLK
```

2. Run TRANSLATE → PLUSASM on the file.
3. Create a symbol by running SymGen from the operating system, as follows:

   `symgen regxor -v` (for Viewlogic)

   or:

   `symgen regxor -o` (for OrCAD)

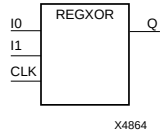


Figure 5-1 The REGXOR Symbol Created by SymGen

   If you are an OrCAD user, you must perform additional steps to prepare your symbol. See "Choosing a Symbol" later in this chapter for details.

4. You can turn off the display of the LEVEL=XILINX attribute and other attributes using the **`Change`** → **`Display`** → **`Attrs`** → **`Off`** command (for Viewlogic).
5. Add the PLD=regxor attribute to the symbol instance when you use the symbol in a schematic.

Choosing the Behavioral Design Method

The following design methods are available:

- Use a PLUSASM file.
- Convert a JEDEC file to a PLUSASM file.
- Use Xilinx ABEL.
- Use a third-party high-level language and convert to a PLUSASM file using a PLD compiler.

Using PLUSASM

Use PLUSASM to develop your PLD equation files if you want to access specific architectural features such as the high speed carry paths of the device.

PLUSASM is the native language for Xilinx EPLDs, based on the PALASM2 Boolean equation syntax. In addition, the language contains constructs that allow you to access the advanced architectural features of the XC7000 architecture.

You can target any PLD or custom component in the schematic library by placing PL22V10, PL20V8, or COMPONENT in the PLUSASM equation file CHIP statement.

PLUSASM File Structure

The basic structure of the PLUSASM file is illustrated in Figure 5-2. The file structure is identical for both full behavioral designs and behavioral modules of schematic designs. However, behavioral modules have a different CHIP statement syntax and pinlist, and they can contain only a limited subset of declaration statements. For detailed information on each PLUSASM command, see the "Plusasm Command Reference" chapter of the *XEPLD Reference Guide*.

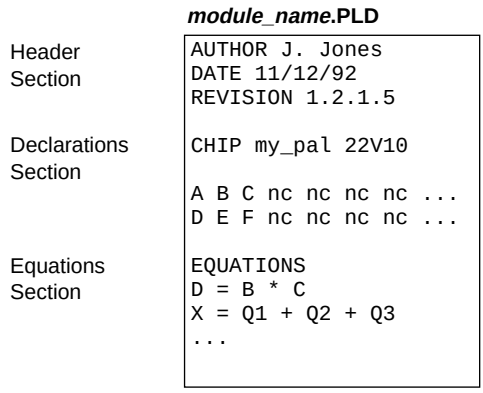


Figure 5-2 PLUSASM File Structure

The **header section** is used for design documentation only; these commands are ignored by XEPLD and do not affect your design. The header can contain the following statements in any order:

- TITLE *any_text*
- AUTHOR *any_text*
- DATE *any_text*
- REVISION *any_text*
- TIME *any_text*
- COMPANY *any_text*

Statements in the **declarations section** specify device I/O pins and affect how your behavioral equations are mapped into a specific device. The first statement (after the Header Section) must be the CHIP statement immediately followed by the pinlist. All other declaration statements may be used in any order.

Table 5-1 PLUSASM Declaration Statements

Declaration Statement	Function Overview
CHIP	Specifies the file type, file name, and pin list.
MINIMIZE	Controls the use of Boolean logic minimization.
PARTITION	Specifies the relative order of equations for arithmetic designs.
STRING	Specifies a global text string substitution.

Specify your behavioral design equations in the **equations section**, which must begin with the EQUATIONS keyword. You may use any valid PLUSASM equation syntax.

Using JEDEC Files

Translate each JEDEC file into a PLUSASM equation file by using the XDM **`Translate`** → **`JED2PLD`** command. Then you can process your schematic design using the **`Translate`** → **`XEMake`** command just as you would any EPLD design.

JEDEC files are useful for easily importing existing files for 22V10 and 20V8 PALs. Always use either the PL22V10 or PL20V8 library symbols to represent your JEDEC files.

XEPLD supports 24-pin 20V8 JEDEC files compatible with Lattice format and 24-pin 22V10 JEDEC files compatible with Cypress, AMD, and TI format.

Using Xilinx ABEL

Use Xilinx ABEL to develop your behavioral modules if you want to take advantage of its high-level language capability but do not need to access device-specific features such as the high-speed carry path.

Use the **`Xilinx ABEL Compile`** → **`Xilinx EPLD Netlist`** command to generate a PLUSASM .PLD file. Then use the **`SymGen`** command to automatically create a symbol for the file.

Using a PLD Compiler

Use a PLD compiler to develop your PLD files if you want to take advantage of the compiler's high level language capability but do not need to access device-specific features such as the high speed carry path. This method is also useful for importing existing PAL files.

The native language for Xilinx EPLDs is PLUSASM, a language based on the PALASM2 Boolean equation syntax (.PDS). Many popular PAL compilers such as ABEL, LOG/iC, and PALASM can generate the PALASM2 boolean equation files required by the XEPLD software. By using your PAL compiler's built-in ability to generate .PDS files (such as the ABEL XFER utility), you can easily generate PLUSASM-compatible equation files.

These equation files can be targeted to the PL22V10, PL20V8, or custom schematic symbols. If you are targeting a custom symbol, change the device type in the CHIP statement to COMPONENT.

Choosing the Symbol

To include a behavioral equation file into your schematic design, you must use a special PLD library component or create a custom component. Table 5-2 shows the various ways in which you can specify that a PLD file is targeted to a schematic symbol.

Table 5-2 Specifying Library Components

Original Implementation	CHIP Statement	Library Component
PAL - 22V10*	22V10	PL22V10
PAL - 20V8, GAL - 20V8*	20V8	PL20V8
Other PALs**, Original Equations	COMPONENT	Custom Symbol (created by SymGen)

* You can also use a custom symbol for these PALs if their logic equations are expressed in a device-independent manner.

** Any PAL device number other than 22V10 or 20V8.

Using the PL22V10 or PL20V8

XEPLD supports the 22V10 and 20V8 PAL devices through special PAL library components. Choose these components if you already have PAL designs targeted for them or if you have experience writing equation files for them. XEPLD supports all implied features of these devices and provides automatic partitioning and equation splitting.

Note: If you do not have access to the PALASM source files for these devices, XEPLD provides automatic JEDEC file conversion.

Using SymGen to Create Custom Symbols

For PALs other than the 22V10 and 20V8 and other behavioral modules such as those using PLUSASM's features, use the SymGen command to create a custom symbol. SymGen processes an .XSF file, which you can create using any of these commands:

- PLUSASM, which you use to assemble the equation file that defines the logic of your custom primitive symbol
- FitNet, which you use to process a schematic design to create an EPLD symbol
- XEMake, which runs the PLUSASM and FitNet commands

Use SymGen on the XDM command line or the operating system command line, as follows:

`symgen` *design_name* **`-`***option*

The –*option* specifies the CAE tool. Use –v for Viewlogic, –o for OrCAD, –m for Mentor Graphics, or –c for CADENCE.

SymGen also produces a report, *design_name*.SMR, which explains how the symbol was created and displays a diagram of the pinout.

Viewlogic Symbols

If you are a Viewlogic user, SymGen creates the symbol and places it in the sym directory below your design directory. You can add it to any schematic in your design just as you would any other symbol. No special conversion steps are necessary.

OrCAD Symbols

If you use the SymGen command with the –o option, SymGen creates a .CMD file and places it in your design directory. To convert the .CMD file into a symbol, you must perform these additional steps:

1. Enter the OrCAD Edit Library utility using one of these methods:
   - From XDM, select **DesignEntry → OrCAD** from the menu, double click on **Design Management Tools**, select your design from the list, and click on **OK**. Double click on **Schematic Design Tools**, then double click on **Edit Library**.
   - From the operating system, type the following command:

     libedit
2. In response to the `Read Library?` prompt, type the following:

   .\userlib.lib↵

   This creates a library called userlib in your design directory.
3. The Library Edit screen appears. Select the **Import** command and type ***file_name*.cmd**↵ to invoke the command file.
4. The symbol appears. Select **Library → Update Current** to save the symbol to memory.
5. Select **Quit → Update File** to save the library to disk.
6. Select **Abandon Edits** to exit the library editor.

You can now add this symbol to any schematic in your design just as you would any other symbol.

Editing Existing Library Components

Most XC7000 library components are defined as primitives. The PLUSASM equations defining these components are supplied in the *$XACT*/examples/behavior/library (for workstations) or \XACT\EXAMPLES\BEHAVIOR\LIBRARY (for PCs) directory for your reference. You may copy and edit these equation files as a convenient way to implement customized logic components.

Follow these steps:

1. Copy the equation file to your design directory and rename the file to the name you wish to call your custom symbol (*symbolname*).
2. Change the CHIP statement in the PLUSASM file as follows:

   `CHIP` *symbolname* `COMPONENT`

3. If you want to use the existing library symbol as a basis for your custom symbol, copy the symbol from the XC7000 library into your design directory and rename the symbol to *symbolname*.
4. Edit the PLUSASM file to specify the custom behavior you need.
5. If you added, deleted, or changed any pin names in your equation file, edit the symbol and make the corresponding changes.

The same symbol may have an equation file in the EPLD library and a schematic in the library for another device family. The *Libraries Guide* lists this information for each library component.

If you are modifying an existing library symbol and wish to use it in designs for more than one device family, you will have to store it in two or more different design directories. If you wish, you can define different logic for the same symbol in each directory so your component is optimized for each device family.

Storing Custom Components

After you create your custom component, you should store it in each design directory where you need to access it.

If you have defined the underlying logic differently for targeting two or more different device families, you should store the component in two or more different project directories or library directories. Each directory would contain the underlying logic for one device family.

Viewlogic Components

You should store your custom library files in your project directory. You cannot add custom symbols to the XC7000 library directory or modify any of the Xilinx-supplied symbols or macros. However, you can copy Xilinx-supplied symbols or macros to your directory, rename them, and edit them.

OrCAD Components

You can store your library file and your macro schematics under the \XACT\XC7000 library directory. Do not add to or modify the xc7000.lib file or any of the library macro schematics supplied by Xilinx. Store equation files for custom primitives in each design directory for which you want to use the component.

Editing Behavioral Modules for Use in Schematics

This section presents some information about editing behavioral files to make them work in schematics. If you are unfamiliar with PLUSASM, see the *XEPLD Reference Guide*.

The circuit shown in Figure 5-3 demonstrates how to handle many common situations when using PALs in schematics. This example deals with the strategies and procedures for incorporating PALs into a schematic and is not intended to be a complete tutorial.

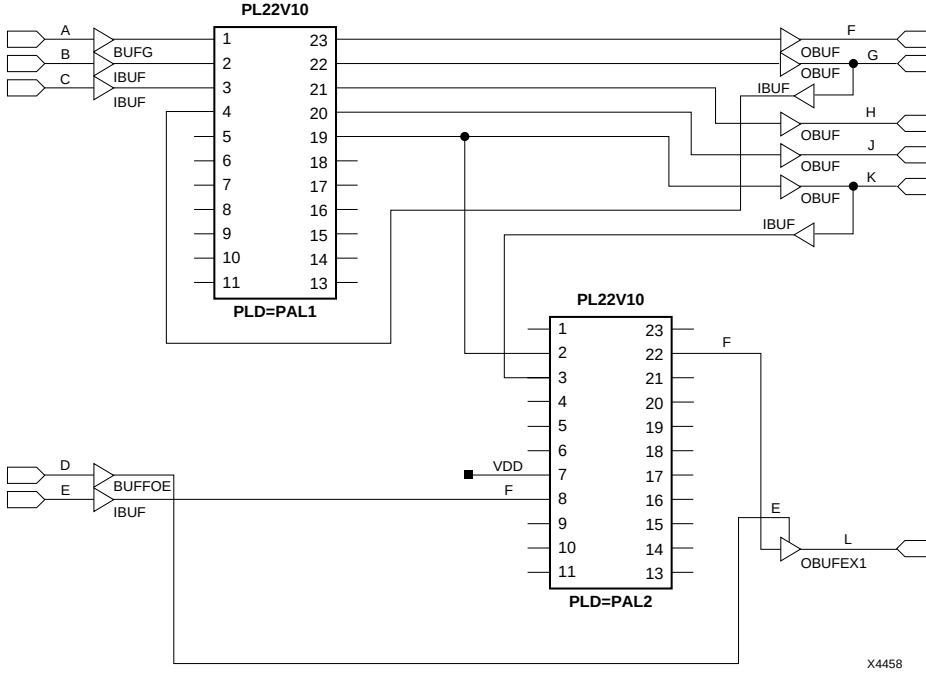


Figure 5-3 Example Schematic Using PALs

The original circuit for this design used 22V10 PALs and therefore it is easy to use the PL22V10 library components. The original PAL1 equation file is shown here:

```
TITLE PAL1
CHIP   PAL1   P22V10;
; PINLIST (Highest pin number = 24)
A B C NC NC NC NC NC NC NC NC NC NC NC NC NC K J H G F NC
; PALCNVT Design Example PAL1
EQUATIONS
F := (B);
G := (F);
G.TRST = (C);
H := (G);
J := (B * K);
K := (B);
K.TRST = (C);
```

In most cases, equation files can be targeted to library PAL components with no modification. However, this example shows you how to modify the equation files when the PAL has tri-state outputs or bidirectional signals that go off-chip. The modified equation files are shown here:

```
TITLE PAL1
CHIP   PAL1   P22V10;
; PINLIST (Highest pin number = 24)
A B C G_PIN NC NC NC NC NC NC NC NC NC NC NC NC NC K J H G
F NC
; PALCNVT Design Example PAL1
EQUATIONS
F := (B);
G := (F);
G.TRST = (C);
H := (G_PIN);
J := (B * K);
K := (B);
K.TRST = (C);
```

Assigning Output Enable Signals to FOE Nets

Assigning signals to the global fast output enable (FOE) nets reduces output enable delays and macrocell resource requirements. The XEPLD optimization software automatically assigns output enable signals to the global FOE nets whenever possible, but you can also explicitly specify FOE nets. To specify signal D as an FOE control signal for output L, do the following:

1. Permanently enable the PAL outputs by connecting the .TRST control pin to VDD in the schematic.
2. Connect signal L to an OBUFEX1 output buffer in the schematic.
3. Connect signal D to a BUFFOE input buffer in the schematic.
4. Connect the BUFFOE input buffer to the OBUFEX1 output enable input in the schematic.

Assigning Functions to Fast Function Blocks

You can assign critical functions to Fast Function Blocks (to take advantage of their higher speed and increased output drive capabilities) using the “F” attribute on inputs or outputs. Using the “F” attribute on inputs assigns signals to Fast Inputs.

To assign signal L to a Fast Function Block and signal E to a fast input pin, do the following:

1. Add the “F” attribute to the net driven by signal L.
2. Add the “F” attribute to the net that is driving signal E.

For more about using Fast Function Blocks, see the “EPLD Architecture and Design Tradeoffs” chapter. For more about the F attribute, see the “Attributes” appendix.

Assigning Bidirectional I/O Signals

Two common uses of bidirectional signals are described in this section. In Case 1, Signal G is a bidirectional output of PAL1 that goes off-chip. In Case 2, signal K is an input of PAL2 that can be driven by PAL1 or by an off-chip signal.

Case 1 — Bidirectional Outputs That Go Off-Chip

To create a bidirectional signal in a schematic for XC7000 devices that uses pin feedback, you must use two separate pins on the PLD symbol (one input and one output) even though the physical implementation in the XC7000 device requires only one I/O pin.

Because signal G in the example was defined as a bidirectional signal in the original PAL, it must use pin feedback in the XC7000 device. To create a bidirectional signal G, do the following:

1. Connect signal G of PAL1 (pin 22) to an OBUF output buffer.
2. Create a new input called G_PIN on PAL1 (pin 4). Connect G_PIN to an IBUF input buffer.
3. Connect the signal G OBUF output and the signal G_PIN IBUF input to an I/O pad.
4. In the PAL1 equation file, change NC to G_PIN in position four of the pinlist. This defines G_PIN as a pin.
5. In the PAL1 equation file, wherever signal G appears on the right side of an equation, change it to G_PIN.

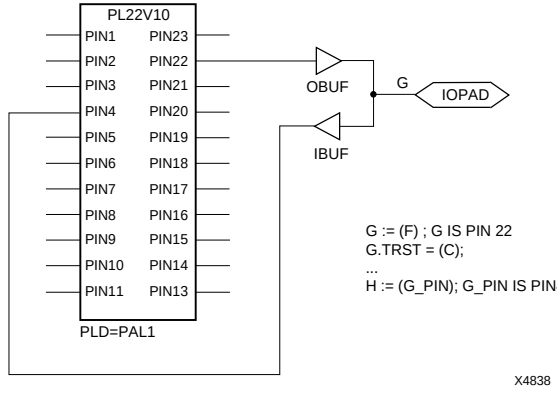


Figure 5-4 A Bidirectional Signal in a PAL

Note: In an XC7000 device, you have the option to use either the device pin (pin feedback) or the macrocell feedback. Macrocell feedback is the default for any signal not explicitly defined as pin feedback by the use of separate input and output pins.

Case 2 — Using Both Macrocell and Pin Feedback

Within the PAL1 equation file, the internal feedback of signal K is always used.

Within the PAL2 equation file, you want to use the signal at the XC7000 device pin.

To change signal K to an XC7000 I/O signal, and to use macrocell feedback for PAL1 and pin feedback for PAL2, do the following:

1. Connect signal K (PAL1, pin 19) to an OBUF output buffer.
2. Connect signal K (PAL2, pin 3) to an IBUF input buffer.
3. Connect the OBUF output and the IBUF input to an I/O pad.

Chapter 6

Design Verification

This chapter describes the simulation methods, and reports available to help you analyze and verify your design.

Simulating Designs

XEPLD supports a variety of third-party simulators, allowing you to perform functional or timing simulation of your finished design.

Functional simulation is supported by models provided in the XC7000 library for each of the library symbols included in that library. If your design contains custom primitives or PLDs defined behaviorally, no simulation model exists for those symbols, and your design cannot be functionally simulated.

To perform timing simulation on a design, you must first translate it into a netlist consisting of XC7000 library models. XEPLD automatically creates simulation files in the XNF netlist format, which can be exported to the Viewlogic ViewSim simulator (.WIR), or the OrCAD simulator (.VST) using the Xilinx-supplied CAE tool interfaces and libraries. You can also use .XNF files with other simulators that support Xilinx.

Note: When XEPLD processes your design, some of your original nodes may be removed due to circuit optimization. These nodes cannot be viewed or stimulated. All of the external I/O signals are always maintained.

Making a ViewSim or VST Functional Simulation Model

To create a Viewlogic functional simulation model, either select the **`Verify`** → **`VSM`** command, or enter Viewlogic (**`DesignEntry`** → **`Workview`**), open the top-level schematic, and select **`Export`** → **`Wirelist`** → **`ViewSim`**. This creates a .VSM file for simulation.

To create an OrCAD functional simulation model, select the **Verify → XSimMake** command from the XDM menu. Select as the program flow **Orcad_Epld_Func**. This creates a VST file for simulation.

See the *OrCAD Interface User Guide* and *Viewlogic Interface User Guide* for more information on simulation.

Making a ViewSim or VST Timing Simulation Model

To create a Viewlogic or OrCAD timing simulation model, follow these steps; for all commands, use the default options:

1. Select the **Translate → XEMake** command from the XDM menu. This compiles your design, creating a *design_name*.VMH file.
2. Select the **Verify → XSimMake** command from the XDM menu.

   On the SUN platform, Viewlogic is the default. On the PC platform, you are prompted for the type of simulator:

   - Select **Orcad_Epld_Timing** for OrCAD.
   - Select **Viewlogic_Epld_Timing** for Viewlogic.
3. Select your file name from the list of .VMH and .VMD files that are displayed.

   XEPLD creates a *design_name*.VST file (for OrCAD) or a *design_name*.WIR file (for Viewlogic).

See the *OrCAD Interface User Guide* and *Viewlogic Interface User Guide* for more information on simulation.

Using XNF-Compatible Simulators

Many third-party simulators can support .XNF files. These files contain all necessary timing and wirelist information.

To create an XNF model of your design:

1. Select the **Translate → XEMake** command from the XDM menu. This compiles your behavioral design creating a *design_name*.VMH file.
2. Select the **Verify → VMH2XNF** command from the XDM menu. This displays a list of .VMH files.
3. Select your file name from the list.

XEPLD creates a *design_name*.XNF file that can be simulated with any XNF-compatible simulator that provides an XC7000 library.

Simulating Board-Level Designs in Viewlogic

You can simulate a circuit that contains one or more EPLD devices and even some non-EPLD devices using Viewlogic software.

You do not use XSimMake to prepare the design for board-based simulation, because XSimMake can only process chip-level designs, and you must run VSM on the entire board-level design.

As with chip-level designs, you must pulse the PRLD signal at the beginning of the simulation to force all registers in the EPLDs to a known state.

Functional Simulation

To perform functional simulation on a board-level design, follow these steps:

1. Make sure all symbols in each EPLD design are from the XC7000 library. The XC7000 alias on each symbol allows you to mix technologies in the board-level simulation.
2. Create a symbol for each EPLD design. A simple way to do this is to run each design through the EPLD fitter to generate *design_name*.XSF files, then run **`SymGen`** on each .XSF file.
3. Create a board-level schematic containing the EPLD chip symbols and any other symbols that are part of the system.
4. Run **`vsm`** on the entire board-level design to generate the system functional model. You are now ready to simulate.

Timing Simulation

To perform timing simulation on a board-level design, follow these steps:

1. Run each design through the EPLD fitter.
2. If you do not already have chip symbols for your EPLDs, run **`SymGen`** on each *design_name*.XSF file the fitter produces to create chip symbols for the board-level schematic.

3. Run **VMH2XNF** on each EPLD design.
4. Run **XNF2WIR** on each EPLD design using the **-l** option. This option tags each component within the EPLD model with the XC7000 alias, allowing you to mix technologies in the board-level simulation.
5. Create a board-level schematic containing the EPLD design symbols and any other symbols that are part of the system.
6. Run **vsm** on the entire board-level design to generate the system timing model. You are now ready to simulate.

Preload Values in Functional and Timing Simulation

The only differences in the functionality of a design expected between functional and timing simulation involve the initial states of registers and latches in the design. Functional simulation assumes that preload values are as defined in the library components. Timing simulation uses the actual preload values implemented by the fitter.

When functional and timing simulation yield different results, it is probably because the XEPLD fitter did not use the library default due to the preload optimization feature, or you specified INIT attributes, which are implemented by the fitter and take effect only during timing simulation.

For example, if an FDR component is mapped to a Fast Function Block, the FDR will appear to preload to 0 during functional simulation, because that is how the library component is defined. However, during timing simulation, this FDR will actually preload to 1, because that is physical preload state of an FFB macrocell register where the FD component was mapped.

See the “EPLD Architecture and Design Tradeoffs” chapter and the PRELOAD_OPT and INIT attribute descriptions in the “Attributes” appendix for more information about preload values.

Verifying Designs

After you have compiled your design using the **Translate → XEMake** command, XEPLD generates reports that tell you how your design fits in the target device and how fast the design will run.

- The Resource Report, *design_name*.RES, gives you a summary of the logic utilization of the device, your I/O usage, and the resources that were left unused.
- The Equation Report, *design_name*.EQN, is a PLUSASM behavioral design file created by the XEPLD optimizer that shows you exactly how all your logic was implemented after XEPLD performed logic optimization. Optimization includes collapsing of combinatorial logic nodes into device outputs and registers, assigning signals to global FastCLK and FOE nets, utilization of input pad registers, and the creation of UIM-AND functions. This report contains all declarations and equations produced by the XEPLD optimizer to implement your design.
- The Pinlist Report *design_name*.PIN shows the final XC7000 device pinout of your design.
- The Timing Report *design_name*.TIM shows the calculated worst-case timing based on the physical implementation of your design.

Verifying Design Fit

When XEPLD has successfully compiled your design, you will see the following message on your screen:

```
Design Successfully Mapped. Examine the following
report files:
```

Examine the Resource Report to determine the amount of chip resources used to implement your design and how much remain. An example Resource Report is shown on the next page. The schematic for this report is the Johnson counter example in the "Getting Started with Schematic Design" chapter. This design was targeted for the XC7318-5PC44.

The Logic Resources section of the Resource Report shows that 4 macrocells were used in the design and 14 remain available for additional logic. The Pin Resources section shows the types of signals required by the design, the types of device pins used to satisfy the

signal requirements, and the remaining device pins that can be used for additional signals.

This report shows that the 2 input signals were placed on input pins, 3 of the 4 output signals were placed on I/O pins, and the remaining output was placed on an FOE pin. A FastCLK pin was also used. A total of 28 pins (14 input and 14 I/O) remain available for additional input signals.

```
XEPLD, Version 5.0                                    Xilinx Inc.
                          Resource Report
 Circuit name: JCOUNT
Target Device: XC7318-5PC44        Integrated:  7- 1-94, 11:38AM


LOGIC RESOURCES

                 Required  Used      Remaining
Function Blocks    1         1         1
Macrocells         4         4        14


PIN RESOURCES:

Type   Req   --------Used-------------  --------Remaining------
            I  O  I/O Fclk Foe Cen Tot  I  O  I/O Fclk Foe CenTot
Inputs   2   2       0                 2 14      14                28
Outputs  4      0    3    0   1   0    4     0   14    0   0   0  14
I/Os     0          0                 0         14                14
Fclks    1               1            1               1            1
Foes     0                   0        0                   1        1
Cens     0                       0    0                       0    0
       --- --- --  ---  --- --- --- ---
         7   2  0    3    1   1   0   7


Note:The design requires 0 pins with Fast Input capability.
     This device has 11 pins with Fast Input capability.

 End of Resource Report
```

Common Questions and Answers

This appendix lists frequently asked questions about EPLD software and its CAE tool interfaces, and gives explanations and solutions.

Drawing the Design

This section lists problems you may encounter because your CAE tool drawing package is not properly configured for XEPLD software.

Why Do I See White Boxes Instead of Components?

If you are a Viewlogic user and your schematic contains symbols from a device family library (such as XC7000) that is not included in your viewdraw.ini file, you see white boxes when you view your schematic.

A likely cause of this problem is forgetting to run the Altran program when converting from one device family to another; see the "Device-Independent Design" chapter for details. Even after you run Altran, components from the old library that are not in the new library appear as white boxes — you should find equivalent components that are compatible with the new library.

Another likely cause is not configuring viewdraw.ini properly, with correct pathnames and library aliases. The example in the "Getting Started with Schematic Design" chapter includes information about how to configure Viewlogic software for the XC7000 device family.

Why Are Some of My Components Missing?

If you are an OrCAD user and your schematic contains symbols from a device family library (such as XC7000) for which OrCAD is not configured, you see missing components when you view your schematic.

You probably were converting this design to a new device family and forgot to do one of the following:

- Configure OrCAD using the XDraft command.
- Substitute symbols compatible with the new library for symbols compatible with the old library.

The example in the "Getting Started with Schematic Design" chapter includes information about how to configure OrCAD software for the XC7000 device family.

Fitting the Design

This section lists problems you may encounter when you run fitter commands such as XEMake or FitNet.

What Does "Component Not Found" Mean?

If you get this error message:

```
ppi0005:instance: component_name type: component_type:
COMPONENT IS NOT FOUND IN A LIBRARY
```

it means one of the following occurred:

- You did not assemble the equation file for a custom primitive component using the PLUSASM command before you ran FitNet.
- You did not assemble a PAL component using the PLUSASM command before you ran FitNet. If this was the cause of the error, you also get this message:

  ```
  ppi2029:[Error] Cannot find PLD 'pld_file'! Make
  sure the component is assembled!
  ```

- You targeted the design to an XC7272 device while running all the commands up to XNFMerge, then targeted another XC7000 device when you ran FitNet, or vice versa. The library files for XC7272 devices have .VMD extensions, while the library files for all other

devices have .VMH extensions. The XEPLD software was looking for the wrong extension.

- One of the symbols in your design is not XC7000-compatible.
- A common library macro in your design has not been converted to XC7000 format using Altran (Viewlogic) and a macro subcomponent is not XC7000-compatible.
- You renamed your library alias in your viewdraw.ini file, and the XEPLD software cannot find the components within a macro schematic because they have the original alias.
- Your XACT variable is not properly set, and the XEPLD software cannot find the CLIB directory.

You can avoid the first three of these situations by using XEMake, which runs PLUSASM automatically and targets the same device from beginning to end, to process your design.

What Does “Component Not Supported” Mean?

If your target device is an XC7200 device, which contains only High-Density Function Blocks, but you used a PLFFB9 component in your design, which can only map to a Fast Function Block, you will see a message such as the following:

```
dr0043:[Error] Component 'component_name' is NOT
supported in the device_name device.
```

If your target device is an XC7336 or XC7318, which contains only Fast Function Blocks, but you used a component, such as an adder, that can only map to a High-Density Function Block, you will see a message such as the following:

```
dr0074:[Error] The instance 'component_name'
uses arithmetic mode. This cannot be used with
the XC7336/XC7318 devices which do not support
arithmetic carry-in from previous macrocells.
Choose a new component or use another XC7000
device.
```

To solve either problem, choose a different XC7000 device or use a different PLD component.

Why Can't I Make a Direct Pin-To-Pin Path?

You see the following error message if you have an input pad connected directly to an output pad with nothing in between:

```
ppi2047:[Error] Input buffer 'buffer_name' directly
connects to output buffer 'buffer_name'. A BUF
component must be inserted between these buffers
to form a direct pin-to-pin data path.
```

Adding a BUF component eliminates this message, but it is a good idea to verify that you did not forget to include part of your design.

A similar error occurs if you try to drive an OBUF directly with VCC or GND:

```
ppi2044:[Error] Illegal connection between VCC
and output buffer 'buffer_name'. Insert a BUF
component to generate a constant-driven output
pin.
```

```
ppi2045:[Error] Illegal connection between GND
and output buffer 'buffer_name'. Insert a BUF
component to generate a constant-driven output
pin.
```

Again, a BUF before the OBUF eliminates the problem.

What Does "Has No Logic Connection" Mean?

The following warning message means that the software removed a hanging input or an input made unnecessary by optimization:

```
pl0112:Port 'port_name' removed because it has no
logic connection!
```

Your design will run despite this warning, but if you have hanging inputs it is a good idea to verify that you did not misspell a net label or forget to include part of your design. If the input has been removed by optimization, you do not need to do anything.

What Do I Do If I Have “Hanging Inputs”?

The following warning message means that you have one or more component inputs that are unused and not tied to VCC or GND:

```
dr0025:Hanging input (component_name:pin_name).
```

Your design will continue mapping despite this warning, but it is a good idea to verify that the hanging input has no possible effect on the functionality of your design.

The result of a hanging input is not always clear and in some cases may produce incorrect functionality. For example, if the select input to a 2-to-1 multiplexer (M2_1) is left hanging, the multiplexer’s output will be the logical OR of its data inputs.

It is a good idea to tie all unused inputs to VCC or GND, because when you next look at the schematic, you will see what you intend to do. For example, tieing a CLR signal to GND makes it obvious that you never intend to clear that register.

You can leave component outputs unconnected. Inputs and outputs of PLD symbols (such as PL22V10) are an exception: you should leave all unused inputs and outputs hanging.

Why Are Some of the Outputs Removed?

You may see this message if you left outputs hanging in your design:

```
pl0111:[Warning] 'component_name' does not drive
anything and it is removed.
```

This message occurs if all the outputs of a component are unused. Usually the removed component is a subcomponent of a macro (see the “Hierarchical Design” section of the “Design Applications” chapter for information about macros). If this is the case, you can ignore this message.

If, however, the removed component appears in your schematic, you should verify that the component outputs are connected properly. A misspelled net label can sometimes cause this message.

What Does “The Tristate Will Affect the Pad” Mean?

If your design includes a BUFE that drives an OBUF that in turn drives an IOPAD, the pad’s output enable is controlled by the BUFE. This is different from FPGA behavior. The XEPLD software will also display the following message:

```
Pin '$1I8:O' is tristated and driving an OBUF(E).
The tristate will affect the pad.
```

What Does “Connects to an External Pad” Mean?

In FPGA designs, a BUF or logic component can drive a BUFG. In EPLD designs, however, this arrangement produces the following error message, because global buffers like BUFG can be connected only to pads:

```
ppi2038:[Error] Component 'component_name' directly
connects to an External Pad! Only i/o buffer
components may connect to External Pads.
```

Placing an OPAD and an IOPIN in front of the BUFG eliminates this message. See “Assigning Logic to Fast Function Blocks” in the “EPLD Architecture and Design Tradeoffs” chapter for more details.

Note: In Version 5.1, you can drive a BUFG with any logic component and you will not get an error message.

What If My Design Doesn’t Fit?

If your design requires more macrocells or signal pins than are available in the target device, you must either reduce your logic requirements or choose a larger device. The XC7000 family includes a wide range of device types and packaging options from which to choose.

However, even if the target device has enough macrocells and signal pins, your design may still not fit due to the limitations your design places on the device. If you have only a few remaining unmapped functions, you can possibly get your design to fit by controlling the optimization of device resources.

The Resource Summary section of the Partitioning Report (*design_name*.PAR) tells you the total amount of logic and pins used and the amount of remaining resources.

There are three reasons why your design may not fit into the available resources of a target device:

- Your design is product term constrained.
- Your design is function block input constrained.
- Your design cannot access the Fast Function Block resources.

If Your Design is Product Term Constrained

Product term constrained function blocks have only a few used macrocells (outputs) but most of the shared product terms are used. The example Partitioning Report shown here illustrates this situation:

```
Part        # of     # of Input    Signal   # of        O/IO  0/IO   Size
Name        Outputs  Lines Used    Inputs   Shared PT   Req   Avail  Factor
FB1           8        24          24         0         0/0   0/8    9
FB2           7        12          12         0         0/0   0/8    7
FB3           2        10          10        12         1/0   2/0    9
FB4           3        12          12        10         0/0   0/3    8
FB5           9        21          32         8         0/0   0/3    9
FB6           1        10          10         6         0/0   0/3    4
OVERFLOW0     1        10          10         7         0/0   */*    5
              __       __          __        __         ___   ____   __
              31                                       1/0   2/25   51
```

The target device in this example was an XC7354 in a 44 pin PLCC package. This device has six function blocks named FB1 through FB6.

Note: The Part Name “OVERFLOW0” is the name assigned by the software to the unmapped logic.

You can decrease the usage of shared product terms by controlling the function splitting parameters contained in the XEPLD.CFG file. An example of this file is shown here:

```
(alias power_port VCC)
(alias ground_port GND)
(alias power_net VDD;VCC)
(alias ground net GND)
(alias pl20V8 pal20V8;gal20V8;g20V8;p20V8;p20V8r;20V8)
(alias pl22V10 pal22V10;gal22V10;p22V10;g22V10;22V10)
(alias fpga hiper;hyperpld;xepld)

Enclose the following advanced user switches in parenthesis to enable them
The following control when PLUSASM splits equations with too many product
terms and the size of the split subfunctions it ceates.

(alias max_shared_before_splitting 12)
alias max_shared_after_splitting 1
```

1. Copy the XEPLD.CFG file from the \XACT\DATA directory into your design directory. This file contains the following line:

   ```
   (alias max_shared_before_splitting 12)
   ```

   This line controls how many shared product terms a function can use before it will be split.

2. Reduce the variable far enough to cause the product term intensive functions to split into multiple macrocells. For example:

   `(alias max_shared_before_splitting 6)`

This increases macrocell count but reduces the number of partitions.

In this example, by reducing the max_shared_before_splitting variable, you can cause the functions mapped into function blocks 3 and 4 to split, allowing the function contained in OVERFLOW0 to be mapped. When you reduce this variable, only the nodes are affected; outputs will not split and design performance usually remains unaffected because the split functions are re-combined in the UIM if possible, which adds no delay.

If Your Design is FB Input Constrained

Input constrained function blocks have only a few used macrocells but most of the function block inputs are used. This situation is illustrated in the example Partitioning Report shown here:

Part Name	# of Outputs	# of Input Lines Used	Signal Inputs	# of Shared PT	O/IO Req	O/IO Avail	Size Factor
FB1	8	24	24	0	0/0	0/8	9
FB2	7	12	12	0	0/0	0/8	7
FB3	2	21	21	3	1/0	2/0	9
FB4	3	21	22	2	0/0	0/3	9
FB5	9	21	21	0	0/0	0/3	9
FB6	1	10	10	4	0/0	0/3	4
OVERFLOW0	1	21	21	1	0/0	*/*	9
	31				1/0	2/25	56

If you have combinatorial nodes in your design, you may benefit from selectively turning off the XEPLD collapser. This benefits both product term and fan-in constrained designs. However, design timing will be affected because the signal path will be lengthened. Use the LOGIC_OPT and OPT attributes to control logic collapsing.

To turn off logic collapsing for the whole design, use the

LOGIC_OPT=OFF attribute on a TBLOCK-type component.

To turn off logic collapsing for all the outputs of a particular component, use the OPT=OFF attribute:

To turn off logic collapsing only for a specific component output, add a BUF component on that output and use the OPT attribute on the output of the BUF component. Figure A-1 summarizes methods of turning logic collapsing off.

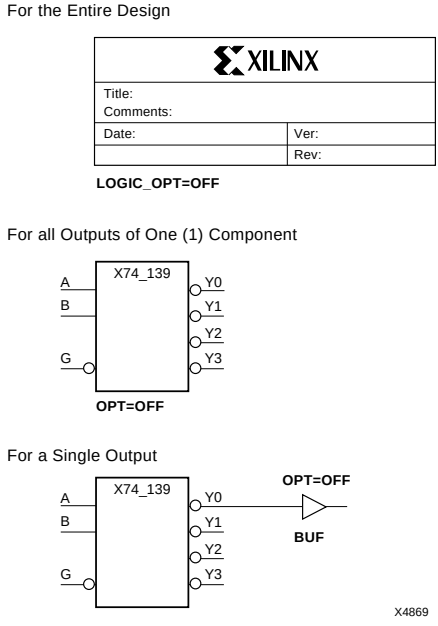


Figure A-1 Methods of Turning Logic Collapsing Off

Note: Manually splitting any registered functions with a large number of inputs may also help.

If Your Design has Unused Fast Function Blocks

If your design does not fit into an XC7300 device and the Partitioner report (*design_name*.PAR) tells you that the Fast Function Blocks have very little mapped to them, you may be able to make the design fit by assigning more logic to the Fast Function Blocks. Refer to the section in the "EPLD Architecture and Design Tradeoffs" chapter entitled "Assigning Logic to Fast Function Blocks."

Simulating the Design

This section lists problems you may encounter during functional or timing simulation.

Why Can't I Functionally Simulate a Design with a Behavioral Module?

Functional simulation uses simulation models that are pre-defined for each library component. Behavioral modules do not have models in the library. Therefore, you can perform only timing simulation on designs containing behavioral modules, after running the fitter.

If you try to perform functional simulation on a design with a behavioral module, you will see some kind of error message from your CAE tool referring to the behavioral module. For example, ViewSim displays a message like this:

```
Error - Could not find WIR file xc7000:pl22v10.1
```

Why Are My Registers Stuck at the Preload Value?

At the beginning of a simulation, you must pulse the MRESET signal Low then High, or pulse the PRLD signal, which is MRESET inverted, High then Low.

The assertion of the pulse forces all registers to a known state. The deassertion allows the registers to change states. If you do not deassert MRESET (or PRLD), your registers cannot change state.

Why Are My Internal Nodes Not Visible During Timing Simulation?

The EPLD fitter optimizes your design for efficiency, eliminating many internal combinatorial nodes. (If you are a Viewlogic user and you view your back-annotated schematic during timing simulation, these nodes appear with a "?".)

Nets between IPADs and IBUFs are observable, as are nets between OBUFs and OPADs. However, the outputs of IBUFs and the inputs of OBUFs are not observable. The outputs of IFDs or registers optimized into input pad registers are also not observable. Other nets may or

may not be observable, depending on the results of optimization. To see which of these other nets are observable, see the *design*.LGX file.

If you wish to observe the outputs of registers mapped into macrocells, and these nets are not listed in the .LGX file, you can view these nets during simulation using the following method. (You can also observe the immediate inputs of these registers, although these are unlikely to be the component inputs.)

1. Label the net connected to the register input or output. (If the node is not optimized, it will be visible by its original name during timing simulation.)
2. Label the register component having the output (or input) you want to view.
3. Fit your design using XEMake.
4. View the mapping report (*design*.map). Look for the component label in the Function Name column.
5. When you find the component label, look for the specific output you want if the component has more than one output (even if you want the input, look for the output). The output name is separated from the component name by a colon (:).

   Inversions are performed before the register in XC7000 devices. If the output name of the register has an _INV suffix, the signal you observe will be inverted from the node in your original design.
6. Look in the Macrocell Location column to find the macrocell number to which the output has been assigned.The Macrocell Location name has the following format.

   FB*fb#-mc#*

   for example:

   FB8-6
7. For the register output, use the following name as the signal name for timing simulation:

   MCQ_*fb#_mc#*

   for example:

   MCQ_8_6

8. You can also observe the signal on the D-input to the same register using the name:

   MCD_*fb*#_*mc*#

   for example:

   MCD_8_6

You can also preserve nodes using the OPT=OFF component attribute. When you apply this attribute to a component, all of that component's outputs remain visible. To preserve a single output of a component with multiple outputs, insert a BUF component on that signal and apply OPT=OFF to the BUF.

Note: Using OPT=OFF can change the fit and timing of your design, especially if you insert a buffer and apply OPT=OFF to it.

Keep in mind that timing simulation simulates your design as it is actually implemented in the chip.

Why Do Functional and Timing Simulation Yield Different Results?

The only functional differences expected between functional and timing simulation involve the initial states of registers and latches in the design. Functional simulation assumes that preload values are as defined in the library components and the .PRLD equations in behavioral modules. Timing simulation uses the actual preload values implemented by the fitter.

When functional and timing simulation yield different results, it is probably because the XEPLD fitter did not use the library default and .PRLD equation values due to the preload optimization feature.

See the "EPLD Architecture and Design Tradeoffs" chapter and the PRELOAD_OPT and INIT attribute descriptions in the "Attributes" appendix for more information about preload values.

Appendix B

Attributes

Attributes, which you place in your schematic, allow you to control the following aspects of how the software processes your design:

- Linking of PLD symbols and PLUSASM equation files
- Pin assignment
- Power consumption
- Optimization of logic, registers, and control signals
- Allocation of Fast Function Block resources

Attributes are used to express information specific to each design, as opposed to run-time options entered through the XDM menu or command line. There are three ways that attributes are placed in the schematic:

- Component attributes, such as PLD, OPT, and LOC, affect only the component instances on which they are placed.
- Global attributes, such as PRELOAD_OPT and LOGIC_OPT, affect the entire design. In OrCAD software, these are text.
- Net or Flag attributes affect individual component outputs or inputs and are represented by attributes applied to nets.

Component Attributes

The component attributes specific to EPLD designs are as follows:

- PLD=*file_name*
- LOC=*pin_name*
- LOWPWR={ON | OFF}
- OPT={OFF | ON | UIM}
- INIT={R | S}

Viewlogic Procedure

Use the **Add → Attr** command to assign a component attribute by using the following procedure:

1. Select the schematic component.
2. Select **Add → Attr**.
3. Click the middle mouse button.
4. Type the attribute string, for example LOWPWR=OFF.
5. Position the text and click the middle mouse button.
6. If you are assigning more than one attribute to the same component, repeat steps 3 through 5 for each attribute.

OrCAD Procedure

Use the OrCAD **Edit** command to assign a component attribute by using the following procedure:

1. Position the cursor over the schematic symbol.
2. Select **Edit**.
3. Select **Edit**(again).
4. Select ***n*th Part Field**, where *n* is any part field you choose from 1st to 8th.
5. In the Name field enter:

   attribute_name=*value*

Global Attributes

The global attributes specific to EPLD designs are as follows:

- LOWPWR=ALL
- LOGIC_OPT=OFF
- MRINPUT=ON
- MINIMIZE=OFF
- UIM_OPT=OFF
- FOE_OPT=OFF

- CLOCK_OPT=OFF
- REG_OPT=OFF
- PRELOAD_OPT=OFF

The PART attribute is a global attribute used by both EPLD and FPGA. For Viewlogic and OrCAD software, the procedure for using the PART attribute is different than the EPLD-specific global attributes; see the PART attribute description later in this chapter.

Viewlogic Procedure

To use one or more EPLD-specific global attributes, first place one of the title block symbols from the library into the schematic. The title block symbols include the various size sheet symbols, such as ASHEETP and ESHEETL, and the TBLOCK symbol. Then apply the global attributes to the title block symbol using the **Add → Attr** command, just as you would apply component attributes to a regular component.

OrCAD Procedure

To use one or more EPLD-specific global attributes, you must first place the keyword text string

```
|GLOBAL
```

into your schematic. Each global attribute is then listed beneath the |GLOBAL keyword. Preceed each attribute string with the pipe character (|). Align the pipe characters of all attribute strings directly beneath the pipe character of the |GLOBAL keyword and leave no blank space between text lines. For example:

```
|GLOBAL
|LOWPWR=ALL
|PRELOAD_OPT=OFF
```

Net or Flag Attributes

The F and H attributes are called Net attributes in Viewlogic software, and Flag attributes in OrCAD software.

Net Attributes (Viewlogic)

Use the **Add → Attr** command to assign a net attribute by using the following procedure:

1. Select the net.
2. Select the **Add → Attr** command. Click with the middle mouse button to activate the command.
3. Type F or H in response to the prompt, for example:

   ```
   Attribute text string: F
   ```

4. The F or H appears on the screen. Move it where you want it.
5. Click with the middle mouse button to place the attribute.

Flag Attributes (OrCAD)

In the XC7000 library you will find symbols named F and H, which are used to apply these attributes. To apply a flag attribute to an input or output pin of a component, place and connect the desired flag attribute symbol to the wire connecting to the chosen pin as shown in Figure B-1.

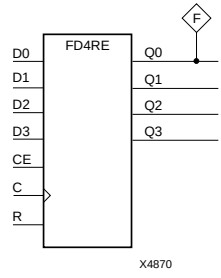


Figure B-1 Applying the F Flag Attribute

Target Device Selection — The PART Attribute

You can place the global PART attribute in your schematic to select the target EPLD device for your design. Refer to the Release Notes or the XDM Part menu for a list of EPLD device names supported by the software.

Note: This attribute is called PARTTYPE in OrCAD.

Selecting a part type in the XDM menu other than InDesign before invoking the Fitter overrides any PART attribute in your schematic.

The format of the PART value is as follows:

PART=*dddd-ssppppp*

dddd is the device number, for example 7354

ss is the speed grade, for example 12

pppp is the package type and pin count, for example PC68

Viewlogic Procedure

Apply the PART attribute as an unattached schematic attribute. Follow these steps:

1. Deselect all components by clicking on a blank area of your schematic.
2. Select the **Add** → **Attr** command.
3. Click with the middle mouse button.
4. Type the PART attribute string.
5. Position the attribute anywhere in the schematic and click with the middle mouse button.

OrCAD Procedure

Unlike other global attributes used for EPLD designs, the PARTTYPE attribute is a stand-alone text string and should not be placed beneath the |GLOBAL keyword in the schematic. Simply place the text

|PARTTYPE=*dddd-ssppppp*

anywhere in your schematic other than the GLOBAL attribute list.

PLD Equation File Name — The PLD Attribute

The PLD=*file_name* attribute on a PLD symbol specifies the name of the file with the logic equations for that PLD. This attribute is valid on custom primitive symbols (target COMPONENT in PLUSASM) and the following PLDs: PL20V8, PL22V10, PL20PIN, PL24PIN, PL48PIN, PLFB9, and PLFFB9.

Do not specify the file extension in the PLD=*file_name* attribute. This file must be in PLUSASM (.pld) or PALASM (.pds) format, although you can start with an ABEL file and convert it to PLUSASM or PALASM.

You must also specify this *file_name* as the first parameter of the CHIP statement inside the equation file, as described in the PLUSASM Language Reference section of the *XEPLD Reference Manual*. For example:

```
CHIP file_name P16V8
```

Within the .pld file, the pin list must contain the names of all the signals connected to all the PLD's pins, in the proper order. For example, if you have the signals shown in Figure B-2:

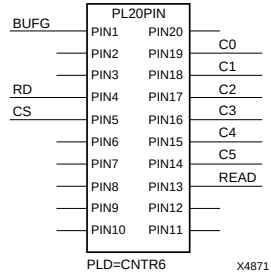


Figure B-2 Example PLD Component

you must include the following pin list in the equation file:

```
TITLE CNTR6

CHIP   CNTR6   P16V8;

;PINLIST (Highest pin number = 20)
      x4clk start NC   rd cs NC NC NC NC NC
      NC    NC    read c5 c4 c3 c2 c1 c0 NC
```

All PLD components in your schematic design must have the PLD attribute. Running XEMAKE automatically assembles all equation files named by all PLD=*file_name* attributes found in the schematic. If you do not use XEMAKE then you must assemble each PLD file in the design using PLUSASM before you run FITNET.

Like PLDs, user-specified (custom) primitives are defined by PLUSASM equation files. The PLD=*file_name* attribute is not required but can be applied as a convenient way to have your equation file

automatically assembled when XEMAKE is invoked. If you omit the PLD attribute, FITNET will expect to find a bitmap file for the symbol (*symbol_name*.VMH) in your local CLIB subdirectory.

If you forget to specify a file name for the PLD attribute, you get this message:

```
ppi2033:[Error] There is a 'PAL_name' -component
missing property PLD!
```

Pin Assignment — The LOC Attribute

Use the LOC=*pin_name* attribute on a PAD symbol to assign the signal to a specific pin. The PAD symbols are IPAD, OPAD, IOPAD, and UPAD. The pin name is P*nn* for PC packages; the *nn* is a pin number. The pin name is *rc* (rowcolumn) for PG packages. Examples are LOC=P24 and LOC=G2.

Pin assignments are unconditional in that the software will not attempt to relocate a pin if it cannot achieve the specified assignment. You can apply the LOC attribute to as many PADs in your design as you like. However, each pin assignment further constrains the software as it automatically allocates logic and I/O resources to internal nodes and I/O pins with no LOC attributes.

To save all resulting pin assignments so they are preserved the next time you modify and re-integrate the design, use the PinSave command in the XDM Translate menu. This saves the pin assignments to a *design_name*.vmf file. You can override individual pin assignments saved in the .vmf fileby changing or add LOC=*pin_name* attributes in the schematic.

Note: Pin assignment using the LOC attribute is not supported for bus components such as OBUF8.

Power Setting — The LOWPWR Attribute

This attribute is valid only for XC7300 designs. You can use this attribute as either a global or component attribute.

The default is LOWPWR=OFF (high speed) for all macrocells used in the design unless otherwise specified.

To make low power the global default power setting, place the global attribute LOWPWR=ALL in the schematic. (See the "Global Attributes" section in this chapter for instructions.)

To determine the power setting of the macrocells used by an individual symbol, use LOWPWR=ON or LOWPWR=OFF (if the global LOWPWR=ALL was used). This attribute is ignored if assigned to a symbol that uses no macrocells, such as an inverter, AND/OR gate (when optimized) input register, and so on.

F/H

Use the F or H net attribute in an XC7300 device to specify whether macrocell implementing a component output should be placed in a Fast Function Block (F), or a High-Density Function Block (H). These attributes are represented in Workview by an attribute applied to the net and in OrCAD by component symbols.

The F attribute is ignored on outputs of components that require features only present in High-Density Function Blocks, such as the following:

- PLFB9
- ADD symbols
- ADSU symbols
- ACC symbols
- BUFCE
- IFD
- IFDX1
- ILD
- COMPM
- LD
- FDCP, FDCPE
- OBUFT
- OFDT
- XOR7, XOR8, XOR9

Note: The BUFT and BUFE symbols are allowed for external outputs only and must allow FOE optimization.

The H attribute is ignored on outputs of a PLFFB9.

For logic not labeled with F or H attributes, the XEPLD software attempts to put as much logic as possible in the Fast Function Blocks first, then starts filling the High-Density Function Blocks.

MRINPUT

Specifying the MRINPUT=ON global attribute in XC7354 or XC7336 designs changes the Master Reset pin to an ordinary input. If this attribute is used, the EPLD device is initialized only on power-up.

Logic Optimization Attributes

Use the logic optimization attributes to control optimization of part or all of your design.

OPT=OFF and OPT=ON

The OPT=OFF component attribute inhibits logic optimization of all macrocells used by a symbol. OPT=ON can override the LOGIC_OPT=OFF global attribute for individual symbols.

The logic optimizer collapses the levels of logic to remove intermediate nodes. Components are optimized forward into components connected to their outputs.

If you build combinational logic using low-level gates and multiplexers, the software attempts to pack all logic bounded between device I/O pins and registers into a single macrocell.

The logic optimizer first removes all internal logic that is not used by any other logic or output buffer and is not explicitly in a PARTITION statement.

The logic optimizer moves logic forward by collapsing combinational expressions into their fanouts. If collapsing an expression into all fanouts succeeds, the original macrocell logic becomes unused and is removed.

The logic optimizer does not collapse an expression into its fanouts if the resulting expression uses too many product terms or inputs.

The logic optimizer also moves forward any logic, whether combinational or sequential, that is buffered by a tri-state buffer.

However, logic that itself contains a tri-state control is not moved forward.

The OPT attribute has no effect on any symbol that contains no macrocell logic, such as an I/O buffer.

OPT=UIM

OPT=UIM forces placement of an AND function in the UIM. It is available in Version 5.1 of the XEPLD software. The software automatically moves product terms into the UIM, but for some designs you may want to force product terms into the UIM.

The UIM can function as a very wide AND gate. This means that single product term functions can be moved into the UIM, leaving space in the macrocells for more product terms. The UIM can also implement DeMorgan equivalent functions.

LOGIC_OPT

To have logic optimization inhibited by default for the entire design, apply the global attribute LOGIC_OPT=OFF. If you do not use this attribute, the default is LOGIC_OPT=ON. You can override this setting for individual symbols using the OPT=ON attribute.

MINIMIZE

Use the MINIMIZE=OFF global attribute to inhibit logic minimization for the whole design. If this attribute is not specified, any redundant or non-effective logic found in any user-specified equation files will be eliminated throught Boolean minimization.

UIM_OPT

To inhibit UIM optimization for the entire design, apply the UIM_OPT=OFF global attribute.

UIM optimization extracts AND expressions and inverters out of macrocell logic functions and moves them into the UIM, which reduces the use of Function Block resources.

FOE_OPT

To inhibit FOE (Fast Output Enable) optimization for the entire design, apply the FOE_OPT=OFF global attribute.

FOE optimization generally applies only to BUFE, OBUFE, or tri-state PLD outputs driving an OBUF. FOE optimization changes a product-term tri-state signal to an FOE global control signal. Like FastCLK assignment, this reduces the number of UIM inputs and product terms required by each Function Block.

CLOCK_OPT

To inhibit FastCLK optimization for the entire design, apply the CLOCK_OPT=OFF global attribute.

FastCLK optimization changes a product-term clock to a FastCLK global signal, which reduces the number of UIM inputs and product terms required by each Function Block.

REG_OPT

To inhibit input register optimization for the entire design, apply the REG_OPT=OFF global attribute.

Input register optimization reduces the number of macrocells in a design by moving simple FD registers connected to IBUFs into a pad register (provided that the IBUF has no other fanouts). The clock by which the input register is controlled must be a FastCLK or an input that can be assigned to a FastCLK pin.

PRELOAD_OPT

Apply the PRELOAD_OPT=OFF global attribute to inhibit the XEPLD software from changing the preload values in the design to match the preload values supported by specified device resources such as FFBs and input registers. The default (PRELOAD_OPT=ON) allows the XEPLD software to map your design most efficiently, using the device resources most suited to the elements of your design.

You can set a High or Low preload for High-Density Function Blocks. The preload value of Fast Function Blocks depends on the use of set or reset. Input register preload values are fixed at 1, except for those on the XC7272, which are undefined.

Note: If you specify PRELOAD_OPT=OFF to control preload values, it prevents registers from mapping into FFBs, and attempting to force such a register into an FFB (using the F attribute or through pin assignment) results in an error.

INIT

To specify the preload value of a registered component, apply INIT=R (for a 0 value) or INIT=S (for a 1 value) to the component. This attribute is available in Version 5.1 of the XEPLD software.

The INIT= attributes are always obeyed by the fitter, regardless of the PRELOAD_OPT attribute, but are ignored in functional simulation.

Note: You cannot change the preload value of an input register to 0 using the INIT=R attribute because input registers physically do not support preload to the 0 state. Also, if you specify INIT=R to control preload values, it prevents registers from mapping into FFBs, and attempting to force such a register into an FFB (using the F attribute or through pin assignment) results in an error.

Index

M

N

O

P

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;