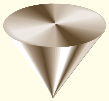
XILINX®

ONLINE

XEPLD
DESIGN GUIDE

TABLE OF CONTENTS

INDEX



GO TO OTHER BOOKS

XACT™ STEP

0401191

Copyright 1994-1995 Xilinx Inc. All Rights Reserved.

Contents

Chapter 5 Advanced Behavioral Design Techniques

Getting Started with Behavioral Design

This chapter will help you quickly understand how to develop a behavioral design using XEPLD. A brief behavioral design example is included, illustrating the automatic PAL conversion process.

An Overview of Behavioral Design Methods

A behavioral design defines the functionality of a logic circuit by using a text-based language rather than a schematic. Using XEPLD, a behavioral design can be created as a single Top-Level File which contains all design control information and behavioral equations. You can also develop your design in a hierarchical format which contains design equations in one or more Include Files linked together through a Top-Level File.

An overview of the behavioral design flow is shown in Figure 1-1.

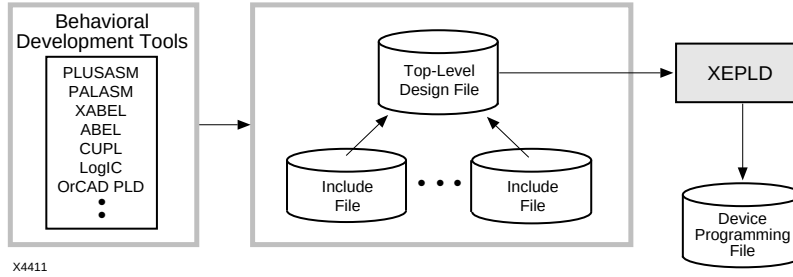


Figure 1-1 Basic Behavioral Design Flow

The following sections provide an overview of methods for creating Top-Level Files and Include Files for behavioral designs.

See Chapter 2 for more information on the structure and requirements for behavioral design files.

Converting Existing PAL Designs

XEPLD allows you to automatically use your existing PAL equation files as Include Files in XEPLD behavioral designs. This design method has several advantages:

- You can use your existing PLD compiler tools, with which you are already familiar, to edit your designs.
- You can combine existing PAL designs with designs created in PLUSASM or with new designs created using a PLD compiler.

An example of the PAL conversion process is included in this chapter. Chapter 3 provides a complete description of PAL conversion.

Using PLUSASM to Create New Designs

PLUSASM is the Xilinx EPLD development language providing complete access to all device features. It uses Boolean equation entry and provides the following capabilities:

- Control of EPLD partitioning, logic placement, pinouts.
- Access to EPLD arithmetic and fast carry features.
- Access to input pad registers and latches including Clock Enable (CE).
- Automatic support for very wide AND gates (supports an unlimited number of macrocell feedbacks and device pin inputs).
- Support for FastInput signals to Fast Function Blocks (bypassing the Universal Interconnect Matrix).
- Automatic support for Fast Output Enable (FOE) control.
- Support for ALU functions (such as adders).
- Support for device speed and power control.

Chapter 2 describes how to use PLUSASM.

Using Xilinx ABEL to Create New Designs

The Xilinx ABEL EPLD/FPGA compiler provides high-level language capability for developing behavioral designs. Xilinx ABEL can produce both Top-Level Files and Include Files. It can also create stand-alone files for use with schematic design methods.

Xilinx ABEL features:

- High-Level Design Language (HDL).
- State Machine design capability.
- Support for PLD files in schematic designs.
- Support for multi-module designs.
- Support for functional simulation.

See the *Xilinx ABEL User Guide* for more information.

Using Third-Party PLD Compilers

Any PLD compiler that produces PLUSASM-compatible output (or PALASM) can be used to create Include Files for XEPLD designs. These PLD compilers (such as ABEL, ORCAD PLD, CUPL, LOG/iC, and so on) may provide a variety of features including:

- Support for a wide range of device types (PALs and EPLDs).
- High-level design language support (HDL).
- State-machine language (SML).
- Bus and vector operations.
- Functional simulation.
- Truth table input.

When using a PLD compiler (other than Xilinx ABEL) it is important to create your equation files in a device-independent format if possible or to target one of the specifically supported PAL types (20V8 or 22V10). These device-independent files are then used as Include Files. Also, existing designs targeted at one or more PAL devices and exported in a PLUSASM-compatible format (such as PALASM) can be used as Include Files.

Converting JEDEC Files

XEPLD can disassemble JEDEC files for 22V10 and 20V8 PALs. You can use the resulting files as Include Files for XEPLD behavioral designs or as PLD files included in a schematic design. However, it is usually easier to work with the original design source files, if available. The JEDEC file conversion process is described in Appendix B of the *XEPLD Reference Guide*.

Design Example — Converting a PAL File

This section provides a behavioral design example demonstrating how to automatically convert an existing PAL design for use in a Xilinx EPLD. It consists of a 4-bit counter originally designed for use in a 22V10 PAL. The equation file is shown in Figure 1-2.

XDM must be properly installed before you can execute the commands contained in this example design. See the XEPLD 5.0 Release Notes (DS-550) for software installation instructions.

Note: This example is intentionally brief. For the complete details of PAL conversion, see Chapter 3.

PAL Conversion Procedure

Step 1 — Copy the PAL File

For your convenience, the PAL equation file used in this example has been created for you. The name of the file is “counter.PLD”. It is located in the XACT/Tutorial directory of the Xilinx software.

Copy this file into your working directory.

Step 2 — Run XDM

From the DOS or UNIX prompt, type:

```
XDM
```

The XDM Main Menu appears as shown in the Preface.

Use the mouse to highlight menu items and select using the left mouse button. Press F1 for help.

```
TITLE COUNTER EXAMPLE FILE
CHIP   COUNTER   22V10;
;Pins (not in pin number order)
     HCLK REGWR SELECT COUNTEN OUTPUTEN DD3 DD2 DD1 DD0 CARRY COL3
GND COL2 COL1 COL0
;Nodes
      LOAD HOLD COUNT
;COUNTER
EQUATIONS
LOAD = (REGWR * /COUNTEN);
HOLD = (REGWR * COUNTEN
     + /REGWR * /COUNTEN);
COUNT = (/REGWR * COUNTEN);
COL0 := (DD0 * LOAD
     + HOLD * COL0
     + COUNT * /COL0);
COL0.CLKF = (HCLK);
COL0.TRST = (SELECT * /OUTPUTEN);
COL1 := (DD1 * LOAD
     + HOLD * COL1
     + COUNT * /COL1 * COL0
     + COUNT * COL1 * /COL0);
COL1.CLKF = (HCLK);
COL1.TRST = (SELECT * /OUTPUTEN);
COL2 := (DD2 * LOAD
     + HOLD * COL2
     + COUNT * COL2 * /COL1
     + COUNT * /COL2 * COL1 * COL0
     + COUNT * COL2 * /COL0);
COL2.CLKF = (HCLK);
COL2.TRST = (SELECT * /OUTPUTEN);
COL3 := (DD3 * LOAD
     + HOLD * COL3
     + COUNT * COL3 * /COL2
     + COUNT * COL3 * /COL1
     + COUNT * /COL3 * COL2 * COL1 * COL0
     + COUNT * COL3 * /COL0);
COL3.CLKF = (HCLK);
COL3.TRST = (SELECT * /OUTPUTEN);
CARRY := (DD3 * DD2 * DD1 * DD0 * LOAD
     + /LOAD * COL3 * COL2 * COL1 * COL0);
CARRY.CLKF = (HCLK);
```

Figure 1-2 4-Bit Counter Equation File (counter.PLD)

Step 3 — Select the Device Family, Type, and Speed

From the XDM menu, select the PROFILE→FAMILY command. XDM displays the devices available in the XC7000 series, as shown in Figure 1-3.

Select "XC7300".

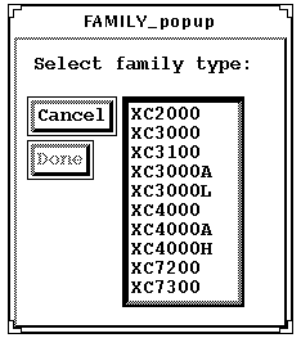


Figure 1-3 Device Family Menu

XDM then displays the part type selection menu as shown in Figure 1-4.

Select "7354PC44".

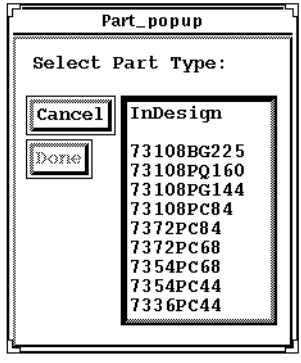


Figure 1-4 Part Type Selection Menu

XDM then displays the speed selection menu as shown in Figure 1-5.

Select "-10".

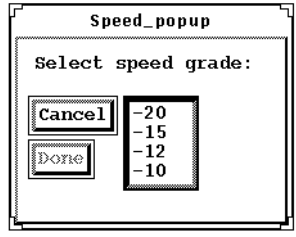


Figure 1-5 Speed Selection Menu

Notice that the "Family:" and "Part:" sections of the XDM main menu (lower left) are now changed to reflect your selections, as shown in Figure 1-6.

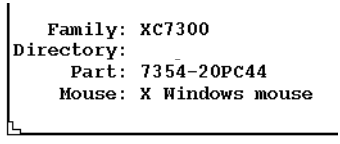


Figure 1-6 XDM Menu after Device Selection

You can also click on these fields directly to change the information.

Step 4 — Convert the PAL Equation File

a) Use the mouse to select the FITTER→PALCONVT command. You are prompted for a design name, as shown in Figure 1-7.

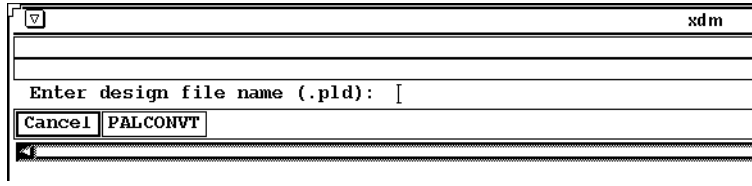


Figure 1-7 PALCONVT Design Name Prompt

b) Enter the design name: "n_design". This design name is used throughout this example. PALCONVT displays all PAL files in the working directory (all files with a .PLD or .PDS extension), as shown in Figure 1-8.

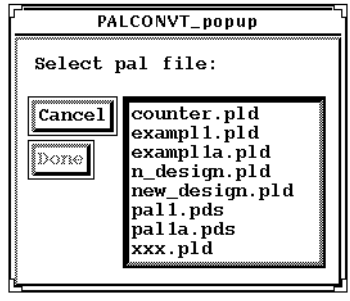


Figure 1-8 PAL File Selection Menu

c) Select "counter.PLD" and click DONE.

You are prompted to select one of two options as illustrated in Figure 1-9.

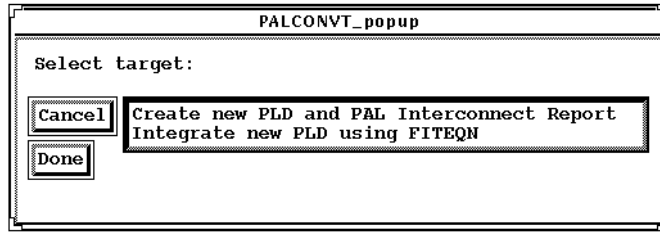


Figure 1-9 PALCONVT Target Selection Prompt

d) Click DONE. ("Create new PLD and PAL Interconnect Report" is the default option.)

PALCONVT automatically creates a Top-Level file for your design and includes the PAL equations (counter.PLD) as an Include File. Figure 1-10 illustrates the result.

```
PATTERN n_design.PLD  - file made by PALCNVT command

CHIP  n_design  XEPLD

INCLUDE_EQN 'counter.pld'

INPUTPIN HCLK REGWR SELECT COUNTEN OUTPUTEN DD3 DD2 DD1 DD0
OUTPUTPIN  CARRY

NODE COL3 COL2 COL1 COL0 LOAD HOLD COUNT

EQUATIONS
```

Figure 1-10 Top-Level File Created by PALCONVT

Step 5 — Compile the Design

a) Use the mouse to select the FITTER→FITEQN command. XEPLD displays the FITEQN_popup menu as shown in Figure 1-11.

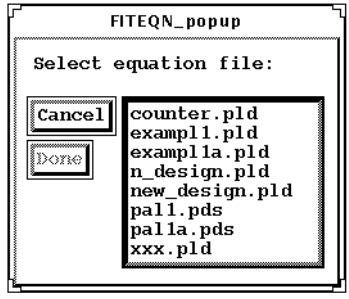


Figure 1-11 FITEQN_popup Menu

b) Select "n_design.PLD" and click DONE.

XEPLD displays the option selection menu as shown in Figure 1-12.

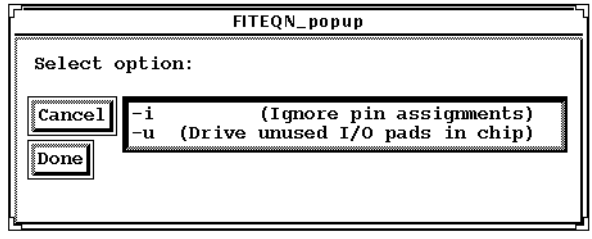


Figure 1-12 FITEQN Option Selection Menu

c) Click DONE. ("Ignore pin assignments" is the default option.)

XEPLD processes your design file, checks for errors, and creates reports. The compiled design is named "n_design.VMH".

Step 6 — Verify the Design

Use the mouse to select the UTILITIES→BROWSE command. You can review the following key reports to verify your design.

- n_design.ERR — the error log. All warning and error messages generated by XEPLD are listed here.
- n_design.INT — the PAL interconnect report showing each signal name and how it was used. Use this report to verify that each signal is connected properly.
- n_design.RES — the device logic and pin resource usage report. Use this report to verify that your design is placed into a device with the proper amount of resources.

Step 7 — Program a Device

a) Generate a programming bit-map file using the XDM VERIFY→MAKEPRG command (for Intel Hex format) or the VERIFY→MAKEJED command (for JEDEC format). The resulting menus are illustrated in Figure 1-13.

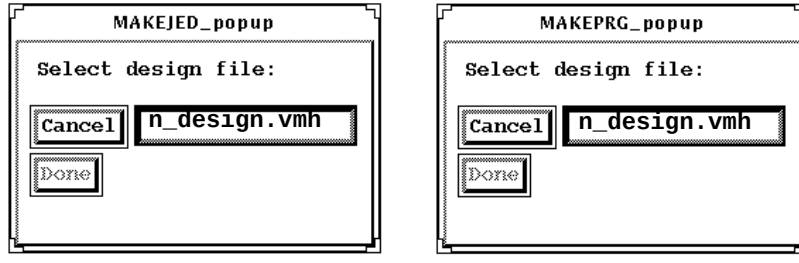


Figure 1-13 Device Programming Menus

b) Select "n_design.VMH" and click DONE.
You are prompted for a signature.

c) Enter "testprg.a" (or any unique identifying text of 8 characters or less). The ".a" extension identifies the signature as ASCII.

XEPLD creates a bit map file named "n_design.JED" or "n_design.PRG" containing the information required to program your design into the selected device.

Refer to your device programmer documentation for instructions on how to download the bit-map file.

Creating Designs with PLUSASM

This chapter demonstrates how to create behavioral designs using PLUSASM, the Xilinx native EPLD design language based on the PALASM2 equation syntax.

All files that come from PLD compilers or schematic entry tools, eventually get translated into PLUSASM by the XEPLD software prior to the fitting process. The fitter can optimize the PLUSASM code and achieve excellent results, or you can generate optimized PLUSASM source code that specifies exactly how the fitter should allocate resources by using attributes and property statements in your high-level design file.

PLUSASM Overview

Using an ASCII text editor, you can create a complete PLUSASM design within a single file, which contains both design control information and behavioral equations. However, it is often convenient to develop your design as separate Include Files and link them together within a Top-Level File to form a complete design. These two design approaches are illustrated below in Figure 2-1.

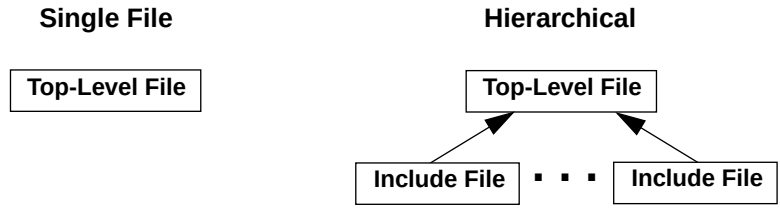


Figure 2-1 PLUSASM Design Approaches

PLUSASM File Structure

The basic structure of the PLUSASM Top-Level File and Include Files is illustrated in Figure 2-2. This figure shows a hierarchical design composed of a Top-Level File containing both embedded equations and three Include Files referenced by INCLUDE_EQN statements.

The file structure is identical for both the Top-Level File and the Include Files. However, the Include Files have a different CHIP statement syntax and they can contain only a limited subset of declaration statements. For detailed information on each PLUSASM command, see Chapter 4 of the *XEPLD Reference Guide*.

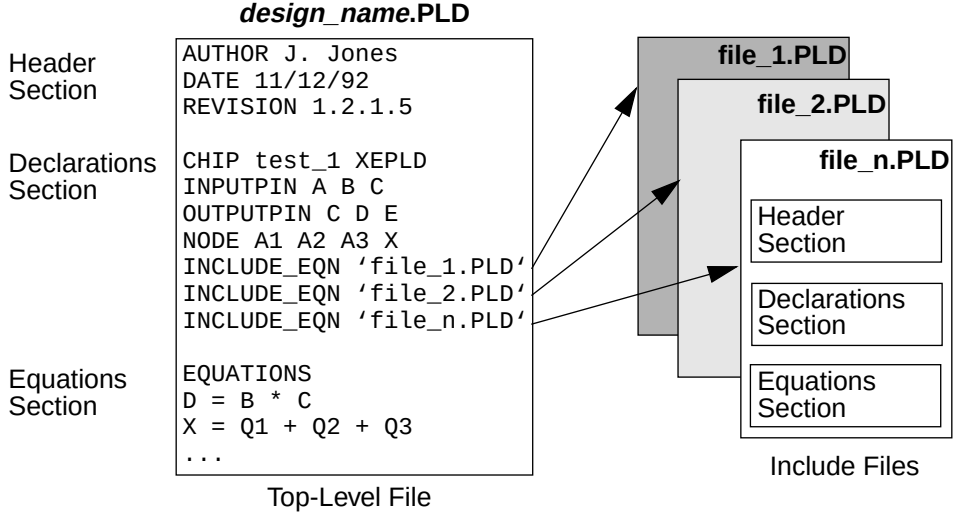


Figure 2-2 PLUSASM File Structure

The Header Section

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*

The Declarations Section

Declaration statements 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.

Declaration Statement	TOP	INC	Function Overview
CEPIN	X		Specifies the global Clock Enable input pins.
CHIP	X	X	Specifies the file type, file name, and pin list.
FASTCLOCK	X	X	Specifies the global FastCLK inputs.
FASTINPUT		X	Declares the signals connected to the FastInput pins.
FOEPIN	X		Specifies global FAST Output Enable input pins.
INCLUDE_EQN	X		Specifies names of Include Files.
INPUTPIN	X		Specifies device input pins.
IOPIN	X		Specifies device I/O pins.
LOGIC_OPT	X		Controls logic optimization.
MINIMIZE	X	X	Controls the use of logic minimization.
MRINPUT	X		Specifies the Master Reset input.
NODE	X	X	Specifies nodes in the design.
OPTIONS	X		Controls the automatic use of device resources.
OUTPUTPIN	X		Specifies device output pins.
PARTITION	X	X	Specifies physical locations for groups of equations.
PWR	X		Controls the device power usage.
STRING	X	X	Specifies a global text string substitution.

TOP = Used in Top-Level Files

INC = Used in Include Files and PLD Files in Schematics

The Equations Section

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

In a Top-Level File, this section is optional if you have equations specified in Include Files. However, you must include the EQUATIONS keyword even if there are no explicit equations in the file.

This section is mandatory in Include Files.

Creating Basic Designs

The Xilinx EPLD fitter automatically optimizes your equation files. It analyzes your design and automatically maps functions into the appropriate device resources without user intervention. The fitter implements dense, high-performance designs by taking advantage of the advanced XC7000 architectural features such as input pad registers, the Universal Interconnection Matrix, Dual-Block Architecture function blocks, and global control signals.

The intent of this section is to introduce you to the various features of the XC7000 architecture and show how you can control device resource allocation with PLUSASM when desired.

This section discusses how to develop optimized equations for each of the five basic structures used in a typical EPLD design:

- Input Pads.
- High Density Function Blocks.
- The Universal Interconnection Matrix.
- Fast Function Blocks.
- Output Pads.

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

Design Example

The following design example for a Pulse-Width Modulator (PWM) is used to illustrate the basic design concepts discussed in this chapter.

Figure 2-3 illustrates the logic of this design and how it is mapped into the basic XC7000 architectural blocks. The equation file for this schematic is shown in Figure 2-4.

The output of the PWM is a constant frequency with a variable high time, controlled by a loadable, 4-bit up-down counter. If the counter is loaded with the value P, the length of the down-count period is P clock periods and the length of the up-count period is (2^n+1)-P. The toggle period of the output flip-flop is 2^n+1, (the sum of the up-count and the down-count periods), which is independent of P. Therefore, the flip-flop output has a constant frequency with its high time proportional to P.

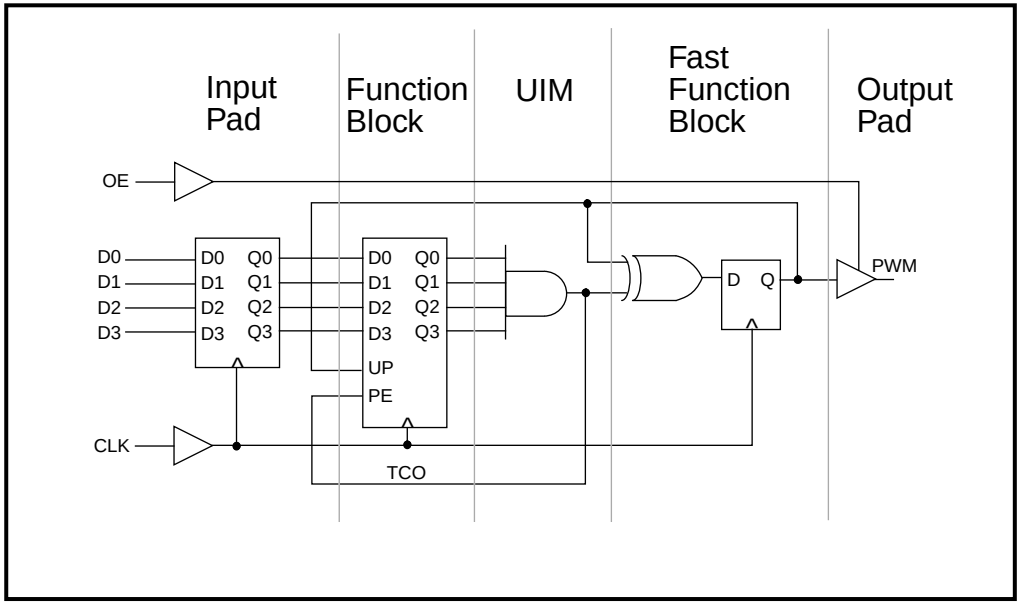


Figure 2-3 4-Bit Pulse Width Modulator

```
TITLE   NEWPWM
CHIP    NEWPWM  XEPLD

; pulse width modulation design

OUTPUTPIN               q0 q1 q2 q3
OUTPUTPIN (FOE = OE)    pwm
INPUTPIN (RCLK=CLK)     d0 d1 d2 d3
NODE (UIM)              tco
FASTCLOCK               clk
FOEPIN                  OE

PARTITION FFB           pwm

EQUATIONS

q3 := (q3 * /tco) :+: (d3 * tco
                      +  pwm *  q0 *  q1 *  q2 * /tco
                      + /pwm * /q0 * /q1 * /q2 * /tco)

q2 := (q2 * /tco) :+: (d2 * tco
                      +  pwm *  q0 *  q1 * /tco
                      + /pwm * /q0 * /q1 * /tco)

q1 := (q1 * /tco) :+: (d1 * tco
                      +  pwm *  q0 * /tco
                      + /pwm * /q0 * /tco)

q0 := (q0 * /tco) :+: (d0
                      + /tco)
pwm := pwm :+: tco

tco = (q0 * q1 * q2 * q3)
```

Figure 2-4 Pulse Width Modulator Equation File

Using Input Pad Structures

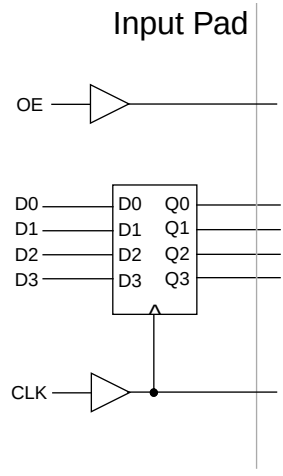
There are two types of input pads in XC7000-series devices: Dedicated input-only pads and I/O pads configured for input.

Input pads can be configured as follows:

- Registered.
- Registered with clock enable.
- Latched.
- Combinatorial (non-registered and non-latched).
- FastInput.

Registered Inputs

Specify registered inputs by adding the clock source modifier (RCLK=*clock_source*) to the INPUTPIN statement, as demonstrated in the following example:




```
INPUTPIN (RCLK=CLK) d0 d1 d2 d3
FASTCLOCK CLK
```

Figure 2-5 Registered Input Pad

Note: Input pad registers may only be clocked by the FastCLK inputs.

Note: The software automatically tries to assign single input registered equations to input pad registers to minimize the usage of macrocell resources where possible.

Registered Inputs with Clock Enable

Configure the registered input pads with clock enable by adding the clock enable (CE) modifier to the INPUTPIN statement as follows:

```
FASTCLOCK CLK
CEPIN /CKEN
INPUTPIN (RCLK=CLK CE=CKEN) D0 D1 D2 D3
```

Latched Inputs

Configure the input pads for latched operation by using the latch modifier (LE) in the INPUTPIN statement as follows:

```
FASTCLOCK ENAB
INPUTPIN (LE=ENAB) D0 D1 D2 D3
```

Combinatorial Inputs

Configure input pads for direct input to the UIM by omitting the clock source specification in the INPUTPIN Statement. For example, the following statement specifies direct input to the UIM:

```
INPUTPIN D0 D1 D2 D3
```

FastInputs

Each XC7300-series device has dedicated FastInput pins that input signals directly to the Fast Function Blocks, bypassing the UIM, and therefore requiring less delay. XC7300 devices for example have 12 dedicated FastInputs. To specify a FastInput use the (FI) modifier in the INPUTPIN statement. For example:

```
INPUTPIN (FI) D0 D1 D2 D3
```

In this example, all instances of D0, D1, D2, and D3 are used as FastInputs. As an alternative, you can specify the .FI extension (as in D0.FI) for only those instances that you want to use the FastInput path; all other instances are routed through the UIM.

Input Polarity

Configure active-low inputs by adding a "/" in front of the pin specification. For example, to invert the D1 input:

```
INPUTPIN (RCLK=CLK) D0 /D1 D2 D3
```

By inverting a signal name in the INPUTPIN statement, you effectively invert each instance of that input in the equations.

Using High-Density Function Blocks

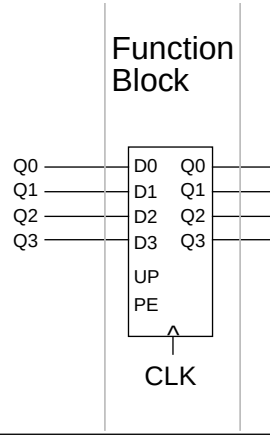
The XC7300-series devices have a combination of High Density Function Blocks and Fast Function Blocks; this is called the "Dual Block Architecture." High Density Function Blocks provide the maximum amount of logic resources for use in your design.

High Density Function Blocks can implement the following:

- Registered Equations.
- Combinatorial Equations.

Registered Equations

Specify registered equations by using the ":=" operator, as shown in the following example:




```
EQUATIONS
q3 := (q3 * /tco) :+: (d3*tco
                      +pwm*q0*q1*q2*/tco
                      +/pwm*/q0*/q1*/q2*/tco)
...
```

Figure 2-6 Registered Equations for HDFBs

Registered Equation Clock Definition

Use the .CLKF equation to specify the clock source for a registered equation. Clocks may be any ordinary input or logical signal (macrocell feedback) in the design, which can be implemented using the macrocell's product term clock. You may also specify a FastCLK signal which is declared in a FASTCLOCK statement. If you do not specify a .CLKF equation, the software uses the first signal named in a FASTCLOCK statement, by default.

Note: The software automatically assigns single-input rising edge clock signals to FastCLKs where possible.

Register Preload Values

Registers in High Density Function Blocks can be configured to power-up to either a logical 0 or 1. Use the .PRLD equation to specify GND or VCC as the power-on value as follows:

```
Q1.PRLD = VCC; preload the Q1 output to a logic 1
```

If you do not specify a preload value, the software will normally assume a logical 0 (GND) but will change it when necessary to allow optimization of the register into an input pad or FFB. If you specify a preload value, you may inhibit the software from performing optimization on that equation.

Macrocell Control Equations

Configure registered equations for asynchronous clear by using the .RSTF extension. For example:

```
Qn.RSTF=CLEAR
```

Configure registered equations for an asynchronous SET input by using the .SETF extension. For example:

```
Qn.SETF=SET
```

Create (product term) 3-state control signals by using the .TRST extension. For example:

```
X.TRST = TRISTATE
```

Note: You can also control 3-state signals by using the global FOEPIN signals. The software automatically assigns single-input active-high .TRST equations to the global FOE signals where possible.

Combinatorial Equations

Specify combinatorial equations by using the "=" operator. For example:

```
tco = q0 * q1 * q2 * q3
```

Manual Placement of Equations

To manually place one or more equations into any High Density Function Block use the PARTITION statement, as follows:

```
PARTITION FB signal_name1 signal_name2 ...
```

To manually place one or more equations into a specific High Density Function Block, (in this example, FB5) use the PARTITION statement, as follows:

```
PARTITION FB5 signal_name1 signal_name2 ...
```

To manually place one or more equations consecutively into a specific High Density Function Block, beginning with a specific macrocell, (in this example, FB5, macrocell 3) use the PARTITION statement, as follows:

```
PARTITION FB5_3 signal_name1 signal_name2 ...
```

Note: The software automatically partitions your design; manual partitioning may prevent the software from producing the most efficient mapping of your design.

Using the Universal Interconnection Matrix (UIM)

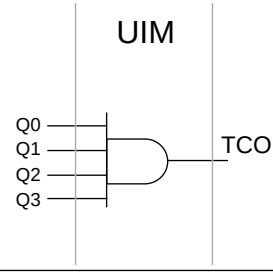
The UIM provides a 100% interconnection matrix allowing any output to drive any input in the device; routing is never blocked. The wired-AND capability of the UIM also allows it to implement logic functions which incur no additional delay.

Using UIM Interconnections

All macrocell inputs (except for the FastInputs) come from the UIM. No special syntax is required to specify these connections which are implied in your equations.

Using Wired-AND Functions

The UIM is capable of performing wired-AND functions which are automatically used by the software when possible to improve resource utilization. To manually specify a UIM AND function, assign the associated nodes to the UIM by using a NODE (UIM) statement. In the Pulse Width Modulator example, the carry-out signal is created by ANDing the Q0-Q3 outputs in the UIM, as shown below:




```
NODE (UIM) TCO
EQUATIONS
tco = q0 * q1 * q2 * q3
```

Figure 2-7 Specifying UIM AND Gates

Note: In most cases there is no need to manually specify UIM AND gates in your design. Your results are usually better if the software is allowed to select which functions to move into the UIM.

XEPLD automatically uses the UIM whenever possible to increase the available macrocell resources. You can disable the UIM optimization function for your whole design by using the OPTIONS statement. For example, to turn off optimization for the whole design:

```
OPTIONS OFF UIM_OPT ; Turn off UIM optimization
```

Signal Polarity

You can create DeMorgan equivalent logical functions (OR, NOR, NAND and so on) by inverting the inputs or outputs of the UIM AND function. For example, to create the logical OR function in the UIM:

```
NODE (UIM) OR_GATE; place OR_GATE into the UIM
EQUATIONS
/OR_GATE = /Q0*/Q1*/Q2*/Q3; a logic OR function
```

Note: If you try to place an equation into the UIM that cannot be implemented due to the lack of signal inversions, the equation is automatically placed into a macrocell instead and the software issues a warning.

Using Fast Function Blocks

The XC7000-series devices have a combination of Fast Function Blocks and High Density Function Blocks. Logic placed into the Fast Function Blocks performs faster than logic placed into High Density Function Blocks.

Fast Function Blocks can implement the following:

- Combinatorial Equations.
- Registered Equations.

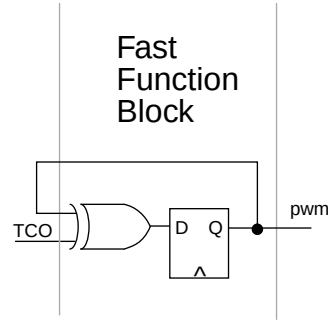
Combinatorial Equations

Specify combinatorial equations by using the "=" operator. For example:

```
TCO = q0 * q1 * q2 * q3
```

Registered Equations

Specify registered equations by using the ":=" operator, as demonstrated in the following example:




```
pwm := pwm :+: tco
```

Figure 2-8 FFB Registered Equations

Note: The preload value for registers in a Fast Function Block is VCC (high), unless you are using the reset input to the register. If you are using the reset input (only available in the XC7336) the preload value is GND (low).

Automatic Placement of Equations

After optimization, XEPLD automatically maps functions that meet the following requirements into the available Fast Function Blocks:

- All clocks must be able to map onto global FastCLK signals.
- All 3-state controls must be able to map onto global FOE signals.
- All registers are asynchronously set only (not reset), except for the XC7336 devices which are selectable for either set or reset.
- All registers must *not* specify a low (GND) preload value, except for the XC7336. (If you use the reset input to a register in the XC7336, the preload value must be GND.)
- All logic must use 4 or less p-terms when implemented with active low outputs. (except for the XC7336 devices, which support either active low or active high outputs.)

Note: If your equations require more than 4 product terms, the XEPLD software will automatically partition your equations into groups of 4 and export the output to adjacent macrocells using the .EXPORT equation. This allows equations using more than 4 product terms to be implemented in an FFB.

Manual Placement of Equations

To manually place one or more equations into any FFB use the PARTITION statement, as follows:

```
PARTITION FFB signal_name1 signal_name2 ...
```

To manually place one or more equations into a specific FFB, (in this example, FB2), use the PARTITION statement, as follows:

```
PARTITION FB2 signal_name1 signal_name2 ...
```

To manually place one or more equations into a specific FFB, beginning with a specific macrocell, (in this example, FB2, macrocell 3), use the PARTITION statement, as follows:

```
PARTITION FB2_3 signal_name1 signal_name2 ...
```

Note: In the XC7300 devices, FB1 and FB2 are actually Fast Function Blocks. Specifying FB1 or FB2 in a PARTITION statement implies using an FFB and not a High Density Function Block.

Note: FFBs do not have ALU resources. Therefore, any .D1 or .D2 equations are converted to their sum-of-products form by the software as needed, to fit equations into Fast Function Blocks.

Using Output Pad Structures

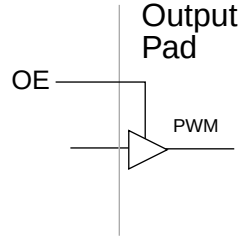
There are two types of output pads for XC7300-series devices: High Density Function Block Outputs (standard drive capability) and Fast Function Block Outputs (high drive capability).

Each of these outputs can be configured for:

- 3-state with product term (logic) control.
- 3-state with FOEPIN control.
- 3-state with both FOEPIN and product term control.
- Direct (always on).

3-State Outputs using Global FOE

Control 3-state outputs with the global FOEPIN inputs by using the FOE modifier in the OUTPUTPIN statement, as demonstrated in the following example:




```
OUTPUTPIN (FOE = OE) pwm
FOEPIN OE
```

Figure 2-9 3-state Control Using FOE

Note: The Global FOE capability is not available in the XC7272.

3-State Outputs using Product Term Control

Control 3-state outputs with logic signals by using the .TRST extension. For example:

```
OUTPUT := TCO
OUTPUT.TRST = X ; X is the 3-state control
```

Note: The software automatically assigns single-input active-high .TRST functions to the global FOE pins where possible.

Note: Product term 3-state control is only valid in High Density Function Blocks; .TRST equations cannot be specified for outputs assigned to FFBs.

Direct Outputs

Configure outputs as always-on (no 3-state capability) by using the OUTPUTPIN statement with no FOE modifiers. For example:

```
OUTPUTPIN D0 D1 D2 D3; Outputs are not 3-state
```

Specifying Feedback Paths

There are two types of signal feedback to the UIM:

- Pin Feedback (from the device pin).
- Macrocell Feedback (from the macrocell outputs).

The PINFBK option in the IOPIN pin statement can be used to globally control the feedback options.

Pin Feedback

Use the PINFBK option of the IOPIN statements to specify that the signal feedback always comes from the device pin. For example:

```
IOPIN (PINFBK) A
Q := A; A comes from the pin
```

Macrocell Feedback

Macrocell feedback is routed through the UIM and is independent of the state of the associated device pin. For example:

```
IOPIN A
Q0:=A; A comes from the macrocell - UIM feedback
Q1:=A.PIN; A comes from the pin - Pin feedback
```

UIM feedback is the default unless otherwise specified.

Specifying 3-State Options

There are two types of product term 3-state control:

- Pin 3-state.
- Node 3-state.

Pin 3-State

When using pin 3-state, the product term controls the 3-state condition of the pin only and not the macrocell feedback to the UIM. For example:

```
OUTPUTPIN Q
Q := A; Q is always fed back to the UIM
Q.TRST = X
```

Pin 3-state is the default for all devices other than the XC7272. When the equation above is implemented in an XC7272 however, the macrocell feedback to the UIM is disabled (a logic 1 state) when the pin is in a 3-state condition.

Node 3-State

Node 3-state is used to emulate 3-state busses in the UIM. However, it is usually better to implement multiplexers instead of 3-state busses because the UIM feedback is lost (disabled in a logic 1 state) when the node is in a 3-state condition. For example:

```
NODE (NODETRST) A B
NODE (UIM) AND_GATE
OUTPUTPIN C
EQUATIONS
A = SIGNAL0 ; A = 1 when X is not true
B = SIGNAL1 ; B = 1 when X is true
A.TRST = X
B.TRST = /X
AND_GATE = A*B ; UIM AND gate
C = AND_GATE
```

Using both Product Term and FOE 3-State Control

If both product term and FOEPIN 3-state control signals are specified for an output, the software automatically ANDs the signals together. Both control signals must be active (high) to enable the pin as shown in Figure 2-10 below. If either control signal is inactive (low), the output pin is forced into a 3-state condition.

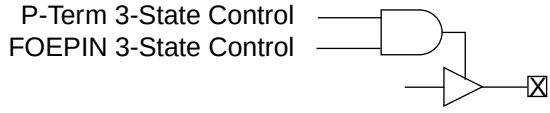


Figure 2-10 Using both P-Term and FOE 3-State Control

Note: This 3-state control AND function is inherent in the device and does not use macrocell or UIM-AND resources.

Converting PAL Designs

This chapter shows you how to use PAL equation files to develop designs for Xilinx EPLDs. This is a simple process because the internal architecture of Xilinx EPLDs is similar to PALs and because the XEPLD software automatically processes your PAL equation files, creating a Top-Level File ready for compilation. Figure 3-1 illustrates the PAL design conversion process.

Also see Chapter 1 for a brief example illustrating the use of PAL equation files.

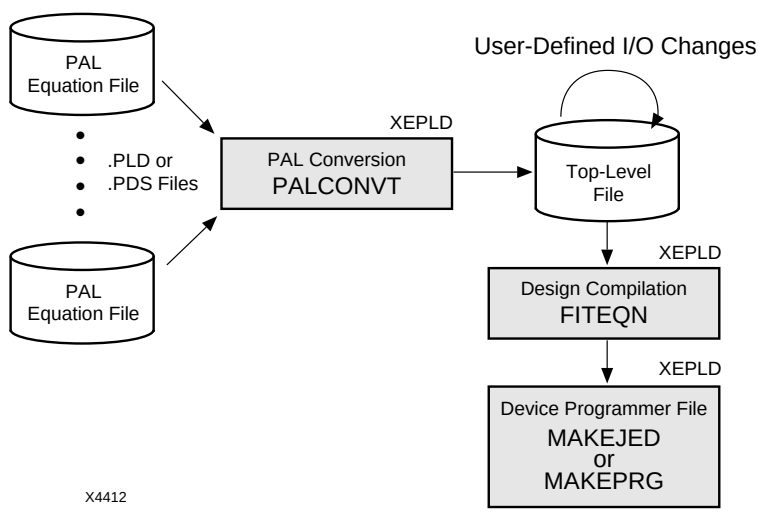


Figure 3-1 PAL File Conversion Diagram

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, CUPL, 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 or the CUPL -c compiler option), you can easily generate PLUSASM-compatible equation files. These equation files are then mapped into the Xilinx EPLD architecture by the XEPLD software.

PAL Conversion Methodology Overview

Use the FITTER→PALCONVT command to combine multiple PAL equation files into a single coherent PLUSASM design. The PALCONVT program reads your PAL files and analyzes their I/O signal configurations, producing a PLUSASM Top-Level File which specifies an interpretation of all signals. Your PAL equation files become Include Files in the Top-Level File.

PALCONVT also resolves any polarity conflicts between PALs, resolves any PAL-specific functionality (for 22V10s and 20V8s), and creates a PAL Interconnect Report summarizing the usage and interconnection of all signals in the design and in the EPLD device I/O interface. This report helps you choose a target device that matches your requirements. See the *XEPLD Reference Guide* for the details of 22V10 and 20V8 support.

Pin and Node Assignment

PALCONVT automatically assigns I/O pin and node declaration statements to the signals in your PAL equation files, based on their use in the circuit. These declarations are written to the Top-Level File.

- Signals used only as equation inputs are assigned to INPUTPIN statements.
- Signals used only as equation outputs are assigned to OUTPUTPIN statements.
- Signals used as both equation inputs and outputs, that have no .TRST control inputs, are assigned to NODE statements.
- Signals used as outputs, that also have .TRST control inputs, are

assigned to OUTPUTPIN statements, regardless of whether they are also used as an input.

You can change any of these assigned nodes to output or I/O pins by modifying the declaration statements in the Top-Level File after running PALCONVT. The default assignments are illustrated below in Table 3-1.

Table 3-1 Automatic Assignment of PAL Signals

PAL Signals	Default Assignment
Signals appearing only on the right side of equations	INPUTPIN
Signals appearing only on the left side of equations	OUTPUTPIN
Signals appearing on both sides (L & R) of equations and have no .TRST control	NODE
Signals used as output that also have .TRST control	OUTPUTPIN

PALCONVT also processes PLUSASM files that were not targeted for PALs. This includes any equation file prepared for a PLD or custom symbol in a schematic design. If your equation files include XEPLD-specific declaration statements, they are automatically assigned as shown in Table 3-2.

Table 3-2 XEPLD Declaration Statement Assignments

XEPLD-Specific Declarations in Include Files	Assignment
FASTCLOCK	FASTCLOCK
NODETRST ON	OUTPUTPIN (NODETRST)
NODE (UIM)	NODE (UIM)
FASTINPUT	INPUTPIN (FI)
PARTITION	PARTITION

FASTCLOCK and FOEPIN Assignment

XEPLD automatically assigns the most frequently used rising-edge clock inputs to FastCLK nets and the most frequently used active-high output enable inputs to FOE nets. See "Editing the Top-Level File" later in this chapter for more information on FASTCLOCK and FOEPIN assignment.

Note: If your clocks or output enable controls are connected to combinatorial logic in your design, they cannot be assigned to the FastCLK or FOE signals which use dedicated wiring in the device that is not accessible to the macrocell logic.

Signal Polarity Conflict Resolution

PALCONVT automatically modifies equations to resolve conflicts in signal polarity declarations. The software automatically inverts all instances of a signal within a PAL file if the signal is declared with active-low polarity in the PAL file's pinlist but is also declared active-high in the pinlist of another PAL file.

PAL Conversion Requirements

You can use any PAL equation file (or behavioral design file) that conforms to PLUSASM syntax (.PLD or .PDS). PLUSASM is based on PALASM and most PAL equation files that are output in PALASM2 Boolean equation format are PLUSASM compatible.

Note: You can recompile ABEL-based designs with Xilinx ABEL, and the resulting files will be PLUSASM compatible.

See the PLUSASM section of the *XEPLD Reference Guide* for a complete description of PLUSASM.

Using 22V10 and 20V8 Files

PALCONVT automatically processes 22V10 and 20V8 PAL files including the implied features (such as default clocks, 3-state controls, global set/reset, and so on). See Appendix B of the *XEPLD Reference Guide* for complete details of 22V10 and 20V8 support.

Using Generic PAL Files

If the CHIP statement of any PAL equation file contains a PAL device type other than “22V10” or “20V8”, or if the CHIP statement contains the keywords, COMPONENT, PLFB9, or PLFFB9, XEPLD will assume that the equation file is for a generic PAL and will process the equations as a behavioral design. All equations must be explicitly defined in PLUSASM syntax; your generic files must not include any implied functions (such as 3-state controls, inversions, and so on). See Appendix B of the *XEPLD Reference Guide* for more information on generic PAL support.

Generic equations may also be obtained when a device-independent behavioral design is processed by a PLD compiler. If the resulting PLUSASM-compatible equation file contains no CHIP statement or if the CHIP statement omits the target PLD type, you must specify a CHIP statement in the file with an unrecognized target PLD type such as the following:

```
CHIP my_pal 16R8; 16R8 is unrecognized
                ; and defaults to generic
```

Note: The signal on pin 1 (of a PAL equation file) is assumed to be the default clock for any registered equation output that does not have an explicit clock definition.

The PAL Conversion Procedure

The procedure for preparing one or more PAL equation files for use in a Xilinx EPLD is illustrated in Figure 3-2. See Chapter 1 for a brief PAL conversion example including the menus that appear at each step.

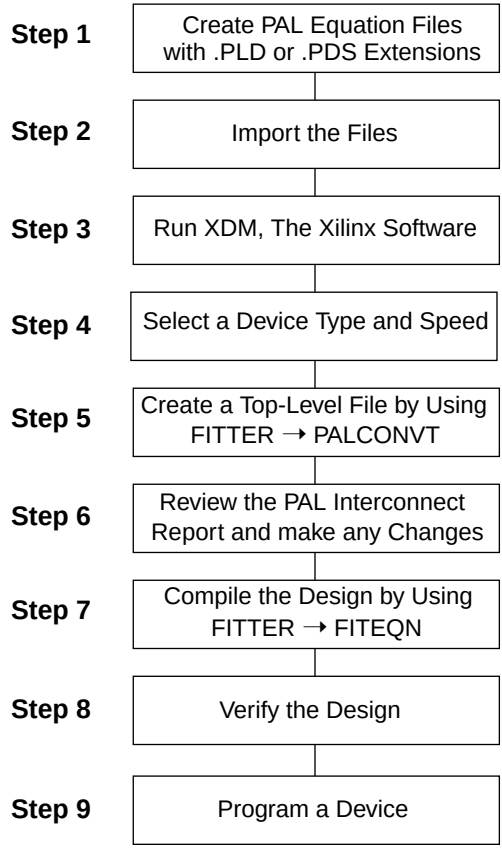


Figure 3-2 Pal Conversion Methodology

Step 1 — Create .PLD or .PDS Files

For each PAL in the design, generate a PALASM2 boolean equation file from the original PAL source code. For example, use the ABEL "XFER" utility or the CUPL "-C" compiler option. All PAL equation files must have either a .PLD or .PDS file extension.

Step 2 — Import the PAL Files

Copy the PAL equation files into a design directory. It is recommended that you *not* use the /XACT directory for designs.

Note: This is a good time to verify that the signal names in each PAL pinlist establish the proper signal connectivity for the design.

Step 3 — Run XDM

XDM is the Xilinx Design Manager program contained in XEPLD.

a) To run XDM, at the system prompt type:

```
XDM
```

The XDM main menu appears.

Step 4 — Select a Device Type and Speed

Choose an EPLD device type and speed based on your design requirements.

a) Open the device family menu by selecting the PROFILE→ FAMILY command from the XDM main menu. XDM displays the various Xilinx device families available to you.

b) Select either "XC7200" or "XC7300." XDM displays the part type selection menu.

c) Select a device type. For example, select 7336PC44. XDM displays a speed selection menu.

d) Select a device speed grade. For example, select "–7".

Notice that the "Family:" and "Part:" sections of the XDM main menu (lower left) change to reflect your selections.

Step 5 — Create a Top-Level File

The PALCONVT program automatically creates a Top-Level File for you.

a) Run PALCONVT by executing the FITTER→PALCONVT command from the XDM main menu. You are prompted for a design name.

b) Enter a design name. If the file name already exists, you are asked if you want to overwrite the file. XDM then lists all .PLD and .PDS files in the design directory.

c) Select one or more PAL files to include in the design, and click "DONE." You are prompted for a target selection.

d) If you select Option One "Create new PLD and PAL Interconnect Report", PALCONVT reads the equation files and creates a Top-Level File. Each selected PAL equation file name is specified in an INCLUDE_EQN statement. The pinlists from each PAL file are merged together and the Top-Level File is saved as *design_name*.PLD. See Figure 3-7 for an example of a Top-Level File.

Note: Option One is recommended.

If you select Option Two "Integrate new PLD using FITEQN", XEPLD runs PALCONVT (as in option one) and compiles your design using FITEQN. Inspect the Top-Level File to verify that the node and I/O pin assignments created by the conversion process are correct. Then proceed to Step 8 "Verify the Design".

If there is a conflict between PAL files, PALCONVT displays an error message and halts. The PAL Interconnect report (*design_name*.INT) will indicate the conflict. Change the PAL source file and run PALCONVT until the design is free of conflicts.

Step 6 — Edit the Top-Level File

After reviewing the PAL Interconnect Report, you can use a text editor to edit the Top-Level File (*design_name*.PLD) and make any necessary changes to the automatic assignments made by PALCONVT.

a) Resolve any signal name inconsistencies to establish the correct interconnection between PAL files.

b) Change the default node and I/O pin assignments created by the automatic conversion process.

c) Specify any global control signals you want to use (FastCLK, FOE).

Note: When you are finished editing, be sure to save the changes before proceeding to the next step.

Step 7 — Compile the Design

When your Top-Level File is complete you are ready to compile your design, creating a physical device layout.

a) Run the compiler by selecting the FITTER→FITEQN command, which displays all .PLD files in the working directory.

b) Select your design file (*design_name*.PLD). XEPLD creates the *design_name*.VMH file which contains a physical description of your logical design as it is mapped into the specified EPLD. XEPLD also creates the reports used to verify your design.

Step 8 — Verify the Design

Use the reports created by the compiler to verify that your design fits within the specified device. Choose the reports you want to read by using a text editor, or by selecting the UTILITIES→BROWSE command. See Chapter 5 for a discussion of design verification.

The primary reports are:

- *design_name*.RES — The resource usage report.
- *design_name*.PIN — The pin assignments for your design.
- *design_name*.ERR — The warning and error log.
- *design_name*.EQN — The optimized equation file.

Step 9 — Program the Device

Generate a device programming bit-map file by using the VERIFY→MAKEPRG command (for Intel Hex format) or the VERIFY→MAKEJED command (for JEDEC format). Refer to your device programmer documentation for instructions on how to download the bit-map file.

Verifying PAL Conversion

PALCONVT creates a PAL Interconnect Report (*design_name*.INT) listing the signals used by each PAL in your design. Use this report to verify that the signal names in the pinlist of each PAL Include File establish the required connectivity between the PAL files.

The following section presents a PAL conversion example. The PAL Interconnect Report for the example design is shown in Figure 3-6. The "Connectivity" column of the Interconnect Report shows the interconnections for each PAL pin. Pins that do not connect to another PAL are marked "External Only." Pins that are used only for feedback within the same PAL are marked "Internal Only." Duplicate output signal names are marked "Conflict *PAL_name*:*pin_number*" to help you easily identify which PAL is generating the conflict.

You can use a text editor with a global search and replace function to edit the PAL Include File pinlist and equations. Once you have established the proper signal connectivity run PALCONVT once again.

An alternative to editing the PAL Include Files is to create a schematic using Xilinx PAL library components to establish the proper connectivity.

Editing the Top-Level File

The PALCONVT software creates a Top-Level File based on its assumptions about your overall design. In some cases you will need to manually edit the Top-Level File and reassign some of the signals assigned to NODE statements. These signals will be reassigned to OUTPUTPIN or IOPIN statements.

The XEPLD fitter will read the Top-Level File and the Include Files, optimize the design, and achieve excellent results. For example, the fitter will automatically collapse your logic and take advantage of global clock nets and global output enable control nets, even though you have not specified them in the Top-Level File. However, if you have any speed-critical paths or bi-directional signals in your design you must specify how the fitter is to handle them.

The following example illustrates how to edit the Top-Level File if you want to map speed-critical signals into Fast Function Blocks and FastInput pins, and how to handle bi-directional signals.

PAL Conversion Example

An example PAL design is illustrated in Figure 3-3. The PAL equation files are shown in Figure 3-4 and Figure 3-5. The PAL Interconnection Report is shown in Figure 3-6. The Top-Level File created by PALCONVT is shown in Figure 3-7. The edited Top-Level File is shown in Figure 3-8.

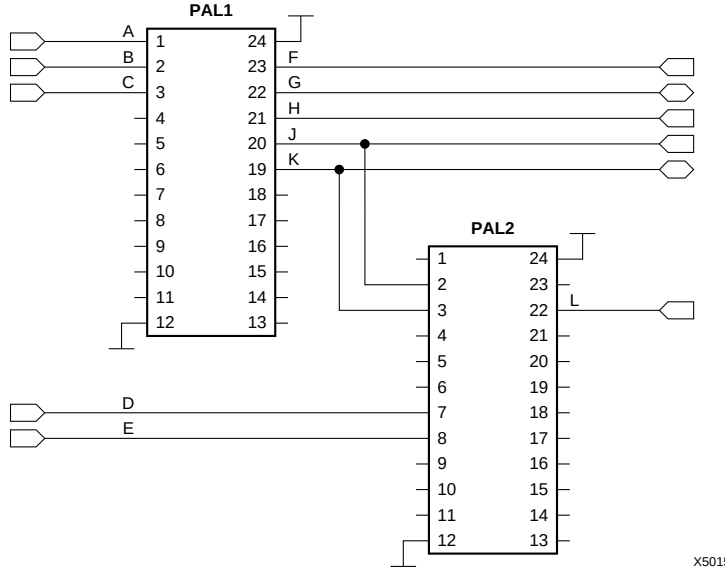


Figure 3-3 The Original PAL Design Example

```
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
; PALCONVT DESIGN EXAMPLE PAL1
EQUATIONS
F := (B);
G:= (F);
G.TRST = C;
H := (G);
J := (B * K)
K := (B);
K.TRST = (C);
```

Figure 3-4 PAL1 Equation File

```
TITLE PAL2
CHIP   PAL2  P22V10
;PINLIST (Highest Pin Number = 24)
NC J K NC NC NC D E NC NC NC NC NC NC NC NC NC NC NC L NC NC
; PALCONVT DESIGN EXAMPLE PAL2
EQUATIONS
L = (J * K * E);
L.TRST = (D);
```

Figure 3-5 PAL2 Equation File

```
XEPLD, Version                                          Xilinx Inc.
                        PAL INTERCONNECT REPORT
 Circuit name: EXAMPL1
Target Device: 7310884        Integrated: 10-25-93,  6:26PM

PAL File: PAL2
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

PAL  Signal    PAL   CHIP
PIN  Name      Use   Use        Connectivity
 2   J         I     NODE       From PAL1:20
 3   K         I     NODE       From PAL1:19
 7   D         I     INPUTPIN   External Only
 8   E         I     INPUTPIN   External Only
22   L         O     OUTPUTPIN  External Only

Unconnected pins:  1 4 5 6 9 10 11 12 13 14 15 16 17 18 19 20 21

PAL File: PAL1
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

PAL  Signal    PAL   CHIP
PIN  Name      Use   Use        Connectivity
 1   A         I     INPUTPIN   External Only
 2   B         I     INPUTPIN   External Only
 3   C         I     INPUTPIN   External Only
19   K         O,FBK NODE       To PAL2:3
20   J         O     NODE       To PAL2:2
21   H         O     OUTPUTPIN  External Only
22   G         O,FBK NODE       Internal Only
23   F         O,FBK NODE       Internal Only

Unconnected pins:  4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

Total number of Output Equations specified  = 6
Total number of device pins used for Input  = 5
Total number of device pins used for Output = 2
Total number of device pins used for I/O    = 0
Total number of Global Control pins used    = 0
```

Figure 3-6 PAL Interconnect Report Example

```
PATTERN exampl1.pld  - file made by PALCNVT command
CHIP  exampl1  XEPLD
INCLUDE_EQN 'pal2.pds'
INCLUDE_EQN 'pal1.pds'
INPUTPIN D E A B C
OUTPUTPIN  L H
NODE J K G F
EQUATIONS
```

Figure 3-7 Example Top-Level File Created by PALCONVT

```
PATTERN exampl1a.pld  - file made by PALCNVT command

CHIP  exampl1a  XEPLD

INCLUDE_EQN 'pal1.pds'
;INCLUDE_EQN 'pal2.pds'  PAL2 equations copied into Top Level
;File to specify pin feedback only for K input of PAL2

FASTCLOCK                  A         ; Changed from INPUTPIN
INPUTPIN                   B C
FOEPIN                     D         ; Changed from INPUTPIN
INPUTPIN (FI)              E         ; Changed from INPUTPIN to
                                     ; fast input
OUTPUTPIN                  F H J     ; Changed from NODE
IOPIN (PINFBK)           G         ; Changed from NODE
IOPIN                    K         ; Changed from NODE
OUTPUTPIN (FOE=D)          L         ; Change from product term
                                     ;to FOE enable

PARTITION FFB L                     ; Place L in a fast function
                                     ;block
EQUATIONS

L = (J * K.PIN * E)                  ; Use pin feedback instead
                                     ;of macrocell feedback
                                     ;for K
```

Figure 3-8 Example Top-Level File (edited)

Assigning Nodes to Outputs

The PALCONVT software interprets all signals that appear on both sides of the equations as nodes and assigns them to NODE statements in the Top-Level File. There are four instances where you will need to change these NODE declarations:

- PAL outputs that drive other PAL inputs and also drive EPLD outputs, must be changed to OUTPUTPIN statements.
- PAL outputs that are used as feedback in the same PAL and also drive EPLD outputs, must be changed to OUTPUTPIN statements.
- External bi-directional signals using pin feedback must be changed to IOPIN (PINFBK) statements.
- External bi-directional signals using both pin feedback and internal macrocell feedback must be changed to IOPIN statements; the instances using pin feedback must be identified using the .PIN extension.

These changes are described in the following sections.

Interconnections Between PALs

Interconnecting signals between PALs are assumed to be nodes. In the PAL conversion example, PALCONVT assigned signal J to a NODE because it appears on both sides of the equations:

```
J := (B * K); PAL1
L = (J * K * B); PAL2
```

This signal must be removed from the NODE statement and placed into an OUTPUTPIN statement in the Top-Level File as shown in Figure 3-8.

PAL Outputs Used as Feedback in the Same PAL

PAL outputs that drive internal logic feedback are assumed to be nodes. In the PAL conversion example, PALCONVT assigned signal F to a NODE instead of a pin because signal F appears on both sides of the equations:

```
F := (B); PAL1
G := (F); PAL1
```

This signal must be removed from the NODE statement and placed into an OUTPUTPIN statement in the Top-Level File as shown in Figure 3-8.

External Bi-Directional — PIN Feedback

Bi-directional signals are always assumed to be nodes because they appear on both sides of equations. In the PAL conversion example, PALCONVT assigned signal G to a NODE instead of a pin because signal G is used both as an input and as an output:

```
G := (F); PAL1
H := (G); PAL1
G.TRST = C; PAL1
```

Signal G must be moved from the NODE statement to an IOPIN statement in the Top-Level File as shown in Figure 3-8.

Use the PINFBK option in the IOPIN statement to specify that the feedback always comes from the XC7000 device pin. For example:

```
IOPIN (PINFBK) G
```

External Bi-Directional — Macrocell and PIN Feedback

Bi-directional signals are always assumed to be nodes. In the PAL conversion example, PALCONVT assigned signal K to a NODE instead of a pin because signal K is used for both input and output:

```
K := (B); PAL 1
L = (J * K * E); PAL 2
K.TRST = (C); PAL1
```

To change signal K to an XC7000 I/O signal, and to use both macrocell feedback (to PAL1) and pin feedback (to PAL 2), do the following:

1. Declare signal K as an IOPIN in the Top-Level File.
2. Change all occurrences of K on the right side of the equations to K.PIN (in PAL2.PDS). Move the equations into the Top-Level File.

Note: An Include File must have at least one equation. In this example, when the equations containing K are moved to the Top-Level File, PAL2.PDS will contain no equations and therefore PAL2 is no longer included.

Assigning Output Enable Signals to FOE Nets

Changing any 3-state output enable signals from product term control (expressed as a .TRST control equation using UIM interconnections) to a global fast output enable (FOE) net results in decreased resource requirements and decreased output enable delays.

Note: The XEPLD optimization software automatically assigns the most frequently used output enable signals to the FOE nets whenever possible without user intervention. However, you can also manually assign FOE signals by editing the Top-Level File.

In the PAL conversion example, to specify signal D as an FOE control for output L, make the following changes to the Top-Level File:

1. Add the "FOE = D" option to the OUTPUTPIN statement of the output (L) to be controlled:

   ```
   OUTPUTPIN (FOE=D) L
   ```

2. Add the statement:

   ```
   FOEPIN D
   ```

3. Delete signal D from the INPUTPIN statement:

   ```
   INPUTPIN E A B C
   ```

4. Delete the .TRST control equation (L.TRST = D) in the Include File.

Note: You cannot use the FOEPIN signal as a logic input; FOEPIN affects pin feedback but not UIM feedback. Only output enable signals that can be controlled by a single active-high pin may be assigned to an FOE net. Active-low output enable signals must come from a product term (macrocell).

Assigning Clock Signals to FastCLK Nets

Changing clock signals from a product term clock (expressed as a .CLKF control equation) to a global FastCLK net will give you decreased clock-to-output delays and reduced resource requirements.

Note: The XEPLD optimization software automatically assigns the most frequently used clock signals to the FastCLK nets whenever possible. However, you can also manually assign signals to FastCLK

nets by editing the Top-Level File.

In the PAL conversion example, to specify signal A as a FastCLK, make the following changes to the Top-Level File:

1. Add the statement:

   ```
   FASTCLOCK A
   ```

2. Delete signal A from the INPUTPIN statement:

   ```
   INPUTPIN E B C
   ```

Note: The XC7000 architecture does not allow the same pin to be used as both a FASTCLOCK and a logic input.

Assigning Equations to Fast Function Blocks

Fast Function Blocks (FFBs) are faster than High Density Function Blocks and provide increased output drive capability; the software automatically assigns output functions to FFBs when possible. You can manually assign output signals and nodes to the FFBs by using PARTITION statements. You can assign FastInput signals to the FFBs by using the INPUTPIN (FI) statement or by using the .FI extension when the signal appears on the right side of an FFB equation.

Note: Registered functions assigned to FFBs may only use FastCLK nets and can only be asynchronously SET (except for the XC7336 which can also be asynchronously reset).

Note: Any function assigned to an FFB may only use an FOE signal for 3-state output control.

Note: All FFB outputs (except for the XC7336) must be active-low at the device pin. If they are not active-low, the software will transform the equation to an active-low output if possible.

In the PAL conversion example, to assign signal L to an FFB and signal E to a fast input pin, make the following changes to the Top-Level File:

1. Add the statement:

   ```
   PARTITION FFB L
   ```

2. Add the statement:

   ```
   INPUTPIN (FI) E
   ```

Because L is assigned to an FFB during the compilation process (FITEQN), the software will automatically create the following statements to change the 3-state output enable control signal from product term control to FOE control:

```
FOEPIN D
OUTPUTPIN (FOE=D) L
```

The software will also delete the product term 3-state control statement from the PAL2 equation file:

```
L.TRST = (D);
```

Note: The software does not modify the original PLD files. It creates a modified version of the file that is compiled by FITEQN. The modified design file is reported in the *design_name*.EQN file.

Using PLD Files in Schematics

This chapter provides a step-by-step procedure for including behavioral equations into schematic designs. This is an easy process because XEPLD automatically processes your schematic, assembles any PLD equation files, and integrates the final design automatically. Though you can create complete Xilinx EPLD designs using predefined schematic library components such as gates, flip-flops, and counters, you can also create custom components expressed as behavioral designs or PLD files, which give you full control of the device architecture.

This chapter includes a design example of using PALs in a schematic. See Chapter 5 "Arithmetic Design Rules" for additional information on creating custom arithmetic components.

Choosing Library Components

To include a behavioral equation file into your schematic design, you must use the special PLD library components or create a custom component. This section discusses the features of each type of PLD component in the XEPLD library and provides guidelines for their use.

Table 4-1 shows the various ways in which you can specify that a PLD file is targeted to a schematic symbol.

Table 4-1 Specifying Library Components

Device Type	CHIP Statement	Library Component
PAL - 22V10	22V10	PL22V10
PAL - 20V8, GAL - 20V8	20V8	PL20V8
Other PALs	Unrecognized*	PL20PIN, PL24PIN, PL48PIN
Custom	COMPONENT	Custom Symbol
High Density Function Block	PLFB9	PLFB9
Fast Function Block	PLFFB9	PLFFB9

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

Note: If the PLD type in the CHIP statement is not recognized it will default to GENERIC.

Using the PL22V10 or PL20V8

XEPLD supports the 22V10 and 20V8 PAL devices through special PAL library components. Generally you would 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 the PL20PIN, PL24PIN, and PL48PIN

PALs other than the 22V10 and 20V8 are supported by the generic 20 pin, 24 pin, and 48 pin PAL library components. Generic PAL equation files must explicitly define all logic functions in an architecturally-independent form; any implied functions such as 3-state control, signal inversion, clocks, and so on (which are specific to the originally targeted PAL device) cannot be recognized by XEPLD. However, XEPLD does provide automatic partitioning and equation splitting for these devices. Refer to the PLUSASM Language reference in the

XEPLD Reference Guide for details on how each equation in your .PLD or .PDS file is interpreted.

Use the word "GENERIC" (or any PAL device type other than 22V10 or 20V8) in the PAL file CHIP statement to target your PAL file to one of the generic PAL components. Each pin provided in these component symbols can be used for any input, output, or I/O function in your PAL equation file. You can also create your own custom symbol to represent your PAL equations, as described in the following section "Creating Custom Components."

Note: XEPLD does not have a JEDEC conversion utility for generic PALs. A PALASM boolean equation file with either a .PLD or .PDS file extension is required.

Using the PLFB9 and PLFFB9

These library components are based on the logic contained in XC7000 High Density Function Blocks (PLFB9) or Fast Function Blocks (PLFFB9). By targeting custom logic to one of these components your logic maps to a single Function Block; partitioning and equation splitting will not be performed by the compiler. The library symbol has enough I/O and control pins to support all possible applications of the selected Function Block. The PLFB9 symbol includes the arithmetic carry-in and carry-out pins.

Generally you would choose these components if you are familiar with PLUSASM and wish to use the special features of the EPLD device architecture. These components allow the experienced user to create the most compact designs by controlling the placement of equations in the device.

Note: Use the keyword "PLFB9" or "PLFFB9" In the equation file CHIP statement to target your PAL file to the PLFB9 or PLFFB9 components.

Creating Custom Component Symbols

You can define custom components with their own unique symbols and pinouts, and place them into your schematic. Custom symbols can have pinouts which are easier to recognize when viewing your schematic. They are especially useful when custom functions are used repetitively throughout your design.

Use the keyword "COMPONENT" in the equation file CHIP statement to target your PAL file to a custom symbol. This informs the software that the pins on the symbol have the same names as the signals in your equation file. The logic for these components can be created using any of the behavioral design methods discussed in this manual. As with any generic PLD equation file, all logic functions must be explicitly defined by equations.

Unlike a hierarchical symbol representing an underlying schematic, custom symbols are processed as primitives by the XEPLD software. You may need to apply an attribute to the symbol to indicate that it is a primitive and is not to be expanded into an underlying schematic.

For information on creating the schematic symbols for these components see the Xilinx CAE tool Interface Manual for your schematic entry tool.

Choosing a PLD Development Method

See Chapter 1 and Chapter 2 of this manual for a complete description of various PLD development methods. Each of the methods as they apply to schematics is described as follows:

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 facility as described in Step 7 of the following section.

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.

Using PLUSASM

Use PLUSASM to develop your PLD equation files if you want to access specific architectural features such as the high speed carry, local feedback, and product term export 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, PLFB9, PLFFB9, GENERIC, or COMPONENT in the PLUSASM equation file CHIP statement.

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 don't need to access device specific features such as the high speed carry, local feedback, and product term export paths. 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, CUPL, 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 or the CUPL -c compiler option), you can easily generate PLUSASM-compatible equation files.

These equation files can be targeted to the PL22V10, PL20V8, generic PAL, or custom schematic symbols.

Design Flow

A variety of design flows for creating schematic designs containing PLD files is shown in Figure 4-1.

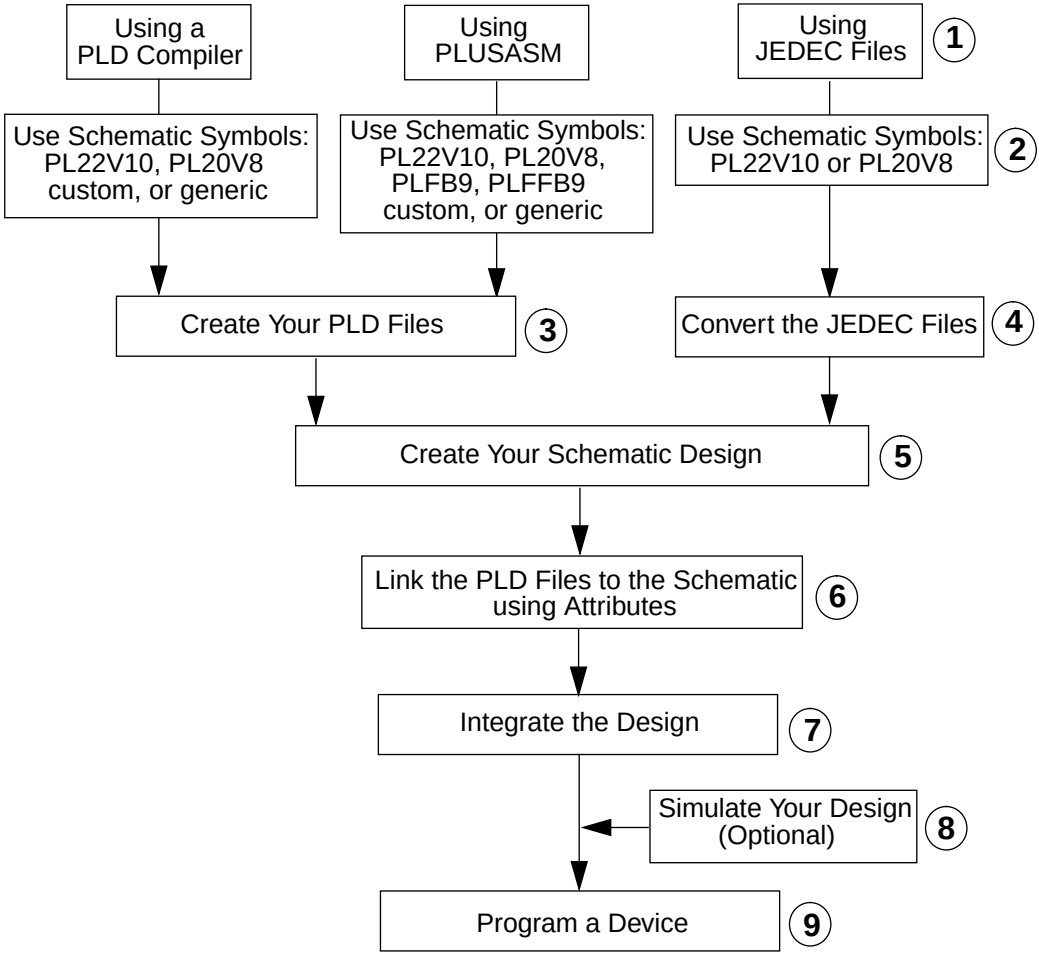


Figure 4-1 Design Flow for Using PLDs in Schematics

Step 1 — Choose a PLD Design Method

PLD design methods are discussed in Chapter 1 and Chapter 2 of this manual.

Step 2 — Choose PLD Components

See "Choosing Library Components" earlier in this chapter for a description of the various PLD components available to you.

Step 3 — Create Your PLD Files

Use any of the methods described in this book to create your .PLD files or import existing PAL files. Many popular PAL compilers such as ABEL, CUPL, LOG/iC, and PALASM can generate the PALASM2 boolean equation files required by the XEPLD software.

Step 4 — Convert Your JEDEC Files

If you are using JEDEC files (for either the 22V10 or 20V8) then execute the TRANSLATE→JED2PLD Command on each file. Figure 4-2 illustrates the JED2PLD menu.

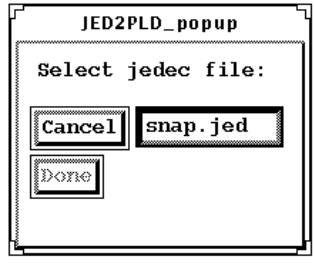


Figure 4-2 JED2PLD Popup Window

Step 5 — Create Your Schematic Design Files

Use your schematic capture tool to enter the schematic using XEPLD library symbols. Use the PLD component symbols provided in the library or create your own custom components.

Step 6 — Link Your Files to the Schematic

Link the PLD symbols in your schematic to the equation files that define their function by placing an attribute on each PLD symbol.

The format is:

PLD=*file_name*

The file name does not include the file extension. For the specific syntax of this attribute, see the Xilinx CAE tool Interface Manual for your specific CAE tool.

Step 7 — Integrate the Design

Execute TRANSLATE→XEMAKE from the XDM main menu.

Figure 4-3 illustrates the XEMAKE menu.

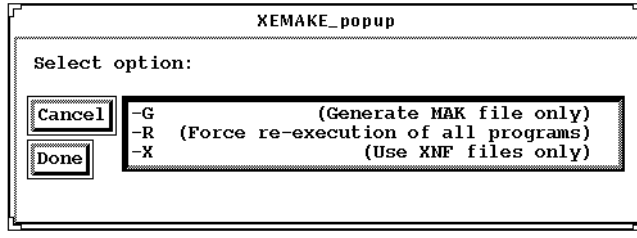


Figure 4-3 XEMAKE Popup Menu

Select "Done" and XEMAKE performs the following functions:

- Prompts you for the name of a schematic file.
- Integrates the schematic and automatically creates a netlist file in the XNF format.
- Runs PLUSASM for each PLD file found in the schematic.
- Runs FITNET to integrate the complete design.
- Generates Resource, Pinlist, Mapping, Error, Partitioning, Equation, and Logic Optimization reports.

Step 8 — Simulate Your Design (optional)

See the Xilinx CAE tool interface manual for simulation instructions relating to your particular simulator.

Step 9 — Program the Device

Generate a programming bit-map file using the XDM VERIFY→MAKEPRG command (for Intel Hex format) or the VERIFY→MAKEJED command (for JEDEC format). Refer to your device programmer documentation for instructions on how to download the bit-map file.

Design Example — Using PALs in a Schematic

The circuit shown in Figure 4-4 demonstrates how to handle many common situations when using PALs in schematics. This example deals with the strategies, equation syntax, and procedures for incorporating PALs into a schematic and is not intended to be a complete tutorial. See the Xilinx CAE Tool Interface Manual for your particular schematic entry tool for more information including net labeling and attribute assignment conventions.

Choosing Library Components

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 in Figure 4-5. 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 3-state outputs or bi-directional signals. The modified equation files are shown in Figure 4-6.The equation file for PAL1 requires editing because it contains a bi-directional signal that goes off-chip and because it has 3-state outputs, otherwise it could have been used unmodified.

Assigning Clock Signals to FastCLK Nets

Assigning clocks to the global FastCLK nets will give you faster clock-to output delays and reduced macrocell resource requirements. The XEPLD optimization software assigns clocks to the FastCLK nets

whenever possible without user intervention. However, you can also specify which clocks are assigned to the FastCLK nets by using the global clock library components. To specify signal A as a FastCLK, use a BUFG (Global Input Buffer) instead of an IBUF (Input Buffer) in the schematic. (If an IBUF is used, the clock is implemented as a product term clock.)

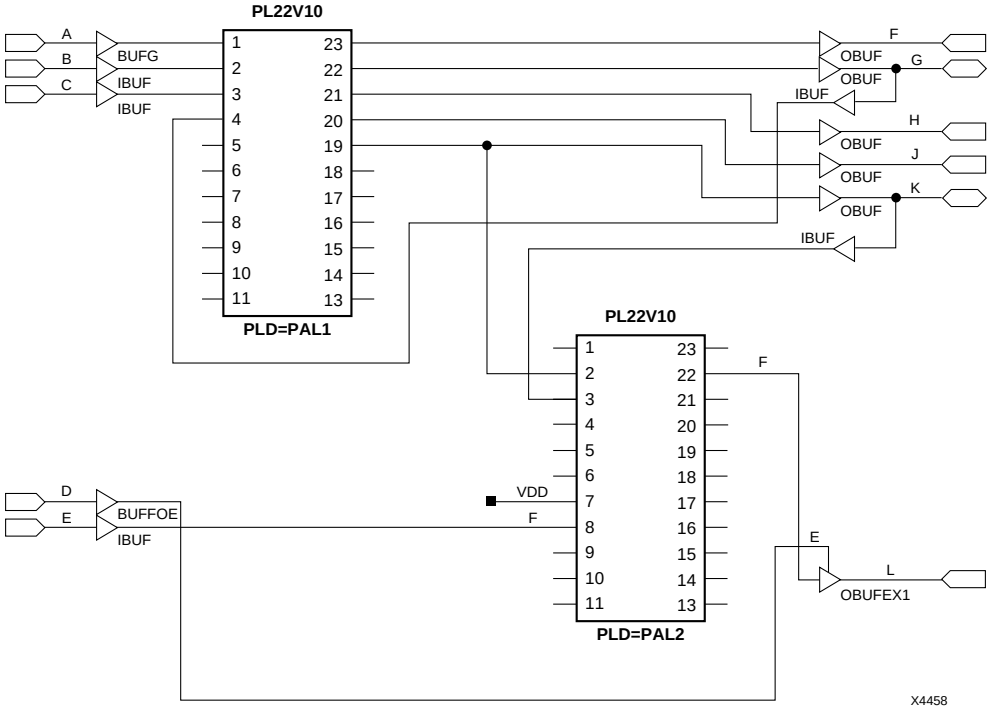


Figure 4-4 Example Schematic Using PALs

```
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 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);
```

Figure 4-5 Original PAL1 Equation File

```
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);
```

Figure 4-6 Modified PAL1 Equation File

Assigning Output Enable Signals to FOE Nets

Assigning signals to the global fast output enable (FOE) nets gives you reduced output enable delays and reduced macrocell resource requirements. The XEPLD optimization software assigns output enable signals to the global FOE nets whenever possible without user intervention. However, you can also specify in your schematic which output enable signals are to be assigned to the 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

Critical functions can be assigned to Fast Function Blocks to take advantage of their higher speed and increased output drive capabilities. This is accomplished by either assigning attributes to the schematic or by making changes to the equation file.

Using Schematic Attributes

You can specify which component outputs are assigned to Fast Function Blocks by assigning an “F” attribute to the nets that are driven by the component outputs.

You can assign fast inputs to Fast Function Blocks by assigning the “F” attribute to the net that drives the component input.

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.

Note: You can assign component pins to FFBs, even if they do not drive another component in the schematic, by attaching them to dangling (unconnected) nets and assigning the “F” attribute to the net.

Using the PLFFB9 Component

You can assign signal outputs to Fast Function Blocks by targeting the PLFFB9 library component in the schematic.

You can assign fast inputs by using the FASTINPUT statement in your equation file.

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

1. Change the target device in the PAL2 equation file CHIP statement to PLFFB9:

   ```
   CHIP PAL2 PLFFB9
   ```

2. Add the following statement to the equation file:

   ```
   FASTINPUT E
   ```

3. Change any 3-state output enable control signals to FOE nets by permanently enabling the PAL outputs. Do this by removing any .TRST control equations from the PAL file. In the equation file delete (or comment out) the following statement:

   ```
   L.TRST = D
   ```

4. Connect signal L to an OBUFEX1 output buffer in the schematic.
5. Connect signal D to a BUFFOE input buffer in the schematic.
6. Connect the OBUFEX1 output enable input to the BUFFOE input buffer in the schematic.

Note: Registered functions assigned to an FFB may only use the global FastCLK nets and be asynchronously SET.

Note: Any function assigned to an FFB may only use the global FOE signal for 3-state output signal control. The .TRST equations are not permitted for PLFFB9 components.

Note: The default clock pin for any registered signal in a PLFFB9 is pin 34. You can move the clock signal to pin 34 or you can override the default by adding .CLKF control equations.

Assigning Bi-Directional I/O Signals

Two of the most common usages of bi-directional signals are described below. In Case 1, Signal G is a bi-directional 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 — Bi-Directional Outputs that Go Off-Chip

To create a bidirectional signal in a schematic for XC7000 devices you must specify both an output signal and an input signal. This requires two separate pins on the PLD symbol (one input and one output) to schematically represent the connections even though the physical implementation in the XC7000 device requires only one I/O pin. In a low-density PAL, the signal at the pin is used for both internal feedback and signal output when the output is bidirectional and therefore requires only one pin in a schematic. Therefore, In a schematic design for XC7000 devices, you must create an extra input to the PLD for every bi-directional I/O signal.

Note: If you do not have enough unused PLD pins to create the extra inputs required to implement bi-directional signals in an XC7000 device, you can compile your equations as a PLD component and place them into a PL48 or custom component. Thus, you are not limited by the number of device pins in the original PAL.

Because signal G in the example was defined as a bi-directional signal in the original PAL, it must use pin feedback in the XC7000 device. To create a bi-directional 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.

Note: Signal G on the right side of an equation implies macrocell feedback. Using G_PIN instead of G explicitly specifies pin feedback because G_PIN is driven by the input buffer of an XC7000 I/O pad.

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.

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 5

Advanced Behavioral Design Techniques

The following subjects are discussed in this chapter:

- Manual Device Pin Assignment.
- Simulating Behavioral Designs.
- Verifying Behavioral Designs.
- Design Fitting Strategies.
- Design Rules for Arithmetic Design.

Manual Device 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 pin declaration and PARTITION statements.

Note: Pin declaration statements override the pin assignments in the pin-save file. This allows you to make changes to your fixed pin specifications. However, any conflict of pin specifications between the PARTITION statements and the pin-save file will cause an error.

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 product term requirements of your logic assigned to pins that are driven by the Fast Function Blocks. Plan ahead for design iterations which may create functions that require the exported product terms from an adjacent macrocell.

Be sure that the pin assignments in your pin declaration statements are not in conflict with your PARTITION statements. These conflicts will result in failures because the fitter is being instructed to either map multiple functions into the same macrocell or to map the same function into multiple macrocells.

Using Pin Declaration Statements

Assign specific device pins to signals by appending the PIN keyword to any pin declaration statement. For example:

```
INPUTPIN (RCLK=C) X Y Z PIN 9 10 11
FOEPIN F PIN 32
FASTCLOCK CLKIN PIN 27
```

All physically possible pin assignments are accepted regardless of their effect on device partitioning or mapping efficiency.

Pin assignments within pin declaration statements override pin assignments from the pin-save file.

Using PARTITION Statements

You can place PARTITION statements in the header of a behavioral design file to control the assignment of equations to macrocells, which determines the pin assignment of those outputs. All linked equations (equations using either .ADD, .SHIFT, .EXPORT, or .ADDMODE) must appear in a PARTITION statement; the order in which their names appear determines the order in which the macrocells are linked. The software will automatically connect the carry chain across function block boundaries if necessary.

Logical PARTITION Statements

Use a logical PARTITION statement to group signals into any function block; the specific function block and the ordering of the signals within that function block are determined by the software.

For example: **`PARTITION TEST_FB X0 Y0 Z0`**

Physical PARTITION Statements

Use physical PARTITION statements to group signals into a specific function block and optionally into specific consecutive macrocells.

For example: **`PARTITION FB3 A1 A2 A3 A4`**
or: **`PARTITION FB3_7 A1 A2 A3 A4`**

See the *XEPLD Reference Guide* for more information on the PARTITION statement.

Simulating Behavioral Designs

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

To simulate a behavioral design, you must first translate it into a netlist consisting of XC7000 library models. XEPLD automatically creates behavioral simulation files in the XNF netlist format which can be exported to the Viewlogic Viewsim simulator (.WIR), the OrCAD simulator (.VST), or the Mentor simulator 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.

Using Viewlogic Viewsim or OrCAD VST

To create a Viewlogic or OrCAD simulation model of your design:

1. Select the FITTER→FITEQN command from the XDM menu. This compiles your behavioral 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_simulation" for OrCAD.
   - Select "Viewlogic_epld_simulation" 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* for more information on OrCAD 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 FITTER→FITEQN 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.

Simulating Board-Level Designs

If you are simulating a circuit that contains one or more EPLD devices you must create a schematic symbol and an XNF file for each EPLD device. The name of each symbol is the name of the EPLD design, as specified in the CHIP statement of the Top-Level File. The pin names of the symbol must match the I/O signal names in the design.

Verifying Behavioral Designs

After you have compiled your design, using the FITTER→FITEQN command, XEPLD generates the 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 equations were 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.

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 in Figure 5-1. This design comes

from Timing Example 1 in the next section and was targeted for the XC73108-12PC84.

The Logic Resources section of the Resource Report shows that 6 macrocells were used in the design and 102 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, 4 output signals were placed on output pins, and 2 I/O signals were placed on I/O pins. Also used were 2 FOE pins and 1 FastCLK pin. A total of 45 pins (10 input and 35 I/O) remain available for additional input signals.

```
XEPLD, Version 5.0                                          Xilinx Inc.
                                Resource Report
 Circuit name: EXAMPL1A
Target Device: XC73108-12PC84                 Integrated: 11- 6-93,
10:24AM

LOGIC RESOURCES

                 Required  Used      Remaining
Function Blocks     3         3         9
Macrocells          6         6       102


PIN RESOURCES:

Type    Req   --------Used--------------  --------Remaining---------
              I  O  I/O Fclk Foe Cen Tot  I  O  I/O Fclk Foe Cen Tot
Inputs    2   2        0                 2 10       35                45
Outputs   4       4    0    0   0   0    4    12    35    2   0   2  51
I/Os      2            2                 2          35                35
Fclks     1                 1            1                2            2
Foes      2                     2        2                    0        0
Cens      0                         0    0                        2    2
        --- --- --  ---  --- --- --- ---
          6   2  4    2    1   2   0   8


Note:The design requires 1 pins with Fast Input capability.
     This device has 12 pins with Fast Input capability.
     The design requires 4 pins with Fast Output capability.
     This device has 12 FO and 0 I/FO remaining from original 16 FO and
0 I/FO.

 End of Resource Report
```

Figure 5-1 Timing Example 1 — Resource Report

Verifying Design Timing

If you are not using a timing simulator you can use the Equation Report to calculate how fast your design will run. This report shows your design equations as they appear after optimization enabling you to trace your logic flow through the device architecture. The declarations section tells you which of your signals were assigned to the FastCLK nets, the FOE nets, and the UIM nodes. The PARTITION statements tell you which signals were assigned to Fast Function Blocks (partitions FB1 and FB2 in XC7000-series devices).

Because the XC7000 device architecture is similar to a PAL, the device timing can be determined by using a relatively small set of fixed timing parameters listed in the EPLD device data sheet.

By viewing the optimized equations in the *design_name*.EQN report, you can easily determine the logic path through the device and apply the timing parameters found in the device data sheet.

Three design examples are provided in this section to demonstrate how to calculate design timing:

- **Example 1** shows you how to calculate timing for designs using global control signals, High Density Function Blocks, and Fast Function Blocks.
- **Example 2** shows you how to calculate timing for designs that require equation splitting of internal nodes. The timing calculations for UIM functions is also demonstrated.
- **Example 3** shows you how to calculate timing for designs that require splitting of external signals. (The splitting of nodes and output signals is performed as needed by the XEPLD software to fit large equations into available function block resources.)

Timing Calculation Example 1

This example is taken from the design in Chapter 3. The Equation Report for this design is shown in Figure 5-2. View this report to see the exact logic paths assigned by the optimization software.

The PARTITION statements show that the optimization software placed output signals F, H, J, and L into FB1, a Fast Function Block. I/O signals G and K were placed into High Density Function Blocks. The pin declaration statements show that both output enable signals C and D were placed on the global FOE nets.

The following timing calculations are based on the August 1993 datasheet for the XC73108-12PC84.

Setup and Hold — Signals B and G to Clock

Signals B and G are used as input signals for equations F and H which have been placed in a Fast Function Block. These input and I/O signals have not been declared as fast inputs, therefore they pass through the UIM before entering the function block.

The setup and hold times at the pin can be determined from the Fast Function Block external setup and hold times, and the internal UIM delay, as follows:

Setup Time = tSUF + tUIM = 6 + 10 = 16 ns

Hold Time = tHF - tUIM = 0 - 10 = -10 ns

Clock-to-Output — Signals F, G, H, J, K, L from Clock

Signals F, H, and J have been placed in a Fast Function Block. Their clock-to-output delay is therefore the Fast Function Block external clock-to-output delay (tCOF):

Clock-To-Output delay = tCOF = 9 ns

Signals G and K have been placed in a High Density Function Block. The clock for these signals (A) is implemented using a FastCLK. Their clock-to-output delay is therefore the High Density Function Block external FastCLK-to-output delay (tCO):

FastCLK-To-Output delay = tCO = 12 ns

Signal L has been placed in a Fast Function Block and is dependent on the pin feedback of a registered function (K) placed in a High Density Function Block. The clock-to-output delay path is calculated from the sum of the High Density Function Block external FastCLK-to-output delay (tCO) and the Fast Function Block external pin-to-pin propagation delay from an I/O pin (tPDFU):

Clock-To Output = tCO + tPDFU = 12 + 22 = 34ns

Output Enable/Disable — Signals G, K, L

These signals have been declared in the pin declaration statements as controlled by FOE pins. The output enable/disable time therefore is equal to the I/O pad output enable/disable delay (tFOEI):

Output Enable/Disable time = tFOEI = 12 ns

Pin-to-Pin Propagation Delay — Signal E to L

Signal L has been placed in a Fast Function Block and signal E is declared as a fast input. The pin-to-pin propagation delay is therefore equal to the Fast Function Block external propagation delay tPDFO):

Pin-to-Pin Propagation Delay = tPDFO = 12 ns

Maximum Frequency — Clock A

The longest signal path in the design is from the output of a High Density Function Block register (G), through an I/O buffer (pin feedback for signal G), through the UIM, and into a Fast Function Block register (H):

Cycle Time = tCOI + tOUT + tIN + tUIM + tFLOGI + tFSUI =
1 + 8 + 4 + 10 + 2 + 3 = 28 ns

```
; This is the .eqn file produced by the partitioner.  It shows      ;
; how your equations were implemented in order to best utilize the  ;
; resources available on the chip.                                  ;
;                                                                   ;
; This design was compiled for the 7310884

PATTERN exampl1a.eqn
DATE  Mon Oct 25 19:03:21 1993
CHIP EXAMPL1A XEPLD
MINIMIZE OFF
PARTITION FB1_2 F H J L
PARTITION FB10_9 G
PARTITION FB6_1 K
INPUTPIN B
INPUTPIN ( FI ) E
OUTPUTPIN  F H J
OUTPUTPIN ( FOE = D ) L
IOPIN  ( FOE = C ) G K
FASTCLOCK A
FOEPIN C D
EQUATIONS
/L  = /K.PIN
      + /J
      + /E
/F := /B
   F.CLKF =  A
/G := /F
   G.CLKF =  A
/H := /G.PIN
/J := /K
      + /B
   J.CLKF =  A
/K := /B
   K.CLKF =  A
```

Figure 5-2 Example 1 Equation Report

Timing Calculation Example 2

This example demonstrates how to calculate timing when automatic equation splitting of nodes is performed by the software. This occurs when the number of product terms required by a function cannot be placed into a single macrocell. When nodes are split the software tries to re-combine the outputs in the UIM. If this is successful, then no timing penalty is incurred by the splitting and re-combining process. However, if the software cannot re-combine the macrocell outputs in the UIM, then the outputs are re-combined in a macrocell which adds delay to the function.

This example also demonstrates how to calculate timing when the software creates UIM AND gates in order to reduce the number of function block inputs when mapping the design.

The design consists of a 24-bit up-counter, a 24-bit data storage register, and a 24-bit comparator as shown in Figure 5-5. The counter counts up until its value is equal to the contents of the data register. On the next clock, the counter is reset to 0, and a timing pulse is generated one clock later. The effects of equation splitting are illustrated in Figure 5-4. The Top-Level File for this design is shown in Figure 5-5. The Equation Report is shown in Figure 5-6.

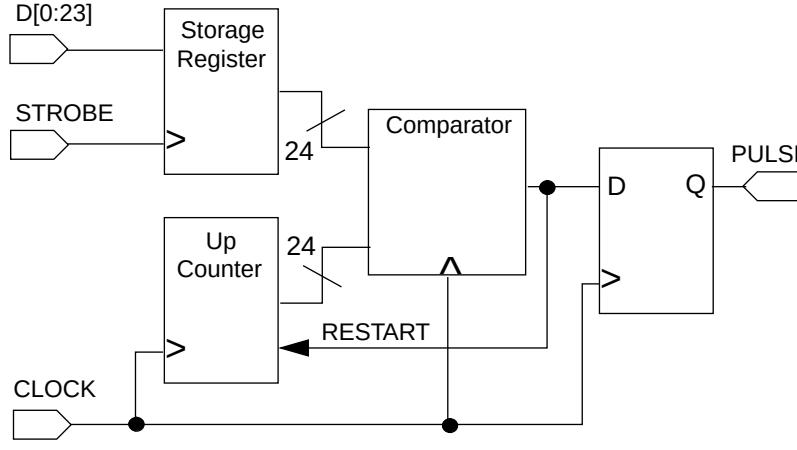


Figure 5-3 24-Bit Counter Block Diagram

The 24-bit data storage register is placed into the device I/O registers, therefore no macrocell resources are used to create it. In the Equation Report, the INPUTPIN (RCLK=CLOCK) statement specifies that the data bits are mapped into the I/O registers (not into macrocells).

The comparator requires 48 product terms and therefore equation splitting and UIM optimization of the resulting OR function are performed by the software. The equation for RESTART, the comparator output, was split into six nodes called RESTART_5 through RESTART_0. The NODE (UIM) RESTART statement shows that the outputs of these nodes were successfully recombined in the UIM and therefore design timing remains unaffected by the equation splitting.

The pin declaration statements in the Equation Report specify that STROBE and CLOCK use the global FastCLK nets.

Automatic UIM optimization was also performed on some of the counter feedback signals by ANDing them together to reduce the number of function block inputs required. These UIM functions, UIM_0 through UIM_6, are declared as UIM nodes and therefore they do not affect the design timing.

The following timing calculations are based on the August 1993 data sheet for the XC7236A-16PC44.

Setup and Hold — D[0:23] to STROBE

The data registers have been placed into the I/O pads and are clocked with STROBE which is placed on a FastCLK net. Therefore, the setup and hold time at the pins is equal to the input register setup and hold time, relative to FastCLK:

Setup Time = tSU2 = 6 ns

Hold Time = tH2 = 0 ns

Clock-to-Output — PULSE

The PULSE signal is placed into a macrocell that is clocked by CLOCK which has been assigned to a FastCLK net. Therefore, the clock-to-output delay is equal to the device FastCLK input-to-registered output delay:

Clock-to-Output = tCO = 10ns

Maximum Frequency — Clock A

The equations in the Top-Level File specify that the longest path in this design should be from the output of a macrocell register used in the counter (Q0-Q23), through the UIM, and into another macrocell register used for the RESTART comparator output. The same path is used from the output of the RESTART register, through the UIM, to the PULSE register. This timing path is unaffected by functions declared as UIM nodes in the Equation Report.

The maximum operating frequency of this design is equal to the maximum sequential toggle frequency of the High Density Function Block:

Maximum Frequency = fCYC = 60MHz

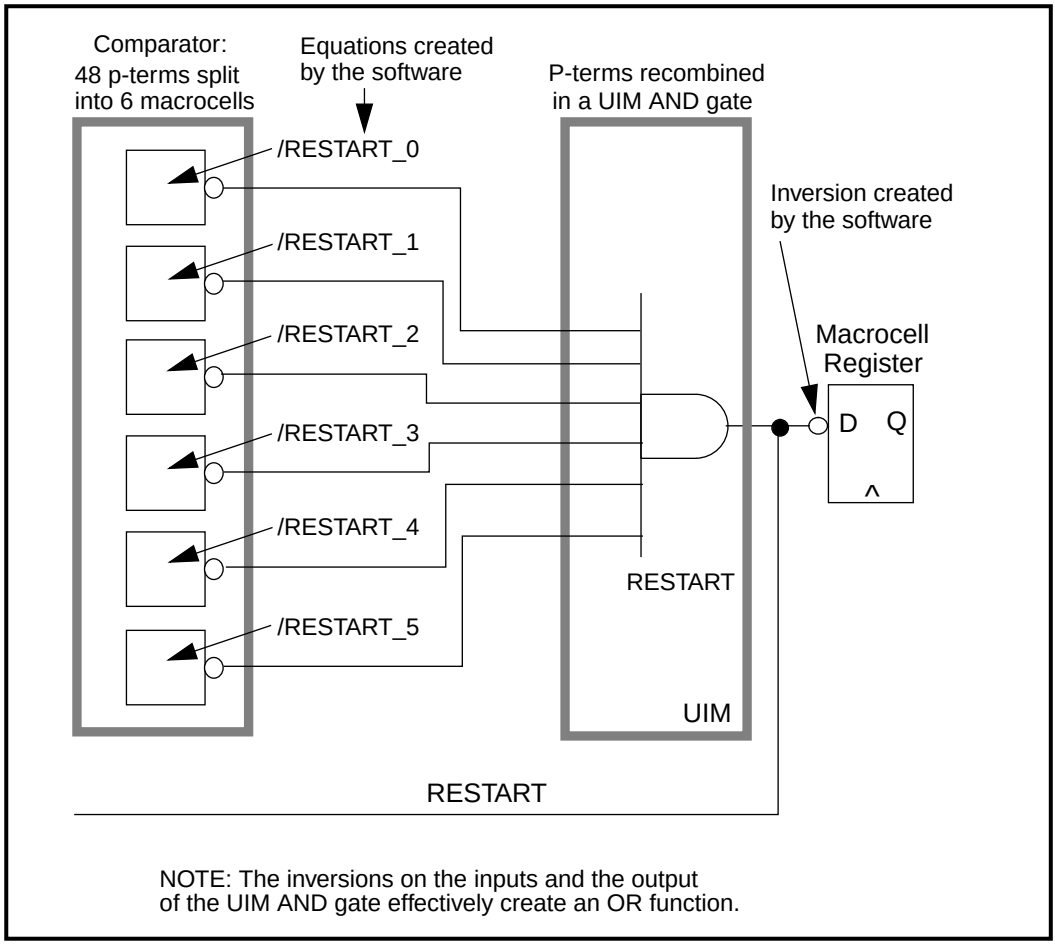


Figure 5-4 The Effects of Equation Splitting

```
TITLE CTR24
CHIP   CTR24   XEPLD;
INPUTPIN
      clock strobe d0 d1 d2 d3 d4 d5 d6 d7 d8 d9 d10 d11 d12 d13 d14
      d15 d16 d17 d18 d19 d20 d21 d22 d23
OUTPUTPIN
      pulse
NODE
      q0 q1 q2 q3 q4 q5 q6 q7 q8 q9 q10 q11 q12 q13 q14 q15 q16 q17
      q18 q19 q20 q21 q22 q23 data0 data1 data2 data3 data4 data5
      data6 data7 data8 data9 data10 data11 data12 data13 data14
      data15 data16 data17 data18 data19 data20 data21 data22 data23
      restart
;24bitctr
minimize off restart
EQUATIONS
data23 :=  d23            ;24-bit data storage register
data23.CLKF =  strobe     ;data stored on rising edge of srrobe
... NOTE: data22 through data1 removed for brevity
data0 :=  d0
data0.CLKF =  strobe
/restart :=  /q0 * data0 + q0 * /data0   ;restart counter from 0 when
counter
         +  /q1 * data1 + q1 * /data1    ;value is equal to data value
... NOTE: + /q2 through + /q22 removed for brevity
          +  /q23 * data23 + q23 * /data23
restart.CLKF =  clock
pulse :=  restart   ;pulse goes high for one clock period
pulse.CLKF =  clock
q23 := (q23 * /restart) :+: (q0 * q1 * q2 * q3 * q4 * q5 * q6 * q7 *
q8 * q9 * q10 * q11 * q12 * q13 * q14 * q15 * q16 * q17 * q18 * q19 *
q20 * q21 * q22 * /restart)
q23.CLKF =  clock
...NOTE: q22 through q1 removed for brevity
q0 := (q0 * /restart) :+: (/restart)
q0.CLKF = clock
```

Figure 5-5 Example 2 Top-Level File

```
; This is the .eqn file produced by the partitioner.  It shows      ;
; how your equations were implemented in order to best utilize the  ;
; resources available on the chip.                                  ;
; This design was compiled for the 723644                           ;
PATTERN ctr24.eqn
DATE  Wed Oct 27 11:55:30 1993
CHIP CTR24 XEPLD
MINIMIZE OFF
PARTITION FB2_1 Q4 Q5 Q6 Q7 Q8 Q12 Q13 Q19 RESTART_2
PARTITION FB4_1 Q2 RESTART_4 Q3 Q9 Q10 Q11 Q18 Q20 Q21
PARTITION FB1_1 PULSE Q0 Q1 Q14 Q15 Q22 Q23 RESTART_3 RESTART_5
PARTITION FB3_1 Q16 Q17 RESTART_0 RESTART_1
INPUTPIN ( RCLK = STROBE ) D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 D10 D11 D12
                        D13 D14 D15 D16 D17 D18 D19 D20 D21 D22 D23
OUTPUTPIN  PULSE
NODE Q0 Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8 Q9 Q10 Q11 Q12 Q13 Q14 Q15 Q16 Q17
                    Q18 Q19 Q20 Q21 Q22 Q23
NODE (UIM) RESTART
NODE RESTART_0 RESTART_1 RESTART_2 RESTART_3 RESTART_4 RESTART_5
NODE (UIM) UIM_0 UIM_1 UIM_2 UIM_3 UIM_4 UIM_5 UIM_6
FASTCLOCK CLOCK STROBE
EQUATIONS
 RESTART  = /RESTART_0 */RESTART_1 */RESTART_2 */RESTART_3 */
 RESTART_4 */RESTART_5
... NOTE: Some equations are removed for brevity
/PULSE := /RESTART
   PULSE.CLKF =  CLOCK
 Q23 :=  Q23.D1 XOR Q23.D2
   Q23.D1 =  Q23 */RESTART
   Q23.D2 =  Q0 * Q1 * Q6 * Q7 * Q8 * Q9 * Q10 * Q14 * Q15 * Q22
           */RESTART * UIM_4 * UIM_5
   Q23.CLKF =  CLOCK
... NOTE: Some equations are removed for brevity
 Q0 :=  Q0.D1 XOR Q0.D2
   Q0.D1 =  Q0 */RESTART
   Q0.D2 = /RESTART
   Q0.CLKF =  CLOCK
 UIM_0  =  Q0 * Q1 * Q2 * Q3
 UIM_1  =  Q15 * Q16 * Q17 * Q18
 UIM_2  =  Q6 * Q7 * Q8
 UIM_3  =  Q12 * Q13 * Q14 * Q15 * Q16 * Q17
 UIM_4  =  Q2 * Q3 * Q4 * Q5 * Q11 * Q12 * Q13
 UIM_5  =  Q16 * Q17 * Q18 * Q19 * Q20 * Q21
 UIM_6  =  Q0 * Q1 * Q2 * Q3 * Q4 * Q5 * Q6 * Q7 * Q8 * Q9 * Q10
           * Q11 * Q12 * Q13 * Q14
```

Figure 5-6 Example 2 Equation Report

Timing Calculation Example 3

This example demonstrates how to calculate timing when automatic equation splitting of external outputs is performed by the software. This occurs when the amount of logic required by a function cannot be placed into a single macrocell.

For this example a minor change was made to the design of the previous timing example 2. The PULSE output was deleted and the RESTART signal was brought off-chip instead. The UIM AND gate output (RESTART) must drive a macrocell (it cannot directly drive an I/O pad) and therefore the software must create an equation (which requires a macrocell) in which to receive the re-combined signal, as shown in Figure 5-8. In example 2 this was not necessary because the recombined signal was input to a macrocell register (PULSE), which could drive the output pad.

The block diagram of this example is shown in Figure 5-7 and the Equation Report is shown in Figure 5-9.

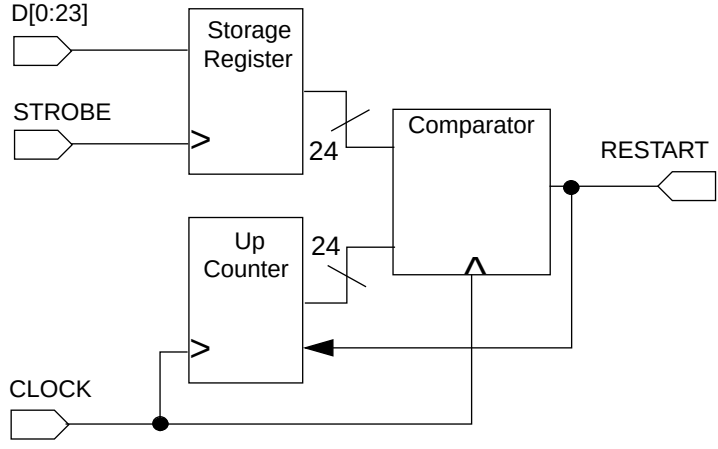


Figure 5-7 Timing Example 3 — Block Diagram

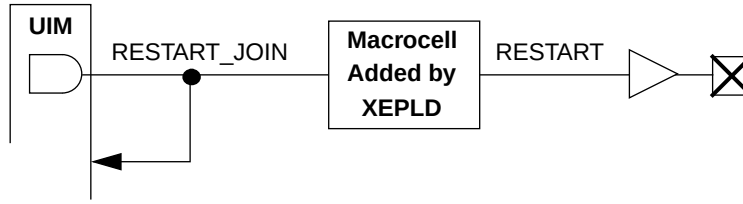


Figure 5-8 Macrocell Buffer Created by the Software

The UIM AND gates created by the splitting of external signals are declared as UIM nodes in the Equation Report with the name:

*split_equation_name*_JOIN.

The output of the *split_equation_name*_JOIN function comes from a UIM AND gate and is used for any internal feedback; therefore this signal does not create extra delay. However, to drive an output buffer, this signal must pass through a macrocell and thus incurs one extra macrocell delay for the external signal. Therefore, when you see a UIM node named *split_equation_name*_JOIN, the Equation Report will show that an equation for a buffer was created by the software. This is the macrocell required to take the signal off-chip, as shown in Figure 5-8. The syntax for this software-created buffer is:

split_equation_name = *split_equation_name*_JOIN

In the Equation Report for this example you will see:

RESTART = RESTART_JOIN

Clock-to-Output — RESTART

The equation was split internally into several registered nodes. The signal then makes an additional internal combinatorial pass through a macrocell before going to the output buffer. Therefore the clock-to-output delay can be calculated by adding one internal delay to the XC7236 clock-to-output delay, as follows:

Clock-To-Output = tCO + tPDF = 10 + 10 = 20 ns

Maximum Frequency

The maximum frequency of Example 3 is determined by the feedback path from the RESTART register back to the counter. According to the .EQN file listing, the Q0–Q23 counter equations use the UIM signal RESTART_JOIN which adds no additional delay to the feedback path. Therefore, the circuit can still be clocked at the maximum High Density Function Block frequency of fCYC:

Maximum Frequency = fCYC = 60Mhz

```
PATTERN ctr24.eqn
DATE  Thu Oct 28 11:25:32 1993
CHIP CTR24 XEPLD
MINIMIZE OFF
PARTITION FB2_1 Q4 Q5 Q6 Q7 Q8 Q12 Q13 Q20 RESTART_2
PARTITION FB4_1 Q2 RESTART_4 Q3 Q9 Q10 Q11 Q18 Q19 Q21
PARTITION FB1_1 RESTART Q0 Q1 Q14 Q15 Q22 Q23 RESTART_3 RESTART_5
PARTITION FB3_1 Q16 Q17 RESTART_0 RESTART_1
INPUTPIN ( RCLK = STROBE ) D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 D10 D11 D12
                        D13 D14 D15 D16 D17 D18 D19 D20 D21 D22 D23
OUTPUTPIN  RESTART
NODE (UIM) RESTART_JOIN
NODE Q0 Q1 Q2 Q3 Q4 Q5 Q6 Q7 Q8 Q9 Q10 Q11 Q12 Q13 Q14 Q15 Q16 Q17
                    Q18 Q19 Q20 Q21 Q22 Q23 RESTART_0 RESTART_1
                    RESTART_2 RESTART_3 RESTART_4 RESTART_5
NODE (UIM) UIM_0 UIM_1 UIM_2 UIM_3 UIM_4 UIM_5 UIM_6
FASTCLOCK CLOCK STROBE
EQUATIONS
 RESTART_JOIN  = /RESTART_0 */RESTART_1 */RESTART_2 */RESTART_3 */
                     RESTART_4 */RESTART_5
 RESTART_0 :=  D23 */Q23
       + /D23 * Q23
       +  D22 */Q22
       + /D22 * Q22
...
 RESTART_5 := /D1 * Q1
       +  D0 */Q0
       + /D0 * Q0
   RESTART_5.CLKF =  CLOCK

 RESTART  =  RESTART_JOIN

 Q23 := /RESTART_JOIN * Q23
       + /RESTART_JOIN * Q0 * Q1 * Q6 * Q7 * Q8 * Q9 * Q10 * Q14
            * Q15 * Q22 * UIM_4 * UIM_5
   Q23.D1 = /RESTART_JOIN * Q23
   Q23.D2 = /RESTART_JOIN * Q0 * Q1 * Q6 * Q7 * Q8 * Q9 * Q10 * Q14
            * Q15 * Q22 * UIM_4 * UIM_5
   Q23.CLKF =  CLOCK
...
 UIM_5  =  Q16 * Q17 * Q18 * Q19 * Q20 * Q21

 UIM_6  =  Q0 * Q1 * Q2 * Q3 * Q4 * Q5 * Q6 * Q7 * Q8 * Q9 * Q10
           * Q11 * Q12 * Q13 * Q14
```

The software-created macrocell buffer for outputting the signal to an output pad

Figure 5-9 Example 3 Equation Report

Design Fitting Strategies

This section discusses the strategies for getting your design to fit within a target device.

If your design requires more macrocells or signal pins than are available in the target device then 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.

For example, the XEPLD software optimizes the performance and mapping efficiency of your design by collapsing combinatorial nodes forward into output pins and registers. This minimizes the number of macrocells required to implement your design but increases the signal fan-in and number of product terms required to implement the resulting optimized functions. In some cases this creates functions that when mapped into a macrocell, use a significant number of function block shared product terms or inputs, which restricts the access to the remaining macrocells in that function block. 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.

Optimizing Device Resources

If you determine that the target device has enough total macrocells and signal pins to contain your design and yet the design will not fit, the easiest solution is to choose a larger device with more resources and re-compile your design. However, if you have only a few remaining unmapped functions, you can possibly get your design to fit by controlling the optimization of device 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. This situation is illustrated in the example Partitioning Report shown in Figure 5-10. 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.

Part Name	# of Outputs	# of Input Lines Used	Signal Inputs	# of Shared PT	O/IO Req	0/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	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

Figure 5-10 Example of a P-Term Constrained Design

You can decrease the usage of shared product terms by controlling the equation splitting parameters contained in the XEPLD.CFG file. An example of this file is shown in Figure 5-11.

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 an equation can use before it will be split.

2. Reduce the variable far enough to cause the product term intensive equations to split into multiple macrocells.

   You might try for example:

   (alias max_shared_before_splitting 6)

This will increase your macrocell count but your logic will require fewer partitions.

In this example, by reducing the max_shared_before_splitting variable, you can cause the equations mapped into function blocks 3 and 4 to split, allowing the equation 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 equations are re-combined in the UIM if possible, which adds no delay.

```
(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
```

Figure 5-11 Example XEPLD.CFG File

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 in Figure 5-10.

Part Name	# of Outputs	# of Input Lines Used	Signal Inputs	# of Shared PT	O/IO Req	0/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

Figure 5-12 Example of an FB Input Constrained Design

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. In the Top-Level File, use the LOGIC_OPT statement to control logic collapsing.

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

```
LOGIC_OPT OFF
```

To turn off logic collapsing only for specific signals, use the following:

```
LOGIC_OPT OFF signal_1 signal_2 ... signal_n
```

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

If Your Design has Unused Fast Function Blocks

This section discusses how to modify your design to fit logic into Fast Function Blocks.

The XEPLD software automatically maps into the Fast Function Blocks any function that meets these requirements:

- All clocks use the global FastCLK signals.
- All 3-state controls use the global FOE signals.
- All registers may only be asynchronously set.
- All registers may only be preloaded to a logic high state or have unspecified preload values.
- All logic must use 4 or less p-terms when implemented as active low.

If your logic output signals must use an internal p-term clock, you can drive 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 logic output 3-state controls must be controlled by internal p-terms you can drive the p-term control signal off-chip through an FOE pin and back into the global FOE net through the I/O buffer on the FOE pin. If your registers require asynchronous reset inputs or if the preload state must be a logic low, then your design will need modification in order to fit into an FFB.

If a function meets all the requirements except that it uses more than four p-terms, you can manually place it into a Fast Function Block by using PARTITION statements. The software will then use the product term .EXPORT capability to implement the function across multiple macrocells.

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

In the following example, File1 is the original file and cannot be placed in a Fast Function Block because it specifies product term clock and 3-state control signals. File2 shows the required changes to make the design work in an FFB.

```
INPUTPIN a b c d e f g
OUTPUTPIN out
EQUATIONS
out := a * B * C * d * e
out.clkf = f * g; use p-term clock
out.trst = /h; use p-term 3-state control
```

Figure 5-13 File1 — Cannot be Placed in an FFB

```
INPUTPIN a b c d e f g
OUTPUTPIN (FOE=out_trst) out
FASTCLOCK out_clk; Global clock signal
FOEPIN out_trst; Global 3-state control signal
PARTITION FFB out; Place this output into a Fast Function Block
EQUATIONS
out := a * B * C * d * e
out_clk = f * g; use global FastCLK
out_trst = /h; use global 3-state control
```

Figure 5-14 File2 — Can be Placed in an FFB

Note: In Figure 5-14 the default clock is out_clk. Therefore you do *not* need to include the following statement:

```
out.clkf = out_clk
```

Design Rules for Arithmetic Design

This section discusses the arithmetic logic and fast carry path capabilities of the Xilinx XC7000-series devices. Arithmetic logic design requires that you not only create the proper equations but also that you place those equations into the correct physical order in the device.

PLUSASM arithmetic logic syntax differs between designs targeted for the XC7272 and the remainder of the XC7000 series devices, due to differences in the circuit implementations. Also, the arithmetic equation extensions are different; the XC7272 uses the .ADDMODE extension, while the other devices use the .ADD extension.

Arithmetic Logic Architecture (Except XC7272)

This section describes the arithmetic logic path within the XC7236 and XC7300 series architectures. Figure 5-15 shows the circuit block diagram for the arithmetic carry logic in each macrocell. Figure 5-16 shows the architecture of the XC7236 including the carry connections between Function Blocks.

To perform arithmetic functions, the function generator in the ALU is programmed to perform an exclusive OR operation on the D1 and D2 ALU inputs. When the carry input to the macrocell is disabled, as is the case when the macrocell represents the least significant bit of an adder, the ALU performs as a half adder and adds only the D1 and D2 inputs. When the carry input to the ALU is enabled for the remaining adder bits, the ALU performs as a full adder, adding the carry to the D1 and D2 inputs.

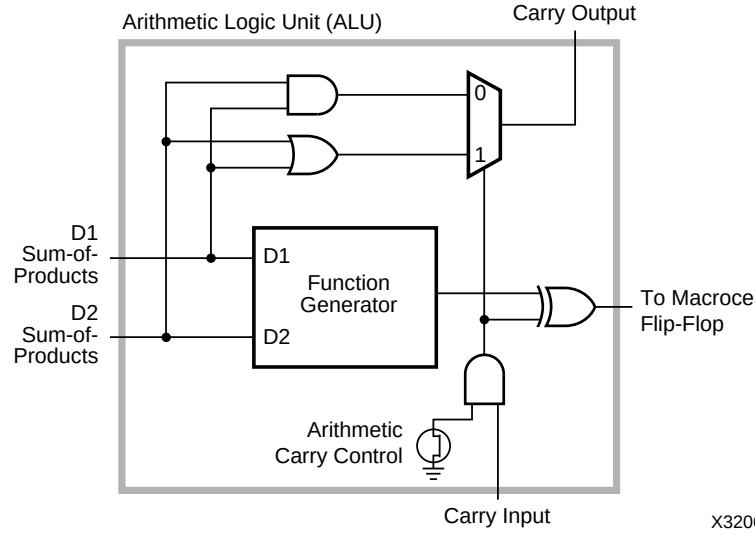


Figure 5-15 XC7000 Series ALU Logic Diagram (except XC7272)

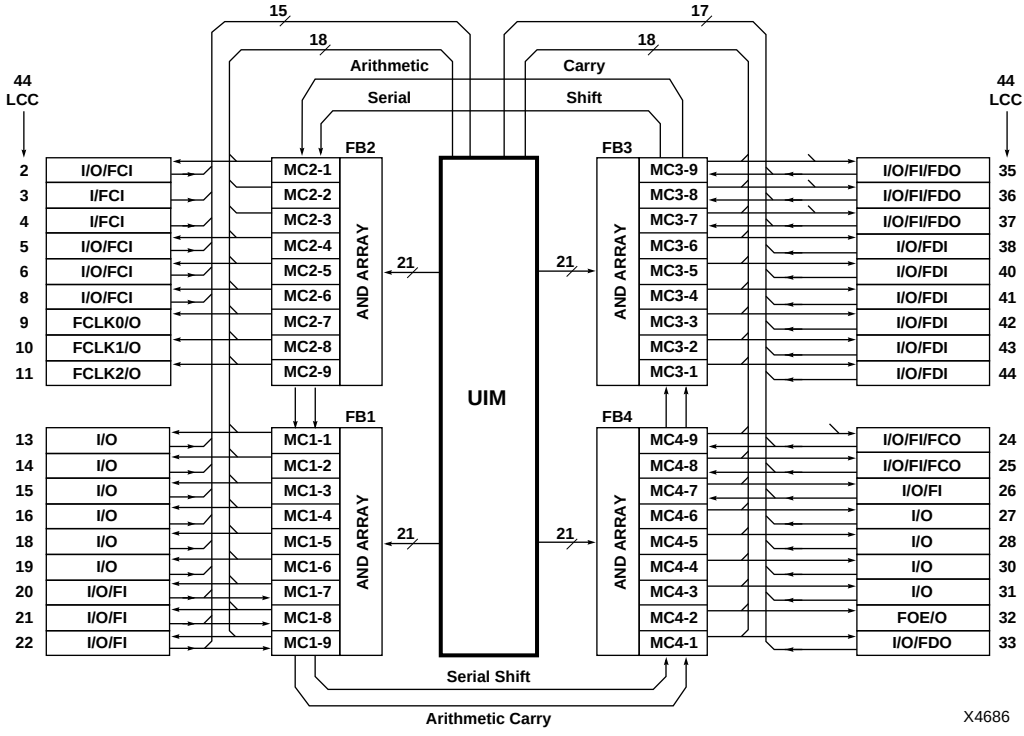


Figure 5-16 XC7236 Architecture

4-Bit Adder Example

The PLUSASM equation file syntax for a four bit adder is listed in Figure 5-17. The arithmetic carry control bit is controlled by the PLUSASM .ADD equation. When .ADD is set to VCC, the carry input is enabled. If the .ADD equation is not present, the carry input is disabled by default. The least-significant bit, S1, does not have a .ADD equation specified and therefore does not receive a carry-in signal. The equation syntax is the same for both PLD equation files and behavioral designs.

```
S1.D1  = A1
S1.D2  = B1
S1     = S1.D1 XOR S1.D2
S2.D1  = A2
S2.D2  = B2
S2     = S2.D1 XOR S2.D2
S2.ADD = VCC
S3.D1  = A3
S3.D2  = B3
S3     = S3.D1 XOR S3.D2
S3.ADD = VCC
S4.D1  = A4
S4.D2  = B4
S4     = S4.D1 XOR S4.D2
S4.ADD = VCC
```

Figure 5-17 Four-Bit Adder Equation File

To have dynamic control of the carry input into the least significant macrocell of an arithmetic function, you can write an equation for another macrocell to control the carry output from the D1 and D2 product terms as shown in Figure 5-18. Be sure to enable the carry into the least significant macrocell of the arithmetic function.

```
DUMMY.D1 = CARRY_IN
DUMMY.D2 = CARRY_IN
DUMMY    = DUMMY.D1 GND DUMMY.D2
S1.ADD   = VCC; Enable carry into S1
```

Figure 5-18 Dynamic Control of the LSB Input

To extract a signal from the carry chain and make it available to a macrocell logic output, use the equation shown in Figure 5-19.

```
CARRY_OUT     = CARRY_OUT.D1 GND CARRY_OUT.D2
CARRY_OUT.ADD = VCC
```

Figure 5-19 Extracting the Carry Signal

Partitioning Arithmetic Equations

Arithmetic macrocells must be physically adjacent to each other and in the correct order for the carry chain to propagate properly. The physical placement of equations is controlled by the PARTITION statement in the Top-Level File of behavioral designs. The macrocell order in the PARTITION statement must be listed from LSB to MSB as shown below; this places the macrocells in the correct order in the carry chain.

```
PARTITION ADD4 DUMMY S1 S2 S3 S4 CARRY_OUT
```

In this case, the software maps these equations in the specified order, entirely within any one function block in the device. If the adder does not fill an entire function block, the software is free to shift the adder to start in any macrocell position that allows the entire function to be mapped into a single function block.

The PARTITION statement can also be used to control the order of arithmetic functions that span multiple function block boundaries. For example, a 16-bit adder can be implemented as follows:

```
PARTITION ADD16 DUMMY S1 S2 S3 S4 S5 S6 S7 S8 S9
S10 S11 S12 S13 S14 S15 S16 CARRY_OUT
```

In this case, the software is free to map the adder into any contiguous group of macrocells that meet the I/O requirements of the logic.

If you need to minimize the number of carry signals between function blocks, use physical PARTITION statements to place the LSB equation into the least significant macrocell of a function block. For example, the following PARTITION statement specifies that the carry chain begins at function block 10, macrocell 3:

```
PARTITION FB10_3 DUMMY S1 S2 ... S16 CARRY_OUT
```

Note: Use the functional block diagram shown in the target device data sheet to determine the direction of the carry path.

For PLD equation files, in order to map the equations into adjacent macrocells and in the correct order according to the carry chain, the output variable names must appear contiguously in the pin list, from least-to-most-significant bit order. For example:

```
CHIP PLFB9 ADD9
A1 A2 A3 A4 A5 A6 A7 A8 A9 B0 B2 B3 B4 B5 B6 B7 B8
B9 S1 S2 S3 S4 S5 S6 S7 S8 S9
```

8-Bit Adder/Subtracter/Accumulator Example

A more complex design that takes advantage of the macrocells' fast feedback path is shown in the following design example of an 8-bit two's compliment adder/subtracter/accumulator. The logic diagram is shown below in Figure 5-20.

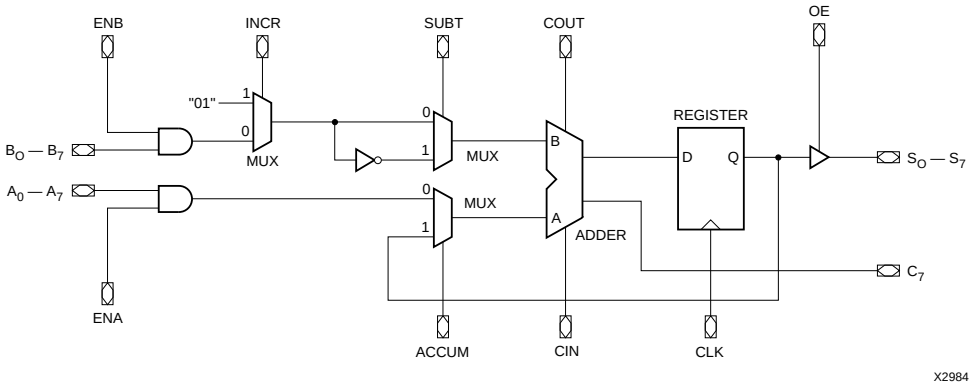


Figure 5-20 8-Bit Adder/Subtracter/Accumulator

The PLUSASM equation syntax for two bits of the adder/subtracter /accumulator is listed below in Figure 5-21.

```
S1.D1 = EB1*/SUBT*/INCR + /EB1*SUBT*/INCR + SUBT*INCR
S1.D2 = EA1*/ACCUM
S1.FBK = ACCUM
S1.ADD = VCC
S1 := S1.D1 XOR S1.D2
S1.CLKF = CLK
S1.TRST = OE
S2.D1 = EB2*/SUBT*/INCR + /EB2*SUBT*/INCR + SUBT*INCR
S2.D2 = EA2*/ACCUM
S2.FBK = ACCUM
S2.ADD = VCC
S2 := S2.D1 XOR S2.D2
S2.CLKF = CLK
S2.TRST = OE
```

Figure 5-21 8-Bit Adder/Subtracter/Accumulator Equations

The accumulator can be cascaded. The least significant bit of the accumulator is preceded by the following equation, which generates the appropriate carry-in, as shown below in Figure 5-22.

```
DUMMY.D1 = SUBT*/INCR + /SUBT*INCR
DUMMY.D2 = VCC
DUMMY    = DUMMY.D1 GND DUMMY.D2
```

Figure 5-22 Generating the Carry-In for the LSB

In this example, the AND-gates enabling the A and B input buses are implemented in the UIM. In a behavioral design, each gated A and B operand would be declared as a UIM node using the statement:

```
NODE (UIM) EA1 EA2 EA3 ... EB1 EB2 EB3 ...
```

The gated operands for the first three bits of the accumulator would then be implemented as follows:

```
EA1=A1*ENA
EA2=A2*ENA
EA3=A3*ENA
...
EB1=B1*ENB
EB2=B2*ENB
EB3=B3*ENB
...
```

XC7272 Arithmetic Logic Architecture

This section describes the arithmetic logic path within the XC7272 architecture, which is different from the other devices in the family. Figure 5-23 shows the circuit block diagram for the arithmetic carry logic in each macrocell. Figure 5-24 shows the XC7272 chip architecture including the carry connections between function blocks.

To perform arithmetic functions, the arithmetic control bit is set to steer the carry bit into one input of the ALU function generator. The function generator is then programmed to perform an exclusive NOR operation on the D1 input and the inverse of the carry-in. When you write an equation for the D1 product term that implements a half-adder in sum-of-products format, the macrocell logic output becomes the sum output of a full-adder.

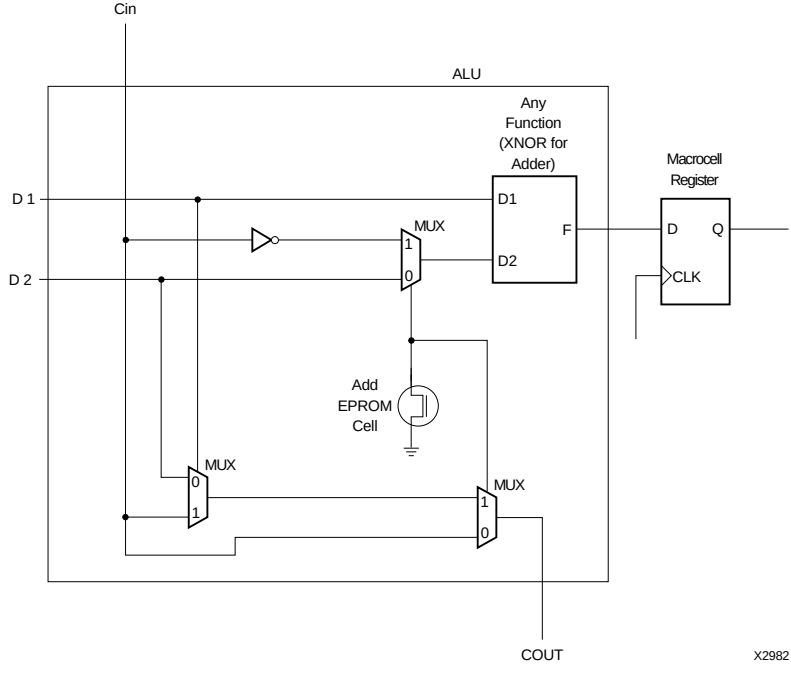


Figure 5-23 XC7272 ALU Logic Diagram

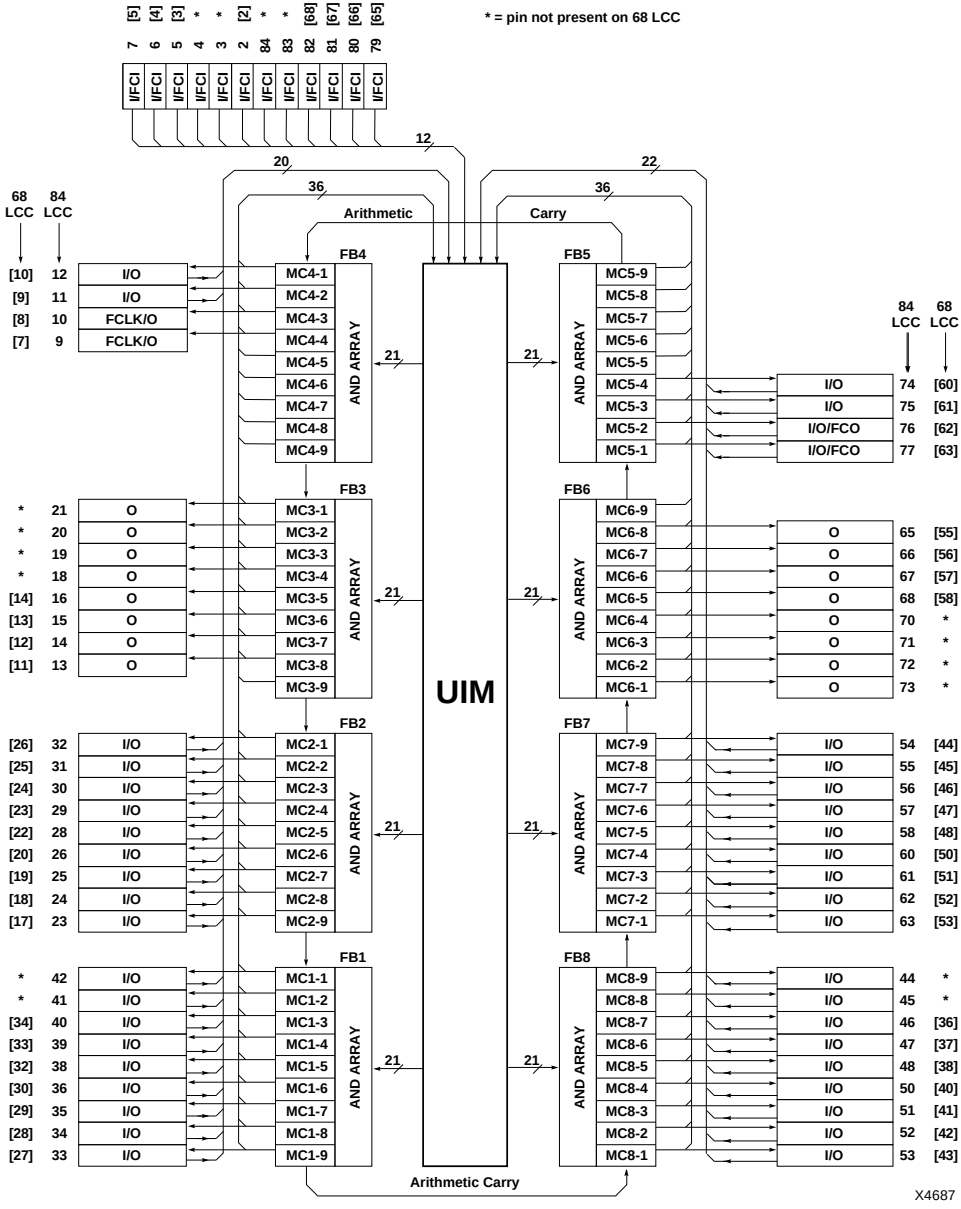


Figure 5-24 XC7272 Architecture Showing Carry Connections

When in arithmetic mode, the carry output of the macrocell comes from the output of a 2-to-1 multiplexer controlled by the D1 product term. When an equation for the D2 product term is written that sets the product term equal to one of the operands, the carry output of a full-adder is generated.

4-Bit Adder Example (XC7272)

The PLUSASM equation syntax for a 4-bit adder is listed below in Figure 5-25. The arithmetic control bit is controlled by the .ADDMODE equation, which must be turned on for the least significant bit of adders implemented in the XC7272. When .ADDMODE is set to VCC, the macrocell generates a carry-out. If the .ADDMODE equation is not present, the carry chain input is transparently passed through the macrocell, and the macrocell will not generate a carry-out.

```
S1.D1 = A1 * /B1 + /A1 * B1
S1.D2 = A1
S1 = S1.D1 XNOR S1.D2
S1.ADDMODE = VCC
S2.D1 = A2 * /B2 + /A2 * B2
S2.D2 = A2
S2 = S2.D1 XNOR S2.D2
S2.ADDMODE = VCC
S3.D1 = A3 * /B3 + /A3 * B3
S3.D2 = A3
S3 = S3.D1 XNOR S3.D2
S3.ADDMODE = VCC
S4.D1 = A4 * /B4 + /A4 * B4
S4.D2 = A4
S4    = S4.D1 XNOR S4.D2
S4.ADDMODE = VCC
```

Figure 5-25 4-Bit Adder (XC7272)

In arithmetic mode, the XC7272 always receives a carry-in. Therefore, you must always use another macrocell to control the carry input to the LSB of an adder. To generate a constant zero carry-in, use the equation shown below in Figure 5-26:

```
DUMMY.D1 = GND; macrocell carry-out = D2 P-Term
DUMMY.D2 = GND; D2 is always = 0
DUMMY    = DUMMY.D1 GND DUMMY.D2
DUMMY.ADDMODE = VCC
```

Figure 5-26 Generating a Constant Zero Carry-In

To create dynamic control of the carry input, use the equation shown below in Figure 5-27:

```
DUMMY.D1 = GND; macrocell carry-out = D2 P-Term
DUMMY.D2 = CARRY_IN
DUMMY = DUMMY.D1 GND DUMMY.D2
DUMMY.ADDMODE = VCC
```

Figure 5-27 Dynamic Carry-In Control

To extract a signal from the carry chain and make it available to a macrocell logic input, use the equation shown below in Figure 5-28:

```
CARRY_OUT.D1 = VCC; transp. carry pass through
CARRY_OUT    = CARRY_OUT.D1 NOTD2 CARRY_OUT.D2
             ; Macrocell output = inverse of D2
CARRY_OUT.ADDMODE = VCC; MUX passes carry input
                       ; to D2
```

Figure 5-28 Using the Carry Signal for Macrocell Input

Adder/Subtracter/Accumulator Example (XC7272)

The next example implements binary addition/subtraction with an accumulator. It is frequently desirable to store one number in a register of flip-flops (called an accumulator) and either add or subtract a second number from it, leaving the result stored in the accumulator. One way to build a parallel adder/subtracter with an accumulator is to add a flip-flop register to an adder/subtracter network, resulting in the circuit shown in Figure 5-29.

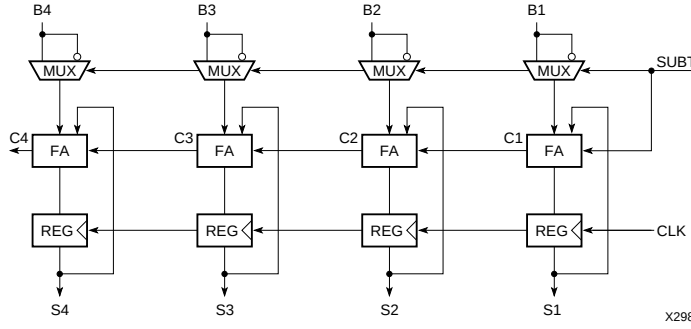


Figure 5-29 Two's-Complement Adder/Subtracter/Accumulator

Subtraction of binary numbers is easily accomplished by adding the complement of the number to be subtracted. For instance, to compute A - B, the two's complement of B is added to A. The two's complement of B can be formed by first finding the one's complement and then adding 1. The one's complement is formed by inverting each bit of B, and then addition of 1 is accomplished by setting the carry input to the full adder to 1.

The PLUSASM equation syntax for four bits of the add/subtract /accumulate function shown above are listed in Figure 5-30.

```
C0.D1 = SUBT*/SUBT           ; always generate carry when D1=0
C0.D2 = SUBT + /SUBT    ; carry_in = SUBT
C0 = C0.D1 GND C0.D2    ; (C0 not used)
C0.ADDMODE = VCC
S1.D1 = S1*/B1*/SUBT + /S1*B1*/SUBT + S1*B1*SUBT  + /S1*/
B1*SUBT
S1.D2 = S1
S1: = S1.D1 XNOR S1.D2
S1.CLKF = CLK
S1.ADDMODE = VCC
S2.D1 = S2*/B2*/SUBT + /S2*B2*/SUBT + S2*B2*SUBT + /S2*/B2*SUBT
S2.D2 = S2
S2: = S2.D1 XNOR S2.D2
S2.CLKF = CLK
S2.ADDMODE = VCC
S3.D1 = S3*/B3*/SUBT + /S3*B3*/SUBT + S3*B3*SUBT + /S3*/B3*SUBT
S3.D2 = S3
S3: = S3.D1 XNOR S3.D2
S3.CLKF = CLK
S3.ADDMODE = VCC
S4.D1 = S4*/B4*/SUBT + /S4*B4* SUBT + S4*B4*SUBT + /S4*/B4*SUBT
S4.D2 = S4
S4: = S4.D1 XNOR S4.D2
S4.CLKF = CLK
S4.ADDMODE = VCC
```

Figure 5-30 Adder/Subtracter/Accumulator Equations (4-bits)

Index

R

Trademark Information

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

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

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

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

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