Programmable Logic Design Quick Start Handbook

by Karen Parnell and Nick Mehta

August 2003

© 2003, Xilinx, Inc.
"Xilinx" is a registered trademark of Xilinx, Inc. Any rights not expressly granted herein are reserved.
The Programmable Logic Company is a service mark of Xilinx, Inc.

All terms mentioned in this book are known to be trademarks or service marks and are the property of their respective owners. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

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

PN 0402205 Rev. 3, 10/03

ABSTRACT

Whether you design with discrete logic, base all of your designs on microcontrollers, or simply want to learn how to use the latest and most advanced programmable logic software, you will find this book an interesting insight into a different way to design.

Programmable logic devices were invented in the late 1970s and have since proved to be very popular, now one of the largest growing sectors in the semiconductor industry. Why are programmable logic devices so widely used? Besides offering designers ultimate flexibility, programmable logic devices also provide a time-to-market advantage and design integration. Plus, they're easy to design with and can be reprogrammed time and time again – even in the field – to upgrade system functionality.

This book was written to complement the popular Xilinx Campus Seminar series, but you can also use it as a stand-alone tutorial and information source for the first of many programmable logic designs. After you have finished your first design, this book will prove useful as a reference guide or quick start handbook.

The book details the history of programmable logic devices; where and how to use them; how to install the free, fully functioning design software (Xilinx WebPACK™ ISE software is included with this book); and then guides you through your first designs. There are also sections on VHDL and schematic capture design entry, as well as a data bank of useful applications examples.

We hope you find this book practical, informative, and above all easy to use.

Karen Parnell and Nick Mehta

Navigating This Book

This book was written for both the professional engineer who has never designed using programmable logic devices and for the new engineer embarking on an exciting career in electronics design.

To accommodate these two audiences, we offer the following navigation section, to help you decide in advance which sections would be most useful.

CHAPTER 1: INTRODUCTION

Chapter 1 is an overview of how and where PLDs are used. It gives a brief history of programmable logic devices and goes on to describe the different ways of designing with PLDs.

CHAPTER 2: XILINX SOLUTIONS

Chapter 2 describes the products and services offered by Xilinx to ensure that your PLD designs enable a time-to-market advantage, design flexibility, and system future-proofing. The Xilinx portfolio includes CPLD and FPGA devices, design software, design services and support, and IP cores.

CHAPTER 3: WEBPACK ISE DESIGN SOFTWARE

Xilinx WebPACK ISE design software offers a complete design suite based on the Xilinx Foundation™ ISE series software. Chapter 3 describes how to install the software and what each module does.

CHAPTER 4: WEBPACK ISE DESIGN ENTRY

Chapter 4 is a step-by-step approach to your first design. The following pages are intended to demonstrate the basic PLD design entry implementation process.

CHAPTER 5: IMPLEMENTING CPLDS

Chapter 5 discusses the synthesis and implementation process for CPLDs. The design targets a CoolRunner™-II CPLD.

CHAPTER 6: IMPLEMENTING FPGAS

Chapter 6 takes the VHDL or schematic design through to a working physical device. The design is the same design as described in previous chapters, but instead targets a Spartan™-3 FPGA.

CHAPTER 7: DESIGN REFERENCE BANK

Chapter 7, the final chapter, contains a useful list of design examples and applications that will give you a jump start into your future programmable logic designs. This section also offers pointers on where to look for and download code and search for IP cores from the Xilinx website.

Table of Contents

Navigating This Book

Table of Contents

Chapter 1: Introduction

The History of Programmable Logic 1
Complex Programmable Logic Devices (CPLDs) 4
    Why Use a CPLD? 4
Field Programmable Gate Arrays (FPGAs) 6
    Design Integration 8
The Basic Design Process 9
HDL File Change Example 13
    Before (16 x 16 multiplier): 13
    After (32 x 32 multiplier): 13
Intellectual Property (IP) Cores 14
Design Verification 14
        Functional Simulation 16
        Device Implementation 16
        Fitting 16
        Place and Route 17
        Downloading or Programming 18
        System Debug 19

Chapter 2: Xilinx Solutions

Introduction 21
Xilinx Devices 22
Platform FPGAs 22
    Virtex FPGAs 22
    Virtex-II Pro FPGAs 23
        The Power of Xtreme Processing 23
        XtremeDSP – 23
        The Ultimate Connectivity Platform 24
        The Power of Integration 24
        Enabling a New Development Paradigm 24
        Industry-Leading Tools 24

Virtex FPGAs 24
Spartan FPGAs 25
Spartan-3 FPGAs 25
Shift register SRL16 blocks 27
As much as 520 Kb distributed SelectRAM™ memory 27
As much as 1.87 Mb Embedded block RAM 27
Memory Interfaces 27
Multipliers 28
XCITE Digitally Controlled Impedance Technology – 28
Spartan-3 XCITE DCI Technology Highlights 28
Full- and half-impedance input buffers 29
Spartan-3 Features and Benefits 29
Spartan-IIE FPGAs 31
Spartan-IIE Architectural Features 32
Logic Cells 35
Block RAM 37
Delay-Locked Loop 38
Configuration 39
Xilinx CPLDs 41
Product Features: 41
Selection Considerations: 41
XC9500 ISP CPLD Overview 42
XC9500 5V Family 42
Flexible Pin-Locking Architecture 42
Full IEEE 1149.1 JTAG Development and Debugging Support 42
XC9500 Product Overview Table 43
XC9500XL 3.3V Family 43
Family Highlights 44
Performance 44
Powerful Architecture 44
Highest Reliability 44
Advanced Technology 44
Outperforms All Other 3.3V CPLDs 45
XC9500XV 2.5V CPLD Family 45
High Performance Through Advanced Technology 45
The System Designer's CPLD 45
CoolRunner Low-Power CPLDs 47
XPLA3 Architecture 48
Logic Block Architecture 49
FoldBack NANDs 50
Macrocell Architecture 51
I/O Cell 52
Simple Timing Model 52
Slew Rate Control 53
XPLA3 Software Tools 53

CoolRunner-II CPLDs 55
CoolRunner-II Architecture Description 56
CoolRunner-II Function Block 57
CoolRunner-II Macrocell 59
Advanced Interconnect Matrix (AIM) 60
I/O Blocks 61
Output Banking 61
DataGATE 62
Additional Clock Options: 63
Design Security 65
CoolRunner-II Application Examples 66
CoolRunner Reference Designs 68
Accessing the Reference Designs 68
Military and Aerospace 71
Automotive and Industrial 71
Xilinx IQ Solutions – Architecting Automotive Intelligence 71
Design-In Flexibility 72
Design Tools 73
Design Entry 73
Synthesis 74
Implementation and Configuration 74
Board-Level Integration 74
Verification Technologies 75
Static Verification 75
Dynamic Verification 76
Debug Verification 76
Board-Level Verification 76
Advanced Design Techniques 76
Embedded SW Design Tools Center 77
Embedded Software Tools for Virtex-II Pro FPGAs 77
Xilinx IP Cores 78
Web-Based Information Guide 78
End Markets 79
Silicon Products and Solutions 80
Design Resources 80
System Resources 81
DSP Central 81
Algorithms/Cores 81
Xilinx Online (IRL) 81
Configuration Solutions 82
Processor Central 82
The Embedded Development Kit (EDK) 82
PowerPC Embedded Processor Solution 82
The UltraController Solution 82

MicroBlaze and PicoBlaze Soft Processor Solutions 83
Third-Party Processors Solution 84
CoreConnect Technology 84
Tools and Partnerships 84
Memory Corner 84
Silicon 85
Design Tools and Boards 85
Technical Literature and Training 85
Connectivity Central 86
Networking and Datapath Products 86
Control Plane and Backplane Products 86
High-Speed Design Resources 86
Signal Integrity Tools 86
Partnerships 86
Signal Integrity 86
Signal Integrity Fundamentals 87
Simulation Tools 87
Multi-Gigabit Signaling 87
Services 87
Xilinx Design Services 87
IP Core Modification 87
FPGA Design From Specification 87
FPGA System Design 87
Embedded Software Design 88
Education Services 88
Live E-Learning Environment 88
Day Segment Courses 89
Computer-Based Training (CBT) 89
University Program 89
Xilinx University Resource Center 89
Xilinx Answers Database 89
Xilinx Student Edition Frequently Asked Questions 90
Design Consultants 90
Technical Support 90

Chapter 3: WebPACK ISE Design Software

Module Descriptions 91
WebPACK Design Suite 93
WebPACK Design Entry 93
WebPACK StateCAD 93
WebPACK MXE Simulator 94
WebPACK HDL Bencher Tool 94
WebPACK FPGA Implementation Tools 94
WebPACK CPLD Implementation Tools 94

WebPACK iMPACT Programmer 94
WebPACK ChipViewer 95
XPower 95
WebPACK CD-ROM Installation 95
Getting Started 96
Licenses 96
Projects 97
Summary 97

Chapter 4: WebPACK ISE Design Entry

Introduction 99
Design Entry 100
The Language Template 104
Close the Language Templates 104
Edit the Counter Module 105
Save the Counter Module 107
Functional Simulation 107
State Machine Editor 112
Top-Level VHDL Designs 120
Top-Level Schematic Designs 125
ECS Hints 125
I/O Markers 128

Chapter 5: Implementing CPLDs

Introduction 131
Synthesis 132
Constraints Editor 133
CPLD Reports 142
Timing Simulation 144
Configuration 145

Chapter 6: Implementing FPGAs

Introduction 147
Synthesis 150
The Constraints File 153
FPGA Reports 158
Programming 159
Summary 159

3. Add and label the input and output buffers. These will define the I/O package pins for the device.
4. Generate a netlist.

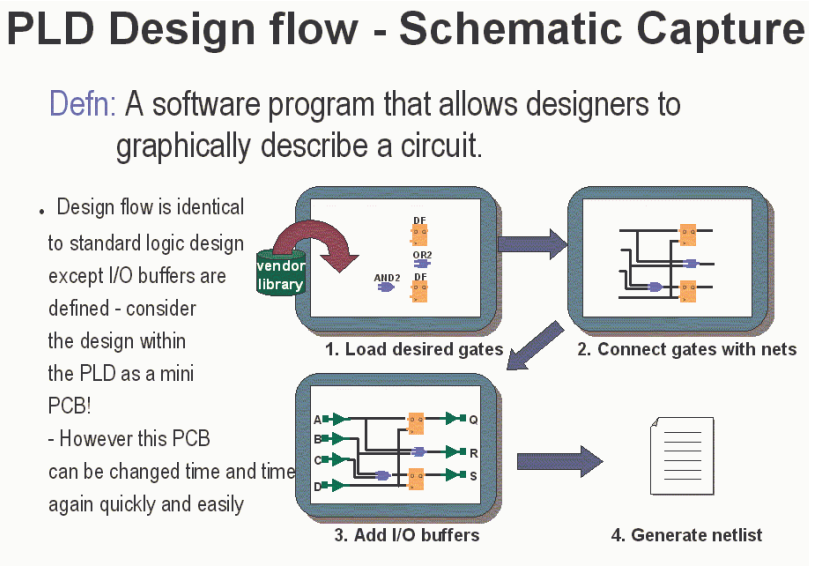


FIGURE 1-7: **PLD DESIGN FLOW**

A netlist is a text equivalent of the circuit. It is generated by design tools such as a schematic capture program. The netlist is a compact way for other programs to understand what gates are in the circuit, how they are connected, and the names of the I/O pins.

In the example below, the netlist reflects the actual syntax of the circuit in the schematic. There is one line for each of the components and one line for each of the nets. Note that the computer assigns names to components (G1 to G4) and to the nets (N1 to N8). When implementing this design, it will have input package pins A, B, C, and D, and output pins Q, R, and S.

EDIF is the industry-wide standard for netlists; many others exist, including vendor-specific ones such as the Xilinx Netlist Format (XNF).

Once you have the design netlist, you have all you need to determine what the circuit does.

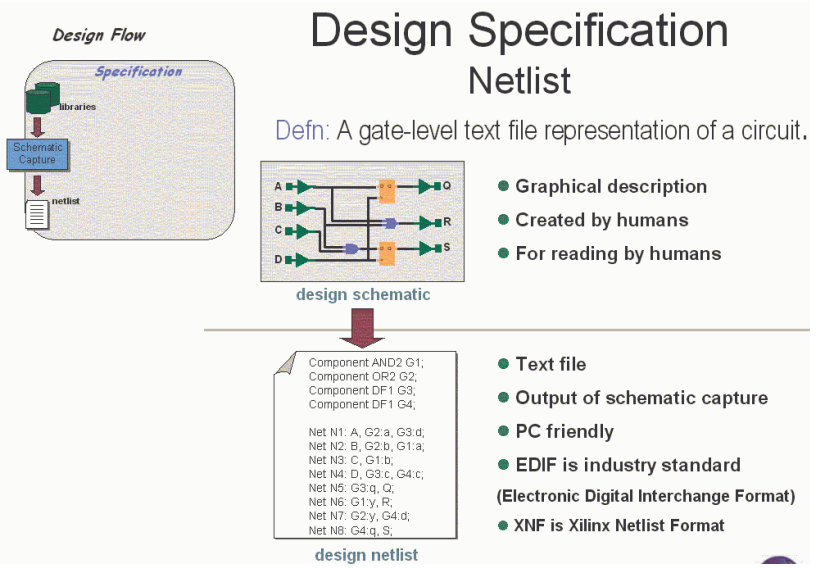


FIGURE 1-8: DESIGN SPECIFICATION – NETLIST

The example on the previous pages is obviously very simplistic. Let's describe a more realistic design of 10,000 equivalent gates.

The typical schematic page contains about 200 gates, contained with soft macros. Therefore, it would require 50 schematic pages to create a 10,000-gate design! Each page needs to go through all the steps mentioned previously: adding components, interconnecting the gates, adding I/Os, and generating a netlist. This is rather time-consuming, especially if you want to have a 20,000, 50,000, or even larger design.

Another inherent problem with using schematic capture is the difficulty in migrating between vendors and technologies. If you initially create your 10,000-gate design with FPGA vendor X and then want to migrate to a gate array, you would have to modify every one of those 50 pages using the gate array vendor's component library.

There has to be a better way ... and of course, there is. It's called high-level design (HLD), behavioral, or hardware description language (HDL). For our purposes, these three terms are essentially the same thing.

The idea is to use a high-level language to describe the circuit in a text file rather than a graphical low-level gate description. The term *behavioral* is used because in this powerful language, you describe the function or behavior of the circuit in words rather than figuring out the appropriate gates needed to create the application.

There are two major flavors of HDL: VHDL and Verilog.

As an example, let's design a 16 x 16 multiplier specified with a schematic capture and an HDL file.

A multiplier is a regular but complex arrangement of adders and registers that requires quite a few gates. Our example has two 16-bit inputs (A and B) and a 32-bit product output (Y = A x B) – that's a total of 64 I/Os. This circuit requires approximately 6,000 equivalent gates.

In the schematic implementation, the required gates would have to be loaded, positioned on the page, and interconnected, with I/O buffers added. That's about three days' worth of work.

The HDL implementation, which is also 6,000 gates, requires eight lines of text and can be done in three minutes. This file contains all the information necessary to define our 16 x 16 multiplier.

So, as a designer, which method would you choose? In addition to the tremendous time savings, the HDL method is completely vendor-independent. This opens up tremendous design possibilities for engineers.

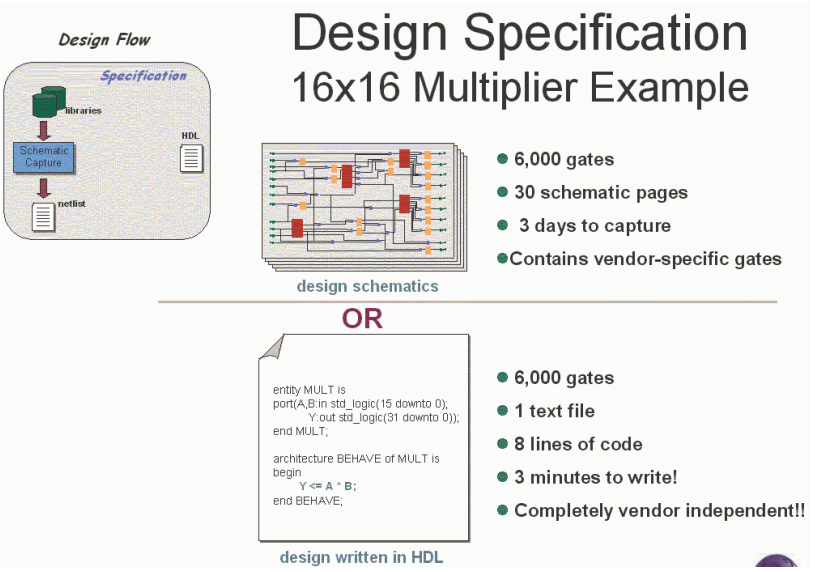


FIGURE 1-9: DESIGN SPECIFICATION – MULTIPLIER

To create a 32 x 32 multiplier, you could simply modify the work you'd already done for the smaller multiplier.

For the schematic approach, this would entail making three copies of the 30 pages, then figuring out where to edit the 90 pages so that they addressed the larger bus widths. This would probably require four hours of graphical editing.

For the HDL specification, it would be a matter of changing the bus references from 15 to 31 in line 2 and 31 to 63 in line 3. This would probably require about four seconds.

HDL File Change Example

BEFORE (16 X 16 MULTIPLIER):

```
entity MULT is
port(A,B:in std_logic(15 downto 0);
       Y:out std_logic(31 downto 0));
end MULT;

architecture BEHAVE of MULT is
begin
       Y <= A * B;
end BEHAVE;
```

AFTER (32 X 32 MULTIPLIER):

```
entity MULT is
port(A,B:in std_logic(31 downto 0);
       Y:out std_logic(63 downto 0));
end MULT;

architecture BEHAVE of MULT is
begin
       Y <= A * B;
end BEHAVE;
```

HDL is also ideal for design re-use. You can share your "library" of parts with other designers at your company, therefore saving and avoiding duplication of effort.

So, now that we have specified the design in a behavioral description, how do we convert this into gates, which is what all logic devices are made of?

The answer is *synthesis.* The synthesis tool does the intensive work of figuring out what gates to use based on the high-level description file you provide. (Using schematic capture, you would have to do this manually.)

Because the resulting netlist is vendor and device family-specific, you must use the appropriate vendor library. Most synthesis tools support a large range of gate array, FPGA, and CPLD device vendors.

In addition, you can specify optimization criteria that the synthesis tool will take into account when selecting the gate-level selection, also called *mapping.*

Some of these options include: optimizing the complete design for the least number of gates, optimizing a certain section of the design for fastest speed,

using the best gate configuration to minimize power, or using the FPGA-friendly, register-rich configuration for state machines.

You can easily experiment with different vendors, device families, and optimization constraints, thus exploring many different solutions instead of just one with the schematic approach.

To recap, the advantages of high level design and synthesis are many. It is much simpler and faster to specify your design using HLD, and much easier to make changes to the design because of the self-documenting nature of the language.

You are relieved from the tedium of selecting and interconnecting at the gate level. Merely select the library and optimization criteria (e.g., speed, area) and the synthesis tool will determine the results.

You can also try different design alternatives and select the best one for the application. In fact, there is no real practical alternative for designs exceeding 10,000 gates.

Intellectual Property (IP) Cores

IP cores are very complex pre-tested system-level functions that are used in logic designs to dramatically shorten development time.

The benefits of using an IP core include:

- Faster time to market
- A simplified development process
- Minimal design risk
- Reduced software compile time
- Reduced verification time
- Predictable performance/functionality.

IP cores are similar to vendor-provided soft macros in that they simplify the design specification step by removing designers from gate-level details of commonly used functions.

IP cores differ from soft macros in that they are generally much larger system-level functions, such as a PCI bus interface, DSP filter, or PCMCIA interface. They are extensively tested (and hence rarely free of charge) to prevent designers from having to verify the IP core functions themselves.

Design Verification

Programmable logic designs are verified by using a simulator, which is a software program that confirms the functionality or timing of a circuit.

The industry-standard formats used ensure that designs can be reused. If a vendors changes its libraries, only a synthesis recompile is necessary.

Even if you decide to move to a different vendor and/or technology, you're just a compile away after selecting the new library. It's even design-tool independent, so you can try synthesis tools from different vendors and pick the best results.

IP cores are more commonly available in HDL format, since that makes them easier to modify and use with different device vendors.

After completing the design specification, you'll need to know if the circuit actually works as it's supposed to. That is the purpose of *design verification.*

A simulator simulates the circuit. You'll need to provide the design information (via the netlist after schematic capture or synthesis) and the specific input pattern, or *test vectors*, that you want checked. The simulator takes this information and determines the outputs of the circuit.

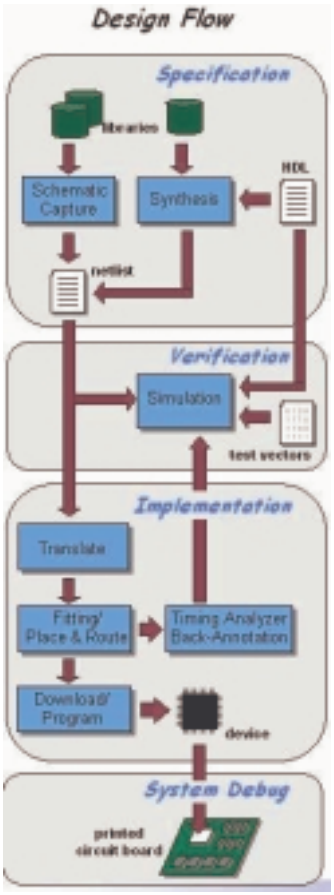


FIGURE 1-10: THE PLD DESIGN FLOW

Functional Simulation

At this point in the design flow, a *functional simulation* only checks that the circuits give the right combinations of ones and zeros. You would conduct a t*iming simulation* a little later in the design flow.

If there are any problems, you can go back to the schematic or HDL file, make changes, re-generate the netlist, and then rerun the simulation. Designers typically spend 50% of their development time going through this loop until the design works as required.

Using HDL offers an additional advantage when verifying the design: You can simulate directly from the HDL source file. This bypasses the time-consuming synthesis process that would normally be required for every design change iteration.

Once the circuit works correctly, running the synthesis tool generates the netlist for the next step in the design flow – *device implementation.*

Device Implementation

A design netlist completely describes the design using the gates for a specific vendor/device family. Once it's fully verified, it's time to put this in a chip, referred to as device implementation.

Translate comprises various programs used to import the design netlist and prepare it for layout. The programs will vary among vendors.

Some of the more common programs during translate include: optimization, translation to the physical device elements, and device-specific design rule checking (e.g,. does the design exceed the number of clock buffers available in this device?).

During the stage of the design flow, you will be asked to select the target device, package, speed grade, and any other device-specific options.

The translate step usually ends with a comprehensive report of the results of all the programs executed. In addition to warnings and errors is usually a listing of device and I/O utilization, which helps you to determine if you've selected the best device.

Fitting

For CPLDs, the design step is called *fitting*, meaning to "fit" the design to the target device. In the diagram above, a section of the design is fit to the CPLD.

CPLDs are a fixed architecture, so the software needs to pick the gates and interconnect paths that match the circuit. This is usually a fast process.

The biggest potential problem is if you had previously assigned the exact locations of the I/O pins, commonly referred to as *pin locking*. Most often, this occurs when using a legacy design iteration that has been committed to the printed circuit board layout.

Architectures that support I/O pin locking (such as the Xilinx XC9500 and CoolRunner CPLDs) have a very big advantage. They allow you to keep the

original I/O pin placements regardless of the number of design changes, utilization, or required performance.

Pin locking is very important when using ISP. If you layout your PCB to accept a specific pin out, and then change the design, you can re-program confident that you pin out will stay the same.

Place and Route

For FPGAs, place and route programs are run after compile. "Place" is the process of selecting specific modules, or logic blocks, in the FPGAs where design gates will reside.

"Route," as the name implies, is the physical routing of the interconnect between the logic blocks.

Most vendors provide automatic place and route tools so that you don't have to worry about the intricate details of the device architecture. Some vendors offer tools that allow expert users to manually place and/or route the most critical parts of their designs to achieve better performance than with the automatic tools. Floorplanner is a type of manual tool.

Place and route programs require the longest time to complete successfully because it's a complex task to determine the location of large designs, ensure that they all get connected correctly, and meet the desired performance.

These programs however, can only work well if the target architecture has sufficient routing for the design. No amount of fancy coding can compensate for an ill-conceived architecture, especially if there are not enough routing tracks.

If you were to encounter this problem, the most common solution would be to use a larger device. And you would likely remember the experience the next time you selected a vendor.

A related program is called *timing-driven place and route (TDPR).* This allows you to specify timing criteria that will be used during device layout.

A *static timing analyzer* is usually part of the vendor's implementation software. It provides timing information about paths in the design. This information is very accurate and can be viewed in many different ways, such as displaying all paths in the design and ranking them from longest to shortest delay.

In addition, at this point you can use the detailed layout information after reformatting and go back to your chosen simulator with detailed timing information.

This process is called *back-annotation* and has the advantage of providing the accurate timing as well as the zeros and ones operation of your design.

In both cases, the timing reflects delays of the logic blocks as well as the interconnect.

The final implementation step is the *download or program.*

Downloading or Programming

Download generally refers to volatile devices such as SRAM FPGAs. As the name implies, you download the device configuration information into the device memory.

The bitstream that is transferred contains all the information to define the logic and interconnect of the design and is different for every design.

Because SRAM devices lose their configuration when the power is turned off, the bitstream must be stored somewhere for a production solution. A common such place is a serial PROM. There is an associated piece of hardware that connects from the computer to a board containing the target device.

Program is used to program all non-volatile programmable logic devices, including serial PROMs. Programming performs the same function as download, except that the configuration information is retained after the power is removed from the device.

For antifuse devices, programming can only be done once per device – hence the term one-time programmable.

Programming of Xilinx CPLDs can be done in-system via JTAG or with a conventional device programmer such as Data I/O.

JTAG Boundary Scan – formally known as IEEE/ANSI standard 1149.1_1190 – is a set of design rules that facilitate testing, device programming, and debugging at the chip, board, and system levels.

In-system programming has an added advantage in that devices can be soldered directly to the PCB (such as TQFP surface-mount-type devices). If the design changes, the devices do not need to be removed from the board but simply re-programmed in-system.

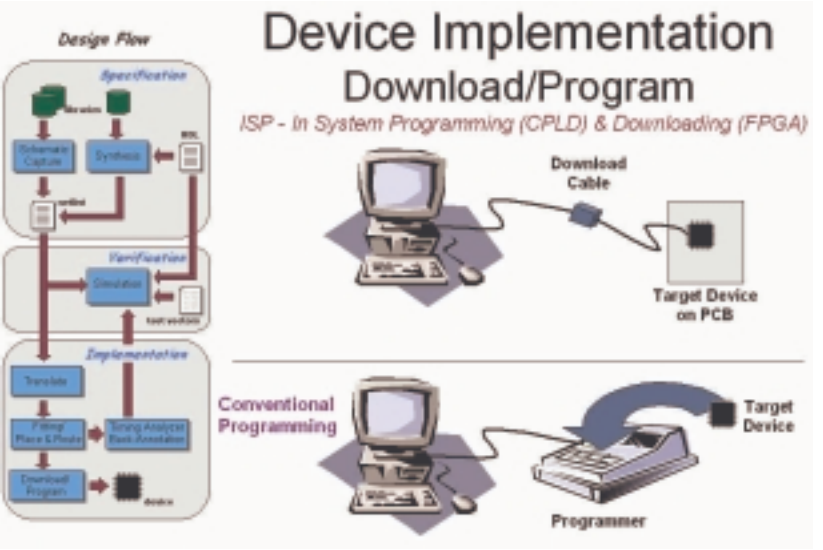


FIGURE 1-11: DEVICE IMPLEMENTATION – DOWNLOAD/PROGRAM

System Debug

The device is now working, but you still need to verify that the device works in the actual board, a process called *system debug*.

Any major problems here mean that you have made an assumption on the device specification that is incorrect, or have not considered some aspect of the signal required to/from the programmable logic device.

If so, you can collect data on the problem and go back to the drawing (or behavioral) board.

Xilinx has the world's first WebPOWERED™ programmable logic devices. This means we have the first WebFITTER™ CPLD design fitting tool, enabling you to fit your design in real time at our website.

Simply take your existing design to our WebFITTER web page – these files can be HDL source code or netlists – and specify your target device or your key design criteria, such as speed or low power. Then press "fit."

You will receive your results moments later via e-mail, which includes full fitter results, design files, and a programming file (JEDEC file). If you like the results, you can then obtain an online price.

You can download your personal copy in modules, so you can decide which parts you need. Modules include the design environment (Project Navigator), XST (Xilinx Synthesis Tool), ModelSim™ Xilinx Edition Starter (a third-party simulator), ChipViewer, and eventually ECS schematic capture and VSS.

Xilinx ChipViewer (a Java utility) graphically represents pin constraints and assignments. You can also use this tool to graphically view design implementations from the chip boundary to the individual macrocell equations.

CHAPTER 2

Xilinx Solutions

Introduction

Xilinx programmable logic solutions help minimize risks for electronic equipment manufacturers by shortening the time required to develop products and take them to market.

You can design and verify the unique circuits in Xilinx programmable devices much faster than by choosing traditional methods such as mask-programmed, fixed logic gate arrays.

Moreover, because Xilinx devices are standard parts that need only to be programmed, you are not required to wait for prototypes or pay large non-recurring engineering (NRE) costs.

Customers incorporate Xilinx programmable logic into products for a wide range of markets. Those include data processing, telecommunications, networking, industrial control, instrumentation, consumer electronics, automotive, defense, and aerospace markets.

Leading-edge silicon products, state-of-the-art software solutions, and world-class technical support make up the total solution that Xilinx delivers. The software component of this solution is critical to the success of every design project.

Xilinx Software Solutions provide powerful tools that make designing with programmable logic simple. Push-button design flows, integrated online help, multimedia tutorials, and high-performance automatic and auto-interactive tools help you achieve optimum results. In addition, the industry's broadest array of programmable logic technology and EDA integration options deliver unparalleled design flexibility.

Xilinx is also actively developing breakthrough technology that will enable the hardware in Xilinx-based systems to be upgraded remotely over any kind of network – including the Internet – even after the equipment has been shipped to a customer.

Xilinx "Online Upgradeable Systems" would allow equipment manufacturers to remotely add new features and capabilities to installed systems, or repair problems without having to physically exchange hardware.

Xilinx Devices

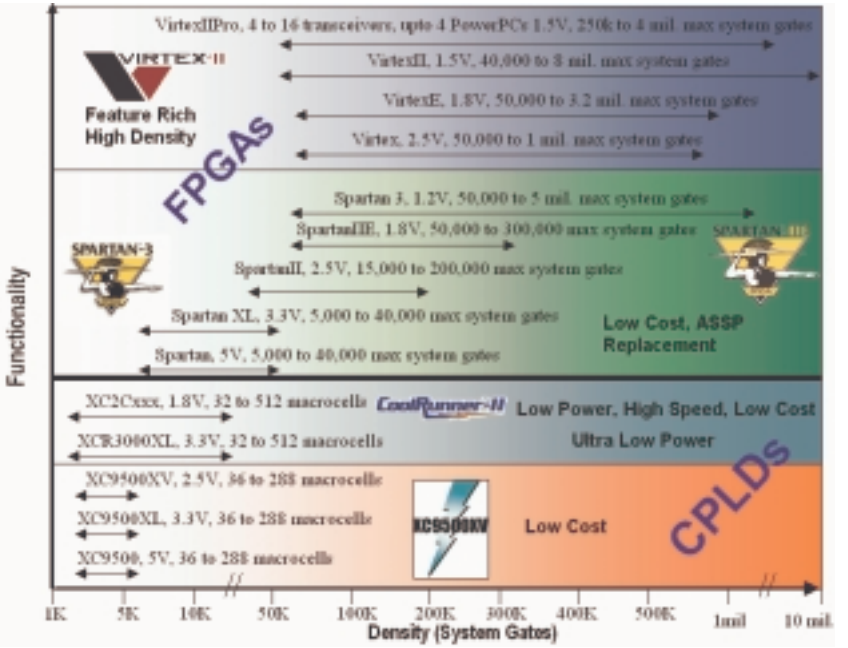


FIGURE 2-1: XILINX DEVICES AT A GLANCE

Platform FPGAs

VIRTEX FPGAS

The Virtex-II product is the first embodiment of the Platform FPGA, once again setting a new benchmark in performance and offering a feature set that is unparalleled.

It's an era where Xilinx leads the way, strengthened by our strategic alliances with IBM, Wind River Systems, Conexant, RocketChips™, The MathWorks, and other technology leaders.

The Platform FPGA delivers:

- SystemIO interfaces to bridge emerging standards
- XtremeDSP™ FPGA-based DSP solution for unprecedented DSP performance (as much as 100 times faster than the leading DSP processor)
- Coming soon, Empower! processor technology for flexible high-performance system processing needs.

With densities ranging from 40,000 to 10 million system gates, the Virtex-II solution delivers enhanced system memory and lightning-fast DSP through a flexible IP-immersion fabric.

Additionally, significant new capabilities address system-level design issues, including flexible system interfaces with signal integrity (SystemIO, DCI); complex system clock management (Digital Clock Manager); and on-board EMI management (EMIControl).

Virtex-II solutions are empowered by advanced design tools that drive time-to-market advantages through fast design, powerful synthesis, smart implementation algorithms, and efficient verification capabilities.

Not only does the fabric provide the ability to integrate a variety of soft IP, but it also has the capability of embedding hard IP cores such as processors and gigabit serial I/Os in future Virtex-II families.

VIRTEX-II PRO FPGAS

"The Platform for Programmable Systems'"

With as many as four IBM PowerPC™ 405 processors immersed into the industry's leading FPGA fabric; Xilinx/Conexant's flawless high-speed serial I/O technology; and Wind River Systems's cutting-edge embedded design tools, Xilinx delivers a complete development platform of infinite possibilities.

The Power of Xtreme Processing

Each PowerPC runs at 300+ MHz delivering 420 Dhrystone MIPS, supported by IBM CoreConnect™ bus technology. With the unique Xilinx IP immersion architecture, you can now harness the power of high-performance processors, along with easy integration of soft IP, into the industry's highest performance programmable logic.

XtremeDSP – The World's Fastest Programmable DSP Solution

The Xilinx XtremeDSP solution is the world's fastest programmable DSP solution. With as many as 556 embedded 18 x 18 multipliers; 10 Mb of embedded block RAM; and an extensive library of DSP algorithms and tools including System Generator for DSP, ISE, and Cadence™ Design Systems SPW, the XtremeDSP tool is the industry's premier programmable solution for enabling TeraMAC/s applications.

The Ultimate Connectivity Platform

The first programmable device to combine embedded processors along with 3.125 Gbps transceivers, the Virtex-II Pro™ series of FPGAs addresses all existing connectivity requirements as well as the emerging high-speed interface standards.

Xilinx RocketIO™ transceivers offer a complete serial interface solution, supporting 10 Gigabit Ethernet with XAUI, 3GIO, and SerialATA, among others. Our SelectIO™-Ultra technology supports 840 Mbps LVDS and high-speed single-ended standards such as XSBI and SFI-4.

The Power of Integration

In a single off-the-shelf programmable device, you can take advantage of microprocessors, the highest density of on-chip memory, multi-gigabit serial transceivers, digital clock managers, on-chip termination, and more. The result is a dramatic simplification of board layout, a reduced bill of materials, and unbeatable time to market.

Enabling a New Development Paradigm

For the first time, you can partition and repartition your systems between hardware and software at any time during the development cycle – even after the product has shipped.

This means you can optimize the overall system, guaranteeing your performance target in the most cost-efficient manner. You can also debug hardware and software simultaneously at speed.

Industry-Leading Tools

Optimized for the PowerPC, Wind River Systems's industry-proven embedded tools are the premier support for real-time microprocessor and logic designs. Driving the Virtex-II Pro FPGA is the Xilinx lightning-fast ISE software, the most comprehensive, easy-to-use development system available.

Virtex FPGAs

The Xilinx Virtex series was the first line of FPGAs to offer one million system gates. Introduced in 1998, the Virtex product line fundamentally redefined programmable logic by expanding the traditional capabilities of FPGAs to include a powerful set of features that address board level problems for high performance system designs.

The latest devices in the Virtex-E series, unveiled in 1999, offer more than three million system gates.

Virtex-EM devices, introduced in 2000 and the first FPGAs to be manufactured using an advanced copper process, offer additional on-chip memory for network switch applications

FIGURE 2-2: **PLATFORM FPGAS**

Spartan FPGAs

Xilinx Spartan FPGAs are ideal for low-cost, high-volume applications and are targeted as replacements for fixed-logic gate arrays and ASSP products such as bus interface chip sets. The five members of the family are the Spartan-3 (1.2V), Spartan-IIE (1.8V), Spartan-II (2.5V), Spartan XL (3.3V), and Spartan (5V) devices.

SPARTAN-3 FPGAS

The Spartan-3 (1.2V, 90 nm) FPGA is not only available for a very low cost, but it integrates many architectural features associated with high-end programmable logic. This combination of low cost and features makes it an ideal replacement for ASICs (gate arrays) and many ASSP devices.

For example, a Spartan-3 FPGA in a car multimedia system could absorb many system functions, including embedded IP cores, custom system interfaces, DSP, and logic. The diagram below shows such a system:

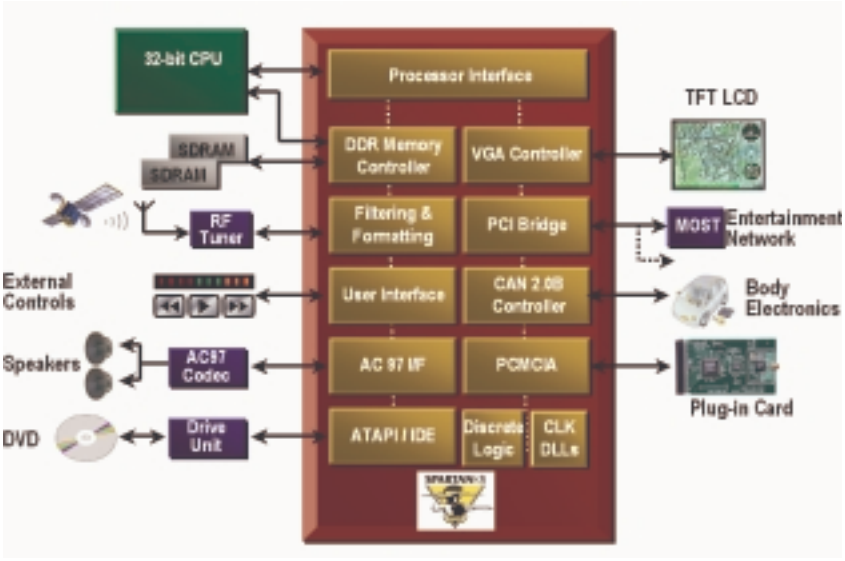


FIGURE 2-3: CAR MULTIMEDIA SYSTEM

In the car multimedia system shown in Figure 2-3, the PCI bridge takes the form of a pre-verified drop in IP core, and the device-level and board-level clocking functions are implemented in the Spartan-3 on-chip DCMs.

CAN core IP can connect to the body electronics modules. These cores are provided by Xilinx AllianceCORE™ partners such as Bosch, Memec Design, CAST, Inc., Xylon, and Intelliga.

On-chip 18 x 18 multipliers can be used in DSP-type activities such as filtering and formatting. Other custom-designed interfaces can be implemented to off-chip processors, an IDE interface to the drive unit of a DVD player, audio, memory, and LCD.

Additionally, the Spartan-3 XCITE digitally controlled impedance technology can reduce EMI and component count by providing on-chip tuneable impedances to provide line matching without the need for external resistors.

The Spartan-3 family is based on IBM and UMC advanced 90 nm, eight-layer metal process technology. Xilinx uses 90 nm technology to drive pricing down to under $20 for a one-million-gate FPGA (approximately 17,000 logic cells), which represents a cost savings as high as 80 percent compared to competitive offerings.

A smaller die size and 300 mm wafers improve device densities and yields, thereby reducing overall production costs. This in turn leads to a more highly integrated, less expensive product that takes up less board space when designed into an end product.

FIGURE 2-4: SPARTAN-3 FEATURES

The Spartan-3 FPGA memory architecture provides the optimal granularity and efficient area utilization.

Shift register SRL16 blocks

- Each CLB LUT works as a 16-bit fast, compact shift register
- Cascade LUTs to build longer shift registers
- Implement pipeline registers and buffers for video or wireless

As much as 520 Kb distributed SelectRAM™ memory

- Each LUT works as a single-port or dual-port RAM/ROM
- Cascade LUTs to build larger memories
- Applications include flexible memory sizes, FIFOs, and buffers

As much as 1.87 Mb Embedded block RAM

- As many as 104 blocks of synchronous, cascadable 18 Kb block RAM
- Configure each 18 Kb block as a single- or dual-port RAM
- Supports multiple aspect ratios, data-width conversion, and parity
- Applications include data caches, deep FIFOs, and buffers

Memory Interfaces

- Enable electrical interfaces such as HSTL and SSTL to connect to popular external memories

Multipliers

- Enable simple arithmetic and math as well as advanced DSP functions, enabling you to derive more than 330 billion MACs/s of DSP performance
- As many as 104 18 x 18 multipliers support 18-bit signed or 17-bit unsigned multiplication, which you can cascade to support wider bits
- Constant coefficient multipliers: On-chip memories and logic cells work hand-in-hand to build compact multipliers with a constant operand
- Logic Cell multipliers: Implement user-preferred algorithms such as Baugh-Wooley, Booth, Wallace tree, and others

DCMs deliver sophisticated digital clock management that's impervious to system jitter, temperature, voltage variations, and other problems typically found with PLLs integrated into FPGAs.

- Flexible frequency generation from 25 MHz to 325 MHz
  - 100 ps jitter
  - Integer multiplication and division parameters
- Quadrature and precision phase shift control
  - 0, 90, 180, 270 degrees
  - Fine grain control (1/256 clock period) for clock data synchronization
- Precise 50/50 duty cycle generation
- Temperature compensation

XCITE Digitally Controlled Impedance Technology – A Xilinx Innovation

I/O termination is required to maintain signal integrity. With hundreds of I/Os and advanced package technologies, external termination resistors are no longer viable.

I/O termination dynamically eliminates drive strength variation due to process, temperature, and voltage fluctuations.

Spartan-3 XCITE DCI Technology Highlights

- Series and parallel termination for single-ended and differential standards
- Maximum flexibility with support of series and parallel termination on all I/O banks
- Input, output, bidirectional, and differential I/O support
- Wide series impedance range
- Popular standard support, including LVDS, LVDSEXT, LVCMOS, LVTTL, SSTL, HSTL, GTL, and GTLP

Full- and half-impedance input buffers

XCITE DCI Technology Advantages

Advantage	Details
2nd generation technology	Proven in the field and used extensively by customers
Lowers cost	Fewer resistors, fewer PCB traces and smaller board area, result in lower PCB costs.
Absolute I/O Flexibility	Any termination on any I/O bank. Non-XCITE technology alternatives deliver limited functionality.
Maximum I/O Bandwidth	Less ringing and reflections maximize I/O bandwidth.
Immunity to temperature and voltage changes	Temperature and voltage variations lead to significant impedance mismatches. XCITE technology dynamically adjusts on-chip impedance to such variations reducing and improving reliability.
Eliminates stub reflection	Improves discrete termination techniques by eliminating the distance between the package pin and resistor.
Increases system reliability	Fewer components on board, deliver higher reliability

FIGURE 2-5: **XCITE DCI TECHNOLOGY**

Spartan-3 Features and Benefits

Spartan-3 FPGA

Device	XC 3S50	XC 3S200	XC 3S400	XC 3S1000	XC 3S1500	XC 3S2000	XC 3S4000	XC 3S5000
System Gates	50K	200K	400K	1000K	1500K	2000K	4000K	5000K
Logic Cells	1,728	4,320	8,064	17,280	29,952	46,080	62,208	74,880
18x18 Multipliers	4	12	16	24	32	40	96	104
Block RAM Bits	72K	216K	288K	432K	576K	720K	1,728K	1,872K
Distributed RAM Bits	12K	30K	56K	120K	208K	320K	432K	520K
DCMs	2	4	4	4	4	4	4	4
I/O Standards	23	23	23	23	23	23	23	23
Max Differential I/O Pairs	56	76	116	175	221	270	312	344
Max Single Ended I/O	124	173	264	391	487	565	712	784
Package	**User I/O**	**User I/O**	**User I/O**	**User I/O**	**User I/O**	**User I/O**	**User I/O**	**User I/O**
VQ100	63	63	-	-	-	-	-	-
TQ144	97	97	97	-	-	-	-	-
PQ208	124	141	141	-	-	-	-	-
FT256	-	173	173	173	-	-	--	-
FG456	-	-	264	333	333	-	-	-
FG676	-	-	-	391	487	489	-	-
FG900	-	-	-	-	-	565	633	633
FG1156	-	-	-	-	-	-	712	784

FIGURE 2-6: **SPARTAN-3 FPGA FAMILY OVERVIEW**

TABLE 2-1: SPARTAN-3 FEATURES AND BENEFITS

Spartan-3 Feature	Benefit
FPGA fabric and routing, up to 5,000,000 system gates	Allows for implementation of system level function blocks, high on-chip connectivity and high-throughput
Block RAM – 18k blocks	Enables implementation of large packet buffers/FIFOs, line buffers
Distributed RAM	For implementing smaller FIDOs/Buffers, DSP coefficients
Shift register mode (SRL16)	16-bit shift register ideal for capturing high speed or burst mode data and to store data in DSP and encryption applications e.g. fast pipelining
Dedicated 18 x 18 multiplier blocks	High speed DSP processing; use of multipliers in conjunction with fabric allows for ultra-fast, parallel DSP operations
Single-ended signalling (up to 622 Mbps) – LVTTL, LVCMOS, GTL, GTL+, PCI, HSTL-I, II, III, SSTL-I,II	Connectivity to commonly used chip-to-chip, memory (SRAM, SDRAM) and chip-to-backplane signalling standards; eliminates the need for multiple translation ICs
Differential signalling (up to 622 Mbps) - LVDS, BLVDS, Ultra LVD, SRSDS and LDT	Differential signalling at low cost – bandwidth management (saving the number of pins, reduced power consumption, reduced EMI, high noise immunity
Digital clock management (DCM)	Eliminate on-chip and board level clock delay, simultaneous multiply and divide, reduction of board level clock speed and number of board level clocks, adjustable clock phase for ensuring coherency
Global routing resources	Distribution of clocks and other signals with very high fanout throughout the device
Programmable output drive	Improves signal integrity, achieving right trade off between Tco and ground bounce

SPARTAN-IIE FPGAS

The Spartan-IIE (1.8V core) family of FPGAs offers some of the most advanced FPGA technologies available today, including programmable support for multiple I/O standards (including LVDS, LVPECL, and HSTL); on-chip block RAM; and digital delay lock loops for both chip-level and board-level clock management.

In addition, Spartan-IIE devices provide superior value by eliminating the need for many simple ASSPs such as phase lock loops, FIFOs, I/O translators, and system bus drivers that in the past have been necessary to complete a system design.

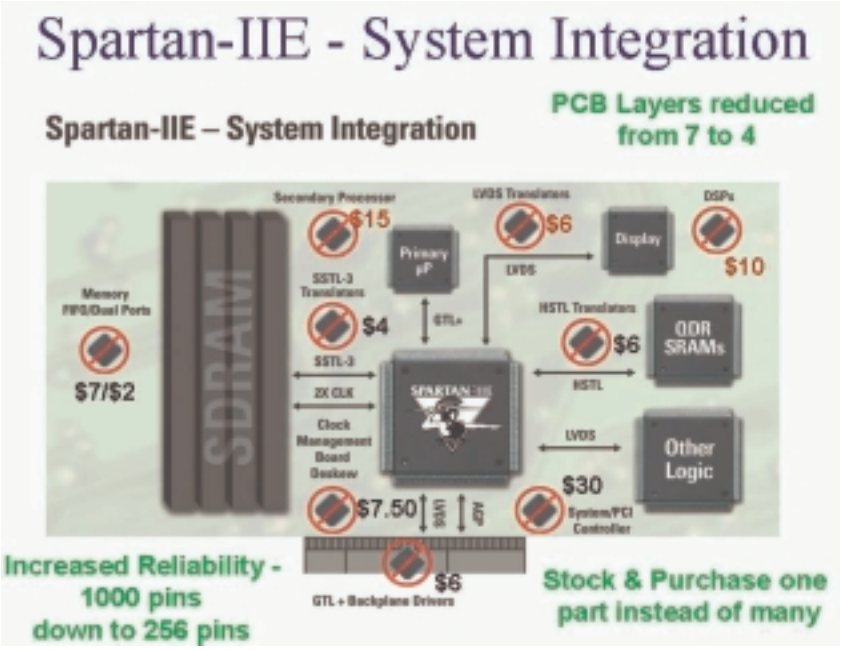


FIGURE 2-7: SPARTAN-IIE SYSTEM INTEGRATION

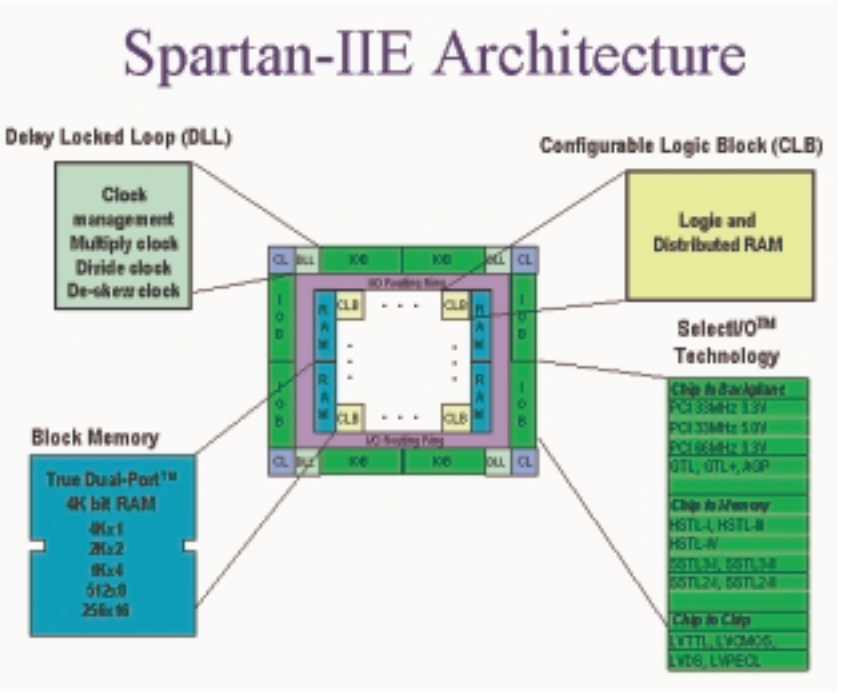


FIGURE 2-8: SPARTAN-IIE ARCHITECTURE

SPARTAN-IIE ARCHITECTURAL FEATURES

Spartan-IIE devices leverage the basic feature set of the Virtex-E architecture to offer outstanding value. The basic CLB structure contains distributed RAM and performs basic logic functions.

Four DLLs are used for clock management and can perform clock de-skew, clock multiplication, and clock division. Clock de-skew can be done on an external (board level) or internal (chip level) basis.

The block memory blocks are 4 Kb each and can be configured from 1 to 16 bits wide. Each of the two independent ports can be configured independently for width.

The SelectIO feature allows many different I/O standards to be implemented in the areas of chip-to-chip, chip-to-memory, and chip-to-backplane interfaces.

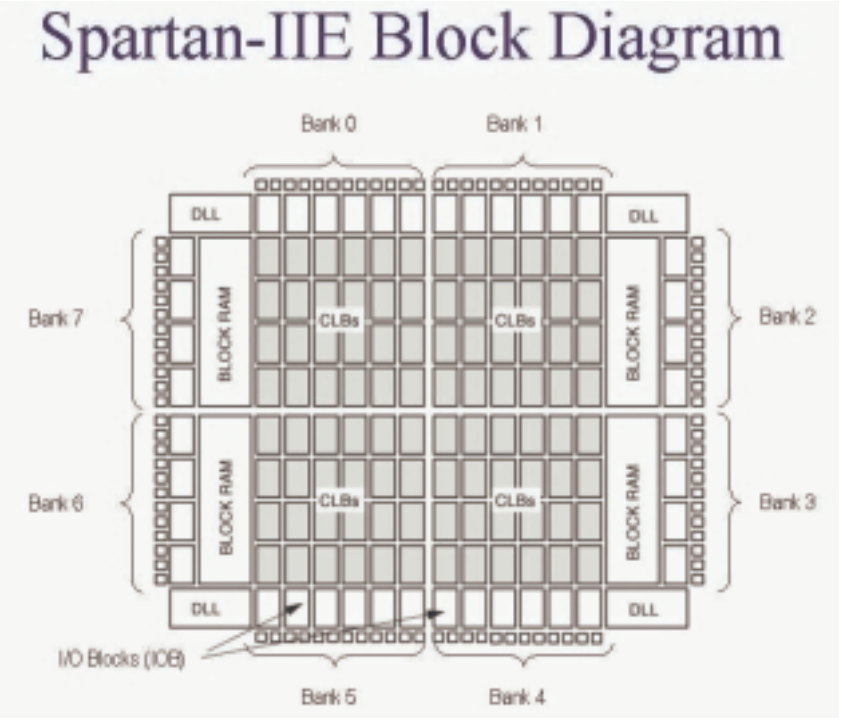


FIGURE 2-9: **SPARTAN-IIE BLOCK DIAGRAM**

The Spartan-IIE family of FPGAs is implemented with a regular, flexible, programmable architecture of CLBs, surrounded by a perimeter of programmable IOBs, interconnected by a powerful hierarchy of versatile routing resources.

The architecture also provides advanced functions such as block RAM and clock control blocks.

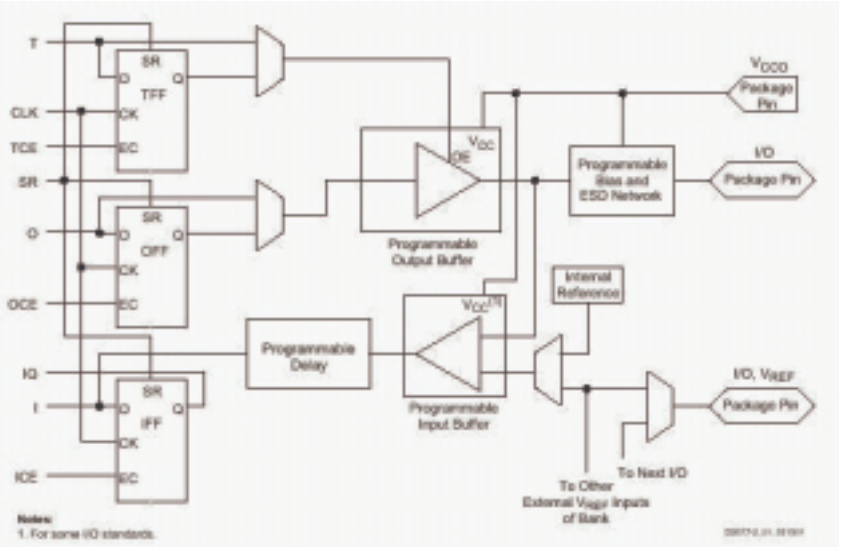


FIGURE 2-10: SPARTAN-IIE INPUT/OUTPUT BLOCK

The Spartan-IIE IOB features inputs and outputs that support 19 I/O signalling standards, including LVDS, BLVDS, LVPECL, LVCMOS, HSTL, SSTL, and GTL.

These high-speed inputs and outputs are capable of supporting various state-of-the-art memory and bus interfaces. Three IOB registers function either as edge-triggered D-type flip-flops or as level-sensitive latches. Each IOB has a CLK shared by the three registers and independent CE signals for each register.

In addition to the CLK and CE control signals, the three registers share a set/reset. For each register, you can independently configure this signal as a synchronous set, a synchronous reset, an asynchronous preset, or an asynchronous clear.

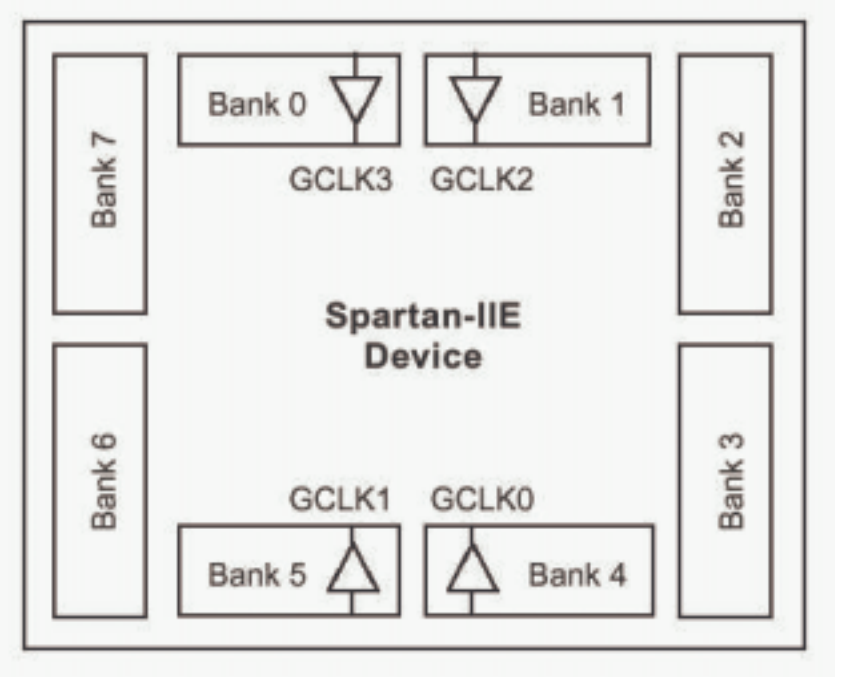


FIGURE 2-11: SPARTAN-IIE BANKING OF I/O STANDARDS

Some of the I/O standards require VCCO and/or VREF voltages. These voltages are connected externally to device pins that serve groups of IOBs, called banks.

Consequently, restrictions exist about which I/O standards can be combined within a given bank. Eight I/O banks result from separating each edge of the FPGA into two banks. Each bank has multiple VCCO pins, all of which must be connected to the same voltage. This voltage is determined by the output standards in use.

Logic Cells

The basic building block of the Spartan-IIE CLB is the logic cell. A logic cell includes a four-input function generator, carry logic, and a storage element.

The output from the function generator in each logic cell drives both the CLB output and the D input of the flip-flop. Each Spartan-IIE CLB contains four logic cells, organized in two similar slices.

In addition to the four basic logic cells, the Spartan-IIE CLB contains logic that combines function generators to provide functions of five or six inputs. Consequently, when estimating the number of system gates provided by a given device, each CLB counts as 4.5 logic cells.

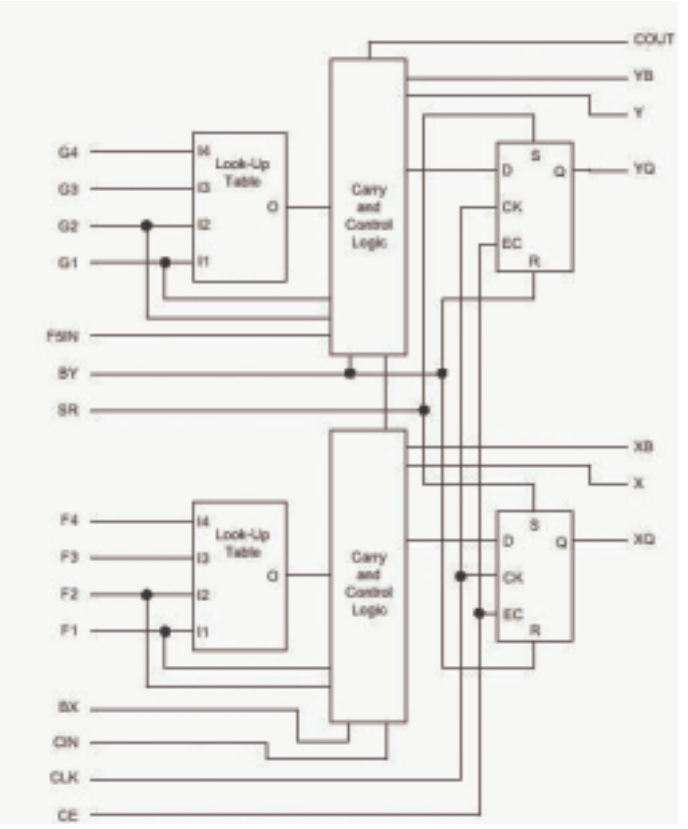


FIGURE 2-12: **SPARTAN-IIE LOGIC CELL**

Spartan-IIE function generators are implemented as 4-input LUTs. In addition to operating as a function generator, each LUT can provide a 16 x 1-bit synchronous RAM.

Furthermore, two LUTs within a slice can be combined to create a 16 x 2-bit or 32 x 1-bit synchronous RAM, or a 16 x 1-bit dual-port synchronous RAM.

The Spartan-IIE LUT can also provide a 16-bit shift register that is ideal for capturing high-speed or burst-mode data. This SRL16 mode can increase the effective number of flip-flops by a factor of 16.

Adding flip-flops enables fast pipelining, which is ideal for DSP applications. The storage elements in the Spartan-IIE slice can be configured either as edge-triggered D-type flip-flops or as level-sensitive latches.

Block RAM

Spartan-IIE FPGAs incorporate several large block SelectRAM+™ memories. These complement the distributed SelectRAM+ resources that provide shallow RAM structures implemented in CLBs.

Block SelectRAM+ memory blocks are organized in columns. All Spartan-II devices contain two such columns, one along each vertical edge. These columns extend the full height of the chip.

Each memory block is four CLBs high, and consequently, a Spartan-IIE device eight CLBs high will contain two memory blocks per column, and a total of four blocks.

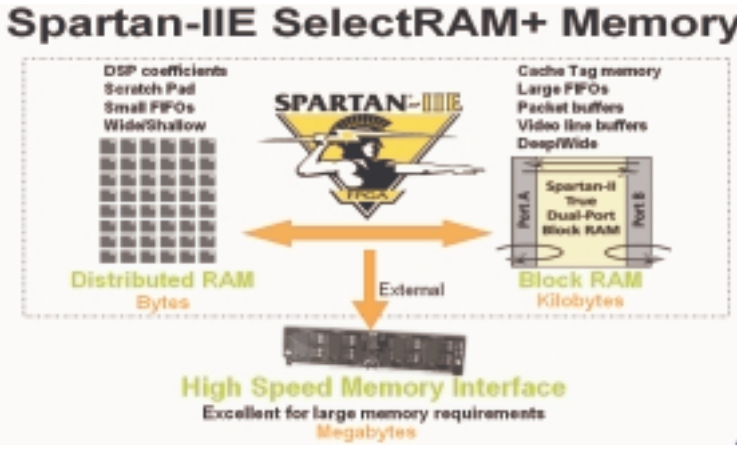


FIGURE 2-13: SPARTAN-IIE ON-CHIP MEMORY

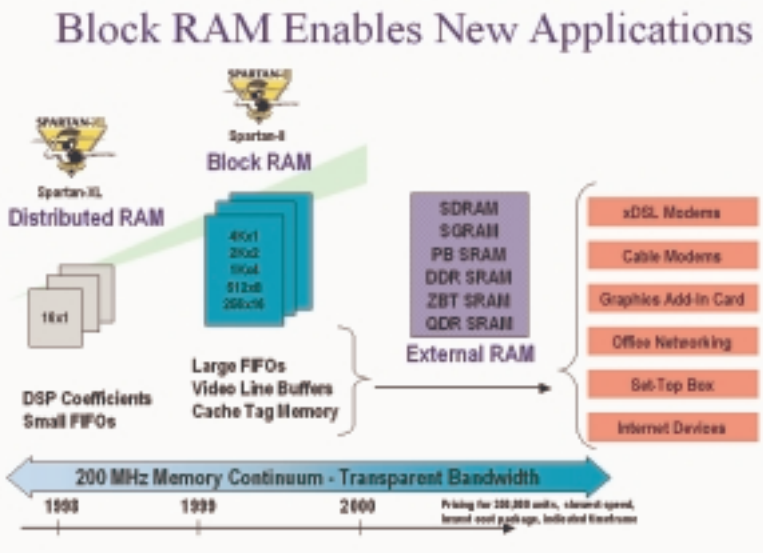


FIGURE 2-14: BLOCK RAM APPLICATIONS

Delay-Locked Loop

Associated with each global clock input buffer is a fully digital DLL that can eliminate skew between the clock input pad and internal clock input pins throughout the device.

Each DLL drives two global clock networks. The DLL monitors the input clock and the distributed clock, and automatically adjusts a clock delay element. Additional delay is introduced such that clock edges reach internal flip-flops exactly one clock period after they arrive at the input.

This closed-loop system effectively eliminates clock-distribution delay by ensuring that clock edges arrive at internal flip-flops in synch with clock edges arriving at the input.

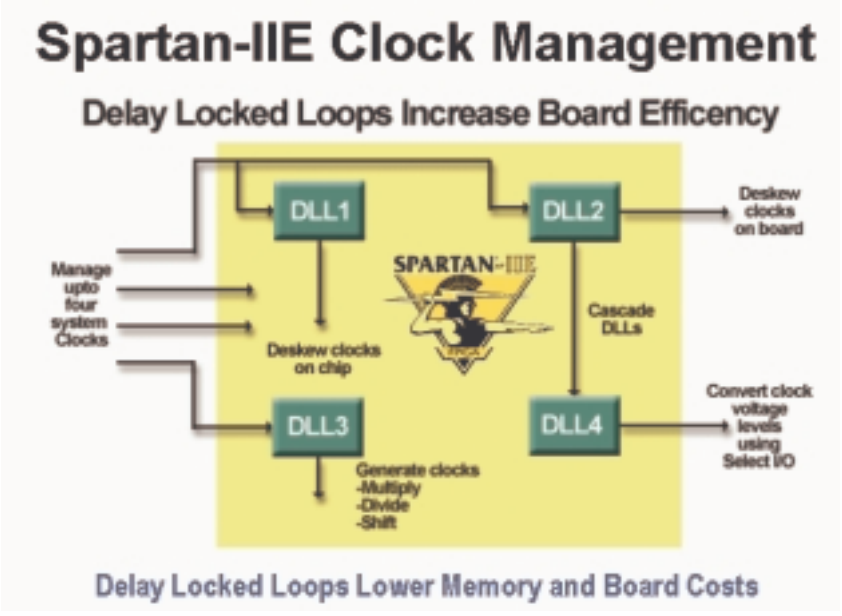
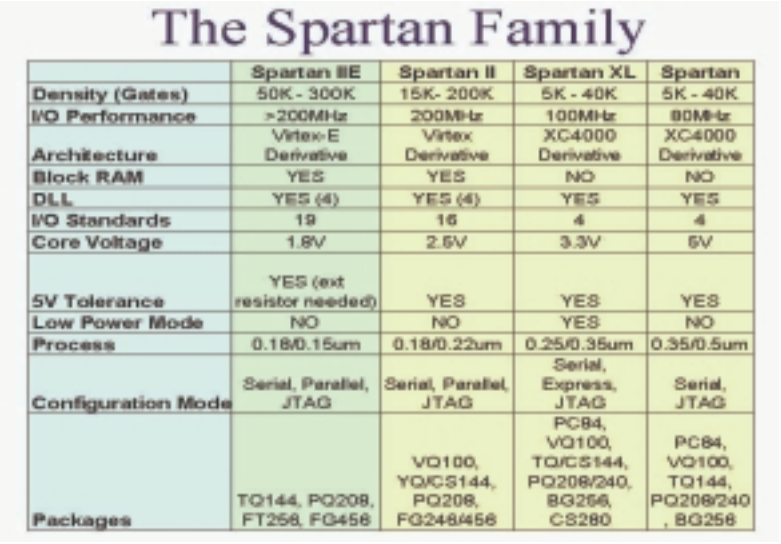


FIGURE 2-15: SPARTAN-IIE CLOCK MANAGEMENT



The Spartan Family

	Spartan IIE	Spartan II	Spartan XL	Spartan
Density (Gates)	50K - 300K	15K- 200K	5K - 40K	5K - 40K
I/O Performance	>200MHz	200MHz	100MHz	80MHz
Architecture	Virtex-E Derivative	Virtex Derivative	XC4000 Derivative	XC4000 Derivative
Block RAM	YES	YES	NO	NO
DLL	YES (4)	YES (4)	YES	YES
I/O Standards	19	16	4	4
Core Voltage	1.8V	2.5V	3.3V	5V
5V Tolerance	YES (ext resistor needed)	YES	YES	YES
Low Power Mode	NO	NO	YES	NO
Process	0.18/0.15um	0.18/0.22um	0.25/0.35um	0.35/0.5um
Configuration Mode	Serial, Parallel, JTAG	Serial, Parallel, JTAG	Serial, Express, JTAG	Serial, JTAG
Packages	TQ144, PQ208, FT256, FG456	VQ100, VQ/CS144, PQ208, FG246/456	PC84, VQ100, TQ/CS144, PQ208/240, BG256, CS280	PC84, VQ100, TQ144, PQ208/240, BG256

FIGURE 2-16: SPARTAN FAMILY COMPARISON

Configuration

Configuration is the process by which the FPGA is programmed with a configuration file generated by the Xilinx development system. Spartan-IIE devices support both serial configuration, using the master/slave serial and JTAG modes, as well as byte-wide configuration employing the slave parallel mode.

Spartan-IIE Family Matrix

System Gates	50K	100K	150K	200K	300K
Logic Cells	1,728	2,700	3,888	5,292	6,912
Block RAM Bits	32K	40K	48K	56K	64K
DLLs	4	4	4	4	4
I/O Standards	19	19	19	19	19
Max Differntial I/O Pairs	84	86	114	120	120
Max Single Ended I/O	182	202	263	289	329
Packages (I/O Max)	TQ144 (102)	TQ144 (102)			
	PQ208 (146)	PQ208 (146)	PQ208 (146)	PQ208 (146)	PQ208 (146)
	FT256 (182)	FT256 (182)	FT256 (182)	FT256 (182)	FT256 (182)
		FG456 (202)	FG456 (263)	FG456 (289)	FG456 (329)

FIGURE 2-17: SPARTAN-IIE FAMILY OVERVIEW

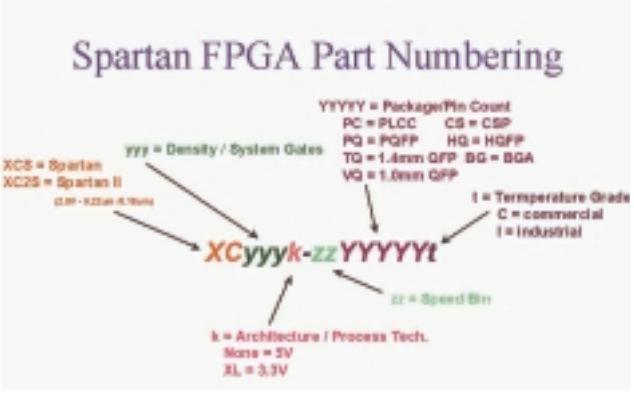


FIGURE 2-18: SPARTAN FPGA PART NUMBERING GUIDE

Xilinx CPLDs

Currently, Xilinx offers CPLD products in two categories: XC9500 and CoolRunner devices. To choose a CPLD that's right for you, review the product features below to identify the product family that fits your application. You should also review the selection considerations to choose the device that best meets your design criteria.

Product Features:

XC9500 Device – The XC9500 ISP CPLD families take complex programmable logic devices to new heights of performance, features, and flexibility.

These families deliver industry-leading speeds while providing the flexibility of enhanced customer-proven pin-locking architecture, along with extensive IEEE Std.1149.1 JTAG Boundary Scan support. This CPLD family is ideal for high-speed, low-cost designs.

CoolRunner Device – The CoolRunner CPLD families offer extreme low power, making them the leaders in an all-new market segment: portable electronics.

With standby current in the low micro amps and minimal operational power consumption, these parts are ideal for any application is that is especially power sensitive, such as battery-powered or portable applications.

The CoolRunner-II CPLD extends usage as it offers system-level features such as LVTTL and SSTL, clocking modes, and input hysteresis.

Selection Considerations:

To decide which device best meets your design criteria, take a minute to jot down your design specs (using the list below as a criteria reference). Next, go to a specific product family page to get more detailed information about the device you need.

Density – Each part gives an equivalent "gate count," or estimate of the logic density of the part.

Number of Registers – Count up the number of registers you need for your counters, state machines, registers, and latches. The number of macrocells in the device must be at least this large.

Number of I/O Pins – How many inputs and outputs does your design need?

Speed Requirements – What is the fastest combinatorial path in your design? This will determine the Tpd (in nanoseconds) of the device. What is the fastest sequential circuit in your design? This will tell you what fMax you need.

Package – What electromechanical constraints are you under? Do you need the smallest ball grid array package possible, or can you use a more ordinary QFP? Or are you prototyping and need to use a socketed device, such as a PLCC package?

Low Power – Is your end product battery- or solar-powered? Does your design require the lowest power devices possible? Do you have heat dissipation concerns?

System-Level Functions – Does your board have multi-voltage devices? Do you need to level shift between these devices? Do you need to square up clock edges? Do you need to interface to memories and microprocessors?

XC9500 ISP CPLD OVERVIEW

The high-performance, low-cost XC9500 families of Xilinx CPLDs are targeted for leading-edge systems that require rapid design development, longer system life, and robust field upgrade capability.

The XC9500 families range in density from 36 to 288 macrocells and are available in 2.5-volt (XC9500XV), 3.3-volt (XC9500XL) and 5-volt (XC9500) versions.

These devices support ISP, which allows manufacturers to perform unlimited design iterations during the prototyping phase, extensive system in-board debugging, program and test during manufacturing, and field upgrades.

Based on advanced process technologies, the XC9500 families provide fast, guaranteed timing; superior pin locking; and a full JTAG-compliant interface. All XC9500 devices have excellent quality and reliability characteristics with a 10,000 program/erase cycle endurance rating and 20-year data retention.

XC9500 5V Family

The XC9500 ISP CPLD family features six devices ranging from 36 to 288 macrocells, with a wide variety of package combinations that both minimize board space and maintain package footprints as designs grow or shrink.

The I/O pins allow direct interfacing to both 3- and 5-volt systems, while the latest in compact, easy-to-use CSP and BGA packaging gives you access to as many as 192 signals.

Flexible Pin-Locking Architecture

XC9500 devices, in conjunction with our fitter software, give you the maximum in routeability and flexibility while maintaining high performance. The architecture is feature-rich, including individual product term (p-term) output enables, three global clocks, and more p-terms per output than any other CPLD.

The proven ability of the architecture to adapt to design changes while maintaining pin assignments has been demonstrated in countless real-world customer designs since the introduction of the XC9500 family.

Full IEEE 1149.1 JTAG Development and Debugging Support

The JTAG capability of the XC9500 family is the most comprehensive of any CPLD on the market. It features the standard support including BYPASS, SAMPLE/PRELOAD, and EXTEST.

Additional Boundary Scan instructions, not found in any other CPLD, include INTEST (for device functional test), HIGHZ (for bypass), and USERCODE (for program tracking), for maximum debugging capability.

The XC9500 family is supported by a wide variety of industry-standard third-party development and debugging tools including Corelis, JTAG Technologies, and Asset Intertech. These tools allow you to develop Boundary Scan test vectors to interactively analyze, test, and debug system failures. The family is also supported on all major ATE platforms, including Teradyne, Hewlett Packard, and Genrad.

XC9500 Product Overview Table

TABLE 2-2: **XC9500 PRODUCT OVERVIEW**

	XC9536	XC9572	XC95108	XC95144	XC95216	XC95288
Macrocells	36	72	108	144	216	288
Usable Gates	800	1600	2400	3200	4800	6400
t_{PD} (ns)	5	7.5	7.5	7.5	10	15
Registers	36	72	108	144	216	288
Max. User I/Os	34	72	108	133	166	192
Packages	44VQ 44PC 48CSP	44PC 84PC 100TQ 100PQ	84PC 100TQ 100PQ 160PQ	100TQ 100PQ 160PQ	160PQ 208HQ 352BG	208HQ 352BG

XC9500XL 3.3V FAMILY

The XC9500XL CPLD family is targeted for leading-edge systems that require rapid design development, longer system life, and robust field upgrade capability.

This 3.3V ISP family provides unparalleled performance and the highest programming reliability, with the lowest cost in the industry.

XC9500XL CPLDs also complement the higher-density Xilinx FPGAs to provide a total logic solution, within a unified development environment. The

XC9500XL family is fully WebPOWERED via its free WebFITTER CPLD design fitting tool and WebPACK ISE software.

Family Highlights

- Lowest cost per macrocell
- State-of-the-art pin-locking architecture
- Highest programming reliability reduces system risk
- Complements Xilinx 3.3V FPGA families

Performance

- 5 ns pin-to-pin speed
- 222 MHz system frequency

Powerful Architecture

- Wide 54-input function blocks
- As many as 90 product-terms per macrocell
- Fast and routable Fast CONNECT™ II switch matrix
- Three global clocks with local inversion
- Individual OE per output, with local inversion

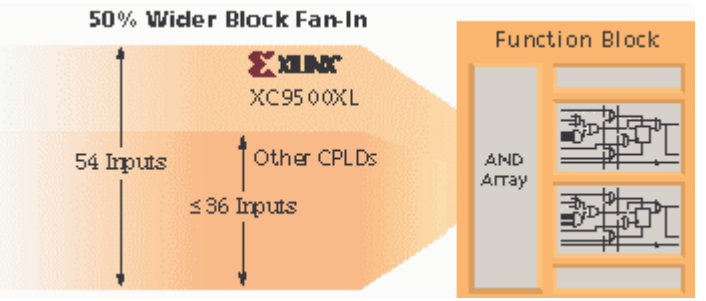


FIGURE 2-19: **XC9500XL BLOCK FAN-IN**

Highest Reliability

- Endurance rating of 10,000 cycles
- Data retention rating of 20 years
- Immune from "ISP Lock-Out" failure mode
- Allows arbitrary mixed-power sequencing and waveforms
-

Advanced Technology

- Third-generation, proven CPLD technology

- Mainstream, scalable, high-reliability processing
- Fast ISP and erase times

Outperforms All Other 3.3V CPLDs

- Extended data retention supports longer system operating life
- Virtually eliminates ISP failures
- Superior pin-locking for lower design risk
- Glitch-free I/O pins during power-up
- Full IEEE 1149.1 (JTAG) ISP and Boundary Scan testing
- Free WebPOWERED software

XC9500XV 2.5V CPLD FAMILY

The Xilinx XC9500XV 2.5V CPLD family is based on an advanced architecture that combines system flexibility and low cost to allow for faster time to market and lower manufacturing and support costs.

Designed to operate with an internal core voltage of 2.5V, the XC9500XV silicon offers 30% lower power consumption than 3.3V CPLDs, resulting in lower heat dissipation and increased long-term device reliability.

The XC9500XV silicon plus the powerful WebPOWERED software offers a valuable logic solution that can't be beat when it comes to cost and ease of use.

High Performance Through Advanced Technology

Manufactured on the latest generation 0.25 µm process, the new XC9500XV CPLDs provide the same advanced architectural features and densities of the 3.3V XC9500XL family, with device offerings of 36, 72, 144, and 288 macrocells.

A high-performance version offering pin-to-pin delays as low as 3.5 ns and system frequencies as fast as 275 MHz will be available later this year.

The 2.5V XC9500XV devices also include optimized support for ISP through the industry's most extensive IEEE1149.1 JTAG and IEEE 1532 programming capability, which helps to streamline the manufacturing, testing, and programming of CPLD-based electronic products, including remote field upgrades.

The System Designer's CPLD

The advanced architecture employed in the XC9500XV CPLD allows for easy design integration, thus empowering you to fully concentrate on your system design and not so much on chip-level details.

The unique features offered in the XC9500XV include a 54-input block fan-in, which contributes to the device's superior pin-locking capability; built-in input hysteresis for improved noise margin; bus-hold circuitry for better I/O control; hot-plugging capability to eliminate the need for power sequencing; and local and global clock control to provide maximum flexibility.

TABLE 2-3: **XC9500XV AND XC9500XL PRODUCT SELECTION**

3.3v ISP 2.5v ISP	XC9536XL XC9536XV	XC9572XL XC9572XV	XC95144XL XC95144XV	XC95288XL XC95288XV
Macrocells	36	72	144	288
Usable Gates	800	1600	3200	6400
t_{pd} (ns) XC9500XL	5	5	5	7.5
t_{pd} (ns) XC9500XV	4	5	5	6
Registers	36	72	144	288
f_{SYSTEM} XC9500XL XC9500XV	178 200	178 178	178 178	125 151
Packages	PC44 CS48 VQ44* VQ64	PC44 CS48 VQ44* VQ64 TQ100	TQ100 TQ144 CS144	TQ144 PQ208 BG256 FG256* CS280*

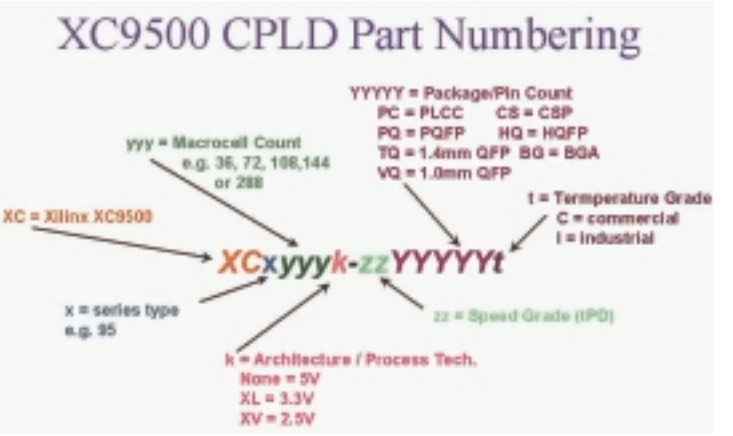


FIGURE 2-20: **XC9500 PART NUMBERING SYSTEM**

COOLRUNNER LOW-POWER CPLDS

There are two members to the CoolRunner series, CoolRunner XPLA3 device (3.3V) and the CoolRunner-II (1.8V) device. Let's look at the CoolRunner XPLA3 devices.

CoolRunner CPLDs combine very low power with high speed, high density, and high I/O counts in a single device. The CoolRunner 3.3V family ranges in density from 32 to 512 macrocells.

CoolRunner CPLDs feature Fast Zero Power™ technology, allowing the devices to draw virtually no power in standby mode. This makes them ideal for the fast-growing market of battery-operated portable electronic equipment, such as:

- Laptop PCs
- Telephone handsets
- Personal digital assistants
- Electronic games
- Web tablets.

These CPLDs also use far less dynamic power during actual operation compared to conventional CPLDs, an important feature for high-performance, heat-sensitive equipment such as telecom switches, video conferencing systems, simulators, high-end testers, and emulators.

FIGURE 2-21: **SENSE AMPLIFIER VS. CMOS CPLDS**

The CoolRunner XPLA3 eXtended family of CPLDs is targeted for low-power applications that include portable, handheld, and power-sensitive applications.

Each member of the XPLA3 family includes Fast Zero Power design technology that combines low power *and* high speed. With this design technique, the XPLA3 family offers true pin-to-pin speeds of 5.0 ns, while simultaneously delivering power that is <100 µA (standby) without the need for special "power down bits" that can negatively affect device performance.

By replacing conventional amplifier methods for implementing product terms (a technique that has been used in PLDs since the bipolar era) with a cas-

caded chain of pure CMOS gates, the dynamic power is also substantially lower than any competing CPLD.

CoolRunner devices are the only total CMOS PLDs, as they use both a CMOS process technology and the patented full CMOS Fast Zero Power design technique.

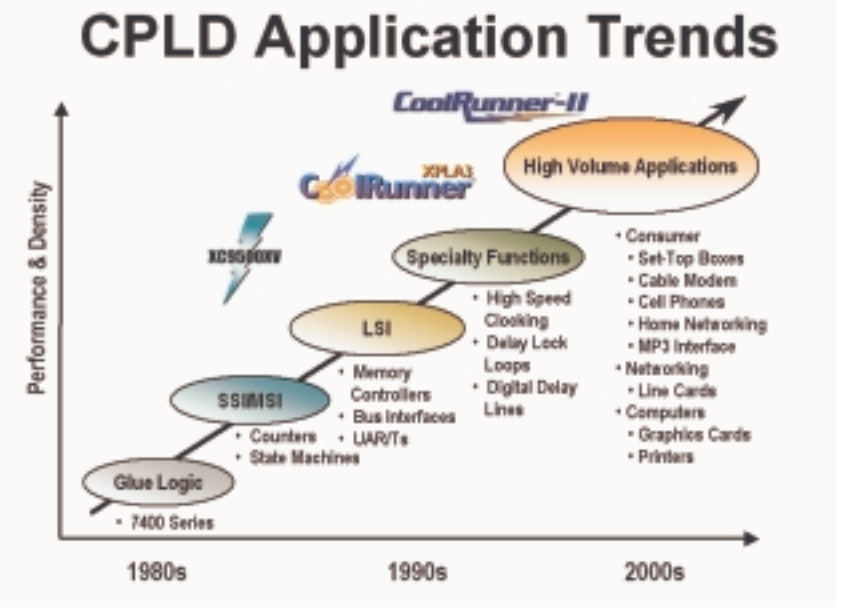


FIGURE 2-22: **CPLD APPLICATION TRENDS**

XPLA3 Architecture

The XPLA3 architecture features a direct input register path, multiple clocks, JTAG programming, 5V-tolerant I/Os, and a full PLA structure. These enhancements deliver high speed coupled with flexible logic allocation, which results in the ability to make design changes without changing pin-outs.

The XPLA3 architecture includes a pool of 48 product terms that can be allocated to any macrocell in the logic block. This combination allows logic to be allocated efficiently throughout the logic block and support as many product terms as needed per macrocell. In addition, using a variable number of product terms per macrocell incurs no speed penalty.

The XPLA3 family features industry-standard IEE 1149.1 JTAG interface, through which (ISP) and reprogramming of the device can occur. The XPLA3 CPLD is electrically reprogrammable using industry-standard device programmers from vendors such as Data I/O, BP Microsystems, and SMS.

Figure 2-23 shows a high-level block diagram of the XPLA3 architecture. The XPLA3 architecture comprises logic blocks inter-connected by ZIA. The ZIA is a virtual cross point switch. Each logic block has 36 inputs from the ZIA and 16 macrocells.

From this point of view, this architecture looks like many other CPLD architectures. What makes the XPLA3 family unique is logic allocation inside each logic block and the design technique used to implement these logic blocks.

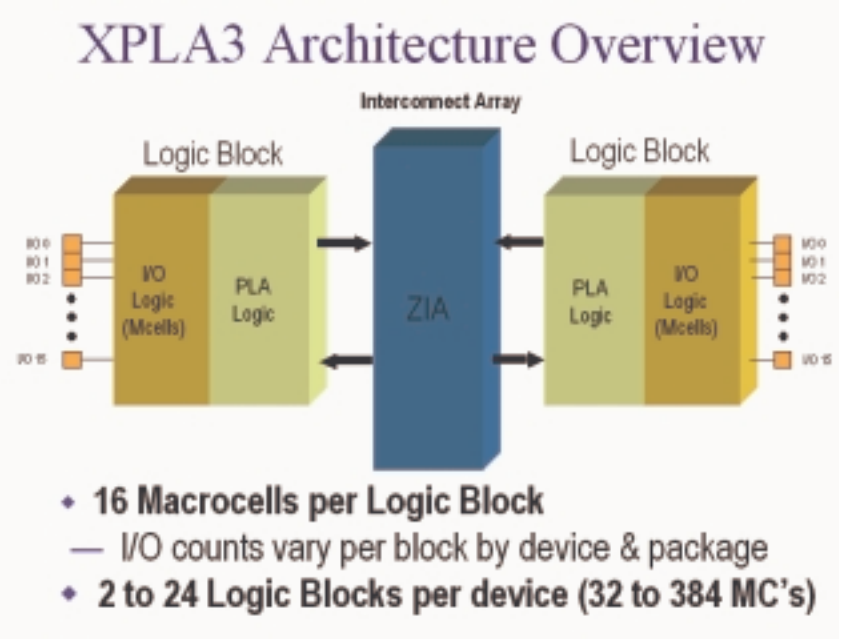


FIGURE 2-23: COOLRUNNER XPLA3 ARCHITECTURE OVERVIEW

Logic Block Architecture

Figure 2-24 illustrates the logic block architecture of CoolRunner XPLA CPLDs.

Each logic block contains a PLA array that generates control terms and macrocells for use as asynchronous clocks, resets, presets, and output enables.

The other p-terms serve as additional single inputs into each macrocell. There are eight FoldBack NAND p-terms that are available for ease of fitting and pin locking.

Sixteen product terms are coupled with the associated programmable OR gate into the VFM. The VFM increases logic optimization by implementing any two input logic functions before entering the macrocell.

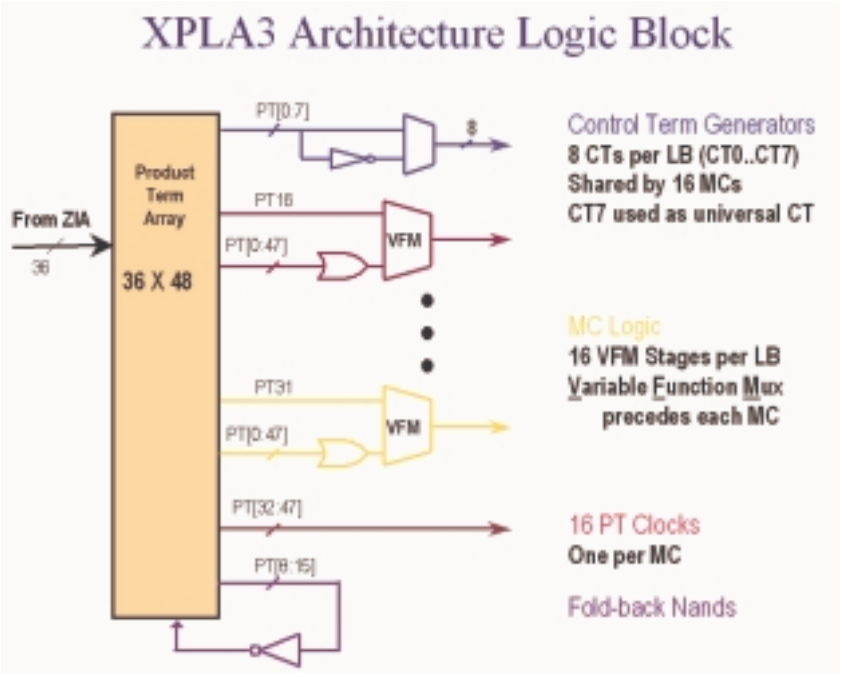


FIGURE 2-24: COOLRUNNER XPLA3 LOGIC BLOCK

Each macrocell supports combinatorial or registered inputs, preset and reset, configurable D, T, or latch functions. If a macrocell needs more product terms, it simply gets the additional product terms from the PLA array.

FoldBack NANDs

XPLA3 utilizes FoldBack NANDs to increase the effective product term width of a programmable logic device. These structures effectively provide an inverted product term to be used as a logic input by all of the local product terms.

Macrocell Architecture

Figure 2-25 shows the architecture of a macrocell used in the CoolRunner XPLA3 CPLD. Any macrocell can be reset or pre-set on power-up.

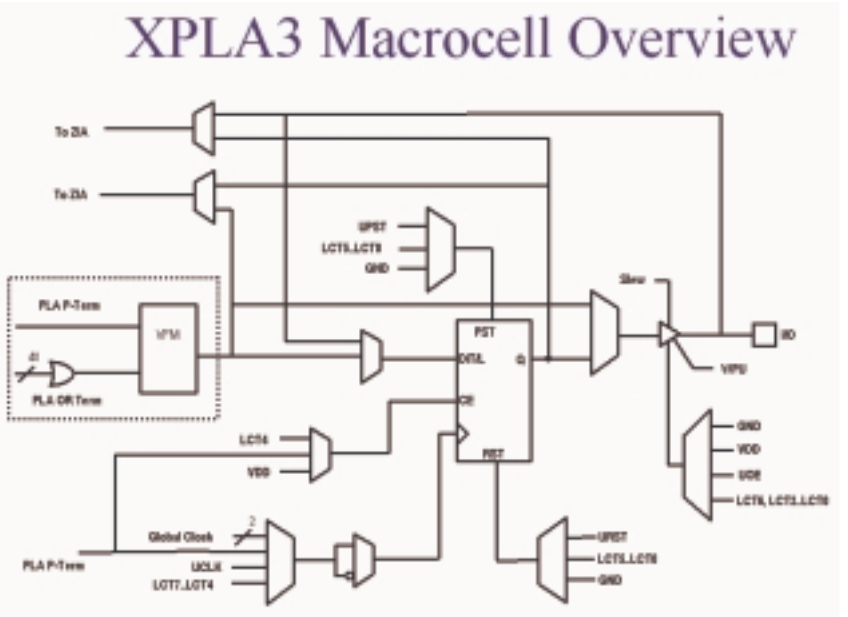


FIGURE 2-25: **COOLRUNNER XPLA3 MACROCELL DIAGRAM**

Each macrocell register can be configured as a D-, T-, or latch-type flip-flop, or combinatorial logic function. Each of these flip-flops can be clocked from any one of eight sources.

Two global synchronous clocks are derived from the four external clock pins. There is one universal clock signal. You can individually configure the clock input signals CT[4:7] (local control terms) as either a PRODUCT term or SUM term equation created from the 36 signals available inside the logic block.

There are two feedback paths to the ZIA: one from the macrocell and one from the I/O pin. Using the I/O pin as an output enables the output buffer, and the macrocell feedback path can be used to feed back the logic implemented in the macrocell.

When an I/O pin is used as an input, the output buffer will be tri-stated and the input signal fed into the ZIA via the I/O feedback path. The logic implemented in the buried macrocell can be fed back to the ZIA via the macrocell feedback path. If the macrocell is configured as an input, a path to the register provides a fast input setup time.

I/O Cell

The output-enable multiplexer has eight possible modes, including a programmable WPU eliminating the need for external termination on unused I/Os. The I/O cell is 5V tolerant and has a single-bit slew-rate control for reducing EMI generation.

Outputs are 3.3V PCI electrical specification compatible (no internal clamp diode).

Simple Timing Model

Figure 2-26 shows the XPLA3 timing model, which has three main timing parameters: T_{PD}, T_{SU}, and T_{CO}.

In other architectures, you may be able to fit the design into the CPLD, but you may not be sure whether system timing requirements can be met until after the design has been fit into the device.

This is because the timing models of other architectures are very complex including such things as timing dependencies on the number of parallel expanders borrowed, sharable expanders, and varying numbers of X and Y routing channels.

In the XPLA3 architecture, you know up-front whether the design will meet system timing requirements. This is because of the simplicity of the timing model.

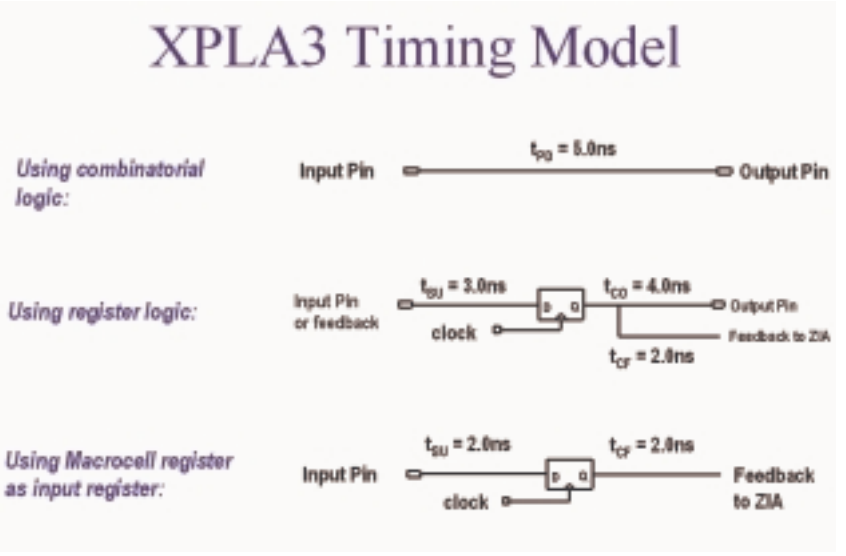


FIGURE 2-26: COOLRUNNER XPLA3 SIMPLE TIMING MODEL

Slew Rate Control

XPLA3 devices have slew rate control for each macrocell output pin. You have the option to enable the slew rate control to reduce EMI. The nominal delay for using this option is 2.0 ns.

XPLA3 Software Tools

Software support for XPLA3 devices is provided by Xilinx WebPOWERED software products, which include the WebFITTER CPLD design fitting tool and WebPACK ISE software. Both tools are free.

In addition, EDIF input is supported for all major third-party software flows, such as Cadence Design Systems, Mentor Graphics, Viewlogic, Exemplar, and Synopsys.

TABLE 2-4: COOLRUNNER SUMMARY OF FEATURES AND BENEFITS

Features	Benefits
Total CMOS architecture with FZP design technology	Lowest stand-by current and total current consumption of any CPLD; therefore longer battery life, increased reliability, and less heat dissipation
32 to 512 macrocell device selections	Suits full range of designs and applications; able to migrate up and down densities if design grows or shrinks
5V tolerant I/Os and multi I/O standards	Simplifies multi-voltage design and level shifting
PLA array	Optimizes sharing and resource utilization (all product terms available)
Bus-friendly I/O	Pull-up resistor for I/O termination
Multiple clocking options	Design flexibility
Fast input registers	Supports direct high-speed interface

TABLE 2-4: COOLRUNNER SUMMARY OF FEATURES AND BENEFITS *(CONTINUED)*

Features	Benefits
VFM (Variable Function MUX) and foldback NANDs	Superior logic optimization and device fitting – fit first time designs and lower costs by being able to use a smaller device
Small, surface mount packages – 0.8 mm and 0.5 mm ball pitch chip scale packages	Smallest footprint and board space savings ideal for handheld devices like PDAs and cellphones
Industrial, commercial, and automotive temperature ranges	Can be used in all application areas from telematics and set-top boxes to medical and harsh environment applications

	XCR3032XL	XCR3064XL	XCR3128XL	XCR3256XL	XCR3384XL	XCR3512XL
Macrocells	32	64	128	256	384	512
Usable Gates	800	1600	3200	6400	9600	12800
t_{PD} (ns)	5	6	6	7.5	7.5	7.5
f_{SYS} (MHz)	200	145	145	140	127	127
Packages (max user I/O)	VQ44 (36) PC44 (36) CS48 (36)	VQ44 (36) PC44 (36) CS48 (48) CP56 (48) VQ100 (68)	VQ100 (84) CS144 (108) TQ144 (108)	TQ144 (120) PQ208 (164) FT256 (164) CS280 (164)	TQ144 (118) PQ208 (172) FT256 (212) FG324 (220)	PQ208 (180) FT256 (212) FG324 (268)

FIGURE 2-27: COOLRUNNER FAMILY OVERVIEW

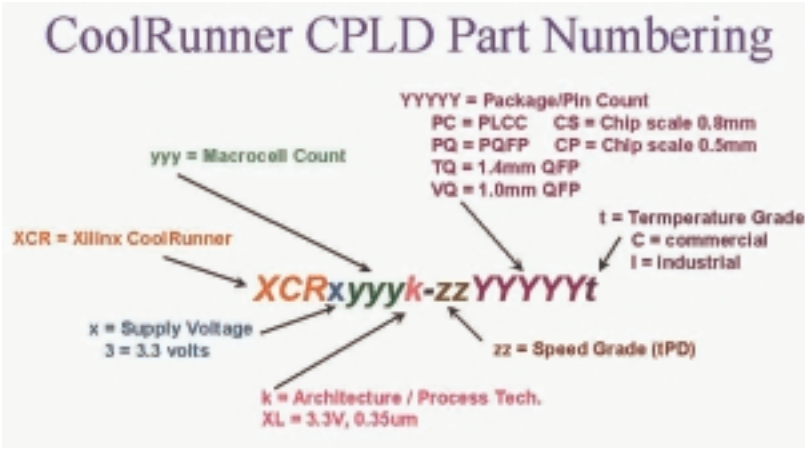


FIGURE 2-28: COOLRUNNER XPLA3 PART NUMBER SYSTEM

COOLRUNNER-II CPLDS

Xilinx CoolRunner-II CPLDs deliver the high speed and ease of use associated with the XC9500/XL/XV CPLD family and the extremely low power versatility of the XPLA3.

This means that the exact same parts can be used for high-speed data communications, computing systems, and leading-edge portable products, with the added benefit of ISP.

Low power consumption and high-speed operation are combined into a single family that is easy to use and cost effective. Xilinx-patented Fast Zero Power architecture inherently delivers extremely low power performance without the need for special design measures.

Clocking techniques and other power-saving features extend your power budget. These design features are supported from Xilinx ISE 4.1i, the WebFITTER CPLD design tool, and WebPACK ISE software onwards.

Figure 2-29 shows the CoolRunner-II CPLD package offering with corresponding I/O count. All packages are surface mount, with more than half of them ball-grid technologies. The ultra-tiny packages permit maximum functional capacity in the smallest possible area.

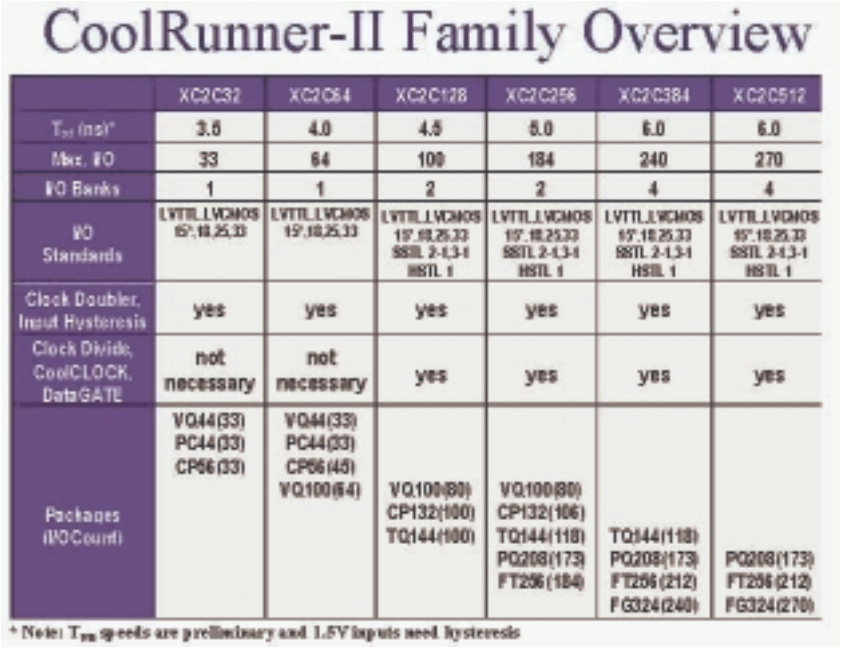
The CMOS technology used in CoolRunner-II CPLDs generates minimal heat, allowing the use of tiny packages during high-speed operation.

At least two densities are present in each package, with three in the VQ100 (100-pin, 1.0 mm QFP) and TQ144 (144-pin, 1.4 mm QFP), and in the FT256 (256-ball, 1.0 mm-spacing FLBGA).

The FT256 is particularly suited for slim-dimensioned portable products with mid- to high-density logic requirements.

Figure 2-29 also details the distribution of advanced features across the CoolRunner-II CPLD family. The family has uniform basic features, with advanced features included in densities where they are most useful. For example, it is unlikely that you would need four I/O banks on 32- and 64-macrocell parts, but very likely for 384- and 512-macrocell parts.

The I/O banks are groupings of I/O pins using any one of a subset of compatible voltage standards that share the same V CCIO level. The clock division capability is less efficient on small parts, but more useful and likely to be used on larger ones. DataGATE™ technology, an ability to block and latch inputs to save power, is valuable in larger parts, but brings marginal benefit to small parts.



CoolRunner-II Family Overview

	XC2C32	XC2C64	XC2C128	XC2C256	XC2C384	XC2C512
T_{pd} (ns)*	3.5	4.0	4.5	5.0	6.0	6.0
Max. I/O	33	64	100	184	240	270
I/O Banks	1	1	2	2	4	4
I/O Standards	LVTTL, LVCMOS 15*, 18, 25, 33	LVTTL, LVCMOS 15*, 18, 25, 33	LVTTL, LVCMOS 15*, 18, 25, 33 SSTL 2-I, 3-I HSTL I	LVTTL, LVCMOS 15*, 18, 25, 33 SSTL 2-I, 3-I HSTL I	LVTTL, LVCMOS 15*, 18, 25, 33 SSTL 2-I, 3-I HSTL I	LVTTL, LVCMOS 15*, 18, 25, 33 SSTL 2-I, 3-I HSTL I
Clock Doubler, Input Hysteresis	yes	yes	yes	yes	yes	yes
Clock Divide, CoolCLOCK, DataGATE	not necessary	not necessary	yes	yes	yes	yes
Packages (I/O Count)	VQ44(33) PC44(33) CP56(33)	VQ44(33) PC44(33) CP56(45) VQ100(64)	VQ100(80) CP132(100) TQ144(100)	VQ100(80) CP132(106) TQ144(118) PQ208(173) FT256(184)	TQ144(118) PQ208(173) FT256(212) FG324(240)	PQ208(173) FT256(212) FG324(270)

* Note: T_{pd} speeds are preliminary and 1.5V inputs need hysteresis

FIGURE 2-29: COOLRUNNER-II FAMILY OVERVIEW

CoolRunner-II Architecture Description

The CoolRunner-II CPLD is a highly uniform family of fast, low-power devices. The underlying architecture is a traditional CPLD architecture, combining macrocells into function blocks interconnected with a global routing matrix, the Xilinx Advanced Interconnect Matrix (AIM).

The function blocks use a PLA configuration that allows all product terms to be routed and shared among any of the macrocells of the function block.

Design software can efficiently synthesize and optimize logic that is subsequently fit to the function blocks and connected with the ability to utilize a very high percentage of device resources.

The software easily and automatically manages design changes, exploiting the 100% routeability of the PLA within each function block. This extremely robust building block delivers the industry's highest pin-out retention under very broad design conditions.

The design software automatically manages device resources so that you can express your designs using completely generic constructs, without needing to know the architectural details. If you're more experienced, you can take advantage of these details to more thoroughly understand the software's choices and direct its results.

Figure 2-30 shows the high-level architecture whereby function blocks attach to pins and interconnect to each other within the internal interconnect matrix. Each function block contains 16 macrocells.

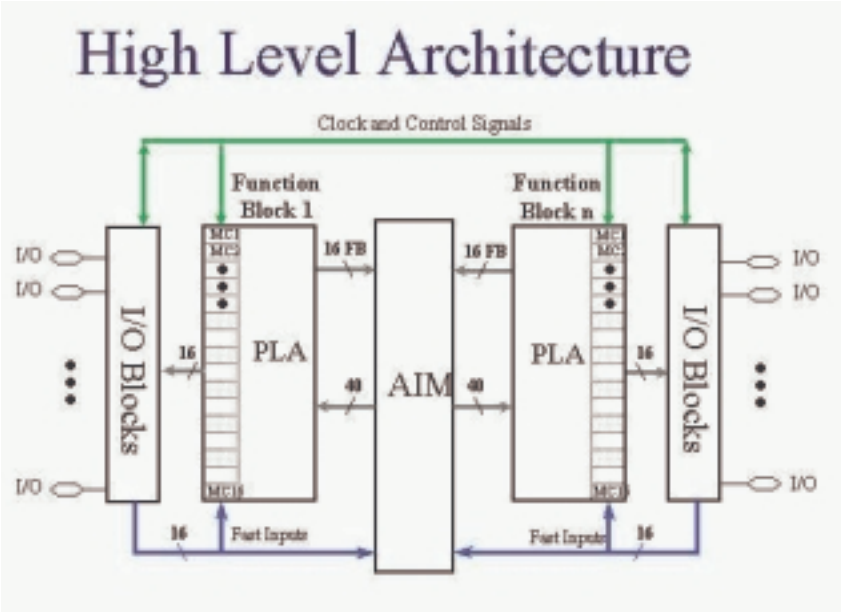


FIGURE 2-30: COOLRUNNER-II HIGH-LEVEL ARCHITECTURE

CoolRunner-II Function Block

The CoolRunner-II CPLD function blocks contain 16 macrocells, with 40 entry sites for signals to arrive for logic creation and connection.

The internal logic engine is a 56-product term PLA. All function blocks, regardless of the number contained in the device, are identical.

At the high level, the p-terms reside in a PLA. This structure is extremely flexible and very robust when compared to fixed or cascaded p-term function blocks.

Classic CPLDs typically have a few p-terms available for a high-speed path to a given macrocell. They rely on capturing unused p-terms from neighboring macrocells to expand their product term tally when needed.

The result of this architecture is a variable timing model and the possibility of stranding unusable logic within the function block.

The PLA is different – and better. First, any p-term can be attached to any OR gate inside the function block macrocell(s).

Second, any logic function can have as many p-terms as needed attached to it within the function block, to an upper limit of 56.

Third, you can reuse product terms at multiple macrocell OR functions so that within a function block, you need only create a particular logical product once, but you can reuse it as many as 16 times within the function block. Naturally, this works well with the fitting software, which identifies product terms that can be shared.

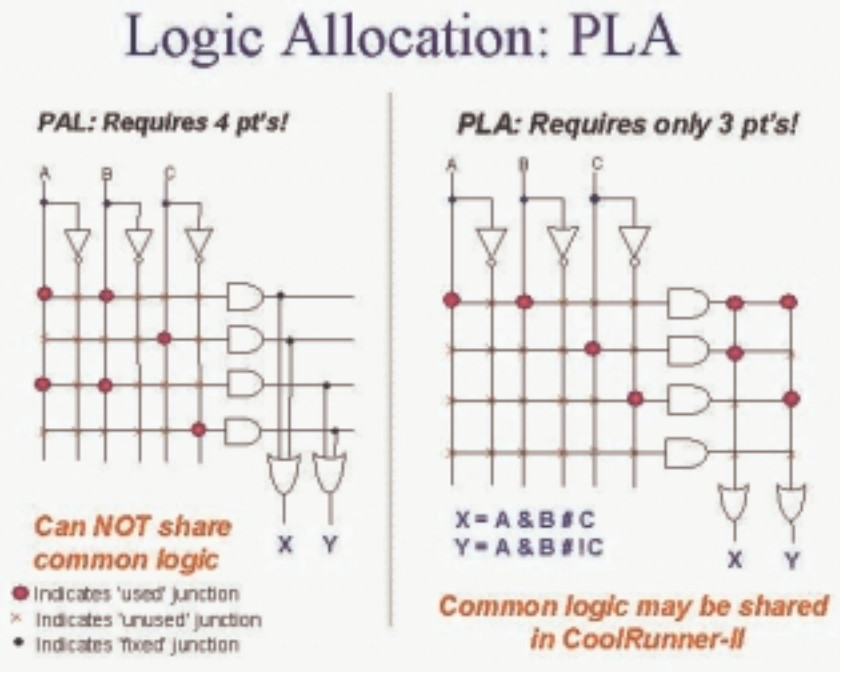


FIGURE 2-31: LOGIC ALLOCATION – TYPICAL PAL VS. PLA

The software places as many functions as it can into function blocks. There is no need to force macrocell functions to be adjacent or have any other restriction except for residing in the same function block, which is handled by the software.

Functions need not share a common clock, common set/reset, or common output enable to take full advantage of the PLA. In addition, every p-term arrives with the same time delay incurred. There are no cascade time adders for putting more product terms in the function block.

When the function block p-term budget is reached, a small interconnect timing penalty routes signals to another function block to continue creating logic. Xilinx design software handles all this automatically.

CoolRunner-II Macrocell

The CoolRunner-II CPLD macrocell is extremely efficient and streamlined for logic creation. You can develop SOP logic expressions comprising as many as 40 inputs and span 56 product terms within a single function block.

The macrocell can further combine the SOP expression into an XOR gate with another single p-term expression.

The resulting logic expression's polarity is also selectable. The logic function can be pure combinatorial or registered, with the storage element operating selectively as a D or T flip-flop, or transparent latch.

Available at each macrocell are independent selections of global, function-block level, or local p-term-derived clocks, sets, resets, and output enables. Each macrocell flip-flop is configurable for either single edge or DualEDGE clocking, providing either double data rate capability or the ability to distribute a slower clock (thereby saving power).

For single-edge clocking or latching, either clock polarity may be selected per macrocell.

CoolRunner-II macrocell details are shown in Figure 2-32. Standard logic symbols are used in the in figure, except the trapezoidal multiplexers have input selection from statically programmed configuration select lines (not shown).

Xilinx application note XAPP376 gives a detailed explanation of how logic is created in the CoolRunner-II CPLD family.

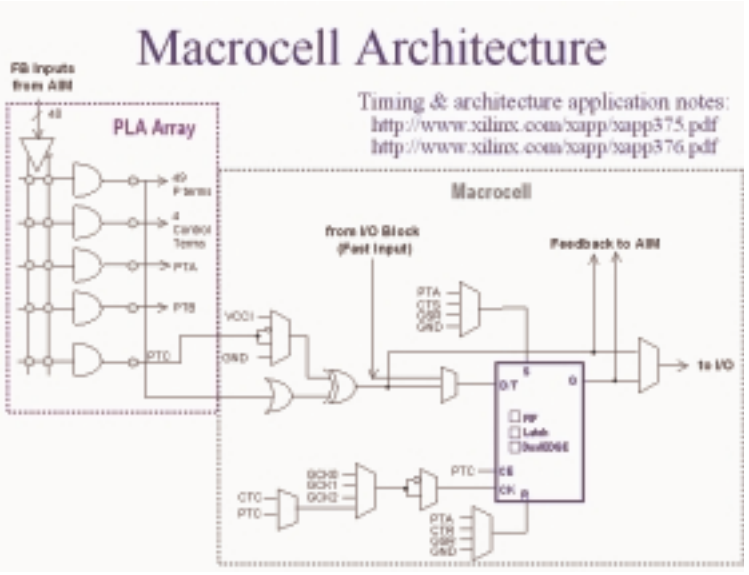


FIGURE 2-32: **COOLRUNNER-II MACROCELL**

When configured as a D-type flip-flop, each macrocell has an optional clock enable signal permitting state hold while a clock runs freely.

Note that control terms are available to be shared for key functions within the function block, and are generally used whenever the exact same logic function would be repeatedly created at multiple macrocells.

The control term product terms are available for function block clocking (CTC), function block asynchronous set (CTS), function block asynchronous reset (CTR), and function block output enable (CTE).

You can configure any macrocell flip-flop as an input register or latch, which takes in the signal from the macrocell's I/O pin and directly drives the AIM. The macrocell combinatorial functionality is retained for use as a buried logic node if needed.

Advanced Interconnect Matrix (AIM)

AIM is a highly connected low-power rapid switch directed by the software to deliver a set of as many as 40 signals to each function block for the creation of logic.

Results from all function block macrocells, as well as all pin inputs, circulate back through the AIM for additional connection available to all other function blocks, as dictated by the design software. The AIM minimizes both propagation delay and power as it makes attachments to the various function blocks.

I/O Blocks

I/O blocks are primarily transceivers. However, each I/O is either automatically compliant with standard voltage ranges or can be programmed to become compliant.

In addition to voltage levels, each input can selectively arrive through Schmitt-trigger inputs. This adds a small time delay, but substantially reduces noise on that input pin.

Hysteresis also allows easy generation of external clock circuits. The Schmitt-trigger path is best illustrated in Figure 2-33. Outputs can be directly driven, tri-stated, or open-drain configured. A choice of slow or fast slew rate output signal is also available.

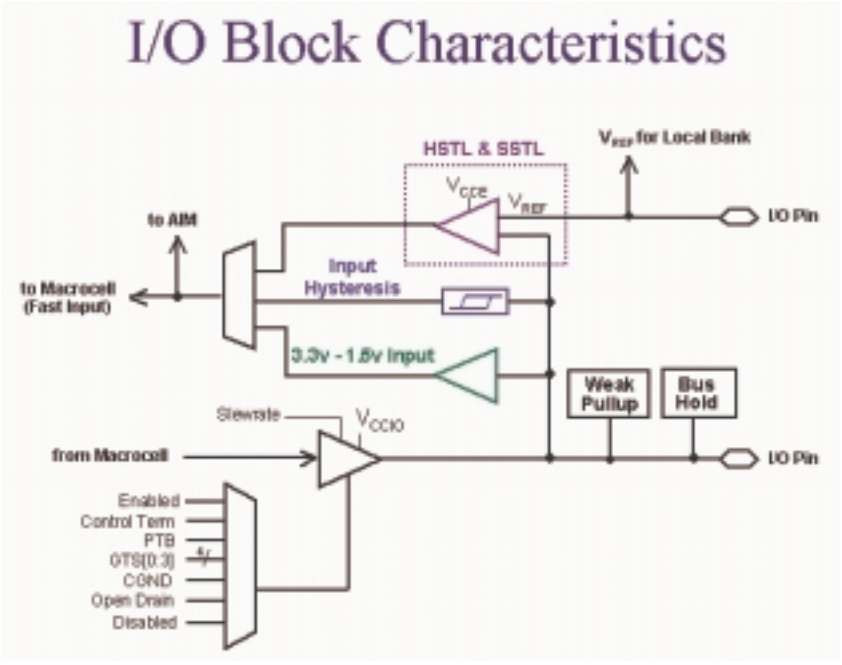


FIGURE 2-33: **COOLRUNNER-II I/O BLOCK**

Output Banking

CPLDs are widely used as voltage interface translators; thus, the output pins are grouped in large banks.

The smallest parts are not banked, so all signals will have the same output swing for 32- and 64-macrocell parts.

The medium parts (128 and 256 macrocell) support two output banks. With two, the outputs will switch to one of two selected output voltage levels, unless both banks are set to the same voltage.

The larger parts (384 and 512 macrocell) support four output banks, split evenly. They can support groupings of one, two, three, or four separate output voltage levels. This kind of flexibility permits easy interfacing to 3.3V, 2.5V, 1.8V, and 1.5V in a single part.

DataGATE

Low power is the hallmark of CMOS technology. Other CPLD families use a sense amplifier approach to create p-terms, which always has a residual current component.

This residual current can be several hundred milliamps, making these CPLDs unusable in portable systems.

CoolRunner-II CPLDs use standard CMOS methods to create the CPLD architecture and deliver the corresponding low current consumption, without any special tricks.

However, sometimes you might want to reduce the system current even more by selectively disabling unused circuitry. The patented DataGATE technology permits a straightforward approach to additional power reduction.

Each I/O pin has a series switch that can block the arrival of unused free-running signals that may increase power consumption. Disabling these switches enables you to complete your design and choose which sections will participate in the DataGATE function.

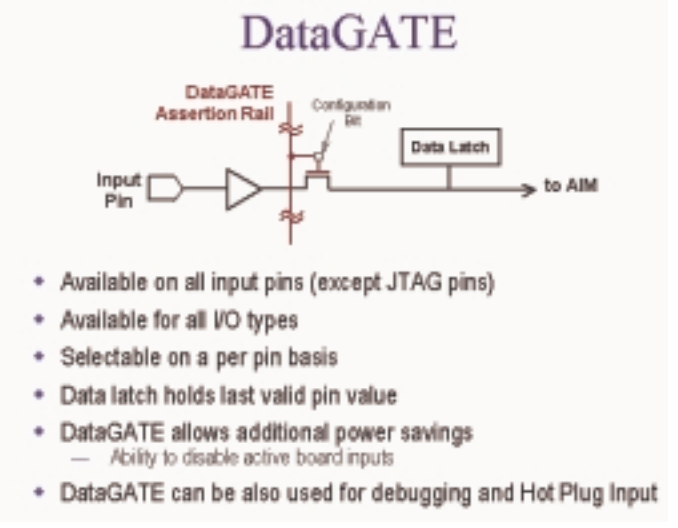


FIGURE 2-34: DATAGATE FUNCTION IN COOLRUNNER-II CPLDS

The DataGATE logic function drives an assertion rail threaded through medium- and high-density CoolRunner-II CPLD parts.

You can select which inputs to block under the control of the DataGATE function, effectively blocking controlled switching signals so that they do not drive internal chip capacitances.

Output signals that do not switch are held by the bus hold feature. You can choose any set of input pins can be chosen to participate in the DataGATE function.

Figure 2-34 shows how DataGATE function works. One I/O pin drives the DataGATE assertion rail. It can have any desired logic function on it – something as simple as mapping an input pin to the DataGATE function or as complex as a counter or state machine output driving the DataGATE I/O pin through a macrocell.

When the DataGATE rail is asserted low, any pass transistor switch attached to it is blocked. Each pin has the ability to attach to the AIM through a DataGATE pass transistor, and be blocked.

A latch automatically captures the state of the pin when it becomes blocked. The DataGATE assertion rail threads throughout all possible I/Os, so each can participate if chosen.

One macrocell is singled out to drive the rail, and that macrocell is exposed to the outside world (through a pin) for inspection. If the DataGATE function is not needed, this pin is an ordinary I/O.

Additional Clock Options: Division, DualEDGE, and CoolCLOCK

Division

Circuitry has been included in the CoolRunner-II CPLD architecture to divide one externally supplied global clock by standard values, with options for division by 2, 4, 6, 8, 10, 12, 14, and 16 (see Figure 2-35). This capability is supplied on the GCK2 pin.

The resulting clock produced will be 50% duty cycle for all possible divisions.

Note that a synchronous reset is included to guarantee that no runt clocks can get through to the global clock nets. The signal is buffered and driven to multiple traces with minimal loading and skew.

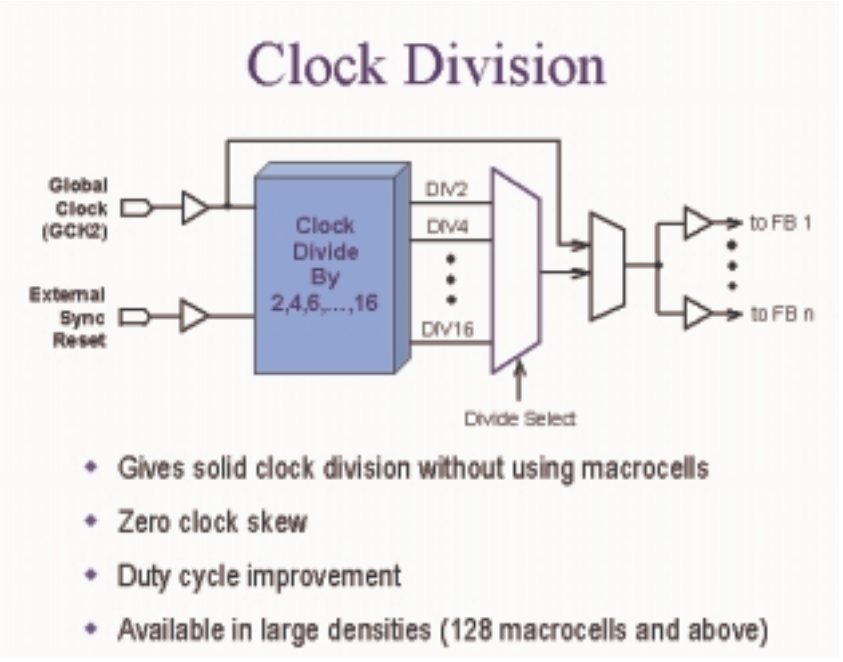


FIGURE 2-35: COOLRUNNER-II CLOCK DIVISION

DualEDGE

Each macrocell has the ability to double its input clock switching frequency. Figure 2-32 shows the macrocell flip-flop with the DualEDGE option (doubled clock) at each macrocell.

The source to double can be a control term clock, a product term clock, or one of the available global clocks. The ability to switch on both clock edges is vital for a number of synchronous memory interface applications as well as certain double data rate I/O applications.

CoolCLOCK

In addition to the DualEDGE flip-flop, you can gain additional power savings by combining the clock division circuitry with the DualEDGE circuitry. This capability is called CoolCLOCK and is designed to reduce clocking power within the CPLD.

Because the clock net can be a significant power drain, you can reduce the clock power by driving the net at half frequency, and then doubling the clock rate using DualEDGE triggering at the macrocells.

Figure 2-36 illustrates how CoolCLOCK is created by internal clock cascading, with the divider and DualEDGE flip-flop working together.

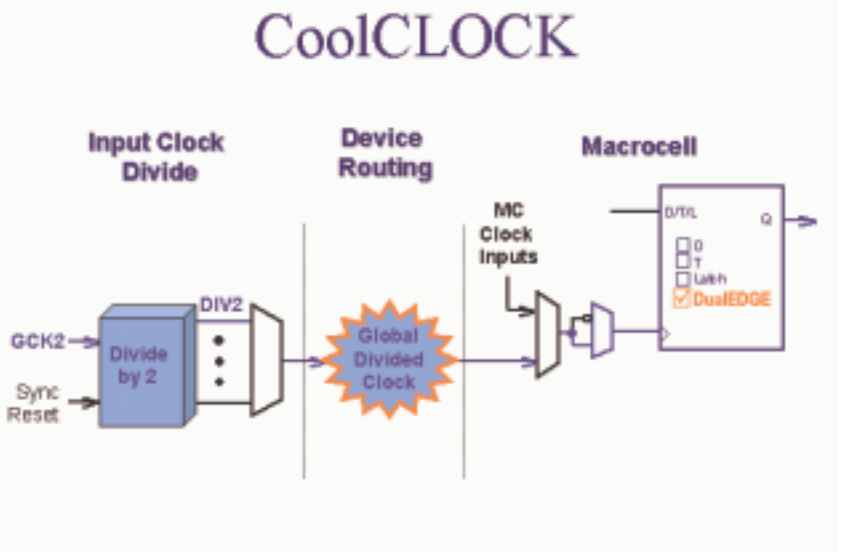


FIGURE 2-36: **COOLCLOCK**

Design Security

You can secure your designs during programming to prevent either accidental overwriting or pattern theft via readback.

CoolRunner-II CPLDs have four independent levels of security provided on-chip, eliminating any electrical or visual detection of configuration patterns.

These security bits can be reset only by erasing the entire device. Additional details are omitted intentionally.

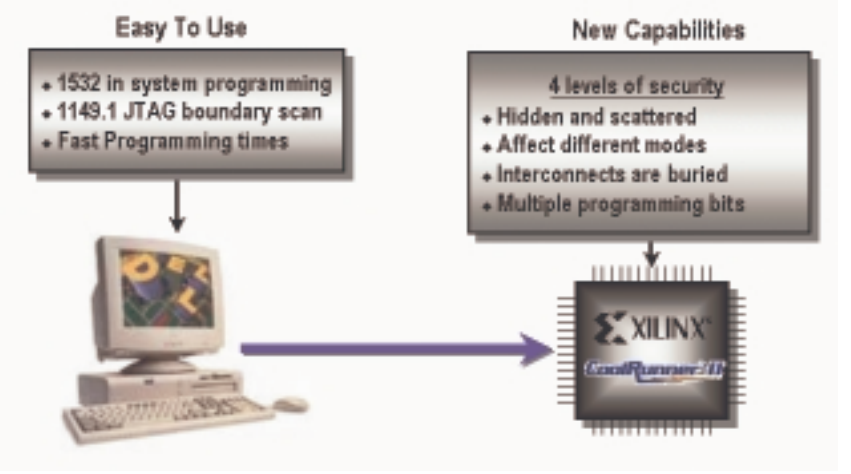


FIGURE 2-37: **COOLRUNNER-II DEVICE SECURITY**

CoolRunner-II Application Examples

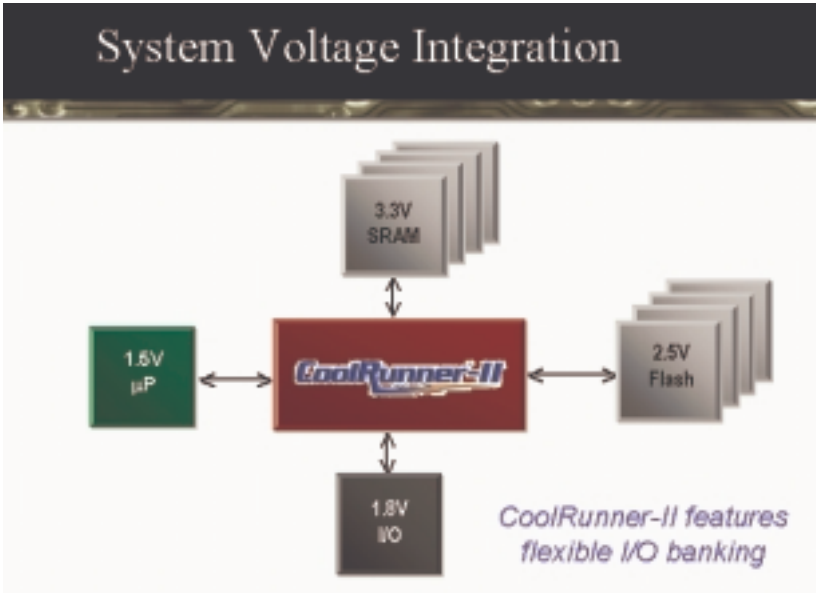


FIGURE 2-38: **SYSTEM VOLTAFE INTEGRATION EXAMPLE**

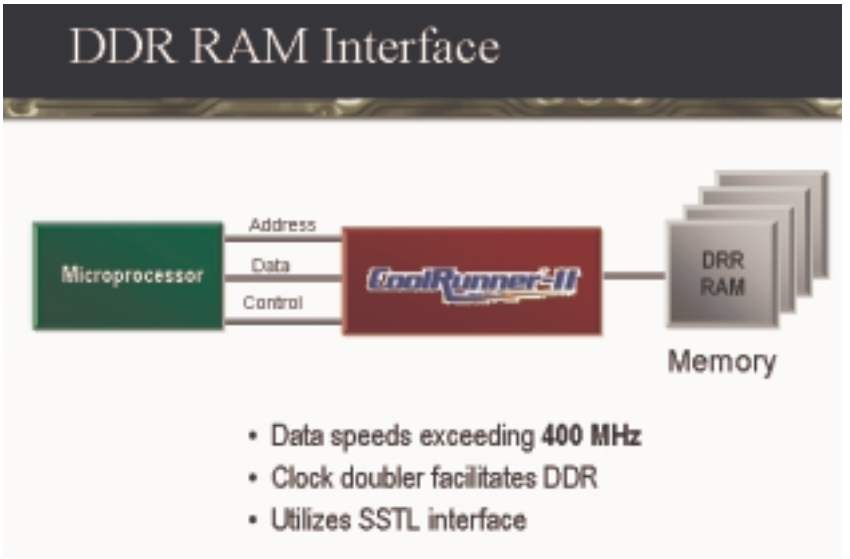


FIGURE 2-39: **DDR RAM INTERFACE EXAMPLE**

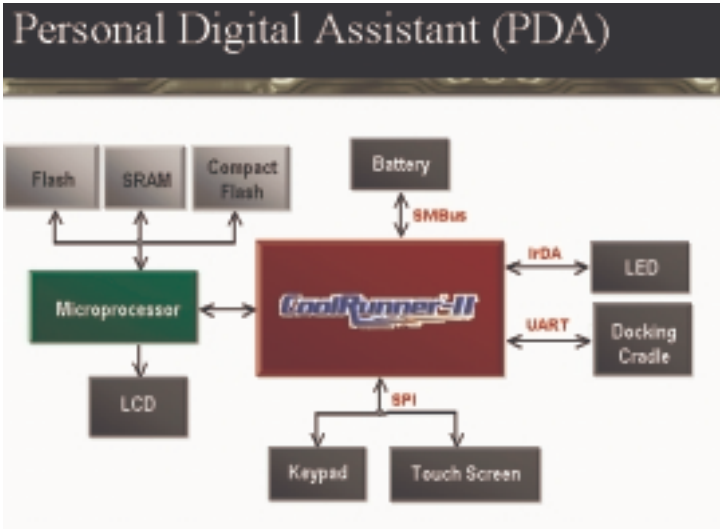


FIGURE 2-40: **PDA USING FREE REFERENCE DESIGNS EXAMPLE**

COOLRUNNER REFERENCE DESIGNS

CoolRunner reference designs are HDL code-based designs that can help reduce the time of CPLD designs. They are available free of charge.

These reference designs take the form of not-for-pay IP, which can be used as is. Unlike purchased IP, these reference designs do not come with direct support.

They are built around application notes and have been tested in WebPACK software. They are fully functional through the WebPACK simulator and testbench.

You can find CoolRunner reference designs in the Xilinx IP Center (http://www.xilinx.com/ipcenter/) by searching on the keyword "CoolRunner."

Accessing the Reference Designs

From the Xilinx.com website, select "Products and Services" from the top bar and then select "Silicon Solutions." Select "CoolRunner-II CPLDs" from the left-hand side bar and then select "CoolRunner Reference Designs."

This will bring up a page that details all of the reference designs available for selection. Click on the application note that of interest and download the PDF file. Open the application and go to the last page of the document.

FIGURE 2-41: APPLICATION NOTE REFERENCE DESIGN LINK

Click on the link and the following page will appear:

FIGURE 2-42: REFERNECE DESIGN DOWNLOAD INSTRUCTIONS

FIGURE 2-43: "REGISTER ME" LINK

Click on the "Register Me" button to gain access to the free HDL code.

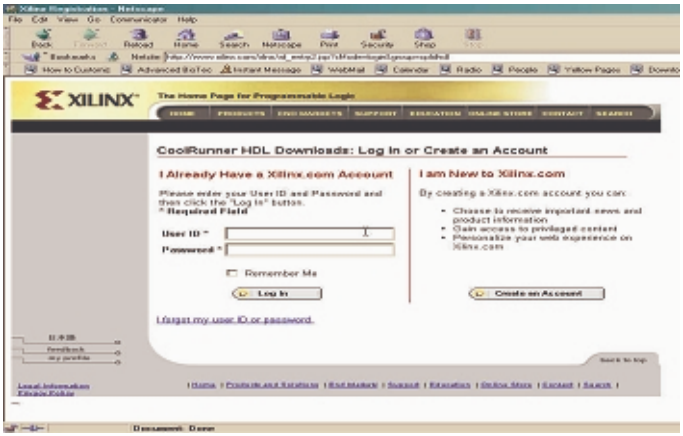


FIGURE 2-44: LOG IN PAGE

The page above will appear. If you are already a Xilinx registered user, please enter your user name and password.

If you are new to the Xilinx web site, you will need to create an account. Once successful, you will see a page saying "You will receive an e-mail with links to the downloadable files."

You will then be sent an e-mail listing all of the HDL files, with links to each file. You can then download and start to use any of the HDL design examples in conjunction with your WebPACK software.

Table 2-5 details the current reference designs. These are continually updated, so check regularly for new listings.

TABLE 2-5: CURRENT REFERENCE DESIGNS

Application	Reference Design	Reference Number	Language	Macrocell	Target Device	% Utilized
PDA	XPATH Module Design	XAPP395	VHDL	225	XC2C384	58
	Springboard Module Design	XAPP147	Pocket C, VHDL	67	XC2C128	52
	8 Channel DVM Springboard	XAPP148	Pocket C, VHDL	184	XC2C256	71
Datacom	SECDED	XAPP383	VHDL	66	XC2C128	52
	N x N Crosspoint Switch	XAPP380	VHDL	193	XC2C256	76
	IrDA and UART	XAPP345	VHDL or Verilog	87	XC2C128	67
	UARTs	XAPP341	VHDL or Verilog	61	XC2C128	47
	16b/20b Encoder/Decoder	XAPP336	VHDL	76	XC2C128	59
Bus Interface	SPI	XAPP386	VHDL	128	XC2C256	50
	Compact Flash Interface	XAPP00X	VHDL		XC2C128	
	I²C Bus Controller	XAPP333	VHDL or Verilog	131	XC2C256	51
	SMBus Controller	XAPP353	VHDL	158	XC2C256	61
	Manchester Encoder/Decoder	XAPP339	VHDL or Verilog	55	XC2C64	86
Memory	NAND Interface	XAPP354	VHDL or Verilog	9	XC2C32	28
	Interface to DDR SDRAM	XAPP384	VHDL		XC2C256	
Wireless	Wireless Transceiver	XAPP358	VHDL	156	XC2C256	60
Multimedia	MP3 Player	XAPP328	VHDL	219	XC2C256	86
Microcontroller	8-bit Microcontroller	XAPP387	VHDL & C	107	XC2C128	84
	8-bit Microcontroller	XAPP387	VHDL & C	212	XC2C256	83
	8051 Microcontroller Interface	XAPP349	VHDL	57	XC2C64	89

Free VHDL design code: www.xilinx.com/products/xaw/coolvhdlq.htm

Military and Aerospace

Xilinx is the leading supplier of high-reliability PLDs to the aerospace and defense markets. These devices are used in a wide range of applications such as electronic warfare, missile guidance and targeting, RADAR, SONAR communications, signal processing, avionics, and satellites.

The Xilinx QPro™ family of ceramic and plastic QML products provides you with advanced programmable logic solutions for next-generation designs. The QPro family also includes select products that are radiation hardened for use in satellite and other space applications.

Our quality management system is fully compliant with all ISO9001 requirements. In 1997, Xilinx became fully qualified as a QML supplier by meeting all of the requirements for MIL Standard 38535.

Automotive and Industrial

XILINX IQ SOLUTIONS – ARCHITECTING AUTOMOTIVE INTELLIGENCE

In-car electronic content is increasing at a phenomenal rate. It includes such applications as navigation systems, entertainment systems, instrument clusters, advanced driver information systems, and communications devices.

To address the needs of automotive electronics designers, Xilinx has created a new family of devices with an extended industrial temperature range option.

This new "IQ" family consists of existing Xilinx industrial grade (I) FPGAs and CPLDs, with the addition of a new extended temperature grade (Q) for selected devices.

The new IQ product grade (-40°C to +125°C ambient for CPLDs and junction for FPGAs) is ideal for automotive and industrial applications. The wide range of device density and package combinations enables you to deliver high-performance, cost-effective, flexible solutions that meet your application needs.

Design-In Flexibility

With Xilinx IQ devices, you can design-in flexibility and get your product to market faster than ever before.

Because many new standards continue to evolve (such as the LIN, MOST, and FlexRay in-car busing standards), you need the flexibility to quickly modify your designs at any time.

With our unique Internet Reconfigurable Logic (IRL) capability, you can remotely and automatically modify your designs, in the field, after your product has left the factory.

By combining our latest IQ PLDs with our solutions infrastructure of high-productivity software, IP cores, design services, and customer education, you can develop advanced, highly flexible products faster than ever before.

For more information, visit *www.xilinx.com/automotive*.

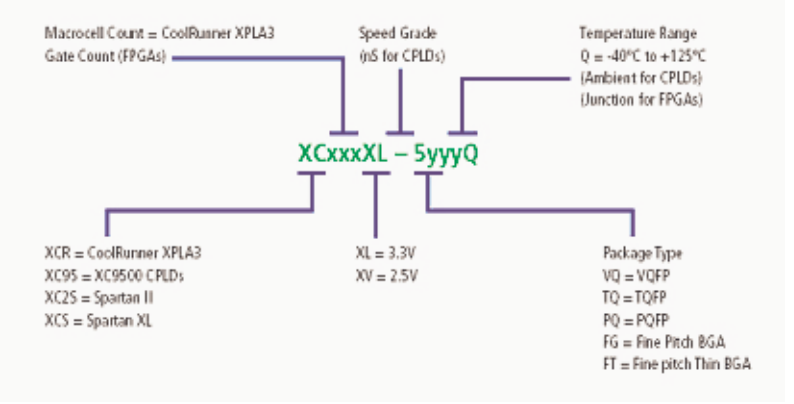


FIGURE 2-45: **IQ DEVICES ORDERING INFORMATION**

TABLE 2-6: IQ TEMPERATURE RANGE

Product Group	Temperature Grade/Range (°C)		
	C	I	Q
FPGA	Tj = 0 to +85	Tj = -40 to +100	Tj = -40 to +125
CPLD	Ta = 0 to +70	Ta = -40 to +85	Ta = -40 to +125

TABLE 2-7: AVAILABLE IQ DEVICES IN EXTENDED TEMPERATURE

IQ Device Family	Densities
Spartan XL (3.3V)	15k gates to 40k gates
XC9500XL (3.3V)	36 and 72 macrocells
CoolRunner XPLA3 (3.3V)	32 to 512 macrocells
Spartan-II (2.5V)	15k gates to 200k gates
CoolRunner-II (1.8V)	32 to 512 macrocells
Spartan-IIE (1.8V)	50k gates to 600k gates
XC9500 (5V)	36 and 72 macrocells

Design Tools

Programmable logic design has entered an era in which device densities are measured in the millions of gates, and system performance is measured in hundreds of megahertz. Given these system complexities, the critical success factor in the creation of a design is your productivity.

Xilinx offers complete electronic design tools that enable the implementation of designs in Xilinx PLDs. These development solutions combine powerful technology with a flexible, easy-to-use graphical interface to help you achieve the best possible designs within your project schedule – regardless of your experience level.

The "Design Tools Center" web pages cover both the Xilinx ISE tools suite as well as design tools from our software partners. It is arranged by the following topics:

DESIGN ENTRY

ISE software greatly improves your time to market and productivity by accelerating the design entry process.

ISE tools support today's most popular methods for design capture, including HDL and schematic entry, integration of IP cores, and robust support for reuse of your own IP.

This rich mixture of design entry capabilities provides the easiest-to-use design environment available today for all logic design.

SYNTHESIS

ISE advanced HDL synthesis engines produce optimized results for PLD synthesis, one of the most essential steps in your design methodology. It takes your conceptual HDL design definition and generates a logical or physical representation for the targeted silicon device.

A state-of-the-art synthesis engine is required to produce highly optimized results with a fast compile and turnaround time.

To meet this requirement, the synthesis engine must be tightly integrated with the physical implementation tool and proactively meet design timing requirements by driving the placement in the physical device.

In addition, cross probing between the physical design report and the HDL design code further enhances the turnaround time.

Xilinx ISE software provides seamless integration with leading synthesis engines from Mentor Graphics, Exemplar, Synopsys, and Synplicity.

The ISE product also includes Xilinx proprietary synthesis technology, or XST. With just the push of a button, you can start any leading synthesis engine within ISE. You can even use multiple synthesis engines to obtain the most optimized result of your programmable logic design.

IMPLEMENTATION AND CONFIGURATION

Programmable logic design implementation assigns the logic created during design entry and synthesis into specific physical resources.

The term "place and route" has historically been used to describe the implementation process for FPGA devices, while "fitting" has been used for CPLDs.

Implementation is followed by device configuration, where a bitstream is generated from the physical place and route information and downloaded into the target PLD.

To ensure that you get your product to market quickly, Xilinx ISE software provides several key technologies required for design implementation, including:

- Ultra-fast runtimes enable multiple "turns" per day
- ProActive Timing Closure drives high-performance results
- Timing-driven place and route combined with push-button ease.

BOARD-LEVEL INTEGRATION

ISE software provides intensive support to help you ensure your programmable logic design works within the context of the entire system.

Xilinx understands critical issues such as complex board layout, signal integrity, high-speed bus interface, high-performance I/O bandwidth, and electromagnetic interference for system-level designers.

To ease the these challenges, ISE software provides support to all Xilinx leading FPGA technologies:

- XCITE
- Digital clock management for system timing
- EMI control management for electromagnetic interference
- Complete pin configurations
- Packaging information for board-level integration
- ISE board-level verification
- IBIS models
- Stamp models
- LMG models
- ChipScope™ ILA.

VERIFICATION TECHNOLOGIES

ISE software includes verification support at all stages of your design, from design entry to board-level integration.

Static Verification

Static verification tools allow you to verify your design without requiring the creation of lengthy test vectors. Verification can be exhaustive or selective, allowing you to rapidly detect implementation errors in advance.

Static verification tools also deliver extensive diagnosis and debug capabilities. The following static verification tools are supported:

- Constraints Editor
- Delay Calculator
- Trace
- Timing Analyzer
- Prime Time
- XPower
- Formality
- Conformal™ LEC
- DRC
- Chip Viewer.

Dynamic Verification

You can save time by using dynamic verification to intercept logical or HDL-based errors early in the design cycle. By exposing a design to realistic and extensive stimuli, you can find many functional problems at this stage.

The following dynamic verification tools are supported:

- HDL Bencher™
- ModelSim XE
- StateBench
- HDL Simulation Libraries.

Debug Verification

Debug verification tools speed up the process of viewing, identifying, and correcting design problems at different stages of the design cycle.

Debug verification includes the ability to view, "live," all internal signals and nodes within an FPGA. These tools can also assist in HDL-based designs by checking coding style for optimum performance.

The following debug verification tools are supported:

- LEDA
- FPGA Editor Probe
- ChipScope ILA
- ChipScope Pro.

Board-Level Verification

Using board-level verification tools ensures that your design performs as intended once integrated with the rest of the system.

The Xilinx ISE environment supports the following board-level verification tools:

- IBIS Models
- Tau
- BLAST
- Stamp Models
- Impact.

ADVANCED DESIGN TECHNIQUES

As your FPGA requirements grow, your design problems can change. High-density design environments mean multiple teams working through distributed nodes on the same project, across the aisle or in different parts of the world.

ISE software's advanced design options are targeted at making high-density designs as easy to realize as the smallest glue logic.

Floorplanner – The Xilinx high-level floorplanner is a graphic planning tool that lets you map your design onto the target chip. Floorplanning can efficiently drive your high-density design process.

Modular Design – This gives you the ability to partition a large design into individual modules. Each module can then be floorplanned, designed, implemented, and locked until the remaining modules are finished.

Partial Reconfigurability – Useful for applications requiring the loading of different designs into the same area of the device, partial reconfiguration allows you to flexibly change portions of a design without having to reset or completely reconfigure the entire device.

Internet Team Design – This allows managers to drive each team and its design module from a standard Internet browser using the corporate intranet structure.

High-Level Languages – As design densities increase, the need for a higher level of abstraction becomes more important. Xilinx is driving and supporting industry standards and their respective support tools.

EMBEDDED SW DESIGN TOOLS CENTER

Embedded Software Tools for Virtex-II Pro Platform FPGAs

The term "embedded software tools" most often applies to the tools required to create, edit, compile, link, load, and debug high-level language code, usually C or C++, for execution on a processor engine.

With the Virtex-II Pro Platform FPGA, you will be able to target design modules for either silicon hardware (FPGA gates) or as software applications, running on process engines like the embedded PowerPC hard core.

When it comes to embedded software development, Xilinx offers multiple levels of support. Xilinx supports the Virtex-II Pro Platform FPGA embedded processors with "Xilinx versions" of established tools for both low-cost and high-performance markets.

For hardware-centric engineers who want to move design modules into software running on the Virtex-II Pro Platform FPGA PowerPC core, Xilinx provides a simple and low-cost solution.

Alternatively, if software-centric engineers want a feature-rich environment in which to develop more complex applications, Xilinx supplies access to specialized best-of-class tools from the embedded industry leader.

This prevents you from having to embrace completely new development methodologies. You will be able to port existing legacy designs more easily to the Virtex-II Pro Platform FPGA.

Xilinx IP Cores

The Xilinx website has a comprehensive database of Xilinx LogiCORE™ and third-party AllianceCORE verified and tested cores. To find them, visit the Xilinx IP Center at *www.xilinx.com/ipcenter*.

The CORE Generator™ tool from Xilinx delivers highly optimized cores compatible with standard design methodologies for Xilinx FPGAs.

This easy-to-use tool generates flexible, high-performance cores with a high degree of predictability. You can also download future core offerings from the Xilinx website.

The CORE Generator tool is provided as part of Xilinx Foundation ISE software.

Web-Based Information Guide

The "Products and Services" and "End Markets" sections on the Xilinx website give you information about where and how Xilinx devices can be used in end applications and markets.

The data ranges from application notes, white papers, and industry information to reference designs and example code.

These pages are updated regularly, making them ideal to bookmark for research purposes or for downloading code or design solutions to shorten your design time to market.

The sections within the "Products and Services" page on the Xilinx website are:

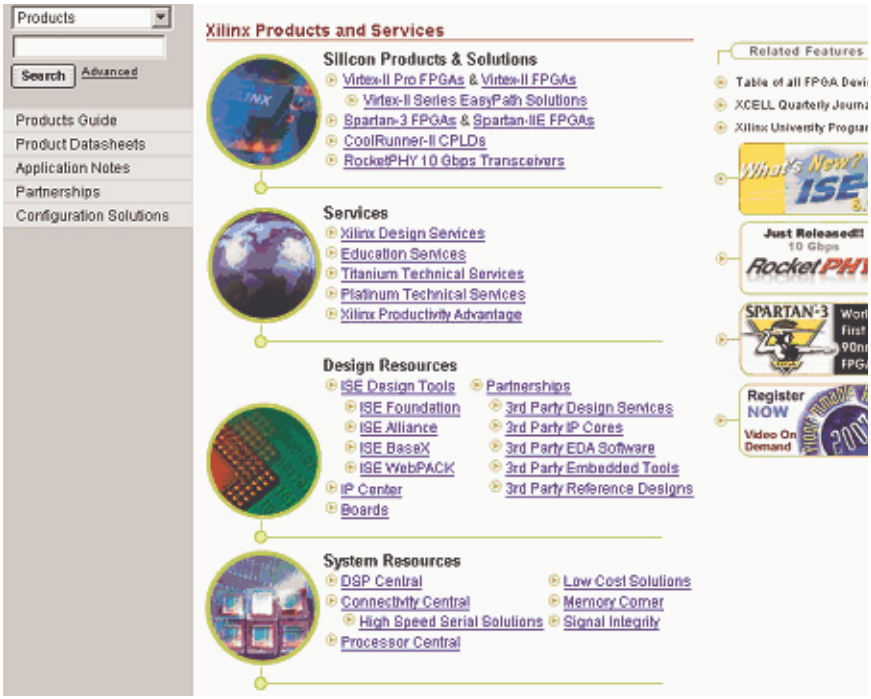


FIGURE 2-46: PRODUCTS AND SERVICES WEBSITE

Let's look at each of these web-based sections.

END MARKETS

The eSP web portal is located within the "End Markets" section on the Xilinx website. It is the industry's first web portal dedicated to providing comprehensive solutions that accelerate product development.

To make it as easy as possible, we've provided a choice for locating material. You can select a specific market solution or a broad-reaching technology from two drop-down menus: "Network Solutions" or "Technologies."

In "Market Solutions," choose from:

- Metro Access Networks
- Automotive
- Broadband
- Industrial
- Medical and Scientific

- Test and Measurement
- Digital Video Technologies (DVT)
- Home Networking
- Wireless Networks.

In the "Technologies" section, choose from:

- Wired Networks
- Consumer
- Additional Topics (such as digital video imaging and capture and editing).

The site was designed to decrease the time spent in the pre-design phase. This phase, which increasingly has become the designers' Achilles' heel, involves visiting seminars, learning new standards, assimilating the data, analyzing market trends, and more.

The eSP web portal saves time by proving up-to-date information about emerging standards and protocols, how and where they are used, impartial information about which one is best for your application, and pre-tested reference designs that can be purchased and used.

The eSP web portal (*www.xilinx.com/esp*) includes:

- White Papers
- System Diagrams
- Ask the Experts
- Glossary of Terms
- System Solutions Boards/reference design boards
- eSP News
- Industry Events
- Tutorials on the latest standards and protocols.

SILICON PRODUCTS AND SOLUTIONS

These web pages allow you to gain in-depth details on Xilinx silicon products, including FPGAs (Virtex and Spartan), CPLDs (CoolRunner and XC9500) and RocketPHY™ transceivers. Data sheets, user guides, FAQs, white papers, and application notes are also available.

DESIGN RESOURCES

This section details the selection of Xilinx and third-party design software. From here, you can also go to the IP Center for details about development boards and platforms.

SYSTEM RESOURCES

DSP Central

These web pages detail the XtremeDSP solutions that deliver the performance and flexibility needed to quickly build complex, high-performance DSP systems.

Driven by the broadband revolution and explosive growth in wireless products, the demand for digital signal processing featuring extreme performance and great flexibility is growing faster than what conventional DSP can deliver.

The rapid convergence of different technology segments – 3G and 4G wireless communication systems, high-bandwidth networking, real-time video broadcasting, and high-performance computing systems – is producing what analysts call "The beginning of a new information technology era."

Xilinx, the recognized leader in programmable logic solutions, is well established in these technology segments and uniquely positioned to address this new DSP paradigm now.

The XtremeDSP solution can give you computing capabilities approaching 1 Tera MAC/s – more than 100 times faster than conventional DSP solutions.

Using our comprehensive line of industry-leading FPGAs, easy-to-use tools, and optimized algorithms, along with the most comprehensive technical support, services, and third-party programs in the industry, you'll have the confidence to tackle even the most challenging applications using the Xilinx XtremeDSP tool.

DSP Central provides information that will enable you to achieve the maximum benefit from Xilinx DSP solutions. This section provides details and design information in the following areas:

Algorithms/Cores

In this comprehensive listing of intellectual property, you can search for algorithms by type:

- DSP Cores
- Communication and Networking
- Video/Image Processing
- IP Updates.

XILINX ONLINE (IRL)

You can access and upgrade hardware from your desktop anywhere in the world with the Xilinx IRL capability.

The mission of the Xilinx Online program is to enable, identify, and promote any Xilinx programmable system that is connected to a network that can

be fixed, upgraded, or otherwise modified after the system has been deployed in the field.

The design technology for creating Xilinx Online applications is called Internet Reconfigurable Logic, or IRL. IRL design technology comprises robust PLD technology, your network connectivity, and software design tools.

Put these individual pieces together and network-based hardware upgradeability becomes a reality. You can access details through the "Systems Resources" main web page or visit *www.xilinx.com/irl*.

CONFIGURATION SOLUTIONS

Located under the "Products and Services" section of the Xilinx website, this section provides easy-to-use, pre-engineered solutions to configure all Xilinx FPGAs and CPLDs.

All aspects of configuration are explained, whether from a PROM for FPGAs or via ISP for CPLDs. The section also includes third-party Boundary Scan tools, embedded software solutions, ISP cables, ATE and programmer support, and configuration storage devices.

PROCESSOR CENTRAL

This section provides information that will enable you to reap maximum benefits from Xilinx FPGA processing solutions.

It offers the freedom to design a custom solution with a choice of hard processors (as many as four embedded PowerPC processors in Virtex-II Pro devices) or soft processors (with the MicroBlaze™ soft processor core and PicoBlaze™ microprocessor core in Virtex-II, Virtex-E, Spartan-II, and Spartan-IIE FPGAs).

Processor Central also includes more than 40 soft processor peripherals and the necessary embedded software tools to easily complete your design.

The Processor Central section has the following detailed web pages:

The Embedded Development Kit (EDK)

EDK is an all-encompassing solution for designing embedded programmable systems using the IBM PowerPC hard processor core and the Xilinx MicroBlaze soft processor core in Xilinx FPGAs.

PowerPC Embedded Processor Solution

Embedding IBM's PowerPC processor core into the Virtex-II Pro FPGA provides the ultimate platform FPGA solution.

The UltraController Solution

Xilinx offers a powerful, easy-to-use Virtex-II Pro FPGA PowerPC-based microcontroller solution ideal for embedded hardware and software applications.

The UltraController Solution is an easy-to-use, pre-engineered Virtex-II Pro device microcontroller.

Based on the PowerPC and internal block RAMs, UltraController delivers performance exceeding many commercial microcontrollers, yet occupies less than 50 logic cells of fabric.

It is optimized for embedding software/hardware applications in Virtex-II Pro devices, while minimizing the time and effort of building a full CPU-based system.

UltraController uses a simplified software design flow for faster development time. It offers a single hardware HDL module with two memory versions, for use across all applications.

Key features of the UltraController include:

- A completely self-contained PowerPC system
- The ability to implement applications using "C" code
- As many as 32 general-purpose inputs and outputs
- Ultra low power – 0.9 mW/MHz
- Full debug support via JTAG
- Integration with simplified ISE/EDK design flows.

MicroBlaze and PicoBlaze Soft Processor Solutions

Xilinx introduces the industry's fastest 32-bit soft processor core running at 100 D-MIPS on a Virtex-II Pro FPGA. The PicoBlaze 8-bit microprocessor core is the clear leader in FPGA-based soft processors.

Formerly known as KCPSM, the PicoBlaze processor runs at speeds of 116 MHz, yet occupies a tiny footprint of just 154 logic cells. This combination of high performance and miniscule size, when coupled with the Xilinx MicroBlaze product, offers you a broad range of "right-sized" solutions from 8 to 32 bits.

All of Xilinx soft CPUs offer performance that is two to four times faster than competitive offerings, at sizes ranging from one-half to one-fifth the size.

TABLE 2-8: XILINX SOFT PROCESSORS

Soft Processor	Architecture	Bus	MIPS/ Speed	Size	FPGA Support	Support
MicroBlaze	32-bit RISC	Harvard style buses 32-bit instruction and data buses	100 D-MIPs 150 MHz	225 CLBs	Virtex Virtex-E Virtex-II Virtex-II Pro Spartan-II Spartan-IIE	MicroBlaze Developments Kit (MDK) – soft processor core, peripherals, GNU-based software tools (Compiler, assembler, debugger, and linker)
PicoBlaze	8-bit	8-bit address and data busses	35 MIPS 116 MHz	35 CLBs	Virtex Spartan-II	Free of charge reference design and application note, assembler

Third-Party Processors Solution

Both soft processor cores and companion processors are available from third-party sources that support Xilinx devices.

CoreConnect Technology

The IBM CoreConnect bus architecture is an on-chip bus that enables communication between the processor core and its peripherals.

TOOLS AND PARTNERSHIPS

Xilinx offers comprehensive tools to design with its hard and soft processor cores by partnering with industry leaders through our XPERTS and AllianceCORE programs.

MEMORY CORNER

The Memory Corner is a one-stop memory shop, providing solutions for leading-edge memory technologies.

The Memory Corner represents the collaborative efforts of Xilinx and major memory manufacturers including Cypress Semiconductor Corporation, Samsung Semiconductor, Integrated Device Technology, Micron Technology Inc., NEC Electronics, and Toshiba America Electronic Components Corp.

The Memory Corner includes a comprehensive overview of the latest memory technologies in the form of application notes, tutorials, and reference

designs to help simplify the memory selection process and the implementation of the controller.

Xilinx provides embedded memory solutions as well as memory controllers for DRAM and SRAM product families.

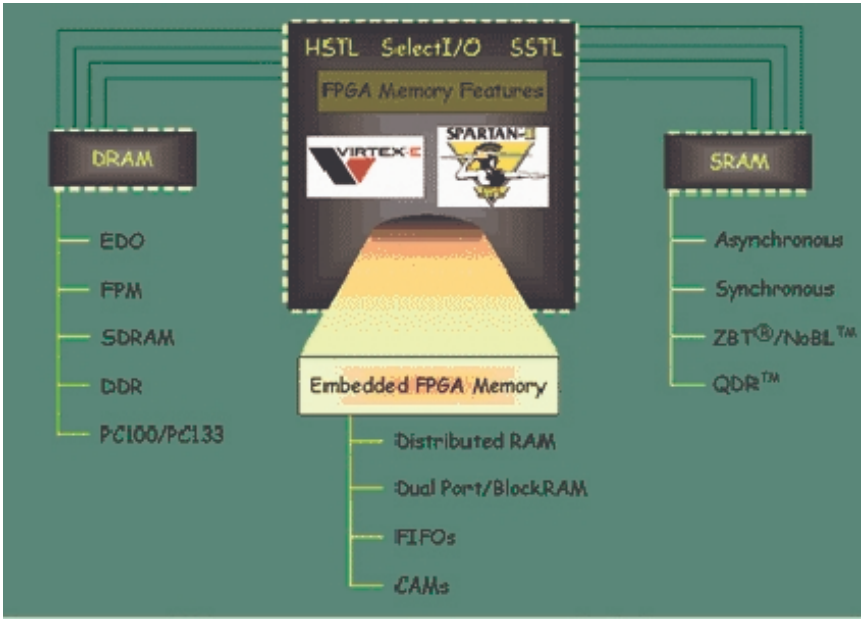


FIGURE 2-47: MEMORY SOLUTIONS ON THE XILINX WEBSITE

SILICON

Xilinx FPGAs are tailored to meet the requirements of different DSP applications. This section helps you to select the most cost-effective silicon solution for applications requiring high-speed DSP.

DESIGN TOOLS AND BOARDS

Xilinx works with industry leaders to provide comprehensive tools for prototyping and development.

- DSP Software Tools (including free evaluations)
- DSP Hardware Tools

TECHNICAL LITERATURE AND TRAINING

This section features an extensive list of DSP application notes, conference papers, white papers, articles, training classes, and online seminars.

CONNECTIVITY CENTRAL

The industry is moving from parallel to serial I/O. By using serial I/O, you can reduce system cost through fewer pins, cheaper PCBs, smaller connectors, lower EMI, better signal integrity, and noise immunity. In addition, serial I/O simplifies your system design and provides scalable bandwidth.

Our dedicated Serial Tsunami Solutions web portal is a one-stop shop for the latest design resources to enable serial design success.

Terabit networks (among other applications) require high-bandwidth system interconnect technology. The Xilinx SystemIO solution provides complete connectivity for high-performance applications, utilizing a combination of the FPGA physical interface, compliant IP cores, design tools, and partnerships.

Networking and Datapath Products

By using the Xilinx SystemIO networking IP cores and reference designs, you can quickly build your edge and core routers, layer2/3+

switches, optical cross connects, and LANs, WANs, and MANs.

Control Plane and Backplane Products

Building on our PCI IP leadership, we are also providing IP cores for more system interconnectivity standards, including PCI Express, PCI-X, Cardbus, and RapidIO.

HIGH-SPEED DESIGN RESOURCES

Xilinx Virtex-II series Platform FPGAs are the ideal solution for building high-performance designs. In addition, Xilinx provides a variety of system tools, reference designs, and application notes for help with your high-speed designs.

SIGNAL INTEGRITY TOOLS

Building a working system today requires knowing a great deal more than just logic design. The documents and links in this area will help you design a reliable PC board quickly.

PARTNERSHIPS

Xilinx works with other networking industry leaders to provide you with a complete connectivity solution, including device interoperability testing, third-party IP, and design services.

SIGNAL INTEGRITY

Building a working system today requires you to know more than just Boolean logic and HDL code. The documents and links in this area are designed to give you everything you need to achieve reliable PCB designs on the first try.

Let's list the areas accessible from this web page:

Signal Integrity Fundamentals

- Overview of SI principles and glossary
- PCB design considerations
- High-density package routing information, PCB checklist, and other resources
- Power supply and bypassing
- Bypass capacitor selection, power consumption, and voltage regulator information
- Thermal design
- Literature and tools for keeping FPGAs cool

Simulation Tools

- IBIS information, models, and simulation tool vendors

Multi-Gigabit Signaling

- Signal integrity in the gigahertz domain

Services

XILINX DESIGN SERVICES

Xilinx Design Services, or XDS, combines skills and experience in system, logic, and embedded software design to provide a unique development partner.

Leveraging these skill sets will help you optimize your budget, schedule, and performance requirements.

The XDS portfolio includes:

IP Core Modification

- Modifications or integration of existing Xilinx LogiCORE products and drivers
- Quick access to LogiCORE source code and a team with experience and expertise with LogiCORE products
- Fixed bid/fixed price contracts

FPGA Design From Specification

- Turnkey FPGA design, ASIC conversions, and driver development and integration
- Expertise in optimizing Xilinx technology to provide the best solution
- Excellent project management to ensure on-time and correct deliveries
- Fixed bid/fixed price contracts

FPGA System Design

- System architecture consulting, FPGA logic, and embedded software

design

- Broad applications experience and immense depth of experience with Xilinx embedded processing tools and products
- Fixed bid/fixed price contracts

Embedded Software Design

- Complex embedded software designs with real-time constraints, driver development, and integration with hardware
- Experience with FPGA platform design, including processors and gates
- Expertise in hardware/software co-design techniques
- Fixed bid/fixed price contracts

Overall, XDS offers:

- Professional project management
- System-level experience around the world
- Faster project ramp-up
- Experienced FPGA design engineers
- FPGA hardware and software experts
- Accelerated knowledge of FPGA systems
- Access to ready-made intellectual property cores.

To find out more, please visit the XDS home page or e-mail *designservices@xilinx.com.*

Education Services

Participation in a Xilinx training course is one of the fastest and most efficient ways to learn how to design with Xilinx FPGA devices. Hands-on experience with the latest information and software allows you to implement your own design in less time, with more effective use of the devices.

Not only design engineers, but test engineers, component engineers, CAD engineers, technicians, and engineering managers may want to participate in the training to understand Xilinx products.

Our learning services provide a number of courses in a variety of delivery methods.

LIVE E-LEARNING ENVIRONMENT

You can choose from more than 70 online classes or modules covering a broad range of topics and skills involving Xilinx products and services.

The one-hour modules are taught weekly at different times throughout the day to support worldwide access. Live instructors present the modules in real time. During each session, you can interact with the instructor or collaborate with online subject experts.

DAY SEGMENT COURSES

Xilinx continues to develop and offer traditional day-length courses.

Working with various Xilinx product development groups, our new courses reflect current product releases. This serves to make training available when you need it and for the products you need.

Classes are held in centers around the world, although specific onsite instruction is also available.

For more information, visit *www.support.xilinx.com* and click on "Courses" under "Education."

COMPUTER-BASED TRAINING (CBT)

Xilinx introduced computer-based training with Verilog CBT, which allows you to learn the Verilog language at your own pace without ever leaving your office.

Verilog CBT is based on the traditional three-day course, converted into a computerized self-study program.

For more information, e-mail *eurotraining@xilinx.com*, telephone +44 (0)870 7350 548, or visit *www.xilinx.com/support/education-home.htm*.

UNIVERSITY PROGRAM

The mission of the Xilinx University Program (XUP) is to promote Xilinx as the technology of choice in the academic community. XUP has provided donations, discounted products, and services to universities since 1985.

Today there are more than 1,600 universities using Xilinx in class labs, or about 18% of all engineering universities worldwide.

The resources available to universities include:

Xilinx University Resource Center

Developed and maintained by the Department of Electrical and Computer Engineering at Michigan State University, the Xilinx University Resource Center website (*xup.msu.edu/*) is designed specifically to support and encourage universities using Xilinx products in the classroom.

Here, you'll find references and resources regarding everything from hardware data sheets to tutorials on using the Xilinx search engine effectively. You can save vast amounts of time and energy by using the resources contained within these pages.

Xilinx Answers Database

The Xilinx Answers Database is located at *www.xilinx.com/support/searchtd.htm*.

Xilinx Student Edition Frequently Asked Questions

The Xilinx Student Edition FAQ is located at *university.xilinx.com/univ/xsefaq1.htm*.

DESIGN CONSULTANTS

The Xilinx XPERTS Program qualifies, develops, and supports design consultants, ensuring that they have superior design skills and the ability to work successfully with customers.

XPERTS is a worldwide program that allows easy access to certified experts in Xilinx device architectures, software tools, and cores.

XPERTS partners also offer consulting in the areas of HDL synthesis and verification, customization and integration, system-level designs, and team-based design techniques.

A listing of partners in the Xilinx XPERTS program is located at *www.xilinx.com/ipecenter*.

For more information on Xilinx products and services, please consult the Xilinx Data Source CDROM in the back of this book, or visit *www.xilinx.com*.

TECHNICAL SUPPORT

Xilinx provides 24-hour access to a set of sophisticated tools for resolving technical issues via the Web.

The Xilinx search utility scans through thousands of answer records to return solutions for the given issue. Several problem-solver tools are also available for assistance in specific areas, like configuration or install.

A complete suite of one-hour modules is also available at the desktop via live or recorded e-learning.

Lastly, if you have a valid service contract, you can access Xilinx engineers over the Web by opening a case against a specific issue.

For technical support, visit *www.support.xilinx.com*.

CHAPTER 3

WebPACK ISE Design Software

WebPACK ISE design software offers a complete design suite based on the Xilinx ISE series software. This chapter describes how to install the software and what each module does.

Module Descriptions

Individual WebPACK ISE modules give you the ability to tailor the design environment to your chosen PLDs as well as the preferred design flow.

In general, the design flow for FPGAs and CPLDs is identical. You can choose whether to enter the design in schematic form or in HDL, such as VHDL, Verilog, or ABEL.

The design can also comprise of a mixture of schematic diagrams and embedded HDL symbols. There is also a facility to create state machines in a diagrammatic form and let the software tools generate optimized code from a state diagram.

WebPACK ISE software incorporates a Xilinx version of the ModelSim simulator from Model Technology (a Mentor Graphics company), referred to as MXE (ModelSim Xilinx Edition).

This powerful simulator is capable of simulating functional VHDL before synthesis, or simulating after the implementation process for timing verification.

WebPACK ISE software offers an easy-to-use GUI to visually create a test pattern. A testbench is then generated and compiled into MXE, along with the design under test.

The flow diagram below shows the similarities and differences between CPLD and FPGA software flows.

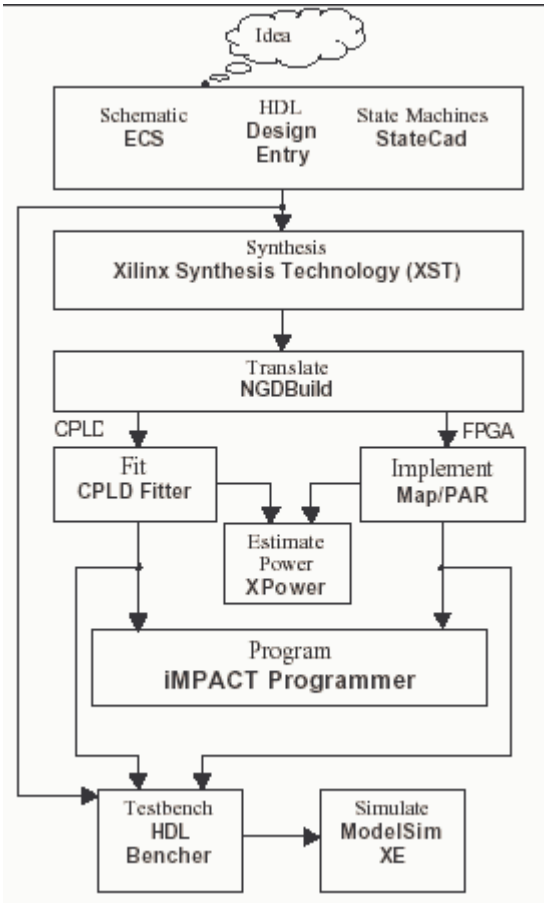


FIGURE 3-1: WEBPACK SOFTWARE DESIGN FLOW

When your design is complete and you're happy with the simulation results, you can then download the design to the required device.

Although we'll discuss details of the software flow in Chapters 5 and 6, here's a basic overview.

For FPGAs, the implementation process includes four key steps.

1. **Translate** – Interprets the design and runs a "design rule check."
2. **Map** – Calculates and allocates resources in the targeted device.
3. **Place and Route** – Places the CLBs in a logical position and utilizes the routing resources.

4. **Generate Programming File** – Creates a programming bitstream.

For CPLDs, the implementation process includes:

1. **Translate** – Interprets the design and runs a "design rule check."
2. **Fit** – Allocates resource usage and connections.
3. **Generate Programming File** – Creates a JED file for programming.

WebPACK Design Suite

TABLE 3-1: **WEBPACK DEVICE SUPPORT**

Device	Support
Virtex-II Pro	Up to XC2VP2
Virtex-II	Up to XC2V250
Virtex-E	Up to XCV300E
Spartan-IIE	Up to XC2S300E
Spartan-II	Up to XC2S200
Spartan-3	Up to XC3S400
CoolRunner-II	All
CoolRunner	All
XC9500 Families	All

WEBPACK DESIGN ENTRY

The WebPACK tool suite supports several different design entries. The XST synthesis tool synthesizes HDL code in VHDL, Verilog, or ABEL into a netlist.

Schematic designs are converted into VHDL or Verilog, which are then synthesized by XST in the same way.

WEBPACK STATECAD

StateCAD is a tool for graphically entering state machines in "bubble diagram" form.

You simply draw the states, transitions, and outputs, and StateCAD gives a visual test facility. State machines are generated in HDL and then added to the WebPACK ISE project.

WEBPACK MXE SIMULATOR

The WebPACK MXE Simulator can be used for both functional and timing simulation. The necessary libraries are already pre-compiled into MXE. Pre-written scripts seamlessly compile the design to be tested, as well as its test-bench.

For functional simulation, the written code is simulated before synthesis. After fitting (CPLDs) or place and route (FPGAs), you can simulate the design using the same original testbench as a test fixture, but with logic and routing delays added.

WEBPACK HDL BENCHER TOOL

The HDL Bencher tool generates the testbenches, allowing you to simulate the design under test.

The HDL Bencher tool reads the design under test. You simply enter signal transitions in a graphical timing diagram GUI.

You can also enter your expected simulation results, allowing the simulator to flag a warning if the simulation did not yield the expected results.

WEBPACK FPGA IMPLEMENTATION TOOLS

As we discussed, there are several steps to implementing an FPGA design. Xilinx FPGA implementation tools perform all of these steps.

WEBPACK CPLD IMPLEMENTATION TOOLS

Similarly, CPLD implementation tools perform all of the steps in the CPLD implementation flow outlined previously.

WEBPACK IMPACT PROGRAMMER

The iMPACT programmer module allows you to program a device in-system for all devices available in the WebPACK software. (You must connect a JTAG cable to the PC's parallel port.)

For FPGAs, the programmer module allows you to configure a device via the JTAG cable.

Xilinx FPGAs are based on a volatile SRAM technology, so the device will not retain configuration data when you remove the power. Therefore, this configuration method is normally only used for test purposes.

CPLDs, however, are non-volatile devices. Once programmed, they will retain their program until it's erased or reprogrammed.

The programmer module also includes a PROM file formatter. The use of an external PROM is a popular method of storing FPGA configuration data.

The PROM file formatter takes in the bitstream generated during the implementation phase and provides an MCS- or HEX-formatted file used by PROM programmers.

WEBPACK CHIPVIEWER

The ChipViewer tool can be used to examine the design after implementation. It shows the connections between pins of the device, as well as the configuration of the internal logical resources.

XPOWER

As power consumption becomes increasingly important in modern digital designs, the XPower tool is available to calculate the power consumption of a design running inside a device.

WebPACK CD-ROM Installation

First, insert the CD. If the installation does not start automatically, navigate to the CD drive using Microsoft Windows Explorer software.

Double-click on the setup.exe file to start the installation process. (The installation process may have already started automatically).

As the installation process starts up, you'll see a window asking for a registration key. To get the registration key, enter the product ID on the CD sleeve at the website given in the window.

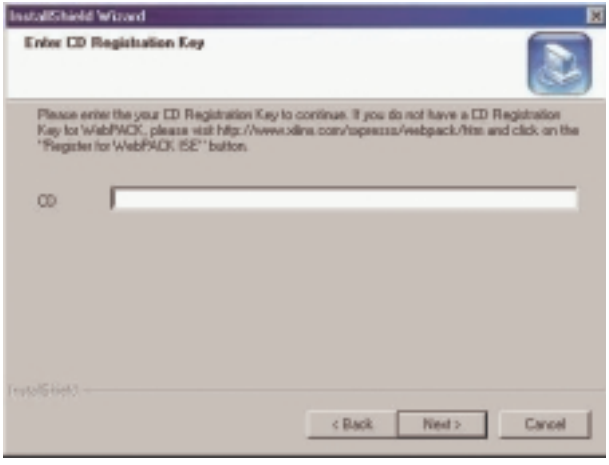


FIGURE 3-2: REGISTRATION KEY WINDOW

Once at the registration web page, follow the online registration process by selecting "new customer please register" from the first online screen.

Enter the data requested at each stage. You'll need to create and enter a memorable user name and password.

When requested, enter the product ID in the appropriate field.

A CD registration key number will then be sent to you via e-mail.

Your key number will look something like this:

1234-xxxx-xxxx

To proceed with installation, enter your key number into the WebInstall Wizard CD Registration Key window and select the "next" button.

Select the WebPACK software configuration you wish to install from the following choices:

TABLE 3-2: **WEBPACK SOFTWARE OPTIONS**

Software Focus Area	Module Type	Description
	Complete Tool Set	Everything required for CPLDs
	Design Environment Only	CPLD Design Entry only
CPLD	Programming Tools Only	CPLD Programming tool only
	Optional Tools	XPower and ChipViewer
CPLD and	Complete ISE WebPACK Software	Everything required for both CPLD and FPGA designs
FPGA	Complete Device Programming Software	Programming support for both CPLDs and FPGAs

If you have enough disk space, we recommend that you install the complete ISE WebPACK software, although it's also possible to upgrade later.

Getting Started

LICENSES

The MXE Simlulator is the only tool that requires a license.

MXE Simulator is licensed via the FlexlM product from Macrovision. It requires you to situate a starter license file on your hard drive, pointed to by a `set lm_license_file` environment setting.

The license is free and is applied for online after installation, after which you'll receive a license.dat file via e-mail.

From the Start menu, go to Programs > ModelSimXE 5.xx > Submit License Request.

PROJECTS

When starting a project the default location of the project will be:

c:\Xilinx_WebPACK\bin\nt

You can create a unique directory on your hard drive for working on projects, e.g. c:\my_projects.

Should you need to uninstall and reinstall WebPACK ISE software due to problems on your system, we recommend that you delete the entire WebPACK ISE directory structure.

Summary

In this chapter, we explained the functions of all WebPACK ISE modules, along with installation instructions of the modules you require.

You can decide which modules are necessary for your intended design and install only relevant modules.

In the next section, we'll take you through your first PLD design using the powerful features of WebPACK ISE software. Our example design is a simple traffic light controller that uses a VHDL counter and a state machine. The design entry process is identical for FPGAs and CPLDs.

CHAPTER 4

WebPACK ISE Design Entry

Introduction

This chapter describes a step-by-step approach to a simple design.

The following pages are intended to demonstrate the basic PLD design entry and implementation process.

In this example tutorial, you'll design a simple traffic light controller in VHDL.

Our design is initially targeted at a CoolRunner-II CPLD, and we also show how you can convert the project to target a Spartan-3 FPGA.

We'll also discuss some of the advanced features of the architecture.

Design Entry

To start WebPACK ISE software, select Start > Programs > Xilinx ISE 6 > Project Navigator.

To create a new project, select File > New Project.

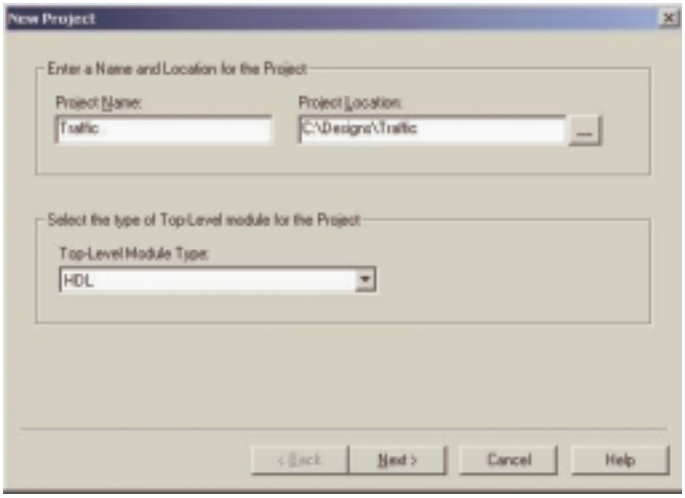


FIGURE 4-1: NEW PROJECT WINDOW – PROJECT NAME

Call the project "Traffic" and put it in your Designs directory. For this tutorial, we will be using an HDL top level.

Click the Next> button.

Enter the following into the New Project dialog box:

```
Device Family:                  CoolRunner-II
Device:                         xc2c256
Package:                        TQ144
Speed Grade:                    -7
Synthesis Tool:                 XST (VHDL/Verilog)
Simulator:                      ModelSim
Generated Simulation Language:  VHDL
```

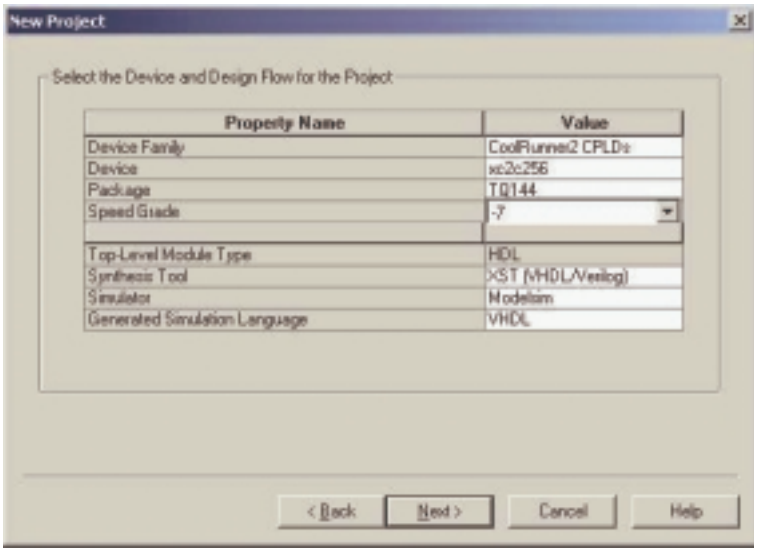


FIGURE 4-2: NEW PROJECT WINDOW – DEVICE AND DESIGN FLOW

Click the Next> button.

Add a new source to the project by clicking on the New Source button.

Add a VHDL module and call it “Counter.”

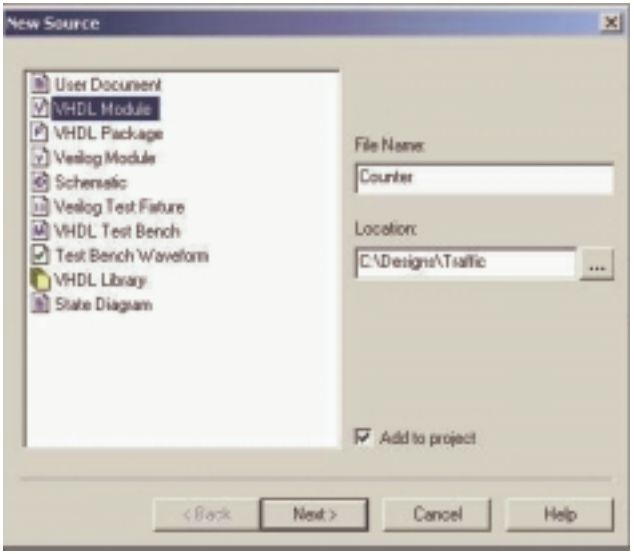


FIGURE 4-3: NEW SOURCE WINDOW

Click the Next> button.

Create a 4-Bit Counter Module

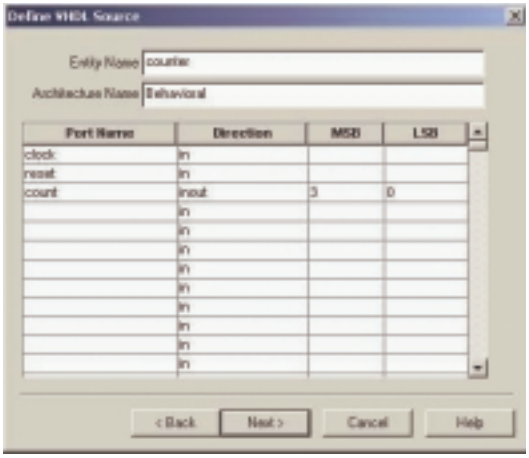


FIGURE 4-4: DEFINE VHDL SOURCE WINDOW

Declare three ports: “clock,” “reset,” and “count.” The clock and reset ports should both be of direction “in.”

Count should be direction “inout” and should be a 4-bit vector with MSB 3, LSB 0.

Click the Next> button.

Review the contents of the final window and click the Finish button.

This has automatically generated the entity in the counter VHDL module.

Notice that a file called "counter.vhd" has been added to the project in the Sources in Project window of the Project Navigator.

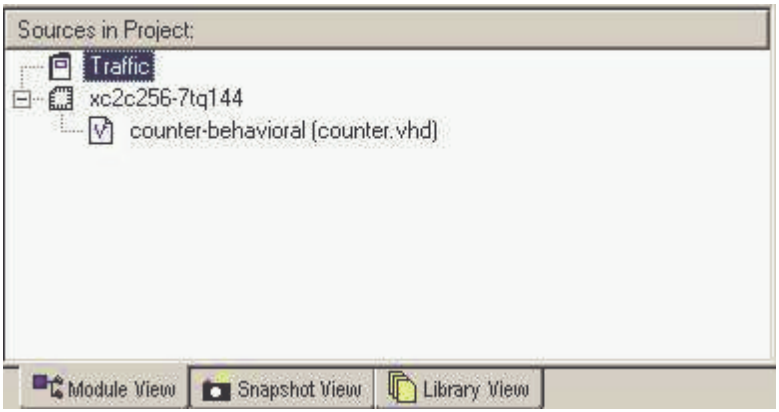


FIGURE 4-5: SOURCE IN PROJECT WINDOW

Double-click on this source to open it in the WebPACK ISE Editor window.

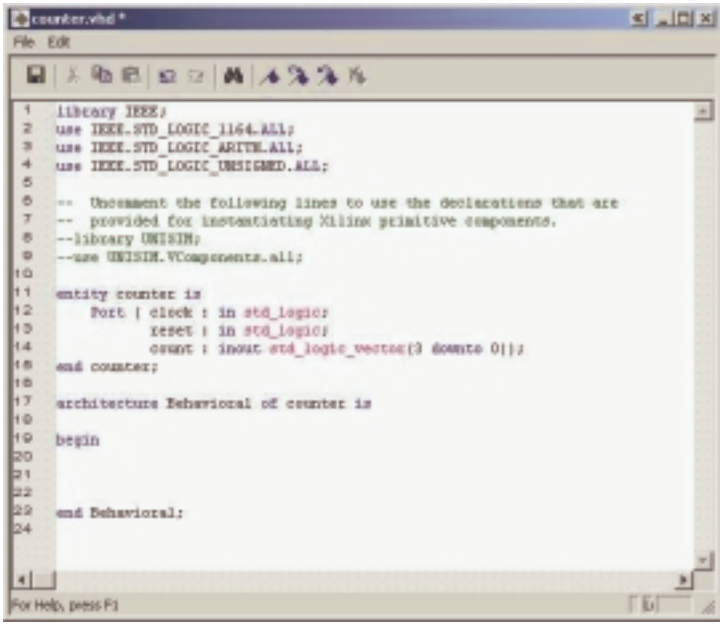


FIGURE 4-6: COUNTER WINDOW

You can remove the source files from the WebPACK ISE GUI by clicking on the add/remove arrow.

As the project builds, you will notice how the WebPACK ISE tool manages hierarchy and associated files in the Sources window.

Double-clicking on any file name in the Sources window allows that file to be edited in the main Text Editor.

THE LANGUAGE TEMPLATE

The language template is an excellent tool to assist you in creating HDL code.

It has a range of popular functions such as counters, multiplexers, decoders, and shift registers. It also has templates for creating common operators (such as "IF/THEN" and "FOR" loops) often associated with software languages.

Language templates are used as a reference. They can be copied and pasted into the design, then customized for their intended purpose.

Usually, you must change the bus width or signal names, and sometimes modify the functionality.

In this tutorial, the template uses the signal name "clk." The design requires the signal to be called "clock." The counter in the template is too complex for this particular requirement, so some sections are deleted.

Open the language templates by clicking the button located on the far right side of the toolbar.

You can also access the language template from the Edit > Language Template menu.

Click and drag the counter template from the VHDL > Synthesis > Templates folder and drop it into the "counter.vhd" architecture between the begin and end statements.

An alternate method is to highlight the code in the language template that you want to use in your code and right-click your mouse button. Then select use in "counter.vhd."

CLOSE THE LANGUAGE TEMPLATES

Notice the color-coding used in the HDL Editor. The green text indicates a comment. The commented text in this template shows which libraries are required in the VHDL header. The port definitions are required if this counter was used in its entirety.

As you have already created the entity, this information is not required. You can delete the green comments if you wish.

The counter from the template shows a loadable bidirectional counter. For this design, only a 4-bit up counter is required.

EDIT THE COUNTER MODULE

Replace "clk" with the word "clock" by using the Edit > Replace function.

Delete this section:

```
if CE='1' then
          if LOAD='1' then
                  COUNT <= DIN;
                else
          if DIR='1' then
```

Delete this section:

```
        else
               COUNT <= COUNT - 1;
              end if;
            end if;
          end if;
```

The counter module should now look like Figure 4-7.

For the purposes of debugging code, several new features are available in the Source Editor window.

A right-click in the gray bar on the left side of the Source Editor window will bring up a menu of these features.

You can toggle the line numbers in the side bar on or off and place bookmarks to mark lines of interest in the source file.

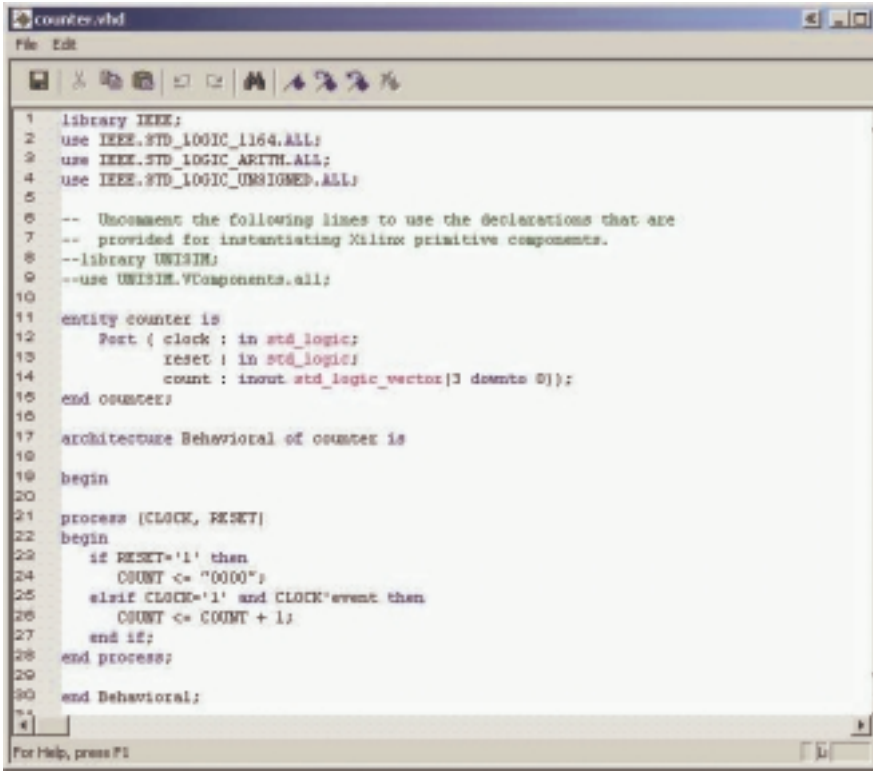


FIGURE 4-7: COUNTER IN VHDL WINDOW

A typical VHDL module consists of library declarations, an entity, and an architecture.

The library declarations are needed to tell the compiler which packages are required.

The entity declares all ports associated with the design. Count (3 down to 0) means that count is a 4-bit logic vector.

This design has two inputs – clock and reset – and one output, a 4-bit bus called "count."

The actual functional description of the design appears after the begin statement in the architecture.

The function of this design is to increment a signal "count" when clock = 1 and there is an event on the clock. This is resolved into a positive edge.

The reset is asynchronous as is evaluated before the clock action.

The area still within the architecture – but before the begin statement – is where declarations reside.

We'll give some examples of component and signal declarations later in this chapter.

SAVE THE COUNTER MODULE

You can now simulate the counter module of the design.

With "counter.vhd" highlighted in the Source window, the Process window will give all the available operations for that particular module.

A VHDL file can be synthesized and then implemented through to a bitstream.

Normally, a design consists of several lower-level modules wired together by a top-level file. This design currently only has one module that can be simulated.

Functional Simulation

To simulate a VHDL file, you must first create a testbench.

From the Project menu, select "New Source" as before.

Select "Test Bench Waveform" as the source type and give it the name "counter_tb."

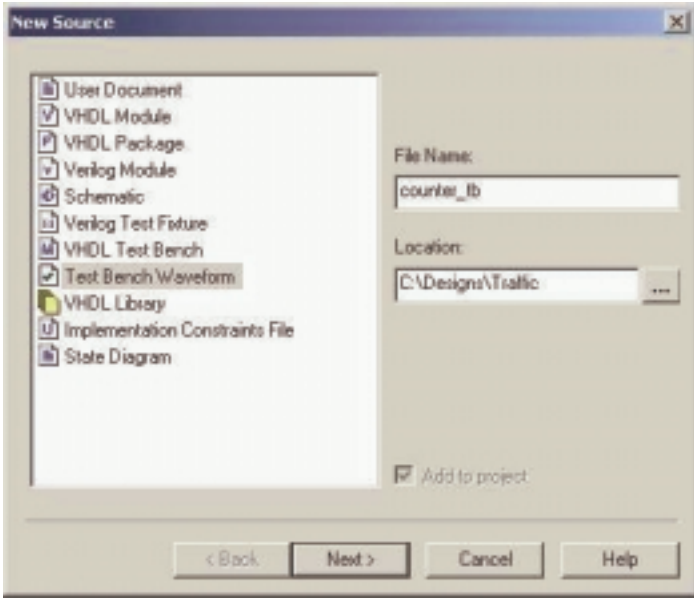


FIGURE 4-8: NEW SOURCE WINDOW

Click the Next> button.

The testbench is going to simulate the counter module, so when asked which source you want to associate the source with, select "Counter" and click the Next> button.

Review the information and click the Finish button.

The HDL Bencher tool now reads in the design. The "Initialize Timing" box sets the frequency of the system clock, setup requirements, and output delays.

Set initialize timing as follows and click OK:

Clock high time:	50 ns
Clock low time:	50 ns
Input setup time:	10 ns
Output valid delay:	10 ns

FIGURE 4-9: **HDL BENCHER WINDOW**

Note that the blue cells are for entering input stimulus and the yellow cells are for entering expected response.

When entering a stimulus, clicking the left mouse button on the cell will cycle through the available values for that cell.

Open a pattern text field and button by double-clicking on a signal's cell or single-clicking on a bus cell. From this Pattern window, you can enter a value in the text field or click on the Pattern button to open a Pattern Wizard.

Enter the input stimulus as follows:

Set the RESET cell below CLK cycle 1 to a value of "1."

Set the RESET cell below CLK cycle 2 to a value of "0."

Enter the expected response as follows:

Click the yellow COUNT[3:0] cell under CLK cycle 1 and click the Pattern button to launch the Pattern Wizard. Set the Pattern Wizard parameters to count up from 0 to 1111 (see Figure 4-10).

Click OK to accept the parameters.

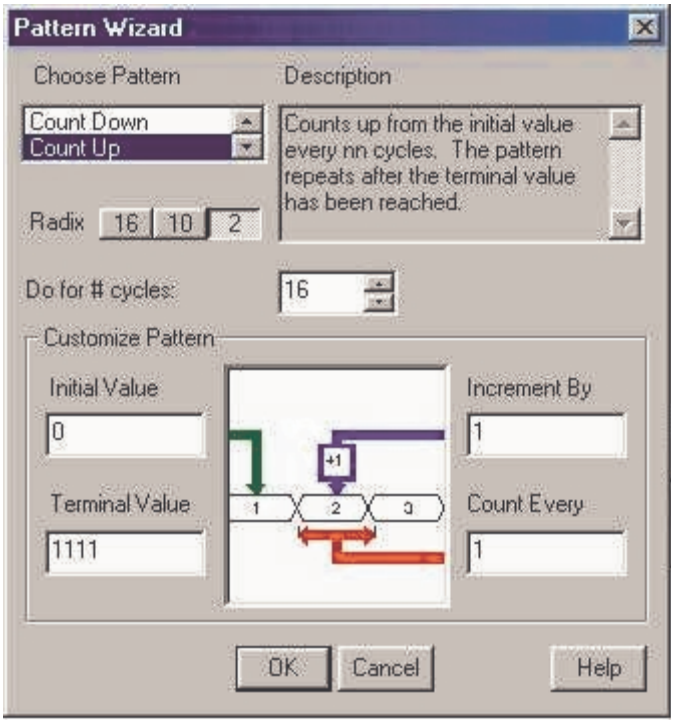


FIGURE 4-10: PATTERN WIZARD WINDOW

Your waveform should look like Figure 4-11.

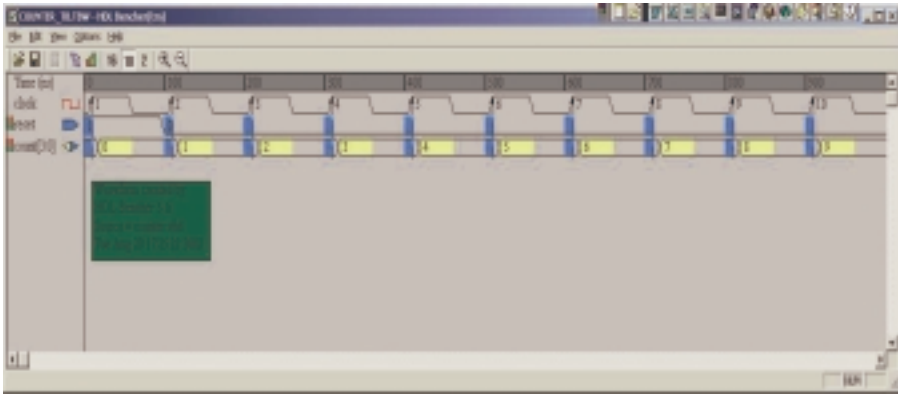


FIGURE 4-11: WAVEFORM WINDOW

Click File > Save to save the waveform.

Close the HDL Bencher tool.

The ISE Sources in Project window should look like Figure 4-12.

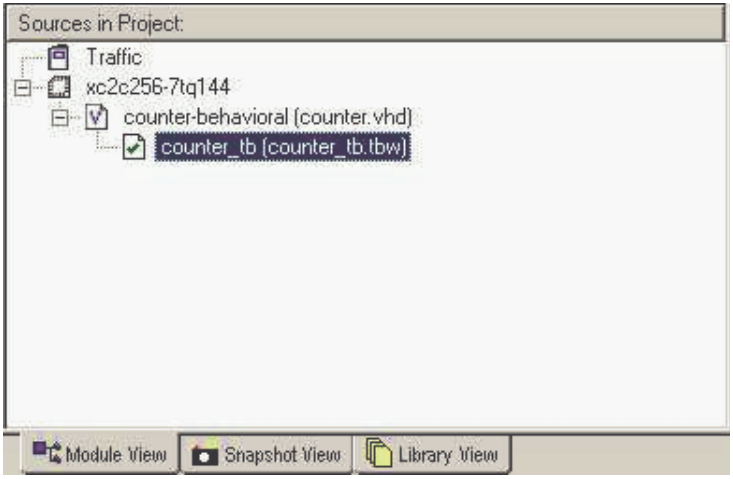


FIGURE 4-12: NEW SOURCES IN PROJECT WINDOW

To make changes to the waveform used to create the testbench, double-click on "counter_tb.tbw."

Now that the testbench is created, you can simulate the design.

Select "counter_tb.tbw" in the ISE Source window. In the Process window, expand the ModelSim simulator by clicking; then right-click on "Simulate Behavioral VHDL Model."

Selelect Properties.

In the Simulation Run Time field, type "–all" and hit OK.

By default, MXE will only run for 1us. The –all property runs MXE until the end of the testbench.

In the Process window, double-click on "Simulate Behavioral VHDL Model."

This will bring up the Model Technology MXE dialog box.

WebPACK ISE software automates the simulation process by creating and launching a simulation macro file (a ".do" file, or in this case an ".fdo" file). This creates the design library, compiles the design and testbench source files, and calls a user-editable ".do file" called "counter_tb.udo."

It also invokes the simulator, opens all the viewing windows, adds all the signals to the Wave window, adds all the signals to the List window, and runs the simulation for the time specified by the simulation run time property.

Maximize the Wave window. From the Zoom menu, select Zoom Full.

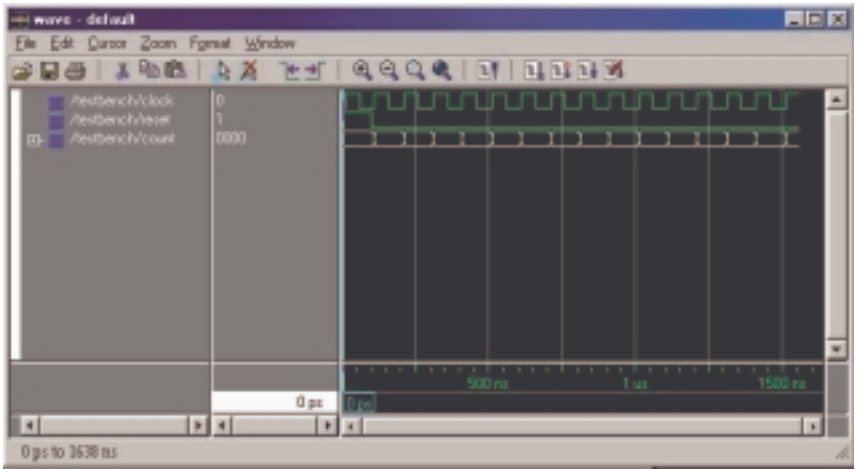


FIGURE 4-13: WAVE WINDOW

Use File > Exit to close the ModelSim simulator.

Alternatively, closing the main ModelSim window using the usual close window button will close down the ModelSim program.

Take a snapshot of your design by selecting Project > Take Snapshot.

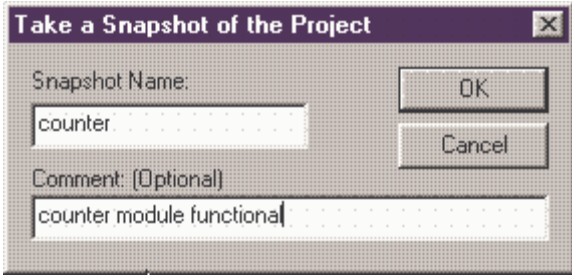


FIGURE 4-14: PROJECT SNAPSHOT WINDOW

Taking a snapshot of your project saves the current state of your project in a subdirectory (with the same name as the snapshot) so that you can go back to it in the future. You can view project snapshots by selecting the Sources window snapshot tab in the Project Navigator.

If the design had only one module (one level of hierarchy), the implementation phase would be the next step.

This design, however, has a further module to represent a more typical VHDL design.

State Machine Editor

For our traffic light design, the counter acts as a timer that determines the transitions of a state machine.

The state machine will run through four states, each state controlling a combination of the three lights.

State 1: Red Light

State 2: Red and Amber Light

State 3: Green Light

State 4: Amber Light

To invoke the state machine editor, select New Source from the project menu.

Highlight State Diagram and give it the name "stat_mac.dia."

Click the Next> button, then the Finish button.

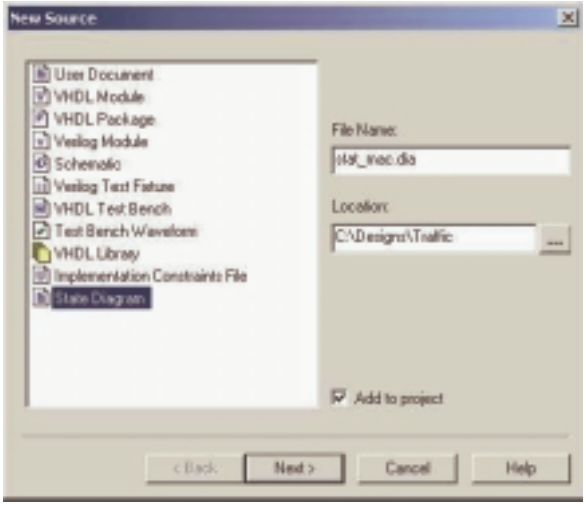


FIGURE 4-15: NEW SOURCE WINDOW

Open the State Machine Wizard by clicking on the button in the main toolbar.

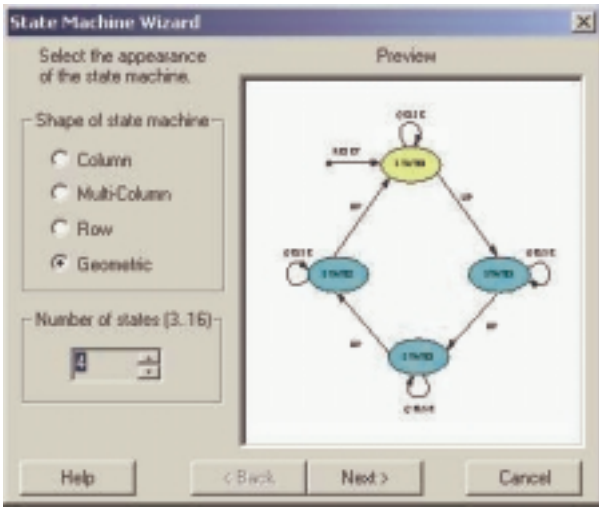


FIGURE 4-16: STATE MACHINE WIZARD WINDOW

Set the number of states to "4" and hit the Next> button.

Click the Next> button to build a synchronous state machine.

In the Setup Transitions box, type "TIMER" in the Next field (shown in Figure 4-17).

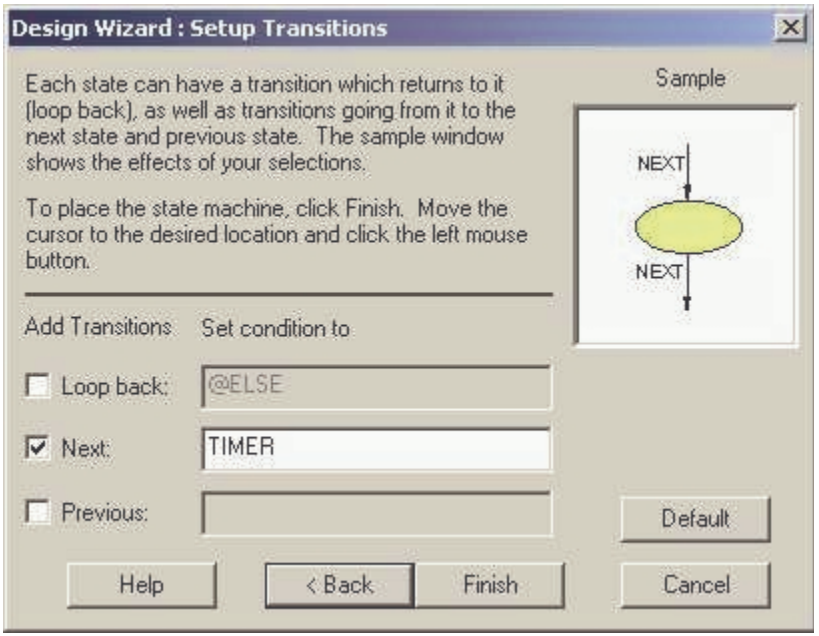


FIGURE 4-17: SETUP TRANSITIONS WINDOW

Click on the Finish button and drop the state machine on the page.

Double-click on the Reset State 0 colored yellow. Rename the state name "RED."

Hit the Output Wizard button.

This design will have three outputs named RD, AMB, and GRN. In the DOUT field, type "RD" to declare an output.

Set RD to a constant “1” with a registered output, as shown in Figure 4-18.

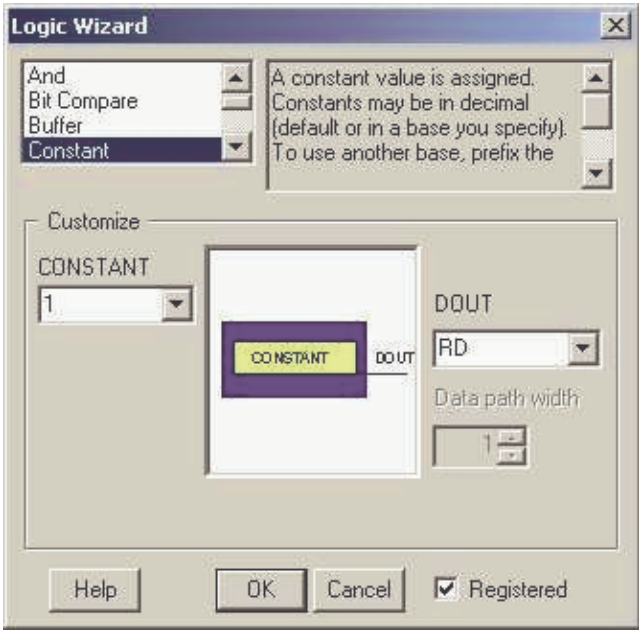


FIGURE 4-18: LOGIC WIZARD WINDOW

Click on OK and then OK the Edit State box.

In a similar fashion, edit the other states:

Rename State 1 to “REDAMB” and use the output wizard to set RD = 1, and a new output AMB equal to “1” with a registered output.

Rename State 2 to “GREEN” and use the output wizard to set a new output GRN equal to “1” with a registered output.

Rename State 3 to “AMBER” and use the output wizard to set AMB = 1.

The state machine should look like Figure 4-19.

(If you set a signal as registered in the Output Wizard and then select Signal and re-open the wizard, it is no longer ticked as registered.)

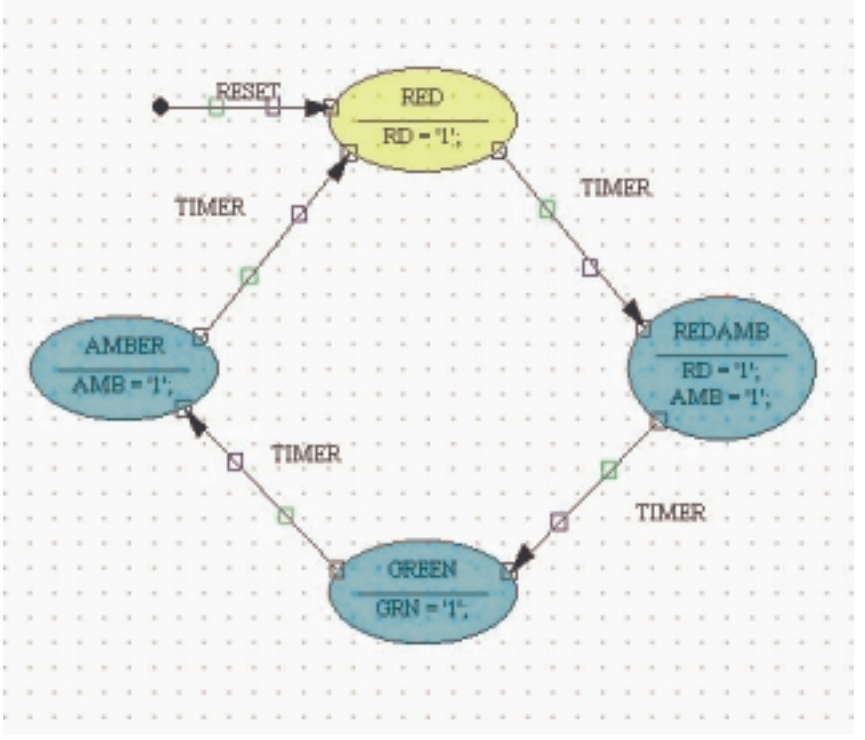


FIGURE 4-19: STATE DIAGRAM

Double-click on the transition line between state "RED" and state "REDAMB."

In the edit condition window, set a transition to occur when timer is 1111 by editing the Condition field to TIMER = "1111." (Don't forget the double quotes ("), as these are part of VHDL syntax.).

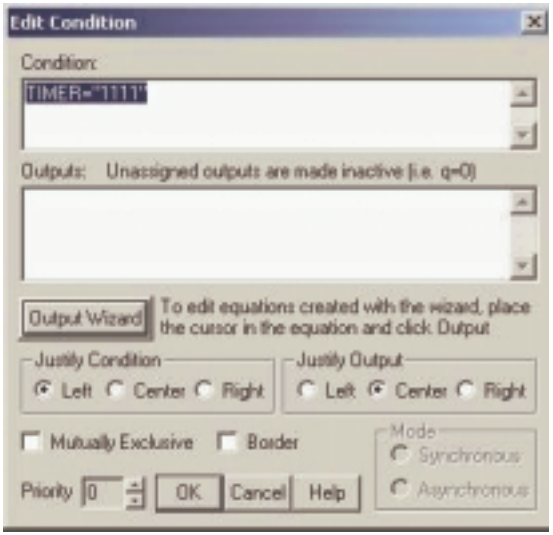


FIGURE 4-20: EDIT CONDITIONS WINDOW

Repeat for the other transitions:

Transition REDAMB to GREEN, TIMER = "0100"

Transition GREEN to AMBER, TIMER = "0011"

Transition AMBER to RED, TIMER = "0000"

Hence, the traffic light completes a RED, REDAMB, GREEN, AMBER once every three cycles of the counter.

Finally, declare the vector TIMER by clicking on the button on the left-hand side of the toolbar.

Drop the marker on the page, double-click on it, and enter the name "TIMER" with a width of 4 bits (Range 3:0).

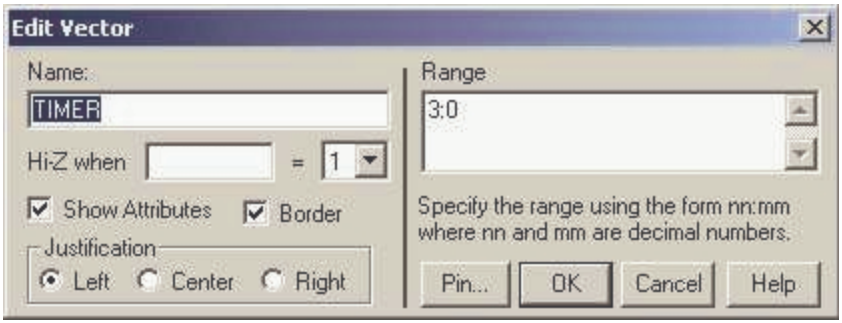


FIGURE 4-21: EDIT VECTOR WINDOW

Click OK.

Your completed state machine should look like Figure 4-22.

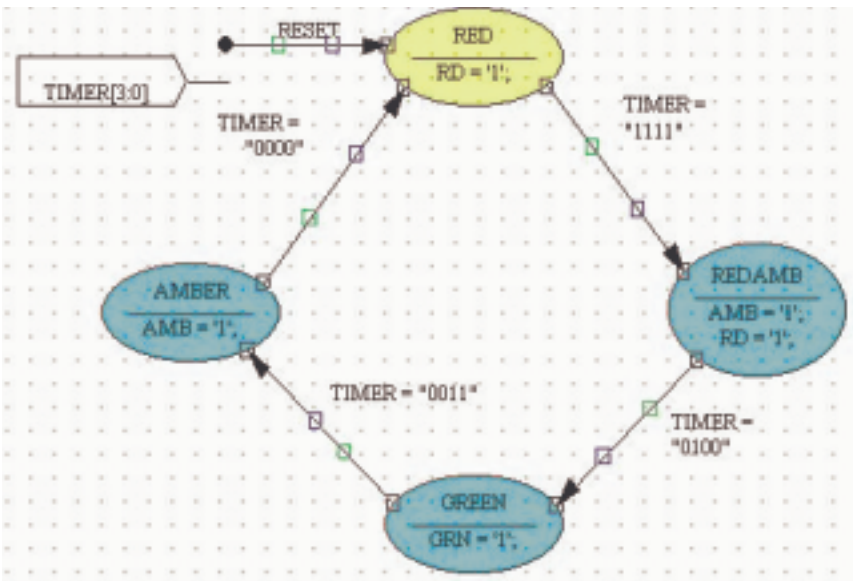


FIGURE 4-22: STATE MACHINE DRAWING

Click on the Generate HDL button on the top toolbar.

The Results window should read "Compiled Perfectly." Close the dialog box and the generated HDL Browser window.

Save and close StateCAD.

The state machine can now be added to the WebPACK ISE project.

In the Project Navigator, go to the Project menu and select Add Source.

In the Add Existing Sources box, find "STAT_MAC.vhd."

Click on Open and declare it as a VHDL Module.

In the Project Navigator, go to the Project menu and select Add Source.

In the Add Existing Sources box, find "stat_mac.dia."

The state diagram will be added to the top of the Sources window. Double-clicking on this file will open up the state diagram in StateCAD.

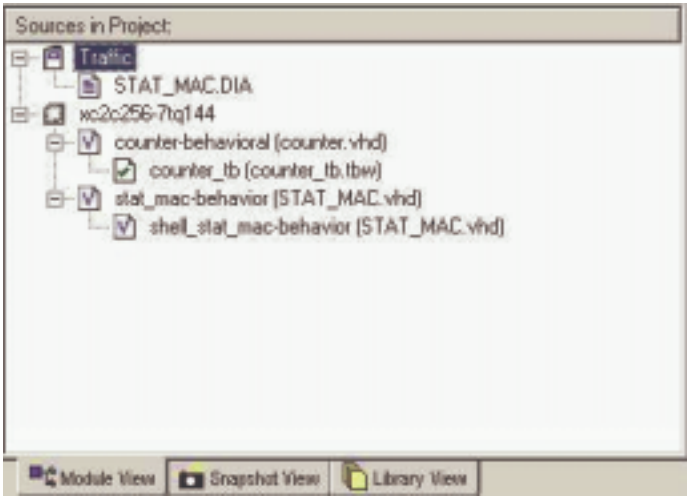


FIGURE 4-23: SOURCE IN PROJECT WINDOW SHOWING MODEL VIEW

Top-Level VHDL Designs

At this point in the flow, two modules in the design are connected together by a top-level file.

Some designers like to create a top-level schematic diagram, while others like to keep the design entirely text-based.

Because this section discusses the latter, the counter and state machine will be connected using a top.vhd file.

If you prefer the former, jump directly to the next section **Top-Level Schematic Designs**, **page 125**. You will have the opportunity to do both by continuing through this tutorial.

Take a snapshot of the project from Project > Take Snapshot.

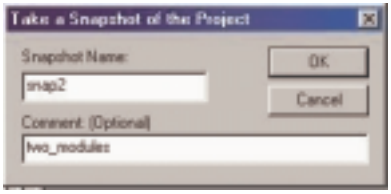


FIGURE 4-24: PROJECT SNAPSHOT

From the Project menu, select New Source and create a VHDL module called "top."

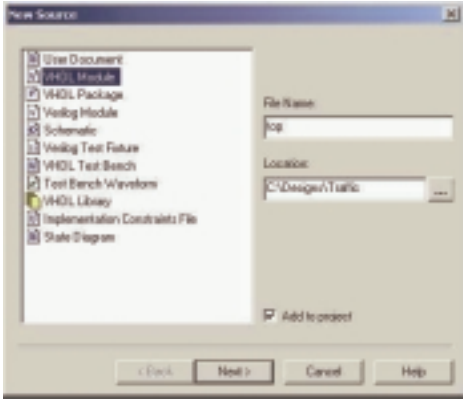


FIGURE 4-25: NEW SOURCE WINDOW SHOWING VHDL MODULE

Click on the Next> button and fill out the Define VHDL Source dialog box, as shown in Figure 4-26.

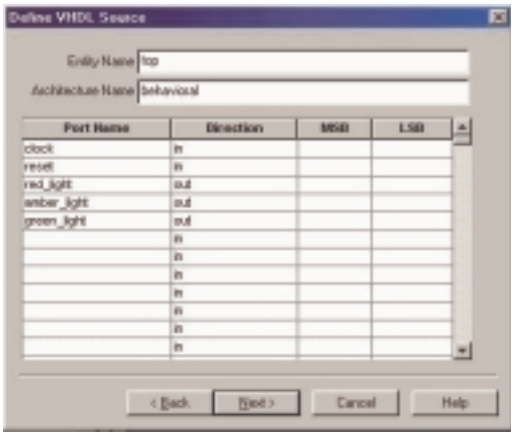


FIGURE 4-26: DEFINE VHDL SOURCE WINDOW

Click on the Next> button, then the Finish button.

Your new file, "top.vhd," should look like Figure 4-27.

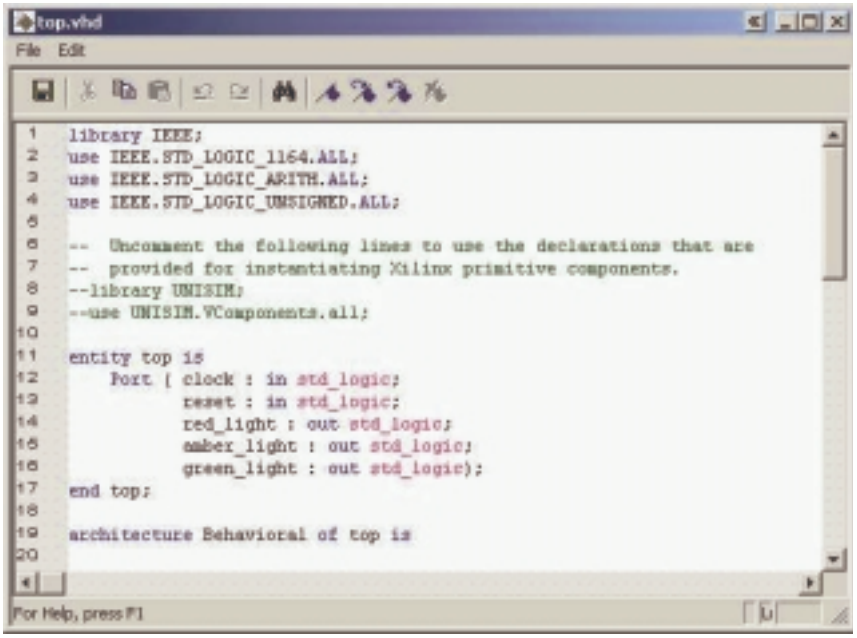


FIGURE 4-27: NEW VHDL FILE

In the Sources window, highlight "counter.vhd."

In the Process window, double-click View VHDL Instantiation Template from the Design Entry Utilities section.

Highlight and copy the component declaration and instantiation.

```
COMPONENT counter
PORT(
        CLK : IN std_logic;
        RESET : IN std_logic;
        COUNT : INOUT std_logic_vector(3 downto 0)
        );
END COMPONENT;

Inst_counter: counter PORT MAP(
        CLK => ,
        RESET => ,
        COUNT =>
);
```

FIGURE 4-28: INSTANTIATION TEMPLATE

Close the instantiation template, as shown in Figure 4-28.

Paste the component declaration and instantiation into "top.vhd."

Rearrange the component declaration so that it lies before the begin statement in the architecture. Rearrange the instantiation so that it lies between the begin and end statement (see Figure 4-29 for reference).

Highlight "stat_mac.vhd" in the Sources window and double-click View VHDL Instantiation Template from the Design Utilities section.

Repeat the copy-and-paste procedure previously described.

Declare a signal called "timer" by adding the following line above the component declarations inside the architecture:

```
signal timer : std_logic_vector(3 downto 0);
```

Connect the counter and state machine instantiated modules so that your "top.vhd" file looks like Figure 4-29.

```
architecture behavioral of top is

signal timer : std_logic_vector (3 downto 0);
COMPONENT counter
   PORT(
      CLOCK : IN std_logic;
      RESET : IN std_logic;
      COUNT : INOUT std_logic_vector(3 downto 0)
      );
   END COMPONENT;
   COMPONENT stat_mac
   PORT(
      TIMER : IN std_logic_vector(3 downto 0);
      CLK : IN std_logic;
      RESET : IN std_logic;
      AMB : OUT std_logic;
      GRN : OUT std_logic;
      RD : OUT std_logic
      );
   END COMPONENT;

begin
   Inst_counter: counter PORT MAP(
      CLOCK => clock,
      RESET => reset,
      COUNT => timer
   );
   Inst_stat_mac: stat_mac PORT MAP(
      TIMER => timer,
      CLK => clock,
      RESET => reset,
      AMB => amber_light,
      GRN => green_light,
      RD => red_light
   );
end behavioral;
```

FIGURE 4-29: TOP.VHD FILE

When you save "top.vhd," notice how the Sources window automatically manages the hierarchy of the whole design, with "counter.vhd" and "stat_mac.vhd" becoming sub-modules of "top.vhd."

You can now simulate the entire design.

Add a new testbench waveform source as before, but this time, associate it with the module "top."

Accept the timing in the Initialize Timing dialog box and click OK.

In the waveform diagram, enter the input stimulus as follows:

Set the RESET cell below CLK cycle 1 to a value of "1."

Click the RESET cell below CLK cycle 2 to reset if low.

Scroll to the 64th clock cycle; right-click and select Set End of Testbench.

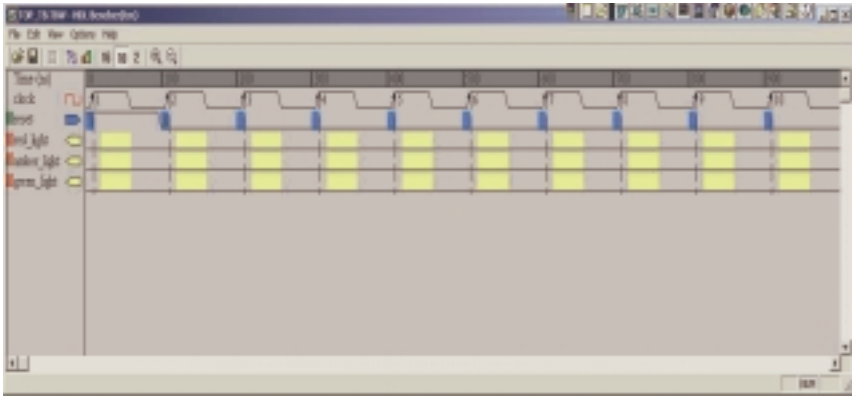


FIGURE 4-30: ***Waveform Diagram***

Close the Edit Test Bench window.

Click the Save Waveform button.

Close the HDL Bencher tool.

The "top_tb.tbw" file will now be associated with the top-level VHDL module.

Simulate Functional VHDL Model in the Process window.

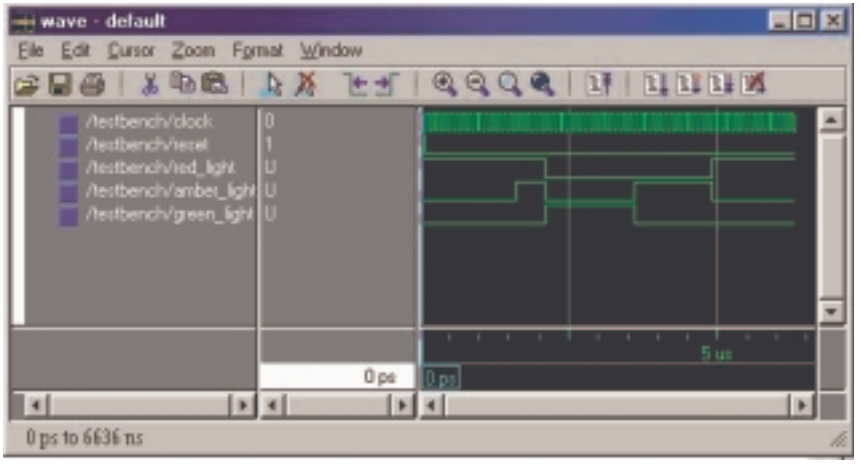


FIGURE 4-31: WAVEFORM WINDOW

You are now ready to go to the implementation stage.

Top-Level Schematic Designs

Sometimes, it's easier to visualize designs when they have a schematic top level that instantiates the individual blocks of HDL. The blocks can then be wired together in the traditional method.

For designs in the WebPACK ISE tool, the entire project can be schematic-based.

This section discusses the method of connecting VHDL modules via the ECS schematic tool.

If you have worked through the previous session, you will first need to revert to the screen shown in Figure 4-32 (two modules with no top-level file).

At the bottom of the Sources window, select the Snapshot View tab.

Highlight Snap2 (two modules). In the Project menu, select Make Snapshot Current.

This action will take you back to the stage in the flow with only the "counter.vhd" and the "stat_mac.vhd" files.

WebPACK ISE software will ask if you would like to take another snapshot of the design in its current state.

Select Yes and create a third snapshot called "vhdl_top."

The Sources window module view should look like Figure 4-32.

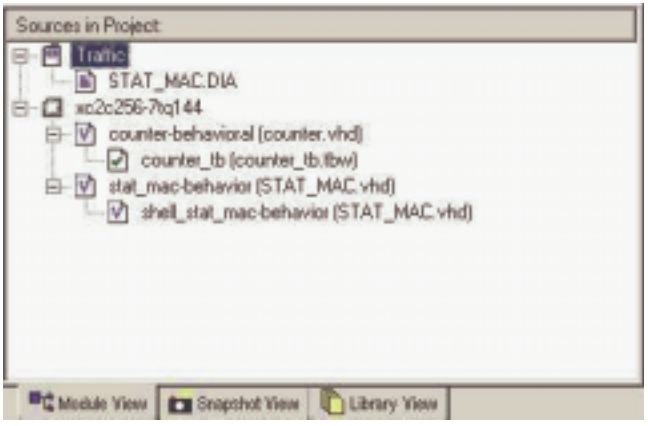


FIGURE 4-32: SOURCES IN PROJECT WINDOW

ECS HINTS

The ECS schematic capture program is designed around you selecting the action you wish to perform, followed by the object on which the action will be performed.

In general, most Windows applications currently operate by selecting the object and then the action to be performed on that object.

Understanding this fundamental philosophy of operation makes learning ECS a much more enjoyable experience.

From the Project menu, select New Source > Schematic and give it the name "top_sch."

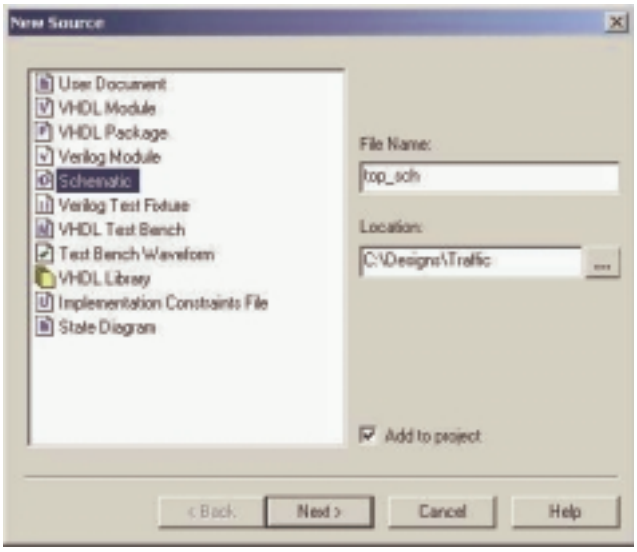


FIGURE 4-33: NEW SOURCE WINDOW SHOWING TOP_SCH

Click the Next> button, then the Finish button.

The ECS Schematic Editor window will now appear.

Back in the Project Navigator, highlight "counter.vhd" in the Sources window.

In the Process window, double-click on Create Schematic Symbol from the Design Entry Utilities section. This will create a schematic symbol and add it to the library in the Schematic Editor.

Create another symbol – this time for the state machine – by highlighting "stat_mac.vhd" and double-clicking on Create Schematic Symbol.

Returning to the Schematic editor, the symbol libraries can be found under the Symbol tab on the left-hand side of the page.

Add the counter and state machine by clicking on the new library in the Categories window at the top right of the ECS page and then selecting Counter.

Move the cursor over the sheet and drop the counter symbol by clicking where it should be placed.

Move the cursor back into the Categories window and place the "stat_mac" symbol on the sheet.

Zoom in using the button so your zoom button and the window looks like Figure 4-34.

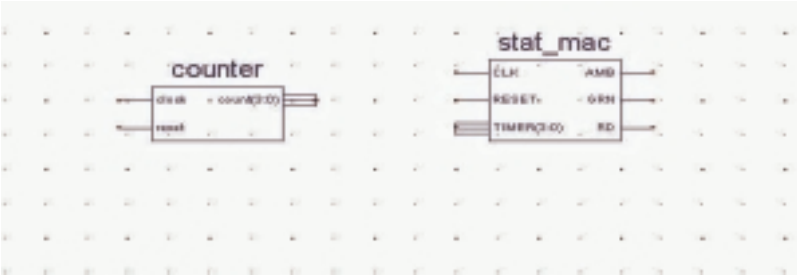


FIGURE 4-34: CLOSE-UP OF COUNTER AND STATE MACHINE SYMBOLS

Select the Add Wire tool from the Drawing toolbar.

To add a wire between two pins, click once on the symbol pin, once at each vertex, and once on the destination pin.

ECS will let you decide whether to use the Autorouter or manually place the signals on the page.

To add a hanging wire, click on the symbol pin to start the wire once at each vertex. Then double-click at the location where you want the wire to terminate.

Wire up the counter and state machine as shown in Figure 4-35:

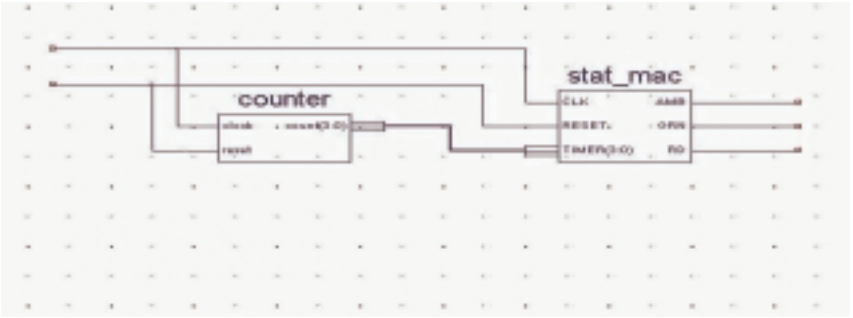


FIGURE 4-35: COUNTER AND STATE MACHINE SYMBOLS WITH WIRE

Select the Add Net Names tool from the Drawing toolbar.

Type "clock" (notice that the text appears in the window in the top left-hand side of the ECS page) and then place the net name on the end of the clock wire.

To add net names to wires that will be connected to your FPGA/CPLD I/Os, place the net name on the end of the hanging wire.

Finish adding net names so that your schematic looks similar to Figure 4-36.

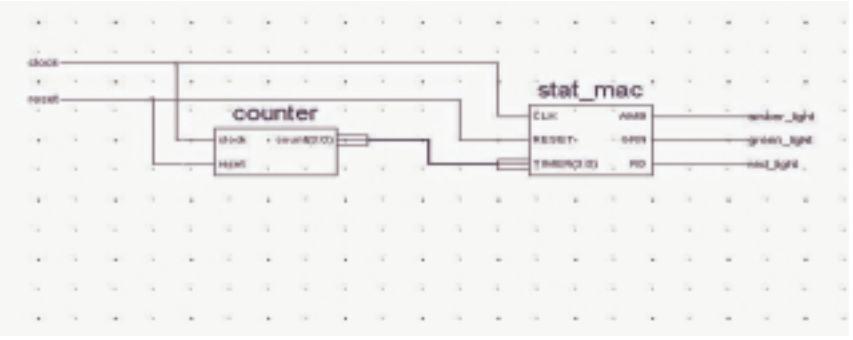


FIGURE 4-36: MORE NET NAMES

ECS recognizes that count(3:0) and TIMER(3:0) are buses, and so connects them together with a bus rather than a single net.

I/O MARKERS

Select the Add I/O Marker tool from the Drawing toolbar.

With the Input type selected, click and drag around all the inputs to which you want to add input markers.

Repeat for the outputs but select Output type.

Your completed schematic should look like Figure 4-37.

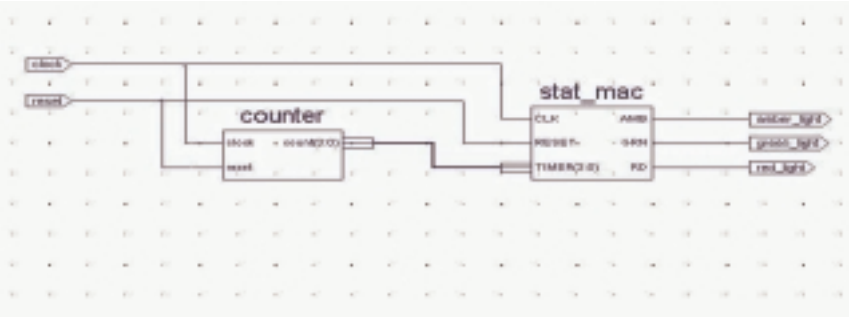


FIGURE 4-37: ADDING I/O MARKERS

Save the design and exit the Schematic editor.

(In the Design Entry utilities, you can view the VHDL created from the schematic when "top_sch" is selected in the Sources window. The synthesis tool actually works from this file.)

You can now simulate the entire design.

Highlight "top_sch.sch" in the Sources window.

Add a new testbench waveform source by right-clicking on "top_sch.sch" and selecting New Source. Call this source "top_sch_tb" and associate it with "top."

Accept the timing in the Initialize Timing dialog box and click OK.

In the waveform diagram, enter the input stimulus as follows:

Set the RESET cell below CLK cycle 1 to a value of "1."

Click the RESET cell below CLK cycle 2 to reset it low.

Go to the 64th clock cycle, right-click and select Set End of Testbench.

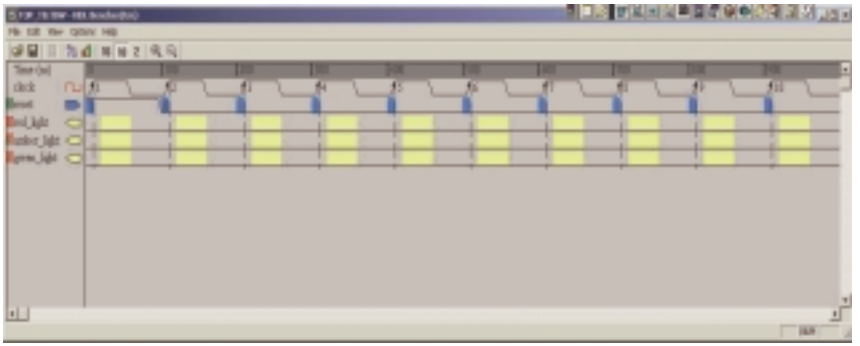


FIGURE 4-38: ***Waveform Diagram***

Close the Edit Test Bench window.

Click the Save Waveform button.

Close the HDL Bencher tool.

With "top_sch_tb.tbw" selected in the Sources window, expand ModelSim simulator and double-click Simulate Behavioral VHDL Model in the Process window.

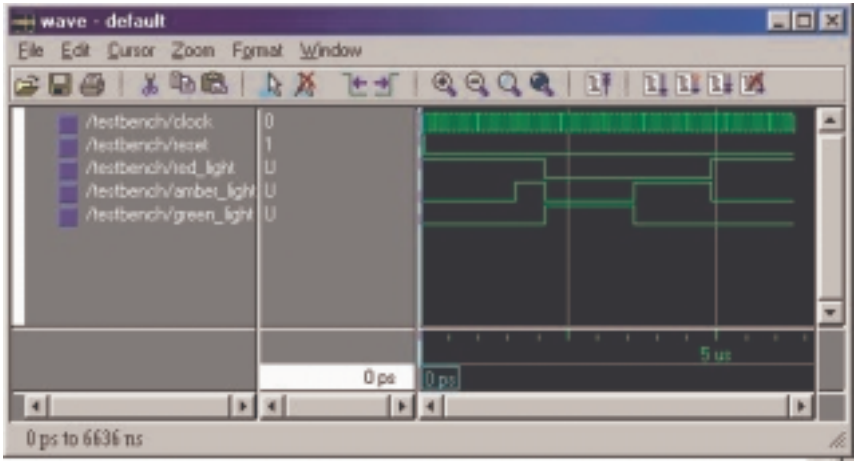


FIGURE 4-39: MODELSIM SIMULATION WINDOW

You are now ready to go to the implementation stage.

CHAPTER 5

Implementing CPLDs

Introduction

After you have successfully simulated your design, the synthesis stage converts the text-based design into an NGC netlist file.

The netlist is a non-readable file that describes the actual circuit to be implemented at a very low level.

The implementation phase uses the netlist and a constraints file to recreate the design using the available resources within the CPLD.

Constraints may be physical or timing and are commonly used for setting the required frequency of the design or declaring the required pinout.

The first step is translate. This step checks the design and ensures that the netlist is consistent with the chosen architecture. Translate also checks the UCF for any inconsistencies. In effect, this stage prepares the synthesized design for use within a CPLD.

The fit stage distributes the design to the resources in the CPLD and places those resources according to the constraints specified. Obviously, if the design is too big for the chosen device, the fit process will not be able to complete its job.

The fitter also uses the UCF file to understand timing and may sometimes decide to change the actual design.

For example, sometimes the fitter will change the D-Type flip-flops in the design to Toggle Type or T-Type registers. It all depends on how well the design converts into product terms.

Once the fitter has completed, it is good practice to re-simulate. As all the logic delays added by the macrocells, switch matrix, and flip-flops are known, MXE can use information for timing simulation.

The fitter creates a JEDEC file, which is used to program the device either on the board via a parallel cable or using programming equipment.

The steps of implementation must be carried out in this order.

WebPACK ISE software will automatically perform the steps required if a particular step is selected.

For example, if the design has only just been functionally simulated and you decide to do a timing simulation, the software will automatically synthesize, translate, and tit the design. It will then generate the timing information before it opens MXE and gives the timing simulation results.

The rest of this chapter demonstrates the steps required to successfully implement our traffic light design.

Synthesis

The XST synthesis tool will only attempt to synthesize the file highlighted in the Sources window.

In our traffic light design, "top.vhd" (for VHDL designs) or "top_sch" (for schematic designs) instantiates two lower level blocks, "stat_mac" and "counter."

The synthesis tool recognizes all the lower level blocks used in the top-level code and synthesizes them together to create a single bitstream.

In the Sources window, ensure that "top.vhd" (or "top_sch" for schematic flows) is highlighted.

In the Process window, expand the Synthesis subsection by clicking on the "+" next to Synthesize.

You can now check your design by double-clicking on Check Syntax.

Ensure that any errors in your code are corrected before you continue. If the syntax check is OK, a tick will appear.

The design should be okay because both the HDL Bencher tool and MXE simulator have already checked for syntax errors. (It is useful, when writing code, to periodically check your design for any mistakes using this feature.)

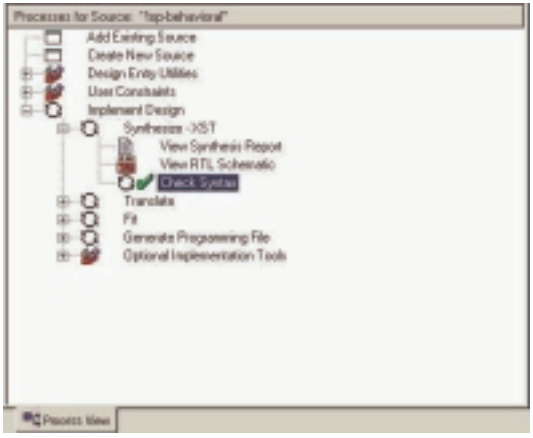


FIGURE 5-1: PROCESS WINDOW SHOWING CHECK SYNTAX

Right-click on Synthesize and select Properties.

A window appears allowing you to influence the way in which your design is interpreted.

The Help feature will explain each of the options in each tab. Click on the HDL Options tab.

In the Xilinx Specific Options tab, ensure that the Add IO Buffers box is ticked. The I/O buffers will be attached to all the port names in the top-level entity of the design.

Clicking on Help in each tab demonstrates the complex issue of synthesis and how the final result could change. The synthesis tool will never alter the function of the design, but it has a huge influence on how the design will perform in the targeted device.

Click OK in the Process Properties window and double-click on Synthesize.

When the synthesis is complete, a green tick will appear next to Synthesize. Double-click on View Synthesis Report.

Constraints Editor

To get the ultimate performance from the device, you must tell the implementation tools what and where performance is required.

This design is particularly slow and timing constraints are unnecessary.

Constraints can also be physical; pin locking is a physical constraint.

For this design, assume that the specification for clock frequency is 100 MHz and that the pin-out has been pre-determined to that of a CoolRunner-II pre-designed board.

In the Source window, add a New Source of type Implementation Constraints File. Call this file "top_constraints "and associate it with the module "top."

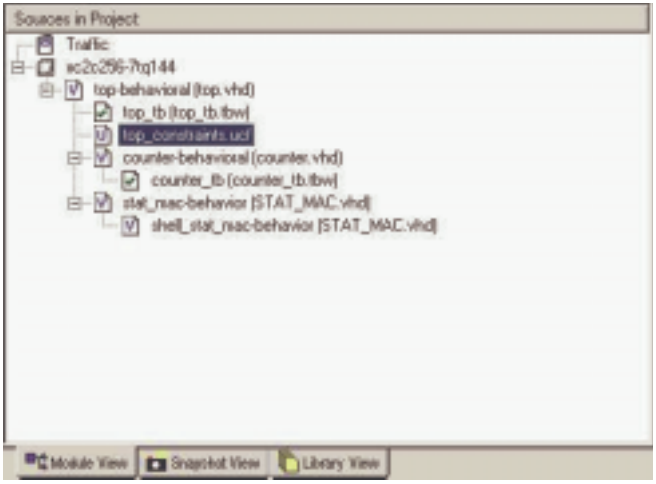


FIGURE 5-2: CONSTRAINTS FILE AS A SOURCE

In the Process window, expand the User Constraints section and double-click on Assign Package Pins.

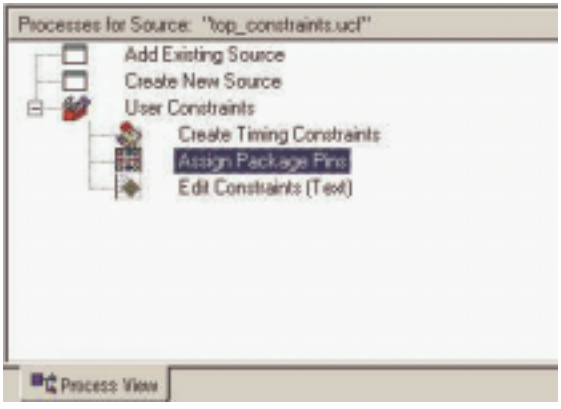


FIGURE 5-3: ASSIGN PACKAGE PINS IN PACE TOOL

Notice that the translate step in the Implement Design section runs automatically. This is because the implementation stage must see the netlist before it can offer you the chance to constrain sections of the design.

When translate has completed, the Xilinx PACE (Pinout Area and Constraints Editor) tool opens.

If there are already constraints in the UCF file, these will be imported by PACE and displayed. As we have an empty UCF file, nothing exists for PACE to import.

In the Design Object List, you can enter a variety of constraints on the I/O pins used in the design. PACE recognizes the five pins in the design and displays them in the list.

Click in the Loc area next to each signal and enter the following location constraints:

clock	p38
reset	p143
red_light	p11
green_light	p13
amber_light	p12

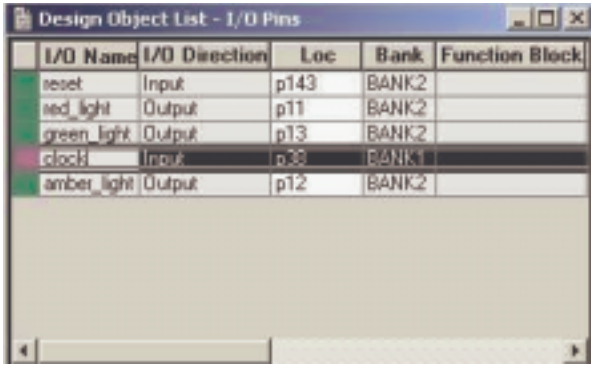


FIGURE 5-4: ENTER LOCATION CONSTRAINTS

When a pin is highlighted in the Design Object List, the pin to which it is "Loc'ed" is highlighted in the Package Pins view.

Save the PACE session and exit the PACE tool.

It is now possible to see the constraints in the UCF file.

With "top_constraints.ucf" highlighted in the Source window, double-click Edit Constraints (Text) in the Process window.

The UCF file will open in the main window of the ISE Project Navigator.

The constraints entered into PACE can be seen in Figure 5-5.

```
1  #PACE: Start of Constraints generated by PACE
2
3  #PACE: Start of PACE I/O Pin Assignments
4  NET "amber_light"  LOC = "p12"  ;
5  NET "clock"  LOC = "p38"  ;
6  NET "green_light"  LOC = "p13"  ;
7  NET "red_light"  LOC = "p11"  ;
8  NET "reset"  LOC = "p143"  ;
9
10 #PACE: Start of PACE Area Constraints
11
12 #PACE: Start of PACE Prohibit Constraints
13
14 #PACE: End of Constraints generated by PACE
15
```

FIGURE 5-5: TEXT CONSTRAINTS IMPORTED FROM PACE

To force a signal onto a global resource, you can apply the BUFG constraint. In this case, we will apply the BUFG constraint to the clock signal.

Enter the following syntax in the text file:

```
NET "clock" BUFG=CLK;
```

Save and close the text file.

The next step is to create timing constraints. With the UCF highlighted in the Source window, double-click on Create Timing Constraints in the Process window.

The Constraints Editor will open. This tool can be used to set location constraints, but for this tutorial it will only be used to create timing constraints.

The Constraints Editor recognizes the one global signal in the design.

DoubLe-click in the Period window of the global clock signal.

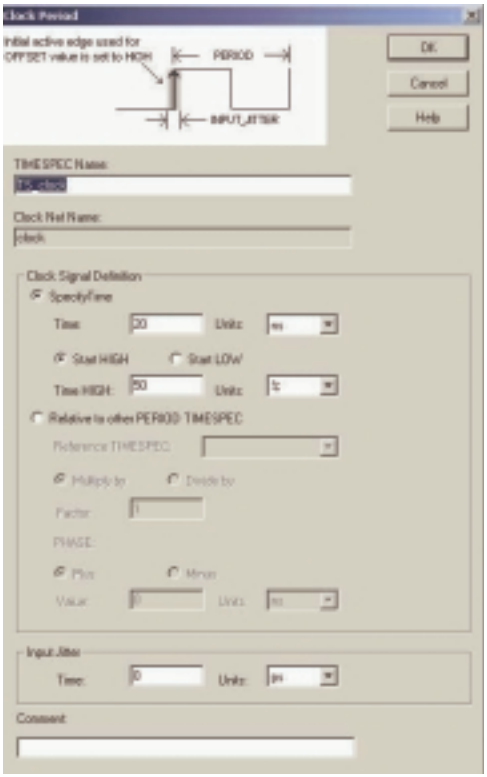


FIGURE 5-6: CLOCK PERIOD EDITOR WINDOW

In the Clock Period definition window, change the Time value to 10 ns. The duty cycle should stay at 50% high, 50% low.

The period constraint is now written into the UCF file and can be seen in the constraints list at the bottom of the Constraints Editor.

A period constraint ensures that the internal paths starting and ending at synchronous points (flip-flop, latch) have a logic delay less than 10 ns.

Click the Ports tab in the Constraints Editor. As there were already constraints in the UCF, they have been imported.

Highlight the three outputs "red_light," "green_light," and "amber_light" using ctrl select.

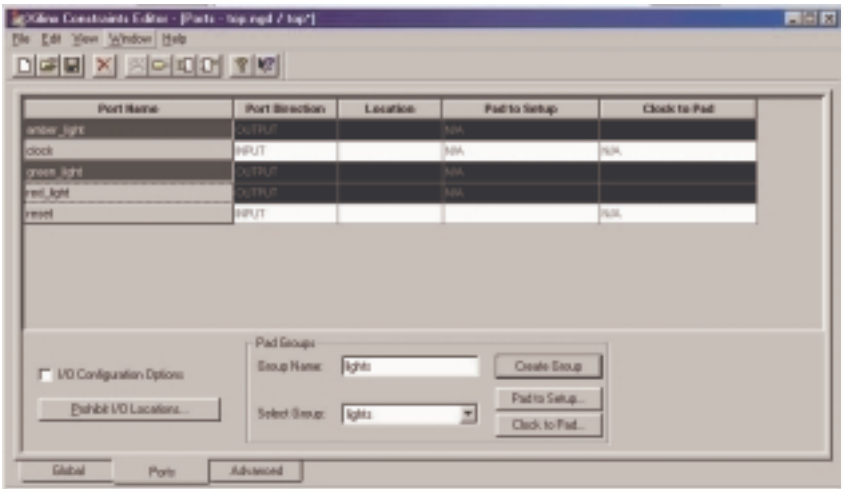


FIGURE 5-7: CONSTRAINTS EDITOR – CREATE GROUP

In the Group Name field, type "lights" and then click the Create Group button.

In the Select Group box, select "lights" and click the Clock to Pad button.

In the Clock to Pad dialog box, set the Time Requirement to 15 ns relative to the clock. (There is only one clock, but in some designs there may be more).

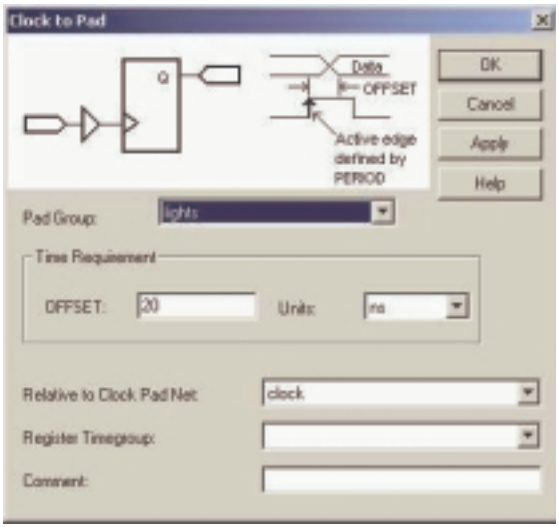


FIGURE 5-8: CLOCK TO PAD DIALOG BOX

Click OK.

Notice that the Clock to Pad fields have been filled in automatically and that the UCF generated has appeared in the UCF constraints tab at the bottom of the screen.

The UCF file should look similar to Figure 5-9.

```
1  #PACE: Start of Constraints generated by PACE
2  #PACE: Start of PACE I/O Pin Assignments
3  NET "amber_light" LOC = "p12";
4  NET "clock" LOC = "p38";
5  NET "green_light" LOC = "p13";
6  NET "red_light" LOC = "p11";
7  NET "reset" LOC = "p143";
8  #PACE: Start of PACE Area Constraints
9  #PACE: Start of PACE Prohibit Constraints
10 #PACE: End of Constraints generated by PACE
11 NET "clock" BUFG=CLK;
12 NET "clock" TNM_NET = "clock";
13 TIMESPEC "TS_clock" = PERIOD "clock" 10 ns HIGH 50 %;
14 INST "amber_light" TNM = "lights";
15 INST "green_light" TNM = "lights";
16 INST "red_light" TNM = "lights";
17 TIMEGRP "lights" OFFSET = OUT 15 ns AFTER "clock"  ;
18
```

FIGURE 5-9: COMPLETE CONSTRAINTS LIST

Save the Constraints Editor session and exit the Constraints Editor.

CoolRunner-II architecture supports the use of non 50:50 duty cycle clocks by implementing input hysteresis. This can be selected on a pin-by-pin basis.

For example, if the clock used in this design is an RC oscillator, the input hysteresis can be used to clean up the clock using the following constraint syntax:

```
NET "clock" schmitt_trigger;
```

The CoolRunner-II CPLD also supports different I/O standards.

If the three light signals had to go to a downstream device that required the signals to conform to a certain I/O standard, you could use the following constraint syntax:

```
NET "red_light" IOSTANDARD=LVTTL;
```

The permissible standards are LVTTL, LVCMOS15, LVCMOS18, LVCMOS25, LVCMOS33. On larger devices (128 macrocell and larger), the permissible standards are HSTL_I, SSTL2_I, and SSTL3_I.

However, you can use only one I/O standard **per bank**, so take care when assigning different I/O standards in a design.

The CoolRunner-II family has several features that are aimed at reducing power consumption in the device.

One of these features is known as CoolClock. The clock signal on Global Clock Input 2 (GCK2) is divided by 2 as soon as it enters the device.

All of the registers clocked by this clock are then automatically configured as dual-edge triggered flip-flops.

The highest toggling net in the design will now be toggling at half the frequency, which will reduce the power consumption of that net without compromising the performance of the design.

The CoolClock attribute can be applied by right-clicking on GCK2 in ChipViewer or by adding the following line in the UCF:

```
NET "clock" COOL_CLK;
```

However, we will not use these features in this tutorial.

For more information on the use of CoolRunner-II CPLDs, and their advanced features, visit *www.xilinx.com/apps/epld.htm* for a number of application notes, often including free code examples.

You must re-run translate so the new constraints can be read.

Click on the "+" next to Implement Design in the Process window.

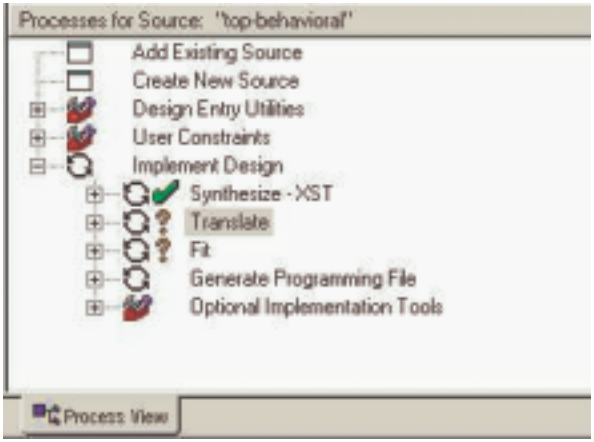


FIGURE 5-10: DESIGN PROCESS WINDOW

The implementation steps are now visible. An orange question mark indicates that translate is now out of date and should be re-run.

A right-click on Implement Design allows you to edit the properties for each particular step.

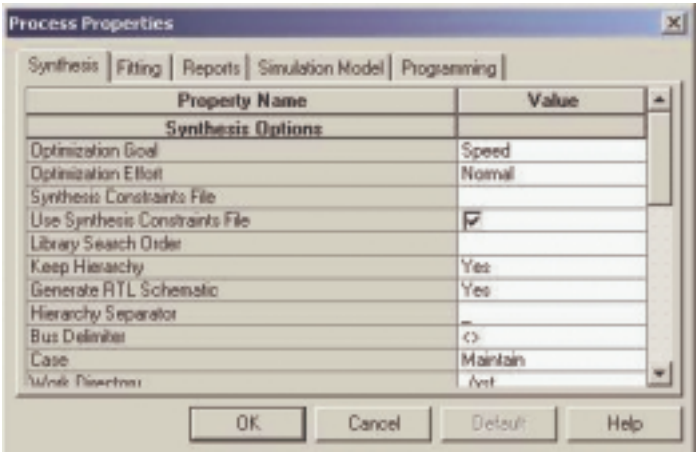


FIGURE 5-11: PROCESS PROPERTIES – IMPLEMENT DESIGN

The Help button will explain the operation of each field.

You can set the default I/O standard under the Fitting tab of the Process Properties window, as shown in Figure 5-11.

In this case, we will set the Output Voltage Standard to LVCMOS18 so that all of our pins are configured to be compliant with the LVCMOS 1.8V standard.

You can implement your design by double-clicking on Implement Design. When there is a green tick next to Implement Design, the design has completed the implementation stage.

The timing analysis is performed by default on the design.

To look at the Timing Report, expand the Optional Implementation Tools branch in the Process window. Then expand the Generate Timing branch and double-click on Timing Report.

CPLD Reports

Two reports are available that detail the fitting results and associated timing of the design. These are:

The Translation Report shows any errors in the design or the UCF.

The CPLD Fitter Report can be opened in two ways, either in a standard text window within the ISE GUI or in a browser window.

To select which format to open, go to Edit > Preferences > General > CPLD Fitter Report.

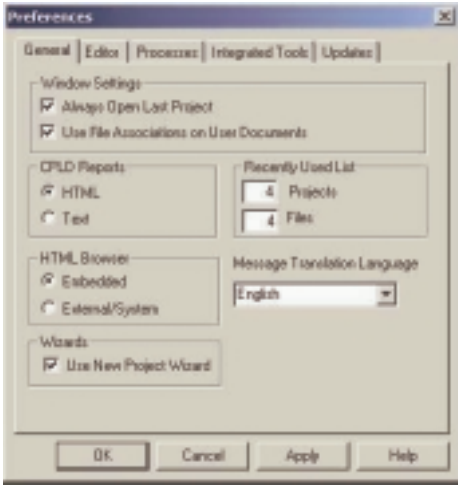


FIGURE 5-12: **ISE PREFERENCES**

To open the CPLD Fitter Report, expand the Fit branch and double-click on the Fitter Report Process.

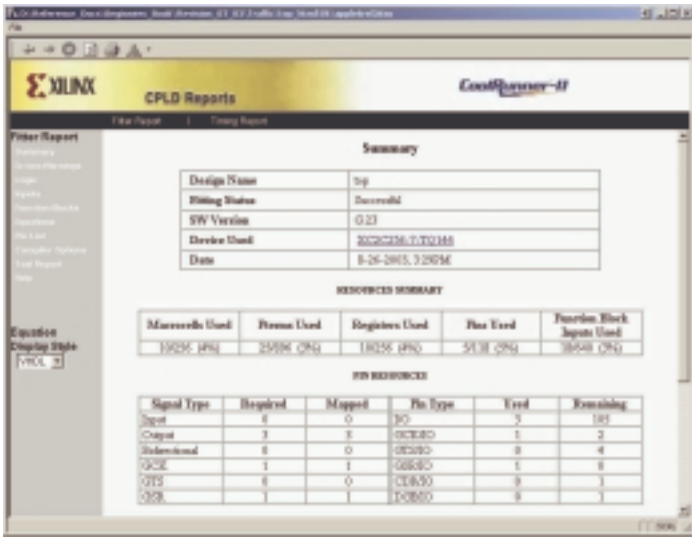


FIGURE 5-13: **CPLD HTML FITTER REPORT**

The same information is contained in both the HTML and text reports, but the HTML report has been designed to make the information more readable and easier to find.

You can browse through several sections of the HTML Fitter Report by using the menu on the left-hand side of the page.

The Summary section of the report gives a summary of the total resources available in the device (256 macrocells, 118 I/O pins, etc.), and how much is used by the design.

The errors and warnings generated during fitting can be seen in the Errors and Warnings section.

The Mapped Inputs and Mapped Logic sections give information about signals, macrocells, and pins in the fitted design. The key to the meaning of the abbreviations is available by pressing the legend button.

The Function Block Summary looks into each function block and shows which macrocell is used to generate the signals on the external pins.

By clicking on a specific function block (e.g., FB1) in the Function Blocks section, all of the macrocells in that function block will be shown.

Clicking on a specific macrocell will bring up a diagram of how that macrocell is configured.

An XC2C256 device has 16 function blocks, of which only two have been used for logic functions in this design.

The design could be packed into a single function block, but the chosen I/O pins dictate which macrocells (and hence which function blocks) are used.

A great feature of CPLDs is the deterministic timing, as a fixed delay exists per macrocell.

The Timing Report is able to give the exact propagation delays and setup times and clock-to-out times.

These values are displayed in the first section of the report you will have created.

The next section lists the longest setup time, cycle time (logic delay between synchronous points as constrained by the period constraint), and clock-to-out time.

The setup and clock-to-out times don't strictly affect the design's performance. These parameter limitations are dependent on the upstream and downstream devices on the board.

The cycle time is the maximum period of the internal system clock. The report shows that this design has a minimum cycle time of 7.1 ns, or 140 MHz.

The next section shows all the inputs and outputs of the design and their timing relationship with the system clock. Three lights will have a 6.0 ns delay with respect to the clock input.

The clock to setup section details the internal nets to and from a synchronous point. The maximum delay in this section dictates the maximum system frequency.

"amber_light", "red_light" and "green_light" are the D-Type flip-flops used to register the outputs.

The last section details all the path type definitions, explaining the difference between the types mentioned previously in the report.

To generate a detailed timing report, right-click on Generate Timing in the Process window and select Properties > Timing Report Format > Detail.

Timing Simulation

The process of timing simulation is very similar to the functional method.

With "top_tb.vhd" (or "top_sch_tb.vhd" for schematic flow) selected in the Sources window, expand the ModelSim simulator section in the Process window and right-click on Simulate Post Fit VHDL Model.

Select Properties. In the Simulation Run Time field, type "all."

Click OK, then double-click on Simulate Post Fit VHDL Model.

MXE will open, but this time implementing a different script file and compiling a post-route VHDL file (time_sim.vhd) is compiled.

Time_sim.vhd is a very low-level VHDL file generated by the implementation tools. It references the resources within the CPLD and takes timing information from a separate file.

Use the zoom features and cursors to measure the added timing delays.

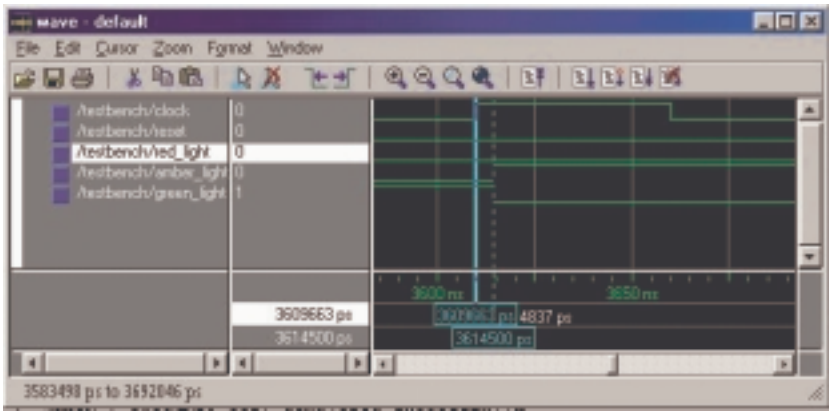


FIGURE 5-14: SIMULATION WAVEFORM

Configuration

A DLC7 Parallel-IV JTAG cable is required to configure the device from the iMPACT programmer.

Ensure that the cable is plugged in to the computer and that the ribbon cable/flying leads are connected properly to the board.

You must also connect the power jack of the Parallel-IV cable to either the mouse or keyboard PS2 port of the PC.

With "top.vhd" highlighted in the Source window, double-click on Configure Device (iMPACT) in the Process window.

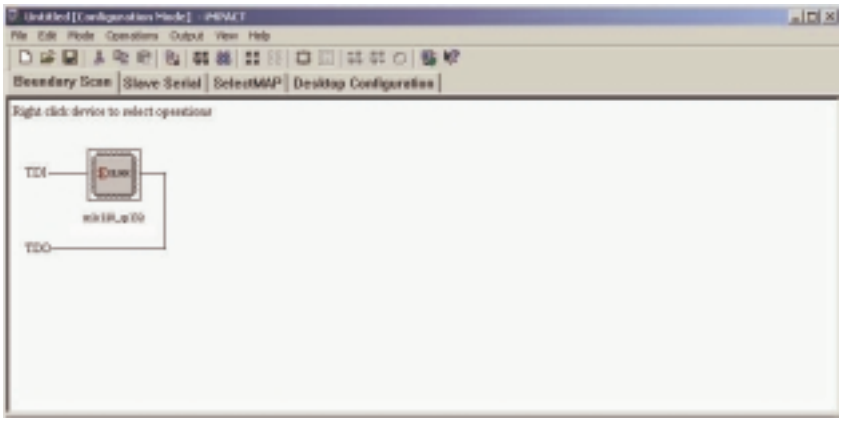


FIGURE 5-15: iMPACT PROGRAMMER MAIN WINDOW

Right-click on the Xilinx XC2C256 icon that appears in the iMPACT window and select Program.

The design will now download into the device.

You have now successfully programmed your first CoolRunner-II CPLD.

CHAPTER 6

Implementing FPGAs

Introduction

After you have successfully simulated your design, the synthesis stage converts the text-based HDL design into an NGC netlist file.

The netlist is a non-readable file that describes the actual circuit to be implemented at a very low level.

The implementation phase uses the netlist and a constraints file to recreate the design using the available resources within the FPGA.

Constraints may be physical or timing and are commonly used for setting the required frequency of the design or declaring the required pin-out.

The map stage distributes the design to the resources available in the FPGA. Obviously, if the design is too big for the specified device, mapping will be incomplete.

The map stage also uses the UCF file to understand timing and may sometimes decide to add further logic (replication) to meet the given timing requirements.

Map has the ability to "shuffle' the design around LUTs to create the best possible implementation for the design. The whole process is automatic and requires little user input.

The place and route stage works with the allocated CLBs and chooses the best location for each block.

For a fast logic path, it makes sense to place relevant CLBs next to each other simply to minimize the path length. The routing resources are then allocated to each connection, again using a careful selection of the best possible routing types.

For example, if you need a signal for many areas of the design, the place and route tool would use a "longline" to span the chip with minimal delay or skew.

At this point, it is good practice to re-simulate. As all of the logic delays added by the LUTs and flip-flops are now known (as well as the routing delays), MXE can use this information for timing simulation.

Finally, a program called "bitgen" takes the output of place and route and creates a programming bitstream.

When developing a design, it may not be necessary to create a bit file on every implementation, as you may only need to ensure that a particular portion of your design passes timing verification.

The steps of implementation must be carried out in this order:

1. Synthesize
2. Fit
3. Timing Simulate
4. Program.

WebPACK ISE software will automatically perform the steps required if a particular step is selected.

For example, if the design has only just been functionally simulated and you decide to do a timing simulation, the software will automatically synthesize and fit. It will then generate the timing information before it opening MXE and giving timing simulation results.

In this chapter, we'll demonstrate the steps required to successfully implement our traffic light design into a Spartan-3 FPGA.

Double-click on "xc2c256-7tq144 – XST VHDL" in the Sources window, shown in Figure 6-1.

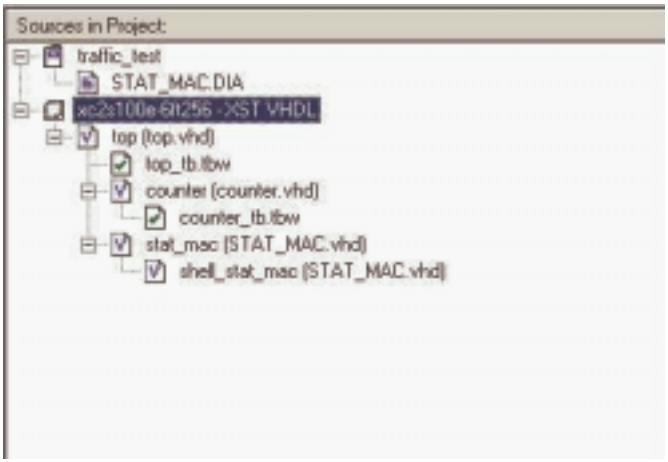


FIGURE 6-1: SOURCES IN PROJECT WINDOW

Enter the following characteristics:

```
Change the Device Family to Spartan3
In the Device field, select xc3s50
Change the Package field to tq144
Enter the Speed Grade as -4
Top Level Module HDL
Synthesis Tool XST (VHDL/Verilog)
Simulator ModelSim
Generated Simulation Language VHDL

Click on OK.
```

The project, originally targeted at a CoolRunner-II CPLD, is now targeting a Xilinx Spartan-3 FPGA.

The green ticks in the Process window should have disappeared and been replaced by orange question marks, indicating that the design must be re-synthesized and re-implemented.

Synthesis

The XST synthesis tool will only attempt to synthesize the file highlighted in the Source window.

In our traffic light design, "top.vhd" (for VHDL designs) or "top_sch" (for schematic designs) instantiates two lower level blocks, "stat_mac" and "counter."

The synthesis tool recognizes all the lower level blocks used in the top-level code and synthesizes them together to create a single netlist.

In the Sources window, ensure that "top.vhd" (or "top_sch" for schematic flows) is highlighted.

In the Process window, expand the Synthesis subsection by clicking on the "+" next to Synthesize.

You can now check your design by double-clicking on Check Syntax.

Ensure that any errors in your code are corrected before you continue. If the syntax check is OK, a tick will appear (as shown in Figure 6-2).

The design should be okay because both the Bencher and MXE have already checked for syntax errors. (It is useful, when writing code, to periodically check your design for any mistakes using this feature.)

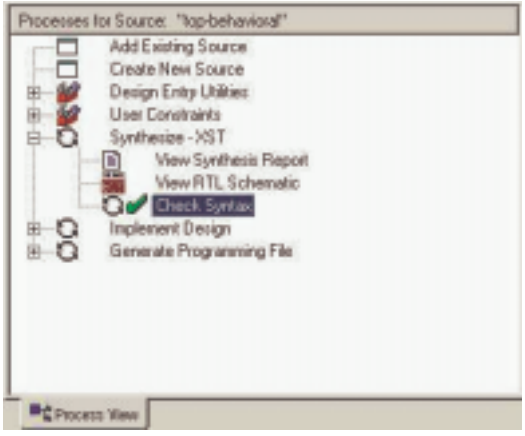


FIGURE 6-2: PROCESSES WINDOW SHOWING CHECK SYNTAX HAS COMPLETED SUCCESSFULLY

Right-click on Synthesize and select Properties.

A window will appear allowing you to influence the way in which your design is interpreted.

The Help feature explains each of the options in each tab.

Click on the HDL Options tab.

The FSM encoding algorithm option looks for state machines and determines the best method of optimizing.

For FPGAs, state machines are usually "one hot" encoded. This is because of the abundance of flip-flops in FPGA architectures.

A one hot encoded state machine will use one flip-flop per state. Although this may seem wasteful, the next state logic is reduced, and the design is likely to run much faster.

Leave the setting on "auto" to achieve this fast one hot encoding.

In the Xilinx Specific Options tab, ensure that the Add I/O Buffers box is ticked.

The I/O buffers will be attached to all of the port names in the top-level entity of the design.

Clicking on Help in each tab demonstrates the complex issue of synthesis and how the final result could change.

The synthesis tool will never alter the function of the design, but it has a huge influence on how the design will perform in the targeted device.

Click OK in the Process Properties window and double-click on Synthesize.

The first section of the report summarizes just the synthesis settings.

Each entity in the design is then compiled and analyzed.

The next section in the report gives synthesis details and documents how the design was interpreted.

Note that the state machine is one hot encoded, as each state name (red, amber, redamb, and green) has been assigned its own 1-bit register.

When synthesis chooses to use primitive macros it is known as "inference." As registered outputs were selected in the state machine, three further registers were inferred.

```
=========================================================================
*                            HDL Synthesis                              *
=========================================================================

Synthesizing Unit <shell_stat_mac>.
    Related source file is D:/Book/traffic_test/STAT_MAC.vhd.
    Found 1-bit register for signal <amb>.
    Found 1-bit register for signal <grn>.
    Found 1-bit register for signal <rd>.
    Found 1-bit register for signal <amber>.
    Found 1-bit register for signal <green>.
    Found 1-bit register for signal <red>.
    Found 1-bit register for signal <redamb>.
    Summary:
   inferred   7 D-type flip-flop(s).
Unit <shell_stat_mac> synthesized.


Synthesizing Unit <stat_mac>.
    Related source file is D:/Book/traffic_test/STAT_MAC.vhd.
Unit <stat_mac> synthesized.
```

FIGURE 6-3: EXTRACT OF SYNTHESIS REPORT

The Final Report section shows the resources used within the FPGA.

```
Cell Usage :
# BELS                          : 33
#      GND                      : 1
#      LUT1                     : 1
#      LUT1_D                   : 1
#      LUT1_L                   : 2
#      LUT2_D                   : 1
#      LUT2_L                   : 2
#      LUT3                     : 3
#      LUT3_L                   : 1
#      LUT4                     : 6
#      LUT4_L                   : 6
#      MUXCY                    : 3
#      MUXF5                    : 2
#      VCC                      : 1
#      XORCY                    : 3
# FlipFlops/Latches             : 10
#      FD                       : 6
#      FDC                      : 4
# Clock Buffers                 : 1
#      BUFGP                    : 1
# IO Buffers                    : 4
#      IBUF                     : 1
```

FIGURE 6-4: RESOURCE REPORT

The Constraints File

To get the ultimate performance from the device, you must tell the implementation tools what and where performance is required.

This design is particularly slow and timing constraints are unnecessary.

Constraints can also be physical; pin locking is a physical constraint.

For this design, assume that the specification for clock frequency is 100 MHz and that the pin-out has been pre-determined to that of a Spartan-3 device.

There are already some constraints in the UCF from the previous project implementation. It will be necessary to delete these constraints.

Highlight "top_constraints.ucf" in the Source window. Expand the "+" next to User Constraints and double-click Edit Constraints (Text).

Highlight all of the constraints and delete them. Save the UCF and close it.

Double-click on Assign Package Pins.

Alternatively, you can highlight the top level ("top.vhd") and expand the User Constraints branch.

FIGURE 6-5: PROCESS WINDOW SHOWING ASSIGN PACKAGE PINS

The PACE tool will be launched.

Assign all I/O pins in the Design Object List as follows.

reset	p36
red_light	p44
green_light	p52
clock	p55
amber_light	p46

Save and Exit the PACE session.

Double-click on Create Timing Constraints in the Process window, as seen above Assign Package Pins in Figure 6-5.

Notice that the Constraints Editor is invoked and picks up the LOC constraints entered in PACE.

These can be edited by double-clicking on them in the read-write window or under the **Ports tab i**n the Main window.

Double-click in the Period window of the global signal clock and enter a period of 10 ns.

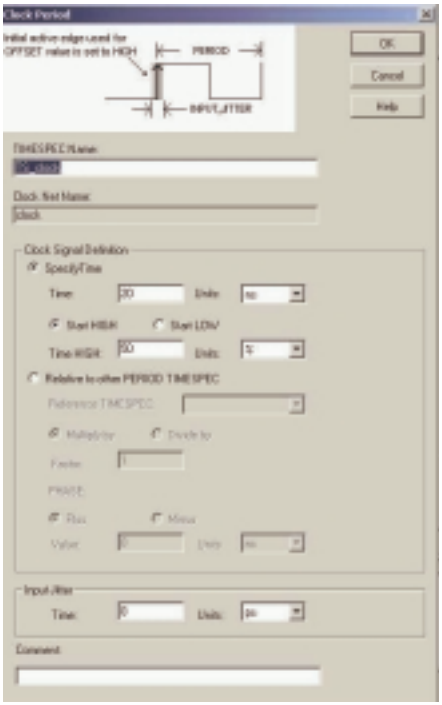


FIGURE 6-6: SPECIFY PERIOD CONSTRAINT

Click OK.

Click on the Ports tab in the Constraints Editor. As there were already constraints in the UCF, they have been imported.

Highlight the three outputs "red_light," "green_light," and "amber_light" using ctrl select.

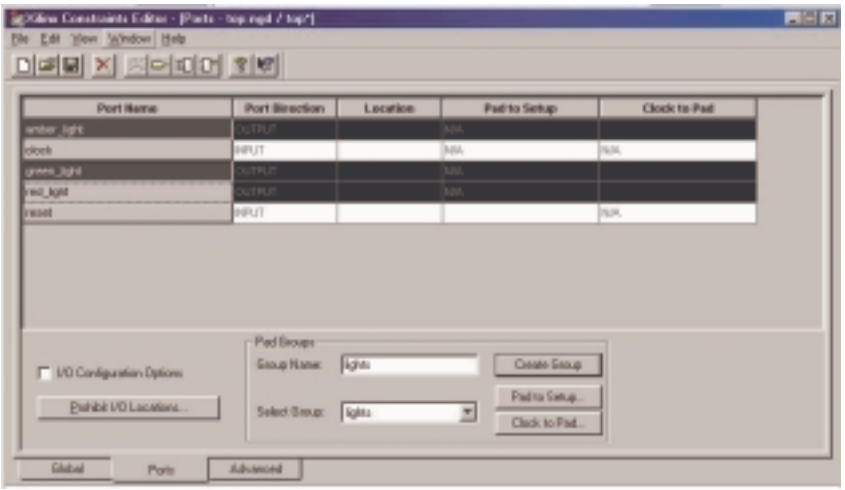


FIGURE 6-7: CONSTRAINTS EDITOR – CREATE GROUP

In the Group Name field, type "lights" and then hit Create Group.

In the Select Group box, select lights and hit the Clock to Pad button.

In the Clock to Pad dialog box, set the time requirement to 15 ns relative to the clock. There is only one clock, but in some designs there may be more.

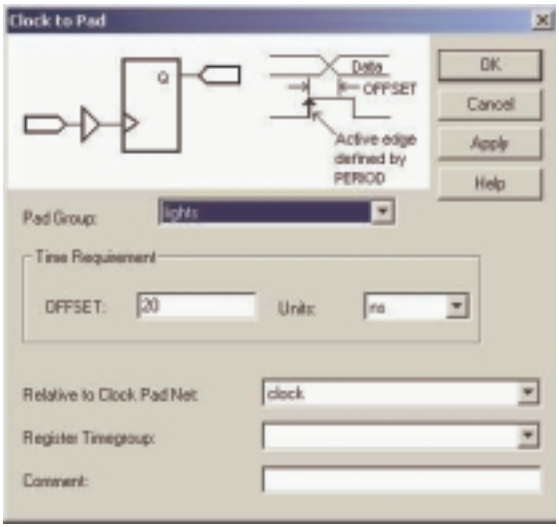


FIGURE 6-8: CLOCK TO PAD DIALOG BOX

Click OK.

Notice that the Clock to Pad fields have been filled in automatically. Also notice that the UCF generated has appeared in the UCF Constraints tab at the bottom of the screen.

The UCF file should look similar to Figure 6-9.

```
1  #PACE: Start of Constraints generated by PACE
2  #PACE: Start of PACE I/O Pin Assignments
3  NET "amber_light" LOC = "p46";
4  NET "clock" LOC = "p55";
5  NET "green_light" LOC = "p52";
6  NET "red_light" LOC = "p44";
7  NET "reset" LOC = "p36";
8  #PACE: Start of PACE Area Constraints
9  #PACE: Start of PACE Prohibit Constraints
10 #PACE: End of Constraints generated by PACE
11 NET "clock" TNM_NET = "clock";
12 TIMESPEC "TS_clock" = PERIOD "clock" 10 ns HIGH 50 %;
13 INST "amber_light" TNM = "lights";
14 INST "green_light" TNM = "lights";
15 INST "red_light" TNM = "lights";
16 TIMEGRP "lights" OFFSET = OUT 15 ns AFTER "clock"  ;
17
```

FIGURE 6-9: COMPLETE CONSTRAINTS FILE

Save and close the Constraints Editor session.

Click on the "+" next to Implement Design in the Process window.

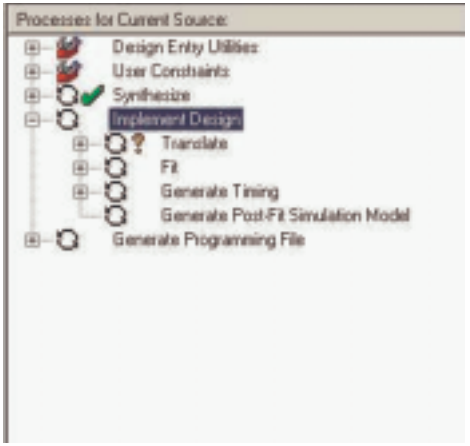


FIGURE 6-10: PROCESS WINDOW SHOWING IMPLEMENT DESIGN

Implement the design by double-clicking on Implement Design. (You could run each stage separately if required.)

When there is a green tick next to Translate, Map, and Place and Route, your design has completed the implementation stage.

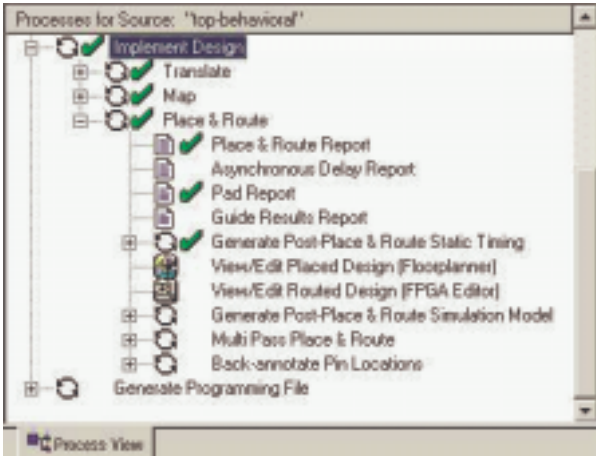


FIGURE 6-11: COMPLETED IMPLEMENTATION

A green tick means that the design ran through without any warnings.

A yellow exclamation point may mean that there is a warning in one of the reports.

A common warning, which can be safely ignored in CPLD designs, is that an "fpga_don't_touch" attribute has been applied to an instance.

If you've followed the design procedure outlined in this example, there should be no errors or warnings.

FPGA Reports

Each stage has its own report. Clicking on the "+" next to each stage lists the reports available:

1. The Translate Report shows any errors in the design or the UCF.
2. The Map Report confirms the resources used within the device and describes trimmed and merged logic. It will also describe exactly where each portion of the design is located in the actual device.

A detailed Map Report can be chosen in the Properties for map.

3. The Post-Map Static Timing Report shows the logic delays only (no routing) covered by the timing constraints. This design has two timing constraints, the clock period and the clock-to-out time of the three lights.

If the logic-only delays don't meet timing constraints, the additional delay added by routing will only add to the problem.

Without a routing delay, these traffic lights would run at 216 MHz!

4. The Place and Route Report gives a step-by-step progress report.

The place and route tool must be aware of timing requirements. It will list the given constraints and report how comfortably the design fell within – or how much it failed – the constraints.

5. The Asynchronous Delay Report is concerned with the worst path delays in the design – both logic and routing.
6. The Pad Report displays the final pin-out of the design, with information regarding the drive strength and signalling standard.
7. The Guide Report shows how well a guide file has been met (if one was specified).

The Post Place and Route Static Timing Report adds the routing delays. Notice that the max frequency of the clock has dropped.

WebPACK ISE software has additional tools for complex timing analysis and floor planning, which are beyond the scope of this introductory book.

Programming

Right-click on Generate Programming file and click on Properties.

Under the Start-Up Options tab, ensure that the startup clock is set to JTAG Clock by selecting JTAG Clock from the drop-down menu.

Double-click on Generate Programming File.

This operation creates a .bit file that can be used by the iMPACT programmer to configure a device.

Expand the Generate Programming File tools subsection.

Double-click on Configure Device (iMPACT).

A DLC7 Parallel-IV JTAG cable is required to configure the device from the iMPACT Programmer.

Ensure that the cable is plugged in to the computer and that the ribbon cable/flying leads are connected properly to the board.

You must also connect the power jack of the Parallel-IV cable to either the mouse or keyboard port of the PC.

If the chain specified in the design is not automatically picked up from the ISE tool, right-click in the top half of the iMPACT window and select Add Xilinx Device.

Browse to the location of the project (c:\designs\traffic) and change the file type to .bit.

Open "top.bit" ("top_sch.bit" for schematic designs). The iMPACT Programmer has drawn a picture of the programming chain.

Click on the picture of the device.

From the Operations Menu, select Program.

Summary

This chapter has taken the VHDL or Schematic design through to a working physical device. The steps discussed were:

- Synthesis and Synthesis report
- Timing and Physical Constraints using the Constraints Editor
- The Reports Generated throughout the Implementation flow
- Timing Simulation
- Creating and Downloading a bitstream.

CHAPTER 7

Design Reference Bank

Introduction

Our final chapter contains a useful list of design examples and applications that will give you a good jump start into your future programmable logic designs.

We selected the application examples from a comprehensive list of application notes available from the Xilinx website, as well as extracts from the Xilinx quarterly magazine, the *Xcell Journal* (To subscribe, click on "Subscribe to *Xcell Journal*" at *www.xilinx.com/publications/xcellonline/*).

This section will also give you pointers on where to look for and download code and search for IP from the Xilinx website.

Get the Most out of Microcontroller-Based Designs

Microcontrollers don't make the world go round, but they most certainly help us get around in the world. You can find microcontrollers in automobiles, microwave ovens, automatic teller machines, VCRs, point-of-sale terminals, robotic devices, wireless telephones, home security systems, and satellites, to name just a few applications.

In the never-ending quest for faster, better, and cheaper products, advanced designers are now pairing CPLDs with microcontrollers to take advantage of the strengths of each.

Microcontrollers are naturally good at sequential processes and computationally intensive tasks, as well as a host of non-time-critical tasks.

CPLDs such as Xilinx CoolRunner devices are ideal for parallel processing, high-speed operations, and applications where lots of inputs and outputs are required.

Although faster and more powerful microcontrollers do exist, 8-bit microcontrollers own much of the market because of their low cost and low power characteristics.

The typical operational speed is around 20 MHz, but some microcontroller cores divide clock frequency internally and use multiple clock cycles per instruction (operations often include fetch-and-execute instruction cycles).

Thus, with a clock division of 2 – with each instruction taking as long as three cycles – the actual speed of a 20 MHz microcontroller is divided by 6. This works out to an operational speed of only 3.33 MHz.

CoolRunner CPLDs are much, much faster than microcontrollers and can easily reach system speeds in excess of 100 MHz. Today, we are even seeing CoolRunner devices with input-to-output delays as short as 3.5 ns, which equates to impressive system speeds as fast as 285 MHz.

CoolRunner CPLDs make ideal partners for microcontrollers, because they not only can perform high-speed tasks, they can perform those tasks with ultra-low power consumption.

Xilinx offers free software and low-cost hardware design tools to support CPLD integration with microcontrollers.

The Xilinx CPLD design process is quite similar to that used on microcontrollers, you can quickly learn how to partition your designs across a CPLD and microcontroller to maximum advantage.

So far, a design partition over a microcontroller and a CPLD sounds good in theory, but will it work in the field?

We will devote the rest of this chapter to design examples that show how you can enhance a typical microcontroller design by utilizing the computational strengths of the microcontroller and the speed of a CoolRunner CPLD.

CONVENTIONAL STEPPER MOTOR CONTROL

A frequent use of microcontrollers is to run stepper motors. Figure 7-1 depicts a typical four-phase stepper motor driving circuit. The four windings have a common connection to the motor supply voltage (Vss), which typically ranges from 5V to 30V.

A high-powered NPN transistor drives each of the four phases. (Incidentally, MOSFETs can also be used to drive stepper motors.)

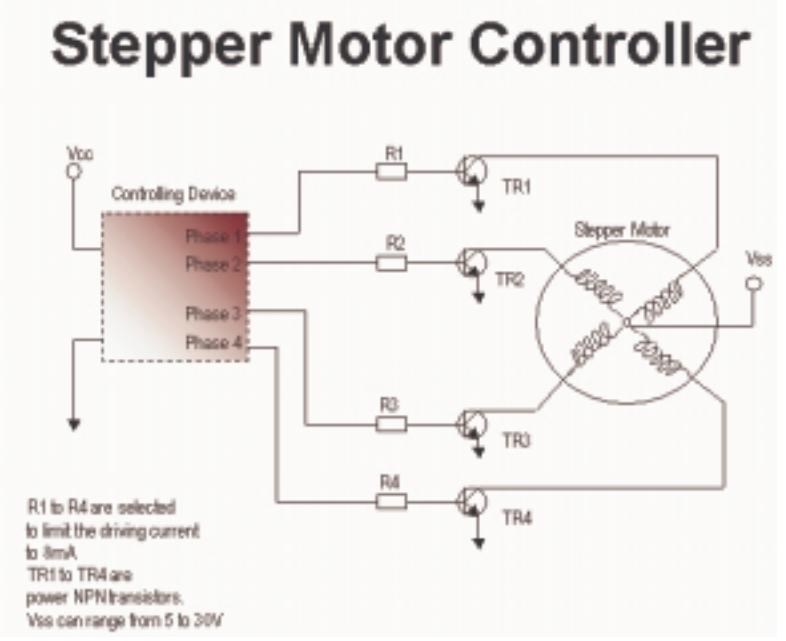


FIGURE 7-1: STEPPER MOTOR CONTROLLER

Each motor phase current may range from 100 mA to as much as 10A. The transistor selection depends on the drive current, power dissipation, and gain.

The series resistors should be selected to limit the current to 8 mA per output to suit either the microcontroller or CPLD outputs. The basic control sequence of a four-phase motor is achieved by activating one phase at a time.

At the low-cost end, the motor rotor rotates through 7.5 degrees per step, or 48 steps per revolution.

More accurate, higher cost versions have a basic resolution of 1.8 degrees per step.

Furthermore, it is possible to half-step these motors to achieve a resolution of 0.9 degrees per step. Stepper motors tend to have a much lower torque than other motors, which is advantageous in precise positional control.

The examples that follow show how either a microcontroller or a CPLD can control stepper motor tasks to varying degrees of accuracy. As Figure 7-2 illustrates, the design flow for both is quite similar.

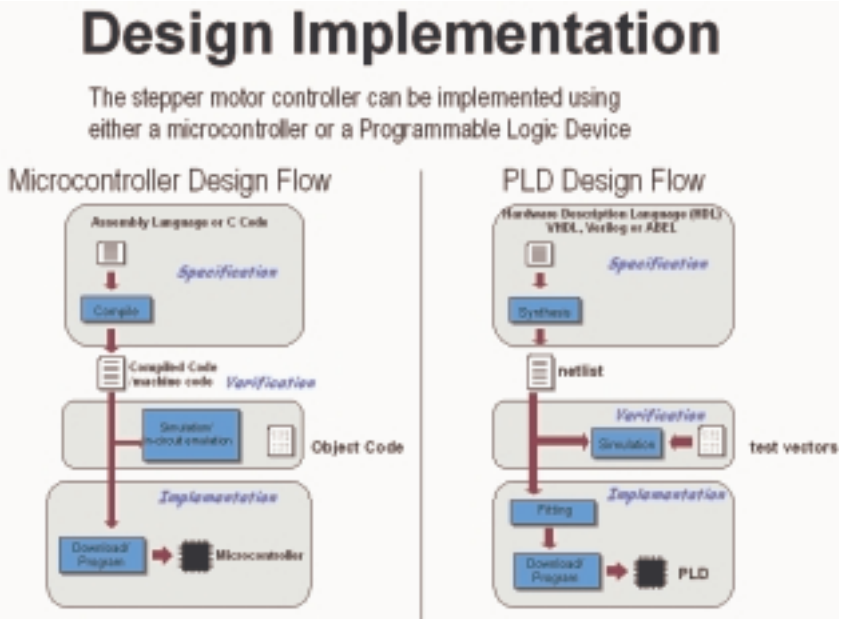


FIGURE 7-2: DESIGN FLOW COMPARISON

Both flows start with text entry. Assembly language targets microcontrollers; the ABEL hardware description language targets PLDs.

After entering the text "description," the design is either compiled (microcontroller) or synthesized (PLD). Next, the design is verified by some form of simulation or test.

Once verified, the design is downloaded to the target device – either a microcontroller or PLD. We can then program the devices in-system using an inexpensive ISP cable.

One of the advantages of a PLD over a microcontroller occurs during board-level testing. Using a JTAG Boundary Scan, the PLD can be fully tested on the board. The PLD can also be used as a "gateway" to test the rest of the board's functionality. After the board-level test is completed, the PLD can then be programmed with the final code in-system via the JTAG port.

Microcontrollers can include monitor debug code internal to the device for limited code testing and debugging. With the advent of flash-based microcontrollers, these can now also be programmed in-system.

USING A MICROCONTROLLER TO CONTROL A STEPPER MOTOR

Figure 7-3 shows assembly language targeting a Philips 80C552 microcontroller.

The stepper motor that the microcontroller will control has four sets of coils. When logic level patterns are applied to each set of coils, the motor steps through its angles.

The speed of the stepper motor shaft depends on how fast the logic level patterns are applied to the four sets of coils. The manufacturer's motor specification data sheet provides the stepping motor code.

A very common stepping code is given by the following hexadecimal numbers:

A 9 5 6

Each hex digit is equal to four binary bits:

1010 1001 0101 0110

These binary bits represent voltage levels applied to each of the coil driver circuits. The steps are:

1010	5V	0V	5V	0V
1001	5V	0V	0V	5V
0101	0V	5V	0V	5V
0110	0V	5V	5V	0V

If you send this pattern repeatedly, then the motor shaft rotates.

The assembly language program in Figure 7-3 continually rotates the stepper motor shaft. By altering the value of R0 in the delay loop, this will give fine control over speed; altering the value of R1 will give coarse variations in speed.

```
$MOD552                 ; include file for 80C552
        ORG 0           ; reset address
        SJMP START      ; jump over reserved area
        ORG 30H         ;program start address
START:  MOV P1,#0AH     ;move hex 0A into lower
                        ;4 bits of port 1
        ACAL DELAY      ;call subroutine step hold
                        ;delay
        MOV P1, #09H    ;move hex 09 into lower
                        ;4 bits of port 1
        ACALL DELAY
        MOV P1,#05H
        ACALL DELAY
        MOV P1, #06H
        ACALL DELAY
        SJMP START      ;repeat stepping pattern
                        ;
                        ;Double loop delay
DELAY:  MOV R1, #0FFH ;put hex FF into register 1
OUTER:  MOV R0, #0FFH ;put hex FF into register 0
INNER:  DJNZ R0, INNER;decrement r0 until it is 0
        DJNZ R1,OUTER;dec r1, go to outer until
                        ; r1 = 0
        RET             ;return from subroutine
        END             ;assembler directive
```

FIGURE 7-3: ASSEMBLY LANGUAGE PROGRAM TO ROTATE THE STEPPER MOTOR SHAFT

STEPPER MOTOR CONTROL USING A CPLD

Figure 7-4 shows a design written in ABEL hardware description language.

Within the Xilinx CPLD, four inputs are required to fully control the stepper motor. The CLK input synchronizes the logic and determines the speed of rotation. The motor advances one step per clock period.

The angle of rotation of the shaft will depend on the specific motor used. The direction (DIR) control input changes the sequence at the outputs (PH1 to PH4) to reverse the motor direction.

The enable input (EN) determines whether the motor is rotating or holding. The active low reset input (RST) initializes the circuit to ensure that the correct starting sequence is provided to the outputs.

```
-- Stepper Motor Controller
        library IEEE;
        use IEEE.std_logic_1164.all;

entity step1 is
```

```
port (
  clk : in std_logic; -- input to determine speed of rotation
  rst : in std_logic; -- resets and initializes the circuit
  en : in std_logic; -- determines whether motor rotating or holding
  dir : in std_logic; -- motor direction control
  ph1 : inout std_logic; -- output to motor phase 1
  ph2 : inout std_logic; -- output to motor phase 2
  ph3 : inout std_logic; -- output to motor phase 3
  ph4 : inout std_logic -- output to motor phase 4
  );
end step1;

architecture equation of step1 is
begin
 Process (rst,clk)
 begin
    if rst = '0' then
     ph1 <= '1';
     ph2 <= '0';
     ph3 <= '0';
     ph4 <= '0';
    else
    if clk'event and clk='1' then
-- Stepper Motor Controller description equations
     ph1 <= (not(dir)and en and not(ph1)and ph2 and not(ph3)and
not(ph4))
     or (dir and en and not(ph1)and not(ph2)and not(ph3)and ph4)
     or ( not(en)and ph1);

     ph2 <= (not(dir)and en and not(ph1)and not(ph2)and ph3 and
not(ph4))
     or (dir and en and ph1 and not(ph2)and not(ph3)and not(ph4))
     or (not(en)and ph2);

     ph3 <= (not(dir)and en and not(ph1)and not(ph2) and not(ph3)and
ph4)
     or (dir and en and not(ph1) and (ph2) and not(ph3)and not(ph4))
     or (not(en)and ph3);

     ph4 <= (not(dir)and en and ph1 and not(ph2) and not(ph3)and
not(ph4))
     or (dir and en and not(ph1) and not(ph2)and ph3 and not(ph4))
     or (not(en) and ph4);
  end if;
  end if;
 end process;
end equation;
```

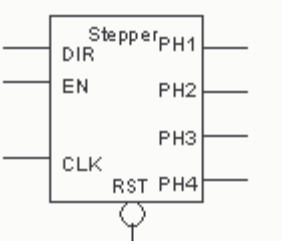


FIGURE 7-4: **CPLD ABEL PROGRAM TO CONTROL A STEPPER MOTOR**

The phase equations (PH1 to PH4) are written with a colon and equal sign (:=) to indicate a registered implementation of the combinatorial equation.

Each phase equation is either enabled (EN), indicating that the motor is rotating, or disabled (!EN), indicating that the current active phase remains on and the motor is locked.

The value of the direction input (DIR) determines which product term is used to sequence clockwise or counterclockwise. The asynchronous equations (for example, ph1.AR=!rst) initialize the circuit.

The ABEL hardware description motor control module can be embedded within a macro function and saved as a reusable standard logic block, which can be shared by many designers within the same organization – this is the beauty of design reuse.

This "hardware" macro function is independent of any other function or event not related to its operation. Therefore, it cannot be affected by extraneous system interrupts or other unconnected system state changes.

Such independence is critical in safety systems. Extraneous system interrupts in a purely software-based system could cause indeterminate states that are hard to test or simulate.

PC-BASED MOTOR CONTROL

Our next example (Figure 7-5 and Figure 7-6) is more complex, because now the motor is connected to a PC-based system via an RS-232 serial connection.

This implementation has a closed loop system controlling rotation, speed, and direction.

There is also the addition of a safety-critical emergency stop, which has the highest level of system interrupt. This means that if the emergency stop is acti-

vated, it will override any other process or interrupt and will immediately stop the motor from rotating.

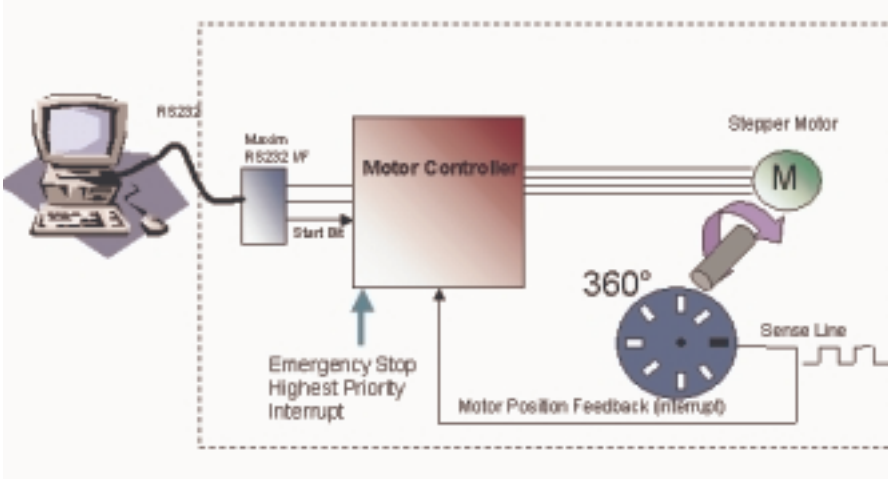


FIGURE 7-5: DESIGN PARTITIONING

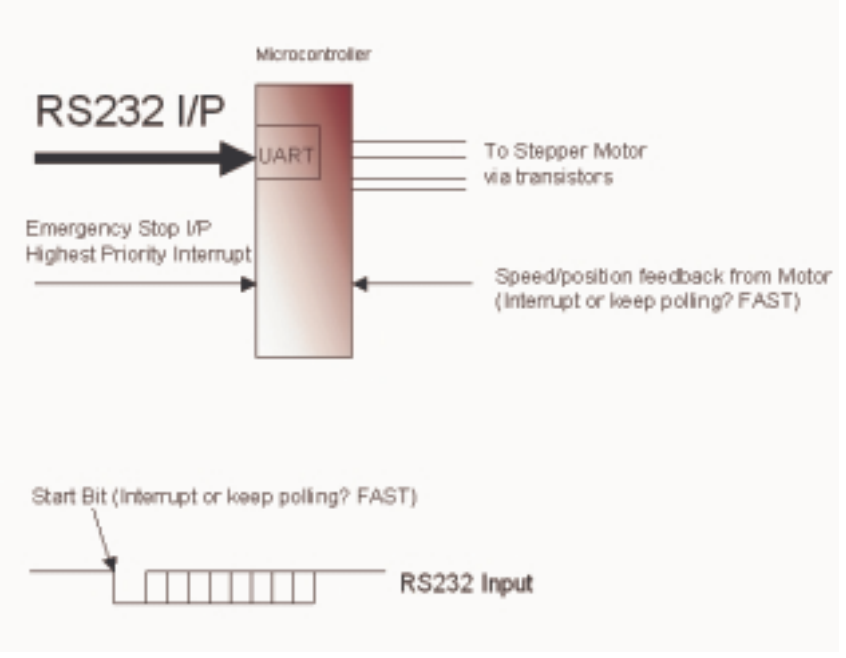


FIGURE 7-6: MICROCONTROLLER IMPLEMENTATION

This design solution purely uses a microcontroller. The main functions it performs are:

- Interrupt control
- Status feedback to the PC
- Accurate motor control.

This configuration would probably be implemented in a single microcontroller device with specific motor control peripherals, such as a capture-compare unit.

This configuration would also need a built-in UART. These extra functions usually add extra cost to the overall microcontroller device.

Due to the nature of the microcontroller, the interrupt handling must be thoroughly mapped out, because interrupts could affect the speed of the motor.

In a safety-critical system, emergency stops implemented in software require exhaustive testing and verification before they can be used in the final system to ensure that they operate properly under all software related conditions, including software bugs and potential software states.

The output from the motor rotation sensor is very fast, so control of the speed of the motor could cause problems if system interrupts occurred.

DESIGN PARTITIONING

As we noted before, microcontrollers are very good at computational tasks, and CPLDs are excellent in high-speed systems, with their abundance of I/Os.

Figure 7-7 shows how we can use a microcontroller and a CPLD in a partitioned design to achieve the greatest control over a stepper motor.

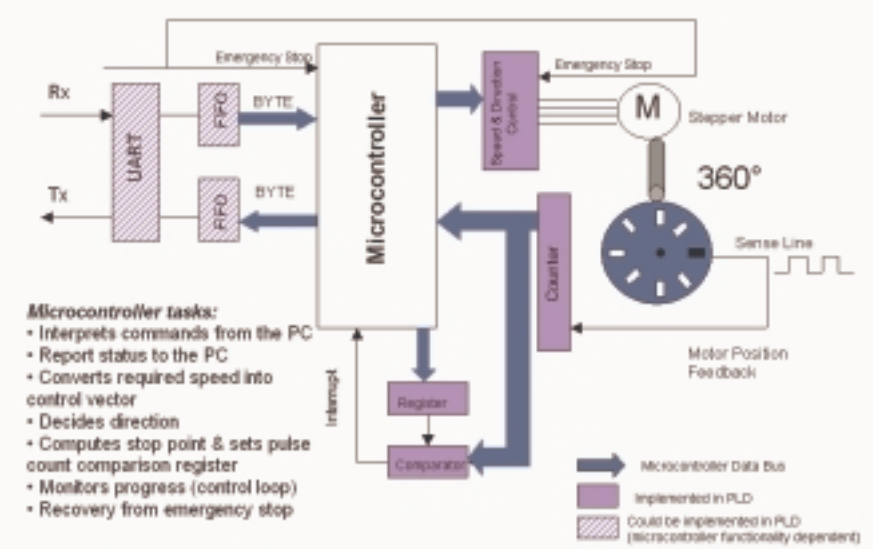


FIGURE 7-7: PARTITIONED DESIGN: MICROCONTROLLER AND CPLD

The microcontroller:

- Interprets ASCII commands from the PC.
- Reports status of the motor to the PC.
- Converts required speed into control vectors (small mathematical algorithm).
- Decides direction of rotation of the motor.
- Computes stop point and sets a value into the pulse count comparison register.
- Monitors progress (control loop) and adapts speed.
- Recovers from emergency stops.

Although the microcontroller performs recovery from emergency stops, the actual emergency stop is implemented by the CPLD, because this is the safety-critical part of the design.

Because the CPLD is considered independent hardware, safety-critical proving and sign-off are more straightforward than software safety systems.

Additionally, all of the high-speed interface functions are also implemented in the CPLD, because it is very fast and has abundant inputs and outputs.

Meanwhile, the UART and FIFO sections of the design can be implemented in the microcontroller in the form of a microcontroller peripheral, or imple-

mented in a larger, more granular PLD such as an FPGA – for example, a Xilinx Spartan device.

Using a PLD in this design has the added benefit of gaining the ability to absorb any other discrete logic elements on the PCB or in the total design into the CPLD.

Under this new configuration, we can consider the CPLD as offering hardware-based subroutines or as a mini co-processor.

The microcontroller still performs ASCII string manipulation and mathematical functions, but it now has more time to perform these operations – without interruption. The motor control is now independently stable and safe.

Microcontroller/CPLD design partitioning can reduce overall system costs. This solution uses low-cost devices to implement the functions they do best – computational functions in the microcontroller and high-speed, high I/O tasks in the CPLD.

In safety-critical systems, why not put the safety-critical functions in "hardware" (CPLDs) to cut down on safety system approval time scales?

System testing can also be made easier by implementing the difficult-to-simulate interrupt handling into programmable logic.

Low-cost microcontrollers are now in the region of US $1.00, but if your design requires extra peripherals (for example, a capture-compare unit for accurate motor control, ADCs, or UARTs), this can quadruple the cost of your microcontroller.

A low cost microcontroller coupled with a low cost CPLD from Xilinx can deliver the same performance at approximately half the cost.

In low-power applications, microcontrollers are universally accepted as low-power devices and have been the automatic choice of designers.

The CoolRunner family of ultra-low power CPLDs are an ideal fit in this arena and may be used to complement your low-power microcontroller to integrate designs in battery-powered, portable designs (<100 µA current consumption at standby).

CONCLUSION

Microcontrollers are ideally suited to computational tasks, whereas CPLDs are suited to very fast, I/O-intensive operations.

Partitioning your design across the two devices can increase overall system speeds, reduce costs, and potentially absorb all of the other discrete logic functions in a Design – thus presenting a truly reconfigurable system.

The design process for a microcontroller is very similar to that of a programmable logic device. This permits a shorter learning and designing cycle.

Full-functioning software design tools for Xilinx CPLDs are free of charge and may be downloaded from the Xilinx website. Thus, your first project using CPLDs can be not only quick and painless, but very cost-effective as well.

(Excerpted from the Xilinx Xcell Journal*, Issue 39, Spring 2001.)*

Documentation and Example Code

The following is a list of selected application notes, white papers, tutorials, and example code that can be downloaded from the Xilinx website.

This list grows longer as more applications are developed. For the latest list, please visit *www.xilinx.com/apps/appsweb.htm*.

TABLE 7-1: DOCUMENTATION LIST

Title	Number	Family	Design Code
Embedded Instrumentation Using XC9500 CPLDs	XAPP076	XC9500	
Configuring Xilinx FPGAs Using an XC9500 CPLD and Parallel PROM	XAPP079	XC9500	
Xilinx FPGAs: A Technical Overview for the First-Time User	XAPP097	FPGA	
Choosing a Xilinx Product Family	XAPP100	All	
XC9500 Remote Field Upgrade	XAPP102	XC9500	
A Quick JTAG ISP Checklist	XAPP104	XC9500	
A CPLD VHDL Introduction	XAPP105	XC9500	
Adapting ASIC Designs for Use with Spartan FPGAs	XAPP119	Spartan	
170 MHz FIFOs using the Virtex Block SelectRAM+ Feature	XAPP131	Virtex	
Synthesizable High-Performance SDRAM Controllers	XAPP134	Virtex	FREE VHDL and Verilog

TABLE 7-1: DOCUMENTATION LIST *(CONTINUED)*

Title	Number	Family	Design Code
Synthesizable 200 MHz ZBT SRAM Interface	XAPP136	Virtex	FREE VHDL and Verilog
Virtex Synthesizable Delta-Sigma DAC v 1.1	XAPP154	Virtex	
MP3 NG: A Next Generation Consumer Platform	XAPP169	Spartan-II	
Implementing an ISDN PCMCIA Modem Using Spartan Device, v1.0	XAPP170	Spartan	
Using Delay-Locked Loops in Spartan-II FPGAs	XAPP174	Spartan-II	FREE VHDL and Verilog
High Speed FIFOs In Spartan-II FPGAs	XAPP175	Spartan-II	FREE VHDL and Verilog
IDCT Implementation in Virtex Devices for MPEG Applications, v 1.1	XAPP208	Virtex	FREE VHDL
PicoBlaze 8-Bit Microcontroller for Virtex Devices	XAPP213	Virtex and Spartan	
CoolRunner Visor Springboard LED Test v1.1	XAPP357	CoolRunner	
CoolRunner XPLA3 SMBus Controller Implementation	XAPP353	CoolRunner	FREE VHDL and Verilog
CoolRunner CPLD3 CPLD 8051 Microcontroller Interface	XAPP349	CoolRunner	FREE VHDL and Verilog

TABLE 7-1: DOCUMENTATION LIST (*CONTINUED*)

Title	Number	Family	Design Code
CoolRunner Serial Peripheral Interface Master v1.2	XAPP348	CoolRunner	FREE VHDL and Verilog
UARTs in Xilinx CPLDs	XAPP341	CoolRunner	FREE VHDL and Verilog
Design of a 16b/20b Encoder/Decoder Using a CoolRunner XPLA3 CPLD	XAPP336	CoolRunner	FREE VHDL and Verilog
CoolRunner CPLD I2C Bus Controller Implementation	XAPP333	CoolRunner	FREE VHDL and Verilog
Manchester Encoder-Decoder for Xilinx CPLDs v1.3	XAPP339	CoolRunner	FREE VHDL and Verilog
Design of a MP3 Portable Player Using a CoolRunner CPLD	XAPP328	CoolRunner	FREE VHDL and Verilog
Content Addressable Memory (CAM) in ATM Applications	XAPP202	Virtex, Virtex-II	FREE VHDL and Verilog
Virtex Analog to Digital Converter v1.1	XAPP155	Virtex	
Design an Eight Channel Digital Volt Meter with a Springboard Development Kit	XAPP146	CoolRunner	FREE VHDL and Verilog

TABLE 7-1: DOCUMENTATION LIST *(CONTINUED)*

Title	Number	Family	Design Code
Exemplar/ModelSim Tutorial for CPLDs	Tutorial	CPLDs	
Workstation Flow for Xilinx CoolRunner CPLDs	Tutorial	CPLDs	
OrCAD/ModelSim Tutorial for CPLDs	Tutorial	CPLDs	
Understanding the CoolRunner-II Timing Model	XAPP375	CoolRunner-II	
Understanding the CoolRunner-II Logic Engine	XAPP376	CoolRunner-II	
Using CoolRunner-II Advanced Features	XAPP378	CoolRunner-II	FREE HDL
High Speed Design with CoolRunner-II CPLDs	XAPP379	CoolRunner-II	
Powering CoolRunner-II CPLDs	XAPP389	CoolRunner-II	
Low Power Design with CoolRunner-II CPLDs	XAPP377	CoolRunner-II	
Title	Number	Family	Design Code
CoolRunner-II I/O Characteristics	XAPP382	CoolRunner-II	
5V Tolerance Techniques for CoolRunner-II Devices	XAPP429	CoolRunner-II	
Building Crosspoint Switches with CoolRunner-II CPLD	XAPP380	CoolRunner-II	FREE HDL
Telematics Digital Convergence – How to Cope with Emerging Standards and Protocols	WP194	All Products	
CipherStream Protocol: How CoolRunner-II CPLDs Protect FPGA IP v1.0	WP197	CoolRunner-II	

TABLE 7-1: DOCUMENTATION LIST (*CONTINUED*)

Title	Number	Family	Design Code
Using CoolRunner CPLDs in Smart Card Reader Applications	WP118	CoolRunner-II	
Could Automotive Processor Obsolescence be History?	WP169	All Products	
Spartan-II Family as a Memory Controller for QDR-SRAMs	WP111	Spartan-II	
Xilinx Spartan-II FIR Filter Solution	WP116	Spartan-II	

Website Reference

The following table is a summary of useful websites and web pages that could help you with your programmable logic designs.

TABLE 7-2: WEBSITE SUMMARY

Website/Page	Topic	Resources Available
www.xilinx.com	General Xilinx Website	Product data, investor information, application notes
www.support.xilinx.com	Technical Support	Comprehensive resource for all technical support
www.xilinx.com/ipcenter	IP Search Engine	Xilinx and third-party IP and cores
www.xilinx.com/esp	Emerging Standards and Protocol web portal	Resource portal, including data on home networking and Bluetooth™ – white papers, application notes, reference designs, block diagrams, “ask the experts,” links to industry websites

TABLE 7-2: WEBSITE SUMMARY *(CONTINUED)*

Website/Page	Topic	Resources Available
www.xilinx.com/support/education-home.htm	Customer Education	List of customer courses and types available
http://xup.msu.edu	University Program	
www.xilinx.com/support/searchtd.htm	Answers Database	
http://university.xilinx.com/univ/xsefaq1.htm	Student Edition FAQ	
http://toolbox.xilinx.com/cgi-bin/forum	Forums and Chat Rooms	
www.model.com	Simulation	Model Technology
www.optimagic.com/	Programmable Logic Jump Station	

Chapter 1:
Navigating This Book

Chapter 2: Table of Contents

Chapter 3: Introduction

The History of Programmable Logic 1
Complex Programmable Logic Devices (CPLDs) 4
Why Use a CPLD? 4
Field Programmable Gate Arrays (FPGAs) 6
Design Integration 8
The Basic Design Process 9
HDL File Change Example 13
Before (16 x 16 multiplier): 13
After (32 x 32 multiplier): 13
Intellectual Property (IP) Cores 14
Design Verification 14
Functional Simulation 16
Device Implementation 16
Fitting 16
Place and Route 17
Downloading or Programming 18
System Debug 19

Chapter 4: Xilinx Solutions

Introduction 21
Xilinx Devices 22
Platform FPGAs 22
Virtex FPGAs 22
Virtex-II Pro FPGAs 23
The Power of Xtreme Processing 23
XtremeDSP – The World's Fastest Programmable DSP Solution 23
The Ultimate Connectivity Platform 24
The Power of Integration 24
Enabling a New Development Paradigm 24
Industry-Leading Tools 24
Virtex FPGAs 24
Spartan FPGAs 25
Spartan-3 FPGAs 25

Shift register SRL16 blocks 27
As much as 520 Kb distributed SelectRAM™ memory 27
As much as 1.87 Mb Embedded block RAM 27
Memory Interfaces 27
Multipliers 28
XCITE Digitally Controlled Impedance Technology – A Xilinx Innovation 28
Spartan-3 XCITE DCI Technology Highlights 28
Full- and half-impedance input buffers 29
Spartan-3 Features and Benefits 29
Spartan-IIE FPGAs....31
Spartan-IIE Architectural Features....32
Logic Cells 35
Block RAM 37
Delay-Locked Loop 38
Configuration 39
Xilinx CPLDs 41
Product Features: 41
Selection Considerations: 41
XC9500 ISP CPLD Overview....42
XC9500 5V Family 42
Flexible Pin-Locking Architecture 42
Full IEEE 1149.1 JTAG Development and Debugging Support 42
XC9500 Product Overview Table 43
XC9500XL 3.3V Family....43
Family Highlights 44
Performance 44
Powerful Architecture 44
Highest Reliability 44
Advanced Technology 44
Outperforms All Other 3.3V CPLDs 45
XC9500XV 2.5V CPLD Family 45
High Performance Through Advanced Technology 45
The System Designer's CPLD 45
CoolRunner Low-Power CPLDs....47
XPLA3 Architecture 48
Logic Block Architecture 49
FoldBack NANDs 50
Macrocell Architecture 51
I/O Cell 52

Simple Timing Model 52
Slew Rate Control 53
XPLA3 Software Tools 53
CoolRunner-II CPLDs 55
CoolRunner-II Architecture Description 56
CoolRunner-II Function Block 57
CoolRunner-II Macrocell 59
Advanced Interconnect Matrix (AIM) 60
I/O Blocks 61
Output Banking 61
DataGATE 62
Additional Clock Options: Division, DualEDGE, and CoolCLOCK 63
Design Security 65
CoolRunner-II Application Examples 66
CoolRunner Reference Designs 68
Accessing the Reference Designs 68
..... Military and Aerospace 71
Automotive and Industrial 71
Xilinx IQ Solutions – Architecting Automotive Intelligence 71
Design-In Flexibility 72
Design Tools 73
Design Entry 73
Synthesis 74
Implementation and Configuration 74
Board-Level Integration 74
Verification Technologies 75
Static Verification 75
Dynamic Verification 76
Debug Verification 76
Board-Level Verification 76
Advanced Design Techniques 76
Embedded SW Design Tools Center 77
Embedded Software Tools for Virtex-II Pro Platform FPGAs 77
Xilinx IP Cores 78
Web-Based Information Guide 78
End Markets 79
Silicon Products and Solutions 80
Design Resources 80
System Resources 81
DSP Central 81
Algorithms/Cores 81

Xilinx Online (IRL) 81
Configuration Solutions 82
Processor Central 82
The Embedded Development Kit (EDK) 82
PowerPC Embedded Processor Solution 82
The UltraController Solution 82
MicroBlaze and PicoBlaze Soft Processor Solutions 83
Third-Party Processors Solution 84
CoreConnect Technology 84
Tools and Partnerships 84
Memory Corner 84
Silicon 85
Design Tools and Boards 85
Technical Literature and Training 85
Connectivity Central 86
Networking and Datapath Products 86
Control Plane and Backplane Products 86
High-Speed Design Resources 86
Signal Integrity Tools 86
Partnerships 86
Signal Integrity 86
Signal Integrity Fundamentals 87
Simulation Tools 87
Multi-Gigabit Signaling 87
Services 87
Xilinx Design Services 87
IP Core Modification 87
FPGA Design From Specification 87
FPGA System Design 87
Embedded Software Design 88
Education Services 88
Live E-Learning Environment 88
Day Segment Courses 89
Computer-Based Training (CBT) 89
University Program 89
Xilinx University Resource Center 89
Xilinx Answers Database 89
Xilinx Student Edition Frequently Asked Questions 90
Design Consultants 90
Technical Support 90

Chapter 5: WebPACK ISE Design Software

Module Descriptions 91
WebPACK Design Suite 93
WebPACK Design Entry 93
WebPACK StateCAD 93
WebPACK MXE Simulator 94
WebPACK HDL Bencher Tool 94
WebPACK FPGA Implementation Tools 94
WebPACK CPLD Implementation Tools 94
WebPACK iMPACT Programmer 94
WebPACK ChipViewer 95
XPower 95
WebPACK CD-ROM Installation 95
Getting Started 96
Licenses 96
Projects 97
Summary 97

Chapter 6: WebPACK ISE Design Entry

Introduction 99
Design Entry 100
The Language Template 104
Close the Language Templates 104
Edit the Counter Module 105
Save the Counter Module 107
Functional Simulation 107
State Machine Editor 112
Top-Level VHDL Designs 120
Top-Level Schematic Designs 125
ECS Hints 125
I/O Markers 128

Chapter 7: Implementing CPLDs

Introduction 131
Synthesis 132
Constraints Editor 133
CPLD Reports 142
Timing Simulation 144
Configuration 145

Chapter 8: Implementing FPGAs

Introduction 147
Synthesis 150
The Constraints File 153
FPGA Reports 158
Programming 159
Summary 159

Chapter 9: Design Reference Bank

Introduction 161
Get the Most out of Microcontroller-Based Designs 161
Conventional Stepper Motor Control 162
Using a Microcontroller to Control a Stepper Motor 165
Stepper Motor Control Using a CPLD 166
PC-Based Motor Control 168
Design Partitioning 170
Conclusion 172
Documentation and Example Code 173
Website Reference 177

Chapter 10: ACRONYMS

Chapter 11: GLOSSARY OF TERMS

ACRONYMS

ABEL	Advanced Boolean Expression Language
ADC	Analog-to-Digital Converter
AIM	Advanced Interconnect Matrix
ANSI	American National Standards Institute
ASIC	Application Specific Integrated Circuit
ASSP	Application Specific Standard Product
ATE	Automatic Test Equipment
BGA	Ball Grid Array
BLVDS	Backplane Low Voltage Differential Signaling
BUFG	Global Clock Buffer
CAD	Computer Aided Design
CAN	Controller Area Network
CBT	Computer Based Training
CDMA	Code Division Multiple Access
CE	Clock Enable
CLB	Configurable Logic Block
CLK	Clock Signal
CMOS	Complementary Metal Oxide Semiconductor
CPLD	Complex Programmable Logic Device
CPLD	Complex Programmable Logic Device
CSP	Chip Scale Packaging
DCI	Digitally Controlled Impedance
DCM	Digital Clock Manager
DCM	Digital Control Management
DES	Data Encryption Standard
DRAM	Dynamic Random Access Memory
DRC	Design Rule Checker
DSL	Digital Subscriber Line
DSP	Digital Signal Processor
DTV	Digital Television
ECS	Schematic Editor
EDA	Electronic Design Automation
EDIF	Electronic Digital Interchange Format

EMI	Electromagnetic Interference
EPROM	Erasable Programmable Read Only Memory
eSP	emerging Standards and Protocols
FAT	File Allocation Table
FIFO	First In First Out
FIR	Finite Impulse Response (Filter)
FIT	Failures in Time
FLBGA	Flip Chip Ball Grid Array
fMax	Frequency Maximum
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GPS	Global Positioning System
GTL	Gunning Transceiver Logic
GTLP	Gunning Transceiver Logic Plus
GUI	Graphical User Interface
HDL	Hardware Description Language
HDTV	High Definition Television
HEX	Hexadecimal
HSTL	High Speed Transceiver Logic
I/O	Inputs and Outputs
IBIS	I/O Buffer Information Specification
IEEE	Institute of Electrical and Electronics Engineers
ILA	Integrated Logic Analyzer
IOB	Input Output Block
IP	Intellectual Property
IRL™	Internet Reconfigurable Logic
ISE	Integrated Software Environment
ISP	In System Programming
JEDEC	Joint Electron Device Engineering Council
JTAG	Joint Test Advisory Group
LAN	Local Area Network
LEC	Logic Equivalence Checker
LMG	Logic Modeling Group
LSB	Least Significant Bit
LUT	Look Up Table
LVCMOS	Low Voltage Complementary Metal Oxide Semiconductor

LVDS	Low Voltage Differential Signaling
LVDSEXT	Low Voltage Differential Signaling Extension
LVPECL	Low Voltage Positive Emitter Coupled Logic
LVTTL	Low Voltage Transistor to Transistor Logic
MAC	Multiply and Accumulate
MAN	Metropolitan Area Network
MCS	Manipulate Comment Section
MIL	Military
MOSFET	Metal Oxide Semiconductor Field Effect Transistors
MP3	MPEG Layer III Audio Coding
MPEG	Motion Picture Experts Group
MSB	Most Significant Bit
MUX	Multiplexer
NAND	Not And
NGC	Native Generic Compiler
NRE	Non-Recurring Engineering (Cost)
OE	Output Enable
OTP	One Time Programmable
PACE	Pinout and Area Constraints Editor
PAL	Programmable Array Logic
PCB	Printed Circuit Board
PCI	Peripheral Component Interconnect
PCMCIA	Personal Computer Memory Card International Association
PCS	Personnel Communications System
PLA	Programmable Logic Array
PLD	Programmable Logic Device
PROM	Programmable Read Only Memory
QFP	Quad Flat Pack
QML	Qualified Manufacturers Listing
QPRO™	QML Performance Reliability of Supply Off the Shelf ASIC
RAM	Random Access Memory
RC	Radio Controlled
ROM	Read Only Memory
SOP	Sum of Product
SPLD	Simple Programmable Logic Device
SRAM	Static Random Access Memory

SRL16	Shift Register LUT
SSTL	Stub Series Terminated Transceiver Logic
TIM	Time in Market
Tpd	Time of Propagation Delay (through the device)
TQFP	Thin Quad Flat Pack
TTM	Time to Market
UCF	User Constraints File
UMTS	Universal Mobile Telecommunications System
UV	Ultraviolet
VCCO	Voltage Current Controlled Oscillator
VFM	Variable Function Multiplexer
VHDL	VHISC High Level Description Language
VHSIC	Very High Speed Integrated Circuit
VREF	Voltage Reference
VSS	Visual Software Solutions
WAN	Wireless Area Network
WLAN	Wireless Local Access Network
WPU	Weak Pull Up
XCITE	Xilinx Controlled Impedance Technology
XOR	Exclusive OR
XST	Xilinx Synthesis Technology
ZIA	Zero Power Interconnect Array

GLOSSARY OF TERMS

ABEL – Advanced Boolean Expression Language, low-level language for design entry, from Data I/O.

AIM – Advanced Interconnect Matrix in the CoolRunner-II CPLD that provides the flexible interconnection between the PLA function blocks.

Antifuse – A small circuit element that can be irreversibly changed from being non-conducting to being conducting with ~100 Ohm. Anti-fuse-based FPGAs are thus non-volatile and can be programmed only once (see OTP).

AQL – Acceptable Quality Level. The relative number of devices, expressed in parts-per-million (ppm), that might not meet specification or be defective. Typical values are around 10 ppm.

ASIC – Application Specific Integrated Circuit, also called a gate array. Asynchronous logic that is not synchronized by a clock. Asynchronous designs can be faster than synchronous ones, but are more sensitive to parametric changes and are thus less robust.

ASSP – Application-Specific Standard Product. Type of high-integration chip or chipset ASIC that is designed for a common yet specific application.

ATM – Asynchronous Transfer Mode. A very high-speed (megahertz to gigahertz) connection-oriented bit-serial protocol for transmitting data and real-time voice and video in fixed-length packets (48-byte payload, 5-byte header).

Back Annotation – Automatically attaching timing values to the entered design format after the design has been placed and routed in an FPGA.

Behavioral Language – Top-down description from an even higher level than VHDL.

Block RAM – A block of 2k to 4k bits of RAM inside an FPGA. Dual-port and synchronous operation are desirable.

CAD – Computer Aided Design, using computers to design products.

CAE – Computer Aided Engineering, analyses designs created on a computer.

CLB – Configurable Logic Block. Xilinx-specific name for a block of logic surrounded by routing resources. A CLB contains two or four LUTs (function generators) plus two or four flip-flops.

CMOS – Complementary Metal-Oxide-Silicon. Dominant technology for logic and memory. Has replaced the older bipolar TTL technology in most applications (except very fast ones). CMOS offers lower power consumption and smaller chip sizes compared to bipolar and now meets or even beats TTL speed.

Compiler – software that converts a higher language description into a lower-level representation. For FPGAs: the complete partition, place and route process.

Configuration – The internally stored file that controls the FPGA so that it performs the desired logic function. Also, the act of loading an FPGA with that file.

Constraints – Performance requirements imposed on the design, usually in the form of max allowable delay, or required operating frequency.

CoolCLOCK – Combination of the clock divider and clock doubler functions in CoolRunner-II CPLDs to further reduce power consumption associated with high-speed clocked-in internal device networks.

CPLD – Complex Programmable Logic Device, synonymous with EPLD. PAL-derived programmable logic devices that implement logic as sum-of-products driving macrocells. CPLDs are known to have short pin-to-pin delays, and can accept wide inputs, but have relatively high power consumption and fewer flip-flops compared to FPGAs.

CUPL – Compiler Universal for Programmable Logic, CPLD development tool available from Logical Devices.

DataGATE – A function within CoolRunner-II devices to block free-running input signals, effectively blocking controlled switching signals so they do not drive internal chip capacitances to further reduce power consumption. Can be selected on all inputs.

Input Hysteresis – Input hysteresis provides designers with a tool to minimize external components, whether using the inputs to create a simple clock source or reducing the need for external buffers to sharpen a slow or noisy input signal. Function found in CoolRunner-II CPLDs (may also be referred to as Schmitt Trigger inputs in the text).

DCM – Digital Clock Manager. Provides zero-delay clock buffering, precise phase control, and precise frequency generation on Xilinx Virtex-II FPGAs.

DCI – Digitally Controlled Impedance in the Virtex-II solution dynamically eliminates drive strength variation due to process, temperature, and voltage fluctuation. DCI uses two external high-precision resistors to incorporate equivalent input and output impedance internally for hundreds of I/O pins.

Debugging – The process of finding and eliminating functional errors in software and hardware.

Density – Amount of logic in a device, often used to mean capacity. Usually measured in gates, but for FPGAs, better expressed in logic cells, each consisting of a 4-input LUT and a flip-flop.

DLL – Delay Locked Loop, A digital circuit used to perform clock management functions on- and off-chip.

DRAM – Dynamic Random Access Memory. A low-cost/read-write memory where data is stored on capacitors and must be refreshed periodically.

DRAMs are usually addressed by a sequence of two addresses – row address and column address – which makes them slower and more difficult to use than SRAMs.

DSP – Digital Signal Processing. The manipulation of analog data that has been sampled and converted into a digital representation. Examples are filtering, convolution, and Fast Fourier Transform

EAB – Embedded Array Block. Altera™ name for block RAM in FLEX10K.

EDIF – Electronic Data Interchange Format. Industry-standard for specifying a logic design in text (ASCII) form.

EPLD – Erasable Programmable Logic Devices, synonymous with CPLDs. PAL-derived programmable logic devices that implement logic as sum-of-products driving macrocells. EPLDs are known to have short pin-to-pin delays, and can accept wide inputs, but have relatively high power consumption and fewer flip-flops than FPGAs.

Embedded RAM – Read-write memory stored inside a logic device. Avoids the delay and additional connections of an external RAM.

ESD – Electro-Static Discharge. High-voltage discharge can rupture the input transistor gate oxide. ESD-protection diodes divert the current to the supply leads.

5-Volt Tolerant – Characteristic of the input or I/O pin of a 3.3V device that allows this pin to be driven to 5V without any excessive input current or device breakdown. Very desirable feature.

FIFO – First-In-First-Out memory, where data is stored in the incoming sequence and is read out in the same sequence. Input and output can be asynchronous to each other. A FIFO needs no external addresses, although all modern FIFOs are implemented internally with RAMs driven by circular read and write counters.

FIT – Failure In Time. Describes the number of device failures statistically expected for a certain number of device-hours. Expressed as failures per one billion device hours. Device temperature must be specified. MTBF can be calculated from FIT.

Flash – Non-volatile programmable technology, an alternative to Electrically-Erasable Programmable Read-Only Memory (EEPROM) technology. The memory content can be erased by an electrical signal. This allows in-system programmability and eliminates the need for ultraviolet light and quartz windows in the package.

Flip-Flop – Single-bit storage cell that samples its Data input at the active (rising or falling) clock edge, and then presents the new state on its Q output after that clock edge, holding it there until after the next active clock edge.

Floorplanning – Method of manually assigning specific parts of the design to specific chip locations. Can achieve faster compilation, better utilization, and higher performance.

Footprint – The printed circuit pattern that accepts a device and connects its pins appropriately. Footprint-compatible devices can be interchanged without modifying the PC board.

FPGA – Field Programmable Gate Array. An integrated circuit that contains configurable (programmable) logic blocks and configurable (programmable) interconnect between those blocks.

Function Generator – Also called look-up-table, with N-inputs and one output. Can implement any logic function of its N-inputs. N is between 2 and 6; 4-input function generators are most popular.

GAL – Generic Array Logic. Lattice name for a variation on PALs Gate. Smallest logic element with several inputs and one output. AND gate output is high when all inputs are high. OR gate output is high when at least one input is high. A 2-input NAND gate is used as the measurement unit for gate array complexity.

Gate Array – ASIC where transistors are pre-defined, and only the interconnect pattern is customized for the individual application.

GTL – Gunning Transceiver Logic. A high-speed, low-power back-plane standard.

GUI – Graphic User Interface. A way of representing the computer output on the screen as graphics, pictures, icons, and windows. Pioneered by Xerox and the Macintosh, now universally adopted (e.g., by Windows 95).

HDL – Hardware Description Language.

Hierarchical Design – Design description in multiple layers, from the highest (overview) to the lowest (circuit details). Alternative: flat design, where everything is described at the same level of detail. Incremental design making small design changes while maintaining most of the layout and routing.

Interconnect – Metal lines and programmable switches that connect signals between logic blocks and between logic blocks and the I/O.

IOB or I/O – Input/Output block. Logic block with features specialized for interfacing with the PC board.

ISO9000 – An internationally recognized quality standard. Xilinx is certified to ISO9001 and ISO9002.

IP – Intellectual Property. In the legal sense: patents, copyrights, and trade secrets. In integrated circuits: pre-defined large functions, called cores, that help you complete large designs faster.

IQ – Extended temperature devices for automotive and industrial applications.

ISP – In-System Programmable device. A programmable logic device that can be programmed after it has been connected to (soldered into) the system PC board. Although all SRAM-based FPGAs are naturally ISP, this term is only used with certain CPLDs, to distinguish them from the older CPLDs that must be programmed in programming equipment.

JTAG – Joint Test Action Group. Older name for IEEE 1149.1 Boundary Scan, a method to test PC boards and ICs.

LogiBLOX – Formerly called X-Blox. Library of logic modules, often with user-definable parameters, like data width. Very similar to LPM.

Logic Cell – Metric for FPGA density. One logic cell is one 4-input look-up table plus one flip-flop.

LPM – Library of Parameterized Modules. Library of logic modules, often with user-definable parameters, like data width. Very similar to LogiBlox.

LUT – Look-Up Table. Also called function generator with N inputs and one output. Can implement any logic function of its N inputs. N is between 2 and 6; 4-input LUTs are most popular.

Macrocell – The logic cell in a sum-of-products CPLD or PAL/GAL.

Mapping – Process of assigning portions of the logic design to the physical chip resources (CLBs). With FPGAs, mapping is a more demanding and more important process than with gate arrays.

MTBF – Mean Time Between Failure. The statistically relevant up-time between equipment failure. See also FIT.

Netlist – Textual description of logic and interconnects. See also XNF and EDIF.

NRE – Non-Recurring Engineering charges. Startup cost for the creation of an ASIC, gate array, or HardWire. Pays for layout, masks, and test development. FPGAs and CPLDs do not require NRE.

Optimization – Design change to improve performance. See also Synthesis.

OTP – One-Time Programmable. Irreversible method of programming logic or memory. Fuses and anti-fuses are inherently OTP. EPROMs and EPROM-based CPLDs are OTP if their plastic package blocks the ultraviolet light needed to erase the stored data or configuration.

PAL – Programmable Array Logic. Oldest practical form of programmable logic, implemented a sum-of-products plus optional output flip-flops.

Partitioning – In FPGAs, the process of dividing the logic into sub-functions that can later be placed into individual CLBs. Partitioning precedes placement.

PCI – Peripheral Component Interface. Synchronous bus standard characterized by short range, light loading, low cost, and high performance. A 33 MHz PCI can support data byte transfers of up to 132 megabytes per second on 36 parallel data lines (including parity) and a common clock. There is also a new 66 MHz standard.

PCMCIA – Personal Computer Memory Card Interface Association. (Also: People Can't Memorize Computer Industry Acronyms). Physical and electrical standard for small plug-in boards for portable computers.

Pin-Locking – Rigidly defining and maintaining the functionality and timing requirements of device pins while the internal logic is still being designed or modified. Pin-locking has become important, since circuit-board fabrication times are longer than PLD design implementation times.

PIP – Programmable Interconnect Point. In Xilinx FPGAs, a point where two signal lines can be connected, as determined by the device configuration.

Placement – In FPGAs, the process of assigning specific parts of the design to specific locations (CLBs) on the chip. Usually done automatically.

PLA – Programmable Logic Array. The first and most flexible programmable logic configuration with two programmable planes providing any combination of "AND" and "OR" gates and sharing of AND terms across multiple ORs. This architecture is implemented in CoolRunner and CoolRunner-II devices.

PLD – Programmable Logic Device. Most generic name for all programmable logic: PALs, CPLDs, and FPGAs.

QML – Qualified Manufacturing Line. For example, ISO9000.

Routing – The interconnection, or the process of creating the desired interconnection, of logic cells to make them perform the desired function. Routing follows partitioning and placement.

Schematic – Graphic representation of a logic design in the form of interconnected gates, flip-flops, and larger blocks. Older and more visually intuitive alternative to the increasingly more popular equation-based or high-level language text description of a logic design.

SelectRAM – Xilinx-specific name for a small RAM (usually 16 bits), implemented in a LUT.

Simulation – Computer modeling of logic and (sometimes) timing behavior of logic driven by simulation inputs (stimuli or vectors).

SPROM – Serial Programmable Read-Only Memory. Non-volatile memory device that can store the FPGA configuration bitstream. The SPROM has a built-in address counter, receives a clock, and outputs a serial bitstream.

SRAM – Static Random Access Memory. Read-write memory with data stored in latches. Faster than DRAM and with simpler timing requirements, but smaller in size and about four times as expensive than DRAM of the same capacity.

SRL16 – Shift Register LUT, an alternative mode of operation for every function generator (LUT) that are part of every CLB in Virtex and Spartan FPGAs. This mode increases the number of flip-flops by 16. Adding flip-flops enables fast pipelining – ideal in DSP applications.

Static Timing – Detailed description of on-chip logic and interconnect delays.

Sub-Micron – The smallest feature size is usually expressed in micron (µ = millionth of a meter, or thousandth of a millimeter) The state of the art is moving

from 0.35µ to 0.25µ, and may soon reach 0.18µ. The wavelength of visible light is 0.4 to 0.8µ. 1 mil = 25.4µ.

Synchronous – Circuitry that changes state only in response to a common clock, as opposed to asynchronous circuitry that responds to a multitude of derived signals. Synchronous circuits are easier to design, debug, and modify, and tolerate parameter changes and speed upgrades better than asynchronous circuits.

Synthesis – Optimization process of adapting a logic design to the logic resources available on the chip, like LUTs, longline, and dedicated carry. Synthesis precedes mapping.

SystemI/O – Technology incorporated in Virtex-II FPGAs that uses the SelectIO-Ultra blocks to provide the fastest and most flexible electrical interfaces available. Each I/O pin is individually programmable for any of the 19 single-ended I/O standards or six differential I/O standards, including LVDS, SSTL, HSTL II, and GTL+. SelectIO-Ultra technology delivers 840 Mbps LVDS performance using dedicated DDR registers.

TBUFs – Buffers with a tri-state option, where the output can be made inactive. Used for multiplexing different data sources onto a common bus. The pull-down-only option can use the bus as a wired AND function.

Timing – Relating to delays, performance, or speed.

Timing Driven – A design or layout method that takes performance requirements into consideration.

UART – Universal Asynchronous Receiver/Transmitter. An 8-bit-parallel-to-serial and serial-to-8-bit-parallel converter, combined with parity and start-detect circuitry and sometimes even FIFO buffers. Used widely in asynchronous serial communications interfaces such as modems.

USB – Universal Serial Bus. A new, low-cost, low-speed, self-clocking bit-serial bus (1.5 MHz and 12 MHz) using four wires (Vcc, ground, differential data) to daisy-chain as many as 128 devices.

VME – Older bus standard, popular with MC68000-based industrial computers.

XNF File – Xilinx proprietary description format for a logic design Alternative: EDIF.