Programmable Logic Design
Quick Start Hand Book

By Karen Parnell & Nick Mehta

January 2002

XILINX®

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 seventies and since then have proved to be very popular and are now one of the largest growing sectors in the semiconductor industry. Why are programmable logic devices so widely used? Programmable logic devices provide designers ultimate flexibility, time to market advantage, design integration, are 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 can also be used as a stand-alone tutorial and information source for the first of your 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, where and how to use them, how to install the free, full functioning design software (Xilinx WebPACK™ ISE included with this book) and then guides you through your first of many designs. There are also sections on VHDL and schematic capture design entry and finally a data bank of useful applications examples.

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

Karen Parnell & Nick Mehta

Programmable Logic Design Quick Start Hand Book

NAVIGATING THE BOOK

This report was written for both the professional engineer who has never designed using programmable logic devices and for the new engineer embarking on their exciting career in electronics design. To accommodate this the following navigation section has been written to help the reader decide in advance which section he/she wishes to read.

Chapter 1
Introduction

This chapter gives an overview of how and where programmable logic devices are used. It gives a brief history of the 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 PLD designs enable time to market advantage, design flexibility and system future proofing. The Xilinx portfolio includes both CPLD & FPGA devices, design software, design services & support, and Cores.

Chapter 3
WebPACK ISE Design Software

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

NAVIGATING THE BOOK (Continued)

Chapter 4
WebPACK ISE Design Entry

This section is a step by step approach to your first simple design. The following pages are intended to demonstrate the basic PLD design entry implementation process.

Chapter 5
Implementing FPGAs

This chapter discusses the Synthesis and implementation process for FPGAs. The design targets a Spartan IIE FPGA.

Chapter 6
Implementing CPLDs

This section takes the VHDL or Schematic design through to a working physical device. The design is the same design as in the previous chapters but targeting a CoolRunner CPLD.

Chapter 7
Design Reference Bank

The 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. It will also give you pointers on where to look for and download code and search for Intellectual Property (IP) Cores from the Xilinx Web site.

CONTENTS

CONTENTS (Continued)

CONTENTS (Continued)

ABBREVIATIONS

ABEL	Advanced Boolean Expression Language
ASIC	Application Specific Integrated Circuit
ASSP	Application Specific Standard Product
ATE	Automatic Test Equipment
CDMA	Code Division Multiple Access
CPLD	Complex Programmable Logic Device
CLB	Configurable Logic Block
DES	Data Encryption Standard
DRAM	Dynamic Random Access Memory
DSL	Digital Subscriber Line
DSP	Digital Signal Processor
DTV	Digital Television
ECS	Schematic Editor
EDA	Electronic Design Automation
FAT	File Allocation Table
FIFO	First In First Out
FIR	Finite Impulse Response (Filter)
Fmax	Frequency Maximum
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GPS	Geo-stationary Positioning System
GUI	Graphical User Interface
HDTV	High Definition Television
IP	Intellectual Property
I/O	Inputs and Outputs
IRL™	Internet Reconfigurable Logic
ISP	In-System Programming
JTAG	Joint Test Advisory Group
LSB	Least Significant Bit
LUT	Look Up Table
MP3	MPEG Layer III Audio Coding

ABBREVIATIONS   (Continued)

MPEG	Motion Picture Experts Group
MSB	Most Significant Bit
NRE	Non-Recurring Engineering (Cost)
PAL	Programmable Array Logic device
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
EPROM	Erasable Programmable Read Only Memory
RAM	Random Access Memory
ROM	Read Only Memory
SPLD	Simple Programmable Logic Device
SRAM	Static Random Access Memory
SRL16	Shift Register LUT
Tpd	Time of Propagation Delay through the device
UMTS	Universal Mobile Telecommunications System
VHDL	VHISC High Level Description Language
VHSIC	Very High Speed Integrated Circuit
VSS	Visual Software Solutions
WLAN	Wireless Local Access Network
XST	Xilinx Synthesis Technology
QML	Qualified Manufacturers Listing
QPRO™	QML Performance Reliability of supply Off-the-shelf ASIC

1

INTRODUCTION

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

1.1 The History of Programmable Logic

By the late 70's, standard logic devices were the rage and printed circuit boards were loaded with them. Then someone asked the question: "What if we gave the designer the ability to implement different interconnections in a bigger device?" This would allow the designer to integrate many standard logic devices into one part. In order to give the ultimate in design flexibility Ron Cline from Signetics (which was later purchased by Philips and then eventually Xilinx®!) came up with the idea of two programmable planes. The two programmable planes provided any combination of 'AND' and 'OR' gates and sharing of AND terms across multiple OR's.

This architecture was very flexible, but at the time due to wafer geometry's of 10um the input to output delay or propagation delay (Tpd) was high which made the devices relatively slow.

Figure 1.1   What is a CPLD?

MMI (later purchased by AMD) was enlisted as a second source for the PLA array but after fabrication issues was modified to become the Programmable Array Logic (PAL) architecture by fixing one of the programmable planes. This new architecture differs from that of the PLA by having one of the programmable planes fixed - the OR array. This PAL architecture had the added benefit of faster Tpd and less complex software but without the flexibility of the PLA structure. Other architectures followed, such as the PLD (Programmable Logic Device). This category of devices is often called Simple PLD (SPLD).

Figure 1.2   SPLD Architectures

The architecture has a mesh of horizontal and vertical interconnect tracks. At each junction, there is a fuse. With the aid of software tools, the user can select which junctions will not be connected by "blowing" all unwanted fuses. (This is done by a device programmer or more commonly nowadays using In-System Programming or ISP). Input pins are connected to the vertical interconnect and the horizontal tracks are connected to AND-OR gates, also called "product terms". These in turn connect to dedicated flip-flops whose outputs are connected to output pins.

PLDs provided as much as 50 times more gates in a single package than discrete logic devices! A huge improvement, not to mention fewer devices needed in inventory and higher reliability over standard logic.

Programmable Logic Device (PLD) technology has moved on from the early days with such companies as Xilinx producing ultra low power CMOS devices based on Flash technology. Flash PLDs provide the

ability to program the devices time and time again electrically
programming and ERASING the device! Gone are the days of erasing
taking in excess of twenty minutes under an UV eraser.

1.2   Complex Programmable Logic Devices (CPLDs)

Complex Programmable Logic Devices (CPLD) are another way to
extend the density of the simple PLDs. The concept is to have a few
PLD blocks or macrocells on a single device with general purpose
interconnect in between. Simple logic paths can be implemented
within a single block. More sophisticated logic will require multiple
blocks and use the general purpose interconnect in between to make
these connections.

Figure 1.3   CPLD Architecture

CPLDs are great at handling wide and complex gating at blistering
speeds e.g. 5ns which is equivalent to 200MHz. The timing model for
CPLDs is easy to calculate so before you even start your design you
can calculate your in to output speeds.

1.2.1   Why Use a CPLD?

CPLDs enable ease of design, lower development costs, more product revenue for your money, and the opportunity to speed your products to market...

Ease of Design: CPLDs offer the simplest way to implement design. Once a design has been described, by schematic and/or HDL entry, a designer simply uses CPLD development tools to optimise, fit, and simulate the design. The development tools create a file, which is then used to customise (program) a standard off-the-shelf CPLD with the desired functionality. This provides an instant hardware prototype and allows the debugging process to begin. If modifications are needed, design changes are just entered into the CPLD development tool, and the design can be re-implemented and tested immediately.

Lower Development Costs: CPLDs offer very low development costs. Ease of design, as described above, allows for shorter development cycles. Because CPLDs are re-programmable, designers can easily and very inexpensively change their designs. This allows them to optimise their designs and continues to add new features to continue to enhance their products. CPLD development tools are relatively inexpensive and in the case of Xilinx, are free. Traditionally, designers have had to face large cost penalties such as re-work, scrap, and development time. With CPLDs, designers have flexible solutions thus avoiding many traditional design pitfalls.

More Product Revenue: CPLDs offer very short development cycles, which means your products get to market quicker and begin generating revenue sooner. Because CPLDs are re-programmable, products can be easily modified using ISP over the Internet. This in turn allows you to easily introduce additional features and quickly generate new revenue from them. (This results in an expanded time for revenue). Thousands of designers are already using CPLDs to get to market quicker and then stay in the market longer by continuing to enhance their products even after they have been introduced into the field. *CPLDs decrease Time To Market (TTM) and extend Time In Market (TIM).*

Reduced Board Area: CPLDs offer a high level of integration (large number of system gates per area) and are available in very small form factor packages. This provides the perfect solution for designers of products which must fit into small enclosures or who have a limited amount of circuit board space to implement the logic design. The CoolRunner® CPLDs are available in the latest chip scale packages, e.g. CP56 which has a pin pitch of 0.5mm and is a mere 6mm by 6mm in size so are ideal for small, low power end products.

Cost of Ownership: Cost of Ownership can be defined as the amount it costs to maintain, fix, or warranty a product. For instance, if a design change requiring hardware rework must be made to a few prototypes, the cost might be relatively small. However, as the number of units that must be changed increases, the cost can become enormous. Because CPLDs are re-programmable, requiring no hardware rework, it costs much less to make changes to designs implemented using them. Therefore cost of ownership is dramatically reduced. And don't forget the ease or difficulty of design changes can also affect opportunity costs. Engineers who are spending a lot of time fixing old designs could be working on introducing new products and features - ahead of the competition.

There are also costs associated with inventory and reliability. PLDs can reduce inventory costs by replacing standard discrete logic devices. Standard logic has a predefined function and in a typical design lots of different types have to be purchased and stocked. If the design is changed then there may be excess stock of superfluous devices. This issue can be alleviated by using PLDs i.e. you only need to stock one device and if your design changes you simply reprogram. By utilising one device instead of many your board reliability will increase by only picking and placing one device instead of many. Reliability can also be increased by using the ultra low power CoolRunner CPLDs i.e. lower heat dissipation and lower power operation leads to decreased Failures In Time (FIT).

1.3   Field Programmable Gate Arrays (FPGAs)

In 1985, a company called Xilinx introduced a completely new idea. The concept was to combine the user control and time to market of PLDs with the densities and cost benefits of gate arrays. A lot of customers liked it - and the FPGA was born. Today Xilinx is still the number one FPGA vendor in the world!

An FPGA is a regular structure of logic cells™ or modules and interconnect which is under the designer's complete control. This means the user can design, program and make changes to his circuit whenever he wants. And with FPGAs now exceeding the 10 million gate limit (Xilinx Virtex® II is the current record holder), the designer can dream big!

FPGA Architecture

interconnect   gates

- **Channel Based Routing**

- **Post Layout Timing**

- **Tools More Complex than CPLDs**

- **Fine Grained**

- **Fast register pipelining**

"What if we could develop the equivalent of a circuit board full of standard logic parts (like TTL & PAL devices) on a single high density programmable logic chip?"

1,000,000+ gates

Xilinx - Freeman - 1985

Figure 1.4   FPGA Architecture

With the introduction of the Spartan® range of FPGAs we can now compete with Gate Arrays on all aspects - price, gate and I/O count,

performance and cost! The new Spartan IIE will provide up to 300k gates at a price point that enables Application Specific Standard Product (ASSP) replacement. For example a Reed Solomon IP Core implemented in a Spartan II XC2S100 FPGA has an effective cost of $9.95 whereas the equivalent ASSP would cost around $20.

There are 2 basic types of FPGAs: SRAM-based reprogrammable and One-time programmable (OTP). These two types of FPGAs differ in the implementation of the logic cell™ and the mechanism used to make connections in the device.

The dominant type of FPGA is SRAM-based and can be reprogrammed by the user as often as the user chooses. In fact, an SRAM FPGA is reprogrammed every time it is powered-up because the FPGA is really a fancy memory chip! (That's why you need a serial PROM or system memory with every SRAM FPGA).

Figure 1.5   Digital Logic History

In the SRAM logic cell, instead of conventional gates there is instead a Look Up Table (LUT) which determines the output based on the values of the inputs. (In the "SRAM logic cell" diagram above you can see 6 different combinations of the 4 inputs that will determine the values of the output). SRAM bits are also used to make connections.

One-time programmable (OTP) FPGAs use anti-fuses (contrary to fuses, connections are made not "blown" during programming) to make permanent connections in the chip and so do not require a SPROM or other means to download the program to the FPGA. However, every time you make a design change, you must *throw away the chip!* The OTP logic cell is very similar to PLDs with dedicated gates and flip-flops.

Design Integration

The integration of 74 series standard logic into a low cost CPLD is a very attractive proposition. Not only do you save Printed Circuit Board (PCB) area and board layers therefore reducing your total system cost but you only have to purchase and stock one generic part instead of upto as many as twenty pre-defined logic devices. In production the pick and place machine only has to place one part - therefore speeding up production. Less parts means higher quality and better Failure In Time (FIT) factor.

By using Xilinx CoolRunner devices (our family of ultra low power parts) in a design customers can benefit from low power consumption and reduced thermal emissions. This in turn leads to the reduction of the use of heat sinks (another cost saving) and a higher reliability end product.

Figure 1.6   Basic Logic Definitions

1.4   The Basic Design Process

The availability of design software such as WebPACK™ ISE has made it much easier to design with programmable logic. Designs can be described easily and quickly using either a description language such as ABEL (Advanced Boolean Expression Language), VHDL (VHSIC Hardware Description Language), Verilog or via a schematic capture package.

Schematic capture is the traditional method that designers have used to specify gate arrays and programmable logic devices. It is a graphical tool that allows the designer to specify the exact gates he requires and how he wants them connected. There are 4 basic steps to using schematic capture.

Step one: After selecting a specific schematic capture tool and device library, the designer begins building his circuit by loading the desired gates from the selected library. He can use any combination of gates that he needs. A specific vendor and device family library must be chosen at this time (e.g. Xilinx XCR3256XL) but he doesn't have to know what device within that family he will ultimately use with respect to package and speed.

Step two: Connect the gates together using nets or wires. The designer has complete control of connecting the gates in whatever configuration is required for his application.

Step three: The input and output buffers are added and labelled. These will define the I/O package pins for the device.

Step four: The final step is to generate a netlist.

Figure 1.7   PLD Design Flow

The netlist is a text equivalent of the circuit which 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 for 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 the nets (N1 to N8). When we implement this design, it will have input package pins A, B, C, D and output pins Q, R, S.

EDIF (Electronic Digital Interchange Format) is the industry-wide standard for netlists although there are many other including vendor-specific ones such as the Xilinx Netlist Format (XNF).

If 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 are obviously very simplistic. A more realistic design of 10,000 equivalent gates is shown here.

The typical schematic page contains about 200 gates included the logic 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 design a 20k, 50k or 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), Behavioural 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 *Behavioural* is used because in this powerful language, the designer describes the function or behaviour of the circuit in words rather than figuring out the appropriate gates needed to create the application.

There are two major flavours of HDL: VHDL and Verilog. Although it's not really important for you to know, VHDL is an acronym for "VHSIC High-level Design Language". And yes, VHSIC is another acronym "Very High Speed Integrated Circuit".

As an example we will design a 16 by 16 multiplier specified with a schematic and with an HDL file. A multiplier is a regular but complex arrangement of adders and registers which requires quite a few gates. Our example has two 16 bit inputs (A and B) and a 32 bit product

output (Y=A*B) - that's a total of 64 I/Os. This circuit requires approximately 6,000 equivalent gates.

In the schematic implementation, all the required gates would have to be loaded, positioned on the page, interconnected, and I/O buffers added. About 3 days worth of work.

The HDL implementation, which is also 6,000 gates, requires 8 lines of text and can be done in 3 minutes. *This file contains all the information necessary to define our 16x16 multiplier!*

So, as a designer, which method would you choose? In addition to the tremendous time savings, the HDL method is completely vendor-independent. That means that this same code could be used to implement a Xilinx FPGA as an LSI Logic gate array! This opens up tremendous design possibilities for engineers. For example, what if you wanted to create a 32X32 multiplier

Figure 1.9   Design Specification – Multiplier

Obviously, you would want to modify the work already done for the smaller multiplier. For the schematic approach, this would entail making 3 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 4 hours of graphical editing. For the HDL specification, it would be a matter of changing the bus references: change 15 to 31 in line 2 and 31 to 63 in line 3 (4 seconds)!

HDL File Change Example

Before (16x 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;
```

So HDL is ideal for design re-use, you can share you 'library' of parts
with other designers at your company therefore saving and avoid
duplication of effort.
I think you can see now why HDL is the way to design logic circuits!

So, now that we have specified the design in a behavioural description,
how do we convert this into gates, which is what all logic devices are
made of?
The answer is *Synthesis.* It is the synthesis tool that does the
intensive work of figuring out what gates to use based on the high level
description file provided by the designer. (Using schematic capture,
the designer has to do this all this manually). Since the resulting
netlist is vendor and device family specific, the appropriate vendor
library must be used. Most synthesis tools support a large range of
gate array, FPGA and CPLD device vendors.

In addition, the user can specify optimisation criteria that the
synthesis tool will take into account when selecting the gate-level
selection or *Mapping.* Some of these options include: optimise the
complete design for the least number of gates, optimise a certain
section of the design for fastest speed, use the best gate configuration
to minimise power, use the FPGA-friendly register rich configuration for
state machines.
The designer can easily experiment with different vendors, device
families and optimisation constraints thus exploring many different
solutions instead of just one with the schematic approach.

To recap, the advantages of high level design & synthesis are many. It
is much simpler and faster to specify your design using HLD. And
much easier to make changes to the design by the designer or
another engineer because of the self-documenting nature of the
language. The designer is relieved from the tedium of selecting and
interconnecting at the gate level. He merely selects the library and
optimisation criteria (e.g. speed, area) and the synthesis tool will
determine the results. The designer can thereby 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.

1.5   Intellectual Property (IP) Cores

Intellectual Property (IP) Cores are defined as very complex pre-tested system-level functions that are used in logic designs to dramatically shorten development time. The IP Core benefits are:

- Faster Time-to-Market
- Simplifies the development process
- Minimal Design Risk
- Reduces 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 the designer 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 PCI bus interface, DSP filter, PCMCIA interface, etc. They are extensively tested (and hence rarely free of charge) to offload the designer from having to verify the IP Core functions himself

1.6   Design Verification

To verify a programmable logic design we will probably use a simulator, which is a software program to verify the functionality and/or timing of a circuit

The industry-standard formats used ensure that designs can be re-used and there is no concerns if a vendors changes their libraries - no rework is necessary, just a synthesis recompile. Even if the customer decides to move to a different vendor and/or technology, it is just a compile away after selecting the new library. It's even design tool independent so the designer can try synthesis tools from different vendors and pick the best results!

It is more common to have cores available in HDL format since that makes them easier to modify and use with different device vendors.

After completing the design specification, you need to know if the circuit actually works as it's supposed to. That is the purpose of *Design Verification.* A simulator is used to well ... simulate the circuit.

You 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 will take this information and determine the outputs of the circuit.

Figure 1.10   The PLD Design Flow

i.   Functional Simulation

At this point in the design flow, we are doing a *Functional Simulation* which means we are only checking to see if the circuits gives us the right combinations of ones and zeros. We will do *Timing Simulation* a little later in the design flow.

If there are any problems, the designer goes back to the schematic or HDL file, makes the changes, re-generates the netlist and then reruns the simulation. Designers typically spent 50% of the 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 by passes the time-consuming synthesis process that would be required for every design change iteration. Once the circuit works correctly, we would need to run the synthesis tool to generate the netlist for the next step in the design flow - *Device Implementation.*

ii.   Device Implementation

We now have a design netlist that completely describes our design using the gates for a specific vendor/ device family and it has been fully verified. It is now time to put this in a chip, referred to as *Device Implementation.*

Translate consists of a number of various programs that are 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: optimisation, translation to the physical device elements, device-specific design rule checking (e.g. does the design exceed the number of clock buffers available in this device). It is during the stage of the design flow that 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, there is usually a listing of device and I/O utilisation, which helps the designer to determine if he has selected the best device.

iii.   Fitting

For CPLDs, the design step is called *Fitting* 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 here is if the designer has previously assigned the exact locations of the I/O pins, commonly referred to as *Pin Locking. (*Most often this is from a previous design iteration and has now been committed to the printed circuit board layout). Architectures (like the Xilinx XC9500 & CoolRunner CPLDs) that support I/O pin locking have a very big advantage. They permit the designer to keep the original I/O pin placements regardless of the number of design changes, utilisation or required performance.

Pin locking is very important when using In-System Programming - ISP. This means that if you layout your PCB to accept a specific pin out then if you need to change the design you can re-programme confident that you pin out will stay the same.

iv.   Place and Route

For FPGAs, the *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 the user does not have to worry about the intricate details of the device architecture. Some vendors have tools that allow expert users to manually place and/or route the most critical parts of their designs and achieve better

performance than with the automatic tools. *Floorplanner* is a form of such manual tools.

These two programs require the longest time to complete successfully since it is a very complex task to determine the location of large designs, ensure 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 is not enough routing tracks. If the designer faces this problem, the most common solution to is to use a larger device. And he will likely remember the experience the next time he is selecting a vendor.

A related program is called *Timing-Driven Place & Route (TDPR).* This allows users to specify timing criteria that will be used during device layout.

A *Static Timing Analyser* 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 (e.g. display all paths in the design and rank them from longest to shortest delay).

In addition, the user at this point can use the detailed layout information after reformatting, and go back to his simulator of choice 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 his 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.*

v.   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. Since 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 one per device. (Hence the term *One-Time Programmable, OTP).*

Programming of Xilinx CPLDs can be done In-System via JTAG (Joint Test Advisory Group) or using a conventional device programmer e.g. Data I/O. JTAG boundary scan – formally known as IEEE/ANSI standard 1149.1_1190 – is a set of design rules, which facilitate testing, device programming and debugging at the chip, board and system levels. In-System programming has the added advantage that devices can be soldered directly to the PCB, e.g. TQFP surface mount type devices, and if the design changes do not need to be removed form the board but simply re-programmed in-system. JTAG stands for Joint Test Advisory Group and is an industry.

Figure 1.11   Device Implementation – Download/Program

vi.   System Debug

At this point in the design flow, the device is now working but we're not done yet. We need to do a *System Debug* - verify that our device works in the actual board. This is truly the moment of truth because any major problems here means the engineer has made a assumption on the device specification that is incorrect or has not considered some aspect of the signal required to/from the programmable logic device. If so, he will then collect data on the problem and go back to the drawing (or behavioural) board!

Xilinx has the world's first WebPOWERED™ programmable logic devices!

This means we have the first WebFITTER™, you can fit your design in real time at our web site. Simply take your existing design to our

WebFITTER webpage - these files can be HDL source code or netlists - and specify your target device or your key design criteria - speed, low power etc and then press 'fit'. You will receive your results moments later via email, which includes full fitter results, design files and programming file (JEDEC file).
If you like the results you can then go on to get an on-line price.

You may then like to download your personal copy, which can be downloaded 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 which is a 3rd party simulator, chip viewer and eventually ECS schematic capture & VSS.

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

2

XILINX SOLUTION

Chapter 2 describes the products and services offered by Xilinx to ensure PLD designs enable time to market advantage, design flexibility and system future proofing. The Xilinx portfolio includes both CPLD & FPGA devices, design software, design services & support, and Cores.

2.1   Introduction

Xilinx programmable logic solutions help minimise risks for manufacturers of electronic equipment by shortening the time required to develop products and take them to market. Designers can design and verify their unique circuits in Xilinx programmable devices much faster than they could 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, defence and aerospace markets.

Leading-edge silicon products, state-of-the-art software solutions and World-class technical support make up the total solution delivered by Xilinx. The software component of this solution is critical to the success of every design project. Xilinx Software Solutions provide powerful tools which make designing with programmable logic simple. Push button design flows, integrated on-line help, multimedia tutorials, plus high

performance automatic and auto-interactive tools, help designers achieve optimum results. And 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. Such Xilinx Online Upgradable Systems would allow equipment manufacturers to remotely add new features and capabilities to installed systems or repair problems without having to physically exchange hardware.

2.2   Devices

Figure 2.2   Xilinx Devices at a Glance

2.2.1   Platform FPGAs

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

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™ for unprecedented DSP performance (up to 100 times faster than the leading DSP processor), and will offer Empower!™ processor technology for flexible high-performance system processing needs.

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

With densities ranging from 40,000 up 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.

2.2.2   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 field programmable gate arrays (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. The 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 2.3   Platform FPGAs

2.2.3   Spartan FPGAs

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

The Spartan-IIE (1.8V core) family offers some of the most advanced FPGA technologies available today, including programmable support for multiple I/O standards (including LVDS, LVPECL & HSTL), on-chip block RAM and digital delay lock loops for both chip-level and board-level clock management. In addition, the 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 2.4   Spartan IIE System Integration

Spartan-IIE Architectural Features

Figure 2.4   Spartan IIE Architecture

The Spartan-IIE family leverages the basic feature set of the Virtex-E
architecture in order to offer outstanding value. The basic CLB structure
contains distributed RAM and performs basic logic functions.
The 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 4K bits each and can be configured from 1
to 16 bits wide. Each of the two independent ports can be configured for
width independently.

The SelectI/O™ 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

Spartan-IIE Block Diagram

Figure 2.5   Spartan IIE Block Diagram

The Spartan-IIE family of Field Programmable Gate Arrays (FPGAs) is implemented with a regular, flexible, programmable architecture of Configurable Logic Blocks (CLBs), surrounded by a perimeter of programmable Input/Output Blocks (IOBs), interconnected by a powerful hierarchy of versatile routing resources. The architecture also provides advanced functions such as Block RAM and clock control blocks.

Spartan-IIE Input/Output Block

Figure 2.6   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.
The three IOB registers function either as edge-triggered D-type flip-flops
or as level sensitive latches. Each IOB has a clock signal (CLK) shared
by the three registers and independent clock enable (CE) signals for
each register.

In addition to the CLK and CE control signals, the three registers share a
Set/Reset (SR). For each register, this signal can be independently
configured as a synchronous Set, a synchronous Reset, an
asynchronous Preset, or an asynchronous Clear.

Spartan-IIE Banking of I/O Standards

Figure 2.7   Spartan IIE Banking of I/O Standards

Some of the I/O standards require VCCO and/or VREF voltages. These voltages externally are connected 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 (LC). An LC includes a four-input function generator, carry logic, and a storage element. The output from the function generator in each LC drives both the CLB output and the D input of the flip-flop. Each Spartan-IIE CLB contains four LCs, organised in two similar slices. In addition to the four basic LCs, 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 LCs.

Figure 2.8   Spartan IIE Logic Cell

Spartan-IIE function generators are implemented as 4-input look-up tables (LUTs). In addition to operating as a function generator, each LUT can provide a 16 x 1-bit synchronous RAM. Furthermore, the two LUTs within a slice can be combined to create a 16 x 2-bit or 32 x 1-bit synchronous RAM, or a 16x1-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 (Shift Register LUT) mode can be used to increase the effective number of flip-flops by a factor of 16. Adding flip-flops enables fast pipelining which are 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 organised 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 8 CLBs high will contain 2 memory blocks per column, and a total of 4 blocks.

Figure 2.9   Spartan IIE on-chip Memory

Figure 2.10   Block RAM Applications

Delay-Locked Loop

Associated with each global clock input buffer is a fully digital Delay-Locked Loop (DLL) that can eliminate skew between the clock input pad and internal clock input pins throughout the device. Each DLL can drive 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 synchronism with clock edges arriving at the input.

Figure 2.11   Spartan IIE Clock Management

	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, YQ/CS144, PQ208, FG246/456	PC84, VQ100, TQ/CS144, PQ208/240, BG256, CS280	PC84, VQ100, TQ144, PQ208/240, BG256

Figure 2.12   Spartan Family Comparison

Configuration

Configuration is the process by which the FPGA is programmed with the 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 2.13   Spartan IIE Family Overview

Figure 2.14   Spartan FPGA Part Numbering Guide

Xilinx CPLDs

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

Product Features:

XC9500 - The XC9500 In-System Programmable (ISP) CPLD families take complex programmable logic devices to new heights of high-performance, feature-richness, and flexibility. These families deliver industry-leading speeds, while giving you 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 - The CoolRunner product families offer extreme low power making them the leaders in an all new market segment for CPLDs - 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, for example, battery powered or portable applications. CoolRunner II extends the CPLD usage as it offers system level features such as LVTTL & 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 - for each part, an equivalent 'gate count' is given. This is an 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 (propagation delay through the device in nano seconds) of the device. What is the fastest sequential circuit in your design? This will tell you what fMax (Maximum frequency) 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 wish to use a socketed device, in this case 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 you 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 In-System Programming (ISP) which allows manufacturers to perform unlimited design iterations during the prototyping phase, extensive system in-board debugging, program and test during manufacturing, as well as field upgrades. Based upon

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 10,000 program/erase cycles endurance rating and 20 year data retention.

XC9500 5V Family

The XC9500™ In-System Programmable (ISP) CPLD family takes complex programmable logic devices to new heights of high-performance, feature-richness, and flexibility. This 5V family delivers industry-leading speeds, while giving you the flexibility of an enhanced customer proven pin-locking architecture along with extensive IEEE Std. 1149.1 JTAG boundary scan support. It features six devices ranging from 36 to 288 macrocells with a wide variety of package combinations that both minimise board space and maintain package footprints as designs grow or shrink. All 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

The 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 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 (pin-locking) has been demonstrated in countless real-world customer designs since the introduction of the XC9500 family. This assured pin-locking means you can take full advantage of in-system-programmability and you can easily change at any time, even in the field.

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, such as INTEST (for device functional test), HIGHZ (for bypass), and USERCODE (for program tracking), allow you the 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 analyse, 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

	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 in-system programmable family provides unparalleled performance and the highest programming reliability, with the lowest cost in the industry. The 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 and WebPACK ISE™ 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
- Up to 90 product-terms per macrocell
- Fast and routable FastCONNECT II switch matrix
- Three global clocks with local inversion
- Individual OE per output, with local inversion

Figure 2.15   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

- 3rd generation, proven CPLD technology
- Mainstream, scalable, high-reliability processing
- Fast in-system programming and erase times

Outperforms All Other 3.3V CPLDs

- Extended data retention supports longer system operating life
- Virtually eliminates in-system programming 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 test
- Free WebPOWERED software

XC9500XV 2.5V CPLD Family

The XC9500XV 2.5V CPLD family from Xilinx is based upon 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 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 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. High performance version offering pin-to-pin delays as low as 3.5ns and system frequencies as fast as 275 MHz will be available later this year. The 2.5V XC9500XV devices also include optimised support for in-system programming (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 Designers' CPLD

The advanced architecture that is employed in the XC9500XV CPLD allows for easy design integration, thus empowering the designer to fully concentrate on this 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.

XC9500XV & XC9500 XL Product Table

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
tpd (ns) XC9500XL	5	5	5	7.5
tpd (ns) XC9500XV	4	5	5	6
Registers	36	72	144	288
fSYSTEM XC9500XL	178	178	178	125
XC9500XV	200	178	178	151
Packages	PC44 CS48 VQ44* VQ64	PC44 CS48 VQ44* VQ64 TQ100	TQ100 TQ144 CS144	TQ144 PQ208 BG256 FG256* CS280*

Figure 2.16   XC9500 CPLD Part Numbering System

CoolRunner Low Power CPLDs

There are two members to the CoolRunner series, CoolRunner XPLA3 (3.3V) and CoolRunner II (1.8V). We will start by looking at the CoolRunner XPLA3 devices.

The CoolRunner™ CPLDs combine very low power with high speed, high density, and high I/O counts in a single device. The CoolRunner 3.3-volt family range 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, making them ideal for the fast growing market for 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 2.17   Sense Amplifier vs. CMOS CPLDs

The CoolRunner™ XPLA3 eXtended Programmable Logic Array 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™ (FZP) 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 <100µA (standby) without the need for special "power down bits" that negatively affect device performance.   By replacing conventional sense amplifier methods for implementing product terms (a technique that has been used in PLDs since the bipolar era) with a cascaded chain of pure CMOS gates, the dynamic power is also substantially lower than any competing CPLD. CoolRunner devices are the only TotalCMOS PLDs, as they use both a CMOS process technology and the patented full CMOS FZP design technique.

Figure 2.18   CPLD Application Trends

XPLA3 Architecture

The XPLA3 architecture features a direct input register path, multiple clocks, JTAG programming, 5 volt tolerant I/Os and a full PLA structure. These enhancements deliver high speed coupled with the best 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, there is no speed penalty for using a variable number of product terms per macrocell.
The XPLA3 family features also include industry standard IEE 1149.1 JTAG interface through In-System Programming (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.

XPLA3 Architecture

The figure below shows a high-level block diagram of the XPLA3 architecture. The XPLA3 architecture consists of logic blocks that are inter-connected by a Zero-power Interconnect Array (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 2.19   CoolRunner XPLA3 Architecture Overview

Logic Block Architecture

The figure below illustrates the logic block architecture. Each logic block contains a PLA array that generates control terms, each macrocell 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 (Variable Function Multiplexer). The VFM increases logic optimization by implementing any two input logic function before entering the macrocell.

Figure 2.20   CoolRunner XPLA3 Logic Block

Each macrocell can support combinatorial or registered inputs, preset and reset on a per macrocell basis and configurable D, T registers, or latch function. 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

The figure below shows the architecture of the macrocell used in
the CoolRunner XPLA3. Any macrocell can be reset or pre-set
on power-up.

Figure 2.21   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. There are two global
synchronous clocks that are derived from the four external clock pins.
There is one universal clock signal. The clock input signals CT[4:7]
(Local Control Terms) can be individually configured 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. When the I/O pin is
used as an output, the output buffer is enabled, 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

3-stated and the input signal will be 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, there is a path to the register to provide a fast input setup time.

I/O Cell

The OE (Output Enable) Multiplexer has eight possible modes, including a programmable weak pull-up (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

The figure overleaf shows the XPLA3 timing model which has three main timing parameters, including T_{PD}, T_{SU}, and T_{CO}. In other architectures, the user may be able to fit the design into the CPLD, but 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 and include such things as timing dependencies on the number of parallel expanders borrowed, sharable expanders, varying number of X and Y routing channels used, etc. In the XPLA3 architecture, the user knows up front whether the design will meet system timing requirements. This is due to the simplicity of the timing model.

XPLA3 Timing Model

Using combinatorial logic:

Input Pin ○──────── $t_{PD} = 5.0$ns ────────○ Output Pin

Using register logic:

Input Pin or feedback

$t_{SU} = 3.0$ns $t_{CO} = 4.0$ns ○ Output Pin

clock   Feedback to ZIA

$t_{CF} = 2.0$ns

Using Macrocell register as input register:

Input Pin $t_{SU} = 2.0$ns $t_{CF} = 2.0$ns   Feedback to ZIA

clock

Figure 2.22   CoolRunner XPLA3 Simple Timing Model

Slew Rate Control

XPLA3 devices have slew rate control for each macrocell output pin. The user has 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 WebFITTER and WebPACK ISE. Both tools are free. In addition, EDIF input for all major 3rd party software flows such as Cadence, Mentor, Viewlogic, Exemplar and Synopsys are supported.

XPLA3 Features and Benefits

Features	Benefits
• Total CMOS architecture with FZP design technology	• Lowest stand-by and total current consumption of any CPLD
• 32 to 384 macrocell device selections	• Full range of densities
• Fast pin to pin timing 32 M/C - 5 ns	• Perfect fit for high speed systems
• 3.3 Volt operation with 5 Volt tolerant I/Os	• Simplifies multivoltage system design
• Full 36 by 48 PLA - Full Programmable AND / Programmable OR structure	• Optimizes sharing and resource utilization (all product terms available)
• Bus friendly I/O	• Pull-up resistor for I/O termination
• Multiple clocking options	• Maximum clocking resources for design flexibility
• Fast input registers	• Supports direct high speed interface
• VFM (Variable Function Mux) and Fold-back NANDs	• Superior logic optimization and device fitting
• Small, surface mount packages - .8mm and .5mm ball pitch Chip Scale BGAs	• Smallest footprint and board space savings
• Advanced 5 metal layer process technology	• Lowest Cost
• Industrial and Commercial temperature ranges	• Full Industrial temperature and operating range (2.7 to 3.6 Volt)

Figure 2.23   CoolRunner XPLA3 Summary of Features and Benefits

CoolRunner XPLA3 Family

	XCR3032XL	XCR3064XL	XCR3128XL	XCR3256XL	XCR3384XL	XCR3512XL
Macrocells	32	64	128	256	384	512
Usable Gates	800	1600	3200	6400	9600	12800
tpD (ns)	5	6	6	7.5	7.5	tbd
fMAX (MHz)	175	145	145	140	127	tbd
Packages (max user I/O)	VQ44 (36) PC44 (36) CS48 (36)	VQ44 (36) PC44 (36) CS48 (40) CP56 (48) VQ100 (68)	VQ100 (84) CS144 (108) TQ144 (108)	TQ144 (120) PQ208 (164) FT256 (164) CS280 (164)	PQ208 (172) FT256 (212) FGxxx(tbd)	PQ208 (172) FT256 (212) FGxxx (tbd)

Figure 2.24   CoolRunner XPLA3 Part Number System

CoolRunner II

Xilinx CoolRunner™-II CPLDs deliver the high speed and ease of use associated with the XC9500/XL/XV CPLD family with the extremely low power versatility of the XPLA3™ family in a single CPLD. 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 In System Programming (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™ (FZP) architecture inherently delivers extremely low power performance with out the need for any special design measures. Clocking techniques and other power saving features extend the users' power budget. The design features are supported
starting with Xilinx ISE 4.1i, WebFITTER, and ISE Web-PACK.

The table show in figure 2.25 overleaf shows the CoolRunner-II CPLD package offering with corresponding I/O count. All packages are surface mount, with over half of them being 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. There are at least two densities present in each package with three in the VQ100 (100-pin 1.0mm QFP) and TQ144 (144-pin 1.4mm QFP), and in the FT256 (256-ball 1.0mm spacing FLBGA). The FT256 is particularly important for slim dimensioned portable products with mid- to high-density logic requirements.

The table 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 very unlikely that four I/O banks are needed on 32 and 64 macrocell parts, but very likely they are 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, 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 2-1,3-1 HSTL 1	LVTTL, LVCMOS 15*,18,25,33 SSTL 2-1,3-1 HSTL 1	LVTTL, LVCMOS 15*,18,25,33 SSTL 2-1,3-1 HSTL 1	LVTTL, LVCMOS 15*,18,25,33 SSTL 2-1,3-1 HSTL 1
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 2.25 CoolRunner II Family Overview

CoolRunner II Architecture Description

CoolRunner-II CPLD is a highly uniform family of fast, low power CPLDs. The underlying architecture is a traditional CPLD architecture combining macrocells into Function Blocks (FBs) interconnected with a global routing matrix, the Xilinx Advanced Interconnect Matrix (AIM). The Function Blocks use a Programmable Logic Array (PLA) configuration which allows all product tems to be routed and shared among any of the macrocells of the FB. Design software can efficiently synthesise and optimise logic that is subsequently fit to the FBs and connected with the ability to utilise a very high percentage of device resources. Design changes are easily and automatically managed by the software, which exploits the 100% routability of the Programmable

Logic Array within each FB. This extremely robust building block delivers the industry's highest pin-out retention, under very broad design conditions. The architecture will be explained by expanding the detail as we discuss the underlying Function Blocks, logic and interconnect.

The design software automatically manages these device resources so that users can express their designs using completely generic constructs without knowledge of these architectural details. More advanced users can take advantage of these details to more thoroughly understand the software's choices and direct its results.

Figure 2.26 below shows the high-level architecture whereby Function Blocks attach to pins and interconnect to each other within the internal interconnect matrix. Each FB contains 16 macrocells.

Figure 2.26 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. For a high-level view of the Function Block. At the high level, it is seen that the product terms (p-terms) reside in a programmable logic array (PLA). This structure is extremely flexible, and very robust when compared to fixed or cascaded product term function blocks. Classic CPLDs typically have a few product terms available for a high-speed path to a given macrocell. They rely on capturing unused p-terms from neighbouring 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 FB.

The PLA is different - and better. First, any product term can be attached to any OR gate inside the FB macrocell(s). Second, any logic function can have as many p-terms as needed attached to it within the FB, to an upper limit of 56. Third, product terms can be re-used at multiple macrocell OR functions so that within a FB, a particular logical product need only be created once, but can be re-used up to 16 times within the FB. Naturally, this works well with the fitting software, which identifies product terms that can be shared.

Figure 2.27   Logic Allocation – Typical PAL vs. PLA

The software places as many of those functions as it can into FBs, so it happens for free. There is no need to force macrocell functions to be adjacent or any other restriction save residing in the same FB, 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. Also, every product term arrives with the same time delay incurred. There are no cascade time adders for putting more product terms in the FB. When the FB product term budget is reached, there is a small interconnect timing penalty to route signals to another FB 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. Users can develop sum of product (SOP) logic expressions that comprise up to 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. As well, the logic function can be pure combinatorial or registered, with the storage element operating selectably 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 2.28. Note that in figure 2.28, standard logic symbols are used 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 2.28   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 (CT) are available to be shared for key functions within the FB, and are generally used whenever the exact same logic function would be repeatedly created at multiple macrocells. The CT product terms are available for FB clocking (CTC), FB asynchronous set (CTS), FB asynchronous reset (CTR), and FB output enable (CTE). Any macrocell flip-flop can be configured 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)

The Advanced Interconnect Matrix is a highly connected low power rapid switch. The AIM is directed by the software to deliver up to a set of 40 signals to each FB for the creation of logic. Results from all FB macrocells, as well as, all pin inputs circulate back through the AIM for additional connection available to all other FBs as dictated by the design software. The AIM minimises both propagation delay and power as it makes attachments to the various FBs.

I/O Block

I/O blocks are primarily transceivers. However, each I/O is either automatically compliant with standard voltage ranges or can be programmed to become so. 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 seen in Figure 2.29. Outputs can be directly driven, 3-stated or open-drain con-figured. A choice of slow or fast slew rate output signal is also available.

I/O Block Characteristics

Figure 2.29   CoolRunner II I/O Block

Output Banking

CPLDs are widely used as voltage interface translators. To that end, 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 creating product terms, which always

has a residual current component being drawn. This residual current can be several hundred milliamps, making them unusable in portable systems. CoolRunner-II CPLDs use standard CMOS methods to create the CPLD architecture and deliver the corresponding low current consumption, without doing any special tricks.

However, sometimes designers would like to reduce their system current even more by selectively disabling circuitry not being used. The patented DataGATE technology was developed to permit a straightforward approach to additional power reduction. Each I/O pin has a series switch that can block the arrival of free running signals that are not of interest. Signals that serve no use may increase power consumption, and can be disabled. Users are free to do their design, then choose sections to participate in the DataGATE function.

- Available on all input pins (except JTAG pins)
- Available for all I/O types
- Selectable on a per pin basis
- Data latch holds last valid pin value
- DataGATE allows additional power savings
  — Ability to disable active board inputs
- DataGATE can be also used for debugging and Hot Plug Input

Figure 2.30   DataGATE Function in CoolRunner II

DataGATE is a logic function that drives an assertion rail threaded through the medium and high-density CoolRunner-II CPLD parts. Designers can select inputs to be blocked under the control of the DataGATE function, effectively blocking controlled switching signals so they do not drive internal chip capacitances. Output signals that do not switch, are held by the bus hold feature. Any set of input pins can be chosen to participate in the DataGATE function.

Figure 2.30 shows how DataGATE basically works. One I/O pin drives the DataGATE Assertion Rail. It can have any desired logic function on it. It can be 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. Note that each pin has the ability to attach to the AIM through a DataGATE pass transistor, and thus 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. Note that one macrocell is singled out to drive the rail, and that macrocell is exposed to the outside world through a pin, for inspection. If DataGATE 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. Division by 2,4,6,8,10, 12, 14 and 16 are the options (see Figure 2.31). 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 no runt clocks can get through to the global clock nets. Note that again, the signal is buffered and driven to multiple traces with minimal loading and skew.

Figure 2.31   CoolRunner II Clock Division

DualEDGE

Each macrocell has the ability to double its input clock switching frequency. Figure 2.28 shows the macrocell flip-flop with the DualEDGE option (doubled clock) at each macro-cell. 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, additional power savings can be had 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 an

appreciable power drain, the clock power can be reduced by driving the net at half frequency, then doubling the clock rate using DualEDGE triggering at the macrocells.

Figure 2.32 shows how CoolCLOCK is created by internal clock cascading with the divider and DualEDGE flip-flop working together.

Figure 2.32   CoolCLOCK

Design Security

Designs can be secured during programming to prevent either accidental overwriting or pattern theft via 'readback'. Four independent levels of security are 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 detail is omitted intentionally.

Figure 2.33   CoolRunner II Device Security

CoolRunner II Application Examples

System Voltage Integration

CoolRunner-II features flexible I/O banking

DDR RAM Interface

- ◆ Data speeds exceeding **400 MHz**
- ◆ Clock doubler facilitates DDR
- ◆ Utilizes SSTL interface

CoolRunner II Application Examples – PDA Using CoolCOREs

Type	CoolCore	Reference Number	Language	Macrocells	Recommended Target Device	% Utilization
PDA	XPATH Module Design	XAPP356	VHDL	223	XCR3256XL	91.02
	8 Channel DVM Springboard	XAPP146	Pocket C	NA	XCR3256XL	NA
	Springboard Module Design	XAPP147	Pocket C	NA	XCR3256XL	NA
Bus Interface	8051 Microcontroller Interface	XAPP349	VHDL	57	XCR3064XL	89.06
	SPI	XAPP348	VHDL	127	XCR3256XL	49.61
	IrDA and UART	XAPP345	Abel, VHDL, Verilog	88	XCR3128XL	68.75
	UARTS	XAPP341	VHDL or Verilog	TBA	XCR3032XL	
	Manchester Encoder/Decoder	XAPP339	VHDL or Verilog	TBA	XCR3032XL	
	16b/20b Encoder/Decoder	XAPP336	VHDL	66	XCR3128XL	51.56
	I2C Bus Controller	XAPP333	VHDL, Verilog	120	XCR3256XL	46.08
Power Management	SMBus Controller	XAPP353	VHDL	179	XCR3256XL	69.54
Reference Designs	Wireless Transceiver	XAPP358	VHDL	168	XCR3256XL	65.63
	IRL Demo	XAPP351	VHDL	51	XCR3064XL	79.69
	MP3 Player	XAPP326	VHDL		XCR3256XL	

Free HDL CoolCore Design Code Downloadable from:
http://www.xilinx.com/products/xaw/coolvhdlq.htm

2.2.5   Military & Aerospace

Xilinx is the leading supplier of High-Reliability programmable logic devices to the aerospace and defence 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 designers 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.

2.3   Design Tools

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

Xilinx offers complete electronic design tools which enable the implementation of designs in Xilinx Programmable Logic devices. 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.

By focussing our resources on the challenges of productivity, Xilinx enables you to spend more time on the creative aspects of your design. This helps you get to market faster, and deliver a more robust product to your customers.

Engineered for Maximum Speed, Xilinx design tools give you the speed you need. With version 3.3i solutions, Xilinx place and route times are as fast as 2 minutes for our 200,000 gate, XC2S200 Spartan™-II device, and 30 minutes for our 1 million gate, system level XCV1000E Virtex™-E device. That makes Xilinx development systems the fastest in the industry.

And with the push of a button, our timing-driven tools are creating designs that support I/O speeds in excess of 800 Mbps, and internal clock frequencies in excess of 300 MHz. It's quick!

Xilinx design tools 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.

2.4   Xilinx Intellectual Property

Intellectual Property (IP) is defined as very complex pre-tested system-level functions that are used in logic designs to dramatically shorten development time. The core benefits are:

- Faster Time-to-Market
- Simplifies the development process
- Minimal Design Risk
- Reduces software compile time
- Reduced verification time
- Predictable performance/functionality

Cores are similar to vendor-provided soft macros in that they simplify the design specification step by removing the designer from gate-level details of commonly used functions. Cores differ from soft macros in that they are generally much larger system-level functions such as, PCI bus interface, DSP filter, PCMCIA interface, etc. They are extensively tested (and hence rarely free of charge) to offload the designer from having to verify the core functions himself. The Xilinx website has a comprehensive data base of Xilinx (LogiCORE™) and 3rd Party (AllianceCORE™) verified & tested cores, these can be found by interrogating the on-line search facility called the 'IP Center'.

www.xilinx.com/ipcenter

The CORE Generator™ tool form Xilinx delivers highly optimised cores that are compatible with standard design methodologies for Xilinx FPGAs. This easy-to-use tool generates flexible, high performance cores with a high degree of predictability and allows customers to download

future core offerings from the Xilinx web site. The CORE Generator™ tool is provided as part of the Xilinx Foundation™ iSE software offering.

2.5   System Solutions – Web Based Information Guide

The System Solutions section of the Xilinx website gives information about where and how Xilinx devices can be used in end applications and markets. The data ranges from application notes, white papers, reference designs, example code, industry information and much more. These System Solutions are updated very regularly so are ideal to book mark and use for research into new areas or for downloading code or design solutions to help shorten your design time to market.

The System Solutions sections on the Xilinx website are:

- eSP – Emerging Standards and Protocols
- Xtreme DSP
- Xilinx at Work
- Xilinx Online
- Configuration Solutions
- Processor Central
- Memory Corner
- Wireless Connection
- Networking Connection
- Video & Image Processing
- Computers
- Communications & Networking
- Education Services
- University Program
- Design Consultants
- Technical Support

Each of these web based sections are briefly described on the following pages.

2.5.1   eSP - Emerging Standards and Protocols Web Portal

eSP is the industry's first web portal dedicated to providing comprehensive solutions that accelerate the development of products based upon emerging standards and protocols. The Web portal features system block diagrams, reference designs, white papers, industry standards, glossary of terms and a knowledge centre. The site was designed to decrease the time spent in the pre-design phase, which has been found to be increasing and proving to be the new Achilles heel of the designer. It has been found that this phase of the design cycle involves visiting seminars, learning new standards, assimilating the data, analysing the market trends and more. The eSP web portal can save 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.

www.xilinx.com/esp

Figure 2.5.1   eSP Web Portal Contents

2.5.2   Xtreme DSP

Xtreme DSP solutions deliver the performance and flexibility needed today to quickly build the complex high performance DSP systems of tomorrow.

Driven by the broadband revolution and explosive growth in wireless, demand for new digital signal processing featuring extreme performance and great flexibility is growing faster than conventional DSP can deliver. The rapid convergence of different technology segments, such as 3G and 4G wireless communication systems, high-bandwidth networking, real-time video broadcasting, and high-performance computing systems is producing what analysts call the "The beginning of a new information technology era".

Xilinx, the recognised leader in programmable logic solutions and well established in all these technology segments, is uniquely positioned to address this new DSP paradigm now. Xilinx XtremeDSP solutions deliver the performance and flexibility you need today to quickly build the complex, high-performance DSP systems of tomorrow.

XtremeDSP can give you computing capabilities approaching 1 Tera MAC per second (1 trillion multiply and accumulate operations per second) – more than 100 times faster than conventional DSP solutions. Using our comprehensive line of industry-leading FPGAs easy-to-use tools, and optimised 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 Xilinx XtremeDSP.

2.5.3   Xilinx at Work

Providing complete system solutions, Xilinx at Work allows you to rapidly develop tomorrow's cutting-edge consumer technology, today.

Xilinx at Work showcases complete solutions of devices, software, and IP cores/services along with illustration of how Xilinx adds value in

designs for particular application segments. The applications range from IP such as FIR Filters, DES/Triple DES data encryption, QDR SRAM controllers and PCI solutions to where PLDs fit in Set Top Boxes, Smart Card Readers & Internet Audio Players (MP3).

2.5.4   Xilinx Online

Access and upgrade hardware from your desktop anywhere in the world with Internet Reconfigurable Logic (IRL™). 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 consists of robust PLD technology, your network connectivity and software design tools. Put these individual pieces together and network-based hardware upgradeability becomes a reality.

2.5.5   Configuration Solutions

The Configuration Solutions section of the Xilinx website provides easy to use pre-engineered solutions to configure all Xilinx FPGAs and CPLDs. All aspects of configuration, whether it be from a PROM for FPGAs or via In-system programming for CPLDs is explained. The section also includes 3rd part boundary scan tools, embedded software solutions, ISP cables, Automatic Test Equipment (ATE) & programmer support and configuration storage devices.
The latest edition to the configuration solutions section is the System ACE™ configuration series. With the System ACE solution, designers can easily tap into the benefits of FPGAs, using the built-in System ACE microprocessor interface to co-ordinate FPGA configuration directly with system requirements. The initial member of this series, System ACE CF, will support CompactFlash and one-inch Microdrive disk drive technology as the storage medium.

In addition to supporting configuration storage of up to 8 gigabits, System ACE CF is pre-engineered to support new capabilities that use the flexibility of reconfigurable FPGAs, including:

- Multi-board configuration from a single source
- Multi-configuration bitstream management
- Configuration updates over a network (IRL)
- Hot-swapping
- Processor core initialisation and software storage
- Encryption

With System ACE CF, designers now have a drop-in configuration solution with the density and flexibility to handle most FPGA configuration needs. The added system capabilities allow designers to use FPGAs in ways that have previously required significant additional design effort and debug time. In addition, JTAG test and microprocessor ports allow seamless integration of System ACE into any system.

Figure 2.5.5   System ACE CF flexibility and support

Flexibility

With System ACE CF, you can use one design to serve multiple applications, drastically reducing time to market. For example, rather than design several similar boards to accommodate different broadcast standards, you can design one board with multiple configurations stored in one System ACE CF memory module. Each board can be "customised" to different standards simply to setting as default the appropriate configuration stored in the ACE memory module.
You can also store multiple configurations of one design in a single System ACE CF. For example, during prototyping you can store operational, test, and debug configurations in the ACE memory module and select different configurations to prove your design.

To help manage multiple bit streams and integrate FPGA configuration control with system operation, System ACE has a built-in system microprocessor port. This port allows a system processor to change default configuration, trigger reconfiguration, directly reconfigure individual or groups of FPGAs, access non-configuration files stored in the CompactFlash module, or use excess CompactFlash memory as generic system memory.

For customers using embedded processor cores in FPGAs, System ACE CF offer a 3-in-1 solution for hardware and software management. System ACE CF can configure the FPGA fabric, initialise the microprocessor core, and supply the applications software used by the core as needed. No extra implementation hardware is required.

Density

With unprecedented density ranging to over 8Gb, one System ACE CF can configure hundreds of FPGAs and replace arrays of configuration PROMs. You can also store a large number of different designs for a given array of FPGAs all in the same memory module. Because System ACE CF uses a standard File Allocation Table (FAT) file system, you can also store non-bitstream files (e.g., release notes, technical schematics, user manuals) or use excess memory as standard system memory.

Centralisation

System ACE CF was designed to handle a variety of configuration management needs. Its flexibility and capacity allow one System ACE CF to configure a board full of FPGAs or multiple boards connected through a back-plane. This centralisation greatly simplifies configuration management and upgrades. To change or upgrade the configuration of a system, you can either remove the memory module and make the necessary alterations on your desktop PC, adjust the contents in-system through the microprocessor port, or download a new configuration over a network using Internet Reconfigurable Logic™.

2.5.6   Processor Central

Processor Central provides the information and resources you need to get the maximum benefit from our programmable solution joined with your preferred microprocessor architecture and tool set.

The market for processors in embedded systems is extremely diverse. No specific processor or particular architecture meets the needs of every application. Today's Platform FPGA must offer high levels of performance, flexibility and time-to-market for users of embedded processors. It should not force users to select one that they are either unfamiliar with or that requires them to modify their application because the solutions available are not well suited to the task at hand.

Xilinx is committed to providing solutions that offer you freedom of choice. The Processor Central website has been created to provide information and resources to help get the maximum benefit from our programmable solution joined with your preferred microprocessor architecture and tool set. This freedom of choice Empowers! you to create competitive solutions that your customers need.

2.5.7   Memory Corner

A one-stop memory shop providing solutions for leading-edge memory technology. The Memory Corner is a one-stop memory shop providing solutions for leading edge memory technology. The Memory Corner represents the collaborative efforts of Xilinx and major memory manufacturers including Cypress Semiconductor Corp., Samsung Semiconductor, IDT, Micron Technology Inc, NEC Electronics and Toshiba America Electronic Components Corp. (TAEC). 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.

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

Figure 2.5.7   Memory Solutions on the Xilinx Website

2.5.8   Wireless Connection

The Wireless Connection site provides an overview of the wireless communications industry. A brief synopsis of most of the defined, and emerging wireless standards like UMTS, CDMA-2000, etc are included. The advantages of using Xilinx FPGAs (Field Programmable Gate Arrays) in this industry are explained and solutions are provided in the form of application notes, reference designs, or cores.

Figure 2.5.8   Wireless Solutions on the Xilinx Website

2.5.9   Networking Connection

The re-configurability of Xilinx FPGAs (Field Programmable Gate Arrays) provides designers with the flexibility to implement fast, efficient, and cost-effective field upgrades. This is especially important in the networking industry, where existing standards are constantly evolving. Xilinx FPGA re-configurability allows for system upgrades resulting from new system features, bug fixes, or evolving standards without an impact on hardware or board layout. This is impossible in the ASIC world thereby making the use of FPGAs a cost-effective solution.

Another advantage of Xilinx FPGAs is that they are in-system re-configurable enabling remote field upgrades. These features enable the next generation networking products to be designed more efficiently. The Virtex family of FPGAs has architectural features that are ideal for implementing networking systems. The architectural features, such as serial Shift Register LUTs (Look Up Table) (SRL16), fast adder carry chains, and efficient multiplier implementations, make the Virtex devices very efficient at repetitive DSP algorithm implementations and high performance networking applications. The on-chip dual port RAM can be used for high-speed buffering of data. These FPGAs also have integrated support for twenty single-ended and differential I/O standards, enabling seamless interface to other devices in the system.

2.5.10   Video & Image Processing

Image and video processing environments have very specific logic requirements. Xilinx is adept at providing cores and expertise to meet the many DSP needs of these markets, including solutions to address the high resolution needs for real-time applications. We offer solutions to meet a wide range of industry standards for image processing, compression and the integration of multiple content sources. Our solution set encompasses a vast array of image and video processing applications prevalent in entertainment and business today, including:

- Video broadcast: DTV and HDTV
- Video conferencing and other multimedia business applications
- Medical image processing
- Industrial imaging, such as robotic vision, pattern recognition, and security
- 3D graphics, image rendering
- Flight simulators
- Digital photography, film scanning systems, movie editing equipment
- Digital copiers and handlers

2.5.11   Computers

Xilinx offers superior system level solutions to meet the needs for today's demanding computer applications. Xilinx and its partners deliver building

blocks and expertise to meet the spectrum of needs that include fast time-to-market, portability, high performance processing and throughput, and most importantly low cost. Spartan FPGAs helps meet the rigid cost constraints by offering an entire family of devices (up to 40,000 programmable system gates) that are all priced under $10.00 eliminating the necessity of designing a custom IC. Xilinx delivers system level solutions for applications including:

- Computer peripherals - storage devices, speciality printers, point of sale terminals, speciality data capture
- PC add-in cards: peripheral controllers, multimedia, network interface, algorithm specific or general purpose acceleration cards

2.5.12   Communications & Networking

Xilinx and its partners provide the building blocks and expertise for many communications applications. The core solutions and consulting services we provide help customers accelerate time to market, keep pace with industry standards, and address the industry's demands for high performance and low power solutions. Xilinx FPGAs have been used to implement system level building blocks for a variety of applications, including:

- Wireless: spread spectrum, satellite modems, cellular/PCS base
- stations, military radios and wireless local loop
- Cable: modems, spectrum management and test equipment
- xDSL technologies for high speed data over copper
- Telecommunications
- Networking
- Communications test equipment

2.5.13   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 will allows you to implement your own design in less time with more effective use of the devices. Not only design engineers, but also test engineers, component engineers, CAD engineers, technicians and engineering managers may want to participate in the training in order to understand the Xilinx products. Learning services provides a number of courses in a variety or delivery methods.

Live E-Learning Environment

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 world-wide access. Live instructors present the modules in real time. During each session, you will be able to interact with the instructor as well as collaborate with online subject experts.

Day Segment Courses

Xilinx continues to develop and instruct traditional day length courses. Working with various Xilinx product development groups, new courses are created and made available to reflect the current product releases. This serves to make training available when you need it and on the products you need it for. These classes are held in centres around the world. Specific onsite instruction is also available at your facility. For more information: www.support.xilinx.com and click on Courses under Education.

Computer Based Training (CBT)

Xilinx introduced computer based training with Verilog CBT. Verilog CBT will allow you to learn the Verilog language at your own pace without ever leaving your office. Verilog CBT is based on the traditional 3-day course, converted into a computerised self-study program.
For more information please email: eurotraining@xilinx.com or telephone: +44 (0)870 7350 548 or visit:

www.xilinx.com/support/education-home.htm

2.5.14   University Program

The mission of the Xilinx University Program (XUP) is to promote Xilinx as the technology of choice in the Academic community. The XUP has provided donations, discounted products, and services to universities since 1985. Today there are over 1600 universities using Xilinx in class labs, or about 18% of all of the engineering universities World-wide.

The resources available to Universities and education include:

Xilinx University Resource Centre

http://xup.msu.edu//

Developed and maintained by the Department of Electrical and Computer Engineering at Michigan State University, this site is designed specifically to support and encourage universities using Xilinx products in the classroom. You will find references and resources regarding everything from hardware data sheets to tutorials on using the Xilinx search engine effectively. Vast amounts of time and energy can be saved by using the resources contained within these pages.

Xilinx Answers Data Base:

http://www.xilinx.com/support/searchtd.htm

Xilinx Student Edition Frequently Asked Questions:

http://university.xilinx.com/univ/xsefaq1.htm

2.5.15   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 world wide program that allows easy access to certified experts in Xilinx devices architectures, software tools and cores. XPERTS partners also offer consulting in the areas of HDL synthesis and verification, customisation and integration, system level designs and team based design techniques. A listing of the partners in the Xilinx XPERTS program is located on the Web at:

www.xilinx.com/ipcenter

For more information on Xilinx Products and Services please look in the **Xilinx Data Source CDROM** in the back of the book or visit our website:

www.xilinx.com

2.5.16   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, users with a valid service contract can access Xilinx engineers over the Web by opening a case against a specific issue. For technical support on the web, log on to:

www.support.xilinx.com

<div style="border: 2px solid black; display: inline-block; padding: 20px 30px;">

3

</div>

WebPACK ISE DESIGN SOFTWARE

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

3.1   Introduction

The individual WebPACK ISE modules give the user the ability to tailor the design environment to the chosen programmable logic devices to be implemented and the preferred design flow.

In general, the design flow for FPGAs and CPLDs is the same. The designer can choose whether to enter the design in schematic form or HDL such as VHDL or ABEL. The design can also comprise of a mixture of schematic diagram with embedded HDL symbols. There is also a facility to create state machines in a diagrammatic form and let the software tools generate optimised code from a state diagram.

For simulation, WebPACK ISE incorporates a Xilinx version of ModelSim from Model Technology, 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 offers an easy to use Graphical User Interface (GUI) to visually create a test pattern. A testbench is then generated and is compiled into MXE along with the design under test.

The diagram 3.1 overleaf gives an indication of the design flow. The various modules show the requirements for the device targeted.

Figure 3.1   WebPACK Design Flow

When the design is complete and the designer is happy with the
simulation results, the design is targeted at the required device.

 For FPGAs the implementation process undertakes 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 utilises
   the routing resources.
4. Configure – Creates a programming bitstream.

For CPLDs the implementation process is as follows:

1. Translate – Interprets the design and runs a 'design rule check'.
2. Fit – Allocates the Macrocell usage
3. Configure – Creates a JED file for programming.
The design is then ready for programming into the device.

3.2   Module Descriptions

i.   WebPACK Design Entry

This module must be installed regardless of the device family targeted or
chosen design flow. The design entry module incorporates the Project
Management functionality, the XST synthesis tool and the basis of the
schematic entry package. (Even schematic designs are synthesised
through XST)

ii.   WebPACK ECS Library

This module comprises of the schematic library primitives for the XC9500
and CoolRunner CPLDs as well as all supported FPGAs.

iii.   WebPACK StateCADÒ

StateCad is an optional tool for graphically entering state machine in 'bubble diagram' form. The user simply draws the states, transitions and outputs. StateCad gives a visual test facility. State machines are generated in HDL and then simply added to the WebPACK ISE project.

iv.   WebPACK MXE Simulator

Modeltech Xilinx Edition (MXE) is the module for both functional and timing simulation. The necessary libraries are already pre-compiled into MXE and pre-written scripts seamlessly compile the design to be tested and its testbench.

For functional simulation the written code is simulated prior to synthesis. After fitting (CPLDs) or Place And Route (PAR) (FPGAs), the design can be simulated using the same original testbench as a test fixture, but with logic and routing delays added.

v.   WebPACK HDL Bencherä

The HDL Bencher™ generates the previously mentioned testbenches allowing the design under test to be stimulated. The HDL bencher reads the design under test and the user enters signal transitions in a graphical timing diagram GUI. The expected simulation results can also be entered allowing the simulator to flag a warning if the simulation did not yield the expected results.

vi.   WebPACK Spartan & Virtex Fitters

These modules give access to the FPGA implementation and synthesis files. It is required for all Spartan II, Spartan-IIE, Virtex-E and Virtex-II designs.

vii.   WebPACK CPLD Fitter

This module gives access to all the XC9500, XC9500XL, XC9500XV, CoolRunner and CoolRunner-II device files and fitting programs.

viii.   WebPACK iMPACT Programmer

For all devices available in WebPACK, the iMPACT Programmer module allows a device to be programmed in-system. (A JTAG cable must be connected to the PC's parallel port.)
For FPGAs the programmer module allows a device to be configured via the JTAG cable. Xilinx FPGAs are based on a volatile SRAM technology, so the device will not retain configuration data when power is removed. Therefore this configuration method is normally only used for test purposes.

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 by the implementation phase and provides an MCS or HEX formatted file used by PROM programmers.

xi.   WebPACK ChipViewer

The ChipViewer module can be used to examine the fitting and pin out of all XC9500 and CoolRunner family CPLDs.

xii.   WebPACK Device Support

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

3.3   WebPACK CDROM Installation Instructions

As the WebPACK ISE software is modular there may be no need to install all of the applications. It is however recommended that all modules are installed from the CD if possible.

1.   Insert the CD and using Windows Explorer navigate to the CD drive.

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

The InstallShield Wizard window will appear as shown below:

3.   Select from the installation methods shown, either 'Typical Installation' or 'WebPACK Live'.

WebPACK Live – WebPACK ISE is run from the CD with a minimal set of files loaded onto your hard drive. This method of operation has a 7-day grace period before CD registration is required. Designers can continue to run the software from the CD beyond this point if so desired by obtaining a CD Key. The CD Key is free and available to new and registered WebPACK users.

The Typical Installation - The desired files are installed to you hard drive. This requires the user to obtain a CD Key. The CD Key is free and available to new and registered WebPACK users.

4.   After selecting which installation method you require you will see the following window:

Either enter you unique CD Key from a previous installation obtain a CD Key from:

www.xilinx.com/sxpresso/webpack.htm

When at the registration web page:
Follow the on-line registration process by selecting New customer please register from the first on-line screen. Enter the data requested at each

stage. You will need to create and enter a memorable user name and password.

When requested enter your product ID code (from your WebPACK CD cover – it begins DWB) in the appropriate field.

Your CD Key number will then be sent to you via email (please ensure that you have carefully entered your correct email address when entering your details).
Your key number will look something like this:

                          2504-xxxx-xxxx

To proceed with the installation enter your key number into the InstallShield Wizard CD Key window and select the 'next' button.

5.   Select the WebPACK modules you wish to install from the following:

Design Entry, ECS Library, Chip Viewer, CPLD Fitter, FPGA Fitter (Spartan and/or Virtex device support), CPLD Fitter, iMPACT Programmer (CPLD and/or FPGA), HDL Bencher, State CAD & ModelSim XE.

The following table gives the minimum install options for each required flow:

	HDL Entry	Schematic	Simulation	State Machines
FPGA	Design Entry Spartan Fitter Virtex Fitter FPGA Prog.	fpga schem lib	hdl_bencher mxe_simulator	statecad
XC9500	Design Entry 9500 Fitter CPLD Prog.	cpld schem lib	hdl_bencher mxe_simulator	statecad
CoolRunner	Design Entry CPLD Fitter CPLD Prog.	cpld schem lib	hdl_bencher mxe_simulator	statecad

If you have enough disk space it is recommended that you install all modules available.

3.3.1   Getting Started

Licenses

The HDL bencher and the MXE simlulator have associated licenses.

HDL Bencher will give limited performance until the application has been
registered. The registration process is automated. When using the
bencher for the first time at the export HDL stage, a window will pop up
asking for registration information (Name, address etc.) The application
creates a host ID which is used to create a password. A password will be
emailed back on application.
An upgrade can also be requested via email. The address is given when
using the bencher.

MXE Simulator is licensed via FlexLM. It requires a soft license file to be
situated on the hard drive pointed to by a *set lm_license_file* environment
setting.
The license is free and is applied for on line after installation.
A license.dat file will be emailed back. The license is valid for 30 days
after which period it will be necessary to reapply. From the Start menu,
Programs > ModelSimXE 5.xx > Submit License Request.

Design Entry Options

On installation of all the modules design entry options are as follows:

	VHDL	Abel	Verilog	ECS	External
FPGAs	v		v	v	x
XC9500	v	v	v	v	v
XPLA	v	v	v	v	v

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

```
c:\Xilinx_WebPACK\bin\nt
```

Create a unique directory on your hard drive for working on projects e.g. c:\designs. If you need to reinstall WebPACK ISE for future releases it is recommended that the entire WebPACK ISE directory structure is deleted.
The external option for design entry refers to a third party design tool output netlist. In this case an EDIF netlist file is generated externally and is implemented by the WebFITTER.

Summary

In this chapter the functions of all the WebPACK ISE modules have been explained along with installation of the modules you require.
You can decide which modules are necessary for your intended design and install only relevant modules. The next section will take you through your first PLD design using the powerful features of the WebPACK ISE software. The example design is a simple traffic light controller which uses a VHDL counter and a state machine. The design entry process is identical for FPGAs and CPLDs.

<div style="border: 2px solid black; display: inline-block; padding: 20px;">

4

</div>

WebPACK ISE DESIGN ENTRY

4.1   Introduction

This section is a step by step approach to your first simple design. The following pages are intended to demonstrate the basic PLD design entry and implementation process.

In this example tutorial a simple traffic light controller is designed in VHDL. The design is initially targeted at a Spartan-IIE FPGA, but the same design entry principles can also be applied to both XC9500 and CoolRunner CPLDs.

CPLD Users
This design entry section also applies to CPLDs. Any additional CPLD specific information is included in italic font.

4.2   Design Entry

Start WebPACK ISE Software
Select Start > Programs > Xilinx WebPACK > WebPACK Project
Navigator

Create a New Project
Select File -> New Project…

Enter the following into the New Project dialogue box:

```
Project Name:        Traffic
Project Location:c:\Designs\Traffic
Device Family:       Spartan2e
Device:              2S100E FT256-5
Synthesis Tool:      XST VHDL
```


Figure 4.2.1   Project Properties Window
CPLD designs
*Other device families can be chosen here including CPLDs. For CPLD
designs the synthesis tool can also be ABEL. Even if the flow is intended
to be purely schematic, the schematic diagram will be converted into HDL
and synthesised through the chosen synthesis tool.*

To name your project, right-click on **Untitled** and select **Properties**.
Enter **Traffic** in the **Project Title** window.

Create a 4-bit Counter Module

Use the Language Templates to create a VHDL module for a counter as

follows:

From the **Project** menu select **New Source**.

Select **VHDL Module** as the source type and give it a file name **counter**.

Click the **Next>** button.

Fill out the **source definition** box as follows and then click **Next.**

Figure 4.2.2   Define VHDL Source Window

This table automatically generates the entity in the counter VHDL
module.

Click the **Finish** button to complete the new source file template.

Notice a file called counter.vhd has been added to the project in the sources window of the project navigator.

Figure 4.2.3   Counter Window

As the project builds you will notice how WebPACK ISE manages hierarchy and associated files in the sources window.
Double clicking on any file name in the sources window will allow that file to be edited in the main text editor.

Figure 4.2.4 Source in project Window

The Language Template

The language template is an excellent tool to assist in creating HDL code. It has a range of popular functions such as counters, multiplexers, decoders and shift registers to assist the designer. There are also 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 customised for their intended purpose. Usually, it is necessary to change the bus width or names of the signals or sometimes modify the functionality. In this tutorial the template uses the signal name 'clk' and 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 of the toolbar.

The language template can also be accessed 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 alternative method is to place your cursor between the **begin** and **end** statements in counter.vhd, select Counter in the **VHDL > Synthesis Templates** folder and the click the *Use in counter.vhd* button in the Language Templates toolbar.

Close the Language Templates.

Notice the colour 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 and the port definitions required if this counter was used in its entirety. As the entity has already been created, this information is not required

Delete the Green Comments

The counter from the template shows a loadable bi-directional 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 the section**

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

- **Delete the section**

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

The counter module should now look like figure 4.2.5 overleaf.

```
counter *                                                        _ □ ×
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity counter is
    Port ( clock : in std_logic;
           reset : in std_logic;
           count : inout std_logic_vector(3 downto 0));
end counter;

architecture behavioral of counter is

begin

process (CLOCK, RESET)
begin
   if RESET='1' then
      COUNT <= "0000";
   elsif CLOCK='1' and CLOCK'event then
        COUNT <= COUNT + 1;
   end if;
end process;

end behavioral;
```

Figure 4.2.5   Counter in VHDL Window

The above design is a typical VHDL module. It 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 the ports associated with the design. Count (3 down to 0) means that count is a 4-bit logic vector. This design has 2 inputs clock and reset, and 1 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 increments 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 it is evaluated before the clock action.

The area still within the Architecture but before the begin statement is where declarations reside. There will be examples of both component declarations and signal declarations later in this chapter.
Save the counter module.

The counter module of the design can now be simulated.

With counter.vhd highlighted in the sources window, the process window will give all the available operations for that particular module. A VHDL file can be synthesised 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 which can be simulated.

4.3   Functional Simulation

To simulate a vhdl file it is necessary to 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 4.3.1   New Source Window

Click **Next**.

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 **Next**. Review the information and click **Finish**.

The HDL bencher tool reads in the design. The Initialise Timing box sets the frequency of the system clock, set up requirements and output delays.

Set **Initialise 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 4.3.1   HDL Bencher Window

Note: 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 available values for that. Open a pattern text field and button by double clicking on a signals 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** shown

below.

Click OK to accept the parameters.

Figure 4.3.2   Pattern Wizard Window

Your waveform should look like the following:

Figure 4.3.3   Waveform Window

Click [Save Waveform] **to save the waveform**.

Close HDL Bencher.

The ISE source window should look like the following:

Figure 4.3.4   New Sources in Project Window

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

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

Select counter_tb.tbw in the ISE source window. In the Process window

expand Modelsim Simulator by clicking and then **right-click Simulate**

Behavioural VHDL Model.

Select **Properties.**

In the **'Simulation run time'** field type **–all**, 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 Behavioural VHDL**

Model. This will bring up the Model Technology MXE dialog box.

Note: ISE automates the simulation process by creating and launching a simulation macro file (a .do file, or in this case a .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.

Select **ModelSim** for the dialog box.

Maximise the **Wave window** and from the Zoom menu select **Zoom Full:**

Figure 4.3.5   Wave Window
Close the wave window in Figure 4.3.5 and the **Modelsim** simulator.

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

Figure 4.3.6   Project Snapshot Window

Note: Taking a snapshot of your project saves the current state of your project in a subdirectory with the same name as the Snapshot name so 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 was to have 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.

4.4   State Machine Editor

For the traffic light design, the counter will act as a timer that determines the transitions of a state machine.
The state machine will run through 4 states, each state controlling a combination of the three lights.

State1 – Red Light
State2 – Red and Amber Light
State3 – Green Light
State4 – 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** and click Next, then finish.

Figure 4.4.1   New Source Window

Open the State Machine Wizard by clicking in the [Draw State Machines] button
on the main toolbar.

Figure 4.4.2   State Machine Wizard Window

Set the Number of states to **4** and hit **Next.**

Click **Next** to build a **synchronous** state machine.

In the **Setup Transitions box**, type **TIMER** in the Next: state transition

field. (Shown in Figure 4.4.3).

Figure 4.4.3   Set-up Transitions Window

Click on finish and drop the state machine on the page.

Double Click on the Reset State 0 coloured 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 4.4.4 below.

Figure 4.4.4   Logic Wizard Window

Click on OK and then OK the Edit State box.

In a similar fashion edit the other states.

Rename State1 to REDAMB and use the output wizard to set RD = 1

and a new output AMB equal to 1 with a registered output.

Rename State2 to GREEN and use the output wizard to set a new

output GRN equal to 1 with a registered output.

Rename State3 to AMBER and use the output wizard to set AMB = 1.

The state machine should look like the following.

Note: If you set a signal as registered in the output wizard then select

signal and re-open wizard – it is no longer ticked as registered.

Figure 4.4.5   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 4.4.6   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 side 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 4.4.7   Edit Vector Window

Click OK.

Your completed state machine drawing should look like the Figure 4.4.8 overleaf.

Figure 4.4.8   State Machine Drawing

Click on the [WHEN SA→ CT←CT-1; 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_cad.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 4.4.9   Source in Project Window showing Model View

4.5   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 whilst others like to keep the design entirely text based.
This section discusses the latter, hence the counter and state machine
will be connected using a top.vhd file.
If you prefer the former, jump directly to the next section, 4.6, entitled
'Top Level Schematic Designs'. There is the opportunity to do both by
continuing through the tutorial.

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

Figure 4.5.1   Project snapshot

From the Project Menu select New Source and create a VHDL Module
called top.

Figure 4.5.2   New Source Window Showing VHDL Module

Click on next and fill out the 'Define VHDL Source' dialog box as shown below in figure 4.5.3:

Figure 4.5.3   Define VHDL Source Window

Click on Next, then Finish.

Your new file, top.vhd should look like figure 4.5.4 below:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity top is
    Port ( clock : in std_logic;
           reset : in std_logic;
           red_light : out std_logic;
           amber_light : out std_logic;
           green_light : out std_logic);
end top;

architecture behavioral of top is
```

Figure 4.5.4   New VHDL File

In the Sources Window highlight counter.vhd

In the Process Window double click **View 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 4.5.5   Instantiation Template

Close the Instantiation Template as shown in figure 4.5.5.

Paste the Component Declaration and Instantiation into top.vhd.

Re-arrange so that the Component Declaration lies before the begin

statement in the architecture and the instantiation lies between the begin

and end statement. (Use the picture on the next page for assistance).

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 above.

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 up the counter and state machine instantiated modules so your top.vhd file looks like figure 4.5.6 below:

```
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 4.5.6   top.vhd File

Save top.vhd and 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.

The entire design can now be simulated.
Add a new **Test Bench Waveform** source as before but this time,
associate it with the module **top.**

Accept the timing in the Initialise 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 4.5.7   Waveform Diagram

Close the **Edit Test Bench** window.

Click the **Save Waveform** button.

Close HDL Bencher.

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

Simulate **Functional VHDL Model** in the **Process** Window.

Figure 4.5.8   Waveform Window

You are now ready to go to the implementation stage.

4.6   Top Level Schematic Designs

It is sometimes easier to visualise designs when they have a schematic top level which instantiates the individual blocks of HDL. The blocks can then be wired together in the traditional method.

For designs in WebPACK ISE, 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 4.6.1 below (two modules with no top level file). This is achieved by:
At the bottom of Sources window select the **Snapshot View** Tab.
Highlight **Snap2** (two modules), then in the Project menu select **Replace with Snapshot.** 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 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 the following figure:

Figure 4.6.1   Sources in Project Window

4.6.1   ECS Hints

The ECS schematic capture program is designed around the user
selecting the action they wish to perform followed by the object the action
is to be performed on. 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 4.6.2   New Source Window showing top_sch

Click **Next** then **Finish**.

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 Drawing Toolbar is permenantly located on the right hand side of the ECS page.

Add the counter and state machine by clicking on the new library in the **Categories** window in the top right of the ECS page, 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 window looks like the following:

Figure 4.6.3   Close Up of Counter and State Machine Symbols

Select the **Add Wire** 🔲 tool from the **Drawing Toolbar**

Note: Click once on the symbol pin, once at each vertex and once on the destination pin to add a wire between two pins. ECS will let the user know that a net can be attached to a port by highlighting it with a red square.

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

Wire up the counter and state machine as shown below in figure 4.6.4:

Figure 4.6.4   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 of the ECS page) and then **place** the **net name** on the end of the clock wire.

Note: 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 your schematic looks similar to the following figure:

Figure 4.6.6   More Net Names

ECS recognises that count(3:0) and TIMER(3:0) are buses 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** that

you want to add input markers to.

Repeat for the outputs but select Output type.

Your completed schematic should look like the following figure, 4.6.7:

Figure 4.6.7   Adding I/O markers

Save the design and exit the schematic editor.

Note: 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.

The entire design can now be simulated.

Highlight **top_sch.sch** in the sources window

Add a new Test Bench Wavefrom 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 **Initialise 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 4.6.8   Waveform Diagram

Close the **Edit Test Bench** window**.**

Click the **Save Waveform** button.

Close HDL Bencher.

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 4.6.9   ModelSim Simulation Window

You are now ready to go to the implementation stage.

Summary

This section covered the following topics

- Hierarchical VHDL structure and simple coding example
- Test Bench Generation
- Functional Simulation
- The State Machine Editor
- ECS Schematic Capture

The next Chapter discusses the Synthesis and implementation process for FPGAs. CPLD users may wish to skip the next chapter. For those intending to target a CPLD, the Constraints Editor and Translate information may be of interest.

5

IMPLEMENTING FPGAs

5.1   Introduction

After the design has been successfully simulated 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 normally 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 first step is *translate*. The translate step checks the design and ensures the netlist is consistent with the chosen architecture. Translate also checks the user constraints file (UCF) for any inconsistencies. In effect, this stage prepares the synthesised design for use within an FPGA.

The Map stage distributes the design to the resources in the FPGA. Obviously, if the design is too big for the chosen device the map process will not be able to complete its job.

Map also uses the UCF file to understand timing and may sometimes decide to actually add further logic (replication) in order to meet the given timing requirements. Map has the ability to 'shuffle' the design around look up tables to create the best possible implementation for the design. This whole process is automatic and requires little user input.

The Place And Route (PAR) stage works with the allocated configurable logic blocks (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 purely to minimise the path length. The routing resources are then allocated to each connection, again using careful selection of the best possible routing types. E.g. if a signal is needed 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 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. Whilst developing a design it may not be necessary to create a bit file on every implementation as the designer may just need to ensure a particular portion of the design passes any timing verification.

The steps of implementation must be carried out in this order. The WebPACK ISE software will automatically perform the steps required if a particular step is selected. E.g. If the design has only just been functionally simulated and the designer then decides to do a timing simulation, WebPACK ISE will automatically Synthesise, Translate, Map and 'PAR' 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 each step required to successfully implement the Traffic Light design in the previous chapter.

5.2   Synthesis

The XST synthesis tool will only attempt to synthesis the file highlighted in the sources window. In the 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 recognises all the lower level blocks used in the top level code and synthesises them all together to create a single bitstream.

In the **Sources** window ensure **top.vhd** (top_sch for schematic flows) is highlighted.
In the **Process** window expand the Synthesis sub-section by clicking on the **+** next to Synthesize.
You can now check your design by double clicking on **Check Syntax**.
Ensure any errors in your code are corrected before you continue. If the syntax check is OK a tick will appear.

The design should be OK because both the HDL 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 5.2.1   Process Window showing Check Syntax

Right Click on **Synthesize** and select **Properties**.

A window appears allowing the user to influence the way in which the design is interpreted.
The help feature will explain each of the options in each tab.
Click on the **HDL options** Tab.

The Finite State Machine (FSM) encoding algorithm option looks for state machines and determines the best method of optimising.
For FPGAs state machines are usually 'one hot' encoded. This is due to 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 the 'Add IO Buffers' box is ticked. The IO 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.

OK the **Process Properties** window and double click on **Synthesize.**

When the synthesis is complete a green tick appears next to Synthesize. Double Click on **View Synthesis Report.**

The first section of the report just summarises the synthesis settings. Each entity in the design is then compiled and analysed.
The next section in the report gives the synthesis details and documents how the design has been interpreted.
It can be seen 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 have been inferred.

```
Analyzing Entity <top> (Architecture <behavioral>).
Entity <top> analyzed. Unit <top> generated.

Analyzing Entity <counter> (Architecture <behavioral>).
Entity <counter> analyzed. Unit <counter> generated.

Analyzing Entity <stat_mac> (Architecture <behavior>).
Entity <stat_mac> analyzed. Unit <stat_mac> generated.

Analyzing Entity <shell_stat_mac> (Architecture <behavior>).
Entity <shell_stat_mac> analyzed. Unit <shell_stat_mac> generated.

Synthesizing Unit <shell_stat_mac>.
    Related source file is c:/nick/designs/my_design/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 c:/nick/designs/my_design/STAT_MAC.vhd.
Unit <stat_mac> synthesized.
```

Figure 5.2.2   Extract of Synthesis Report

The final results section shows the resources used within the FPGA.

```
Cell Usage :
# BELS                              : 34
#       GND                         : 1
#       LUT1                        : 4
#       LUT2                        : 2
#       LUT3                        : 1
#       LUT3_D                      : 2
#       LUT3_L                      : 5
#       LUT4                        : 6
#       LUT4_L                      : 1
#       MUXCY                       : 3
#       MUXF5                       : 5
#       VCC                         : 1
#       XORCY                       : 3
# FlipFlops/Latches                 : 10
#       FD                          : 6
#       FDC                         : 4
# Clock_Buffers                     : 1
#       BUFGP                       : 1
# IO_Buffers                        : 4
#       IBUF                        : 1
#       OBUF                        : 3
```

Figure 5.2.3   Resource Report

5.3   Constraints Editor

To get the ultimate performance from the device it is necessary to tell
the implementation tools what and where performance is required. This
design is particularly slow and timing constraints are unnecessary.
Constrains can also be physical and pin locking is a physical constraint.
For this design, assume the specification for clock frequency is 100MHz
and the pin out has been pre determined to that of a Spartan-IIE pre
designed board.

In the **Process window** expand the **Design Entry Utilities** section then
expand the **User Constraints** sub section.

Figure 5.3.2   Process Window showing User Constraints

Double click on **Edit Implementation Constraints File.**

Notice the Translate step in the Implement Design section runs
automatically. This is because the implementation stage must see the
netlist before it can offer the user the chance to constrain sections of the
design. When 'Translate' has completed the Constraints Editor Opens.

There is one global net in the design, this is the clock. Translate
detected the clock assigned it to the global tab.

Double Click in **Period field**.
Give the clock a Period Constraint of 10ns with a 50% duty cycle as
follows.

Figure 5.3.2   Clock Period Editor Window

A period constraint ensures the internal paths stating and ending at
synchronous points (Flip-Flop, Ram, Latch) have a logic delay less than
10ns.

OK the clock period and hit the **Ports tab**
The ports section lists all the IO in the design. The location field sets
which pin on the device the signal will connect to.

Double click in the **location** field for amber_light. Then, in the location
dialogue box, type **G16.** (If a non-Ball Grid package is used, such as a
PQ208, the syntax is slightly different. The correct syntax for each
package can be found in the online datasheet).

Repeat for the other outputs, the Clock and Reset input.

amber_light	G16
Clock	T9
green_light	G15
red_light	H16
Reset	H13

Highlight the three outputs 'red_light', 'green_light' and 'amber_light' using **ctrl select**.

Figure 5.3.3   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 dialogue box set the time requirement to 15ns relative to the clock. (There is only one clock but in some designs there may be more).

Figure 5.3.4   Clock to Pad Dialogue Box

Hit OK and notice that the clock to pad fields have been filled in automatically. Also notice that the User Constraints File (UCF) generated has appeared in the UCF constraints tab at the bottom of the screen. The UCF file should look similar to the following:

```
NET "clock" TNM_NET = "clock";
TIMESPEC "TS_clock" = PERIOD "clock" 10 ns HIGH 50 %;
NET "amber_light" LOC = "G16";
NET "clock" LOC = "T9"; #global clock
NET "green_light" LOC = "G15";
NET "red_light" LOC = "H16";
NET "reset" LOC = "H13";
INST "amber_light" TNM = "lights";
INST "green_light" TNM = "lights";
INST "red_light" TNM = "lights";
TIMEGRP "lights" OFFSET = OUT 15 ns AFTER "clock";
```

Save the Constraints Editor session.

Translate must be re-run so the new constraints can be read. OK the **'run translate'** window and exit the constraints editor and hit **reset** in the **notice** window.

Click on the **+** next to **Implement Design** in the **Process window**.

Figure 5.3.5   Design Process Window

The implementation steps are now visible. The green tick next to translate indicates this step has completed once before.

A right Click on each step allows the user to edit the properties for that particular step. The properties for all the steps can be edited by right clicking on Implement Design. There is a tab for each step.

Figure 5.3.6   Process Properties

The **help** button will explain the operation of each field.
Implement the design by double clicking on **Implement Design.** (Each
stage could be run separately if required).

When there is a **green tick** next to **Translate, Map** and **Place and
Route** the design has completed the implementation stage. For a 'post
route' timing report manually run the **Generate Post-Route Static
Timing** section.

Figure 5.3.7   Generate Post-Route Timing

5.4   Reports

Each of the stages has its own report. Clicking on the **+** next to each stage lists the reports available. The various reports available are as follows:

i. Translate Report – Shows any errors in the design or the UCF.

ii. Map Report – Confirms the resources used within the device. A detailed map report can be chosen in the Properties for map. The detailed map report describes trimmed and merged logic. It will also describe exactly where each portion of the design is located in the actual device.

iii. 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 the timing constraints the additional delay added by routing will only add to the problem.
If there was no routing delay these traffic lights would run at 216 MHz!!

iv. 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.

v. Asynchronous Delay Report – is concerned with the worst path delays in the design, both logic and routing.

vi. Pad Report – Displays the final pin out of the design with information regarding the drive strength and signalling standard.

vii. Post Place and Route Static Timing Report – Adds the routing delays. It can now be seen that the max frequency of the clock has dropped to 135MHz.

WebPACK has additional tools for complex timing analysis and floor planning. Neither of these tools are covered in this introductory booklet.

5.5   Timing Simulation

The process of timing simulation is very similar to the functional method.

With **top_tb.tbw** or (top_sch_tb.tbw for schematic flow) selected in the sources window, expand the **Modelsim Simulator** section in the Process window and rightclick on **Simulate Post-Place and Route VHDL model**.

Select **Properties** and in the **Simulation Run Time** field type **'all'.**

Click **OK** then double click on **Simulate Post Route VHDL model**

MXE opens but this time a different script file is implemented and the 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 FPGA and takes timing information from a separate file.

Use the **Zoom** features and **Cursors** to measure the added timing delays.

Figure 5.5.1   Simulation Window showing Timing

5.6   Configuration

Right click on **Generate Programming file** and then click on **Properties**. Under the Startup Options tab, ensure that the Start-Up 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 which can be used by the iMPACT programmer to configure a device.

Expand the **Generate Programming File** tools sub section.
Double Click on **Configure Device (iMPACT)**.

A DLC5 Parallel JTAG cable or a MultiLINX™ cable is required to configure the device from the iMPACT Programmer. Ensure the cable is

plugged in to the computer and the flying leads are connected properly to the device and power supply.

Cable	Device on Board
Vcc	5v, 3.3v or 2.5v
GND	GND
TDI	TDI Pin
TDO	TDO Pin
TMS	TMS Pin
TCLK	TCLK Pin

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.

The next chapter details a similar process but this time a CoolRunner CPLD is targeted rather than a Spartan-IIE FPGA. FPGA users may wish to skip the next chapter.

<div style="border: 2px solid black; display: inline-block; padding: 20px;">

6

</div>

IMPLEMENTING CPLDs

6.1   Introduction

After the design has been successfully simulated 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 normally a constraints file to recreate the design using the available Macrocells 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 pin-out.

Obviously, if the design is too big for the chosen device the fitter 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 JED 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 (Synthesise, Fit, Timing Simulate, Program). The WebPACK ISE software will automatically perform the steps required if a particular step is selected. E.g. if the design has only just been functionally simulated and the designer then decides to do a timing simulation, WebPACK ISE will automatically Synthesise and Fit. It will then generate the timing information before it opens MXE and gives the timing simulation results.

The rest of this chapter demonstrates each step required to successfully implement the Traffic Light design in the previous chapter but now targeting a CoolRunner XPLA3 low power CPLD.

A Spartan-IIE FPGA was chosen at the start of this tutorial it must now be changed to an XPLA3 CPLD. The project can be changed at any time to any device BUT, when a device family, type, package of speed grade is changed, the design must be re-synthesised.

Double click on xc2s100e-7FT256 – XST VHDL in the Sources Window shown below in figure 6.1.1.

Figure 6.1.1   Sources in Project Window

*Change the Device Family to **Xilinx XPLA3 CPLDs***
*In the device field Select **XCR3256XL CS280***
*Click on **OK**.*

The Project, originally targeted at a Spartan-IIE FPGA is now targeting a Xilinx CoolRunner CPLD. The Green ticks in the process window have now disappeared indicating that the design must be re-synthesised and re-implemented.

6.2   Synthesis

The XST synthesis tool will only attempt to synthesise the file highlighted in the sources window. In the 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 recognises all the lower level blocks used in the top level code and synthesises them all together to create a single netlist.

In the **Sources** window ensure **top.vhd** (top_sch for schematic flows) is **highlighted**.

In the **Process** window expand the Synthesis sub-section by clicking on the **+** next to Synthesize.

You can now check your design by double clicking on **Check Syntax**. Ensure any errors in your code are corrected before you continue. If the syntax check is OK a tick will appear (as shown in figure 6.2.1). The design should be OK 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 6.2.1   Processes Window Showing Check Syntax has
                   Completed Successfully*

Right Click on **Synthesize** and select **Properties**.

A window appears allowing the user to influence the way in which the 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 the **'Add IO Buffers'** box is ticked. The IO 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.

OK the **Process Properties** window and double click on **Synthesize.**

6.3   The Constraints File

To get the ultimate performance from the device it is necessary to tell
the implementation tools what and where performance is required. The
requirement for design is particularly slow and timing constraints are
unnecessary.
Constrains can also be physical and pin locking is a physical constraint.
For this design, assume the specification for clock frequency is 100MHz
and the pin out has been pre-determined to that of a CoolRunner pre-
designed board.

In the **Process window** expand the **Design Entry Utilities** section then
expand the **User Constraints** sub section.

Figure 6.3.1   Process window showing synthesised design

Double click on **Edit Implementation Constraints File** shown above in
Figure 6.3.1.

The constraints for the design are entered in the text editor.

The **PERIOD** constraint attached to the clock informs the fitter that the
logic delay between synchronous points (flip-flops) can be a maximum of
10ns.
The LOC constraint tells the fitter which pins on the device are to be
used for a particular signal.

Figure 6.3.2   UCF File

Type in the constraints above shown in figure 6.3.2.

Save the Constraints file session. Select Reset so that the changes in
the UCF will be read.

Click on the **+** next to **Implement Design** in the **Process window**.

Figure 6.3.3   Process Window Showing Implement Design

The implementation sub-sections are now visible.

A Right Click on **Implement Design** allows the user to edit the properties for each particular step.

Figure 6.3.4   Process Properties – Implement Design

The **Help** button will explain the operation of each field.

The UCF will automatically be read by the tools. It is possible to navigate to a different UCF in the **Implementation User Conatraints File** window

Implement the design by double clicking on **Implement Design.** When there is a **green tick** next to **Implement Design** the design has completed the implementation stage. For timing report manually run the **Timing Report** section.

Note: A green tick means that the design ran through without any warnings. A yellow exclamation may mean that there is a warning in one of the reports. A common warning, that can be safely ignored in CPLD designs, is that an "fpga_don't_touch" attribute has been applied to an instance. If the design procedure outlined in this example has been followed, there should be no errors or warnings.

6.4   CPLD Reports

The are two reports available detailing the fitting results and the associated timing of the design. These are:

i. Fitter Report – An XCR3256XL has 16 function blocks of which only 3 have been used in this design. The design could be packed into a single function block but the chosen IO pins dictate which macrocells, hence which function blocks are utilised.
The first section of the report gives a summary of the total resources available in the device (256 Macrocells, 156 IO pins etc), and how much is used by the design. This information is then broken down into each individual function block.
The Partition Summary looks into each function block and shows which macrocell is used to generate the signals on the external pins.
The final section gives detailed information regarding the actual Boolean equations implemented. A '.D' indicates the logical input to a D type flip flop. The '.T' indicates a toggle flip flop provided a better implementation.

ii. Timing Report – 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, set up times and clock to out times. These values are displayed in the first section of the timing report you will have created.

The next section lists the longest set up time, cycle time (logic delay between synchronous points as constrained by the PERIOD constraint) and clock to out time.
The set up and clock to out times don't strictly effect the performance of the design. 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 this design has a minimum cycle time of 7.1ns or 140.8 MHz. This delay is created within the state machine.

The next section shows all the inputs and outputs of the design and their timing relationship with the system clock. It can be seen that the three lights will have a 4.5ns delay with respect to the clock input.
The clock to set up section details the internal nets from and to a synchronous point. The maximum delay in this section dictates the maximum system frequency.

The last section details all the path delays adding up the internal timing parameters shown at the top of the report.

A_0_, B_0_, C_0_ and D_0_ are T-Type flip-flops used to implement the state machine.

(One drawback of using GUI's to generate code is the designer has little control over the internal net names).

'inst_counter_I_count_0' through to 'inst_counter_I_count_3' are the counter T-type flip-flops.

'amber_light, red_light' and 'green_light' are the D-Type flip-flops used to register the outputs.

6.5   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** and in the **Simulation Run Time** field type **'all'.**

Click OK then double click on **Simulate Post Fit VHDL model.**

MXE opens but this time a different script file is implemented and the 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 6.5.1   Simulation Waveform

6.6   Programming

A DLC5 Parallel JTAG cable is required to configure the device from the iMPACT Programmer. Ensure the cable is plugged in to the computer and the flying leads are connected properly to the device and power supply.

Cable	Device on Board
Vcc	5v or 3.3v
GND	GND
TDI	TDI Pin
TDO	TDO Pin
TMS	TMS Pin
TCLK	TCLK Pin

With top.vhd highlighted in the sources window, double Click on **Configure Device (iMPACT)** in the Processes window.

Figure 6.6.1   iMPACT Programmer Main Window

Right click on the Xilinx XCR3256XL that appears in the iMPACT window and select **Program…**

The design will now download in to the device. Well done, you have now successfully programmed your first CoolRunner CPLD!

Summary

This chapter has taken the VHDL or Schematic design through to a working physical device. The steps discussed were:

- Synthesis and Synthesis report
- Creating User Constraints files for Timing and Pin Constraints
- The Fitting and Timing Reports
- Timing Simulation
- The XPLA programmer.

<div style="border: 1px solid black; display: inline-block; padding: 20px 30px;">

7

</div>

DESIGN REFERENCE BANK

7.1   Introduction

The 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. The applications examples have been selected from a comprehensive list of applications notes available from the Xilinx website and also extracts from the Xilinx quarterly magazine called 'Xcell' (to subscribe please visit the following web page: www.xilinx.com/xcell/xcell.htm). This section will also give you pointers on where to look for and download code and search for Intellectual Property (IP) from the Xilinx website.

7.2   Get the Most out of Microcontroller-Based Designs: Put a Xilinx CPLD Onboard

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, just to name a very few applications.

In the never-ending quest for faster, better, cheaper products, advanced designers are now pairing complex programmable logic devices (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 there are faster and more powerful microcontrollers in the field, eight-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 two and with each instruction taking up to three cycles, the actual speed of a 20 MHz microcontroller is divided by six. This works out to an operational speed of only 3.33MHz.

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 (nanoseconds), 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 perform those tasks with ultra low power consumption.

Also, 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, so designers can quickly learn how to partition their 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 article to design examples that show how you can enhance a typical microcontroller design by utilising the computational strengths of the microcontroller and the speed of a CoolRunner CPLD.

7.2.1   Conventional Stepper Motor Control

A frequent use of microcontrollers is to run stepper motors. Figure 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 5 volts to 30 volts. A high power NPN transistor drives each of the four phases. (Incidentally, MOSFETs – metal oxide semiconductor field effect transistors – can also be used to drive stepper motors).

Figure 7.2.1   Stepper Motor Controller

Each motor phase current may range from 100 mA to as much as 10 A. 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. The 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 be used to control stepper motor tasks to varying degrees of
accuracy.

The examples that follow show how either a microcontroller or a CPLD
can be used to control stepper motor tasks to varying degrees of
accuracy. We can see from Figure 2 that the design flow for both is quite
similar.

Figure 7.2.2   Design Flow Comparison

Both flows start with text entry. Assembly language targets
microcontrollers. ABEL (Advanced Boolean Expression Language)
hardware description language targets PLDs. After the text "description"
is entered, the design is either compiled (microcontroller) or synthesised
(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 (in-system programming) 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 functionality. After the board level test is completed, the PLD can then be programmed with the final code in-system via the JTAG port.

(A JTAG boundary scan – formally known as IEEE/ANSI standard 1149.1_1190 – is a set of design rules, which facilitate the testing, device programming, and debugging at the chip, board, and system levels.)

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

7.2.2   Using a Microcontroller to Control a Stepper Motor

Figure 3 shows assembly language targeting a Philips 80C552 microcontroller. The stepper motor 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 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 7.2.3   Assembly language program to rotate the stepper motor
                shaft*

7.2.3   Stepper Motor Control Using a CPLD

Figure 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 clock (CLK) input synchronises 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) initialises the circuit to ensure the correct starting sequence is provided to the outputs.

Figure 7.2.4   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 counter-clockwise. The asynchronous equations (for example, ph1.AR=!rst) initialise the circuit.

The ABEL hardware description motor control module can be embedded within a macro function and saved as a re-useable standard logic block, which can be shared by many designers within the same organisation – this is the beauty of design re-use. 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.

7.2.4   PC-Based Motor Control

Our next example (Figure 5 and 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 activated, it will override any other process or interrupt and will immediately stop the motor from rotating.

Figure 7.2.5   Design Partitioning

Figure 7.2.6   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 (Universal Asynchronous Receiver Transmitter). 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.

7.2.5 Design Partitioning

As we noted before, microcontrollers are very good at computational tasks, and CPLDs are excellent in high speed systems and have an abundance of I/Os. Figure 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 7.2.7   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 & FIFO sections of the design can be implemented in the microcontroller in the form of a costed microcontroller peripheral or may be implemented in a larger more granular programmable logic device like a field programmable gate array (FPGA) – for example, a Xilinx Spartan™ device. Using a programmable logic device in a design has the added benefit of 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 sub-routines 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 (e.g. emergency stop), in "hardware" (CPLDs) to cut down 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 (e.g., 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).

7.2.6   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 not only be quick and painless, but very cost-effective.

Extract from the Xilinx Xcell journal, Issue 39, Spring 2001.

To receive regular copies of the Xcell magazine please register at:

http://www.xilinx.com/xcell/xcell.htm

7.3   Application Notes and Example Code

The following is a list of selected application notes and example code that can be downloaded from the Xilinx website. This list is added to regularly as more applications are developed, for the latest list please visit the Xilinx website (www.xilinx.com/apps/appsweb.htm).

Title	Number	Family	Design Code
Embedded Instrumentation Using XC9500 CPLDs	XAPP076	XC9500	
Configuring Xilinx FPGAs using an XC9500 CPLD and a parallel PROM	XAPP079	XC9500	
Supply Voltage migration, 5V to 3.3V.	XAPP080	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 CPLD VHDL Introduction	XAPP105	XC9500	
Adapting ASIC Designs for Use with Spartan FPGAs	XAPP119	Spartan	

Title	Number	Family	Design Code
A quick JTAG ISP Checklist	XAPP104	XC9500	
170 MHz FIFOs Using the Virtex Block SelectRAM+ Feature	XAPP131	Virtex	
Virtex Synthesizable High Performance SDRAM Controller	XAPP134	Virtex	VHDL & Verilog
Synthesizable 143 MHz ZBT SRAM Interface	XAPP136	Virtex	VHDL & Verilog
MP3 NG: A Next generation Consumer Platform	XAPP169	Spartan II	
Virtex Synthesizable Delta-Sigma DAC	XAPP154	Virtex	
Implementing an ISDN PCMCIA Modem	XAPP170	Spartan	
Using Delay-Locked Loops in Spartan-II FPGAs	XAPP174	Spartan II	VHDL & Verilog
High Speed FIFOs In Spartan-II FPGAs	XAPP175	Spartan II	VHDL & Verilog

Title	Number	Family	Design Code
An Inverse Discrete Cosine Transform (IDCT) Implementation in Virtex Devices for MPEG Video Applications	XAPP208	Virtex	VHDL
8-Bit Microcontroller for Virtex Devices	XAPP213	Virtex & Spartan	
CoolRunner Visor™ Springboard™ LED Test	XAPP357	CoolRunner	
CoolRunner XPLA3 SMBus Controller Implementation	XAPP353	CoolRunner	VHDL & Verilog
CoolRunner CPLD 8051 Microcontroller Interface	XAPP349:	CoolRunner	VHDL & Verilog
CoolRunner XPLA3 Serial Peripheral Interface Master	XAPP348	CoolRunner	VHDL & Verilog
UARTs in Xilinx CPLDs	XAPP341	CoolRunner	VHDL & Verilog
Design of a 16b/20b Encoder/Decoder Using a CoolRunner CPLD	XAPP336	CoolRunner	VHDL & Verilog
CoolRunner XPLA3 I2C Bus Controller Implementation	XAPP333	CoolRunner	VHDL & Verilog
Manchester Encoder-Decoder for Xilinx CPLDs	XAPP339	CoolRunner	VHDL & Verilog

Title	Number	Family	Design Code
Design of a MP3 Portable Player using a CoolRunner CPLD	XAPP328	CoolRunner	VHDL & Verilog
Content Addressable Memory (CAM) in ATM Applications	XAPP202	Virtex, Virtex II	VHDL & Verilog
Virtex analogue to digital converter	XAPP155	Virtex	
Designing an Eight Channel Digital Volt Meter with the Insight Springboard Kit	XAPP146	CoolRunner	VHDL & Verilog
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	

7.4   Website Reference

The following table is a summary of useful web pages and websites that could help with your programmable logic designs.

Website/Page	Topic	Resources Available
Www.xilinx.com	General Xilinx website	Product data, investor information, application notes etc
Www.support.xilinx.com	Technical Support	Comprehensive resource for all technical support.
Www.xilinx.com/ipcenter	IP search engine	Xilinx and 3rd 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
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 frequently asked questions	
Http://toolbox.xilinx.com/ cgi-bin/forum	Forums and chat rooms	
www.model.com	Simulation	Model Technology
Www.optimagic.com/	Programmable logic jump station	

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- Applications-Specific Integrated Circuit, also called a gate array Asynchronous Logic that is not synchronised 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.

GLOSSARY OF TERMS (Continued)

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 2 or 4 look-up-tables (function generators) plus 2 or 4 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 size 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 & 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 to further reduce power consumption associated with high speed clocked in internal device networks.

GLOSSARY OF TERMS   (Continued)

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 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 up 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.

GLOSSARY OF TERMS   (Continued)

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 look-up table 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, Fast-Fourier-Transform, etc.

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.

GLOSSARY OF TERMS (Continued)

5-volt tolerant- Characteristic of the input or I/O pin of a 3.3 V device that allows this pin to be driven to 5 V 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.

Floor planning- Method of manually assigning specific parts of the design to specific chip locations. Can achieve faster compilation, better utilisation, 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.

GLOSSARY OF TERMS   (Continued)

FPGA- Field Programmable Gate Array. An integrated circuit that contains configurable (programmable) logic blocks and configurable (programmable) interconnect between these blocks.

Function Generator- Also called look-up-table (LUT), with N-inputs and one output. Can implement any logic function of its N-inputs. N is between 2 and 6, most popular are 4-input function generators.

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 customised for the individual application.

GTL- Gunning Transceiver Logic, is a high speed, low power back-plane standard.

GUI- Graphic User Interface. The 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 Windows95.

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 lay-out and routing.

Interconnect- Metal lines and programmable switches that connect signals between logic blocks and between logic blocks and the I/O.

GLOSSARY OF TERMS   (Continued)

IOB or I/O- Input/Output Block. Logic block with features specialised for interfacing with the pc-board.

ISO9000- An internationally recognised 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 the user complete a large design faster.

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 also ICs.

LogiBLOXä - 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 Parameterised 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, most popular are 4-input LUTs.

Macrocell- The logic cell in a sum-of-products CPLD or PAL/GAL.

GLOSSARY OF TERMS   (Continued)

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 XNF and EDIF.

NRE- Non-Recurring Engineering charges. Start-up cost for the creation of an ASIC, gate array, or HardWire™. Pays for lay-out, masks, and test development. FPGAs and CPLDs do not require NRE.

Optimisation- 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 characterised by short range, light loading, low cost, and high performance. 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.

GLOSSARY OF TERMS (Continued)

PCMCIA- Personal Computer Memory Card Interface Association, also: People Can't Memorise 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 OR's. This architecture is implemented in the 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 after partitioning and placement.

GLOSSARY OF TERMS (Continued)

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 textual description of a logic design.

Select-RAM- Xilinx-specific name for a small RAM (usually 16 bits), implemented in a LUT.

Simulation- Computer modelling of logic and (sometimes) timing behaviour 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 4-times more expensive than DRAM of the same capacity.

SRL16 - Shift Register LUT, an alternative mode of operation for every function generator (look up table) which 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 millimetre) 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.

GLOSSARY OF TERMS   (Continued)

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- Optimisation process of adapting a logic design to the logic resources available on the chip, like look-up-tables, Longline, dedicated carry. Synthesis precedes Mapping.

SystemI/O- technology incorporated in Virtex II FPGAs that uses the SelectI/O-Ultra™ blocks to provide the fastest and most flexible electrical interfaces available. Each user 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+. SelectI/O-Ultra technology delivers 840 Mbps LVDS performance using dedicated Double Data Rate (DDR) registers.

TBUFs- Buffers with a 3-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, (e.g. modems).

USB- Universal Serial Bus. A new, low-cost, low-speed, self-clocking bit-serial bus (1.5 MHz and 12 MHz) using 4 wires (Vcc, ground, differential data) to daisy-chain up to 128 devices.

GLOSSARY OF TERMS   (Continued)

VME- Older bus standard, popular with MC68000-based industrial computers.

XNF File- Xilinx-proprietary description format for a logic design (Alternative: EDIF).

Peter Alfke - Glossary, September 1997(Revised for this book in June 2001 and January 2002)