Beginning FPGA: Programming Metal

Your brain on hardware

Aiken Pang
Peter Membrey
Beginning FPGA: Programming Metal: Your Brain on Hardware

Aiken Pang
Chelmsford, Massachusetts
USA
DOI 10.1007/978-1-4302-6248-0

Peter Membrey
Lai Chi Kok, Kowloon, Hong Kong
China

Library of Congress Control Number: 2016962137

Copyright © 2017 by Aiken Pang and Peter Membrey

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Managing Director: Welmoed Spahr
Lead Editor: Natalie Pao
Technical Reviewer: Brendan Horan
Editorial Board: Steve Anglin, Pramila Balan, Laura Berendson, Aaron Black, Louise Corrigan, Jonathan Gennick, Robert Hutchinson, Celestin Suresh John, Nikhil Karkal, James Markham, Susan McDermott, Matthew Moodie, Natalie Pao, Gwenan Spearing
Coordinating Editor: Jessica Vakili
Copy Editor: Lori Jacobs
Compositor: SPI Global
Indexer: SPI Global
Artist: SPI Global

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a Delaware corporation.

For information on translations, please e-mail rights@apress.com, or visit www.apress.com.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–EBook Licensing web page at www.apress.com/bulk-sales.

Any source code or other supplementary materials referenced by the author in this text are available to readers at www.apress.com. For detailed information about how to locate your book’s source code, go to www.apress.com/source-code/. Readers can also access source code at SpringerLink in the Supplementary Material section for each chapter.

Printed on acid-free paper
Contents at a Glance

About the Authors........................................................................................................... xiii
About the Technical Reviewer .......................................................................................... xv

■ Part I: Getting Started with FPGA ................................................................................ 1
  ■ Chapter 1: What Is an FPGA and What Can It Do? ......................................................... 3
  ■ Chapter 2: Our Weapon of Choice .................................................................................. 13
  ■ Chapter 3: Lock and Load ............................................................................................... 23
  ■ Chapter 4: Hello World! ................................................................................................. 47

■ Part II: Time Out for Theory .......................................................................................... 105
  ■ Chapter 5: FPGA Development Timeline .................................................................... 107
  ■ Chapter 6: VHDL 101 .................................................................................................. 117
  ■ Chapter 7: Number Theory for FPGAs ........................................................................ 125
  ■ Chapter 8: Telling the Truth: Boolean Algebra and Truth Tables .............................. 137
  ■ Chapter 9: Simplifying Boolean Algebra for FPGA ...................................................... 159
  ■ Chapter 10: Sequential Logic: IF This, THEN That .................................................... 171
  ■ Chapter 11: Combinatorial Logic: Putting It All Together on the FPGA ................. 191

■ Part III: Let’s Make Something! .................................................................................... 219
  ■ Chapter 12: Light Sensors: Turning a Laser Pointer into a Hi-Tech Tripwire .......... 221
  ■ Chapter 13: Temperature Sensors: Is It Hot in Here, or Is It Just Me? ...................... 267
  ■ Chapter 14: How Fast Can You Run? Ask the Accelerometer! ................................. 313
CONTENTS AT A GLANCE

■ Part IV: Taking It Further: Talking to the Raspberry Pi and LED Displays .............................................................. 345
■ Chapter 15: Two-Way Communications with Your Raspberry Pi: SPI .... 347
■ Chapter 16: Up in Lights: How to Drive LED Segment Displays ............... 367

Index ......................................................................................................................... 381
Contents

About the Authors .................................................................................................................. xiii
About the Technical Reviewer ............................................................................................ xv

Part I: Getting Started with FPGA .................................................................................... 1

Chapter 1: What Is an FPGA and What Can It Do? ......................................................... 3
  1.1  Field-Programmable .................................................................................................... 4
    1.1.1  Configuration Technology ................................................................................ 4
  1.2  Gates = Logic .......................................................................................................... 5
    1.2.1  The Basic Gate Design Block No. 1: Logic Element ........................................ 5
    1.2.2  The Basic Gate Design Block No. 2: Configurable IO Block ......................... 8
    1.2.3  The Basic Gate Design Block No. 3: Internal RAM ........................................ 8
  1.3  Arrays Have Many Connections .............................................................................. 8
  1.4  What Can It Do? ...................................................................................................... 9
  1.5  It Can Get the Job Done Fast! .................................................................................. 10
  1.6  FPGA vs. Processor ............................................................................................... 11
  1.7  Summary ............................................................................................................... 12

Chapter 2: Our Weapon of Choice .................................................................................... 13
  2.1  What Weapons (FPGAs) Are Available .................................................................. 13
  2.2  The BeMicro Max 10: Our Weapon of Choice ....................................................... 14
    2.2.1  The Master: Altera MAX 10 FPGA ................................................................. 15
    2.2.2  The Emissaries: BeMicro MAX 10 Board Features ...................................... 17
## CONTENTS

2.3  Other Tools ............................................................................................................ 18  
2.3.1  The Place to Connect Everything: The Breadboard................................................. 19  
2.3.2  Making the Invisible Visible: The Multi-meter.............................................................. 20  
2.4  Wrap-up ................................................................................................................ 21  

### Chapter 3: Lock and Load ......................................................................................... 23  
3.1  Getting the Development Toolchain Up and Running ............................................ 23  
3.2  Downloading Altera Tools ...................................................................................... 24  
3.2.1  Altera Toolchains ........................................................................................................... 25  
3.2.2  Create an Altera Account ................................................................................................. 25  
3.2.3  Download the Altera Toolchains .................................................................................... 26  
3.3  Install Altera Quartus Prime Lite Edition .................................................................... 33  
3.4  Download BeMicro10 files and Documentation .................................................... 44  
3.5  Summary ................................................................................................................ 45  

### Chapter 4: Hello World! ....................................................................................... 47  
4.1  Launch Quartus Prime and Create a New Project ................................................. 47  
4.2  Write Code ............................................................................................................. 61  
4.3  Implement Design ................................................................................................. 69  
4.4  Simulate Design .................................................................................................... 75  
4.5  Burn It! .................................................................................................................. 89  
4.5.1  Install USB Blaster Driver ............................................................................................... 89  
4.5.2  Program Design ............................................................................................................. 92  
4.6  Recapping What We Just Completed .................................................................... 97  
4.6.1  Timing constraints ......................................................................................................... 97  
4.6.2  The Implementation ........................................................................................................ 97  
4.6.3  The Test Bench .............................................................................................................. 101  
4.7  Summary ............................................................................................................. 103  
4.7.1  But I Don’t have a Mercury Module! ........................................................................... 103
Part II: Time Out for Theory ................................................................. 105

Chapter 5: FPGA Development Timeline .............................................. 107
5.1 1847—First Theory—Boolean Logic .................................................. 107
5.2 1935—First Boolean Logic in Real World .......................................... 107
5.3 1942—First Electronic Digital Computer ............................................ 107
5.4 1960—First MOSFET ................................................................. 108
5.5 1960—First Practical Commercial Logic IC Module ......................... 109
5.6 1962—First Standard Logic ICs Family ............................................. 110
5.7 1963—First CMOS ................................................................. 110
5.8 1964—First SRAM ................................................................. 110
5.9 1965—The Well-Known Law: Moore’s Law ...................................... 110
5.10 1970—First PROM ................................................................. 111
5.11 1971—First EPROM ................................................................. 111
5.12 1975—First F-PLA ................................................................. 111
5.13 1978—First PAL ................................................................. 111
5.14 1983—First EEPROM ............................................................... 111
5.15 1983—First GAL ................................................................. 111
5.16 1983—First Programming Language/Tools for Hardware ............. 112
5.17 1985—First FPGA by Xilinx ...................................................... 112
5.18 FPGA vs. ASIC ............................................................................. 112
5.18.1 FPGA Advantages ................................................................. 112
5.18.2 FPGA Disadvantages .............................................................. 113
5.18.3 ASIC Advantages ................................................................. 113
5.18.4 ASIC Disadvantages .............................................................. 114
5.19 Other Technology .......................................................................... 114
5.19.1 CPLD ...................................................................................... 114
5.19.2 Cypress-PSoC ......................................................................... 114
5.20 Summary ....................................................................................... 115
Chapter 6: VHDL 101 ................................................................. 117
  6.1 It Is NOT Another Computer Language ........................................... 117
  6.2 VHDL File Basic Structure ................................................................. 118
    6.2.1 Entity Declaration ........................................................................... 118
    6.2.2 Architecture Body ........................................................................... 119
  6.3 Summary ............................................................................................ 123

Chapter 7: Number Theory for FPGAs ...................................................... 125
  7.1 Vocabulary in VHDL ............................................................................. 125
    7.1.1 Identifiers .......................................................................................... 125
    7.1.2 Reserved Words—Keywords ............................................................... 126
    7.1.3 Signal, Variable, and Constant ............................................................... 128
    7.1.4 Literal: Word for Word ....................................................................... 132
  7.2 Grammar in VHDL ................................................................................ 134
    7.2.1 Statements in VHDL ........................................................................... 134
    7.2.2 How to Comment ............................................................................... 134
    7.2.3 <= and := sign .................................................................................. 134
    7.2.4 Begin and End .................................................................................... 134
    7.2.5 Coding Your VHDL with Style ............................................................. 135
  7.3 Summary ............................................................................................ 135

Chapter 8: Telling the Truth: Boolean Algebra and Truth Tables................... 137
  8.1 Boolean Algebra .................................................................................. 137
    8.1.1 Simulation Steps for Boolean Algebra Example 2 ........................... 139
    8.1.2 Truth Tables ..................................................................................... 149
  8.2 Standard Logic in VHDL ................................................................. 150
    8.2.1 Standard Logic Data Types ............................................................... 151
    8.2.2 4-Bit Adder Examples with Standard Logic Types .......................... 151
  8.3 Combinational Logic Design in FPGA ................................................ 157
  8.4 Summary ............................................................................................ 158
Contents

Part III: Let’s Make Something! ................................................................. 219

Chapter 12: Light Sensors: Turning a Laser Pointer into a Hi-Tech Tripwire .... 221

12.1 Introduction ........................................................................................... 221
12.2 Photo Resistor Circuit 101 ................................................................. 222
12.3 BeMicro MAX10 LED Circuit .............................................................. 222
12.4 FPGA IP—Altera ADC IP (Hard IP and Soft IP) ............................... 224
   12.4.1 Hard IP .......................................................................................... 225
   12.4.2 Soft IP ......................................................................................... 225
   12.4.3 How to Configure Your First IP .................................................... 226
12.5 FPGA IP—Altera PLL IP ................................................................. 233
   12.5.1 Generate ADC PLL IP ................................................................. 233
   12.5.2 Generate Cascade PLL IP ............................................................. 237
12.6 Hi-Tech Tripwire Design Example .................................................... 240
   12.6.1 Light Sensor ADC Sequencer Module ........................................... 241
   12.6.2 Light Sensor Counter LED .......................................................... 244
   12.6.3 Light Sensor Top Level ................................................................. 249
   12.6.4 Add All Files to the Project and Create the Tripwire Device .......... 253
   12.6.5 Program the Tripwire Design to the FPGA .................................... 258
12.7 Summary .............................................................................................. 265

Chapter 13: Temperature Sensors: Is It Hot in Here, or Is It Just Me? .......... 267

13.1 Introduction ........................................................................................... 267
13.2 UART with Control Memory Map ...................................................... 269
   13.2.1 UART IP ........................................................................................ 270
   13.2.2 UART PC Software ................................................................. 272
13.3 I2C Communication ............................................................................. 274
   13.3.1 Basic I2C ...................................................................................... 274
   13.3.2 I2C Master .................................................................................. 274
   13.3.3 Temperature Sensor—Analog Device ADT7420 ......................... 276
   13.3.4 I2C Slave .................................................................................... 278
13.4 FPGA IP—Altera PLL IP ..................................................................................... 280
13.5 PC Control Temperature Sensor Design Example .............................................. 282
  13.5.1 Define What Needs to Be Done—Command and Status Registers .......... 284
  13.5.2 Example Design Codes .................................................................................. 285
  13.5.3 Example Simulation Codes ........................................................................... 292
  13.5.4 Create Temperature Sensor Project Design and Program It ..................... 297
  13.5.5 Hardware Setup for the Temperature Sensor Project Design .................. 305
  13.5.6 UART Software Setup—RealTerm ................................................................. 305
  13.5.7 Command Your FPGA to Read the Temperature ....................................... 307
13.6 Summary ........................................................................................................... 311

Chapter 14: How Fast Can You Run? Ask the Accelerometer! ............................ 313
14.1 Introduction ....................................................................................................... 313
14.2 Steps to Build Your First Interface Module ..................................................... 314
  14.2.1 Understanding the SPI .................................................................................. 314
  14.2.2 What Do You Need for an SPI Master Module? .......................................... 314
  14.2.3 Create the SPI Master Module Entity Port List ........................................... 316
  14.2.4 Create Processes in VHDL for the Requirements ....................................... 318
14.3 PC Control Accelerometer Sensor Design Example ........................................ 322
  14.3.1 Add New Command and Status Registers ................................................ 324
  14.3.2 Create the Temperature Sensor Project Design and Program It .............. 326
  14.3.3 Example Design Codes ................................................................................ 326
  14.3.4 Hardware Setup for the Accelerator Sensor Project Design ..................... 338
  14.3.5 Initialize the Accelerometer—ADXL362 ..................................................... 338
14.4 Summary ........................................................................................................... 343

Part IV: Taking It Further: Talking to the Raspberry Pi and LED Displays ............ 345

Chapter 15: Two-Way Communications with Your Raspberry Pi: SPI .................... 347
15.1 Introduction ....................................................................................................... 347
15.2 Define Our SPI Slave Interface for the Raspberry Pi ...................................... 347
15.3 Design SPI Slave in FPGA ................................................................. 349
  15.3.1 New SPI Slave Module Port List ..................................................... 350
  15.3.2 Raspberry Pi SPI Master 0 Default Setting and Data Format ............... 351
  15.3.3 Writing VHDL for the SPI Slave ..................................................... 352
15.4 Create the FPGA Top-Level Design .................................................. 360
  15.4.1 Top-Level Design VHDL Code ....................................................... 361
  15.4.2 Generate and Program the FPGA .................................................. 363
15.5 How to Use Raspberry Pi SPI Master Interface .................................... 365
  15.5.1 Python Code to Read and Write SPI Master .................................... 365
15.6 Summary ............................................................................................ 366

Chapter 16: Up in Lights: How to Drive LED Segment Displays ............... 367
16.1 Introduction ....................................................................................... 367
16.2 How to drive a 7 segment display ..................................................... 367
  16.2.1 Connecting 7 segment display to FPGA ....................................... 368
16.3 Designing the 7 segment display counter ......................................... 371
  16.3.1 Simple counter design section ...................................................... 372
  16.3.2 7 segment decoder section ............................................................ 374
  16.3.3 End of the counter design ............................................................. 374
16.4 7 Segment display example design ................................................... 375
  16.4.1 Code for the top level design ....................................................... 375
  16.4.2 Generate and program the FPGA ................................................ 378
16.5 Control the 7 segment counter from Raspberry Pi .............................. 378
16.6 Summary ............................................................................................ 379

Index ......................................................................................................... 381
About the Authors

Aiken Pang was born in Hong Kong in the early 1980s just in time for the home micro-computer revolution. He developed an interest in computers after finding an Apple IIe at his uncle’s home; he installed a TV card and a floppy drive and loaded his first game (Pacman) by himself at the age of 10. Since then he has self-taught himself a number of programming languages and built his own computers from scratch.

Aiken followed his passion for computing into more formal studies and holds a BEng in Electronics Engineering and an MSc in Computer Engineering from Hong Kong Polytechnic University and the University of Massachusetts at Lowell, respectively. He has been using VHDL (VHSIC (very high speed integrated circuit) Hardware Description Language) to design hardware in FPGAs (field-programmable gate arrays) for more than ten years.

He lives in the United States with his wife France and their daughter Mayah.

Hailing from the UK, Peter Membrey has worked for Red Hat, holds a RHCE certification, and has worked and taught at a number of educational institutions since the beginning of his career. He knows what Linux users like and need and hopes that CentOS will get the kudos it deserves. He lives in Hong Kong and is teaching and consulting on all matters having to do with Linux Enterprise networking, while studying for his master’s degree.
About the Technical Reviewer

Brendan Horan is a hardware fanatic, with a full high rack of all types of machine architectures in his home. He has more than 10 years of experience working with large UNIX systems and tuning the underlying hardware for optimal performance and stability. Brendan’s love for all forms of hardware has helped him throughout his IT career, from fixing laptops to tuning servers and their hardware in order to suit the needs of high-availability designs and ultra low-latency applications. Brendan takes pride in the Open Source Movement and is happy to say that every computer in his house is powered by open source technology. He resides in Hong Kong with his wife, Vikki, who continues daily to teach him more Cantonese.
Getting Started with FPGA
CHAPTER 1

What Is an FPGA and What Can It Do?

Field-programmable gate arrays (FPGA) are a special type of integrated circuits (ICs) or chip that can be programmed in the field after manufacture and has three basic building blocks: logic gates, flip-flops + memories, and wires. In this chapter we will quickly review what FPGA is and some of the things it can do.

An IC has input and/or output pins (the gray color in Figure 1-1). The black box is the “brain” of the IC and most ICs have white or gray markings on top of the black box. Most of the ICs mention their specific functions in their datasheet (e.g., is it an amplifier, a processor, a counter, an Ethernet MAC, or a combination of the lot). If you read the FPGA’s datasheets and are looking for a specific function, you will probably get very frustrated. That’s because they don’t mention the purpose or any of the FPGA’s feature sets. All you will be able to find is how many logic gates, how much memory, and how to program the FPGA, but you won’t find functions or features. You just cannot figure them out from the datasheet.

Figure 1-1. Integrated circuits, or chip, look like this
FPGAs allow designers to modify their designs very late in the design cycle—even after the end product has been manufactured and deployed in the field. Sound familiar? It should sound a lot like Windows updates or Android/Apple phone software updates. That’s one of the most powerful and compelling features of an FPGA but please don’t treat FPGA design as a software programming exercise for a microcontroller or processor. An FPGA is not a processor with software. It is an amazing device that allows the average person to create his or her very own digital circuit. You are designing a hardware digital circuit when you are creating an FPGA design. You are going to use a hardware description language (which, incidentally, is used by Intel CPU chip designers too) to design your FPGA. This is a very important concept. We will provide more details when we are putting together some example designs in the later chapters. In the following sections we’ll provide a little bit more detail about field-programmable, gates, and arrays and what they can do.

1.1 Field-Programmable

The most valuable FPGA feature is that the end user can program or configure it within seconds. This means that the end user can change the hardware design in the FPGA chip quickly and at will. For example, the FPGA can change from temperature sensor to LED (light-emitting diode) driver within a few seconds. This means that FPGAs are useful for rapid product development and prototyping. This field-programmable magic is done by a configuration file, often called a bit file which is created by a designer (you). Once loaded, the FPGA will behave like the digital circuit you designed!

1.1.1 Configuration Technology

Table 1-1 lists the three types of configuration technologies: static random access memory (SRAM), flash memory, and antifuse.

<table>
<thead>
<tr>
<th></th>
<th>SRAM</th>
<th>Flash</th>
<th>Antifuse</th>
</tr>
</thead>
<tbody>
<tr>
<td>Achronix</td>
<td>YES</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Altera</td>
<td>YES</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Lattice</td>
<td>YES</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Microsemi</td>
<td>--</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>Xilinx</td>
<td>YES</td>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>

Most of the FPGA vendors are using SRAM technology. It is fast and small, and it offers unlimited reprogrammability. One of the drawbacks though is that the FPGA needs time to reload the entire design into SRAM every time you power up the FPGA. This approach also takes more power.

Flash and antifuse technologies are non-volatile (meaning that they can retain data even if the power is turned off) so that they provide the benefit of “instant on” without needing to reload the FPGA bit file every time we power up the FPGA or the system. They also draw less power than the SRAM approach.

Antifuse technology can only be programmed once and can’t match the performance of SRAM technology. The only reasons to use an antifuse technology FPGA today is due to its super high reliability and security.
The world's largest and most powerful particle accelerator—Large Hadron Collider (LHC)—uses antifuse FPGAs to implement radiation protection digital circuits.

This book focuses on SRAM technology because it is the most common technology and is the easiest to program. Most of the SRAM-based FPGAs have an external EEPROM (electrically erasable programmable read-only memory) for storing the bit file, similar to how a computer stores programs on disk. All you need to do is "burn" your bit file into the EEPROM and the FPGA will autoload the bit file from the EEPROM when it powers up.

1.2 Gates = Logic

The gate is the most basic element in digital logic and is more formally known as a logic gate. All modern digital designs are based on CMOS (complementary metal oxide semiconductor) logic gates. To support complex digital designs, FPGAs contain tens of thousands, hundreds of thousands, or even more individual logic elements which are built by logic gates.

1.2.1 The Basic Gate Design Block No. 1: Logic Element

The logic element (LE) is one of the smallest elements in FPGA design. It basically consists of a look-up table (LUT), flip-flop, and multiplexer (which we will cover shortly). FPGAs are used extensively for compute problems that can benefit from parallel computing architectures—for example, cleaning up images being received from an image sensor, local processing on image pixels, and computing difference vectors in H.264 compression. LEs can form any complex or even simple digital function inside the FPGA. Figure 1-2 shows the basic configurable logic elements. Although different FPGA vendors have their own LE designs, they are very similar to the one shown in Figure 1-2. Most of the inputs to a LE are connected to the LUT and followed with a flip-flop (register). The output of the LE is selected by a multiplexer (MUX). The LUT and MUX are the major configurable blocks in the LE. Don’t worry, this will all make sense after you read the next section.

![Figure 1-2. Basic configurable LEs](image-url)
1.2.1.1 The Magic Block: LUT

The LUT is basically just a small amount of read-only memory. A four-input, one output LUT, can generate any four-input Boolean function (AND/OR/XOR/NOT) (Figure 1-2). We will provide more details on Boolean function in Chapters 8 and 9. When you configure the FPGA, the contents of the LUT will be configured accordingly. Table 1-2 shows an OR gate with all four possible inputs the LUT supports. The four inputs act as an address bus, addressing one of 16 \(2^4\) stored bits; therefore it can emulate any Boolean function that can be handled in 4 bits (Figure 1-3). In some FPGA LE designs, LUTs can be combined to form a 16-bit shift register or memory block. There is another similar block, the MUX, in the basic LE. It is used to select the output source from a register or directly from LUT. In reality, the LEs are a bit more complex in that they may very well have more than one LUT and MUX.

Table 1-2. Four-Input, One-Output LUT

<table>
<thead>
<tr>
<th>Address</th>
<th>I3</th>
<th>I2</th>
<th>I1</th>
<th>I0</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

If you are serious about FPGA digital design, then you may want to read the LUT datasheet from the FPGA vendor. You can do more with the same FPGA, if you know how to fully use the LUT blocks.
1.2.1.2 The 1-bit storage block: Flip-flop

The flip-flop (or register) is the smallest storage unit in an FPGA. Each flip-flop can store 1 bit of information at a time. Like a light switch, it can either be on (a value of 1) or off (a value of 0). The basic function of a flip-flop is to hold information and make it available to the logic elements for the computing process. This flip-flop is one of the most important blocks in a computer. Figure 1-4 shows the digital design of a D-type flip-flop, which is a basic flip-flop used in FPGA.

![Flip-flop logic gate design (D-type flip-flop)](image)

Figure 1-4. Flip-flop logic gate design (D-type flip-flop)

There are two inputs (D and Clock) and two outputs (Q and ~Q) in a flip-flop. The D flip-flop captures the value of the D input at a defined stage of the clock cycle (such as the rising edge of the clock input). That captured value becomes the Q output. At other times, the output Q does not change. The D flip-flop can be viewed as a memory cell, a zero-order hold, or a delay line. Multiple flip-flops connected together can form a shift register.

Tip You can go to the following web site for more details about D-type flip-flop: [http://electronics-course.com/d-flip-flop](http://electronics-course.com/d-flip-flop).

Most of the FPGA digital designs are synchronous design. It means that all the inputs and outputs are synchronous to one clock or more. All the examples in this book are synchronous design with the rising edge of the clock because most of the FPGAs are built for implementing synchronous design.

Note A synchronous circuit is a digital circuit in which the changes in the state of memory elements are synchronized by a clock signal. In a sequential digital logic circuit, data is stored in memory devices called flip-flops or latches. Source: [www.wikipedia.org](http://www.wikipedia.org). It is like shooting pictures. Each picture stores the “data” when you hit the shutter button. The picture is the storage elements and the button is the clock signal.
1.2.2 The Basic Gate Design Block No. 2: Configurable IO Block

Another basic logic design element is configurable input/output block (IOB). It is used to connect the LE to the outside world. It can be configured as an input or output, as bi-directional, or not connected. The IOB can support different types of electrical input/output (I/O) specifications (e.g., TTL logic, 3.3V CMOS, and PCIe) and add internal pull-up or pull-down resistors. Remember to select the correct I/O specification. It is because different specifications have different driving strength (voltage and current) and most of the time they are not compatible. The worst is overload the FPGA IO and break the FPGA.

The latest FPGA IOB can support bandwidth higher than 10 Gbps. Some FPGAs even have direct optical input and output physical connections (Source: www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/wp/wp-01161-optical-fpga.pdf). Chapter 12 provides a detailed discussion on how to use IOBs.

1.2.3 The Basic Gate Design Block No. 3: Internal RAM

The internal RAM (random access memory) block is another configurable unit in the FPGA. The main configuration parameter is number of read/write ports. You can read and write the same memory with different locations at the same time. It is like one instruction to read and write on the same physical memory with different address. Can you think of a use case of this type of configuration (read and write at the same time/clock cycle)? One possibility is FIFO (first in, first out memory) which is used to pass data from module A to module B. There is an FPGA vendor that puts DRAM (dynamic random access memory) (which is the one you’ll find in your computer because of its density—it can pack more bits into the same physical size) inside the FPGA and allows the rest of the logic to access it directly. Some FPGA vendors put flash memory (which is a non-volatile memory such as you’d find in a memory card or USB stick) into their FPGA. If your application needs non-volatile memory, then an FPGA with built-in flash memory would be a good fit for you. In Parts 3 and 4, we will show how to configure internal RAM.

1.3 Arrays Have Many Connections

An array is a large group of things put together with a particular order. At this moment, we know that FPGA has a lot of LE, IOB, and internal memory. All of them can be configured to do whatever you would like them to do. There is a last piece of kit inside the FPGA that can be configured; the connections between the LEs and IOBs. To make the connection efficient, FPGA vendors put the LEs and IOBs into a two-dimensional array (some newer FPGA’s have three-dimensional arrays instead). Figure 1-5 shows an FPGA array example. All of the IOBs are close to the IC edge such that it has the shortest distance to the outside world. The wires between each LE and IOB are the configurable connection (wires). These are extremely flexible and easy to use. Most of the time, the FPGA vendor tools take care of all the connections for you.
Chapter 1

What Is an FPGA and What Can It Do?

Digital logic theory: If you have enough NAND gates then you can build anything that you want. In an ideal world we don’t need to have complicated LE blocks. All we need to have are NAND gates and connections in between.

In the real world: The routing between NAND gates will very quickly run out of room to finish even a simple job. Traditional FPGAs are optimized for more complex configurable logic element blocks with a relatively limited number of interconnections between them. Today, FPGAs have added more interconnections by using 3D arrays. These 3D arrays are stacked like multiple chips with connections between chips.

Keep in mind: It is possible to run out of connections between LEs and IOB and thus fail to generate a viable FPGA design.

1.4 What Can It Do?

Generally speaking, all the FPGA does is generate ones (3.3 V) and zeroes (0 V) which means it can do everything or nothing. That doesn’t sound terribly impressive does it? However, you need to ask the right question in order to get the right answer. The right question is, “What you want it to do?” The more you do, the more it can do for you! Think of the FPGA as a piece of clay. You can mold that clay into any number of shapes, make a plate, make a statue, or make a tiny house. Its potential is limited only by your imagination, and that’s also true of an FPGA—it has the capability to take on practically any function you can imagine!
Chapter 1  What Is an FPGA and What Can It Do?

It is very difficult to design your own chip completely from scratch. The big difference between an FPGA and every other chip you can buy on the market is that an FPGA doesn’t actually do anything. It has no intended function when you buy it. It is not like a microcontroller that has a defined function or a generic process ready to run software. An FPGA gives you the ultimate in flexibility, allowing you to design anything you can imagine in the digital domain. If you want to turn your FPGA into an 8051 microcontroller, you can do it. You can configure your FPGA to be a custom LED driver for 1,000 LEDs, a 100 PWM (Pulse Width Modulator), or a universal asynchronous receiver/transmitter (UART or COM port) device, which is a common computer interface. However, just because it can be done with an FPGA does not mean that it will necessarily be easy to do so.

When you play a join-the-dots puzzle, you need to follow the numbers (features) to draw lines (design). It is like you get a processor; you have a fixed amount of counters, timers, and interface types (features); and you write your code (design) to follow the features. FPGAs are like join-the-dots puzzles but without any numbers at all (Figure 1-6). You need to design all of the features and rules in the FPGA to make it “work.” Another way to look at it is rather than you writing software for a predetermined feature set, you have the ability with an FPGA to actually define the feature set yourself—you don’t have to follow the numbers because with an FPGA, there simply aren’t any numbers to follow!

![Figure 1-6. Connect-the-dots puzzles without number](image)

Some fun: Try to join all 25 dots in Figure 1-6 by drawing no more than eight straight lines. The straight lines must be continuous.

**Tips**  Think out of the box. This tip is valid for designing FPGA too!

### 1.5  It Can Get the Job Done Fast!

FPGAs can get the job done faster than a computer, if you can separate your job into discrete pieces. This is due to order reduction in FPGA tools and parallelism operation in FPGA. The FPGA tool (mention in Chapter 3) is smart enough to reduce complicated operations to simpler ones. This operation is order reduction! The tools can reduce complex operation like multiplying by 2 into an addition which runs faster and uses less energy. An FPGA can parallelize a task that was already much slower to run as software on a CPU (central processing unit). Once the designer starts to realize the following, “I can perform in 25 FPGA clock cycles a task which takes my CPU 200,000 clock cycles, and I can do this task in parallel 5 items at a time,” you can easily see why an FPGA could be a heck of a lot faster than a CPU! Reversing the order of bits inside a 32-bit integer is a very good example for order reduction. Computers need to use a FOR loop to do it which will take a number of clock cycles to complete. You can accomplish the same thing in a single cycle with an FPGA.
Data flow into and out of the FPGA is expandable too! So you can send a lot of data to an FPGA to get more jobs done at the same time. Image, video, and Ethernet packet processing all need this kind of high bandwidth data flow.

For example, suppose you want to do some processing work on an image, and let’s say you want to rotate it. Let’s also say that it takes one second to do this, and you have 10,000 images to do. A single processor would take nearly three hours to process this workload. Of course, modern CPUs are multicore, so let’s be generous and assume we have eight cores at our disposal. Now we can do the same work in just over 20 minutes. With the FPGA, though, we might have our “image processing” unit, and maybe 500 of those fit on a single FPGA. If we could keep those units full of data, we’d complete our task in only 20 seconds. If we also consider that it is now being processed in hardware rather than software, we could say that it only takes half a second to process an image, dropping our total time down to ten seconds. Okay, this example is a bit contrived and there are plenty of ifs, buts, and maybes, but it gives you an idea of the power of an FPGA.

### 1.6 FPGA vs. Processor

The major difference between a processor and an FPGA is that an FPGA doesn’t have a permanent hardware configuration; on the contrary, it is configurable according to the end-user needs. However, processors have a permanent hardware configuration which means that all the transistors, registers, interface structures, and all of the connections are permanent. A processor can only do predefined tasks (accumulation, multiplication, I/O switch, etc.). Designers make the processor do these tasks “in a consecutive manner” by using software, in accordance with their own functions.

Hardware configuration in the FPGA is not fixed so it is defined by the end user. Although logic elements are fixed in FPGA, functions they achieve and the interconnections between them are controlled by the user. So tasks that FPGAs can do are not predefined. You can have the task done according to the written hardware description language (HDL) code “in parallel,” which means concurrently. The capability of parallel processing is one of the most important features that separates FPGAs from processors and makes them superior in many areas.

Processors are generally more useful for repetitive control of specific circuits. For example, using an FPGA for simple functions such as turning on or off a device from a computer may be overkill. This process can easily done with many conventional microcontrollers. However, FPGA solutions are more reasonable, if you want to process 4K video data on the computer. Table 1-3 compares FPGAs and processors in a few more ways.

| **Table 1-3. FPGA vs. Computer** |
|-----------------------------|-----------------------------|
| FPGA | Processor |
| **Cost** | High | Low |
| **Hardware structure** | Flexible | Fix |
| **Execution** | Concurrent | Sequential |
| **Programming** | HDL | Assembly language |
| **Development time** | Long | Short |
| **Power** | Efficient | Not efficient |

There is something that the both have in common and we will find out more about that together in Chapter 2.
1.7 Summary

Field-programmable gate arrays have three basic building blocks.

1. Logic gates
2. Flip-flops + memories
3. Wires

FPGAs can do many things, or they can do nothing, with these three building blocks. It all depends on you. You need to design the hardware structure to handle the task you want. FPGAs can concurrently process your tasks and this is the main difference between FPGA and a processor. Chapters 2, 3, and 4 will get you ready to design real digital logic.

In Chapter 2, you will get hands on with the hardware platform we will be using.
In Chapter 3, you will install FPGA digital design environment software.
In Chapter 4, you will create your first digital design in the FPGA world!

As you work your way through this book (Part 2 and Part 3), you will create a number of design blocks; each block can be reused. Each block is like LEGO, you can reuse all the blocks you create in other designs.

“Nothing will work unless you do.”

—Maya Angelou

Electronic supplementary material The online version of this chapter (doi:10.1007/978-1-4302-6248-0_1) contains supplementary material, which is available to authorized users.
CHAPTER 2

Our Weapon of Choice

This chapter will briefly review the various FPGAs (field-programmable gate arrays) available and then examine the features of BeMicro MAX 10 board, which is what we are using throughout the book.

Finally, we introduce some tools for simple hardware projects. You will find that they are very useful for this book and for your hardware projects.

2.1 What Weapons (FPGAs) Are Available

You may think FPGAs are expensive things. This was true ten years ago, but right now, you can always find a FPGA fit for your budget and application. A quick browse on one FPGA manufacturer’s web sites turns up more than five differently named FPGAs. This is similar to browsing Intel’s web site where you’ll find a wide range of processors available: Celeron, Pentium, Core, Atom, Xeon, and Itanium. All this choice is great, but what does it really mean for you? Well, just like you need the right processor for a particular job, you’ll also need the right FPGA. Don’t worry, though, our job is to get you up to speed so that you’ll be able to navigate through all the choices with ease.

Most of the FPGA companies don’t have that much support for people who are just getting started, so let’s begin with a quick look at the different types of FPGAs you can get your hands on:

In general, FPGA manufacturers provide three types of FPGA:

1. The Aircraft Carrier—System on Chip (SoC): This one comes with pretty much everything onboard (FPGA chip). SoC means that the FPGA has a central processor unit (CPU) and many standard interfaces built-in. An ARM core is one the most common processor designs that you’ll find in SoC FPGAs. Because it comes with all these features, a SoC can be used to quickly develop prototypes and they tend to have a lot of custom interfaces wrapped around a CPU for which existing software is already written. This type of FPGA is best suited for your need for software flexibility and hardware speed. One of the hottest examples is Advanced Driver Assistance Systems (ADAS). It is a system to help the car driver in the driving process. The system should improve car and road safety. All of the autopilot cars (Google self-driving car, Telsa Autopilot system, and Volvo’s IntelliSafe Autopilot) have ADAS.

2. The Laser Cannon—High End: It provides high-density computational power and super high-speed digital interfaces in one go. The computational power comes from a specially designed logic element(LE) which is digital signal processing(DSP) blocks that we discussed in the first chapter. One of the super high-speed digital interfaces available is PCIe Gen 3 (close to 8GBs per second. It can transfer one and a half DVDs within seconds). This type of FPGA is designed for consuming a lot of digital data. For example, High-Performance Computing (HPC), Optical Transport Networks, and Software Define Radio are using this type of FPGA.
3. The Pocket Knife—Low End: It provides the most cost-effective solution for a general digital interface. This type of FPGA is like a sea of gates which has only basic elements (logic, memory, and clocks). Anything you can think of using logic gate to design, you can finish with this type of FPGA. This type of FPGA is like an 8051 MCU, which has all you need to do basic things. This type of FPGA is best suited for consumer products, low-cost prototyping, and education.

Table 2-1 shows the FPGA names from different FPGA manufacturers. You can use this table to narrow down what FPGA may suit your next project FPGA need.

<table>
<thead>
<tr>
<th>FPGA</th>
<th>SoC</th>
<th>High End</th>
<th>Low End</th>
</tr>
</thead>
<tbody>
<tr>
<td>Achronix</td>
<td>N/A</td>
<td>Speedster 22i</td>
<td>N/A</td>
</tr>
<tr>
<td>Altera</td>
<td>Stratix SoC</td>
<td>Stratix/ Arria</td>
<td>Max 10 (used in BeMicro Max 10)</td>
</tr>
<tr>
<td>Micorsemi</td>
<td>Smart Fusion 2</td>
<td>N/A</td>
<td>IGLOO 2</td>
</tr>
<tr>
<td>Lattice</td>
<td>N/A</td>
<td>ECP/Lattice</td>
<td>iCE40 / MachXO</td>
</tr>
<tr>
<td>Xilinx</td>
<td>Zynq</td>
<td>Virtex</td>
<td>Spartan-6</td>
</tr>
</tbody>
</table>

If you’re hoping you can simply search for the name of the FPGA and find just the development kit you’re looking for, you’re probably going to get lost very quickly. The problem is that there are literally thousands of different combinations out there ranging in cost from hundreds to tens of thousands of dollars. Often just looking at the description between two parts, it isn’t even obvious why one is five times the price of the other. Once you get a few projects under your belt, you will welcome this bounty of choice—but for now, how do you even know where to start?

That’s where the BeMicro Max10 development board comes in: it is small, practical, and reasonably priced. It’s powerful enough for you to do a lot of different things and it includes enough hardware built in to start building projects with practical real-world applications straight away. Every component is representative of what you’ll find in industry. This board puts you on equal footing with the big boys, but without the big cost.

In subsequent sections we’ll look at the BeMicro Max 10 and the other tools you’ll need in this book.

Tips Please go to the following web site to order your BeMicro Max10 development board: http://www.arrow.com/bemicromax10.

2.2 The BeMicro Max 10: Our Weapon of Choice

The BeMicro MAX 10 development board is developed by Arrow. The purpose of this board is to demonstrate any many of the MAX 10 features as possible. Arrow’s web site says it best: “A compact and low-cost hardware evaluation platform for a broad range of embedded applications.” This makes it a great fit for our purposes. Figure 2-1 shows the BeMicro Max 10.
The most attractive feature for us is the price. For only US$30 at the time of this writing, it should fit anyone’s budget. In terms of price, it is also very competitive with similar boards on the market.

2.2.1 The Master: Altera MAX 10 FPGA

The BeMicro MAX 10 development board is based on a single Altera MAX® 10 FPGA. If you’re wondering whether or not Altera offers the MAX 9, 8, 7, then the answer is no. For some reason Altera only has MAX® 10, MAX V (5), and MAX II (2). Strangely though, MAX® V and MAX® II are not even FPGA products, just in case things weren’t confusing enough. They are Altera’s CPLD (Complex Programmable Logic Device). In the next chapter we will describe what CPLDs are and what the big deal is of changing from a CPLD to FPGA. There are a lot of impressive features offered in the MAX® 10 FPGA, such as an analog-to-digital converter (ADC) block, a temperature-sensing diode, and flash memory. In case you wanted to find out more about the FPGA specifically, the part number of the FPGA on our BeMicro MAX 10 is 10M08DAF484C8G.

Here is a brief summary of the features in this FPGA:

- 8,000 logic elements (LEs)—Look-up table (LUT)/logic array block
- 378 Kbit embedded SRAM (static random access memory). Altera name: M9k
- 256 Kbit user flash memory (UFM)—You can use it to store anything you like
- Non-volatile self-configuration flash memory (CFM)—This is used to store the FPGA configuration file
- Two phase locked loops (PLLs)—Very useful for creating clocks inside the FPGA
- 24 18x18-bit multipliers—DSP block: These are hard (i.e., burned into silicon rather than made up of LEs from the FPGA itself) which means it runs really fast!
- 1 ADC block with 18 channels—17 of them are external inputs and 1 is the FPGA internal temperature sensor
- 250 general-purpose input/outputs (I/O)—You can use them to do a lot of different input and output at the same time
This version of the MAX 10 FPGA brings logic element densities up to 8K logic elements. It looks small compared with all other big FPGAs, but it can provide us with enough logic gates for our needs. Each logic element is equal to around a 25 logic gate design plus one flip-flop. 8k logic elements then are roughly equivalent to 400,000 transistors. Consider for a moment that in the early to mid-1980s a high-end personal computer’s CPU had well under 100,000 transistors in it. Can you even begin to imagine what’s possible with 400,000 transistor logic gates?

The MAX 10 includes analog block (ADCs and temperature-sensing diodes), which enables the device to be used in system monitoring applications. Figure 2-2 shows the basic blocks location on the chip. It may be easy for you to figure out what the FPGA can do from the figure. The LUT, M9K block ram, and DSP block are very interesting and important basic blocks in Altera FPGAs. In Part II of this book, we will describe these basic blocks in more detail with design examples.

With this kind of density, the MAX 10 can easily integrate an Altera Nios II soft-core embedded processor, turning it into a non-volatile almost instant-on SoC. Today, it can actually integrate “complete system-level designs,” at least for a sufficiently modest definition of “system.” (system definition: computer with inputs and outputs).