Chapter 1
Preliminaries

1.1 About This Book

This book covers a lot of ground that is constantly shifting. Hardware is constantly changing, software is always being updated, toolchains are always being improved, and development methodologies change over time. So yes, this book is likely to be out of date as soon as it is "printed." So why bother with this book? There are three primary reasons:

Reason 1: Students gain a ***system level*** view of computers where they see how hardware and software interact.
Reason 2: System-on-Chip (SoC) Field Programmable Gate Arrays (FPGAs) are an ideal platform for teaching hardware and software interactions and low cost SoC FPGAs are affordable for students.
Reason 3: Students get jobs. Students have reported great feedback in their job interviews. They report that being able to explain how they created their own custom hardware in the FPGA fabric and being able to explain how they wrote their own device drivers in Linux have impressed interviewers from some very large companies.

This book takes the ***you do not understand it until you build it*** approach to student learning. This means that the labs in the book are the central focus of the book. It is the process of building a complete computer system that cements the various elements together. The chapters exist to support the labs and provide background information that is needed for completion of the labs.

The choices of hardware, software, and methods are all the fault of the author. In defense of these choices, the author will argue that in computer science, if there are choices to be made, they will all be made. This means that one needs to be familiar with the various approaches that exist and pragmatic when it comes to a particular choice. Ultimately, a choice has to be made and this book reflects the biases of the author. Are there better ways? Yes, and this will always be the case in a field that is moving quickly and constantly changing. This book reflects a current

R. K. Snider, *Advanced Digital System Design using SoC FPGAs*,
https://doi.org/10.1007/978-3-031-15416-4_1

snapshot in time with no claim that it is the optimal way of doing things. However, it does accomplish the goal of creating a system-level understanding of computers for students in spite of the fact that one can quibble about the particular choice of hardware, software, or target application.

1.1.1 GitHub Repository

The GitHub repositories associated with the book are listed in Table 1.1.

Table 1.1: GitHub book repositories

Book repository link	Description
ADSD-SoC-FPGA	Primary GitHub site for the book
Code	Code repository
Updates	Update repository. It is anticipated that material in the book will constantly be changing due to advances in hardware, software, toolchains, and methodologies, so check here for updates. **Note:** If something does not work, make sure that you are using the same version of software that the book used. There are no guarantees that newer versions of software or toolchains will work the same way. In fact, using the latest versions of software or toolchains is a good way to break things, so you are on your own if you choose to use different versions than what the book uses. Always expect to have toolchain fights when you upgrade to a new version

1.2 Why Learn About SoC FPGAs?

Why do we learn about *System-on-Chip* (**SoC**) *Field Programmable Gate Arrays* (**FPGAs**)? In short, SoC FPGAs are extremely flexible digital devices that allow you to create custom hardware for embedded computing systems with high performance, high bandwidth, and low deterministic latency.

Computer *engineering* is a discipline about creating systems using computers for a particular purpose. This includes creating both custom hardware and software. Creating custom hardware is what distinguishes computer *engineering* from computer

science. A rough dividing line between computer engineering and computer science, painted with a broad stroke, is the operating system on a computer. Computer engineering is concerned primarily with everything below the operating system, down to the hardware circuits, and how the computer interfaces to the physical world and other systems. Computer science is concerned about what is theoretically possible, about abstracting computers to make them easier to use, and creating languages with the right amount of useful abstractions for a particular purpose. And while we are painting in broad strokes, *engineers are the people who make science useful*.

In a fantasy world where cost is of no concern, one would create a custom chip, known as an *application-specific integrated circuit* (**ASIC**), for each product developed. However, creating a custom SoC chip that is ideal for a single purpose and that is fabricated using a 7 nm process will cost you hundreds of millions of dollars [1, 2]. This means that creating ASICs that use a leading edge fab process is only economical if you have a large market that supports it. This is because you can spread the *non-reoccurring engineering* (**NRE**) development costs over millions of units.

If your target market does not justify rolling an ASIC and you have multiple customers needing the same functionality, you then create what is known as an *application-specific standard product* (**ASSP**). An example of an ASSP is the AD1939 audio codec[1] from Analog Devices [3] that we use to acquire and play audio signals as explained in Chap. 5. Analog Devices markets this chip to customers who need to convert audio signals from analog to digital and then back to analog. This works well for customers like us who need an audio codec but do not have the deep pockets to fund the infrastructure and expertise to create a mixed-signal audio design. It is challenging to put both digital and analog systems on the same chip since one has to keep the noisy digital system from injecting noise into the analog system.

However, what if you want to develop a *custom* computer hardware system but do not want the cost and development effort associated with developing ASICs or ASSPs? A typical choice is to use the familiar CPU to create your system. CPUs range from cheap microcontrollers where 28.1 billion of them were shipped in 2018 with an average selling price of $0.63 [4] to high-end CPUs such as the Intel Xeon 8180M that cost $13K at introduction [5].

[1] Codec stands for coder-decoder, where the coder is an analog-to-digital converter (ADC) and the decoder is a digital-to-analog converter (DAC).

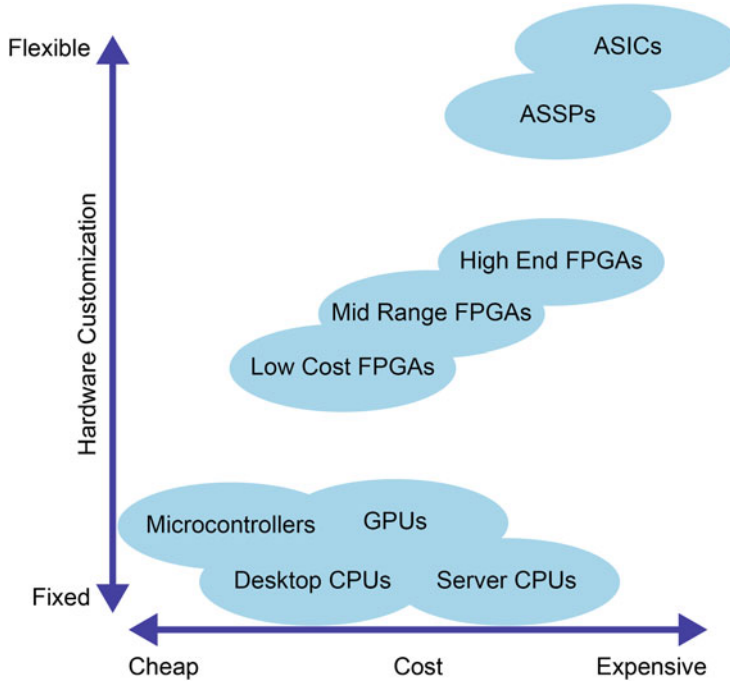

Fig. 1.1: Digital Hardware Devices. ASICs and ASSPs allow great flexibility for creating custom systems but are very expensive to develop. Microcontrollers are very cheap but are limited in their performance and flexibility. FPGAs take the middle ground where the FPGA fabric is programmable, which allows one to create custom hardware without the costs associated with ASICs and ASSPs

When using a CPU, customization is limited to what can be done in software. If you want to develop custom hardware, but without the costs associated with ASICs and ASSPS, this is where FPGAs come into play. FPGAs allow the development of custom hardware, but the NRE for developing the FPGA device has already been spread over thousands of FPGA customers.

What are the *advantages* of FPGAs compared to CPU systems? FPGAs have a programmable hardware fabric and custom I/O. This allows data to enter and exit the device with very low latency. The programmable fabric allows the creation of a custom data plane that gives high performance. An example of this is the creation of a custom interface and data plane for audio processing that we create in this book.

What are the *disadvantages* of FPGAs compared to CPU systems? FPGAs are much harder and take longer to develop. One needs to be familiar with computer architecture and low-level hardware description languages. When designing FPGA logic, one has to have a pretty good idea how the logic will be implemented in the FPGA fabric and what fabric building blocks will be used. If not, the design will be non-optimal and the design will not fit well in the device. Another disadvantage is

cost. You will not be using an FPGA for the new toaster oven design that can make use of a cheap micro-controller.

SoC FPGAs are ideal devices for learning computer engineering. In one device you can create custom hardware in the FPGA fabric, create Linux device drivers for your custom hardware, and then control the hardware from a software application in Linux. It allows you to understand how hardware and software interact and how to start thinking about hardware–software co-design where you partition up tasks that are best implemented in hardware and tasks that are best implemented as software. Understanding this system-level design will provide you with computer engineering skills that are in high demand. Knowing how hardware works is a great foundation for being a software developer, which is highlighted in the Alan Kay quote [6] "People who are really serious about software should make their own hardware."

1.2.1 Further Reading

A good introduction to FPGAs for those that are new to them is the ebook *FPGAs for Dummies* [7].

1.3 Prerequisites

The material covered in this textbook is quite broad, which means that the student should already be familiar with the topics listed below. As an analogy, we are starting out on an expedition to climb a mountain peak and the expedition requires that the expedition members, while not having direct experience climbing mountain peaks themselves, are familiar with camping, starting fires, and cooking outdoors, even when the weather is uncertain and could end up dismal and raining. However, summiting a peak on a clear sunny day where the vista stretches for miles makes the effort to get there all worth it. Being familiar with the topics below will ensure that you are OK camping in the woods by yourself.

1.3.1 Prior Hardware Knowledge

It is assumed that you are familiar with basic digital electronics and computer architecture as sketched below.

- **Basic Logic Gates** that includes CMOS logic and how NAND and NOR gates are used to construct digital logic. How these logic gates are used in both combinational and sequential logic designs. How numbers are represented in digital systems such as 2's complement numbers?

- **Basic Digital Components** that includes encoders/decoders, multiplexers, flip-flops, registers, adders, multipliers, finite state machines, etc.
- **Basic Computer Architecture** that includes memory (SRAM and DRAM), FIFOs, data path, pipelining, I/O, control, etc.

1.3.2 Prior Software Knowledge

It is assumed that you are familiar with basic programming concepts and have been exposed to the following three languages. Other languages used will be described as we use them (e.g., Python, TCL).

- **VHDL**, which is a hardware description language. This includes the std_logic_vector data type, entity, architecture, concurrent statements, processes, the rising_edge function, if/else and case control statements, etc. A recommended text for this subject matter is *The Designer's Guide to VHDL* by Peter Ashenden [8].
- **C**, which is widely used in embedded systems and in the Linux kernel. There are many online resources that can be uncovered by a quick "C tutorial" Google search.
- **Matlab**, which is the environment that we will use to create our audio processing systems. We will also use **Simulink** for creating our data plane models. A recommended resource for Matlab and Simulink is the Matlab Onramp [9] and Simulink Onramp [10].

1.4 Hardware Needed

The hardware needed for this book is listed in the following sections. The hardware was chosen so that students could purchase their own boards and hardware at minimal cost, which allows them to develop in their own room rather than having to go to a laboratory to use an FPGA board. The majority of students have laptops with Windows 10 installed as the operating system, so this is the PC configuration taken in this book. It is possible to use Linux as the operating system on your laptop or computer, but we will not take this approach. It is assumed that if a user already has Linux running on their laptop, then they are already capable of configuring Linux and installing software on their own if they choose to do so. And, if they have trouble, it is assumed that they are capable of figuring out their own Linux solutions.

1.4.1 Laptop

It is expected that students have their own laptop (or desktop computer) with the following capabilities:

- **Windows 10 Operating System**. Using another operating system is possible, but you are on your own if you choose to do so. This textbook assumes you are using Windows 10 on your laptop or desktop computer.
- **8 GB of RAM** (ideally 16 GB). Currently, 8 GB is the most common RAM size that you will find in laptops, which is adequate for the projects in this textbook. If you have less, it is still possible, but your computer will run slower, especially when using a virtual machine.
- **Wi-Fi**. You will need to connect to the Internet and practically all laptops come with Wi-Fi, so you should be good on this front.
- **Two USB Ports**. You will need two USB ports on your laptop to connect to two different connections on the FPGA board. One USB port will be used to program the FPGA via JTAG, and the other USB port will be used with a terminal window when booting Linux on the SoC FPGA. If you do not have two USB ports, *get a USB adapter for your laptop that has two USB ports and an Ethernet port*.
- **Ethernet Port**. You will need an Ethernet port that is in addition to the Internet Wi-Fi connection on your laptop. This is because you will be connecting to the FPGA board using an Ethernet cable. Some laptops do not have an Ethernet port with an RJ45 jack, so if this is your case, you will need to get an Ethernet adapter for your laptop. If you need to get an Ethernet adapter, get an adapter that has at least two USB ports as well.

1.4.2 DE10-Nano FPGA Board

The FPGA board that is used by this textbook is the DE10-Nano Kit produced by Terasic (www.terasic.com) that contains an Intel Cyclone V SoC FPGA. The reason this board was chosen was because it was the lowest cost SoC FPGA board making it possible for students to get their own board. It provides great value because it contains the largest SoC FPGA (110K LEs) in the low cost FPGA category. A comparison to other low cost SoC FPGA boards can be seen in Table 1.2, where it can be seen that (at the time of this writing) the DE10-Nano had the best value (Fig. 1.2).

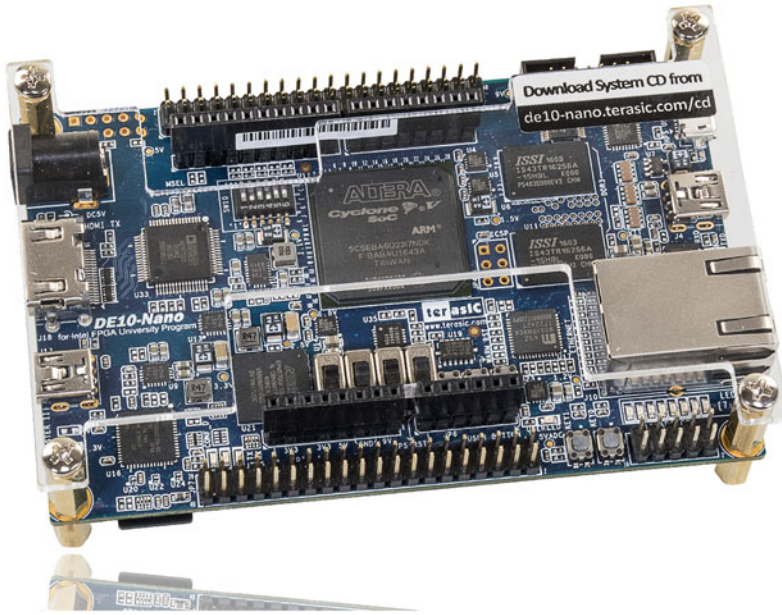

Fig. 1.2: The Terasic DE10-Nano SoC FPGA board is the FPGA board used by this textbook. https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=167&No=1046

1.4.3 Audio Board

The real-time system that students develop in this textbook targets audio signal processing. Since there was no audio codec on the DE10-Nano, nor was there an audio card available, a high fidelity audio board was created for the DE10-Nano. This audio board contains a 24-bit audio codec (Analog Devices AD1939) that can sample up to 192 kHz. Further information on how this audio board was designed and how it can be used is found in Chap. 5 Introduction to the Audio Mini Board. The audio board is shown in Fig. 1.3 and can be purchased from SensorLogic (Audio Mini Link).

1.4.4 Miscellaneous Hardware

Some additional hardware will need to be purchased as well:

- **microSD Card**. The DE10-Nano SoC FPGA board comes with a microSD card that allows it to boot Linux. We will be creating our own version, so you will

Table 1.2: Low cost SoC FPGA boards

Board	Price[1]	SoC FPGA	FPGA Fabric			ARM CPU[5]	
			LEs[2]	DSP Slices[3]	Memory[4]	Speed	DRAM
DE10-Nano	$146	Intel Cyclone V 5CSEBA6U23I7	110 K	112	696 KB	800 MHz	1 GB
DE1-SoC	$175	Intel Cyclone V 5CSEMA5F31C6	85 K	87	496 KB	800 MHz	1 GB
Alchitry Au	$100	Xilinx Artix 7 XC7A35T	33 K	90	225 KB	No ARM CPUs Not a SoC FPGA	256 MB (standalone)
Zybo Z7-10	$159	Xilinx Zynq XC7Z010	17 K	80	270 KB	667 MHz	1 GB
Zybo Z7-20	$239	Xilinx Zynq XC7Z020	53 K	220	630 KB	667 MHz	1 GB

[1]Educational price in 2021 not including shipping. [2]Intel's LE contains an 8-input lookup table. Xilinx's contains a 6-input lookup table. [3]Intel's DSP Slice contains a 27×27 bit multiplier and a 64-bit accumulator. Xilinx's DSP Slice contains a 18×25 bit multiplier and a 48-bit accumulator. [4]Intel's contains 10 Kb M10K blocks. Xilinx's contains 36 kb blocks. [5]Dual Core Cortex-A9

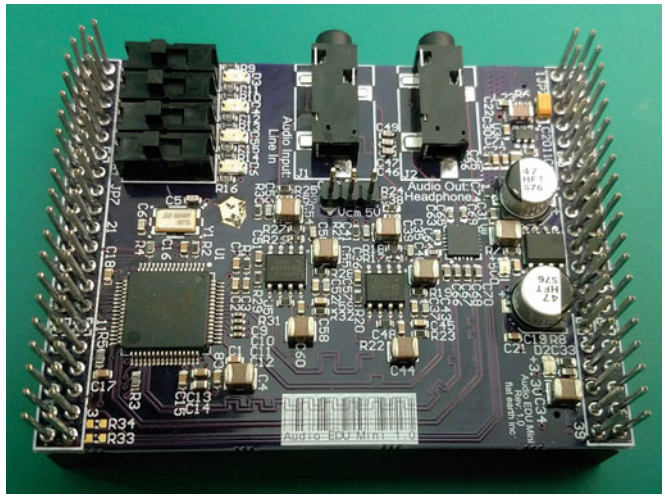

Fig. 1.3: Audio Mini Board that plugs into the DE10-Nano and contains a 24-bit audio codec (Analog Devices AD1939) that can sample up to 192 kHz

need another microSD card so that you can always plug the factory default image back into the DE10-Nano board. Any size greater than 8 GB is fine. Currently, a 32 GB is practically the same cost as an 8 GB, so you might as well get the 32 GB microSD card. You can always allocate the extra space to the Linux root file system on the DE10-Nano.

- **microSD Card Reader**. You will need a USB microSD Card Reader so that you can read/write the microSD card and modify the card images.
- **Type A to Mini-B USB Cable**. The DE10-Nano kit comes with one Type A to Mini-B USB Cable, but we will be using both Mini-B ports on the DE10-Nano, so having an extra cable will be more convenient.
- **Ethernet Cable**. A short Ethernet cable, ~1 foot (or longer), to connect the DE10-Nano board to your laptop.

1.5 Software Needed

The following software will be used in this textbook. Most of the software is free or there are free commercial versions with the exception of Matlab, which has a student version that one must buy if one is not associated with an institution with a Mathworks site license. The list below is given as an overview of the software that will be used. Instructions for setting up the software are found in Sect. 11.1 Software Setup.

- **Windows 10**. It is assumed that the student has a Windows 10 laptop or PC and does not have much experience with Linux.
- **Windows Subsystem for Linux**. *Windows Subsystem for Linux* (**WSL**) comes in two versions, WSL 1 and WSL 2. WSL 1 (and not WSL 2) is required for Intel's Quartus software.
- **VirtualBox**. We will create an Ubuntu *virtual machine* (**VM**) using VirtualBox on Windows 10.
- **Ubuntu 20.04 LTS** . We will install Ubuntu as a virtual machine in VirtualBox.
- **Matlab and Simulink**. The following toolboxes are required in Matlab:
  - HDL Coder
  - Matlab Coder
  - Simulink Coder
  - Fixed-Point Designer
  - DSP System Toolbox (strongly suggested). Required for some example designs.
  - Signal Processing Toolbox (strongly suggested). Required for some example designs.
- **Python**. Version 3.8.x or later
- **Quartus**. The free version Quartus Prime Lite Edition can be used for the Cyclone V FPGA. Note: Quartus requires WSL 1 with Ubuntu 18.04.
- **Putty**. Which is a terminal emulator.

1.6 The Development Landscape

Knowing where you need to be to implement certain development steps, type software commands, or install or run software can be confusing since we will be operating across *two different hardware platforms with two different CPU types:*

Platform 1: **DE10-Nano** FPGA board that contains an ARM CPU inside the Cyclone V SoC FPGA
Platform 2: **Laptop** or PC that contains an x86 CPU

and *three operating systems:*

OS 1: **Windows 10** on a Laptop or PC
OS 2: **Ubuntu VM**, which is Ubuntu Linux running on a virtual machine in VirtualBox that is running on Windows 10, which in turn is using an x86 CPU.
OS 3: **Ubuntu ARM**, which is Ubuntu Linux running on the ARM CPUs inside the Cyclone V SoC FPGA, which is on the DE10-Nano board. Furthermore, the **Root File System** for Ubuntu ARM on the DE10-Nano can be located in two different locations:

a. **On the microSD card** that is inserted into the DE10-Nano board. When Linux uses the root file system on the microSD card, this is known as the **Ship Boot Mode**. This is the setup that comes when you buy the DE10-Nano board, but it is not the setup that you want to develop with.
b. **In an Ubuntu VM folder** served over Ethernet by the Ubuntu VM NFS server. When Linux uses the root file system served by the Ubuntu VM, this is known as the **Developer Boot Mode**. We will be using this boot setup in this book because it is way more convenient to develop with than using the ship boot mode, which is not practical for development.

Each operating system has its own software packages, command lines, and terminal windows. It also means that the DE10-Nano FPGA board and the Laptop/PC can be connected in any one or in all of the following manners:

Connection 1: **USB JTAG** using a USB cable with a Mini-B connector plugged into the USB Blaster port on the left side of the DE10-Nano board. This connection is used to program the FPGA via JTAG.
Connection 2: **USB UART** using a USB cable with a Mini-B connector plugged into the UART port on the right side of the DE10-Nano board. This connection is used to create a terminal window to interact with Linux booting on the DE10-Nano.
Connection 3: **Ethernet** where an Ethernet cable connects the DE10-Nano to the Laptop/PC. This is used so that Linux can boot from the Ubuntu VM when using the Developer Boot Mode.

References

1. Semiconductor Engineering. 10nm-versus-7nm. https://semiengineering.com/10nm-versus-7nm/. Accessed 22 June 2022
2. semiengineering.com. 5nm-vs-3nm. https://semiengineering.com/5nm-vs-3nm/. Accessed 22 June 2022
3. Analog Devices. AD1939 Audio Codec. https://www.analog.com/en/products/ad1939.html. Accessed 22 June 2022
4. IC Insights. Microcontrollers Will Regain Growth After 2019 Slump. https://www.icinsights.com/news/bulletins/Microcontrollers-Will-Regain-Growth-After-2019-Slump/. Accessed 22 June 2022
5. CPU World. Intel Xeon 8180M Specifications. https://www.cpu.world.com/CPUs/Xeon/Intel-Xeon%208180M.html. Accessed 22 June 2022
6. A. Kay, Alan Kay Quote. https://en.wikiquote.org/wiki/Alan_Kay. Accessed 22 June 2022

7. A. Moore, R. Wilson, FPGAs For Dummies (2017). https://www.intel.com/content/dam/support/us/en/programmable/support-resources/bulk-container/pdfs/literature/misc/fpgas-for-dummies-ebook.pdf
8. P.J. Ashenden, *The Designer's Guide to VHDL* (Morgan Kaufmann, Burlington, 2008)
9. MathWorks. MATLAB Onramp. https://www.mathworks.com/learn/tutorials/matlab.onramp.html. Accessed 22 June 2022
10. MathWorks. Simulink Onramp. https://www.mathworks.com/learn/tutorials/simulink-onramp.html. Accessed 22 June 2022