

Chapter 1

So you got your first IXP12xx, now what?

“In theory there is no difference between theory and practice. In practice there is.” – Unknown

This book is going to show you the practical side of programming the 12xx series of Intel’s Internet eXchange network Processors (IXP). Much has been written about network processors in the last few years. Perhaps too much, because nowadays defining the term ‘network processor’ involves writing a dissertation. So while network processor definitions are different, the motivations for using network processors are similar. Ostensibly, network processors enable you to add, through software, the latest-and-greatest network service while maintaining high packet throughput and low packet latency. Of course, since we are talking about services delivered via software, as opposed to through hardware such as Application Specific Integrated Circuits (ASICs), we should be able to get these latest-and-greatest network services out faster, right? Well, in theory, maybe.

In practice, the time-to-market advantages provided by implementing a new network service in software on a network processor can be offset by the steep learning curve for programmers of the network processor. The new tools, languages, and programming environments used to help network processors achieve their high performance can quickly humble even the most experienced programmer. The IXP12xx processor is no exception.

The IXP12xx processor offers a wide range of programmability through hardware units and co-processors specifically tuned to network programming, in addition to a general-purpose core processor. Due to this specialization, the IXP12xx can provide flexible programmability without sacrificing perform-

ance as might be expected with general-purpose processors. At the same time, the new programming paradigm presented by the IXP12xx can be daunting. When first programming on the IXP12xx, accomplishing seemingly trivial tasks such as receiving or transmitting packets has been known to invoke an all-out engineer's victory dance. This initial programming experience can be especially frustrating for programmers already well-versed in the standard write-debug-optimize programming methods. The languages, tools, and optimization techniques are all different from what you may be used to. Optimizing code is less about finding ways to execute fewer lines of code and more about understanding ways to offload work to the hardware.

Is this book for you?

Presumably you are reading this book because you are somehow already invested in the IXP12xx processor. Let us guess, your manager has asked you to develop, or evaluate the suitability of, the next great network product on the IXP12xx. You have a feature set, short timeline, and some helpful URL like www.intel.com; thank you very much.

You may have tried to read the IXP12xx data sheet and various reference manuals. Perhaps you downloaded and installed the IXP12xx Software Development Kit (SDK), but exploring the complete microcode reference designs left you wondering where to begin. And if you did get started with the reference microcode, you might still be wondering why the microcode does things the way it does, especially the mysterious receive process and transmit process.

Additionally, you might even have hardware available, but beyond the blinking LED that you get when you turn it on, nothing seems familiar or friendly.

If you are facing any of the above problems and want practical advice suited to a software person, this book is our attempt to help.

It isn't that scary

The IXP12xx can be programmed, and software people like us can program it. You'll find that while much of the programming environment, like the language, instruction set and tools, is unfamiliar, an equal amount is familiar including: threads, mutexes, critical sections, and signals. The trick is to filter out the information about the hardware that is relevant to a programmer. This is the purpose of this book.

What is most familiar for network programmers when programming the IXP12xx is the basic packet flow of receive, process, then transmit. The process step of the flow can range from simple operations like bridging, switching, or routing to more complex quality of service (QoS) metering, marking, and policing, to access control lists (ACLs) and other forms of deep packet inspection. In theory, implementing the latest-and-greatest network service on the IXP12xx means adding more code to the packet-processing step within this basic packet flow. In practice, implementing the latest-and-greatest network service on the IXP12xx means understanding what the IXP12xx hardware will do for your code. Hiding memory latencies through signals, and proper data structure design to reduce synchronization is the kind of practical knowledge you need.

Typically, programmers focus on the packet processing step while the receive and transmit operations are blissfully ignored. This is not the case on the IXP12xx. The flexibility of the receive and transmit processes on the IXP12xx dictates the overall threading model of the software. However, as a result of this flexibility, these same processes are also complicated. The choices made on the receive and transmit operations have a profound effect on overall system performance. Thus, understanding the fundamentals of receive and transmit operations on the IXP12xx is another piece of practical knowledge you need for effective IXP12xx programming.

Luckily, all of this knowledge can be learned without resorting to reading hardware schematics or wheeling the logic analyzer into your cubicle. We'll show you how to get this knowledge by presenting the details as they relate to your software. So if you are interested in the practice of effective IXP12xx microengine programming we'll take you from start to finish and along the way hopefully find time for a few victory dances.

A guide to the rest of the book

The following summary of the book's chapters includes notes on which chapters can be read independently, which chapters should be read together, and which chapters can be skimmed or skipped if you are already familiar with IXP12xx hardware and software tools.

- Chapter 2 is an overview of the IXP12xx hardware and software tools. The hardware details are presented from a programmer's perspective. The software overview covers the main components of the IXA Software Development Kit (SDK) version 2.0. Readers familiar with the IXP12xx hardware architecture and the IXA SDK 2.0 can

safely skim this chapter. The remaining chapters assume knowledge of the information in this chapter.

- Chapter 3 is a getting started guide to writing microengine C on the IXP12xx. Readers already comfortable with using the Developer's Workbench to compile and debug a microengine C program can skip this chapter. The remaining chapters assume the reader can use the Developer's Workbench to compile and debug microengine C programs.
- Chapter 4 is a brief overview of the microengine coding philosophy followed throughout the book. Microengine code, like all software, can be written and organized in a myriad of ways, ranging from task-specific, monolithic blobs, to completely decoupled, reusable code components. Any microengine code organization pits performance against flexibility and reusability. The approach that we found most successful is a balance of performance and modularity, called microblocks. This chapter provides insight into the microblock code structure used in Chapters 4 through 11.
- Chapters 5 through 7 progressively build upon the simple application of receiving and counting packets. Chapter 5 accomplishes these tasks with a single thread focusing on understanding the packet reception process. Chapter 6 extends the example to run on multiple threads within a single microengine, and Chapter 7 expands to show how to run multiple threads on multiple microengines. Chapters 5 through 7 should be read in order.
- Chapter 8 replaces the simple counting application of Chapters 5 through 7, with a more realistic network application: Ethernet bridging. The focus is on proper design of data structures and algorithms to take advantage of the IXP12xx. This chapter is best read in conjunction with Chapters 5 through 7.
- Chapter 9 completes the examples of Chapters 5 through 8 by adding in details of transmitting packets with IXP12xx. This chapter can be read independently, but the final code examples build upon the code from Chapter 8.
- Chapter 10 covers advanced programming topics on the IXP12xx. Included in this chapter are subtle features of the receive and transmit processes, Cyclic Redundancy Check (CRC) features of the IXP1240 and IXP1250, and the ready-bus sequencer programming. The receive and transmit details depend on the previous chapters, but the other topics in this chapter can be read independently.

- Chapter 11 details how to write a microACE. The microACE framework allows a programmer to integrate fast-path microengine packet processing with configuration and exception-path packet processing on the Intel® StrongARM† core of the IXP12xx. This chapter assumes an understanding of writing programs in microengine C, but can otherwise be read independently of the other chapters.
- Chapter 12 is a set of IXP12xx programming tips, tricks, and common mistakes. This chapter can be read independently from the other chapters.
- Chapter 13 covers differences between the IXP12xx and the next generation IXP2xxx. Assuming an understanding of the current generation IXP12xx's, this chapter can be read independently from the other chapters.

Conventions

Throughout the book we will use the terms byte, word, long-word, and quad-word to represent 8, 16, 32, and 64 bits of data respectively.

All but the most trivial microengine code examples are available on the accompanying CD-ROM. To help you link the code in the book to the proper file on the CD-ROM, the format for most code examples contains the appropriate information to locate the example code. When exceptions to this rule are made, for example on code segments too small to merit this overhead, pointers are added to the code on the CD-ROM separately in the text.

The following is an example of the code format used in the book.

```

Function() – if appropriate
File:      ChapterX\


---



```

The function, file, and project names are provided when appropriate. Each actual line of code has a line number to make it easy to dissect the code and refer to individual code segments in the text.

The inherent lag time in writing and publishing a book, coupled with the inevitable progression of technology, mean that even as this book comes out, future generations of the IXP12xx processor family will be in the works.

6 ■ IXP1200 Programming

Where appropriate, we will use notes like the following one to point out these differences.

Note

Throughout the book look for notes like this one. These notes point out differences in the next generation IXP2xxx hardware's programming. While these notes are based on the latest available information, they are not guarantees of future functionality.