COMPUTER SYSTEM DESIGN

System-on-Chip

Michael J. Flynn

Wayne Luk

WILEY

A JOHN WILEY & SONS, INC., PUBLICATION

COMPUTER SYSTEM DESIGN

COMPUTER SYSTEM DESIGN

System-on-Chip

Michael J. Flynn

Wayne Luk

WILEY

A JOHN WILEY & SONS, INC., PUBLICATION

Published by John Wiley & Sons, Inc., Hoboken, New Jersey.
Published simultaneously in Canada.

Library of Congress Cataloging-in-Publication Data:

Flynn, M. J. (Michael J.), 1934–
Computer system design : system-on-chip / Michael J. Flynn, Wayne Luk.
p. cm.
Includes bibliographical references and index.
ISBN 978-0-470-64336-5 (hardback)
1. Systems on a chip. I. Luk, Wayne. II. Title.
TK7895.E42F65 2011
004.1–dc22

2010040981

Printed in Singapore

oBook ISBN: 9781118009925
ePDF ISBN: 9781118009901
ePub ISBN: 9781118009918

10 9 8 7 6 5 4 3 2 1

CONTENTS

PREFACE

The next generation of computer system designers will be concerned more about the elements of a system tailored to particular applications than with the details of processors and memories.

Such designers would have rudimentary knowledge of processors and other elements in the system, but the success of their design would depend on their skills in making system-level trade-offs that optimize the cost, performance, and other attributes to meet application requirements.

This text is organized to introduce issues in computer system design, particularly for system-on-chip (SOC). Managing such design requires knowledge of a number of issues, as shown in Figure 1.

After Chapter 1, the introduction chapter, Chapter 2 looks at issues that define the design space: area, speed, power consumption, and configurability. Chapters 3–5 provide background knowledge of the basic elements in a system: processor, memory, and interconnect.

The succeeding chapters focus on computer systems tailored to specific applications and technologies. Chapter 6 covers issues in customizing and configuring designs. Chapter 7 addresses system-level trade-offs for various applications, bringing together earlier material in this study. Finally, Chapter 8 presents future challenges for system design and SOC possibilities.

The tools that illustrate the material in the text are still being developed. The Appendix provides an overview of one such tool. Since our tools are evolving, please check from time to time to see what is available at the companion web site: www.soctextbook.com.

Moreover, material useful for teaching, such as slides and answers to exercises, is also being prepared.

This book covers a particular approach to computer system design, with emphasis on fundamental ideas and analytical techniques that are applicable to a range of applications and architectures, rather than on specific applications, architectures, languages, and tools. We are aware of complementary treatments on these and also on other topics, such as electronic system-level design, embedded software development, and system-level integration and test. We have included brief descriptions and references to these topics where appropriate; a more detailed treatment can be covered in future editions or in different volumes.

SOC is a quickly developing field. Although we focused on fundamental material, we were forced to draw a line on the inclusion of the latest

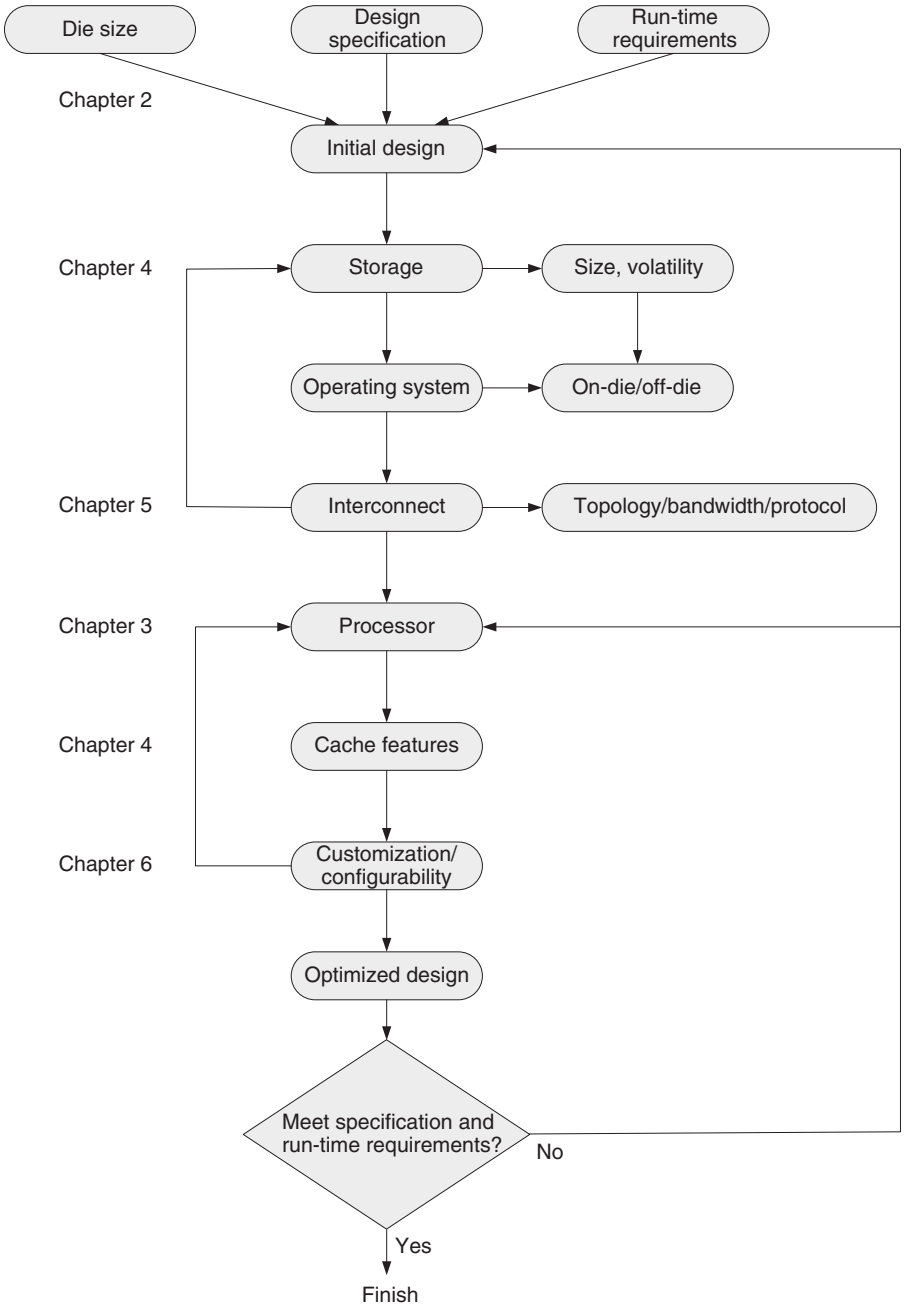

Figure 1 An approach to SOC system design described in this book.

technological advances for the sake of completing the book. Such advances, instead, are captured as links to relevant sources of information at the companion web site described above.

Many colleagues and students, primarily at Imperial College London and Stanford University, have contributed to this book. We are sorry that we are not able to mention them all by name here. However, a number of individuals deserve special acknowledgment. Peter Cheung worked closely with us from the beginning; his contributions shaped the treatment of many topics, particularly those in Chapter 5. Tobias Becker, Ray Cheung, Rob Dimond, Scott Guo, Shay Ping Seng, David Thomas, Steve Wilton, Alice Yu, and Chi Wai Yu contributed significant material to various chapters. Philip Leong and Roger Woods read the manuscript many times carefully and provided many excellent suggestions for improvement. We also greatly benefited from comments by Jeffrey Arnold, Peter Boehm, Don Bouldin, Geoffrey Brown, Patrick Hung, Sebastian Lopez, Oskar Mencer, Kevin Rudd, and several anonymous reviewers. We thank Kubilay Atasu, Peter Collingbourne, James Huggett, Qiwei Jin, Adrien Le Masle, Pete Sedcole, and Tim Todman, as well as those who prefer to remain anonymous, for their invaluable assistance.

Last, but not least, we thank Cassie Strickland, of Wiley, and Janet Hronek, of Toppan Best-set, for their help in the timely completion of this text.

LIST OF ABBREVIATIONS AND ACRONYMS

AC	Autonomous chip
A/D	Analog to digital
AES	Advanced Encryption Standard
AG	Address generation
ALU	Arithmetic and logic unit
AMBA	Advanced Microcontroller Bus Architecture
ASIC	Application-specific integrated circuit
ASIP	Application-specific instruction processor
ASOC	Autonomous system-on-chip
AXI	Advanced eXtensible Interface
BC	Branch conditional
BIST	Built-in-self-test
BRAM	Block random access memory
BTB	Branch target buffer
CAD	Computer aided design
CBWA	Copy-back write allocate cache
CC	Condition codes
CFA	Color filter array
CGRA	Coarse-grained reconfigurable architecture
CIF	Common Intermediate Format
CISC	Complex instruction set computer
CLB	Configurable Logic Block
CMOS	Complementary metal oxide semiconductor
CORDIC	COordinate Rotation Digital Computer
CPI	Cycles per instruction
CPU	Central processing unit
DCT	Discrete Cosine Transform
DDR	Double data rate
DES	Data Encryption Standard
3DES	Triple Data Encryption Standard

DF	Data fetch
DMA	Direct memory access
DRAM	Dynamic random access memory
DSP	Digital signal processing (or processor)
DTMR	Design Target Miss Rates
ECC	Error correcting code
eDRAM	Embedded dynamic random access memory
EX	Execute
FIFO	First in first out
FIR	Finite impulse response
FO4	Fan-out of four
FP	Floating-point
FPGA	Field programmable gate array
FPR	Floating-point register
FPU	Floating-point unit
GB	Giga bytes, a billion (10^9) bytes
GIF	Graphics interface
GPP	General-purpose processor
GPR	General-purpose register
GPS	Global Positioning System
GSM	Global System for Mobile Communications
HDTV	High definition television
HPC	High performance computing
IC	Integrated circuit
ICU	Interconnect interface unit
ID	Instruction decode
IF	Instruction fetch
ILP	Instruction-level parallelism
I/O	Input/output
IP	Intellectual property
IR	Instruction register
ISA	Instruction set architecture
JPEG	Joint Photographic Experts Group (image compression standard)
Kb	Kilo bits, one thousand (10^3) bits
KB	Kilo bytes, one thousand bytes
L1	Level 1 (for cache)
L2	Level 2 (for cache)
LE	Logic Element

LRU	Least recently used
L/S	Load-store
LSI	Large scale integration
LUT	Lookup table
Mb	Mega bits, one million (10^6) bits
MB	Mega bytes, one million bytes
MEMS	Micro electro mechanical systems
MIMD	Multiple instruction streams, multiple data streams
MIPS	Million instructions per second
MOPS	Million operations per second
MOS	Metal oxide semiconductor
MPEG	Motion Picture Experts Group (video compression standard)
MTBF	Mean time between faults
MUX	Multiplexor
NOC	Network on chip
OCP	Open Core Protocol
OFDM	Orthogonal Frequency-Division Multiplexing
PAN	Personal area network
PCB	Printed circuit board
PLCC	Plastic leaded chip carrier
PROM	Programmable read only memory
QCIF	Quarter Common Intermediate Format
RAM	Random access memory
RAND	Random
RAW	Read-after-write
rbe	Register bit equivalent
RF	Radio frequency
RFID	Radio frequency identification
RISC	Reduced instruction set computer
R/M	Register-memory
ROM	Read only memory
RTL	Register transfer language
SAD	Sum of the absolute differences
SDRAM	Synchronous dynamic random access memory
SECDED	Single error correction, double error detection
SER	Soft error rate
SIA	Semiconductor Industry Association
SIMD	Single instruction stream, multiple data streams
SMT	Simultaneous multithreading

SOC	System on chip
SRAM	Static random access memory
TLB	Translation look-aside buffer
TMR	Triple modular redundancy
UART	Universal asynchronous receiver/transmitter
UMTS	Universal mobile telecommunications system
UV	Ultraviolet
VCI	Virtual Component Interface
VLIW	Very long instruction word
VLSI	Very large scale integration
VPU	Vector processing unit
VR	Vector register
VSIA	Virtual Socket Interface Alliance
WAR	Write after read
WAW	Write after write
WB	Write back
WTNWA	Write-through cache, no write allocate

1 Introduction to the Systems Approach

1.1 SYSTEM ARCHITECTURE: AN OVERVIEW

The past 40 years have seen amazing advances in silicon technology and resulting increases in transistor density and performance. In 1966, Fairchild Semiconductor [84] introduced a quad two input NAND gate with about 10 transistors on a die. In 2008, the Intel quad-core Itanium processor has 2 billion transistors [226]. Figures 1.1 and 1.2 show the unrelenting advance in improving transistor density and the corresponding decrease in device cost.

The aim of this book is to present an approach for computer system design that exploits this enormous transistor density. In part, this is a direct extension of studies in computer architecture and design. However, it is also a study of system architecture and design.

About 50 years ago, a seminal text, *Systems Engineering—An Introduction to the Design of Large-Scale Systems* [111], appeared. As the authors, H.H. Goode and R.E. Machol, pointed out, the system's view of engineering was created by a need to deal with complexity. As then, our ability to deal with complex design problems is greatly enhanced by computer-based tools.

A system-on-chip (SOC) architecture is an ensemble of processors, memories, and interconnects tailored to an application domain. A simple example of such an architecture is the Emotion Engine [147, 187, 237] for the Sony PlayStation 2 (Figure 1.3), which has two main functions: behavior simulation and geometry translation. This system contains three essential components: a main processor of the reduced instruction set computer (RISC) style [118] and two vector processing units, VPU0 and VPU1, each of which contains four parallel processors of the single instruction, multiple data (SIMD) stream style [97]. We provide a brief overview of these components and our overall approach in the next few sections.

While the focus of the book is on the system, in order to understand the system, one must first understand the components. So, before returning to the issue of system architecture later in this chapter, we review the components that make up the system.

Computer System Design: System-on-Chip, First Edition. Michael J. Flynn and Wayne Luk.

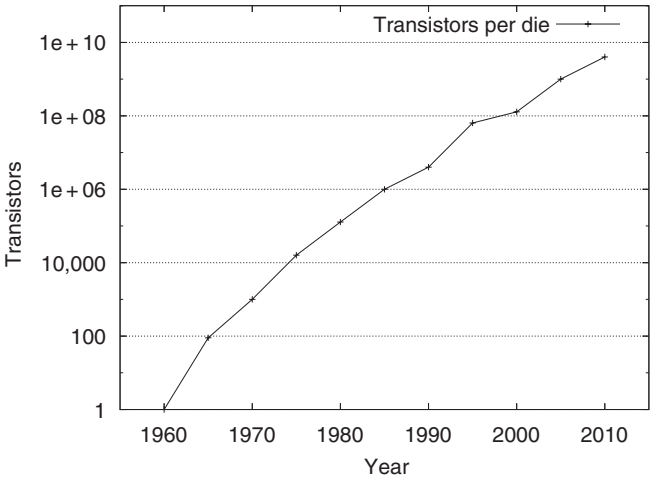

Figure 1.1 The increasing transistor density on a silicon die.

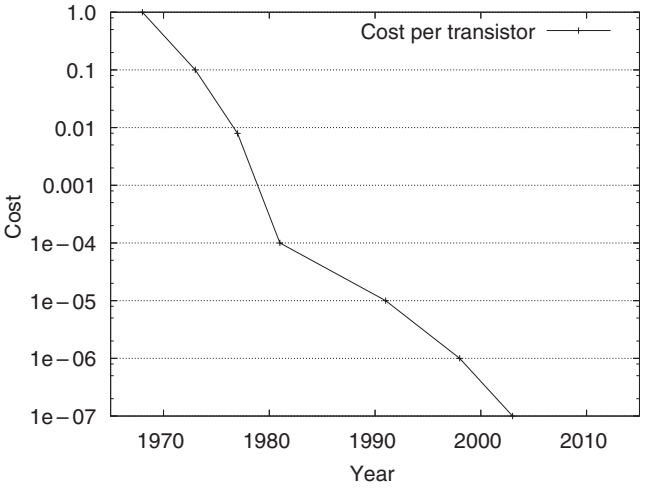

Figure 1.2 The decrease of transistor cost over the years.

1.2 COMPONENTS OF THE SYSTEM: PROCESSORS, MEMORIES, AND INTERCONNECTS

The term *architecture* denotes the operational structure and the user's view of the system. Over time, it has evolved to include both the functional specification and the hardware implementation. The system architecture defines the system-level building blocks, such as processors and memories, and the

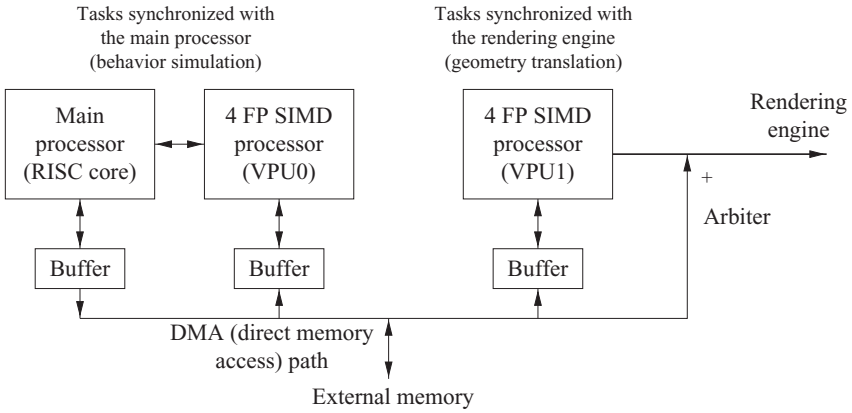

Figure 1.3 High-level functional view of a system-on-chip: the Emotion Engine of the Sony PlayStation 2 [147, 187].

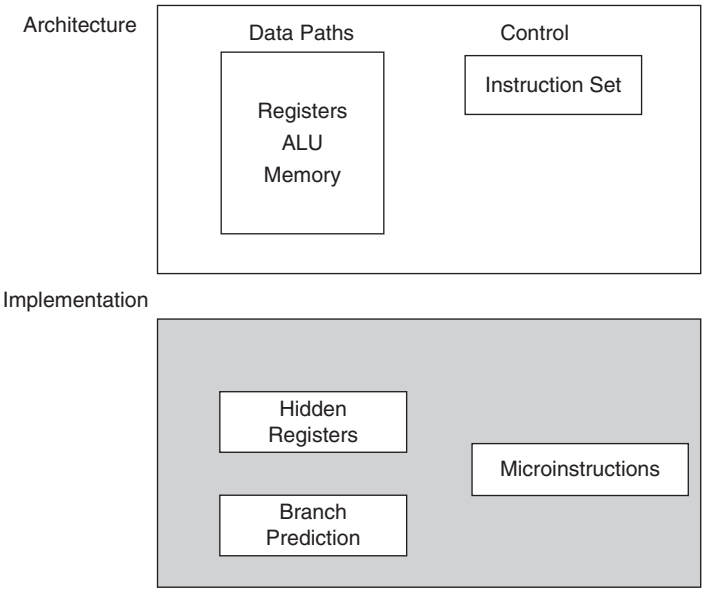

Figure 1.4 The processor architecture and its implementation.

interconnection between them. The processor architecture determines the processor's instruction set, the associated programming model, its detailed implementation, which may include hidden registers, branch prediction circuits and specific details concerning the ALU (arithmetic logic unit). The implementation of a processor is also known as *microarchitecture* (Figure 1.4).

The system designer has a programmer's or user's view of the system components, the system view of memory, the variety of specialized processors, and

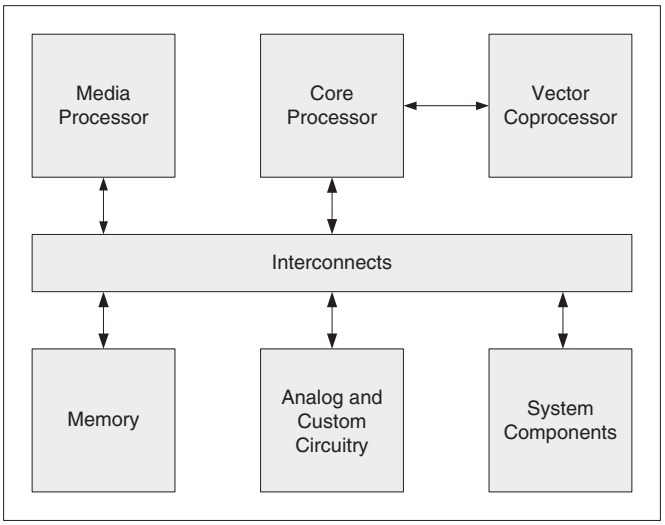

Figure 1.5 A basic SOC system model.

their interconnection. The next sections cover basic components: the processor architecture, the memory, and the bus or interconnect architecture.

Figure 1.5 illustrates some of the basic elements of an SOC system. These include a number of heterogeneous processors interconnected to one or more memory elements with possibly an array of reconfigurable logic. Frequently, the SOC also has analog circuitry for managing sensor data and analog-to-digital conversion, or to support wireless data transmission.

As an example, an SOC for a smart phone would need to support, in addition to audio input and output capabilities for a traditional phone, Internet access functions and multimedia facilities for video communication, document processing, and entertainment such as games and movies. A possible configuration for the elements in Figure 1.5 would have the core processor being implemented by several ARM Cortex-A9 processors for application processing, and the media processor being implemented by a Mali-400MP graphics processor and a Mali-VE video engine. The system components and custom circuitry would interface with peripherals such as the camera, the screen, and the wireless communication unit. The elements would be connected together by AXI (Advanced eXtensible Interface) interconnects.

If all the elements cannot be contained on a single chip, the implementation is probably best referred to as a system on a board, but often is still called a SOC. What distinguishes a system on a board (or chip) from the conventional general-purpose computer plus memory on a board is the specific nature of the design target. The application is assumed to be known and specified so that the elements of the system can be selected, sized, and evaluated during the design process. The emphasis on selecting, parameterizing, and configuring system components tailored to a target application distinguishes a system architect from a computer architect.

In this chapter, we primarily look at the higher-level definition of the processor—the programmer's view or the instruction set architecture (ISA), the basics of the processor microarchitecture, memory hierarchies, and the interconnection structure. In later chapters, we shall study in more detail the implementation issues for these elements.

1.3 HARDWARE AND SOFTWARE: PROGRAMMABILITY VERSUS PERFORMANCE

A fundamental decision in SOC design is to choose which components in the system are to be implemented in hardware and in software. The major benefits and drawbacks of hardware and software implementations are summarized in Table 1.1.

A software implementation is usually executed on a general-purpose processor (GPP), which interprets instructions at run time. This architecture offers flexibility and adaptability, and provides a way of sharing resources among different applications; however, the hardware implementation of the ISA is generally slower and more power hungry than implementing the corresponding function directly in hardware without the overhead of fetching and decoding instructions.

Most software developers use high-level languages and tools that enhance productivity, such as program development environments, optimizing compilers, and performance profilers. In contrast, the direct implementation of applications in hardware results in custom application-specific integrated circuits (ASICs), which often provides high performance at the expense of programmability—and hence flexibility, productivity, and cost.

Given that hardware and software have complementary features, many SOC designs aim to combine the individual benefits of the two. The obvious method is to implement the performance-critical parts of the application in hardware, and the rest in software. For instance, if 90% of the software execution time of an application is spent on 10% of the source code, up to a 10-fold speedup is achievable if that 10% of the code is efficiently implemented in hardware. We shall make use of this observation to customize designs in Chapter 6.

Custom ASIC hardware and software on GPPs can be seen as two extremes in the technology spectrum with different trade-offs in programmability and

TABLE 1.1 Benefits and Drawbacks of Software and Hardware Implementations

	Benefits	Drawbacks
Hardware	Fast, low power consumption	Inflexible, unadaptable, complex to build and test
Software	Flexible, adaptable, simple to build and test	Slow, high power consumption

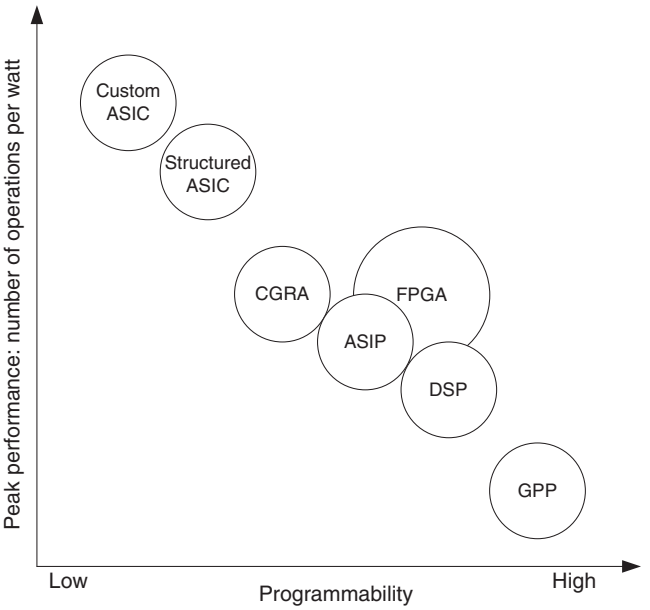

Figure 1.6 A simplified technology comparison: programmability versus performance. GPP, general-purpose processor; CGRA, coarse-grained reconfigurable architecture.

performance; there are various technologies that lie between these two extremes (Figure 1.6). The two more well-known ones are application-specific instruction processors (ASIPs) and field-programmable gate arrays (FPGAs).

An ASIP is a processor with an instruction set customized for a specific application or domain. Custom instructions efficiently implemented in hardware are often integrated into a base processor with a basic instruction set. This capability often improves upon the conventional approach of using standard instruction sets to fulfill the same task while preserving its flexibility. Chapters 6 and 7 explore further some of the issues involving custom instructions.

An FPGA typically contains an array of computation units, memories, and their interconnections, and all three are usually programmable in the field by application builders. FPGA technology often offers a good compromise: It is faster than software while being more flexible and having shorter development times than custom ASIC hardware implementations; like GPPs, they are offered as off-the-shelf devices that can be programmed without going through chip fabrication. Because of the growing demand for reducing the time to market and the increasing cost of chip fabrication, FPGAs are becoming more popular for implementing digital designs.

Most commercial FPGAs contain an array of fine-grained logic blocks, each only a few bits wide. It is also possible to have the following:

- *Coarse-Grained Reconfigurable Architecture (CGRA).* It contains logic blocks that process byte-wide or multiple byte-wide data, which can form building blocks of datapaths.
- *Structured ASIC.* It allows application builders to customize the resources before fabrication. While it offers performance close to that of ASIC, the need for chip fabrication can be an issue.
- *Digital Signal Processors (DSPs).* The organization and instruction set for these devices are optimized for digital signal processing applications. Like microprocessors, they have a fixed hardware architecture that cannot be reconfigured.

Figure 1.6 compares these technologies in terms of programmability and performance. Chapters 6–8 provide further information about some of these technologies.

1.4 PROCESSOR ARCHITECTURES

Typically, processors are characterized either by their application or by their architecture (or structure), as shown in Tables 1.2 and 1.3. The requirements space of an application is often large, and there is a range of implementation options. Thus, it is usually difficult to associate a particular architecture with a particular application. In addition, some architectures combine different implementation approaches as seen in the PlayStation example of Section 1.1. There, the graphics processor consists of a four-element SIMD array of vector processing functional units (FUs). Other SOC implementations consist of multiprocessors using very long instruction word (VLIW) and/or superscalar processors.

TABLE 1.2 Processor Examples as Identified by Function

Processor Type	Application
Graphics processing unit (GPU)	3-D graphics; rendering, shading, texture
Digital signal processor (DSP)	Generic, sometimes used with wireless
Media processor	Video and audio signal processing
Network processor	Routing, buffering

TABLE 1.3 Processor Examples as Identified by Architecture

Processor Type	Architecture/Implementation Approach
SIMD	Single instruction applied to multiple functional units (processors)
Vector (VP)	Single instruction applied to multiple pipelined registers
VLIW	Multiple instructions issued each cycle under compiler control
Superscalar	Multiple instructions issued each cycle under hardware control

From the programmer's point of view, sequential processors execute one instruction at a time. However, many processors have the capability to execute several instructions concurrently in a manner that is transparent to the programmer, through techniques such as pipelining, multiple execution units, and multiple cores. Pipelining is a powerful technique that is used in almost all current processor implementations. Techniques to extract and exploit the inherent parallelism in the code at compile time or run time are also widely used.

Exploiting program parallelism is one of the most important goals in computer architecture.

Instruction-level parallelism (ILP) means that multiple operations can be executed in parallel within a program. ILP may be achieved with hardware, compiler, or operating system techniques. At the loop level, consecutive loop iterations are ideal candidates for parallel execution, provided that there is no data dependency between subsequent loop iterations. Next, there is parallelism available at the procedure level, which depends largely on the algorithms used in the program. Finally, multiple independent programs can execute in parallel.

Different computer architectures have been built to exploit this inherent parallelism. In general, a computer architecture consists of one or more interconnected processor elements (PEs) that operate concurrently, solving a single overall problem.

1.4.1 Processor: A Functional View

Table 1.4 shows different SOC designs and the processor used in each design. For these examples, we can characterize them as general purpose, or special purpose with support for gaming or signal processing applications. This functional view tells little about the underlying hardware implementation. Indeed, several quite different architectural approaches could implement the same generic function. The graphics function, for example, requires shading, rendering, and texturing functions as well as perhaps a video function. Depending

TABLE 1.4 Processor Models for Different SOC Examples

SOC	Application	Base ISA	Processor Description
Freescale e600 [101]	DSP	PowerPC	Superscalar with vector extension
ClearSpeed CSX600 [59]	General	Proprietary ISA	Array processor of 96 processing elements
PlayStation 2 [147, 187, 237]	Gaming	MIPS	Pipelined with two vector coprocessors
ARM VFP11 [23]	General	ARM	Configurable vector coprocessor