

---

# Chapter 1

## Microprocessors, Platforms, and Systems

---

This chapter examines the process of designing and implementing a microprocessor and then examines issues that arise in the process of designing and implementing a 3D graphics PC platform. Toward the end of the chapter, several Eckert-Mauchly award winners share their insights regarding important references in the field of computer architecture. Although this chapter is written for each of the books intended<sup>1</sup> audiences, there are some subsections that have details that will be of more interest and use to practitioners and those in universities. The following *road map* for this chapter identifies these sections.

### ROAD MAP OF CHAPTER 1

Section	Audience
<i>The following are more detailed subsections:</i> Model at the Gate and Circuit Levels Gate and Circuit Level Simulation and Hardware Emulation Generate Netlists and Physical Layout Mask Generation, Wafer Fabrication, and Packaging	Practitioners, University Professors, and Students

What makes microprocessors especially difficult to design and implement are their complex nature, large size as measured in the number of electronic devices required in their implementation and the number of highly dependent trade-offs that must be made to achieve given size, yield, voltage, power, temperature, price, and performance points. The situation

**DESIGNING AND  
IMPLEMENTING A  
MICROPROCESSOR**

---

<sup>1</sup> See the Preface

hardware-software co-design

is complicated by the fact that the performance of the microprocessor *should not* be looked at independently of how it is integrated into a platform and a system. Issues such as the performance characteristics (bandwidth, latency, clock rate, etc.) of the core logic chipset, the memory architecture, the number and types of buses and their characteristics, and the I/O device support integrated onto the motherboard must be considered in a number of the design trade-offs. It makes little sense to have the fastest microprocessor in the world integrated into a system with an inappropriate bus or memory architecture or I/O support.

This implies that the core logic chipset, bus, memory architecture, and motherboard design teams must work intimately with the microprocessor design team. Each of these system elements is also significantly impacted by the rate of increases in clock speed and circuit density as well as important changes in packaging technology. Instead of advocating *hardware-software co-design*, we advocate *hardware-system co-design*. The “hardware” component of hardware-software co-design often refers only to processor design. Even when used in this restricted context, the hardware-software co-design process has substantive advantages when contrasted with the approach used just a few years ago of designing a processor without on-going, detailed interactions with software teams, such as compiler writers and operating systems implementers, that ultimately leads to the co-evolution, testing, and integration of the processor and the software.

ARTICLE ON CD-ROM



An insightful article that clearly shows the dependent nature of processor and compiler design is, “Compiler Technology for Future Microprocessors,” by Wen-Mei Hwu, Richard E. Hank, Daniel M. Lavery, Grant E. Haab, John C. Gyllenhaal, and David I. August, *Proceedings of the IEEE*, December 1995.

hardware-system co-design

We use the term *hardware-system co-design* to extend these notions to include on-going, detailed interactions with the teams involved in the design of the core logic chipset, the BIOS, the bus and memory architectures, the motherboard and any special device controller chips that will be integrated on the motherboard, (e.g., graphics controller chip, a high-speed FAX-data-voice modem chip, or a high-performance network controller chip). The second half of this book deals with many of these issues. In the first half, we focus on the microarchitecture of the microprocessor.

## WHAT NEEDS TO BE DONE?

How do you go about implementing a microprocessor? What needs to be done can be stated quite simply: We need to take the high-level instruction set architecture representation or model of the microprocessor's architecture and transform it into a correctly working, high-performance silicon chip that will have a long, failure-free life. Doing that is, of course, not so simple.

*We use the terms “representation” and “model” interchangeably in this discussion.*

Models play an integral role in the *transformation* process. Models are used to explore and analyze alternative design solutions and to verify behavioral (functional) specifications, compatibility requirements, and adherence to standards. A model—a representation of the microprocessor expressed in some language—is typically implemented in a simulator, emulator, or a combination of both. The model can represent the microprocessor at the architecture, microarchitecture, logical implementation, or chip level, or it can span one or more of these levels. While computer architects typically tend to deal with issues at the architecture and microarchitecture levels, digital design engineers tend to be more directly involved with issues at the logical implementation and chip levels. As a result, these two groups often approach the modeling and simulations issues from different perspectives and often with a different set of terminology and tools. For example, some digital system designers might refer to the levels of abstraction shown in Table 1.1 when discussing the modeling and simulation of digital systems:

**Table 1.1** LEVELS OF ABSTRACTION COMPARISON

Digital Design Engineer's Level	Examples of Modeled Entity	Approximate Equivalence to Computer Architect's Level
system level	pipelines, instruction decoders, and TLBs	microarchitecture
register transfer level (RTL)	registers, buses, multiplexers, and combinational logic	microarchitecture and logical implementation
gate level or logic level	library cells of AND, OR, etc. gates	logical implementation
transistor level	transistors in various process technologies	chip
layout level	geometries, transistor placements	chip

These levels of abstractions (sometimes also called stages) are shown in more detail in Figure 1.1. The differences between the terminology and tools each group employs are lessening because:

1. the size and complexity of the microprocessors and the impact of the implementation-related issues on the cost/performance of the overall system require these groups to work closer together, with the result that the boundaries between some of the adjacent stages in the design process are becoming increasingly blurred.
2. the decreasing cost of high-performance computer platforms has precipitated a movement of electronic design automation tools from specialized workstations to more widely available desktop platforms resulting in an increase in the power and scope of these tools.

Let us now take a look at the design and implementation process represented in Figure 1.1 in more detail.

### CONSTRAINTS

We assume that the instruction set architecture to be implemented has been previously defined. Thus the design and implementation process does not start with the design of the instruction set architecture but with the design of the microarchitecture, (i.e., the set of resources and methods used to implement the instruction set architecture). Recall from the earlier definition of the term microarchitecture that it includes the way in which these resources are organized as well as the design techniques used to reach the target cost and performance goals. Presumably, the instruction set architecture has been designed with an intimate knowledge of what the programs implemented actually do, (e.g., knowing what systems resources are used and how frequently, and knowing what memory bandwidth various classes of algorithms require). Knowing what programs do is absolutely essential to identifying what resources and instructions are required in the architecture.

It cannot be emphasized strongly enough the importance of understanding what the processor is doing when it executes specific benchmarks or workloads. This assumes that the benchmarks and workloads are truly representative of how the system will be used. This assumption, however, is one that is difficult to meet. Not only must suitable workloads be created, but representative traces from those workloads must also be developed. Using bad trace data in good performance models has been the cause of more than one processor's performance deficiencies. Moreover, deciding on what workloads are representative of current (or future) target markets can be extremely difficult.

*The detailed analysis of performance data and the subsequent use of that analysis in the design process is the cornerstone of many, if not most, architectural design decisions.*

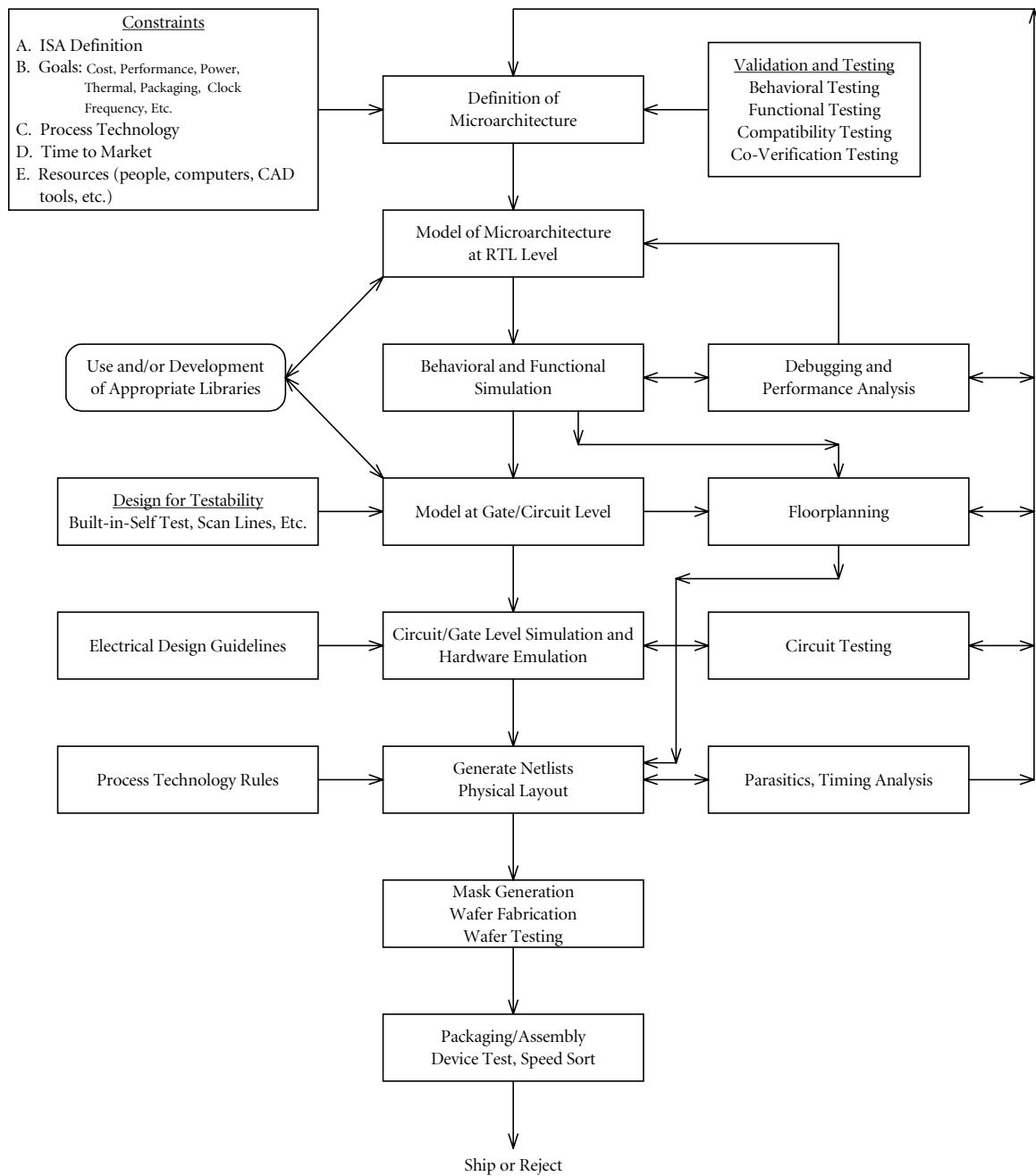


Figure 1.1 DESIGN PROCESS

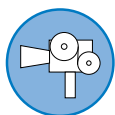
*The design of specific workloads and benchmarks is an art in itself.*

Even given these difficulties with developing good performance models and good trace data, it is safe to say that the detailed analysis of performance data and the subsequent use of that analysis is the cornerstone of many, if not most, central architectural design decisions. Some examples are data from address traces that yields information about such things as the frequency of occurrence of conditional branches, address modes, exception conditions, and interrupts.

The workloads and benchmarks and subsequent analysis may be targeted at a specific component of the processor (e.g., the integer unit or the floating-point unit), a group of components (e.g., the chunks of logic involved in the fetching, predication, and decoding of instructions), the entire processor itself, or the processor interconnected with one or more of its external system components (e.g., the chipset or off-chip cache). The design of specific workloads and benchmarks is an art in itself as often millions of instruction executions are needed to gain some understanding of the specific behavior you are attempting to study. Both the architect and the digital systems engineer need to know where the bottlenecks are in their respective models of the microprocessor in order to remove these bottlenecks.

Clearly, one of the most important constraints placed on the microarchitecture design team is the definition of the instruction set architecture that they need to support. For the K6 3D microarchitecture, it is the widely used x86 instruction set architecture. Indeed, one view of the K6 3D is that it is a high-performance CISC-on-RISC microprocessor. The CISC-component is the x86 instruction set architecture, while the underlying RISC-component is known as the Enhanced RISC86 microarchitecture. The most important implication of this constraint is that the K6 must be fully x86 binary code compatible, including code using the x86 MMX multimedia instruction set extensions. We will learn later that the K6 3D also provides an additional set of AMD-developed instruction set extensions called the AMD-3D instructions which support high-performance 3D graphics, audio, and physics-based modeling and simulation processing.

#### VIDEO ON CD-ROM



Amos Ben-Meir, Principal Designer of the K6 3D, addresses two questions of interest in this video clip: “What is needed to design state-of-the art microprocessors?” and “What were the design methods that enabled the K6 3D to be implemented in such a short period of time?”

Other constraints, both from a platform and performance point of view have to do with systems-related issues. Among the design goals for the K6 3D processor were explicit objectives for it to be Socket 7 compatible

and for it to fit within the electrical, power and thermal specifications, and the EMI envelope of the Intel Pentium processor. If it were successful in meeting these objectives, it could readily integrate into industry-standard Pentium-compatible motherboards, chipsets, power supplies, and thermal designs.

### COMPARATIVE ANALYSIS

#### Socket 7 Compatible

*Socket 7 compatible* means that the system bus interface is compatible with the industry-standard 64-bit Pentium P55C bus protocol and motherboard socket. One of the most important differences between the K6 and Intel's Pentium Pro and Pentium II processors is the way in which the microprocessor chips connect to the rest of the system and the system bus interface.

The K6, like the Intel Pentium and the Intel Pentium MMX (P55C), are "Socket 7 compatible" as defined above. However the Intel Pentium Pro and the Intel Pentium II (Klamath) use a different bus interface or protocol (called the P6 bus interface) and each uses a different physical motherboard connector. The Intel Pentium Pro uses a dual-cavity PGA packaging technology. The Intel Pentium II uses Intel's Single Edge Cartridge (SEC) technology and its Slot 1 and Slot 2 connectors. These issues and their design implications (e.g., the way in which the L2-Cache is interfaced to the system and the performance issues involved) are discussed in Chapter 4 and Chapter 6.

The analysis of the instruction set architecture is based on knowledge of typical application and system software, typical coding practice, behavior or nature of code generated by various compilers, and analysis of instruction and data reference traces of actual software. On this last point, which is one of the most significant inputs to the microarchitecture development process, the following elements are key:

1. the capability to capture traces of the execution of any and all code, (e.g., privileged OS code, device driver code, and application code).
2. the capability to use long traces—ten million to one billion instructions long.
3. the need to have an accurate and detailed trace-driven performance model of the design that can output a wide variety of performance-related statistics and that can be readily changed to explore alternative design options.

TESTING

During the design process, we undertake different types of testing—functional (i.e., behavioral) testing, structural testing, compatibility testing, and performance testing. There are basically three different levels of testing that need to be considered for each of these types. The testing of:

- 1. the design (i.e., verifying the design realizes the instruction set architecture without any errors).
- 2. the logical implementation (i.e., verifying the logical implementation realizes the design without any errors).
- 3. the chip (i.e., verifying the physical device realizes the logical implementation without any errors).

Functional (Behavioral) Testing

Functional (behavioral) testing is a method for verifying design correctness via simulation. A “block” or “component” representing a set of functions or system behavior is modeled. A simulator is constructed for the model. An input stimulus is applied to the block’s inputs and the block’s outputs are compared with the expected outputs. Trace buffers are used to chronicle the behavior of the block, permitting examination of the behavior for an arbitrary period of time prior to matching results. Such computer modeling of the design is begun at an early stage in the design process to verify design concepts in a top-down fashion. Functional testing is continued as more detail is added to the design and changes are made.

Initially the simulations are strictly at a highly abstract behavioral level and are performed on large blocks that model major functional areas of the design. As design implementation progresses, the original blocks are usually hierarchically decomposed into sub-blocks. The model is managed to track the design hierarchy and thus becomes more detailed over time. Testing is generally performed first on each new sub-block and then interactions between blocks are confirmed.

DEFINITION
<p>Test Vector</p> <p>Test vectors are the collection of values of input stimulus and expected output results for each sequential stage of simulation. The test vectors are intended to cover all inputs and outputs (pins entering or leaving) each block being tested. Test vectors for larger blocks don’t need to include I/Os of smaller blocks that do not appear at the boundary of the larger block.</p>



The development of test vectors is time-consuming and may be naively omitted from project scheduling or its extent may be underestimated. Writing the vectors generally requires a detailed knowledge of the blocks being tested. Often the logic designer of the blocks must write the vectors or at least define an initial template for others to follow. Insuring comprehensive testing coverage for large complex blocks is a specialized field, because exhaustive testing of all possible combinations of inputs may not be practical. Once confidence in the test vectors is obtained, vectors may be used in gate-level or other lower-level simulations, to verify bottom-up design correctness.

Functional simulators are generally restricted to behavioral modeling. When the gate-level design of a block is synthesized or manually designed, it too can be modeled using a gate-level simulation. Gate-level simulators generally can handle the simulation of sub-blocks that have behavioral descriptions. Thus some blocks may be at a gate-level, while others are still using a behavioral model. Often the focus is on verifying the gate-level design of a particular block, and blocks providing stimulus to or sampling outputs from the block of interest need only be behaviorally modeled.

### *Design for Testability*

A digital circuit is an implementation of the specification of a desired function, (i.e., it exhibits a desired behavior). A microprocessor is a collection of hundreds of thousands of such circuits. Given the complexity of the resulting microprocessor chips, the testing of them must be integral in their design from the onset. Incorporating testing technology into a design from its inception is often referred to as “design for testability” or DFT. Using DFT techniques invariably reduces costs and design time.

There have been a number of important DFT advances made at the logical implementation and chip level, such as boundary scan testing, full operational scan of internal state elements (e.g., flip-flops), built-in self-test (BIST), test vector generation, signature analysis, and observers. A number of these approaches required additional circuitry to be included in the design, solely for the testing function, reducing the amount of the total gates on the chip that are available to implement the microprocessor. Thus, another trade-off emerges regarding the distribution of gates between functionality and testing and the benefits from the use of DFT techniques.

A substantive treatment of testing—dealing with issues such as fault modeling, fault manifestation,<sup>2</sup> fault detection (controllability and

---

<sup>2</sup> For example, at the chip level one needs to conjecture how fabrication faults such as holes in insulating layers, bridging connections in metal layers, missed contacts, and poor control of etching will be manifested so that it is possible to test for them.

observability), test design, test data (collection and generation), and test coverage—is well beyond the scope of this book. Particularly since each of these topics needs to be discussed in the context of testing the design, the implementation, and the chip. However, a number of fundamental ideas are presented so the reader has an appreciation of the issues involved.

### Suggested Readings

#### Design for Testability

1. H. P. G. Vranken, M. F. Willeman, and R. C. van Wuijswinkel, “Design for Testability in Hardware-Software Systems,” *IEEE Design & Test of Computers*, Vol. 13, No. 3, Fall 1996, pp. 79-87.
2. K. P. Parker, *The Boundary Scan Handbook*, Kluwer Academic Publishers, 1992.
3. IEEE Standard 1149.1-1990, *Test Access Port and Boundary Scan Architecture*, January 1992 and the associated standard, 1149.1b, *Boundary Scan Description Language (BSDL)*, 1991.
4. M. Tegethoff, “IEEE Standard 1149.1: Where Are We? Where From Here?” *IEEE Design & Test of Computers*, Vol. 12, No. 2, Summer 1995, pp. 53-59.
5. B. T. Murray and J. P. Hayes, “Testing ICs: Getting to the Core of the Problem,” *Computer*, November 1996.
6. V. D. Agrawal and C. R. Kime and K. K. Saluja, “A Tutorial on Built-in Self-Test, Part 1: Principles,” *IEEE Design & Test of Computers*, Vol. 10., No. 1, May 1993, pp. 73-82.
7. V. D. Agrawal and C. R. Kime and K. K. Saluja, “A Tutorial on Built-in Self-Test, Part 2: Applications,” *IEEE Design & Test of Computers*, Vol. 10., No. 2, June 1993, pp. 69-77.



You can find the full text versions of *Part 1* and *Part 2* of the Agrawal, Kime, and Saluja tutorial as well as the Murray and Hayes article, on the companion CD-ROM.

### Compatibility Testing

The difficulty of achieving x86 architectural compatibility is generally underestimated. The business issues surrounding compatibility are generally equally underestimated. We will see that these business issues contribute directly to the technical difficulties in achieving compatibility.

From a business perspective, x86 compatibility is an absolute requirement. First of all, if it is perceived that the affected microprocessors are flawed merchandise, there is the real possibility of being held liable to recall or field-replace the flawed units. Equally important however, is that consumers simply will not accept a microprocessor if there is a lack of confidence that the microprocessor will not successfully run popular or legacy software. This latter problem is aggravated by the fact that the program-

mers of such software may have relied on unintended and undocumented behaviors, which historically have been prevalent in x86 microprocessors.

Undocumented or imprecisely documented behaviors are at the center of the difficulty in achieving compatibility. The user and programming manuals for x86 microprocessors have historically not compared with documents like the *IBM/370 Principles of Operation* manuals. The x86 manuals have generally been little more than abstractions that approximate the exact behavior of the microprocessor. Such x86 manuals cannot be considered as formal reference documents, written and edited with extreme care, and thereby suitable for design purposes.

Generally speaking, the first vendor to implement a commercially successful microprocessor will de facto determine its behavior for all implementations to follow. Even if part of that initial behavior includes unintended and undocumented artifacts, the earliest implementation defines the compatibility requirements for all other vendors. The first vendor does not need to reverse engineer the behavior, at least not until a subsequent generation's implementation. Even then, this vendor has knowledge of the exact logic underlying the original implementation, and can use that to retroactively deduce the precise behavior. The second implementer to market the instruction set architecture of the microprocessor does not have these luxuries.

Designers take great risks when making extensions to instruction sets. If the extensions are not useful, they will increase the cost of the processor without returning benefit. If the extensions are useful, but implemented inefficiently, they leave room for competitors to make improvements. The x86 architecture has had a number of extensions over its history. An early extension was the introduction of the 80286 instruction set, which extended the addressable memory from 1-MByte to 16-MBytes. This extension was not very successful. It failed to support compatibility with the underlying 8086 architecture in a way that could be used efficiently in practice. Some commercial software used the larger memory model, but two key operating systems at the time, Microsoft's Windows and IBM's OS/2, were unable to produce widely accepted versions using these extensions. Subsequent instruction set extensions embodied in the 80386 architecture provided an efficient 8086 emulation mode, and that enabled new operating systems to run 80386 and legacy 8086 code with equal efficiency.

One of the lessons to be learned from these examples is that instruction set extensions should be done in concert with the development of both the operating systems and important software applications that will use them. This suggests that computer architects and software designers should sit down together to determine what the extensions should do and how they should be implemented (see *hardware-software co-design* and *hardware-system co-design* on page 2). From a competitive point of view, this means that some proprietary information has to be shared across the

*Compatibility must not only deal with “official” or “publicly” documented features, but must also deal with “unofficial” and “undocumented” features and with behaviors in obscure cases.*

industry with some partners while kept secret from companies in the business of designing competitive microprocessors.

The ideal strategy is to publicly announce the instruction set extensions and the operating systems and applications software use of the extensions at the same time. In reality, the software announcements usually always follow the availability of the instruction set extensions, often by as much as a year. As an example, the x86 MMX instruction set extensions were announced in the spring of 1997. By early 1998, very little MMX software had been released, although a large number of the potential developers who could make use of the extensions had committed to use them in their applications. So, it looks promising that these extensions will indeed succeed.

On the other hand, the 80286 instruction set extensions were never truly successful. Yet they were used by enough software to force 80286 compatibility to a component of x86 designs through the 1990s and possibly beyond. Virtually all 80286 applications that were running in the late 1990s had long ago been converted to the 80386 memory model. But the possibility that some still exist for the 80286 memory model creates what amounts to a tax on the x86 architecture to support software that, in all probability, is no longer in use. Given this background into some of the problems involved in extending an instruction set, let us return to the issues regarding compatibility testing.

#### *compatibility test suites*

Because the published x86 documentation is not adequate for compatible design, design houses for x86 processors must make a large investment in time to develop extensive compatibility test suites. These suites are used to compare their parts against the de facto standard Intel x86 microprocessors. The compatibility suites exercise the functionality of individual instructions and inter-instruction interactions. Many tests are written that create result arrays in memory. Comparison of the result arrays for the microprocessor with the device under test, is more efficient than tests that require comparison of register values. Subsequent to such tests, compatibility testing includes extensive trials of major applications and operating systems. Comprehensive system tests are also performed using a wide range of peripherals. After much internal testing, outside compatibility laboratories are employed to give independent certification, and hence added credibility that the microprocessor is indeed compatible.

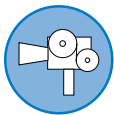
Clearly part of the difficulty in compatibility testing stems from x86 architectural complexity, particularly in the context of high-performance implementations, which act to further complicate compatibility testing. Some x86 architectural complexity issues are well known. These include the existence of complex instructions with variable-length instructions, non uniform instruction decoding, many address modes and inherently multiple cycle operations; a segmented addressing model, requiring con-

tinual effective and linear address calculations; precise interrupts; and IEEE compatible floating-point.

The foregoing well-known complexity issues are just the beginning. Beyond these are many more problems. Historically, x86 programmers have exploited the use of self-modifying code. Compatibility requires close monitoring for store-into-instruction stream events and extensive design efforts to ensure instruction cache coherency, which must extend over multiple levels of cache hierarchy, deep into the branch prediction, prefetch, and instruction decode logic. The x86 has multiple operating modes, including real-mode, virtual 86 mode, segmentation without paging, and segmentation with paging. Paging involves a 2-level translation with a TLB and also introduces additional user and supervisor-like protection features. Beyond addressing issues, segmentation has extensive protection model features, including the use of selectors and segment descriptors, call-gate transitions between protection levels, task-gate transitions between tasks, protected stack operations, and virtualized I/O. Miscellaneous x86 complexity that complicates compatibility testing stems from the existence of a “System Management” mode for facilitating system power management, instruction prefix operations, non uniform register operations (8-bit, 16-bit, and 32-bit operations), and the implicit instruction use of dedicated registers.

Because the K6 3D executes x86 instructions directly, a significant compatibility effort was undertaken from the very beginning of the project to ensure its x86 binary code compatibility. Its verification included all of the steps above plus validation for several major operating systems environments. Some of the validation steps were simplified because of tools and experience in building three earlier generations of x86 processors, but no steps were omitted..

#### VIDEO ON THE COMPANION CD-ROM



There is a video interview on the CD-ROM with Warren Stapleton, Leader of Model Development and Verification of the K6 3D and Anu Mitra, Verification Manager of the K6 3D in which they discuss the design process employed and the extensive role verification played in the project from its inception.

## DEFINING A PROCESSOR'S INSTRUCTION SET ARCHITECTURE

The genesis of a processor's instruction set architecture is often quite informal in nature—ranging from discussions where diagrams are drawn on backs of envelopes to talking one or two people into writing a report or a white paper discussing what might be done.

### Historical Comment and Suggested Reading

#### The von Neumann Machine

Two of the most significant papers in the history of computer architecture are related to what is called the “*von Neumann machine*.” They are:

1. John von Neumann, “The First Draft of a Report on the EDVAC,” Moore School of Electrical Engineering, University of Pennsylvania, June 30, 1945, republished in the *IEEE Annals of the History of Computing*, 1993.
2. Arthur W. Burks, Herman H. Goldstine, and John von Neumann, “Preliminary Discussion of the Logical Design of an Electronic Computing Instrument,” report prepared for U.S. Army Ordnance Dept., 1946, reprinted in *Datamation*, Vol. 8, No. 9, pp. 24-31, September 1962 (Part I) and *Datamation*, Vol. 8, No. 10, pp. 36-41, October, 1962 (Part II).

An interesting augmentation to the von Neumann paper cited above can be found in:

3. M. D. Godfrey and D. F. Henry, “The Computer as von Neumann Planned It,” *IEEE Annals of the History of Computing*, 1993.

An historical account of RISC technology within IBM, a good deal of which is relevant to topics discussed in this book, can be found in:

4. John Cocke and V. Markstein, “The Evolution of RISC Technology at IBM,” *IBM Journal of Research and Development*, January 1990.



Complete full-text versions of *each* of the above important, historical articles can be found on the companion CD-ROM. We encourage you to read each of them.

These modest beginnings often lead to forming a small group of people to undertake both analytical studies and simulations to explore the feasibility of some of the newer concepts inherent in the design, to define some aspects of the instruction set architecture with a bit more rigor, to examine the implications of a number of the design constraints. If the results of these efforts are promising, a project to design and implement the processor is typically launched, consistent with available resources. It should come as no surprise that a microarchitecture also evolves in a similar way to an instruction set architecture.

There are a wide variety of factors that influence the nature of the processor’s instruction set and the candidate microarchitectures for implementing the resultant instruction set architecture.

Some of the higher-level issues are:

1. target applications and operating systems.
2. target platform context—e.g., desktop, portable, or server.
3. target cost/performance level.
4. need to support legacy 16-bit and 32-bit code.

Some of the lower-level issues are:

1. target die size and cost.
2. target platform/system environment—e.g., external caches, bus speeds, and I/O speeds.

### Suggested Readings

#### Instruction Set Design

Consider what Michael Flynn on pp. 2-3 in *Computer Architecture, Pipelined and Parallel Processor Design*, Jones and Bartlett, 1995 has to say about instruction set design:

*“There are always trade-offs in instruction set design. A well-designed instruction set allows variability in implementation technology and is less sensitive to technology changes. As time goes by, even a well-designed set must undergo changes—additions to accommodate new functionality and perhaps a de-emphasis of older features. Thus, at any moment, a successful architecture includes an instruction set consisting of:*

- *A core of frequently used instructions.*
- *Some features extending or correcting limitations in the original design.*
- *Some instructions no longer expected to be used (either superseded or “out-of-vogue”), which remain for reasons of compatibility.”*

Also, consider Harold Stone’s related comments on p. 9 in *High Performance Computer Architecture*, 3rd Edition, Addison Wesley, 1993:

*“... The architect should measure the quality of the architecture across a number of applications that characterize how an architecture is to be used. The effectiveness may vary considerably from application to application, and such measurements should reveal where the architecture is truly beneficial to the user and where other approaches are superior.*

*A computer architecture might well have some minor but costly inherent flaws that escape the scrutiny of its designer. A different designer who can build essentially the same architecture with those flaws repaired can produce a more effective, and therefore more competitive, machine. Architects cannot hide inefficiency by arguing that hardware costs nothing.”*

Some typical design decisions that need to be made are:

1. will the design focus on employing multiple parallel pipelines or fewer, deeper pipelines—i.e., a maximum superscalar versus a maximum frequency approach.
2. how many instruction decodes/clock.
3. how many functional units and how deeply will they be pipelined.
4. how will functions be apportioned or assigned to each pipeline stage.
5. what type of caching.

The models employed by the hardware/system co-design teams need to be flexible enough and detailed enough to allow alternatives to this wide range of design issues to be evaluated.

### MODEL OF THE MICROPROCESSOR AT THE RTL LEVEL

There has been a rich history of the use of both formal and informal textual and graphical representations of computer architecture. In the following extended Historical Comment, Dr. Mario Barbacci of CMU shares some of that history with us.

#### HISTORICAL COMMENT

##### The Evolution of Architecture Description Languages by Dr. Mario Barbacci CMU

Designers and students of computer architectures have always made use of graphical and textual conventions to describe computer architectures. Early notations varied in their degree of formality and descriptive power and were not in widespread use (for reasons that will become apparent later). A significant event in the evolution of architecture description languages took place in 1964 with the publication of a formal description of the recently announced IBM SYSTEM/360 [5]. This description provided a definition for a computer architecture namely, the behavior and the state visible to the programmer:

*“This paper presents a precise formal description of a complete computer system, the IBM SYSTEM/360. The description is functional: it describes the behavior of the machine as seen by the programmer, irrespective of any particular physical implementation, and expressly specifies the state of every register or facility accessible to the programmer for every moment of system operation at which this information is actually available.”*

continued on next page...



## HISTORICAL COMMENT (CONT.)

## The Evolution of Architecture Description Languages

The description of SYSTEM/360 consisted of a set of programs in APL [7] organized in two sections, the central processing system (nine programs) and the input/output (five programs). The programs were complemented with auxiliary tables that provided, for example, definitions of variables and locations (line numbers) in the programs where the variables are read or written. The legendary terseness of the language makes a study of the description a slow process at best, and the process is not helped by the naming conventions (e.g., “For brevity, single characters are used for all variables except for those which occur infrequently, such as the panel switches occurring in CP, the control panel program.”) Nevertheless, this is a milestone in the evolution of Architecture Description Languages and must read for any serious student of the subject.

As integrated circuits increased in density during the 1960’s, new computers began to proliferate and it was possible, for the first time, to collect, study, and classify these artifacts, just like one could study plants or animals or rocks. The publication of [3] was the first attempt to organize computer structures into levels, represented with uniform notations:

*“The structures that we call computer systems continue to grow in complexity, in size, and in diversity. This book is linked firmly to the nature of this growth. The book is about the upper levels of computer structure: about instruction sets, which define a computer system at the programming level; and about organizations of processors, memories, switches, input-output devices, controllers, and communication links, which provide the ultimate functioning system. These levels are just emerging into well-defined system levels, with developed symbolic techniques of analysis and synthesis and accumulated engineering know-how, all expressed in a crystallized representation.”*

The book provided a large collection of detailed examples illustrating actual computers. The authors felt that a sufficiently large number of computers had been designed over the previous 25 years that it was possible to systematize the space of computer designs, to provide a framework for the study of computers as a class of artifacts and not as isolated, independent inventions.

The framework consisted of a hierarchy of levels of descriptions complemented by two notations, one for instruction sets, called ISP, and the other for configurations of major components, called PMS.

According to [3] a digital system can be described at many different levels of detail in order to depict structural or behavioral aspects. Thus a system can be described at the gate level as a network of logic gates and flip-flops whose behavior is specified by timing diagrams, Boolean equations, or truth tables. While a complete digital computer could be described at this level, the amount of information to be conveyed would be too extensive for a human designer to comprehend, and higher levels are introduced to abstract details: Combinatorial and Sequential Register Transfer levels.

The existence of digital components capable of interpreting instructions stored in memory (i.e., instruction set processors) motivated Bell and Newell to introduce the programming level of description.

continued on next page...

## HISTORICAL COMMENT (CONT.)

## The Evolution of Architecture Description Languages

At the programming level, the basic components are the interpretation cycle, the machine instructions, and operations (all of which are defined as register transfer level operations). The programming level arises from the need to describe the behavior rather than the structure of processors—in particular the behavior as seen by the programmers of the machine (i.e., the goal of Falkoff et al, in the SYSTEM/360 description)

The system levels correspond closely to the technology available for analysis and synthesis of computer systems. During the 1940's and 1950's computer architectures were simple, often linked to one unique implementation, and the need for a description language were satisfied by logic diagrams and Boolean equations. The situation changed with the introduction of SYSTEM/360 because it consisted of a large family of implementation of the same instruction set—the architecture had to be abstracted from the implementation, thus the need for a different notation. In 1971, Bell and Newell characterized the situation thus,

*“Each of these levels exists in fact, precisely to the extent that a technology has become well developed. Thus both the circuit level and the lower half of the logic level (combinatorial and sequential circuits) are highly polished technologies. They are what one learns today, if one wants to become a computer engineer. Textbooks exist, courses are taught, and there is a flourishing, cumulative technical literature. As we progress up the systems levels, matters become progressively worse. The register-transfer level is not yet well established, although there is considerable current activity and the next few years may see its universal establishment.”*

No such consensus was apparent at the programming or system levels although the increased complexity of computer systems was increasing the importance of these higher levels.

One decade later technology advances had led to an explosion in the number of computer types, with a large number of instruction sets and data types, as reflected in a revised and expanded version of [3]. By the time [8] was published, the programming level of description was firmly established and the leading notation, ISPS, a formally defined programming language, based on the original ISP notation had been used in a variety of analysis and synthesis applications. In ISPS [1] a processor is described by declarations of carriers and procedures specifying the behavior of the system:

1. information carriers - registers and memories used to store programs, data, and other state information.
2. instruction set - procedures describing the behavior of the processor instructions.
3. addressing modes - procedures describing the operand and instruction fetch and store operations.
4. interpretation cycle - typically, the main procedure of an ISP description. It defines the fetch, decode, and execute sequence of a digital processor.

continued on the next page....

## HISTORICAL COMMENT (CONT.)

## The Evolution of Architecture Description Languages

The PMS notation remained a graphical language for describing uniprocessor structures but never evolved to the point of being formalized and implemented as a serious design tool.

The story was very different at the lower levels of detail, where a number of notations known collectively as “Hardware Description Languages” continued to be developed and used. Hardware Description Languages (HDL) are notations and languages that facilitate the documentation, design, simulation, and manufacturing of digital computer systems [2]. Most of these languages (see suggested reading material) were used mostly in research and academic environments, as input notations for experimental simulation, analysis, or synthesis tools. In the industrial world, however, additional requirements had to be considered, namely the need to create, modify and support many design and manufacturing details across manufacturers and throughout the product’s lifecycle.

To address these requirements, several industry-supported efforts have led to standard formats to represent product data in a standard format. Two of these efforts were the Very High Speed Integrated Circuits Hardware Description Languages (VHDL) [6] and the Electronic Design Interchange Format (EDIF). VHDL and EDIF became standards in 1987 (IEEE Standard, 1076 and EIA RS44 respectively) [4].

In VHDL each hardware entity has an interface and a body or architecture. The interface descriptions consist of input and output ports and various attributes associated with the interface, such as pin names, timing constraints, etc. The body describes the function or the structure of the design. The body may be written as an algorithm or as a combination of algorithms and real hardware representations (e.g., gates, arithmetic-logic units) or made up totally as a structure of real hardware representations.

EDIF provides a hierarchical syntax for data necessary for chip and printed circuit board fabrication. Note that EDIF is a format, not a language. EDIF’s primary application is as a means of transferring design data from the design environment to the fabrication environment. The format provides for libraries, cells, views, interfaces, and information on the content within each cell. Test data, mask layout data, physical layout data, connectivity data, and simulation data can be represented in EDIF.

These various standards attempt to answer the needs of the various product life-cycle activities. However, the development of these standards have not been coordinated, and users still need a thorough understanding of the objectives and uses of each standard. The technology of HDLs has not matured to the point that a standard language or format can satisfy the wide diversity of product description requirements, at least for the foreseeable future.

By the early 80’s, the dimensions of an emerging level, the network level, were noticed. It had the character of a different level because the performance of a network was far more dependent on operating systems, network topology, protocols, bandwidth, than on the instruction sets of individual processors.

At the time of this writing, 15 years after [8], an explosion on the number of network types, protocols, and communications has taken place. Millions of personal computer users connected to intranets and the Internet have created an enormous demand for new technology. The technology is changing so rapidly that it will be a while before “the level” begins to settle down and “the notation” emerges.

[continued on the next page...](#)

## HISTORICAL COMMENT (CONT.)

## References: The Evolution of Architecture Description Languages

1. [Barbacci 81] Mario R. Barbacci, "Instruction Set Processor Specifications (ISPS): The Notation and its Applications," *IEEE Transactions on Computers*, C-30, 1, (January), 1981.
2. [Barbacci 93] Mario R. Barbacci, Ron Waxman, "Hardware Description Languages," in *Encyclopedia of Computer Science*, 3rd Edition, Anthony Ralston and Edwin D. Reilly (Eds.), Van Nostrand Reinhold, 1993.
3. [Bell and Newell 71] C. Gordon Bell and Allen Newell, *Computer Structures: Readings and Examples*, McGraw-Hill Book Company, 1971.
4. [EDF 87] *EIA/EDIF/IS-44 Specification, Electronic Design Interchange Format*, Version 2.0.0, May 1987.
5. [Falkoff 1964] A.D. Falkoff, K.E. Iverson, E.H. Sussenguth, "A formal description of SYSTEM/360," *IBM Systems Journal*, Vol. 3, No. 3, 1964.
6. [IEEE 87] IEEE, "VHDL Language Reference Manual," Standard 1076, December 1987.
7. [Iverson 62] Kenneth E. Iverson, *A Programming Language*, Wiley, 1962.
8. [Siewiorek 82], Siewiorek, Daniel P., Bell, C. Gordon and Newell, Allen, *Computer Structures: Principles and Examples*, McGraw-Hill, 1982.

## Additional Readings: The Evolution of Architecture Description Languages

1. Mario Barbacci, "A Comparison of Register Transfer Languages for Describing Computers and Digital Systems," *IEEE Transactions on Computers*, Vol C-24, No. 2, February 1975, pp. 137-150.
2. Robert Piloty, Mario Barbacci, Dominique Borriane, Donald Dietmeyer, Fredrick Hill, Pat Skelly, "CONLAN Report," *Lecture Notes in Computer Science*, 151, Springer-Verlag.
3. Yaohan Chu (Guest Editor), special issue on Computer Hardware Description Languages, *Computer*, Vol. 7 No. 12, December 1974, pp. 18-22.
4. Stephen Y.S. Su (Guest Editor), special issue on Computer Hardware Description Languages, *Computer*, Vol. 10 No. 6, June 1977, pp. 10-13.
5. Mario Barbacci and Takao Uehara (Guest Editors), special issue on Computer Hardware Description languages, *Computer*, Vol. 18 No. 2, February 1985, pp. 6-8.
6. Allen Dewey (Guest Editor), "VHDL and Next-Generation Design Automation," Guest Editor's Introduction to *IEEE Design and Test of Computers*, Vol. 9, No. 2, June 1992, special issue on VHDL.
7. "Three Decades of HDLs," Collections of short notes by various authors: Part 1, "CDL Through TI-HDL," *IEEE Design and Test of Computers*, Vol. 9, No. 2, June 1992; Part 2, "CONLAN through Verilog," *IEEE Design and Test of Computers*, Vol. 9, No. 3, September 1992.

Among the important issues to consider when modeling a microprocessor are: what the model will be used for, the type of model required and its representation, the ease of use of the model, and cost. The specific level of abstraction is selected consistent with the goals of the modeling effort. It is not only important to decide how the logic and circuit designs of the actual microprocessor are going to be represented but also how the “simulation environment” within which the microprocessor will be “exercised” will itself be designed and implemented. This is the part of the system that generates stimulus for the model and checks that it is behaving properly. For a microprocessor, the simulation environment must be able to display and check the state of various conditions within the processor and it must accommodate such things as distributing clock signals and modeling of the main memory and system bus architecture and protocols. A software simulator may be coupled to various hardware emulators to form a hardware/software co-simulation environment.

Modeling is a compromise between accuracy (the level of representation, the details described, and the precision with which they are described) and speed. An accurate model of a microprocessor at the physical level would have to model physical device characteristics. A simulation of such a model would be so slow that it would be completely useless for functional compatibility tests. Moreover, there needs to be a tool that can be used to conveniently describe a microprocessor’s complex behavior and structure. Specialized high-level programming languages have evolved to meet this need.

### Suggested Readings

#### Full-Custom and Semi-Custom Design

From page 3 of the book by Ulrich Golze, (with Peter Blinzer, Elmar Cochlovius, Michael Schafers, and Klaus-Peter Wachsmann), *VLSI Chip Design with the Hardware Description Language VERILOG*, Springer, 1996, which was mentioned earlier, we have:

*“In full-custom design, all details of the circuit had now to be designed, the transistors had to be dimensioned and composed to meaningful geometrical layouts which were afterward verified by an analog simulator. A layout is a true-to-scale template for the structures to be produced, however strongly enlarged. ... Around the middle of the 1980s, semi-custom design style became the workhorse of VLSI design. With the user interface again moving upward, the semi-custom design employs optimized library cells, typically logic gates, adders, etc., composes them to logic wiring diagrams (gate netlists, schematics) and simulates them logically. The transformation into a geometric layout is achieved by efficient placement and layout programs. The designer, in general, is not involved with single transistors, he often does not even know the internal structure of the library cells used.”*

*There is substantial debate in the industry about the adequacy of current hardware description languages to handle system-level designs, behavioral and logic synthesis, simulation, and formal verification. See for example, “DAC 97 Panel: Next-Generation HDLs,” by L. Lavango and N. Collins, IEEE Design & Test of Computers, July-September, 1997, Vol. 13, No. 3, p. 7.*

Programming languages, extended to include notions of time, parallelism and synchronization, and architectural and hardware data structure extensions, have emerged to dominate in semi-custom design. Such languages are called hardware description languages or HDLs. A model, which is the definition of the microprocessor implemented as a program in the HDL, can be simulated by compiling and executing it. Since the program can be written to describe the microprocessor at any level of abstraction, the simulation will be of the microprocessor at that particular level.

## GOLDEN REPRESENTATION

The HDL description becomes the “*golden representation*” of the microprocessor. Changes are made to the golden representation, and all other representations must change to match the golden representation. The flow of change is one way, starting with the golden representation. When this representation becomes detailed enough, it can be transformed by a process called *logic synthesis* to be input to the layout and placement stage of the design process.

Independent descriptions of the microprocessor can be made for each level of abstraction. Beginning with its behavioral description and employing step-wise refinement to model more and more of the underlying structure, one could extend the description down to the gate level. These lower-level design representations must be “cross-verified” against the *golden representation* to ensure they are functionally equivalent. Cross-verification is necessary mainly when the lower-level representations are generated in part or totally by hand. If they are completely machine-generated, cross-verification is less necessary unless the software tools that generate the lower-level representations are suspect.

## ARTICLES ON CD-ROM



A full text version of “Introduction to High-Level Synthesis,” by Daniel D. Gajski and Loganath Ramachandran, *IEEE Design & Test of Computers*, Winter 1994 can be found on the companion CD-ROM. Two related articles, “Specification and Design of Embedded Hardware-Software Systems,” by Daniel D. Gajski and Frank Vahid, *IEEE Design & Test of Computers*, Spring 1995 and “Introduction to the Scheduling Problem,” by Robert A. Walker and Samit Chaudhuri, *IEEE Design & Test*, Summer, 1995 are also on the CD-ROM.

A growing number of digital system designs are currently represented in one of two popular hardware description languages, Verilog or VHDL.

Selecting one over the other is seen by some as mostly a matter of religion. However there are some concrete reasons why one might be better suited to a particular design than the other. If Verilog was to be compared to C, then it might be reasonable to say that VHDL is like C++. VHDL is a much more complex language that allows practically all aspects of its definition to be redefined. In most cases VHDL is used with standard packages that define the operators to work in the most logical and expected fashion. A typical complaint is that VHDL is difficult to learn and is a verbose language. Some authors state that one advantage of VHDL is that its simulation semantics are reasonably well defined; thus, most vendors' simulators for VHDL behave exactly the same. However, we caution that the same can be said about Verilog simulators noting that they are based on a de facto definition as evidenced by the Verilog simulator of Cadence Design Systems, Inc. VHDL is the accepted HDL used for military applications and also is often used for describing extensible libraries. On the other hand, Verilog has gained widespread acceptance because it is easy to learn and many consider it more practical. Many engineers say they were able to use Verilog in just a few days. The Verilog language looks familiar because it is like a combination of Pascal and C mixed together with additional constructs to represent hardware design and simulation semantics.

## BEHAVIORAL (FUNCTIONAL) SIMULATION

Unfortunately, hardware designs, like software designs, usually have some errors (bugs). For a complex design like a microprocessor, there are likely to be thousands of bugs that have to be identified and fixed throughout the design process. Obviously any speedup in the process of identifying bugs will shorten the entire design cycle. Simulators typically provide mechanisms for displaying selected design variables, either every cycle or when they change value. This information can be used by other tools that allow the data to be displayed in a more convenient and meaningful manner as either waveforms or state dumps. There are two basic styles of writing simulators to study the behavior of a particular model—cycle-based and event-driven. Cycle-based simulation corresponds to examining model variables “every cycle” whereas event-driven simulation corresponds to examining model variables “when they change.”

One of the problems with describing and simulating hardware is that hardware can execute a number of things in parallel. However, these items are resolved at different stages of execution in a simulator. For example, a 3-input AND gate may have only two inputs ready when the simulator has reached the stage of executing the model that represents the AND gate. This means that the evaluation of the AND gate will not produce the final result and will have to be evaluated again once the 3<sup>rd</sup> input is ready.

*cycle-based simulation*



## DESIGN NOTE AND SUGGESTED READING

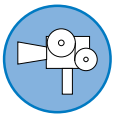
## Using C++ as an HDL

The K6 family of microprocessors requires a rather large systems-based simulation environment. The team decided that contemporary hardware description languages were not good for the general purpose programming that would be required. They chose to model the microarchitecture using C++ and to then take advantage of this general purpose programming language to represent the entire model (i.e., the design plus the simulation environment). There were a number of advantages to using C++:

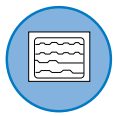
1. *object-oriented nature*: The object-oriented nature of the C++ programming language allowed the team to define an elegant way to represent the logic of the design, and was flexible enough to generate complex models of expected behavior.
2. *ability to override operators*: One main advantage that HDLs have over programming languages is their ability to manipulate individual bits of data. To overcome this limitation the operator() function was overridden to represent bit and part selection.
3. *expressiveness*: It was possible to write equations in the form of “signal1(3,2) = signal2(1,0)” or “signal1 = signal 2 & signal 3” which are just as convenient as using one of the hardware description languages.
4. *speed*: The K6 3D simulator needed to be very fast. This was another key reason for choosing a compiled language. Minimizing the amount of time to debug a problem, making a modification, recompiling, and verifying that the problem is indeed fixed was key to the success of the project.
5. *debugging aids*: design assertions and instrumentation code were easily included in the simulator.
6. *execution in desktop PCs*: ability to run simulations of large systems on desktop PCs and workstations with relatively modest amounts of main memory as well as on servers.
7. *quick development loop*: relatively fast simulate-debug-change-compile loop compared to typical HDL environments.



For a discussion of related issues related to the use of “traditional” programming languages as HDLs, see the article, “Using a Programming Language for Digital System Design,” by R. K. Gupta and S. Y. Liao, *IEEE Design & Test of Computers*, April-June 1997 on the book’s companion CD-ROM.



In this video clip, Amos Ben-Meir, Principal Designer of the K6 3D, addresses the question, “Was software simulation used in the verification of the K6 3D?” and Warren Stapleton, the K6 3D’s Leader of Model Development and Verification, addresses the question, “Why did the K6 3D team choose to use its own proprietary modeling and simulation tools?”



Given the importance simulation plays in the design and implementation of microprocessors, we have included three simulators on the CD-ROM. In Chapter 3, we give pseudo-RTL descriptions that describe various chunks of microarchitecture logic and recommend the reader simulated them on these simulators.



In *cycle-based* simulation many things (e.g., register loads and signal transitions) can occur in parallel. These actions are “flattened and re-ordered” in order to create the effect of all events occurring in their proper order. Cycle-based simulation eliminates the need to re-evaluate sections of code due to having unresolved terms. Again, this is done by flattening and re-ordering so that everything computes and resolves in one pass through the simulator. In a cycle-based simulation, there are inputs defined and expected outputs derived for every cycle. Cycle-based simulators have difficulty in dealing with multiple clock domains and generally cannot deal with delay simulation at all.

*Event-driven* simulators take the approach of re-evaluating models every time there is an event on one of the inputs or variables of the model. It is easier to handle multiple clocking domains with this type of simulator, as well as delay simulation. The disadvantage is that this type of simulator is generally slower than cycle-based simulators. *event-driven*

*Cycle-accurate model* refers to a model describing the behavior of a functional unit down to the cycle level. This means that the functional unit’s behavior is modeled accurately enough so that the model’s pins behave identically to the “real thing” on a cycle-by-cycle basis. This allows generating test vectors from the model so that the “real thing” can be verified with these. An example of a model that may not be cycle-accurate is a performance model. A performance model may only approximate the cycle behavior since its purpose is to gauge the performance of the functional unit within some range of accuracy. *cycle-accurate*

As mentioned earlier, one of the most important uses of simulators is to verify the behavior of and to debug the system being modeled.

## HISTORICAL COMMENT

### Verification Technology

From P. Shepherd, *Integrated Circuit Design, Fabrication, and Test*, McGraw-Hill, 1996, pp. 120-121, we learn:

*“Before software tools were developed, the verification of a particular circuit design could only be achieved by constructing a prototype circuit. While the design could use standard digital and analogue techniques to design the circuit on paper, it was almost impossible to determine whether the circuit would perform as expected to in practice. .... When built, the circuit would be thoroughly tested and design modifications made on the basis of these tests. The next version of the process was then constructed and the process repeated. Such a technique was very time-consuming and expensive. .... Redesign and rework of the mask set added further delay and expense to the product development.”*

Debugging a design usually differs significantly from debugging a program. When writing programs, source level debug tools that allow the code to be single stepped while examining individual variables are key to

boosting debug productivity. Although this capability is available in some simulator environments it is not used that often. A good environment for debugging a design should provide fast access to all of the key signal values, in a logically formatted display, after each cycle has completed. Also, many simulators keep a history of selected variables so that one can step backwards in time from an erroneous state to determine the events that forced the system into that state. Keeping in mind the simplicity of most hardware description languages, the bugs are more likely to be conceptual in nature rather than simple coding errors.

### **MODEL AT THE GATE AND CIRCUIT LEVELS**


Gate-level models are models that describe the function of a particular block or chip at the gate level. A gate is generally a basic building block of the design that implements a simple Boolean function. The gates are then connected together to create a more complex functional unit. Circuit-level models are typically models that go one level below the gates (i.e., to the transistor level).

### **GATE AND CIRCUIT LEVEL SIMULATION AND HARDWARE EMULATION**

The simulation models for the basic gates are generally part of a gate library that contains multiple representations of these gates (in chip design, this is typically layout, timing models, ATPG models, transistor level netlist/schematic and simulation models). When doing gate-level simulation, the functional unit being simulated must already have a netlist with a gate implementation. This netlist can then be simulated with one of the multiple commercial simulators or proprietary in-house simulators.

When doing functional simulation with circuit-level models (transistor-level models), there is usually an abstraction phase where the transistors are translated into gates and then these gates are simulated. This is done to improve the speed of transistor level simulations. There are simulators that are able to simulate at the transistor level, treating the transistor as a 3-node switch and computing the values on each node. This is typically very slow and requires much compute time. In addition, many transistor topologies that have analog behavior do not lend themselves well to switch-level simulation—i.e., it is difficult for the simulator to resolve what the circuit is doing. These analog sections typically require creation of simple RTL or gate models to describe their logic behavior.

## ARTICLES ON CD-ROM

	<p>Full text versions of two articles that deal with the topics discussed in this subsection: see “Circuit Techniques in a 266-MHz MMX-Enabled Processor,” by Donald A. Draper, Matt Crowley, John Holst, Greg Favor, Albrecht Schoy, Jeff Trull, Amos Ben-Meir, Rajesh Khanna, Dennie Wendell, Ravi Krishna, Joe Nolan, Dhiraj Mallick, Hamid Partovi, Mark Roberts, Mark Johnson, and Thomas Lee, <i>IEEE Journal of Solid-State Circuits</i>, November 1997 and “An x86 Microprocessor with Multimedia Extensions,” by Donald A. Draper, Matthew P. Crowley, John Holst, Greg Favor, Albrecht Schoy, Amos Ben-Meir, Jeff Trull, Raj Khanna, Dennie Wendell, Ravi Krishna, Joe Nolan, Hamid Partovi, Mark Johnson, Tom Lee, Dhiraj Mallick, Gene Frydel, Anderson Vuong, Stanley Yu, Reading Maley, and Bruce Kaufmann 1997 <i>ISSCC Digest of Technical Papers</i>. You can find the full text versions of both of the above articles on the companion CD-ROM.</p>
---	---

## DEFINITIONS

## Co-simulation and Co-verification

Co-simulation and co-verification are terms generally used to describe a situation where two different types of models (gates and RTL for example) are simulated together. Both models receive identical stimulus, then their outputs are compared on every cycle to guarantee that the two models behave identically. The goal of this type of simulation is to prove functional equivalence between two representations of a design.

Hardware emulation is the process of taking a functional unit netlist or a full-chip netlist and building it in some form of real hardware, such as FPGAs and memories. Then that hardware can be plugged into a real system for testing (though at fairly low frequencies, 100's of KHz or a few number of MHz). The hardware can also be used as a very fast simulator provided there is an environment that allows passing stimulus to the emulator and receiving the outputs from the emulator and then comparing the outputs with expected results to check for correctness.

*hardware emulation*

One of the goals of emulation is to provide a way to run the design in a real-world environment. Usually this means being able to plug the emulator into the actual socket that the chip will plug into and using the complete final product as if it had a real chip installed. This goal can be achieved in several different ways. One approach is simply to build an extremely fast simulator using parallel processors to get the required speed. The more traditional approach is to use a large number of interconnected FPGAs (Field Programmable Gate Arrays) and program them with a version of the gate-level netlist of the design.

#### DESIGN NOTE

##### Using Emulation in the Design and Testing of the K6 Family of Microprocessors

Hardware emulation was one of the keys to the success of the K6 project. The team programmed interconnected FPGAs with a version of the gate-level netlist of the design. They used the commercially available Quickturn emulation system to do this. What made the K6 emulation unique was that the team was able to get the gate-level design working quite some time before the initial fabrication of the chip. Prior to committing the design to manufacturing they were able to initialize and run all of the available x86 operating systems and run a significant number of standard applications, thus proving their design and its compatibility very early in the design cycle.

In addition to finding a handful of obscure bugs that probably would not have been found with conventional simulation, they were able to verify the built-in engineering debug features of the chip that would have required too many cycles to verify with the C++ model.

There were also some intangible benefits of emulation: The emulation lab provided experience for a multi-disciplined team that included BIOS developers, system experts, and chip designers. This experience was valuable when silicon returned from the fabrication facility. When the team booted the Windows 95 Operating System for the first time in the emulation laboratory the event gave an additional boost to the morale of the entire development team, which helped them get through the last few months before chip tapeout.

#### GENERATE NETLISTS AND PHYSICAL LAYOUT

The gates described at the behavioral level are selected from a library of cells which have been created for optimal realization of the logical functions in a particular process. These cells consist of nands, nors, inverters, flip-flops, latches, multiplexers, and other specialized cells. The first task is defining the physical and electrical characteristics of these cells.

The dimensions and pin placements of the cell needs to be expressed in multiples of the metal pitch, which is defined by the process capabilities. The routing pickup points likewise are determined by the metal pitch, and as many as possible should be placed in the cell to optimize the routing density. The power and ground supplies are designed to minimize the

resistive drop to the transistors and to avoid creating excessively high current densities which could lead to early reliability failures due to electromigration. To avoid performance loss due to resistance in the diffusions of the transistors, many metal-to-diffusion contacts need to be used. This is still true even with modern diffusions which use a silicided layer for reducing resistance. Another characteristic of cell design is that the gate resistance combined with the gate capacitance causes a delay of the input signal from the pickup point to the other end of the gate represented by the transistor width. This delay increases as the square of the length of the gate over thin gate oxide. Furthermore, it is a characteristic of the silicidation process that narrower gates form the silicide poorly, resulting in a higher effective sheet resistance, which further aggravates the problem. For this reason, it is necessary to limit the maximum length of gate poly which can be done by using smaller transistors with many legs, by strapping out the poly, or by using pickup points in the center of the gate, between the n-channel and p-channel transistor blocks. All these things impair the routeability of the cell, which needs to be balanced against the performance loss of the poly resistance.

Next, the sizes of the n-channel and p-channel transistor blocks need to be defined. The optimum ratio for speed is in the range of 1.4 to 1.8, for p-channel width relative to n-channel. The switching point will be slightly less than  $V_{DD}/2$ . This switching point should be the same for all gates, whether nands, nors, or other gates. This means that not all the available transistor width in the cell will be used, but this makes timing simulation using the static timing analyzer more accurate. Cells of different drive strengths are required for optimum timing, but for drive strengths beyond three or four times the minimum, buffer cells should be used. It is also possible to have all the cells have versions optimized for both rising and falling edges, although this will lead to an extensive proliferation of the number of cells.

The design of the flip-flops is optimized for speed and other characteristics, such as minimum hold time and setup time. Dynamic logic can be incorporated into the flip-flops to achieve a performance increase. Another specialized design is to put delay cells into the clock input path of the flip-flop to achieve cycle-stealing or delay transfer between critical paths. Similar strategies can be used with level-sensitive transparent latches. To facilitate testing and debugging, the flip-flops need to have scan designed in. This means adding extra logic and routing scan clocks and the scan data in and scan data out, all of which add cost and complexity. But, there is probably no other way to achieve a high level of fault coverage or to be able to debug the chip when there is a logical bug or a pattern sensitivity.

After building the cells, it is very important to characterize them for timing. The first requirement is to determine the maximum delays for the

*characterizing cell timing*

frequency-limiting maximum-delay paths. This is done by simulating the cells with the typical process at nominal voltage and worst-case temperature. The propagation delay needs to be determined for each input to each output path or arc. The delay is simulated as a function of the output loading and the input transition time, and is commonly represented as a matrix from which the actual timing delay is interpolated or extrapolated. In the case of state-dependent delays for cells such as exclusive-or gates, the delay of one arc is dependent on the logical state of the other input. This cannot be known to a static timing analyzer, so the worst-case delay needs to be selected. In the case where there are simultaneously switching inputs, such as from a bus, the delay time is again affected. For example, in a nand gate, if all the inputs switch from low to high within a small specified time of each other, the output delay is significantly increased, as compared to the case in which only one input switches while the other inputs stay high.

#### *hold-time requirements*

It is also necessary to guarantee that min-path, or hold time requirements, are met. The simulation conditions use the fast process corner at high voltage and low temperature. In the case of state-dependent delays, the shortest delay needs to be selected. Similarly for simultaneously switching inputs, the condition for the fastest output needs to be considered. For example, in the case of the nand again, if all of the inputs switch at the same time from high to low, the load is pulled up by all the p-channel transistors in the nand gate, not just one of them.

The above analysis becomes much more complicated for complex gates such as and-or-invert (AOI) cells and all combinations of input timing need to be exhaustively simulated. Similarly, the logical function needs to be verified in comparison to the Verilog or behavioral model by running an exhaustive combination of all possible inputs. This is especially important for complex gates such as tristate drivers and for AOI gates.

### *Full-Custom Macro Blocks*

The other major category of physical development involves the full-custom designed macro blocks, such as cache memories, register files, input/output drivers, phase-locked loop and clock distribution systems. As with the standard cells, these blocks need to have their timing characteristics and logical function thoroughly specified and verified. The timing is determined after design and layout of the macro blocks by extracting the capacitance of the nodes and resimulating. The functional verification is accomplished by simulating the circuit and comparing the outputs, vector by vector, with the behavioral model.

The design of arrayed structures, such as memories, requires speed to be balanced with the margin of bit-line signal at the sense amps. This is a very carefully balanced race condition, with the sense-amp strobe arriving not too early before the signal has developed on the bit lines and not too

late, losing performance. Usual signal requirements at the sense amp are 150 mV at typical timing condition. To guarantee robust design, every race in the circuit needs to be simulated extensively to ensure adequate margin.

Testability is addressed by means of a built-in self test mechanism. Testing a large array solely by means of external parallel vectors or by scan is prohibitively expensive in test time, so an internal engine to generate test vectors is included. It should completely test the array for most of the known failure mechanisms in memories; e.g., such a built-in self-test pattern is the 13N algorithm, which tests the complete array in thirteen passes.

Due to the large area or number of transistors in the design, the yield of regular, arrayed structures (such as memories) can be increased by including redundant rows or columns or both. The regular array is tested and if a defect is found, the defective row is deselected and a spare row is switched in instead.

Among other design features for arrayed structures, it is important to include the ability to bit-map the array for debug purposes, using scan or by using parallel vectors. Some arrayed structures use single-ended bit lines, and in these cases, bit-line coupling needs to be considered and designed. Some designs have used cascade sense amps, but these are too noise sensitive and should be avoided.

Chip Input/Output designs need to match the requirements of the external system at the same time as interfacing to the internal circuitry. Since I/O voltages are often different from the supply voltage used for the rest of the chip, reliable interfaces need to be designed between the two voltage domains, often requiring level shifting. The I/O voltage, being usually higher, imposes special design requirements involving the gate oxide. Excessively high voltage across the gate oxide is a wearout mechanism and a reliability hazard which needs to be addressed.

A special macro block is the phase locked loop, or PLL, which is used for synchronizing the internal clock signal with the external and for multiplying up the external or system bus clock frequency. The requirements are for stability and frequency tracking over a wide range of process conditions. The output frequency of the PLL should be as constant as possible. Jitter, which is variation in the clock period, effectively decreases the amount of time in the cycle available to logic.

The clock distribution system is required to deliver the clock signal to all the flip-flops, latches, and macros in the chip at nearly the same time. The deviation from this time, called skew, is an important component of the hold time analysis. The main cause of skew in the clock distribution system is local variations in the amount of clock load. After the chip is assembled, this load is extracted and the required number of local clock buffers is programmed in using one of the metal contact layers.

*clock distribution*

Netlists describe the topology and connectivity of the circuit elements. Netlists usually adhere to a standard format (such as EDIF), which allow them to be used on different computer platforms and enabling various tools to be integrated based on the standard format.

### *Timing Analysis*

Each major functional block of the chip is typically realized as a top-level module based on the gate-level netlists and on the full-custom macro blocks. The initial timing of the gate netlist is accomplished with a static timing analyzer. A default wire load model is used and can be based, for example, on a non linear, statistical capacitive loading model which is a function of fan-out. The module is then floorplanned and the cells placed by hand with the help of some in-house placement programs. Following this, the cells are connected together according to the netlist. Using the placement data, a minpath analysis tool is run using clock skew estimates to ensure hold time constraints are met. After routing, a parasitic-extraction tool is run on the routed database to extract distributed RC delay values. The net delays are computed (e.g., by Ultima Delay Calculator) and input into a timing analyzer. Timing analysis is performed on each top level module, and ultimately on the whole chip. Design rule checks and layout versus schematic checks are run on the completed route. Finally, the database is analyzed for electrical integrity purposes (wire lengths, electromigration, max-transition violations, electrical rules checking, etc.). Any type of undesired results along the design flow causes looping iterations to take place until the entire chip meets all of the design constraints and is ready for tapeout.

The timing analysis and allocation or budgeting methodology is based on gate-level static timing analysis tools, more-accurate RC extraction tools, and delay calculation tools. Various in-house programs are often used to bind all of the above together. The timing analysis and budgeting methodologies are both designed to work together to provide consistency and accuracy throughout the evolution of the chip by making use of as much detailed design information as is available at any given time. As the design evolves, the timing methodology is required to support the following activities. In the early timing phase, time budgeting is done at the block and sub-block level to drive and check the consistency of timing constraints for synthesis or manual design. In the middle timing phase, pre- and post-layout timing analysis of major design blocks is done in the context of the whole chip, before the entire chip is ready to be timed. In the late timing phase, post-layout RC extraction and timing analysis are done on the entire chip.



## Cell Placement

Most cells are placed by using text-based directives which are read by an in-house set of programs. A graphical display of the results aids in integration. Most of the chip is placed using these scripts which greatly aid layout productivity and yield density and timing results similar to a full custom design.

The top-level modules are constructed such that they include all logic and any wires passing through their “air-space”. The inputs and outputs of each top level module are routed to a predefined I/O footprint which was derived by understanding the routing requirements of each top-level net. This allows most of the chip construction to be done early during the construction of the top level modules. Construction of the chip then consists merely of placing the top level modules and stitching them together with very short final routes.

## MASK GENERATION, WAFER FABRICATION, AND PACKAGING

Following assembly of the chip, the database goes through an extended release procedure which checks the layout versus schematic and verifies that the drawn geometries match the design rules defined by the process development group. At this point the database is taped out to the mask generation group. At mask generation, the first operation is to size several of the layers according to the latest data from the fab. Here the poly layer is tuned to get the best transistor channel length, for example. A final extended release procedure is done at this step. Following this, the database is fractured, that is, the complex polygons are broken up into rectangles, so that they can be handled by the mask writing hardware. The data are written on to the reticle which is a chrome-covered quartz plate used to project the patterns onto the resist-covered wafer when exposed by ultraviolet light in a machine called a stepper. The data are written onto the reticle using an electron beam machine. The data are then etched in the chrome and the plate inspected. This is repeated for all the mask layers used in the fab.

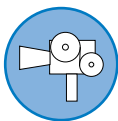
When the mask plates are shipped to the fab, the wafer fabrication process begins. The initial AMD-K6 processor was fabricated, for example, on a 0.35 micron CMOS process with five layers of metal, shallow trench isolation, tungsten local interconnect, and C4 flip-chip die attach. As is discussed in Chapter 2, the K6 3D design has migrated to a similar 0.25 micron process. The shallow trench isolation is required for tighter active area packing and smaller transistor width variation. The tungsten local interconnect is used to connect poly and diffusion without an intervening contact layer. It provides tighter layout of SRAM cells, standard cells, and custom macros. It is realized by a damascene tungsten process, which means the interconnect pattern is defined by trenches etched

in the oxide, filled with tungsten and later polished off. Chemical-mechanical polishing is an integral part of the process, providing planarization, stacked vias, and metalization density unobtainable by any other means. The C4 flip-chip die attach allows better power routing on the chip and low-inductance power supply connections to the package. The process is based on p-epi on a p+ substrate for less susceptibility to latchup. The transistors use a seventy-angstrom-thick gate oxide and the metal pitch is 1.4 microns, 1.8 microns, and 4.8 microns for metal layers 1-3, 4, and 5, respectively.

Following completion of the wafers and the deposition of the lead-tin solder bumps, they are tested by an automatic tester using an instrument called a cobra probe which contacts all the solder bumps with probe pins. The test program consists of a series of routines to verify the operation of every part of the processor. The built-in self-test structures are driven by this procedure to exercise each of the full-custom macros (except the PLL, which is verified by measuring the generated clock signal). There are four scan chains that are used to load the test patterns into the flip-flops. The data is then clocked through, captured in the downstream flip-flops, scanned out and compared with the expected values. In addition to this, the tester applies parallel test vectors to the input and output pins of the processor to complete the test. The test results are then recorded and the wafer is sent for packaging.

The wafers are cut up into separate die and, using a die location map, the good die are mounted onto a ceramic substrate with metalized contacts on the surface for contacting the solder bumps. The solder bumps melt during the refill heating step and make strong mechanical and electrical contact. A potting compound is then forced under the die, between the solder bumps, to make a more rigid assembly which increases reliability. Then decoupling capacitors are attached by reflow. These capacitors provide more stability to the power supply on the chip in the presence of electrical noise. After attaching the lid, the packaged part is sent for final test and measurement of its maximum frequency. Based on their performance, the parts are shipped in various speed grades to the customer.

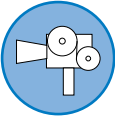
#### VIDEO ON CD-ROM



On the companion CD-ROM, there is a tutorial-level video on the steps involved in the fabrication of contemporary microprocessors and chips. There is also a film interview with Bill Siegle, AMD's Chief Scientist, about this technology.

The following video brings together a number of the issues discussed in the section concerning designing and implementing a microprocessor in the context of an actual microprocessor.

## VIDEO ON CD-ROM

	<p>Atiq Raza, Executive Vice-President and Chief Technology Officer of AMD answers the following question, “What was the discipline used to design and implement the K6 3D?”</p>
---	--

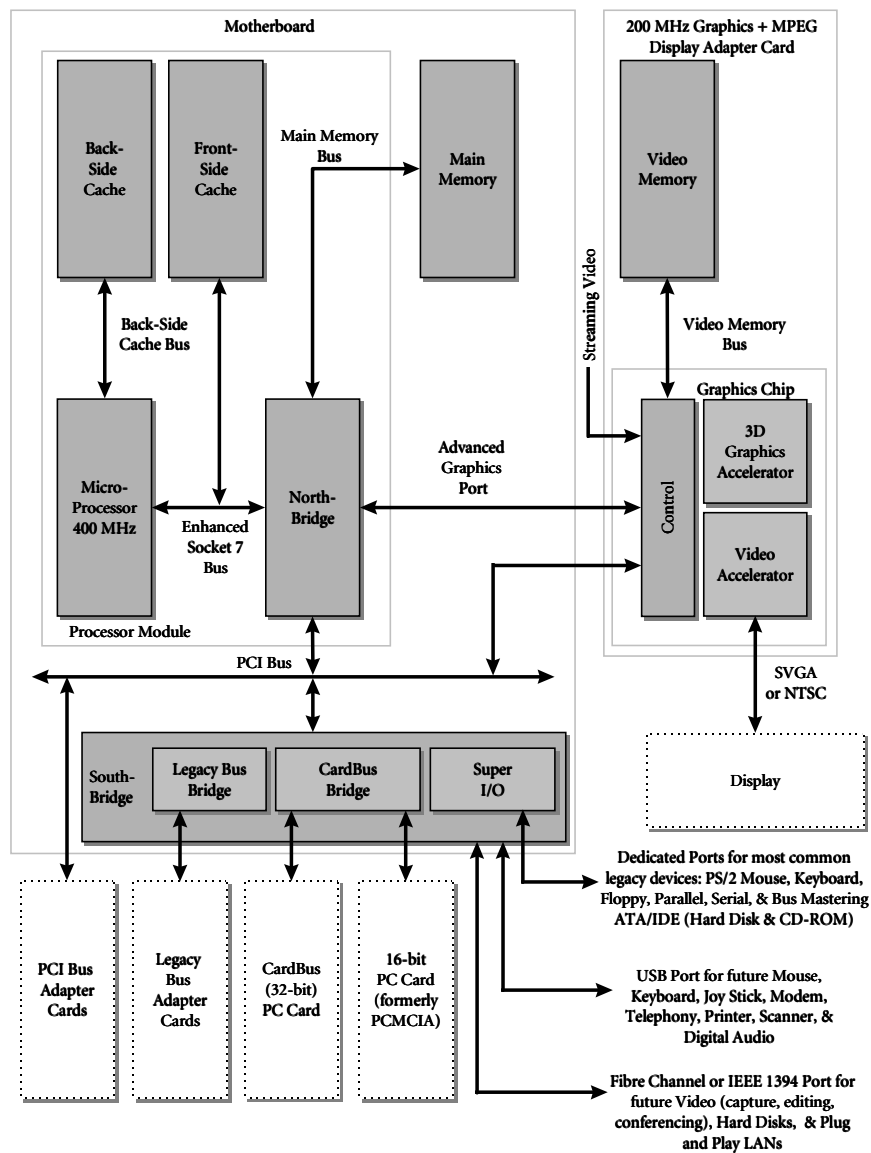
Having completed out overview of the design and implementation of microprocessors, we will not begin our examination of some of the related platform and systems issues.

We stated earlier in this chapter that because the PC platform dominates high-performance microprocessor-based systems, both in terms of unit and dollar volume, it exerts tremendous influence on the design of PC platform compatible processors, such as the K6 3D. Furthermore we noted that the PC-based systems environment is dominated by Microsoft Windows-based PC platforms. Therefore, we explore this environment by providing a guide to the hardware architecture of PC platforms that support Microsoft Windows. The examples used in the platform and systems chapters are based on a PC platform targeted for desktop consumer 3D graphics applications. 3D graphics continues to evolve, presenting a seemingly ever-growing demand on the bandwidth of the platform’s buses, peripherals, and sub-systems. We begin our journey into the systems-related issues of PC platforms by giving a high-level hardware overview of a 3D Graphics PC Platform for the consumer market. It is from this point of view that the systems-related chapters will examine the design and implementation of PC platforms and systems.

## DESIGNING AND IMPLEMENTING A 3D GRAPHICS PC PLATFORM

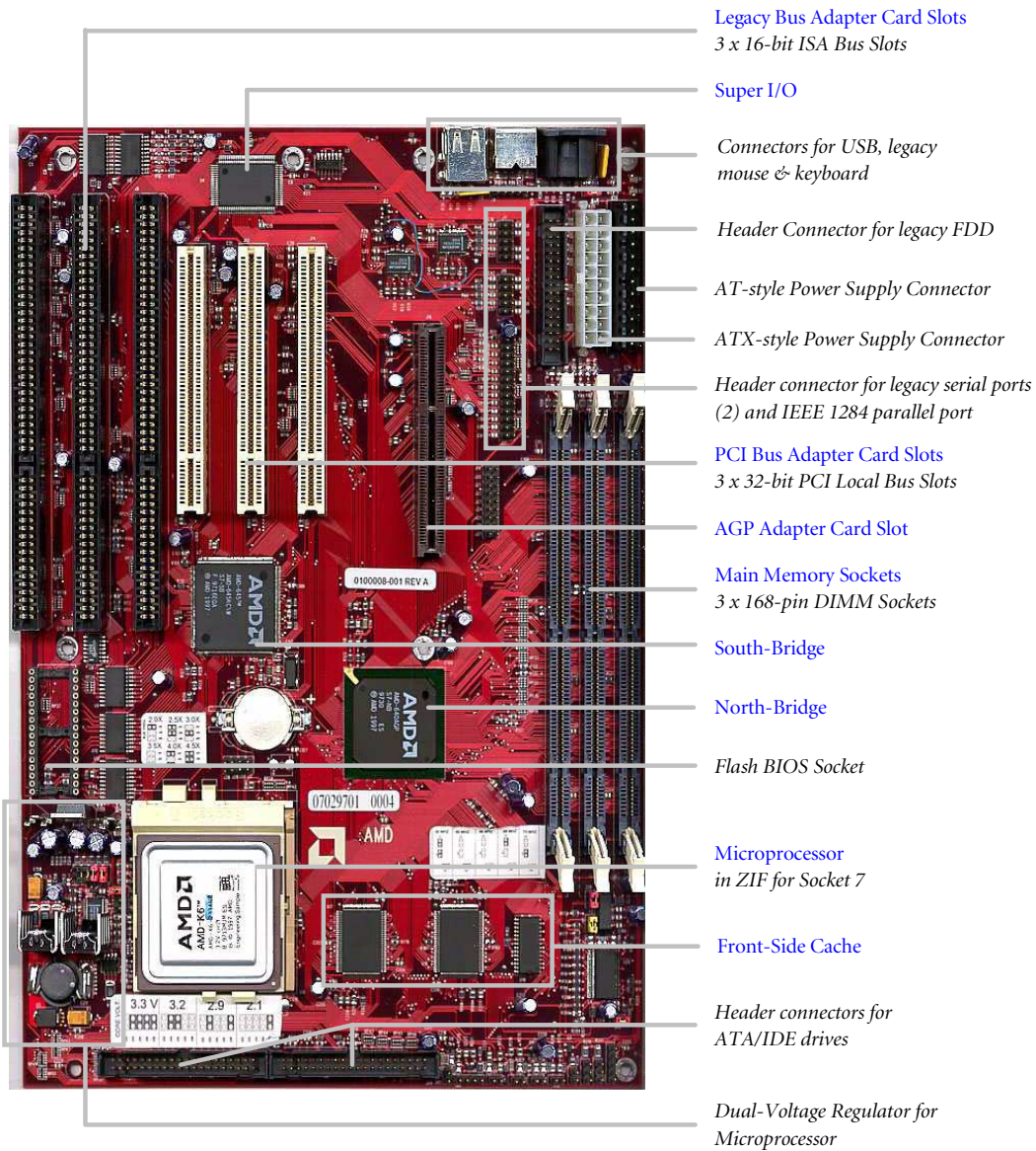
### PC PLATFORMS: KEY COMPONENTS AND INTERCONNECTIONS

An abstract view of a 3D graphics PC platform and associated system peripherals is provided by Figure 1.2. Consistent with the definition given for platforms on page xiv in the Preface, the platform consists of a number of key components and interconnections on a motherboard and typically includes a high-performance peripheral bus and ports, main memory, an I/O module, a processor module, and appropriate BIOS code. Generally, the processor and I/O modules are not physically distinct components, although the processor module shown in Figure 1.2 roughly corresponds with Intel’s Pentium II cartridge and its Slot 1 interface or AMD’s proposed K7 Processor Module and its Slot “A” interface (discussed in the Next-Generation Platforms section in Chapter 6). The processor module typically includes the processor, processor local bus, optional external cache, and a controller for the peripheral bus and main memory. The I/O module, as embodied by the South-Bridge shown, typically includes bus controllers and ports for standardized and optional peripherals.



This is a conceptual block diagram of a PC graphics platform for the 1998-1999 time frame. A front-side or back-side cache is generally present on the processor module. Connection to the Display Adapter Card is made via either the Advanced Graphics Port or the PCI Bus. PC Cards are generally found in laptop platforms and not in desktop or server platforms. Conversely, PCI Bus and legacy bus adapter cards are generally found in desktop or server platforms and not in laptops.

**Figure 1.2** 3D GRAPHICS PC PLATFORM



**Figure 1.3** PHOTOGRAPH OF A 3D GRAPHICS MOTHERBOARD

a. Board Courtesy of Advanced Micro Devices Inc., photo by Smith.

We have printed those features that closely correspond to features shown in Figure 1.2 in regular typeface (but with color accent on the CD-ROM), while new features, or additional feature details, are annotated in italics (and without color accent on the CD-ROM). The microprocessor, front-side cache, and North-Bridge, are implemented directly on the motherboard.

A complete 3D graphics PC system would, consistent with the definition for systems on page xiv in the Preface, consist of extending the foregoing platform descriptions to include an operating system, device drivers including BIOS extensions, other configuration and power management software, and a basic set of applications software. The selection, interaction, configuration and optimization of the components that make up Figure 1.2 and Figure 1.3 is the focus of Chapter 4.

### *The Microprocessor*

The microprocessor, under control of a multi-tasking operating system, carries out interleaved threads of execution for a variety of processes including: the various components of the operating system itself, one or more Application Programming Interfaces (APIs),<sup>3</sup> a number of system-software services and utilities, and one or more user-launched applications.<sup>4</sup> In addition, for 3D graphics-oriented applications, the processor maps graphics scenes to the display's viewport. For this mapping, floating-point operations on vectors are used to perform object modeling, geometry transform, clipping, and lighting calculations. The processor then performs rendering, setup operations, and prepares 3D display (execution) lists in main memory, which contain the mapped scene information and rendering commands for the 3D graphics accelerator. As an alternative to building display lists in main memory, the graphics system may be designed such that the processor directly writes triangle data, parameters, and commands directly to the 3D graphics accelerator.

The processor local bus shown is a 100-MHz Enhanced *Socket 7 Bus* and is discussed in the Processor Bus — Socket 7 section in Chapter 4 and again discussed in the Directions in Optimization of Contemporary Systems section in Chapter 6. The microprocessor is shown mounted in a *Zero Insertion Force (ZIF) socket* compliant with the Socket 7 standard. The term "Socket 7" was introduced in the text inset "Socket 7 Compatible" on page 7.

As discussed further in Chapter 4, many microprocessors today require two power-supply voltages, one for the core and one for the I/O circuitry. Such processors require a dual-voltage regulator, as shown.

*ZIF socket, or Zero Insertion Force socket, refers to a socket that permits a device with large numbers of pins to be dropped into the socket rather than requiring pressure insertion.*

---

<sup>3</sup> An API, or Application Programming Interface, is the collection of software routines that comprise a particular system-software facility. Application programs and other system software use API calls to access the services provided by the facility. The API is formally defined by a set of human readable procedure call definitions, including call and return parameters.

<sup>4</sup> Other than the operating system's user interface, user-launched applications, a few of the utilities and services, and operating system facilities explicitly invoked by the user, the user is generally unaware of the many processes being executed.



## The North-Bridge and South-Bridge

Two of the most important components used to couple together the other components on the motherboard are bridge chips referred to as the North-Bridge and the South-Bridge. Here are formal definitions for these components.

### DEFINITIONS

#### Bridge Chips, Chipset, Core Logic, System Logic

In a highly abstract view, *bridge chips* selectively couple (or isolate) two buses. In a more general view, bridge chips couple collections of motherboard components together. Bridge chips historically have been referred to as the *chipset*, *core logic*, or *system logic* of the motherboard. The chipset includes control logic for many of the platform components and I/O ports as well as providing data staging and selective coupling among the various buses and components of the platform.

Based on this, we can define the *North-Bridge* and *South-Bridge*.

### DEFINITIONS

#### North-Bridge and South-Bridge

The chipset is frequently partitioned into a North-Bridge device and a South-Bridge device. In the abstract, the North-Bridge selectively couples the processor to the primary peripheral bus (such as the present standard *Peripheral Component Interconnect Bus*, or *PCI Bus*, discussed in the Backplane Bus — PCI section in Chapter 4). More generally however, the *North-Bridge* typically has separate ports (interfaces) to the processor, main-memory, the primary peripheral bus, possibly an external cache, and possibly an AGP (Advanced Graphics Port).

In the abstract, the *South-Bridge* selectively couples the primary peripheral bus with a secondary peripheral bus (such as the ISA Bus, defined shortly in this section). More generally however, the South-Bridge typically has ports coupling the high-performance primary peripheral bus with a number of standard I/O ports and optional peripherals.



Other perspectives on the principal buses and ports of the North-Bridge and South-Bridge can be found on the CD-ROM in the data sheets for the AMD-640 System Controller and the AMD-645 Peripheral Bus Controller. These devices are the North-Bridge and South-Bridge shown in the motherboard photo in Figure 1.3.

*Peripheral Component Interconnect Bus or PCI Bus*

The North-Bridge serves a dual role as main memory controller and a bus controller. The bus controller manages the selective coupling of the buses

*Super I/O is a PC platform component that implements many popular secondary peripheral buses and standard I/O ports.*

*The 16-bit PC Cards were originally called PCMCIA cards. A CardBus PC Card is a 32-bit device and is not electrically or physically compatible with the older PCMCIA slots.*


*Flash EPROM*  
*shadow BIOS*

*Real Time Clock (RTC)*  
*CMOS Memory*

connected to the North-Bridge: the PCI Bus, the AGP (when present), the processor's local bus, and the main memory bus. The bus controller tries whenever possible to isolate these buses in order to maximize the speed of each and to permit concurrent operation of as many buses as possible. However, the bus controller couples buses together whenever a crossing transfer is necessary. The motherboard of Figure 1.3 on page 37 has three 32-bit PCI adapter card slots.

The South-Bridge typically resides on the PCI Bus and in conjunction with a separate or integrated *Super I/O* module, typically implements the following secondary peripheral buses and standard I/O ports: a legacy Bus (in particular the ISA Bus), dual interfaces for ATA/IDE drives (the most common form of hard disk drive), parallel (compliant with the IEEE 1284 standard), dual legacy serial (compliant with the RS-232 standard), keyboard, legacy Floppy Disk Controller (FDC), and a pointing device (compliant with a PS/2 mouse port). As discussed shortly, support for the ISA Bus is being phased out, support for the Universal Serial Bus (USB) is being added in the near term, and support for the IEEE 1394 (FireWire) is to be added eventually. In laptops, instead of providing legacy bus slots, the South-Bridge is typically used to couple credit-card size adapter cards called PC Cards, via the CardBus, to the PCI Bus. PC Cards come in newer 32-bit, and older 16-bit, versions. All of these I/O ports are again discussed in the PCI-based Ports section in Chapter 4.

REPORT ON CD-ROM

	A significant amount of detail regarding the many standard I/O ports associated with the Super I/O component can be found on the CD-ROM in the standard Microsystems Corporation (SMSC) data sheet for their FDC37B78x part, a "128-pin Ultra I/O with ACPI Support and Infrared Remote Control."
---	---

The South-Bridge usually provides access to an external nonvolatile<sup>5</sup> memory, which holds the system BIOS. Increasingly, the nonvolatile memory is in the form of *Flash EPROM*, permitting the BIOS to be upgraded with revisions downloaded from the Internet. Typically, the BIOS is copied during system initialization from the relatively slow nonvolatile memory, to an otherwise unused portion of the faster DRAM that composes main memory. The North-Bridge memory controller subsequently transparently maps requests for BIOS addresses to the *shadow-BIOS* in the DRAM. The South-Bridge also implements the system *Real Time Clock (RTC)* and *CMOS Memory*, a small memory for holding key system hardware configuration parameters. Both the RTC and CMOS memory are provided with independent battery backup, such that they remain functional when power is removed. The large round object between the South-Bridge and the microprocessor in Figure 1.3 on page 37 is the backup battery.

<sup>5</sup> Nonvolatile memory retains its contents when power is removed.



## Legacy Issues

HISTORICAL COMMENT	
Legacy Hardware	
The term <i>legacy</i> is frequently used as an adjective to describe various PC hardware and software standards that continue to be implemented long after the introduction of better alternatives. The legacy standards live on primarily because of continuing market demand for absolute backward compatibility with earlier generation products. Such compatibility demands persist until the market perceives that the benefits of upgrading overwhelmingly outweigh the costs to upgrade.	<i>legacy</i>
Generally, legacy PC platform standards are traceable to the <i>PC/AT (Personal Computer/Advanced Technology)</i> . This was IBM's very successful 1984 PC-design that firmly established the de facto industry standard for PCs. The PC/AT's broad success has been attributed to the fact that it was perceived to be a largely <i>open-architecture</i> <sup>a b</sup> design endorsed by the world's largest computer company. The only impediment to copying the PC/AT was its copyrighted BIOS. Soon functionally equivalent but independently developed BIOSes were written and less-expensive PC/AT <i>clones</i> were widely available.	<i>PC/AT</i>  <i>open-architecture</i>  <i>clones</i>
Despite numerous and ongoing technology advances, the PC/AT architecture has continued to have a pervasive residual impact on many aspects of the design of PC platforms. A notable example is the existence of several <i>legacy buses</i> . The ISA Bus was the peripheral (or expansion) bus used in the PC/AT, although the ISA terminology was not coined until years later. The term <i>Industry Standard Architecture (ISA) Bus</i> was coined in conjunction with the 1988 launch of the <i>Extended Industry Standard Architecture (EISA) Bus</i> . The EISA Bus was intended to be a relatively open alternative to IBM's 1987 <i>Micro Channel Bus</i> , which was perceived to be a <i>proprietary</i> , or closed-architecture, design. <sup>c d</sup> The Micro Channel Bus was intended as an ISA replacement for IBM's new PS/2 (Personal System/2) line of PCs, which in turn was intended to retake the PC market from the clone-makers. The EISA Bus was an initiative primarily pushed by Compaq and other system vendors. Like the ISA and EISA buses, the Micro Channel Bus is considered a legacy bus, but it is much less common. The ISA Bus is the most important legacy bus as it is found in the majority of desktops already in use. However, beginning in the early 1990s, servers have frequently used the EISA Bus instead.	<i>legacy buses</i>  <i>ISA Bus</i> <i>EISA Bus</i> <i>Micro Channel Bus</i> <i>proprietary</i>

<sup>a</sup> A fully open-architecture design is one whose associated intellectual property (such as patents, copyrights and trade secrets) is generally licensed to all interested parties with possibly only modest administrative fees.

<sup>b</sup> The PC/AT design was perceived to be largely open-architecture because it was built entirely from generally available components and sub-systems and key design documentation was not treated as a trade secret. However, IBM had never indicated that the design was freely licensed. IBM later began pursuing licensing fees for PC/AT-related patents from clone-makers that were not otherwise licensed for IBM patents.

<sup>c</sup> A proprietary, or closed-architecture, design is one whose associated intellectual property is not generally licensed. Many "de facto" standards have been "closed" during formulation but have been made "open" later. While very important, the ability to influence the standards development process is often less important than the ability to implement a given "standard."

<sup>d</sup> Key Micro Channel Bus design documentation was treated as trade secret and it was believed that IBM was asking prohibitively high licensing fees to use the intellectual property associated with the new bus.

### *Legacy Bus Bridge*

The ubiquitous PCI Bus has replaced the various legacy buses as the primary peripheral bus in PC platforms. In past desktop and server platforms, a *Legacy Bus Bridge* has been provided in the South-Bridge to selectively couple the PCI Bus and a legacy bus. Such a bus bridge between the PCI Bus and a legacy bus permits the continued use of legacy adapter cards. This hardware compatibility with ISA and EISA adapter cards greatly eased migration to PCI-based platforms, because expensive peripheral upgrades could be deferred.

In spite of the advantages of providing for backward compatibility, industry platform design standards (a key subject discussed in Chapter 4) proscribe certification of systems sold after mid-'98, if the ISA/EISA adapter card slots are populated prior to sale. Legacy adapter card slots are entirely proscribed from all systems sold after mid-'99. There are a number of reasons for this. These cards are discouraged because they generally do not incorporate the latest Plug and Play features (discussed in the Plug and Play Configuration and Maintenance section in Chapter 4), and thereby generally pose system configuration problems. Legacy cards also generally do not have power management features, and have generally narrower bus-widths and generally slower circuitry than do PCI adapter cards. Finally, the use of ISA adapter cards can reduce performance of the execution thread utilizing the card to a small fraction of that possible with PCI cards and may potentially starve other threads from execution while the legacy card is being accessed. As the performance of systems and the reliance on multiple threads of execution (e.g., to implement multi-media and execute background tasks) has increased, this last issue has become the foremost problem with the use of ISA cards.

As discussed above, the Super I/O component provided now largely obviates the need for legacy adapter cards. The Super I/O includes dedicated ports for the most common legacy devices. Generally, integrated ports are provided for a PS/2 mouse, serial devices (for modems and other communications), a keyboard, floppy drives, parallel devices, and bus-mastering ATA/IDE hard disks and CD-ROMs.<sup>42</sup>

*USB, or Universal Serial Bus, is a new standard for low to medium speed serial peripherals designed for hot plug and play connectivity.*

From 1998 and onward, platforms will likely provide a *Universal Serial Bus (USB)* port, which is intended by its promoters to obsolete and obviate the need for legacy expansion cards or Super I/O integrated legacy device ports, such as Sound Blaster audio, PS/2 mouse, and PC/AT-style game, serial communications, keyboard, and parallel interfaces. The USB is intended for replacement upgrade devices for the pointer/stylus (mouse), keyboard, joy stick, modem, telephony, printer, scanner, and digital audio devices. USB devices are low to medium speed serial peripherals designed for hot<sup>6</sup> Plug and Play connectivity. In spite of USB promoters intentions, the emergence of general purpose USB devices has

---

<sup>6</sup> "Hot" connectivity indicates that devices can be regularly added or removed while the system is operating normally and without any adverse effects.

been very slow. This is expected to change with the introduction of Windows 98, which has integral support for USB.

Future platforms are also stated to include either an *IEEE 1394* port, or a Fibre Channel port. These ports are for high-speed serial peripherals designed for hot Plug and Play connectivity. For example high-speed devices are video capture, editing, and conferencing units; hard disks; and Local Area Networks (LANs). 1394 has been touted as the serial technology of the future for many years now, yet such devices have not been generally available. With integral support for 1394 in Windows 98, 1394 devices are expected to begin their ascension. The advent of the Device Bay Standard (discussed in the Device Bay section in Chapter 4) should further accelerate the usage of 1394 devices.

*IEEE 1394 is a new standard for high-speed serial peripherals designed for hot plug and play connectivity.*

#### DEFINITION

##### Sealed PC

The USB and 1394 serial ports enable a consumer-oriented easy-to-use sealed PC. The sealed PC ideal is to provide a PC that never needs to be opened for the installation of after-market adapter cards. The name should not be taken too literally.



The 1394-1995 IEEE Standard for a High-Performance Serial Bus is included on the CD-ROM.

## Main Memory

The main memory controller arbitrates access to the main memory bus and main memory from the other buses. By using sophisticated data staging (including caching, prefetching, and posting of write data and commands), the North-Bridge creates the general illusion that the main memory has ports dedicated to each of the other buses.

The main memory uses Dynamic RAM (DRAM)<sup>7</sup> to provide volatile<sup>8</sup> storage of executing code and data. Multiple pairs of 72-pin DRAM *Single Inline Memory Modules (SIMMs)*, with each SIMM supporting a 32-bit (36-bits with parity, a method of detecting single-bit errors) wide memory data width, have typically been installed into platforms via sockets on the motherboard.

*SIMM, or Single Inline Memory Module, is a popular type of DRAM packaging that has 72-pins and supports 32-bit wide memory data widths.*

<sup>7</sup> DRAM memory relies on charge storage techniques and requires periodic refresh (reading and rewriting) to maintain the integrity of its contents.

DRAM is known for high-density storage.

<sup>8</sup> Volatile memory loses its contents when power is removed.

*DIMM, or Dual SIMM, is an emerging type of DRAM packaging that has 168-pins and supports 64-bit wide memory data widths.*

More recently, one or more 168-pin DRAM *Dual SIMMs (DIMMs)* are being used, with each DIMM supporting a 64-bit (72-bits with ECC, or Error Correcting Code, a method of correcting single-bit errors) wide memory data width. The motherboard of Figure 1.3 on page 37 uses three 168-pin DIMM sockets.

Areas in main memory used for code may be allocated to the operating system, APIs, and applications. Areas used for data include system and applications, graphics display lists, and graphics texture maps. Information in both areas is typically dynamically arranged and allocated using a virtual memory management mechanism. 3D graphics related APIs include DirectDraw and particularly Direct3D. Main memory is accessed frequently by the processor and cache, PCI bus-master peripherals, and the AGP. Thus, there are a number of trade-offs to be made when designing a platform centered around minimizing latency to instructions and data while maintaining as high a bandwidth as possible. Memory organizations are extremely important in optimizing system performance. In Chapter 5, we will examine a variety of memory organizations such as the waning mainstream EDO, the waxing mainstream SDRAM, and emerging memory technologies such as Rambus DRAM and SLDRAM.

Caches

A number of microprocessors like the K6 and the Pentium have on-chip L1-Caches. Others, such as the K6 3D, also have on-chip L2-Cache.

DEFINITIONS
<p>L1 Cache, L2 Cache, and External Cache</p> <p>An <i>L1-Cache</i>, or <i>Level-One-Cache</i>, is the cache that is placed closest to the processor in the memory hierarchy. An <i>L2-Cache</i>, or <i>Level-Two-Cache</i>, is one level removed from the processor in the memory hierarchy by the intervening L1-Cache.</p> <p>External caches intervene between the on-chip caches and main memory in the memory hierarchy. Each higher-level of cache is further removed from the processor (and closer to the main memory) and is larger and higher in latency than the caches closer to the processor. Additional external caching typically supplements the on-chip caches, especially in higher performance systems.</p>

External caches are connected to the microprocessor in two basic ways which we will refer to as a *front-side cache* and a *back-side cache*.

**DEFINITIONS****Front-Side Cache and Back-Side Cache**

A *front-side cache* is placed on the processor's local bus. A *back-side cache* uses a dedicated bus, separate from the processor local bus. A back-side cache generally operates faster than front-side cache. A back-side cache greatly reduces traffic on the local bus, while permitting more aggressive code and data prefetching from the attached cache. The advantage becomes less pronounced as the size of on-chip cache increases.

The external cache generally uses a variant of Static RAM (SRAM)<sup>9</sup> to hold recently used subsets of the code and data in the main memory. The SRAM provides transfers that are lower in latency, and higher in throughput, than the DRAM-based main memory. Properly managed, the combination of high-speed cache and large main memory virtualizes a single memory that has the capacity of the main memory and approximates the speed of the cache. Because the cache services most accesses, the processor to main memory traffic is reduced. This means that more of the main memory bus bandwidth is available for other sources and destinations, such as PCI bus-masters and the AGP.

### *Mechanical and Electrical Considerations*

PC motherboards are typically oriented in the ubiquitous tower-type (vertically oriented and floor standing) system unit housing such that the board edge shown at the bottom side of the photo is pointing toward the front of the tower and the board edge shown at the left side of the photo is near the bottom of the tower. The board edge shown at the topside of the photo is near the back of the tower, with system chassis access cutouts for the connectors shown in the top-right of the photo. External to the system unit, USB devices, a mouse, and a keyboard are cabled directly to these connectors. Additional cutouts are provided for mechanically securing adapter cards into the various adapter card slots and permitting access to adapter card connectors from the back of the tower. The board edge shown at the right side of the photo is pointing toward the top of the tower, generally just below the system power supply. In full-height towers, the board only occupies the lower portion of the tower.

---

<sup>9</sup> SRAM memory relies on active flip-flop (cross-coupled inverters) storage techniques and does not require refresh. SRAM is known for its high-speed storage.

Connectors are cabled directly to their respective motherboard headers for a pair of legacy Floppy Disk Drives (FDDs) and up to two pairs of ATA/IDE drives, the drives being generally mounted in the front top portion of the tower. (Only one device in each pair mentioned need be populated.) Connectors for the two legacy serial ports and the IEEE 1284 parallel port are generally mounted on the back of the tower either via dedicated cutouts or via covers for unused adapter card slots. These connectors are then cabled directly to their respective headers on the motherboard.

The dimensions of the original PC-AT motherboard are now known as the full AT form factor. The full AT form factor has been largely replaced for some time by a smaller variant (roughly two-thirds the size of the full AT), the *Baby AT form factor*. The Baby AT board shown in Figure 1.3 on page 37 is 8.5" x 12" in size. Recently, several new motherboard standards have been introduced that have new sizes, new board orientation, and new component placements. These include the NLX (New Low-profile eXtension), ATX (AT eXtension), and mini-ATX standards, which are briefly discussed in the Mechanical Design section in Chapter 4. These new motherboard standards require the use of a system chassis that is specifically designed to accommodate the new boards.

A new power supply configuration was also developed in association with the new ATX boards. The *ATX Power Supply* that is part of the ATX standard specifies a new standby voltage (5VSB), a new power enable control signal (PS-ON), and a power status signal (PW-OK), that enable motherboard BIOS control over the power supply to provide energy savings. New boards still using the Baby AT form factor, but providing BIOS control for power supply management, are also implementing the new ATX-style Power Supply Connector for the new control and status signals. This permits Baby AT form factor boards to work with ATX power supplies, which can be used in a chassis designed for the older Baby AT boards.

## DISPLAY ADAPTERS

The *Display Adapter* card shown in Figure 1.2 on page 36 includes video memory, a 2D/3D graphics accelerator, and a video accelerator. The *video memory* generally uses high-performance or specialty DRAM to provide storage of bit-maps and related parameters. The video memory may include data areas devoted to frame buffers, video buffers, a texture map cache, cursors and sprites, a depth or z buffer, an alpha buffer, and window coordinates. Platform features that access the video memory include the graphics and video accelerators, the streaming video input, PCI bus-master peripherals, and the AGP port.

Integrating some or all of the video memory, graphics accelerator, video accelerator, and AGP functions of the display adapter onto the

*Baby AT form factor*

*An ATX Power Supply is capable of being managed by the platform's BIOS to provide energy savings.*

*Display Adapter video memory*

motherboard and possibly into the north-bridge is tempting. Display adapter integration is commonplace in portable platforms and has been attempted at the low-end of both the consumer and business markets in efforts to increase the functionality of entry-level systems. More importantly, if a baseline for high-performance graphics functionality were established, software developers could rely on the baseline in developing their applications. All applications would benefit from the higher standard and the overall experience of the end user would be enhanced.

Other than portable platforms, where display adapter integration is a practical necessity, such integration is nevertheless generally not done, even for low-end systems. This is due to a number of factors. Historically, there has generally always been a wide range in performance, features, and cost for display adapters corresponding to a large variation in end-user needs. This makes the feature set selection difficult. Also, display technology and software generally are evolving rapidly and a number of competing solutions will exist. This presents both development and marketing risks. It is unlikely that the display adapter market will stabilize anytime soon, so these problems can be expected to continue. In AGP systems, providing both an integrated solution and the ability to later upgrade with an expansion display adapter card may require a revision to the specification. This is due to the extra loading that would exist on an interconnect originally envisioned as having only a source and a single destination.

#### DEFINITION

##### Frame Buffer

A *frame buffer* is that portion of the video memory used to compose images for subsequent display. There is a dedicated memory location corresponding to each addressable pixel of the active display area. In a color system, the frame buffer usually consists of separate RGB “planes” for each of the color components. There may also be additional planes for special pixel attributes. Generally, each of the color planes has an 8-bit byte, for a total color depth of twenty four bits.

During image composition, the frame buffer is generally written at random (nonsequential) locations. The frame buffer need only be written when a change is desired in the displayed image.

During display, frame buffer locations are read sequentially, in conjunction with the raster scanning of the attached video display. Because the light emissions from the present phosphor-based CRT displays decay with time, the displayed image needs to be continually refreshed, and hence the frame buffer needs to be continually read.



Pixel, or picture element, refers to the smallest resolvable or addressable feature of a computer display.

raster

HISTORICAL COMMENT

Raster Displays

A *raster display* is characterized as having a rectangular array of discrete *pixels* (picture elements) of varying color or gray-scale intensity. Cathode Ray Tube (CRT)-based displays were the first raster displays. Such raster displays have horizontal and vertical deflection coils that are driven by fixed frequency saw-tooth waveform “sweep” oscillators. The active portion of the beam traces a rectangular scanning pattern on the face of the CRT known as the *raster*. Raster displays are by far the most prevalent paradigm for implementing computer displays. In contrast, vector displays may create images from line segments of generally arbitrary length and orientation. While the underlying technology and low-level electronics is quite different, LCD panels are managed at a high-level like CRTs. Flat LCD panels are expected to eventually replace the CRT for mainstream applications.

front buffer  
back buffer

video DRAMS or VRAM

2D and 3D graphics accelerators

Implementations that use a single frame buffer must effectively manage contention between image-building writes, display-refresh reads, and the refresh of the frame buffer's DRAM storage cells. Single frame buffers built from standard DRAMs must generally be written only during periods when the CRT display is undergoing horizontal and vertical retrace. This greatly restricts accesses, generally increasing the latency for performing frame buffer writes and dramatically reducing the effective read and write bandwidth for other than display refresh.

In contemporary systems there are typically two frame buffers, having the designations front buffer and back buffer. Frames are rendered to the *back buffer* as the display is painted (refreshed) from the *front buffer*. The roles of the buffers are then reversed during the next frame.

High-end frame buffers are often built from two-ported specialty DRAMs known as *video DRAMS* or *VRAM*. One port is a conventional random-access port that is used for the image-building writes. The second port is a serial output port that shifts out sequential locations used by the display refresh function. VRAMs increase the bandwidth available for frame buffer accesses, but are substantially more expensive than conventional DRAM.

A *2D graphics accelerator* is a special processor designed to execute the graphics display lists set up in main memory to build and move bit-maps and pixel maps in the video memory. A *3D graphics accelerator* also creates or renders 3D triangles into pixels in the frame buffer, often incorporating texture maps in the process. PCI bus-master capability permits the graphics accelerator to fetch the display commands and data from main mem-



ory without further involvement of the processor. Management of the various dedicated areas of the video memory is closely associated with the graphics accelerator.

3D acceleration features provide hardware support for *rasterizing* and *rendering* the 3D objects modeled by the 3D application running on the microprocessor once the objects have been broken down into sets of triangles with screen coordinates, color, and texture data computed by the graphics software. 3D-specific acceleration includes depth queuing and texture, transparency, and shading effects. 2D acceleration features provide hardware for 2D drawing (circles, triangles, lines, and points), raster operations (including window management and acceleration), and VGA (Video Graphics Adapter, a display adapter standard) register and memory compatibility. Display adapters intended for extensive 3D acceleration should make use of the AGP.

*rasterizing*  
*rendering*

#### DEFINITION

##### Advanced Graphics Port (AGP)

The AGP provides a high-bandwidth path between main memory and the display adapter for the large volume of texture data associated with 3D graphics. It also keeps this traffic off of the PCI Bus, increasing the available PCI bandwidth.

The *video accelerator*'s primary job is to pump streams of pixel data to the display. Digital pixel data is fetched from the frame buffer or video buffer in video memory, the pixels having been rendered previously by the graphics accelerator or previously received from the streaming video input. The pixel data may be stored in memory in a variety of formats, which have various color depths<sup>10</sup> and associated storage packing densities, color spaces,<sup>11</sup> and degrees of sub-sampling.<sup>12</sup> After being fetched from video memory, the various pixel data memory formats are respectively unpacked, interpolated,<sup>13</sup> color space converted, and scaled (if desired), to convert all pixel data to fully sampled RGB (Red, Green, Blue).

*video accelerator*

<sup>10</sup> "Color depth" relates to the number of available colors that can be explicitly specified for a pixel.

<sup>11</sup> "Color space" relates to different standard paradigms for specifying color.

<sup>12</sup> "Subsampling" is a color-video specific data compression technique.

<sup>13</sup> Interpolation is the inverse of subsampling.

*RasterOps*

*BitBLT (Bit-aligned Block Transfer)*

*RGB color space model*

*YUV color space model*

DEFINITIONS
<p>Rasterizing and Rendering</p> <p>Rasterizing and rendering are terms for related processes. While the two terms are often used synonymously, there are differences between the two. <i>Rasterizing</i> is a pixel-centric process of taking image data in any continuous form or model and processing it for storage, transfer, or display as a 2D matrix of modulated-pixel values. <i>Rendering</i> may be loosely used to mean simply rasterizing, but it often connotes surface modeling using a more comprehensive process that is image-perception-centric. Specifically, the color or gray-scale intensity of each pixel in the 2D matrix may be established through processing that may rely upon adjacent regions of pixels, textures, depth information, and other sophisticated techniques.</p> <p>Raster operations (frequently called <i>RasterOps</i>) are logical primitives defined for manipulating and moving bit-map and pixel-map data. <i>BitBLT (Bit-aligned Block Transfer)</i> is perhaps the most important <i>RasterOp</i>. <i>BitBLT</i> moves a bit-mapped image from a source area at a first bit-origin to an equal sized destination area, having a second bit-origin. <i>BitBLTs</i> are extensively used for <i>intra</i> memory transfers from main memory to the back buffer, and <i>intra</i> back buffer transfers. Other <i>RasterOps</i> extend the basic <i>BitBLT</i> via an additional operand that defines various pixel manipulations on the image being transferred. In the most general case, the resulting data may be a function of the source data, the preexisting destination data and the additional operand.</p>

Just as there are multiple coordinate representations (e.g. cartesian and polar) for physical space, there are multiple coordinate representations for color space. Two popular representations are RGB and YUV. *RGB* is a color space model that is directly usable by hardware at the sensor and display level. If a different color space model is used elsewhere in the system, color space conversion must be performed between the two. *YUV* is a reference to the color video image coding components Y, U, and V, which are formally defined for composite (single signal) analog color video standards such as NTSC, PAL, and S-video. For component (three signal) digital video, as used in computer graphics, the corresponding system is correctly referred to as Y'CbCr. However, in the general PC platform literature, the usage of the term YUV to connote computer digital component video is nearly ubiquitous.

After digitization and before display, the *YUV color space model* is common for the transmission and intermediate storage of digital “full-motion” video images. YUV is related to RGB color-space via a 3x3-

matrix manipulation, enabling either representation to be derived from the other.

The technology associated with YUV color-coding is complex, relying on the four disciplines of physics, perception, photography, and video.  $Y'$  is the luma signal, which is a scaled and gamma corrected (end-to-end-video-path compensated) representation of the luminance information in the video image. A video industry standard defines luminance as an objective definition of brightness, given by  $Y' = 0.299R + 0.587G + 0.114B$ . The different weightings given to each component take into account human vision sensitivities to power at different visible wavelengths.

Collectively,  $U'$  and  $V'$  as a pair are representations of the chromatic or chroma information in the video image. In component video, Chroma ( $C$ ) is a quadrature modulated signal given by  $C = U'(\cos t) + V'(\sin t)$ . Individually,  $U'$  and  $V'$  are scaled and gamma corrected color difference signals.  $U'$  is defined to be Blue ( $B$ ) minus  $Y'$ , and is sometimes referred to as the Blue Chroma component.  $V'$  is defined to be Red ( $R$ ) minus  $Y'$ , and is sometimes referred to as the Red Chroma component.

YUV systems often exploit the fact that human perception of color resolution is coarser than perception of luminance resolution. These systems subsample (periodically omit samples of) the chroma in order to transmit and store lower bandwidth signals without any human perceivable degradation. Interpolation is the process of using the sub-sampled chroma information to recreate the missing chroma samples prior to display.

Color depth is the number of bits assigned per pixel to represent its color. The greater the number of bits, the deeper or higher the color depth, and the more colors may be used. Unless further qualified, this view of color depth implicitly means one is talking about RGB pixels in a form ready for processing by the final stages of the video accelerator. In the context of YUV pixels, and sub-sampling, one must find an effective number of bits per pixel.

PC color-depth schemes may include 16-color, 256-color, 16-bit color, and 24-bit color modes. The 24-bit color mode is known as the *true color*, and generally corresponds to the highest capabilities of the display adapter. The true color mode is suitable for photographic quality images such as digitized photographs, or the results of photo-realistic graphics rendering, both of which require smooth shading of geometric objects. 16-bit color is known as *high color*, achieving very good quality images, but with some degradation.

The 16-color and 256-color modes are also known as *pseudocolor* or colormapped modes. They are also described as having “palletized” pixels. In the pseudocolor modes, the color pixel value stored in the frame buffer is not itself the ultimately displayed color, but is instead an index into a Color Look Up Table (CLUT), also known as a colormap, or palette. The CLUTs are used to convert palletized pixels to the desired high or true-

*Primes are used to represent that the signals are gamma corrected, which is a compensation necessary due to nonlinear intensity reproduction transfer functions inherent in video systems.*

*When used on a PC, the Quick-Time videos on the companion CD-ROM should generally be viewed with 16-bit or greater color depth.*

*true color*

*high color*

*pseudocolor*

color pixel data. Following color look up, which is required only for the pseudocolor modes, the pixel data streams drive an RGB triplet of Digital to Analog Converters (DACs), for generating analog waveforms suitable for driving the display.

HISTORICAL COMMENT
<p>RAMDAC</p> <p>CLUTs are often implemented using embedded RAMs and the term RAMDAC was coined to refer to a module that integrated both the CLUTs and the DACs. The term is still used even though much higher levels of integration are common today. Especially at the low-end, display adapter cards frequently consist of little more than video memory, one or two display controller chips, and other minor components.</p>

Thus a palletized pixel value corresponds to a higher true color only via the defined palette mapping, and is otherwise arbitrary. The pseudocolor modes have been analogized<sup>14</sup> to “painting by numbers,” where the set of numbers is relatively small, but each individual color can be chosen from the full spectrum of colors. The pseudocolor modes are suitable for artwork such as business presentations and drawn illustrations and for controls and program displays in alphanumeric and GUI (discussed in the Graphical User Interface (GUI) section in Chapter 4) based systems. Pseudocolor modes are very space efficient for such uses. Additionally, special effects may be achieved by dynamic changes to the color palette.

The choice of color mode is usually limited by the finite memory installed in the display adapter in conjunction with the addressable resolution selected. Given that the installed memory is a constant, the addressable resolution must be traded off against color-depth. That is, an increase in one will often demand a decrease in the other.

When low color depth is used, and in particular for hardware with wide data paths, multiple pixels may be *packed* into a single data word. It is also possible for packing to be done in such a manner that pixels straddle word boundaries. *Unpacking* is the process of parsing a data word stream to extract the individual pixels.

*Scaling* is the magnification or reduction of a raster image. Pixel replication, or decimation, may be used to perform crude integer magnification, or reduction, respectively. Noninteger scaling usually requires special hardware support or else performance may dramatically suffer.

We have now completed an overview of the process of designing and implementing a microprocessor and the issues involved in designing and

<sup>14</sup> Charles A. Poynton, *A Technical Introduction to Digital Video*, John Wiley and Sons, 1996, p. 36.

*packed pixels*

*unpacking*

*scaling*

implementing microprocessor-based platforms and systems. Before summarizing the chapter and discussing the road ahead, we have a small but hopefully interesting side-tour to make.

Each year since 1979, the IEEE Computer Society and the Association for Computing Machinery (ACM) have jointly recognized substantial contributions to computer and digital systems architecture by awarding the prestigious Eckert-Mauchly Award to a single individual. We asked a number of the recipients of this award the following two questions:

1. Looking back, what are the most important 5-6 books or articles that affected the way that you approach the central issues in computer architecture?
2. Looking forward, what are the most important 5-6 books or articles that you would recommend all of those interested in the field—student or practitioner—to read because you believe they are concerned with issues that will be increasingly important in future architectures?

Given the sustained contributions these individuals have made to the field and the impact each has had on it, we thought that most readers of this book could benefit from their responses to these questions. As you probably suspect, the answers are both interesting and insightful. The responses are presented in reverse chronological order, starting with the most recent award winner who we approached, and working backwards.

### 1997 Award Winner, Robert Tomasulo

**Award Citation:** *For the ingenious “Tomasulo’s Algorithm,” which enabled out-of-order executive processors to be implemented.*

#### Looking Back

*“Books played a negligible role in my development for two reasons. When I started there were virtually no books in the field. Even useful articles or papers were rare. Books also tend to lag too far behind practice in a rapidly developing field. This is still the case today. By far the most significant influence was the people I worked with and the design community to which we belonged. I learned the ABC’s of computer architecture from Amdahl, Brooks and Blaauw, Cocke et al. I learned computer design from Anderson, Sparacio and other colleagues on the Model 91 and subsequent projects.*

*A powerful external influence was the work of Seymour Cray. A paper on the 6600 and a video lecture (much later) come to mind. Sadly, much excellent early work, even at IBM, was not well documented publicly. Starting*

## PERSPECTIVES FROM ECKERT-MAUCHLY AWARD WINNERS

*with the Model 91 the IBM Journal has changed that, at least with respect to those of their designs that become products.”*

#### Looking Forward

*“I cannot improve on Prof. Flynn’s<sup>15</sup> answer to this difficult question but would like to add a greater emphasis on mastering the past. Technology advances in both hardware and software have rendered most Instruction Set Architecture conflicts moot (and I include CISC/RISC). Computer design is focusing more on the memory bandwidth and multiprocessing interconnection problems. Neither of these is new. Even in his special field, Cray realized that problems are ultimately memory (including I/O) limited. The microprocessor field has recapitulated in the last ten years the past forty years of mainframe evolution. Therefore, the problems it faces today were encountered ten or twenty years ago by mainframe designers. A first step toward solving these problems should be an understanding of past attempts, both those that succeeded in their day and, even more importantly, those that failed.”*

#### **1996 Award Winner, Prof. Yale N. Patt**

**Award Citation: For important contributions to instruction level parallelism and superscalar processor design.**

#### Looking Back

*“It is not clear that there are five or six books or articles that have affected how I approach central issues in computer architecture. The fact is, I have been influenced overwhelmingly more by lectures than by books or articles. I have thought about this a lot (on airplanes etc.), trying to identify the books and articles that have mattered to me, and frankly come up with the realization that it was lectures, not books and articles, that mattered far more. One exception: Knuth, Volume 1, which is not that it related directly to computer architecture, but rather it provided clear insight into what computing is about, where problems are, and how to approach problems.*

*As for lectures, I would count the following four, giving me the five that you requested:*

- 1. Professor W. K. Linvill, Systems Analysis, Stanford, 1964. Taught me to cut through the complexity and big words and get at the heart of the problem. Also taught me the value of analogies to understanding new situations.*
- 2. Professor W.K. Linvill, Doctoral Qualifying exam, Stanford, 1963. Taught me that tough problems can be cracked if they can be transformed into problems that I understand.*
- 3. Professor Donald Epley, Switching Theory, Stanford, 1962. Taught me that a body of knowledge, if organized systematically and compre-*

---

<sup>15</sup> Note: Prof. Flynn’s response appears later in this section.

*hensively, lends itself to understanding far better than a collection of badly interconnected ideas.*

4. *Professor Michael Harrison, lecture at Duke University, 1973. Taught me that an advancement of knowledge is best explained in the context of simple ideas, and that only after conveying that should one attempt to translate the advancement into heavy mathematics.*

*My response departs sufficiently from what you intended that I feel obliged to add a statement, lest some regard it as frivolous. While none of my five are limited to computer architecture, all five have had such a pronounced effect on how I approach everything about computer architecture that they dwarf any computer-architecture-specific books or articles that I could mention.*

### Looking Forward

1. *“Harold Stone, textbook published by A-W on high-performance computer architecture, because Harold writes lucidly, and provides a careful foundation that is useful to everyone’s base of knowledge in computer architecture.*
2. *John Wakerly, textbook on logic design, because one should have a solid grounding in digital logic design if one wishes to deal with computer architecture.*
3. *Martin Graham, textbook on high-performance circuit design, because one should be exposed to the lower levels of implementation as part of one’s preparation to work at the higher level of microarchitecture.*
4. *Stephen Melvin and Yale Patt, HICSS 1987, I believe. Hopefully I have transcended the obvious self-serving element of referencing my own work; this paper differentiates hardware/software interface from dynamic/static interface from user/builder interface, and in so doing, focuses on the important problems that must be dealt with to really produce highest performance engines.*

*I hope the above is useful.”*

### 1992 Award Winner, Prof. Michael J. Flynn

**Award Citation: For his important and seminal contributions to processor organization and classification, computer arithmetic, and performance evaluation.**

### Looking Back

*“I started in computer design when the field was relatively young. Aside from von Neumann’s classic papers [2], most of the books or articles that affected my personal approach to computer architecture were based upon machines themselves—either instruction sets or implementations, or both.*



Thus, Buchholz's book on the Stretch computer [1] was an important book describing a deeply pipelined machine and many of the problems which still exist in processor implementations (for instance, speculating on a branch was a feature in the Stretch machine). Other works that emphasized the evolution of instruction sets included System 360's Principles of Operation [3]. The DEC VAX and PDP-11 instruction sets [4] and the Burroughs B6500 series [5] all illustrated important advances in our understanding of optimizing the executable representation of programs. The Intel x86 instruction set and its evolution was interesting for a different reason. As technology enabled more robust implementations, it also enabled more fully functional instruction sets.

Machine implementations are equally important, especially machines that were able to break new ground in achieving performance or functionality. Books and papers on the CDC 6600 [6], the IBM System 360 Model 91 [7], and the CRAY-1 [8], all present to me valuable insights.

The reader will note that many of the items mentioned are of some vintage. It is easier to see in the hindsight of, say, a decade, which machines have changed and significantly influenced the field and hence all of our thinking. It is much more difficult to assess the impact of machines introduced in current months. Still, it is no less important to be aware of them."

#### References:

1. W. Buchholz and R. S. Ballance, Planning a Computer System, McGraw-Hill, 1962.
2. John von Neumann, Collected Works, Volume V, Pergamon Press, 1963.
3. IBM Corporation, IBM System/360 Principles of Operation, Technical Report GA22-6821-4, 1970.
4. C. G. Bell, J. C. Mudge, and J. E. McNamara, Computer Engineering: A DEC View of Hardware System Design, Digital Press, 1978.
5. E. Organick, Computer Systems Organization: The B5700/B6700 Series, Academic Press, 1973.
6. J. E. Thornton, Design of a Computer: The Control Data 6600, Scott, Foresman and Co., 1970.
7. D. W. Anderson, F. J. Sparacio, and R. M. Tomasulo, "The IBM System/360 Model 91: Machine philosophy and instruction-handling," IBM Journal of Research and Development, 11(1):8-24, 1967.
8. R. M. Russell, The CRAY-1 Computer System, Communications of the ACM, 21(1):63-72, January 1978."

#### Looking Forward

"It is difficult to predict trends in a field where the technology is rapidly changing, so the architect must be aware of possible avenues of machine implementation. But, underlying these avenues are somewhat more perma-



nent support tools. Computer architecture, like all engineering, is the art of the possible: bringing together ideas for machine implementation with state-of-the-art technology details to provide for the best possible system implementation at a given cost.

I think that the future computer architect is a systems architect, not simply a processor architect; so one must bring together software technology, systems applications, arithmetic, all in a complex system which has a statistical behavior that is not immediately or simply analyzed, so the architect must be aware of current techniques [1]. Here, basic books such as Hayes [2] or Hennessy and Patterson [3] through advanced work by Hwang [4] represent a sampling of the various avenues that the architect should be aware of. But just as importantly, the architect should have basic familiarity with probability and queuing theory, compiler theory, operating systems, and of course VLSI technology. In the more distant future, as the system moves to the chip, the architect needs to know signal processing, graphics, audio, and the human-machine interface.”

#### References:

1. A good source includes Microprocessor Reports or IEEE Micro.
2. J. Hayes, Computer Architecture and Organization, McGraw-Hill, 1988.
3. J. L. Hennessy and D. A. Patterson, Computer Architecture: A Quantitative Approach, Morgan Kaufmann, 1990.
4. K. Hwang, Advanced Computer Architecture: Parallelism, Scalability, Programmability, Computer Engineering Series, McGraw-Hill, 1993.”

#### 1988 Award Winner, Prof. Daniel P. Siewiorek

**Award Citation: For outstanding contributions to parallel computer architecture, reliability, and computer architecture education.**

“In my early career I studied the IBM 360 architectural papers, especially the 360/Model 91 implementation papers, and the CDC 6600. Taken together these were perhaps the first mass-produced, reduced instruction set architectures with high-performance implementations. The concepts in these two architectures fueled high-performance designs for over three decades. The DEC PDP-11 architecture introduced me to the interface between hardware implementation and software (both operating system and application programs). The single most influential book on my early career was C. G. Bell and A. Newell, Computer Structures: Readings and Examples, McGraw-Hill, 1971. Bell and Newell formalized the hierarchical levels in computer systems, initiated the concept of a computer space (taxonomies of alternatives that made design decisions explicit), introduced ISP as a language to describe

the programming interface (a language to which many concepts in contemporary hardware description languages can be traced), and the concept of the computer as a system reaching out beyond the data paths and controllers into software and networks.

As you can guess, it is easier identifying historically important articles/books than predicting the future. Hence there are more entries to your question 1 than to question 2. After I cite each reference, I will make a brief comment about it.

### Looking Back

1. Arthur W. Burks, Herman H. Goldstine, and John von Neumann, "Preliminary Discussion of the Logical Design of an Electronic Computing Instrument," in A. H. Taub (ed), *Collected Works of John von Neumann*, Vol. 5, pp. 34-79, The Macmillan Company, 1963. 'Introduces the basic organization of a processor, instruction set, multiple level storage hierarchy, number representations, reliability (e.g., duplicated processors), and graphics output.'
2. T. Kilburn, D. B. G. Edwards, M. J. Lanigan, and F. H. Sumner, "One-level Storage System," *IRE Transactions, EC-11*, Vol. 2, pp. 223-235, April 1962. 'Introduces the principles of an automatically controlled memory hierarchy.'
3. R. M. Tomasulo, "An Efficient Algorithm for Exploiting Multiple Arithmetic Units," *IBM Journal*, Vol. 11, January 1967, pp. 25-33. 'Introduces basic algorithms for controlling multiple functional units.'
4. L. G. Roberts and B. D. Wessler, "Computer Network Development to Achieve Resource Sharing," *Proc. AFIPS SJCC*, Vol. 36, 1970, pp. 543-549. 'Introduces basic concepts for networking and wide area network goals.'
5. Robert M. Metcalfe and David R. Boggs, "Ethernet: Distributed Packet Switching for Local Computer Networks," *Comm. ACM*, Vol. 19, No. 7, July 1976, pp. 395-404. 'Introduces local area network concepts.'
6. C. P. Thacker, E. M. McCreight, B. W. Lampson, R. F. Sproull, and D. R. Boggs, "Alto: A Personal Computer," in D. P. Siewiorek, C. G. Bell, and A. Newell, eds. *Computer Structures: Principles and Examples*, McGraw-Hill, 1982, pp. 549-572. 'Introduces the concept of a single user workstation with bit-mapped graphics, precursor to widely adopted icon, mouse interface with 'what you see is what you get' whole screen editors.'
7. C. G. Bell, J. C. Mudge, and J. E. McNamara, *Computer Engineering: A DEC View of Hardware System Design*, Digital Press, 1978.

*'Evolutionary design and computer families.'*

8. M. Y. Hsiao, W. C. Carter, J. W. Thomas, and W. R. Stringfellow, "Reliability, Availability, and Serviceability of IBM Computer Systems: A Quarter Century of Progress," IBM J. Res. and Development, Vol. 25, September 1981, pp. 453-465. *'The evolution of reliability techniques.'*

#### Looking Forward

1. A. S. Tanenbaum, Computer Networks, 3rd ed. Prentice Hall, 1996, Upper "Computer networks."
2. H. Cragon, Memory Systems and Pipeline Processors, Jones and Bartlett Publishers, 1996. "Memory hierarchy and pipelining."
3. J. L. Hennessy and D. A. Patterson, Computer Architecture: A Quantitative Approach, 2nd ed. Morgan Kaufmann, 1996. "Uniprocessor/cache design."
4. J. D. Foley and A. van Dam, Fundamentals of Interactive Computer Graphics, Addison-Wesley, 1982 "Computer graphics."
5. D. P. Siewiorek and R. S. Swarz, Reliable Computer Systems: Design and Evaluation, 2nd Edition, Digital Press, 1992. Butterworth Heineman Publishing Co. "System reliability continues to increase in importance."

#### 1986 Award Winner, Prof. Harvey G. Cragon

**Award Citation: For major contributions to computer architecture and for pioneering the application of integrated circuits for computer purposes. For serving as architect of Texas Instruments, scientific computer and for playing a leading role in many other computing developments in that company.**

#### Looking Back

*"There are two books and two papers that had a profound influence on my early life as a computer architect. The two books are:*

1. Planning a Computer System, Project Stretch edited by W. Buchholz, 1962. *"This book spelled out the design decisions that went into Stretch. To a young engineer, learning that design decisions come hard to others was a revelation."*
2. *"The second book is Design of a Computer: the Control Data 6600 by J. E. Thornton, 1970. The insight into the causes of dependencies and their solutions in concurrent processors was revealing."*

*"For the two papers, I still make reference to:*

1. *"Preliminary Discussions of the Logical Design of an Electronic Com-*

*puting Instrument” by Burks, Goldstine, and von Neumann.” I have a photo copy of the original. The insight shown in this paper is truly astounding. And to think that we still follow this model today with concurrency the only major change.”*

2. *“The second paper is the collection of papers found in the IBM Journal of Research and Development, Vol. 11, No. 1, January 1967. “This collection of papers concerned the design of the IBM S360 91. The paper topics include: floating-point arithmetic, the Common Data Bus, memory system design, and a first look at multiple instruction issue (what we call Superscalar today).”*

#### Looking Forward

*“For contemporary reading, I will mention only two books.*

1. *The first is The Supermen, by Charles Murray. This is the story of Seymour Cray and all of his machines, both successes and failures. An important story that emerges is the never-ending battle between: higher clock rates leading to higher circuit density leading to higher power density leading to exotic cooling techniques. This is a battle that is still fought today and will in the future as far as I can tell.”*
2. *“The second book is the monumental work Computer Architecture, Concepts and Evolution by G. Blaauw and Fred. Brooks Jr. This book should be on the desk of an engineer who claims to be a computer architect. The architecture design space that they have pulled together is truly outstanding and is a valuable reference work.”*

#### 1985 Award Winner, John Cocke

**Award Citation: For contributions to the high-performance computer architecture through look ahead, parallelism and pipeline utilization, and to reduced instruction set computer architecture through the exploitation of hardware-software trade-offs and compiler utilization.**

#### Looking Back

*“Here are the thoughts you asked me to send to you. First, in 1957 when I was employed by IBM, there were few Computer Science departments and no major books related to Computer Science. I read a 1946 report prepared for the U.S. Army Ordnance Department by Burks, von Neumann, and Goldstine [1]. There were also available journal articles from the ACM, which contained papers related to such things as error correcting codes, etc. The Bell Systems Technical Journal contained articles by Shannon [2] related to information theory. There was also a book by Richards, [3] which showed how to design adders, etc. Most of the things I learned from people like Fred Brooks and Jim Pomerene, who had worked designing computers at Harvard*

and the Institute of Advanced Study, respectively. Andrew Gleason gave the Stretch Planning Committee some papers related to sorting, which were quite informative. For many years there were no books which would be usable for a Computer Science course, as far as I knew, until Knuth [4] started releasing his books.”

### Looking Forward

“In answer to the question you asked related to books that I feel are appropriate for the new generation of computer architecture, I would recommend Mike Flynn’s [5] *Computer Architecture* book and the *Computer Architecture* book by Fred Brooks and Gerrit Blaauw [6].”

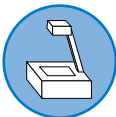
1. Burks, Arthur W., Herman H. Goldstine, and John von Neumann “Preliminary Discussion of the Logical Design of an Electronic Computing Instrument” (Pt. 1. Vol. 1) Report prepared for the U.S. Army Ordnance Department 1946, in A. H. Taub (ed.) *Collected Works of John von Neumann*, Vol. 5, pp. 34-79, *The Macmillan Company*, 1963.
2. Shannon, C. E. “A Mathematical Theory of Communication,” *Bell System Technical Journal*, Vol. 27, pp. 379-423, 623-656, 1948.
3. Richards, R. K., *Arithmetic Operations in Digital Computers*, D. VanNostrand Company, Inc., 1955.
4. Knuth, D. E., *The Art of Computer Programming*, Vol. I, (1968), Vol. II, (1969), Vol. III, (1975), *Addison-Wesley*.
5. Flynn, M. J., *Computer Architecture*, *Jones and Bartlett Publishers*, 1995.
6. Blaauw, G. and Frederick Brooks, *Computer Architecture Concepts and Evolution*, *Addison Wesley Longman*, 1997.”

In this chapter, we reviewed the process of designing and implementing a microprocessor in reasonable detail. In particular, we stated that the development of appropriate representative workloads, the subsequent analysis of the resulting performance measurement data, and the use of that analysis in the design and implementation process is the “cornerstone of many, if not most, architectural design decisions.” We explained the importance of simulation and testing and emphasized the need for a “golden representation” of the microprocessor. We continued with a discussion of the process of designing and implementing a 3D graphics PC platform. In this section, we identified the key components and interconnections central to contemporary platform architectures. This section will prove fundamental to the discussions that occur in Chapters 4, 5, and 6. Lastly, we benefited from the perspectives of several Eckert-Mauchly Award winners regarding what influenced their individual views of computer architecture and what

## SUMMARY OF CHAPTER AND HOW TO PROCEED

they think might prove to be important references in the field in the future.

#### TECHNICAL PRESENTATIONS ON CD-ROM



Two technical presentations by Bruce Shriver that deal with various topics discussed in this chapter might be of interest to the reader: (1) *An Introduction to Computer Architecture* and (2) *The Design and Implementation Process*.

We recommend that you examine the Table of Contents and then page through each of the chapters to get a sense of the specific topics that are covered and the depth to which the material is presented. Similarly, we suggest you page through the cross-references and the combined glossary/index at the rear of the book.

We suggest you have the companion CD-ROM in your CD-ROM reader as you use the hard copy of the book in order to take advantage of the many hyperlinked and cross-reference features of it. Furthermore, during certain sections of the book, you may want to be connected to the Web as a number of the hyperlinks will connect you to relevant Web sites. We should also mention that the figures and tables that appear in each chapter can be printed out from the *Technical Presentations* road map on the CD-ROM on a chapter-by-chapter basis.

Chapter 2 and Chapter 3 should be read sequentially. However, Chapters 4, 5, and 6 can be read in any order and independently of Chapters 2 and 3.