E

Embedded Systems

By Thomas M. Conte
North Carolina State University

Where a calculator on the ENIAC is equipped with 18,000 vacuum tubes and weighs 30 tons, computers in the future may have only 1,000 vacuum tubes and perhaps weigh 1 1/2 tons.

Popular Mechanics
March 1949

E.1 Introduction

Embedded computer systems—computers lodged in other devices where the presence of the computers is not immediately obvious—are the fastest-growing portion of the computer market. These devices range from everyday machines (most microwaves, most washing machines, printers, network switches, and automobiles contain simple to very advanced embedded microprocessors) to handheld digital devices (such as PDAs, cell phones, and music players) to video game consoles and digital set-top boxes. Although in some applications (such as PDAs) the computers are programmable, in many embedded applications the only programming occurs in connection with the initial loading of the application code or a later software upgrade of that application. Thus, the application is carefully tuned for the processor and system. This process sometimes includes limited use of assembly language in key loops, although time-to-market pressures and good software engineering practice restrict such assembly language coding to a fraction of the application.

Compared to desktop and server systems, embedded systems have a much wider range of processing power and cost—from systems containing low-end 8-bit and 16-bit processors that may cost less than a dollar, to those containing full 32-bit microprocessors capable of operating in the 500 MIPS range that cost approximately 10 dollars, to those containing high-end embedded processors that cost hundreds of dollars and can execute several billions of instructions per second. Although the range of computing power in the embedded systems market is very large, price is a key factor in the design of computers for this space. Performance requirements do exist, of course, but the primary goal is often meeting the performance need at a minimum price, rather than achieving higher performance at a higher price.

Embedded systems often process information in very different ways from general-purpose processors. Typically these applications include deadline-driven constraints—so-called *real-time constraints.* In these applications, a particular computation must be completed by a certain time or the system fails (there are other constraints considered real time, discussed in the next subsection).

Embedded systems applications typically involve processing information as *signals.* The lay term "signal" often connotes radio transmission, and that is true for some embedded systems (e.g., cell phones). But a signal may be an image, a motion picture composed of a series of images, a control sensor measurement, and so on. Signal processing requires specific computation that many embedded processors are optimized for. We discuss this in depth below. A wide range of benchmark requirements exist, from the ability to run small, limited code segments to the ability to perform well on applications involving tens to hundreds of thousands of lines of code.

Two other key characteristics exist in many embedded applications: the need to minimize memory and the need to minimize power. In many embedded applications, the memory can be a substantial portion of the system cost, and it is important to optimize memory size in such cases. Sometimes the application is expected to fit

entirely in the memory on the processor chip; other times the application needs to fit in its entirety in a small, off-chip memory. In either case, the importance of memory size translates to an emphasis on code size, since data size is dictated by the application. Some architectures have special instruction set capabilities to reduce code size. Larger memories also mean more power, and optimizing power is often critical in embedded applications. Although the emphasis on low power is frequently driven by the use of batteries, the need to use less expensive packaging (plastic versus ceramic) and the absence of a fan for cooling also limit total power consumption. We examine the issue of power in more detail later in this appendix.

Another important trend in embedded systems is the use of processor cores together with application-specific circuitry—so-called "core plus ASIC" or "system on a chip" (SOC), which may also be viewed as special-purpose multiprocessors (see Section E.4). Often an application's functional and performance requirements are met by combining a custom hardware solution together with software running on a standardized embedded processor core, which is designed to interface to such special-purpose hardware. In practice, embedded problems are usually solved by one of three approaches:

1. The designer uses a combined hardware/software solution that includes some custom hardware and an embedded processor core that is integrated with the custom hardware, often on the same chip.
2. The designer uses custom software running on an off-the-shelf embedded processor.
3. The designer uses a digital signal processor and custom software for the processor. *Digital signal processors* are processors specially tailored for signal-processing applications. We discuss some of the important differences between digital signal processors and general-purpose embedded processors below.

Figure E.1 summarizes these three classes of computing environments and their important characteristics.

Real-Time Processing

Often, the performance requirement in an embedded application is a real-time requirement. A *real-time performance requirement* is one where a segment of the application has an absolute maximum execution time that is allowed. For example, in a digital set-top box the time to process each video frame is limited, since the processor must accept and process the frame before the next frame arrives (typically called *hard real-time systems*). In some applications, a more sophisticated requirement exists: The average time for a particular task is constrained as well as is the number of instances when some maximum time is exceeded. Such approaches (typically called *soft real-time*) arise when it is possible to occasionally miss the time constraint on an event, as long as not too many are missed. Real-time

Feature	Desktop	Server	Embedded
Price of system	$1000–$10,000	$10,000–$10,000,000	$10–$100,000 (including network routers at the high end)
Price of microprocessor module	$100–$1000	$200–$2000 (per processor)	$0.20–$200 (per processor)
Microprocessors sold per year (estimates for 2000)	150,000,000	4,000,000	300,000,000 (32-bit and 64-bit processors only)
Critical system design issues	Price-performance, graphics performance	Throughput, availability, scalability	Price, power consumption, application-specific performance

Figure E.1 A summary of the three computing classes and their system characteristics. Note the wide range in system price for servers and embedded systems. For servers, this range arises from the need for very large-scale multiprocessor systems for high-end transaction processing and Web server applications. For embedded systems, one significant high-end application is a network router, which could include multiple processors as well as lots of memory and other electronics. The total number of embedded processors sold in 2000 is estimated to exceed 1 billion, if you include 8-bit and 16-bit microprocessors. In fact, the largest-selling microprocessor of all time is an 8-bit microcontroller sold by Intel! It is difficult to separate the low end of the server market from the desktop market, since low-end servers—especially those costing less than $5000—are essentially no different from desktop PCs. Hence, up to a few million of the PC units may be effectively servers.

performance tends to be highly application dependent. It is usually measured using kernels either from the application or from a standardized benchmark (see Section E.3).

The construction of a hard real-time system involves three key variables. The first is the rate at which a particular task must occur. Coupled to this are the hardware and software required to achieve that real-time rate. Often, structures that are very advantageous on the desktop are the enemy of hard real-time analysis. For example, branch speculation, cache memories, and so on introduce *uncertainty* into code. A particular sequence of code may execute either very efficiently or very inefficiently, depending on whether the hardware branch predictors and caches "do their jobs." Engineers must analyze code assuming the *worst-case execution time* (WCET). In the case of traditional microprocessor hardware, if one assumes that *all branches are mispredicted* and *all caches miss,* the WCET is overly pessimistic. Thus, the system designer may end up overdesigning a system to achieve a given WCET, when a much less expensive system would have sufficed.

In order to address the challenges of hard real-time systems, and yet still exploit such well-known architectural properties as branch behavior and access locality, it is possible to change how a processor is designed. Consider branch prediction: Although dynamic branch prediction is known to perform far more accurately than static "hint bits" added to branch instructions, the behavior of static hints is much more predictable. Furthermore, although caches perform better than software-managed on-chip memories, the latter produces predictable memory latencies. In some embedded processors, caches can be converted into software-managed on-chip memories via *line locking*. In this approach, a cache line can be locked in the cache so that it cannot be replaced until the line is unlocked

E.2 Signal Processing and Embedded Applications: The Digital Signal Processor

A digital signal processor (DSP) is a special-purpose processor optimized for executing digital signal processing algorithms. Most of these algorithms, from time-domain filtering (e.g., infinite impulse response and finite impulse response filtering), to convolution, to transforms (e.g., fast Fourier transform, discrete cosine transform), to even forward error correction (FEC) encodings, all have as their kernel the same operation: a multiply-accumulate operation. For example, the discrete Fourier transform has the form:

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn} \quad \text{where} \quad W_N^{kn} = e^{j\frac{2\pi kn}{N}} = \cos\left(2\pi\frac{kn}{N}\right) + j\sin\left(2\pi\frac{kn}{N}\right)$$

The discrete cosine transform is often a replacement for this because it does not require complex number operations. Either transform has as its core the *sum* of a *product*. To accelerate this, DSPs typically feature special-purpose hardware to perform *multiply-accumulate* (MAC). A MAC instruction of "MAC A,B,C" has the semantics of "A = A + B * C." In some situations, the performance of this operation is so critical that a DSP is selected for an application based solely upon its MAC operation throughput.

DSPs often employ *fixed-point* arithmetic. If you think of integers as having a binary point to the right of the least-significant bit, fixed point has a binary point just to the right of the sign bit. Hence, fixed-point data are fractions between −1 and +1.

Example Here are three simple 16-bit patterns:

0100 0000 0000 0000

0000 1000 0000 0000

0100 1000 0000 1000

What values do they represent if they are two's complement integers? Fixedpoint numbers?

Answer Number representation tells us that the *i*th digit to the left of the binary point represents 2^{i-1} and the *i*th digit to the right of the binary point represents 2^{-i}. First assume these three patterns are integers. Then the binary point is to the far right, so they represent 2^{14}, 2^{11}, and (2^{14}+ 2^{11}+ 2^3), or 16,384, 2048, and 18,440.

Fixed point places the binary point just to the right of the sign bit, so as fixed point these patterns represent 2^{-1}, 2^{-4}, and (2^{-1} + 2^{-4} + 2^{-12}). The fractions are 1/2, 1/16, and (2048 + 256 + 1)/4096 or 2305/4096, which represents about 0.50000, 0.06250, and 0.56274. Alternatively, for an *n*-bit two's complement,

fixed-point number we could just divide the integer presentation by 2^{n-1} to derive the same results:

$$16,384/32,768 = 1/2,\ 2048/32,768 = 1/16,\ \text{and}\ 18,440/32,768 = 2305/4096.$$

Fixed point can be thought of as a low-cost floating point. It doesn't include an exponent in every word and doesn't have hardware that automatically aligns and normalizes operands. Instead, fixed point relies on the DSP programmer to keep the exponent in a separate variable and ensure that each result is shifted left or right to keep the answer aligned to that variable. Since this exponent variable is often shared by a set of fixed-point variables, this style of arithmetic is also called *blocked floating point*, since a block of variables has a common exponent.

To support such manual calculations, DSPs usually have some registers that are wider to guard against round-off error, just as floating-point units internally have extra guard bits. Figure E.2 surveys four generations of DSPs, listing data sizes and width of the accumulating registers. Note that DSP architects are not bound by the powers of 2 for word sizes. Figure E.3 shows the size of data operands for the TI TMS320C55 DSP.

In addition to MAC operations, DSPs often also have operations to accelerate portions of communications algorithms. An important class of these algorithms revolve around encoding and decoding *forward error correction codes*—codes in which extra information is added to the digital bit stream to guard against errors in transmission. A code of rate *m*/*n* has *m* information bits for (*m* + *n*) check bits. So, for example, a 1/2 rate code would have 1 information bit per every 2 bits. Such codes are often called *trellis codes* because one popular graphical flow diagram of

Generation	Year	Example DSP	Data width	Accumulator width
1	1982	TI TMS32010	16 bits	32 bits
2	1987	Motorola DSP56001	24 bits	56 bits
3	1995	Motorola DSP56301	24 bits	56 bits
4	1998	TI TMS320C6201	16 bits	40 bits

Figure E.2 Four generations of DSPs, their data width, and the width of the registers that reduces round-off error.

Data size	Memory operand in operation	Memory operand in data transfer
16 bits	89.3%	89.0%
32 bits	10.7%	11.0%

Figure E.3 Size of data operands for the TMS320C55 DSP. About 90% of operands are 16 bits. This DSP has two 40-bit accumulators. There are no floating-point operations, as is typical of many DSPs, so these data are all fixed-point integers.

their encoding resembles a garden trellis. A common algorithm for decoding trellis codes is due to Viterbi. This algorithm requires a sequence of compares and selects in order to recover a transmitted bit's true value. Thus DSPs often have compare-select operations to support Viterbi decode for FEC codes.

To explain DSPs better, we will take a detailed look at two DSPs, both produced by Texas Instruments. The TMS320C55 series is a DSP family targeted toward battery-powered embedded applications. In stark contrast to this, the TMS VelociTI 320C6x series is a line of powerful, eight-issue VLIW processors targeted toward a broader range of applications that may be less power sensitive.

The TI 320C55

At one end of the DSP spectrum is the TI 320C55 architecture. The C55 is optimized for low-power, embedded applications. Its overall architecture is shown in Figure E.4. At the heart of it, the C55 is a seven-staged pipelined CPU. The stages are outlined below:

- *Fetch stage* reads program data from memory into the instruction buffer queue.
- *Decode stage* decodes instructions and dispatches tasks to the other primary functional units.
- *Address stage* computes addresses for data accesses and branch addresses for program discontinuities.
- *Access 1/Access 2 stages* send data read addresses to memory.
- *Read stage* transfers operand data on the B bus, C bus, and D bus.
- *Execute stage* executes operation in the A unit and D unit and performs writes on the E bus and F bus.

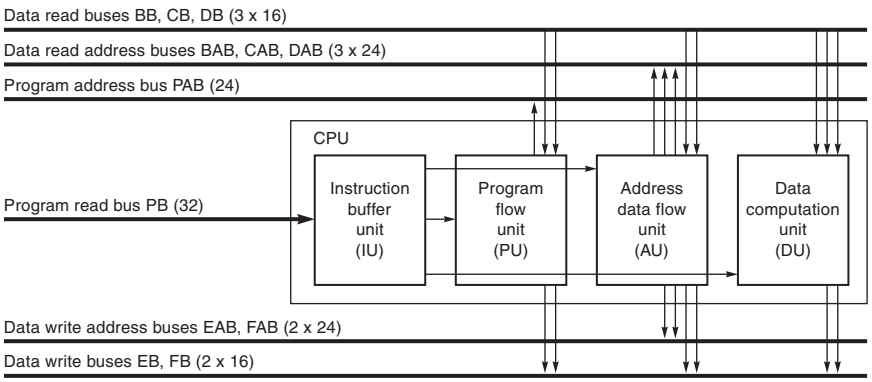

Figure E.4 Architecture of the TMS320C55 DSP. The C55 is a seven-stage pipelined processor with some unique instruction execution facilities. (Courtesy Texas Instruments.)

The C55 pipeline performs pipeline hazard detection and will stall on write after read (WAR) and read after write (RAW) hazards.

The C55 does have a 24 KB instruction cache, but it is configurable to support various workloads. It may be configured to be two-way set associative, direct-mapped, or as a "ramset." This latter mode is a way to support hard realtime applications. In this mode, blocks in the cache cannot be replaced.

The C55 also has advanced power management. It allows dynamic power management through software-programmable "idle domains." Blocks of circuitry on the device are organized into these idle domains. Each domain can operate normally or can be placed in a low-power idle state. A programmer-accessible Idle Control Register (ICR) determines which domains will be placed in the idle state when the execution of the next `IDLE` instruction occurs. The six domains are CPU, direct memory access (DMA), peripherals, clock generator, instruction cache, and external memory interface. When each domain is in the idle state, the functions of that particular domain are not available. However, in the peripheral domain, each peripheral has an Idle Enable bit that controls whether or not the peripheral will respond to the changes in the idle state. Thus, peripherals can be individually configured to idle or remain active when the peripheral domain is idled.

Since the C55 is a DSP, the central feature is its MAC units. The C55 has two MAC units, each comprised of a 17-bit by 17-bit multiplier coupled to a 40-bit dedicated adder. Each MAC unit performs its work in a single cycle; thus, the C55 can execute two MACs per cycle in full pipelined operation. This kind of capability is critical for efficiently performing signal processing applications. The C55 also has a compare, select, and store unit (CSSU) for the add/compare section of the Viterbi decoder.

The TI 320C6x

In stark contrast to the C55 DSP family is the high-end Texas Instruments VelociTI 320C6x family of processors. The C6x processors are closer to traditional very long instruction word (VLIW) processors because they seek to exploit the high levels of instruction-level parallelism (ILP) in many signal processing algorithms. Texas Instruments is not alone in selecting VLIW for exploiting ILP in the embedded space. Other VLIW DSP vendors include Ceva, StarCore, Philips/TriMedia, and STMicroelectronics. Why do these vendors favor VLIW over superscalar? For the embedded space, code compatibility is less of a problem, and so new applications can be either hand tuned or recompiled for the newest generation of processor. The other reason superscalar excels on the desktop is because the compiler cannot predict memory latencies at compile time. In embedded, however, memory latencies are often much more predictable. In fact, hard real-time constraints force memory latencies to be statically predictable. Of course, a superscalar would also perform well in this environment with these constraints, but the extra hardware to dynamically schedule instructions is both wasteful in terms of precious chip area and in terms of power consumption. Thus VLIW is a natural choice for high-performance embedded.

The C6x family employs different pipeline depths depending on the family member. For the C64x, for example, the pipeline has 11 stages. The first four stages of the pipeline perform instruction fetch, followed by two stages for instruction decode, and finally four stages for instruction execution. The overall architecture of the C64x is shown below in Figure E.5.

The C6x family's execution stage is divided into two parts, the left or "1" side and the right or "2" side. The L1 and L2 units perform logical and arithmetic operations. D units in contrast perform a subset of logical and arithmetic operations but also perform memory accesses (loads and stores). The two M units perform multiplication and related operations (e.g., shifts). Finally the S units perform comparisons, branches, and some SIMD operations (see the next subsection for a detailed explanation of SIMD operations). Each side has its own 32-entry, 32-bit register file (the A file for the 1 side, the B file for the 2 side). A side may access the other side's registers, but with a 1- cycle penalty. Thus, an instruction executing on side 1 may access B5, for example, but it will take 1- cycle extra to execute because of this.

VLIWs are traditionally very bad when it comes to code size, which runs contrary to the needs of embedded systems. However, the C6x family's approach "compresses" instructions, allowing the VLIW code to achieve the same density as equivalent RISC (reduced instruction set computer) code. To do so, instruction fetch is carried out on an "instruction packet," shown in Figure E.6. Each instruction has a *p* bit that specifies whether this instruction is a member of the current

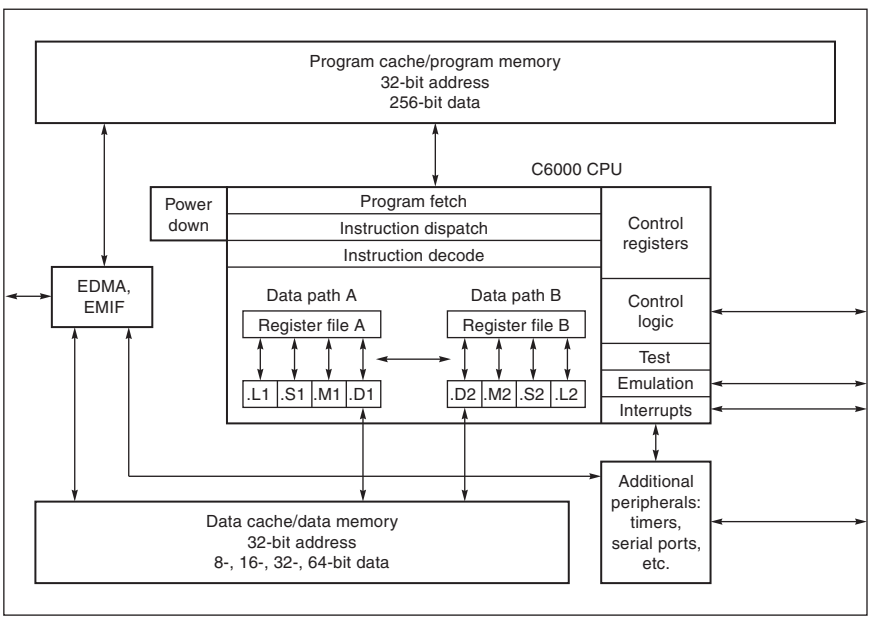

Figure E.5 Architecture of the TMS320C64x family of DSPs. The C6x is an eight-issue traditional VLIW processor. (Courtesy Texas Instruments.)

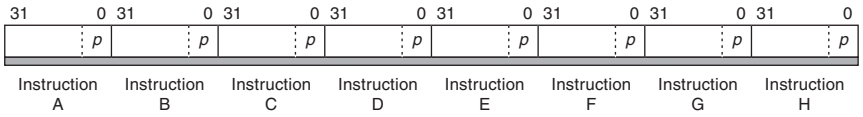

Figure E.6 Instruction packet of the TMS320C6x family of DSPs. The *p* bits determine whether an instruction begins a new VLIW word or not. If the *p* bit of instruction *i* is 1, then instruction *i* + 1 is to be executed in parallel with (in the same cycle as) instruction *i*. If the *p* bit of instruction *i* is 0, then instruction *i* + 1 is executed in the cycle after instruction *i*. (Courtesy Texas Instruments.)

VLIW word or the next VLIW word (see the figure for a detailed explanation). Thus, there are now no NOPs that are needed for VLIW encoding.

Software pipelining is an important technique for achieving high performance in a VLIW. But software pipelining relies on each iteration of the loop having an identical schedule to all other iterations. Because conditional branch instructions disrupt this pattern, the C6x family provides a means to conditionally execute instructions using *predication.* In predication, the instruction performs its work. But when it is done executing, an additional register, for example A1, is checked. If A1 is zero, the instruction does not write its results. If A1 is nonzero, the instruction proceeds normally. This allows simple if-then and if-then-else structures to be collapsed into straight-line code for software pipelining.

Media Extensions

There is a middle ground between DSPs and microcontrollers: *media extensions.* These extensions add DSP-like capabilities to microcontroller architectures at relatively low cost. Because media processing is judged by human perception, the data for multimedia operations are often much narrower than the 64-bit data word of modern desktop and server processors. For example, floating-point operations for graphics are normally in single precision, not double precision, and often at a precision less than is required by IEEE 754. Rather than waste the 64-bit arithmetic-logical units (ALUs) when operating on 32-bit, 16-bit, or even 8-bit integers, multimedia instructions can operate on several narrower data items at the same time. Thus, a *partitioned add* operation on 16-bit data with a 64-bit ALU would perform four 16-bit adds in a single clock cycle. The extra hardware cost is simply to prevent carries between the four 16-bit partitions of the ALU. For example, such instructions might be used for graphical operations on pixels. These operations are commonly called *single-instruction multiple-data* (SIMD) or *vector* instructions.

Most graphics multimedia applications use 32-bit floating-point operations. Some computers double peak performance of single-precision, floating-point operations; they allow a single instruction to launch two 32-bit operations on operands found side by side in a double-precision register. The two partitions must be insulated to prevent operations on one half from affecting the other. Such floating-point operations are called *paired single operations.* For example, such an operation

might be used for graphical transformations of vertices. This doubling in performance is typically accomplished by doubling the number of floating-point units, making it more expensive than just suppressing carries in integer adders.

Figure E.7 summarizes the SIMD multimedia instructions found in several recent computers.

DSPs also provide operations found in the first three rows of Figure E.7, but they change the semantics a bit. First, because they are often used in real-time applications, there is not an option of causing an exception on arithmetic overflow (otherwise it could miss an event); thus, the result will be used no matter what the inputs. To support such an unyielding environment, DSP architectures use *saturating arithmetic*: If the result is too large to be represented, it is set to the largest representable number, depending on the sign of the result. In contrast, two's complement arithmetic can add a small positive number to a large positive.

Instruction category	Alpha MAX	HP PA-RISC MAX2	Intel Pentium MMX	PowerPC AltiVec	SPARC VIS
Add/subtract		4H	8B, 4H, 2W	16B, 8H, 4W	4H, 2W
Saturating add/subtract		4H	8B, 4H	16B, 8H, 4W	
Multiply			4H	16B, 8H	
Compare	8B (>=)		8B, 4H, 2W (=, >)	16B, 8H, 4W (=, >, > =, <, <=)	4H, 2W (=, not =, >, <=)
Shift right/left		4H	4H, 2W	16B, 8H, 4W	
Shift right arithmetic		4H		16B, 8H, 4W	
Multiply and add				8H	
Shift and add (saturating)		4H			
AND/OR/XOR	8B, 4H, 2W	8B, 4H, 2W	8B, 4H, 2W	16B, 8H, 4W	8B, 4H, 2W
Absolute difference	8B			16B, 8H, 4W	8B
Maximum/minimum	8B, 4W			16B, 8H, 4W	
Pack ($2n$ bits $\rightarrow$ n bits)	2W $\rightarrow$ 2B, 4H $\rightarrow$ 4B	2*4H $\rightarrow$ 8B	4H $\rightarrow$ 4B, 2W $\rightarrow$ 2H	4W $\rightarrow$ 4B, 8H $\rightarrow$ 8B	2W $\rightarrow$ 2H, 2W $\rightarrow$ 2B, 4H $\rightarrow$ 4B
Unpack/merge	2B $\rightarrow$ 2W, 4B $\rightarrow$ 4H		2B $\rightarrow$ 2W, 4B $\rightarrow$ 4H	4B $\rightarrow$ 4W, 8B $\rightarrow$ 8H	4B $\rightarrow$ 4H, 2*4B $\rightarrow$ 8B
Permute/shuffle		4H		16B, 8H, 4W	

Figure E.7 Summary of multimedia support for desktop processors. Note the diversity of support, with little in common across the five architectures. All are fixed-width operations, performing multiple narrow operations on either a 64-bit or 128-bit ALU. B stands for byte (8 bits), H for half word (16 bits), and W for word (32 bits). Thus, 8B means an operation on 8 bytes in a single instruction. Note that AltiVec assumes a 128-bit ALU, and the rest assume 64 bits. Pack and unpack use the notation 2*2W to mean 2 operands each with 2 words. This table is a simplification of the full multimedia architectures, leaving out many details. For example, HP MAX2 includes an instruction to calculate averages, and SPARC VIS includes instructions to set registers to constants. Also, this table does not include the memory alignment operation of AltiVec, MAX, and VIS.

E.3 Embedded Benchmarks

It used to be the case just a couple of years ago that in the embedded market, many manufacturers quoted Dhrystone performance, a benchmark that was criticized and given up by desktop systems more than 20 years ago! As mentioned earlier, the enormous variety in embedded applications, as well as differences in performance requirements (hard real time, soft real time, and overall cost-performance), make the use of a single set of benchmarks unrealistic. In practice, many designers of embedded systems devise benchmarks that reflect their application, either as kernels or as stand-alone versions of the entire application.

For those embedded applications that can be characterized well by kernel performance, the best standardized set of benchmarks appears to be a new benchmark set: the EDN Embedded Microprocessor Benchmark Consortium (or EEMBC, pronounced "embassy"). The EEMBC benchmarks fall into six classes (called "subcommittees" in the parlance of EEMBC): automotive/industrial, consumer, telecommunications, digital entertainment, networking (currently in its second version), and office automation (also the second version of this subcommittee). Figure E.8 shows the six different application classes, which include 50 benchmarks.

Although many embedded applications are sensitive to the performance of small kernels, remember that often the overall performance of the entire application (which may be thousands of lines) is also critical. Thus, for many embedded systems, the EMBCC benchmarks can only be used to partially assess performance.

Benchmark type ("subcommittee")	Number of kernels	Example benchmarks
Automotive/industrial	16	6 microbenchmarks (arithmetic operations, pointer chasing, memory performance, matrix arithmetic, table lookup, bit manipulation), 5 automobile control benchmarks, and 5 filter or FFT benchmarks
Consumer	5	5 multimedia benchmarks (JPEG compress/decompress, filtering, and RGB conversions)
Telecommunications	5	Filtering and DSP benchmarks (autocorrelation, FFT, decoder, encoder)
Digital entertainment	12	MP3 decode, MPEG-2 and MPEG-4 encode and decode (each of which is applied to five different datasets), MPEG Encode Floating Point, 4 benchmark tests for common cryptographic standards and algorithms (AES, DES, RSA, and Huffman decoding for data decompression), and enhanced JPEG and color-space conversion tests
Networking version 2	6	IP Packet Check (borrowed from the RFC1812 standard), IP Reassembly, IP Network Address Translator (NAT), Route Lookup, OSPF, Quality of Service (QOS), and TCP
Office automation version 2	6	Ghostscript, text parsing, image rotation, dithering, Bézier

Figure E.8 The EEMBC benchmark suite, consisting of 50 kernels in six different classes. See *www.eembc.org* for more information on the benchmarks and for scores.

Power Consumption and Efficiency as the Metric

Cost and power are often at least as important as performance in the embedded market. In addition to the cost of the processor module (which includes any required interface chips), memory is often the next most costly part of an embedded system. Unlike a desktop or server system, most embedded systems do not have secondary storage; instead, the entire application must reside in either FLASH or DRAM. Because many embedded systems, such as PDAs and cell phones, are constrained by both cost and physical size, the amount of memory needed for the application is critical. Likewise, power is often a determining factor in choosing a processor, especially for battery-powered systems.

EEMBC EnergyBench provides data on the amount of energy a processor consumes while running EEMBC's performance benchmarks. An EEMBC-certified Energymark score is an optional metric that a device manufacturer may choose to supply in conjunction with certified scores for device performance as a way of indicating a processor's efficient use of power and energy. EEMBC has standardized on the use of National Instruments' LabVIEW graphical development environment and data acquisition hardware to implement EnergyBench.

Figure E.9 shows the relative performance per watt of typical operating power. Compare this figure to Figure E.10, which plots raw performance, and notice how different the results are. The NEC VR 4122 has a clear advantage in performance per watt, but is the second-lowest performing processor! From the viewpoint of power consumption, the NEC VR 4122, which was designed for battery-based systems, is the big winner. The IBM PowerPC displays efficient use of power to achieve its high performance, although at 6 W typical, it is probably not suitable for most battery-based devices.

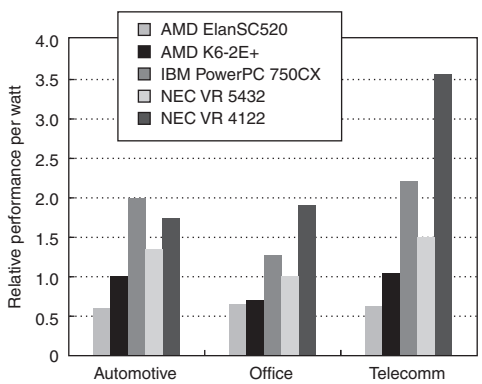

Figure E.9 Relative performance per watt for the five embedded processors. The power is measured as typical operating power for the processor and does not include any interface chips.

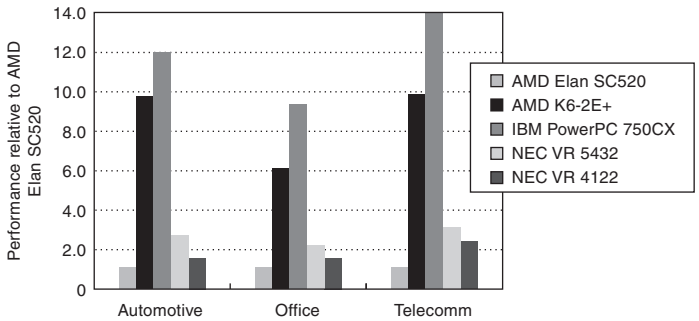

Figure E.10 Raw performance for the five embedded processors. The performance is presented as relative to the performance of the AMD ElanSC520.

E.4 Embedded Multiprocessors

Multiprocessors are now common in server environments, and several desktop multiprocessors are available from vendors, such as Sun, Compaq, and Apple. In the embedded space, a number of special-purpose designs have used customized multiprocessors, including the Sony PlayStation 2 (see Section E.5).

Many special-purpose embedded designs consist of a general-purpose programmable processor or DSP with special-purpose, finite-state machines that are used for stream-oriented I/O. In applications ranging from computer graphics and media processing to telecommunications, this style of special-purpose multiprocessor is becoming common. Although the interprocessor interactions in such designs are highly regimented and relatively simple—consisting primarily of a simple communication channel—because much of the design is committed to silicon, ensuring that the communication protocols among the input/output processors and the general-purpose processor are correct is a major challenge in such designs.

More recently, we have seen the first appearance, in the embedded space, of embedded multiprocessors built from several general-purpose processors. These multiprocessors have been focused primarily on the high-end telecommunications and networking market, where scalability is critical. An example of such a design is the MXP processor designed by empowerTel Networks for use in voice-over-IP systems. The MXP processor consists of four main components:

- An interface to serial voice streams, including support for handling jitter
- Support for fast packet routing and channel lookup
- A complete Ethernet interface, including the MAC layer
- Four MIPS32 R4000-class processors, each with its own cache (a total of 48 KB or 12 KB per processor)

The MIPS processors are used to run the code responsible for maintaining the voice-over-IP channels, including the assurance of quality of service, echo cancellation, simple compression, and packet encoding. Since the goal is to run as many independent voice streams as possible, a multiprocessor is an ideal solution.

Because of the small size of the MIPS cores, the entire chip takes only 13.5 M transistors. Future generations of the chip are expected to handle more voice channels, as well as do more sophisticated echo cancellation, voice activity detection, and more sophisticated compression.

Multiprocessing is becoming widespread in the embedded computing arena for two primary reasons. First, the issues of binary software compatibility, which plague desktop and server systems, are less relevant in the embedded space. Often software in an embedded application is written from scratch for an application or significantly modified (note that this is also the reason VLIW is favored over superscalar in embedded instruction-level parallelism). Second, the applications often have natural parallelism, especially at the high end of the embedded space. Examples of this natural parallelism abound in applications such as a settop box, a network switch, a cell phone (see Section E.7) or a game system (see Section E.5). The lower barriers to use of thread-level parallelism together with the greater sensitivity to die cost (and hence efficient use of silicon) are leading to widespread adoption of multiprocessing in the embedded space, as the application needs grow to demand more performance.

E.5 Case Study: The Emotion Engine of the Sony PlayStation 2

Desktop computers and servers rely on the memory hierarchy to reduce average access time to relatively static data, but there are embedded applications where data are often a continuous stream. In such applications there is still spatial locality, but temporal locality is much more limited.

To give another look at memory performance beyond the desktop, this section examines the microprocessor at the heart of the Sony PlayStation 2. As we will see, the steady stream of graphics and audio demanded by electronic games leads to a different approach to memory design. The style is high bandwidth via many dedicated independent memories.

Figure E.11 shows a block diagram of the Sony PlayStation 2 (PS2). Not surprisingly for a game machine, there are interfaces for video, sound, and a DVD player. Surprisingly, there are two standard computer I/O buses, USB and IEEE 1394, a PCMCIA slot as found in portable PCs, and a modem. These additions show that Sony had greater plans for the PS2 beyond traditional games. Although it appears that the I/O processor (IOP) simply handles the I/O devices and the game console, it includes a 34 MHz MIPS processor that also acts as the emulation computer to run games for earlier Sony PlayStations. It also connects to a standard PC audio card to provide the sound for the games.

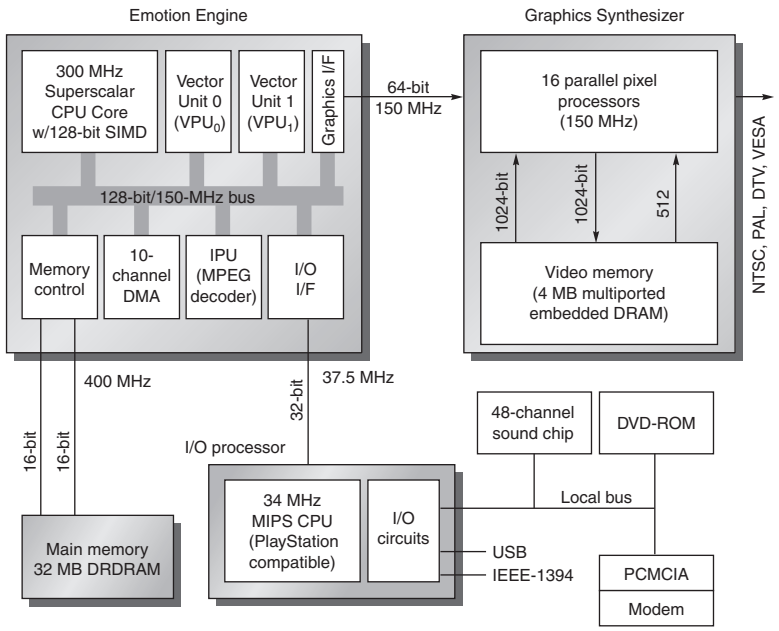

Figure E.11 Block diagram of the Sony PlayStation 2. The 10 DMA channels orchestrate the transfers between all the small memories on the chip, which when completed all head toward the Graphics Interface so as to be rendered by the Graphics Synthesizer. The Graphics Synthesizer uses DRAM on chip to provide an entire frame buffer plus graphics processors to perform the rendering desired based on the display commands given from the Emotion Engine. The embedded DRAM allows 1024-bit transfers between the pixel processors and the display buffer. The Superscalar CPU is a 64-bit MIPS III with two-instruction issue, and comes with a two-way, set associative, 16 KB instruction cache; a two-way, set associative, 8 KB data cache; and 16 KB of scratchpad memory. It has been extended with 128-bit SIMD instructions for multimedia applications (see Section E.2). Vector Unit 0 is primarily a DSP-like coprocessor for the CPU (see Section E.2), which can operate on 128-bit registers in SIMD manner between 8 bits and 32 bits per word. It has 4 KB of instruction memory and 4 KB of data memory. Vector Unit 1 has similar functions to VPU0, but it normally operates independently of the CPU and contains 16 KB of instruction memory and 16 KB of data memory. All three units can communicate over the 128-bit system bus, but there is also a 128-bit dedicated path between the CPU and VPU0 and a 128-bit dedicated path between VPU1 and the Graphics Interface. Although VPU0 and VPU1 have identical microarchitectures, the differences in memory size and units to which they have direct connections affect the roles that they take in a game. At 0.25-micron line widths, the Emotion Engine chip uses 13.5M transistors and is 225 mm^2, and the Graphics Synthesizer is 279 mm^2. To put this in perspective, the Alpha 21264 microprocessor in 0.25-micron technology is about 160 mm^2 and uses 15M transistors. (This figure is based on Figure 1 in "Sony's Emotionally Charged Chip," *Microprocessor Report* 13:5.)

Thus, one challenge for the memory system of this embedded application is to act as source or destination for the extensive number of I/O devices. The PS2 designers met this challenge with two PC800 (400 MHz) DRDRAM chips using two channels, offering 32 MB of storage and a peak memory bandwidth of 3.2 GB/sec.

What's left in the figure are basically two big chips: the Graphics Synthesizer and the Emotion Engine.

The Graphics Synthesizer takes rendering commands from the Emotion Engine in what are commonly called *display lists.* These are lists of 32-bit commands that tell the renderer what shape to use and where to place them, plus what colors and textures to fill them.

This chip also has the highest bandwidth portion of the memory system. By using embedded DRAM on the Graphics Synthesizer, the chip contains the full video buffer *and* has a 2048-bit-wide interface so that pixel filling is not a bottleneck. This embedded DRAM greatly reduces the bandwidth demands on the DRDRAM. It illustrates a common technique found in embedded applications: separate memories dedicated to individual functions to inexpensively achieve greater memory bandwidth for the entire system.

The remaining large chip is the Emotion Engine, and its job is to accept inputs from the IOP and create the display lists of a video game to enable 3D video transformations in real time. A major insight shaped the design of the Emotion Engine: Generally, in a racing car game there are foreground objects that are constantly changing and background objects that change less in reaction to the events, although the background can be most of the screen. This observation led to a split of responsibilities.

The CPU works with VPU0 as a tightly coupled coprocessor, in that every VPU0 instruction is a standard MIPS coprocessor instruction, and the addresses are generated by the MIPS CPU. VPU0 is called a vector processor, but it is similar to 128-bit SIMD extensions for multimedia found in several desktop processors (see Section E.2).

VPU1, in contrast, fetches its own instructions and data and acts in parallel with CPU/VPU0, acting more like a traditional vector unit. With this split, the more flexible CPU/VPU0 handles the foreground action and the VPU1 handles the background. Both deposit their resulting display lists into the Graphics Interface to send the lists to the Graphics Synthesizer.

Thus, the programmers of the Emotion Engine have three processor sets to choose from to implement their programs: the traditional 64-bit MIPS architecture including a floating-point unit, the MIPS architecture extended with multimedia instructions (VPU0), and an independent vector processor (VPU1). To accelerate MPEG decoding, there is another coprocessor (Image Processing Unit) that can act independent of the other two.

With this split of function, the question then is how to connect the units together, how to make the data flow between units, and how to provide the memory bandwidth needed by all these units. As mentioned earlier, the Emotion Engine designers chose many dedicated memories. The CPU has a 16 KB scratch pad memory (SPRAM) in addition to a 16 KB instruction cache and an 8 KB data cache. VPU0 has a 4 KB instruction memory and a 4 KB data memory, and VPU1 has a 16 KB instruction memory and a 16 KB data memory. Note that these are four *memories*, not caches of a larger memory elsewhere. In each memory the latency is just 1 clock cycle. VPU1 has more memory than VPU0 because it creates the bulk of the display lists and because it largely acts independently.

The programmer organizes all memories as two double buffers, one pair for the incoming DMA data and one pair for the outgoing DMA data. The programmer then uses the various processors to transform the data from the input buffer to the output buffer. To keep the data flowing among the units, the programmer next sets up the 10 DMA channels, taking care to meet the real-time deadline for realistic animation of 15 frames per second.

Figure E.12 shows that this organization supports two main operating modes: serial, where CPU/VPU0 acts as a preprocessor on what to give VPU1 for it to create for the Graphics Interface using the scratchpad memory as the buffer, and parallel, where both the CPU/VPU0 and VPU1 create display lists. The display lists and the Graphics Synthesizer have multiple context identifiers to distinguish the parallel display lists to produce a coherent final image.

All units in the Emotion Engine are linked by a common 150 MHz, 128-bit-wide bus. To offer greater bandwidth, there are also two dedicated buses: a 128-bit path between the CPU and VPU0 and a 128-bit path between VPU1 and the Graphics Interface. The programmer also chooses which bus to use when setting up the DMA channels.

Looking at the big picture, if a server-oriented designer had been given the problem, we might see a single common bus with many local caches and cache-coherent mechanisms to keep data consistent. In contrast, the PlayStation 2 followed the tradition of embedded designers and has at least nine distinct memory modules. To keep the data flowing in real time from memory to the display, the PS2 uses dedicated memories, dedicated buses, and DMA channels. Coherency is the responsibility of the programmer, and, given the continuous flow from main memory to the graphics interface and the real-time requirements, programmer-controlled coherency works well for this application.

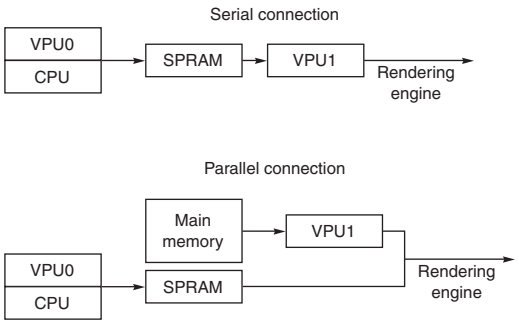

Figure E.12 Two modes of using Emotion Engine organization. The first mode divides the work between the two units and then allows the Graphics Interface to properly merge the display lists. The second mode uses CPU/VPU0 as a filter of what to send to VPU1, which then does all the display lists. It is up to the programmer to choose between serial and parallel data flow. SPRAM is the scratchpad memory.

E.6 Case Study: Sanyo VPC-SX500 Digital Camera

Another very familiar embedded system is a digital camera. Here we consider the Sanyo VPC-SX500. When powered on, the microprocessor of the camera first runs diagnostics on all components and writes any error messages to the liquid crystal display (LCD) on the back of the camera. This camera uses a 1.8-inch low-temperature polysilicon thin-film transistor (TFT) color LCD. When a photographer takes a picture, he first holds the shutter halfway so that the microprocessor can take a light reading. The microprocessor then keeps the shutter open to get the necessary light, which is captured by a charge-coupled device (CCD) as red, green, and blue pixels. The CCD is a 1/2-inch, 1360 × 1024-pixel, progressive-scan chip. The pixels are scanned out row by row; passed through routines for white balance, color, and aliasing correction; and then stored in a 4 MB frame buffer. The next step is to compress the image into a standard format, such as JPEG, and store it in the removable Flash memory. The photographer picks the compression, in this camera called either *fine* or *normal*, with a compression ratio of 10 to 20 times. A 512 MB Flash memory can store at least 1200 fine-quality compressed images or approximately 2000 normal-quality compressed images. The microprocessor then updates the LCD display to show that there is room for one less picture.

Although the previous paragraph covers the basics of a digital camera, there are many more features that are included: showing the recorded images on the color LCD display, sleep mode to save battery life, monitoring battery energy, buffering to allow recording a rapid sequence of uncompressed images, and, in this camera, video recording using MPEG format and audio recording using WAV format.

The electronic brain of this camera is an embedded computer with several special functions embedded on the chip [Okada et al. 1999]. Figure E.13 shows the block diagram of a chip similar to the one in the camera. As mentioned in Section E.1, such chips have been called *systems on a chip* (SOCs) because they essentially integrate into a single chip all the parts that were found on a small printed circuit board of the past. A SOC generally reduces size and lowers power compared to less integrated solutions. Sanyo claims their SOC enables the camera to operate on half the number of batteries and to offer a smaller form factor than competitors' cameras. For higher performance, it has two buses. The 16-bit bus is for the many slower I/O devices: SmartMedia interface, program and data memory, and DMA. The 32-bit bus is for the SDRAM, the signal processor (which is connected to the CCD), the Motion JPEG encoder, and the NTSC/PAL encoder (which is connected to the LCD). Unlike desktop microprocessors, note the large variety of I/O buses that this chip must integrate. The 32-bit RISC MPU is a proprietary design and runs at 28.8 MHz, the same clock rate as the buses. This 700 mW chip contains 1.8M transistors in a 10.5 × 10.5 mm die implemented using a 0.35-micron process.

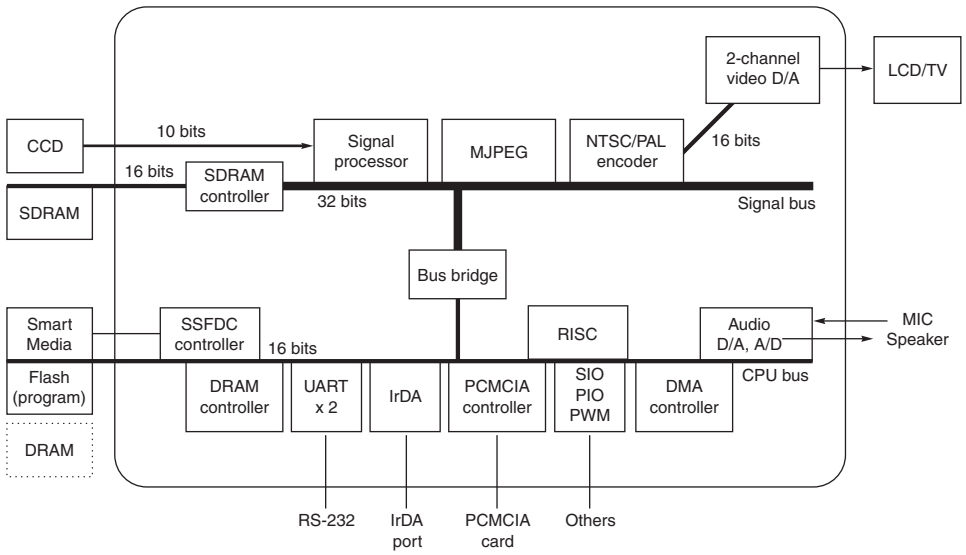

Figure E.13 The system on a chip (SOC) found in Sanyo digital cameras. This block diagram, found in Okada et al. [1999], is for the predecessor of the SOC in the camera described in the text. The successor SOC, called *Super Advanced IC*, uses three buses instead of two, operates at 60 MHz, consumes 800 mW, and fits 3.1M transistors in a 10.2 × 10.2 mm die using a 0.35-micron process. Note that this embedded system has twice as many transistors as the state-of-the-art, high-performance microprocessor in 1990! The SOC in the figure is limited to processing 1024 × 768 pixels, but its successor supports 1360 × 1024 pixels.

E.7 Case Study: Inside a Cell Phone

Although gaming consoles and digital cameras are familiar embedded systems, today the most familiar embedded system is the cell phone. In 1999, there were 76 million cellular subscribers in the United States, a 25% growth rate from the year before. That growth rate is almost 35% per year worldwide, as developing countries find it much cheaper to install cellular towers than copper-wire-based infrastructure. Thus, in many countries, the number of cell phones in use exceeds the number of wired phones in use.

Not surprisingly, the cellular handset market is growing at 35% per year, with about 280 million cellular phone handsets sold worldwide in 1999. To put that in perspective, in the same year sales of personal computers were 120 million. These numbers mean that tremendous engineering resources are available to improve cell phones, and cell phones are probably leaders in engineering innovation per cubic inch [Grice and Kanellos 2000].

Before unveiling the anatomy of a cell phone, let's try a short introduction to wireless technology.

Background on Wireless Networks

Networks can be created out of thin air as well as out of copper and glass, creating *wireless networks.* Much of this section is based on a report from the National Research Council [1997].

A radio wave is an electromagnetic wave propagated by an antenna. Radio waves are modulated, which means that the sound signal is superimposed on the stronger radio wave that carries the sound signal, and hence is called the *carrier signal.* Radio waves have a particular wavelength or frequency: They are measured either as the length of the complete wave or as the number of waves per second. Long waves have low frequencies, and short waves have high frequencies. FM radio stations transmit on the band of 88 MHz to 108 MHz using frequency modulations (FM) to record the sound signal.

By tuning in to different frequencies, a radio receiver can pick up a specific signal. In addition to AM and FM radio, other frequencies are reserved for citizens band radio, television, pagers, air traffic control radar, Global Positioning System, and so on. In the United States, the Federal Communications Commission decides who gets to use which frequencies and for what purpose.

The *bit error rate* (BER) of a wireless link is determined by the received signal power, noise due to interference caused by the receiver hardware, interference from other sources, and characteristics of the channel. Noise is typically proportional to the radio frequency bandwidth, and a key measure is the *signal-to-noise ratio* (SNR) required to achieve a given BER. Figure E.14 lists more challenges for wireless communication.

Typically, wireless communication is selected because the communicating devices are mobile or because wiring is inconvenient, which means the wireless network must rearrange itself dynamically. Such rearrangement makes routing

Challenge	Description	Impact
Path loss	Received power divided by transmitted power; the radio must overcome signal-to-noise ratio (SNR) of noise from interference. Path loss is exponential in distance and depends on interference if it is above 100 meters.	1 W transmit power, 1 GHz transmit frequency, 1 Mbit/sec data rate at 10^{-7} BER, distance between radios can be 728 meters in free space vs. 4 meters in a dense jungle.
Shadow fading	Received signal blocked by objects, buildings outdoors, or walls indoors; increase power to improve received SNR. It depends on the number of objects and their dielectric properties.	If transmitter is moving, need to change transmit power to ensure received SNR in region.
Multipath fading	Interference between multiple versions of signal that arrive at different times, determined by time between fastest signal and slowest signal relative to signal bandwidth.	900 MHz transmit frequency signal power changes every 30 cm.
Interference	Frequency reuse, adjacent channel, narrow band interference.	Requires filters, spread spectrum.

Figure E.14 Challenges for wireless communication.

more challenging. A second challenge is that wireless signals are not protected and hence are subject to mutual interference, especially as devices move. Power is another challenge for wireless communication, both because the devices tend to be battery powered and because antennas radiate power to communicate and little of it reaches the receiver. As a result, raw bit error rates are typically a thousand to a million times higher than copper wire.

There are two primary architectures for wireless networks: *base station* architectures and *peer-to-peer* architectures. Base stations are connected by landlines for longer-distance communication, and the mobile units communicate only with a single local base station. Peer-to-peer architectures allow mobile units to communicate with each other, and messages hop from one unit to the next until delivered to the desired unit. Although peer-to-peer is more reconfigurable, base stations tend to be more reliable since there is only one hop between the device and the station. *Cellular telephony*, the most popular example of wireless networks, relies on radio with base stations.

Cellular systems exploit exponential path loss to reuse the same frequency at spatially separated locations, thereby greatly increasing the number of customers served. Cellular systems will divide a city into nonoverlapping hexagonal cells that use different frequencies if nearby, reusing a frequency only when cells are far enough apart so that mutual interference is acceptable.

At the intersection of three hexagonal cells is a base station with transmitters and antennas that is connected to a switching office that coordinates handoffs when a mobile device leaves one cell and goes into another, as well as accepts and places calls over landlines. Depending on topography, population, and so on, the radius of a typical cell is 2 to 10 miles.

The Cell Phone

Figure E.15 shows the components of a radio, which is the heart of a cell phone. Radio signals are first received by the antenna, amplified, passed through a mixer, then filtered, demodulated, and finally decoded. The antenna acts as the interface between the medium through which radio waves travel and the electronics of the transmitter or receiver. Antennas can be designed to work best in particular directions, giving both transmission and reception directional properties. Modulation encodes information in the amplitude, phase, or frequency of the signal to increase its robustness under impaired conditions. Radio transmitters go through the same steps, just in the opposite order.

Originally, all components were analog, but over time most were replaced by digital components, requiring the radio signal to be converted from analog to digital. The desire for flexibility in the number of radio bands led to software routines replacing some of these functions in programmable chips, such as digital signal processors. Because such processors are typically found in mobile devices, emphasis is placed on performance per joule to extend battery life, performance per square millimeter of silicon to reduce size and cost, and bytes per task to reduce memory size.

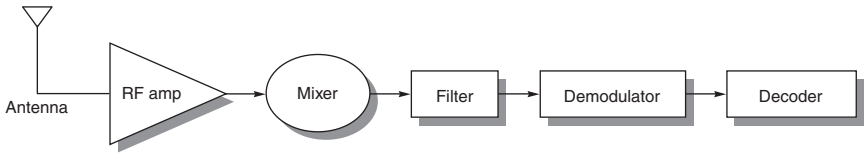

Figure E.15 A radio receiver consists of an antenna, radio frequency amplifier, mixer, filters, demodulator, and decoder. A mixer accepts two signal inputs and forms an output signal at the sum and difference frequencies. Filters select a narrower band of frequencies to pass on to the next stage. Modulation encodes information to make it more robust. Decoding turns signals into information. Depending on the application, all electrical components can be either analog or digital. For example, a car radio is all analog components, but a PC modem is all digital except for the amplifier. Today analog silicon chips are used for the RF amplifier and first mixer in cellular phones.

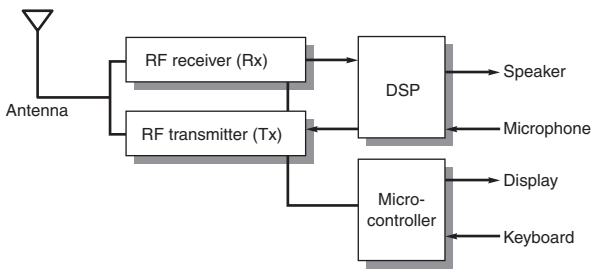

Figure E.16 Block diagram of a cell phone. The DSP performs the signal processing steps of Figure E.15, and the microcontroller controls the user interface, battery management, and call setup. (Based on Figure 1.3 of Groe and Larson [2000].)

Figure E.16 shows the generic block diagram of the electronics of a cell phone handset, with the DSP performing the signal processing and the microcontroller handling the rest of the tasks. Cell phone handsets are basically mobile computers acting as a radio. They include standard I/O devices—keyboard and LCD display—plus a microphone, speaker, and antenna for wireless networking. Battery efficiency affects sales, both for standby power when waiting for a call and for minutes of speaking.

When a cell phone is turned on, the first task is to find a cell. It scans the full bandwidth to find the strongest signal, which it keeps doing every seven seconds or if the signal strength drops, since it is designed to work from moving vehicles. It then picks an unused radio channel. The local switching office registers the cell phone and records its phone number and electronic serial number, and assigns it a voice channel for the phone conversation. To be sure the cell phone got the right channel, the base station sends a special tone on it, which the cell phone sends back to acknowledge it. The cell phone times out after 5 seconds if it doesn't hear the supervisory tone, and it starts the process all over again. The original base station makes a handoff request to the incoming base station as the signal strength drops offs.

To achieve a two-way conversation over radio, frequency bands are set aside for each direction, forming a frequency pair or *channel.* The original cellular base stations transmitted at 869.04 to 893.97 MHz (called the *forward path*), and cell phones transmitted at 824.04 to 848.97 MHz (called t*he reverse path*), with the frequency gap to keep them from interfering with each other. Cells might have had between 4 and 80 channels. Channels were divided into setup channels for call setup and voice channels to handle the data or voice traffic.

The communication is done digitally, just like a modem, at 9600 bits/sec. Since wireless is a lossy medium, especially from a moving vehicle, the handset sends each message five times. To preserve battery life, the original cell phones typically transmit at two signal strengths—0.6 W and 3.0 W—depending on the distance to the cell. This relatively low power not only allows smaller batteries and thus smaller cell phones, but it also aids frequency reuse, which is the key to cellular telephony.

Figure E.17 shows a circuit board from a Nokia digital phone, with the components identified. Note that the board contains two processors. A Z-80 microcontroller is responsible for controlling the functions of the board, I/O with the keyboard and display, and coordinating with the base station. The DSP handles all signal compression and decompression. In addition there are dedicated chips for analog-to-digital and digital-to-analog conversion, amplifiers, power management, and RF interfaces.

In 2001, a cell phone had about 10 integrated circuits, including parts made in exotic technologies like gallium arsinide and silicon germanium as well as standard CMOS. The economics and desire for flexibility have shrunk this to just a few chips. However, these SOCs still contain a separate microcontroller and DSP, with code implementing many of the functions just described.

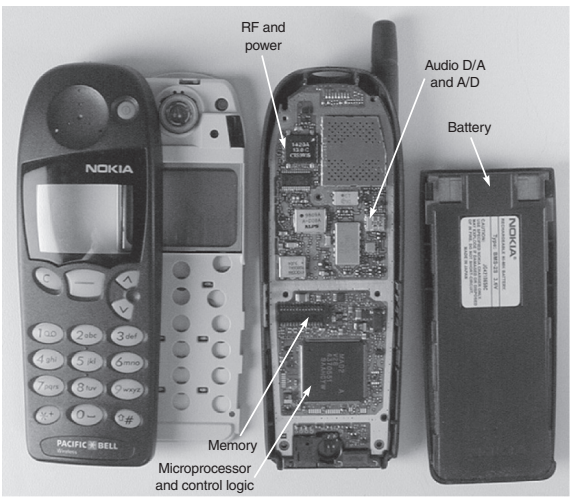

Figure E.17 Circuit board from a Nokia cell phone. (Courtesy HowStuffWorks, Inc.)

Cell Phone Standards and Evolution

Improved communication speeds for cell phones were developed with multiple standards. *Code division multiple access* (CDMA), as one popular example, uses a wider radio frequency band for a path than the original cell phones, called *advanced mobile phone service* (AMPS), a mostly analog system. The wider frequency makes it more difficult to block and is called *spread spectrum.* Other standards are *time division multiple access* (TDMA) and *global system for mobile communication* (GSM). These second-generation standards—CDMA, GSM, and TDMA—are mostly digital.

The big difference for CDMA is that all callers share the same channel, which operates at a much higher rate, and it then distinguishes the different calls by encoding each one uniquely. Each CDMA phone call starts at 9600 bits/sec; it is then encoded and transmitted as equal-sized messages at 1.25 Mbits/sec. Rather than send each signal five times as in AMPS, each bit is stretched so that it takes 11 times the minimum frequency, thereby accommodating interference and yet successful transmission. The base station receives the messages, and it separates them into the separate 9600 bit/sec streams for each call.

To enhance privacy, CDMA uses pseudorandom sequences from a set of 64 predefined codes. To synchronize the handset and base station so as to pick a common pseudorandom seed, CDMA relies on a clock from the Global Positioning System, which continuously transmits an accurate time signal. By carefully selecting the codes, the shared traffic sounds like random noise to the listener. Hence, as more users share a channel there is more noise, and the signal-to-noise ratio gradually degrades. Thus, the capacity of the CDMA system is a matter of taste, depending upon the sensitivity of the listener to background noise.

In addition, CDMA uses speech compression and varies the rate of data transferred depending upon how much activity is going on in the call. Both these techniques preserve bandwidth, which allows for more calls per cell. CDMA must regulate power carefully so that signals near the cell tower do not overwhelm those from far away, with the goal of all signals reaching the tower at about the same level. The side benefit is that CDMA handsets emit less power, which both helps battery life and increases capacity when users are close to the tower.

Thus, compared to AMPS, CDMA improves the capacity of a system by up to an order of magnitude, has better call quality, has better battery life, and enhances users' privacy. After considerable commercial turmoil, there is a new third-generation standard called *International Mobile Telephony 2000* (IMT-2000), based primarily on two competing versions of CDMA and one TDMA. This standard may lead to cell phones that work anywhere in the world.

E.8 Concluding Remarks

Embedded systems are a very broad category of computing devices. This appendix has shown just some aspects of this. For example, the TI 320C55 DSP is a relatively "RISC-like" processor designed for embedded applications, with very

fine-tuned capabilities. On the other end of the spectrum, the TI 320C64x is a very high-performance, eight-issue VLIW processor for very demanding tasks. Some processors must operate on battery power alone; others have the luxury of being plugged into line current. Unifying all of these is a need to perform some level of signal processing for embedded applications. Media extensions attempt to merge DSPs with some more general-purpose processing abilities to make these processors usable for signal processing applications. We examined several case studies, including the Sony PlayStation 2, digital cameras, and cell phones. The PS2 performs detailed three-dimensional graphics, whereas a cell phone encodes and decodes signals according to elaborate communication standards. But both have system architectures that are very different from general-purpose desktop or server platforms. In general, architectural decisions that seem practical for general-purpose applications, such as multiple levels of caching or out-of-order superscalar execution, are much less desirable in embedded applications. This is due to chip area, cost, power, and real-time constraints. The programming model that these systems present places more demands on both the programmer and the compiler for extracting parallelism.