With such a wide range of computer platforms with vastly varying performance and I/O capabilities, in this chapter we will define some key attributes of embedded systems. A key characteristic of all embedded systems is that they are designed to perform a specific task or function. The software developed for the platform is specific to the function of the overall device. In many cases the compute capability of the device is invisible to the end user; the function provided by the device is all that matters. In contrast to the rapidly growing application-centric ecosystem developing in the smartphone ecosystem, End users of embedded systems do not generally install application software on such devices (although they may have limited ability to upgrade the software).

Embedded systems cover the entire landscape of devices and products. As shown in Figure 2.1, many consumer devices are embedded compute platforms. Internet-enabled TV, Blu-ray™ players, video game consoles, streaming Internet TV devices, and printers are just a few consumer devices that are characterized by a fairly high level of performance.

On the communications side, embedded devices are deployed throughout the infrastructure at every level. Consider the varied devices that work in unison to enable your Internet experience. For example, when visiting a web page, TCP/IP packets are generated on your device and sent out on the wireless interface (e.g., Wi-Fi). The packets are picked up by the wireless router that is connected to your broadband access device (such as an ADSL or cable modem). Once the packets leave the modem, they travel along the Internet infrastructure. On the infrastructure side, your packets will be routed through a digital subscriber line access multiplexer (DSLAM), then on to Internet network IP routers. The packets are then routed to the web server hosting environment, where the packets pass through firewalls and web load balancers and finally to the web server. Figure 2.1 shows the overall flow.

Another pervasive application of embedded control systems is in the automated manufacturing of goods. These industrial applications require a wide range of control elements with compute capabilities including motor controllers, actuators, sensors, advanced programmable logic controllers (PLCs), robotics, and control panels with user interfaces (HMI).

In the automotive domain, the capabilities being offered are increasing at an aggressive pace. What has traditionally been a media playback system (radio, CD, DVD, MP3) and simple navigation capability has been rapidly evolving into a highly graphics intensive platform, providing realistic 3D cityscape navigation while connected to the Internet, offering streaming media capability, location-based services, live map updates, and remote car diagnostics. In addition, the domain of driver assistance is becoming increasingly important, with systems such as collision avoidance (braking assist), lane departure warnings, driver alertness indications, night vision, object identification, and trajectory warnings (such as pedestrians). There are also a number of government-sponsored efforts to create intelligent transport systems (see http://www.its.dot.gov/).
Embedded systems can range from a simple stand-alone device to a chassis of networked cards to a system composed of many separate networked embedded elements. They all work collaboratively to achieve the objectives of the overall system. Such systems operate largely autonomously once set up. A chassis-based system is a common configuration for telecommunications and data communications.
equipment. In some cases the chassis form factor is defined by a standard such as the Advanced Telecommunications Computing Architecture (ATCA), an example of which is shown in Figure 2.2.

In some cases, embedded systems are referred to as cyber-physical systems, where the system exercises significant physical interactions in the real world. Much of the critical infrastructure in our society is controlled by embedded systems, termed supervisory control and data acquisition (SCADA) systems. Given our increasing dependence on connectivity and increased security threats to such systems, the security features of embedded devices are also becoming a critical attribute to comprehend.

Embedded systems often have specific real-time constraints that must be adhered to. In many cases, when an embedded system interfaces with the real world (physically), there are tight real-time responses demanded. These domains can place design constraints on all aspects of the system design, from the peripheral interfaces and quality of service in the system on chip to the operating system selection and the application design.

FIGURE 2.2
Chassis-Based Router.
We have just touched on a tiny fraction of the many varied roles for embedded computing; the
overall spectrum covers an enormous dynamic range in terms of performance, power, connectivity, and
usage models. However, the trend is toward higher compute and connectivity demands for many
embedded systems.

This chapter illustrates the high-level concepts with discussion of a particular device/platform.
When introducing each component/subsystem/capability, we will explicitly reference the chapter that
describes it in detail. The system description shall serve as a reference to the remainder of the book.

EMBEDDED PLATFORM CHARACTERISTICS

This section describes some critical attributes of embedded systems and identifies some of the dynamic
range for the capabilities identified.

Central Processing Unit (CPU)

As the examples of embedded systems in Figure 2.1 show, it is clear that there is an enormous range of
compute capabilities in embedded systems.

The number of bits assigned to registers within the integer pipeline 8, 16, 32, or 64 is the first CPU
characteristic to focus on. Although mainstream PC CPUs have been running 32-bit for many years,
and more recently 64-bit is the standard desktop platform, and there remains a very large number of 8-
bit and 16-bit CPUs in many simple control applications. For example, Atmel makes a product line of
8-bit controllers known as Atmel AVR Microcontrollers. These parts run up to the 20–30 MHz range
and include integrated flash, RAM, and peripherals. In the 16-bit space, PIC microcontrollers are
widely used (predominantly) from MicroChip. These platforms are very low power and can often be
run from an AA battery for extended periods. In the case of 8-bit controllers, connectivity options such
as IP over Ethernet require additional devices to handle the communications stack, while on 16-bit
controllers limited low-speed connectivity is offered (for example, a 10-Mb Ethernet connectivity). In
many cases these devices do not run any operating system, although a simple real-time operating
system (RTOS) such as FreeRTOS is available.

While 8-bit and 16-bit devices will remain in use for quite some time, it’s clear that there is also
a large need for 32-bit devices in the embedded space. In many cases the vendors offering 8-/16-bit
controllers have also introduced 32-bit variants of their product line. There are also ARM licensees
introducing ARM M3–based microcontroller devices.

The devices mentioned above are generally classified as microcontrollers. Although they occupy
a large portion of the embedded space, we will not focus on this class of device in this book. Instead,
we are going to focus on a generally higher performance class of device, with sophisticated
connectivity and graphics that, generally speaking, relies on a RTOS or a full-featured operating
system such as an embedded Linux distribution. As the expectations placed on embedded systems
grow exponentially, the ability to ensure that the software scales with these expectations is critical. In
such cases full-featured operating systems such as Linux offer compelling capabilities.

These higher-performance devices are generally known as embedded microprocessors, as distinct
from microcontrollers, and are dominated by 32-bit CPU architectures at this time. This class of
processors is used in all of the embedded examples shown in Figure 2.1. In some cases, the processor
provides a supervisory control function for the system; in other cases, it performs the entire application and data path workload.

Given the wide range of applications, the clock speed of the processor is an important consideration; the range of an embedded microprocessor system is from the low hundreds of megahertz (200 MHz) to over 1 GHz. Clock speed is an important attribute because it is the first order indicator of performance; however, the overall architecture of the processor and the inclusion of caches (and their size) are critical aspects that contribute to overall system performance. In fact, the faster the processor speed, the more important it is to have some form of fast memory close to the processor (with cache being the easiest to take advantage of).

Another important characteristic is the level of parallelism offered by the processor. Instruction-level parallelism, where the CPU performs a number of operations simultaneously, is quite common. A wide range of processor microarchitecture techniques can improve instruction-level parallelism, such as instruction pipelining, superscalar execution, and out-of-order execution. Processors also offer instructions that are single instructions, multiple data (SIMD) to optimize the execution of algorithms that exhibit data-level parallelism. Some processors also allow for thread-level parallelism, known as symmetric multithreading, where with the addition of a relatively small amount of logic, the processor presents two separate logical cores (to the operating system). While one logical core is blocked, perhaps waiting for a memory fetch, the other logical core makes use of the arithmetic logic units.

Many embedded applications require the use of floating-point arithmetic. If floating-point operations are not an insignificant portion of the application workload, the process should include a hardware-based floating-point unit. If the floating-point unit is provided as a hardware function, it is ideal if it conforms to the IEEE Standard for Floating-Point Arithmetic (IEEE 754); otherwise, you have to become very familiar with the deviations from such a standard.

As a reference data point in terms of capabilities, the current Intel Atom core in the E6XX services processor is dual superscalar, with in-order execution and data-level parallelism supported by Intel Supplemental Streaming SIMD Extensions 3 (Intel SSSE3, IEEE 754 compliant hardware), has a hardware floating-point unit, and supports two hardware threads per core. It is offered with speeds from 600 MHz to 1.6 GHz. It provides 32-kB four-way level 1 instruction cache, 32-kB six-way level 1 data cache, and a 512-kB level 2 (unified) cache.

Integration Level
The increasing demand for lower-cost, higher-density platforms and smaller form factors has driven the need to increase the level of integration for each of the devices that makes up the embedded platform. Initially, embedded platforms were composed of separate discrete components. The processor was a separate component with just a memory bus interface, and all peripherals were attached to this bus.

As integration levels increase, more and more logic is added to the processor die, creating families of application-specific service processors. The term system on chip (SOC) is often used to describe these highly integrated processors. These SOCs include much of the logic and interfaces that are required for a range of specific target applications. The silicon vendors that develop these SOC devices often create families of SOCs all using the same processor core, but with a wide range of integrated capabilities.
SOCs integrate capabilities to connect the SOC to external memory devices and nonvolatile storage devices using glue-less interfaces. Glue-less is a term used to indicate that there is no additional logic needed to connect the two devices, for example, connect the SOC to DDR DRAM.

In addition to attaching to memory devices, an SOC provides segment- or application-specific interfaces. Examples of integrated devices are general purpose input/output pins, interfaces such as Ethernet, USB, PCIe, serial ports, I2C, expansion parallel buses, and integrated display controllers. Many of these devices interface to nonvolatile storage such as NOR Flash via Serial Peripheral Interconnect (SPI), and native bus interface types are described in Chapter 4.

As a general rule, these integrated items are predominantly digital logic elements. Because we need to add analog capabilities, features such as flash memory and digital/analog converters are common, but these capabilities require special features of the silicon manufacturing process. As you review the capabilities of embedded microprocessors you may notice that, as the performance range of the processor increases, there is less likelihood that the device will include such analog capabilities. This is due to the tension in creating a silicon process that is optimized for both high-speed digital logic designs and the mixed signal analog domain. This boundary is always changing but is an important dynamic in what capabilities are viable for integration into any SOC device.

**Power Consumption**

The power consumed by the devices is measured in many different ways. The traditional power quoted for embedded devices provides the typical power consumption of the device. This is measured by running an application on the processor that exercises a representative portion of the I/O capabilities. The current into the device is measured using the current and supply voltage. The power can be calculated using $\text{Power} = \text{Current} \times \text{voltage}$. In many cases there are actually several different voltage rails in use, so this must be summed across all power supplies.

Many silicon vendors also provide the total device power figure (TDP). TDP is the maximum amount of power the cooling system is required to dissipate. This does not mean that the system needs active power cooling, such as a fan. In many embedded cases a heat sink is all that is required. The TDP is used as part of an overall thermal design that must ensure that the CPU/SOC does not overheat. There can be a considerable delta between the average power dissipation of a system and the figures quoted for TDP. TDP is and must be an extremely conservative figure. When choosing a system where power is an important attribute, make sure the datasheet comparisons are comparing the same type of figure.

Power figures are often highly dependent on the activity level of the system. Many processors systems have very low power idle states. In order to develop efficient power aware systems, it is often beneficial for an embedded application to group work into bursts of activity with the processor running at full clock rate followed by periods of processor idle states. For example, duty cycles of much less than 10% (full activity 10%, sleep/idle for 90% of the time) are not uncommon in some sensor type applications.

The power of the system must also include loss in conversion of power from the primary single supply into the power rails needed by all components in the system; an efficient conversion of 60–80% would be normal.

A platform power measurement is not just the power of the processor/SOC; other aspects of the system can contribute significantly to the power. Rich, colorful displays, for example, are often one of
highest power-consuming devices. Although we are not focused on mobile cell phone embedded systems in this book, Figure 2.3 is a snapshot from an Android phone (battery information).

Overall, platform power comparisons are nuanced, and an awareness of the overall application and how it behaves over time as well as the activity level of other devices on the platform is critical in assessing the overall actual power of the system. We discuss power analysis and optimization in Chapter 9, “Power Optimization.”

Form Factor

The form factors for embedded systems are as diverse as the embedded use cases themselves. A large number of embedded systems are composed of a single PCB and are often called single-board computers (SBCs). The platform provides a power connect for a single input voltage (for example, 12V) and provides connectors for devices such as mass storage SATA/SDIO, USB, and displays. These connectors are not necessarily those found in standards-based platforms such as the PC (this is especially the case for the display). Single-board platforms are the most cost-effective way to produce an embedded platform design for a specific target use case, but the platform is not readily upgradeable over time.
In many cases, considerable effort is expended in designing a core compute module that can be employed in a number of different product lines with varying I/O or interface capabilities. In such cases it is often more cost-effective to develop a single daughterboard that provides the key compute capability; this board can be mounted on a motherboard where the application-specific embedded capabilities (and I/O) are instantiated.

Given that many people and companies have a need for a generic compute module, there are a number of standard form factors and connector formats for such modules. Also given that the connector and form factor is standardized, there is a rich ecosystem that develops such boards for reuse by other companies. In this case, you just have to create the daughterboard required for your target embedded application, or often you can use a generic carrier motherboard provided by the same vendors.

The PC/104 Consortium (http://www.pc104.org) has developed a number of standards. The first PC/104 is a stacked format where the compute module can be stacked on top of a number of different I/O carrier boards. The latest standard specification included PCIe between boards. The specification can be found at http://www.pc104.org/pci104_Express_specs.php.

- Compact, 3.6 x 3.8 inch (90 x 96 mm) module size
- Self-stacking: expands without backplanes or card cages
- Rugged, reliable connectors: reliable in harsh environments
- Four-corner mounting holes: resistance to shock and vibration
- Fully PC-compatible: reduced development costs and time to market

The PC/104 organization also standardized the ECPI and EBC form factors, widely used by many vendors (not IA-32 CPUs, despite the PC moniker). COM Express is a computer-on-module (COM) form factor.
Another standard is the Qseven standard (http://www.qseven-standard.org/). An Intel Atom E600 Q7 module is shown in Figure 2.4. The module contains all the required logic for a complete IA-32 Intel architecture system.

Many of these standards are supported by module and SBC manufacturers such as Kontron (http://us.kontron.com/products/computeronmodules/) and Congatec AG. Many vendors provide multiple versions of each module with different compute processors used from Intel, ARM-based SOCs, and others.

**Expansion**

An attribute often sacrificed in designing embedded systems is the ability to expand hardware capabilities over time. Given that the platform has a specific purpose, the designer has the ability to dimension the platform for the specific usage. The DRAM and nonvolatile memory are usually soldered down on the platform. There are generally no expansion slots to add additional hardware.

Software capabilities are, however, often added over time. Perhaps all the software features were not ready at the time of product launch, or the marketing team came up with a killer new feature—either way you now have to ensure that you can add these capabilities to a system that cannot be expanded. If you are fortunate, the platform designers shipped the system with a margin of DRAM and flash. As a general guideline, use no more than 70% of the installed DRAM/flash at the time the product is released if you plan on adding software features over time. Leaving a margin for additional features is a difficult embedded systems trade-off. There is a cost associated with providing additional resources that may or may not actually be used. In many cases, it is best to provide the margin in the first release of the product and subsequently create versions with a reduced margin over time. These are known as cost reduction cycles and are quite common. Naturally, CPU headroom is another key consideration in ensuring that sufficient headroom is left to add software features over time.

**Application-Specific Hardware**

Given the tremendous range of embedded applications, there is a likelihood that there aren’t any system-on-chip devices that perfectly suit your needs. In some cases it is simply not economically viable for a silicon provider to add such a capability due to a limited market size; in other cases, the intellectual property is not available, and in fact the capability that has been added is a part of your company’s “secret sauce.”

To meet this need, you can add capabilities by adding a field-programmable gate array (FPGA) or by developing an application-specific integrated circuit (ASIC). The trade-offs between these two options usually have to do with cost and level of risk. The development cost of an ASIC will be higher, but the cost to manufacture will be lower (this is volume-dependent), and vice versa for FPGAs, depending on the volume of the production.

PCI Express (PCIe) is the interface of choice used to connect between the SOC and an FPGA/ASIC. FPGA vendors such as Altera (http://www.altera.com) and Xilinx (http://www.xilinx.com) provide PCIe capabilities for the FPGAs. This PCIe is usually a “hard” capability and does not use up any of the programmable resources you would need for your application.

Figure 2.5 shows multiple IP blocks behind the PCIe endpoint. It is important (for software) that separate capabilities be exposed as separate PCIe functions. Hardware aspects of PCIe are discussed in Chapter 4 and software aspects in Chapter 8.
Naturally, you could extend the platform through other interfaces such as USB, SDIO, I2C, or a general expansion bus, but PCIe provides excellent general purpose interconnect capabilities, and most operating systems have a very well-defined device driver model for PCIe devices.

**Certification**

An important aspect of many embedded systems is that they form part of a system that itself must be certified. The certification requirements are usually industry specific, but a range of cross-industry certifications may apply to the system. For example, if your device uses wireless connectivity that uses licensed radio spectrum it must be certified by the Federal Communications Commission for sale in the United States. Your company may not have RF expertise; in this case we recommend using pre-certified wireless modules since your system must still be certified but it takes less time. It is important to be aware of such certification considerations early in the design of your embedded system, and you should plan for the certification timeline in your product release schedule.

There are many safety and security standards that may be applicable to the industry you work in, such as multilevel secure (MLS), Safety Integrity Level (SIL), and Federal Information Processing Standards Publications (FIPs). Many of these standards influence every detail of the embedded system you are developing and the processes you use to develop it. A brief mention of such considerations is provided here, but these topics are outside the scope of this book.

**Reliability/Availability**

Many embedded systems must remain running for significant amounts of time without any intervention (often years). In such cases small bugs that may not be apparent in desktop type applications can become debilitating over time. A simple small memory leak (failure to free previously allocated memory once it is no longer needed) may not affect the operation of a program that is executed once a day, but could consume all available heap memory over time in an embedded system. It is important to validate and test your system by running your system for several days and reviewing such resources. It’s not a good idea to assume your system will be restarted regularly.
In many cases you must provide a handler to catch fatal errors (such as no more memory) and trigger a system restart; this is a failsafe option but does result in a loss of system availability during restart. Additionally, it’s good to partition the system so that subsystems can be restarted without having to restart the entire system. For example, if a Linux system is providing packet routing in the kernel but an application has become unresponsive, a better strategy is to restart the application and not the entire system. Features such as control groups (cgroups) in Linux provide an excellent mechanism to improve the robustness of systems that must have high availability (not just embedded systems).

There are also features such as error-correcting-code (ECC) memory, which automatically corrects single bit errors and detects multiple bit errors in memory.

**User Interfaces**

There is tremendous variability in the user interface requirements of embedded systems. There are two general classes of embedded devices, **headed** (those providing a display) and **headless** (those without a display).

For headless devices, the system must still be managed or controlled by a user, and the device usually provides a command console or simple web interface to the device, as shown in Figure 2.6.

However, many devices are now headed devices with display capabilities. There are many display types, from a simple two-line monochrome LCD screen to an HD 1920 × 1080 pixel widescreen...
display. When users directly interact with the display-based embedded device there are increasingly higher expectations placed on the user experience, in terms of ease of use, high quality graphics, and touch screen controls.

Chapter 10 provides detail on embedded graphics and multimedia acceleration. Figure 2.7 shows a rich user interface for an embedded automotive head unit based on Meego.

**Connectivity**

A dramatic increase in the level of connectivity for all embedded devices is expected. There are many industry reports that claim that there will be about 15 billion Internet-attached devices by 2015 (http://www.bbc.co.uk/news/technology-13613536). Many of these will be embedded devices. The embedded platforms must support the latest IP stacks in particular. Finally, a transition to IPv6 is likely.

From a physical connectivity view, many wired and wireless interfaces are used. Ethernet is the ubiquitous wired interface available on many platforms. For wireless interfaces, 802.11 and Wi-Fi are the most prevalent. Other wireless technologies such as Bluetooth and those based on IEEE 802.15.4 such as Zigbee are provided, depending on the application.

Wide area wireless technologies such as those based on 3G/4G cellular technologies are also an important growing area of connectivity for remote managing of devices for which mobility is important, such as vehicle fleet management, but not exclusively for such cases—they are also employed, for example, in vending machines or ATMs.

**Security**

The security of embedded systems is becoming an increasingly critical aspect. Security covers a very broad range of topics. In many embedded cases today, there is no attention paid to security aspects,
which leaves systems vulnerable to compromise and attack. The security of embedded systems used in key infrastructure elements within countries is becoming a particular focus of governments (http://www.truststc.org/scada/).

You may think this does not apply to your system, but there are many examples of compromised embedded systems. In one such case the DNS settings on wireless routers were updated (by a Windows-based worm) (http://voices.washingtonpost.com/securityfix/2008/06/malware_silently_alters_wirele_1.html). There is also a much more sophisticated and widely published case known as Stuxnet, which is perhaps the first known malware that was designed to propagate and then target a specific Supervisory Control and Data Acquisition system (SCADA). The malware propagated through systems and only activated itself when it identified a particular SCADA configuration. The malware then altered the control functions in the programmable logic controller (PLC) to physically destroy the equipment. This is the earliest (at this time) known case of a malware causing a cyber-physical attack.

No matter how secure you believe your platform is, you should assume that you have released a platform with vulnerabilities. Embedded systems are often deployed and never updated, even if vulnerabilities are later detected. It is critical that embedded devices have an active life cycle, where security-related updates are pushed to the devices just as they are in the desktop environment. Even if you believe your software to be without vulnerabilities, it is rare that you have written all the code. For example, you can track Linux security vulnerabilities at http://cve.mitre.org. The http://www.linuxsecurity.com web site also has many tips on avoiding common bugs that can lead to serious vulnerabilities.

There is also considerable industry effort in developing white list–based systems to lock down the activity of the embedded system. This will reduce the likelihood of malware being able to attach to the device, but can put a validation burden on the development of system.

In addition to the security of the platform and designing software to reduce the number of vulnerabilities, there is also a class of content security known as Digital Rights Management (DRM), which is required on devices that present media that has specific licensing constraints (such as Blu-ray movies) or streaming media content such as that provided by Netflix. In Chapter 15 we provide some insights into this topic.

**SUMMARY**

This chapter has provided insight into the overall landscape and characteristics of embedded systems. The remainder of the book will delve into detail on many different aspects of such embedded devices. We believe that, overall, embedded devices are becoming more sophisticated, and connectivity (local and Internet) is an increasingly important attribute of the system. Where specific examples are needed we will use the Intel Atom platform to illustrate our point. However, in most cases the points being made are applicable to a more general embedded context.
This page intentionally left blank