I

Introduction to Embedded Systems

Introduction to Embedded Systems

The field of embedded systems is wide and varied, and it is difficult to pin down exact definitions or descriptions. However, Chapter 1 introduces a useful model that can be applied to any embedded system. This model is introduced as a means for the reader to understand the major components that make up different types of electronic devices, regardless of their complexity or differences. Chapter 2 introduces and defines the common standards adhered to when building an embedded system. Because this book is an overview of embedded systems architecture, covering every possible standards-based component that could be implemented is beyond its scope. Therefore, significant examples of current standards-based components were selected, such as networking and Java, to demonstrate how standards define major components in an embedded system. The intention is for the reader to be able to use the methodology behind the model, standards, and real-world examples to understand any embedded system, and to be able to apply any other standard to an embedded system's design.

A Systems Engineering Approach to Embedded Systems Design

● ●

In This Chapter

▶ *Define embedded system*
▶ *Introduce the design process*
▶ *Define an embedded systems architecture*
▶ *Discuss the impact of architecture*
▶ *Summarize the remaining sections of the book*

● ●

1.1 What Is an Embedded System?

An embedded system is an applied computer system, as distinguished from other types of computer systems such as personal computers (PCs) or supercomputers. However, you will find that the definition of "embedded system" is fluid and difficult to pin down, as it constantly evolves with advances in technology and dramatic decreases in the cost of implementing various hardware and software components. In recent years, the field has outgrown many of its traditional descriptions. Because the reader will likely encounter some of these descriptions and definitions, it is important to understand the reasoning behind them and why they may or may not be accurate today, and to be able to discuss them knowledgeably. Following are a few of the more common descriptions of an embedded system:

- *Embedded systems are more limited in hardware and/or software functionality than a personal computer (PC).* This holds true for a significant subset of the embedded systems family of computer systems. In terms of hardware limitations, this can mean limitations in processing performance, power consumption, memory, hardware functionality, and so forth. In software, this typically means limitations relative to a PC—fewer applications, scaled-down applications, no operating system (OS) or a limited OS, or less abstraction-level code. However, this definition is only partially true today as boards and software typically found in PCs of past and present have been repackaged into more complex embedded system designs.

- *An embedded system is designed to perform a dedicated function.* Most embedded devices are primarily designed for one specific function. However, we now see devices such as personal data assistant (PDA)/cell phone hybrids, which are embedded systems designed to be able to do a variety of primary functions. Also, the latest digital TVs include interactive applications that perform a wide variety of general

functions unrelated to the "TV" function but just as important, such as e-mail, web browsing, and games.

- *An embedded system is a computer system with higher quality and reliability requirements than other types of computer systems.* Some families of embedded devices have a very high threshold of quality and reliability requirements. For example, if a car's engine controller crashes while driving on a busy freeway or a critical medical device malfunctions during surgery, very serious problems result. However, there are also embedded devices, such as TVs, games, and cell phones, in which a malfunction is an inconvenience but not usually a life-threatening situation.

- *Some devices that are called embedded systems, such as PDAs or web pads, are not really embedded systems.* There is some discussion as to whether or not computer systems that meet some, but not all of the traditional embedded system definitions are actually embedded systems or something else. Some feel that the designation of these more complex designs, such as PDAs, as embedded systems is driven by nontechnical marketing and sales professionals, rather than engineers. In reality, embedded engineers are divided as to whether these designs are or are not embedded systems, even though currently these systems are often discussed as such among these same designers. Whether or not the traditional embedded definitions should continue to evolve, or a new field of computer systems be designated to include these more complex systems will ultimately be determined by others in the industry. For now, since there is no new industry-supported field of computer systems designated for designs that fall in between the traditional embedded system and the general-purpose PC systems, this book supports the evolutionary view of embedded systems that encompasses these types of computer system designs.

Electronic devices in just about every engineering market segment are classified as embedded systems (see Table 1-1). In short, outside of being "types of computer systems," the only specific characterization that continues to hold true for the wide spectrum of embedded system devices is that *there is no single definition reflecting them all*.

Table 1-1: Examples of embedded systems and their markets [1-1]

Market	Embedded Device
Automotive	Ignition System
	Engine Control
	Brake System (i.e., Antilock Braking System)
Consumer Electronics	Digital and Analog Televisions
	Set-Top Boxes (DVDs, VCRs, Cable Boxes, etc.)
	Personal Data Assistants (PDAs)
	Kitchen Appliances (Refrigerators, Toasters, Microwave Ovens)
	Automobiles
	Toys/Games
	Telephones/Cell Phones/Pagers
	Cameras
	Global Positioning Systems (GPS)

Table 1-1: Examples of embedded systems and their markets [1-1] (continued)

Market	Embedded Device
Industrial Control	Robotics and Control Systems (Manufacturing)
Medical	Infusion Pumps
	Dialysis Machines
	Prosthetic Devices
	Cardiac Monitors
Networking	Routers
	Hubs
	Gateways
Office Automation	Fax Machine
	Photocopier
	Printers
	Monitors
	Scanners

1.2 Embedded Systems Design

When approaching embedded systems architecture design from a systems engineering point of view, several models can be applied to describe the cycle of embedded system design. Most of these models are based upon one or some combination of the following development models:[1-5]

- The *big-bang* model, in which there is essentially no planning or processes in place before and during the development of a system.

- The *code-and-fix* model, in which product requirements are defined but no formal processes are in place before the start of development.

- The *waterfall* model, in which there is a process for developing a system in steps, where results of one step flow into the next step.

- The *spiral* model, in which there is a process for developing a system in steps, and throughout the various steps, feedback is obtained and incorporated back into the process.

This book supports the model shown in Figure 1-1, which I refer to as the Embedded Systems Design and Development Lifecycle Model. This model is based on a combination of the popular waterfall and spiral industry models.[1-2] When I investigated and analyzed the many successful embedded projects that I have been a part of or had detailed knowledge about over the years, and analyzed the failed projects or those that ran into many difficulties meeting technical and/or business requirements, I concluded that the successful projects contained at least one common factor that the problem projects lacked. This factor is the process shown in Figure 1-1, and this is why I introduce this model as an important tool in understanding an embedded system's design process.

As shown in Figure 1-1, the embedded system design and development process is divided into four phases: creating the architecture, implementing the architecture, testing the system, and
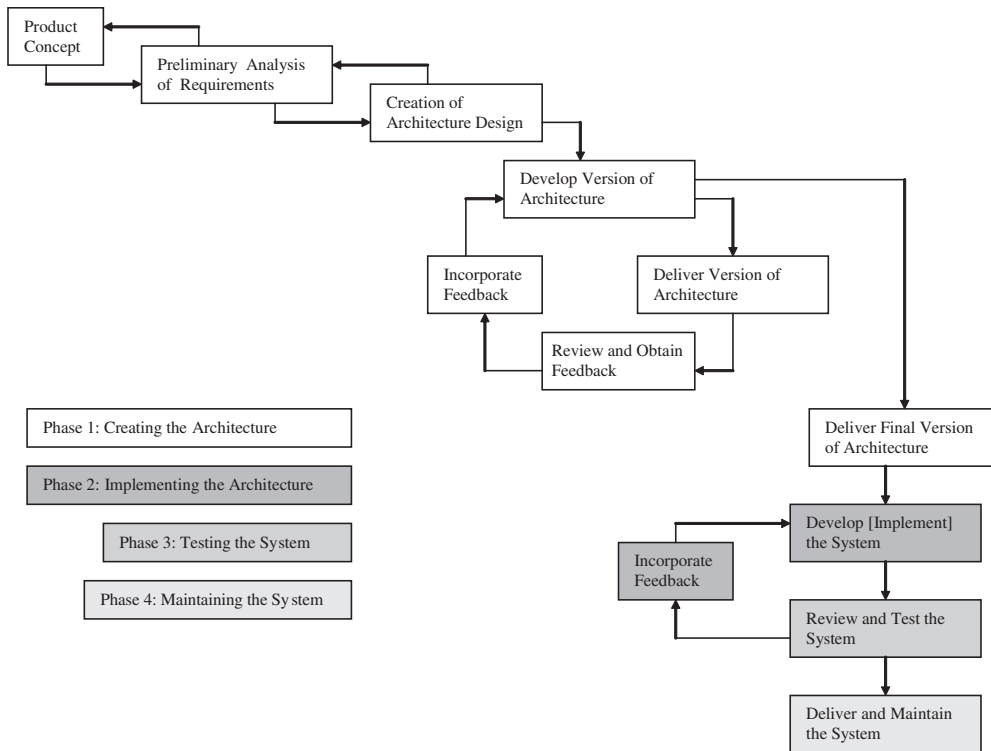
Figure 1-1: Embedded Systems Design and Development Lifecycle Model [1-2]

maintaining the system. Most of this book is dedicated to discussing phase 1, and the rest of *this* chapter is dedicated to discussing why so much of this book has been devoted to creating an embedded system's architecture.

Within this text, phase 1 is defined as being made up of six stages: having a strong technical foundation (stage 1), understanding the Architectural Business Cycle (stage 2), defining the architectural patterns and models (stage 3), defining the architectural structures (stage 4), documenting the architecture (stage 5), and analyzing and reviewing the architecture (stage 6)[1-3]. Chapters 2–10 focus on providing a strong technical foundation for understanding the major components of an embedded system design. Chapter 11 discusses the remaining stages of phase 1, and Chapter 12 introduces the last three phases.

1.3 An Introduction to Embedded Systems Architecture

The **architecture** of an embedded system is an *abstraction* of the embedded device, meaning that it is a generalization of the system that typically doesn't show detailed implementation information such as software source code or hardware circuit design. At the architectural level, the hardware and software components in an embedded system are instead represented as some composition of interacting *elements*. Elements are representations of hardware and/or software whose implementation details have been abstracted out, leaving only behavioral and inter-relationship information. Architectural elements can be internally integrated within the embedded device, or exist externally to the embedded system and interact with internal elements. In short, an embedded architecture includes elements of the embedded system, elements interacting with an embedded system, the properties of each of the individual elements, and the interactive relationships between the elements.

Architecture-level information is physically represented in the form of *structures*. A structure is one possible representation of the architecture, containing its own set of represented elements, properties, and inter-relationship information. A structure is therefore a "snapshot" of the system's hardware and software at design time and/or at run-time, given a particular environment and a given set of elements. Since it is very difficult for one "snapshot" to capture all the complexities of a system, an architecture is typically made up of more than one structure. All structures within an architecture are inherently related to each other, and it is the *sum* of all these *structures* that is the embedded *architecture* of a device. Table 1-2 summarizes some of the most common structures that can make up embedded architectures, and shows generally what the elements of a particular structure represent and how these elements interrelate. While Table 1-2 introduces concepts to be defined and discussed later, it also demonstrates the wide variety of architectural structures available to represent an embedded system. Architectures and their structures—how they interrelate, how to create an architecture, and so on—will be discussed in more detail in Chapter 11.

Table 1-2: Examples of architectural structures [1-4]

Structure Types*			Definition
Module			Elements (referred to as modules) are defined as the different functional components (the essential hardware and/or software that the system needs to function correctly) within an embedded device. Marketing and sales architectural diagrams are typically represented as modular structures, since software or hardware is typically packaged for sale as modules (i.e., an operating system, a processor, a JVM, and so on).
	Uses (also referred to as subsystem and component)		A type of modular structure representing system at runtime in which modules are inter-related by their usages (what module uses what other module, for example).
		Layers	A type of Uses structure in which modules are organized in layers (i.e., hierarchical) in which modules in higher layers use (require) modules of lower layers.
		Kernel	Structure presents modules that use modules (services) of an operating system kernel or are manipulated by the kernel.
		Channel Architecture	Structure presents modules sequentially, showing the module transformations through their usages.
		Virtual Machine	Structure presents modules that use modules of a virtual machine.
	Decomposition		A type of modular structure in which some modules are actually subunits (decomposed units) of other modules, and inter-relations are indicated as such. Typically used to determine resource allocation, project management (planning), data management (encapsulation, privitization, etc.).
	Class (also referred to as generalization)		This is a type of modular structure representing software and in which modules are referred to as classes, and inter-relationships are defined according to the object-oriented approach in which classes are inheriting from other classes, or are actual instances of a parent class (for example). Useful in designing systems with similar foundations.
Component and Connector			These structures are composed of elements that are either components (main hw/sw processing units, such as processors, a Java Virtual Machine, etc.) or connectors (communication mechanism that inter-connects components, such as a hw bus, or sw OS messages, etc.).
	Client/Server (also referred to as distribution)		Structure of system at runtime where components are clients or servers (or objects), and connectors are the mechanisms used (protocols, messages, packets, etc.) used to intercommunicate between clients and servers (or objects).
	Process (also referred to as communicating processes)		This structure is a SW structure of a system containing an operating system. Components are processes and/or threads (see Chapter 9 on OSes), and their connecters are the inter-process communication mechanisms (shared data, pipes, etc.) Useful for analyzing scheduling and performance.
	Concurrency and Resource		This structure is a runtime snap shot of a system containing an OS, and in which components are connected via threads running in parallel (see Chapter 9, Operating Systems). Essentially, this structure is used for resource management and to determine if there are any problems with shared resources, as well as to determine what sw can be executed in parallel.
		Interrupt	Structure represents the interrupt handling mechanisms in system.
		Scheduling (EDF, priority, round-robin)	Structure represents the task scheduling mechanism of threads demonstrating the fairness of the OS scheduler.
	Memory		This runtime representation is of memory and data components with the memory allocation and deallocation (connector) schemes—essentially the memory management scheme of the system.
		Garbage Collection	This structure represents the garbage allocation scheme (more in Chapter 2).
		Allocation	This structure represents the memory allocation scheme of the system (static or dynamic, size, and so on).
	Safety and Reliability		This structure is of the system at runtime in which redundant components (hw and sw elements) and their intercommunication mechanisms demonstrate the reliability and safety of a system in the event of problems (its ability to recover from a variety of problems).
Allocation			A structure representing relationships between sw and/or hw elements, and external elements in various environments.
	Work Assignment		This structure assigns module responsibility to various development and design teams. Typically used in project management.
	Implementation		This is a sw structure indicating where the sw is located on the development system's file system.
	Deployment		This structure is of the system at runtime where elements in this structure are hw and hw, and the relationship between elements are where the sw maps to in the hardware (resides, migrates to, etc).

* Note that in many cases the terms "architecture" and "structure" (one snapshot) are sometimes interchangeably, and this will be the case in this book.

1.4 Why Is the Architecture of an Embedded System Important?

This book uses an architectural systems engineering approach to embedded systems because it is one of the most powerful tools that can be used to understand an embedded systems design or to resolve challenges faced when designing a new system. The most common of these challenges include:

- defining and capturing the design of a system
- cost limitations
- determining a system's integrity, such as reliability and safety
- working within the confines of available elemental functionality (i.e., processing power, memory, battery life, etc.)
- marketability and sellability
- deterministic requirements

In short, an embedded systems architecture can be used to resolve these challenges early in a project. Without defining or knowing any of the internal implementation details, the architecture of an embedded device can be the first tool to be analyzed and used as a high-level blueprint defining the infrastructure of a design, possible design options, and design constraints. What makes the architectural approach so powerful is its ability to informally and quickly communicate a design to a variety of people with or without technical backgrounds, even acting as a foundation in planning the project or actually designing a device. Because it clearly outlines the requirements of the system, an architecture can act as a solid basis for analyzing and testing the quality of a device and its performance under various circumstances. Furthermore, if understood, created, and leveraged correctly, an architecture can be used to accurately estimate and reduce costs through its demonstration of the risks involved in implementing the various elements, allowing for the mitigation of these risks. Finally, the various structures of an architecture can then be leveraged for designing future products with similar characteristics, thus allowing design knowledge to be reused, and leading to a decrease of future design and development costs.

By using the architectural approach in this book, I hope to relay to the reader that **defining and understanding the architecture of an embedded system is an essential component of good system design**. This is because, in addition to the benefits listed above:

1. *Every embedded system has an architecture, whether it is or is not documented, because every embedded system is composed of interacting elements (whether hardware or software). An architecture by definition is a set of representations of those elements and their relationships. Rather than having a faulty and costly architecture forced on you by **not** taking the time to define an architecture before starting development, take control of the design by defining the architecture first.*

2. *Because an embedded architecture captures various views, which are representations of the system, it is a useful tool in understanding **all** of the major elements, why each component is there, and why the elements behave the way they do. None of the*

elements within an embedded system works in a vacuum. Every element within a device interacts with some other element in some fashion. Furthermore, externally visible characteristics of elements may differ given a different set of other elements to work with. Without understanding the "whys" behind an element's provided functionality, performance, and so on, it would be difficult to determine how the system would behave under a variety of circumstances in the real world.

Even if the architectural structures are rough and informal, *it is still better than nothing*. As long as the architecture conveys in some way the critical components of a design and their relationships to each other, it can provide project members with key information about whether the device can meet its requirements, and how such a system can be constructed successfully.

1.5 The Embedded Systems Model

Within the scope of this book, a variety of architectural structures are used to introduce technical concepts and fundamentals of an embedded system. I also introduce emerging architectural tools (i.e., reference models) used as the foundation for these architectural structures. At the highest level, the primary architectural tool used to introduce the major elements located within an embedded system design is what I will refer to as the Embedded Systems Model, shown in Figure 1-2.
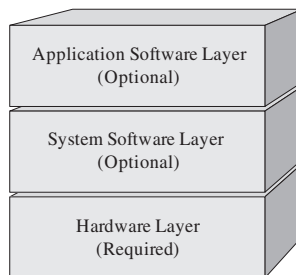

Figure 1-2: Embedded Systems Model

What the Embedded Systems Model indicates is that all embedded systems share one similarity at the highest level; that is, they all have at least one layer (hardware) or all layers (hardware, system software and application software) into which all components fall. The hardware layer contains all the major physical components located on an embedded board, whereas the system and application software layers contain all of the software located on and being processed by the embedded system.

This reference model is essentially a layered (modular) representation of an embedded systems architecture from which a modular architectural structure can be derived. Regardless of the differences between the devices shown in Table 1-1, it is possible to understand the architecture of all of these systems by visualizing and grouping the components within these devices as *layers*. While the concept of layering isn't unique to embedded system design (architectures are relevant to all computer systems, and an embedded system is a type

of computer system), it is a useful tool in visualizing the possible combinations of hundreds, if not thousands, of hardware and software components that can be used in designing an embedded system. In general, I selected this modular representation of embedded systems architecture as the primary structure for this book for two main reasons:

1. *The visual representation of the main elements and their associated functions.* The layered approach allows readers to visualize the various components of an embedded system and their interrelationship.

2. *Modular architectural representations are typically the structures leveraged to structure the entire embedded project.* This is mainly because the various modules (elements) within this type of structure are usually functionally independent. These elements also have a higher degree of interaction, thus separating these types of elements into layers improves the structural organization of the system without the risk of oversimplifying complex interactions or overlooking required functionality.

Sections 2 and 3 of this book define the major modules that fall into the layers of the Embedded Systems Model, essentially outlining the major components that can be found in most embedded systems. Section 4 then puts these layers together from a design and development viewpoint, demonstrating to the reader how to apply the technical concepts covered in previous chapters along with the architectural process introduced in this chapter. Throughout this book, real-world suggestions and examples are provided to present a pragmatic view of the technical theories, and as the key teaching tool of embedded concepts. As you read these various examples, in order to gain the maximum benefits from this text and to be able to apply the information provided to future embedded projects, I recommend that the reader note:

- *the patterns that all these various examples follow,* by mapping them not only to the technical concepts introduced in the section, but ultimately to the higher-level architectural representations. These patterns are what can be universally applied to understand or design any embedded system, regardless of the embedded system design being analyzed.

- *where the information came from.* This is because valuable information on embedded systems design can be gathered from a variety of sources, including the internet, articles from embedded magazines, the Embedded Systems Conference, data sheets, user manuals, programming manuals, and schematics—to name just a few.

1.6 Summary

This chapter began by defining what an embedded system is, including in the definition the most complex and recent innovations in the market. It then defined what an embedded systems architecture is in terms of the sum of the various representations (structures) of a system. This chapter also introduced why the architectural approach is used as the approach to introducing embedded concepts in this book, because it presents a clear visual of what the system is, or could be, composed of and how these elements function. In addition, this approach can provide early indicators into what may and may not work in a system, and possibly improve the integrity of a system and lower costs via reusability.

The next chapter contains the first real-world examples of the book in reference to how industry standards play into an embedded design. Its purpose is to show the importance of knowing and understanding the standards associated with a particular device, and leveraging these standards to understand or create an architecture.

Chapter 1 Problems

1. Name three traditional or not-so-traditional definitions of embedded systems.

2. In what ways do traditional assumptions apply and not apply to more recent complex embedded designs? Give four examples.

3. [T/F] Embedded systems are all:
   A. medical devices.
   B. computer systems.
   C. very reliable.
   D. All of the above.
   E. None of the above.

4. [a] Name and describe five different markets under which embedded systems commonly fall.
   [b] Provide examples of four devices in each market.

5. Name and describe the four development models which most embedded projects are based upon.

6. [a] What is the Embedded Systems Design and Development Lifecycle Model [draw it]?
   [b] What development models is this model based upon?
   [c] How many phases are in this model?
   [d] Name and describe each of its phases.

7. Which of the stages below is not part of creating an architecture, phase 1 of the Embedded Systems Design and Development Lifecycle Model?
   A. Understanding the architecture business cycle.
   B. Documenting the architecture.
   C. Maintaining the embedded system.
   D. Having a strong technical foundation.
   E. None of the above.

8. Name five challenges commonly faced when designing an embedded system.

9. What is the architecture of an embedded system?

10. [T/F] Every embedded system has an architecture.

11. [a]   What is an element of the embedded system architecture?
    [b]   Give four examples of architectural elements.

12. What is an architectural structure?

13. Name and define five types of structures.

14. [a]   Name at least three challenges in designing embedded systems.
    [b]   How can an architecture resolve these challenges?

15. [a]   What is the Embedded Systems Model?
    [b]   What structural approach does the Embedded Systems Model take?
    [c]   Draw and define the layers of this model.
    [d]   Why is this model introduced?

16. Why is a modular architectural representation useful?

17. All of the major elements within an embedded system fall under:
    A.   The Hardware Layer.
    B.   The System Software Layer.
    C.   The Application Software Layer.
    D.   The Hardware, System Software, and Application Software Layers.
    E.   A or D, depending on the device.

18. Name six sources that can be used to gather embedded systems design information.