EMBEDDED SYSTEM DESIGN PROCESS

This section provides an overview of the embedded system design process aimed at two objectives. First, it will give us an introduction to the various steps in embedded system design before we delve into them in more detail. Second, it will allow us to consider the design ***methodology*** itself. A design methodology is important for three reasons.

First, it allows us to keep a scorecard on a design to ensure that we have done everything we need to do, such as optimizing ***performance*** or performing functional tests.

Second, it allows us to develop computer-aided design tools. Developing a single program that takes in a concept for an embedded system and emits a completed design would be a daunting task, but by first breaking the process into manageable steps, we can work on automating (or at least semi automating) the steps one at a time.

Third, a design methodology makes it much easier for members of a design team to communicate. By defining the overall process, team members can more easily understand what they are supposed to do, what they should receive from other team members at certain times, and what they are to hand off when they complete their assigned steps. Since most embedded systems are designed by teams, coordination is perhaps the most important role of a well-defined design methodology.

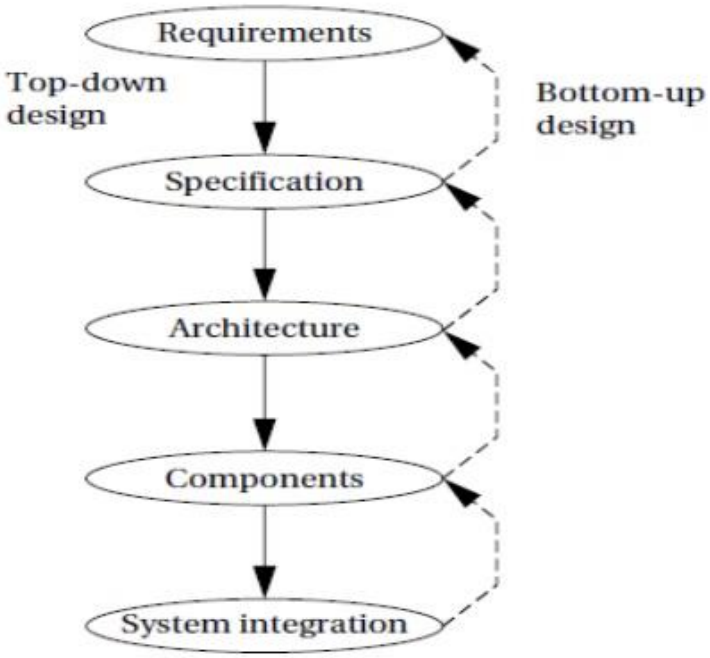


Figure 1.2.1 Embedded System Design Process

[Source: Computers as Components - Principles of Embedded Computing System Design by Marilyn Wolf.]

Figure 1.2.1 summarizes the major steps in the embedded system design process. In this top-down view, we start with the system ***requirements***. In the next step, ***specification***, we create a more detailed description of what we want.

But the specification states only how the system behaves, not how it is built. The details of the system's internals begin to take shape when we develop the architecture, which gives the system structure in terms of large components.

Once we know the components we need, we can design those components, including both software modules and any specialized hardware we need. Based on those components, we can finally build a complete system. In this section we will consider design from the ***top-down*** we will begin with the most abstract description of the system and conclude with concrete details. The alternative is a **bottom-up** view in which we start with components to build a system.

Bottom-up design steps are shown in the figure as dashed-line arrows. We need bottom-up design because we do not have perfect insight into how later stages of the design process will turn out. Decisions at one stage of design are based upon estimates of what will happen later: How fast can we make a particular function run? How much memory will we need? How much system bus capacity do we need? If our estimates are inadequate, we may have to backtrack and amend our original decisions to take the new facts into account. In general, the less experience we have with the design of similar systems, the more we will have to rely on bottom-up design information to help us refine the system. But the steps in the design process are only one axis along which we can view embedded system design. We also need to consider the major goals of the design:

- Manufacturing cost
- Performance (both overall speed and deadlines)
- Power consumption

We must also consider the tasks we need to perform at every step in the design process. At each step in the design, we add detail: We must *analyze* the design at each step to determine how we can meet the specifications. We must then *refine* the design to add detail. We must verify the design to ensure that it still meets all system goals, such as cost, speed, and so on.

Clearly, before we design a system, we must know what we are designing. The initial stages of the design process capture this information for use in creating the architecture and components. We generally proceed in two phases: First, we gather an informal description from the customers known as requirements, and we refine the requirements into a specification that contains enough information to begin designing the system architecture.

Separating out requirements analysis and specification is often necessary because of the large gap between what the customers can describe about the system they want and what the architects need to design the system.

Consumers of embedded systems are usually not themselves embedded system designers or even product designers. Their understanding of the system is based on how they envision users' interactions with the system. They may have unrealistic expectations as to what can be done within their budgets; and they may also express their desires in a language very different from system architects' jargon.

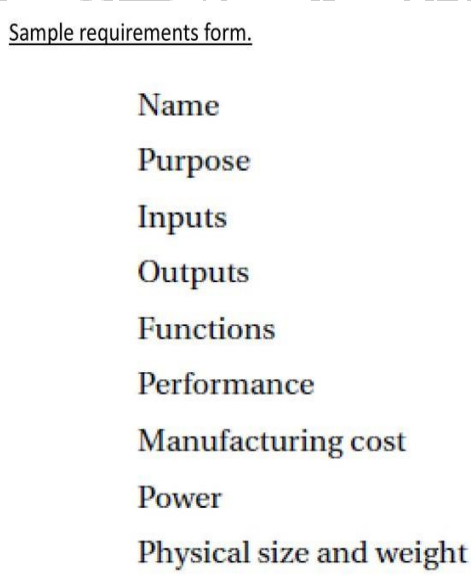
Capturing a consistent set of requirements from the customer and then massaging those requirements into a more formal specification is a structured way to manage the process of translating from the consumer's language to the designer's.

Requirements may be ***functional*** or ***nonfunctional***. We must of course capture the basic functions of the embedded system, but functional description is often not sufficient. Typical nonfunctional requirements include:

- ***Performance:*** The speed of the system is often a major consideration both for the usability of the system and for its ultimate cost. As we have noted, performance may be a combination of soft performance metrics such as approximate time to perform a user-level function and hard deadlines by which a particular operation must be completed.
- ***Cost:*** The target cost or purchase price for the system is almost always a consideration. Cost typically has two major components: ***manufacturing cost*** includes the cost of components and assembly; ***nonrecurring engineering*** **(NRE)** costs include the personnel and other costs of designing the system.
- ***Physical size and weight:*** The physical aspects of the final system can vary greatly depending upon the application. An industrial control system for an assembly line may

be designed to fit into a standard-size rack with no strict limitations on weight. A handheld device typically has tight requirements on both size and weight that can ripple through the entire system design.

- ***Power consumption:*** Power, of course, is important in battery-powered systems and is often important in other applications as well. Power can be specified in the requirements stage in terms of battery life—the customer is unlikely to be able to describe the allowable wattage.
- Validating a set of requirements is ultimately a psychological task since it requires understanding both what people want and how they communicate those needs. One good way to refine at least the user interface portion of a system's requirements is to build a ***mock-up***. The mock-up may use canned data to simulate functionality in a restricted demonstration, and it may be executed on a PC or a workstation. But it should give the customer a good idea of how the system will be used and how the user can react to it. Physical, nonfunctional models of devices can also give customers a better idea of characteristics such as size and weight.


Sample requirements form.

Name
Purpose
Inputs
Outputs
Functions
Performance
Manufacturing cost
Power
Physical size and weight

Figure 1.2.2 Sample Requirements Form

[Source: Computers as Components - Principles of Embedded Computing System Design by Marilyn Wolf.]

Requirements analysis for big systems can be complex and time consuming. However, capturing a relatively small amount of information in a clear, simple format is a good start toward understanding system requirements. To introduce the discipline of requirements analysis as part of system design, we will use a simple requirements methodology.

Figure 1.2.2 shows a sample ***requirements form*** that can be filled out at the start of the project. We can use the form as a checklist in considering the basic characteristics of the system.

Let's consider the entries in the form:

- ***Name:*** This is simple but helpful. Giving a name to the project not only simplifies talking about it to other people but can also crystallize the purpose of the machine.
- ***Purpose:*** This should be a brief one- or two-line description of what the system is supposed to do. If you can't describe the essence of your system in one or two lines, chances are that you don't understand it well enough.
- ***Inputs and Outputs:*** These two entries are more complex than they seem. The inputs and outputs to the system encompass a wealth of detail:
    a. Types *of data:* Analog electronic signals? Digital data? Mechanical inputs?
    b. *Data characteristics:* Periodically arriving data, such as digital audio samples? Occasional user inputs? How many bits per data element?
    c. *Types of I/O devices:* Buttons? Analog/digital converters? Video displays?
- ***Functions:*** This is a more detailed description of what the system does. A good way to approach this is to work from the inputs to the outputs: When the system receives an input, what does it do? How do user interface inputs affect these functions? How do different functions interact?
- ***Performance:*** Many embedded computing systems spend at least some time controlling physical devices or processing data coming from the physical world. In most of these cases, the computations must be performed within a certain time frame. It is essential that the performance requirements be identified early since they must be carefully measured during implementation to ensure that the system works properly.
- ***Manufacturing Cost:*** This includes primarily the cost of the hardware components. Even if you don't know exactly how much you can afford to spend on system components, you should have some idea of the eventual cost range. Cost has a substantial influence on architecture: A machine that is meant to sell at $10 most likely has a very different internal structure than a $100 system.

- ***Power:*** Similarly, you may have only a rough idea of how much power the system can consume, but a little information can go a long way. Typically, the most important decision is whether the machine will be battery powered or plugged into the wall. Battery-powered machines must be much more careful about how they spend energy.
- ***Physical Size and Weight:*** You should give some indication of the physical size of the system to help guide certain architectural decisions. A desktop machine has much more flexibility in the components used than, for example, a lapel mounted voice recorder.