MANTECH Publications
Journal of Embedded Systems & Its Applications
Volume 3 Issue 2, July-December 2018

Fundamentals of Circuits used in Embedded Systems

Sunil Choudhary, Rita Jha, Nitin Dev
Dept. of E&EE
Ghanshamdas Educational Institute of Technology, Meerut, UP
Corresponding Author: *Sunil_choudhary86@rediffmail.com*

Abstract

An embedded system is a combination of computer hardware and software—and perhaps additional parts, either mechanical or electronic—designed to perform a dedicated function. A good example is the microwave oven. Very few people realize that a computer processor and software are involved in the preparation of their lunch or dinner. In this paper, we will explain the fundamentals of circuit designing in embedded systems.

Keywords: *Circuits, Embedded Systems, Fundamental of Circuits*

INTRODUCTION

One of the more surprising developments of the last few decades has been the ascendance of computers to a position of prevalence in human affairs. Today there are more computers in our homes and offices than there are people who live and work in them. Yet many of these computers are not recognized as such by their users. In this paper, we'll explain what embedded systems are and where they are found. We will also introduce the subject of embedded programming and discuss what makes it a unique form of circuit designing and software programming. We'll explain why we have selected C as the language for this paper and describe the hardware used in the examples.

EMBEDDED SYSTEMS

An embedded system is a combination of computer hardware and software—and perhaps additional parts, either mechanical or electronic—designed to perform a dedicated function. A good example is the microwave oven. Almost every household has one, and tens of millions of them are used every day, but very few people realize that a computer processor and software are

involved in the preparation of their lunch or dinner.

The design of an embedded system to perform a dedicated function is in direct contrast to that of the personal computer. It too is comprised of computer hardware and software and mechanical components (disk drives, for example). However, a personal computer is not designed to perform a specific function. Rather, it is able to do many different things. Many people use the term general-purpose computer to make this distinction clear. As shipped, a generalpurpose computer is a blank slate; the manufacturer does not know what the customer will do with it. One customer may use it for a network file server, another may use it exclusively for playing games, and a third may use it to write the next great American novel. Frequently, an embedded system is a component within some larger system. For example, modern cars and trucks contain many embedded systems. One embedded system controls the antilock brakes, another monitors and controls the vehicle's emissions, and a third displays information on the dashboard. Some luxury car manufacturers have even touted the number of processors (often more than 60, including one in each headlight) in advertisements. In most cases, automotive embedded systems are connected by a communications network. It is important to point out that a general purpose computer interfaces to numerous embedded systems. For example, a typical computer has a keyboard and mouse, each of which is an embedded system. These peripherals each contain a processor and software and are designed to perform a specific function. Another example is a modem, which is designed to send and receive digital data over an analog telephone line; that's all it does. And the specific function of other peripherals can each be summarized in a single sentence as well.

The existence of the processor and software in an embedded system may be unnoticed by a user of the device. Such is the case for a microwave oven, MP3 player, or alarm clock. In some cases, it would even be possible to build a functionally equivalent device that does not contain the processor and software. This could be done by replacing the processor-software combination with a custom integrated circuit (IC) that performs the same functions in hardware. However, the processor and software combination typically offers more flexibility than a hardwired design. It is generally much easier, cheaper, and less power intensive to use a processor and software in an embedded system.

EVOLUTION OF EMBEDDED SYSTEMS

Given the definition of embedded systems presented earlier in this paper, the first such systems could not possibly have appeared before 1971. That was the year Intel introduced the world's first single-chip microprocessor. This chip, the 4004, was designed for use in a line of business calculators produced by the Japanese company Busicom. In 1969, Busicom asked Intel to design a set of custom integrated circuits, one for each of its new calculator models. The 4004 was Intel's response. Rather than design custom hardware for each calculator, Intel proposed a general-purpose circuit that could be used throughout the entire line of calculators. This general-purpose processor was designed to read and execute a set of instructions—software—stored in an external memory chip. Intel's idea was that the software would give each calculator its unique set of features and that this design style would drive demand for its core business in memory chips. The microprocessor was an overnight success, and its use increased steadily over the next decade. Early embedded applications included unmanned space probes, computerized traffic lights, and aircraft flight control systems. In the 1980s and 1990s, embedded systems quietly rode the waves of the microcomputer age and brought microprocessors into every part of our personal and professional lives. Most of the el ectronic devices in our kitchens (bread machines, food processors, and microwave ovens), living rooms (televisions, stereos, and remote controls), and workplaces (fax machines, pagers, laser printers, cash registers, and credit card readers) are embedded systems; over 6 billion new microprocessors are used each year. Less than 2 percent (or about 100 million per year) of these microprocessors are used in general-purpose computers.

It seems inevitable that the number of embedded systems will continue to increase rapidly. Already there are promising new embedded devices that have enormous market potential: light switches and thermostats that are networked together and can be controlled wirelessly by a central computer, intelligent air-bag systems that don't inflate when children or small adults are present, medical monitoring devices that can notify a doctor if a patient's physiological conditions are at critical levels, and dashboard navigation systems that inform you of the best route to your destination under current traffic conditions. Clearly, individuals who possess the skills and the desire to design the next generation

of embedded systems will be in demand for quite some time.

REAL-TIME SYSTEMS

One subclass of embedded systems deserves an introduction at this point. A real-time system has timing constraints. The function of a real-time system is thus partly specified in terms of its ability to make certain calculations or decisions in a timely manner. These important calculations or activities have deadlines for completion. The crucial distinction among real-time systems lies in what happens if a deadline is missed. For example, if the real-time system is part of an airplane's flight control system, the lives of the passengers and crew may be endangered by a single missed deadline. However, if instead the system is involved in satellite communication, the damage could be limited to a single corrupt data packet (which may or may not have catastrophic consequences depending on the application and error recovery scheme). The more severe the consequences, the more likely it will be said that the deadline is "hard" and thus, that the system is a hard real-time system. Real-time systems at the other end of this continuum are said to have "soft" deadlines—a soft real-time system. Figure 1-1 shows some examples of hard and soft real-time systems.

Real-time system design is not simply about speed. Deadlines for real-time systems vary; one deadline might be in a millisecond, while another is an hour away. The main concern for a real-time system is that there is a guarantee that the hard deadlines of the system are always met. In order to accomplish this system must be predictable.

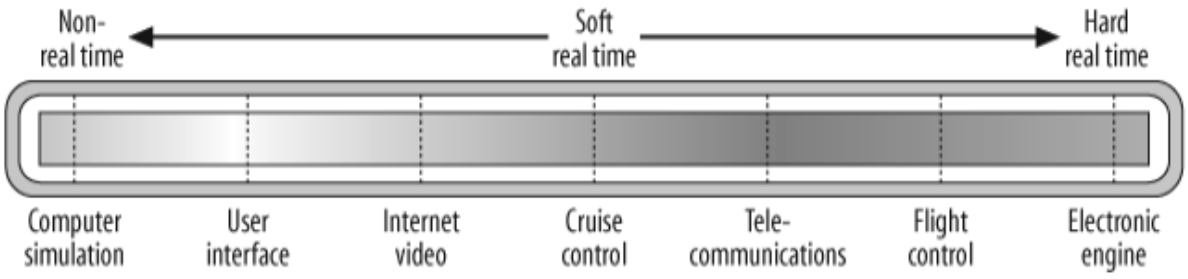

Figure 1: A range of example real-time systems

The architecture of the embedded software, and its interaction with the system hardware, play a key role in ensuring that real-time systems meet their deadlines. Key software design issues include whether polling is sufficient or interrupts should be used, and what priorities should be assigned to the various tasks and interrupts. Additional forethought must go into understanding the worst-case performance requirements of the specific system activities. All of the topics and examples presented in this paper are applicable to the designers of real-time systems. The designer of a realtime system must be more diligent in his work. He must guarantee reliable operation of the software and hardware under all possible conditions. And, to the degree that human lives depend upon the system's proper execution, this guarantee must be backed by engineering calculations and descriptive paperwork.

VARIATIONS ON A THEME

Unlike software designed for general purpose computers, embedded software cannot usually be run on other embedded systems without significant modification. This is mainly because of the incredible variety of hardware in use in embedded systems. The hardware in each embedded system is tailored specifically to the application, in order to keep system costs low. As a result, unnecessary circuitry is eliminated and hardware resources are shared wherever possible.

In this section, you will learn which hardware features are common across all embedded systems and why there is so much variation with respect to just about everything else. Later in the paper, we will look at some techniques that can be used to minimize the impact of software changes so they are not needed throughout all layers of the software.

COMMON SYSTEM COMPONENTS

By definition, all embedded systems contain a processor and software, but what other features do they have in common? Certainly, in order to have software, there must be a place to store the executable code and temporary storage for runtime data manipulation. These take the form of readonly memory (ROM) and random access memory (RAM), respectively; most embedded systems have some of each. If only a small amount of memory is required, it might be contained within the same chip as the processor. Otherwise, one or both types of memory reside in external memory chips.

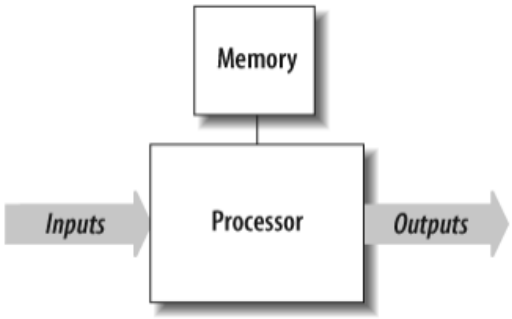

Figure 2: A generic embedded system

All embedded systems also contain some type of inputs and outputs. For example, in a microwave oven, the inputs are the buttons on the front panel and a temperature probe, and the outputs are the human readable display and the microwave radiation. The outputs of the embedded system are almost always a function of its inputs and several other factors (elapsed time, current temperature, etc.). The inputs to the system usually take the form of sensors and probes, communication signals, or control knobs and buttons. The outputs are typically displays, communication signals, or changes to the physical world. See Figure 2 for a general example of an embedded system.

A FEW WORDS ABOUT HARDWARE

It is the nature of programming that papers about the subject must include examples. Typically, these examples are selected so that interested readers can easily experiment with them. That means readers must have access to the very same software development tools and hardware platforms used by the authors. Unfortunately, it does not make sense to run any of the example programs on the platforms available to most readers—PCs, Macs, and Unix workstations. Even selecting a standard embedded platform is difficult. As you have already learned, there is no such thing as a “typical” embedded system. Whatever hardware is selected, the majority of readers will not have access to it. But despite this rather significant problem, we do feel it is important to select a reference hardware platform for use in the examples. In so doing, we hope to make the examples consistent and, thus, the entire discussion more clear—whether you have the chosen hardware in front of you or not. In choosing an example platform, our first criterion was that the platform had to have a mix of peripherals to support numerous examples in the paper. In addition, we sought a platform that would allow readers to carry on their study of embedded software development by expanding on our examples with more advanced projects. Another criterion was to find a development board that supported the GNU software development tools; with their open source licensing and coverage on a wide variety of embedded processors, the

GNU development tools were an ideal choice.

The chosen hardware consists of a 32-bit processor (the XScale ARM),[3] a hefty amount of memory (64 MB of RAM and 16 MB of ROM), and some common types of inputs, outputs, and peripheral components. The board we've chosen is called the VIPER-Lite and is manufactured and sold by Arcom. A picture of the Arcom VIPERLite development board (along with the add-on module and other supporting hardware) is shown in **Figure 3**. If you have access to the reference hardware, you will be able to work through the examples in the paper as they are presented. Otherwise, you will need to port the example code to an embedded platform that you do have access to. Toward that end, we have made every effort to make the example programs as portable as possible. However, the reader should bear in mind that the hardware is different in each embedded system and that some of the examples might be meaningless on hardware different from the hardware we have chosen here. For example, it wouldn't make sense to port our flash memory driver to a board that had no flash memory devices.

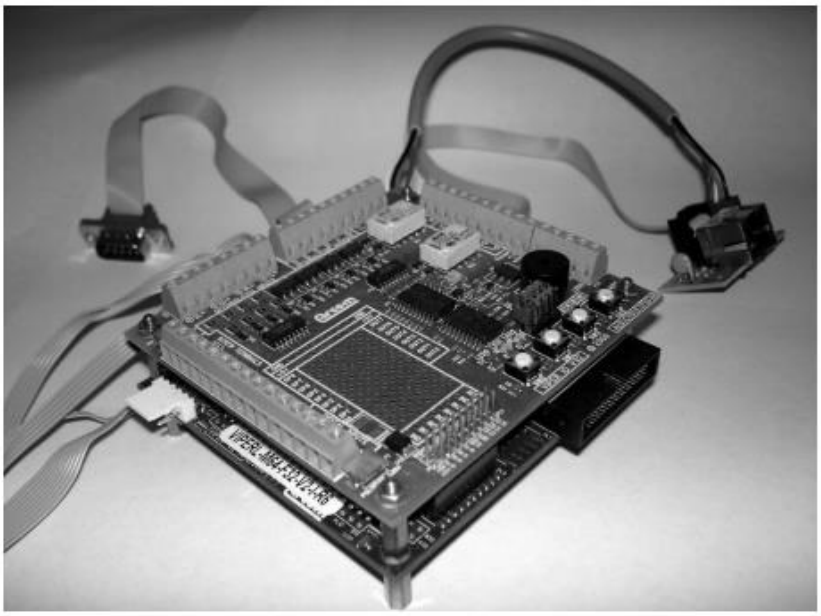

Figure 3: The Arcom VIPER-Lite development boards

CONCLUSION

The standards discussed here for embedded system circuitry and software give you guidelines on everything from directory structures to variable names and are a great starting point; you can incorporate into them the styles that you find necessary and helpful. If a coding standard for the entire team is not something you can sell your company on, use one yourself and stick to it.

REFERENCES

I. Programming Embedded Systems, 2nd Edition, By OReilly. Pp. 345-365 [2] Software Engineering for Embedded Systems: Methods, Practical Techniques and Applications, By Robert Oshana and Mark Kraeling. Pp. 134-245.

II. Embedded Systems and Software Validation, By Abhik Roychoudhury. Pp. 126-132.

III. Embedded Systems Security: Practical Methods for Safe and Secure Software and Systems Development, By David Kleidermacher, Mike Kleidermacher. Pp. 245-256.

IV. An Embedded Software Primer, Volume 1 , By David E. Simon. Pp. 167-187 [6] Better Embedded System Software , By Philip Koopman. Pp. 235-256.

V. Cracking The Code Programming For Embedded SYSTEM (With CD), By Dreamtech Software Team. 131-143.

VI. Design Patterns for Embedded Systems in C: An Embedded Software Engineering. By Bruce Powel Douglass pp. 432-445