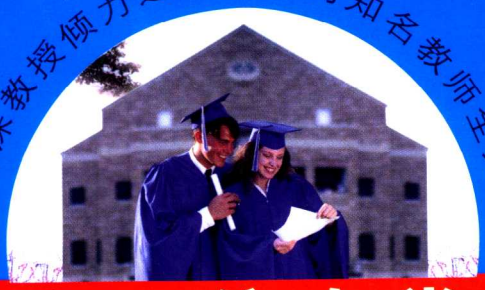
国外资深教授倾力之作　国内知名教师全力推荐

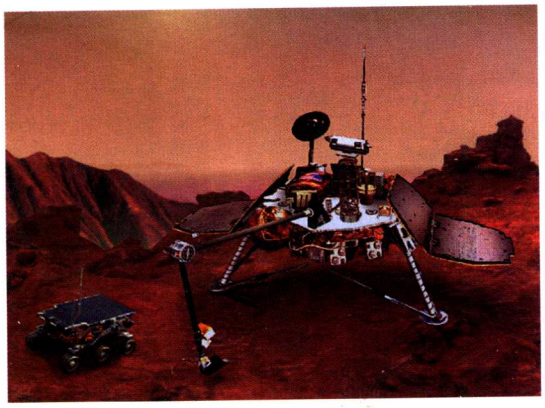


国外高校电子信息类优秀教材

嵌入式微控制器

Embedded Microcontrollers

（英文影印版）



Todd D. Morton 著

科学出版社
www.sciencep.com

国外高校电子信息类优秀教材(英文影印版)

嵌入式微控制器
Embedded Microcontrollers

Todd D. Morton 著

科学出版社

北京

内 容 简 介

本书为国外高校电子信息类优秀教材(英文影印版)之一。

本书从工程的角度对八式微控制器开发流程所涉及的硬件和软件技术进行了系统的介绍,包括 8 位和 16 位微控制器、汇编语言、C 语言、实时操作系统内核。本书着重于设计与调试中的概念、过程、约定和技术。

本书可作为控制专业高年级本科生的教材,也可作为嵌入式系统技术人员的参考书。

图字:01-2003-4429

图书在版编目(CIP)数据

嵌入式微控制器/(美)莫顿(Morton, T. D.)著. —影印本. —北京:科学出版社,2003
(国外高校电子信息类优秀教材)
ISBN 7-03-011690-9

Ⅰ.嵌… Ⅱ.莫… Ⅲ.嵌入式微控制器-高等学校-教材-英文 Ⅳ.TP332.3

中国版本图书馆 CIP 数据核字(2003)第 050474 号

责任编辑:巴建芬 李 宇/封面设计:黄华斌 陈 敬/责任印制:刘秀平

科学出版社 出版
北京东黄城根北街16号
邮政编码:100717
http://www.sciencep.com

新蕾印刷厂 印刷

科学出版社发行 各地新华书店经销

*

2003 年 8 月第 一 版 开本:787×1092 1/16
2003 年 8 月第一次印刷 印张:44 1/4
印数:1—3 000 字数:1 050 000

定价:56.00 元

(如有印装质量问题,我社负责调换〈环伟〉)

国外高校电子信息类优秀教材(英文影印版)

丛书编委会

(按姓氏笔画排序)

Preface

This book is intended for anyone who wants to design small- to medium-sized embedded systems. It was written primarily for electronics engineering technology students, but should also be appropriate for most engineering students and practicing engineers. It is currently being used for two one-quarter courses in electronics engineering technology. The first is a required junior-level course on microprocessor-based applications, and the second is a senior elective course on embedded systems. The first course covers assembly language only, and the second course uses C.

Prerequisites for this book include a background in electronic circuits, basic digital logic, and general C programming. The C programming course is required only for Parts 4 and 5. A background in C and/or C++ is required because this book does not cover C language programming basics. It only addresses the concepts and techniques required for using C in a microcontroller-based embedded system.

SOFTWARE AND HARDWARE USED IN THE BOOK

When writing a book on embedded systems, it is difficult to maintain the balance between being too specific to one hardware/software set and too general to be practical. I have tried to keep this balance by focusing on the concepts, processes, conventions, and techniques used in design and debugging.

The book focuses on the M68HC12 microcontroller from Motorola, but has also been successfully used in courses that use the M68HC11 family. It is intended to supplement, not replace, vendor documentation. I expect every student to have the complete vendor documentation for both the MCU and the development board he or she is using. For example, for the M68HC12 MCU the student should have the *CPU12 Reference Manual* and the *Technical Specifications* for the specific part. If the M68HC11 is used, the famous "pink books" are a requirement.

The development hardware used throughout the text is the Motorola 68HC912B32 EVB. In the first part of the text, only a single board is required, and all code is loaded into RAM. In the second half of the book, the background debug system is used, and the code is loaded into the target Flash ROM. This requires either two EVBs or one EVB as a target and a 68HC12 BDM pod such as the Noral 68HC12 BDM debugger. The concepts regarding the debug process and testing should apply to most modern development systems.

The development software used throughout is the Introl-CODE development system. Except for the sections that specifically address development using the Introl-CODE system, the C code is all ANSI-C, and therefore it would be reasonable to use another compiler.

The real-time kernel covered in this text is MicroC/OS-II. It is available in source form and is widely used. Many of the concepts apply to other kernels, especially the applications for typical kernel services.

CHAPTER DESCRIPTIONS

The book is divided into five parts. The first half of the book emphasizes assembly code and the second half focuses on C. Hardware is covered throughout the text, especially in Part 3. Because of the use of pseudo-C from Chapter 6 on, it is reasonable to cover that material with an emphasis on C instead of assembly.

Part 1 Introduction. This part introduces the reader to the background and perspective required in learning about embedded systems.

Part 2 Assembly Language Programming. This part introduces programming in assembly language and covers the CPU12 programming model and program design. Students should be able to build a prewritten program after completing Chapter 3, and write complete programs that are executed by the D-Bug12 monitor in RAM after completing Chapter 6. Chapter 7 covers some basic applications that are appropriate for assembly code.

Part 3 Microcontroller Hardware and I/O. Real-time concepts and I/O hardware are introduced, including interrupts and basic multitasking. All of the 68HC912B32 I/O resources are covered with the exception of the BDLC. MCU configuration for stand-alone systems along with bus expansion are covered in Chapters 10 and 11.

Part 4 Programming Microcontrollers in C. Concepts in C for programming real-time embedded systems are covered. Emphasis is placed on memory usage and program efficiency appropriate for small MCUs.

Part 5 Real-Time Multitasking Kernels. In this part basic multitasking design is covered along with using MicroC/OS-II, an off-the-shelf kernel.

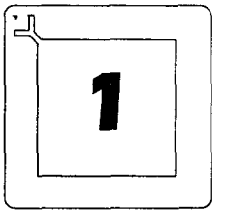
ACKNOWLEDGMENTS

Without the help and patience of the following people, this text would not have been possible: Rich Pennington at Introl Corp, Jim Sibigtroth at Motorola University and Austin Community College, Jean Labrosse at Micrium, Tony Plutino and Dave Hyder at Motorola, Marsh Faber and Mel Downs at Hewlett-Packard, Phil Meek and Harry Erickson at Noral Micrologics, Dave Garza and staff at Prentice-Hall, Kathleen Kitto and Andrew Pace at WWU, George Sweiss at ITT Technical Institute, Malvern Phillips at British Columbia Institute of Technology, the contributors to the Motorola 68HC11 and 68HC12 listserv, and, of course, all of my students. You will finally get those labs and homework assignments graded.

To Edye, Jeva, and Perry

Contents

PART I
Introduction



Introduction to Microcontrollers

We have all heard the stories about how the microprocessor has revolutionized many aspects of our everyday lives. The most visible examples are desktop computer systems and the Internet. Another part of this revolution that we do not often hear about is embedded systems. *Embedded systems* are electronic systems that contain a microprocessor or microcontroller, but we do not think of them as computers—the computer is hidden, or embedded, in the system. Examples of embedded systems include automobiles, industrial controllers, instrumentation, network routers, and household appliances, now even including rice cookers and toasters. Homes in the United States have an average of 30 to 40 microprocessors each, yet only 45% of these homes have a desktop computer. The rest of these processors are used in embedded applications.

In this book we will concentrate on the largest segment of the embedded systems market—the small systems. These are systems that require 8- or 16-bit microprocessors or microcontrollers. Figure 1.1 shows a typical system, a digital thermometer. It is made up of a temperature sensor connected to an ADC, a microprocessor (CPU), RAM, ROM, chip select logic, and an LCD module.

If there are two essential design characteristics for embedded systems they are cost sensitivity and diversity. It does not make sense to use the same system for an infrared remote control as an unmanned spacecraft. Of course this is obvious, but the wide range of complexity is what has guided the evolution of the technology used for embedded systems. In

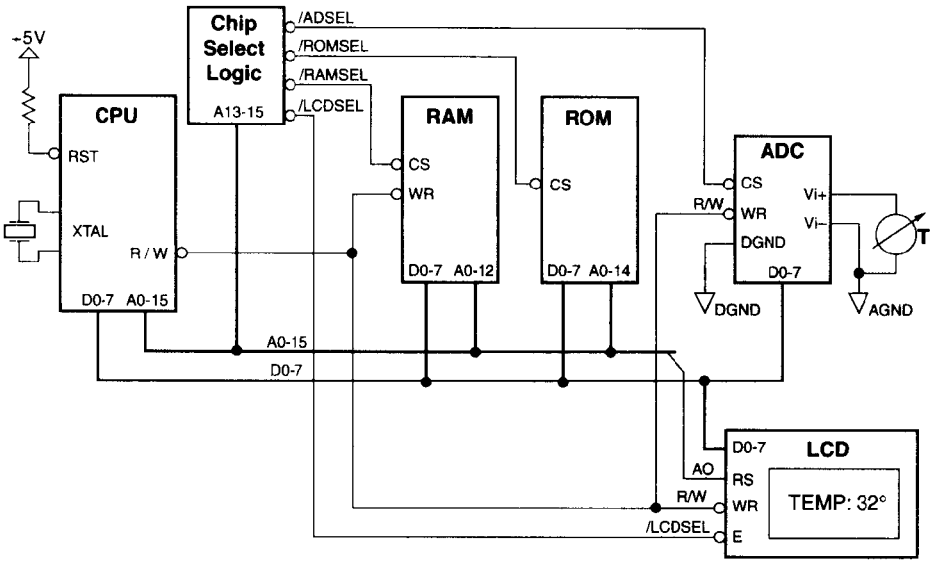


Figure 1.1 A Typical Small Embedded System—Digital Thermometer

addition to this diversity is the requirement that costs must be reduced as much as possible. A video game system will not sell if it costs $1,000. Yet the processor technology used in a video game is on a par with that used in desktop PCs. Therefore there cannot be a *one-system-fits-all* approach to embedded systems design. Embedded system designs require *application-specific* hardware and software, hardware and software designed for each different application.

Another characteristic of embedded systems development is the wide range of skills required to design a system. In the simple example shown in Figure 1.1 there are several technologies represented, as explained in the following list:

- The CPU requires skills in software design and microprocessor interfacing.
- The chip select logic requires familiarity with digital logic.
- The ADC and temperature sensor require skills in analog design and sampling theory. There may also be some knowledge of basic physics required to understand the characteristics of the sensor.
- The LCD requires an understanding of user interfaces and the optical characteristics of the LCD.

In addition, the system may be networked, may run on alternate power sources, or may be placed in a harsh environment. Some embedded system designs are large enough so that it is practical to break the design tasks down into specialties. However, it is most beneficial to understand or at least be interested in learning all the technologies involved. Because of the diverse skills required, this can be a fascinating and rewarding field for an engineer or technician.

▶ 1.1 THE MICROCOMPUTER

We will start by covering the heart of all embedded systems—the *microcomputer*. As shown in Figure 1.2, the microcomputer is made up of the CPU, memory devices, I/O devices, and the bus system.

The CPU or central processing unit is another name for a *microprocessor*. It controls the system and processes data, the memory stores the CPU's programs and data, the I/O devices provide an interface with the outside world, and the bus system provides for a flexible interconnection system. The thermometer in Figure 1.1 is a microcomputer. It has a CPU, its memory includes a RAM and a ROM device, and the I/O consists of an ADC connected to a temperature sensor and an LCD module. The microcomputer is a very flexible system. It allows the designer to include only the devices required for the specific application, which is especially important for embedded systems.

If a microcomputer system is combined into a single integrated circuit (IC), it is called a *single-chip microcomputer* or a *microcontroller (MCU)*. These terms are often used interchangeably but they really represent two different devices designed for different applications. A single-chip microcomputer contains resources typically used for computer systems, such as a memory management unit and a disk controller. The microcontroller, on the other hand, contains resources typically used for embedded systems, such as timers and ADCs. We will be focusing on microcontrollers in this text.

To see the effect of using a single-chip microcontroller, let's go back to our simple embedded system example in Figure 1.1. There are several ways to implement this design. We can use a microprocessor-based system, we can use a microcontroller with an external bus, or we can use a microcontroller in single-chip mode.

If the design is implemented using a microprocessor, all the blocks shown in Figure 1.1 are separate ICs. This means that at least five ICs are required in addition to the temperature sensor and LCD module. Since most of the ICs are connected to the bus, they are large ICs with a large number of pins. This would result in a relatively expensive large printed circuit board.

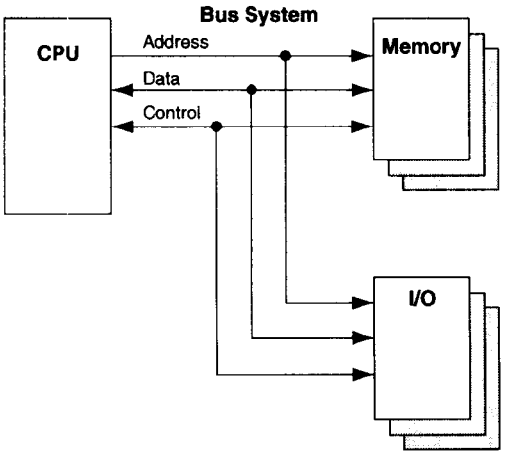


Figure 1.2 The Microcomputer

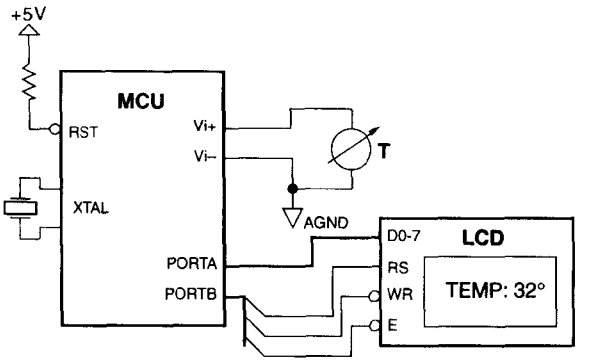


Figure 1.3 A Microcontroller-Based System

The next option is to use a microcontroller that is configured in expanded mode. *Expanded mode* means the bus system is available outside of the microcontroller IC. If the microcontroller contains the chip select logic, the RAM, and the ADC, then the system is down to two ICs, the MCU and the ROM. The PCB size is reduced and typically the cost will go down. This is a compromise design. The number of ICs has been reduced and, the flexibility provided by the external bus still exists. It still requires a large ROM device and the power consumption is still relatively high.

The last option is to implement the design with a microcontroller in single-chip mode. An MCU in single-chip mode does not provide the bus for external connections. The extra pins vacated by removing the bus can be used to reduce the size of the IC package or to add extra I/O. As shown in Figure 1.3 this option results in a single-chip solution. Only the LCD module and temperature sensor circuits are required, so this can be an extremely cost effective and compact design.

1.1.1 The Microprocessor

The microprocessor or CPU is the controller for the microcomputer system. It controls all the bus activity, performs calculations, and makes decisions. The microprocessor is programmable—its operations are controlled by a sequence of instructions. These instructions include three general types: data transfer instructions, arithmetic and logic instructions, and program control instructions. A sequence of microprocessor instructions is called a *program* or *software.*

The combination of a programmable CPU and the bus system results in an extremely flexible system that can easily be customized for a given application. In embedded systems this flexibility is used to create application-specific hardware that runs a single application-specific program.

The microprocessors used for embedded systems are relatively simple when compared with microprocessors designed for desktop computers. Currently the highest volume microcontrollers are those with 8-bit CPUs. Small packages and cost effectiveness are the guiding factors when selecting a microcontroller for an embedded system. It would not make sense to use an expensive 32-bit CPU in a television remote control or a toaster. There are embedded systems on the market that do have powerful 32-bit CPUs, however.

These devices are used in systems that require moving a large amount of data at high speed, or systems that require a large number of complex calculations. Common applications for these microcontrollers include video games, laser printers, network routers, or automotive engine control systems. In this book we focus on the intermediate range of 8- and 16-bit microcontrollers and the design constraints typical of these devices. Some of the material is certainly applicable to the smallest 4- and 8-bit designs, and some of the material also is applicable to the larger 32-bit designs.

1.1.2 The Bus System

The *bus system* for a microcomputer provides a flexible means to transfer data among the CPU, the memory, and the I/O devices. It is flexible because it is a shared bus. To add a memory or peripheral device to the system, you simply have to connect them to the bus system and add the required decoding logic. The CPU controls the bus system by providing a device address on the address bus and bus control signals on the control bus for direction and timing. It then either provides data (writes) or samples data (reads) on the data bus.

The Address Bus. The address bus is made up of CPU outputs that contain the source or destination location for a data transfer. Access to specific locations is controlled on two levels. Chip select logic decodes the address bus to determine which memory or peripheral device to access. Then the device address decoding logic decodes the address to determine the specific location within the device. This is analogous to the postal system in which the mail is directed first to an area post office based on the address. The post office then directs the mail to the specific mailbox based on the address.

Most small microcontrollers have a linear address space. A *linear address space* is one in which each address referred to by an instruction directly corresponds to that location in memory. It is the easiest type of addressing to use but it can result in inefficiencies in the CPU. Paged memory systems can increase the CPU efficiency but can be difficult to work with. In a paged system the address bus contains the location information within a current *page* of addresses. Another CPU register must be used to select the current page.

The size of the address bus determines the total number of locations that are directly accessible by the CPU. For each location to have a unique address you can only have as many locations as you have unique combinations in the address word. Therefore, there are 2^N possible addresses for an N-bit address bus.

EXAMPLE 1.1

Addressable Space for a 16-bit Address Bus

The 68HC11 microcontroller has a 16-bit address bus, which is the typical size for a small microcontroller. What is the maximum number of directly accessible locations?

Solution

For a 16-bit address bus there are $2^{16} = 65{,}536$ locations. For this case we would normally say there are 64K-bytes of memory space (1K-byte = 2^{10} = 1,024 bytes). Note this is not the same as 10^3!