EMBEDDED TECHNOLOGY™ SERIES

Programming Microcontrollers in C

Second Edition

CD-ROM Included!

Contains:
• source code
• pdf versions of Motorola microcontroller reference manuals and databooks
• a full, searchable version of this book!

Ted Van Sickle

Programming Microcontrollers in C

Second Edition

Ted Van Sickle

A Volume in the EMBEDDED TECHNOLOGY™ Series

LLH Technology Publishing

Eagle Rock, Virginia
www.LLH-Publishing.com

ISBN: 1-878707-57-4
Library of Congress Control Number: 00-134094

Printed in the United States of America
10 9 8 7 6 5 4 3 2 1

Project management and developmental editing:
Carol S. Lewis, LLH Technology Publishing

Interior design and production services:
Kelly Johnson, El Cajon, California

Cover design: Brian McMurdo, Valley Center, California

LLH Technology Publishing

Visit us on the web: www.LLH-Publishing.com

Introduction to Second Edition

Today, even more than when the first edition of this book was written, the use of microcontrollers has expanded to an almost unbelievable level. A typical car has 15 microcontrollers. A modern home can have more than 50 microcontrollers controlling everything from the thermostat, to the furnace, to the microwave. Microcontrollers are everywhere! In the meantime, many new chips have been placed on the market as well.

Also, there have been significant modifications to our programming languages. The standard C language is now called C99 rather than C89. There have been several changes in the language, but most of these changes will not be available to us for some time. Many of the modifications to the language will be of little use to programs for embedded systems. For example, complex arithmetic has been added to the language. It is rare that we use even floating-point arithmetic, and I have never seen an application for an embedded system where complex arithmetic was needed. However, other additions allow improved optimization processes, such as the restrict keyword and the static keyword used to modify the index of an array. Other changes have less impact on the generation of code, such as the // opening to a single line comment. Also, today you will have no implicit int return from a function. All in all, expect the new versions of C compilers to be significant improvements over the older versions. Also, expect that the new compilers will not break older code. The features of the new standard should begin showing up in any new version of the compilers that you use.

The C++ standard committee has completed its work on the first language standard for C++. There is much clamor about the use of C++ for embedded systems. C++ as it stands is an excellent language, but it is aimed primarily at large system programs, not the small programs that we will be developing into the future. C still remains the grand champion at giving us embedded programmers the detailed control over the computer that we need and that other computer languages seem to overlook.

The first six chapters of the book have been revised and any errors that were found were corrected. Every chapter has been altered, but not so much that you would not recognize it. Chapter 7 has been added. In that chapter, a relatively complex program is developed to run on the M68HC912B32. The development system was based on this chip and it had no significant RAM to hold the code during development. Therefore, all of the code was completely designed, coded, and tested on a DOS-based system. Extensive tests were completed to make certain that there were no hidden bugs. The modules were small and easy to test. Each module was tested with a program written to exercise all parts of the module. When the several modules were integrated into a single program, the program worked in the DOS-based system. All changes needed to convert this program were implemented under the control of conditional compiler commands. When the program was converted to the M68HC12 version and compiled, it loaded correctly and ran.

Chapter 8 introduces a new chip for Motorola, the MMC2001. This chip is a RISC chip. Many of the good things to be said of RISC configurations are absolutely true. This chip is very fast. Each of its instructions requires only one word, 32 bits, of memory. Almost all instructions execute in a single clock cycle. The chip that I used here ran at 32 mHz, and you could not feel any temperature rise on the chip. It is from a great family of chips that should become a future standard.

The first edition of this book had several appendices. These were needed to show general background material that the reader should not be expected to know. Also, quite a few specialized header files used to interconnect the program to the peripheral components on the microcontroller were included. Also, with the first edition, there was a card with which the reader could order two diskettes that contained all of the source code in the book, demonstration compilers that would compile the source code, and other useful information. All of these things have been included on the CD-ROM that comes with this edition. Additionally, you will find PDF versions of all appropriate Motorola data manuals and reference manuals for all of the chips discussed in the book. Also included are copies of all header files used with the programs, and some more that will probably be useful to you.

Introduction to First Edition

Early detractors of the C language often said that C was little more than an over-grown assembler. Those early disparaging remarks were to some extent true and also prophetic. C is indeed a high level language and retains much of the contact with the underlying computer hardware that is usually lost with a high level language. It is this computer relevance that makes people say that C is a transform of an assembler, but this computer relevance also makes C the ideal high level language vehicle to deal with microcontrollers. With C we have all of the advantages of an easily understood language, a widely standardized language, a language where programmers are readily available, a language where any trained programmer can understand the work of another, and a language that is very productive.

The main purpose of this book is to explore the use of C as a programming tool for microcontrollers. We assume that you are familiar with the basic concepts of programming. A background in C is not necessary, but some experience with a programming language is required. I have been teaching C programming for microcontrollers for several years, and have found that my students are usually excellent programmers with many years of experience programming microcontrollers in assembly language. Most have little need or interest in learning a new language. I have never had a class yet where I was able to jump into programming microcontrollers without providing substantial background in the C language. In many instances, students believe that a high-level language like C and microcontrollers are incompatible. This forces me, unfortunately, to turn part of my class into a sales presentation to convince some students that microcontrollers and C have a future together. I am usually able to show that the benefits gained from using C far outweigh the costs attributed to its use. The first two chapters are included for those who are unfamiliar with C. If you are already familiar with C, feel free to skip ahead to Chapter 3.

C is a very powerful high level language that allows the programmer access to the inner workings of the computer. Access to computer details, memory maps, register bits, and so forth, are not usually available with high level languages. These features are hidden deliberately from the programmer to make the languages universal and portable between machines. The authors of C decided that it is desirable to have access to the heart of the machine because it was intended to use C to write operating systems. An operating system must be master of all aspects of the machine it is controlling. Therefore, no aspect of the machine could be hidden from the programmer. Features like bit manipulation, bit field manipulation, direct memory addressing, and the ability to manipulate function addresses as pointers have been included in C. All of these features are used in programming microcontrollers. In fact, C is probably the only popular high level language that can be conveniently used for a microcontroller.

Every effort has been made to present the C aspects of programming these machines clearly. Example programs and listings along with their compiled results are presented whenever needed. If there are problems hidden in the C code, these problems are explored and alternate methods of writing the code are shown. General rules that will result in more compact code or quicker execution of the code are developed. Example programs that demonstrate the basis for these rules will be shown.

C is a rich and powerful language. Beyond the normal high level language capability, C makes extensive use of pointers and address indirection that is usually available only with assembly language. C also provides you with a complete set of bit operations, including bit manipulations and bit fields in addition to logical bit operations. In C, the programmer knows much about the memory map which is often under programmer control. A C programmer can readily write a byte to a control register of a peripheral component to the computer. These assembly language-like features of the C language serve to make C the high level language of choice for the microcontroller programmer.

As a language, C has suffered many well-intended upgrades and changes. It was written early in the 1970s by Dennis Ritchie of Bell Laboratories. As originally written, C was a “free wheeling” language with few constraints on the programmer. It was assumed that any programmer using the language would be competent, so there was little need for the controls and hand-holding done by popular compilers of the day. Therefore, C was a typed language but it was not strongly typed. All function returns were assumed to be integer unless otherwise specified. Function arguments were typed, but these types were never checked for validity when the functions were called. The programmer could specify an integer argument and then pass a floating point number as the argument. These kinds of errors are made easily by the best programmer, and they are usually very difficult to find when debugging the program.

Another set of problems with the language was the library functions that always accompanied a compiler. No standard library was specified. C does not have built-in input/output capability. Therefore, the basic C standard contained the specifications for a set of functions needed to provide sensible input/output to the language. A few other features such as a math library, a string handling library, and so forth started out with the language. But these and other features were included along with other enhancements in a helter-skelter manner in different compilers as new compiler versions were created.

In 1983, an ANSI Committee (The X3J11 ANSI C Standards Committee) was convened to standardize the C language. The key results of the work of this committee has been to create a strongly typed language with a clear standard library. One of the constraints that the ANSI committee placed upon itself was that the existing base of C code must compile error free with an ANSI C compiler. Therefore, all of the ANSI dictated typing requirements are optional under an ANSI C compiler. In this text, it is always assumed that an ANSI compliant compiler will be used, and the ANSI C form will be used throughout.

C compilers for microcontrollers—especially the small devices—must compromise some of the features of a compiler for a large computer. The small machines have limited resources that are required to implement some of the code generated by a compiler for a large computer. When the computer is large, the compiler writer need not worry about such problems as limited stack space or a small register set. But when the computer is small, these limitations will often force the compiler writer to take extraordinary steps just to be able to have a compiler. In this book, we will discuss the C programming language, not an abbreviated version that you might expect to use with some of the smaller microcontrollers. In the range of all microcontrollers, you will find components with limited register sets, memory, and other computer necessary peripherals. You will also find computers with many megabytes of memory space, and all of the other important computer features usually found only on a large computer. Therefore, we will discuss the language C for the large computer, and when language features must be abbreviated because of computer limitations, these points will be brought out.

All of the programs found in this book have been compiled and tested. Usually source code that has been compiled has been copied directly from computer disks into the text so that there should be few errors caused by hand copying of the programs into the text. The compilers used to test these programs are available from Byte Craft Ltd. of Hamilton, Ontario, Canada (for the MC68HC05) and Intermetrics of Cambridge, Massachusetts (for the MC68HC11 and MC68HC16). If you wish to develop serious programs for any of these microcontrollers, you should purchase the appropriate compiler from the supplier.

How does one partition a book on C programming for microcontrollers? First, the text must contain a good background on the C language. Second, it is necessary to include a rather extensive background on some microcontrollers. Finally, C must be used to demonstrate the creation of code for the specified microcontrollers. This approach is used here. The C