MSP430 Microcontroller Basics

MSP430 Microcontroller Basics

John H. Davies

ELSEVIER

AMSTERDAM • BOSTON • HEIDELBERG • LONDON
NEW YORK • OXFORD • PARIS • SAN DIEGO
SAN FRANCISCO • SINGAPORE • SYDNEY • TOKYO

Newnes is an imprint of Elsevier

Newnes is an imprint of Elsevier
30 Corporate Drive, Suite 400, Burlington, MA 01803, USA
Linacre House, Jordan Hill, Oxford OX2 8DP, UK

Library of Congress Cataloging-in-Publication Data
Application submitted

British Library Cataloguing-in-Publication Data
A catalogue record for this book is available from the British Library.

ISBN: 978-0-7506-8276-3

For information on all Newnes publications,
visit our Web site at: *http://www.books.elsevier.com*

08 09 10 11 12 13 10 9 8 7 6 5 4 3 2 1

Printed in the United States of America

"*To Elizabeth*."

Contents

Preface

About a decade ago, I took over the teaching of a first-year, second-semester course on digital electronics. It covered flip-flops, counters, and state machines, all built from small-scale integrated circuits. One of the projects at the end was to build a digital die. In many ways it was an excellent exercise because there were so many feasible ways in which it could be approached—simple counters, Johnson counters, or state machines. My concern was that it was very close to the project that I had experienced in my first course on digital electronics, which was back in the mid-1970s. The technology was close to the state of the art then, but was it still appropriate after so many years? Another feature of our course is that it is taken not only by electronic engineers but also by students from the science faculty, mostly computer scientists. I wanted these students to leave with a feeling for what can readily be done with modern programmable electronics in smaller-scale systems. I therefore replaced the material in the second half of the course with microcontrollers. (Do not worry, state machines were not abandoned—they are taught with hardware description languages in the context of programmable logic devices.)

More recently, I thought that the time had come to review the choice of microcontroller. We traditionally used 8-bit processors because modern devices have versatile peripherals and sophisticated embedded emulation and are quite powerful enough for most applications. Then the Texas Instruments MSP430 caught my eye. A problem with 8-bit microcontrollers is that 8 bits are too few for addresses, which are typically 16 bits long, and this means that data and addresses cannot be treated on an equal footing. In contrast, the MSP430 has a uniform, 16-bit architecture throughout: The address bus, data bus, and registers in the CPU are all 16 bits wide. The CPU has a modern design with plenty of registers, most of which can be used equally for data or addresses. It has a small instruction set with orthogonal addressing and an ingenious constant generator, which is used to emulate many operations that would otherwise need their own, distinct instructions. In many ways these features make the 16-bit MSP430 simpler than a typical 8-bit processor.

Of course an elegant architecture does not generate many sales in the real world. More important are the range of peripherals and development tools. The MSP430 offers the usual selection of peripherals plus some less common modules, including sigma–delta analog-to-digital converters and operational amplifiers. Some devices include hardware multipliers and digital-to-analog converters, which provide a complete signal chain (although, of course, Texas Instruments also offers an enormous range of digital signal processors). There is a choice of two free development environments (always an important consideration in education). One is IAR Embedded Workbench, which is available for a wide range of microcontrollers. Another, Code Composer Essentials, is produced by Texas Instruments itself. A third option is the GCC toolchain for MSP430 at mspgcc.sourceforge.net.

I have not yet mentioned the major selling point of the MSP430, which is its low power consumption. Many microcontrollers are based on long-established designs with low-power modes grafted onto them. This means that returning to full power from a low-power mode is often awkward and in some cases is virtually a reset operation. The MSP430 is refreshingly different because it was designed from the outset for low-power operation. Entry to low-power modes and exit from them is straightforward, supported by a versatile clock system. For example, the clock module includes a digitally controlled oscillator that restarts at full speed from a low-power mode in less than 1 μs in newer devices. In many applications the MSP430 is put into a low-power mode, from which it is awakened by interrupts. These automatically restore full power for the interrupt service routine and return the processor to low power when it has finished. No extra code is needed for this: It is an intrinsic part of the interrupt mechanism. Most peripherals are designed for low power, although this can sometimes make them a little more complicated than would otherwise be necessary. The main point is that low-power modes are easy to use.

The quality of the data sheets and user's guides is another issue in education and those for the MSP430 are fine. Unfortunately one item was missing in the area of documentation: a suitable textbook in English. I wrote this book to fill the gap.

Outline

Most textbooks on microcontrollers follow one of two approaches. The first is to present a sequence of projects to explore successive aspects of the device. I think that this works well for simpler architectures, notably the 8-bit PICs, because it enables the reader to write functioning programs rapidly. This always feels good. Unfortunately I am not sure that it works as well for more advanced peripherals, which need considerable explanation before the reader can learn to use them fully.

The alternative approach is to describe each module in the microcontroller fully and in turn, starting with the CPU and instruction set and working out to the peripherals. This makes for a well-organized reference book but can be tedious as a textbook.

I tried to steer a course between these two. My inspiration is Kernighan and Ritchie's *The C Programming Language*, which starts with a "Tutorial Introduction" before exploring the language systematically in subsequent chapters. I think that it takes rather more introduction to a microcontroller so the "simple tour," which is my equivalent to the tutorial, does not start until Chapter 4. Before that, the first chapter contains a general introduction to embedded systems and microcontrollers. This sets the scene for Chapter 2, which focuses on the MSP430 and gives a broad view of its features. I include a chapter on hardware and software for developing applications, which I hope will be particularly useful for readers who are new to microcontrollers. It also contains some reminders of features of the C language that are more prominent in programs for microcontrollers than desktop computers—bit fields for instance. This leads into the tour, which runs through some simple programs to illustrate input and output, the inevitable flashing LEDs, and an introduction to one of the timers (the MSP430 has several).

The remainder of the book provides a more systematic description of the MSP430. I start with the CPU and instruction set, and show how the constant generator is used to provide further "emulated" instructions. The clock system is also described in this chapter. It is followed by Chapter 6 on subroutines, interrupts, and low-power modes. I already mentioned that a major feature of the MSP430 is the way in which low-power modes are handled automatically when interrupts are serviced.

Subsequent chapters are concerned with the most widely used peripherals. Chapter 7 on digital input and output starts with the usual parallel ports and goes on to describe liquid crystal displays, which many MSP430s can drive directly. There is a wide selection of timers in the MSP430, which are covered in the next chapter. This is followed by a lengthy chapter on analog input and output. The MSP430 offers many peripherals for analog-to-digital conversion, ranging from a simple comparator to a 16-bit sigma–delta module. I do not think that you can use any of these without some understanding of their characteristics, which explains the length of this chapter. Some MSP430s include operational amplifiers and digital-to-analog converters, which I described briefly. The final long chapter is on communication. I cover only three types of communication—serial peripheral interface, inter-integrated circuit bus, and asynchronous—but there are several peripherals for these in different variants of the MSP430, so there is a lot to explain.

The very last chapter provides an introduction to the MSP430X, an extended architecture with a 20-bit address bus that can handle 1 MB of memory. There is also an appendix to take the reader through the steps of editing, building, and debugging the first project, which can sometimes be a frustrating experience.

I find it annoying when books contain large chunks copied directly from data sheets and have tried to avoid this. You cannot hope to program a microcontroller without the data sheet at your side. Having said that, I start by going through each bit of the registers that control the peripherals used for the early programs. The idea is to explain how a typical peripheral is configured. After that I become more selective and concentrate on the overall function of the peripheral instead. Usually I pick out a few details that I think need extra explanation but skip the more mundane aspects. They are in the example programs in any case.

I include links to many of Texas Instruments' application notes because I can see no point in repeating material that has been thoroughly explained already. I find that many students are strangely reluctant to use this valuable resource. There are a few reminders about code examples for the same reason.

C or Assembly Language?

Most small microcontrollers are now programmed using the C language so the question might seem redundant. In fact often columns in newletters on embedded systems often carry articles with titles such as "Is Assembly Language Dead?" However, the answer seems to be clearly that assembly language is *not* dead for small microcontrollers, such as the MSP430. Most code is written in C but you may occasionally need to write a subroutine in assembly language to perform an operation that cannot be written out directly in C. Two examples are operations that require bitwise rotations rather than shifts and calculations that can be done more efficiently by exploiting special instructions of the CPU, such as binary-coded decimal arithmetic. Intrinsic functions often avoid the need for assembly language but not always.

More important, assembly language is often needed for debugging and this is the most compelling reason for describing it in a textbook. Small microcontrollers typically spend much of their time interacting with hardware by manipulating the registers that control the peripherals. Debugging may require stepping through lines of assembly language to check each step. You have to look at the manual to check the details of each instruction, but it helps to have a general idea of how the assembly language works.

From a pedagogical point of view, assembly language is useful to illustrate the architecture of the processor. In fact the MSP430 is simple enough that you can explore the thinking behind the design of the instruction set. Besides, assembly language can be fun (in small doses).

My approach is to develop the first, simple programs in Chapter 4 using both C and assembly language to show the relation between them. However, C dominates by the end of the chapter. Assembly language makes a strong showing in the next two chapters, which cover architecture, subroutines, and interrupts, including a section on mixing C and assembly language. Almost all remaining programs are in C, with assembly language reappearing only briefly for a function to convert numbers to binary-coded decimal. The listings in the text are read directly from the programs that I tested.

Companion Web Site

Please visit the companion Web site for this book at www.elsevierdirect.com/companions/9780750682763 and download the programs used as examples in the book. These programs were read into the text of the book from the workspaces that I used for testing, which means that the downloaded files should match the book perfectly. Links are also provided for data sheets, user's guides, and development tools. Solutions to the odd-numbered examples are freely available on the companion Web site but the remaining solutions are offered only to instructors.

Acknowledgments

It is a pleasure to thank numerous people who have helped me in various ways to write this book. Many are from Texas Instruments: Bonnie Baker, Jacob Borgeson, Andreas Dannenberg, Colin Garlick, Thomas Mitnacht, and Robert Owen. I am particularly grateful to Adrian Valenzuela for his comments on the final draft. Several engineers from other companies were kind enough to provide advice and assistance: Edward Gibbins and Steve Duckworth from IAR, Tom Baugh of SoftBaugh, Paul Curtis of Rowley Associates, David Dyer of Ericsson and Fernando Rodriguez while he was at Texas Instruments. Finally, I am grateful to colleagues and students at Glasgow University, from whom I have learnt an enormous amount over the years. I'd like to thank Fernando Rodriguez (not the same person who was at Texas Instruments) and David Muir in particular, with both of whom I have run a wide range of projects on embedded systems and microcontrollers—from tutor boxes with flip-flops to the electronic systems of a Formula Student racing car.

John Davies, Milngavie