

Prospectus for Ultramicrocontroller Engineering Book

Professor Jonathan Scott

Summer 2010

Contents

1	Introduction	2
2	Need for the Book	3
2.1	Book Sales	3
2.2	The Target Audience	4
3	Scope	4
4	Coverage	7
4.1	Proposed Contents	8
5	About The Author	9
A	Book Reviews	10
A.1	Wolf[1]	10
A.2	Bates[2]	11
A.3	Barrett[3]	13
A.4	Peatman[4]	14
A.5	Barnett[5]	15
A.6	Spasov[6]	16
A.7	Ganssle[7]	19
A.8	Wilmshurst [8]	21
A.9	Wilmshurst [9]	24

1 Introduction

The proposed book concerns the programming of embedded microcontrollers. The precise scope is explained in more detail in section 3. Search keywords for the area would be “embedded programming”, “microcontroller programming”, “software engineering”, “C programming”, or even “mechatronics”. The book will suit early and established practitioners who want more theoretical grounding, but could be used with 500-level masters or honours undergraduate students. It is intended to provide new ideas and techniques, not simply present existing information in a new way.

The differentiating factors in this book are intended to be

- concentration on the practice of embedded programming of *ultramicrocontrollers*¹,
- minimal association with any particular microcontroller manufacturer or family,
- primary concentration on C-language with mention of assembly programming (instead of the other way around), with emphasis on the non-ANSI embedded adaption of the language,
- integration of software engineering principles, especially firmware standards [14] and formal methods [17], and
- use of a Threshold Concept[11] pedagogical approach, emphasizing the crucial concepts.

The title is undecided—I am open to recommendation. The buzz-words would seem to be “embedded”, “microcontroller”/“ultramicrocontroller”, “real-time”, “programming”, “software engineering” and potentially “mechatronic” and “gadget”. Some possible titles are:

- Theory of Real-Time Embedded Microcontroller Programming: Engineering Gadgets
- Modern Microcontroller Programming: Theory and Practice of Software Engineering Mechatronic Gadgets
- Theory of Embedded Ultramicrocontroller Programming
- Software Engineering of Embedded Microcontrollers
- Embedded Real-Time Programming of Ultramicrocontrollers,
- Real-Time Programming of Microcontrollers for Engineers,
- Embedded Programming of Microcontrollers for Mechatronic Devices,
- Microcontroller Programming of Small Mechatronic Systems

“Gadget” is a tempting but risky word: On one hand it tends to trivialise the device in the reader’s mind, on the other it connotes toys and fun and covers the necessary aspects of mechanical nature and ingenious design. The definition of the word is “a mechanical contrivance or device; any ingenious article”, all very positive and relevant, but a search for synonyms and related words yields “contraption, doohickey, thingamajig, appliance, contrivance, convenience, gismo” betraying the connotation. Engineers might say to each other when using the word “not something silly, we know what we mean”. I like it.

¹The term “ultramicrocontroller” is invented to delineate the smallest class of microcontroller, typically a low pin-count, 8-bit, single-chip device. The word is formed by concatenating ultra- and micro- and controller the same way that micro- and controller gave us microcontroller but in the sense of “ultra-micro controller”. One should note that the term “Embedded” is slightly redundant when associated with programming such an “ultramicrocontroller”.

2 Need for the Book

There are many textbooks available on the general subject of “microcontroller programming” or “embedded programming”, yet many academics complain about the lack of a “truly decent” book on embedded programming. A number of the current offerings are reviewed in various degrees of detail in Appendix A. The reviews expose the reasons for the poor offerings:

- The better books, such as [1] and [8], face a discipline that is so large, and growing so fast, that one book cannot encompass all that a practitioner needs. This leads to incomplete, or worse superficial, treatment, and key ideas become buried in ancillary detail. Books become long and expensive, trying to be everything to everyone.
- Many books, such as [2]–[3], are so connected to a particular platform or architecture that they have difficulty delivering any general principles, and where they do, the lesson is buried in hardware-specific detail. These books tend to say little more than can be deduced from the user manuals supplied by the makers of microcontrollers, compilers, and software tools.
- Embedded/real-time programming sources concentrate on the hardware at the expense of recent software and organisational developments—for example software standards such as [14], and software engineering principles such as are found in the likes of [7].
- Most books available to date pay only lip-service to pedagogical principles, and come out as piles of information that the student must order for him or herself, rather than emphasising principles, theories and techniques that fit into a schema and address general, recurrent, high-level ideas of consequence to the long-term practice of embedded system design.

There is little doubt that there is a need for a book on embedded programming that avoids the above problems. *I plan to achieve this by limiting the scope of the book to the most fundamental type of microcontrollers that are typically used in mechatronic devices, through using the now-common C-language customised to support embedded microcontrollers, avoiding close adherence to any single platform, by integrating software engineering principles from the get-go, and by paying proper attention to the pedagogy—all the differentiating aspects itemised in the introduction.*

2.1 Book Sales

I do not have any figures on the absolute value of sales of embedded system books, but the internet provides a measure of the relative popularity through the Amazon “sales rank”. Table 1 shows a number of figures obtained from the US and UK Amazon sites ranking the sales performance of some relevant books, while table 2 indicates the relative rating from 1 to 5 stars obtained from readers’ reviews. One may immediately observe that performance in the US is not the same as in the UK.

Book	US\$	amazon.com	amazon.co.uk	.com ES	.co.uk ES
Wolf [1]	\$66.32	777,791	682,496	96	62
Bates [2]	\$58.17	337,192	346,399	41	24
Barrett [3]	\$39.21	103,883	1,430,923	12	—
Peatman [4]	\$82.95	373,316	723,278	50	66
Barnett [5]	\$109.15	805,822	531,860	—	—
Spasov [6]	\$119.68	662,758	3,850,954	—	—
Ganssle [7]	\$64.20	813,894	558,214	—	47
Wilmshurst [8]	\$58.62	2,627,489	273,035	—	16
Wilmshurst [9]	\$38.65	84,496	58,217	6	—

Table 1: Price at amazon.com and sales rank at amazon.com and amazon.co.uk in books overall and “books $\gg \dots \gg$ Embedded Systems” (designated ES), for a selection of relevant books. Low numbers are better. Information retrieved July 2010.

Book	amazon.com	amazon.co.uk
Wolf [1]	★ ★ ★	—
Bates [2]	★ ★ ★	★★
Peatman [4]	★ ★ ★ $\frac{1}{2}$	★ ★ ★ ★ ★
Barnett [5]	★ ★ ★ $\frac{1}{2}$	—
Spasov [6]	★ ★ ★ $\frac{1}{2}$	—
Ganssle [7]	★ ★ ★	—
Wilmshurst [9]	★ ★ ★ ★ $\frac{1}{2}$	★ ★ ★ ★ $\frac{1}{2}$

Table 2: Popularity according to reader reviews at amazon.com and amazon.co.uk for a selection of relevant books. Unreviewed books have been deleted from the list in Table 1. Information retrieved July 2010.

2.2 The Target Audience

The vision is for a book targeted at practitioners. The aim is to capture the practical, “useful” aspect of books such as [7], but for the target audience who might otherwise purchase [8]. It would be of interest to practitioners faced with designing a product that stands to use any microcontroller, not only ones whose characteristics resemble those that are the concern of this book, as described in section 3.

The proposed book could be used with a level 400 or 500 course on microcontroller programming. It would be suitable to Mechatronics practice courses, especially ones where students build their own hardware. Given this possibility, we would need to decide the course-specific support such as Learning Objectives, laboratories, exercises, etc.

3 Scope

The book is to be about programming small microcontrollers, “*ultramicros*”, roughly speaking, ones where

- the application is the operating system as well, i.e., typically no Real-Time Operating System (RTOS) kernel between the application and the hardware,
- the task is small enough that it might involve only one programmer,
- reporting an error or getting a field upgrade may not be an option,

- programs are written in C perhaps with some assembly, certainly not a higher language,
- often timing is critical and precision can be comparable to the clock period, and
- often there is no direct evidence visible to the user of a computer at work, no keyboard, no character display, and so forth.

What are these ultramicrocontrollers and where do they fit in? They are the smallest class of microcontrollers. The two most popular manufacturers are Microchip and Atmel, though there are many others including Motorola/Freescale, Intel, Fujitsu, etc. They are used in systems that want low-cost, 8-bit, single-chip microcontrollers of the type epitomised by the PIC12- and Atmel Tiny families. These are low pin-count devices, 6–20 pins in total, typically. They are used in products where it is not immediately apparent that there is even a computer involved, devices that might have been made with analog electronics and discrete logic, except for the fact that it is now easier and cheaper to use an “ultra-micro”. Programming is inevitably “very close to the machine” for reasons of timing, memory space, or hardware specificity.

Examples of products that employ ultramicrocontrollers include tap timers (see figure 1), electronic models and toys (see figure 2), electric toothbrushes, domestic appliances, medical devices, battery chargers, automobile components, industrial equipment, etc., small mechatronic “gadgets”² that usually have no alphanumeric display, no data communication capability and no keyboard interface. Significantly this class of device is “on its own”, meaning that there is no option to signal an error, no higher entity to which a problem can be referred. The best that these systems offer in that regard is internal logging of the error in the hope that the device may one day be pulled apart and analysed. Field updates are not usually an option.

In these “gadgets” the application is the operating system (OS). This has a number of implications. The size of the application is such that the programming is often carried out by one individual, and there is no OS, real-time or otherwise, between the hardware and the programmer. Software maintenance tends to pass from one individual to another, in contrast to the situation where individuals are replaced in a team. There is no abstraction layer, leading to very tight coupling between hardware and firmware.

Timing is often critical. Programs operate in “real time”. Even where the microcontroller is capable of clocking far faster than might be required in order not to miss a deadline, it may have its clock lowered until timing *is* critical, for reasons of power consumption.

The book should not be thought of as being limited to ultramicrocontroller products. *This limitation exists only to make the book tractable. The principles apply to all embedded platforms.*

²I use the word “gadget” without the common connotation of something trivial. These systems may well be inside critical systems, like a pacemaker or a medical drip-pump.



Figure 1: Example of a garden tap timer. One (in this case) or two knobs set the frequency and duration for which the timer turns on the water flow. Devices typically operate for a year on a battery.



Figure 2: Model railway locomotive showing a small electronic circuit that implements a “DCC decoder” using an ultramicrocontroller.

4 Coverage

It is acknowledged that the engineering of mechatronic systems involves the confluence of several skill areas—

1. computer science (programming),
2. electronics engineering, and
3. mechanical engineering;
and it is occasionally important for the practitioner to also be familiar with the field of
4. control.

It is common for Electrical Engineers (EEs) to be employed as embedded programmers. This is the case because EEs understand electronic hardware, and are taught control theory and some programming, while computer scientists are typically taught very little about electronics or control—and the mechanical aspects are developed by Mechanical Engineers (MEs) independently, or, being the least demanding, are “picked up” during the project. On this basis, the majority of books emphasize the software aspects, for the simple reason that there is a developed literature addressing the EE and ME skill requirements, but a much less well-developed literature for the embedded-programming skills.

Two questions influencing the content of the book are:

1. “What level of prior knowledge, in each of these areas, do we assume?”, and
2. “What degree of revision or support is provided to help readers understand the prerequisites and come up to speed if they lack one of them?”.

In view of the typical provision of programmers from an EE background, the book can assume an education in electronics, and some familiarity with control theory. This will roughly correspond to what is covered in the first three years of an undergraduate EE program. Furthermore, virtually all engineering programs worldwide expose students to the C language; even if this was not to be the situation, there are many excellent textbooks that teach C, not the least of which is the original, [10], written by the inventors of the language. It is not the place of this book to attempt an education in ANSI-style C. This can also be assumed.

I believe it will work best if the book only covers the aspects of programming that are unique to the ultramicrocontroller world. The contents in this case are outlined in section 4.1. I envisage a book of approximately 250 pages.

Extras

If coverage was expanded to include all the topics involved with the small mechatronic systems in which ultramicros are used, a huge book would result, perhaps 600 pages plus. I am open to this alternative, but I don’t think it makes sense up front, in a first edition. It is worth noting that mechatronic textbooks often attempt to encompass too much mechanical and electronic material and wind up being immense and still incomplete.

Since the mechanical aspects are typically less demanding, and not covered in EE degrees, they can be of great interest to embedded programmers working on small mechatronic gadgets. An option

would be to have a substantial part of the book on belts, gears, and bearings. Another option would be to have sections on electric motors and actuators—a subject area in the intersection between EE and ME—and drive electronics for inductive loads such as motors and actuators. Section 4.1 lists some possible options as “appendices under consideration”.

4.1 Proposed Contents

The main table of contents of the book is currently planned to be:

Introduction What the book is about and for whom it is ideally written. I would like to include a detailed “teardown report”, perhaps on the DCC train depicted in figure 2.

Case Studies One to three system designs could be presented. This chapter would take the reader through a whole process from specification to debugging.³ For example, we might present the design of an intelligent sprinkler (version of the device pictured in figure 1). A timer for a coffee grinder is simple enough that two platforms could both be described. Competing coding approaches can be compared, for example the interrupt-driven and interrupt-free implementations of an ultrasonic depth gauge. (These are examples with which the author is familiar.)

Design Methodology Moving from specification to architecture in a top-down fashion. This really hinges (I believe) on the definition of tasks and their identification as synchronous and asynchronous, and the layout of the program in the light of this. This chapter would cover some basic architectures, such as a design without interrupts, and a wholly interrupt-driven example, and the more common “division-of-labour” structure. The “modularity” of software can be described here. Ultramicros require modularisation as much as any software project, but the use of functions and discrete files used in large projects is inconvenient and inappropriate for ultramicros. Discuss RISC/Harvard and CISC/Von Neumann platforms. In this section we would cover UML, its place in Software Engineering, and its applicability in the case of ultramicros.

Some ideas involved in the topic of this chapter are covered by Wilmshurst in [8], but I believe he does not place sufficient emphasis on them.

Coding Paradigms and Firmware Standards This section would discuss a number of recurrent coding problems and solutions of varying elegance. Examples to include the production of delays, measurement of intervals, measurement of frequency, event counting, handling of buffers, coding of finite state machines, handling multiple tasks without an RTOS, and other topics. The solutions would be provided at both “pseudocode” level and as C-language or C-and-assembly language examples on two or three platforms with different compilers. The elegance of structure would be covered here, as well as adherence to standards such as [14], techniques for achieving this, and the reasons for exceptions.

Testing and Debugging Debugging using an oscilloscope. Tracing execution and checking values using a “spared” pin. Debugging using non-volatile RAM. ICE and high-level debugging methods. Software simulation?

Interruption Interrupt prioritisation with single-vector and multiple-vector architecture. Interrupt timing, interrupt load, and making deadlines. Un-interruptible code, and preventing bugs from interruption during variable access: Atomic operations Watchdog timers.

Asynchronous Process Communication Principles for communication between interrupt and mainline code: Interrupt masking to atomise operations, semaphores, double-checking. (Give explicit principles as occurs implicitly in Ch. 4 of [7].)

³The question of where to start in a book is often effectively resolved by “going through an example of the whole process” in an early chapter, and then going into the detail in subsequent chapters once the reader has a feel for where they are going. This chapter would do that.

Pin Assignment and Hardware Design The ultramicrocontroller has far fewer pins than functions available, and the pin assignment, hardware and software design go hand in hand. This chapter will exemplify this. Initialisation code is, on account of this, complicated but brief, and not amenable to portability or robust documentation, a matter that can be addressed here.

Arithmetic and Precision Principles behind math libraries, execution speed and precision, coping with overflow and underflow (in assembly and C?) , impact of promotion of variables, and compiler specialisations for embedded situations.

Compilation, Assembly and Disassembly Situations where assembly-level coding is desirable or unavoidable, and analysing what a compiler does by looking at code size or disassembly. Sensitivity to compilers and optimisation, variables overlapping in memory. How compilers work. How subroutines, functions, procedures, and interrupts are implemented. In-line subroutine calling. How data structures are mapped to memory. The compiler's dynamic stack and expression evaluation.

I am inclined to include a chapter discussing formal methods:

Formal Methods Formal methods, exemplified by Z or Isabelle/HOL and involving theorem-proving techniques, were thought to be ready for “prime time” by the end of the 1990s. The anticipated spread has not happened, although there are proven products such as the L4 microkernel that underlies Android-based mobile devices. These are especially appealing in the case of embedded devices upon whose operation lives can depend. Since highly-embedded ultramicrocontrollers are very appealing in many life-support situations, these methods have some relevance to readers of this book. This chapter will describe Formal Methods and explain why they have yet to proliferate and what is involved.

Some appendices that are under consideration:

Mechanical Primer For anyone who wants to know the basics of gears, bearings, and belts, with particular emphasis on small and low-power versions.

Motors An introduction to the theory and practical characteristics of various types of electric motors including Permanent-Magnet Brushed DC, Brushless and Induction types, designed to allow their selection and use.

Drive Electronics for Inductive Loads Power electronics for switching using BJTs, FETs and IGBTs. Protection for inductive loads. Pulse width modulation (PWM). H-bridge circuits. Half-bridge drive. Continuous and discontinuous switching. Voltage stiffness.

IP and its Protection Definition. Design, light-bulb moments and originality. What is patentable and how to patent. Working with a patent firm's people. Software patentability, look and feel. Trade Secrets. Confidential disclosure including contamination. Market lead and sustainable competitive advantage.

Laboratory Projects A number of tasks suitable as hands-on exercises presented with partial or suggested solutions. This could be the basis of lab exercises if the book is used as a course text.

5 About The Author

Professor Scott is the foundation Professor of Electronics in Engineering at the University of Waikato. From 1998 to 2006 he worked for Hewlett-Packard (subsequently Agilent Technologies) in the Microwave Technology Center, California. He is Adjunct Professor at Macquarie University, Sydney, a visiting scholar at The University of Sydney, and a regular reviewer for a number of IEEE journals. He is the author of over 100 patents and refereed publications, a textbook and several book chapters.

A Book Reviews

A.1 Wolf[1]

Summary and Comments

Lynne Conway says of this book in the Foreword that it is “the first textbook to systematically deal with” the “new system design requirements and challenges” of embedded computing. This an accurate assessment; it is the only book I have found that brings a proper, holistic development approach to embedded computing, and one that is mindful of the possibility of designing your own processor as well as simply using an existing MCU. As Wolf says in the preface to the first edition, microprocessors have only just (as of 2000) become powerful enough to take on “truly sophisticated functions”. The book concentrates on these “very sophisticated functions”, and is fundamentally addressing the shift to embedded status, on platforms of complexity approaching that of a desktop PC. This is quite a different arena from the ultramicro embedded systems I seek to address. Wolf addresses the wide context of embedded computers-in-general, rather than the niche of “highly embedded and very lean” (for want of better words) machines that concern this proposal.

To my way of thinking, his Chapter 2 is mis-titled, and really covers “low-level machine architecture and its impact on embedded programming”. He claims that the instruction set is key to understanding the “performance of programs”, in other words knowing what is possible and what your compiler/assembly code has delivered. He sets himself a huge task, because he seeks on one hand to operate at a high level, on PC-sized machines, but on the other he wants to know the nuts and bolts of the MCU instruction set. However, he relies upon a specific couple of instruction sets when in fact a few general principles might be enough, and he assumes a lot of base knowledge.

The chapter-naming scheme is spookily reminiscent of textbooks such as [4] that start with a discussion of CPU and instruction set. This is probably historical, and seems logical to Wolf because of his CMOS IC-design background.

The Content

Chapter 1 lists all the competing concerns and motivations, and presents the idea of a design methodology. It introduces modeling of the software architecture. It covers the idea of Behavioural Description, and introduces UML upon which the book will rely to a great extent.

Chapter 2 covers “Instruction Sets”. Wolf comes from a CMOS design background, hence this rather unusual emphasis on the instruction sets of machines. He uses this chapter to discuss all-important machine architecture, e.g. Harvard vs. Turing/Von Neumann, and CPU architecture itself, including little- vs. big-endian, as well as the mapping between C (compiler) and instructions (assembly). A lot of assumptions of assembler syntax familiarity have been made. The details of the architectures of (Wolf’s two preferred) processor families are given in painfully-fast detail.

Chapter 3 covers “CPUs”, and is also uninformatively titled. It covers such things as busy-wait and interrupt-driven I/O programming, the idea of foreground and background, interrupt priorities and vectors, user and supervisor modes, exceptions and traps, coprocessors, memory management, pipelining, etc. It introduces object-oriented programs and shows an implementation in C and C++.

Chapter 4 looks at hardware system architecture, “bus-based” systems. Since the ARM is available as an “IP core” the buses can be on-chip or off-chip. This opens a new vista of possibilities to be

taken into account. Memory read and write is discussed. I/O into things like timers and counters, on one hand, and ADCs, keyboards and displays are all covered. The delineation between on-chip and off-chip is not a clear divide, appropriate for designers of large (and costly) embedded systems. Development and debugging, breakpoints and performance analysis are discussed. An example with flowcharts is given.

Chapter 5 introduces Program Analysis and Design. This is a key part of the fundamental and long term knowledge upon which Wolf claims to concentrate. Importantly, a state machine coding example is given. Handling buffers, queued and circular, is covered. Program modeling, data flow graphs, and data+control flow graphs are introduced. Assembly, linking, levels of code, and the compilation process is analysed, including the mechanisms for subroutines. A number of (compiler?) optimisations are discussed, as well as scheduling and parallelism. Program-level performance analysis is discussed, from benchmark-style testing to almost “debugging-level” tests with a logic analyser. Coding to increase speed is mentioned. Wolf discusses a number of ideas only available to the “System-on-Chip” (SoC) designer; fine but very much more IC design than embedded programming! Chapter 5 finishes with a section on the ideas of software validation.

Chapter 6 is dedicated to Processes and OSs. The most critical idea in this chapter is the RTOS and the ideas that follow from it, such as processes, contexts, interprocess communication, deadlines, waits, scheduling, etc. The chapter also covers evaluation of an OS and its processes.

Chapter 7 covers multiprocessors. I skimmed this chapter as I am not interested in multiprocessor architectures. It discusses the predictable subjects when elevating multiple threads to multiple processors. The chapter seems to move towards using several discrete processors as one might see in an appliance with multiple PCBs. (I would call this distributed rather than multiprocessor.)

Chapter 8 covers networks. This chapter is basically about inter-device communications. The hardware of ethernet, CAN bus, I²C, etc., are associated with the OSI 7-layer model for communications. Sensor networks are mentioned, but the author has very large systems in mind.

Chapter 9 covers System Design. This chapter is more about organising software design (teams?) than anything else, about iteration in the design process, and refinement of designs. Formal ideas from the literature are discussed. Specification to architecture is the goal of much of the chapter. Quality control is brought up.

A.2 Bates[2]

Summary and Comments

I consider this to be the worst kind of textbook, unsure of what might be a reader’s previous experience, at one and the same time assuming certain information while supplying information that would likely be learnt at the same time as the assumed information. It is vague in its starting position, theoretically weak in its content, and constituted mainly from material that is available in manuals.

I do not consider that it is any competition at all.

The Content

The foreword is talking about the transition from UV-erasable parts to electrically erasable ones, an event that obsoleted a number of older textbooks in the late 1990s. This is a very passé reflection

for a book published in 2008. The Introduction does a good job of identifying the contents of the 5 Parts and 6 Appendices of the book.

Part 1, entitled “PIC Microcontroller Systems”, spends most of its 33 pages telling the reader what he or she would get from the data sheet of a given microcontroller, to the extent of reproducing the file register map. The rest of the chapter describes tools such as programmers and the manufacturer’s IDE. In the case of this book all of these are related to a particular microcontroller, the PIC16F877, an early-success model of a popular line. The approach seems to assume that the reader has no idea what is inside a microcontroller, and yet it assumes that the reader knows what is inside a computer; memory, ALU, and so forth. Fundamental aspects of the device paradigm, such as the observation that it is a Harvard-architecture structure, are not mentioned. No emphasis is placed on facilities, such as brown-out reset, that are peculiar to highly-embedded systems. The first truly useful piece of information regarding the job of design using microcontrollers is given to the reader almost casually: “The general approach to microcontroller application design followed here is to develop a design using a chip that has spare capacity, then later select a related device that has the set of features most closely matching the application requirements”. To my way of thinking this chapter gives too much information that most readers can get elsewhere, and does not emphasize elements of the process that they will need to know to be effective embedded systems designers.

Part 2 is entitled “C Programming Essentials”. It purports to introduce the reader to C. It clearly assumes that the reader understands programming. It is absurd to imagine a reader who knows anything about programming but who does not have even a passing familiarity with C constructs. It is a poor subset of the material in [10]. It entirely fails to address matters such as the ways in which C for a microcontroller differs from standard ANSI C. The conceptual overlap between an ‘INT8’ and a ‘char’ in the compiler Bates chooses to use is not addressed, nor are the alternatives embraced by other compilers. Crucial concepts such as promotion seem to be ignored. The need for an infinite loop is not discussed explicitly. Assembly programming is mentioned.

Part 3 is entitled “C Peripheral Interfaces”, and includes discussion of analog inputs, interrupts, hardware timers (including application specific hardware such as PWM attached to them), serial communications via UART, SPI, etc., and interface to outside hardware such as DACs and EEPROM(!). This part of the book is almost a manual of hardware examples, without the hardware. Towards that it is a good series of examples.

Part 4 is entitled “Mechatronics Applications”, and it consists of a combination of the manual for a demonstrator board, and a series of elementary applications implemented on that board. This part would have appeal to educational institutions because it is close to a “hands-off laboratory instructor’s manual”. The basic criticism of this book—that it teaches virtually no theory or design skills behind the programs—surfaces again here; for example a temperature controller is produced, but it is of the bang-bang design and no control theory whatsoever (and thus the risk of instability) appears. The use of a simulator for program test and debug is introduced but it relies on preparations that are not explained.

Part 5 entitled “PIC16 C Applications and Systems” might have addressed the lack of any instructions on how to design hardware and programs. Sadly it is at a low level, more like draughting than design. “Hardware Design” really means “how to draw block diagrams”, “Software Design” only suggests the student use a flowchart of pseudocode variations of the same. Every example (throughout the entire book) is a sequential loop structure program. There is a discussion of polling vs. interrupts. There is an example using a simple RTOS, but it is provided by the compiler and its workings are not discussed, nor what happens if tasks have interrupts.

A series of Appendices discuss Schematic Capture, the CCS compiler, simulation testing, a comparison between various compilers, CCS syntax, and the CCS function reference.

A.3 Barrett[3]

Summary and Comments

This book concerns itself only with one processor family, and is squarely designed as a[n undergraduate?] textbook even to the extent of starting chapters with the “learning objectives” and finishing with problems. As the book progresses, it becomes so closely wedded to the chosen microcontroller that it takes liberties and shortcuts that probably would not fit well with other choices.

Pedagogically it is a very poor performer. The learning objectives may be an “afterthought” as they do not fit well with the actual content. This is a passable textbook for the self-learner wanting to become involved with Atmel AVR microcontrollers, but is not aimed at general principles. I say “passable” because code snippets contain some very crude and undesirable constructions that might cause havoc in a serious system, for example delays constructed of nested for-loops with large constants. I say that it is weak in terms of general principles because of many omissions and because it does not practice the rigor it preaches: Having introduced UML it does not use it to explain or document its own code snippets. Chapter 6 is embarrassing in its shortcomings (see notes below)!

The Content

Chapter 1 covers the definition of, and the process of outlining, an embedded system. It introduces ideas such as planning and documentation using UML. Notably, *it does not achieve the “learning objectives” stated at its outset*. For example, one objective is to “Apply embedded system design practices in the design of a microcontroller system employing several interacting subsystems”, while the chapter never gets near interacting subsystems with microcontrollers, but rather uses a simple design request to exemplify the process, concluding that a pure electronic hardware solution is more appropriate. This latter is a pedagogically very good and useful example, but not what the objectives implied. I can but think that the author was asked to add the objectives as an afterthought!

Chapter 2 covers what can be obtained from the data sheet of the selected microcontroller family, but it does so with a little bit of relevant theory added. For example, it discusses RISC and Harvard architecture ideas (if not their alternatives), and exemplifies use and application of the components in the micro. It then goes into the software used to program the demo board that accompanies the selected microcontroller, this section reading like a lab startup set of instructions. A knowledge of C is assumed. In this chapter certain C-software habits are suggested, and while noble they suit the selected AVR family but perhaps not all machines.

Chapter 3, entitled “Serial Communication Subsystem”, describes the serial communications provided by the chosen processor family. It covers some general matters, such as ASCII, voltage levels, and Baud rates. Snapshots of C-code are included, but appear to be incompletely explained, for example the `usart_delay` variable in the subsystem intercommunication example circa p56.

Chapter 4 covers AD Conversion. The chapter starts with an extremely-condensed version of sampling theory, and analog-to-digital conversion theory including quantisation. Then it gives a very brief recap on opamp circuit results. Finally it describes successive-approximation ADC logic. Thereafter it describes the chosen controller’s ADC subsystem. (The author and editor do not seem to know how to spell prescaler.)

Chapter 5 is entitled “The Interrupt Subsystem”. It describes, mostly with code snippets, how to use interrupts in the chosen microcontroller. It discusses foreground and background processing. In a strange way this chapter simultaneously assumes the reader knows what is an interrupt, and explains

what they are and why they are useful. The concept of an interrupt is a Threshold Concept[11] when viewed in software terms, but perhaps not so when viewed in the hardware-centric terms of this chapter, as epitomised in Figure 5.1. The chapter omits discussion of schemes for distinguishing interrupt sources other than that used by the AVR family. Some of the code snippets are very inefficient.

Chapter 6 carries on with ideas that appeared in chapter 5, moving from interrupts to a “Timing Subsystem”. The chapter discusses the AVR family’s components that are relevant to real-time operation, much of which is in the microcontroller’s datasheet, but it also introduces timing terminology, implicitly covers the connection with the system “timebase” (master clock), and through examples explains how to achieve some key functions such as period measurement. Unfortunately the basic principles that it introduces, for example the polling vs. interrupt approaches to interval measurement, are not even explained in terms of flowcharts, let alone the UML approach presented in chapter 1, and the interrupt method is so cursorily treated it does not address the risk of timer overflow!

Chapter 7 discusses the voltage and current levels on the microcontroller pins, pullups, switch debouncing, and similar interface matters. It covers a small selection of example sensors and output devices including displays, inductive loads and motors, including elementary single-ended, H-bridge, and stepper-motor drives. In EE terms the treatment is very shallow. Although interesting to students, the treatment is not sufficient to allow design of interface circuits in commercial applications.

Chapter 8 covers “System Level Design” by presenting 3 case studies. Activity diagrams (soft and hard blocks, flowcharts) are shown for some projects. The logic behind the construction of diagrams is not given. Code is given, but ignores certain subtleties (such as timer overflows).

A.4 Peatman[4]

Summary and Comments

I have only skimmed this book. It is extremely old-fashioned, and all programming is carried out in assembly. Chapters often cover only a specific piece of hardware, from the hardware platform that the author uses throughout. The book is rather specifically a course textbook, or more a textbook for a rather specific course. Some chapters do introduce a “principle” rather than an “item”, for example “Interrupts and timing” introduces the idea of priority, and code that must not be interrupted, and “Math Subroutines” necessarily covers the idea of library functions, albeit in a context of structured assembly, and most importantly covers the ideas of precision, and the assessment of memory space consumed by, and cycles required to execute, a library-style function. The chapter on “IO Pin functionality” inadvertently establishes the principle of defining functions and associating them with pins. “Time-Interval Measurement” implicitly introduces the implementation of timeout functionality.

This book is not modern enough, in spite of its date, to be considered as a threat; it is also a pedagogically ill-designed book.

The Content

The chapters are:

1. Intro

2. CPU architecture
3. Instruction Set
4. Target PCB
5. Program Development: An Example
6. Structured Assembly Preprocessor (conditional assembly of code)
7. LCD Display
8. Rotary pulse generators
9. Interrupts and timing
10. ADC
11. IO Pin functionality
12. LCD screens
13. Time-Interval Measurement
14. Math Subroutines
15. SPI
16. PWM
17. SMB/I2C
18. UART
19. Programmed Peripheral Chips
20. Miscellaneous Features

A.5 Barnett[5]

Summary and Comments

I have only skimmed this book very quickly. It assumes a PIC18-series processor, but this is not the main reason for my cursory attention. The book contains very little original material, see the content summary below, and assumes enormous previous capabilities on the part of the reader. Indeed, it is hard to believe that a reader who satisfies the reading prerequisites could be ignorant of what this book introduces.

The Content

Chapters 1–4 are no more than regurgitated manuals for the microcontroller, its compiler, and associated programs.

Chapter 5 starts with definition by means of which the hardware pin functionality is fixed. The chapter goes on to introduce, almost as part of a single design example, the selection of motors, and the electronic hardware design required. A knowledge of PID control is assumed as such a loop is coded.

A.6 Spasov[6]

Summary and Comments

In the preface the author states that he delivers principles, and that the platform is not important. He also starts out with the possibility of readers who know so little that they need his appendix on binary numbers and arithmetic. The idea of data storage, of hardware memory and the memory map, consumes a few pages, yet is a Threshold Concept that takes some considerable internalisation on the part of a student new to it. I respectfully suggest that a reader will only like this book if already familiar with this idea of memory and maps of memory. Likewise indirect addressing consumes only a few pages, but is a big shock for students not already familiar with pointers in C, another Threshold Concept. An Amazon reviewer states: “I found this book to be very long and drawn out. It is very hard to follow and easily confuses the reader. I have found the Motorola 68HC11 Data and Technical books to be a much better resource for information. They are much easier to follow and understand. Save your money for another book, this one is overpriced”. This seems to be the result of the approach, the principles may be there, but they are in a lot of possibly-redundant material, and the reader loses interest or focus, in this case where they basically have the ideas of higher-level programming, microprocessors, etc.

The reverse case—a novice reader coming in cold—will not like the book because it attempts to bring them up to speed on some things too quickly. For example, Spasov mentions for the novice that more information on C can be obtained from Kernighan and Ritchie’s (1978 version of) [10] at one moment, and within the same chapter introduces the “*ptr” construct and then pointer addition!

Spasov’s assertion of his platform not being important also takes a beating early, because he dives into his chosen 68HC11 without noting that it is a Turing-style, rather than Harvard, machine, or even acknowledging the difference. A similar situation exists with instructions sets and the CISC/RISC tradeoff. It is a bit unfortunate that many microcontrollers do not even have general-purpose stack, or explicit stack pointers, as the book assumes them.

Spasov does use a number of very powerful and useful analogies, such as the playing-card analogy for instruction bytes. On the other hand he misses a number of pedagogical no-brainers, for example, where code is presented with C and assembly versions doing the same job, the pairing is not treated “side by side”.

The Content

The book is divided into parts, and each part into chapters:

1. Introduction to Microcontrollers

Chapter 1 This chapter explains what a microprocessor/microcontroller is, offers an appendix for readers who do not understand binary numbers and arithmetic, gives some 20th-century history, lists some applications, introduces the idea of development hardware, gives block diagrams (such as found in a product manual), introduces the idea of memory and the block architecture, registers, etc.

2. Software

Chapter 2 talks about “high level” (C) programming, but mostly assembly language, barely mentioning microcode. The chapter spends some pages preparing the reader to understand the instruction set for the target machine, which is given in an appendix, and a lot of pages

explaining it. The idea of a simulator is mentioned. Looping and branching is introduced. Compilation is mentioned. Assembly instructions are explained in terms of data moving around the parts of the CPU and controller. Instruction addressing modes (a very CISC idea) and multiple-byte instructions are described. The various addressing modes are in separate subsections (e.g. 2.5.8) but these are not contiguous. The chapter has a good description of 2's complement and 10's complement arithmetic and the condition register.

Chapter 3 covers “stack, subroutines, interrupts and resets”, all of which require understanding of the operation of the stack and program counters when changing context in a program. The chapter assumes that the reader has never programmed a subroutine before. The author seems to use the words function and subroutine interchangeably (I think a function returns a value while a subroutine does not.) The chapter talks about “Top-down Problem Solving” and pseudocode. Breaking a problem down into ever smaller steps (“structured programming”) is exemplified with a greatly-reduced version of a large task. The idea of subroutine libraries is introduced. The importance of documentation is discussed. Parameter-passing methods, reentrancy and nesting are covered. Reset and interrupt concepts are introduced. Interrupt sources and vectors are described, along with masking, enabling, and priority. The 68HC11 does not use flags, but the author mentions polling. Stack overflow is noted.

Chapter 4 covers cross assembly and describes the nuts and bolts of programming a 68HC11 on a demo board. The chapter provides a lot of information that would be available in the documentation provided with the assembler, and explains the two-pass mechanism of assemblers, and conditional assembly. The chapter contains a number of CISC-only concepts.

3. Hardware

Chapter 5 discusses the general operation of the digital bus, together with processor-specific modes that affect bus operation in the author's target microcontroller. Address, data and control lines are described, as are the tri-state idea, decoding addresses and bus interfacing. The 68HC11 has various hardware-different ways of operating (modes) and they and their uses are covered.

Chapter 6 covers, in the first part of its 24 pages, “semiconductor technology” and “transistor technology”, logic levels and CMOS families. Then it covers the electrical characteristics from the device data sheet, and types of memory (RAM, ROM, EPROM, etc). On the plus side it does finish with the all-important configuration register and a description of its function.

Chapter 7 is entitled “Clocked Operation”. It covers timing of the bus introduced in Ch5, and generally introduces the idea of watching lines clock and using an oscilloscope or logic analyzer to track operation. The chapter finishes with the idea of transmission lines (a threshold RF engineering concept!) and “transmission line effects”, meaning delay and no mention of dispersion. (This is pedagogically risible.)

4. Interfacing

Because the 68HC11 is not intended as a SoC-style MCU, interfacing to it is important. This is not the case with ultramicros, so I have only skimmed chapters 8–12.

Chapter 8 covers “Interfacing”, meaning ways the MCU is connected to external hardware. The chapter tries to do this at a “high level”, but in effect manages only to become rather vague. For example, the 68HC11 uses an external IC called a PRU to “recover” ports when their pins are used for bus interfacing in one particular mode, but exactly how it does this is not explained in any concrete, digital-hardware way. The chapter covers memory mapping and the idea of a memory map, polled and interrupt-driven interfacing, and various functions controlled through registers, leading to data and status registers.

Chapter 9 under the banner of “Parallel I/O” covers bit- and byte-size input and output, and ends with a description of handshaking and strobing. It mostly presents examples of wiring up switches, single LEDs, and displays (7-segment and LCD), and matrix multiplexing. Centronics, GPIB, and SCSI buses are mentioned, as is wired-OR logic.

Chapter 10 covers serial data interfaces, including Baud rate, data frames, flow control protocols, error detection, half- and full-duplex for asynchronous methods. The 68HC11 hardware support for serial is introduced. Synchronous SPI is described. Packet data protocols are mentioned but not fully explained (TCP/IP, UDP, etc). The chapter gets up to an external ethernet IC.

Chapter 11 covers the use of timers including PWM, pulse counting, pulse width/frequency/period measurement, missing-pulse detection, long period measurement, etc., plus driving stepper motors.

Chapter 12 describes the ADC subsystem of the 68HC11. It implicitly touches on resolution, quantisation, offset, span, etc. A selection of transducers is given. The idea of signal conditioning is introduced with a number of opamp circuits given briefly in recipe-book style. Next actuators are discussed with occasional cavalier omission of flywheel diodes. DACs are shown interfaced to the MCU. Various ADC topologies are defined (in the fashion of an electronics textbook but very briefly) although the 68HC11 has an in-built ADC system. The ADC is briefly discussed but it is rather quirky as integrated ADCs go and its limits are not pushed.

5. The Microcontroller World

Chapter 13 is entitled “Application Control Software”, but the meaning of the title, the aspect common to its contents, is not clear. The chapter discusses controlling things that are either on or off, like a warning light; machines that follow a simple time/event-stepped sequence, like washing machine; state machines, like an elevator or indicators with memory; control of analog things, such as temperature or depth. *In this category the chapter tries to explain the field of control in 500 words, the impact of sample rate without the idea of filters or Nyquist theory, PID control for people who do not understand complex numbers or Laplace theory!* The chapter then talks about DSPs, digital filters and FFTs in 10 pages or less. Finally it returns to C and says some more about variables and stack in the presence of C code. It returns to pointers and interrupts in C. It implicitly covers the idea of working out what a compiler does by disassembling compiled code. A section is then dedicated to multitasking. A large section gives an example of Fuzzy Logic and fuzzy control.

Chapter 14 entitled “Applications”, presents case studies. The first is engine control, ECM of an internal combustion engine. I do not think this example is described well, the spark timing control in particular is not easy to understand, even though it is implementing no more than a variable phase delay. More seriously, the text says that the MCU can adjust the air-fuel mixture, *but the diagrams show a system that cannot do this.* (There may have been an incomplete update to the example with the advent of fuel injection... Spasov seems to think that a “LAMBDA” sensor measures air mass rather than oxygen.) The second example is an automatic (film) camera controller. The author describes the system, but without much detail. The optical details are not covered. Code is given in pseudocode snippets. Control is essentially a series of triggered sequences. The third example is a dot-matrix printer. The author suggests replacement of printer electronics as a good technical challenge. The description is unexciting except for the absence of a free-wheel diode on a solenoid-driver BJT. The chapter concludes with a page of philosophical reflections on the reliability, morality, and impact on society of MCUs.

Chapter 15 discusses 8-bit microcontroller families, that is the groups of similar circuits that differ in amounts of memory or numbers of peripherals but have the same CPU architecture. The chapter introduces the idea of a “core” and the 8051 core. By core, the author means not a design the user purchases but the heart of any series of products in

an MCU family. Much comparative information would be best presented as a table, such as manufacturers do, but it is presented as in-line descriptions here. To a great extent, this chapter achieves what might as well be achieved by browsing manufacturer's product selection guides. It concludes with a discussion of how one might go about identifying the project need, and thus choosing the appropriate MCU. "Changes in architecture are beyond the scope of this book."

A.7 Ganssle[7]

Summary and Comments

An Amazon reviewer says of this book "it is more political than technical". This is an (older?) engineer saying that this book teaches "Software Engineering" (SE), rather than Computer Science (CS). The worth of this *excellent book* lies in its emphasis on SE. If the book has a fault it is that it rambles, pontificates, and entertains as much as informing.

For examples, Chapter 3 contains much pontification about firmware habits. The key (threshold?) idea should be to find a sensible standard and adhere to it, but Ganssle tries to dictate the standard without sufficient rigor. Chapter 2 basically says "use more, smaller, microcontrollers wherever possible", and thaose are the target of my proposed book. Ganssle implicitly appreciates being "pin limited". The proposed book would use that as a central design paradigm, not a glancing comment on planning. Chapter 5 contains much useful information but is not a complete treatment of anything. Chapter 4 on "things real-time" is a font of examples, especially bad ones, but is light on any principles or limits of a global nature. Ganssle even cites the MISRA-C standard, but does not debate its lessons.

The Content

Chapter 1 This is a short introduction that identifies the reader as an engineer. It identifies engineers as problem solvers who need to share experience to learn. It paints microcontrollers as a major development of our time, and defines embedded systems. It points out that coding fell to hardware engineers. The book is a way for them to learn software skills for that coding job. Today we would call this a software engineering text, and Ganssle says this in later chapters. Thus we would say that this book is about "The Software Engineering of Embedded Microcontrollers".

Chapter 2 discusses high-level, project-approach matters. It stresses the need for modularity, breaking up programming jobs into manageable, small pieces, to reduce effort/cost. It uses amusing engineering-analogy terms like "mental cache" to explain the reasons for this paradigm. It presents statistics of programmer productivity variation and identifies important factors. Modularity is extended to hardware (micros), suggesting use of separate processors rather than multiple jobs handled by one CPU/firmware. It presents the lesson of never do in firmware what you can do in hardware—use microcontrollers. The chapter goes on to say that CPU load impacts software difficulty—keep load low. The chapter discusses the economics of NRE (non-recurrent engineering, or development cost) and COGS (cost of goods sold, or per-unit production costs). Statistics of personal productivity with person and task size, and the Dunning-Kruger effect, are used to provide management advice. Product features are used to prioritize modules of a project and the tradeoff between "features, schedule and quality" is introduced. The idea of integrating testing and a continuous development cycle is discussed, stressing the need to add into a solid framework rather than shooting for it all at once (do not wait for final

hardware). Estimating projects is discussed, including requirements/specification documentation, and methods given.

This chapter is a good, readable summary of the SE project literature.

Chapter 3 presents the idea of a firmware standard. Ganssle suggests many “rules” (guidelines). Some make sense, others are very arguable, for example about variable names. Commenting attracts a lot of comments, some disagree with each other. He describes commonly-seen trends, such as the decline in commenting effort with time. Code Inspection (software review to a hardware engineer) is described in detail, and its importance proven. Design by Contract—techniques for automatic code checking and a precursor to provable specification—is introduced. MISRA-C and Lint are described. Encapsulation, information hiding and Globals are discussed.

Chapter 4 deals with real-time concepts. Interrupts are identified as the most important but the weakest area for practitioners. Much of this chapter simply gives advice, with principles implicit, so it is useful but not rigorous. The idea of an ISR context saving, and vectoring are introduced. The need to measure execution time is noted. The idea of “leaving enough time” is implicitly given (as the only real insurance of making deadlines?). The idea of “unloading the ISR” is introduced. Ensure that all possible interrupts are caught even if they are not expected to ever occur. Risks with external interrupt hardware uncertainties (unSchmitted/undebounced lines) are noted. The need to compare interrupt rates and clock rates and to know when to add hardware to reduce ISR load is discussed. The C/assembly decision is reconsidered as critical in the ISR. C compilers come better and worse, know yours. The need to iterate and build the code up slowly, testing at each stage, is described. Techniques to detect missed interrupts are identified as important, but the only techniques given suit a bus-vectorized platform. The author discourages use of NMIs for anything routine. ISR debugging cannot be done using any conventional tool, so must ultimately be done by observation, another reason for short ISRs.

The RTOS is introduced briefly, and advice is given for the RTOS/no RTOS decision, basically if the system is big, must be certified, or has many activities, use one. Rate Monotonic Scheduling in an RTOS is discussed, and the RMS calculation of loading given. The need to measure execution times and frequencies is implicitly brought up.

Re-entrancy and atomic code/variables are introduced (the idea of multiple executions of the same program with different data areas, not just recursion). The impact on interrupts is discussed (no elaboration on passing values to ISRs and the reentrancy rule “do not use the hardware in a non-atomic way” is not explained). The idea of pushing interrupt state and disabling interrupts to get re-entrant code is explained. Some connections are not made; for example the C ‘int’ is a recognised problem but a core reason (it is not the CPU-register-sized in 8 bit systems) is not mentioned.

Hardware metastable states are identified. A large portion of the chapter rambles about common sense and guesstimation, leading to the conclusion that one must “tinker” with the subject. (This looks more like a reflection in the source of the writing as a series of articles not a textbook.) This discussion leads to use of oscilloscopes (the “tinkering”?). The idea of timing events by watching outputs is introduced, debugging using a ‘scope. Latency and ISR durations are measured singly and statistically. Tracing code in real time using added events that a ‘scope can observe is described in some detail. Measurements, such as duty cycle, using a voltmeter are shown. Mapping processes using an ad-hoc DAC is presented.

Floating point (arithmetic) problems are discussed, speed of execution, variation with compiler, and rounding risks. Numerical approximation, by polynomials and other methods, and speed gains are noted. How to approximate log, exponential and trig functions. Compiler transcendental function accuracy is discussed.

Chapter 5 is entitled “The Real World”, and deals with hardware, or rather its shortcomings. The chapter is very much an amusing story, but acknowledges that physics and maths lie behind simple rules, then touches on dispersion of fast digital signals, mismatch and overshoot/ringing. It talks about delay and skew, admires the oscilloscope as a tool, and advises the reader

about other references. There is an extensive discussion of debouncing switches and relays, in hardware and software, with measured data on switches. A number of code snippets are presented (some nasty typos) but the algorithms are not explained (logic, rules or flowcharts).

Chapter 6 is entitled “Disciplined Development”, which title is true but does not say what the chapter is about. It is about tools and practices that embedded people, teams and management should use to bring order and productivity to a project. It is allied to Appendix C.

The cost of firmware is given, and the “Capability Maturity” model is given as a way to gauge the quality of SE PM (Project Management), while the improvements possible are quantified. The need for version control and rigorous backup is described. The need for firmware standards is noted. The use of code inspections and a proper personnel environment are quantitatively evaluated. Performance metrics and their routine use are described. The risk of speed bypassing thought and care is a central idea. The need to monitor SE literature is noted. The use “standard” engineering management practices such as postmortems is advocated.

Appendix A Presents a firmware standard. This meshes well with documents such as MISRA-C, etc.

Appendix B resents a standard for drawings. This idea is not new, manufacturing companies have had master drawings and documents for decades. A major contribution is to make it clear that a computer age tool be available. The standard mainly identifies the documentation that is key to an embedded system.

Appendix C is entitled “A Boss’s Guide to Process Improvement” but is really a laundry list of tools and ideas to help get efficient, safe, maintainable code from staff; Version control, Firmware standard, Modularise hrd- and software; code inspections (reviews), willingness to “start over”, pay for needed tools, value staff and provide them with a good environment. These latter are not specific to the firmware/SE environment.

A.8 Wilmshurst [8]

Summary and Comments

This is the best book available on the subject of “ultramicrocontrollers” to the best of my knowledge. It contains much of the software engineering (SE) that is lacking in practice. It contains most of the technical detail practitioners require. It is well-referenced. The reasons for its failure in the US market are not clear. (Any idea Julie?)

Like every book on the subject it talks about rigorous top-down design and techniques for planning code before writing it, but it does not adhere to that maxim in its own examples. It loses focus on the small-scale aspects. It contains little Mechatronics.

Nevertheless it is the book to beat.

The Content

Chapter 1 defines an embedded system, describes what knowledge about microprocessors is assumed on the part of the reader, identifies the need for reliability and real-time capability in embedded systems, discusses microprocessor concepts such as Turing and Harvard architecture, and CISC vs. RISC in instruction sets. It also describes what is means by a “small microcontroller”, the scope of the book. This chapter introduces the platforms that will be used for examples (they cover RISC and CISC, Harvard and Turing).

Chapter 2 introduces a "minimum system hardware", to be expanded in later chapters. It introduces special-function registers, and the memory-mapped hardware interface. It introduces interrupts, including enabling, masking, flags, vectors, and priority. It introduces ports, including pin multiplexing, direction control registers, driving capability, weak pull-up, individual pin access, interrupts from port pins, and the "quasi-bidirectional" port arrangement and its risks. The chapter then explains some minimal interfacing jobs such reading switches and driving LEDs. Next counter-timers are introduced, and example applications given, such as generating regular interrupts, and measuring intervals. The Watchdog/COP timer is described. Power supply functions such as reset and brown-out reset are described, followed by various clock-frequency options and how they are selected. Finally prototyping strategies and options are given.

Chapter 3 does for software what chapter 3 does for hardware: It provides the minimum to get started intelligently. Programming is done in assembly. The need to know the instruction set and the hardware architecture map is explained. The need to know about addressing modes in order to use assembler is explained. Assembler syntax is explained. (This is all close to understanding the device data sheet, and this chapter adheres closely to the PIC architecture in examples.) The operation of assemblers is described, and branching in assemblers is explained. A general starting format for an assembly program is given for the PIC16F84. Subroutines and ISRs are explained, and flowcharts introduced. Two methods of achieving delay are given ("soft" (loop) and "hard" (interrupt-driven)). Variables, buffers, tables and memory are discussed, including stacks (data and PC). Debugging is introduced through a simulator. Breakpoints are introduced. A number of common error types are described, but they are not usually exemplified or associated with a circumstance such as assembly or C languages. The key documentation (circuit, listing) is identified. Initial hardware checks are suggested. The author gives problems. Actual exam questions are included and identified.

Chapter 4 Types of memory (RAM, ROM, EEPROM, static, dynamic, etc), their features, advantages and disadvantages are described. Notional circuits are given. Programming procedures are described.

Chapter 5 covers analog interfacing. Quantisation, resolution and accuracy are described. PWM DACs are examined, including ripple and filtering. Soft and hard PWM are described. ADC sampling issues are described in the time domain. Signal conditioning using opamps is briefly mentioned and references provided for readers who need depth. Multiplexing and filtering are mentioned. Sample-and-hold terminology and performance is covered. Linearity and accuracy of the ADC process is described. Successive approximation ADCs are described, including switched-capacitor topologies. Some methods to obtain analog values without an ADC are described. SoC ADCs are described to the extent of explaining the data sheet parameters, conversion speed, etc. System ADC selection is described.

Chapter 6 covers serial communication. Asynchronous and synchronous mechanisms are explained. Framing is described. Jargon terms are explained. Waveform dispersion is described, first as an RC effect on short links, then crudely as a transmission line with delay and reflections. EMI and crosstalk are noted. Ground and differential signalling are discussed. The OSI 7-layer model is mentioned but it is not exemplified. Various serial systems are described including Microwire, SPI, RS-232, CAN, and I²C, each in enough detail to allow analysis with a 'scope. Some hardware serial port implementations are described at the logic-diagram level. A soft serial (bit-banging) port is described.

Chapter 7 under the title of "System Software" introduces some ideas of Software Engineering (SE) and the C-language as a "high level" tool. The software life cycle is introduced and analysed in terms of project aims. Flow charts are discussed and some common algorithmic program-flow structures listed. The Module Flow Diagram is introduced. Top-down problem decomposition and structure diagrams are described, and Jackson's Structured Design is used to modularise some tasks otherwise viewed as flow diagrams. Modules can then be coded

in assembler, compiled HLL (C) or interpreted language if appropriate. The C-language is described in terms of how it gives machine independence, and compilation is discussed. C customisation and portability is considered. MISRA-C and the idea of coding guidelines are noted. Version control is mentioned.

Chapter 8 discusses the real-time keystones of systems, namely using interrupts and counter/timers, and coping with multiple tasks in parallel. The term “real-time” is defined in terms of response time, time measurement, and deadlines. The difference in the hardware ways of handling multiple interrupt sources/priorities in different processors is nicely summarised. Interrupt logic and latency are discussed, but nesting (handling multiply-occurring events or interrupting interrupts) is NOT discussed. Context saving is discussed (but not the difference between assembly and C). Interrupt masking and non-interruptible code are brought up (but the problems of communicating values to interrupt routines are but mentioned, not addressed).

Next in this chapter the author switches to counter/timer matters. Resolution and accuracy of times is described. The ideas of measuring period or frequency are described. Prescalers and capture registers are described. The operation of watchdog timers on various platforms is described.

Next the author considers dealing with multiple tasks. Various program structures based on the “main loop and interrupts” are given with advantages and disadvantages of each. The author is referred to an appendix for an example. When certain circumstances are met, an RTOS is suggested as the solution. (The decision process is not described as well as in [7].) The author describes an RTOS, and some ideas of resource sharing, but does not go into any detail. Notably, he makes no exact comment upon the appropriateness of an RTOS on his target 8-bit platforms.

Chapter 9 deals with hardware interfacing. Schmitt triggers and input protection are applied to single-pin digital inputs. Several methods of hardware switch debouncing are given. Matrix keypad input is described. Wilmshurst recommends that keypresses generate interrupts but does not explain or discuss methods for keypad arrays. Switching of BJTs and MOSFETs by microcontroller pins is discussed next. The reader is assumed to be familiar with transistors, and a few circuits are given without much explanation of advantages or disadvantages, while a couple of numerical examples are the only way the reader might know of how to apply a circuit. Protection is not discussed at all. Then the driving of discrete LEDs and 7-segment LED displays is covered, and driving LCD displays. Opto-isolators, reflective and other mechanical sensors, and shaft encoders are described. Next the author returns to switching of loads and discusses driving inductive loads, relays (solid-state ones are mentioned but their use or selection is not discussed), and H-bridge principles. PWM drive into an inductive load is discussed, but motor terminologies such as current-continuousness/discontinuousness is not mentioned. PWM is applied to the bridge. The PM BDC motor is introduced and equations given (dc, no Laplace). PWM motor drive is described. Mention of speed control by sensing terminal voltage is made but this is not described. Stepper motors are described. It is obvious that the author knows a great deal about stepper-motor drive, and the operations are described in some logical detail, if not electronic detail. Brushless DC motors resemble the PM version of stepper that is mentioned, but the brushless DC motor is not mentioned at all.

Chapter 10 essentially discusses power supply design. It starts with analysis of the pros and cons of primary battery, rechargeable battery and mains power, and discusses different battery types. Power conditioning using linear and switching regulators is discussed. Boost, buck and charge-pump switching topologies are shown. Brownout and failure warning are introduced. The low power properties of CMOS are explained in terms of a simple CMOS inverter, and low-power microcontrollers are described. Techniques for low power system design are listed.

Chapter 11 covers numerical methods. Fixed-point arithmetic and implicit scaling is described. Addition and subtraction of numbers longer than the register length are described, including overflow detection. Multiplication and division algorithms are given. Extending length on

multiplication/division is discussed. Binary/BCD conversion is described. Floating point representation, truncation and rounding are discussed. Averaging, differentiation, extrapolation and integration methods for sampled data are described.

Chapter 12 defines and exemplifies the entire process of designing and commissioning a system that uses microcontrollers. It starts by defining design and showing models of the design process. It talks about the formal process of refining broad ideas to specific details. It explains evolutionary (variant) design. (No mention of IP or its protection.) A long case study of a charging system for car parking is then presented. This includes the process of test and debugging. Some diagnostic techniques and tools (hard and soft) are described including ICE and logic analyzer. This chapter conspicuously departs from its focus on *small-scale* systems, as many of the processes and tools that are mentioned are inapplicable to ultramicrocontrollers. The chapter concludes with a look at the latest trends.

Various appendices provide information but these are not comprehensive or material to the book content.

A.9 Wilmshurst [9]

Summary and Comments

This is essentially an updated and less-focused version of [8], with the PIC18 series and C added and program structure ideas ignored. I do not like this book at all. It is my impression that the book has been “diluted” perhaps with the aim of appealing to US markets or to account for the “dumbing down” that has occurred in science and engineering curricula to accommodate trends in preparedness of high-school leavers. I consider it much less pedagogically and theoretically sound.

The first few chapters cover the same content, or at least address the same pedagogical aims, but have been updated to account of changes in the technology. The book still seems to favour assembly-level programming, which is surprising, as the last decade has seen a major shift to programming in C where compilers are adapted to the hardware and can yield as efficient a code as can an assembly programmer, while being more portable and more receptive to standardisation.

What was in Chapter 7 in the previous book is brought forward to Chapter 5 and introduces the reader sooner to the structure of programs, flow- and state diagrams, a good pedagogical move. JSP seems to have been dropped, it is unfashionable these days.

Chapter 7 introduces “Larger Systems”, but these are not “large” as are those treated by Wolf, but are merely less tiny.

Chapter 8, “Human and physical interfaces” is really the old Chapter 9, and there is actually no “human interface” engineering at all. It is about hardware interfaces, not any process of deciding what they should be in the light of the user.

Chapter 9, “Taking Timing Further”, is part of the old Chapter 8, covering use of timers, interrupts, PWM, etc., especially on the Derbot platform.

Chapter 10, “Starting with serial”, covers synchronous and asynchronous serial communications, mostly SPI/I²C and UARTS in PIC devices. It is a slightly reduced version of the old Chapter 6.

Chapter 11, “Data acquisition and manipulation”, is no more than the use of the ADC and com-

parators of the PIC family.

At Chapter 12, “Section 4”, this book switches gear to the PIC18 family. The author moves to C-language. The author pitches this switch as a significant advance. I find this position odd, as the main differences are simply an increase in memory and instruction sets, much of which is invisible below C. Some discussion of the instruction set is included nevertheless. Perhaps this attitude reflects a change in Wilmschurst’s thinking as he moved to C? Why the chapter is called “Smarter systems...” is anybody’s guess.

Chapter 13 simply describes the PIC18 series peripherals. It is largely a rehash of the data sheet.

Chapter 14 is “Introducing C”. It is open to the classic criticism: It is too brief to truly introduce *any* language! It also introduces the **goto** construct with little explanation beyond saying it is not a well-loved thing in C circles. The quintessential C break and continue constructs, no less dubious but ever more popular than goto, get introduced in the next chapter. While is used to create an infinite loop, odd in C terms but routine in embedded terms (kind of assumes the reader gets this from the assembly part of the book, hate to read it out of order). The ideas and impact of [14], [15], and [16] do not appear.

At this point I switch to considering this Wilmschurst book as being vastly less pedagogically sound than [8].

Chapter 15, “C and the embedded environment”, is partly a description of C-customisations to support the processor (although this is not really explained), and partly a simple introduction of more of the C language. Processor-specific library functions are used, but their processor-specificity is not noted.

Chapter 16, “Acquiring and using data with C”, continues an introduction to C and programming and ADCs. Pointers, arrays and strings appear here in about 2 pages. (Phew.) He starts using serial communications in C. There is not any structured planning to any of this!

Chapter 17, “More C and the wider C environment”, is more of the same. The idea of calling assembly from C is introduced.

Chapter 18 introduces the reader to the idea of an RTOS. The chapter must cover many important ideas, synchronous and asynchronous tasks, deadlines and the definition of real-time, prioritisation, the key challenges of an RTOS such as interprocess communications, sandboxing, scheduling, etc. The chapter is a mish-mash, and threshold and key ideas are not identified well.

Chapter 19 introduces an RTOS. (Salvo.) The idea of a clock tick is introduced.

Chapter 20 concludes with a discussion of connectivity and networks. It is more a catalog of technology options than anything else.

References

- [1] Wayne Wolf, *Computers as Components: Principles of Embedded Computing System Design*, 2nd Edition, Morgan Kaufmann (Elsevier), 2008.
- [2] Martin P. Bates, *Programming 8-bit PIC Microcontrollers in C with Interactive Hardware Simulation*, Newnes/Elsevier, 2008. ISBN:978-0-7506-8960-1

- [3] Steven F. Barrett, *Embedded Systems Design with the Atmel AVR Microcontroller*, Morgan and Claypool, 2010. Part of the “Synthesis Lectures on Digital Circuits and Systems”, Mitchell A. Thornton, series editor.
- [4] Peatman, John B., *Embedded Design with the PIC 18F452 microcontroller*, Prentice-Hall, 2003.
- [5] Richard H Barnett, *Embedded C Programming and the Microchip PIC*, Clifton Park, N.Y., Thomson/Delmar Learning, 2004.
- [6] Peter Spasov, *Microcontroller Technology: The 68HC11 and 68HC12*, 5th Edition, Prentice Hall, 2004.
- [7] Jack Ganssle, *The Art of Designing Embedded Systems*, Second Edition, Newnes, 2008.
- [8] Tim Wilmshurst, *An Introduction to the Design of Small-scale Embedded Systems: With Examples From PIC, 80C51 And 68HC05/08 Microcontrollers*, Palgrave Macmillan, 2001.
- [9] Tim Wilmshurst, *Designing Embedded Systems with PIC Microcontrollers: Principles and Applications*, Newnes; 2nd edition, 2009.
- [10] Brian W. Kernighan and Dennis M. Ritchie, *The C Programming Language*, 2nd Edition, Prentice-Hall, 1988.
- [11] Meyer, J. & Land, R. (2006). “Threshold concepts and troublesome knowledge: An introduction”, in J. Meyer & R. Land (eds) *Overcoming barriers to student understanding: Threshold concepts and troublesome knowledge*. London: Routledge.
- [12] Jackson, M., *Principles of Program Design*, Academic Press, 1975.
- [13] Jackson, M., *Software Requirements and Specifications: a lexicon of practice, principles and prejudices*, Addison-Wesley, 1995.
- [14] *MISRA-C:2004 Guidelines for the use of the C language in critical systems*, published by MIRA Limited, Warwickshire, UK, October 2004. w.misra-c.com
- [15] Andrew Koenig, *C Traps and Pitfalls*, Addison-Wesley Professional, 1989.
- [16] L. Hatton, *Safer C: Developing Software for High-integrity and Safety-critical Systems*, McGraw-Hill Education, 1994.
- [17] Jonathan Jacky, *The Way of Z: Practical Programming with Formal Methods*, Cambridge University Press, 1996.
- [18] James Rumbaugh, Ivar Jacobson, and Grady Booch, *Unified Modeling Language Reference Manual*, 2nd Edition, Addison-Wesley Professional, 2010.
- [19] Clive L. Dym and Patrick Little, *Engineering Design: A Project-Based Introduction*, 2nd Edition, Wiley, 2004.
- [20] MIX Software, Inc., *Using Multi-C: A portable multithreaded C programming library*, Prentice-Hall, 1994.