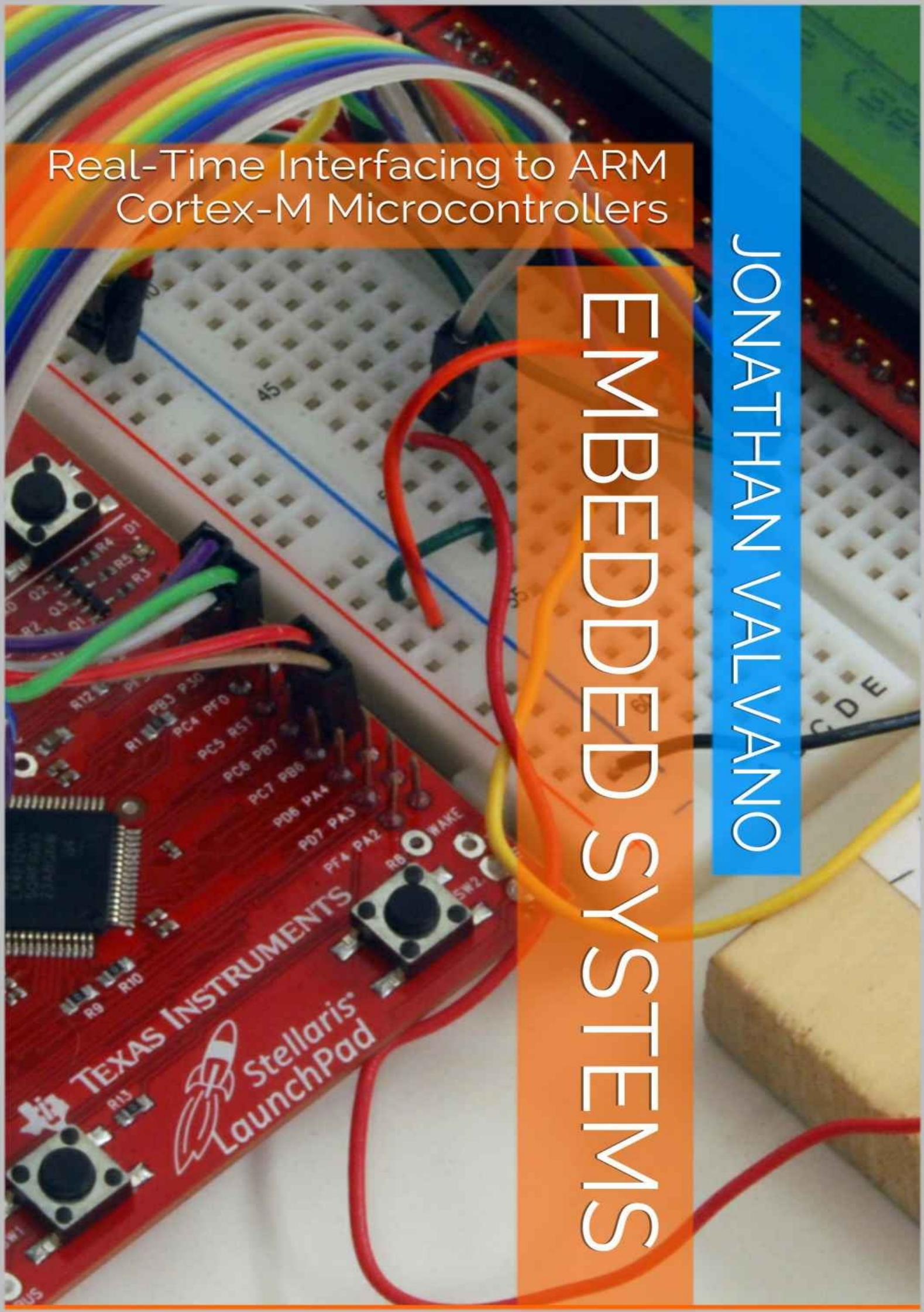
Real-Time Interfacing to ARM Cortex-M Microcontrollers
EMBEDDED SYSTEMS
JONATHAN VALVANO
TEXAS INSTRUMENTS
Stellaris LaunchPad

EMBEDDED SYSTEMS:

REAL-TIME INTERFACING TO ARM® CORTEX™-M MICROCONTROLLERS

Volume 2

Fourth Edition,

July 2014

Jonathan W. Valvano

Fourth edition

2nd Printing

July 2014

In order to reduce costs, this college textbook has been self-published. For more information about my classes, my research, and my books, see http://users.ece.utexas.edu/~valvano/

For corrections and comments, please contact me at: valvano@mail.utexas.edu. Please cite this book as: J. W. Valvano, Embedded Systems: Real-Time Interfacing to ARM ® Cortex™-M Microcontrollers, http://users.ece.utexas.edu/~valvano/, ISBN: 978-1463590154, 2014.

ISBN-13: 978-1463590154

ISBN-10: 1463590156

Table of Contents

Preface to Third Edition

There are a new features added to this third edition. The new development platform based on the TM4C123 is called Tiva LaunchPad. Material in this book on the TM4C also applies to the LM4F because Texas Instruments rebranded the LM4F series as TM4C (same chips new name), and rebranded StellarisWare™ as TivaWare™. These new microcontrollers run at 80 MHz, include single-precision floating point, have two 12-bit ADCs, and support DMA and USB. A wonderful feature of these new boards is their low cost. As of December 2013, the boards are available on TI.com as part number EK-TM4C123GXL for $12.99. They are also available from $13 to $24 at regular electronics retailers like arrow.com, newark.com, mouser.com, and digikey.com. The book can be used with either a LM3S or TM4C microcontroller. Although this edition now focuses on the M4, the concepts still apply to the M3, and the web site associated with this book has example projects based on the LM3S811, LM3S1968, and LM3S8962.

Preface to Fourth Edition

This fourth edition includes the new TM4C1294-based LaunchPad. Most of the code in the book is specific for the TM4C123-based LaunchPad. However, the book website includes corresponding example projects for the LM3S811, LM3S1968, LM4F120, and TM4C1294, which are ARM ® Cortex™-M microcontrollers from Texas Instruments. There are now two lost-cost development platforms called Tiva LaunchPad. The EK-TM4C123GXL LaunchPad retails for $12.99, and the EK-TM4C1294XL Connected LaunchPad retails for $19.99. The various LM3S, LM4F and TM4C microcontrollers are quite similar, so this book along with the example code on the web can be used for any of these microcontrollers. Compared to the TM4C123, the new TM4C1294 microcontroller runs faster, has more RAM, has more ROM, includes Ethernet, and has more I/O pins. This fourth edition switches the syntax from C to the industry-standard C99, adds a line-tracking robot, designs an integral controller for a DC motor, and includes an expanded section on wireless communication and Internet of Things.

Preface

Embedded systems are a ubiquitous component of our everyday lives. We interact with hundreds of tiny computers every day that are embedded into our houses, our cars, our toys, and our work. As our world has become more complex, so have the capabilities of the microcontrollers embedded into our devices. The ARM ® Cortex™-M family represents a new class of microcontrollers much more powerful than the devices available ten years ago. The purpose of this book is to present the design methodology to train young engineers to understand the basic building blocks that comprise devices like a cell phone, an MP3 player, a pacemaker, antilock brakes, and an engine controller.

This book is the second in a series of three books that teach the fundamentals of embedded systems as applied to the ARM ® Cortex™-M family of microcontrollers. The three books are primarily written for undergraduate electrical and computer engineering students. They could also be used for professionals learning the ARM platform. The first book Embedded Systems: Introduction to ARM Cortex-M Microcontrollers is an introduction to computers and interfacing focusing on assembly language and C programming. This second book focuses on interfacing and the design of embedded systems. The third book Embedded Systems: Real-Time Operating Systems for ARM Cortex-M Microcontrollers is an advanced book focusing on operating systems, high-speed interfacing, control systems, and robotics.

An embedded system is a system that performs a specific task and has a computer embedded inside. A system is comprised of components and interfaces connected together for a common purpose. This book presents components, interfaces and methodologies for building systems. Specific topics include the architecture of microcontrollers, design methodology, verification, hardware/software synchronization, interfacing devices to the computer, timing diagrams, real-time operating systems, data collection and processing, motor control, analog filters, digital filters, real-time signal processing, wireless communication, and the internet of things.

In general, the area of embedded systems is an important and growing discipline within electrical and computer engineering. The educational market of embedded systems has been dominated by simple microcontrollers like the PIC, the 9S12, and the 8051. This is because of their market share, low cost, and historical dominance. However, as problems become more complex, so must the systems that solve them. A number of embedded system paradigms must shift in order to accommodate this growth in complexity. First, the number of calculations per second will increase from millions/sec to billions/sec. Similarly, the number of lines of software code will also increase from thousands to millions. Thirdly, systems will involve multiple microcontrollers supporting many simultaneous operations. Lastly, the need for system verification will continue to grow as these systems are deployed into safety critical applications. These changes are more than a simple growth in size and bandwidth. These systems must employ parallel programming, high-speed synchronization, real-time operating systems, fault tolerant design, priority interrupt handling, and networking. Consequently, it will be important to provide our students with these types of design experiences. The ARM platform is both low cost and provides the high performance features required in future embedded systems. Although the ARM market share is currently not huge, its share will grow. Furthermore, students

trained on the ARM will be equipped to design systems across the complete spectrum from simple to complex. The purpose of writing these three books at this time is to bring engineering education into the 21st century.

This book employs many approaches to learning. It will not include an exhaustive recapitulation of the information in data sheets. First, it begins with basic fundamentals, which allows the reader to solve new problems with new technology. Second, the book presents many detailed design examples. These examples illustrate the process of design. There are multiple structural components that assist learning. Checkpoints, with answers in the back, are short easy to answer questions providing immediate feedback while reading. Simple homework, with answers to the odd questions on the web, provides more detailed learning opportunities. The book includes an index and a glossary so that information can be searched. The most important learning experiences in a class like this are of course the laboratories. Each chapter has suggested lab assignments. More detailed lab descriptions are available on the web. Specifically, look at the lab assignments for EE445L and EE445M.

There is a web site accompanying this book **http://users.ece.utexas.edu/~valvano/arm**. Posted here are ARM Keil™ uVision® projects for each the example programs in the book. Code Composer Studio™ versions are also available for most examples. You will also find data sheets and Excel spreadsheets relevant to the material in this book.

These three books will cover embedded systems for ARM ® Cortex™-M microcontrollers with specific details on the LM3S811, LM3S1968, LM3S8962, LM4F120, TM4C123, and TM4C1294. Most of the topics can be run on the low-cost TM4C123. Ethernet examples can be run on the LM3S8962 and TM4C1294. In these books the terms **LM3S** and **LM4F** and **TM4C** will refer to any of the Texas Instruments ARM ® Cortex™-M based microcontrollers. Although the solutions are specific for the **LM3S LM4F** and **TM4C** families, it will be possible to use these books for other ARM derivatives.

Acknowledgements

I owe a wonderful debt of gratitude to Daniel Valvano. He wrote and tested most of the software examples found in this book. Secondly, he created and maintains the example web site, **http://users.ece.utexas.edu/~valvano/arm.** Lastly, he meticulously proofread this manuscript.

Many shared experiences contributed to the development of this book. First I would like to acknowledge the many excellent teaching assistants I have had the pleasure of working with. Some of these hard-working, underpaid warriors include Pankaj Bishnoi, Rajeev Sethia, Adson da Rocha, Bao Hua, Raj Randeri, Santosh Jodh, Naresh Bhavaraju, Ashutosh Kulkarni, Bryan Stiles, V. Krishnamurthy, Paul Johnson, Craig Kochis, Sean Askew, George Panayi, Jeehyun Kim, Vikram Godbole, Andres Zambrano, Ann Meyer, Hyunjin Shin, Anand Rajan, Anil Kottam, Chia-ling Wei, Jignesh Shah, Icaro Santos, David Altman, Nachiket Kharalkar, Robin Tsang, Byung Geun Jun, John Porterfield, Daniel Fernandez, Deepak Panwar, Jacob Egner, Sandy Hermawan, Usman Tariq, Sterling Wei, Seil Oh, Antonius Keddis, Lev Shuhatovich, Glen Rhodes, Geoffrey Luke, Karthik Sankar, Tim Van Ruitenbeek, Raffaele Cetrulo, Harshad Desai, Justin Capogna, Arindam Goswami, Jungho Jo, Mehmet Basoglu, Kathryn Loeffler, Evgeni Krimer, Nachiappan Valliappan, Razik Ahmed, Sundeep Korrapati, Song Zhang, Zahidul Haq, Matthew Halpern, Cruz Monrreal II, Pohan Wu, Saugata Bhattacharyya, Omar Baca Aditya Saraf, and Mahesh Srinivasan. These teaching assistants have contributed greatly to the contents of this book and particularly to its laboratory assignments. Since 1981, I estimate I have taught embedded systems to over 5000 students. My students have recharged my energy each semester with their enthusiasm, dedication, and quest for knowledge. I have decided not to acknowledge them all individually. However, they know I feel privileged to have had this opportunity.

Next, I appreciate the patience and expertise of my fellow faculty members here at the University of Texas at Austin. From a personal perspective Dr. John Pearce provided much needed encouragement and support throughout my career. In addition, Drs. John Cogdell, John Pearce, and Francis Bostick helped me with analog circuit design. The book and accompanying software include many finite state machines derived from the digital logic examples explained to me by Dr. Charles Roth. Over the last few years, I have enjoyed teaching embedded systems with Drs. Ramesh Yerraballi, Mattan Erez, Andreas Gerstlauer, Vijay Janapa Reddi, Nina Telang, and Bill Bard. Bill has contributed to both the excitement and substance of our laboratory based on this book. With pushing from Bill and TAs Robin, Glen, Lev, and John, we have added low power, PCB layout, systems level design, surface mount soldering, and wireless communication to our lab experience. You can see descriptions and photos of our EE445L design competition at **http://users.ece.utexas.edu/~valvano/**. Many of the suggestions and corrections from Chris Shore and Drew Barbier of ARM about Volume 1 applied equally to this volume. Austin Blackstone created and debugged the Code Composer Studio™ versions of the example programs posted on the web. Austin also taught me how to run the CC3000 and CC3100 WiFi examples on the LaunchPad.

Sincerely, I appreciate the valuable lessons of character and commitment taught to me by my parents and grandparents. I recall how hard my parents and grandparents worked to make the world a better place for the next generation. Most significantly, I acknowledge the love, patience and support of my wife, Barbara, and my children, Ben Daniel and Liz. In particular, Dan designed and tested most of the LM3S and LM4F/TM4C software presented in this book.

By the grace of God, I am truly the happiest man on the planet, because I am surrounded by these fine people. Good luck.

Jonathan W. Valvano

> *The true engineering experience occurs not with your eyes and ears, but rather with your fingers and elbows. In other words, engineering education does not happen by listening in class or reading a book; rather it happens by designing under the watchful eyes of a patient mentor. So, go build something today, then show it to someone you respect!*

1. Introduction to Embedded Systems

Chapter 1 objectives are to:

- Review computer architecture
- Introduce embedded systems
- Present a process for design
- Discuss practical aspects of digital logic, including open collector
- Review how numbers are represented in binary
- Define ethics

The overall objective of this book is to teach the design of embedded systems. It is effective to learn new techniques by doing them. But, the dilemma in teaching a laboratory-based topic like embedded systems is that there is a tremendous volume of details that first must be learned before hardware and software systems can be designed. The approach taken in this book is to learn by doing, starting with very simple problems and building up to more complex systems later in the book.

In this chapter we begin by introducing some terminology and basic components of a computer system. In order to understand the context of our designs, we will overview the general characteristics of embedded systems. It is in these discussions that we develop a feel for the range of possible embedded applications. Next we will present a template to guide us in design. We begin a project with a requirements document. Embedded systems interact with physical devices. Often, we can describe the physical world with mathematical models. If a model is available, we can then use it to predict how the embedded system will interface with the real world. When we write software, we mistakenly think of it as one dimensional, because the code looks sequential on the computer screen. Data flow graphs, call graphs, and flow charts are multidimensional graphical tools to understand complex behaviors. Because courses taught using this book typically have a lab component, we will review some practical aspects of digital logic.

Next, we show multiple ways to represent numbers in the computer. Choosing the correct format is necessary to implement efficient and correct solutions. Fixed-point numbers are the typical way embedded systems represent non-integer values. Floating-point numbers, typically used to represent non-integer values on a general purpose computer, will also be presented.

Because embedded systems can be employed in safety critical applications, it is important for engineers be both effective and ethical. Throughout the book we will present ways to verify the system is operating within specifications.

1.1. Computer Architecture

1.1.1. Computers, microprocessors, memory, and microcontrollers

A **computer** combines a processor, random access memory (RAM), read only memory (ROM), and input/output (I/O) ports. The common bus in Figure 1.1 defines the von Neumann architecture, where instructions are fetched from ROM on the same bus as data fetched from RAM. **Software** is an ordered sequence of very specific instructions that are stored in memory, defining exactly what and when certain tasks are to be performed. The **processor** executes the software by retrieving and interpreting these instructions one at a time. A **microprocessor** is a small processor, where small refers to size (i.e., it fits in your hand) and not computational ability. For example, Intel Xeon, AMD FX and Sun SPARC are microprocessors. An ARM ® Cortex™-M microcontroller includes a processor together with the bus and some peripherals. A **microcomputer** is a small computer, where again small refers to size (i.e., you can carry it) and not computational ability. For example, a desktop PC is a microcomputer.

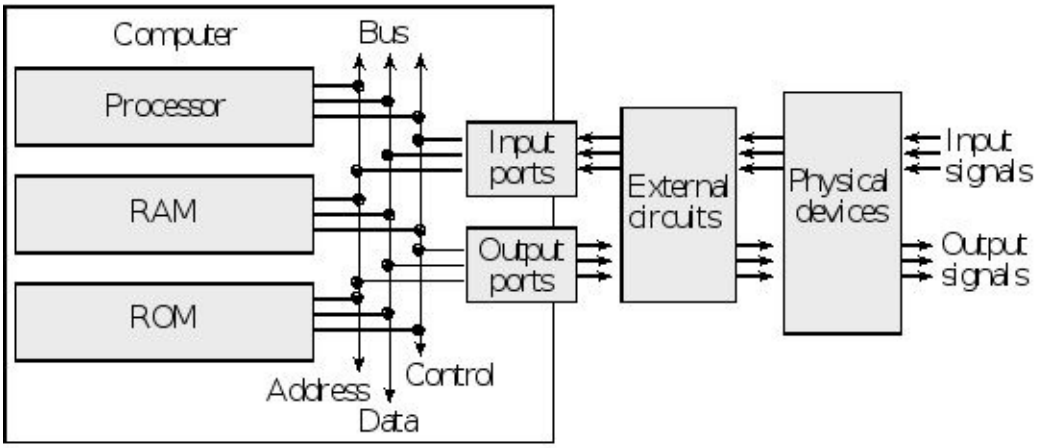

Figure 1.1. The basic components of a computer system include processor, memory and I/O.

A very small microcomputer, called a **microcontroller**, contains all the components of a computer (processor, memory, I/O) on a single chip. As shown in Figure 1.2, the Atmel ATtiny, the Texas Instruments MSP430, and the Texas Instruments TM4C123 are examples of microcontrollers. Because a microcomputer is a small computer, this term can be confusing because it is used to describe a wide range of systems from a 6-pin ATtiny4 running at 1 MHz with 512 bytes of program memory to a personal computer with state-of-the-art 64-bit multi-core processor running at multi-GHz speeds having terabytes of storage.

The computer can store information in **RAM** by writing to it, or it can retrieve previously stored data by reading from it. Most RAMs are **volatile**; meaning if power is interrupted and restored the information in the RAM is lost. Most microcontrollers have **static RAM** (SRAM) using six metal-oxide-semiconductor field-effect transistors (MOSFET) to create each memory bit. Four transistors are used to create two cross-coupled inverters that store the binary information, and the other two are used to read and write the bit.

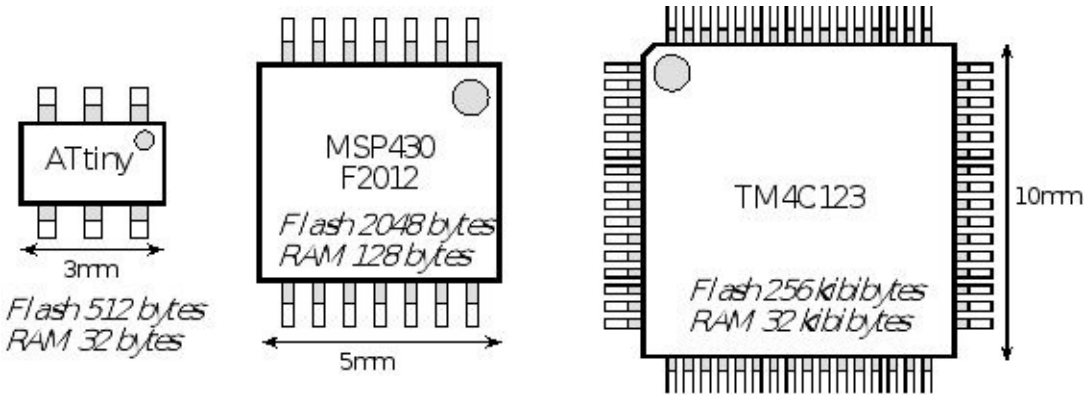

Figure 1.2. A microcontroller is a complete computer on a single chip.

Information is programmed into **ROM** using techniques more complicated than writing to RAM. From a programming viewpoint, retrieving data from a ROM is identical to retrieving data from RAM. ROMs are nonvolatile; meaning if power is interrupted and restored the information in the ROM is retained. Some ROMs are programmed at the factory and can never be changed. A Programmable ROM (PROM) can be erased and reprogrammed by the user, but the erase/program sequence is typically 10000 times slower than the time to write data into a RAM. PROMs used to need ultraviolet light to erase, and then we programmed them with voltages. Now, most PROMs now are electrically erasable (EEPROM), which means they can be both erased and programmed with voltages. We cannot program ones into the ROM. We first erase the ROM, which puts ones into its storage memory, and then we program the zeros as needed. **Flash ROM** is a popular type of EEPROM. Each flash bit requires only two MOSFET transistors. The input (gate) of one transistor is electrically isolated, so if we trap charge on this input, it will remain there for years. The other transistor is used to read the bit by sensing whether or not the other transistor has trapped charge. In regular EEPROM, you can erase and program individual bytes. Flash ROM must be erased in large blocks. On many of LM3S/LM4F/TM4C microcontrollers, we can erase the entire ROM or just a 1024-byte block. Because flash is smaller than regular EEPROM, most microcontrollers have a large flash into which we store the software. For all the systems in this book, we will store instructions and constants in flash ROM and place variables and temporary data in static RAM.

Checkpoint 1.1: What are the differences between a microcomputer, a microprocessor and a microcontroller?

Checkpoint 1.2: Which has a higher information density on the chip in bits per mm^2: static RAM or flash ROM? Assume all MOSFETs are approximately the same size in mm^2.

Observation: Memory is an object that can transport information across time.

The external devices attached to the microcontroller provide functionality for the system. An **input port** is hardware on the microcontroller that allows information about the external world to be entered into the computer. The microcontroller also has hardware called an **output port** to send information out to the external world. Most of the pins shown in Figure 1.2 are input/output ports.

An **interface** is defined as the collection of the I/O port, external electronics, physical devices, and the software, which combine to allow the computer to communicate with the external world. An example of an input interface is a switch, where the operator toggles the switch, and the software can recognize the switch position. An example of an output interface is a light-emitting diode (LED), where the software can turn the light on and off, and the operator can see whether or not the light is shining. There is a wide range of possible inputs and outputs, which can exist in either digital or analog form. In general, we can classify I/O interfaces into four categories

Parallel - binary data are available simultaneously on a group of lines

Serial - binary data are available one bit at a time on a single line

Analog - data are encoded as an electrical voltage, current, or power

Time - data are encoded as a period, frequency, pulse width, or phase shift

Checkpoint 1.3: What are the differences between an input port and an input interface?

Checkpoint 1.4: List three input interfaces available on a personal computer.

Checkpoint 1.5: List three output interfaces available on a personal computer.

In this book, numbers that start with 0x (e.g., 0x64) are specified in **hexadecimal**, which is base 16 ($0x64 = 6*16^1+4*16^0 = 100$). Some assemblers start hexadecimal numbers with $ (e.g., $64). Other assembly languages add an "H" at the end to specify hexadecimal (e.g., 64H or 64h). Yale Patt's LC3 assembler uses just the "x" (e.g., x64).

In a system with **memory mapped I/O**, as shown in Figure 1.1, the I/O ports are connected to the processor in a manner similar to memory. I/O ports are assigned addresses, and the software accesses I/O using reads and writes to the specific I/O addresses. The software inputs from an input port using the same instructions as it would if it were reading from memory. Similarly, the software outputs from an output port using the same instructions as it would if it were writing to memory. A **bus** is defined as a collection of signals, which are grouped for a common purpose. The bus has three types of signals: address signals, data signals, and control signals. Together, the bus directs the data transfer between the various modules in the computer. There are five buses on ARM ® Cortex™-M processor, as illustrated in Figure 1.3. The address specifies which module is being accessed, and the data contains the information being transferred. The control signals specify the direction of transfer, the size of the data, and timing information. The **ICode bus** is used to fetch instructions from flash ROM. All ICode bus fetches contain 32 bits of data, which may be one or two instructions. The **DCode bus** can fetch data or debug information from flash ROM. The **system bus** can read/write data from RAM or I/O ports. The **private peripheral bus** (PPB) can access some of the

common peripherals like the interrupt controller. The multiple-bus architecture allows simultaneous bus activity, greatly improving performance over single-bus architectures. For example, the processor can simultaneously fetch an instruction out of flash ROM using the ICode bus while it writes data into RAM using the system bus. From a software development perspective, the fact that there are multiple buses is transparent. This means we write code like we would on any computer, and the parallel operations occur automatically. The TM4C123 has 256 kibibytes (2^{18} bytes) of flash ROM and 32768 bytes of RAM. The TM4C1294 has 1024 kibibytes (2^{20} bytes) of flash ROM and 256 kibibytes of RAM. The RAM begins at 0x2000.0000, and the flash ROM begins at 0x0000.0000.

	TM4C123			TM4C1294
0x0000.0000	256k		**0x0000.0000**	1024k
…	Flash		**…**	Flash
0x0003.FFFF	ROM		**0x000F.FFFF**	ROM
0x2000.0000	32k		**0x2000.0000**	256k
…	Static		**…**	Static
0x2000.7FFF	RAM		**0x2003.FFFF**	RAM

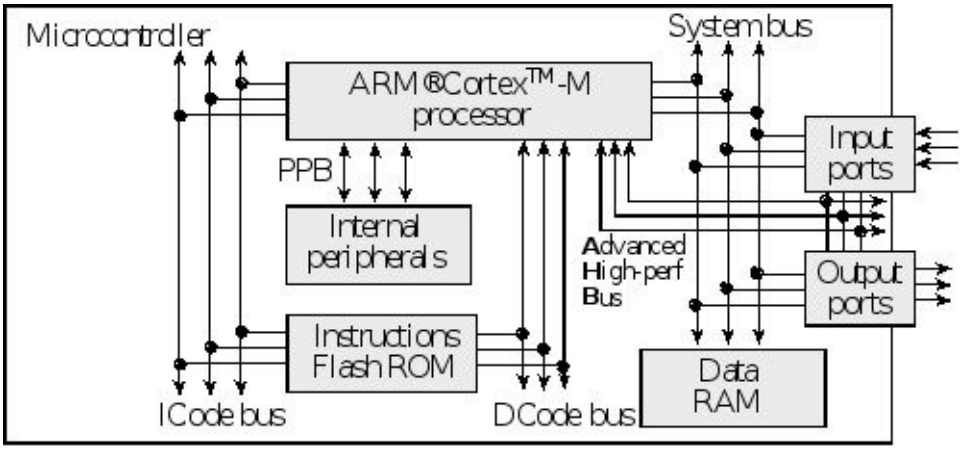

Figure 1.3. Harvard architecture of an ARM® Cortex™-M-based microcontroller.

The Cortex™-M4 series includes an additional bus called the Advanced High-Performance Bus (AHB or AHPB). This bus improves performance when communicating with high-speed I/O devices like USB. In general, the more operations that can be performed in parallel, the faster the processor will execute. In summary:

ICode bus Fetch opcodes from ROM

DCode bus	Read constant data from ROM
System bus	Read/write data from RAM or I/O, fetch opcode from RAM
PPB	Read/write data from internal peripherals like the NVIC
AHB	Read/write data from high-speed I/O and parallel ports (M4 only)

Instructions and data are accessed the same way on a von Neumann machine. The Cortex™-M processor is a Harvard architecture because instructions are fetched on the ICode bus and data accessed on the system bus. The address signals on the ARM ® Cortex™-M processor include 32 lines, which together specify the memory address (0x0000.0000 to 0xFFFF.FFFF) that is currently being accessed. The address specifies both which module (input, output, RAM, or ROM) as well as which cell within the module will communicate with the processor. The data signals contain the information that is being transferred and also include 32 bits. However, on the system bus it can also transfer 8-bit or 16-bit data. The control signals specify the timing, the size, and the direction of the transfer. We call a complete data transfer a bus cycle. Two types of transfers are allowed, as shown in Table 1.1. In most systems, the processor always controls the address (where to access), the direction (read or write), and the control (when to access.)

Type	*Address Driven by*	*Data Driven by*	*Transfer*
Read Cycle	Processor	RAM, ROM or Input	Data copied to processor
Write Cycle	Processor	Processor	Data copied to output or RAM

Table 1.1. Simple computers generate two types of bus cycles.

A **read cycle** is used to transfer data into the processor. During a read cycle the processor first places the address on the address signals, and then the processor issues a read command on the control signals. The slave module (RAM, ROM, or I/O) will respond by placing the contents at that address on the data signals, and lastly the processor will accept the data and disable the read command.

The processor uses a **write cycle** to store data into memory or I/O. During a write cycle the processor also begins by placing the address on the address signals. Next, the processor places the information it wishes to store on the data signals, and then the processor issues a write command on the control signals. The memory or I/O will respond by storing the information into the proper place, and after the processor is sure the data has been captured, it will disable the write command.

The **bandwidth** of an I/O interface is the number of bytes/sec that can be transferred. If we wish to transfer data from an input device into RAM, the software must first transfer the data from input to the processor, then from the processor into RAM. On the ARM, it will take multiple instructions to perform this transfer. The bandwidth depends both on the speed of the I/O hardware and the software performing the transfer. In some microcontrollers like the TM4C123 and TM4C1294, we will be able to transfer data directly from input to RAM or RAM to output using direct memory access (DMA). When using DMA the software time is removed, so the bandwidth only depends on the speed of the I/O hardware. Because DMA is faster, we will use this method to interface high bandwidth devices like disks and networks. During a **DMA read cycle** data flows directly from the memory to the output device. General purpose computers also support DMA allowing data to be transferred from memory to memory. During a **DMA write cycle** data flows directly from the input device to memory.

Input/output devices are important in all computers, but they are especially significant in an embedded system. In a computer system with **I/O-mapped I/O**, the control bus signals that activate the I/O are separate from those that activate the memory devices. These systems have a separate address space and separate instructions to access the I/O devices. The original Intel 8086 had four control bus signals MEMR, MEMW, IOR, and IOW. MEMR and MEMW were used to read and write memory, while IOR and IOW were used to read and write I/O. The Intel x86 refers to any of the processors that Intel has developed based on this original architecture. Even though we do not consider the personal computer (PC) an embedded system, there are embedded systems developed on this architecture. One such platform is called the PC/104 Embedded-PC. The Intel x86 processors continue to implement this separation betweenmemory and I/O. Rather than use the regular memory access instructions, the Intel x86 processor uses special **in** and **out** instructions to access the I/O devices. The advantages of I/O-mapped I/O are that software can not inadvertently access I/O when it thinks it is accessing memory. In other words, it protects I/O devices from common software bugs, such as bad pointers, stack overflow, and buffer overflows. In contrast, systems with memory-mapped I/O are easier to design, and the software is easier to write.

1.1.2. Cortex™-M processor

The ARM ® Cortex™-M processor has four major components, as illustrated in Figure 1.4. There are four **bus interface units** (BIU) that read data from the bus during a read cycle and write data onto the bus during a write cycle. Both the TM4C123 and TM4C1294 microcontrollers support DMA. The BIU always drives the address bus and the control signals of the bus. The **effective address register** (EAR) contains the memory address used to fetch the data needed for the current instruction. Cortex™-M microcontrollers executeThumb ® instructions extended with Thumb-2 technology. An overview of these instructions will be presented in Chapter 2. The Cortex™-M4F microcontrollers include a floating-point processor. However, in this book we will focus on integer and fixed-point arithmetic.

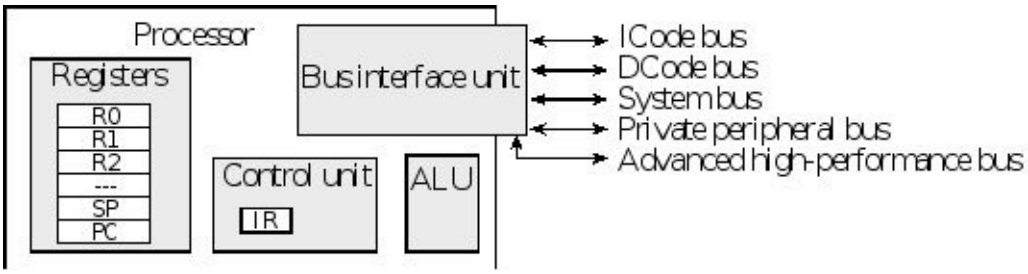

Figure 1.4. The four basic components of a processor.

The **control unit** (CU) orchestrates the sequence of operations in the processor. The CU issues commands to the other three components. The **instruction register** (IR) contains the operation code (or op code) for the current instruction. When extended with Thumb-2 technology, op codes are either 16 or 32 bits wide. In an embedded system the software is converted to machine code, which is a list of instructions, and stored in nonvolatile flash ROM. As instructions are fetched, they are placed in a **pipeline**. This allows instruction fetching to run ahead of execution. Instructions are fetched in order and executed in order. However, it can execute one instruction while fetching the next.

The **registers** are high-speed storage devices located in the processor (e.g., R0 to R15). Registers do not have addresses like regular memory, but rather they have specific functions explicitly defined by the instruction. Registers can contain data or addresses. The **program counter** (PC) points to the memory containing the instruction to execute next. On the ARM ® Cortex™-M processor, the PC is register 15 (R15). In an embedded system, the PC usually points into nonvolatile memory like flash ROM. The information stored in nonvolatile memory (e.g., the instructions) is not lost when power is removed. The **stack pointer** (SP) points to the RAM, and defines the top of the stack. The stack implements last in first out (LIFO) storage. On the ARM ® Cortex™-M processor, the SP is register 13 (R13). The stack is an extremely important component of software development, which can be used to pass parameters, save temporary information, and implement local variables. The **program status register** (PSR) contains the status of the previous operation, as well as some operating mode flags such as the interrupt enable bit. This register is called the flag register on the Intel computers.

The **arithmetic logic unit** (ALU) performs arithmetic and logic operations. Addition, subtraction, multiplication and division are examples of arithmetic operations. And, or, exclusive or, and shift are examples of logical operations.

Checkpoint 1.6: For what do the acronyms CU DMA BIU ALU stand?

In general, the execution of an instruction goes through four phases. First, the computer fetches the machine code for the instruction by reading the value in memory pointed to by the program counter (PC). Some instructions are 16 bits, while others are 32 bits. After each instruction is fetched, the PC is incremented to the next instruction. At this time, the instruction is decoded, and the effective address is determined (EAR). Many instructions require additional data, and during phase 2 the data is retrieved from memory at the effective address. Next, the actual function for this instruction is performed. During the last phase, the results are written back to memory. All instructions have a phase 1, but the other three phases may or may not occur for any specific instruction.

On the ARM ® Cortex™-M processor, an instruction may read memory or write memory, but it does not both read and write memory in the same instruction. Each of the phases may require one or more bus cycles to complete. Each bus cycle reads or writes one piece of data. Because of the multiple bus architecture, most instructions execute in one or two cycles. For more information on the time to execute instructions, see Table 3.1 in the Cortex™-M Technical Reference Manual. ARM is a **reduced instruction set computer** (RISC), which achieves high performance by implementing very simple instructions that run extremely fast.

Phase	*Function*	*Bus*	*Address*	*Comment*
1	Instruction fetch	Read	PC++	Put into IR
2	Data read	Read	EAR	Data passes through ALU
3	Operation	-	-	ALU operations, set PSR
4	Data store	Write	EAR	Results stored in memory

Table 1.2. Four phases of execution.

An instruction on a RISC processor does not have both a phase 2 data read cycle and a phase 4 data write cycle. In general, a RISC processor has a small number of instructions, instructions have fixed lengths, instructions execute in 1 or 2 bus cycles, there are only a few instructions (e.g., load and store) that can access memory, no one instruction can both read and write memory in the same instruction, there are many identical general purpose registers, and there are a limited number of addressing modes.

Conversely, processors are classified as **complex instruction set computers** (CISC), because one instruction is capable of performing multiple memory operations. For example, CISC processors have instructions that can both read and write memory in the same instruction. Assume **Data** is an 8-bit memory variable. The following Intel 8080 instruction will increment the 8-bit variable, requiring a read memory cycle, ALU operation, and then a write memory cycle.

INR Data ; Intel 8080

Other CISC processors like the 6800, 9S12, 8051, and Pentium also have memory increment instructions requiring both a phase 2 data read cycle and a phase 4 data write cycle. In general, a CISC processor has a large number of instructions, instructions have varying lengths, instructions execute in varying times, there are many instructions that can access memory, the processor can both read and write memory in one instruction, the processor has fewer and more specialized registers, and the processor has many addressing modes.

1.1.3. History

In 1968, two unhappy engineers named Bob Noyce and Gordon Moore left the Fairchild Semiconductor Company and created their own company, which they called Integrated Electronics (Intel). Working for Intel in 1971, Federico Faggin, Ted Hoff, and Stan Mazor invented the first single chip microprocessor, the Intel 4004. It was a four-bit processor designed to solve a very specific application for a Japanese company called Busicon. Busicon backed out of the purchase, so Intel decided to market it as a "general purpose" microprocessing system. The product was a success, which lead to a series of more powerful microprocessors: the Intel 8008 in 1974, the Intel 8080 also in 1974. Both the Intel 8008 and the Intel 8080 were 8-bit microprocessors that operated from a single +5V power supply using N-channel metal-oxide semiconductor (NMOS) technology.

Seeing the long term potential for this technology, Motorola released its MC6800 in 1974, which was also an 8-bit processor with about the same capabilities of the 8080. Although similar in computing power, the 8080 and 6800 had very different architectures. The 8080 used isolated I/O and handled addresses in a fundamentally different way than data. Isolated I/O defines special hardware signals and special instructions for input/output. On the 8080, certain registers had capabilities designed for addressing, while other registers had capabilities for specific for data manipulation. In contrast, the 6800 used memory-mapped I/O and handled addresses and data in a similar way. As we defined earlier, input/output on a system with memory-mapped I/O is performed in a manner similar to accessing memory.

During the 1980s and 1990s, Motorola and Intel traveled down similar paths. The microprocessor families from both companies developed bigger and faster products: Intel 8085, 8088, 80x86, … and the Motorola 6809, 68000, 680x0… During the early 1980's another technology emerged, the microcontroller. In sharp contrast to the microprocessor family, which optimized computational speed and memory size at the expense of power and physical size, the microcontroller devices minimized power consumption and physical size, striving for only modest increases in computational speed and memory size. Out of the Intel architecture came the 8051 family (www.semiconductors.philips.com), and out of the Motorola architecture came the 6805, 6811, and 6812 microcontroller family (www.freescale.com). Many of the same fundamental differences that existed between the original 8-bit Intel 8080 and Motorola 6800 have persisted over forty years of microprocessor and microcontroller developments. In 1999, Motorola shipped its 2 billionth MC68HC05 microcontroller. In 2004, Motorola spun off its microcontroller products as Freescale Semiconductor. Microchip is a leading supplier of 8-bit microcontrollers.

The first ARM processor was conceived in the 1983 by Acorn Computers, which at the time was one of the leaders of business computers in the United Kingdom. The first chips were delivered in 1985. At that time ARM referred to Acorn RISC Machine. In 1990, a new company ARM Ltd was formed with Acorn, Apple, and VLSI Technology as founding partners, changing the ARM acronym to Advanced RISC Machine. As a company, the ARM business model involves the designing and licensing of intellectual property (IP) rather than the manufacturing and selling of actual semiconductor chips. ARM has sold 600 processor licenses to more than 200 companies. Virtually every company that manufacturers integrated circuits in the computer field produces a variant of the ARM processor. ARM currently dominates the high-performance low-power embedded system market. ARM processors account for approximately 90% of all embedded 32-bit RISC processors and are used in consumer electronics, including PDAs, cell phones, music players, hand-held game consoles, and calculators. The ARM Cortex-A is used in applications processors, such as smartphones. The ARM Cortex-R is appropriate for real-time applications, and ARM Cortex-M targets microcontrollers. Examples of microcontrollers built using the ARM ® Cortex™-M core are LM3S/TM4C by Texas Instruments, STM32 by STMicroelectronics, LPC17xx by NXP Semiconductors, TMPM330 by Toshiba, EM3xx by Ember, AT91SAM3 by Atmel, and EFM32 by Energy Micro. As of June 2014 over 50 billion ARM processors have shipped from over 950 companies.

What will the future unfold? One way to predict the future is to study the past. How embedded systems interact with humans has been and will continue to be critical. Improving the human experience has been the goal of many systems. Many predict the number of microcontrollers will soon reach into the trillions. As this happens, communication, security, energy, politics, resources, and economics will be become increasingly important. When there are this many computers, it will be possible to make guesses about how to change, then let a process like evolution select which changes are beneficial. In fact, a network of embedded systems with tight coupling to the real world, linked together for a common objective, is now being called a **cyber-physical system** (CPS).

One constant describing the history of computers is continuous change coupled with periodic monumental changes. Therefore, engineers must focus their education on fundamental principles rather than the voluminous details. They must embrace the concept of lifelong learning. Most humans are fundamentally good, but some are not. Therefore, engineers acting in an ethical manner can guarantee future prosperity of the entire planet.

1.2. Embedded Systems

An **embedded system** is an electronic system that includes a one or more microcontrollers that is configured to perform a specific dedicated application, drawn previously as Figure 1.1. To better understand the expression “embedded system,” consider each word separately. In this context, the word embedded means “a computer is hidden inside so one can’t see it.” The word “system” refers to the fact that there are many components which act in concert achieving the common goal. As mentioned earlier, input/output devices characterize the embedded system, allowing it to interact with the real world.

The software that controls the system is programmed or fixed into flash ROM and is not accessible to the user of the device. Even so, software maintenance is still extremely important. Software maintenance is verification of proper operation, updates, fixing bugs, adding features, and extending to new applications and end user configurations. Embedded systems have these four characteristics.

First, embedded systems typically perform a single function. Consequently, they solve a limited range of problems. For example, the embedded system in a microwave oven may be reconfigured to control different versions of the oven within a similar product line. But, a microwave oven will always be a microwave oven, and you can’t reprogram it to be a dishwasher. Embedded systems are unique because of the microcontroller’s I/O ports to which the external devices are interfaced. This allows the system to interact with the real world.

Second, embedded systems are tightly constrained. Typically, system must operate within very specific performance parameters. If an embedded system cannot operate with specifications, it is considered a failure and will not be sold. For example, a cell-phone carrier typically gets 832 radio frequencies to use in a city, a hand-held video game must cost less than $50, an automotive cruise control system must operate the vehicle within 3 mph of the set-point speed, and a portable MP3 player must operate for 12 hours on one battery charge.

Third, many embedded systems must operate in real-time. In a **real-time system**, we can put an upper bound on the time required to perform the input-calculation-output sequence. A real-time system can guarantee a worst case upper bound on the response time between when the new input information becomes available and when that information is processed. Another real-time requirement that exists in many embedded systems is the execution of periodic tasks. A periodic task is one that must be performed at equal time intervals. A real-time system can put a small and bounded limit on the time error between when a task should be run and when it is actually run. Because of the real-time nature of these systems, microcontrollers in the TM4C family have a rich set of features to handle all aspects of time.

The fourth characteristic of embedded systems is their small memory requirements as compared to general purpose computers. There are exceptions to this rule, such as those which process video or audio, but most have memory requirements measured in thousands of bytes. Over the years, the memory in embedded systems as increased, but the gap memory size between embedded systems and general purpose computers remains. The original microcontrollers had thousands of bytes of memory and the PC had millions. Now, microcontrollers can have millions of bytes, but the PC has billions.

There have been two trends in the microcontroller field. The first trend is to make microcontrollers smaller, cheaper, and lower power. The Atmel ATtiny, Microchip PIC, and Texas Instruments MSP430 families are good examples of this trend. Size, cost, and power are critical factors for high-volume products, where the products are often disposable. On the other end of the spectrum is the trend of larger RAM and ROM, faster processing, and increasing integration of complex I/O devices, such as Ethernet, radio, graphics, and audio. It is common for one device to have multiple microcontrollers, where the operational tasks are distributed and the microcontrollers are connected in a local area network (LAN). These high-end features are critical for consumer electronics, medical devices, automotive controllers, and military hardware, where performance and reliability are more important than cost. However, small size and low power continue as important features for all embedded systems.

The RAM is volatile memory, meaning its information is lost when power is removed. On some embedded systems a battery powers the microcontroller. When in the off mode, the microcontroller goes into low-power sleep mode, which means the information in RAM is maintained, but the processor is not executing. The MSP430 and ATtiny require less than a A of current in sleep mode.

Checkpoint 1.7: What is an embedded system?

Checkpoint 1.8: What goes in the RAM on a smartphone?

Checkpoint 1.9: Why does your smartphone need so much flash ROM?

The computer engineer has many design choices to make when building a real-time embedded system. Often, defining the problem, specifying the objectives, and identifying the constraints are harder than actual implementations. In this book, we will develop computer engineering design processes by introducing fundamental methodologies for problem specification, prototyping, testing, and performance evaluation.

A typical automobile now contains an average of ten microcontrollers. In fact, upscale homes may contain as many as 150 microcontrollers and the average consumer now interacts with microcontrollers up to 300 times a day. The general areas that employ embedded systems encompass every field of engineering:

- Consumer Electronics
- Home
- Communications
- Automotive
- Military
- Industrial
- Business
- Shipping

• Medical

• Computer components

In general, embedded systems have inputs, perform calculations, make decisions, and then produce outputs. The microcontrollers often must communicate with each other. How the system interacts with humans is often called the **human-computer interface** (HCI) or **man-machine interface** (MMI). To get a sense of what "embedded system" means we will present brief descriptions of four example systems.

Example 1.1: The goal of a **pacemaker** is to regulate and improve heart function. To be successful the engineer must understand how the heart works and how disease states cause the heart to fail. Its inputs are sensors on the heart to detect electrical activity, and its outputs can deliver electrical pulses to stimulate the heart. Consider a simple pacemaker with two sensors, one in the right atrium and the other in the right ventricle. The sensor allows the pacemaker to know if the normal heart contraction is occurring. This pacemaker has one right ventricular stimulation output. The embedded system analyzes the status of the heart deciding where and when to send simulation pulses. If the pacemaker recognizes the normal behavior of atrial contraction followed shortly by ventricular contraction, then it will not stimulate. If the pacemaker recognizes atrial contraction without a following ventricular contraction, then is will pace the ventricle shortly after each atrial contraction. If the pacemaker senses no contractions or if the contractions are too slow, then it can pace the ventricle at a regular rate. A pacemaker can also communicate via radio with the doctor to download past performance and optimize parameters for future operation. Some pacemakers can call the doctor on the phone when it senses a critical problem. Pacemakers are real-time systems because the time delay between atrial sensing and ventricular triggering is critical. Low power and reliability are important.

Example 1.2: The goal of a **smoke detector** is to warn people in the event of a fire. It has two inputs. One is a chemical sensor that detects the presence of smoke, and the other is a button that the operator can push to test the battery. There are also two outputs: an LED and the alarm. Most of the time, the detector is in a low-power sleep mode. If the test button is pushed, the detector performs a self-diagnostic and issues a short sound if the sensor and battery are ok. Once every 30 seconds, it wakes up and checks to see if it senses smoke. If it senses smoke, it will alarm. Otherwise it goes back to sleep.

Advanced smoke detectors should be able to communicate with other devices in the home. If one sensor detects smoke, all alarms should sound. If multiple detectors in the house collectively agree there is really a fire, they could communicate with the fire department and with the neighboring houses. To design and deploy a collection of detectors, the engineer must understand how fires start and how they spread. Smoke detectors are not real-time systems. However, reliability and low power are important.

Example 1.3: The goal of a **motor controller** is to cause a motor to spin in a desired manner. Sometimes we control speed, as in the cruise control on an automobile. Sometimes we control position as in moving paper through a printer. In a complex robotics system, we may need to simultaneously control multiple motors and multiple parameters such as position, speed, and torque. Torque control is important for building a robot that walks. The engineer must understand the mechanics of how the motor interacts with its world and the behavior of the interface electronics. The motor controller uses sensors to measure the current state of the motor, such as position, speed, and torque. The controller accepts input commands defining the desired operation. The system uses actuators, which are outputs that affect the motor. A typical actuator allows the system to set the electrical power delivered to the motor. Periodically, the microcontroller senses the inputs and calculates the power needed to minimize the difference between measured and desired parameters. This needed power is output to the actuator. Motor controllers are real-time systems, because performance depends greatly on when and how fast the controller software runs. Accuracy, stability, and time are important.

Example 1.4: The goal of a **traffic controller** is to minimize waiting time and to save energy. The engineer must understand the civil engineering of how city streets are laid out and the behavior of human drivers as they interact with traffic lights and other drivers. The controller uses sensors to know the number of cars traveling on each segment of road. Pedestrians can also push walk buttons. The controller will accept input commands from the fire or police department to handle emergencies. The outputs are the traffic lights at each intersection. The controller collects sensor inputs and calculates the traffic pattern needed to minimize waiting time, while maintaining safety. Traffic controllers are not real-time systems, because human safety is not sacrificed if a request is delayed. In contrast, an air traffic controller must run in real time, because safety is compromised if a response to a request is delayed. The system must be able to operate under extreme conditions such as rain, snow, freezing temperature, and power outages. Computational speed and sensor/light reliability are important.

Checkpoint 1.10: There is a microcontroller embedded in an alarm clock. List three operations the software must perform.

When designing embedded systems we need to know how to interface a wide range of signals that can exist in digital, analog, or time formats.

Table 1.3 lists example products and the functions performed by their embedded systems. The microcontroller accepts inputs, performs calculations, and generates outputs.

Functions performed by the microcontroller

Consumer/Home:

Washing machine	Controls the water and spin cycles, saving water and energy
Exercise equipment	Measures speed, distance, calories, heart rate
Remote controls	Accepts key touches, sends infrared pulses, learns how to interact with user
Clocks and watches	Maintains the time, alarm, and display
Games and toys	Entertains the user, joystick input, video output
Audio/video	Interacts with the operator, enhances performance with sounds and pictures
Set-back thermostats	Adjusts day/night thresholds saving energy

Communication:

Answering machines	Plays outgoing messages and saves incoming messages
Telephone system	Switches signals and retrieves information
Cellular phones	Interacts with key pad, microphone, and speaker
Satellites	Sends and receives messages

Automotive:

Automatic braking	Optimizes stopping on slippery surfaces
Noise cancellation	Improves sound quality, removing noise
Theft deterrent devices	Allows keyless entry, controls alarm
Electronic ignition	Controls sparks and fuel injectors
Windows and seats	Remembers preferred settings for each driver
Instrumentation	Collects and provides necessary information

Military:

Smart weapons	Recognizes friendly targets
Missile guidance	Directs ordnance at the desired target
Global positioning	Determines where you are on the planet, suggests