CHAPTER 1

Embedded Electronic Systems and Microcontrollers

This chapter provides a short introduction to embedded electronic systems, where they are used, and ways in which they can be implemented. Microcontrollers were originally developed from microprocessors for use in embedded electronic control systems, as their name implies. They include a processor and most or all of the memory, clock, and other systems needed to support it. Everything is inside a single package, which is why a microcontroller is often described as a "computer on a chip." I review the main features of a typical small microcontroller before setting the scene for the rest of the book with the MSP430.

1.1 What (and Where) Are Embedded Systems?

Suppose that you asked people in the developed world to show you the products in their house that contained "computer chips." (Admittedly, this term is deliberately vague.) Probably they would point to a personal computer and stop there. If you tried harder, you might be offered a game console or personal digital assistant. It is unlikely that they would mention cellular phones, which contain a startling degree of processing power just for communication, to say nothing of taking photographs and playing games. There is hardly an electrical consumer product nowadays that does not rely on digital control. This seems reasonable for washing machines and video recorders, but one might wonder why a toaster or a kettle needs any digital electronics. These products contain *embedded* electronic systems: The processor supports the operation of the product but is not the main reason for purchasing it. There are said to be about 100 embedded processors for each computer, so high-profile, leading-edge microprocessors make up a small part of the market in terms of volume. Fancy modern cars have approaching 100 processors and even a personal computer has embedded processors in its keyboard, mouse, screen, disk drives, and so on. The snag for an engineer is that *embedded* seems synonymous with *invisible* and few people appreciate the extent to which they rely on electronics.

Embedded systems encompass a broad range of computational power. A crude classification is given by the number of bits that can be manipulated at a time. Many processors perform very simple tasks, for which 8 or even 4 bits are sufficient. For example, I have an electric toothbrush that pauses briefly every 30 s to remind me to move on to the next quarter of my mouth. The electronics in a remote control, kettle, or toaster need not be very sophisticated either. A bigger device that can handle 8 bits is needed for something like a washing machine. These may be comparable in power with the processors used in the first personal computers and, in fact, the descendents of the microprocessors of those days are still widely used. Digital control has been employed in car engines for many years, since the first legislation was introduced to reduce pollution and raise efficiency. These relied on 16-bit processors for a long time, but 32 bits are now needed to provide the necessary performance. A cellular phone also has a 32-bit processor as well as specialized hardware for digital signal processing. The subject of this book is the Texas Instruments MSP430, which is a straightforward, modern 16-bit processor designed specially for low-power applications.

1.2 Approaches to Embedded Systems

Many different approaches can be taken for the design of embedded systems. The general trend is toward digital systems and increasing integration: Systems that used analog electronics or small-scale integrated circuits (ICs) in the past are now more likely to use larger digital ICs. It is easier to follow this evolution for a simple system so we look at a couple of these first.

1.2.1 Timer for an Electric Toothbrush

First consider a very simple application, the timer for the toothbrush mentioned earlier. Here are a few possible ways of building a timer to give a 30 s delay.

Small-Scale Integration: The 555

Mention "timer" to many electronic engineers and they will immediately think of the 555. Introduced by Signetics in 1972, this has proven so versatile that complete books have been devoted to it. In a straightforward circuit it provides a square wave whose period is given very roughly by RC, the product of the values of a resistor and capacitor. (Really there are two resistors, and RC should be multiplied by a constant, but that does not affect the argument.) We want $RC \approx 30\,\mathrm{s}$ and it is usually a good idea not to exceed $R = 1\,\mathrm{M}\Omega$, or leakage currents can become a problem. This means that $C \approx 33\,\mu\mathrm{F}$, which is a standard value. The snag is that capacitors of this value are usually electrolytic, which are physically large and leaky, but the circuit would probably work. This solution needs an eight-pin 555, two resistors, and one electrolytic capacitor.

Medium-Scale Integration: 4000 Series CMOS

A large capacitor is needed with the 555 because the period of oscillation is so long. A remedy is to reduce the period (increase the frequency) and feed the output into a counter, which could be selected from the 4000 series of small- and medium-scale ICs. This was the first family of CMOS logic and dates from 1968. They are slow but operate from a wide range of voltages and are particularly suitable for battery-powered circuits. Unlike the 7400 series, which provides mostly straightforward logic components, several unusual functions are available.

A suitable device for this application is the 4060. It contains a 14-bit ripple counter and an internal oscillator circuit to provide a clock, which needs either a crystal or two resistors and a capacitor. The outputs from ten stages of the counter are brought to pins so it need not just divide the clock by the maximum factor of $2^{14} = 16{,}384$. Without designing the system in detail, the oscillator can now run at a few hundred Hz. This permits a much smaller, nonelectrolytic capacitor. The snag is that the 4060 comes in a 16-pin package, so the overall circuit might take up more space than with the 555.

Large-Scale Integration: Small Microcontroller

I suspect that the real product contains a small microcontroller. This is a complete "computer on a chip" and is described in the next section. Several manufacturers produce these in eight-pin and even six-pin packages, only a few millimeters across. They have complete internal oscillators, so no external components are needed (except perhaps a decoupling capacitor). Even these tiny components can do far more than timing 30 s intervals. Perhaps they also supervise the charging of the battery, which is indicated by a light-emitting diode (LED). Maybe they could drive a small speaker to play a tune as well.

1.2.2 Electronic Dice

The project in my undergraduate course on digital electronics (a long time ago) was to build an electronic dice from gates and flip-flops. The top circuit in Figure 1.1 is more

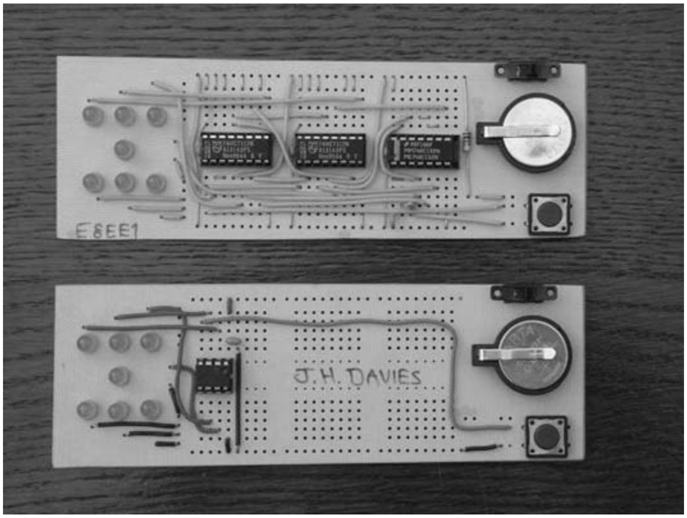

Figure 1.1: Electronic dice built using (top) *JK* flip-flops and gates and (bottom) an eight-pin microcontroller.

modern but uses 7400 series logic ICs and the same idea (it was formerly used in one of my department's courses). Two packages contain *JK* flip-flops and the third contains gates, which drive the flip-flops through the correct sequence and provide the clock. This is a simple example of a Moore state machine.

The lower photograph shows a circuit built on the same board using an eight-pin microcontroller. The economy in components and size is obvious. In fact, the on–off switch is superfluous because the microcontroller automatically switches off the LEDs after a few seconds and enters a low-power mode until the button is pressed again.

1.2.3 Larger Systems

Large embedded systems might contain fairly standard personal computers inside them. Many automatic teller machines (ATMs) are built like this. The advantage is that the hardware is standard and a huge range of software is available, including operating systems. On the other hand, the systems are large and consume a lot of power. Their reliability may also be questionable. Three general approaches can be taken between this extreme and small-scale integration.

Application-specific integrated circuits (ASICs): Specially designed for a particular application as their name implies. They provide the best performance but are extremely expensive to design and test. This restricts them to applications with very large volume or where performance must be bought at any price.

Field-programmable gate arrays (FPGAs) and programmable logic devices (PLDs): Essentially an array of gates and flip-flops, which can be connected by programming the device to produce the desired function. This is specified using a hardware description language such as VHDL or Verilog. Older programmable logic devices, such as the 22V10, contain a set of flip-flops (ten in this case) whose inputs come from an array of AND and OR gates. They are often used to provide the "glue" logic needed to support a large processor. Field-programmable gate arrays have a more versatile structure and may be enormous, with over a billion (10^9) transistors.

Microcontrollers: These have nearly fixed hardware built around a central processing unit (CPU). The CPU controls a range of peripherals, which may provide both digital and analog functions such as timers and analog-to-digital converters. Small devices usually include both volatile and nonvolatile memory on the chip but larger processors may need separate memory. Their operation is usually programmed using a language such as C or C++.

In practice the distinction between these is blurred, particularly for larger ICs. For example, part of a FPGA may be designed to act as a processor (FPGAs are often used to test the design of microcontrollers). Some devices include a fixed processor with something like a FPGA that can be configured to provide the desired digital peripherals.

Example 1.1

Estimate the number of embedded electronic systems in your living room.

1.3 Small Microcontrollers

A *microprocessor* contains a complete digital processor, which includes at least the arithmetic logic unit and associated registers. The earliest devices, such as the Intel 4004 and Texas Instruments' TMS1000, were introduced at the beginning of the 1970s. Their breathtaking evolution since then has been toward increasing computational power and complexity. They are also more powerful in an electrical sense. Large, modern microprocessors need huge heat sinks and fans and can draw over 100 A of current. The reduction of power dissipation is a major thrust of current development, now that so many microprocessors are used in portable equipment, whose battery should last for as long as possible. A microprocessor needs many other components to support it. These include a (large) external memory and the other components that can be found on the motherboard of a personal computer.

It was realized from the start that microprocessors would also be useful to control electronic equipment, such as photocopiers. Here the emphasis was less on computational power; the drive was more to reduce the complexity of the hardware and increase reliability. The trend was therefore to integrate as many functions as possible on to the same chip as the processor. This gave rise to the *microcontroller* (MCU or μC), which typically contains all of the functions needed to make a complete computer system, including memory. A microcontroller also contains a selection of peripheral modules to provide commonly needed digital functions, such as timers, and often analog functions as well. Inevitably the distinction between microprocessors and microcontrollers is blurred.

Although microprocessors have evolved almost entirely toward increasing computational power, this is not true of microcontrollers. One trend has been toward increasing integration so that everything is in one package, including the clock oscillator. Another trend has been the increasing integration of analog functions, so that it is often cheaper to buy a microcontroller with an analog-to-digital converter (ADC) than it is to buy a standalone ADC. Processing power has increased in some families. For instance the PowerPC processor, which powered Macintosh computers for many years, is now widely used in microcontrollers for engine management. However, there has also been vigorous development of smaller, cheaper devices. There is now a wide selection of microcontrollers in eight-pin and even six-pin packages, costing well under $1. These are aimed at applications that might previously have used discrete components and small-scale integration or, more likely, did not include electronics at all.

From now on, I shall consider only "small" microcontrollers (MCUs) although as usual this term cannot be defined precisely. Typically these devices can process 8 or 16 bits of data and have a 16-bit address bus, which means that they can address 64 KB of memory directly with no paging or banking. (The upper-case letter K stands for a "binary" kilo, meaning $2^{10} = 1024$. A lower-case k means the decimal value $10^3 = 1000$.) Their main function is likely to be sequential control rather than computation. They are designed for low power and low cost. There is a wide range of products despite these limitations in packages with 6 to over 100 pins. Many billions have been sold—this is a big business.

Another distinction between microprocessors and microcontrollers is the operating system. It is very unlikely that a modern microprocessor would be used without an operating system such as linux, MacOS, or Windows. In contrast, small microcontrollers are unlikely to use an operating system at all: Software is written to run directly on the hardware without any additional support. When an operating system is used, it is very different from that on a desktop computer. This is because microcontrollers are often used in *real-time* systems, where they are required to respond to external events within a prescribed time. A specialized *real-time operating system* (RTOS) is used in such applications. The relation between a user's program and an RTOS is quite different from that between a program and the operating system on a desktop computer. When you turn on a desktop computer, you have to wait for the operating system to load before you can do anything. In contrast, a microcontroller starts up with the user's program, whose first job is to start the RTOS, configure it, and launch the desired tasks. A more obvious difference is that an RTOS may occupy only a few hundred bytes of memory, not megabytes.

Given that the role of a microcontroller is to control the system in which it is embedded, its inputs and outputs are clearly important. Look briefly at a washing machine as a typical application to see what is likely to be required:

- The switches on the front panel are usually on–off buttons, which provide digital inputs to the MCU. On the other hand, there might be some knobs that offer continuous adjustment of the temperature or spin speed, in which case the MCU must also handle an analog input.
- Some sensors, such as one to check that the door is closed, also give digital inputs.
- Other sensors, such as those that measure the temperature and level of the water, require analog inputs.
- The front panel has some sort of display. This may be something very simple, such as a few light-emitting diodes (LEDs), which need only a digital output (on or off). Numerical seven-segment displays are also simple to drive, although they may be multiplexed to reduce the number of connections needed; see the section "Digital Input and Output: Parallel Ports" on page 208. On the other hand, specialized hardware is needed for a liquid crystal display (LCD), which may be either in the microcontroller or built into the display itself.
- Some components inside the washing machine, such as the valves for water, need only to be switched on or off. They can therefore be driven from a digital output, although additional components may be needed to provide the voltage and current needed by the valves.
- In other cases, it appears that the output can be varied continuously. For example, the motor rotates slowly for washing and through a range of increasing

speeds for spinning. This appears to need an analog output, but in practice, this is simulated using *pulse-width modulation* (PWM), described in the section "Output in the Up Mode: Edge-Aligned Pulse-Width Modulation" on page 330.

How much processing power is needed for this washing machine? The most rapidly changing outputs are the LCD and motor using PWM, which may need to be driven at frequencies around a kilohertz, but this will not load the processor itself, because dedicated peripherals are used for these tasks. The inputs need to be scanned but this need be done only a few times per second. Each phase in the washing takes several seconds at least, and a timer is included in the hardware for this. It seems that the processor itself has almost nothing to do. Sophisticated machines may use fuzzy logic to adjust the washing cycle, but there clearly is plenty of time for this.

Suppose that we add an "out of balance" sensor to the washing machine, which monitors the force on the drum as it rotates. This is to prevent the machine vibrating dangerously if the load is unevenly distributed. Machines typically spin the clothes at around 1200 rpm or 20 revolutions per second. The force might be measured 10 times per revolution or 200 times per second. The output of the sensor probably is a continuously variable voltage, so the MCU needs to perform an analog-to-digital conversion. This is a relatively slow process and may take 50 clock cycles. The processor therefore needs about 10,000 clock cycles per second for this task. Typical clock frequencies are several megahertz, so this is unlikely to tax the microcontroller. The key point is that the demanding tasks are handled by special hardware, so the processor itself need not be particularly powerful.

Example 1.2

A bread maker is another common domestic appliance that probably contains a microcontroller. How much processing power is needed for this?

1.4 Anatomy of a Typical Small Microcontroller

We are now in a position to set out the functions required inside a practical microcontroller. The following features, shown in Figure 1.2, are essential.

Central processing unit: I define this to include

- Arithmetic logic unit (ALU), which performs computation.
- Registers needed for the basic operation of the CPU, such as the program counter (PC), stack pointer (SP), and status register (SR).

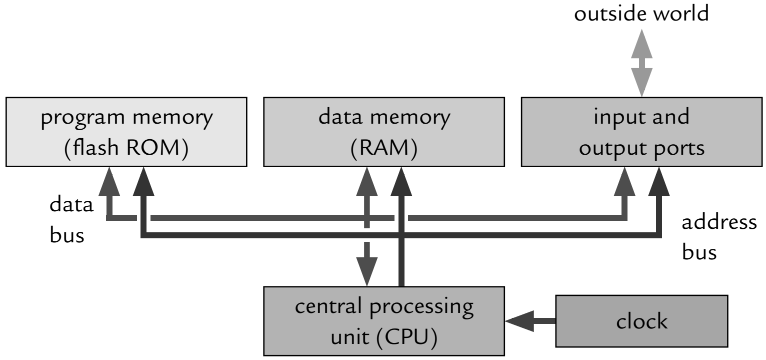

Figure 1.2: Essential components of a microcontroller.

- Further registers to hold temporary results.
- Instruction decoder and other logic to control the CPU, handle resets, and interrupts, and so on.

Memory for the program: Nonvolatile (read-only memory, ROM), meaning that it retains its contents when power is removed.

Memory for data: Known as random-access memory (RAM) and usually volatile.

Input and output ports: To provide digital communication with the outside world.

Address and data buses: To link these subsystems to transfer data and instructions.

Clock: To keep the whole system synchronized. It may be generated internally or obtained from a crystal or external source; modern MCUs offer considerable choice of clocks.

It is unlikely that any processor would lack any of these features, although their implementation may differ substantially. The big differences between devices comes from the range of *peripherals* included. These functions needed completely separate pieces of equipment long ago, but as technology improved, they could be included on the same printed circuit board as the processor. Now most peripherals are on the same integrated circuit as the processor and the nomenclature is no longer appropriate, but it has stuck. Here are some of the more common peripherals.

Timers: Most microcontrollers have at least one timer because of the wide range of functions that they provide. Here are just a few.

- The time at which transitions occur on an input can be recorded. This may be used to deduce the speed of a bicycle, for instance, if the input is driven by a sensor that gives a pulse every time the wheel completes a revolution.
- Outputs can be driven on and off automatically at a specified frequency. This is used for pulse-width modulation to control the speed of the motor in a washing machine, described previously.
- They provide a regular "tick" that can be used to schedule tasks in a program. Many programs are awakened periodically by the timer to perform some action—measure the temperature and transmit it to a base station, for example—then go to sleep (enter a low-power mode) until awakened again. This conserves power, which is vital in battery-powered applications.

Watchdog timer: This is a safety feature, which resets the processor if the program becomes stuck in an infinite loop.

Communication interfaces: A wide choice of interfaces is available to exchange information with another IC or system. They include serial peripheral interface (SPI), inter-integrated circuit (I^2C or IIC), asynchronous (such as RS-232), universal serial bus (USB), controller area network (CAN), ethernet, and many others.

Nonvolatile memory for data: This is used to store data whose value must be retained when power is removed. Serial numbers for identification and network addresses are two obvious candidates.

Analog-to-digital converter: This is very common because so many quantities in the real world vary continuously.

Digital-to-analog converter: This is much less common, because most analog outputs can be simulated using PWM. An important exception used to be sound, but even here, the use of PWM is growing in what are called *class D* amplifiers.

Real-time clock: These are needed in applications that must track the time of day. Clocks are obvious examples but data loggers are also an important case.

Monitor, background debugger, and embedded emulator: These are used to download the program into the MCU and communicate with a desktop computer during development. See the section "Access to the Microcontroller for Programming and Debugging" on page 57.

The processor communicates with these peripherals by reading from, and writing to, particular addresses in memory. These memory locations are called *special function registers* or *peripheral registers* to distinguish them from ordinary memories, which simply store data, but exactly the same commands are used—no special commands are needed. In practice, microcontrollers spend much of their time handling the peripheral registers. This shows the central role of memory in a microcontroller so I review its organization next.

1.5 Memory

Memory lies at the heart of any computer. It can be pictured as a pile (or piles) of pigeonholes. Each location can typically store 1 byte (8 bits or 1 B) of data and is often called a *register*, although this term is sometimes reserved for memories within the CPU. Memory is linked to the CPU by buses for data, address, and control as shown in Figure 1.2. Buses are shared sets of wires that join several components, rather like a multilane highway. Access to the bus must be controlled, as it is for the highway, to define when data are valid and ensure that two components do not try to write at the same time. This is the job of the control bus, which I have not drawn and shall ignore from now on. The number of wires in a bus defines its width, and processors are commonly characterized by width of the data bus, which is generally the same as the size of data that can be processed by the CPU. For example, an 8-bit processor has a data bus of this width and most operations in its CPU use 8 bits (although there may be further instructions to handle 16 bits as well). The address bus need not have the same width as the data bus and is often wider; an 8-bit address bus would only carry $2^8 = 256$ distinct addresses, which is too small to be useful. Many 8-bit processors have 16-bit address buses, for instance.

Addresses are always quoted in *hexadecimal* (hex) notation, where each digit has a range of 0–15. The values 10–15 are written as a–f or A–F. Hexadecimal values are distinguished by a prefix of “0x” in the C programming language and I follow this usage; other notations are often required for assembly language. The reason for using hexadecimal notation is that an 8-bit byte can be written as a two-digit hexadecimal number in the range 0x00–0xFF, corresponding to 0–255 in decimal. Similarly, a 16-bit address lies in the range 0x0000–0xFFFF. The four bits that correspond to each hexadecimal digit are (regrettably) known as nibbles.

The buses in a microprocessor are brought out to pins for access to external memory. Larger microcontrollers may also do this, but the buses usually remain hidden inside small microcontrollers. External memory can be added using a separate interface such as SPI.

1.5.1 Volatile and Nonvolatile Memory

Memory can be classified into two main varieties:

Volatile: Loses its contents when power is removed. It is usually called *random-access memory* or RAM, but the name is misleading because access to most other types of memory is equally random. The vital feature is that data can be read or written with equal ease. Volatile memory is used for data, and small microcontrollers often have very little RAM, sometimes only a few tens of bytes. The memory is usually *static* RAM, which means that it retains its data even if the clock is stopped (provided that power is maintained, of course). A single cell of static RAM needs six transistors. RAM therefore takes up a large area of silicon, which makes it expensive. Most memory in a desktop computer is *dynamic* RAM. This needs only one transistor per cell but must be refreshed regularly to maintain its contents, so it is not used in small microcontrollers.

Nonvolatile: Retains its contents when power is removed and is therefore used for the program and constant data. It is usually called *read-only memory* or ROM, but again this traditional name has become misleading. Most modern microcontrollers can write to their nonvolatile memory but it is much slower and more complicated than writing to RAM.

There are many types of nonvolatile memory in use:

Masked ROM: The data are encoded into one of the masks used for photolithography and written into the IC during manufacture. This memory really is read-only. It is used for the high-volume production of stable products, because any change to the data requires a new mask to be produced at great expense. Some MSP430 devices can be ordered with ROM, shown by a *C* in their part number. An example is the MSP430CG4619.

EPROM (electrically programmable ROM): As its name implies, it can be programmed electrically but not erased. Devices must be exposed to ultraviolet (UV) light for about ten minutes to erase them. The usual black epoxy encapsulation is opaque, so erasable devices need special packages with quartz windows, which are expensive. These were widely used for development before flash memory was widely available.

OTP (one-time programmable memory): This is just EPROM in a normal package without a window, which means that it cannot be erased. Devices with OTP ROM are still widely used and the first family of the MSP430 used this technology.

Flash memory: This can be both programmed and erased electrically and is now by far the most common type of memory. It has largely superseded electrically erasable, programmable ROM (EEPROM). The practical difference is that individual bytes of EEPROM can be erased but flash can be erased only in blocks. Most MSP430 devices use flash memory, shown by an *F* in the part number.

A higher voltage is needed to write to flash memory than is necessary for normal operation. This had to be supplied externally to early devices but modern components include a charge pump to generate the programming voltage internally. Flash memory is of course widely used in portable storage devices—memory cards, USB drives, and so on—but the technology is different. Microcontrollers use NOR flash, which is slower to write but permits random access. NAND flash is used in bulk storage devices and can be accessed only serially in rows.

Although flash currently dominates nonvolatile memory, new technologies may be on the way. There has been vigorous research into ferroelectric memory, silicon nanocrystals, and other approaches, which their promoters confidently expect to be adopted in the next few years.

1.5.2 Harvard and von Neumann Architectures

The two types of memory that we just reviewed, volatile and nonvolatile, can be treated in the two general ways as illustrated in Figure 1.3. General-purpose processors use almost exclusively the von Neumann architecture but both are used in microcontrollers.

Harvard Architecture

The volatile (data) and nonvolatile (program) memories are treated as separate systems, each with its own address and data bus. Many microcontrollers use this architecture, including Microchip PICs, the Intel 8051 and descendents, and the ARM9. The principal advantage is efficiency.

- It allows simultaneous access to the program and data memories. For instance, the CPU can read an operand from the data memory at the same time as it reads the next instruction from the program memory.
- The two systems can be separately optimized. For example, the PIC16 has a data memory with an 8-bit data bus and a 9-bit address bus. On the other hand, the program memory is 14 bits wide, so that each word holds a complete instruction, with a 13-bit address bus.

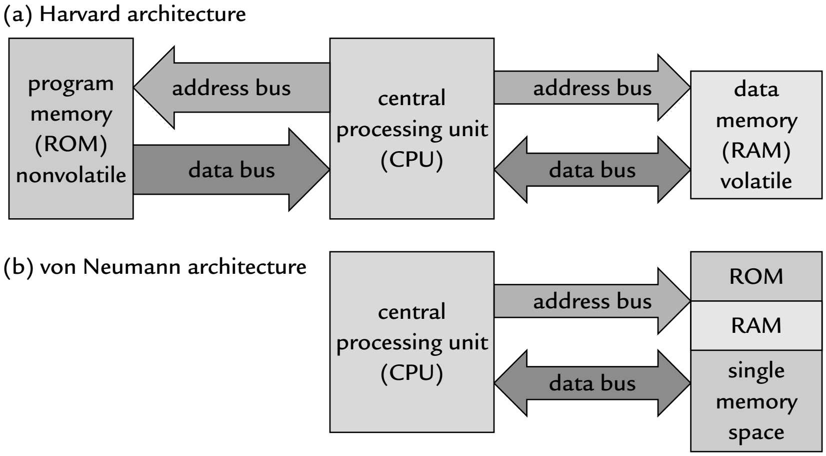

Figure 1.3: Harvard and von Neumann architectures for memory.

A problem with the Harvard architecture is that constant data (often lookup tables) must be stored in the program memory because it is nonvolatile. This means that constants cannot be read in the same way as volatile values from the data memory. Special "table read" instructions must therefore be provided or part of the program memory is mapped into data memory.

von Neumann Architecture

There is only a single memory system in the von Neumann or Princeton architecture. This means that only one set of addresses covers both the volatile and nonvolatile memories. The memory map, which shows the addresses at which each type of memory is located, becomes particularly important. The architecture is intrinsically less efficient because several memory cycles may be needed to extract a full instruction from memory. However, the system is simpler and there is no difference between access to constant and variable data. Microcontrollers with a von Neumann architecture include the MSP430, the Freescale HCS08, and the ARM7.

Example 1.3

What is the maximum number of locations that can be addressed in the data and program memories of the PIC16? Express your answer in decimal and hexadecimal notations.

Example 1.4

Suppose that the memory of a von Neumann processor is organized in bytes (as usual) and can store the same total number of *bits* as the PIC16. How many bytes are needed and what is the minimum width of the address bus?

1.6 Software

I have already mentioned the biggest difference between writing a program for a desktop computer and for a small microcontroller: the absence of an operating system (in most cases). The impact of this will become clear in Chapter 4. The tasks carried out by the program may be very different too. Desktop computers often spend considerable time on calculations, whether it is analyzing data or computing the next view in a game. The CPU in a microcontroller spends much of its time interacting with peripherals, although it may have to perform some calculations on the values received from a sensor, for instance.

Several languages may be used for programming a small microcontroller:

Machine code: The binary data that the processor itself understands. Each instruction has a binary value called an *opcode*. It is unrecognizable to humans, unless you spent a very long time on low-level debugging. Some very early computers had to be programmed in machine code, but that was long ago, thank goodness. You will see it, however, because the contents of memory are shown in the debugger and machine code is included in the "disassembly" (see later).

Assembly language: Little more than machine code translated into English. The instructions are written as words called *mnemonics* rather than binary values and a program called an *assembler* translates the mnemonics into machine code. It does a little more than direct translation, but not a lot, nothing like a compiler for a high-level language.

A major disadvantage of assembly language is that it is intimately tied to a processor and is therefore different for each architecture. Even worse, the detailed usage varies between development environments for the same processor. Most programming of small microcontrollers was done in assembly language until recently, despite these problems, mainly because compilers for C produced less-efficient code. Now the compilers are better and modern processors are designed with compilers in mind, so assembly language has been pushed to the fringes. A few operations, notably bitwise rotations, cannot be written directly in C, and for these assembly language may be much more efficient. However, the main argument for learning assembly language is for debugging. There is no escape if you need to check the operation of the processor, one instruction at a time. *Disassembly* is the opposite process to assembly, the translation of machine code to assembly language.

C: The most common choice for small microcontrollers nowadays. A compiler translates C into machine code that the CPU can process. This brings all the power of a high-level language—data structures, functions, type checking and so on—but C can usually be compiled into efficient code. Compilation used to go through assembly language but this is now less common and the compiler produces machine code directly. A disassembler must then be used if you wish to review the assembly language.

C++: An object-oriented language that is widely used for larger devices. A restricted set can be used for small microcontrollers but some features of C++ are notorious for producing highly inefficient code. Embedded C++ is a subset of the language intended for embedded systems. Java is another object-oriented language, but it is interpreted rather than compiled and needs a much more powerful processor.

BASIC: Available for a few processors, of which the Parallax Stamp is a well-known example. The usual BASIC language is extended with special instructions to drive the peripherals. This enables programs to be developed very rapidly, without detailed understanding of the peripherals. Disadvantages are that the code often runs very slowly and the hardware is expensive if it includes an interpreter.

1.7 Where Does the MSP430 Fit?

An enormous number of microcontrollers is available. They fill many pages of the distributors' catalog with a tiny typeface. Where does the subject of this book, the MSP430, fit into this spectrum?

The MSP430 was introduced in the late 1990s, although its ancestry goes back to the 4-bit TSS400. In summary, it is a particularly straightforward 16-bit processor with a von Neumann architecture, designed for low-power applications. The CPU is often described as a reduced instruction set computer (RISC) but this is debatable (if unimportant) and is considered in the section "Reflections on the CPU and Instruction Set" on page 153. Both the address and data buses are 16 bits wide. The registers in the CPU are also all 16 bits wide and can be used interchangeably for either data or addresses. This makes the MSP430 simpler than an 8-bit processor with 16-bit addresses. Such a processor must use its general-purpose registers in pairs for addresses or provide separate, wider registers.

In many ways, the MSP430 fits between traditional 8- and 16-bit processors. The 16-bit data bus and registers clearly define it as a 16-bit processor. On the other hand, it can address only $2^{16} = 64$ KB of memory. This is the same as the Freescale HCS08, an 8-bit family that also uses von Neumann memory and whose architecture goes back to the 6800 in the very early days of microprocessors. The corresponding 16-bit family is the HCS12, which uses paging to address up to 8 MB of memory. The absence of pages or banks in the memory makes the MSP430 very simple to use. (The picture is a little different with the MSP430X, which has extended registers and a wider address bus that can handle up to 1 MB of memory. I leave this until Chapter 11.)

Another feature of the MSP430 that stems from its recent introduction is that it is designed with compilers in mind. Most small microcontrollers are now programmed in C, and it is important that a compiler can produce compact, efficient code. The MSP430 has 16 registers in its CPU, which enhances efficiency because they can be used for local variables, parameters passed to subroutines, and either addresses or data. This is a typical feature of a RISC, but unlike a "pure" RISC, it can perform arithmetic directly on values in main memory. Microcontrollers typically spend much of their time on such operations.

The MSP430 is the simplest microcontroller in TI's current portfolio. Its more powerful siblings include the TMS470, which is based on the 32/16-bit ARM7, and the C2000, which incorporates a digital signal processor. Several features make the MSP430 suitable for low-power and portable applications:

- The CPU is small and efficient, with a large number of registers.
- It is extremely easy to put the device into a low-power mode. No special instruction is needed: The mode is controlled by bits in the status register. The MSP430 is awakened by an interrupt and returns automatically to its low-power mode after handling the interrupt.
- There are several low-power modes, depending on how much of the device should remain active and how quickly it should return to full-speed operation.
- There is a wide choice of clocks. Typically, a low-frequency watch crystal runs continuously at 32 KHz and is used to wake the device periodically. The CPU is

clocked by an internal, digitally controlled oscillator (DCO), which restarts in less than 1 μs in the latest devices. Therefore the MSP430 can wake from a standby mode rapidly, perform its tasks, and return to a low-power mode.

- A wide range of peripherals is available, many of which can run autonomously without the CPU for most of the time.
- Many portable devices include liquid crystal displays, which the MSP430 can drive directly.
- Some MSP430 devices are classed as application-specific standard products (ASSPs) and contain specialized analog hardware for various types of measurement.

It is impossible to pick a single number to demonstrate the low-power consumption and great caution is needed when comparing different manufacturers' claims. For example, the F2013 draws around 4.5 mA when operating at its top speed of 16 MHz. This also needs its maximum supply voltage of 3.5 V. However, the supply can be reduced to 1.8 V and the current falls to 0.2 mA if a speed of 1 MHz is acceptable. In many applications, the microcontroller spends most of its time in standby mode, when a typical current is below 1 μA. Many batteries have a larger self-discharge current than this. The MSP430 can restart quickly because of its DCO, which may be an important factor in the overall power budget. The application note *MSP430 Competitive Benchmarking* (slaa205b) contains a comparison of the MSP430 with a range of other microcontrollers.

Currently four families of MSP430 are available. The letter after MSP430 shows the type of memory. Most part numbers include F for flash memory but some have C for ROM. There is a second letter for ASSPs to show the type of measurement for which they are intended: E for electricity, W for water, and G for signals that require a gain stage, provided by operational amplifiers. The next digit shows the family and the final two or three digits identify the specific device.

MSP430x1xx: Provides a wide range of general-purpose devices from simple versions to complete systems for processing signals. There is a broad selection of peripherals and some include a hardware multiplier, which can be used as a rudimentary digital signal processor. Packages have 20–64 pins.

MSP430F2xx: A newer, general-purpose family introduced in 2005. Its CPU can run at 16 MHz, double the speed of earlier devices, while consuming only half the current at the same speed. Some come in 14-pin packages, including a traditional plastic dual-in-line (PDIP) option, which is attractive for anybody who has to build circuits by hand. They do not require a crystal for their low-frequency clock. Pull-up or pull-down resistors are provided on the inputs to reduce the number of external components needed. There are many options for analog inputs. Even the smallest, 14-pin devices offer a 16-bit sigma–delta ADC.

This family is intended to supersede the MSP430x1xx over the next few years. Devices with the same pin-out and related model numbers are intended as drop-in replacements. For example, the F24x can replace the F14x.

MSP430x3xx: The original family, which includes drivers for LCDs. It is now obsolescent.

MSP430x4xx: Can drive LCDs with up to 160 segments. Many of them are ASSPs, but there are general-purpose devices as well. Their packages have 48–113 pins, many of which are needed for the LCD.

MSP430X: The original MSP430 architecture, extended to give the MSP430X in 2006, mainly so that it can address extra memory but with other improvements as well. Curiously, this is not marketed as a separate family: The devices are included in the MSP430F2xx and MSP430F4xx families with nothing in their part number to distinguish them. The CPU is a MSP430x if there is more than 64 KB of memory.

The letters MSP stand for *mixed signal processor*, which is a reminder that many practical applications require analog inputs. There is a selection of analog-to-digital converters with a resolution of up to 16 bits. An example of a system where this choice is important is the weighing machine shown in Figure 1.4, which is another project formerly used in my department. It includes the following functional blocks:

- The sensor has four resistive elements arranged as a Wheatstone bridge. Ideally, this is balanced when there is no load, giving $V_+ = V_-$. Two of the resistances increase and two decrease when a weight is placed on the scale pan, driving the bridge out of balance.
- A differential amplifier magnifies the difference in voltage between its input terminals, giving $V_{\text{out}} = A(V_+ - V_-)$, where A is the gain.
- The analog output of the amplifier is converted to a binary value in an analog-to-digital converter.
- The microcontroller multiplies the input by an appropriate factor so that the display gives the weight in grams or ounces and subtracts an offset so that the

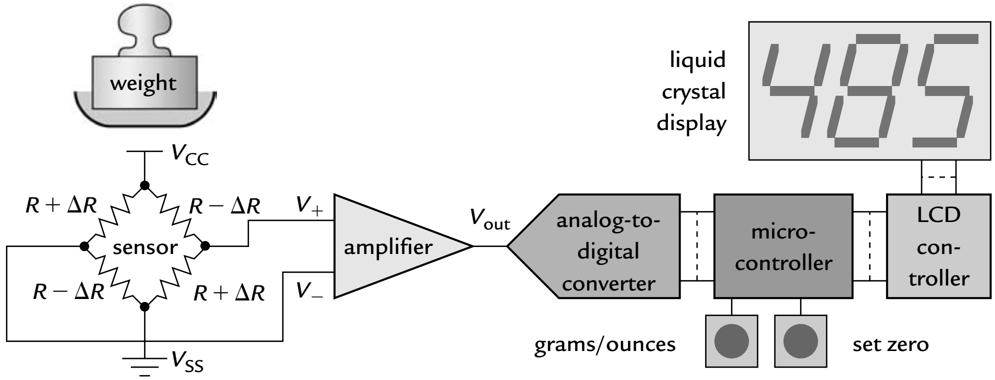

Figure 1.4: Weighing machine with a liquid crystal display, broken down into individual functions.

display reads zero when no weight is present. It also reads the buttons and supervises the complete system.

- There is a serial interface between the microcontroller and the liquid crystal display, which has a built-in controller.

This system clearly needs a lot of components, including several integrated circuits. The project was designed this way for pedagogical reasons, to pull together functions that had been covered in other courses that students had taken on analog electronics, embedded systems, digital electronics, circuits, and so on. It was not intended as a practical product, although weighing machines would have been designed this way in the past.

In contrast, the whole system can be constructed from a sensor, an MSP430, a simple LCD without a controller, and a couple of decoupling capacitors. The MSP430x4xx family drives segmented LCDs directly, which eliminates the need for a controller. Several devices contain ADCs with high-resolution, differential inputs, which would work directly from the sensor without the need for an amplifer. The microcontroller can also manage the power drawn by the circuit so that the processor would be switched off when it was not needed and the whole system shut down after a period of inactivity. This is an ideal application for the MSP430—so well suited, in fact, that it is described in the application note *MSP430F42x Single-Chip Weight Scale* (slaa220). I focus on the MSP430 in the next chapter.