Review of Chapter 15

The briefest of summaries, of Chapter 15's already brief treatment of microcontrollers.

¶A. Microcontrollers – the 10,000-foot View.

In 1960 the predictions of the future had folks flying around in jetpacks, traveling on supersonic passenger jets, and, most daring of all …*a push-button phone in every home!* Put another way, they extrapolated the technologies they knew (transportation, wired telephones), but they missed the Big One (microelectronics, and particularly embedded microcontrollers). We don't have the personal jetpacks, but we've got portable and instant interactive access to news, information, and people. Hey, we don't *need* no stinkin' jetpack!

Microcontrollers (μCs) are standalone processors with a full suite of peripherals integrated on a single inexpensive chip (Figure 15.1). You get substantial computational performance (32-bit processor, 100 MIPs, no problem), plus an assortment of ADCs, DACs, Ethernet, USB, PWM, LCD controller, SPI, multiple UARTs and timers, and of course on-chip program memory and SRAM, all in one chip for a price well below \$10; see, for example, the now-vintage example in Figure 10.86, or take a look at NXP's implementations of the ARM Cortex-M4 MCU (microcontroller unit), for example the LPC4088 series. Microcontrollers are useless without their stored *firmware* programs, whose creation can be the major stumbling block to a successful design. So, along with the creation of silicon with stunning integration and performance, the industry has been developing and streamlining the integrated development environment (IDE) – the process of initial code development, simulation, loading onto the target device, and in-circuit debugging.

To the circuit designer, the microcontroller should be thought of as a circuit *component*, like an op-amp and sometimes even less expensive. Even in their simplest varieties they are particularly useful as interfaces between the user and the other circuitry (see for example Figure 15.5); and in their complex varieties they can carry out most of the instrument's functions (as in Figure 15.18). In this chapter we introduced the vast subject of microcontrollers with some illustrative examples, to give a sense of what's possible; we included with each its corresponding *pseudocode*, and for the first example (suntan monitor) we listed the detailed C-language code.

¶B. Popular Microcontroller Families.

In §15.3 we provided an annotated listing of contemporary favorites. The dominant species are the simpler AVR (Atmel) and PIC (Microchip) devices, and the go-to choice for higher performance is the hugely popular ARM-derived processors (licensed by ARM Holdings to more than a dozen semiconductor manufacturers). The latter are used in most of the world's smartphones. The attractive Arduino platform (§15.9.4) includes both AVR- and ARM-based single-board computers (SBCs).

¶C. External Peripherals.

Microcontrollers love to pull the strings of other chips, easily done with a few direct connections (Figure 15.20, §15.8.1), or with simple inter-chip serial buses like SPI (Figure 15.21, §15.8.2) and I^2C (Figure 15.22, §15.8.3). The figures and notes list and describe more than 50 useful peripheral devices.

¶D. Design Hints (Hardware).

The five design examples described in this chapter (suntan monitor, ac power control, frequency synthesizer, thermal controller, and stabilized mechanical platform) included plenty of circuit designs, with corresponding lessons. Here are some of them:

(a) Most μCs include on-chip ADCs, which are attractive when dealing with analog inputs; but don't ignore the simple on-chip comparator, which can sometimes promote both simplicity and better performance (Figure 15.3).

(b) A digital output port bit can happily drive external MOSFETs or BJTs directly (Figure 15.3) or with the assistance of an external gate-driver IC (Figure 15.10); it can also drive a solid-state relay (Figure 15.4).

(c) Don't overlook the simplicity of simple RS-232 serial communication; it's supported in all μCs, and it's alive and well in laptops and PCs running a terminal emulator and connected via a USB or Ethernet adapter (Figure 15.5).

(d) Pushbutton switches can be connected in a matrix arrangement, to minimize wiring (Figure 15.8); they are read with digital port bit polling.

(e) Microcontrollers intended for sensing and control let you connect directly to low-level sensors (Figure 15.11); their PWM output provides an easy way to implement proportional control.

(f) Most μCs include ADCs, making it easy to attach analog-output sensors (e.g., a gyro or accelerometer, Figure 15.18). You can find a zillion little gadgets of this sort at hobbyist sites like sparkfun.com or adafruit.com.

(g) And an easy way to get started is with the Arduino boards and software (§15.9.4); open the box, and you're up and running in 20 minutes.

¶E. Programming Hints (Firmware).

You need microcontroller-specific software tools (compiler, assembler, simulator, debugger) that run on a host

PC; and you need a hardware pod (§15.9.3) to load the object code onto the target μC and to run debugging tools. You also need to be aware of ways in which microcontroller programming differs from ordinary computer programming. Here are some hints:
(a) There are vendor-specific variations in C/C++ having to do with idiosyncrasies of on-chip peripherals (ports, timers, converters, etc.); you're not writing in vanilla C.
(b) The code for a μC must do a significant amount of initialization of modes and internal peripherals; this is fussy stuff, requiring dozens of perfectly configured bytes.
(c) You many need to program in assembly language for time-critical tasks; if so, watch out for too-smart compilers that try to optimize away your code.
(d) Products intended for manufacture should be reviewed and blessed by someone skilled in human interfaces (see §15.2.2E).
(e) Enable the watchdog! Well coded microcontrollers should not crash, but they do.
(f) Keep interrupt routines short; in a simple system you may not need interrupts at all.
(g) Internal timers can generate signals at output pins, very useful when you want to trigger external devices (e.g., an ADC) with constant time intervals (and similarly for on-chip peripherals).

¶F. When to use Microcontrollers.

Almost always! Certainly for electronic systems that (a) have character or graphic displays as part of their user interface; (b) include chips requiring configuration of internal registers or operating modes; (c) communicate with a host computer, standalone peripherals, network, or wireless devices; (d) require some computation, storage, format conversion, signal processing, etc.; (e) require calibration or linearization; (f) involve sequencing events over time; or (g) are subject to upgrades or feature revisions. Microcontrollers should be considered even for traditional "analog" functions such as measurement and control, especially with the growing emphasis on analog-oriented internals in processors from companies such as Analog Devices and Cypress.

Programmable Logic Devices (including FPGAs, Chapter 11), by contrast, are generally preferred for tasks requiring critical timing, or a high degree of parallelism. They are, however, considerably more difficult than microcontrollers to program and debug.

¶G. How to select a Microcontroller.

Look first at a processor family that the people around you are using, and therefore have available the necessary software and hardware tools, and the experience. Then look at factors like these (depending on your application): (a) ports – analog, digital, and communication; (b) internal functions (e.g., converters, PWM, bare LCD drivers, etc.); (c) compute speed; (d) flash, EEPROM, and SRAM memory size; (e) package configurations; (f) power dissipation, low-power clock modes, and suspend modes; (g) software programming, simulation, and in-circuit debugging tools. The compendium in §15.3 may provide a good starting point.

The quality and quantity of included libraries can be quite important (e.g., the Arduino Project, with the large community providing code that has grown around it). Likewise, it's desirable to choose a microcontroller that has a stable compiler–debugger.

When choosing a particular part within a microcontroller family, the choices can be overwhelming. It's usually easiest to start with the "premium" part in the series, that is, the one with the fastest clock and most data (RAM) and code memory (flash ROM). It is often the case that most of the family consists of reduced versions of a few premium parts, perhaps with specialized peripherals.