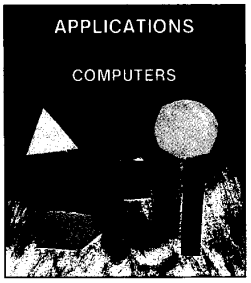
How to select a microcontroller

Word length, input/output requirement, and the economics of on- and off-chip memory are among the factors to be weighed



To most of the world, the microcontroller is an unseen workhorse embedded in automobiles, microwave ovens, guided missiles, talking dolls, and a myriad of other applications. Unlike microprocessors, which are designed for a broad range of applications, microcontrollers are generally designed with a specific application in mind.

For the designer of an embedded system, however, the choice of a microcontroller is the earliest and probably the most important to be made. Delivering the final product on schedule, within budget, and to specification will depend heavily on making the right decision.

But how best to decide? This set of guidelines is for the neophyte designer to use as a checklist for evaluating candidate devices. And even experienced designers may discover in it selection criteria they have overlooked.

A microcontroller is generally dedicated to a single application and embedded in it. Unlike a microprocessor, it tends to include all of the peripheral features needed to implement the computer portion of an embedded application. Because their application constraints are known at the time of design, the designer is able to select the lowest-cost controller that can do the job. (By contrast, microprocessors usually wind up in general-purpose computers whose final use is not known in advance to their designers.)

Bounding the problem

The first decision to make is the level of performance required, that, on the word size of the controller. Those available can be grouped broadly as 4-, 8-, 16- and 32-bit products.

Standard benchmarks of the type used to describe microprocessors are inappropriate for this evaluation. They must be replaced by critical real-time calculations and physical constraints peculiar to the system being designed. Critical calculations are those that, if not completed within a specified period of time, cause the system to lose synchronization, create a dangerous situation, or generate wrong results. For instance, if a simple timer is to generate a square wave of a given frequency, the central processing unit (CPU) must be fast enough to calculate the time at which each edge should occur and to program the timer appropriately.

Physical constraints imposed by the system, such as precision and dynamic range, determine CPU class. For instance, a numerically controlled machine might require a relative accuracy of 1 degree within a single revolution of an axis. For this accuracy, an 8-bit data value, which can resolve no more than 1 part in 256, is inadequate, whereas a 16-bit data value will represent 1 part in 65 536, or about 20 seconds of arc.

Dynamic range further constrains the system. If the same machine must also be able to provide an absolute accuracy of 1 degree over 1000 revolutions of a lead screw, even 16 bits cannot represent the full range. In such a case, the designer could choose a 32-bit solution or, if extremely wide dynamic range is necessary, resort to a floating-point number system.

The designer is cautioned, however, that choosing a CPU on the basis of a single calculation could be a mistake. If the application generally requires lower-resolution calculations, it may be better to select a processor that matches just the predominant operations, while performing the higher-resolution calculations in subroutines. In most applications, for example, a good floating-point math package is superior to floating-point hardware.

The second step—the most often overlooked—is to identify the quantity, frequency, and type (analog, digital, serial, and so on) of all input/output (I/O) signals, as well as any other special requirements, including those imposed by mechanical aspects of the system. Key to this part of the selection process is a complete top-level block diagram of the overall system. Although it is not important to assign microcontroller peripheral functions at this stage, a preliminary partitioning might be helpful in identifying system requirements.

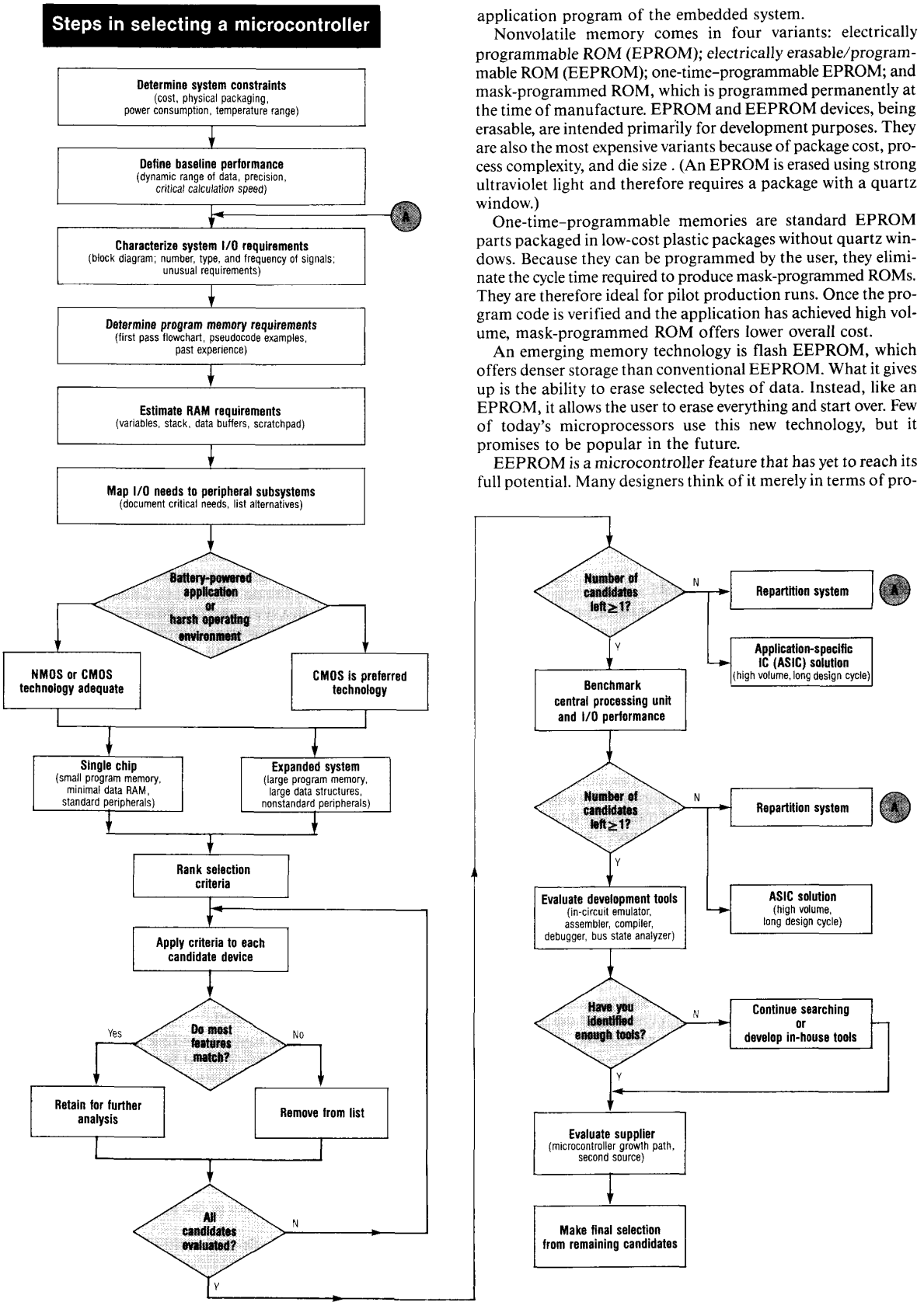
The third consideration is the application's memory requirements, which should be further broken down into program memory and data memory. Microcontroller memory may be categorized broadly as volatile and nonvolatile. Volatile random-access memory (RAM) retains data only while power is applied; it is generally used for data memory. Nonvolatile read-only memory (ROM) retains its contents permanently; it typically contains the

Defining terms

Benchmark: a standardized test or suite of tests for comparing computer performance; also, the act of determining a benchmark.
Dynamic range: the ratio of the largest to the smallest values of a range, often expressed in decibels.
Embedded system: a system into which one or more computing devices (which may be microprocessors or microcontrollers) are incorporated in such a way that the embedded device or devices are not directly accessible to the user of the system.
Floating point: the representation of numbers in scientific notation, with the exponent and mantissa given separately, so as to be able to accommodate a very wide dynamic range.
Mask-programmed: said of a semiconductor device, most often a read-only memory, that is permanently programmed as a step in its manufacture. In contrast, "field-programmed" applies to memory devices that are programmed after manufacture.
NMOS: n-channel metal-oxide-semiconductor, a type of MOS field-effect transistor.
RFI: Radio-frequency interference. (Usually) unintentionally radiated electromagnetic energy that may interfere with the operation of, or even damage, electronic equipment.

John J. Vaglica and Peter S. Gilmour Motorola Inc.

0018-9235/90/1100-00106$1.00©1990 IEEE



application program of the embedded system.

Nonvolatile memory comes in four variants: electrically programmable ROM (EPROM); electrically erasable/programmable ROM (EEPROM); one-time-programmable EPROM; and mask-programmed ROM, which is programmed permanently at the time of manufacture. EPROM and EEPROM devices, being erasable, are intended primarily for development purposes. They are also the most expensive variants because of package cost, process complexity, and die size . (An EPROM is erased using strong ultraviolet light and therefore requires a package with a quartz window.)

One-time-programmable memories are standard EPROM parts packaged in low-cost plastic packages without quartz windows. Because they can be programmed by the user, they eliminate the cycle time required to produce mask-programmed ROMs. They are therefore ideal for pilot production runs. Once the program code is verified and the application has achieved high volume, mask-programmed ROM offers lower overall cost.

An emerging memory technology is flash EEPROM, which offers denser storage than conventional EEPROM. What it gives up is the ability to erase selected bytes of data. Instead, like an EPROM, it allows the user to erase everything and start over. Few of today's microprocessors use this new technology, but it promises to be popular in the future.

EEPROM is a microcontroller feature that has yet to reach its full potential. Many designers think of it merely in terms of pro-

gram storage, not as a strategic peripheral. However, it can do much more. For example, in an application developed in the computer peripheral industry, mechanical and electrical adjustments traditionally implemented with set screws and potentiometers were replaced with parameters stored in EEPROM.

Additional locations in the EEPROM store the date of manufacture, model and serial numbers, and other data describing the product. Warranty service can make use of the information stored in the product during manufacture, as well as additional data gathered during field operation.

Estimating the amount of memory to allocate for program storage will be one of the tougher parts of the selection process. There is no magical rule of thumb. Past experience coupled with a first-pass flowchart and pseudocode examples are likely to be the most appropriate available tools. Simple applications, such as electronic thermostats, may require as few as 2K bytes, while complex applications, such as engine controls, may require 64K or more.

Sufficient RAM must be allocated in the system to store variables, the system stack frame, a scratchpad for intermediate calculations, and any data arrays or buffers. These values will be affected by the programming language used to develop the application (system stack), the selected partitioning of the problem (variables), and even the hardware configuration (data buffers). Preliminary estimates for variable stack and scratchpad storage are often done by taking a percentage of the ROM estimate. The rule of thumb used by the authors is a ROM to RAM ratio in the range of 12–20:1. Applications written in assembly language will tend toward the lower number, while compiled code will require the higher amounts. Large data buffers should be added to this estimate.

Mapping I/O to peripherals

Peripheral functions commonly integrated on microcontrollers include timers, serial and parallel communications ports, and analog-to-digital (A/D) converters. Timers range from simple counters to complex subsystems with dedicated microengines incorporating reduced-instruction-set computer (RISC) architecture. Timer systems are commonly called upon to generate periodic interrupts, capture the time an input event occurs, or generate output events at specified times. The more complex timers can produce the pulse trains required of multiphase stepper motors or even of sequencing the fuel injectors in an automobile engine without CPU intervention. The frequencies of the input and output signals generally dictate how timer complexity should be traded off against the CPU overhead required for servicing the peripheral.

External peripherals added to provide functions not usually found on microcontrollers may be interfaced through either a parallel (address, data and control) bus or a serial port. The parallel bus is conceptually simple, but fraught with many practical problems because of its rapid switching of many data lines. It generates a lot of radio frequency interference (RFI), and it consumes a lot of power.

For applications that would be handled by a single chip except for a single special function, it is best to add the peripheral via a serial bus. Only two or three pins are required, RFI is generated only during serial transmissions, and power consumption rises only slightly. A variety of chips that perform peripheral functions are marketed with serial interfaces. Among them are A/D converters, phase-locked loop (PLL) building blocks, real-time clocks, display drivers, and EEPROMs.

Parallel I/O can be found on virtually all microcontrollers, but the ports are not all equivalent. The more versatile permit pins to be defined as input or output on a per-bit basis. That can be important if system parameters are subject to change before the design is complete (as is usually the case). Other, less flexible designs offer fixed direction, input-only, or output-only pins. Beware also of manufacturers' claims for large numbers of I/O pins, which in reality are available only if all the other on-chip peripherals are disabled.

A/D converters are found on many microcontrollers. Converters with 8- or 10-bit ranges are most common, but not all manufacturers offer the same resolution for a given range. The specsmanship issues that apply to stand-alone A/D converters are an even bigger problem with converter ICs. If the intended purpose is much more demanding than checking battery voltage, care must be taken to examine all the specifications of this subsystem.

An application that seems to require a unique peripheral may not after all, for it is often possible to minimize cost by using available peripherals in innovative ways. Digital-to-analog (D/A) converters, for example, are seldom integrated into microcontrollers. When one is needed, however, the resourceful engineer will provide one economically by integrating a pulse-width-modulated waveform generated by an on-chip timer. The integrator can be as simple as a passive RC low-pass filter.

Peripherals requiring frequent interrupt service consume valuable CPU bandwidth. Several methods employed by microcontroller designers reduce interrupt overhead. Unique vectors for each interrupt source, multiple priority levels, and hardware priority resolution circuits eliminate software polling of interrupt sources. Interrupt-driven direct-memory-access (DMA) peripherals transfer data with minimal CPU service overhead, although they still require considerable bus bandwidth. Other designs get rid of service requirements altogether by distributing enough intelligence to the peripheral to eliminate CPU servicing completely during normal operation.

Bus bandwidth is affected by two components: bus width and transfer rate. Boosting either causes a corresponding increase in bandwidth. Unfortunately, this bandwidth increase normally implies a cost increase as well. Microcontroller architects have designed features into high-end products that minimize the effect of greater bandwidth on system cost. Seldom will all memory and peripherals in a system need to be accessed at the highest possible rate. Infrequently accessed devices can be replaced with lower-cost, slower devices if wait state capabilities have been designed into the bus structure.

Microcontrollers with dynamically sized buses permit memories of different widths to coexist on the same bus. Resources requiring high bandwidth, such as the stack RAM, can occupy the full bus width. Locations accessed less often, such as the boot ROM, can be configured for the width of a single memory part. Reducing the number of devices by means of dynamic sizing minimizes use of printed-circuit board area, increases reliability, and decreases RFI.

Cost, packaging, operating environment, and other physical conditions further constrain the designer. These constraints must be spelled out clearly early on, for they bear directly on the microcontroller selected. Power consumption and temperature range dictate the processing technology(ies) suitable for the application. If the product is battery powered or will operate over an extended temperature range, a CMOS version would be the better choice. For an application in which these criteria are less important, an NMOS processor could offer a cost advantage.

Single-chip or expanded?

Microcontrollers often support multiple modes of operation for a better match to the application at hand. The two most common modes are single-chip and expanded. In the first of these, all aspects of the computer are contained on the microcontroller chip. The address, data, and control buses required for memory or peripheral expansion are not brought out to pins. In the expanded mode, these buses are made available. Microcontrollers capable of both single-chip and expanded modes fit well in applications where future upgrades or cost reductions are likely and a consequent complete rewrite of the application software undesirable.

Choosing between a single- and multi-chip solution depends heavily on memory size. Technology currently limits on-chip memories to 32K bytes for program (nonvolatile) storage and 1–2K bytes for data RAM. Otherwise, an expanded system using

Common microcontroller peripheral functions and their applications

Function	Automotive	Consumer	Computer peripheral
4-, 8-bit central processing unit (CPU)	Radio	Microwave oven	Keyboard
16-bit CPU	Antilock brake system	Audio	Disk drive
32-bit CPU	Engine control	—	Intelligent disk controller
Electrically erasable/programmable ROM	Odometer	Television channel programming	Modem configuration parameters
Timer	Fuel injection timing	Camera shutter speed	Mouse optical shaft encoder
Asynchronous serial	Communication throughout car	Alarm system keypad communication	RS232 link for modem
Synchronous serial	Vacuum fluorescent driver interface	Audio interchassis communication	Real-time clock interface on PC
Analog/digital converter	Manifold air pressure (engine control)	Temperature sensor for air-conditioning thermostat	Battery voltage for portable PC
Parallel I/O	Dashboard lamp driver	Videocassette recorder front panel switches	Status lights on keyboard

external memory ICs will be needed.

The distribution of on-chip versus external memory can affect total IC cost significantly. Since memories are only available in a limited range of sizes and on-chip memories are more expensive (on a per-bit basis) than external memory, it may be advantageous to move all program and/or data memory off chip in exchange for a less expensive microcontroller chip. Several manufacturers offer microcontrollers with little or no on-chip memory for this very reason.

In addition to memory size, other obvious factors that affect the single-chip vs. multi-chip decision are power consumption, peripheral mix, and cost. But there are also less obvious factors. For instance, microcontroller-generated RFI is a major concern in RF communication applications. High-speed digital outputs contribute significant energy to the radio spectrum. Confining these signals on chip, where capacitive loads and signal line lengths are reduced up to 100 times, significantly reduces emission levels. Realizing the problems that high-frequency signals cause, some microcontroller designers have gone so far as to provide software-programmable disables on potentially unused high-speed outputs.

The selection process

During the initial search, an absolute match between requirements and features is not necessary. Minor alterations to the requirements or the addition of a peripheral chip could create the most cost-effective solution from a less-than-perfect pairing.

The feature evaluation process should begin with a ranking of the requirements in order of descending priority, as determined by the application. If any requirements were not mapped to peripheral functions during the partitioning for the block diagram, they should be mapped at this point. This mapping will not restrict the selection to a particular device or manufacturer, but will identify the microcontroller features sufficiently for the selection process to progress. The designer is cautioned that devices may surface during the selection process that would be a perfect fit if only the system had been partitioned differently. An open mind will permit these possibilities to emerge.

The procedure at this stage is to compare each candidate microcontroller with the peripheral mapping and other requirements, retaining for further study any devices that match most of the requirements. If none is appropriate, the questions to ask are: Which criteria caused the most problems in finding a match? Could the offending requirements be altered? If not, would a repartitioning of the problem alleviate them? Several iterations may be necessary at this stage before a suitable chip or chips are found.

The first lists are sifted by applying more stringent criteria at each stage. For instance, after the field has been narrowed to a handful of candidates, actual application code sequences will be written and used to benchmark CPU performance. If undertaken before the list was narrowed, this step would drag out the evaluation inordinately.

An established growth path could be an important evaluation point if a family of products is planned. Many vendors offer microcontroller families built around a common CPU core where different family members feature different I/O and memory mixes. CPU performance improvements are made through clock frequency increases and upgrades to the core.

The final selection criteria are nontechnical. Potential vendors should be qualified after considering the following criteria: product line breadth, manufacturing excellence, financial status, second-sourcing strategy, and delivery performance. Development tools must also be taken into account. If inadequate, they could place the entire project in jeopardy; therefore, it is advisable to spend some time understanding what is available for the microcontroller chosen. The entry cost of development tools can be high.

Microcontroller selection is an arduous task entailing an inordinate number of decisions. There is no universal checklist that the designer can follow, only suggestions and guidelines. Experience, in fact, is the second most useful tool the designer can possess; the most useful is a well-defined set of system requirements. Without a list, an already difficult task becomes almost impossible.

To probe further

A good basic book is *Design with Microcontrollers*, by John B. Peatman (McGraw-Hill, 1988). It begins with an overview of microcontroller applications and goes on to explore the variety of available on-chip resources. Real-time control, the main use, is developed in depth.

Real-Time Microcomputer System Design: An Introduction, by Peter D. Lawrence and Konrad Mauch (McGraw-Hill, 1987), should be useful to practicing engineers and scientists working in the physical, biological, and applied sciences. It presents a methodology for the design of real-time microcomputer systems.

IEEE Micro magazine regularly publishes articles on new microcontroller and microprocessor products. The articles are generally written by the product designers and can provide information not generally available.

Embedded Systems Programming magazine is dedicated primarily to the software-related aspects of embedded-system design.

About the authors

John J. Vaglica (M) is a microcontroller designer with 10 years of experience. He is currently employed by the Motorola Microprocessor Group, Austin, Texas, where he works in the Advanced Microcontroller Systems Design Group. He was the systems project leader for the MC68332 CPU and has worked on M68HC11 microcontrollers and their accompanying port replacement units. He has a BSEE and an MSEE, both from Texas A&M University in College Station.

Peter S. Gilmour is a senior systems analyst with over 16 years experience. He is currently employed by the Motorola Microprocessor Group on the Motorola Development Systems Group Jewelbox line of real-time emulators. He has a B.S. from Case Institute of Technology (now Case Western Reserve University, Cleveland, Ohio) and an M.S. from Arizona State University in Tempe.

Both authors can be reached at 6501 William Cannon Dr. W., Austin, Texas 78735-8598. ◆