Chapter

1

Introduction to Embedded Systems

What We Will Learn

In this chapter, we will learn the following:

Section 1.1

1. Definitions of ***system*** and ***embedded system***.
2. Classification of embedded systems into three types.
3. Skills needed in designing an embedded system.

Section 1.2 The processing unit (s) of the embedded system

1. *Processor in an Embedded System* A processor is an important unit in the embedded system hardware. A ***microcontroller*** is an integrated chip that has the processor, memory and several other hardware units in it; these form the microcomputer part of the embedded system. An ***embedded processor*** is a processor with special features that allow it to be embedded into a system. A ***digital signal processor (DSP)*** is a processor meant for applications that process digital signals. [For example, filtering, noise cancellation, echo elimination, compression and encryption].
2. Commonly used microprocessors, microcontrollers and DSPs in the small-, medium-and large-scale embedded systems.
3. A recently introduced technology that additionally incorporates the ***application-specific system processors*** (***ASSP***s) in the embedded systems.
4. Multiple processors in a system.

Section 1.3

1. Embedded system ***power source(s)*** and the need for controlled power-dissipation.
2. Embedded system ***Clock*** oscillator circuit and clocking unit. It lets a processor execute and process instructions.
3. ***Real time clock (RTC),*** timers and various timing needs of the system.
4. ***Reset circuit*** and ***watchdog timer***.
5. ***System memories***. [In the second part of Chapter 2, we will learn the system memories in detail].
6. System Input Output (***IO***) ports, serial Universal Asynchronous Receiver and Transmitter ***(UART)*** and other ports, multiplexers and demultiplexers and interfacing buses. [These will be dealt with in detail in Chapter 3].
7. Interrupt Handler [The latter part of Chapter 4 has the details].
8. Interfacing units- DAC (Digital to Analog Converter) using PWM (Pulse Width Modulation), ADC (Analog to Digital Converter), LED and LCD display units, keypad and keyboard, pulse dialer, modem and transceiver
9. Hardware required for exemplary embedded systems.

Section 1.4 Different levels of languages that are used to develop the *embedded* ***software*** for a system [Chapter 5 details the high-level programming aspects, 'C' and 'C++' language structures and the application of these for coding embedded software].

1. System device drivers, device management and multitasking using an operating system (OS) and ***real time operating system*** (RTOS). [RTOS (s) are dealt with in detail in Chapters 9 and 10, and case studies using RTOSs in Chapter 11].
2. Software tools in system designing.
3. Software tools required in six exemplary cases.
4. Programming models for software designing. [Software designing models are detailed in Chapter 6].

Section 1.5 ***Exemplary applications*** of each type of embedded system.

Section 1.6 Designing an embedded system on a VLSI chip.

1. Embedded ***SoC*** (System on Chip) and ***ASIC*** (Application Specific Integrated Circuit) and examples of their applications. These use (i) Application Specific Instruction Processor (ASIP), (ii) Intellectual Property (IP) core, (iii) Field Programmable Gate Arrays (FPGA) core with single or multiple processor units on an ASIC chip.
2. ***Smart card,*** an example of the units of an embedded system on a chip (SoC).

1.1 AN EMBEDDED SYSTEM

1.1.1 System

A system is a way of working, organizing or doing one or many tasks according to a fixed plan, program, or set of rules. A system is also an arrangement in which all its units assemble and work together according to the plan or program. Let us examine the following two examples.

Consider a watch. It is a **time-display system**. Its parts are its hardware, needles and battery with the beautiful dial, chassis and strap. *These parts organize to show* the real time every second and continuously update the time every second. The system-program updates the display using three needles after each second. *It follows a set of rules.* Some of these rules are as follows: (i) All needles move clockwise only. (ii) A thin and long needle rotates every second such that it returns to same position after a minute. (iii) A long needle rotates every minute such that it returns to same position after an hour. (iv) A short needle rotates every hour such that it returns to same position after twelve hours. (v) All three needles return to the same inclinations after twelve hours each day.

Consider a washing machine. It is an **automatic clothes-washing system**. The important hardware parts include its status display panel, the switches and dials for user-defined programming, a motor to rotate or spin, its power supply and control unit, an inner water-level sensor, a solenoid valve for letting water in and another valve for letting water drain out. *These parts organize* to wash clothes automatically according to a program preset by a user. *The system-program* is to wash the dirty clothes placed in a tank, which rotates or spins in pre-programmed steps and stages. *It follows a set of rules.* Some of these rules are as follows: (i) Follow the steps strictly in the following sequence. Step I: Wash by spinning the motor according to a programmed period. Step II: Rinse in fresh water after draining out the dirty water, and rinse a second time if the system is not programmed in water-saving mode. Step III: After draining out the water completely, spin fast the motor for a programmed period for drying by centrifuging out water from the clothes. Step IV: Show the wash-over status by a blinking display. Sound the alarm for a minute to signal that the wash cycle is complete. (ii) At each step, display the process stage of the system. (iii) In case of an interruption, execute only the remaining part of the program, starting from the position when the process was interrupted. There can be no repetition from Step I unless the user resets the system by inserting another set of clothes and resets the program.

1.1.2 Embedded System

A computer is a system that has the following or more components.

1. A microprocessor
2. A large memory comprising the following two kinds:
   (a) Primary memory (*semiconductor* memories - RAM, ROM and fast accessible caches)
   (b) Secondary memory (*magnetic* memory located in hard disks, diskettes and cartridge tapes and *optical* memory in CD-ROM)
3. Input units like keyboard, mouse, digitizer, scanner, etc.
4. Output units like video monitor, printer, etc.
5. Networking units like Ethernet card, front-end processor-based drivers, etc.

6. I/O units like a modem, fax cum modem, etc.

An embedded system is one that has computer-hardware with software embedded in it as one of its most important component. It is a dedicated computer-based system for an application(s) or product. It may be either an independent system or a part of a larger system. As its software usually embeds in ROM (Read Only Memory) it does not need secondary memories as in a computer. An embedded system has three main components:

1. It has hardware. Figure 1.1 shows the units in the hardware of an embedded system.
2. It has main application software. The application software may perform concurrently the series of tasks or multiple tasks.
3. It has a real time operating system (RTOS) that supervises the application software and provides a mechanism to let the processor run a process as per scheduling and do the context-switch between the various processes (tasks). RTOS defines the way the system works. It organizes access to a resource in sequence of the series of tasks of the system. It schedules their working and execution by following a plan to control the latencies and to meet the deadlines. [Latency refers to the waiting period between running the codes of a task and the instance at which the need for the task arises.] It sets the rules during the execution of the application software. A small-scale embedded system may not need an RTOS.

An embedded system has software designed to keep in view three constraints: (i) available system-memory, (ii) available processor speed and (iii) the need to limit power dissipation when running the system continuously in cycles of wait for events, run, stop and wake-up.

There are several definitions of embedded systems given in books published recently. Given below is a series of definitions from others in the field:

Wayne Wolf author of *Computers as Components – Principles of Embedded Computing System Design:*. "What is an *embedded computing system*? Loosely defined, it is any device that includes a programmable computer but is not itself intended to be a general-purpose computer" and "a fax machine or a clock built from a microprocessor is an embedded computing system".

Todd D. Morton author of *Embedded Microcontrollers*: "*Embedded Systems* are electronic systems that contain a microprocessor or microcontroller, but we do not think of them as computers - the computer is hidden or embedded in the system."

David E. Simon author of *An Embedded Software Primer*: "People use the term *embedded system* to mean any computer system hidden in any of these products."

Tim Wilmshurst author of *An Introduction to the Design of Small Scale Embedded System with examples from PIC, 80C51 and 68HC05/08 Microcontrollers*: (1) "An *embedded system* is a system whose principal function is not computational, but which is controlled by a computer embedded within it. The computer is likely to be a microprocessor or microcontroller. The word *embedded* implies that it lies inside the overall system, hidden from view, forming an integral part of greater whole". (2) "An embedded system is a microcontroller-based, software-driven, reliable, real time control system, autonomous, or human- or network-interactive, operating on diverse physical variables and in diverse environments, and sold into a competitive and cost-conscious market".

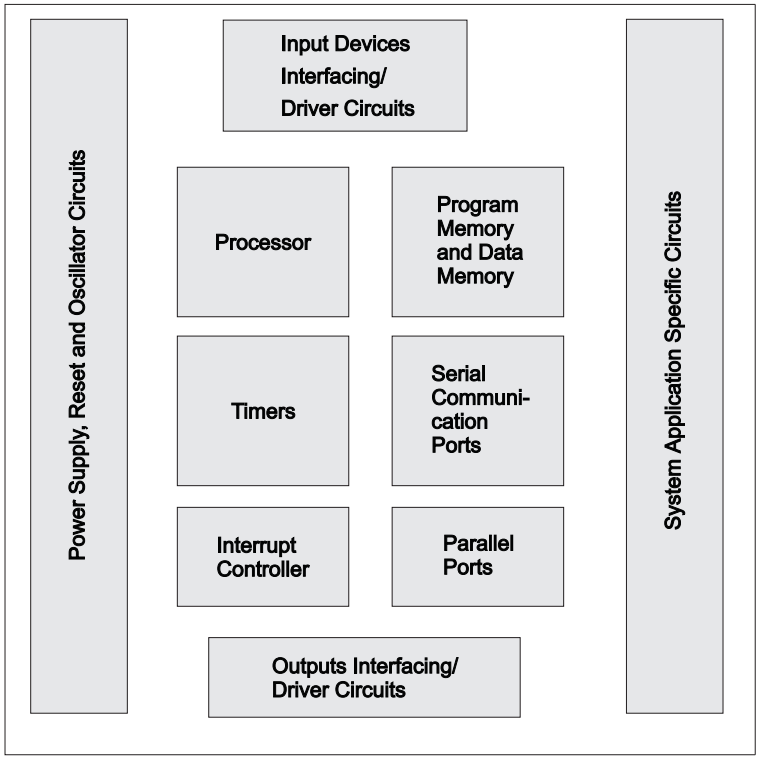

Figure 1.1

The components of an embedded system hardware.

1.1.3 Classification of Embedded Systems

We can classify embedded systems into three types as follows. [Section 1.5 will give examples of each type later]:

1. **Small Scale Embedded Systems:** These systems are designed with a single 8- or 16-bit microcontroller; they have little hardware and software complexities and involve board-level design. They may even be battery operated. When developing embedded software for these, an editor, assembler and cross assembler, specific to the microcontroller or processor used, are the main programming tools. Usually, 'C' is used for developing these systems. 'C' program compilation is done into the assembly, and executable codes are then appropriately located in the system memory. The software has to fit within the memory available and keep in view the need to limit power dissipation when system is running continuously.
2. **Medium Scale Embedded Systems:** These systems are usually designed with a single or few 16- or 32-bit microcontrollers or DSPs or Reduced Instruction Set Computers (RISCs). These have both hardware and software complexities. For complex software design, there are the following programming tools: RTOS, Source code engineering tool, Simulator, Debugger and Integrated Development Environment (IDE). Software tools also provide the solutions to the

hardware complexities. An assembler is of little use as a programming tool. These systems may also employ the readily available ASSPs and IPs (explained later) for the various functions—for example, for the bus interfacing, encrypting, deciphering, discrete cosine transformation and inverse transformation, TCP/IP protocol stacking and network connecting functions. [ASSPs and IPs may also have to be appropriately configured by the system software before being integrated into the system-bus.]

3. **Sophisticated Embedded Systems**: Sophisticated embedded systems have enormous hardware and software complexities and may need scalable processors or configurable processors and programmable logic arrays. They are used for cutting edge applications that need hardware and software co-design and integration in the final system; however, they are constrained by the processing speeds available in their hardware units. Certain software functions such as encryption and deciphering algorithms, discrete cosine transformation and inverse transformation algorithms, TCP/IP protocol stacking and network driver functions are implemented in the hardware to obtain additional speeds by saving time. Some of the functions of the hardware resources in the system are also implemented by the software. Development tools for these systems may not be readily available at a reasonable cost or may not be available at all. In some cases, a compiler or retargetable compiler might have to be developed for these. [A retargetable compiler is one that configures according to the given target configuration in a system.]

1.1.4 Skills required for an Embedded System Designer

An embedded system designer has to develop a product using the available tools within the given specifications, cost and time frame. [Chapters 7 and 12 will cover the design aspects of embedded systems. See also Section 1.5.]

1. *Skills for Small Scale Embedded System Designer:* Author Tim Wilmshurst in the book referred above (see page 4), has said that the following skills are needed in the individual or team that is developing a small-scale system: “Full understanding of microcontrollers with a basic knowledge of computer architecture, digital electronic design, software engineering, data communication, control engineering, motors and actuators, sensors and measurements, analog electronic design and IC design and manufacture”. Specific skills will be needed in specific situations. For example, control engineering knowledge will be needed for design of control systems and analog electronic design knowledge will be needed when designing the system interfaces. Basic aspects of the following topics will be described in this book to prepare the designer who already has a good knowledge of the microprocessor or microcontroller to be used. (i) Computer architecture and organization. (ii) Memories. (iii) Memory allocation. (iv) Interfacing the memories. (v) Burning (a term used for porting) the executable machine codes in PROM or ROM (Section 2.3.1). (vi) Use of decoders and demultiplexers. (vii) Direct memory accesses. (viii) Ports. (ix) Device drivers in assembly. (x) Simple and sophisticated buses. (xi) Timers. (xii) Interrupt servicing mechanism. (xiii) C programming elements. (xiv) Memory optimization. (xv) Selection of hardware and microcontroller. (xvi) Use of ICE (In-Circuit-Emulators), cross-assemblers and testing equipment. (xvii) Debugging the software and hardware bugs by using test vectors. Basic knowledge in the other areas—data communication, control engineering, motors and actuators, sensors and measurements, analog electronic design and IC design and manufacture—can be obtained from the standard textbooks available.

A designer interested in small-scale embedded systems may not need at all concepts of interrupt latencies and deadlines and their handling, the RTOS programming tools described in Chapters 9 and 10 and program designing models given in Chapter 6.

2. *Skills for Medium Scale Embedded System Designer:* 'C' programming and RTOS programming and program modeling skills are a must to design a medium-scale embedded-system. Knowledge of the following becomes critical. (i) Tasks and their scheduling by RTOS. (ii) Cooperative and preemptive scheduling. (iii) Inter processor communication functions. (iv) Use of shared data, and programming the critical sections and re-entrant functions. (v) Use of semaphores, mailboxes, queues, sockets and pipes. (vi) Handling of interrupt-latencies and meeting task deadlines. (vii) Use of various RTOS functions. (viii) Use of physical and virtual device drivers. [Refer to Chapters 8 to 10 for detailed descriptions of these seven along with examples and to Chapter 11 to learn their use with the help of case studies.] A designer must have access to an RTOS programming tool with Application Programming Interfaces (APIs) for the specific microcontroller to be used. Solutions to various functions like memory-allocation, timers, device drivers and interrupt handing mechanism are readily available as the APIs of the RTOS. The designer needs to know only the hardware organization and use of these APIs. The microcontroller or processor then represents a small system element for the designer and a little knowledge may suffice.
3. *Skills for Sophisticated Embedded System Designer:* A team is needed to co-design and solve the high level complexities of the hardware and software design. An embedded system hardware engineer should have full skills in hardware units and basic knowledge of 'C', RTOS and other programming tools. Software engineer should have basic knowledge in hardware and a thorough knowledge of 'C', RTOS and other programming tools. A final optimum design solution is then obtained by system integration.

1.2 PROCESSOR IN THE SYSTEM

A ***processor*** is the heart of the embedded system. For an embedded system designer, knowledge of microprocessors and microcontrollers is a prerequisite. In the following explanations, too, it has been presumed that the reader has a thorough understanding of microprocessors or microcontrollers. [The reader may refer to a standard text or the texts listed in the 'References' at the end of this book for an in-depth understanding of microprocessors, microprocessors and DSPs that are incorporated in embedded system design.]

1.2.1 Processor in a System

A processor has two essential units: Program Flow Control Unit (CU) and Execution Unit (EU). The CU includes a fetch unit for fetching instructions from the memory. The EU has circuits that implement the instructions pertaining to data transfer operations and data conversion from one form to another. The EU includes the Arithmetic and Logical Unit (ALU) and also the circuits that execute instructions for a program control task, say, halt, interrupt, or jump to another set of instructions. It can also execute instructions for a call or branch to another program and for a call to a function.

A processor runs the cycles of fetch and execute. The instructions, defined in the processor instruction set, are executed in the sequence that they are fetched from the memory. A processor is

mostly in the form of an IC chip; alternatively, it could be in core form in an ASIC or at a SoC. Core means a part of the functional circuit on the VLSI chip.

An embedded system processor chip or core can be one of the following.

1. General Purpose Processor (GPP):
   a. Microprocessor. [Refer to Section 1.2.2.]
   b. Microcontroller. [Refer to Section 1.2.3.]
   c. Embedded Processor. [Refer to Section 1.2.4.]
   d. Digital Signal Processor (DSP). [Refer to Section 1.2.5.]
   e. Media Processor. [Refer to Appendix E Section E.1.]
2. Application Specific System Processor (ASSP) as additional processor [Refer to Section 1.2.6.]
3. Multiprocessor system using General Purpose processors (GPPs) and Application Specific Instruction Processors (ASIPs) [Refer to Section 1.2.7.]
4. GPP core (s) or ASIP core (s) integrated into either an Application Specific Integrated Circuit (ASIC), or a *Very Large Scale Integrated Circuit* (VLSI) circuit or an FPGA core integrated with processor unit(s) in a VLSI (ASIC) chip. [Refer to Section 1.6.]

For a system designer, the following are important considerations when selecting a processor:

1. Instruction set.
2. Maximum bits in an operand (8 or 16 or 32) in a single arithmetic or logical operation.
3. Clock frequency in MHz and processing speed in Million Instructions Per Second (***MIPS***). [Refer to Appendix B for an alternate metric ***Dhyrystone*** for processing performance.]
4. Processor ability to solve the complex algorithms used in meeting the deadlines for their processing.

A general-purpose processor is used because of the following: (i) Processing by the known instructions available at predefined General-Purpose Instruction-Set results in fast system development. (ii) Once the board and I/O interfaces are designed for a GPP, these can be used for a new system by just changing the embedded software in the board ROM. (iii) Ready availability of a compiler facilitates embedded software development in high-level language. (iv) Ready availability of well-tested and debugged processor-specific APIs and the codes previously designed for other applications results in fast development of a new system.

1.2.2 Microprocessor

The CPU is a unit that centrally fetches and processes a set of general-purpose instructions. The CPU instruction set (Section 2.4) includes instructions for *data transfer* operations, *ALU* operations, *stack* operations, *input and output* (I/O) operations and *program control*, *sequencing* and *supervising* operations. The general purpose instruction set (refer to Appendix A, Section A.1) is always specific to a specific CPU. Any CPU must possess the following basic functional units.

1. A control unit to fetch and control the sequential processing of a given command or instruction and for communicating with the rest of the system.
2. An ALU for the arithmetic and logical operations on the bytes or words. It may be capable of processing 8, 16, 32 or 64 bit words at an instant.

A microprocessor is a single VLSI chip that has a CPU and may also have some other units (for examples, *caches, floating point processing arithmetic unit, pipelining* and *super-scaling units*) that are additionally present and that result in faster processing of instructions. [Refer to Section 2.1.]

The earlier generation microprocessor's fetch-and-execute cycle was guided by clock frequency of the order of ~1 MHz. Processors now operate at clock frequency of 2 GHz. [Intel released a 2 GHz processor on August 25, 2001. This also marked the twentieth anniversary of the introduction of the IBM PC. Intel released 3 GHz Pentium 4 on April 14, 2003.] Since early 2002, a few highly sophisticated embedded systems (for examples, Gbps transceiver and encryption engine) have incorporated the GHZ processor. [Gbps means Giga bit per second. Transceiver means a transmitting cum receiving circuit with appropriate processing and controls, for example, for bus-collisions.]

One example of an older generation microprocessor is Intel 8085. It is an 8-bit processor. Another is Intel 8086 or 8088, which is a 16-bit processor. *Intel 80x86 (also referred as x86) processors are the 32-bit successors of 8086.* [The *x* here means extended 8086 for 32 bits.] Examples of 32-bit processors in 80x86 series are Intel 80386 and 80486. Mostly, the IBM PCs use 80*x*86 series of processors and the embedded systems incorporated inside the PC for specific tasks (like graphic accelerator, disk controllers, network interface card) use these microprocessors.

An example of the new generation 32- and 64-bit microprocessor is the classic ***Pentium*** series of processors from Intel. These have superscalar architecture [Section 2.1]. They also possess powerful ALUs and Floating Point Processing Units (FLPUs) [Table 2.1]. An example of the use of Pentium III operating at 1 GHz clock frequency in an embedded system is the 'Encryption Engine'. This gives encrypted data at the rate of 0.464 Gbps.

Table 1.1.lists the important microprocessors used in the embedded systems. The microprocessors are among the following streams of families:

Table 1.1
Important Microprocessors used in the Embedded Systems

Stream	Microprocessor Family	Source	CISC or RISC or Both features
Stream 1	68HCxxx	Motorola	CISC
Stream 2	(a) 80x86	Intel	CISC
	(b) i860	Intel	CISC with RISC
Stream 3	SPARC	Sun	RISC
Stream 4	(a) PowerPC 601, 604	IBM	RISC
	(b) MPC 620	Motorola	

The microprocessors from Streams 1 and 2 have Complicated Instruction Set Computer (CISC) architecture [Section A.1]. Microprocessors form Streams 3 and 4 have Reduced Instruction Set Computer (RISC) architecture [Section A.1.4]. An RISC processor provides speedy processing of the instructions, each in a single clock-cycle. Further, besides the greatly enhanced capabilities mentioned above, there is great enhancement of the speed by which an instruction from a set is processed. Thumb' Instruction set is a new industry standard that also gives a reduced code density in a RISC processor. [The concepts of architecture features of the processor in an embedded system, ***CISC and***

RISC processors and processor instruction-set will be explained later in Appendices A and B.] ***RISCs are used when the system needs to perform intensive computation, for example, in a speech processing system.***

How does a system designer select a microprocessor? This will be explained in Section 2.2.

A microprocessor is used when large embedded software is to be located in the external memory chips. RISC core microprocessor is used when intensive computations are to be performed.

1.2.3 Microcontroller

Just as a microprocessor is the most essential part of a computing system, a microcontroller is the most essential component of a control or communication circuit. ***A microcontroller is a single-chip VLSI unit (also called 'microcomputer') which, though having limited computational capabilities, possesses enhanced input-output capabilities and a number of on-chip functional units***. [Refer to Section 1.3 for various functional units.] Microcontrollers are particularly suited for use in embedded systems for real-time control applications with *on-chip* program memory and devices.

Figure 1.2 shows the functional circuits present (in solid boundary boxes) in a microcontroller. It also shows the application-specific units (in dashed boundary boxes) in a specific version of a given microcontroller family. A few of the latest microcontrollers also have high computational and superscalar processing capabilities. [For the meaning of superscalar architecture, refer to Section 2.1.] Appendix C gives the comparative functionalities of select microcontroller representatives from these families.

Important microcontroller chips for embedded systems are usually among the following five streams of families given in Table 1.2.

Table 1.2
Important Microcontrollers[@] Used in the Embedded Systems

Stream	Microcontroller Family	Source	CISC or RISC or Both features
Stream 1	68HC11xx, HC12xx, HC16xx	Motorola	CISC
Stream 2	8051, 80251	Intel	CISC
Stream 3	80x86[$]	Intel	CISC
Stream 4	PIC 16F84 or 16C76, 16F876 and PIC18	Microchip	CISC
Stream 5[*]	Enhancements of ARM9, ARM7	ARM, Texas, etc.	CISC with RISC Core

[@] Other popular microcontrollers are as follows. (i) Hitachi H8x family and SuperH 7xxx. (ii) Mitsubishi 740, 7700, M16C and M32C families. (iii) National Semiconductor COP8 and CR16 /16C. (iv) Toshiba TLCS 900S (v) Texas Instruments MSP 430 for low voltage battery based system. (vi) Samsung SAM8. (vii) Ziglog Z80 and eZ80
[$] 80x86 Microcontroller versions (typically 80188 eight bit processor or 80386 sixteen bit processor) with each, there are the 64 kB memory, 3 timers and 2 DMA channels.
[*] Refer Sections 1.2.4 and B.1

Figure 1.3 shows commonly used microcontrollers in the small-, medium- and large-scale embedded systems. In Section C.1 (refer to Tables C.1.1 to C.1.3 therein) those features will be described that have to be considered by a system designer before choosing a microcontroller as a processing unit.

A microcontroller is used when a small or part of the embedded software has to be located in internal memory and when the on-chip functional units like interrupt-handler, port, timer, ADC and PWM are needed.

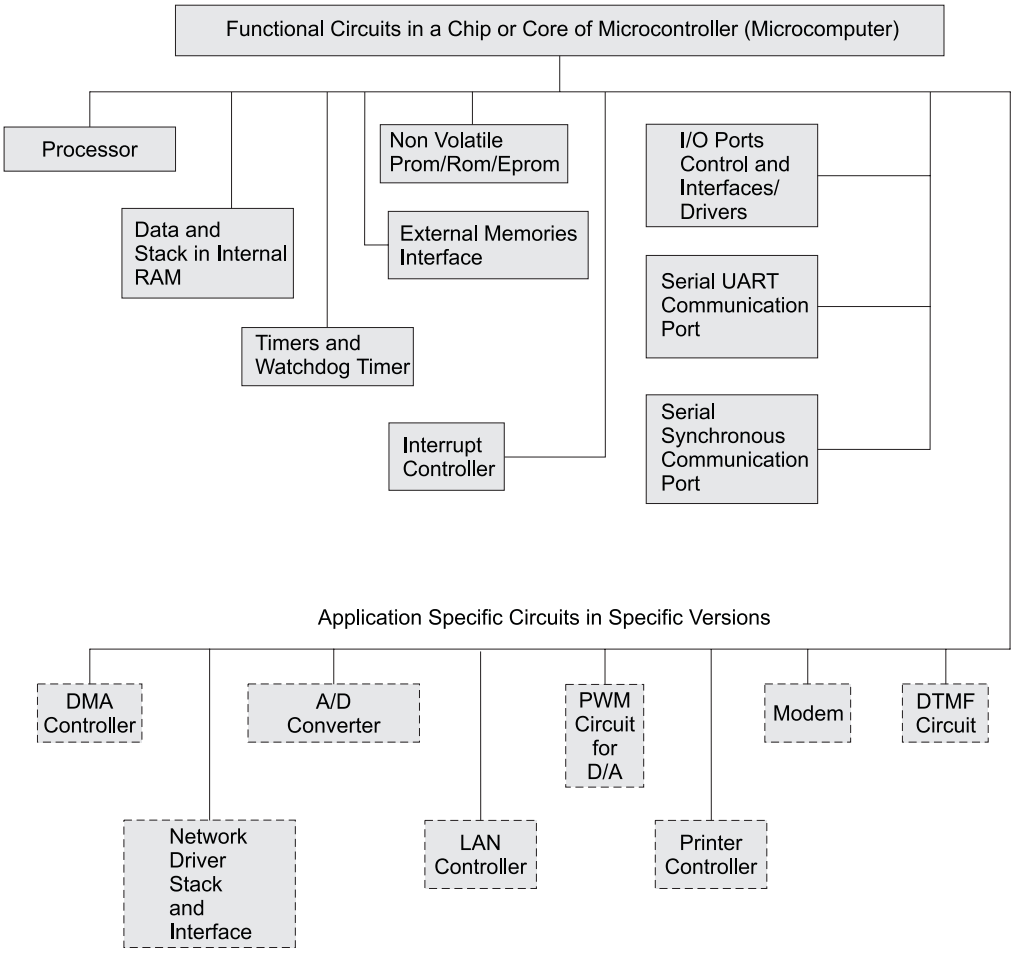

Figure 1.2

Various functional circuits (solid boundary boxes) in a microcontroller chip or core in an embedded system. Also shown are the application-specific units (dashed boundary boxes) in a specific version of a microcontroller.

1.2.4 Embedded Processor for a Complex System

For fast, precise and intensive calculations and for complex real time applications, the microcontrollers and microprocessors mentioned above do not suffice. An electronics warfare system, for example, an Advanced Warning and Control System (AWACS), which also associates tracking radar, is an example of a complex real-time system. Special microprocessors and microcontrollers, often called embed-

ded processors, are required.When a microcontroller or microprocessor is specially designed such that it has the following capabilities, then the term *embedded processor* is preferred instead of *microcontroller or microprocessor*.

1. Fast context switching and thus lower latencies of the tasks in complex real time applications. [Refer to Section 4.6]
2. Atomic ALU operations and thus no shared data problem. The latter occurs due to an incomplete ALU (non-atomic) operation when an operand of a larger number of bits is placed in two or four registers. [Refer to Section 2.1.]
3. RISC core for fast, more precise and intensive calculations by the embedded software.

Calculations for real time image processing and for aerodynamics are two examples where there is a need for fast, precise and intensive calculations and fast context-switching. Important embedded processor chips for embedded systems belong to the following two streams of families.

- Stream 1: ARM family ARM 7* and ARM 9*
- Stream 2: Intel family i960.
- Stream 3: AMD family 29050.

* Appendix B describes ARM family processors. These are available in single chip CPU version as well in file version for embedding on a VLSI chip or for a SoC solution for the embedded system. Refer to Section 1.6.

Intel family i960 microcontrollers are also called embedded processors, as these possess the required features including CISC and RISC (Section A.1.4). In one of the versions, these also have a 4-channel DMA controller (Section 2.6). An 80960 includes an 8-channel, 248-vector programmable interrupt controller.

An embedded processor is used when besides fast processing fast context-switching and atomic ALU operations are needed.

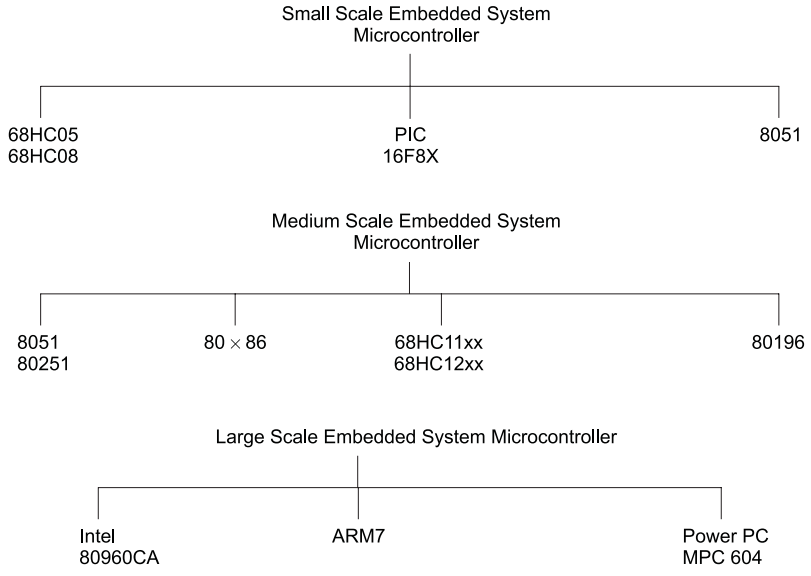

Figure 1.3

Commonly used microcontrollers in small-, medium- and large-scale embedded systems.

1.2.5 Digital Signal Processor (DSP)

Just as a microprocessor is the most essential unit of a computing system, a digital signal processor (DSP) is an essential unit of an embedded system for a large number of applications needing processing of signals. Exemplary applications are in image processing, multimedia, audio, video, HDTV, DSP modem and telecommunication processing systems. DSPs also find use in systems for recognizing an image pattern or a DNA sequence fast. Appendix D describes in detail the embedded system DSPs.

The DSP as a GPP is a single chip VLSI unit. It possesses the computational capabilities of a microprocessor and also has a Multiply and Accumulate (MAC) unit(s). Nowadays, a typical DSP has a 16 x 32 MAC unit.

A DSP provides fast, discrete-time, signal-processing instructions. It has Very Large Instruction Word (VLIW) processing capabilities; it processes Single Instruction Multiple Data (SIMD) instructions fast; it processes Discrete Cosine Transformations (DCT) and inverse DCT (IDCT) functions fast. The latter are a must for fast execution of the algorithms for signal analysing, coding, filtering, noise cancellation, echo-elimination, compressing and decompressing, etc.

Important DSPs for the embedded systems are from three streams as given in Table 1.3.

Table 1.3
Important Digital Signal Processor@ Used in the Embedded Systems

Stream	DSP Family	Source
Stream 1	TMS320Cxx+	Texas
Stream 2	SHARC	Analog Device
Stream 3	5600xx	Motorola

+ For example, TMS320C62XX for fixed point DSP at clock speed of 200 MHz. Refer to Section D.4 for a detailed description.

A DSP is used when signal-processing functions are to be processed fast.

1.2.6 Application Specific System Processors (ASSPs) in Embedded Systems

Lately a new class of embedded systems has emerged. Systems additionally incorporates the Application Specific System processor ASSP chip(s) or core (s) in its design. These have been recently become available.

Assume that there is an embedded system for real-time video processing. Real-time processing need in embedded systems arises for digital television, high definition TV decoders, set-up boxes, DVD (Digital Video Disc) players, Web phones, video-conferencing and other systems. The processing needs a video compression and decompression system, which incorporates an MPEG 2 or MPEG 4 standard. [MPEG stands for Motion Picture Expert Group.] MPEG 2 or MPEG 4 compression of signals is done before storing or transmitting; decompression is done before retrieving or receiving these signals. For MPEG compression algorithms, if a GPP embedded software is run, separate DSP(s) are required to achieve real-time processing. An ASSP that is dedicated to these specific tasks

alone provides a faster solution. The ASSP is configured and interfaced with the rest of the embedded system.

Assume that there is an embedded system that interconnects using a specific protocol the system units through a specific bus architecture to another system. Also, assume that there is a need for suitable encryption and decryption. [The output bit stream encryption protects the messages or design from passing to an unknown external entity.] For these tasks, besides embedding the software, it may also be necessary to embed some RTOS features. [Section 1.4.6]. If the software alone is used for the above tasks, it may take a longer time than a hardwired solution for application-specific processing. An ASSP chip provides such a solution. For example, an ASSP chip [from i2Chip (http://www.i2Chip.com)] has a TCP, UDP, IP, ARP, and Ethernet 10/100 MAC (Media Access Control) hardwired logic included into it. The chip from i2Chip, W3100A, is a unique hardwired Internet connectivity solution. Much needed TCP/IP stack processing software for networking tasks is thus available as a hardwired solution. This gives output five times faster than a software solution using the system's GPP. It is also an RTOS-less solution. Using the same microcontroller in the embedded system to which this ASSP chip interfaces, Ethernet connectivity can be added. [For terms TCP, UDP, IP, ARP, Ethernet 10/100 and MAC, refer to a suitable reference book on computer networking. Refer to *Internet and Web Technologies by RajKamal, Tata McGraw-Hill, 2002,* to understand the meaning of each bit in these protocols.]

Another ASSP example is 'Serial-to-Ethernet Converter' (IIM7100). It does real-time data processing by a hardware protocol stack. It needs no change in the application software or firmware and provides the economical and smallest RTOS solution.

An ASSP is used as an additional processing unit for running the application specific tasks in place of processing using embedded software.

1.2.7 Multi-processor Systems using General Purpose Processors (GPP)

In an embedded system, several processors may be needed to execute an algorithm fast and within a strict dead line. For example, in real-time video processing, the number of MAC operations needed per second may be more than is possible from one DSP unit. An embedded system then may have to incorporate two or more processors running in synchronization.

In a cell-phone, a number of tasks have to be performed: (a) Speech signal-compression and coding. (b) Dialing. (c) Modulating and Transmitting. (d) Demodulating and Receiving. (e) Signal decoding and decompression. (f) Keypad interface and display interface handling. (g) Short Message Service (SMS) protocol-based messaging. (h) SMS message display. For all these tasks, a single processor does not suffice. Suitably synchronized multiple processors are required.

Consider a video conferencing system. In this system, a quarter common intermediate format—Quarter-CIF)—is used. Image pixel is just 144 x 176 as against 525 x 625 pixels in a video picture on TV. Even then, samples of the image have to be taken at a rate of 144 x 176 x 30 = 760320 pixels per second and have to be processed by compression before transmission on a telecommunication or Virtual Private Network (VPN). [Note: The number of frames should be 25 or 30 per second (as per the standard adopted) for real-time displays and in motion pictures and between 15 and 10 for video conferencing.] A single DSP-based embedded system does not suffice to get real-time images. *Real-time video processing and multimedia applications most often need a multiprocessor unit in the*

embedded system. [A Media processor, described in Appendix E, is an alternate solution in place of use of multiprocessors for real time video processing.]

Multiple processors are used when a single processor does not meet the needs of the different tasks that have to be performed concurrently. The operations of all the processors are synchronized to obtain an optimum performance.

1.3 OTHER HARDWARE UNITS

1.3.1 Power Source and Managing the Power Dissipation and Consumption

Most systems have a ***power supply*** of their own. The supply has a specific operation range or a range of voltages. Various units in an embedded system operate in one of the following four operation ranges:

(i) $5.0V \pm 0.25V$
(ii) $3.3V \pm 0.3V$
(iii) $2.0 \pm 0.2V$
(iv) $1.5V \pm 0.2V$

Additionally, a $12V \pm 0.2V$ supply is needed for a flash (a memory form used in systems like latest digital cameras) or Electrically Erasable and Programmable Read Only memory (EEPROM) when present in the microcontroller of an embedded system and for RS232C serial Interfaces (Section 2). [Lately, flash memory needed supply voltages are 5V or less.]

Voltage is applied on the chips of an embedded system as follows. The flow of voltage and the connections depend on the number of supply pins provided within the processor plus the pins in the associated chips and circuits. The pins are in pairs, consisting of the supply in and the ground line. The following points have to be taken care of while connecting the supply rails (lines):

1. A processor may have more than two pins of V_{DD} and V_{SS}. This distributes the power in all the sections and reduces interference between sections. There should be a separate radio frequency interference bypassing capacitor as close as possible to each pair of V_{DD} and V_{SS} pins in the system processor as well as in other units.
2. Supply should separately power the (a) external I/O driving ports (b) timers and (c) clock and reset circuits. Clock and reset circuits (Sections 1.3.2 and 1.3.3) need to be specially designed to be free from any radio frequency interference. An I/O device may dissipate more power than the other internal units of the processor. A timer may dissipate a constant power even in *Wait* state. Hence, these three circuits are powered separately.
3. From the supply, there should be separate interconnections for pairs of V_{DD} and V_{SS} pins, analog ground, analog reference and analog input voltage lines, the ADC unit digital ground and other analog parts in the system. An ADC needs stringent noise-free supply inputs.

Certain systems do not have a power source of their own: they connect to an external ***power supply*** or are powered by the use of ***charge pumps***. (1) Network Interface Card (NIC) and Graphic Accelerator are examples of embedded systems that do not have their own power supply and connect to PC power-supply lines. (2) A charge pump consists of a diode in the series followed by a charging capacitor. The diode gets forward bias input from an external signal; for example, from an RTS signal

in the case of the mouse used with a computer. Charge pumps bring the power from a non-supply line. [Ninepins COM port has a signal called Request To Send (RTS). It is an active low signal. Most of the time it is in inactive state logic '1' (~5V). The charge pump inside the mouse uses it to store the charge when the mouse is in an idle state; the pump dissipates the power when the mouse is used]. A regulator circuit getting input from this capacitor gives the required voltage supply. A charge pump in a contact-less smart card uses the radiations from a host machine when inserted into that [Section 1.6.6].

Low voltage systems are built using LVCMOS (Low Voltage CMOS) gates and LVTTL (Low Voltage TTL). Use of 3.3V, 2.5V, 1.8V and 1.5 Volt systems and IO (Input-Output) Interfaces other than the conventional 5V systems results in significantly reduced power-consumption and can be advantageously used in the following cases:

(a) In portable or hand-held devices such as a cellular phone [Compared to 5V, a CMOS circuit power dissipation reduces by half, ~$(3.3/5)^2$, in 3.3V operation. This also increases the time intervals needed for recharging the battery by a factor of two]. (b) In a system with smaller overall geometry, the low voltage system processors and IO circuits generate lesser heat and thus can be packed into a smaller space.

There is generally an inverse relationship between the propagation delay in the gates and operational voltage. Therefore, the 5V-system processor and units are also used in most systems.

An embedded system may need to be run continuously, without being switched off; the system design, therefore, is constrained by the need to limit power dissipation while it is running. Total power consumption by the system in running, waiting and idle states should also be limited. The *current* needed at any instant in the processor of an embedded system depends on the state and mode of the processor. The following are the typical values in six states of the processor.

(i) 50 mA when only the processor is running:that is, the processor is executing instructions.
(ii) 75 mA when the processor plus the external memories and chips are in running: state: that is, fetching and execution are both in progress.
(iii) 15mA when only the processor is in stop state: that is, fetching and execution have both stopped and the clock has been disabled from all structural units of the processor.
(iv) 15 mA when the processor plus the external memories and chips are in stop state: that is, fetching and execution have both stopped and the clock disabled from all system units.
(v) 5 mA when only the processor is in waiting state: that is, fetching and execution have both stopped but the clock has not been disabled from the structural units of the processor, such as timers.
(vi) 10 mA when the processor, the external memories and the chips are in waiting state. Waiting state now means that fetching and execution have both stopped; but the clock has not been disabled from the structural units of the processor and the external IO units and dynamic RAMs refreshing also has not stopped.

An embedded system has to perform tasks continuously from power-up and may also be left in power-ON state; therefore, power saving during execution is important A microcontroller used in the embedded system must provide for executing *Wait* and *Stop* instructions and operation in power-down mode. One way to do this is to cleverly incorporate into the software the *Wait* and *Stop* instructions. Another is to operate the system at the lowest voltage levels in the idle state by selecting power-down mode in that state. Yet another method is to disable use of certain structural units of the processor—for example, caches—when not necessary and to keep in disconnected state those structure units that are not needed during a particular software-portion execution, for example timers or IO units. In a

CMOS circuit, power dissipates only at the instance of change in input. Therefore, unnecessary glitches and frequent input changes increase power dissipation. VLSI circuit designs have a unique way of avoiding power dissipation. A circuit design is made such that it eliminates all removable glitches, thereby eliminating any frequent input changes.

Note 1 The processor goes into a stop state when it receives a Stop instruction. The stop state also occurs in the following conditions: (1) On disabling the clock inputs to the processor. (2) On stopping the external clock circuit functions. (3) On the processor operating in auto-shutdown mode. When in stop state, the processor disconnects with the buses. [Buses become in tri-state.] The stop state can change to a running state. The transition to the running state is either because of a user interrupt or because of the periodically occurring wake-up interrupts.

Note 2 The processor goes into a waiting state either on receiving (i) an instruction for *Wait*, which slows or disables the clock inputs to some of the processor units including ALU, or (ii) when an external clock-circuit becomes non-functional. The timers are still operating in the waiting state. The waiting state changes to the running state when either (i) an interrupt occurs or (ii) a reset signals.

Note 3 Power dissipation reduces typically by 2.5 mW per 100 kHz reduced clock rate. So reduction from 8000 kHz to 100 kHz reduces power dissipation by about 200 mW, which is nearly similar to when the clock is non-functional. [Remember, total power dissipated (energy required) may not reduce. This is because on reducing the clock rate the computations will take a longer time at the lower clock rate and the total energy required equals the power dissipation per second multiplied by the time.] The power 25 mW is typically the residual dissipation needed to operate the timers and few other units. By operating the clock at lower frequency or during the power-down mode of the processor, the advantages are as follows: (i) Heat generation reduces. (ii) Radio frequency interference also then reduces due to the reduced power dissipation within the gates. [Radiated RF (Radio Frequency) power depends on the RF current inside a gate, which reduces due to increase in 'ON' state resistance between the drain and channel when there is reduced heat generation.]

Lately, a new technology is the use of clock manager circuits in conjunction with oscillator circuits. It is used in sophisticated embedded systems on chips (SoCs). Two to sixteen synchronous clocks are created by the combination of clock doublers and clock dividers (by 2) . Further, incoming clock signals at the bus may be divided first and then multiplied before being applied to a fast operation circuit. This reduces the power consumption between gates. The clock manager circuit is configured for the smart delivery of the appropriate frequency clock to each section of the circuit being managed during real-time processing. [Note: A sophisticated technology— *phased delay locked loops*—has to be used. When using the common logic gates of counters, there are continuously varying delays at the gates (say, for example, 10 ns plus minus 2 ns). The synchronous clocks cannot be designed by using the counters alone.]

An internal power source or a charge pump is essential in every system. An embedded system has to perform tasks continuously from power-up to power-off and may even be kept 'on' continuously. Clever real-time programming by using 'Wait' and 'Stop' instructions and disabling certain units when not needed is one method of saving power during program execution. Operations can also be performed at reduced clock rate when needed in order to control power dissipation; yet all the tasks must execute within the set deadlines and all tasks needing full

speed processing must process fast. [For a definition of 'deadline', refer to Section 4.6.] For embedded system software, a performance analysis during its design phase must also include the analysis of power dissipation during program execution and during standby. Good design must optimize the conflicting needs of low power dissipation and fast and efficient program execution.

1.3.2 Clock Oscillator Circuit and Clocking Unit (s)

After the power supply, the clock is the next important unit of a system. A processor needs a **clock oscillator** circuit. The clock controls the various clocking requirements of the CPU, of the system timers and the CPU machine cycles. The machine cycles are for

(i) fetching the codes and data from memory and then decoding and executing at the processor, and
(ii) transferring the results to memory.

The *clock* controls the time for executing an instruction. The clock circuit uses either a crystal (external to the processor) or a ceramic resonator (internally associated with the processor) or an external oscillator IC attached to the processor. (a) The crystal resonator gives the highest stability in frequency with temperature and drift in the circuit. The crystal in association with an appropriate resistance in parallel and a pair of series capacitance at both pins resonates at the frequency, which is either double or single times the crystal-frequency. Further, the crystal is kept as near as feasible to two pins of the processor. (b) The internal ceramic resonator, if available in a processor, saves the use of the external crystal and gives a reasonable though not very highly stable frequency. [A typical drift of the ceramic resonator is about ten minutes per month compared to the typical drift of 1 or 5 minutes per month of a crystal]. (c) The external IC-based clock oscillator has a significantly higher power dissipation compared to the internal processor-resonator. However, it provides a higher driving capability, which might be needed when the various circuits of embedded system are concurrently driven. For example, a multiprocessor system needs the clock circuit, which should give a high driving capability and enables control of all the processors concurrently.

For the processing unit(s), a highly stable oscillator is required and the processor clock-out signal provides the clock for synchronizing all the system units.

1.3.3 Real Time Clock (RTC) and Timers for Various Timing and Counting Needs of the System

A timer circuit suitably configured is the ***system-clock***, also called real-time clock (RTC). An RTC is used by the schedulers and for real-time programming. An RTC is designed as follows: Assume a processor generates a clock output every 0.5 ms. When a system timer is configured by a software instruction to issue timeout after 200 inputs from the processor clock outputs, then there are 10000 interrupts (ticks) each second. The RTC ticking rate is then 10 kHz and it interrupts every 100 ms. The RTC is also used to obtain software-controlled delays and time-outs.

More than one timer using the system clock (RTC) may be needed for the various timing and counting needs in a system. Refer to Section 3.2 for a description of timers and counters.

For scheduling the various tasks and for real-time programming, a system clock (RTC) is needed. The system clock also drives the timers for various timing and counting needs in a system.

1.3.4 Reset Circuit, Power-up Reset and Watchdog-Timer Reset

Reset means that the processor starts the processing of instructions from a starting address. That address is one that is set by default in the processor program counter (or instruction pointer and code segment registers in x86 processors) on a power-up. From that address in memory, the fetching of program-instructions starts following the *reset* of the processor. [In certain processors, for example, 68HC11 and HC12, there are two start-up addresses. One is as per power-up reset vector and other is as per reset vector after the *Reset* instruction or after a time-out (for example from a watchdog timer)].

The reset circuit activates for a fixed period (a few clock cycles) and then deactivates. The processor circuit keeps the reset pin active and then deactivates to let the program proceed from a default beginning address. The reset pin or the internal reset signal, if connected to the other units (for example, I/O interface or Serial Interface) in the system, is activated again by the processor; it becomes an outgoing pin to enforce reset state in other sister units of the system. On deactivation of the *reset* that succeeds the processor activation, a program executes from start-up address.

Reset can be activated by one of the following:

1. An external reset circuit that activates on the power-up, on switching-on reset of the system or on detection of a low voltage (for example < 4.5V when what is required is 5V on the system supply rails). This circuit output connects to a pin called the reset pin of the processor. This circuit may be a simple RC circuit, an external IC circuit or a custom-built IC. The examples of the ICs are MAX 6314 and Motorola MC 34064.
2. By (a) software instruction or (b) time-out by a programmed timer known as watchdog timer (or on an internal signal called COP in 68HC11 and 68HC12 families) or (c) a clock monitor detecting a slowdown below certain threshold frequencies due to a fault.

The watchdog timer is a timing device that resets the system after a predefined timeout. This time is usually configured and the watchdog timer is activated within the first few clock cycles after power-up. It has a number of applications. In many embedded systems reset by a watchdog timer is very essential because it helps in rescuing the system if a fault develops and the program gets stuck. On restart, the system can function normally. Most microcontrollers have on-chip watchdog timers.

Consider a system controlling the temperature. Assume that when the program starts executing, the sensor inputs work all right. However, before the desired temperature is achieved, the sensor circuit develops some fault. The controller will continue delivering the current nonstop if the system is not reset. Consider another example of a system for controlling a robot. Assume that the interfacing motor control circuit in the robot arm develops a fault during the run. In such cases, the robot arm may continue to move unless there is a watchdog timer control. Otherwise, the robot will break its own arm!

An important circuit that associates a system is its reset circuit. A program that is reset and runs on a power-up can be one of the following: (i) A system program that executes from the beginning. (ii) A system boot-up program. (iii) A system initialization program.
The watchdog timer reset is a very useful feature in control applications.

1.3.5 Memories

In a system, there are various types of memories. Figure 1.4 shows a chart for the various forms of memories that are present in systems. These are as follows: (i) Internal RAM of 256 or 512 bytes in a microcontroller for registers, ***temporary data and stack.*** (ii) Internal ROM/PROM/EPROM for about 4 kB to 16 kB of program (in the case of microcontrollers). (iii) External RAM for the ***temporary data and stack*** (in most systems). (iv) Internal caches (in the case of certain microprocessors). (v) EEPROM or flash (in many systems saving the results of processing in nonvolatile memory: for example, system status periodically and digital-camera images, songs, or speeches after a suitable format compression). (vi) External ROM or PROM for embedding software (in almost all non-microcontroller-based systems). (vii) RAM Memory buffers at the ports. (viii) Caches (in superscaler microprocessors). [Refer to Sections 2.1 and 2.3 for further details of these.]

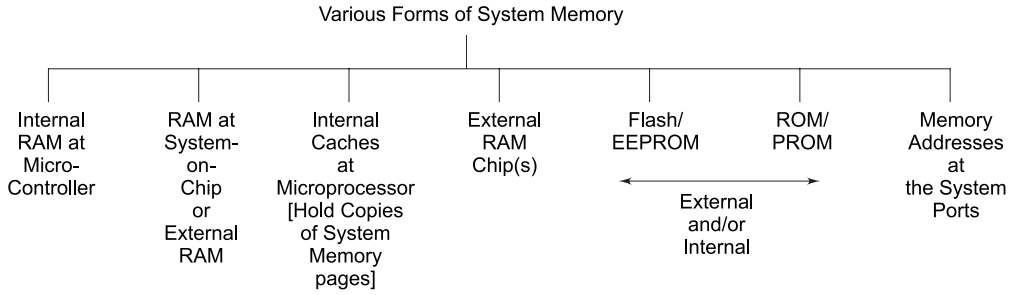

Figure 1.4

The various forms of memories in the system.

Table 1.4 gives the functions assigned in the embedded systems to the memories. ROM or PROM or EPROM embeds the embedded software specific to the system.

Table 1.4
Functions Assigned to the Memories in a System

Memory Needed	Functions
ROM or EPROM	Storing Application programs from where the processor fetches the instruction codes. Storing codes for system booting, initializing, Initial input data and Strings. Codes for RTOS. Pointers (addresses) of various service routines.
RAM (Internal and External) and RAM for buffer	Storing the variables during program run and storing the stack. Storing input or output buffers, for example, for speech or image.
EEPROM or Flash	Storing non-volatile results of processing.
Caches	Storing copies of instructions and data in advance from external memories and storing temporarily the results during fast processing.

A system embeds (locates) the following either in the microcontroller's internal ROM, PROM or in an external ROM or PROM: boot-up programs, initialization data, strings for an initial screen-display or initial state of the system, the programs for various tasks, ISRs and kernel. The system has RAMs for saving temporary data, stack and buffers that are needed during a program run. The system also has flash for storing non-volatile results.

1.3.6 Input, Output and I/O Ports, IO Buses and IO Interfaces

The system gets inputs from physical devices (such as, for example, the key-buttons, sensors and transducer circuits) through the input ports. A controller circuit in a system gets the inputs from the sensor and transducer circuits. A receiver of signals or a network card gets the input from a communication system. [A communication system could be a fax or modem, a broadcasting service]. Signals from a network are also received at the ports. Consider the system in an Automatic Chocolate Vending Machine. It gets inputs from a port that collects the coins that a child inserts. Presume that only a child would use this wonder machine! Consider the system in a mobile phone. The user inputs the mobile number through the buttons, directly or indirectly (through recall of the number from its memory). A panel of buttons connects to the system through the input port or ports. Processor identifies each input port by its memory buffer address(es), called port address(es). Just as a memory location holding a byte or word is identified by an address, each input port is also identified by the address. ***The system gets the inputs by the read operations at the port addresses***.

The system has output ports through which it sends output bytes to the real world. An output may be to an LED (Light Emitting diode) or LCD (Liquid Crystal Display) panel. For example, a calculator or mobile phone system sends the output-numbers or an SMS message to the LCD display. A system may send output to a printer. An output may be to a communication system or network. A control system sends the outputs to alarms, actuators, furnaces or boilers. A robot is sent output for its various motors. Each output port is identified by its memory-buffer address(es) (called port address). The ***system sends the output by a write operation to the port address.***

There are also general-purpose ports for both the input and output (I/O) operations. For example, a mobile phone system sends output as well as gets input through a wireless communication channel. Each I/O port is also identified by an address to which the *read* and *write* operations both take place.

Refer to Section 3.1 for the details regarding ports. There are ***two types of I/O ports: Parallel and Serial***. From a serial port, a system gets a serial stream of bits at an input or sends a stream at an output. For example, through a serial port, the system gets and sends the signals as the bits through a modem. A serial port also facilitates long distance communication and interconnections. A serial port may be a ***Serial UART port,*** a ***Serial Synchronous port*** or some other ***Serial Interfacing port***. [UART stands for Universal Asynchronous Receiver and Transmitter].

A system port may get inputs from multiple channels or may have to send output to multiple channels. A ***demultiplexer*** takes the inputs from various channels and transfers the input from a select channel to the system. A ***multiplexer*** takes the output from the system and sends it to another system.

A system might have to be connected to a number of other devices and systems. For networking the systems, there are different types of buses: for example, I^2C, CAN, USB, ISA, EISA and PCI. [Refer to Sections 3.3, 3.4 and Appendix F for buses in detail.]

A system connects to external physical devices and systems through parallel or serial I/O ports. Demultiplexers and multiplexers facilitate communication of signals from multiple channels through a common path. A system often networks to the other devices and systems through an I/O bus: for example, I^2C, CAN, USB, ISA, EISA and PCI bus.

1.3.7 Interrupts Handler

A system may possess a number of devices and the system processor has to control and handle the requirements of each device by running an appropriate Interrupt Service Routine (ISR) for each. ***An interrupts-handling mechanism must exist in each system to handle interrupts from various processes in the system:*** for example, to transfer data from a keyboard or a printer. [Refer to Chapter 4 for a detailed description of the interrupts and their control (handling) mechanism in a system]. Important points regarding the interrupts and their handling by programming are as follows:

1. There can be a number of interrupt sources and groups of interrupt sources in a processor. [Refer to Section 4.5.] An interrupt may be a hardware signal that indicates the occurrence of an event. [For example, a real-time clock continuously updates a value at a specified memory address; the transition of that value is an event that causes an interrupt.] An interrupt may also occur through timers, through an interrupting instruction of the processor program or through an error during processing. The error may arise due to an illegal op-code fetch, a division by zero result or an overflow or underflow during an ALU operation. An interrupt can also arise through a software timer. A software interrupt may arise in an exceptional condition that may have developed while running a program.
2. The system may prioritize the sources and service them accordingly [Section 4.6.5].
3. Certain sources are not maskable and cannot be disabled. Some are defined to the highest priority during processing.
4. The processor's current program has to divert to a service routine to complete that task on the occurrence of the interrupt. For example, if a key is pressed, then an ISR reads the key and stores the key value in the processor memory address. If a sequence of keys is pressed, for instance in a mobile phone, then an ISR reads the keys and also calls a task to dial the mobile number.
5. There is a programmable unit on-chip for the interrupt handling mechanism in a microcontroller.
6. The application program or scheduler is expected to schedule and control the running of routines for the interrupts in a particular application.

The scheduler always gives priority to the ISRs over the tasks of an application.

A system must have an interrupt handling mechanism for executing the ISRs in case of the interrupts from physical devices, systems and software exceptions.

1.3.8 DAC (Using a PWM) and ADC

Suppose a system needs to give an analog output of a control circuit for automation. The analog output may be to a power system for a d.c. motor or furnace. A *Pulse Width Modulator* (PWM) unit in the microcontroller operates as follows: Pulse width is made proportional to the analog-output needed. PWM inputs are from 00000000 to 11111111 for an 8-bit DAC operation. The PWM unit outputs to an external integrator and then provides the desired analog output.

Suppose an integrator circuit (external to the microcontroller) gives an output of 1.024 Volt when the pulse width is 50% of the total pulse time period, and 2.047V when the width is 100%. When the width is made 25% by reducing by half the value in PWM output control-register, the integrator output will become 0.512 Volt.

Now assume that the integrator operates with a dual (plus-minus) supply. Also assume that when an integrator circuit gives an output of 1.023 Volt, the pulse width is 100% of total pulse time period and –1.024 Volt when the width is 0%. When the width is made 25% by reducing by half the value in an output control register, the integrator output will be 0.512 Volt; at 50% the output will be 0.0 Volt.

From this information, finding the formulae to obtain converted bits for a given PWM register bits ranging from 00000000 to 11111111 in both the situations is left as an exercise for the reader.

The ADC in the system microcontroller can be used in many applications such as Data Acquisition System (DAS), analog control system and voice digitizing system. Suppose a system needs to read an analog input from a sensor or transducer circuit. If converted to bits by the ADC unit in the system, then these bits, after processing, can also give an output. This provides a control for automation by a combined use of ADC and DAC features.

The converted bits can be given to the port meant for digital display. The bits may be transferred to a memory address, a serial port or a parallel port.

A processor may process the converted bits and generate a Pulse Code Modulated (PCM) output. PCM signals are used digitizing the voice in the digital format].

Important points about the ADC are as follows:

1. Either a single or dual analog reference voltage source is required in the ADC. It sets either the analog input's upper limit only or the lower and upper limits both. For a single reference source, the lower limit is set to 0V (ground potential). When the analog input equals the lower limit the ADC generates all bits as 0s, and when it equals the upper limit it generates all bits as 1s. [As an example, suppose in an ADC the upper limit or reference voltage is set as 2.255 Volt. Let the lower limit reference Voltage be 0.255V. Difference in the limits is 2 Volt. Therefore, the resolution will be (2/256) Volt. If the 8-bit ADC analog-input is 0.255V, the converted 8 bits will be 00000000. When the input is (0.255V + 1.000V) = 1.255V, the bits will be 10000000. When the analog input is (0.255V + 0.50V), the converted bits will be 01000000. [From this information, finding a formula to obtain converted bits for a given analog input = v Volt is left as an exercise for the reader].
2. An ADC may be of eight, ten, twelve or sixteen bits depending upon the resolution needed for conversion.
3. The start of the conversion signal (STC) signal or input initiates the conversion to 8 bits. In a system, an instruction or a timer signals the STC.
4. There is an end of conversion (EOC) signal. In a system, a flag in a register is set to indicate the end of conversion and generate an interrupt.
5. There is a conversion time limit in which the conversion is definite.
6. A Sample and Hold (S/H) unit is used to sample the input for a fixed time and hold till conversion is over.

An ADC unit in the embedded system microcontroller may have multi-channels. It can then take the inputs in succession from the various pins interconnected to different analog sources.

For automatic control and signal processing applications, a system must provide necessary interfacing circuit and software for the Digital to Analog Conversion (DAC) unit and Analog to Digital Conversion (ADC) unit. A DAC operation is done with the help of a combination of PWM unit in the microcontroller and an external integrator chip. ADC operations are needed in systems for voice processing, instrumentation, data acquisition systems and automatic control.

1.3.9 LCD and LED Displays

A system requires an interfacing circuit and software to display the status or message for a line, for multi-line displays, or flashing displays. An LCD screen may show up a multi-line display of characters or also show a small graph or icon (called pictogram). A recent innovation in the mobile phone system turns the screen blue to indicate an incoming call. Third generation system phones have both image and graphic displays. An LCD needs little power. It is powered by a supply or battery (a solar panel in the calculator). LCD is a diode that absorbs or emits light on application of 3 V to 4 V and 50 or 60 Hz voltage-pulses with currents less than ~50 mA. The pulses are applied with the same polarity on crystal front and back plane for no light, or with opposite polarity for light. Here polarity at an instance means logic '1' or '0']. An LSI (Lower Scale Integrated Circuit) display-controller is often used in the case of matrix displays.

For indicating ON status of the system there may be an LED, which glows when it is ON. A flashing LED may indicate that a specific task is under completion or is running. It may indicate a wait status for a message. The LED is a diode that emits yellow, green, red (or infrared light in a remote controller), on application of a forward voltage of 1.6 to 2 V. An LED needs current up to 12 mA above 5 mA (less in flashing display mode) and is much brighter than the LCD. Therefore, for flashing display and for display limited to few digits, LEDs are used in a system.

For displaying and messaging, the LCD matrix displays and LED arrays are used in a system. The system must provide necessary interfacing circuit and software for the output to LCD display controller and LED interfacing ports.

1.3.10 Keypad /Keyboard

The keypad or keyboard is an important device for getting user inputs. The system must provide the necessary interfacing and key-debouncing circuit as well as the software for the system to receive input from a set of keys or from a keyboard or keypad. A keypad has up to a maximum of 32 keys. A keyboard may have 104 or more keys. The keypad or keyboard may interface serially or as parallel to the processor directly through a parallel or serial port or through a controller.

For inputs, a keypad or board may interface to a system. The system must provide necessary interfacing circuit and software to receive inputs directly from the keys or through a controller.

1.3.11 Pulse Dialer, Modem and Transceiver

For user connectivity through the telephone line, wireless or a network, a system provides the necessary interfacing circuit. It also provides the software for pulse dialing through the telephone line, for modem interconnection for fax, for Internet packets routing, and for transmitting and connecting a WAG (Wireless Gateway) or cellular system. *A transceiver is a circuit that can transmit as well as receive byte streams.*

In communication systems, the Pulse Dialer, Modem or Transceiver are used. A system must then provide the necessary interfacing circuit and software for dialing, for the modem and transceiver, directly or through a controller.

1.3.12 GPIB (IEEE 488) Link

A system may need linking to another instrument or system. The IEEE 488 GPIB (General Purpose Interface Bus) link is a standard bus originally developed by HP [Hewlett Packard] that links the measuring and instrumentation systems. The embedded system used in the instrumentation systems uses this interfacing standard.

1.3.13 Linking and Interfacing Buses and Units of the Embedded System Hardware

The buses and units in the embedded system hardware need to be linked and interfaced. One way to do this is to incorporate a **glue logic circuit.** [Instead of using individual gates, buffers and decoders, we use the glue logic circuit.] A glue circuit is a circuit that is placed (glued) for all the bus logic actions between circuits and between all chips and main chips (processors and memories). The glue logic circuit of an embedded system may be a circuit for interconnecting the processor to external memories so that the appropriate chip-select signals, according to the system memory, map each of the memory chips [Section 2.5]. The glue logic circuit also includes a circuit to interconnect the parallel and serial ports to the peripherals. [Refer to Chapter 3 for more information on ports.] The glue circuit simplifies the overall embedded system circuit greatly. An example of the use of the glue circuit is to connect the processor, memories and the ports interfacing the LCD display matrix and the keypad.

Programming and configuring one of the followings gives a glue circuit. (i) PAL (Programmable Array Logic). (ii) GAL (Generic Array Logic). (iii) PLD (Programmable Logic Device). (iv) CPLD (Combined PLD). (v) FPGA. These devices are configurable and programmable by a system called ***device programmer***.

PAL has the AND–OR logic arrays. PAL implements only the combinational logic circuit. GAL is another array logic, an advanced version of PAL, which provides sequential circuit implementation. It thus provides the latches, counters and register circuits also. PLD is another logic device that is programmable. A ROM is also a PLD. CPLD is a combination circuit integrated with a PLD. It is a logic device for implementing mixed functions, analog and digital. A CPLD also helps in the control functions and designing a PLC (Programmable Logic Controller). FPGA has a macro cell, which is a combination of gates and flip-flops. An array has number of macro cells. The links within the array or in between macro cells are fusable by a device programmer in these devices.

A glue-circuit designed by configuring and programming PAL, GAL or CPLD or FPGA links provides the interfaces for the buses as well as for all the units of the embedded system hardware in a single chip.

1.3.14 Hardware Units Required in Exemplary Cases

Table 1.5 lists the hardware units that must be present in the embedded systems. Six examples have been chosen to represent systems of varying sophistication. These are:

- ∑ Automatic chocolate vending machine (Case study for its programming has been given in Section 11.1),
- ∑ Data acquisition System,
- ∑ Robot,
- ∑ Mobile Phone,
- ∑ Adaptive Cruise Control (ACC) system with car string stability (Case study for its programming has been given in Section 11.3) and
- ∑ Voice-Processor and Storage System (input, compression, store, decompression, recording and replay).

Remember, RTCs, Timers, Idle-mode, Power-down mode, Watchdog timer and Serial IO Port, UART port and glue-logic circuit are needed practically in all the applications and have, therefore, not listed in the Table. The values given here refer to a typical system only.

Table 1.5

Hardware Required in Six Exemplary Embedded Systems with Typical Values

Hardware Required	Automatic Chocolate Vending Machine[&]	Data Acquisition System	Robot	Mobile Phone	Adaptive Cruise Control System with String Stability[#]	Voice Processor
Processor	Micro-controller	Micro-controller	Micro-controller	Multi-processor System on a Chip	Micro-processor	Micro-processor +DSP
Processor Internal Bus Width in Bits	8	8	8	32	32	32
CISC or RISC Processor Architecture	CISC	CISC	CISC	RISC	RISC	RISC
Caches and MMU	No	No	No	Yes	No	Yes
PROM or ROM Memory	4 kB	8 kB	8 kB	1 MB	64 kB	1 MB
EEPROM + Flash	No	512 B	256 B	32 kB	4 kB	4 MB [+]
RAM Interrupts Handler	256 B On-chip	256B On-chip	256B On-chip	1 MB[+] On SoC	4 kB Off-chip	1 MB[@] Off-chip
Input–output Ports	Multiple ports- Input for coin sorter port, delivery port and display port	Multiple Ports for sensors and Actuators	Multiple Ports for Motors and for angle encoders	Keypad and Display Ports	Switch Buttons and Display Ports	Input Port for Speech and output port for replay

Hardware Required	Automatic Chocolate Vending Machine &	Data Acquisition System	Robot	Mobile Phone	Adaptive Cruise Control System with String Stability#	Voice Processor
Transceiver	No	No	No	Yes for connection to cell service	Yes for tracking radar	No
GPIB Interface	No	Yes	No	No	No	No
Real time detection of an event or signal (Capture and Compare time on an event)	No	Yes	No	Yes	Yes	Yes
Pulse Width Modulation for DAC	No	Yes	Yes	Yes	Yes	Yes
Analog to Digital conversion (bits)	No	Yes	Yes	Yes	Yes	Yes
Modulation Demodulation	No	No	No	Yes	No	No
Digital Signal Processing Instructions	No	No	No	Yes	No	Yes
Non linear controller Instructions	No	No	No	No	Yes	No

& Refer to case study in Section 11.1. # Refer to case study 11.2. String stability means maintaining a constant distance between the cars. A radar and transceiver pair is used to measure in-front car distance. EEPROM in needed to store adaptive algorithm parameters.
@ Excessive need of RAM is due to buffer memory RAM for voice inputs being processed.
+ Excessive need of RAM is due to buffer memory RAM for voice inputs and outputs being processed
&Buffer-memory for the speech signals
+ For storing the voice

Embedded systems for a wide spectrum of applications may need different processing hardware platform. However, it is also true that by changing the embedded software, the same hardware platform can be used for entirely different applications or for new upgrades of the same system.

1.4 SOFTWARE EMBEDDED INTO A SYSTEM

The software is the most important aspect, the brain of the embedded system.

1.4.1 Final Machine Implementable Software for a Product

An embedded system processor and the system need software that is specific to a given application of that system. The processor of the system processes the instruction codes and data. In the final stage, these are placed in the memory (ROM) for all the tasks that have to be executed. The final stage software is also called ROM image. Why? Just as an image is a unique sequence and arrangement of pixels, embedded software is also a unique placement and arrangement of bytes for instructions and data.

Each code or datum is available only in bits and byte(s) format. The system requires bytes at each ROM-address, according to the tasks being executed. A *machine implement-able software file is therefore like a table of address and bytes at each address of the system memory.* The table has to be readied as a ROM image for the targeted hardware. Figure 1.5 shows the ROM image in a system memory. The image consists of the boot up program, stack (s) address pointer(s), program counter address pointer(s), application tasks, ISRs (Section 4.12), RTOS, input data, and vector addresses. [Refer to Section 2.5 for the details.]

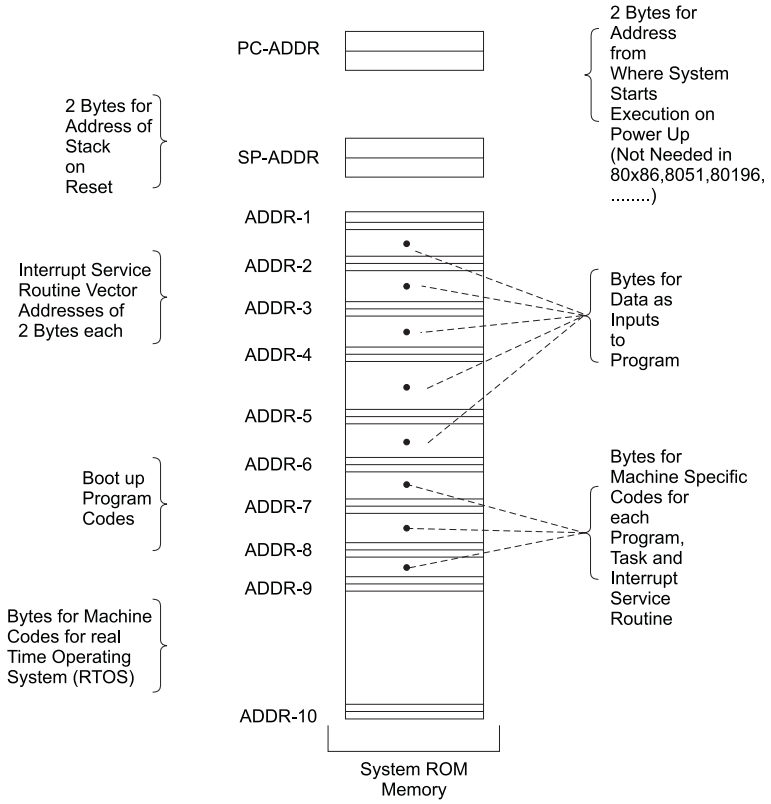

Figure 1.5

System ROM memory embedding the software, RTOS, data, and vector addresses.

Final stage software is also called the ROM image. The final machine implement-able software for a product embeds in the ROM (or PROM) as an image at a frame. Bytes at each address must be defined for creating the ROM image. By changing this image, the same hardware platform will work differently and can be used for entirely different applications or for new upgrades of the same system.

1.4.2 Coding of Software in Machine Codes

During coding in this format, the programmer defines the addresses and the corresponding bytes or bits at each address. In configuring some specific physical device or subsystem, machine code-based coding is used. For example, in a transceiver, placing certain machine code and bits can configure it to transmit at specific Mbps or Gbps, using a specific bus protocol and networking protocol. Another example is using certain codes for configuring a control register with the processor. During a specific code-section processing, the register can be configured to enable or disable use of its internal cache. However, coding in ***machine implement-able codes*** is done only in specific situations: ***it is time consuming because the programmer must first understand*** the processor instructions set and then memorize the instructions and their machine codes.

1.4.3 Software in Processor Specific Assembly Language

When a programmer understands the processor and its instruction set thoroughly, a program or a small specific part can be coded in the ***assembly language***. An exemplary assembly language program in ARM processor instruction set will be shown in Example given in Section A.2.

Coding in assembly language is easy to learn for a designer who has gone through a microprocessor or microcontroller course. Coding is extremely useful for configuring physical devices like ports, a line-display interface, ADC and DAC and reading into or transmitting from a buffer. These codes can also be device driver codes. [Section 4.1]. They are useful to run the processor or device specific features and provide an optimal coding solution. *Lack of knowledge of writing device driver codes or codes that utilize the processor-specific features-invoking codes in an embedded system design team can cost a lot. Vendors may not only charge for the API but also charge intellectual property fees for each system shipped out of the company*.

To do all the coding in *assembly language* may, however, be very time consuming. Full coding in assembly may be done only for a few simple, small-scale systems, such as toys, automatic chocolate vending machine, robot or data acquisition system.

Figure 1.6 shows the process of converting an *assembly language program* into the machine implement-able software file and then finally obtaining a ROM image file.

1. An ***assembler*** translates the assembly software into the machine codes using a step called *assembling*.
2. In the next step, called *linking*, a ***linker*** links these codes with the other required assembled codes. Linking is necessary because of the number of codes to be linked for the final binary file. For example, there are the standard codes to program a delay task for which there is a reference in the assembly language program. The codes for the delay must link with the assembled codes. The delay code is sequential from a certain beginning address. The assembly software code is also sequential from a certain beginning address. Both the codes have to at the distinct addresses as well as available addresses in the system. Linker links these. The linked file in binary for *run*

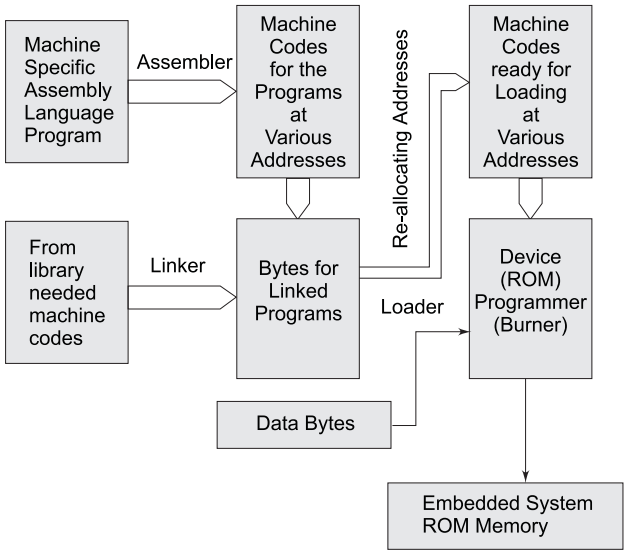

Figure 1.6

The process of converting an assembly language program into the machine codes and finally obtaining the ROM image

on a computer is commonly known as executable file or simply '.exe' file. After linking, there has to be re-allocation of the sequences of placing the codes before actually placement of the codes in the memory.

3. In the next step, the ***loader*** program performs the task of *reallocating* the codes after finding the physical RAM addresses available at a given instant. The loader is a part of the operating system and places codes into the memory after reading the '.exe' file. This step is necessary because the available memory addresses may not start from 0x0000, and binary codes have to be loaded at the different addresses during the run. The loader finds the appropriate start address. In a computer, the loader is used and it loads into a section of RAM the program that is ready to run.
4. The final step of the system design process is *locating* the codes as a ROM image and permanently placing them at the actually available addresses in the ROM. In embedded systems, there is no separate program to keep track of the available addresses at different times during the running, as in a computer. The designer has to define the available addresses to load and create files for permanently locating the codes. A program called ***locator*** reallocates the linked file and creates a file for permanent location of codes in a standard format. This format may be Intel Hex file format or Motorola S-record format. [Refer to Appendix G for details.]

The locator locates the I/O tasks and hardware device driver codes at the unchanged addresses. This is because the port addresses for these are fixed for a given system.

5. Lastly, either (i) a laboratory system, called ***device programmer,*** takes as input the ROM image file and finally *burns* the image into the PROM or EPROM or (ii) at a foundry, a mask is created for the ROM of the embedded system from the image file**.** [The process of placing the codes in

PROM or EPROM is also called burning.] **The** mask created from the image gives the ROM in IC chip form.

For configuring some specific physical device or subsystem like transceiver, the machine codes can be straightaway used. For physical device driver codes or codes that utilize the processor-specific features-invoking codes, 'processor-specific' assembly language is used. A file is then created in three steps using 'Assembler', 'Linker' and 'Locator'. The file has the ROM image in a standard format. A device programmer finally burns the image in PROM or EPROM. A mask created from the image gives the ROM in IC chip form.

1.4.4 Software in High Level Language

To do all the coding in *assembly language* may be very time consuming in most cases. Software is therefore developed in a high-level language, 'C' or 'C++' or 'Java'. Most of the times, 'C' is the preferred language. [Refer to Sections 5.1, 5.8 and 5.9 to understand the advantages available in each and to Section 5.11 for the use of 'C' source-code programming tools.] For coding, there is little need to understand assembly language instructions and the programmer does not have to know the machine code for any instruction at all. The programmer needs to understand only the hardware organization. As an example, consider the following problem:

Add 127, 29 and 40 and print the square root.

An exemplary C language program for all the processors is as follows: (i) *# include <stdio.h>* (ii) *# include <math.h>* (iii) *void main (void) {* (iv) *int i1, i2, i3, a;* float result; (v) i1 = 127; i2 = 29; i3 = 40; *a = i1 + i2 + i3; result = sqrt (a);* (vi) *printf (result);}*

It is evident, then, that coding for square-root will need many lines of code and can be done only by an expert assembly language programmer. To write the program in a high level language is very simple compared to writing it in the assembly language. 'C' programs have a feature that adds the assembly instructions when using certain processor-specific features and coding for the specific section, for example, port device driver. Figure 1.7 shows the different programming layers in a typical embedded

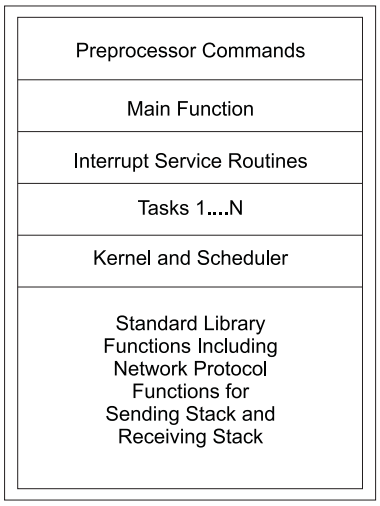

Figure 1.7

The different program layers in the embedded software

'C' software. [Refer to appropriate sections in Chapters 3 to 5.] These layers are as follows. (i) Processor Commands. (ii) Main Function. (iii) Interrupt Service Routine. (iv) Multiple tasks, say, 1 to N. (v) Kernel and Scheduler. (vi) Standard library functions, protocol functions and stack allocation functions.

Figure 1.8 shows the process of converting a C *program* into the ROM image file. A ***compiler*** generates the object codes. The compiler assembles the codes according to the processor instruction set and other specifications. The 'C' compiler for embedded systems must, as a final step of compilation, use a *code-optimizer*. It optimizes the codes before linking. After compilation, the *linker* links the object codes with other needed codes. For example, the linker includes the codes for the functions, *printf* and sqrt codes. Codes for device management and driver (device control codes) also link at this stage: for example, printer device management and driver codes. After linking, the other steps for creating a file for ROM image are the same as shown earlier in Figure 1.6.

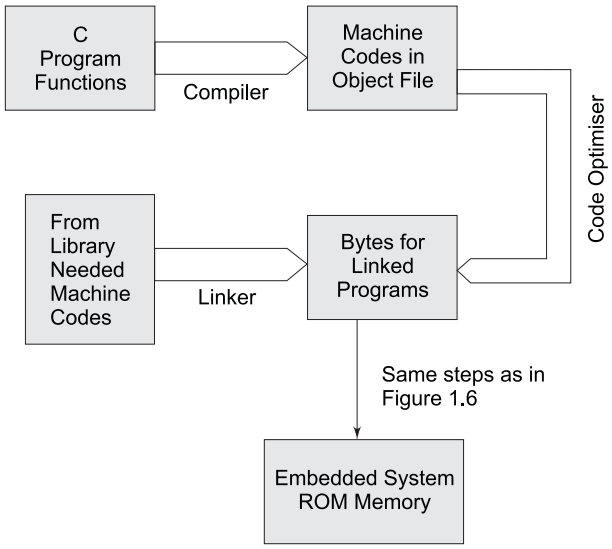

Figure 1.8

The process of converting a C program into the file for ROM image

For most systems, software codes are developed in 'C'. 'C' program has various layers; processor commands, main function, task functions and library functions, interrupt service routines and kernel (scheduler). The compiler generates an object file. Using linker and locator, the file for ROM image is created for the targeted hardware. C++ and Java are other languages used for software coding.

1.4.5 Software for the Device Drivers and Device Management using an Operating System

In an embedded system, there are a number of *physical devices*. Exemplary physical devices are keyboard, display, disk, parallel port and network-card.

*An innovative concept is use of virtual devices during programming. A v*irtual device example is a ***file*** (for reading and writing the stream of bytes or words) *or a* ***pipe*** (for buffering a stream of bytes). The term virtual device follows from the analogy that just as a keyboard gives an input to the processor on a *read*, a file also gives an input to the processor. The processor gives an output to a printer on a *write*. Similarly, the processor writes an output to the file. Most often, an embedded system is designed to perform multiple functions and has to control multiple physical and virtual devices.

A **device** for the purpose of control, handling, reading and writing actions can be taken as consisting of three components. (i) Control Register or Word – It stores the bits that, on setting or resetting by a device driver, control the device actions. (ii) Status Register or Word – It provides the flags (bits) to show the device status. (iii) Device Mechanism that controls the device actions. There may be input data buffers and output data buffers at a device. Device action may be to get input into or send output from the buffer. (*The control registers, input data buffers, output data buffers and status registers form part of the device hardware.*)

A **device driver** is software for controlling, receiving and sending a byte or a stream of bytes from or to a device. In case of physical devices, a driver uses the hardware status flags and control register bits that are in set and reset states. In case of virtual devices also, a driver uses the status and control words and the bits that exist in set and reset states.

Driver controls three functions (i) Initializing that is activated by placing appropriate bits at the control register or word. (ii) Calling an ISR on interrupt or on setting a status flag in the status register and run (drive) the ISR (also called Interrupt Handler Routine). (iii) Resetting the status flag after interrupt service. A driver may be designed for asynchronous operations (multiple times use by tasks one after another) or synchronous operations (concurrent use by the tasks). This is because a device may get activated when an interrupt arises and the device driver routine services that.

Using Operating System (OS) functions, a device driver coding can be made such that the underlying hardware is hidden as much as possible. An API then defines the hardware separately. This makes the driver usable when the device hardware changes in a system.

A device driver accesses a parallel port or serial port, keyboard, mouse, disk, network, display, file, pipe and socket at specific addresses. An OS may also provide *Device driver* codes for the system-port addresses and for the access mechanism (read, save, write) for the device hardware.

Device Management software modules provide codes for detecting the presence of devices, for initializing these and for testing the devices that are present. The modules may also include software for allocating and registering port (in fact, it may be a register or memory) addresses for the various devices at distinctly different addresses, including codes for detecting any collision between these, if any. It ensures that any device accesses to one task at any given instant. It takes into account that virtual devices may have addresses that can be relocated by a *locator* (for PROM). [The actual physical or hardware devices have predefined fixed addresses (the addresses are not relocated by the locator)].

An OS also provides and executes modules for managing devices that associate with an embedded system. The underlying principle is that at an instant, only one physical device should get access to or from one task only. The OS also provides and manages the virtual devices like pipes and sockets [Section 8.3].

For designing embedded-software, two types of devices are considered: physical and virtual. Physical devices include keypads, printers or display matrix. A virtual device could be a file (for reading and writing the stream of bytes or words) or pipe (for buffering the stream of bytes). Device drivers and device management software are needed in the system. The operating system has modules for the device driver and for device management functions.

1.4.6 Software Design for Scheduling Multiple Tasks and Devices Using an RTOS

Most often, an embedded system is designed to perform scheduling of multiple functions while controlling multiple devices. An embedded system program is therefore designed as a multitasking system program. [Refer to Section 8.1 for definitions of the *tasks* (functions) and *task-states*.]

In a multitasking OS, *each process (task) has a distinct memory allocation of its own and a task has one or more functions or procedures for a specific job. A task may share the memory (data) with other tasks. A processor may process multiple tasks separately or concurrently*. The OS software includes scheduling features for the processes (tasks, ISRs and Device Drivers) *An OS or RTOS has a kernel*. [Refer to Section 9.2 for understanding kernel functions in detail.] *The kernel's important function is to schedule the transition of a task from a ready state to a running state*. It also schedules the transition of a task from a blocked state to the running state. The kernel may block a task to let a higher priority task be in running state. [It is called preemptive scheduling]. The *kernel* coordinates the use of the processor for the multiple tasks that are in ready state at any instant, such that only one task among many is in the running state. This is so because there is only one processor in the system. The kernel schedules and dispatches a task to a different state than the present. [For multiprocessor systems, scheduling and synchronization of various processors are also necessary]. The kernel controls the inter process (task) messaging and sharing of variables, queues and pipes.

RTOS functions can thus be highly complex. Chapters 9 to 11 will describe the RTOS functions in an embedded system. In an embedded system, RTOS has to be scalable. *Scaleable OS* is one in which memory is optimized by having only that part of features that are needed associate with the final system software.

There are a number of popular and readily available RTOSs. Chapters 9 and 10 will describe these. Case studies employing these RTOSs will be taken up in Chapter 11.

Embedded software is most often designed for performing multiple actions and controlling multiple devices and their ISRs. Multitasking software is therefore essential. For scheduling multiple tasks, popular, readily available RTOS kernel with functions (like device-drivers) are most often used.

1.4.7 Software Tools in Designing of an Embedded System

Table 1.6 lists the applications of software tools for assembly language programming, high level language programming, RTOS, debugging and system integration tools.

Table 1.6
Software Modules and Tools for the Detailed Designing of an Embedded System

Software Tool	Application
Editor	For writing C codes or assembly mnemonics using the keyboard of the PC for entering the program. Allows the entry, addition, deletion, insert, appending previously written lines or files, merging record and files at the specific positions. Creates a source file that stores the edited file. It also has an appropriate name [provided by the programmer].
Interpreter	For expression-by-expression (line-by-line) translation to the machine executable codes.
Compiler	Uses the complete sets of the codes. It may also include the codes, functions and expressions from the library routines. It creates a file called object file.
Assembler	For translating the assembly mnemonics into binary opcodes (instructions), i.e., into an executable file called a binary file. It also creates a list file that can be printed. The list file has address, source code (assembly language mnemonic) and hexadecimal object codes. The file has addresses that adjust during the actual run of the assembly language program.
Cross Assembler	For converting object codes or executable codes for a processor to other codes for another processor and vice versa. The cross-assembler assembles the assembly codes of target processor as the assembly codes of the processor of the PC used in the system development. Later, it provides the object codes for the target processor. These codes will be the ones actually needed in the finally developed system.
Simulator	To simulate all functions of an embedded system circuit including additional memory and peripherals. It is independent of a particular target system. It also simulates the processes that will execute when the codes execute on the targeted particular processor.
Source-code Engineering Software	For source code comprehension, navigation and browsing, editing, debugging, configuring (disabling and enabling the C++ features) and compiling.
RTOS	Refer Chapters 9 and 10.
Stethoscope	For dynamically tracking the changes in any program variable. It tracks the changes in any parameter. It demonstrates the sequences of multiple processes (tasks, threads, service routines) that execute. It also records the entire time history.
Trace Scope	To help in tracing the changes in the modules and tasks with time on the X-axis. A list of actions also produces the desired time scales and the expected times for different tasks.
Integrated Development Environment	Software and hardware environment that consists of simulators with editors, compilers, assemblers, RTOS, debuggers, stethoscope, tracer, emulators, logic analyzers, EPROM EEPROM application codes' burners for the integrated development of a system.
Prototyper~	For simulating, source code engineering including compiling, debugging and, on a Browser, summarizing the complete status of the final target system during the development phase.
Locator#	Uses cross-assembler output and a memory allocation map and provides the locator program output. It is the final step of software design process for the embedded system.

For locator refer to Section 2.5. Locator program output is in the Intel hex file or Motorola S- record format.
~ An Example is Tornado Prototyper from WindRiver, for integrated cross-development environment with a set of tools.

The assembler codes in assembly language. For high-level language programming, a special source code engineering tool may be needed while designing sophisticated systems. RTOS is necessary in most embedded systems, as a system, in most cases, has to schedule multiple tasks to meet their deadline, drive a number of physical and virtual devices and handle many ISRs. Debugging tools like stethoscope and trace scope are needed for debugging. It is an important step in the system development. A sophisticated tool—Integrated Development Environment or Prototype development tool—is needed for integrated development of system software and hardware.

1.4.8 Needed Software Tools in the Exemplary Cases

Table 1.7 gives the various tools needed to design exemplary systems.

Table 1.7
Software Tools Required in Exemplary Systems

Software Tools	Automatic Chocolate Vending Machine[&]	Data Acquisition System	Robot	Mobile Phone	Adaptive Cruise Control System with String Stability[#]	Voice Processor
Editor	Yes	Yes	Yes	NR	NR	NR
Interpreter	Yes	NR	Yes	NR	NR	NR
Compiler	NR	Yes	No	Yes	Yes	Yes
Assembler	Yes	Yes	Yes	No	No	No
Cross Assembler	NR	Yes	No	No	No	No
Locator[#]	Yes	Yes	Yes	Yes	Yes	Yes
Simulator	NR	Yes	Yes	Yes	Yes	Yes
Source-code Engineering Software	NR	NR	NR	Yes	Yes	Yes
RTOS	MR	MR	MR	Yes	Yes	Yes
Stethoscope	NR	NR	NR	Yes	Yes	Yes
Trace Scope	NR	NR	NR	Yes	Yes	Yes
Integrated Development Environment	NR	Yes	Yes	Yes	Yes	Yes
Prototyper[~]	NR	No	No	Yes	Yes	Yes

[~] An Example is Tornado prototyper WindRiverâ for integrated cross-development environment with a set of tools.
Note: NR means not required. MR means may be required in specific complex system but not compulsorily needed.
For locator, refer to Section 2.5.

RTOS is needed in most embedded systems, as a system may have to schedule multiple tasks, drive a number of physical and virtual devices and handle many ISRs. Embedded systems for medium scale and sophisticated applications may need a number of sophisticated software tools.

1.4.9 Models for Software Designing

In complex or multiprocessor systems, there are different models that are employed during the design processes of the embedded software and its RTOS. The models are as following. (i) Finite State Machine (FSM). (ii) Petrinet model. (iii) Control and Data flow graph. (iv) Activity diagrams based UML Model. For multiprocessor systems, the following additional models are needed. (i) Synchronous Data Flow (SDF) Graph. (ii) Timed Petri Nets and Extended Predicate/Transition Net. (iii) Multi Thread Graph (MTG) System. These models are explained in Chapter 6.

1.5 EXEMPLARY EMBEDDED SYSTEMS

1.5.1 Exemplary Applications of Each Type of Embedded System

Embedded systems have very diversified applications. A few select application areas of embedded systems are Telecom, Smart Cards, Missiles and Satellites, Computer Networking, Digital Consumer Electronics, and Automotive. Figure 1.9 shows the applications of embedded systems in these areas.

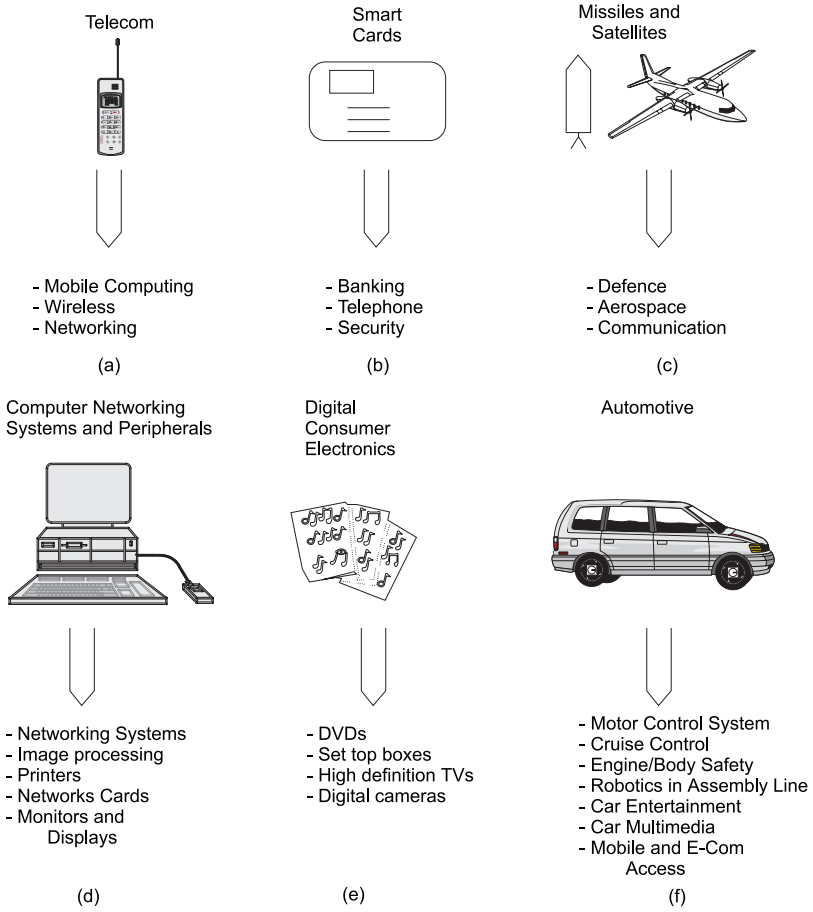

Figure 1.9

Applications of the three types of embedded systems in various areas

A few examples of *small scale embedded system* applications are as follows:

- ∑ Automatic Chocolate Vending Machine
- ∑ Stepper motor controllers for a robotics system
- ∑ Washing or cooking system
- ∑ Multitasking Toys
- ∑ Microcontroller-based single or multi-display digital panel meter for voltage, current, resistance and frequency
- ∑ Keyboard controller
- ∑ Serial port cards
- ∑ Computer Mouse
- ∑ CD drive or Hard Disk drive controller
- ∑ The peripheral controllers of a computer, for example, a CRT display controller, a keyboard controller, a DRAM controller, a DMA controller, a printer-controller, a laser printer-controller, a LAN controller, a disk drive controller
- ∑ Fax or photocopy or printer or scanner machine
- ∑ Digital diary
- ∑ Remote (controller) of TV
- ∑ Telephone with memory, display and other sophisticated features
- ∑ Motor controls Systems - for example, an accurate control of speed and position of d.c. motor, robot, and CNC machine; automotive applications such as a close loop engine control, a dynamic ride control, and an anti-lock braking system monitor
- ∑ Electronic data acquisition and supervisory control system
- ∑ Electronic instruments, such as an industrial process controller
- ∑ Electronic smart weight display system and an industrial moisture recorder cum controller
- ∑ Digital storage system for a signal wave form or Electric or Water Meter Readings
- ∑ Spectrum analyzer
- ∑ Biomedical systems such as an ECG LCD display-cum-recorder, a blood- cell recorder cum analyzer, and a patient monitor system

Some examples of *medium scale embedded systems* are as follows:

- ∑ Computer networking systems, for example, a router, a front-end processor in a server, a switch, a bridge, a hub and a gateway
- ∑ For Internet appliances, there are numerous application systems (i) An intelligent operation, administration and maintenance router (IOAMR) in a distributed network and (ii) Mail Client card to store e-mail and personal addresses and to smartly connect to a modem or server
- ∑ Entertainment systems – such as a video game and a music system
- ∑ Banking systems for example, Bank ATM and Credit card transactions
- ∑ Signal Tracking Systems for example, an automatic signal tracker and a target tracker
- ∑ Communication systemssuch as a mobile-communication SIM card, a numeric pager, a cellular phone, a cable TV terminal, and a FAX transceiver with or without a graphic accelerator
- ∑ Image Filtering, Image Processing, Pattern Recognizer, Speech Processing and Video Processing
- ∑ A system that connects a pocket PC or PDA (Personal Digital Assistant) to the automobile driver mobile phone and a wireless receiver. The system then connects to a remote server for Internet or e-mail or to a remote computer at an ASP (application Service Provider). This system forms the backbone of m-commerce (mobile e-commerce) and mobile computing.

- ∑ A personal information manager using frame buffers in hand-held devices
- ∑ Thin Client [A Thin Client provides disk-less nodes with remote boot capability]. Application of thin-clients accesses to a data center from a number of nodes; in an Internet Laboratory accesses to the Internet leased line through a remote Server.
- ∑ Embedded Firewall/ Router using ARM7/ i386 multi-processor and 32 MB of Flash ROM. The load balancing and two Ethernet interfaces are its other important functions. These interfaces support PPP, TCP/IP and UDP protocols.
- ∑ DNA Sequence and pattern storage card and DNA pattern recognizer

Examples of *sophisticated embedded systems* are as follows:

- ∑ Embedded systems for wireless LAN and for convergent technology devices
- ∑ Embedded systems for real time video and speech or multimedia processing systems
- ∑ Embedded Interface and Networking systems using high speed (400 MHz plus), ultra high speed (10 Gbps) and large bandwidth: Routers, LANs, switches and gateways, SANs (Storage Area Networks), WANs (Wide Area Networks), Video, Interactive video and broadband IPv6 (Internet Protocol version 6) Internet and other products
- ∑ Security products and High-speed Network security. Gigabit rate encryption rate products
- ∑ Embedded sophisticated system for space lifeboat (NASA's X-38 project) under development. It is for a rescue lifeboat that will be used in the future with the ISS (International Space Station). In an emergency, it will bring the astronauts and crewmembers back to the Earth from the ISS. With a press of a button this lifeboat will detach from ISS and travel back to Earth resisting all the climatic/atmospheric conditions and meeting exact timing constraints. This will also be a fault tolerant system.

1.6 EMBEDDED SYSTEM-ON-CHIP (SOC) AND IN VLSI CIRCUIT

Lately, embedded systems are being designed on a single silicon chip, called *System on chip (SoC). SoC is a new design innovation for embedded systems*. An embedded processor is a part of the SoC VLSI circuit. A SoC may be embedded with the following components: multiple processors, memories, multiple standard source solutions, called IP (Intellectual Property) cores and other logic and analog units. A SoC may also have a network protocol embedded into it. It may also embed an encryption function unit. It can embed discrete cosine transforms for signal processing applications. It may embed FPGA (Field Programmable Gate Array) cores [Section 1.6.5].

For a number of applications, the GPP (microcontrollers, microprocessors or DSPs) cores may not suffice. For security applications, killer applications, smart card, video game, palm top computer, cell-phone, mobile-Internet, hand-held embedded systems, Gbps transceivers, Gigabits per second LAN systems and satellite or missile systems, we need special processing units in a VLSI designed circuit to function as a processor. These special units are called Application Specific Instruction Processors (ASIP). For an application, both the configurable processors (called FPGA cum ASIP processors) and non-configurable processors (DSP or Microprocessor or Microcontrollers) might be needed on a chip. One example of a killer application using multiple ASIPs is high-definition television signals processing. [High definition means that the signals are processed for a noise-free, echo-canceled transmission, and for obtaining a flat high-resolution image (1920 x 1020 pixels) on the television screen.] A cell-phone is another killer application. [A killer application is one that is useful to millions of users.]

Recently, embedded SoCs have been designed for functioning as *DNA chips*. Consider an FPGA with a large number of gate arrays. Now, using VLSI design techniques, we can configure these arrays to process the specific tasks on an SoC. This gives an SoC as a DNA chip. Each set of arrays has a specific and distinct DNA complex structure. These structures as well as the processor embeds on the DNA chip.

1.6.1 Exemplary SoC for Cell-Phone

Figure 1.10 shows an SoC that integrates two internal ASICs, two internal processors (ASIPs), shared memories and peripheral interfaces on a common bus. Besides a processor and memories and digital circuits with embedded software for specific application (s), the SoC may possess analog circuits as well]. An exemplary application of such an ASIC embedded SoC is the cell-phone. One ASIP in it is configured to process encoding and deciphering and another does the voice compression. One ASIC dials, modulates, demodulates, interfaces the keyboard and multiple line LCD matrix displays, stores data input and recalls data from memory. ASICs are designed using the VLSI design tools with processor GPP or ASIP and analog circuits embedded into the design. The designing is done using the Electronic Design Automation (EDA) tool. [For design of ASIC digital circuits, a 'High Level Design Language (HDL)' is used].

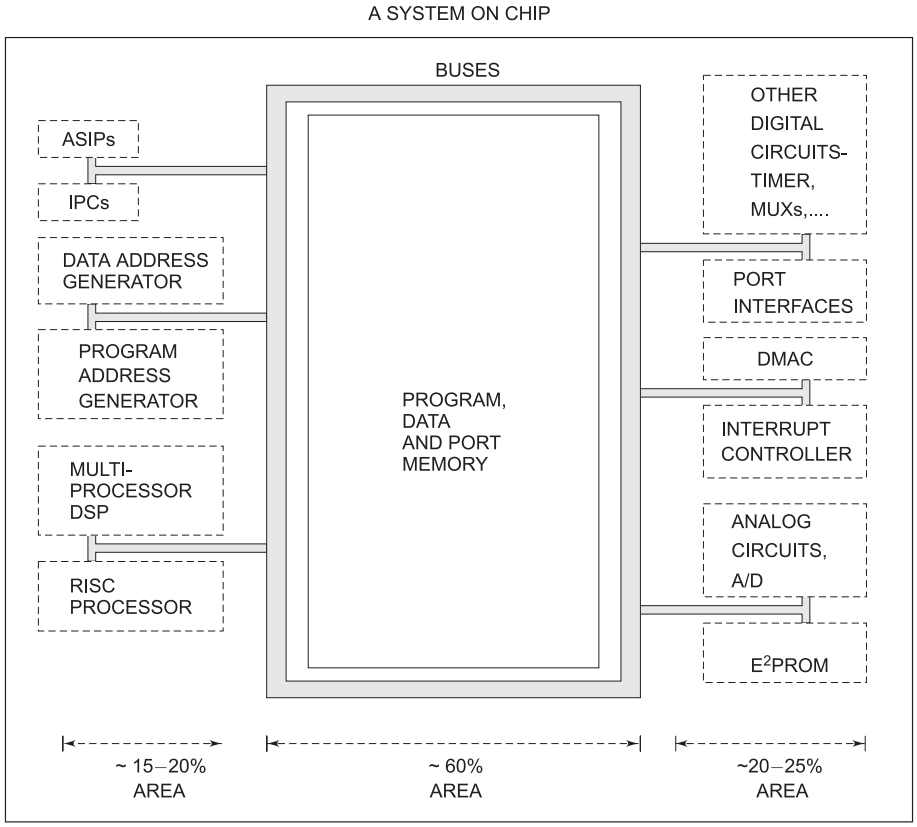

Figure 1.10

A SoC embedded system and its common bus with two internal ASICs, two internal processors, shared memories and peripheral interfaces.

1.6.2 ASIP

Using VLSI tools, a processor itself can be designed. A system specific processor (ASIP) is the one that does not use the GPP (standard available CISC or RISC microprocessor or microcontroller or signal processor). The processor on chip incorporates a section of the CISC or RISC instruction set. This specific processor may have especially configurable instruction-set for an application. An ASIP can also be configurable. Using appropriate tools, an ASIP can be designed and configured for the instructions needed in the following exemplary functions: DSP functions, controller signals processing function, adaptive filtering functions and communication protocol–implementing functions. On a VLSI chip, an embedded ASIP in a special system can be a unit within an ASIC or SoC.

1.6.3 IP Core

On a VLSI chip, there may be high-level components. These are components that possess gate-level sophistication in circuits above that of the counter, register, multiplier, floating point operation unit and ALU. A standard source solution for synthesizing a higher-level component by configuring FPGA core or a core of VLSI chip may be available as an Intellectual Property, called (IP). The copyright for the synthesized design of a higher-level component for gate-level implementation of an IP is held by the designer or designing company. One has to pay royalty for every chip shipped. An embedded system may incorporate an IP(s) .

- ∑ An IP may provide hardwired implement-able design of a *transform* or of an *encryption algorithm* or a *deciphering algorithm*.
- ∑ An IP may provide a design for *adaptive filtering* of a signal.
- ∑ An IP may provide full design for implementing Hyper Text Transfer Protocol (HTTP) or File Transfer Protocol (FTP) to transmit a web page or a file on the Internet.
- ∑ An IP may be designed for the PCI or USB bus controller. [Sections 3.3 and 3.4]

1.6.4 Embedding a GPP

A General Purpose Processor (GPP) can be embedded on a VSLI chip. Recently, exemplary GPPs, called ARM 7 and ARM 9, which embed onto a VLSI chip, have been developed by ARM and their enhancements by Texas Instruments. [Refer to http:/www.ti.com/sc/ docs/asic/modules/arm7.htm and arm9.htm]. An ARM-processor VLSI-architecture is available either as a CPU chip or for integrating it into VLSI or SoC. [The instruction set features of the ARM is given in Section A.1.] ARM provides CISC functionality with RISC architecture at the core. An application of ARM embed circuit is ICE [Section 12.3.2]. ICE is used for debugging an embedded system. Exemplary ARM 9 applications are setup boxes, cable modems, and wireless devices such as mobile handsets.

ARM9 [Section 2.2.5] has a single cycle 16 x 32 multiply accumulate unit. It operates at 200 MHz. It uses 0.15 mm GS30 CMOSs. It has a five-stage pipeline. It incorporates RISC. It integrates with a DSP when designing an ASIC solution having multiprocessors' architecture [Section 6.3]. An example is its integration with DSP with TMS320C55x. A lower capability but very popular version of ARM9 is ARM7. It operates at 80 MHz clock. It uses 0.18 mm based GS20 CMOSs. Using ARM7, a large number of embedded systems have recently become available. One recent application is in integrating the operating system Linux Kernel 2.2 and the device drivers into an ASIC.

1.6.5 FPGA (Field Programmable Gate Arrays] Core with a Single or Multiple Processor

A new innovation is Field Programmable Gate Arrays (FPGA) core with a single or multiple processor units on chip. One example is Xilinx Virtex-II Pro FPGA XC2VP125. Other example is 90 nm Spartan-3 FPGA released on April 14, 2003 by Xilinx. [An FPGA consists of a large number of programmable gates on a VLSI chip. There is a set of gates in each FPGA cell, called 'macro cell'. Each cell has several inputs and outputs. All cells interconnect like an array (matrix). Each interconnection is fusible (detachable) using a FPGA programming tool.] The interested reader can refer to the recent articles in 'Xcell Journal' 2002 and 'Xcell Journal' 2003.

Consider the algorithms for the following: an SIMD instruction, Fourier transform and its inverse, DFT or Laplace transform and its inverse, compression or decompression, encrypting or deciphering, a specific pattern-recognition (for recognizing a signature or finger print or DNA sequence). We can configure (fuse) an algorithm into the logic gates of the FPGA. It gives us hardwired implementation for a processing unit. It is specific to the needs of the embedded system. An algorithm of the embedded software can implement in one of the FPGA section and another algorithm in its other section.

An exemplary latest SoC design is on the *XC2VP125 system. It has 125136 logic cells in the FPGA core with four IBM PowerPCs*. It has been very recently used for developing embedded systems integrated with programmable logic. For example, there is a solution reported for data security with encryption engine and data rate of 1.5 gigabits per second. Other exemplary embedded systems integrated with logic arrays are DSP-enabled, real-time video processing systems and line echo eliminators for the Public Switched Telecommunication Networks (PSTN) and packet switched networks. [A packet is a unit of a message or a flowing data such that it can follow a programmable route among the number of optional open routes available at an instance].

1.6.6 Components in an Exemplary SoC–Smart Card

Figure 1.11 shows embedded-system hardware components on an SoC for a contact-less smart card. Its components are as follows: [Section 11.4 will describe a case study of embedded software design.]

- ASIP (Application Specific Instruction Processor)
- RAM for temporary variables and stack
- ROM for application codes and RTOS codes for scheduling the tasks
- EEPROM for storing user data, user address, user identification codes, card number and expiry date
- Timer and Interrupt controller
- A carrier frequency ~16 MHz generating circuit and Amplitude Shifted Key (ASK) Modulator. ASK modulator gives 10% excess amplitude of carrier pulses for bit '1' and 10% less for bit '0'. A load modulation sub-carrier has one-sixteenth of this frequency and modulates the 1s and 0s by Binary Phase Shifted Keying (BPSK).
- Interfacing circuit for the I/Os
- Charge pump for delivering power to the antenna (of ~5 mm range) for transmission and for system circuits. The charge pump stores charge from received RF (radio frequency) at the card antenna hear its host in vicinity. [The charge pump is a simple circuit that consists of a diode and a high value ferroelectrics material-based capacitor.]

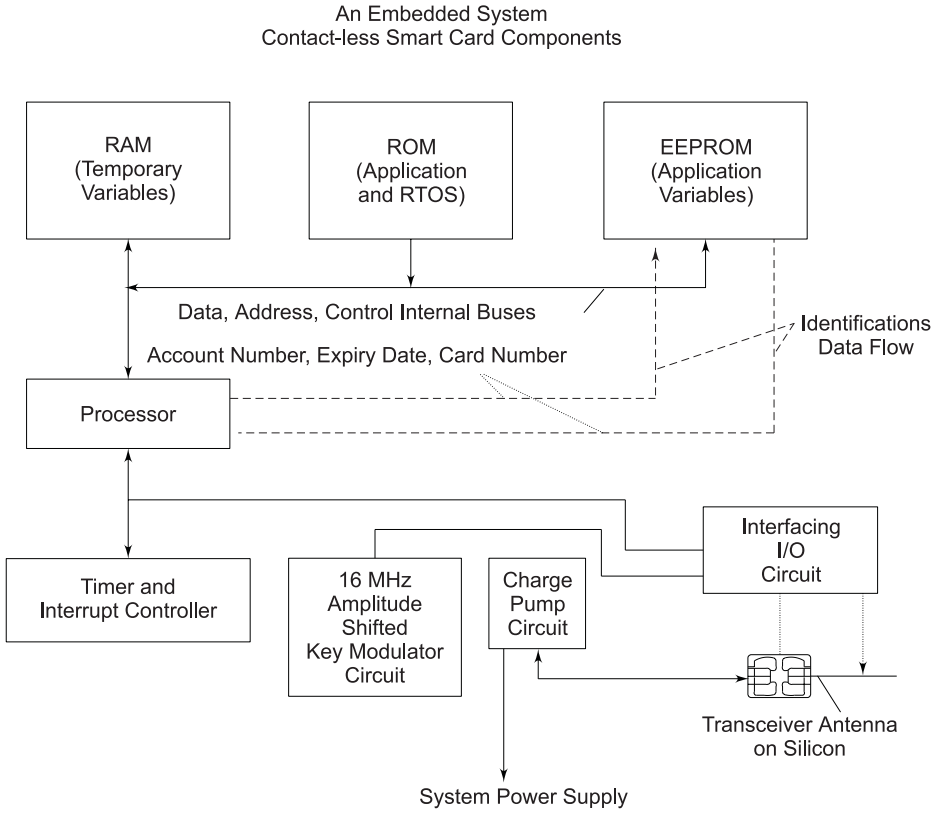

Figure 1.11

An embedded-system hardware components in a contact-less smart card.

SUMMARY

- The embedded system is a sophisticated system consisting of several hardware and software components, and its design may be several times more complex than that of a PC and the programs running on a PC.
- The embedded system processor can be a general-purpose processor chosen from number of families of processors, microcontrollers, embedded processors and digital signal processors (DSPs). Alternatively, an application specific instruction processor (ASIP) may be designed for specific application on a VLSI chip. An ASSP may be additionally used for fast hardwired implementation of a certain part of the embedded software. A sophisticated embedded system may use a multiprocessor unit also.
- Embedded system embeds (locates) a software image in the ROM. The image mostly consists of the following: (i) Boot up program. (ii) Initialization data. (iii) Strings for an initial screen-display or system state. (iv) Programs for the multiple tasks that the system performs. (v) RTOS kernel.

- The embedded system needs a power source and controlled and optimized power-dissipation from the total energy requirement for given hardware and software. The charge pump provides a power-supply-less system in certain embedded systems.
- The embedded system needs clock and reset circuits. Use of the clock manager is a recent innovation.
- The embedded system needs interfaces: Input Output (*IO*) ports, serial *UART* and other pots to accept inputs and to send outputs by interacting with the peripherals, display units, keypad or keyboard.
- The embedded system may need bus controllers for networking its buses with other systems.
- The embedded system needs timers and a watchdog timer for the system clock and for real-time program scheduling and control.
- The embedded system needs an *interrupt controlling unit.*
- The embedded system may need ADC for taking analog input from one or multiple sources. It needs DAC using PWM for sending analog output to motors, speakers, sound systems, etc.
- The embedded system may need an LED or LCD display units, keypad and keyboard, pulse dialer, modem, transmitter, multiplexers and demultiplexers.
- Embedded software is usually made in the high-level languages C or C++ or Java with certain features added, enabled or disabled for programming. 'C' and C++ also facilitates the incorporation of assembly language codes.
- The embedded system most often needs a real-time operating system for real-time programming and scheduling, device drivers, device management and multitasking.
- There are a number of software tools needed in the development and design phase of an embedded system. [Refer to Table 1.7.]
- There are a large number of applications and products that employ embedded systems.
- A VLSI chip can embed IPs for the specific applications, besides the ASIP or a GPP core. A system-on-chip is the latest concept in embedded systems, for example, a mobile phone. A contact-less smart card is one such application, the detail units of which were shown in Figure 1.11.

LIST OF KEYWORDS AND THEIR DEFINITIONS

- *System*: A way of working, organizing or doing some task or series of tasks by following the fixed plan, program and set of rules
- *Embedded system*: A sophisticated system that has a computer (hardware with application software and RTOS embedded in it) as one of its components. An embedded system is a dedicated computer-based system for an application or product.
- *Processor*: A processor implements a process or processes as per the command (instruction) given to it.
- *Process*: A program or task or thread that *has a distinct memory allocation of its own and has one or more functions or procedures for specific job. The process may share the memory (data) with other tasks. A processor may run multiple processes separately or concurrently.*
- *Microcontroller*: A unit with a processor. Memory, timers, watchdog timer, interrupt controller, ADC or PWM, etc. are provided as required by the application.

- ∑ *GPP (General-purpose processor)*: A processor from a number of families of processors, microcontrollers, embedded processors and digital signal processors (DSPs) having a general-purpose instruction set and readily available compilers to enable programming in a high level language.
- ∑ *ASSP (Application Specific System Processor)*: A processing unit for specific tasks, for example, image compression, and that is integrated through the buses with the main processor in the embedded system.
- ∑ *ASIP (Application specific Instruction processor)*: A processor designed for specific application on a VLSI chip.
- ∑ *FPGA*: These are Field Programmable Gate Arrays on a chip. The chip has a large number of arrays with each element having fusable links. Each element of array consists of several XOR, AND, OR, multiplexer, demultiplexer and tristate gates. By appropriate programming of the fusable links, a design of a complex digital circuit is created on the chip.
- ∑ *Registers*: These are associated with the processor and temporarily store the variable values from the memory and from the execution unit during processing of an instruction (s).
- ∑ *Clock*: Fixed frequency pulses that an oscillator circuit generates and that controls all operations during processing and all timing references of the system. Frequency depends on the needs of the processor circuit.A processor, if it needs 100 MHz clock then its minimum instruction processing time is a reciprocal of it, which is 10 ns.
- ∑ *Reset*: A processor state in which the processor registers acquire initial values and from which starts an initial program; this program is usually the one that also runs on power up.
- ∑ *Reset circuit*: A circuit to force reset state and that gets activated for a short period on power up. When reset is activated, the processor generates a reset signal for the other system units needing reset.
- ∑ *Memory*: This stores all the programs, input data and output data. The processor fetches instructions from it to execute and gives the processed results back to it as per the instruction.
- ∑ *ROM*: A read only memory that locates the following in its ROM- embedded software, initial data and strings and operating system or RTOS.
- ∑ *RAM*: This is a Random Access Read and Write memory that the processor uses to store programs and data that are volatile and which disappear on power down or off.
- ∑ *Cache*: A fast *read and write* on-chip unit for the processor execution unit. It stores a copy of a page of instructions and data. It has these fetched in advance from the ROM and RAM so that the processor does not have to wait for instruction and data from external buses.
- ∑ *Timer*: A unit to provide the time for the system clock and real-time operations and scheduling.
- ∑ *Watchdog timer*: A timer the timeout from which resets the processor in case the program gets stuck for an unexpected time.
- ∑ *Interrupt controller*: A unit that controls the processor operations arising out of an interrupt from a source.
- ∑ *ADC*: A unit that converts, as required, the analog input between + and – pins with respect to the reference voltage (s) to digital 8 or 10 or 12 bits.
- ∑ *PWM*: Pulse width modulator to provide a pulse of width scaled to the analog output desired. On integrating PWM output, the DAC operation is achieved.
- ∑ *DAC*: Digital bits (8 or 10 or 12) converted to analog signal scaled to a reference voltage (s).

- *Input Output (IO) ports*: The system gets the inputs and outputs from these. Through these, the keypad or LCD units attach to the system.
- *UART*: Universal Asynchronous Receiver and Transmitter.
- *LED*: Light Emitting diode—a diode that emits red, green, yellow or infrared light on forward biasing between 1.6V to 2 V and currents between 8 - 15mA. Multi-segment and multi-line LED units are used for bright display of digits, characters, charts and short messages.
- *LCD*: Liquid Crystal diode—a diode that absorbs or emits light on application of 3 to 4 V 50 or 60 Hz voltage pulses with currents ~ 50 mA. Multi-segment and multi-line LCD units are used for a display of digits, characters, charts and short messages with very low power dissipation.
- *Modem*: A circuit to modulate the outgoing bits into pulses usually used on the telephone line and to demodulate the incoming pulses into bits for incoming messages.
- *Multiplexer*: A digital circuit that has digital inputs from multiple channels. It sends only one channel output at a time. The channel at the output has the same address as the channel address bits in its input.
- *Demultiplexer*: A digital circuit that has digital outputs at any instance in multiple channels. The channel that is connected is the one that has the same address as the channel address bits in its input.
- *Compiler*: A program that, according to the processor specification, generates machine codes from the high level language. The codes are called object codes.
- *Assembler*: A program that translates assembly language software into the machine codes placed in a file called '.exe' (executable) file.
- *Linker*: A program that links the compiled codes with the other codes and provides the input for a loader or locator.
- *Loader:* It is a program that reallocates the physical memory addresses for loading into the system RAM memory. Reallocation is necessary, as available memory may not start from 0x0000 at a given instant of processing in a computer. The loader is a part of the OS in a computer.
- *Locator*: It is a program to reallocate the linked files of the program application and the RTOS codes at the actual addresses of the ROM memory. It creates a file in a standard format. File is called ROM image.
- *Device Programmer*: It takes the inputs from a file generated by the locator and burns the fusable link to actually store the data and codes at the ROM.
- *Mask and ROM mask*: Created at a foundry for fabrication of a chip. The ROM mask is created from the ROM image file.
- *Physical Device*: A device like a printer or keypad connected to the system port.
- *Virtual Device*: A file or pipe that is programmed for opening and closing and for reading and writing, such as a program for attaching and detaching a physical device and for input and output.
- *Pipe*: A data structure (*or virtual device*) which is sent a byte stream from a data source (for example, a program structure) and which delivers the byte stream to the data sink (for example, a printer).
- *File*: A data structure (*or virtual device*) which sends the records (characters or words) to a data sink (for example, a program structure) and which stores the data from the data source (for example, a program structure). A file in computer may also be stored at the hard disk.

- *Device Driver*: Interrupt Service routine Software, which runs after the programming of the control register (or word) of a peripheral device (or virtual device) and to let the device get the inputs or outputs. It executes on an interrupt to or from the device.
- *Device manager*: Software to manage multiple devices and drivers.
- *Multitasking*: Processing codes for the different tasks as directed by the scheduler.
- *Kernel*: A program with functions for memory allocation and de-allocation, task scheduling, inter-process communication, effective management of shared memory access by using the signals, exception (error) handling signals, semaphores, queues, mailboxes, pipes and sockets [See Section 8.3], I/O management, interrupts control (Handler), device drivers and device management.
- *Real-time operating system*: Operating System software for real-time programming and scheduling, process and memory manager, device drivers, device management and multitasking.
- *VLSI chip*: A very large-scale integrated circuit made on silicon with ~ 1M transistors.
- *System on Chip*: A system on a VLSI chip that has all of needed analog as wells as digital circuits, for example, in a mobile phone.

REVIEW QUESTIONS

1. Define a system. Now define embedded system.
2. What are the essential structural units in (a) microprocessor (b) Embedded processor (c) Microcontroller (d) DSP (e) ASIP (f) ASIP? List each of these.
3. How does a DSP differ from a general-purpose processor (GPP)? Refer Sections 1.2.5 and D.2.
4. What are the advantages and disadvantages of (a) a processor with only fixed-point arithmetic unit and (b) a processor with additional floating-point arithmetic processing unit?
5. A new innovation is media processors [Section E.1.] Refer Sections 1.2.5, D.2 and E.1. How does a media processor differ from a DSP?
6. Explain the media processor use in convergence technology embedded system like mobile phone with mail client, Internet connectivity and image-frame downloads.
7. Compare features in an exemplary family chip (or core) of each of the following: Microprocessor, Microcontroller, RISC Processor, Digital Signal Processor, ASSP, Video processor and media processor. Refer Section 1.2 and Appendices A to E.
8. Why does late generation systems operate processor at low voltages (<2 V) and IO at (~3.3V)?
9. What are the techniques of power and energy management in a system?
10. What is the advantage of running a processor at reduced clock speed in certain section of instructions and at full speed in other section of instructions?
11. What is the advantage of the followings? (a) Stop instruction (b) Wait instruction (c) Processor idle mode operation (d) Cache-use disable instruction (e) Cache with multi-ways and blocks in an embedded system.
12. What do we mean by charge pump? How does a charge pump supply power in an embedded system without using the power supply lines?
13. What do you mean by ‘real time’ and ‘real time clock’?

14. What is the role of *processor reset* and *system reset*?
15. Explain the need of watchdog timer and reset after the watched time.
16. What is the role of RAM in an embedded system?
17. Why do we need multiple actions and multiple controlling tasks for the devices in an embedded system? Explain it with an example of the embedded system, a *remote* of color TV.
18. When do we need multitasking OS?
19. When do we need an RTOS?
20. Why should be embedded system RTOS be scalable?
21. Explain the terms IP core, FPGA, CPLD, PLA and PAL
22. What do you mean by System-on-Chip (SoC)? Examine the designed table in Question 1.3 above. How will the definition of embedded system change with System-on-Chip?
23. What are the advantages offered by an FPGA for designing an embedded system?
24. What are the advantages offered by an ASIC for designing an embedded system?
25. What are the advantages offered by an ASIP for designing an embedded system?
26. Real time video processing needs sophisticated embedded systems with hard real time constraints. Why? Explain it.
27. Why does a processor system always need an 'Interrupts Handler (Interrupt Controller)'?
28. What does role linker play?
29. Why do we use loader in a computer system and locator in an embedded system?
30. Why does a program reside in ROM in the embedded system?
31. Define ROM image and explain each section of an ROM image in an exemplary system.
32. When will you use the compressed program and data in ROM? Give five examples of embedded systems having these in their ROM images.
33. When will you use SRAM and when DRAM? Explain your replies.
34. What do we mean by the following: Physical device, Virtual device, Plug and Play device, Bus self-powered device, device Management and Device Specific Processor.

PRACTICE EXERCISES

35. Search definitions of embedded system from books referred in 'References' and tabulate these with definitions in column 1 and reference in column 2.
36. Classify the embedded systems into small scale, medium scale and sophisticated systems. Now, reclassify these embedded systems with and without real-time (response time constrained) systems and give 10 examples of each.
37. An automobile cruise control system is to be designed in a project. What will be skills needed in the team of hardware and software engineers?
38. Take a value, x = 1.7320508075688. It is squared once again by a floating-point arithmetic processor unit. Now x is squared by a 16-bit integer fixed point arithmetic processing unit. How does the result differ? [Note: Fixed-point unit will multiply only 17320 with 17320, divide the result by 10000 and then again divide the result by 10000.]
39. Design four columns table two examples of embedded systems in each row's columns 2 and 3. Column 1: the type of processor needed among the followings: Microprocessor, Microcontroller, Embedded Processor, Digital Signal Processor, ASSP, Video-processor and Media processor. Give reasoning in column 4.

40. Why does a CMOS IO circuit power dissipation reduces by compared to 5V, factor of half, ~$(3.3/5)^2$, in IO 3.3V operation?
41. How much shall be reduction in power dissipation for a processor CMOS circuit when V reduces from 5V to 1.8V operation?
42. Refer Sections 1.3.5 and G.1 to G.3. List various type of memories and application of each in the followings: Robot, Electronic smart weight display system, ECG LCD display-cum-recorder, Router, Digital Camera, Speech Processing, Smart Card, Embedded Firewall/ Router, Mail Client card, and Transreceiver system with a collision control and jabber control [Collision control means transmission and reception when no other system on the network is using the network. Jabber control means control of continuous streams of random data flowing on a network, which eventually chokes a network.]
43. Tabulate hardware units needed in each of the system mentioned in Question 42 above.
44. Give two examples of embedded systems, which need one or more of following units. (a) DAC (Using a PWM) (b) ADC (c) LCD display (d) LED Displays (e) Keypad (f) Pulse Dialer (g) Modem (h) Transceiver (i) GPIB (IEEE 488) Link
45. An ADC is a 10-bit ADC? It has reference voltages, $V_{ref-} = 0.0V$ and $V_{ref+} = 1.023V$. What will be the ADC outputs when inputs are (a) – 0.512 V (b) + 0.512 V and (c) +2047V? What shall be ADC outputs in three situations when (i) $V_{ref-} = 0.512$ V and $V_{ref+} = 1.023V$ (ii) $V_{ref-} = 1.024$ V and $V_{ref+} = 2.047V$ and (ii) $V_{ref-} = -1.024$ V and $V_{ref+} = +2.047V$.
46. Refer Sections 1.4, 5.1, 5.8 and 5.9. Tabulate the advantages and disadvantages of using coding language as following: (a) Final Machine Implementable (b) ALP (Assembly Language Programming (c) 'C' (d) C++ (e) Java
47. List the software tools needed in designing each of the Embedded System examples in Question 42.
48. Justify the importance of device drivers in an embedded system. Refer to Section 1.4.5 and 4.1.3.
49. Cost of designing an embedded system may be thousands of times the cost of its processor and hardware units. Explain this statement.
50. FPGA (Field Programmable Gate Arrays) core integrated with a single or multiple processor units on chip and FPSLIC (Field Programmable System Level Integrated Circuits) are recent novel innovations. How do these help in the design of sophisticated embedded systems for real time video processing?