1.1 EMBEDDED SYSTEMS

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.

Understanding the basic concepts is essential in the learning of any subject. Designing an *Embedded System* is not a herculean task if you know the fundamentals. Like any general computing systems, Embedded Systems also possess a set of characteristics which are unique to the embedded system under consideration. In contrast to the general computing systems, Embedded Systems are highly domain and application specific in nature, meaning; they are specifically designed for certain set of applications in certain domains like consumer electronics, telecom, automotive, industrial control, measurement systems etc. Unlike general computing systems it is not possible to replace an embedded system which is specifically designed for an application catering to a specific domain with another embedded system catering to another domain. The designer of the embedded system should be aware of its characteristics, and its domain and application specific nature.

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 only clockwise. (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 inclination 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 activated to wash the dirty clothes placed in a tank, which rotates or spins in preprogrammed 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 the motor fast 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

Definition One of the definitions of *embedded system* is as follows:

"An embedded system is a system that has embedded software and computer-hardware, which makes it a system dedicated for an application(s) or specific part of an application or product or a part of a larger system."

Embedded systems have been defined in books published recently in several ways. 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."

An embedded system is an electrical/electro mechanical system which is specifically designed for an application catering to a specific domain. It is a combination of specialised hardware and firmware (software), which is tailored to meet the requirements of the application under consideration. An embedded system contains a processing unit which can be a microprocessor or a microcontroller or a System on Chip (SoC) or an Application Specific Integrated Circuit (ASIC)/Application Specific Standard Product (ASSP) or a Programmable Logic Device (PLD) like FPGA or CPLD, an I/O subsystem which facilitates the interfacing of sensors and actuators which acts as the messengers from and to the '*Real world*' to which the embedded system is interacting, on-board and external communication interfaces for communicating between the various on-board subsystems and chips which builds the embedded system and external systems to which the embedded system interacts, and other supervisory systems and support units like watchdog timers, reset circuits, brown-out protection circuits, regulated power supply unit, clock generation circuit etc. which empower and monitor the functioning of the embedded system. The design of embedded system has two aspects: The hardware design which takes care of the selection of the processing unit, the various I/O sub systems and communication interface and the inter connection among them, and the design of the embedded firmware which deals with configuring the various sub systems, implementing data communication and processing/controlling algorithm requirements.

Depending on the response requirements and the type of applications for which the embedded system is designed, the embedded system can be a *Real-time* or a Non-real time system. The response requirements for a real-time system like *Flight Control System*, *Airbag Deployment System* for Automotive etc, are time critical and the hardware and firmware design aspects for such systems should take these into account, whereas the response requirements for non-real time systems like *Automatic Teller Machines* (*ATM*), *Media Playback Systems* etc, need not be time critical and missing deadlines may be acceptable in such systems.

Like any other systems, embedded systems also possess a set of quality attributes, which are the non-functional requirements like security, scalability, availability, maintainability, safety, portability etc. The non-functional requirements for the embedded system should be identified and should be addressed properly in the system design. The designer of the embedded system should be aware of the different non-functional requirement for the embedded system and should handle this properly to ensure high quality.

1.4 CLASSIFICATION OF EMBEDDED SYSTEMS

It is possible to have a multitude of classifications for embedded systems, based on different criteria. Some of the criteria used in the classification of embedded systems are:

1. Based on generation
2. Complexity and performance requirements
3. Based on deterministic behaviour
4. Based on triggering.

The classification based on deterministic system behaviour is applicable for 'Real Time' systems. The application/task execution behaviour for an embedded system can be either deterministic or non-deterministic. Based on the execution behaviour, Real Time embedded systems are classified into *Hard* and *Soft*. We will discuss about hard and soft real time systems in a later chapter. Embedded Systems which are 'Reactive' in nature (Like process control systems in industrial control applications) can be classified based on the trigger. Reactive systems can be either *event triggered* or *time triggered*.

1.4.1 Classification Based on Generation

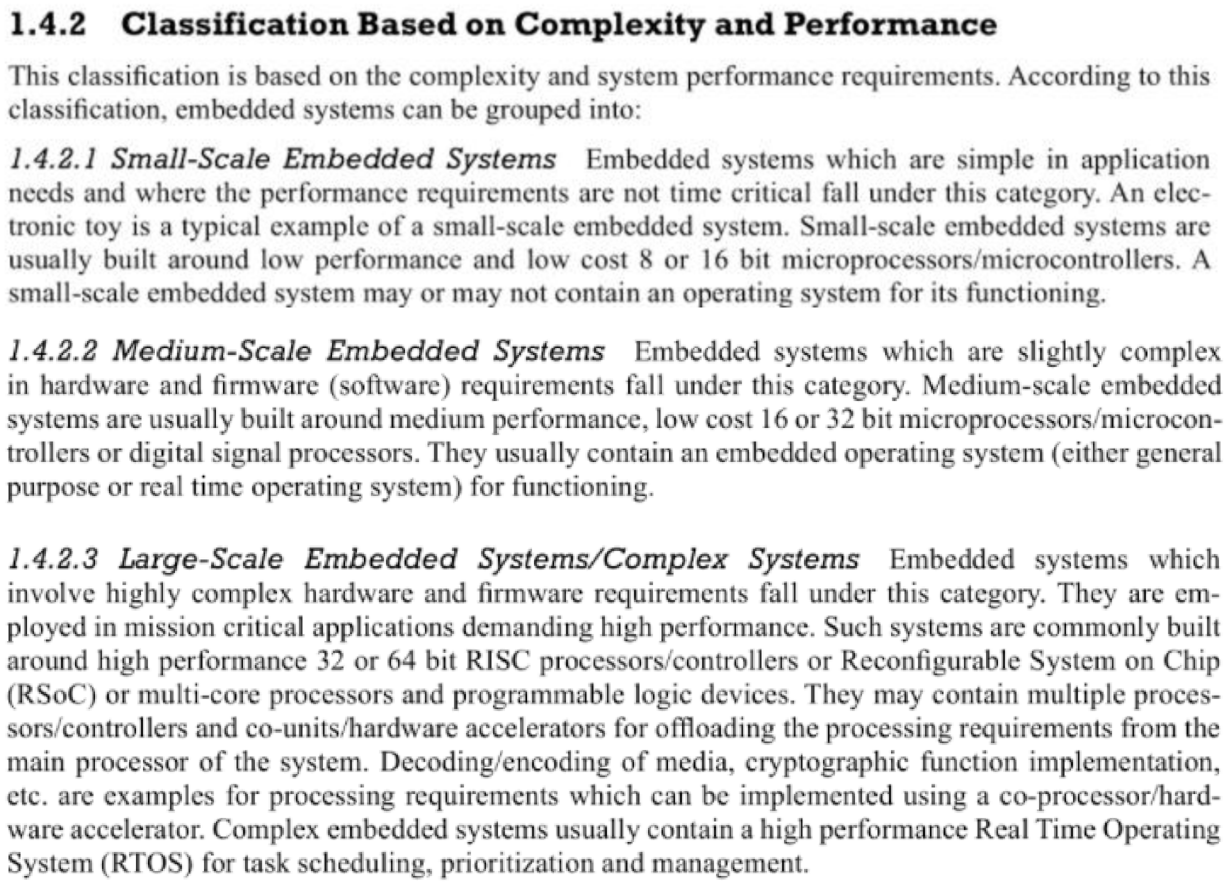
This classification is based on the order in which the embedded processing systems evolved from the first version to where they are today. As per this criterion, embedded systems can be classified into:

1.4.1.1 First Generation The early embedded systems were built around 8bit microprocessors like 8085 and Z80, and 4bit microcontrollers. Simple in hardware circuits with firmware developed in Assembly code. Digital telephone keypads, stepper motor control units etc. are examples of this.

1.4.1.2 Second Generation These are embedded systems built around 16bit microprocessors and 8 or 16 bit microcontrollers, following the first generation embedded systems. The instruction set for the second generation processors/controllers were much more complex and powerful than the first generation processors/controllers. Some of the second generation embedded systems contained embedded operating systems for their operation. Data Acquisition Systems, SCADA systems, etc. are examples of second generation embedded systems.

1.4.1.3 Third Generation With advances in processor technology, embedded system developers started making use of powerful 32bit processors and 16bit microcontrollers for their design. A new concept of application and domain specific processors/controllers like Digital Signal Processors (DSP) and Application Specific Integrated Circuits (ASICs) came into the picture. The instruction set of processors became more complex and powerful and the concept of instruction pipelining also evolved. The processor market was flooded with different types of processors from different vendors. Processors like Intel Pentium, Motorola 68K, etc. gained attention in high performance embedded requirements. Dedicated embedded real time and general purpose operating systems entered into the embedded market. Embedded systems spread its ground to areas like robotics, media, industrial process control, networking, etc.

1.4.1.4 Fourth Generation The advent of System on Chips (SoC), reconfigurable processors and multicore processors are bringing high performance, tight integration and miniaturisation into the embedded device market. The SoC technique implements a total system on a chip by integrating different functionalities with a processor core on an integrated circuit. We will discuss about SoCs in a later chapter. The fourth generation embedded systems are making use of high performance real time embedded operating systems for their functioning. Smart phone devices, mobile internet devices (MIDs), etc. are examples of fourth generation embedded systems.



1.4.2 Classification Based on Complexity and Performance

This classification is based on the complexity and system performance requirements. According to this classification, embedded systems can be grouped into:

1.4.2.1 Small-Scale Embedded Systems Embedded systems which are simple in application needs and where the performance requirements are not time critical fall under this category. An electronic toy is a typical example of a small-scale embedded system. Small-scale embedded systems are usually built around low performance and low cost 8 or 16 bit microprocessors/microcontrollers. A small-scale embedded system may or may not contain an operating system for its functioning.

1.4.2.2 Medium-Scale Embedded Systems Embedded systems which are slightly complex in hardware and firmware (software) requirements fall under this category. Medium-scale embedded systems are usually built around medium performance, low cost 16 or 32 bit microprocessors/microcontrollers or digital signal processors. They usually contain an embedded operating system (either general purpose or real time operating system) for functioning.

1.4.2.3 Large-Scale Embedded Systems/Complex Systems Embedded systems which involve highly complex hardware and firmware requirements fall under this category. They are employed in mission critical applications demanding high performance. Such systems are commonly built around high performance 32 or 64 bit RISC processors/controllers or Reconfigurable System on Chip (RSoC) or multi-core processors and programmable logic devices. They may contain multiple processors/controllers and co-units/hardware accelerators for offloading the processing requirements from the main processor of the system. Decoding/encoding of media, cryptographic function implementation, etc. are examples for processing requirements which can be implemented using a co-processor/hardware accelerator. Complex embedded systems usually contain a high performance Real Time Operating System (RTOS) for task scheduling, prioritization and management.

Classification based on deterministic behaviour

This classification is applicable for “Real Time” systems.

- The task execution behaviour for an embedded system may be deterministic or non-deterministic.
- Based on execution behaviour Real Time embedded systems are divided into:
  - Hard Real Time Systems.
  - Soft Real Time Systems.

Classification based on triggering

Embedded systems which are “Reactive” in nature can be based on triggering.

- Reactive systems can be:
  - Event triggered
  - Time triggered

1.5 MAJOR APPLICATION AREAS OF EMBEDDED SYSTEMS

We are living in a world where embedded systems play a vital role in our day-to-day life, starting from home to the computer industry, where most of the people find their job for a livelihood. Embedded technology has acquired a new dimension from its first generation model, the Apollo guidance computer, to the latest radio navigation system combined with in-car entertainment technology and the microprocessor based "Smart" running shoes launched by Adidas in April 2005. The application areas and the products in the embedded domain are countless. A few of the important domains and products are listed below:

1. *Consumer electronics:* Camcorders, cameras, etc.
2. *Household appliances:* Television, DVD players, washing machine, fridge, microwave oven, etc.
3. *Home automation and security systems:* Air conditioners, sprinklers, intruder detection alarms, closed circuit television cameras, fire alarms, etc.
4. *Automotive industry:* Anti-lock breaking systems (ABS), engine control, ignition systems, automatic navigation systems, etc.
5. *Telecom:* Cellular telephones, telephone switches, handset multimedia applications, etc.
6. *Computer peripherals:* Printers, scanners, fax machines, etc.
7. *Computer networking systems:* Network routers, switches, hubs, firewalls, etc.
8. *Healthcare:* Different kinds of scanners, EEG, ECG machines etc.
9. *Measurement & Instrumentation:* Digital multi meters, digital CROs, logic analyzers PLC systems, etc.
10. *Banking & Retail:* Automatic teller machines (ATM) and currency counters, point of sales (POS)
11. *Card Readers:* Barcode, smart card readers, hand held devices, etc.

1.6 PURPOSE OF EMBEDDED SYSTEMS

As mentioned in the previous section, embedded systems are used in various domains like consumer electronics, home automation, telecommunications, automotive industry, healthcare, control & instrumentation, retail and banking applications, etc. Within the domain itself, according to the application usage context, they may have different functionalities. Each embedded system is designed to serve the purpose of any one or a combination of the following tasks:

1. Data collection/Storage/Representation
2. Data communication
3. Data (signal) processing
4. Monitoring
5. Control
6. Application specific user interface

1.6.1 Data Collection/Storage/Representation

Embedded systems designed for the purpose of data collection performs acquisition of data from the external world. Data collection is usually done for storage, analysis, manipulation and transmission. The term "data" refers all kinds of information, viz. text, voice, image, video, electrical signals and any other measurable quantities. Data can be either analog (continuous) or digital (discrete). Embedded systems with analog data capturing techniques collect data directly in the form of analog signals whereas embedded systems with digital data collection mechanism converts the analog signal to corresponding digital signal using analog to digital (A/D) converters and then collects the binary equivalent of the analog data. If the data is digital, it can be directly captured without any additional interface by digital embedded systems.

1.6.1 Data Collection/Storage/Representation

Embedded systems designed for the purpose of data collection performs acquisition of data from the external world. Data collection is usually done for storage, analysis, manipulation and transmission. The term "data" refers all kinds of information, viz. text, voice, image, video, electrical signals and any other measurable quantities. Data can be either analog (continuous) or digital (discrete). Embedded systems with analog data capturing techniques collect data directly in the form of analog signals whereas embedded systems with digital data collection mechanism converts the analog signal to corresponding digital signal using analog to digital (A/D) converters and then collects the binary equivalent of the analog data. If the data is digital, it can be directly captured without any additional interface by digital embedded systems.

The collected data may be stored directly in the system or may be transmitted to some other systems or it may be processed by the system or it may be deleted instantly after giving a meaningful representation. These actions are purely dependent on the purpose for which the embedded system is designed. Embedded systems designed for pure measurement applications without storage, used in control and instrumentation domain, collects data and gives a meaningful representation of the collected data by means of graphical representation or quantity value and deletes the collected data when new data arrives at the data collection terminal. Analog and digital CROs without storage memory are typical examples of this. Any measuring equipment used in the medical domain for monitoring without storage functionality also comes under this category.

Some embedded systems store the collected data for processing and analysis. Such systems incorporate a built-in/plug-in storage memory for storing the captured data. Some of them give the user a meaningful representation of the collected data by visual (graphical/quantitative) or audible means using display units [Liquid Crystal Display (LCD), Light Emitting Diode (LED), etc.] buzzers, alarms, etc. Examples are: measuring instruments with storage memory and monitoring instruments with storage memory used in medical applications. Certain embedded systems store the data and will not give a representation of the same to the user, whereas the data is used for internal processing.

A digital camera is a typical example of an embedded system with data collection/storage/representation of data. Images are captured and the captured image may be stored within the memory of the camera. The captured image can also be presented to the user through a graphic LCD unit.

Fig. 1.1 **A digital camera for image capturing/storage/display**
(Photo courtesy of Casio-Model EXILIM ex-Z850 (www.casio.com))

1.6.2 Data Communication

Embedded data communication systems are deployed in applications ranging from complex satellite communication systems to simple home networking systems. As mentioned earlier in this chapter, the data collected by an embedded terminal may require transferring of the same to some other system located remotely. The transmission is achieved either by a wire-line medium or by a wireless medium. Wire-line medium was the most common choice in all olden days embedded systems. As technology is changing, wireless medium is becoming the de-facto standard for data communication in embedded systems. A wireless medium offers cheaper connectivity solutions and make the communication link free from the hassle of wire bundles. Data can either be transmitted by analog means or by digital means. Modern industry trends are settling towards digital communication.

The data collecting embedded terminal itself can incorporate data communication units like wireless modules (Bluetooth, ZigBee, Wi-Fi, EDGE, GPRS, etc.) or wire-line modules (RS-232C, USB, TCP/IP, PS2, etc.). Certain embedded systems act as a dedicated transmission unit between the sending and receiving terminals, offering sophisticated functionalities like data packetizing, encrypting and decrypting. Network hubs, routers, switches, etc. are typical examples of dedicated data transmission embedded systems. They act as mediators in data communication and provide various features like data security, monitoring etc.

Fig. 1.2 **A wireless network router for data communication**
(Photo courtesy of Linksys (www.linksys.com). A division of CISCO system)

1.6.3 Data (Signal) Processing

As mentioned earlier, the data (voice, image, video, electrical signals and other measurable quantities) collected by embedded systems may be used for various kinds of data processing. Embedded systems with signal processing functionalities are employed in applications demanding signal processing like speech coding, synthesis, audio video codec, transmission applications, etc.

A digital hearing aid is a typical example of an embedded system employing data processing. Digital hearing aid improves the hearing capacity of hearing impaired persons.

Fig. 1.3 **A digital hearing aid employing signal processing technique**
(Siemens TRIANO 3 Digital hearing aid; Siemens Audiology Copyright© 2005)

1.6.4 Monitoring

Embedded systems falling under this category are specifically designed for monitoring purpose. Almost all embedded products coming under the medical domain are with monitoring functions only. They are used for determining the state of some variables using input sensors. They cannot impose control over variables. A very good example is the electro cardiogram (ECG) machine for monitoring the heartbeat of a patient. The machine is intended to do the monitoring of the heartbeat. It cannot impose control over the heartbeat. The sensors used in ECG are the different electrodes connected to the patient's body.

Some other examples of embedded systems with monitoring function are measuring instruments like digital CRO, digital multimeters, logic analyzers, etc. used in Control & Instrumentation applications. They are used for knowing (monitoring) the status of some variables like current, voltage, etc. They cannot control the variables in turn.

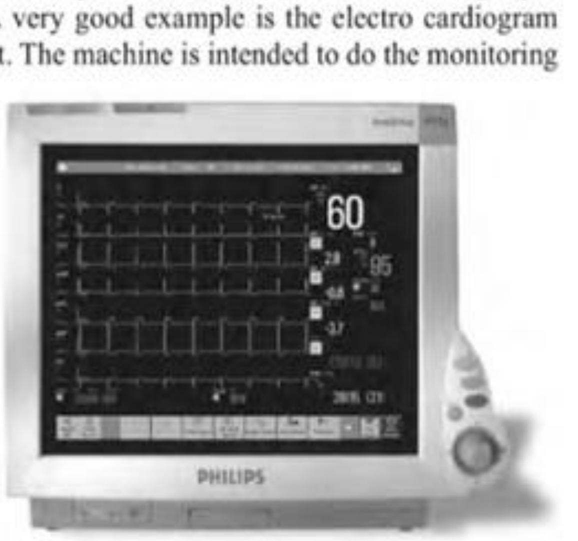


Fig. 1.4 **A patient monitoring system for monitoring heartbeat**
(Photo courtesy of Philips Medical Systems (www.medical.philips.com/))

1.6.5 Control

Embedded systems with control functionalities impose control over some variables according to the changes in input variables. A system with control functionality contains both sensors and actuators. Sensors are connected to the input port for capturing the changes in environmental variable or measuring variable. The actuators connected to the output port are controlled according to the changes in input variable to put an impact on the controlling variable to bring the controlled variable to the specified range.

Air conditioner system used in our home to control the room temperature to a specified limit is a typical example for embedded system for control purpose. An airconditioner contains a room temperature-sensing element (sensor) which may be a thermistor and a handheld unit for setting up (feeding) the desired temperature. The handheld unit may be connected to the central embedded unit residing inside the airconditioner through a wireless link or through a wired link. The air compressor unit acts as the actuator. The compressor is controlled according to the current room temperature and the desired temperature set by the end user.

Fig. 1.5 **"An Airconditioner for controlling room temperature. Embedded System with Control functionality"**
(Photo courtesy of Electrolux Corporation (www.electrolux.com/au))

Here the input variable is the current room temperature and the controlled variable is also the room temperature. The controlling variable is cool air flow by the compressor unit. If the controlled variable and input variable are not at the same value, the controlling variable tries to equalise them through taking actions on the cool air flow.

Components of an Embedded System:

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

1. A microprocessor
2. A large memory of the following two kinds:
   (a) Primary memory (*semiconductor* memories: Random Access Memory (RAM), Read Only Memory (ROM) and fast accessible caches)
   (b) Secondary memory [(*magnetic* memory located in hard disks, diskettes and cartridge tapes, *optical* memory in CD-ROMs or memory sticks (in mobile computers)] using which different user programs can be loaded into the primary memory and run
3. I/O units such as touch screen, modem, fax cum modem, etc.
4. Input units such as keyboard, mice, digitizer, scanner, etc.
5. Output units such as an LCD screen, video monitor, printer, etc.
6. Networking units such as an Ethernet card, front-end processor-based server, bus drivers, etc.
7. An operating system (OS) that has general purpose user and application software in the secondary memory

An embedded system is a system that has three main components embedded into it:

1. It embeds hardware similar to a computer. Figure 1.1 shows the units in the hardware of an embedded system. As its software usually embeds in the ROM or flash memory, it usually do not need a secondary hard disk and CD memory as in a computer
2. It embeds main application software. The application software may concurrently perform a series of tasks or processes or threads
3. It embeds a real-time operating system (RTOS) that supervises the application software running on hardware and organizes access to a resource according to the priorities of tasks in the system. It provides a mechanism to let the processor run a process as scheduled and context-switch between the various processes. (The concept of process, thread and task explained later in Sections 7.1 to 7.3.) It sets the rules during the execution of the application software. (A small-scale embedded system may not embed the RTOS.)

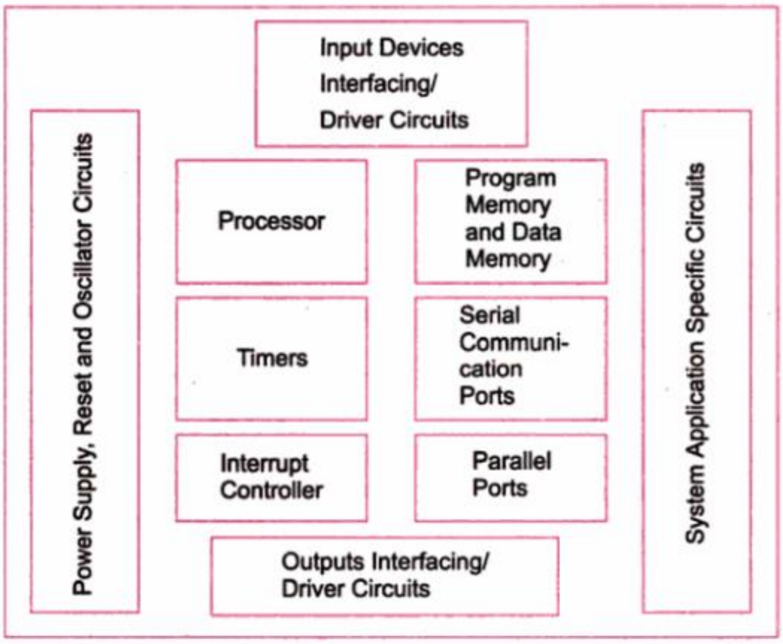

Input Devices Interfacing/ Driver Circuits
Power Supply, Reset and Oscillator Circuits
Processor
Program Memory and Data Memory
Timers
Serial Communi-cation Ports
Interrupt Controller
Parallel Ports
Outputs Interfacing/ Driver Circuits
System Application Specific Circuits

Fig. 1.1 The components of embedded system hardware

1.2 PROCESSOR EMBEDDED INTO A SYSTEM

A processor is an important unit in the embedded system hardware. It is the heart of the embedded system. Knowledge of basic concept of microprocessors and microcontrollers is must for an embedded system designer. A 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, microcontrollers and DSPs that are incorporated in embedded system

1.2.1 Embedded Processors 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 in the form of an IC chip; alternatively, it could be in core form in an Application Specific Integrated Circuit (ASIC) or System on Chip (SoC). Core means a part of the functional circuit on the Very Large Scale Integrated (VLSI) chip.

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

1. General Purpose Processor (GPP): A GPP is a general-purpose processor with instruction set designed not specific to the applications.
   (a) Microprocessor. [Section 1.2.2]
   (b) Embedded Processor [Section 1.7.7]
2. Application Specific Instruction-Set Processor (ASIP). An ASIP is a processor with an instruction set designed for specific applications on a VLSI chip.
   (a) Microcontroller [Section 1.2.3]
   (b) Embedded microcontroller [Section 1.7.7]
   (c) Digital Signal Processor (DSP) and media processor [Section 1.7.3]
   (d) Network processor, IO processor or domain-specific programmable processor
3. Single Purpose Processors as additional processors: Single purpose processor examples are as follows: (1) Coprocessor (e.g., as used for graphic processing, floating point processing, encrypting, deciphering, discrete cosine transformation and inverse transformation or TCP/IP protocol stacking and network connecting functions). (2) Accelerator (e.g., Java codes accelerator). (3) Controllers (e.g., for peripherals, direct memory accesses and buses). [Section 1.7.7]
4. GPP or ASIP cores integrated into either an ASIC or a VLSI circuit or a Field Programmable Gate Array (FPGA) core integrated with processor units in a VLSI (ASIC) chip. [Sections 1.6 and 1.7]
5. Application Specific System Processor (ASSP). [Section 1.7.9]
6. Multicore processors or multiprocessor [Section 1.7]

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) or in an alternate metric *Dhrystone* for measuring processing performance [Section 2.6]
4. Processor ability to solve complex algorithms while meeting deadlines for their processing

A microprocessor or GPP is used because: (i) processing based on the instructions available in a predefined general purpose instruction set results in quick 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 ROM. (iii) Ready availability of a compiler facilitates embedded software development in high-level languages. (iv) Ready availability of well-tested and debugged processor-specific APIs (Application Program Interfaces) and codes previously designed for other applications results in new systems developed quickly.

2.1.1 General Purpose and Domain Specific Processors

Almost 80% of the embedded systems are processor/controller based. The processor may be a microprocessor or a microcontroller or a digital signal processor, depending on the domain and application. Most of the embedded systems in the industrial control and monitoring applications make use of the commonly available microprocessors or microcontrollers whereas domains which require signal processing such as speech coding, speech recognition, etc. make use of special kind of digital signal processors supplied by manufacturers like, Analog Devices, Texas Instruments, etc.

2.1.1.1 Microprocessors A Microprocessor is a silicon chip representing a central processing unit (CPU), which is capable of performing arithmetic as well as logical operations according to a pre-defined set of instructions, which is specific to the manufacturer. In general the CPU contains the Arithmetic and Logic Unit (ALU), control unit and working registers. A microprocessor is a dependent unit and it requires the combination of other hardware like memory, timer unit, and interrupt controller, etc. for proper functioning. Intel claims the credit for developing the first microprocessor unit *Intel 4004*, a 4bit processor which was released in November 1971. It featured 1K data memory, a 12bit program counter and 4K program memory, sixteen 4bit general purpose registers and 46 instructions. It ran at a clock speed of 740 kHz. It was designed for olden day's calculators. In 1972, 14 more instructions were added to the 4004 instruction set and the program space is upgraded to 8K. Also interrupt capabilities were added to it and it is renamed as *Intel 4040*. It was quickly replaced in April 1972 by *Intel 8008* which was similar to *Intel 4040*, the only difference was that its program counter was 14 bits wide and the *8008* served as a terminal controller. In April 1974 Intel launched the first 8 bit processor, the *Intel 8080*, with 16bit address bus and program counter and seven 8bit registers (A-E,H,L: BC, DE, and HL pairs formed the 16bit register for this processor). *Intel 8080* was the most commonly used processors for industrial control and other embedded applications in the 1975s. Since the processor required other hardware components as mentioned earlier for its proper functioning, the systems made out of it were bulky and were lacking compactness.

Immediately after the release of *Intel 8080*, Motorola also entered the market with their processor, *Motorola 6800* with a different architecture and instruction set compared to *8080*.

In 1976 Intel came up with the upgraded version of *8080 – Intel 8085*, with two newly added instructions, three interrupt pins and serial I/O. Clock generator and bus controller circuits were built-in and the power supply part was modified to a single +5 V supply.

In July 1976 Zilog entered the microprocessor market with its *Z80 processor* as competitor to *Intel*. Actually it was designed by an ex-Intel designer, *Frederico Faggin* and it was an improved version of Intel's *8080* processor, maintaining the original *8080* architecture and instruction set with an 8bit data bus and a 16bit address bus and was capable of executing all instructions of *8080*. It included 80 more new instructions and it brought out the concept of register banking by doubling the register set. *Z80* also included two sets of index registers for flexible design.

Technical advances in the field of semiconductor industry brought a new dimension to the microprocessor market and twentieth century witnessed a fast growth in processor technology. 16, 32 and 64 bit processors came into the place of conventional 8bit processors. The initial 2 MHz clock is now an old story. Today processors with clock speeds up to 2.4 GHz are available in the market. More and more competitors entered into the processor market offering high speed, high performance and low cost processors for customer design needs.

Intel, AMD, Freescale, IBM, TI, Cyrix, Hitachi, NEC, LSI Logic, etc. are the key players in the processor market. Intel still leads the market with cutting edge technologies in the processor industry.

Different instruction set and system architecture are available for the design of a microprocessor. Harvard and Von-Neumann are the two common system architectures for processor design. Processors based on Harvard architecture contains separate buses for program memory and data memory, whereas processors based on Von-Neumann architecture shares a single system bus for program and data memory. We will discuss more about these architectures later, under a separate topic. Reduced Instruction Set Computing (RISC) and Complex Instruction Set Computing (CISC) are the two common Instruction Set Architectures (ISA) available for processor design. We will discuss the same under a separate topic in this section.

2.1.1.2 General Purpose Processor (GPP) vs. Application-Specific Instruction Set Processor (ASIP) A General Purpose Processor or GPP is a processor designed for general computational tasks. The processor running inside your laptop or desktop (Pentium 4/AMD Athlon, etc.) is a typical example for general purpose processor. They are produced in large volumes and targeting the general market. Due to the high volume production, the per unit cost for a chip is low compared to ASIC or other specific ICs. A typical general purpose processor contains an Arithmetic and Logic Unit (ALU) and Control Unit (CU). On the other hand, Application Specific Instruction Set Processors (ASIPs) are processors with architecture and instruction set optimised to specific-domain/application requirements like network processing, automotive, telecom, media applications, digital signal processing, control applications, etc. ASIPs fill the architectural spectrum between general purpose processors and Application Specific Integrated Circuits (ASICs). The need for an ASIP arises when the traditional general purpose processor are unable to meet the increasing application needs. Most of the embedded systems are built around application specific instruction set processors. Some microcontrollers (like automotive AVR, USB AVR from Atmel), system on chips, digital signal processors, etc. are examples for application specific instruction set processors (ASIPs). ASIPs incorporate a processor and on-chip peripherals, demanded by the application requirement, program and data memory.

2.1.1.3 Microcontrollers A Microcontroller is a highly integrated chip that contains a CPU, scratch pad RAM, special and general purpose register arrays, on chip ROM/FLASH memory for program storage, timer and interrupt control units and dedicated I/O ports. Microcontrollers can be considered as a super set of microprocessors. Since a microcontroller contains all the necessary functional blocks for independent working, they found greater place in the embedded domain in place of microprocessors. Apart from this, they are cheap, cost effective and are readily available in the market.

Texas Instrument's *TMS 1000* is considered as the world's first microcontroller. We cannot say it as a fully functional microcontroller when we compare it with modern microcontrollers. TI followed Intel's *4004/4040*, 4 bit processor design and added some amount of RAM, program storage memory (ROM) and I/O support on a single chip, there by eliminated the requirement of multiple hardware chips for self-functioning. Provision to add custom instructions to the CPU was another innovative feature of TMS 1000. TMS 1000 was released in 1974.

In 1977 Intel entered the microcontroller market with a family of controllers coming under one umbrella named *MCS-48*™ family. The processors came under this family were *8038HL*, *8039HL*, *8040AHL*, *8048H*, *8049H* and *8050AH*. *Intel 8048* is recognised as Intel's first microcontroller and it was the most prominent member in the MCS-48™† family. It was used in the original IBM PC keyboard. The inspiration behind *8048* was Fairchild's *F8* microprocessor and Intel's goal of developing a low cost and small size processor. The design of *8048* adopted a true Harvard architecture where program and data memory shared the same address bus and is differentiated by the related control signals.

Eventually Intel came out with its most fruitful design in the 8bit microcontroller domain–the *8051 family* and its derivatives. It is the most popular and powerful 8bit microcontroller ever built. It was developed in the 1980s and was put under the family MCS-51. Almost 75% of the microcontrollers used in the embedded domain were *8051 family* based controllers during the 1980–90s. *8051* processor cores are used in more than 100 devices by more than 20 independent manufacturers like Maxim, Philips, Atmel, etc. under the license from Intel. Due to the low cost, wide availability, memory efficient instruction set, mature development tools and Boolean processing (bit manipulation operation) capability, *8051 family* derivative microcontrollers are much used in high-volume consumer electronic devices, entertainment industry and other gadgets where cost-cutting is essential.

Another important family of microcontrollers used in industrial control and embedded applications is the **PIC** family micro controllers from Microchip Technologies (It will be discussed in detail in a later section of this book). It is a high performance RISC microcontroller complementing the CISC (complex instruction set computing) features of *8051*. The terms RISC and CISC will be explained in detail in a separate heading.

Some embedded system applications require only 8bit controllers whereas some embedded applications requiring superior performance and computational needs demand 16/32bit microcontrollers. Infineon, Freescale, Philips, Atmel, Maxim, Microchip etc. are the key suppliers of 16bit microcontrollers. Philips tried to extend the *8051* family microcontrollers to use for 16bit applications by developing the Philips XA (eXtended Architecture) microcontroller series.

8bit microcontrollers are commonly used in embedded systems where the processing power is not a big constraint. As mentioned earlier, more than 20 companies are producing different flavours of the *8051* family microcontroller. They try to add more and more functionalities like built in SPI, I2C serial buses, USB controller, ADC, Networking capability, etc. So the competitive market is driving towards a one-stop solution chip in microcontroller domain. High processing speed microcontroller families like ARM11 series are also available in the market, which provides solution to applications requiring hardware acceleration and high processing capability.

Freescale, NEC, Zilog, Hitachi, Mitsubishi, Infineon, ST Micro Electronics, National, Texas Instruments, Toshiba, Philips, Microchip, Analog Devices, Daewoo, Intel, Maxim, Sharp, Silicon Laboratories, TDK, Triscend, Winbond, Atmel, etc. are the key players in the microcontroller market. Of these Atmel has got special significance. They are the manufacturers of a variety of Flash memory based microcontrollers. They also provide In-System Programmability (which will be discussed in detail in a later section of this book) for the controller. The Flash memory technique helps in fast reprogramming of the chip and thereby reduces the product development time. Atmel also provides another special family of microcontroller called AVR (it will be discussed in detail in a later chapter), an 8bit RISC Flash microcontroller, fast enough to execute powerful instructions in a single clock cycle and provide the latitude you need to optimise power consumption.

The instruction set architecture of a microcontroller can be either RISC or CISC. Microcontrollers are designed for either general purpose application requirement (general purpose controller) or domain-specific application requirement (application specific instruction set processor). The *Intel 8051* microcontroller is a typical example for a general purpose microcontroller, whereas the automotive AVR microcontroller family from Atmel Corporation is a typical example for ASIP specifically designed for the automotive domain.

Difference between Microprocessor and Microcontroller:

Microprocessor	Microcontroller
A silicon chip representing a central processing unit (CPU), which is capable of performing arithmetic as well as logical operations according to a pre-defined set of instructions	A microcontroller is a highly integrated chip that contains a CPU, scratchpad RAM, special and general purpose register arrays, on chip ROM/FLASH memory for program storage, timer and interrupt control units and dedicated I/O ports
It is a dependent unit. It requires the combination of other chips like timers, program and data memory chips, interrupt controllers, etc. for functioning	It is a self-contained unit and it doesn't require external interrupt controller, timer, UART, etc. for its functioning
Most of the time general purpose in design and operation	Mostly application-oriented or domain-specific
Doesn't contain a built in I/O port. The I/O port functionality needs to be implemented with the help of external programmable peripheral interface chips like 8255	Most of the processors contain multiple built-in I/O ports which can be operated as a single 8 or 16 or 32 bit port or as individual port pins
Targeted for high end market where performance is important	Targeted for embedded market where performance is not so critical (At present this demarcation is invalid)
Limited power saving options compared to microcontrollers	Includes lot of power saving features

2.1.1.5 Digital Signal Processors Digital Signal Processors (DSPs) are powerful special purpose 8/16/32 bit microprocessors designed specifically to meet the computational demands and power constraints of today's embedded audio, video, and communications applications. Digital signal processors are 2 to 3 times faster than the general purpose microprocessors in signal processing applications. This is because of the architectural difference between the two. DSPs implement algorithms in hardware which speeds up the execution whereas general purpose processors implement the algorithm in firmware and the speed of execution depends primarily on the clock for the processors. In general, DSP can be viewed as a microchip designed for performing high speed computational operations for 'addition', 'subtraction', 'multiplication' and 'division'. A typical digital signal processor incorporates the following key units:

Program Memory Memory for storing the program required by DSP to process the data

Data Memory Working memory for storing temporary variables and data/signal to be processed.

Computational Engine Performs the signal processing in accordance with the stored program memory. Computational Engine incorporates many specialised arithmetic units and each of them operates simultaneously to increase the execution speed. It also incorporates multiple hardware shifters for shifting operands and thereby saves execution time.

I/O Unit Acts as an interface between the outside world and DSP. It is responsible for capturing signals to be processed and delivering the processed signals.

Audio video signal processing, telecommunication and multimedia applications are typical examples where DSP is employed. Digital signal processing employs a large amount of real-time calculations. Sum of products (SOP) calculation, convolution, fast fourier transform (FFT), discrete fourier transform (DFT), etc, are some of the operations performed by digital signal processors.

Blackfin®[†] processors from Analog Devices is an example of DSP which delivers breakthrough signal-processing performance and power efficiency while also offering a full 32-bit RISC MCU programming model. Blackfin processors present high-performance, homogeneous software targets, which allows flexible resource allocation between hard real-time signal processing tasks and non real-time control tasks. System control tasks can often run in the shadow of demanding signal processing and multimedia tasks.

2.1.1.6 RISC vs. CISC Processors/Controllers

The term RISC stands for Reduced Instruction Set Computing. As the name implies, all RISC processors/controllers possess lesser number of instructions, typically in the range of 30 to 40. CISC stands for Complex Instruction Set Computing. From the definition itself it is clear that the instruction set is complex and instructions are high in number.

From a programmers point of view RISC processors are comfortable since s/he needs to learn only a few instructions, whereas for a CISC processor s/he needs to learn more number of instructions and should understand the context of usage of each instruction (This scenario is explained on the basis of a programmer following Assembly Language coding. For a programmer following C coding it doesn't matter since the cross-compiler is responsible for the conversion of the high level language instructions to machine dependent code). Atmel AVR microcontroller is an example for a RISC processor and its instruction set contains only 32 instructions. The original version of *8051* microcontroller (e.g. *AT89C51*) is a CISC controller and its instruction set contains 255 instructions. Remember it is not the number of instructions that determines whether a processor/controller is CISC or RISC. There are some other factors like pipelining features, instruction set type, etc. for determining the RISC/CISC criteria. Some of the important criteria are listed below:

RISC	**CISC**
Lesser number of instructions	Greater number of Instructions
Instruction pipelining and increased execution speed	Generally no instruction pipelining feature
Orthogonal instruction set (Allows each instruction to operate on any register and use any addressing mode)	Non-orthogonal instruction set (All instructions are not allowed to operate on any register and use any addressing mode. It is instruction-specific)
Operations are performed on registers only, the only memory operations are load and store	Operations are performed on registers or memory depending on the instruction
A large number of registers are available	Limited number of general purpose registers
Programmer needs to write more code to execute a task since the instructions are simpler ones	Instructions are like macros in C language. A programmer can achieve the desired functionality with a single instruction which in turn provides the effect of using more simpler single instructions in RISC
Single, fixed length instructions	Variable length instructions
Less silicon usage and pin count	More silicon usage since more additional decoder logic is required to implement the complex instruction decoding.
With Harvard Architecture	Can be Harvard or Von-Neumann Architecture

2.1.1.7 Harvard vs. Von-Neumann Processor/Controller Architecture

The terms Harvard and Von-Neumann refers to the processor architecture design.

Microprocessors/controllers based on the **Von-Neumann** architecture shares a single common bus for fetching both instructions and data. Program instructions and data are stored in a common main memory. Von-Neumann architecture based processors/controllers first fetch an instruction and then fetch the data to support the instruction from code memory. The two separate fetches slows down the controller's operation. Von-Neumann architecture is also referred as **Princeton** architecture, since it was developed by the Princeton University.

Microprocessors/controllers based on the **Harvard** architecture will have separate data bus and instruction bus. This allows the data transfer and program fetching to occur simultaneously on both buses. With Harvard architecture, the data memory can be read and written while the program memory is being accessed. These separated data memory and code memory buses allow one instruction to execute while the next instruction is fetched ("pre-fetching"). The pre-fetch theoretically allows much faster execution than Von-Neumann architecture. Since some additional hardware logic is required for the generation of control signals for this type of operation it adds silicon complexity to the system. Figure 2.2 explains the Harvard and Von-Neumann architecture concept.

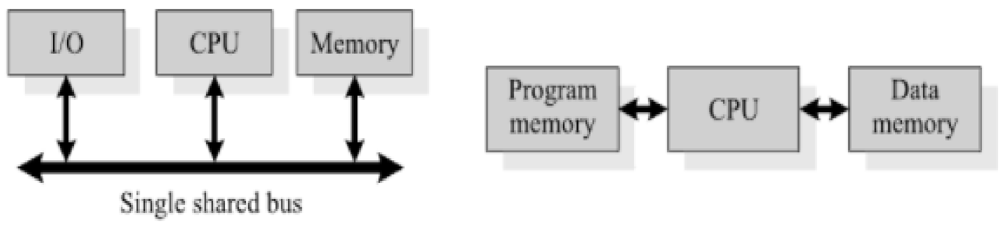


Fig. 2.2 **Harvard vs Von-Neumann architecture**

The following table highlights the differences between Harvard and Von-Neumann architecture.

Harvard Architecture	**Von-Neumann Architecture**
Separate buses for instruction and data fetching	Single shared bus for instruction and data fetching
Easier to pipeline, so high performance can be achieved	Low performance compared to Harvard architecture
Comparatively high cost	Cheaper
No memory alignment problems	Allows self modifying codes†
Since data memory and program memory are stored physically in different locations, no chances for accidental corruption of program memory	Since data memory and program memory are stored physically in the same chip, chances for accidental corruption of program memory

2.1.1.8 Big-Endian vs. Little-Endian Processors/Controllers Endianness specifies the order in which the data is stored in the memory by processor operations in a multi byte system (Processors whose word size is greater than one byte). Suppose the word length is two byte then data can be stored in memory in two different ways:

1. Higher order of data byte at the higher memory and lower order of data byte at location just below the higher memory.
2. Lower order of data byte at the higher memory and higher order of data byte at location just below the higher memory.

Little-endian (Fig. 2.3) means the lower-order byte of the data is stored in memory at the lowest address, and the higher-order byte at the highest address. (The little end comes first.) For example, a 4 byte long integer **Byte3 Byte2 Byte1 Byte0** will be stored in the memory as shown below:

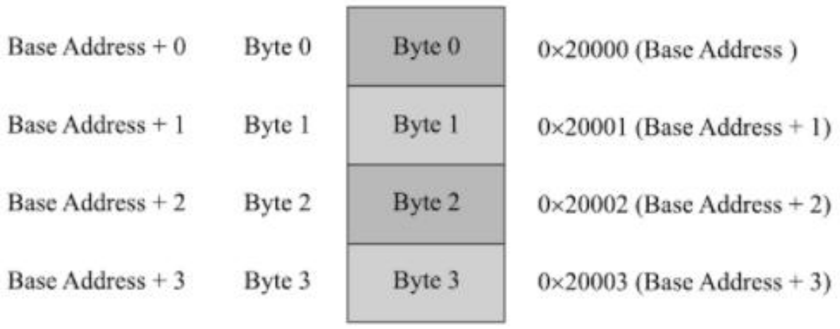


Fig. 2.3 **Little-Endian operation**

Big-endian (Fig. 2.4) means the higher-order byte of the data is stored in memory at the lowest address, and the lower-order byte at the highest address. (The big end comes first.) For example, a 4 byte long integer **Byte3 Byte2 Byte1 Byte0** will be stored in the memory as follows‡:

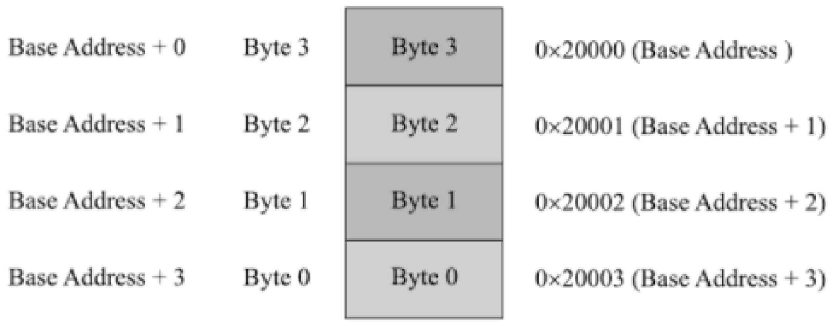


Fig. 2.4 **Big-Endian operation**

2.1.1.9 Load Store Operation and Instruction Pipelining As mentioned earlier, the RISC processor instruction set is orthogonal, meaning it operates on registers. The memory access related operations are performed by the special instructions *load* and *store*. If the operand is specified as memory location, the content of it is loaded to a register using the *load* instruction. The instruction *store* stores data from a specified register to a specified memory location. The concept of **Load Store Architecture** is illustrated with the following example:

Suppose *x*, *y* and *z* are memory locations and we want to add the contents of *x* and *y* and store the result in location *z*. Under the load store architecture the same is achieved with 4 instructions as shown in Fig. 2.5.

The first instruction *load R1, x* loads the register R1 with the content of memory location *x*, the second instruction *load R2,y* loads the register R2 with the content of memory location *y*. The instruction

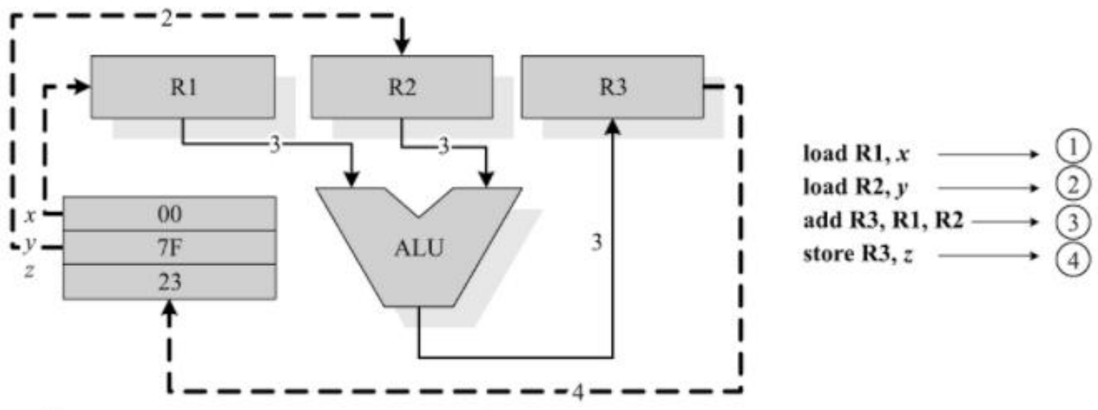


Fig. 2.5 **The concept of load store architecture**

add R3, R1, R2 adds the content of registers R1 and R2 and stores the result in register R3. The next instruction *store R3,z* stores the content of register R3 in memory location *z*.

2.1.2 Application Specific Integrated Circuits (ASICs)

Application Specific Integrated Circuit (ASIC) is a microchip designed to perform a specific or unique application. It is used as replacement to conventional general purpose logic chips. It integrates several functions into a single chip and there by reduces the system development cost. Most of the ASICs are proprietary products. As a single chip, ASIC consumes a very small area in the total system and thereby helps in the design of smaller systems with high capabilities/functionalities.

ASICs can be pre-fabricated for a special application or it can be custom fabricated by using the components from a re-usable '*building block*' library of components for a particular customer application. ASIC based systems are profitable only for large volume commercial productions. Fabrication of ASICs requires a non-refundable initial investment for the process technology and configuration expenses. This investment is known as Non-Recurring Engineering Charge (NRE) and it is a one time investment.

If the Non-Recurring Engineering Charges (NRE) is borne by a third party and the Application Specific Integrated Circuit (ASIC) is made openly available in the market, the ASIC is referred as Application Specific Standard Product (ASSP). The ASSP is marketed to multiple customers just as a general-purpose product is, but to a smaller number of customers since it is for a specific application. "The ADE7760 Energy Metre ASIC developed by Analog Devices for Energy metreing applications is a typical example for ASSP".

Since Application Specific Integrated Circuits (ASICs) are proprietary products, the developers of such chips may not be interested in revealing the internal details of it and hence it is very difficult to point out an example of it. Moreover it will create legal disputes if an illustration of such an ASIC product is given without getting prior permission from the manufacturer of the ASIC. For the time being, let us forget about it. We will come back to it in another part of this book series (Namely, Designing Advanced Embedded Systems).

2.1.3 Programmable Logic Devices

Logic devices provide specific functions, including device-to-device interfacing, data communication, signal processing, data display, timing and control operations, and almost every other function a system must perform. Logic devices can be classified into two broad categories–fixed and programmable. As the name indicates, the circuits in a fixed logic device are permanent, they perform one function or set of functions–once manufactured, they cannot be changed. On the other hand, Programmable Logic Devices (PLDs) offer customers a wide range of logic capacity, features, speed, and voltage characteristics–and these devices can be re-configured to perform any number of functions at any time.

With programmable logic devices, designers use inexpensive software tools to quickly develop, simulate, and test their designs. Then, a design can be quickly programmed into a device, and immediately tested in a live circuit. The PLD that is used for this prototyping is the exact same PLD that will be used in the final production of a piece of end equipment, such as a network router, a DSL modem, a DVD player, or an automotive navigation system. There are no NRE costs and the final design is completed much faster than that of a custom, fixed logic device. Another key benefit of using PLDs is that during the design phase customers can change the circuitry as often as they want until the design operates to their satisfaction. That's because PLDs are based on re-writable memory technology–to change the design, the device is simply reprogrammed. Once the design is final, customers can go into immediate production by simply programming as many PLDs as they need with the final software design file.

2.1.3.1 CPLDs and FPGAs The two major types of programmable logic devices are Field Programmable Gate Arrays (FPGAs) and Complex Programmable Logic Devices (CPLDs). Of the two, FPGAs offer the highest amount of logic density, the most features, and the highest performance. The largest FPGA now shipping, part of the Xilinx **Virtex™**[†] line of devices, provides eight million "system gates" (the relative density of logic). These advanced devices also offer features such as built-in hardwired processors (such as the IBM power PC), substantial amounts of memory, clock management systems, and support for many of the latest, very fast device-to-device signaling technologies. FPGAs are used in a wide variety of applications ranging from data processing and storage, to instrumentation, telecommunications, and digital signal processing.

CPLDs, by contrast, offer much smaller amounts of logic–up to about 10,000 gates. But CPLDs offer very predictable timing characteristics and are therefore ideal for critical control applications. CPLDs such as the Xilinx **CoolRunner™**[†] series also require extremely low amounts of power and are very inexpensive, making them ideal for cost-sensitive, battery-operated, portable applications such as mobile phones and digital handheld assistants.

Advantages of PLD Programmable logic devices offer a number of important advantages over fixed logic devices, including:

- PLDs offer customers much more flexibility during the design cycle because design iterations are simply a matter of changing the programming file, and the results of design changes can be seen immediately in working parts.
- PLDs do not require long lead times for prototypes or production parts–the PLDs are already on a distributor's shelf and ready for shipment.
- PLDs do not require customers to pay for large NRE costs and purchase expensive mask sets–PLD suppliers incur those costs when they design their programmable devices and are able to amortize those costs over the multi-year lifespan of a given line of PLDs.
- PLDs allow customers to order just the number of parts they need, when they need them, allowing them to control inventory. Customers who use fixed logic devices often end up with excess inventory which must be scrapped, or if demand for their product surges, they may be caught short of parts and face production delays.
- PLDs can be reprogrammed even after a piece of equipment is shipped to a customer. In fact, thanks to programmable logic devices, a number of equipment manufacturers now tout the ability to add new features or upgrade products that already are in the field. To do this, they simply upload a new programming file to the PLD, via the Internet, creating new hardware logic in the system.

Over the last few years programmable logic suppliers have made such phenomenal technical advances that PLDs are now seen as the logic solution of choice from many designers. One reason for this is that PLD suppliers such as Xilinx are "fabless" companies; instead of owning chip manufacturing foundries, Xilinx outsource that job to partners like Toshiba and UMC, whose chief occupation is making chips. This strategy allows Xilinx to focus on designing new product architectures, software tools, and intellectual property cores while having access to the most advanced semiconductor process technologies. Advanced process technologies help PLDs in a number of key areas: faster performance,

Over the last few years programmable logic suppliers have made such phenomenal technical advances that PLDs are now seen as the logic solution of choice from many designers. One reason for this is that PLD suppliers such as Xilinx are "fabless" companies; instead of owning chip manufacturing foundries, Xilinx outsource that job to partners like Toshiba and UMC, whose chief occupation is making chips. This strategy allows Xilinx to focus on designing new product architectures, software tools, and intellectual property cores while having access to the most advanced semiconductor process technologies. Advanced process technologies help PLDs in a number of key areas: faster performance,

integration of more features, reduced power consumption, and lower cost.

FPGAs are especially popular for prototyping ASIC designs where the designer can test his design by downloading the design file into an FPGA device. Once the design is set, hardwired chips are produced for faster performance.

Just a few years ago, for example, the largest FPGA was measured in tens of thousands of system gates and operated at 40 MHz. Older FPGAs also were relatively expensive, costing often more than $150 for the most advanced parts at the time. Today, however, FPGAs with advanced features offer

2.2 MEMORY

Memory is an important part of a processor/controller based embedded systems. Some of the processors/controllers contain built in memory and this memory is referred as **on-chip memory**. Others do not contain any memory inside the chip and requires external memory to be connected with the controller/processor to store the control algorithm. It is called **off-chip memory**. Also some working memory is required for holding data temporarily during certain operations. This section deals with the different types of memory used in embedded system applications.

2.2.1 Program Storage Memory (ROM)

The program memory or code storage memory of an embedded system stores the program instructions and it can be classified into different types as per the block diagram representation given in Fig. 2.8.

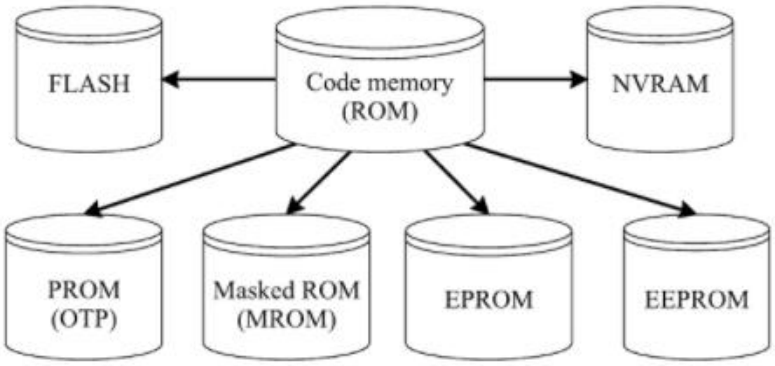


Fig. 2.8 **Classification of Program Memory (ROM)**

The code memory retains its contents even after the power to it is turned off. It is generally known as non-volatile storage memory. Depending on the fabrication, erasing and programming techniques they are classified into the following types.

2.2.1.1 Masked ROM (MROM) Masked ROM is a one-time programmable device. Masked ROM makes use of the hardwired technology for storing data. The device is factory programmed by masking and metallisation process at the time of production itself, according to the data provided by the end user. The primary advantage of this is low cost for high volume production. They are the least expensive type of solid state memory. Different mechanisms are used for the masking process of the ROM, like

1. Creation of an enhancement or depletion mode transistor through channel implant.
2. By creating the memory cell either using a standard transistor or a high threshold transistor. In the high threshold mode, the supply voltage required to turn ON the transistor is above the normal ROM IC operating voltage. This ensures that the transistor is always off and the memory cell stores always logic 0.

Masked ROM is a good candidate for storing the embedded firmware for low cost embedded devices. Once the design is proven and the firmware requirements are tested and frozen, the binary data (The firmware cross compiled/assembled to target processor specific machine code) corresponding to it can be given to the MROM fabricator. The limitation with MROM based firmware storage is the inability to modify the device firmware against firmware upgrades. Since the MROM is permanent in bit storage, it is not possible to alter the bit information.

2.2.1.2 Programmable Read Only Memory (PROM) / (OTP) Unlike Masked ROM Memory, One Time Programmable Memory (OTP) or PROM is not pre-programmed by the manufacturer. The end user is responsible for programming these devices. This memory has *nichrome* or *polysilicon* wires arranged in a matrix. These wires can be functionally viewed as fuses. It is programmed by a PROM programmer which selectively burns the fuses according to the bit pattern to be stored. Fuses which are not blown/burned represents a logic "1" whereas fuses which are blown/burned represents a logic "0". The default state is logic "1". OTP is widely used for commercial production of embedded systems whose proto-typed versions are proven and the code is finalised. It is a low cost solution for commercial production. OTPs cannot be reprogrammed.

2.2.1.3 Erasable Programmable Read Only Memory (EPROM) OTPs are not useful and worth for development purpose. During the development phase the code is subject to continuous changes and using an OTP each time to load the code is not economical. Erasable Programmable Read Only Memory (EPROM) gives the flexibility to re-program the same chip. EPROM stores the bit information by charging the floating gate of an FET. Bit information is stored by using an EPROM programmer, which applies high voltage to charge the floating gate. EPROM contains a quartz crystal window for erasing the stored information. If the window is exposed to ultraviolet rays for a fixed duration, the entire memory will be erased. Even though the EPROM chip is flexible in terms of re-programmability, it needs to be taken out of the circuit board and put in a UV eraser device for 20 to 30 minutes. So it is a tedious and time-consuming process.

2.2.1.4 Electrically Erasable Programmable Read Only Memory (EEPROM) As the name indicates, the information contained in the EEPROM memory can be altered by using electrical signals at the register/Byte level. They can be erased and reprogrammed in-circuit. These chips include a chip erase mode and in this mode they can be erased in a few milliseconds. It provides greater flexibility for system design. The only limitation is their capacity is limited when compared with the standard ROM (A few kilobytes).

2.2.1.5 FLASH FLASH is the latest ROM technology and is the most popular ROM technology used in today's embedded designs. FLASH memory is a variation of EEPROM technology. It combines the re-programmability of EEPROM and the high capacity of standard ROMs. FLASH memory is organised as sectors (blocks) or pages. FLASH memory stores information in an array of floating gate MOSFET transistors. The erasing of memory can be done at sector level or page level without affecting the other sectors or pages. Each sector/page should be erased before re-programming. The typical erasable capacity of FLASH is 1000 cycles. W27C512 from WINBOND (www.winbond.com) is an example of 64KB FLASH memory.

2.2.1.6 NVRAM Non-volatile RAM is a random access memory with battery backup. It contains static RAM based memory and a minute battery for providing supply to the memory in the absence of external power supply. The memory and battery are packed together in a single package. The life span of NVRAM is expected to be around 10 years. *DS1644* from Maxim/Dallas is an example of 32KB NVRAM.

2.2.2 Read-Write Memory/Random Access Memory (RAM)

RAM is the data memory or working memory of the controller/processor. Controller/processor can read from it and write to it. RAM is volatile, meaning when the power is turned off, all the contents are destroyed. RAM is a direct access memory, meaning we can access the desired memory location directly without the need for traversing through the entire memory locations to reach the desired memory position (i.e. random access of memory location). This is in contrast to the Sequential Access Memory (SAM), where the desired memory location is accessed by either traversing through the entire memory or through a 'seek' method. Magnetic tapes, CD ROMs, etc. are examples of sequential access memories. RAM generally falls into three categories: Static RAM (SRAM), dynamic RAM (DRAM) and non-volatile RAM (NVRAM) (Fig. 2.9).

2.2.2.1 Static RAM (SRAM) Static RAM stores data in the form of voltage. They are made up of flip-flops. Static RAM is the fastest form of RAM available. In typical implementation, an SRAM cell (bit) is realised using six transistors (or 6 MOSFETs). Four of the transistors are used for building the

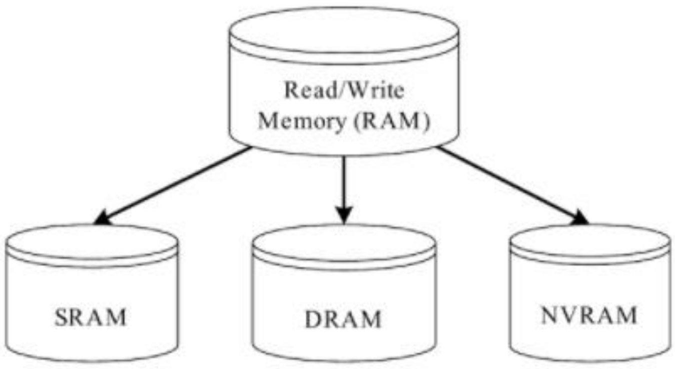


Fig. 2.9 **Classification of Working Memory (RAM)**

latch (flip-flop) part of the memory cell and two for controlling the access. SRAM is fast in operation due to its resistive networking and switching capabilities. In its simplest representation an SRAM cell can be visualised as shown in Fig. 2.10:

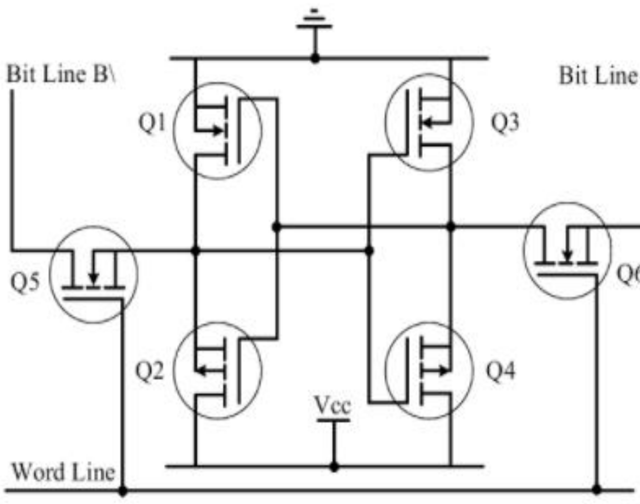


Fig. 2.10 **SRAM cell implementation**

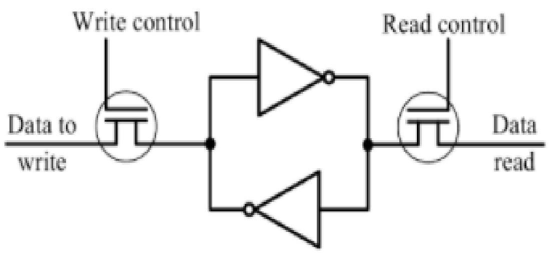


Fig. 2.11 **Visualisation of SRAM cell**

This implementation in its simpler form can be visualised as two-cross coupled inverters with read/write control through transistors. The four transistors in the middle form the cross-coupled inverters. This can be visualised as shown in Fig. 2.11.

From the SRAM implementation diagram, it is clear that access to the memory cell is controlled by the line Word Line, which controls the access transistors (MOSFETs) Q5 and Q6. The access transistors control the connection to bit lines B & B\. In order to write a value to the memory cell, apply the desired value to the bit control lines (For writing 1, make B = 1 and B\ =0; For writing 0, make B = 0 and B\ =1) and assert the Word Line (Make Word line high). This operation latches the bit written in the flip-flop. For reading the content of the memory cell, assert both B and B\ bit lines to 1 and set the Word line to 1.

The major limitations of SRAM are low capacity and high cost. Since a minimum of six transistors are required to build a single memory cell, imagine how many memory cells we can fabricate on a silicon wafer.

2.2.2.2 Dynamic RAM (DRAM) Dynamic RAM stores data in the form of charge. They are made up of MOS transistor gates. The advantages of DRAM are its high density and low cost compared to SRAM. The disadvantage is that since the information is stored as charge it gets leaked off with time and to prevent this they need to be refreshed periodically. Special circuits called DRAM controllers are used for the refreshing operation. The refresh operation is done periodically in milliseconds interval. Figure 2.12 illustrates the typical implementation of a DRAM cell.

The MOSFET acts as the gate for the incoming and outgoing data whereas the capacitor acts as the bit storage unit. Table given below summarises the relative merits and demerits of SRAM and DRAM technology.

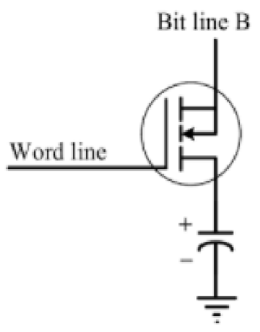


Fig. 2.12 **DRAM cell implementation**

SRAM cell	**DRAM cell**
Made up of 6 CMOS transistors (MOSFET)	Made up of a MOSFET and a capacitor
Doesn't require refreshing	Requires refreshing
Low capacity (Less dense)	High capacity (Highly dense)
More expensive	Less expensive
Fast in operation. Typical access time is 10ns	Slow in operation due to refresh requirements. Typical access time is 60ns. Write operation is faster than read operation.

1.7 COMPLEX SYSTEMS DESIGN AND PROCESSORS

1.7.1 Embedding a Microprocessor

A General Purpose Processor microprocessor can be embedded on a VSLI chip. Table 1.4 lists different streams of microprocessors embedded in a complex system design.

Table 1.4 Important microprocessors used in embedded systems

Stream	*Microprocessor Family*	*Source*	*CISC or RISC or Both features*
Stream 1	68HCxxx	Motorola	CISC
Stream 2	80x86	Intel	CISC
Stream 3	SPARC	Sun	RISC
Stream 4	ARM	ARM	RISC with CISC functionality

1.7.2 Embedding a Microcontroller

Microcontroller VLSI cores or chips for embedded systems are usually among the five streams of families given in Table 1.5.

Table 1.5 Major microcontrollers[@] used in the embedded systems

Stream	*Microcontroller Family*	*Source*	*CISC or RISC or Both*
Stream 1	68HC11xx, HC12xx, HC16xx	Motorola	CISC
Stream 2	8051, 8051MX	Intel, Philips	CISC
Stream 3	PIC 16F84 or 16C76, 16F876 and PIC18	Microchip	CISC
Stream 4[*]	Microcontroller Enhancements of CORTEX-M3 ARM9/ARM7 from Philips, Samsung and ST Microelectronics	ARM, Texas, Philips, Samsung and ST Microelectronics etc.	RISC Core with CISC functionality

[@] 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

1.7.3 Embedding a DSP

A *digital signal processor (DSP)* is a processor core or chip for the applications that process digital signals. [For example, filtering, noise cancellation, echo elimination, compression and encryption applications.] Just as a microprocessor is the most essential unit of a computing system, a DSP is essential unit of an embedded system in 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 image pattern or DNA sequence.

DSP as an ASIP is a single chip or core in a VLSI unit. It includes the computational capabilities of a microprocessor and Multiply and Accumulate (MAC) units. A typical MAC has a 16×32 MAC unit.

DSP executes discrete-time, signal-processing instructions. It has Very Large Instruction Word (VLIW) processing capabilities; it processes Single Instruction Multiple Data (SIMD) instructions; it processes Discrete Cosine Transformations (DCT) and inverse DCT (IDCT) functions. The latter are used in algorithms for signal analyzing, coding, filtering, noise cancellation, echo elimination, compressing and decompressing, etc.

Major DSPs for embedded systems are from the three streams given in Table 1.6.

Table 1.6 Important digital signal processor[@] used in the embedded systems

Stream	*DSP Family*	*Source*
Stream 1	TMS320Cxx, OMAP[1]	Texas
Stream 2	Tiger SHARC	Analog Device
Stream 3	5600xx	Motorola
Stream 4	PNX 1300, 1500[2]	Philips

[1] For example, TMS320C62XX a fixed point 200 MHz DSP (Section 2.3.5).

[2] Media processor, which besides multimedia DSP operations, also does network stream data packet processing.

1.7.4 Embedding an RISC

A RISC microprocessor provides the speedy processing of instructions, each in a single clock-cycle. This facilitates pipelining and superscalar processing. Besides greatly enhanced capabilities mentioned above, there is great enhancement of 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 ARM RISC processor. RISCs are used when the system needs to perform intensive computation, for example, in a speech processing system.

1.7.5 Embedding an ASIP

ASIP is a processor with an instruction set designed for specific application areas on a VLSI chip or core. ASIP examples are microcontroller, DSP, IO, media, network or other domain-specific processor.

Using VLSI design tools, an ASIP with instructions sets required in the specific application areas can be designed. The ASIP is programmed using the instructions of the following functions: DSP, control signals processing, discrete cosine transformations, adaptive filtering and communication protocol-implementing functions.

1.7.6 Embedding a Multiprocessor or Dual Core Using GPPs

In an embedded system, several processors or dual core processors may be needed to execute an algorithm fast within a strict deadline. 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 incorporates two or more processors running in synchronization. An example of using multiple ASIPs is high-definition television signals processing. [High definition means that the signals are processed for a noise-free, echo-cancelled transmission, and for obtaining a flat high-resolution image (1920 × 1020 pixels) on the television screen.] A cell phone or digital camera is another application with multiple ASIPs.

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 used.

Consider a video conferencing system. In this system, a quarter common intermediate format—Quarter-CIF—is used. The number of image pixels is just 144 × 176 as against 525 × 625 pixels in a video picture on TV. Even then, samples of the image have to be taken at a rate of 144 × 176 × 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 are 25 or 30 per second (as per the standard adopted) for real-time displays and in motion pictures.] A single DSP-based embedded system does not suffice to get real-time images during video conferencing. Real-time video processing and multimedia applications most often need a multiprocessor unit in the embedded system.

1.7.7 Embedded Processor/Embedded Microcontroller

An embedded processor is a processor with special features that allow it to embed multiple processes into the system.

Real time image processing and aerodynamics are two areas where fast, precise and intensive calculations and fast context switching (from one program to another) are essential. Embedded processor is the term sometimes used for processor that has been a specially designed such that it has the following capabilities:

1. Fast context switching and thus lower latencies of the tasks in complex real time applications. [Section 4.6] Fast context switching means that the calling program or interrupted service routine CPU registers save and retrieve fast [Section 4.6].
2. 32-bit or 64-bit atomic addition and multiplication, and no shared data problem in the operations with large operands with each operand placed in two or four registers. [Section 7.8.1]

3. 32-bit RISC core for fast, more precise and intensive calculations by the embedded software.

Embedded microcontroller is the term sometimes used for specially designed microcontrollers that have the following capabilities:

1. When a microcontroller has internal RAM, large flash or ROM, timer, interrupt handler, devices and peripherals and there is no external memory or device or peripheral required for the given application.
2. Fast context switching and thus lower latencies of the tasks in complex real time applications. For example, ARM and 68HC1x microcontrollers save all CPU registers fast

1.7.8 Embedding ARM processor

Examples of Stream 4 GPPs in Table 1.4 are ARM 7 and ARM 9. The core of these processors can be embedded onto a VLSI chip or an SoC. An ARM-processor VLSI-architecture is available either as a CPU chip or for integrating it into VLSI or SoC. ARM, Intel and Texas Instruments and several other companies have developed such processors. ARM provides CISC functionality with RISC architecture at the core. The cores of ARM7, ARM9 and their DSP enhancements are available for embedding in systems. [Refer to http:/ www.ti.com/sc/ docs/asic/modules/arm7.htm and arm9.htm].

ARM integrates with other features (for example DSP) in new GPPs, which are available from several sources, for example, Intel and Texas Instruments. Exemplary ARM 9 applications are setup boxes, cable modems, and wireless-devices such as mobile handsets.

ARM9 has a single cycle 16×32 multiple accumulate unit. It operates at 200 MHz. It uses 0.15 μm GS30 CMOSs. It has a five-stage pipeline. It incorporates RISC core with CISC functions. It integrates with a DSP when designed for an ASIC solution. An example is its integration with DSP is TMS320C55x from Texas Instruments. [Refer to http:/www.ti.com/sc/docs/asic/modules/arm7.htm and arm9.htm]

A lower performance but very popular version of ARM9 is ARM7. It operates at 80 MHz. It uses 0.18 μm based GS20 μm CMOSs. Using ARM7, ARM9 and CORTEX-M3, a large number of embedded systems have recently become available.

Lately, a new class of embedded systems has emerged that additionally incorporates ASSP chips or cores in its design.

1.7.9 Embedding ASSP

Assume that there is an embedded system for real-time video processing. Real-time processing arises for digital television, high definition TV decoders, set-up boxes, DVD (Digital Video Disc) players, web phones, video-conferencing and other systems. 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 using a specific protocol interconnects, its units through specific bus architecture to another system. Also, assume that suitable encryption and decryption is required. [The output bit stream encryption protects 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

2.5 EMBEDDED FIRMWARE

Embedded firmware refers to the control algorithm (Program instructions) and or the configuration settings that an embedded system developer dumps into the code (Program) memory of the embedded system. It is an un-avoidable part of an embedded system. There are various methods available for developing the embedded firmware. They are listed below.

1. Write the program in high level languages like Embedded C/C++ using an Integrated Development Environment (The IDE will contain an editor, compiler, linker, debugger, simulator, etc. IDEs are different for different family of processors/controllers. For example, Keil micro vision3 IDE is used for all family members of *8051* microcontroller, since it contains the generic *8051* compiler C51).
2. Write the program in Assembly language using the instructions supported by your application's target processor/controller.

The instruction set for each family of processor/controller is different and the program written in either of the methods given above should be converted into a processor understandable machine code before loading it into the program memory.

The process of converting the program written in either a high level language or processor/controller specific Assembly code to machine readable binary code is called '*HEX File Creation*'. The methods used for '*HEX File Creation*' is different depending on the programming techniques used. If the program is written in Embedded C/C++ using an IDE, the cross compiler included in the IDE converts it into corresponding processor/controller understandable '*HEX File*'. If you are following the Assembly language based programming technique (method 2), you can use the utilities supplied by the processor/controller vendors to convert the source code into '*HEX File*'. Also third party tools are available, which may be of free of cost, for this conversion.

The embedded software development process in assembly language is tedious and time consuming. The developer needs to know about all the instruction sets of the processor/controller or at least s/he should carry an instruction set reference manual with her/him. A programmer using assembly language technique writes the program according to his/her view and taste. Often he/she may be writing a method or functionality which can be achieved through a single instruction as an experienced person's point of view, by two or three instructions in his/her own style. So the program will be highly dependent on the developer. It is very difficult for a second person to understand the code written in Assembly even if it is well documented.

3.1 CHARACTERISTICS OF AN EMBEDDED SYSTEM

Unlike general purpose computing systems, embedded systems possess certain specific characteristics and these characteristics are unique to each embedded system. Some of the important characteristics of an embedded system are:

1. Application and domain specific
2. Reactive and Real Time
3. Operates in harsh environments
4. Distributed
5. Small size and weight
6. Power concerns

3.1.1 Application and Domain Specific

If you closely observe any embedded system, you will find that each embedded system is having certain functions to perform and they are developed in such a manner to do the intended functions only. They cannot be used for any other purpose. It is the major criterion which distinguishes an embedded system from a general purpose system. For example, you cannot replace the embedded control unit of your microwave oven with your air conditioner's embedded control unit, because the embedded control units of microwave oven and airconditioner are specifically designed to perform certain specific tasks. Also you cannot replace an embedded control unit developed for a particular domain say telecom with another control unit designed to serve another domain like consumer electronics.

3.1.2 Reactive and Real Time

As mentioned earlier, embedded systems are in constant interaction with the Real world through sensors and user-defined input devices which are connected to the input port of the system. Any changes happening in the Real world (which is called an Event) are captured by the sensors or input devices in Real Time and the control algorithm running inside the unit reacts in a designed manner to bring the controlled output variables to the desired level. The event may be a periodic one or an unpredicted one. If the event is an unpredicted one then such systems should be designed in such a way that it should be scheduled to capture the events without missing them. Embedded systems produce changes in output in response to the changes in the input. So they are generally referred as Reactive Systems.

Real Time System operation means the timing behaviour of the system should be deterministic; meaning the system should respond to requests or tasks in a known amount of time. A Real Time system should not miss any deadlines for tasks or operations. It is not necessary that all embedded systems should be Real Time in operations. Embedded applications or systems which are mission critical, like flight control systems, Antilock Brake Systems (ABS), etc. are examples of Real Time systems. The design of an embedded Real time system should take the worst case scenario into consideration.

3.1.3 Operates in Harsh Environment

It is not necessary that all embedded systems should be deployed in controlled environments. The environment in which the embedded system deployed may be a dusty one or a high temperature zone or an area subject to vibrations and shock. Systems placed in such areas should be capable to withstand all these adverse operating conditions. The design should take care of the operating conditions of the area where the system is going to implement. For example, if the system needs to be deployed in a high temperature zone, then all the components used in the system should be of high temperature grade. Here we cannot go for a compromise in cost. Also proper shock absorption techniques should be provided to systems which are going to be commissioned in places subject to high shock. Power supply fluctuations, corrosion and component aging, etc. are the other factors that need to be taken into consideration for embedded systems to work in harsh environments.

3.1.4 Distributed

The term distributed means that embedded systems may be a part of larger systems. Many numbers of such distributed embedded systems form a single large embedded control unit. An automatic vending machine is a typical example for this. The vending machine contains a card reader (for pre-paid vending systems), a vending unit, etc. Each of them are independent embedded units but they work together to perform the overall vending function. Another example is the Automatic Teller Machine (ATM). An ATM contains a card reader embedded unit, responsible for reading and validating the user's ATM card, transaction unit for performing transactions, a currency counter for dispatching/vending currency to the authorised person and a printer unit for printing the transaction details. We can visualise these as independent embedded systems. But they work together to achieve a common goal.

Another typical example of a distributed embedded system is the Supervisory Control And Data Acquisition (SCADA) system used in Control & Instrumentation applications, which contains physically distributed individual embedded control units connected to a supervisory module.

3.1.5 Small Size and Weight

Product aesthetics is an important factor in choosing a product. For example, when you plan to buy a new mobile phone, you may make a comparative study on the pros and corns of the products available in the market. Definitely the product aesthetics (size, weight, shape, style, etc.) will be one of the deciding factors to choose a product. People believe in the phrase "Small is beautiful". Moreover it is convenient to handle a compact device than a bulky product. In embedded domain also compactness is a significant deciding factor. Most of the application demands small sized and low weight products.

3.1.6 Power Concerns

Power management is another important factor that needs to be considered in designing embedded systems. Embedded systems should be designed in such a way as to minimise the heat dissipation by the system. The production of high amount of heat demands cooling requirements like cooling fans which in turn occupies additional space and make the system bulky. Nowadays ultra low power components are available in the market. Select the design according to the low power components like low dropout regulators, and controllers/processors with power saving modes. Also power management is a critical constraint in battery operated application. The more the power consumption the less is the battery life.

3.2 QUALITY ATTRIBUTES OF EMBEDDED SYSTEMS

Quality attributes are the non-functional requirements that need to be documented properly in any system design. If the quality attributes are more concrete and measurable it will give a positive impact on the system development process and the end product. The various quality attributes that needs to be addressed in any embedded system development are broadly classified into two, namely 'Operational Quality Attributes' and 'Non-Operational Quality Attributes'.

3.2.1 Operational Quality Attributes

The operational quality attributes represent the relevant quality attributes related to the embedded system when it is in the operational mode or 'online' mode. The important quality attributes coming under this category are listed below:

1. Response
2. Throughput
3. Reliability
4. Maintainability
5. Security
6. Safety

3.2.1.1 Response Response is a measure of quickness of the system. It gives you an idea about how fast your system is tracking the changes in input variables. Most of the embedded systems demand fast response which should be almost Real Time. For example, an embedded system deployed in flight control application should respond in a Real Time manner. Any response delay in the system will create potential damages to the safety of the flight as well as the passengers. It is not necessary that all embedded systems should be Real Time in response. For example, the response time requirement for an electronic toy is not at all time-critical. There is no specific deadline that this system should respond within this particular timeline.

3.2.1.2 Throughput Throughput deals with the efficiency of a system. In general it can be defined as the rate of production or operation of a defined process over a stated period of time. The rates can be expressed in terms of units of products, batches produced, or any other meaningful measurements. In the case of a Card Reader, throughput means how many transactions the Reader can perform in a minute or in an hour or in a day. Throughput is generally measured in terms of 'Benchmark'. A 'Benchmark' is a reference point by which something can be measured. Benchmark can be a set of performance criteria that a product is expected to meet or a standard product that can be used for comparing other products of the same product line.

3.2.1.3 Reliability Reliability is a measure of how much % you can rely upon the proper functioning of the system or what is the % susceptibility of the system to failures.

Mean Time Between Failures (MTBF) and Mean Time To Repair (MTTR) are the terms used in defining system reliability. MTBF gives the frequency of failures in hours/weeks/months. MTTR specifies how long the system is allowed to be out of order following a failure. For an embedded system with critical application need, it should be of the order of minutes.

3.2.1.4 Maintainability Maintainability deals with support and maintenance to the end user or client in case of technical issues and product failures or on the basis of a routine system checkup. Reliability and maintainability are considered as two complementary disciplines. A more reliable system means a system with less corrective maintainability requirements and vice versa. As the reliability of the system increases, the chances of failure and non-functioning also reduces, thereby the need for maintainability is also reduced. Maintainability is closely related to the system availability. Maintainability can be broadly classified into two categories, namely, 'Scheduled or Periodic Maintenance (preventive maintenance)' and 'Maintenance to unexpected failures (corrective maintenance)'. Some embedded products may use consumable components or may contain components which are subject to wear and tear and they should be replaced on a periodic basis. The period may be based on the total hours of the system usage or the total output the system delivered. A printer is a typical example for illustrating the two types of maintainability. An inkjet printer uses ink cartridges, which are consumable components and as per the printer manufacturer the end user should replace the cartridge after each '*n*' number of printouts to get quality prints. This is an example for '**Scheduled or Periodic maintenance**'. If the paper feeding part of the printer fails the printer fails to print and it requires immediate repairs to rectify this problem. This is an example of '**Maintenance to unexpected failure**'. In both of the maintenances (scheduled and repair), the printer needs to be brought offline and during this time it will not be available for the user. Hence it is obvious that maintainability is simply an indication of the availability of the product for use. In any embedded system design, the ideal value for availability is expressed as

3.2.1.5 Security Confidentiality, 'Integrity', and 'Availability' (The term 'Availability' mentioned here is not related to the term 'Availability' mentioned under the 'Maintainability' section) are the three major measures of information security. Confidentiality deals with the protection of data and application from unauthorised disclosure. Integrity deals with the protection of data and application from unauthorised modification. Availability deals with protection of data and application from unauthorized users. A very good example of the 'Security' aspect in an embedded product is a Personal Digital Assistant (PDA). The PDA can be either a shared resource (e.g. PDAs used in LAB setups) or an individual one. If it is a shared one there should be some mechanism in the form of a user name and password to access into a particular person's profile–This is an example of 'Availability'. Also all data and applications present in the PDA need not be accessible to all users. Some of them are specifically accessible to administrators only. For achieving this, Administrator and user levels of security should be implemented –An example of Confidentiality. Some data present in the PDA may be visible to all users but there may not be necessary permissions to alter the data by the users. That is Read Only access is allocated to all users–An example of Integrity.

3.2.1.6 Safety 'Safety' and 'Security' are two confusing terms. Sometimes you may feel both of them as a single attribute. But they represent two unique aspects in quality attributes. Safety deals with the possible damages that can happen to the operators, public and the environment due to the breakdown of an embedded system or due to the emission of radioactive or hazardous materials from the embedded products. The breakdown of an embedded system may occur due to a hardware failure or a firmware failure. Safety analysis is a must in product engineering to evaluate the anticipated damages and determine the best course of action to bring down the consequences of the damages to an acceptable level. As stated before, some of the safety threats are sudden (like product breakdown) and some of them are gradual (like hazardous emissions from the product).

3.2.2 Non-Operational Quality Attributes

The quality attributes that needs to be addressed for the product 'not' on the basis of operational aspects are grouped under this category. The important quality attributes coming under this category are listed below.

1. Testability & Debug-ability
2. Evolvability
3. Portability
4. Time to prototype and market
5. Per unit and total cost.

3.2.2.1 Testability & Debug-ability Testability deals with how easily one can test his/her design, application and by which means he/she can test it. For an embedded product, testability is applicable to both the embedded hardware and firmware. Embedded hardware testing ensures that the peripherals and the total hardware functions in the desired manner, whereas firmware testing ensures that the firmware is functioning in the expected way. Debug-ability is a means of debugging the product as such for figuring out the probable sources that create unexpected behaviour in the total system. Debug-ability

has two aspects in the embedded system development context, namely, hardware level debugging and firmware level debugging. Hardware debugging is used for figuring out the issues created by hardware problems whereas firmware debugging is employed to figure out the probable errors that appear as a result of flaws in the firmware.

3.2.2.2 Evolvability Evolvability is a term which is closely related to Biology. Evolvability is referred as the non-heritable variation. For an embedded system, the quality attribute 'Evolvability' refers to the ease with which the embedded product (including firmware and hardware) can be modified to take advantage of new firmware or hardware technologies.

3.2.2.3 Portability Portability is a measure of 'system independence'. An embedded product is said to be portable if the product is capable of functioning 'as such' in various environments, target processors/controllers and embedded operating systems. The ease with which an embedded product can be ported on to a new platform is a direct measure of the re-work required. A standard embedded product should always be flexible and portable. In embedded products, the term 'porting' represents the migration of the embedded firmware written for one target processor (e.g. Intel x86) to a different target processor (say Hitachi SH3 processor). If the firmware is written in a high level language like 'C' with little target processor-specific functions (operating system extensions or compiler specific utilities), it is very easy to port the firmware for the new processor by replacing those 'target processor-specific functions' with the ones for the new target processor and re-compiling the program for the new target processor-specific settings. Re-compiling the program for the new target processor generates the new target processor-specific machine codes. If the firmware is written in Assembly Language for a particular family of processor (say x86 family), it will be very difficult to translate the assembly language instructions to the new target processor specific language and so the portability is poor.

If you look into various programming languages for application development for desktop applications, you will see that certain applications developed on certain languages run only on specific operating systems and some of them run independent of the desktop operating systems. For example, applications developed using Microsoft technologies (e.g. Microsoft Visual C++ using Visual studio) is capable of running only on Microsoft platforms and will not function on other operating systems; whereas applications developed using 'Java' from Sun Microsystems works on any operating system that supports Java standards.

3.2.2.4 Time-to-Prototype and Market Time-to-market is the time elapsed between the conceptualisation of a product and the time at which the product is ready for selling (for commercial product) or use (for non-commercial products). The commercial embedded product market is highly competitive and time to market the product is a critical factor in the success of a commercial embedded product. There may be multiple players in the embedded industry who develop products of the same category (like mobile phone, portable media players, etc.). If you come up with a new design and if it takes long time to develop and market it, the competitor product may take advantage of it with their product. Also, embedded technology is one where rapid technology change is happening. If you start your design by making use of a new technology and if it takes long time to develop and market the product, by the time you market the product, the technology might have superseded with a new technology. Product prototyping helps a lot in reducing time-to-market. Whenever you have a product idea, you may not be certain about the feasibility of the idea. Prototyping is an informal kind of rapid product development in which the important features of the product under consideration are developed. The time to prototype is also another critical factor. If the prototype is developed faster, the actual estimated development time

can be brought down significantly. In order to shorten the time to prototype, make use of all possible options like the use of off-the-shelf components, re-usable assets, etc.

3.2.2.5 Per Unit Cost and Revenue Cost is a factor which is closely monitored by both end user (those who buy the product) and product manufacturer (those who build the product). Cost is a highly sensitive factor for commercial products. Any failure to position the cost of a commercial product at a nominal rate, may lead to the failure of the product in the market. Proper market study and cost benefit analysis should be carried out before taking a decision on the per-unit cost of the embedded product. From a designer/product development company perspective the ultimate aim of a product is to generate marginal profit. So the budget and total system cost should be properly balanced to provide a marginal profit. Every embedded product has a product life cycle which starts with the design and development phase. The product idea generation, prototyping, Roadmap definition, actual product design and development are the activities carried out during this phase. During the design and development phase there is only investment and no returns. Once the product is ready to sell, it is introduced to the market. This stage is known as the Product Introduction stage. During the initial period the sales and revenue will be low. There won't be much competition and the product sales and revenue increases with time. In the growth phase, the product grabs high market share. During the maturity phase, the growth and sales will be steady and the revenue reaches at its peak. The Product Retirement/Decline phase starts with the drop in sales volume, market share and revenue. The decline happens due to various reasons like competition from similar product with enhanced features or technology changes, etc. At some point of the decline stage, the manufacturer announces discontinuing of the product. The different stages of the embedded

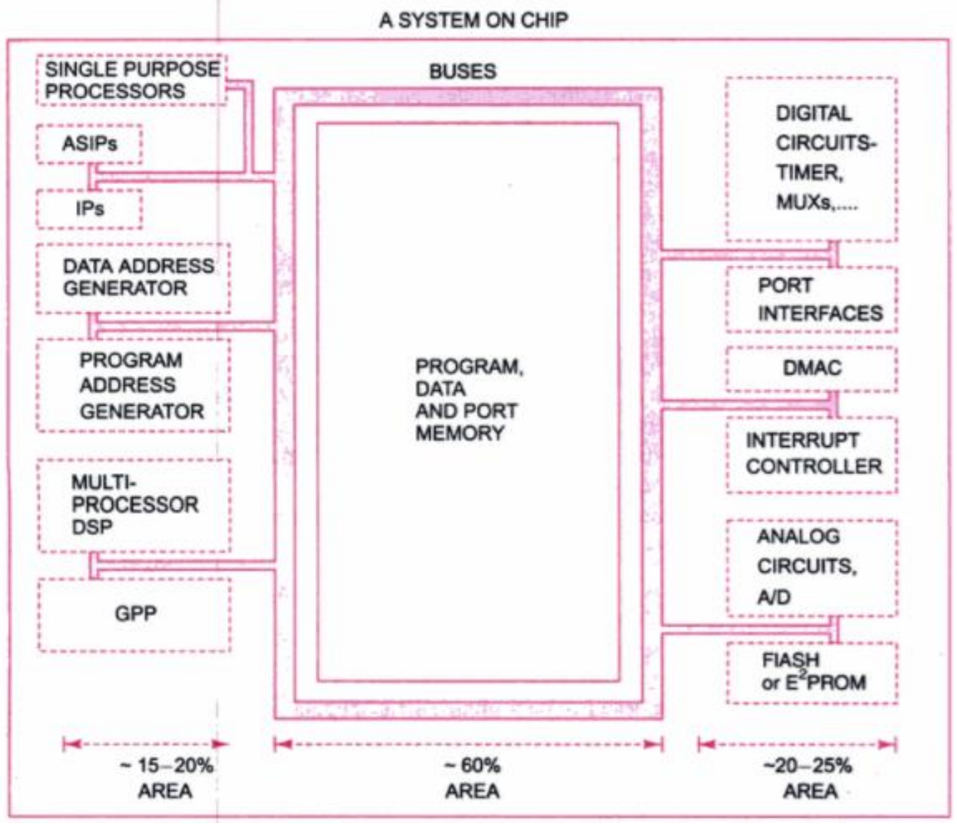
1.6 EMBEDDED SYSTEM-ON-CHIP (SoC) AND USE OF VLSI CIRCUIT DESIGN TECHNOLOGY

Lately, embedded systems are being designed on a single silicon chip, called *System on chip (SoC), a design innovation*. SoC is a system on a VLSI chip that has all the necessary analog as well as digital circuits, processors and software.

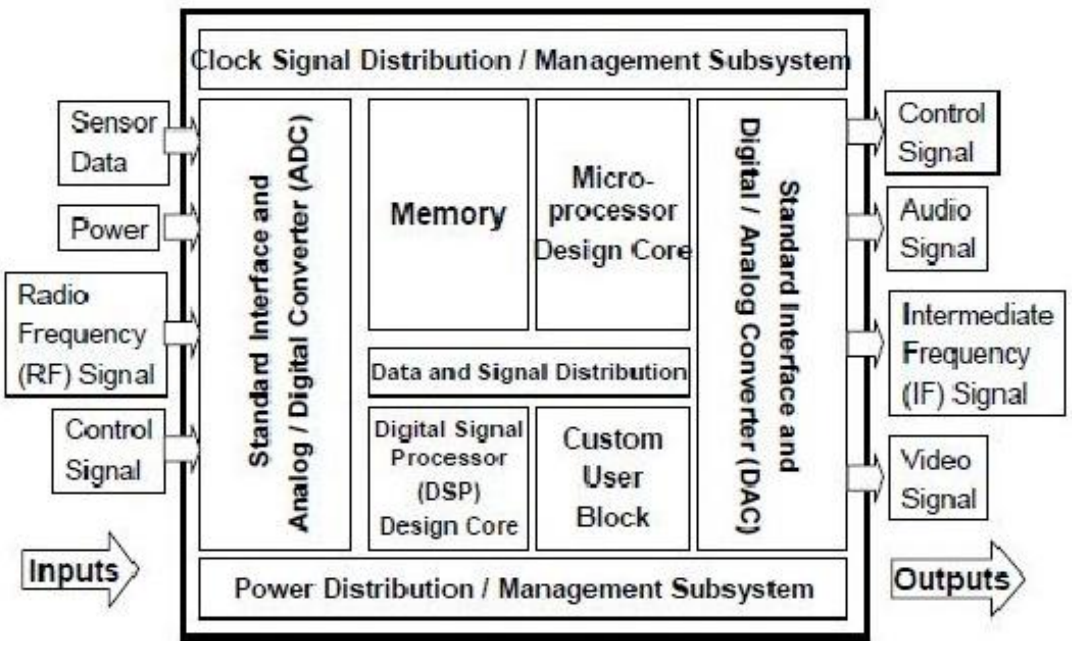
A SoC may be embedded with the following components:

1. Embedded processor GPP or ASIP core,
2. Single purpose processing cores or multiple processors,
3. A network bus protocol core,
4. An encryption function unit,
5. Discrete cosine transforms for signal processing applications,
6. Memories,
7. Multiple standard source solutions, called IP (Intellectual Property) cores,
8. Programmable logic device and FPGA (Field Programmable Gate Array) cores,
9. Other logic and analog units.

An exemplary application of such an embedded SoC is the mobile phone. Single purpose processors, ASIPs and IPs on an SoC are configured to process encoding and deciphering, dialing, modulating, demodulating, interfacing the key pad and multiple line LCD matrix displays or touch screen, storing data input and recalling data from memory. Figure 1.10 shows an SoC that integrates internal ASICs, internal processors (ASIPs), shared memories and peripheral interfaces on a common bus. Besides a processor, memories and digital circuits with embedded software for specific applications, the SoC may possess analog circuits as well.



Example of Complex System-on-Chip

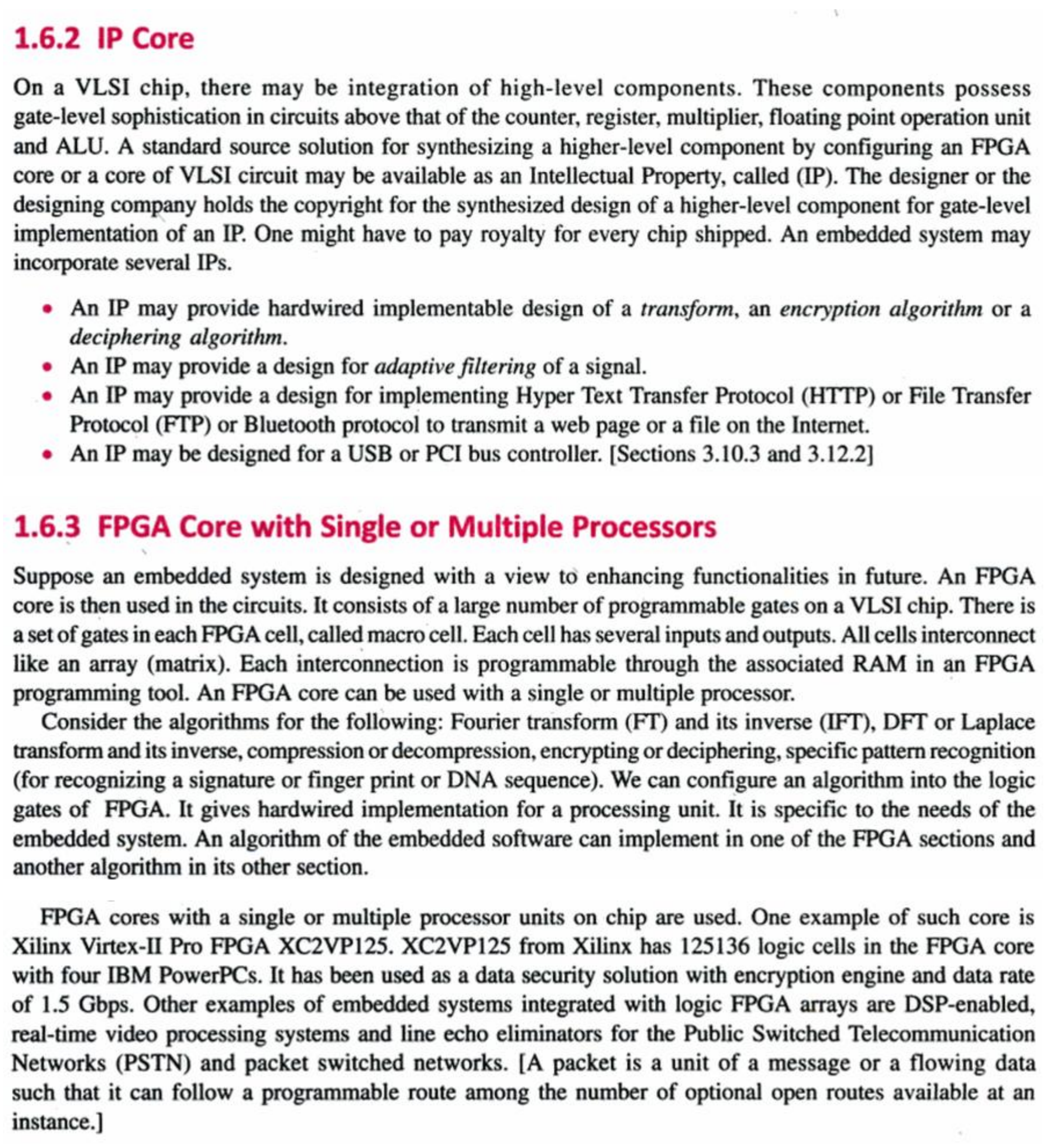

Clock Signal Distribution / Management Subsystem
Sensor Data
Power
Radio Frequency (RF) Signal
Control Signal
Inputs
Standard Interface and Analog / Digital Converter (ADC)
Memory
Micro-processor Design Core
Data and Signal Distribution
Digital Signal Processor (DSP) Design Core
Custom User Block
Standard Interface and Digital / Analog Converter (DAC)
Control Signal
Audio Signal
Intermediate Frequency (IF) Signal
Video Signal
Outputs
Power Distribution / Management Subsystem

Structure of a System on chip(SOC)

- Microcontroller, Microprocessor or DSP cores.
- Memory Blocks – ROM, RAM, EEPROM, Flash
- Clock sources – Local Oscillator, PLL
- Power Circuits – Voltage Regulator, DC-DC converters, etc.
- Peripherals – Counter-timers, real-time timers, etc.
- External Interfaces – USB, Ethernet, UART, RS-232, etc.
- Analog Interfaces – ADCs and DACs
- Blocks are connected by either a proprietary or industry-standard bus.
- Embedded Software

1.6.1 Application Specific IC (ASIC)

ASICs are designed using the VLSI design tools with the 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 an ASIC, a High-level Design Language (HDL) is used].



1.6.2 IP Core

On a VLSI chip, there may be integration of high-level components. These components 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 an FPGA core or a core of VLSI circuit may be available as an Intellectual Property, called (IP). The designer or the designing company holds the copyright for the synthesized design of a higher-level component for gate-level implementation of an IP. One might have to pay royalty for every chip shipped. An embedded system may incorporate several IPs.

- An IP may provide hardwired implementable design of a *transform*, an *encryption algorithm* or a *deciphering algorithm*.
- An IP may provide a design for *adaptive filtering* of a signal.
- An IP may provide a design for implementing Hyper Text Transfer Protocol (HTTP) or File Transfer Protocol (FTP) or Bluetooth protocol to transmit a web page or a file on the Internet.
- An IP may be designed for a USB or PCI bus controller. [Sections 3.10.3 and 3.12.2]

1.6.3 FPGA Core with Single or Multiple Processors

Suppose an embedded system is designed with a view to enhancing functionalities in future. An FPGA core is then used in the circuits. It 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 programmable through the associated RAM in an FPGA programming tool. An FPGA core can be used with a single or multiple processor.

Consider the algorithms for the following: Fourier transform (FT) and its inverse (IFT), DFT or Laplace transform and its inverse, compression or decompression, encrypting or deciphering, specific pattern recognition (for recognizing a signature or finger print or DNA sequence). We can configure an algorithm into the logic gates of FPGA. It gives 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 sections and another algorithm in its other section.

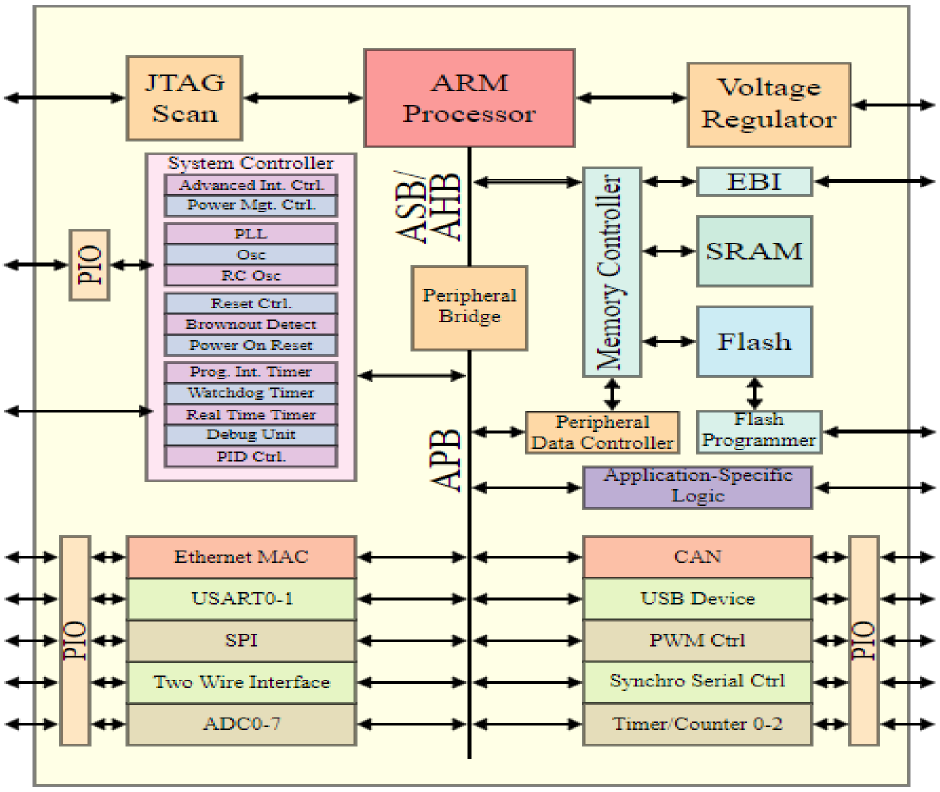
FPGA cores with a single or multiple processor units on chip are used. One example of such core is Xilinx Virtex-II Pro FPGA XC2VP125. XC2VP125 from Xilinx has 125136 logic cells in the FPGA core with four IBM PowerPCs. It has been used as a data security solution with encryption engine and data rate of 1.5 Gbps. Other examples of embedded systems integrated with logic FPGA 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.]

SoC Challenges:

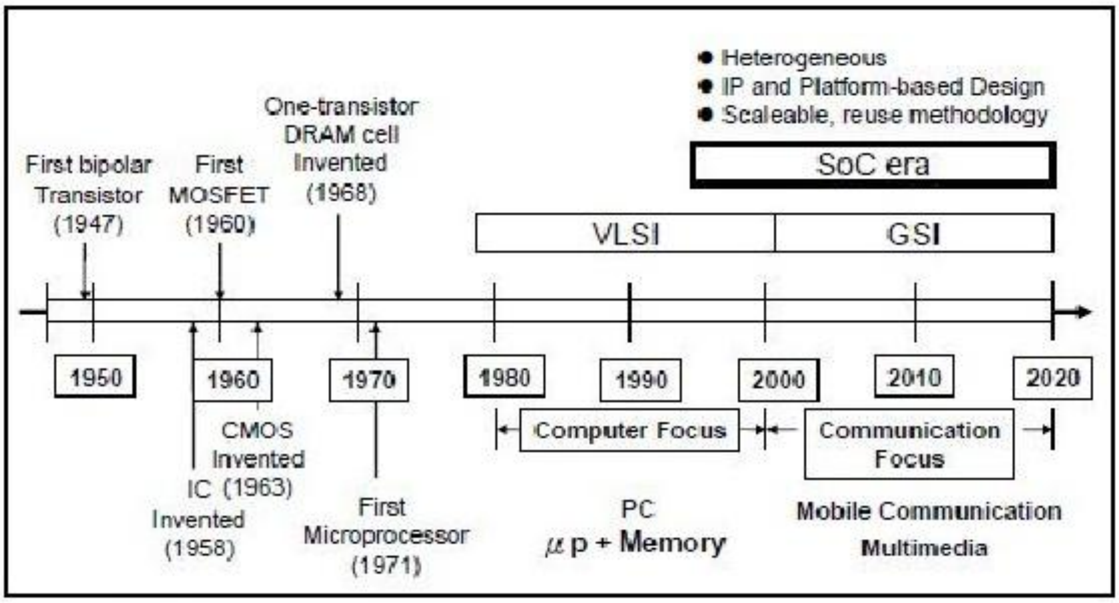
- SoC Designs
  - More complex, more functions, higher gate counts
  - Faster, cheaper, smaller
  - More reliable

- How to handle complexity?
    - System design at multiple abstraction levels
    - Integration of heterogeneous technologies & tools
    - Signal integrity & timing
    - Power management
    - SoC test methodology

System on chip (SOC) based on ARM processor:



Evolution of Semiconductor Device Technology



Benefits of Using SoC

- There are several benefits in integrating a large digital system into a single integrated circuit .
- These include
  - Reduce overall system cost
  - Lower power consumption .
  - Faster circuit operation .
  - More reliable implementation .
  - Smaller physical size .
  - Greater design security .
  - Lower cost per gate .

Major SoC Applications

- Speech Signal Processing .
- Image and Video Signal Processing .
- Information Technologies
  - PC interface (USB, PCI,PCI-Express, IDE,..etc) Computer peripheries (printer control, LCD monitor controller, DVD controller,.etc) .
- Data Communication
  - Wireline Communication: 10/100 Based-T, xDSL, Gigabit Ethernet,.. Etc
  - Wireless communication: BlueTooth, WLAN, 2G/3G/4G, WiMax, UWB, …,etc

1.1.5 Challenges in Embedded Computing System Design

External constraints are one important source of difficulty in embedded system design. Let's consider some important problems that must be taken into account in embedded system design.

How much hardware do we need?
We have a great deal of control over the amount of computing power we apply to our problem. We cannot only select the type of microprocessor used, but also select the amount of memory, the peripheral devices, and more. Since we often must meet both performance deadlines and manufacturing cost constraints, the choice of hardware is important—too little hardware and the system fails to meet its deadlines, too much hardware and it becomes too expensive.

How do we meet deadlines?
The brute force way of meeting a deadline is to speed up the hardware so that the program runs faster. Of course, that makes the system more expensive. It is also entirely possible that increasing the CPU clock rate may not make enough difference to execution time, since the program's speed may be limited by the memory system.

How do we minimize power consumption?
In battery-powered applications, power consumption is extremely important. Even in nonbattery applications, excessive power consumption can increase heat dissipation. One way to make a digital system consume less power is to make it

run more slowly, but naively slowing down the system can obviously lead to missed deadlines. Careful design is required to slow down the noncritical parts of the machine for power consumption while still meeting necessary performance goals.

How do we design for upgradability?
The hardware platform may be used over several product generations, or for several different versions of a product in the same generation, with few or no changes. However, we want to be able to add features by changing software. How can we design a machine that will provide the required performance for software that we haven't yet written?

Does it really work?
Reliability is always important when selling products—customers rightly expect that products they buy will work. Reliability is especially important in some applications, such as safety-critical systems. If we wait until we have a running system and try to eliminate the bugs, we will be too late—we won't find enough bugs, it will be too expensive to fix them, and it will take too long as well. Another set of challenges comes from the characteristics of the components and systems themselves. If workstation programming is like assembling a machine on a bench, then embedded system design is often more like working on a car—cramped, delicate, and difficult. Let's consider some ways in which the nature of embedded computing machines makes their design more difficult.

- *Complex testing:* Exercising an embedded system is generally more difficult than typing in some data. We may have to run a real machine in order to generate the proper data. The timing of data is often important, meaning that we cannot separate the testing of an embedded computer from the machine in which it is embedded.

- *Limited observability and controllability:* Embedded computing systems usually do not come with keyboards and screens. This makes it more difficult to see what is going on and to affect the system's operation. We may be forced to watch the values of electrical signals on the microprocessor bus, for example, to know what is going on inside the system. Moreover, in real-time applications we may not be able to easily stop the system to see what is going on inside.

- *Restricted development environments:* The development environments for embedded systems (the tools used to develop software and hardware) are often much more limited than those available for PCs and workstations. We generally compile code on one type of machine, such as a PC, and download it onto the embedded system. To debug the code, we must usually rely on programs that run on the PC or workstation and then look inside the embedded system.

1.8.1 Design Metrics

A design process takes into account design metrics. There are several design metrics for an embedded system,

A design metric is a measurable feature of the system's performance, cost, time for implementation and safety etc.

Design metrics typically compete with one another : Improving one often leads to worsening of another.

e.g. If the implementation's size is reduced, the implementation's performance may suffer.

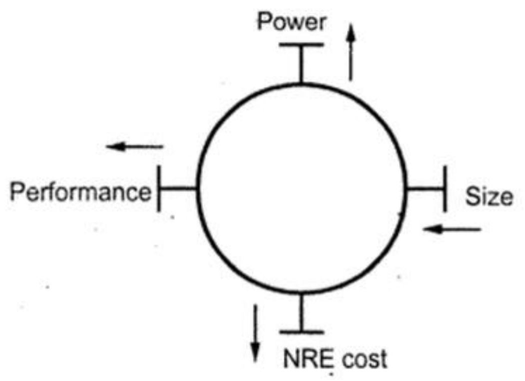


Fig. 1.4 Design metric competition-improving one may worsen others

Following design metrics are generally taken into account while designing an embedded systems.

i) NRE (Non-recurring Engineering) cost :

It is the one-time monetary cost of designing the system. Once the system is designed, any number of units can be manufactured without incurring any additional design cost; hence the name non-recurring.

Suppose three technologies are available for use in a particular product. Assume that implementing the product using technology A would result in an NRE cost of $ 1500 and unit cost of $ 100, technology B would have an NRE cost of $ 2500 and unit cost of $ 30 and technology C would have an NRE cost of $ 90,000 and unit cost of $ 2. Ignoring all other design metrics like time-to-market, the best technology choice will depend on the number of units we plan to produce.

ii) Unit cost :

It is the monetary cost of manufacturing each copy of the system, excluding NRE cost.

iii) Size :

It is the physical space required by the system, often measured in bytes for software and gates or transistors for hardware.

iv) Performance :

It is the execution time of the system.

v) Power consumption :

It is the amount of power consumed by the system, which may determine the lifetime of a battery or the cooling requirements of IC, since more power means more heat.

vi) Flexibility :

It is the ability to change the functionality of the system without incurring heavy NRE cost. Typically software is considered very flexible.

vii) Time-to-prototype :

It is the time needed to build a working version of the system, which may be bigger or more expensive than the final system implementation, but it can be used to verify the system's usefulness and correctness and to refine the system's functionality.

vii) Time-to-market

It is the time required to develop a system to the point that it can be released and sold to customers. This design metric has become especially demanding in recent years. Introducing an embedded system to the marketplace early can make a big difference in the system's profitability. The main contributors are design time, manufacturing time and testing time.

ix) Maintainability

It is the ability to modify the system after its initial release, especially by designers who did not originally design the system.

x) Correctness

It is the measure of the confidence that we have implemented the system's functionality correctly. The functionality can be checked throughout the process of designing the system and test circuitry can be inserted to check that manufacturing was correct.

xi) Safety :

It is the probability that the system will not cause harm.

1.2 THE EMBEDDED SYSTEM DESIGN PROCESS

This section provides an overview of the embedded system design process aimed at two objectives. First, it will give us an introduction to the various steps in embedded system design before we delve into them in more detail. Second, it will allow us to consider the design ***methodology*** itself. A design methodology is important for three reasons. First, it allows us to keep a scorecard on a design to ensure that we have done everything we need to do, such as optimizing ***performance*** or performing functional tests. Second, it allows us to develop computer-aided design tools. Developing a single program that takes in a concept for an embedded system and emits a completed design would be a daunting task, but by first breaking the process into manageable steps, we can work on automating (or at least semiautomating) the steps one at a time. Third, a design methodology makes it much easier for members of a design team to communicate. By defining the overall process, team members can more easily understand what they are supposed to do, what they should receive from other team members at certain times, and what they are to hand off when they complete their assigned steps. Since most embedded systems are designed by teams, coordination is perhaps the most important role of a well-defined design methodology.

1.8.2 Abstraction of Steps in the Design Process

A design process is called bottom-to-top design if it builds by starting from the components. A design process is called top-to-down design if it first starts with abstraction of the process and then after abstraction the details are created. Top-to-down design approach is the most favoured approach. The following lists the five levels of abstraction from top to bottom in the design process:

In this top–down view, we start with the system ***requirements***. In the next step,

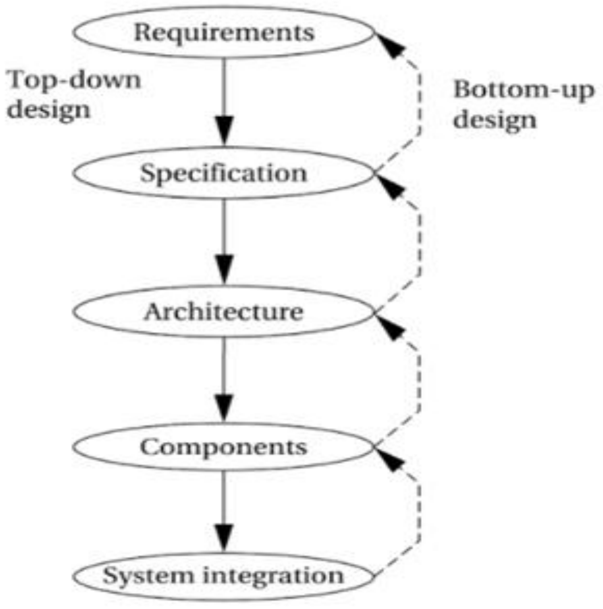


FIGURE 1.1

Major levels of abstraction in the design process.

(1) **Requirements:** Definition and analysis of system requirement. It is only by a complete clarity of the required *purpose, inputs, outputs, functioning, design metrics* (Table 1.8) and *validation requirements* for finally developed systems specifications that a well designed system can be created. There has to be consistency in the requirements.

Requirements may be ***functional*** or ***nonfunctional***. We must of course capture the basic functions of the embedded system, but functional description is often not sufficient. Typical nonfunctional requirements include:

- *Performance:* The speed of the system is often a major consideration both for the usability of the system and for its ultimate cost. As we have noted, performance may be a combination of soft performance metrics such as approximate time to perform a user-level function and hard deadlines by which a particular operation must be completed.
- *Cost:* The target cost or purchase price for the system is almost always a consideration. Cost typically has two major components: ***manufacturing cost*** includes the cost of components and assembly; ***nonrecurring engineering*** **(NRE)** costs include the personnel and other costs of designing the system.
- *Physical size and weight:* The physical aspects of the final system can vary greatly depending upon the application. An industrial control system for an assembly line may be designed to fit into a standard-size rack with no strict limitations on weight. A handheld device typically has tight requirements on both size and weight that can ripple through the entire system design.
- *Power consumption:* Power, of course, is important in battery-powered systems and is often important in other applications as well. Power can be specified in the requirements stage in terms of battery life—the customer is unlikely to be able to describe the allowable wattage.

Validating a set of requirements is ultimately a psychological task since it requires understanding both what people want and how they communicate those needs. One good way to refine at least the user interface portion of a system's requirements is to build a ***mock-up***. The mock-up may use canned data to simulate functionality in a restricted demonstration, and it may be executed on a PC or a workstation. But it should give the customer a good idea of how the system will be used and how the user can react to it. Physical, nonfunctional models of devices can also give customers a better idea of characteristics such as size and weight.

Name

Purpose

Inputs

Outputs

Functions

Performance

Manufacturing cost

Power

Physical size and weight

FIGURE 1.2

Sample requirements form.

Requirements analysis for big systems can be complex and time consuming. However, capturing a relatively small amount of information in a clear, simple format is a good start toward understanding system requirements. To introduce the discipline of requirements analysis as part of system design, we will use a simple requirements methodology.

Figure 1.2 shows a sample ***requirements form*** that can be filled out at the start of the project. We can use the form as a checklist in considering the basic characteristics of the system. Let's consider the entries in the form:

- *Name:* This is simple but helpful. Giving a name to the project not only simplifies talking about it to other people but can also crystallize the purpose of the machine.
- *Purpose:* This should be a brief one- or two-line description of what the system is supposed to do. If you can't describe the essence of your system in one or two lines, chances are that you don't understand it well enough.
- *Inputs and outputs:* These two entries are more complex than they seem. The inputs and outputs to the system encompass a wealth of detail:
  - — *Types of data:* Analog electronic signals? Digital data? Mechanical inputs?
  - — *Data characteristics:* Periodically arriving data, such as digital audio samples? Occasional user inputs? How many bits per data element?
  - — *Types of I/O devices:* Buttons? Analog/digital converters? Video displays?
- *Functions:* This is a more detailed description of what the system does. A good way to approach this is to work from the inputs to the outputs: When the system receives an input, what does it do? How do user interface inputs affect these functions? How do different functions interact?

- *Performance:* Many embedded computing systems spend at least some time controlling physical devices or processing data coming from the physical world. In most of these cases, the computations must be performed within a certain time frame. It is essential that the performance requirements be identified early since they must be carefully measured during implementation to ensure that the system works properly.
- *Manufacturing cost:* This includes primarily the cost of the hardware components. Even if you don't know exactly how much you can afford to spend on system components, you should have some idea of the eventual cost range. Cost has a substantial influence on architecture: A machine that is meant to sell at \$10 most likely has a very different internal structure than a \$100 system.
- *Power:* Similarly, you may have only a rough idea of how much power the system can consume, but a little information can go a long way. Typically, the most important decision is whether the machine will be battery powered or plugged into the wall. Battery-powered machines must be much more careful about how they spend energy.
- *Physical size and weight:* You should give some indication of the physical size of the system to help guide certain architectural decisions. A desktop machine has much more flexibility in the components used than, for example, a lapel-mounted voice recorder.

After writing the requirements, you should check them for internal consistency: Did you forget to assign a function to an input or output? Did you consider all the modes in which you want the system to operate? Did you place an unrealistic number of features into a battery-powered, low-cost machine?

To practice the capture of system requirements, Example 1.1 creates the requirements for a GPS moving map system.

Example 1.1

Requirements analysis of a GPS moving map

The moving map is a handheld device that displays for the user a map of the terrain around the user's current position; the map display changes as the user and the map device change position. The moving map obtains its position from the GPS, a satellite-based navigation system. The moving map display might look something like the following figure.

- *Functionality:* This system is designed for highway driving and similar uses, not nautical or aviation uses that require more specialized databases and functions. The system should show major roads and other landmarks available in standard topographic databases.

- *User interface:* The screen should have at least 400 × 600 pixel resolution. The device should be controlled by no more than three buttons. A menu system should pop up on the screen when buttons are pressed to allow the user to make selections to control the system.
- *Performance:* The map should scroll smoothly. Upon power-up, a display should take no more than one second to appear, and the system should be able to verify its position and display the current map within 15 s.
- *Cost:* The selling cost (street price) of the unit should be no more than $100.
- *Physical size and weight:* The device should fit comfortably in the palm of the hand.
- *Power consumption:* The device should run for at least eight hours on four AA batteries.

Note that many of these requirements are not specified in engineering units—for example, physical size is measured relative to a hand, not in centimeters. Although these requirements must ultimately be translated into something that can be used by the designers, keeping a record of what the customer wants can help to resolve questions about the specification that may crop up later during design.

Based on this discussion, let's write a requirements chart for our moving map system:

Name	GPS moving map
Purpose	Consumer-grade moving map for driving use
Inputs	Power button, two control buttons
Outputs	Back-lit LCD display 400 × 600
Functions	Uses 5-receiver GPS system; three user-selectable resolutions; always displays current latitude and longitude
Performance	Updates screen within 0.25 seconds upon movement
Manufacturing cost	$30
Power	100 mW
Physical size and weight	No more than 2" × 6, " 12 ounces

This chart adds some requirements in engineering terms that will be of use to the designers. For example, it provides actual dimensions of the device. The manufacturing cost was derived from the selling price by using a simple rule of thumb: The selling price is four to five times the ***cost of goods sold*** (the total of all the component costs).

(2) *Specifications:* Clear specifications of the required system are must. Specifications need to be precise. Specifications guide customer expectations from the product. They also guide system architecture. The designer needs specifications for (i) hardware, for example, peripherals, devices processor and memory specifications, (ii) data types and processing specifications, (iii) expected system behaviour specifications, (iv) constraints of design, and (v) expected life cycle specifications. Process specifications are analysed by making lists of inputs on events, outputs on events and how the processes activate on each event (interrupt).

The specification is more precise—it serves as the contract between the customer and the architects. As such, the specification must be carefully written so that it accurately reflects the customer's requirements and does so in a way that can be clearly followed during design.

Specification is probably the least familiar phase of this methodology for neophyte designers, but it is essential to creating working systems with a minimum of designer effort. Designers who lack a clear idea of what they want to build when they begin typically make faulty assumptions early in the process that aren't obvious until they have a working system. At that point, the only solution is to take the machine apart, throw away some of it, and start again. Not only does this take a lot of extra time, the resulting system is also very likely to be inelegant, kludgey, and bug-ridden.

The specification should be understandable enough so that someone can verify that it meets system requirements and overall expectations of the customer. It should also be unambiguous enough that designers know what they need to build. Designers can run into several different types of problems caused by unclear specifications. If the behavior of some feature in a particular situation is unclear from the specification, the designer may implement the wrong functionality. If global characteristics of the specification are wrong or incomplete, the overall system architecture derived from the specification may be inadequate to meet the needs of implementation.

A specification of the GPS system would include several components:

- Data received from the GPS satellite constellation.
- Map data.
- User interface.
- Operations that must be performed to satisfy customer requests.
- Background actions required to keep the system running, such as operating the GPS receiver.

(3) ***Architecture:*** Data modeling designs of attributes of data structure, data flow graphs (Section 6.2), program models (Section 6.1), software architecture layers and hardware architecture are defined. Software architectural layers are as follows:

1. The first layer is an architectural design. Here, a design for system architecture is developed. The question arises as to how the different elements—data structures, databases, algorithms, control functions, state transition functions, process, data and program flow—are to be organised.
2. The second layer consists of data-design. Questions at this stage are as follows. What design of data structures and databases would be most appropriate for the given problem? Whether data organised as a tree- like structure will be appropriate? What will be the design of the components in the data? [For example, video information will have two components, image and sound.]

3. The third layer consists of interface design. Important questions at this stage are as follows. What shall be the interfaces to integrate the components? What is the design for system integration? What shall be design of interfaces used for taking inputs from the data objects, structures and databases and for delivering outputs? What will be the port structure for receiving inputs and transmitting outputs?

The specification does not say how the system does things, only what the system does. Describing how the system implements those functions is the purpose of the architecture. The architecture is a plan for the overall structure of the system that will be used later to design the components that make up the architecture. The creation of the architecture is the first phase of what many designers think of as design.

To understand what an architectural description is, let's look at a sample architecture for the moving map of Example 1.1. Figure 1.3 shows a sample system architecture in the form of a ***block diagram*** that shows major operations and data flows among them.

This block diagram is still quite abstract—we have not yet specified which operations will be performed by software running on a CPU, what will be done by special-purpose hardware, and so on. The diagram does, however, go a long way toward describing how to implement the functions described in the specification. We clearly see, for example, that we need to search the topographic database and to render (i.e., draw) the results for the display. We have chosen to separate those functions so that we can potentially do them in parallel—performing rendering separately from searching the database may help us update the screen more fluidly.

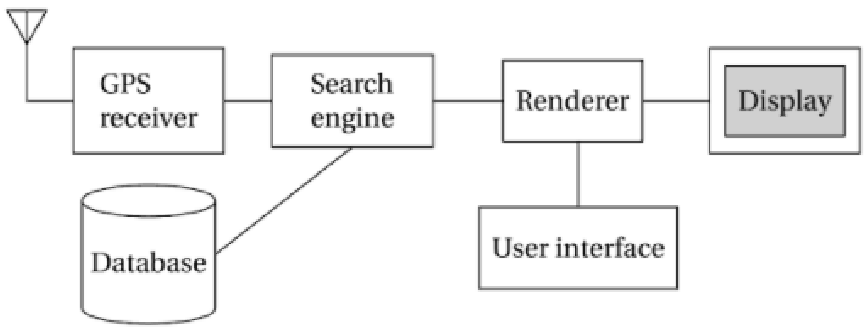


FIGURE 1.3

Block diagram for the moving map.

Only after we have designed an initial architecture that is not biased toward too many implementation details should we refine that system block diagram into two block diagrams: one for hardware and another for software. These two more refined block diagrams are shown in Figure 1.4. The hardware block diagram clearly shows that we have one central CPU surrounded by memory and I/O devices. In particular, we have chosen to use two memories: a frame buffer for the pixels to be displayed and a separate program/data memory for general use by the CPU. The software block diagram fairly closely follows the system block diagram, but we have added a timer to control when we read the buttons on the user interface and render data onto the screen. To have a truly complete architectural description, we require more detail, such as where units in the software block diagram will be executed in the hardware block diagram and when operations will be performed in time.

Architectural descriptions must be designed to satisfy both functional and nonfunctional requirements. Not only must all the required functions be present, but we must meet cost, speed, power, and other nonfunctional constraints. Starting out with a system architecture and refining that to hardware and software architectures

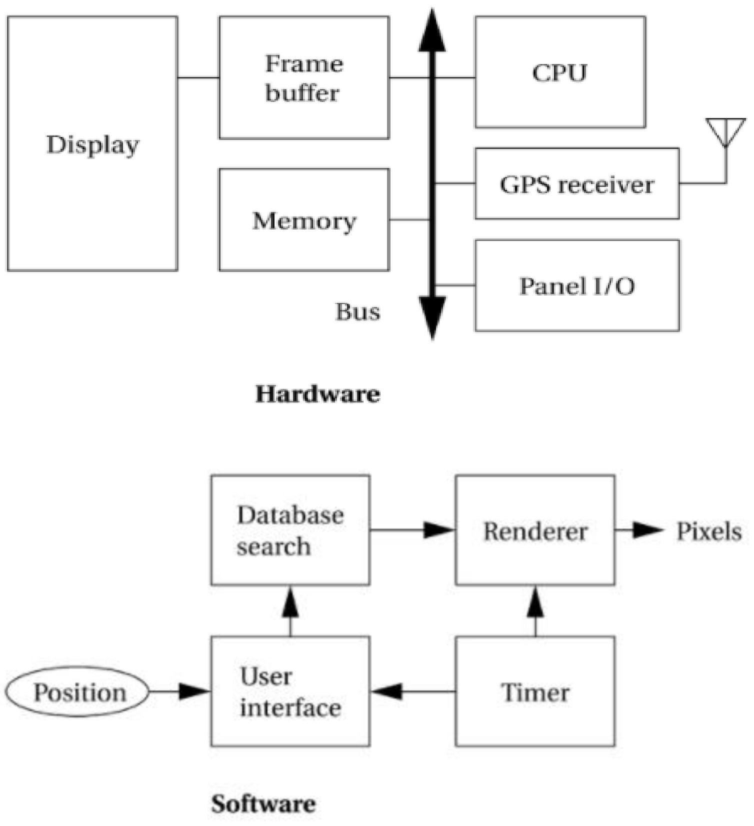


FIGURE 1.4

Hardware and software architectures for the moving map.

is one good way to ensure that we meet all specifications: We can concentrate on the functional elements in the system block diagram, and then consider the nonfunctional constraints when creating the hardware and software architectures.

How do we know that our hardware and software architectures in fact meet constraints on speed, cost, and so on? We must somehow be able to estimate the properties of the components of the block diagrams, such as the search and rendering functions in the moving map system. Accurate estimation derives in part from experience, both general design experience and particular experience with similar systems. However, we can sometimes create simplified models to help us make more accurate estimates. Sound estimates of all nonfunctional constraints during the architecture phase are crucial, since decisions based on bad data will show up during the final phases of design, indicating that we did not, in fact, meet the specification.

(4) **Components:** The fourth layer is a component level design. The question at this stage is as follows. What shall be the design of each component? There is an additional requirement in the design of embedded systems, that each component should be optimised for memory usage and power dissipation. Components of hardware, processes, interfaces and algorithms. The following lists the common hardware components:

1. Processor, ASIP and single purpose processors in the system
2. Memory RAM, ROM or internal and external flash or secondary memory in the system
3. Peripherals and devices internal and external to the system
4. Ports and buses in the system
5. Power source or battery in the system

The architectural description tells us what components we need. The component design effort builds those components in conformance to the architecture and specification. The components will in general include both hardware—FPGAs, boards, and so on—and software modules.

Some of the components will be ready-made. The CPU, for example, will be a standard component in almost all cases, as will memory chips and many other components. In the moving map, the GPS receiver is a good example of a specialized component that will nonetheless be a predesigned, standard component. We can also make use of standard software modules. One good example is the topographic database. Standard topographic databases exist, and you probably want to use standard routines to access the database—not only is the data in a predefined format, but it is highly compressed to save storage. Using standard software for these access functions not only saves us design time, but it may give us a faster implementation for specialized functions such as the data decompression phase.

(5) **System Integration:** Built components are integrated in the system. Components may work fine independently, but when integrated may not fullfil the design metrics. The system is made to function and validated. Appropriate tests are chosen. Debugging tools are used to correct erroneous functioning.

Each component and its interface system is integrated after the design stage. Program implementation is in a language and may use an integrated development environment (IDE), and source code engineering tools, which should follow the model, software architecture and design specifications. Program simplicity should be maintained during the implementation process.