UNIT – V

Principles of Embedded Systems: Introduction -Embedded systems description, definition, design considerations and requirements - Overview of Embedded system Architecture (CISC and RISC) -Categories of Embedded Systems -Embedded processor selection and tradeoffs.

5.1:INTRODUCTION -EMBEDDED SYSTEMS DESCRIPTION

Embedded systems

An embedded system is a complex system also known as an integrated system; It has software embedded into hardware (*also known as Firmware*) to perform specific tasks or a single task. It typically contains one or more microprocessors for executing a set of programs defined at design time and stored in memory. Which makes a system dedicated to a specific function, within a more extensive system. It is a low-power Microcontroller/Microprocessor based computer system, which dedicated to providing specific functionality.

5.2:DEFINITION:

An Electronic/Electro mechanical system which is designed to perform a specific function and is a combination of both hardware and firmware (Software)

E.g. Electronic Toys, Mobile Handsets, Washing Machines, Air Conditioners, Automotive Control Units, Set Top Box, DVD Player etc

Embedded Systems Vs General Computing Systems

General Purpose Computing System	Embedded System
A system which is a combination of generic hardware and General Purpose Operating System for executing a variety of applications	A system which is a combination of special purpose hardware and embedded OS for executing a specific set of applications
Contain a General Purpose Operating System	May or may not contain an operating system for functioning
Applications are alterable (programmable) by user (It is possible for the end user to re-install the Operating System, and add or remove user applications)	The firmware of the embedded system is pre-programmed and it is non-alterable by end-user
Performance is the key deciding factor on the selection of the system. Always „Faster is Better	Application specific requirements (like performance, power requirements, memory usage etc) are the key deciding factors
Less/not at all tailored towards reduced operating power requirements, options for different levels of power management	Highly tailored to take advantage of the power saving modes supported by hardware and Operating System
Response requirements are not time critical	For certain category of embedded systems like mission critical systems, the response time requirement is highly critical
Need not be deterministic in execution behavior	Execution behavior is deterministic for certain type of embedded systems like „Hard Real Time" systems

Characteristics of an Embedded System

An embedded system is a hardware and software package that combines four main characteristics:

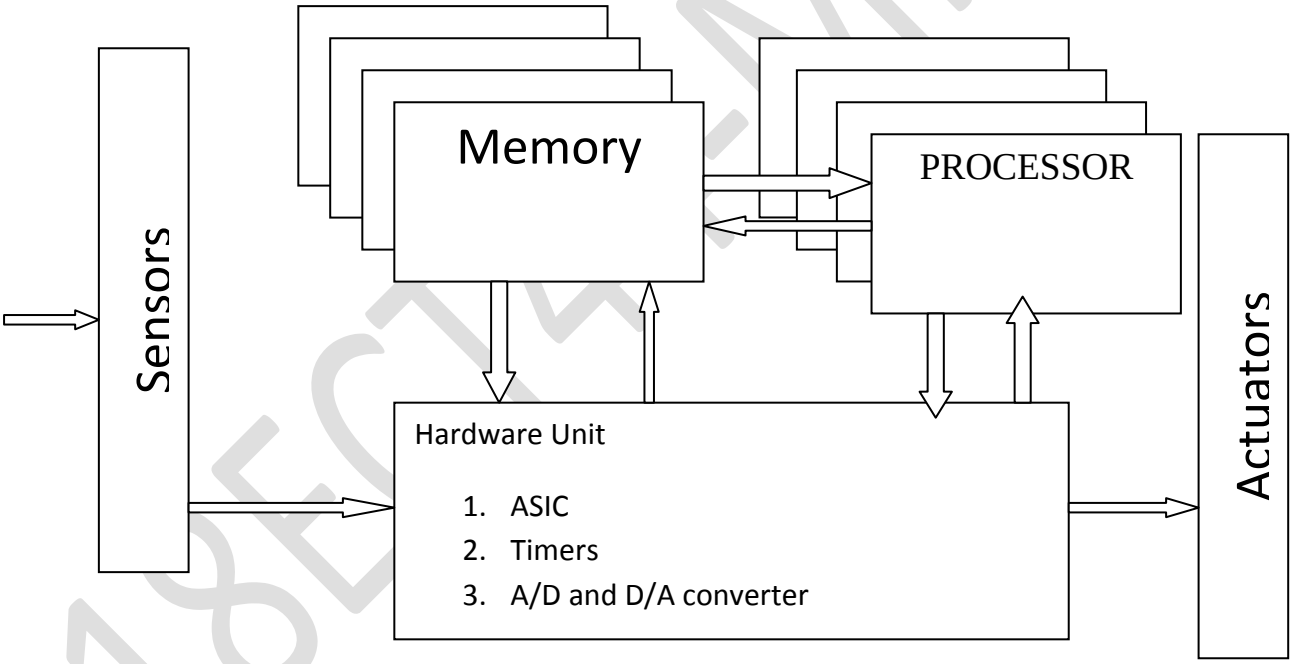
1. **Autonomous**: it must fulfill its mission for long periods and without human intervention with very little power consumption.
2. **Real-time**: The response times of these systems are as important as deliver accurate results within set deadlines and are also deficient maintenance with low cost.
3. It is specialized in a specific task.
4. Its resources are limited; in other words, we will seek to optimize its size, its consumption and its memories to reduce the overall cost.

Major Application Areas of Embedded Systems:

- **Consumer Electronics:** Camcorders, Cameras etc.
- **Household Appliances:** Television, DVD players, washing machine, Fridge, Microwave Oven etc
- **Home Automation and Security Systems:** Air conditioners, sprinklers, Intruder detection alarms, Closed Circuit Television Cameras, Fire alarms etc.
- **Automotive Industry:** Anti-lock breaking systems (ABS), Engine Control, Ignition Systems, Automatic Navigation Systems etc.
- **Telecom:** Cellular Telephones, Telephone switches, Handset Multimedia Applications etc.
- **Computer Peripherals:** Printers, Scanners, Fax machines etc.
- **Computer Networking Systems:** Network Routers, Switches, Hubs, Firewalls etc.
- **Health Care:** Different Kinds of Scanners, EEG, ECG Machines etc.
- **Measurement & Instrumentation:** Digital multi meters, Digital CROs, Logic Analyzers PLC systems etc.
- **Banking & Retail:** Automatic Teller Machines (ATM) and Currency counters, Point of Sales(POS)
- **Card Readers:** Barcode, Smart Card Readers, Hand held Devices etc.

Parts of Embedded system

A typical embedded system has the following



Part of embedded system

- Actuators - These are mechanical components (e.g., valve, switches).
- Sensors - input data (e.g., accelerometer, temperature sensor, light sensor).
- Memory - These are the storage devices used to store input, output and processing data.
- Hardware units - these are used for data conversion and connecting a processor with sensors and actuators.
- Processor - These are used for processing the data and decision-making.

These days all the above components are done with a single-chip implementation known as **system on a chip**.

5.3:DESIGN CONSIDERATIONS AND REQUIREMENTS

The Embedded Design Life Cycle

Design considerations and requirements:

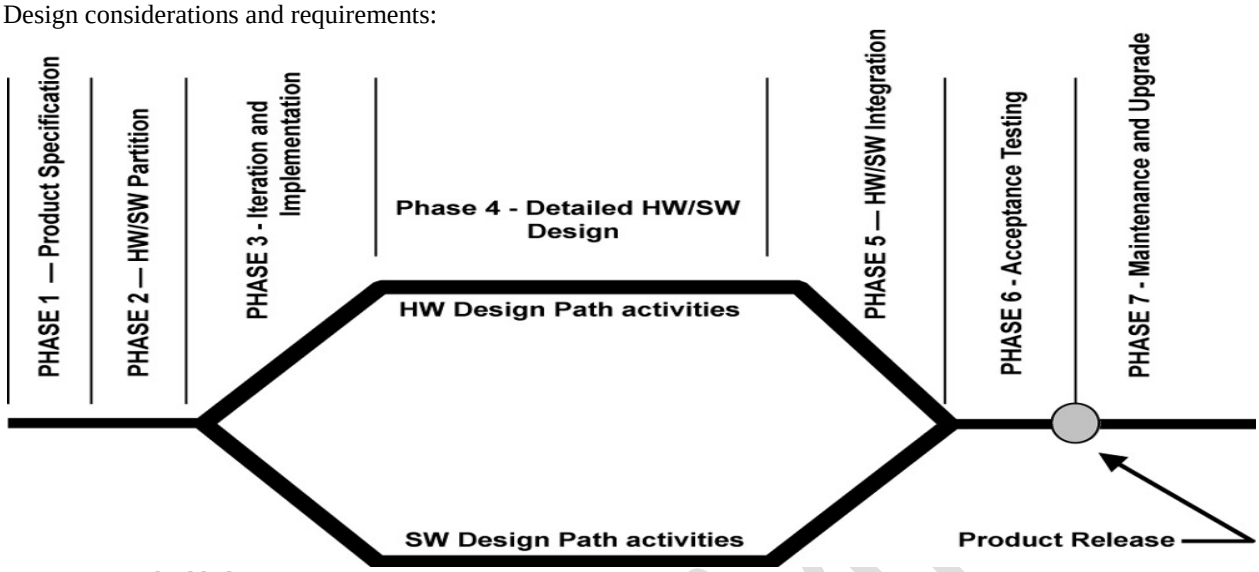


Figure 1.1 Embedded Design Process

Time flows from the left and proceeds through seven phases:

- Product specification
- Partitioning of the design into its software and hardware components
- Iteration and refinement of the partitioning
- Independent hardware and software design tasks
- Integration of the hardware and software components
- Product testing and release
- On-going maintenance and upgrading

The embedded design process is not as simple as Figure 1.1 depicts. A considerable amount of iteration and optimization occurs within phases and between phases it can be argued that including the choice of the microprocessor and some of the other key elements of a design in the specification phase is the correct approach. For example, if your existing code base is written for the 80X86 processor family, it's entirely legitimate to require that the next design also be able to leverage this code base.

Product Specification

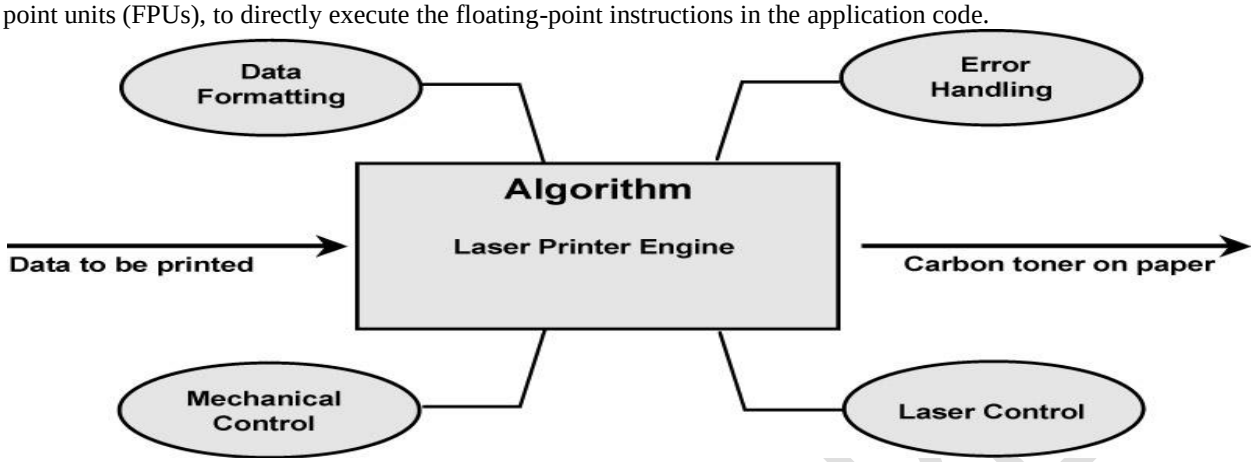
The ideal research team is three or four people, usually a marketing or sales engineer and two or three R&D types. Each member of the team has a specific role during the visit. Often, these roles switch among the team members so each has an opportunity to try all the roles. The team prepares for the visit by developing a questionnaire to use to keep the interviews flowing smoothly. In general, the questionnaire consists of a set of open-ended questions that the team members fill in as they speak with the customers. For several customer visits, my research team spent more than two weeks preparing and refining the questionnaire.

Participating in the customer research tour as an R&D engineer on the project has a side benefit. Not only do you have a design specification (hopefully) against which to design, you also have a picture in your mind's eye of your team's ultimate objective. A little voice in your ear now biases your endless design decisions toward the common goals of the design team. This extra insight into the product specifications can significantly impact the success of the project. A senior engineering manager studied projects within her company that were successful not only in the marketplace but also in the execution of the product development process

Hardware/Software Partitioning

Embedded design will involve both hardware and software components, someone must decide which portion of the problem will be solved in hardware and which in software. This choice is called the "partitioning decision." Application developers, who normally work with pre-defined hardware resources, may have difficulty adjusting to the notion that the hardware can be enhanced to address any arbitrary portion of the problem. However, they've probably already encountered examples of such a hardware/software tradeoff. For example, in the early days of the

PC (i.e., before the introduction of the 80486 processor), the 8086, 80286, and 80386 CPUs didn't have an on-chip floating-point processing unit. These processors required companion devices, the 8087, 80287, and 80387 floating-point units (FPUs), to directly execute the floating-point instructions in the application code.



These two very different design philosophies are successfully applied to the design of laser printers in two real-world companies today. One company has highly developed its ability to fine-tune the processor performance to minimize the need for specialized hardware. Conversely, the other company thinks nothing of throwing a team of ASIC designers at the problem. Both companies have competitive products but implement a different design strategy for partitioning the design into hardware and software components.

The partitioning decision is a complex optimization problem. Many embedded system designs are required to be

i. Price sensitive
ii. Leading-edge performers
iii. Non-standard
iv. Market competitive
v. Proprietary

Iteration and Implementation

The iteration and implementation part of the process represents a somewhat blurred area between implementation and hardware/software partitioning in which the hardware and software paths diverge. This phase represents the early design work before the hardware and software teams build "the wall" between them. The design is still very fluid in this phase. Even though major blocks might be partitioned between the hardware components and the software components, plenty of leeway remains to move these boundaries as more of the design constraints are understood and modeled.. The hardware designers might be using simulation tools, such as architectural simulators, to model the performance of the processor and memory systems. The software designers are probably running code benchmarks on self-contained, single-board computers that use the target micro processor. These single-board computers are often referred to as evaluation boards because they evaluate the performance of the microprocessor by running test code on it. The evaluation board also provides a convenient software design and debug environment until the real system hardware becomes available.

Hardware/Software Integration

The hardware/software integration phase of the development cycle must have special tools and methods to manage the complexity. The process of integrating embedded software and hardware is an exercise in debugging and discovery.

Big Endian/Little Endian Problem

One of my favorite integration discoveries is the "little endian/big endian" syndrome. The hardware designer assumes big endian organization, and the software designer assumes little endian byte order. What makes this a classic example of an interface and integration error is that both the software and hardware could be correct in isolation but fail when integrated because the "endianness" of the interface is misunderstood.Suppose, for example that a serial port is designed for an ASIC with a 16-bit I/O bus. The port is memory mapped at address 0x400000. Eight bits of the word are the data portion of the port, and the other eight bits are the status portion of the port. Even though the hardware designer might specify what bits are status and what bits are data, the software designer could easily assign the wrong port address if writes to the port are done as byte accesses .

Figure 1.2: An example of the endianness problem in I/O addressing. If byte addressing is used and the big endian model is assumed, then the algorithm should check the status at address 0x400001. Data should be read from and written to address 0x400000. If the little endian memory model is assumed, then the reverse is true. If 16-bit addressing is used, i.e., the port is declared as unsigned short int * io_port ; then the endianness ambiguity problem goes away. This means that the software might become more complex because the developer will need to do bit manipulation in order to read and write data, thus making the algorithm more complex. The Holy Grail of embedded system design is to combine the first hardware prototype, the application software, the driver code, and the operating system software together with a pinch of optimism and to have the design work perfectly out of the chute. No green wires on the PC board, no "dead bugs," no no redesigning the ASICs or Field Programmable Gate Arrays (FPGA), and no rewriting the software.

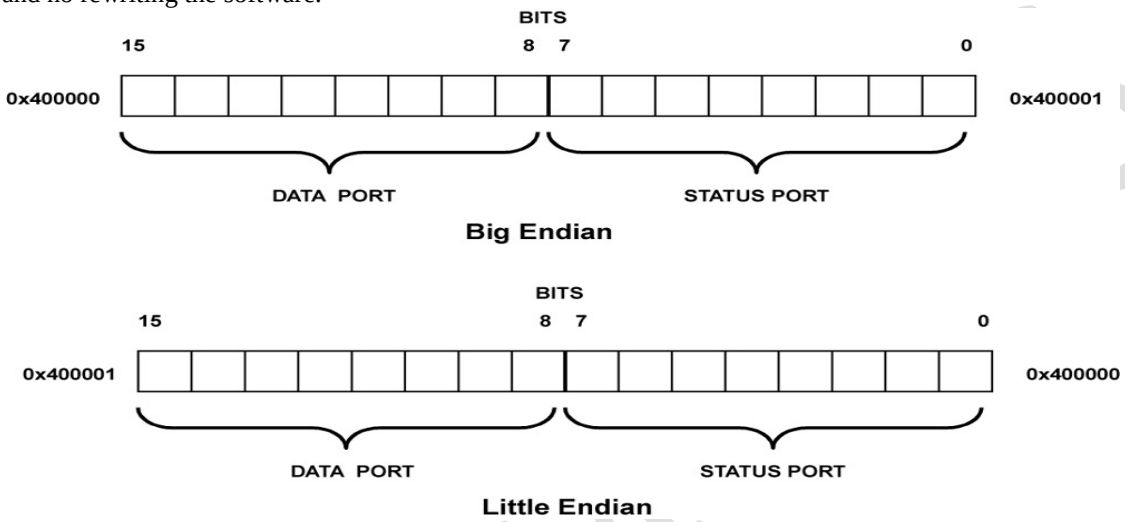


Figure 1.2: An example of the endianness problem in I/O addressing

For one thing, the real-time nature of embedded systems leads to highly complex, nondeterministic behavior that can only be analyzed as it occurs. Attempting to accurately model or simulate the behavior can take much longer than the usable lifetime of the product being developed.

Debugging an Embedded System

debugging an embedded system is similar to debugging a host based application. If the target system contains an available communications channel to the host computer, the debugger can exist as two pieces: a debug kernel in the target system and a host application that communicates with it and manages the source database and symbol tables.

Many embedded systems are impossible to debug unless they are operating at full speed. Running an embedded program under a debugger can slow the program down by one or more orders of magnitude. In most cases, scaling all the real-time dependencies back so that the debugger becomes effective is much more work than just using the correct tools to debug at full speed. Manufacturers of embedded microprocessors also realize the difficulty of controlling these variables, so they've provided on-chip hooks to assist in the debugging of embedded systems containing their processors. Most designers won't even consider using a microprocessor in an embedded application unless the silicon manufacturer can demonstrate a complete tool chain for designing and debugging its silicon. In general, there are three requirements for debugging an embedded or real-time system:

- Run control — The ability to start, stop, peak, and poke the processor and memory.
- Memory substitution — Replacing ROM-based memory with RAM for rapid and easy code download, debug, and repair cycles.
- Real-time analysis — Following code flow in real time with real-time trace analysis.

For many embedded systems, it is necessary also to integrate a commercial or inhouse real-time operating system (RTOS) into the hardware and application software. This integration presents its own set of problems (more variables); the underlying behavior of the operating system is often hidden from the designers because it is obtained as object code from the vendor, which means these bugs are now masked by the RTOS and that another special tool must be used. This tool is usually available from the RTOS vendor (for a price) and is indispensable for debugging the system with the RTOS present. The added complexity doesn't change the three requirements previously listed; it

just makes them more complex. Add the phrase "and be RTOS aware" to each of the three listed requirements, and they would be equally valid for a system containing a RTOS.

Product Testing and Release

Product testing takes on special significance when the performance of the embedded system has life or death consequences attached.

The testing and reliability requirements for an embedded system are much more stringent than the vast majority of desktop applications.

However, testing is more than making sure the software doesn't crash at a critical moment, although it is by no means an insignificant consideration. Because embedded systems usually have extremely tight design margins to meet cost goals, testing must determine whether the system is performing close to its optimal capabilities. This is especially true if the code is written in a high-level language and the design team consists of many developers. Many desktop applications have small memory leaks. Presumably, if the application ran long enough, the PC would run out of heap space, and the computer would crash. However, on a desktop machine with 64MB of RAM and virtual swap space, this is unlikely to be a problem. On the other side, in an embedded system, running continuously for weeks at a time, even a small memory leak is potentially disastrous.

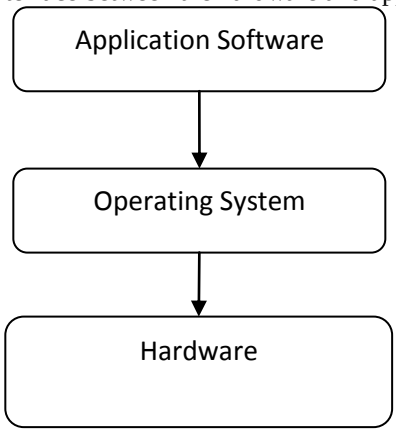
Maintaining and Upgrading Existing Products

The embedded system tool community has made almost no effort to develop tools specifically targeted to products already in service.

The majority of embedded system designers (around 60 percent) maintain and upgrade existing products, rather than design new products. Most of these engineers were not members of the original design team for a particular product, so they must rely on only their experience, their skills, the existing documentation, and the old product to understand the original design well enough to maintain and improve it.

5.4 OVERVIEW OF EMBEDDED SYSTEMS ARCHITECTURE:

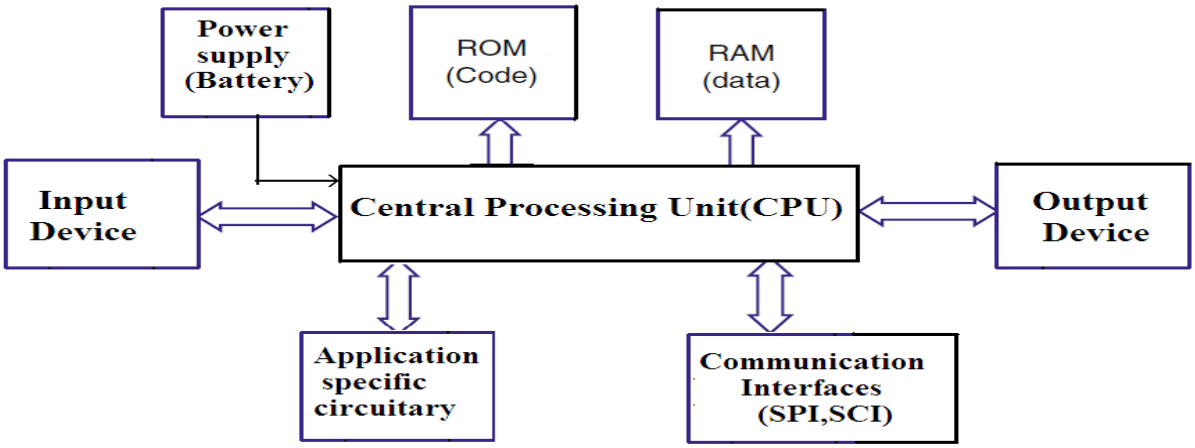
- Every embedded system consists of customer-built hardware components supported by a Central Processing Unit (CPU), which is the heart of a microprocessor (μP) or microcontroller (μC).
- A microcontroller is an integrated chip which comes with built-in memory, I/O ports, timers, and other components.
- Most embedded systems are built on microcontrollers, which run faster than a custom-built system with a microprocessor, because all components are integrated within a single chip.
- Operating system plays an important role in most of the embedded systems. But all the embedded systems do not use the operating system.
- The systems with high end applications only use operating system. To use the operating system the embedded system should have large memory capability.
- So, this is not possible in low end applications like remote systems, digital cameras, MP3 players, robot toys etc.
- The architecture of an embedded system with OS can be denoted by layered structure as shown below.
- The OS will provide an interface between the hardware and application software.



In the case of embedded systems with OS, once the application software is loaded into memory it will run the application without any host system. Coming to the hardware details of the embedded system, it consists of the following important blocks.

1. CPU(Central Processing Unit)
2. RAM and ROM
3. I/O Devices
4. Communication Interfaces
5. Sensors etc. (Application specific circuitry)

This hardware architecture can be shown by the following block diagram.



Central Processing Unit:

- A CPU is composed of an Arithmetic Logic Unit (ALU), a Control Unit (CU), and many internal registers that are connected by buses.
- The ALU performs all the mathematical operations (Add, Sub, Mul, Div), logical operations (AND, OR), and shifting operations within CPU.
- The timing and sequencing of all CPU operations are controlled by the CU, which is actually built of many selection circuits including latches and decoders. The CU is responsible for directing the flow of instruction and data within the CPU and continuously running program instructions step by step.
- The CPU works in a cycle of fetching an instruction, decoding it, and executing it, known as the fetch-decode-execute cycle.
- For embedded system design, many factors impact the CPU selection, e.g., the maximum size (number of bits) in a single operand for ALU (8, 16, 32, 64 bits), and CPU clock frequency for timing tick control, i.e. the number of ticks (clock cycles) per second in measures of MHz
- CPU contains the core and the other components which support the core to execute programs. Peripherals are the components which communicate with other systems or physical world (Like ports, ADC,DAC, Watch dog Timers etc.). The core is separated from other components by the system bus.

When data and code lie in different memory blocks, then the architecture is referred as **Harvard architecture**.
In case data and code lie in the same memory block, then the architecture is referred as **Von Neumann architecture**.

Von-Neumann Architecture vs Harvard Architecture

Von-Neumann Architecture	**Harvard Architecture**
Single memory to be shared by both code and data.	Separate memories for code and data.
Processor needs to fetch code in a separate clock cycle and data in another clock cycle. So it requires two clock cycles.	Single clock cycle is sufficient, as separate buses are used to access code and data.
Higher speed, thus less time consuming.	Slower in speed, thus more time-consuming.
Simple in design.	Complex in design.

Memory:

1. Embedded system memory can be either on-chip or off-chip.
2. On chip memory access is much fast than off-chip memory, but the size of on-chip memory is much smaller than the size of off-chip memory.
3. Usually, it takes at least two I/O ports as external address lines plus a few control lines such as R/W and ALE control lines to enable the extended memory. Generally the data is stored in RAM and the program is stored in ROM.

ROM :

- The ROM, EPROM, and Flash memory are all read-only type memories often used to store code in an embedded system.
- When the power is on, the first instruction in ROM is loaded into the PC and then the CPU fetches the instruction from the location in the ROM pointed to by the PC and stores it in the IR to start the continuous CPU fetch and execution cycle. The PC is advanced to the address of the next instruction depending on the length of the current instruction or the destination of the Jump instruction.
- The memory is divided into Data Memory and Code Memory.

RAM:

- Most of data is stored in Random Access Memory (RAM) and code is stored in Read Only Memory (ROM).
- This is due to the RAM constraint of the embedded system and the memory organization.
- The RAM is readable and writable, faster access and more expensive volatile storage, which can be used to store either data or code.
- Once the power is turned off, all information stored in the RAM will be lost.

I/O Ports:

- The I/O ports are used to connect input and output devices. The common input devices for an embedded system include keypads, switches, buttons, knobs, and all kinds of sensors (light, temperature, pressure, etc).
- The output devices include Light Emitting Diodes (LED), Liquid Crystal Displays (LCD), printers, alarms, actuators, etc. Some devices support both input and output, such as communication interfaces including Network Interface Cards (NIC), modems, and mobile phones.

Communication Interfaces:

To transfer the data or to interact with other devices, the embedded devices are provided the various communication interfaces like RS232, RS422, RS485 ,USB, SPI(Serial Peripheral Interface) ,SCI (Serial Communication Interface) ,Ethernet etc.

Application Specific Circuitry:

- The embedded system sometimes receives the input from a sensor or actuator. In such situations certain signal conditioning circuitry is needed. This hardware circuitry may contain ADC, Op-amps, DAC etc. Such circuitry will interact with the embedded system to give correct output.

ADC & DAC:

- Many embedded system application need to deal with non-digital external signals such as electronic voltage, music or voice, temperature, pressures, and many other signals in the analog form.
- The digital computer does not understand these data unless they are converted to digital formats. The ADC is responsible for converting analog values to binary digits.
- The DAC is responsible for outputting analog signals for automation controls such as DC motor or HVDC furnace control.

In addition to these peripherals, an embedded system may also have sensors, Display modules like LCD or Touch screen panels, Debug ports certain communication peripherals like I2C, SPI, Ethernet, CAN, USB for high speed data transmission. Sensors like temperature sensors, light sensors, PIR sensors, gas sensors are widely used in application specific circuitry.

Address bus and data bus:

- Data bus is bidirectional, while address bus is unidirectional. That means data travels in both directions but the addresses will travel in only one direction.
- The reason for this is that unlike the data, the address is always specified by the processor. The width of the data bus is determined by the size of the individual memory block, while the width of the address bus is determined by the size of the memory that should be addressed by the system.

Power supply:

- Most of the embedded systems now days work on battery operated supplies.
- Because low power dissipation is always required. Hence the systems are designed to work with batteries.

Clock:

- The clock is used to control the clocking requirement of the CPU for executing instructions and the configuration of timers.
- A timer is a real-time clock for real-time programming. Every timer comes with a counter which can be configured by programs to count the incoming pulses.
- In addition to time delay generation, the timer is also widely used in the real-time embedded system to schedule multiple tasks in multitasking programming.
- The **watchdog timer** is a special timing device that resets the system after a preset time delay in case of system anomaly. The watchdog starts up automatically after the system power up.

Application Specific software:

It sits above the O.S. The application software is developed according to the features of the development tools available in the OS. These development tools provide the function calls to access the services of the OS. These function calls include, creating a task ,to read the data from the port and write the data to the memory etc.
The various function calls provided by an operating system are

- To create ,suspend and delete tasks.
- To do task scheduling to providing real time environment.
- To create inter task communication and achieve the synchronization between tasks.
- To access the I/O devices.
- To access the communication protocol stack .

CISC and RISC Architecture:

- CISC is a Complex Instruction Set Computer. It is a computer that can address a large number of instructions.
- In the early 1980s, computer designers recommended that computers should use fewer instructions with simple constructs so that they can be executed much faster within the CPU without having to use memory. Such computers are classified as Reduced Instruction Set Computer or RISC.

Difference CISC vs RISC:

CISC	**RISC**
Larger set of instructions. Easy to program	Smaller set of Instructions. Difficult to program.
Simpler design of compiler, considering larger set of instructions.	Complex design of compiler.
Many addressing modes causing complex instruction formats.	Few addressing modes, fix instruction format.
Instruction length is variable.	Instruction length varies.
Higher clock cycles per second.	Low clock cycle per second.
Emphasis is on hardware.	Emphasis is on software.
Control unit implements large instruction set using micro-program unit.	Each instruction is to be executed by hardware.
Slower execution, as instructions are to be read from memory and decoded by the decoder unit.	Faster execution, as each instruction is to be executed by hardware.
Pipelining is not possible.	Pipelining of instructions is possible, considering single clock cycle.
Mainly used in normal pc‘s, workstations & servers.	Mainly used for real time applications.

5.5 CATEGORIES OF EMBEDDED SYSTEMS:

Embedded systems can be classified into the following 4 categories based on their functional and performance requirements.

- Based on the functional requirements,
    1. Stand alone embedded systems
    2. Real time embedded system
       a)Hard real time E.S
       b)Soft Real time E.S
    3. Networked embedded system
    4. Mobile embedded system

- Based on the performance requirements,
    1. small scale embedded system
    2. medium scale embedded s/m
    3. large scale embedded system

Based on the functional requirements:

1. **Stand alone Embedded systems:** A stand-alone embedded system works by itself. It is a self-contained device which does not require any host system like a computer. It takes either digital or analog inputs from its input ports, calibrates, converts, and processes the data, and outputs the resulting data to its attached output device, which either displays data, or controls and drives the attached devices. **EX:** Temperature measurement systems, Video game consoles, MP3 players, digital cameras, and microwave ovens are the examples for this category.
2. **Real-time embedded systems:** An embedded system which gives the required output in a specified time or which strictly follows the time deadlines for completion of a task is known as a Real time system. i.e. a Real Time system , in addition to functional correctness, also satisfies the time constraints .
   Ex: A Microwave Oven, washing machine, TV remote etc
   These are classified in to two types
   - **Soft Real time Systems:** Missing a deadline may not be critical and can be tolerated to a certain degree
   - **Hard Real time systems:** Missing a program/task execution time deadline can have catastrophic consequences (financial, human loss of life, etc.
3. **Networked embedded systems:** The networked embedded systems are related to a network with network interfaces to access the resources. The connected network can be a Local Area Network (LAN) or a Wide Area Network (WAN), or the Internet. The connection can be either wired or wireless. The networked embedded system is the fastest growing area in embedded systems applications. The embedded web server is such a system where all embedded devices are connected to a web server and can be accessed and controlled by any web browser. **Ex:** A home security system is an example of a LAN networked embedded system where all sensors (e.g. motion detectors, light sensors, or smoke sensors) are wired and running on the TCP/IP protocol.
4. **Mobile Embedded systems:** The portable embedded devices like mobile and cellular phones, digital cameras, MP3 players, PDA (Personal Digital Assistants) are the example for mobile embedded systems. The basic limitation of these devices is the limitation of memory and other resources.

Based on the performance of the Microcontroller they are also classified into (i) Small scaled embedded system (ii) Medium scaled embedded system and (iii) Large scaled embedded system.

1. **Small scaled embedded system:** An embedded system supported by a single 8–16 bit Microcontroller with on-chip RAM and ROM designed to perform simple tasks is a Small scale embedded system.
2. **Medium scaled embedded system:**
   An embedded system supported by 16–32 bit Microcontroller /Microprocessor with external RAM and ROM that can perform more complex operations is a Medium scale embedded system.
3. **Large scaled embedded system:** An embedded system supported by 32-64 bit multiple chips which can perform distributed jobs is considered as a Large scale embedded system.

5.6:EMBEDDED PROCESSOR SELECTION AND TRADEOFFS

Processor selection for an embedded system

With numerous kinds of processors with various design philosophies available at our disposal for using in our design, following considerations need to be factored during processor selection for an embedded system.

- Performance Considerations
- Power considerations
- Peripheral Set
- Operating Voltage
- Specialized Processing Units
- Price

Performance considerations

The performance speed of a processor is dependent primarily on its architecture and its silicon design. Evolution of fabrication techniques helped packing more transistors in same area there by reducing the propagation delay. Also presence of cache reduces instruction/data fetch timing. Pipelining and super-scalar architectures further improves the performance of the processor. Branch prediction, speculative execution etc are some other techniques used for improving the execution rate. Multi-cores are the new direction in improving the performance.

- Rather than simply stating the clock frequency of the processor which has limited significance to its processing power, it makes more sense to describe the capability in a standard notation. MIPS (Million Instructions Per Second) or MIPS/MHz was an earlier notation followed by Dhrystones and latest EEMBC's **CoreMark**. CoreMark is one of the best ways to compare the performance of various processors.

Processor architectures with support for extra instruction can help improving performance for specific applications.So size of cache, processor architecture, instruction set etc has to be taken in to account when comparing the performance.

Power Considerations

Increasing the logic density and clock speed has adverse impact on power requirement of the processor. A higher clock implies faster charge and discharge cycles leading to more power consumption. More logic leads to higher power density there by making the heat dissipation difficult. Further with more emphasis on greener technologies and many systems becoming battery operated, it is important the design is for optimal power usage.

Techniques like

- **Frequency Scaling** – reducing the clock frequency of the processor depending on the load,
- **Voltage Scaling** – varying the voltage based on load can help in achieving lower power usage. Further asymmetric multiprocessors, under near idle conditions, can effectively power off the more powerful core and load the less powerful core for performing the tasks.
- **SOC** comes with advanced power gating techniques that can shut down clocks and power to unused modules.

Peripheral Set

Every system design needs, apart from the processor, many other peripherals for input and output operations. Since in an embedded system, almost all the processors used are SoCs, it is better if the necessary peripherals are available in the chip itself.

This offers various benefits compared to peripherals in external IC's such as optimal power architecture, effective data communication using DMA, lower BoM etc. So it is important to have peripheral set in consideration when selecting the processor.

Oprating Voltages

Each and every processor will have its own operating voltage condition. The operating voltage maximum and minimum ratings will be provided in the respective data sheet or user manual.

While higher end processors typically operate with 2 to 5 voltages including 1.8V for Cores/Analogue domains, 3.3V for IO lines, needs specialized PMIC devices, it is a deciding factor in low end micro-controllers based on the input voltage. For example it is cheaper to work with a 5V micro-controller when the input supply is 5V and a 3.3 micro-controllers when operated with Li-on batteries.

Specialized Processing

Apart from the core, presence of various co-processors and specialized processing units can help achieving necessary processing performance. Co-processors execute the instructions fetched by the primary processor thereby reducing the load on the primary. Some of the popular co-processors include

Floating Point Co-processor:

RISC cores supports primarily integer only instruction set. Hence presence of a FP co-processor can be very helpful in application involving complex mathematical operations including multimedia, imaging, codecs, signal processing etc.

Graphic Processing Unit:

GPU(Graphic Processing Unit) also called as Visual processing unit is responsible for drawing images on the frame buffer memory to be displayed. Since human visual perception needed at-least 16 Frames per second for a smooth viewing, drawing for HD displays involves a lot of data bandwidth. Also with increasing graphic requirements such as textures, lighting shaders etc, GPU's have become a mandatory requirements for mobile phones, gaming consoles etc.

Digital Signal Processors

DSP is a processor designed specifically for signal processing applications. Its architecture supports processing of multiple data in parallel. It can manipulate real time signal and convert to other domains for processing. DSP's are either available as the part of the SoC or separate in an external package. DSP's are very helpful in multimedia applications. It is possible to use a DSP along with a processor or use the DSP as the main processor itself.

Price

Various considerations discussed above can be taken in to account when a processor is being selected for an embedded design. It is better to have some extra buffer in processing capacities to enable enhancements in functionality without going for a major change in the design. While engineers (especially software/firmware engineers) will want to have all the functionalities, price will be the determining factor when designing the system and choosing the right processor.

References:

1. Frank Vahid,Givargis,"Embedded System Design :A unified hardware /Software introduction", student edition wiley publication,2009
2. Arnold S. Berger," Embedded Systems Design: An Introduction to Processes, Tools, and Techniques" CMP Books,2002