UNIT-III

Overview of Microcontroller and Embedded Systems

Embedded Hardware and Various Building Blocks:

The basic hardware components of an embedded system shown in a block diagram in below figure.

These include the processing unit, sensors and actuators, ADC, DAC, I/O unit and the memory block. The processing unit could be a microprocessor, a microcontroller, FPGA i.e. field programmable gate array or ASIC (Application Specific IC) depending on the application requirements.

Sensors such as sound sensor, ambient temperature sensor, motion sensor etc. are generally analog in nature since they sense the data from outside world. This data is converted from analog to digital and sent to a processing unit, post which required action is performed by actuators.

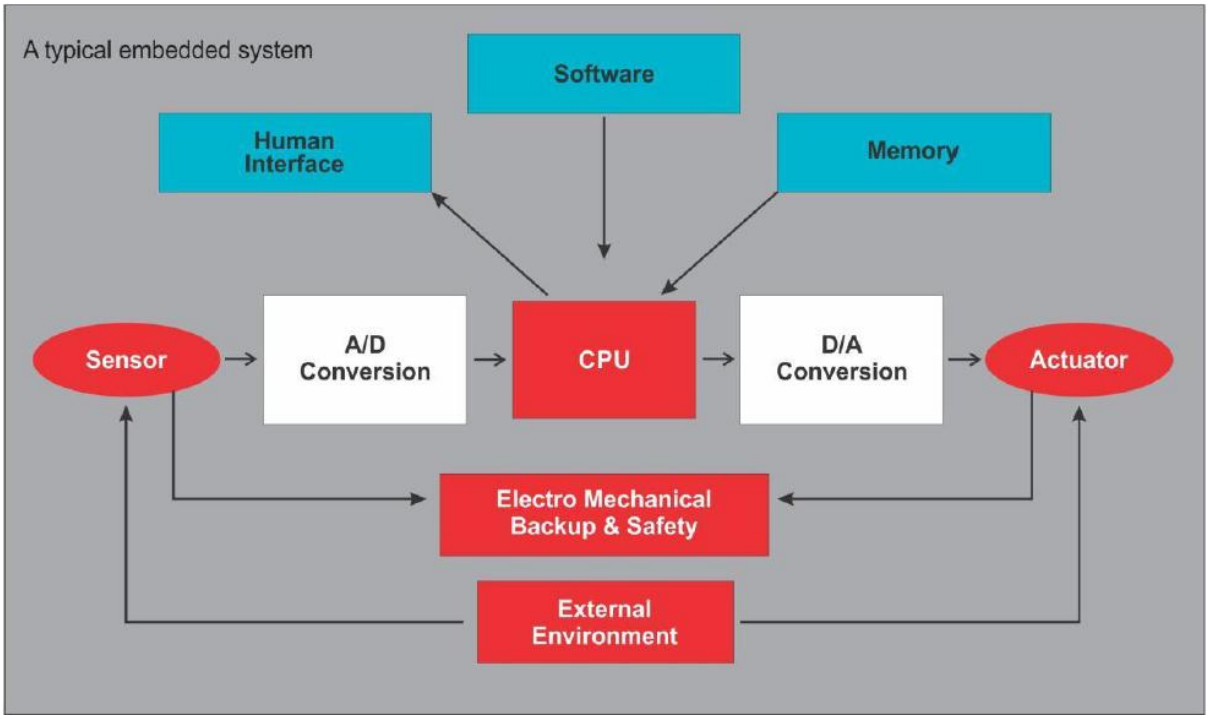


Figure: Basic block diagram of an embedded system

Processor Selection for an Embedded System:

The processing unit could be a microprocessor, a microcontroller, embedded processor, DSP, ASIC or FPGA selected for an embedded system based on the application requirements.

This processing unit executes the application program that is saved in the program memory ROM (read only memory). The RAM (random access memory) is used as the data memory to hold the system stack and the variables used in the program.

Stack is a portion in the RAM reserved to hold back the status of the program when the control is transferred by a branch instruction. To make a system interactive, input-output (I/O) unit is required. The memory block and the I/O units communicate with the processing unit through the system bus.

The system bus consists of three different bus systems: address bus, data bus and control bus. Processor sends the address of the destination through the address bus. So address bus is unidirectional from processor to the external end. Data can be sent or received from any unit to any other unit in the diagram.

So data bus is bidirectional. Control bus is basically a group of control signals from the processing unit to the external units.

Interfacing Processor:

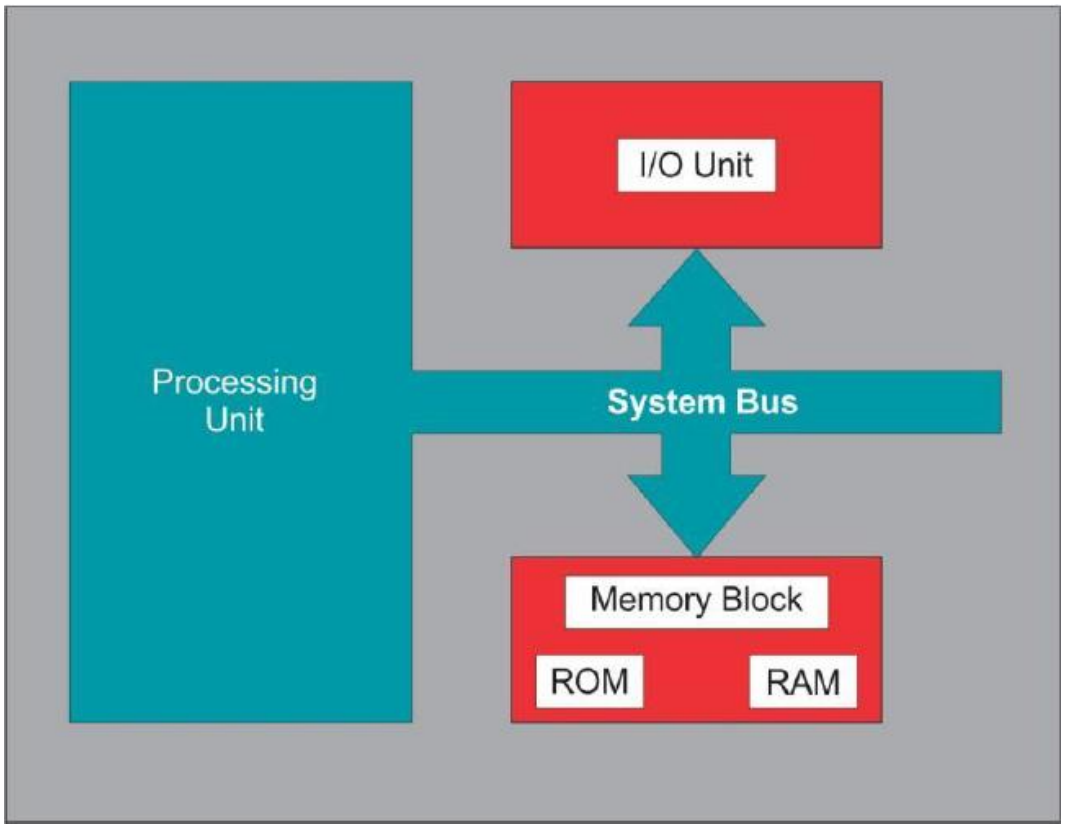


Figure: Processing Unit and System Bus

Microprocessor

Microprocessor is a programmable digital device which has high computational capability to run a number of applications in general purpose systems. It does not have memory or I/O ports built within its architecture. So, these devices need to be added externally to make a system functional. In embedded systems, the design is constrained with limited memory and I/O features. So microprocessors are used where system capability needs to be expanded by adding external memory and I/O.

Microcontroller

A microcontroller has a specific amount of program and data memory, as well as I/O ports built within the architecture along with the CPU core, making it a complete system. As a result, most embedded systems are microcontroller based, where are used to run one or limited number of applications.

Embedded Processor

Embedded processors are specifically designed for embedded systems to meet design constraints. They have the potential to handle multitasking applications. The performance and power efficiency requirements of embedded systems are satisfied by the use of embedded processors.

DSP

Digital signal processors (DSP) are used for signal processing applications such as voice or video compression, data acquisition, image processing or noise and echo cancellation.

ASIC

Application specific integrated circuit (ASIC) is basically a proprietary device designed and used by a company for a specific line of products (for example Samsung cell phones or Cisco routers etc.). It is specifically an algorithm called intellectual property core implemented on a chip.

FPGA

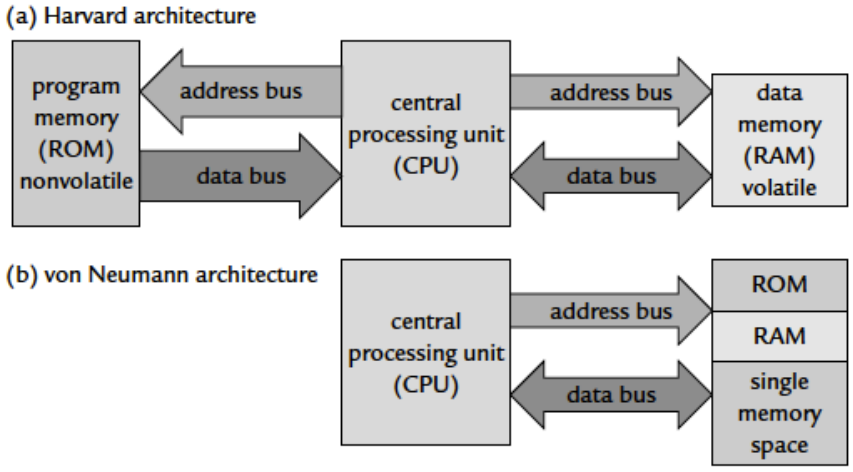
Field programmable gate arrays (FPGA) have programmable macro cells and their interconnects are configured based on the design. They are used in embedded systems when it is required to enhance the computational capability of the existing system or to make a system reprogrammable and reconfigurable when the need arises.

Memories and I/O Devices:

Memory Block:

The memory block consists of program and data memory. ROM is used as the program memory and RAM is used as the data memory. There are two memory architectures: Harvard and Von-Neumann.

In Harvard architecture, the program and data memories are segregated with separate address and data bus drawn to each. So there can be parallel access to both and performance of the system can be improved at the cost of hardware complexity. On the other-hand, the Von-Neumann architecture has one unified memory used for both program and data. The system is comparatively slower, but the design implementation is simple and cost effective for an embedded system. Various ROM and RAM devices are used in embedded systems based on the applications.



Harvard and von Neumann architectures for memory.

ROM

Read only memory (ROM) is non-volatile i.e. it retains the contents even after power goes off. It is used as the program memory. In embedded systems, the application program after being compiled is saved in the ROM. The processing unit accesses the ROM to fetch instructions sequentially and executes them within the CPU. There are different categories of ROM such as: programmable read only memory (PROM), erasable programmable read only memory (EPROM), electrically erasable programmable read only memory (EEPROM) etc. There is also flash memory which is the updated version of EEPROM and extensively used in embedded systems.

RAM

Random access memory (RAM) is volatile i.e. it does not retain the contents after the power goes off. It is used as the data memory in an embedded system. It holds the variables declared in the program, the stack and intermediate data or results during program run time. The Processing unit accesses the RAM for instruction execution to save or retrieve data. There are different variations of RAM such as: static RAM (SRAM), dynamic RAM (DRAM), pseudo static RAM (PSRAM), non-volatile RAM (NVRAM), synchronous DRAM, (SDRAM) etc.

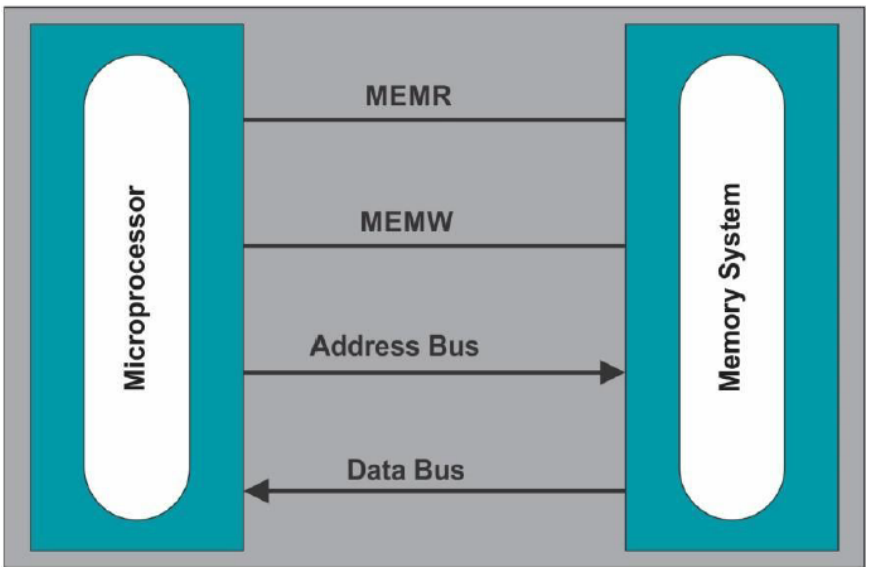


Figure: Interconnection of RAM with Microprocessor

I/O Devices:

Embedded systems have to interact with the external environment through the input/output devices.

Input Device

Embedded systems receive user commands from input devices such as keypad, switch or a touch screen device at the input port. The processing unit executes software instructions to process these inputs to make decisions that further guide the operation of the system. A port is a termination point that gives connectivity between the processing unit and the peripherals.

Output Device

Output devices are used to display results from the system or to sending data to another connected system at the output port. Some examples of output devices are: light emitting diodes (LEDs), liquid crystal diodes (LCDs), printers etc.

I/O Interfacing Concepts:

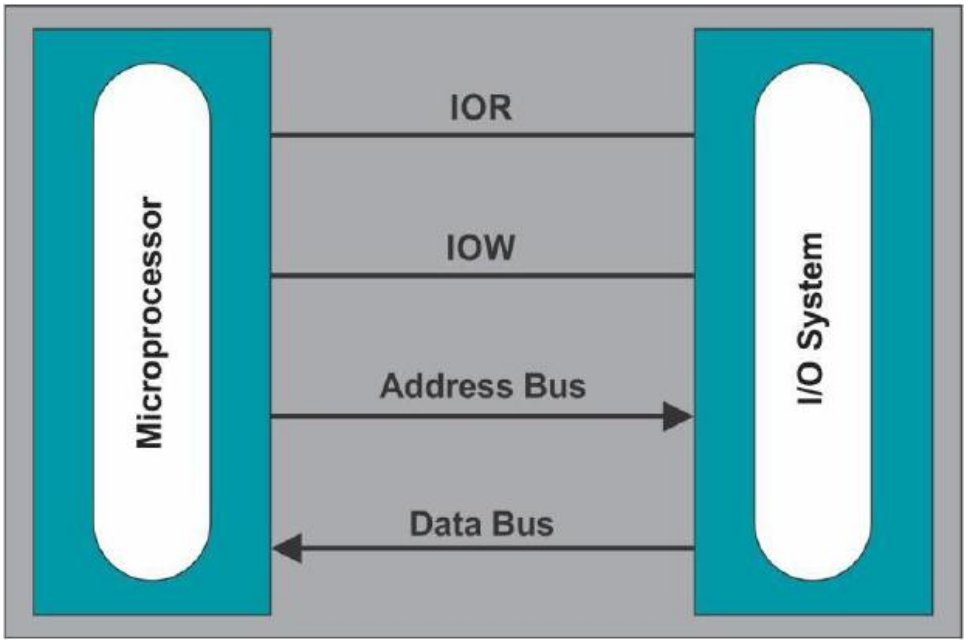


Figure: Interconnection of external devices with Microprocessor

I/O Communication Bus

I/O communication buses and protocols are used to communicate with the slower I/O devices. There are two communication methods used in any system: serial communication and parallel communication. Some of the communication protocols are: universal serial bus (USB), inter-integrated circuit (I2C), serial peripheral interface (SPI), peripheral component interconnect (PCI), IBM standard architecture (ISA) etc. Each protocol defines a standard way of communication between the devices. The features and usage of these protocols have been explained in subsequent chapters.

Sensors & Actuators

Sensors and electromechanical actuators are input and output devices used in real time embedded systems to exchange real time data between the system and the external environment. Sensors measure physical parameters such as temperature, pressure acceleration, proximity etc. being connected at the system input ports through analog to digital converters (ADCs). Some of the actuators used in embedded systems are: motor speed controllers, stepper motor controllers, relays and power drivers etc. Actuators are connected at the system output ports through the digital to analog converters (DACs).

Timer and Counting Devices:

Timers are basic constituents of most microcontrollers. Today, just about every microcontroller comes with one or more built-in timers. These are extremely useful to the embedded programmer – perhaps second in usefulness only to GPIO. The timer can be described as the counter hardware and can usually be constructed to count either regular or irregular clock pulses. Depending on the above usage, it can be a timer or a counter respectively.

Sometimes, timers may also be termed as "hardware timers" to distinguish them from software timers. Software timers can be described as a stream of bits of software that achieve some timing function.

The TM4C123GH6PM General-Purpose Timer Module (GPTM) contains six 16/32-bit GPTM blocks and six 32/64-bit Wide GPTM blocks. These programmable timers can be used to count or time external events that drive the Timer input pins. Timers can also be used to trigger μDMA transfers, to trigger analog-to-digital conversions (ADC) when a time-out occurs in periodic and one-shot modes.

The GPT Module is one timing resource available on the Tiva™ C Series microcontrollers. Other timer resources include the System Timer (SysTick) and the PWM timer in PWM modules.

The General-Purpose Timer Module (GPTM) blocks with the following functional options:

- **16/32-bit operating modes:**
  - ⇨ 16- or 32-bit programmable one-shot timer
  - ⇨ 16- or 32-bit programmable periodic timer
  - ⇨ 16-bit general-purpose timer with an 8-bit prescaler
  - ⇨ 32-bit Real-Time Clock (RTC) when using an external 32.768-KHz clock as the input
  - ⇨ 16-bit input-edge count- or time-capture modes with an 8-bit prescaler
  - ⇨ 16-bit PWM mode with an 8-bit prescaler and software-programmable output inversion of the PWM signal
- **32/64-bit operating modes:**
  - ⇨ 32- or 64-bit programmable one-shot timer
  - ⇨ 32- or 64-bit programmable periodic timer

- ⇨ 32-bit general-purpose timer with a 16-bit prescaler
- ⇨ 64-bit Real-Time Clock (RTC) when using an external 32.768-KHz clock as the input
- ⇨ 32-bit input-edge count- or time-capture modes with a16-bit prescaler
- ⇨ 32-bit PWM mode with a 16-bit prescaler and software-programmable output inversion of the PWM signal

- ❑ Count up or down
- ❑ Twelve 16/32-bit Capture Compare PWM pins (CCP)
- ❑ Twelve 32/64-bit Capture Compare PWM pins (CCP)
- ❑ Daisy chaining of timer modules to allow a single timer to initiate multiple timing events
- ❑ Timer synchronization allows selected timers to start counting on the same clock cycle
- ❑ ADC event trigger
- ❑ User-enabled stalling when the microcontroller asserts CPU Halt flag during debug (excluding RTC mode)
- ❑ Ability to determine the elapsed time between the assertion of the timer interrupt and entry into the interrupt service routine
- ❑ Efficient transfers using Micro Direct Memory Access Controller (μDMA)
  - ⇨ Dedicated channel for each timer
  - ⇨ Burst request generated on timer interrupt

Design Cycle in the Development Phase for an Embedded System:

Unlike the design of a software application on a standard platform, the design of an embedded system implies that both software and hardware are being designed in parallel. Although this isn't always the case, it is a reality for many designs today. The profound implications of this simultaneous design process heavily influence how systems are designed.

The following figure provides a schematic representation of the Design Cycle in the Development Phase for an Embedded System.

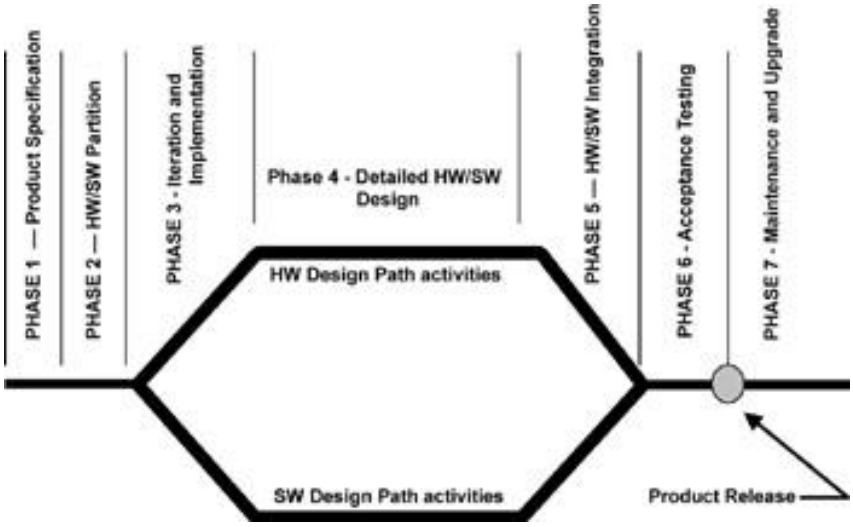


Figure: Embedded Design Life Cycle Diagram

A Phase representation of the Embedded Design Life Cycle

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 above figure depicts. A considerable amount of iteration and optimization occurs within phases and between phases. Defects found in later stages often cause you to go back to square 1. For example, when product testing reveals performance deficiencies that render the design non-competitive, you might have to rewrite algorithms, redesign custom hardware such as Application-Specific Integrated Circuits (ASICs) for better performance speed up the processor, choose a new processor, and so on.

Uses of Target System or its Emulator and In-Circuit Emulator (ICE):

An in-circuit emulator (ICE) is a hardware interface that allows a programmer to change or debug the software in an embedded system. The ICE is temporarily installed between the embedded system and an external terminal or personal computer so that the programmer can observe and alter what takes place in the embedded system, which has no display or keyboard of its own.

An in-circuit emulator (ICE) provides a window into the embedded system. The programmer uses the emulator to load programs into the embedded system, run them, step through them slowly, and view and change data used by the system's software.

An emulator gets its name because it emulates (imitates) the central processing unit (CPU) of the embedded system's computer. Traditionally it had a plug that inserts into the socket where the CPU integrated circuit chip would normally be placed. Most modern systems use the target system's CPU directly, with special JTAG-based debug access. Emulating the processor, or direct JTAG access to it, lets the ICE do anything that the processor can do, but under the control of a software developer.

ICEs attach a computer terminal or personal computer (PC) to the embedded system. The terminal or PC provides an interactive user interface for the programmer to investigate and control the embedded system. For example, it is routine to have a source code level debugger with a graphical windowing interface that communicates through a JTAG adapter (emulator) to an embedded target system which has no graphical user interface.

Notably, when their program fails, most embedded systems simply become inert lumps of nonfunctioning electronics. Embedded systems often lack basic functions to detect signs of software failure, such as a memory management unit (MMU) to catch memory

access errors. Without an ICE, the development of embedded systems can be extremely difficult, because there is usually no way to tell what went wrong. With an ICE, the programmer can usually test pieces of code, then isolate the fault to a particular section of code, and then inspect the failing code and rewrite it to solve the problem.

In usage, an ICE provides the programmer with execution breakpoints, memory display and monitoring, and input/output control. Beyond this, the ICE can be programmed to look for any range of matching criteria to pause at, in an attempt to identify the origin of a failure.

Most modern microcontrollers use resources provided on the manufactured version of the microcontroller for device programming, emulating, and debugging features, instead of needing another special emulation-version (that is, bond-out) of the target microcontroller.[1] Even though it is a cost-effective method, since the ICE unit only manages the emulation instead of actually emulating the target microcontroller, trade-offs must be made to keep prices low at manufacture time, yet provide enough emulation features for the (relatively few) emulation applications.

Note: Debugging is the process of finding and resolving defects or problems within a computer program that prevent correct operation of computer software or a system.