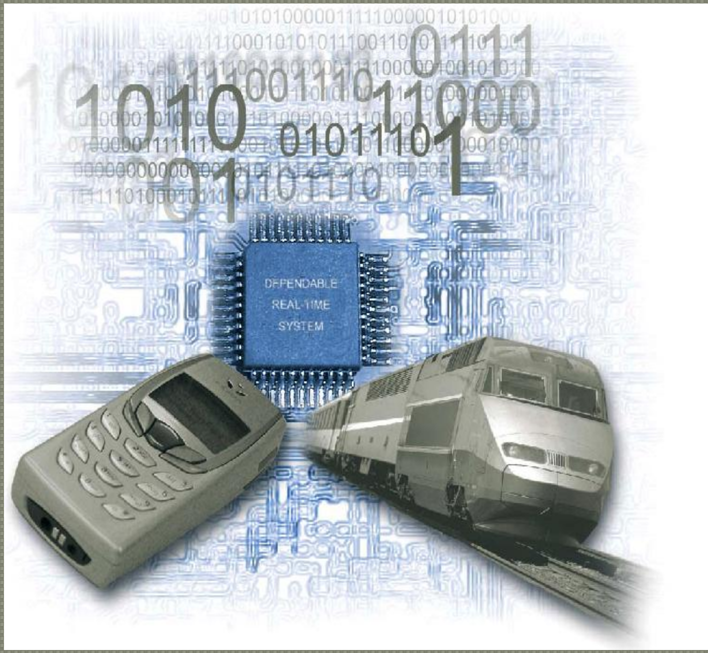
EMBEDDED SYSTEM DESIGN
IV BTECH I-R15- ECE

Mr. N Paparao
Assistant Professor
Mr. S Lakshmanachari
Assistant Professor
Mr. MD Khadir
Assistant Professor

INSTITUTE OF AERONAUTICAL ENGINEERING
(Autonomous)
DUNDIGAL, HYDERABAD - 500 043



UNIT I
Introduction to EMBEDDED SYSTEM

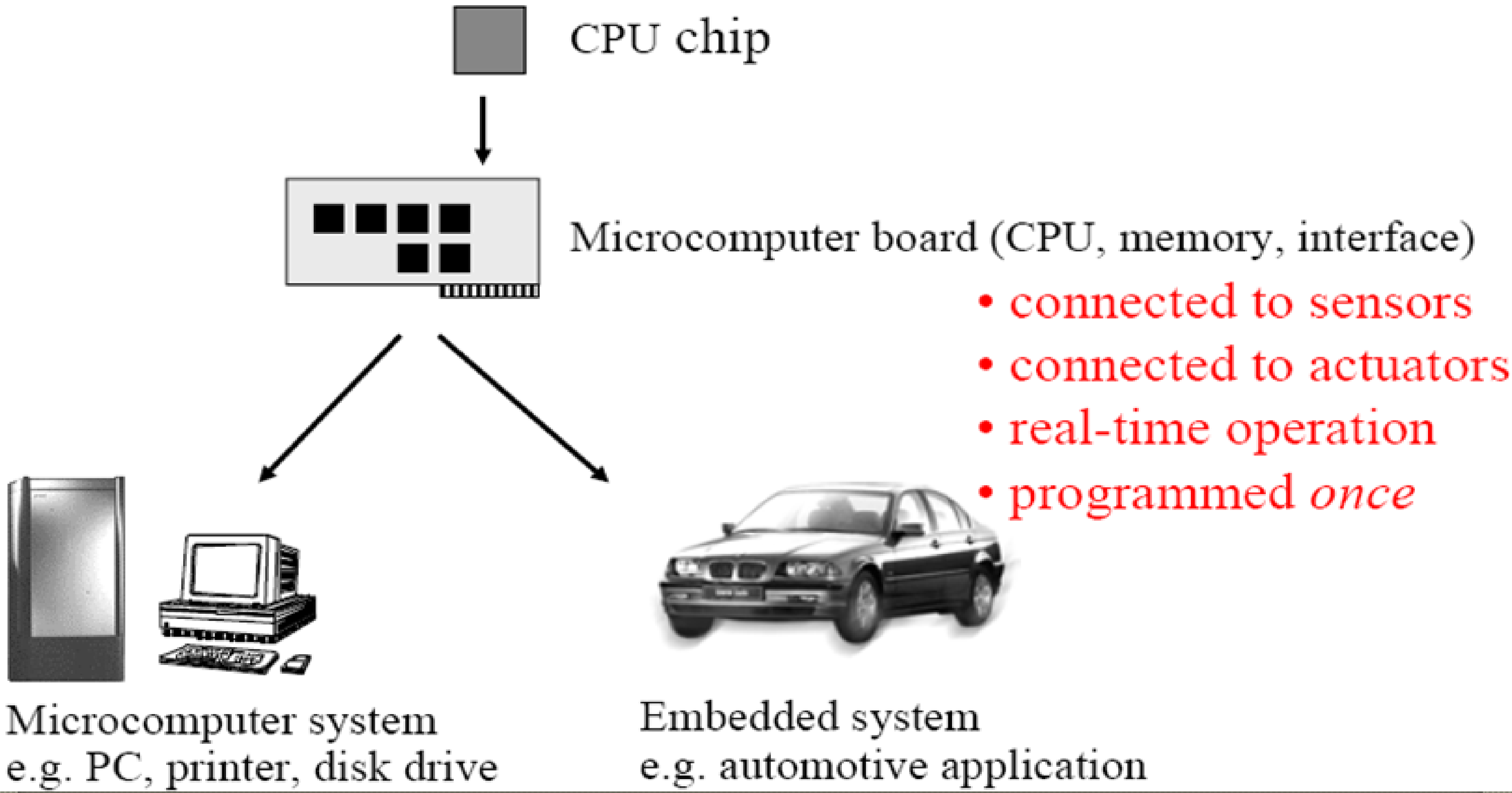
Introduction:-

Definition

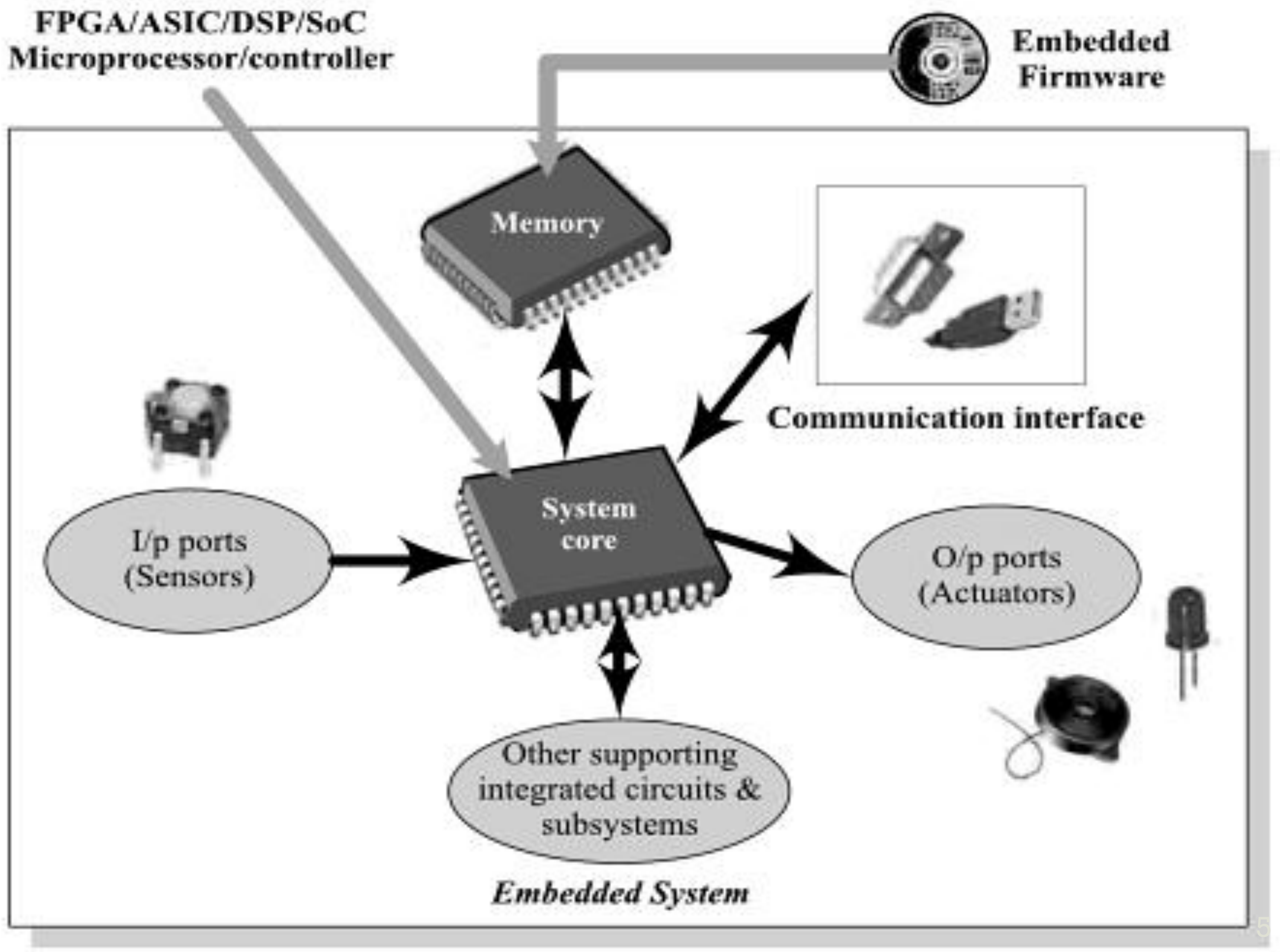
- *It is an Electronic/Electro-mechanical system designed to perform a specific function and is a combination of both hardware & software.*

OR

- *A combination of hardware and software which together form a component of a larger machine.*



- An example of an embedded system is a microprocessor that controls an automobile engine.
- An embedded system is designed to run on its own without human intervention, and may be required to respond to events in real time.


FPGA/ASIC/DSP/SoC
Microprocessor/controller
Embedded Firmware
Memory
Communication interface
System core
I/p ports (Sensors)
O/p ports (Actuators)
Other supporting integrated circuits & subsystems
Embedded System

History of Embedded Systems:-

- One of the very first recognizably modern embedded systems was the Apollo Guidance Computer, developed by Charles Stark Draper at the MIT Instrumentation Laboratory

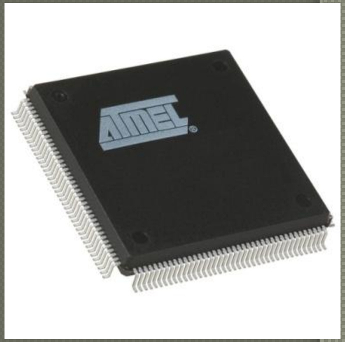
Apollo Guidance Computer:-

1. The Apollo Guidance Computer was the first modern system to collect and provide flight information, and to automatically control all of the navigational functions of the Apollo spacecraft.
2. It was developed in the early 1960s for the Apollo program by the MIT Instrumentation Lab under Charles Stark Draper.
3. "The guidance computer made the moon landings possible.
4. It was designed almost entirely by MIT faculty and alumni from the Draper Lab (then called the Instrumentation Lab) and contractors staffed by MIT alumni.
5. The man on the moon was a huge milestone in the history of technology and of the Cold War, made possible entirely by MIT ingenuity.
6. "The Apollo Guidance Computer (AGC) was the first recognizably modern embedded system, used in real-time by astronaut pilots to collect and provide flight information, and to automatically control all of the navigational functions of the Apollo spacecraft.""

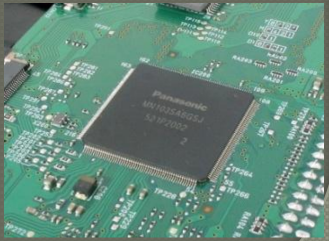
CLASSIFICATIONS OF EMBEDDED SYSTEM

1. **Small Scale Embedded System**

2. **Medium Scale Embedded System**



3. **Sophisticated Embedded System**



SMALL SCALE EMBEDDED SYSTEM

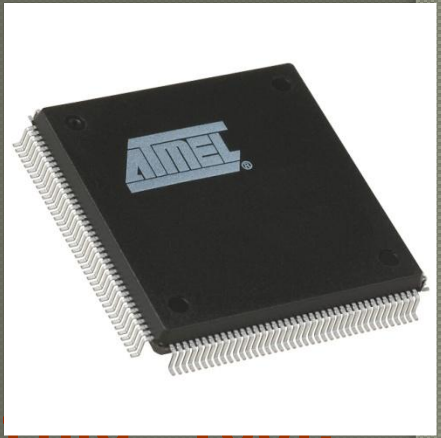
- **Single 8 bit or 16bit Microcontroller.**
- **Little hardware and software complexity.**
- **They May even be battery operated.**
- **Usually "C" is used for developing these system.**
- **The need to limit power dissipation when system is running continuously.**

Programming tools:

Editor, Assembler and Cross Assembler

MEDIUM SCALE EMBEDDED SYSTEM

- **Single or few 16 or 32 bit microcontrollers or Digital Signal Processors (DSP) or Reduced Instructions Set Computers (RISC).**



- **Both hardware and software complexity.**

Programming tools:

RTOS, Source code Engineering Tool, Simulator, Debugger and Integrated Development Environment (IDE).

SOPHISTICATED EMBEDDED SYSTEM

- **Enormous hardware and software complexity**
- **Which may need scalable processor or configurable processor and programming logic arrays.**
- **Constrained by the processing speed available in their hardware units.**

Programming Tools:
For these systems may not be readily a reasonable cost or may not be available at all. A compiler or retargetable compiler might have to br developed for this.

Major Application Areas Of Embedded Systems

1. **Consumer Electronics**
   - ❖ Camcorders, Cameras, etc…
2. **Household Appliances**
   - ❖ Television, DVD Player, Washing machine, fridge, microwave oven, etc.
3. **Home automation and security system**
   - ❖ Air conditioners, Sprinkler, intruder detection alarms, fire alarms, closed circuit television cameras, etc
4. **Automotive industry**
   - ❖ Anti-lock breaking system (ABS), engine control, ignition control, automatic navigation system, etc..
5. **Telecommunication**
   - ❖ Cellular telephones, telephone switches, Router, etc…

Continue…

6. **Computer peripherals**
   - ❖ Printers, scanners, fax machines, etc…
7. **Computer Networking systems**
   - ❖ Network routers, switches, hubs, firewalls, etc…
8. **Health care**
   - ❖ CT scanner, ECG , EEG , EMG ,MRI, Glucose monitor, blood pressure monitor, medical diagnostic device, etc.
9. **Measurement & Instrumentation**
   - ❖ Digital multi meters, digital CROs, logic analyzers PLC systems, etc…
10. **Banking & Retail**
    - ❖ Automatic Teller Machine (ATM) and Currency counters, smart vendor machine, cash register ,Share market, etc..
11. **Card Readers**
    - ❖ Barcode, smart card readers, hand held devices, etc…

Purpose Of Embedded Systems:-

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. Data collection/Storage/Representation

1. Data collection is usually done for storage, analysis, manipulation and transmission.
2. The term "Data" refers all kinds of information, viz. text, voice, image, electrical signals & other measurable quantities.
3. Data can be either analog (continues) or Digital (discrete).
4. Embedded system with analog data capturing techniques collect data directly in the form of analog and converts the analog to digital signal by using A/D converters and then collect the binary equivalent of the analog data.
5. If the signal is digital it can be directly captured without any additional interface by digital embedded system.
6. The collected data may be stored directly in the system or may be transmitted to other systems or it may be processed by the system or it may be deleted instantly after giving a meaningful representation.

- ❖ 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 with in the memory of the camera. The captured image can also be presented to the user through a LCD display unit.

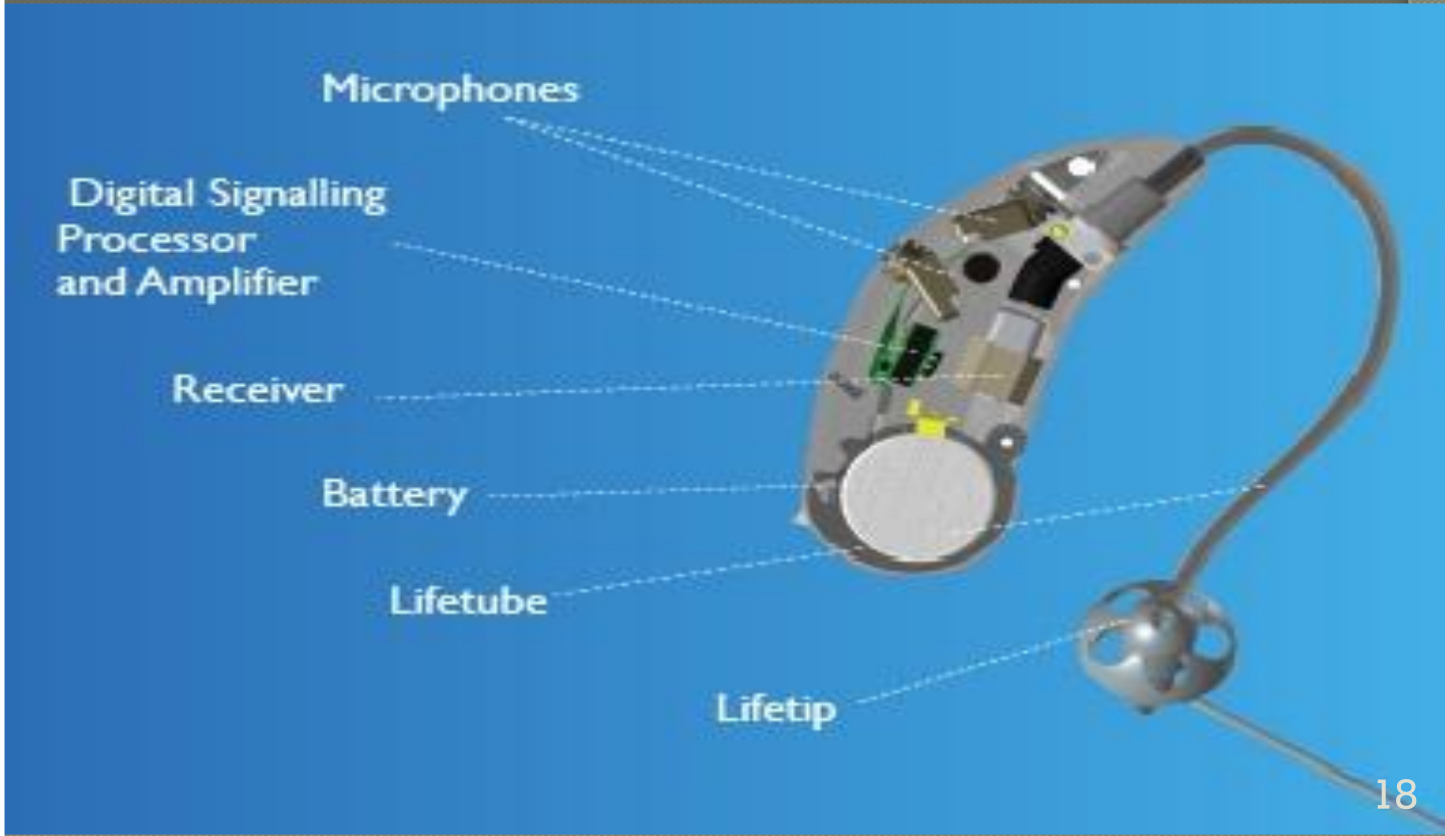
2. Data communication

- ❖ Embedded data communication systems are developed in applications ranging from complex satellite communication systems to simple home networking systems.

Figure: - A wireless network router for data communication

3. Data (Signal) Processing

- The data collected by embedded system may be used for various kinds of signal processing.
- A digital hearing aid is a typical example of an embedded system employing data processing.



4. Monitoring

- All embedded products coming under the medical domain are with monitoring functions only. They are used for determing the state of some variables using input sensors.
- A very good example is the electro cardiogram (ECG) machine for monitoring the heartbeat of patient.

Figure:- A patient monitoring system for monitoring for heartbeat

5. Control

- Embedded system with control functionalities impose control over some variables according to the input variables.
- A system with control functionality contains both sensors and actuators.
- Sensors are inputs ports for capturing the changes in environment variables or measuring variable.
- Actuators are output ports are controlled according to the changes in input variable.

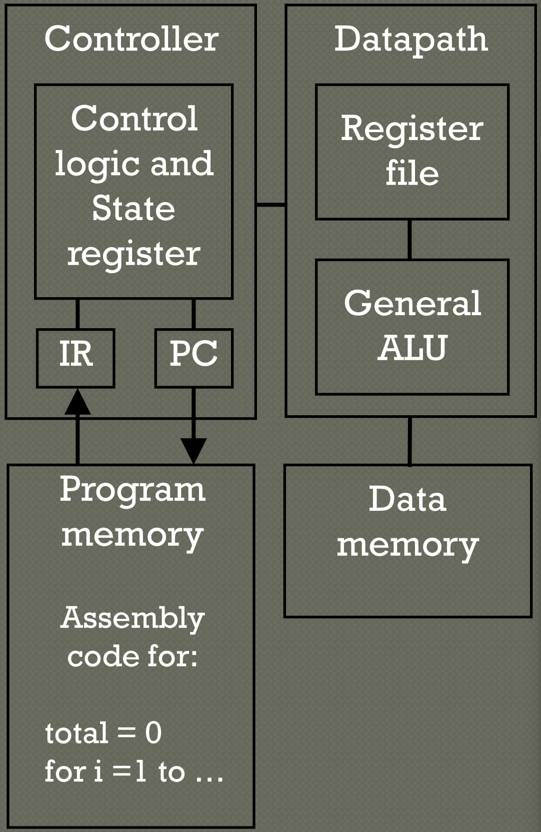
Figure:- An Air conditioner for controlling room temperature

6. Application specific user interface

- These are embedded systems with application specific user interfaces like buttons, switches, keypad, lights, bells, display units, etc..
- Mobile phone is an example for this, in mobile phone the user interface is provided through the keyboard, graphic LCD module, system speaker, vibration alert, etc...

General-purpose processors

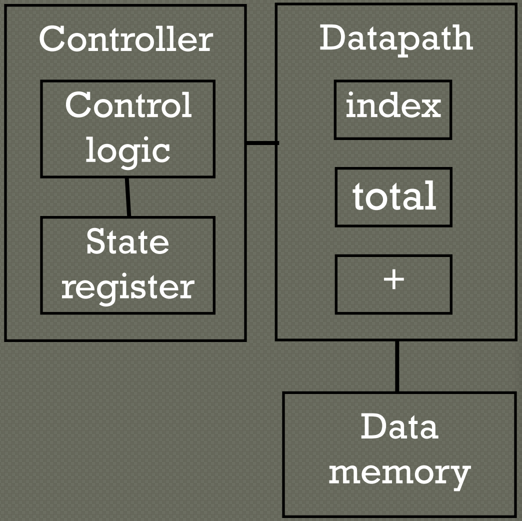
- Programmable device used in a variety of applications
  - Also known as "microprocessor"
- Features
  - Program memory
  - General datapath with large register file and general ALU
- User benefits
  - Low time-to-market and NRE costs
  - High flexibility
- "Pentium" the most well-known, but there are hundreds of others



Slide credit Vahid/Givargis, Embedded Systems Design: A Unified Hardware/Software Introduction, 2000

Single-purpose processors

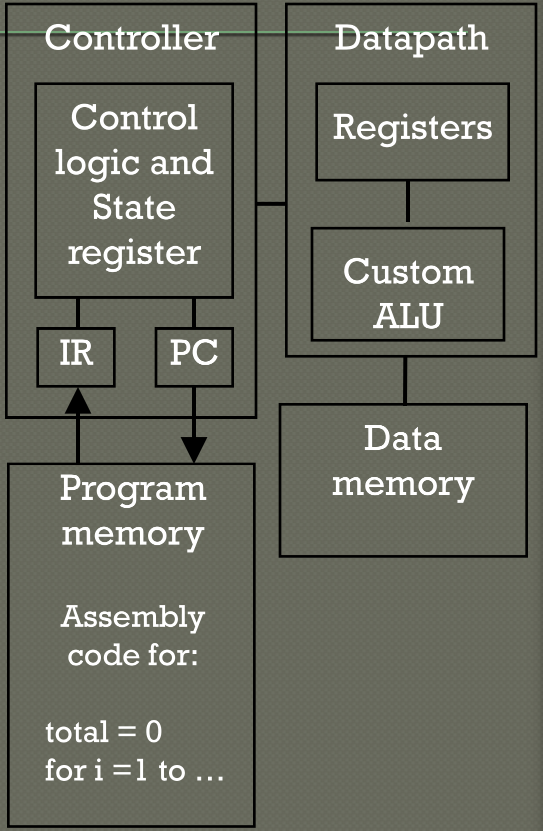
- Digital circuit designed to execute exactly one program
  - a.k.a. coprocessor, accelerator or peripheral
- Features
  - Contains only the components needed to execute a single program
  - No program memory
- Benefits
  - Fast
  - Low power
  - Small size



Slide credit Vahid/Givargis, Embedded Systems Design: A Unified Hardware/Software Introduction, 2000

Application-specific processors

- Programmable processor optimized for a particular class of applications having common characteristics
  - Compromise between general-purpose and single-purpose processors
- Features
  - Program memory
  - Optimized datapath
  - Special functional units
- Benefits
  - Some flexibility, good performance, size and power
- DSP จัดอยู่ในประเภทนี้ด้วย



Slide credit Vahid/Givargis, Embedded Systems Design: A Unified Hardware/Software Introduction, 2000

Criteria	General Purpose Computer	Embedded system
Contents	It is combination of generic hardware and a general purpose OS for executing a variety of applications.	It is combination of special purpose hardware and embedded OS for executing specific set of applications
Operating System	It contains general purpose operating system	It may or may not contain operating system.
Alterations	Applications are alterable by the user.	Applications are non-alterable by the user.
Key factor	Performance" is key factor.	Application specific requirements are key factors.
Power Consumption	More	Less
Response Time	Not Critical	Critical for some applications

QUALITY ATTRIBUTES OF EMBEDDED SYSTEM

These are the attributes that together form the deciding factor about the quality of an embedded system.
There are two types of quality attributes are:-

•**Operational Quality Attributes.**

1.These are attributes related to operation or functioning of an embedded system. The way an embedded system operates affects its overall quality.

•**Non-Operational Quality Attributes.**

1.These are attributes **not** related to operation or functioning of an embedded system. The way an embedded system operates affects its overall quality.
2.These are the attributes that are associated with the embedded system before it can be put in operation.

Operational Attributes

a) Response

• Response is a measure of quickness of the system.
•It gives you an idea about how fast your system is tracking the input variables.
•Most of the embedded system demand fast response which should be real-time.

b) Throughput

•Throughput deals with the efficiency of system.
• It can be defined as rate of production or process of a defined process over a stated period of time.
• In case of card reader like the ones used in buses, throughput means how much transaction the reader can perform in a minute or hour or day.

Reliability

Reliability is a measure of how much percentage you rely upon the proper functioning of the system .

Mean Time between failures and Mean Time To Repair are terms used in defining system reliability.

Mean Time between failures can be defined as the average time the system is functioning before a failure occurs.

Mean time to repair can be defined as the average time the system has spent in repairs.

Maintainability

Maintainability deals with support and maintenance to the end user or a client in case of technical issues and product failures or on the basis of a routine system checkup

It can be classified into two types

I. Scheduled or Periodic Maintenance
II. Maintenance to unexpected failure

Security

•Confidentiality, Integrity and Availability are three corner stones of information security.

•Confidentiality deals with protection data from unauthorized disclosure.

•Integrity gives protection from unauthorized modification.

•Availability gives protection from unauthorized user

•Certain Embedded systems have to make sure they conform to the security measures.

•Ex. An Electronic Safety Deposit Locker can be used only with a pin number like a password.

Safety

Safety deals with the possible damage that can happen to the operating person and environment due to the breakdown of an embedded system or due to the emission of hazardous materials from the embedded products.

Non Operational Attributes

Testability and Debug-ability

•It deals with how easily one can test his/her design, application and by which mean he/she can test it.

•In hardware testing the peripherals and total hardware function in designed manner

•Firmware testing is functioning in expected way

•Debug-ability is means of debugging the product as such for figuring out the probable sources that create unexpected behavior in the total system

Evolvability

For embedded system, the qualitative attribute "Evolvability" refer to ease with which the embedded product can be modified to take advantage of new firmware or hardware technology.

Portability

•Portability is measured of “system Independence”.

•An embedded product can be called portable if it is capable of performing its operation as it is intended to do in various environments irrespective of different processor and or controller and embedded operating systems.

Time to prototype and market

•Time to Market is the time elapsed between the conceptualization of a product and time at which the product is ready for selling or use

•Product prototyping help in reducing time to market.

•Prototyping is an informal kind of rapid product development in which important feature of the under consider are develop.

•In order to shorten the time to prototype, make use of all possible option like use of reuse, off the self component etc.

Per unit and total cost

•Cost is an important factor which needs to be carefully monitored. Proper market study and cost benefit analysis should be carried out before taking decision on the per unit cost of the embedded product.

•When the product is introduced in the market, for the initial period the sales and revenue will be low

•There won’t be much competition when the product sales and revenue increase.

UNIT II
TYPICAL EMBEDDED SYSTEM

Core of the Embedded Systems:-

Embedded systems are domain and application specific and are built around a central core. The core of the embedded system falls into any one of the following categories.

1. General Purpose and Domain Specific Processors
   - 1.1 Microprocessors
   - 1.2 Microcontrollers
   - 1.3 Digital Signal Processors
2. Application Specific Integrated Circuits (ASICs)
3. Programmable Logic Devices (PLDs)
4. Commercial Of The Shelf Component (COTS)

1. General Purpose and Domain Specific Processors

1.1 Microprocessors

1.2 Microcontrollers

1.3 Digital Signal Processors

- Almost 80% of Embedded systems are processor/Controller based. The processor may be a Microprocessor or a Micro-controller or a Digital signal Processor depending on domain and application.
- Most of the embedded system in the industrial control and monitoring applications make use of the commonly available microprocessors or microcontrollers.
- where as domains which require signal processing such as speech coding, speech reorganization, etc. make use of Digital signal processors supplied by manufactures like Analog Devices, Texas Instruments, etc.

2. Application Specific Integrated Circuits (ASICs)

- Application Specific Integrated Circuits (ASICs) is a micro chip 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 reduce s the system development cost.

3.Programmable Logic Devices (PLDs)

- Logic devices provides specific functions, including device to device interfacing, data communication, signal processing, data display, timing & control operations, and almost every other function a system must perform.
- $$\begin{tikzcd}[row sep=tiny] & \text{Fixed logic devices} \\ \text{Logic devices} \arrow[ur] \arrow[dr] & \\ & \text{Programmable Logic devices} \end{tikzcd}$$
- Fixed logic devices are permanent they perform one function or set of functions once manufactured, they cannot be changed.
- Programmable Logic devices offer customers a wide range of logic capacity, features, speed, and voltage characteristics and these devices can be re-configured to perform any

4. Commercial Of The Shelf Component (COTS)

- A Commercial Off the Shelf product is one which is used "as-is".
- COTS products are designed in such a way to provide easy integration and interoperability with existing system components.
- The COTS component itself may be developed around a general purpose or domain specific processor or an Application Specific Integrated Circuit or a Programmable Logic Device.
- Typical Examples of COTS hardware unit is remote controlled toy car controlled units including RF circuitry part, high performance, high frequency microwave electronics (2-200GHz),electro-optic IR imaging arrays, UV/IR detectors, etc..
- The major advantage of using COTS is that they are readily available in the market.

Sensors and Actuators

Sensor:-

A sensor is a transducer device that converts energy from one form to another for any measurement or control purpose.

Actuator:-

Actuator is a form of transducer device which converts signals to corresponding physical action(motion). Actuator act as output device

COMMUNICATION INTERFACES

For any embedded system, the communication interfaces can broadly classified into:

Onboard Communication Interfaces

These are used for internal communication of the embedded system i.e: communication between different components present on the system.

Common examples of onboard interfaces are:

- Inter Integrated Circuit (I2C)
- Serial Peripheral Interface (SPI)
- Universal Asynchronous Receiver Transmitter (UART)
- 1-Wire Interface
- Parallel Interface

Example :Inter Integrated Circuit (I2C)

- It is synchronous
- Bi-directional, half duplex , two wire serial interface bus
- Developed by Phillips semiconductors in 1980

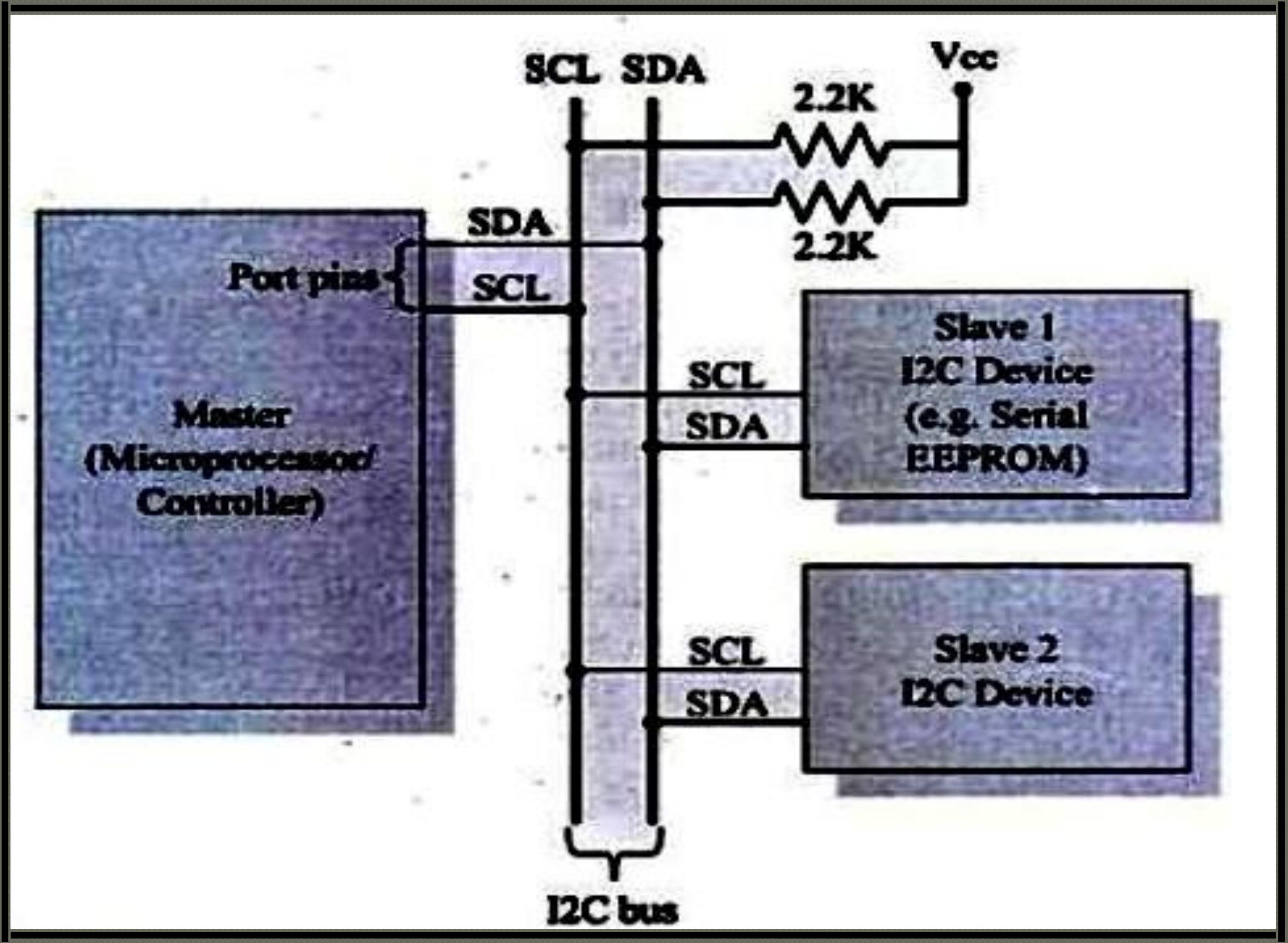

SCL SDA
Vcc
2.2K
2.2K
SDA
Port pins
SCL
Master (Microprocessor/ Controller)
SCL
SDA
Slave 1 I2C Device (e.g. Serial EEPROM)
SCL
SDA
Slave 2 I2C Device
I2C bus

Figure: I2C Bus Interfacing

External or Peripheral Communication Interfaces

These are used for external communication of the embedded system i.e: communication of different components present on the system with external or peripheral components/devices.

Common examples of external interfaces are:

- RS-232 C & RS-485
- Universal Serial Bus (USB)
- IEEE 1394 (Firewire)
- Infrared (IrDA)
- Bluetooth
- Wi-Fi
- Zig Bee
- General Packet Radio Service (GPRS)

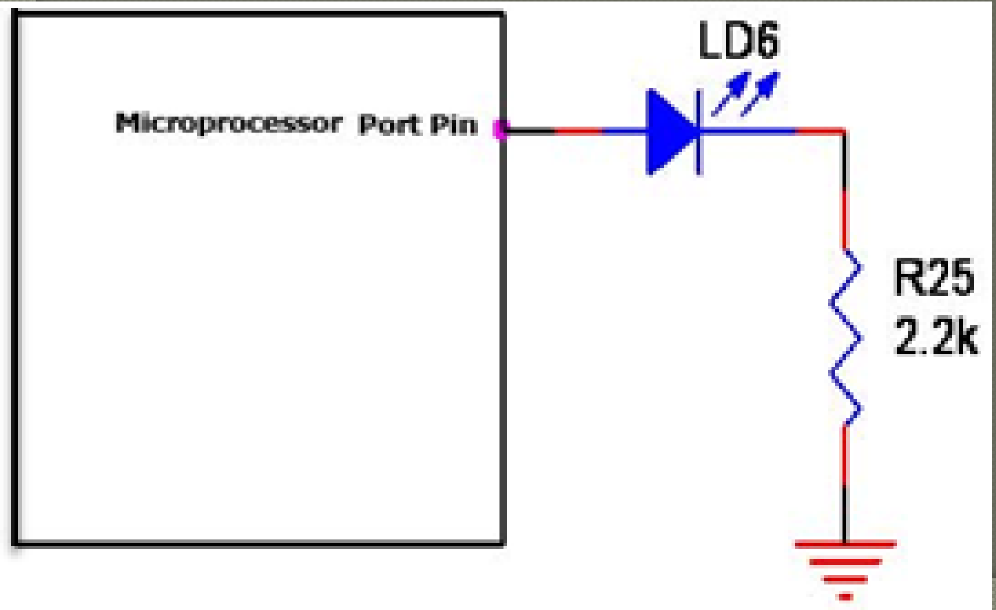
Example: RS-232 C & RS-485

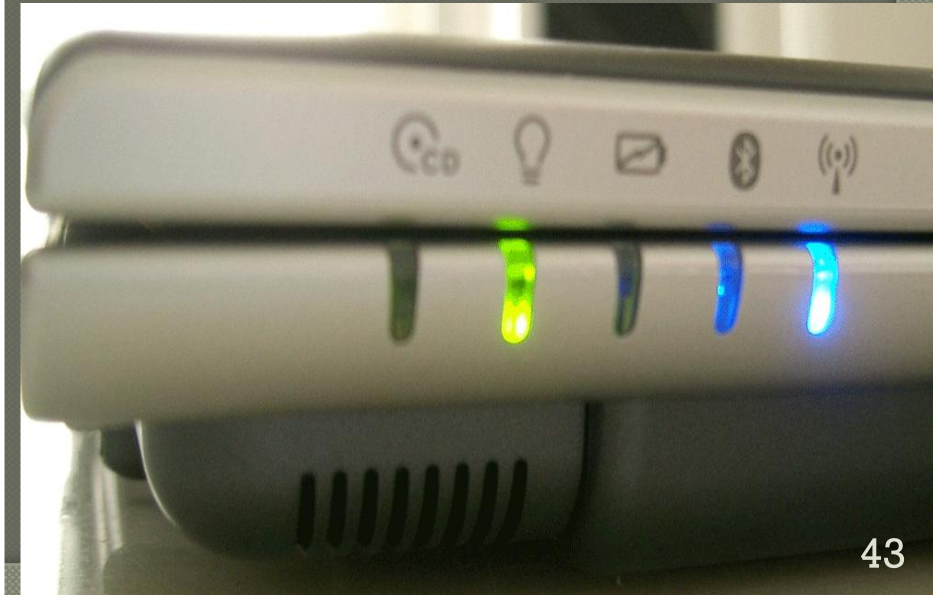
The I/O Subsystem

- Examples for some of the sensors & Actuators used in embedded system.
  - LED
  - 7 segment LED display
  - Optocoupler
  - Stepper motor
  - Relay
  - Piezo Buzzer
  - Push button switch
  - Keyboard
  - Programmable Peripheral Interface (PPI)

LED (Light Emitting Diode):-

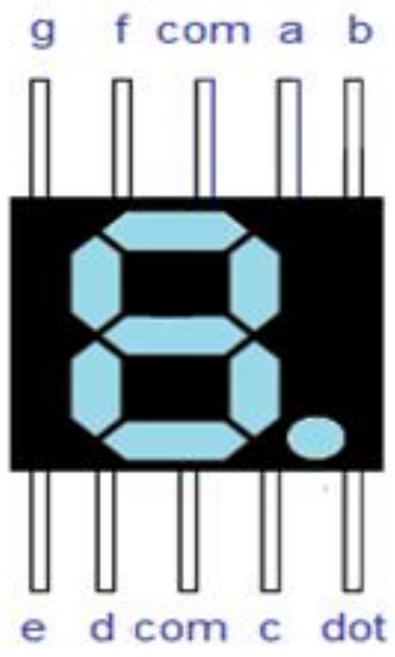
- ❖ It is an important output device for visual indications in any embedded system.
- ❖ LED can be used as an indicator for the status of various signals or situations.
- ❖ Typical examples are indicating the presence of power conditions like 'Device ON', 'Battery low', or ' Charging of Battery' for battery operated handheld embedded devices.

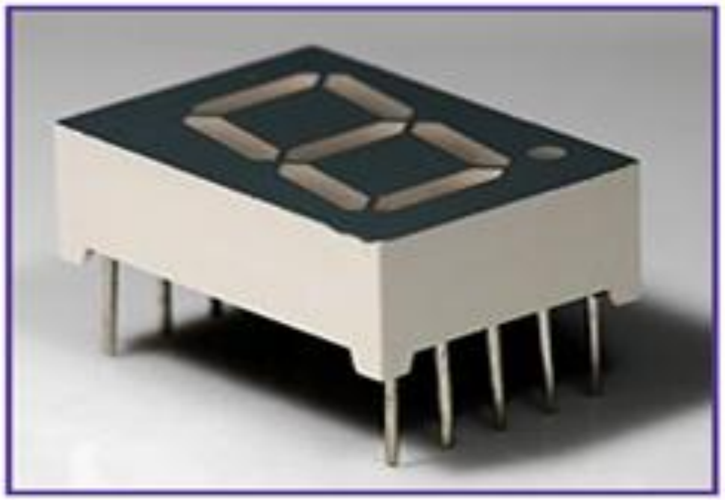




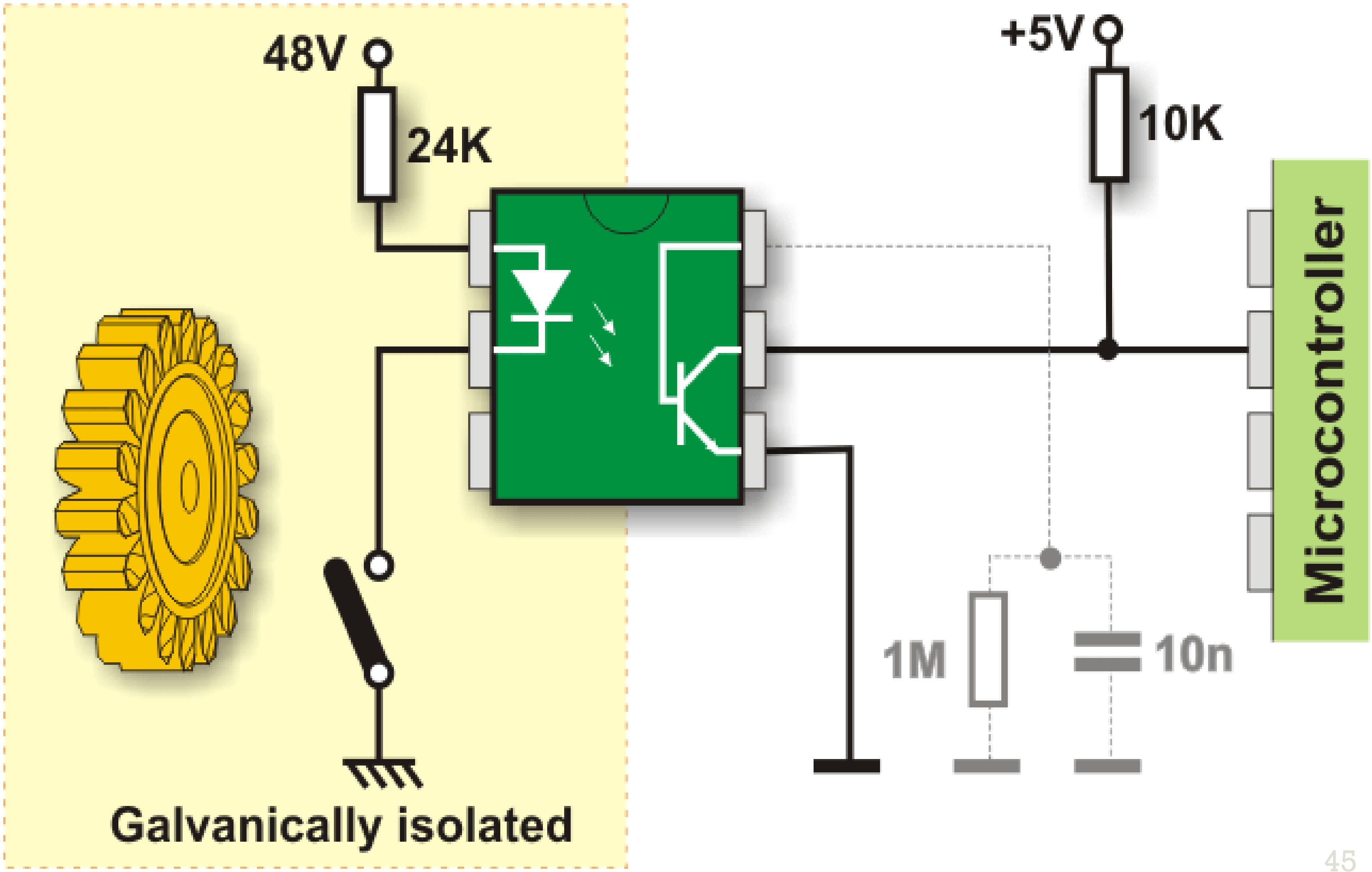
7 segment LED display:-

- It is an output device for displaying alpha numeric characters.
- It contains 8 light emitting diode (LED) segments arranged in a special form.
- Out of 8 LED segments 7 are used for displaying alpha numeric characters and 1 LED is used for representing 'decimal point' in decimal numbers.





Optocoupler:-



Communication Interface

1. On board Communication Interface or

   (Device/Board level communication interface)

2. External Communication Interface or

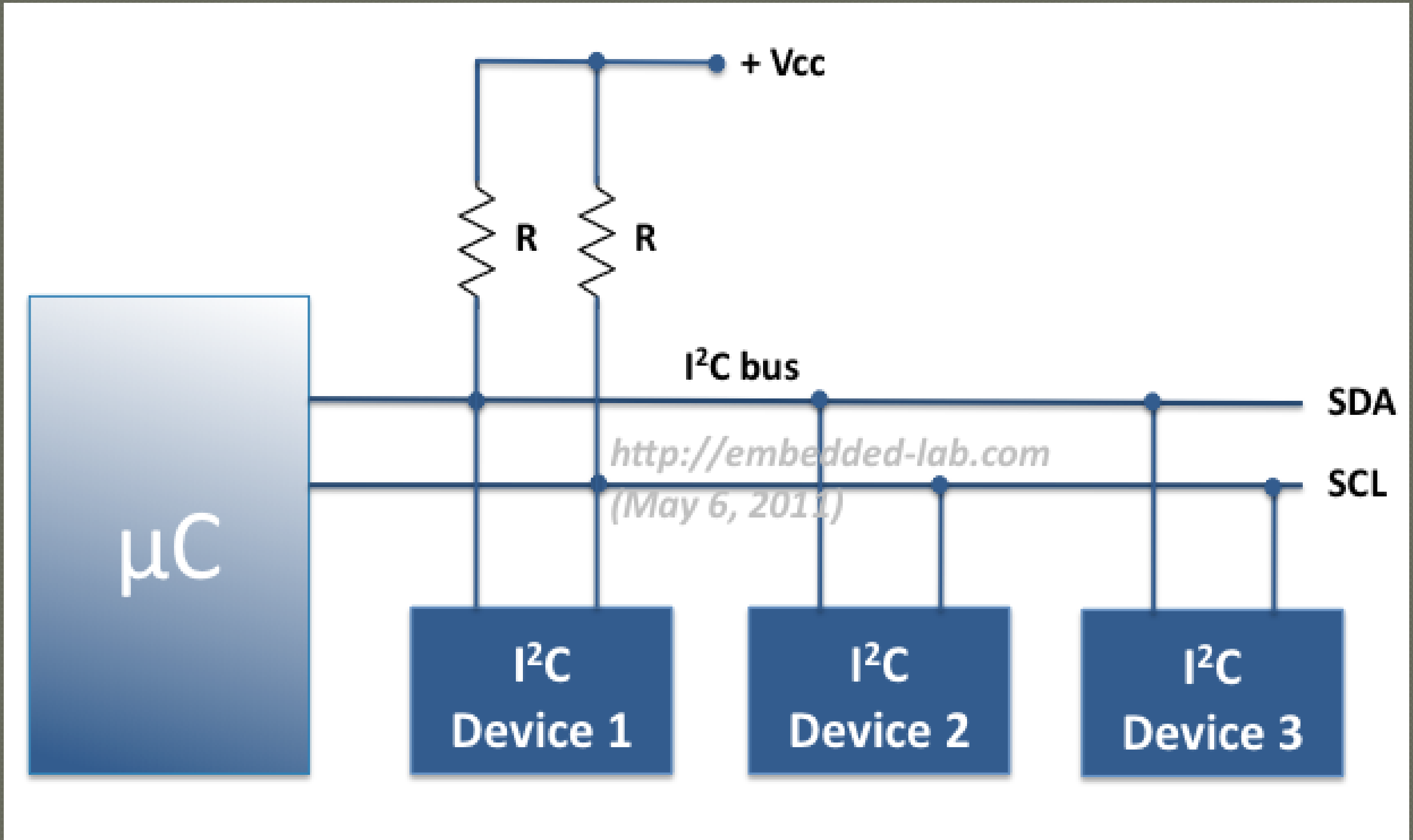
   (Product level communication interface)

1. On board Communication Interface or

(Device/Board level communication interface)

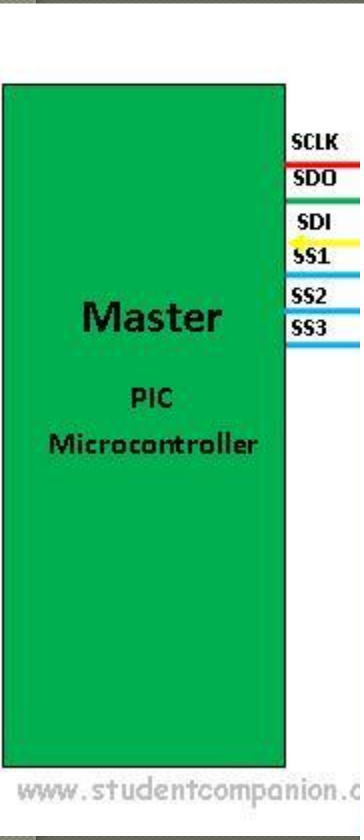
a) I2C Inter Integrated Circuit
b) SPI (Serial Communication Interface)
c) UART (Universal Asynchronous Rx and Tx)
d) 1-WIRE
e) Parallel Communication Interface

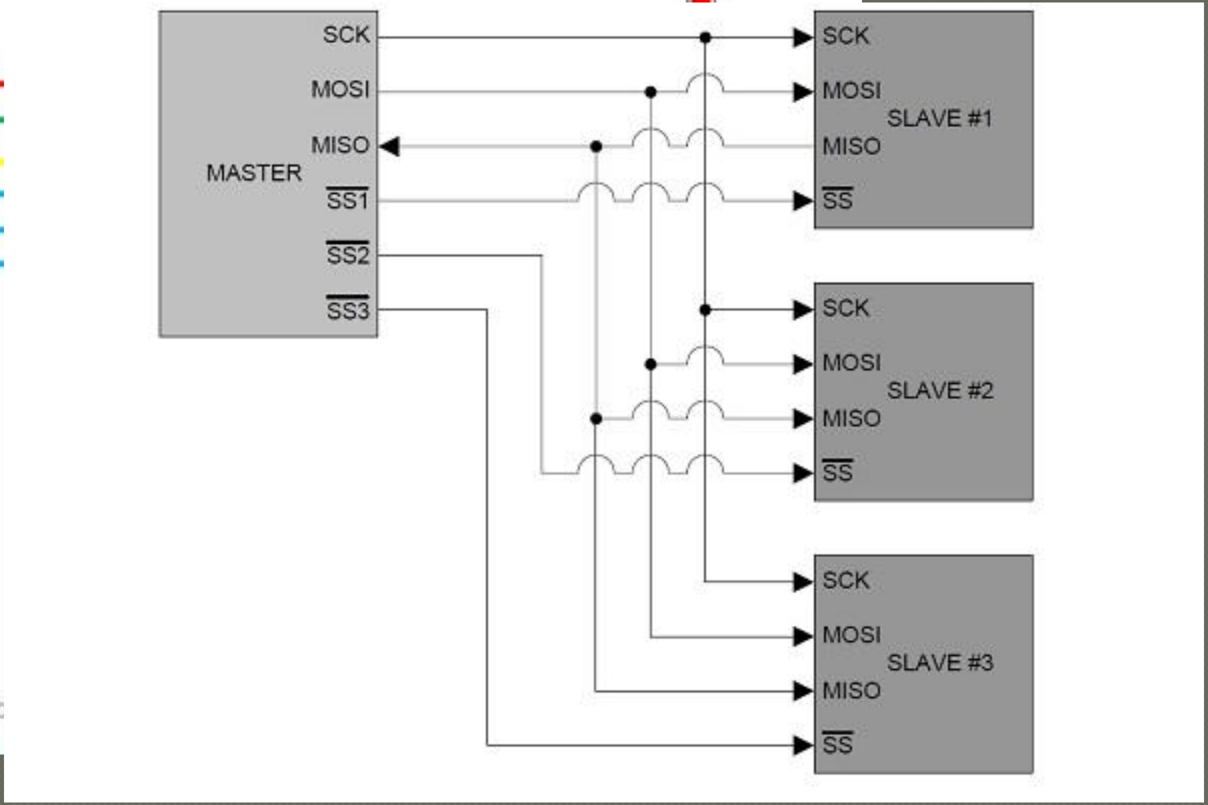
a) I2C Inter Integrated Circuit



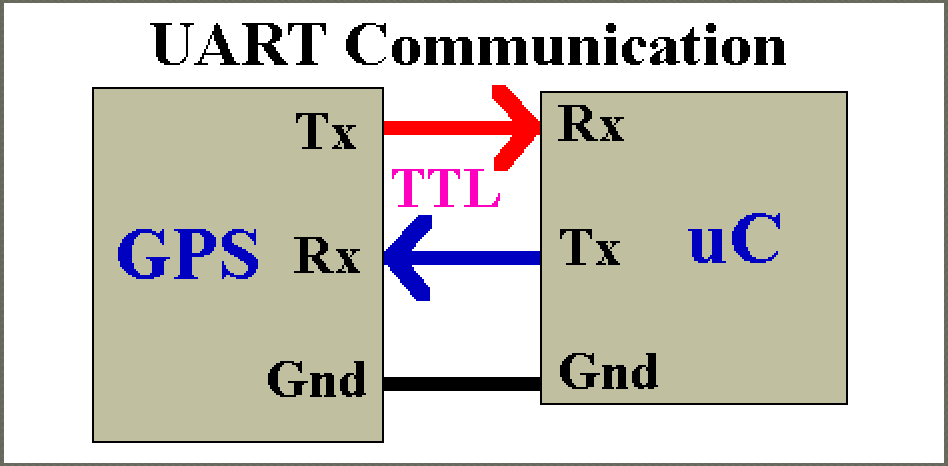
Multiple devices on common I²C bus

b) SPI (Serial Communication Interface)

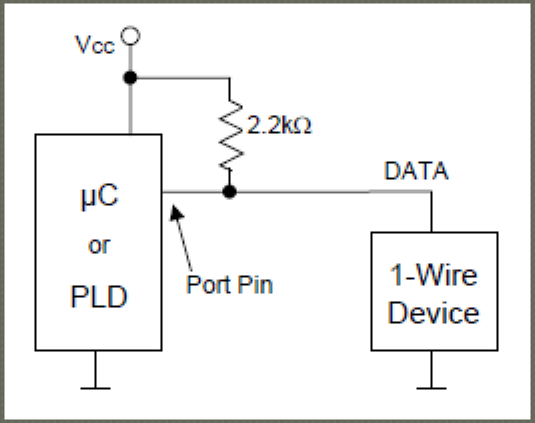




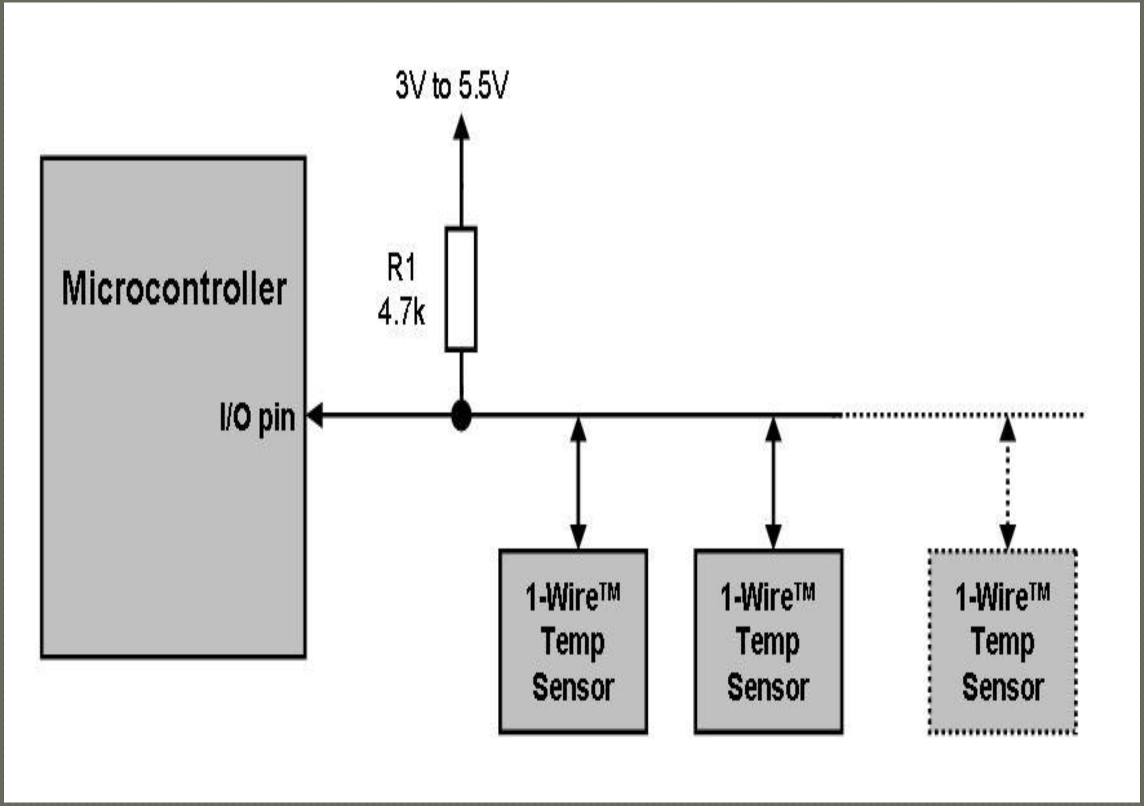
c) UART (Universal Asynchronous Rx and Tx)



d) 1-WIRE

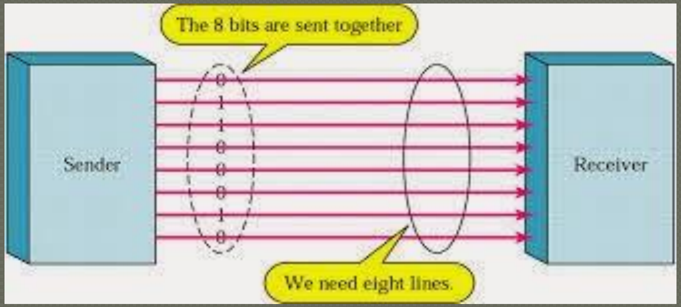


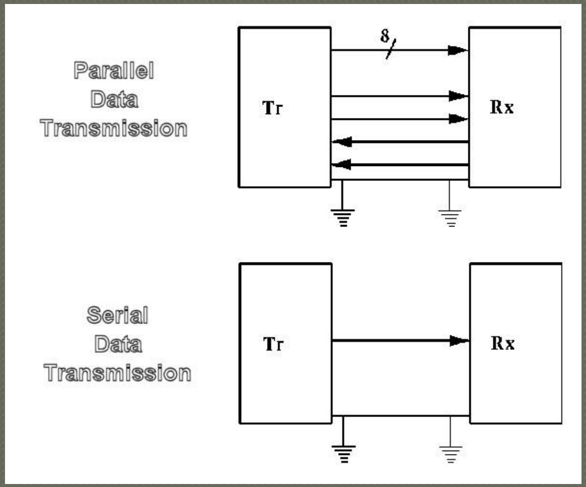
Vcc
2.2kΩ
DATA
µC
or
PLD
Port Pin
1-Wire
Device



3V to 5.5V
Microcontroller
R1
4.7k
I/O pin
1-Wire™ Temp Sensor
1-Wire™ Temp Sensor
1-Wire™ Temp Sensor

e) Parallel Communication Interface

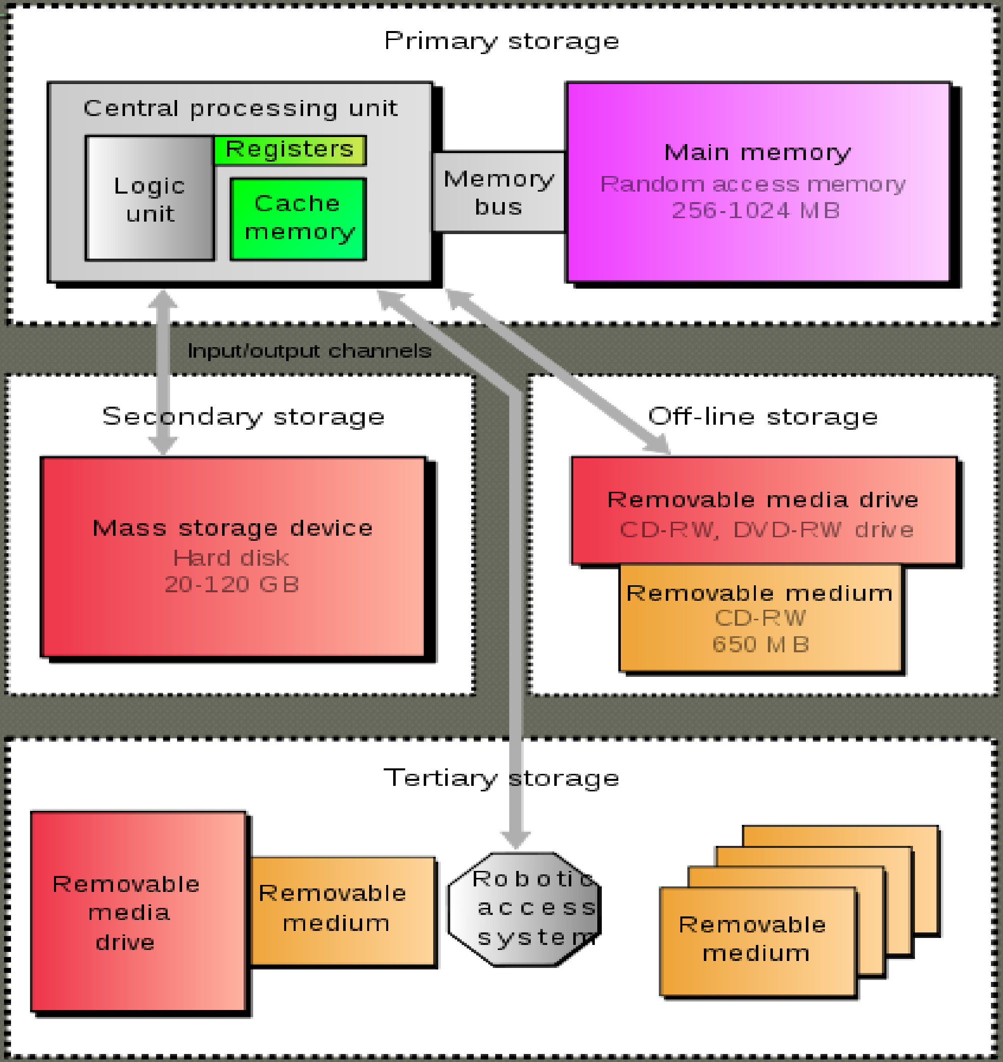




Memory

Memory Types

I. Secondary Memory
II. Primary Memory
   a) RAM
      i. SRAM
      ii. DRAM
   b) ROM
      i. PROM
      ii. EPROM
   c) Hybrid
      i. EEPROM
      ii. NVRAM
      iii. Flash Memory
   d) Cache Memory
   e) Virtual Memory



Secondary Memory

- The computer usually uses its input/output channels to access secondary storage and transfers the desired data using intermediate area in primary storage. Secondary storage does not lose the data when the device is powered down—it is non-volatile. Per unit, it is typically also an order of magnitude less expensive than primary storage.
- The secondary storage is often formatted according to a file system format, which provides the abstraction necessary to organize data into files and directories, providing also additional information (called metadata) describing the owner of a certain file, the access time, the access permissions, and other information. Hard disk are usually used as secondary storage.

Primary Memory

- Primary storage (or main memory or internal memory), often referred to simply as memory, is the only one directly accessible to the CPU. The CPU continuously reads instructions stored there and executes them as required.
- Main memory is directly or indirectly connected to the CPU via a *memory bus*. It is actually two buses (not on the diagram): an address bus and a data bus. The CPU firstly sends a number through an address bus, a number called memory address, that indicates the desired location of data. Then it reads or writes the data itself using the data bus.
- It is divided into RAM and ROM.

RAM

The RAM family includes two important memory devices: static RAM (SRAM) and dynamic RAM (DRAM). The primary difference between them is the lifetime of the data they store.

1) SRAM retains its contents as long as electrical power is applied to the chip. If the power is turned off or lost temporarily, its contents will be lost forever.

2) DRAM, on the other hand, has an extremely short data lifetime-typically about four milliseconds. This is true even when power is applied constantly. DRAM controller is used to refresh the data before it expires, the contents of memory can be kept alive for as long as they are needed. So DRAM is as useful as SRAM after all.

Types of RAM

Double Data Rate synchronous dynamic random access memory or also known as **DDR1 SDRAM** is a class of memory integrated circuits used in computers. The interface uses double pumping (transferring data on both the rising and falling edges of the clock signal) to lower the clock frequency. One advantage of keeping the clock frequency down is that it reduces the signal integrity requirements on the circuit board connecting the memory to the controller.

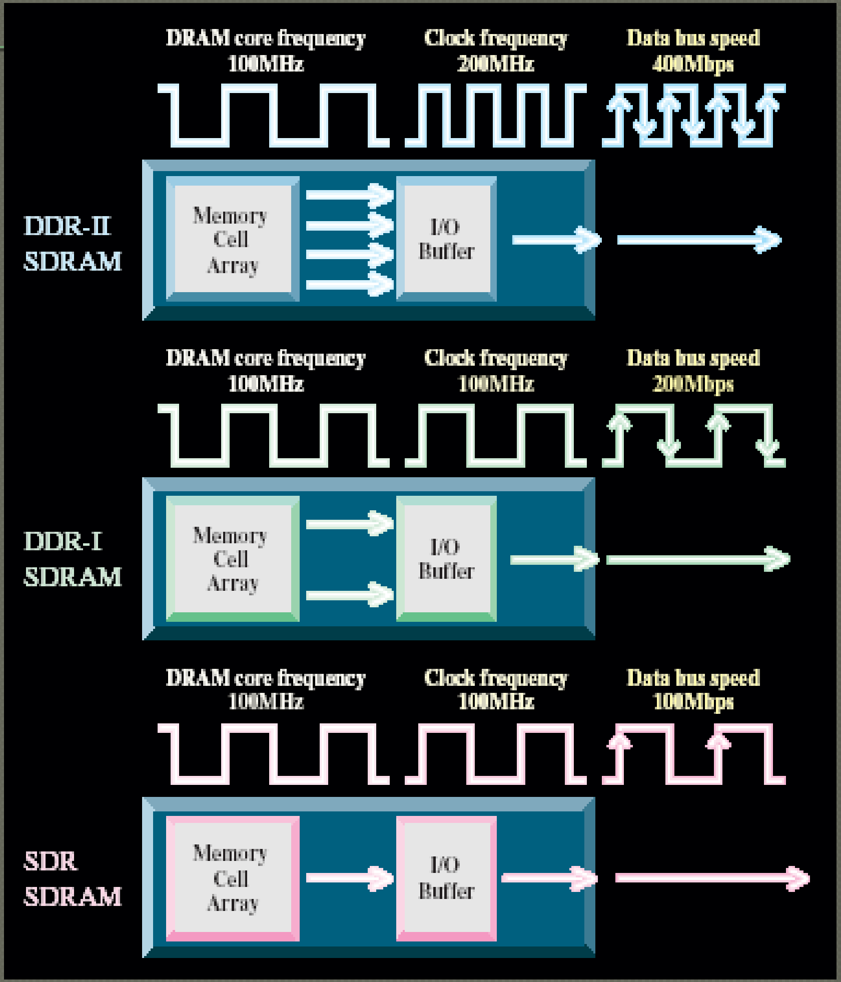
DDR2, DDR and SDRAM

✓DDR2 memory is fundamentally similar to DDR SDRAM. Still, while DDR SDRAM can transfer data across the bus two times per clock, DDR2 SDRAM can perform four transfers per clock. DDR2 uses the same memory cells, but doubles the bandwidth by using the multiplexing technique.

✓The DDR2 memory cell is still clocked at the same frequency as DDR SDRAM and SDRAM cells, but the frequency of the input/output buffers is higher with DDR2 SDRAM (as shown in Fig. on next Slide). The bus that connects the memory cells with the buffers is twice wider compared to DDR.

✓Thus, the I/O buffers perform multiplexing: the data is coming in from the memory cells along a wide bus and is going out of the buffers on a bus of the same width as in DDR SDRAM, but of a twice bigger frequency. This allows to increase the memory bandwidth without increasing the operational frequency.

- The interface uses double pumping (transferring data on both the rising and falling edges of the clock signal to lower the clock frequency.
- One advantage of keeping the clock frequency down is that it reduces the signal integrity requirements on the circuit board connecting the memory to the controller.



Types of ROM

- Memories in the ROM family are distinguished by the methods used to write new data to them (usually called programming), and the number of times they can be rewritten.
- This classification reflects the evolution of ROM devices from hardwired to programmable to erasable-and-programmable. A common feature is their ability to retain data and programs forever, even during a power failure.
- The contents of the ROM had to be specified before chip production, so the actual data could be used to arrange the transistors inside the chip.

PROM

- ✓ One step up from the masked ROM is the PROM (programmable ROM), which is purchased in an unprogrammed state. If you were to look at the contents of an unprogrammed PROM, the data is made up entirely of 1's.
- ✓ The process of writing your data to the PROM involves a special piece of equipment called a device programmer. The device programmer writes data to the device one word at a time by applying an electrical charge to the input pins of the chip.
- ✓ Once a PROM has been programmed in this way, its contents can never be changed. If the code or data stored in the PROM must be changed, the current device must be discarded. As a result, PROMs are also known as one-time programmable (OTP) devices.

EPROM

- An EPROM (erasable-and-programmable ROM) is programmed in exactly the same manner as a PROM. However, EPROMs can be erased and reprogrammed repeatedly.
- To erase an EPROM, you simply expose the device to a strong source of ultraviolet light. (A window in the top of the device allows the light to reach the silicon.)
- By doing this, you essentially reset the entire chip to its initial-un programmed-state. Though more expensive than PROMs, their ability to be reprogrammed makes EPROMs an essential part of the software development and testing process.

Hybrid types

- ✓ As memory technology has matured in recent years, the line between RAM and ROM has blurred. Now, several types of memory combine features of both.
- ✓ These devices do not belong to either group and can be collectively referred to as hybrid memory devices. Hybrid memories can be read and written as desired, like RAM, but maintain their contents without electrical power, just like ROM.
- ✓ Two of the hybrid devices, EEPROM and flash, are descendants of ROM devices. These are typically used to store code. The third hybrid, NVRAM, is a modified version of SRAM. NVRAM usually holds persistent data.

- **EEPROMS** are electrically-erasable-and-programmable. Internally, they are similar to EPROMs, but the erase operation is accomplished electrically, rather than by exposure to ultraviolet light. Any byte within an EEPROM may be erased and rewritten.
- Once written, the new data will remain in the device forever-or at least until it is electrically erased. The primary tradeoff for this improved functionality is higher cost, though write cycles are also significantly longer than writes to a RAM. So you wouldn't want to use an EEPROM for your main system memory.

- **Flash memory** combines the best features of the memory devices described thus far. Flash memory devices are high density, low cost, nonvolatile, fast (to read, but not to write), and electrically reprogrammable. These advantages are overwhelming and, as a direct result, the use of flash memory has increased dramatically in embedded systems. From a software viewpoint, flash and EEPROM technologies are very similar. The major difference is that flash devices can only be erased one sector at a time, not byte-by-byte. Typical sector sizes are in the range 256 bytes to 16KB. Despite this disadvantage, flash is much more popular than EEPROM and is rapidly displacing many of the ROM devices as well.

- The third member of the hybrid memory class is NVRAM (non-volatile RAM). Non volatility is also a characteristic of the ROM and hybrid memories discussed previously. However, an NVRAM is physically very different from those devices. An NVRAM is usually just an SRAM with a battery backup.
- When the power is turned on, the NVRAM operates just like any other SRAM. When the power is turned off, the NVRAM draws just enough power from the battery to retain its data. NVRAM is fairly common in embedded systems.
- However, it is expensive-even more expensive than SRAM, because of the battery-so its applications are typically limited to the storage of a few hundred bytes of system-critical information that can't be stored in any better way.

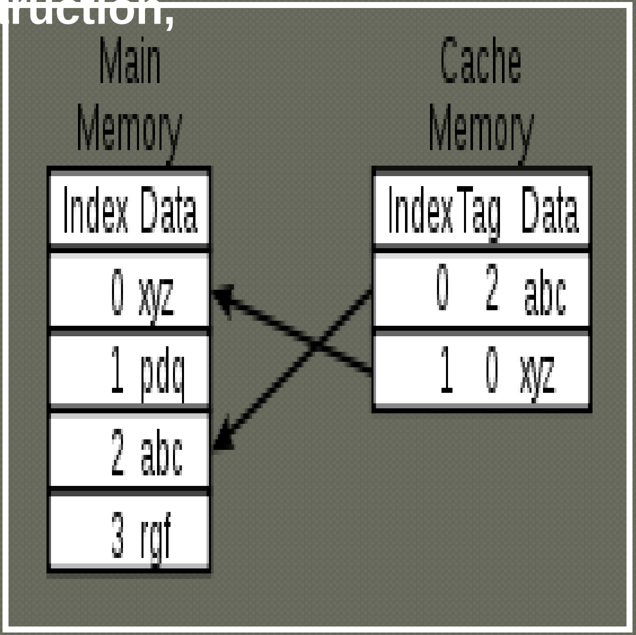
Cache Memory

- A CPU cache is a cache used by the central processing unit of a computer to reduce the average time to access memory. The cache is a smaller, faster memory which stores copies of the data from the most frequently used main memory locations. As long as most memory accesses are cached memory locations, the average latency of memory accesses will be closer to the cache latency than to the latency of main memory.
- When the processor needs to read from or write to a location in main memory, it first checks whether a copy of that data is in the cache. If so, the processor immediately reads from or writes to the cache, which is much faster than reading from or writing to main memory

Cache Memory

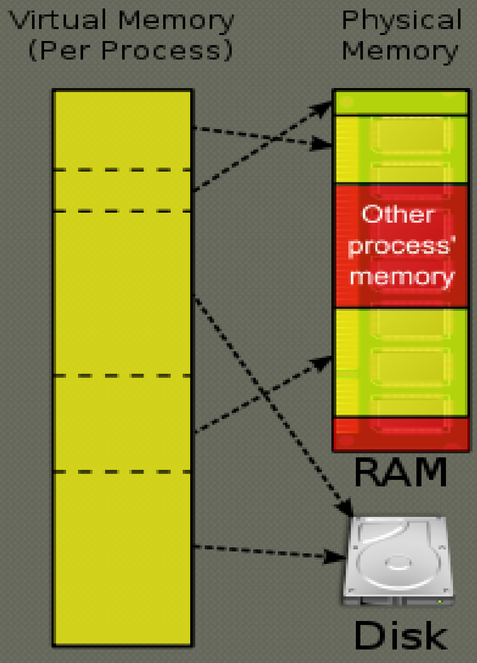
The diagram on the right shows two memories. Each location in each memory has a datum (a *cache line*), which in different designs ranges in size from 8 to 512 bytes. The size of the cache line is usually larger than the size of the usual access requested by a CPU instruction,

- which ranges from 1 to 16 bytes.
- Each location in each memory also has an index, which is a unique number used to refer to that location.The index for a location in main memory is called an address.
- Each location in the cache has a tag that contains the index of the datum in main memory that has been cached. In a CPU's data cache these entries are called *cache lines* or *cache blocks*.



Virtual Memory

- It is a computer system technique which gives an application program the impression that it has contiguous working memory (an address space), while in fact it may be physically fragmented and may even overflow on to disk storage.
- computer operating systems generally use virtual memory techniques for ordinary applications, such as word processors, spreadsheets,multimedia,players accounting, etc., except where the required hardware support (memory management unit) is unavailable or insufficient.



Characteristics of the various memory types

Type	Volatile?	Writeable?	Erase Size	Max Erase Cycles	Cost (per Byte)	Speed
SRAM	Yes	Yes	Byte	Unlimited	Expensive	Fast
DRAM	Yes	Yes	Byte	Unlimited	Moderate	Moderate
Masked ROM	No	No	n/a	n/a	Inexpensive	Fast
PROM	No	Once, with a device programmer	n/a	n/a	Moderate	Fast
EPROM	No	Yes, with a device programmer	Entire Chip	Limited (consult datasheet)	Moderate	Fast
EEPROM	No	Yes	Byte	Limited (consult datasheet)	Expensive	Fast to read, slow to erase/write
Flash	No	Yes	Sector	Limited (consult datasheet)	Moderate	Fast to read, slow to erase/write
NVRAM	No	Yes	Byte	Unlimited	Expensive (SRAM + battery)	

UNIT III
EMBEDDED FIRMWARE

RTOS v/s General Purpose OS

1. Determinism
2. Task Scheduling
3. Preemptive kernel
4. Priority Inversion
5. Usage

Sophisticated Embedded System Characteristics

(1) Dedicated functions

(2) Dedicated complex algorithms

(3)Dedicated (GUIs) and other user interfaces for the application

(4) Real time operations— Defines the ways in which the system works, reacts to the events and interrupts, schedules the system functioning in real time and executes by following a plan to control the latencies and to meet the deadlines.

Contd..

(5) Multi-rate operations -Different operations may take place at distinct rates. For example, the audio, video, network data or stream and events have the different rates and time constraints to finish associated processes..

Constraints of an Embedded System Design

- Available system-memory
- Available processor speed
- Limited power dissipation when running the system continuously in cycles of the system start, wait for event, wake-up and run, sleep and stop.

- Performance
- power
- size
- non-recurring design cost, and manufacturing costs

PROCESSOR IN EMBEDDED SYSTEM

- Processor is the heart of embedded system

- Processor has two essential units:

    Control Unit(CU)
    Execution Unit(EU)

- Program Flow and data path (CU) **Control Unit**—includes a fetch unit for fetching instructions from the memory.

- **Execution Unit (EU)** —includes circuits for arithmetic and logical unit (ALU), and for instructions for a program control task, say, data transfer instructions, halt, interrupt, or jump to another set of instructions or call to another routine or sleep or reset.

A Processor is in the form of an **IC** or it could be in core form in an Application Specific Integrated Circuit **(ASIC)** or System on Chip (**SoC**).

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

1. General Purpose Processor (GPP) : instruction set designed not specific to the applications.

example: Microprocessor

- A Microprocessor is a single VLSI chip that has a CPU and has some other units such as caches, pipelining and super scaling units.

- is an essential part of a computing system.

Intel 80x86 family, ARM, 68HCxxx family.

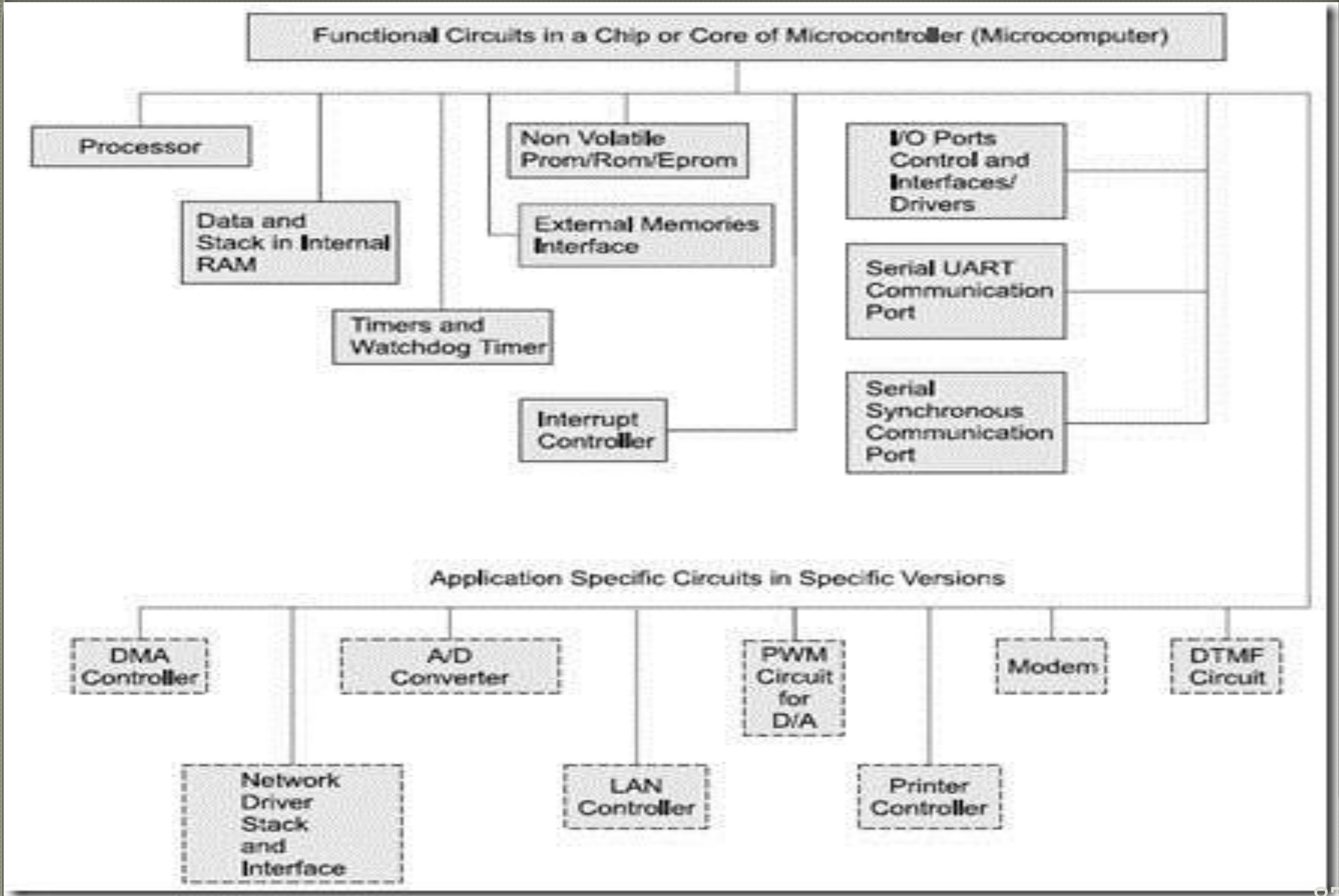
2. Application Specific Instruction-set Processor (ASIP): is a processor with instruction set designed for specific applications on a VLSI chip.

Examples: micro controller
embedded micro controller
DSP and media processor
Network processor
IO processor or
domain-specific programmable processor

- A **Microcontroller** is an integrated chip (IC) that has a processor, memory and several other hardware units in it such as timers, watchdog timer, interrupt controller, ADC or PWM .
- is an essential component of a control or communication circuit.

8051, 8051 MX, 68HC11xx, PIC18

Various functional circuits in a microcontroller chip:



Functional Circuits in a Chip or Core of Microcontroller (Microcomputer)
Processor
Data and Stack in Internal RAM
Timers and Watchdog Timer
Non Volatile Prom/Rom/Eprom
External Memories Interface
Interrupt Controller
I/O Ports Control and Interfaces/ Drivers
Serial UART Communication Port
Serial Synchronous Communication Port
Application Specific Circuits in Specific Versions
DMA Controller
Network Driver Stack and Interface
A/D Converter
LAN Controller
PWM Circuit for D/A
Printer Controller
Modem
DTMF Circuit

3. Single Purpose Processors or additional processors:

Examples:

- Coprocessors: used for graphic processing, floating point processing, encrypting, deciphering, discrete cosine transformation and inverse transformation on TCP/IP protocol stacking and networking connecting functions.

- Accelerator: Java codes accelerator

- Controllers: for peripherals, direct memory access and buses

4. GPP or ASIP cores integrated into either an ASIC or VLSI circuit or a Field Programmable Gate Array (FPGA) core integrated with processor units in a VLSI chip.

5. Application Specific system Processor (ASSP)

6.Multi core processors or multi processors.

Embedded hardware units and Devices in a system

1. Power source

- A power supply source or charge pump is essential in every system.
- The systems which do not have power supply of their own are powered by the use of charge pumps or they connect to external power supply.

2. Clock Oscillator circuit and Clocking units:

- the clock controls the time for executing an instruction.
- also controls the various clocking requirements of CPU, of system timers, CPU machine cycles.
- A processor needs a clock oscillator circuit to establish a reference frequency used for timing purposes.

3. System timers and Real-Time Clock(RTC):

- to schedule the various tasks and for real time programming , an RTC or system clock is needed.

- A timer circuit is configured as system clock , which ticks and generates interrupts periodically.

4. Reset circuit , Power-up Reset and Watchdog timer Reset :

- Reset circuit can change the Program Counter (PC) to a power-up default value.
- A program that is reset and runs on a power-up can be one of the following:

 i) system program that executes from the beginning.

ii) A system boot-up program

iii) A system initialization program

Reset:

Processor begins the processing of instructions from a starting address.

The reset circuit can be activated by any one of the following:

1. software instruction
2. time-out by a programmable timer known as watchdog timer.
3. a clock monitor detecting a slow down below certain frequencies.

Watchdog Timer:

A timing device that resets the system after a predefined timeout.

Helps in rescuing the system if a fault develops.

Memory:

a. Functions Assigned to the ROM or EPROM or Flash

1. Storing 'Application' program from where the processor fetches the instruction codes.

2. Storing codes for system booting, initializing, Initial input data and Strings.

3. Storing Codes for RTOS.

4. Storing Pointers (addresses) of various service routines.

b. Functions Assigned to the Internal, External and Buffer RAM:

1. Storing the variables during program run,
2. Storing the stacks,
3. Storing input or output buffers for example,

for speech or image .

c. Functions Assigned to the EEPROM or Flash :

- Storing non-volatile results of processing

d. Functions Assigned to the Caches:

1. Storing copies of the instructions, data and branch-transfer instructions in advance from external memories and

2. Storing temporarily the results in write back caches during fast processing

e. Functions Assigned to Memory Stick:

It stores high definition video, images, songs,
Or speeches after a suitable format compression
And stores large persistent data.

in digital camera, mobile computing system

UNIT IV
Real Time Operating Systems

Topics

- Real Time Systems
- Real Time Operating Systems & VxWorks
- Application Development
- Loading Applications
- Testing Applications

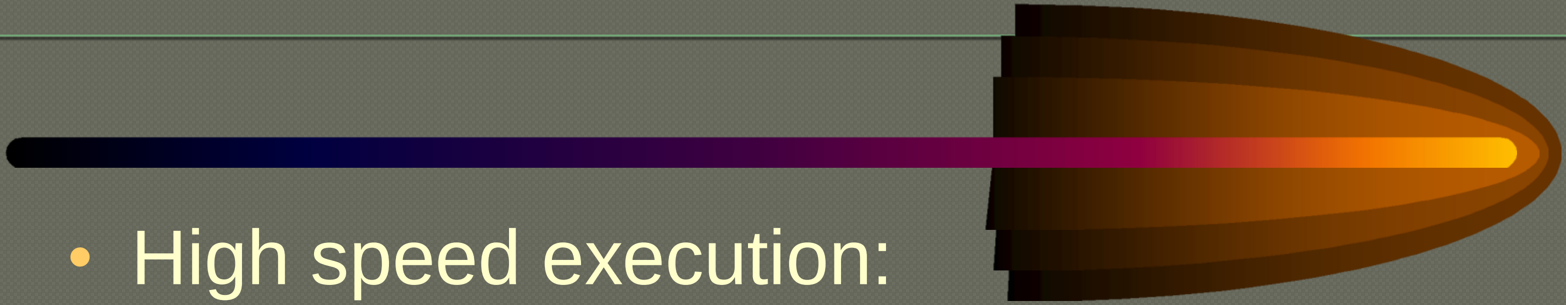
Real Time Systems

- Real-time is the ability of the control system to respond to any external or internal events in a fast and deterministic way.
- We say that a system is *deterministic* if the response time is predictable.

Real Time Systems

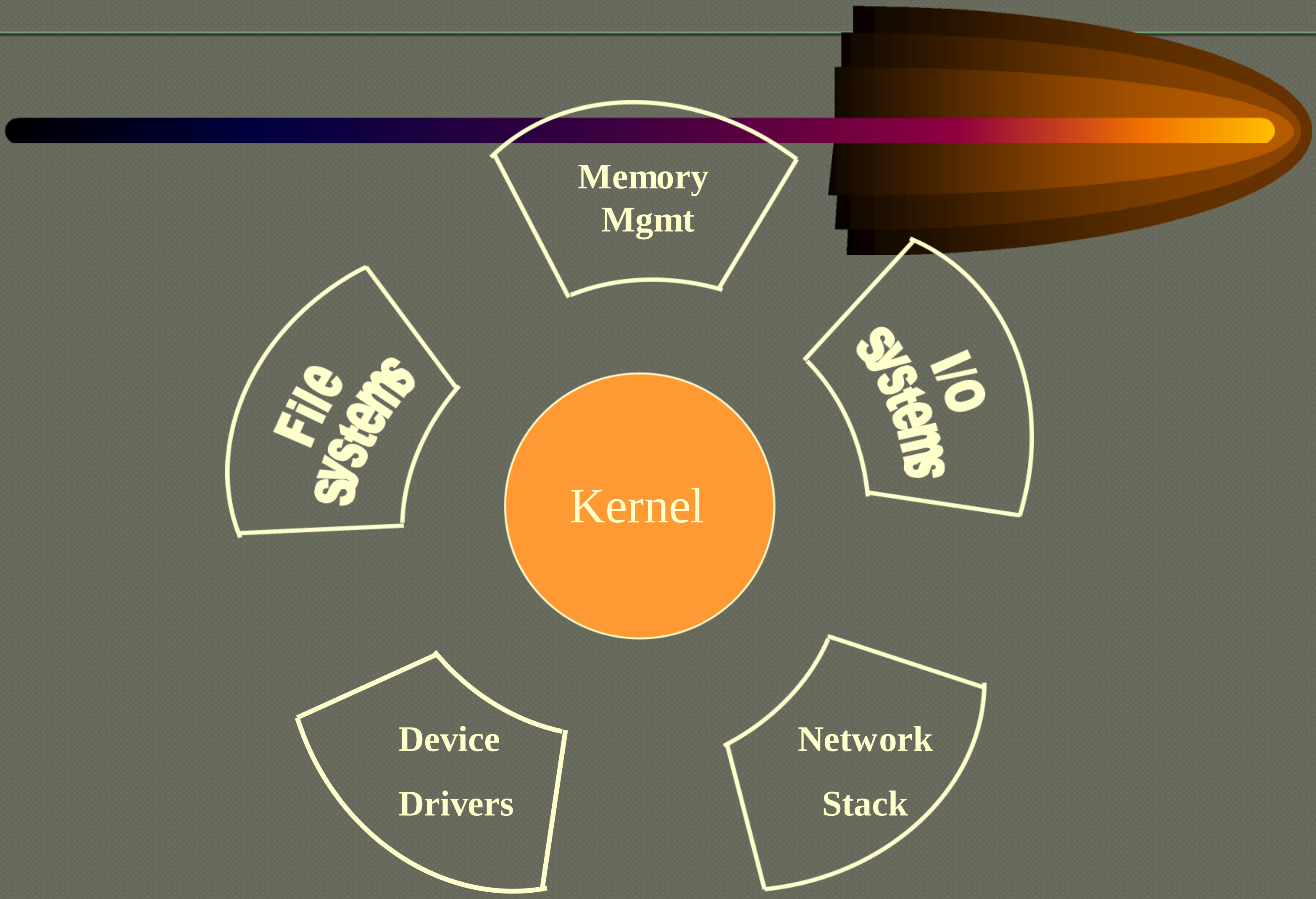
- The lag time between the occurrence of an event and the response to that event is called *latency*
- Deterministic performance is key to Real-time performance.

Real Time System



- High speed execution:
  - – Fast response
  - – Low overhead
- Deterministic operation:
  - – A late answer is a wrong answer.

Real Time Systems



vxWorks

- What is vxWorks ?
  - vxWorks is a networked RTOS which can also be used in distributed systems.
  - vxWorks topics
    - Hardware Environment Requirements
    - Development tools
    - Testing

Hardware requirements

- vxWorks runs on range of platforms
- MC680x0
- MC683xx
- Intel i960
- Intel i386
- R3000
- SPARC based systems

What is a real time OS

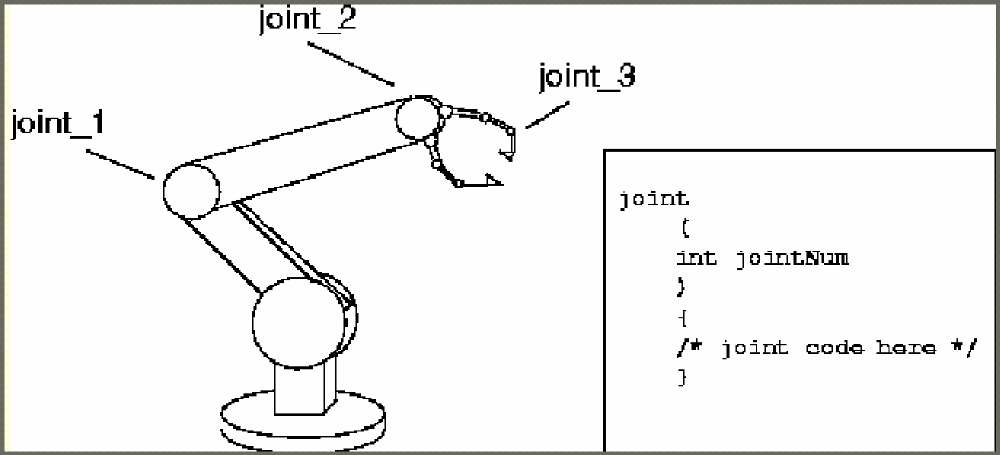
- A real time OS is a operating system which will help implement any real time system

RTOS Requirements

- Multitasking
- Intertask communications
- Deterministic response
- Fast Response
- Low Interrupt Latency

Uni tasking

- Sample Application



```
joint
    (
    int  jointNum
    )
    {
    /*  joint  code  here  */
    }
```

Uni tasking

- One task controlling all the components is a loop.
- arm ( )

```
  {
  for (;;)
      {
        if (shoulder needs moving)
                    moveShoulder( ) ;
        if (elbow needs moving)
                    moveElbow( );
        if (wrist need moving)
                    moveWrist( );
                    . . . .
      }
   }
```

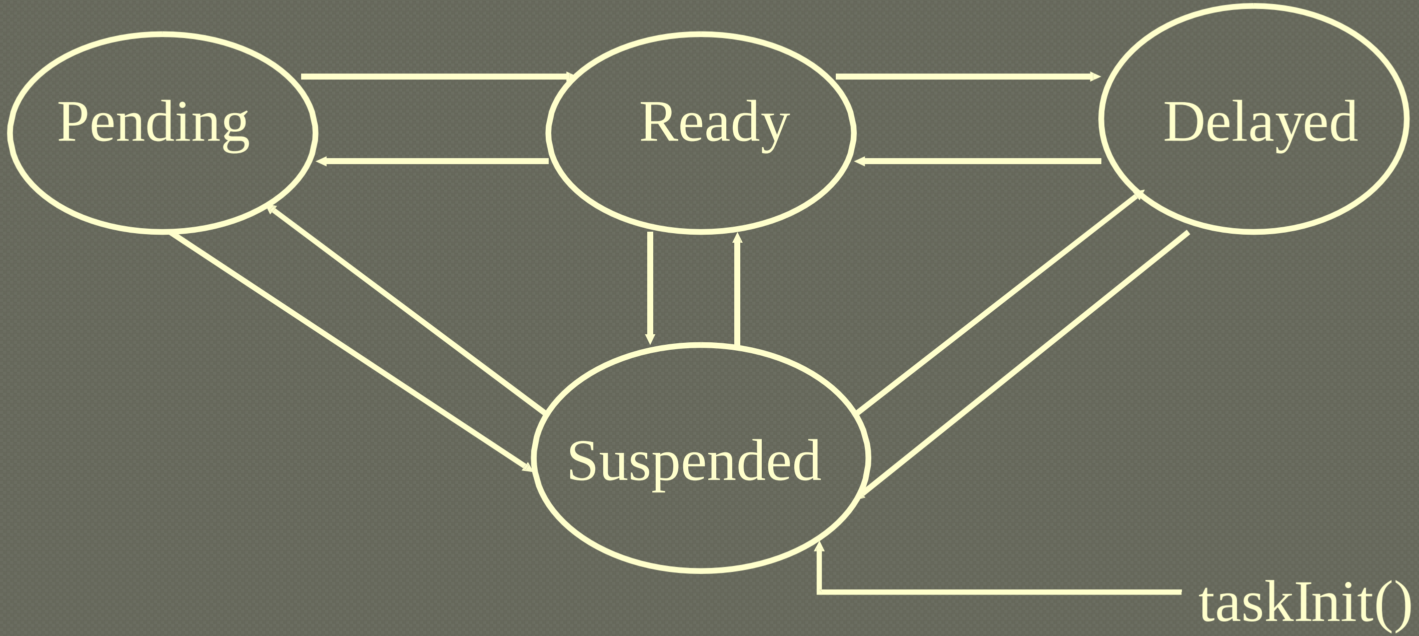
Multitasking Approach

- Create a separate task to manipulate each joint:

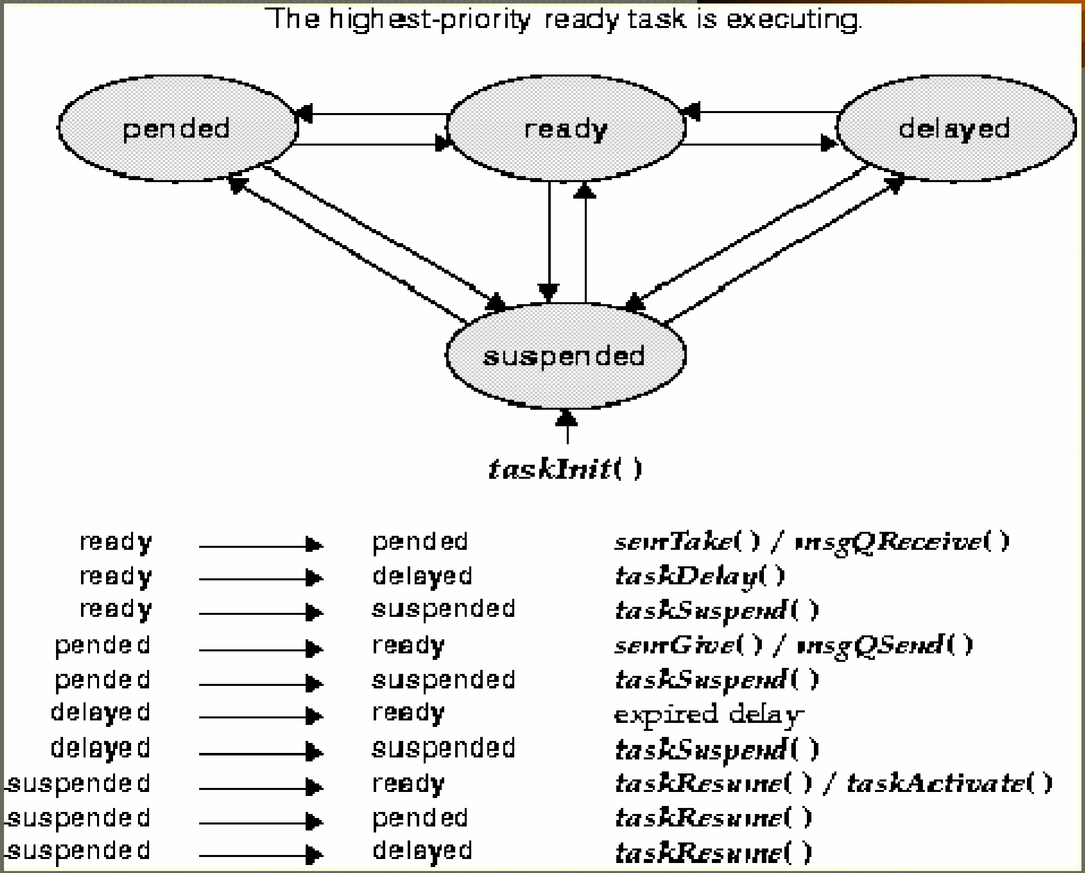
```
joint ( )
{
        for (;;)
        {
                wait; /* Until this joint needs moving */
                moveJoint ( );
        }
}
```

Multitasking and Task Scheduling

- Task State Transition



Multitasking and Task Scheduling


The highest-priority ready task is executing.

ready	pended	*semTake( ) / msgQReceive( )*
ready	delayed	*taskDelay( )*
ready	suspended	*taskSuspend( )*
pended	ready	*semGive( ) / msgQSend( )*
pended	suspended	*taskSuspend( )*
delayed	ready	expired delay
delayed	suspended	*taskSuspend( )*
suspended	ready	*taskResume( ) / taskActivate( )*
suspended	pended	*taskResume( )*
suspended	delayed	*taskResume( )*

Multitasking and Task Scheduling

- Manages tasks.
- Transparently interleaves task execution, creating the appearance of many programs executing simultaneously and independently.

Multitasking and Task Scheduling

- Uses *Task Control Blocks* (TCBs) to keep track of tasks.
  - One per task.
  - WIND_TCB data structure defined in **taskLib.h**
  - O.S. control information
    - e.g. task priority,
    - delay timer,
    - I/O assignments for stdin, stdout, stderr
  - CPU Context information
    - PC, SP, CPU registers,
    - FPU registers FPU registers

Multitasking and Task Scheduling

- Task Context.
  - Program thread (i.e) the task program counter
  - All CPU registers
  - Optionally floating point registers
  - Stack dynamic variables and functionals calls
  - I/O assignments for stdin, stdout and stderr
  - Delay timer
  - Timeslice timer
  - Kernel control structures
  - Signal handlers
  - Memory address space is not saved in the context

Multitasking and Task Scheduling

- To schedule a new task to run, the kernel must: :
  - Save context of old executing task into associated TCB.
  - Restore context of next task to execute from associated TCB.
- Complete context switch must be very fast

Multitasking and Task Scheduling

- Task Scheduling
  - Pre-emptive priority based scheduling
  - CPU is always alloted to the “ready to run” highest priority task
  - Each task has priority numbered between 0 and 255
  - It can be augmented with round robin scheduling.

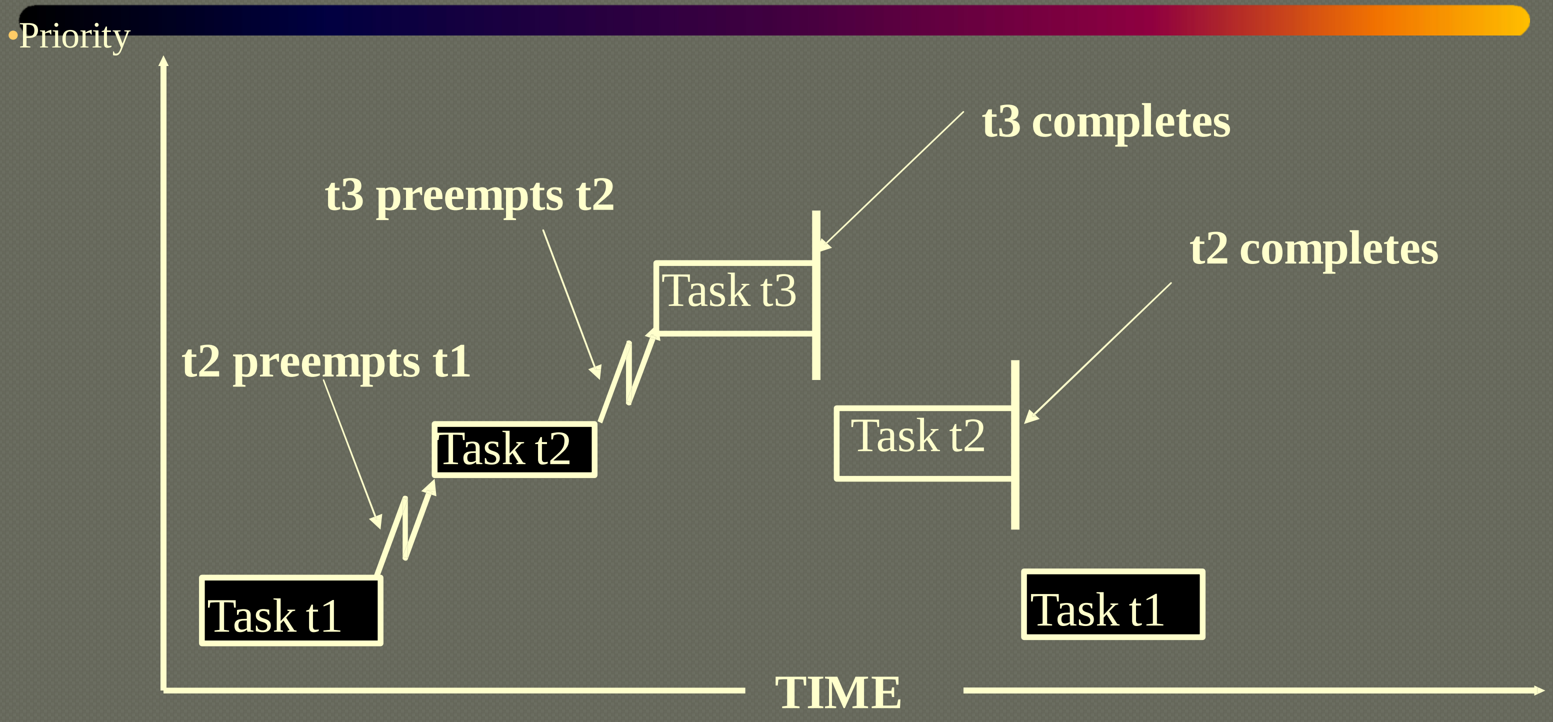
Priority Scheduling

- Work may have an inherent precedence.
- Precedence must be observed when allocating CPU.
- Preemptive scheduler is based on priorities.
- Highest priority task ready to run (not pended or delayed) is allocated to the CPU

Priority Scheduling

- Reschedule can occur anytime, due to:
  - Kernel calls.
  - System clock tick

Priority based pre-emption

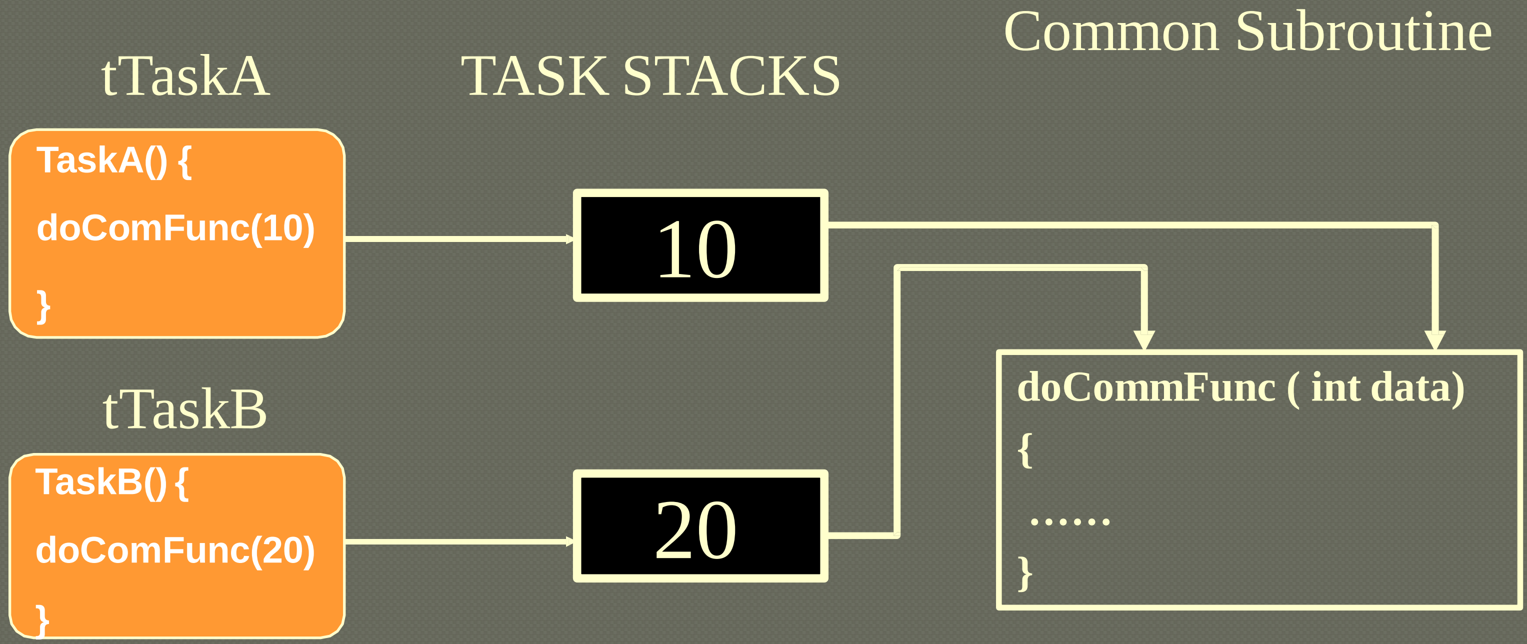


Kernel Time Slicing

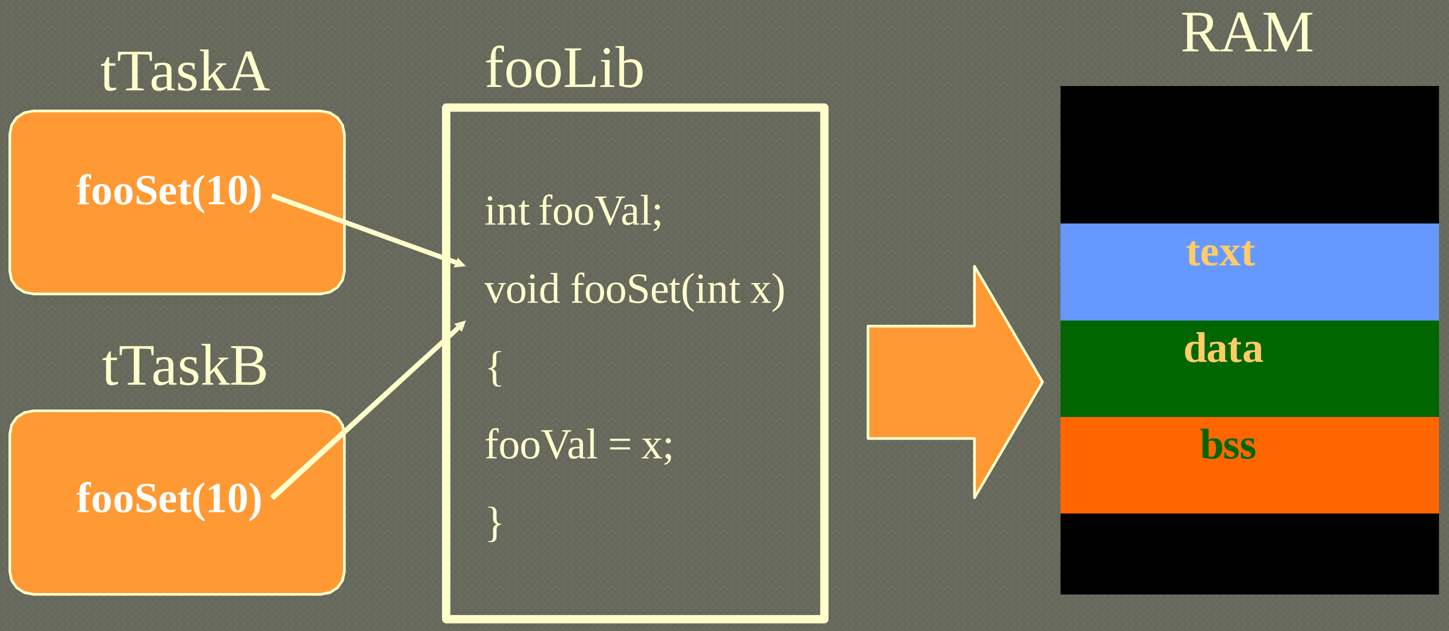
- To allow equal priority tasks to preempt each other, time slicing must be turned on:
  - kernelTimeSlice (ticks)
  - If ticks is 0, time slicing turned off
- Priority scheduling always takes precedence.
  - Round-robin only applies to tasks of the same priority..

Performance Enhancements

- All task reside in a common address space



VxWorks Real Time System



```
int fooVal;
void fooSet(int x)
{
fooVal = x;
}
```

Performance Enhancements

- All tasks run in privileged mode

How real time OS meets the real time requirements.

- Controls many external components.
  - Multitasking allows solution to mirror the problem.
  - Different tasks assigned to independent functions.
  - Inter task communications allows tasks to cooperate.

How real time OS meets the real time requirements.

- High speed execution
  - Tasks are cheap (light-weight).
  - Fast context switch reduces system overhead.
- Deterministic operations
  - Preemptive priority scheduling assures response for high priority tasks.

UNIT V
Task Communication

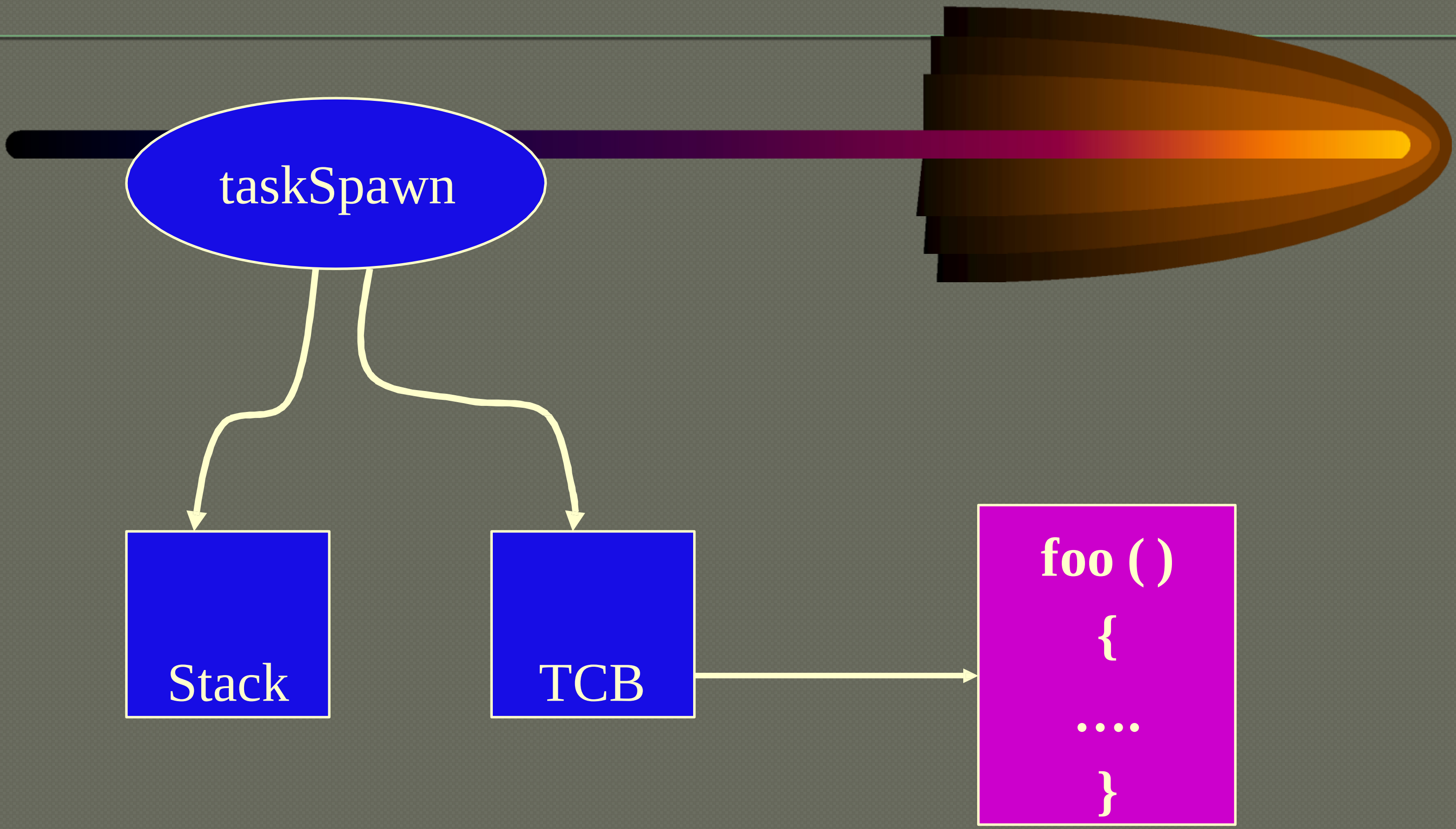
Overview of Multitasking

- Low level routines to create and manipulate tasks are found in **taskLib**..
- A VxWorks task consists of:
  - A stack (for local storage such as automatic variables and parameters passed to routines).
  - A TCB (for OS control).

Overview of Multitasking

- Code is not specific to a task.
  - Code is downloaded before tasks are spawned.
  - Several tasks can execute the same code (e.g., ***printf( )***)

Creating a Task



Creating a Task

int taskSpawn (name , priority, options, stackSize, entryPt, arg1, … arg10)

– name

– priority

– options

– stackSize

– entryPt
)

- Task name
- Task priority (0-255)
- Task Options eg VX_UNBREAKABLE size of stack to be allocated Address of code to execute (initial PC

– arg1 … arg10 Arguments to entry point routine.

Task IDs

- Assigned by kernel when task is created.
- Unique to each task.
- Efficient 32 bit handle for task.
- May be reused after task exits.
- If tid is zero, reference is to task making call (self).

Task IDs

- Relevant **taskLib** routines:

– ***taskIdSelf( )***	Get ID of calling task
– ***taskIdListGet( )***	Fill array with Ids of all existing tasks.
– ***taskIdVerifty( )***	Verify a task ID is valid

Task Names

- Provided for human convenience.
  - Typically used only from the shell (during development).
  - Use task Ids programmatically.
- Should start with a **t**.
  - Then shell can interpret it as a task name.
  - Default is an ascending integer following a **t**.

Task Names

- Doesn’t have to be unique (but usually is).
- Relevant **taskLib** routines: routines:
  - ***taskName( )*** Get name from tid.
  - ***taskNameToId( )*** Get tid from task name.

Task Priorities

- Range from 0 (highest) to 255 (lowest).
- No hard rules on how to set priorities. There are two (often contradictory) “rules of thumb”:
  - More important = higher priority.
  - Shorter deadline = higher priority.
- Can manipulate priorities dynamically with:
  - **taskPriorityGet (tid, &priority)**
  - **taskPrioritySet (tid, priority)**

Task Delete

taskDelete (tid)

- Deletes the specified task.
- Deallocates the TCB and stack.

Task Delete

- exit (code)

- Analogous to a *taskDelete( )* of self. of Code parameter gets stored in the TCB field *exitCode*.
- TCB may be examined for post mortem debugging by:
  - Unsetting the **VX_DELLOC_STACK** option or,
  - Using a delete hook. Using a delete

Resource reclamation

- Contrary to philosophy of system resources sharable by all tasks.
- User must attend to. Can be expensive.
- TCB and stack are the only resources automatically reclaimed.

Resource reclamation

- Tasks are responsible for cleaning up after themselves.
  - Deallocating memory.
  - Releasing locks to system resources.
  - Closing files which are open.
  - Deleting child/client tasks when parent/server exists.

Task Control

taskRestart (tid)

- Task is terminated and respawned with original arguments and tid.
- Usually used for error recovery.

Task Suspend and Resume

taskSuspend (tid)

- Makes task ineligible to execute.
- Can be added to pended or delayed state.

taskResume (tid)

- Removes suspension.
- Usually used for debugging and development

Intertask Communications

- Shared Memory
- Semaphores: Timeout and queues mechanism can be specified
  - Binary Semaphore
  - Mutual Exclusion Semaphores
  - Counting Semaphores

Shared Memory

```
foo.h
extern char buffer[512];
extern int fooSet();
extern char *fooGet();

foo.c
#include foo.h
char buffer[512];
fooSet
{
}
fooGet()
{
}
```

```
taskA()
{
            …
            fooSet();
            ….
}

taskB()
{

            …
            fooGet()
            …
}
```

Semaphores

Int semTake (SEMID)

if a task calls this function

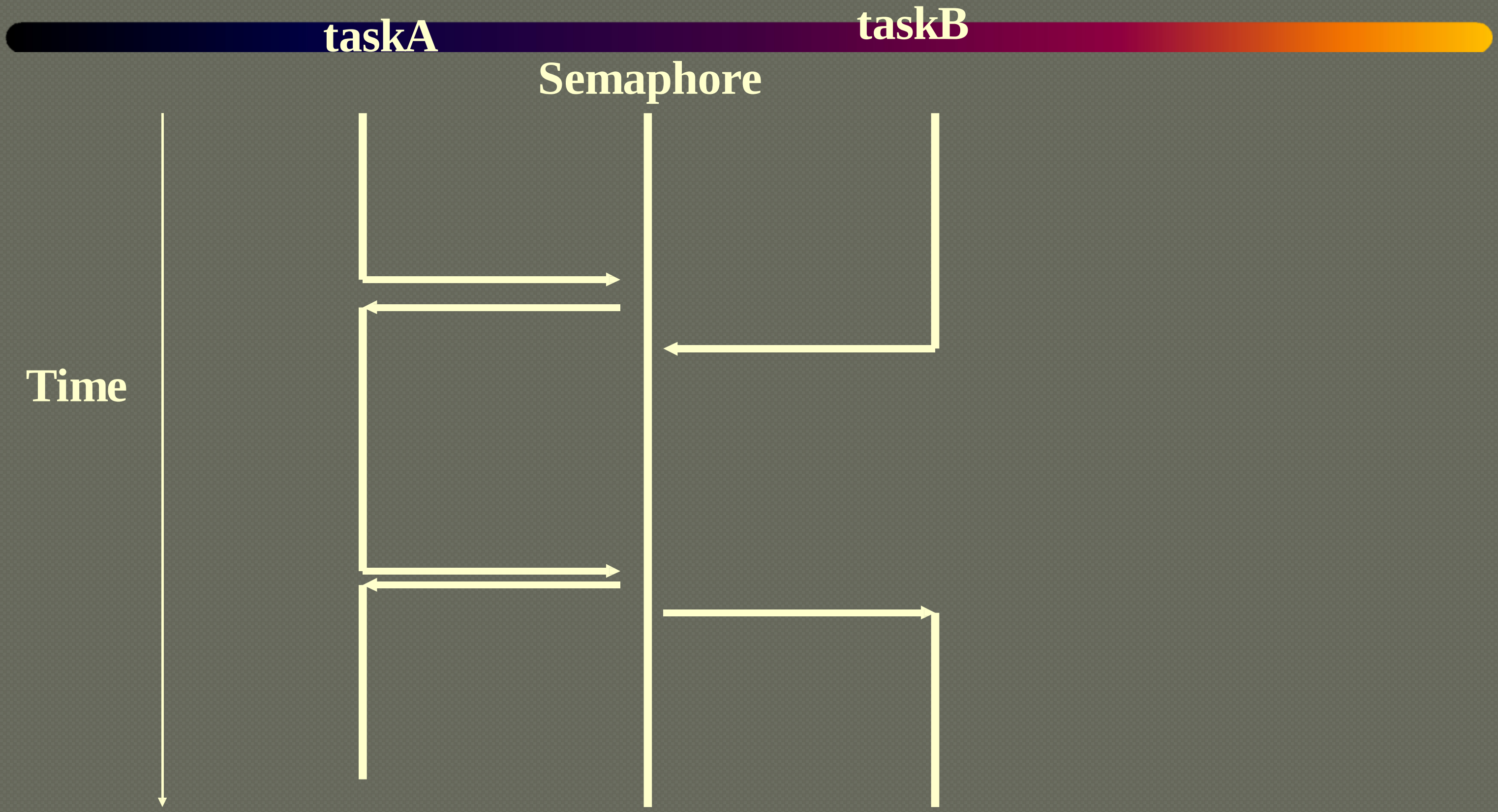
- **this function will return if the semaphore is not taken already**
- **this function will block if the semaphore is taken already**

Int semGive (SEMID)

if a task call this function and

- **there is a task which blocked that task will continue**
- **if there is no task blocked the semaphore is free**

Semaphores



Intertask Communications

- Message Queues
  - Any task including the interrupt handler can send message to message queues.
  - Any task can get message from message queues(excl. interrupt context).
  - Full duplex communications between 2 tasks requires two message queues
  - Timeout can be specified for reading writing and urgency of message is selectable

Intertask Communications

- Message Queues
  - MSG_Q_ID msgQCreate (maxMsgs, maxMsgLength, Options)
  - maxMsgs
    - max number of messages in the queue.
  - maxMsgLength
    - max size of messages
  - options
    - MSG_Q_FIFO or MSG_Q_PRIORITY

Intertask Communications

- STATUS msgQSend (msgQId, buffer, nBytes, timeout, priority)
- int msqQReceive (msgQId, buffer, nBytes, timeout)
- STATUS msgQDelete (msgQId);

Intertask Communications

- Pipes
  - Named I/O device
  - Any task can read from/write to a PIPE
  - an ISR can write to a PIPE
  - select () can used on a pipe
- N/W Intertask Communication
  - Sockets (BSD 4.3)
  - RPC

Features VxWorks supports

- Interrupt handling Capabilities
- Watchdog Timer
- Memory management

Interrupt Handling

- Interrupt Service routines
  - They can bound to user C code through intConnect.
  - intConnect takes I_VEC, reference to ISR and 1 argument to ISR

Don’t’s of ISR

- All ISR use a common stack verify through checkStack()
- Interrupts have no task control block and they do not run in a regular task context.

Donts of ISR(cont)

- ISR must not invoke functions that might cause blocking of caller like
  - semTake(), malloc(), free(), msgQRecv()
  - No I/O through drivers
- floating point co-processors are also discouraged as the floating point registers are not saved or restored.

Exceptions at Interrupt level

- Stores the discriptions of the exception in a special location in memory.
- System is restarted
- In boot ROM the presence of the exception description is tested, if present it prints it out.
- For re displaying ‘e’ command in the boot ROM can be used.

Errors and Exceptions

- ‘errno’ is a global int defined as a macro in “errno.h”
- This return the last error status
- Default expection handler of the OS merely suspends the task that caused it and displays the saved state of the task in stdout

Errors and Exceptions (cont)

- ti and tt can be used to probe into the status
- Unix compatible signals() facility is used to tackle exception other than that of the OS

Watchdog Timers

- Mechanism that allows arbitary C functions to be executed after specified delay
- function is executed as an ISR at the inrrupt level of the system clock
- All restrictions of ISR applies

Watchdog Timers

- Creation of a WD timer is through wdCreate()
- Deletion of a WD timer through wdDelete()
- Start a WD timer through wdStart()
- Cancel a WD timer through wdCancel()

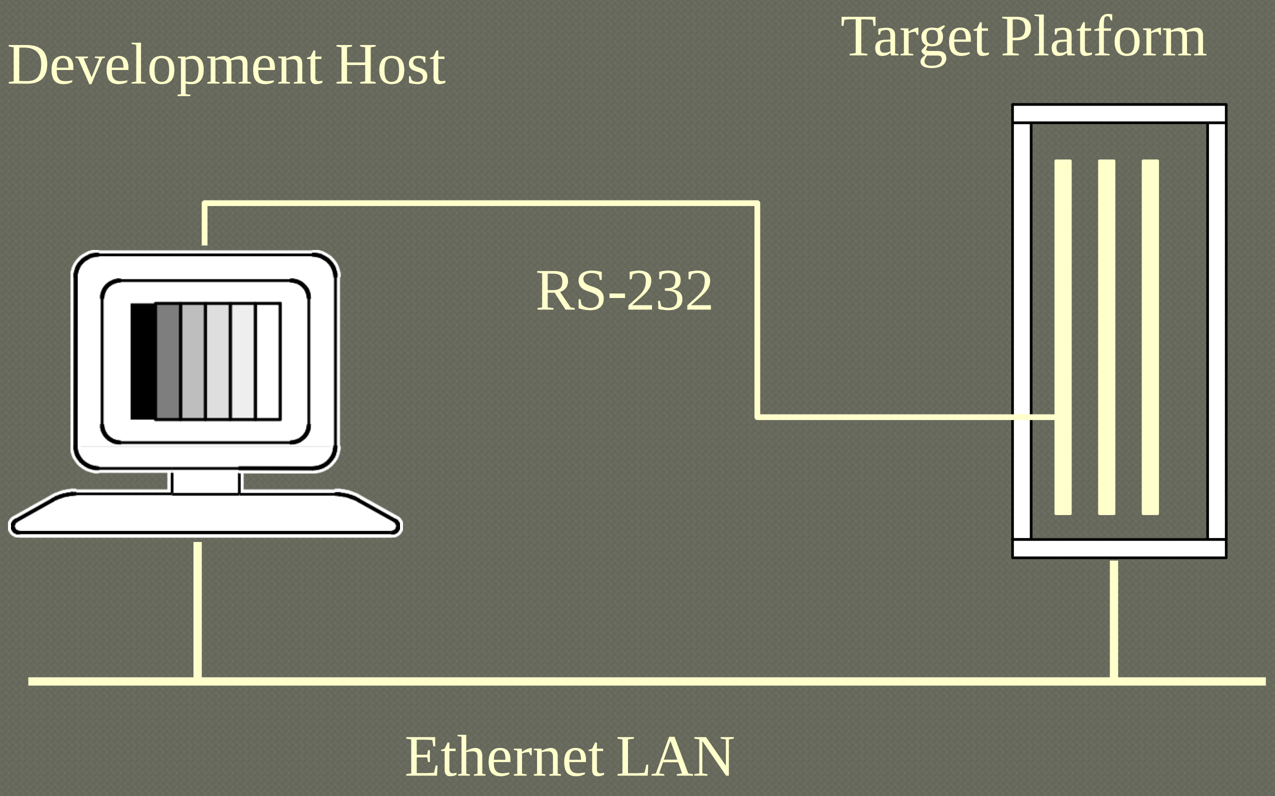
Network Capablities in vxWorks

- Normally uses Internet protocol over standard ethernet connections
- Transperency in access to other vxWorks/Unix systems thro’ Unix compatible sockets
- Remote command execution
- Remote Login

Network Capabilities in vxWorks

- Remote Procedure calls
- Remote debugging
- Remote File access
- Proxy ARP

Software development Environment



Libraries

- vxWorks routines are grouped into libraries
- Each library has corresponding include files

- Library
  - taskLib
  - memPartLib
  - semLib
  - lstLib
  - sockLib

- Routine
  - taskSpawn
  - malloc
  - semTake
  - lstGet
  - send

- Include files
  - taskLib.h
  - stdlib.h
  - semLib.h
  - lstLib.h
  - types.h, sockets.h, sockLib.h

Host Machine

- Any workstation (could run Unix)
- OS should support networking
- Cross/Remote Development Software
- Unix platform
  - Edit, Compile, Link programmes
  - Debug using vxWorks Shell or gdb

Target Machine

- Any H/W which is supported by vxWorks.
- Could be Custom Hardware also.
- Individual object code (.o files) downloaded dynamically.
- Finished application could be burnt into ROM or PROM or EPROM.

Loader and System Symbol Table

- Global system symbol table
- Dynamic loading and unloading of object modules
- Runtime relocation and linking.

Shared Code and reentrancy

- A single copy of code executed by multiple tasks is called shared code
- Dynamic linking provides a direct advantage
- Dynamic stack variables provide inherent reentrancy. Each task has ints own task. Eg linked list in lstLib

Shared Code and reentrancy

- Global and static variables that are inherently non-reentrant, mutual exclusion is provided through use of semaphores eg. semLib and memLib
- Task variables are context specific to the calling task. 4 byte task vars are added to the task's context as required by the task.

The Shell

- Command Line interpreter allows execution of C language expressions and vxWorks functions and already loaded functions
- Symbolic evaluations of variables

The Shell

- -> x=(6+8)/4
  - x=0x20ff378: value=12=0xc
- -> nelson = “Nelson”
- new symbol “name” added to symbol table
- -> x
  - x=0x20ff378: value=12=0xc

The Shell

- Commands

```
-> lkup ( "stuff")
stuff              0x023ebfffe    bss
value = 0 = 0x0
-> lkup("Help")
_netHelp           0x02021a90     text
_objHelp           0x02042fa0     text
value = 0 = 0x0
```

The Shell

- Commands

* sp	creates a task with default options
* td	deletes a task
* ts/tr	Suspend/resume a task
* b	set/display break points
* s	single step a task
* c	continue a task
* tt	Trace a tasks stack
* i/ti	give (detailed) task information
* ld	load a module
* unld	unload a module

The Shell

- Commands

  * period
  * repeat
  * cd ("/u/team3"); the quotes are required
  * ll shows directory contents
  * ls() same as ll

The Shell

- Commands

Shell redirection
-> < script
shell will use the input as from the file
-> testfunc() > testOutput
shell will execute the function and the output will be stored in a file "testOutput"

Debugging in vxWorks

- vxWorks provides Source level debugging
- Symbolic disassembler
- Symbolic C subroutine traceback
- Task specific break points
- Single Stepping
- System Status displays

Debugging in vxWorks

- Exception handlers in hardware
- User routine invocations
- Create and examine variable symbolically

The Shell based Debugging

- Shell based debugging commands

  *b funcName() will set a break point in
  the beginning of the function
  funcName()
  *b 0xb08909f   will set a break point at
  the address 0xb08909f
  *bd funcName() will delete the breakpoint
  at the beginning of the function
  funcName()
  * l will disassemble the code

System Tasks

- tUsrRoot
  - 1st task to be executed by the kernel
    - File: usrConfig.c
    - Spawns tShell, tLogTask, tExecTask, tNetTask and tRlogind
- tShell
  - The Application development support task

System Tasks

- tLogTask
  - Log message hander task
- tNetTask
  - Network support task
- tTelnetd
  - Telenet Support task

System Tasks

- tRlogind
  - Rlogin support for vxWorks. Supports remote user tty interface through terminal driver
- tPortmapd
  - RPC Server support
- rRdbTask
  - RPC server support for remote source level debugging.

Why use RTOS?

- Unix
  - except QNX, most unices don’t live up to the expectation of Real Time situations
  - Present day unix kernel scheduler do support Realtime requirements
  - So Unix kernel can prioritize realtime processes
  - a flag to indicate a RT process is often provided to this effect

vxWorks Vs Unix

- vxWorks does not provide resource reclamation
  - Deviation: user must write their own routine when need.
- vxWorks has a smaller context switch and restore
  - Hence time taken to change context is much smaller.

vxWorks Vs Unix(contd)

- vxWorks requires special care to be taken when writing multitasking code
  – Semaphore are used to achieve reentrancy
- vxWorks has a minimal interrupt latency

vxWorks Vs Unix(contd)

- vxWorks execute threads in a flat memory architechure as part of OS
- vxWorks has no processes. The so-called tasks are actually threads
- vxWorks scheduling could be on round-robin time slicing or pre-emptive scheduling with priorities.

vxWorks Vs Unix

- vxWorks networking is completely Unix compatible. Portable to BSD4.2/4.3 Unix supporting TCP/IP
- vxWorks support BSD sockets
- vxWorks does not distinguish between kernal mode and user mode execution
  – Hence minimal mode change overhead on a give hardware
- vxWorks has Virtual memory model

vxWorks Vs Unix

- vxWorks is not “Realtime Unix” OS or even a variant of Unix
- vxWorks and Unix enjoy a symbiotic relationship
- vxWorks can use Unix as application development platform