ROBOTICS AND AUTOMATION
(R20A0453)

Lecture Notes

B.TECH
(III YEAR – II SEM)
(2022-2023)

Prepared by:

T. Manasa Veena

(Asst Professor, Dept of ECE)

Dept. Of CSE (Cyber security)

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

(Autonomous Institution – UGC, Govt. of India)

Recognized under 2(f) and 12 (B) of UGC ACT 1956
(Affiliated to JNTUH, Hyderabad, Approved by AICTE-Accredited by NBA &NAAC–'A'Grade-ISO9001:2015 Certified) Maisammaguda,Dhulapally(PostVia.Kompally),Secunderabad–500100,Telangana State, India

OPENELECTIVE-III
(R20A0453)ROBOTICSANDAUTOMATION

COURSEOBJECTIVES:

1) To studyoverviewofEmbeddedSystems,Robots,Microprocessors&Microcontrollers.
2) TostudyindetailaboutRoboticsandsensors.
3) TostudyaboutAVRRISCMicrocontrollerarchitectureindetail.
4) TostudyaboutARMProcessorindetail.
5) TostudyaboutArtificialIntelligenceinRobotics.

UNIT-I

IntroductiontoEmbeddedSystemDesign,CategoriesofES,OverviewofEmbeddedSystemArchitecture, Recent Trends in Embedded Systems, Hardware Architecture of Embedded System,Real-timeEmbeddedSystems,RobotsandRobotics,MicroprocessorsandMicrocontrollers,Microcontroller orEmbeddedController

UNIT- II

Robotics: Classification of Robots, Links and Joint, Degree of freedom, Motors-DCmotors,StepperMotors,ServoMotors;PowerTransmission-TypeofGears, Robotic Sensors, Applications of Robot, S/w used for Robot programming.

UNIT- III

The AVR RISC microcontroller architecture: Introduction, AVR family architecture, register file, Pin diagram of AVR, memory organization, I/O ports, timers,USART,Interruptstructure.

UNIT-IV

ARMProcessor: Fundamentals,Registers,currentprogramstatusregister,pipelineconcept,Interrupt andthevectortable.

UNITV

AI IN ROBOTICS: Robotic perception, localization, mapping- configuring space, planning uncertainmovements,dynamicsandcontrolofmovement,Ethicsandrisksofartificialintelligenceinrobotics.

TEXTBOOKS:

1) SubrataGhoshal,"EmbeddedSystems&Robots",CengageLearning
2) StuartRussell, PeterNorvig,“ArtificialIntelligence: Amodernapproch”, PearsonEducation,India2003.

3) ARMSystemDeveloper’sGuide:DesigningandOptimizingSystemSoftware-AndrewN.Sloss,DominicSymes,ChrisWright,ElsevierInc.,2007

UNIT-1

Introduction to Embedded Systems Design:

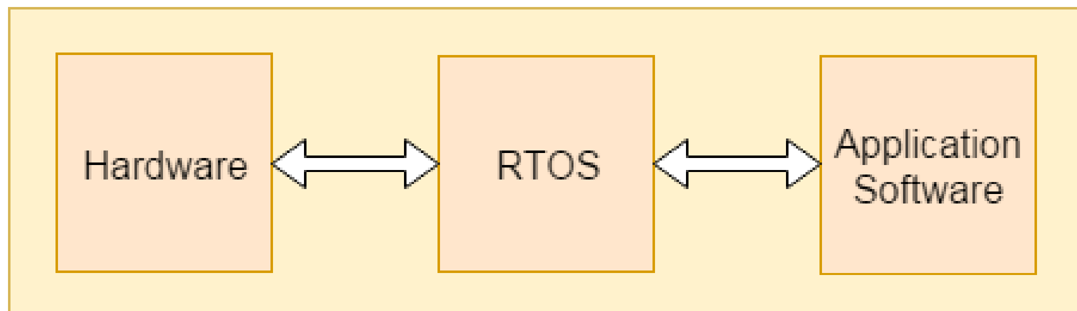
- **Embedded system** is an Electronic/Electro mechanical system which is designed **to perform a specific function** and is a **combination of both hardware and firmware** (Software).
- **E.g.** Electronic Toys, Mobile Handsets, Washing Machines, Air Conditioners, Automotive Control Units, Set Top Box, DVD Player etc…
- An embedded system can be a small independent system or a large combinational system. It is **a microcontroller-based control system** used to perform a specific task of operation.
- An embedded system is a combination of three major components:
- **Hardware:** It comprises of microcontroller based integrated circuit, power supply, LCD display etc.
- **Application software:** Application software allows the user to perform varieties of applications to be run on an embedded system by changing the code installed in an embedded system.
- **Real Time Operating system (RTOS):** RTOS supervises the way an embedded system work. It **acts as an interface between hardware and application software** which supervises the application software and provide mechanism to let the processor run on the basis of scheduling.

An Embedded system is configured to perform a specific dedicated application.

Characteristics of an Embedded System: The important characteristics of an embedded system are:

- Speed (bytes/sec): Should be high speed
- Power (watts): Low power dissipation
- Size and weight: As far as possible small in size and low weight
- Accuracy (% error): Must be very accurate
- Adaptability: High adaptability and accessibility.
- Distributed: embedded systems may be part of larger systems.

- Reliability(probability that the system works properly for a specific period of time): Must be reliable over a long period of time



- So, an embedded system must perform the operations at **a high speed so that it can be readily used for real time applications** and its power consumption must be very low and the size of the system should be as for as possible small and the readings must be accurate with minimum error. The system must be easily adaptable for different situations.

The term distributed means that embedded systems may be a part of larger systems. Many numbers of such distributed embedded systems form a single large embedded control unit. An automatic vending machine is a typical example for this. The vending machine contains a card reader (for pre-paid vending systems), a vending unit, etc. Each of them are independent embedded units but they work together

to perform the overall vending function. Another example is the Automatic Teller Machine (ATM). An ATM contains a card reader embedded unit, responsible for reading and validating the user's ATM card, transaction unit for performing transactions, a currency counter for dispatching/vending currency to the authorised person and a printer unit for printing the transaction details. We can visualise these as independent embedded systems. But they work together to achieve a common goal.

Embedded Systems vs General Computing systems:

General Purpose Computing System	Embedded System
A system which is a combination of a generic hardware and a 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
Contains a General Purpose Operating System (GPOS)	May or may not contain an operating system for functioning

Applications are alterable (programmable) by the user (It is possible for the end user to re-install the operating system, and also add or remove user applications)	The firmware of the embedded system is pre-programmed and it is non-alterable by the end-user (There may be exceptions for systems supporting OS kernel image flashing through special hardware settings)
Performance is the key deciding factor in 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 the hardware and the 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 behaviour	Execution behaviour is deterministic for certain types of embedded systems like 'Hard Real Time' systems

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 aloneEmbedded 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, digital cameras, and microwave ovens, washing machines are the examples for this category

2) Real time embedded system: 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.

a) **Hard real time E.S:** A Real time system in which, the **violation of time constraints will cause critical failure and loss of life or property damage** or catastrophe is known as a Hard Real time system. The hardware and software of hard real-time systems must allow a worst case execution (WCET)

analysis that guarantees the execution be completed within a strict deadline. The **chip selection and RTOS** selection become important factors for hard real-time system design.

Ex: Deadline in a missile control embedded system , Delayed alarm during a Gas leakage , car airbag control system, A delayed response in pacemakers ,Failure in RADAR functioning etc.

b) **Soft Real time E.S:** A Real time system in which, **the violation of time constraints will cause only the degraded quality, but the system can continue to operate** is known as a Soft real time system. In soft real-time systems, the design focus is to offer a guaranteed bandwidth to each real-time task and to distribute the resources to the tasks.

Ex: A Microwave Oven, washing machine, TV remote etc.

3) Networked embedded systems: The networked embedded systems are related to a network. 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.

3) 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 Requirements:

- **Small scale embedded systems:** Embedded systems which are simple in application needs and where the performance requirements are not time critical fall under this category. An 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. A small scale embedded **system may or may not contain an operating system**.

Eg: mp3 player, digital camera

- **Medium scale embedded systems**: Embedded systems which are slightly complex in hardware and firmware requirements. An embedded system supported **by 16–32 bit Microcontroller /Microprocessor with external RAM and ROM or digital signal processors** that can perform more complex operations is a Medium scale embedded system.

Eg: Routers used in networking

- **Large scale embedded systems:** An embedded system supported by 32-64 bit processors/controllers or Reconfigurable system on chip (RSoC) or multicore processors which can perform distributed jobs is considered as a Large scale embedded system. Complex Embedded systems usually contain a high performance RTOS for task scheduling , prioritization and management.

Eg: Airline traffic control system

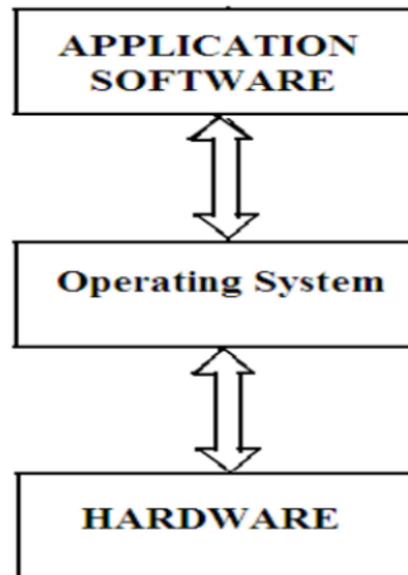
MAJOR APPLICATION AREAS:

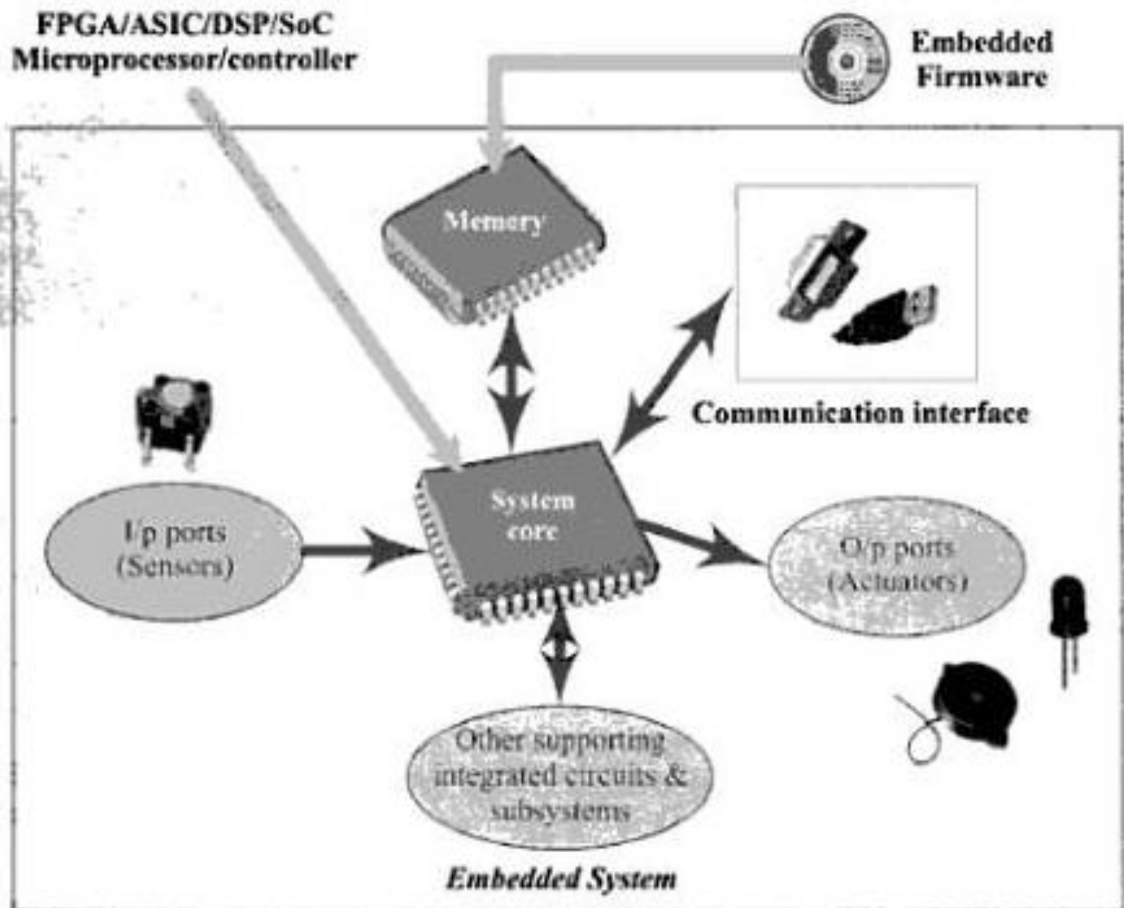
1. *Consumer electronics:* Camcorders, cameras, etc.
2. *Household appliances:* Television, DVD players, washing machine, fridge, microwave oven, etc.
3. *Home automation and security systems:* Air conditioners, sprinklers, intruder detection alarms, closed circuit television cameras, fire alarms, etc.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

Overview of embedded systems architecture:

- Every embedded system consists of customized 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.
- CPU (Central Processing Unit)
- RAM and ROM
- I/O Devices
- Communication Interfaces

- Sensors etc. (Application specific circuitry)





Embedded hardware/software systems are basically designed to regulate a physical variable or to manipulate the state of some devices by sending some control signals to the Actuators or devices connected to the O/p ports of the system, in response to the input signals provided by the end users or Sensors which are connected to the input ports. Hence an embedded system can be viewed as a reactive system. The control is achieved by processing the information coming from the sensors and user interfaces, and controlling some actuators that regulate the physical variable.

Key boards, push button switches, etc. are examples for common user interface input devices whereas LEDs, liquid crystal displays, piezoelectric buzzers, etc. are examples for common user interface output devices for a typical embedded system. It should be noted that it is not necessary that all embedded systems should incorporate these I/O user interfaces. It solely depends on the type of the application for which the embedded system is designed. For example, if the embedded system is designed for any handheld application, such as a mobile handset application, then the system should contain user interfaces like a keyboard for performing input operations and display unit for providing users the status of various activities in progress.

Some embedded systems do not require any manual intervention for their operation. They automatically sense the variations in the input parameters in accordance with the changes in the real world, to which they are interacting through the sensors which are connected to the input port of the system. The

sensor information is passed to the processor after signal conditioning and digitisation. Upon receiving the sensor data the processor or brain of the embedded system performs some pre-defined operations with the help of the firmware embedded in the system and sends some actuating signals to the actuator connected to the output port of the embedded system, which in turn acts on the controlling variable to bring the controlled variable to the desired level to make the embedded system work in the desired manner.

The Memory of the system is responsible for holding the control algorithm and other important configuration details. For most of embedded systems, the memory for storing the algorithm or configuration data is of fixed type, which is a kind of Read Only Memory (ROM) and it is not available for the end user for modifications, which means the memory is protected from unwanted user interaction by implementing some kind of memory protection mechanism. The most common types of memories used in embedded systems for control algorithm storage are OTP, PROM, UVEPROM, EEPROM and FLASH. Depending on the control application, the memory size may vary from a few bytes to megabytes. We will discuss them in detail in the coming sections. Sometimes the system requires temporary memory for performing arithmetic operations or control algorithm execution and this type of memory is known as "working memory". Random Access Memory (RAM) is used in most of the systems as the working memory. Various types of RAM like SRAM, DRAM and NVRAM are used for this purpose. The size of the RAM also varies from a few bytes to KB or MB depending on the application.

An embedded system without a control algorithm implemented memory is just like a new born baby. It is having all the peripherals but is not capable of making any decision depending on the situational as well as real world changes. The only difference is that the memory of a new born baby is self-adaptive, meaning that the baby will try to learn from the surroundings and from the mistakes committed. For embedded systems it is the responsibility of the designer to impart intelligence to the system.

In a controller-based embedded system, the controller may contain internal memory for storing the control algorithm and it may be an EEPROM or FLASH memory varying from a few kilobytes to megabytes. Such controllers are called controllers with on-chip ROM, e.g. Atmel AT89C51. Some controllers may not contain on-chip memory and they require an external (off-chip) memory for holding the control algorithm, e.g. Intel 8031AH.

Recent Trends in Embedded Systems:

- With the fast developments in semiconductor industry and VLSI technology ,one can find tremendous changes in the embedded system design in terms of processor speed , power , communication interfaces including network capabilities and software developments like operating systems and programming languages etc.
- Processor speed and Power :With the advancements in processor technology ,the embedded systems are now a days designed with 32,64 bit processors which can work in real time environment. These processors are able to perform high speed signal processing activities

which resulted in the development of high definition communication devices like 3G mobiles etc.

- Also the recent developments in VLSI technology has paved the way for low power battery operated devices which are very handy and have high longetivity. Also , the present day embedded systems are provided with higher memory capabilities ,so that most of them are based on tiny operating systems like android etc
- **Communication interfaces** : Most of the present day embedded systems are aimed at internet based applications. So,the communication interfaces like Ethernet, USB, wireless LAN etc.have become very common resources in almost all the embedded systems. The developments in memory technologies also helped in porting the TCP/IP protocol stack and the HTTP server software on to the embedded systems. Such embedded systems can provide a link between any two devices anywhere in the globe.
- **Operating systems :**With recent software developments ,there is a considerable growth in the availability of operating systems for embedded systems. Mainly new operating systems are developed which can be used in real time applications. There are both commercial RTOSes like Vx Works , QNX,WIN-CE and open source RTOSes like RTLINUX etc. The Android OS in mobiles has revolutionized the embedded industry.
- **Programming Languages :**There is also a remarkable development in the programming languages. Languages like C++, Java etc. are now widely used in embedded application programming. For example by having the Java virtual machine in a mobile phones ,one can download Java applets from a server and can be executed on your mobile. In addition to these developments, now a days we also find new devices like ASICs and FPGAs in the embedded system market. These new hardware devices are popular as programmable devices and reconfigurable devices.

Hardware Architecture of Embedded System

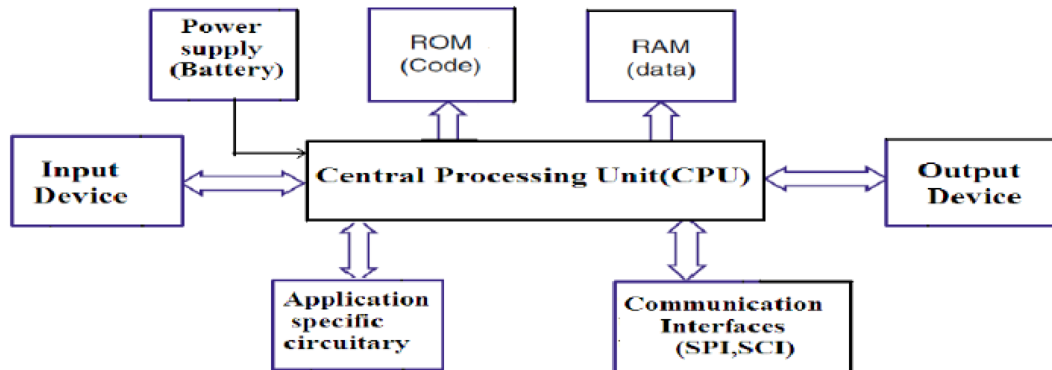


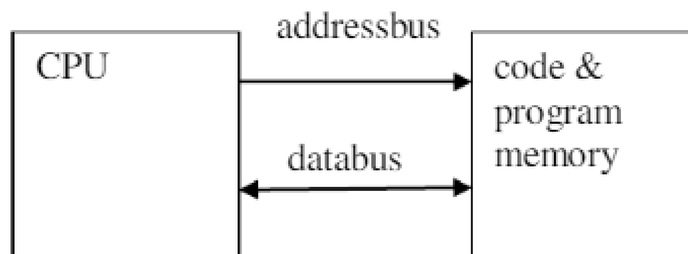
Fig: hardware architecture of embedded system

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.
- 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 .
- The CORE in the embedded system may be a general purpose processor like a microcontroller or a special purpose processor like a DSP (Digital signal processor). But any CORE consists of an Arithmetic Logic Unit (ALU), a Control Unit (CU), and many internal registers that are connected by buses.
- In an embedded system, the CPU may never stop and run forever .The CPU works in a cycle of fetching an instruction, decoding it, and executing it, known as the fetch-decode-execute cycle. The cycle begins

when an instruction is fetched from a memory location pointed to by the PC to the IR via the data 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:**
- The Von Neumann architecture was first proposed by a computer scientist John von Neumann. In this architecture, one data path or bus exists for both instruction and data. As a result, the CPU does one operation at a time. It either fetches an instruction from memory, or performs read/write operation on data. So an instruction fetch and a data operation cannot occur simultaneously, sharing a common bus.
- Von-Neumann architecture supports simple hardware. It allows the use of a single, sequential memory. Today's processing speeds vastly outpace memory access times, and we employ a very fast but small amount of memory (cache) local to the processor.



Harvard Architecture:

- Computers have separate memory areas for program instructions and data using internal data buses, allowing simultaneous access to both instructions and data.

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:

- Embedded system memory can be either on-chip or off-chip.
- 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.
- The ROM, EPROM, and Flash memory are all read-only type memories often used to store code in an embedded system.
- The embedded system code does not change after the code is loaded into memory. The ROM is programmed at the factory and cannot be changed over time.
- The newer microcontrollers come with EPROM or Flash instead of ROM.
- Most microcontroller development kits come with EPROM as well.
- EPROM and Flash memory are easier to rewrite than ROM. EPROM is an Erasable Programmable
- The size of EPROM ranges up to 32kb in most embedded systems.
- Flash memory is an EPROM which can be programmed from software so that the developers don‘t need to physically remove the EPROM from the circuit to re-program it.

- It is much quicker and easier to re-write Flash than other types of EPROM.
- 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.
- 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.
- The RAM chip can be SRAM (static) or DRAM (dynamic) depending on the manufacturer. SRAM is faster than DRAM, but is more expensive

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.

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.
- In addition to these peripherals, an embedded system may also have sensors, Display modules like LCD or Touch screen panels, Debug ports c,ertain communication peripherals like I2C, SPI, Ethernet, CAN, USB for high speed data transmission. Now a days various sensors are also becoming an important part in the design of real time embedded systems. Sensors like temperature sensors, light sensors, PIR sensors, gas sensors are widely used in application specific circuitry.

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.

- 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.
- One need to reboot the PC now and then due to various faults caused by hardware or software. An embedded system cannot be rebooted manually, because it has been embedded into its system. That is why many microcontrollers come with an on-chip watchdog timer which can be configured just like the counter in the regular timer. After a system gets stuck (power supply voltage out of range or regular timer does not issue timeout after reaching zero count) the watchdog eventually will restart the system to bring the system back to a normal operational condition.

Real Time Embedded Systems

- A real-time embedded system is a particular version of an embedded system that works on the basis of real-time computing represented by a dedicated type of operating system — RTOS.
- Its working principles are as follows:
- Quick response to external factors: an embedded system must work within fixed time constraints.
- Predictability: an embedded system must be deterministic or predictable, meaning that no deviations are allowed.
- The deadline is above all: meeting the deadline is more important than other performance characteristics.
- Operational failures may lead to catastrophe: if a task does not meet time limits, it negatively affects users and may even lead to fatal results.

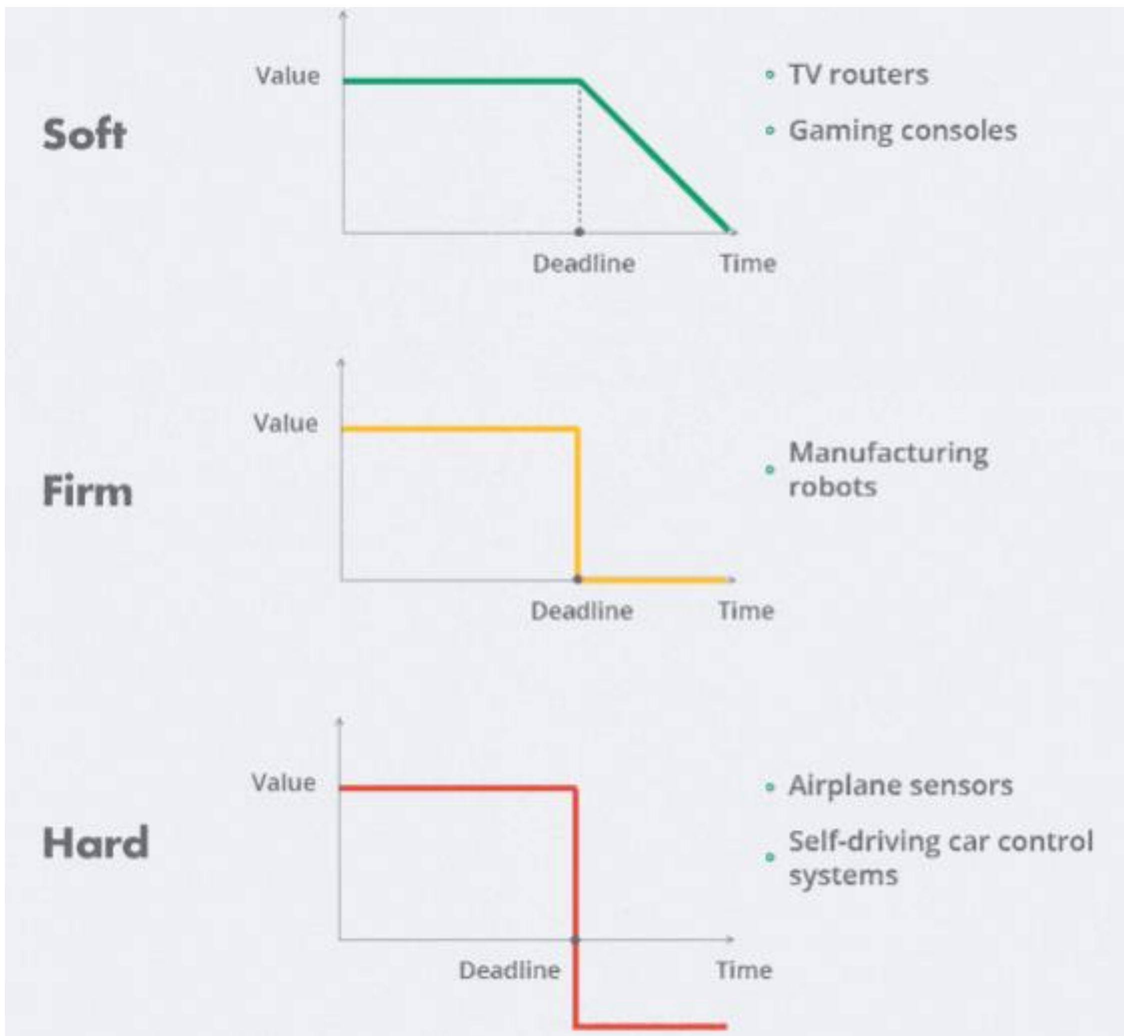
Categories of Real-Time Embedded Systems:

Real-time embedded systems fall into three categories —

- soft,
- firm and
- hard

depending on the acceptability of violation of time constraints:

- **Hard** — timing constraints must not be violated. For such built-in systems, it's crucial that the deadline is met in all cases. No errors are acceptable as they will lead to harmful effects and the device will totally lose its operation value. An air defense system that needs to detect and intercept an attacking missile within milliseconds has this type of embedded system. Its failure jeopardizes human lives. Other examples are airplane sensors or self-driving car control systems.



- **Firm** — exceeding the deadline is occasionally permitted, though it is undesirable. Operating failures of firm real-time systems don't result in harmful effects. But devices lose their performance value because of such failures. Manufacturing robots refer to this category.
- **Soft** — exceeding the deadline is acceptable. Response failures of soft real-time systems diminish user experience, but they don't reduce the performance value at once. If such a system fails to meet a deadline, it will either recover or gradually diminish its operation. For example, TV router boxes and gaming consoles refer to soft real-time embedded systems. Time lags in their operation may happen, but they have either insignificant or no consequences.

Physical Constraints:

- These refer to hardware components, the required characteristics of the device itself and external environmental factors. Embedded engineers usually have to create a product that should satisfy the following conditions:
- Definite device size
- Spatial constraints for device installation
- Limited memory and power consumption
- Certain environmental conditions for device operation (temperature, humidity, pressure)

Timing Constraints:

The very essence of real-time systems is that they must respond to events within predefined time limits. The most severe challenge for developers is to create a system capable of meeting the deadline under any conditions. They must do accurate calculations and build the appropriate task scheduling system to succeed.

Task Scheduling

- For real-time systems, it's crucial to organize data processing strictly following timing constraints. Real-time operating systems comprise scheduling algorithms that are responsible for managing these constraints. Thus, quick responses to events directly depend on which scheduling algorithm you choose for your real-time embedded solution.
- There are *preemptive* and *non- preemptive* algorithms, and software engineers can choose from several popular types.
- **Priority scheduling.** This algorithm prioritizes all tasks and puts forward the task with the highest priority to be performed first by the processor. A preventive version of the algorithm stops a running task if there is another one with a higher priority in the queue. A non-preventive version doesn't stop running tasks, but a higher priority task will be the next in the line. This approach doesn't suit tasks with equal priority.

- **Round-robin scheduling.** This is a preventive scheduling algorithm that doesn't prioritize tasks. Instead, it allocates an equal time interval (e.g., 500 ms) for each task, and the CPU processes them one by one. One task may go several rounds to be completed. This approach is relatively easy and straightforward.
- **First come, first served (FCFS).** This is a non-preventive algorithm that puts tasks into the running state depending on the time they arrive. The process is straightforward: the task that comes first is the first to utilize the computing power. This algorithm ensures a high response time.
- **Shortest job first (SJF).** The non-preventive version of this algorithm allocates tasks depending on their execution time: the task with the shortest execution time is run next. The preventive variant can interrupt running tasks if a task with a shorter remaining execution period arrives.

Robots and Robotics:

- A robot is a type of automated machine that can execute specific tasks with little or no human intervention and with speed and precision. The field of robotics, which deals with robot design, engineering and operation
- Robots can perform some tasks better than humans, but others are best left to people and not machines.
- **Robot:** It is a machine capable of carrying out a complex series of actions automatically, especially one programmable by computer.
- **According to ISO** It is an automatically controlled, reprogrammable, multipurpose manipulator (robot with fixed base) programmable in three or more axes, which can either be fixed in place or mobile for use in industrial automation applications.
- **According to RIA** (Root Institute of America), It is reprogrammable multi-Functional manipulator designed to move materials, parts, tools or specialized devices through variable programmed motions for the performance of a variety of tasks.

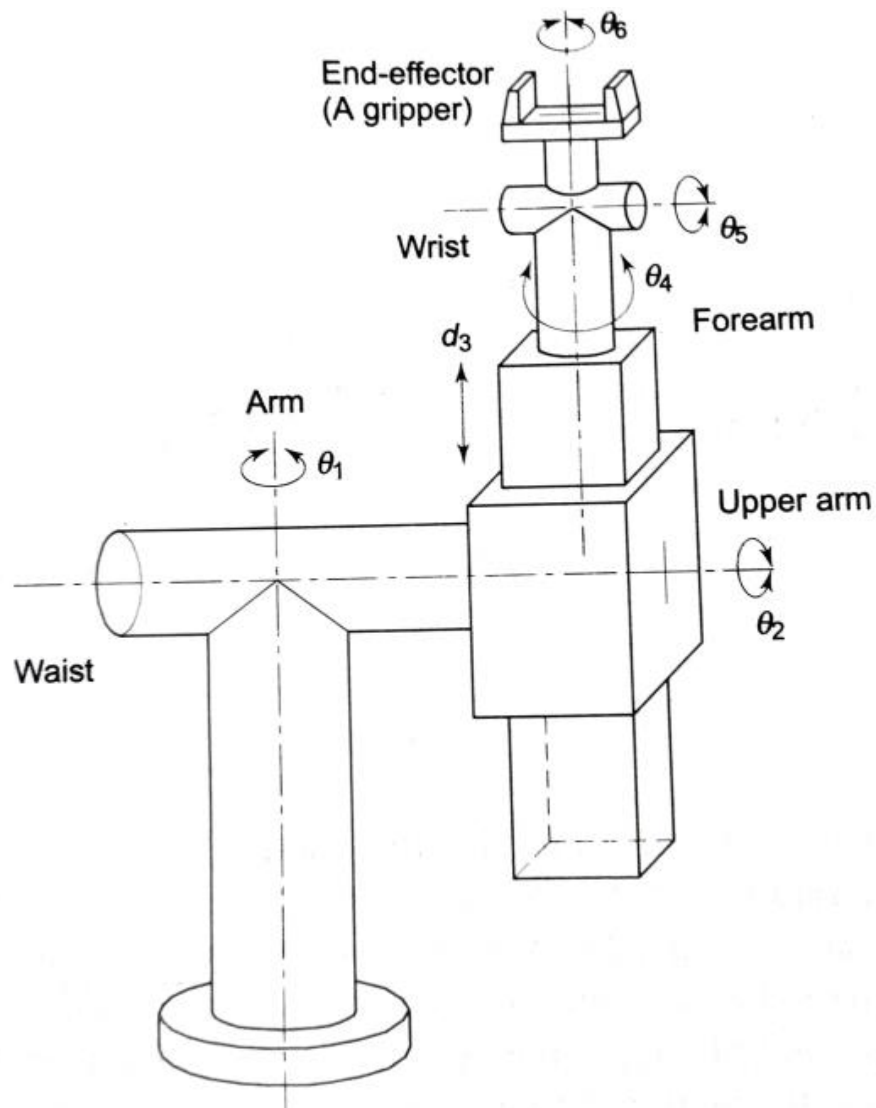
- **Manipulator:** Mechanical Hand. We model, design and develop the human hand in the form of an artificial Hand. It is reprogrammable and multifunctional.
- **CNC:** Computerized Numerical Control Machine-We can perform a variety of tasks by changing the program. Similarly, the same Robot can be used to perform variety of tasks by reprogramming it. But the level of reprogrammability differs in CNC machine and Robot(more).
- **Robotics:** It is the science which deals with the issues related to design, manufacturing and usage of robots.
- 3H's of Human beings are copied into Robotics such as
- Hand-with Manipulator
- Head-with Intelligence
- Heart-with emotions

Machines that can replace human beings as regards to physical work and decision making are categorized as Robots and their study as robotics.

- Czech writer ,Karel Capek in his drama introduced the word robot to the world in1921.It is derived from the Czech word robota meaning "forced labourer".
- Isaac Asimov the well known Russian science fiction writer coined the word Robotics in his story "Runaround" , published in 1942 to denote the science devoted to the study of Robotics.
- The robot technology is advancing rapidly. Robots and robot like manipulators are now commonly employed in hostile environment such as at various places in an atomic plant for handling radio active materials.
- Robots are being employed to construct and repair space stations and satellites.
- There are now increasing number of applications of robots such as in nursing and aiding a patient. Microrobots are being designed to do damage control inside human veins.

- One type of robot commonly used in the industry is a robotic manipulator or simply a robotic arm.
- It is an open or closed kinematic chain of rigid links interconnected by movable joints.
- In some configurations, links can be considered to correspond to human anatomy as waist, upper arm, and forearm with joints at shoulder and elbow.
- At the end of the arm, a wrist joint connects an end effector to the forearm.
- The end effector may be a tool and its a fixture or a gripper or any other device to do work . The end effector is similar to the human hand with or

without fingers.



Motivation behind Robotics: To cope up with increasing demands of a dynamic and competitive market, modern manufacturing methods should satisfy the following requirements:

- Reduced Production Cost

- Increased Productivity
- Improved Product Quality
- **Laws of Robotics:**

  1)A robot should not injure a human being or through inaction allow a human to be harmed.

  2)A robot must obey orders given by humans except when that conflicts with the first law.

  3)A robot must protect its own existence unless that conflicts with the first or second law.

The following are things robots do better than humans:

- Automate manual or repetitive activities in corporate or industrial settings.
- Work in unpredictable or hazardous environments to spot hazards like gas leaks.
- Process and deliver reports for enterprise security.
- Fill out pharmaceutical prescriptions and prep IVs.
- Deliver online orders, room service and even food packets during emergencies.
- Assist during surgeries.
- Robots can also make music, monitor shorelines for dangerous predators, help with search and rescue and even assist with food preparation.

Different Types of robots

- There are as many different types of robots as there are tasks.

1. Androids

Androids are robots that resemble humans. They are often mobile, moving around on wheels or a track drive. According to the American Society of Mechanical Engineers, these humanoid robots are used in areas such as caregiving and personal assistance, search and rescue, space exploration and research, entertainment and education, public relations and healthcare, and manufacturing.

2. Telechir

A telechir is a complex robot that is remotely controlled by a human operator for a telepresence system. It gives that individual the sense of being on location in a remote, dangerous or alien environment, and enables them to interact with it since the telechir continuously provides sensory feedback.

3. Telepresence robot

A telepresence robot simulates the experience -- and some capabilities -- of being physically present at a location. It combines remote monitoring and control via telemetry sent over radio, wires or optical fibers, and enables remote business consultations, healthcare, home monitoring, childcare and more.

4. Industrial robot

- The IFR (International Federation of Robotics) defines an industrial robot as an "automatically controlled, reprogrammable multipurpose manipulator programmable in three or more axes." Users can adapt these robots to different applications as well. Combining these robots with AI has helped businesses move them beyond simple automation to higher-level and more complex tasks.
- In 2019, there were over 390,000 industrial robots installed worldwide, according to the IFR -- with China, Japan and the U.S. leading the way.

In industrial settings, such robots can do the following:

- optimize process performance;
- automate production to increase productivity and efficiency;
- speed up product development;

- enhance safety; and

lower costs.

5.Swarm robot

Swarm robots (aka insect robots) work in fleets ranging from a few to thousands, all under the supervision of a single controller. These robots are analogous to insect colonies, in that they exhibit simple behaviors individually, but demonstrate behaviors that are more sophisticated with an ability to carry out complex tasks in total.

6. Smart robot

This is the most advanced kind of robot. The smart robot has a built-in AI system that learns from its environment and experiences to build knowledge and enhance capabilities to continuously improve. A smart robot can collaborate with humans and help solve problems in areas like the following:

- agricultural labor shortages;
- food waste;
- study of marine ecosystems;
- product organization in warehouses; and
- clearing of debris from disaster zones.

Progressive Advancements in Robots

The growth of robots can be grouped into *robot generations*, based on characteristic breakthroughs in robot's capabilities. These generations are overlapping and include futuristic projections.

First Generation

The first generation robots are repeating, nonservo, pick-and-place, or point-to-point kind. The technology for these is fully developed and at present about 80% robots in use in the industry are of this kind. It is predicted that these will continue to be in use for a long time.

Second Generation

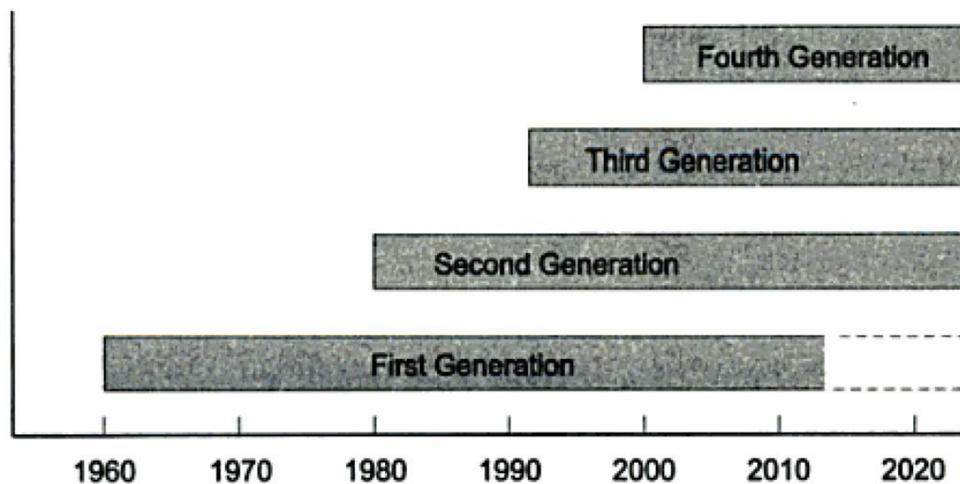
The addition of sensing devices and enabling the robot to alter its movements in response to sensuary feedback marked the beginning of second generation. These robots exhibit path-control capabilities. This technological breakthrough came around 1980s and is yet not mature.

Third Generation

The third generation is marked with robots having human-like intelligence. The growth in computers led to high-speed processing of information and, thus, robots also acquired artificial intelligence, self-learning, and conclusion-drawing capabilities by past experiences. On-line computations and control, artificial vision, and active force/torque interaction with the environment are the signifjcant characteristics of these robots. The technology is still in infancy and has to go a long way.

Fourth Generation

This is futuristic and may be a reality only during this millennium. Prediction about its features is difficult, if not impossible. It may be a true android or an artificial biological robot or a super humanoid capable of producing its own clones. This might provide for fifth and higher generation robots.



Microprocessors and Microcontrollers

Microprocessor

- The microprocessor is useful in very intensive processes. It only contains a CPU (central processing unit) but there are many other parts needed to work with the CPU to complete a process. These all other parts are connected externally.
- Microprocessors are not made for a specific task as well as they are useful where tasks are complex and tricky like the development of software, games, and other applications that require high memory and where input and output are not defined.
- Eg: Complex home security, Home computers, Video game systems

Microcontroller

- The microcontroller is designed for a specific task or to perform the assigned task repeatedly. Once the program is embedded on a microcontroller chip, it can't be altered easily and you may need some special tools to reburn it.
- As per application, the process is fixed in microcontroller. Hence, the output depends on the input given by the user or sensors or predefined inputs.

- e.g. Calculator, Washing Machine, ATM machine, Robotic Arm, Camera, Microwave oven, Oscilloscope, Digital multimeter, ECG Machine, Printer

Differences between Microprocessor and Micro-controller

	Microprocessor	Microcontroller
1.	We need to connect peripherals externally. So it makes circuit bulky.	The presence of peripherals such as RAM, ROM, Input-output, and Timers are In-built. So It is available on a single chip.
2.	It increases the overall cost of the system high.	The overall cost of the system is less.
3.	We can connect external memory in ranges of Mbytes and even Gbytes. But speed is less.	The inbuilt finite memory helps to improve the speed of operations.
4.	You can't use it in a compact system.	You can use it in compact systems.
5.	Due to external components, the total power consumption is high. Therefore, it is not ideal for the devices running on stored power like batteries.	As external components are low, total power consumption is less. So it can be used with devices running on stored power like batteries.
6.	Most of the microprocessors do not have power-saving features.	Most of the microcontrollers offer power-saving mode.
7.	The microprocessor has a smaller number of registers, so more operations are memory-based.	The microcontroller has more register. Hence the programs are easier to write.
8.	These are based on the von Neumann model where program and data are stored in the same memory module.	These are based on Harvard architecture where program memory and data memory are separate.
9.	It is a central processing unit on a single silicon-based integrated chip.	It is a byproduct of the development of microprocessors with a CPU along with other peripherals.
10	It uses an external bus to interface to RAM, ROM, and other peripherals.	It uses an internal controlling bus.
11	Microprocessor-based systems can run at a very high speed because of the technology involved.	Microcontroller based systems run up to 200MHz or more depending on the architecture.
12	It's useful for general purpose applications that allow you to handle loads of data.	It's useful for application-specific systems.
13	It's complex and expensive, with a large number of instructions to process.	It's simple and inexpensive with less number of instructions to process.

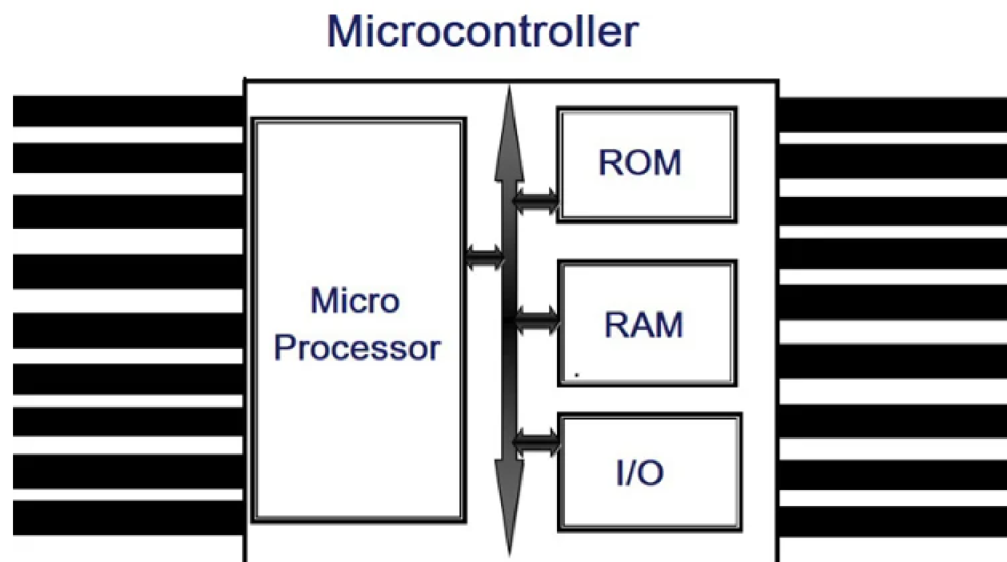
Brief overview

- **Microprocessor** consists of only a Central Processing Unit, whereas Micro Controller contains a CPU, Memory, I/O all integrated into one chip.
- The microprocessor is useful in Personal Computers whereas Micro Controller is useful in an embedded system.

- The microprocessor uses an external bus to interface to RAM, ROM, and other peripherals, on the other hand, Microcontroller uses an internal controlling bus.
- Microprocessors are based on Von Neumann model Microcontrollers are based on Harvard architecture
- The microprocessor is complicated and expensive, with a large number of instructions to process but Microcontroller is inexpensive and straightforward with fewer instructions to process.

Microcontrollers(Embedded Controllers)

- A **microcontroller** is a compact integrated circuit designed to govern a specific operation in an embedded system. A typical microcontroller includes a processor, memory and input/output (I/O) peripherals on a single chip.
- Sometimes referred to as an embedded controller or microcontroller unit (MCU), microcontrollers are found in vehicles, robots, office machines, medical devices, mobile radio transceivers, vending machines and home appliances, among other devices. They are essentially simple miniature personal computers (PCs) designed to control small features of a larger component, without a complex front-end operating system (OS).



How do microcontrollers work?

- A microcontroller is embedded inside of a system to control a singular function in a device. It does this by interpreting data it receives from its I/O peripherals using its central processor. The temporary information that the microcontroller receives is stored in its data memory, where the processor accesses it and uses instructions stored in its program memory to decipher and apply the incoming data. It then uses its I/O peripherals to communicate and enact the appropriate action.
- Devices often utilize multiple microcontrollers that work together within the device to handle their respective tasks.
- For example, a car might have many microcontrollers that control various individual systems within, such as the anti-lock braking system, traction control, fuel injection or suspension control. All the microcontrollers communicate with each other to inform the correct actions. Some might communicate with a more complex central computer within the car, and others might only communicate with other microcontrollers. They send and receive data using their I/O peripherals and process that data to perform their designated tasks.

Elements of a microcontroller:

The core elements of a microcontroller are:

- **The processor (CPU)** -- A processor can be thought of as the brain of the device. It processes and responds to various instructions that direct the microcontroller's function. This involves performing basic arithmetic, logic and I/O operations. It also performs data transfer operations, which communicate commands to other components in the larger embedded system.

Memory -- A microcontroller's memory is used to store the data that the processor receives and uses to respond to instructions that it's been programmed to carry out. A microcontroller has two main memory types:

    - Program memory, which stores long-term information about the instructions that the CPU carries out. Program memory is non-volatile memory, meaning it holds information over time without needing a power source.

- Data memory, which is required for temporary data storage while the instructions are being executed. Data memory is volatile, meaning the data it holds is temporary and is only maintained if the device is connected to a power source.

I/O peripherals -- The input and output devices are the interface for the processor to the outside world. The input ports receive information and send it to the processor in the form of binary data. The processor receives that data and sends the necessary instructions to output devices that execute tasks external to the microcontroller.

There are many supporting components that can be classified as peripherals.

- **Analog to Digital Converter (ADC)** -- An ADC is a circuit that converts analog signals to digital signals. It allows the processor at the center of the microcontroller to interface with external analog devices, such as sensors.
- **Digital to Analog Converter (DAC)** -- A DAC performs the inverse function of an ADC and allows the processor at the center of the microcontroller to communicate its outgoing signals to external analog components.
- **System bus** -- The system bus is the connective wire that links all components of the microcontroller together.
- **Serial port** -- The serial port is one example of an I/O port that allows the microcontroller to connect to external components. It has a similar function to a USB or a parallel port but differs in the way it exchanges bits.

Microcontroller features:

- A microcontroller's processor will vary by application. Options range from the simple 4-bit, 8-bit or 16-bit processors to more complex 32-bit or 64-bit processors.
- Microcontrollers can use volatile memory types such as random access memory (RAM) and non-volatile memory types -- this includes flash

memory, erasable programmable read-only memory (EPROM) and electrically erasable programmable read-only memory (EEPROM).

- Generally, microcontrollers are designed to be readily usable without additional computing components because they are designed with sufficient onboard memory as well as offering pins for general I/O operations, so they can directly interface with sensors and other components.
- Microcontroller architecture can be based on the Harvard architecture or von Neumann architecture, both offering different methods of exchanging data between the processor and memory. With a Harvard architecture, the data bus and instruction are separate, allowing for simultaneous transfers. With a Von Neumann architecture, one bus is used for both data and instructions.
- Most of the microcontrollers are based on RISC architecture.
- MCUs feature input and output pins to implement peripheral functions. Such functions include analog-to-digital converters, liquid crystal display (LCD) ,Sensors gathering data.
- Common MCUs include the Intel MCS-51, often referred to as an 8051 microcontroller, which was first developed in 1985; the AVR microcontroller developed by Atmel in 1996; the programmable interface controller (PIC) from Microchip Technology; and various licensed Advanced RISC Machines (ARM) microcontrollers.
- **Applications:** Microcontrollers are used in multiple industries and applications, including in the home and enterprise, building automation, manufacturing, robotics, automotive, lighting, smart energy, industrial automation, communications and internet of things (IoT) deployments

UNIT-2

Classification of Robots

- Robots can be classified based on the application or by their locomotion / kinematics.
- Classifying Robots by their Application: Based on this classification, there are two broad ways of categorizing robots.

1)Industrial Robots

2)Service Robots

- Industrial Robots: These were one of the first robots to be used commercially. In a factory assembly line, these are usually in the form of articulated arms specifically developed for such applications as welding, material handling, painting and others.
- They can be further subdivided as:

1)Manufacturing Robots

2)Logistics Robots

- **Manufacturing robots** are designed to **move materials**, as well as perform a variety of programmed tasks in manufacturing and production settings. They are often used to perform duties that are dangerous or unsuitable for human workers.
- **Logistics robots** are mobile automated guided vehicles primarily used in warehouses and storage facilities to transport goods.
- **Service robots:** The International Organization for Standardization defines a service robot as 'a robot that performs useful tasks for humans.' They can be further subdivided as:

1. Medical robots
2. Home robots
3. Defence robots

4. Entertainment robots
5. Agricultural robots
6. Educational robots

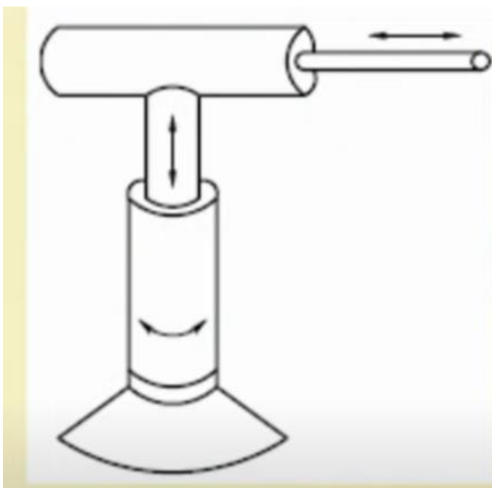
- **Medical robots** are professional service robots that are used in and out of hospital settings to improve the level of patient care. These robots reduce the workload of the medical staff, which allows them to spend more time caring directly for patients. Mobile medical robots are used for the delivery of medication and other sensitive materials in a hospital.
- **Home robots** automate tasks like cleaning and disinfecting.
- The primary purpose of **Education robots** is to make kids aware of their potential, utility, and help kids build their own robots using readymade kits. Educational robots are used extensively in schools, both in classrooms and in extracurricular activities.
- One of the most important uses of **robots in defence** is to ensure the safety of soldiers and civilians. For example, remotely operated vehicles (ROVs) are used to carry out dangerous tasks or activities in hazardous environments, drones are used for surveillance, and so on.
- **Agricultural robots** sense weather pattern and can adjust the watering of crop as needed, can be used for sowing, de-weeding, and harvesting crops.

Classifying Robots by their Kinematics or Locomotion: Robots can also be classified according to how they move – or not move.

- **Cartesian Robots:** these are perhaps the most common types of robots. They have three axes which are linear i.e, they can only move in straight lines rather than rotating and are mounted at right angles to each other. Because of their rigid structure, this type of robots usually can offer good levels of precision and repeatability. Cartesian robots are mostly used in the industrial and the manufacturing sector for pick and place type of operations.

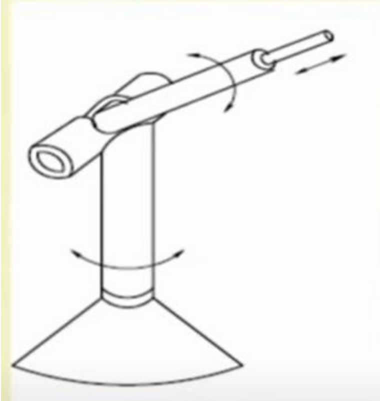
- **Cylindrical robots:** The body of this type of robot is such that the robotic arm can move up and down along a vertical member. The arm can rotate about that vertical axis and the arm can also extend or contract. This construction makes the manipulator able to work in a cylindrical space. They are used for assembly operations, spot welding and for die casting machines. These cannot reach the objects lying on the floor.

  Eg:Versatran 600



Spherical coordinate Robots(Polar coordinate Robots): This type of robot works in a spherical system. It can move in a bi-angular and single linear direction. SCARA Robots: SCARA stands for Selective Compliance Arm for Robotic Assembly. This type of robot has one linear and two rotary movements. Can be used to pick up objects lying on the floor. They are used for assembly purposes all over the world

Eg: Unimate 2000B



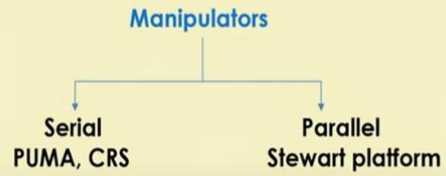
- **Revolute coordinate or Articulated Robots:** These are robots with a wide range of movements that include forward, backward, upward and downward motion. Because of their large work envelope, articulated robots can be used for several different applications like assembly, arc welding, material handling, machine tending, and packaging.

Rigidity and accuracy may not be good enough
Examples: T3, PUMA

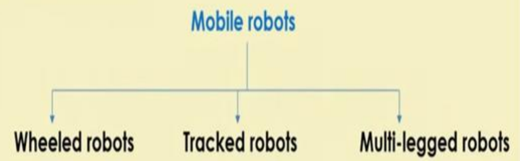
Note: PUMA stands for Programmable Universal Machine for Assembly

- **Based on Mobility Levels**

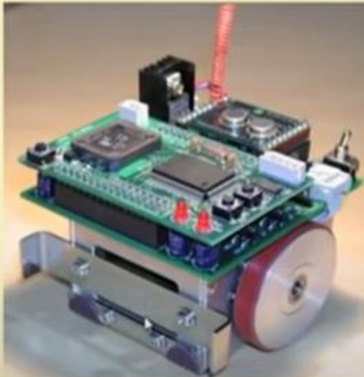
1. **Robots with fixed base (also known as manipulators)**



2. **Mobile robots**



- **Mobile Robots:** Robots with moving base are called as mobile robots.
- Wheeled robots can be used on smooth terrains.
- For rough terrains or for staircases we can go for multi-legged robots.
- If the terrain is neither to smooth nor too rough then we go for tracked robots.



Wheeled Robot

Six-legged Robot

❖ Based on the Type of Controllers

1. Non-Servo-Controlled Robots

❑ Open-loop control system

Examples: Seiko PN-100

- Less accurate and less expensive

2. Servo-Controlled Robots

❑ Closed-loop control system

Examples: Unimate 2000, PUMA, T^3

- More accurate and more expensive

Other types:

- **Airborne Robots:** these robots can fly through the air. Drones are an extremely popular example of flying robots.
- **Aquatic Robots:** These robots can work on or under water. They are mostly used for underwater exploration of oil, gas or minerals.

Joints and links

Two basic typestypes of joints are used in industrial robots

1)Revolute

2)Prismatic

The relative motion of the adjoining links of a joint is either rotary or linear depending on the type of joint.

Revolute joint: It is sketched in Fig. 1.8(a). The two links are jointed by a pin (pivot) about the axis of which the links can rotate with respect to each other.
Prismatic joint: It is sketched in Fig. 1.8(b). The two links are so jointed that these can slide (linearly move) with respect to each other. Screw and nut (slow linear motion of the nut), rack and pinon are ways to implement prismatic joints.

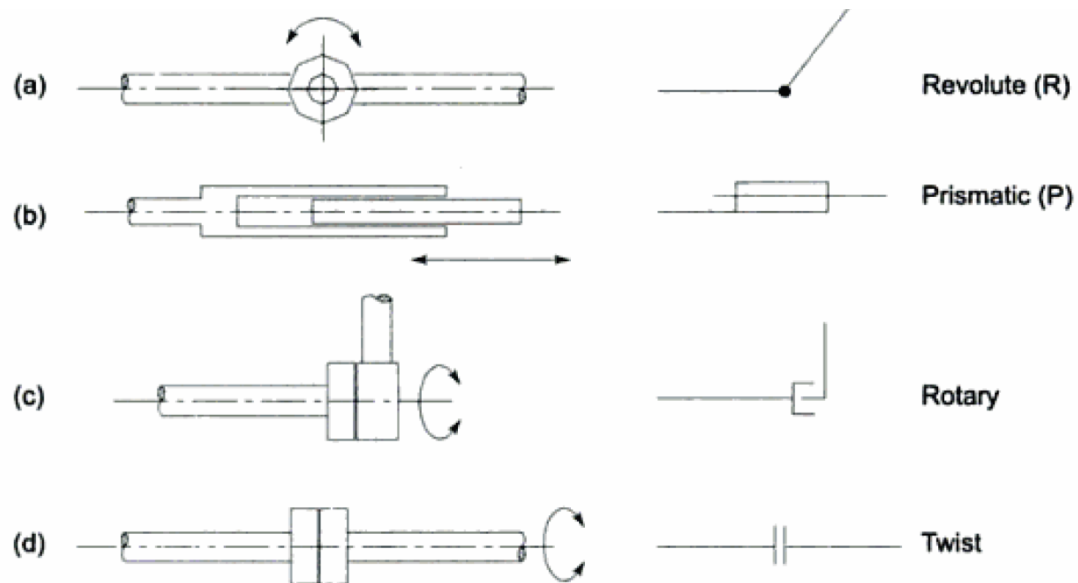


Fig. 1.8 *Joint types and their symbols*

Other types of possible joints used are: planar (one surface sliding over another surface); cylindrical (one link rotates about the other at 90° angle, Fig. 1.8(c)); and spherical (one link can move with respect to the other in three dimensions). Yet another variant of rotary joint is the 'twist' joint, where two links remain aligned along a straight line but one turns (twists) about the other around the link axis, Fig. 1.8(d).

- At a joint ,links are connected such that they can be made to move relative to each other by the actuators.
- A **rotary joint** allows a pure rotation of one link relative to the connecting link and **prismatic joint** allows a pure translation of one link relative to the connecting link.

The kinematic chain formed by joining two links is extended by connecting more links. To form a manipulator, one end of the chain is connected to the base or ground with a joint. Such a manipulator is an open kinematic chain. The end-effector is connected to the free end of the last link, as illustrated in Fig. 1.5. Closed kinematic chains are used in special purpose manipulators, such as parallel manipulators, to create certain kind of motion of the end-effector.

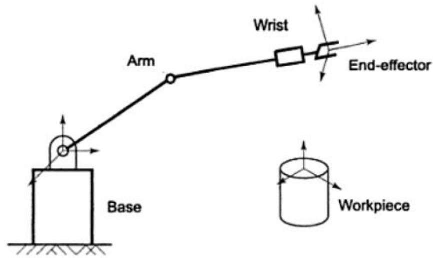


Fig. 1.5 *The base, arm, wrist, and end-effector forming the mechanical structure of a manipulator*

Degree of freedom

The number of independent movements that an object can perform in a 3-D space is called the number of *degrees of freedom* (DOF). Thus, a rigid body free in space has six degrees of freedom—three for position and three for orientation. These six independent movements pictured in Fig. 1.9 are:

(i) three translations (T_1, T_2, T_3), representing linear motions along three perpendicular axes, specify the position of the body in space.

(ii) three rotations (R_1, R_2, R_3), which represent angular motions about the three axes, specify the orientation of the body in space.

Note from the above that six independent variables are required to specify the location (position and orientation) of an object in 3-D space, that is, $2 \times 3 = 6$. Nevertheless, in a 2-D space (a plane), an object has 3-DOF—two translatory and one rotational. For instance, link 1 and link 2 in Fig. 1.6 have 3-DOF each.

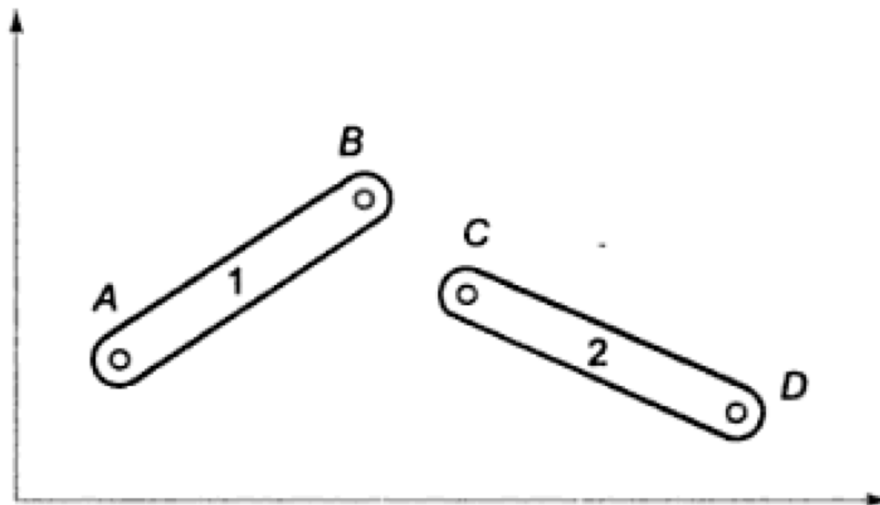


Fig. 1.6 *Two rigid binary links in free space*

Consider an open kinematic chain of two links with revolute joints at A and B (or C), as shown in Fig. 1.10. Here, the first link is connected to the ground by a joint at A. Therefore, link 1 can only rotate about joint 1 (J_1) with respect to ground and contributes one independent variable (an angle), or in other words, it contributes one degree of freedom. Link 2 can rotate about joint 2 (J_2) with respect to link 1, contributing another independent variable and so another DOF. Thus, by induction, conclude that an open kinematic chain with one end connected to the ground by a joint and the farther end of the last link free, has as many degrees of freedom as the number of joints in the chain. It is assumed that each joint has only one DOF.

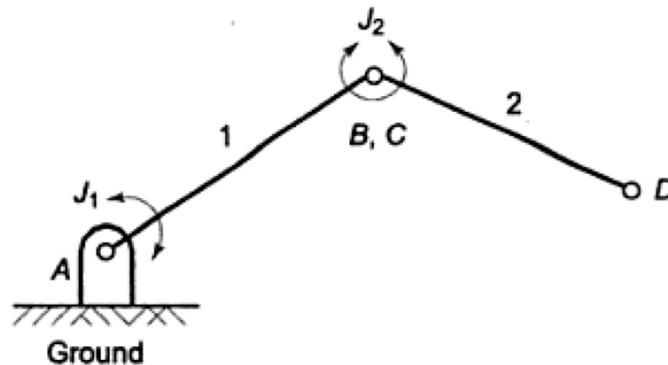


Fig. 1.10 *A two-DOF planar manipulator–two links, two joints*

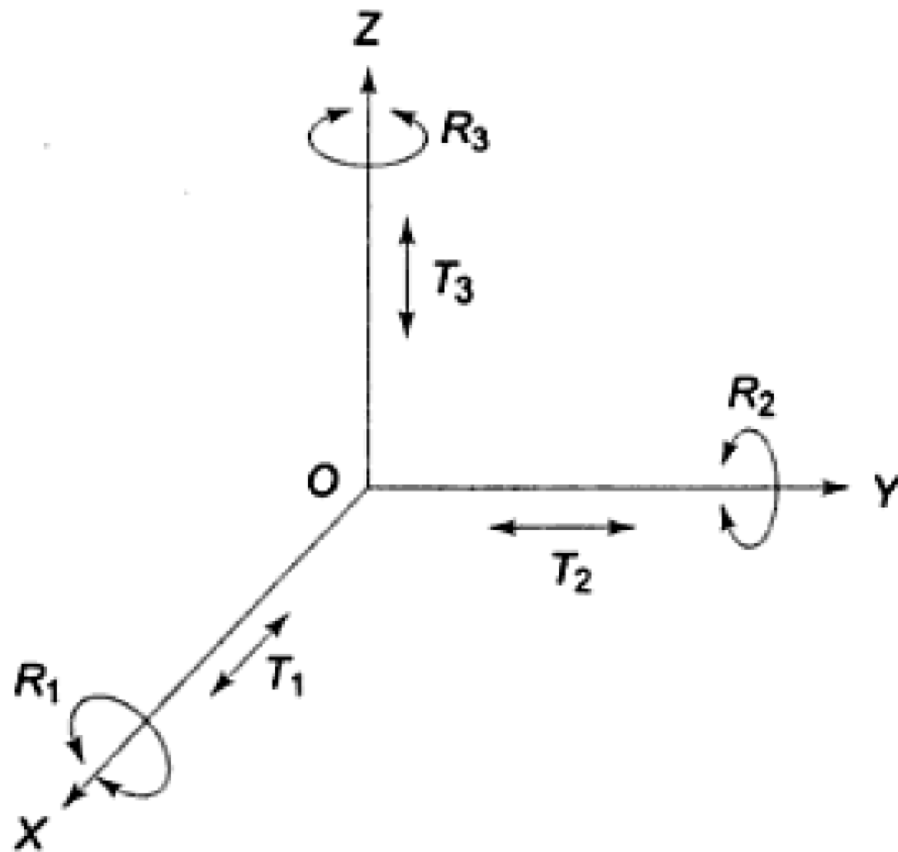


Fig 1.9: Representation of six degrees of freedom

DOF of a system: Defined as the minimum number of independent parameters/variables/coordinates needed to describe the system completely.

Notes

- A point in 2-D: 2 dof; in 3-D space: 3 dof
- A rigid body in 3-D: 6 dof
- Spatial Manipulator: 6 dof
- Planar Manipulator: 3 dof

Redundant Manipulator

Either a Spatial Manipulator with more than 6 dof or a Planar Manipulator with more than 3 dof

Under-actuated Manipulator

Either a Spatial Manipulator with less than 6 dof or a Planar Manipulator with less than 3 dof

Mobility/dof of Spatial Manipulator

Let us consider a manipulator with n rigid moving links and m joints

C_i: Connectivity of i-th joint; i = 1, 2, 3,………, m

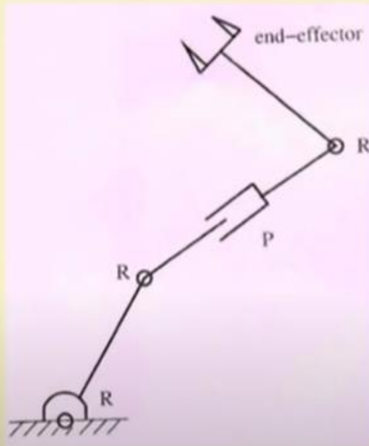
No. of constraints put by i-th joint $= (6 - C_i)$

Total no. of constraints $= \sum_{i=1}^{m}(6 - C_i)$

Mobility of the manipulator $M = 6n - \sum_{i=1}^{m}(6 - C_i)$

It is known as Grubler's criterion.

Serial planar manipulator



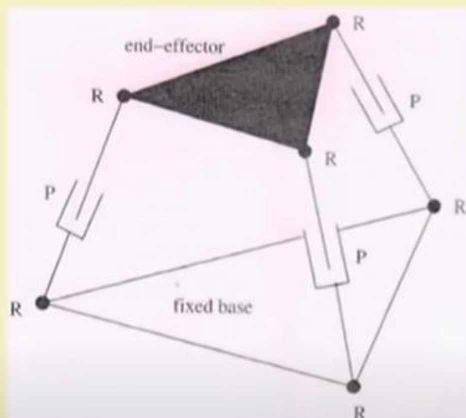
$n = 4, \quad m = 4$

$C_1 = C_2 = C_3 = C_4 = 1$

Mobility/dof:

$$M = 3n - \sum_{i=1}^{m} (3 - C_i) = 3 \times 4 - 8 = 4$$

Parallel planar manipulator



$n = 7, \quad m = 9$

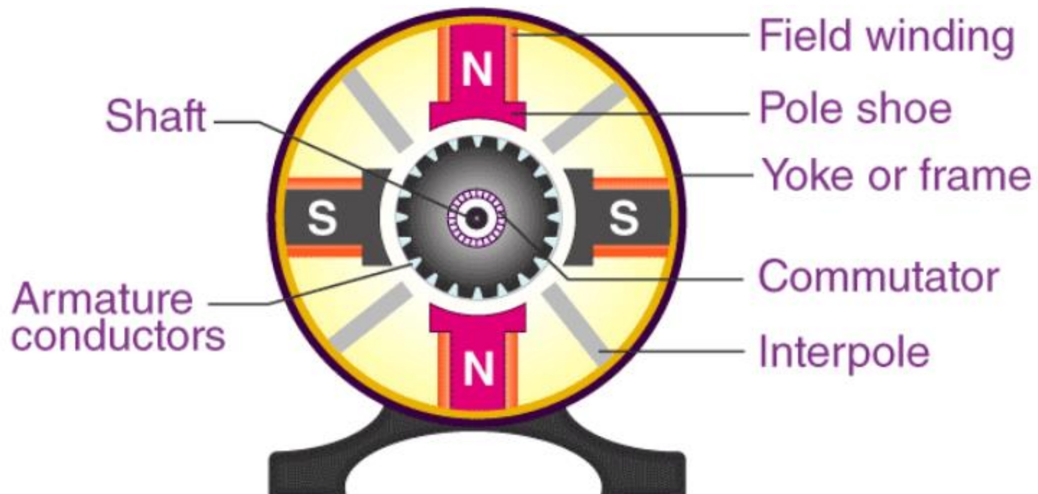
$C_i = 1, \qquad where\ i = 1, \ldots, 9$

Mobility/dof:

$$M = 3n - \sum_{i=1}^{m} (3 - C_i) = 3 \times 7 - 18 = 3$$

Motors:

- DC Motor: A DC motor is defined as a class of electrical motors that convert direct current electrical energy into mechanical energy.
- From the above definition, we can conclude that any electric motor that is operated using direct current or DC is called a DC motor.



Parts of a DC Motor

Armature or Rotor

The armature of a DC motor is a cylinder of magnetic laminations that are insulated from one another. The armature is perpendicular to the axis of the cylinder. The armature is a rotating part that rotates on its axis and is separated from the field coil by an air gap.

Field Coil or Stator

A DC motor field coil is a non-moving part on which winding is wound to produce a magnetic field. This electro-magnet has a cylindrical cavity between its poles.

Commutator and Brushes

Commutator

The commutator of a DC motor is a cylindrical structure that is made of copper segments stacked together but insulated from each other using mica. The primary function of a commutator is to supply electrical current to the armature winding.

Brushes

The brushes of a DC motor are made with graphite and carbon structure. These brushes conduct electric current from the external circuit to the rotating commutator. Hence, we come to understand that the commutator and the

brush unit are concerned with transmitting the power from the static electrical circuit to the mechanically rotating region or the rotor.

DC Motor Working:

A magnetic field arises in the air gap when the field coil of the DC motor is energised. The created magnetic field is in the direction of the radii of the armature. The magnetic field enters the armature from the North pole side of the field coil and “exits” the armature from the field coil’s South pole side.

The conductors located on the other pole are subjected to a force of the same intensity but in the opposite direction. These two opposing forces create a torque that causes the motor armature to rotate.

Working principle of DC motor

When kept in a magnetic field, a current-carrying conductor gains torque and develops a tendency to move. In short, when electric fields and magnetic fields interact, a mechanical force arises. This is the principle on which the DC motors work.

Types of DC motor

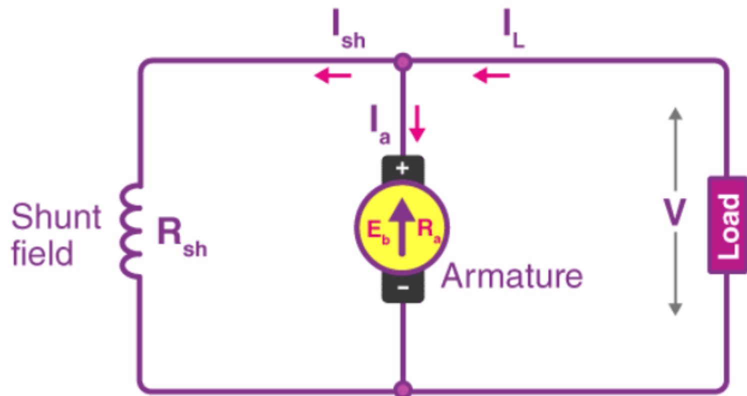
- Self Excited DC Motor
- Separately Excited DC Motor

Self Excited DC Motor

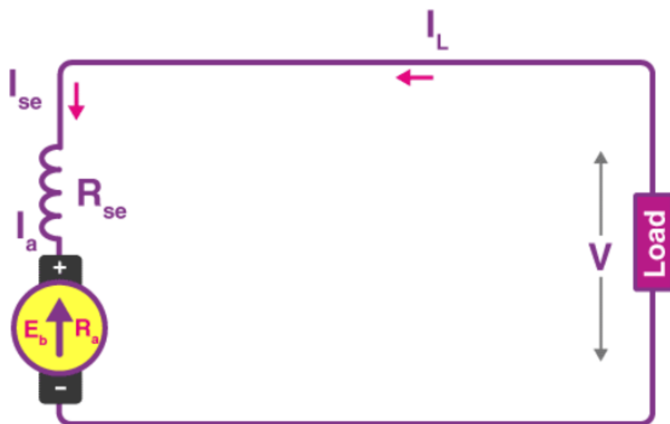
- In self-excited DC motors, the field winding is connected either in series or parallel to the armature winding. Based on this, the self-excited DC motor can further be classified as:

1. Shunt wound DC motor
2. Series wound DC motor
3. Compound wound DC motor

Shunt wound DC motor: In a shunt wound motor, the field winding is connected parallel to the armature



Series wound DC motor: In a series wound DC motor, the field winding is connected in series with the armature winding



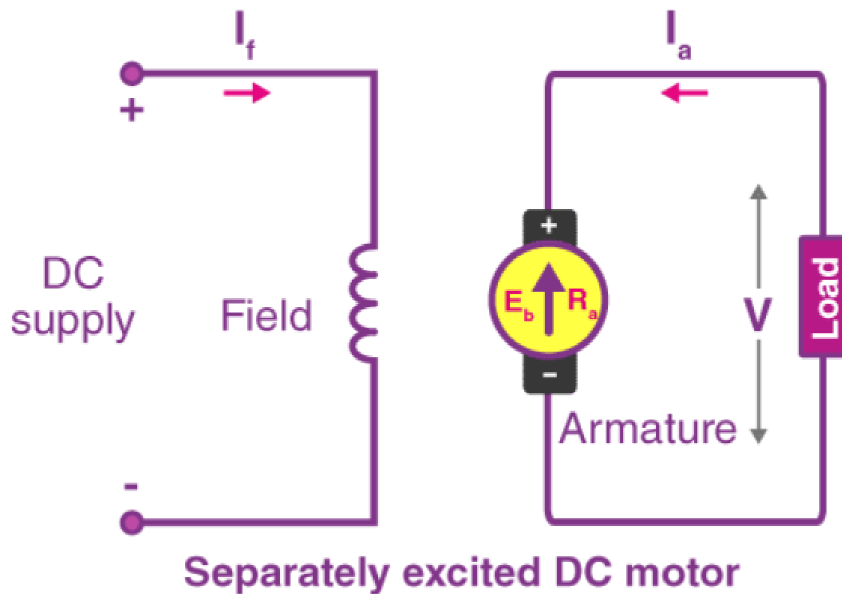
Compound wound DC motor:DC motors having both shunt and series field winding is known as Compound DC motor, The compound motor is further divided into:

Cumulative Compound Motor

Differential Compound Motor

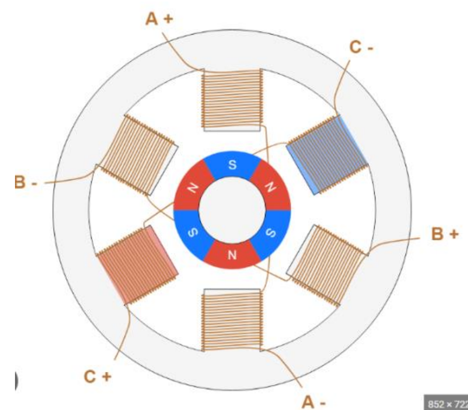
In a cumulative compound motor, the magnetic flux produced by both the windings is in the same direction. In a differential compound motor, the flux produced by the series field windings is opposite to the flux produced by the shunt field winding.

2) **Separately Excited DC Motor:** In a separately excited DC motor, the field coils are energised from an external source of DC supply



Stepper Motor:

- A stepper motor is an electromechanical device it converts electrical power into mechanical power. Also, it is a brushless, synchronous electric motor that can divide a full rotation into an expansive number of steps. The motor's position can be controlled accurately without any feedback mechanism, as long as the motor is carefully sized to the application.Construction & Working Principle:
- The construction of a stepper motor is fairly related to a DC motor. It includes a permanent magnet like Rotor which is in the middle & it will turn once force acts on it. This rotor is enclosed through a no. of the stator which is wound through a magnetic coil all over it. The stator is arranged near to rotor so that magnetic fields within the stators can control the movement of the rotor.

- The stepper motor can be controlled by energizing every stator one by one. So the stator will magnetize & works like an electromagnetic pole which uses repulsive energy on the rotor to move forward. The stator's alternative magnetizing as well as demagnetizing will shift the rotor gradually &allows it to turn through great control.
- The **stepper motor working principle** is Electro-Magnetism. It includes a rotor which is made with a permanent magnet whereas a stator is with electromagnets. Once the supply is provided to the winding of the stator then the magnetic field will be developed within the stator. Now rotor in the motor will start to move with the rotating magnetic field of the stator. So this is the fundamental working principle of this motor.
- 

- In this motor, there is a soft iron that is enclosed through the electromagnetic stators. The poles of the stator as well as the rotor don't depend on the kind of stepper. Once the stators of this motor are energized then the rotor will rotate to line up itself with the stator otherwise turns to have the least gap through the stator. In this way, the stators are activated in a series to revolve the stepper motor.

Driving Techniques

Single Excitation Mode

The basic method of driving a stepper motor is a single excitation mode. It is an old method and not used much at present but one has to know about this

technique. In this technique every phase otherwise stator next to each other will be triggered one by one alternatively with a special circuit. This will magnetize & demagnetize the stator to move the rotor forward.

Full Step Drive

In this technique, two stators are activated at a time instead of one in a very less time period. This technique results in high torque & allows the motor to drive the high load.

Half Step Drive

This technique is fairly related to the Full step drive because the two stators will be arranged next to each other so that it will be activated first whereas the third one will be activated after that. This kind of cycle for switching two stators first &after that third stator will drive the motor. This technique will result in improved resolution of the stepper motor while decreasing the torque.

Micro Stepping

- This technique is most frequently used due to its accuracy. The variable step current will supply by the stepper motor driver circuit toward stator coils within the form of a sinusoidal waveform. The accuracy of every step can be enhanced by this small step current. This technique is extensively used because it provides high accuracy as well as decreases operating noise to a large extent.

Based on the coil winding arrangements, a two-phase stepper motor is classified into two. They are:

1. Unipolar
2. Bipolar

1. Unipolar A unipolar stepper motor contains two windings per phase. The direction of rotation (clockwise or anticlockwise) of a stepper motor is controlled by changing the direction of current flow. Current in one direction flows through one coil and in the opposite direction flows through the other coil. It is easy to shift the direction of rotation by just switching the terminals to which the coils are connected. Figure 2.18 illustrates the working of a two-phase unipolar stepper motor.

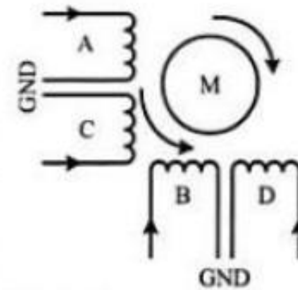


Fig. 2.18 **2-Phase unipolar stepper motor**

The coils are represented as A, B, C and D. Coils A and C carry current in opposite directions for phase 1 (only one of them will be carrying current at a time). Similarly, B and D carry current in opposite directions for phase 2 (only one of them will be carrying current at a time).

2. Bipolar A bipolar stepper motor contains single winding per phase. For reversing the motor rotation the current flow through the windings is reversed dynamically. It requires complex circuitry for current flow reversal. The stator winding details for a two phase unipolar stepper motor is shown in Fig. 2.19.

The stepping of stepper motor can be implemented in different ways by changing the sequence of activation of the stator windings. The different stepping modes supported by stepper motor are explained below.

Full Step In the full step mode both the phases are energised simultaneously. The coils A, B, C and D are energised in the following order:

Step	Coil A	Coil B	Coil C	Coil D
1	H	H	L	L
2	L	H	H	L
3	L	L	H	H
4	H	L	L	H

It should be noted that out of the two windings, only one winding of a phase is energised at a time.

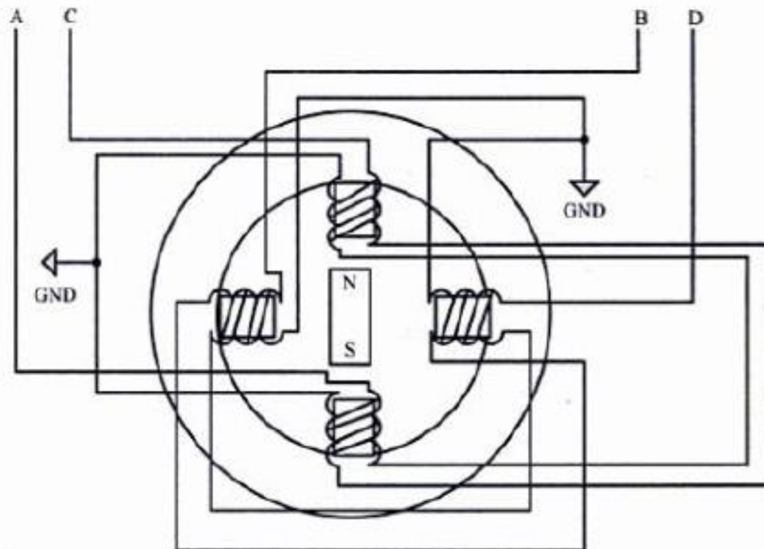


Fig. 2.19 **Stator Winding details for a 2 Phase unipolar stepper motor**

Wave Step In the wave step mode only one phase is energised at a time and each coils of the phase is energised alternatively. The coils A, B, C and D are energised in the following order:

Step	Coil A	Coil B	Coil C	Coil D
1	H	L	L	L
2	L	H	L	L
3	L	L	H	L
4	L	L	L	H

Half Step It uses the combination of wave and full step. It has the highest torque and stability. The coil energising sequence for half step is given below.

Step	Coil A	Coil B	Coil C	Coil D
1	H	L	L	L
2	H	H	L	L
3	L	H	L	L
4	L	H	H	L
5	L	L	H	L
6	L	L	H	H
7	L	L	L	H
8	H	L	L	H

The rotation of the stepper motor can be reversed by reversing the order in which the coil is energised.

Two-phase unipolar stepper motors are the popular choice for embedded applications. The current requirement for stepper motor is little high and hence the port pins of a microcontroller/processor may not be able to drive them directly. Also the supply voltage required to operate stepper motor varies normally in the range 5V to 24 V. Depending on the current and voltage requirements, special driving circuits are required to interface the stepper motor with microcontroller/processors. Commercial off-the-shelf stepper motor driver ICs are available in the market and they can be directly interfaced to the microcontroller port. ULN2803 is an octal peripheral driver array available from ON semiconductors and ST microelectronics for driving a 5V stepper motor. Simple driving circuit can also be built using transistors.

The following circuit diagram (Fig. 2.20) illustrates the interfacing of a stepper motor through a driver circuit connected to the port pins of a microcontroller/processor.

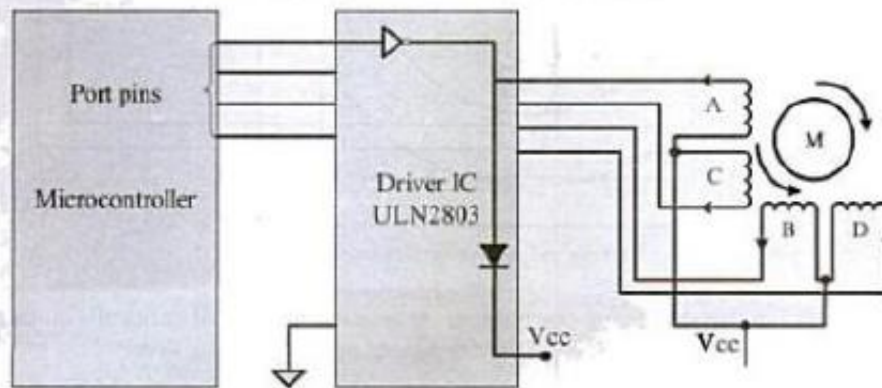
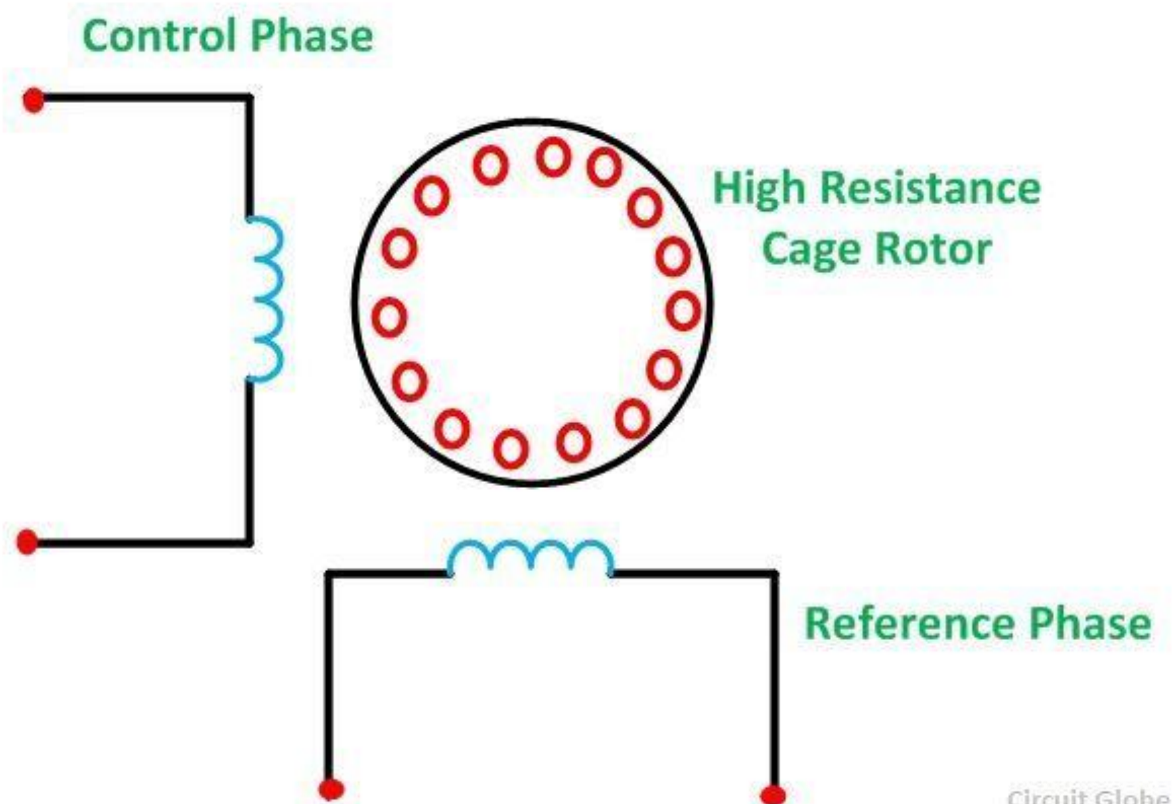


Fig. 2.20 Interfacing of stepper motor through driver circuit

Servo Motors

- A servo motor is a type of motor that can rotate with great precision. Normally this type of motor consists of a control circuit that provides feedback on the current position of the motor shaft, this feedback allows the servo motors to rotate with great precision. If you want to rotate an object at some specific angles or distance, then you use a servo motor. It is just made up of a simple motor which runs through a servo mechanism.
- If motor is powered by a DC power supply then it is called DC servo motor, and if it is AC-powered motor then it is called AC servo motor. For this tutorial, we will be discussing only about the DC servo motor working. Apart from these major classifications, there are many other types of servo motors based on the type of gear arrangement and operating characteristics.

- A servo motor usually comes with a gear arrangement that allows us to get a very high torque servo motor in small and lightweight packages. Due to these features, they are being used in many applications like toy car, RC helicopters and planes, Robotics, etc.
- Servo motors are rated in kg/cm (kilogram per centimeter) most hobby servo motors are rated at 3kg/cm or 6kg/cm or 12kg/cm. This kg/cm tells you how much weight your servo motor can lift at a particular distance. For example: A 6kg/cm Servo motor should be able to lift 6kg if the load is suspended 1cm away from the motors shaft, the greater the distance the lesser the weight carrying capacity. The position of a servo motor is decided by electrical pulse and its circuitry is placed beside the motor.



Servo Motor Working Mechanism

- It consists of three parts:

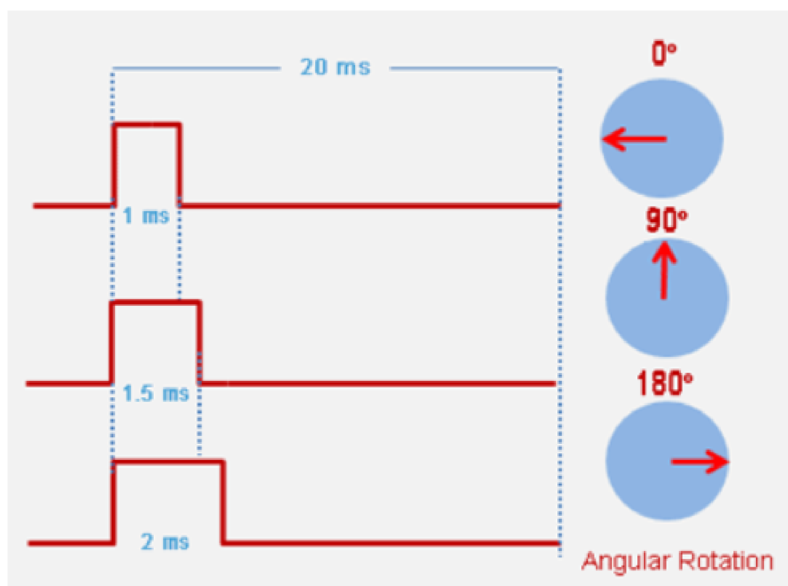
1. Controlled device

2. Output sensor
3. Feedback system

- It is a closed-loop system where it uses a positive feedback system to control motion and the final position of the shaft. Here the device is controlled by a feedback signal generated by comparing output signal and reference input signal.
- Here reference input signal is compared to the reference output signal and the third signal is produced by the feedback system. And this third signal acts as an input signal to the control the device. This signal is present as long as the feedback signal is generated or there is a difference between the reference input signal and reference output signal. So the main task of servomechanism is to maintain the output of a system at the desired value at presence of noises.
- Servo Motor Working Principle
- A servo consists of a Motor (DC or AC), a potentiometer, gear assembly, and a controlling circuit. First of all, we use gear assembly to reduce RPM and to increase torque of the motor. Say at initial position of servo motor shaft, the position of the potentiometer knob is such that there is no electrical signal generated at the output port of the potentiometer.
- Now an electrical signal is given to another input terminal of the error detector amplifier. Now the difference between these two signals, one comes from the potentiometer and another comes from other sources, will be processed in a feedback mechanism and output will be provided in terms of error signal. This error signal acts as the input for motor and motor starts rotating. Now motor shaft is connected with the potentiometer and as the motor rotates so the potentiometer and it will generate a signal. So as the potentiometer’s angular position changes, its output feedback signal changes.
- After sometime the position of potentiometer reaches at a position that the output of potentiometer is same as external signal provided. At this condition, there will be no output signal from the amplifier to the motor input as there is no difference between external applied signal and the

signal generated at potentiometer, and in this situation motor stops rotating.

- Servo motor works on **PWM (Pulse width modulation)** principle, means its angle of rotation is controlled by the duration of applied pulse to its Control PIN. Basically servo motor is made up of **DC motor which is controlled by a variable resistor (potentiometer) and some gears**. High speed force of DC motor is converted into torque by Gears. We know that WORK= FORCE X DISTANCE, in DC motor Force is less and distance (speed) is high and in Servo, force is High and distance is less. The potentiometer is connected to the output shaft of the Servo, to calculate the angle and stop the DC motor on the required angle.



- Servo motor can be rotated from 0 to 180 degrees, but it can go up to 210 degrees, depending on the manufacturing. This degree of rotation can be controlled by applying the **Electrical Pulse** of proper width, to its Control pin. Servo checks the pulse in every 20 milliseconds. The pulse of 1 ms (1 millisecond) width can rotate the servo to 0 degrees, 1.5ms can rotate to 90 degrees (neutral position) and 2 ms pulse can rotate it to 180 degree.

- All servo motors work directly with your +5V supply rails but we have to be careful about the amount of current the motor would consume if you

are planning to use more than two servo motors a proper servo shield should be designed.

Types of Gears:

- A gear is a toothed cylindrical or roller shape component of a machine which meshes with another toothed cylindrical to transmit power from one shaft to another. It is mainly used to obtain different torque and speed ratio or changing the direction of driving shaft and driven shaft.
- **Principle:**
- It works on the basic principle of thermodynamic which state that energy is neither be created or destroyed or we can say it is conservative. it can be converted into one form to another. We know that power is the function of speed and torque or we can say that power is product of torque (Force in rotary motion) and speed (P = TV) of the shaft. So when we connect a small gear on driving shaft and a larger gear on driven shaft, the speed decreases of driven shaft per unit rotation of driving shaft.

Types of Gears:

Gears can be classified in various types according to construction of teeth, Use, the direction of motion transfer etc. but basically it is classified according to design of teeth.

1. Spur Gear:

These gears are used to transmit the power in same plane or when the driving and driven shafts are parallel to each other. In this type of gear teeth are cut parallel to the axis of the shafts so when is meshes with another spur gear it transmit the power in parallel shaft and when it connects with the helical gear it will transmit power at an angle from the driving axis.

Spur Gear

2. Helical Gear:

- On the helical gears teeth are cut at an angle from the axis of it. It has cylindrical roller with helicoid teeth. The main advantage of helical gears is that they work with less noise and vibration because the load is distributed on the full helix as compared to spur gears. It also has less wear and tear due to which they are widely used in industries. It also used for transmit power in parallel shaft but sometime they are used to transmit power in non-parallel shaft also. In the helical gears if the pinion (driving gear) is cut with right handed teeth then the gear (driven gear) is cut with left handed of in opposite direction.

Helical Gear

3. Double Helical or Herringbone Gear:

- This gear has both right and left handed teeth on one gear. This gear is use to provide additional shear area on gear which further required for higher torque transmission. This is same as helical.

Double Helical Gear

4. Bevel Gear:

- This gear is used to transmit power between perpendiculars. The driving shaft and driven shaft makes a right angle with each other and both the axis of shaft meets each other at one point. This gear has helical or spiral teeth on a conical shaped geometry and meshes with the same gear.

Bevel Gear

5. Rack and Pinion Gear:

- This gear is used in steering system of automobile. In this type of gear, teeth are cut on a straight rectilinear geometry know as rack and one

spur gear known as pinion. This is used to transmit rotary motion to linear motion. It is seen as the infinite radius driven gear.

Rack and Pinion Gear

6. Worm Gear:

- This type of gear is used to transmit the power in nonintersecting shaft which makes right angle. In this type of arrangement the driving gear is a screw gear and the driven gear is helical gear or gear with spiral teeth as shown in figure.

Worm Gear

Software used for robot programming:

- A "robotics software platform" is a software package which simplifies programming of several kinds of robotic devices by providing
- a unified programming environment;
- a unified service execution environment;
- a set of reusable components;

- a debugging/simulation environment;
- a package of "drivers" for most wide-spread robotics hardware
- a package of common facilities such as computer vision, navigation or robotic arm control

Table 1 Robotic software platforms

Platform	Type	
Evolution Robotics ERSP	Platform	Commercial
Microsoft Robotics Studio	Platform	Commercial
		Free of charge for research and hobby
OROCOS	Machine and robot control libraries	Open source & free
Skilligent	Robot learning add-on	Commercial
URBI	Platform	Commercial
Webots	Simulation environment	Commercial
Player, Stage, Gazebo	Platform	Open Source & Free
iRobot AWARE	Platform	Commercial
OpenJAUS	Platform	Open source
CLARAty	Platform	Open source

1.Offline Programming

- RoboDK, offline programming software provides a way for you to program your industrial robot without needing to be physically connected to the robot at the time. This means that you don't need to take the robot out of production to program it. It reduces downtime, improves the quality of programming, and allows you to change between product lines quickly, amongst other benefits.

2. Simulators

Robot simulators come in many forms. Some only allow for simple 2D simulation of specific aspects of robotics whilst others include 3D simulation with complex physics engines and realistic environments.

- As well as being an offline programming tool, RoboDK is also a great simulator. It is simple enough to allow you to easily program your robot whilst being powerful enough to handle many different use cases.

3. Middleware:

Middleware is the “software glue” that helps robot builders to avoid reinventing the wheel when they are designing a new robotic system. Robot middleware provides a framework for running and managing complex robotic systems from a single unified interface. f you were building your own robotic system with multiple components or looking to coordinate multiple robots, you might use middleware

4**. Mobile Robot Planning :**Mobile robots are programmed in a different way from other robots which means using a different type of software too. For example, **path planners** are used to program the route that the robot will take through the environment while **obstacle avoidance** algorithms react to changes in the moment.

5. Real-Time Path Planning

Path planning software is used in many areas of robotics. Basic path planners, are simply used to speed up the programming phase for industrial robotics. Real-time path planning is much more complex and is based on teaching–learning-based optimization (TLBO) USING AI.

6. UAV (Drone) Control

A growing type of robotic software is drone control. This refers to any software which is used to program and coordinate unmanned aerial vehicles (UAVs/drones). DroneDeploy, PIX4D are examples of software used in drone control

(https://surveyinggroup.com/top-5-drone-mapping-softwares-that-you-will-need-on-your-project)

7. Artificial Intelligence for Robots

- Artificial intelligence (AI) has been used with robotics for many years — almost as long as robotics have been around. However, there has recently been a rising number of software solutions specifically for using AI with robots in particular application areas. As with the other types of robot software, AI tends to be focused on specific aspects of these applications, such as analyzing images collected in agricultural settings,

filtering operational data in manufacturing environments, or coordinating swarms of mobile robots in logistics.

UNIT – 3

The AVR RISC microcontroller architecture

AVRMicrocontroller:

Microcontroller: MicrocontrollercanbetermedasasingleonchipcomputerwhichincludesnumberofperipheralslikeRAM,EEPROM,Timersetc.,requiredtoperformsomepredefinedtask.

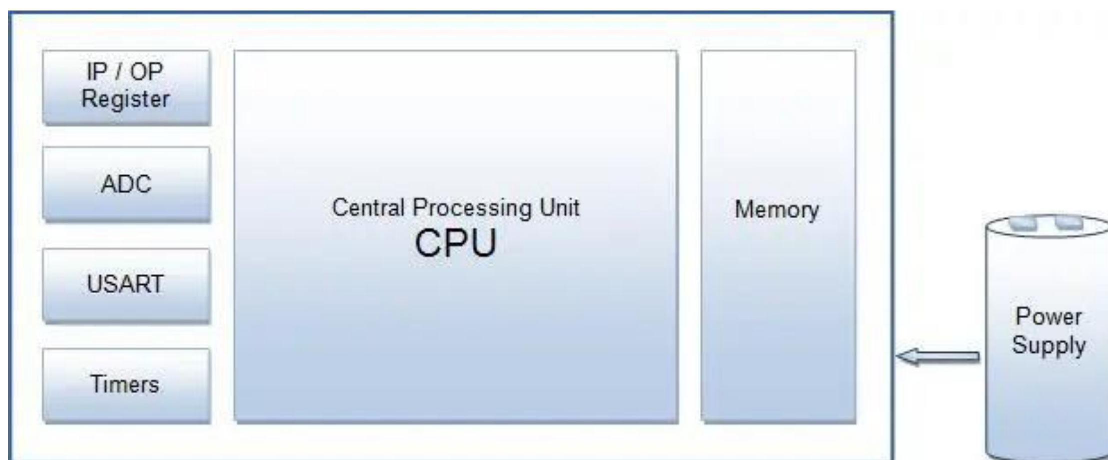


Fig. 1: Block Diagram Showing Architecture of AVR Microcontroller

Doesthismeanthatthe**microcontroller**isanothernameforacomputer…?TheanswerisNO!

The computer on one hand is designed to perform all the general purpose tasks on a single machine like you can use acomputer to run a software to perform calculations or you can use a computer to store some multimedia file or to accessinternetthroughthebrowser,whereasthemicrocontrollersaremeantto

performonlythespecifictasks,fore.g.,switchingtheACoffautomaticallywhenr oomtemperaturedropstoacertaindefinedlimitandagainturningitONwhente mperaturerisesabovethedefinedlimit.

There are number of popular families of microcontrollers which are used in different applications as per their capability andfeasibilitytoperformthedesiredtask,mostcommonoftheseare**8051**,**AVR** andPICmicrocontrollers.Inthisarticlewewillintroduceyouwith**AVR**familyofm icrocontrollers.

History of AVR

AVRwasdevelopedintheyear1996byAtmel Corporation. The architecture of **AVR** was developed by Alf-EgilBogen and VegardWollan. AVR derives its name from its developers and stands for **Alf-EgilBogenVegardWollanRISCmicrocontroller**,alsoknownas**A**dvanced**V**irtua l**R**ISC.TheAT90S8515wasthefirstmicrocontrollerwhichwasbasedon**AVRarchitecture**howeverthefirstmicrocontrollertohitthecommercialmarketwasAT90S12 00intheyear1997.

AVRmicrocontrollersareavailableinthreecategories:

1. **TinyAVR**–Lessmemory,smallsize,suitableonlyforsimplerapplications
2. **MegaAVR**–
Thesearethemostpopularoneshavinggoodamountofmemory(upto256KB),h ighernumberofinbuiltperipheralsandsuitableformoderatetocomplexapplica tions.
3. **XmegaAVR**–

Usedcommerciallyforcomplexapplications,whichrequirelargeprogrammemoryandhighspeed.

ThefollowingtablecomparestheabovementionedAVRseriesofmicrocontrollers:

Series Name	Pins	Flash Memory	Special Feature
TinyAVR	6-32	0.5-8 KB	Small in size
MegaAVR	28-100	4-256KB	Extended peripherals
XmegaAVR	44-100	16-384KB	DMA , Event System included

Importance of AVR

What's special about AVR?

They are fast: **AVR microcontroller** executes most of the instructions in single execution cycle. AVRs are about 4 timesfaster than PICs, they consume less power and can be operated in different power saving modes. Let's do the comparisonbetweenthethreemostcommonlyusedfamiliesofmicrocontrollers.

	8051	PIC	AVR

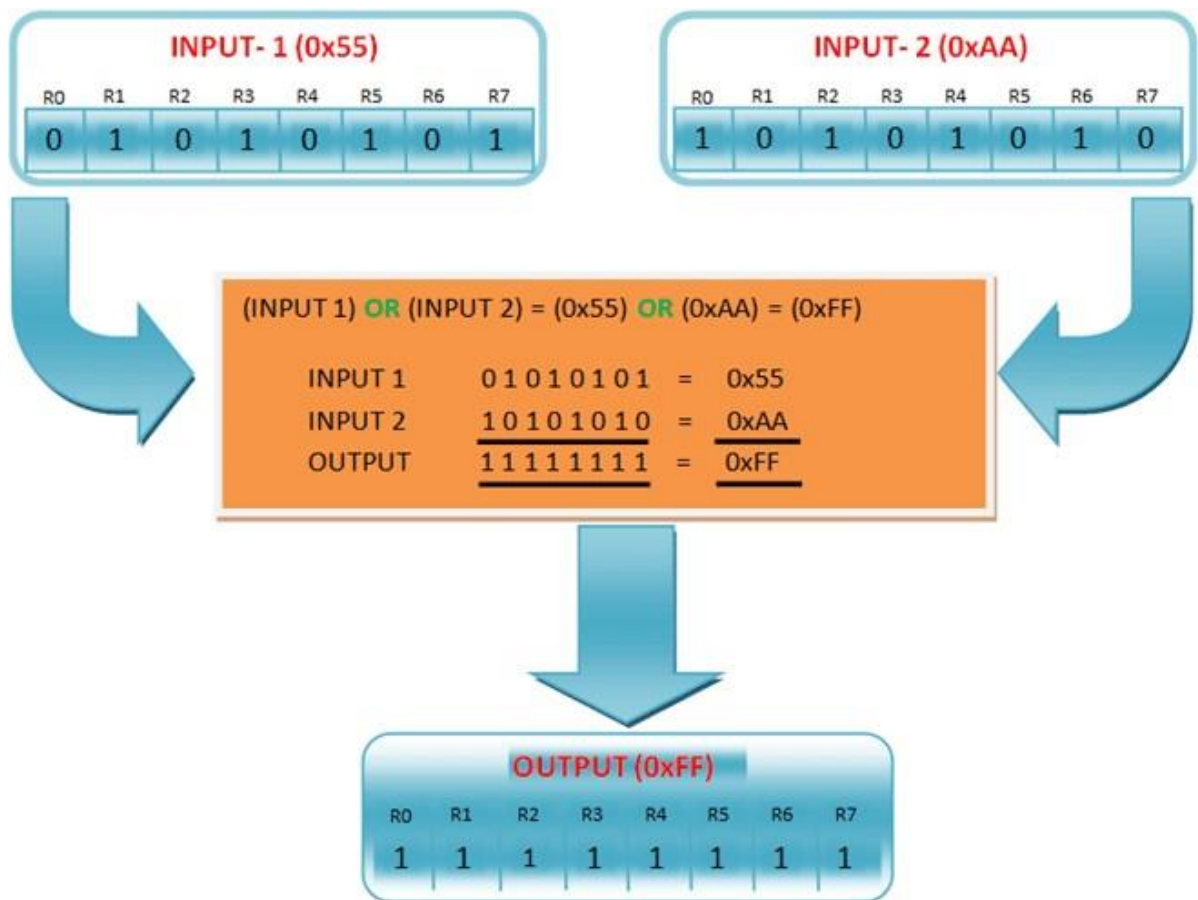
SPEED	Slow	Moderate	Fast
MEMORY	Small	Large	Large
ARCHITECTURE	CISC	RISC	RISC
ADC	Not Present	Inbuilt	Inbuilt
Timers	Inbuilt	Inbuilt	Inbuilt
PWM Channels	Not Present	Inbuilt	Inbuilt

AVRisan8-bitmicrocontrollerbelongingtothefamilyofReducedInstructionSetComputer (**RISC**).InRISCarchitecturetheinstruction set of the computer are not only fewer in number but also simpler and faster in operation. The other type ofcategorizationisCISC(ComplexInstructionSetComputers).Clicktofindoutdifferencesbetween RISCandCISC.Wewill exploremoreonthiswhenwewilllearnaboutthearchitectureofAVRmicrocontrollersinfollowingsection.

Let'sseewhatallthismeans.Whatis8-bit?Thismeansthatthemicrocontrolleriscapableoftransmittingandreceiving 8-bitdata. The input/output registers available are of 8-bits. The AVR family controllers have register based architecture whichmeans that both the operands for an operation are stored in a register and the result of the operation is also stored in

aregister.FollowingfigureshowsasimpleexampleperformingORoperationbet

weentwoinputregistersandstoringthevalueinOutputRegister.



Block Diagram Showing Simple Example Carrying Out OR Operation Between Two Input Registers And ValueStorage In Output Register

TheCPUtakesvaluesfromtwoinputregistersINPUT-1andINPUT-2,performsthelogicaloperationandstoresthevalueinto theOUTPUTregister.Allthishappensin1executioncycle.

InourjourneywiththeAVRwewillbeworkingonAtmega16microcontroller,whichisa40-pinICandbelongstothemegaAVRcategoryofAVRfamily.SomeofthefeaturesofAtmega16are:

- 16KBofFlashmemory
- 1KBofSRAM
- 512BytesofEEPROM
- Availablein40-PinDIP
- 8-Channel10-bitADC
- Two8-bitTimers/Counters

- One16-bitTimer/Counter
- 4PWMChannels
- InSystemProgrammer(ISP)
- SerialUSART
- SPIInterface
- Digitalto AnalogComparator.

Architecture of AVR

The AVR microcontrollers are based on the advanced RISC architecture and consist of 32 x 8-bit general purpose workingregisters. Within one single clock cycle, AVR can take inputs from two general purpose registers and put them to ALU forcarryingouttherequestedoperation,andtransferbacktheresulttoanarbitraryregister.TheAL Ucanperformarithmeticaswellaslogicaloperations overtheinputsfromtheregisterorbetweentheregisterandaconstant.Singleregisteroperationsli ketakingacomplementcan also be executed in ALU. We can see that AVR does not have any register like accumulator as in 8051 family ofmicrocontrollers;theoperationscanbeperformedbetweenanyoftheregistersandcanbestore dineitherofthem.

AVRfollowsHarvardArchitectureformatinwhichtheprocessorisequippedwithseparateme moriesandbusesforProgramand the Data information. Here while an instruction is being executed, the next instruction is pre-fetched from the programmemory.

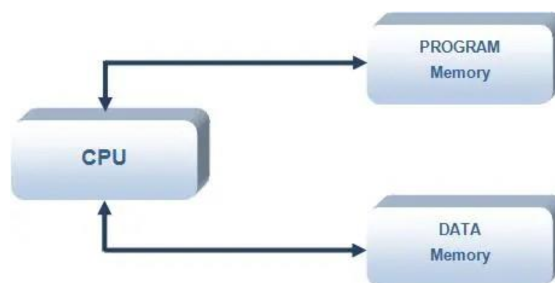


Fig. 3: Block Diagram Of memory architecture In AVR

Since AVR can perform single cycle execution, it means that AVR can execute 1 million

instructions per second if cyclefrequency is 1MHz. The higher is the operating frequency of the controller, the higher will be its processing speed. We need tooptimizethepowerconsumptionwithprocessingspeedandhenceneedtoselecttheoperatingf requencyaccordingly.

ThereaaretwoflavorsforAtmega16microcontroller:

1. **Atmega16**:-Operatingfrequencyrangeis0–16MHz.
2. **Atmega16L**:-Operatingfrequencyrangeis0–8MHz.

Ifweareusingacrystalof8MHz=8x10^6Hertz=8Millioncycles,thenAVRcanexecute8millioninstructions.

Naming Convention.!

The **AT** refers to Atmel the manufacturer, **Mega** means that the microcontroller belong to MegaAVR category, **16** signifies thememoryofthecontroller,whichis16KB.

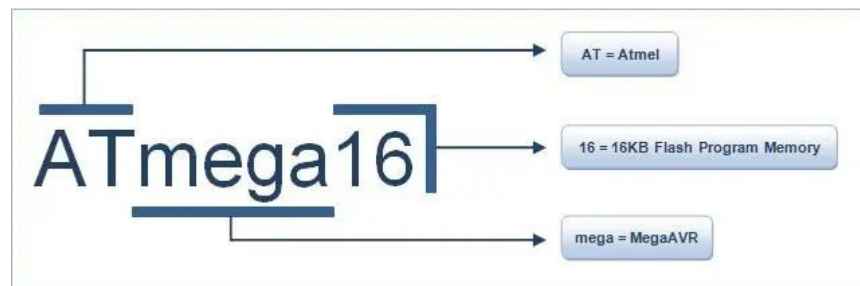


Fig. 4: Naming Convention Of AVR Microcontroller

Architecture Diagram:Atmega16

Followingpointsexplainthebuildingblocksof**Atmega16architecture**:

- **I/OPorts:**Atmega16hasfour(PORTA,PORTB,PORTCandPORTD)**8-bit**input-outputports.

- **Internal Calibrated Oscillator:** Atmega16 is equipped with an internal oscillator for driving its clock. By defaultAtmega16 is set to operate at internal calibrated oscillator of 1 MHz. The maximum frequency of internal oscillator

is8Mhz.Alternatively,ATmega16canbeoperatedusinganexternalcrystaloscillatorwithamaximumfr equencyof16MHz.Inthiscaseyouneedtomodifythefusebits.(FuseBitswillbeexplainedinaseparatetutorial).

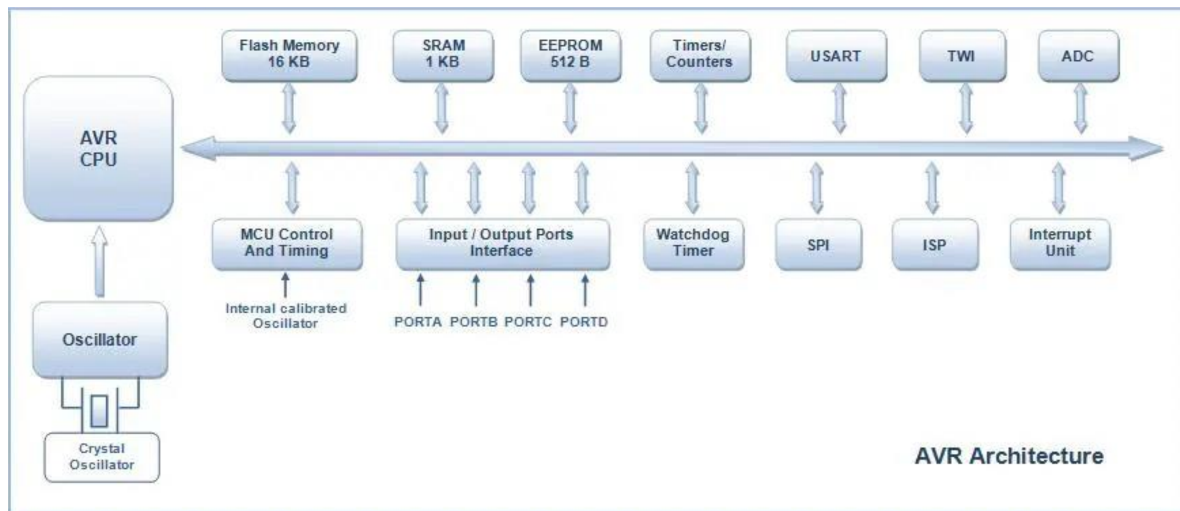


Fig. 5: Block Diagram Explaining AVR Architecture

ADC Interface: Atmega16 is equipped with an 8 channel ADC (**Analog to Digital Converter**) with a resolution of 10-bits.ADCreadstheanaloginputfore.g.,asensorinputandconvertsitintodigitalinformationwhichisunderstandablebythemicrocontroller.

· **Timers/Counters:** Atmega16 consists of two 8-bit and one 16-bit timer/counter. Timers are useful for generatingprecisionactionsfore.g.,creatingtimedelaysbetweentwooperations.

· **WatchdogTimer:**Watchdogtimerispresentwithinternaloscillator.Watchdogtimercontinuouslymonitorsandresetsthecontrollerifthecodegetsstuckatanyexecutionactionformorethanadefinedtimeinterval.

· **Interrupts:** Atmega16 consists of 21 interrupt sources out of which four are external. The remaining are internalinterruptswhichsupporttheperipheralslikeUSART,ADC,Timersetc.

· **USART:UniversalSynchronousandAsynchronousReceiverandTransmitter**interfaceisavailableforinterfacingwithexternaldevicecapableofcommunicatingserially(datatransmissionbitbyb

it).

GeneralPurposeRegisters: Atmega16isequippedwith32generalpurposeregisterswhichare coupleddirectlywiththeArithmeticLogicalUnit(ALU)ofCPU.

· **Memory:**Atmega16consistofthreedifferentmemorysections:

1. **FlashEEPROM**:FlashEEPROMorsimpleflashmemoryisusedtostoretheprogramdu mpedorburntbytheuseron to the microcontroller. It can be easily erased electrically as a single unit. Flash memory is non-volatile i.e., itretains the program even if the power is cut-off. Atmega16 is available with 16KB of in system programmable FlashEEPROM.

2. **Byte Addressable EEPROM**: This is also a nonvolatile memory used to store data like values of certain variables.Atmega16 has 512 bytes of EEPROM, this memory can be useful for storing the lock code if we are designing anapplicationlikeelectronicdoorlock.

3. **SRAM**: Static Random Access Memory, this is the volatile memory of microcontroller i.e., data is lost as soon aspower is turned off. Atmega16 is equipped with 1KB of internal SRAM. A small portion of SRAM is set aside forgeneralpurposeregistersusedbyCPUandsomefortheperipheralsubsystemsofthemicrocont roller.

· **ISP:**AVRfamilyofcontrollershave**InSystemProgrammable**FlashMemorywhichcan beprogrammedwithoutremovingtheICfromthecircuit,ISPallowstoreprogramthecontrollerwh ileitisintheapplicationcircuit.

· **SPI:SerialPeripheralInterface**,SPIportisusedforserialcommunicationbetweentwodevic esonacommonclocksource.ThedatatransmissionrateofSPIismorethanthatofUSART.

· **TWI:TwoWireInterface**(TWI)canbeusedtosetupanetworkofdevices,manydevicescanb econnectedoverTWIinterfaceforminganetwork,thedevicescansimultaneouslytransmitandre ceiveandhavetheirownuniqueaddress.

· **DAC:**Atmega16isalsoequippedwitha**DigitaltoAnalogConverter**(DAC)interfacewhichca nbeusedforreverseactionperformedbyADC.DACcanbeusedwhenthereisaneedofconvertingadigit

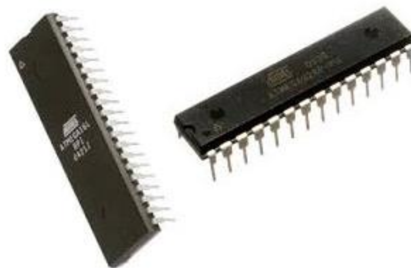
alsignaltoanalogsignal.

MegaAVR Family

VariousmicrocontrollerofMegaAVRseries:

Part Name	ROM	RAM	EEPROM	I/0 Pins	Timer	Interrupts	Operation Voltage	Operating frequency	Packaging
ATmega8	8KB	1KB	512B	23	3	19	4.5-5.5 V	0-16 MHz	28
ATmega8L	8KB	1KB	512B	23	3	19	2.7-5.5 V	0-8 MHz	28
ATmega16	16KB	1KB	512B	32	3	21	4.5-5.5 V	0-16 MHz	40
ATmega16L	16KB	1KB	512B	32	3	21	2.7-5.5 V	0-8 MHz	40
ATmega32	32KB	2KB	1KB	32	3	21	4.5-5.5 V	0-16 MHz	40
ATmega32L	32KB	2KB	1KB	32	3	21	2.7-5.5 V	0-8 MHz	40

AVR Microcontroller:



AVR microcontroller is an electronic chip manufactured by Atmel, which has several advantages over other types of microcontroller.

We can understand microcontroller by comparing it with Personal Computer (PC), which has a motherboard inside it. In that motherboard a microprocessor (AMD, Intel chips) is used

that provides the intelligence, EEPROM and RAM memories for interfacing to the system like serial ports, display interfaces and disk drivers. A microcontroller has all or most of these features built into a single chip, therefore it doesn?t require a motherboard and any other components.

AVR microcontroller comes in different configuration, some designed using surface mounting and some designed using hole mounting. It is available with 8-pins to 100-pins, any microcontroller with 64-pin or over is surface mount only.

Some mostly used AVR microcontrollers are:-

- ATmega8 microcontroller
- ATmega16 microcontroller
- ATmega32 microcontroller
- ATmega328 microcontroller

ATmega32-8 Bit AVR MicroController:

The AVR microController is based on the advanced Reduced Instruction Set Computer (RISC) architecture. ATmega32 microController is a low power CMOS technology based controller. Due to RISC architecture AVR microcontroller can execute 1 million of instructions per second if cycle frequency is 1 MHz provided by crystal oscillator.

40 pin DIP Photograph of ATmega32

Key Features:

Consider some general features of ATmega32 microcontroller is:-

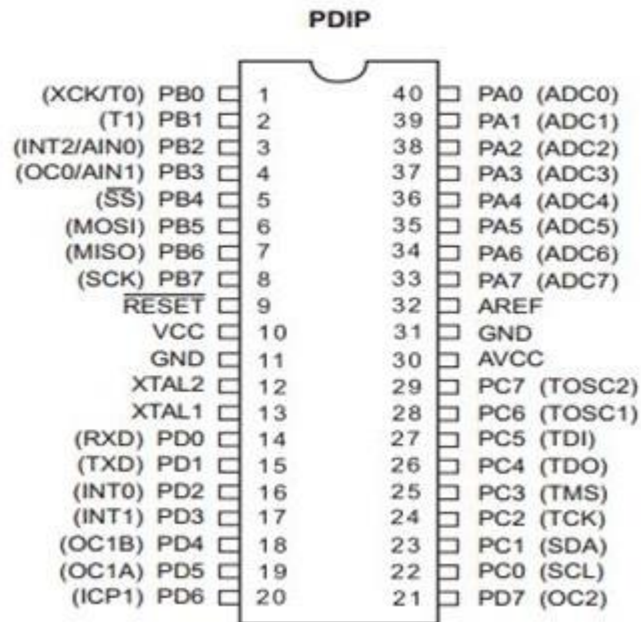
- 2 Kilo bytes of internal Static RAM
- 32 X 8 general working purpose registers
- 32 Kilo bytes of in system self programmable flash program memory.
- 1024 bytes EEPROM
- Programmable serial USART
- 8 Channel, 10 bit ADC
- One 16-bit timer/counter with separate prescaler, compare mode and capture mode.
- Available in 40 pin DIP, 44-pad QFN/MLF and 44-lead QTFP
- Two 8-bit timers/counters with separate prescalers and compare modes
- 32 programmable I/O lines
- In system programming by on-chip boot program
- Master/slave SPI serial interface
- 4 PWM channels
- Programmable watch dog timer with separate on-chip oscillator

Special Microcontroller Features:

- External and internal interrupt sources
- Six sleep modes: Idle, ADC noise reduction, power-save, power-down, standby and extended standby.
- Power on reset and programmable brown-out detection.
- Internal calibrated RC oscillator

ATmega32 Microcontroller Pin Diagram

For explaining the ATmega32 Microcontroller Pin diagram, consider a 40-pin Dual Inline Package (DIP) of microcontroller integrated circuit is:



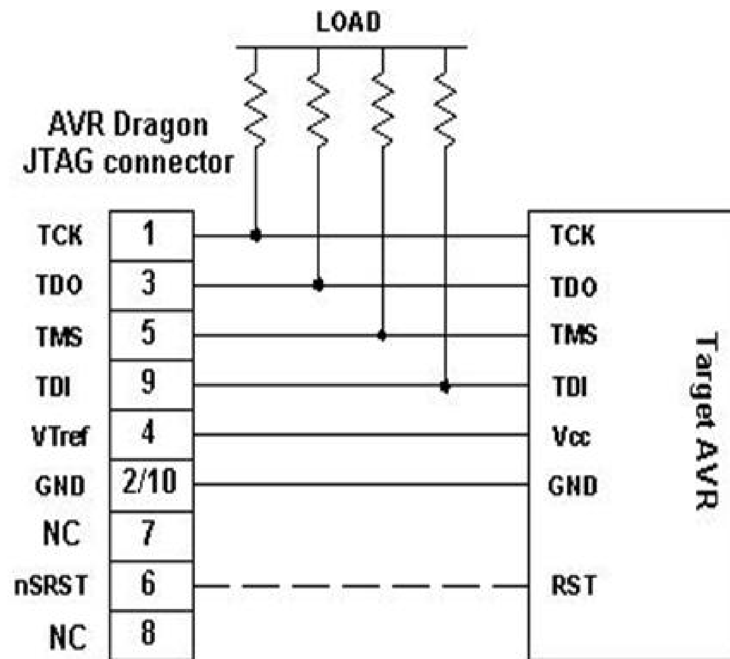
Pin Descriptions:

Port A (PA7-PA0): Port A serves as analog inputs for A/D converter. It also acts as an 8-bit bidirectional I/O port if the A/D converter is not used internally.

Port B (PB7-PB0) and Port D (PD7-PD0): These ports are 8-bit bidirectional I/O ports. Their output buffers have symmetrical drive characteristics with high source and sink capability. As inputs, these are pulled low if the pull-up resistors are used. It also provides various special functional features of the ATmega32.

Port C (PC7-PC0): Port C is an 8-bit bidirectional I/O port. If the Joint Test Action Group (JTAG) interface is enabled, the pull-up resistors on pins PC2 (TCK), PC3 (TMS), and PC5 (TDI) will be activated.

Consider the interfacing of Joint Test Action Group (JTAG) using Port C of ATmega32 is:-



Vcc: Digital voltage supply

GND: Ground

RESET: It is a RESET pin which is utilized to set the microcontroller ATmega32 to its primary value. During the beginning of an application the RESET pin is to be set elevated for two machine rotations.

XTAL1: It is an input for the inverting oscillator amplifier and input to an internal clock operating circuit.

XTAL2: It is an output from an inverting oscillator amplifier.

AVcc: It is a supply voltage pin for A/D converter and Port A. It must be connected with Vcc.

AREF: AREF is an analog signal reference pin for the analog to digital converter.

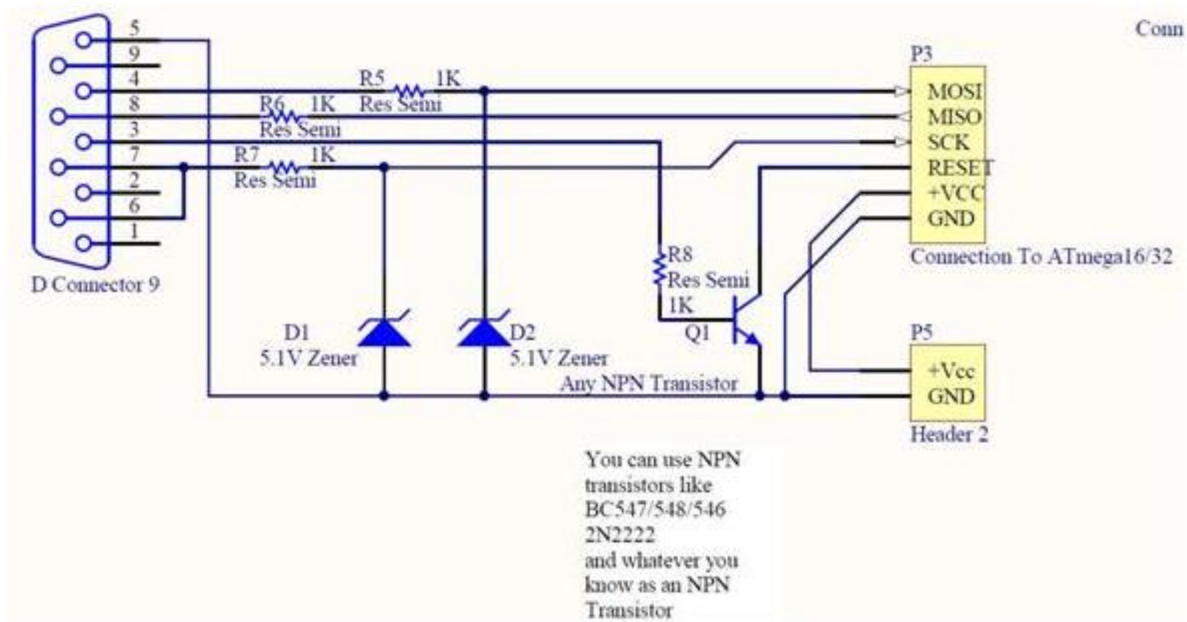
ATmega32 Memories

In ATmega32 microcontroller two main memory spaces i.e. the program memory and data memory space are used. In addition it uses an EEPROM memory for data storage.

In System Programmable Flash Program Memory:

ATmega32 microcontroller contains 32Kb of on-chip in system programmable flash memory for program storage. Flash memory is organized as 16K X 16K structure and its memory is divided into two sections application program section and boot program section.

Consider the ISP programmer circuit diagram is:



SRAM Data Memory:

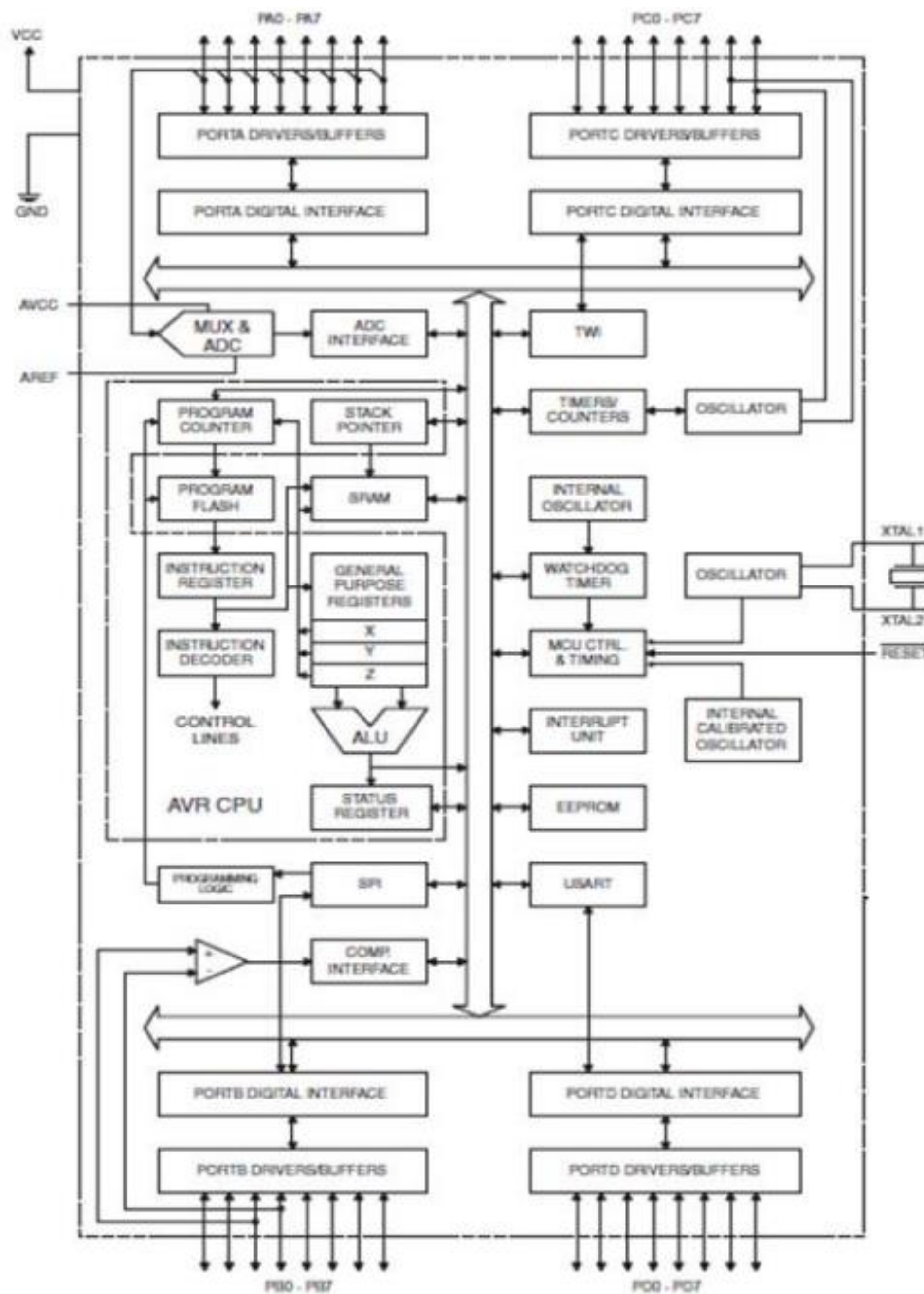
The Register file, the internal data SRAM and I/O memory are addressed by the lower 2144 data memory locations. The first 96 locations address the I/O memory and Register file, and the internal data Static RAM is addressed by the next 2048 locations.

Consider the five different addressing modes for the data memory is:-

- Direct addressing modes
- Indirect addressing modes
- Indirect with displacement addressing modes
- Indirect with pre-decrement addressing modes
- Indirect with post-decrement addressing modes

SRAM data memory have 32 general purpose registers, 2048 bytes of internal data SRAM, and 64 I/O registers are accessible by using the above addressing modes.

Consider the SRAM data memory structure shown in block diagram of ATmega32 is:



EEPROM Data Memory:

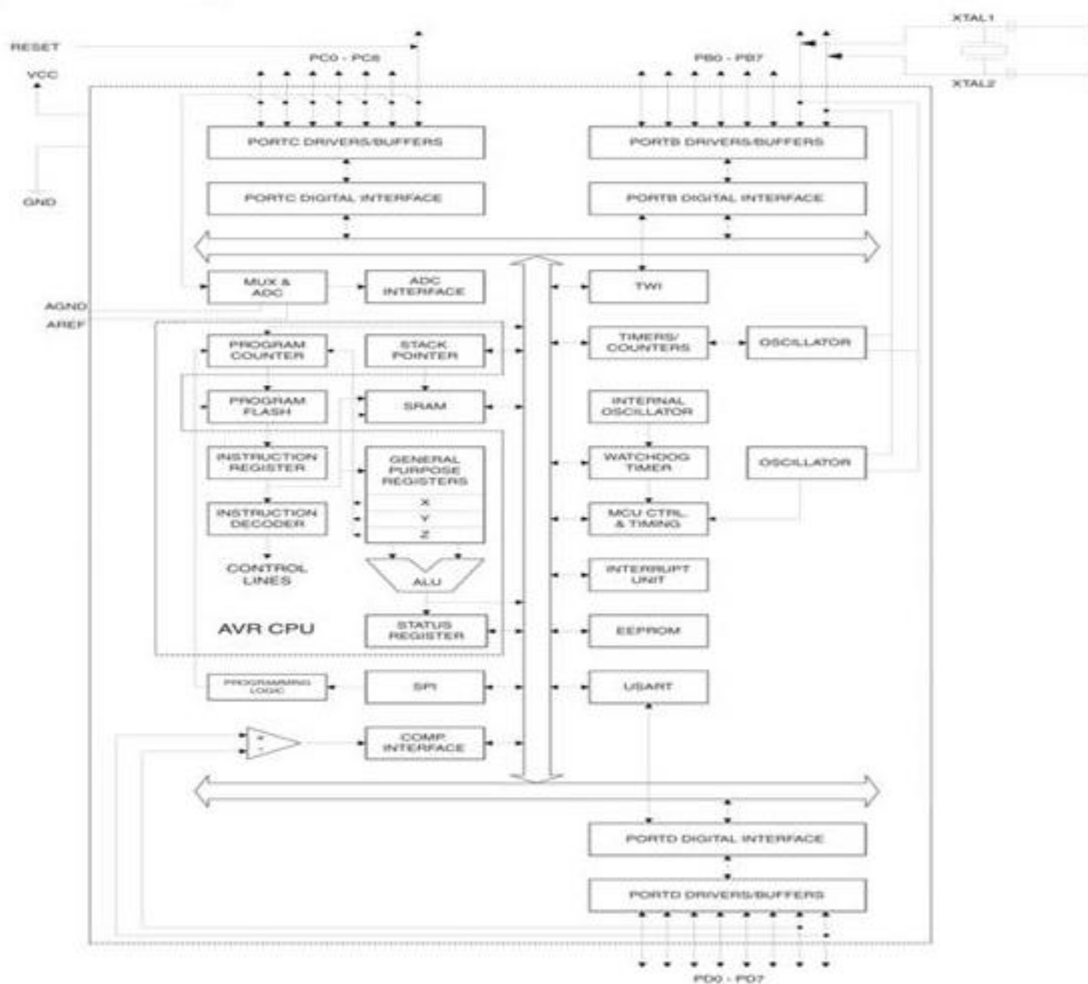
ATmega32 contains 1024 bytes of data EEPROM memory. It can be used as a separate data space in which single bytes can be read and written.

ATmega8 Microcontroller

It is an 8 bit CMOS technology based microcontroller belongs to the AVR family of microcontroller developed in 1996. It is built on RISC (Reduced Instruction Set Computer) architecture. Their main advantage is it doesn?t contain any accumulator register and the result of any operation can be stored inside any register, defined by an instruction.

ATmega8 Architecture:

Consider the block diagram representation of internal architecture configuration of ATmega8 microcontroller is:



Memory:

ATmega8 microcontroller consists of 1KB of SRAM, 8KB of flash memory and 512 bytes of EEPROM.

The 8KB flash memory is divided into two parts:-

- The upper part used as application flash section
- The lower part used as boot flash section

In ATmega8 microcontroller all the registers are connected directly with Arithmetic Logic Unit (ALU). The EEPROM memory is used for storing the user defined data.

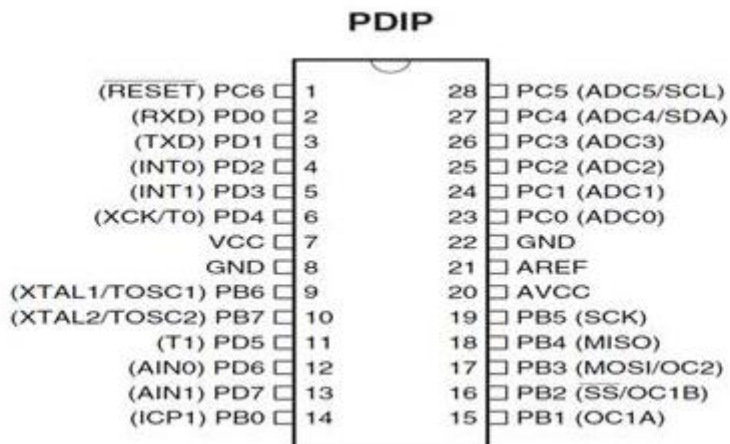
Input/output ports:

ATmega8 microcontroller consists of 3 I/O ports, named as B, C and D with a combination of 23 I/O lines. Port D consists of 8 I/O lines, Port C consists of 7 I/O lines, and Port B consists of 8 I/O lines.

Registers corresponding to the Input/output port X (B, C or D) are:

- **DDRX:** Data Direction Register of Port X
- **PORTX:** Data register of Port X
- **PINX:** Input register of Port X

ATmega8 Pin Diagram



One of the most important features of ATmega8 microcontroller is that except 5 pins, all other pins can be used for supporting two signals.

- Pins 9,10,14,15,16,17,18,19 are used for port B, Whereas Pins 23,24,25,26,27,28 and 1 are used for port C and Pins 2,3,4,5,6,11,12 are used for port D.
- Pin 1 is used as Reset pin and on applying low level signal for time longer than minimum pulse length will generate a reset signal.
- Pins 3 and 2 can also be used in serial communication for USART (Universal Synchronous and Asynchronous Receiver Transmitter).
- Pin 5 and 4 are used as external interrupts.
- Pins 10 and 9 are used as timer counter oscillators as well as external oscillator where the crystal is connected directly between the pins.
- Pin 19 is used as slave clock input or master clock output for Serial Peripheral Interface (SPI) channel.
- Pin 18 is used as slave clock output or master clock input
- Pins 23 to 28 are used for analog to digital conversion (ADC) channels.
- Pin 12 and 13 are used as Analog Comparator inputs.
- Pins 6 and 11 are used as counter/timer sources.

ATmega8 Microcontroller Sleep Modes:

The Microcontroller operates in 5 sleep modes as given below:

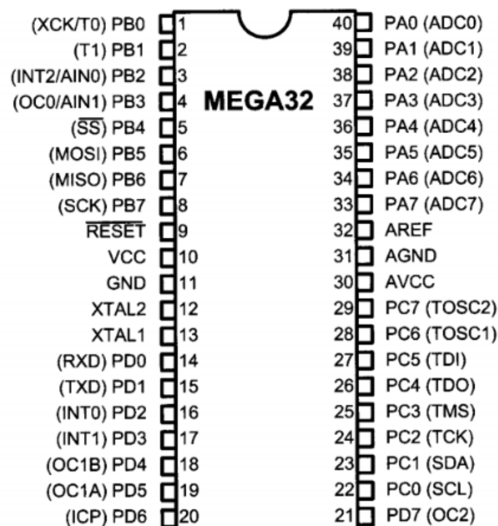
- **Power save Mode:** It is used when Counter/Timer is clocked asynchronously. In general this mode used for saving the operational power requirement of microcontroller.
- **Idle Mode:** It stops the functioning of CPU, but allows operation of ADC, TWI, SPI, and interrupts system and Watchdog. It is achieved by setting SM0 to SM2 bits of Microcontroller Unit register flag at zero.
- **Power down Mode:** It enables external interrupts, the 2-wire serial interface, and watchdog while disabling the external oscillator. It stops all generated clocks.

- **ADC Noise Reduction Mode:** It stops the central processing unit but allows the functioning of ADC, timer/counter and external interrupts.
- **Stand By mode:** In this mode, only oscillator is allowed to operate by slowing all other operation of microcontroller.

AVR I/O Port Programming

In AVR microcontroller family, there are many ports available for I/O operations, depending on which family microcontroller you choose. For the ATmega32 40-pin chip 32 Pins are available for I/O operation. The four ports PORTA, PORTB, PORTC, and PORTD are programmed for performing desired operation.

The Pin diagram of ATmega32 microcontroller is shown below:



The number of ports in AVR family varies depending on number of pins available on chip. The 8-pin AVR has port B only, while the 64-pin version has ports A to ports F, and the 100-pin AVR has ports A to ports L.

The table showing Numbers of ports in some AVR family members is shown below:

Pins	8-pin	28-pin	40-pin	64-pin	100-pin
Chip	**ATtiny25/45/85**	**ATmega8/48/88**	**ATmega32/16**	**ATmega64/128**	**ATmega1280**
Port A			X	X	X
Port B	6 bits	X	X	X	X
Port C		7 bits	X	X	X
Port D		X	X	X	X
Port E				X	X
Port F				X	X
Port G				5 bits	6 bits
Port H					X
Port J					X
Port K					X
Port L					X

Note: X indicates that the port is available.

The 40-pin AVR has four ports for using any of the ports as an input or output port, it must be accordingly programmed. In AVR microcontroller not all ports have 8 pins. For example:-in the ATmega8, Port C has 7 pins.

The Registers Addresses for ATmega32 Ports is given below:

Table 4-2: Register Addresses for ATmega32 Ports

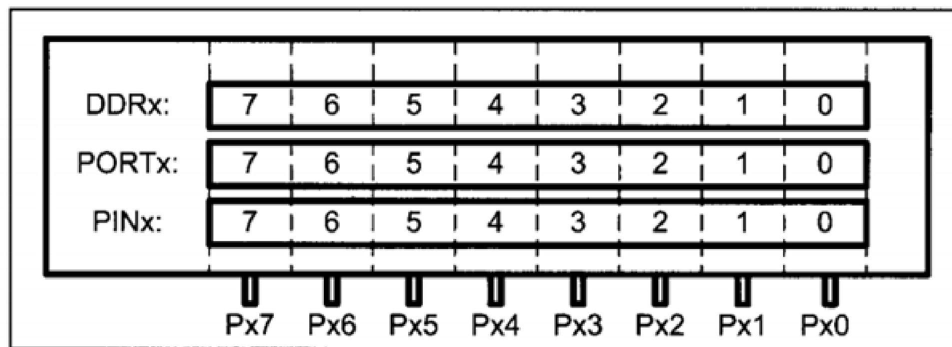
Port	Address	Usage
PORTA	$3B	output
DDRA	$3A	direction
PINA	$39	input
PORTB	$38	output
DDRB	$37	direction
PINB	$36	input
PORTC	$35	output
DDRC	$34	direction
PINC	$33	input
PORTD	$32	output
DDRD	$31	direction
PIND	$30	input

Each port in AVR microcontroller has three I/O registers associated with it. They are designated as PORTx, DDRx and PINx. For example: - in case of Port B we have PORTB, DDRB, and PINB. Here **DDR stands for Data Direction Registers, and PIN stands for Port Input pins.**

Each of I/O registers is 8 bits wide, and each port has a maximum of 8 pins, therefore each bit of I/O registers affects one of the pins.

For accessing I/O registers associated with the ports the common relationship between the registers and the pins of AVR microcontroller is used.

The relation between the Registers and the Pins of AVR is shown below:



AVR Registers

AVR is 8 bit microcontroller therefore all its ports are 8 bit wide. Every port has 3 registers associated with it each one have size of 8 bits. Every bit in those registers configures the pins of particular port. Bit0 of these registers are associated with Pin0 of the port, Bit1 of these registers are associated with Pin1 of the port, and same as for other bits.

The three registers available in AVR microcontroller are as follows:

- DDRx register
- PORTx register
- PINx register

DDRx register:

Data Direction Register configures the data direction of port pins. These registers are used for determining whether port pins will be used for input or output. On writing 1 to a bit in

DDRxmakes corresponding port pin as output, while writing 0 to a bit in DDRx makes corresponding port pin as input.

For example:

- For making all pins of port A as output pins:

1. DDRA= 0b11111111;

- For making all pins of port A as input pins:

1. DDRA= 0b00000000;

- For making lower nibble of port B as output and higher nibble as input:

1. DDRB=0b00001111;

PINx register:

PINx register used to read the data from port pins. In order to read the data from port pin, first we have to change the port?s data direction to input. This is done by setting bits in DDRx to zero. If port is made output, then reading PINx register will give a data that has been output on port pins.

There are two input modes. Either we can use port pins as internal pull up or as tri stated inputs. It will be explained as shown below:

For reading the data from port A,

1. DDRA = 0x00;   //Set port A as input
2. x = PINA;   //Read contents of port a

PORTx register:

In general PORTx register can be used for two purposes:

- **To output data:** when port is configured as output then PORTx register is used. When we set bits in DDRx to 1, corresponding pins becomes output pins. Now we can write the data into respective bits in PORTx register. This will immediately change the output state of pins according to data we have written on the ports.For example:

To output data in variable x on port A

1. DDRA = 0xFF; //make port A as outputs
2. PORTA = x; //output variable on port

To output 0xFF data on port B

1. DDRB = 0b11111111; //set all the pins of port B as outputs
2. PORTB = 0xFF; //write the data on port

To output data on only 0th bit of port C

1. DDRC.0 = 1; //set only 0th pin of port C as an output
2. PORTC.0 = 1; //make it high signal.

To activate/deactivate pull up resistors: when port is configured as input we set the bits in DDRx to 0, i.e. make port pins as inputs the corresponding bits in PORTx registers used to activate/deactivate pull-up registers associated with that pin. In order for activating pull-up resistor, set the bit in PORTx register to 1, and for deactivating (i.e. to make port as tri stated) set it to zero.

In input mode, when pull-up is enabled, default state of the pin is '1'. So if we don't add anything to pin and if we try to read it then it will read as 1.

Note: While using on chip Analog to Digital Converter (ADC), ADC port pins must be used as tri stated input.

For example:

To make lower nibble of port A as output, higher nibble as input with pull-ups enabled

1. DDRA = 0x0F; // higher nib> input, lower nib> output
2. PORTA = 0xF0; //lower nib> set output pins to 0

To make port B as tri stated input

1. DDRB = 0x00;   //use port B as input
2. PORTB = 0x00;   //Disable pull-ups register and make it tri state

To make port C as input with pull-ups enabled and read data from port a

1. DDRC = 0x00;   //make port C as input
2. PORTC = 0xFF;   //enable all pull-ups
3. y = PINC;   //read data from port C pins

External Interrupts in AVR Microcontroller

Microcontrollers can accept inputs from I/O ports, interrupts are used for accepting inputs generated by external events. Interrupt event directs the flow of program execution with a totally independent piece of code, known as "Interrupt Sub-Routine". There are many sources of interrupts that are available for a microcontroller. Most of them are generated by internal modules and are called as internal interrupts.

For running an interrupt subroutine following requirements are necessary:

- The interrupt source must be activated by setting the corresponding interrupt mask/ Interrupt Enable Bit.
- The enable bit in AVR status register must be set to 1. For this the instruction named 'sei' (Set Interrupt Enable).
- The interrupt sub routine must present. If there is no code to b e run, then an empty subroutine must occur at particular memory spaced to that interrupt.
- Finally the event must occur, so the execution of the routine gets triggered.

Writing an Interrupt Subroutine in AVR Studio:

It is tricky to use an interrupt subroutine into a C code of a microcontroller. Therefore the AVR GCC developers use a few symbols to represent the interrupts and macros that minimized the code size in many programs.

The interrupt subroutine for External Interrupt 0 and External Interrupt 1 is given below:

```
1.  // Interrupt sub routine code starts
2.  ISR(INT1_vect)
3.  {
4.  // Code for interrupt 1
5.  }
6.  ISR(INT0_vect)
7.  {
8.  // Code for interrupt 0
9.  }
10.// Interrupt Subroutine Code Ends
```

Registers Associated with External Interrupts:

The table showing microcontroller unit control register is given below:

MCU Control Register- MCUCR								
Bit	7	6	5	4	3	2	1	0
Bit Name	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00
Initial value	0	0	0	0	0	0	0	0
Read/Write	RW	RW	RW	RW	RW	RW	RW	RW

The table showing Interrupt Sense Control truth table is:

ISCx1	**ISCx0**	**Interrupt Generated Upon**
0	0	The lower Level of INTx pin
0	1	Any logical change inside INTx pin

1	0	Falling edge of INTx pin
1	1	Rising edge of INTx pin

USART in AVR ATmega16/ATmega32

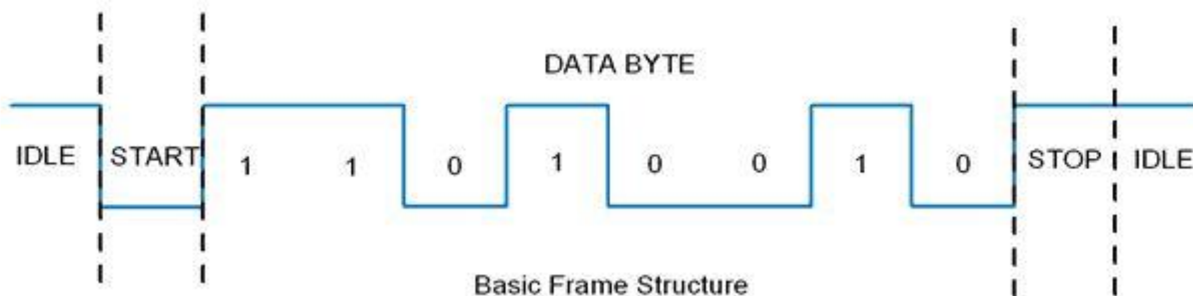
Introduction

AVR ATmega has flexible USART, which can be used for serial communication with other devices like computers, serial GSM, GPS modules, etc.

Before beginning with AVR USART, we will walk through the basics of serial communication.

Serial data framing

While sending/receiving data, some bits are added for the purpose of knowing the beginning/ending of data, etc. commonly used structure is: 8 data bits, 1 start bit (logic 0), and 1 stop bit (logic 1), as shown:



Basic Frame Structure

There are also other supported frame formats available in UART, like parity bit, variable data bits (5-9 data bits).

Speed (Baud rate)

As we know the bit rate is "Number of bits per second (bps)", also known as Baud rate in Binary system. Normally this defines how fast the serial line is. There are some standard baud

rates defined e.g. 1200, 2400, 4800, 19200, 115200 bps, etc. Normally 9600 bps is used where speed is not a critical issue.

Wires and Hardware connection

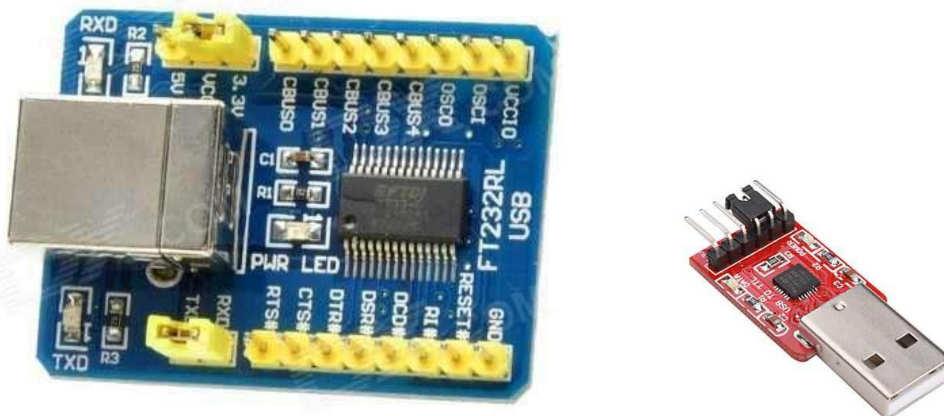
Normally in USART, we only need Tx (Transmit), Rx(Receive), and GND wires.

- AVR ATmega USART has a TTL voltage level which is 0 v for logic 0 and 5 v for logic 1.
- In computers and most of the old devices, RS232 protocol is used for serial communication, where normally 9 pin ‘D’ shape connector is used. RS232 serial communication has different voltage levels than ATmega serial communication i.e. +3 v to +25 v for logic zero and -3 v to -25 v for logic 1.
- So to communicate with RS232 protocol, we need to use a voltage level converter like MAX232 IC.

Although there are 9 pins in the DB9 connector, we don’t need to use all the pins. Only 2nd Tx(Transmit), 3rd Rx(Receive), and 5th GND pin need to be connected.

ATmega16/32 Serial Interface Connection Diagram

With a new PC and laptops, there is no RS232 protocol and DB9 connector. We have to use serial to the USB connector. There are various serial to USB connectors available e.g. CP2102, FT232RL, CH340, etc.



Serial to USB converter

To see serial communication, we can use a serial terminal like Realterm, Teraterm, etc. By selecting the serial port number (COM port in windows) and baud rate, we can open a serial port for communication.

As stated in the datasheet ATmega 16 USART has the following features

- Full Duplex Operation (Independent Serial Receive and Transmit Registers)
- Asynchronous or Synchronous Operation
- Master or Slave Clocked Synchronous Operation
- High-Resolution Baud Rate Generator
- Supports Serial Frames with 5, 6, 7, 8, or 9 Data Bits and 1 or 2 Stop Bits
- Odd or Even Parity Generation and Parity Check Supported by Hardware
- Data OverRun Detection
- Framing Error Detection
- Noise Filtering Includes False Start Bit Detection and Digital Low Pass Filter
- Three Separate Interrupts on TX Complete, TX Data Register Empty, and RX Complete
- Multi-processor Communication Mode
- Double Speed Asynchronous Communication Mode

Programming of USART in AVR

To program, first, we need to understand the basic registers used for USART

AVR basic Registers

1. **UDR: USART Data Register**

It has basically two registers, one is Tx. Byte and the other is Rx Byte. Both share the same UDR register. Do remember that, when we write to the UDR reg. Tx buffer will get written and when we read from this register, Rx Buffer will get read. Buffer uses the FIFO shift register to transmit the data.

2. **UCSRA**: USART Control and Status Register A. As the name suggests, is used for control and status flags. In a similar fashion, there are two more USART control and status registers, namely UCSRB and UCSRC.

3. **UBRR**: USART Baud Rate Register, this is a 16-bit register used for the setting baud rate.

We will see this register in detail:

UCSRA: USART Control and Status Register A

- **Bit 7 – RXC:** USART Receive Complete

  This flag bit is set when there is unread data in UDR. The RXC Flag can be used to generate a Receive Complete interrupt.

- **Bit 6 – TXC:** USART Transmit Complete

  This flag bit is set when the entire frame from Tx Buffer is shifted out and there is no new data currently present in the transmit buffer (UDR). The TXC Flag bit is automatically cleared when a transmit complete interrupt is executed, or it can be cleared by writing a one to its bit location. The TXC Flag can generate a Transmit Complete interrupt.

- **Bit 5 – UDRE:** USART Data Register Empty

  If UDRE is one, the buffer is empty which indicates the transmit buffer (UDR) is ready to receive new data. The UDRE Flag can generate a Data Register Empty Interrupt. UDRE is set after a reset to indicate that the transmitter is ready.

- **Bit 4 – FE:** Frame Error
- **Bit 3 – DOR:** Data OverRun

This bit is set if a Data OverRun condition is detected. A Data OverRun occurs when the receive buffer is full (two characters) and a new character is waiting in the receive Shift Register.

- **Bit 2 – PE:** Parity Error
- **Bit 1 – U2X:** Double the USART Transmission Speed
- **Bit 0 – MPCM:** Multi-processor Communication Mode

UCSRB: USART Control and Status Register B

7	6	5	4	3	2	1	0
RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8

Bit 7 – RXCIE: RX Complete Interrupt Enable

Writing one to this bit enables interrupt on the RXC Flag.

- **Bit 6 – TXCIE:** TX Complete Interrupt Enable

  Writing one to this bit enables interrupt on the TXC Flag.

- **Bit 5 – UDRIE:** USART Data Register Empty Interrupt Enable

  Writing one to this bit enables interrupt on the UDRE Flag.

- **Bit 4 – RXEN:** Receiver Enable

  Writing one to this bit enables the USART Receiver.

- **Bit 3 – TXEN:** Transmitter Enable

  Writing one to this bit enables the USART Transmitter.

- **Bit 2 – UCSZ2:** Character Size

The UCSZ2 bits combined with the UCSZ1:0 bit in UCSRC sets the number of data bits (Character Size) in a frame the receiver and transmitter use.

- **Bit 1 – RXB8:** Receive Data Bit 8
- **Bit 0 – TXB8:** Transmit Data Bit 8

UCSRC: USART Control and Status Register C

7	6	5	4	3	2	1	0
URSEL	UMSEL	UPM1	UPM0	USBS	USCZ1	USCZ0	UCPOL

Bit 7 – URSEL: Register Select

This bit selects between accessing the **UCSRC** or the **UBRRH** Register, as both register shares the same address. The URSEL must be one when writing the UCSRC or else data will be written in the UBRRH register.

- **Bit 6 – UMSEL:** USART Mode Select

  This bit selects between the Asynchronous and Synchronous mode of operation.

  0 = Asynchronous Operation

  1 = Synchronous Operation

- **Bit 5:4 – UPM1:0**: Parity Mode

These bits enable and set the type of parity generation and check. If parity a mismatch is detected, the PE Flag in UCSRA will be set.

UPM1	UPM0	Parity Mode
0	0	Disabled
0	1	Reserved
1	0	Enabled, Even Parity

UPM1	UPM0	Parity Mode
1	1	Enabled, Odd Parity

- **Bit 3 – USBS**: Stop Bit Select

  This bit selects the number of Stop Bits to be inserted by the Transmitter. The Receiver ignores this setting.

  0 = 1-bit

  1 = 2-bit

- **Bit 2:1 – UCSZ1:0**: Character Size

  The UCSZ1:0 bits combined with the UCSZ2 bit in UCSRB sets the number of data bits (Character Size) in a frame the Receiver and Transmitter use.

UCSZ2	UCSZ1	UCSZ0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	**1**	**1**	**8-bit**
1	0	0	Reserved
1	0	1	Reserved

UCSZ2	UCSZ1	UCSZ0	Character Size
1	1	0	Reserved
1	1	1	9-bit

- **Bit 0 – UCPOL:** Clock Polarity

  This bit is used for synchronous mode only. Write this bit to zero when the asynchronous mode is used.

  UBRRL and UBRRH: USART Baud Rate Registers

  Bit 15 – URSEL: Register Select

This bit selects between accessing the **UCSRC** or the **UBRRH** Register, as both register shares the same address. The URSEL must be one when writing the UCSRC or else data will be written in the UBRRH register.

- **Bit 11:0 – UBRR11:0:** USART Baud Rate Register.

  Used to define the baud rate

$$UBBR = \frac{Fosc}{16 * BaudRate} - 1 \qquad BaudRate = \frac{Fosc}{16 * (UBBR + 1)}$$

Example: suppose Fosc=8 MHz and required baud rate= 9600 bps.

Then the value of **UBRR= 51.088 i.e. 51.**

We can also set this value by c code using pre-processor macro as follow.

```
#define F_CPU 8000000UL                    /* Define frequency here its 8MHz */

#define USART_BAUDRATE 9600

#define BAUD_PRESCALE (((F_CPU / (USART_BAUDRATE * 16UL))) - 1)
```

BAUD_PRESCALE is the value that we have to load in the UBRR register to set the defined baud rate.

Timer in AVR ATmega16/ATmega32

Introduction

Generally, we use a timer/counter to generate time delays, waveforms, or to count events. Also, the timer is used for PWM generation, capturing events, etc.

In AVR ATmega16 / ATmega32, there are three timers:

- **Timer0**: 8-bit timer
- **Timer1**: 16-bit timer
- **Timer2**: 8-bit timer

Basic registers and flags of the Timers

TCNTn: Timer / Counter Register

Every timer has a timer/counter register. It is zero upon reset. We can access value or write a value to this register. It counts up with each clock pulse.

TOVn: Timer Overflow Flag

Each timer has a Timer Overflow flag. When the timer overflows, this flag will get set.

TCCRn: Timer Counter Control Register

This register is used for setting the modes of timer/counter.

OCRn: Output Compare Register

The value in this register is compared with the content of the TCNTn register. When they are equal, the OCFn flag will get set.

Let us see Timer0 to understand the timers in ATmega16 / ATmega32

Timer0

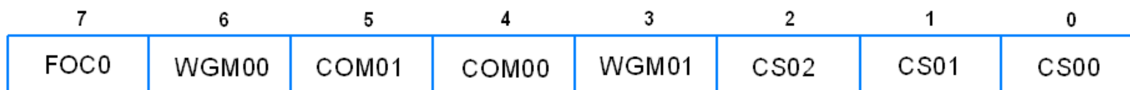
First, we need to understand the basic registers of the Timer0

1. **TCNT0: Timer / Counter Register 0**

   It is an 8-bit register. It counts up with each pulse.

2. **TCCR0: Timer / Counter Control register 0**

   This is an 8-bit register used for the operation mode and the clock source selection.



Bit 7- FOC0: Force compare match

Write only a bit, which can be used while generating a wave. Writing 1 to this bit causes the wave generator to act as if a compare match has occurred.

Bit 6, 3 - WGM00, WGM01: Waveform Generation Mode

WGM00	WGM01	Timer0 mode selection bit
0	**0**	Normal
0	**1**	CTC (Clear timer on Compare Match)
1	**0**	PWM, Phase correct
1	**1**	Fast PWM

Bit 5:4 - COM01:00: Compare Output Mode

These bits control the waveform generator. We will see this in the compare mode of the timer.

Bit 2:0 - CS02:CS00: Clock Source Select

These bits are used to select a clock source. When CS02: CS00 = 000, then timer is stopped. As it gets a value between 001 to 101, it gets a clock source and starts as the timer.

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer / Counter stopped)
0	0	1	clk (no pre-scaling)
0	1	0	clk / 8
0	1	1	clk / 64
1	0	0	clk / 256
1	0	1	clk / 1024
1	1	0	External clock source on T0 pin. Clock on falling edge
1	1	1	External clock source on T0 pin. Clock on rising edge.

3. **TIFR: Timer Counter Interrupt Flag register**

7	6	5	4	3	2	1	0
OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0

Bit 0 - TOV0: Timer0 Overflow flag

0 = Timer0 did not overflow

1 = Timer0 has overflown (going from 0xFF to 0x00)

Bit 1 - OCF0: Timer0 Output Compare flag

0 = Compare match did not occur

1 = Compare match occurred

Bit 2 - TOV1: Timer1 Overflow flag

Bit 3 - OCF1B: Timer1 Output Compare B match flag

Bit 4 - OCF1A: Timer1 Output Compare A match flag

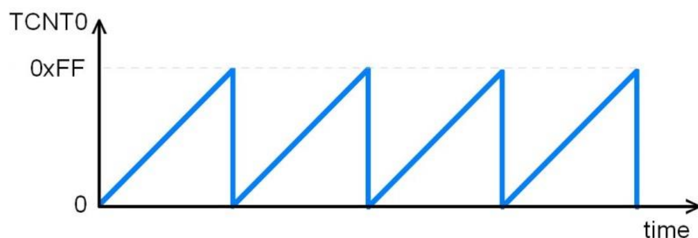
Bit 5 - ICF1: Input Capture flag

Bit 6 - TOV2: Timer2 Overflow flag

Bit 7 - OCF2: Timer2 Output Compare match flag

Timer0 Overflow

Normal mode: When the counter overflows i.e. goes from 0xFF to 0x00, the TOV0 flag is set.



Creating Delay Using Timer0

Steps to Program Delay using Timer0

1. Load the TCNT0 register with the initial value (let’s take 0x25).

2. For normal mode and the pre-scaler option of the clock, set the value in the TCCR0 register. As soon as the clock Prescaler value gets selected, the timer/counter starts to count, and each clock tick causes the value of the timer/counter to increment by 1.

3. Timer keeps counting up, so keep monitoring for timer overflow i.e. TOV0 (Timer0 Overflow) flag to see if it is raised.

4. Stop the timer by putting 0 in the TCCR0 i.e. the clock source will get disconnected and the timer/counter will get stopped.

5. Clear the TOV0 flag. Note that we have to write 1 to the TOV0 bit to clear the flag.

6. Return to the main function.

The time delay generated by above code

As Fosc = 8 MHz

T = 1 / Fosc = 0.125 µs

Therefore, the count increments by every **0.125 µs**.

In above code, the number of cycles required to roll over are:

0xFF - 0x25= 0xDA i.e. decimal 218

Add one more cycle as it takes to roll over and raise TOV0 flag: 219

Total Delay = 219 x 0.125 µs = **27.375 µs**

Example

Let us generate a square waveform having 10 ms high and 10 ms low time:

First, we have to create a delay of 10 ms using timer0.

*Fosc = 8 MHz

Use the pre-scalar 1024, so the timer clock source frequency will be,

8 MHz / 1024 = **7812.5 Hz**

Time of 1 cycle = 1 / 7812.5 = **128 µs**

Therefore, for a delay of 10 ms, number of cycles required will be,

10 ms / 128 µs = **78 (approx)**

We need 78 timer cycles to generate a delay of 10 ms. Put the value in TCNT0 accordingly.

Value to load in TCNT0 = 256 – 78 (78 clock ticks to overflow the timer)

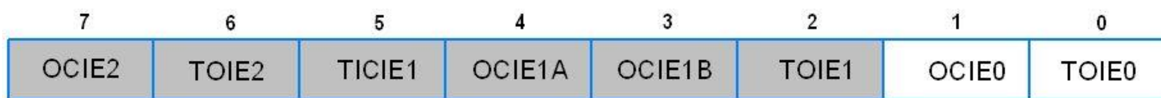
= **178 i.e. 0xB2 in hex**

Thus, if we load 0xB2 in the TCNT0 register, the timer will overflow after 78 cycles i.e. precisely after a delay of 10 ms.

*Note - All calculations are done by considering 8 MHz CPU frequency. If you are using another value of CPU frequency, modify the calculations accordingly; otherwise, the delay will mismatch.

Timer Interrupt

TIMSK: Timer / Counter Interrupt Mask Register

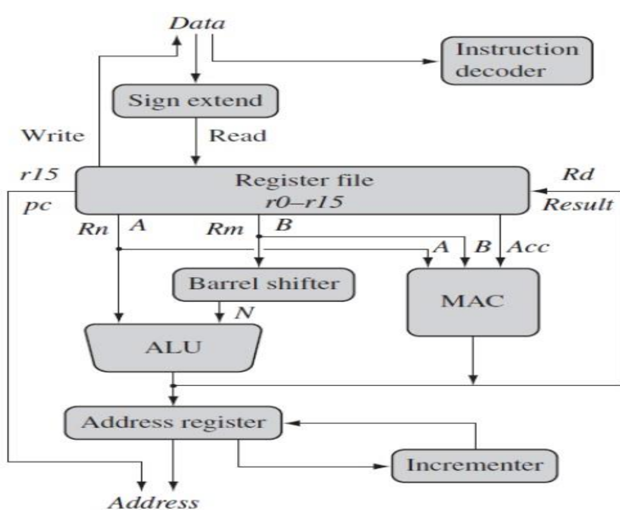


We have to set the TOIE0 (Timer0 Overflow Interrupt Enable) bit in the TIMSK register to set the timer0 interrupt so that as soon as the Timer0 overflows, the controller jumps to the Timer0 interrupt routine.

Unit 4-ARM PROCESSOR

- ARM is Advanced RISC Machine, earlier known as Acorn RISC Machine.
- It was built by Acorn Computers along with VLSI technology in 1990/11.
- ARM is a 32-bit microprocessor with RISC (Reduced Instruction Set Computer) architecture.
- RISC processors are designed to perform a smaller number of computer instructions so that they can operate at a higher speed, performing more millions of instructions per second (MIPS) as compared to CISC (Complex instruction set Computer) processors.
- ARM is one of the most licensed and thus widespread processor cores in the world.
- Used especially in portable devices due to low power consumption and reasonable performance (MIPS/watt).
- ARM does not manufacture its own VLSI devices. It licenses out its core to many companies such as TI, Philips, Intel etc.
- Because of their reduced instruction set, they require fewer transistors, which enables a smaller die size for the integrated circuitry (IC).
- The ARM processor's smaller size, reduced complexity and lower power consumption makes them suitable for increasingly miniaturized devices.
- ARM7 and older versions support Von Neumann Architecture.
- ARM9 and newer versions support Harvard Architecture.
- In Von Neumann implementation data items and instructions share same bus.
- In Harvard implementation two different buses -ARM high bus (AHB) and ARM Peripheral Bus (APB), are used for data and instructions.

ARM DATA FLOW MODEL



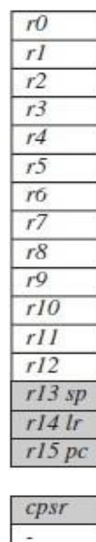
- Functional units are connected by Data Buses. The arrow represents the flow of data and lines represent the buses.
- The boxes represent the either an operation unit or storage area. The figure also shows abstract components that make ARM processor.

- The instruction decoder translates instructions. Data items are placed in the register file.
- A storage bank made up of 32-bit registers.
- Most instructions treat the registers as holding signed or unsigned 32-bit values.
- The sign extend hardware converts signed 8-bit and 16-bit numbers into 32-bit values.
- ARM instructions typically have two source registers, *Rn*and*Rm*, and a single result or destination register, *Rd,* Source operands are read from the register file using the internal bus.

- The ARM Processor like all processor uses a **Load-store architecture**
- **Load** instruction: copy data from memory to register
- **Store** instruction: copy data from register to memory
- There are no data processing instructions that directly manipulate data in memory.
- **ALU and MAC** (Multiply-accumulate) unit takes the register values Rn and Rm from A and B buses and computes a result.
- Load and store instructions use the **ALU** to generate an address to be held in the address register and broadcast on the Address bus.
- The register Rm can be alternatively pre-processed in the barrel shifter before it enters the **ALU**.
- For load and store instructions the incrementer updates the address register before the core reads or writes the next register value from or to the next sequential memory location.

Registers

- General purpose registers hold either data or an address.
- All registers are 32-bit in size.
- 18 active registers
  - 16 data registers – r0 to r15



2 process status registers

- **CPSR – Current Program Status Register**
- **SPSR – Saved Program Status Register**

- r13, r14, r15 have special functions
  - R13 – stack pointer (sp) and stores the head of the stack in the current processor mode
  - R14 – link register (lr) where the core puts the return address whenever it calls a subroutine.
  - R15 – programme counter (pc) and contains the address of the next instruction to be fetched by the processor.
- r13 and r14 can also be used as general purpose register as these registers are banked during a processor mode change.
- The registers r0 to r13 are orthogonal. Any instruction that you can apply to r0, you can equally apply to other registers.
- There are instructions that treat r14 and r15 in a special way.

Current Program Status Register (CPSR)

- ARM core uses CPSR to monitor and control internal operations.
- CPSR is a dedicated 32-bit register and resides in the register file.
- CPSR is divided into four fields each 8-bits wide:
  1. Flags – holding instruction conditions
  2. Status – reserved for future use
  3. Extension - reserved for future use
  4. Control – indicate processor mode, state and interrupt mask bits

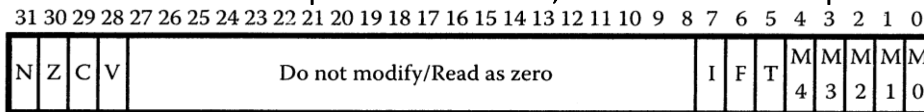
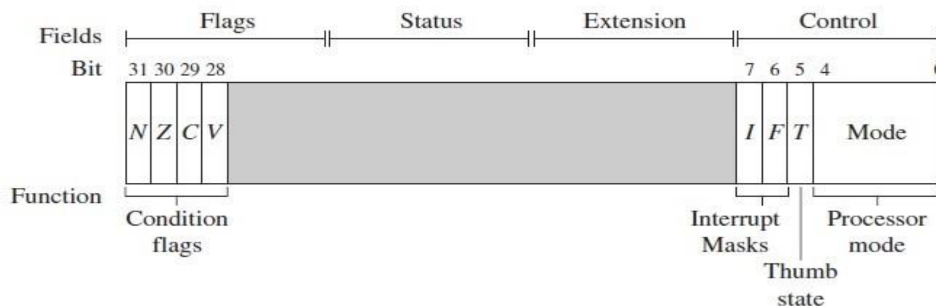


FIGURE 2.4 Format of the program status registers.

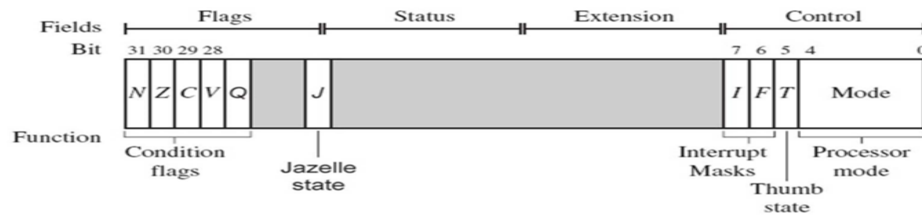


State & Instruction Sets

- The state of the core determines which instruction set is being executed.
- There are three instruction sets

- ARM: Active in ARM state
- Thumb: Active in Thumb state
- Jazelle: Active in Jazelle state

- The Jazelle (J) and Thumb (T) bits in the cpsr reflect the state of the processor.
- Jazelle instruction set is a closed set and is not openly available.
- When both J and T bits are 0, the processor is in ARM state and executes ARM instructions.



- ARM has introduced a set of extensions to the ARM architecture that will allow an ARM processor to directly execute Java byte code alongside exiting operating systems, middleware and application code.
- To execute Java bytecodes, the Jazelle technology is required and a modified version of the Java virtual machine.
  - the hardware portion of Jazelle only supports a subset of the Java bytecodes and rest are emulated in software .

State & Instruction Set Features

	ARM	Thumb	Jazelle
Instruction Size	32-bit	16-bit	8-bit
Core instructions	58	30	Over 60% of Java : H/W The rest : S/W
cpsr	T=0 J=0	T=1 J=0	T=0, J=1

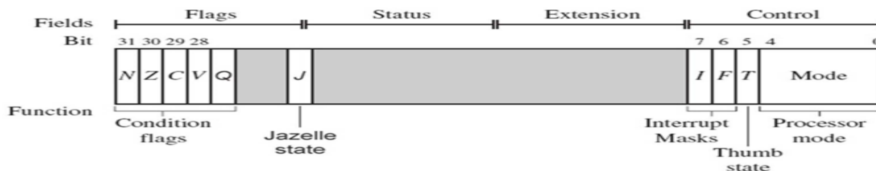
Interrupt Masks

- Interrupt masks are used to stop specific interrupt requests from interrupting the processor.
- Two types of interrupt request
  - Interrupt Request (IRQ)
  - Fast interrupt request (FIQ)
  - The I bit in the cpsr masks IRQ when set to binary 1

- The F bit in the cpsr masks FIQ when set to binary 1

Condition Flags

- Condition flags are updated by comparison and the result of ALU operations that specify the S instruction suffix
  - If a SUBS subtract instruction results in a register value of zero, then the Z flag in the cpsr is set
- Condition flags
  - N : Negative result from ALU
  - Z : Zero result from ALU
  - C : ALU operation Carried out
  - V : ALU operation overflowed
  - Q : Overflow & Saturation : In processors with DSP extensions, the Q bit indicates overflow or saturation that has occurred in an enhanced DSP instruction.



Processor Modes

- The processor mode determines which registers are active and the access rights to the CPSR register itself.
- Each processor mode is either privileged or non- privileged.
- A privileged mode allows full read-write access to CPSR.
- A non-privileged mode only allows read access to the control field in the CPSR but still allows R/W access to the condition flags.

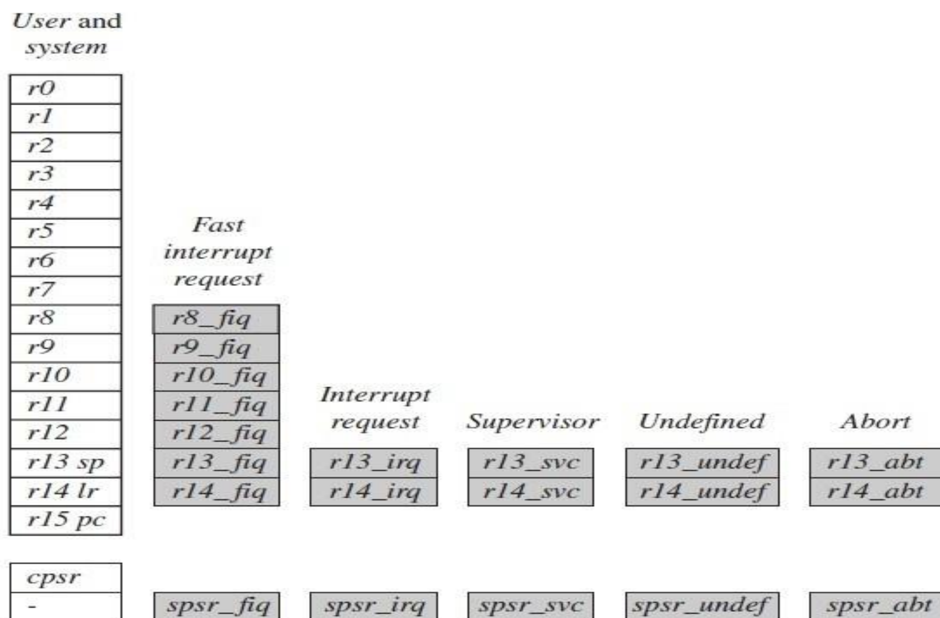
Mode	Description	
Supervisor (SVC)	Entered on reset and when a Software Interrupt (SWI) instruction is executed	Privileged modes
FIQ	Entered when a high priority (fast) interrupt is raised	
IRQ	Entered when a low priority (normal) interrupt is raised	
Abort	Used to handle memory access violations	
Undef	Used to handle undefined instructions	
System	Privileged mode using the same registers as User mode	
User	Mode under which most applications/OS tasks run	Unprivileged mode

Exception modes: Supervisor (SVC), FIQ, IRQ, Abort, Undef, System

FIGURE 2.1 Processor modes.

Banked Registers

- ARM has 37 registers in the register file.
- Of those, 20 registers are hidden from a program at different times. These registers are called banked registers (shown as shaded region).
- They are available only when the processor is in a particular mode.
- Every processor mode except user mode can change mode by writing directly to the mode bits of the CPSR.
- All modes except system mode have a set of associated banked registers that are a subset of the main 16 registers.
- A banked register maps one-to-one onto a user mode register.
- If we change processor mode, a banked register from the new mode will replace an existing register.
- There is no *spsr*available in the *user* mode.



Complete ARM register set.

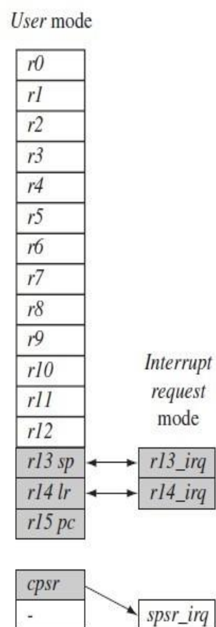
Mode					
User/System	Supervisor	Abort	Undefined	Interrupt	Fast Interrupt
R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7
R8	R8	R8	R8	R8	R8_FIQ
R9	R9	R9	R9	R9	R9_FIQ
R10	R10	R10	R10	R10	R10_FIQ
R11	R11	R11	R11	R11	R11_FIQ
R12	R12	R12	R12	R12	R12_FIQ
R13	R13_SVC	R13_ABORT	R13_UNDEF	R13_IRQ	R13_FIQ
R14	R14_SVC	R14_ABORT	R14_UNDEF	R14_IRQ	R14_FIQ
PC	PC	PC	PC	PC	PC
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR_SVC	SPSR_ABORT	SPSR_UNDEF	SPSR_IRQ	SPSR_FIQ

▒ = *banked register*

FIGURE 2.2 Register organization.

Changing ProcessorMode

- The processor mode can be changed
  - by a program that writes directly to CPSR (the processor has to be in privileged mode) or
  - by hardware when core responds to an exception or interrupt.
- Following exceptions and interrupts cause a mode change:
  - Reset
  - Interrupt Request
  - Fast interrupt request
  - Software interrupt
  - Data abort
  - Prefetch abort
  - Undefined instruction
- Exception and interrupt suspend the normal execution of sequential instructions and jump to a specific location.
- The processor mode changes form user mode to interrupt mode when an interrupt request occurs due to an external device raising an interrupt to the process core.
- This change causes user registers r13 and r14 to be banked.They are replaced with registers *r13_irq* and *r14_irq*respectively.
- *r13_irq*contains the stack pointer for the interrupt request mode.
- *r14_irq*contains the return address.
- The *cpsr*is copied to *spsr_irq*.
- To return back to the user mode, a special return instruction is used that instructs the core to restore the original cpsr from spsr_irq and bank in the user registers r13 and r14.
- The register spsr can only be modified in a privileged mode. There is no spsr available in the user mode.



Changing mode on an exception.

Note: cpsr is not copied into the spsr when a mode change isforced due to a program writing directly to the cpsr. The saving of cpsr only occurs when an exception or interrupt is raised.

Mode bits & Vector Table

Processor mode.

Mode	Abbreviation	Privileged	Mode[4:0]
Abort	abt	yes	10111
Fast interrupt request	fiq	yes	10001
Interrupt request	irq	yes	10010
Supervisor	svc	yes	10011
System	sys	yes	11111
Undefined	und	yes	11011
User	usr	no	10000

TABLE 2.2
Exception Vectors

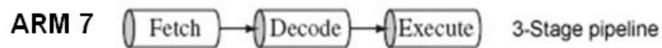
Exception Type	Mode	Vector Address
Reset	SVC	0x00000000
Undefined instructions	UNDEF	0x00000004
Software Interrupt (SWI)	SVC	0x00000008
Prefetch abort (instruction fetch memory abort)	ABORT	0x0000000C
Data abort (data access memory abort)	ABORT	0x00000010
IRQ (interrupt)	IRQ	0x00000018
FIQ (fast interrupt)	FIQ	0x0000001C

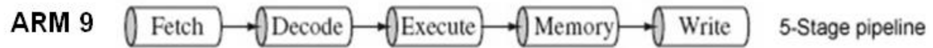
Exception Vector Table

- The exception vector table consists of designated addresses in externalmemory that holds information to handle an exception,an interrupt or atypical event (e.g. reset).
- For example, when an interrupt request comes along, the processor will change the program counter to 0x18 and begin fetching instructions from there.
- One can put a branch instruction at this location so that the processor can jump to the interrupt handler located at some other location in the memory.
- On reset, the core jumps to address 0x0 and starts fetching instruction. We either need to provide a reset exception handler (initialization routine) or we can begin coding at this address.

Pipeline Concept

- The mechanism a RISC processor uses to execute instructions in parallel to speed up execution.

ARM 7 

ARM 9 

ARM 10

- As the pipeline length increases, the amount of work done at each stage is reduced, which allows the processor attain a higher operating frequency
  - This in turn increases the performance
  - This also increases the latency

UNIT – 5

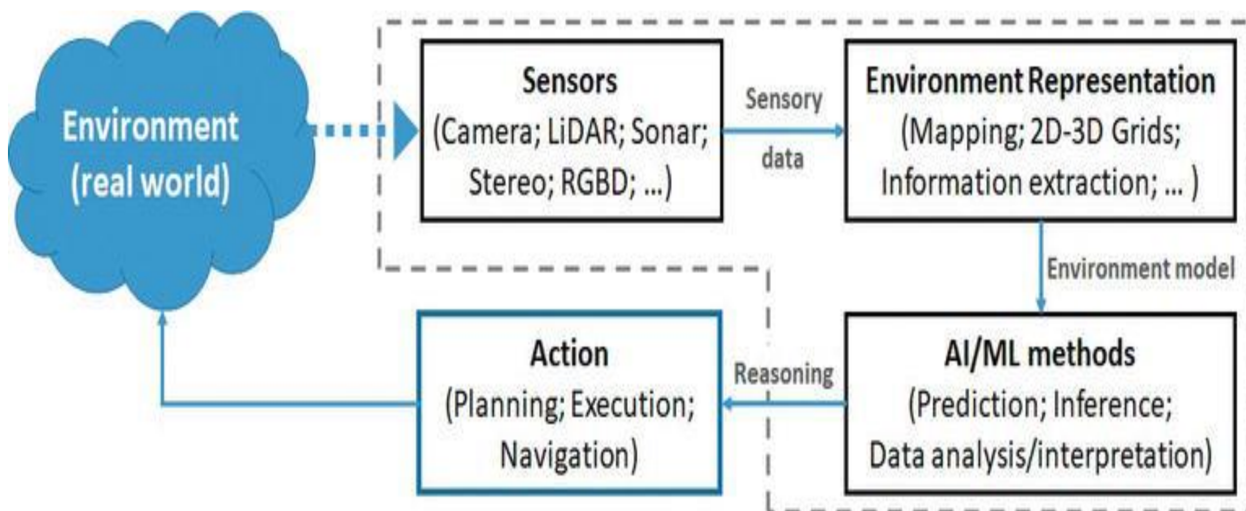
AI IN ROBOTICS

AI IN ROBOTICS: Robotic perception, localization, mapping- configuring space, planning uncertain movements, dynamics and control of movement, Ethics and risks of artificial intelligence in robotics.

- **Robotic perception**

Perception is the process by which robots map sensor measurements into internal representations of the environment. Perception is difficult because sensors are noisy, and the environment is partially observable, unpredictable, and often dynamic. In other words, robots have all the problems of state estimation.As a rule of thumb, good internal representations for robots have three properties:

- they contain enough information for the robot to make good decisions,
- they are structured so that they can be updated efficiently, and
- they are natural in the sense that internal variables correspond to natural state variables in the physical world



Active perception

The most well-known instance of active perception is active vision. The term “active vision” is essentially synonymous with moving cameras. Active vision work on Cog is oriented towards opening up the potentially rich area of manipulation-aided vision, which is still largely unexplored. But there is much to be gained by taking advantage of the fact that robots are actors in their environment, not simply passive observers.They have the opportunity to examine the world using causality, by performing probing actions and learning from the response. In conjunction with a developmental framework, this could allow the robot’s experience to expand outward from its sensors into its environment, from its own arm to the objects it encounters, and from those objects both back to the robot itself and outwards to other actors that encounter those same objects.

Developmental perception

The robot could reliably segment objects from the background (even if it is similar in appearance) by poking them. It can determine the shape of an object boundary in this special situation, even though it cannot do this normally. This is precisely the kind of situation that a developmental framework could exploit.Particular, familiar situations allow the robot to perceive something about objects and actors (such as a human or the robot itself) that could not be perceived outside those situations. These objects and actors can be tracked into other, less familiar situations,

which can then be characterized and used for further discovery. Throughout, existing perceptual capabilities ("primitive features") can be refined as opportunities arise.

Interpersonal perception

Perception is not a completely objective process; there are choices to be made. For example, whether two objects are judged to be the same depends on which of their many features are considered essential and which are considered incidental. For a robot to be useful, it should draw the same distinctions a human would for a given task. To achieve this, there must be mechanisms that allow the robot's perceptual judgments to be channeled and moulded by a caregiver. This is also useful in situations where the robot's own abilities are simply not up to the challenge, and need a helping hand.

Robotics and Artificial Intelligence

Robotics is a separate entity in Artificial Intelligence that helps study the creation of intelligent robots or machines. Robotics combines electrical engineering, mechanical engineering and computer science & engineering as they have mechanical construction, electrical component and programmed with programming language. Although, Robotics and Artificial Intelligence both have different objectives and applications, but most people treat robotics as a subset of Artificial Intelligence (AI). Robot machines look very similar to humans, and also, they can perform like humans, if enabled with AI.

What are Artificially Intelligent Robots?

Artificial intelligent robots connect AI with robotics. AI robots are controlled by AI programs and use different AI technologies, such as Machine learning, computer vision, RL learning, etc. Usually, most robots are not AI robots, these robots are programmed to perform repetitive series of movements, and they don't need any AI to perform their task. However, these robots are limited in functionality.

AI algorithms are necessary when you want to allow the robot to perform more complex tasks.

What are the advantages of integrating Artificial Intelligence into robotics?

- The major advantages of artificially intelligent robots are social care. They can guide people, especially come to aid for older people, with chatbot like social skills and advanced processors.
- Robotics also helps in Agricultural industry with the help of developing AI based robots. These robots reduce the farmer's workload.
- In Military industry, Military bots can spy through speech and vision detectors, along with saving lives by replacing infantry
- Robotics also employed in volcanoes, deep oceans, extremely cold places, or even in space where normally humans can't survive.
- Robotics is also used in medical and healthcare industry as it can also perform complex surgeries that have a higher risk of a mistake by humans, but with a pre-set of instructions and added Intelligence. AI integrated robotics could reduce the number of casualties greatly.

Difference in Robot System and AI Programs

Here is the difference between Artificial Intelligence and Robots:

1. AI Programs

Usually, we use to operate them in computer-simulated worlds.

Generally, input is given in the form of symbols and rules.

To operate this, we need general-purpose/Special-purpose computers.

2. Robots

Generally, we use robots to operate in the real physical world.

Inputs are given in the form of the analogue signal or in the form of the speech waveform.

Also, to operate this, special hardware with sensors and effectors are needed

Localization and mapping

Localization is the problem of finding out where things are—including the robot itself. Knowledge about where things are is at the core of any successful physical interaction with the environment. For example, robot manipulators must know the location of objects they seek to manipulate; navigating robots must know where they are to find their way around.

In some situations, no map of the environment is available. Then the robot will have to acquire a map. This is a bit of a chicken-and-egg problem: the navigating robot will have to determine its location relative to a map it doesn't quite know, at the same time building this map while it doesn't quite know its actual location. This problem is important for many robot applications, and it has been studied extensively under the name simultaneous localization and mapping, abbreviated as SLAM.

Simultaneous Localization and Mapping

SLAM is the estimation of the pose of a robot and the map of the environment simultaneously. SLAM is hard because a map is needed for localization and a good pose estimate is needed for mapping

- **Localization**: inferring location given a map.
- **Mapping**: inferring a map given locations.
- **SLAM**: learning a map and locating the robot simultaneously.

SLAM problem is hard because it is kind of a paradox i.e :

- In order to build a map, we need now the position.
- To determine our position, we need a map.

It is like a chicken-egg problem.

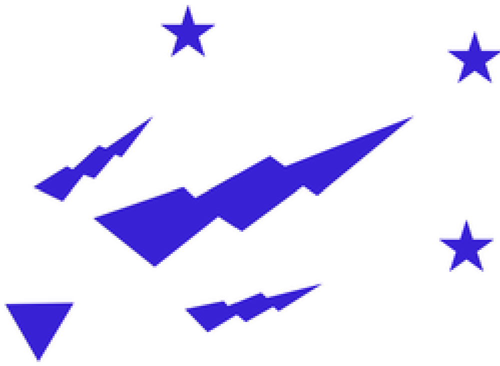
SLAM has multiple parts and each part can be executed in many different ways:

- Landmark detection
- Data association
- State Estimation
- State Update

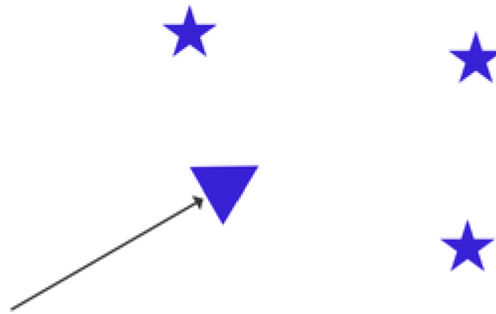
- Landmark Update

SLAM step by step

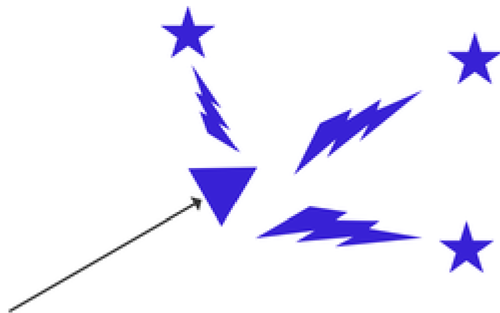
SLAM step 1



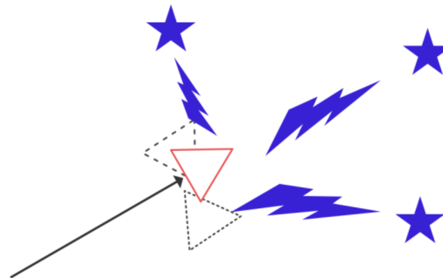
>SLAM step 2



SLAM step 3



SLAM step 4



- SLAM process consists of the following steps:

- In the first step, it uses the environment to update the position of the robot. We can use Odometry but it can be erroneous, we cannot only rely directly on odometry.
- We can use laser scans of the environment to correct the position of the robot. But, it won't work in some environments like underwater.
- Thus, the position of the robot can be better identified by extracting features from the environment.

Data Association

Data association or data matching is that of matching observed landmarks from different (laser) scans with each other. There are some challenges associated with the Data Association,

- The algorithm might not re-observe landmarks in every frame.
- The algorithm wrongly associates a landmark to a previously observed landmark.

There are few approaches to perform data association, we will be discussing the nearest neighbor algorithm first:

- First, when you get the data from the laser scan use landmark extraction to extract all visible landmarks.
- After that, we associate all the extracted landmarks to the closest landmark that can be observed >N times.
- Now, we input the list of extracted landmarks and list of previously detected landmarks that are in the database, if the landmark is already in the database then, we increase the their count by N, and if they are not present then set their count to 1.

After the above step, we need to perform the following update steps:

- **State Estimation:** In this step, we use the odometer data to get the current state estimate.
- **State update:** In this stage, we update our new estimated state by re-observing landmarks.
- **Landmarks update:** In this step, we add new landmarks that are detected in current stage.

Applications of SLAM

- SLAM problem is fundamental for getting robots autonomous. It has wide variety of application where we want to represent surroundings with a map such as Indoor, Underwater, Outer space etc.

Configuration Space

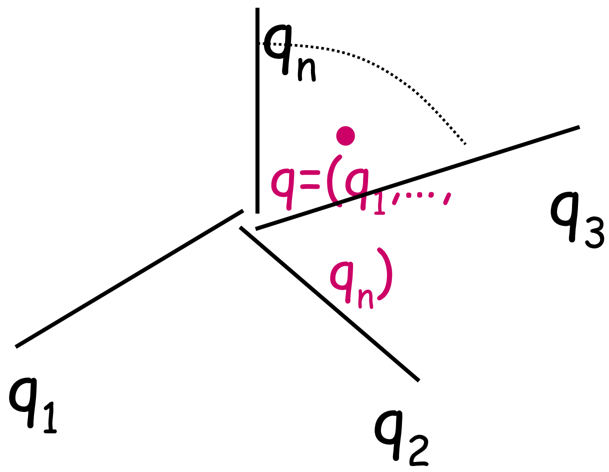
The configuration space is **a transformation from the physical space in which the robot is of finite-size into another space in which the robot is treated as a point**. In other words, the configuration space is obtained by shrinking the robot to a point, while growing the obstacles by the size of the robot.

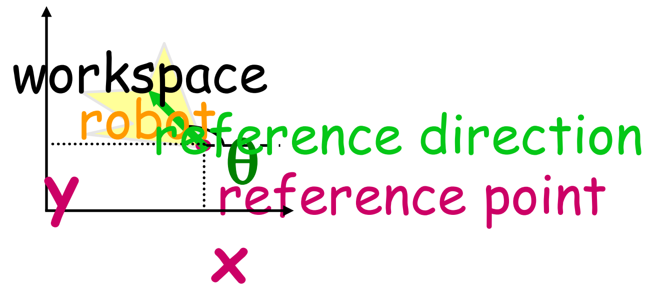
A key concept for motion planning is a configuration: – a complete specification of the position of every point in the system

• A simple example: a robot that translates but does not rotate in the plane: – what is a sufficient representation of its configuration?

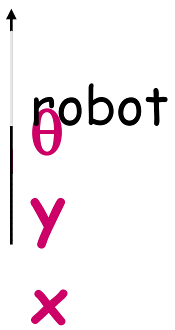
• The space of all configurations is the configuration space or Cspace.

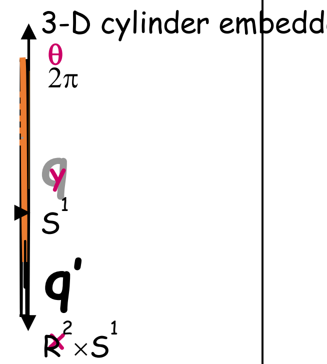
- A robot **configuration** is a specification of the positions of all robot points relative to a fixed coordinate system
- Usually a configuration is expressed as a "vector" of parameters

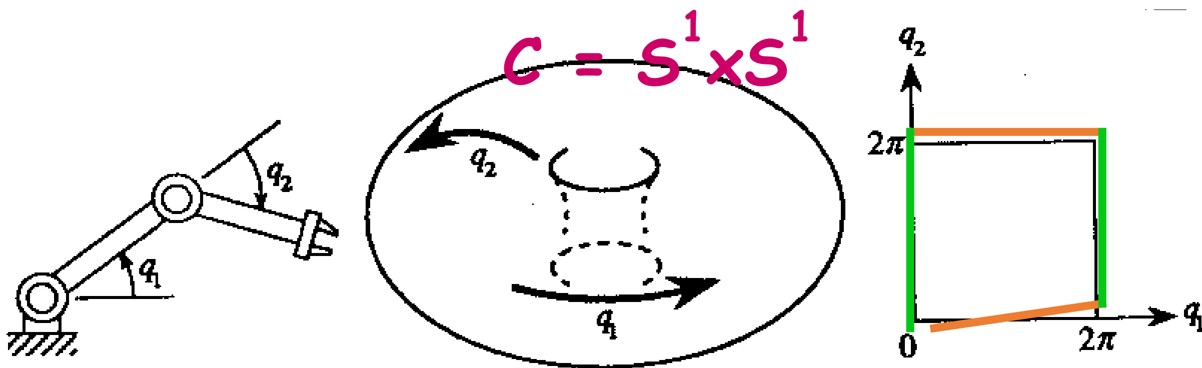


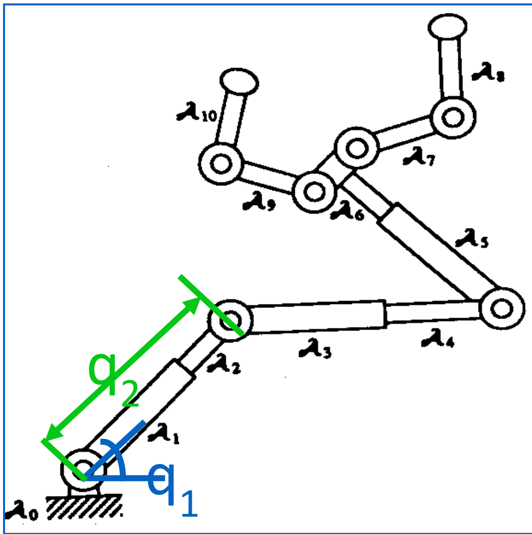


- Space of all its possible configurations
- But the topology of this space is usually not that of a Cartesian space









$(S1)^7 \times I^3$(I: Interval of reals)

Structure of Configuration Space

- It is a manifold, i.e., for each point q, there is a 1-to-1 map between a neighborhood of q and a Cartesian space $\mathbf{R}^n$, where *n* is the dimensionality of C
- This map is a local coordinate system called a chart.
- C can always be covered by a finite number of charts. Such a set is called an atlas

Metric in Configuration Space

A metric or distance function d in C is a map d:(q1,q2) ∈ C2 → d(q1,q2) > 0 such that:

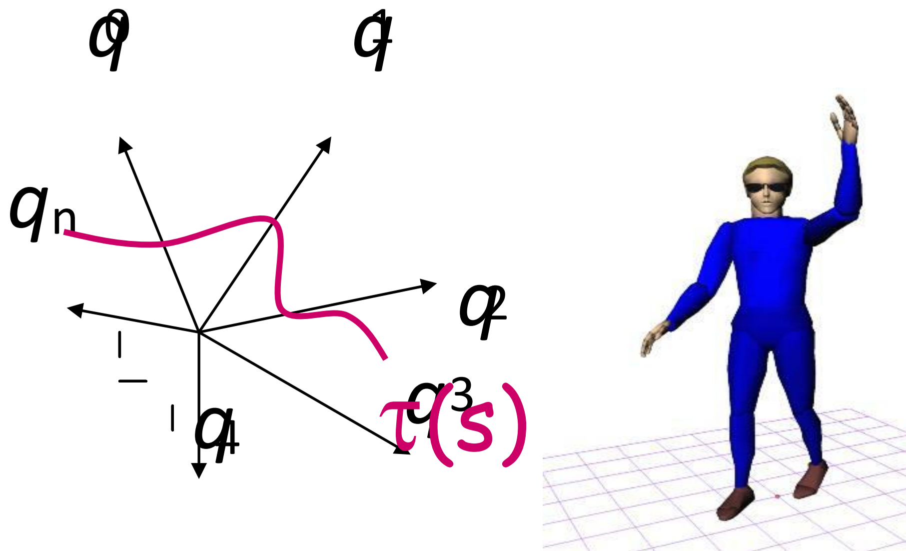
- $d(q_1,q_2) = 0$ if and only if $q_1 = q_2$
- $d(q_1,q_2) = d(q_2,q_1)$
- $d(q_1,q_2) \leq d(q_1,q_3) + d(q_3,q_2)$

Example:

- Robot **A** and point x of **A**
- x(q): location of x in the workspace when A is at configuration q
- A distance d in C is defined by: $d(q,q') = \max_{x \in A} ||x(q)-x(q')||$

  where ||a - b|| denotes the Euclidean distance between points a and b in the workspace

Notion of a Path



- A path in C is a piece of continuous curve connecting two configurations q and q’: $\mathbf{t} : s \in [0,1] \rightarrow \mathbf{t}(s) \in C$
- $s' \rightarrow s \;\Rightarrow\; d(\mathbf{t}(s),\mathbf{t}(s')) \rightarrow 0$

Cell decomposition methods

The first approach to path planning uses cell decomposition—that is, it decomposes thefree space into a finite number of contiguous regions, called cells. These regions have the important property that the path-planning problem within a single region can be solved by simple means (e.g., moving along a straight line).

Cell decomposition methods can be improved in a number of ways, to alleviate some of these problems. The first approach allows further subdivision of the mixed cells—perhaps using cells of half the original size. This can be continued recursively until a path is found that lies entirely within free cells. (Of course, the method only works if there is a way to decide if a given cell is a mixed cell, which is easy only if the configuration space boundaries have relatively simple mathematical descriptions.) This method is complete provided there is a bound on the smallest passageway through which a solution must pass. Although it focuses most of the computational effort on the tricky areas within the configuration space, it still fails to scale well to high-dimensional problems because each recursive splitting of a cell creates 2d smaller cells. A second way to obtain a complete algorithm is to insist on an exact cell decomposition of the free space.

Modified cost functions

Anyone who has driven a car knows that a parking space with one millimeter of clearance on either side is not really a parking space at all; for the same reason, we would prefer solution paths that are robust with respect to small motion errors.This problem can be solved by introducing a potential field. A potential field is a function defined over state space, whose value grows with the distance to the closest obstacle.

The potential field can be used as an additional cost term in the shortest-path calculation. This induces an interesting tradeoff. On the one hand, the robot seeks to minimize path length to the goal. On the other hand, it tries to stay away from obstacles by virtue of minimizing the potential function.

Skeletonization methods

The second major family of path-planning algorithms is based on the idea of skeletonization. These algorithms reduce the robot’s free space to a one-dimensional representation, for which the planning problem is easier. This lower-dimensional representation is called a skeleton of the configuration space.

planning uncertain movements

The basic motion planning problem is a relaxed version of the motion planning problem. The robot is a single, rigid body that can move freely and has no dynamics. It acts in a static, known environment. Due to these assumptions the problem is simplified, limiting the practical implementations of the solutions to the problem. Therefore, to meet with the conditions of the actual problem three extensions of the basic motion planning problem are regarded:

- planning in a dynamic environment
- planning with uncertainty
- planning with constraints

The way to deal with extensions that encompass the actual problem is called a planning approach. A planning approach views the motion planner as a whole. So at this point the representation method and the search algorithm come together.

Planning in a Dynamic Environment

In the basic motion planning problem the environment is considered to be completely static as the robot A is the only moving object in the environment. The environment can also be dynamic, when it contains moving objects. Another type of environment occurs when not only the motion of the robot A, but of multiple robots Ai is to be planned (Erdmann and Lozano-Pérez, 1986; Latombe, 1990). This case differs from an environment with moving objects, as now the motion of more than one robots is under control.

Finally, a special case arises when manipulation (Choset, 2005; Li et al., 1989) is considered. In this case, the ability to alter the environment during movement, by moving objects itself, must be taken into account by the motion planner. Planning for manipulation is such a broad topic in itself that it is also been addressed with techniques that are outside the scope of motion planning. This study will therefore not go into depth on this subject.

Planning with Uncertainty

The basic motion planning problem is based on assumptions about the robot and obstacles in the workspace. It assumes exact knowledge of the workspace and the obstacles' location and geometry. Furthermore, it is assumed that the planned path is executed exactly. Such assumptions are generally not realistic and therefore uncertainty must be considered in: a priori knowledge on the workspace; in sensor information that is acquired during the execution of planning; and in the execution of the plan itself.

- **Robust methods**

Uncertainty can also be handled using so-called robust control methods (see page 836) rather than probabilistic methods. A robust method is one that assumes a bounded amount of uncertainty in each aspect of a problem, but does not assign probabilities to values within the allowed interval. A robust solution is one that works no matter what actual values occur, provided they are within the assumed interval.

A fine-motion plan consists of a series of guarded motions. Each guarded motion consists of (1) a motion command and (2) a termination condition, which is a predicate on the robot's sensor values, and returns true to indicate the end of the guarded move.The motion commands are typically compliant motions that allow the effector to slide if the motion command would cause collision with an obstacle.

Dynamics and control

the notion of dynamic state, which extends the kinematic state of a robot by its velocity. For example, in addition to the angle of a robot joint, the dynamic state also captures the rate of change of the angle, and possibly even its momentary acceleration. The transition model for a dynamic state representation includes the effect of forces on this rate of change. Such models are typically expressed via differential equations, which are equations that relate a quantity (e.g., a kinematic state) to the change of the quantity over time (e.g., velocity). In principle, we could have chosen to plan robot motion using dynamic models, instead of our kinematic models. Such a methodology would lead to superior robot performance, if we could generate the plans. However, the dynamic state has higher dimension than the kinematic space, and the curse of dimensionality would render many motion planning algorithms inapplicable for all but the most simple robots. For this reason, practical robot system often rely on simpler kinematic path planners

A common technique to compensate for the limitations of kinematic plans is to use a separate mechanism, a controller, for keeping the robot on track. Controllers are techniques for generating robot controls in real time using feedback from the environment, so as to achieve a control objective. If the objective is to keep the robot on a preplanned path, it is often referred to as a reference controller and the path is called a reference path. Controllers that optimize a global cost function are known as optimal controllers. Optimal policies for continuous MDPs are, in effect, optimal controllers. Controllers that provide force in negative proportion to the observed error are known as P controllers. The letter 'P' stands for proportional, indicating that the actual control is proportional to the error of the robot manipulator. More formally, let $y(t)$ be the reference path, parameterized by time index t. The control a_t generated by a P controller has the form:

$$a_t = K_P (y(t) - x_t) .$$

Here x_t is the state of the robot at time t and K_P is a constant known as the gain parameter of the controller and its value is called the gain factor); K_p regulates how strongly the controller corrects for deviations between the actual state x_t and the desired one $y(t)$. In our example, $K_P = 1$. At first glance, one might think that choosing a smaller value for K_P would remedy the problem.

Our P controller appears to be stable but not strictly stable, since it fails to stay anywhere near its reference trajectory. The simplest controller that achieves strict stability in our domain is a PD controller. The letter 'P' stands again for proportional, and 'D' stands for derivative. PD controllers are described by the following equation:

$$a_t = K_P (y(t) - x_t) + K_D \frac{\partial(y(t) - x_t)}{\partial t} .$$

As this equation suggests, PD controllers extend P controllers by a differential component, which adds to the value of a_t a term that is proportional to the first derivative of the error $y(t) - x_t$ over time. What is the effect of such a term? In general, a derivative term dampens the system that is being controlled. To see this, consider a situation where the error $(y(t)-x_t)$ is changing rapidly over time, as is the case for our P controller above. The derivative of this error will then counteract the proportional term, which will reduce the overall response to the perturbation. However, if the same error persists and does not change, the derivative will vanish and the proportional term dominates the choice of control.