ROHINI COLLEGE OF ENGINEERING AND TECHNOLOGY

Department of Electronics and Communication Engineering

VALUE ADDED COURSE ON EMBEDDED SYSTEMS AND INTERNET OF THINGS (IoT)

	EMBEDDED CONTROL SYSTEMS AND INTERNET OF THINGS (IoT)	L	T	P	C
		3	0	0	3

COURSE OBJECTIVES:

- To expose the students about the fundamentals of Embedded System.
- To educate about Firmware design and development.
- To discuss on aspects required in embedded system design techniques.
- To understand the fundamentals of Internet of Things.
- To build a small low cost embedded system using Arduino / Raspberry Pi or equivalent boards and to apply the concept of Internet of Things in the real world scenario.

Embedded System Vs General Computing System - Classification of Embedded System, Purpose of Embedded system, Quality Attributes of Embedded System - Typical Embedded System- Core of Embedded System, Memory, Sensors and Actuators, Communication Interface- Onboard communication interface, External communication interface.

Embedded Firmware Design Approaches- Embedded Firmware Development Languages - Embedded System Development Environment - IDE, Compiler, Linker - Types of File Generated on Cross Compilation-Simulator, Emulator and Debugging- Fundamental issues in Hardware Software Co-design- Integration and Testing of Embedded Hardware and Firmware.

Introduction-Characteristics - Physical design - protocols – Logical design – Enabling technologies – IoT Levels – Domain Specific IoTs – IoT vs. M2M. IoT systems management – IoT Design Methodology – Specifications Integration and Application Development.

Physical device – Raspberry Pi Interfaces – Programming – APIs / Packages – Web services. Intel Galileo Gen2 with Arduino- Interfaces - Arduino IDE – Programming - APIs and Hacks. Various Real time applications of IoT- Connecting IoT to cloud – Cloud Storage for IoT – Data Analytics for IoT – Software & Management Tools for IoT.

IoE – Overview – Architecture-Smart objects and LLNs-Secure mobility. Home automation – Cities: Smart parking – Environment: Weather monitoring – Agriculture: Smart irrigation – Data analytics for IoT – Software & management tools for IoT cloud storage models & Communication APIs – Cloud for IoT – Amazon Web Services for IoT.

UNIT-1

EMBEDDED SYSTEM VS GENERAL COMPUTER SYSTEMS

- The computing revolution began with the general purpose computing requirements. Later it was realized that the general computing requirements are not sufficient for the embedded computing requirements. The embedded computing requirements demand 'something special' if the terms of response to stimuli, meeting the computational deadlines, power efficiency, limited memory availability, etc. Let's take the case of your personal computer, which may be either a desktop PC or a laptop PC or a palmtop PC.
- It is built around a general purpose processor like an Intel@ Centrino or a Duo/Quad core or an AMD Turion^{TM} processor and is designed to support a set of multiple peripherals like multiple USB 2.0 ports, Wi-Fi, Ethernet, video port, IEEEI 394, SD/CF/MMC external interfaces, Bluetooth, etc and with additional interfaces like a CD read/writer, on-board Hard Disk Drive (HDD), gigabytes of RAM, etc.
- You can load any supported operating system (like Windows@ XP (Vista/7, or Red Hat Linux/ Ubuntu Linux, UNIX etc) into the hard disk of your PC. You can write or purchase a multitude of applications for your PC and can use your PC for running a large number of applications (like printing your dear's photo using a printer device connected to the PC and printer software, creating a document using Microsoft @ Office Word tool, etc.) Now let us think about the DVD player you use for playing DVD movies.
- Is it possible for you to change the operating system of your DVD? Is it possible for you to write an application and download it to your DVD player for executing? Is it possible for you to add printer software to your DVD player and connect a printer to your DVD player to take a printout? Is it possible for you to change the functioning of your DVD player to a television by changing the embedded software? The answers to all these questions are 'NO'. Can you see any general purpose interface like Bluetooth or Wi-Fi on your DVD player? Of course 'NO'. The only interface you can find out on the DVD player is the interface for connecting the DVD player with the display screen and one for controlling the DVD player through a remote (May be an IR or any other specific wireless interface).
- Indeed your DVD player is an embedded system designed specifically for decoding digital video and generating a video signal as output to your TV or any other display screen which supports the display inter- face supported by the DVD Player. Let us summaries our findings from the comparison of embedded system and general purpose computing system with the help of a table:

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 the operating system for functioning

CLASSIFICATION OF EMBEDDED SYSTEM

- It is possible to have a multitude of classifications for embedded systems, based Go different criteria. Some of the criteria used in the classification of embedded systems are:
    1. Based on generation
    2. Complexity and performance requirements
    3. Based on deterministic behavior
    4. Based on triggering.
- The classification based on deterministic system behavior is applicable for 'Real Time' systems. The application/task execution behavior for an embedded system can be deterministic or non- deterministic. Based on the execution behavior, Real Time embedded systems are classified into Hard and Soft. We will discuss about hard and soft real time systems in a later chapter. Embedded Systems which are 'Reactive' in nature (Like process control systems in industrial control applications) can be classified based on the trigger. Reactive systems can be either event triggered or time triggered.

Classification Based on Generation

- This classification is based on the Girder in which the embedded processing systems evolved from the First version to where they are today. MAs per this criterion, embedded systems can be classified into:
- **First Generation:** early embedded systems were built around 8bit microprocessors like 8085 and Z80, and 4bit microcontrollers. Simple in hardware circuits with firmware developed in Assembly code. Digital telephone keypads, stepper motor control units etc. are examples of this.

- **Second Generation:** These are embedded systems built around 16bit microprocessors and 8 or 16 bit microcontrollers, following the first generation embedded systems. The instruction set for the second generation processors/controllers were much more complex and powerful than the first generation processors/controllers. Some of the second generation embedded systems contained embedded operating systems for their operation. Data Acquisition Systems, SCADA systems, etc. are examples of second generation embedded systems.
- **Third Generation:** With advances in processor technology, embedded system developers started making use of powerful 32bit processors and 16bit microcontrollers for their design. A new concept of application and domain specific processor controllers like Digital Signal Processors (DSP) and Application Specific Integrated Circuits (ASICs) came into the picture. The instruction set of processors became more complex and powerful and the concept of instruction pipelining also evolved. The processor market was flooded with different types of processors from different vendors. Processors like Intel Pentium, Motorola 68K, etc. gained attention in high performance embedded requirements. Dedicated embedded real time and general purpose operating systems entered into the embedded market. Embedded systems spread its ground to areas like robotics, media, industrial process control, networking, etc.
- **Fourth Generation:** The advent of System on Chips (SOC), reconfigurable processors and multi core processors are bringing high performance, tight integration and miniaturization into the embedded device market. The SOC technique implements a total system on a chip by integrating different functionalities with a processor core on an integrated circuit. We will discuss about SOCs in a later chapter. The fourth generation embedded systems are making use of high performance real time embedded operating systems for their functioning. Smart phone devices, mobile internet devices (MIDs), etc. are examples of fourth generation embedded systems.
- **Classification Based on Complexity and Performance:** This classification is based on the complexity and system performance requirements. According to this classification, embedded systems can be grouped into:
- **Small-Scale Embedded Systems:** Embedded Systems which are simple in application needs and where the performance requirements are not time critical fall under this category. An electronic toy is a typical example of a small-scale embedded system. Small-scale embedded systems are usually built around low performance and low post or 16 bit microprocessors/microcontrollers. A small-scale embedded system may or may not contain an operating system for its functioning.

- **Medium-Scale Embedded Systems:** Embedded systems which are slightly complex in hardware and firmware (software) requirements fall under this category. Medium-scale embedded systems are usually built around medium performance, low cost 16 or 32 bit microprocessors/microcontrollers or digital signal processors, they usually contain an embedded operating system (either general purpose or real time operating system) for functioning.
- **Large-Scale embedded Systems/Complex Systems:** Embedded systems which involve highly complex hardware and firmware requirements fall under this category. They are employed in mission critical applications demanding high performance. Such systems are commonly built around high performance 32 or 64 bit RISC processors/controllers or Reconfigurable System on Chip (RSOC) or multi-core processors and programmable logic devices. They may contain multiple processors/controllers and co-units/hardware accelerators for offloading the processing requirements from the main processor of the system. Decoding/encoding of media, cryptographic function implementation, etc. are examples for processing requirements which can be implemented using a co-processor/hard- ware accelerator. Complex embedded systems usually contain high performance Real Time Operating System (RTOS) for task scheduling, prioritization and management:

Major Application Areas of Embedded Systems

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

    1. ***Consumer electronics:*** Camcorders, cameras, etc.
    2. ***Household appliances:*** Television, DVD players, washing machine, fridge, microwave oven, etc.
    3. ***Home automation and security systems:*** Air conditioners, sprinklers, intruder detection alarms, closed circuit television cameras, fire alarms, etc.
    4. ***Automotive industry:*** Anti-lock breaking systems (ABS), engine control, ignition systems, automatic navigation systems, etc.

5. ***Telecom:*** Cellular telephones, telephone Switches, handset multimedia applications, etc.
6. ***Computer peripherals:*** Printers, scanners, fax machines, etc.
7. ***Computer networking systems:*** Network routers, switches, hubs, firewall etc.
8. ***Healthcare:*** Different kinds of scanners, EEG, ECG •machines etc.
9. ***Measurement & Instrumentation:*** Digital multi meters, digital Os, 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.

PURPOSE OF EMBEDDED SYSTEMS

- As mentioned in the previous section, embedded systems are used in various domains like consumer Electronics, home automation, telecommunications, automotive industry, healthcare, control & instrumentation, retail and banking the domain itself, according to the application usage context, they may have different functionalities. Each embedded system is designed to serve the purpose of any one or a combination following tasks:
    1. Data collection/Storage/Representation
    2. Data communication
    3. Data (signal) processing.
    4. Monitoring
    5. Control
    6. Applications specific user interface

1. Data Collection/Storage/Representation

- Embedded systems designed for the purpose of data collection perform acquisition of data from the external world. Data collection is usually done for storage, analysis, manipulation and transmission. The term "data" refers all kinds of information, viz. text, voice, image, video, electrical signals and any other measurable quantities. Data can be either analog (continuous) or digital (discrete). Embedded systems with analog data capturing techniques collect data directly in the form of analog signals whereas embedded systems with digital data collection mechanism converts the analog signal to

corresponding digital signal using analog to digital (A/D) conveners and then collects the binary equivalent of the analog data. If the data is digital, it can be directly captured without any additional interface by digital embedded systems.

- The collected data may be stored directly in the system or may be transmitted to some other systems or it may be processed by the system or it may be deleted instantly after giving a meaningful representation. These actions are purely dependent on the purpose for which the embedded system is designed. Embedded systems designed for pure measurement applications without storage, used in control and instrumentation domain collects data and gives a meaningful representation of the collected data by means of graphical representation or quantity value and deletes the collected data when new data arrives at the data collection terminal. Analog and digital CROs without storage memory are typical examples of this. Any measuring equipment used in the medical domain for monitoring without storage functionality also comes under this category. Some embedded systems store the collected data for processing and analysis. Such systems incorporate a built-plug-in storage memory for storing the captured data. Some of them give the user a meaningful representation of the collected data by visual (graphical/quantitative) or audible means using display units [Liquid Crystal Display (LCD), Light Emitting Diode (LED), etc.] buzzers, alarms, etc. Examples are: measuring instruments with storage memory and monitoring instruments with storage memory used in medical applications. Certain embedded systems store the data and will not give a representation of the same to the user, whereas the data is used for internal processing. A digital camera is a typical example of embedded system with data collection / storage representation of data. Images are captured and digital camera for image capturing/ the captured image may be stored within the memory of the camera. The captured image can also be presented to the user through a graphic LCD unit.

2. Data Communication

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

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

3. Data (Signal) Processing

- As mentioned earlier, the data (voice, image, video, electrical signals and other measurable quantities) collected by embedded systems may be used for various kinds of data processing. Embedded systems with signal processing functionalities are employed in applications demanding signal processing like speech coding, synthesis, audio video codec, transmission applications, etc. A digital hearing aid is a typical example of an embedded system employing data processing. Digital hearing aid improves the hearing capacity of hearing impaired persons. A digital hearing aid employ

4. Monitoring

- Embedded systems falling under the category are specifically designed for monitoring purpose. Almost all embedded products coming under the medical domain are with monitoring functions only. They are used for determining the state of some variables using input sensors. They cannot impose control over variables. A very good example is the electro cardiogram (ECG) machine for monitoring the heartbeat of a patient. The machine is intended to do the monitoring of the heartbeat. It cannot impose control over the heartbeat. The sensors used in ECG are the different electrodes connected to the patient's body. Some other examples of embedded systems with monitoring function are measuring instruments like digital CRO, digital millimeters, logic analyzers, etc. used in Control & Instrumentation applications. They are used for knowing (monitoring) the status of some variables like current, voltage, etc. They cannot control the variables in turn.

5. Control

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

- Air conditioner system used in our home to control the room temperature to a specified limit is a typical example for embedded system for control purpose. An air conditioner contains a room temperature- sensing element (sensor) which may be a thermostat and a handheld unit for setting up (feeding) the desired temperature. The handheld unit may be connected to the central embedded unit residing inside the air conditioner through a wireless link or through a wired link. The air compressor unit acts as the actuator. The compressor is controlled according to the current room temperature and the desired temperature set by the end user. Here the input variable is the current room temperature and the controlled variable is also the room temperature. The controlling variable is cool air flow e compressor unit. If the controlled variable and input variable are not at the same value, controlling variable tries to equalize them through t taking actions on the cool air flow.

6. Application Specific User Interface

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

TYPICAL EMBEDDED SYSTEM

- A typical embedded system (Fig. 2.1) contains a single chip controller, which acts as the master brain of the system. The controller can be a Microprocessor' (e.g. Intel 8085) or a microcontroller (e.g. Atmel AT89C51) or a Field Programmable Gate Array (FPGA) device (e.g. Xilinx Spartan) or a Digital Signal Processor (DSP) (e.g. Black fin @ Processors from Analog Devices) or an Application Specific Integrated
- Circuit (ASIC)/ Application specific Standard Product (ASSP) (e.g. ADE7760 Single Phase Energy Metering IC from) Analog Devices for energy metering applications). 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 where- as 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 inter- faces 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 digitization. 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 con-figuration 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 imply menting some kind of memory protection mechanism. The most common types of memories used in embedded systems for control algorithm storage are OTP, PROM, UVEPROM, EEPR () M and FLASH. Depending on the control application, the memory size may vary or a few bytes to megabytes. We will discuss them in detail in the coming sections. Sometimes the "stem 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 kilobytes or megabytes depending on the application. The details given under the section Memory will give you a more detailed description of the working memory. 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. He only difference is that the memory of a new born baby is self-adaptive, meaning that the baby will two 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 maybe an EEPROM or FLASH memory varying from a few kilobytes to mega- bytes. 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 803 IAH.

CORE OF THE EMBEDDED SYSTEM

- Embedded systems are domain and application specific and are built around a central core. The core of the embedded system falls into any one of the following categories:
  1. General Purpose and Domain Specific Processors
     a. Microprocessors
     b. Microcontrollers
     c. Digital Signal Processors
  2. Application Specific Integrated Circuits (ASICs)
  3. Programmable Logic Devices (PLDs)
  4. Commercial off-the-shelf Components (COTS)
- If you examine any embedded system you will find that it is built around any of the core units mentioned above.

1. General Purpose and Domain Specific Processors

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

a). Microprocessors:

- A Microprocessor is a silicon chip representing a central processing unit (CPU), which is capable of performing arithmetic as well as logical operations according to a pre-de- fined set of instructions, which is specific to the manufacturer. In general the CPU contains the Arithmetic and Logic Unit (ALU), control unit and working registers. A microprocessor is a dependent unit and it requires the combination of other hardware like memory, timer unit, and interrupts controller, etc. for proper functioning. Intel claims the credit for developing the first microprocessor

unit Intel 4004, a 4bit processor which was released in November 1971. It featured 1k data memory, a 12bit program counter and 4K program memory, sixteen 4bit general purpose registers and 46 instructions. It ran at a clock speed of 740 kHz. It was designed for olden day's calculators. In 1972, 14 more instructions were added to the 4004 instruction set and the program space is upgraded to 8K. Also interrupt capabilities were added to it and it is renamed as Intel 4040. It was quickly replaced in April 1972 by Intel 8008 which was similar to Intel 4040, the only difference was that its program counter was 14 bits wide and the 8008 served as a terminal controller. In April 1974 Intel launched the first 8 bit processor, the Intel 8080, with 16bit address bus and program counter and seven 8bit registers (A-E,H,L: BC, DE, and HL pairs formed the 16bit register for this processor). Intel 8080 was the most commonly used processors for industrial control and other embedded applications in the 1975s. Since the processor required other hardware components as mentioned earlier for its proper functioning, the systems made out of it were bulky and were lacking compactness.

- Immediately after the release of Intel 8080, Motorola also entered the market with their processor, Motorola 6800 with a different architecture and instruction set compared to 8080.
- In 1976 Intel came up with the upgraded version of 8080 — Intel 8085, with two newly added instructions, three interrupt pins and serial I/O. Clock generator and bus controller circuits were built-in and the power supply part was modified to a single +5 V supply.
- In July 1976 Zilog entered the microprocessor market with its Z80 processor as competitor to Intel. Actually it was designed by an ex-Intel designer, Frederico Faggin and it was an improved version of Intel' 8080 processor, maintaining the original 8080 architecture and instruction set with an 8bit data bus and a 16bit address bus and yes capable of executing all instructions of 8080..It included 80 more new instructions and it brought out the concept of register banking by doubling the register set. Z80 also included two sets of index registers for flexible design.
- Technical advances in the field of semiconductor industry brought a new dimension to the micro- processor market and twentieth century witnessed a fast growth in, processor technology. 16, 32 and 64 bit processors came into the place of conventional 8bit processors. The initial 2 MHz clock is now an old story. Today processors with clock speeds up to 2.4 GHz are available in the market. More and more competitors entered

into the processor market offering high speed, high performance and low cost processors for customer design needs.

- Intel, AMD, Freescale, IBM, TI, Cyrix, Hitachi, NEC, LSI Logic, 'etc. are the key players in the processor market. Intel still leads the market with cutting edge technologies in the processor industry.
- Different instruction set and system architecture are available for the design of a microprocessor. Harvard and Von-Neumann are the two common system architectures for processor design. Processors based on Harvard architecture contains separate buses for program memory and data memory, whereas processors based on Von-Neumann architecture shares a single system bus for program and data memo- ry. We will discuss more about these architectures later, under a separate topic. Reduced Instruction Set Computing (RISC) and Complex Instruction Set Computing (CISC) are the two common Instruction Set Architectures (ISA) available for processor design. We will discuss the same under a separate topic in this section...

General Purpose Processor (GPP) vs. Application-Specific Instruction Set Processor (ASIP)

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

b). Microcontroller

- Microcontroller is a highly integrated chip that contains a CPU, scratch pad RAM, special and general purpose register arrays, on chip ROWFLASH memory for program storage, timer and interrupt control units and dedicated I/O ports. Microcontrollers can be considered as a super set of microprocessors. Since a microcontroller contains all the necessary functional blocks for independent working, they found greater place in the embedded domain in place of microprocessors. Apart from this, they are cheap, cost effective and are readily available in the market. Texas Instrument's TMS 1000 is considered as the world's first microcontroller. We cannot say it as a fully functional microcontroller when we compare it with modern microcontrollers. TI followed Intel's 4004/4040, 4 bit processor design and added some amount of RAM, program storage memory (ROM) and I/O support on a single chip, there by eliminated the requirement of multiple hardware chips for self-functioning. Provision to add custom instructions to the CPU was another innovative feature of TMS 1000. TMS 1000 was released in 1974. In 1977 Intel entered the microcontroller market with a family of controllers coming under one umbrella named MCS-48TM family. The processors came under this family were, 8038HL, 8039HL, 8040AHL, 8048H, 8049H and 8050AH. Intel 8048 is recognized as Intel's first microcontroller and it was the most prominent member in the MCS-48TMt family. It was used in the original IBM PC key- board. The inspiration behind 8048 was Fairchild's F8 microprocessor and Intel's goal o' developing a low cost and small size processor. The design of 8048 adopted a true Harvard architecture where pro- gram and data memory shared the same address bus and is differentiated by the related control signals.
- Eventually Intel came out with its most fruitful design in the 8bit microcontroller domain—the 8051family and its derivatives. It is the most popular and powerful 8bit microcontroller ever built. It was developed in the 1980s and was put under the family MCS-51. Almost 75% of the microcontrollers used in the embedded domain were 8051 family based controllers during the 1980—90s. 8051 processor cores are used in more than 100 devices by more than 20 independent manufacturers like Maxim, Philips, Atmel, etc. under the license from Intel. Due to the low cost, wide availability, memory efficient instruction set, mature development tools and Boolean processing (bit manipulation operation) capability, 8051 family derivative microcontrollers are much used in high-volume consumer electronic devices, entertainment industry and other gadgets where cost-cutting is essential.
- Another important family of microcontrollers used in industrial control and embedded applications is the PIC family micro controllers from Microchip Technologies (It will be discussed in detail in a later section of this book). It is a high performance RISC microcontroller complementing the CISC (complex instruction set computing) features of 8051. The terms RISC and CISCS will are explained in detail in a separate heading.

- Some embedded system applications require only 8bit controller whereas some embedded applications requiring superior performance and computational need demand 16/32bit microcontrollers. Infineon, Free scale, Philips, Atmel, Maxim, Microchip etc. key suppliers of 16bit microcontrollers. Philips tried to extend the 8051 family microcontrollers to use for 16bit applications by developing the Philips XA (extended Architecture) microcontroller series.
- 8bit microcontrollers are commonly used in embedded systems where the processing power is not a big constraint. As mentioned earlier, more than 20 companies are producing different flavors of the 8051 family microcontroller. They try to add more and more functionalities like built in SPI, 12C serial buses, USB controller, ADC, Networking capability, etc. So the competitive market is driving towards a one-stop solution chip in microcontroller domain. High processing speed microcontroller families like ARM11 series are also available m the market, which provides solution to applications requiring hardware acceleration and high processing capability. Free scale, NEC, Zilog, Hitachi, Mitsubishi, Infineon, ST Micro Electronics, National, Texas Instruments, Toshiba, Philips, Microchip, Analog Devices, Daewoo, Intel, Maxim, Sharp, Silicon Laboratories, TDK, Triscend Win bond, Atmel, etc. are the key players in the microcontroller market. Of these at me] has got special significance. They are the manufacturers of a variety of Flash memory based microcontrollers. They also provide In-System Programmability (which will be discussed in detail in a later section of this book) for the controller. The Flash memory technique helps in fast reprogramming of the chip and thereby reduces the product development time. Atmel also provides another special family of microcontroller called AVR (it will be discussed in detail in a later chapter), an 8bit RISC Flash microcontroller, and fast enough to execute powerful instructions in a single clock cycle and provide the latitude you need to optimize power consumption. The instruction set architecture of a microcontroller can be either RISC or CISC. Microcontrollers are designed for either general purpose application requirement (general purpose controller) or domain- specific application requirement (application specific instruction set processor). The Intel 8051 micro- controller is a typical example for a general purpose microcontroller, whereas the automotive AVR microcontroller family from Atmel Corporation is a typical example for ASIP specifically designed for the automotive domain.

Microprocessor vs. Microcontroller The following table summarizes the differences between a microcontroller and microprocessor.

MICROPROCESSOR	MICROCONTROLLER
A silicon chip representing a central processing unit (CPU), which is capable of performing arithmetic as well as logical operation according to a pre defined set of instructions	A micro controller is a highly integrated chip that contains a CPU, scratch pad ram, special and general purpose register arrays on chip ROM/flash memory for program storage, timer and interrupted and control units and dedicated I/O ports.
It is a dependent unit. It requires the combination of other chips like timers, program and data memory chips, interrupt controls, etc. for functioning	It is a self contain unit And it does not require external interrupt controller, timer, UART, etc. for its functioning.
Most of the time general purpose in design and operation	Mostly application oriented or domain specific
Does not contain a built in I/o port functionally needs to be implemented with the help external programmable peripheral inter phase chips like 8255.	Most of the processor contains multiple build in I/o ports which can be operated as a single 8 or 16 or 32 bit port or as individual port pins.
Targeted for high end market where performance is important.	Targeted for embedded market where performance is not so critical (at present this demarcation is invalid)
Limited power saving options compare to micro controllers	Includes lot of power savings features

c). Digital Signal Processors

- Digital Signal Processors (DSPs) are powerful special purpose 8/16/32 bit microprocessors designed specifically to meet the computational demands and power constraints of today's embedded audio, video, and communications applications. Digital signal processors are 2 to 3 times faster than the general purpose microprocessors in signal processing applications. This is because of the architectural difference between the two. DSPs implement algorithms in hardware

which speeds up the execution

whereas general purpose processors implement the algorithm in firm- ware and the speed of execution depends primarily on the clock for the processors. In general, DSP can be viewed as a microchip designed for performing high speed computational operations for 'addition', 'subtraction', 'multiplication' and 'division'. A typical digital signal processor incorporates the following key units:

- **Program Memory**: for storing the program required by DSP to process the data
- **Data Memory:** Working memory for storing temporary variables and data/signal to be processed.
- **Computational Engine:** performs the signal processing in accordance with the stored program memory. Computational Engine incorporates many specialized arithmetic units and each of them operates simultaneously to increase the execution speed. It also incorporates multiple hardware shifters for shifting operands and thereby saves execution time.
- **I/O Unit:** Acts as an interface between the outside world and DSP. It is responsible for capturing signals to be processed and delivering the processed signals.

2. Application Specific Integrated Circuits. (ASIC)

- ASICs are a microchip design to perform specific and unique applications.
- Because of using single chip for integrates several functions there by reduces the system development cost.
- Most of the ASICs are proprietary (which having some trade name) products, it is referred as Application Specific Standard Products (ASSP).
- As a single chip ASIC consumes a very small area in the total system. Thereby helps in the design of smaller system with high capabilities or functionalities.
- The developers of such chips may not be interested in revealing the internal detail of it.

3. Programmable logic devices (PLD's)

- A PLD is an electronic component. It used to build digital circuits which are reconfigurable.
- A logic gate has a fixed function but a PLD does not have a defined function at the time of manufacture.
- PLDs offer customers a wide range of logic capacity, features, speed, voltage characteristics.
- PLDs can be reconfigured to perform any number of functions at any time

- A variety of tools are available for the designers of PLDs which are inexpensive and help to develop, simulate and test the designs.
- PLDs having following two major types.
    - CPLD (Complex Programmable Logic Device): CPLDs offer much smaller amount of logic up to 1000 gates.
    - FPGAs (Field Programmable Gate Arrays): It offers highest amount of performance as well as highest logic density, the most features.

Advantages of PLDs:-

- PLDs offer customer much more flexibility during the design cycle.
- PLDs do not require long lead times for prototypes or production parts because PLDs are already on a distributor's shelf and ready for shipment.
- PLDs can be reprogrammed even after a piece of equipment is shipped to a customer

4. Commercial off-the-shelf components (COTs)

- A Commercial off the Shelf product is one which is used 'as-is'.
- The COTS components itself may be develop around a general purpose or domain specific processor or a ASICs or a PLDs.
- The major advantage of using COTS is that they are readily available in the market, are chip and a developer can cut down his/her development time to a great extent
- The major drawback of using COTS components in embedded design is that the manufacturer of the COTS component may withdraw the product or discontinue the production of the COTS at any time if rapid change in technology occurs.

Advantages of COTS:

- Ready to use
- Easy to integrate
- Reduces development time

Disadvantages of COTS:

- No operational or manufacturing standard (all proprietary)
- Vendor or manufacturer may discontinue production of a particular COTS product

MEMORY

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

Program Storage Memory (ROM)

- The program memory or code storage memory of an embedded system stores the program instructions and it can be classified into different types as per the block diagram representation given in Fig. 2.8.
- Classification of Program Memory (ROM): The code memory retains its contents even after the power to it is turned off. It is generally known as non-volatile storage memory. Depending on fabricate, erasing and programming techniques they are classified into the following types.

Masked ROM (MROM):

- Masked ROM is a one-time programmable device. Masked ROM makes use of the hardwired technology for storing data. The device is factory programmed by masking and metallization process at the time of production itself, according to the data provided by the end user. The primary •advantage of this is low, cost for high volume production. They are the least expensive type of solid state memory. Different mechanisms are used for the masking process of the ROM, like
    1. Creation of an enhance mentor depletion mode transistor through channel implant.
    2. By creating the memory all either uses a standard transistor or a high threshold transistor. In the high threshold mode, the supply voltage required to turn ON the transistor is above the normal ROM IC operating Voltage. This ensures that the transistor is always off and the memory cell stores always logic O.
- Masked ROM is a good candidate for storing the embedded firmware for low cost embedded devices. Once the design is proven and the firmware requirements are tested and frozen, the binary data (The firmware cross compiled/assembled to target processor specific machine code) corresponding to it can be given to the MROM fabricator. The limitation with MROM based firmware storage is the inability to modify

the device firmware against firmware upgrades. Since the MROM is permanent in bit storage, it is not possible to alter the bit information.

Programmable Read Only Memory (PROM) / (OTP)

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

Erasable Programmable Read Only Memory (EPROM)

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

Electrically Erasable Programmable Read Only Memory (EEPROM)

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

FLASH

- FLASH is the Latest ROM technology and is the most popular ROM technology used in today's embedded designs. FLASH memory is @variation of EEPROM technology. It

combines the re-programmability of EEPROM and the high capacity of standard ROMs. FLASH memory is organized as sectors (blocks) or pages. FLASH memory stores information in an array of floating gate MOS- FET transistors. The erasing of memory can be done at sector level or page level without affecting the other sectors or pages. Each sector/page should be erased before re-programming. The typical erasable capacity of FLASH is 1000 cycle 7C512 from WINBOND is an example of 64KB FLASH memory.

NVRAM

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

Read-Write Memory/Random Access Memory (RAM)

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

Static RAM (SRAM)

- Static RAM stores data in the form of voltage. They are made up of flip-flops. Static RAM is the fastest form of RAM available. In typical implementation, an SRAM cell (bit) is realized using six transistors (or 6 MOSFETs). Four of the transistors are used for building the
- Classification of Working Memory (RIM):
- Latch (flip-flop) part of the memory cell and two for controlling the, access. SRAM is fast in operation due to its resistive networking and switching capabilities. To its simplest representation an SRAM cell can be visualized as shown in Fig. 2.10:

SRAM cell implementation:

- This implementation in its simpler form can be visualized as two-cross coupled inverters with read/ write control through transistors. The four transistors in the middle form the cross-coupled inverters. This can be visualized as shown in Fig. 2.11 From the SRAM implementation diagram, it is clear that access to the memory cell is controlled by the line Word Line, which controls the access transistors (MOSFETs) Q5 and Q6. The access transistors control the connection to bit lines B & B\. In order to write a value to the memory cell, apply the desired value to the bit control lines (For writing I, make B — I and B\ =O; for writing O, make B = O and B\ —I) and assert the Word Line (Make Word line high). This operation latches the bit written in the flip-flop. For reading the content of the memory cell, assert both B and bit lines to me and set the Word line to me. The major limitations of SRAM are low capacity and high cost. Since a minimum of six transistors are required to build a single memory cell, imagine how many memory cells we can fabricate on a silicon wafer.

Dynamic RAM (DRAM)

- Dynamic RAM stores data in the form of charge. They are made up of MOS transistor gates. The advantages of DRAM are its high density and low cost compared to SRAM. The disadvantage is that since the information is stored as charge it gets leaked off with time and to prevent this they need to be refreshed periodically. Special circuits called DRAM controllers are used for the refreshing operation. The refresh operation is done periodically in millisecond's interval. Figure 2.12 illustrates the typical implementation of a DRAM cell.
- The MOSFET acts as the gate for the incoming and outgoing data whereas the capacitor acts as the bit storage unite Table given below summarizes the relative merits and demerits of SRAM and DRAM technology.

NVRAM

- On-volatile RAM is a random access memory with battery backup. It contains static RAM based memory and a minute battery for providing supply to the memory in the absence of external power supply. The memory and battery are packed together in a single package. NVRAM is used for the non-volatile storage of results of operations or for setting up of flags, etc. The life span Of NV RAM is expected to be around 10 years. DS1744 from Maxim/DaIIas is an example for 32KB NVRAM.

Memory According to the Type of Interface

- The interface (connection) of memory with the processor/controller can be of various types. It may be a parallel interface [The parallel data lines (DO-D7) for an 8 bit

processor/controller will be connected to DO-D7 of the memory] or the interface may be a serial interface like 12C (Pronounced as I Square C. It is a 2 line serial interface) or it may be an SPI (Serial peripheral interface, 2+n line interface where n stands for the total number of SPI bus devices in the system). It can also be of a single wire interconnection (like Dallas I-Wire interface). Serial interface is commonly used for data storage memory like EEPROM. The memory density of a serial memory is usually expressed in terms of kilobits, whereas that of a parallel interface memory' is expressed in terms of kilobytes. Atmel Corporations AT24C512 is an example for serial memory with capacity 512 kilobits and 2-wire interface. Please refer to the section 'Communication Interface' for more details on 12C, SPI and I-Wire Bus.

Memory Shadowing

- Generally the execution of a program or a configuration from Read Only Memory (ROM) is very slow (120 to 200 ns) compared to the execution from a random access memory (40 to70 ns). From the timing parameters it is obvious that RAM access is about three times as fast as ROM access. Shadowing of me only is a technique adopted to solve the execution speed problem in processor-based systems. In computer systems and video systems there will be a configuration holding ROM called Basic Input Output Configuration ROM or simply BIOS. In personal computer systems BIOS stores the hardware configuration information like the address assigned for various serial ports and other non-plug 'n' play devices, etc. Usually it is read and the system is configured according to it during system boot up and it is time consuming. Now the manufactures included a RAM behind the logical layer of BIOS at its same address as a shadow to the BIOS and the first step that happens during the boot up is copying the BIOS to the shadowed RAM and write protecting the RAM then disabling the BIOS reading. You may be thinking that what a stupid idea it is and why both RAM and ROM are needed for holding the same data. The answer is: RAM is volatile and it cannot hold the configuration data which is copied from the BIOS when the power supply is switched off. Only a ROM can hold it permanently. But for high system performance it should be accessed from RAM instead of accessing from a ROM.

Memory Selection for Embedded Systems

- Embedded systems require a program memory for holding the control algorithm (For a super-loop based design) or embedded OS and the applications designed to run on top of it (for OS based designs), data memory for holding variables and temporary data during task execution, and memory for holding non- volatile data (like configuration data, look up table etc) which are modifiable by the application (Unlike program memory which is non-volatile as well unalterable by the end user). The memory requirement for an embedded system in terms of RAM and ROM (EEPROM/FLASH/NVRAM) is solely dependent on the type of the embedded system and

the applications for which it is designed. There is no hard and fast rule for calculating the memory requirements. Lot of factors need to be considered when selecting the type and size of memory for embedded system. For example, if the embedded system is designed using SOC or a microcontroller with on-chip RAM and ROM (FLASH/EEPROM), depending on the application need the on-chip memory may be sufficient for designing the total system. As a rule of thumb, identify your system requirement and based on the type of processor (SOC or microcontroller with on- chip memory) used for the design, take a decision on whether the on-chip memory is sufficient or external memory is required. Let's consider a simple electronic toy design as an example. As the complexity of requirements are less and data memory requirement are minimal, we can think of a microcontroller with a few bytes Of internal RAM, a few bytes or kilobytes (depending on the number of tasks and the complexity of tasks) of FLASH memory and a few bytes of EEPROM (if required) for designing the system. Hence there is no need for external memory at all. A PIC microcontroller device which satisfies the I/O and memory requirements can be used in this case. If the embedded design is based on an RTOS, the RTOS requires certain amount of RAM for its execution and ROM for storing the RTOS image (Image is the common name give) for the binary data generated by the compilation of all RTOS source files). Normally the binary code for RTOS kernel containing all the services is stored in a non-volatile memory (Like FLASH) as either compressed or non-compressed data. During boot-up of the device, the RTOS files are copied from the program storage memory, decompressed if required and then loaded to the RAM for execution. The supplier of the RTOS usually gives a rough estimate on the run time RAM requirements and program memory requirements for the RTOS. On top of this add the RAM requirements for executing user tasks and ROM for storing user applications. On a safer side, always add a buffer value to the total estimated RAM and ROM size requirements. A smart phone device with Windows mobile operating system is a typical example for embedded device with ()S. Say 64MB RAM and 128MB ROM are the minimum requirements for running the Windows mobile device, indeed you need extra RAM and ROM for running user applications. So while building the system, count the memory for that also and arrive at a value which is always at the safer side, so that you won't end up in a situation where you don't have sufficient memory to install and run user applications. There are two parameters for representing a memory. The first one is the size of the memory ship (Memory density expressed in terms of number of memory bytes per chip). There is no option to get a memory chip with the exact required number of bytes. Memory chips come in standard sizes like 512bytes, 1024bYtes (1 kilobyte), 2048bytes (2 kilobytes), 4Kb,t 8Kb, 16Kb, 32Kb, 256Kb, 512Kb, 1024Kb (I megabytes), etc. Suppose your embedded application requires only 750 bytes

of RAM, you don't have the option of getting a memory chip with size 750 bytes the only option left with is to choose the memory chip with a size closer to the size needed. Here 1024 bytes is the least possible option. We cannot go for 512 bytes, because the minimum requirement 750 bytes. While you select a memory size, always keep in mind the address range supported by your processor. For example, for a processor/ controller with 16 bit address bus, the maximum number of memory locations that can be addressed is 216 = 65536 bytes = 64Kb. Hence it is meaningless to select a 128Kb memory chip for a processor with 16bit wide address bus. Also, the entire memory range supported by the processor/controller may not be available to the memory chip alone. It may be shared between I/O, other ICs and memory. Suppose the address bus is 16bit wide and only the lower 32Kb address range is assigned to the memory chip, the memory size maximum required is 32Kb 6nIy. It is not worth to use a memory chip with size 64Kb in such a situation. The second parameter that needs to be considered in selecting a memory is the word size of the memory. The word Size refers to the number of memory bits that can be read/write together at a time. 4, 8, 12, 16, 24, 32, etc. are the word sizes supported by memory chips. Ensure that the word size supported by the memory chip matches with the data bus width of the processor/controller. FLASH memory is the popular choice for ROM (program storage memory) in embedded applications. It is a powerful and cost-effective solid-state storage technology for mobile electronics devices and other consumer applications. FLASH memory comes in two major variants, namely, NAND and NOR FLASH. NAND FLASH is a high-density low cost non-volatile storage memory: On the other hand, NOR FLASH is less dense and slightly expensive. But it supports the Execute in Place (XIP) technique for program execution. The XIP technology allows the execution of code memory from ROM itself without the need for copying it to the RAM as in the case of conventional execution method. It is a good practice to use a combination of NOR and NAND memory for storage memory requirements, where NAND can be used for storing the program code and or data like the data captured in a camera device. NAND FLASH doesn't support XIP and if NAND FLASH is used for storing program code, a DRAM can be used for copying and executing the program code. NOR FLASH supports XIP and it can be used as the memory for boot loader or for even storing the complete program code. The EEPROM data storage memory is available as either serial interface or parallel interface chip. If the processor/controller of the device supports serial interface and the amount of data to write and read to and from the device is less, it is better to have a serial EEPROM chip. The serial EEPROM saves the address space of the total system. The memory capacity of the serial EEPROM is usually expressed in bits or kilobits. 512 bits, 1Kbits, 2Kbits, 4Kbits, etc. are examples for serial EEPROM memory representation. For embedded systems with low power requirements like portable devices, choose low power memory devices.

Certain embedded devices may be targeted for operating at extreme environmental conditions like high temperature, high humid area, etc. Select an industrial grade memory chip in place of the commercial grade chip for such devices.

SENSORS AND ACTUATORS

- At the very beginning of this chapter it is already mentioned that an embedded system is in constant interaction with the Real world and the controlling/monitoring functions executed by the embedded system is achieved in accordance with the changes happening to the Real world. The changes in sys- tem environment or variables are detected by the sensors connected to the input port of the embedded system. If the embedded system is designed for any controlling purpose the system will produce some changes in the controlling variable to bring the controlled variable to the desired value. It is achieved through an actuator connected to the output port of the embedded system. If the embedded system is designed for monitoring purpose only, then there is no need for including an actuator in the system. For example, take the case of an ECG machine. It is design to monitor the heart beat status of a patient and it cannot impose a control over the patient's heart beat and its order. The sensors used here are the different electrode sets connected to the body of the patient. The variations are captured and presented to the user (may be a doctor) through a visual display or some printed chart.

SENSORS

- Sensors are also called as detectors.
- The changes in the system environment or variables are detected by the sensors connected to the input port of the embedded system.
- It is a transducer that converts energy from one type to another type for any particular purpose.
- Example- ECG machine it is designed to monitor the heartbeat status of a patient and it cannot impose a control over the patient’s heart beat and its order. The sensors are used here are the different electrode sets connected to the body of the patient.
- The variations are captured and presented to the user through a visual display or some printed chart.

ACTUATORS

- Actuator is a form of transducer device which converts signals to corresponding

physical action.

- Actuator acts as an output device.
- If the embedded system is designed for any controlling purpose the system will produce some changes in the controlling variable to bring the controlled variable to the desired value. This is achieved through an actuator connected to the output port of the embedded system.
- If the E.S is designed for monitoring purpose only then there is no need for including an actuator in the system.
- Types of Actuators
    - Multi-Turn Actuator
    - Part-Turn Actuator Linear Actuator
    - Multi-turn Actuator-
- It is an actuator which transmits to the valve torque for at least one full revolution. It is capable of withstanding thrust.
- It is required for the automation of multi-turn valves.
- One of the main types of this is the gate valve

Part-turn actuators –

- It is an actuator which transmits a torque to the valve for less than one full revolution. It is not capable of withstanding thrust.
- The major representatives of this type are butterfly valves and ball valves.

Linear Actuator –

- The major representative of this type is the control valves.
- Just like the plug in the bathtub is pressed into the drain the plug is pressed into the plug seat by a stroke.

The I/O Subsystem

- The I/O subsystem of the embedded system facilitates the interaction of the embedded system with the external world. As mentioned earlier the interaction happens through the sensors and actuators connected to the input and output ports respectively of the embedded system. The sensors may not be directly interfaced to the input ports; instead they may be interfaced through signal conditioning and translating systems like ADC, up to couplers, etc. This section illustrates some of the sensors and actuators used in embedded systems and the I/O systems to facilitate the interaction of embedded systems with external world.

COMMUNICATION INTERFACES

These are the devices through with the E.S can interact with various subsystems and the external world. For embedded product communication interface can be viewed in two different perspectives:

1. Device/board level communication interface (Onboard communication Interface)
2. Product level communication interface (External communication interface)

ONBOARD COMMUNICATION INTERFACE

- The communication channel which interconnects the various components with in an embedded product is referred as device/broad level communication interface.
- Examples – Serial interfaces like I2C, I-Wire, and parallel bus interface.

Inter Integrated Circuit Bus (I2C Bus) –

- It is a synchronous bi-directional half duplex two–wire serial bus which provides communication link between integrated circuits.
- It was designed by Philips Semiconductors in1980s.
- It was developed to provide an easy way of connection between a microprocessor /microcontroller system and peripheral chips in television sets.
- It comprises of two bus lines i.e. Serial Clock-SCL and Serial Data-SDA.
- SCL line is responsible for generating synchronization clock pulses.
- SDA is responsible for transmitting the serial data across devices.
- I2C bus is a shared bus system to which many number ofI2C devices can be connected.
- Devices connected to the I2C runs can act as either "Master" device or "Slave" device.
- The Master device is responsible for controlling the communication by initiating or terminating data transfer, sending data and generating necessary synchronization clock pulses.
- The Slave devices wait for the commands from the Master and respond upon receiving the commands.
- Master and Slave devices can act as either transmitter or receiver.
- Regardless whether a master is acting as transmitter or receiver the synchronization clock signal is generated by Master device only.
- I2C supports multi masters on the same bus.

EXTERNAL COMMUNICATION INTERFACE

- These are the E.S which may be a part of large distributed system and they require interaction and data transfer between various devices and sub modules.
- The product level communication interface is responsible for data transfer between the
  E.S and other devices or modules.
- The external communication interface can be either a wired media or a wireless media and it can be a serial or a parallel interface.
- Examples – Infrared (IR), Bluetooth (BT), Wireless LAN (Wi-Fi), Radio Frequency waves etc.

Infrared –

- Infrared is a serial, half duplex, line of sight based wireless technology for data communication between devices. The remote control of TV, AC works on the infrared data communication principle.IR uses infrared waves of the electromagnetic spectrum for transmitting the data. It supports point-point and point-to-multipoint communication. The typical communication range for IR lies in the range of 10 cm to1m.The range can be increased by increasing the transmitting power of the IR device.IR supports data rates ranging from 9600bits/sec to 16Mbps.

UNIT-2

EMBEDDED

FIRMWARE

INTRODUCTION:

- The control algorithm (Program instructions) and or the configuration settings that an embedded system developer dumps into the code (Program) memory of the embedded system.
- It is an un-avoidable part of an embedded system.
- The embedded firmware can be developed in various methods like
    - Write the program in high level languages like Embedded C/C++ using an Integrated Development Environment (The IDE will contain an editor, compiler, linker, debugger, simulator etc. IDEs are different for different family of processors/controllers.
    - Write the program in Assembly Language using the Instructions Supported by your application's target processor/controller

EMBEDDED FIRMWARE DESIGN & DEVELOPMENT:

- The embedded firmware is responsible for controlling the various peripherals of the embedded hardware and generating response in accordance with the functional requirements of the product.
- The embedded firmware is the master brain of the embedded system.
- The embedded firmware imparts intelligence to an embedded system.
- It is a onetime process and it can happen at any stage.
- The product starts functioning properly once the intelligence imparted to the product by embedding the firmware in the hardware.
- The product will continue serving the assigned task till hardware breakdown occurs or a corruption in embedded firmware.
- In case of hardware breakdown, the damaged component may need to be replaced and for firmware corruptions the firmware should be re-loaded, to bring back the embedded product to the normal functioning.
- The embedded firmware is usually stored in a permanent memory (ROM) and it is non alterable by end users.

- Designing Embedded firmware requires understanding of the particular embedded product hardware, like various component interfacing, memory map details, I/O port details, configuration and register details of various hardware chips used and some programming language (either low level Assembly Language or High level language like C/C++ or a combination of the two)
- The embedded firmware development process starts with the conversion of the firmware requirements into a program model using various modeling tools.
- The firmware design approaches for embedded product is purely dependent on the complexity of the functions to be performed and speed of operation required.
- There exist two basic approaches for the design and implementation of embedded firmware, namely;
    - The Super loop based approach
    - The Embedded Operating System based approach
- The decision on which approach needs to be adopted for firmware development is purely dependent on the complexity and system requirements

A). Embedded firmware Design Approaches – The Super loop:

- The Super loop based firmware development approach is Suitable for applications that are not time critical and where the response time is not so important (Embedded systems where missing deadlines are acceptable).
- It is very similar to a conventional procedural programming where the code is executed task by task
- The tasks are executed in a never ending loop.
- The task listed on top on the program code is executed first and the tasks just below the top are executed after completing the first task
- A typical super loop implementation will look like:
    - Configure the common parameters and perform initialization for various hardware components memory, registers etc.
    - Start the first task and execute it
    - Execute the second task
    - Execute the next task
    - Execute the last defined task

- Jump back to the first task and follow the same flow.

- The ‘C’ program code for the super loop is given below

```
void main ()
{
Configurations
();
Initializations
(); while (1)
{
Task 1 ();
Task 2 ();
:
:
Task n ();
}
}
```

Pros:

- Doesn’t require an Operating System for task scheduling and monitoring and free from OS related overheads
- Simple and straight forward design
- Reduced memory footprint

Cons:

- Non Real time in execution behavior (As the number of tasks increases the frequency at which a task gets CPU time for execution also increases)
- Any issues in any task execution may affect the functioning of the product (This can be effectively tackled by using Watch Dog Timers for task execution monitoring)

Enhancements:

- Combine Super loop based technique with interrupts
- Execute the tasks (like keyboard handling) which require Real time attention as Interrupt Service routines.

B). Embedded firmware Design Approaches – Embedded OS based Approach:

The embedded device contains an Embedded Operating System which can be oneof:

- A Real Time Operating System (RTOS)
- A Customized General Purpose Operating System (GPOS)
- The Embedded OS is responsible for scheduling the execution of user tasks and the allocation of system resources among multiple tasks
- It Involves lot of OS related overheads apart from managing and executing user defined tasks
- Microsoft® Windows XP Embedded is an example of GPOS for embedded devices
- Point of Sale (PoS) terminals, Gaming Stations, Tablet PCs etc are examples of embedded devices running on embedded GPOSs
- 'Windows CE', 'Windows Mobile', 'QNX', 'VxWorks', 'ThreadX', 'MicroC/OS-II', 'Embedded Linux', 'Symbian' etc are examples of RTOSs employed in Embedded Product development
- Mobile Phones, PDAs, Flight Control Systems etc are examples of embedded devices that runs on RTOSs

EMBEDDED FIRMWARE DEVELOPMENT LANGUAGES/OPTIONS

- Assembly Language
- High Level Language
    - Subset of C (Embedded C)
    - Subset of C++ (Embedded C++)
    - Any other high level language with supported Cross-compiler
- Mix of Assembly & High level Language
    - Mixing High Level Language (Like C) with Assembly Code
    - Mixing Assembly code with High Level Language (Like C)
    - Inline Assembly

(A). Assembly Language

- 'Assembly Language' is the human readable notation of 'machine language'
- 'Machine language' is a processor understandable language

- Machine language is a binary representation and it consists of 1s and 0s
- Assembly language and machine languages are processor/controller dependent
- An Assembly language program written for one processor/controller family will not work with others
- Assembly language programming is the process of writing processor specific machine code in mnemonic form, converting the mnemonics into actual processor instructions (machine language) and associated data using an assembler
- The general format of an assembly language instruction is an Opcode followed by Operands
- The Opcode tells the processor/controller what to do and the Operands provide the data and information required to perform the action specified by the opcode
- It is not necessary that all opcode should have Operands following them. Some of the Opcode implicitly contains the operand and in such situation no operand is required. The operand may be a single operand, dual operand or more

The 8051 Assembly Instruction

MOV A, #30

Moves decimal value 30 to the 8051 Accumulator register. Here MOV A is the Opcode and 30 is the operand (single operand). The same instruction when written in machine language will look like

01110100 00011110

The first 8 bit binary value 01110100 represents the opcode MOV A and the second 8 bit binary value 00011110 represents the operand 30.

- Assembly language instructions are written one per line
- A machine code program consists of a sequence of assembly language instructions, where each statement contains a mnemonic (Opcode + Operand)
- Each line of an assembly language program is split into four fields as:

LABEL OPCODE OPERAND COMMENTS

- LABEL is an optional field. A 'LABEL' is an identifier used extensively in programs to reduce the reliance on programmers for remembering where data or code is located.
  LABEL is commonly used for representing
  - A memory location, address of a program, sub-routine, code portion etc.

- The maximum length of a label differs between assemblers.ϖ Assemblers insist strict formats for labeling. Labels are always suffixed by a colon and begin with a valid character. Labels can contain number from 0 to 9 and special character _ (underscore).

```
#################################################################
; SUBROUTINE FOR GENERATING DELAY
; DELAY PARAMETR PASSED THROUGH REGISTER R1
; RETURN VALUE NONE, REGISTERS USED: R0, R1
#################################################################
#####
DELAY:      MOV R0, #255      ; Load Register R0 with
255 DJNZ R1, DELAY            ; Decrement R1 and loop
till        R1= 0 RET                          ;
Return to calling program
```

- The symbol; represents the start of a comment. Assembler ignores the♣ text in a line after the ; symbol while assembling the program
- DELAY is a label for representing the start address of the memory♣ location where the piece of code is located in code memory
- The above piece of code can be executed by giving the label DELAY as♣ part of the instruction. E.g. LCALL DELAY; LMP DELAY

Assembly Language – Source File to Hex File Translation:

- The Assembly language program written in assembly code is saved as .asm (Assembly file) file or a .src (source) file or a format supported by the assembler
- Similar to 'C' and other high level language programming, it is possible to have multiple source files called modules in assembly language programming. Each module is represented by a '.asm' or '.src' file or the assembler supported file format similar to the '.c' files in C programming
- The software utility called 'Assembler' performs the translation of assembly code to machine code
- The assemblers for different family of target machines are different. A51 Macro Assembler from Keil software is a popular assembler for the 8051 family micro controller

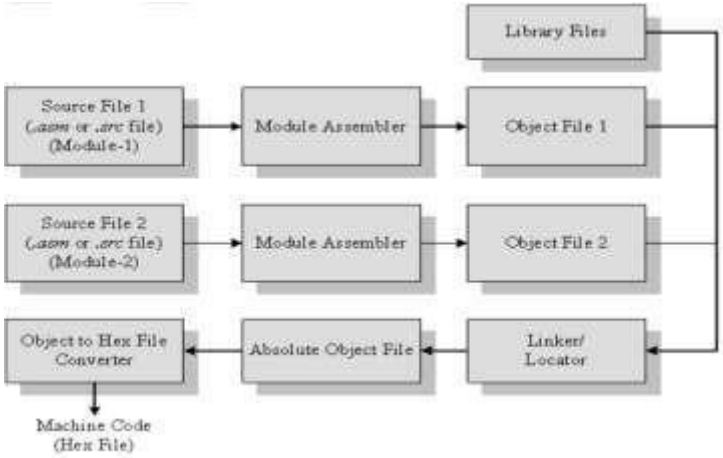


Figure 5: Assembly Language to machine language conversion process

- Each source file can be assembled separately to examine the syntax errors and incorrect assembly instructions
- Assembling of each source file generates a corresponding object file. The object file does not contain the absolute address of where the generated code needs to be placed (a re-locatable code) on the program memory
- The software program called linker/locater is responsible for assigning absolute address to object files during the linking process
- The Absolute object file created from the object files corresponding to different source code modules contain information about the address where each instruction needs to be placed in code memory
- A software utility called 'Object to Hex file converter' translates the absolute object file to corresponding hex file (binary file)

Advantages:

1. ***Efficient Code Memory & Data Memory Usage (Memory Optimization):***
   - The developer is well aware of the target processor architecture and¬ memory organization, so optimized code can be written for performing operations.
   - This leads to less utilization of code memory and efficient utilization¬ of data memory.
2. ***High Performance:***
   - Optimized code not only improves the code memory usage but also¬ improves the total system performance.

- Through effective assembly coding, optimum performance can be achieved for target processor.

3. ***Low level Hardware Access:***

- Most of the code for low level programming like accessing external device specific registers from OS kernel ,device drivers, and low level interrupt routines, etc are making use of direct assembly coding.

4. ***Code Reverse Engineering:***

- It is the process of understanding the technology behind a product by extracting the information from the finished product.
- It can easily be converted into assembly code using a dis-assembler program for the target machine.

Drawbacks:

1. ***High Development time:***

- The developer takes lot of time to study about architecture, memory organization, addressing modes and instruction set of target processor/controller.
- More lines of assembly code are required for performing a simple action.

2. ***Developer dependency:***

- There is no common written rule for developing assembly language based applications.

3. ***Non portable:***

- Target applications written in assembly instructions are valid only for that particular family of processors and cannot be re-used for other target processors/controllers.
- If the target processor/controller changes, a complete re-writing of the application using assembly language for new target processor/controller is required.

(B). High Level Language

- The embedded firmware is written in any high level language like C, C++
- A software utility called 'cross-compiler' converts the high level language to target processor specific machine code

- The cross-compilation of each module generates a corresponding object file. The object file does not contain the absolute address of where the generated code needs to be placed (a re-locatable code) on the program memory
- The software program called linker/locater is responsible for assigning absolute address to object files during the linking process
- The Absolute object file created from the object files corresponding to different source code modules contain information about the address where each instruction needs to be placed in code memory
- A software utility called 'Object to Hex file converter' translates the absolute object file to corresponding hex file (binary file)

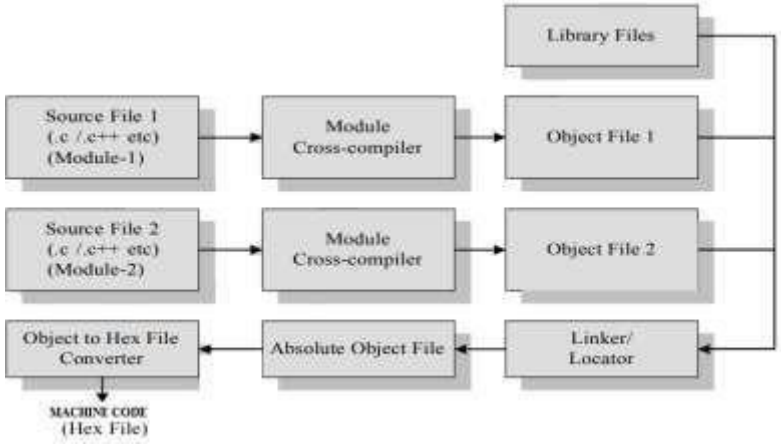


Figure 6: High level language to machine language conversion process

Advantages:

- ***Reduced Development time:*** Developer requires less or little knowledge on internal hardware details and architecture of the target processor/Controller.
- ***Developer independency:*** The syntax used by most of the high level languages are universal and a program written high level can easily understand by a second person knowing the syntax of the language
- ***Portability:*** An Application written in high level language for particular target processor
  /controller can be easily be converted to another target processor/controller specific application with little or less effort

Drawbacks:

- The cross compilers may not be efficient in generating the optimized target processor specific instructions.

- Target images created by such compilers may be messy and no optimized in terms of performance as well as code size.
- The investment required for high level language based development tools (IDE) is high compared to Assembly Language based firmware development tools.

EMBEDDED SYSTEM DEVELOPMENT ENVIRONMENT

- The most important characteristic of E.S is the cross-platform development technique.
- The primary components in the development environment are the host system, the target system and many connectivity solutions between the host and the target E.S.
- The development tools offered by the host system are the cross complier, linker and source-level debugger.
- The target embedded system offers a dynamic loader, link loader, a monitor and a debug agent.
- Set of connections are required between the source computer and the target system.
- These connections can be used for transmitting debugger information between the host debugger and the target debug agent.

IDE:

- In E.S IDE stands for an integrated environment for developing and debugging the target processor specific embedded firmware.
- An IDE is also known as integrated design environment or integrated debugging environment.
- IDE is a software package which bundles a "Text Editor", "Cross-compiler", "Linker" and a "Debugger".
- IDE is a software application that provides facilities to computer programmers for software development.
- IDEs can either command line based or GUI

based. IDE consists of:

1. Text Editor or Source code editor
2. A compiler and an interpreter
3. Build automation tools

4. Debugger
5. Simulators
6. Emulators and logic analyzer

- The example of IDE is Turbo C/C++ which provides platform on windows for development of application programs with command line interface.
- The other category of IDE is known as Visual IDE which provides the platform for visual development environment, ex- Microsoft Visual C++.
- IDEs used in embedded firmware are slightly different from the generic IDE used for high level language based development for desktop applications.
- In Embedded applications the IDE is either supplied by the target processor/controller manufacturer or by third party vendors or as Open source.
- An Integrated Development Environment (IDE) is software that assists programmers in developing software
- IDEs normally consist of a source code editor, a compiler, a linker/locater and usually a debugger.
- Sometimes, an IDE is devoted to one specific programming language or one (family of) specific processor or hardware
- But more often the IDEs support multiple languages, processors, etc. Some commonly used IDEs for embedded systems are the GNU compiler collection (GCC), Eclipse, Delphi,

EDITOR:

- A source code editor is a text editor program designed specifically for editing source code to control embedded systems. It may be a standalone application or it may be built into an integrated development environment (e.g. IDE).Source code editors may have features specifically designed to simplify and speed up input of source code, such as syntax highlighting and auto complete functionality. These features ease the development of code

COMPILER:

- A compiler is a computer program that translates the source code into computer language (object code).Commonly the output has a form suitable for processing by other programs (e.g., a linker), but it may be a human readable text file. A compiler translates source code from a high level language to a lower level language (e.g., assembly language or machine language). The most common reason for wanting to translate source code is to create a program that can be executed on a computer or on an embedded system. The compiler is called a cross

compiler if the source code is

compiled to run on a platform other than the one on which the cross compiler is run. For embedded systems the compiler always runs on another platform, so a cross compiler is needed.

LINKER:

- A linker or link editor is a program that takes one or more objects generated by compilers and assembles them into a single executable program or a library that can later be linked to in it. All of the object files resulting from compiling must be combined in a special way before the program locator will produce an output file that contains a binary image that can be loaded into the target ROM. A commonly used linker/ locater for embedded systems ISLD (GNU).

TYPES OF FILES GENERATED ON CROSS COMPILATION

• The various files generated during cross compilation process are:

1. List File
2. Hex File (.hex)
3. Pre-processor Output file
4. Map File (File extension linker dependent)
5. Object File (.obj)

List Files

- At the time of cross compilation the .lst file is generated by the system which contains the information code generated from the source file.

Hex file

- The Hex file is an ASCII text file with lines of text that follow the Intel Hex file format.
- Intel Hex files are often used to transfer the program and data that would be stored in a Rom or EPROM.

Map Files

- These files are used to keep the information of linking and locating process.
- Map files use extensions .H,.HH,.HM
- Object files
- It is the lowest level file format for any platform.
    - Cross compiling each source module converts the various embedded instructions and other directives present in the module to an object (.OBJ) file.

- The object file is specially formatted file with data records for symbolic information, object code, debugging information etc.

Disassembler

- Disassembler is a utility program which converts machine codes into target processor specific Assembly instructions.
- The process of converting machine codes into Assembly code is known as “Disassembling”.
- Disassembling is complementary to assembling or cross assembling.
- The output of the disassembler is often formatted for human-read ability rather than suitability for input to an assembler.
- Assemble language source code generally permits the use of constant and programmer comments.
- These are usually removed from the assembled machine code by the assembler.
- A disassembler operating on the machine code would produce disammembly lacking these constants and comments; the disassembled output becomes more difficult for a human to interpret than the original annotated sources code.
- The interactive disassembler allows the human user to make up mnemonic symbols for values of code in an interactive session.
- A disassembler may be stand-alone or interactive.
- A stand-alone disassembler when executed generates an assembly language file which can be examined.
- Interactive shows the effect of any change the user makes immediately.

Decompiler

- Decompiler is the utility program for translating machine codes into corresponding high level language instructions.
- A decompiler is the name given to a computer program that performs the reverse operation to that of a compiler.
- The tool that accomplishes this task is called a decompiler.
- The decompiler does not reconstruct the original source code and its output is far less intelligible to a human than original source code

- Decompilation was first used in 1960s to facilitate the migration of a program form one platform to another.
- Decompile means to convert executable program code into some form of higher-level programming language so that it can be read by a human.
- Decompilation is a type of reverse engineering that does the opposite of what a compiler does.
- The are many reasons for decompilation such as understanding a program, recovering the source code for purposes of achieving or updating, finding viruses, debugging programs and translating obsolete code.

SIMULATOR

- Simulator and emulators are two important tools used in embedded system development.
- Simulator is a software tool used for simulating the various conditions for checking the functionality of the application firmware.
- It is a host-based program that simulates the functionality and instruction set of the target processor.
- The features of Simulator based debugging are:
    - Purely software based
    - Doesn't require a real target system
    - Very primitive
    - Lack of Real –time behavior.
- Simulation is used whenever trying things in the physical world would be inconvenient, expensive.
- Simulation allows experimenter to try things with more control over parameters and better insight into the results.
- Simulating an embedded computer system can be broken down into five main parts:
- The computer board itself, the piece of hardware containing one or more processor, executing the embedded software.
- The software running on the computer board. This includes the user applications, also the boot ROM or BIOS, hardware drivers, OS and various libraries.
- The communication network or networks that the board is connected to hand over which the software communicates with software on other computers.

- The environment in which the computer operates and that it measures using sensors and affects using actuators.

Advantages of Simulator Based Debugging

- The simulator based debugging techniques are simple and straightforward.
- Other advantages are: No need for original Target Board
    - It is purely software oriented.
    - IDEs software support simulates the CPU of the target board.
    - User's only needs to know about the memory map of various devices within the target board and the firmware should be written on the bases of it.
    - Since real hardware is not required the firmware development can start well in advance immediately after the device interface and memory maps are finalized.
    - This saves development time.

Simulate I/O peripherals

- It provides the option to simulate various I/O peripherals.
- Using simulator's I/O support we can edit the values for I/O registers ad can be used as the input/output value in the firmware execution.
- Hence it eliminates the need for connection I/O devices for debugging the firmware.

Simulates Abnormal Conditions

- With simulator's simulation support we can input any desired value for any parameter during debugging the firmware and can observe the control flow of firmware.
- It helps the developer in simulating abnormal operational environment for firmware and helps the firmware developer to study the behavior of the firmware under abnormal input conditions.

Limitations OF Simulator Based Debugging Deviation from Real Behavior

- Simulation-based firmware debugging is always carried out in a development environment where the developer may not be able to debug the firmware under all possible combinations of input.
- Under certain operating conditions we may get some particular result and it need notbe the same when the firmware runs in a production environment.

Lack of Real timeliness

- The major limitation is that it is not real-time in behavior.
- The debugging is developer driven and it is no way capable of creating a real time behavior.
- Moreover in a real application the I/O condition may be varying or unpredictable.

EMULATOR

- It is a piece of hardware that exactly behaves like the real microcontroller chip with all its integrated functionality.
- It is the most powerful debugging of all.
- A microcontroller's functions are emulated in real-time and non-intrusively.
- All emulators contain 3 essential function:
    - The emulator control logic, including emulation memory
    - The actual emulation device
    - A pin adapter that gives the emulator's target connector the same "package" and pin out as the microcontroller to be emulated.
- An emulator is a piece of hardware that looks like a processor, has memory like a processor, and executes instructions like a processor but it is not a processor.
- The advantage is that we can probe points of the circuit that are not accessible inside a chip.
- It is a combination of hardware and software.

DEBUGGERS

- Debugging in embedded application is the process of diagnosing the firmware execution, monitoring the target processor's registers and memory while the firmware is running.
- Debugging is classified into two namely Hardware debugging and firmware debugging.
- Hardware debugging deals with the monitoring of various bus signals and checking the status lines of the target hardware.

- Firmware debugging deals with examining the firmware execution, execution flow, changes to various CPU registers and status registers on execution of the firmware to ensure that the firmware is running as per the design.
- It is a special program used to find errors or bugs in other programs.
- A debugger allows a programmer to stop a program at any point and examine and change the values of the variables.
- A debugger or debugging tool is a computer program that is used to test and debug other programs.
- Some of the debuggers offer two modes of operation like full or partial simulation.
- A crash happens when the program cannot normally continue because of a programming bug.
- Ex- The program might have tried to use an instruction not available on the current version of the CPU to access unavailable or protected memory.
- When program crashes or reaches a preset condition the debugger shows the position in the original code if it is a source-level debugger or symbolic debugger.

CODESIGN DEFINITION AND KEY

CONCEPTS CODESIGN

- The meeting of system-level objectives by exploiting the trade-offs between hardware and software in a system through their concurrent design

Key concepts

- Concurrent: hardware and software developed at the same time on parallel paths
- Integrated: interaction between hardware and software developments to produce designs that meet performance criteria and functional specifications

Motivations for Code sign

- Factors driving codesign (hardware/software systems):
    - Instruction Set Processors (ISPs) available as cores in many design kits (386s, DSPs, microcontrollers, etc.)
    - Systems on Silicon - many transistors available in typical processes (> 10 million transistors available in IBM ASIC process, etc.)

    - Increasing capacity of field programmable devices - some devices even able to be reprogrammed on-the-fly (FPGAs, CPLDs, etc.)
    - Efficient C compilers for embedded processors
    - Hardware synthesis capabilities
- The importance of codesign in designing hardware/software systems:
    - Improves design quality, design cycle time, and cost
- Reduces integration and test time
    - Supports growing complexity of embedded systems
    - Takes advantage of advances in tools and technologies
- Processor cores
- High-level hardware synthesis capabilities
- ASIC development

Categorizing Hardware/Software Systems

- Application Domain
    - Embedded systems
- Manufacturing control
- Consumer electronics
- Vehicles
- Telecommunications
- Defense Systems
    - Instruction Set Architectures
    - Reconfigurable Systems
- Degree of programmability
    - Access to programming
    - Levels of programming
- Implementation Features
    - Discrete vs. integrated components
    - Fabrication technologies

Categories of Codesign Problems

- Codesign of embedded systems
    - Usually consist of sensors, controller, and actuators
    - Are reactive systems
    - Usually have real-time constraints
    - Usually have dependability constraints
- Codesign of ISAs
    - Application-specific instruction set processors (ASIPs)
    - Compiler and hardware optimization and trade-offs
- Codesign of Reconfigurable Systems
    - Systems that can be personalized after manufacture for a specific application
    - Reconfiguration can be accomplished before execution or concurrent with execution (called evolvable systems)

Components of the Codesign Problem

- Specification of the system
- Hardware/Software Partitioning
    - Architectural assumptions - type of processor, interface style between hardware and software, etc.
    - Partitioning objectives - maximize speedup, latency requirements; minimize size, cost, etc.
    - Partitioning strategies - high level partitioning by hand, automated partitioning using various techniques, etc.
- Scheduling
    - Operation scheduling in hardware
    - Instruction scheduling in compilers
    - Process scheduling in operating systems
- Modeling the hardware/software system during the design process

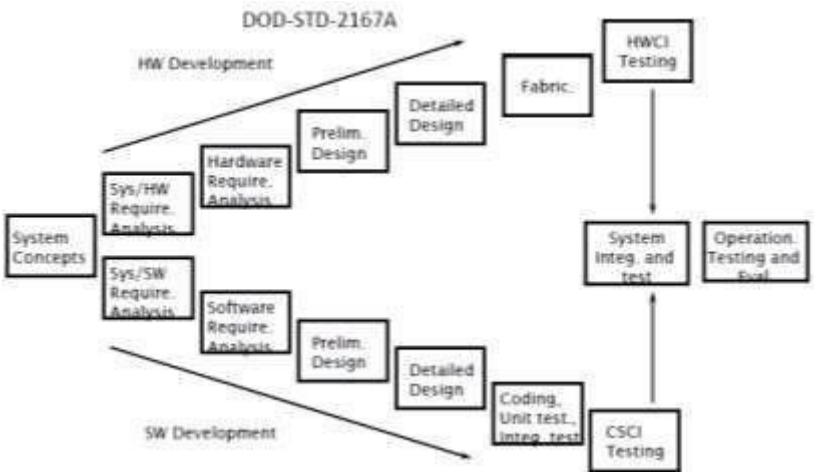


Figure: A Model of the Current Hardware/Software Design Process

Current Hardware/Software Design Process

- Basic features of current process:
  - System immediately partitioned into hardware and software components
  - Hardware and software developed separately
  - "Hardware first" approach often adopted
- Implications of these features:
  - HW/SW trade-offs restricted
- Impact of HW and SW on each other cannot be assessed easily
  - Late system integration
- Consequences these features:
  - Poor quality designs
  - Costly modifications
  - Schedule slippages

Incorrect Assumptions in Current Hardware/Software Design Process

- Hardware and software can be acquired separately and independently, with successful and easy integration of the two later
- Hardware problems can be fixed with simple software modifications
- Once operational, software rarely needs modification or maintenance

- Valid and complete software requirements are easy to state and implement in code

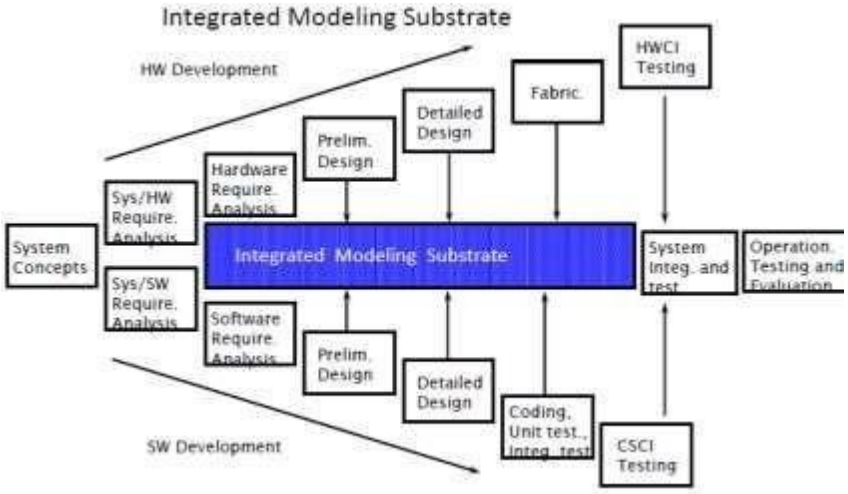


Figure: Directions of the HW/SW Design Process

Requirements for the Ideal Codesign Environment

- Unified, unbiased hardware/software representation
  - Supports uniform design and analysis techniques for hardware and software
  - Permits system evaluation in an integrated design environment
  - Allows easy migration of system tasks to either hardware or software
- Iterative partitioning techniques
  - Allow several different designs (HW/SW partitions) to be evaluated
  - Aid in determining best implementation for a system
  - Partitioning applied to modules to best meet design criteria (functionality and performance goals)
- Integrated modeling substrate
  - Supports evaluation at several stages of the design process
  - Supports step-wise development and integration of hardware and software
- Validation Methodology
  - Insures that system implemented meets initial system requirements

Cross-fertilization between Hardware and Software Design

- Fast growth in both VLSI design and software engineering has raised awareness of similarities between the two

    - Hardware synthesis
    - Programmable logic
    - Description languages
  - Explicit attempts have been made to “transfer technology” between the domains
  - EDA tool technology has been transferred to SW CAD systems
    - Designer support (not automation)
    - Graphics-driven design
    - Central database for design information
  - Tools to check design behavior early in process
- Software technology has been transferred to EDA tools
    - Single-language design
  - Use of 1 common language for architecture spec. and implementation of a chip
    - Compiler-like transformations and techniques
  - Dead code elimination
  - Loop unrolling
    - Design change management
  - Information hiding
  - Design families

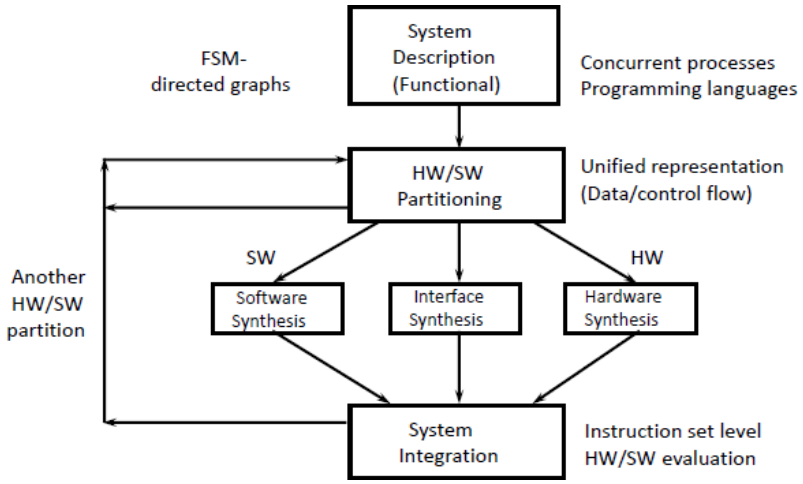


Figure: Typical Codesign Process

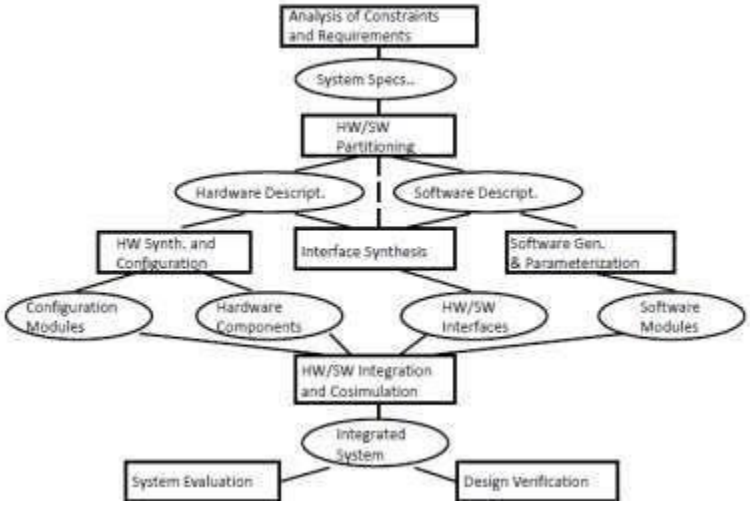


Figure: Conventional Codesign Methodology

Codesign Features

Basic features of a codesign process

- Enables mutual influence of both HW and SW early in the design cycle
    - Provides continual verification throughout the design cycle
    - Separate HW/SW development paths can lead to costly modifications and schedule slippages
- Enables evaluation of larger design space through tool interoperability and automation of codesign at abstract design levels
- Advances in key enabling technologies (e.g., logic synthesis and formal methods) make it easier to explore design tradeoffs

INTEGRATION AND TESTING OF EMBEDDED HARDWARE AND FIRMWARE.

Hardware Testing

- Hardware testing is to check/ensure the functionality, stability of hardware component and ensure that it should not have process fault. It also includes the heavy workload task for memory and CPU to check the performance and durability.
- Now a day's hardware design become much complex which demands the methods for testing to adhere and adapt to the challenges that arise, hence test development with

new standards for hardware become advance. There are many components involve in hardware testing like BIOS, CPU/Processor. Test the hardware to ensure its logical correctness and to ensure that follow appropriate standards. Using functional tests to determine whether met the test criteria. There are the few techniques commonly used for hardware testing.

  - Software-based self-testing
  - ATPG (Automatic test pattern generation)
  - BIST (Built-in self-test)

- ***The Software-Based Self-Testing:*** Modern microprocessors impose significant challenges to the testing hardware, because of their high complexity and heterogeneity. The software-based self-testing alternate way to hardware based self-testing, which cover the testing of a microprocessor using its instruction set. The benefit of software- based self test is that it can be applied in the normal operation mode of the microprocessor, thus applying the required tests at-speed.
- ***The ATPG (Automatic test pattern generation):*** Starting with a chip net list, inserting scan-chains (Scan chain is a technique used in design for testing), and generating vectors is the most direct and effortless approach and doesn't require complete knowledge of the DUT (Device under test). The recent EDA (Electronic Design Automation) tools are capable of deducing how to partition a design into blocks, and to isolate them by scan- chains.
- ***Built In Self Test (BIST):*** BIST is best for testing complex system, due to less accessibility to internal nets as design complexity increased, has spawned various design techniques that increases testability. BIST implementations are based on full scan architecture. This means that all the storage elements in the DUT concatenated to form several scan chains. These way test patterns can be serially shifted in and out of the storage elements. BIST requires no interaction with a large, expensive external test system. The testing is all built-in and only testers are needed to start the test.
- There are many tools available for hardware test and hardware diagnose like.
  - Automatic Test Equipment (ATE)
  - Sandra Lite – SiSoftware
- To execute load tests, simulate and observe variety of conditions and using or exceeding the amounts of data that could be expected in an actual situation. Following tools allows to measure different aspects of a system.

- Bonnie++
- IOZone
- Net pipe
- Linpack
- NFS Connectathon package

Firmware Testing

- Firmware is a computer program that is embedded in a hardware device that provides control, monitoring and data manipulation of engineered products and systems. The firmware contained devices provides the low-level control program for the device. Examples of devices containing firmware are embedded systems, computers, computer peripherals, mobile phones, etc.
- Importance of Firmware testing is the certification of firmware system meets its requirements with respect to functional correctness as well as performance, operational, and implementation properties. Then to reduce the risk and improve the performance.
- The firmware functionality changed from conventional instruction set emulators to more extensive and powerful instruction sets, diagnostic programs, interpreters for high level languages, and operating system functions. These are the three techniques of firmware testing.
    - Tests the micro program level that considers complete micro programs by analyzing their code or investigating the machine states after execution.
    - Tests the microinstruction level that considers single microinstructions by analyzing the assignment of micro-operations to them or investigating the machine states after execution.
    - Tests the micro-operation level that consider individual micro-operations by monitoring the execution
- The firmware testing is huge and complex task to complete, to overcome this challenge there are some automated tools available.
    - Firmware Test Suite (fwts): FWTS is a Linux tool that automates firmware checking. Tests that are designed to exercise and test different aspects of a machine's firmware – including ACPI, UEFI, hardware configuration, power- saving and so on.

UNIT-3

INTRODUCTION

- Internet of Things (IoT) is the networking of physical objects that contain electronics embedded within their architecture in order to communicate and sense interactions amongst each other or with respect to the external environment. In the upcoming years, IoT-based technology will offer advanced levels of services and practically change the way people lead their daily lives. Advancements in medicine, power, gene therapies, agriculture, smart cities, and smart homes are just a very few of the categorical examples where IoT is strongly established.
- Over 9 billion 'Things' (physical objects) are currently connected to the Internet, as of now. In the near future, this number is expected to rise to a whopping 20 billion.

There are four main components used in IoT:

1. **Low-power embedded systems:** Less battery consumption, high performance is the inverse factors play a significant role during the design of electronic systems.
2. **Cloud computing:** Data collected through IoT devices is massive and this data has to be stored on a reliable storage server. This is where cloud computing comes into play. The data is processed and learned, giving more room for us to discover where things like electrical faults/errors are within the system.
3. **Availability of big data:** We know that IoT relies heavily on sensors, especially real-time. As these electronic devices spread throughout every field, their usage is going to trigger a massive flux of big data.
4. **Networking connection:** In order to communicate, internet connectivity is a must where each physical object is represented by an IP address. However, there are only a limited number of addresses available according to the IP naming. Due to the growing number of devices, this naming system will not be feasible anymore. Therefore, researchers are looking for another alternative naming system to represent each physical object.

There are two ways of building IoT:

- Form a separate internetwork including only physical objects.
- Make the Internet ever more expansive, but this requires hard-core technologies such as rigorous cloud computing and rapid big data storage (expensive).

IoT Enablers:

1. **RFIDs:** uses radio waves in order to electronically track the tags attached to each physical object.
2. **Sensors:** devices that are able to detect changes in an environment (ex: motion detectors).
3. **Nanotechnology:** as the name suggests, these are extremely small devices with dimensions usually less than a hundred nanometers.
4. **Smart networks:** (ex: mesh topology).

CHARACTERISTICS OF IOT

1. ***Interconnectivity:*** With regard to the IoT, anything can be interconnected with the global information and communication infrastructure.
2. ***Things-related services:*** The IoT is capable of providing thing-related services within the constraints of things, such as privacy protection and semantic consistency between physical things and their associated virtual things. In order to provide thing-related services within the constraints of things, both the technologies in physical world and information world will change.
3. ***Heterogeneity:*** The devices in the IoT are heterogeneous as based on different hardware platforms and networks. They can interact with other devices or service platforms through different networks.
4. ***Dynamic changes:*** The state of devices change dynamically, e.g., sleeping and waking up, connected and/or disconnected as well as the context of devices including location and speed. Moreover, the number of devices can change dynamically.
5. ***Enormous scale:*** The number of devices that need to be managed and that communicate with each other will be at least an order of magnitude larger than the devices connected to the current Internet. Even more critical will be the management of the data generated and their interpretation for application purposes. This relates to semantics of data, as well as efficient data handling.
6. ***Safety:*** As we gain benefits from the IoT, we must not forget about safety. As both the creators and recipients of the IoT, we must design for safety. This includes the safety of our personal data and the safety of our physical well-being. Securing the endpoints, the networks, and the data moving across all of it means creating a security paradigm that will scale.

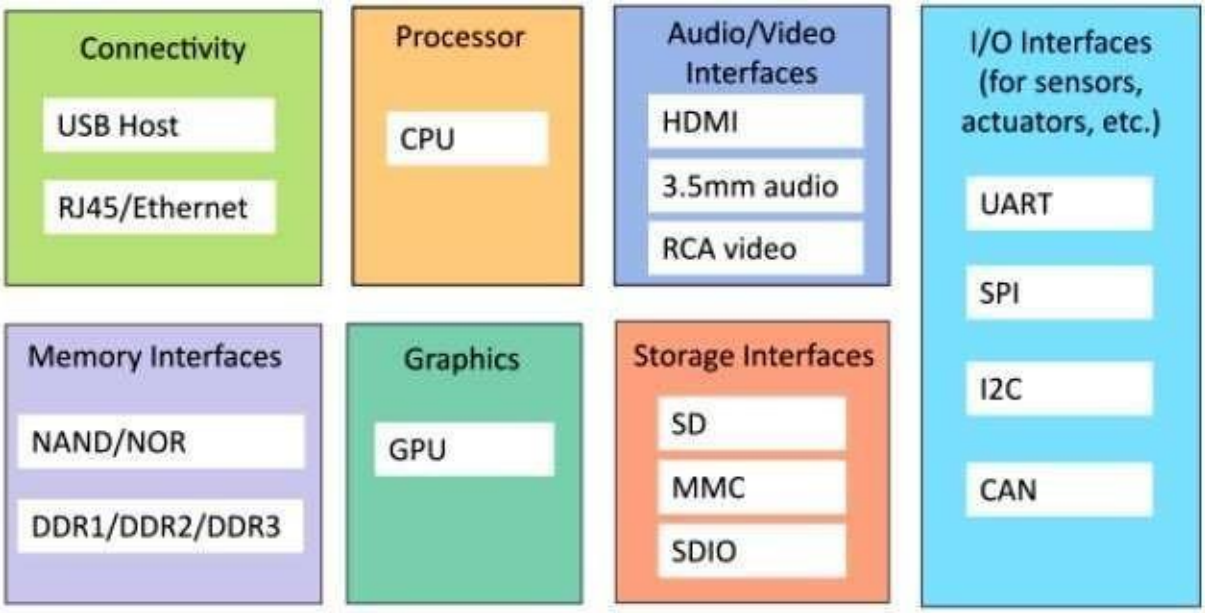
7. ***Connectivity:*** Connectivity enables network accessibility and compatibility. Accessibility is getting on a network while compatibility provides the common ability to consume and produce data.

PHYSICAL DESIGN OF IOT

1. **Things in IoT**
2. **IoT Protocols**

Things in IoT:

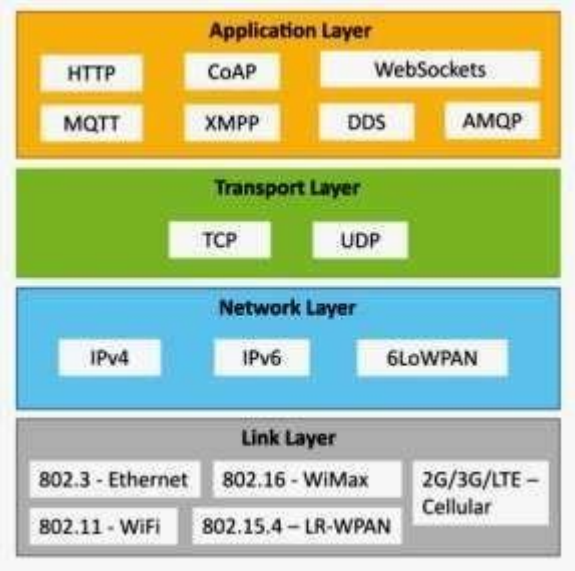
- Refers to IoT devices which have unique identities that can perform sensing, actuating and monitoring capabilities.
- IoT devices can exchange data with other connected devices or collect data from other devices and process the data either locally or send the data to centralized servers or cloud – based application back-ends for processing the data.



Generic Block Diagram of an IoT Device

- An IoT device may consist of several interfaces for connections to other devices, both wired and wireless.
- I/O interfaces for sensors
- Interfaces for internet connectivity
- Memory and storage interfaces
- Audio/video interfaces

IOT PROTOCOLS



IoT Protocols-Link Layer-Ethernet

LOGICAL DESIGN OF IOT

- In this article we discuss Logical design of Internet of things. Logical design of IoT system refers to an abstract representation of the entities & processes without going into the low-level specifies of the implementation. For understanding Logical Design of IoT, we describes given below terms.
    1. IoT Functional Blocks
    2. IoT Communication Models
    3. IoT Communication APIs

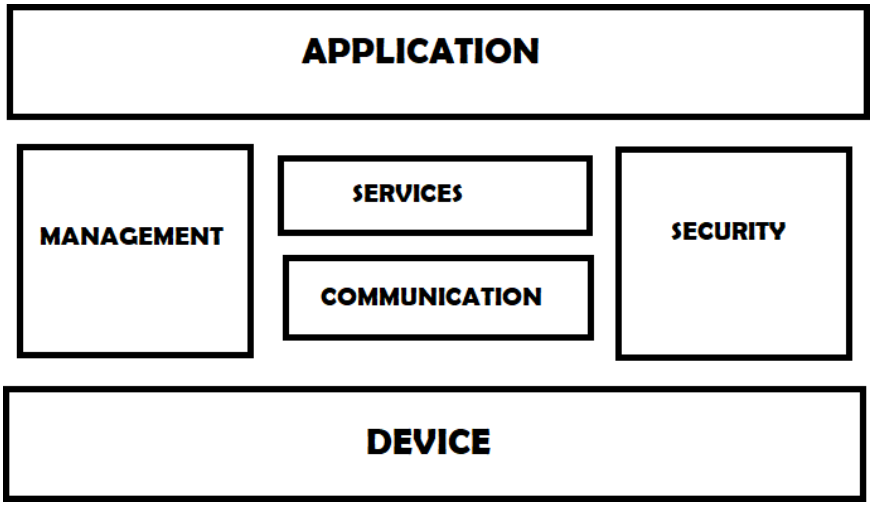
1. IoT Functional Blocks

- An IoT system comprises of a number of functional blocks that provide the system the capabilities for identification, sensing, actuation, communication and management.

Functional blocks are:

- ***Device:*** An IoT system comprises of devices that provide sensing, actuation, and monitoring and control functions.
- ***Communication:*** Handles the communication for the IoT system.
- ***Services:*** services for device monitoring, device control service, data publishing services and services for device discovery.

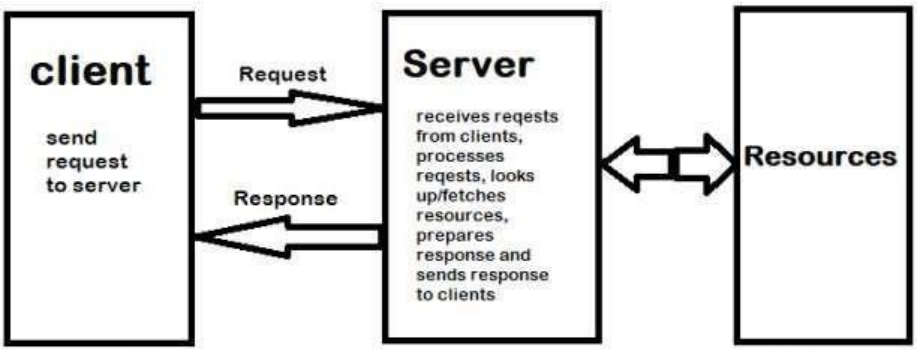
- ***Management:*** This block provides various functions to govern the IoT system.
- ***Security:*** this block secures the IoT system and by providing functions such as authentication, authorization, message and content integrity, and data security.
- ***Application:*** This is an interface that the users can use to control and monitor various aspects of the IoT system. Application also allows users to view the system status and view or analyze the processed data.



2. IoT Communication Models:

Request-Response Model

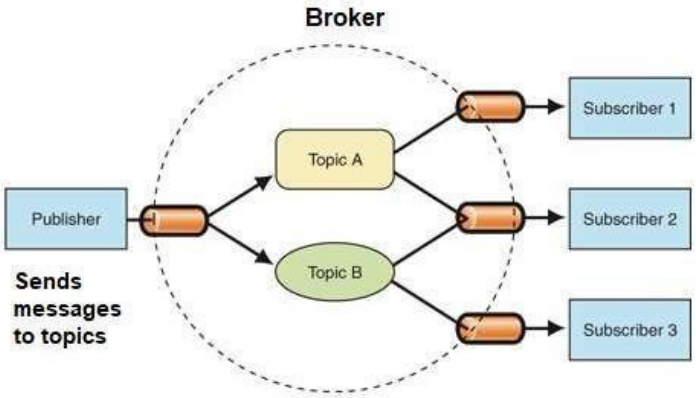
- Request-response model is communication model in which the client sends requests to the server and the server responds to the requests. When the server receives a request, it decides how to respond, fetches the data, retrieves resource representation, prepares the response, and then sends the response to the client. Request-response is a stateless communication model and each request-response pair is independent of others.
- HTTP works as a request-response protocol between a client and server. A web browser may be the client, and an application on a computer that hosts a web site may be the server.
- Example: A client (browser) submits an HTTP request to the server; then the server returns a response to the client. The response contains status information about the request and may also contain the requested content.



Request-Response Communication Model

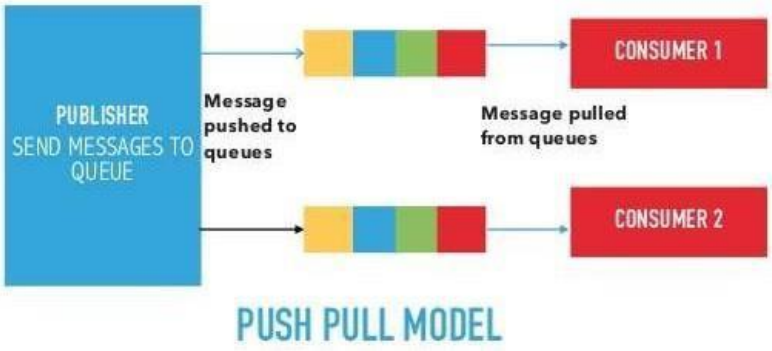
Publish-Subscribe Model

- Publish-Subscribe are a communication model that involves publishers, brokers and consumers. Publishers are the source of data. Publishers send the data to the topics which are managed by the broker. Publishers are not aware of the consumers. Consumers subscribe to the topics which are managed by the broker. When the broker receives data for a topic from the publisher, it sends the data to all the subscribed consumers.



Push-Pull Model

- Push-Pull is a communication model in which the data producers push the data to queues and the consumers pull the data from the Queues. Producers do not need to be aware of the consumers. Queues help in decoupling the messaging between the Producers and Consumers. Queues also act as a buffer which helps in situations when there is a mismatch between the rate at which the producers push data and the rate at which the consumer pull data.



PUSH PULL MODEL

Exclusive Pair Model

- Exclusive Pair is a bidirectional, fully duplex communication model that uses a persistent connection between the client and server. Connection is setup it remains open until the client sends a request to close the connection. Client and server can send messages to each other after connection setup. Exclusive pair is state full communication model and the server is aware of all the open connections.

EXCLUSIVE PAIR COMMUNICATION MODEL

IoT Communication APIs:

Generally we used Two APIs for IoT Communication. These IoT Communication APIs are:

- REST-based Communication APIs
- Web Socket-based Communication APIs

REST-based Communication APIs

Representational state transfer (REST) is a set of architectural principles by which you can design Web services the Web APIs that focus on systems resources and how resource states are addressed and transferred. REST APIs that follow the request response communication model,

the rest architectural constraint apply to the components, connector and data elements, within a distributed hypermedia system. The rest architectural constraint is as follows:

1. ***Client-server:*** The principle behind the client-server constraint is the separation of concerns. For example clients should not be concerned with the storage of data which is concern of the serve. Similarly the server should not be concerned about the user interface, which is concern of the client. Separation allows client and server to be independently developed and updated.
2. ***Stateless:*** Each request from client to server must contain all the information necessary to understand the request, and cannot take advantage of any stored context on the server. The session state is kept entirely on the client.
3. ***Cache-able:*** Cache constraints requires that the data within a response to a request be implicitly or explicitly leveled as cache-able or non cache-able. If a response is cache- able, then a client cache is given the right to reuse that response data for later, equivalent requests. Caching can partially or completely eliminate some instructions and improve efficiency and scalability.
4. ***Layered system:*** layered system constraints, constrains the behavior of components such that each component cannot see beyond the immediate layer with they are interacting. For example, the client cannot tell whether it is connected directly to the end server or two an intermediary along the way. System scalability can be improved by allowing intermediaries to respond to requests instead of the end server, without the client having to do anything different.
5. ***Uniform interface:*** uniform interface constraints require that the method of communication between client and server must be uniform. Resources are identified in the requests (by URIs in web based systems) and are themselves is separate from the representations of the resources data returned to the client. When a client holds a representation of resources it has all the information required to update or delete the resource you (provided the client has required permissions). Each message includes enough information to describe how to process the message.
6. ***Code on demand:*** Servers can provide executable code or scripts for clients to execute in their context. This constraint is the only one that is optional.

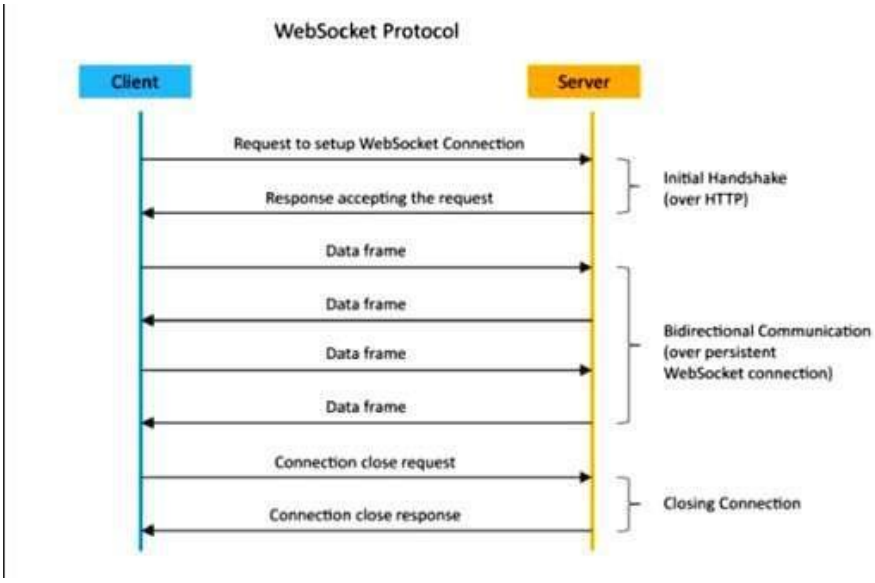
A Restful web service is a" Web API" implemented using HTTP and REST principles. REST is most popular IoT Communication APIs.

HTTP methods

Uniform Resource Identifier (URI)	GET	PUT	PATCH	POST	DELETE
Collection, such as https://api.example.com/resources/	List the URIs and perhaps other details of the collection's members.	Replace the entire collection with another collection.	Not generally used	Create a new entry in the collection. The new entry's URI is assigned automatically and is usually returned by the operation.	Delete the entire collection.
Element, such as https://api.example.com/resources/item5	Retrieve a representation of the addressed member of the collection, expressed in an appropriate Internet media type.	Replace the addressed member of the collection, or if it does not exist, create it.	Update the addressed member of the collection.	Not generally used. Treat the addressed member as a collection in its own right and create a new entry within it.	Delete the addressed member of the collection.

web Socket based communication API

Web socket APIs allows bi-directional, full duplex communication between clients and servers. Web socket APIs follows the exclusive pair communication model. Unlike request-response model such as REST, the Web Socket APIs allow full duplex communication and do not require new connection to be setup for each message to be sent. Web socket communication begins with a connection setup request sent by the client to the server. The request (called web socket handshake) is sent over HTTP and the server interprets it is an upgrade request. If the server supports web socket protocol, the server responds to the web socket handshake response. After the connection setup client and server can send data/messages to each other in full duplex mode. Web socket API reduces the network traffic and latency as there is no overhead for connection setup and termination requests for each message. Web socket suitable for IoT applications that have low latency or high throughput requirements. So Web socket is most suitable IoT Communication APIs for IoT System.



IOT ENABLING TECHNOLOGIES

Wireless Sensor Networks

A wireless sensor network comprises of distributed device with sensor which are used to monitor the environmental and physical conditions. A WSN consists of a number of end-nodes and routers and a coordinator. End Nodes have several sensors attached to them in node can also act as routers. Routers are responsible for routing the data packets from end-nodes to the coordinator. The coordinator collects the data from all the nodes. Coordinator also acts as a gateway that connects the WSN to the internet. Some examples of WSNs used in IoT systems are described as follows:

- Weather monitoring system use WSNs in which the nodes collect temperature humidity and other data which is aggregated and analyzed.
- Indoor air quality monitoring systems use WSNs to collect data on the indoor air quality and concentration of various gases
- Soil moisture monitoring system use WSNs to monitor soil moisture at various locations.
- Surveillance system use WSNs for collecting Surveillance data (such as motion detection data)
- Smart grid use WSNs for monitoring the grid at various points.
- Structural health monitoring system use WSNs to monitor the health of structures (buildings, bridges) by collecting vibration data from sensor nodes de deployed at various points in the structure.

You may like also:

- LiteOS: an IoT operating system and middleware
- Contiki OS: The Open Source OS for IoT

Cloud Computing

- Cloud computing is a trans-formative computing paradigm that involves delivering applications and services over the Internet Cloud computing involves provisioning of computing, networking and storage resources on demand and providing these resources as metered services to the users, in a “pay as you go” model. C loud computing resources can be provisioned on demand by the users, without requiring interactions with the cloud service Provider. The process of provisioning resources is automated. Cloud computing resources can be accessed over the network using standard access mechanisms that provide platform independent access through the use of heterogeneous client platforms such as the workstations, laptops, tablets and smart phones.

Cloud computing services are offered to users in different forms:

- ***Infrastructure as a Service (IaaS):*** hardware is provided by an external provider and managed for you
- ***Platform as a Service (PaaS):*** in addition to hardware, your operating system layer is managed for you
- ***Software as a Service (SaaS):*** further to the above, an application layer is provided and managed for you – you won’t see or have to worry about the first two layers.

Big Data Analytics

- Big Data analytics is the process of collecting, organizing and analyzing large sets of data (called Big Data) to discover patterns and other useful information. Big Data analytics can help organizations to better understand the information contained within the data and will also help identify the data that is most important to the business and future business decisions. Analysts working with Big Data typically want the knowledge that comes from analyzing the data.

Some examples of big data generated by IoT systems are described as follows:

- Sensor data generated by IoT system such as weather monitoring stations.
- Machine sensor data collected from sensors embedded in industrial and energy systems for monitoring their health and detecting Failures.
- Health and fitness data generated by IoT devices such as wearable fitness bands
- Data generated by IoT systems for location and tracking of vehicles
- Data generated by retail inventory monitoring systems
- **Characteristics**
- Big data can be described by the following characteristics:
- **Volume:** The quantity of generated and stored data. The size of the data determines the value and potential insight and whether it can be considered big data or not.
- **Variety:** The type and nature of the data. This helps people who analyze it to effectively use the resulting insight. Big data draws from text, images, audio, video; plus it completes missing pieces through data fusion.
- **Velocity:** In this context, the speed at which the data is generated and processed to meet the demands and challenges that lie in the path of growth and development. Big data is often available in real-time. Compared to small data, big data are produced more continually. Two kinds of velocity related to Big Data are the frequency of generation and the frequency of handling, recording, and publishing.
- **Veracity:** It is the extended definition for big data, which refers to the data quality and the data value. The data quality of captured data can vary greatly, affecting the accurate analysis.

Communication protocols

- Communication protocols form the backbone of IoT systems and enable network connectivity and coupling to applications. Communication protocols allow devices to exchange data over the network. Multiple protocols often describe different aspects of a single communication. A group of protocols designed to work together are known as a protocol suite; when implemented in software they are a protocol stack.
- Internet communication protocols are published by the Internet Engineering Task Force (IETF). The IEEE handles wired and wireless networking, and the International Organization for Standardization (ISO) handles other types. The ITU-T handles telecommunication protocols and formats for the public switched telephone network (PSTN). As the PSTN and Internet converge, the standards are also being driven towards convergence.
- In IoT we used MQTT, COAP, AMQP etc. protocols. You can read in detail by given below links.
- **You may like also:**
    - IoT Data Protocols
    - Wireless IoT Network Protocols
    - IoT Open Source Development Tools

Embedded Systems:

As its name suggests, Embedded means something that is attached to another thing. An embedded system can be thought of as a computer hardware system having software embedded in it. An embedded system can be an independent system or it can be a part of a large system. An embedded system is a controller programmed and controlled by a real-time operating system (RTOS) with a dedicated function within a larger mechanical or electrical system, often with real-time computing constraints. It is embedded as part of a complete device often including hardware and mechanical parts. Embedded systems control many devices in common use today. Ninety-eight percent of all microprocessors are manufactured to serve as embedded system component.

An embedded system has three components:

- It has hardware.
- It has application software.
- It has Real Time Operating system (RTOS) that supervises the application software and provide mechanism to let the processor run a process as per scheduling by following a

plan to control the latencies. RTOS defines the way the system works. It sets the rules during the execution of application program. A small scale embedded system may not have RTOS.

IOT LEVELS

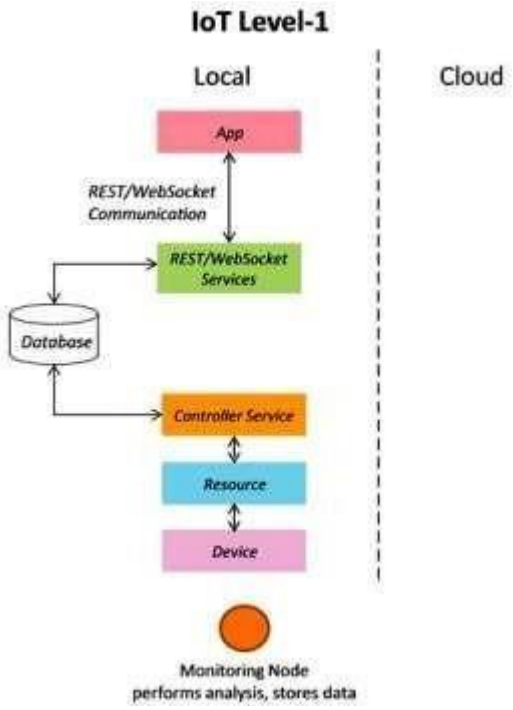
An IoT system comprises the following components: **Device, Resource, Controller Service, Database, and Web service, Analysis, Component and Application.**

- ***Device:*** An IoT device allows identification, remote sensing, and remote monitoring capabilities.
- ***Resource:***
    - Software components on the IoT device for
        - accessing, processing and storing sensor information,
        - Controlling actuators connected to the device.
        - Enabling network access for the device.
- ***Controller Service:*** Controller service is a native service that runs on the device and interacts with the web services. It sends data from the device to the web service and receives commands from the application (via web services) for controlling the device.
- ***Database:*** Database can be either local or in the cloud and stores the data generated by the IoT device.
- ***Web Service:*** Web services serve as a link between the IoT device, application, and database and analysis components. It can be implemented using HTTP and REST principles (REST service) or using the Web Socket protocol (Web Socket service).
- ***Analysis Component:*** Analysis Component is responsible for analyzing the IoT data and generating results in a form that is easy for the user to understand.
- ***Application:*** IoT applications provide an interface that the users can use to control and monitor various aspects of the IoT system. Applications also allow users to view the system status and the processed data.

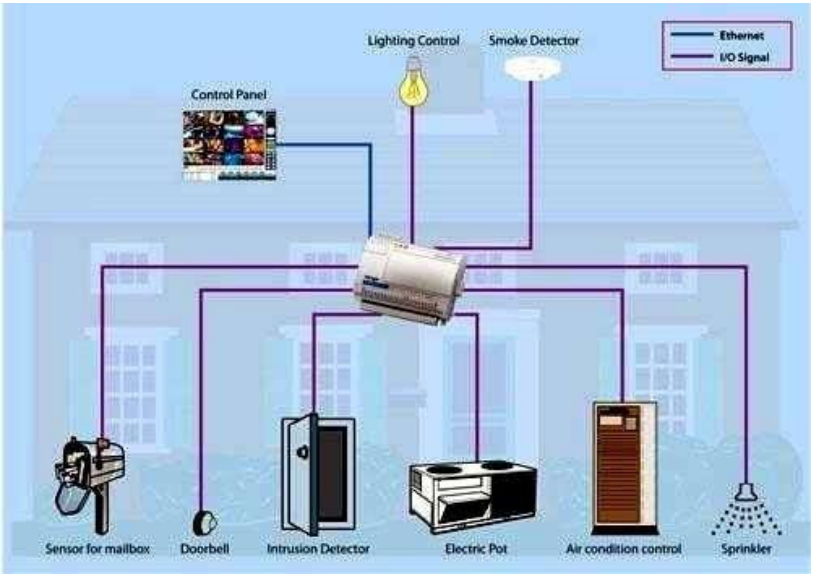
IoT Level-1

- A level-1 IoT system has a single node/device that performs sensing and/or actuation, stores data, performs analysis and hosts the application.

- Level-1 IoT systems are suitable for modeling low- cost and low-complexity solutions where the data involved is not big and the analysis requirements are not computationally intensive.

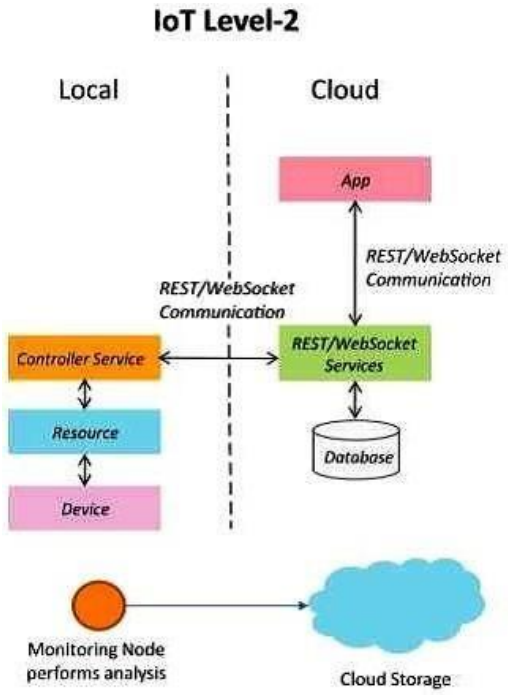


- IoT – Level 1 Example : Home Automation System



IoT Level-2

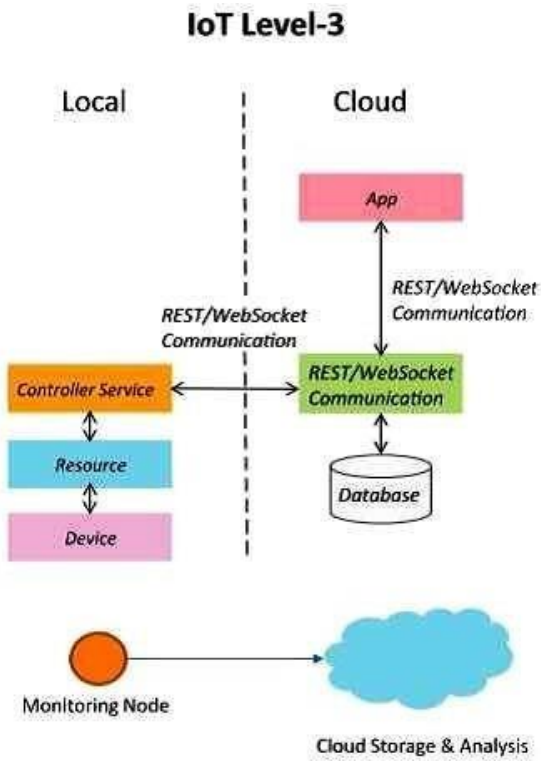
- A level-2 IoT system has a single node that performs sensing and/or actuation and local analysis. Data is stored in the cloud and the application is usually cloud-based.
- Level-2 IoT systems are suitable for solutions where the data involved is big; however, the primary analysis requirement is not computationally intensive and can be done locally



- **IoT – Level 2 Example: Smart Irrigation**

IoT Level-3

- A level-3 IoT system has a single node. Data is stored and analyzed in the cloud and the application is cloud based.
- Level-3 IoT systems are suitable for solutions where the data involved is big and the analysis requirements are computationally intensive.



- IoT – Level 3 Example: Tracking Package Handling

Sensors used

Accelerometer sense movement or

Gyroscope

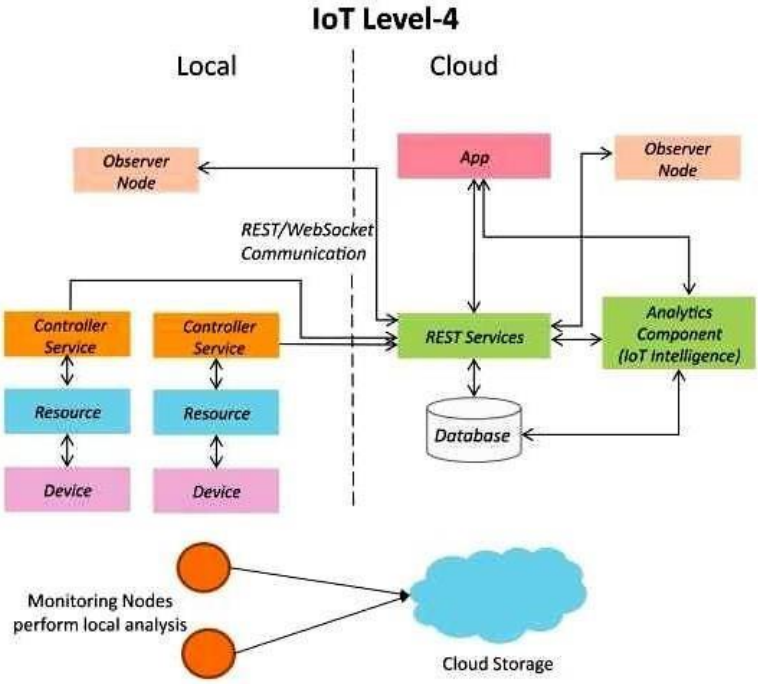
Gives orientation info

Vibrations

- Web socket service is used because sensor data can be sent in real time.

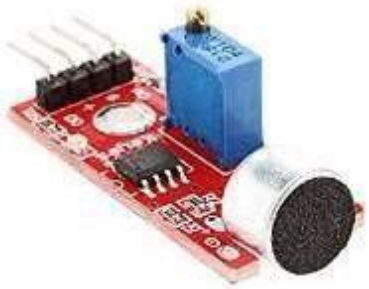
IoT Level-4

- A level-4 IoT system has multiple nodes that perform local analysis. Data is stored in the cloud and the application is cloud-based.
- Level-4 contains local and cloud based observer nodes which can subscribe and receive information collected in the cloud from IoT devices.
- Level-4 IoT systems are suitable for solutions where multiple nodes are required, the data involved is big and the analysis requirements are computationally intensive.



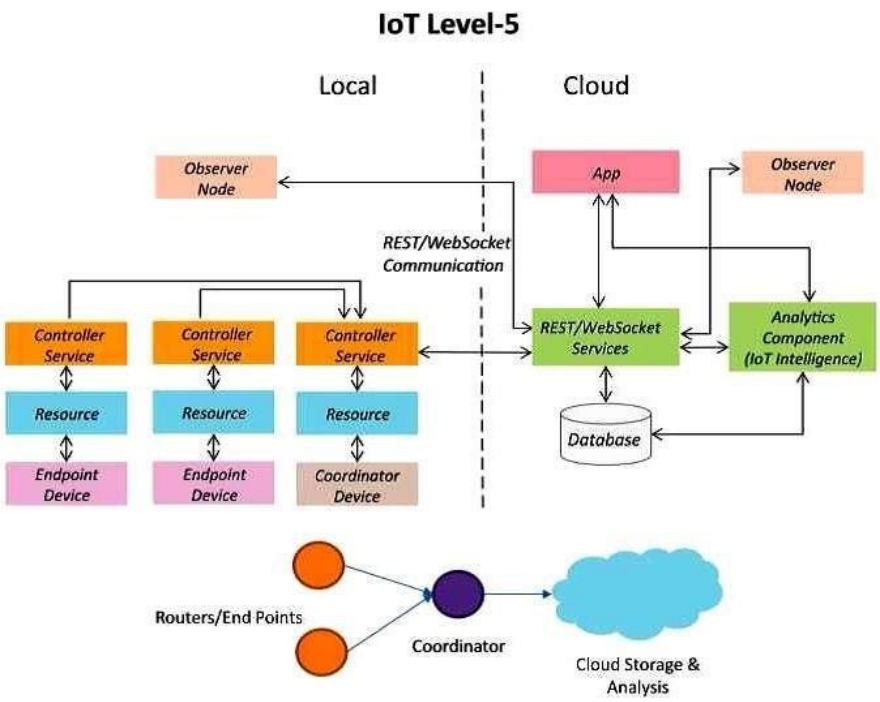
- IoT – Level 4 Example: Noise Monitoring

Sound Sensors are used

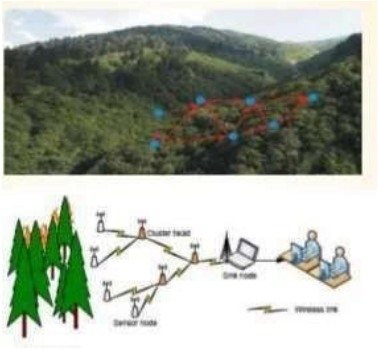


IoT Level-5

- A level-5 IoT system has multiple end nodes and one coordinator node.
- The end nodes perform sensing and/or actuation.
- The coordinator node collects data from the end nodes and sends it to the cloud.
- Data is stored and analyzed in the cloud and the application is cloud- based.
- Level-5 IoT systems are suitable for solutions based on wireless sensor networks, in which the data involved is brigand the analysis requirements are computationally intensive.



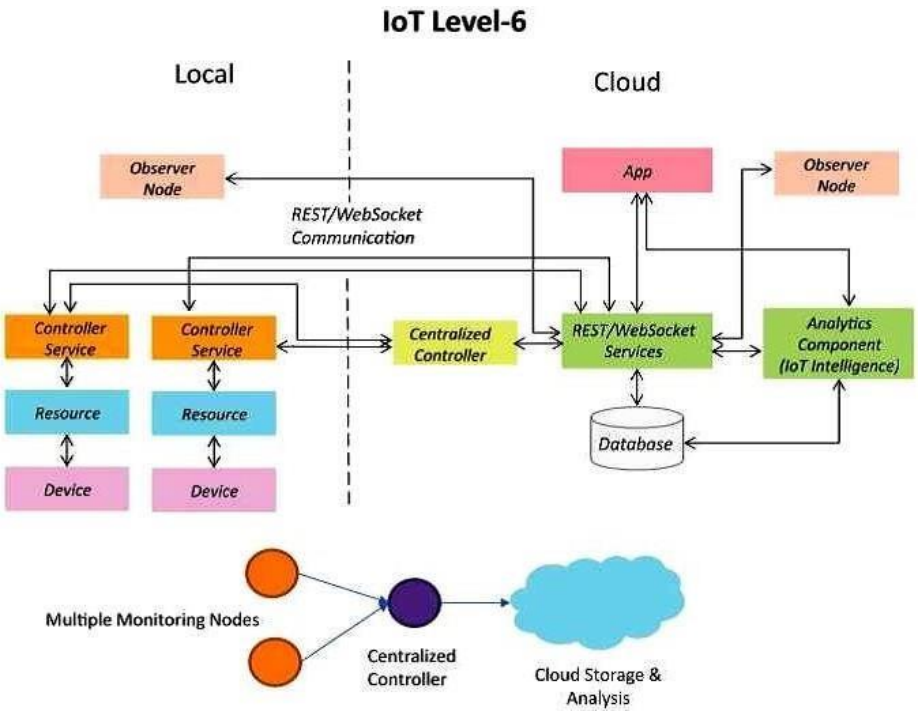
- IoT – Level 5 Example: Forest Fire Detection
- Detect forest fire in early stages to take action while the fire is still controllable. Sensors measure the temperature, smoke, weather, slope of the earth, wind speed, speed of fire spread, flame length



IoT Level-6

- A level-6 IoT system has multiple independent end nodes that perform sensing and/or actuation and send data to the cloud.
- Data is stored in the cloud and the application is cloud-based.
- The analytics component analyzes the data and stores the results in the cloud database.
- The results are visualized with the cloud-based application.

- The centralized controller is aware of the status of all the end nodes and sends control commands to the nodes.



- IoT – Level 6 Example: Weather Monitoring System
- ***Sensors used :*** Wind speed and direction, Solar radiation, Temperature (air, water, soil), Relative humidity, Precipitation, Snow depth, Barometric pressure, Soil moisture

DOMAIN SPECIFIC IOTS

IoT Applications for:

1. Home
2. Cities
3. Environment
4. Energy Systems
5. Retail
6. Logistics
7. Industry
8. Agriculture
9. Health & Lifestyle

1. Home Automation

IoT applications for smart homes: a). Smart Lighting

b). Smart Appliances c). Intrusion Detection

d). Smoke / Gas Detectors

a). Smart Lighting

- Smart lighting achieves energy savings by sensing the human movements and their environments and controlling the lights accordingly.
- Key enabling technologies for smart lighting include :
  - Solid state lighting (such as LED lights)
  - IP-enabled lights
- Wireless-enabled and Internet connected lights can be controlled remotely from IoT applications such as a mobile or web application.
- Paper: Energy-aware wireless sensor network with ambient intelligence for smart LED lighting system control [IECON, 2011] presented controllable LED lighting system that is embedded with ambient intelligence gathered from a distributed smart WSN to optimize and control the lighting system to be more efficient and user-oriented.

b). Smart Appliances

- Smart appliances make the management easier and provide status information of appliances to the users remotely. E.g.: smart washer/dryer that can be controlled remotely and notify when the washing/drying cycle is complete.
- Open Remote is an open source automation platform for smart home and building that can control various appliances using mobile and web applications.
- It comprises of three components:
    - A Controller: manages scheduling and runtime integration between devices.
    - A Designer: allows creating both configurations for the controller and user interface designs.
    - Control Panel: allows interacting with devices and controlling them.
- Paper: An IoT-based Appliance Control System for Smart Home [ICICIP, 2013] [implemented an IoT based appliance control system for smart homes that uses a smart- central controller to set up a wireless sensor and actuator network and control modules for appliances.

c). Intrusion Detection

- Home intrusion detection systems use security cameras and sensors to detect intrusions and raise alerts.
- The form of the alerts can be in form:
    - SMS
    - Email
    - Image grab or a short video clip as an email attachment
- Papers :
    - Could controlled intrusion detection and burglary prevention stratagems in home automation systems [BCFIC, 2012] present a controlled intrusion detection system that uses location-aware services, where the geo-location of each node of a home automation system is independently detected and stored in the cloud?
    - An Intelligent Intrusion Detection System Based on UPnP Technology for Smart Living [ISDA, 2008] implement an intrusion detection system that uses image processing to recognize the intrusion and extract the intrusion subject and generate Universal-Plug-and-Play (UPnP-based) instant messaging for alerts.

d). Smoke / Gas Detectors

- Smoke detectors are installed in homes and buildings to detect smoke that is typically an early sign of fire.
- It uses optical detection, ionization or air sampling techniques to detect smoke
- The form of the alert can be in form :
  - Signals that send to a fire alarm system
- Gas detector can detect the presence of harmful gases such as carbon monoxide (CO), liquid petroleum gas (LPG), etc.
- Paper: Development of Multipurpose Gas Leakage and Fire Detector with Alarm System [TIIEC, 2013] designed a system that can detect gas leakage and smoke and gives visual level indication.

2. Cities

- IoT applications for smart cities: a). Smart Parking b). Smart Lighting for Road c). Smart Road d). Structural Health Monitoring e). Surveillance f). Emergency Response

a). Smart Parking

- Finding the parking space in the crowded city can be time consuming and frustrating
- Smart parking makes the search for parking space easier and convenient for driver.
- It can detect the number of empty parking slots and send the information over the Internet to the smart parking applications which can be accessed by the drivers using their smart phones, tablets, and in car navigation systems.
- Sensors are used for each parking slot to detect whether the slot is empty or not, and this information is aggregated by local controller and then sent over the Internet to database.

- Paper :
  - Design and implementation of a prototype Smart Parking (SPARK) system using WSN [International Conference on Advanced Information Networking and Applications Workshop, 2009] designed and implemented a prototype smart parking system based on wireless sensor network technology with features like remote parking monitoring, automate guidance, and parking reservation mechanism.

b). Smart Lighting for Roads

- It can help in saving energy
- Smart lighting for roads allows lighting to be dynamically controlled and also adaptive to ambient conditions.
- Smart light connected to the Internet can be controlled remotely to configure lighting schedules and lighting intensity.
- Custom lighting configurations can be set for different situations such as a foggy day, a festival, etc.
- Paper :
  - Smart Lighting solutions for Smart Cities [International Conference on Advance Information Networking and Applications Workshop, 2013] described the need for smart lighting system in smart cities, smart lighting features and how to develop interoperable smart lighting solutions.

c). Smart Roads

- Smart Roads provides information on driving conditions, travel time estimates and alerts in case of poor driving conditions, traffic congestions and accidents.
- Such information can help in making the roads safer and help in reducing traffic jams
- Information sensed from the roads can be communicated via internet to cloud-based applications and social media and disseminated to the drivers who subscribe to such applications.
- Paper:
  - Sensor networks for smart roads [PerCom Workshop, 2006] proposed a distributed and autonomous system of sensor network nodes for improving driving safety on public roads, the system can provide the driver and passengers with a consistent view of the road situation a few hundred meters ahead of them or a few dozen miles away, so that they can react to potential dangers early enough.

d). Structural Health Monitoring

- It uses a network of sensors to monitor the vibration levels in the structures such as bridges and buildings.
- The data collected from these sensors is analyzed to assess the health of the structures.
- By analyzing the data it is possible to detect cracks and mechanical breakdowns, locate the damages to a structure and also calculate the remaining life of the structure.
- Using such systems, advance warnings can be given in the case of imminent failure of the structure.
- Paper:
  - Environmental Effect Removal Based Structural Health Monitoring in the Internet of Things [International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, 2013] proposed an environmental effect removal based structural health monitoring scheme in an IoT environment.
  - Energy harvesting technologies for structural health monitoring applications [IEEE Conference on Technologies for Sustainability, 2013] Explored energy harvesting technologies of harvesting ambient energy, such as mechanical vibrations, sunlight, and wind.

e). Surveillance

- Surveillance of infrastructure, public transport and events in cities is required to ensure safety and security.
- City wide surveillance infrastructure comprising of large number of distributed and Internet connected video surveillance cameras can be created.
- The video feeds from surveillance cameras can be aggregated in cloud-based scalable storage solutions.
- Cloud-based video analytics applications can be developed to search for patterns of specific events from the video feeds.

f). Emergency Response

- IoT systems can be used for monitoring the critical infrastructure cities such as buildings, gas, and water pipelines, public transport and power substations.
- IoT systems for critical infrastructure monitoring enable aggregation and sharing of information collected from lager number of sensors.

- Using cloud-based architectures, multi-modal information such as sensor data, audio, video feeds can be analyzed I near real-time to detect adverse events.
- The alert can be in the form :
  - Alerts sent to the public
  - Re-rerouting of traffic
  - Evacuations of the affected areas

3. Environment

IoT applications for smart environments: a). Weather Monitoring b). Air Pollution Monitoring c). Noise Pollution Monitoring d). Forest Fire Detection e). River Flood Detection

a). Weather Monitoring

- It collects data from a number of sensors attached such as temperature, humidity, pressure, etc and sends the data to cloud-based applications and store back-ends.
- The data collected in the cloud can then be analyzed and visualized by cloud-based applications.
- Weather alert can be sent to the subscribed users from such applications.
- AirPi is a weather and air quality monitoring kit capable of recording and uploading information about temperature, humidity, air pressure, light levels, UV levels, carbon monoxide, nitrogen dioxide and smoke level to the Internet.
- Paper:
  - PeWeMoS – Pervasive Weather Monitoring System [ICPCA, 2008] Presented a pervasive weather monitoring system that is integrated with buses to measure weather variables like humidity, temperature, and air quality during the bus path

b). Air Pollution Monitoring

- IoT based air pollution monitoring system can monitor emission of harmful gases by factories and automobiles using gaseous and meteorological sensors.
- The collected data can be analyzed to make informed decisions on pollutions control approaches.
- Paper:
  - Wireless sensor network for real-time air pollution monitoring [ICCSPA, 2013] [Presented a real time air quality monitoring system that comprises of several distributed monitoring stations that communicate via wireless with a backend server using machine-to machine communication.

c). Noise Pollution Monitoring

- Noise pollution monitoring can help in generating noise maps for cities.
- It can help the policy maker in making policies to control noise levels near residential areas, school and parks.
- It uses a number of noise monitoring stations that are deployed at different places in a city.
- The data on noise levels from the stations is collected on servers or in the cloud and then the collected data is aggregate to generate noise maps.
- Papers :
  - Noise mapping in urban environments: Applications at Suez city center [ICCIE, 2009] presented a noise mapping study for a city which revealed that the city suffered from serious noise pollution.
  - Sound Of City – Continuous noise monitoring for a health city [PerComW,2013] [Designed a Smartphone application that allows the users to continuously measure noise levels and send to a central server here all generated information is aggregated and mapped to a meaningful noise visualization map.

d). Forest Fire Detection

- IoT based forest fire detection system use a number of monitoring nodes deployed at different location in a forest.
- Each monitoring node collects measurements on ambient condition including temperature, humidity, light levels, etc.
- Early detection of forest fires can help in minimizing the damage.
- Papers:

- A novel accurate forest fire detection system using wireless sensor networks [International Conference on Mobile Adhoc and Sensor Networks, 2011] Presented

  a forest fire detection system based on wireless sensor network. The system uses multi-criteria detection which is implemented by the artificial neural network. The ANN fuses sensing data corresponding to, multiple attributes of a forest fire such as temperature, humidity, infrared and visible light to detect forest fires.

e). River Flood Detection

- IoT based river flood monitoring system uses a number of sensor nodes that monitor the water level using ultrasonic sensors and flow rate using velocity sensors.
- Data from these sensors is aggregated in a server or in the cloud, monitoring applications raise alerts when rapid increase in water level and flow rate is detected.
- Papers:
  - RFMS : Real time flood monitoring system with wireless sensor networks [MASS, 2008] Described a river flood monitoring system that measures river and weather conditions through wireless sensor nodes equipped with different sensors
  - Urban Flash Flood Monitoring, Mapping and Forecasting via a Tailored Sensor Network System [ICNSC, 2006] Described a motes-based sensor network for river flood monitoring that includes a water level monitoring module, network video recorder module, and data processing module that provides floods information n the form of raw data, predict data, and video feed.

4. Energy

IoT applications for smart energy

systems: a). Smart Grid

b). Renewable Energy

Systems c). Prognostics

a). Smart Grids

- Smart grid technology provides predictive information and recommendation s to utilize, their suppliers, and their customers on how best to manage power.
- Smart grid collect the data regarding :
  - Electricity generation

- Electricity consumption
- Storage
- Distribution and equipment health data

- By analyzing the data on power generation, transmission and consumption of smart grids can improve efficiency throughout the electric system. Storage collection and analysis of smarts grids data in the cloud can help in dynamic optimization of system operations, maintenance, and planning.
- Cloud-based monitoring of smart grids data can improve energy usage levels via energy feedback to users coupled with real-time pricing information.
- Condition monitoring data collected from power generation and transmission systems can help in detecting faults and predicting outages.

b). Renewable Energy System

- Due to the variability in the output from renewable energy sources (such as solar and wind), integrating them into the grid can cause grid stability and reliability problems.
- IoT based systems integrated with the transformer at the point of interconnection measure the electrical variables and how much power is fed into the grid
- To ensure the grid stability, one solution is to simply cut off the overproductions.
- Paper:
  - Communication systems for grid integration of renewable energy resources [IEEE Network, 2011] -provided the closed-loop controls for wind energy system that can be used to regulate the voltage at point of interconnection which coordinate wind turbine outputs and provides reactive power support.

c). Prognostics

- IoT based prognostic real-time health management systems can predict performance of machines of energy systems by analyzing the extent of deviation of a system from its normal operating profiles.
- In the system such as power grids, real time information is collected using specialized electrical sensors called Phasor Measurement Units (PMU)
- Analyzing massive amounts of maintenance data collected from sensors in energy systems and equipment can provide predictions for impending failures.
- OpenPDC is a set of applications for processing of streaming time-series data collected from Phasor Measurements Units (PMUs) in real-time.

5. Retail

IoT applications in smart retail systems: a). Inventory Management

b). Smart Payments

c). Smart Vending Machines

a). Inventory Management

- IoT system using Radio Frequency Identification (RFID) tags can help inventory management and maintaining the right inventory levels.
- RFID tags attached to the products allow them to be tracked in the real-time so that the inventory levels can be determined accurately and products which are low on stock can be replenished.
- Tracking can be done using RFID readers attached to the retail store shelves or in the warehouse.
- Paper:
  - RFID data-based inventory management of time-sensitive materials [IECON, 2005] described an RFID data-based inventory management system for time-sensitive materials

b). Smart Payments

- Smart payments solutions such as contact-less payments powered technologies such as near field communication (NFC) and Bluetooth.
- NFC is a set of standards for smart-phones and other devices to communicate with each other by bringing them into proximity or by touching them
- Customer can store the credit card information in their NFC-enabled smart-phones and make payments by bringing the smart-phone near the point of sale terminals.
- NFC maybe used in combination with Bluetooth, where NFC initiates initial pairing of devices to establish a Bluetooth connection while the actual data transfer takes place over Bluetooth.

c). Smart Vending Machines

- Smart vending machines connected to the Internet allow remote monitoring of inventory levels, elastic pricing of products, promotions, and contact-less payments using NFC.
- Smart-phone applications that communicate with smart vending machines allow

user preferences to be remembered and learned with time. E.g.: when a user moves from one vending machine to the other and pair the smart-phone, the user preference and favorite product will be saved and then that data is used for predictive maintenance.