JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY: KAKINADA
KAKINADA – 533 003, Andhra Pradesh, India
DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

III Year - II Semester		L	T	P	C
		3	0	0	3

INTERNET OF THINGS

Course Objectives:
- To learn and understand elements of IoTsystem.
- Acquire knowledge about various protocols ofIoT.
- To learn and understand design principles and capabilities ofIoT.

UNIT I: Introduction to IoT
Introduction to IoT, Architectural Overview, Design principles and needed capabilities, Basics of Networking, M2M and IoT Technology Fundamentals- Devices andgateways, Data management, Business processes in IoT, Everything as a Service (XaaS), Role ofCloud in IoT, Security aspects inIoT.

UNIT II: Elements of IoT
Hardware Components- Computing- Arduino, Raspberry Pi, ARM Cortex-A class processor, Embedded Devices – ARM Cortex-M class processor, Arm Cortex-M0 Processor Architecture, Block Diagram, Cortex-M0 Processor Instruction Set, ARM and Thumb Instruction Set.

UNIT III: IoT Application Development
Communication, IoT Applications, Sensing, Actuation, I/O interfaces.
Software Components- Programming API's (using Python/Node.js/Arduino) for CommunicationProtocols-MQTT, ZigBee, CoAP, UDP, TCP, Bluetooth.
Bluetooth Smart Connectivity
Bluetooth overview, Bluetooth Key Versions, Bluetooth Low Energy (BLE) Protocol, Bluetooth, Low Energy Architecture, PSoC4 BLE architecture and Component Overview.

UNIT IV: Solution framework for IoT applications
Implementation of Device integration, Data acquisitionand integration, Device data storage-Unstructured data storage on cloud/local server,Authentication, authorization of devices.

UNIT V: IoT Case Studies
IoT case studies and mini projects based on Industrial automation, Transportation, Agriculture,Healthcare, HomeAutomation.

Text Books:
1. Raj Kamal, "Internet of Things: Architecture and Design Principles", 1st Edition, McGraw Hill Education,2017.
2. The Definitive Guide to the ARM Cortex-M0 by JosephYiu,2011
3. Vijay Madisetti, ArshdeepBahga, Internet of Things, "A Hands on Approach", UniversityPress,2015.

References:

1. Cypress Semiconductor/PSoC4 BLE (Bluetooth Low Energy) Product TrainingModules.
2. Pethuru Raj and Anupama C. Raman, "The Internet of Things: EnablingTechnologies, Platforms, and Use Cases", CRC Press,2017.

Course Outcomes:

The student will be able to:

- Understand internet of Things and its hardware and softwarecomponents.
- Interface I/O devices, sensors &communicationmodules.
- Remotely monitor data and controldevices.
- Design real time IoT basedapplications

IOT

UNIT-1

INTERNET OF THINGS: Internet of Things (IoT) is a concept which enables communication between internetworking devices and applications, whereby physical objects or 'things' communicate through the Internet.

The concept of IoT beganwith things classified as identity communication devices. Radio Frequency Identification Device (RFID) is an example of an identity communication device. Things are tagged to these devices for their identification in future and can be tracked, controlled and monitored using remote computers connected through the Internet.

The concept of IoT enables, for example, GPS-based tracking, controlling and monitoring of devices; machine-to-machine (M2M) communication; connected cars; communication between wearable and personal devices and Industry 4.0.

1 **IoT Definition** The Internet is a vast global network of connected servers, computers, tablets and mobiles that is governed by standard protocols for connected systems. It enables sending, receiving, or communication of information, connectivity with remote servers, cloud and analytics platforms.

Thing in English has number of uses and meanings. In a dictionary, thing is a word used to refer to a physical object, an action or idea, a situation or activity, in case when one does not wish to be precise. Example of reference to an object is—an umbrella is a useful thing in rainy days. Streetlight is also referred to as a thing. Example of reference to an action is— such a thing was not expected from him. Example of reference to a situation is—such things were in plenty in that regime. Thus, combining both the terms, the definition of IoT can be explained as follows:

Internet of Things means a network of physical things (objects) sending, receiving, or communicating information using the Internet or other communication technologies and network just as the computers, tablets and mobiles do, and thus enabling the monitoring, coordinating or controlling process across the Internet or another data network.

Another source, defines the term IoT as follows:

Internet of Things is the network of physical objects or 'things' embedded with electronics, software, sensors and connectivity to enable it to achieve greater value and service by exchanging data with the manufacturer, operator and/or other connected devices. Each thing is uniquely identifiable through its embedded computing system but is able to interoperate within the existing Internet infrastructure.

IoT Vision Internet of Things is a vision where things (wearable watches, alarm clocks, home devices, surrounding objects) become 'smart' and function like living entities by sensing, computing and communicating through embedded devices which interact with remote objects (servers, clouds, applications, services and processes) or persons through the Internet or Near-Field Communication (NFC) etc. The vision of IoT can be understood through Examples 1.1 and 1.2.

Example 1.1:

Through computing, an umbrella can be made to function like a living entity. By installing a tiny embedded device, which interacts with a web based weather service and the devices owner through the Internet the following communication can take place. The umbrella, embedded with a circuit for the purpose of computing and communication connects to the Internet. A website regularly publishes the weather report. The umbrella receives these reports each morning, analyses the data and issues reminders to the owner at intermittent intervals around his/her office-going time. The reminders can be distinguished using differently coloured LED flashes such as red LED flashes for hot and sunny days, yellow flashes for rainy days.

A reminder can be sent to the owner's mobile at a pre-set time before leaving for office using NFC, Bluetooth or SMS technologies. The message can be—(i) Protect yourself from rain. It is going to rain. Don't forget to carry the umbrella; (ii) Protect yourself from the sun. It is going to be hot and sunny. Don't forget to carry the umbrella. The owner can decide to carry or not to carry the umbrella using the Internet connected umbrella.

Example 1.2:

Streetlights in a city can be made to function like living entities through sensing and computing using tiny embedded devices that communicate and interact with a central control-and-command station through the Internet. Assume that each light in a group of 32 streetlights comprises a sensing, computing and communication circuit. Each group connects to a group-controller (or coordinator) through Bluetooth or ZigBee. Each controller further connects to the central command-and-control station through the Internet.

The station receives information about each streetlight in each group in the city at periodic intervals. The information received is related to the functioning of the 32 lights, the faulty lights, about the presence or absence of traffic in group vicinity, and about the ambient conditions, whether cloudy, dark or normal daylight.

The station remotely programs the group controllers, which automatically take an appropriate action as per the conditions of traffic and light levels. It also directs remedial actions in case a fault develops in a light at a specific location. Thus, each group in the city is controlled by the 'Internet of streetlights'. Figure 1.1 shows the use of the IoT concept for streetlights in a city

IoT CONCEPTUAL FRAMEWORK Example 1.1 showed a single object (umbrella) communicating with a central server for acquiring data. The following equation describes a simple conceptual framework of IoT2 :

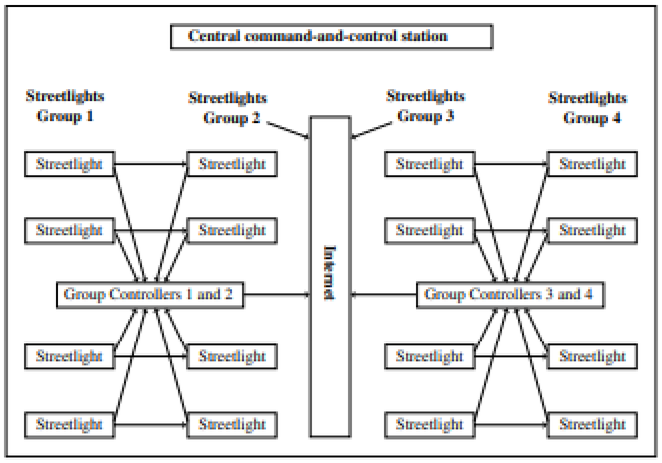


Figure 1.1 Use of Internet of Things concept for streetlights in a city

Physical Object + Controller, Sensor and Actuators + Internet = Internet of Things … 1.1

Equation 1.1 conceptually describes the Internet of umbrellas as consisting of an umbrella, a controller, sensor and actuators, and the Internet for connectivity to a web service and a mobile service provider.

Generally, IoT consists of an internetwork of devices and physical objects wherein a number of objects can gather the data at remote locations and communicate to units managing, acquiring, organising and analysing the data in the processes and services. Example 1.2 showed the number of streetlights communicating data to the group controller which connects to the central server using the Internet. A general framework consists of the number of devices communicating data to a data centre or an enterprise or a cloud server. The IoT framework of IoT used in number of applications as well as in enterprise and business processes is therefore, in general, more complex than the one represented by Equation 1.1. The equation below conceptually represents the actions and communication of data at successive levels in IoT consisting of internetworked devices and objects.

Gather + Enrich + Stream + Manage + Acquire + Organise and Analyse … 1.2

Equation 1.2 is an IoT conceptual framework for the enterprise processes and services, based on a suggested IoT architecture given by Oracle (Figure 1.5 in Section 1.3). The steps are as as follows:

1. At level 1 data of the devices (things) using sensors or the things gather the pre data from the internet.

2. A sensor connected to a gateway, functions as a smart sensor (smart sensor refers to a sensor with computing and communication capacity). The data then enriches at level 2, for example, by transcoding at the gateway. Transcoding means coding or decoding before data transfer between two entities.

3. A communication management subsystem sends or receives data streams at level 3.

4. Device management, identity management and access management subsystems receive the device's data at level 4.

5. A data store or database acquires the data at level 5.

6. Data routed from the devices and things organises and analyses at level 6. For example, data is analysed for collecting business intelligence in business processes.

The equation below is an alternative conceptual representation for a complex system. It is based on IBM IoT conceptual framework. The equation shows the actions and communication of data at successive levels in IoT. The framework manages the IoT services using data from internetwork of the devices and objects, internet and cloud services, and represents the flow of data from the IoT devices for managing the IoT services using the cloud server. Gather + Consolidate + Connect + Collect + Assemble + Manage and Analyse … 1.3 Equation 1.3 represents a complex conceptual framework for IoT using cloud-platformbased processes and services.

The steps are as follows: 1. Levels 1 and 2 consist of a sensor network to gather and consolidate the data. First level gathers the data of the things (devices) using sensors circuits. The sensor connects to a gateway. Data then consolidates at the second level, for example, transformation at the gateway at level 2.

2. The gateway at level 2 communicates the data streams between levels 2 and 3. The system uses a communication-management subsystem at level 3.

3. An information service consists of connect, collect, assemble and manage subsystems at levels 3 and 4. The services render from level 4.

4. Real time series analysis, data analytics and intelligence subsystems are also at levels 4 and 5. A cloud infrastructure, a data store or database acquires the data at level 5. Figure 1.3 shows blocks and subsystems for IoT in the IBM conceptual framework. New terms in the figure will be explained in the subsequent chapters. Various conceptual frameworks of IoT find number of applications including the ones in M2M communication networks, wearable devices, city lighting, security and surveillance and home automation. Smart systems use the things (nodes) which consist of smart devices, smart objects and smart services. Smart systems use the user interfaces (UIs), application programming interfaces (APIs), identification data, sensor data and communication ports.

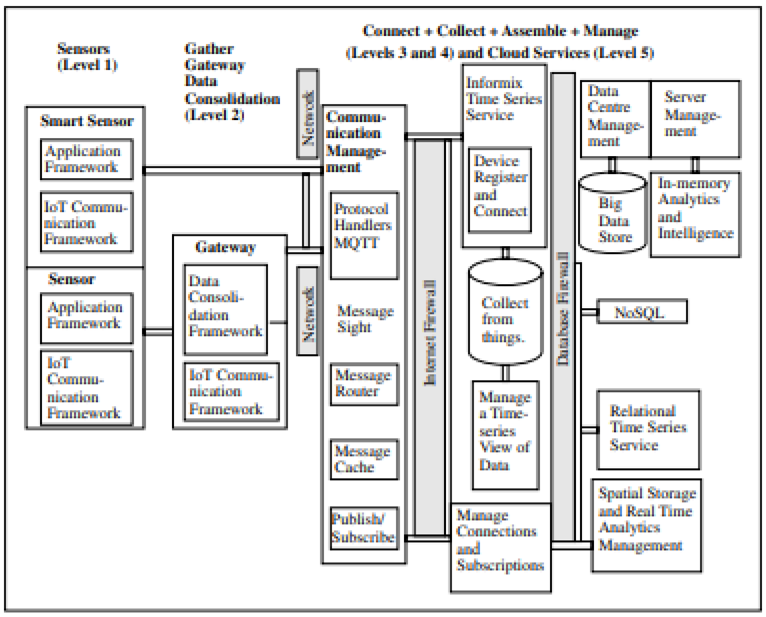

Sensors (Level 1)
Gather Gateway Data Consolidation (Level 2)
Connect + Collect + Assemble + Manage (Levels 3 and 4) and Cloud Services (Level 5)
Smart Sensor
Application Framework
IoT Communication Framework
Sensor
Application Framework
IoT Communication Framework
Gateway
Data Consolidation Framework
IoT Communication Framework
Network
Network
Communication Management
Protocol Handlers MQTT
Message Sight
Message Router
Message Cache
Publish/ Subscribe
Internet Firewall
Informix Time Series Service
Device Register and Connect
Collect from things.
Manage a Time-series View of Data
Manage Connections and Subscriptions
Database Firewall
Data Centre Management
Server Management
Big Data Store
In-memory Analytics and Intelligence
NoSQL
Relational Time Series Service
Spatial Storage and Real Time Analytics Management

Figure 1.3 IBM IoT conceptual framework

IoT ARCHITECTURAL VIEW:

An IoT system has multiple levels (Equations 1.1 to 1.3). These levels are also known as tiers. A model enables conceptualisation of a framework. A reference model can be used to depict building blocks, successive interactions and integration. An example is CISCO's presentation of a reference model comprising seven levels (Figure 1.4). New terms in the figure will be explained in the subsequent chapters.

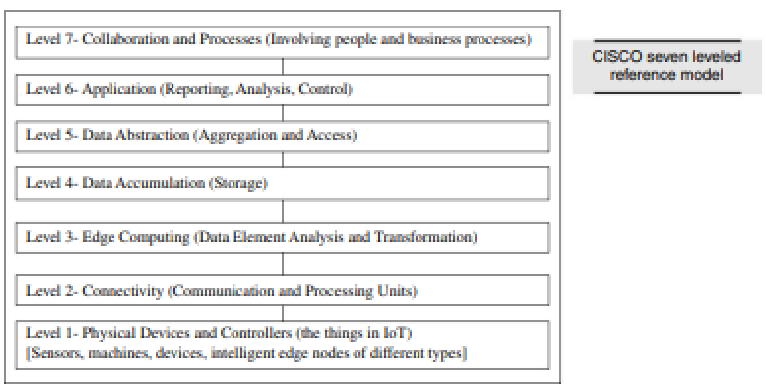

Level 7- Collaboration and Processes (Involving people and business processes)
Level 6- Application (Reporting, Analysis, Control)
Level 5- Data Abstraction (Aggregation and Access)
Level 4- Data Accumulation (Storage)
Level 3- Edge Computing (Data Element Analysis and Transformation)
Level 2- Connectivity (Communication and Processing Units)
Level 1- Physical Devices and Controllers (the things in IoT) [Sensors, machines, devices, intelligent edge nodes of different types]
CISCO seven leveled reference model

Figure 1.4 An IoT reference model suggested by CISCO that gives a conceptual framework for a general IoT system

A reference model could be identified to specify reference architecture. Several reference architectures are expected to co-exist in the IoT domain. Figure 1.5 shows an Oracle suggested IoT architecture. New terms in the figure will be explained in the subsequent chapters.

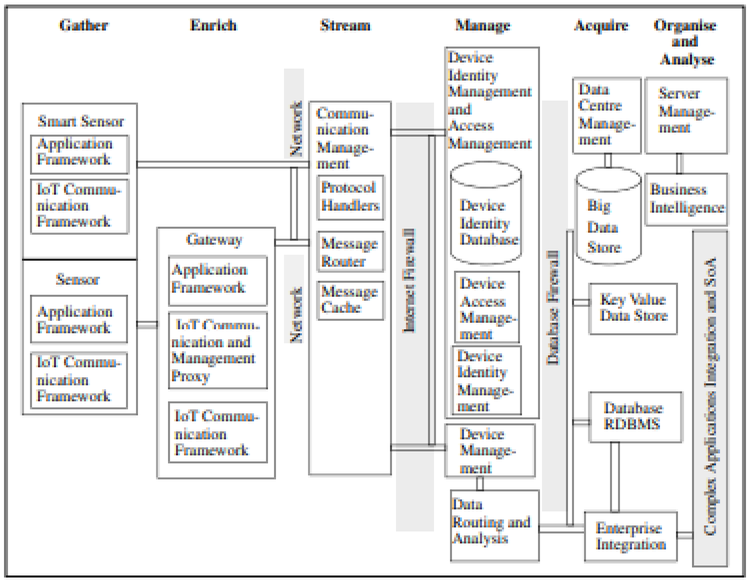


Figure 1.5 Oracle's IoT architecture (Device identity management means identifying a device, registering a device for actions after identifying, de-registering the device, assigning unique identity to the device. Device access management means enabling, disabling the device

An architecture has the following features:

● The architecture serves as a reference in applications of IoT in services and business processes.

● A set of sensors which are smart, capture the data, perform necessary data element analysis and transformation as per device application framework and connect directly to a communication manager.

● A set of sensor circuits is connected to a gateway possessing separate data capturing, gathering, computing and communication capabilities. The gateway receives the data in one form at one end and sends it in another form to the other end.

● The communication-management subsystem consists of protocol handlers, message routers and message cache.

● This management subsystem has functionalities for device identity database, device identity management and access management.

● Data routes from the gateway through the Internet and data centre to the application server or enterprise server which acquires that data.

● Organisation and analysis subsystems enable the services, business processes, enterprise integration and complex processes (These terms are explained in Chapter 5).

A number of models (CISCO, Purdue and other models) have been proposed at SWG (Sub Working Group) Teleconference of December 2014. Standards for an architectural framework for the IoT have been developed under IEEE project P2413. IEEE working group is working on a set of guidelines for the standard IEEE suggested P24133 standard for architecture of IoT. It is a reference architecture which builds upon the reference model(s). The reference architecture covers the definition of basic architectural building blocks and their integration capability into multi-tiered systems

P2413 architectural framework4 is a reference model that defines relationships among various IoT verticals, for example, transportation and healthcare. P2413 provides for the following:

Follows top-down approach (consider top layer design first and then move to the lowest)

● Does not define new architecture but reinvent existing architectures congruent with it

● Gives a blueprint for data abstraction

● Specifies abstract IoT domain for various IoT domains

● Recommends quality 'quadruple' trust that includes protection, security, privacy and safety

● Addresses how to document

● Strives for mitigating architecture divergence(s) Scope of IEEE P2413 standard defines an architectural framework for the IoT. It includes descriptions of various IoT domains, definitions of IoT domain abstractions and identification of commonalities between different IoT domains. Smart manufacturing, smart grid, smart buildings, intelligent transport, smart cities and e-health are different IoT domains. P2413 leverages existing applicable standards. It identifies planned or ongoing projects with a similar or overlapping scope.5

TECHNOLOGY BEHIND IoT: The following entities provide a diverse technologyenvironment and are examples of technologies, which are involved in IoT.

● Hardware (Arduino Raspberry Pi, Intel Galileo, Intel Edison, ARM mBed, Bosch XDK110, Beagle Bone Black and Wireless SoC)

● Integrated Development Environment (IDE) for developing device software, firmware and APIs

● Protocols [RPL, CoAP, RESTful HTTP, MQTT, XMPP (Extensible Messaging and Presence Protocol)]

● Communication (Powerline Ethernet, RFID, NFC, 6LowPAN, UWB, ZigBee, Bluetooth, WiFi, WiMax, 2G/3G/4G)

● Network backbone (IPv4, IPv6, UDP and 6LowPAN) ● Software (RIOT OS, Contiki OS, Thingsquare Mist firmware, Eclipse IoT)

● Internetwork Cloud Platforms/Data Centre (Sense, ThingWorx, Nimbits, Xively, openHAB, AWS IoT, IBM BlueMix, CISCO IoT, IOx and Fog, EvryThng, Azure, TCS CUP)

● Machine learning algorithms and software. An example of machine-learning software is GROK from Numenta Inc. that uses machine intelligence to analyse the streaming data from clouds and uncover anomalies, has the ability to learn continuously from data and ability to drive action from the output of GROK's data models and perform high level of automation for analysing streaming data.

The following five entities can be considered for the five levels behind an IoT system (Figure 1.3): 1. Device platform consisting of device hardware and software using a microcontroller (or SoC or custom chip), and software for the device APIs and web applications 2. Connecting and networking (connectivity protocols and circuits) enabling internetworking of devices and physical objects called things and enabling the internet connectivity to remote servers 3. Server and web programming enabling web applications and web services 4. Cloud platform enabling storage, computing prototype and product development platforms 5. Online transactions processing, online analytics processing, data analytics, predictive analytics and knowledge discovery enabling wider applications of an IoT system

Server-end Technology :IoT servers are application servers, enterprise servers, cloud servers, data centres and databases. Servers offer the following software components: ● Online platforms ● Devices identification, identity management and their access management ● Data accruing, aggregation, integration, organising and analysing ● Use of web applications, services and business processes

Major Components of IoT System

Major components of IoT devices are:

1. Physical object with embedded software into a hardware.

2. consisting of a microcontroller, firmware, sensors, control unit, actuators and communication module.

3. Communication module: Software consisting of device APIs and device interface for communication over the network and communication circuit/port(s), and middleware for creating communication stacks using 6LowPAN, CoAP, LWM2M, IPv4, IPv6 and other protocols.

4. for actions on messages, information and commands which the devices receive and then output to the actuators, which enable actions such as glowing LEDs, robotic hand movement etc.

Sensors and Control Units Sensors Sensors are electronic devices that sense the physical environments. An industrial automation system or robotic system has multiple smart sensors embedded in it. Sensor-actuator pairs are used in control systems. A smart sensor includes computing and communication circuits.

Recall Example 1.2 of Internet of streetlights. Each light has sensors for measuring surrounding light-intensity and surrounding traffic-proximity for sensing and transmitting the data after aggregation over a period. Sensors are used for measuring temperature, pressure, humidity, light intensity, traffic proximity, acceleration in an accelerometer, signals in a GPS, proximity sensor, magnetic fields in a compass, and magnetic intensity in a magnetometer.

Sensors are of two types. The first type gives analog inputs to the control unit. Examples are thermistor, photoconductor, pressure gauge and Hall sensor. The second type gives digital inputs to the control unit. Examples are touch sensor, proximity sensor, metal sensor, traffic presence sensor, rotator encoder for measuring angles and linear encoders for measuring linear displacements. Sensors and circuits are explained in detail in Chapter 7.

Control Units Most commonly used control unit in IoT consists of a Microcontroller Unit (MCU) or a custom chip. A microcontroller is an integrated chip or core in a VLSI or SoC. Popular microcontrollers are ATmega 328, ATMega 32u4, ARM Cortex and ARM LPC. An MCU comprises a processor, memory and several other hardware units which are interfaced together. It also has firmware, timers, interrupt controllers and functional IO units.

Additionally, an MCU has application-specific functional circuits designed as per the specific version of a given microcontroller family. For example, it may possess Analog to Digital Converters (ADC) and Pulse Width Modulators (PWM). Figure 1.6 shows various functional units in an MCU that are embedded in an IoT device or a physical object. New terms in the figure will be discussed in detail in Chapter 8. Sensor types—analog and digital output sensors Internet of Things: An Overview 15 Microcontroller

Communication Module

A communication module consists of protocol handlers, message queue and message cache. A device message-queue inserts the messages in the queue and deletes the messages from the queue in a first-in first-out manner. A device message-cache stores the received messages.

Representational State Transfer (REST) architectural style can be used for HTTP access by GET, POST, PUT and DELETE methods for resources and building web services. Communication protocols and REST style are explained in detail Chapter 3 and 4.

Software IoT software consists of two components—software at the IoT device and software at the IoT server. Figure 1.7 shows the software components for the IoT device hardware and server. Embedded software and the components are explained in Chapter 8. Software APIs, online component APIs and web APIs are explained in Section 9.4.

Middleware OpenIoT is an open source middleware. It enables communication with sensor clouds as well as cloud-based 'sensing as a service'. IoTSyS is a middleware which enables provisioning of communication stack for smart devices using IPv6, oBIX, 6LoWPAN, CoAP and multiple standards and protocols. The oBIX is standard XML and web services protocol oBIX (Open Building Information Xchange).

Operating Systems (OS) Examples of OSs are RIOT, Raspbian, AllJoyn, Spark and Contiki.

RIOT is an operating system for IoT devices. RIOT supports both developer and multiple architectures, including ARM7, Cortex-M0, Cortex-M3, Cortex-M4, standard x86 PCs and TI MSP430.

Raspbian is a popular Raspberry Pi operating system that is based on the Debian distribution of Linux

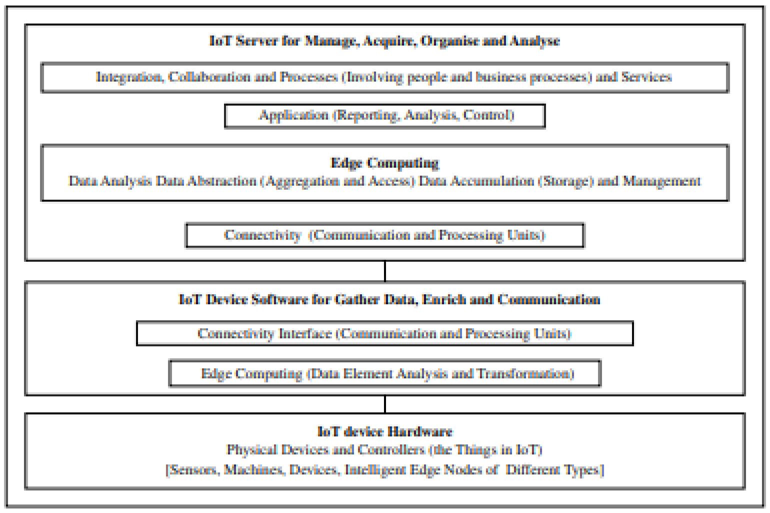

IoT Server for Manage, Acquire, Organise and Analyse
Integration, Collaboration and Processes (Involving people and business processes) and Services
Application (Reporting, Analysis, Control)
Edge Computing
Data Analysis Data Abstraction (Aggregation and Access) Data Accumulation (Storage) and Management
Connectivity (Communication and Processing Units)
IoT Device Software for Gather Data, Enrich and Communication
Connectivity Interface (Communication and Processing Units)
Edge Computing (Data Element Analysis and Transformation)
IoT device Hardware
Physical Devices and Controllers (the Things in IoT)
[Sensors, Machines, Devices, Intelligent Edge Nodes of Different Types]

Figure 1.7 IoT software components for device hardware

AllJoyn is an open-source OS created by Qualcomm. It is a cross platform OS with APIs available for Android, iOS, OS X, Linux and Windows OSs. It includes a framework and a set of services. It enables the manufacturers to create compatible devices.

Spark is a distributed, cloud-based IoT operating system and web-based IDE. It includes a command-line interface, support for multiple languages and libraries for working with several different IoT devices

Contiki OS7 is an open-source multitasking OS. It includes 6LowPAN, RPL, UDP, DTLS and TCP/IP protocols which are required in low-power wireless IoT devices. Example of applications are street lighting in smart cities, which requires just 30 kB ROM and 10 kB RAM.

UNIT IV: M2M and IoT Technology Fundamentals

Devices and gateways, Local and wide area networking, Data management, Business processes in IoT, Everything as a Service(XaaS), M2M and IoT Analytics, Knowledge Management.

4.1 Devices and gateways

4.1.1 Introduction

- There is a growing market for small-scale embedded processing such as 8-, 16-, and 32-bit microcontrollers with on-chip RAM and flash memory, I/O capabilities, and networking interfaces such as IEEE 802.15.4 that are integrated on tiny System-on-a-Chip (SoC) solutions.
- Such devices enable very constrained devices with a small footprint of a few mm2 and with a very low power consumption in the milli- to micro-Watt range, but which are capable of hosting an entire Transmission Control Protocol/Internet Protocol (TCP/IP) stack, including a small web server.
- A device is a hardware unit that can sense aspects of it's environment and/or actuate, i.e. perform tasks in its environment.
- A device can be characterized as having several properties, including:

  • Microcontroller: 8-, 16-, or 32-bit working memory and storage.

  • Power Source: Fixed, battery, energy harvesting, or hybrid.

  • Sensors and Actuators: Onboard sensors and actuators, or circuitry that allows them to be connected, sampled, conditioned, and controlled.

  • Communication: Cellular, wireless, or wired for LAN and WAN communication.

  • Operating System (OS): Main-loop, event-based, real-time, or full featured OS.

  • Applications: Simple sensor sampling or more advanced applications.

  • User Interface: Display, buttons, or other functions for user interaction.

  • Device Management (DM): Provisioning, firmware, bootstrapping, and monitoring.

  • Execution Environment (EE): Application lifecycle management and Application Programming Interface (API).

4.1.1.1 Device types

- Group devices into two categories

  • **Basic Devices:** Devices that only provide the basic services of sensor readings and/or actuation tasks, and in some cases limited support for user interaction. LAN communication is supported via wired or wireless technology, thus a gateway is needed to provide the WAN connection.

• **Advanced Devices:** In this case the devices also host the application logic and a WAN connection. They may also feature device management and an execution environment for hosting multiple applications. Gateway devices are most likely to fall into this category.

4.1.1.2 Deployment scenarios for devices

➢ Example deployment scenarios for basic devices include:

• **Home Alarms:** Such devices typically include motion detectors, magnetic sensors, and smoke detectors. A central unit takes care of the application logic that calls security and sounds an alarm if a sensor is activated when the alarm is armed. The central unit also handles the WAN connection towards the alarm central. These systems are currently often based on proprietary radio protocols.

• **Smart Meters:** The meters are installed in the households and measure consumption of, for example, electricity and gas. A concentrator gateway collects data from the meters, performs aggregation, and periodically transmits the aggregated data to an application server over a cellular connection. By using a capillary network technology it’s possible to extend the range of the concentrator gateway by allowing meters in the periphery to use other meters as extenders, and interface with handheld devices on the Home Area Network side.

• **Building Automation Systems (BASs):** Such devices include thermostats, fans, motion detectors, and boilers, which are controlled by local facilities, but can also be remotely operated.

• **Standalone Smart Thermostats:** These use Wi-Fi to communicate with web services. Examples for advanced devices, meanwhile, include:

• **Onboard units** in cars that perform remote monitoring and configuration over a cellular connection.

• **Robots and autonomous vehicles** such as unmanned aerial vehicles that can work both autonomously or by remote control using a cellular connection.

• **Video cameras** for remote monitoring over 3G and LTE.

• **Oil well monitoring** and collection of data points from remote devices.

• **Connected printers** that can be upgraded and serviced remotely.

4.1.2 Basic devices

- These devices are often intended for a single purpose, such as measuring air pressure or closing a valve. I
- In some cases several functions are deployed on the same device, such as monitoring humidity, temperature, and light level.
- The main focus is on keeping the bill of materials (BOM) as low as possible by using inexpensive microcontrollers with built-in memory and storage, often on an SoC-integrated circuit with all main components on one single chip (Figure 5.1).
- Another common goal is to enable battery as a power source, with a lifespan of a year and upwards by using ultra-low energy microcontrollers.

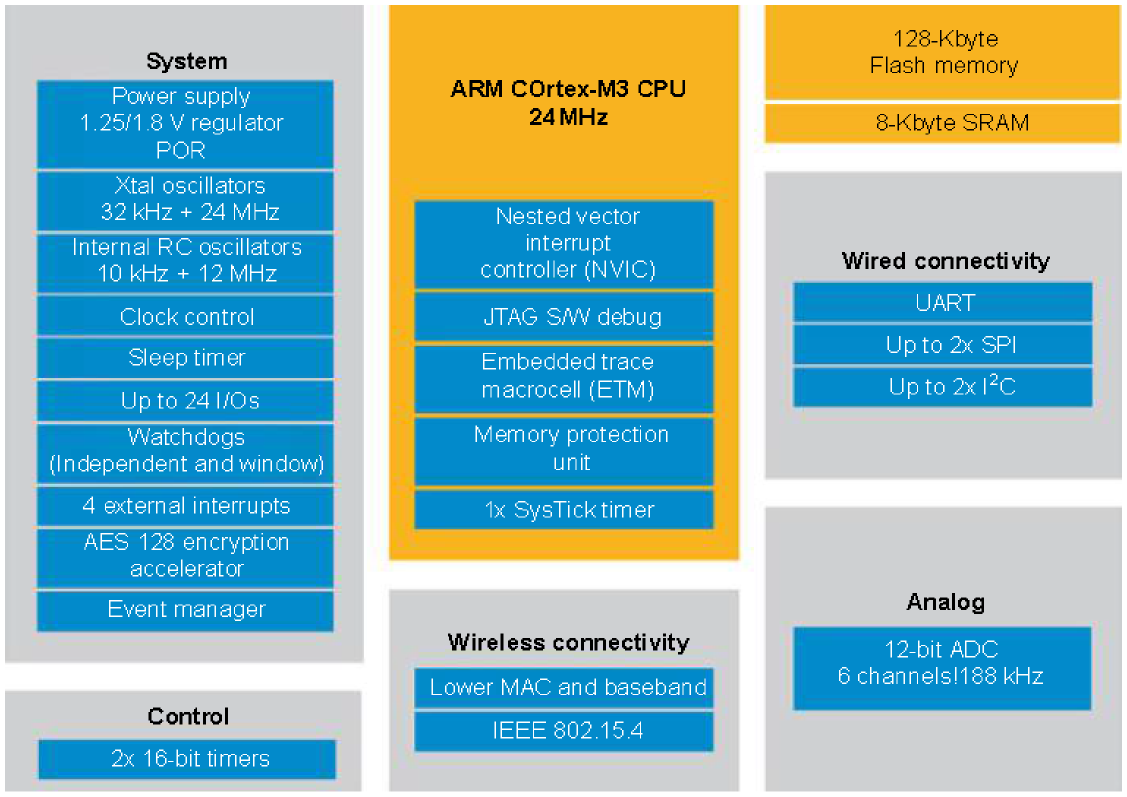


FIGURE 5.1

Example of a microcontroller with integrated STM32W-RFCKIT.

- The microcontroller typically hosts a number of ports that allow integration with sensors and actuators, such as General Purpose I/O (GPIO) and an analog-to-digital converter (ADC) for supporting analog input.
- For certain actuators, such as motors, pulse-width modulation (PWM) can be used.
- As low-power operation is paramount to battery-powered devices, the microcontroller hosts functions that facilitate sleeping, such as interrupts that can wake up the device on external and internal events.

- Some devices even go as far as harvesting energy from their environment, e.g. in the form of solar, thermal, and physical energy.
- To interact with peripherals such as storage or display, it's common to use a serial interface such as SPI, I2C, or UART.
- These interfaces can also be used to communicate with another microcontroller on the device.
- This is common when the there is a need for offloading certain tasks, or when in some cases the entire application logic is put on a separate host processor.
- It's not unusual for the micro controller to also contain a security processor,e.g. to accelerate Advanced Encryption Standard (AES).
- This is necessary to allow encrypted communication over the radio link without the need for a host processor.
- The gateway together with the connected devices form a capillary network.
- The microcontroller contains most of the radio functions needed for communicating with the gateway and other devices in the same capillary network.
- An external antenna is, however, necessary, and preferably a filter that removes unwanted frequencies, e.g. a surface acoustic wave (SAW) filter.
- Due to limited computational resources, these devices commonly do not use a typical OS.
- It may be something as simple as a single-threaded main-loop or a low-end OS such as FreeRTOS, Atomthreads, AVIX-RT, ChibiOS/RT, ERIKA Enterprise, TinyOS, or Thingsquare Mist/Contiki.
- These OSes offer basic functionality, e.g. memory and concurrency model management, (sensor and radio) drivers, threading, TCP/IP, and higher level protocol stacks.
- The actual application logic is located on top of the OS or in the mainloop.
- A typical task for the application logic is to read values from the sensors and to provide these over the LAN interface in a semantically correct manner with the correct units.

4.1.3 Gateways

- A gateway serves as a translator between different protocols, e.g. between IEEE 802.15.4 or IEEE 802.11, to Ethernet or cellular.
- There are many different types of gateways, which can work on different levels in the protocol layers.
- A gateway refers to a device that performs translation of the physical and link layer, but application layer gateways (ALGs) are also common.
- The latter is preferably avoided because it adds complexity and is a common source of error in deployments.
- Some examples of ALGs include the ZigBee Gateway Device which translates from ZigBee to SOAP and IP, or gateways that translate from Constrained Application Protocol (CoAP) to HyperText Transfer Protocol/Representational State Transfer (HTTP/REST).

- Tthe gateway device is also used for many other tasks, such as data management, device management, and local applications.

4.1.3.1 Data management

- Typical functions for data management include performing sensor readings and caching this data, as well as filtering, concentrating, and aggregating the data before transmitting it to back-end servers.

4.1.3.2 Local applications

- Examples of local applications that can be hosted on a gateway include closed loops, home alarm logic, and ventilation control, or the data management function above
- The benefit of hosting this logic on the gateway instead of in the network is to avoid downtime in case of WAN connection failure, minimize usage of costly cellular data, and reduce latency.
- To facilitate efficient management of applications on the gateway, it's necessary to include an execution environment.
- The execution environment is responsible for the lifecycle management of the applications, including installation, pausing, stopping, configuration, and uninstallation of the applications.
- A common example of an execution environment for embedded environments is OSGi, which is based on Java: applications are built as one or more Bundles, which are packaged as Java JAR files and installed using a so-called Management Agent.
- The Management Agent can be controlled from, for example, a terminal shell or via a protocol such as CPE WAN Management Protocol (CWMP).
- Bundle packages can be retrieved from the local file system or over HTTP, for example. OSGi also provides security and versioning for Bundles, which means that communication between Bundles is controlled, and several versions of them can exist.
- The benefit of versioning and the lifecycle management functions is that the OSGi environment never needs to be shut down when upgrading, thus avoiding downtime in the system.
- Also, Linux can be used as an execution environment.

4.1.3.3 Device management

- Device management (DM) is an essential part of the IoT and provides efficient means to perform many of the management tasks for devices:
  • **Provisioning:** Initialization (or activation) of devices in regards to configuration and features to be enabled.
  • **Device Configuration:** Management of device settings and parameters.
  • **Software Upgrades:** Installation of firmware, system software, and applications on the device.

• **Fault Management:** Enables error reporting and access to device status.

- Examples of device management standards include TR-069 and OMA-DM.
- In the simplest deployment, the devices communicate directly with the DM server.
- This is, however, not always optimal or even possible due to network or protocol constraints, e.g. due to a firewall or mismatching protocols.
- In these cases, the gateway functions as mediator between the server and the devices, and can operate in three different ways:
  - If the devices are visible to the DM server, the gateway can simply forward the messages between the device and the server and is not a visible participant in the session.
  - In case the devices are not visible but understand the DM protocol in use, the gateway can act as a proxy, essentially acting as a DM server towards the device and a DM client towards the server.
  - For deployments where the devices use a different DM protocol from the server, the gateway can represent the devices and translate between the different protocols (e.g. TR-069, OMA-DM, or CoAP).
- The devices can be represented either as virtual devices or as part of the gateway

4.1.4 Advanced devices

- An advanced device are the following:
  - A powerful CPU or microcontroller with enough memory and storage to host advanced applications, such as a printer offering functions for copying, faxing, printing, and remote management.
  - A more advanced user interface with, for example, display and advanced user input in the form of a keypad or touch screen.
  - Video or other high bandwidth functions.

4.1.5 Summary and vision

- The most important of these is security, both in terms of physical security as well as software and network security.
- External factors that can affect the operation of the devices, such as rain, wind, chemicals, and electromagnetic influences.
- One of the major effects that the IoT will have on devices is to disrupt the current value chains, where one actor controls everything from device to service.
- This will happen due to standardization and consolidation of technologies, such as protocols, OSes, software and programming languages (e.g. Java for embedded devices), and the business
- New types of actors will be able to enter the market, e.g. specialized device vendors, cloud solution providers, and service providers.

- Standardization will improve interoperability between devices, as well as between devices and services, resulting in commoditization of both.
- Another expected outcome of improved interoperability is the possibility to reuse the same device for multiple services;
- for example, a motion detector can be used both for security purposes as well as for reducing energy consumption by detecting when no one is in the room.
- Thanks to developments in hardware and network technologies, entirely new device classes and features are expected, such as:
    - Battery-powered devices with ultra-low power cellular connections.
    - Devices that harvest energy from their environment.
    - Smart bandwidth management and protocol switching, i.e. using adaptive RF mechanisms to swap between, for example, Bluetooth LE and IEEE 802.15.4.
    - Multi-radio/multi-rate to switch between bands or bit rates
    - Microcontrollers with multicore processors.
    - Novel software architectures for better handling of concurrency.
    - The possibility to automate the design of integrated circuits based on business-level logic and use case.

4.2 Local and wide area networking

4.2.1 The need for networking

- A network is created when two or more computing devices exchange data or information.
- The ability to exchange pieces of information using telecommunications technologies has changed the world
- Devices are known as “nodes” of the network, and they communicate over “links.”
- In modern computing, nodes range from personal computers, servers, and dedicated packet switching hardware, to smart phones, games consoles, television sets and, increasingly, heterogeneous devices that are generally characterized by limited resources and functionalities.
- Limitations typically include computation, energy, memory, communication (range, bandwidth, reliability, etc.) and application specificity (e.g. specific sensors, actuators, tasks), etc. Such devices are typically dedicated to specific tasks, such as sensing, monitoring, and control.
- Network links rely upon a physical medium, such as electrical wires, air, and optical fibers, over which data can be sent from one network node to the next.
- A selected physical medium determines a number of technical and economic considerations.

- Nodes of the network must have an awareness of all nodes in the network with which they can indirectly communicate. This can be a direct connection over one link (edge, the transition or communication between two nodes over a link), or knowledge of a route to the desired (destination) node by communicating through cooperating nodes, over multiple edges.

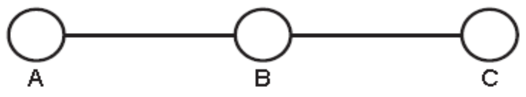


FIGURE 5.2

A network.

- In Figure 5.2 is the simplest form of network that requires knowledge of a route to communicate between nodes that do not have direct physical links.
- if node A wishes to transfer data to node C, it must do so through node B.
- Thus, node B must be capable of the following:
    - Communicating with both node A and node C,
    - advertising to node A and node C that it can act as an intermediary.
- Basic networking requirements have become explicit.
- It is essential to uniquely identify each node in the network, and it is necessary to have cooperating nodes capable of linking nodes between which physical links do not exist.
- In modern computing, this equates to IP addresses and routing tables.
- Consider the differences between streaming video from a surveillance camera, for example, and an intrusion-detection system based on a passive sensor.
- Streaming video requires high bandwidth, whereas transmitting a small amount of information about the detection of an intruder requires a tiny amount of bandwidth, but a higher degree of reliability with respect to both the communications link and the accuracy of the detection.
- Node A is a device that can only communicate over a particular wireless channel of limited range
- Node B is cap able of communicating with node A, but also with an application server with service capabilities (node C, with which it can connect using wired Ethernet, e.g. over a complex link using a standardized protocol and/or web service such as REST at the application layer) over the Internet.
- Node B may be connected to a sub-network (of child nodes, similar to node A) of up to thousands of similarly constrained devices (A1. . .An).
- These thousands of devices may be equipped with sensors, deployed specifically to monitor some physical phenomenon.
- They can only communicate with one another and node B, and may communicate with each other over single or multiple hops.

- Consider that the owner of the WSN wishes to obtain the data from each of the (A1. . .An) devices in the WSN.
- However, the preferred way to read the data is through a web browser, or application on a smartphone/tablet, via node C.
- Therefore, a networking solution is required to transfer all of the WSN data from nodes A1. . .An to node C, through node B.
- This concept maps directly to the M2M Functional Architecture, where nodes A1. . .An are an M2M Area Network, node B is an M2M Gateway, and node C is representative of M2M Service Capabilities and Applications.
- A Local Area Network (LAN) was traditionally distinguishable from a Wide Area Network (WAN) based on the geographic coverage requirements of the network, and the need for third party, or leased, communication infrastructure.
- In the case of the LAN, a smaller geographic region is covered, such as a commercial building, an office block, or a home, and does not require any leased communications infrastructure.
- WANs provide communication links that cover longer distances, such as across metropolitan, regional, or by textbook definition, global geographic areas.
- In practice, WANs are often used to link LANs and Metropolitan Area Networks (MAN)
- LANs tended to cover distances of tens to hundreds of meters, whereas WAN links spanned tens to hundreds of kilometers.
- The most popular wired LAN technology is Ethernet. Wi-Fi is the most prevalent wireless LAN (WLAN) technology.
- Wireless WAN (WWAN), as a descriptor, covers cellular mobile telecommunication networks, a significant departure from WLAN in terms of technology, coverage, network infrastructure, and architecture.
- Difference between LAN and WAN

S.NO	LAN	WAN
1.	LAN stands for Local Area Network.	Whereas WAN stands for Wide Area Network.
2.	LAN's ownership is private.	But WAN's ownership can be private or public.
3.	The speed of LAN is	While the speed of WAN is slower

S.NO	LAN	WAN
	high(more than WAN).	than LAN.
4.	The propagation delay is short in LAN.	Whereas the propagation delay in WAN is long(longer than LAN).
5.	There is less congestion in LAN(local area network).	While there is more congestion in WAN(Wide Area Network).
6.	There is more fault tolerance in LAN.	While there is less fault tolerance in WAN.
7.	LAN’s design and maintenance is easy.	While it’s design and maintenance is difficult than LAN.

- The current generation of WWAN technology includes LTE (or 4G) and WiMAX.
- Acting as a link between LANs and Wireless Personal Area Networks (WPANs), M2M Gateway Devices typically include cellular transceivers, and allow seamless IP-connectivity over heterogeneous physical media.
- In the home, the “wireless router” typically behaves as a link between the Wi-Fi (WLAN, and thus connected laptops, tablets, smartphones, etc. commonly found in the home) and Digital Subscriber Line (DSL) broadband connectivity, traditionally arriving over telephone lines. “DSL” refers to Internet access carried over legacy (wired) telephone networks, and encompasses numerous standards and variants.
- “Broadband” indicates the ability to carry multiple signals over a number of frequencies, with a typical minimum bandwidth of 256 kbps.
- In the office, the Wi-Fi wireless access points are typically connected to the wired corporate (Ethernet) LAN, which is subsequently connected to a wider area network and Internet backbone, typically provided by an Internet Service Provider (ISP).
- The need exists to interconnect devices (generally integrated microsystems) with central data processing and decision support systems, in addition to one another.
- In WLAN technologies, a geographic region can be covered by a network of devices that connect to the Internet via a gateway device, which may use a leased network connection.
- For example, a gateway device can access the IP backbone over a WWAN (e.g. GPRS/UMTS/LTE/WiMAX) link, or over a WLAN link.

- WPANs is the for newer standards that govern low-power, low-rate networks suitable for M2M and IoT applications.
- "IEEE 802.15.4 _ Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs).
- This is similar to the evolution of Wi-Fi WLAN technology (e.g. IEEE 802.11, a, b, g, n, etc.).
- Communication ranges for IEEE 802.15.4 technology may range from tens of meters to kilometers.
- Devices in an M2M Area Network connect to the IP backbone, or Network Domain, via an M2M Gateway device.
- Gateway device is equipped with a cellular transceiver that is physically compatible with UMTS or LTE-Advanced, for example, WWAN.
- The same device will also be equipped with the necessary transceiver to communicate on the same physical medium as the M2M Area Network(s) in the M2M Device Domain.
- M2M Area Networks may include a plethora of wired or wireless technologies, including: Bluetooth LE/Smart, IEEE 802.15.4 (LR-WPAN; e.g. ZigBee, IETF 6LoWPAN, RPL, CoAP, ISA100.11a, WirelessHART, etc.),
- The "Internet of Things," as a term, originated from Radio Frequency Identification (RFID) research, wherein the original IoT concept was that any RFID-tagged "thing" could have a virtual presence on the "Internet."
- RFID ,bar codes and QR codes use different technological means to achieve the same result.
- M2M applications become more synonymous with IoT, it is necessary to understand the technologies, limitations, and implications of the networking infrastructure.

<u>4.2.2 Wide area networking</u>

- WANs are typically required to bridge the M2M Device Domain to the backhaul network, thus providing a proxy that allows information (data, commands etc) to traverse heterogeneous networks.
- It is used to provide communications services between the M2M service enablement and the physical deployments of devices in the field.
- WAN is capable of providing the bi-directional communications links between services and devices which is achieved by means of physical and logical proxy.
- The proxy is achieved using an M2M Gateway Device.
- M2M Gateway Device is typically an integrated microsystem with multiple communications interfaces and computational capabilities.
- It is a critical component in the functional architecture, as it must be capable of handling all of the necessary interfacing to the M2M Service Capabilities and Management Functions.

- Example: consider a device that incorporates both an IEEE 802.15.4-compliant transceiver, capable of communicating with a capillary network of similarly equipped devices, and a cellular transceiver that connects to the Internet using the UMTS network.
- Transceivers (sometimes referred to as modems) are typically available as hardware modules with which the central intelligence of the device (gateway or cell phone) interacts by means of standardized AT Commands.
- This device is now capable of acting as a physical proxy between the LR-WPAN, or M2M Device Domain, and the M2M Network Domain.
- The latest ETSI M2M Functional Architecture is illustrated in Figure 5.3.

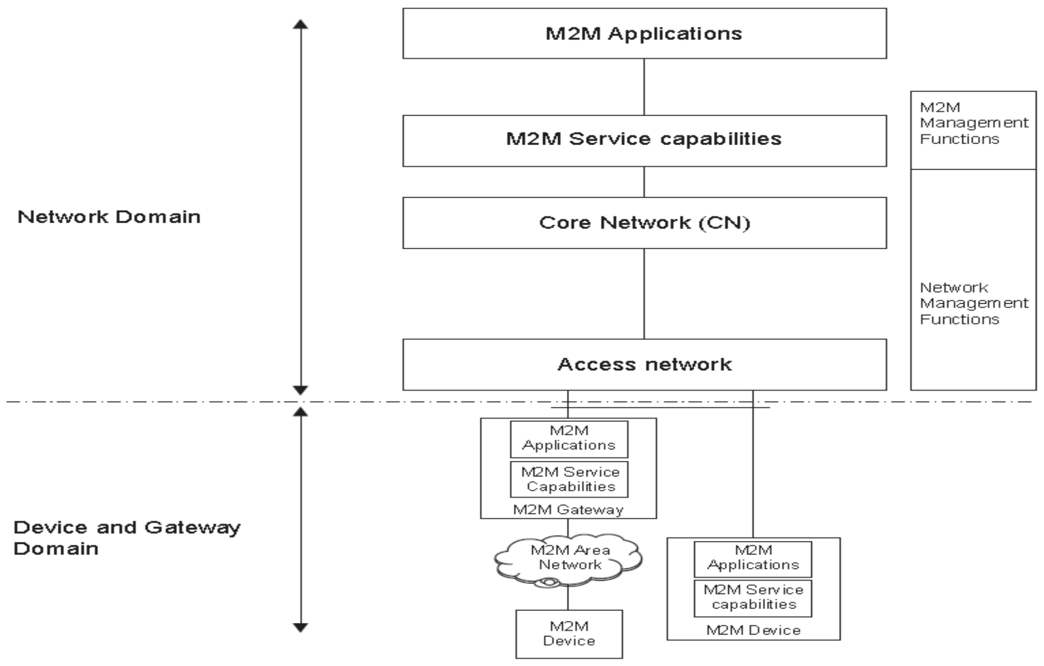


FIGURE 5.3

ETSI M2M Functional Architecture.

- The Access and Core Network in the ETSI M2M Functional Architecture are foreseen to be operated by a Mobile Network Operator (MNO), and can be thought of simply as the "WAN" for the purposes of interconnecting devices and backhaul networks (Internet), thus, M2M Applications, Service Capabilities, Management Functions, and Network Management Functions.
- The WAN covers larger geographic regions using wireless as well as wire-based access.
- WAN technologies include cellular networks (using several generations of technologies), DSL, WiMAX, Wi-Fi, Ethernet, Satellite, and so forth.
- The WAN delivers a packet-based service using IP as default. Circuit-based services can also be used in certain situations.
- important functions of the WAN include:
    - The main function of the WAN is to establish connectivity between capillary

networks, hosting sensors, and actuators, and the M2M service enablement.

- The default connectivity mode is packet-based using the IP family of technologies.
- Many different types of messages can be sent and received. for example, a message sent from a sensor in an M2M Area Network and resulting in an SMS received from the M2M Gateway or Application
- Use of identity management techniques (primarily of M2M devices) in cellular and non-cellular domains to grant right-of-use of the WAN resource.
- The following techniques are used for these purposes:
    - ✓ MCIM (Machine Communications Identity Module) for remote provisioning of SIM targeting M2M devices.
    - ✓ xSIM (x-Subscription Identity Module), like SIM, USIM, ISIM.
    - ✓ Interface identifiers, an example of which is the MAC address of the device, typically stored in hardware.
    - ✓ Authentication/registration type of functions (device focused).
    - ✓ Authentication, Authorization, and Accounting (AAA), such as RADIUS services.
    - ✓ Dynamic Host Configuration Protocol (DHCP), e.g. employing deployment-specific configuration parameters specified by device, user, or application-specific parameters residing in a directory.
    - ✓ Subscription services (device-focused).
    - ✓ Directory services, e.g. containing user profiles and various device (s) parameter(s), setting(s), and combinations thereof.
- M2M-specific considerations include, in particular:
    - ✓ MCIM (cf. 3GPP SA3 work).
    - ✓ User Data Management (e.g. subscription management).
    - ✓ Network optimizations (cf. 3GPP SA2 work).

4.2.2.1 3rd generation partnership project technologies and machine type communications

- ➢ Machine Type Communications (MTC) is heavily referred to in the ETSI documentation.
- ➢ MTC refers to small amounts of data that are communicated between machines (devices to back-end services and vice versa) without the need for any human intervention. In the 3rd Generation Partnership Project (3GPP), MTC is used to refer to all M2M communication.

<u>4.2.3 Local area networking</u>

- ➢ Capillary networks are typically autonomous, self-contained systems of M2M devices that may be connected to the cloud via an appropriate Gateway.

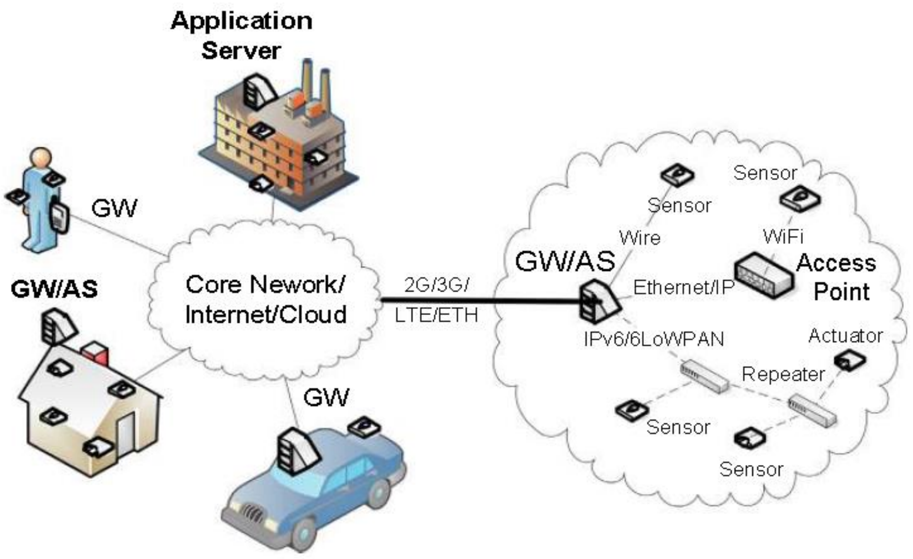


FIGURE 5.4

Capillary networks and their inside view.

- They are often deployed in controlled environments such as vehicles, buildings, apartments, factories, bodies, etc. (Figure 5.4) in order to collect sensor measurements, generate events should sensing thresholds be breached, and sometimes control specific features of interest (e.g. heart rate of a patient, environmental data on a factory floor, car speed, air conditioning appliances, etc.).
- There will exist numerous capillary networks that will employ short-range wired and wireless communication and networking technologies.
- For certain application areas, there is a need for autonomous local operation of the capillary network.
- In the event that application-level logic is enforceable via the cloud, some will still need to be managed locally.
- The complexity of the local application logic varies by application.
- For example, a building automation network may need local control loop functionality for autonomous operation, but can rely on external communication for configuration of control schemas and parameters.
- The M2M devices in a capillary network are typically thought to be low-capability nodes (e.g. battery operated, with limited security capabilities) for cost reasons, and should operate autonomously.
- For this reason, a GW/application server will naturally also be part of the architected solution for capillary networks.
- More and more (currently closed) capillary networks will open up for integration with the enterprise back end systems.
- For capillary networks that expose devices to the cloud/Internet, IP is envisioned to be the common waist.

- IPv6 will be the protocol of choice for M2M devices that operate a 6LoWPAN-based stack.
- IPv4 will still be used for capillary networks operating in non-6LoWPAN IP stacks (e.g. Wi-Fi capillary networks).
- In terms of short-range communication technology convergence, an IPv6 stack with 6LoWPAN running above the physical medium is expected.
- The development of the IEEE 802.15.4g standard, a physical layer amendment to support Smart Utility Networks (SUN) _ smart grid in particular _ designed to operate over much larger geographic distances (wireless links spanning tens of kilometers), and specifically designed for minimal infrastructure, low power, many-device networks.

4.2.3.1 Deployment considerations

- There are increasing numbers of innovative IoT applications (hardware and software) marketed as consumer products.
- These range from intelligent thermostats for effectively managing comfort and energy use in the home, to precision gardening tools (sampling weather conditions, soil moisture, etc.).
- Scaling up for industrial applications and moving from laboratories into the real world creates significant challenges that are not yet fully understood.
- Low-rate, low-power communications technologies are known to be "lossy." The reasons can relate to environmental factors, which impact upon radio performance, technical factors such as performance trade-offs based on the characteristics of medium access control and routing protocols, and physical limitations of devices (including software architectures, runtime and execution environments, computational capabilities, energy availability, local storage, etc), and practical factors such as maintenance opportunities (scheduled, remote, accessibility, etc.).
- Numerous deployment environments (factories, buildings, roads, vehicles) are expected in addition to wildly varying application scenarios and operational and functional requirements of the systems.
- ETSI describes a set of use cases, namely eHealth, Connected Consumer, Automotive, Smart Grid, and Smart Meter, that only capture some of the breadth of potential deployment scenarios and environments that are possible.
- Assuming that IP connectivity can be the fundamental mechanism to bridge heterogeneous physical and link layer technologies, it stands to reason that fragmentation can continue such that appropriate technologies are available for the breadth of potential application scenarios.

4.2.3.2 Key technologies

- Power Line Communication (PLC) refers to communicating over power (or phone, coax, etc.) lines.

- This amounts to pulsing, with various degrees of power and frequency, the electrical lines used for power distribution.
- PLC comes in numerous flavors. At low frequencies (tens to hundreds of Hertz) it is possible to communicate over kilometers with low bit rates (hundreds of bits per second). Typically, this type of communication was used for remote metering, and was seen as potentially useful for the smart grid.
- Enhancements to allow higher bit rates have led to the possibility of delivering broadband connectivity over power lines.
- There have been a number of attempts to standardize PLC in recent years. NIST recently included IEEE 1901 and ITU-T G.hn as standards for further review for potential use in the smart grid in the United States.
- LAN (and WLAN) continues to be important technology for M2M and IoT applications.
- This is due to the high bandwidth, reliability, and legacy of the technologies. Where power is not a limiting factor, and high bandwidth is required, devices may connect seamlessly to the Internet via Ethernet (IEEE 802.3) or Wi-Fi (IEEE 802.11).
- The IEEE 802.11 (Wi-Fi) standards continue to evolve in various directions to improve certain operational characteristics depending on usage scenario.
- A widely adopted recent release was IEEE 802.11n, which was specifically designed to enhance throughput (typically useful for streaming multimedia).
- Ongoing work such as IEEE 802.11ac is developing an even higher throughput version to replace this, focusing efforts in the 5 GHz band.
- IEEE 802.11ah is allow a number of networked devices to cooperate in the ,1 GHz (ISM) band.
- The idea is to exploit collaboration (relaying, or networking in other words) to extend range, and improve energy efficiency (by cycling the active periods of the radio transceiver).
- Bluetooth Low Energy (BLE; “Bluetooth Smart”) is designed for short-range (,50 m) applications in healthcare, fitness, security, etc., where high data rates (millions of bits per second) are required to enable application functionality.
- It is deliberately low cost and energy efficient by design, and has been integrated into the majority of recent smart phones.
- Low-Rate, Low-Power Networks are another key technology that form the basis of the IoT.
- For example, the IEEE 802.15.4 family of standards was one of the first used in practical research and experimentation in the field of WSNs.
- Low-Rate Wireless Personal Area Networks (LR-WPAN)- It covered the Physical and Medium Access Control layers, specifying use in the ISM bands at frequencies around 433 MHz, 868/915 MHz, and 2.4 GHz. This supported data rates of between 20 kbps up to 256 kbps, depending on selected band, over distances ranging from tens of meters to kilometers.

- Radio duty cycling refers to managing the active periods of the Radio Frequency Integrated Circuit (RFIC) during transmission, and listening to the medium.
- IEEE 802.15.4 defines the PHY layer, and in some instances the MAC layer, upon which a number of low-energy communications specifications have been built. Namely, ZigBee.
- Recent developments, such as the PHY Amendment for Smart Utility Networks (SUN), IEEE 802.15.4g, seek to extend the operational coverage of these networks up to tens of kilometers in order to provide extremely wide geographic coverage with minimal infrastructure.
- 6LoWPAN (IPv6 Over Low Power Wireless Personal Area Networks) was developed initially by the 6LoWPAN Working Group (WG) of the IETF as a mechanism to transport IPv6 over IEEE 802.15.4-2003 networks.
- Specifically, methods to handle fragmentation, reassembly, and header compression were the primary objectives.
- The WG also developed methods to handle address autoconfiguration, the hooks for mesh networking, and network management.
- RPL (IPv6 Routing Protocol for Low Power and Lossy Networks) was developed by the IETF Routing over Low Power and Lossy Networks (RoLL) WG.
- They defined Low Power Lossy Networks as those typically characterized by high data loss rates, low data rates, and general instability.
- No specific physical or medium access control technologies were specified, but typical links considered include PLC, IEEE 802.15.4, and low-power Wi-Fi.
- Typical use cases involve the collection of data from many (for example) sensing points, nodes towards a sink, or alternatively, flooding information from a sink to many nodes in the network.
- Thus, the well-known concept of a Directed Acyclic Graph (DAG) structure was concentrated to a Destination Oriented DAG (DODAG) for the purposes of initial development.
- The group defined a new ICMPv6 message, with three possible types, specific for RPL networks.
- These include a DAG Information Object (DIO), that allows a node to discover an RPL instance, configuration parameters and parents, a DAG Information Solicitation (DIS) to allow requests for DIOs from RPL nodes, and Destination Advertisement Object (DAO), used to propagate destination information upwards (i.e. towards the root) along the DODAG (specific RPL details are available in RFC 6550 and related RFCs).
- The Trickle Algorithm is an important enabler of RPL message exchange.
- CoAP (Constrained Application Protocol) is being developed by the IETF Constrained RESTful Environments (CoRE) WG as a specialized web transfer protocol for use with severe computational and communication constraints typically characteristic of M2M and IoT applications.

4.3 Data management

4.3.1 Introduction

- In the era of M2M, where billions of devices interact and generate data at exponential growth rates, data management is of critical importance as it sets the basis upon which any other processes can rely and operate
- Some of the key characteristics of M2M data include:

• **Big Data:** Huge amounts of data are generated, capturing detailed aspects of the processes where devices are involved.

• **Heterogeneous Data:** The data is produced by a huge variety of devices and is itself highly heterogeneous, differing on sampling rate, quality of captured values, etc.

• **Real-World Data:** The overwhelming majority of the M2M data relates to real-world processes and is dependent on the environment they interact with.

• **Real-Time Data:** M2M data is generated in real-time and overwhelmingly can be communicated also in a very timely manner.

• **Temporal Data:** The overwhelming majority of M2M data is of temporal nature, measuring the environment over time.

• **Spatial Data:** Increasingly, the data generated by M2M interactions are not only captured by mobile devices, but also coupled to interactions in specific locations, and their assessment may dynamically vary depending on the location.

• **Polymorphic Data:** The data acquired and used by M2M processes may be complex and involve various data, which can also obtain different meanings depending on the semantics applied and the process they participate in.

• **Proprietary Data:** Up to now, due to monolithic application development, a significant amount of M2M data is stored and captured in proprietary formats. However, increasingly due to the interactions with heterogeneous devices and stakeholders, open approaches for data storage and exchange are used.

• **Security and Privacy Data Aspects:** Due to the detailed capturing of interactions by M2M, analysis of the obtained data has a high risk of leaking private information and usage patterns, as well as compromising security.\

4.3.2 Managing M2M data

- The data flow from the moment it is sensed (e.g. by a wireless sensor node) up to the moment it reaches the backend system has been processed manifold (and often redundantly), either to adjust its representation in order to be easily integrated by the diverse applications, or to compute on it in order to extract and associate it with respective business intelligence (e.g. business process affected, etc.).

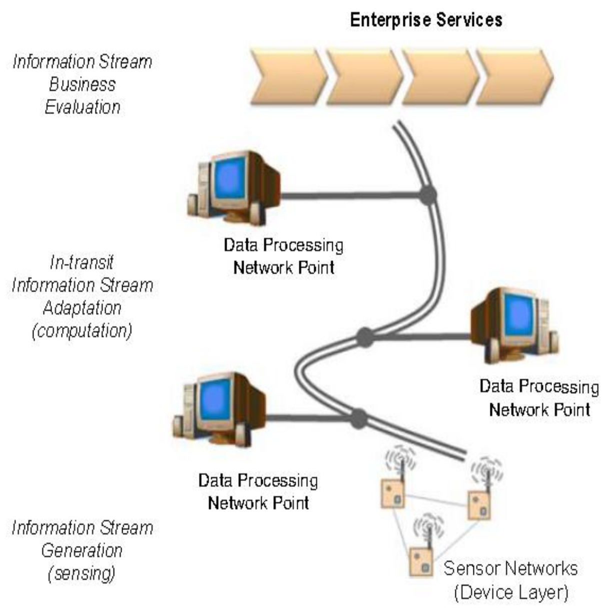

Enterprise Services
Information Stream Business Evaluation
Data Processing Network Point
In-transit Information Stream Adaptation (computation)
Data Processing Network Point
Data Processing Network Point
Information Stream Generation (sensing)
Sensor Networks (Device Layer)

FIGURE 5.5

M2M data from point of generation to business assessment.

- In Figure 5.5, we see a number of data processing network points between the machine and the enterprise that act on the datastream (or simply forwarding it) based on their end-application needs and existing context.
- Dealing with M2M data may be decomposed into several stages.
- Additionally, the degree of focus in each stage heavily depends on the actual usage requirements put upon the data as well as the infrastructure.

4.3.2.1 Data generation

- Data generation is the first stage within which data is generated actively or passively from the device, system, or as a result of its interactions.
- The sampling of data generation depends on the device and its capabilities as well as potentially the application needs.
- Usually default behaviors for data generation exist, which are usually further configurable to strike a good benefit between involved costs, e.g. frequency of data collection vs. energy used in the case of WSNs, etc.

4.3.2.2 Data acquisition

- Data acquisition deals with the collection of data (actively or passively) from the device, system, or as a result of its interactions.
- The data acquisition systems usually communicate with distributed devices over wired or wireless links to acquire the needed data, and need to respect security, protocol, and application requirements.
- The nature of acquisition varies, e.g. it could be continuous monitoring, interval-poll, event-based, etc.
- The frequency of data acquisition overwhelmingly depends on, or is customized by, the application requirements (or their common denominator).
- The data acquired at this stage (for non-closed local control loops) may also differ from the data actually generated.
- In simple scenarios, due to customized filters deployed at the device, a fraction of the generated data may be communicated.
- Data aggregation and even on-device computation of the data may result in communication of key performance indicators of interest to the application.

4.3.2.3 Data validation

- Data acquired must be checked for correctness and meaningfulness within the specific operating context.
- This is usually done based on rules, semantic annotations, or other logic.
- The acquired data may not conform to expectations and data may be intentionally or unintentionally corrupted during transmission, altered, or not make sense in the business context.
- As real-world processes depend on valid data to draw business-relevant decisions
- Several known methods are deployed for consistency and data type checking;
- for example, imposed range limits on the values acquired, logic checks, uniqueness, correct time-stamping, etc.
- In addition, semantics may play an increasing role here, as the same data may have different meanings in various operating contexts, and via semantics one can benefit while attempting to validate them.
- Another part of the validation may deal with fallback actions such as requesting the data again if checks fail, or attempts to “repair” partially failed data.
- Failure to validate may result in security breaches.
- Tampered-with data fed to an application is a well known security risk as its effects may lead to attacks on other services, privilege escalation, denial of service, database corruption, etc.

4.3.2.4 Data storage

- The data generated by M2M interactions is what is commonly referred to as "Big Data."
- Machines generate an incredible amount of information that is captured and needs to be stored for further processing.
- As this is proving challenging due to the size of information, a balance between its business usage vs. storage needs to be considered; that is, only the fraction of the data relevant to a business need may be stored for future reference.
- However, one has to carefully consider what the value of such data is to business not only in current processes, but also potentially other directions that may be followed in the future by the company as different assessments of the same data may provide other, hidden competitive advantages in the future.
- Due to the massive amounts of M2M data, as well as their envisioned processing (e.g. searching), specialized technologies such as massively parallel processing DBs, distributed file systems, cloud computing platforms, etc. are needed.

4.3.2.5 Data processing

- Data processing enables working with the data that is either at rest (already stored) or is in-motion (e.g. stream data).
- The scope of this processing is to operate on the data at a low level and "enhance" them for future needs.
- Typical examples include data adjustment during which it might be necessary to normalize data, introduce an estimate for a value that is missing, re-order incoming data by adjusting timestamps, etc.
- Similarly, aggregation of data or general calculation functions may be operated on two or more data streams and mathematical functions applied on their composition.
- Another example is the transformation of incoming data; for example, a stream can be converted on the fly (e.g. temperature values are converted from _F to _C), or repackaged in another data model, etc. Missing or invalid data that is needed for the specific time-slot may be forecasted and used until, in a future interaction, the actual data comes into the system.

4.3.2.6 Data remanence

- Even if the data is erased or removed, residues may still remain in electronic media, and may be easily recovered by third parties _ often referred to as data remanence.
- Several techniques have been developed to deal with this, such as overwriting, degaussing, encryption, and physical destruction.
- For M2M, not only the DBs where the M2M data is collected, but also the points of action, which generate the data, or the individual nodes in between, which may cache it.

- At the current technology pace, those buffers (e.g. on device) are expected to be less at risk since their limited size means that after a specific time has elapsed, new data will occupy that space; hence, the window of opportunity is rather small.
- In addition, for large-scale infrastructures the cost of potentially acquiring “deleted” data may be large; hence, their hubs or collection end-points, such as the DBs who have such low cost, may be more at risk.

4.3.2.7 Data analysis

- Data available in the repositories can be subjected to analysis with the aim to obtain the information they encapsulate and use it for supporting decision-making processes.
- The analysis of data at this stage heavily depends on the domain and the context of the data.
- For instance, business intelligence tools process the data with a focus on the aggregation and key performance indicator assessment.
- Data mining focuses on discovering knowledge, usually in conjunction with predictive goals.
- Statistics can also be used on the data to assess them quantitatively (descriptive statistics), find their main characteristics (exploratory data analysis), confirm a specific hypothesis (confirmatory data analysis), discover knowledge (data mining), and for machine learning, etc.
- This stage is the basis for any sophisticated applications that take advantage of the information hidden directly or indirectly on the data.

4.3.3 Considerations for M2M data

- The M2M infrastructure in place heavily depends on real-world processes, implying also that a big percentage of data will be generated by machines that interact with the real-world environment, while the rest will be purely virtual data.
- Many of the machines generating this data, which can then be communicated to others (e.g. analytics specialists).
- The end-beneficiaries might acquire information, but do not necessarily need to have access or to process the data by themselves.
- There is a rise of specialists in the various stages of M2M data management that will cooperate with application providers, users, etc. for the common benefit.
- Sharing of data and usage in multiple applications, security and trust are of key importance.
- Security is mandatory for enabling confidentiality, integrity, availability, authenticity, and nonrepudiation of data from the moment of generation to consumption.
- Due to the large-scale IoT infrastructure, heterogeneous devices, and stakeholders involved, this will be challenging.

- In addition, trust will be another major issue, as even if data is securely communicated or verified, the level of trust based on them will impact the decision-making process and risk analysis.
- Managing security and trust in the highly federated M2M-envisioned infrastructures poses a significant challenge, especially for mission critical applications that also exercise control.
- Privacy is also expected to be a significant issue in IoT infrastructures.
- Currently, a lot of emphasis is put on acquiring the data, and no real solutions exist for large-scale systems to share data in a controlled way.
- Once data is shared, the originator has no more control over its lifetime.
- A typical example here constitutes the usage of private citizen data, which could be controllably shared as wished; it should also be possible to (partially) revoke that right at will.
- Data Science in the IoT era is a cross-discipline approach building on mathematics, statistics, high-performance computing, modeling, machine learning, engineering, etc. that will play a key role in understanding the data, assessing their information at large scale, and hopefully enabling the better studying of complex systems of systems and their emergent characteristics.

4.3.4 Conclusions

- Data and its management hold the key to unveiling the true power of M2M and IoT.
- To do so, however, we have to think and develop approaches that go beyond simple data collection, and enable the management of their whole lifecycle at very large scale, while in parallel considering the special needs and the usage requirements posed by specific domains or applications.

4.4 Business processes in IoT

4.4.1 Introduction

- A business process refers to a series of activities, often a collection of interrelated processes in a logical sequence, within an enterprise, leading to a specific result.
- There are several types of business processes such as management, operational, and supporting, all of which aim at achieving a specific mission objective.
- As business processes usually span several systems and may get very complex, several methods and techniques have been developed for their modeling, such as the Business Process Model and Notation (BPMN), which graphically represents business processes in a business process model.

- Several key business processes in modern enterprise systems heavily rely on interaction with real-world processes, largely for monitoring, but also for some control (management), in order to take business-critical decisions and optimize actions across the enterprise.

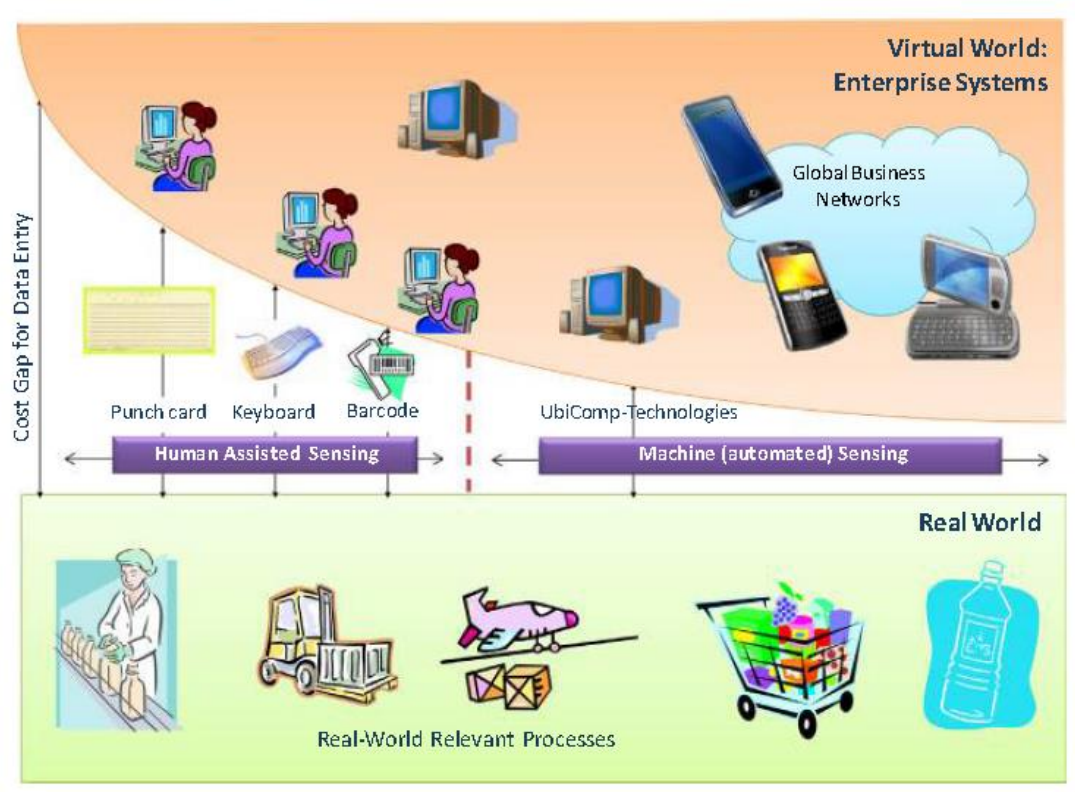


FIGURE 5.6

The decreasing cost of information exchange between the real-world and enterprise systems with the advancement of M2M.

- In Figure 5.6, the dramatic reduction of the data acquisition from the real world
- Initially all these interactions were human-based (e.g. via a keyboard) or human-assisted (e.g. via a barcode scanner); however, with the prevalence of RFID, WSNs, and advanced networked embedded devices, all information exchange between the real-world and enterprise systems can be done automatically without any human intervention and at blazing speeds.
- In the M2M era, connected devices can be clearly identified, and with the help of services, this integration leads to active participation of the devices to the business processes.
- Existing modeling tools are hardly designed to specify aspects of the real world in modeling environments and capture their full characteristics. To this direction, the existence of SOA-ready devices

- (i.e. devices that offer their functionalities as a web service) simplifies the integration and interaction as they can be considered as a traditional web service that runs on a specific device.
- A layered approach for developing, deploying, and managing WSN applications that natively interact with enterprise information systems such as a business process engine and the processes running therein is proposed and assessed.
- M2M and IoT empower business processes to acquire very detailed data about the operations, and be informed about the conditions in the real world in a very timely manner.

4.4.2 IoT integration with enterprise systems

- M2M communication and the vision of the IoT pose a new era where billions of devices will need to interact with each other and exchange information in order to fulfill their purpose.

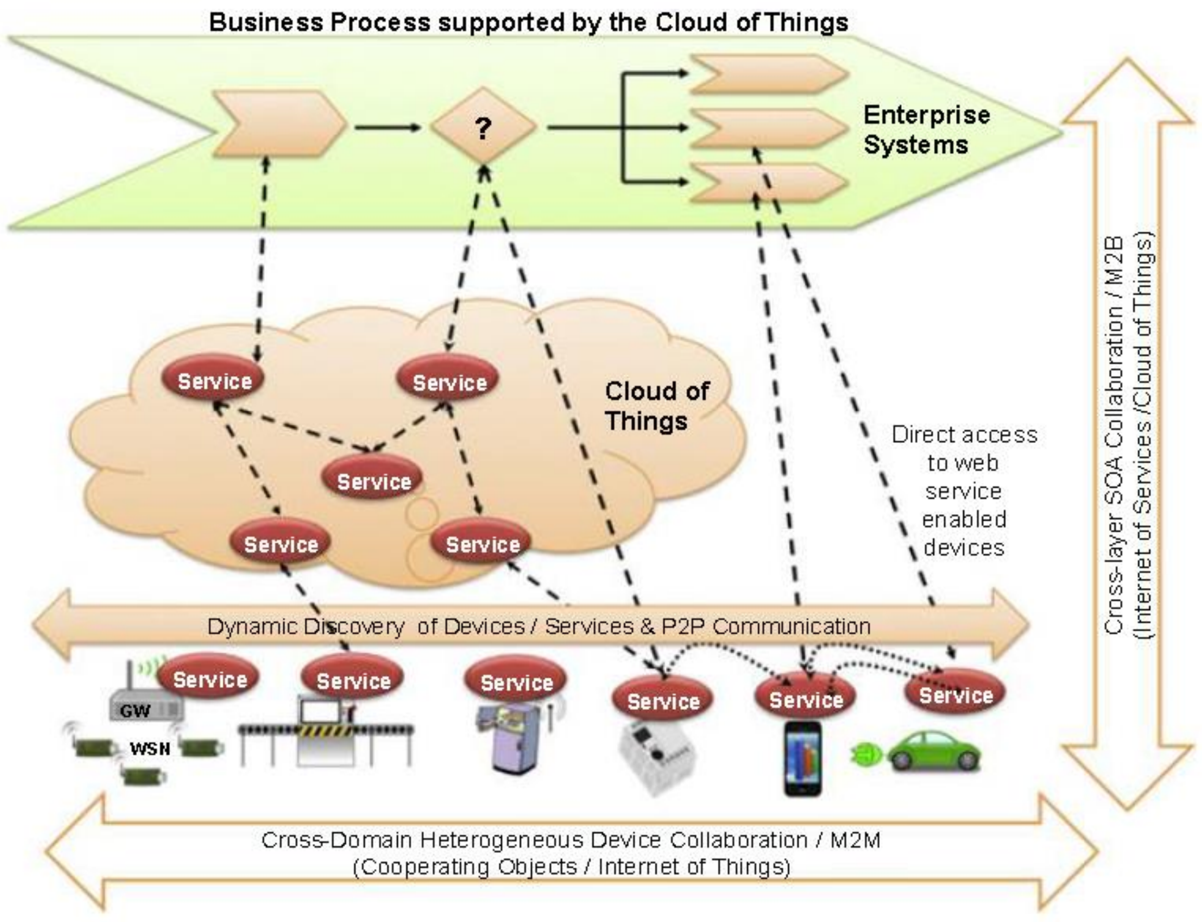

Business Process supported by the Cloud of Things
?
Enterprise Systems
Service
Cloud of Things
Direct access to web service enabled devices
Cross-layer SOA Collaboration / M2B (Internet of Services /Cloud of Things)
Dynamic Discovery of Devices / Services & P2P Communication
GW
WSN
Cross-Domain Heterogeneous Device Collaboration / M2M (Cooperating Objects / Internet of Things)

FIGURE 5.7

A collaborative infrastructure driven by M2M and M2B.

- In Figure 5.7, cross-layer interaction and cooperation can be pursued:

  • at the M2M level, where the machines cooperate with each other (machine-focused interactions)

  • at the machine-to-business (M2B) layer, where machines cooperate also with network-based services, business systems (business service focus), and applications.

- Several devices in the lowest layer. These can communicate with each other over short-range protocols (e.g. over ZigBee, Bluetooth), or even longer distances (e.g. over Wi-Fi, etc.).
- Some of them may host services (e.g. REST services), and even have dynamic discovery capabilities based on the communication protocol or other capabilities (e.g. WS-Eventing in DPWS).
- Some of them may be very resource constrained, which means that auxiliary gateways could provide additional support such as mediation of communication, protocol translation, etc.
- Independent of whether the devices are able to discover and interact with other devices and systems directly or via the support of the infrastructure, the M2M interactions enable them to empower several applications and interact with each other in order to fulfill their goals.
- Promising real-world integration is done using a service-oriented approach by interacting directly with the respective physical elements, for example, via web services running on devices (if supported) or via more lightweight approaches such as REST.
- Many of the services that will interact with the devices are expected to be network services available, for example, in the cloud.
- The main motivation for enterprise services is to take advantage of the cloud characteristics such as virtualization, scalability, multi-tenancy, performance, lifecycle
- management, etc.
- A key motivator is the minimization of communication overhead with multiple endpoints by, for example, transmission of data to a single or limited number of points in the network, and letting the cloud do the load balancing and further mediation of communication.
- Content Delivery Network (CDN) can be used in order to get access to the generated data from locations that are far away from the M2M infrastructure (geographically, network-wise, etc.).
- To this end, the data acquired by the device can be offered without overconsumption
- of the device's resources, while in parallel, better control and management can be applied.

<u>4.4.3 Distributed business processes in IoT</u>

- In Figure 5.9, the integration of devices in business processes merely implies the acquisition of data from the device layer, its transportation to the backend systems, its assessment, and once a decision is made, potentially the control (management) of the device, which adjusts its behavior.
- In future, due to the large scale of IoT, as well as the huge data that it will generate, such approaches are not viable.
- Enterprise systems trying to process such a high rate of non- or minor-relevancy data will be overloaded.

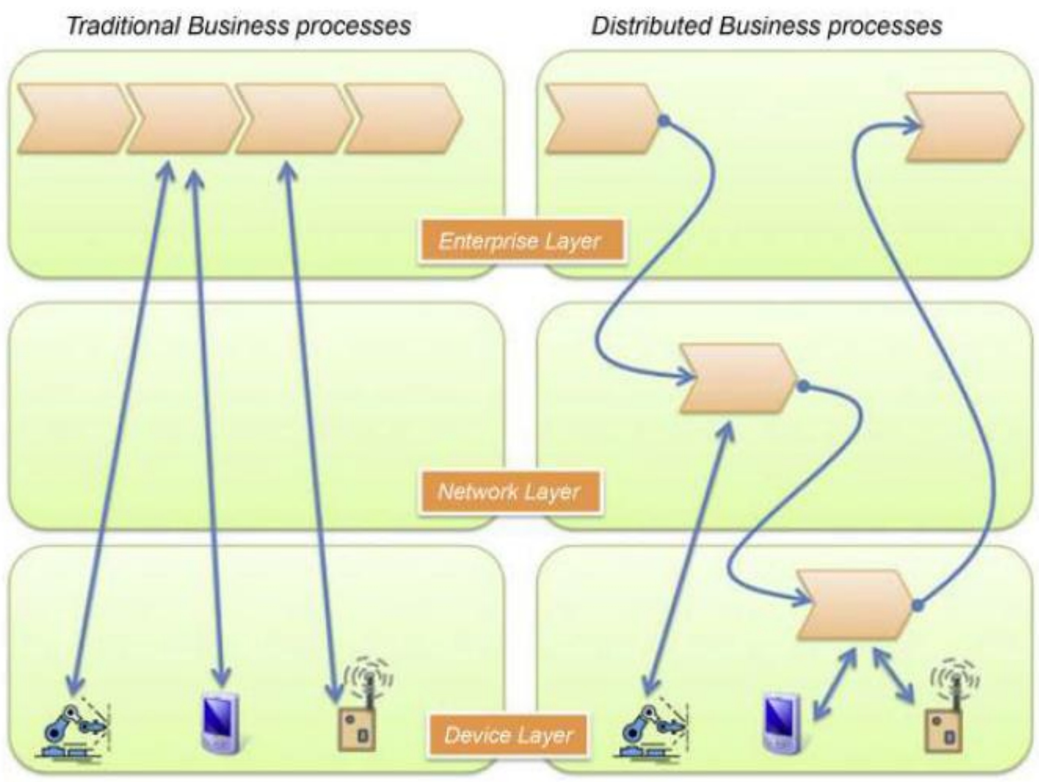

Traditional Business processes
Distributed Business processes
Enterprise Layer
Network Layer
Device Layer

FIGURE 5.9

Distributed Business Processes in M2M era.

- The first step is to minimize communication with enterprise systems to only what is relevant for business. With the increase
- in resources (e.g. computational capabilities) in the network, and especially on the devices themselves (more memory, multi-core CPUs, etc.), it makes sense not to host the intelligence and the computation required for it only on the enterprise side, but actually distribute it on the network, and even on the edge nodes (i.e. the devices themselves), as depicted on the right side of Figure 5.9.

- Partially outsourcing functionality traditionally residing in backend systems to the network itself and the edge nodes means we can realize distributed business processes whose sub-processes may execute outside the enterprise system.
- As devices are capable of computing, they can either realize the task of processing and evaluating business relevant information they generate by themselves or in clusters.
- Business processes can bind during execution of dynamic resources that they discover locally, and integrate them to better achieve their goals.

4.4.4 Considerations

- Existing tools and approaches need to be extended to the make the business processes IoT aware.
- Distributed execution of processes exists (e.g. in BPMN), additional work is needed to be able to select the devices in which such processes execute and consider their characteristics or dynamic resources, etc.
- The dynamic aspect is of key importance in the IoT, as this is mobile and availability is not guaranteed, which means that availability in modeling time does not guarantee availability at runtime and vice-versa.
- Scalability is an aspect that needs to be considered in the business process modeling and execution.
- In addition, event-based interactions among the processes play a key role in IoT, as a business process flow may be influenced by an event, or as its result, trigger a new event.

4.4.5 Conclusions

- Modern enterprises operate on a global scale and depend on complex business processes.
- Efficient information acquisition, evaluation, and interaction with the real world are of key importance.
- The infrastructure envisioned is a heterogeneous one, where millions of devices are interconnected, ready to receive instructions and create event notifications, and where the most advanced ones depict self-behavior (e.g. self-management, self-healing, selfoptimization, etc.) and collaborate.
- Business logic can now be intelligently distributed to several layers such as the network, or even the device layer, creating new opportunities, but also challenges that need to be assessed.
- Future Enterprise systems will be in position to better integrate state and events of the physical world in a timely manner, and hence to lead to more diverse, highly dynamic, and efficient business applications.

<u>4.5 Everything as a service (XaaS)</u>

- Cloud computing is a model for enabling ubiquitous, on-demand network access to a shared pool of configurable computing resources (e.g. networks, servers, storage, applications, and services) that can be provisioned, configured, and made available with minimal management effort or service provider interaction.
- All applications need access to three things: compute, storage, and data processing capacities.
- With cloud computing, a fourth element is added _ distribution services _ i.e. the manner in which the data and computational capacity are linked together and coordinated.

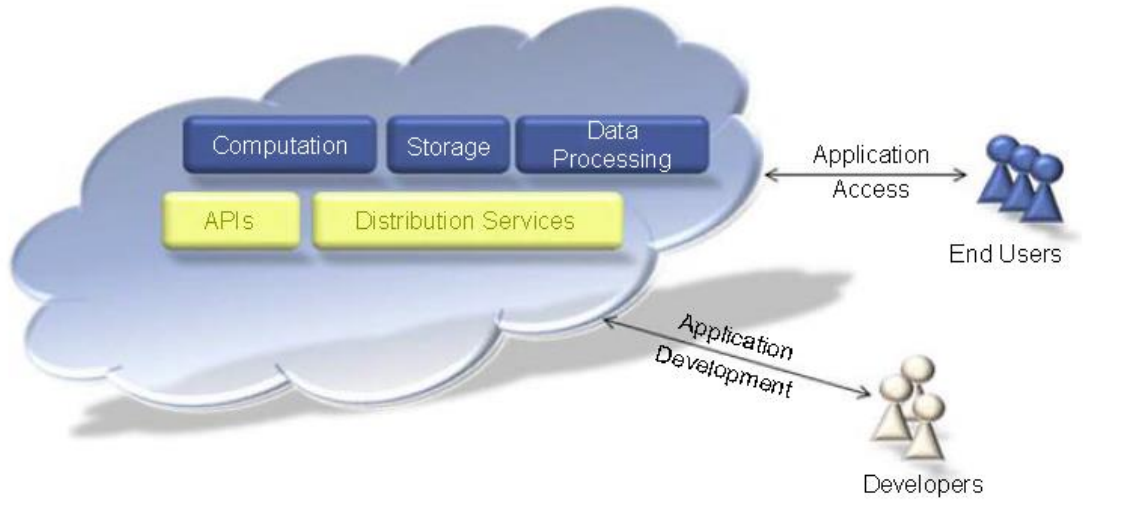


FIGURE 5.11

Conceptual Overview of Cloud Computing.

Characteristics of cloud computing

- On-Demand Self-Service.

  A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed, or automatically, without requiring human interaction with each service provider.
- Broad Network Access.

  Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms (e.g. mobile phones, tablets, laptops, and workstations).
- Resource Pooling.

  The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. Examples of resources include storage, processing, memory, and network bandwidth.

- Rapid Elasticity.

  Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand.

- Measured Service.
  - ✓ Cloud systems automatically control and optimize resource use by leveraging a metering capability, at some level of abstraction, appropriate to the type of service (e.g. storage, processing, bandwidth, and active user accounts).
  - ✓ Resource usage can be monitored, controlled, and reported, providing transparency for both the provider and consumer of the utilized service.

For M2M and IoT, these infrastructures provide the following:

1. Storage of the massive amounts of data that sensors, tags, and other "things" will produce.

2. Computational capacity in order to analyze data rapidly and cheaply.

3. Over time, cloud infrastructure will allow enterprises and developers to share datasets, allowing for rapid creation of information value chains.

7.1 What is an IoT Device

As described earlier, a "Thing" in Internet of Things (IoT) can be any object that has a unique identifier and which can send/receive data (including user data) over a network (e.g., smart phone, smart TV, computer, refrigerator, car, etc.). IoT devices are connected to the Internet and send information about themselves or about their surroundings (e.g. information sensed by the connected sensors) over a network (to other devices or servers/storage) or allow actuation upon the physical entities/environment around them remotely. Some examples of IoT devices are listed below:

- A home automation device that allows remotely monitoring the status of appliances and controlling the appliances.
- An industrial machine which sends information abouts its operation and health monitoring data to a server.
- A car which sends information about its location to a cloud-based service.
- A wireless-enabled wearable device that measures data about a person such as the number of steps walked and sends the data to a cloud-based service.

7.1.1 Basic building blocks of an IoT Device

An IoT device can consist of a number of modules based on functional attributes, such as:

- Sensing: Sensors can be either on-board the IoT device or attached to the device. IoT device can collect various types of information from the on-board or attached sensors such as temperature, humidity, light intensity, etc. The sensed information can be communicated either to other devices or cloud-based servers/storage.
- Actuation: IoT devices can have various types of actuators attached that allow taking actions upon the physical entities in the vicinity of the device. For example, a relay switch connected to an IoT device can turn an appliance on/off based on the commands sent to the device.
- Communication: Communication modules are responsible for sending collected data to other devices or cloud-based servers/storage and receiving data from other devices and commands from remote applications.
- Analysis & Processing: Analysis and processing modules are responsible for making sense of the collected data.

The representative IoT device used for the examples in this book is the widely used single-board mini computer called Raspberry Pi (explained in later sections). The use of Raspberry Pi is intentional since these devices are widely accessible, inexpensive, and available from multiple vendors. Furthermore, extensive information is available on their programming and use both on the Internet and in other textbooks. The principles we teach in this book are just as applicable to other (including proprietary) IoT endpoints, in addition to Raspberry Pi. Before we look at the specifics of Raspberry Pi, let us first look at the building blocks of a generic single-board computer (SBC) based IoT device.

Figure 7.1 shows a generic block diagram of a single-board computer (SBC) based IoT device that includes CPU, GPU, RAM, storage and various types of interfaces and peripherals.

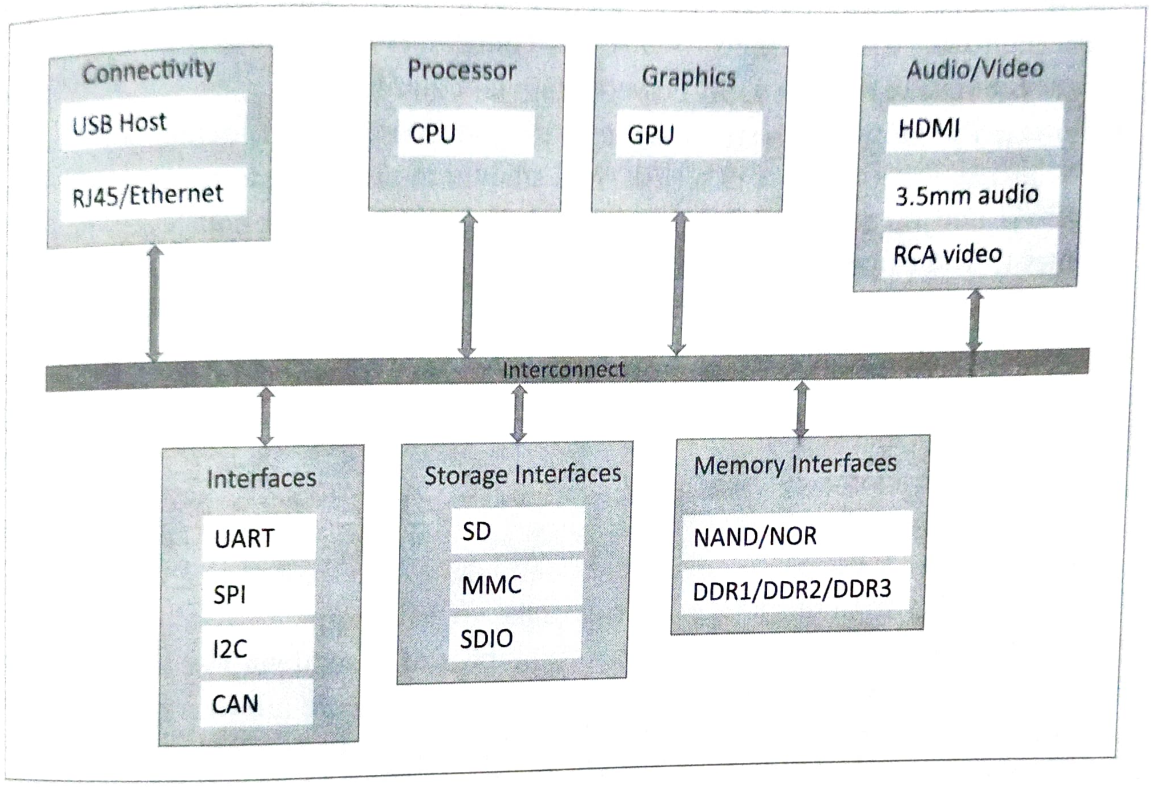


Figure 7.1: Block diagram of an IoT Device

7.2 Exemplary Device: Raspberry Pi

Raspberry Pi [104] is a low-cost mini-computer with the physical size of a credit card. Raspberry Pi runs various flavors of Linux and can perform almost all tasks that a normal desktop computer can do. In addition to this, Raspberry Pi also allows interfacing sensors and actuators through the general purpose I/O pins. Since Raspberry Pi runs Linux operating system, it supports Python "out of the box".

7.3 About the Board

Figure 7.2 shows the Raspberry Pi board with the various components/peripherals labeled.

- **Processor & RAM** : Raspberry Pi is based on an ARM processor. The latest version of Raspberry Pi (Model B, Revision 2) comes with 700 MHz Low Power ARM1176JZ-F processor and 512 MB SDRAM.
- **USB Ports** : Raspberry Pi comes with two USB 2.0 ports. The USB ports on Raspberry Pi can provide a current upto 100mA. For connecting devices that draw current more than 100mA, an external USB powered hub is required.
- **Ethernet Ports** : Raspberry Pi comes with a standard RJ45 Ethernet port. You can connect an Ethernet cable or a USB Wifi adapter to provide Internet connectivity.
- **HDMI Output** : The HDMI port on Raspberry Pi provides both video and audio output. You can connect the Raspberry Pi to a monitor using an HDMI cable. For monitors that have a DVI port but no HDMI port, you can use an HDMI to DVI adapter/cable.
- **Composite Video Output** : Raspberry Pi comes with a composite video output with an RCA jack that supports both PAL and NTSC video output. The RCA jack can be used to connect old televisions that have an RCA input only.
- **Audio Output** : Raspberry Pi has a 3.5mm audio output jack. This audio jack is used for providing audio output to old televisions along with the RCA jack for video. The audio quality from this jack is inferior to the HDMI output.
- **GPIO Pins** : Raspberry Pi comes with a number of general purpose input/ouput pins. Figure 7.3 shows the Raspberry Pi GPIO headers. There are four types of pins on Raspberry Pi - true GPIO pins, I2C interface pins, SPI interface pins and serial Rx and Tx pins.
- **Display Serial Interface (DSI)** : The DSI interface can be used to connect an LCD panel to Raspberry Pi.
- **Camera Serial Interface (CSI)** : The CSI interface can be used to connect a camera module to Raspberry Pi.
- **Status LEDs** : Raspberry Pi has five status LEDs. Table 7.1 lists Raspberry Pi status LEDs and their functions.
- **SD Card Slot** : Raspberry Pi does not have a built in operating system and storage. You can plug-in an SD card loaded with a Linux image to the SD card slot. Appendix-A provides instructions on setting up New Out-of-the-Box Software (NOOBS) on Raspberry Pi. You will require atleast an 8GB SD card for setting up NOOBS.
- **Power Input** : Raspberry Pi has a micro-USB connector for power input.

Status LED	Function
ACT	SD card access
PWR	3.3V Power is present
FDX	Full duplex LAN connected
LNK	Link/Network activity
100	100 Mbit LAN connected

Table 7.1: Raspberry Pi Status LEDs

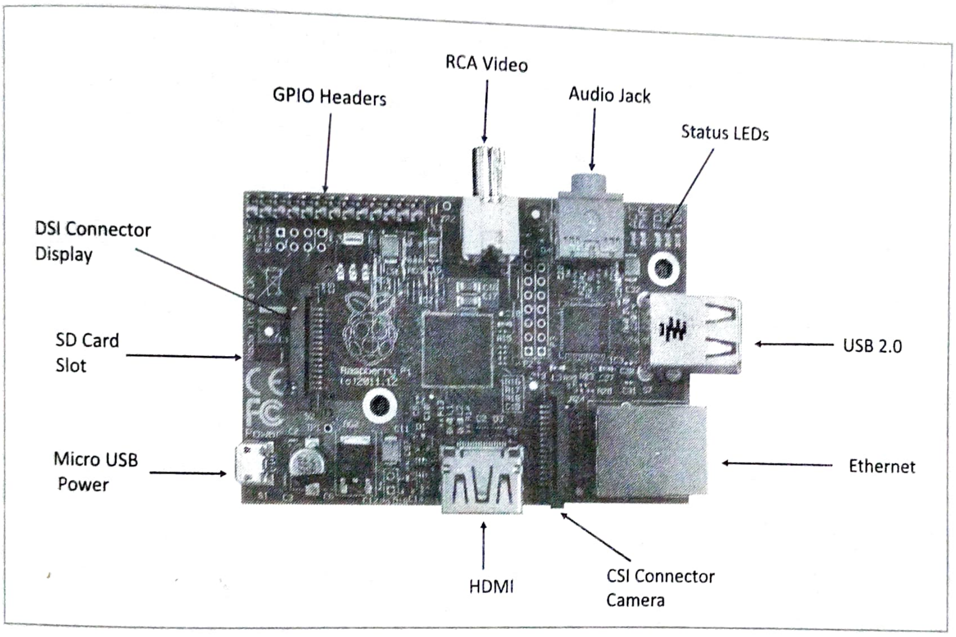


Figure 7.2: Raspberry Pi board

7.4 Linux on Raspberry Pi

Raspberry Pi supports various flavors of Linux including:

- **Raspbian** Raspbian Linux is a Debian Wheezy port optimized for Raspberry Pi. This is the recommended Linux for Raspberry Pi. Appendix-1 provides instructions on setting up Raspbian on Raspberry Pi.
- **Arch** : Arch is an Arch Linux port for AMD devices.
- **Pidora** : Pidora Linux is a Fedora Linux optimized for Raspberry Pi.
- **RaspBMC** : RaspBMC is an XBMC media-center distribution for Raspberry Pi.
- **OpenELEC** : OpenELEC is a fast and user-friendly XBMC media-center distribution.
- **RISC OS** : RISC OS is a very fast and compact operating system.

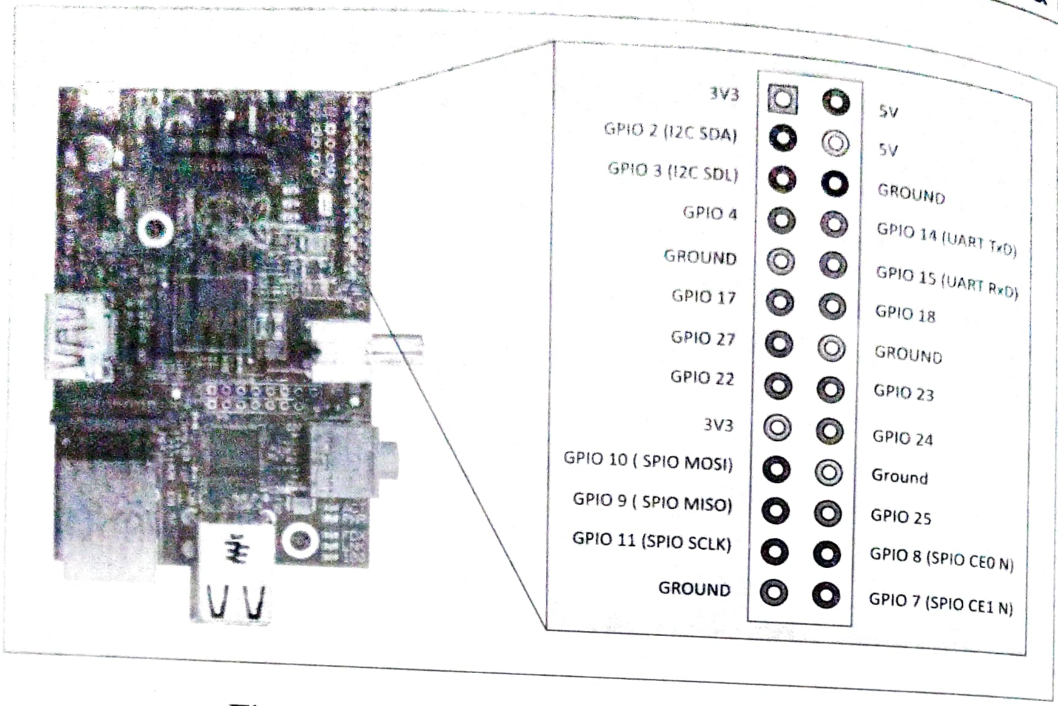


Figure 7.3: Raspberry Pi GPIO headers

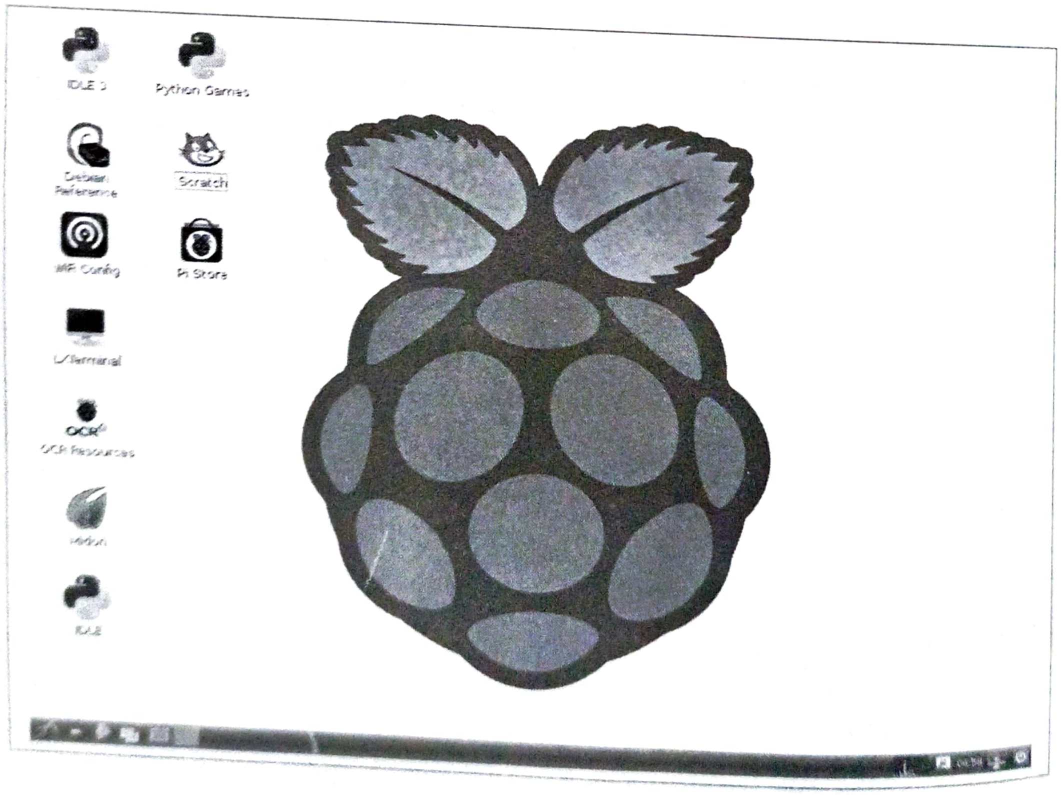


Figure 7.4: Rasbian Linux desktop

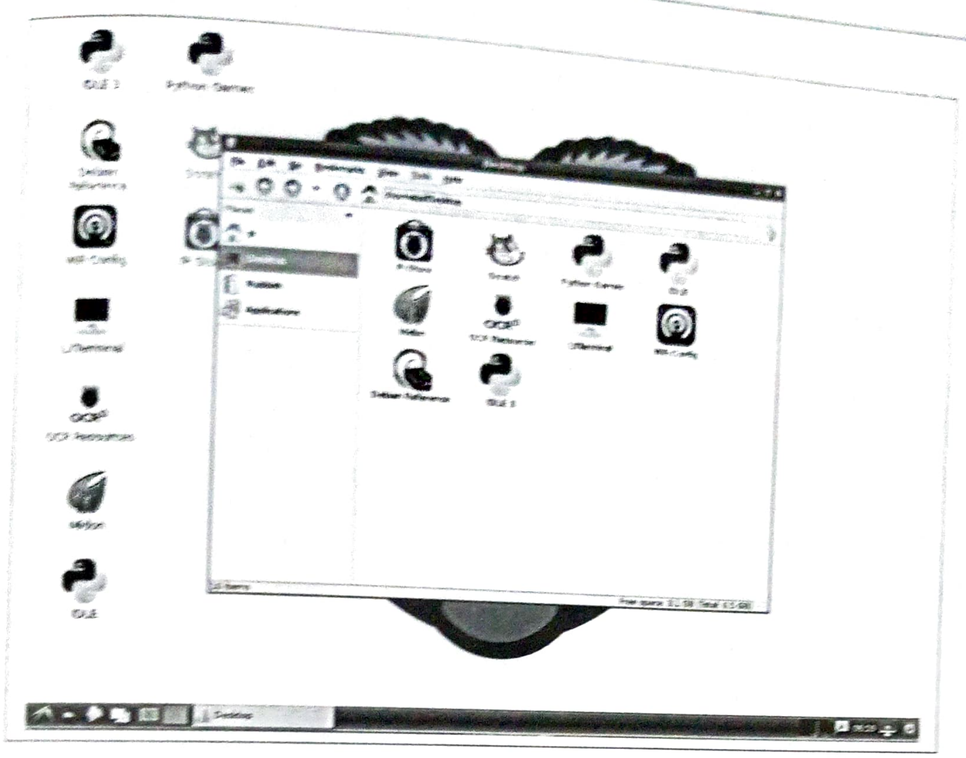
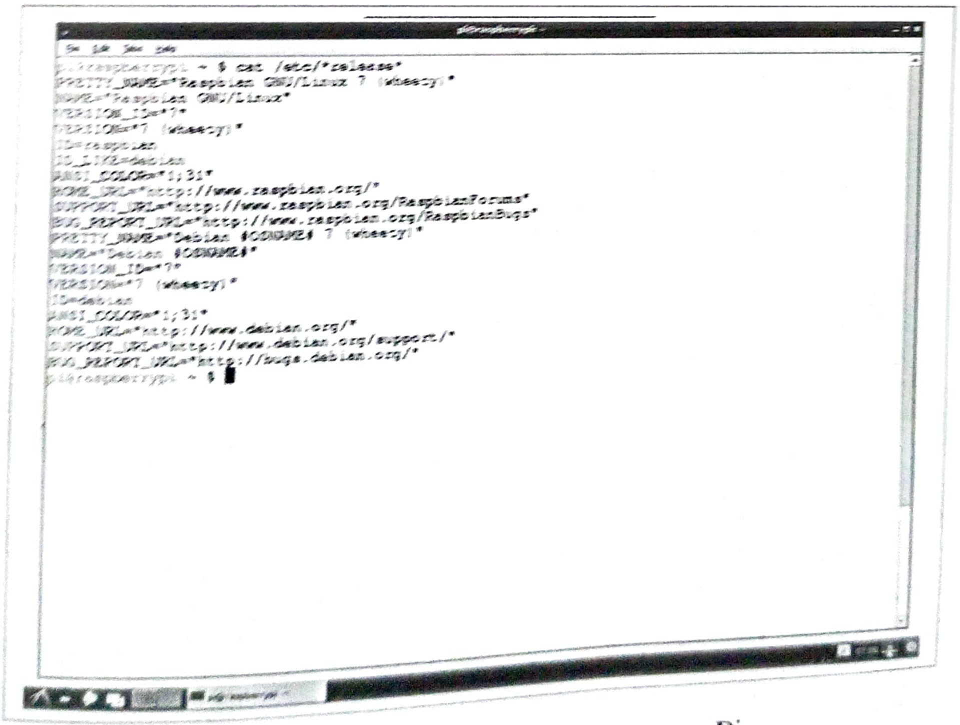
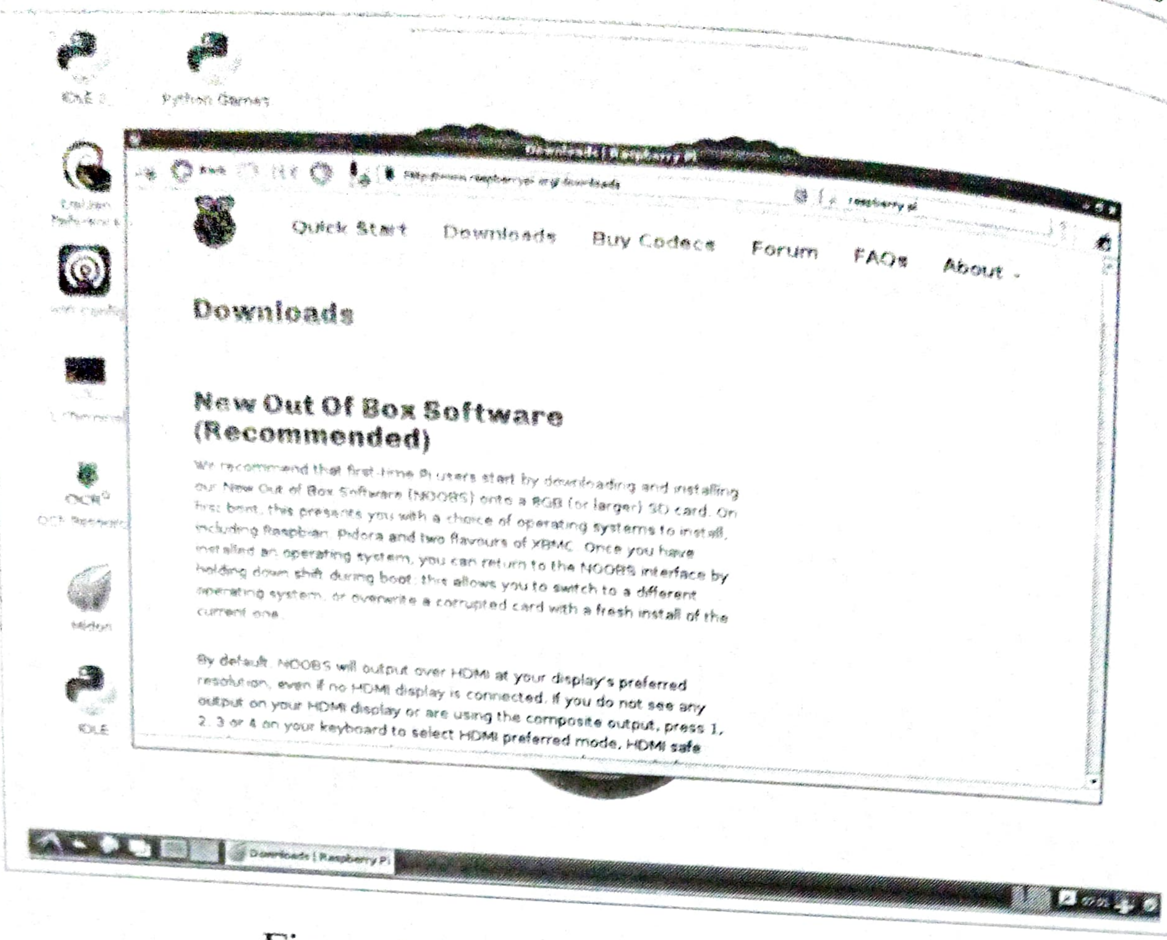


Figure 7.5: File explorer on Raspberry Pi



```
pi@raspberrypi ~ $ cat /etc/*release*
PRETTY_NAME="Raspbian GNU/Linux 7 (wheezy)"
NAME="Raspbian GNU/Linux"
VERSION_ID="7"
VERSION="7 (wheezy)"
ID=raspbian
ID_LIKE=debian
ANSI_COLOR="1;31"
HOME_URL="http://www.raspbian.org/"
SUPPORT_URL="http://www.raspbian.org/RaspbianForums"
BUG_REPORT_URL="http://www.raspbian.org/RaspbianBugs"
PRETTY_NAME="Debian $CODNAME$ 7 (wheezy)"
NAME="Debian $CODNAME$"
VERSION_ID="7"
VERSION="7 (wheezy)"
ID=debian
ANSI_COLOR="1;31"
HOME_URL="http://www.debian.org/"
SUPPORT_URL="http://www.debian.org/support/"
BUG_REPORT_URL="http://bugs.debian.org/"
pi@raspberrypi ~ $
```

Figure 7.6: Console on Raspberry Pi



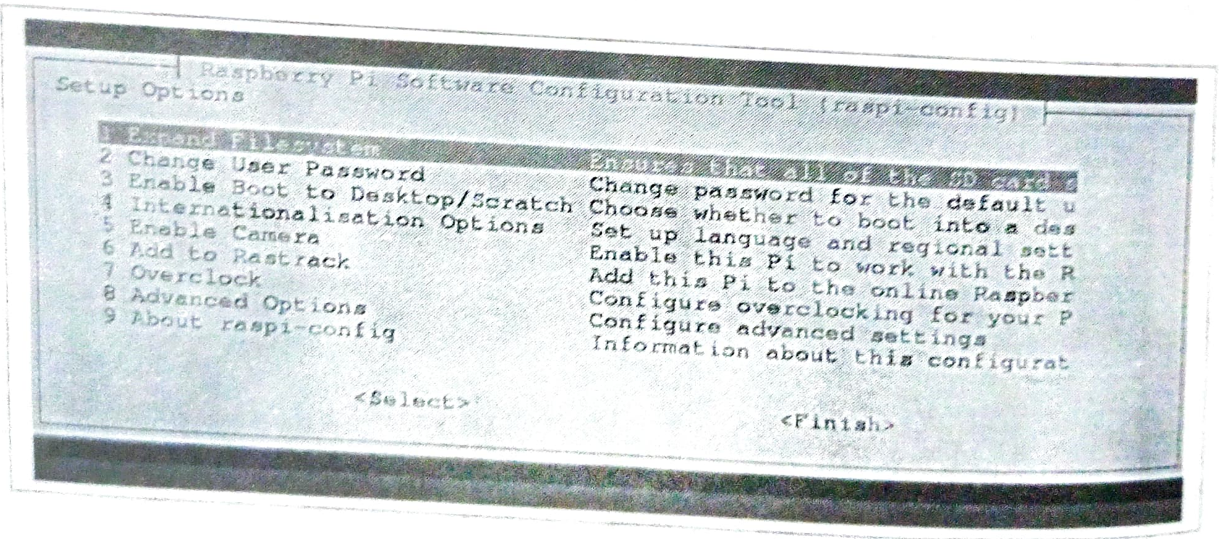
Downloads

New Out Of Box Software (Recommended)

We recommend that first-time Pi users start by downloading and installing our New Out of Box Software (NOOBS) onto a 8GB (or larger) SD card. On first boot, this presents you with a choice of operating systems to install, including Raspbian, Pidora and two flavours of XBMC. Once you have installed an operating system, you can return to the NOOBS interface by holding down shift during boot: this allows you to switch to a different operating system, or overwrite a corrupted card with a fresh install of the current one.

By default, NOOBS will output over HDMI at your display's preferred resolution, even if no HDMI display is connected. If you do not see any output on your HDMI display or are using the composite output, press 1, 2, 3 or 4 on your keyboard to select HDMI preferred mode, HDMI safe

Figure 7.7: Browser on Raspberry Pi



```
Raspberry Pi Software Configuration Tool (raspi-config)
Setup Options

 1 Expand Filesystem              Ensures that all of the SD card s
 2 Change User Password           Change password for the default u
 3 Enable Boot to Desktop/Scratch Choose whether to boot into a des
 4 Internationalisation Options   Set up language and regional sett
 5 Enable Camera                  Enable this Pi to work with the R
 6 Add to Rastrack                Add this Pi to the online Raspber
 7 Overclock                      Configure overclocking for your P
 8 Advanced Options               Configure advanced settings
 9 About raspi-config             Information about this configurat

                 <Select>                       <Finish>
```

Figure 7.8: Raspberry Pi configuration tool

Figure 7.4 shows the Raspbian Linux desktop on Raspberry Pi. Figure 7.5 sh the default file explorer on Raspbian. Figure 7.6 shows the default console on Raspb Figure 7.7 shows the default browser on Raspbian. To configure Raspberry Pi, the raspi-con tool is used which can be launched from command line as ($raspi-config) as shown in Figure 7.8. Using the configuration tool you can expand root partition to fill SD card, set keyboard layout, change password, set locale and timezone, change memory split, enable or disable SSH server and change boot behavior. It is recommended to expand the root file-system so that you can use the entire space on the SD card.

Though Raspberry Pi comes with an HDMI output, it is more convenient to access the device with a VNC connection or SSH. This does away with the need for a separate display for Raspberry Pi and you can use Raspberry Pi from your desktop or laptop computer. Appendix-A provides instructions on setting up VNC server on Raspberry Pi and the instructions to connect to Raspberry Pi with SSH. Table 7.2 lists the frequently used commands on Raspberry Pi.

Command	Function	Example
cd	Change directory	cd /home/pi
cat	Show file contents	cat file.txt
ls	List files and folders	ls /home/pi
locate	Search for a file	locate file.txt
lsusb	List USB devices	lsusb
pwd	Print name of present working directory	pwd
mkdir	Make directory	mkdir /home/pi/new
mv	Move (rename) file	mv sourceFile.txt destinationFile.txt
rm	Remove file	rm file.txt
reboot	Reboot device	sudo reboot
shutdown	Shutdown device	sudo shutdown -h now
grep	Print lines matching a pattern	grep -r “pi” /home/
df	Report file system disk space usage	df -Th
ifconfig	Configure a network interface	ifconfig
netstat	Print network connections, routing tables, interface statistics	netstat -lntp
tar	Extract/create archive	tar -xzf foo.tar.gz
wget	Non-interactive network downloader	wget http://example.com/file.tar.gz

Table 7.2: Raspberry Pi frequently used commands

7.5 Raspberry Pi Interfaces

Raspberry Pi has serial, SPI and I2C interfaces for data transfer as shown in Figure 7.3.

7.5.1 Serial

The serial interface on Raspberry Pi has receive (Rx) and transmit (Tx) pins for communication with serial peripherals.

7.5.2 SPI

Serial Peripheral Interface (SPI) is a synchronous serial data protocol used for communicating with one or more peripheral devices. In an SPI connection, there is one master device and one or more peripheral devices. There are five pins on Raspberry Pi for SPI interface:

- **MISO (Master In Slave Out)** : Master line for sending data to the peripherals.
- **MOSI (Master Out Slave In)** : Slave line for sending data to the master.
- **SCK (Serial Clock)** : Clock generated by master to synchronize data transmission
- **CE0 (Chip Enable 0)** : To enable or disable devices.
- **CE0 (Chip Enable 1)** : To enable or disable devices.

7.5.3 I2C

The I2C interface pins on Raspberry Pi allow you to connect hardware modules. I2C interface allows synchronous data transfer with just two pins - SDA (data line) and SCL (clock line).

7.6 Programming Raspberry Pi with Python

In this section you will learn how to get started with developing Python programs on Raspberry Pi. Raspberry Pi runs Linux and supports Python out of the box. Therefore, you can run any Python program that runs on a normal computer. However, it is the general purpose input/output capability provided by the GPIO pins on Raspberry Pi that makes it useful device for Internet of Things. You can interface a wide variety of sensor and actuators with Raspberry Pi using the GPIO pins and the SPI, I2C and serial interfaces. Input from the sensors connected to Raspberry Pi can be processed and various actions can be taken, for instance, sending data to a server, sending an email, triggering a relay switch.

7.6.1 Controlling LED with Raspberry Pi

Let us start with a basic example of controlling an LED from Raspberry Pi. Figure 7.9 shows the schematic diagram of connecting an LED to Raspberry Pi. Box 7.1 shows how to turn the LED on/off from command line. In this example the LED is connected to GPIO pin 18. You can connect the LED to any other GPIO pin as well.

Box 7.2 shows a Python program for blinking an LED connected to Raspberry Pi every second. The program uses the RPi. GPIO module to control the GPIO on Raspberry Pi. In this program we set pin 18 direction to output and then write *True*/*False* alternatively after a delay of one second.

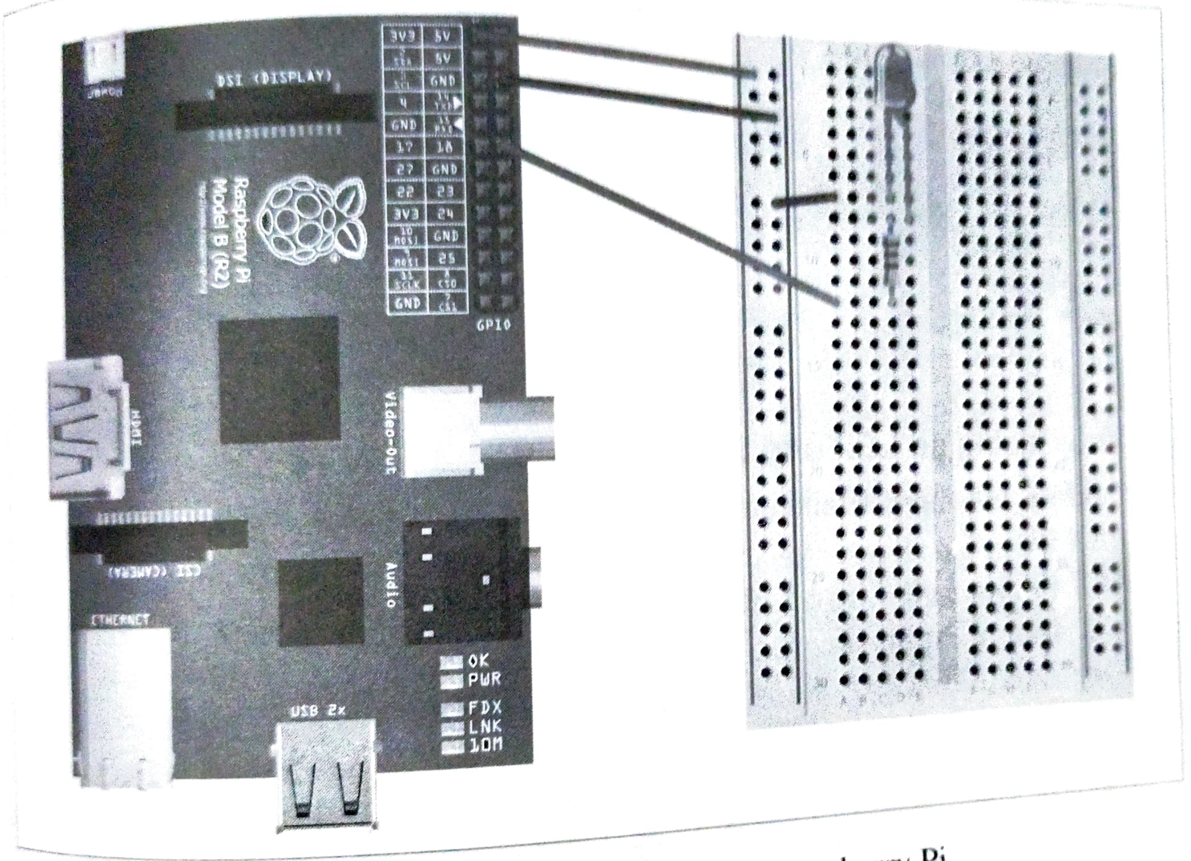


Figure 7.9: Controlling LED with Raspberry Pi

Box 7.1: Switching LED on/off from Raspberry Pi console

```
$echo 18 > /sys/class/gpio/export
$cd /sys/class/gpio/gpio18

#Set pin 18 direction to out
$echo out > direction

#Turn LED on
$echo 1 > value
```

```
#Turn LED off
$echo 0 > value
```

■ **Box 7.2: Python program for blinking LED**

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
GPIO.setup(18, GPIO.OUT)

while True:
    GPIO.output(18, True)
    time.sleep(1)
    GPIO.output(18, False)
    time.sleep(1)
```

7.6.2 Interfacing an LED and Switch with Raspberry Pi

Now let us look at a more detailed example involving an LED and a switch that is used to control the LED.

Figure 7.10 shows the schematic diagram of connecting an LED and switch to Raspberry Pi. Box 7.3 shows a Python program for controlling an LED with a switch. In this example the LED is connected to GPIO pin 18 and switch is connected to pin 25. In the infinite while loop the value of pin 25 is checked and the state of LED is toggled if the switch is pressed. This example shows how to get input from GPIO pins and process the input and take some action. The action in this example is toggling the state of an LED. Let us look at another example, in which the action is an email alert. Box 7.4 shows a Python program for sending an email on switch press. Note that the structure of this program is similar to the program in Box 7.3. This program uses the Python SMTP library for sending an email when the switch connected to Raspberry Pi is pressed.

■ **Box 7.3: Python program for controlling an LED with a switch**

```
from time import sleep
import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BCM)
```

```
#Switch Pin
GPIO.setup(25, GPIO.IN)

#LED Pin
GPIO.setup(18, GPIO.OUT)

state=false

def toggleLED(pin):
  state = not state
  GPIO.output(pin, state)

while True:
  try:
    if (GPIO.input(25) == True):
      toggleLED(pin)
    sleep(.01)
  except KeyboardInterrupt:
    exit()
```

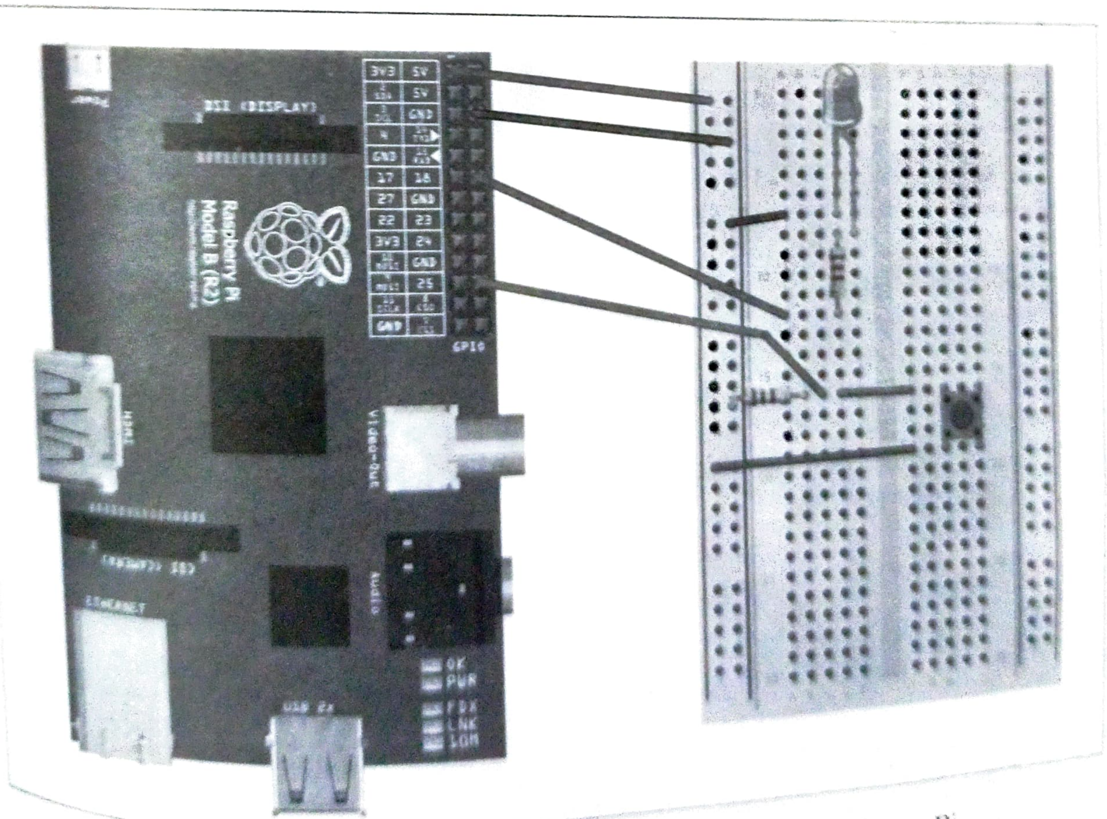


Figure 7.10: Interfacing LED and switch with Raspberry Pi

■ Box 7.4: Python program for sending an email on switch press

```
import smtplib
from time import sleep
import RPi.GPIO as GPIO
from sys import exit

from_email = '<my-email>'
receipients_list = ['<receipient-email>']
cc_list = []
subject = 'Hello'
message = 'Switch pressed on Raspberry Pi'
username = '<Gmail-username>'
password = '<password>'
server = 'smtp.gmail.com:587'

GPIO.setmode(GPIO.BCM)
GPIO.setup(25, GPIO.IN)


def sendemail(from_addr, to_addr_list, cc_addr_list,
      subject, message,
      login, password,
      smtpserver):

   header = 'From:  %s \n' % from_addr
   header += 'To:  %s \n' % ','.join(to_addr_list)
   header += 'Cc:  %s \n' % ','.join(cc_addr_list)
   header += 'Subject:  %s \n \n' % subject
   message = header + message

   server = smtplib.SMTP(smtpserver)
   server.starttls()
   server.login(login,password)
   problems = server.sendmail(from_addr, to_addr_list, message)
   server.quit()

while True:
   try:
      if (GPIO.input(25) == True):
         sendemail(from_email, receipients_list,
            cc_list, subject, message,
            username, password, server)
      sleep(.01)
```

```
    except KeyboardInterrupt:
        exit()
```

7.6.3 Interfacing a Light Sensor (LDR) with Raspberry Pi

So far you have learned how to interface LED and switch with Raspberry Pi. Now let us look at an example of interfacing a Light Dependent Resistor (LDR) with Raspberry Pi and turning an LED on/off based on the light-level sensed.

Figure 7.11 shows the schematic diagram of connecting an LDR to Raspberry Pi. Connect one side of LDR to 3.3V and other side to a 1μF capacitor and also to a GPIO pin (pin 18 in this example). An LED is connected to pin 18 which is controlled based on the light-level sensed.

Box 7.5 shows the Python program for the LDR example. The *readLDR*() function returns a count which is proportional to the light level. In this function the LDR pin is set to output and low and then to input. At this point the capacitor starts charging through the resistor (and a counter is started) until the input pin reads high (this happens when capacitor voltage becomes greater than 1.4V). The counter is stopped when the input reads high. The final count is proportional to the light level as greater the amount of light, smaller is the LDR resistance and greater is the time taken to charge the capacitor.

▪ Box 7.5: Python program for switching LED/Light based on reading LDR reading

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
ldr_threshold = 1000
LDR_PIN = 18
LIGHT_PIN = 25

def readLDR(PIN):
    reading=0
    GPIO.setup(LIGHT_PIN, GPIO.OUT)
    GPIO.output(PIN, False)
    time.sleep(0.1)
    GPIO.setup(PIN, GPIO.IN)
    while (GPIO.input(PIN)==False):
        reading=reading+1
    return reading
```

[illegible]

```
def switchOnLight(PIN):
    GPIO.setup(PIN, GPIO.OUT)
    GPIO.output(PIN, True)

def switchOffLight(PIN):
    GPIO.setup(PIN, GPIO.OUT)
    GPIO.output(PIN, False)


while True:
    ldr_reading = readLDR(LDR_PIN)
    if ldr_reading < ldr_threshold:
        switchOnLight(LIGHT_PIN)
    else:
        switchOffLight(LIGHT_PIN)

    time.sleep(1)
```

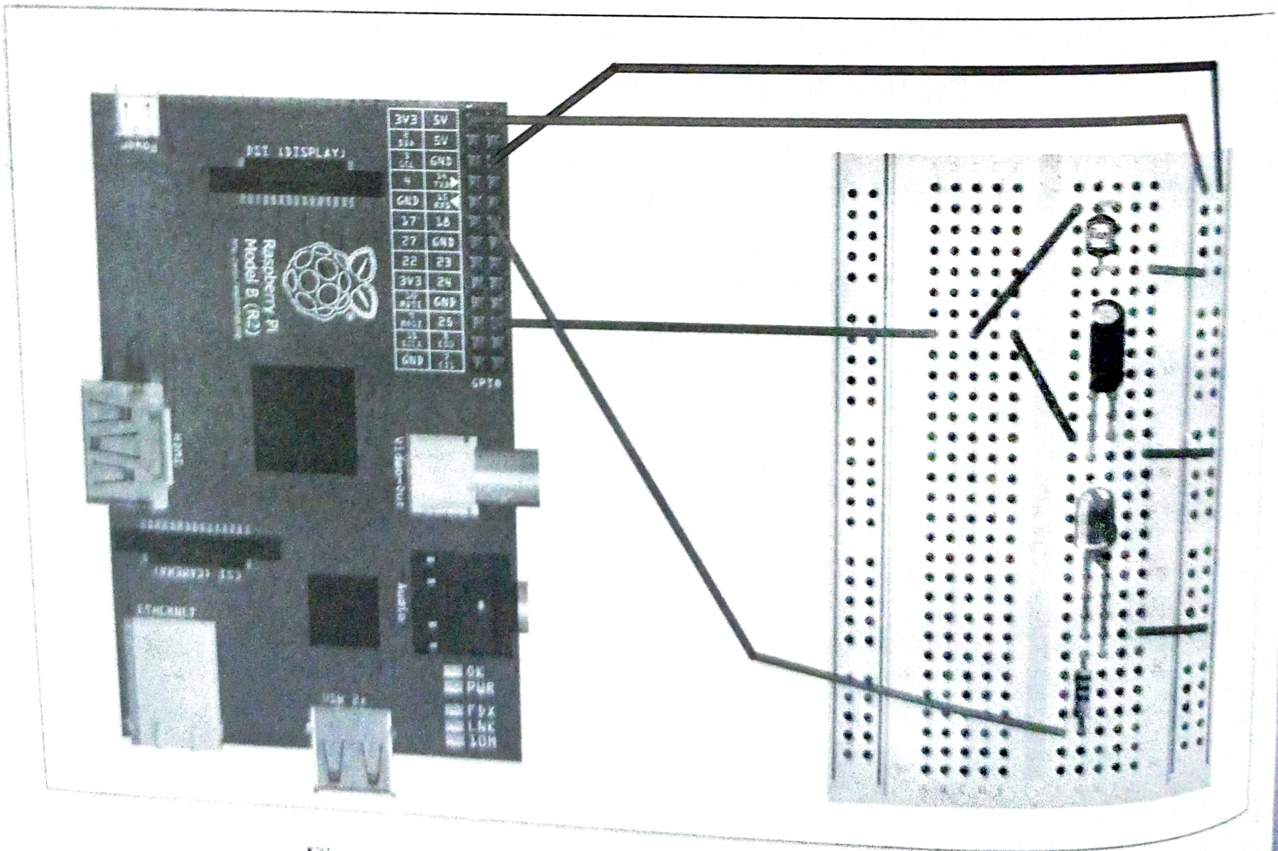


Figure 7.11: Interfacing LDR with Raspberry Pi

Transport Layer

The transport layer protocols provide end-to-end message transfer capability independent of the underlying network. The message transfer capability can be set up on connections, either using handshakes (as in TCP) or without handshakes/acknowledgements (as in UDP). The transport layer provides functions such as error control, segmentation, flow control and congestion control.

- **TCP :** Transmission Control Protocol (TCP) is the most widely used transport layer protocol, that is used by web browsers (along with HTTP, HTTPS application layer protocols), email programs (SMTP application layer protocol) and file transfer (FTP). TCP is a connection oriented and stateful protocol. While IP protocol deals with sending packets, TCP ensures reliable transmission of packets in-order. TCP also provides error detection capability so that duplicate packets can be discarded and lost packets are retransmitted. The flow control capability of TCP ensures that rate at which the sender sends the data is not too high for the receiver to process. The congestion control capability of TCP helps in avoiding network congestion and congestion collapse which can lead to degradation of network performance. TCP is described in RFC 793 [9].

- **UDP :** Unlike TCP, which requires carrying out an initial setup procedure, UDP is a connectionless protocol. UDP is useful for time-sensitive applications that have very small data units to exchange and do not want the overhead of connection setup. UDP is a transaction oriented and stateless protocol. UDP does not provide guaranteed delivery, ordering of messages and duplicate elimination. Higher levels of protocols can ensure reliable delivery or ensuring connections created are reliable. UDP is described in RFC 768 [10].

Application Layer

Application layer protocols define how the applications interface with the lower layer protocols to send the data over the network. The application data, typically in files, is encoded by the application layer protocol and encapsulated in the transport layer protocol which provides connection or transaction oriented communication over the network. Port numbers are used for application addressing (for example port 80 for HTTP, port 22 for SSH, etc.). Application layer protocols enable process-to-process connections using ports.

- **HTTP :** Hypertext Transfer Protocol (HTTP) is the application layer protocol that forms the foundation of the World Wide Web (WWW). HTTP includes commands such as GET, PUT, POST, DELETE, HEAD, TRACE, OPTIONS, etc. The protocol follows a request-response model where a client sends requests to a server using the HTTP commands. HTTP is a stateless protocol and each HTTP request is independent of the other requests. An HTTP client can be a browser or an application running on the client (e.g., an application running on an IoT device, a mobile application or other software). HTTP protocol uses Universal Resource Identifiers (URIs) to identify HTTP resources. HTTP is described in RFC 2616 [11].
- **CoAP :** Constrained Application Protocol (CoAP) is an application layer protocol for machine-to-machine (M2M) applications, meant for constrained environments with constrained devices and constrained networks. Like HTTP, CoAP is a web transfer protocol and uses a request-response model, however it runs on top of UDP instead of TCP. CoAP uses a client-server architecture where clients communicate with servers using connectionless datagrams. CoAP is designed to easily interface with HTTP. Like HTTP, CoAP supports methods such as GET, PUT, POST, and DELETE. CoAP draft specifications are available on IEFT Constrained environments (CoRE) Working Group website [12].

- **WebSocket :** WebSocket protocol allows full-duplex communication over a single socket connection for sending messages between client and server. WebSocket is based on TCP and allows streams of messages to be sent back and forth between the client and server while keeping the TCP connection open. The client can be a browser, a mobile application or an IoT device. WebSocket is described in RFC 6455 [13].
- **MQTT :** Message Queue Telemetry Transport (MQTT) is a light-weight messaging protocol based on the publish-subscribe model. MQTT uses a client-server architecture where the client (such as an IoT device) connects to the server (also called MQTT Broker) and publishes messages to topics on the server. The broker forwards the messages to the clients subscribed to topics. MQTT is well suited for constrained environments where the devices have limited processing and memory resources and the network bandwidth is low. MQTT specifications are available on IBM developerWorks [14].
- **XMPP :** Extensible Messaging and Presence Protocol (XMPP) is a protocol for real-time communication and streaming XML data between network entities. XMPP powers wide range of applications including messaging, presence, data syndication, gaming, multi-party chat and voice/video calls. XMPP allows sending small chunks of XML data from one network entity to another in near real-time. XMPP s a decentralized protocol and uses a client-server architecture. XMPP supports both client-to-server and server-to-server communication paths. In the context of IoT, XMPP allows real-time communication between IoT devices. XMPP is described in RFC 6120 [15].
- **DDS :** Data Distribution Service (DDS) is a data-centric middleware standard for device-to-device or machine-to-machine communication. DDS uses a publish-subscribe model where publishers (e.g. devices that generate data) create topics to which subscribers (e.g., devices that want to consume data) can subscribe. Publisher is an object responsible for data distribution and the subscriber is responsible for receiving published data. DDS provides quality-of-service (QoS) control and configurable reliability. DDS is described in Object Management Group (OMG) DDS specification [16].
- **AMQP :** Advanced Message Queuing Protocol (AMQP) is an open application layer protocol for business messaging. AMQP supports both point-to-point and publisher/subscriber models, routing and queuing. AMQP brokers receive messages from publishers (e.g., devices or applications that generate data) and route them over connections to consumers (applications that process data). Publishers publish the messages to exchanges which then distribute message copies to queues. Messages are either delivered by the broker to the consumers which have subscribed to the queues or the consumers can pull the messages from the queues. AMQP specification is

- **CoAP :** Constrained Application Protocol (CoAP) is an application layer protocol for machine-to-machine (M2M) applications, meant for constrained environments with constrained devices and constrained networks. Like HTTP, CoAP is a web transfer protocol and uses a request-response model, however it runs on top of UDP instead of TCP. CoAP uses a client-server architecture where clients communicate with servers using connectionless datagrams. CoAP is designed to easily interface with HTTP. Like HTTP, CoAP supports methods such as GET, PUT, POST, and DELETE. CoAP draft specifications are available on IEFT Constrained environments (CoRE) Working Group website [12].

a mobile application or an IoT device. WebSocket is described in RFC 6455 [13].

- **MQTT :** Message Queue Telemetry Transport (MQTT) is a light-weight messaging protocol based on the publish-subscribe model. MQTT uses a client-server architecture where the client (such as an IoT device) connects to the server (also called MQTT Broker) and publishes messages to topics on the server. The broker forwards the messages to the clients subscribed to topics. MQTT is well suited for constrained environments where the devices have limited processing and memory resources and the network bandwidth is low. MQTT specifications are available on IBM developerWorks [14].

CHAPTER 1

Introduction

1.1 Welcome to the World of Embedded Processors

1.1.1 Where Are the Processors Used?

If you are new to microcontrollers or ARM® processors, first I would like to give you a very warm welcome.

Processors are used in majority of electronic products. For example, your mobile phones, televisions, washing machines, cars, bank card (smartcards), and even simple devices like the remote control for your radio can have processors inside. In most cases, these processors are placed inside in chips called **microcontrollers**. In modern microcontrollers, the chip also contains the essential elements like memory systems and interface hardware (often called peripherals). There are many different types of microcontrollers; they can be available with different processors, memory sizes, and peripherals inside, and can be available in different packages (Figure 1.1).

Large numbers of microcontrollers are designed for general purpose, which means they can be used in wide range of applications. Sometimes processors are used in chips that are

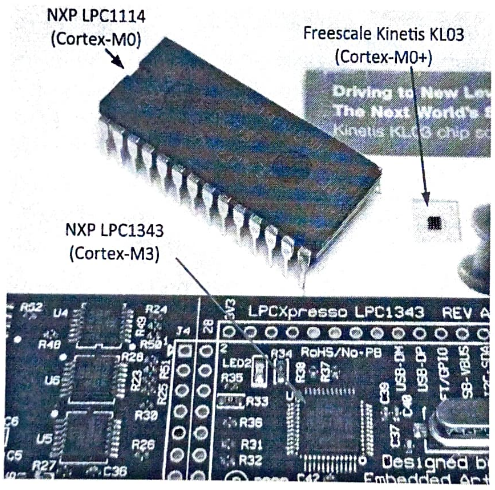


Figure 1.1
Microcontrollers are available in wide range of physical packages.

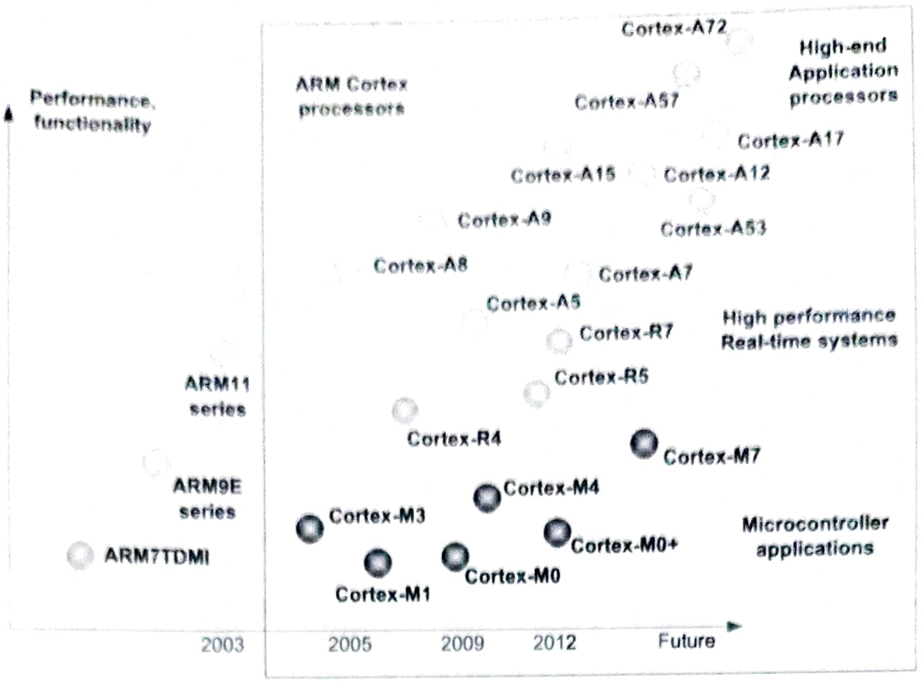


Figure 1.3
Overview of the ARM processor family.

In around 2003, ARM realized that it needs to diversify the processor products to address different technical requirements in different markets. As a result, three product profiles are defined, and the Cortex® processor brand name is created for the naming of these new processors:

Cortex-A processors—These are Application processors, which are designed to provide high performance and include features to support advanced operation systems (e.g., Android, Linux, Windows, iOS). These processors typically have longer processor pipeline and can run at relatively high clock frequency (e.g., over 1 GHz). In terms of features, these processors have Memory Management Unit (MMU) to support virtual memory addressing required by advanced OS, optional enhanced Java support, and a secure program execution environment called TrustZone®.

The Cortex-A processors are typically used in mobile phone, mobile computing devices (e.g., tablets), television, and some of the energy efficient servers.

While the Cortex-A processors have high performance, the processor is not designed to provide rapid response time to hardware events (i.e., real-time requirements). As a result, a different profile of high-performance processors is needed, and they are the Cortex-R processors.

Cortex-R processors—These are Real-Time, high performance processors that are very good at data crunching, can run at fairly high clock speed (e.g., 500 MHz to 1 GHz range), and at the same time can be very responsive to hardware events. They have cache memories as well as Tightly Coupled Memories, which enable deterministic behavior for interrupt handling. The Cortex-R processors are also designed with additional features to enable much higher system reliability such as Error Correction Code (ECC) support for memory systems and dual-core lock-step feature (i.e., redundant core logic for error detection).

The Cortex-R processors can be found in hard disk drive controllers, wireless baseband controllers/modem, specialized microcontrollers such as automotive and industrial controllers.

While the Cortex-R processors can be very good at high-performance microcontroller applications, they are quite complex designs and can consume fair amount of power. Therefore, another group of processors are need for the very low-power embedded products, and they are the Cortex-M processors.

Cortex-M Processors—The Cortex-M Processors are designed for main stream microcontroller market where the processing requirement is less critical, but need to be very low power. Most of the Cortex-M Processors are designed with a fairly short pipeline, for example, two stage in the Cortex-M0+ processor and three stages in Cortex-M0, Cortex-M3, and the Cortex-M4 Processors. The Cortex-M7 processor has a longer pipeline (six stages) due to higher performance requirement, but still the pipeline is a lot shorter than the designs of high-end application processors. As a result of the shorter pipeline and low power optimizations in the design, the maximum clock frequencies for these processors are slower than Cortex-R and Cortex-A processors, but this is rarely a problem because even a 100 MHz Cortex-M-based microcontroller can do a lot of work.

The Cortex-M processors are designed to provide very quick and deterministic interrupt responses. To achieve this, the processor's execution control part is closely coupled with a built-in interrupt controller called Nested Vectored Interrupt Controller (NVIC). The NVIC provides powerful and yet easy-to-use interrupt's management. In general, the Cortex-M processors are very easy to use, with almost everything can be programmed in C.

Due to their low power, fairly high performance, and ease of use benefits, the Cortex-M processors are selected by most major microcontroller vendors in their flagship microcontroller products. The Cortex-M processors are also used in some of the sensors, wireless communication chipsets, mixed signal ASICs/ASSPs, and even used as controller in some of the subsystems in complex application processors/SoC products.

In addition to the Cortex processor families, ARM also has processors specially designed for security-sensitive products, which included temper-resistance features. These processors are the SecurCore® series. For example, the SC000™, one of the SecurCore is designed based on the Cortex-M0 processor (same instruction set, and uses NVIC for interrupt management). The SecurCore products can be found in SIM cards, banking/payment systems, and even some electronic ID cards.

1.2.3 Blurring the Boundaries

In some ways, the term microcontroller can be a bit vague. Some of the microcontrollers are based on application processors such as ARM926EJ-S, one of the processor in the ARM9E processor family. In last few years, some of the microcontroller vendors starting to produce microcontroller products based on the ARM Cortex-A processors (e.g., Freescale Vybrid, Atmel SAMA5D3), and ARM Cortex-R processors (e.g., Texas Instruments TMS570, Spansion Traveo Family).

At the same time, the Cortex-M processors are also being used in many complex SoC devices as power management controller, I/O subsystem controller, etc.

In the next generation of Cortex-R processor based on the ARMv8-R architecture, the architecture definition also allows the processor to incorporate a MMU so that it can be used with a full feature OS like Linux or Android, and at the same time handle real-time tasks based on a virtualization mechanism.

1.2.4 ARM Cortex-M Processor Series

There are a number of processors in the Cortex-M processor family, as shown in Table 1.1.

If we look at the instruction set in a bit more details (Figure 1.4), we can see that the Cortex-M0, Cortex-M0+, and Cortex-M1 processors only support a small instruction set (56 instructions). Most of these instructions are 16 bit, thus provide a very good code density—which means it need a smaller program memory require for the same task compared to many architecture.

The instruction set of the Cortex-M0 and Cortex-M0+ processors are fairly simple. But if an application task involves complex data processing, then potentially a long sequence of instructions is needed to accomplish the operations in the Cortex-M0/M0+ processor because of the simple instruction set. In those cases, it might be better to use the Cortex-M3 processor because the Cortex-M3 processor supports a number of extra instructions (mostly 32 bit) that supports the following:

- More memory addressing modes
- Larger immediate data in the 32-bit instructions

Table 1.1: The Cortex®-M Processor family

Processor	Descriptions
Cortex-M0	The smallest ARM® processor—only approximately 12000[a] logic gates at minimum configuration. It is very low power and energy efficient.
Cortex-M0+	The most energy efficient ARM processor—it has a similar size as the Cortex-M0 processor, but with additional system level and debug features (all optional), and have higher energy efficiency than the Cortex-M0 processor design. It supports the same instruction set as the Cortex-M0 processor.
Cortex-M1	It is a small processor design optimized for field programmable Gate Array (FPGA) applications. It has the same instruction set and architecture as in the Cortex-M0 processor, but has FPGA specific memory system features.
Cortex-M3	When compared to the Cortex-M0 and Cortex-M0+ processors, the Cortex-M3 has a much more powerful instruction set, and its memory system is designed to provide higher processing throughput (e.g., use of Harvard bus architecture). It also has more system level and debug features, but at a cost of larger silicon area (minimum gate count is about 40000 gates) and slightly lower energy efficiency. In general, the energy efficiency of the Cortex-M3 processor is still a lot better than many traditional 8-bit and 16-bit microcontroller devices because the performance is substantially higher. The Cortex-M3 processor is very popular in the 32-bit microcontroller market.
Cortex-M4	The Cortex-M4 processor contains all the features of the Cortex-M3 processor, but with additional instructions to support DSP applications and have an option to include a floating point unit (FPU). It has the same system level and debug features as the Cortex-M3 processor.
Cortex-M7	It is a high performance processor designed to cover application spaces where the existing Cortex-M3 and Cortex-M4 processors cannot reach. Its instruction set is a superset of the Cortex-M4 processor, for example, supporting both single and double precision floating point calculations. It also has many advanced features, which are usually find in high-end processors such as caches and branch predictions.

[a]The exact gate count of a processor depends on many factors such as the semiconductor process library used, the chip design tool used, the design optimization options, signal routing constraints, etc.

- Longer branch and conditional branch ranges
- Additional branch instructions
- Hardware divide instructions
- Multiply accumulate (MAC) instructions
- Bit field processing instructions
- Saturation adjustment instructions

As a result, the Cortex-M3 processor can handle complicate data processing quicker. The code size might be similar to Cortex-M0 or Cortex-M0+ processor because although fewer number of instructions are required to perform the same operations, and these powerful instructions are mostly 32 bit instead of 16 bit. These 32-bit instructions also enable the Cortex-M3 processor to utilize the registers in the register bank better.

In some applications, however, you might need to perform some DSP operations such as filtering, signal transformations (e.g., Fast Fourier Transform), etc. In these applications,

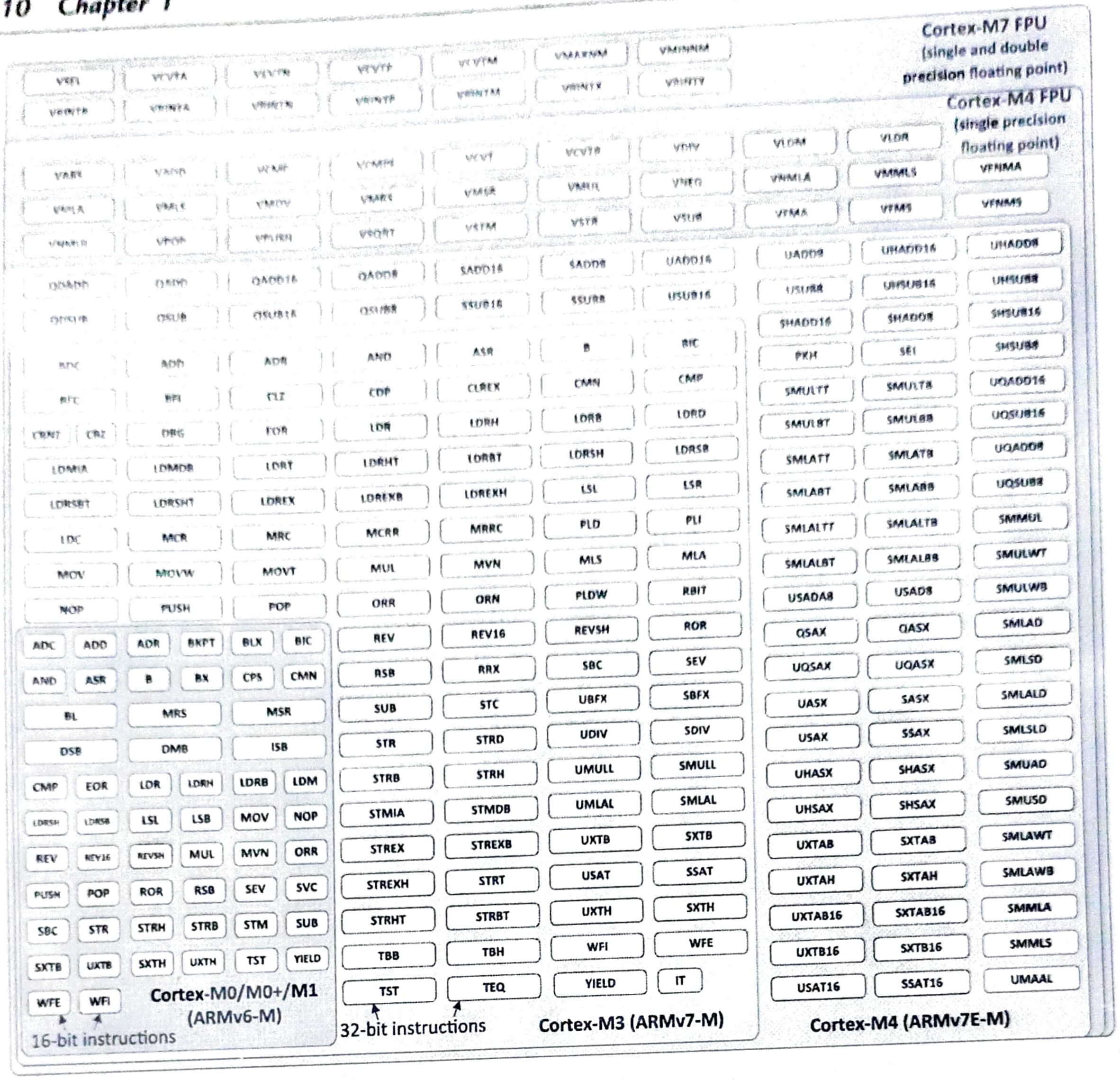

Cortex-M7 FPU (single and double precision floating point)
VSEL VCVTA VCVTN VCVTP VCVTM VMAXNM VMINNM
VRINTR VRINTA VRINTN VRINTP VRINTM VRINTX VRINTZ
Cortex-M4 FPU (single precision floating point)
VABS VADD VCMP VCMPE VCVT VCVTR VDIV VLDM VLDR VFNMA
VMLA VMLS VMOV VMRS VMSR VMUL VNEG VNMLA VNMLS VFNMS
VNMUL VPOP VPUSH VSQRT VSTM VSTR VSUB VFMA VFMS
QDADD QADD QADD16 QADD8 SADD16 SADD8 UADD16 UADD8 UHADD16 UHADD8
QDSUB QSUB QSUB16 QSUB8 SSUB16 SSUB8 USUB16 USUB8 UHSUB16 UHSUB8
SHADD16 SHADD8 SHSUB16
ADC ADD ADR AND ASR B BIC PKH SEL SHSUB8
BFC BFI CLZ CDP CLREX CMN CMP SMULTT SMULTB UQADD16
CBNZ CBZ DBG EOR LDC LDR LDRH LDRB LDRD SMULBT SMULBB UQSUB16
LDMIA LDMDB LDRT LDRHT LDRBT LDRSH LDRSB SMLATT SMLATB UQADD8
LDRSBT LDRSHT LDREX LDREXB LDREXH LSL LSR SMLABT SMLABB UQSUB8
LDC MCR MRC MCRR MRRC PLD PLI SMLALTT SMLALTB SMMUL
MOV MOVW MOVT MUL MVN MLS MLA SMLALBT SMLALBB SMULWT
NOP PUSH POP ORR ORN PLDW RBIT USADA8 USAD8 SMULWB
ADC ADD ADR BKPT BLX BIC REV REV16 REVSH ROR QSAX QASX SMLAD
AND ASR B BX CPS CMN RSB RRX SBC SEV UQSAX UQASX SMLSD
BL MRS MSR SUB STC UBFX SBFX UASX SASX SMLALD
DSB DMB ISB STR STRD UDIV SDIV USAX SSAX SMLSLD
CMP EOR LDR LDRH LDRB LDM STRB STRH UMULL SMULL UHASX SHASX SMUAD
LDRSH LDRSB LSL LSB MOV NOP STMIA STMDB UMLAL SMLAL UHSAX SHSAX SMUSD
REV REV16 REVSH MUL MVN ORR STREX STREXB UXTB SXTB UXTAB SXTAB SMLAWT
PUSH POP ROR RSB SEV SVC STREXH STRT USAT SSAT UXTAH SXTAH SMLAWB
SBC STR STRH STRB STM SUB STRHT STRBT UXTH SXTH UXTAB16 SXTAB16 SMMLA
SXTB UXTB SXTH UXTH TST YIELD TBB TBH WFI WFE UXTB16 SXTB16 SMMLS
WFE WFI TST TEQ YIELD IT USAT16 SSAT16 UMAAL
Cortex-M0/M0+/M1 (ARMv6-M)
16-bit instructions
32-bit instructions
Cortex-M3 (ARMv7-M)
Cortex-M4 (ARMv7E-M)

Figure 1.4
Instruction set of the Cortex®-M processor family.

you might want to use the Cortex-M4 processor because the Cortex-M4 processor added another group of instructions targeted for these applications—these included Single Instruction Multiple Data (SIMD) operations and saturated arithmetic instructions. The internal data path of the processor is also redesigned to enable single cycle MAC operations.

The Cortex-M4 processor also has an optional floating point unit that support IEEE-754 single precision floating point calculations. It does not mean that you cannot perform floating point processing in the Cortex-M0, Cortex-M0+, or other processors without the floating point unit. If you are using these processors for floating point operations, the compiler will insert runtime library functions to handle the floating point calculation using software, which can take much longer to do and need additional code size overhead.

For applications that demand very high data-processing requirements, or if double precision floating point calculation is needed, then the Cortex-M7 processor might be the best choice. It is designed to provide very high data-processing performance, but use the same programmer's model and a superset of the instruction set as Cortex-M4 processor.

To decide which processor to use in a project, you need to understand the processing requirements of the application. Some general guideline is shown in Table 1.2.

Please note that you might also need to consider the differences of the system-level features and performance when selecting the right Cortex-M processor. An overview of the comparison is shown in Table 1.3 and a comparison of the performance is shown in Table 1.4. Please note that the Cortex-M processors are very configurable and the exact features can be customized by the chip designers and vary among different devices.

In general, the ARM Cortex-M0 and Cortex-M0+ processors are both very suitable for ultra-low power applications, and because the instruction set and programmer's model are relatively simple, and the architecture is very C-friendly, they are also very suitable for beginners. For example, there is no need to learn a lot of tool chain-specific keywords or data types to get the application to work on a Cortex-M microcontroller, unlike many 8-bit or 16-bit architectures.

Table 1.2: The applications for various Cortex®-M Processors

Processor	Applications
Cortex-M0, Cortex-M0+ processors	General data processing and I/O control tasks. Ultra low power applications. Upgrade/replacement for 8-bit/16-bit microcontrollers. Low-cost ASICs, ASSPs
Cortex-M1	Field Programmable Gate Array(FPGA) applications with small to medium data processing complexity. (For high-complexity data processing there are FPGAs with built-in Cortex-A processors such as Xilinx Zynq-7000 and some of the Altera Arria V SoCs and Cyclone V SoCs).
Cortex-M3	Feature-rich/high-performance/low-power microcontrollers. Light-weight DSP applications.
Cortex-M4	Feature-rich/high-performance/low-power microcontrollers. DSP applications. Applications with frequent single precision floating point operations.
Cortex-M7	Feature-rich/very high performance power microcontrollers. DSP applications. Applications with frequent single or double precision floating point operations.

Table 1.3: An overview of the system level and debug features for various Cortex®-M Processors

Features	Cortex-M0	Cortex-M0+	Cortex-M1	Cortex-M3	Cortex-M4	Cortex-M7
Number of interrupts	1–32	1–32	1, 8, 16, 32	1–240	1–240	1–240
Interrupt priority levels	4	4	4	8–256	8–256	8–256
FPU	-	-	-	-	Optional (single precision)	Optional (single precision/single + double precision)
OS support	Y	Y	Optional	Y	Y	Y
Memory Protection unit	-	Optional	-	Optional	Optional	Optional
Cache	-	-	-	-	-	Optional
Debug	Optional	Optional	Optional	Optional	Optional	Yes
Instruction trace	-	Optional MTB	-	Optional ETM	Optional ETM	Optional ETM
Other trace	-	-	-	Optional	Optional	Optional

Table 1.4: Performance of various Cortex®-M Processors with commonly used benchmarks

Features	Cortex-M0	Cortex-M0+	Cortex-M3	Cortex-M4	Cortex-M7
Dhrystone 2.1 (per MHz)	0.9	0.95	1.25	1.25	2.14
CoreMark 1.0 (per MHz)	2.33	2.46	3.34	3.40	5.01

1.2.5 Quick Glance on the ARM Cortex-M0 and Cortex-M0+ Processor

The Cortex-M0 and Cortex-M0+ Processors:

- Are 32-bit Reduced Instruction Set Computing (RISC) processor, based on an architecture specification called ARMv6-M Architecture. The bus interface and internal data paths are 32-bit width.
- Have 16 32-bit registers in the register bank (r0 to r15). However, some of these registers have special purposes (e.g., R15 is the Program Counter, R14 is a register called Link Register, and R13 is the Stack Pointer).
- The instruction set is a subset of the Thumb Instruction Set Architecture. Most of the instructions are 16 bit to provide very high code density.
- Support up to 4 GB of address space. The address space is architecturally divided into a number of regions.
- Based on Von Neumann bus architecture (although arguably the Cortex-M0+ processor have a hybrid bus architecture because of an optional separate bus interface for fast peripheral register accesses, see section 4.3.2 Single Cycle I/O Interface in Chapter 4).

- Designed for low-power applications, including architectural support for sleep modes and have various low power features at the design/implementation level.
- Includes an interrupt controller called NVIC. The NVIC provides very flexible and powerful interrupt management.
- The system bus interface is pipelined, based on a bus protocol called Advanced High-performance Bus (AHB™) Lite. The bus interface supports transfers of 8-bit, 16-bit, and 32-bit data, and also allows wait states to be inserted. The Cortex-M0+ processor also have an optional bus interface (Single Cycle I/O interface, see section 4.3.2) for high-speed peripheral registers, which is separated from the main system bus.
- Support various features for the OS (Operating System) implementation such as a system tick timer, shadowed stack pointer, and dedicated exceptions for OS operations.
- Includes various debug features to enable software developers to create applications efficiently.
- Designed to be very easy to use. Almost everything can be programmed in C and in most cases no need for special C language extension for data types or interrupt handling support.
- Provide good performance in most general data processing and I/O control applications.

The Cortex-M0 and Cortex-M0+ processors do not include any memory and have only got one built-in timer which is primarily for OS operations. Therefore a chip designer needs to add additional components in the chip design themselves.

1.2.6 From Cortex-M0 Processor to Cortex-M0+ Processor

The ARM Cortex-M0 processor was released in 2009. It was a ground-breaking product because it is the first product that demonstrated it is possible to cramp a 32-bit processor into the silicon footprint similar to an 8-bit or 16-bit processors, while still able to make the design usable and provide excellent energy efficiency and a decent performance for a 32-bit processor.

Although the Cortex-M0 processor is a lot smaller than the Cortex-M3 processor (which was released in 2005), it maintains a number of key advantages as in Cortex-M3 processor:

- Flexible interrupt management using a built-in interrupt controller called NVIC
- OS support features including a timer hardware called SysTick (System Tick timer) and exception types dedicated to OS operations
- High code density
- Low power support such as sleep modes
- Integrated debug support
- Easy to use (almost everything programmable in plain C language)

The Cortex-M0 processor has been a very successful product, and was the fastest licensed ARM processor in 2009.[1] After the Cortex-M0 processor is released, the designers in ARM have received additional feedback from customers, microcontroller users and chip designers, and ARM decided that there is an opportunity for an enhanced version for the Cortex-M0 processor, which was subsequently called the Cortex-M0+ processor.

The Cortex-M0+ processor supports all the features available in the Cortex-M0 processor, but additional features were added to make it more powerful (these are all configurable by the chip designers):

- Unprivileged execution level and Memory Protection Unit (MPU)—this feature is available in other ARM processors such as the Cortex-M3 processor. It allows an OS to execute some of the application tasks with an unprivileged level so that the OS can impose memory access restrictions. For example, the unprivileged software cannot access critical system registers in the processors like NVIC registers, and memory access permissions can be managed by the MPU. In this way, a system can be made more robust because a misbehaving unprivileged task cannot corrupt critical data used by the OS kernel and other tasks.
- Vector Table relocation—again, this is a feature already existing in the Cortex-M3 processor. By default, the vector table is defined as the start of the memory (address 0x00000000). The Vector Table Offset Register allows the vector table to be defined in other memory locations such as a different program memory location or in SRAM. This is very useful for microcontroller devices, which might have separated vector table for boot process and user applications.
- Single Cycle I/O interface—this is a separate bus interface specifically added to allow frequently accessed I/O registers to be read/write in a single cycle. Without this feature, a load/store operation needs to go through the pipelined system bus, which needs two clock cycles per access. This feature enables microcontrollers or embedded system to have higher I/O performance, as well as higher energy efficiency in I/O intensive operations.

Internally to the processor design, there are also some significant changes. Instead of using a three-stage pipeline as in the Cortex-M0 and Cortex-M3 processors, the Cortex-M0+ processor is designed with a two-stage pipeline. This reduces the number of flip-flops in the processor, and hence reduces the dynamic power, and provides slightly higher performance at the same time because the branch penalty is reduced by one clock cycle.

In the Cortex-M0+ processor pipeline, as shown in Figure 1.5, a small part of the instruction decoding operations is carried out as soon as the instruction enters the

[1] Cortex-M0 Processor—Fastest Licensing ARM Processor (http://www.arm.com/about/newsroom/26419.php).

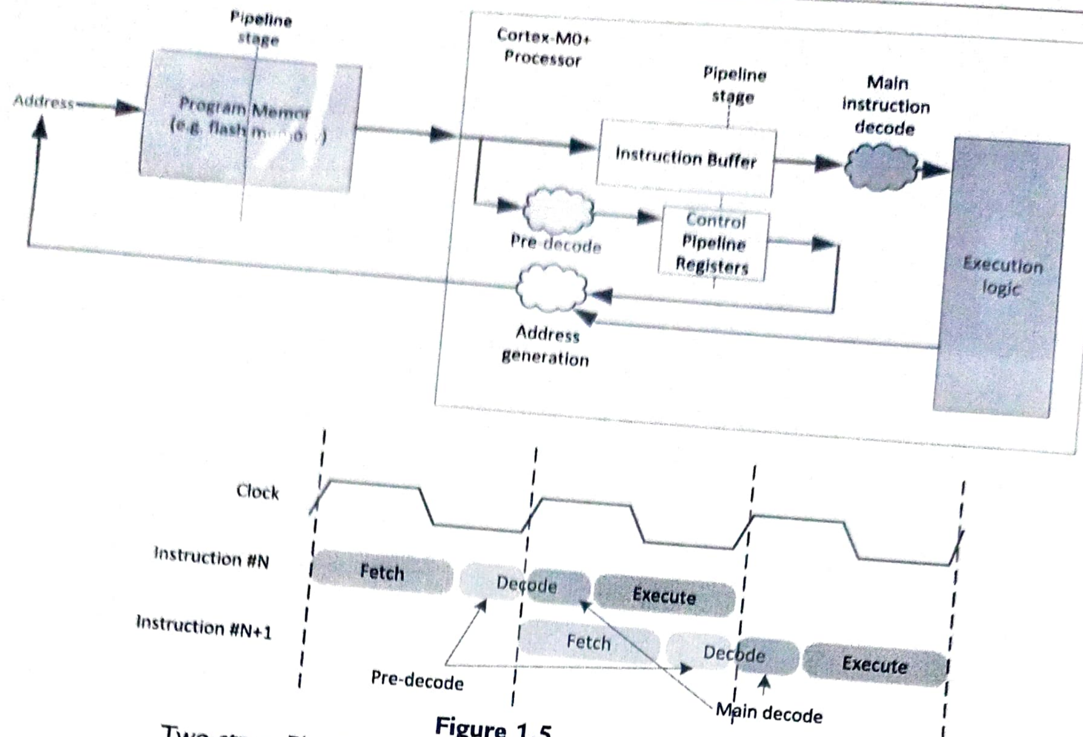


Figure 1.5
Two-stage Pipeline in the ARM® Cortex®-M0+ Processor.

processor bus interface. The rest of the instruction decoding is combined with the execution stage.

Adding decode logic to the instruction fetch stage do have some impact to the timing of the design. However, the balance between predecode and main decode logic was selected carefully to minimize the impact to the achievable maximum clock frequency. In addition, most of the low-power microcontrollers run at fairly low clock frequency in comparison to the maximum processor speed. Therefore this is not a problem to most of the silicon designs.

In some cases, the power consumption of the processor is reduced by 30% when comparing between Cortex-M0 processor and the Cortex-M0+ Processor. However, at the system level, the difference would be much smaller because most of the power could be consumed by the memory system.

In order to reduce system-level power, additional optimizations have been implemented to reduce the program memory accesses:

First, by shortening the processor to a two-stage pipeline design, the branch shadow of the processor is reduced. In a pipeline processor, when a branch instruction is executed, the

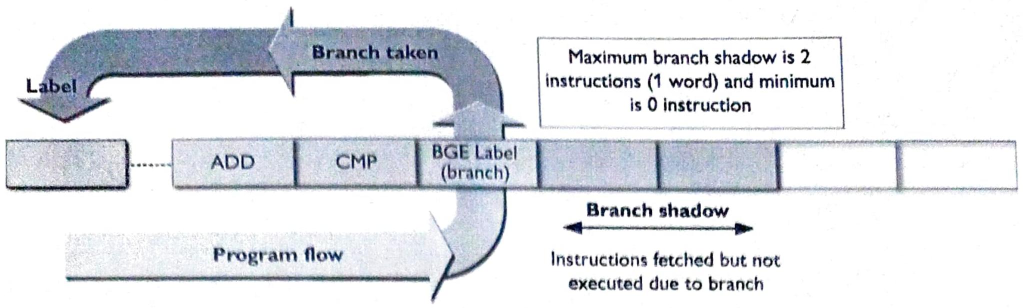

Branch taken
Label
Maximum branch shadow is 2 instructions (1 word) and minimum is 0 instruction
ADD
CMP
BGE Label (branch)
Branch shadow
Program flow
Instructions fetched but not executed due to branch

Figure 1.6
Power wastage reduction by reducing branch shadow. *Image courtesy of ARM®.*

instructions following the branch instruction would have been fetched by the processor. These instructions fetched are called branch shadow (Figure 1.6), and they are discarded by the processor and hence a long branch-shadow means wasting more energy.

Secondly, when a branch operation takes place and if the branch target instruction occupies only the second half of a 32-bit memory space (as shown in Figure 1.7), the instruction fetch is carried out as a 16-bit transfer. In this way, the program memory can switch off half of the byte lanes to reduce power.

The amount of power reduction by these techniques depends on how often branch operations are carried out in the application code.

Finally, in linear code execution, the program fetches are handled as 32-bit accesses. Since most of the instructions are 16-bit, each instruction fetch can provide up to two instructions. This means that the processor bus can be in idle state half of the time if there

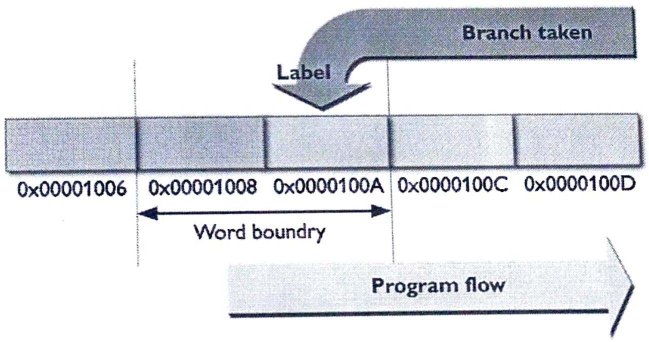


Figure 1.7
Power wastage reduction by fetching branch target with minimum transfer size. *Image courtesy of ARM®.*

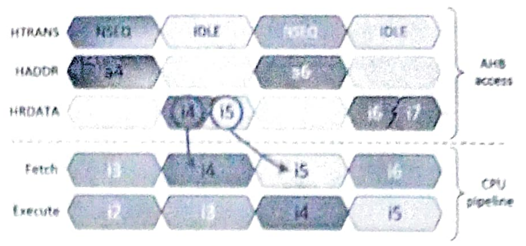


Figure 1.8
Program fetch power reduction by fetching up to two instructions at a time.
Image courtesy of ARM®.

is no data access instruction executed (Figure 1.8). Chip designers can utilize this characteristic to reduce the power consumption in the program memory (e.g., flash memory).

Another important enhancement in the Cortex-M0+ processor is the adding of a feature called Micro Trace Buffer (MTB). This unit enables low-cost instruction trace, which is very useful during software development, for example, helping to investigate the reason for a software failure. The details of the MTB are covered in Chapter 13 and appendix E.

The Cortex-M0+ processor have additional enhancements when compared to the Cortex-M0 processor in terms of chip design aspects (most of these are invisible to microcontroller users). For example, a hardware interface was added to allow the startup sequence of the processor to be delayed, which is useful for many SoC designs with multiple processors.

Today, many microcontroller vendors already started offering microcontroller products based on the Cortex-M0+ processors.

1.2.7 Applications of the Cortex-M0 and Cortex-M0+ Processor

The Cortex-M0 and Cortex-M0+ processors are used in a wide range of products.

Microcontrollers

The most common usage is microcontrollers. Many Cortex-M0 and Cortex-M0+ microcontrollers are low-cost devices and are designed for low-power applications. They can be used in applications including computer peripherals and accessories, toys, white goods, industrial and HVAC (heating, ventilating, and air conditioning) controls, home automation, etc.

When comparing the microcontrollers based on the Cortex-M0 and Cortex-M0+ processors to traditional 8-bit and 16-bit microcontroller products, the Cortex-M microcontrollers allow embedded products to be built with more features, more sophisticated user interface, due to support of larger address space, powerful interrupt control, and higher performance.

The better performance and small size also bring the benefit of higher energy efficiency. For example, for the same processing task, you can finish the processing quicker and allow the system to stay in sleep modes longer.

Another advantage of using ARM Cortex-M processors for microcontroller applications is that they are very easy to use. Therefore it is very appealing to many microcontroller vendors as product support and educating the users can be challenging for some other processor architectures.

ASICs and ASSPs

Another important group of applications for the Cortex-M0 and Cortex-M0+ processors are ASICs and ASSPs. For example, there are a number of touch screen controllers, sensors, wireless controllers, Power Management ICs (PMIC), and smart battery controllers designed based on the Cortex-M0 or Cortex-M0+ processors.

In these applications, the low gate count advantage of the Cortex-M0 and Cortex-M0+ processors allow high performance processing capability to be included in chip designs that traditionally only allow 8-bit or simple 16-bit processors to be used.

System on Chips

For complex SoC, the designs are often divided into a main application processor system and a number of subsystems for: I/O controls, communication protocol processing, and system management. In some cases, the Cortex-M0 and Cortex-M0+ processor can be used in part of the subsystems to off-load some activities from the main application processor, and to allow small amount of processing be carried out while the main processor is in standby mode (e.g., in battery powered products). It might also be used as a System Control Processor (SCP) for boot sequence management and power management.

1.3 What Is Inside a Microcontroller

1.3.1 Typical Elements Inside a Microcontroller

There can be many components inside a basic microcontroller. For example, a simplified block diagram is shown in Figure 1.9:

In the diagram there are a lot of acronyms. They are explained in Table 1.5.

As shown in Figure 1.9, there can be a lot of components in a microcontroller (not to mention other complex interfaces like Ethernet, USB, etc.). In some microcontrollers you

Technical Overview

2.1 What are the Cortex®-M0 and Cortex-M0+ Processors?

The ARM® Cortex-M0 processor and Cortex-M0+ processors are both 32-bit processors. Their internal registers in the register banks, data paths, and the bus interfaces are all 32 bit. Both of them have a single main system bus interface, therefore they are considered as Von Neumann bus architecture.

The Cortex-M0+ processor has an optional single cycle I/O interface that is primarily for faster peripheral I/O register accesses. Therefore, it is possible to say the Cortex-M0+ processor has limited Harvard bus architecture capability as instruction access and I/O register accesses could be carried out at the same time, but it is important to understand that although there can be two bus interfaces, the memory space is shared (unified) and therefore the extra bus interface does not bring additional addressable memory space.

The key characteristics of the Cortex-M0 and Cortex-M0+ processors are as follows:

Processor pipeline

- The Cortex-M0 processor has a three-stage pipeline (fetch, decode, and execute)
- The Cortex-M0+ processor has a two-stage pipeline (fetch + predecode, decode + execute)

Instruction set

- The instruction set is based on Thumb® Instruction Set Architecture (ISA). Only a subset of the Thumb ISA is used (56 of them). Most of the instructions are 16 bit in size, only a few of them are 32 bit.
- In general, the Cortex-M processors are classified as Reduced Instruction Set Computing although they have instructions of different sizes.
- Support optional single cycle 32 bit × 32 bit multiply, or a smaller multicycle multiplier for designs that need small silicon area.

Memory addressing

- 32-bit addressing supporting up to 4 GB of memory space
- The system bus interface is based on an on-chip bus protocol called AHB-Lite, supporting 8-bit, 16-bit, and 32-bit data transfers
- The AHB-Lite protocol is pipelined, support high operation frequency for the system. Peripherals can be connected to a simpler bus based on APB protocol (Advanced Peripheral Bus) via an AHB to APB bus bridge.

Interrupt Handling

- The processors include a built-in interrupt controller called the Nested Vectored Interrupt Controller (NVIC). This unit handles interrupt prioritization and masking functions. It supports up to 32 interrupt requests from various peripherals (chip design dependent), an additional Non-Maskable Interrupt (NMI) input, and also support a number of system exceptions.
- Each of the interrupts can be set to one of the four programmable priority levels. NMI has a fixed priority level.

Operating Systems (OS) support

- Two system exception types (SVCall and PendSV) are included to support OS operations.
- An optional 24-bit hardware timer called SysTick (System Tick Timer) is also included for periodic OS time keeping.
- The Cortex-M0+ processor support privileged and unprivileged execution level (optional to chip designers). This allows OS to run some of the application tasks with unprivileged execution level and impose memory access restrictions to these tasks.
- The Cortex-M0+ processor has an optional Memory Protection Unit (MPU) to allow OS to define memory access permission for application tasks during run time.

Low Power support

- Architecturally two sleep modes are defined as normal sleep and deep sleep. The exact behaviors in these sleep modes are device specific (depends on which chip you are using). Chip designers can also add device specific power saving mode control registers to expand the number of sleep modes or to allow the sleep mode behavior for each part of the chip to be defined.
- Sleep mode can be entered using WFI (Wait for Interrupt) or WFE (Wait for Event) instructions, or using a feature called Sleep-on-Exit to allow the processor to enter sleep automatically.
- Additional hardware level supports to enable chip designers to create better power reductions based on the sleep mode features, for example, the Wake-up Interrupt Controller (WIC).

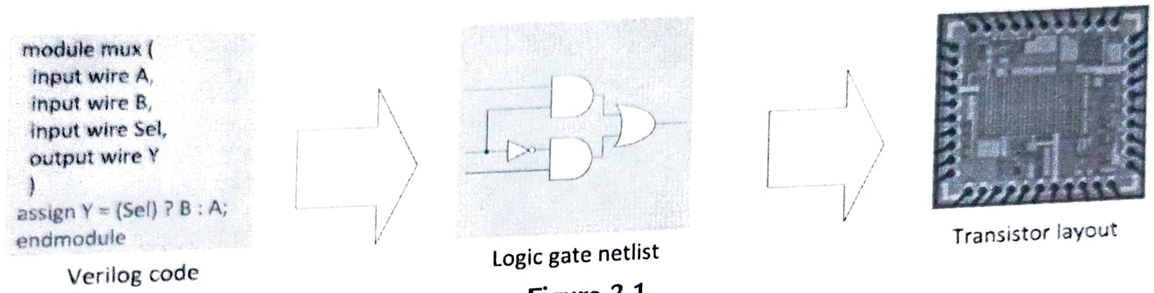
Debug

- The debug system is based on the ARM CoreSight™ Debug Architecture. It is a scalable debug architecture that can support simple-single processor designs to complex multi-processor designs.
- A debug interface that can either be based on JTAG protocol (4 or five pins), or Serial Wire Debug protocol (2 pins). The debug interface allows software developers to access debug features of the processors.
- Support up to four hardware breakpoints, two data watchpoints, and unlimited software breakpoint using BKPT (breakpoint) instruction.
- Support basic program execution profiling using a feature called Program Counter (PC) Sampling via the debug connection.

- The Cortex-M0+ Processor has an optional feature called Micro Trace Buffer (MTB), this provide instruction trace.

The Cortex-M Processors are configurable designs. They are delivered to chip designers in form of Verilog source code files with a number of parameters that chip designers can select. In this way, chip designers can omit some of the features that are unnecessary for their projects to save power and reduce silicon area. As a result, you can find microcontrollers based on the Cortex-M0 and Cortex-M0+ processor with different number of supported interrupts, and Cortex-M0+ processor with and without the optional MPU.

During the design process (Figure 2.1), the processor is integrated with the rest of the system and converted to a design composed of logic gates and then transistors layout using chip design tools. The timing characteristics like maximum clock frequency are defined at these stages based on the semiconductor process selected for the project and various design constraints. In addition, the exact maximum speed and power consumption of the Cortex-M0 or Cortex-M0+ processor on different products can also be different from each other.



```
module mux (
  input wire A,
  input wire B,
  input wire Sel,
  output wire Y
 )
assign Y = (Sel) ? B : A;
endmodule
```

Figure 2.1
Simplified chip design flow.

2.2 Block Diagrams

A simplified block diagram of the Cortex®-M0 processor is shown in Figure 2.2.

The processor core contains the register banks, ALU, data path, and control logic. It is a three-stage pipeline design with fetch stage, decode stage, and execution stage. The register bank has sixteen 32-bit registers. A few of the registers in the register bank have special usages (e.g., PC). The rest are available for general data processing.

The NVIC accepts up to 32 interrupt request signals and a NMI input. It contains the functionality required for comparing priority between interrupt requests and current priority level so that nested interrupts can be handled automatically. If an interrupt is accepted, the NVIC communicates with the processor so that the processor can execute the correct interrupt handler.

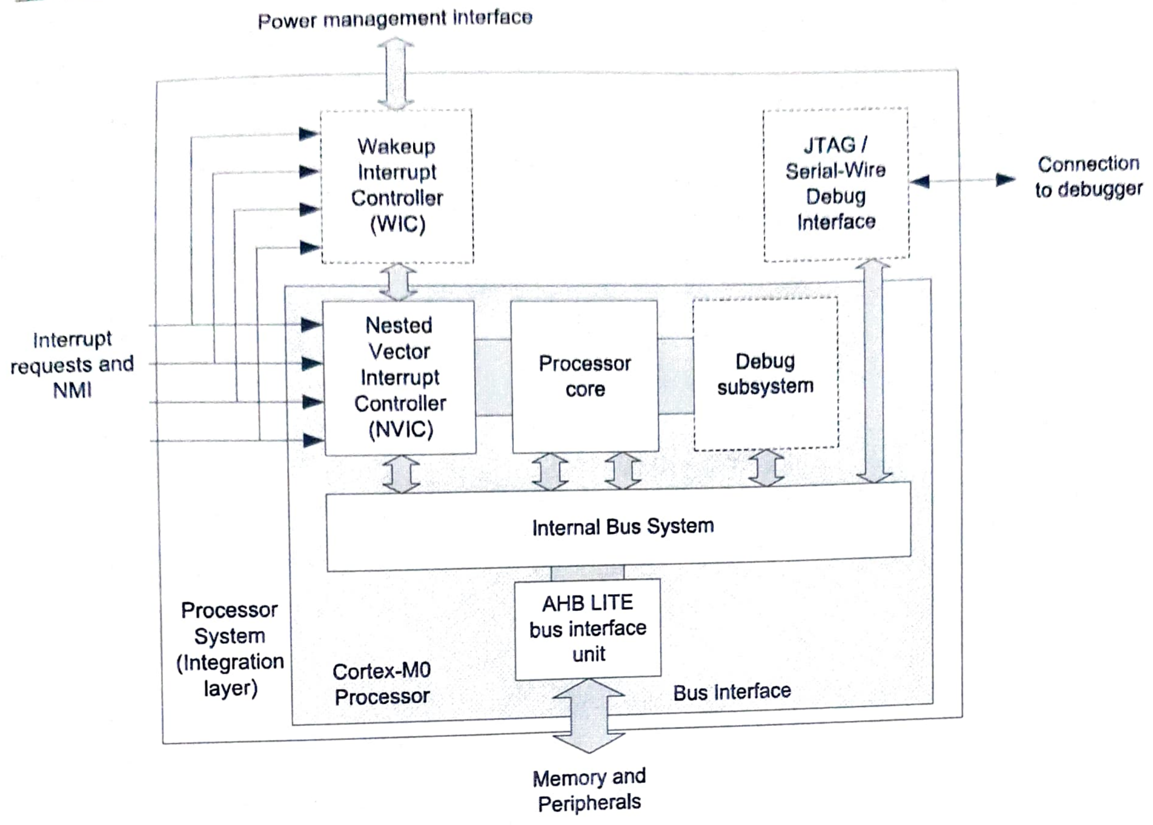

Power management interface
Wakeup Interrupt Controller (WIC)
JTAG / Serial-Wire Debug Interface
Connection to debugger
Interrupt requests and NMI
Nested Vector Interrupt Controller (NVIC)
Processor core
Debug subsystem
Internal Bus System
AHB LITE bus interface unit
Processor System (Integration layer)
Cortex-M0 Processor
Bus Interface
Memory and Peripherals

Figure 2.2
A simplified block diagram of the Cortex®-M0 Processor.

The WIC is an optional unit. In low-power applications, the microcontroller can enter standby state with most parts of the processor powered down. Under this situation, the WIC can perform the function of interrupt masking while the NVIC and the processor core are inactive. When an interrupt request is detected, the WIC informs the power management to power up the system so that the NVIC and the processor core can then handle the rest of the interrupt processing.

The debug subsystem contains various functional blocks to handle debug control, program breakpoints, and data watchpoints. When a debug event occurs, it can put the processor core in a halted state so that embedded developers can examine the status of the processor at that point.

The internal bus system, data path in the processor core, and the AHB-Lite bus interface are all 32-bit wide. AHB-Lite is an on-chip bus protocol used in many ARM® processors. This bus protocol is part of the AMBA® (Advanced Microcontroller Bus Architecture) specification, which is a bus architecture developed by ARM and widely used in the IC design industry.

The JTAG or Serial Wire interface units provide access to the bus system and debugging functionalities. The JTAG protocol is a popular 4-pin (5-pin if including a reset signal) communication protocol commonly used for IC and PCB testing. The Serial Wire protocol is a newer communication protocol that only requires two wires, but it can handle the same debug functionalities as JTAG. As illustrated in the block diagrams (Figures 2.2 and 2.3), the debug interface module is separated from the processor design. This is required in the CoreSight™ Debug Architecture where multiple processors can share the same debug connections. There are a number of additional signals for multiprocessor debug support not shown in the diagrams.

The Cortex-M0+ processor is very similar (as shown in Figure 2.3) to Cortex-M0 processor. The only addition is the adding of the optional MPU, single cycle I/O interface bus and the interface for the MTB. The processor core internal design is also changed to a two-stage pipeline arrangement.

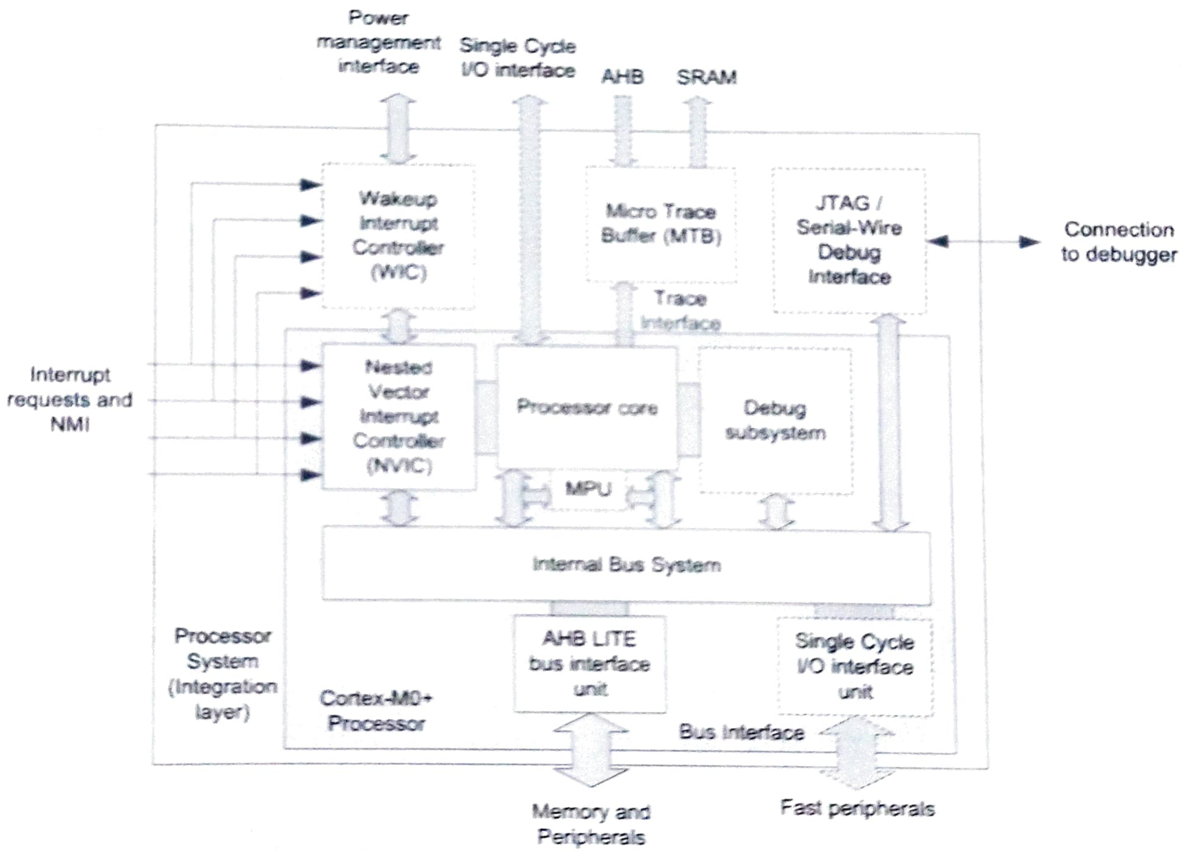

Power management interface
Single Cycle I/O interface
AHB
SRAM
Wakeup Interrupt Controller (WIC)
Micro Trace Buffer (MTB)
JTAG / Serial-Wire Debug Interface
Connection to debugger
Trace Interface
Interrupt requests and NMI
Nested Vector Interrupt Controller (NVIC)
Processor core
Debug subsystem
MPU
Internal Bus System
Processor System (Integration layer)
Cortex-M0+ Processor
AHB LITE bus interface unit
Single Cycle I/O interface unit
Bus Interface
Memory and Peripherals
Fast peripherals

Figure 2.3
A simplified block diagram of the Cortex®-M0+ processor.

The MPU is a programmable device used to define access permission of the memory map. In some of the applications where an OS is used, application tasks can be executed with an unprivileged execution level with restrict memory access defined by the MPU, which is programmed by the OS.

The single cycle I/O interface provides another bus interface with faster access compared to the AHB-Lite system bus (pipelined operation). The MTB is used to provide instruction trace.

In both Cortex-M0 and Cortex-M0+ processors, a number of components in the processors are optional. For example, the debug support, MPU and the WIC are all optional. Some other components like the NVIC are configurable: allowing chip designers to define the features available, for example, the number of interrupt requests (IRQ).

2.3 Typical Systems

As you can see from the block diagrams, the Cortex®-M0 and Cortex-M0+ processors do not contain memories and peripherals. Chip designers need to add these components to the designs. As a result, different Cortex-M processor-based microcontrollers can have different memory sizes, address map, peripherals, interrupt assignment, etc.

In a simple microcontroller design based on a Cortex-M processor, the design would consist of the following:

- A memory for program code storage, usually a Read-Only-Memory (ROM) component, or reprogrammable memory technologies such as flash memory.
- A read—write memory for data (including variables, stack, etc.), usually based on Static Random Access Memory (SRAM).
- Various types of peripherals.
- Bus infrastructure components for joining the processor to all the memories and peripherals.

In some cases, there can also be a separate ROM device with boot code to boot up the microcontroller before the program in the user flash is executed. This is typically called boot ROM or boot loader.

For a simple design with Cortex-M0 processor, the design could look like the one shown in Figure 2.4.

A typical design based on the Cortex-M0 processor might partition the bus system into two parts, which are as follows:

- System bus connected to the memories including ROM, flash memory (for user program storage), the SRAM, a few number of peripherals, and a bus bridge to the peripheral bus system.

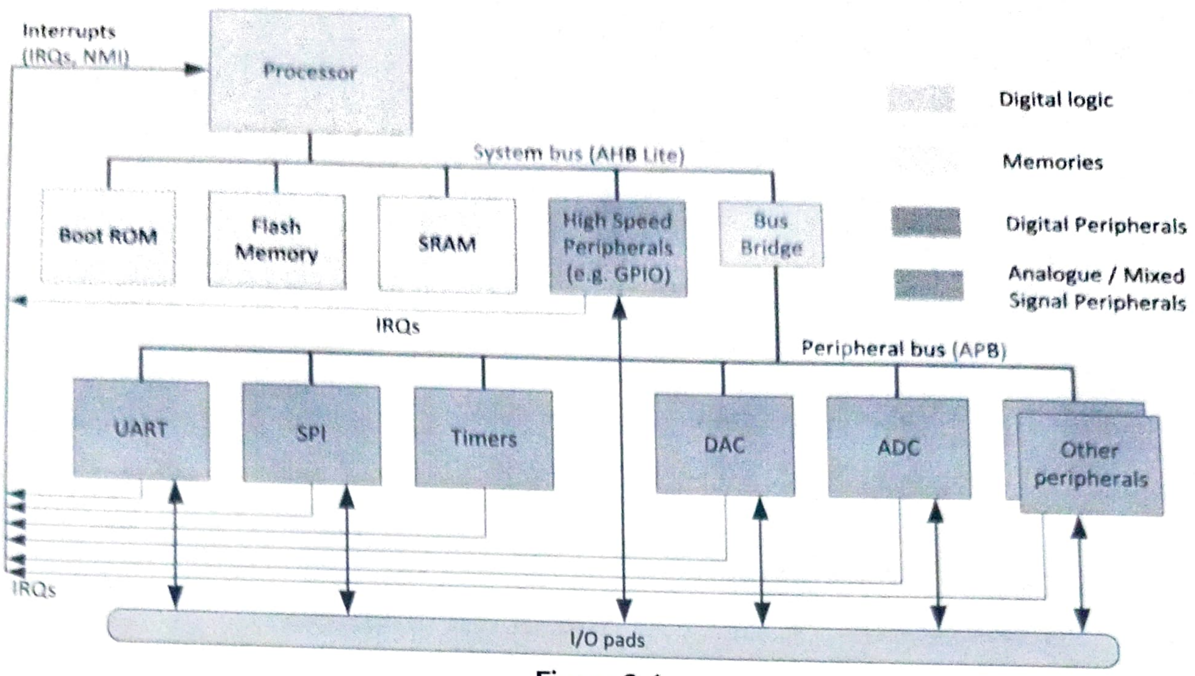


Figure 2.4
A simple system with the Cortex®-M0 Processor.

- The peripherals are connected to the peripheral bus, which might have a different operating frequency compared to the system bus.

It is quite common for some of the peripherals to be connected to a separated peripheral bus, which is linked to the main system bus via a bus bridge. This bus protocol for the peripheral bus is typically based on APB, which is a bus protocol defined in the AMBA®.

The uses of a separated APB peripheral bus are as follows:

- Allows lower hardware cost because the APB protocol (non-pipelined operations) is simpler than AHB-Lite (pipelined operations)
- Allows the peripheral bus to run at a different clock frequency than the main system bus
- Avoids large combinational logic in the bus infrastructure for the main system bus, which could become the bottle neck in terms of getting to get high operating frequency. Many peripherals might present in a microcontroller designs and the bus fabric for peripherals can become quite large.

Another group of important connections are the interrupts—A number of peripherals can generate interrupt requests, including the General Purpose Input/Output (GPIO) modules. In most microcontroller designs, external devices connected to certain GPIO pins can generate interrupt request to the processor via some additional conditioning and synchronization logic.

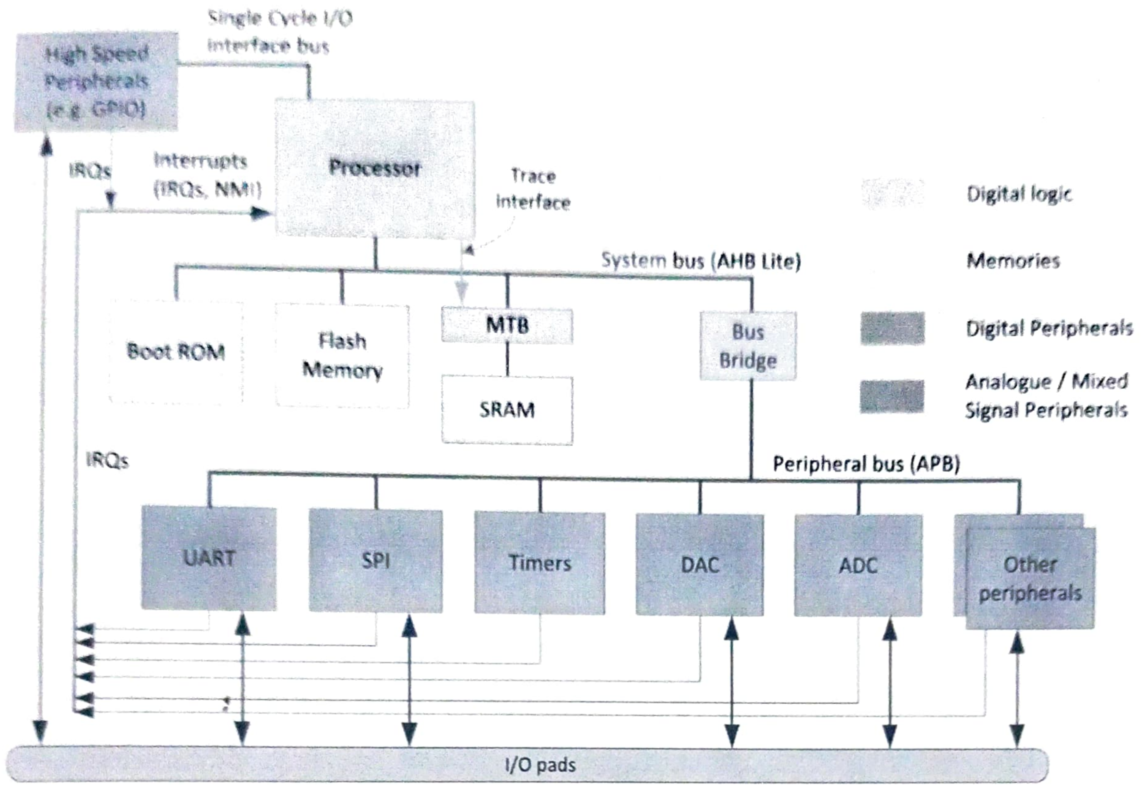


Figure 2.5
A simple system with the Cortex®-M0+ Processor.

For a system based on the Cortex-M0+ processor, the system design can be very similar, like the one shown in Figure 2.5.

In this design, the high-speed peripherals are moved to the single cycle I/O interface bus for faster I/O performance, and the MTB is added between the AHB-Lite system bus and the SRAM for support instruction trace capture.

Potentially the processor might not be the only component in the system that can generate bus transactions. In many microcontroller products, there is also a component called Direct Memory Access (DMA) controller. Once programmed, the DMA controller can carry out memory accesses on requests from peripherals without processor intervention (Figure 2.6)

The DMA controller can perform data transfers between memory and peripherals, or between memories (e.g., to accelerate memory copy). This is commonly needed for microcontrollers with high bandwidth communication interface like Ethernet or USB. However, it can also benefit some low-power applications, for example, by avoiding waking up the processor from sleep mode to collect small amount of data from peripherals.

mode, with privileged access level by default.

4.2.2 Registers and Special Registers

In order to perform data processing and controls, a number of registers are required inside the processor core. If data from memory is to be processed, it has to be loaded from the memory to a register in the register bank, processed inside the processor, and then written back to the memory if needed, or kept in the register bank for another operation. This is commonly called "load-store architecture." By having a sufficient number of registers in the register bank, this mechanism is easy to use, and is C-friendly. It is easy for C compilers to compile a C program into machine code with good performance.

The Cortex-M0 and Cortex-M0+ processor provides a register bank of 16 32-bit registers (most are general purposed, R13–R15 has special purposes), and a number of special registers (Figure 4.3).

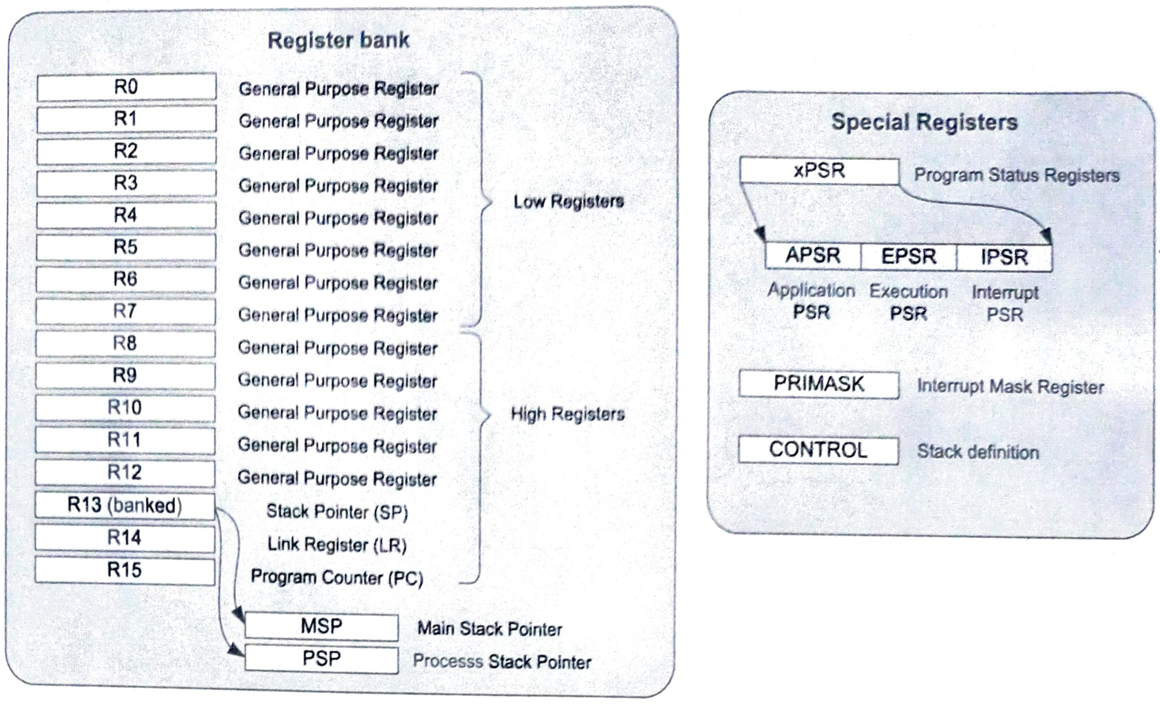


Figure 4.3
Registers in the Cortex®-M0 and Cortex-M0+ processors.

The detailed descriptions for these registers are as follows:

R0–R12

Registers R0–R12 are for general uses. Due to the limited space in the 16-bit Thumb® instructions, many of the Thumb instructions can only access R0–R7, which are also called the low registers. While some instructions, like MOV (move), can be used on all registers. When using these registers with ARM® development tools such as the ARM assembler, you can use either upper case (e.g., R0) or lower case (e.g., r0) to specify the register to be used. The initial values of R0–R12 at reset are undefined.

R13, Stack Pointer

R13 is the Stack Pointer. It is used for accessing the stack memory via PUSH and POP operations. There are physically two different stack pointers in Cortex-M0 and Cortex-M0+ Processors.

- The Main Stack Pointer (MSP, or SP_main in ARM documentation) is the default Stack Pointer after reset, and is used when running exception handlers.
- The Process Stack Pointer (PSP, or SP_process in ARM documentation) can only be used in Thread mode (when not handling exceptions).

The stack pointer selection is determined by the CONTROL register, one of the special registers which will be introduced later (*CONTROL—Special Register*).

When using ARM development tools, you can access the stack pointer using either "R13" or "SP." Both upper case and lower case (e.g., "r13" or "sp") can be used. Only one of the stack pointers is visible at a given time. However, you can access to the MSP or PSP directly when using the special register access instructions MRS and MSR. In such cases, the register names "MSP" or "PSP" should be used.

The lowest 2 bits of the stack pointers are always zero and writes to these 2 bits are ignored. In ARM processors, PUSH and POP are always 32-bit accesses because the registers are 32-bit, and the transfers in stack operations must be aligned to a 32-bit word boundary. The initial value of MSP is loaded from the first 32-bit word of the vector table from the program memory during the start-up sequence. The initial value of PSP is undefined.

It is not necessary to use the PSP. In many applications, the system can completely rely on the MSP. The PSP is normally used in designs with an OS, where the stack memory for OS Kernel and the thread-level application codes must be separated.

R14, Link Register

R14 is the Link Register (LR). The LR is used for storing the return address of a subroutine or function call. When BL or BLX is executed, the return address is stored in LR. At the end of the subroutine or function, the return address stored in LR is loaded into the program counter (PC) so that the execution of the calling program can be resumed. In the case where an exception occurs, the LR also provides a special code value which is used by the exception return mechanism. When using ARM development tools, you can access to the LR using either "R14" or "LR." Both upper and lower case (e.g., "r14" or "lr") can be used.

Although the return address in the Cortex-M0/M0+ processor is always an even address (bit[0] is zero because smallest instruction are 16-bit and must be half-word aligned), bit zero of LR is readable and writeable. In the ARMv6-M architecture, some instructions require bit zero of a function address set to 1 to indicate Thumb state.

R15, Program Counter

R15 is the PC. It is readable and writeable. A read returns the current instruction address plus four (this is caused by the pipeline nature of the design). Writing to R15 will cause a branch to take place (but unlike a function call, the LR does not get updated).

In the ARM assembler, you can access the PC using either "R15" or "PC," in either upper or lower case (e.g., "r15" or "pc"). Instruction addresses in the Cortex-M0/M0+ processor must be aligned to half-word address, which means the actual bit zero of the PC should be zero all the time. However, when attempting to carry out a branch using the branch instructions (BX or BLX), the LSB of the PC should be set to1.[1] This is to indicate that

[1] Not required when a move (MOV) or add (ADD) instruction is used to modify the PC.

the branch target is a Thumb program region. Otherwise, it can imply an attempt to switch the processor to ARM state (depending on the instruction used), which is not supported and will cause a fault exception.

xPSR, Combined Program Status Register

The combined Program Status Register (PSR) provides information about program execution and the ALU flags. It consists of the following three PSRs (Figure 4.4):

- Application PSR (APSR),
- Interrupt PSR (IPSR), and
- Execution PSR (EPSR)

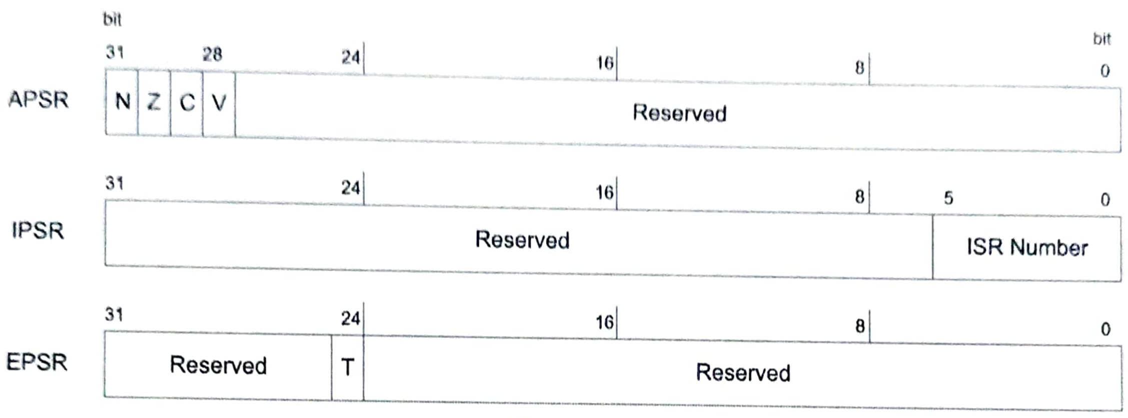


Figure 4.4
Application PSR (APSR), Interrupt PSR (IPSR), and Execution PSR (EPSR).

The APSR contains the ALU flags: N (negative flag), Z (zero flag), C (carry or borrow flag), and V (overflow flag). These bits are at the top 4 bits of the APSR. The common use of these flags is to control conditional branches.

The IPSR contains the current executing ISR (Interrupt Service Routine) number. Each exception on the Cortex-M0/M0+ processor has a unique associated ISR number (exception type). This is useful for identifying the current interrupt type during debugging and allows an exception handler that is shared by several exceptions to know which exception it is serving.

The EPSR on the Cortex-M0/M0+ processor contains the T bit which indicates that the processor is in the Thumb state. On the Cortex-M0/M0+ processor, this bit is normally set to 1 because the Cortex-M processors only support Thumb state. If this bit is cleared, a HardFault exception will be generated in the next instruction execution.

These three registers can be accessed as one register called xPSR. For example, when an interrupt takes place, the xPSR is one of the registers that is stored on to the stack memory automatically and restored automatically after returning from an exception. During the stack store and restore, the xPSR is treated as one register (Figure 4.5).

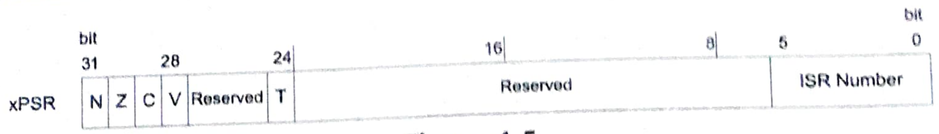


Figure 4.5
xPSR.

Direct access to the PSRs is only possible through special register access instructions. However, the value of the APSR can affect conditional branches and the carry flag in the APSR can also be used in some data processing instructions.

PRIMASK—Interrupt Mask Special Register

The PRIMASK register is a 1-bit wide interrupt mask register. When set, it blocks all interrupts apart from the Non-Maskable Interrupt (NMI) and the HardFault exception. Effectively it raises the current interrupt priority level to 0 which is the highest value for a programmable exception (Figure 4.6).

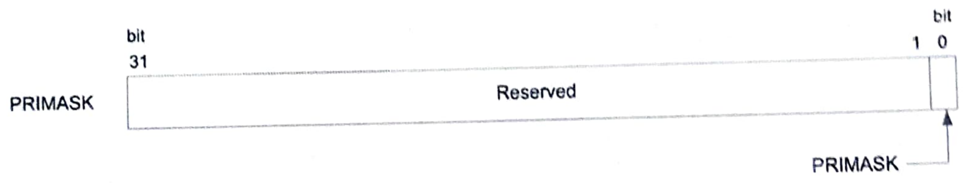


Figure 4.6
PRIMASK.

The PRIMASK register can be accessed using special register access instructions (MSR, MRS) as well as using an instruction called CPS. This is commonly used for handling time critical routines.

CONTROL—Special Register

As mentioned earlier, there are two stack pointers in the Cortex-M0 and Cortex-M0+ processors. The stack pointer selection is determined by the processor mode as well as the configuration of the CONTROL register (bit 1—SPSEL). The Thread mode of the Cortex-M0+ processor can either be privileged or unprivileged, and this is also controlled by CONTROL (bit 0—nPRIV) (Figure 4.7).

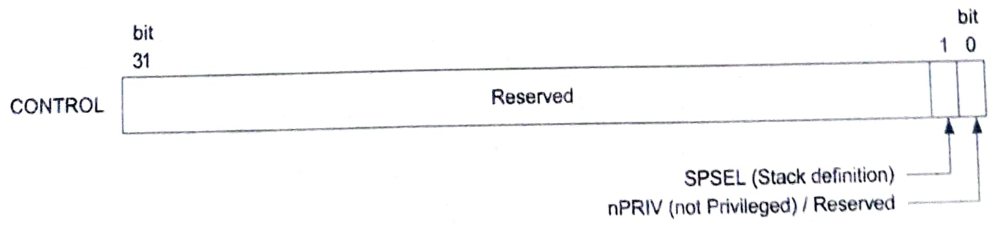


Figure 4.7
CONTROL.

After reset, the MSP is used, but can be switched to the PSP in Thread mode (when not running an exception handler) by setting bit[1] in the CONTROL register. During running of an exception handler (when the processor is in handler mode), only the MSP is used, and the CONTROL register reads as zero. The bit[1] of CONTROL register can only be changed in Thread mode, or via the exception entrance and return mechanism (Figure 4.8).

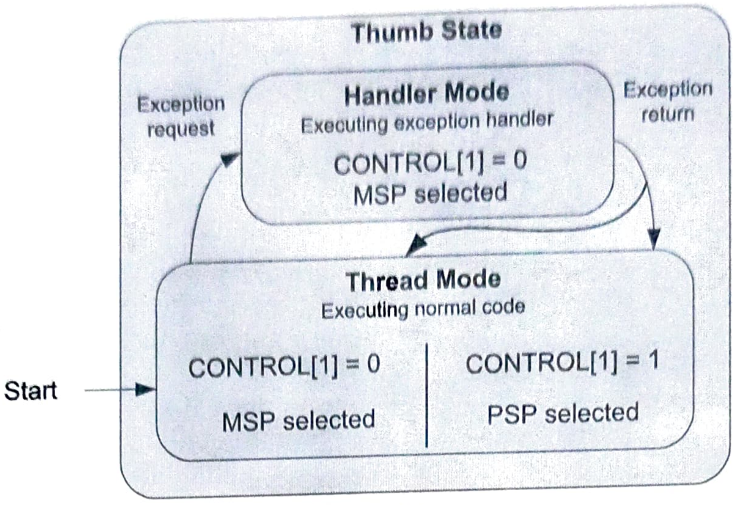


Figure 4.8
Stack pointer selection.

Bit[0] of the CONTROL register is for selecting between Privileged and Unprivileged states during Thread mode. Some of the Cortex-M0+ devices and all Cortex-M0 processor-based devices do not support unprivileged state and therefore this bit is always zero (Figure 4.9).

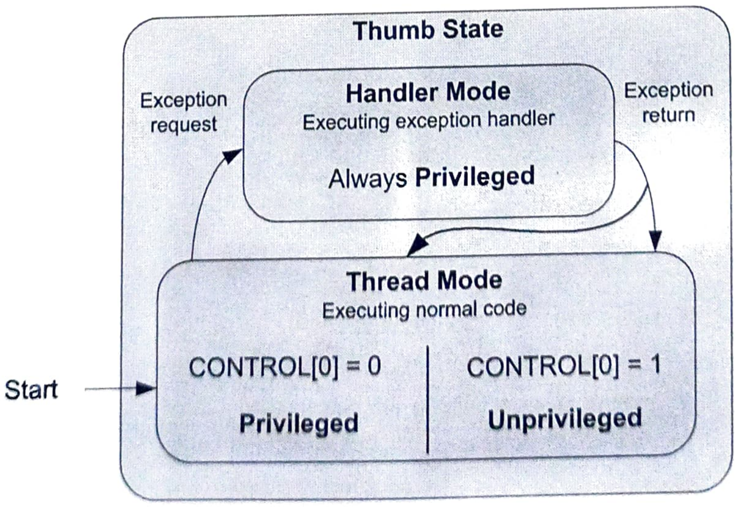


Figure 4.9
Privileged state selection.

Access of Registers and Special Registers

In C/C++ programming or any other high level languages, the registers in the register bank (R0–R12) can be utilized by the compiler automatically. In most cases, you do not need to worry about which registers being used, unless you are interfacing assembly code and C/C++ code (such mixed language development will be cover in Chapter 21).

The other special registers need to be accessed using some special instructions (MRS and MSR). The CMSIS-CORE provides a number of APIs for such usages. However, please note that some of these special registers cannot be accessed or changed by software (Table 4.1).

Table 4.1: Access limitations to special registers

	Privileged	Unprivileged
APSR	R/W	R/W
EPSR	No access (T bit read as zero)	No access (T bit read as zero)
IPSR	Read only	Read only
PRIMASK	R/W	Read only
CONTROL	R/W	Read only

4.2.3 *Behaviors of the APSR*

Data processing instructions can affect destination registers as well as the APSR which is commonly known as ALU status flags in other processor architectures. The APSR is essential for controlling conditional branches. In addition, one of the APSR flags, the C (Carry) bit, can also be used in add and subtract operations.

There are four APSR flags in the Cortex-M0 ad Cortex-M0+ processors (Table 4.2).

Table 4.2: ALU flags on the Cortex®-M0 and Cortex-M0+ processors

Flag	Descriptions
N (bit 31)	Set to bit[31] of the result of the executed instruction. When it is "1," the result has a negative value (when interpreted as a signed integer). When it is "0," the result has a positive value or equal zero.
Z (bit 30)	Set to "1" if the result of the executed instruction is zero. It can also be set to "1" after a compare instruction is executed if the two values are the same.
C (bit 29)	Carry flag of the result. For unsigned addition, this bit is set to "1" if an unsigned overflow occurred. For unsigned subtract operations, this bit is the inverse of the borrow output status.
V (bit 28)	Overflow of the result. For signed addition or subtraction, this bit is set to "1" if a signed overflow occurred.

A few examples of the ALU flag results are as given in Table 4.3.

Table 4.3: ALU flags operation examples

Operation	Results, flags
0x70000000 + 0x70000000	Result = 0xE0000000, N = 1, Z = 0, C = 0, V = 1
0x90000000 + 0x90000000	Result = 0x20000000, N = 0, Z = 0, C = 1, V = 1
0x80000000 + 0x80000000	Result = 0x00000000, N = 0, Z = 1, C = 1, V = 1
0x00001234 − 0x00001000	Result = 0x00000234, N = 0, Z = 0, C = 1, V = 0
0x00000004 − 0x00000005	Result = 0xFFFFFFFF, N = 1, Z = 0, C = 0, V = 0
0xFFFFFFFF − 0xFFFFFFFC	Result = 0x00000003, N = 0, Z = 0, C = 1, V = 0
0x80000005 − 0x80000004	Result = 0x00000001, N = 0, Z = 0, C = 1, V = 0
0x70000000 − 0xF0000000	Result = 0x80000000, N = 1, Z = 0, C = 0, V = 1
0xA0000000 − 0xA0000000	Result = 0x00000000, N = 0, Z = 1, C = 1, V = 0

In the Cortex-M0 and Cortex-M0+ processors, almost all of the data processing instructions modify the APSR; however, some of these instructions do not update the V flag or the C flag. For example, the MULS (multiply) instruction only changes the N flag and the Z flag.

The ALU flags can be used for handling data that is larger than 32-bits. For example, we can perform a 64-bit addition by splitting the operation into two 32-bit additions. The pseudo form of the operation can be written as follows:

```
// Calculating Z = X + Y, where X, Y and Z are all 64-bit
Z[31:0] = X[31:0] + Y[31:0];      // Calculate lower word addition,
                                   // carry flag get updated
Z[63:32] = X[63:32] + Y[63:32] + Carry;  // Calculate upper word addition.
```

An example of carry out such 64-bit add operation in assembly code can be found in Chapter 6 (Section 6.5.1).

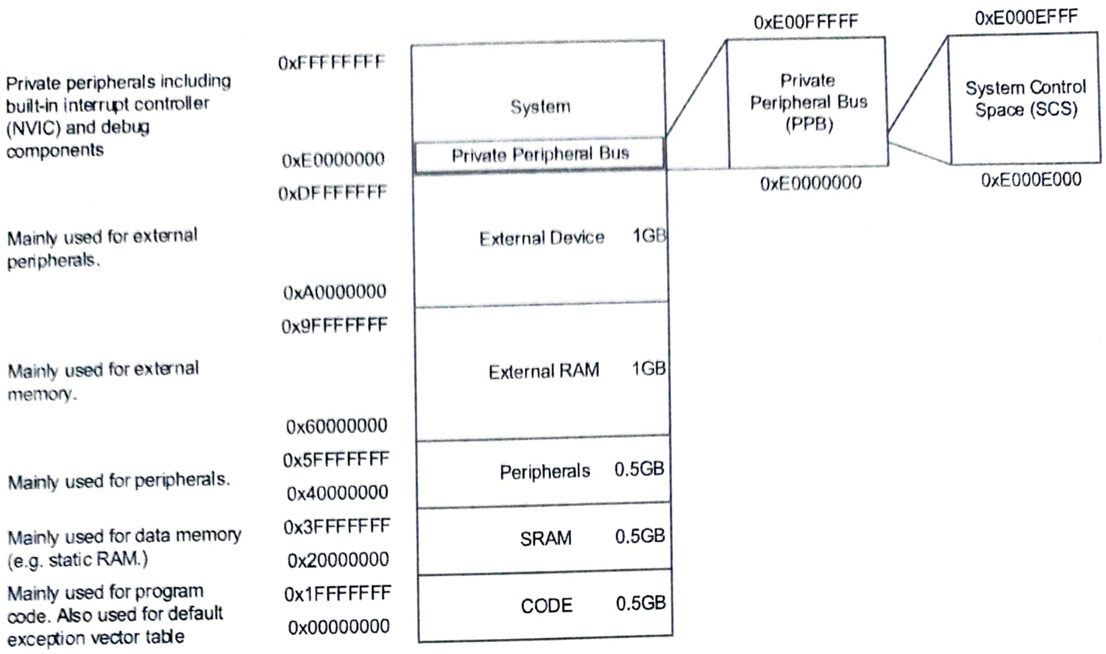
The other common usage of APSR flag is to control branching. More on this will be covered in Chapter 5 (Section 5.4.8), where the details of the condition branch instruction will be covered.

4.3 Memory System

4.3.1 Overview

All ARM® Cortex®-M processors have a 4 GB of memory address space. The memory space is architecturally defined into a number of regions, with each region having a recommended usage to help software porting between different devices (Figure 4.10).

The Cortex-M0 and Cortex-M0+ processors contain a number of built-in components like the NVIC (the interrupt controller) and a number of debug components. These are located in fixed memory locations within the system region of the memory map. As a result, all the devices based on the Cortex-M processors have the same programming model for interrupt control and debug. This makes it convenient for software porting as well as helping debug



Private peripherals including built-in interrupt controller (NVIC) and debug components
Mainly used for external peripherals.
Mainly used for external memory.
Mainly used for peripherals.
Mainly used for data memory (e.g. static RAM.)
Mainly used for program code. Also used for default exception vector table
0xFFFFFFFF
0xE0000000
0xDFFFFFFF
0xA0000000
0x9FFFFFFF
0x60000000
0x5FFFFFFF
0x40000000
0x3FFFFFFF
0x20000000
0x1FFFFFFF
0x00000000
System
Private Peripheral Bus
External Device 1GB
External RAM 1GB
Peripherals 0.5GB
SRAM 0.5GB
CODE 0.5GB
0xE00FFFFF
Private Peripheral Bus (PPB)
0xE0000000
0xE000EFFF
System Control Space (SCS)
0xE000E000

Figure 4.10
Memory map.

tool vendors to develop debug solutions for the Cortex-M0-based microcontroller or System-on-Chip (SoC) products.

The memory space is shared between instruction memory, data memory, peripherals processor's built-in peripherals (e.g., the interrupt controller), and processor's debug components. However, the debug components are not visible to the software running on the processor (from architecture point of view this is implementation defined, and existing Cortex-M0 and Cortex-M0+ processors are designed to make the debug components to be visible only from debugger). This is different from Cortex-M3, Cortex-M4, and Cortex-M7 processors, where privileged codes can access the debug components.

In most cases, the memories connected to the Cortex-M processors are 32-bits, but it is also possible to connect memory of different data widths to a Cortex-M processor with suitable memory interface hardware. The memory system in Cortex-M processors supports memory transfers of different sizes such as byte (8-bit), half word (16-bit), and word (32-bit). The Cortex-M0 and Cortex-M0+ processor designs can be configured to support either little endian or big endian memory systems, but cannot switch from one to another in an implemented design.

Since the memory system and peripherals connected to the Cortex-M0 or Cortex-M0+ processors are developed by microcontroller vendors or SoC designers, different memory sizes and memory types can be found in different Cortex-M0/M0+-based products.

4.3.2 Single Cycle I/O Interface

The Cortex-M0+ Processor has an optional feature, which allows chip designer to add a separated bus interface (in addition to the main system bus), which allows certain peripheral registers to be accessed in a single clock cycle. This enables the microcontroller product to provide better performance in I/O operations, as well as improve energy efficiency in I/O intensive applications.

When this feature is implemented, the address space connect to the single cycle I/O interface appears as a part of the main memory space, so from software point of view the peripheral registers in the single cycle I/O bus works in the same way as registers on the AHB-Lite system bus. However, this interface can only be used for data accesses and does not support instruction accesses (Figure 4.11).

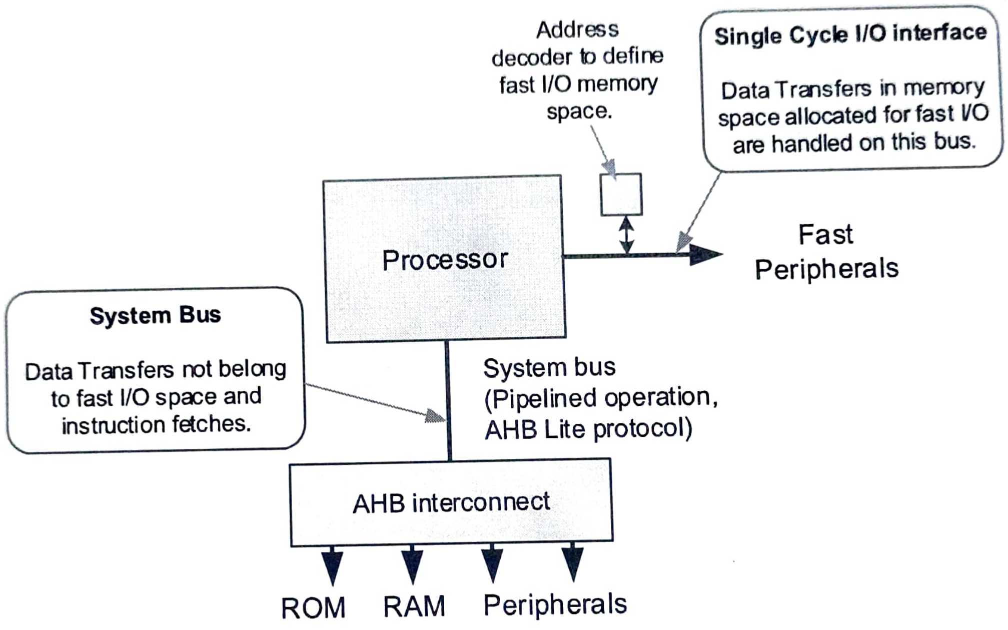

Address decoder to define fast I/O memory space.
Single Cycle I/O interface
Data Transfers in memory space allocated for fast I/O are handled on this bus.
Processor
Fast Peripherals
System Bus
Data Transfers not belong to fast I/O space and instruction fetches.
System bus (Pipelined operation, AHB Lite protocol)
AHB interconnect
ROM
RAM
Peripherals

Figure 4.11
Optional single Cycle I/O Interface on the Cortex®-M0+ Processor.

The single cycle I/O interface is intended for connecting small number of peripherals, which need faster access speed (e.g., GPIO). Peripherals like UART and timers are normally connected via the AHB-Lite system bus because the associated operations typically do not have short-latency requirement and do not occur frequently.

4.3.3 Memory Protection Unit

Another optional feature in the Cortex-M0+ processor is the MPU (MPU). This is a programmable unit and is to be used with the privileged—unprivileged states of the processor. The MPU provides up to eight programmable regions, and each region can be defined with different starting addresses, sizes, and memory access permissions.

In a multitasking system, an OS can run some of the application tasks in unprivileged state and the OS can program the optional MPU each time it switches between tasks, so each of the unprivileged application tasks run in their own permitted memory space and can only access to memory locations allocated to them.

The configuration registers of the MPU is privileged access only so that an unprivileged task cannot change the access permission to bypass the MPU.

More information about the MPU is covered in Chapter 12.

4.4 Stack Memory Operations

Stack memory is a memory usage mechanism that allows the system memory to be used as temporary data storage that behaves as a first-in-last-out buffer. One of the essential elements of stack memory operation is a register called the Stack Pointer. The stack pointer indicates where the current stack memory location is, and is adjusted automatically each time a stack operation is carried out.

In the Cortex®-M processors, the Stack Pointer is register R13 in the register bank. Physically there are two stack pointers in the Cortex-M processors, but only one of them is used at a time, depending on the current value of the CONTROL register and the state of the processor (see Figure 4.8).

In common terms, storing data to the stack is called pushing (using the PUSH instruction) and restoring data from the stack is called popping (using the POP instruction). Depending on processor architecture, some processors perform storing of new data to stack memory using incremental address indexing and some use decrement address indexing. In the Cortex-M processors, the stack operation is based on a "full-descending" stack model. This means the stack pointer always points to the last filled data in the stack memory, and the stack pointer predecrements for each new data store (PUSH) (Figure 4.12).

PUSH and POP are commonly used at the beginning and at the end of a function or subroutine. At the beginning of a function, the current contents of the registers used by the calling program are stored onto the stack memory using PUSH operations, and at the end of the function, the data on the stack memory is restored to the registers using POP operations. Typically, each register PUSH operation should have a corresponding register POP operation; otherwise the stack pointer will not be able to restore registers to their original values. This can result in unpredictable behaviors, for example, function return to incorrect addresses.

The minimum data size to be transferred for each push and pop operations is one word (32-bit) and multiple registers can be pushed or popped in one instruction. The stack

Bluetooth Classic: Version 1.0 – 3.0

We have 3 factors that enable someone to distinguish the different Bluetooth versions. They are power consumption, range, and data speed. Data packets used and modulation schemes are the primary determinates of these factors. The first Bluetooth version's release paved the way for the emergence of wireless items such as speakers, headphones, Bluetooth beacons, and game controllers used today.

Bluetooth Low Energy: Versions 4.0 – 5.0:

Bluetooth 4.0 was announced to the marketplace forming a new grouping named Bluetooth Low Energy (BLE). It was geared towards installing applications that require low power consumption and a GFSK modulation scheme that would enable it to return insufficient data output of 1Mbps. Even though its maximum data output is 1Mbps, BLE is still unsuitable for products that need continuous data streaming.

Specifications and Features from Bluetooth 1.0 to Bluetooth 5.0

a) Bluetooth 1.0

It was invented in 1998 was a significant groundbreaking discovery. As the technology was somehow immature, challenges such as no anonymity were encountered, but the technology is now outmoded with today's standards.

Some of the minor challenges were fixed by Bluetooth version 1.1, but the most significant problems were fixed after Bluetooth version 1.2. Significant improvements included sustenance for adaptive frequency-hopping spread spectrum (AFH) that minimized interference, quicker speed transmissions of close to 721kbit/s, Host Controller Interface (HCI), improved discovery, and Extended Synchronous Connections (ESCO).

b) Bluetooth 2.0

This version 2.0 was released in 2004. GFSK and phase-shift keying modulation (PSK) are some of the main improved features in this version. The role of GFSK is to improve the speed of data transfer by supporting the Enhanced Data Rate (EDR).

The technology improved further after the launch of Bluetooth version 2.1 by supporting a new feature dabbed "simple, secure pairing" (SSP). It enhanced the pairing experience, security, and extended inquiry response (EIR), thus allowing improved devices' filtering before establishing a connection.

c) Bluetooth 3.0

This Bluetooth version was announced into the market in 2009. Over a collocated 802.11 link, the Bluetooth 3.0 through the High Speed (HS) mode enables the transfer of data with speeds of up to 24 Mbps. The Bluetooth version 3.0 comes with other new specifications such as Ultra-wideband, Enhanced Power Control, L2CAP Enhanced modes, Unicast Connectionless Data, and Alternate MAC/PHY. Its high rate of power consumption has significant drawbacks.

d) Bluetooth 4.0

Bluetooth version 4.0 was released in 2010. Back in those days, the version was marketed as Bluetooth Smart and Wibree, although it still supported all the previous versions' features. BLE devices are powered by coin-cell batteries making power consumption its significant change.

e) Bluetooth version 4.1

Bluetooth version 4.1 was released in 2013, hence improving the users' experience further. This version enabled easy transfers of bulk data. It also allowed multiple simultaneous roles and co-existed with LTE.

Other new features supported by this version include:

- 11n PAL
- Minor duty cycle directed publicizing
- Partial time of discovery
- L2CAP Connection
- Dual-mode and topology
- LE link-layer topology
- Comprehensive interlaced scanning
- A fast interval of data advertising
- Mobile wireless coexistence signaling services
- Wideband speech from audio architecture updates

f) Bluetooth version 4.2

After the release of Bluetooth version 4.2 in 2014, it made it possible for the release of the Internet of Things (IoT). MOKOBlue and other manufacturers are the first to enter the Bluetooth Internet of things industry, and also make a total contribution to the development of Bluetooth.. Its main area of improvements includes:

- Link-layer privacy that extended the policies for scanner filters
- Low energy secure connection that extended the length of Data packets
- Version 6 of the Internet Protocol Support Profile (IPSP)

g) Bluetooth 5.0

The version was presented by Bluetooth SIG in 2016, although it was Sony in their product Xperia XZ Premium who first implemented this technology. Both Bluetooth 5 vs. 4.2 primarily focused on refining connectivity and experience of the Internet of Things (IoT), thereby offering a unified flow of data. Between Bluetooth 5.0 vs 5.1, the Bluetooth 5.1 range is a bit higher. Its main areas of improvements include:

- Slot Availability Mask (SAM)
- Extensions of LE Advertising
- 2 Mbit/s PHY for LE
- LE Channel Selection Algorithm #2
- Long-range LE Long
- Non-Connectable advertising high duty cycle

h) Bluetooth version 5.1

Bluetooth 5.1 was unconfined in 2019. When Bluetooth 5.0 vs. 5.1 are compared, version 5.1 was the first to support the Mesh-based model hierarchy. Its main improvements areas are;

- The angle of Departure (AoD) and Angle of Arrival (AoA)
- GATT Caching
- Periodic Advertising Sync Transfer
- Advertising Channel Index

i) Bluetooth Version 5.2

The latest Bluetooth version 5.2 was introduced by the Bluetooth SIG during the CES 2020 which was held in January 2020. This version was introduced into the market alongside the next generation of Bluetooth LE Audio. The most significant change made between Bluetooth 5.1 vs. 5.2 was that version 5.2 has Isochronous Channels (ISOC). Isochronous Channels supports BLE devices with Bluetooth 5.2 or later where it acts as the base during the implementation of LE Audio. The other 3 features that come with Bluetooth version 5.2 are;

- Isochronous Channels (ISOC)
- Enhanced Attribute Protocol (EATT)
- LE Power Control (LEPC)

Bluetooth devices Ranges by class

Bluetooth devices have 3 classes that compromise 3
standard anticipated ranges. Class 1 devices have a range of 328 feet or 100 meters, transmitting at 100 mW. Class 2 devices have a range of 33 feet or 10 meters, transmitting at 2.5 mW, whereas the range of Class 3 devices is less than 10 meters transmitting at 1 mW.

These are the anticipated ranges, where they can radically decrease due to an obstacle between the two devices, for instance, walls that weaken signals. Therefore, the transmitter's strength, the device's proximity obstruction, and the receiver's sensitivity are the most common factors influencing the range of Bluetooth devices.

Bluetooth mesh version	**Range in ft**	**Speed in (Mbit/s)**
Class 1	100mW	100 meters
Class 2	2.5mW	10 meters
Class 2	1mW	Less than 10 meters

When New Bluetooth versions are used with compatible peripherals, they come with improvements. Before the invention of Bluetooth 4.2 back in 2014, the other major version of the standard, Bluetooth 4.0, was in 2011. On the other hand, Bluetooth 5.0 is configured with far better improvements than previous standards (Bluetooth 4.0 & 4.2). The specifications of Bluetooth 4.2 features are ratified. Hence it can be supported by everything ranging from mobile phones to beacons. The table below will highlight the typical basic features that differentiate Bluetooth 5.0 and Bluetooth 4.2 versions.

Features or Specifications	**Bluetooth 4.2**	**Bluetooth 5.0**
Speed	Bluetooth 4.2 speed is lower, only supporting about 1 Mbps	Higher speed supporting about 2 Mbps, twice the speed of the Bluetooth 4.2 version
Range	Bluetooth 4.2 range is low, only supporting 10 meters indoors and 50 meters Outdoors	Bluetooth 5.0 range is high, supporting 40 meters in indoor areas and 20 meters in outdoor locations in Line Of Sight (LOS), four times than Bluetooth 4.2 version
Power Requirement	High power requirement	Low power requirement
Message Capacity	Small message capacity of about 31 bytes. Its actual data payload gives 17 – 20 bytes	Large message capacity of about 255 bytes
Robustness to operate in a congested environment	Its robustness to operate in congested environs is less	Its robustness to work in congested areas is more
Battery Life	Short battery life	Longer battery life
Security Control	Less secure than Bluetooth 5.0	More secure than Bluetooth 4.2
Theoretical Data Throughput	It has a theoretical output of 1 Mbps	It has a theoretical output of 2 Mbps and an overhead of about 1.6 Mbps
Reliability	Less reliable	Highly reliable

Features or Specifications	Bluetooth 4.2	Bluetooth 5.0
Digital Life	Less good digital life than Bluetooth 4.0 vs. 5.0	Better digital life than Bluetooth 4.2
Support for IoT devices	Bluetooth 4.2 do not support IoT devices	It supports IoT devices
Bluetooth Beacon	Due to its lower speed and range, Beacons were less popular. Their message capacity is low, at about 31 bytes	With increased speed and range in Bluetooth 5.0 version, Beacons become more popular

Can Bluetooth 4.0 connect to multiple devices?

The Bluetooth version 4.0 specification has two modes of devices; dual-mode devices and single-mode devices. All passive Bluetooth 4.0 devices can implement both or either of the ways. The classic model (BR/EDR) and Low energy mode are the two Bluetooth version 4.0 modes.

To the question, A single-mode low-energy-only device cannot connect to classic mode devices. A dual-mode Bluetooth version 4.0 device can connect with several Bluetooth Low Energy (BLE) devices.

Difference Analysis on Bluetooth 4.0 vs. Bluetooth 4.1 vs. Bluetooth 4.2

New standards that add new features or more Hardware resources required for running more complicated protocols and algorithms are issued by the SIG each year. Hence, without the latest software, it becomes tough to eliminate old natural hardware. The main differences between the 3 versions are;

Bluetooth 4.0 vs. Bluetooth 4.1

1. Increased rate of data transfer

The Bluetooth version 4.1 has a single packet data of 20 bytes, while Bluetooth 4.1 has a maximum transfer maximum of 23 bytes. This raises the rate of data transfer by 15%. Modifying the transmission rate of 23 bytes when the chip is supporting Bluetooth version 4.0 is irrelevant as it drops the packet or complies with an error.

2. Master-slave coexistence

The Bluetooth version 4.2 has an updated link-layer topology that allows concurrent master-slave coexistence and topology with master-to-multiple slave connection.

3. Supports the 32-Bit UUID

The broadcast packet carries a 32-Bit UUID. This UUID is not about the attribute list that has the 16-bit and 128-bit. To obtain the full 128-bit UUID on Bluetooth version 4.1, you only need to broadcast the 32-Bit UUID mapping as it increases the active broadcast data length in a broadcast packet.

Bluetooth 4.1 vs. Bluetooth 4.2

1. LE connection security

The AES-CCM encryption bases the specifications of pairing encryption links of Bluetooth versions 4.0 and 4.1; because Bluetooth 4.1 stocks the identical key, some dangers, and vulnerabilities might be cracked. The Diffie-Hellman Key Exchange algorithm encrypts the pairing link of Bluetooth version 4.2. Every Bluetooth 4.2 device has two keys; a private key and a public key. The users' private key and the other party's public key encrypts the encrypted file, while the receiver decrypts both the transmitting party's private and public keys. This effectively prevents the intermediary from key event cracking.

2. Privacy protection

Bluetooth continuously broadcasts a Bluetooth device address with a unique Bluetooth Mac address. The address is essential to some applications, for instance, logistics tracking app which fixes logistics equipment as stated by Bluetooth device address.

3. Improved data transmission rate

When it comes to transmission of single packet data, Bluetooth version 4.1 supports up to 23 bytes, whereas Bluetooth version 4.2 provides up to 255 bytes, thereby improving the rate of data transmission.

Is Bluetooth 4.0 the same as BLE?

Bluetooth version 4.0 rebranding by the group controlling technology helped individuals differentiate Bluetooth Smart and Bluetooth Low Energy. The Bluetooth SIG stated that version 4.0 devices would be called Bluetooth Smart Ready and Bluetooth Smart to distinguish the products featuring this technology.

Bluetooth Smart will characterize a new class of Bluetooth 4.0 peripherals. It features sensor-type devices such as pedometers and heart-rate monitors specially made to collect unique data. Meanwhile, devices using dual-mode radios referred to as Bluetooth Smart Ready can handle both the Bluetooth 4.2 BLE technology and classic Bluetooth capabilities, for instance, connecting to a hands-free device or transferring files.

Why you should Update your Bluetooth to 5.2

Since the introduction of Bluetooth 5.0 in December 2016, the technology has become more user-friendly and advanced. The Bluetooth SIG introduced into the market a radical Bluetooth version 5.2 receiver known as Bluetooth LE Audio on 7 January 2020. The version is modified with an LE Audio that enables multiple devices to share data. However, it has a limit of two devices where files can be transferred from a phone, tablet, or computer. Also, the LE Audio gives a better audio experience to individuals with hearing problems. Some of the technical specifications of the latest Bluetooth Version 5.1 vs. 5.2 are;

1. Enhanced Attribute Protocol (EATT)

combination of enhancements to the Generic Attribute profile and an upgraded version of Attribute Protocol (ATT) lead to the birth of Enhanced Attribute Protocol (EATT). This new protocol enables end-users to reduce end-to-end latency with development in the sensitivity of applications.

2. Low Energy Power Control

Bluetooth 5.2 devices have an LE Power control that exercises an essential part in improving transmission power when two devices are connected. They can also enthusiastically demand transmission power changes to lower power usage and trade-off the signal's quality.

Some benefits of LE Power control are;

I. Less power consumption.

ii. It enhances the receiver signal dependability.

iii. Growth of existing and upcoming wireless devices

3. Low Energy Isochronous Channels

Improved quality of sound hearing aids has been made promising by the introduction of Low Energy Isochronous Channels. The Isochronous Channels have made broadcasting and connection of sound to multiple devices possible. Also, multi-language audio systems have been developed due to this technology.

(a) Low Energy Audio

LE Audio transmits sound data on low-energy spectrum devices. A new compression algorithm is used to maintain the Bluetooth's quality.

(b) LC3 – Low Complexity Communication Codec

LE Audio encompasses the new low robust and high-quality audio codec LC3. With better audio high-quality and less power consumption, inventors now have a colossal elasticity as they can design new wireless merchandise easily.

(c) Hearing Aid Improvements

Many individuals have benefited from Bluetooth technology, where wireless calling has made driving safer. Productivity has increased as people can take calls while driving to the office or home.

BLE (Bluetooth Low Energy)

Introduction:

BLE (Bluetooth Low Energy) is wireless PAN technology designed and maintained by Bluetooth Special Interest Group (SIG). There are various versions of bluetooth. The version 4.2 and above is referred as BLE. The latest in the series are v5.0 and v5.1. BLE specifications are intended to reduce power consumption and cost of devices while maintaining coverage range. BLE is known as "Bluetooth Smart" where as previous version is known as "bluetooth classic".

• BLE is not backward compatible with BR/EDR protocols.

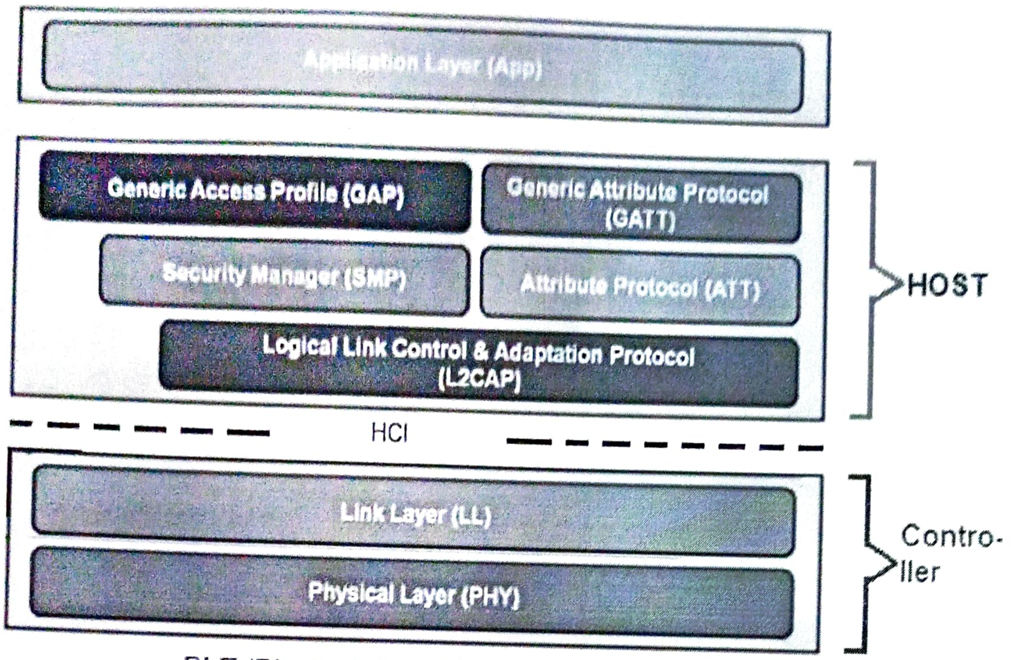
• BLE uses 2.4 GHz ISM frequency band either in dual mode or single mode. Dual mode supports both bluetooth classic and low energy peripherals.

• All BLE devices use the GATT profile (Generic Attribute Profile). The GATT protocol provides series of commands for the client to discover information about BLE server.

• The BLE protocol stack architecture consists of two parts viz. controller and host. Both are interfaced using HCI (Host to Controller Interface).

• Any profiles and applications run on top of GAP & GATT protocol layers.

BLE Protocol Stack | BLE System Architecture



BLE (Bluetooth Low Energy) Protocol Stack

The figure-2 depicts **BLE system architecture**. Let us understand functions of different layers of this BLE protocol stack.

• **Physical Layer :**

• The transmitter uses GFSK modulation and operates at unlicensed 2.4 GHz frequency band.

• Using this PHY layer, BLE offers data rates of 1 Mbps (Bluetooth v4.2)/2 Mbps (Bluetooth v5.0).

• It uses frequency hopping transceiver.

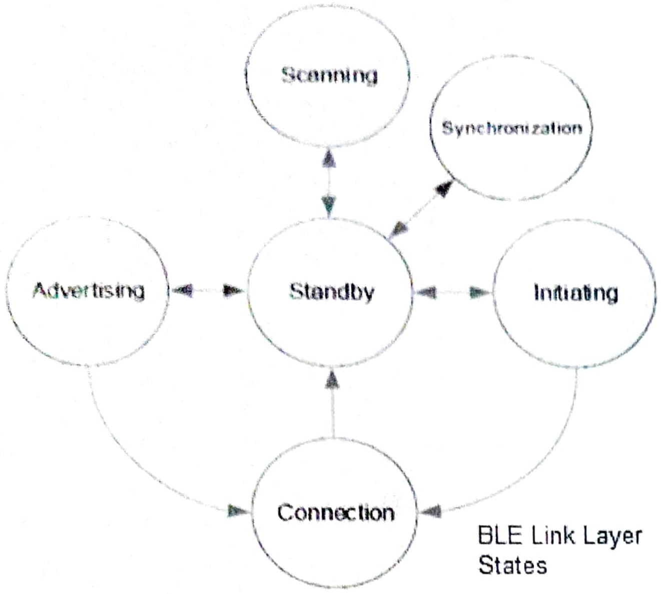
• Two modulation schemes are specified to deliver 1 Msym/s and 2 Msym/s.

• Two PHY layer variants are specified viz. uncoded and coded.

• A Time Division Duplex (TDD) topology is employed in both of the PHY modes.

• **Link Layer :** This layer sits above the Physical layer. It is responsible for advertising, scanning, and creating/maintaining connections. The role of BLE devices changes in peer to peer (i.e. Unicast) or broadcast modes. The common roles are Advertiser/Scanner (Initiator), Slave/Master or Broadcaster/Observer.

Link layer states are defined in the figure below.



BLE Link Layer States

The figure-1 depicts BLE device states >>. The device will be in any one of these states which include Standby state, Advertising state, Scanning state, Initiating state, Connection State and Synchronization state.

• **HCI :** It provides communication between controller and host through standard interface types. This HCI layer can be implemented either using API or by interfaces such as UART/SPI/USB. Standard HCI commands and events are defined in the bluetooth specifications.

• **L2CAP** :This layer offers data encapsulation services to upper layers. This allows logical end to end data communication.

• **SMP** :This security Manager layer provides methods for device pairing and key distributions. It offers services to other protocol stack layers in order to securely connect and exchange data between BLE devices.

• **GAP** : This layer directly interfaces with application layer and/or profiles on it. It handles device discovery and connection related services for BLE device. It also takes care of initiation of security features.

• **GATT** : This layer is service framework which specifies sub-procedures to use ATT. Data communications between two BLE devices are handled through these sub-procedures. The applications and/or profiles will use GATT directly.

• **ATT** : This layer allows BLE device to expose certain pieces of data or attributes.

• **Application Layer :**

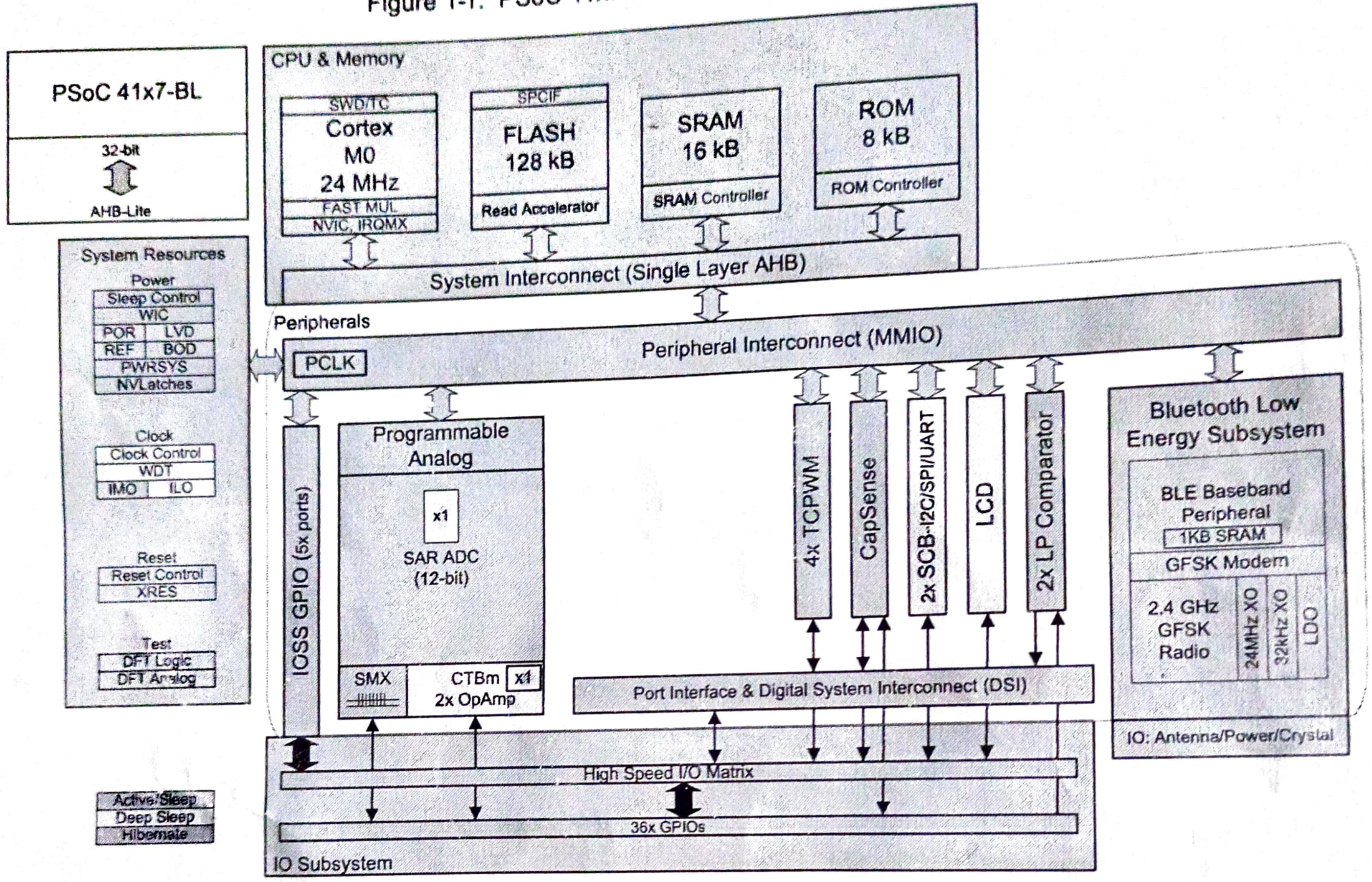
• The BLE protocol stack layers interact with applications and profiles as desired. Application interoperability in the Bluetooth system is accomplished by Bluetooth profiles.

• The profile defines the vertical interactions between the layers as well as the peer-to-peer interactions of specific layers between devices.

• A profile composed of one or more services to address particular use case. A service consists of characteristics or references to other services.

• Any profiles/applications run on top of GAP/GATT layers of BLE protocol stack. It handles device discovery and connection related services for the BLE device.

Figure 1-1. PSoC 41x7-BL4xx Family Block Diagram



1. Introduction

PSoC® 4 is a programmable embedded system controller with an ARM® Cortex®-M0 CPU. It combines programmable analog, programmable interconnect, user-programmable digital logic, and commonly used fixed-function peripherals with a high-performance ARM Cortex-M0 subsystem. The PSoC 4xxx-BL family is based on the PSoC 4 architecture which supports Bluetooth. This is upward-compatible with larger members of PSoC 4.

PSoC 4 devices have these characteristics:

- High-performance, 32-bit single-cycle Cortex-M0 CPU core
- BLE radio and subsystem
  - On-chip BLE transceiver
  - Link layer controller compliant with Bluetooth 4.2
- Fixed-function and configurable digital blocks
- Programmable digital logic
- High-performance analog system
- Flexible and programmable interconnect
- Capacitive touch sensing (CapSense®)
- Low-power operating modes – Sleep, Deep-Sleep, Hibernate, and Stop modes
- Direct memory access (DMA)

This document describes each functional block of the PSoC device in detail. This information will help designers to create system-level designs.

1.1 Top Level Architecture

Figure 1-1 shows the major components of the PSoC 41x7-BL4xx architecture and Figure 1-2 shows the major components of the PSoC 42x7-BL4xx architecture. Figure 1-3 shows the major components of the PSoC 41x8-BL4xx architecture and Figure 1-4 shows the major components of the PSoC 42x8-BL4xx architecture. Figure 1-5 shows the major components of the PSoC 41x8-BL5xx architecture and Figure 1-6 shows the same for PSoC 42x8-BL5xx architecture.

Figure 1-1. PSoC 41x7-BL4xx Family Block Diagram

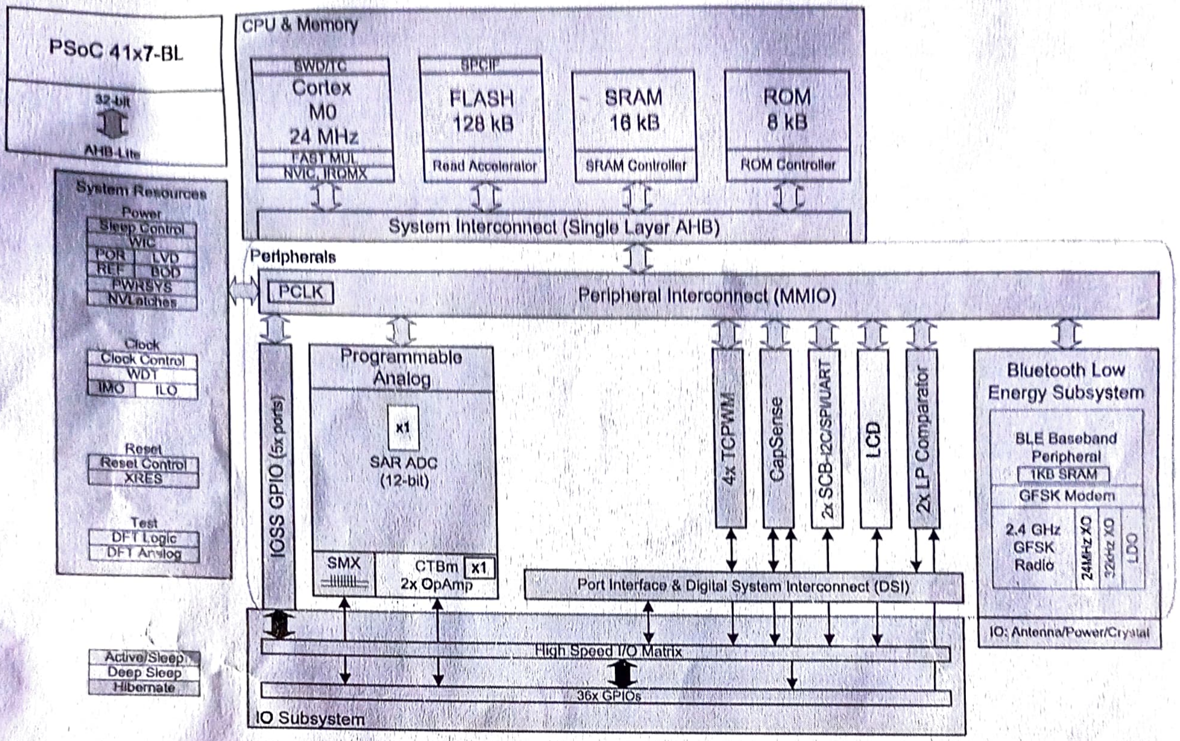


Figure 1-3. PSoC 41x8-BL4xx Family Block Diagram

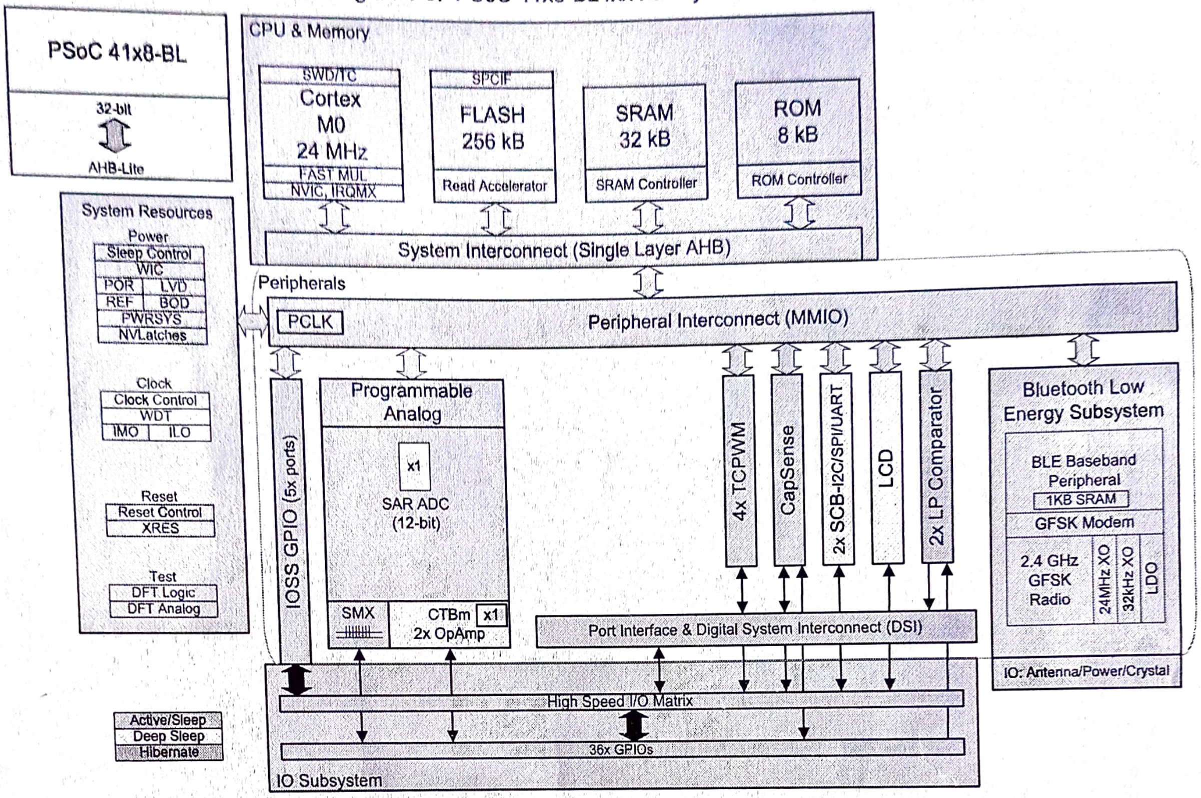


Figure 1-2. PSoC 42x7-BL4xx Family Block Diagram

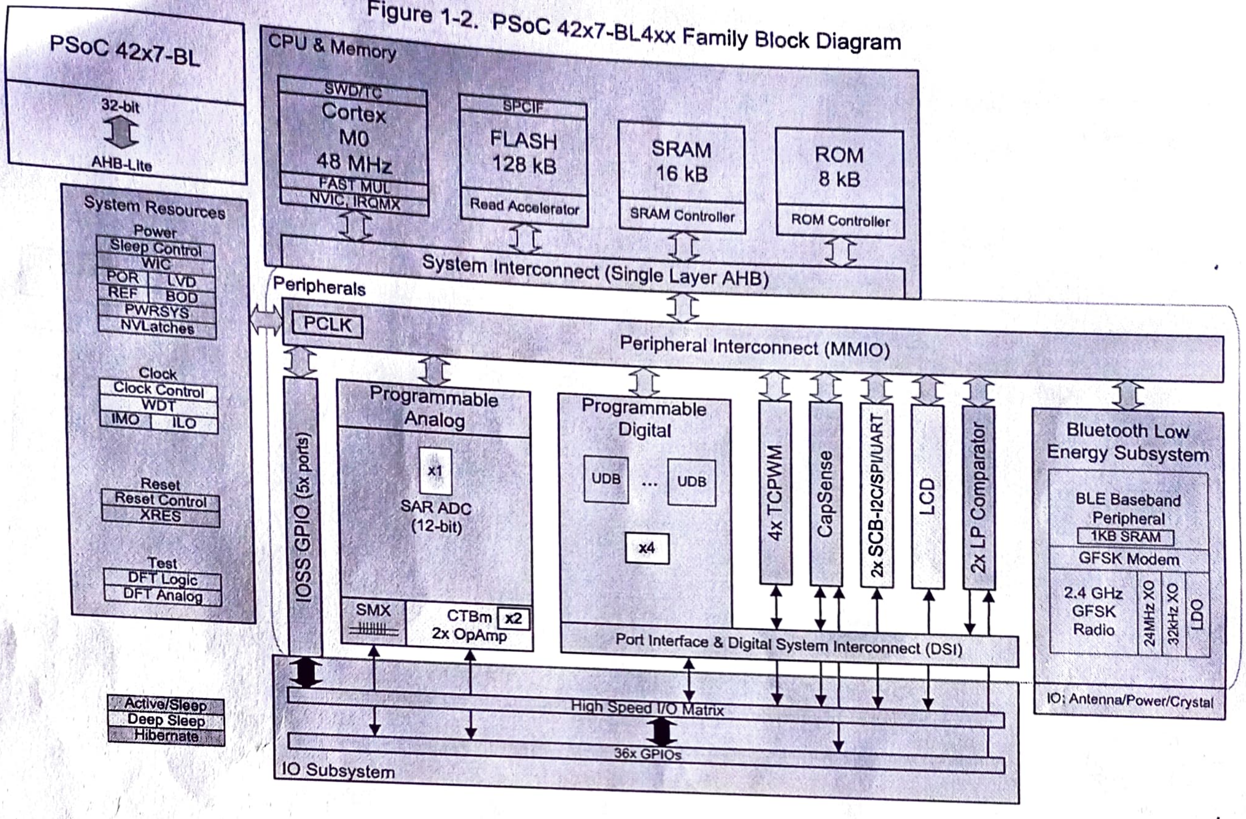


Figure 1-4. PSoC 42x8-BL4xx Family Block Diagram

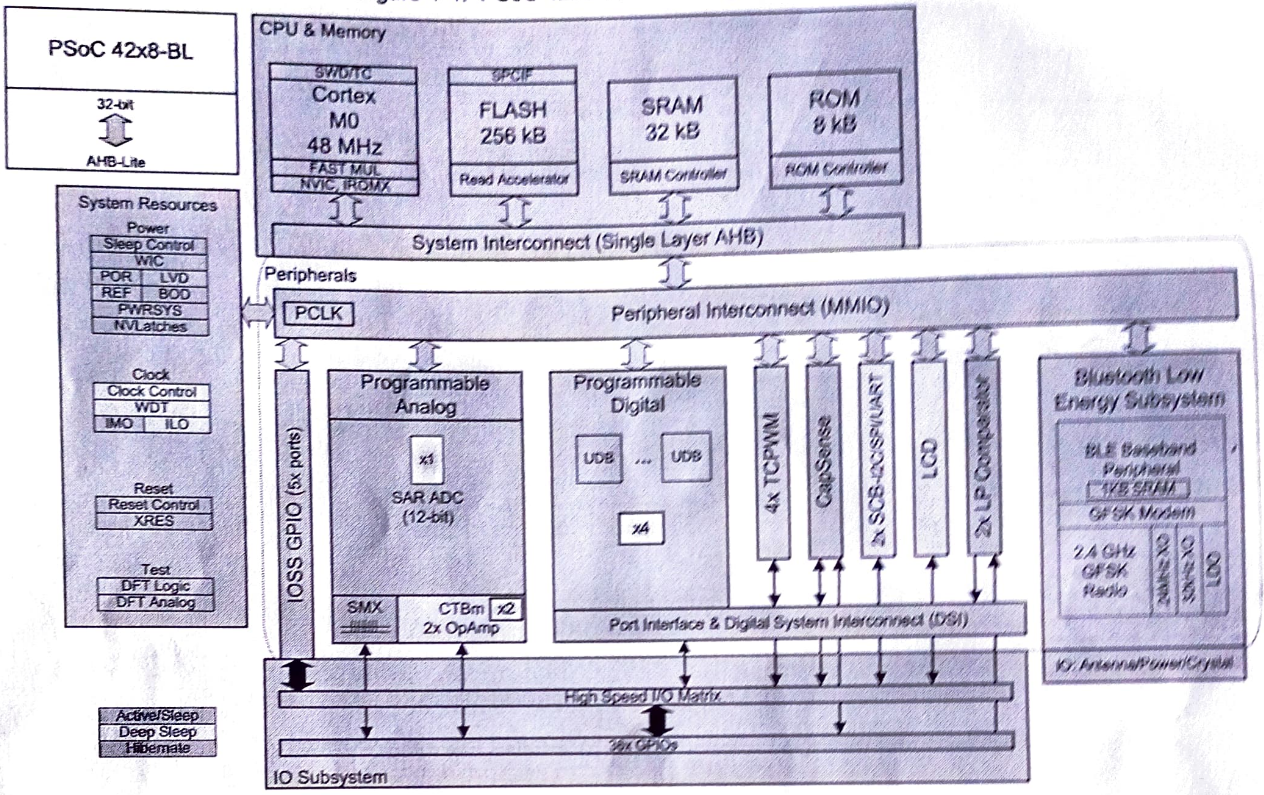


Figure 1-5. PSoC 41x8-BL5xx Family Block Diagram

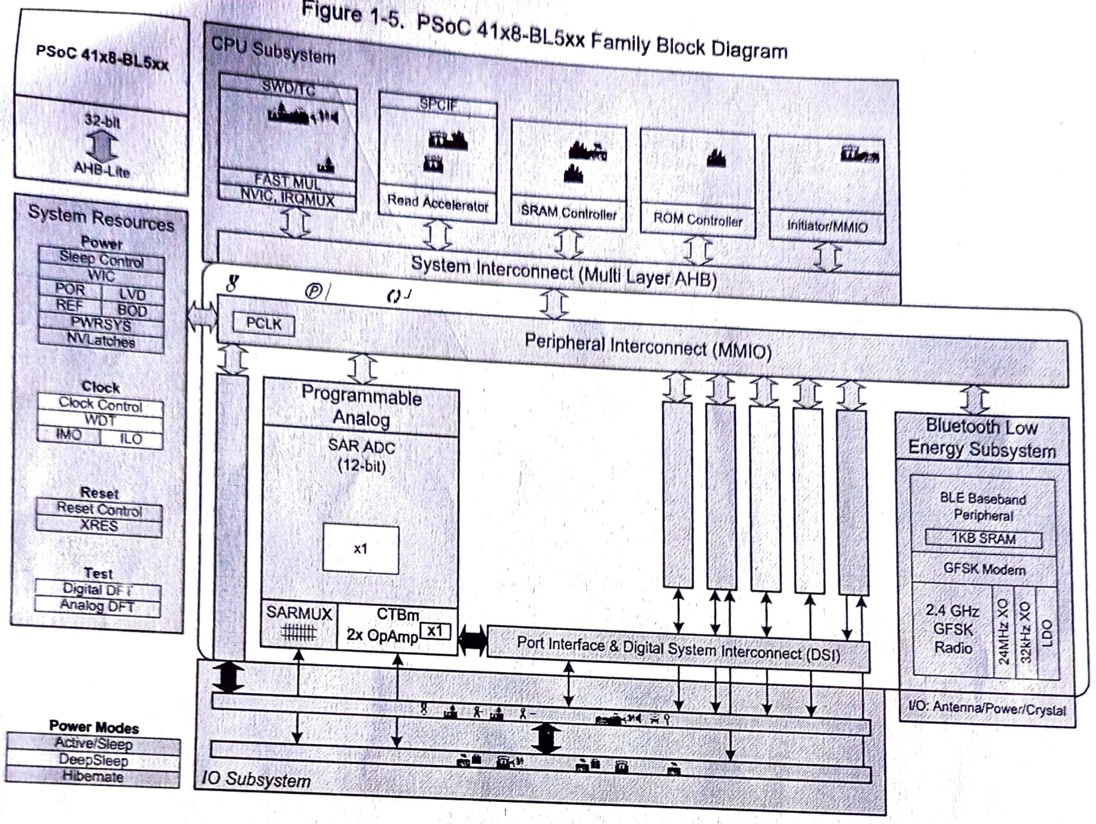
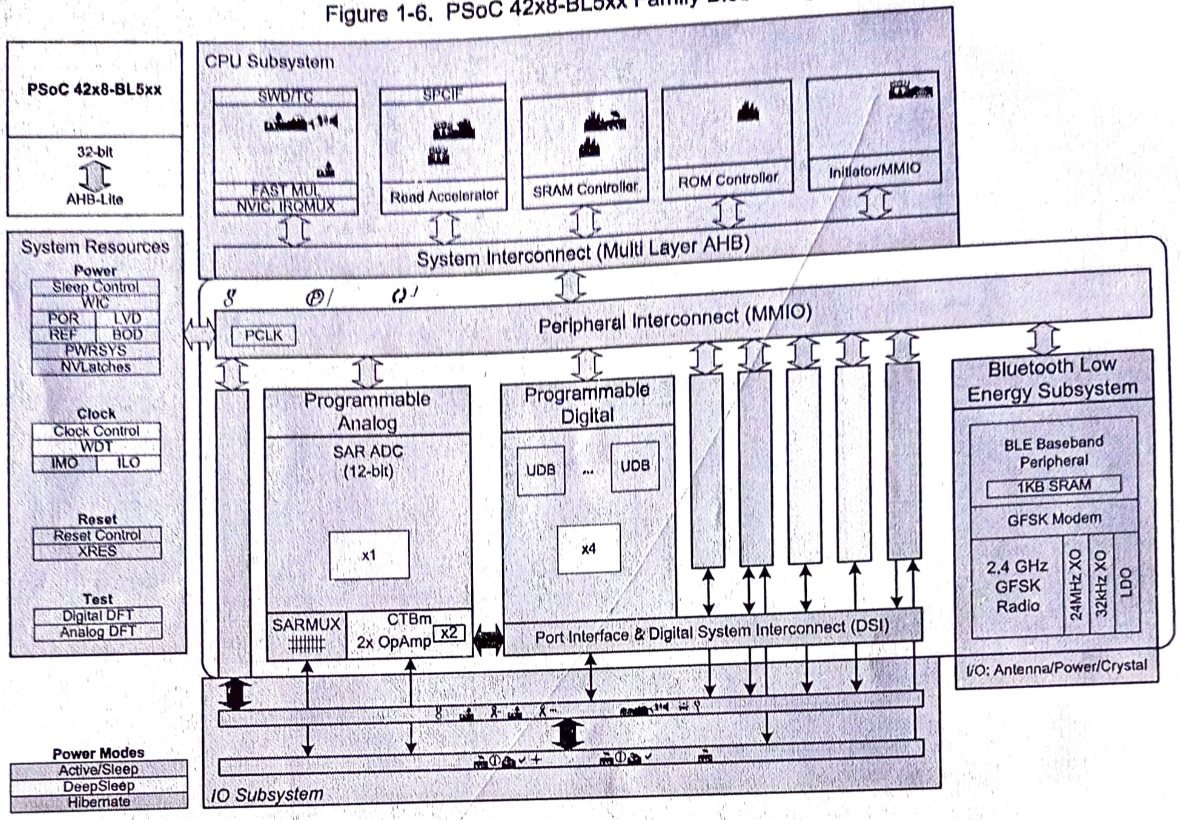


Figure 1-6. PSoC 42x8-BL5xx Family Block Diagram



1.2 Features

The PSoC 4xxx-BL family has these major components:

- BLE radio and subsystem
- 32-bit Cortex-M0 CPU with single-cycle multiply, delivering up to 43 DMIPS at 48 MHz
- Up to 256 KB flash and 32 KB SRAM
- Direct memory access (DMA)
- Four independent center-aligned pulse-width modulators (PWMs) with complementary, dead-band programmable outputs
- Twelve-bit SAR ADC (with a sampling rate of 1 Msps in PSoC 42xx-BL and 806 ksps in PSoC 41xx-BL) with hardware sequencing for multiple channels
- Up to four opamps that can be used for analog signal conditioning and as a comparator
- Two low-power comparators
- Two serial communication blocks (SCB) that can work as SPI, UART, I^{2}C, and local interconnect network (LIN) slave serial communication channels
- Up to four programmable logic blocks, known as universal digital blocks (UDBs)
- CapSense
- Segment LCD direct drive
- Low-power operating modes: Sleep, Deep-Sleep, Hibernate, and Stop
- Programming and debugging system through serial wire debug (SWD)
- Fully supported by PSoC Creator™ IDE tool

1.3 CPU System

1.3.1 Processor

The heart of the PSoC 4 is a 32-bit Cortex-M0 CPU core running up to 48 MHz for PSoC 42xx-BL and 24 MHz for PSoC 41xx-BL. It is optimized for low-power operation with extensive clock gating. It uses 16-bit instructions and executes a subset of the Thumb-2 instruction set. This instruction set enables fully compatible binary upward migration of the code to higher performance processors such as Cortex M3 and M4.

The CPU has a hardware multiplier that provides a 32-bit result in one cycle.

1.3.2 Interrupt Controller

The CPU subsystem includes a nested vectored interrupt controller (NVIC) with 32 interrupt inputs and a wakeup interrupt controller (WIC), which can wake the processor from Deep-Sleep mode. The Cortex-M0 CPU of PSoC 4 implements a non-maskable interrupt (NMI) input, which can be tied to digital routing for general-purpose use.

1.3.3 Direct Memory Access

The DMA engine is capable of independent data transfers anywhere within the memory map (peripheral-to-peripheral and peripheral-to/from-memory) with a programmable descriptor chain.

Note: DMA is available only in PSoC 41x8-BL5xx and PSoC 42x8-BL5xx families.

1.4 Memory

The PSoC 4 memory subsystem consists of flash and SRAM. A supervisory ROM, containing boot and configuration routines, is also present.

1.4.1 Flash

The PSoC 4 has a flash module, with a flash accelerator tightly coupled to the CPU, to improve average access times from the flash block. The flash accelerator delivers 85 percent of single-cycle SRAM access performance on an average.

1.4.2 SRAM

The PSoC 4 provides SRAM, which is retained during Hibernate mode.

1.5 System-Wide Resources

1.5.1 Clocking System

The clocking system for the PSoC 4 device consists of the internal main oscillator (IMO) and internal low-speed oscillator (ILO) as internal clocks and has provision for an external clock, external crystal oscillator (ECO), and watch crystal oscillator (WCO).

The IMO with an accuracy of ±2 percent is the primary source of internal clocking in the device. Multiple clock derivatives are generated from the main clock frequency to meet various application needs.

The ILO is a low-power, less accurate oscillator and is used as a source for LFCLK, to generate clocks for peripheral operation in Deep-Sleep mode. Its clock frequency is 32 kHz with ±60 percent accuracy.

An external clock source ranging from 0 MHz to 48 MHz can be used to generate the clock derivatives for the functional blocks instead of the IMO.

The ECO is used to generate a highly accurate 24-MHz clock without any external components. It is primarily used to clock the BLE subsystem, which contains the Link Layer engine, the digital PHY modem, and the RF transceiver. The high-accuracy ECO clock can also be used as a clock source for the PSoC 4 device.

The WCO is used as a source for LFCLK. WCO is used to accurately maintain the time interval of advertising events and connection events during Deep Sleep mode. Similar to the ILO, WCO is also available in all modes, except Hibernate and Stop modes.

1.5.2 Power System

The PSoC 4 operates with a single external supply in the range 1.71 V to 5.5 V.

PSoC 4 has four low-power modes – Sleep, Deep-Sleep, Hibernate, and Stop – in addition to the default Active mode. In Active mode, the CPU runs with all the logic powered. In Sleep mode, the CPU is powered off with all other peripherals functional. In Deep-Sleep mode, the CPU, SRAM, and high-speed logic are in retention; the main system clock is OFF while the low-frequency clock is ON and the low-frequency peripherals are in operation. In Hibernate mode, even the low-frequency clock is OFF and low-frequency peripherals stop operating.

Multiple internal regulators are available in the system to support power supply schemes in different power modes.

1.5.3 GPIO

Every GPIO in PSoC 4 has the following characteristics:

- Eight drive strength modes
- Individual control of input and output disables
- Hold mode for latching previous state
- Selectable slew rates
- Interrupt generation – edge triggered
- CapSense and LCD drive support

PSoC 4 also has two over-voltage tolerant ports , which enable I2C Fast Mode power down specification compliance and have the ability to connect to higher voltage buses while operating at lower V_{DD}.

The pins are organized in a port that is 8-bit wide. A high-speed I/O matrix is used to multiplex between various signals that may connect to an I/O pin. Pin locations for fixed-function peripherals are also fixed.

1.6 Bluetooth Low-Energy Subsystem

PSoC 4xxx Bluetooth Low-Energy (BLE) subsystem integrates the RF transceiver, digital PHY modem, and link layer controller.

1.6.1 RF Transceiver

The RF transceiver contains an integrated balun, which provides a single-ended RF port pin to drive a 50-ohm antenna via a matching/filtering network. In the receive direction, this block converts the RF signal from the antenna to a 1-MHz intermediate frequency and digitizes the analog signal to 10-bit digital signal. In the transmit direction, this block takes 1 Mbps GFSK modulated from digital PHY, up-converts it to radio frequency, and transmit it to air through antenna.

1.6.2 Digital PHY Modem

In the transmit direction, this sub-block takes the 1-Mbps serial data from the link layer controller, generates GFSK direct modulated data, and sends it to the BLE analog section. On the receive side, it takes the 1-MHz IF ADC data from the BLE analog section and uses digital demodulator to generate the 1-Mbps serial data.

1.6.3 Link Layer Controller

The link layer controller implements all timing critical functions specified in the Bluetooth Low-Energy Link Layer specifications (packet framing/de-framing, CRC generation/checking, encryption/decryption, state machines, and packet transmission); it also provides interface to the digital PHY. The communication between link layer hardware and firmware is done through interrupt, FIFO, and registers.

1.7 Programmable Digital

The PSoC 42xx-BL has up to four UDBs. Each UDB contains structured data-path logic and uncommitted PLD logic with flexible interconnect. The UDB array provides a switched routing fabric called the digital signal interconnect (DSI). The DSI allows routing of signals from peripherals and ports to and within the UDBs.

The UDB arrays in PSoC 42xx-BL enable custom logic or additional timers/PWMs and communication interfaces such as I²C, SPI, I2S, and UART.

Note PSoC 41xx-BL does not have UDBs.

1.8 Fixed-Function Digital

1.8.1 Timer/Counter/PWM Block

The Timer/Counter/PWM block consists of four 16-bit counters with user-programmable period length. The functionality of these counters can be synchronized. Each block has a capture register, period register, and compare register. The block supports complementary, dead-band programmable outputs. It also has a kill input to force outputs to a predetermined state. Other features of the block include center-aligned PWM, clock prescaling, pseudo random PWM, and quadrature decoding.

1.8.2 Serial Communication Blocks

The device has two SCBs. Each SCB can implement a serial communication interface as I²C, UART, local interconnect network (LIN) slave, or SPI.

The features of each SCB include:

- Standard I²C multi-master and slave function
- Standard SPI master and slave function with Motorola, Texas Instruments, and National (MicroWire) mode
- Standard UART transmitter and receiver function with SmartCard reader (ISO7816), IrDA protocol, and LIN
- Standard LIN slave with LIN v1.3 and LIN v2.1/2.2 specification compliance
- EZ function mode support for SPI and I²C with 32-byte buffer

1.9 Analog System

1.9.1 SAR ADC

PSoC 42xx-BL has a configurable 12-bit 1-Msps SAR ADC and PSoC 41xx-BL has a similar 12-bit SAR ADC with 806 ksps. The ADC provides three internal voltage references (V_{DDA}, $V_{DDA}/2$, and V_{REF}) and an external reference through a GPIO pin. The SAR hardware sequencer is available, which scans multiple channels without CPU intervention.

1.9.2 Continuous Time Block mini

The Continuous Time Block mini (CTBm) provides continuous time functionality at the entry and exit points of the analog subsystem. The CTBm has two highly configurable and high-performance opamps with a switch routing matrix. The opamps can also work in comparator mode. PSoC 42xx-BL has two such CTBm blocks, while PSoC 41xx-BL has one CTBm block.

The block allows open-loop opamp, linear buffer, and comparator functions to be performed without external components. PGAs, voltage buffers, filters, and trans-impedance amplifiers can be realized with external components.CTBm block can work in Active, Sleep, and Deep-Sleep modes.

1.9.3 Low-Power Comparators

The PSoC 4xxx-BL has a pair of low-power comparators, which can operate in all device power modes. This functionality allows the CPU and other system blocks to be disabled while retaining the ability to monitor external voltage levels during low-power modes. Two input voltages can both come from pins, or one from an internal signal through the AMUX-BUS.

1.10 Special Function Peripherals

1.10.1 LCD Segment Drive

The PSoC 4 has an LCD controller, which can drive up to four commons and every GPIO can be configured to drive common or segment. It uses full digital methods (digital correlation and PWM) to drive the LCD segments, and does not require generation of internal LCD voltages.

1.10.2 CapSense

PSoC 4 devices have the CapSense feature, which allows you to use the capacitive properties of your fingers to toggle buttons and sliders. CapSense functionality is supported on all GPIO pins in PSoC 4 through a CapSense Sigma-Delta (CSD) block. The CSD also provides waterproofing capability.

1.10.2.1 IDACs and Comparator

The CapSense block has two IDACs and a comparator with a 12-V reference, which can be used for general purposes, if CapSense is not used.

1.11 Program and Debug

PSoC 4 devices support programming and debugging features of the device via the on-chip SWD interface. The PSoC Creator IDE provides fully integrated programming and debugging support. The SWD interface is also fully compatible with industry standard third-party tools.

1.12 Device Feature Summary

Table 1-1 shows the PSoC 41xx-BL/42xx-BL device summary.

Table 1-1. PSoC 41xx-BL/42xx-BL Device Summary

Feature	PSoC 41xx-BL	PSoC 42xx-BL
Maximum CPU Frequency	24 MHz	48 MHz
Flash	PSoC 41x7-BL: 128 KB PSoC 41x8-BL: 256 KB	PSoC 42x7-BL: 128 KB PSoC 42x8-BL: 256 KB
SRAM	PSoC 41x7-BL: 16 KB PSoC 41x8-BL: 32 KB	PSoC 42x7-BL: 16 KB PSoC 42x8-BL: 32 KB
GPIOs (maximum)	38	38
CapSense	Available	Available
LCD Driver	Available	Available
Timer, Counter, PWM (TCPWM)	4	4
Serial Communication Block (SCB)	2	2
Universal Digital Block (UDB)	Not Available	4
IDAC (part of CapSense)	2	2
Opamp	2	4
Comparator	2	2
ADC	12-bit SAR, 806 ksps	12-bit SAR, 1 Msps
Bluetooth	Available	Available

4. Cortex-M0 CPU

The PSoC® 4 ARM Cortex-M0 core is a 32-bit CPU optimized for low-power operation. It has an efficient three-stage pipeline, a fixed 4-GB memory map, and supports the ARMv6-M Thumb instruction set. The Cortex-M0 also features a single-cycle 32-bit multiply instruction and low-latency interrupt handling. Other subsystems tightly linked to the CPU core include a nested vectored interrupt controller (NVIC), a SYSTICK timer, and debug.

This section gives an overview of the Cortex-M0 processor. For more details, see the ARM Cortex-M0 user guide or technical reference manual, both available at www.arm.com.

4.1 Features

The PSoC 4 Cortex-M0 has the following features:

- Easy to use, program, and debug, ensuring easier migration from 8- and 16-bit processors
- Operates at up to 0.9 DMIPS/MHz; this helps to increase execution speed or reduce power
- Maximum CPU clock frequency of 24 MHz in PSoC41xx_BL and 48 MHz in PSoC 42xx_BL.
- Supports the Thumb instruction set for improved code density, ensuring efficient use of memory
- NVIC unit to support interrupts and exceptions for rapid and deterministic interrupt response
- Extensive debug support including:
  - SWD port
  - Breakpoints
  - Watchpoints

4.2 Block Diagram

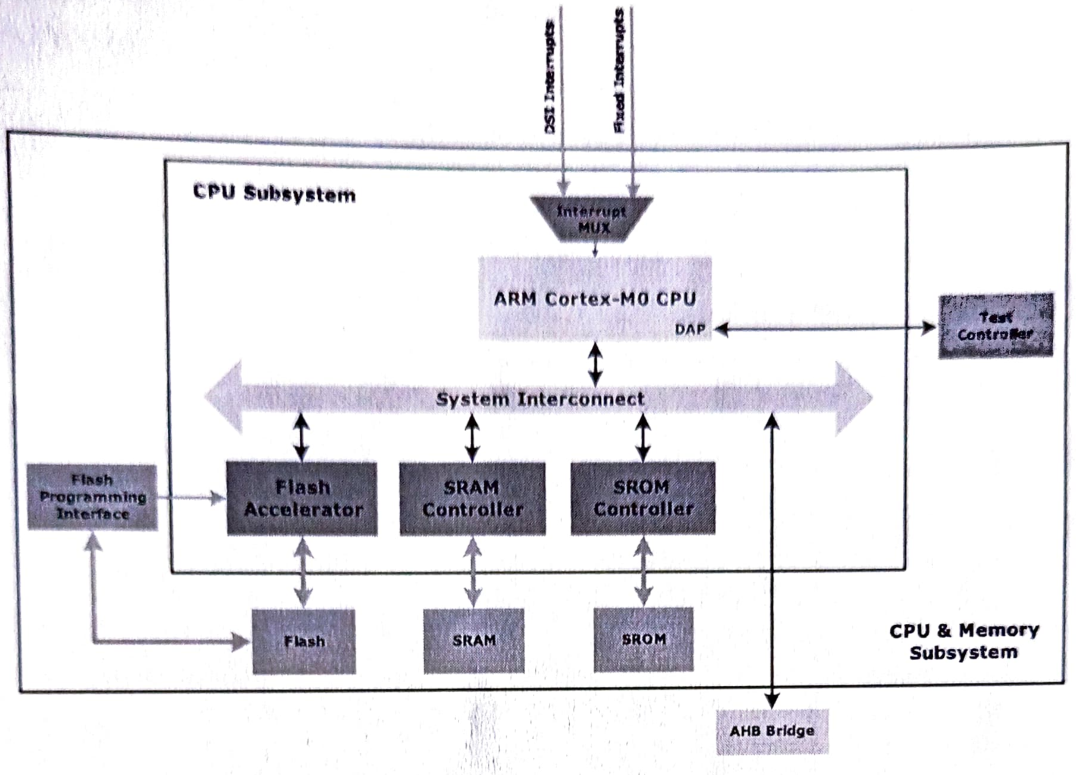


Figure 4-1. PSoC 4 CPU Subsystem Block Diagram

4.3 How It Works

The Cortex-M0 is a 32-bit processor with a 32-bit data path, 32-bit registers, and a 32-bit memory interface. It supports most 16-bit instructions in the Thumb instruction set and some 32-bit instructions in the Thumb-2 instruction set.

The processor supports two operating modes (see "Operating Modes" on page 40). It has a single-cycle 32-bit multiplication instruction.

4.4 Address Map

The ARM Cortex-M0 has a fixed address map allowing access to memory and peripherals using simple memory access instructions. The 32-bit (4 GB) address space is divided into the regions shown in Table 4-1. Note that code can be executed from the code and SRAM regions.

Table 4-1. Cortex-M0 Address Map

Address Range	Name	Use
0x00000000 - 0x1FFFFFFF	Code	Program code region. You can also place data here. Includes the exception vector table, which starts at address 0.
0x20000000 - 0x3FFFFFFF	SRAM	Data region. You can also execute code from this region.
0x40000000 - 0x5FFFFFFF	Peripheral	All peripheral registers. You cannot execute code from this region.
0x60000000 - 0xDFFFFFFF		Not used.
0xE0000000 - 0xE00FFFFF	PPB	Peripheral registers within the CPU core.
0xE0100000 - 0xFFFFFFFF	Device	PSoC 4 implementation-specific.

4.5 Registers

The Cortex-M0 has 16 32-bit registers, as Table 4-2 shows:

- R0 to R12 – General-purpose registers. R0 to R7 can be accessed by all instructions; the other registers can be accessed by a subset of the instructions.
- R13 – Stack pointer (SP). There are two stack pointers, with only one available at a time. In thread mode, the CONTROL register indicates the stack pointer to use, Main Stack Pointer (MSP) or Process Stack Pointer (PSP).
- R14 – Link register. Stores the return program counter during function calls.
- R15 – Program counter. This register can be written to control program flow.

Table 4-2. Cortex-M0 Registers

Name	Type[a]	Reset Value	Description
R0-R12	RW	Undefined	R0-R12 are 32-bit general-purpose registers for data operations.
MSP (R13) PSP (R13)	RW	[0x00000000]	The stack pointer (SP) is register R13. In thread mode, bit[1] of the CONTROL register indicates which stack pointer to use: 0 = Main stack pointer (MSP). This is the reset value. 1 = Process stack pointer (PSP). On reset, the processor loads the MSP with the value from address 0x00000000.
LR (R14)	RW	Undefined	The link register (LR) is register R14. It stores the return information for subroutines, function calls, and exceptions.
PC (R15)	RW	[0x00000004]	The program counter (PC) is register R15. It contains the current program address. On reset, the processor loads the PC with the value from address 0x00000004. Bit[0] of the value is loaded into the EPSR T-bit at reset and must be 1.
PSR	RW	Undefined	The program status register (PSR) combines: Application Program Status Register (APSR). Execution Program Status Register (EPSR). Interrupt Program Status Register (IPSR).
APSR	RW	Undefined	The APSR contains the current state of the condition flags from previous instruction executions.
EPSR	RO	[0x00000004].0	On reset, EPSR is loaded with the value bit[0] of the register [0x00000004].
IPSR	RO	0	The IPSR contains the exception number of the current ISR.
PRIMASK	RW	0	The PRIMASK register prevents activation of all exceptions with configurable priority.
CONTROL	RW	0	The CONTROL register controls the stack used when the processor is in thread mode.

a. Describes access type during program execution in thread mode and handler mode. Debug access can differ.

Table 4-3 shows how the PSR bits are assigned.

Table 4-3. Cortex-M0 PSR Bit Assignments

Bit	PSR Register	Name	Usage
31	APSR	N	Negative flag
30	APSR	Z	Zero flag
29	APSR	C	Carry or borrow flag
28	APSR	V	Overflow flag

Table 4-3. Cortex-M0 PSR Bit Assignments

Bit	PSR Register	Name	Usage
27 – 25	–	–	Reserved
24	EPSR	T	Thumb state bit. Must always be 1. Attempting to execute instructions when the T bit is 0 results in a HardFault exception.
23 – 6	–	–	Reserved
5 – 0	IPSR	N/A	Exception number of current ISR: 0 = thread mode 1 = reserved 2 = NMI 3 = HardFault 4 – 10 = reserved 11 = SVCall 12, 13 = reserved 14 = PendSV 15 = SysTick 16 = IRQ0 ... 47 = IRQ31

Use the MSR or CPS instruction to set or clear bit 0 of the PRIMASK register. If the bit is 0, exceptions are enabled. If the bit is 1, all exceptions with configurable priority, that is, all exceptions except HardFault, NMI, and Reset, are disabled. See the Interrupts chapter on page 57 for a list of exceptions.

4.6 Operating Modes

The Cortex-M0 processor supports two operating modes:

- Thread Mode – used by all normal applications. In this mode, the MSP or PSP can be used. The CONTROL register bit 1 determines which stack pointer is used:
  - 0 = MSP is the current stack pointer
  - 1 = PSP is the current stack pointer
- Handler Mode – used to execute exception handlers. The MSP is always used.

In thread mode, use the MSR instruction to set the stack pointer bit in the CONTROL register. When changing the stack pointer, use an ISB instruction immediately after the MSR instruction. This ensures that instructions after the ISB execute using the new stack pointer.

In handler mode, explicit writes to the CONTROL register are ignored, because the MSP is always used. The exception entry and return mechanisms automatically update the CONTROL register.

4.7 Instruction Set

The Cortex-M0 implements a version of the Thumb instruction set, as Table 4-4 shows. For details, see the *Cortex-M0 Generic User Guide.*

An instruction operand can be an ARM register, a constant, or another instruction-specific parameter. Instructions act on the operands and often store the result in a destination register. Many instructions are unable to use, or have restrictions on using, the PC or SP for the operands or destination register.

Table 4-4. Thumb Instruction Set

Mnemonic	Brief Description
ADCS	Add with carry
ADD{S}[a]	Add
ADR	PC-relative address to register
ANDS	Bit wise AND
ASRS	Arithmetic shift right
B{cc}	Branch {conditionally}
BICS	Bit clear
BKPT	Breakpoint
BL	Branch with link
BLX	Branch indirect with link
BX	Branch indirect
CMN	Compare negative
CMP	Compare
CPSID	Change processor state, disable interrupts
CPSIE	Change processor state, enable interrupts
DMB	Data memory barrier
DSB	Data synchronization barrier
EORS	Exclusive OR
ISB	Instruction synchronization barrier
LDM	Load multiple registers, increment after
LDR	Load register from PC-relative address
LDRB	Load register with word
LDRH	Load register with half-word
LDRSB	Load register with signed byte
LDRSH	Load register with signed half-word
LSLS	Logical shift left
LSRS	Logical shift right
MOV{S}[a]	Move
MRS	Move to general register from special register
MSR	Move to special register from general register
MULS	Multiply, 32-bit result
MVNS	Bit wise NOT
NOP	No operation
ORRS	Logical OR
POP	Pop registers from stack
PUSH	Push registers onto stack
REV	Byte-reverse word
REV16	Byte-reverse packed half-words
REVSH	Byte-reverse signed half-word
RORS	Rotate right
RSBS	Reverse subtract
SBCS	Subtract with carry
SEV	Send event
STM	Store multiple registers, increment after
STR	Store register as word
STRB	Store register as byte
STRH	Store register as half-word
SUB{S}[a]	Subtract
SVC	Supervisor call
SXTB	Sign extend byte
SXTH	Sign extend half-word
TST	Logical AND-based test
UXTB	Zero extend a byte
UXTH	Zero extend a half-word
WFE	Wait for event
WFI	Wait for interrupt

a. The 'S' qualifier causes the ADD, SUB, or MOV instructions to update APSR condition flags.

4.7.1 Address Alignment

An aligned access is an operation where a word-aligned address is used for a word or multiple word access, or where a half-word-aligned address is used for a half-word access. Byte accesses are always aligned.

No support is provided for unaligned accesses on the Cortex-M0 processor. Any attempt to perform an unaligned memory access operation results in a HardFault exception.

4.7.2 Memory Endianness

The PSoC 4 Cortex-M0 uses the little-endian format, where the least-significant byte of a word is stored at the lowest address and the most significant byte is stored at the highest address.

4.8 Systick Timer

The Systick timer is integrated with the NVIC and generates the SYSTICK interrupt. This interrupt can be used for task management in a real-time system. The timer has a reload register with 24 bits available to use as a countdown value. The Systick timer uses the Cortex-M0 internal clock as a source.

4.9 Debug

PSoC 4 contains a debug interface based on SWD; it features four breakpoint (address) comparators and two watchpoint (data) comparators.

IOT UINIT-4

DATA ACQUIRING AND STORAGE:

Following subsections describe devices data, and steps in acquiring and storing data for an application, service or business process.

Data Generation:

- Data generates at devices that later on, transfers to the Internet through a gateway. Data generates as follows:
- ● Passive devices data: Data generate at the device or system, following the result of interactions. A passive device does not have its own power source. An external source helps such a device to generate and send data. Examples are an RFID (Example 2.2) or an ATM debit card (Example 2.3). The device may or may not have an associated microcontroller, memory and transceiver. A contactless card is an example of the former and a label or barcode is the example of the latter.
- Active devices data: Data generates at the device or system or following the result of interactions. An active device has its own power source. Examples are active RFID, streetlight sensor (Example 1.2) or wireless sensor node. An active device also has an associated microcontroller, memory and transceiver.
- Event data: A device can generate data on an event only once. For example, on detection of the traffic or on dark ambient conditions, which signals the event. The event on darkness communicates a need for lighting up a group of streetlights (Example 1.2). A system consisting of security cameras can generate data on an event of security breach or on detection of an intrusion. A waste container with associate circuit can generate data in the event of getting it filled up 90% or above. The components and devices in an automobile generate data of their performance and functioning. For example, on wearing out of a brake lining, a play in steering wheel and reduced air-conditioning is felt. The data communicates to the Internet. The communication takes place as and when the automobile reaches near a Wi-Fi access point.
- Device real-time data: An ATM generates data and communicates it to the server instantaneously through the Internet. This initiates and enables Online Transactions Processing (OLTP) in real time.

- Event-driven device data: A device data can generate on an event only once. Examples are: (i) a device receives command from Controller or Monitor, and then performs action(s) using an actuator. When the action completes, then the device sends an acknowledgement; (ii) When an application seeks the status of a device, then the device communicates the status.

Data Acquisition:

Data acquisition means acquiring data from IoT or M2M devices. The data communicates after the interactions with a data acquisition system (application). The application interacts and communicates with a number of devices for acquiring the needed data. The devices send data on demand or at programmed intervals. Data of devices communicate using the network, transport and security layers (Figure 2.1). An application can configure the devices for the data when devices have configuration capability. For example, the system can configure devices to send data at defined periodic intervals. Each device configuration controls the frequency of data generation. For example, system can configure an umbrella device to acquire weather data from the Internet weather service, once each working day in a week (Example 1.1). An ACVM can be configured to communicate the sales data of machine and other information, every hour. The ACVM system can be configured to communicate instantaneously in event of fault or in case requirement of a specific chocolate flavour needs the Fill service

- Application can configure sending of data after filtering or enriching at the gateway at the data-adaptation layer. The gateway in-between application and the devices can provision for one or more of the following functions—transcoding, data management and device management. Data management may be provisioning of the privacy and security, and data integration, compaction and fusion (Section 2.3).

- Device-management software provisions for device ID or address, activation, configuring (managing device parameters and settings), registering, deregistering, attaching, and detaching (Section 2.3.2). Example 5.2 gives the process of acquiring data from the embedded component devices in the automobiles for Automotive Components and Predictive Automotive Maintenance System (ACPAMS) application.

Data Validation:

- Data acquired from the devices does not mean that data are correct, meaningful or consistent. Data consistency means within expected range data or as per pattern or data not corrupted during transmission. Therefore, data needs validation checks. Data validation software do the validation checks on the acquired data. Validation software applies logic, rules and semantic annotations. The applications or services depend on valid data. Then only the analytics, predictions, prescriptions, diagnosis and decisions can be acceptable

- Large magnitude of data is acquired from a large number of devices, especially, from machines in industrial plants or embedded components data from large number of automobiles or health devices in ICUs or wireless sensor networks, and so on. Validation software, therefore, consumes significant resources. An appropriate strategy needs to be adopted. For example, the adopted strategy may be filtering out the invalid data at the gateway or at device itself or controlling the frequency of acquiring or cyclically scheduling the set of devices in industrial systems. Data enriches, aggregates, fuses or compacts at the adaptation layer.

⦿ Data Categorisation for Storage:

- ⦿ Data from large number of devices and sources categorises into a fourth category called Big data. Data is stored in databases at a server or in a data warehouse or on a Cloud as Big data.
- ⦿ Assembly Software for the Events A device can generate events. For example, a sensor can generate an event when temperature reaches a preset value or falls below a threshold. A pressure sensor in a boiler generates an event when pressure exceeds a critical value which warrants attention.

⦿ Each event can be assigned an ID. A logic value sets or resets for an event state. Logic 1 refers to an event generated but not yet acted upon. Logic 0 refers to an event generated and acted upon or not yet generated. A software component in applications can assemble the events (logic value, event ID and device ID) and can also add Date time stamp. Events from IoTs and logic-flows assemble using software.

⦿ Data Store:

⦿ A data store is a data repository of a set of objects which integrate into the store. Features of data store are: ● Objects in a data-store are modeled using Classes which are defined by the database schemas. ● A data store is a general concept. It includes data repositories such as database, relational database, flat file, spreadsheet, mail server, web server, directory services and VMware ● A data store may be distributed over multiple nodes. Apache Cassandra is an example of distributed data store.

⦿ A data store may consist of multiple schemas or may consist of data in only one scheme. Example of only one scheme data store is a relational database. Repository in English means a group, which can be related upon to look for required things, for special information or knowledge

⦿ For example, a repository of paintings of artists. A database is a repository of data which can be relied upon for reporting, analytics, process, knowledge discovery and intelligence.

- A flat file is another repository. Flat file means a file in which the records have no structural interrelationship (Section 5.3). Section 5.5.1 explains the spreadsheet concept. VMware uses data store to refer to a file that stores a virtual machine

⦿ Data Centre Management:

- A data centre is a facility which has multiple banks of computers, servers, large memory systems, high speed network and Internet connectivity. The centre provides data security and protection using advanced tools, full data backups along with data recovery, redundant data communication connections and full system power as well as electricity supply backups.
- Large industrial units, banks, railways, airlines and units for whom data are the critical components use the services of data centres. Data centres also possess a dust free, heating, ventilation and air conditioning (HVAC), cooling, humidification and dehumidification equipment, pressurisation system with a physically highly secure environment.
- The manager of data centre is responsible for all technical and IT issues, operations of computers and servers, data entries, data security, data quality control, network quality control and the management of the services and applications used for data processing

⦿ Server Management:

- Server management means managing services, setup and maintenance of systems of all types associated with the server.
- A server needs to serve around the clock. Server management includes managing the following:
- ● Short reaction times when the system or network is down
- ● High security standards by routinely performing system maintenance and updation
- ● Periodic system updates for state-of-the art setups ● Optimised performance
- ● Monitoring of all critical services, with SMS and email notifications
- ● Security of systems and protection
- ● Maintaining confidentiality and privacy of data

⦿ ● High degree of security and integrity and effective protection of data, files and databases at the organisation

⦿ ● Protection of customer data or enterprise internal documents by attackers which includes spam mails, unauthorised use of the access to the server, viruses, malwares and worms

⦿ ● Strict documentation and audit of all activities

⦿ Spatial Storage:

⦿ Consider goods with RFID tags. When goods move from one place to another, the IDs of goods as well as locations are needed in tracking or inventory control applications. Spatial storage is storage as spatial database which is optimised to store and later on receives queries from the applications. Suppose a digital map is required for parking slots in a city. Spatial data refers to data which represents objects defined in a geometric space. Points, lines and polygons are common geometric objects which can be represented in spatial databases. Spatial database can also represent database for 3D objects, topological coverage, linear networks, triangular irregular networks and other complex structures. Additional functionality in spatial databases enables efficient processing

⦿ Internet communication by RFIDs, ATMs, vehicles, ambulances, traffic lights, streetlights, waste containers are examples of where spatial database are used.

⦿ Spatial database functions optimally for spatial queries. A spatial database can perform typical SQL queries, such as select statements and performs a wide variety of spatial operations. Spatial database has the following features:

⦿ ● Can perform geometry constructors. For example, creating new geometries

⦿ ● Can define a shape using the vertices (points or nodes)

⦿ ● Can perform observer functions using queries which replies specific spatial information such as location of the centre of a geometric object Can perform spatial measurements which mean computing distance between geometries, lengths of lines, areas of polygons and other parameters

⦿ Can change the existing features to new ones using spatial functions and can predicate spatial relationships between geometries using true or false type queries

- Can perform spatial measurements which mean computing distance between geometries, lengths of lines, areas of polygons and other parameters
- ● Can change the existing features to new ones using spatial functions and can predicate spatial relationships between geometries using true or false type queries

Cloud Computing Features and Advantages:

- Essential features of cloud storage and computing are:
- ● On demand self-service to users for the provision of storage, computing servers, software delivery and server time
- ● Resource pooling in multi-tenant model
- ● Broad network accessibility in virtualised environment to heterogeneous users, clients, systems and devices
- ● Elasticity
- ● Massive scale availability
- ● Scalability
- ● Maintainability
- ● Homogeneity

● Virtualisation

Cloud Computing Concerns:

- Concerns in usage of cloud computing are:
- ● Requirement of a constant high-speed Internet connection
- Limitations of the services available
- ● Possible data loss
- ● Non delivery as per defined SLA specified performance
- ● Different APIs and protocols used at different clouds
- ● Security in multi-tenant environment needs high trust and low risks

- ● **Loss of users’ control**

Cloud Deployment Models:

- Following are the four cloud deployment models: 1. Public cloud: This model is provisioned by educational institutions, industries, government institutions or businesses or enterprises and is open for public use.
- 2. Private cloud: This model is exclusive for use by institutions, industries, businesses or enterprises and is meant for private use in the organisation by the employees and associated users only.
- 3. Community cloud: This model is exclusive for use by a community formed by institutions, industries, businesses or enterprises, and for use within the community organisation, employees and associated users. The community specifies security and compliance considerations
- 4. Hybrid cloud: A set of two or more distinct clouds (public, private or community) with distinct data stores and applications that bind between them to deploy the proprietary or standard technology.
- Cloud platform architecture is a virtualised network architecture consisting of a cluster of connected servers over the data centres and Service Level Agreements (SLAs) between them.
- A cloud platform controls and manages resources, and dynamically provisions the networks, servers and storage. Cloud platform applications and network services are utility, grid and distributed services. Examples of cloud platforms are Amazon EC2, Microsoft Azure, Google App Engine, Xively, Nimbits, AWS IoT, CISCO IoT, IOx and Fog, IBM IoT Foundation, TCS Connected Universe Platform.

EVERYTHING AS A SERVICE AND CLOUD SERVICE MODELS:

- Cloud connects the devices, data, applications, services, persons and business. Cloud services can be considered as distribution service—a service for linking the resources (computing functions, data store, processing functions, networks, servers and applications) and for provision of coordinating between the resources.
- Figure 6.2 shows four cloud service models and examples. Cloud computing can be considered by a simple equation: Cloud Computing = SaaS + Paas + IaaS + DaaS … 6.2

- SaaS means Software as a Service. The software is made available to an application or service on demand. SaaS is a service model where the applications or services deploy and host at the cloud, and are made available through the Internet on demand by the service user. The software control, maintenance, updation to new version and infrastructure, platform and resource requirements are the responsibilities of the cloud service provider.
- PaaS means Platform as a Service. The platform is made available to a developer of an application on demand. PaaS is a service model where the applications and services develop and execute using the platform (for computing, data store and distribution services) which is made available through the Internet on demand for the developer of the applications. The platform, network, resources, maintenance, updation and security as per the developers' requirements are the responsibilities of the cloud service provider.
- IaaS means Infrastructure as a Service. The infrastructure (data stores, servers, data centres and network) is made available to a user or developer of application on demand. Developer

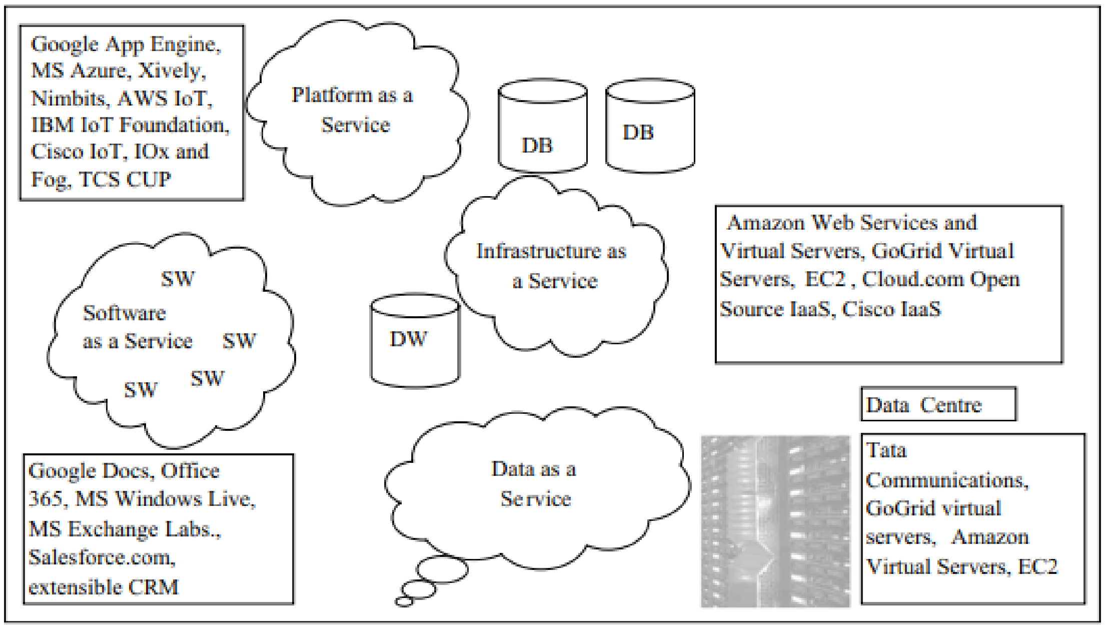

Google App Engine, MS Azure, Xively, Nimbits, AWS IoT, IBM IoT Foundation, Cisco IoT, IOx and Fog, TCS CUP
Platform as a Service
DB
DB
Amazon Web Services and Virtual Servers, GoGrid Virtual Servers, EC2 , Cloud.com Open Source IaaS, Cisco IaaS
Infrastructure as a Service
SW
Software as a Service
SW
SW
SW
DW
Data Centre
Tata Communications, GoGrid virtual servers, Amazon Virtual Servers, EC2
Google Docs, Office 365, MS Windows Live, MS Exchange Labs., Salesforce.com, extensible CRM
Data as a Service

Figure 6.2 PaaS, SaaS, IaaS and DaaS Cloud Service model
[DW—Data Warehouses; DB—Databases; EC2—Elastic Computing Cloud; SW—Software; MS—Microsoft; CRM—Customer Customer Relations Management]

installs the OS image, data store and application and controls them at the infrastructure. IaaS is a service model where the applications develop or use the infrastructure which is made available through the Internet on demand on rent (pay as per use in multi-tenancy model) by a developer or user. IaaS computing systems, network and security are the responsibilities of the cloud service provider. DaaS means Data as a Service

- Data at a data centre is made available to a user or developer of application on demand. DaaS is a service model where the data store or data warehouse is made available through the Internet on demand on rent (pay as per use in multi tenancy model) to an enterprise. The data centre management, 24×7 power, control, network, maintenance, scale up, data replicating and mirror nodes and systems as well as physical security are the responsibilities of the data centre service provider.

- Data at a data centre is made available to a user or developer of application on demand. DaaS is a service model where the data store or data warehouse is made available through the Internet on demand on rent (pay as per use in multi tenancy model) to an enterprise. The data centre management, 24×7 power, control, network, maintenance, scale up, data replicating and mirror nodes and systems as well as physical security are the responsibilities of the data centre service provider.