PIE Tech

POLLACHI INSTITUTE OF ENGINEERING AND TECHNOLOGY

(Approved by AICTE and Affiliated to Anna University)

sky is the limit

Department of Electronics and Communication Engineering

Regulation 2021

III Year – VI Semester

ET3491- Embedded Systems and IOT Design

UNIT – I 8 BIT EMBEDDED PROCESSOR

Microcontrollers for an Embedded System – 8051 – Architecture – Addressing Modes – Instruction Set – Program and Data Memory – Stacks – Interrupts – Timers/Counters – Serial Ports – Programming.

Microcontrollers for Embedded system:

- ✓ A Microcontroller is a single chip computer.
- ✓ A CPU with all the peripherals like RAM, ROM, I/O Ports, Timers, and ADCs etc. on the same chip.

  For Ex: Motorola 6811, Intel 8051, Zilog Z8 and PIC 16X etc…

Microprocessor:

- ✓ A CPU built into a single VLSI chip is called a microprocessor.
- ✓ It is a general-purpose device and additional external circuitry is added to make it a microcomputer.
- ✓ The microprocessor contains arithmetic logic unit (ALU), Control unit, Instruction register, Program counter (PC), clock circuit (internal or external), reset circuit (internal or external) and registers.
- ✓ But the microprocessor has no on chip I/O Ports, Timers, Memory etc.
- ✓ For example, Intel 8085 is an 8-bit microprocessor and Intel 8086/8088 a 16-bit microprocessor.
- ✓ The block diagram of the Microprocessor is shown in Fig.1

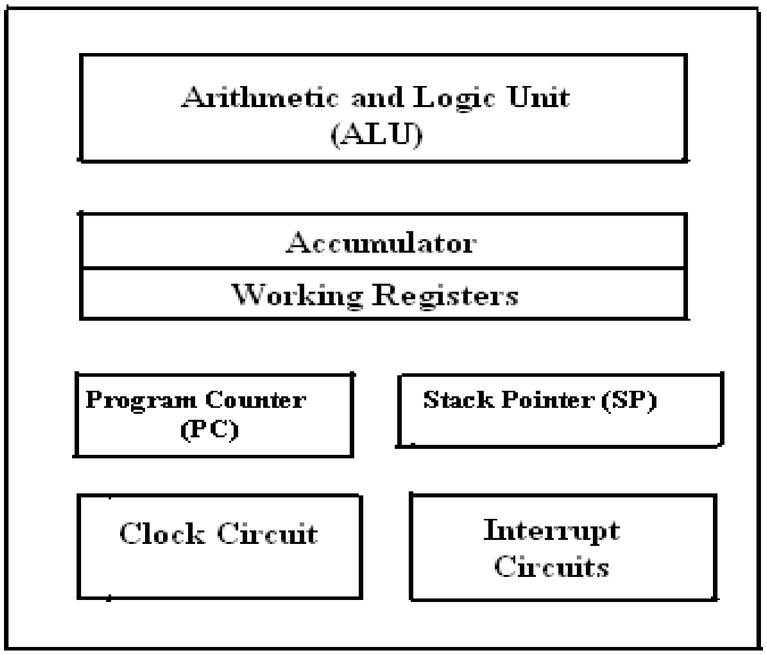


Fig.1: Block diagram of a Microprocessor.

MICROCONTROLLER :

- ✓ A microcontroller is an integrated single chip, which consists of CPU, RAM, EPROM/PROM/ROM, I/O ports, timers, interrupt controller.
- ✓ For example, Intel 8051 is 8-bit microcontroller and Intel 8096 is 16-bit microcontroller.
- ✓ The block diagram of Microcontroller is shown in Fig.2.

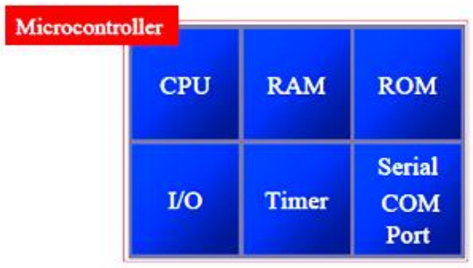


Fig.2.Block Diagram of a Microcontroller

Distinguish between Microprocessor and Microcontroller

S.No	**Microprocessor**	**Microcontroller**
1	A microprocessor is a general purpose device.	A microcontroller is a dedicated chip which is also called as single chip computer.
2	A microprocessor does not contain on chip I/O Ports, Timers, Memories etc.	A microcontroller includes RAM, ROM, serial and parallel interface, timers, interrupt circuitry in a single chip.
3	Microprocessor is used as the CPU in microcomputer system.	Microcontroller is used to perform control-oriented applications.
4	Microprocessor instructions are nibble or byte addressable	Microcontroller instructions are both bit addressable as well as byte addressable.
5	Microprocessor based system design is complex and expensive	Microcontroller based system design is simple and cost effective
6	The Instruction set of microprocessor is complex with large number of instructions.	The instruction sets are simple with less number of instructions.

****

4.2: INTEL 8051 MICRCONTROLLER:

Draw the architectural block diagram of 8051 microcontroller and explain. (NOV 2011, MAY 2010, NOV 2009, NOV2008, May 2008, MAY 2007, MAY 2006, NOV 2016, May 2016)

Features of 8051 Microcontroller:

The 8051 is an 8-bit Controller:

- ✓ The CPU can works on only 8 bits of data at a time
- ✓ The 8051 has
  - 128 bytes of RAM
  - 4K bytes of on-chip ROM
  - Two timers
  - One serial port
  - Four I/O ports, each 8 bits wide
  - 6 interrupt sources

ARCHITECTURE & BLOCK DIAGRAM OF 8051 MICROCONTROLLER:

- ✓ It has hardware architecture with RISC (Reduced Instruction Set Computer) concept.
- ✓ The block diagram of 8051 microcontroller is shown in Fig 3.
- ✓ 8051 has 8-bit ALU.
- ✓ ALU can perform all the 8-bit arithmetic and logical operations in one machine cycle.
- ✓ The ALU is associated with two registers A & B

A and B Registers:

- ✓ The A and B registers are special function registers.
- ✓ A & B registers hold the results of many arithmetic and logical operations of 8051.
- ✓ The A register is also called the **Accumulator.**
- ✓ A register is used as a general register to accumulate the results of a large number of instructions.
- ✓ By default, it is used for all mathematical operations and data transfer operations between CPU and external memory.
- ✓ The B register is mainly used for multiplication and division operations along with A register.
  - ▪ Ex: MUL AB : DIV AB.
- ✓ It has no other function other than as a store data.

R registers:

- ✓ "R" registers are a set of eight registers that are named R0, R1, etc. up to R7.
- ✓ These registers are used as auxiliary registers in many operations.
- ✓ The "R" registers are also used to temporarily store values.

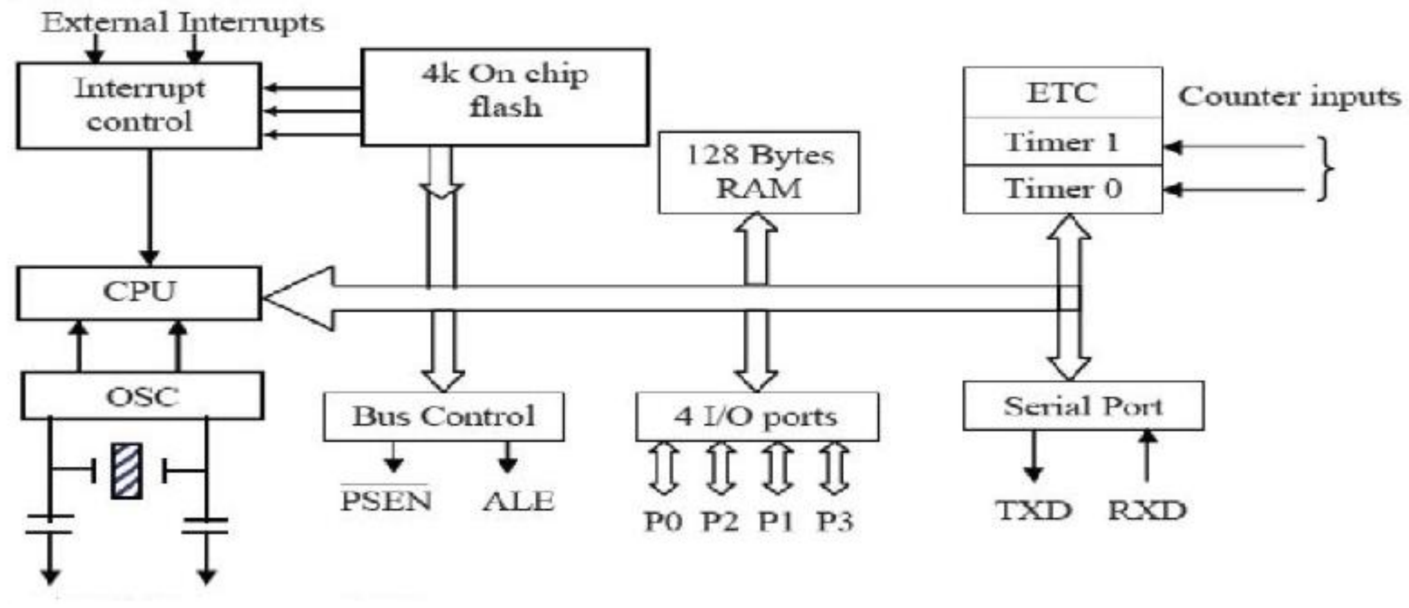


Fig.3. Block Diagram of 8051 Microcontroller

Program Counter (PC) :

- ✓ 8051 has a 16-bit program counter.
- ✓ The program counter holds address of the next instruction to be executed.
- ✓ After execution of one instruction, the program counter is incremented.

Data Pointer Register (DPTR):

- ✓ It is a 16-bit register which is the only user-accessible.
- ✓ DPTR is used to point the data. 8051 will access external memory at the address indicated by DPTR.
- ✓ DPTR can also be used as two 8-registers DPH and DPL.

Stack Pointer Register (SP) :

- ✓ It is an 8-bit register which stores the address of the stack top.
- ✓ When a value is pushed onto the stack, the 8051 first increments the value of SP and then stores the value.
- ✓ Similarly when a value is popped off the stack, the 8051 returns the value from the memory location indicated by SP and then decrements the value of SP.
- ✓ Since the SP is only 8-bit wide.
- ✓ It is incremented or decremented by two.
- ✓ SP is modified directly by the 8051 by six instructions: PUSH, POP, ACALL, LCALL, RET, and RETI.
- ✓ It is also used intrinsically whenever an interrupt is triggered.

Block Diagram

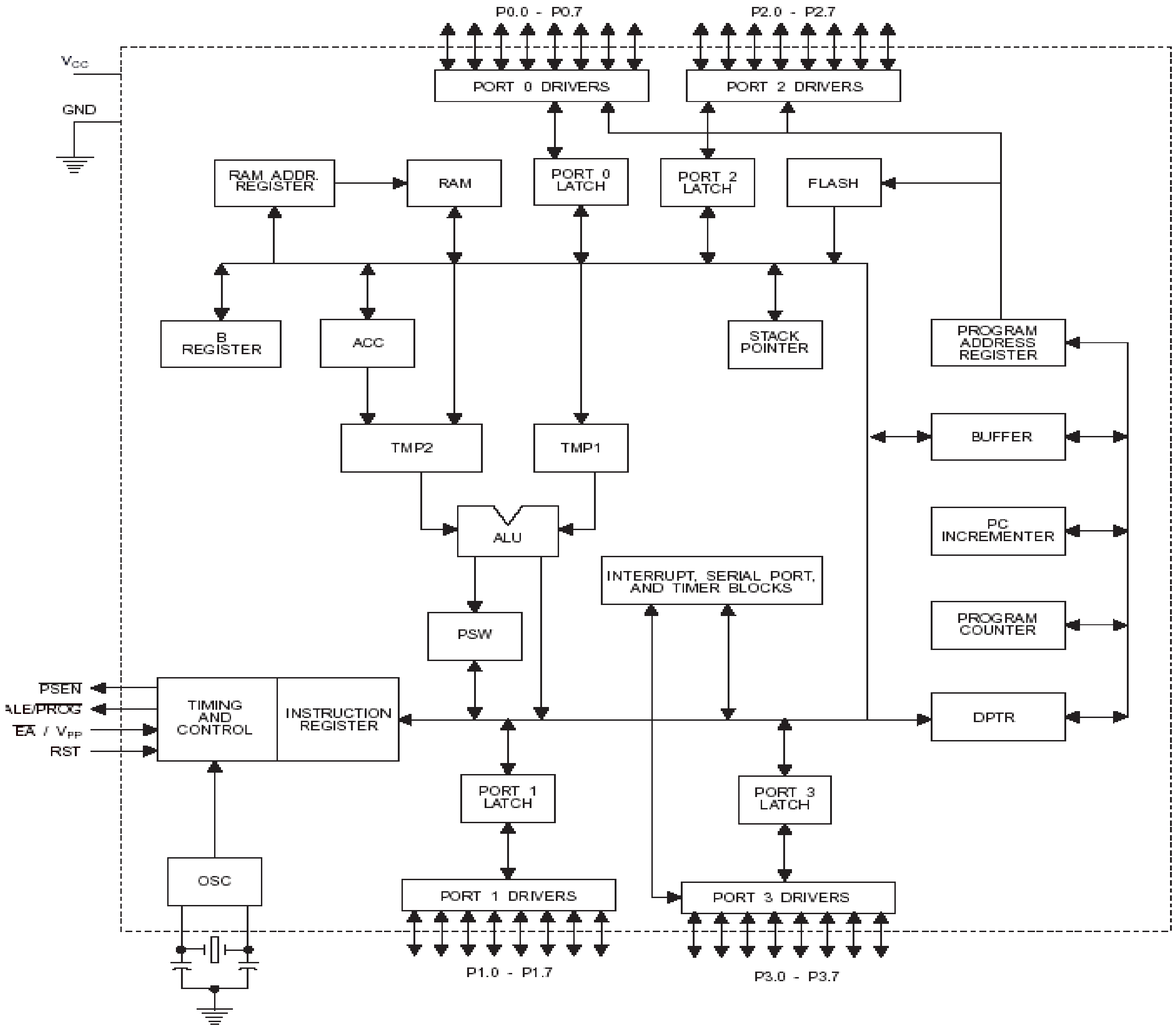
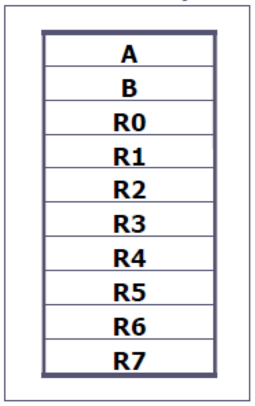


Fig 3a: Internal architecture diagram of 8051 Microcontroller



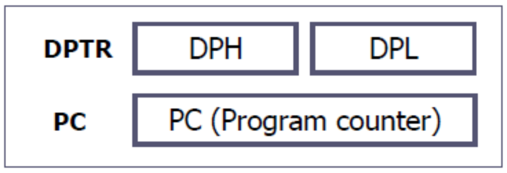
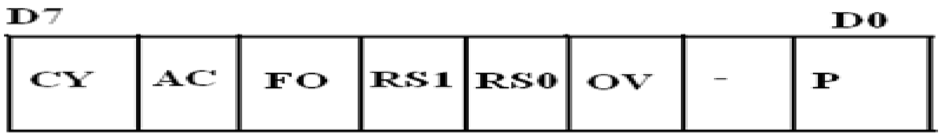


Fig: Structure of registers

Program Status Register (PSW):

Give PSW of 8051 and describe the use of each bit in PSW. (NOV 2015)

- ✓ The 8051 has an 8-bit PSW register which is also known as Flag register.
- ✓ In the 8-bit register only 6-bits are used by 8051.The two unused bits are user definable bits.
- ✓ In the 6-bits, four of them are conditional flags. They are Carry –CY, Auxiliary Carry-AC, Parity-P, and Overflow-OV.
- ✓ These flag bits indicate some conditions of result after an instruction was executed.



- ✓ The bits PSW3 and PSW4 are denoted as RS0 and RS1.
- ✓ These bits are used to select the bank registers of the RAM location.
- ✓ The meaning of various bits of PSW register is shown below.

CY	PSW.7	Carry Flag
AC	PSW.6	Auxiliary Carry Flag
FO	PSW.5	Flag 0 available for general purpose
RS1	PSW.4	Register Bank select bit 1
RS0	PSW.3	Register bank select bit 0
OV	PSW.2	Overflow flag
---	PSW.1	User definable flag
P	PSW.0	Parity flag .set/cleared by hardware.

- ✓ The selection of the register Banks and their addresses are given below.

RS1	**RS0**	**Register Bank**	**Address**
0	0	0	00H-07H
0	1	1	08H-0FH
1	0	2	10H-17H
1	1	3	18H-1FH

RAM & ROM:

- ✓ The 8051 microcontroller has 128 bytes of Internal RAM and 4KB of on chip ROM.
- ✓ The RAM is also known as Data memory and the ROM is known as program (Code) memory.
- ✓ Code memory holds program that is to be executed.
- ✓ Program Address Register holds address of the ROM/ Flash memory.
- ✓ Data Address Register holds address of the RAM.

I/O ports:

- ✓ The 8051 microcontroller has 4 parallel I/O ports, each of 8-bits.
- ✓ So, it provides 32 I/O lines for connecting the microcontroller to the peripherals.
- ✓ The four ports are P0 (Port 0), P1 (Port1), P2 (Port 2) and P3 (Port3).

ADDRESSING MODES OF 8051 :

Explain different types addressing modes of 8051 microcontroller. (NOV 2008, NOV 2015, April 2017)

- ✓ The way in which the data operands are specified is known as the addressing modes. There are various methods of denoting the data operands in the instruction.
- ✓ The 8051 microcontroller supports 5 addressing modes. They are
  1. Immediate addressing mode
  2. Direct Addressing mode
  3. Register addressing mode
  4. Register indirect addressing mode
  5. Indexed addressing mode

Immediate addressing mode:

- ✓ The addressing mode in which the data operand is a constant and it is a part of the instruction itself is known as Immediate addressing mode.
- ✓ Normally the data must be preceded by a # sign.
- ✓ This addressing mode can be used to transfer the data into any of the registers including DPTR.

Examples:

▪ MOV A, # 27 H	: The data (constant) 27 is moved to the accumulator register
▪ ADD R1, #45 H	: Add the constant 45 to the contents of the accumulator
▪ MOV DPTR, # 8245H	: Move the data 8245 into the data pointer register.

Direct addressing mode:

- ✓ In the addressing mode, the data operand is in the RAM location (00 -7FH) and the address of the data operand is given in the instruction.
- ✓ The direct addressing mode uses the lower 128 bytes of Internal RAM and the SFRs

Examples:

- MOV R1, 42H : Move the contents of RAM location 42 into R1 register
- MOV 49H, A : Move the contents of the accumulator into the RAM location 49.
- ADD A, 56H : Add the contents of the RAM location 56 to the accumulator

Register addressing mode:

- ✓ In the addressing mode, the data operands are available in the registers.

Examples:

- MOV A,R0 : Move the contents of the register R0 to the accumulator
- MOV P1, R2 :Move the contents of the R2 register into port 1
- MOV R5, R2 : This is invalid. The data transfer between the registers is not allowed.

Register Indirect addressing mode:

- ✓ In the addressing mode, a register is used as a pointer to the data memory block.

Examples:

- MOV A,@ R0 :Move the contents of RAM location whose address is in R0 into **A** (accumulator)
- MOV @ R1 , B : Move the contents of B into RAM location whose address is held by R1
- When R0 and R1 are used as pointers, they must be preceded by @ sign
- ✓ **Advantage: It makes accessing the data more dynamic than static as in the case of direct addressing mode.**

Indexed addressing mode:

- ✓ This addressing mode is used in accessing the data elements of lookup table entries, located in program ROM.

Example: MOVC A, @ A+DPTR

- The 16-bit register DPTR and register A are used to form the address of the data element stored in on-chip ROM.

<u>INSTRUCTIONS SET OF 8051:</u>

Discuss in detail the 8051 instruction set. (NOV 2008)

Arithmetic instructions:

With example, explain arithmetic instructions in 8051 microcontroller. (NOV 2012)

- ✓ ADD
  - 8-bit addition between the accumulator (A) and a second operand.
  - The result is always in the accumulator.
  - The CY flag is set/reset appropriately.
- ✓ ADDC
  - 8-bit addition between the accumulator, a second operand and the previous value of the CY flag.
  - Useful for 16-bit addition in two steps.
  - The CY flag is set/reset appropriately.
- ✓ DAA
  - Decimal adjust the accumulator.
  - Format the accumulator into a proper 2 digit packed BCD number.
  - Operates only on the accumulator.
  - Works only after the ADD instruction.
- ✓ SUBB
  - Subtract with Borrow.
  - Subtract an operand and the previous value of a borrow (carry) flag from the accumulator.
  - A ← A - <operand> - CY.
  - The result is always saved in the accumulator.
  - The CY flag is set/reset appropriately.
- ✓ INC
  - Increment the operand by one.
  - The operand can be a register, a direct address, an indirect address, the data pointer.
- ✓ DEC
  - Decrement the operand by one.
  - The operand can be a register, a direct address, an indirect address.
- ✓ MUL AB / DIV AB
  - Multiply A by B and place result in A and B registers.
  - Divide A by B and place quotient in A register & remainder in B register.
  -

Logical instructions in 8051.

- ✓ ANL : It performs AND logical operation between two operands.
  - ➢ Work on byte sized operands or the CY flag.
    - ANL A, Rn
    - ANL A, direct
    - ANL A, @Ri
    - ANL A, #data
    - ANL direct, A
    - ANL direct, #data
    - ANL C, bit
    - ANL C, /bit
- ✓ ORL: It performs OR logical operation between two operands.
  - ➢ Work on byte sized operands or the CY flag.
    - ORL A, Rn
    - ORL A, direct
    - ORL A, @Ri
    - ORL A, #data
- ✓ XRL
  - ➢ Works on bytes only.
    - XRL A, Rn
    - XRL A, direct
- ✓ CPL / CLR
  - ➢ Complement / Clear.
  - ➢ Work on the accumulator or a bit.
    - CLR P1.2
    - CPL Rn
- ✓ RL / RLC / RR / RRC
  - ➢ Rotate the accumulator.
    - RL and RR without the carry
    - RLC and RRC rotate through the carry.
    - SWAP A: Swap the upper and lower nibbles of the accumulator.

Data transfer instructions in 8051.

Briefly explain the data transfer instructions available in 8051 microcontroller. (NOV 2014)

MOV

- ➢ 8-bit data transfer for internal RAM and the SFR.
    - MOV A, Rn
    - MOV A, direct
    - MOV A, @Ri
    - MOV A, #data
    - MOV Rn, A
    - MOV Rn, direct
    - MOV Rn, #data
    - MOV direct, A
    - MOV direct, Rn
    - MOV direct, direct
    - MOV direct, @Ri
    - MOV direct, #data
    - MOV @Ri, A
    - MOV @Ri, direct
    - MOV @Ri, #data

- ✓ MOV
    - ➢ 1-bit data transfer involving the CY flag
        - MOV C, bit
        - MOV bit, C
- ✓ MOV
    - ➢ 16-bit data transfer involving the DPTR
        - MOV DPTR, #data
- ✓ MOVC
    - ➢ Move Code Byte
        - Load the accumulator with a byte from program memory.
        - Must use indexed addressing
        - MOVC A, @A+DPTR
        - MOVC A, @A+PC

- ✓ MOVX
  - ➢ Data transfer between the accumulator and a byte from external data memory.
    - MOVX A, @Ri
    - MOVX A, @DPTR
    - MOVX @Ri, A
    - MOVX @DPTR, A
- ✓ PUSH / POP
  - ➢ Push and Pop a data byte onto the stack.
  - ➢ The data byte is identified by a direct address from the internal RAM locations.
    - PUSH DPL
    - POP 40H
- ✓ XCH
  - ➢ Exchange accumulator and a byte operand
    - XCH A, Rn
    - XCH A, direct
    - XCH A, @Ri
- ✓ XCHD
  - ➢ Exchange lower digit of accumulator with the lower digit of the memory location specified.
    - XCHD A, @Ri
    - The lower 4-bits of the accumulator are exchanged with the lower 4-bits of the internal memory location identified indirectly by the index register.
    - The upper 4-bits of each are not modified.

Boolean (or) Bit manipulation instructions in 8051.

- ✓ This group of instructions is associated with the single-bit operations of the 8051.
- ✓ This group allows manipulating the individual bits of bit addressable registers and memory locations as well as the CY flag.
    - The P, OV, and AC flags cannot be directly altered.
- ✓ This group includes:
    - Set, clear, and, or complement, move.
    - Conditional jumps.
- ✓ CLR

- Clear a bit or the CY flag.
- CLR P1.1
- CLR C

✓ SETB

- Set a bit or the CY flag.
- SETB A.2
- SETB C

✓ CPL

- Complement a bit or the CY flag.
- CPL 40H; Complement bit 40 of the bit addressable memory

✓ ORL / ANL

- OR / AND a bit with the CY flag.
- ORL C, 20H; OR bit 20 of bit addressable memory with the CY flag
- ANL C, 34H; AND bit 34 of bit addressable memory with the CY flag.

✓ MOV

- Data transfer between a bit and the CY flag.
- MOV C, 3FH; Copy the CY flag to bit 3F of the bit addressable memory.
- MOV P1.2, C; Copy the CY flag to bit 2 of P1.

✓ JC / JNC

- Jump to a relative address if CY is set / cleared.

✓ JB / JNB

- Jump to a relative address if a bit is set / cleared.
- JB ACC.2, <label>

✓ JBC - Jump to a relative address, if a bit is set and clear the bit.

Branching instructions:

With example, explain branching instructions in 8051 microcontroller. (May 2010, NOV 2012)

Explain the working of program control transfer instructions of 8051. (May 2012)

✓ The 8051 provides four different types of unconditional jump instructions:

➢ Short Jump – SJMP

- Uses an 8-bit signed offset relative to the 1st byte of the next instruction.
- Long Jump – LJMP
- Uses a 16-bit address.

    - 3 byte instruction capable of referencing any location in the entire 64K of program memory.
  - ➢ Absolute Jump – AJMP
    - Uses an 11-bit address.
    - 2 byte instruction
    - The 11-bit address is substituted for the lower 11-bits of the PC to calculate the 16-bit address of the target.
    - The location referenced must be within the 2K Byte memory.
  - ➢ Indirect Jump – JMP
    - JMP @A + DPTR
- ✓ The 8051 provides 2 forms for the CALL instruction:
  - ➢ Absolute Call – ACALL
    - Uses an 11-bit address similar to AJMP
    - The subroutine must be within the same 2K page.
  - ➢ Long Call – LCALL
    - Uses a 16-bit address similar to LJMP
    - The subroutine can be anywhere.
  - ➢ Both forms push the 16-bit address of the next instruction on the stack and update the stack pointer.
- ✓ The 8051 provides 2 forms for the return instruction:
  - ➢ Return from subroutine – RET
    - Pop the return address from the stack and continue execution there.
  - ➢ Return from Interrupt Service Routine – RETI
    - Pop the return address from the stack.
    - Continue execution at the address retrieved from the stack.
    - The PSW is not automatically restored.
- ✓ The 8051 supports 5 different conditional jump instructions.
  - ➢ ALL conditional jump instructions use an 8-bit signed offset.
  - ➢ Jump on Zero – JZ / JNZ
    - Jump if the A == 0 / A != 0
      - The check is done at the time of the instruction execution.

- ➢ Jump on Carry – JC / JNC
    - Jump if the C flag is set / cleared.
- ➢ Jump on Bit – JB / JNB
    - Jump if the specified bit is set / cleared.
    - Any addressable bit can be specified.
- ➢ Jump if the Bit is set then Clear the bit – JBC
    - Jump if the specified bit is set.
    - Then clear the bit.

✓ Compare and Jump if Not Equal – CJNE

- ➢ Compare the magnitude of the two operands and jump if they are not equal.
    - The values are considered to be unsigned.
    - The Carry flag is set / cleared appropriately.
    - CJNE A, direct, rel
    - CJNE Rn, #data, rel
    - CJNE @Ri, #data, rel

✓ Decrement and Jump if Not Zero – DJNZ

- ➢ Decrement the first operand by 1 and jump to the location identified by the second operand if the resulting value is not zero.
    - DJNZ Rn, rel
    - DJNZ direct, rel
- ➢ NOP – No operation

<u>Memory organization :</u>

Explain in detail the internal memory organization of 8051 microcontroller (NOV 2014, May 2012, NOV 2011, NOV 2010, May 2010, MAY 2009, NOV 2008, NOV 2007)

- ✓ The 8051 microcontroller has 128 bytes of Internal RAM and 4kB of on chip ROM.
- ✓ The RAM is also known as Data memory and the ROM is known as program (Code) memory.
- ✓ Code memory holds the actual 8051 program to be executed.
- ✓ In 8051, memory is limited to 64KB.
- ✓ Code memory may be found on-chip, as ROM or EPROM.
- ✓ It may also be stored completely off-chip in an external ROM / EPROM.
- ✓ The 8051 has only 128 bytes of Internal RAM but it supports 64KB of external RAM.

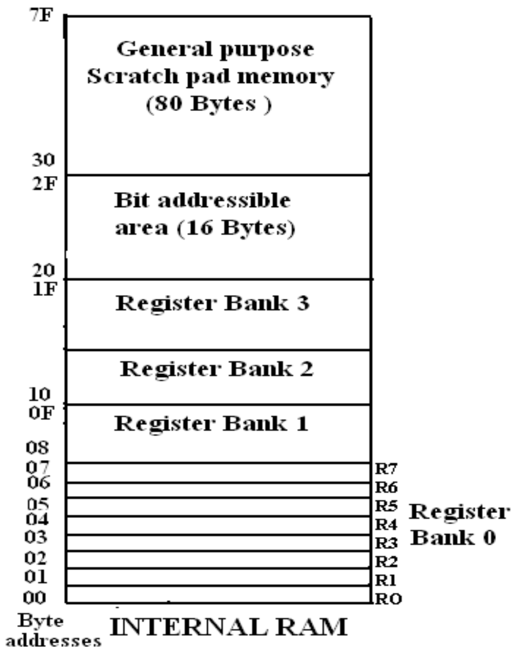
- ✓ Since the memory is off-chip, it is not as flexible for accessing and is also slower.

Structure of Internal RAM OF 8051(Data Memory) :

Explain the Data memory structure of 8051. (NOV 2011)

- ✓ Internal RAM is found on-chip on the 8051. So it is the fastest RAM available.
- ✓ It is flexible in terms of reading, writing and modifying its contents.
- ✓ Internal RAM is volatile.
- ✓ When the 8051 is reset, internal RAM is cleared.
- ✓ The 128 bytes of internal RAM is organized as below.
- ✓ Four register banks (Bank0, Bank1, Bank2 and Bank3) each of 8-bits (total 32 bytes).
- ✓ The default bank register is Bank0.
- ✓ The remaining Banks are selected with the help of RS0 and RS1 bits of PSW Register.
- ✓ 16 bytes of bit addressable area and
- ✓ 80 bytes of general purpose area (Scratch pad memory) of internal RAM as shown in the diagram below.
- ✓ This area is utilized by the microcontroller as a storage area for the operating stack.
- ✓ The 32 bytes of RAM from address 00 H to 1FH are used as working registers organized as four banks of eight registers each.
- ✓ The registers are named as R0-R7.
- ✓ Each register can be addressed by its name or by its RAM address.

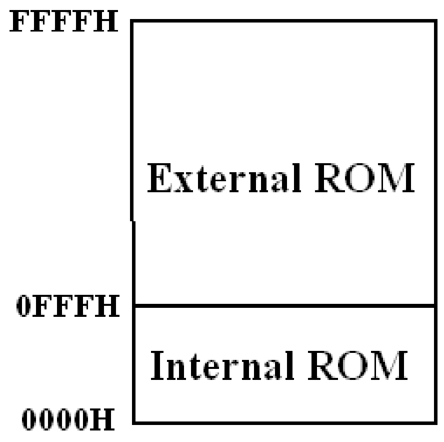
For example: MOV A, R7 or MOV R7,#05H



INTERNAL RAM

Structure of Internal ROM (On –chip ROM / Program Memory / Code Memory):

- ✓ The 8051 microcontroller has 4KB of on chip ROM, but it can be extended up to 64KB.
- ✓ This ROM is also called program memory or code memory.
- ✓ The CODE segment is accessed using the program counter (PC) for opcode fetches and by DPTR for data.
- ✓ The external ROM is accessed when the EA pin is connected to ground or the contents of program counter exceeds 0FFFH.
- ✓ When the Internal ROM address is exceeded the 8051 automatically fetches the code bytes from the external program memory.



SPECIAL FUNCTION REGISTERS (SFRs)

Write the available special function registers in 8051. Explain each register with its format and functions. (April 2017, NOV 2015)

- ✓ In 8051 microcontroller, there are registers which uses the RAM addresses from 80h to FFh.
- ✓ They are used for certain specific operations. These registers are called Special Function Registers (SFRs).
- ✓ Most of SFRs are bit addressable and other few registers are byte addressable.
- ✓ In these SFRs, some of them are related to I/O ports (P0, P1, P2 and P3) and some of them are for control operations (TCON, SCON & PCON).
- ✓ Remaining are the auxiliary SFRs, that they don't directly configure the 8051.
- ✓ The list of SFRs and their functional names are given below.

- ✓ **The * indicates the bit addressable SFRs**

S.No	Symbol		Name of SFR	Address (Hex)
1	ACC*		Accumulator	**0E0**
2	B*		B-Register	**0F0**
3	PSW*		Program Status word register	**0DO**
4	SP		Stack Pointer Register	**81**
5	DPTR	DPL	Data pointer low byte	**82**
		DPH	Data pointer high byte	**83**
6	P0*		Port 0	**80**
	P1*		Port 1	**90**
8	P2*		Port 2	**0A**
9	P3*		Port 3	**0B**
10	IP*		Interrupt Priority control	**0B8**
11	IE*		Interrupt Enable control	**0A8**
12	TMOD		Timer mode register	**89**
13	TCON*		Timer control register	**88**
14	TH0		Timer 0 Higher byte	**8C**
15	TL0		Timer 0 Lower byte	**8A**
16	TH1		Timer 1Higher byte	**8D**
17	TL1		Timer 1 lower byte	**8B**
18	SCON*		Serial control register	**98**
19	SBUF		Serial buffer register	**99**
20	PCON		Power control register	**87**

Table: Special Function Registers

STACK in 8051 Microcontroller:

- ✓ The stack is a part of RAM used by the CPU to store information temporarily.
- ✓ This information may be either data or an address.

- ✓ The register used to access the stack is called the Stack pointer (SP).
- ✓ SP is an 8-bit register. So, it can take values of 00 to FF H.
- ✓ When the 8051 is powered up, the SP register contains the value 07.i.e the RAM location value 08 is the first location being used for the stack by the 8051 controller.
- ✓ There are two important instructions to handle stack. One is the PUSH and the other is the POP.
- ✓ The loading of data from CPU registers to the stack is done by PUSH.
- ✓ The loading of the contents of the stack back into a CPU register is done by POP.

Interrupts:

Explain interrupt structure of 8051 microcontroller. (NOV 2011, MAY 2009)

Interrupt Structure:

- ✓ An interrupt is an external or internal event that disturbs the microcontroller to inform it that a device needs its service.
- ✓ The program which is associated with the interrupt is called the **interrupt service routine** (ISR) or **interrupt handler**.
- ✓ Upon receiving the interrupt signal, the microcontroller finishes current operation and saves the PC on stack.
- ✓ Jumps to a fixed location in memory depending on type of interrupt.
- ✓ Starts to execute the interrupt service routine until RETI.
- ✓ Upon executing the RETI the microcontroller returns to the place where it was interrupted. Get pop PC from stack.
- ✓ The 8051 microcontroller has **FIVE** interrupts in addition to Reset. They are
    - Timer 0 overflow Interrupt
    - Timer 1 overflow Interrupt
    - External Interrupt 0(INT0)
    - External Interrupt 1(INT1)
    - Serial Port Interrupt
- ✓ Each interrupt has a specific place in code memory where program execution begins.

• External Interrupt 0:	0003 H
• Timer 0 overflow:	000B H

- External Interrupt 1 :   0013 H
- Timer 1 overflow: 001B H
- Serial Interrupt :   0023 H

✓ Upon reset all Interrupts are disabled & do not respond to the Microcontroller.

✓ These interrupts must be enabled by software. This is done by an 8-bit register called Interrupt Enable Register (IE).

Interrupt Enable Register :

EA	——	ET2	ES	ET1	EX1	ET0	EX0

- EA : Global enable/disable. To enable the interrupts, this bit must be set high.
- --- : Undefined-reserved for future use.
- ET2 : Enable /disable Timer 2 overflow interrupt.
- ES : Enable/disable Serial port interrupts.
- ET1 : Enable /disable Timer 1 overflow interrupt.
- EX1 : Enable/disable External interrupt1.
- ET0 : Enable /disable Timer 0 overflow interrupt.
- EX0 : Enable/disable External interrupt0

✓ Upon reset, the interrupts have the following priority from top to down. The interrupt with the highest PRIORITY gets serviced first.

1. External interrupt 0 (INT0)
2. Timer interrupt0 (TF0)
3. External interrupt 1 (INT1)
4. Timer interrupt1 (TF1)
5. Serial communication (RI+TI)

✓ Priority can also be set to “high” or “low” by 8-bit IP register.

Interrupt priority register:

—	—	PT2	PS	PT1	PX1	PT0	PX0

- IP.7: reserved
- IP.6: reserved
- IP.5: Timer 2 interrupt priority bit (8052 only)
- IP.4: Serial port interrupt priority bit
- IP.3: Timer 1 interrupt priority bit
- IP.2: External interrupt 1 priority bit
- IP.1: Timer 0 interrupt priority bit
- IP.0: External interrupt 0 priority bit

PROGRAMMING TIMERS OF 8051

1. **Explain the different modes of operation of timers in 8051 in detail with its associated registers. Describe different modes of operation of timers /counters in 8051 with its associated registers. (NOV 2009, MAY 2009. May 2007, May 2016)**
   Draw and explain the functions of TCON and TMOD registers of 8051. (Dec 2008)
   Explain the on-chip timer modes of an 8051 Microcontroller. (April 2010, NOV 2016)

Timer Registers.

✓ The 8051 has two timers/counters, they can be used either as timers (used to generate a time delay) or as event counters.

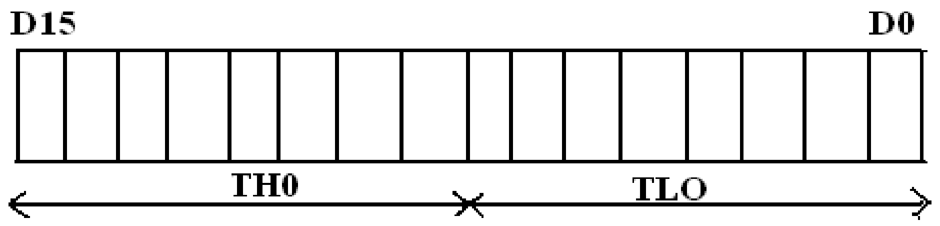
TIMER 0:

✓ Timer 0 is a 16-bit register and can be treated as two 8-bit registers (TL0 & TH0).

✓ These registers can be accessed similar to any other registers like A, B or R1 etc

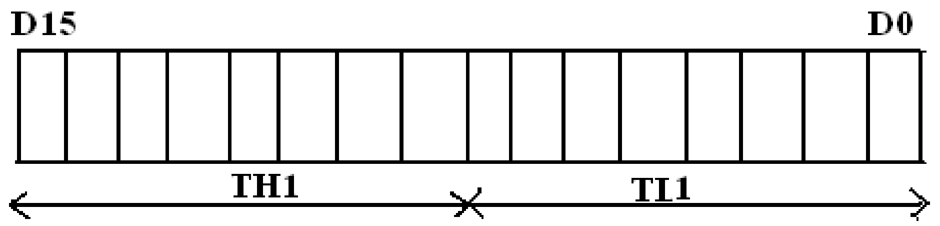
✓ Ex : The instruction MOV TL0,#07 moves the value 07 into lower byte of Timer0.

✓ Similarly MOV R1, TH0 saves the contents of TH0 in the R1 register.



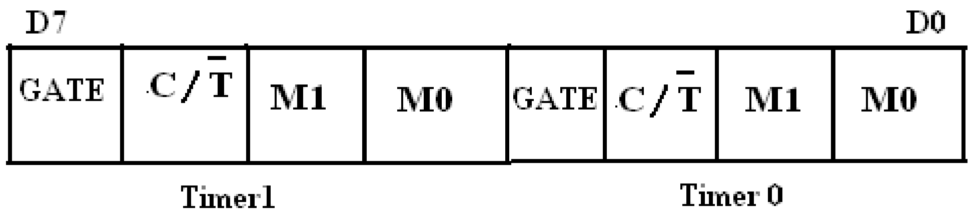
TIMER 1:

- ✓ Timer 1 is also a 16-bit register and can be treated as two 8-bit registers (TL1 & TH1).
- ✓ These registers can be accessed similar to any other registers like A, B or R1etc
- ✓ Ex : The instruction MOV TL1,#05 moves the value 05 into lower byte of Timer1.
- ✓ Similarly MOV R0,TH1 saves the contents of TH1 in the R0 register.



TMOD (Timer mode Register):

- ✓ The various operating modes of both the timers T0 and T1 are set by a TMOD register.
- ✓ TMOD is a 8-bit register.
- ✓ The lower 4 bits are for Timer 0
- ✓ The upper 4 bits are for Timer 1
- ✓ In each case,
  - The lower 2 bits are used to set the timer mode
  - The upper 2 bits to specify the operation



GATE:

- ✓ This bit is used to start or stop the timers by hardware.
- ✓ When GATE= 1, the timers can be started / stopped by the external sources.
- ✓ When GATE= 0, the timers can be started or stopped by software instructions like SETB TR_X or CLR TR_X.

C/T (Counter/Timer):

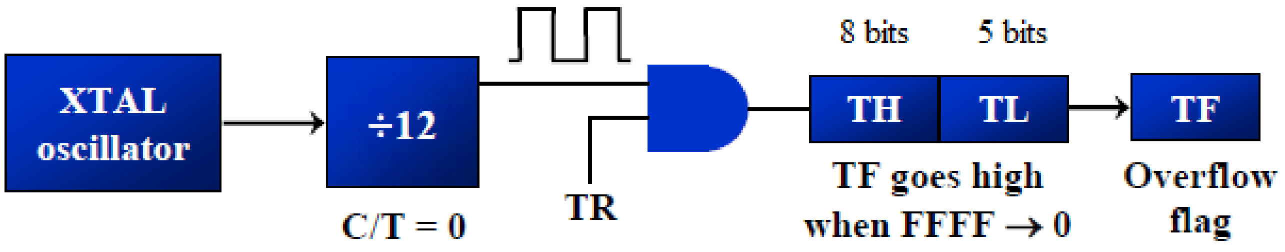
- ✓ This bit decides whether the timer is used as delay generator or event counter.
- ✓ When $C/\overline{T}$ **= 0,** timer is used as delay generator.
- ✓ When $C/\overline{T}$ **=1,** timer is used as an event counter.
- ✓ The clock source for the time delay is the crystal frequency of 8051.
- ✓ The clock source for the event counter is the external clock source.

M1, M0 (Mode):

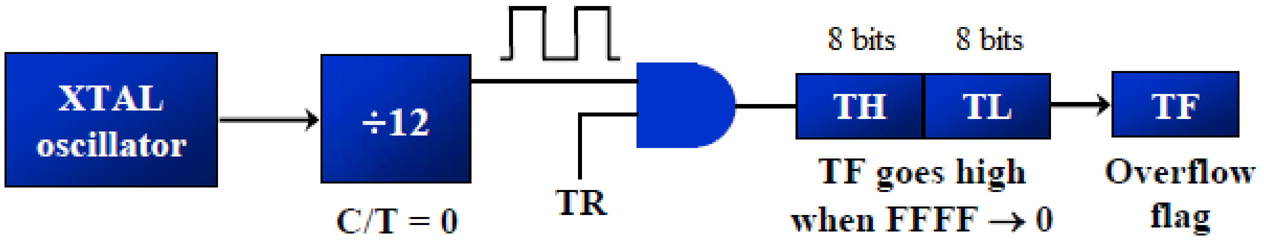
- ✓ These two bits are the timer mode bits.
- ✓ The timers of the 8051 can be configured in four modes Mode0, Mode1, Mode2 & Mode 3.
- ✓ The selection and operation of the modes is shown below.

S.No	M0	M1	Mode	Operation
1	0	0	Mode 0	**13-bit Timer mode.** 8-bit Timer/counter THx with TLx as 5-bit prescaler
2	0	1	Mode 1	**16-bit Timer mode**.16-bit timer /counter THx and TLx are cascaded. There is no presacler
3	1	0	Mode 2	**8-bit auto reload.** 8-bit auto reload timer/counter. THx holds a value which is to be reloaded TLx each time it overflows
4	1	1	Mode 3	**Split timer mode**

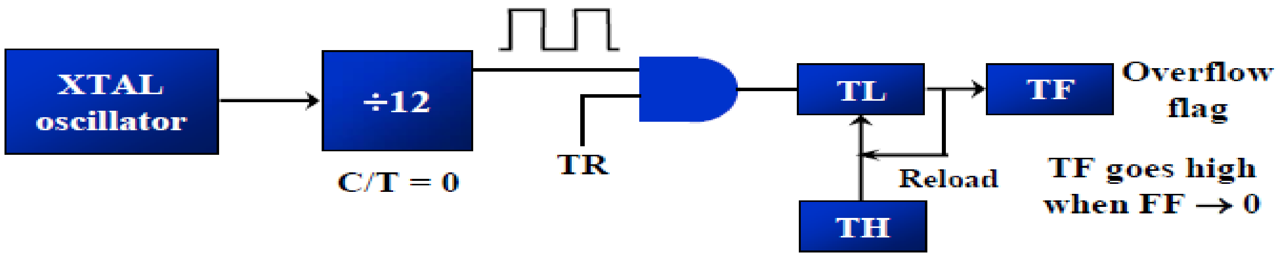
Mode 0: 13 bit Timer mode



Mode 1: 16 bit Timer mode



Mode 2: 8 bit auto reload mode



Mode 3: Split Timer mode

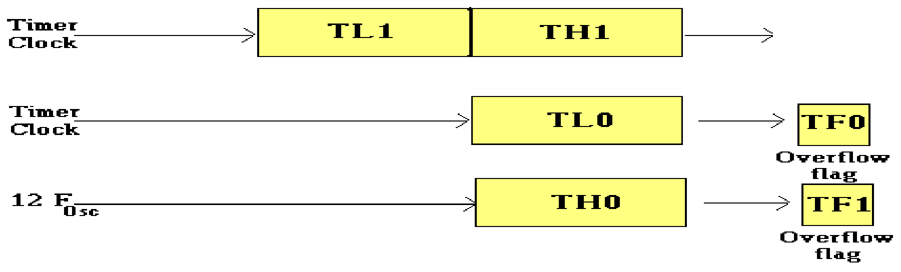


Figure: Modes of operation of Timer

TCON (Timer control register)

- ✓ TCON (timer control) register is an 8-bit register. TCON register is a bit-addressable register.

7	6	5	4	3	2	1	0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

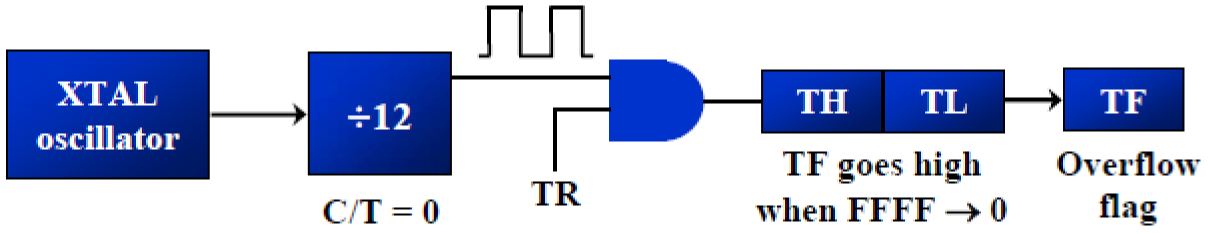
Bit Number	Bit Mnemonic	Description
7	TF1	Timer 1 overflow flag Cleared by hardware when processor vectors to interrupt routine. Set by hardware on timer/counter overflow, when the timer 1 register overflows.
6	TR1	Timer 1 run control bit Clear to turn off time/counter 1. Set to turn on timer/counter 1.
5	TF0	Timer 0 overflow flag Cleared by hardware when processor vectors to interrupt routine. Set by hardware on timer/counter overflow, when the timer 0 register overflows.
4	TR0	Timer 0 run control bit Clear to turn off time/counter 0. Set to turn on timer/counter 0.
3	IE1	External interrupt 1 edge flag. Cleared by hardware when interrupt is processed if edge-triggered. Set by hardware when external interrupt is detected on INT1 pin.
2	IT1	External interrupt 1 type control bit Clear to select low level active (level triggered) for external interrupt 1. Set to select falling edge active (edge triggered) for external interrupt 1.
1	IE0	External interrupt 0 edge flag Cleared by hardware when interrupt is processed if edge-triggered. Set by hardware when external interrupt is detected on INT0 pin.
0	IT0	External interrupt 0 type control bit Clear to select low level active (level triggered) for external interrupt 0. Set to select falling edge active (edge triggered) for external interrupt 0.

Timers of 8051 do starting and stopping by either software or hardware control

- ✓ For using software to start and stop the timer where GATE=0
- ✓ The start and stop of the timer are controlled by software using TR (timer start) bits TR_X and CLR_X
- ✓ The SETB instruction starts it, and it is stopped by the CLR instruction.
- ✓ These instructions start and stop the timers as long as GATE=0 in the TMOD register
- ✓ The hardware way of starting and stopping the timer is achieved by making GATE=1 in the TMOD register.

The following are the characteristics and operations of mode 1:

1. It is a 16-bit timer.
2. It allows value from 0000 to FFFFH.
3. Value to be loaded into the timer register TL and TH.
4. After TH and TL are loaded with a 16-bit initial value, the timer must be started
    - This is done by SETB TR0 for timer 0 and SETB TR1 for timer 1
5. After the timer is started, it starts to count up
    - It counts up until it reaches its limit of FFFFH
    - When it rolls over from FFFFH to 0000, it sets high a flag bit called TF (timer flag)
    - Each timer has its own timer flag.
    - There are TF0 for timer 0, and TF1 for timer 1.



6. Timer flag can be monitored,
    - When this timer flag is raised, to stop the timer with the CLR instructions.
    - CLR TR0 and CLR TR1, for timer 0 and timer 1 respectively.
    - After the timer reaches its limit and rolls over.
    - In order to repeat the process, TH and TL must be reloaded with the original value and TF must be reloaded to 0.

To generate a time delay

1. Load the TMOD register indicating which timer is to be used and which timer mode is selected.
2. Load registers TL and TH with initial count value.
3. Start the timer
4. Keep monitoring the timer flag (TF) with the JNB TFx , target to see if it is raised
    - Get out of the loop when TF becomes high
5. Stop the timer
6. Clear the TF flag for the next round

7. Go back to Step 2 to load TH and TL again.

Example 1:

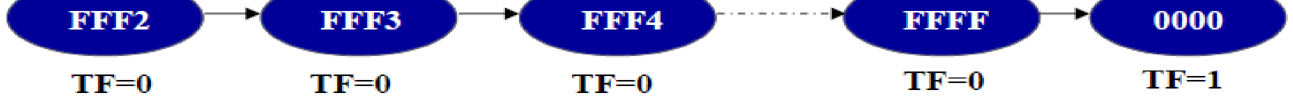
In the following program, we create a square wave of 50% duty cycle (with equal portions high and low) on the P1.5 bit. Timer 0 is used to generate the time delay. Analyze the program. (Nov 2014)

```
          MOV TMOD,#01      ;Timer 0, mode 1(16-bit mode)
HERE:     MOV TL0,#0F2H     ;TL0=F2H, the low byte
          MOV TH0,#0FFH     ;TH0=FFH, the high byte
          CPL P1.5          ;toggle P1.5
          ACALL DELAY
          SJMP HERE
DELAY:    SETB TR0          ;start the timer 0
AGAIN:    JNB TF0,AGAIN     ;monitor timer flag 0 until it rolls over
          CLR TR0           ;stop timer 0
          CLR TF0           ;clear timer 0 flag
          RET
```

In the above program notice the following steps.

1. TMOD is loaded.
2. FFF2H is loaded into TH0-TL0.
3. P1.5 is toggled for the high and low portions of the pulse.
4. The DELAY subroutine using the timer is called.
5. In the DELAY subroutine, timer 0 is started by the SETB TR0 instruction.
6. Timer 0 counts up with the passing of each clock, which is provided by the crystal oscillator.
   - As the timer counts up, it goes through the states of FFF3, FFF4, FFF5, FFF6, and so on until it reaches FFFFH.
   - One more clock rolls it to 0, raising the timer flag (TF0=1). At that point, the JNB instruction falls through.
7. Timer 0 is stopped by the instruction CLR TR0.
   - The DELAY subroutine ends and the process is repeated.

Notice that to repeat the process, we must reload the TL and TH registers, and start the process is repeated.



Example 2:

In Example 1, calculate the amount of time delay in the DELAY subroutine generated by the timer. Assume XTAL = 11.0592 MHz.

Solution:

- ✓ The timer works with a clock frequency of 1/12 of the XTAL frequency, we have 11.0592 MHz / 12 = 921.6 kHz as the timer frequency.
- ✓ As a result, each clock has a period of T =1/921.6kHz,T=1.085µs.
- ✓ In other words, Timer 0 counts up each 1.085µs resulting in delay = number of counts × 1.085µs.
- ✓ The number of counts for the roll over is FFFFH – FFF2H = 0DH (13 decimal).
- ✓ Add one to 13 because of the extra clock needed when it rolls over from FFFF to 0 and raise the TF flag.
- ✓ This gives 14 × 1.085µs = 15.19µs for half the pulse. For the entire period it is T = 2 × 15.19µs = 30.38µs as the time delay generated by the timer.

(a) In hexadecimal

(FFFF – YYXX + 1) ×1.085 µs, where YYXX are TH, TL initial values respectively. Notice that value YYXX are in hex.

(b) In decimal

Convert YYXX values of the TH, TL register to decimal to get a NNNN decimal, then (65536 - NNNN) × 1.085 µs

Example 3:

In Example 1, calculate the frequency of the square wave generated on pin P1.5.

Solution:

- ✓ In the timer delay calculation of Example 1, we did not include the overhead due to instruction in the loop.
- ✓ To get a more accurate timing, we need to add clock cycles due to these instructions in the loop.
- ✓ To do that, we use the machine cycle as shown below.

```
                                        Cycles
HERE:        MOV TL0,#0F2H              2
             MOV TH0,#0FFH              2
             CPL P1.5                   1
             ACALL DELAY                2
             SJMP HERE                  2
```

```
DELAY:   SETB TR0          1
AGAIN:   JNB TF0, AGAIN    14
         CLR TR0           1
         CLR TF0           1
         RET               2
                  Total    28
```

T = 2 × 28 × 1.085 us = 60.76 µs and F = 16458.2 Hz

Example 4:

Find the delay generated by timer 0 in the following code, using both of the Methods. Do not include the overhead due to instruction.

```
         CLR P2.3          ;Clear P2.3
         MOV TMOD,#01      ;Timer 0, 16-bitmode
HERE:    MOV TL0,#3EH      ;TL0=3Eh, the low byte
         MOV TH0,#0B8H     ;TH0=B8H, the high byte
         SETB P2.3         ;SET high timer 0
         SETB TR0          ;Start the timer 0
AGAIN:   JNB TF0,AGAIN     ;Monitor timer flag 0
         CLR TR0           ;Stop the timer 0
         CLR TF0           ;Clear TF0 for next round
         CLR P2.3
```

Solution:

(FFFFH – B83E + 1) = 47C2H = 18370 in decimal and 18370 × 1.085 µs = 19.93145 ms

<u>The following are the characteristics and operations of mode 2:</u>

1. It is an 8-bit timer. It allows only values of 00 to FFH to be loaded into the timer register TH.

2. After TH is loaded with the 8-bit value, the 8051 copies value to TL register.

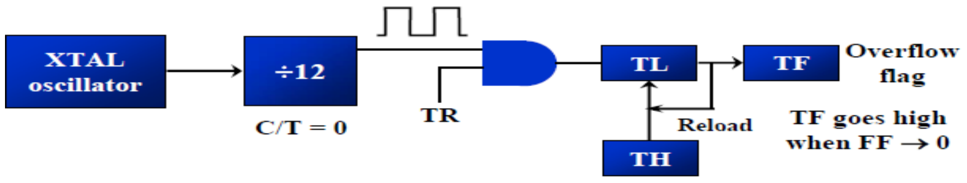
- Then the timer must be started.
- This is done by the instruction SETB TR0 for timer 0 and SETB TR1 for timer 1.

3. After the timer is started, it starts to count up by incrementing the TL register.

- It counts up until it reaches its limit of FFH
- When it rolls over from FFH to 00, it sets high the TF (timer flag)
- When the TL register rolls from FFH to 00 and TF is set to 1.
- TL is reloaded automatically with the original value kept by the TH register.
- To repeat the process, simply clear TF.

4. This makes mode 2 an auto-reload, in contrast with mode 1 in which the programmer has to reload

TH and TL



To generate a time delay

1. Load the TMOD value register indicating which timer is to be used, and the timer mode (mode 2) is selected.
2. Load the TH register with the initial count value.
3. Start timer.
4. Keep monitoring the timer flag (TF) with the JNB TFx, target, to see whether it is raised
   Get out of the loop when TF goes high
5. Clear the TF flag.
6. Go back to Step4, since mode 2 is auto reload.

Example 5:

Assume XTAL = 11.0592 MHz, find the frequency of the square wave generated on pin P1.0.

```
          MOV TMOD,#20H   ;T1/8-bit/auto reload
          MOV TH1,#5      ;TH1 = 5
          SETB TR1        ;start the timer 1
BACK:     JNB TF1,BACK    ;till timer rolls over
          CPL P1.0        ;P1.0 to hi, lo
          CLR TF1         ;clear Timer 1 flag
          SJMP BACK       ;mode 2 is auto-reload
```

Solution:

- ✓ In mode 2, no need to reload TH since it is auto-reload.
- ✓ Now (256 - 05) × 1.085 µs =251 × 1.085 µs = 272.33 µs is the high portion of the pulse.
- ✓ Since it is a 50% duty cycle square wave, the period T is twice.
- ✓ As a result T = 2 × 272.33 µs = 544.67 µs and the frequency = 1.83597 kHz

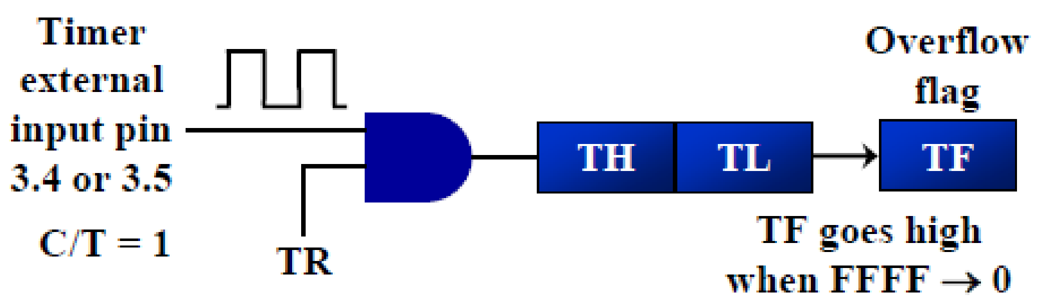
5.2: Timers as counters

Timers can also be used as counters.

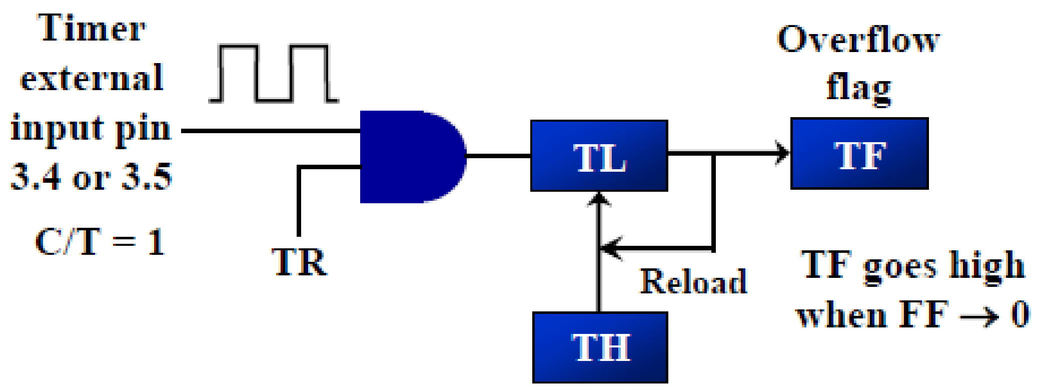
Which are used for counting events happening outside the 8051.

- When it is used as a counter, it is a pulse outside of the 8051 that increments the TH, TL register.
- TMOD and TH, TL registers are the same as in timer concept, except the source of the frequency.
- The C/T bit in the TMOD register decides the source of the clock for the timer
- When C/T = 1, the timer is used as a counter and gets its pulses from outside the 8051.
- The counter counts up as pulses are fed from pins 14 and 15.
- these pins are called T0 (timer 0 input) and T1 (timer 1 input)

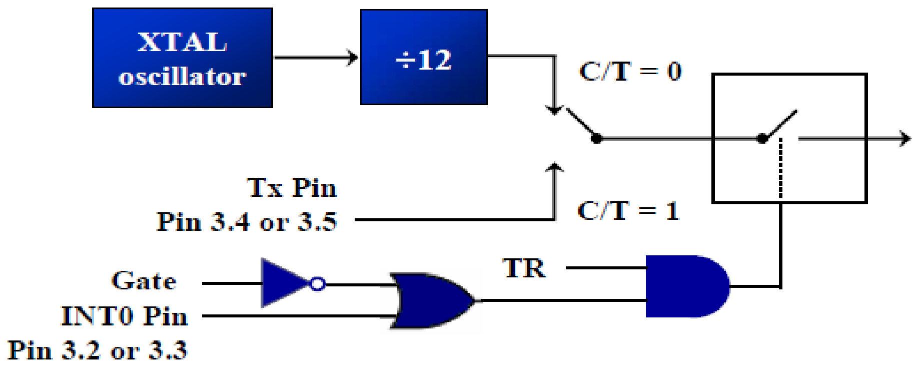
Timer with external input (Mode 1)


Timer external input pin 3.4 or 3.5
C/T = 1
TR
TH
TL
Overflow flag
TF
TF goes high when FFFF → 0

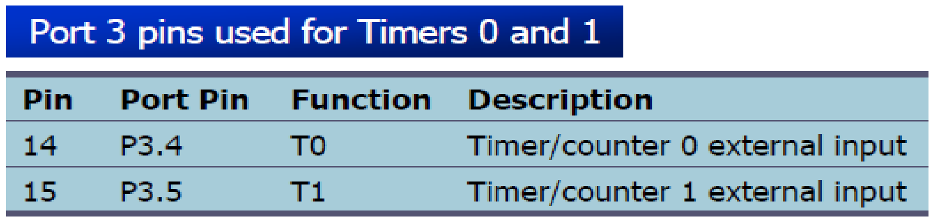
Timer with external input (Mode 2)



- ✓ If GATE = 1, the start and stop of the timer are done externally through pins P3.2 and P3.3 for timers 0 and 1, respectively
- ✓ This hardware allows starting or stopping the timer externally at any time via a simple switch



- ✓ The frequency for the timer is always 1/12th the frequency of the crystal attached to the 8051.



Port 3 pins used for Timers 0 and 1

Pin	Port Pin	Function	Description
14	P3.4	T0	Timer/counter 0 external input
15	P3.5	T1	Timer/counter 1 external input

Example 6:

Assuming that clock pulses are fed into pin T1, write a program for counter 1 in mode 2 to count the pulses and display the state of the TL1 count on P2, which connects to 8 LEDs.

Solution:

```
            MOV TM0D,#01100000B    ;counter 1, mode 2, C/T=1 external pulses
            MOV TH1,#0             ;clear TH1
            SETB P3.5              ;make T1 input
AGAIN:      SETB TR1               ;start the counter
BACK:       MOV A,TL1              ;get copy of TL
            MOV P2,A               ;display it on port 2
            JNB TF1,Back           ;keep doing, if TF = 0
            CLR TR1                ;stop the counter 1
            CLR TF1                ;make TF=0
            SJMP AGAIN             ;keep doing it
```

- ✓ Notice in the above program the role of the instruction SETB P3.5.
- ✓ Since ports are set up for output when the 8051 is powered up.
- ✓ So, we make P3.5 an input port by making it high.

- ✓ In other words, we must configure (set high) the T1 pin (pin P3.5) to allow pulses to be fed into it.

5.3: SERIAL COMMUNICATION

2. Explain the serial programming of 8051 with its associated registers. (May 2014, 2013)(Or)

Explain how to program for sending and receiving data serially using 8051 (April 2010, 2011)

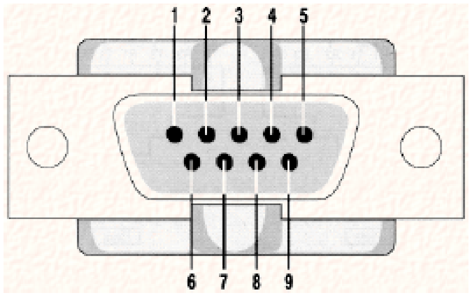
Explain 8051 serial port programming with examples. (May 2016, NOV 2012)

Explain the serial modes of operation of 8051 microcontroller. (May 2007)

RS232

- ✓ It is an interfacing standard RS232.
- ✓ It was set by the Electronics Industries Association (EIA) in 1960.
- ✓ The standard was set long before the advent of the TTL logic family.
- ✓ Its input and output voltage levels are not TTL compatible.
- ✓ In RS232, a 0 is represented by -3 to -25 V, while a 1 bit is +3 to +25 V.
- ✓ IBM introduced the DB-9 version of the serial I/O standard.

RS232 Connector DB-9



RS232 DB-9 Pins

Pin	Description
1	Data carrier detect (-DCD)
2	Received data (RxD)
3	Transmitted data (TxD)
4	Data terminal ready (DTR)
5	Signal ground (GND)
6	Data set ready (-DSR)
7	Request to send (-RTS)
8	Clear to send (-CTS)
9	Ring indicator (RI)

Handshake signals of MODEM

DTR (data terminal ready)

- When DTR =1, indicate that it is ready for communication.

DSR (data set ready)

- When DSR =1, indicate that it is ready for communication.

RTS (request to send)

- It asserts RTS to signal the modem that it has a byte of data to transmit.

CTS (clear to send)

- It is to receive, it sends out signal CTS,

DCD (data carrier detect)

- The modem asserts signal DCD to inform the DTE that a valid carrier has been detected.

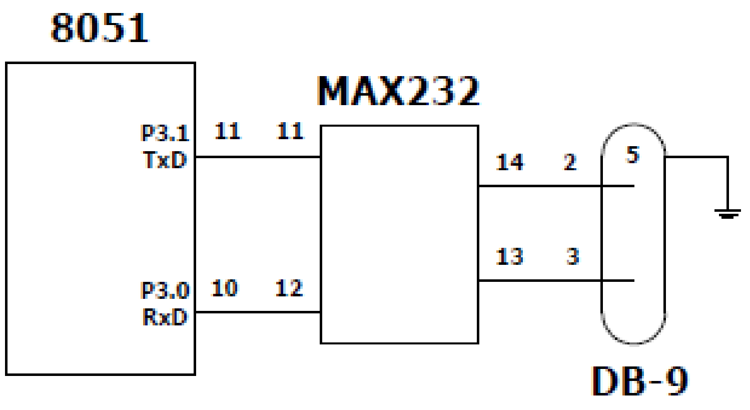
RI (ring indicator)

- An output from the modem and an input to a PC indicates that the telephone is ringing.

MAX232

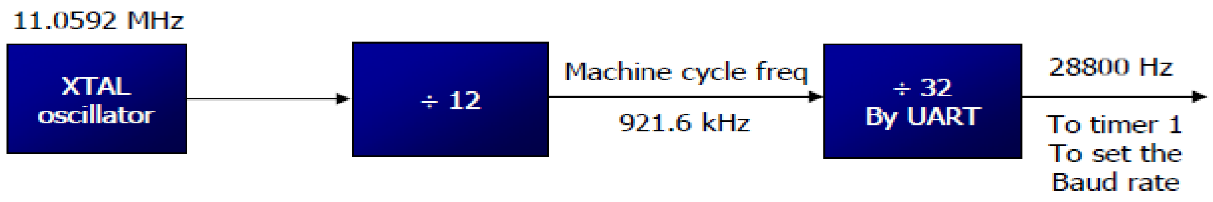
A line driver (MAX232) is required to convert RS232 voltage levels to TTL levels, and vice versa.

- 8051 has two pins that are used specifically for transferring and receiving data serially.
- These two pins are called TxD and RxD and are part of the port 3 (P3.0 and P3.1).
- These pins are TTL compatible.
- They require a line driver to make them RS232 compatible.



Baud rate:

- The baud rates in 8051 are programmable.
- 8051 divides the crystal frequency by 12 to get machine cycle frequency.
- 8051 UART circuitry divides the machine cycle frequency by 32.



- Timer 1 is used to set baud rate using TH1 register

Baud rate	**TH1 (decimal)**	**TH1(Hex)**
9600	-3	FD
4800	-6	FA
2400	-12	F4
1200	-24	E8

Explain in detail the serial communication registers of the 8051. (NOV 2009)

SBUF:

- It is an 8-bit register used for serial communication.
- For a byte data to be transferred via the TxD line:
- Byte must be placed in the SBUF register.
- Bytes are framed with the start and stop bits and transferred serially via the TxD line.
- SBUF holds the byte of data when it is received by 8051 RxD line.
- When the bits are received serially via RxD.
- 8051 de-frames byte by eliminating the stop and start bits.

SCON:

- It is an 8-bit register used to program the start bit, stop bit and data bits of data framing.

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

Bit Number	Bit Mnemonic	Description
SCON.7	SM0	Serial port mode specifier
SCON.6	SM1	Serial port mode specifier
SCON.5	SM2	Used for multiprocessor communication
SCON.4	REN	Set/Cleared by software to enable/disable reception
SCON.3	TB8	Not widely used
SCON.2	RB8	Not widely used
SCON.1	TI	Transmit interrupt flag. Set by hardware at the begin of the stop bit mode 1. And cleared by software
SCON.0	RI	Receive interrupt flag. Set by hardware at the begin of the stop bit mode 1. And cleared by software

SM0, SM1: Serial port mode specifiers

SM0	SM1	
0	0	Serial Mode 0
0	1	Serial Mode 1; 8-bit data, 1 stop bit, 1 start bit
1	0	Serial Mode 2
1	1	Serial Mode 3

In programming the 8051 to transfer character bytes serially

1. TMOD register is loaded with the value 20H, indicating the use of timer 1 in mode 2 (8-bit auto-reload) to set baud rate.
2. The TH1 is loaded with one of the values to set baud rate for serial data transfer.
3. The SCON register is loaded with the value 50H, indicating serial mode 1, where an 8-bit data is framed with start and stop bits.
4. TR1 is set to 1 to start timer 1
5. TI is cleared by CLR TI instruction.
6. The character byte to be transferred serially is written into SBUF register.
7. The TI flag bit is monitored with the use of instruction JNB TI, xx, to see if the character has been transferred completely.
8. To transfer the next byte, go to step 5.

Write a program for the 8051 to transfer letter "A" serially at 4800 baud, continuously.

Solution:

```
            MOV   TMOD, #20H ;timer 1, mode 2 (auto reload)
            MOV   TH1, #-6     ;4800 baud rate
            MOV   SCON, #50H  ;8-bit, 1 stop, REN enabled
            SETB  TR1          ;start timer 1
AGAIN:      MOV   SBUF, #"A"   ;letter "A" to trtansfer
HERE:       JNB   TI, HERE     ;wait for the last bit
            CLR   TI           ;clear TI for next char
            SJMP AGAIN         ;keep sending A
```

The steps that 8051 goes through in transmitting a character via TxD

1. The byte character to be transmitted is written into the SBUF register
2. The start bit is transferred
3. The 8-bit character is transferred on bit at a time
4. The stop bit is transferred
   - It is during the transfer of the stop bit that 8051 raises the TI flag, indicating that the last character was transmitted
5. By monitoring the TI flag, we make sure that we are not overloading the SBUF
   - If we write another byte into the SBUF before TI is raised, the un-transmitted portion of the previous byte will be lost.
6. After SBUF is loaded with a new byte, the TI flag bit must be forced to 0 by CLR TI in order for this new byte to be transferred

By checking the TI flag bit, we know whether or not the 8051 is ready to transfer another byte

- It must be noted that TI flag bit is raised by 8051 itself when it finishes data transfer
- It must be cleared by the programmer with instruction CLR TI
- If we write a byte into SBUF before the TI flag bit is raised, we risk the loss of a portion of the byte being transferred
- The TI bit can be checked by the instruction JNB TI,xx Using an interrupt.

Write a program for the 8051 to transfer "YES" serially at 9600 baud, 8-bit data, 1 stop bit do this continuously. (May 2006)

Solution:

```
            MOV   TMOD, #20H ;timer 1, mode 2 (auto reload)
            MOV   TH1, #-3      ;9600 baud rate
            MOV   SCON, #50H ;8-bit, 1 stop, REN enabled
            SETB  TR1           ;start timer 1
AGAIN:      MOV   A, # "Y"      ;transfer "Y"
            ACALL TRANS
            MOV   A, # "E"      ;transfer "E"
            ACALL TRANS
            MOV   A, # "S"      ;transfer "S"
            ACALL TRANS
            SJMP  AGAIN        ;keep doing it
;serial data transfer subroutine
TRANS:      MOV   SBUF, A      ;load SBUF
HERE:       JNB   TI, HERE     ;wait for the last bit
            CLR    TI          ;get ready for next byte
            RET   `
```

In programming the 8051 to receive character bytes serially

1. TMOD register is loaded with the value 20H, indicating the use of timer 1 in mode (8-bit auto-reload) to set baud rate
2. TH1 is loaded to set baud rate
3. The SCON register is loaded with the value 50H, indicating serial mode 1, where an 8-bit data is framed with start and stop bits
4. TR1 is set to 1 to start timer 1
5. RI is cleared by CLR RI instruction
6. The RI flag bit is monitored with the use of instruction JNB RI, xx to see if an entire character has been received yet
7. When RI is raised, SBUF has the byte, its contents are moved into a safe place.
8. To receive the next character, go to step 5.

Write a program for the 8051 to receive bytes of data serially and put them in P1, set the baud rate at 4800, 8-bit data and 1 stop bit. (NOV 2016)

Solution:

```
            MOV   TMOD, #20H ;timer 1, mode 2 (auto reload)
            MOV   TH1, #-6     ;4800 baud rate
            MOV   SCON, #50H  ;8-bit, 1 stop, REN enabled
            SETB  TR1          ;start timer 1
HERE:       JNB    RI, HERE    ;wait for char to come in
            MOV   A, SBUF      ;saving incoming byte in A
            MOV   P1, A        ;send to port 1
            CLR   RI           ;get ready to receive next byte
            SJMP  HERE         ;keep getting data
```

In receiving bit via its RxD pin, 8051 goes through the following steps.

1. It receives the start bit
   - Indicating that the next bit is the first bit of the character byte it is about to receive
2. The 8-bit character is received one bit at time
3. The stop bit is received
   - When receiving the stop bit 8051 makes RI = 1,indicating that an entire character byte has been received.
5. After the SBUF contents are copied into a safe place.
   - The RI flag bit must be forced to 0 by CLR RI in order to allow the next received character byte to be placed in SBUF.
   - Failure to do this causes loss of the received character.

There are two ways to increase the baud rate of data transfer

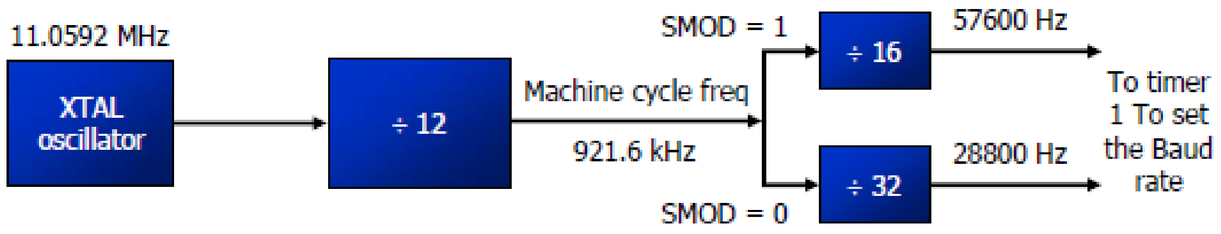
- To use a higher frequency crystal
- To change a bit in the PCON register

PCON

- PCON register is an 8-bit register
- When 8051 is powered up, SMOD is zero.
- We can set it to high by software and thereby double the baud rate.
- GF1, GF0: General flag bits
- PD: Power down mode
- IDL: Ideal mode

SMOD	--	--	--	GF1	GF0	PD	IDL

```
MOV   A,PCON      ;place a copy of PCON in ACC
SETB  ACC.7       ;make D7=1
MOV   PCON,A      ;changing any other bits
```



PIN Diagram of 8051 Microcontroller:

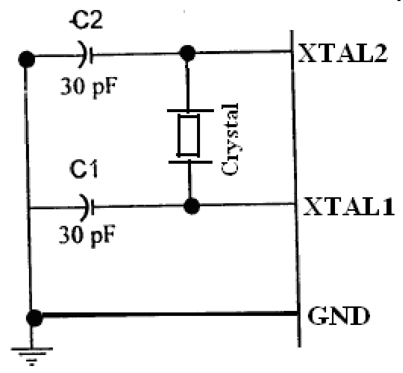
Explain Pin details of 8051 microcontroller. (MAY 2006)

Describe the functions of the following signals in 8051. RST, EA, PSEN and ALE. (NOV 2015)

- ✓ The 8051 microcontroller is available as a 40 pin DIP chip and it works at +5 volts DC.
- ✓ Among the 40 pins, a total of 32 pins are allotted for the four parallel ports P0, P1, P2 and P3 i.e each port occupies 8-pins.
- ✓ The remaining pins are VCC, GND, XTAL1, XTAL2, RST, EA ,PSEN.

XTAL1, XTAL2:

- ✓ These two pins are connected to Quartz crystal oscillator which runs the on-chip oscillator.
- ✓ The quartz crystal oscillator is connected to the two pins along with a capacitor of 30pF as shown in the circuit.
- ✓ If use a source other than the crystal oscillator, it will be connected to XTAL1 and XTAL2 is left unconnected.



RST:

- ✓ The RESET pin is an input pin and it is an active high pin.
- ✓ When a high pulse is applied to this pin, the microcontroller will reset and terminate all activities.
- ✓ Upon reset all the registers will reset to 0000 Value and SP register will reset to 0007 value.

$\overline{\text{EA}}$ (External Access):

- ✓ This pin is an active low pin.
- ✓ This pin is connected to ground when microcontroller is accessing the program code stored in the external memory.
- ✓ This pin is connected to Vcc when it is accessing the program code in the on chip memory.

$\overline{\text{PSEN}}$ (Program Store Enable):

- ✓ This is an output pin which is active low.
- ✓ When the microcontroller is accessing the program code stored in the external ROM, this pin is connected to the OE (Output Enable) pin of the ROM.

ALE (Address latch enable):

- ✓ This is an output pin, which is active high.
- ✓ This ALE pin will demultiplex the address and data bus.
- ✓ When the pin is high, the Address/ Data bus will act as address bus, otherwise the Address/ Data bus will act as Data bus.

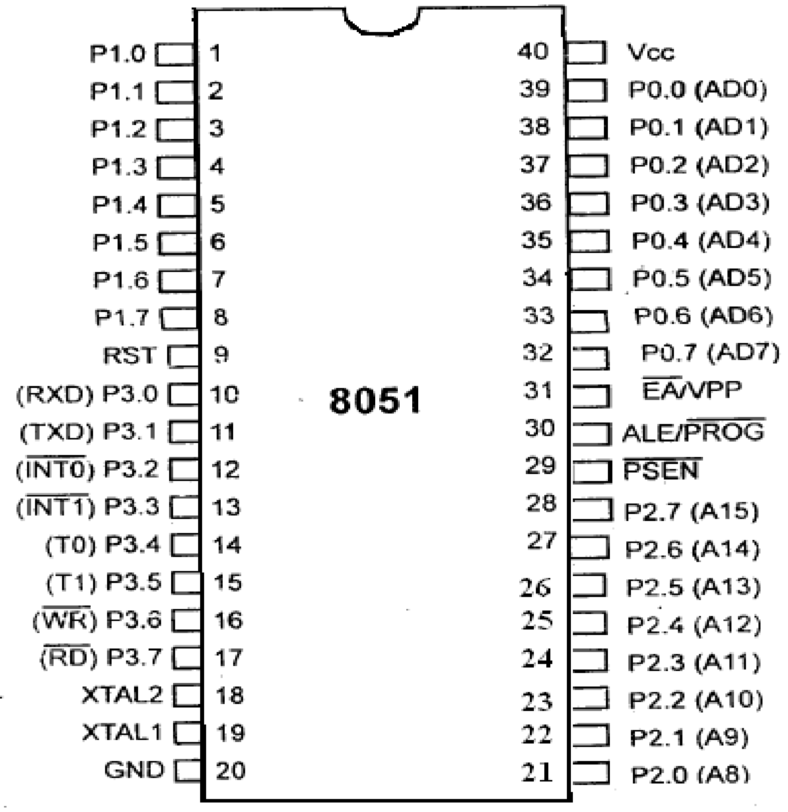


Figure: Pin diagram of 8051

P0.0- P0.7(AD0-AD7) :

- ✓ The port 0 pins multiplexed with Address/data pins.
- ✓ If the microcontroller is accessing external memory, these pins will act as address/data pins, otherwise they are used for Port 0 pins.

P2.0- P2.7 (A8-A15) :

- ✓ The port2 pins are multiplexed with the higher order address pins**.**
- ✓ When the microcontroller is accessing external memory, these pins provide the higher order address byte, otherwise they act as Port 2 pins.

P1.0- P1.7 :

- ✓ These 8-pins are dedicated to perform input or output port operations.

P3.0- P3.7:

- ✓ These 8-pins are meant for Port3 operations and also for some control operations like read, Write, Timer0, Timer1, INT0, INT1, RxD and TxD.

Program 1: Using timers in 8051 write a program to generate square wave 100ms, 50% duty cycle. (NOV 2014, May 2016, May 2012)

```
MOV TMOD, #01
Here: MOV TL0, #D7
MOV TH0, #B4
CPL P1.3
SETB TRO
Again: JNB TF0, Again
CLR TR0
CLR TF0
SJMP Here
```

Program 2: Write an 8051 ALP to multiply the given number 48H and 30H. (April 2017)

Mnemonics		Comments
Opcode	Operand	
MOV	A,#48	;Store data1 in accumulator
MOV	B,#30	;Store data2 in B register
MUL	AB	;Multiply both
MOV	DPTR,#4500	;Initialize memory location
MOVX	@DPTR,A	;Store lower order result
INC	DPTR	;Go to next memory location
MOV	A,B	;Store higher order result
MOVX	@DPTR,A	
L1: SJMP	L1	;Stop the program

Program 3: Write a program to add two 16 bit numbers. The numbers are 8C8D and 8D8C. Place the sum in R7 and R6. R6 should have the lower byte. (NOV 2010)

Mnemonics		Comments
Opcode	Operand	
MOV	A, #8D	;Store LSB data1 in accumulator
MOV	B, #8C	;Store LSB data2 in B register
ADD	A, B	;Add both
MOV	R6, A	;Store LSB result
MOV	A, #8C	;Store MSB data1 in accumulator
MOV	B, #8D	;Store MSB data2 in B register
ADD	A, B	;Add both
MOV	R7, A	;Store MSB result
L1: SJMP	L1	;Stop the program

UNIT- II EMBEDDED SYSTEMS

Embedded System Design Process – Model Train Controller – ARM Processor – Instruction Set Preliminaries – CPU – Programming Input and Output – Supervisor Mode – Exceptions and Trap –Models for programs – Assembly, Linking and Loading – Compilation Techniques – Program Level Performance Analysis.

Definition of Embedded system:

- An embedded system is a system that has software embedded in to computer-hardware, which makes a system dedicated for an application (s) or specific part of an application or product or part of a larger system.
- An embedded system is any device that includes a programmable computer but is not itself intended to be a general-purpose computer.
- Systems are the electronic systems that contain a microprocessor or a microcontroller, but we do not think of them as computers– the computer is hidden or embedded in the system.

Examples and Applications of Embedded systems

Examples

- ▯ Telecom
- ▯ Smart Cards,
- ▯ Missiles and Satellites,
- ▯ Computer Networking,
- ▯ Digital Consumer Electronics, and
- ▯ Automotive

Applications

▯ Mobile phone
▯ Digital camera
▯ Robots
▯ Point of sales terminals
▯ Automatic Chocolate Vending Machine
▯ Stepper motor controllers for a robotics system
▯ Washing or cooking system
▯ Multitasking toys

Challenges in Embedded System:

What are the Challenges in Embedded systems? (May 2012, 14, November 2008, May 2023)

The Challenges in Embedded systems are

How much hardware do we need?

For the great deal of control over the amount of computing power, we cannot only select microprocessor used but also the amount of memory and peripheral device etc.

The choice of hardware must meet both performance deadlines and manufacturing cost constraints.

How do we meet deadlines?

Brute - force way to meet deadline by speedup the hardware so that the program runs faster. Increase in speed makes the system more expensive and also increasing the CPU clock rate.

How do we minimize power consumption?

In battery powered applications, power consumption is very important.

In non-battery powered applications, excessive power consumption can increase heat.

One way to consume less power is to run the system more slowly. But slow down the system can obviously lead to missed deadlines.

Careful design is required to slow down the non-critical parts of the machine.

How do we design for upgradeability?

Hardware platform may be used over several product generations or for several different versions of a product in the same generation with few or no changes.

Hardware is designed such that the features are added by changing software.

Does it really work?

Reliability is very important when selling products. Reliability is important because running system try to eliminate bugs will too late and fixing bugs will more expensive.

1.1 COMPLEX SYSTEMS AND MICROPROCESSORS

- A PC is not itself an embedded computing system, although PCs are often used to build embedded computing systems. But a fax machine or a clock built from a microprocessor is an embedded computing system.
- This means that embedded computing system design is a useful skill for many types of product design. Automobiles, cell phones, and even household appliances make extensive use of microprocessors.
- Designers in many fields must be able to identify where microprocessors can be used, design a hardware platform with I/O devices that can support the required tasks, and implement software that performs the required processing.
- Computer engineering, like mechanical design or thermodynamics, is a fundamental discipline that can be applied in many different domains. But of course, embedded computing system design does not stand alone.

- Many of the challenges encountered in the design of an embedded computing system are not computer engineering—for example, they may be mechanical or analog electrical problems.

1.1 Embedding Computers

Briefly explain about the history of embedding computers with an example (April 2010)
Give building block of one consumer electronic product application system built using a typical embedded processor and highlight its two features that make its role effective (Dec 2022/Jan2023)

- Computers have been embedded into applications since the earliest days of computing. One example is the Whirlwind, a computer designed at MIT in the late 1940s and early 1950s. Whirlwind was also the first computer designed to support ***real-time*** operation and was originally conceived as a mechanism for controlling an aircraft simulator.
- Even though it was extremely large physically compared to today's computers (e.g., it contained over 4,000 vacuum tubes), its complete design from components to system was attuned to the needs of real-time embedded computing.
- The utility of computers in replacing mechanical or human controllers was evident from the very beginning of the computer era—for example, computers were proposed to control chemical processes in the late 1940s.
- A microprocessor is a single-chip CPU. Very large scale integration (VLSI) stet the acronym is the name technology has allowed us to put a complete CPU on a single chip since 1970s, but those CPUs were very simple.
- The first microprocessor, the Intel 4004, was designed for an embedded application, namely, a calculator.
- The calculator was not a general-purpose computer—it merely provided basic arithmetic functions. However, Ted Hoff of Intel realized that a general-purpose computer programmed properly could implement the required function, and that the computer-on-a-chip could then be reprogrammed for use in other products as well.
- Since integrated circuit design was (and still is) an expensive and time-consuming process, the ability to reuse the hardware design by changing the software was a key breakthrough.
- The HP-35 was the first handheld calculator to perform transcendental functions [Whi72]. It was introduced in 1972, so it used several chips to implement the CPU, rather than a single-chip microprocessor.
- However, the ability to write programs to perform math rather than having to design digital circuits to perform operations like trigonometric functions was critical to the successful design of the calculator.
- Automobile designers started making use of the microprocessor soon after single-chip CPUs became available.
- The most important and sophisticated use of microprocessors in automobiles was to control the engine: determining when spark plugs fire, controlling the fuel/air mixture, and so on. There was a trend toward electronics in automobiles in general—electronic devices could be used to replace the mechanical distributor.
- But the big push toward microprocessor-based engine control came from two nearly simultaneous developments: The oil shock of the 1970s caused consumers to place much

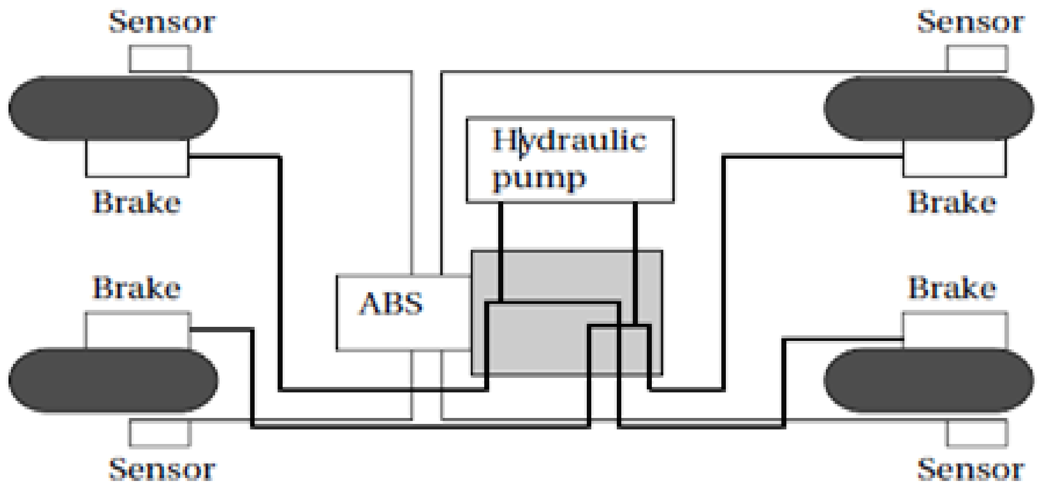
higher value on fuel economy, and fears of pollution resulted in laws restricting automobile engine emissions.

- The combination of low fuel consumption and low emissions is very difficult to achieve; to meet these goals without compromising engine performance, automobile manufacturers turned to sophisticated control algorithms that could be implemented only with microprocessors.
- Microprocessors come in many different levels of sophistication; they are usually classified by their word size.
- An 8-bit microcontroller is designed for low-cost applications and includes on-board memory and I/O devices; a 16-bit microcontroller is often used for more sophisticated applications that may require either longer word lengths or off-chip I/O and memory; and a 32-bit ***RISC*** microprocessor offers very high performance for computation-intensive applications.
- A programmable CPU was used rather than a hardwired unit for two reasons: First, it made the system easier to design and debug; and second, it allowed the possibility of upgrades and using the CPU for other purposes.
- A high-end automobile may have 100 microprocessors, but even inexpensive cars today use 40 microprocessors. Some of these microprocessors do very simple things such as detect whether seat belts are in use. Others control critical functions such as the ignition and braking systems.
- There are many household uses of microprocessors. The typical microwave oven has at least one microprocessor to control oven operation. Many houses have advanced thermostat systems, which change the temperature level at various times during the day.
- The modern camera is a prime example of the powerful features that can be added under microprocessor control. Digital television makes extensive use of embedded processors.
- In some cases, specialized CPUs are designed to execute important algorithms, an example is the CPU designed for audio processing in the SGS Thomson chip set.
- This processor is designed to efficiently implement programs for digital audio decoding.

BMW 850i brake and stability control system:

- The BMW 850i was introduced with a sophisticated system for controlling the wheels of the car. An antilock brake system (ABS) reduces skidding by pumping the brakes.
- An automatic stability control (ASC +T) system intervenes with the engine during maneuvering to improve the car’s stability. These systems actively control critical systems of the car; as control systems, they require inputs from and output to the automobile.
- Let’s first look at the ABS. The purpose of an ABS is to temporarily release the brake on a wheel when it rotates too slowly—when a wheel stops turning, the car starts skidding and becomes hard to control. It sits between the hydraulic pump, which provides power to the brakes, and the brakes themselves as seen in the following diagram.

- This hookup allows the ABS system to modulate the brakes in order to keep the wheels from locking.
- The ABS system uses sensors on each wheel to measure the speed of the wheel. The wheel speeds are used by the ABS system to determine how to vary the hydraulic fluid pressure to prevent the wheels from skidding.



Antilock brake system (ABS)

- The ASC+Tsystem's job is to control the engine power and the brake to improve the car's stability.
- The ASC+T controls four different systems: throttle, ignition timing, differential brake, and (on automatic transmission cars) gear shifting.
- The ASC + T can be turned off by the driver, which can be important when operating with tire snow chains.
- The ABS and ASC+ T must clearly communicate because the ASC + T interacts with the brake system. Since the ABS was introduced several years earlier than the ASC + T, it was important to be able to interface ASC + T to the existing ABS module, as well as to other existing electronic modules.
- The engine and control management units include the electronically controlled throttle, digital engine management, and electronic transmission control. The ASC + T control unit has two microprocessors on two printed circuit boards, one of which concentrates on logic-relevant components and the other on performance-specific components.

1.1.1 Characteristics of Embedded Computing Applications

Enumerate the characteristics of embedded computing systems.

(NOV/DEC 2010, May 2014, May 2023)

Embedded computing is in many ways much more demanding than the sort of programs that you may have written for PCs or workstations. Functionality is important in both general-purpose computing and embedded computing, but embedded applications must meet many other constraints as well.

On the one hand, embedded computing systems have to provide sophisticated functionality:

- ***Complex algorithms**:* The operations performed by the microprocessor may be very sophisticated. For example, the microprocessor that controls an automobile engine must perform complicated filtering functions to optimize the performance of the car while minimizing pollution and fuel utilization.
- ***User interface**:* Microprocessors are frequently used to control complex user interfaces that may include multiple menus and many options. The moving maps in Global Positioning System (GPS) navigation are good examples of sophisticated user interfaces.

To make things more difficult, embedded computing operations must often be performed to meet deadlines:

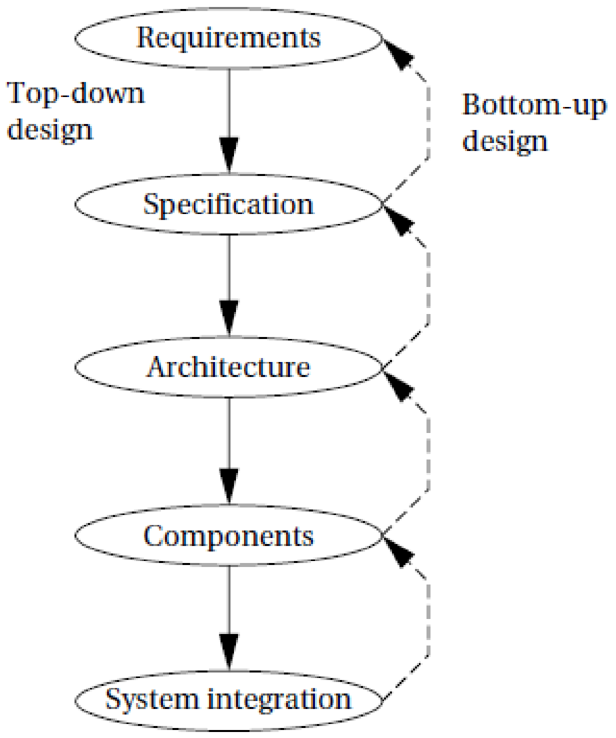
- ***Real time:*** Many embedded computing systems have to perform in real time— if the data is not ready by a certain deadline, the system breaks.
- In some cases, failure to meet a deadline is unsafe and can even endanger lives. In other cases, missing a deadline does not create safety problems but does create unhappy customers—missed deadlines in printers, for example, can result in scrambled pages.
- ***Multirate:*** Not only must operations be completed by deadlines, but many embedded computing systems have several real-time activities going on at the same time.
- They may simultaneously control some operations that run at slow rates and others that run at high rates. Multimedia applications are prime examples of ***multirate*** behavior.
- The audio and video portions of a multimedia stream run at very different rates, but they must remain closely synchronized. Failure to meet a deadline on either the audio or video portions spoils the perception of the entire presentation.
- ***Power and energy:***Power consumption directly affects the cost of the hardware, since a larger power supply may be necessary.
- Energy consumption affects battery life, which is important in many applications, as well as heat consumption, which can be important even in desktop applications.
- ***Manufacturing cost:*** The total cost of building the system is very important in many cases Manufacturing cost is determined by many factors, including the type of microprocessor used, the amount of memory required, and the types of I/O devices.

1.2 THE EMBEDDED SYSTEM DESIGN PROCESS:

Briefly explain about the steps involved in embedded system design. (NOV/DEC 2006, 2007, 2009, May 2012, April 2018, Dec20) Designing with Computing platforms (Dec2022/Jan 2023, May 2023)

The embedded system design process aimed at two objectives.

- First, it will give us an introduction to the various steps in embedded system design before we delve into them in more detail. Second, it will allow us to consider the design ***methodology*** itself. A design methodology is important for three reasons.
- First, it allows us to keep a scorecard on a design to ensure that we have done everything we need to do, such as optimizing ***performance*** or performing functional tests.
- Second, it allows us to develop computer-aided design tools. Developing a single program that takes in a concept for an embedded system and emits a completed design would be a daunting task, but by first breaking the process into manageable steps, we can work on automating the steps one at a time.
- Third, a design methodology makes it much easier for members of a design team to communicate.
- By defining the overall process, team members can more easily understand what they are supposed to do, what they should receive from other team members at certain times and what they are to hand off when they complete their assigned steps.
- Since most embedded systems are designed by teams, coordination is perhaps the most important role of a well-defined design methodology.
-   In this top–down view, we start with the system ***requirements***. In the next step, ***specification***, we create a more detailed description of what we want.
- But the specification states only how the system behaves, not how it is built.
- The details of the system's internals begin to take shape when we develop the architecture, which gives the system structure in terms of large components.
- Once we know the components we need, we can design those components, including both software modules and any specialized hardware we need. Based on those components, we can finally build a complete system.



Major levels of abstraction in the design process.

- The ***top–downdesign*** will begin with the most abstract description of the system and conclude with concrete details. The alternative is a **bottom–up** view in which we start with components to build a system.
- Bottom–up design steps are shown in the figure as dashed-line arrows. We need bottom–up design because we do not have perfect insight into how later stages of the design process will turn out.
- Decisions at one stage of design are based upon estimates of what will happen later: How fast can we make a particular function run? How much memory will we need? How much system bus capacity do we need? We also need to consider the major goals of the design.
    - Manufacturing cost.
    - Performance (both overall speed and deadlines); and
    - Power consumption.
    - We must also consider the tasks we need to perform at every step in the design process. At each step in the design, we add detail:
    - We must *analyze* the design at each step to determine how we can meet the Specifications.
    - We must then *refine* the design to add detail.
    - We must verify the design to ensure that it still meets all system goals, such as cost, speed and so on.

1.2.1 Requirements

- Clearly, before we design a system, we must know what we are designing. The initial stages of the design process capture this information for use in creating the architecture and components.
- We generally proceed in two phases: First, we gather an informal description from the customers known as requirements, and we refine the requirements into a specification that contains enough information to begin designing the system architecture.
- Separating out requirements analysis and specification is often necessary because of the large gap between what the customers can describe about the system they want and what the architects need to design the system.
- Consumers of embedded systems are usually not themselves embedded system designers or even product designers. Their understanding of the system is based on how they envision users' interactions with the system. They may have unrealistic expectations as to what can be done within their budgets; and they may also express their desires in a language very different from system architects' jargon.
- Capturing a consistent set of requirements from the customer and then massaging those requirements into a more formal specification is a structured way to manage the process of translating from the consumer's language to the designer's.
- Requirements may be ***functional*** or ***nonfunctional***. We must of course capture the basic functions of the embedded system, but functional description is often not sufficient.

Typical nonfunctional requirements include:

Performance: The speed of the system is often a major consideration both for the usability of the system and for its ultimate cost. As we have noted, performance may be a combination of soft performance metrics such as approximate time to perform a user-level function and hard deadlines by which a particular operation must be completed.

- *Cost:* The target cost or purchase price for the system is almost always a consideration. Cost typically has two major components: ***manufacturing cost*** includes the cost of components and assembly; ***nonrecurring engineering*** **(NRE)** costs include the personnel and other costs of designing the system.
- *Physical size and weight:* The physical aspects of the final system can vary greatly depending upon the application. An industrial control system for an assembly line may be designed to fit into a standard-size rack with no strict limitations on weight. A handheld device typically has tight requirements on both size and weight that can ripple through the entire system design.
- *Power consumption:* Power, of course, is important in battery-powered systems and is often important in other applications as well. Power can be specified in the requirements stage in terms of battery life—the customer is unlikely to be able to describe the allowable wattage.
- Validating a set of requirements is ultimately a psychological task since it requires understanding both what people want and how they communicate those needs. One good

way to refine at least the user interface portion of a system's requirements is to build a ***mock-up***.

- The mock-up may use canned data to simulate functionality in a restricted demonstration, and it may be executed on a PC or a workstation. But it should give the customer a good idea of how the system will be used and how the user can react to it.
- Physical, nonfunctional models of devices can also give customers a better idea of characteristics such as size and weight.

Sample requirements form.

Name
Purpose
Inputs
Outputs
Functions
Performance
Manufacturing cost
Power
Physical size and weight

- Requirements analysis for big systems can be complex and time consuming. However, capturing a relatively small amount of information in a clear, simple format is a good start toward understanding system requirements.
- To introduce the discipline of requirements analysis as part of system design, we will use a simple requirements methodology.

We can use the requirement form as a checklist in considering the basic characteristics of the system.

- ***Name:*** This is simple but helpful. Giving a name to the project not only simplifies talking about it to other people but can also crystallize the purpose of the machine.
- ***Purpose:*** This should be a brief one- or two-line description of what the system is supposed to do. If you can't describe the essence of your system in one or two lines, chances are that you don't understand it well enough.
- ***Inputs and outputs:*** These two entries are more complex than they seem. The inputs and outputs to the system encompass a wealth of detail:
- **Types *of data:*** Analog electronic signals? Digital data? Mechanical inputs?
- ***Data characteristics:*** Periodically arriving data, such as digital audio samples? Occasional user inputs? How many bits per data element?
- ***Types of I/O devices:*** Buttons? Analog/digital converters? Video displays?
- ***Functions:*** This is a more detailed description of what the system does. A good way to approach this is to work from the inputs to the outputs: When the system receives an input, what does it do? How do user interface inputs affect these functions? How do different functions interact?

- ***Performance:*** Many embedded computing systems spend at least some time controlling physical devices or processing data coming from the physical world. In most of these cases, the computations must be performed within a certain time frame. It is essential that the performance requirements be identified early since they must be carefully measured during implementation to ensure that the system works properly.
- ***Manufacturing cost:*** This includes primarily the cost of the hardware components. Even if you don't know exactly how much you can afford to spend on system components, you should have some idea of the eventual cost range. Cost has a substantial influence on architecture: A machine that is meant to sell at $10 most likely has a very different internal structure than a $100 system.
- ***Power:*** Similarly, you may have only a rough idea of how much power the system can consume, but a little information can go a long way. Typically, the most important decision is whether the machine will be battery powered or plugged into the wall. Battery-powered machines must be much more careful about how they spend energy.
- ***Physical size and weight:***You should give some indication of the physical size of the system to help guide certain architectural decisions. A desktop machine has much more flexibility in the components used than, for example, a lapel mounted voice recorder.

1.3 DESIGN EXAMPLE: MODEL TRAIN CONTROLLER

Enumerate the steps involved in design of model train controller. (NOV/DEC 2007, Dec 2010, May 2012, Nov 2017, April 2018, Dec20, Apr21, Dec 21) or Design a Model Train Controller with suitable diagrams and explain (Nov/Dec 2018) (Dec 2022/Jan 2023)

- In order to learn how to use UML to model systems, we will specify a simple system, a model train controller. The user sends messages to the train with a control box attached to the tracks.
- The control box may have familiar controls such as a throttle, emergency stop button, and so on. Since the train receives its electrical power from the two rails of the track, the control box can send signals to the train over the tracks by modulating the power supply voltage. As shown in the figure, the control panel sends packets over the tracks to the receiver on the train.
- The train includes analog electronics to sense the bits being transmitted and a control system to set the train motor's speed and direction based on those commands.
- Each packet includes an address so that the console can control several trains on the same track; the packet also includes an error correction code (ECC) to guard against transmission errors. This is a one-way communication system the model train cannot send commands back to the user.
- We start by analyzing the requirements for the train control system.We will base our system on a real standard developed for model trains.We then develop two specifications: a simple, high-level specification and then a more detailed specification.

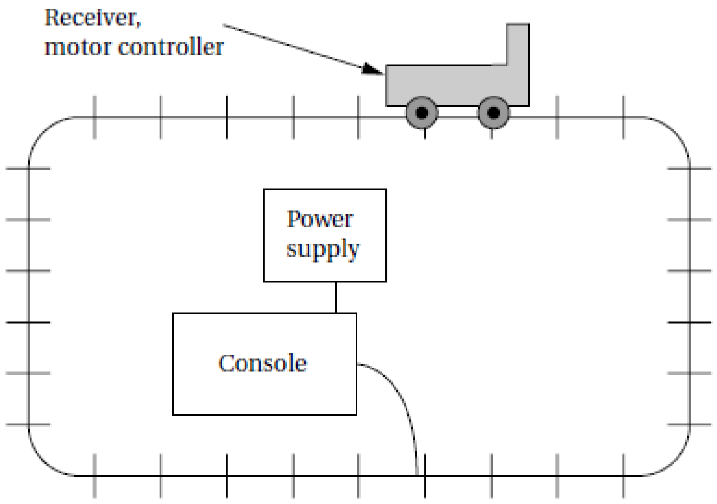
1.3.1 Requirements

Before we can create a system specification, we have to understand the requirements. Here is a basic set of requirements for the system:

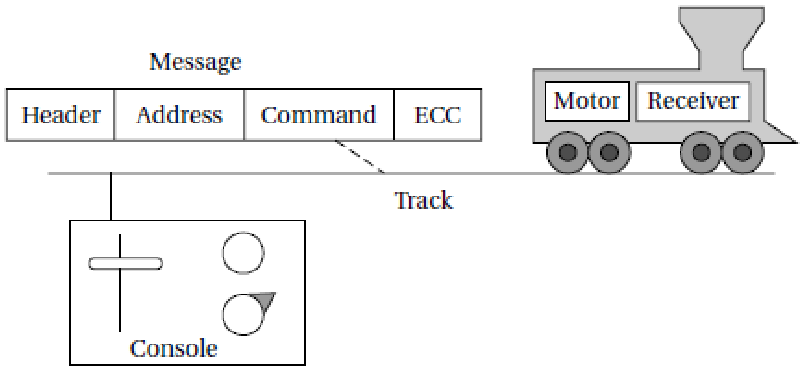
- The console shall be able to control up to eight trains on a single track.
- The speed of each train shall be controllable by a throttle to at least 63 different levels in each direction (forward and reverse).
- There shall be an inertia control that shall allow the user to adjust the responsiveness of the train to commanded changes in speed.
- There shall be an emergency stop button.
- An error detection scheme will be used to transmit messages
- Higher inertia means that the train responds more slowly to a change in the throttle, simulating the inertia of a large train. The inertia control will provide at least eight different levels.

Putting the requirements into chart format:

Name	Model train controller
Purpose	Control speed of up to eight model trains
Inputs	Throttle, inertia setting, emergency stop, train
Outputs	number Train control signals
Functions	Set engine speed based upon inertia settings;
Performance	respond to emergency stop ,Can update train speed at least 10 times per second
Manufacturing cost	$50
Power ,Physical size and weight	10W (plugs into wall) Console should be comfortable for two hands, approximate size of standard keyboard; weight<2 pounds



System setup



Signaling the train

Let's consider the entries in the form: **A Model train control system**

- ***Name:*** This is simple but helpful. Giving a name to the project not only simplifies talking about it to other people but can also crystallize the purpose of the machine.
- ***Purpose:*** This should be a brief one- or two-line description of what the system is supposed to do. If you can't describe the essence of your system in one or two lines, chances are that you don't understand it well enough.
- ***Inputs and outputs:*** These two entries are more complex than they seem. The inputs and outputs to the system encompass a wealth of detail:
- **Types *of data:*** Analog electronic signals? Digital data? Mechanical inputs?
- ***Data characteristics:*** Periodically arriving data, such as digital audio samples? Occasional user inputs? How many bits per data element?
- ***Types of I/O devices:*** Buttons? Analog/digital converters? Video displays?
- ***Functions:*** This is a more detailed description of what the system does. A good way to approach this is to work from the inputs to the outputs

When the system receives an input, what does it do? How do user interface inputs affect these functions? How do different functions interact?

- ***Performance:*** Many embedded computing systems spend at least some time controlling physical devices or processing data coming from the physical world.
- In most of these cases, the computations must be performed within a certain time frame. It is essential that the performance requirements be identified early since they must be carefully measured during implementation to ensure that the system works properly.
- ***Manufacturing cost:*** This includes primarily the cost of the hardware components. Even if you don't know exactly how much you can afford to spend on system components, you should have some idea of the eventual cost range. Cost has a substantial influence on architecture.
- A machine that is meant to sell at $10 most likely has a very different internal structure than a $100 system.
- ***Power:*** Similarly, you may have only a rough idea of how much power the system can consume, but a little information can go a long way. Typically, the most important decision is whether the machine will be battery powered or plugged into the wall. Battery-powered machines must be much more careful about how they spend energy.
- ***Physical size and weight:*** You should give some indication of the physical size of the system to help guide certain architectural decisions. A desktop machine has much more flexibility in the components used than, for example, a lapel mounted voice recorder.

1.3.2 DCC

- The **Digital Command Control (DCC)** was created by the National Model Railroad Association to support interoperable digitally-controlled model trains.
- Hobbyists started building homebrew digital control systems in the 1970s and Marklin developed its own digital control system in the 1980s.
- DCC was created to provide a standard that could be built by any manufacturer so that hobbyists could mix and match components from multiple vendors.

The DCC standard is given in two documents:

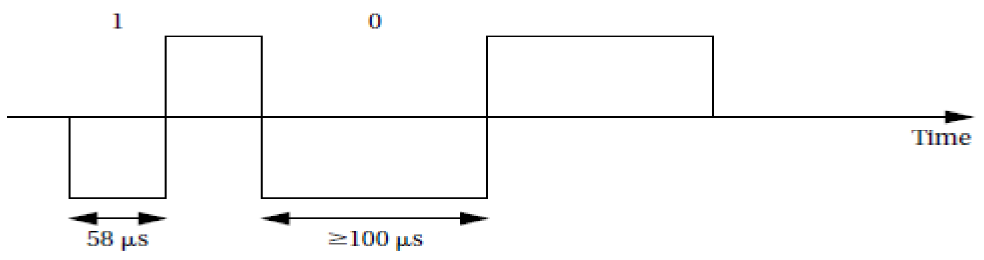
  - Standard S-9.1, the DCC Electrical Standard, defines how bits are encoded on the rails for transmission.
  - Standard S-9.2, the DCC Communication Standard, defines the packets that carry information.

- Any DCC-conforming device must meet these specifications. DCC also provides several recommended practices. These are not strictly required but they provide some hints to manufacturers and users as to how to best use DCC.
- The DCC standard does not specify many aspects of a DCC train system. It doesn't define the control panel, the type of microprocessor used, the programming language to be used, or many other aspects of a real model train system.
- The standard concentrates on those aspects of system design that are necessary for interoperability.

- Over standardization, or specifying elements that do not really need to be standardized, only makes the standard less attractive and harder to implement.
- The Electrical Standard deals with voltages and currents on the track
- The standard must be carefully designed because the main function of the track is to carry power to the locomotives. The signal encoding system should not interfere with power transmission either to DCC or non-DCC locomotives. A key requirement is that the data signal should not change the DC value of the rails.
- The data signal swings between two voltages around the power supply voltage. bits are encoded in the time between transitions, not by voltage levels. A 0 is at least 100 ms while a 1 is nominally 58ms.
- The durations of the high (above nominal voltage) and low (below nominal voltage) parts of a bit are equal to keep the DC value constant. The specification also gives the allowable variations in bit times that a conforming DCC receiver must be able to tolerate.
- The standard also describes other electrical properties of the system, such as allowable transition times for signals.
- The DCC Communication Standard describes how bits are combined into packets and the meaning of some important packets.
- Some packet types are left undefined in the standard but typical uses are given in Recommended Practices documents.

We can write the basic packet format as a regular expression:

PSA (sD) + E (1.1)



Bit encoding in DCC.

In this regular expression:

- *P* is the preamble, which is a sequence of at least 10 1 bits. The command station should send at least 14 of these 1 bits, some of which may be corrupted during transmission.
- *S* is the packet start bit. It is a 0 bit.
- *A* is an address data byte that gives the address of the unit, with the most significant bit of the address transmitted first. An address is eight bits long. The addresses 00000000, 11111110, and 11111111 are reserved.
- *s* is the data byte start bit, which, like the packet start bit, is a 0.

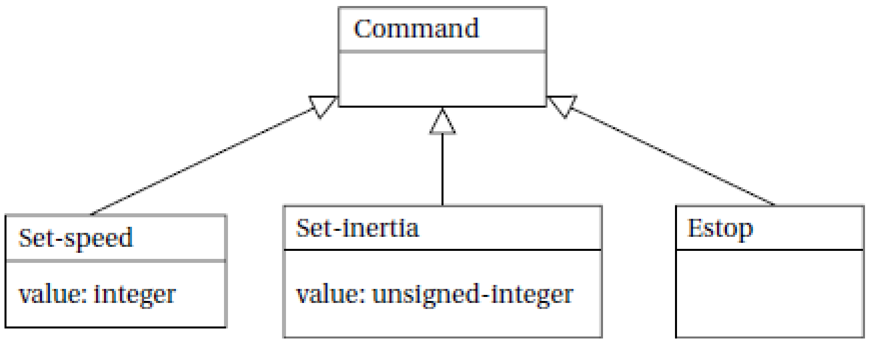
- *D* is the data byte, which includes eight bits. A data byte may contain an address, instruction, data, or error correction information.
- *E* is a packet end bit, which is a 1 bit.
- A packet includes one or more data byte start bit/data byte combinations. Note that the address data byte is a specific type of data byte.
- A ***baseline packet*** is the minimum packet that must be accepted by all DCC implementations. More complex packets are given in a Recommended Practice document.
- A baseline packet has three data bytes: an address data byte that gives the intended receiver of the packet; the instruction data byte provides a basic instruction; and an error correction data byte is used to detect and correct transmission errors.
- The instruction data byte carries several pieces of information. Bits 0–3 provide a 4-bit speed value.
- Bit 4 has an additional speed bit, which is interpreted as the least significant speed bit. Bit 5 gives direction, with 1 for forward and 0 for reverse. Bits 7–8 are set at 01 to indicate that this instruction provides speed and direction.
- The error correction data byte is the bitwise exclusive OR of the address and instruction data bytes.
- The standard says that the command unit should send packets frequently since a packet may be corrupted. Packets should be separated by at least 5 ms.

1.3.3 Conceptual Specification:

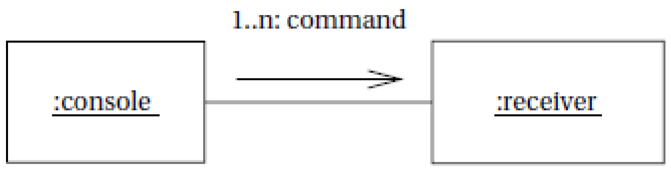
Write short notes on conceptual specification of model train controller.

- Digital Command Control specifies some important aspects of the system, particularly those that allow equipment to interoperate. But DCC deliberately does not specify everything about a model train control system.
- A conceptual specification allows us to understand the system a little better. This specification does not correspond to what any commercial DCC controllers do, butis simple enough to allow us to cover some basic concepts in system design
- A train control system turns ***commands*** into ***packets***. A command comes from the command unit while a packet is transmitted over the rails.
- Commands and packets may not be generated in a 1-to-1 ratio. In fact, the DCC standard says that command units should resend packets in case a packet is dropped during transmission.
- We now need to model the train control system itself. There are clearly two major subsystems: the command unit and the train-board component. Each of these subsystems has its own internal structure.
- The command unit and receiver are each represented by objects; the command unit sends a sequence of packets to the train's receiver, as illustrated by the arrow.

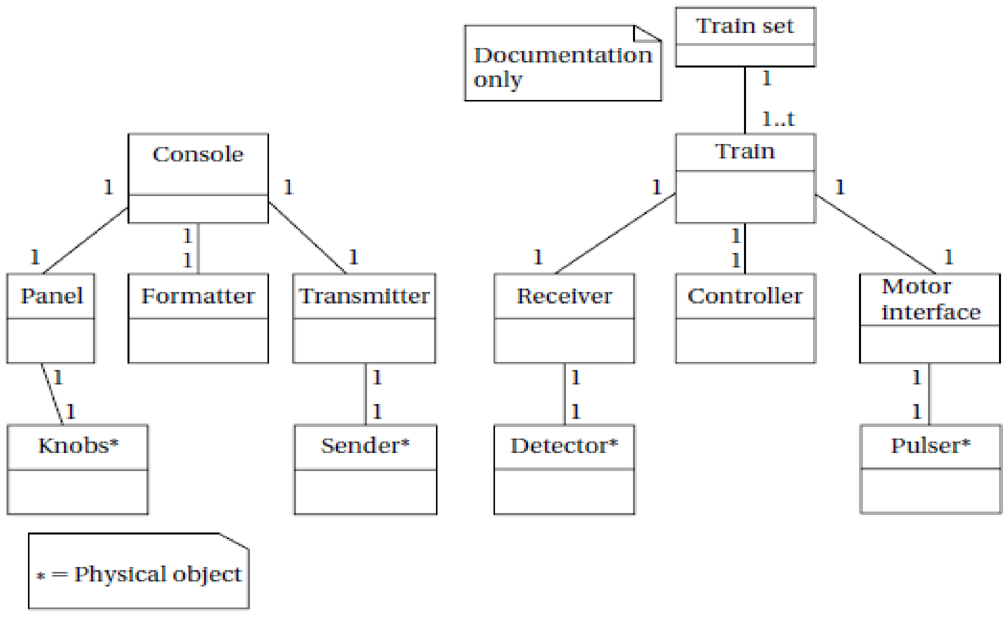
- The notation on the arrow provides both the type of message sent and its sequence in a flow of messages; since the console sends all the messages, we have numbered the arrow's messages as 1..*n*. Those messages are of course carried over the track.
- Since the track is not a computer component and is purely passive, it does not appear in the diagram. However, it would be perfectly legitimate to model the track in the collaboration diagram, and in some situations it may be wise to model such nontraditional components in the specification diagrams. For example, if we are worried about what happens when the track breaks, modeling the tracks would help us identify failure modes and possible recovery mechanisms.



Class diagram for the train controller messages.



UML collaboration diagram for major subsystems of the train controller system.



A UML class diagram for the train controller showing the composition of the subsystems.

- Let’s break down the command unit and receiver into their major components. The console needs to perform three functions: read the state of the front panel on the command unit, format messages, and transmit messages. The train receiver must also perform three major functions: receive the message, interpret the message.
- It shows the console class using three classes, one for each of its major components. These classes must define some behaviors, but for the moment we will concentrate on the basic characteristics of these classes:
- The *Console* class describes the command unit’s front panel, which contains the analog knobs and hardware to interface to the digital parts of the system.
- The *Formatter* class includes behaviors that know how to read the panel knobs and creates a bit stream for the required message.
- The *Transmitter* class interfaces to analog electronics to send the message along the track.

- There will be one instance of the *Console* class and one instance of each of the component classes, as shown by the numeric values at each end of the relationship links. We have also shown some special classes that represent analog components, ending the name of each with an asterisk:
- *Knobs* describes the actual analog knobs, buttons, and levers on the control panel.
- *Sender* describes the analog electronics that send bits along the track.
- Likewise, the Train makes use of three other classes that define its components:
- The *Receiver* class knows how to turn the analog signals on the track into digital form.
- The *Controller* class includes behaviors that interpret the commands and figures out how to control the motor.
- The *Motor interface* class defines how to generate the analog signals required to control the motor.

We define two classes to represent analog components:

  - *Detector* detects analog signals on the track and converts them into digital form.
  - *Pulser* turns digital commands into the analog signals required to control the motor speed.
  - We have also defined a special class, *Train set*, to help us remember that the system can handle multiple trains.

- The values on the relationship edge show that one train set can have *t* trains. We would not actually implement the train set class, but it does serve as useful documentation of the existence of multiple receivers.

ARM PROCESSOR

Introduction:

ARM is actually a family of RISC architectures that have been developed over many years. ARM does not manufacture its own VLSI devices; rather, it licenses its architecture to companies who either manufacture the CPU itself or integrate the ARM processor into a larger system. The textual description of instructions, as opposed to their binary representation, is called an assembly language. ARM instructions are written one per line, starting after the first column. Comments begin with a semicolon and continue to the end of the line. A label, which gives a name to a memory location, comes at the beginning of the line, starting in the first column.

Q1. Discuss about ARM architecture versions.

ARM Architecture Versions:

The ARM architecture has evolved significantly and will continue to be developed in the future. Six major versions of the instruction set have been defined to date, denoted by the version numbers from 1 to 6. Of these, the first three versions including the original 26-bit architecture (the 32-bit architecture was introduced at ARMv3) are now OBSOLETE. Other versions are Version 4, version 5 and version6.Versions can be qualified with variant letters to specify collections of additional instructions that are included as an architecture extension. Extensions are typically included in the base architecture of the next version number, ARMv5T being the notable exception. Provision is also made to exclude variants by prefixing the variant letter with x, for example the XP variant described below in the summary of version 5 features.

The valid architecture variants are as follows: ARMv4, ARMv4T, ARMv5T, (ARMv5TExP), ARMv5TE, ARMv5TEJ, and ARMv6

The following architecture variants are now OBSOLETE: ARMv1, ARMv2, ARMv2a, ARMv3, ARMv3G, ARMv3M, ARMv4xM, ARMv4TxM, ARMv5, ARMv5xM, and ARMv5TxM.

Version 4 and the introduction of Thumb (T variant):

The Thumb instruction set is a re-encoded subset of the ARM instruction set. Thumb instructions execute in their own processor state, with the architecture defining the mechanisms required to transition between ARM and Thumb states. The key difference is that Thumb instructions are half the size of ARM instructions (16 bits compared with 32 bits).

- Thumb code usually uses more instructions for a given task, making ARM code best for maximizing performance of time-critical code.
- ARM state and some associated ARM instructions are required for exception handling.

New features in Version 5T:

- Improved efficiency of ARM/Thumb interworking.
- Count leading zeros (ARM only) and software breakpoint (ARM and Thumb) instructions added
- Additional options for coprocessor designers (coprocessor support is ARM only)
- Tighter definition of flag setting on multiplies (ARM and Thumb).
- Introduction of the E variant, adding ARM instructions which enhance performance of an ARM processor on typical digital signal processing (DSP) algorithms:
    - Several multiply and multiply-accumulate instructions that act on 16-bit data items.
    - Addition and subtraction instructions that perform saturated signed arithmetic. Saturated arithmetic produces the maximum positive or negative value instead of wrapping the result if the calculation overflows the normal integer range.

New features in Version 6:

The following ARM instructions are added from the older version to an improved version:

- CPS, SRS and RFE instructions for improved exception handling
- REV, REV16 and REVSH byte reversal instructions
- SETEND for a revised endian (memory) model
- LDREX and STREX exclusive access instructions
- SXTB, SXTH, UXTB, UXTH byte/half word extend instructions
- A set of Single Instruction Multiple Data (SIMD) media instructions
- Additional forms of multiply instructions with accumulation into a 64-bit result.

Q2. Explain ARM Architecture. (DEC 21)/Functional Blocks of ARM processor (Dec 2022/Jan 2023)

3. ARM Architecture (ARM7 TDMI):

- T: Thumb, 16-bit instruction set
- D: on-chip Debug support, enabling the processor to halt in response to a debug request
- M: enhanced Multiplier, yield a full 64-bit result, high performance
- I: Embedded ICE hardware

The ARM architecture supports two basic types of data:

- The standard ARM word is 32 bits long.
- The word may be divided into four 8-bit bytes.

ARM7 consists of

- 1 dedicated program counter
- 1 dedicated current program status register
- 5 dedicated saved program status registers
- 30 general purpose registers

The architecture of ARM7 is given below.

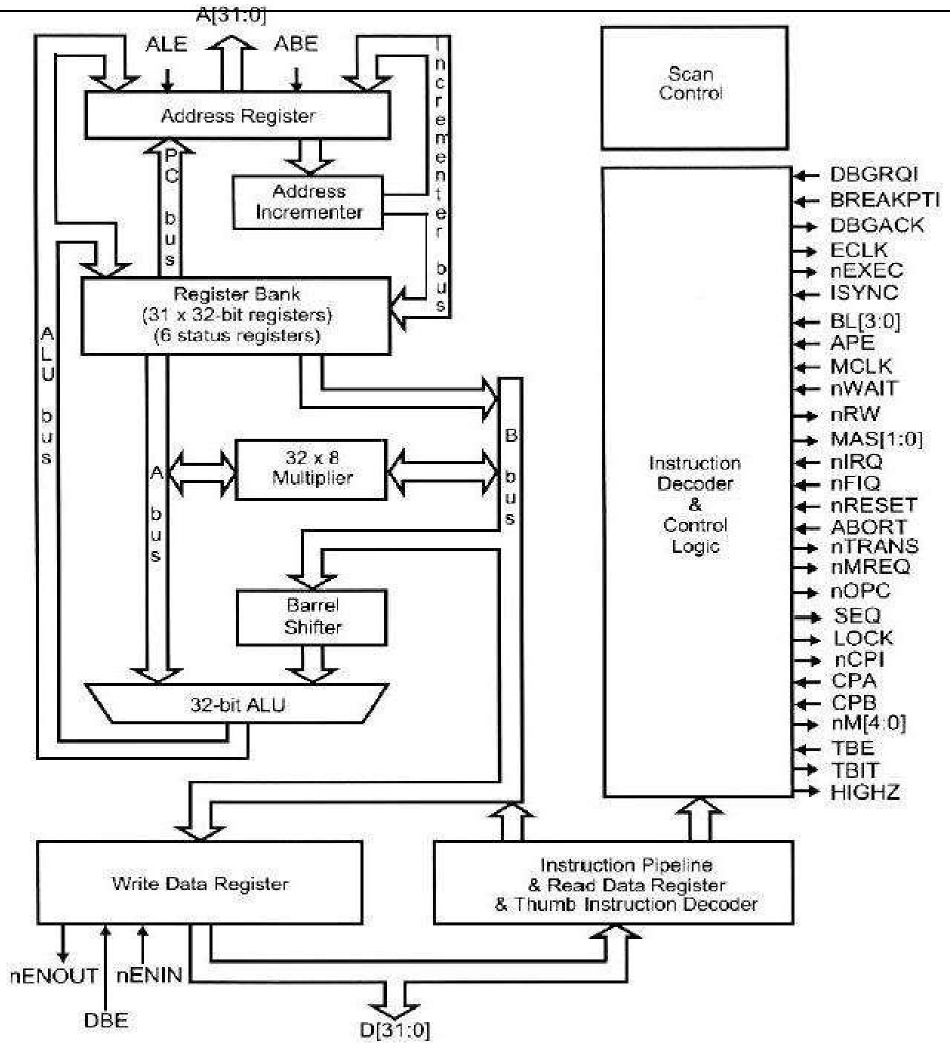


Fig: Architecture of ARM7 Processor.

The current processor mode governs which of several banks is accessible. Each mode can access

- a particular set of r0-r12 registers
- a particular r13 (the stack pointer, sp) and r14 (the link register)
- the program counter, r15 (pc)
- the current program status register, CPSR
- Privileged modes (except System) can also access
- a particular SPSR(saved program status register)

Features of ARM Processor:

The ARM processors provide advanced features for a variety of applications.

- Several extensions provide improved digital signal processing.
- Saturation arithmetic can be performed with no overhead.
- A new instruction is used for arithmetic normalization.
- Multimedia operations are supported by single instruction multiple data operations.
- A separate monitor mode allows the processor to enter a secure world to perform operations not permitted in normal mode.

Q3. Compare ARM processor with other processors.

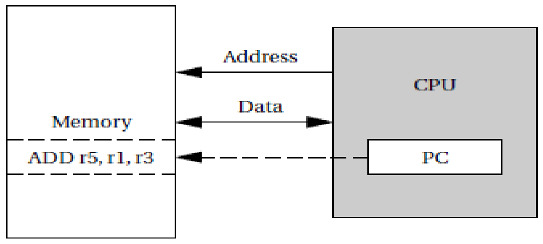
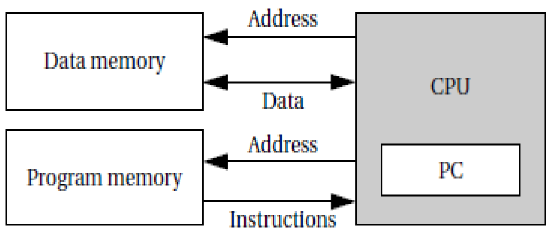
ARM Processor vs other Processors:

- ARM is designed on RISC Architecture, hence it is having reduced instructions, where Intel and AMD are designed on x86 CISC architecture.
- The core difference between these is ARM instructions operates only on registers with a few instructions for loading and saving data from or to memory while x86 can operate on directly memory as well.
- ARM is a simpler architecture, leading to small silicon area and lots of power save features while x86 becoming a power beast in terms of both power consumption and production.

The ARM is a Reduced Instruction Set Computer (RISC), as it incorporates these typical RISC architecture features:

- o A large uniform register file.
- o A load/store architecture, where data-processing operations only operate on register contents, not directly on memory contents.
- o Uniform and fixed-length instruction fields, to simplify instruction decode.

Harvard architecture Vs Von-Neumann architecture:

Von-Neumann architecture	**Harvard architecture**
	 Data memory Program memory Address Data Address Instructions CPU PC
The memory holds both data and instructions, and can be read or written whengiven an address..	Harvard architecture is like the von Neumann architecture. Harvard machine has separate memories for data and program.

Q4. Explain various registers in ARM.

ARM Registers:

ARM has 31 general-purpose 32-bit registers. At any one time, 16 of these registers are visible. The other registers are used to speed up exception processing. All the register specifiers in ARM instructions can address any of the 16 visible registers.

The main bank of 16 registers is used by all unprivileged code. These are the User mode registers. User mode is different from all other modes as it is unprivileged, which means:

- User mode can only switch to another processor mode by generating an exception. The SWI instruction provides this facility from program control.
- Memory systems and coprocessors might allow User mode less access to memory and coprocessor functionality than a privileged mode.

Three of the 16 visible registers have special roles:

Stack pointer:

Software normally uses R13 as a Stack Pointer (SP). R13 is used by the PUSH and POP instructions in T variants, and by the SRS and RFE instructions from ARMv6.

Link register:

Register 14 is the Link Register (LR). This register holds the address of the next instruction after a Branch and Link (BL or BLX) instruction, which is the instruction used to make a subroutine call. It is also used for return address information on entry to exception modes. At all other times, R14 can be used as a general-purpose register.

Program counter:

Register 15 is the Program Counter (PC). It can be used in most instructions as a pointer to the instruction which is two instructions after the instruction being executed. In ARM state, all ARM instructions are four bytes long (one 32-bit word) and are always aligned on a word boundary.

Supervisor mode:

•Supervisor mode is an execution mode on some processors which enables the execution of all instructions, including privileged instructions.

•It is thus capable of executing both input/output operations and privileged operations. The operating system of a computer usually operates in this mode.

•Supervisor mode helps in preventing applications from corrupting the data of the operating system.

•The ARM instruction that puts the CPU in supervisor mode is called SWI.(SWI CODE_1)

•The argument to SW1 is a 24-bit immediate value that is passed on to the supervisor mode code; it allows the program to request various services from the supervisor mode.

•In supervisor mode control programs are executed

Exceptions:

An exception (or exceptional event) is a problem that arises during the execution of a program. It is an internally detected error. When an Exception occurs the normal flow of the program is disrupted and the program/Application terminates abnormally, therefore these exceptions are to be handled.

A simple example is division by zero. One way to handle this problem would be to check every divisor before division to be sure it is not zero, but this would both substantially increase the size of numerical programs and cost a great deal of CPU time evaluating the divisor's value.

ARM supports seven types of exception, and a privileged processing mode for each type. The seven types of exception are:

- reset
- attempted execution of an Undefined instruction
- software interrupt (SWI) instructions, can be used to make a call to an operating system
- Prefetch Abort, an instruction fetch memory abort
- Data Abort, a data access memory abort
- IRQ, normal interrupt
- FIQ, fast interrupt.

Traps:

- A trap, also known as a software interrupt, is an instruction that explicitly generates an exception condition. The most common use of a trap is to enter supervisor mode. The entry into supervisor mode must be controlled to maintain security, if the interface between user and supervisor mode is improperly designed, a user program may be able to inform code into the supervisor mode that could be executed to perform harmful operations.

Status registers:

All processor state other than the general-purpose register contents is held in status registers.

Current program status register (CPSR) is the register which is set automatically during every arithmetic, logical, or shifting operation. The top four bits of the CPSR hold the following useful information about the results of that arithmetic/logical operation:

- The negative (N) bit is set when the result is negative in two's-complement arithmetic.
- The zero (Z) bit is set when every bit of the result is zero.
- The carry (C) bit is set when there is a carry out of the operation.
- The overflow (V) bit is set when an arithmetic operation results in an overflow.

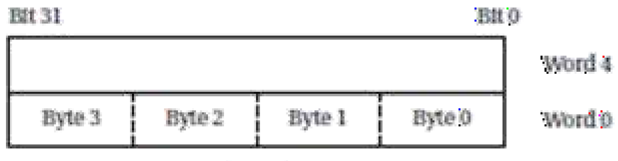
CISC Vs RISC:

Complex instruction set computers (CISC) provides a variety of instructions that may perform very complex tasks, such as string searching; they also generally used a number of different instruction formats of varying lengths.

Reduced instruction set computers (RISC) tends to provide somewhat fewer and simpler instructions. The instructions were also chosen so that they could be efficiently executed in pipelined processors. RISC techniques can also use efficiently to execute at least a common subset of CISC instruction sets, so the performance gap between RISC-like and CISC-like instruction sets has somewhat narrowed.

Little Endian mode:

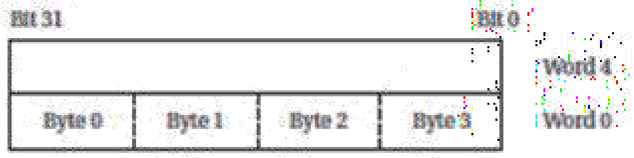
Little-Endian mode is the one in which the lower byte addresses are used for the least significant (right most) bytes of the word. Lower addresses contain lower order bytes of data.



Little-endian

Big Endian mode:

Big-endian mode little-endian mode is the one in which word is given the same address as that of the most significant (left most) byte of the word. High bytes of data are stored at lower addresses and low bytes of data at high addresses.



Big-endian

Instruction Set Preliminaries:

Computer Architecture Taxonomy:

The computing system consists of a central processing unit (CPU) and a memory. The memory holds both data and instructions, and can be read or written when given an address. A computer whose memory holds both data and instructions is known as a von Neumann machine.

An alternative to the von Neumann style of organizing computers is the Harvard architecture, which is nearly as old as the von Neumann .

Assembly Language:

Assembly language follows this relatively structured form to make it easy for the assembler to parse the program and to consider most aspects of the program line by line.

One instruction appears per line.

- Labels, which give names to memory locations, start in the first column.
- Instructions must start in the second column or after to distinguish them from labels.
- Comments run from some designated comment character (; in the case of ARM) to the end of the line.

Q5. Discuss different addressing modes in ARM processor.

Addressing:

Base-plus-offset addressing in ARM processor:

The ARM also supports several forms of base-plus-offset addressing, which is related to indirect addressing. But rather than using a register value directly as an address, the register value is added to another value to form the address. For instance,

LDR r0, [r1,#16] loads r0 with the value stored at location r1_16.

Here, r1 is referred to as the base and the immediate value the offset. When the offset is an immediate, it may have any value up to 4,096; another register may also be used as the offset.

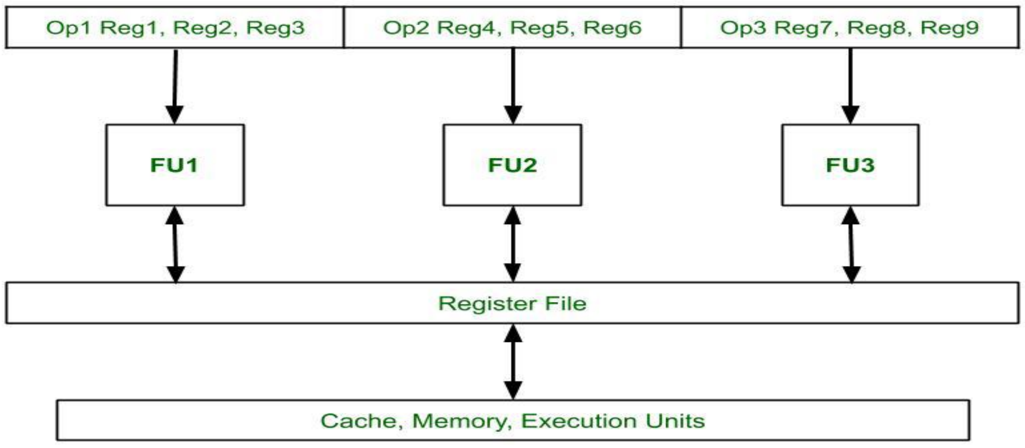
The base-plus-offset addressing mode has two other variations: auto-indexing and post-indexing. Auto-indexing updates the base register, such that

Eg: LDR r0,[r1,#16]!

LDR r0, [r1],#16

VLIW vs Superscalar:

VLIW architectures are distinct from traditional RISC and CISC architectures implemented in current mass-market microprocessors. It is important to distinguish instruction-set architecture—the processor programming model—from implementation— the physical chip and its characteristics.

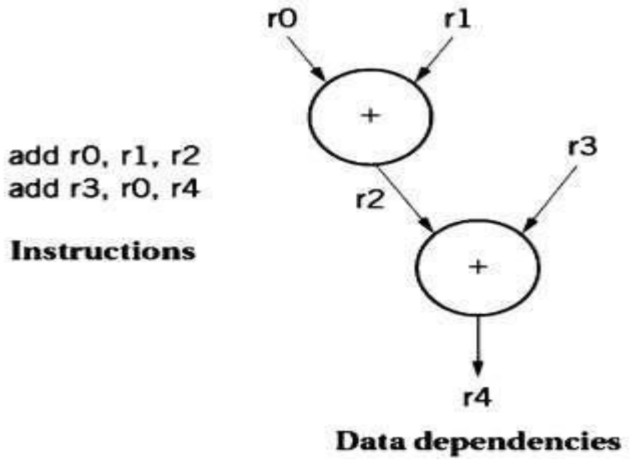


Very long instruction word (VLIW) processors are most likely used by digital signal, processing systems. These Processors rely on the compiler to identify sets of instructions that can be executed in parallel. The efficiencies of the VLIW processors can more easily be leveraged by digital signal processing software. VLIW processors consume less power and are very smaller than superscalar processors. Since it is very small, it is used in signal processing and multimedia applications.

Superscalar Processor:

A single-issue processor executes one instruction at a time. Although it may have several instructions at different stages of execution, only one can be at any particular stage of execution. Several other types of processors allow multiple-issue instruction. A superscalar processor uses specialized logic to identify at run time instructions that can be executed simultaneously. Superscalar processors often use too much energy and are too expensive for widespread use in embedded systems.

A **data dependency** is a relationship between the data operated on by instructions. In the example of Figure given below, the first instruction writes into r0 while the second instruction reads from it. As a result, the first instruction must finish before the second instruction can perform its addition. The data dependency graph shows the order in which these operations must be performed.



Q6. Discuss Instruction sets of ARM Processor (Dec20, Apr21)

General Instruction Set of an ARM assembly language (Nov/Dec 2023)

4. ARM Instruction Set:

The instruction set of the computer defines the interface between software modules and the underlying hardware; the instructions define what the hardware will do under certain circumstances. Instructions can have a variety of characteristics, including:

- Fixed versus variable length;
- Addressing modes;
- Numbers of operands;
- Types of operations supported.

The ARM instruction set can be divided into six broad classes of instruction:

- Branch instructions
- Data-processing instructions on page
- Status register transfer instructions on page
- Load and store instructions on page
- Coprocessor instructions on page
- Exception-generating instructions on page.

Branch Instructions:

The B (branch) instruction is the basic mechanism in ARM for changing the flow of control. The address that is the destination of the branch is often called the branch target.

Branches are PC-relative— the branch specifies the offset from the current PC value to the branch target. The offset is in words, but because the ARM is byte addressable, the offset is multiplied by four (shifted left two bits, actually) to form a byte address. Thus, the instruction B #100 will add 400 to the current PC value.

There are also branch instructions which can switch instruction set, so that execution continues at the branch target using the Thumb instruction set. Thumb support allows ARM code to call Thumb subroutines, and ARM subroutines to return to a Thumb caller. Similar instructions in the Thumb instruction set allow the corresponding Thumb → ARM switches.

Data-processing instructions on page

The data-processing instructions perform calculations on the general-purpose
Registers. There are five types of data-processing
instructions:

- Arithmetic/logic instructions
- Comparison instructions
- Single Instruction Multiple Data (SIMD) instructions
- Multiply instructions on page
- Miscellaneous Data Processing instructions on page.

Arithmetic/logic instructions:

The arithmetic or logical instructions is used to perform arithmetic or logical operations. The source operands used here is two and the result is made to store at the destination register. Based on the result the code flag condition will get updated. Of the two source operands:

- One is always a register
- The other has two basic forms:
  - an immediate value
  - a register value, optionally shifted.

The arithmetic operations perform addition and subtraction; the with-carry versions include the current value of the carry bit in the computation.

ADD r0, r1, r2

This instruction sets register r0 to the sum of the values stored in r1 and r2.
In addition to specifying registers as sources for operands, instructions may also provide immediate operands, which encode a constant value directly in the instruction. For example,

ADD r0, r1,#2 sets r0 to r1+2.

Comparison instructions:

The comparison instructions use the same instruction format as the arithmetic/logic instructions. These perform an arithmetic or logical operation on two source operands, but do not write the result to a register. They always update the condition flags, based on the result. The source operands of comparison instructions take the same forms as those of arithmetic/logic instructions, including the ability to incorporate a shift operation.

Comparison of Instruction and MOVE instruction in ARM processor:

The compare instruction

CMP r0, r1 computes r0 – r1, sets the status bits, and throws away the result of the subtraction.

CMN uses an addition to set the status bits. TST performs a bit-wise AND on the operands, while TEQ performs an exclusive-OR.

The instruction

MOV r0, r1 sets the value of r0 to the current value of r1.

The MVN instruction complements the operand bits (one's complement) during themove.

Single Instruction Multiple Data (SIMD) instructions:

The add and subtract instructions treat each operand as two parallel 16-bit numbers, or four parallel 8-bit numbers. They can be treated as signed or unsigned. The operations can optionally be saturating, wrap around, or the results can be halved to avoid overflow. These instructions are available in ARMv6.

Multiply instructions on page:

There are several classes of multiply instructions, introduced at different times intothe architecture.

Miscellaneous Data Processing instructions on page:

These include Count Leading Zeros (CLZ) and Unsigned Sum of Absolute Differences with optional Accumulate (USAD8 and USADA8).

Status register transfer instructions:

The status register transfer instructions transfer the contents of the CPSR or an SPSR to or from a general-purpose register. Writing to the CPSR can:

- Set the values of the condition code flags
- Set the values of the interrupt enable bits
- Set the processor mode and state
- Alter the endianness of Load and Store operations.

Load and store instructions:

Load-Store instructions are used to transfer the Values between registers and memory.

LDRB and STRB load and store bytes rather than whole words, while LDRH and SDRH operate on half-words and LDRSH extends the sign bit on loading.

For example: LDR r0,[r1],STR r0,[r1]

An ARM address may be 32 bits long. The ARM load and store instructions do not directly refer to main memory addresses, since a 32-bit address would not fit into an instruction that included an opcode and operands. Instead it uses Register Indirect Addressing.

The following load and store instructions are available:

- Load and Store Register
- Load and Store Multiple registers on page
- Load and Store Register Exclusive on page.

There are also swap and swap byte instructions, but their use is deprecated in ARMv6. It is recommended that all software migrates to using the load and store register exclusive instructions.

Load and Store Register:

Load Register instructions can load a 64-bit double word, a 32-bit word, a 16-bit half word, or an 8-bit byte from memory into a register or registers. Byte and half word loads can be automatically zero-extended or sign-extended as they are loaded. Store Register instructions can store a 64-bit double word, a 32-bit word, a 16-bit half word, or an 8-bit byte from a register or registers to memory.

Load and Store Register instructions have three primary addressing modes, all of which use a base register and an offset specified by the instruction:

• In **offset addressing**, the memory address is formed by adding or subtracting an offset to or from the base register value.

• In **pre-indexed addressing**, the memory address is formed in the same way as for offset addressing. As a side effect, the memory address is also written back to the base register.

• In **post-indexed addressing**, the memory address is the base register value. As a side effect, an offset is added to or subtracted from the base register value and the result is written back to the base register.

Load and Store Multiple registers :

Load Multiple (LDM) and Store Multiple (STM) instructions perform a block transfer of any number of the general-purpose registers to or from memory. Four addressing modes are provided:

- Pre-increment
- Post-increment
- Pre-decrement
- Post-decrement.

Load and Store Register Exclusive:

These instructions support cooperative memory synchronization. They are designed to provide the atomic behaviour required for semaphores without locking all system resources between the load and store phases.

Register indirect Addressing:

Register indirect addressing is defined as that the address which will be used by the instruction (known as the "effective address") is taken from the contents of a register, rather than being encoded directly within the instruction itself (which is "absolute" addressing).Here the value stored in the register is used as the address to be fetched from memory; the result of that fetch is the desired operand value.

Coprocessor instructions:

There are three types of coprocessor instructions:

Data-processing instructions

These start a coprocessor-specific internal operation.

Data transfer instructions

These transfer coprocessor data to or from memory. The address of the transfer is calculated by the ARM processor.

Register transfer instructions

These allow a coprocessor value to be transferred to or from an ARM register, or apair of ARM registers.

Exception-generating instructions:

Two types of instruction are designed to cause specific exceptions to occur.

Software interrupt instructions

SWI instructions cause a software interrupt exception to occur. These are normally
used to make calls to an operating system, to request an OS-defined service. The exception entry caused by a SWI instruction also changes to a privileged processor mode. This allows an unprivileged task to gain access to privileged functions, but only in ways permitted by the OS.

Software breakpoint instructions

BKPT instructions cause an abort exception to occur. If suitable debugger software is
installed on the abort vector, an abort exception generated in this fashion is treated as a breakpoint. If debug hardware is present in the system, it can instead treat a BKPT instruction directly as a breakpoint, preventing the abort exception from occurring.

Additional Instructions in ARM Processor:

4.7.1 Function of RSB instruction and BIC instruction:

RSB performs a subtraction with the order of the two operands reversed, so that

RSB r0, r1, r2 sets r0 to be r2-r1.

The bit-wise logical operations perform logical AND, OR, and XOR operations (the exclusive or is called EOR). The BIC instruction stands for bit clear:

BIC r0, r1, r2 sets r0 to r1 and not r2.

This instruction uses the second source operand as a mask: Where a bit in the mask is 1, the corresponding bit in the first source operand is cleared.

MUL instruction and MLA instructions in ARM Processor:

The MUL instruction multiplies two values, but with some restrictions: No operand may be an immediate, and the two source operands must be different registers. The MLA instruction performs a multiply-accumulate operation, particularly useful in matrix operations and signal processing. The instruction is

MLA r0, r1, r2, r3 sets r0 to the value r1 x r2 x r3.

Shift operations in ARM Processor:

The shift operations are not separate instructions—rather; shifts can be applied to arithmetic and logical instructions. The shift modifier is always applied to the second source operand. A left shift moves bits up toward the most-significant bits, while a right shift moves bits down to the least-significant bit in the word. The LSL and LSR modifiers perform left and right logical shifts, filling the least-significant bits of the operand with zeroes. The arithmetic shift left is equivalent to an LSL, but the ASR copies the sign bit—if the sign is 0, a 0 is copied, while if the sign is 1, a 1 is copied. The rotate modifiers always rotate right, moving the bits that fall off the least-significant bit up to the most-significant bit in the word.

Operation of the BL instructions:

All these instructions cause a branch to the address indicated by label or contained in the register specified by Rm. In addition:

The BL and BLX instructions write the address of the next instruction to LR, the link register R14.

The BX and BLX instructions result in a Hard Fault exception if bit[0] of Rm is 0.

BL and BLX instructions also set bit [0] of the LR to 1. This ensures that the value is suitable for use by a subsequent POP {PC} or BX instruction to perform a successful return branch.

Examples

```
B       loop A      ; Branch to loop A
BL      funC        ; Branch with link (Call) to function funC, return
address; stored in LR
BX      LR          ; Return from function call
BLX     R0          ;Branch with link and exchange (Call) to a address
stored ; in R0
BEQ     labelD      ; Conditionally branch to labelD if last flag setting ;
instruction set the Z flag, else do not branch.
```

Q7. Explain stacks and subroutines in ARM Processor (Apr21)

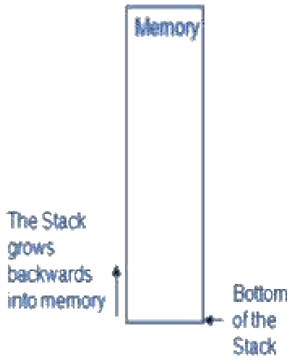
Stacks and Subroutines:

Stack:

- The stack is an area of memory identified by the programmer for temporary storage of information.
- The stack is a LIFO (Last in First Out.) structure.
- The stack normally grows backwards into memory.
- In other words, the programmer defines the bottom of the stack and the stack grows up into reducing address range.
- Given that the stack grows backwards into memory, it is customary to place the bottom of the stack at the end of memory to keep it as far away from user programs as possible.
- In the 8085, the stack is defined by setting the SP (Stack Pointer) register.

LXI SP, FFFFH.

- This sets the Stack Pointer to location FFFFH (end of memory for the 8085).

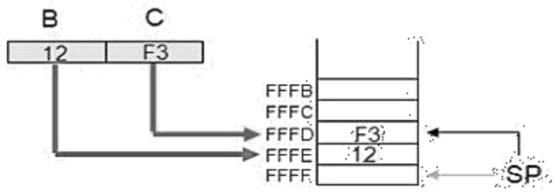


Saving Information on the stack:

- Information is saved on the stack by PUSHing it on.
- It is retrieved from the stack by POPing it off.
- The 8085 provides two instructions: PUSH and POP for storing information on the stack and retrieving it back.
- Both PUSH and POP work with register pairs ONLY.

The PUSH Instruction:

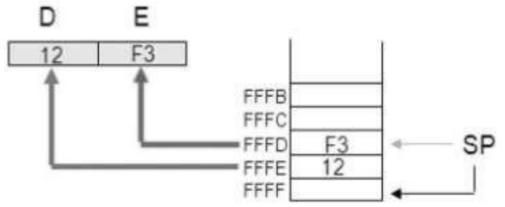
- PUSH B/D/H/PSW
    - o Decrement SP
    - o Copy the contents of register B to the memory location pointed to by SP
    - o Decrement SP
    - o Copy the contents of register C to the memory location pointed to by SP.



The POP Instruction:

- POP B/D/H/PSW
  - o Copy the contents of the memory location pointed to by the SP to register E
  - o Increment SP
  - o Copy the contents of the memory location pointed to by the SP to register D

  - o Increment SP



Operation of the Stack:

- During pushing, the stack operates in a ,decrement then store` style.
- The stack pointer is decremented first, then the information is placed on the stack.
- During poping, the stack operates in a ,use then increment` style.
- The information is retrieved from the top of the stack and then the pointer is incremented.
- The SP pointer always points to ,the top of the stack`.

LIFO

- The order of PUSHs and POPs must be opposite of each other in order to retrieve information back into its original location.
  PUSH B
  PUSH D
  ...
  POP D
  POP B
- Reversing the order of the POP instructions will result in the exchange of the contents of BC and DE.

The PSW Register Pair:

- The 8085 recognizes one additional register pair called the PSW (Program Status Word).
- This register pair is made up of the Accumulator and the Flags registers.
  - It is possible to push the PSW onto the stack, do whatever operations are needed, then POP it off of the stack.
  - The result is that the contents of the Accumulator and the status of the Flags are returned to what they were before the operations were executed.

Cautions with PUSH and POP:

- PUSH and POP should be used in opposite order.
- There has to be as many POP's as there are PUSH's.
- If not, the RET statement will pick up the wrong information from the top of the stack and the program will fail.
- It is not advisable to place PUSH or POP inside a loop.

Program to Reset and display Flags:

- Clear all Flags.
- Load 00H in the accumulator, and demonstrate that the zero flag is not affected by data transfer instruction.
- Logically OR the accumulator with itself to set the Zero flag, and display the flag at PORT1 or store all flags on the stack.

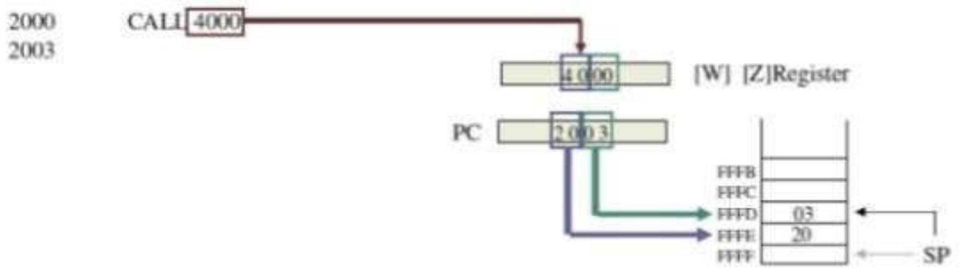
○ XX00	LXI SP, XX99H	Initialize the stack
○ 03	MVI L, 00H	Clear L
○ 05	PUSH H	Place (L) on stack
○ 06	POP PSW	Clear Flags
○ 07	MVI A, 00H	Load 00H
○ 09	PUSH PSW	Save Flags on stack
○ 0A	POP H	Retrieve flags in L
○ 0B	MOV A, L	
○ 0C	OUT PORT0	Display Flags (00H)
○ 0E	MVI A, 00H	Load 00H Again
○ XX10	ORA A	Set Flags and reset CY, AC
○ 11	PUSH PSW	Save Flags on Stack
○ 12	POP H	Retrieve Flags in L
○ 13	MOV A, L	
○ 14	ANI 40H	Mask all Flags except Z
○ 16	OUT PORT1	Displays 40H
○ 18	HLT	End of Program

Subroutine:

- A subroutine is a group of instructions that will be used repeatedly in different locations of the program.
- Rather than repeat the same instructions several times, they can be grouped into a subroutine that is called from the different locations.
- In Assembly language, a subroutine can exist anywhere in the code.
- However, it is customary to place subroutines separately from the main program.
- The 8085 has two instructions for dealing with subroutines.
- The **CALL** instruction is used to redirect program execution to the subroutine.
- The **RTE** instruction is used to return the execution to the calling routine.

The CALL instruction:

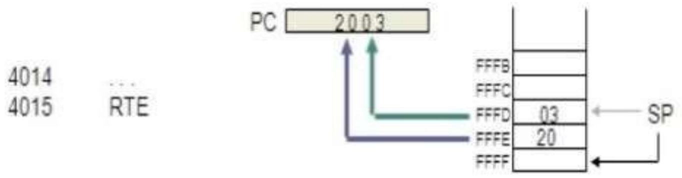
- **CALL 4000H**
  - 3-byte instruction.
  - Push the address of the instruction immediately following the CALL onto the stack and decrement the stack pointer register by two.
  - Load the program counter with the 16-bit address supplied with the CALL instruction.
  - Jump unconditionally to memory location.



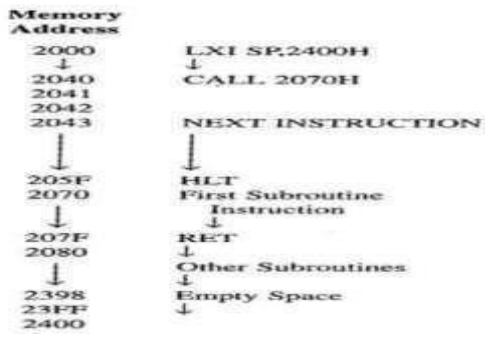
- MP reads the subroutine address from the next two memory location and stores the higher order 8bit of the address in the W register and stores the lower order 8bit of the address in the Z register.
- Push the address of the instruction immediately following the CALL onto the stack(Return address)
- Loads the program counter with the 16-bit address supplied with the CALL instruction from WZ register.

The RTE Instruction

- **RTE**
  - 1-byte instruction
  - Retrieve the return address from the top of the stack and increments stack pointer register by two.
  - Load the program counter with the return address.
  - Unconditionally returns from a subroutine.



Illustrates the exchange of information between stack and Program Coun



```
Memory
Address
 2000        LXI SP,2400H
  ↓          ↓
 2040        CALL 2070H
 2041
 2042
 2043        NEXT INSTRUCTION
  ↓          ↓
 205F        HLT
 2070        First Subroutine
                Instruction
  ↓               ↓
 207F        RET
 2080        ↓
  ↓          Other Subroutines
             ↓
 2398        Empty Space
 23FF        ↓
 2400
```

Passing Data to a Subroutine:

- In Assembly Language data is passed to a subroutine through registers.
- The data is stored in one of the registers by the calling program and the subroutine uses the value from the register.
- The other possibility is to use agreed upon memory locations.
- The calling program stores the data in the memory location and the subroutine retrieves the data from the location and uses it.

MODELS OF PROGRAMS:

Write short notes on Data flow graph (November 2008)

Data Flow Graphs:

- A ***data flow graph*** is a model of a program with no conditionals. In a high-level programming language, a code segment with no conditionals—more precisely, with only one entry and exit point is known as a basic block.
- As the C code is executed, we would enter this basic block at the beginning and execute all the statements.

```
w = a+b;
x = a-c;
y = x+d;
x = a+c;
z = y+e;
```

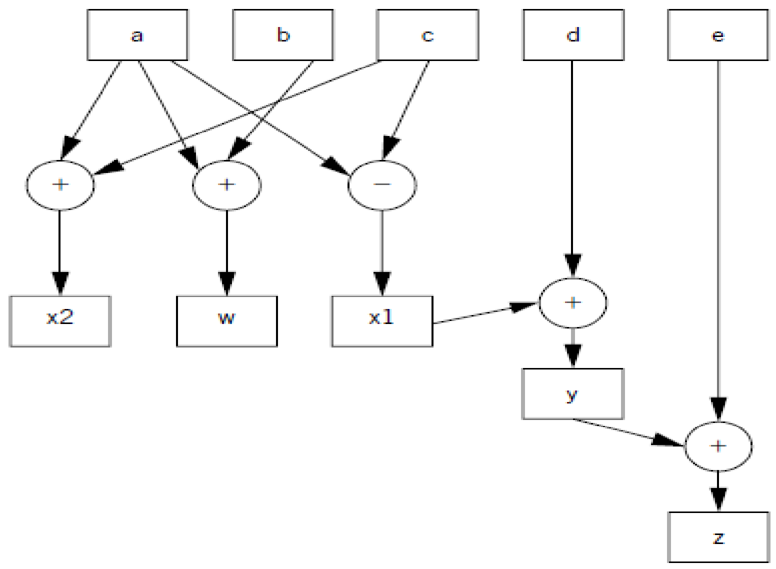
A basic block in C.

```
w = a+b;
x = a-c;
y = x1+d;
x2= a+c;
z = y+e;
```

The basic block in single-assignment form.

- Before drawing the data flow graph for this code, we need to modify it slightly. There are two assignments to the variable *x*—it appears twice on the left side of an assignment. The code is rewritten in ***single-assignment form***, in which a variable appears only once on the left side.
- Since the specification is C code, we assume that the statements are executed sequentially, so that any use of a variable refers to its latest assigned value.
- In this case, *x* is not reused in this block (presumably it is used elsewhere), so just eliminate the multiple assignments to *x*.
- The single-assignment form is important because it allows us to identify a unique location in the code where each named location is computed.
- As an introduction to the data flow graph, we use two types of nodes in the graph round nodes denote operators and square nodes represent values.

- The value nodes may be either inputs to the basic block, such as *a* and *b*, or variables assigned to within the block, such as *w* and *x*1.



An extended data flow graph for our sample basic block.

- The data flow graph for our single-assignment code The single-assignment form means that the data flow graph is acyclic
- If we assigned to *x* multiple times, then the second assignment would form a cycle in the graph including *x* and the operators used to compute *x*.

3.9. 1 Explain Control Data Flow Graph.

Control/Data Flow Graphs

- A CDFG uses a data flow graph as an element, adding constructs to describe control.
- In a basic CDFG, we have two types of nodes: ***decision nodes*** and ***data flow nodes***. \
- A data flow node encapsulates a complete data flow graph to represent a basic block.
- We can use one type of decision node to describe all the types of control in a sequential program. (The jump/branch is, after all, the way we implement all those high-level control constructs.)

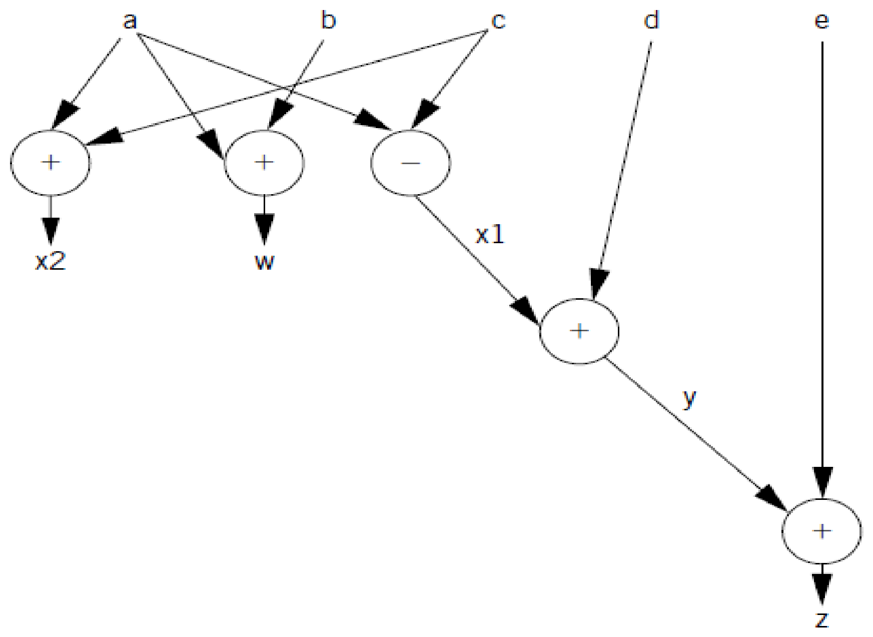


FIGURE 5.5

Standard data flow graph for our sample basic block.

- Figure 5.6 shows a bit of C code with control constructs and the CDFG constructed from it. The rectangular nodes in the graph represent the basic blocks.
- The basic blocks in the C code have been represented by function calls for simplicity. The diamond-shaped nodes represent the conditionals.
- The node's condition is given by the label, and the edges are labeled with the possible outcomes of evaluating the condition.

- Building a CDFG for a while loop is straightforward, as shown in Figure 5.7.
  - The while loop consists of both a test and a loop body, each of which we know how to represent in a CDFG. We can represent for loops by remembering that, in C, a for loop is defined in terms of a while loop.

The following for loop

```
for (i = 0; i < N; i++) {
        loop_body();
}
```

is equivalent to

```
i = 0;
while (i < N) {
        loop_body();
        i++;
}
if (cond1)
        basic_block_1( );
else
        basic_block_2();
```

```
basic_block_3( );
switch (test1) {
          case c1: basic_block_4( ); break;
          case c2: basic_block_5( ); break;
          case c3: basic_block_6( ): break;
}
```

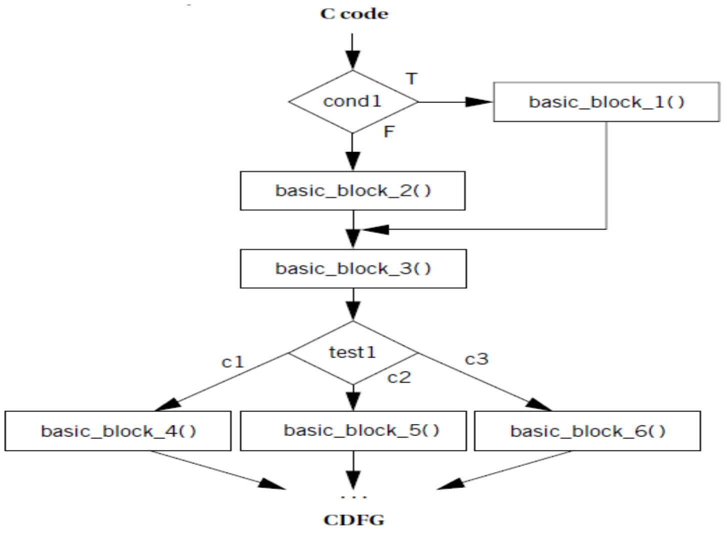
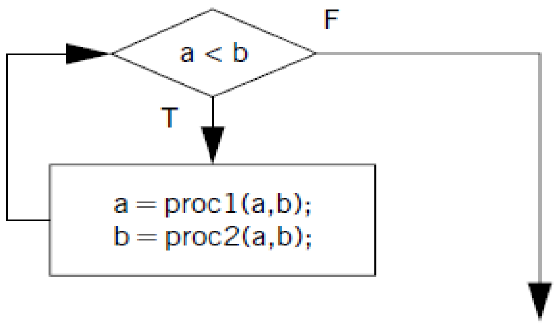


FIGURE 5.6
C code and its CDFG.

- For a complete CDFG model, we can use a data flow graph to model each data flow node. Thus, the CDFG is a hierarchical representation data flow CDFG can be expanded to reveal a complete data flow graph.
- An execution model for a CDFG is very much like the execution of the program it represents. The CDFG does not require explicit declaration of variables, but we assume that the implementation has sufficient memory for all the variables.

```
while (a < b) {
      a = proc1(a,b);
      b = proc2(a,b);
}
```

C code



CDFG

FIGURE 5.7
CDFG for a while loop.

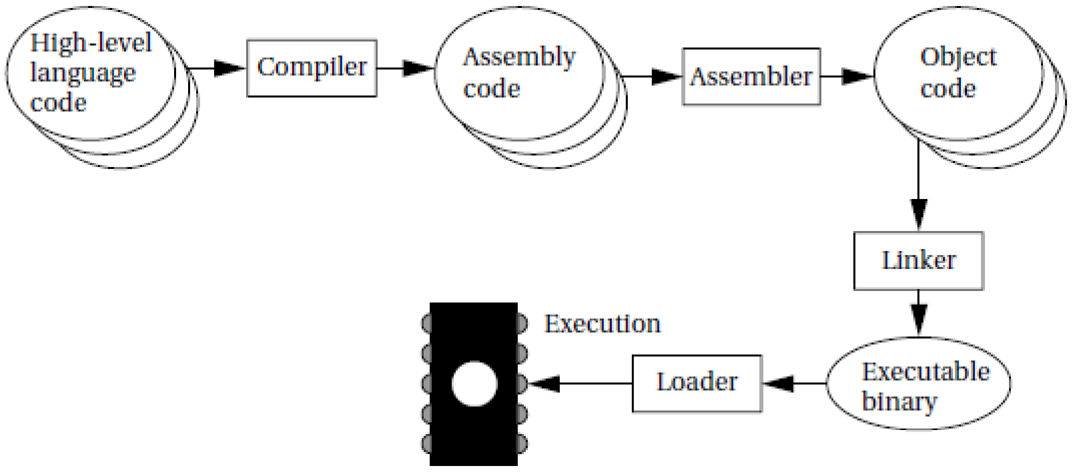
- As we execute the program, we either execute the data flow node or compute the decision in the decision node and follow the appropriate edge, depending on the type of node the program counter points on.

- Even though the data flow nodes may specify only a partial ordering on the data flow computations, the CDFG is a sequential representation of the program. There is only one program counter in our execution model of the CDFG, and operations are not executed in parallel.The CDFG is not necessarily tied to high-level language control structures. We can also build a CDFG for an assembly language program. A jump instruction corresponds to a nonlocal edge in the CDFG.

ASSEMBLY AND LINKING:

Write short notes on assembly and linking.(NOV/DEC 2008, May 2011)

- Assembly and linking are the last steps in the compilation process they turn a list of instructions into an image of the program's bits in memory.
- Loading actually puts the program in memory so that it can be executed.
- The compilation process is often hidden from us by compilation commands that do everything required to generate an executable program.
- As the figure shows, most compilers do not directly generate machine code, but instead create the instruction-level program in the form of human-readable assembly language.
- Generating assembly language rather than binary instructions frees the compiler writer from details extraneous to the compilation process, which includes the instruction format as well as the exact addresses of instructions and data.
- The assembler's job is to translate symbolic assembly language statements into bit-level representations of instructions known as ***object code***. T
- The assembler takes care of instruction formats and does part of the job of translating labels into addresses.
- However, since the program may be built from many files, the final steps in determining the addresses of instructions and data are performed by the linker, which produces an ***executable binary*** file.
- That file may not necessarily be located in the CPU's memory, however, unless the linker happens to create the executable directly in RAM. The program that brings the program into memory for execution is called a ***loader***.
- The simplest form of the assembler assumes that the starting address of the assembly language program has been specified by the programmer. The addresses in such a program are known as ***absolute addresses***.

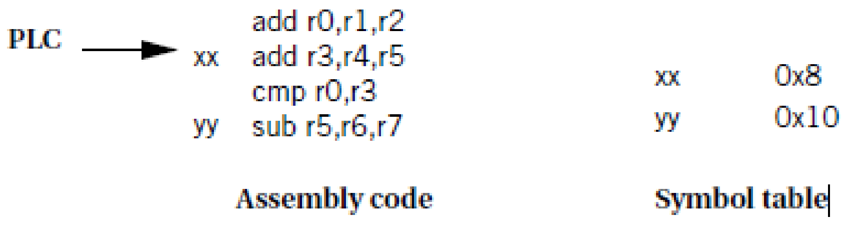


Program generation from compilation through loading.

3.9.3. Assemblers

- When translating assembly code into object code, the assembler must translate opcodes and format the bits in each instruction, and translate labels into addresses.
- Labels make the assembly process more complex, but they are the most important abstraction provided by the assembler.
- Labels let the programmer (a human programmer or a compiler generating assembly code) avoid worrying about the locations of instructions and data. Label processing requires making two passes through the assembly source code as follows:
- The first pass scans the code to determine the address of each label.
- The second pass assembles the instructions using the label values computed in the first pass.
- The name of each symbol and its address is stored in a ***symbol table*** that is built during the first pass. The symbol table is built by scanning from the first instruction to the last.
- During scanning, the current location in memory is kept in a ***program location counter (PLC)***.
- Despite the similarity in name to a program counter, the PLC is not used to execute the program, only to assign memory locations to labels.
- For example, the PLC always makes exactly one pass through the program, whereas the program counter makes many passes over code in a loop.
- Thus, at the start of the first pass, the PLC is set to the program’s starting address and the assembler looks at the first line.
- After examining the line, the assembler updates the PLC to the next location and looks at the next instruction.
- If the instruction begins with a label, a new entry is made in the symbol table, which includes the label name and its value. The value of the label is equal to the current value of the PLC.
- At the end of the first pass, the assembler rewinds to the beginning of the assembly language file to make the second pass.

- During a second pass when a label name is found, the label is looked up in the symbol table and its value substituted into the appropriate place in the instruction.



Symbol table processing during assembly.

- But how do we know the starting value of the PLC? The simplest case is absolute addressing.
- In this case, one of the first statements in the assembly language program is a pseudo-op that specifies the ***origin*** of the program, that is, the location of the first address in the program.
- A common name for this pseudo-op (e.g., the one used for the ARM) is the ORG statement, which puts the start of the program at location 2000.

```
ORG 2000
```

- This pseudo-op accomplishes this by setting the PLC's value to its argument's value, 2000 in this case.
- Assemblers generally allow a program to have many ORG statements in case instructions or data must be spread around various spots in memory.

3.9.4 Linking:

- Many assembly language programs are written as several smaller pieces rather than as a single large file.
- Breaking a large program into smaller files helps delineate program modularity.
- If the program uses library routines, those will already be preassembled, and assembly language source code for the libraries may not be available for purchase.
- A ***linker*** allows a program to be stitched together out of several smaller pieces.
- The linker operates on the object files created by the assembler and modifies the assembled code to make the necessary links between files.
- Some labels will be both defined and used in the same file. Other labels will be defined in a single file but used elsewhere.
- The place in the file where a label is defined is known as an ***entry point***. The place in the file where the label is used is called an ***external reference***.
- The main job of the loader is to *resolve* external references based on available entry points.
- As a result of the need to know how definitions and references connect, the assembler passes to the linker not only the object file but also the symbol table.

- Even if the entire symbol table is not kept for later debugging purposes, it must at least pass the entry points.
- External references are identified in the object code by their relative symbol identifiers.

```
label1   LDR r0,[r1]              label2    ADR var1
         ...                                ...
         ADR a                              B label3
         ...                                ...
         B label2                 x         %  1
         ...                      y         %  1
var1     %  1                     a         %  10
```

External references	Entry points
a	label1
label2	var1

File 1

External references	Entry points
var1	label2
label3	x
	y
	a

File 2

External references and entry points.

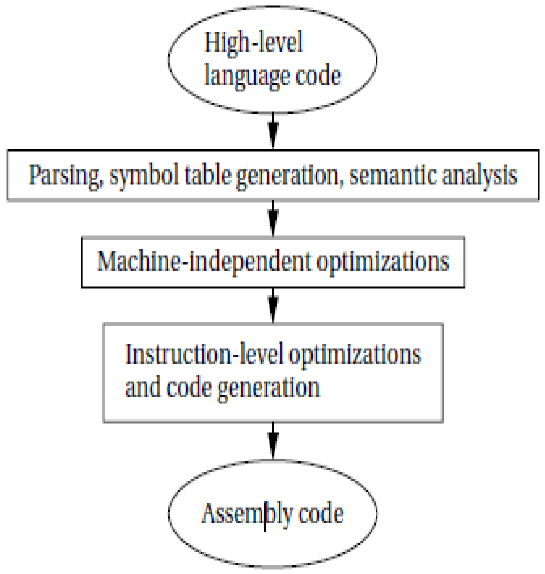
The linker proceeds in two phases.

- First, it determines the address of the start of each object file.
- The order in which object files are to be loaded is given by the user, either by specifying parameters when the loader is run or by creating a ***load map*** file that gives the order in which files are to be placed in memory.
- Given the order in which files are to be placed in memory and the length of each object file, it is easy to compute the starting address of each file.
- At the start of the second phase, the loader merges all symbol tables from the object files into a single, large table.
- It then edits the object files to change relative addresses into addresses. This is typically performed by having the assembler write extra bits into the object file to identify the instructions and fields that refer to labels.
- If a label cannot be found in the merged symbol table, it is undefined and an error message is sent to the user.

3.9.5. BASIC COMPILATION TECHNIQUE:

Discuss in detail about basic compilation technique. (DEC20, APR21, May 2023)

- It is useful to understand how a high-level language program is translated into instructions.
- Since implementing an embedded computing system often requires controlling the instruction sequences used to handle interrupts, placement of data and instructions in memory, and so forth, understanding how the compiler works can help you know when you cannot rely on the compiler.
- Next, because many applications are also performance sensitive, understanding how code is generated can help to meet the performance goals, either by writing high-level code that gets compiled into the instructions you want or by recognizing when we must write our own assembly code.


High-level language code
Parsing, symbol table generation, semantic analysis
Machine-independent optimizations
Instruction-level optimizations and code generation
Assembly code

The compilation process

- Compilation begins with high-level language. Simplifying arithmetic expressions is one example of a machine-independent optimization. Not all compilers do such optimizations, and compilers can vary widely regarding which combinations of machine-independent optimizations they do perform.
- Instruction-level optimizations are aimed at generating code.
- They may work directly on real instructions or on a pseudo-instruction format that is later mapped onto the instructions of the target CPU.
- This level of optimization also helps modularize the compiler by allowing code generation to create simpler code that is later optimized. For example, consider the following array access code:

```
x[i] = c*x[i];
```

- A simple code generator would generate the address for x[i] twice, once for each appearance in the statement.

- While in this simple case it would be possible to create a code generator that never generated the redundant expression, taking into account every such optimization at code generation time is very difficult.
- Better code and more reliable compilers are get by generating simple code first and then optimizing it.

3.9.6. Explain several techniques for optimizing software performance? (May 2023)

SOFTWARE PERFORMANCE OPTIMIZATION

a) Loop Optimizations

Loops are important targets for optimization because programs with loops tend to spend a lot of time executing those loops.

There are three important techniques in optimizing loops:

- *code motion*
- *induction variable elimination* and
- *Strength reduction*.

- ➢ Code motion lets us move unnecessary code out of a loop. If a computations result does not depend on operations performed in the loop body, then we can safely move it out of the loop.
- ➢ Code motion opportunities can arise because programmers may find some computations clearer and more concise when put in the loop body, even though they are not strictly dependent on the loop iterations.

A simple example of code motion is also common. **Consider the following loop:**

```
for (i = 0; i < N*M; i++)
{
        z[i] = a[i] + b[i];
}
```

- ➢ The code motion opportunity becomes more obvious when we draw the loop s CDFG as shown in Figure 5.23.
- ➢ The loop bound computation is performed on every iteration during the loop test, even though the result never changes.
- ➢ We can avoid *N* _*M* _1 unnecessary executions of this statement by moving it before the loop, as shown in the figure.

An ***induction variable*** is a variable whose value is derived from the loop iteration variables value.

- ➢ The compiler often introduces induction variables to help it implement the loop.
- ➢ Properly transformed, we may be able to eliminate some variables and apply strength reduction to others.
- ➢ A nested loop is a good example of the use of induction variables.

Here is a simple nested loop:

```
for (i = 0; i < N; i++)
        for (j = 0; j < M; j++)
                z[i][j] = b[i][j];
```

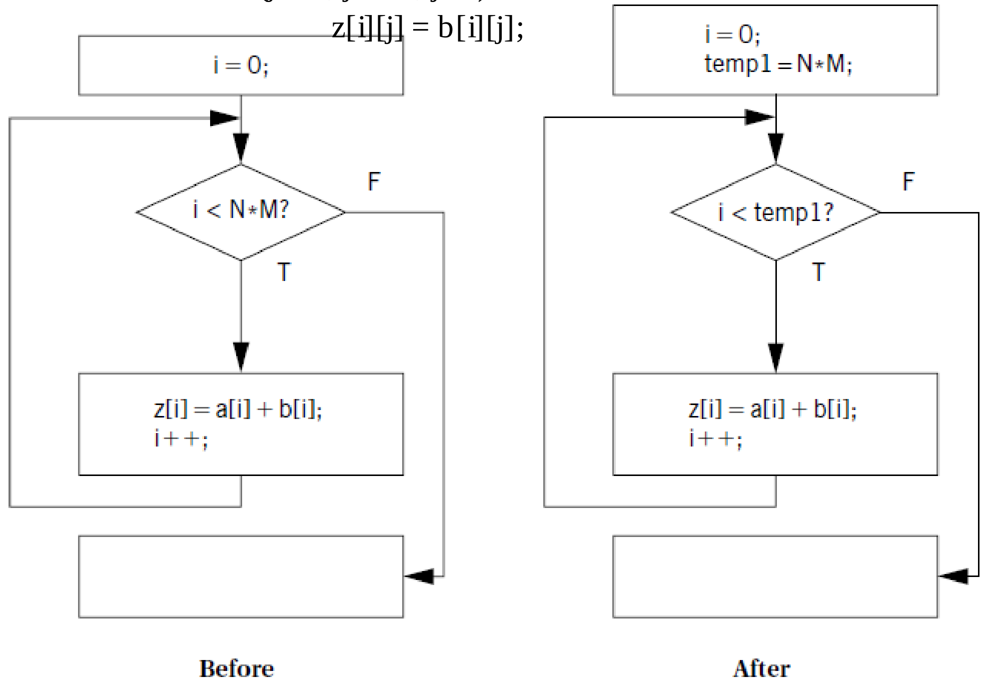


FIGURE 5.23

Code motion in a loop.

- ➢ The compiler uses induction variables to help it address the arrays.
- ➢ Let us rewrite the loop in C using induction variables and pointers.
- ➢ (Later, we use a common induction variable for the two arrays, even though the compiler would probably introduce separate induction variables and then merge them.)

```
for (i = 0; i < N; i++)
        for (j = 0; j < M; j++) {
                zbinduct = i*M + j;
              *(zptr + zbinduct) = *(bptr + zbinduct);
                }
```

➢ In the above code, zptr and bptr are pointers to the heads of the z and b arrays and zbinduct is the shared induction variable.➢ However, we do not need to compute zbinduct afresh each time. Since we are stepping through the arrays sequentially, we can simply add the update value to the induction variable:

```
zbinduct = 0;
for (i = 0; i < N; i++) {
        for (j = 0; j < M; j++) {
             *(zptr + zbinduct) = *(bptr + zbinduct);
               zbinduct++;
               }
          }
```

- This is a form of strength reduction since we have eliminated the multiplication from the induction variable computation.
- Strength reduction helps us reduce the cost of a loop iteration. Consider the following assignment:

y = x * 2;

- In integer arithmetic, we can use a left shift rather than a multiplication by 2 (as long as we properly keep track of overflows).
- If the shift is faster than the multiply, we probably want to perform the substitution.
- This optimization can often be used with induction variables because loops are often indexed with simple expressions.
- Strength reduction can often be performed with simple substitution rules since there are relatively few interactions between the possible substitutions.

b) Cache Optimizations

- A *loop nest* is a set of loops, one inside the other.
- Loop nests occur when we process arrays. A large body of techniques has been developed for optimizing loop nests.
- Rewriting a loop nest changes the order in which array elements are accessed. This can expose new parallelism opportunities that can be exploited by later stages of the compiler, and it can also improve cache performance.

Example 5.10

<u>Data realignment and array padding</u>

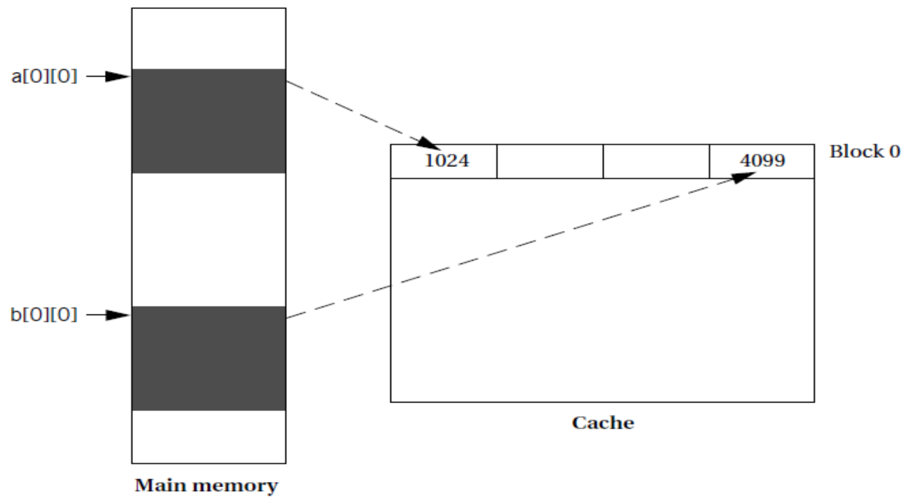
Assume we want to optimize the cache behavior of the following code:

```
for (j = 0; j < M; j++)
        for (i = 0; i < N; i++)
                a[j][i] = b[j][i] * c;
```

- Let us also assume a *and b* arrays are sized with *M* at 265 and *N* at 4 and a 256-line, four-way set-associative cache with four words per line.
- Even though this code does not reuse any data elements, cache conflicts can cause serious performance problems because they interfere with spatial reuse at the cache line level.
- Assume that the starting location for *a*[] is 1024 and the starting location for *b*[] is 4099. Although *a*[0][0] and *b*[0][0] do not map to the same word in the cache, they do map to the same block.

As a result, we see the following scenario in execution:

- The access to *a*[0][0] brings in the first four words of *a*[].
- The access to *b*[0][0] replaces *a*[0][0] through *a*[0][3] with *b*[0][3] and the contents of the three locations before *b*[].
- When *a*[0][1] is accessed, the same cache line is again replaced with the first four elements of *a*[].



Once the *a*[0][1] access brings that line into the cache, it remains there for the *a*[0][2] and *a*[0][3] accesses since the *b*[] accesses are now on the next line. However, the scenario repeats itself at *a*[1][0] and every four iterations of the cache.

- One way to eliminate the cache conflicts is to move one of the arrays. We do not have to move it far. If we move *b* start to 4100, we eliminate the cache conflicts.
- However, that fix won't work in more complex situations. Moving one array may only introduce cache conflicts with another array.
- In such cases, we can use another technique called padding. If we extend each of the rows of the arrays to have four elements rather than three, with the padding word placed at the beginning of the row, we eliminate the cache conflicts.
- In this case, *b* [0] [0] is located at 4100 by the padding. Although padding wastes memory, it substantially improves memory performance.
- In complex situations with multiple arrays and sophisticated access patterns, we have to use a combination of techniques relocating arrays and padding them to be able to minimize cache conflicts.

3.9.7. How to analyze programs to estimate their run times and examine how to optimize programs to improve their execution times? (May 2023)

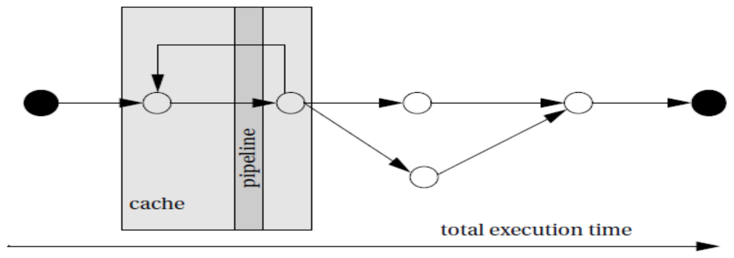


FIGURE 5.22
Execution time is a global property of a program.

- It is important to keep in mind that CPU performance is not judged in the same way as program performance.
- Certainly, CPU clock rate is a very unreliable metric for program performance. But more importantly, the fact that the CPU executes part of our program quickly does not mean that it will execute the entire program at the rate we desire.
- As illustrated in Figure 5.22, the CPU pipeline and cache act as windows into our

program. In order to understand the total execution time of our program, we must look at execution paths, which in general are far longer than the pipeline and cache windows.

- The pipeline and cache influence execution time, but execution time is a global property of the program.

The execution time of programs could be precisely determined; this is in **fact difficult to do in practice:**

■ The execution time of a program often varies with the input data values because those values select different execution paths in the program.

For example, loops may be executed a varying number of times, and different branches may execute blocks of varying complexity.

■ The cache has a major effect on program performance, and once again, the cache's behavior depends in part on the data values input to the program.

- Execution times may vary even at the instruction level. Floating-point operations are the most sensitive to data values, but the normal integer execution pipeline can also introduce data-dependent variations.
- In general, the execution time of an instruction in a pipeline depends not only on that instruction but on the instructions around it in the pipeline.

We can measure program performance in several ways:

- Some microprocessor manufacturers supply simulators for their CPUs: The simulator runs on a workstation or PC, takes as input an executable for the microprocessor along with input data, and simulates the execution of that program.
- Some of these simulators go beyond functional simulation to measure the execution time of the program. Simulation is clearly slower than executing the program on the actual microprocessor, but it also provides much greater visibility during execution.
- Be careful some microprocessor performance simulators are not 100% accurate, and simulation of I/O-intensive code may be difficult.
- A timer connected to the microprocessor bus can be used to measure performance of executing sections of code.
- The code to be measured would reset and start the timer at its start and stop the timer at the end of execution. The length of the program that can be measured is limited by the accuracy of the timer.
- A logic analyzer can be connected to the microprocessor bus to measure the start and stop times of a code segment. This technique relies on the code being able to produce identifiable events on the bus to identify the start and stop of execution.
- The length of code that can be measured is limited by the size of the logic analyzer's buffer.

The following three different types of performance measures on programs:

■ ***Average-case execution time:*** This is the typical execution time we would expect for typical data. Clearly, the first challenge is defining typical inputs.

■ ***Worst-case execution time:*** The longest time that the program can spend on any input sequence is clearly important for systems that must meet deadlines. In some cases, the input set that causes the worst-case execution time is obvious, but in many cases it is not.

■ ***Best-case execution time:*** This measure can be important in multi-rate real-time systems

Elements of Program Performance

The key to evaluating execution time is breaking the performance problem into parts. Program execution time can be seen as

execution time = program path+_ instruction timing

- The path is the sequence of instructions executed by the program (or its equivalent in the high-level language representation of the program).
- ➢ The instruction timing is determined based on the sequence of instructions traced by the program path, which takes into account data dependencies, pipeline behavior, and caching. Luckily, these two problems can be solved relatively independently.
- For example, if a program contains a loop with a large, fixed iteration bound or if one branch of a conditional is much longer than another, we can get at least a rough idea that these are more time-consuming segments of the program.
- Of course, a precise estimate of performance also relies on the instructions to be executed, since different instructions take different amounts of time.
- (In addition, to make life even more difficult, the execution time of one instruction can depend on the instructions executed before and after it). Example
  5.7 illustrates data-dependent program path.

UNIT-III PROCESSES AND OPERATING SYSTEMS

Structure of a real – time system – Task Assignment and Scheduling – Multiple Tasks and Multiple Processes – Multirate Systems – Pre emptive real – time Operating systems – Priority based scheduling – Interprocess Communication Mechanisms – Distributed Embedded Systems –MPSoCs and Shared Memory Multiprocessors – Design Example – Audio Player, Engine Control Unit and Video Accelerator.

(*Structure of a Real Time SystemTask Assignment and Scheduling.)*

- Most operating systems (even those that support multiple scheduling policies) schedule all applications according to the same scheduling algorithm at any given time.
- Whether each application can meet its timing requirements is determined by a global schedulability analysis based on parameters of every task in the system.
- The necessity of detailed timing and resource usage information of all applications that may run together often forces the applications to be developed together and, thus, keeps the system closed.
- Hard real-time applications can run with soft real-time and non-real-time applications in this environment. It makes use of the two-level scheduling scheme.
- This scheme enables each real- time application to be scheduled in a way best suited for the application and the schedulability of the application to be determined independent of other applications that may run with it on the same hardware platform.

Structure of a Real Time System

Q1. Discuss about task and processes.

Tasks and Processes

- Many (if not most) embedded computing systems do more than one thing—that is, the environment can cause mode changes that in turn cause the embedded system to behave quite differently.
- For example, when designing a telephone answering machine, one can define recording a phone call and operating the user‘s control panel as distinct tasks, because they perform logically distinct operations and they must be performed at very different rates.
- These different tasks are part of the system‘s functionality, but that application-level organization of functionality is often reflected in the structure of the program as well.

- A process is a single execution of a program. If a user runs the same program two different times, then two different processes are created. Each process has its own state that includes not only its registers but all of its memory.
- In some OSs, the memory management unit is used to keep each process in a separate address space. In others, particularly lightweight RTOSs, the processes run in the same address space. Processes that share the same address space are often called threads. The terms tasks and processes somewhat interchangeably, as do many people in the field. To be more precise, task can be composed of several processes or threads; it is also true that a task is primarily an implementation concept and processes more of an implementation concept.
- To understand why the separation of an application into tasks may be reflected in the program structure, consider how to build a stand-alone compression unit based on the compression algorithm. This device is connected to serial ports on both ends.
- The input to the box is an uncompressed stream of bytes. The box emits a compressed string of bits on the output serial line, based on a predefined compression table. Such a box may be used, for example, to compress data being sent to a modem.
- The program's need to receive and send data at different rates for example, the program may emit 2 bits for the first byte and then 7 bits for the second byte will obviously find itself reflected in the structure of the code.
- It is easy to create irregular, ungainly code to solve this problem; a more elegant solution is to create a queue of output bits, with those bits being removed from the queue and sent to the serial port in 8-bit sets.
- But beyond the need to create a clean data structure that simplifies the control structure of the code, and it is necessary to ensure that input and output signals are to be processed at the proper rates.
- For example, if too much of time is spent in packaging and emitting output characters, then input character can be dropped. Solving timing problems is a more challenging problem.

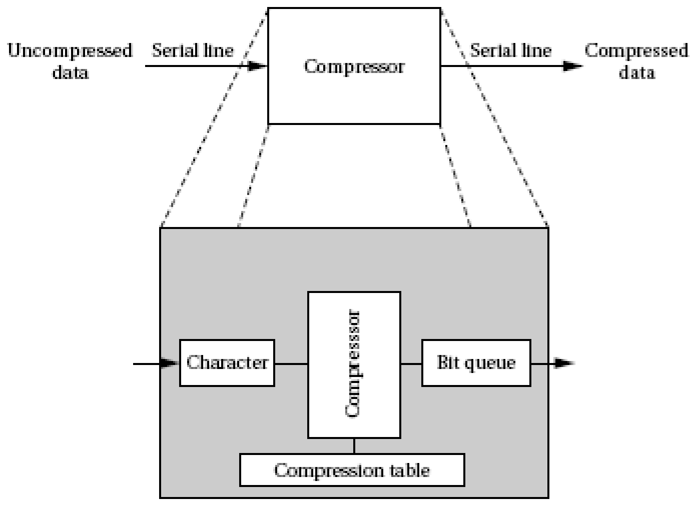


Figure 4.1 An on-the-fly compression box.

Major States. Our subsequent discussion focuses primarily on priority-driven systems. There are five major states of a thread.

- Sleeping: A periodic, aperiodic, or server thread is put in the sleeping state immediately after it is created and initialized. It is released and leaves the state upon the occurrence of an event of the specified types.
- Upon the completion of a thread that is to execute again, it is reinitialized and put in the sleeping state. A thread in this state is not eligible for execution.
- Ready: A thread enters the ready state after it is released or when it is preempted. thread in this state is in the ready queue and eligible for execution.
- Executing: A thread is the executing state when it executes.
- Suspended (or Blocked): A thread that has been released and is yet to complete enters the suspended (or blocked) state when its execution cannot proceed for some reason. The kernel puts a suspended thread in the suspended queue.
- Terminated: A thread that will not execute again enters the terminated state when it completes. A terminated thread may be destroyed.

4.1.2. The Kernel

Q2. Explain the structure of Microkernel.

- Again, with a few exceptions, a real-time operating system consists of a microkernel that provides the basic operating system functions described below. Figure 4.2 shows a general structure of a microkernel.
- There are three reasons for the kernel to take control from the executing thread and execute itself: to respond to a system call, do scheduling and service timers, and handle external interrupts. The kernel also deals with recovery from hardware and software exceptions, but activities are ignored.
- System Calls. The kernel provides many functions which, when called, do some work on behalf of the calling thread. An application can access kernel data and code via these functions. They are called Application Program Interface (API) functions.
- A system call is a call to one of the API functions. In a system that provides memory protection, user and kernel threads execute in separate memory spaces.
- Upon receiving a system call, the kernel saves the context of the calling thread and switches from the user mode to the kernel mode. It then picks up the function name and arguments of the call from the thread's stack and executes the function on behalf of the thread.
- When the system call completes, the kernel executes a return from exception. As a result, the system returns to the user mode. The calling thread resumes if it still has the highest priority. If the system call causes some other thread to have the highest priority, then that thread executes.
- The calling thread is blocked until the kernel completes the called function. When the call is asynchronous (e.g., in the case of an asynchronous I/O request), the calling thread

continues to execute after making the call. The kernel provides a separate thread to execute the called function.
- Many embedded operating systems do not provide memory protection; the kernel and user execute in the same space.

- Reasons for this choice are the relative trustworthiness of embedded applications and the need to keep overhead small. (The extra memory space needed to provide full memory protection is on the order of a few kilobytes per process. This overhead is more serious for small embedded applications than the higher context-switch overhead that also incurs with memory protection.)
- In such a system, a system call is just like a procedure or function call within the application. Figure 4.2 shows examples of thread management functions: create thread, suspend thread, resume thread and destroy thread.
- The timer functions listed below them exemplify the time services a real-time operating system provides.

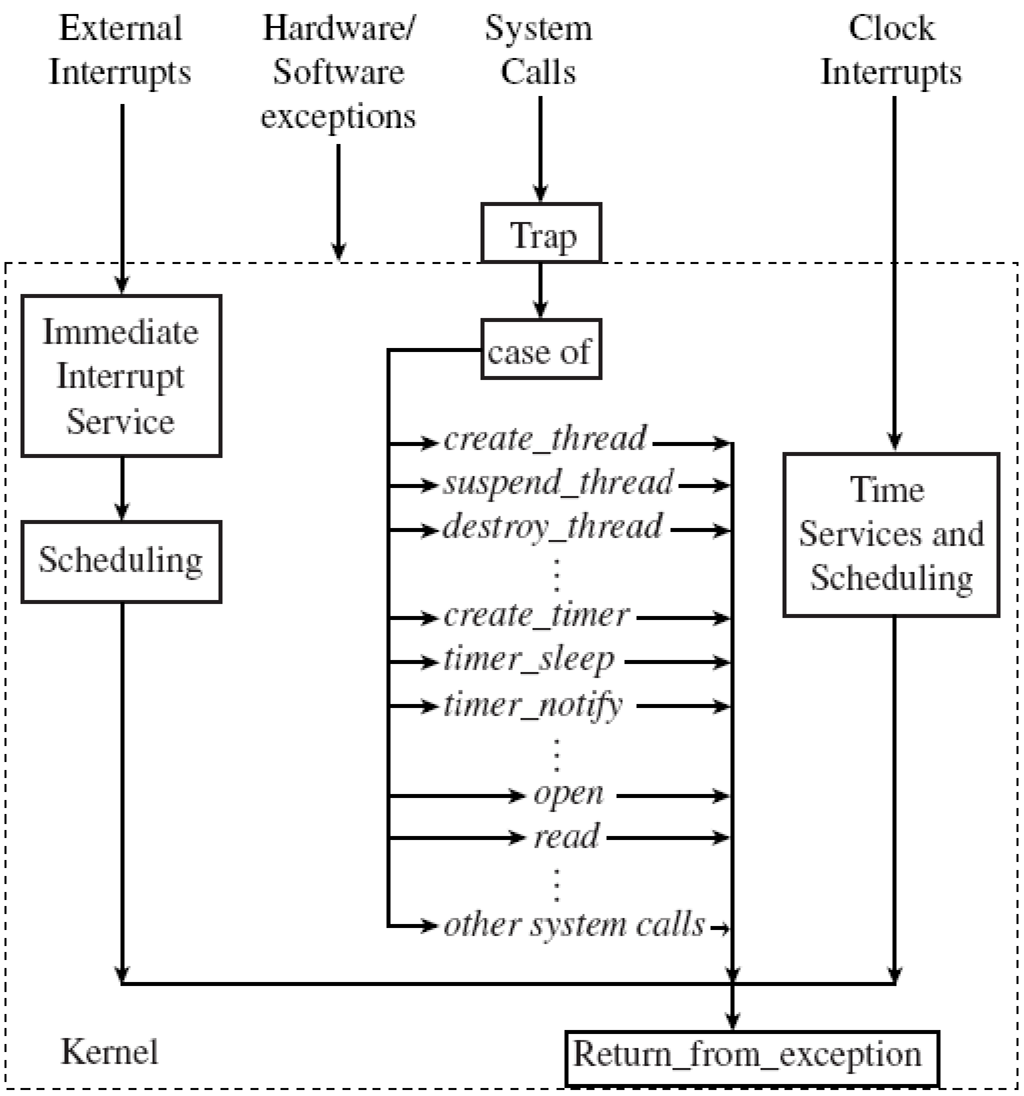


Figure 4.2 Structure of a microkernel.4.1.3. A Basic Model of a Real-Time System

Q3. Explain the basic model of real time systems (DEC20, APR21)

- The output interface, output conditioning, and the actuator are interfaced in a complementary manner. In the following, content briefly describe the roles of the different functional blocks of a real-time system.
- Sensor: A sensor converts some physical characteristic of its environment into electrical signals. An example of a sensor is a photo-voltaic cell which converts light energy into electrical energy. A wide variety of temperature and pressure sensors are also used.
- A temperature sensor typically operates based on the principle of a thermocouple. Temperature sensors based on many other physical principles also exist.
- For example, one type of temperature sensor employs the principle of variation of electrical resistance with temperature (called a varistor). A pressure sensor typically operates based on the piezoelectricity principle. Pressure sensors based on other physical principles also exist.
- Actuator: An actuator is any device that takes its inputs from the output interface of a computer and converts these electrical signals into some physical actions on its environment.
- The physical actions may be in the form of motion, change of thermal, electrical, pneumatic, or physical characteristics of some objects. A popular actuator is a motor. Heaters are also very commonly used. Besides, several hydraulic and pneumatic actuators are also popular.
- Signal Conditioning Units: The electrical signals produced by a computer can rarely be used to directly drive an actuator. The computer signals usually need conditioning before they can be used by the actuator.
- This is termed output conditioning. Similarly, input conditioning is required to be carried out on sensor signals before they can be accepted by the computer.

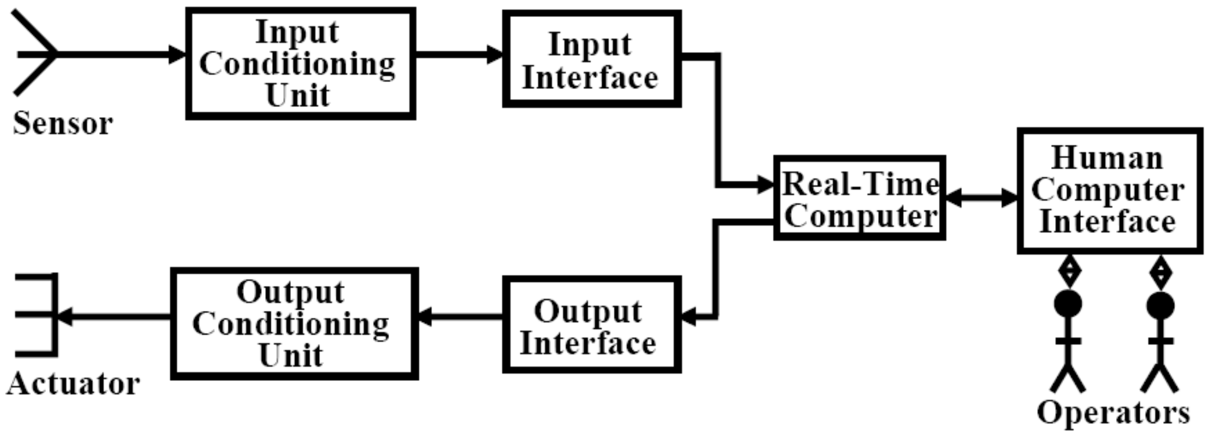


Figure 4.3 A model of real time operating system

For example, analog signals generated by a photo-voltaic cell are normally in the milli- volts range and need to be conditioned before they can be processed by a computer.

- The following are some important types of conditioning carried out on raw signals generated by sensors and digital signals generated by computers:

- Voltage Amplification: Voltage amplification is normally required to be carried out to match the full scale sensor voltage output with the full scale voltage input to the interface of a computer.
- For example, a sensor might produce voltage in the millivolts range, whereas the input interface of a computer may require the input signal level to be of the order of a volt.
- Voltage Level Shifting: Voltage level shifting is often required to align the voltage level generated by a sensor with that acceptable to the computer.
- For example, a sensor may produce voltage in the range -0.5 to +0.5 volt, whereas the input interface of the computer may accept voltage only in the range of 0 to 1 volt. In this case, the sensor voltage must undergo level shifting before it can be used by the computer.
- Frequency Range Shifting and Filtering: Frequency range shifting is often used to reduce the noise components in a signal. Many types of noise occur in narrow bands and the signal must be shifted from the noise bands so that noise can be filtered out.
- Signal Mode Conversion: A type of signal mode conversion that is frequently carried out during signal conditioning involves changing direct current into alternating current and vice-versa.
- Another type signal mode conversion that is frequently used is conversion of analog signals to a constant amplitude pulse train such that the pulse rate or pulse width is proportional to the voltage level.
- Conversion of analog signals to a pulse train is often necessary for input to systems such as transformer coupled circuits that do not pass direct current. D/A

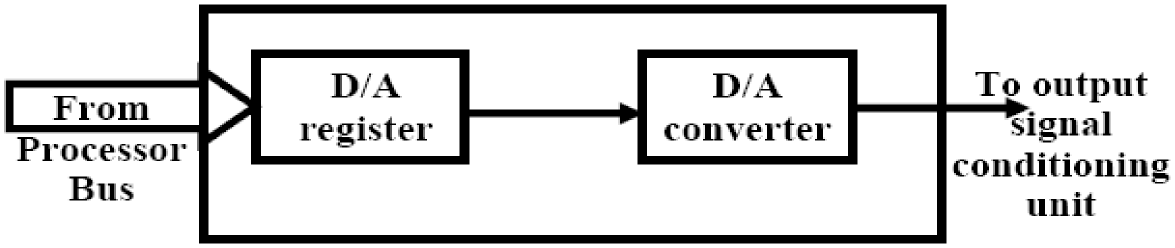


Figure 4.4 An output interface.

- Interface Unit: Normally commands from the CPU are delivered to the actuator through an output interface. An output interface converts the stored voltage into analog form and then outputs this to the actuator circuitry.

4.1.4. Estimating program run times

Q4. Discuss about estimating program run times /Enumerate the need for host based systems for stages of simulation, porting kernels and estimating program run times in embedded application deployment (DEC2022/JAN2023)

- Processes can have several different types of timing requirements imposed on them by the application. The timing requirements on a set of processes strongly influence the type of scheduling that is appropriate.
- A scheduling policy must define the timing requirements that it uses to determine whether a schedule is valid. Before studying scheduling proper, some outline of the types of process timing requirements that are useful in embedded system design.
- Figure 4.5 illustrates different ways in which two important requirements on processes can be defined: **release time** and **deadline**.
- The release time is the time at which the process becomes ready to execute; this is not necessarily the time at which it actually takes control of the CPU and starts to run.
- An aperiodic process is by definition initiated by an event, such as external data arriving or data computed by another process.
- The release time is generally measured from that event, although the system may want to make the process ready at some interval after the event itself.
- For a periodically executed process, there are two common possibilities. In simpler systems, the process may become ready at the beginning of the period. More sophisticated systems, such as those with data dependencies between processes, may set the release time at the arrival time of certain data, at a time after the start of the period.
- A deadline specifies when a computation must be finished. The deadline for an aperiodic process is generally measured from the release time, since that is the only reasonable time reference. The deadline for a periodic process may in general occur at some time other than the end of the period. Some scheduling policies make the simplifying assumption that the deadline occurs at the end of the period.

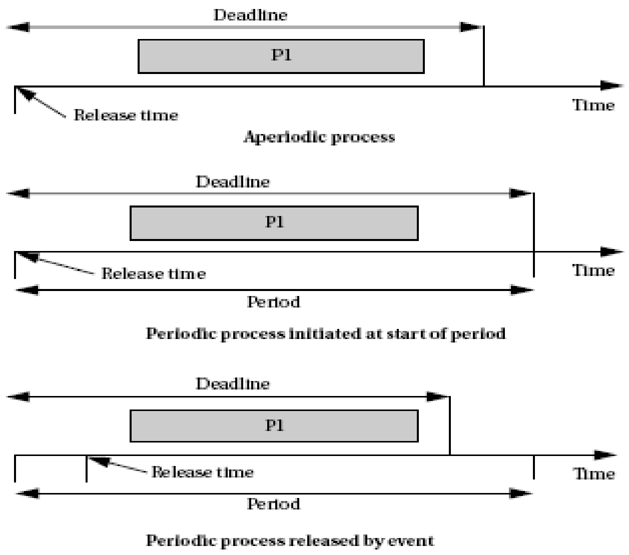


Figure 4.5 Example definitions of release times and deadlines.

- Rate requirements are also fairly common. A rate requirement specifies how quickly processes must be initiated.
- The **period** of a process is the time between successive executions. For example, the period of a digital filter is defined by the time interval between successive input samples.
- The process‘s **rate** is the inverse of its period. In a multirate system, each process executes at its own distinct rate.
- The most common case for periodic processes is for the initiation interval to be equal to the period. However, pipelined execution of processes allows the initiation interval to be less than the period. Figure 4.6 illustrates process execution in a system with four CPUs.
- The various execution instances of program P1 have been subscripted to distinguish their initiation times. In this case, the initiation interval is equal to one fourth of the period.
- It is possible for a process to have an initiation rate less than the period even in single-CPU systems.
- If the process execution time is significantly less than the period, it may be possible to initiate multiple copies of a program at slightly offset times violation depend on the application—the results can be catastrophic in an automotive control system, whereas a missed deadline in a multimedia system may cause an audio or video glitch.
- The system can be designed to take a variety of actions when a deadline is missed. Safety-critical systems may try to take compensatory measures such as approximating data or switching into a special safety mode. Systems for which safety is not as important may take simple measures to avoid propagating bad data, such as inserting silence in a phone line, or may completely ignore the failure.
- Even if the modules are functionally correct, their timing improper behavior can introduce major execution errors. Application Example 4.6 describes a timing problem in space shuttle software that caused the delay of the first launch of the shuttle.

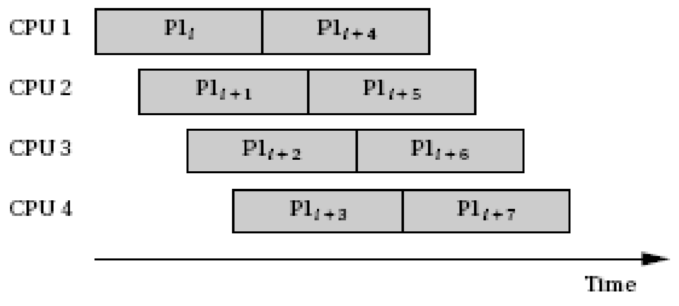


Figure 4.6 A sequence of processes with a high initiation rate.

- The order of execution of processes may be constrained when the processes pass data between each other. Figure 4.6 shows a set of processes with data dependencies among them. Before a process can become ready, all the processes on which it depends must complete and send their data to it.

- The data dependencies define a partial ordering on process execution—P1 and P2 can execute in any order (or in interleaved fashion) but must both complete before P3, and P3 must complete before P4.All processes must finish before the end of the period.
- The data dependencies must form a directed acyclic graph (DAG)—a cycle in the data dependencies is difficult to interpret in a periodically executed system.
- A set of processes with data dependencies is known as a **task graph**. Although the terminology for elements of a task graph varies from author to author, some of the component of the task graph is considered (a set of nodes connected by data dependencies) as a **task** and the complete graph as the **task set**.
- The figure also shows a second task with two processes. The two tasks ({P1, P2, P3, P4} and {P5, P6}) have no timing relationships between them.
- Communication among processes that run at different rates cannot be represented by data dependencies because there is no one-to-one relationship between data coming out of the source process and going into the destination process Nevertheless, communication among processes of different rates is very common.
- Figure 4.7 illustrates the communication required among three elements of an MPEG audio/video decoder.
- Data come into the decoder in the system format, which multiplexes audio and video data. The system decoder process demultiplexes the audio and video data and distributes it to the appropriate processes.
- Multirate communication is necessarily one way—for example, the system process writes data to the video process, but a separate communication mechanism must be provided for communication from the video process back to the system process.

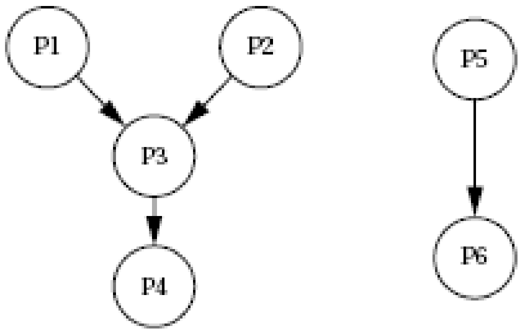


Figure 4.7 Data dependencies among processes.

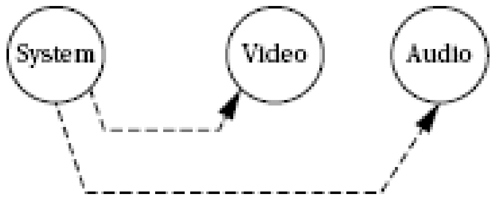


Figure 4.8 Communication among processes at different rates.

CPU usage metrics

A basic measure of the efficiency with usage of CPU. The simplest and most direct measure is utilization: Utilization U is equal to

$$U = \frac{\textit{CPU time for useful work}}{\textit{Total available CPU time}}$$

Utilization is the ratio of the CPU time that is being used for useful computations to the total available CPU time. This ratio ranges between 0 and 1, with 1 meaning that all of the available CPU time is being used for system purposes. The utilization is often expressed as a percentage.

4.1.5. Task Assignment and Scheduling

Q5. Explain Task Assignment and Scheduling/ Scheduling of real time systems/ what is the purpose of Priority based scheduling? Discuss in detail with appropriate diagrams (Dec 2022/Jan 2023, May 2023, Dec 2023)

Scheduling

- The first job of the OS is to determine that process runs next. The work of choosing the order of running processes is known as scheduling.
- The OS considers a process to be in one of three basic scheduling states: waiting, ready, or executing. There is at most one process executing on the CPU at any time. (If there is no useful work to be done, an idling process may be used to perform a null operation.) Any process that could execute is in the ready state; the OS chooses among the ready processes to select the next executing process.
- A process may not, however, always be ready to run. For instance, a process may be waiting for data from an I/O device or another process, or it may be set to run from a timer that has not yet expired. Such processes are in the waiting state.
- A process goes into the waiting state when it needs data that it has not yet received or when it has finished all its work for the current period.
- A process goes into the ready state when it receives its required data and when it enters a new period.
- A process can go into the executing state only when it has all its data, is ready to run, and the scheduler selects the process as the next process to run.

Scheduling Policies

- A scheduling policy defines how processes are selected for promotion from the ready state to the running state. Every multitasking OS implements some type of scheduling policy.

- Choosing the right scheduling policy not only ensures that the system will meet all its timing requirements, but it also has a profound influence on the CPU horsepower required to implement the system‘s functionality.
- Schedulability means whether there exists a schedule of execution for the processes in a system that satisfies all their timing requirements. In general, it is necessary to construct a schedule to show schedulability, but in some cases some sets of processes as unschedulable using some very simple tests can be eliminated.
- Utilization is one of the key metrics in evaluating a scheduling policy. Our most basic requirement is that CPU utilization be no more than 100% since one can‘t use the CPU more than 100% of the time.

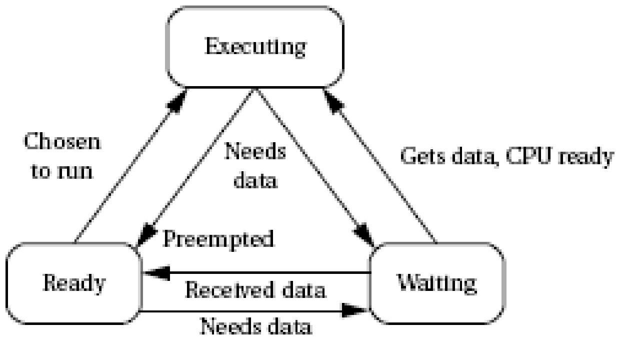


Figure 4.9 Scheduling states of a process.

Priority-Based Scheduling

- ➢ After assigning priorities, the OS takes care of the rest by choosing the highest-priority ready process.
- ➢ There are two major ways to assign priorities: static priorities that do not change during execution and dynamic priorities that do change.

Rate-Monotonic Scheduling

- ➢ Rate-monotonic scheduling (RMS), introduced by Liu and Layland, was one of the first scheduling policies developed for real-time systems and is still very widely used. RMS is a static scheduling policy.
- ➢ It turns out that these fixed priorities are sufficient to efficiently schedule the processes in many situations. The theory underlying RMS is known as rate-monotonic analysis (RMA).
- ➢ This theory, as summarized below, uses a relatively simple model of the system.

- All processes run periodically on a single CPU.
- Context switching time is ignored.
- There are no data dependencies between processes.
- The execution time for a process is constant.

- All deadlines are at the ends of their periods.
- The highest-priority ready process is always selected for execution.

➢ The major result of RMA is that a relatively simple scheduling policy is optimal under certain conditions.

➢ Priorities are assigned by rank order of period, with the process with the shortest period being assigned the highest priority.

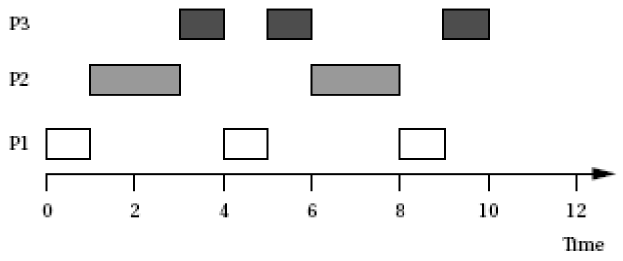
➢ This fixed-priority scheduling policy is the optimum assignment of static priorities to processes, in that it provides the highest CPU utilization while ensuring that all processes meet their deadlines.

4.1.6. Example: Rate-monotonic scheduling.

Here is a simple set of processes and their characteristics.

Process	Execution time	Period
P1	1	4
P2	2	6
P3	3	12

Applying the principles of RMA, P1 is provided with the highest priority, P2 the middle priority, and P3 the lowest priority. To understand all the interactions between the periods, it is needed to construct a time line equal in length to hyper period, which are 12 in this case.



- All three periods start at time zero. P1's data arrive first. Since P1 is the highest-priority process, it can start to execute immediately.
- After one time unit, P1 finishes and goes out of the ready state until the start of its next period. At time 1, P2 starts executing as the highest-priority ready process. At time 3, P2 finishes and P3 starts executing. P1's next iteration starts at time 4, at which point it interrupts P3. P3 gets one more time unit of execution between the second iterations of P1 and P2, but P3 does not get to finish until after the third iteration of P1.
- Consider the following different set of execution times for these processes, keeping the same deadlines.
- In this case, no feasible assignment of priorities that guarantees scheduling can be shown. Even though each process alone has an execution time significantly less than its period, combinations of processes can require more than 100% of the available CPU cycles.

- For example, during one 12 time-unit interval, P1 was executed three times, requiring 6 units of CPU time; P2 twice, costing 6 units of CPU time; and P3 one time, requiring 3 units of CPU time. The total of 6 + 6 + 3 = 15 units of CPU time is more than the 12 time units available, clearly exceeding the available CPU capacity.

Process	Execution time	Period
P1	2	4
P2	3	6
P3	3	12

4.1.7. Earliest-Deadline-First Scheduling

- Earliest deadline first (EDF) is another well-known scheduling policy that was also studied by Liu and Layland. It is a dynamic priority scheme—it changes process priorities during execution based on initiation times. As a result, it can achieve higher CPU utilizations than RMS.
- The EDF policy is also very simple: It assigns priorities in order of deadline. The highest-priority process is the one whose deadline is nearest in time, and the lowest priority process is the one whose deadline is farthest away.
- Clearly, priorities must be recalculated at every completion of a process. However, the final step of the OS during the scheduling procedure is the same as for RMS—the highest-priority ready process is chosen for execution.
- The implementation of EDF is more complex than the RMS code. The major problem is keeping the processes sorted by time to deadline—since the times to deadlines for the processes change during execution.
- To avoid resorting the entire set of records at every change, a binary tree to keep the sorted records can be built and incrementally update the sort.
- At the end of each period, the records are moved to its new place in the sorted list by deleting it from the tree and then adding it back to the tree using standard tree manipulation techniques.
- And process priorities by traversing them in sorted order need to be updated, so the incremental sorting routines must also update the linked list pointers that let us traverse the records in deadline order. (The linked list lets us avoid traversing the tree to go from one node to another, which would require more time.)
- After putting in the effort to building the sorted list of records, selecting the next executing process is done in a manner similar to that of RMS. However, the dynamic sorting adds complexity to the entire scheduling process.
- Each update of the sorted list requires O (log n) steps. The EDF code is also significantly more complex than the RMS code.

RMS vs. EDF

- Which scheduling policy is better: RMS or EDF? That depends on criteria of a user. EDF can extract higher utilization out of the CPU, but it may be difficult to diagnose the possibility of an imminent overload.
- Because the scheduler does take some overhead to make scheduling decisions, a factor that is ignored in the schedulability analysis of both EDF and RMS, running a scheduler at very high utilizations is somewhat problematic.
- RMS achieves lower CPU utilization but is easier to ensure that all deadlines will be satisfied. In some applications, it may be acceptable for some processes to occasionally miss deadlines. For example, a set-top box for video decoding is not a safety-critical application, and the occasional display artifacts caused by missing deadlines may be acceptable in some markets.

What if your set of processes is unschedulable and you need to guarantee that they complete their deadlines? There are several possible ways to solve this problem:

- Get a faster CPU. That will reduce execution times without changing the periods, giving you lower utilization. This will require you to redesign the hardware, but this is often feasible because you are rarely using the fastest CPU available.
- Redesign the processes to take less execution time. This requires knowledge of the code and may or may not be possible.
- Rewrite the specification to change the deadlines. This is unlikely to be feasible, but may be in a few cases where some of the deadlines were initially made tighter than necessary.

MULTIPLE TASKS AND MULTIPLE PROCESSES:

Write short notes on multiple tasks and multiple processes (May 2008)

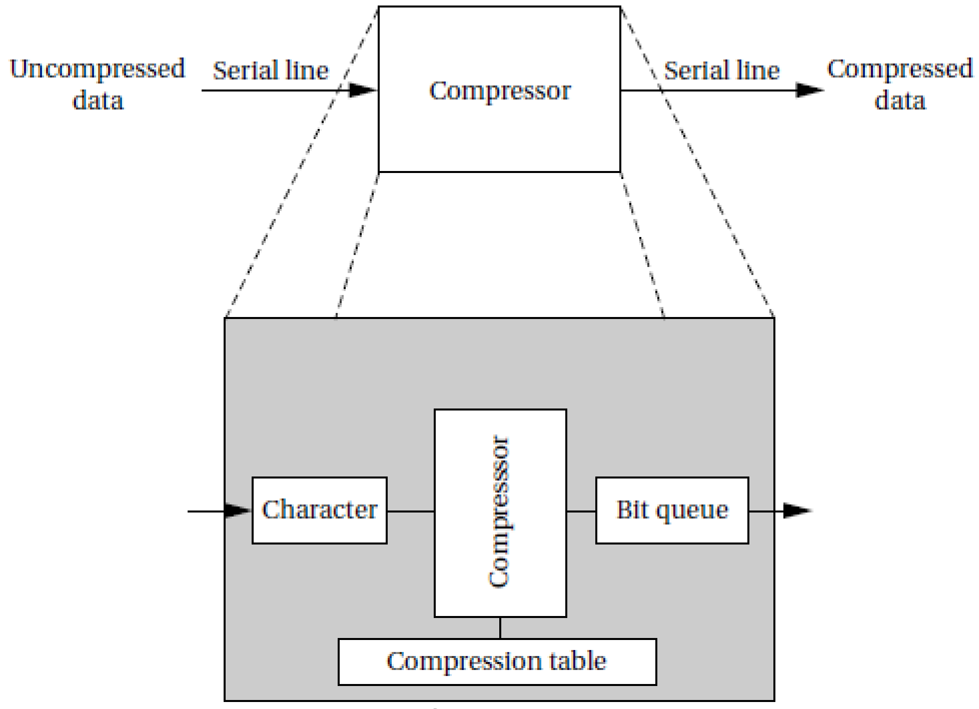
5.1.1 Tasks and Processes

- Most of the embedded systems are too complex. Hence programming the system are also complex. To reduce the complexity we break the system in to Multiple tasks
- Task is nothing but different parts of functionality in a single system. Thus the various application in a system and each system is called Task
- For example, when designing a telephone answering machine, we can define recording a phone call and operating the user’s control panel as distinct tasks.
- A ***process*** is a single execution of a program.
- If we run the same program two different times, we have created two different processes.
- Each process has its own state that includes not only its registers but all of its memory.
- In some OSs, the memory management unit is used to keep each process in a separate address space.
- In others, particularly lightweight RTOSs, the processes run in the same address space. Processes that share the same address space are often called ***threads***.

Example:

- ✓ Consider a standalone compression unit, this device is connected to serial ports on both ends.
- ✓ The input to the box is an uncompressed stream of bytes.
- ✓ The box emits a compressed string of bits on the output serial line, based on a predefined compression table.
- ✓ Such a box may be used, to compress data being sent to a modem.
- ✓ The program's need to receive and send data at different rates.
- ✓ For example, the program may emit 2 bits for the first byte and then 7 bits for the second byte will obviously find itself reflected in the structure of the code.2

✓ It is easy to create irregular code to solve this problem; a more elegant solution is to create a queue of output bits, with those bits being removed from the queue and sent to the serial port in 8-bit sets.



An on-the-fly compression box.

✓ Ensure that processing of the inputs and outputs are at the proper rates.

- ✓ For example, if too much time is spend in packaging and emitting output characters, we may drop an input character. Solving timing problems is a more challenging problem.
- ✓ The text compression box provides a simple example of rate control problems.
- ✓ A control panel on a machine provides an example of a different type of rate control problem, the ***asynchronous input***.
- ✓ The control panel of the compression box may include a compression mode button that disables or enables compression, so that the input text is passed through unchanged when compression is disabled.

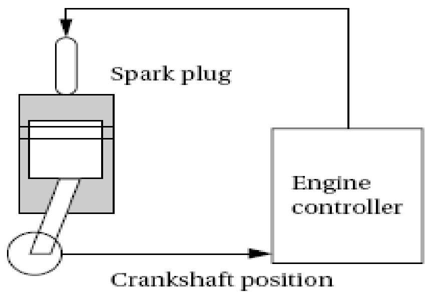
5.1.2 Multirate Systems

Briefly explain about multirate systems with an example. (May2011)

- The systems which are embedded with more than one application are called multi rate system.
- Implementing code that satisfies timing requirements is even more complex when multiple rates of computation must be handled.
- Multirate embedded computing systems are very common, including automobile engines, printers, and cell phones.
- In all these systems, certain operations must be executed periodically, and each operation is executed at its own rate.

Example: Automotive engine controller:

- ✓ The simplest automotive engine controllers, such as the ignition controller for a basic motorcycle engine, perform only one task, timing the firing of the spark plug, which takes the place of a mechanical distributor.
- ✓ The spark plug must be fired at a certain point in the combustion cycle, but to obtain better performance, the phase relationship between the piston's movement and the spark should change as a function of engine speed.
- ✓ Using a microcontroller that senses the engine crankshaft position allows the spark timing to vary with engine speed.
- ✓ Firing the spark plug is a periodic Process.
- ✓ The control algorithm for a modern automobile engine is much more complex, making the need for microprocessors that much greater.
- ✓ Automobile engines must meet strict requirements on both emissions and fuel economy.
- ✓ On the other hand, the engines must still satisfy customers not only in terms of performance, ease of starting in extreme cold and heat and low maintenance
- ✓ Automobile engine controllers use additional sensors, including the gas pedal position and an oxygen sensor used to control emissions.
- ✓ They also use a multimode control scheme. For example, one mode may be used for engine warm-up, another for cruise, and yet another for climbing steep hills, and so forth.
- ✓ The larger number of sensors and modes increases the number of discrete tasks that must be performed.
- ✓ The engine controller takes a variety of inputs that determine the state of the engine.
- ✓ It controls two basic engine parameters: the spark plug firings and the fuel/air mixture.



5.1.3 Timing Requirements on Processes

Explain in detail about timing requirement on processes.

- ❖ Processes can have several different types of timing requirements imposed on them by the application.
- ❖ The timing requirements on a set of processes strongly influence the type of scheduling that is appropriate.
- ❖ A scheduling policy must define the timing requirements that it uses to determine whether a schedule is valid.

There are two important timing requirements on processes: ***release time*** and ***deadline***.

Release time:

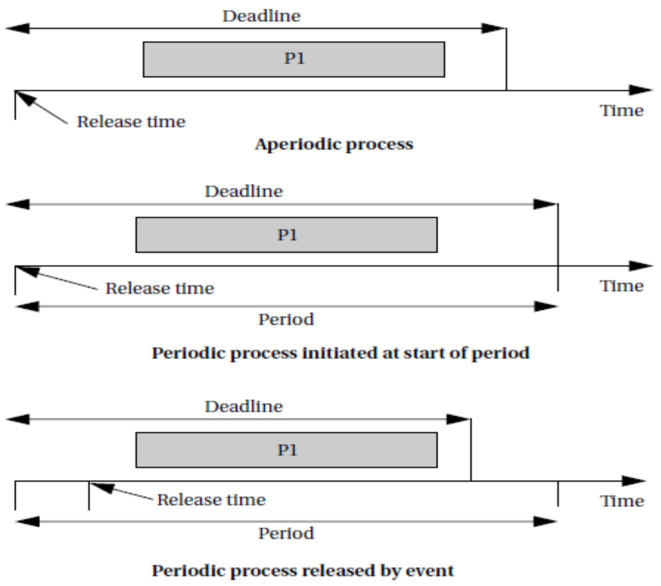
- ✓ The release time is the time at which the process becomes ready to execute.
- ✓ An aperiodic process is initiated by an event, such as external data arriving or data computed by another process.
- ✓ The release time is generally measured from that event.

For a periodically executed process, there are two common possibilities.

- ✓ In simpler systems, the process may become ready at the beginning of the period
- ✓ More sophisticated systems may set the release time at the arrival time of certain data, at a time after the start of the period.

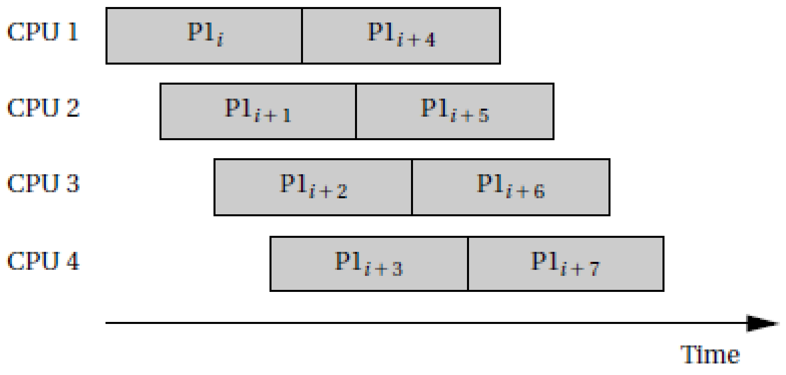
Deadline:

- A deadline specifies when a computation must be finished.
- The deadline for an aperiodic process is generally measured from the release time
- The deadline for a periodic process may occur at some time other than the end of the period.
- The ***period*** of a process is the time between successive executions. For example, the period of a digital filter is defined by the time interval between successive input samples.
- The process's ***rate*** is the inverse of its period. In a multirate system, each process executes at its own distinct rate.



Example definitions of release times and deadlines.

- ✓ For periodic processes the initiation interval to be equal to the period. However, pipelined execution of processes allows the initiation interval to be less than the period.



A sequence of processes with a high initiation rate.

5.1.4 CPU Metrics

Write short notes on CPU Metrics.

CPU metrics are described by ***initiation time*** and ***completion time***

Initiation time:

- The ***initiation time*** is the time at which a process actually starts executing on the CPU.

Completion time:

- The ***completion time*** is the time at which the process finishes its work.

- ❖ The most basic measure of work is the amount of ***CPU time*** expended by a process.
- ❖ The CPU time of process *i* is called *Ci*.

- The CPU time is not equal to the completion time minus initiation time; several other processes may interrupt execution.
- The total CPU time consumed by a set of processes is

$$T| = \sum_{1 \le i \le n} T_i .$$

- To measure the efficiency of CPU, the simplest and most direct measure is ***utilization***:

$$U = \frac{\text{CPU time for useful work}}{\text{Total available CPU time}}$$

U=T/t

- Utilization is the ratio of the CPU time that is being used for useful computations to the total available CPU time.
- This ratio ranges between 0 and 1, with 1 meaning that all of the available CPU time is being used for system purposes.
- The utilization is often expressed as a percentage.

5.2 OPERATING SYSTEMS:

Discuss in detail about Preemptive real time operating systems.
(NOV/DEC 2007, May 2012, NOV 2017, APRIL/MAY 2019)

- An Operating system is a program that controls the execution of application programs and acts as an interface between the user of a computer and the computer hardware.
- A more common definition is that the operating system is the one program running at all times on the computer (usually called the kernel), with all application programs.
- An Operating system is concerned with the allocation of resources and services, such as memory, processors, devices and information. The Operating System correspondingly includes programs to manage these resources, such as a traffic controller, a scheduler, memory management module, I/O programs, and a file system.

PREEMPTIVE REAL-TIME OPERATING SYSTEMS

- A RTOS executes processes based upon timing constraints provided by the system designer.
- The most reliable way to meet timing constraints accurately is to build a ***preemptive*** OS and to use ***priorities*** to control what process runs at any given time.
  This operating system runs on many different platforms.

Preemption

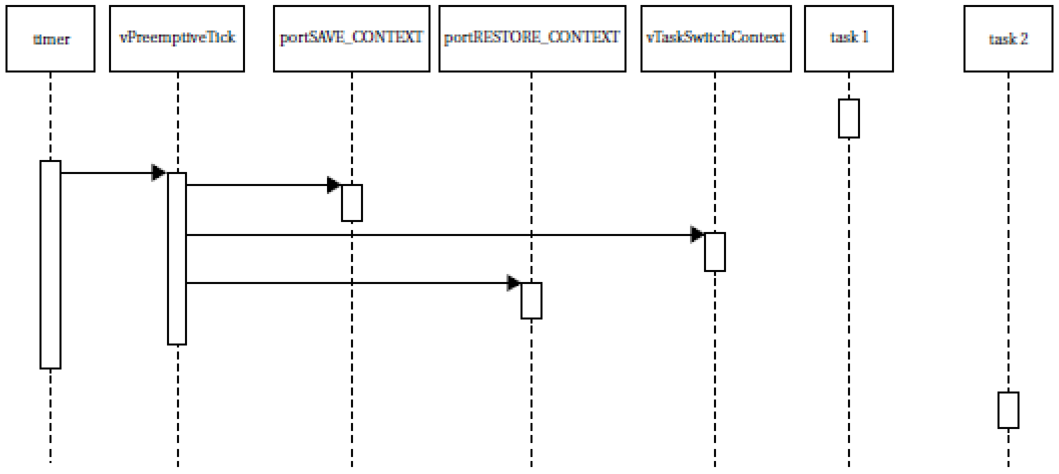
- Preemption is an alternative to the C function call as a way to control execution.
- To be able to take full advantage of the timer, assume process as something more than a function call.
- Create new routines that allow us to jump from one subroutine to another at any point in the program.

- That, together with the timer, will allow moving between functions whenever necessary based upon the system’s timing constraints.
- The CPU is shared across two processes. The ***kernel*** is the part of the OS that determines what process is running.
- The kernel is activated periodically by the timer.
- The length of the timer period is known as the ***time quantum*** because it is the smallest increment in which we can control CPU activity.
- The kernel determines what process will run next and causes that process to run.
- On the next timer interrupt, the kernel may pick the same process or another process to run.
- Before, using the timer to control loop iterations, with one loop iteration including the execution of several complete processes.
- Here, the time quantum is in general smaller than the execution time of any of the processes.
- The timer interrupts causes control to change from the currently executing process to the kernel; assembly language can be used to save and restore registers.
- Similarly use assembly language to restore registers not from the process that was interrupted by the timer but to use registers from any process we want.
- The set of registers that define a process are known as its ***context*** and switching from one process’s register set to another is known as ***context switching***.
- The data structure that holds the state of the process is known as the ***process control block***.

PROCESSES AND CONTEXT SWITCHING:

Write short notes on context switching.

- ✓ The first job of the OS is to determine the process that runs next.
- ✓ The work of choosing the order of running processes is known as scheduling.
- ✓ The OS considers a process to be in one of three basic ***scheduling states*: *waiting***, ***ready***, or ***executing***.
- ✓ The best way to understand processes and context is to dive into an RTOS implementation.
- ✓ A process is known in FreeRTOS.org as a task.
- ✓ Task priorities in FreeRTOS.org are ranked opposite to the convention, higher numbers denote higher priorities and the priority 0 task is the idle task.



Sequence diagram for freeRTOS.org context switch.

- ✓ To understand the basics of a context switch, assume that the set of tasks is in steady state.
- ✓ Everything has been initialized, the OS is running, and we are ready for a timer interrupt.
- ✓ The sequence diagram shows the application tasks, the hardware timer, and all the functions in the kernel that are involved in the context switch:
- vPreemptiveTick () is called when the timer ticks.
- Port SAVE_CONTEXT() swaps out the current task context..
- vTaskSwitchContext ( ) chooses a new task.
- Port RESTORE_CONTEXT() swaps in the new context.

5.4 SCHEDULING POLICIES:

Explain in detail about different types of scheduling policies. (April 2013)

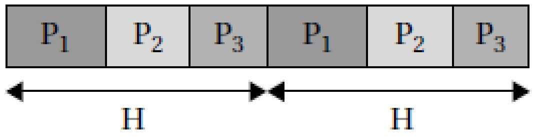
A ***scheduling policy*** defines how processes are selected for promotion from the ready state to the running state.

- Choosing the right scheduling policy not only ensures that the system will meet all its timing requirements, it also influence the CPU horsepower required to implement the system's functionality.
- Schedulability means whether there exists a schedule of execution for the processes in a system that satisfies all their timing requirements.
- Utilization is one of the key metrics in evaluating a scheduling policy.
- The most basic requirement is that CPU utilization be no more than 100%
- For periodic processes, the length of time that must be considered is the ***hyperperiod***, which is the least-common multiple of the periods of all the processes.
- The complete schedule for the least-common multiple of the periods is sometimes called the ***unrolled schedule***.

Types of scheduling:

Cyclostatic Scheduling:

- One very simple scheduling policy is known as ***cyclostatic*** scheduling or sometimes as ***Time Division Multiple Access*** scheduling.
- A cyclostatic schedule is divided into equal-sized time slots over an interval equal to the length of the hyperperiod *H*.
- Processes always run in the same time slot.

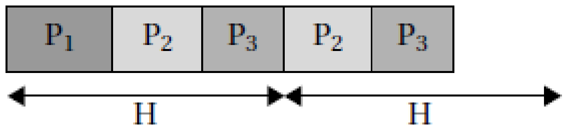


Cyclostatic scheduling.

- Two factors affect utilization in cyclostatic scheduling
  - The number of time slots used
  - The fraction of each time slot that is used for useful work.
- Depending on the deadlines for some of the processes, some time slots may need to beleave empty.
- Since the time slots are of equal size, some short processes may have time left over in their time slot

Round-robin scheduling:

- Round robin uses the same hyperperiod as in cyclostatic.
- It also evaluates the processes in order.



Round-robin scheduling.

- But unlike cyclostatic scheduling, if a process does not have any useful work to do, the round-robin scheduler moves on to the next process in order to fill the time slot with useful work.
- In this example, all three processes execute during the first hyperperiod, but during the second one, *P*1 has no useful work and is skipped.
- The processes are always evaluated in the same order.
- The last time slot in the hyperperiod is left empty; if we have occasional, non-periodic tasks without deadlines.
- Round-robin scheduling is often used in hardware such as buses because it is very simple to implement but it provides some amount of flexibility.

- ❖ In addition to utilization, also consider **scheduling overhead**—the execution time required to choose the next execution process, which is incurred in addition to any context switching overhead.
- ❖ In general, the more sophisticated the scheduling policy, the more CPU time it takes during system operation to implement it
- ❖ The final decision on a scheduling policy must take into account both theoretical utilization and practical scheduling overhead.

5.5 PRIORITY-BASED SCHEDULING

Briefly explain about priority based scheduling and its types (NOV/DEC 2006, 2007, 2009, May 2012, May 2023, Dec 2023)

- ❖ To determine an algorithm to assign priorities to the processes, the OS takes care of the rest by choosing the highest-priority ready process. There are two major ways to assign priorities:

  I) Static: Static priorities are that do not change during execution

  II) Dynamic: Dynamic priorities that do change during execution.

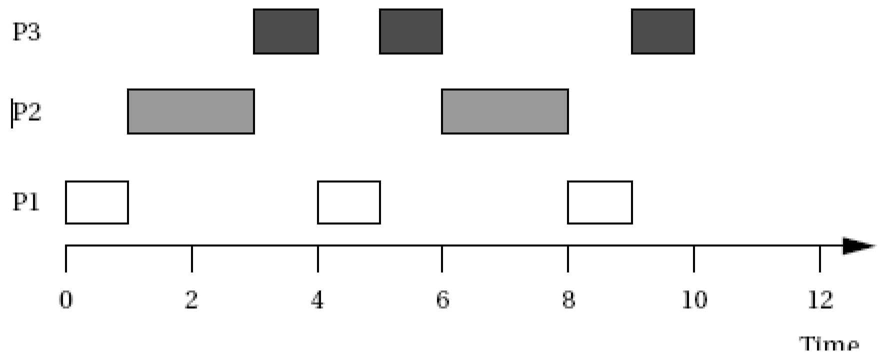
  Depending on the static and dynamic way of assigning priority there are two methods to schedule the process. They are

Rate-Monotonic Scheduling

- ❖ ***Rate-monotonic scheduling (RMS)***, introduced by Liu and Layland [Liu73],was one of the first scheduling policies developed for real-time systems and is still very widely used.
- ❖ RMS is a static scheduling policy.
- ❖ It turns out that these fixed priorities are sufficient to efficiently schedule the processes in many situations.
- ❖ The theory underlying RMS is known as ***rate-monotonic analysis (RMA)***.This theory, as summarized below.
  - ✓ All processes run periodically on a single CPU.
  - ✓ Context switching time is ignored. There are no data dependencies between processes.
  - ✓ The execution time for a process is constant.
  - ✓ All deadlines are at the ends of their periods.
  - ✓ The highest-priority ready process is always selected for execution.
- ❖ The major result of RMA is that a relatively simple scheduling policy is optimal under certain conditions.
- ❖ Priorities are assigned by rank order of period, with the process with the shortest period being assigned the highest priority.
- ❖ This fixed-priority scheduling policy is the optimum assignment of static priorities to processes, in that it provides the highest CPU utilization while ensuring that all processes meet their deadlines.
- ❖ Here is a simple set of processes and their characteristics.

Process	Execution time	Period
P1	1	4
P2	2	6
P3	3	12

- ❖ Applying the principles of RMA, we give P1 the highest priority, P2 the middle priority, and P3 the lowest priority.
- ❖ To understand all the interactions between the periods, construct a time line equal in length to hyperperiod, which is 12 in this case.



- ❖ All three periods start at time zero. P1's data arrive first. Since P1 is the highest-priority process, it can start to execute immediately.
- ❖ After one time unit, P1 finishes and goes out of the ready state until the start of its next period. At time 1, P2 starts executing as the highest-priority ready process.
- ❖ At time 3, P2 finishes and P3 starts executing. P1's next iteration starts at time 4, at which point it interrupts P3.
- ❖ P3 gets one more time unit of execution between the second iterations of P1 and P2, but P3 does not get to finish until after the third iteration of P1.
- ❖ In this case the CPU time execute with in the period of 12 units. Even though each process alone has an execution time significantly less than its period, combinations of processes can require more than 100% of the available CPU cycles.

Response Time: Response time of a process is the time at which the process finishes.

Critical Instant: The ***critical*** instant for a process is defined as the instant during execution at which the task has the largest response time.

- ❖ The proof using critical instants is easy while knowing the RMA when it is ready and all higher priority processes are also ready.
- ❖ Also critical instant is used to determine whether there is any feasible schedule for the system.
- ❖ Critical-instant analysis also implies that priorities should be assigned in order of periods.

Earliest –Dead line – First scheduling:

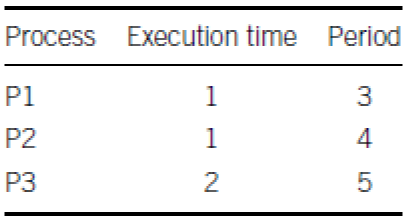
Write short notes on Earliest –Dead line – First scheduling.

- ***Earliest deadline first (EDF)*** is another well-known scheduling policy
- It is a dynamic priority scheme, it changes process priorities during execution based on initiation times.
- As a result, it can achieve higher CPU utilizations than RMS.
- The EDF policy is also very simple.
- It assigns priorities in order of deadline.
- The highest-priority process is the one whose deadline is nearest in time, and the lowest priority process is the one whose deadline is farthest away.
- Clearly, priorities must be recalculated at every completion of a process.
- The final step of the OS during the scheduling procedure is the same as for RMS and the highest-priority ready process is chosen for execution.

Example:

For *Earliest-deadline-first scheduling*

Consider the following processes:



Process	Execution time	Period
P1	1	3
P2	1	4
P3	2	5

The hyperperiod is 30. According to the above system the P1 has the highest priority, P2 is the middle priority and P3 is the lowest priority. Then the deadline table is written as

Time	Running process	Deadline
0	P1	
1	P2	
2	P3	
3	P3	
4	P1	P1
5	P2	P2
6	P1	P3
7	P3	P1
8	P3	
9	P1	P2
10	P2	P1
11	P3	P3
12	P1	
13	P3	P1, P2
14	P2	P2
15	P1	P2, P3
16	P2	
17	P3	P1
18	P1	P2 P3
19	Idle	
20	P3	P1
21	P2	
22	P1	P2,P3
23	P3	P1
24	P3	P2
25	P1	P3
26	P2	P1, P2
27	P2	P3
28	P1	P1
29	P3	P2, P3

The one time slot is idle at t =19 then the CPU utilization is 19/30. Hence the EDF is achieved nearly to 100% utilization of CPU.

Compare RMS versus EDF (NOV/DEC 2018)

RMS vs EDF

Compare Rate-Monotonic Scheduling and Earliest –Dead line – First scheduling

Which scheduling policy is better: RMS or EDF? That depends on the criteria.

- ❖ EDF can extract higher utilization out of the CPU, but it may be difficult to diagnose the possibility of an imminent overload.
- ❖ RMS achieves lower CPU utilization but it is easier to ensure that all deadlines. Able to diagnose the possibility of an imminent overload.

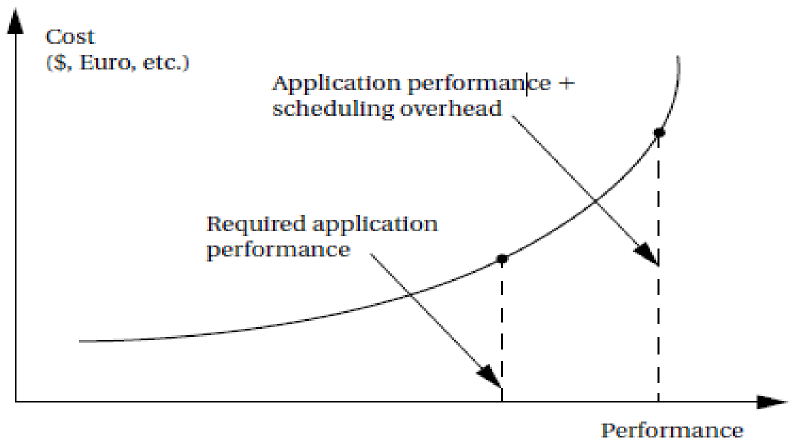
If a set of processes is unschedulable and we need to guarantee that they complete their deadlines? There are several possible ways to solve this problem:

- ***Get a faster CPU***. That will reduce execution times without changing the periods, giving you lower utilization.
- ***Redesign the processes to take less execution time***. This requires knowledge of the code and may or may not be possible.
- ***Rewrite the specification to change the deadlines***. This is unlikely to be feasible, but may be in a few cases where some of the deadlines were initially made tighter than necessary.

5.6. MULTIPROCESSOR:

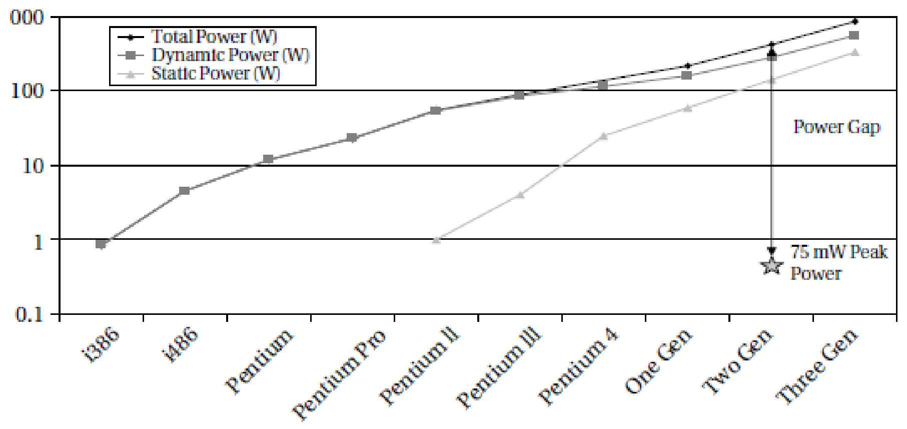
Write short notes on Multiprocessor (April 2010)

- A ***multiprocessor*** is, in general, any computer system with two or more processors coupled together.
- Multiprocessors used for scientific or business applications tend to have regular architectures: several identical processors that can access a uniform memory space.
- Embedded system designers must take a more general view of the nature of multiprocessors.
- The first reason for using an embedded multiprocessor is that they offer significantly better cost/performance that is, performance and functionality per dollar spent on the system
- The cost of a microprocessor increases greatly as the clock speed increases.
- Clock speeds are normally distributed by normal variations in VLSI processes; because the fastest chips are rare, they naturally command a high price in the marketplace.
- Because the fastest processors are very costly, splitting the application so that it can be performed on several smaller processors is usually much cheaper.
- Even with the added costs of assembling those components, the total system comes out to be less expensive.
- In addition to reducing costs, using multiple processors can also help with real time performance.
- It may take an extremely large and powerful CPU to provide the same responsiveness that can be had from a distributed system.
- Many of the technology trends encourage us to use multiprocessors for performance also lead us to multiprocessing for low power embedded computing.
- Some Processors running at slower clock rates consume less power than a single large processor: performance scales linearly with power supply voltage but power scales with V2.



Scheduling overhead is paid for at a nonlinear rate

- ➢ Austin *et al.* [Aus04] shows that general-purpose computing platforms are not keeping up with the strict energy budgets of battery-powered embedded computing.



Power consumption trends for desktop processors [Aus04].

- ➢ Desktop processors require close to 1000 times that amount of power to run.
- ➢ That huge gap cannot be solved by tweaking processor architectures or software.
- ➢ Multiprocessors provide a way to break this power barrier and build substantially more efficient embedded computing platforms.

5.7. INTERPROCESS COMMUNICATION MECHANISMS:

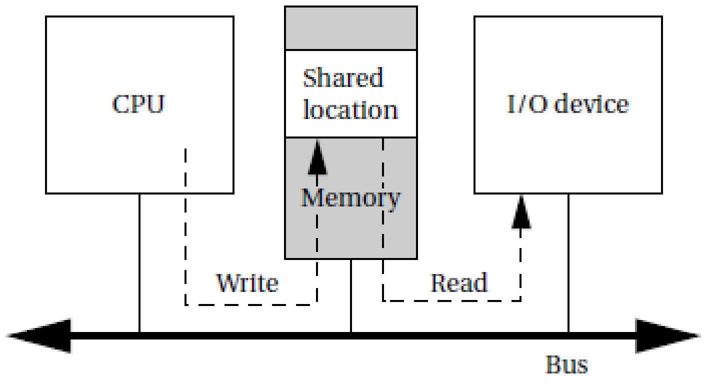
Explain in detail about Interprocess communication mechanisms.

(NOV/DEC 2010,May 2014, NOV 2017, April 2018, NOV/DEC 2018, APRIL/MAY 2019, APRIL/MAY 2023)

- ✓ Processes often need to communicate with each other. ***Interprocess communication mechanisms*** are provided by the operating system as part of the process abstraction.
- ✓ The process can send a communication in one of two ways: ***blocking*** or ***non-blocking***. **Blocking communication**: The process goes into the waiting state until it receives a response is called blocking communication.
- ✓ **Non-blocking communication**: It allows the process to continue execution after sending the communication. Both types of communication are useful.

There are two major styles of interprocess communication: ***shared memory*** and ***message passing***.

3.6.1 Shared Memory Communication:



Shared memory communication implemented on a bus.

- Shared memory communication works in a bus-based system.
- Two components, such as a CPU and an I/O device, communicate through a shared memory location.
- The software on the CPU has been designed to know the address of the shared location.
- The shared location has also been loaded into the proper register of the I/O device.
- If the CPU wants to send data to the device, it writes to the shared location.
- The I/O device then reads the data from that location. The read and write operations are standard and can be encapsulated in a procedural interface.
- If the CPU and the I/O device wants to communicate through a shared memory block. There must be a flag that tells the CPU when the data from the I/O device is ready.
- The flag, an additional shared data location, has a value of 0 when the data are not ready and 1 when the data are ready.
- If the flag is used only by the CPU, then the flag can be implemented using a standard memory write operation.
- If the same flag is used for bidirectional signaling between the CPU and the I/O device, care must be taken.

To care the flag following scenario must be followed:

1. CPU reads the flag location and sees that it is 0.

2. I/O device reads the flag location and sees that it is 0.

3. CPU sets the flag location to 1 and writes data to the shared location.

4. I/O device erroneously sets the flag to 1 and overwrites the data left by the CPU.

By used the bidirectional flag a critical timing race between the two programs is caused. To avoid this, the microprocessor bus must support an atomic test and set operation.

Test and Set operation:

- The test-and-set operation first reads a location and then sets it to a specified value. It returns the result of the test.
- If the location was already set, then the additional set has no effect but the test-and-set instruction returns a false result.
- If the location was not set, the instruction returns true and the location is in fact set. The bus supports this as an***atomic*** operation that cannot be interrupted. A test-and-set can be used to implement a ***semaphore***.

Semaphore:

- ✓ ***Semaphore*** is a language-level synchronization construct. Let's assume that the system provides one semaphore that is used to guard access to a block of protected memory.
- ✓ Any process that wants to access the memory must use the semaphore to ensure that no other process is actively using it.

Test-and-set operation example:

- ✓ The SWP (swap) instruction is used in the ARM to implement atomic test-and-set:

SWP Rd,Rm,Rn

- ✓ The SWP instruction takes three operands—the memory location pointed to by *Rn* is loaded and saved into *Rd*, and the value of *Rm* is then written into the location pointed to by *Rn*.
- ✓ When *Rd* and *Rn* are the same register, the instruction swaps the register's value and the value stored at the address pointed to by *Rd/Rn*. For example, consider this code sequence:

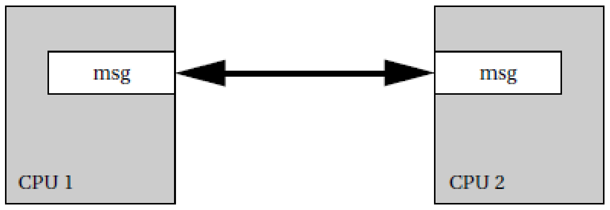
```
ADR r0, SEMAPHORE: get semaphore address
LDR r1, #1
GETFLAG SWP r1,r1, [r0]: test-and-set the flag
BNZ GETFLAG; no flag yet, try again
HASFLAG
```

- ✓ The program first loads the constant 1 into *r*1 and the address of the semaphore FLAG1 into register *r* 2, then reads the semaphore into *r*0 and writes the 1 value into the semaphore.
- ✓ The code then tests whether the semaphore fetched from memory is zero; if it was, the semaphore was not busy and we can enter the critical region that begins with the HASFLAG label.

- ✓ If the flag was nonzero, we loop back to try to get the flag once again.

5.7.1. Message Passing:

- ❖ Message passing communication complements the shared memory model each communicating entity has its own message send/receive unit.
- ❖ The message is not stored on the communications link, but rather at the senders/ receivers at the end points.
- ❖ In contrast, shared memory communication can be seen as a memory block used as a communication device, in which all the data are stored in the communication link/memory.

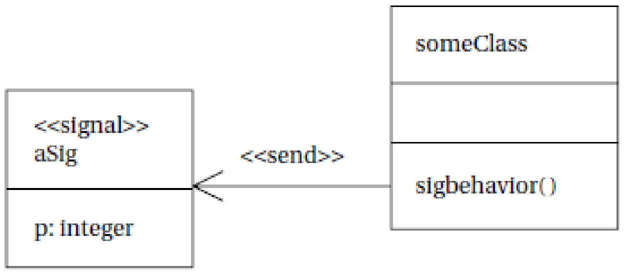


Message passing communication.

- ❖ Applications in which units operate relatively autonomously are natural candidates for message passing communication.
- ❖ For example, a home control system has one microcontroller per household device—lamp, thermostat, faucet, appliance, and so on.
- ❖ The devices must communicate relatively infrequently; furthermore, their physical separation is large enough that we would not naturally think of them as sharing a central pool of memory.
- ❖ Passing communication packets among the devices is a natural way to describe coordination between these devices.
- ❖ Message passing is the natural implementation of communication in many 8-bit microcontrollers that do not normally operate with external memory.

5.7.2. Signals

- Another form of interprocess communication commonly used in UNIX is the ***signal***.
- A signal is simple because it does not pass data beyond the existence of the signal itself.
- A signal is analogous to an interrupt, but it is entirely a software creation.
- A signal is generated by a process and transmitted to another process by the operating system.



Use of a UML signal.

- A UML signal is actually a generalization of the UNIX signal.
- While a UNIX signal carries no parameters other than a condition code, a UML signal is an object.
- The *sigbehavior* ( ) behavior of the class is responsible for throwing the signal, as indicated by <<*send*>>.The signal object is indicated by the <<*signal*>>stereotype.

to save and restore context and we must execute additional instructions to implement the scheduling policy.

- ❖ On the other hand, context switching can be implemented efficiently context, switching need not kill performance.
- ❖ The effects of nonzero context switching time must be carefully analyzed in the context of a particular implementation to be sure that the predictions of an ideal scheduling policy are sufficiently accurate.
- ❖ In most real-time operating systems, a context switch requires only a few hundred instructions, with only slightly more overhead for a simple real-time scheduler like RMS.
- ❖ When the overhead time is very small relative to the task periods, then the zero-time context switch assumption is often a reasonable approximation.
- ❖ Problems are most likely to manifest themselves in the highest-rate processes, which are often the most critical in any case.
- ❖ Completely checking that all deadlines will be met with nonzero context switching time requires checking all possible schedules for processes and including the context switch time at each preemption or process initiation

5.8.3. DISTRIBUTED EMBEDDED SYSTEMS

1. **Discuss in detail about the distributed embedded architecture. (Nov/Dec 2014, April 2018, NOV/DEC 2018)**
2. **Explain about distributed embedded architecture with suitable examples. (Apr/May2015) (Dec2022/Jan2023)**

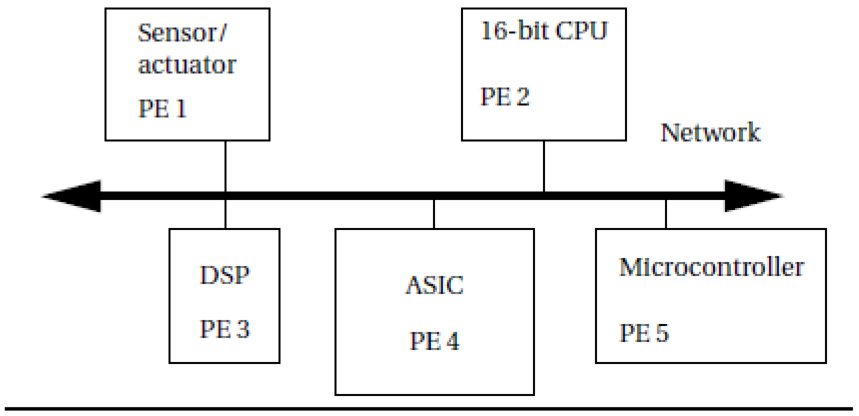
- Distributed system is more than two CPUs communicated in a tightly coupled manner.
- The main reason to implementing distributed system in the embedded field is it will provide high performance to perform complicated task in an easy manner, high processing rate.

Types

1. System Architecture
2. Software Architecture

- A distributed embedded system can be organized in many different ways, but its basic units are PE and the network processing element (PE) is either microprocessors or ASIC used to connect by a network.
- Processing element allows the network to communicate.
- A distributed embedded system can be organized in many different ways,
- A PE may be an instruction set processor such as a *DSP*, *CPU*, or *microcontroller*, as well as a nonprogrammable unit such as the *ASICs* used to implement *PE 4*.
- An I/O device such as *PE 1* (which we call here a ***sensor*** or ***actuator***, depending on whether it provides input or output) may also be a PE, so long as it can speak the network protocol to communicate with other PEs.
- The network in this case is a bus, but other network topologies are also possible.
- It is also possible that the system can use more than one network, such as when relatively independent functions require relatively little communication among them.
- We often refer to the connection between PEs provided by the network as a ***communication link***.
- The system of PEs and networks forms the ***hardware platform*** on which the application runs.
- In particular, PEs do not fetch instructions over the network as they do on the microprocessor bus

- The speed at which PEs can communicate over the bus would be difficult if not impossible to predict if we allowed arbitrary instruction and data fetches as we do on microprocessor buses.



An example of Distributed Embedded System

5.8.4. REASON FOR DISTRIBUTED SYSTEM IN EMBEDDED SYSTEM

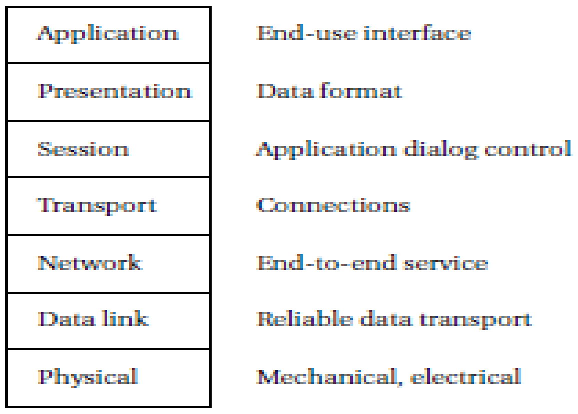
In distributed embedded system having several PEs and network it leads to more complicated than using a single large microprocessor to perform the same tasks.

1. In distributed systems the PEs communicates with physically separated manner.
2. Deadlines for processing the data are short.
3. It has more cost-effective performance
4. One part of the system can be used to help solve problems in another part.
5. Error identification is easier.
6. Several CPUs involved in the processing
7. One CPU use to generate inputs for another CPU and watch its output.

4.5.2. NETWORK ABSTRACTIONS

- Networks are complex systems. Ideally, they provide high-level services while hiding many of the details of data transmission from the other components.
- In order to help understand (and design) networks, the International Standards Organization has developed a seven-layer model for networks known as Open Systems Interconnection (OSI) models.
- Understanding the OSI layers will help us to understand the details of real networks.
- The seven layers of the **OSI model**, are intended to cover a broad spectrum of networks and their uses.

- Some networks may not need the services of one or more layers because the higher layers may be totally missing or an intermediate layer may not be necessary.
- However, any data network should fit into the OSI model.



Application	End-use interface
Presentation	Data format
Session	Application dialog control
Transport	Connections
Network	End-to-end service
Data link	Reliable data transport
Physical	Mechanical, electrical

The OSI layers from lowest to highest level of abstraction are described below.

✓ **Physical Layer**

- The physical layer defines the basic properties of the interface between systems, including the physical connections, electrical properties, basic functions of the electrical and physical components and the basic procedures for exchanging bits.

✓ **Data Link Layer**

- The primary purpose of this layer is error detection and control across a single link.
- If the network requires multiple hops over several data links, the data link layer does not define the mechanism for data integrity between hops, but only within a single hop.

✓ **Network Layer**

- This layer defines the basic end-to-end data transmission service. The network layer is particularly important in multi hop networks.
- This layer divides the message into packet form.

✓ **Transport Layer**

- The transport layer defines connection oriented services that ensures that data are delivered in the proper order and without errors across multiple links.
- This layer may also try to optimize network resource utilization.

✓ **Presentation Layer***:*

- Presentation layer defines data exchange formats and provides transformation utilities to application programs.

✓ **Session Layer**

- A session provides mechanisms for controlling the interaction of end user services across a network, such as data grouping and check pointing.

✓ **Application Layer**

- The application layer provides the application interface between the network and end-user programs.

- Simple embedded networks provide internet service that will implement the full range of functions in the OSI model.

NETWORKS FOR EMBEDDED SYSTEMS

- Networks for embedded computing span a broad range of requirements; many of those requirements are very different from those for general-purpose networks.
- Some networks are used in safety-critical applications, such as automotive control.
- Some networks, such as those used in consumer electronics systems, must be very inexpensive.
- Other networks, such as industrial control networks, must be extremely rugged and reliable.

Several interconnect networks have been developed especially for distributed embedded computing:

- ✓ The I^2C bus is used in microcontroller-based systems.
- ✓ The Controller Area Network (CAN) bus was developed for automotive electronics. It provides megabit rates and can handle large numbers of devices.
- ✓ Ethernet and variations of standard Ethernet are used for a variety of control applications

MPSOCS AND SHARED MEMORY MULTIPROCESSORS

5.9.5. Write short notes on MPSoCs and Shared memory multiprocessors (NOV 2017) (Dec2022/Jan 2023)

- Shared memory processors are well-suited to applications that require a large amount of data to be processed. Signal processing systems stream data and can be well-suited to shared memory processing.

- Most MPSoCs are shared memory systems. Shared memory allows for processors to communicate with varying patterns. If the pattern of communication is very fixed and if the processing of different steps is performed in different units, then a networked multiprocessor may be most appropriate.

- If the communication patterns between steps can vary, then shared memory provides that flexibility. If one processing element is used for several different steps, then shared memory also allows the required flexibility in communication.

Heterogeneous shared memory multiprocessors

- Many high-performance embedded platforms are heterogeneous multiprocessors.

- Different processing elements perform different functions. The PEs may be programmable processors with different instruction sets or specialized accelerators that provide little or no programmability.

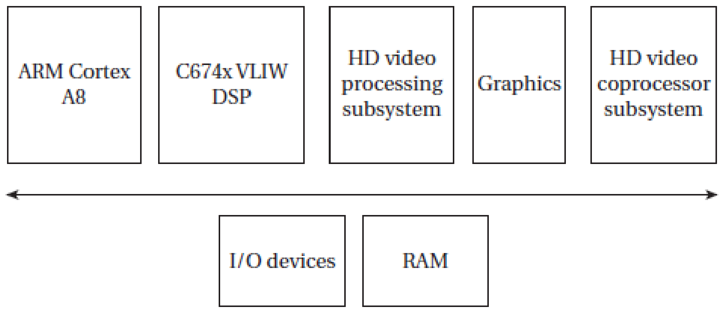
- In both cases, the motivation for using different types of PEs is efficiency. Processors with different instruction sets can perform different tasks faster and using less energy. Accelerators provide even faster and lower-power operation for a narrow range of functions.

Example: TI TMS320DM816x DaVinci

- The DaVinci816xis designed for high-performance video applications.
- It includes both a CPU, a DSP, and several specialized units: The 816x has two main programmable processors.
- The ARM Cortex A8 includes the Neon multimedia instructions. It is an in-order dual-issue machine.

  The C674x is a VLIWDSP. It has six ALUs and 64 general-purpose registers.

- The HD video coprocessor subsystem (HDVICP2) provides image and video acceleration.
- It natively supports several standards, such as H.264 (used in BluRay), MPEG-4, MPEG-2 and JPEG.



- It includes specialized hardware for major image and video operations, including transform and quantization, motion estimation, and entropy coding. It also has its own DMA engine.
- It can operate at resolutions up to 1080P /me at 60 frames/sec. The HD video processing subsystem (HDVPSS) provides additional video processing capabilities. It can process up to three high-definition and one standard-definition video streams simultaneously.
- It can perform operations such as scan rate conversion, Chroma key, and video security. The graphics unit is designed for 3D graphics operations that can process up to 30 M triangles/sec.

AUDIO PLAYER

5.9a.1. Design an Audio Player (NOV 2017)

- ❖ Audio players are defined as any media player which can only play audio files.
- ❖ Players capable of video playback are included under comparison of video player software, even if they are primarily well known for audio playback.

Theory of operation and requirements:

- ❖ Audio players are often called MP3 players.
- ❖ After the popular audio data format, a number of audio compression formats have been developed and are in regular use.
- ❖ The earliest portable MP3 players were based on compact disc mechanisms and modern MP3 players use either flash memory or disk drivers to store music.
- ❖ **Functions:**

  An MP3 player performs three basic functions such as

  1. Audio storage
  2. Audio compression
  3. User Interface
- ❖ **Audio decompression:** It is relatively light weight. The incoming bit stream has been encoded using a Huffmann style code, which must be encoded. The audio data itself is applied to reconstruction filter, along with a new other parameters.
- ❖ **Audio compression:** It is a lossy process that relies on perceptual coding. The coder eliminates certain features of the audio stream so that the result can be encoded in fewer bits. It tries to eliminate features that are not easily perceived by human audio system.

❖ **Masking:** It is one perceptual phenomenon that is exploited by perceptual coding. One tone can be masked by another if the tones are sufficiently close in frequency. Some audio features can also be masked if they occur too close in time after another feature.

MPEG layer 1 encoder

Filter bank: It splits the signal into set of 32 sub bands that are equally spaced in the frequency domain and together cover the entire frequency range of the audio.

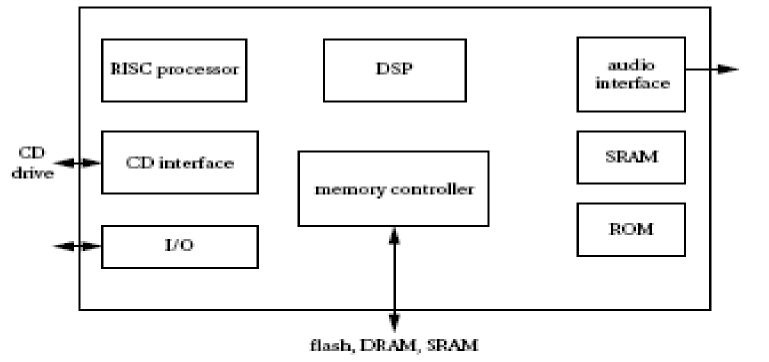


FIGURE 7.22

Architecture of a Cirrus audio processor for CD/MP3 players.

❖ **Encoder:** Audio signals tend to be more correlated within a narrower band, so splitting into sub bands help the encoder reduce the bit rate.

❖ **Quantizer**: It scales each sub band so that it fits within 6 bits of dynamic range, then quantizes based upon the current scale factor for that sub band.

❖ **Masking model**: It selects the scale factors. It driven by a separate Fast Fourier Transform (FFT), the filter bank could be used for masking; a separate FFT provides better results.

❖ **Multiplexer**: The multiplexer at the output of the encoder passes along all the required data.

MPEG Layer 1 decoder:

 MPEG audio decoding is a straight forward process.

 After disassembling the data frame, the data are unscaled and inverse quantized to produce sample streams for the sub band.

 An inverse filter bank reassembles the sub bands into the uncompressed data.

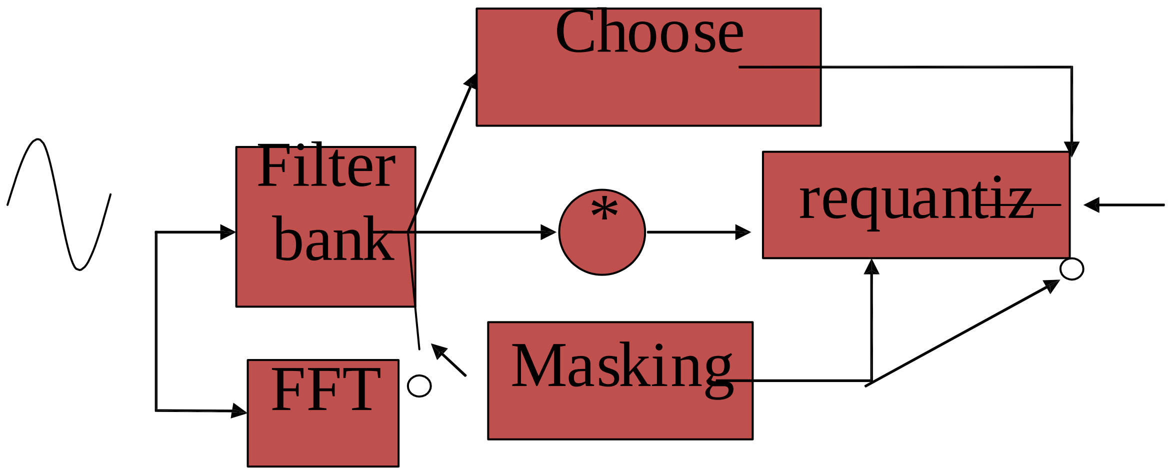


Fig MPEG layer 1 encoder

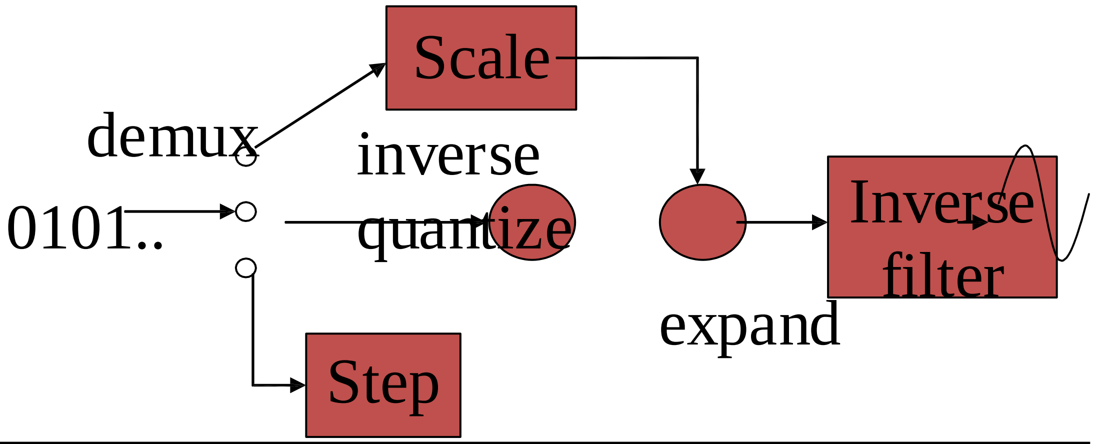


Fig.MPEG layer 1 decoder

User interface: The user interface of an MP3 player is usually kept simple to minimize both physical size and power consumption of the device. Many players provide only a simple display and a few buttons.

File system: The file system of the player must be **compatible** with PCs. The CD/MP3 players used compact discs that had been created on PCs. Today's players can be plugged into USB ports and treated as disk drivers on the host processor.

<u>Specification:</u>

- ✓ The file ID class is an abstraction of a file in the flash file system. The controller class provides the method that operates the player.

✓ The file management is performed on a host device, then the basic operations to be specified are simple.

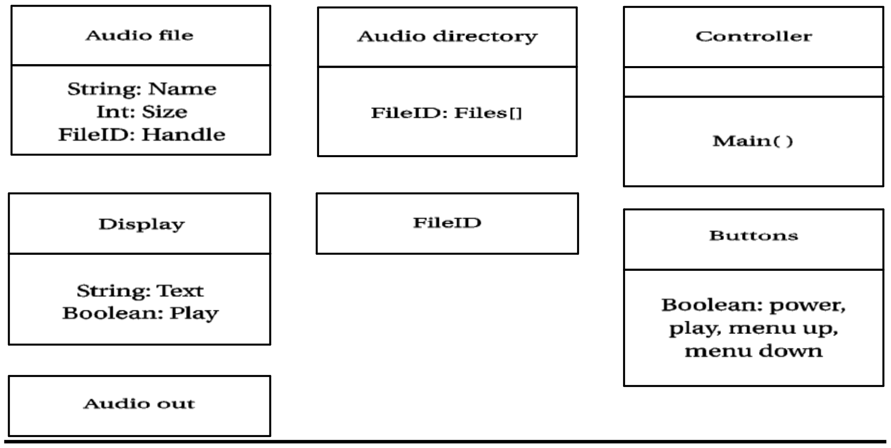


Fig. Classes in the audio player

State diagram for file display/Selection:

This specification assumes that all files are in the root directory and the files are playable audio.

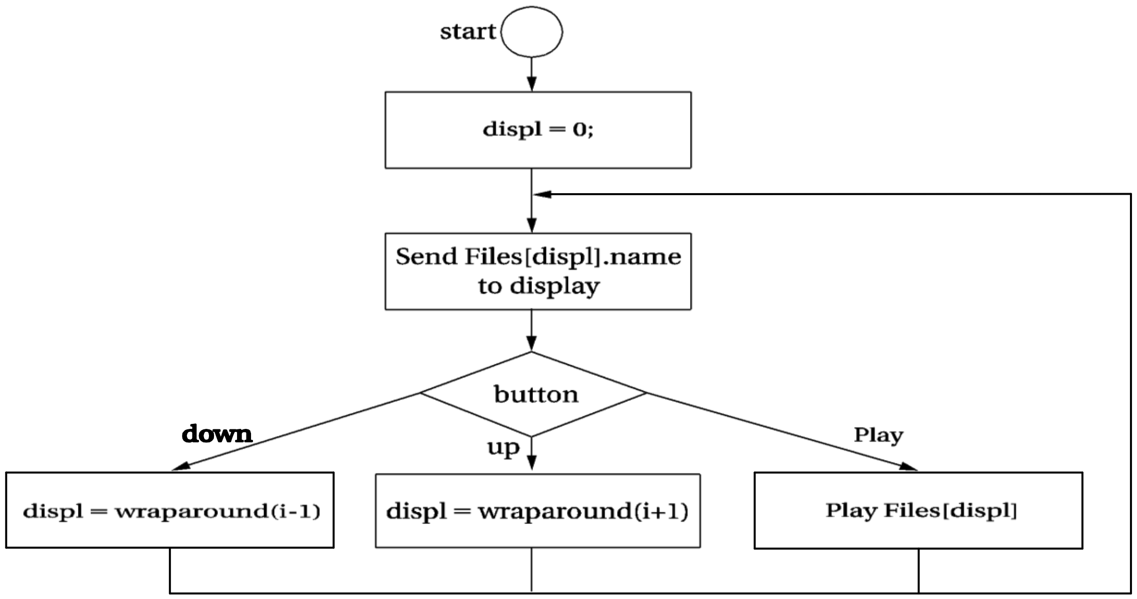
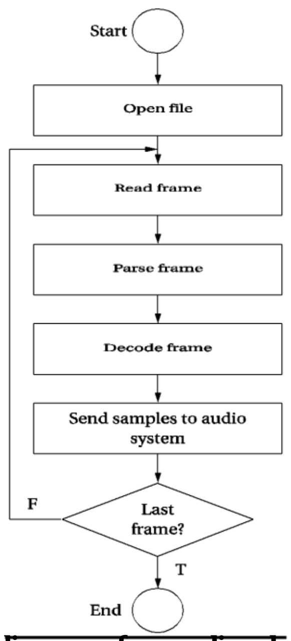


Fig. State diagram for file display & selection

State diagram for audio playback:

✓ The details of this operation depend on the format of the audio file.

- ✓ This state diagram refers to send the samples to the audio system rather than explicitly sending them because playback and reading the next data frame must be overlapped to ensure continuous operation.



State diagram for audio playback

System Architecture:

The cirrus CS7410 is an audio controller designed for CD/MP3 players. The audio controller includes two processors.

1. The 32 bit RISC processor is used to perform system control and audio decoding.
2. The 16 bit DSP is used to perform audio effects such as equalization.

The memory controller can be interfaced top several different types of memory such as

1. Flash
2. DRAM
3. SRAM

- ❖ Flash memory can be used for data or code storage and DRAM can be used as a buffer to handle temporary disruptions of the CD data stream.
- ❖ The audio interface units puts out video in formats that can be used by A/D converters.
- ❖ General purpose I/O pins can be used to decode buttons ,run displays, etc.,
- ❖ Cirrus provides Reference design for a CD/MP3 player.

<u>Components design and Testing:</u>

- The audio decompression object can be implemented from existing code or created as new software.
- In case of an audio system that does not confirm to a standard, it may be necessary to create an audio compression program to a create test files.

<u>System integration and debugging:</u>

- System integration and debugging process ensuring that audio plays smoothly and without interruption.
- Any file access and audio output operations are tested separately using recognizable test signal.

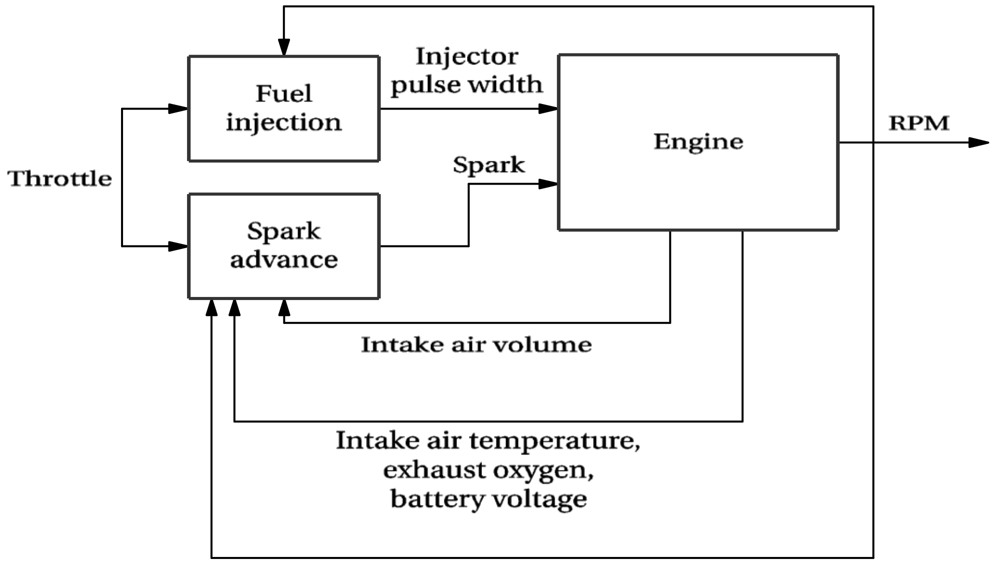
5.9a.4. ENGINE CONTROL UNIT (ECU)

Explain in detail the design of engine control unit (NOV 2017, NOV/DEC 2018, APRIL/MAY2019)/ Multitasking capacity of RTOS helps in engine control automation(DEC2022/JAN2023)

- Engine Control Unit (ECU) is a generic term for any embedded system that controls one or more of the electrical system or subsystems in a motor vehicle.
- An Engine Control Unit (ECU) is a type of electric control unit that controls a series of actuator on an internal combustion engine to ensure optimal engine performance.
- It does this by reading values from a multitude of sensors within the engine body, interpreting the data using multidimensional performance maps called Lockup tables and adjusting the engine actuators accordingly.
- Engine control unit controls the operation of a fuel injected engine based on several measures taken from the running engine.

THEORY OF OPERATION AN REQUIREMENTS

- Design a basic engine controller for a simple Fuel injected engine. The block diagram of engine is shown in below figure.
- The Throttle is the command input. The engine measures throttle. RPM intake air volume, and other variables.



The engine controller computes injector pulse width and spark. This doesn't compute all the outputs required by a real engine.

Requirements

Requirements for the engine Control unit shown in the below figure.

Name	**ECU**
Purpose	Engine controller for fuel-injected engine
Inputs	Throttle, RPM, intake air volume, intake manifold pressure
Outputs	Injector pulse width, spark advance angle.
Functions	Compute injector pulse width and spark advance angle as a function of throttle, RPM, intake air volume, intake manifold pressure.
Performance	Injector pulse updated at 2-ms period, spark advance angle updated at 1-ms period.
Manufacturing Cost	Approximately $ 50
Power	Power by engine generator
Physical size and weight	Approx. 4 in x 4 in, less than 1 pound

Fig. Requirements for the engine controller

SPECIFICATION

❖ The engine controller must deal with processes that happen at different rates. Below figure shows the updates periods for the different signals.
❖ Use NE and T to represent the change in RPM and throttle position, respectively. Our controller computes two output signals, injector pulse width PW and spark advance angle S.

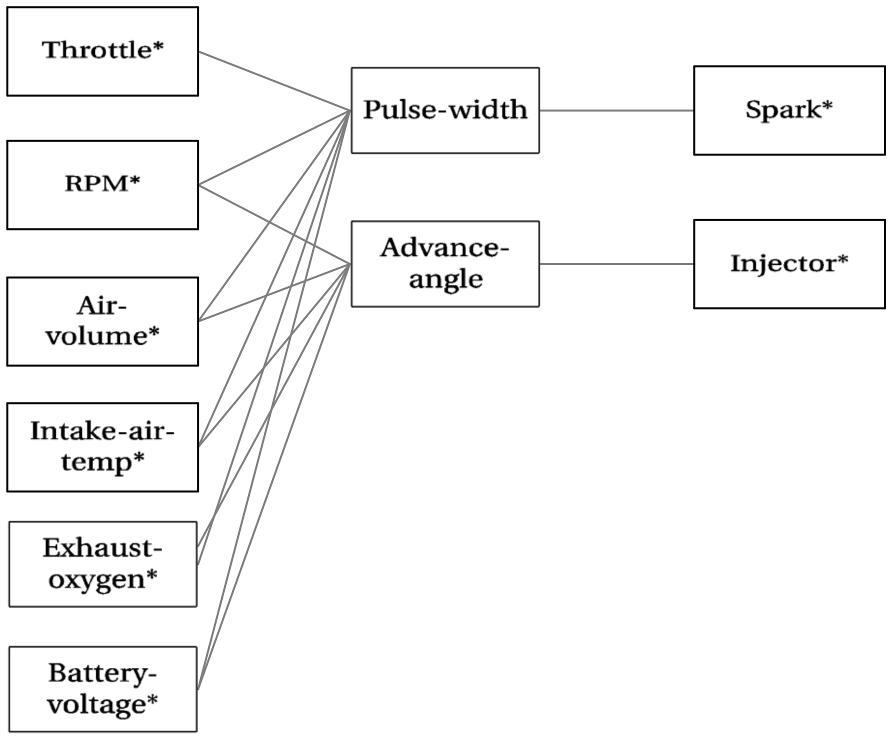
The Controller then applies corrections to these initial values:

- As the intake air temperature (THA) increases during the engine warm-up, the controller reduces the injection duration.
- As the throttle opens, the controller temporarily increases the injection frequency.
- The Controller adjusts duration up or down based upon readings from the exhaust oxygen sensor (OX).
- The injection duration is increased as the battery voltage (+B) drops.

Signal	Variable name		
Throttle		input	
		input	
Intake air volume		input	
		output	
		output	
Intake air temperature		input	
Exhaust oxygen		input	
Battery voltage		input	

SYSTEM ARCHITECTURE

Below figure shows the class diagram for the engine controller. The two major process, pulse-width and advance-angle, compute the control parameters for the sparkplugs and injectors.

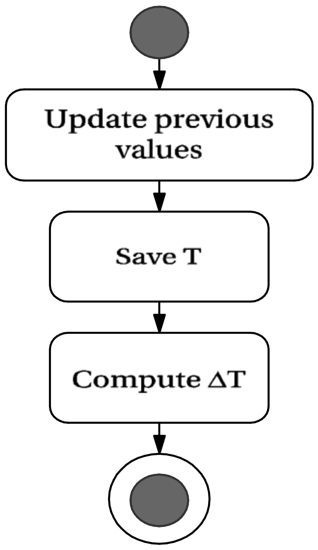


Class diagram for the engine controller

- ❖ The Control Parameters rely on charges in some of the input signals. Use the physical sensor classes to compute these values.
- ❖ Each change must be updated at the variables sampling rate. The update processes is simplified by performing it in a task runs at the required update rate.

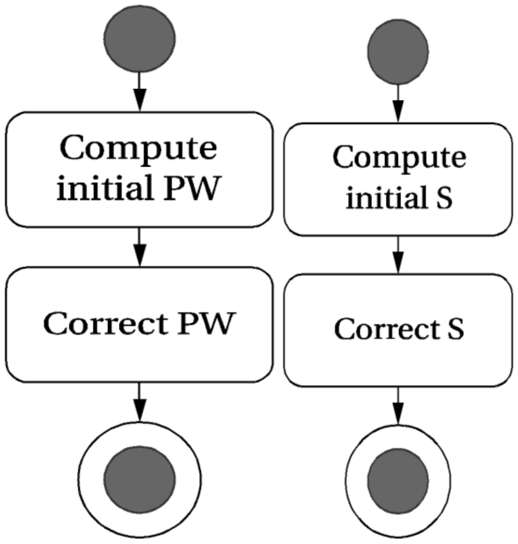
State diagram

- ❖ State diagram for throttle position sensing is shown in below figure. It saves both the current value and change in value of the throttle.
- ❖ Use similar control flow to compute changes to the other variables.



Throttle position sensing state diagram

Below figure shows the state diagram for injector pulse width and spark advance angle. In each case the value is computed in two stages, first an initial value followed by a correction.



- ❖ The pulse width and advance angle processes do not generate the waveforms to drive the spark and injector waveforms.
- ❖ These waveforms must be carefully timed to the engine's current state.
- ❖ Each spark plug and injector must fix at exactly the right time in the engine cycle, taking into account the engine's current speed as well as the control parameters.

- Some engine controller platforms provide Hardware units that generate high rate, changing waveforms.
- For example consider MPC5602D. Then main processor is a power PC processor. The enhanced modular I/O subsystem provides 28 input and output channels controlled by Timers.
- Each channel can perform a variety of functions.
- The output pulse width and frequency modulation buffered mode will automatically generate a waveform whose period and duty cycle can be varied by writing registers in the enhanced modular I/) subsystems.
- The details of the waveform timing are handled by the output channel hardware.
- Because these objects must be updated at different rates, their execution will be controlled by an RTOS. Depending on the RTOS Latency, separate the I/O functions into interrupt service handlers and threads.

COMPONENT DESIGN AND TESTING

- The various tasks must be coded to satisfy the requirements of RTOS processes.
- Variables that are maintained across task execution must be allocated and saved in appropriate memory locations.
- The RTOS initialization phase is used to set up the task periods.
- Because some of the output variables depend on changes in states, these tasks should be tested with multiple input variables sequences to ensure that both the basic and adjustment calculations are performed correctly.

SYSTEM INTEGRATION AND TESTING

Engine generates huge amounts of electrical noise that can cripple digital electronics. They also operate over very wide temperature ranges.

1. Hot during engine operation
2. Very cold before the engine is started.

Any testing performed on an actual engine must be conducted using an engine controller that has been designed to withstand the harsh environment of the engine compartment.

5.9a.5. VIDEO ACCELERATOR

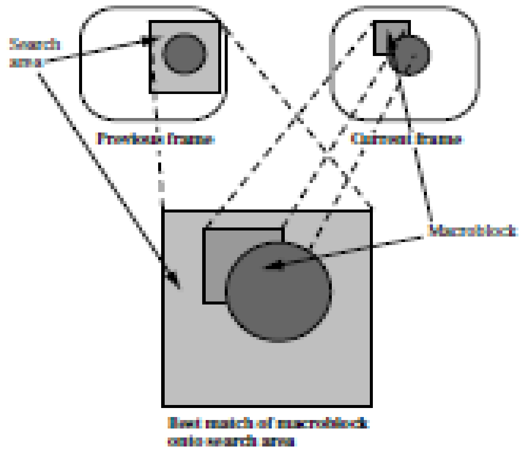
Discuss in detail about embedded concepts in the design of video accelerator (8 Marks) Nov/Dec 2016 or Illustrate the working of video player (NOV/DEC 2018) or Illustrate video accelerator using UML methodology (NOV/DEC 2018) (April/May 2019) (Dec2022/Jan2023) (April/May 2023)

A video accelerator is a hardware circuit on a display adapter that speed up full motion video, which also frees the CPU to take care of other tasks. Motion estimation engines are used in real-time search engines; we may want to have one attached to our personal computer to experiment with video processing techniques.

Algorithm and Requirements

- Block motion estimation is used in digital video compression algorithms so that one frame in the video can be described in terms of the differences between it and another frame.
- Because objects in the frame often move relatively little, describing one frame in terms of another greatly reduces the number of bit
- The goal is to perform a two-dimensional correlation to find the best match between regions in the two frames.
- We divide the current frame into *macroblocks*.
- We want to find the region in the previous frame that most closely matches the macroblock.
- Searching over the entire previous frame would be too expensive, so we usually limit the search to a given area, centered around the macroblock and larger than the macroblock.
- Intensity is measured as an 8-bit luminance that represents a monochrome pixel—color information is not used in motion estimation.
- We choose the macroblock position relative to the search area that gives us the smallest value for this metric.

- The offset at this chosen position describes a vector from the search area center to the macroblock's center that is called the *motion vector*.
- For simplicity, we will build an engine for a full search, which compares the macroblock and search area at every possible point.



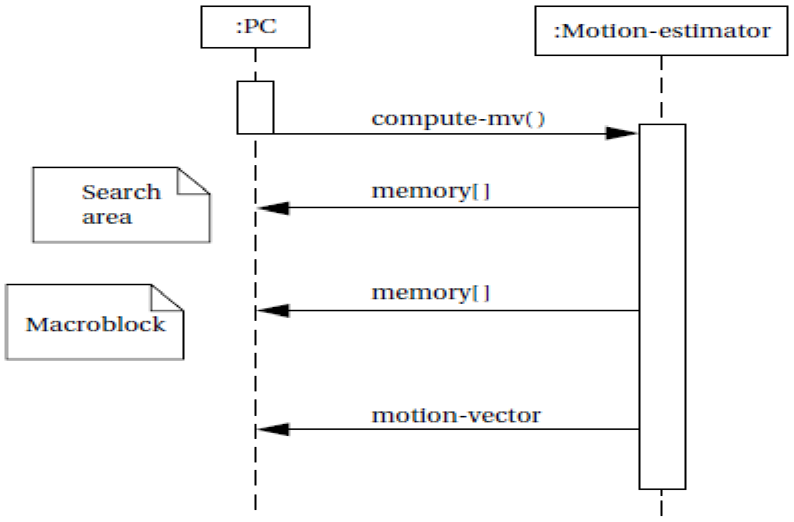
Requirements:

Name:	Block motion estimator
Purpose:	Perform block motion estimation within a PC system
Inputs:	Macroblocks and search areas
Outputs:	Motion vectors
Functions:	Compute motion vectors using full search algorithms
Performance:	As fast as we can get
Manufacturing cost:	Hundreds of dollars
Power:	Powered by PC power supply
Physical size:	Packaged as PCI card for PC

Specification

- The specification for the system is relatively straightforward because the algorithm is simple.
- Because the behavior is simple, we need to define only two classes to describe it: the accelerator itself and the PC.

- The PC makes its memory accessible to the accelerator.
- The accelerator provides a behavior compute-mv( ) that performs the block motion estimation algorithm.
- After initiating the behavior, the accelerator reads the search area and macroblock from the PC; after computing the motion vector, it returns it to the PC.



Sequence diagram of accelerator

Architecture

- The accelerator will be implemented in an FPGA on a card connected to a PC's PCI slot.
- Such accelerators can be purchased or they can be designed from scratch.
- If you design such a card from scratch, you have to decide early on whether the card will be used only for this video accelerator or if it should be made general enough to support other applications as well.

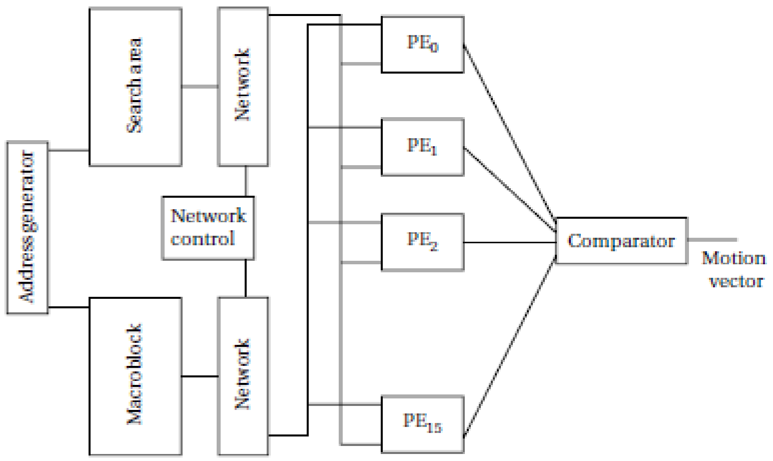


FIGURE 7.31

An architecture for the motion estimation accelerator [Dut96].

Component Design

- If we want to use a standard FPGA accelerator board to implement the accelerator, we must first make sure that it provides the proper memory required for *M* and *S*.
- Once we have verified that the accelerator board has the required structure, we can concentrate on designing the FPGA logic.
- Designing an FPGA is, for the most part, a straightforward exercise in logic design. Because the logic for the accelerator is very regular, we can improve the FPGA’s clock rate by properly placing the logic in the FPGA to reduce wire lengths.
- If we are designing our own accelerator board, we have to design both the video accelerator design proper and the interface to the PCI bus.
- We can create and exercise the video accelerator architecture in a hardware description language like VHDL or Verilog and simulate its operation.
- Designing the PCI interface requires somewhat different techniques since we may not have a simulation model for a PCI

System Testing

- Testing video algorithms requires a large amount of data. Luckily, the data represents images and video, which are plentiful.

- Because we are designing only a motion estimation accelerator and not a complete video compressor, it is probably easiest to use images, not video, for test data.
- You can use standard video tools to extract a few frames from a digitized video and store them in JPEG format.
- Open source for JPEG encoders and decoders is available. These programs can be modified to read JPEG images and put out pixels in the format required by your accelerator.
- With a little more cleverness, the resulting motion vector can be written back onto the image for a visual confirmation of the result. If you want to be adventurous and try motion estimation on video, open source MPEG encoders and decoders are also available

Internet – of – Things – Physical Design, Logical Design – IoT Enabling Technologies – DomainSpecific IoTs – IoT and M2M – IoT System Management with NETCONF – YANG – IoT PlatformDesign – Methodology – IoT Reference Model – Domain Model – Communication Model – IoTReference Architecture – IoT Protocols - MQTT, XMPP, Modbus, CANBUS and BACNet.

Internet – of – Things

The Internet of Things (IoT) is a network of connected devices that can communicate with each other, share data, and perform tasks without human intervention. The importance of communication in IoT cannot be overstated, as it is the foundation on which the entire system is built. The devices that make up the IoT ecosystem need to be able to communicate with each other in order to function properly and achieve their intended purpose.

Effective communication in IoT enables devices to share data, receive instructions, and respond to requests in a timely and accurate manner. This is critical for the successful implementation of IoT solutions across various industries, such as healthcare, manufacturing, transportation, and smart homes.`

For example, in a smart home, the communication between the devices (such as lights, thermostats, and security systems) allows them to work together to create a more convenient and secure living environment for the occupants. Similarly, in a healthcare setting, IoT devices can be used to monitor patients remotely and alert healthcare providers in case of an emergency, ensuring that timely medical intervention is provided.

Physical Design, Logical Design

While talking about the **logical and physical design of IoT** we are talking about the physical devices and the protocols that take data from one device to another. All of the work together as a single unit. Each of the physical devices is called a node and each device has its own unique identity with the help of protocol they do things like monitoring, sensing and tracking. IoT today are becoming immensely popular. Companies today are implementing IoT technologies more so if you want to be relevant to the tech industry you have to know one or two things about IoT sadly very few do this.

Physical Design Of IoT

Physical Devices/Things

Devices are physical electronic components that are used to build a connection to process data, provide interfaces that offer storage, graphics and storage and also power source sometimes in the IoT system. Most of these physical components collect data from the environment and send raw data to be processed and analyzed. After analyzing it sends the information to the actuators to act accordingly. Instead of collecting data from the environment some IoT also collects user data to provide more refined performance to the users.

For example, a small propeller sensor inside the tap of a wine barrel spins whenever the wine is poured from the tap. After that, the data of the number of times it spun gets sent to the analyzer and tells how much whine is spent and when to shut off the flow. The analyzing part gets done by the algorithms that were put into it. Here are some examples of physical components.

The Connectivity

Whatever physical device provides connectivity and either transmits or receives data come in the connectivity part or physical devices of IoT. Here it can be USB ports, Ethernet cables etc.

Processor

The second, essential component of the physical design of IoT will be the CPU or the processor. All the data processing happens here and it improves the decision making quality in the IoT system.

Sound & Visual

The third element is the visual component of IoT. It shows all the information that the processor sends to the screen. It uses things like HDMI and RCA. The video player is the audio and visual part of the physical design of IoT.

Storage Component

Not only a hard disc every component that stores data is the storage component of IoT. Things like SD, MMC and SDIO. It's different from the DDR and GPU is used to control the activity of an IoT system

Logical Design Of IoT

The "Logical Design" of IoT is the framework or the imaginary ideal design in which the components including software and the hardware components will be laid out. It doesn't go into the depth of describing how each component will be built with low-level programming specifics.

IoT Functional Blocks

What is a functional block?

An IoT system consists of a number of functional blocks that provide the system with the capabilities for identification, sensing, actuation, communication and management. The function of the Communication functional block in short **Handles the communication for the IoT system**.

Any IoT system will have several functional blocks like Devices, communication, security, services, and application. With the help of the functional blocks, we provide sensing, identification, actuation, management, and communication capability. These blocks also include the physical components too.

IoT communications models

There are endless options of models available in an IoT system. It connects the IoT system to the server. Here are some examples

Request-response model
Push-pull model
Publish-subscribe model
Exclusive pair model

IoT Communication API

In simpler terms APIs are used to communicate between the server and the system in IoT. Some API includes.

REST-basedcommunicationAPIs
Client-server
Stateless
Cacheable
Websocket based communication API

The IoT Protocols

In simpler terms, IoT protocols are a number of procedures or sets of rules that decides how data will be transmitted between two devices generally those two computers there are many protocols each of them works differently than others on how any data will be structured and how it will be sent between devices and it will be received. There are basically 11 types of protocols for IoT. We have an article dedicated completely to the IoT protocols. Do have a read if you want to know about the IoT protocols in detail, go check it out. There are 4 types of

- Bluetooth
- 6LowPAN
- Zigbee
- Z-Wave
- WiFi
- Cellular
- Thread
- NFC
- Neul
- LoRaWAN

IoT(internet of things) enabling technologies

Wireless Sensor Network

Cloud Computing

Big Data Analytics

Communications Protocols

Embedded System

1. Wireless Sensor Network(WSN) :

A WSN comprises distributed devices with sensors which are used to monitor the environmental and physical conditions. A wireless sensor network consists of end nodes, routers and coordinators. End nodes have several sensors attached to them where the data is passed to a coordinator with the help of routers. The coordinator also acts as the gateway that connects WSN to the internet.

Example –

Weather monitoring system

Indoor air quality monitoring system

Soil moisture monitoring system

Surveillance system

Health monitoring system

2. Cloud Computing :

It provides us the means by which we can access applications as utilities over the internet. Cloud means something which is present in remote locations.

With Cloud computing, users can access any resources from anywhere like databases, webservers, storage, any device, and any software over the internet.

Characteristics –

Broad network access

On demand self-services

Rapid scalability

Measured service

Pay-per-use

Provides different services, such as –

IaaS (Infrastructure as a service)

Infrastructure as a service provides online services such as physical machines, virtual machines, servers, networking, storage and data center space on a pay per use basis. Major IaaS providers are Google Compute Engine, Amazon Web Services and Microsoft Azure etc.

Ex : Web Hosting, Virtual Machine etc.

PaaS (Platform as a service)

Provides a cloud-based environment with a very thing required to support the complete life cycle of building and delivering West web based (cloud) applications – without the cost and complexity of buying and managing underlying hardware, software provisioning and hosting. Computing platforms such as hardware, operating systems and libraries etc. Basically, it provides a platform to develop applications.

Ex : App Cloud, Google app engine

SaaS (Software as a service)

It is a way of delivering applications over the internet as a service. Instead of installing and maintaining software, you simply access it via the internet, freeing yourself from complex software and hardware management.

SaaS Applications are sometimes called web-based software on demand software or hosted software.

SaaS applications run on a SaaS provider's service and they manage security availability and performance.

Ex : Google Docs, Gmail, office etc.

3. Big Data Analytics :

It refers to the method of studying massive volumes of data or big data. Collection of data whose volume, velocity or variety is simply too massive and tough to store, control, process and examine the data using traditional databases.

Big data is gathered from a variety of sources including social network videos, digital images, sensors and sales transaction records.

Several steps involved in analyzing big data –

Data cleaning

Munging

Processing

Visualization

Examples –

Bank transactions

Data generated by IoT systems for location and tracking of vehicles

E-commerce and in Big-Basket

Health and fitness data generated by IoT system such as a fitness bands

4. Communications Protocols :

They are the backbone of IoT systems and enable network connectivity and linking to applications. Communication protocols allow devices to exchange data over the network. Multiple protocols often describe different aspects of a single communication. A group of protocols designed to work together is known as a protocol suite; when implemented in software they are a protocol stack.

They are used in

Data encoding

Addressing schemes

5. Embedded Systems :

It is a combination of hardware and software used to perform special tasks.

It includes microcontroller and microprocessor memory, networking units (Ethernet Wi-Fi adapters), input output units (display keyword etc.) and storage devices (flash memory).

It collects the data and sends it to the internet.

Embedded systems used in

Examples –

Digital camera

DVD player, music player

Industrial robots

Wireless Routers etc.

Types of Communication Models in IoT

There are several communication models that can be used in the Internet of Things (IoT) ecosystem, depending on the requirements of the use case. The three main communication models used in IoT are −

Depending upon of its usages, the software may be classified as generic or specific. Generic software is a software that can perform multiple tasks in a different environment without being modified like a word processor software that can be used by anyone to make different types of documents as a report, whitepaper, training material, etc. Specific software is software for a particular application, like railway reservation system, weather forecasting, etc.

Some Domain Specific Tools :

School Management System : School management system handles various activities and processes of a school to facilitate campus management like examination, attendance, admission, student’s fees, timetable, teacher’s training, etc. It provides a healthy interaction among teachers, students, parents.

Inventory Management : Managing multiple tasks like purchase, sales, order, delivery, stock maintenance, etc. associated with raw or processed goods in any business is called inventory management. The inventory management software ensures that stocks are never below specified limits and purchase/deliveries are done in time. Inventory management system is very useful for forecasting, utilizing economies of scale and timing.

Payroll Management System : Payroll management system deals with the financial aspects of the employee’s salary, taking care of leaves, bonus, loans, etc. Some advantages of using this kind of management system are managed employee information efficiently, generate pay-slip at the convenience of a mouse click, manages its own security. Payroll software is generally a component of HR (Human Resource) management software in big organizations.

Employee Definition

|

Salary Structure

|

Pay Element

|

Tax Details Recording

|

Leave and Time Sheet Booking

|

Employee Appraisal

|

Employee Payroll Generation

|

Salary Payment

Block diagram for Salary Payment Process

Financial Accounting : Financial management software keeps an electronic record of all financial transactions of the organization. Objectives of financial accounting

Record financial transactions as and when they occur so that the data can be analyzed for preparing a financial statement.

Calculate profit or loss, to enable management to take course-correction strategies if required.

Ascertain the financial strength of the company by determining its assets and liabilities.

Communicate the information to stakeholders through statements and reports, so that these stakeholders can take appropriate decisions on their investments in the business.

Hotel Management :Hotel management software helps hotel managers to keep track of inventory levels, daily orders, customer management, employee scheduling, table booking, etc.

Reservation System :A reservation system is a software that handles multiple modules like train routes, train management, seat booking, meal booking, train maintenance, train status, travel package, etc.

Weather Forecasting System : Weather forecasting system is a real-time software that predicts the weather of a place by collecting live data about atmospheric temperature, humidity, wind level, etc. It is used to predict major disasters like earthquakes, hurricanes, tsunamis, etc.

IoT applications span a wide range of domains like:

Home Automation

Smart Cities

Environment

Energy systems

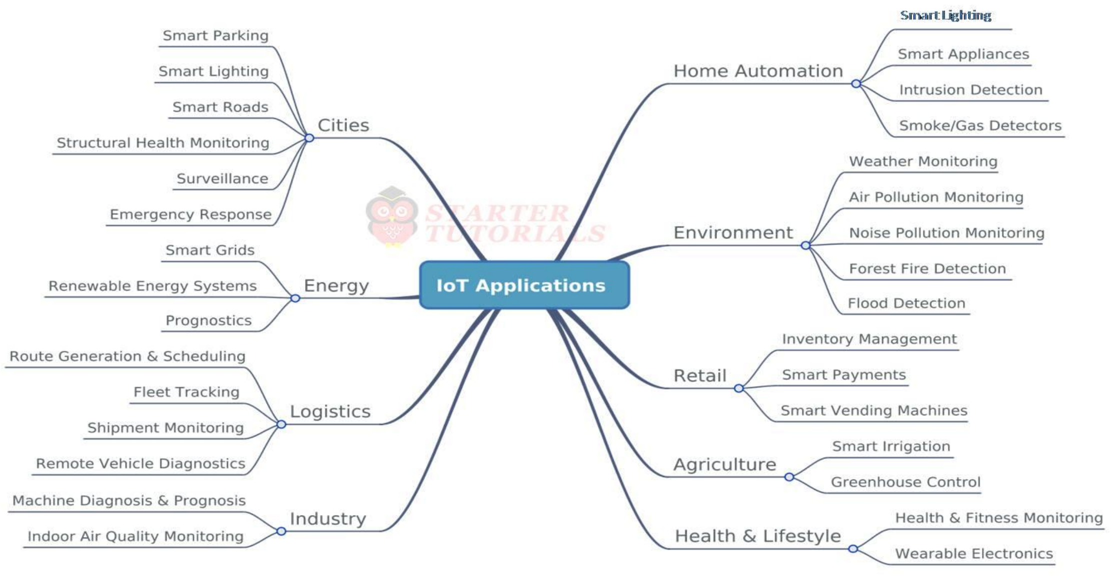
Retail

Logistics

Industry

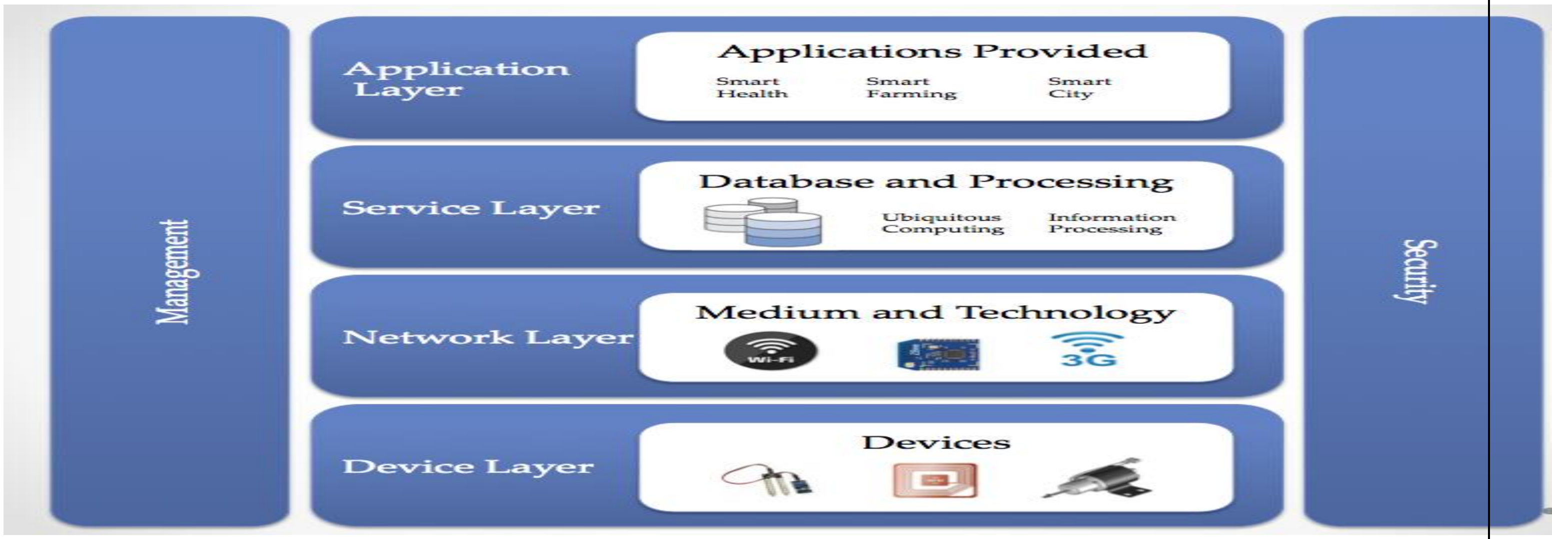
Agriculture

Health



IoT reference model

Just like the OSI reference model for the internet, IoT architecture is defined through six layers: four horizontal layers and two vertical layers. The two vertical layers are **Management** and **Security** and they're spread over all four horizontal layers, as seen in the following diagram:



IoT layers

The **Device Layer**: At the bottom of the stack, we have the device layer, also called the **perception layer**. This layer contains the physical things needed to sense or control the physical world and acquire data (that is, by perceiving the physical world). Existing hardware, such as sensors, RFID, and actuators, constitutes the perception ...

What is IoT Reference Architecture?

IoT (Internet of Things) reference architecture is a framework that provides a common understanding of the key components and their interactions in an IoT system. It serves as a blueprint for designing and implementing IoT systems and helps ensure interoperability, scalability, and security.

The IoT reference architecture typically consists of the following layers:

Perception layer: This layer consists of sensors, actuators, and other devices that collect and generate data from the physical environment. These devices are responsible for sensing and measuring various physical parameters, such as temperature, humidity, pressure, and motion.

Network layer: This layer consists of various networking technologies, such as Wi-Fi, Bluetooth, and cellular networks, that enable devices to communicate with each other and with cloud-based servers. This layer also includes various networking protocols, such as MQTT and CoAP, that are optimized for machine-to-machine communication.

Middleware layer: This layer provides various services and functions that enable data processing, storage, and analysis. It includes various platforms, such as cloud-based platforms and edge computing platforms, that provide data processing, analytics, and storage services.

Application layer: This layer consists of various applications and services that use the data generated by IoT devices to provide value to end-users. These applications and services can be used in a wide range of domains, such as smart homes, industrial automation, healthcare, transportation, and energy management.Security layer: This layer provides various security mechanisms, such as authentication, authorization, and encryption, to ensure the confidentiality, integrity, and availability of data in an IoT system. It also includes various security protocols, such as TLS and DTLS, that are used to secure data transmission between devices and cloud-based servers.

IOT Protocols

The Internet of Things (IoT) is about the network of sensor devices to the web in real-time. IoT devices communicate with each other over the network, so certain standards and rules need to be set to determine how data is exchanged. These rules are called IoT Network Protocols. Today, a wide variety of IoT devices are available, and therefore different protocols have been designed.

Depending on the IoT application's functionality, its workflow or architecture varies. Basic architecture involves four layers, i.e., the Sensing layer, Network layer, Data processing layer, and Application layer.

The Sensing layer contains all the hardware, like sensors, actuators, chips, etc., that collect information. This layer is connected to the successive layer, which is the network layer, through protocols. The Network layer allows communications among devices using network protocols like cellular, Wi-Fi, Bluetooth, Zigbee, etc. The data collected by IoT devices is processed in the Data processing layer using technologies like data analytics and machine learning algorithms. This processed data can be displayed to the user through web portals, apps, or interfaces provided by the application layer. Users can directly interact and visualize the data obtained from IoT devices through these interfaces.

As IoT devices have very few components-little batteries and sensors, there is a small amount of power available. Hence, it is tough to design protocols for IoT. Also, we need to perform everything (construct topological structures, do address assignments, etc.) on wireless.

IoT Protocols Should also Satisfy These Requirements

- Allow communication among various devices simultaneously.
- IoT is being used in critical areas like health, industries, home surveillance, etc. hence communication security needs to be ensured.
- Transport data efficiently.
- IoT devices can be added or removed from the IoT network. Hence protocols must provide scalability.

There are many such protocols developed for IoT, then how to choose one??

One way to decide which protocol to use is to consider the environment for which these protocols are designed. Some are designed for small ranges; some are for wide ranges, high data rates, low data rates, etc. They vary based on power consumption, range, cost, data rate, etc.

Short Range Communication, Low Data Rate, Low Power

Bluetooth

Bluetooth works in a frequency range of 2.4GHz. It covers a range of 10m to 100m, and its data rate goes up to 1MBPS. It supports two network topologies – point-to-point and mesh. It is suitable to send a small amount of data to personal devices like speakers, earphones, smart watches, smart shoes, etc. This protocol can also be used for Smart Homes, including Alarms, HVAC, lighting, etc.

Zigbee

This is based on the IEEE802.15.4 standard. Its frequency range is the same as that of Bluetooth, which is 2.4GHz. Its range is up to 100 meters, and the data rate is a maximum of 250KBPS. Zigbee protocol can transmit small amounts of data within a short range. This can be used in systems that require high authentication and robustness. It supports star topology, mesh topology, and cluster tree topology. Major applications observed are sensing device health in industries, smart homes, etc.,

6LoWPAN

PAN stands for Personal Area Network, and 6LoWPAN refers to IPV6 Low Power PAN. It works in a frequency ranging from 900 to 2400MHz. The data rate is 250KBPS, supporting two network topologies - star and mesh.

Short Range Communication, High Data Rate

WirelessLAN - Wi-Fi

Wi-Fi has high bandwidth and allows a data rate of 54MBPS and goes up to 600MBPS. Covers a range of 50m in the local area where providing private antennas goes to 30 km. IoT devices can be easily connected using Wi-Fi and share a large amount of data. This protocol is used in smart homes, smart cities, offices, etc

Long Range Communication, High Data Rate, Low power

LoRaWAN

This stands for Long Range Wide Area Network. Its range is approximately 2.5km and can go up to 15km. The data rate is very low, which is 03, and KBPS and goes up to a maximum of 50KBPS. It can support many connected devices and is used in applications like Smart City, Supply Chain Management, etc.

LTE-M

LTE-M stands for Long Term Evolution for Machines. This is a type of LPWAN – Low Power Wide Area Network. This is used along with cellular networks to provide security. LTE-M works in a frequency range of 1.4MHz-5MHz, and the data rate can go up to 4MBPS.

Long Range, Low Data Rate, Low Power Consumption

Sigfox

Sigfox is used when wide area coverage is required with minimum power consumption. It aims at connecting billions of IoT devices. This protocol's frequency range is 900MHZ, covering a range of 3km to 50km. The maximum data rate is very low, which is 1KBPS.

Long Range, Low Data Rate, High Power Consumption

Cellular

This is also known as a mobile network. Cellular networks are 2G, 3G, 4G, and 5G. It Has frequency ranges – 900MHz, 1.8/1.9/2.1 GHz. The range is approximately 35km and goes up to 200km. The average data rate is 35KBPS – 170KBPS. Cellular networks consume high power. This protocol is not used for most IoT devices due to frequency and security issues. It can be used with IoT applications like connected cars.

IOT Communication models:

Client-Server Model

In the Client-Server communication model, the client sends encoded requests to the server for information as needed. This model is stateless, meaning that each request is handled independently and data is not retained between requests. The server categorizes the request, retrieves the data from the database or resource representation, and converts it to an encoded response that is sent back to the client. The client then receives the response.

On the other hand, in the Request-Response communication model, the client sends a request to the server and the server responds to the request by deciding how to retrieve the data or resources needed to prepare the response. Once prepared, the server sends the response back to the client.

Publish-Subscribe Model

The Publish-Subscribe communication model consists of three entities: Publishers, Brokers, and Consumers.

Publishers are responsible for generating and sending data to specific topics managed by the broker. Publishers are not aware of the consumers subscribed to the topic.

Consumers subscribe to the topics managed by the broker to receive data from the publishers. The broker is responsible for sending the data to the appropriate consumers based on their subscription to specific topics.

The broker is responsible for accepting data from the publishers and forwarding it to the appropriate consumers subscribed to that specific topic. The broker is the only entity that has information regarding the consumer to which a particular topic belongs, and publishers are not aware of this information.

Push-Pull Model

The Push-Pull communication model consists of three entities: data publishers, data consumers, and data queues. Publishers and consumers are not aware of each other. Publishers push messages or data into the queue, and consumers on the other end pull data out of the queue. The queue acts as a buffer for messages when there is a difference in the rate of data push or pull by the publisher and consumer.

Queues play an essential role in decoupling messaging between the producer and consumer, and they act as a buffer in situations where there is a mismatch in the rate at which data is pushed by producers and pulled by consumers. This buffer helps ensure smooth communication between the two entities.

Exclusive Pair Model

Exclusive Pairs are communication models that provide full-duplex, bidirectional communication between a client and server. These models are designed for constant or continuous connections between the two entities.

Once a connection is established, both the client and server can exchange messages with each other. As long as the client does not request to close the connection, it remains open, and the server is aware of every open connection. This enables the client and server to communicate seamlessly and in real-time.

Future of IoT Communication Models

The future of IoT communication models is exciting and promising. As the number of connected devices and applications continue to increase, the need for efficient and effective communication models will become even more critical.

One of the most significant trends in IoT communication models is the shift towards edge computing. This approach involves processing data closer to the source, rather than transmitting it to a centralized cloud server. By moving processing closer to the edge of the network, latency can be reduced, and real-time responses can be achieved. This approach also reduces the amount of data that needs to be transmitted, reducing bandwidth requirements and improving efficiency.

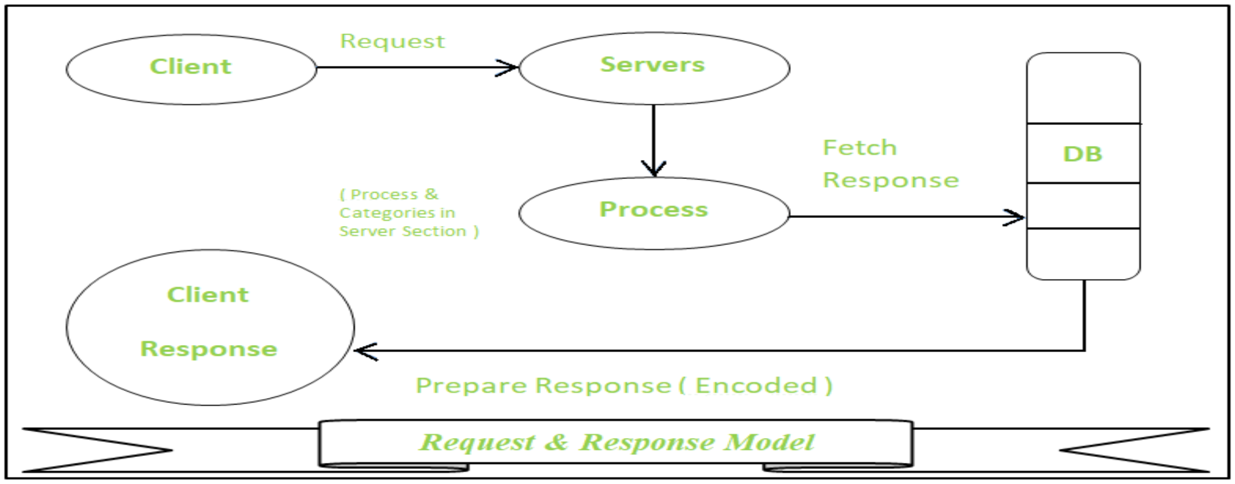
Another trend is the development of hybrid communication models that combine different communication protocols to achieve the best possible results. For example, a hybrid model might combine the Publish-Subscribe model with the Request-Response model to achieve real-time data updates while still allowing for targeted data requests.

IoT devices are found everywhere and will enable circulatory intelligence in the future. For operational perception, it is important and useful to understand how various IoT devices communicate with each other. Communication models used in IoT have great value. The IoTs allow people and things to be connected any time, any space, with anything and anyone, using any network and any service.

Types of Communication Model :

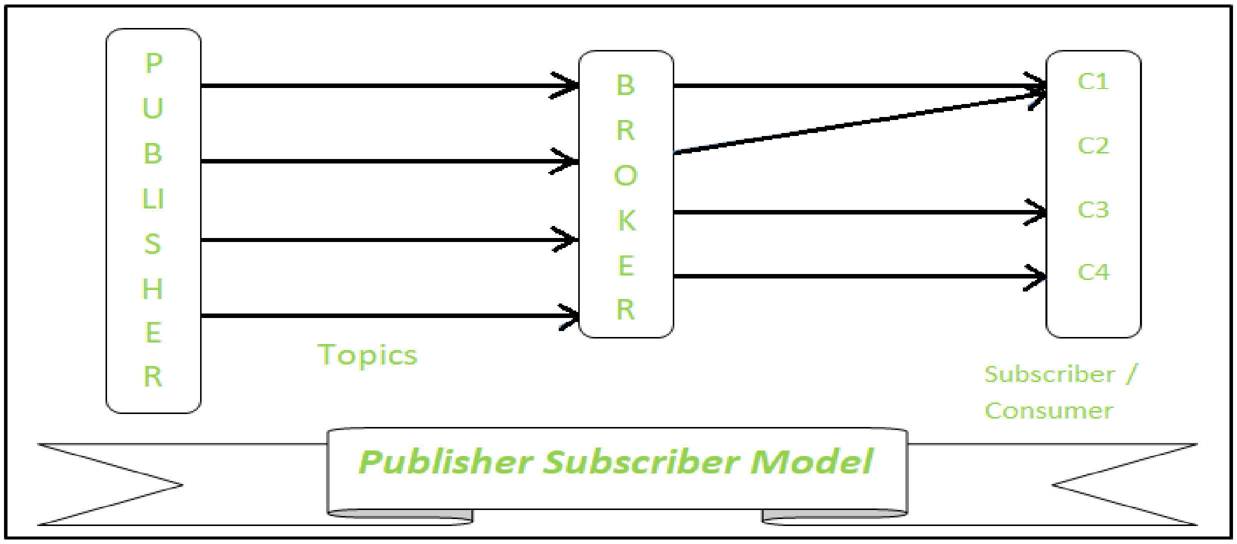
1.Request & Response Model –
This model follows a client-server architecture.

- The **client**, when required, requests the information from the server. This request is usually in the encoded format.
- This model is stateless since the data between the requests is not retained and each request is independently handled.
- The server Categories the request, and fetches the data from the database and its resource representation. This data is converted to response and is transferred in an encoded format to the client. The client, in turn, receives the response.
- On the other hand — In **Request-Response** communication model client sends a request to the server and the server responds to the request. When the server receives the request it decides how to respond, fetches the data retrieves resources, and prepares the response, and sends it to the client.



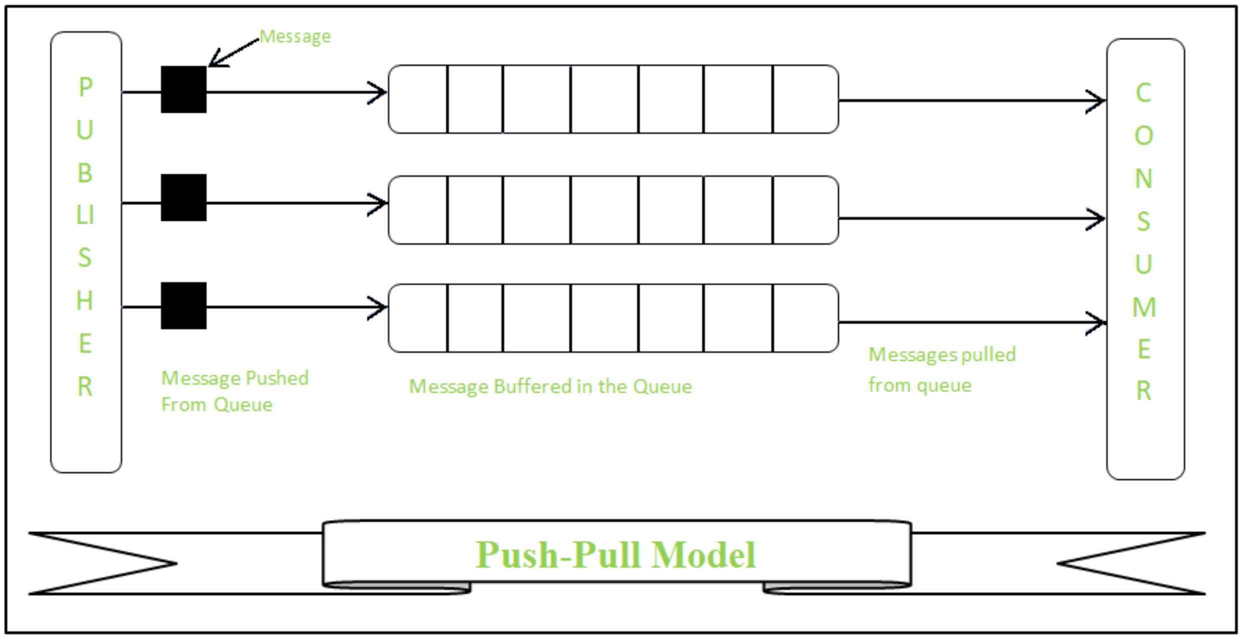
2. Publisher-Subscriber Model –
This model comprises three entities: Publishers, Brokers, and Consumers.

- **Publishers** are the source of data. It sends the data to the topic which are managed by the broker. They are not aware of consumers.
- **Consumers** subscribe to the topics which are managed by the broker.
- Hence, **Brokers** responsibility is to accept data from publishers and send it to the appropriate consumers. The broker only has the information regarding the consumer to which a particular topic belongs to which the publisher is unaware of.



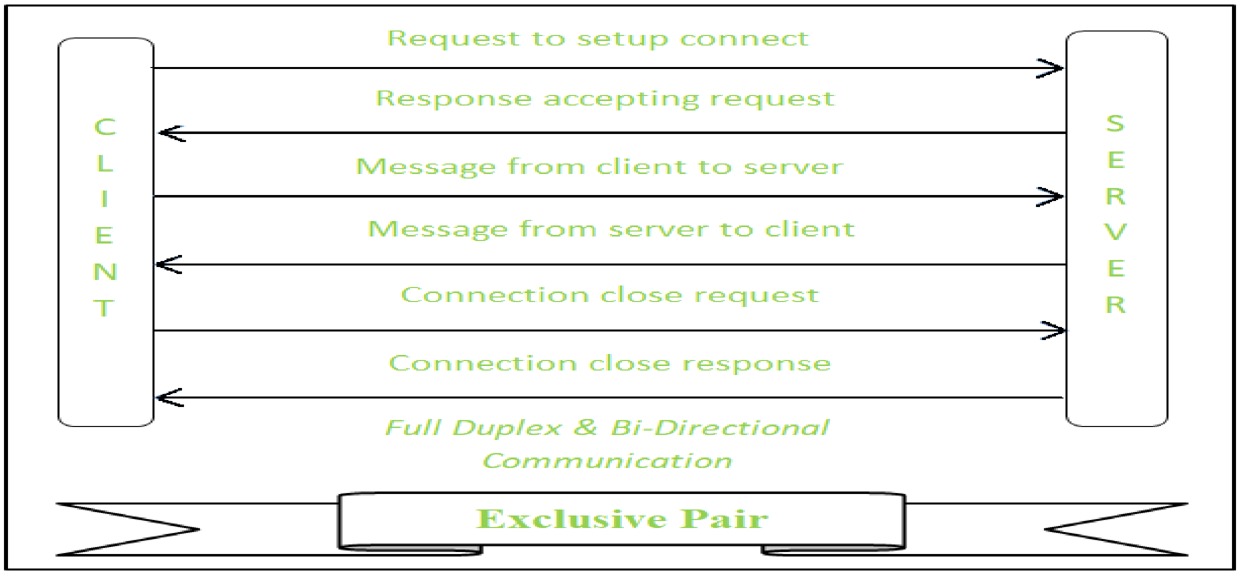
3. Push-Pull Model –
The push-pull model constitutes data publishers, data consumers, and data queues.

- **Publishers** and **Consumers** are not aware of each other.
- Publishers publish the message/data and push it into the queue. The consumers, present on the other side, pull the data out of the queue. Thus, the queue acts as the buffer for the message when the difference occurs in the rate of push or pull of data on the side of a publisher and consumer.
- **Queues** help in decoupling the messaging between the producer and consumer. Queues also act as a buffer which helps in situations where there is a mismatch between the rate at which the producers push the data and consumers pull the data.


PUBLISHER
Message
Message Pushed From Queue
Message Buffered in the Queue
Messages pulled from queue
CONSUMER
Push-Pull Model

4. Exclusive Pair –

- **Exclusive Pair** is the bi-directional model, including full-duplex communication among client and server. The connection is constant and remains open till the client sends a request to close the connection.
- The **Server** has the record of all the connections which has been opened.
- This is a state-full connection model and the server is aware of all open connections.
- WebSocket based communication API is fully based on this model.


CLIENT
SERVER
Request to setup connect
Response accepting request
Message from client to server
Message from server to client
Connection close request
Connection close response
Full Duplex & Bi-Directional Communication
Exclusive Pair

Conclusion

Finally, there is a growing focus on security and privacy in IoT communication models. As the number of connected devices continues to grow, the risk of security breaches and data theft also increases. Communication models that prioritize security and privacy will become increasingly important in the future to ensure the safe and secure exchange of data.

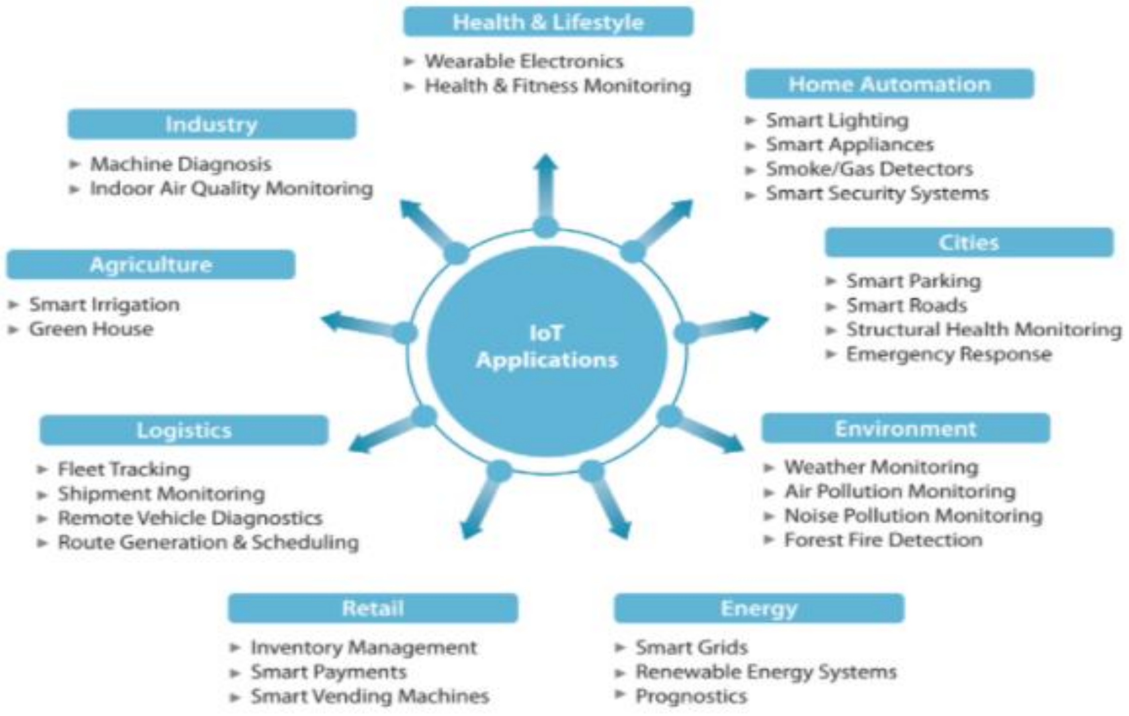
UNIT-5

UNIT V IOT SYSTEM DESIGN **9**

Basic building blocks of an IoT device – Raspberry Pi – Board – Linux on Raspberry Pi – Interfaces– Programming with Python – Case Studies: Home Automation, Smart Cities, Environment andAgriculture.

Basic building blocks of an IoT device

The Internet of Things denotes the connection of devices, machines, and sensors to the Internet. An IoT system comprises four basic building blocks: sensors, processors, gateways, and applications. This article will thoroughly discuss what each component of the IoT architecture represents.



The architecture of IoT components:

1. **Sensors** convert a non-electrical input to an electrical signal. Sensors are classified into two types: active and passive sensors. Whereas active sensors use and emit their own energy to collect real-time data (ex.: GPS, X-ray, radars), passive sensors use energy from external sources (ex: cameras). Additionally, sensors differentiate themselves by position, occupancy, and motion, velocity and acceleration, force, pressure, flow, humidity, light, radiation, temperature, etc.
2. **Processors** are the brain, the main part of the IoT system. They process the raw data captured by the sensors and extract valuable information. Examples of processors are microcontrollers and microcomputers.

3. **Gateways** are the combination of hardware and software used to connect one network to another. Gateways are responsible for bridging sensor nodes with the external Internet or World Wide Web. The figure below depicts how using gateways works.

4. **Applications** provide a user interface and effective utilization of the data collected.

The figure above illustrates some examples of IoT applications.

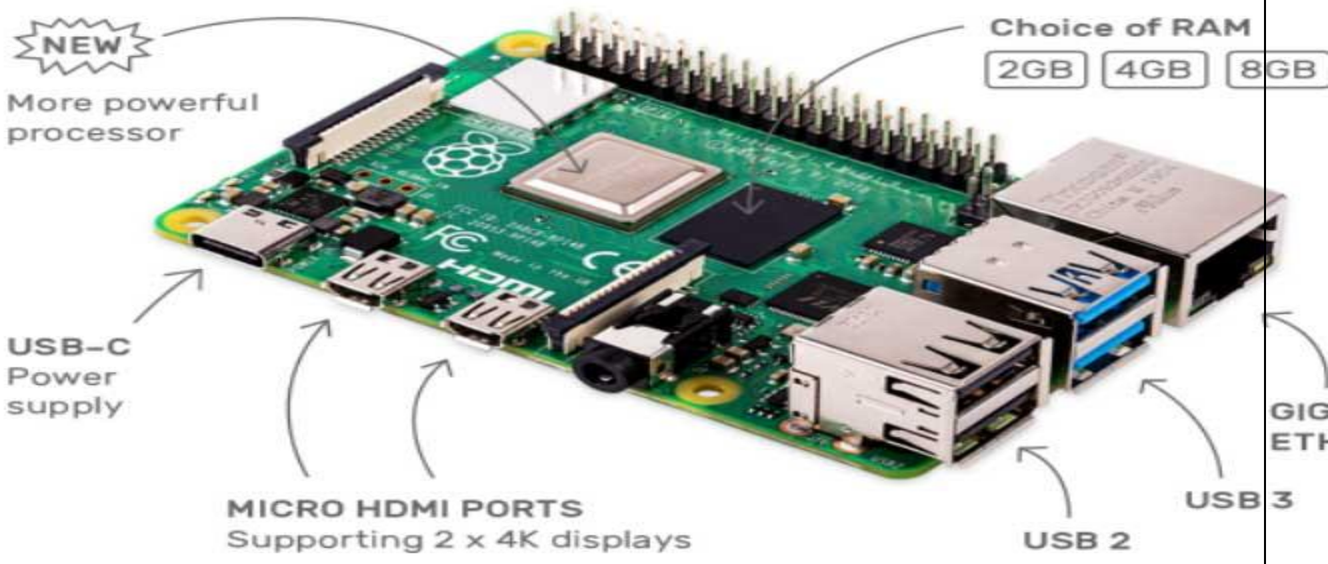
In summary, the IoT architecture comprises four basic building blocks: sensors, processors, gateways, and applications. Sensors are responsible for converting a non-electrical input to an electrical signal; processors "handle" the signals; gateways are used to connect a network to another, and, ultimately, an application offers a user interface and effective utilization of the data collected.

What Is A Raspberry Pi?

- The Raspberry Pi is a fully integrated computer (palmtop) mounted on a circuit board measuring approximately 7 cm x 5.5cm.
- It is a small, capable device that enables people of all ages to scan a computer and learn to edit in languages such as Scratch and Python. It can do everything you would expect a desktop computer to do, from browsing online and playing high-definition video, creating spreadsheets, word processing, and playing games.
- The Raspberry Pi has the ability to interact with the outside world and has been used in many digital maker projects, from music machines and parental finders to weather stations and tweeting birdhouses with infra-red cameras. We want to see the Raspberry Pi used by children all over the world to learn how to plan and understand how computers work.

History Of Raspberry Pi

- One-board Raspberry Pi computers have been developed in the United Kingdom by the Raspberry Pi Foundation to promote basic computer science teaching in schools and developing countries.
- The original model became more popular than expected, selling out of its target market for use as robots. Includes peripherals (such as keyboards and mice) or cases.UK Relief Society registered in the UK (No. 1129409), May 2009.
- Supported by the University of Cambridge Computer Laboratory and technology firm Broadcomm.
- Raspberry Pi Hardware has been upgraded with several versions that include memory capacity variations and peripheral compatible device support.



Important Components Of Hardware

- The Raspberry Pi has a Broadcom BCM2835system on chip (SoC), which includes the ARM1176JZF-S 700 MHz processor, which was later upgraded to Broadcom BCM2711, Quad-core Cortex-A72 (ARM v8) 64-bit SoC 1.5GHz.
- Originally shipped with 256 megabytes of RAM, later upgraded to 4GB.
- Does not include a built-in hard disk, but uses an SD card for boot and long-term storage.
- OS Support: Linux-based (Fedora, Raspbian, Debian, ArchLinux ARM, etc ..).

Planning Languages

- By default, it supports Python as a language of instruction.
- Any integrated ARMv6 language can be used with Raspberry Pi.
- Automatically installed in Raspberry Pi:
    - C or C ++ or Java or Ruby or Scratch

Application

- Can be used to make high-end computers.
- Raspberry Pi Medical Device Shield.
- Solar Raspberry Pi Power Pack.
- Voice Crafted Coffee Machine.
- Raspberry Pi Dynamic Bike Headlight Prototype.
- IoT Based Smart Application.

Raspberry Pi Interfaces

Raspberry Pi is most popular SBC(Single Board Computer). We can used Raspberry Pi as an IoT device and IoT Gateway. In this article we discuss Raspberry Pi Interfaces. Interfaces used for connecting Sensors and actuators.

What is Raspberry pi ?
The Raspberry Pi is a low cost, **credit-card sized computer** that plugs into a computer monitor or TV, and uses a standard keyboard and mouse. It is a capable little device that enables people of all ages to explore computing, and to learn how to program in languages like Scratch and Python. It's capable of doing everything you'd expect a desktop computer to do, from browsing the internet and playing high-definition video, to making spreadsheets, word-processing, and playing games."
If you know about Raspberry Pi more, Visit this : ***Raspberry Pi Tutorials***
If you have a Raspberry Pi and you want to setup for use in Headless mode, Visit This : ***Raspberry Pi Headless Mode Setup***
Raspberry pi has ***Serial, SPI and I2C*** interfaces for data transfer.
Serial : The Serial interface on Raspberry Pi has receive (Rx) and transmit (Tx) pins for communication with serial peripherals.
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 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

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)** an **SCL (Clock Line)**.

WHAT IS A PYTHON PROGRAM?

Python is a very useful programming language that has an easy to read syntax, and allows programmers to use fewer lines of code than would be possible in languages such as assembly, C, or Java.

The Python programming language actually started as a scripting language for Linux. Python programs are similar to shell scripts in that the files contain a series of commands that the computer executes from top to bottom.

Compare a "hello world" program written in C to the same program written in Python:

Unlike C programs, Python programs don't need to be compiled before running them. However, you will need to install the Python interpreter on your computer to run them. The Python interpreter is a program that reads Python files and executes the code.

It is possible to run Python programs without the Python interpreter installed though. Programs like Py2exe or Pyinstaller will package your Python code into stand-alone executable programs.

WHAT CAN A PYTHON PROGRAM DO?

Like shell scripts, Python can automate tasks like batch renaming and moving large amounts of files. It can be used just like a command line with IDLE, Python's REPL (read, eval, print, loop) function. However, there are more useful things you can do with Python. For example, you can use Python to program things like:

- Web applications
- Desktop applications and utilities
- Special GUIs
- Small databases
- 2D games

Python also has a large collection of libraries, which speeds up the development process. There are libraries for everything you can think of – game programming, rendering graphics, GUI interfaces, web frameworks, and scientific computing.
Many (but not all) of the things you can do in C can be done in Python. Python is generally slower at computations than C, but its ease of use makes Python an ideal language for prototyping programs and designing applications that aren't computationally intensive.

HOW TO WRITE AND RUN A PROGRAM IN PYTHON

We'll only cover the basics of writing and executing a Python program here, but a great tutorial covering everything a programmer needs to know about Python is the book Learning Python 5th Ed. (O'Reilly) by Mark Lutz.

INSTALLING AND UPDATING PYTHON

Python 2 and Python 3 come pre-installed on Raspbian operating systems, but to install Python on another Linux OS or to update it, simply run one of these commands at the command prompt:

```
sudo apt-get install python3
```

Installs or updates Python 3.

```
sudo apt-get install python
```

Installs or updates Python 2.

OPENING THE PYTHON REPL

To access the Python REPL (where you can enter Python commands just like the command line) enter `python` or `python3` depending on which version you want to use:

Enter Ctrl-D to exit the REPL.

WRITING A PYTHON PROGRAM

To demonstrate creating and executing a Python program, we'll make a simple "hello world" program. To begin, open the Nano text editor and create a new file named hello-world.py by entering this at the command prompt:

```
sudo nano hello-world.py
```

Enter this code into Nano, then press Ctrl-X and Y to exit and save the file:

```
#!/usr/bin/python

print "Hello, World!";
```

All Python program files will need to be saved with a ".py" extension. You can write the program in any text editor such as Notepad or Notepad++, just be sure to save the file with a ".py" extension.

RUNNING A PYTHON PROGRAM

To run the program without making it executable, navigate to the location where you saved your file, and enter this at the command prompt:

```
python hello-world.py
```

MAKE A PYTHON FILE EXECUTABLE

Making a Python program executable allows you to run the program without entering `python` before the file name. You can make a file executable by entering this at the command prompt:

```
chmod +x file-name.py
```

Now to run the program, all you need to enter is:

```
./file-name.py
```

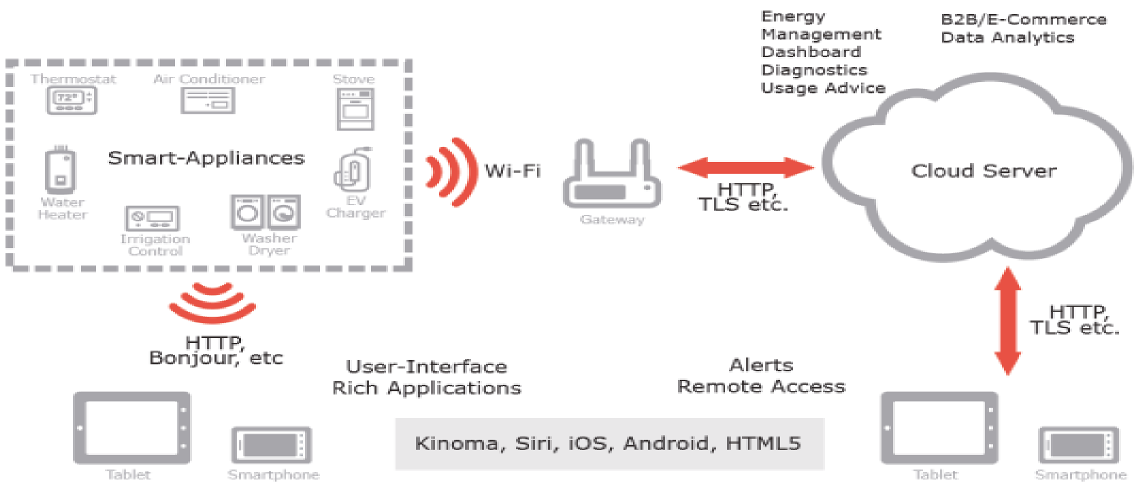
Here are some additional resources that will help you make the most out of programming in Python:

- Complete list of Python syntax
- The Python Package Index (PyPi)
- Installing Python packages on the Raspberry Pi

Case studies:

HOME AUTOMATION

IoT home automation is the ability to control domestic appliances by electronically controlled, internet-connected systems. It may include setting complex heating and lighting systems in advance and setting alarms and home security controls, all connected by a central hub and remote-controlled by a mobile app.



The rise of Wi-Fi’s role in home automation has primarily come about due tothe networked nature of deployed electronics where electronic devices (TVsand AV receivers, mobile devices, etc.) have started becoming part of thehome IP network and due the increasing rate of adoption of mobile computingdevices (smartphones, tablets, etc.), see above Figure.

The networking aspectsare bringing online streaming services or network playback, while becoming amean to control of the device functionality over the network. At the same timemobile devices ensure that consumers have access to a portable ‘controller’ forthe electronics connected to the network. Both types of devices can be used asgateways for IoT applications.

In this context many companies are consideringbuilding platforms that integrate the building automation with entertainment,healthcare monitoring, energy monitoring and wireless sensor monitoring inthe home and building environments.

IoT applications using sensors to collect information about the operating conditions combined with cloud hosted analytics software that analyzes disparatedata points will help facility managers become far more proactive about managingbuildings at peak efficiency

THREE LEVELS OF HOME AUTOMATION

1. **Monitoring:**

Description: Monitoring is the foundational level of home automation. It involves the ability to observe and track the status and conditions of various devices and systems within the home.

Characteristics:

Sensors and Devices: Implementation of sensors to monitor environmental conditions, security, energy usage, and more.

Alerts and Notifications: Receive notifications or alerts based on monitored events. For example, receive a notification if a door is left open or if there's a sudden change in temperature.

Remote Viewing: Access real-time data and status updates remotely through a mobile app or web interface.

2. **Control:**

Description: The control level builds upon monitoring by allowing users to actively manage and manipulate connected devices and systems within the home.

Characteristics:

Remote Control: Enable users to remotely control devices and systems. This includes turning lights on/off, adjusting thermostat settings, or locking/unlocking doors.

Manual Input: Users have direct control through interfaces such as mobile apps, voice commands, or dedicated control panels.

Customization: Users can set preferences and customize the behavior of devices based on their needs. For instance, adjusting the brightness and color of smart lights.

3. Automation:

Description: Automation represents the highest level of sophistication in home automation. It involves creating predefined scenarios or rules that trigger automated actions based on certain conditions or events.

Characteristics:

Scenes and Routines: Users can define scenes or routines that involve multiple devices. For example, a "Good Morning" routine might turn on lights, adjust the thermostat, and start the coffee maker.

Event Triggers: Automation rules can be triggered by specific events, such as motion detection, a door opening, or a scheduled time.

Adaptive Behavior: Intelligent automation that learns from user behavior and adjusts settings over time without explicit user input. For example, learning when to adjust the thermostat based on occupancy patterns.

SMART HOME AUTOMATION

All smart home devices can communicate using openHAB.11. HAB drives from **Home Automation Bus**. The developer deploys Java and OSGi services and uses the open source development environment and deployment platform. Its accompanying cloud platform is **my.openHAB** provides communication between that with the cloud. **The my.openHAB cloud-connector also provides REST and cloud-based services.**

Figure 12.7 shows architectural layers in openHAB development environment. A service in figure refers to service capabilities, which can be called upon whenever needed. The figure shows the following:

1. Core openHAB objects—REST service and repository; base library

2. openHAB add-on objects—Item provider, protocol bindings, automation logics, user interfaces and libraries

3. OSGi framework services—Configuration administration, event administration service, declarative services, log back, runtime and HTTP services

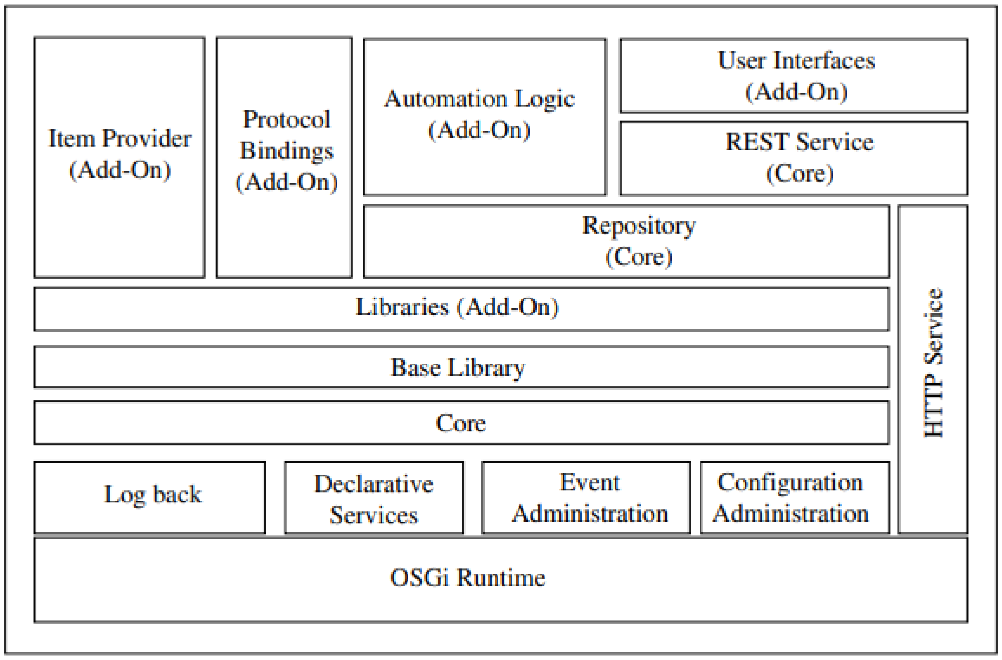


Figure 12.7 Architectural layers in openHAB development environment

4. **OpenHAB** deploys event administration service of OSGi with pub/sub mode.
5. A stateful repository is for querying and for use by **automation logics**. Some functions are stateless and do not depend on previous action(s). Remaining actions are stateful and dependent on previous chain of actions. State of items in repository is as per the actions.

Two domains and their high-level service capabilities in the home automation system in IoT architecture reference model are:

1. Device and Gateway Domain: Assume that the system deploys j lighting devices, each with a proximity sensor. Automation logic provides that if no change is found in proximity due to presence of person(s) then the devices switch off. Assume that the system also deploys k intrusion sensors and l appliances. Automation logic provides on intrusion, communicate trigger(s) to a local or remote web-service as per configuration setting at the configuration administration service of OSGi framework.

2. Application and Network Domain: Applications and network domain deploys applications and services and have high-level capabilities. Domain Architectural Reference Model.

Figure 12.8 shows the data-flow diagram and domain architecture reference model for home automation lighting, appliances and intrusion monitoring services.

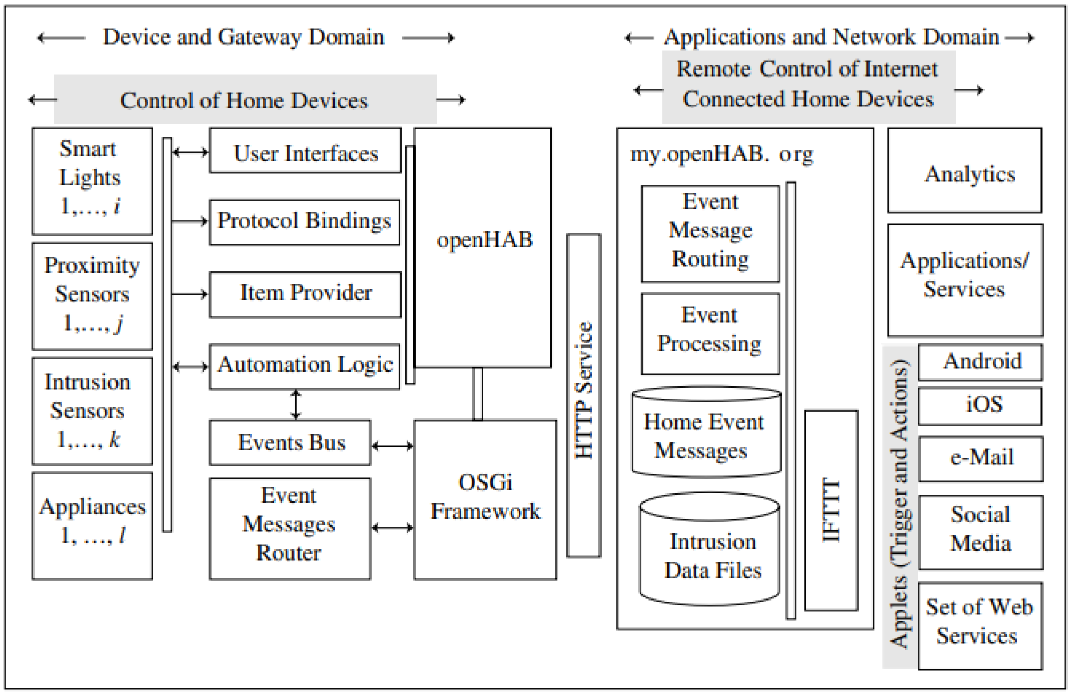


Figure 12.8 Data-flow diagram and domain architecture reference model for home automation lighting, appliances and intrusion monitoring services

Figure shows that openHAB has an event bus. The bus is asynchronous. The event bus refers to a communication bus for all protocol bindings. The bindings link to the hardware. The event bus is the base service of openHAB.

Example of the event is command, which triggers an action or a state change of some item or device. Another example of event is **status update** which informs about a status change of some item or device.

For example, in response to a command. The openHAB service is **integration-hub between such devices and bindings between different protocols** used for networking the home devices, OSGi and HTTP service.

Usually just one instance of openHAB runs on a central coordinator (computer) at home. Event Administration Service of OSGi service is used for remote services. Several distributed openHAB instances can connect and deploy the event bus.

SMART CITY

- Smart city applications and services connect people, process, data and things.
- A smart city can be defined as a vision which integrates multiple ICT and IoT solutions in a secure fashion to manage a city's assets such as information systems, schools, libraries, transportation systems, hospitals, power plants, water supply networks, waste management, law enforcement and other community services.
- Sectors that have been developing smart city technology include government services, transport and traffic management, energy, health care, water, innovative urban agriculture and waste management.

Smart-city solutions integrate a number of city services and can include the following:

1. Smart parking spaces
2. Smart street lightings and smart lighting solutions
3. Smart traffic solutions
4. Smart water management,
5. Smart connected bike share services
6. Smart health services
7. Smart structures
8. Smart city system integrator

Four-layer architectural framework developed at CISCO cloud IoT for a city.
Layers consists of:

(i) devices network and distributed nodes,
(ii) distributed data capture, processing and analyzing,
(iii) data centers and cloud and
(iv) applications, such as waste containers monitoring.

Figure shows data-flow diagram and domain architecture reference model for the smart city applications and services.

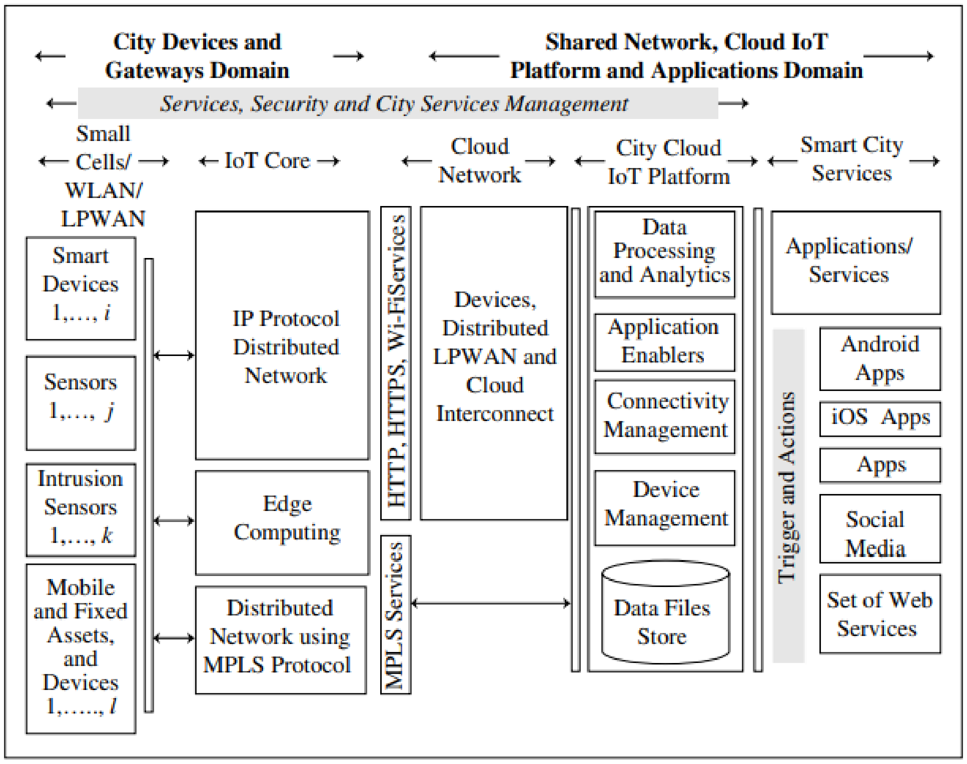

City Devices and Gateways Domain
Shared Network, Cloud IoT Platform and Applications Domain
Services, Security and City Services Management
Small Cells/ WLAN/ LPWAN
IoT Core
Cloud Network
City Cloud IoT Platform
Smart City Services
Smart Devices 1,..., i
Sensors 1,..., j
Intrusion Sensors 1,..., k
Mobile and Fixed Assets, and Devices 1,....., l
IP Protocol Distributed Network
Edge Computing
Distributed Network using MPLS Protocol
HTTP, HTTPS, Wi-FiServices
MPLS Services
Devices, Distributed LPWAN and Cloud Interconnect
Data Processing and Analytics
Application Enablers
Connectivity Management
Device Management
Data Files Store
Trigger and Actions
Applications/ Services
Android Apps
iOS Apps
Apps
Social Media
Set of Web Services

Figure 12.9 Data flow diagram and domain architecture reference model for the smart City Applications and Services

Two domains are

(i) City devices and Gateways domain,

(ii) shared network, cloud IoT platform and applications domains.

- Services, security and city services management are cross-domain functions.
- Assume the edge sensors and devices consist of, say i-smart devices, j-sensors, k-intrusion sensors and l- mobile and fixed assets and devices whereas i, j, k and l can be very large numbers.
- The edge sensors and devices wirelessly connect within small cells; systems connect with WLAN (Wireless LAN). They communicate using LPWAN.
- The distributed network of edge-computing systems connects using IP protocols or using the Multiprotocol Label Switching (MPLS).
- The MPLS assigns the labels to data packets and forward the labels to city cloud IoT platform. City cloud IoT platform collects messages, triggers, alerts and data files at data store.
- The platform does device and connectivity management functions, application enabler functions, and data processing and analytics.

- The platform generates triggers which follows actions, such as connect to social media, set of web services, applications, iOS apps and Android apps. The platform connects to a number of city applications and services.
- Smart city applications and services can deploy CISCO IoT, IOx and Fog. This is because of usages of shared networks and distributed access point nodes, and the need of an ecosystem with ability to transform sensor data and perform the control functions within the distributed network nodes.
- This enables development of applications, such as site asset management, energy monitoring, and smart parking infrastructure and connected cities.

SMART CITY PARKING

A growing problem in cities is of vehicular traffic congestion and parking spaces. A modern city, therefore, provides a number of multilevel parking spaces which spread all over the city.

A driver needs a mobile app. Significant fuel saving can result from provisioning of smart parking spaces in a city. **A smart parking-service should enable the following:**

1. Guides the drivers for the available parking slots and spaces

2. Provides a mobile app, and the app assists a driver and enables him/her to obtain the appropriate parking-slot information remotely.

3. Publishes messages in real time for available slots and alerts for slot unavailability at the parking utility

4. Consists of a central supervisory control and monitoring system (CSS) which connects the edge sensors and devices, accurately senses the slots available for occupancy of vehicles in real time, and predicts the expected availability time in case of non-availability of slots

5. Optimizes the usages of parking spaces and reaching time

6. Provides display boards at road traffic junctions for status of availability

7. Provides good parking experience to users

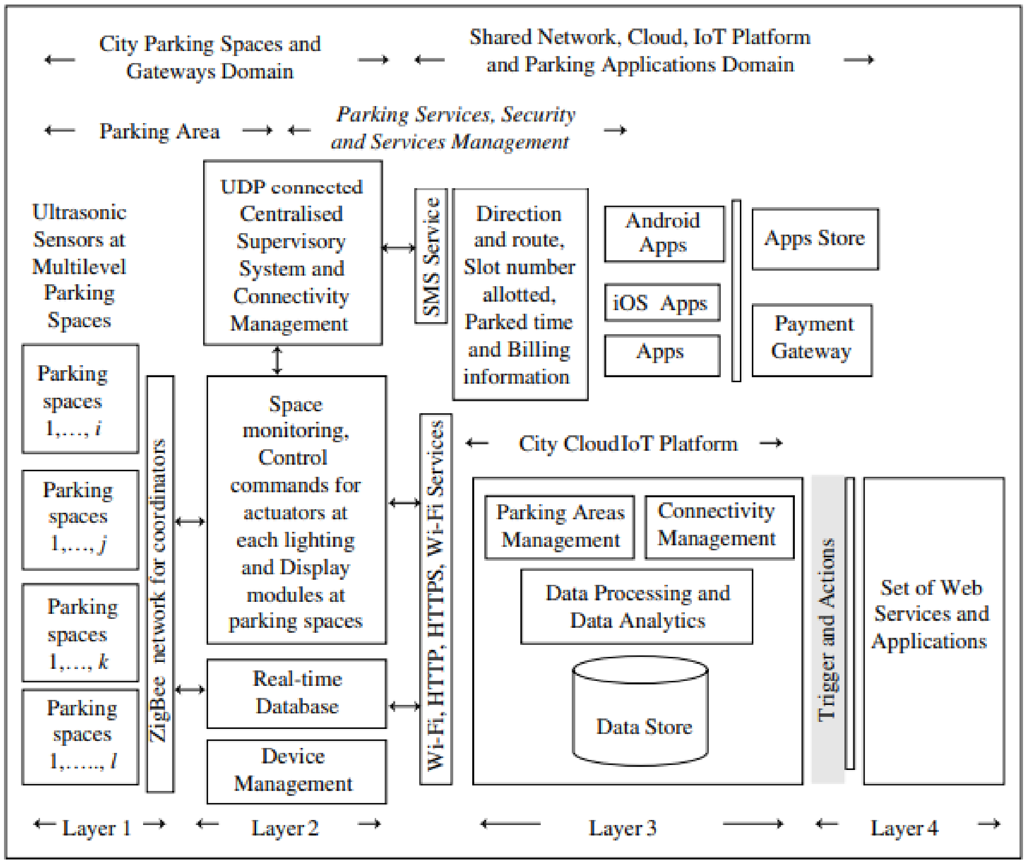
8. Adds value for all parking stakeholders, drivers and service providers

Sensors play vital role in the smart parking. The application is ranked as topmost among sensor-applications for a smarter world.

Figure 12.10 shows data-flow diagram, domains and architecture reference model for smart parking applications and services.

The figure shows four layers at two domains.

- **Parking spaces are at layer 1**. They are sensed using coordinators at each level in multilevel parking spaces. An actuator for the light at each slot is used. Lighting control module at the coordinator actuates the parking space lights. The lightings can be switched on and off as per requirement for each space.
- **A Parking Assistance System (PAS) is at layer 2.** This includes CCS and three modules for monitoring, control and display.
- Layers 1 and 2 communication protocols are ZigBee, LWM2M and UDP. The CSS maintains a real-time database of time-series data of the parking spaces.
- **The system connects layer 3, which includes the SMS gateway and cloud IoT platform.**



City Parking Spaces and Gateways Domain
Shared Network, Cloud, IoT Platform and Parking Applications Domain
Parking Area
Parking Services, Security and Services Management
Ultrasonic Sensors at Multilevel Parking Spaces
UDP connected Centralised Supervisory System and Connectivity Management
SMS Service
Direction and route, Slot number allotted, Parked time and Billing information
Android Apps
iOS Apps
Apps
Apps Store
Payment Gateway
Parking spaces 1,..., i
Parking spaces 1,..., j
Parking spaces 1,..., k
Parking spaces 1,....., l
ZigBee network for coordinators
Space monitoring, Control commands for actuators at each lighting and Display modules at parking spaces
Real-time Database
Device Management
Wi-Fi, HTTP, HTTPS, Wi-Fi Services
City CloudIoT Platform
Parking Areas Management
Connectivity Management
Data Processing and Data Analytics
Data Store
Trigger and Actions
Set of Web Services and Applications
Layer 1
Layer 2
Layer 3
Layer 4

Figure 12.10 Data flow diagram, domain architecture reference model and four layers for the design of smart parking applications and services

Layer 2 connects CSS with all coordinators. Layer 2 includes a real-time time-series database.

Layer 2 also connects with three modules for (i) displaying, (ii) space monitoring and (iii) control commands for actuators for each parking slot.

Layer 3 consists of SMS gateway and City cloud IoT platform that connects CSS, modules and database using the Wi-Fi, HTTP and HTTPS services.

The platform has data store, data processing and analytics, and parking areas and connectivity management modules. The CSS sends the UDP packets using MPLS and uses a SMS service to communicate with the mobile app.

The SMS service communicates parking information. A packet provides information such as slot available, slot allocated, time parked, billing information and directional and parking space route details to the user's mobile phone.

The user downloads a PAS app from the App store. The user's mobile also connects to a payment gateway for parking service bill payment.

Layer 4 web services connect the cloud data store, and use the PaaS cloud for the analytics.

Hardware Prototype Development and Deployment described sensors for ultrasonic pulse detectors.

When a car parking slot is occupied, then the parked car reflects back ultrasonic pulses to the source. The sensor measures the reflected directional intensity and delay period for the reflections. The coordinator updates the parked slots status on each alert from a circuit. The sensor associated circuit at coordinator alerts the CSS for status change and generates time-series messages from the sensor data and communicates to the CSS for saving at the real-time database.

Figure 12.11 shows the design principle for a set up for identifying vacant spaces and slot IDs using ultrasonic pulses and back reflections to the transceiver (emitter and sensors) at the coordinator.

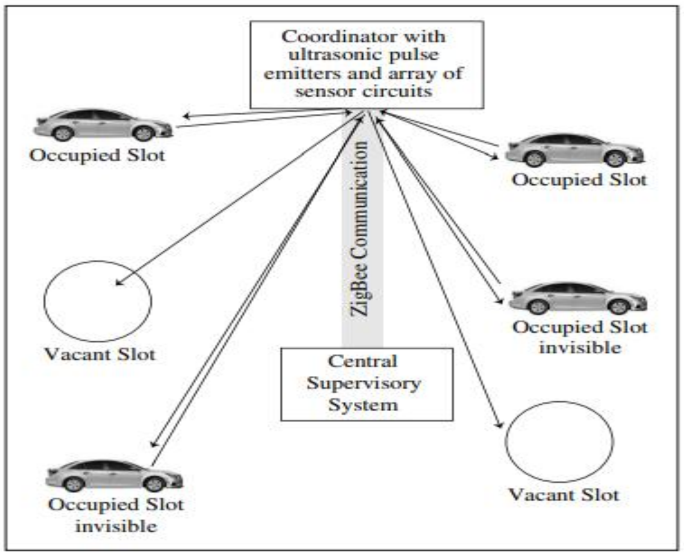


Figure 12.11 Design principle for the set up for identifying vacant spaces using ultrasonic pulses and back reflections from cars to the transceiver at the coordinator

SMART ENVIRONMENT

Smart Environment-Monitoring Environment monitoring refers to actions that are required for characterizing and monitoring the quality of the environment.

A smart environment monitoring system should enable the following:

1. Preparations for assessment of environment impact

2. Establish the trends in environmental parameters and current status of the environment

3. Interpretation of data and evaluate environmental quality indices

4. Monitor the air, soil and water quality parameters

5. Monitor harmful chemicals, biological, microbiological, radiological and other parameters

Weather Monitoring System

A smart weather monitoring system should enable the following:

1.Each **measuring node** for weather parameters is assigned an ID. Each lamppost deploys a wireless sensor node. Each node measures the T, RH and other weather parameters at assigned locations. A group of WSNs communicates using ZigBee and forms a network. Each network has an access point, which receives the messages from each node. They depicted interconnections between nodes, coordinators, routers and access points. Each access point associates a gateway.

2. The **nodes communic**ate the parameters up to the access point using WSNs at multiple locations.

3. Forward and store the parameters on an Internet cloud platform

4. Publishes weather messages for the display boards at specific locations in the city and communicates to weather API at mobile and web users

5. Publishes the messages in real time and send alerts using a weather reporting application 6. Analyze and assess the environment impact .

7.Enables intelligent decisions using data and historical analytics reports at city cloud weather data store

Two domains and their high-level service capabilities in the weather monitoring services in IoT architecture reference model are:

1. Device and Gateway Domain: Assume that the system deploys m weather-sensor embedded devices, each with a location-data sensor and n access-points for the WSNs. A sensor node does minimum required computations, gathers sensed information and communicates with other connected nodes in the network

A data adaptation layer for the data, messages, triggers and alerts does the main computations and puts the result in real time updated database. The items identified for communication from gateway are queried from the database.

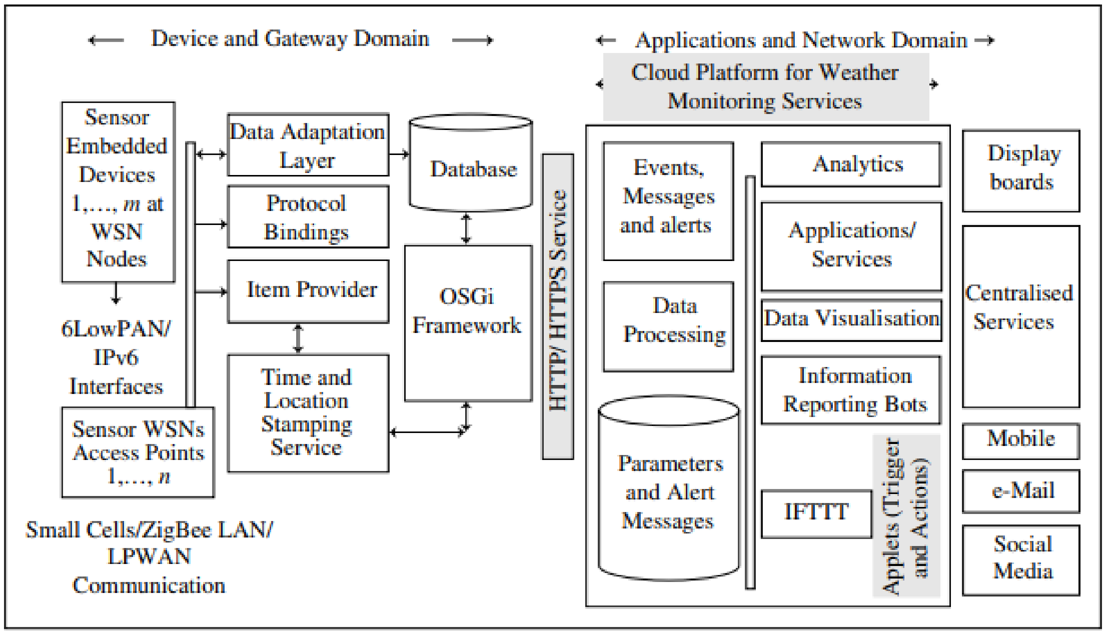
The items communicate from gateway using network protocols and HTTP/HTTPS services**.**

(i) **Device subdomain:** Hardware WSN board consists of sensors for weather parameters. A board example is Waspmote.

(ii) **Gateway Subdomain:** The parameters and alerts communicate to a local or remote web service, time-and location-stamping service, item provider, protocol bindings and 6LowPAN/IPv6 modules as per configuration setting at the configurationadministration service of OSGi framework. The bindings between

ZigBee LANs, 6LowPAN and LPWAN and IPv6 protocols are used for networking of the devices, WSNs, OSGi with the HTTP/HTTPS services.

2. Application and Network Domain: Applications and network domain deploys the applications and services and has high-level capabilities, such analytics, data visualization, display-board feeds, weather reporting application



Device and Gateway Domain
Applications and Network Domain
Cloud Platform for Weather Monitoring Services
Sensor Embedded Devices 1,…, m at WSN Nodes
Data Adaptation Layer
Protocol Bindings
Item Provider
Time and Location Stamping Service
Database
OSGi Framework
HTTP/ HTTPS Service
6LowPAN/ IPv6 Interfaces
Sensor WSNs Access Points 1,…, n
Small Cells/ZigBee LAN/ LPWAN Communication
Events, Messages and alerts
Data Processing
Parameters and Alert Messages
Analytics
Applications/ Services
Data Visualisation
Information Reporting Bots
IFTTT
Applets (Trigger and Actions)
Display boards
Centralised Services
Mobile
e-Mail
Social Media

Figure 12.12 Data-flow diagram and domain architecture reference model for the WSNs based monitoring services

Devices Hardware Design and Code Development Environment, Development, Debugging and Deployment

Devices Hardware Design and Code Development Environment, Development, Debugging and Deployment

A microcontroller circuit consists of memory, over the air programmability (OTP) and transceiver associated with each sensor or node. The weather monitoring circuit deploys sensors for T, RH and atmospheric pressure (Patm) and may include solar visible radiation, wind speed and direction, and rainfall.Hardware design of the sensor and WSN node can use Arduino board with ZigBee shield.

Weather Reporting Bot

A bot is an application that runs automated or semi-automated scripts a for specific set of tasks and communicates the results over the Internet. A bot generally performs the task which are simple and structurally repetitive, such as a, weather reporting bot. The word 'bot' is derided from the word robot. A bot can communicate with an API using Instant Messaging (IM) or Internet Relay Chat (IRC) or to Twitter or Facebook.

A bot can also chat and give responses to the questions from user API. The bot uses the weather parameters and generates the alert messages from the database and messages for forecast by a cloud analytics service

A mobile app can display the report in two succeeding frames repeatedly.

The first frame shows the weather condition of the current day, as:

1. First line: condition such as clear, rain, partly rain, cloudy or partly cloudy

2. Second line, first part text gives the day current T and four or five spaces

3. Second line, second part text gives superscripted text for the maximum T expected and subscripted text for minimum T expected, followed by four or five spaces

4. Second line, third part text gives superscripted text for current RH% value and subscripted text for wind-speed in kmph (kilometer per hour).

Thus, first reporting frame displays the current condition and day's forecast for Tmax and Tmin. Second frame shows the forecast for today, tomorrow and day after for weather as:
1. First line: "Sat Sun Mon"

2. Second line shows a symbol which is completely unfilled circle for sunny, or cloud image with sun for partly cloudy, or cloud sign for fully cloudy below each day Sat, Sun and Mon.

After the sign, a superscripted word gives maximum T, and a subscripted word, the minimum expected on that day. Thus, forecast for three days is reported, viz. today, tomorrow and day after. Example of creating a weather bot is Slack weatherbot API.

The bot uses the codes given a Farnciskim site.

The API is a node.js module for the bot. It displays Second frame shows the forecast for today, tomorrow and day after days for weather, for example as follows:

1. First line: Bot name Current Runtime (such as 09:15 A.M.)

2. Second line shows “Condition for City Name, Place Name, Current Time, Standard (such as IST, GMT)

3. Third line shows “Today (such as SAT): T, condition (such as sunny, partly cloudy, cloudy or rain)”

4. Fourth line shows “Tomorrow (such as SUN) Current: T, condition”

5. Fifth line shows “Day after (such as MON) Current: T, condition”

Air Pollution Monitoring

A growing problem for all residents is air pollution from cars, toxic gases generated in factories and farms, such as carbon monoxide (CO). Pollution needs monitoring and to ensure the safety of workers and goods inside chemical plants.

The monitoring does the following tasks:

1. Monitoring and measuring levels of CO, a gas dangerous above 50–100 ppm level; carbon dioxide (CO_2), a gas causes which greenhouse effect; and ozone (O_3), a gas dangerous above 0.1 mg/per kg air level, for controlling air pollution

2. Monitoring and measuring levels of hydrogen sulfide (H_2S), a highly toxic gas. It is a greenhouse gas so its increase may contribute to global warming as well.

3. Monitoring and measuring levels of hydrocarbons, such as ethanol, propane.

4. Measure T, RH and Patm parameters for calibrations of sensed gaseous parameters of each node

5. Investigate air quality and the effects of air pollution.

6. Compute Air Quality Index (AQI) from the parameters, such as hourly or daily averages of air pollutant concentration, particulate matter (such as dust or carbon particle)

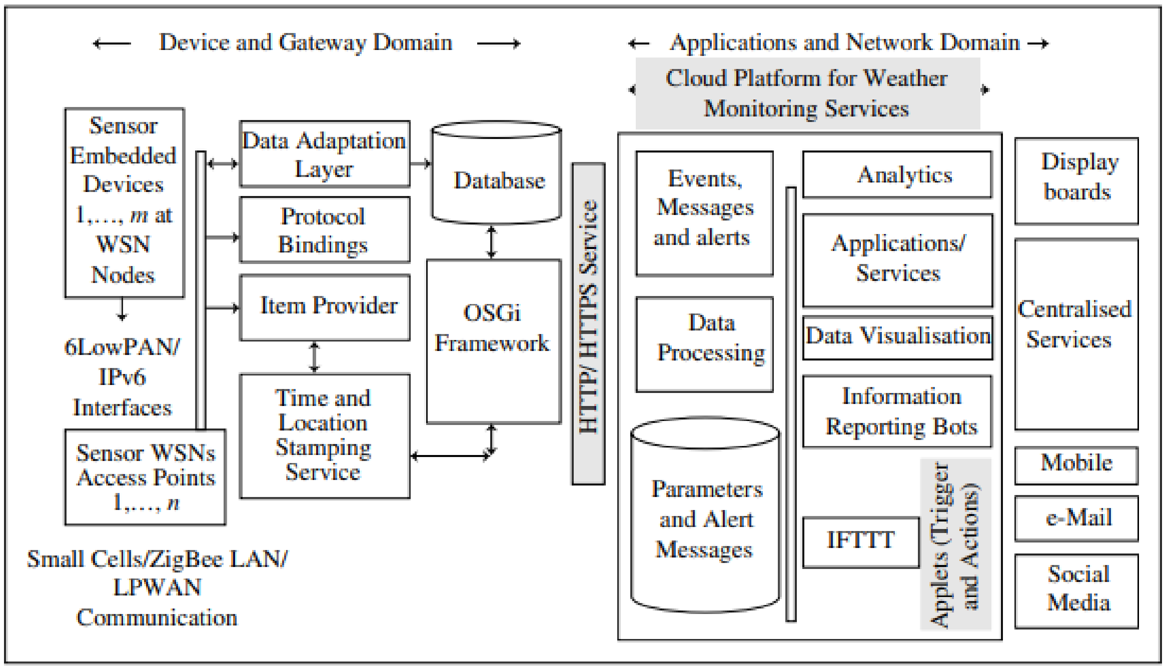
7. Compute source and spatial dispersion of pollutants as a function of day conditions, wind-speed and direction, air temperature and air temperature gradient with altitude and topography using analytics.

8. Data visualization

9. Report the pollution status to monitoring authorities

Sensors play a vital role in air-quality monitoring. The application has eleventh ranking among sensor-applications for a smarter world.

A data-flow diagram and domain architecture reference model for air pollution monitoring services are similar to Figure 12.12.



Device and Gateway Domain
Applications and Network Domain
Cloud Platform for Weather Monitoring Services
Sensor Embedded Devices 1,..., m at WSN Nodes
6LowPAN/ IPv6 Interfaces
Sensor WSNs Access Points 1,..., n
Small Cells/ZigBee LAN/ LPWAN Communication
Data Adaptation Layer
Protocol Bindings
Item Provider
Time and Location Stamping Service
Database
OSGi Framework
HTTP/ HTTPS Service
Events, Messages and alerts
Data Processing
Parameters and Alert Messages
Analytics
Applications/ Services
Data Visualisation
Information Reporting Bots
IFTTT
Applets (Trigger and Actions)
Display boards
Centralised Services
Mobile
e-Mail
Social Media

Figure 12.12 Data-flow diagram and domain architecture reference model for the WSNs based monitoring services

Two domains and their high-level service capabilities in the air quality and pollution monitoring services in IoT architecture reference model are:

1.Device and Gateway Domain: Assume that the system deploys m gas sensor embedded devices at each WSN with a location-data sensor and n access-points for the WSNs **(Figures 7.17).**

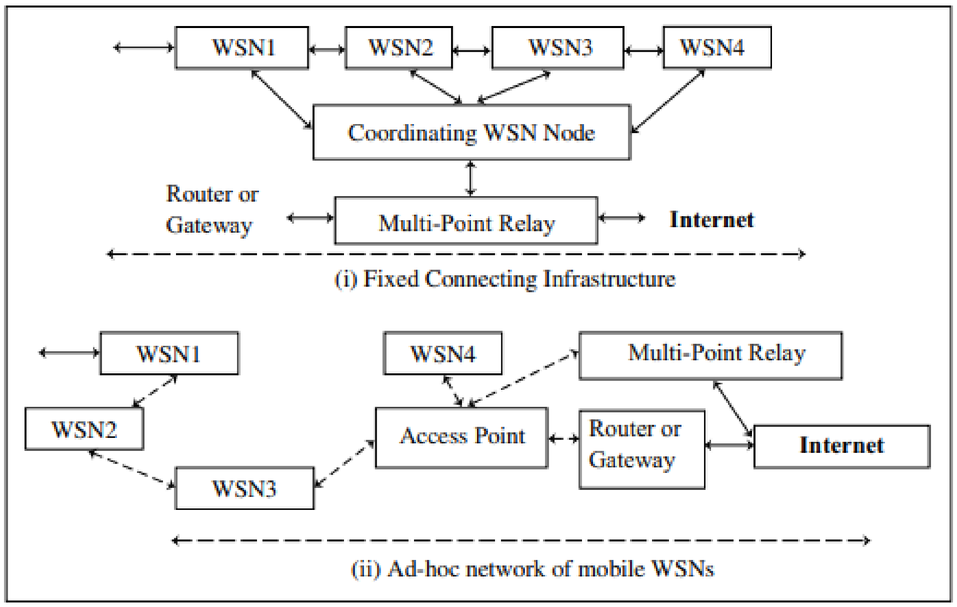


Figure 7.17 Architecture for connecting WSN nodes: (i) Fixed connecting infrastructure of WSN nodes, coordinators, relays, gateways and routers and (ii) Mobile ad-hoc network of moving WSNs

The data-adaptation layer at gateway does the aggregation, compaction and fusion computations for each sensor node data. The queries gather sensed information from the database and the items selected communicate using HTTP/HTTPS/MPLS services. WSN board IO ports connect the sensors for gaseous, particulate matter and weather parameters. Each sensor node is configured by assigning a node ID. A node ID maps with the GPS location found earlier from GPS modules at the data adaptation layer at the gateway.

A sensor ID is configured for each sensor at the node. Each sensor associated circuit is also configured for frequency of measurements every day and interval between two successive measurements. The sensor circuit is configured to activate only for measurement duration at a measuring instance followed by long inactive intervals.

An example is Waspmote board which can be used with sensors such as city pollution CO, NO, NO2, O3, SO2 and dust particles sensors and air-quality finding sensors for SO2, NO2, dust particles, CO, CO2, O3 and NH3. The Arduino or Eclipse IDE can be used to develop codes for the Waspmote.

2.The Applications and Network Domain: The applications and network domain deploys the applications and services and have high-level capabilities, such as events, messages, alerts and data processing, databases, applications and services, analytics, data visualization, display-board feeds, pollution reporting applications and services, and IFTTT triggers and actions. The cloud platform can be TCUP, AWS IoT, IBM Blue mix or Nimbits.

Forest Fire Detection

A big problem for countries with large forest areas is forest fires. A fire monitoring service does the following tasks:

1. Uses OTP features for programmable WSNs and gateways
2. Measures and monitors the T, RH, CO, CO2 and infrared light (fire generated) intensity in real time at preset intervals
3. Each WSN uploads the program and preset measured intervals of t1 (say, 300 s) each and the preset measured intervals of t2 (say, on 1 or 5 s) on sensed parameters values exceeding thresholds can instantaneously trigger the fire-alarm algorithm
4. Configures the data-adaptation layers with calibration parameters
5. Communicates the WSN messages at the preset intervals to the access point associated for specific network area
6. Communicates alerts, triggers, messages and data at data-adaptation layer using an uploaded program at associated gateway
7. Uploads connectivity programs for gateways
8. Runs at the data-adaptation layer the faulty or inaccessible sensors at periodic intervals
9. Integrates data with the node locations found from mapping with node IDs, compute, and activate the alarms using an algorithm, input-sensed and calibrated coefficients
10. Processes the layer data and database information, and communicates instantaneously to nearest mobiles and fire-fighting services near the access point gateway
11. Updates the database and communicates to a cloud platform, such as Nimbits, my.openHAB, TCUP, AWS or Blue mix platform

12. Modifies the preset measured intervals to t2 on activation of the fire alarm after value changes above the configured threshold values
13. Uses analytics to evaluate reliability index of the preset, threshold and configuration values and need to update alarm-algorithm and if needs improvement then upload new algorithms
14. Uses analytics to generate and communicate topological maps for the currently fireinfected forest area and reachability maps for fire-fighting service equipment's Sensors play a vital role in forest-fire monitoring.

The application has tenth ranking among 50 sensor-applications for a smarter world.

Figure 12.13 shows a data-flow diagram and domain architecture reference model for the monitoring service.

The figure shows that the service deploys m embedded-sensor devices at each of n WSN associated with x access points.

Device and gateway domain functions in the fire monitoring service for forests in IoT architecture reference model is as follows:

A lookup table enables mapping of two entities. Location-data stamping uses sensor IDs at a lookup table. Data adaptation of each sensor is at the layer. Data aggregates, compacts and fuses, computes, gathers sensed information and the algorithms use that for alarm and faulty sensor identification and configuration management. Data store at the database, updates in real time. The alerts and messages communicate to IoT cloud platform. Hardware WSN board and sensors can use Waspmote board.24 Each WSN communicates to access points using a multiprotocol wireless router

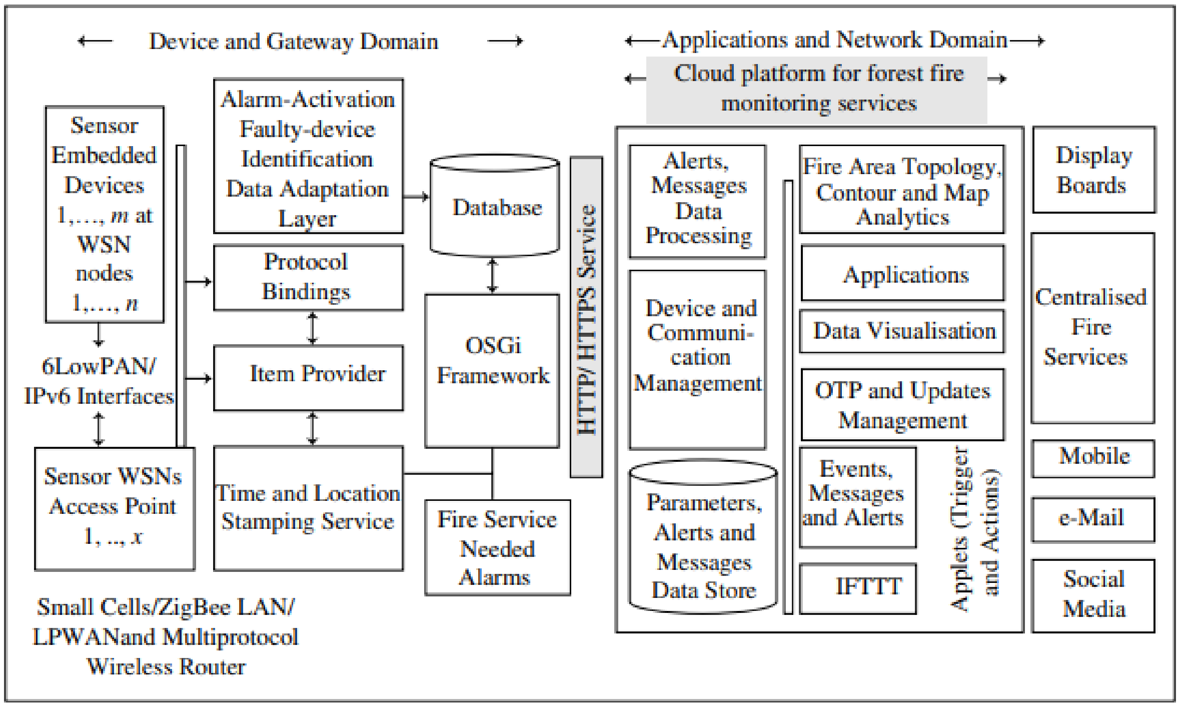

Device and Gateway Domain
Applications and Network Domain
Cloud platform for forest fire monitoring services
Sensor Embedded Devices 1,…, m at WSN nodes 1,…, n
Alarm-Activation Faulty-device Identification Data Adaptation Layer
Database
HTTP/ HTTPS Service
Alerts, Messages Data Processing
Fire Area Topology, Contour and Map Analytics
Display Boards
Protocol Bindings
Applications
6LowPAN/ IPv6 Interfaces
Item Provider
OSGi Framework
Device and Communi-cation Management
Data Visualisation
Centralised Fire Services
OTP and Updates Management
Sensor WSNs Access Point 1, …, x
Time and Location Stamping Service
Fire Service Needed Alarms
Parameters, Alerts and Messages Data Store
Events, Messages and Alerts
IFTTT
Applets (Trigger and Actions)
Mobile
e-Mail
Social Media
Small Cells/ZigBee LAN/ LPWANand Multiprotocol Wireless Router

Figure 12.13 Data flow diagram and domain architecture reference model for the WSNs based Forest Fire monitoring service

SMART AGRICULTURE

Following section describes two applications viz., smart irrigation in crop fields and smart wine quality enhancing

Smart Irrigation

Smart irrigation deploys sensors for moisture.

A smart irrigation monitoring service does the following tasks:

1. Sensors for moisture and actuators for watering channels are used in smart irrigation.

2. Uses soil moisture sensors with a sensor circuitry board with each one installed at certain depth in the fields.

3. Uses an array of actuators (solenoid valves) which are placed along the water channels and that control deficiencies in moisture levels above thresholds during a given crop period. 4. Uses sensors placed at three depths for monitoring of moisture in fruit plants such as grapes or mango, and monitors evapotranspiration (evaporation and transpiration)

5. Measures and monitors actual absorption and irrigation water needs

6. Each sensor board is in a waterproof cover and communicates to an access point using ZigBee protocol. An array of sensor circuits forms a WSN.

7.Access point receives the data and transfers it to an associated gateway. Data adapts at the gateway and then communicates to a cloud platform using LPWAN.

8. The cloud platform may be deployed such as Nimbits, my.openHAB, AWS or Bluemix. 9. Analytics at the platform analyses the moisture data and communicates to the actuators of water irrigation channels as per the water needs and past historical data

10. Measurements at the sensors are at preset intervals and actuators activate at analysed required values of the intervals.

11. The platform uploads the programs to sensors and actuators circuitry and sets preset measurement intervals of T1 (say, 24 hour) each and the preset actuation interval of t2 (say, on 120 hour)

12. Sensed moisture values when exceed preset thresholds then trigger the alarm

13. An algorithm uploads and updates the programs for the gateways and nodes.

14. Runs at the data-adaptation layer and finds the faulty or inaccessible moisture sensors at periodic intervals

15. Open source SDK and IDE are used for prototyping the monitoring system

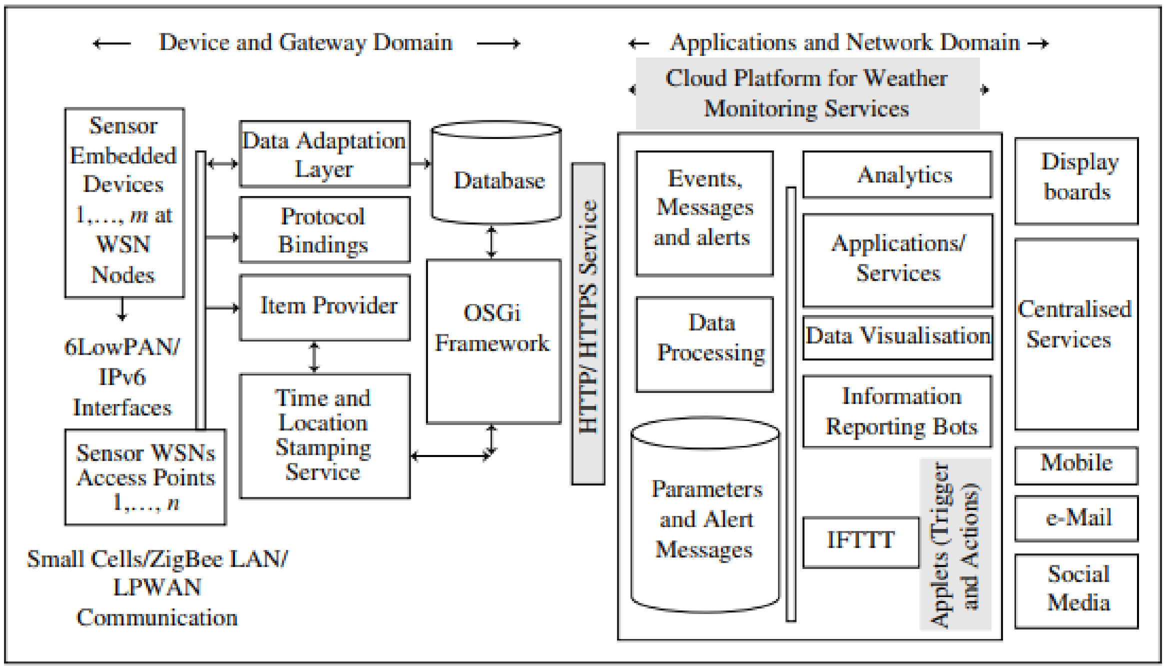

Device and Gateway Domain
Applications and Network Domain
Cloud Platform for Weather Monitoring Services
Sensor Embedded Devices 1,…, m at WSN Nodes
Data Adaptation Layer
Protocol Bindings
Item Provider
Time and Location Stamping Service
Database
OSGi Framework
HTTP/ HTTPS Service
6LowPAN/ IPv6 Interfaces
Sensor WSNs Access Points 1,…, n
Small Cells/ZigBee LAN/ LPWAN Communication
Events, Messages and alerts
Data Processing
Parameters and Alert Messages
Analytics
Applications/ Services
Data Visualisation
Information Reporting Bots
IFTTT
Applets (Trigger and Actions)
Display boards
Centralised Services
Mobile
e-Mail
Social Media

Figure 12.12 Data-flow diagram and domain architecture reference model for the WSNs based monitoring services

Smart Wine Quality Enhancing

The sensors monitor the soil moisture and trunk diameter in vineyards. The monitoring controls the sugar content in grapes and health of grapevines. Data-flow diagram and domain architecture reference-model for the monitoring service are similar to ones shown in **Figure 12.12.**

Device and Gateway Domain

A WSN measures moisture and other parameters and has an ID. Each node is a WSN. Each WSN measures at assigned places in a crop or vineyard at certain depth(s) inside the soil. Sensors at three equally spaced depths are used for the vineyard grapes sugar-control.

A group of WSNs communicate among themselves using ZigBee and form a network. Each network has an access point, which receives the messages from each node using LPWAN. **Figures 7.17 show the WSNs**. They show interconnections between nodes, coordinators, routers and access points. Each access point associates a gateway. Each gateway communicates to the cloud using LPWAN

ENERGY

Energy IoT applications for smart energy systems:

a). Smart Grid b). Renewable Energy Systems c). Prognostics

SMART GRIDS

Smart grid technology provides predictive information and recommendations to utilize, their suppliers, and their customers on how best to manage power.

Smart grid collects the data regarding:

- Electricity generation
- Electricity consumption
- Storage
- Distribution and equipment health data

• By analyzing the data on power generation, transmission and consumption of smart grids can improve efficiency throughout the electric system. Storage collection and analysis of smarts grids data in the cloud can help in dynamic optimization of system operations, maintenance, and planning.

• Cloud-based monitoring of smart grids data can improve energy usage levels viaenergy feedback to users coupled with real-time pricing information.

• Condition monitoring data collected from power generation and transmission systems can help in detecting faults and predicting outages.

RENEWABLE ENERGY SYSTEM

• Due to the variability in the output from renewable energy sources (such as solar and wind), integrating them into the grid can cause grid stability and reliability problems.

• IoT based systems integrated with the transformer at the point of interconnection measure the electrical variables and how much power is fed into the grid

• To ensure the grid stability, one solution is to simply cut off the overproductions.
• Communication systems for grid integration of renewable energy resources [IEEE Network, 2011] -provided the closed-loop controls for wind energy system that can be used to regulate the voltage at point of interconnection which coordinate wind turbine outputs and provides reactive power support.

PROGNOSTICS

• IoT based prognostic real-time health management systems can predict performance of machines of energy systems by analyzing the extent of deviation of a system from its normal operating profiles.

• In the system such as power grids, real time information is collected using specialized electrical sensors called Phasor Measurement Units (PMU)

• Analyzing massive amounts of maintenance data collected from sensors in energy systems and equipment can provide predictions for impending failures.

• OpenPDC is a set of applications for processing of streaming time-series data collected from Phasor Measurements Units (PMUs) in real-time.

ApplicationsofIoTinLogistics

Implementation ofIoT in logistics industry canboostthe constituentsof these pillarsand helpthe logistics industry augment by leap and bounds. Below are some of the advantages that aconventionallogistics companycanenjoyfrom theapplication ofInternetof Things.

1) **LocationandRouteManagement**:

Trucks are the lifeline of any logistics company. In the US alone, more than 70% of all the goodsare transported by trucks. In fact, around 95% of all the manufactured

goods at one point aretransported via trucks. Logistics and fleet companies hence require systems that can help themmanagetheirtruck operations.

The location and route management solution of IoT for logistics industry is hence quite popular.This solution enables a logistics manager to monitor the location of their trucks in real-time. Byusing GPS tracking systems and geofencing techniques, the route taken by the trucks can also bemonitored from remote locations. This further helps the logistics companies to track driveractivitiesand ensuretimelycargo delivery.

Moreover, the real-time alert system of these vehicle tracking solutions alarm managers aboutany anomaly like thunderstorms or accident on a freeway via push notifications that may affectthestatusof shipment.

Thesefeaturesactasanassistantforlogisticscompaniesandassistintheplanningandmanagement of delivery schedules. Time-delaying barriers are instantly identified and mitigatedthatresult instreamlined businessprocessesandcent percent customersatisfaction.

2) **InventoryTrackingandWarehousing**

IoT in logistics other than providing fleet management services also facilitates the

storage ofgoods and management of stock levels. In a logistics ecosystem, it enables a company to haveclear-cuttransparencyinitsvariousoperations,furthersupportinginseamlessinventoryorganization.

RFID tags and sensorsallow companies to easily keep track of their inventory itemsalong with their status and position. In other terms, IoT facilitates the development of a smartwarehouse system that allows a company to prevent losses, ensure safe storage of goods, andefficientlylocatetheitemsinneed.Furthermore,italsohelpscompaniestorevamptheir warehousingoperations,resultinginthereductionoflaborcostsandanincreaseinefficiency dueto less manual handlingerrors.

3) **CBMand BreakdownPrevention:**

IoTapplicationsinlogisticssegmentarenotonlylimitedtothemonitoringandmanag ementofassets.However,itsmostbeneficialapplicationistheidentificationofbottlenecks that may result in the breakdown of these assets. Internet of Things has helpedindustriestojumponpredictivemaintenanceandcondition-basedmaintenanceinsteadofdependingon scheduled inspection procedures.

Bymeasuringandanalyzingparametersthatdefinetheperformanceofthetrucks,companies can predict patterns related to common truck breakdown. Similarly, real-time alertsystems can be used to gain alerts about probable unexpected malfunctions that can be preventedviacondition-based maintenance.

These predictive applications of IoT will help companies to identify defects before theybecomecatastrophic.Logisticscompanieswillbeabletoimprovetheirdecision-makingprocessesandcreateeffectiveinspection/repairstrategies.Moreover,thesepreventiveinsightsabout their assets will help companies reduce risks and downtime that will further result inseamlessprocess execution and timelydeliveryoperations.

4) **IoTandBlock chainforDigitalBOL:**

Applications of IoT in logistics industry when blended with the technology of Block chain createa digital Bill of Lading (BOL) that creates whole new transparency in the supply chains. ThisBOL allows a company and its customers to trace the transportation cycle of the products beingshipped.

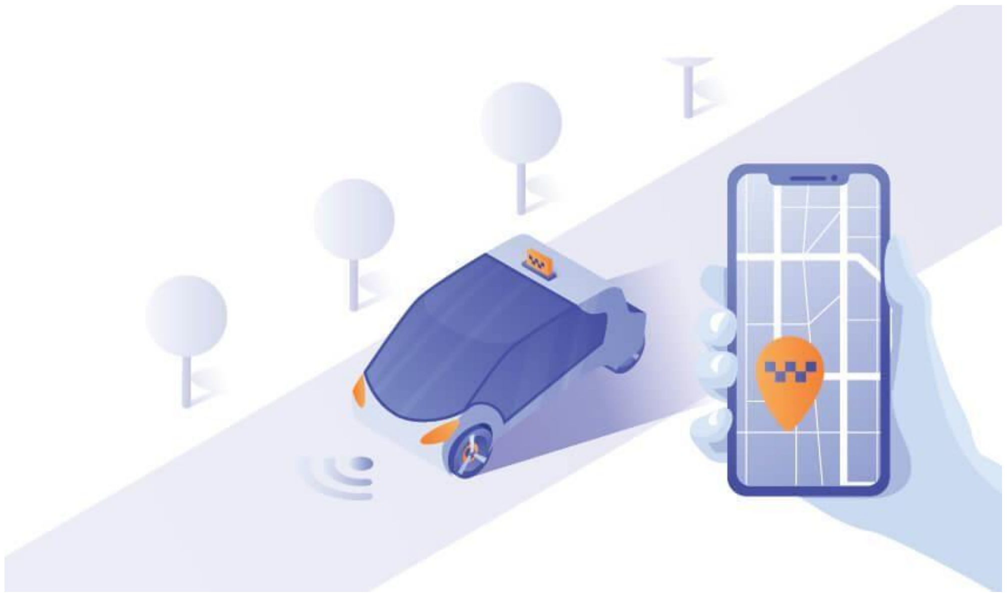
The amalgamation of both these technologieshas resultedin the creationof smart contractsolutions (BOL being one of its many constituents) that enable monitoring of all the stagesbetween the originof the goodsand their finaldelivery inthe customer's hands.Sensors andGPS trackers play a crucial role in this solution as well. Both parties can measure temperature,humidity, location, and other parameters from

remote locations in real-time during the shipmentandmakesurethat all the conditions of thecontract arefulfilled.

AsdataisstoredinaBlock chain,theprobabilityofdatatheftorcyber-attackisconsiderably reduced. Hence, transactions are instantly released from the customer's account ifall the pre-described conditions comply. The customer can also cancel the contract if the contractis breached due to reasons like spoiled cargoor delivery delay.This maintains a two-wayauthorityoverthecontractspecificationsfurtherenforcingthesecurity,transparency,an dtraceabilityof thesupplychain.

5) AutonomousandSelf-DrivingCars:

Logistics managers are not only responsible to manage the management of assets beingtransported. They are also supposed to ensure the safety of truckers and the cargo being shipped.Thiscan beaccomplished bythe implementationof self-



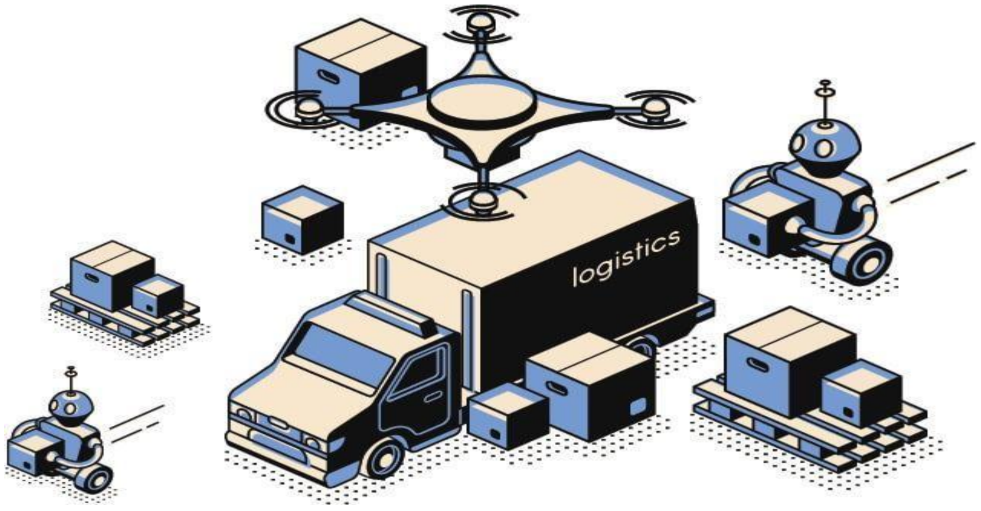
drivingvehicles.

The hypeofautonomous and self-driving vehicles is atanall-time high.Intelligenttechnologies like artificial intelligence and machine learning are being blended by connectedinfrastructures formed by the Internet of Things. Using such infrastructures of IoT in logisticswillbethefirststepforbusinessestoincludetheconceptofautonomousvehicles.Datacorrespondingtovariousshipmentparameterswillbeanalyzedandprocessedtodevelopsmartdrivingroutesanddirections.Logisticswillhencebeabletoreducetheiroperationcosts, minimizecar accidents, and ensuretimelycargo deliverybased on trafficconditions.

6) Drone-BasedDelivery:

Unmanned Aerial Vehicles (UAVs) or drones are the new medium to deliver packages. Theirpotential lies strongly in the field of retail, logistics, agriculture, and e-commerce. Amazon, oneof the Big 4 tech companies in the world has also unveiled the use of drones for deliveringordereditems to people located in remoteareas.

Drones applications and implementation of Internet of Things in logistics can ensure automatedprocess execution and quick delivery of goods. The market of drones based delivery systems isgrowing at a rapid rate and is expected to reach a market valuation



of

$11.2 billion by the end of2020.

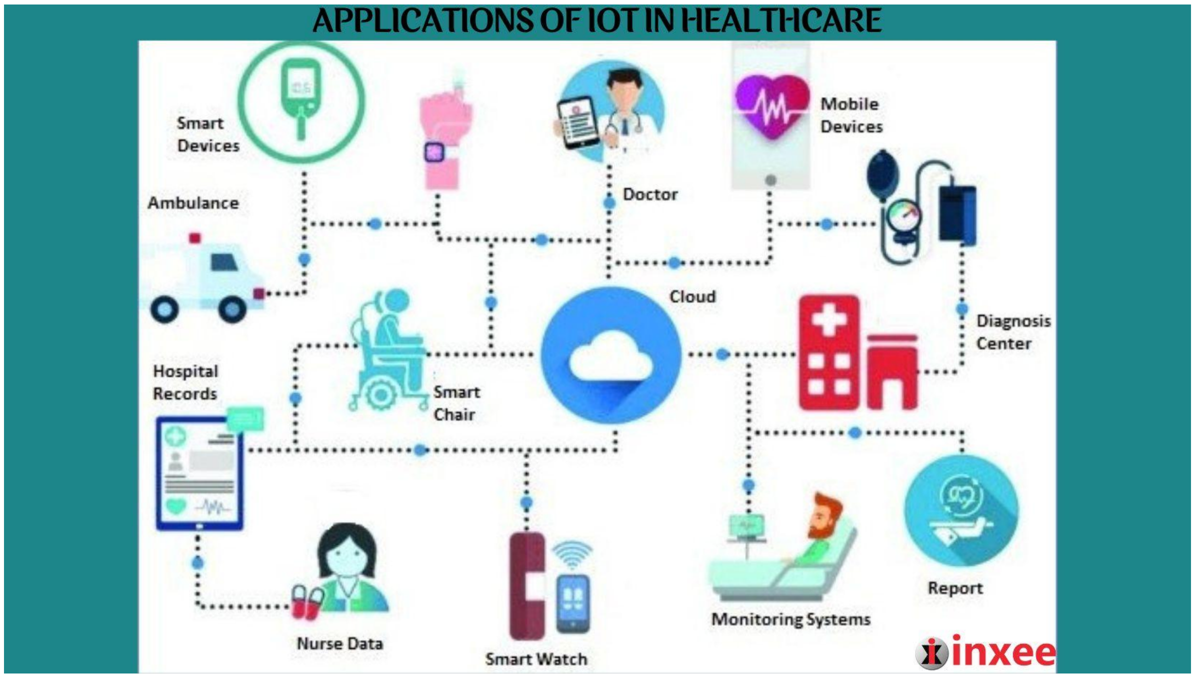
IoT in Health and life style

IoT devices offer a number of new opportunities for healthcare professionals to monitor patients, as well as for patients to monitor themselves. By extension, the varieties of wearable IoT devices provide an array of benefits and challenges, for healthcare providers and their patients alike.

Remote patient monitoring

Remote patient monitoring is the most common application of IoT devices for healthcare. IoT devices can automatically collect health metrics like heart rate, blood pressure, temperature, and more from patients who are not physically present in a healthcare facility, eliminating the need for patients to travel to the providers, or for patients to collect it themselves.

When an IoT device collects patient data, it forwards the data to a software application where healthcare professionals and/or patients can view it. Algorithms may be used to analyze the data in order to recommend treatments or generate alerts. For example, an IoT sensor that detects a patient's unusually low heart rate may generate an alert so that healthcare professionals can intervene.



A major challenge with remote patient monitoring devices is ensuring that the highly personal data that these IoT devices collect is secure and private.

Glucose monitoring

For the more than 30 million Americans with diabetes, glucose monitoring has traditionally been difficult. Not only is it inconvenient to have to check glucose levels and manually record results, but doing so reports a patient's glucose levels only at the exact time the test is provided. If levels fluctuate widely, periodic testing may not be sufficient to detect a problem.

IoT devices can help address these challenges by providing continuous, automatic monitoring ofglucose levels in patients. Glucose monitoring devices eliminate the need to keep records manually, and they can alert patients when glucose levels are problematic.

Challenges include designing an IoT device for glucose monitoring that:

- Is small enough to monitor continuously without causing a disruption to patients
- Does not consume so much electricity that it needs to be recharged frequently.
- These are not insurmountable challenges, however, and devices that address them promise to revolutionize the way patients handle glucose monitoring.

Heart-rate monitoring

Like glucose, monitoring heart rates can be challenging, even for patients who are present in healthcare facilities. Periodic heart rate checks don't guard against rapid fluctuations in heart rates, and conventional devices for continuous cardiac monitoring used in hospitals require patients to be attached to wired machines constantly, impairing their mobility.

Today, a variety of small IoT devices are available for heart rate monitoring, freeing patients to move around as they like while ensuring that their hearts are monitored continuously. Guaranteeing ultra-accurate results remains somewhat of a challenge, but most modern devices can deliver accuracy rates of about 90 percent or better.

Hand hygiene monitoring

Traditionally, there hasn't been a good way to ensure that providers and patients inside a healthcare facility washed their hands properly in order to minimize the risk of spreading contagion.

Today, many hospitals and other health care operations use IoT devices to remind people to sanitize their hands when they enter hospital rooms. The devices can even give instructions on how best to sanitize to mitigate a particular risk for a particular patient.

A major shortcoming is that these devices can only remind people to clean their hands; they can't do it for them. Still, research suggests that these devices can reduce infection rates by more than 60 percent in hospitals.

Depression and mood monitoring

Information about depression symptoms and patients' general mood is another type of data that has traditionally been difficult to collect continuously. Healthcare providers might periodically ask patients how they are feeling, but were unable to anticipate sudden mood swings. And, often, patients don't accurately report their feelings.

"Mood-aware" IoT devices can address these challenges. By collecting and analyzing data such as heart rate and blood pressure, devices can infer information about a patient's mental state. Advanced IoT devices for mood monitoring can even track data such as the movement of a patient's eyes.

The key challenge here is that metrics like these can't predict depression symptoms or other causes for concern with complete accuracy. But neither can a traditional in-person mental assessment.

Parkinson's disease monitoring

In order to treat Parkinson's patients most effectively, healthcare providers must be able to assess how the severity of their symptoms fluctuate through the day.

IoT sensors promise to make this task much easier by continuously collecting data about Parkinson's symptoms. At the same time, the devices give patients the freedom to go about their lives in their own homes, instead of having to spend extended periods in a hospital for observation.

Other examples of IoT/IoMT

While wearable devices like those described above remain the most commonly used type of IoT device in healthcare, there are devices that go beyond monitoring to actually providing treatment, or even "living" in or on the patient. Examples include the following.

Connected inhalers

Conditions such as asthma or COPD often involve attacks that come on suddenly, with little warning. IoT-connected inhalers can help patients by monitoring the frequency of attacks, as well as collecting data from the environment to help healthcare providers understand what triggered an attack.

In addition, connected inhalers can alert patients when they leave inhalers at home, placing them at risk of suffering an attack without their inhaler present, or when they use the inhaler improperly.

Ingestible sensors

Collecting data from inside the human body is typically a messy and highly disruptive affair. No no enjoys having a camera or probe stuck into their digestive tract, for example.

With ingestible sensors, it's possible to collect information from digestive and other systems in a much less invasive way. They provide insights into stomach PH levels, for instance, or help pinpoint the source of internal bleeding.

These devices must be small enough to be swallowed easily. They must also be able to dissolve or pass through the human body cleanly on their own. Several companies are hard at work on ingestible sensors that meet these criteria.

Connected contact lenses

Smart contact lenses provide another opportunity for collecting healthcare data in a passive, non- intrusive way. They could also, incidentally, include micro cameras that allow wearers effectively to take pictures with their eyes, which is probably why companies like Google have patented connected contact lenses.

Whether they're used to improve health outcomes or for other purposes, smart lenses promise to turn human eyes into a powerful tool for digital interactions.

Robotic surgery

By deploying small Internet-connected robots inside the human body, surgeons can perform complex procedures that would be difficult to manage using human hands. At the same time, robotic surgeries performed by small IoT devices can reduce the size of incisions required to perform surgery, leading to a less invasive process and faster healing.