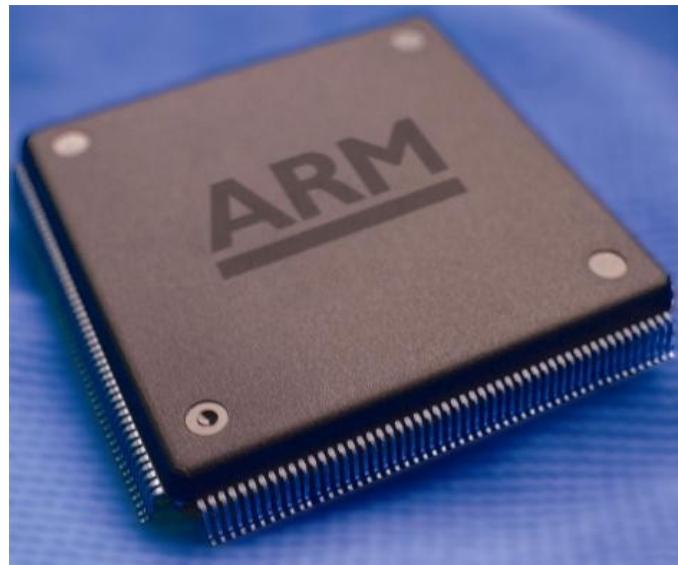


# Module – 1

## Microprocessor and Microcontroller

### (18CS44)



Edit with WPS Office

# Introduction to ARM

- “Acorn Computers Ltd. (Cambridge, England)  
Nov. 1990
- First called Acorn RISC Machine, then Advanced RISC Machine
- Based on RISC architecture work done at UCal Berkley and Stanford
- ARM cores are widely used in mobile phones, handheld organizers, and a multitude of other everyday portable consumer devices.
- Optimized for low power & performance



# ARM

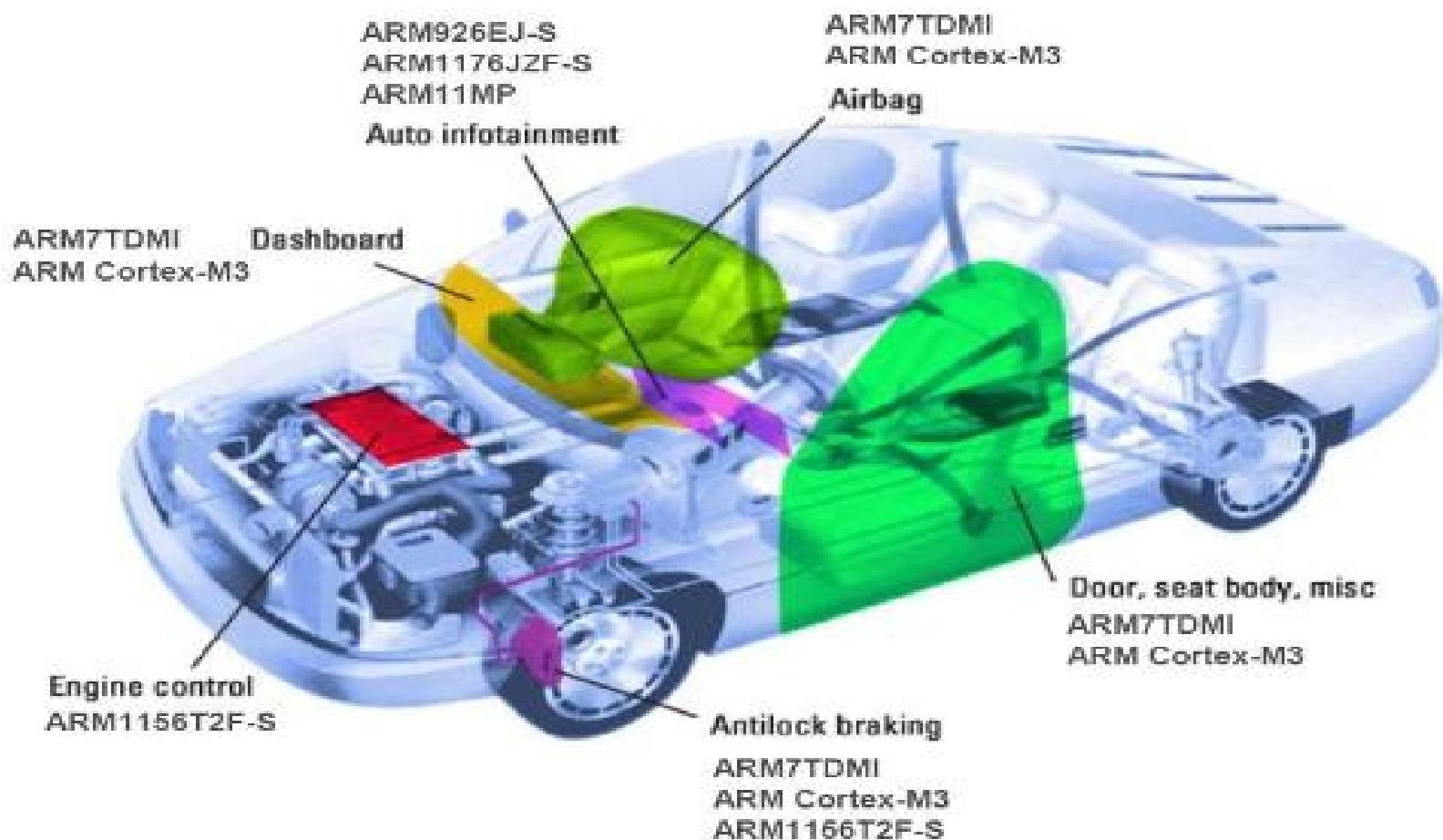
## Powered Products



Edit with WPS Office

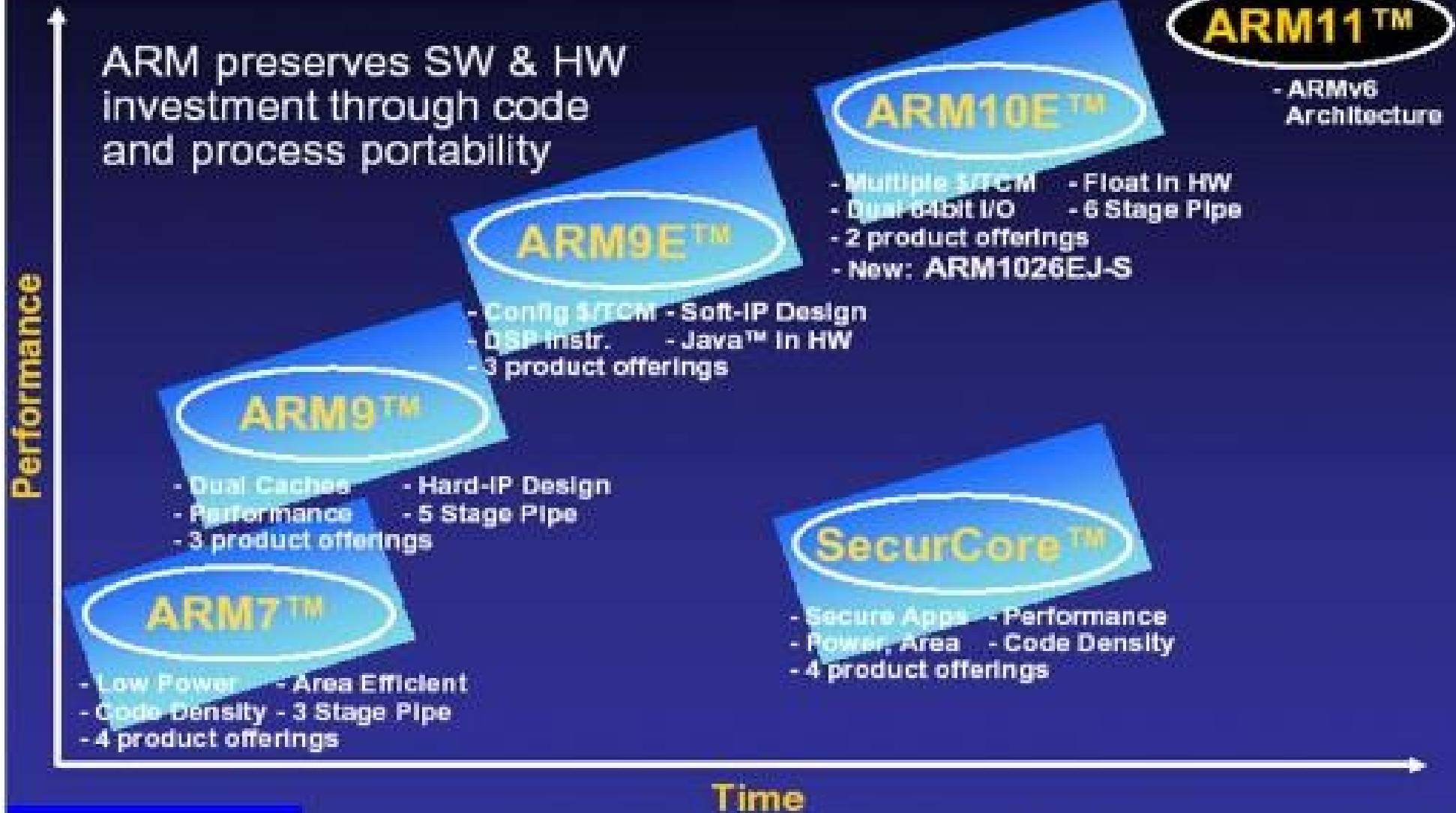
# ARM Product Example

ARM Products



Edit with WPS Office

# Five Families of ARM Processor IP



# Brief History of ARM

- 1985, First ARM (ARM1)
  - Core
- 1995, ARM7TDMI (Thumb instruction , Debugger, Multiplier , In circuit emulator )
  - Most successful ARM core
  - 3-stage pipeline, 120 Dhrystone MIPS (VAX 11/780)
- 1997, ARM9
  - 5-stage pipeline
  - Harvard (I+D cache), MMU (OS's VM)
- 1999, ARM10
  - 6-stage pipeline
  - VFP(Vector Float Point) (7-stage pipeline)
- 2003, ARM11
  - 8-stage pipeline



Edit with WPS Office

# NOMENCLATURE OF ARM FAMILY

## ARM-XYZ TDMI EJF- S

- X - series of ARM processor
  - Y - Support of CACHE MEMORY
  - Z – Support of Memory management Unit.
- 
- T – Thumb architecture Support of 16 bit instruction.
  - D – Debugger support
  - M – Fast Multiplier
  - I – Embedded ICE - In Circuit Emulator
- 
- E – Embedded Trace Macro-cell.
  - J – Jazelle Instruction set / Java byte code, support to java programs
  - F – Floating point co-processor
- 
- S – Synthesizable version means the ARM is a set of software instruction engine that can be compiled on a suitable compiler.



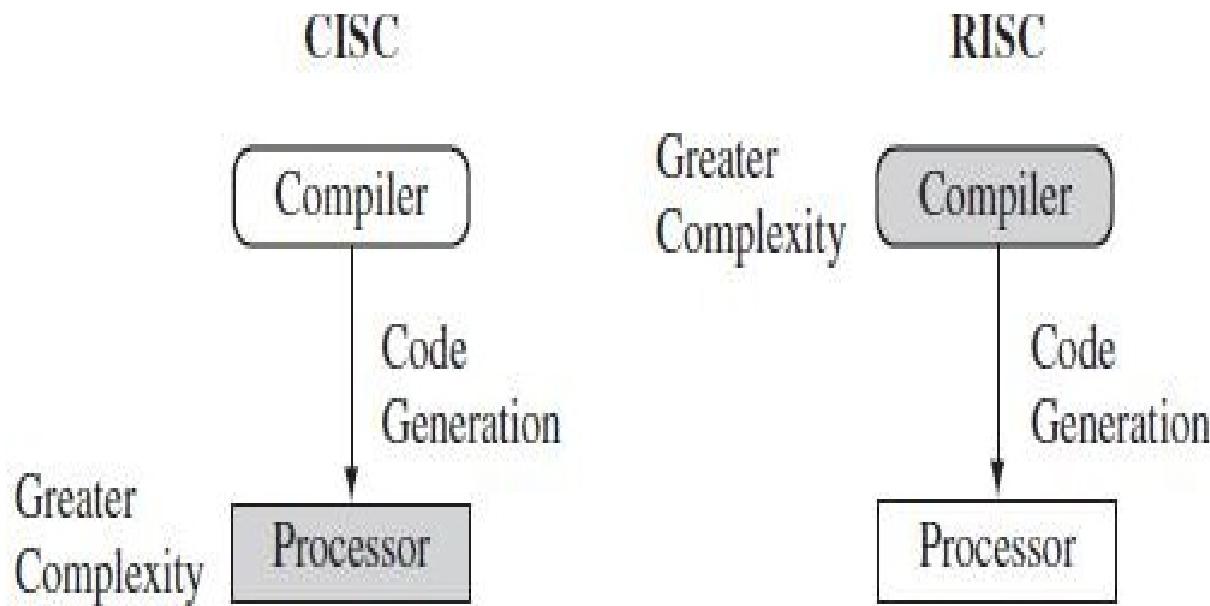
Edit with WPS Office

<b>CISC</b>	<b>RISC</b>
Emphasis on hardware	Emphasis on software
Multiple instruction sizes and formats	Instructions of same set with few formats
Less registers	Uses more registers
More addressing modes	Fewer addressing modes
Extensive use of microprogramming	Complexity in compiler
Instructions take a varying amount of cycle time	Instructions take one cycle time
Pipelining is difficult	Pipelining is easy



# CISC vs. RIS C

---



# 1.1 The RISC design philosophy

- The ARM core uses a RISC architecture.
- RISC aimed at delivering
  - simple and powerful instructions that execute within a single cycle at a high clock speed.
  - As it provides greater flexibility and intelligence in software rather than hardware.
- CISC relies more on the hardware for instruction functionality



Edit with WPS Office

# RISC Design

- Four major ~~Rules~~ design rules:

## 1. *Instructions*—

- Has a reduced number of instruction classes.
- These classes provide simple operations that can each execute in a single cycle.
- Complicated instructions are synthesized by combining simpler instructions.
- Each instruction has fixed length- For Pipelining concept.

**Note:** In CISC –variation in length and no. of cycles for execution.



Edit with WPS Office

# RISC Design Rules

## 2. *Pipelines*:

- The processing of instructions is broken down into smaller units that can be executed in parallel by pipelines.
- Ideally the pipeline advances by one step on each cycle for maximum throughput.
- Instructions can be decoded in one pipeline stage. **Note: In CISC – An instruction is executed by a miniprogram called microcode.**



Edit with WPS Office

# RISC Design Rules

## 3. *Registers*

- RISC machines have a large general-purpose register set.
- Any register can contain either data or an address.
- Registers act as the fast local memory store for all data processing operations.

**Note: In CISC – dedicated registers for specific purposes.**



Edit with WPS Office

# RISC Design

## Rules

### 4. *Load-store architecture:*

- The processor operates on data held in registers.
- Separate load and store instructions transfer data between the register bank and external memory.
- As memory accesses are costly, they perform operations only on register data.

**Note:** In CISC – data processing operations can act on memory directly.



Edit with WPS Office

# 1.2 The ARM Design Philosophy

- Physical features of ARM processor design:
  1. Power: The ARM processor has been specifically designed to be small to reduce power consumption and extend battery operation
    - essential for applications such as mobile phones and personal digital assistants (PDAs).
  2. High Code Density: since embedded systems have limited memory
    - due to cost and/or physical size restrictions.

# Physical features of ARM processor design

## 3. Price sensitive :

§ use slow and low-cost memory devices.

## 4. Reduce the area of the die :

- For a single-chip solution, the smaller the area used by the embedded processor, the more available space for specialized peripherals.
- This in turn reduces the cost.



Edit with WPS Office

# Physical features of ARM processor design

## 5. Debug technology within the processor:

- So the software engineers can view what is happening while the processor is executing code. With greater visibility.
- software engineers can resolve issues faster
- Hence reduces development costs.



Edit with WPS Office

# Physical features of ARM processor design

- Note: The ARM core is not a pure RISC architecture:
  - Primary application is the embedded system. In some sense.
  - the strength of the ARM core is that it does not take the RISC concept too far.
  - In today's systems the key is not raw processor speed but total effective system performance and power consumption.



Edit with WPS Office

# 1.2.1 Instruction Set for Embedded

## 1. *Variable cycle Systems for certain instructions:*

- Not every ARM instruction executes in a single cycle.
- For example, load-store-multiple instructions vary in the number of execution cycles depending upon the number of registers being transferred.(Mainly depends on whether the access is sequential/random)



Edit with WPS Office

# 1.2.1 Instruction Set for Embedded

*2. Inline barrel shifter systems to more complex instructions:*

- The inline barrel shifter is a hardware component that preprocesses one of the input registers before it is used by an instruction.
- This expands the capability of many instructions to improve core performance and code density.



Edit with WPS Office

# 1.2.1 Instruction Set for Embedded

## 3. *Thumb 16-bit instruction set.*

- ARM enhanced the processor core by adding a second 16-bit instruction set called Thumb that permits the ARM core to execute either 16- or 32- bit instructions.
- The 16-bit instructions improve code density by about 30% over 32-bit fixed-length instructions.



Edit with WPS Office

# 1.2.1 Instruction Set for Embedded

## 4. *Conditional execution: Systems*

- An instruction is only executed when a specific condition has been satisfied.
- This feature improves performance and code density by reducing branch instructions.

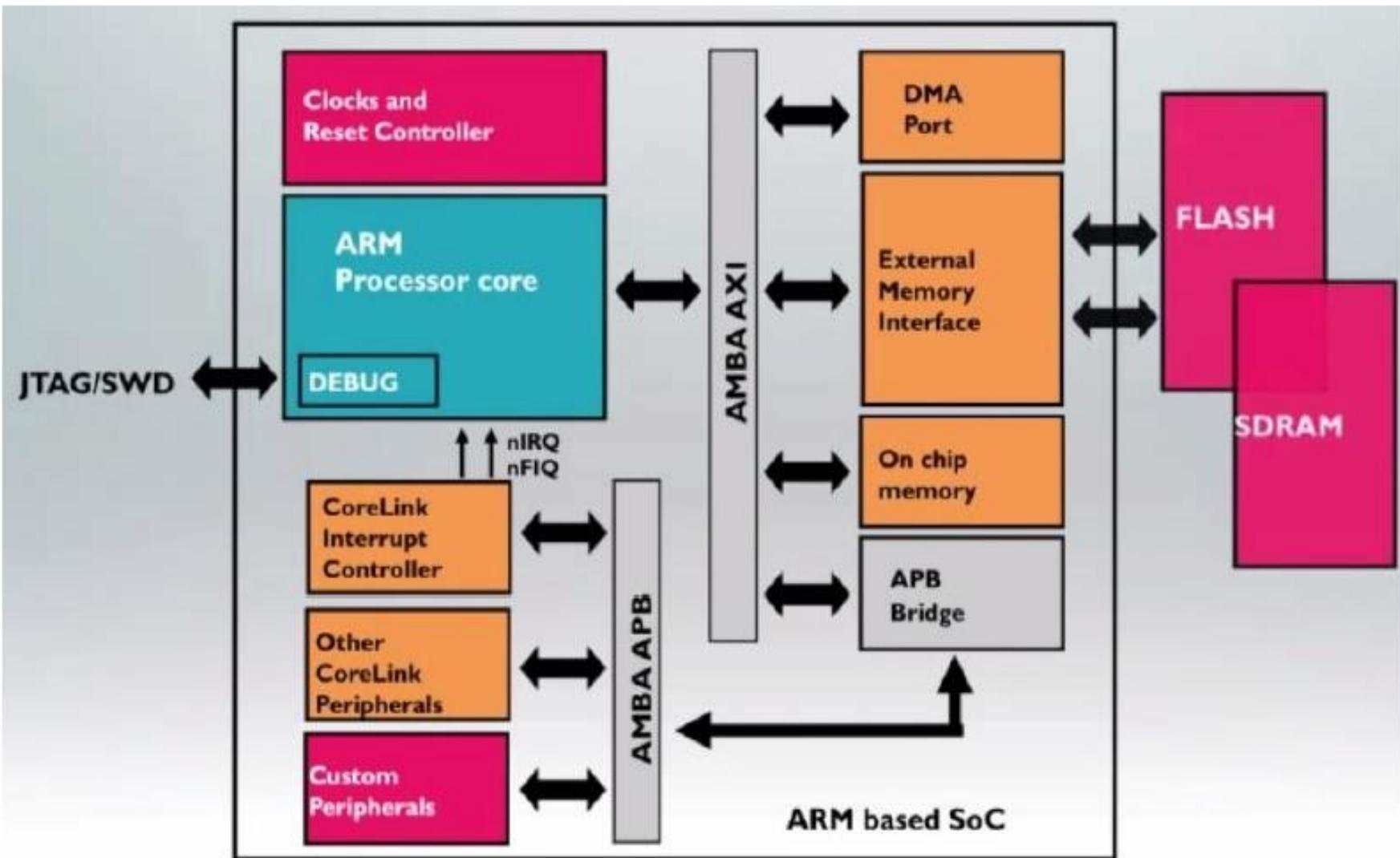
## 5. *Enhanced instructions:*

- The enhanced digital signal processor (DSP) instructions were added to the standard ARM instruction set to support fast  $16 \times 16$ -bit multiplier operations and saturation.
- These instructions allow a faster-performing ARM processor.



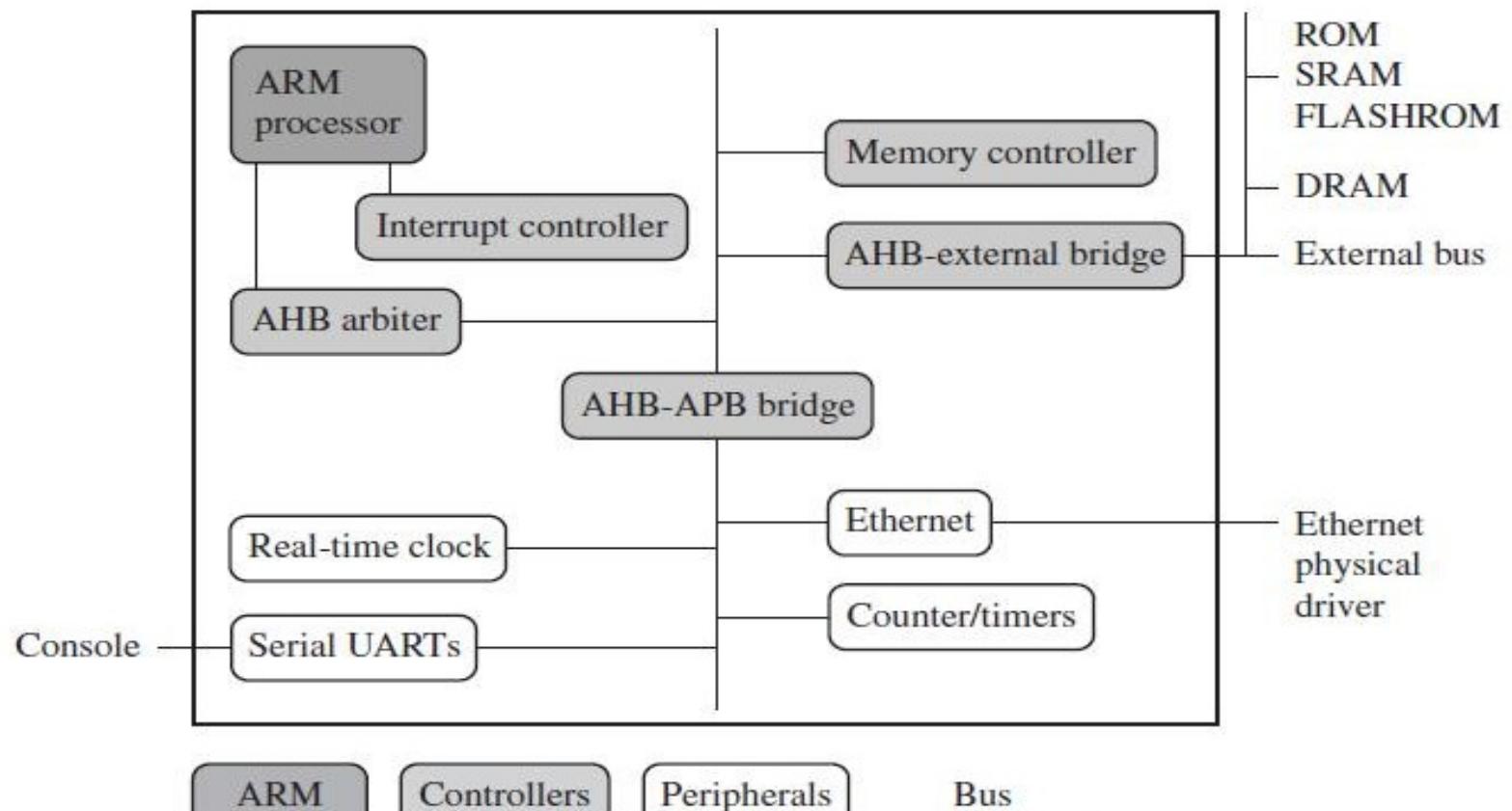
Edit with WPS Office

# Inside ARM based System



# 1.3 Embedded System Hardware

---



---

Figure 1.2 An example of an ARM-based embedded device, a microcontroller.



Edit with WPS Office

# 1.3 Embedded System

## Hardware

- Each box represents a feature or function.
- The lines connecting the boxes are the buses carrying data.
- The device into four main hardware components:
  - *ARM processor*
  - *Controllers*
  - *Peripherals*
  - *Bus*



Edit with WPS Office

# 1.3 Embedded System

## Hardware

### 1. ARM Processor :

- An ARM processor comprises a core (the execution engine that processes instructions and manipulates data)
- It is surrounded with components that interface it with a bus.
- These components can include memory management and caches.



Edit with WPS Office

# 1.3 Embedded System

## 2. *Hardware*

- Coordinates important functional blocks of the system.
- 2 Controllers:
  - Interrupt controllers
  - memory controllers

## 3. *Peripherals*:

- Required for input-output operation external to the chip

## 4. *A bus* :

- Used to communicate between different parts of the device



Edit with WPS Office

## 1.3.1 ARM Bus

### Technology

- Embedded systems use different bus technologies than those designed for x86 PC.
  - Embedded device use an **on-chip bus**
  - Core is master who initiates a data transfer.
- A Bus has two architecture levels
  - The First is a physical level that covers the **electrical characteristics and bus width (16, 32, or 64 bits)**.
  - The Second level deals with protocol.– the logical rules governing the **communication between processor and peripheral**.
- ARM seldom implements the electrical characteristics of the bus, but it routinely specifies the bus protocol.



# 1.3.2 AMBA Bus Protocol

- **AMBA:** Advanced Micro controller Bus Architecture
  - 1996, it's introduced and widely adopted as the on-chip bus architecture for ARM processors.
  - The first AMBA buses introduced were
    - ASB : ARM System Bus, and
    - APB : ARM Peripheral Bus
  - Later, ARM introduced another bus design
    - AHB: ARM High-performance Bus
- Using AMBA,
  - peripheral designers can reuse the same design on multiple projects (with different processor architecture).
  - Plug-and-play



Edit with WPS Office

# 1.3.2 AMBA Bus Protocol

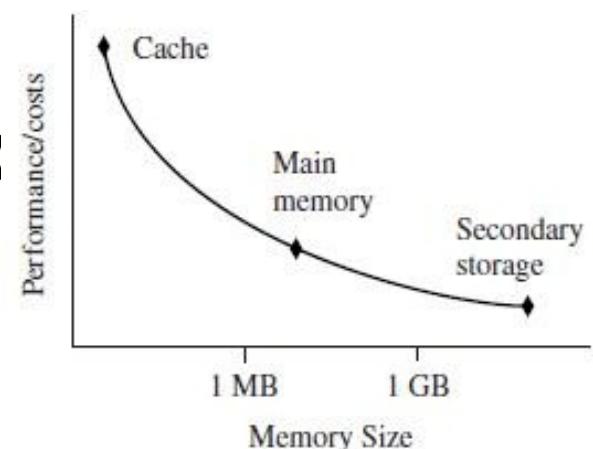
- AHB
  - provides higher data throughput than ASB. Because
    - It use a Centralized Multiplexed Bus Scheme
    - This change allows the AHB bus to run at higher clock speed.
    - 64/128 bits width.
- Two variations on the AHB bus
  - Multi-layer AHB, and
    - allows multiple active bus masters,
  - AHB-Lite: only one master



Edit with WPS Office

### 1.3.3 Memory

- Memory is necessary
  - An embedded system has to have some form of memory to store and execute code.
- You have to consider
  - price, performance, and power consumption
- Specific memory characteristics
  - hierarchy, width, and type



### 1.3.3

## Memory(Hierarchy)

- Cache
  - is used to speed up data transfer between Core and Main Memory (DRAM);
- But,
  - It makes the performance unpredicted;
  - It doesn't help Real-Time system response;
    - Note that many small embedded systems do not require the benefit of a cache.



Edit with WPS Office

### 1.3.3

- Memory width:  

## Memory(Width)

  - In number of bits the memory returns on each access
    - 8, 16, 32, or 64 bits.
  - 32-bitARM instructions and **16-bit-wide memory chips**:
    - then the processor will have to make two memory fetches per instruction. Each fetch requires two 16-bit loads.
  - Hence it is better to have 16-bit Thumb instructions with 16-bit-wide memory .

Fetching instructions from memory.

Instruction size	8-bit memory	16-bit memory	32-bit memory
ARM 32-bit	4 cycles	2 cycles	1 cycle
Thumb 16-bit	2 cycles	1 cycle	1 cycle



## 1.3.3

# Memory(Types)

- DRAM
  - the most commonly used RAM for devices;
  - Dynamic: need to have its storage cells refreshed and given a new electronic charge every few milliseconds, so you need to set up a DRAM controller before using the memory.
- SRAM
  - is faster than the more traditional DRAM (SRAM does not require a pause between data access).
- SDRAM
  - is one of many subcategories of DRAM.
  - accessed pipelined, transferred in a burst.



Edit with WPS Office

## 1.3.4

# Peripherals

- Embedded system that interact with the outside world need some form of peripheral device.
  - Peripherals range from a simple serial communication device to a more complex 802.11 wireless device.
- All ARM peripherals are memory mapped – the programming interface is a set of memory addressed register.
- Controllers are specialized peripherals that implement higher level of functionality within an embedded system.
  - Two important types of controllers are
    - Memory Controller
    - Interrupt Controller
      - Normal IC
      - Vectoring IC



Edit with WPS Office

## 1.3.4 Peripherals (Memory

- Memory **Controller**) connect different types of memory to the processor bus.
  - On power-up a memory controller is configured in hardware to allow certain memory device to be active. These memory devices allow the initialization code to be executed.
  - Some memory devices must be set up by software.
    - e.g. When using DRAM, you first have to set up the memory timings and refresh rate before it can be accessed.



Edit with WPS Office

# 1.3.4 Peripherals (Interrupt Controller)

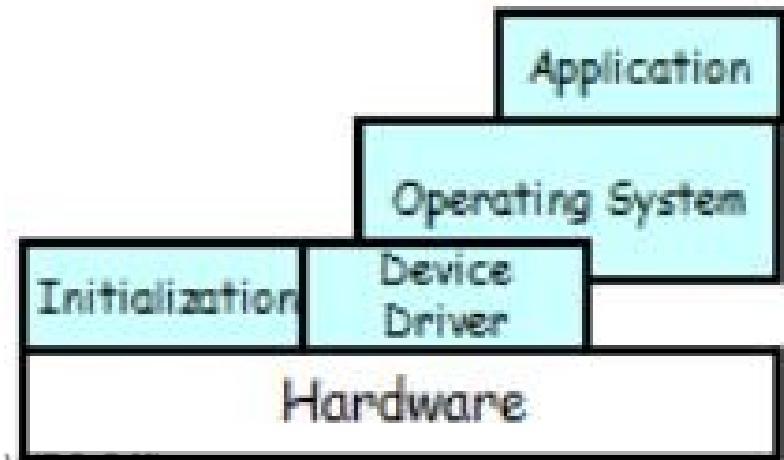
- When a peripheral or device requires attention,
  - it raise an interrupt to the processor.
- An interrupt controller
  - provides a programmable governing policy
- There are two types of interrupt controller available for the ARM processor
  - Standard interrupt controller
    - Sends an interrupt signal; Can be programmed to ignore or mask an individual or set of devices.
    - It's interrupt handler determines which device requiring service.
  - Vector interrupt controller (VIC)
    - Associate a "priority" and a "handler address" to each interrupt.
    - Depending on its type, VIC will either call the standard interrupt exception handler (loading the handler address from VIC) or cause core to jump to the handler for the device directly.



Edit with WPS Office

# 1.4 Embedded System Software

- An embedded system needs software to drive it.
- There are four typical software components required to control an embedded device.
  - Each software component in the stack uses a higher level of abstraction to separate the code from the hardware device.
  - Initialization Code (e.g. Boot loader)
  - Operating System
  - Device Drivers
  - Application



# 1.4.1 Initialization (Boot)

## Code

- Initialization code (or boot code)
  - takes the processor from the reset state to a state where the operating system can run.
    - Configuring memory controller, caches
    - Initializing some devices
    - \* Debug Monitor (replace OS in simple system)
- Three phases
  - Initial hardware configuration
    - Satisfy the requirements of the booted image
      - e.g. re-organization of the memory map
  - Diagnostics
    - Fault identification and isolation
  - Booting
    - Loading an image and handing control over to the image
    - The boot process may be complicated if the system must boot different operating systems or different versions of the same operating system



# 1.4.1 Initialization (Boot) Code

## Example: Memory

- Start from **Reorganization** ROM
- Remap to RAM
  - easy IVT modification

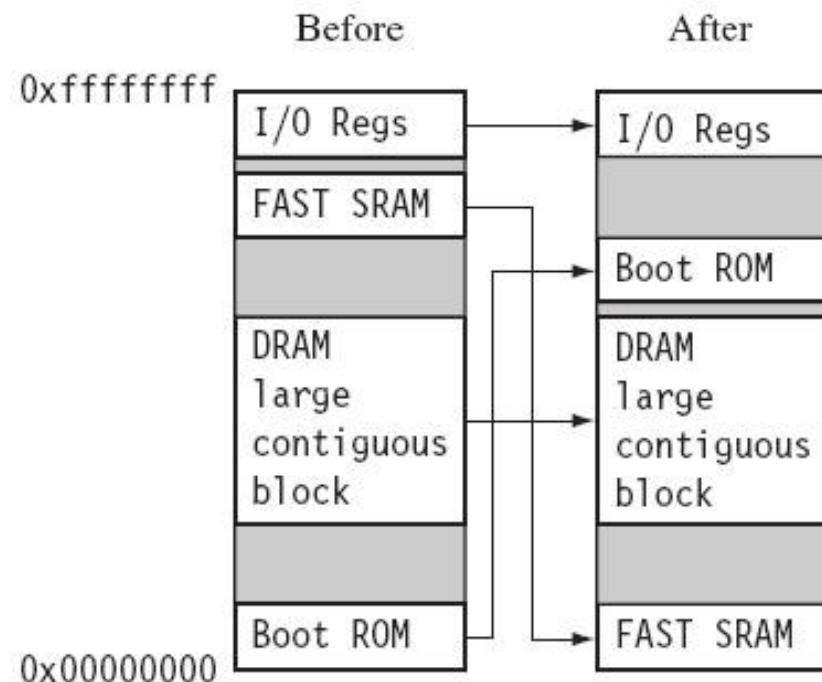


Figure 1.5 Memory remapping.



Edit with WPS Office

## 1.4.2 Operating System

- OS organizes the system resources
  - peripherals, memory, and processing time
    - With an OS controlling these resources, they can be efficiently used by different applications running within the OS environment.
- ARM processors support over 50 OSes
  - Two main categories: RTOS, platform OS
    - **RTOS**: guarantee response times to event
    - **platform OS**: require MMU(**Memory Mgmt Unit**) and tend to have secondary storage (for large application).
      - These two categories of OSes are not mutually exclusive.
  - ARM has developed a set of processor cores that specially target each category.



Edit with WPS Office

### 1.4.3

- The OS ~~sets up applications~~ applications
  - code dedicated to handling a particular task.
- ARM processors are found in numerous market segments, including
  - networking, automotive, mobile and consumer devices, mass storage, and imaging.
- In contrast, ARM processors are not found in applications that require leading-edge high performance.(Because they will be low volume and high cost)

# ARM Processor Fundamentals

## Data-path in ARM

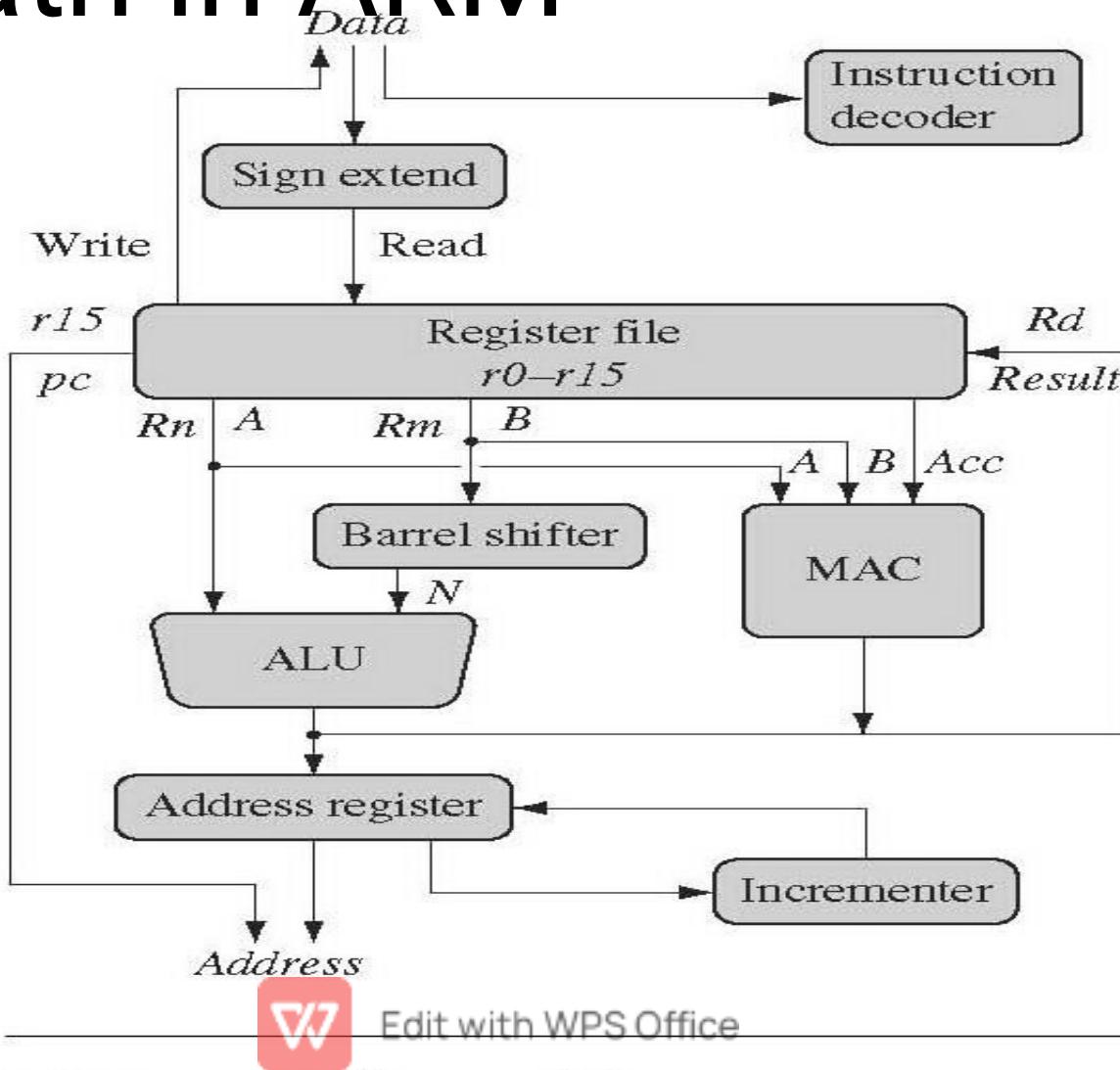


Figure 2.1 ARM core dataflow model.

## 2.1

# Registers

- Orthogonal Registers (ref. VAX, PDP-11)
  - We say R0~R13 are orthogonal, for given instruction, if it can use R0, then others can also be used.
- SPRs
  - R13(stack pointer), R14(link register), R15(program counter)
- Current PSR/Saved PSR
  - Condition Codes: N, Z, C, V
  - Interruption mask: I(IRQ), F(FIQ)
  - Thumb Enable Bit
  - Mode( 5 bit )

<i>r0</i>
<i>r1</i>
<i>r2</i>
<i>r3</i>
<i>r4</i>
<i>r5</i>
<i>r6</i>
<i>r7</i>
<i>r8</i>
<i>r9</i>
<i>r10</i>
<i>r11</i>
<i>r12</i>
<i>r13 sp</i>
<i>r14 lr</i>
<i>r15 pc</i>

<i>cpsr</i>
-

Figure 2.2

Registers available in *user mode*.



Edit with WPS Office

# 2.2 Current Program Status Register

- The *cpsr*: Dedicated 32-bit register and resides in the register file

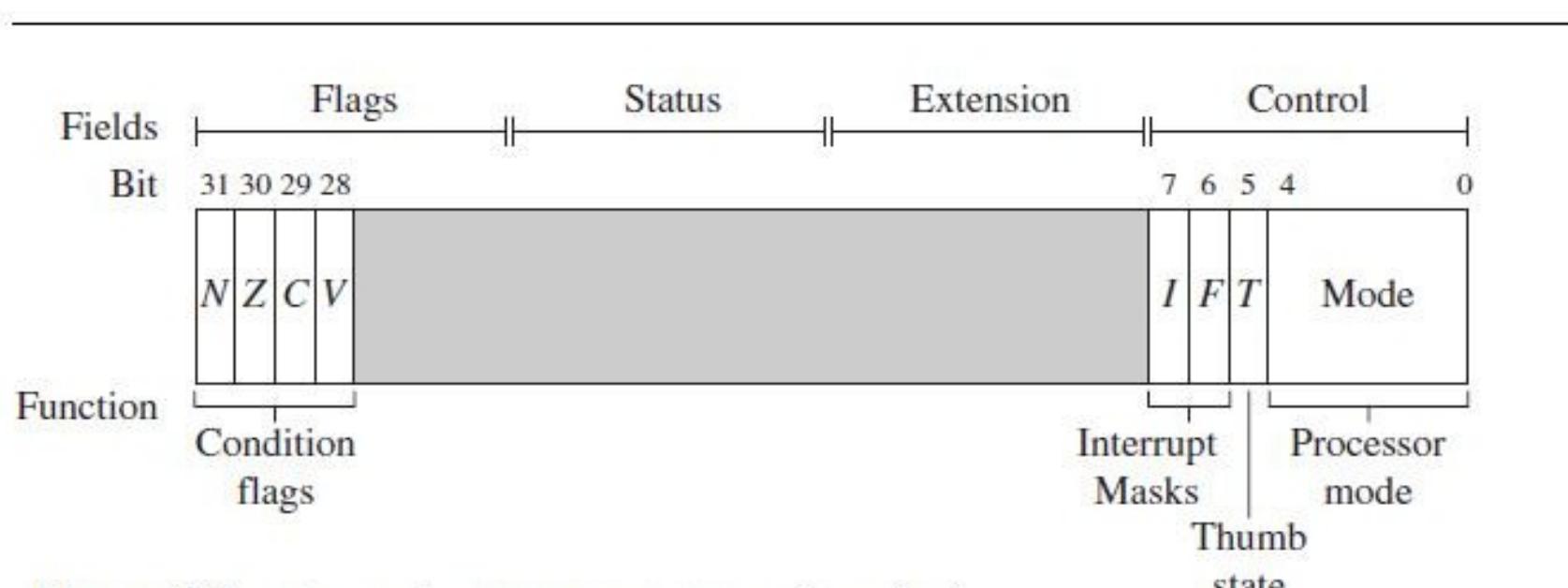


Figure 2.3 A generic program status register (*psr*).



Edit with WPS Office

# 2.2 Current Program Status Register

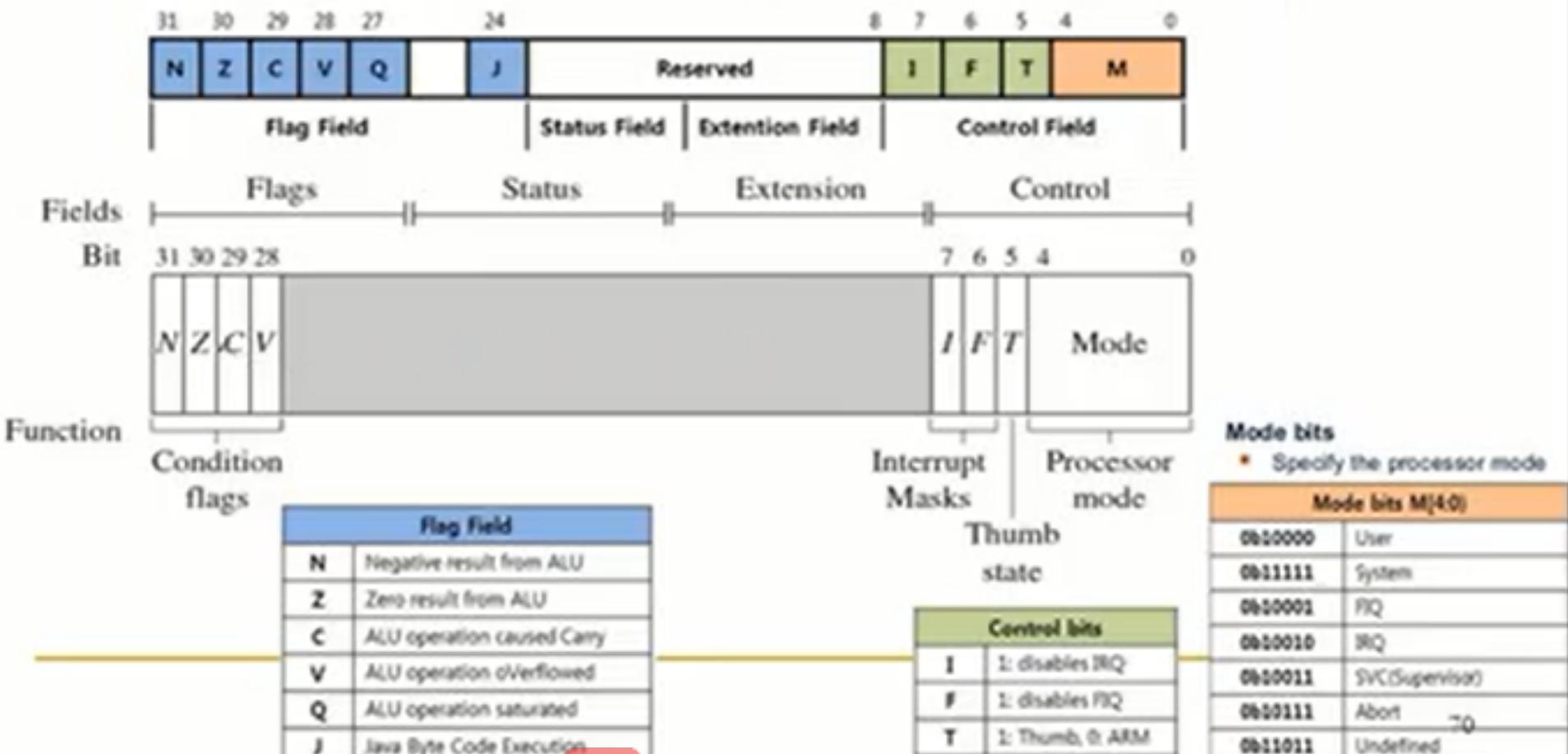
- *cpsr* :
  - Divided into four fields, each 8 bits wide:
    - flags, status, extension, and control.
  - The control field contains the processor mode, state, and interrupt mask bits.
  - The flags field contains the condition flags.
    - Condition Codes: N, Z, C, V
    - Interruption mask: I(IRQ), F(FIQ)
    - Thumb Enable Bit
    - Mode(5-bit)



Edit with WPS Office

# CURRENT PROGRAM STATUS REGISTERS

- » The ARM core uses the *cpsr* to monitor and control internal operations.
- » The *cpsr* is a dedicated 32-bit register and resides in the register file.
- » The following Figure shows the basic layout of a generic program status register



## 2.2.1 Processor Modes

- Determines which registers are active and the access rights to the *cpsr* register
- Two modes: either privileged or nonprivileged
  - A privileged mode allows full read-write access to the *cpsr*.
  - In nonprivileged mode only allows read access to the control field in the *cpsr* but still allows read- write access to the condition flags.



Edit with WPS Office

## 2.2.1 Processor Modes

- There are seven processor modes in total:
  - six privileged modes
    - *abort, fast interrupt request, interrupt request, supervisor, system, and undefined* one
  - one nonprivileged mode
    - *user*



Edit with WPS Office

- The ARM has seven basic operating modes:

- **User** : unprivileged mode under which most tasks run
- **FIQ** : entered when a high priority (fast) interrupt is raised
- **IRQ** : entered when a low priority (normal) interrupt is raised
- **Supervisor** : entered on reset and when a Software Interrupt instruction is executed
- **Abort** : used to handle memory access violations
- **Undef** : used to handle undefined instructions
- **System** : privileged mode using the same registers as user mode



Edit with WPS Office

## 2.2.2 Banked Registers

*User and system*

<i>r0</i>					
<i>r1</i>					
<i>r2</i>					
<i>r3</i>					
<i>r4</i>					
<i>r5</i>					
<i>r6</i>					
<i>r7</i>					
<i>r8</i>					
<i>r9</i>					
<i>r10</i>					
<i>r11</i>					
<i>r12</i>					
<i>r13 sp</i>					
<i>r14 lr</i>					
<i>r15 pc</i>					
<i>cpsr</i>					
-	<i>spsr_fiq</i>	<i>spsr_irq</i>	<i>spsr_svc</i>	<i>spsr_undef</i>	<i>spsr_abt</i>



Edit with WPS Office

## 2.2.2 Banked Registers

- 37 registers in the register file:
  - 20 registers are hidden
    - (Banked Registers-shaded in diagram)

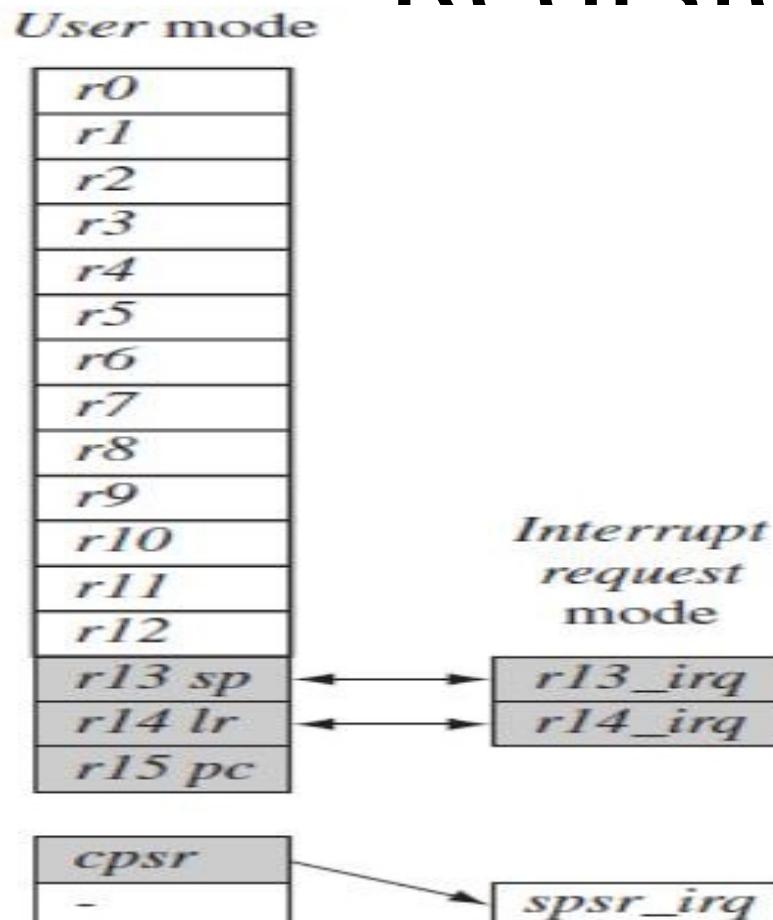
Table 2.1 Processor mode.

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



Edit with WPS Office

## 2.2.2 Banked Registers



- As per the diagram the change of mode from user->IRQ
- - values of register changes

Figure 2.5      Changing mode on an exception.



Edit with WPS Office

## 2.2.3 State and Instruction Sets

- There are three instruction sets: ARM, Thumb, and Jazelle.

Table 2.2 ARM and Thumb instruction set features.

	ARM ( <i>cpsr T = 0</i> )	Thumb ( <i>cpsr T = 1</i> )
Instruction size	32-bit	16-bit
Core instructions	58	30
Conditional execution <sup>a</sup>	most	only branch instructions
Data processing instructions	access to barrel shifter and ALU	separate barrel shifter and ALU instructions
Program status register	read-write in privileged mode	no direct access
Register usage	15 general-purpose registers +pc	8 general-purpose registers +7 high registers +pc



## 2.2.3 State and Instruction Sets

Table 2.3 Jazelle instruction set features.

Jazelle ( <i>cpsr T = 0, J = 1</i> )	
Instruction size	8-bit
Core instructions	Over 60% of the Java bytecodes are implemented in hardware; the rest of the codes are implemented in software.



Edit with WPS Office

## 2.2.4 Interrupt Masks

- Interrupt masks are used to stop specific interrupt requests from interrupting the processor.
- Two interrupt request levels available on the
  - *interrupt request* (IRQ) =>(I bit=1)
  - *fast interrupt request* (FIQ) =>(F bit=1)



Edit with WPS Office

## 2.2.5 Condition Flags

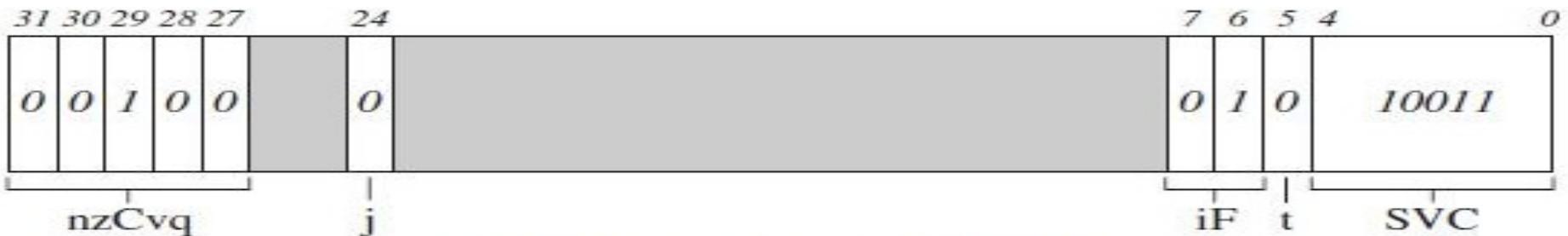
---

Flag	Flag name	Set when
Q	Saturation	the result causes an overflow and/or saturation
V	oVerflow	the result causes a signed overflow
C	Carry	the result causes an unsigned carry
Z	Zero	the result is zero, frequently used to indicate equality
N	Negative	bit 31 of the result is a binary 1

---



## 2.2.5 Condition Flags



Edit with WPS Office

# 2.2.6 Conditional Execution

Condition mnemonics.

Mnemonic	Name	Condition flags
EQ	equal	Z
NE	not equal	z
CS HS	carry set/unsigned higher or same	C
CC LO	carry clear/unsigned lower	c
MI	minus/negative	N
PL	plus/positive or zero	n
VS	overflow	V
VC	no overflow	v
HI	unsigned higher	zC
LS	unsigned lower or same	Z or c
GE	signed greater than or equal	NV or nv
LT	signed less than	Nv or nV
GT	signed greater than	NzV or nzv
LE	signed less than or equal	Z or Nv or nV
AL	always (unconditional)	ignored



Edit with WPS Office



## 2.3 Pipeline

- A pipeline is the mechanism a RISC processor uses to execute instructions.
  - Using a pipeline speeds up execution by fetching the next instruction while other instructions are being decoded and executed.

**Figure 2.7 ARM7 Three-stage pipeline.**



## 2.3 Pipeline(Example)

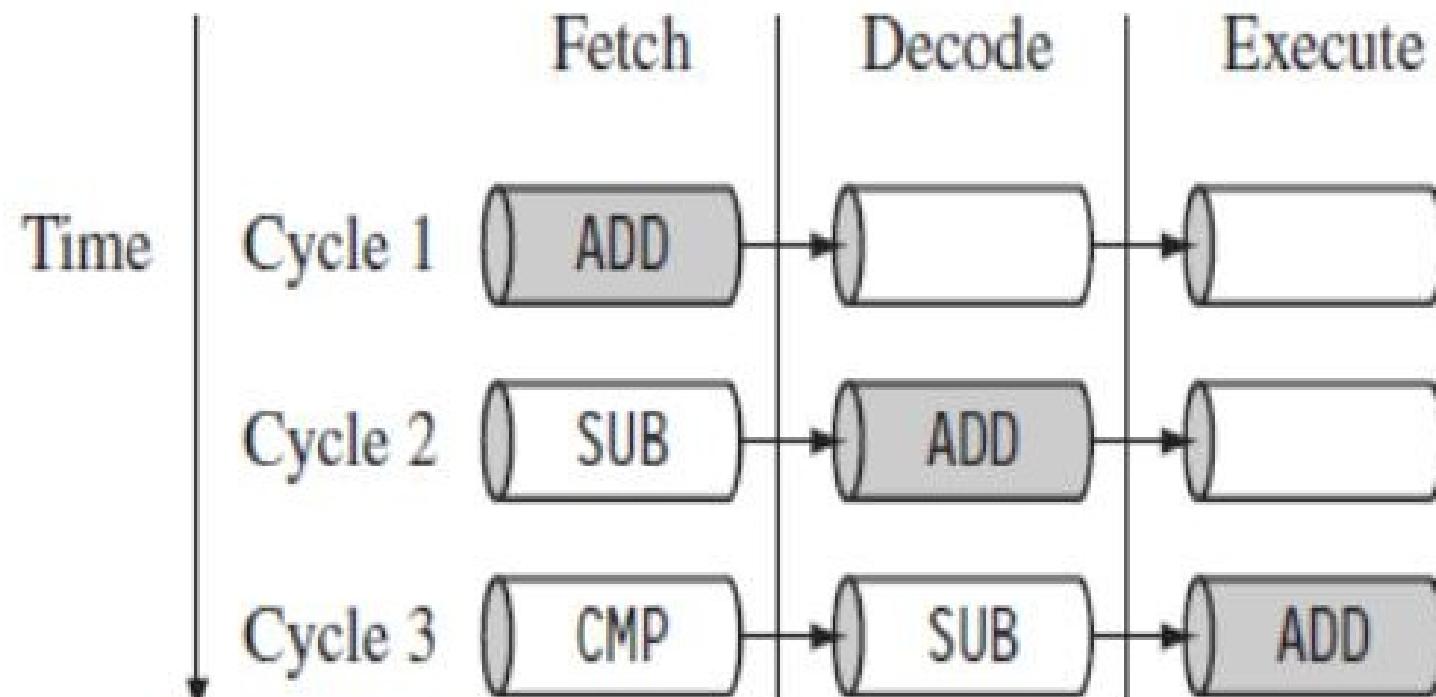


Figure 2.8 Pipelined instruction sequence.



## 2.3 Pipeline

- ARM7
  - 3 stages: Fetch, Decode, Execute
    - More stages (deeper pipeline)
  - means “More latency”, “More Dependence”
- ARM9 (+13% ARM7)
  - 5 stages: FI, DI, EX, M, WB
- ARM10 (+34% ARM7)
  - 6 stages: FI, Issue, DI, EX, M, WB
- ARM7 instruction runs on ARM9/10 ?
  - Yes, same pipeline architecture as ARM7



## 2.3 Pipeline(Example)

---

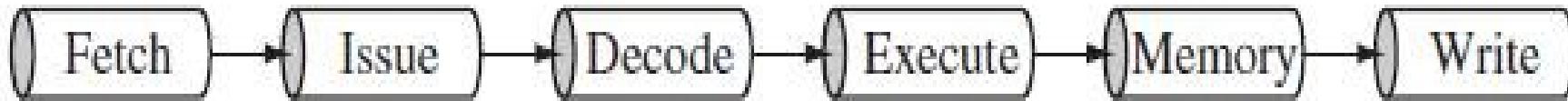


Figure 2.9 ARM9 five-stage pipeline.

---

Figure 2.10 ARM10 six-stage pipeline.

---





# 2.3.1 Pipeline Executing

Figure 2.11 ARM instruction sequence.

## Characteristics

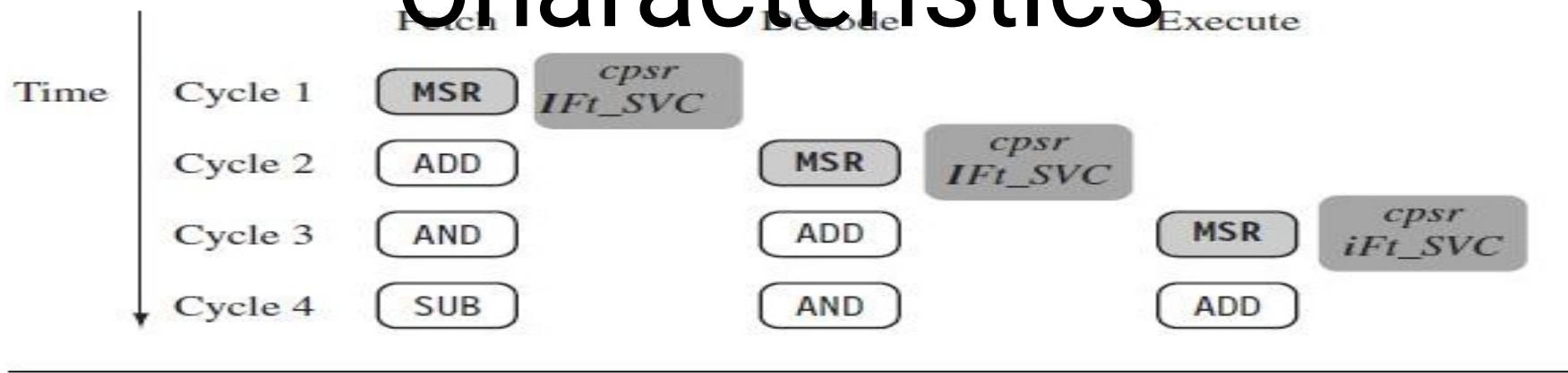
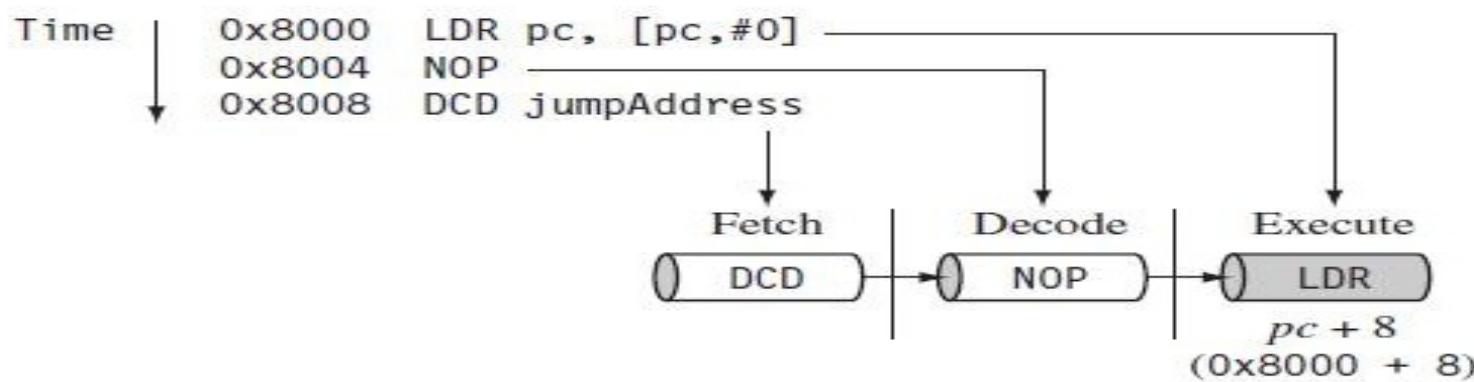


Figure 2.12 Example:  $pc = \text{address} + 8$ .



Edit with WPS Office



## 2.4 Exceptions, Interrupts, and the Vector

- When an exception or interrupt occurs:
  - the processor sets the *pc* to a specific memory address.
  - The address is within a special address range called the *vector table*. The entries in the vector table are instructions that branch to specific routines designed to handle a particular exception or interrupt.

# 2.4 Exceptions, Interrupts, and the Vector



- The memory map address 0x00000000  
**Table**
  - Is reserved for the vector table, a set of 32-bit words.
  - the vector table can be optionally located at a higher address in memory (starting at the offset 0xffff0000)



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt



# Vector Table

## (Entries)

Each vector table entry contains a form of branch instruction pointing to the start of a specific routine:

- Reset vector
  - It is the location of 1st instruction after power-up;
- Undefined instruction vector
  - Is used when processor cannot decoded instruction.
- Software interrupt vector is called when:
  - SWI instruction being executed (To invoke OS routine)
- Prefetch Abort vector (PABT) occurs :
  - try to access invalid address for instruction;
- Data Abort (DABT)
  - Try to access invalid address for data;
- IRQ
- FIQ



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

# 2.5 Core Extensions

- Cache & Tightly Coupled Memory(TCM)
  - Unified vs. I/D
  - TCM: fast SRAM, very near Core (unwired with AMBA)

Figure 2.13 A simplified Von Neumann architecture with cache.

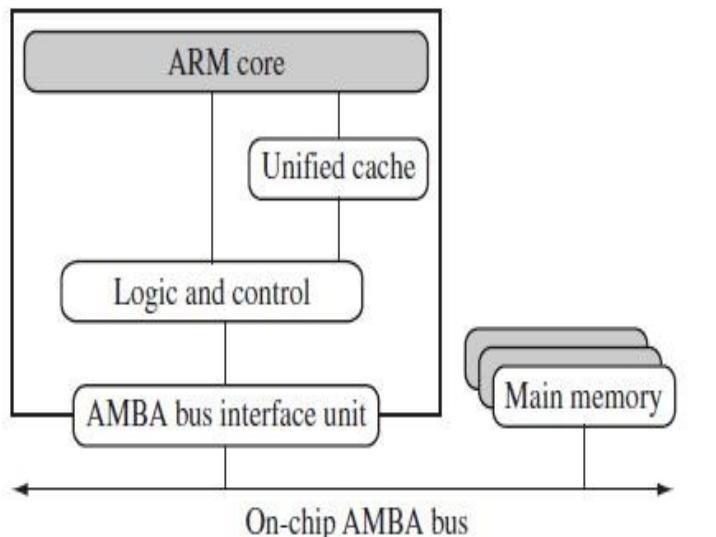
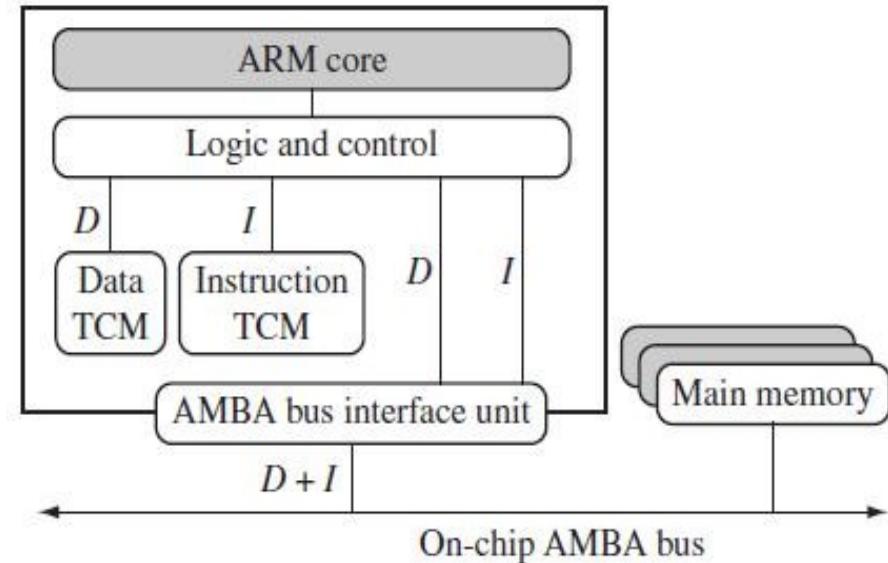


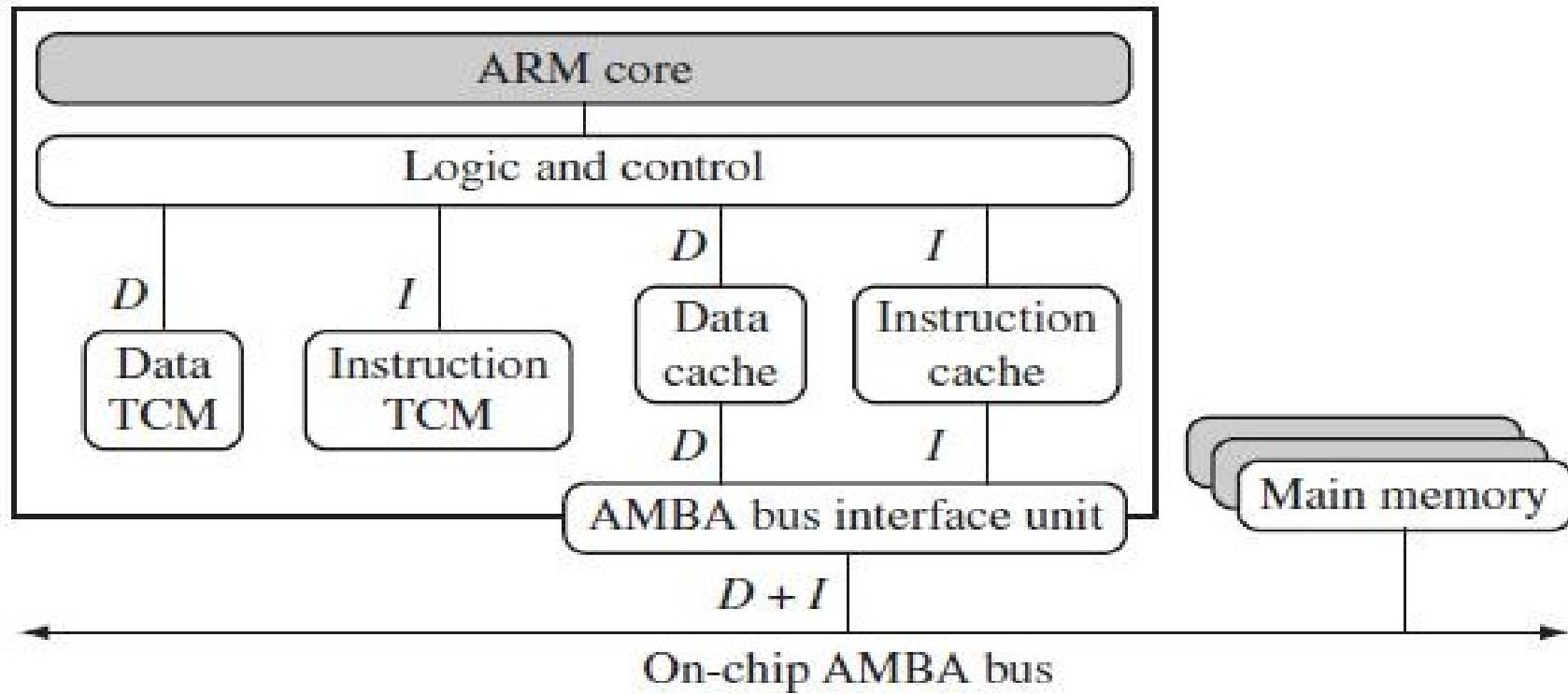
Figure 2.14 A simplified Harvard architecture with TCMs.





# 2.5 Core Extensions

Figure 2.15 A simplified Harvard architecture with caches and TCMs.





# 2.5 Core Extensions

- Memory Management interface
  - No MM: for simple embedded system;
  - MPU(Memory Protect Unit): section protection;
  - MMU is an h/w available on ARM: Translation table to provide Fine-grain protection on memory (Virtual- Physical address Map)
- Coprocessor interface
- The coprocessor can be accessed through a group of dedicated ARM instructions that provide a load-store type interface.
  - By Extend Instruction Set vs. CSR register;
  - E.g.
    - CP15: cache, TCM and MMU via load/store like instr.



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

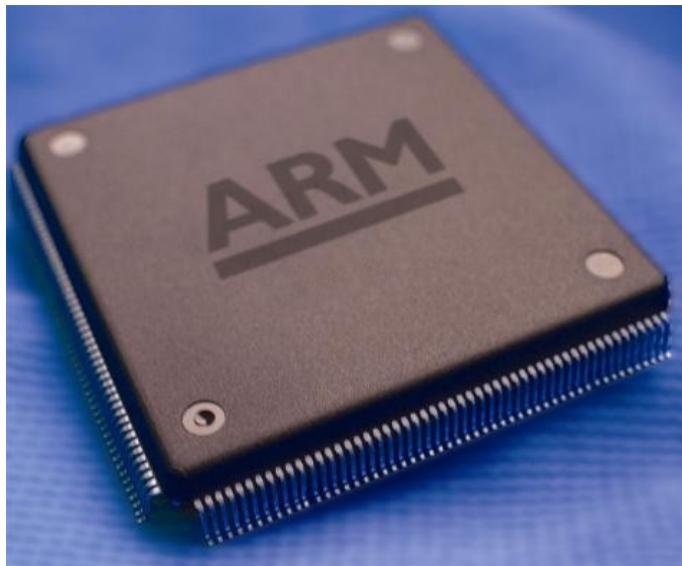


**Mrs. Ashwini N**  
Assistant Professor  
Dept.of.ISE  
BMSIT&M, Bengaluru Email:  
ashwinilaxman@bmsit.in

# Module – 2

## Microprocessor and Microcontroller

### (18CS44)



Edit with WPS Office

# Introduction to the ARM Instruction Set

- The ARM Architecture is a **Load/Store** architecture
  - No direct manipulation of memory contents
  - Memory must be loaded into the CPU to be modified, then written back out
- Cores are either in ARM *state* or Thumb *state*
  - This determines which instruction set is being executed
  - An instruction must be executed to switch between states
- The architecture allows programmers and compilation tools to reduce branching through the use of conditional execution
  - Method differs between ARM and Thumb, but the principle is that most (ARM) or all (Thumb) instructions can be executed conditionally.



Edit with WPS Office

# Introduction to the ARM Instruction

- Each instruction **Set** has the following format:

PRE	<pre-conditions>
	<instruction/s>
POST	<post-conditions>

- In the pre- and post-conditions, memory is denoted as
  - mem<data\_size>[address]
- **ARM instructions process data held in registers and only access memory with load and store instructions.**
- Example:

Instruction Syntax	Destination register ( <i>Rd</i> )	Source register 1 ( <i>Rn</i> )	Source register 2 ( <i>Rm</i> )
ADD r3, r1, r2	<i>r3</i>	<i>r1</i>	<i>r2</i>



# 3.1 Data Processing

- These instructions operate on the contents of registers
  - They DO NOT affect memory

	arithmetic		logical		move
manipulation (has destination register)	ADC ADD	SBC SUB RSB RSC	BIC AND	ORR EOR ORN	MVN MOV
comparison (set flags only)	CMN (ADDs)	CMP (SUBs)	TST (ANDs)	TEQ (EORS)	

- Syntax:  
 $\langle\text{Operation}\rangle\{\langle\text{cond}\rangle\} \{S\} \{Rd, \} \ Rn, \ \text{Operand2}$
- Examples:
  - ADD r0, r1, r2 ;  $r0 = r1 + r2$
  - TEQ r0, r1 ; if  $r0 = r1$ , Z flag will be set
  - MOV r0, r1 ; copy r1 to r0



## 3.1.1 Move Instructions

- Syntax:

Syntax: <instruction>{<cond>} {S} Rd, N

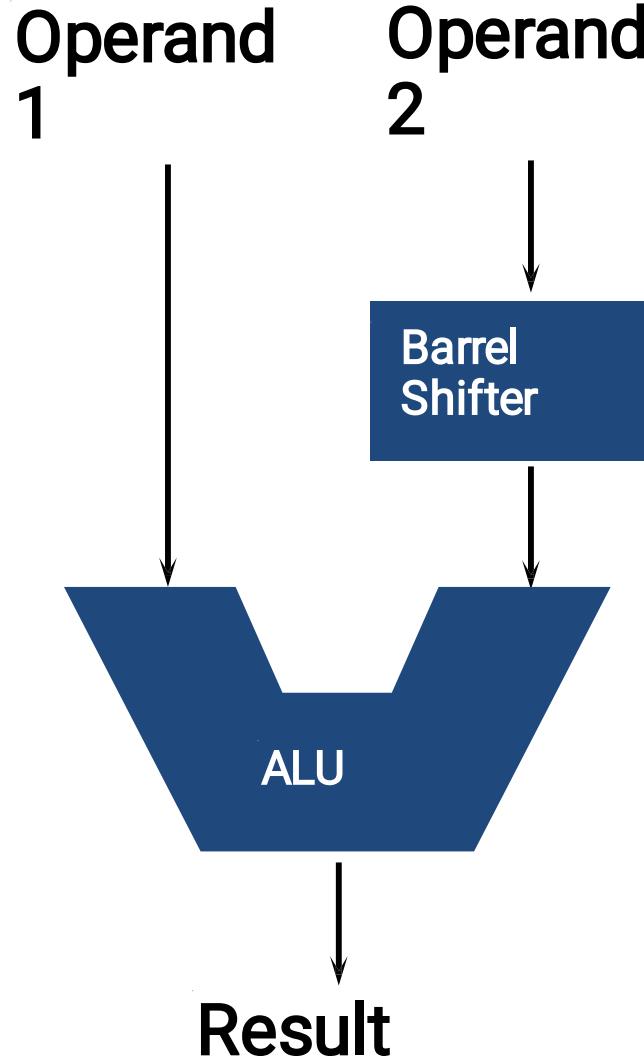
MOV	Move a 32-bit value into a register	$Rd = N$
MVN	move the NOT of the 32-bit value into a register	$Rd = \sim N$



Edit with WPS Office



# Using a Barrel Shifter: The 2nd Operand



## Operan

Register, optionally with shift operation

- Shift value can be either be:
  - 5 bit unsigned integer
  - Specified in bottom byte of another register.
- Used for multiplication by constant

Immediate value

- 8 bit number, with a range of 0 -255.
  - Rotated right through even number of positions
- Allows increased range of 32-bit constants to be loaded directly into registers



Edit with WPS Office

# Barrel Shifter Operation

Table 3.3 Barrel shift operation syntax for data processing instructions.

<i>N</i> shift operations	Syntax
Immediate	#immediate
Register	Rm
Logical shift left by immediate	Rm, LSL #shift_imm
Logical shift left by register	Rm, LSL Rs
Logical shift right by immediate	Rm, LSR #shift_imm
Logical shift right with register	Rm, LSR Rs
Arithmetic shift right by immediate	Rm, ASR #shift_imm
Arithmetic shift right by register	Rm, ASR Rs
Rotate right by immediate	Rm, ROR #shift_imm
Rotate right by register	Rm, ROR Rs
Rotate right with extend	Rm, RRX



# Barrel Shifter

Table 3.2 Barrel shifter operations.

Mnemonic	Description	Shift	Result	Shift amount y
LSL	logical shift left	$x \text{LSL } y$	$x \ll y$	#0–31 or $Rs$
LSR	logical shift right	$x \text{LSR } y$	$(\text{unsigned})x \gg y$	#1–32 or $Rs$
ASR	arithmetic right shift	$x \text{ASR } y$	$(\text{signed})x \gg y$	#1–32 or $Rs$
ROR	rotate right	$x \text{ROR } y$	$((\text{unsigned})x \gg y) \mid (x \ll (32 - y))$	#1–31 or $Rs$
RRX	rotate right extended	$x \text{RRX}$	$(c \text{ flag} \ll 31) \mid ((\text{unsigned})x \gg 1)$	none

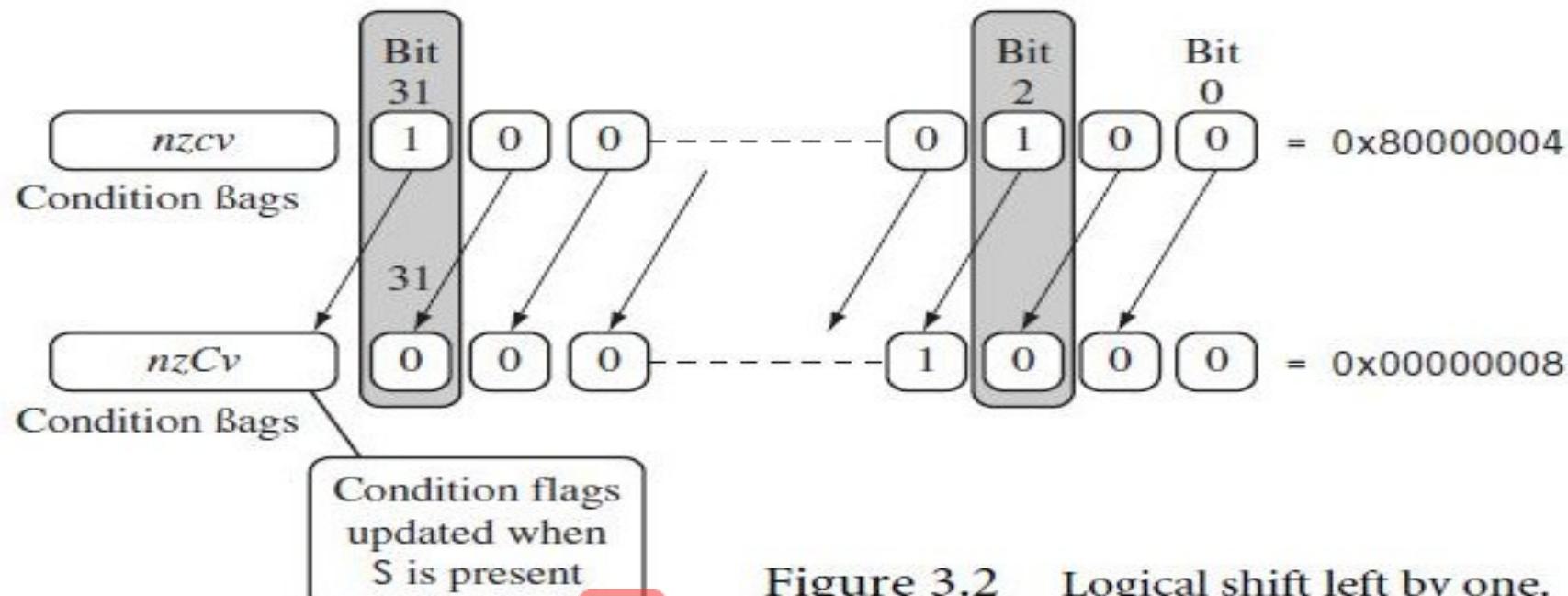


Figure 3.2 Logical shift left by one.

# 3.1.3 Arithmetic Instructions

Syntax: <instruction>{<cond>} {S} Rd, Rn, N

ADC	add two 32-bit values and carry	$Rd = Rn + N + \text{carry}$
ADD	add two 32-bit values	$Rd = Rn + N$
RSB	reverse subtract of two 32-bit values	$Rd = N - Rn$
RSC	reverse subtract with carry of two 32-bit values	$Rd = N - Rn - !(\text{carry flag})$
SBC	subtract with carry of two 32-bit values	$Rd = Rn - N - !(\text{carry flag})$
SUB	subtract two 32-bit values	$Rd = Rn - N$

N is the result of the shifter operation. The syntax of shifter operation is shown in Table 3.3.



Edit with WPS Office

### 3.1.4 Using the Barrel Shifter with Arithmetic Instructions

- The wide range of second operand shifts available on arithmetic and logical instructions is a very powerful feature of the ARM instruction set.
- The instruction multiplies the value stored in register  $r1$  by three.

PRE       $r0 = 0x00000000$   
             $r1 = 0x00000005$

ADD       $r0, r1, r1, LSL \#1$

POST      $r0 = 0x0000000f$   
             $r1 = 0x00000005$



Edit with WPS Office

## 3.1.5 Logical Instructions

- Logical instructions perform bitwise logical operations on the two source registers

Syntax: <instruction>{<cond>} {S} Rd, Rn, N

AND	logical bitwise AND of two 32-bit values	$Rd = Rn \& N$
ORR	logical bitwise OR of two 32-bit values	$Rd = Rn   N$
EOR	logical exclusive OR of two 32-bit values	$Rd = Rn \wedge N$
BIC	logical bit clear (AND NOT)	$Rd = Rn \& \sim N$



Edit with WPS Office

## 3.1.6 Comparison Instructions

- Used to compare or test a register with a 32-bit value.
- They update the *cpsr* flag bits according to the result, but do not affect other

~~registers~~

Syntax: <instruction>{<cond>} Rn, N

CMN	compare negated	flags set as a result of $Rn + N$
CMP	compare	flags set as a result of $Rn - N$
TEQ	test for equality of two 32-bit values	flags set as a result of $Rn \wedge N$
TST	test bits of a 32-bit value	flags set as a result of $Rn \& N$



Edit with WPS Office

# 3.1.7 Multiply Instructions

Syntax: **MLA**{<cond>} {S} Rd, Rm, Rs, Rn  
**MUL**{<cond>} {S} Rd, Rm, Rs

<b>MLA</b>	multiply and accumulate	$Rd = (Rm * Rs) + Rn$
<b>MUL</b>	multiply	$Rd = Rm * Rs$

Syntax: <instruction>{<cond>} {S} RdLo, RdHi, Rm, Rs

<b>SMLAL</b>	signed multiply accumulate long	$[RdHi, RdLo] = [RdHi, RdLo] + (Rm * Rs)$
<b>SMULL</b>	signed multiply long	$[RdHi, RdLo] = Rm * Rs$
<b>UMLAL</b>	unsigned multiply accumulate long	$[RdHi, RdLo] = [RdHi, RdLo] + (Rm * Rs)$
<b>UMULL</b>	unsigned multiply long	$[RdHi, RdLo] = Rm * Rs$



Edit with WPS Office

# 3.2 Branch Instructions

Syntax: B{<cond>} label

BL{<cond>} label

BX{<cond>} Rm

BLX{<cond>} label | Rm

B	branch	$pc = label$
BL	branch with link	$pc = label$ $lr = \text{address of the next instruction after the BL}$
BX	branch exchange	$pc = Rm \ \& \ 0xffffffff, T = Rm \ \& \ 1$
BLX	branch exchange with link	$pc = label, T = 1$ $pc = Rm \ \& \ 0xffffffff, T = Rm \ \& \ 1$ $lr = \text{address of the next instruction after the BLX}$



Edit with WPS Office

# 3.3 Load-Store Instructions (3.3.1 Single-Register Transfer)

LDRSTR

Word

LDRB

STRB

Byte

LDRH

STRH

Halfword

LDRSB

Signed byte load

LDRSH

Signed halfword

load

- Memory system must support all access sizes

- Syntax:

- **LDR{<cond>}{<size>}** Rd, <address>
  - **STR{<cond>}{<size>}** Rd, <address>

e.g. **LDREQB**



Edit with WPS Office

# 3.3 Load-Store Instructions (3.3.1 Single-Register Transfer)

Address accessed by LDR/STR is specified by a base register with an offset

- For word and unsigned byte accesses, offset can be:
  - An unsigned 12-bit immediate value (i.e. 0 - 4095 bytes)  
`LDR r0, [r1, #8]`
  - A register, optionally shifted by an immediate value  
`LDR r0, [r1, r2]`  
`LDR r0, [r1, r2, LSL#2]`
- This can be either added or subtracted from the base register:
  - `LDR r0, [r1, #-8]`
  - `LDR r0, [r1, -r2, LSL#2]`
- For halfword and signed halfword / byte, offset can be:
  - An unsigned 8 bit immediate value (i.e. 0 - 255 bytes)
  - A register (unshifted)
- Choice of *pre-indexed* or *post-indexed* addressing
- Choice of whether to update the base pointer (pre-indexed only)  
`LDR r0, [r1, #-8]!`



Edit with WPS Office

## 3.3.2 Single-Register Load-Store Addressing

- The ARM instruction set provides different modes for addressing memory.
- These modes incorporate one of the indexing methods:
  - Preindex with writeback,
  - Preindex
  - Postindex

Index methods.

Index method	Data	Base address register	Example
Preindex with writeback	$mem[base + offset]$	$base + offset$	LDR r0,[r1,#4]!
Preindex	$mem[base + offset]$	<i>not updated</i>	LDR r0,[r1,#4]
Postindex	$mem[base]$	$base + offset$	LDR r0,[r1],#4



Edit with WPS Office

# 3.3.2 Single-Register Load-Store Addressing

## Mod

Single-register load-store addressing, word or unsigned byte.

Addressing <sup>1</sup> mode and index method	Addressing <sup>1</sup> syntax
Preindex with immediate offset	[Rn, #+/-offset_12]
Preindex with register offset	[Rn, +/-Rm]
Preindex with scaled register offset	[Rn, +/-Rm, shift #shift_imm]
Preindex writeback with immediate offset	[Rn, #+/-offset_12]!
Preindex writeback with register offset	[Rn, +/-Rm]!
Preindex writeback with scaled register offset	[Rn, +/-Rm, shift #shift_imm]!
Immediate postindexed	[Rn], #+/-offset_12
Register postindex	[Rn], +/-Rm
Scaled register postindex	[Rn], +/-Rm, shift #shift_imm



## Multiple-Register Transfer

- Multiple-Register Transfer- Load Store instructions, can transfer data between multiple registers and memory in a single instruction.
- Efficient from single-register transfers for moving blocks of data, saving and restoring context and stacks.
- These instructions can increase interrupt latency. (Find out how)
- The transfer occurs from a base address register  $R_n$  pointing into memory.



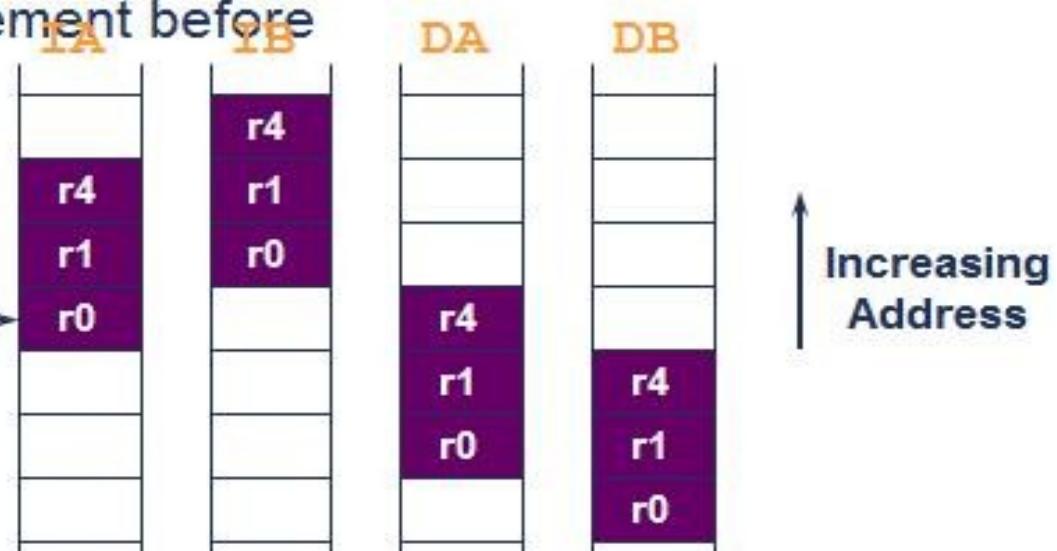
### 3.3.3 Multiple-Register

- Syntax:
  - **<LDM|STM>{<cond>}<addressing\_mode> Rb{!}, <register list>**
- 4 addressing modes:
  - **LDMIA / STMIA** increment after
  - **LDMIB / STMIB** increment before
  - **LDMDA / STMDA** decrement after
  - **LDMDB / STMDB** decrement before

**LDMxx r10, {r0,r1,r4}**

**STMxx r10, {r0,r1,r4}**

Base Register (Rb) **r10** →



### 3.3.3 Multiple-Register Transfer

Addressing mode for load-store multiple instructions.

Addressing mode	Description	Start address	End address	$Rn!$
IA	increment after	$Rn$	$Rn + 4^*N - 4$	$Rn + 4^*N$
IB	increment before	$Rn + 4$	$Rn + 4^*N$	$Rn + 4^*N$
DA	decrement after	$Rn - 4^*N + 4$	$Rn$	$Rn - 4^*N$
DB	decrement before	$Rn - 4^*N$	$Rn - 4$	$Rn - 4^*N$



## 3.3.3.1 Stack Operations

- The load-store multiple instructions to carry out stack operations.
  - The *pop* operation (removing data from a stack) uses a load multiple instruction
  - The *push* operation (placing data onto the stack) uses a store multiple instruction.
- A stack is either *ascending* (A) or *descending* (D).
  - Ascending stacks grow towards higher memory addresses
  - Descending stacks grow towards lower memory addresses.
  - *full stack* (F)
  - *empty stack* (F)



Edit with WPS Office

# 3.3.3.1 Stack Operations

Addressing methods for stack operations.

Addressing mode	Description	Pop	= LDM	Push	= STM
FA	full ascending	LDMFA	LDMDA	STMFA	STMIB
FD	full descending	LDMFD	LDMIA	STMF D	STMDB
EA	empty ascending	LDMEA	LDMDB	STMEA	STMIA
ED	empty descending	LDMED	LDMIB	STMED	STMDA



---

PRE	Address	Data	POST	Address	Data
	0x80018	0x00000001		0x80018	0x00000001
sp →	0x80014	0x00000002		0x80014	0x00000002
	0x80010	Empty		0x80010	0x00000003
	0x8000c	Empty	sp →	0x8000c	0x00000002

---

Figure 3.7 STMFD instruction—full stack push operation.

POST    r1 = 0x00000002  
        r4 = 0x00000003  
        sp = 0x0008000c



Edit with WPS Office

**EXAMPLE  
3.21**

In contrast, Figure 3.8 shows a push operation on an empty stack using the STMED instruction. The STMED instruction pushes the registers onto the stack but updates register *sp* to point to the next empty location.

**PRE**      *r1* = 0x00000002  
              *r4* = 0x00000003  
              *sp* = 0x00080010

STMED    *sp!*. {*r1,r4*}

**POST**     *r1* = 0x00000002  
              *r4* = 0x00000003  
              *sp* = 0x00080008

---

<b>PRE</b>	<b>Address</b>	<b>Data</b>	<b>POST</b>	<b>Address</b>	<b>Data</b>
<i>sp</i> →	0x80018	0x00000001	<i>sp</i> →	0x80018	0x00000001
	0x80014	0x00000002		0x80014	0x00000002
	0x80010	Empty		0x80010	0x00000003
	0x8000c	Empty		0x8000c	0x00000002
	0x80008	Empty		0x80008	Empty

---

Figure 3.8 STMED instruction—empty stack push operation.



Edit with WPS Office

### 3.3.4 Swap Instruction

Syntax: SWP{B} {<cond>} Rd, Rm, [Rn]

SWP	swap a word between memory and a register	$tmp = mem32[Rn]$ $mem32[Rn] = Rm$ $Rd = tmp$
SWPB	swap a byte between memory and a register	$tmp = mem8[Rn]$ $mem8[Rn] = Rm$ $Rd = tmp$



Edit with WPS Office

# 3.4 Software Interrupt Instruction

- Causes a software interrupt exception
  - which provides a mechanism for applications to call operating system routines.

Syntax: SWI{<cond>} SWI\_number

SWI	software interrupt	$lr\_svc$ = address of instruction following the SWI $spsr\_svc$ = $cpsr$ $pc$ = vectors + 0x8 $cpsr$ mode = SVC $cpsr I$ = 1 (mask IRQ interrupts)
-----	--------------------	---



Edit with WPS Office

## EXAMPLE

- Here we have a simple example of an SWI call with SWI number 0x123456, used by ARM toolkits as a debugging SWI. Typically the SWI instruction is executed in user mode.

**PRE**    cpsr = nzcvqift\_USER  
          pc = 0x00008000  
          lr = 0x003ffff; lr = r14  
          r0 = 0x12

0x00008000 SWI 0x123456

**POST**    cpsr = nzcvqift\_SVC  
          spsr = nzcvqift\_USER  
          pc = 0x00000008  
          lr = 0x00008004  
          r0 = 0x12

Exception	Mode	Vector address
Reset	SVC	0x00000000
Undefined instruction	UND	0x00000004
Software interrupt (SWI)	SVC	0x00000008
Prefetch abort (instruction fetch memory fault)	Abort	0x0000000C
Data abort (data access memory fault)	Abort	0x00000010
IRQ (normal interrupt)	IRQ	0x00000018
FIQ (fast interrupt)	FIQ	0x0000001C



Edit with WPS Office

# 3.5 Program Status Register

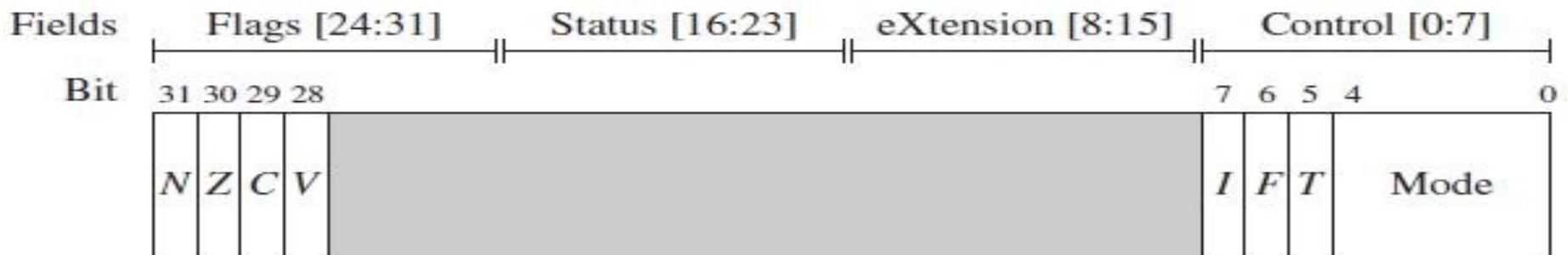
- Two instructions control a program status register (*psr*).
  - The MRS instruction transfers the contents of either the *cpsr* or *spsr* into a register
  - the MSR instruction transfers the contents of a register into the *cpsr* or *spsr*.



Edit with WPS Office

# 3.5 Program Status Register

Syntax: `MRS{<cond>} Rd,<cpsr|spsr>`  
`MSR{<cond>} <cpsr|spsr>_<fields>,Rm`  
`MSR{<cond>} <cpsr|spsr>_<fields>,#immediate`



MRS	copy program status register to a general-purpose register	$Rd = psr$
MSR	move a general-purpose register to a program status register	$psr[field] = Rm$
MSR	move an immediate value to a program status register	$psr[field] = immediate$



- Examples:

- Program to enable FIQ (executed in svc mode)

MRS	<code>r1, cpsr</code>	; copies CPSR into r1
BIC	<code>r1, #0x40</code>	; clears B6, i.e. FIQ interrupt mask bit
MSR	<code>cpsr, r1</code>	; copies r1 into CPSR



## 3.5.1 Coprocessor Instructions

- A coprocessor can either provide additional computation capability or be used to control the memory subsystem including caches and memory management.
- The coprocessor instructions include data processing, register transfer, and memory transfer instructions.



Edit with WPS Office

## 3.5.1 Coprocessor Instructions

Syntax: `CDP{<cond>} cp, opcode1, Cd, Cn {, opcode2}`  
`<MRC|MCR>{<cond>} cp, opcode1, Rd, Cn, Cm {, opcode2}`  
`<LDC|STC>{<cond>} cp, Cd, addressing`

CDP	coprocessor data processing—perform an operation in a coprocessor
MRC MCR	coprocessor register transfer—move data to/from coprocessor registers
LDC STC	coprocessor memory transfer—load and store blocks of memory to/from a coprocessor



# 3.6 Loading Constants

- There is no ARM instruction to move a 32-bit constant into a register.
- Since ARM instructions are 32 bits in size, they obviously cannot specify a general 32-bit constant.

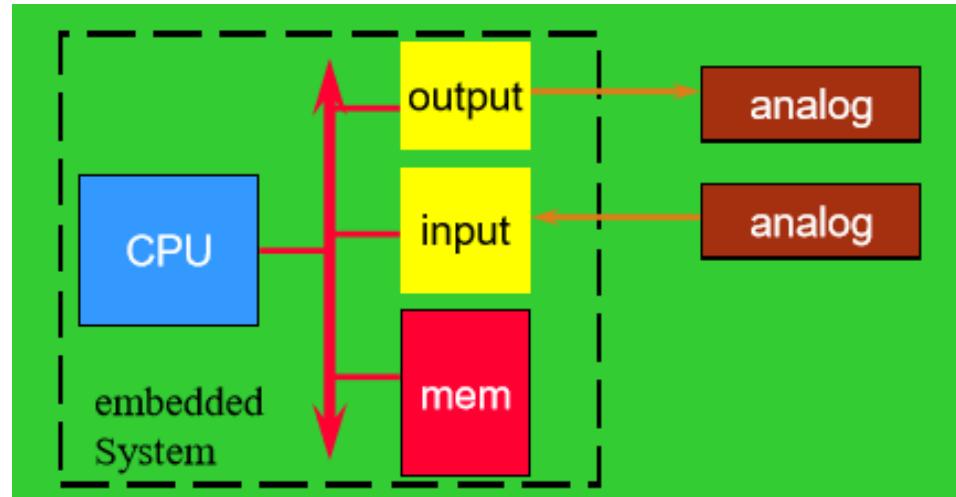
Syntax: LDR Rd, =constant  
ADR Rd, label

LDR	load constant pseudoinstruction	$Rd = 32\text{-bit constant}$
ADR	load address pseudoinstruction	$Rd = 32\text{-bit relative address}$



Edit with WPS Office

# What is an Embedded System



- An ES is an **electronic /electro-mechanical system** designed to perform a **specific function**
- And a combination of both **hardware** and **firmware (software)**
- Every ES is **Unique** and **hardware** as well as the **firmware** is highly specialized to the **application domain**

# Embedded System Vs General Purpose Computing System

Based on	General Purpose Computing System	Embedded System
Contents	A system which is a combination of a generic hardware and a <b>General Purpose Operating System</b> for executing a <b>variety</b> of applications.	A system which is a combination of a <b>special purpose hardware</b> and embedded OS/firmware for executing a <b>specific</b> set of applications
OS	General purpose operating system ( <b>GPOS</b> ).	It <b>may or not contain</b> an operating system for functioning.
Alterations	<b>Applications</b> are alterable by the user.	Applications are not-alterable by the user.

# Embedded System Vs General Purpose Computing System

Based on	General Purpose Computing System	Embedded System
Key factor	<b>Performance</b> is key factor.	Application specific <b>requirements</b> are key factors.
Power Consumption	<b>More</b>	<b>Less</b>
Response Time	<b>Not</b> critical	<b>Critical</b> for some applications
Execution	Need <b>not</b> be deterministic	Deterministic for certain types of ES like ' <b>Hard Real Time</b> ' systems.



Edit with WPS Office

# History of Embedded System

- ES exists before-IT revolution
  - i.e. it was used in old vacuum tubes, transistors
  - And : Embedded algorithms are developed in low level language.
- First ES is:
  - Apollo Guidance computer(AGC):developed by MIT Instrumentation Laboratory
    - For lunar expedition.
  - Original design : ROM(4K words) & RAM(256 words)
  - In June 1963: ROM(10K words) & RAM(1K words)
  - Final configuration: ROM(36K words) & RAM(2K words)

Contd..





OLD VACUUM TUBES

APOLLO GUIDANCE COMPUTER



Edit with WPS Office

Contd...

## History of Embedded System

- First modern ES is AGC :(Contd..)
  - Clock frequency in microchip: 1.024MHz-derived from 2.048 crystal clock.
  - Computing unit: 11 instructions with 16 bit word logic
  - 5000 ICs used: with 3-input NOR gates , RTL logic
  - User interface unit : uses DS/KY(display/keyboard)
- First modern mass produced ES:
  - For Minuteman-I missile in 1961:-Autonetics D-17
    - With discrete transistor logic and hard disk for main memory
- First integrated circuit in 1963:
  - 0 5/13/2022



Edit with WPS Office



Edit with WPS Office

# Classification of Embedded System

- Classification of ES is based upon following criteria:
  1. Based on generation
  2. Complexity and performance requirements
  3. Based on deterministic behaviour
    - Applicable to Real time systems
    - Execution behaviour can be deterministic/undeterministic
  4. Based on triggering
    - Systems which are reactive are trigger based.
    - Reactive systems are event/time triggered. **Studied in later chapters**

# Classification of Embedded System

- Based on Generation:
  - First Generation:
    - 8bit – $\mu$ P like 8085, 4bit - $\mu$ C
    - Circuits uses Assembly code
    - E.g.: Stepper motor control unit, digital telephone keypads
  - Second Generation:
    - 16bit – $\mu$ P, 8 or 16bit - $\mu$ C
    - Instruction set is much powerful
    - Few ES were using Embedded OS.
    - E.g.: Data Acquisition System

# Classification of Embedded System

- **Based on Generation:**

- Third Generation:

- 32bit – $\mu$ P , 16bit - $\mu$ C
    - DSPs(Digital Signal Processors) and ASICs (Application Specific Integrated Circuits) came into existence
    - Instruction set became much powerful : with pipelining concept
    - OS:RTOS and GPOS were used
    - E.g.: Intel Pentium, Motorola 68K

- Fourth Generation:

- SoC(System on Chips), reconfigurable processors, multicore processor came into existence.
    - High performance ICs are used

- OS:RTOS



Edit with WPS Office

- E.g.: Smart phones, Mobile internet devices(MIDs)

# Classification of Embedded System

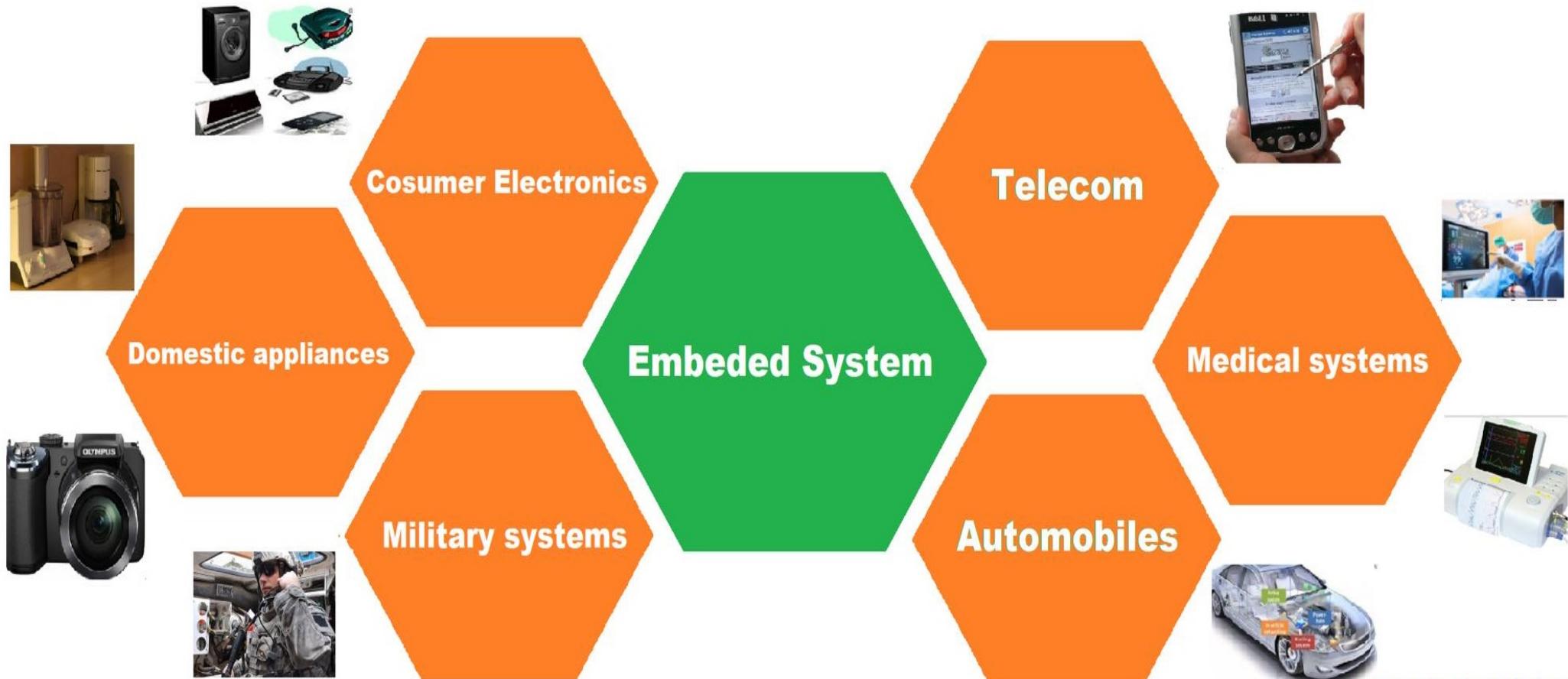
- Based on Complexity and Performance:
  - Small Scale ES:
    - 8 or 16 bit – $\mu$ P/ $\mu$ C
    - Systems are built around: Low performance and low cost
    - Applications: simple and Performance :not critical
    - E.g.: Electronic toys
  - Medium Scale ES:
    - 16 or 32 bit – $\mu$ P/ $\mu$ C
    - Systems are built around: medium performance and low cost
    - Slightly complex in hardware and software
    - OS:RTOS and GPOS were used
    - E.g.: DSPs



# Classification of Embedded System

- **Based on Complexity and Performance:**
  - Large Scale ES/ complex systems:
    - 32 or 64 bit –RISC processors/controllers
    - (Or) Reconfigurable SoC(RSoC)
    - (Or) Multi core processors and Programmable logic devices.
    - Consists of Co-units for processing the requirements from main processors
    - OS:RTOS is used with task scheduling , prioritisation
    - E.g.: Decoding and encoding of media, cryptographic function

## Real Life Examples of Embedded



# Major Applications of ES

The **application areas** and the products in the embedded domain are countless. A few of the important domains and products are listed below:

- i. Consumer electronics: Camcorders, cameras, etc. 
- ii. Household appliances: Television, DVD players, washing machine, fridge, microwave oven, etc. 
- iii. Home automation and security systems: Air conditioners, sprinklers, intruder detection alarms, closed circuit television cameras, fire alarms, etc. 
- iv. Automotive industry: Anti-lock breaking systems (ABS), engine control, ignition systems, automatic navigation systems, etc. 
- v. Telecom: Cellular telephones, telephone switches, handset multimedia applications, etc. 



Contd..

# Major Applications of ES



vi. Computer peripherals: Printers, scanners, fax machines, etc.



vii. Computer Networking systems: Network routers, switches, hubs, firewalls, etc.



viii. Healthcare: Different kinds of scanners, EEG, ECG machines etc.



ix. Measurement & Instrumentation: Digital multi meters, digital CROs, logic analyzers PLC systems, etc.



x. Banking & Retail: Automatic teller machines (ATM) and currency counters, point of sales (POS).



xi. Card Readers: Barcode, smart card readers, hand held devices, etc.



Edit with WPS Office

# Purpose of Embedded System

Each embedded system is designed to serve the purpose of any one or a combination of the following tasks:

1. Data collection/Storage/Representation
2. Data Communication
3. Data (signal) processing
4. Monitoring
5. Control
6. Application specific user interface



Edit with WPS Office

Contd..



# Purpose of Embedded System

## 1. Data

### collection/Storage/Representation:

- ES is designed for one of the purpose:
  - Data collection from external world.
  - Data collection is done mainly for:
    - Storage
    - Analysis
    - Manipulation
    - Manipulation
  - Transmission
  - Examples:
    - Videos, images, text, electrical signals etc.
    - Videos, images, text, electrical signals etc.



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt



# Purpose of Embedded System

## 1. Data collection/Storage/Representation:

- Data can be either :
  - Analog (Continuous)
  - Digital(Discrete)
- If data collected :
  - Digitally :- then no conversion is required as the data is in binary representation.
  - Analog:-then we need to convert analog to digital signal by A/D converters and later use digital data.
- Best Example device :
  - Digital camera
  - Digital camera



Edit with WPS Office



# Purpose of Embedded System

## 1. Data collection/Storage/Representation

- Digital camera for:
- Digital camera for:
  - Image capturing
  - Storage: captured image stored in camera's memory
  - Image can be given to the user:
    - Through graphic LCD unit





# Purpose of Embedded System

## 2. Data communication:

- Communication means :
  - The way data is transmitted / received
  - It means communication can be from:
  - Complex satellite communication systems -> **simple home networking systems.**
- The transmission medium can be:
  - Wired
  - Wireless

**Note**

Data can be transmitted either by analog / digital / digital



Edit with WPS Office



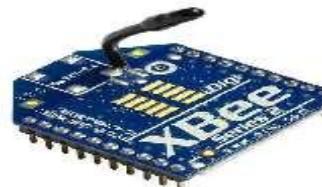
# Purpose of Embedded System

## 2. Data communication:

- The data collecting embedded terminal itself can incorporate data communication units like:

data communication units like:

- Wireless modules (Bluetooth, ZigBee, WiFi, EDGE, GPRS, etc.)



- Wire-line modules (RS-232C, USB, TCP/IP, PS2, etc).



Edit with WPS Office

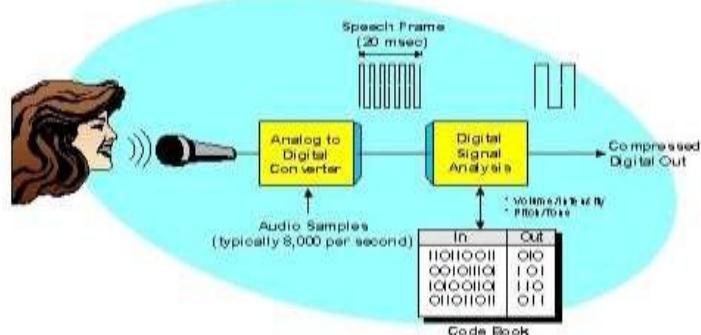


# Purpose of Embedded System

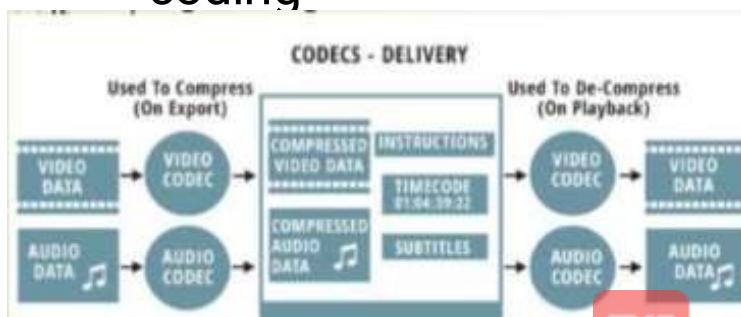
## 3. Data (Signal) Processing:

Data of any type collected can be used for :

- for :
- Data processing:- like

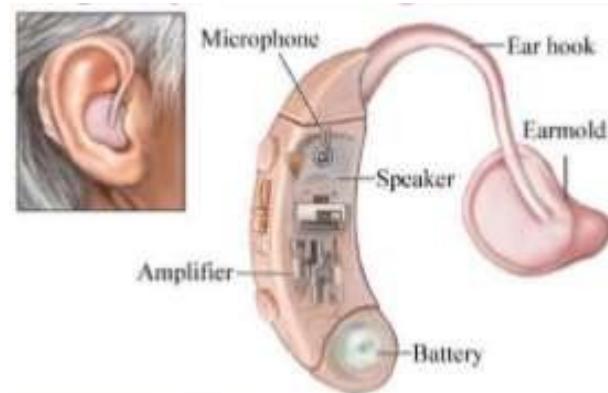


- Audio video coding



### Best Example:

Digital hearing aid(improves the hearing capacity by data processing)





# Purpose of Embedded System

## 4. Monitoring:

- Almost all embedded products coming under the medical domain are with monitoring functions only.
- Electro cardiogram machine (ECG) is intended to do the monitoring of the heartbeat of a patient but it cannot impose control over the heartbeat.
- Electro cardiogram machine (ECG) is intended to do the monitoring of the heartbeat of a patient but it cannot impose control over the heartbeat.
- Other examples with monitoring function are digital CRO, digital multimeters, and logic analyzers.





# Purpose of Embedded System

## 5. Control:

- A system with control functionality contains both sensors and actuators.
- Sensors are connected to the input port for capturing the changes in measuring variable.
- Sensors are connected to the input port for capturing the changes in measuring variable.
- The actuators connected to the output port are controlled according to the changes in the input variable.
- Example: Air conditioner system:
  - The actuators connected to the output port are controlled according to the changes in the input variable.
- Example: Air conditioner system:
  - In our home used to control the room temperature to a specified limit



# Purpose of Embedded System

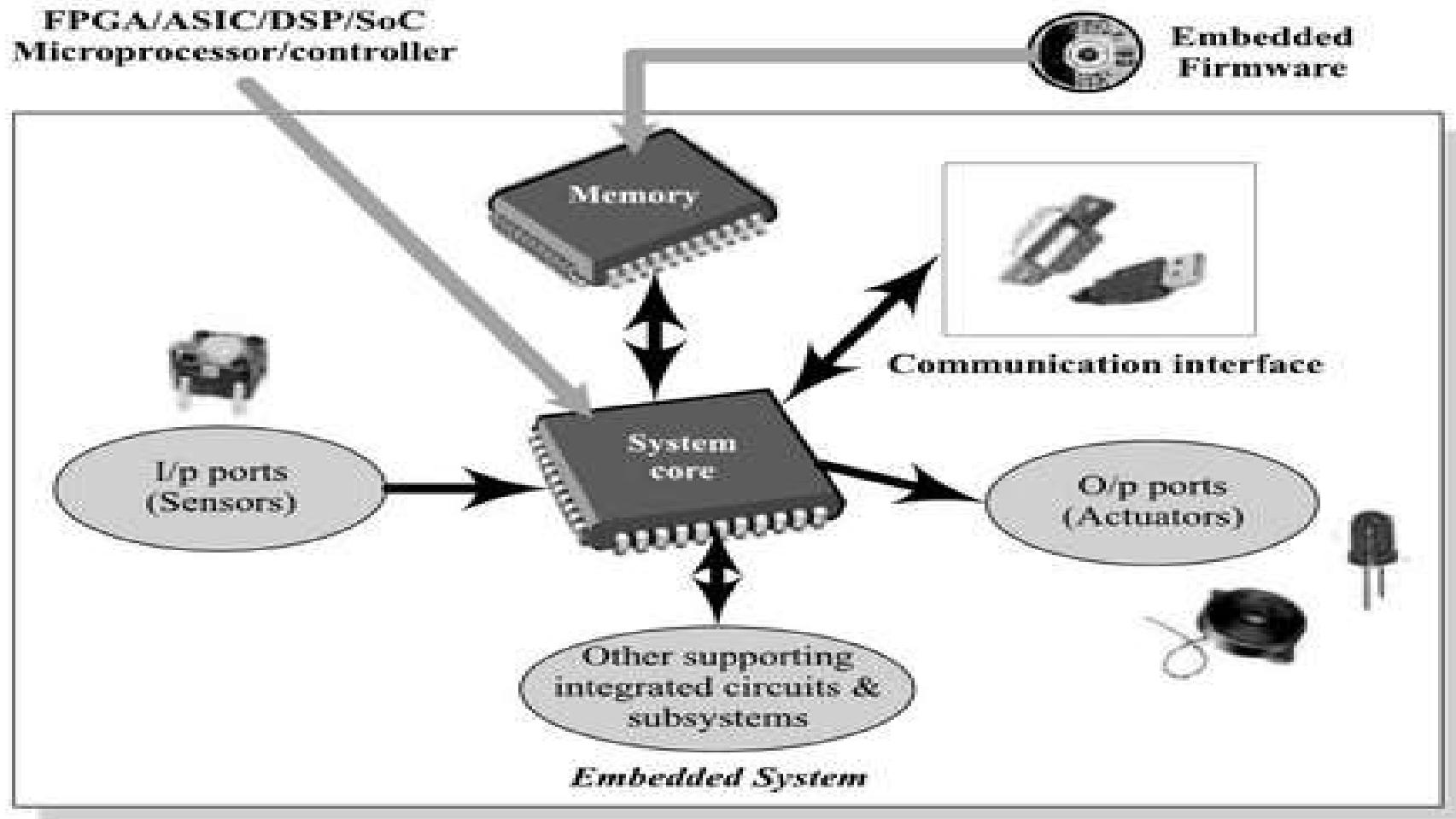
## 6. Application Specific user interface:

- Buttons, switches, keypad, lights, speakers, display units etc. are application-specific user interfaces.
- are application-specific user interfaces.
- Example: Mobile phone -> the user interface is provided through the keypad, graphic LCD module, system speaker, vibration alert, etc.
- Example: Mobile phone -> the user interface is provided through the keypad, graphic LCD module, system speaker, vibration alert, etc.



# Chapter-2

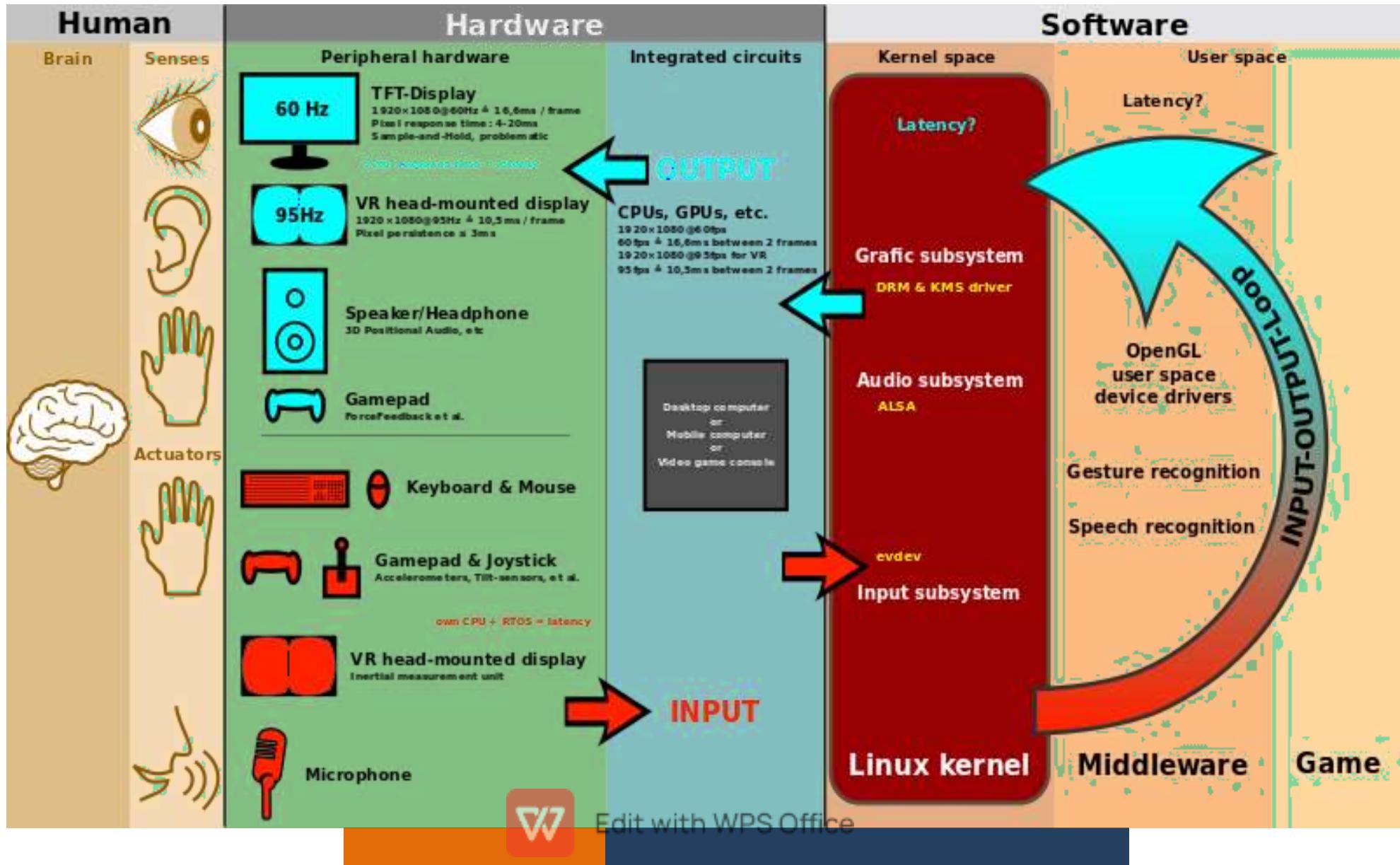
## The Typical Embedded System



Edit with WPS Office

# Chapter-2

## The Typical Embedded System



# Core of Embedded System

The core of the embedded system falls into any of the following categories:

1. General Purpose and Domain Specific Processors
  - Microprocessors
  - Microcontrollers
  - Digital Signal Processors
2. Application Specific Integrated Circuit(ASICs)
3. Programmable Logic devices(PLDs)
4. Commercial off-the-shelf Components(COTS)



Edit with WPS Office

# Core of Embedded System

## 1. General Purpose and Domain Specific Processors

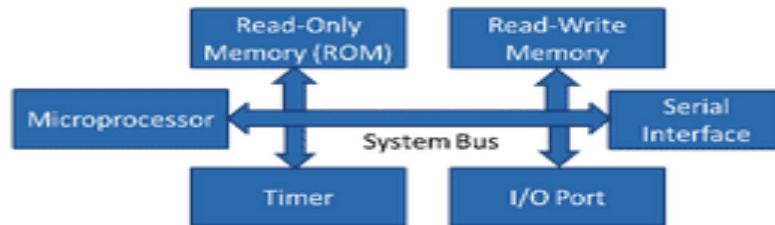
- Microprocessors
- Microcontrollers
- Digital Signal Processors etc.

As we already know in detail of microprocessor and microcontroller-we will take an overlook of their difference

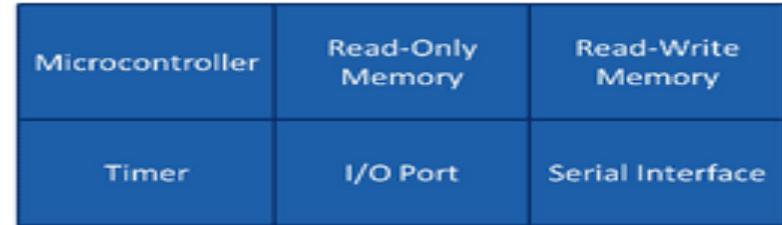


Edit with WPS Office

## Microprocessor



## Micro Controller



Microprocessor is heart of Computer system.

It is just a processor. Memory and I/O components have to be connected externally

Since memory and I/O has to be connected externally, the circuit becomes large.

Cannot be used in compact systems and hence inefficient

Cost of the entire system increases

Due to external components, the entire power consumption is high. Hence it is not suitable to use with devices running on stored power like batteries.

Most of the microprocessors do not have power saving features.

Since memory and I/O components are all external, each instruction will need external operation, hence it is relatively slower.

Microprocessor have less number of registers, hence more operations are memory based.

Microprocessors are based on von Neumann model/architecture where program and data are stored in same memory module

Mainly used in personal computers

Micro Controller is a heart of embedded system.

Micro controller has external processor along with internal memory and i/O components

Since memory and I/O are present internally, the circuit is small.

Can be used in compact systems and hence it is an efficient technique

Cost of the entire system is low

Since external components are low, total power consumption is less and can be used with devices running on stored power like batteries.

Most of the micro controllers have power saving modes like idle mode and power saving mode. This helps to reduce power consumption even further.

Since components are internal, most of the operations are internal instruction, hence speed is fast.

Micro controller have more number of registers, hence the programs are easier to write.

Micro controllers are based on Harvard architecture where program memory and Data memory are separate



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

Used mainly in washing machine, MP3 players

# Core of Embedded System:-General Purpose and Domain Specific Processors

Contd..

## 1.2 Digital Signal Processors:-

- DSPs are powerful special purpose:
  - 8/16/32 bit microprocessors designed
  - To meet the computational demands and power constraints
  - Of today's embedded audio, video, and communications applications.
- Digital signal processors are 2 to 3 times faster than the general purpose microprocessors in signal processing



# Core of Embedded System:-General Purpose and Domain Specific Processors

## 2. Digital Signal Processors: Contd..

A typical digital signal processor incorporates the following key units:

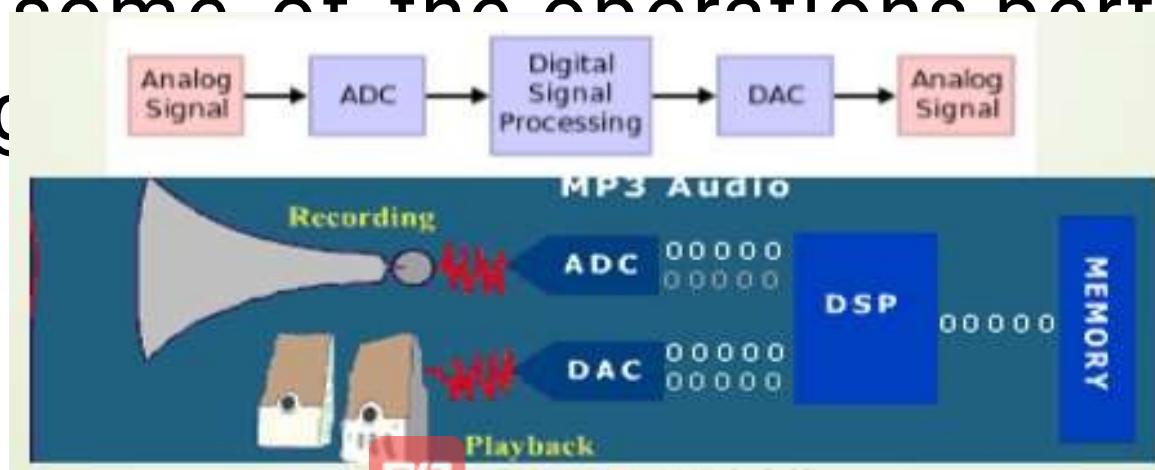
- i. **Program Memory** : Memory for **storing** the program required by DSP to process the data.
- ii. **Data Memory** : **Working** variables/information and data/signal to be processed for memory
- iii. **Computational Engine** : Performs the **signal/math** processing, **storing** **temporary** accessing the program from the Program Memory and the data from the Data Memory.
- iv. **I/O Unit** :Acts as an **interface** between the outside world and **DSP**. It is **responsible** for **capturing** **signals** to be processed and **delivering** the processed signals.

# Core of Embedded System:-General Purpose and Domain Specific Processors

## 1.2 Digital Signal Processors:<sup>Contd..</sup>

Application areas : Audio video signal processing, telecommunication and multimedia applications.

DSP employs a large amount of real-time calculations, Sum of products (SOP) calculation, convolution, fast Fourier transform (FFT), discrete Fourier transform (DFT), etc. are some of the operations performed by digital signal processor.



# Core of Embedded System:-General Purpose and Domain Specific Processors



## 1.3 RISC vs CISC Processors/Controllers: Contd

As we already know in detail of RISC vs CISC architecture -we will take an overlook of their difference



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

# Core of Embedded System:-General Purpose and Domain Specific Processors

RISC	CISC
<b>Lesser number of instructions</b>	<b>Greater number of Instructions</b>
<b>Instruction pipelining</b> and increased execution speed	Generally <b>no instruction pipelining feature</b>
<b>Orthogonal instruction set</b>	<b>Non-orthogonal instruction set</b>
Operations are performed <b>on registers only</b> , the only memory operations are load and store.	Operations are performed <b>on registers or memory</b> depending on the instruction.
<b>A large number of registers</b> are available.	<b>Limited</b> number of general purpose registers.
Programmer <b>needs to write more code to execute a task</b> since the <b>instructions are simpler ones</b> .	Instructions are like macros in <b>C language</b> . A programmer can achieve the desired functionality with a single instruction which in turn provides the effect of using <b>more</b>



# Core of Embedded System:-General Purpose and Domain Specific Processors

RISC	CISC
<b>Single, fixed length instructions</b>	<b>Variable length instructions</b>
<b>Less</b> silicon usage and pin count	<b>More</b> silicon usage since more additional decoder logic is required to implement the complex instruction decoding.
<b>With Harvard Architecture</b>	Can be <b>Harvard</b> or <b>Von-Neumann Architecture</b>



# Core of Embedded System:-General Purpose and Domain Specific Processors

## 1.4 Big-Endian vs. Little-Endian Processors/ Controllers: Contd.

Endianness specifies the order:

- In which a sequence of bytes are stored in computer memory.

Little-endian is an order in which the “little end”/ the lower-order byte of the data (least significant value in the sequence) is stored in memory at the lowest address. (The little end comes first.)

For example, a 4 byte long integer Byte3, Byte2, Byte1, Byte0 will be stored in the memory as shown below:

# Little-Endian Processors

Base Address+0	Byte 0	Byte 0	0x20000 (Base Address)
Base Address+1	Byte 1	Byte 1	0x20001 (Base Address+1)
Base Address+2	Byte 2	Byte 2	0x20002 (Base Address+2)
Base Address+3	Byte 3	Byte 3	0x20003 (Base Address+3)

Example : **90AB12CD** (Hexadecimal)

Address	Value
1000	CD
1001	12
1002	AB
1003	90



Edit with WPS Office

# Big-Endian Processors

## Processors

- Big-endian is an order in which the “big end” / the higher-order byte of the data (most significant value in sequence) is stored in memory at the lowest address. (The big end comes first.)
- For example, a 4 byte long integer **Byte3, Byte2, Byte1, Byte0** will be stored in the memory as shown below:





# Big-Endian Processors

## Diagram

Base Address+0

Byte 3

Byte 3

0x20000 (Base Address)

Base Address+1

Byte 2

Byte 2

0x20001 (Base Address+1)

Base Address+2

Byte 1

Byte 1

0x20002 (Base Address+2)

Base Address+3

Byte 0

Byte 0

0x20003 (Base Address+3)

Example : **90AB12CD** (Hexadecimal)

Address	Value
1000	90
1001	AB
1002	12
1003	CD



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

# Core of Embedded System:-General Purpose and Domain Specific Processors

Contd..

## 1.5 Harvard vs. Von-neumann Proces

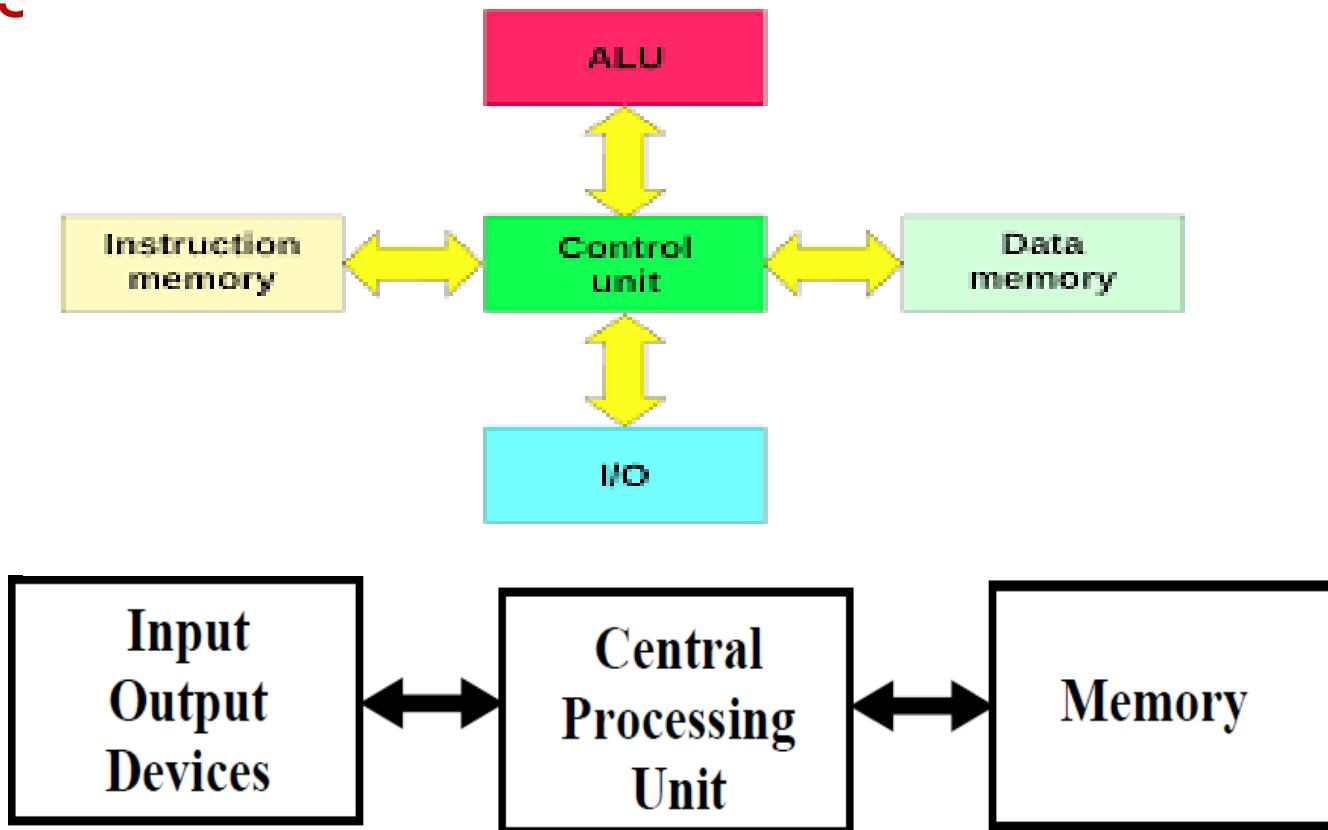


Fig. 5.1 The Von Neumann Architecture

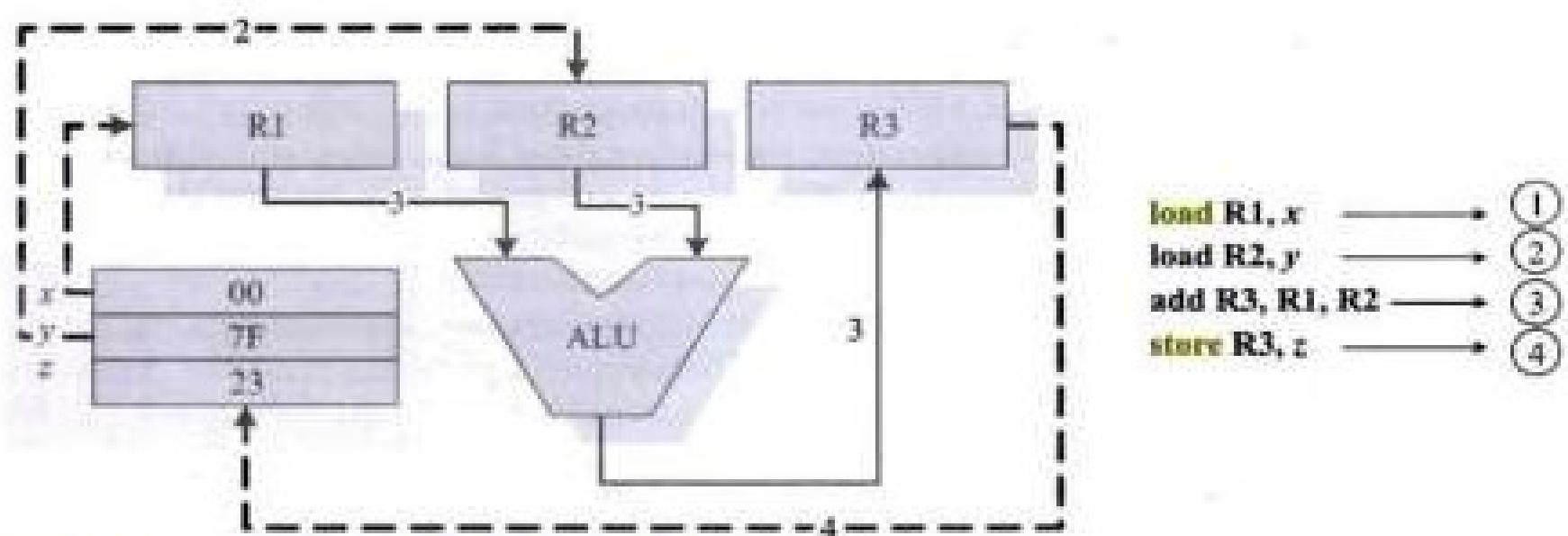


Edit with WPS Office

# Core of Embedded System:-General Purpose and Domain Specific Processors

Contd..

## 1.6 Load/store operation and Instruction



**Fig. 2.5** The concept of load-store architecture

# Core of Embedded System:-General Purpose and Domain Specific Processors

Contd..

## 1.6 Load/store operation and Instruction

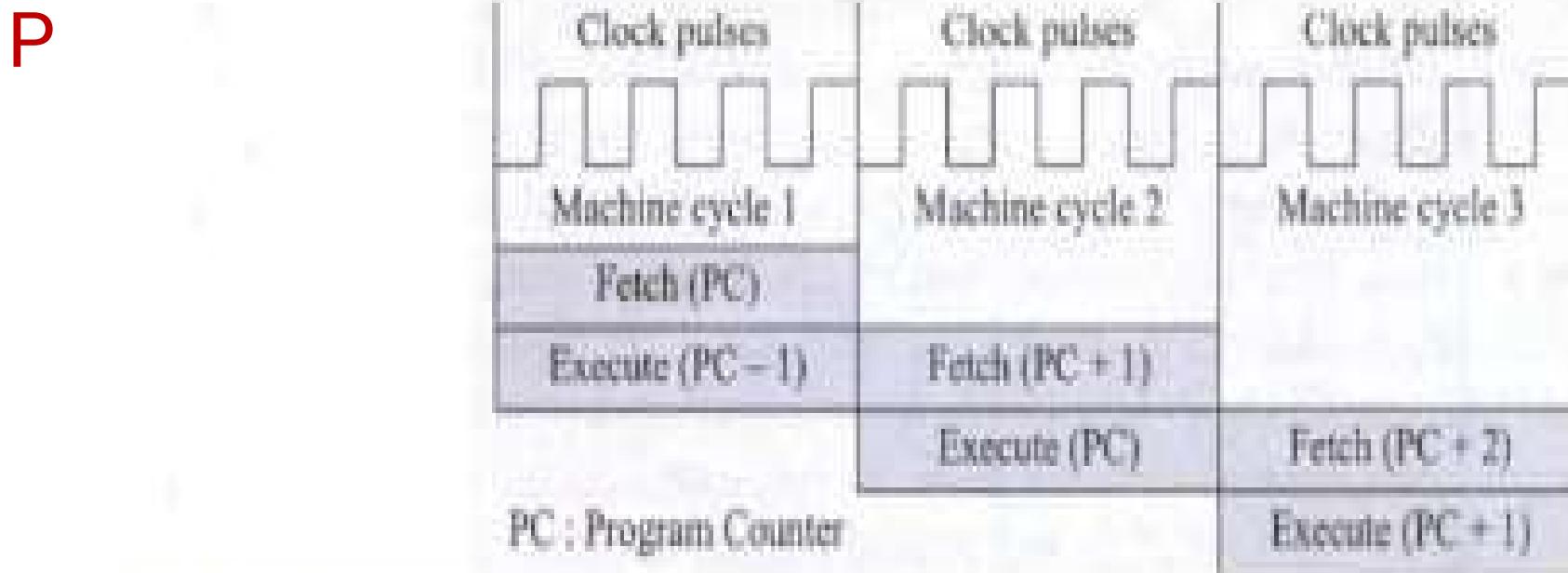


Fig. 2.6 The single-stage pipelining concept



Edit with WPS Office

## Core of Embedded System:-

Application Specific Integrated Circuit(ASICs) Contd..

ASICs :

- It is a Microchip designed to perform a specific or unique application.
- It integrates several functions on a single chip
  - Which reduces development cost.
  - ASICs consumes less space hence helps in the design of smaller system with high functionalities.
- ASICs can be :
  - Pre-fabricated for a specific application (OR)
  - Custom fabricated : by re-usable 'building blocks'
    - >library

## Core of Embedded System:-

Application Specific Integrated Circuit(ASICs) Contd..

ASICs :

Contd..

- This system are prof table :
  - For large commercial applications
  - Initial product requires:
    - Process technology
    - Configuration Investment
  - This initial investment is called NRE.
    - NRE: Non-Recurring Engineering charge
    - It is one time investment.
    - NRE: will be bared by Third party
  - ASICs will be sold like general

## Core of Embedded System:-

### Programmable Logic Devices(PLDs) Contd...

Contd..

PLDs :

- Logic devices can perform many specific applications like:
  - Device to device interfacing, data communication, signal processing, data display etc.
- Logic devices are classified as
  - Fixed → perform **defined** function once manufactured
  - Programmable → **Reconfigured** to perform any number of functions at any time.



Edit with WPS Office

## Core of Embedded System:-

### Programmable Logic Devices(PLDs) Contd..

#### PLDs :

- We are dealing with Programmable:
  - Designers will use → **software tools** to develop → stimulate → **test** their design
  - Design will be **quickly programmed** , tested in **lived circuit**.
  - It can be **prototype** so that can be used in other equipment's.
  - There is **no NRE**
  - Once **final** design is ready any number can be moved to market.



Edit with WPS Office

# Core of Embedded System:-

## Programmable Logic Devices(PLDs) Contd..

### PLDs :

- Two types:
  - Fixed Programmable Gate Arrays(FPGAs)
    - Highest amount of **Logic density**
    - Advanced devices even provide: Clock management, hardwired processors, device –device signal processing
    - **Applications:** Instrumentation, telecommunication, signal processing
  - Complex Programmable Logic Devices(CPLDs)
    - **Small** amount of **logic**
    - Offers very **predictable timing characteristics** → hence used in **critical control applications**
    - **Applications:** Mobile phones, Digital handheld assistants

# Core of Embedded System:-

## Programmable Logic Devices(PLDs) Contd..



### Advantages PLDs :

- Offer **flexibility** for customers.
- PLDs **don't require large** production **time**
- Customer need **not** have to pay **large NRE** hence the product will not be expensive.
- Allowing customers to order just **how many number** they need.
- They can be **reprogrammed** even a piece of equipment is delivered → even upgrading when equipment required.

# Core of Embedded System:-

## Commercial Off-the-Shelf components(COTS) Contd..

**COTS:**

**Contd..**

- It is used like 'as-is'
  - That means the seller is selling and the buyer is buying the product → in whatever condition it is.
- To provide :
  - Easy integration
  - Interoperability with existing components
- Example: Remote controlled cars





## 2.2

# Memory

- Memory is an important part of a  
– Processor / controller → embedded systems.
- Depending on the existence of the memory in  
processors/controllers it is referred as:  
– on-chip memory  
processors/controllers it is referred as:
  - Memory inside the chip
- on-chip memory
- off-chip memory
  - Memory inside the chip
  - External memory o be connected to  
Processor/controller
- off-chip memory
  - External memory o  
be connected



Edit with WPS Office

Department of ISE

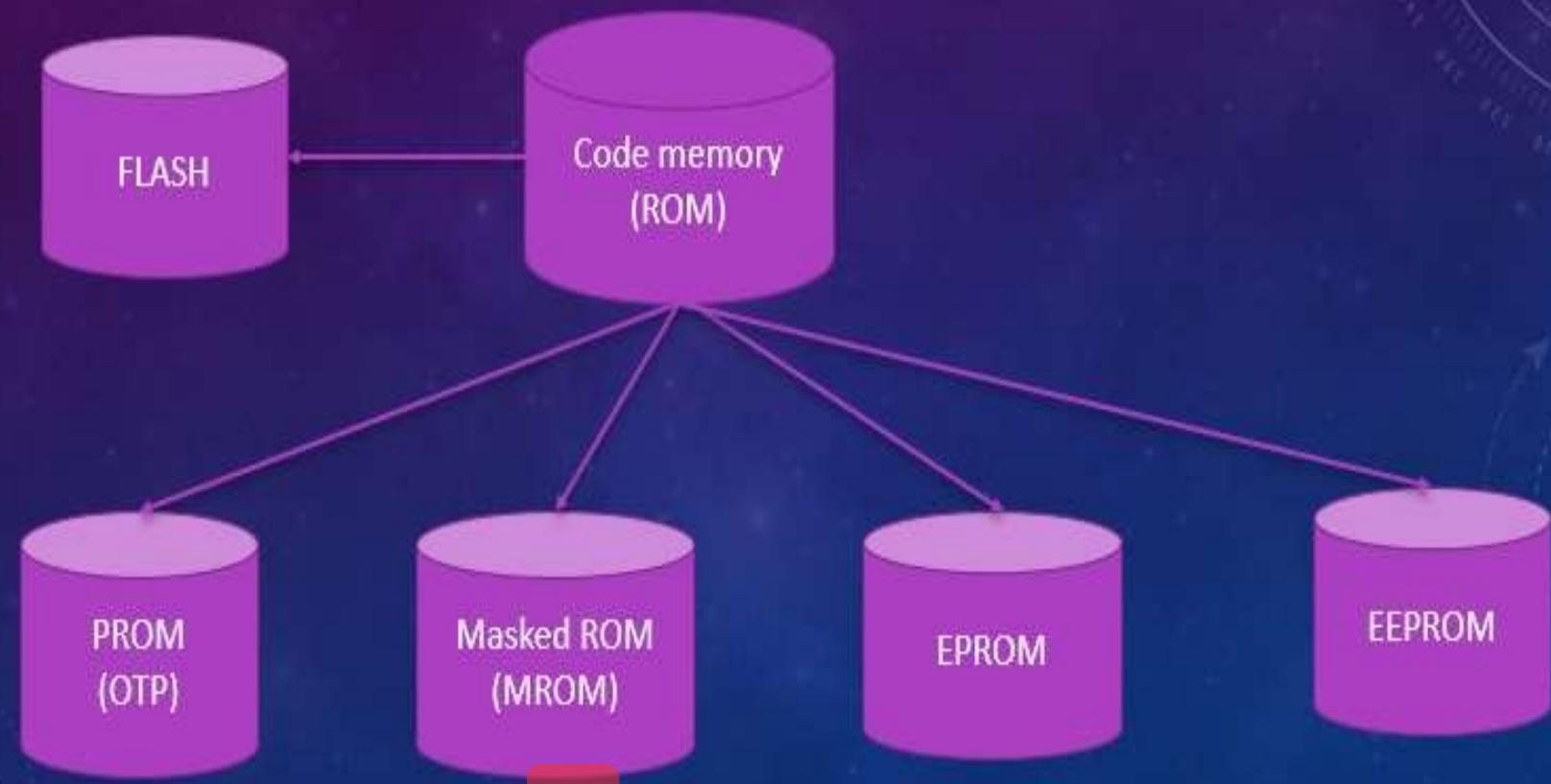
BMS Institute of Technology and Mgmt

to

# Types of Memory Non-



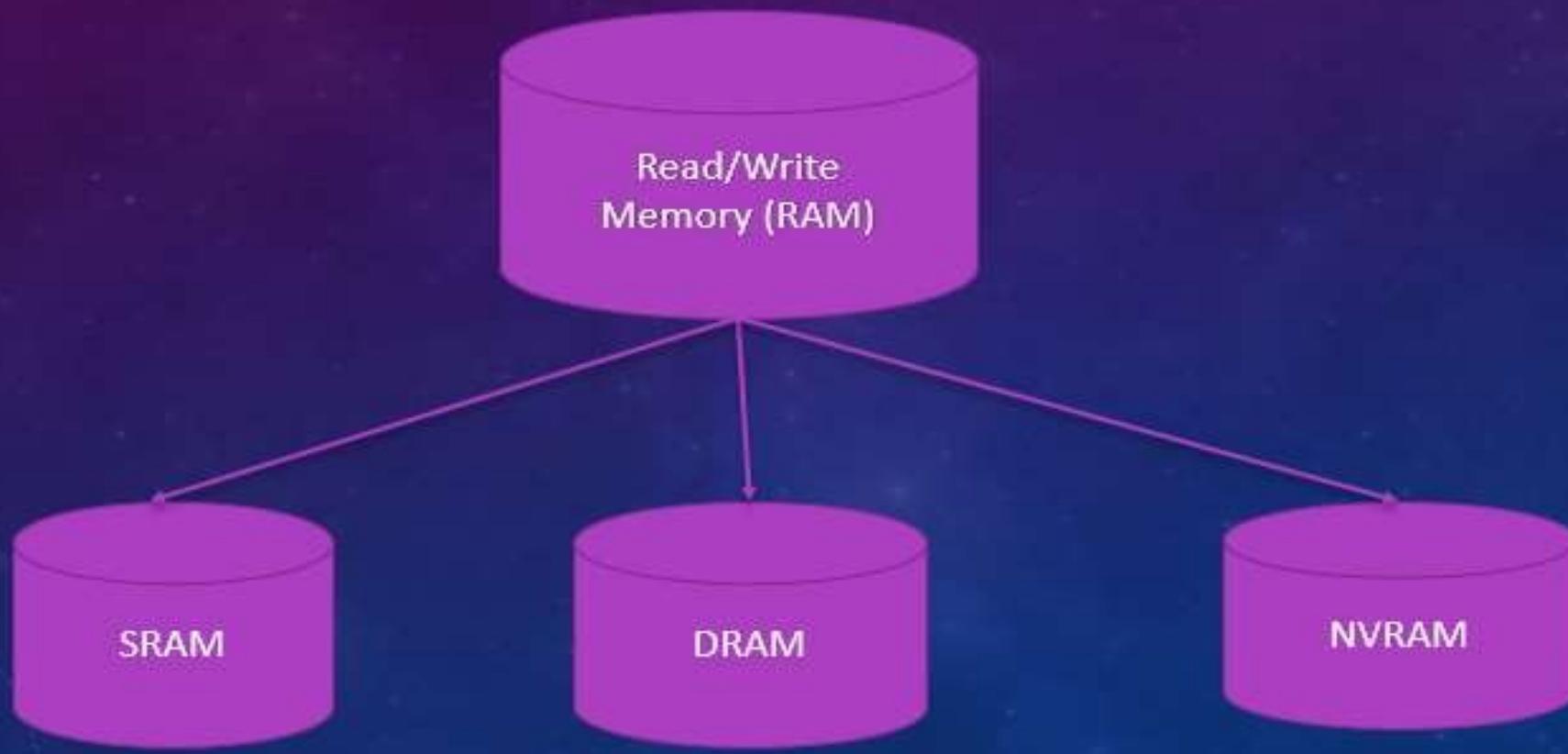
## CLASSIFICATION OF PROGRAM STORAGE MEMORY (ROM)



# Types of Memory



## CLASSIFICATION OF WORKING MEMORY (RAM)



## 2.2



### Types of Memory in Embedded System: **Memory**

#### 1. Program Storage Memory (ROM)

- a) Masked ROM (**MROM**)
- b) Programmable Read Only Memory (**PROM**)/ (**OTP**)
- c) Erasable Programmable Read Only Memory (**EPROM**)
- d) Electrically Erasable Programmable Read Only Memory (**EEPROM**)
- e) **FLASH**

#### 2. Read-Write Memory/Random Access Memory (RAM)

- a) Static RAM (**SRAM**)
- b) Dynamic RAM (**DRAM**)
- c) Non-volatile RAM(**NVRAM**)



## 2.2

# Program Storage Memory (ROM)

## Memory

### a) Masked ROM (MROM)

- Is a static ROM
- Programmed into an IC by its manufacturer.
- Uses of the hardwired technology for storing data.
- Used for low cost embedded devices.
- Advantage: low cost for high volume production.
- The limitation: Inability to modify the device against upgrades.
- Used in: network operating systems, server operating systems, storing of fonts for laser printers, sound data in electronic musical instruments.
- Used in: network operating systems, server operating systems, storing of fonts for laser printers, sound data in electronic musical instruments.

## 2.2

# Program Storage Memory (ROM) **Memory**

## b) Programmable Read Only Memory (PROM)/ (OTP)



- It is One Time Programmable Memory (OTP) or PROM is not pre-programmed by the manufacturer.
- The end user is responsible for programming these devices.
- Applications: cell phones, video game consoles, medical devices, and other electronics.
- Applications: cell phones, video game consoles, medical devices, and other electronics.

## 2.2



### Program Storage Memory (ROM)

### c) Erasable Programmable Read Only Memory (EPROM)

• Flexibility to re-program the same chip.

• **Flexibility to re-program the same chip.**

• **EPROM stores the bit information**

• **EPROM stores the bit information**

– By charging the floating gate of an FET

– By charging the floating gate of an FET

– Contains a quartz crystal window for erasing the stored information.

– Contains a quartz crystal

window

• In order to erase the data:

for erasing

– It needs to be taken out of the circuit board for erasing the stored

– Put in a UV eraser device for 20 to 30 minutes.

• In order to erase the data: So it is a tedious and time consuming process.

– It needs to be taken out of the circuit board

– Put in a UV eraser device for 20 to 30 minutes.

## 2.2

# Program Storage Memory (ROM)



## Memory d) Electrically Erasable Programmable Read Only Memory (EEPROM)

- The information can be altered by using :
  - Electrical signal at the register/Byte level.
- They can be erased and reprogrammed in-circuit.
  - These chips include a chip erase mode
  - In this mode they can be erased in a few milliseconds.
  - It provides greater flexibility for system design.
- It is used for storing the computer system BIOS.
- Only limitation:
  - Capacity is limited → compared with the standard ROM (a few kilobytes).
  - standard  
ROM



Edit with WPS Office



## 2.2

# Program Storage Memory (ROM)

## e) FLASH : **Memory**

- It is an enhanced version of EEPROM, It combines
  - Re-programmability of EEPROM
  - High capacity of standard ROMs.
- FLASH memory is organized as:
  - Sectors (blocks) or pages.
  - FLASH memory stores information in an array of floating gate MOS-FET transistors.
- Erasing of memory:
  - Can be done at sector/page level → without affecting the other sectors or pages.
    - Each sector/ page should be erased before re-programming.
    - Each sector/ page should be erased before re-programming.



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt



## 2.2

# Memory

### e) FLASH :

- The typical erasable capacity of FLASH is 1000 cycles.
- Many modern PCs have their BIOS stored on a flash memory chip, called as flash BIOS
- Many modern PCs have their BIOS stored on a flash memory chip, called as flash BIOS
- Also used: memory cards, USB flash drives, modems as well.
- Also used: memory cards, USB flash drives, modems as well.



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt



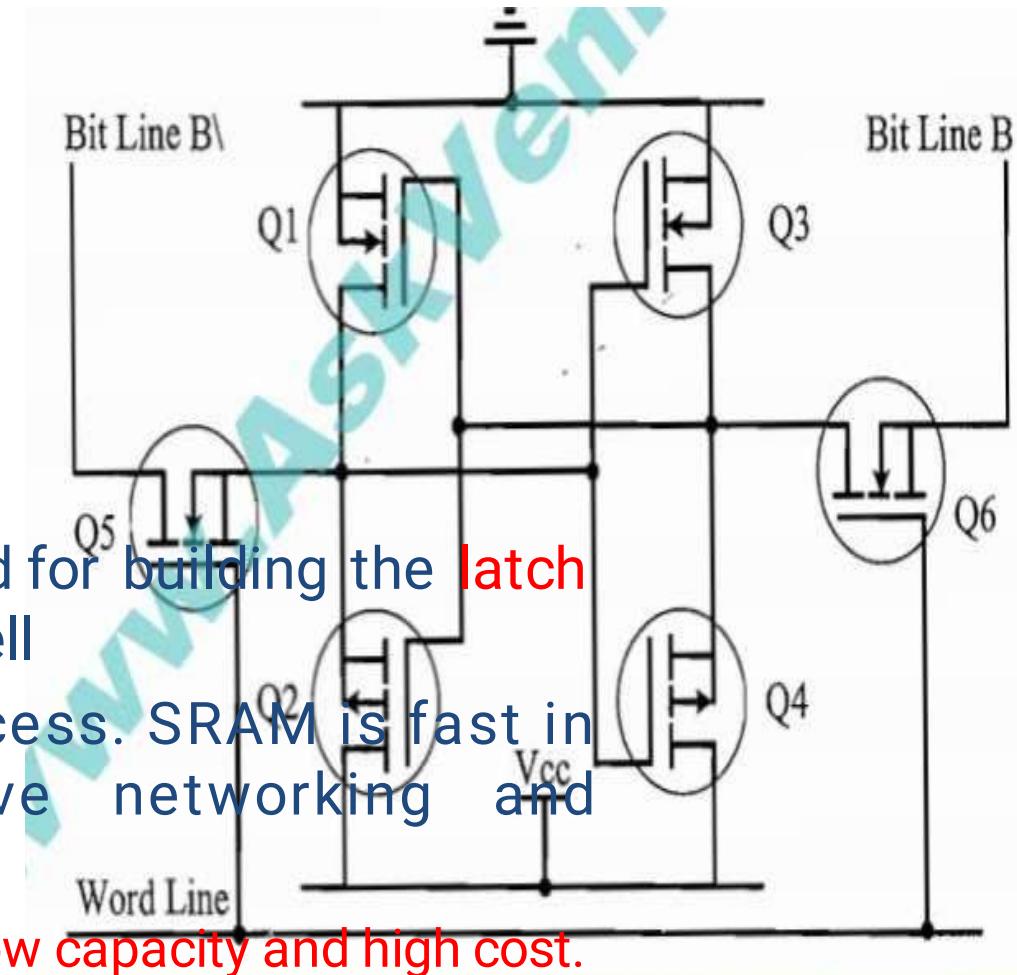
## 2.2

# Read-Write Memory/Random Access Memory (RAM)

Static RAM (SRAM):

Static RAM (SRAM): the form of

- SRAM stores data in the ~~voltage~~
- ~~The cell is made up of flip-flops.~~
- ~~A flip-flop for memory cell~~
- A flip flop ~~consists of four transistors~~ (or 6 MOSFETs) along with some wiring,
  - ~~Four of the transistors are used for building the latch (flip-flop) part of the memory cell~~
  - ~~Two for controlling the access.~~ SRAM is fast in operation due to its resistive networking and switching capabilities



The major limitations of SRAM are low capacity and high cost.  
switching capabilities.

The major limitations of SRAM are low capacity and high cost.



Edit with WPS Office



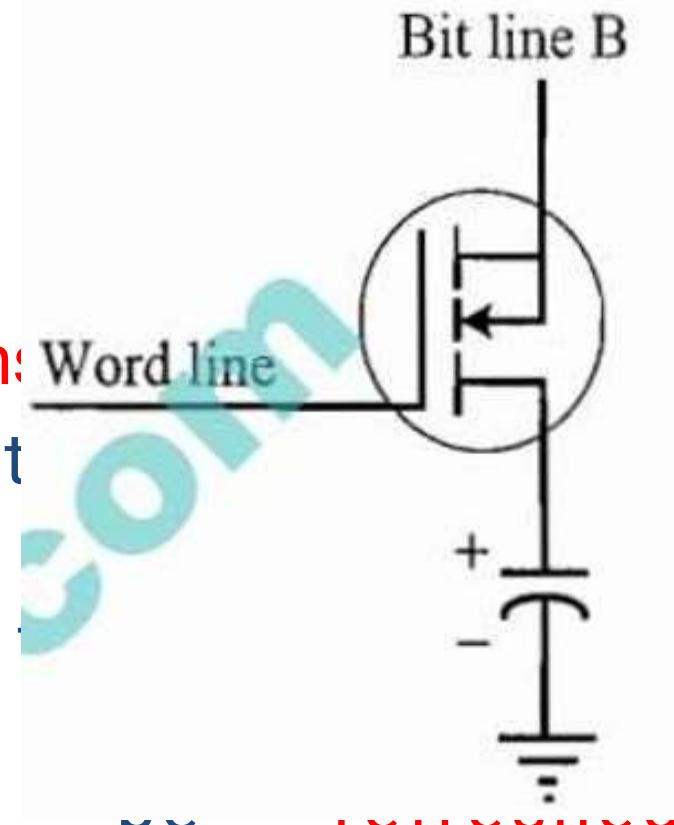
## 2.2

# Read-Write Memory/Random Access Memory (RAM) **Memory**

Dynamic RAM (SRAM).

**Dynamic RAM (SRAM):**

- DRAM stores data in the form of charge.
- They are made up of MOS transistors.
- Advantages: its high density and low cost compared to SRAM.
- Disadvantage: is that since it is stored as charge it gets leaked off over time and to prevent this they need to be refreshed periodically.



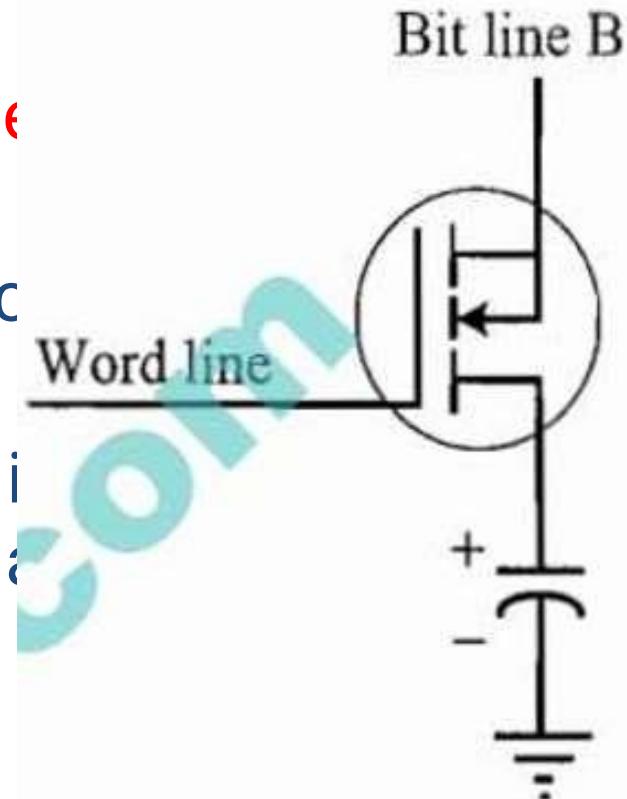


## 2.2

# Read-Write Memory/Random Access Memory (RAM)

Dynamic RAM (SRAM):

- Dynamic RAM (SRAM):
- Special circuits called DRAM controllers handle the refresh operation.
- The refresh operation is done periodically in milliseconds.
- The MOSFET acts as the gate for the incoming and outgoing data whereas the capacitor acts as the storage unit.



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

# 2.2 Memory

## SRAM vs



### SRAM Cell

### DRAM Cell

Made up of 6 CMOS transistors (MOSFET)

Made up of a MOSFET and a capacitor

Doesn't require refreshing

Requires refreshing

Low capacity (Less dense)

High capacity (Highly dense)

More expensive

Less expensive

Fast in operation.

Typical access time is 10 ns.

Slow in operation due to refresh requirements.

Typical access time is 60 ns. Write operation is faster than read operation.



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

## 2.2 Memory



### Read-Write Memory/Random Access Memory (RAM)

#### NVRAM

- Non-volatile RAM is a random access memory with battery backup.
- It contains:
  - Static RAM based memory
  - A minute battery for providing supply to the memory → in the absence of external power supply.
- The memory and battery are packed together in a single package.
- The life span of NVRAM is expected to be around 10 years.
- The life span of NVRAM is expected to be around 10 years.



Edit with WPS Office



## 2.3 Sensors and Actuators

- A sensor is a transducer device :
  - That converts energy from one form to another
  - For any measurement or control purpose.
- Sensors:
  - Capture the input signals/ upgrade the input signal with the changes of real time.
  - Sensors connected to the input port of the embedded system.



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt



## 2.3 Sensors and

- **Actuator** is a form of transducer device:
  - (mechanical or electrical) which converts signals to corresponding physical action (motion).
  - Actuator acts as an output device.
- If the embedded system is designed for any controlling purpose:
  - the system will produce some changes in the controlling variable to bring the controlled variable to the desired value.
- It is achieved through an actuator connected to the **output** port of the embedded system.



## 2.3 Sensors and

### Actuators

- If the embedded system is designed for monitoring purpose only:
  - Then there is **no need for including an actuator** in the system.
- For example, **ECG machine**:
  - It is **designed** to monitor the heart beat status of a patient
  - It cannot impose a control over the patient's heart beat and its order.
  - It **cannot** impose a control over the patient's heart beat and its order.
  - Sensors used to read the status.
  - **Sensors used to read the status.**
  - The variations are captured and presented to the user (may be a doctor) through a visual display or some printed chart.



# I/O Subsystem

- The I/O subsystem of the embedded system
  - Facilitates the interaction of the embedded system
  - With the external world.
- The interaction happens through:
  - Sensors connected to input Port: → can also be interfaced through other devices like ADC, Optocouplers etc.
  - Actuators connected to the output port



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

# Few I/O Subsystem



- Light Emitting Diode (LED)
- 7-Segment LED Display
- Optocoupler
- Stepper Motor
- Relay
- Piezo Buzzer
- Push Button Switch
- Keyboard
- Programmable Peripheral Interface (PPI)



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

# Light Emitting Diode (LED)

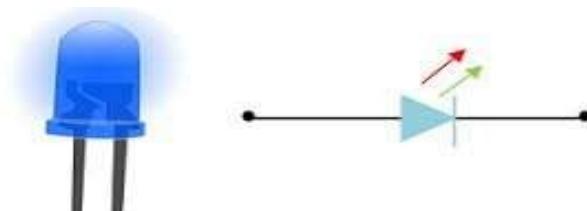


**LED Symbol:** It is an **Output device** → indicated by an arrow in the symbol



**LED** → Emits light as an output

**Examples:** Device ON state, Battery Low state, charging of battery → some Embedded systems

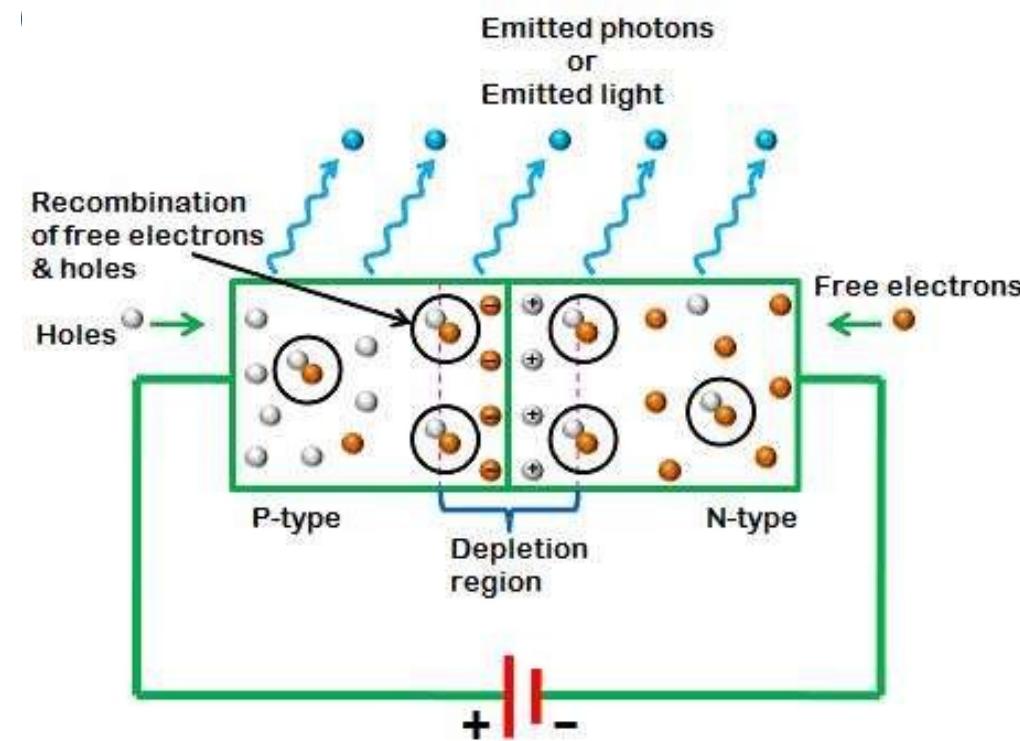


Light Emitting Diode

# Light Emitting Diode (LED)



LED: It is a p-n junction diode:



Light Emitting Diode (LED)

Physics and Radio-Electronics



Department of ISE

Edit with WPS Office

BMS Institute of Technology and Mgmt

When p-n junction is forward biased i.e. Anode  $\rightarrow$  +ve voltage  
Cathode  $\rightarrow$  -ve voltage

Working:

Holes in p-n junction gains energy from the voltage supplied and moves towards electrons

Recombination of free electrons and holes occurs by emitting energy.

Emitted energy is released as light  $\rightarrow$  output

# Light Emitting Diode (LED)



## LED Interface: Interface:

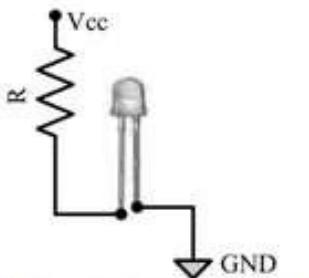


Fig. 2.13 LED interfacing

The current flowing through LED can be limited by:

- Connecting resistor in series to LED

LED interfaced to the port pin of the processor / controller in 2 ways:

1. Anode connected to port pin:

1. Anode connected to port pin:
  - Port pin sources current to LED when port pin is logic HIGH('1') i.e. +ve voltage

2. Cathode connected to port pin and Anode to voltage through current limiting resistor:

2. Cathode connected to port pin and Anode to voltage through current limiting resistor:



# 7-Segment LED Display

## 7-segment LED display:

- Is an **output device** for displaying **alphanumeric characters**.
  - It contains **8 LED segments** arranged in a special form.
  - It contains **8 LED segments** arranged in a special form.
  - Out of the 8 LED segments
    - 7 are used for displaying alphanumeric characters and are named as A-G
    - 1 is used for representing 'decimal point' in decimal representation
    - 1 is used for number display.
- number display.

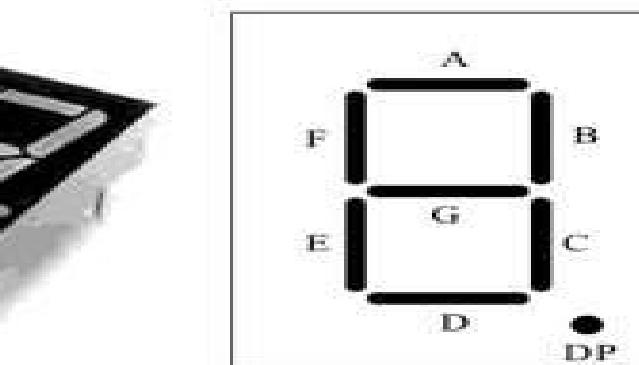
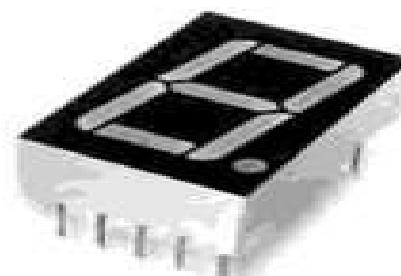


Fig. 2.14) 7-Segment LED Display



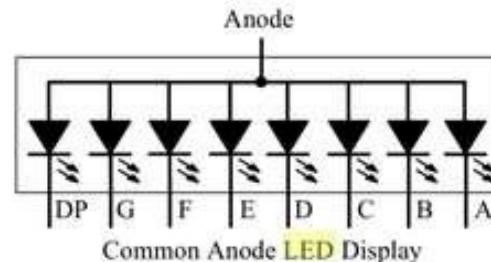
# 7-Segment LED Display



7-segment LED display: Available in two different configurations:

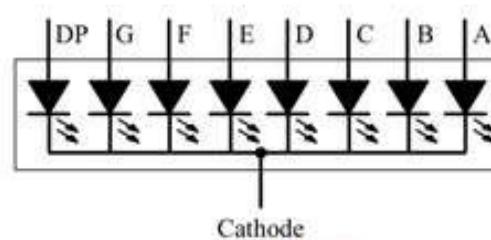
- Common Anode

- = Anodes of the 8 segments are connected commonly



- Common Cathode

- = Cathode of the 8 segments are connected commonly



Edit with WPS Office

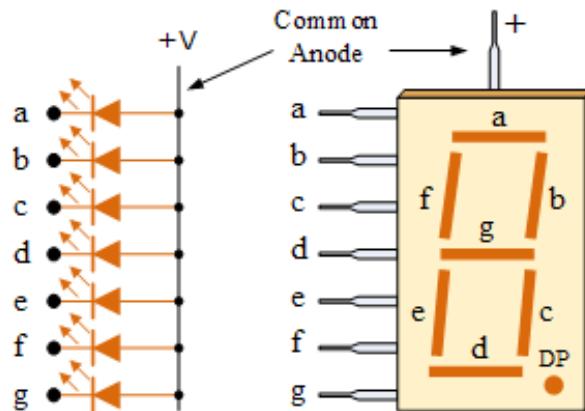
# 7-Segment LED Display:

Available in two different configuration:

## Display:



- Common Anode :

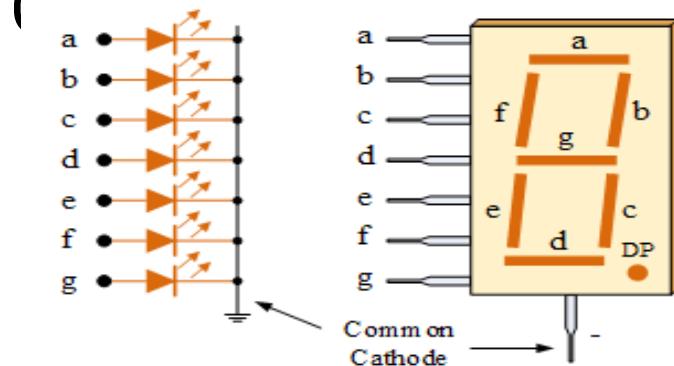


- **Anode** connected to +V i.e. High
- **Cathode**: Each LED grounded

All

Current limiting  
resister → Used to  
interface in any  
Circuit

- Common Cathode :



- **Anode** : Each LED to Voltage
- **Cathode**: All 7 LED segments connected to GND i.e. Low



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt



# Stepper Motor

A stepper motor:

- Is an **electro-mechanical device** which generates discrete displacement (motion) in response to dc electrical signals.
- It **differs from the normal dc motor** in its operation.
  - DC motor produces continuous rotation on applying dc voltage
  - Stepper motor produces discrete rotation in response to the dc voltage applied to it.
  - Stepper motor produces **discrete rotation** in response to the dc voltage applied to it.
- The paper feed mechanism of a printer/fax makes use of stepper motors for its functioning.
- **The paper feed mechanism** of printer/fax makes use of



Department of ISE

Edit with WPS Office

BMS Institute of Technology and Mgmt

makes use of

# Stepper Motor



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

- Unipolar
- Bipolar
- Bipolar

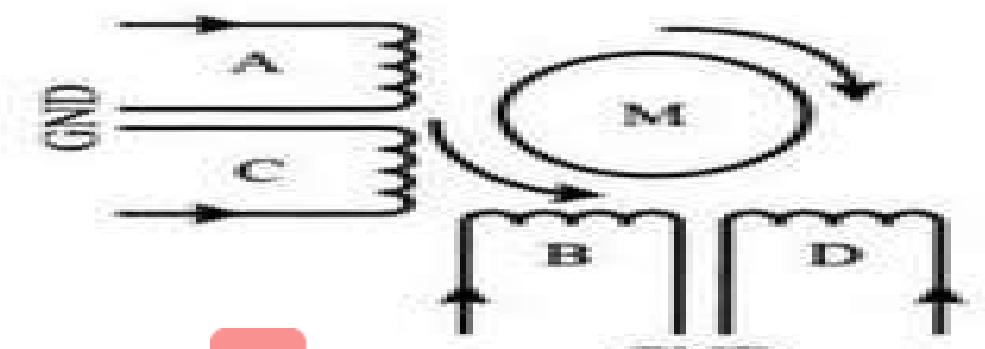
# Unipolar:-Stepper Motor



A unipolar stepper motor contains **two windings per phase**.

The direction of rotation (clockwise or anticlockwise) of a stepper

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

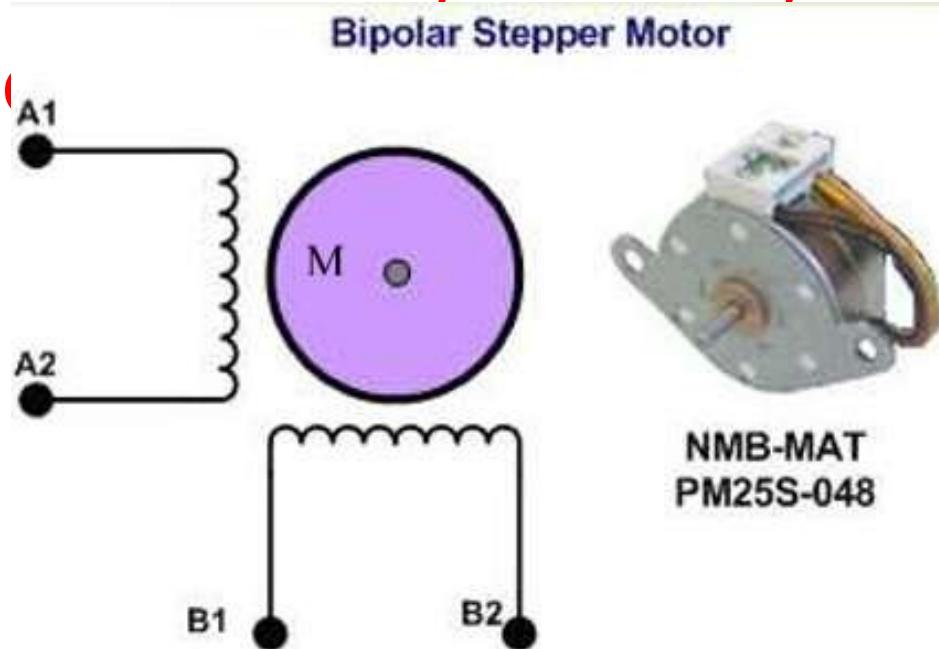


# Bipolar:-Stepper Motor



A bipolar stepper motor contains single winding per phase.

- For reversing the motor rotation the current flow through the windings is reversed dynamically.
- For reversing the motor rotation the current flow through the windings is reversed dynamically.
- It requires complex circuitry for current flow reversal.
- It requires current flow reversal.



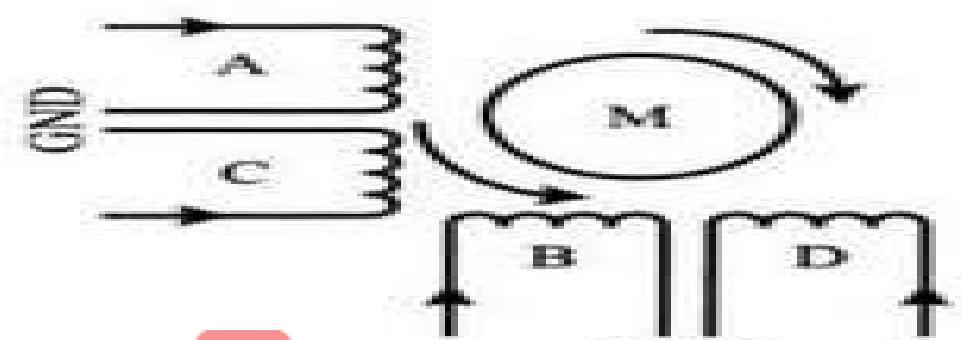


# Stepper Motor

The different stepping modes supported by stepper motor are explained below.

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

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



Edit with WPS Office

Department of EEE

BMS Institute of Technology and Management

Department of EEE



# Stepper Motor

Wave Step: In the step mode only one phase is energized at a time.

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

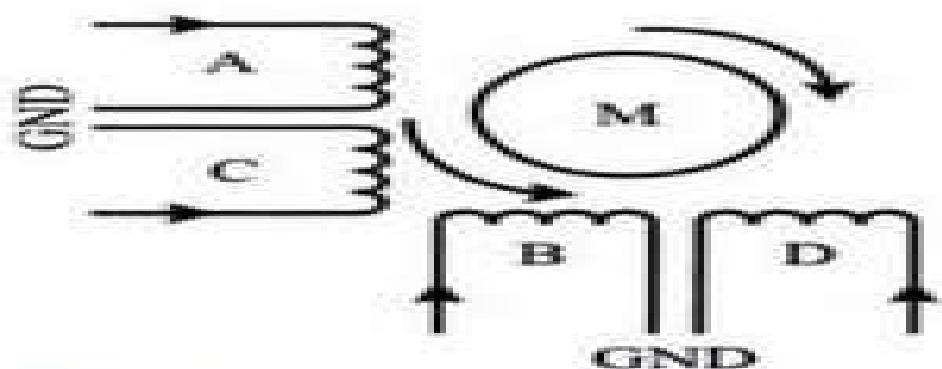


Fig. 2.18 2-Phase unipolar step-  
motor driver



# Stepper Motor

## Motor

Half Step: In the combination of Full and Half step.

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



Full Step

Step	Coil A	Coil B	Coil C	Coil D
5	H	L	L	L
6	L	H	L	L
7	L	L	H	L
8	L	L	L	H



Half Step

# Stepper Motor

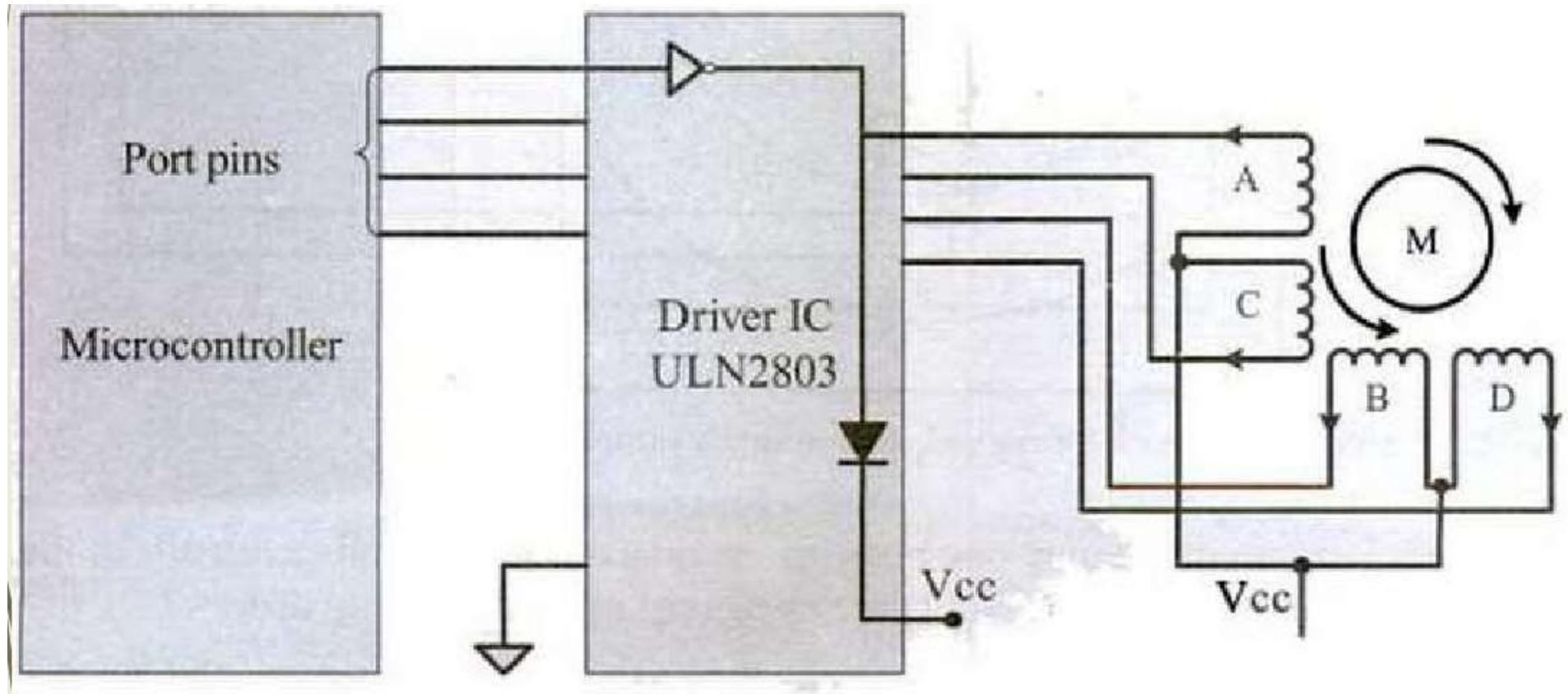


- The rotation of the stepper motor can be reversed by reversing the order in which the coil is energized.
- Two-phase unipolar stepper motors are the popular choice for embedded applications.
- The current requirement for stepper motor is little high and hence the port pins of a microcontroller/processor may not be able to drive them directly.
- Also the supply voltage required to operate stepper motor varies normally in the range 5V to 24 V.
- Depending on the current and voltage requirements, special driving circuits are required to interface the stepper motor with microcontroller/processors.



Edit with WPS Office

# Interfacing of a stepper motor through a driver circuit connected to the port pins of a microcontroller/processor.



# PushButton Switch



It is an input device. Push button switch comes in two configurations

- 'Push to Make':
    - The switch is normally in the open state and it makes a circuit contact when it is pushed or pressed.
  - 'Push to Break':
  - 'Push to Break':
    - The switch is normally in the closed state and it breaks the circuit contact when it is pushed or pressed.
- In the embedded application push button is generally used as reset and start switch.
- In the embedded application push button is generally used as reset and start switch.

# Push Button Switch



Depending on the way :- Push button interfaced to the controller:

- A Low Pulse Generator
- A High Pulse is Generator

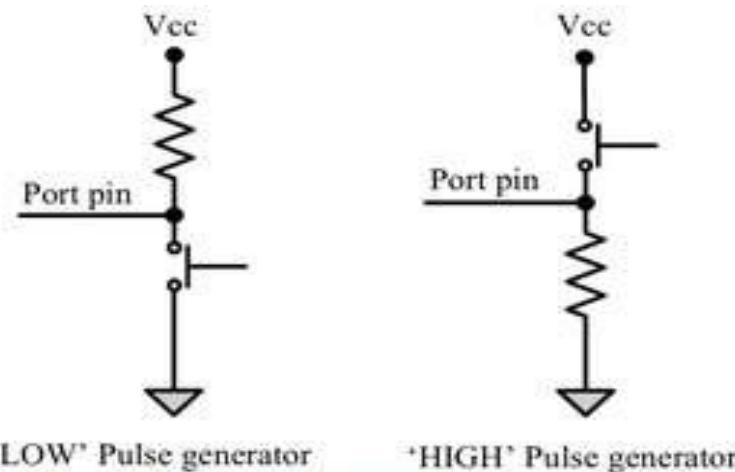


Fig. 2.23 Push button switch configurations



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

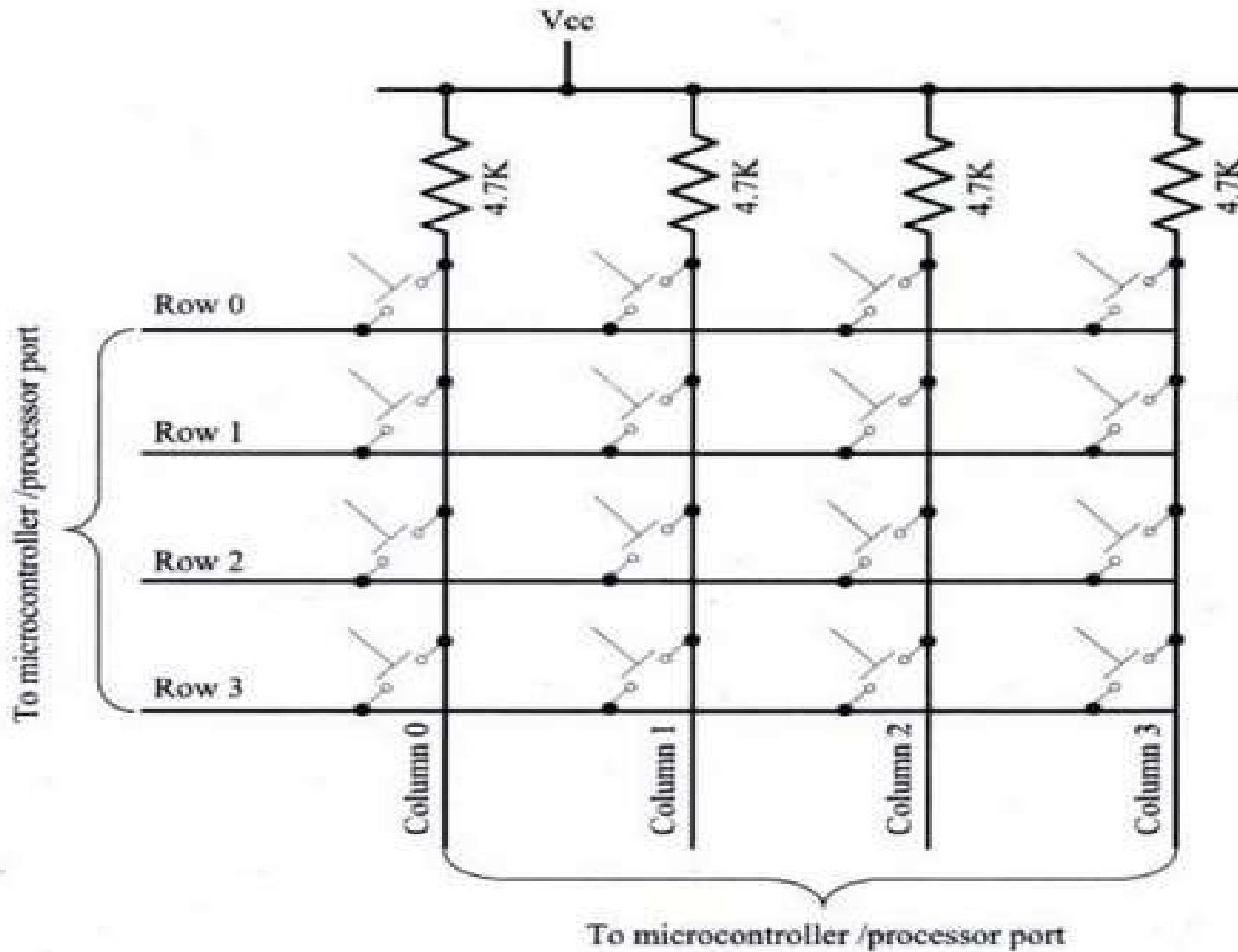


# Keyboard

- Keyboard is an **input** device for user interfacing.
  - If the number of keys required is **very limited**, push button switches can be used.
  - They can be **directly interfaced to the port pins for reading**.
- Matrix keyboard is an optimum solution for handling large key requirements.
- **Matrix keyboard** is an optimum solution for handling large key requirements.
  - It greatly reduces the number of interface connections.
  - It greatly **reduces the number of interface connections**.
- For example, for interfacing 16 keys, in the direct interfacing technique 16 port pins are required, 16 keys, in the direct interfacing technique 16 port pins are required, whereas the matrix keyboard only 8 lines are required. The 16 keys are arranged in a 4 column\*4 row matrix.
- For example, for interfacing 16 keys, in the direct interfacing technique 16 port pins are required, whereas the matrix keyboard only 8 lines are required.

# Keyboard

## d



Edit with WPS Office

# Working of Matrix Keyboard



- Matrix Keyboard has 4 row and 4 columns
  - Initially pull down Row0 and then read the columns connected to that Row.
  - Next go to Row1 and read Columns of that Row1,
  - Next go to Row1 and read Columns of that Row1,
    - Go on to the other Rows till Row3
  - When a key is pressed, the status information of Column will have Logic '0'.
- Since the key press is a mechanical device, there can be multiple key press:
  - Hardware De-bouncing Technique:
    - Overcome with de-bouncing technique
  - Software De-bouncing Technique:
    - Overcome with de-bouncing technique
- Since the key press is a mechanical device,
  - Hardware De-bouncing Technique:
    - Additional circuit is used.
  - Software De-bouncing Technique:
    - Key is read twice with de-bounce delay.
- Software De-bouncing Technique:
  - Key is read twice



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

# 2.4 Communication Interface

Communication Interface is essential for communicating with:

- Various subsystems of ES
- With external world

Two types of Communication Interface:

1. Onboard: → Device/board level

2. External: → Product level

– Example: Printed Circuit Board, Serial Interfaces like

UART, Parallel Bus Interfaces

|

s

e

– Communication between ES

– Example: Serial and Parallel Interfaces like Infrared(IR),  
Bluetooth(BT), Wi-Fi, GPRS etc.

Infrared(IR),

Bluetooth(BT), Wi-Fi, GPRS etc.



r

## 2.4 Communication Interface

Onboard Communication Interface refers:



• Different Communication channel/ buses for integrating for various IC's and other peripheral devices.

• Different types are:

I. Inter Integrated Circuit (I2C) Bus

II. Serial Peripheral Interface (SPI) Bus

III. Universal Asynchronous Receiver Transmitter

IV. Universal Asynchronous Receiver (UART)

V. 1-Wire Interface

VI. Parallel Interface

VII. Parallel Interface

# Onboard Communication Interface

## Integrated Circuit (I2C) Bus:



### • It is a

– Synchronous

– Bi-directional

– (But ) half duplex: e.g. Walkie Talkie(Hand held Radio)      e.g. **Walkie-Talkie(Hand held Radio)**

### • Two wire serial interface bus.

### • Two wire serial interface bus.

– i.e. 2 bus lines

– i.e. 2 bus lines

• SCL(Serial Clock):→ responsible for generating synchronization clock pulses

• SCL(Serial Clock):→ responsible for generating synchronization clock pulses

• SDA(Serial Data):→ responsible for transmitting the serial data across devices

• SDA(Serial Data):→ responsible for transmitting the serial data across devices

# Onboard Communication Interface

## Inter Integrated Circuit (I2C) Bus: Interface



2 Buses: SCL, SDA

2 Devices: Master & Slave

**Master Device:** For controlling

- Initiating/terminating data transfer
- Sending data
- Generating necessary synchronization clock pulses.

**Slave Device:**

- Wait for the commands from the master
- Respond upon receiving the commands.

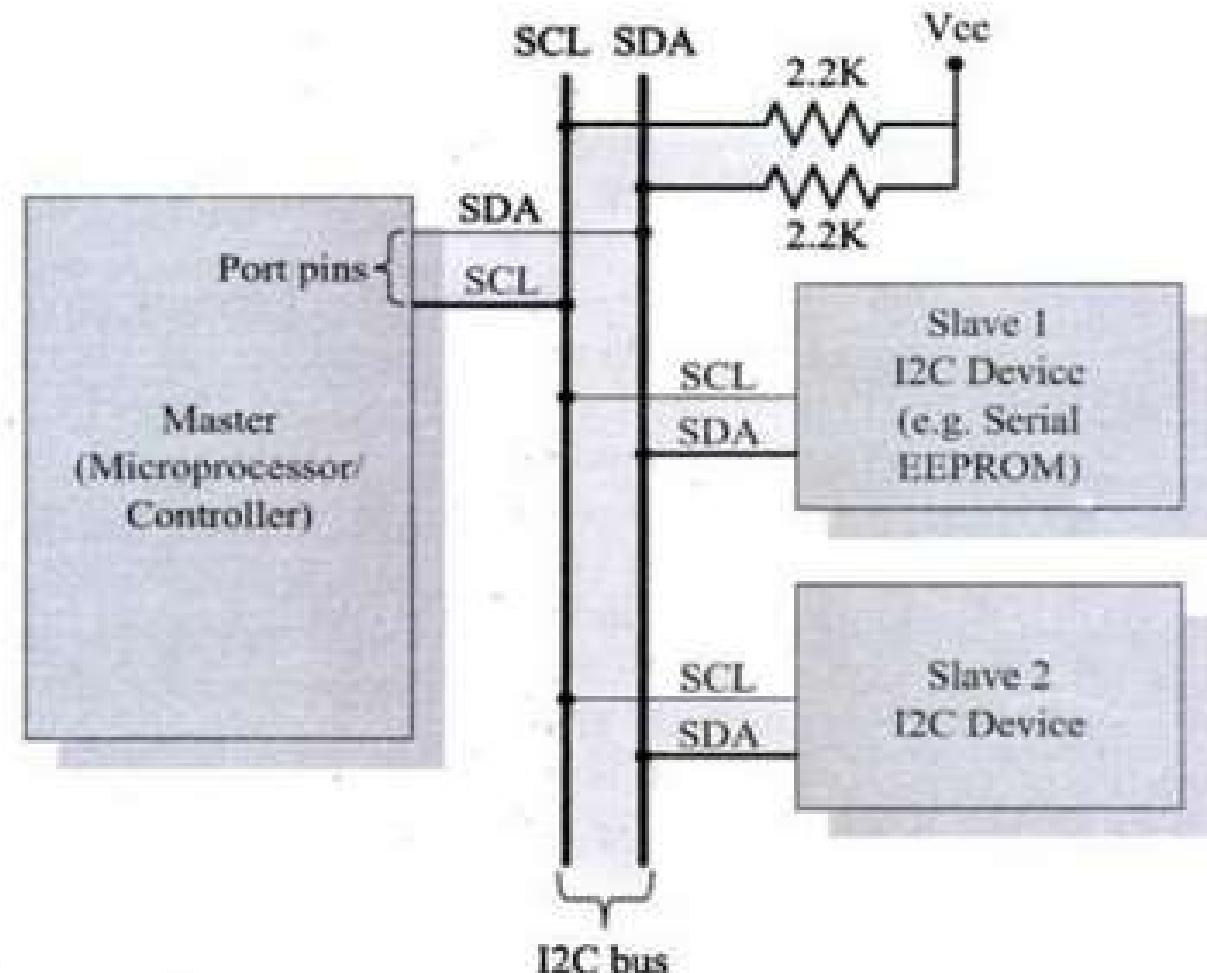


Fig. 2.26

I2C Bus Interfacing



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

# Onboard Communication Interface



## Working of I2C bus interface:

### 1. Master device operation:

- $SCL \rightarrow$  Logic 'High': As  $SCL=1$ , It means to start Data operation
- $SDA \rightarrow$  Logic 'Low'
- Sends address of the slave for further communication & MSB is important
  - $MSB=1 \rightarrow$  Read operation
  - $MSB=0 \rightarrow$  Write operation
- Waits for the slave for acknowledgement
- Waits for the slave for acknowledgement

### 2. Slave device operation:

- #### 1. Slave device operation:
- Sends a bit=1 as an acknowledgement
  - Sends a bit=1 as an acknowledgement
  - Performs read/ write operation according to the masters
  - Performs read/ write operation according to the masters

### 3. Master device finally pulls SDA $\rightarrow$ High performing STOP

#### 2. Master device finally pulls SDA $\rightarrow$ High performing STOP

operation.



Department of ISE

Edit with WPS Office

BMS Institute of Technology and Mgmt

STOP

# Onboard Communication Interface

## iii. Serial Peripheral Integrated (SPI) Bus:



- It is a
  - = Synchronous
  - = Bi-directional
  - = (But ) Full duplex: e.g. Telephone(1 persons can walk and other can hear, Vice versa )
  - = Single master multi-slave system: At a time one master is active
- Four wire serial interface bus.
- MOSI(Master Out Slave IN):→data from Master to Slave
- MOSI(Master Out Slave IN):→data from Master to Slave
- MISO(Master IN Slave Out):→data from Slave to Master
- SCLK(Serial Clock):→carrying clock signals
- MISO(Master IN Slave Out):→data from Slave to Master
- SS(Slave Select):→In active low, slave device is selected.
- SCLK(Serial Clock):→carrying clock signals
- SS(Slave Select):→In active low, slave device is selected.



# Onboard Communication Interface

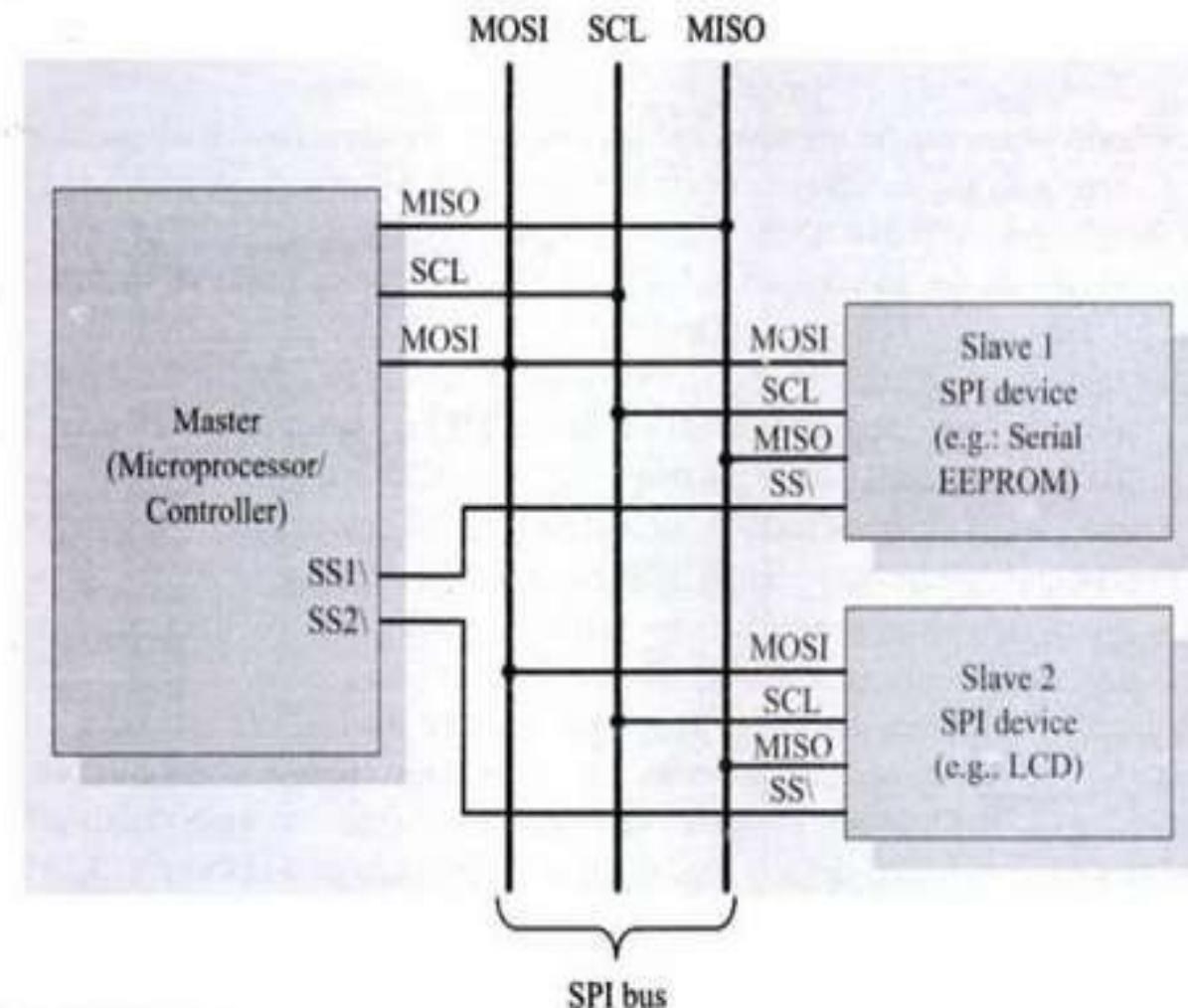
## Serial Peripheral Integrated (SPI) Bus:

### One master & Multi Slave

**Master Device:** Will send active low to different devices for communication.

**Slave Device:** e

- If Slave → **Low**, then those devices will be active
- If Slave → **High**, then those devices will be in-active



# Onboard Communication Interface



Working of SPI bus interface:

Working of SPI bus interface:  
**SPI works as a Shift Register :**

- Register is in multiple of 8
- In **MOSI** → Data from Master Shift register is sent to Slave shift register.
- In **MISO** → Data from Slave Shift register is sent to Master shift register



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

# Onboard Communication

## iii) Universal Asynchronous Receiver Transmitter (UART)



- It is a

- Asynchronous form of serial data transmission:-
    - Means → Doesn't require clock to synchronous transmitting / receiving end
  - Relies on the initial agreement of the type of data transfer:-
  - Relies on the initial agreement of the type of data transfer:-
    - Includes: Baud rate, parity, No. of bits per byte etc.

- It contains:-

- Includes: Baud rate, parity, No. of bits per byte etc.

- It contains:-

- Start bit: specifies start of data stream of communication followed by start bit.

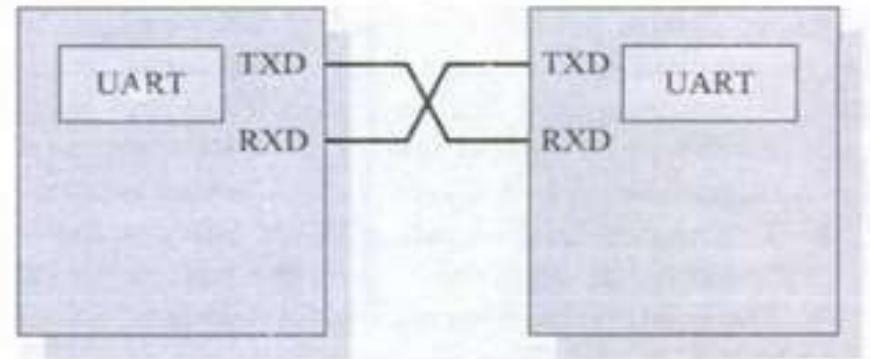
- Stop bit: specifies stop of data stream of communication followed by stop bit.

# Onboard Communication Interface

## Universal Asynchronous Receiver Transmitter (UART)



### Universal Asy (UART)



TXD: Transmitter line

RXD: Receiver line

- Start bit specifies the receiver that data will be sent.
- According to the Baud rate receiver gets ready to accept data.
- Sender sends data along with a parity bit and stop bit
- Once the receiver receives data, parity is checked to see data received is corrupted/not



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

# Onboard Communication Interface

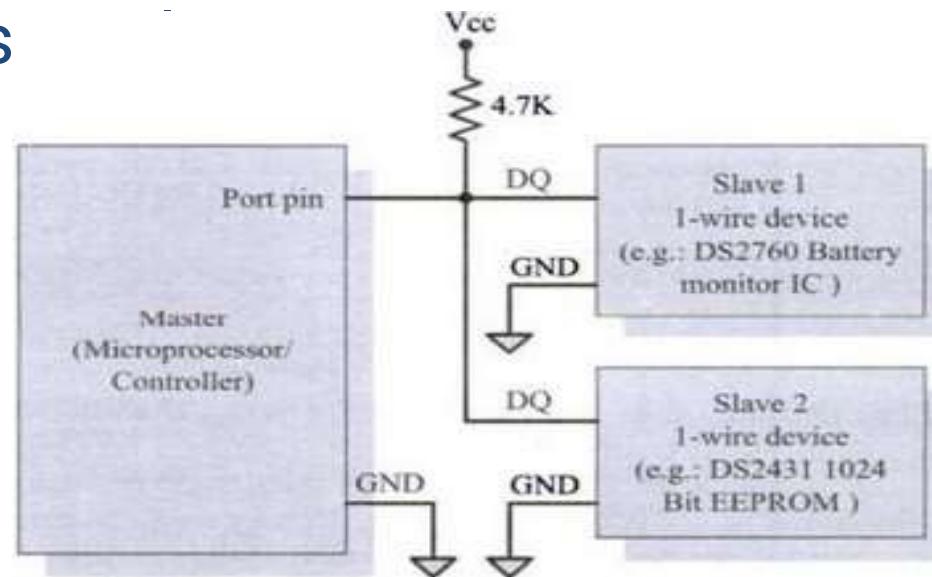
## 1-Wire Interface:

- It is a

– Asynchronous half duplex communication protocol  
It uses single signal line (Wire) is called DQ for  
communication.

– Uses Master-Slave communication model.

– Uses Master model.



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt



# Onboard Communication Interface

## Working of 1-wire interface:



1. Master device :- sends a 'Reset' pulse on 1-wire bus.
2. Slave device connected to that wire responses as 'Present' pulse
3. Master send a ROM command: → Initiates communication at slave device.
4. Master send a ROM command: → Initiates communication at slave device.
5. Master send read/ write of control command
6. Master send read/ write of control command
7. Master device initiates read / write data between devices



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

# Onboard Communication

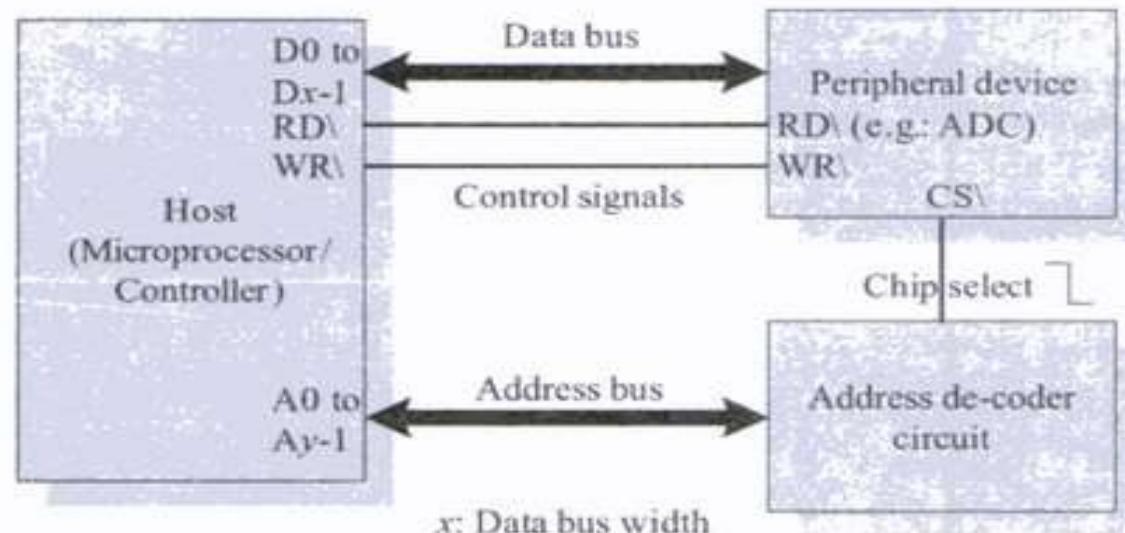


## Parallel Interface:

It is used for **Communication** with **peripheral devices** connected to the **host**

- Contains:

- Contains:
  - Address lines:- to select peripheral device
  - Address lines:- to select peripheral device
  - Control Signals:- to specify read/ write operation
  - Data lines:- data is sent/ received to/ from peripheral devices



# 2.4 Communication Interface



**External Communication Interface refers:**

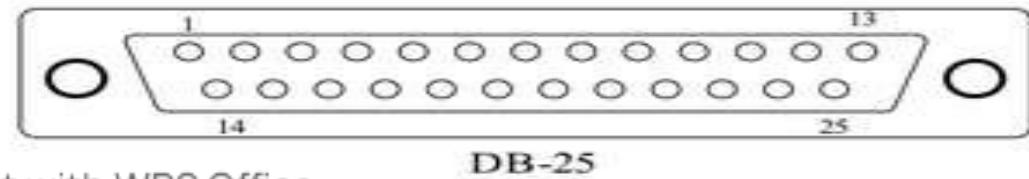
- Different Communication **channel/buses** used by the Embedded System to communicate with the External world.
- The various interfaces for external communication are as follows:
- The various interfaces for external communication are as follows:
  - I. RS-232 C & RS-485
  - II. Universal Serial Bus (USB)
  - III. IEEE 1394 (Firewire)
  - IV. RS-232 C & RS-485
  - V. Bluetooth (BT)
  - VI. Infrared (IrDA)
  - VII. Zigbee
  - VIII. GPRS
  - IX. Wi-Fi
  - X. 5G



# External Communication Interface

## RS-232C & RS-485

- RS-232 is a
  - Legacy
  - Full duplex
  - Wired
  - Asynchronous
  - Supports only point-to-point communication and not suitable for multi-drop communication
- RS-232 has 2 different type connectors.
  - DB-9 → 9-Pin connector
  - DB-25 → 25-Pin connector



Edit with WPS Office

# External Communication Interface

## RS-232C & RS-485



- RS-485 is a

- Supports multi-drop communication.
  - (With up to) 32 transmitting devices (drivers)
  - 32 receiving devices on the bus.
  - It uses addressing mechanism for identifying devices.
  - It uses addressing mechanism for identifying devices.

f

# External Communication Interface

## Universal Serial Bus (USB)



### Universal Serial Bus (USB) is a:

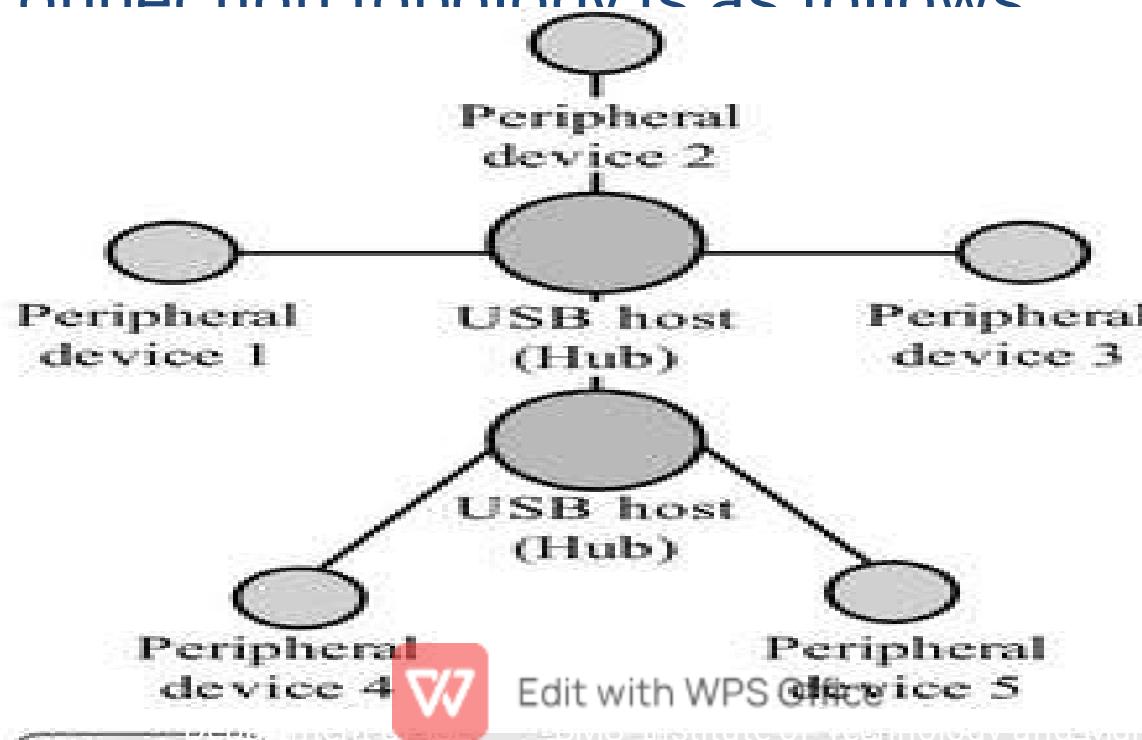
- Wired high speed serial bus for data communication.
- The USB host can support connections up to 127, including:
- The USB host can support connections up to 127, including:
  - Slave peripheral devices
  - Other USB hosts.
- USB transmits data in packet format with standard format.
- USB is initiated by the host and host is responsible for:
  - Data communication
  - Connecting peripheral(Slaves) devices
  - Packetizing and formatting the data format.
- USB is initiated by the host and host is responsible for:
  - Data communication
  - Connecting peripheral(Slaves) devices
  - Packetizing and formatting the data format.

# External Communication Interface

## Universal Serial Bus (USB)



- USB host controller can be implemented in different standards
  - Open Host Control Interface(OHCI)
  - Open Host Control Interface(OHCI)
  - Universal Host Control Interface(UHCI)
  - Universal Host Control Interface(UHCI)
- USB Connection topology is as follows:
- USB Connection topology is as follows:





# External Communication Interface

## IEEE 1394 (Firewire)

- **Wired**
- **Isochronous high speed serial communication bus.**
- **It is also known as High Performance Serial Bus (HPSB).**
- **Unlike USB interface,**
  - IEEE 1394 **doesn't** require a **host** for communicating between devices.
  - Data rate is higher than USB
  - Data rate is **higher** than USB
  - Hardware implementation is **costlier** than USB
- **1394 is a popular communication interface for connecting embedded devices like:**
  - Digital Camera, Camcorder, Scanners to desktop computers for data transfer and storage.
  - Digital Camera, Camcorder, Scanners to desktop computers for data transfer and storage.
  - For example, you can directly connect a scanner with a printer for printing.
  - For example, you can directly connect a scanner with a printer for printing.

# External Communication Interface

## IV. Infrared (IrDA)



- Serial
- Half duplex,
- Uses wireless technology for data communication between devices.
- It is in use from the olden days of communication and you may be very familiar with it.
- The remote control of your TV, VCD player, etc works on Infrared data communication principle.
- The ren
- works c



# External Communication Interface

## IV. Bluetooth (BT)



- Bluetooth is a
  - low cost, low power, short range wireless technology for data and voice communication.
  - Bluetooth supports point-to-point (device to device) and point-to-multipoint (device to multiple device broadcasting) wireless communication.
  - A Bluetooth device can function as either master or slave.
    - When a network is formed with one Bluetooth device as master and more than one device as slaves, it is called a Piconet.
    - A Piconet supports a maximum of seven slave devices
    - A Piconet supports a maximum of seven slave devices



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

# External Communication Interface

## V. Bluetooth (BT)



- Bluetooth is a
  - Bluetooth technology is very popular among cell phone users as they are the easiest communication channel for transferring ringtones, music files, pictures, media files, etc. between neighboring Bluetooth enabled phones.
  - It supports a data rate of up to 1 Mbps and a range of approximately 30 feet for data communication.



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

# External Communication Interface

## VI. WiFi (Wireless Fidelity)



- Is the popular **wireless** communication technique for **networked** communication of devices.
- Wi-Fi is intended for **network** communication and it supports **Internet Protocol (IP)** based communication.
- Wi-Fi is intended for **network** communication and it supports **Internet Protocol (IP)** based communication.
- It is essential to have device identities in a **multipoint** communication to address **specific** devices for data communication.
- It is essential to have device identities in a **multipoint communication** to **address specific** devices for data communication.
- Wi-Fi based communications require an **intermediate agent** called **Wi-Fi router/Wireless access point** to manage the communications.
- Wi-Fi supports data rates ranging from 1 Mbps to 150 Mbps and offers a range of 100 to 300 feet.
- Wi-Fi supports data rates ranging from 1 Mbps to 150 Mbps and offers a range of 100 to 300 feet.

# External Communication Interface

## Wireless ZigBee Interface



- Is a low power, low cost

- Is a low power, low cost
  - Wireless network communication protocol based on the IEEE 802.15.4-2006 standard.
  - ZigBee is targeted for low power, low data rate and secure applications for Wireless Personal Area Networking (WPAN).
  - ZigBee operates worldwide at the unlicensed bands of Radio spectrum, mainly at 2.400 to 2.484 GHz, 902 to 928 MHz and 868.0 to 868.6 MHz.
  - ZigBee operates worldwide at the unlicensed bands of Radio spectrum, mainly at 2.400 to 2.484 GHz, 902 to 928 MHz and 868.0 to 868.6 MHz.
  - ZigBee supports an operating distance of up to 100 meters and a data rate of 20 to 250Kbps.
  - ZigBee supports an operating distance of up to 100 meters and a data rate of 20 to 250Kbps.



# External Communication Interface

ZigBee device categories are as follows:

- **ZigBee Coordinator (ZC)/Network Coordinator:**
  - Acts as the root of the ZigBee network.
  - The ZC is responsible for initiating the ZigBee network and it has the capability to store information about the network.
- **ZigBee Router (ZR)/Full Function Device (FFD):**
  - Responsible for passing information from one device to another device or to another ZR.
- **ZigBee End Device (ZED)/Reduced Function Device (RFD):**
  - End device containing ZigBee functionality for data communication.
- **ZigBee End Device (ZED)/Reduced Function Device (RFD):**
  - End device containing ZigBee functionality for data communication.



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt



# External Communication Interface

## VIII General Packet Radio Service (GPRS)

- GPRS is a communication technique for transferring data over a mobile communication network like GSM.
- GPRS supports a theoretical maximum transfer rate of 171.2 kbps.
- The GPRS communication divides the channel into 8 timeslots and transmits data over the available channel.



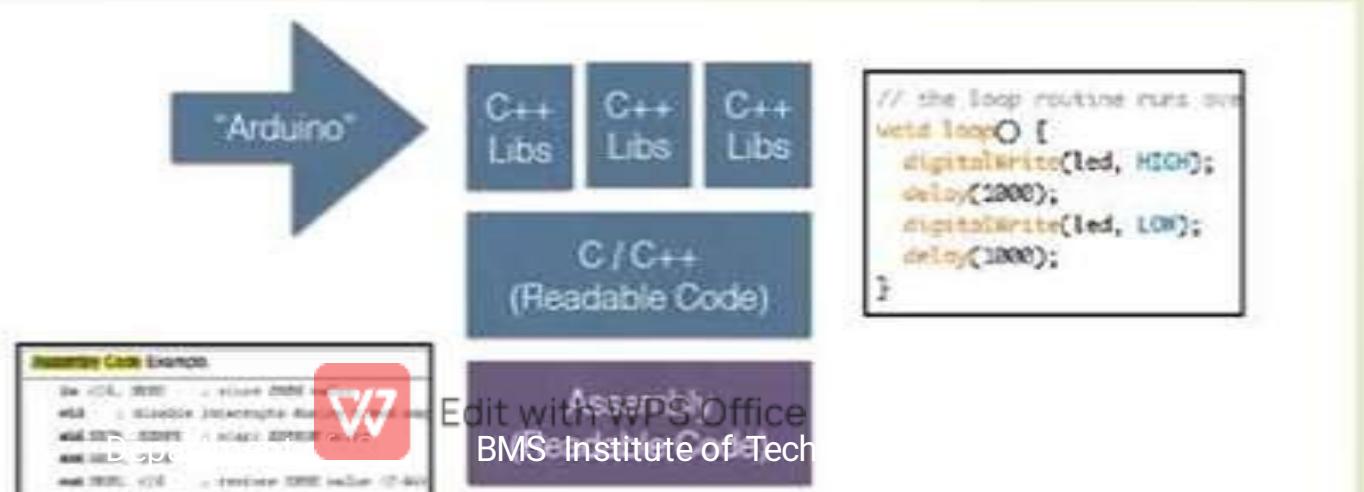
# 2.5 Embedded Firmware



Embedded Firmware refers to:

- To the **control algorithm** (Program instructions) and/or
- The **configuration settings** that an embedded system developer dumps into the **code** (program) memory of the embedded system.
- There are various methods available for developing the embedded firmware. They are Writing the program
- There are **various methods** available for developing the embedded firmware. They are Writing the program
- 1. In high level languages like Embedded C/C++ using an Integrated Development Environment.
- 2. In Assembly language using the instructions supported by your applications target processor/controller.
- 1. In high level languages like C/C++

g  
Developm  
2. In Assem  
pported  
application's ta



Edit with WPS Office  
BMS Institute of Tech

# 2.5 Embedded Firmware



- The **program** written in either of the methods given above :
  - Should be converted into a processor understandable **machine code before loading** it into the program memory.
- The process of converting is called ‘**HEX File Creation**’.
- If the program is written in **Embedded C/C++** using an IDE, **machine code before loading** it into the program memory.
- The process of converting is called ‘**HEX File Creation**’.
- If the program is written in **Embedded C/C++** using an IDE, **machine code before loading** it into the program memory.
- If you are following the Assembly language based programming technique:
  - The **cross compiler included in the IDE converts** it into corresponding processor/controller understandable ‘**HEX File**’.
  - Can use the utilities supplied by the processor/controller vendors to convert the source code into ‘**HEX File**’.
- If you are following the **Assembly language** based programming technique:
  - Can use the utilities supplied by the

## 2. 6 Other System Components



### Components

The other system components refers

- To the **components/circuits/ICs** which are necessary for the **proper functioning of the embedded system**.

Few of the essential components are:

Few of the essential components are:

- I. Reset Circuit
- II. Brown-out Protection Circuit
- III. Oscillator Unit
- IV. Real Time Clock (RTC)
- V. Watchdog Timer



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

## 2. 6 Other System Components



I. Reset Circuit: It is used during the system ON:

### Components

- = Where the system **will get damaged** if the voltage level goes high.
- = This Reset signal brings the system into **Known state** i.e.
  - All the internal registers and different h/w components of the controller to a known state
  - **All the internal registers and different h/w components of the controller to a known state**
  - And then **starts the firmware execution** from the reset vector.
  - And then **starts the firmware execution** from the reset vector.
- The reset signal can be either:
  - Active high :→ The processor undergoes reset when the reset pin of the processor is at logic high
  - The reset signal can be either:
    - Active high :→ The processor undergoes reset when the reset pin of the processor is at logic high
    - Active low:→ The processor undergoes reset when the reset pin of the processor is at logic low.
- As this operation is synchronized i.e. controlled by clock, the reset signal should give clock oscillator sufficient time to perform the task.
  - Active low:→ The processor undergoes reset when the reset pin of the processor is at logic low.
- As this operation is **synchronized** i.e. controlled by

## 2. 6 Other System Components



### I. Reset Circuit Components

Circuit:

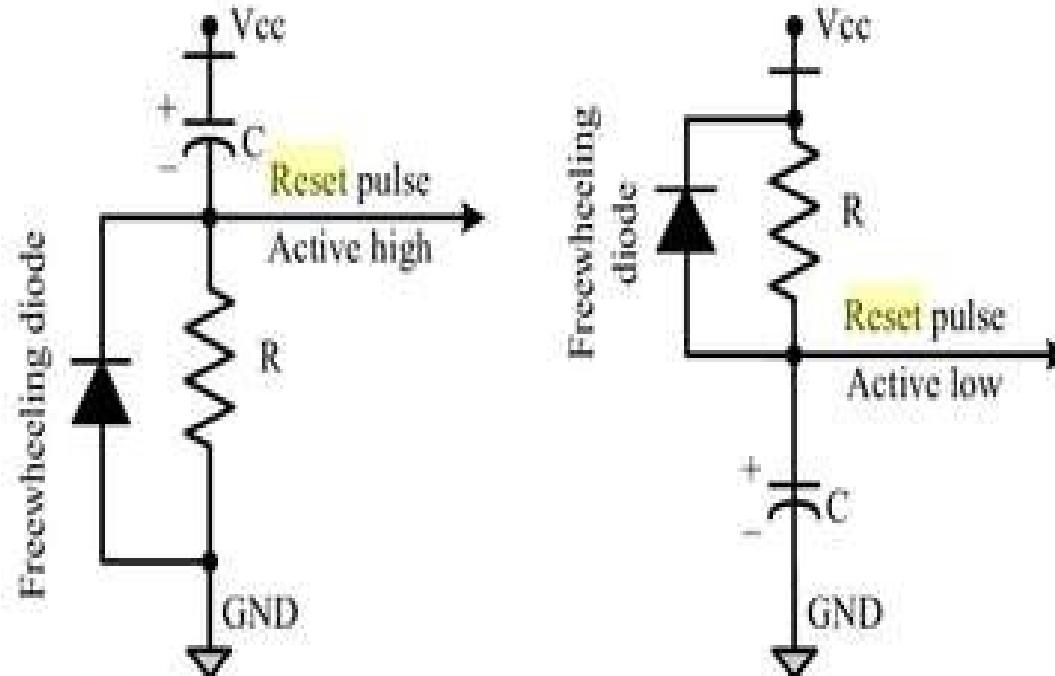


Fig. 2.35 RC based reset circuit



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

## 2. 6 Other System Components



### II. Brown-out Protection Circuit

The brown-out protection circuit **prevents** the processor/controller:

- From **unexpected program execution behavior** when the **supply voltage** to the processor/controller **falls** below a specified **(threshold) voltage**.
- This might lead to the **data corruption**.
- This might lead to the **data corruption**.
- So Brown-out Protection Circuit holds the processor/controller at **reset state** till the **voltage reaches to the threshold voltage**



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

# 2. 6 Other System Components



## II. Brown-out Protection Circuit

Heart of brown-out protection circuit

is Zener diode and Transistor : circuit

is Zener diode and Transistor :

• The transistor conducts only when:

• The transistor conducts only

when:  $V_{cc} > V_{BE} + V_Z$

• The transistor stops conduction

when:  $V_{cc} < V_{BE} + V_Z$

• The transistor stops conduction when:

$V_{cc} < V_{BE} + V_Z$ , Hence a Reset pulse

is sent till  $V_{cc}$  is greater.

Hence a Reset pulse is sent till  $V_{cc}$  is greater.

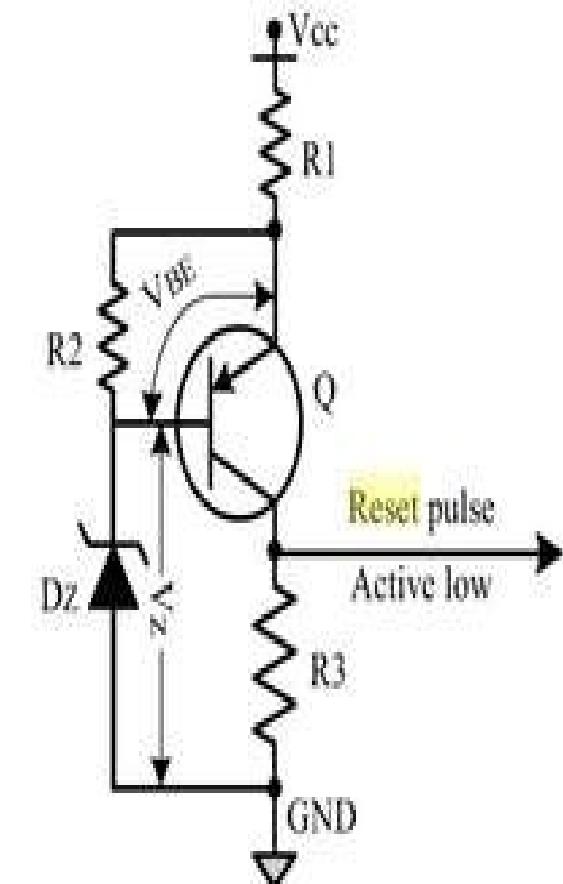


Fig. 2.36

Brown-out protection circuit with Active low output



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

## 2. 6 Other System Components



### III. Oscillator Unit Components

The Oscillator unit generates clock signals for synchronizing the operations of the processor.

Oscillator Unit can be:

- Present within processor/Controller where we require external quartz Crystal to supply clock.
- Present outside processor/Controller : Uses Quartz Crystal Oscillator are available in the form of Chips
- Present outside processor/Controller : Uses Quartz Crvstal

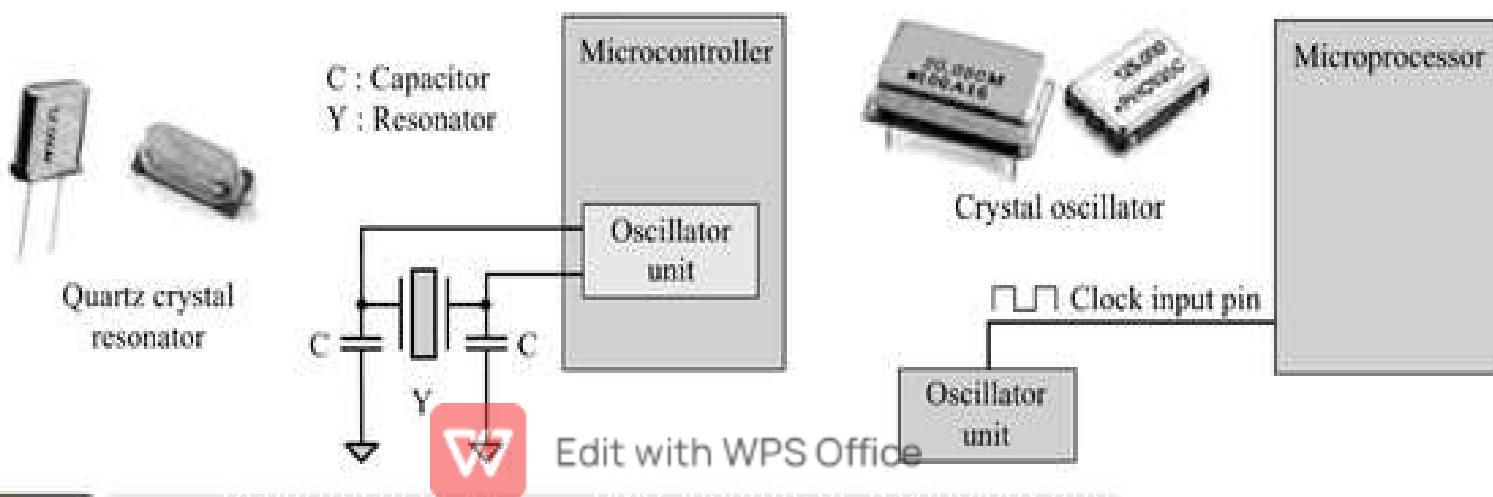


Fig. 2.37

Oscillator circuitry using quartz crystal and quartz crystal oscillator

## 2. 6 Other System Components



### IV. Real-Time Clock (RTC):

- Is a system component responsible for keeping track of time.
- RTC holds information like:
  - Current time (in hours, minutes and seconds) in 12 hour/24 hour format, and hours, minutes and seconds)
  - Date, month, year, day of the week, etc.
  - hour/24 hour format,
  - Supplies timing reference to the system.
- RTC is intended to function even in the absence of power.
  - Supplies timing reference to the system.
- RTC is intended to function even in the absence of power.
  - The RTC chip contains a microchip for holding the time and date related information.
  - Backup battery cell for functioning in the absence of power in a single IC package.
- Backup battery cell for functioning in the absence of power in a single IC package.

## 2. 6 Other System Components



### V. Watchdog Timer:

- Is to monitor the **firmware execution** and **reset the system processor/microcontroller**
  - When the **program execution hangs up** (or)
  - Generates an **Interrupt** in case the **execution time for a task is exceeding the maximum allowed limit.**
- If the **firmware execution** doesn't complete due to **malfunctioning**, within the **time required by the watchdog**
  - A **reset pulse** is generated and this will **reset the processor** (if it is connected to the **reset line of the processor**).



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

## 2. 6 Other System Components



### V. Watchdog Timer Components

- Most of the processors implement
  - Watchdog as a built-in component
    - Provides status register to control the watchdog timer
- If the processor/controller doesn't contain a built in watchdog timer,
- If the processor/controller doesn't contain a built in watchdog timer,
  - The same can be implemented using an external watchdog timer IC circuit.

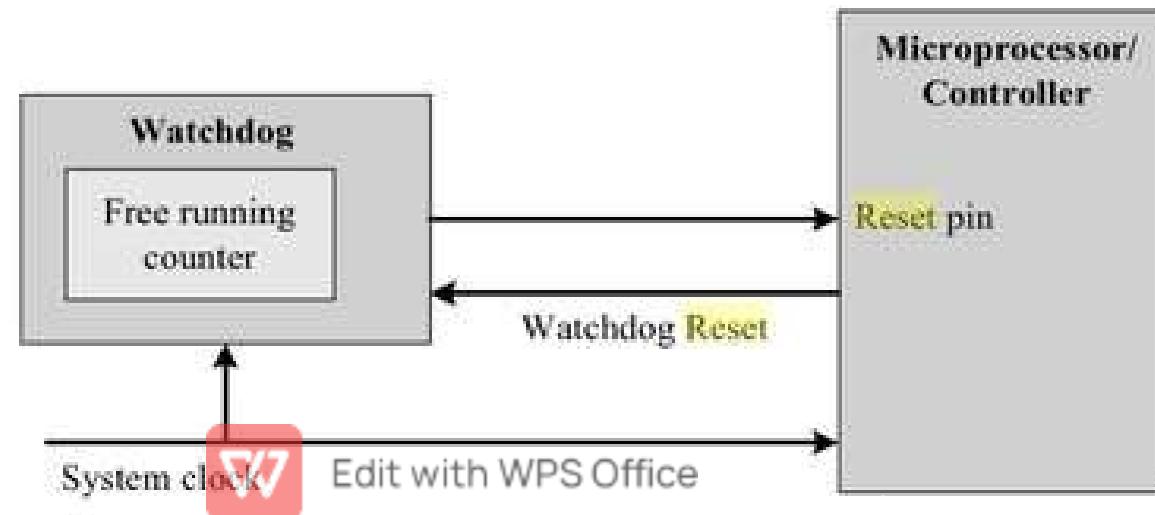


Fig. 2.38 Watchdog timer for firmware execution supervision



**Mrs. Ashwini N**  
Assistant Professor  
Dept.of.ISE  
BMSIT&M, Bengaluru  
**Email: ashwinilaxman@bmsit.in**

# Module – 4

## Embedded System Design Concepts

### (18CS44)



By,

**Mrs. Ashwini N** Assistant Professor  
Dept. of Information Science & Engg.  
BMS Institute of Technology, Bengaluru.

BMS Institute of Technology, Bengaluru.



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt



# Characteristics of an Embedded System

Unlike general purpose computing systems, embedded systems possess certain specific characteristics

These characteristics are unique to each embedded system.

Some of the important characteristics of an embedded system are:

Some of the important characteristics of an embedded system are:

1. Application and domain specific

2. Reactive and Real Time

3. Operates in harsh environments

4. Distributed

5. Small size and weight

6. Power concerns

5. Small size and weight

6. Power concerns



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt



# Characteristics: Application and Domain Specific

## Specific

- An embedded system is designed for a **specific** (application) purpose only.
  - It will **not do any other task**.
  - Ex. Air conditioner's embedded control unit, it cannot replace microwave oven...
  - Because the embedded control units of microwave oven and air conditioner are specifically designed to perform certain specific tasks.
- Certain embedded systems are **specific to a domain**:
  - Ex. A hearing aid is an application
  - Ex. Telecom with another control unit
    - belongs to the **domain of signal processing**
    - designed to serve another domain like consumer electronics.
  - Ex. Telecom with another control unit
    - belongs to the **domain of signal processing**
    - designed to serve another domain like consumer electronics.

# Characteristics : Reactive and Real Time



- Certain embedded systems are

- Designed to react to the events that occur in the near by environment.
  - These events also occur real-time.
  - These events also occur real-time.

- Example of Real time systems

- Flight control systems, Antilock Brake Systems (ABS), etc.

- Example of Reactive System.

- An air conditioner adjusts its mechanical parts as soon as it gets a signal from its sensors to increase or decrease the temperature when the user operates it using a remote control.

- An embedded system uses Sensors to take inputs and has actuators to bring out the required functionality.

to take inputs

and has actuators

to bring out the required functionality.



Edit with WPS Office

# Characteristics : Operates in harsh environments



- Certain embedded systems are designed to operate in harsh environments like:
  - A dusty one or a high temperature zone /
  - An area subject to vibrations and shock /
  - Very high temperature of the deserts /
  - Very low temperature of the mountains or extreme rains.
  - Very high temperature of the deserts
- These embedded systems have to be capable of
  - Sustaining the environmental conditions it is designed to operate in.



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

# Characteristics:

## Distributed



The term distributed means that embedded systems:

- May be a part of a larger system.
  - = Larger system → Consists of components that are independent of each other but have to work together
  - = Ex. An automatic vending machine is a typical example for this. It consists of:
    - A card reader (for pre-paid vending systems),
    - A vending unit, etc.
    - Each of them are independent embedded units but they work together to perform the overall vending function.
  - = Ex. Automatic Teller Machine (ATM) contains
    - A card reader embedded unit, responsible for reading and validating the user's ATM card,
    - Transaction unit for performing transactions,
    - Transaction unit for performing transactions,
    - A currency counter for dispatching/vending currency to the authorized person
    - A currency counter for dispatching/vending currency to the authorized person
    - Printer unit for printing the transaction details



# Characteristics : Small Size and Weight

An embedded system that is :

- **Compact in size and has light weight will be desirable or more popular than one that is bulky and heavy.**
- **popular than one that is bulky and heavy.**

Ex. Currently available cell phones. The cell phones that have the maximum features are popular

Ex. Currently available cell phones. The cell phones that have the maximum features are popular

- **But also their size and weight is an important characteristic.**
- **maximum features are popular**
- **But also their size and weight is an important characteristic.**



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt



# Characteristics :Power concerns

It is desirable that the power utilization and heat dissipation of any embedded system should be low.

any embedded system should be low.

- If more heat is dissipated then additional units like
  - Heat sinks or cooling fans need to be added to the circuit.
- Additional units → In turn occupies additional space and make the system bulky.
- Nowadays ultra low power components are available in the market.
- Nowadays ultra low power components are available in the market.

Also power management is a critical constraint in battery operated application.

- The more the power consumption the less is the battery life.
- The more the power consumption the less is the battery life.



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt



# Quality Attributes of an Embedded System

- Quality attributes are the :

- Non-functional requirements that need to be documented properly in any system design.
- If the quality attributes are more concrete and measurable,
- If the quality attributes are more concrete and measurable,
  - It will give a positive impact on the system development process and the end product.
- It will give a positive impact on the system development process and the end product.
- The quality attributes are broadly classified into two, namely
- The quality attributes are broadly classified into two, namely
  - i. Operational Quality Attributes
  - ii. Non-Operational Quality Attributes
  - i. Operational Quality Attributes
  - ii. Non-Operational Quality Attributes



Edit with WPS Office



# Operational Quality Attributes

## Attributes

### The operational quality attributes

= Represents attributes in the **operational mode** or **'online'** mode.

### The important quality attributes coming under this category are listed below:

listed below:

1. Response

1. Response

2. Throughput

2. Throughput

3. Reliability

3. Reliability

4. Maintainability

4. Maintainability

5. Security

5. Security

6. Safety



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt



# Operational Quality Attributes

## Attributes

1. **Response:** Is a measure of **quickness** of the system. It gives you an idea about how fast your system is tracking the

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

• **Most of the embedded system demand fast response which should be real-time.**  
Ex: An embedded system deployed in flight control application  
: Any response delay → creates potential damages to the safety of the flight as well as the passengers.

• **Ex. An embedded system deployed in flight control application**  
It is not necessary that all embedded systems should be Real Time in response.

• **Any response delay → creates potential damages to the safety of the flight as well as the passengers.**  
Ex: The response time requirement for an electronic toy is not at all time-critical.

• **It is not necessary that all embedded systems should be Real Time in response.**



# Operational Quality Attributes

## Attributes

2. Throughput: Throughput deals with the efficiency of system.

It can be defined as rate of production or process of a defined process

- It can be defined as **rate of production or process** of a defined process
- The **rates** can be expressed in terms of : over a **stated period of time**.
  - Units of products, batches produced, or any other meaningful measurements.

• **Units** of products, batches produced, or any other meaningful measurements.

• In case of **card reader** like the ones used in **buses**,

• throughput means: → how much **transactions** the Reader can perform in a minute or hour or day.

• Throughput is generally measured in terms of '**Benchmark**'

• → A '**Benchmark**' is a **reference point** by which something can be measured.

• Benchmark can be a **set of performance criteria** that a product is expected to meet or a **standard product** that can be used for comparing other products of the same



# Operational Quality Attributes

## 3. Reliability: Reliability is a measure of:

- How much percentage you rely upon the proper functioning of the system? (/, or)
- What is the % susceptibility of the system to failure?
- What is the % susceptibility of the system to failure?

The terms used in defining system reliability are:

1. Mean Time Between Failures (MTBF)

1. Mean Time Between Failures (MTBF)

- Gives the frequency of failures in hours/weeks/months

2. Mean Time To Repair (MTTR)

2. Mean Time To Repair (MTTR)

- Specifies how long the system is allowed to be out of order following a failure
- Specifies how long the system is allowed to be out of order following a failure

For an embedded system with critical application need → it should be of the order of minutes

For an embedded system with critical application need → it should be of the order of minutes



Edit with WPS Office



# Operational Quality Attributes

## Attributes

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

• Reliability and maintainability are considered as two complementary disciplines.

- A more reliable system means a system with less corrective maintainability requirements and vice versa
- A more reliable system means a system with less corrective requirements

• Maintainability can be classified into two types:

1. **Maintainability can be classified into two types:**

1. **Scheduled or Periodic Maintenance (Preventive Maintenance)**:  
→ An inkjet printer uses ink cartridges → as per the printer manufacturer the end user should replace the cartridge after each 'n' number of printouts to get quality prints.

2. **Maintenance to Unexpected Failures (Corrective Maintenance)** → If the paper feeding part of the printer fails the printer fails to print and it requires immediate repairs to rectify this problem.

2. **Maintenance to Unexpected Failures (Corrective Maintenance)**  
→ If the paper feeding part of the printer fails the printer fails to print and it requires immediate repairs to rectify this problem.



# Operational Quality Attributes

## Attributes

### 4. Maintainability:

- Hence it is obvious that maintainability is simply an indication of the availability of the product for use.
- In any embedded system design, the ideal value for availability is expressed as:  
availability is expressed as:

$$Ai = MTBF / (MTBF + MTTR)$$

Where

$$Ai = \frac{MTBF}{MTBF + MTTR}$$

MTBF=Mean Time Between Failures

Where

**Ai**=Availability in the ideal condition

**MTBF**=Mean Time Between Failures

**MTTR**= Mean Time To Repair



Edit with WPS Office



# Operational Quality Attributes

## 5. Security

Three major measures of information security:

1. 'Confidentiality' → Deals with the protection of data and application from unauthorized disclosure and application from unauthorized disclosure
2. 'Integrity' → Deals with the protection of data and application from unauthorized modification
3. 'Availability' → Deals with protection of data and application from unauthorized users

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

**Certain embedded systems have to make sure they conform to the security measures.**

Ex. An **electronic safety Deposit Locker** can be used only with a pin



# Operational Quality Attributes

## 6. Safety Attributes

6. Safety: Safety deals with the :

- Possible damages that can happen to the operators, public and the environment
- Due to the breakdown of an embedded system ( or )
- Due to the breakdown of an embedded system ( or )
- Due to the emission of radioactive or hazardous materials from the embedded products.
- The breakdown of an embedded system may occur due to a hardware failure or a firmware failure
- The breakdown of an embedded system
- Safety analysis is a must in product engineering to evaluate:
  - The anticipated damages occur due to a hardware failure or a firmware failure.
  - Determine the best course of action to bring down the consequences of the damages to an acceptable level
- Safety analysis is a must in product engineering to evaluate:
  - The anticipated damages
  - Determine the best course of action to bring down the consequences of

# Non-Operational Quality Attributes



## Attributes

- The quality attributes that needs to be addressed for the product 'not' on the basic of operational aspects are grouped under this category.
- The important quality attributes coming under this category are listed below:
- The important quality attributes coming under this category are listed below:
  1. Testability & Debug-ability
  2. Evolvability
  3. Portability
  4. Time to prototype and market
  5. Per unit and total cost

# Non-Operational Quality Attributes



## Attributes

### 1. Testability & Debug-ability:

- Testability deals with many ways:
  - How easily one can test his/her design, application ?
  - Which means he/she can test it?
- For an embedded product, testability is applicable to both the **embedded hardware and firmware**.
- Debug-ability is a means of debugging the product as such for figuring out the probable sources that create unexpected behavior in the total system.



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

# Non-Operational Quality Attributes



## Attributes

### 1. Testability & Debug-ability:

- Debug-ability has two aspects in the embedded system development context, namely:
  - Hardware level debugging
    - Hardware debugging is used for figuring out the issues created by hardware problems
  - Firmware level debugging
    - Firmware debugging is employed to figure out the probable errors that appear as a result of flaws in the firmware.



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

# Non-Operational Quality Attributes



## Attributes

### 2. Evolvability:

- Evolvability is a term which is closely related to Biology.
- Evolvability is referred as the non-heritable variation.
- For an embedded system, the quality attribute 'Evolvability' refers to:
  - To the ease with which the embedded product (including firmware and hardware) can be modified to take advantage of new firmware or hardware technologies.
- For an embedded system, the quality attribute 'Evolvability' refers to:
  - To the ease with which the embedded product (including firmware and hardware) can be modified to take advantage of new firmware or hardware technologies.

'Evolvability' refers :

- To the ease with which the embedded product (including firmware and hardware) can be modified to take advantage of new firmware or hardware technologies.

# Non-Operational Quality Attributes



## Attributes

### 3. Portability:

- Portability is a measure of 'system independence'.
- An embedded product can be called portable if it is:
- An embedded product can be called portable if it is:
  - Capable of functioning in various environments, target processors/controllers and embedded operating systems.
- A standard embedded product should always be flexible and portable.
- A standard embedded product should always be flexible and portable.



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

# Non-Operational Quality Attributes



## Attributes

### 4. Time-to-Prototype and Market:

- Time-to-market is the time elapsed between:
  - The conceptualization of a product.
  - The time at which the product is ready for selling (for commercial products) or use (for non-commercial products).
- The commercial embedded product market is highly competitive and time to market the product is a critical factor in the success of a commercial embedded product.
- Product prototyping helps a lot in reducing time-to-market.



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

# Non-Operational Quality Attributes



## Attributes

### 5. Per unit and total cost

- Cost is a factor which is closely monitored by both:
  - End user (those who buy the product)
  - Product manufacturer (those who build the product).
- Cost is a highly sensitive factor for commercial products.
- Cost is a highly sensitive factor for commercial products.
- Proper market study and cost benefit analysis should be carried out before taking decision on the per unit cost of the embedded product.
- Proper market study and cost benefit analysis should be carried out before taking decision on the per unit cost of the embedded product.



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

# Non-Operational Quality Attributes



## Attributes

### 5. Per unit and total cost

- When the product is introduced in the market, for the initial period the sales and revenue will be low.
- There won't be much competition when the product sales and revenue increase.
- During the maturing phase, the growth will be steady and revenue reaches highest point and at retirement time there will be a drop in sales volume.
- During the maturing phase, the growth will be steady and revenue reaches highest point and at retirement time there will be a drop in sales volume.



# Washing Machine

## Application Specific Embedded System

### System

# Washing Machine Application Specific Embedded System



- Washing Machine is a typical example of an **embedded system** providing extensive support in home automation applications.
- An embedded system contains **sensors**, **actuators**, **control unit** and **application-specific user interfaces** like keyboards, display units, etc. You can see all these components in a washing machine if you have a closer look at it. Some of them are **visible** and some of them may be **invisible** to you.



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

# Washing Machine

## Application Specific Embedded System

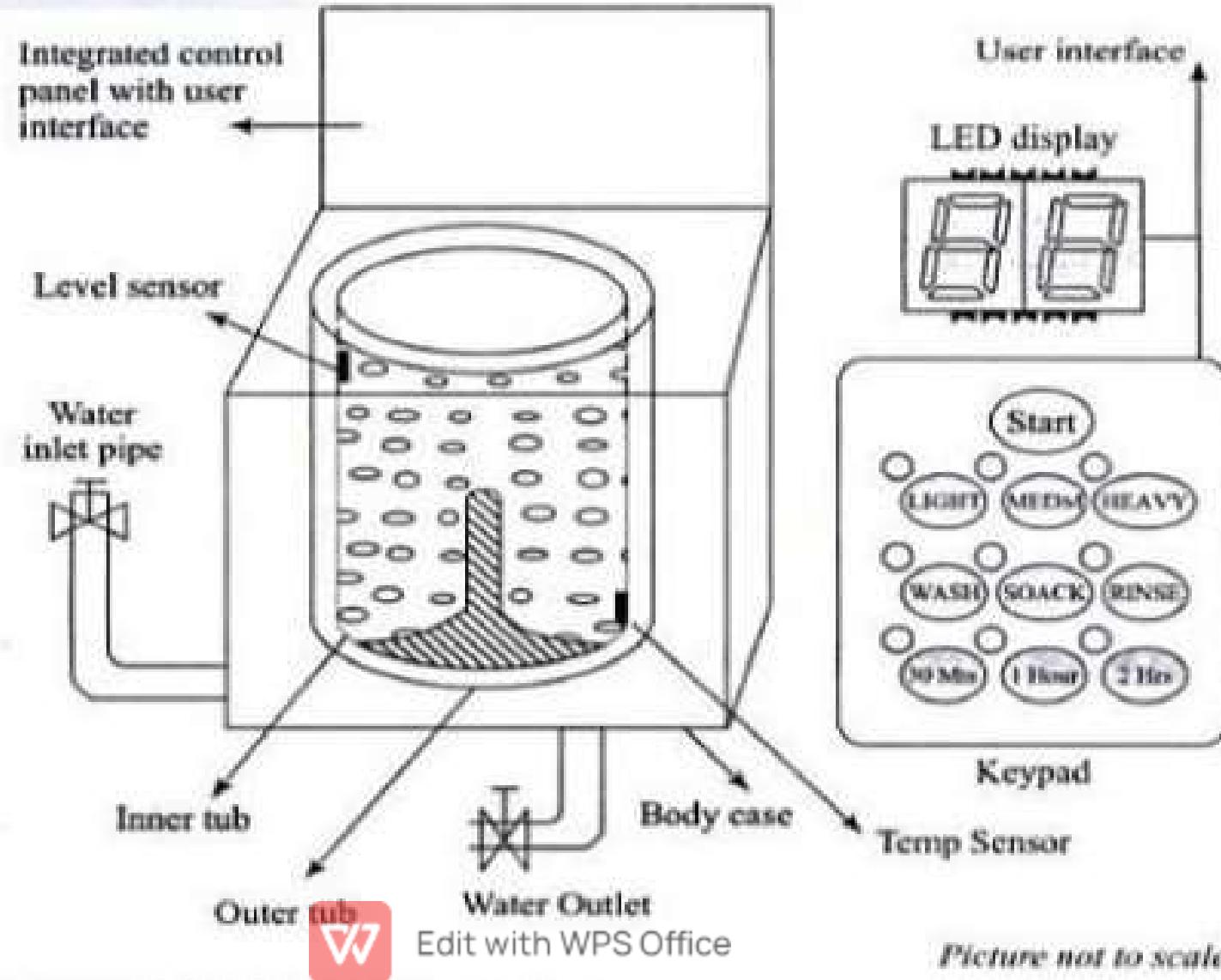


Fig. 4.2 Washing machine – Functional block diagram

# Washing Machine

## Application Specific Embedded System



- **Actuator part:** → Motorized agitator, tumble tub, water drawing pump and inlet valve to control the flow of water into the unit.
- **Sensor part:** → water temperature sensor, level sensor, etc.
- **Control part:** → microprocessor/controller based board with interfaces to the sensors and actuators.
- **Sensor data:** → Is fed back to the control unit and the control unit generates the necessary actuator outputs.
- **Sensor data:** → Is fed back to the control unit and the control unit generates the necessary actuator outputs.



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

# Washing Machine

## Application Specific Embedded System



- **Control unit:** → Provides connectivity to user interfaces like keypad for setting the washing time, selecting the type of material to be washed like light, medium, heavy duty, etc.
- **User feedback:** → Is reflected through the display unit and LEDs connected to the control board.
- **Integrated control panel:** → microprocessor/controller based board with I/O interfaces and a control algorithm running in it.
- **Integrated control panel:** → microprocessor/controller based board with I/O interfaces and a control algorithm running in it.



Edit with WPS Office

# Washing Machine

## Application Specific Embedded System



- **Input interface:** → Includes the keyboard which consists of wash type selector namely Wash, Spin and Rinse, cloth type selector namely Light, Medium, Heavy duty and washing time setting, etc.
- **Output Interface:** → LED/LCD displays, status indication LEDs, etc. connected to the I/O bus of the controller.
- **Output interface:** → LED/LCD displays, status indication LEDs, etc. connected to the I/O bus of the controller.
- It is to be noted that this interface may vary from manufacturer to manufacturer and model to model.
- It is to be noted that this interface **may vary** from manufacturer to manufacturer and model to model.



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt



# Automotive (Examples)

## (Examples)

# Domain Specific Embedded System

# Domain Specific Embedded System



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

# Automotive Domain Specific Embedded System



The major application domains of embedded domain  
of embedded system

are:

- Consumer
- Industrial
- Automotive
- Automotive
- Telecom, etc.
- Telecom, etc.



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

# Automotive Domain Specific Embedded System



Fig. 4.3

Embedded systems in the automotive domain  
(Photo courtesy of Honda Siel Car India ([www.hondacarindia.com](http://www.hondacarindia.com)))



Edit with WPS Office

# Inner Workings of Automotive Embedded Systems



- Automotive embedded systems are the one where **electronics** take control over the mechanical systems.
- The presence of automotive embedded system in a vehicle varies from simple mirror and wiper controls to complex air bag controller and antilock brake systems (ABS).
- Automotive embedded systems are normally built around microcontrollers or DSPs or a hybrid of the two and are generally known as Electronic Control Units (ECUs).
- Automotive embedded systems are the one where **electronics** take control over the mechanical systems.
- The presence of automotive embedded system in a vehicle varies from simple mirror and wiper controls to complex air bag controller and antilock brake systems (ABS).
- Automotive embedded systems are normally built around microcontrollers or DSPs or a hybrid of the two and are generally known as Electronic Control Units (ECUs).



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

# Inner Workings of Automotive Embedded Systems



- The various types of electronic control units (ECUs) are:
  1. High speed Electronic Control Units (HECUs) :
    - = They are deployed in **critical control units** requiring **fast response**, like fuel injection systems, antilock brake systems, etc.
  2. Low speed Electronic Control Units (LECUs) :
    - = They are deployed in applications where **response time** is **not so critical**. They are generally built around **low cost microprocessors/microcontrollers** and **digital signal processors**. **Audio controllers, passenger and driver door locks, door glass controls, etc.**, are examples for LECUs.



# Automotive Communication Buses

Automotive applications use: **serial buses** for communication.

Few important automotive communication buses are:

- **Controller Area Network (CAN):**
  - Event driven **serial** protocol interface with support for **error handling** in data transmission.
  - It is generally employed in safety system like airbag control,
  - It is generally employed in powertrain systems like engine control and Antilock Brake Systems **safety** system
- **Local Interconnect Network (LIN):**
  - It is a single master multiple slave (up to 16 independent slave nodes) communication interface.
- **Local Interconnect Network (LIN):**
  - It is a **single master multiple slave** (up to 16 independent slave nodes) communication interface.
  - LIN is a **low speed, single wire** communication interface with



# Automotive Communication Buses

Few important automotive communication buses are:

- Media Oriented System Transport (MOST)
  - = MOST bus is targeted for automotive audio video equipment interfacing.
  - = MOST bus is a multimedia fiber-optic point-to-point network implemented in a star, ring or daisy chained topology over optical fibers cables.



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

# Key Players of Automotive Embedded Market



Key players can be visualized in 3 verticals:

1. **Silicon Providers** : → Providers of necessary chips

- Analog Devices, Xilinx, NXP Semiconductor, Texas Instruments, Microchip etc.
- Texas Instruments, Semiconductor, Texas Instruments, Microchip etc.

2. **Tools and Platform Providers**: → Suppliers of Tools and OS required for Embedded system

2. **Tools and Platform Providers**: → Suppliers

- The MathWorks, Keil Software, Microsoft etc.

3. **Solution Providers**: → Providers of complete solution for automotive applications

- Bosch Automotive, Infosys Technologies, DENSO Automotive etc.

3. **Solution Providers**: → Providers of complete solution for automotive applications

# Hardware Software Co-Design and Program Modelling(Ch -7→7.1,7.2)



Edit with WPS Office

# Hardware Software Co-Design

## Design

Before going to the Hardware Software Co-Design we have known:

e know:  
Traditional Embedded System Development Approach:

**Traditional Embedded System Development Approach:**

- The hardware software partitioning is done at an early stage
- Engineers from the software group take care of the software architecture development and implementation
- Engineers from the software group take care of the software architecture development and implementation
- Engineers from the hardware group are responsible for building the hardware required for the product
- There is less interaction between the two teams and the development happens either serially or in parallel and once the hardware and software are ready, the integration is performed
- Engineers from the hardware group are responsible for building the hardware required for the product
- There is less interaction between the two teams and

# Hardware Software Co-Design

## Design

### Hardware Software Co-design Approach for Embedded System Development

#### Development

- The product requirements captured from the customer are converted into system level needs or processing requirements rather than partitioning them to either h/w or s/w
- The system level processing requirements are then transferred into functions which can be simulated and verified against performance and functionality
- The Architecture design follows the system design. The partition of system level processing requirements into hardware and software takes place during the this phase
- Each system level processing requirement is mapped as either hardware and/or software requirement
- The partitioning is performed based on the hardware-software trade-offs
- The partitioning is performed based on the hardware-software trade-offs

# Fundamental Issues in Hardware Software Co-Design

Fundamental Issues of H/w, S/w Co-Design: Few are listed below:

below:

1. Selecting the model
2. Selecting the Architecture
3. Selecting the Language
4. Partitioning system requirements into hardware and software
4. Partitioning system requirements into hardware and software



Edit with WPS Office

# Fundamental Issues in Hardware Software Co-Design

Fundamental Issues of H/w, S/w Co-Design: Few are listed below:

## 1. Selecting the model:

### 1. Selecting the model:

- Models are used for capturing and describing the system characteristics
- Models are used for capturing and describing the system characteristics
- A model is a formal system consisting of objects and composition rules
- A model is a formal system consisting of objects and composition rules
- It is hard to make a decision on which model should be followed in a particular system design.
- It is hard to make a decision on which model should be followed in a particular system design.
- Most often designers switch between a variety of models from the requirements specification to the implementation aspect of the system design.
- Most often designers switch between a variety of models from the requirements specification to the implementation aspect of the system design.
- Most often designers switch between a variety of models from the requirements specification to the implementation aspect of the system design.

# Fundamental Issues in Hardware Software Co-Design

Fundamental Issues of H/w, S/w Co-Design: Few are listed below:

2. Selecting the Architecture:

## 2. Selecting the Architecture:

- A model only captures the system characteristics
  - Does not provide information on how the system can be manufactured?
- The architecture specifies:
- The architecture specifies:
  - how a system is going to implement in terms of the number and types of different components and the interconnection among them.



Edit with WPS Office

# Fundamental Issues in Hardware Software Co-Design



Fundamental Issues of H/w, S/w Co-Design: Few are listed below:

2. Selecting the Architecture:

## 2. Selecting the Architecture:

• Few of the commonly used architecture are:

• Few of the commonly used architecture are:

– Controller architecture

– Finite State Machine Model

– Datapath Architecture

– Data Flow Graph

– Complex Instruction Set Computing (CISC)

– Reduced Instruction Set Computing (RISC)

– Very long Instruction Word (VLIW)

– Implements ALUs, Multipliers etc.

– Parallel Processing architecture

– Single Instruction Multiple Data (SIMD), Multiple Instruction Multiple Data (MIMD) etc.

– Single Instruction Multiple Data (SIMD), Multiple Instruction Multiple Data (MIMD) etc.



# Fundamental Issues in Hardware Software Co-Design

Fundamental Issues of H/w, S/w Co-Design: Few are listed below:

3. Selecting the Language:

## 3. Selecting the Language:

- A programming Language captures:
  - Computational Model and maps it into architecture
- A computational model can be captured:
  - Multiple programming languages like C, C++, C#, Java etc. for software implementations
  - Languages like VHDL, System C, Verilog etc. for hardware implementations
- The only pre-requisite in selecting a programming language for capturing a model is that → the language should capture the model easily
  - Languages like VHDL, System C, Verilog etc.



# Fundamental Issues in Hardware Software Co-Design

Fundamental Issues of H/w, S/w Co-Design: Few are listed below:

4. Partitioning system requirements into hardware and software:

## 4. Partitioning system requirements into hardware and software:

- Deals with the implementation aspect of a System level Requirement
- It may be possible to implement the system requirements in either hardware or software (firmware)
- It may be possible to implement the system requirements in either hardware or software (firmware)
- Various hardware software trade-offs like performance, re-usability, effort etc. are used for making a decision on the hardware-software partitioning



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

# Hardware Software Co-Design: → Computational Models in ES



Few of the computational models used in Embedded system are listed below:

listed below:

1. Data Flow Graph / Diagram (DFG) Model
1. Data Flow Graph / Diagram (**DFG**) Model
2. Control Data Flow Graph / Diagram (CDFG) Model
2. Control Data Flow Graph / Diagram (**CDFG**) Model
3. State Machine Model
3. State Machine Model
4. Sequential Program Model
4. Sequential Program Model
5. Concurrent / Communicating Process Model
5. Concurrent / Communicating Process Model
6. Object-Oriented Model
6. Object-Oriented Model



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

# Hardware Software Co-Design: → Computational Models in ES

## 1. Data Flow Graph / Diagram (DFG) Model:

- Translates the data processing requirements into a data flow graph
- A data driven model in which the program execution is determined by data.
- Emphasizes on the data and operations on the data which transforms the input data to output data.

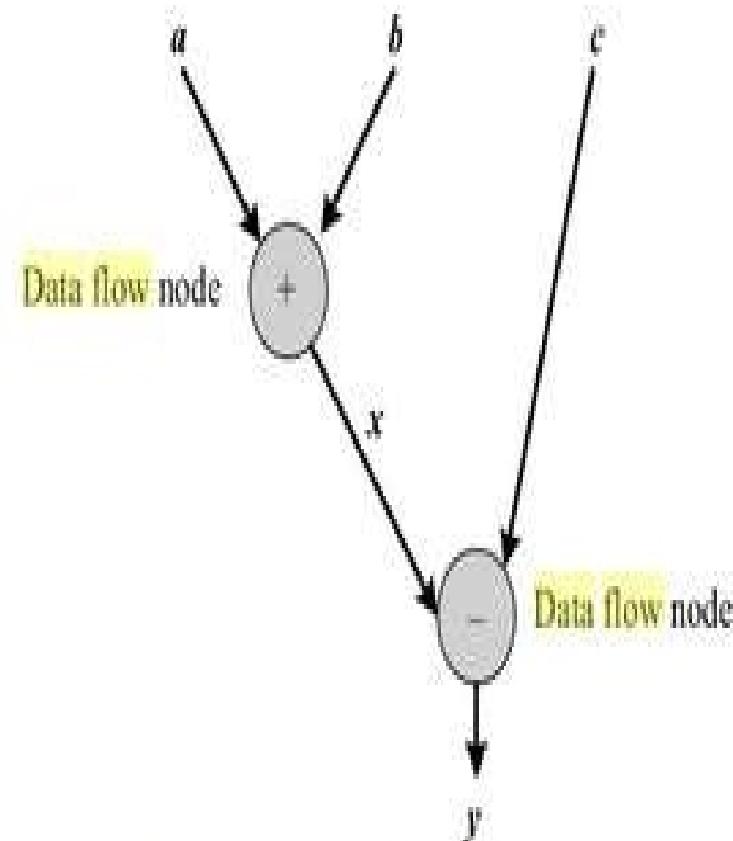


Fig. 7.1 Data flow graph (DFG) model

# Hardware Software Co-Design: → Computational Models in ES

## 1. Data Flow Graph / Diagram (DFG)

Model :→ Consist of

- Process: → The operation on the data,   
• Process: → The operation (like) on the data, represented using a block (circle)
- Data flow: → Is represented using arrows.   
– An inward arrow to the process (circle) represents input data
- Data flow: → Is represented using arrows (circle) represents input data
  - An inward arrow to the process (circle) represents input data
  - An outward arrow from the process (circle) represents output data
- Best suited for modeling Embedded systems which are computation intensive (like DSP applications) intensive (like DSP applications)

E.g. Model the requirement  
x = a + b; and y = x - c;

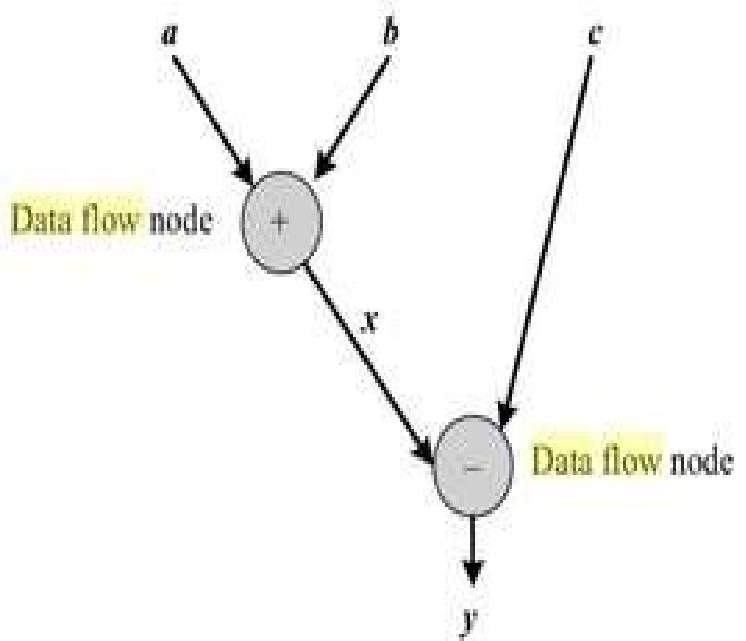


Fig. 7.1 Data flow graph (DFG) model

# Hardware Software Co-Design: → Computational Models in ES

## 1. Data Flow Graph / Diagram (DFG) Model : → Consist of

**Data path:** The data flow path from input to output

- A DFG model is said to be **acyclic DFG (ADFG)**:
  - If it doesn't contain:
    - Multiple values for the input variable
    - Multiple output values for a given set of input(s).
- A DFG model is said to be **non-acyclic DFG** :
  - Feedback inputs (Output is feed back to Input), events etc are examples for non-acyclic inputs.
- A DFG model translates the program as a **single sequential process execution**.



Edit with WPS Office

# Hardware Software Co-Design: → Computational Models in ES

## 2. Control Data Flow Graph / Diagram (CDFG) (Model)

Model:

- It is similar to DFG model but only difference is: → Contains both data operations and control operations
- Model used for: → conditional program execution
- Model used for: → conditional program
- Construction is same as DFG in addition: → it has conditional (constructs) as decision makers known as decision nodes.
- Construction is same as DFG in addition: → it has conditional (constructs) as decision makers known as decision nodes.
- The control node is represented by a decision diamond block which is the decision making element in a normal flow chart based design

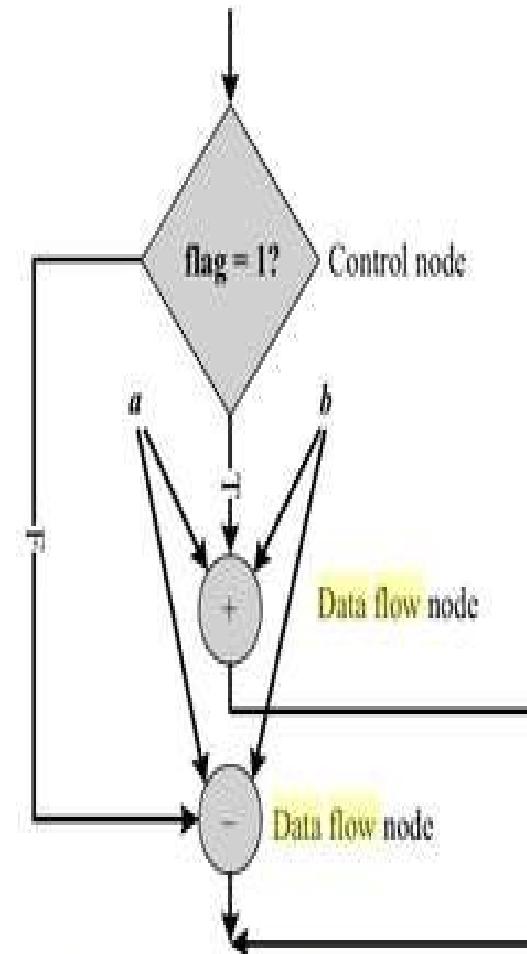


Fig. 7.2 Control Data Flow Graph (CDFG) Model

# Hardware Software Co-Design: → Computational Models in ES

## 2. Control Data Flow Graph / Diagram (CDFG) Model :

• Translates the requirement, which is modeled to a concurrent process model

- The decision on which process is to be executed is determined by the control node
- Example(Application): Digital Camera
  - Capturing of image
  - Storing it in the format selected (bmp, jpg, tiff, etc.)
  - Storing it in the format selected (bmp, jpg, tiff, etc.)

E.g. Model the requirement  
flag = 1?  
if flag = 1 then  
y = a + b;  
else  
y = a - b;  
end if

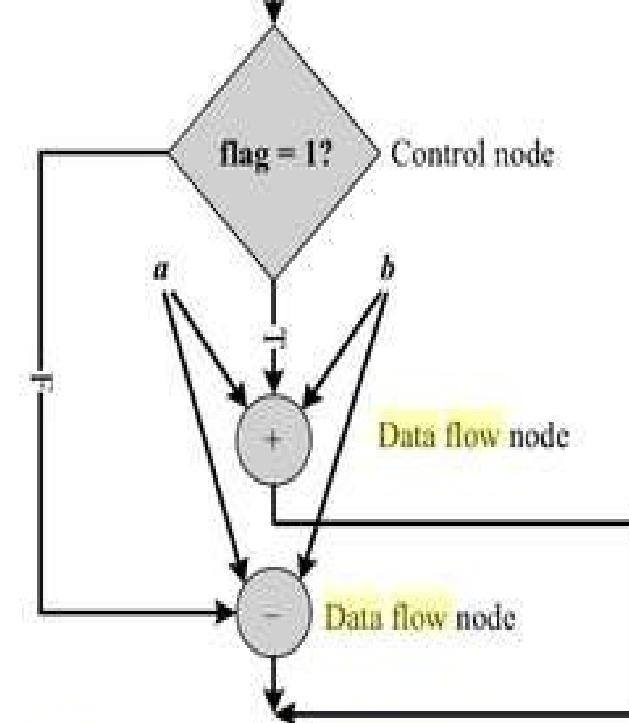


Fig. 7.2 Control Data Flow Graph (CDFG) Model

# Hardware Software Co-Design: → Computational Models in ES

## 3. State Machine Model :

- Based on 'States' and 'State Transition'
- Describes the system behavior with:
  - States
    - Representation of a current situation.
  - Events
    - Input to the state
    - It acts as stimuli for state transition.
  - Actions
    - Action is an activity to be performed by the state machine.
    - the state machine.
  - Transitions
    - Is the movement from one state to another. the movement from one state to another.

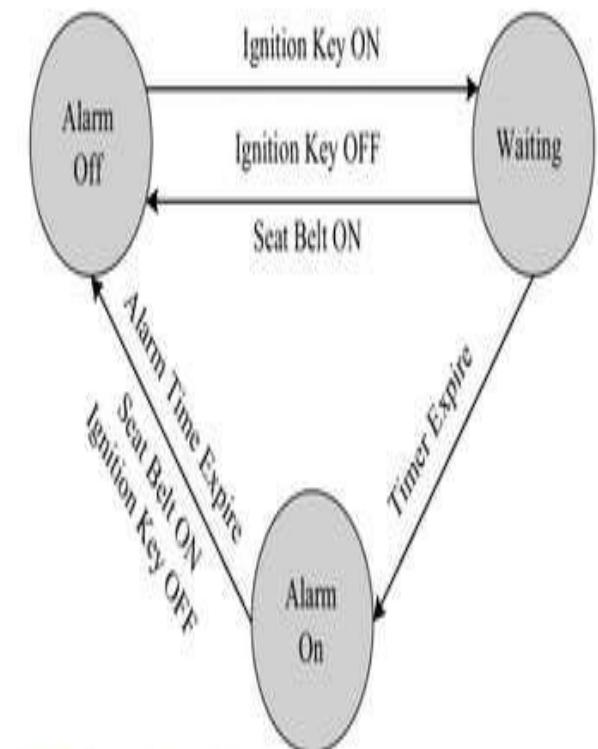


Fig. 7.3 FSM Model for Automatic seat belt warning system



# Hardware Software Co-Design: → Computational Models in ES

## 3. State Machine

Model A Finite State Machine (FSM) Model is one in which the number of states are finite. In other words the system is described using a finite number of possible states.

- E.g. Automatic 'Seat Belt Warning' in an automotive

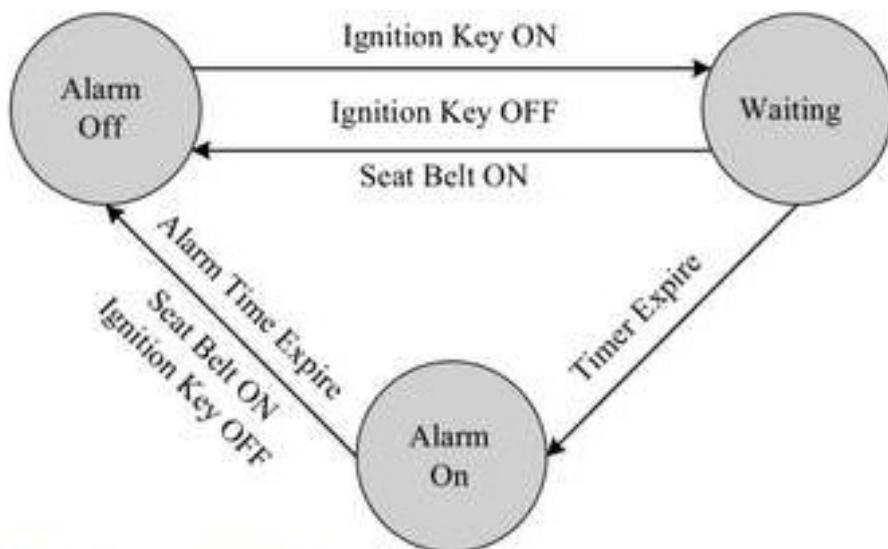


Fig. 7.3 FSM Model for Automatic seat belt warning system

- When the vehicle ignition is turned on:
  - The seat belt is not fastened within 10 seconds of ignition ON
  - The system generates an alarm signal for 5 seconds.
- The Alarm is turned off :
  - The alarm time (5 seconds) expires
- The Alarm is turned off :
  - If the alarm pastime fasten seconds) expires ignition switch is turned off, whichever happens first.



# Hardware Software Co-Design: → Computational Models in ES

## 3. State Machine Model :

- E.g. Model of Timer

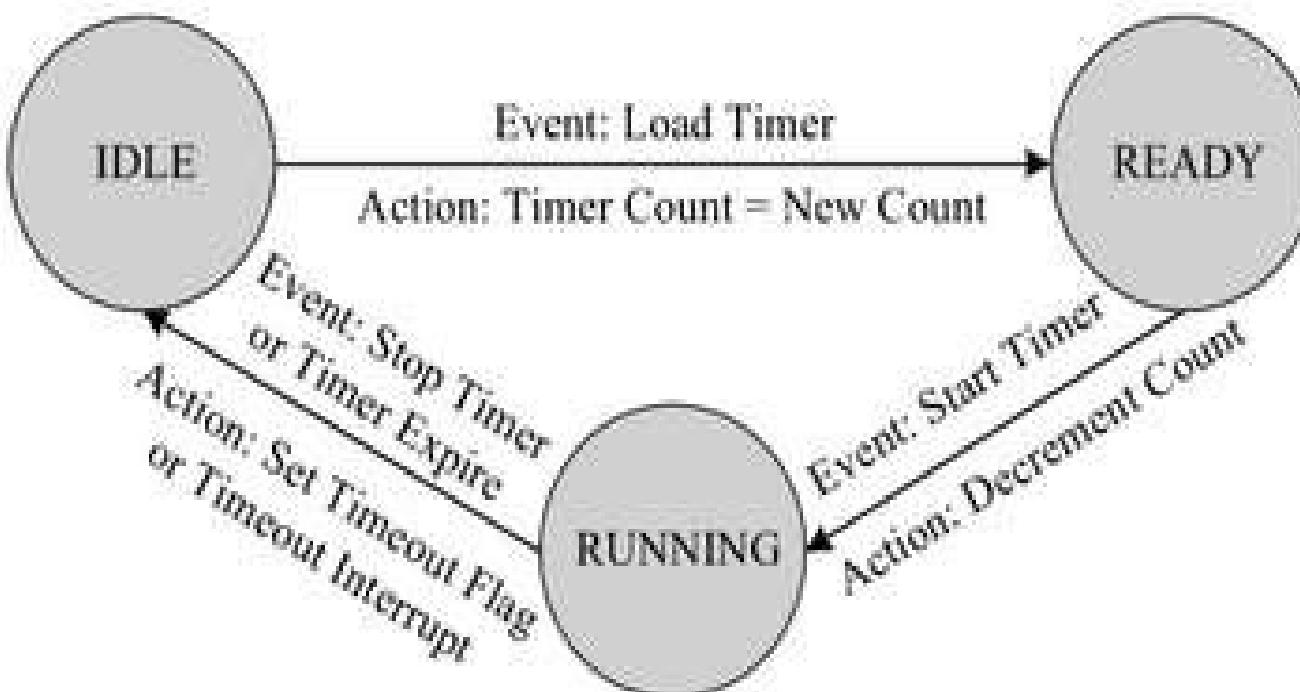


Fig. 7.4 FSM Model for timer



Edit with WPS Office

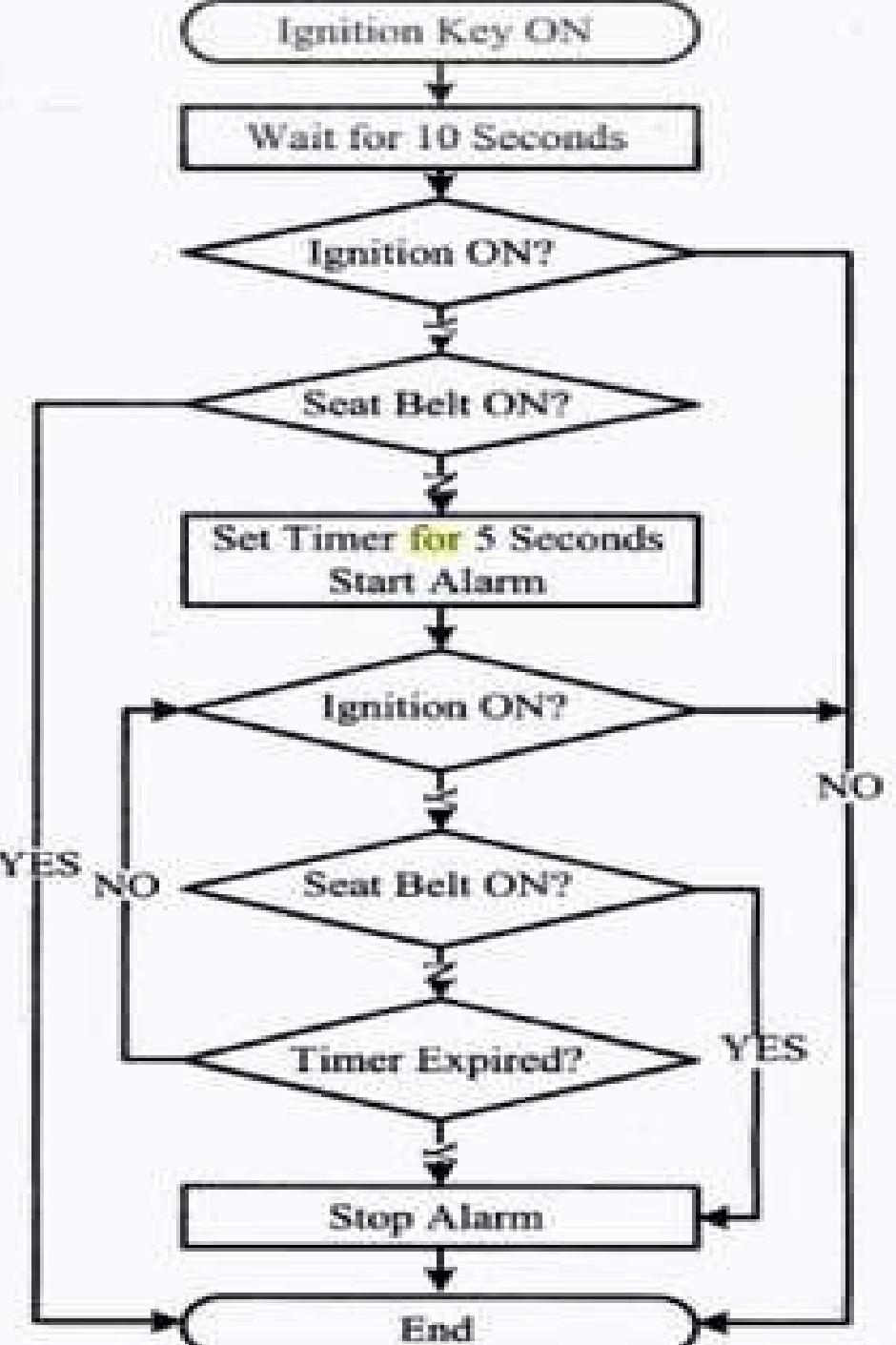
# Hardware Software Co-Design: → Computational Models in ES

## 3. Sequential Program Model:

- The functions or processing requirements are executed in sequence
- The program instructions are iterated and executed
- The program instructions are iterated and executed conditionally and the data gets transformed through a series of operations
- FSMs are good choice for sequential Program modeling.
- **FSMs** are good choice for sequential Program modeling.
- **Flow Charts** is another important tool used for modeling sequential program
- The FSM approach represents the states, events, transitions and actions, whereas the Flow Chart models the execution flow
- The FSM approach represents the **states, events, transitions and actions**, whereas the Flow Chart models the execution flow



# Sequential Program Model: E.g. **Automatic 'Seat Belt Warning'** in an automotive



# Sequential Program Model:

## E.g. Automatic 'Seat Belt Warning' in an automotive

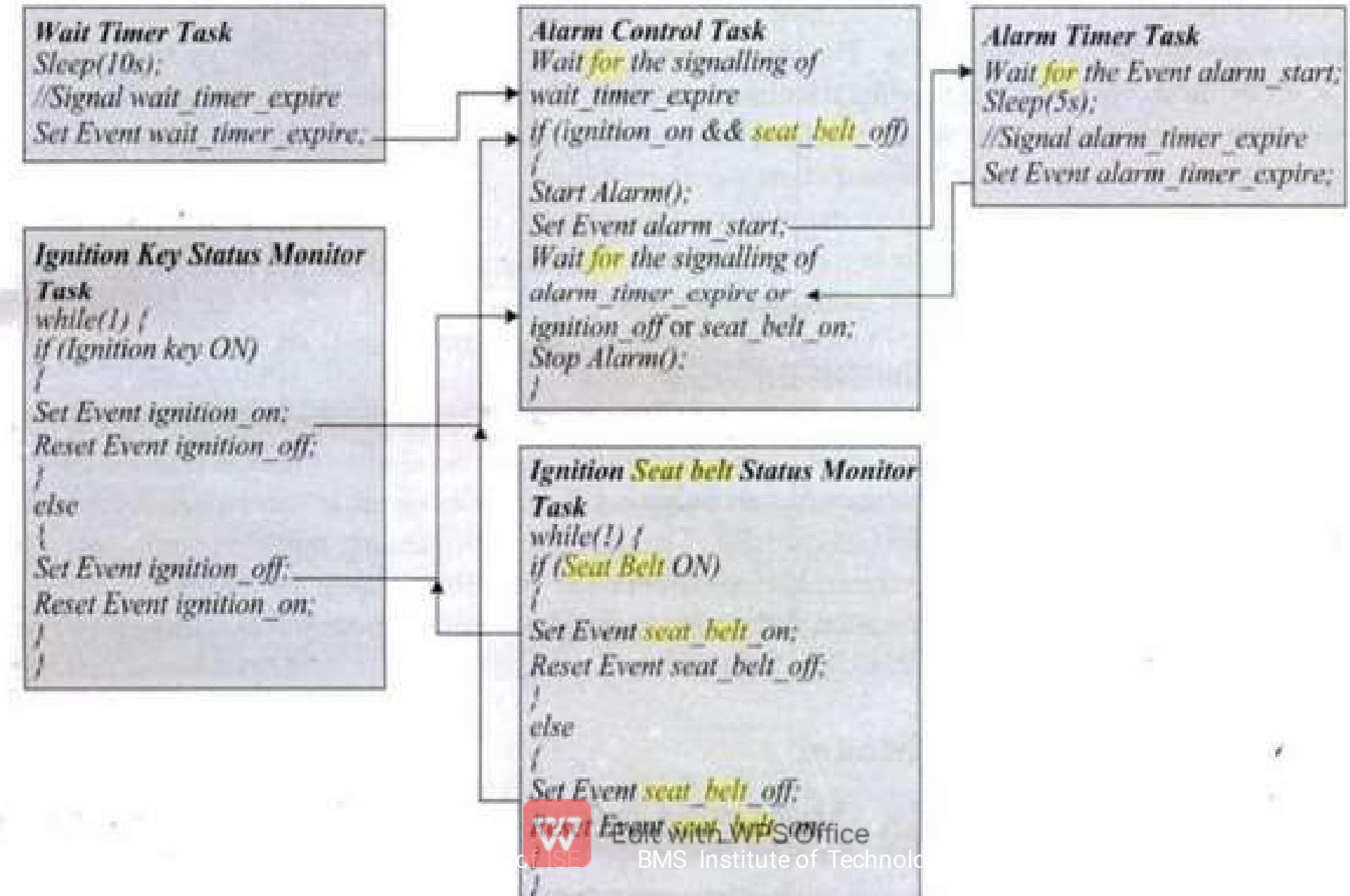
```
#define ON 1
#define OFF 0
#define YES 1
#define NO 0
void seat_belt_warn()
{
    wait_10sec();
    if (check_ignition_key() == ON)
    {
        if (check_seat_belt() == OFF)
        {
            set_timer(5);
            start_alarm();
            while ((check_seat_belt() == OFF) && (check_ignition_key() == OFF) && (timer_expire() == NO));
            stop_alarm();
        }
    }
}
```

# Hardware Software Co-Design: → Computational Models in ES

## 4. Concurrent / Communicating Process Model:

- Certain processing requirements are easier to model in concurrent processing model than the conventional sequential execution.
- Sequential execution leads to a single sequential execution of task and thereby leads to poor processor utilization, when the task involves I/O waiting, sleeping for specified duration etc.
- If the task is split into multiple subtasks, it is possible to tackle the CPU usage effectively, when the subtask under execution goes to a wait or sleep mode, by switching the task execution.
- Concurrent processing model requires additional overheads in task scheduling, task synchronization and communication
- Concurrent processing model requires additional overheads in task scheduling, task synchronization and communication

# Concurrent/Communicating Process Model: E.g. Automatic 'Seat Belt Warning' in an Automobile



# Hardware Software Co-Design: → Computational Models in ES

## 5. Object-Oriented Model:

- It is **object** based model for system requirements.
- A Complex requirement → into **simple** pieces called **objects**.
- This model brings:
- **This model brings:**
  - Reusability, Maintainability and Productivity.
  - Reusability, Maintainability and Productivity.
- **Object is represented as an entity**
  - Each object has a set of unique behaviour and state
- **A class is an abstract description i.e. blue print of object.**
- **Member variables, member functions, private, public, protected, inheritance, hiding, abstraction etc.**



Edit with WPS Office

# Embedded Firmware Design and development (Ch -9→9.1,9.2)



Edit with WPS Office

# Embedded Firmware Design and development

- The embedded firmware is responsible for:
  - Controlling the various peripherals of the embedded hardware
  - Generating response in accordance with the functional requirements of the product
- The embedded firmware is usually stored in a permanent memory (ROM) and it is non alterable by end users
- Designing Embedded firmware requires:
  - Understanding of the particular embedded product hardware
  - Some programming language
- The embedded firmware development process starts with:
  - Conversion of the firmware requirements → A program model using various modeling tools
  - A program model using

# Embedded Firmware Design and development

- There exist two basic approaches for the design and implementation of embedded firmware, namely;
  - The Super loop based approach
  - The Embedded Operating System based approach
- The decision on which approach needs to be adopted for firmware development is purely:
  - The Embedded Operating System based approach
- The decision on which approach needs to be adopted for firmware development is purely:
  - Dependent on the complexity and system requirements



Edit with WPS Office

# Embedded Firmware Design Approaches:-

## The Super loop based approach

- Suitable for applications that are:
  - Not time critical where the response time is not so important
- Similar to a conventional procedural programming → where the code is executed task by task
- Similar to a conventional procedural
  - The tasks just below the top are executed after completing the first task

where the code is executed task by task

- The task listed on top on the program code is executed first
- The tasks just below the top are executed after



Edit with WPS Office

# Embedded Firmware Design Approaches:-

## The Super loop based approach

- A typical super loop implementation will look like: → Infinite Loop(not-ending → while(1))
  1. Configure the common parameters and perform initialization for various hardware components memory, registers etc.
  2. Start the first task and execute it
  3. Execute the second task
  4. Execute the next task
  5. .
  6. .
  7. Execute the last defined task
  8. Jump back to the first task and follow the same flow

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



Edit with WPS Office

# Embedded Firmware Design Approaches:-



## The Super loop based approach

Advantages:

### Advantages based approach

- Doesn't require an **Operating System** for task scheduling and monitoring and free from OS related overheads
- Doesn't require an **Operating System** for task scheduling and monitoring
- Deployed in **low-cost** embedded products → where response time is not critical.
- Deployed in **low-cost** embedded products → where response time is not critical.
  - E.g. : Electronic Toy
- Simple and straight forward design without any OS overheads.

Dis-advantages:

- Non Real time in execution behavior i.e.:
  - E.g. : Electronic Toy
- Simple and straight forward design without any OS overheads.

Dis-advantages:

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

# Embedded Firmware Design Approaches:-

## Approaches:- The Super loop based approach

### Enhancements:

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



Edit with WPS Office

# Embedded Firmware Design Approaches:-

## The Embedded Operating Systems(OS) Based Approach



- **Embedded System Approach** consists of Operating System which can be any one of the:
  - A Real Time Operating System (RTOS) which can be any one of the:
  - A Customized General Purpose Operating System (GPOS)
  - A Real Time Operating System (RTOS)
- The Embedded OS is responsible:
  - For scheduling the execution of user tasks
  - The allocation of system resources among multiple tasks
- Involves lot of OS related overheads apart from managing and executing user defined tasks
- The Embedded OS is responsible:
  - For scheduling the execution of user tasks

# Embedded Firmware Design Approaches:-

## The Embedded Operating Systems(OS) Based Approach



- E.g. for GPOS:→
  - Microsoft® Windows XP Embedded
  - Devices using GPOS are: Point of Sale (PoS) terminals, Gaming Stations, Tablet PCs
- E.g. for RTOS:→
  - Windows CE, Windows Mobile, QNX, VxWorks, ThreadX , MicroC/OS-II, Embedded Linux, Symbian etc.
  - Devices using RTOS are: Mobile Phones, PDAs, Flight Control Systems
  - Devices using RTOS are: Mobile Phones, PDAs, Flight Control Systems



Edit with WPS Office



# Embedded Firmware Development

## Development

## Languages

- **Embedded firmware Development Languages/Options :-**

1. **Assembly Language Based Development**
2. **High Level Language Based Development**
3. **Mixing Assembly and High Level Language**



Edit with WPS Office

# Embedded Firmware Development

## Assembly Language Based Development



- 'Assembly Language' is the human readable notation with Mnemonics
- 'Machine language' is a processor understandable language with 1s and 0s
- Assembly language and machine languages are:
  - Processor/controller dependent
  - I.e. Program written for one processor will not work with others
- Assembly language programming:
  - Is the process of writing processor specific code in mnemonic form
  - Converting the mnemonics into actual processor instructions → machine language
  - By using an assembler
  - By using an assembler



Edit with WPS Office

# Embedded Firmware Development

## Assembly Language Based Development



- The general format of an assembly language instruction :
  - Opcode followed by Operands
  - In 8051 Processor:
  - In 8085 Processor:

Language	OPCODE (Action to be carried out)	OPERAND (data)
Assembly Language	MOV A (Mov to Accumulator)	#30
Machine Language	01110100	00011110

- An opcode can have no-operand / single operand / dual operand / more

# Embedded Firmware Development

## Assembly Language Based Development



- Each line in 'Assembly Language' consist of four fields :-

**LABEL**

**OPCODE** **OPERAND**

**COMMENTS**

- LABEL is an optional field. LABEL is commonly used for representing :
  - A memory location, address of a program, sub-routine, code portion etc.
  - A memory location, address of a program, sub-routine, code portion etc.
- The maximum length of a label differs between assemblers.
- The maximum length of a label differs between assemblers.
- Format : Suffix with a colon, Begin with a valid character, can contain number from 0 to 9 and special character \_ (underscore).
- Format : Suffix with a colon, Begin with a valid character, can contain number from 0 to 9 and special character \_ (underscore).

**DELAY: MOV R0, #255 ; Load Register R0 with 255**



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

# Embedded Firmware Development

## Assembly Language Based Development



- **Assembly Language – Source File to Hex File**

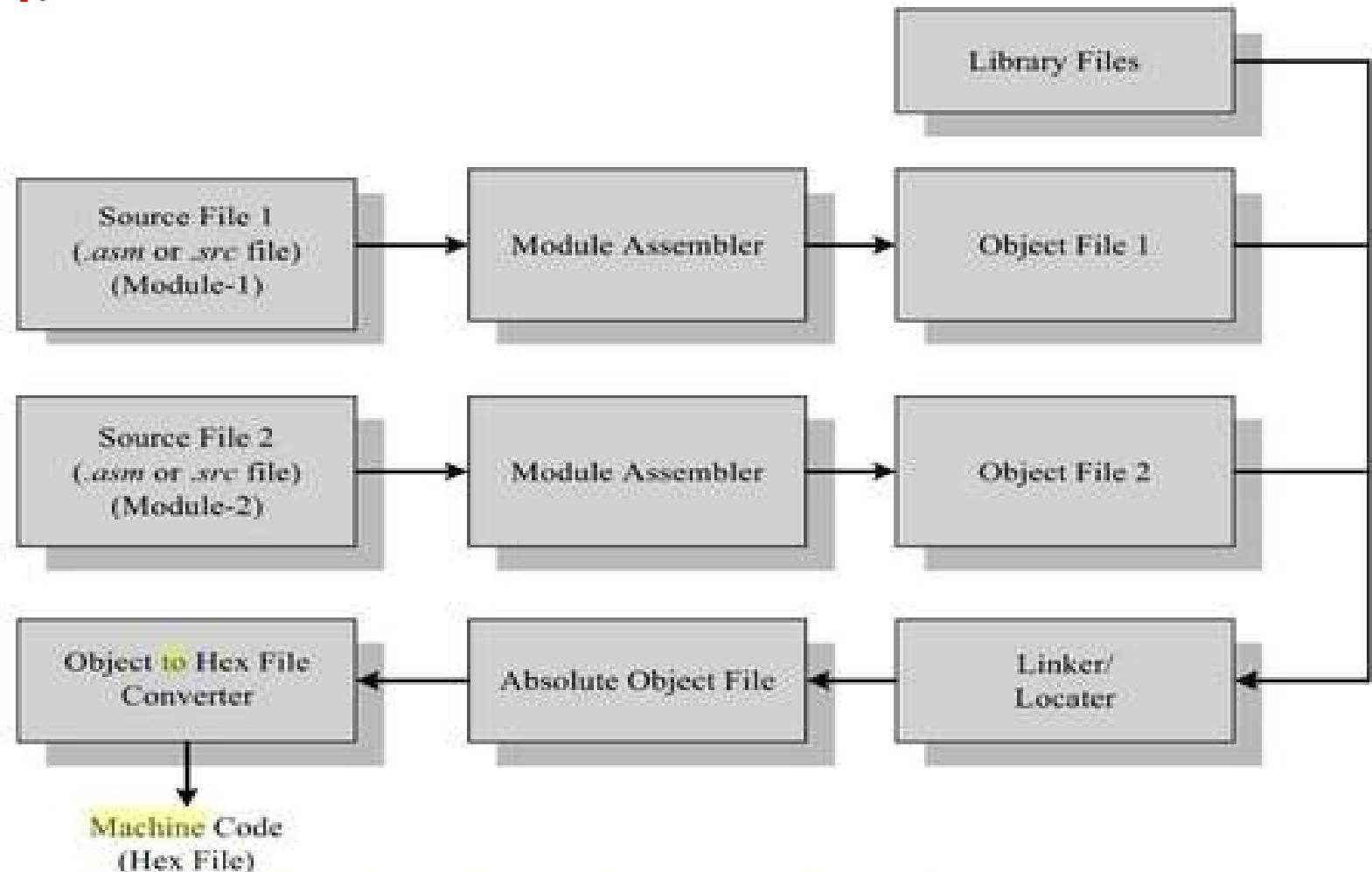


Fig. 9.1

Assembly language to machine language conversion process



Edit with WPS Office

# Embedded Firmware Development

## Assembly Language Based Development



### • Assembly Language – Source File to Hex File Translation

- Assembly code is saved as **.asm** (Assembly file) file or a **.src** (source) file or a format supported by the assembler
- The software utility called '**Assembler**' performs the translation of assembly code to machine code
- The assemblers for different family of target machines are **different**.
  - A51 Macro Assembler from Keil software is a popular assembler for the 8051 family micro controller
- Each **source file can be assembled separately** to :
  - Examine the syntax errors and incorrect assembly instructions
- Assembling of each **source file** generates → a **object file**.
- The software program called **linker/locater** is responsible for assigning **absolute** address to object files during the linking process
- The Absolute object file created from the object files corresponding to different source code modules by including library
- A software utility called '**Object to Hex file convertor**' translates the absolute



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

# Embedded Firmware Development

## Assembly Language Based Development



## Assembly Language Based Development

- **Advantages:**

- Efficient Code Memory & Data Memory Usage (Memory Optimization)
- High Performance
- Low level Hardware Access
- Code Reverse Engineering

- **Disadvantages:**

- High Development time
- Developer dependency
- Non portable



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

# Embedded Firmware Development

## High Level Language Based Development



- **High Level Language : Like C,C++**
- A software utility called 'cross-compiler':
  - Converts the high level language to target processor specific machine code
- The cross-compilation of each module generates a corresponding object file.
- The cross-compilation of each module generates a corresponding object file.
- The software program called linker/locater is responsible for assigning absolute address to object files during the linking process.
- The software program called **linker/locater** is responsible for assigning absolute address to object files during the linking process.
- The Absolute object file created from the object files corresponding to different source code modules contain information about the address where each instruction needs to be placed in code memory.
- The Absolute object file created from the object files corresponding to different source code modules contain information about the address where each instruction needs to be placed in code memory.
- A software utility called 'Object to Hex file converter' translates the absolute object file to corresponding hex file (binary file).
- A software utility called **Object to Hex file converter** translates the absolute object file to corresponding hex file (binary file).



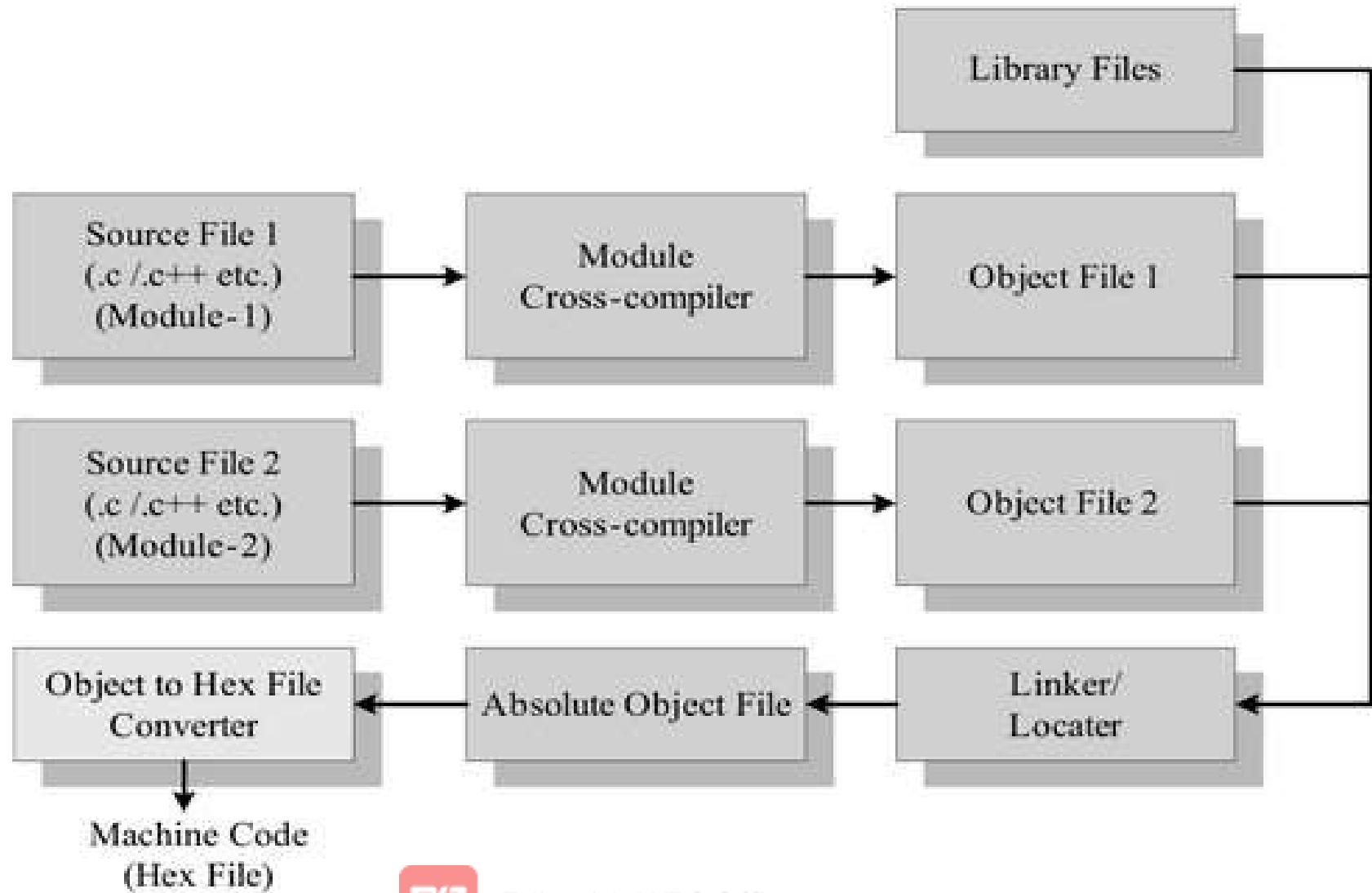
Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

# Embedded Firmware Development

# Highlevel Language Based Development





### High Level Language Based Development

- **Advantages:**

- Reduced Development time
- Developer independency
- Portability

- **Disadvantages:**

- Some cross compiler are not efficient for generating optimized target code.
- Not optimized in terms of performance and code size.
- Few code developed in high level language is not efficient in accessing low level hardware.
- The investment for high level language tools are high compared to assembly level language.



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt



## Mixing Assembly and High Level Language

### Mixing Assembly and High Level Language

- High Level language and Assembly level language can be mixed in three different ways
  - Mixing Assembly Language with High level language like 'C'
  - Mixing Assembly Language with High level language like 'C'
  - Mixing High level language like 'C' with Assembly Language
  - Inline Assembly
  - **Inline Assembly**
- The passing of parameters and return values between the high level and assembly level language is cross-compiler specific
- The passing of parameters and return values between the high level and assembly level language is **cross-compiler specific**



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt



## Mixing Assembly and High Level Language

### Mixing Assembly and High Level Language

- High Level language and Assembly level language can be mixed in three different ways
- **Mixing Assembly Language with High level language like 'C'**
- **Mixing High level language like 'C' with Assembly Language**
- **Inline Assembly**
- The passing of parameters and return values between the high level and assembly level language is cross-compiler specific
- The passing of parameters and return values between the high level and assembly level language is **cross-compiler specific**



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

# Embedded Firmware Development



## Mixing Assembly and High Level Language

### Mixing Assembly and High Level Language

#### I. Mixing Assembly Language with High level language :→Assembly with 'C'

– Used in situations :

– Used in situations :

→when cross-complier doesn't have built in to support:

→when cross-complier doesn't have built in to support:

- Interrupt Service Routine(**ISR**)

→when cross-complier is not able to generate optimized code with speed than Assembly code is

→when cross-complier is not able to provide the required timing

e  
specification of low level hardware  
with speed than Assembly code

– Mixing Assembly with C is complicated has:

→when cross-complier is not able to provide the required timing

- Passing parameters from C to Assembly code

specification of low level hardware

- Return values from Assembly to C code

specification of low level hardware

- Invoking of Assembly code from C code

– Mixing Assembly with C is complicated has:

All the above is cross compiler dependent and there is no universal rule.

- Passing parameters from C to Assembly code

- Return values from Assembly to C code

# Embedded Firmware Development



## Mixing Assembly and High Level Language

### Mixing Assembly and High Level Language

Example: Write a program in C that passes parameters and returns values from Assembly Language.

```
#pragma SRC
unsigned char my_assembly_func (unsigned int argument)
{
    return (argument + 1); // Insert dummy lines to access all args and
                          // retvals
}
```

- Here **#pragma SRC** is used to tell cross compiler to generate **Assembly** level code after compilation **not machine** language.

MOV	A, R7
INC	A
MOV	R7, A

- By this we can see how a C code is converted to .asm/.src code.



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

# Embedded Firmware Development



## Mixing Assembly and High Level Language

### Mixing Assembly and High Level Language

#### II. Mixing High Level language with Assembly language : → 'C' with Assembly

– Used in situations :

– Used in situations :

• When the entire source code is available in assembly only a routine is to be

• When the entire source code is available in assembly only a routine is to be  
written in C

• Some portions of the code may be difficult to code in Assembly language.

• To include built in library file in C . E.g. → Graphics library file

• Some portions of the code may be difficult to code in Assembly language.

– Mixing C with Assembly also has parameter passing and returning

values which are taken care by: → CPU registers, stack memory, fixed

memory

– E.g. **LCALL \_Cfunction**

• 'L' specifies cross compiler that parameters are passed through registers.

• 'C' specifies cross compiler that parameters are passed through fixed locations.

• If 'L' doesn't exist as a prefix then the parameters are passed through

fixed locations.

# Embedded Firmware Development

## Embedded Firmware Development Languages:-

### Mixing Assembly and High Level Language



## Mixing Assembly and High Level Language

### III. Inline Assembly

- Is another technique to insert assembly code at any location of source code written in ‘C’.
- This avoids the delay in calling an assembly routine from a ‘C’ code.
- This avoids the delay in calling an assembly routine from a ‘C’ code.
- Special keywords (`#pragma asm`, `#pragma endasm`) are used to indicate start and end of assembly instructions.
- These keywords are cross compiler specific.
- These keywords are cross compiler specific.

E.g. `#pragma asm`  
`MOV A, #13H`  
`#pragma endasm`

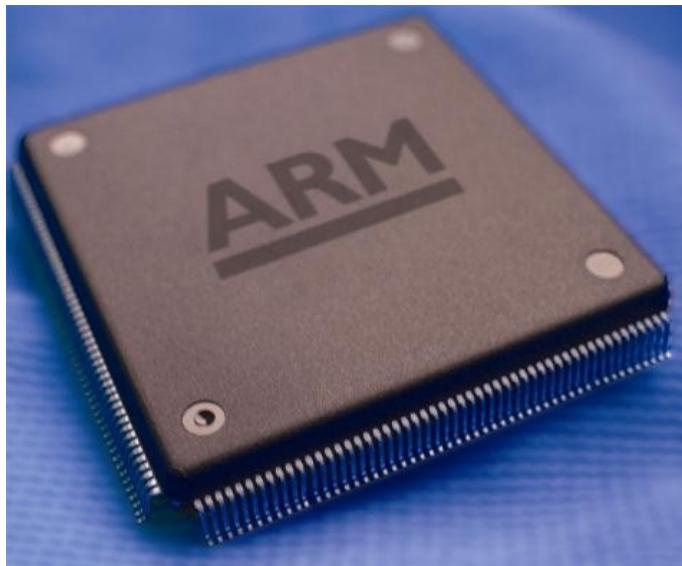


**Mrs. Ashwini N**  
Assistant Professor  
Dept.of.ISE  
BMSIT&M, Bengaluru  
**Email: ashwinilaxman@bmsit.in**

# Module – 5

## RTOS and IDE for Embedded System Design

### (18CS44)



Edit with WPS Office

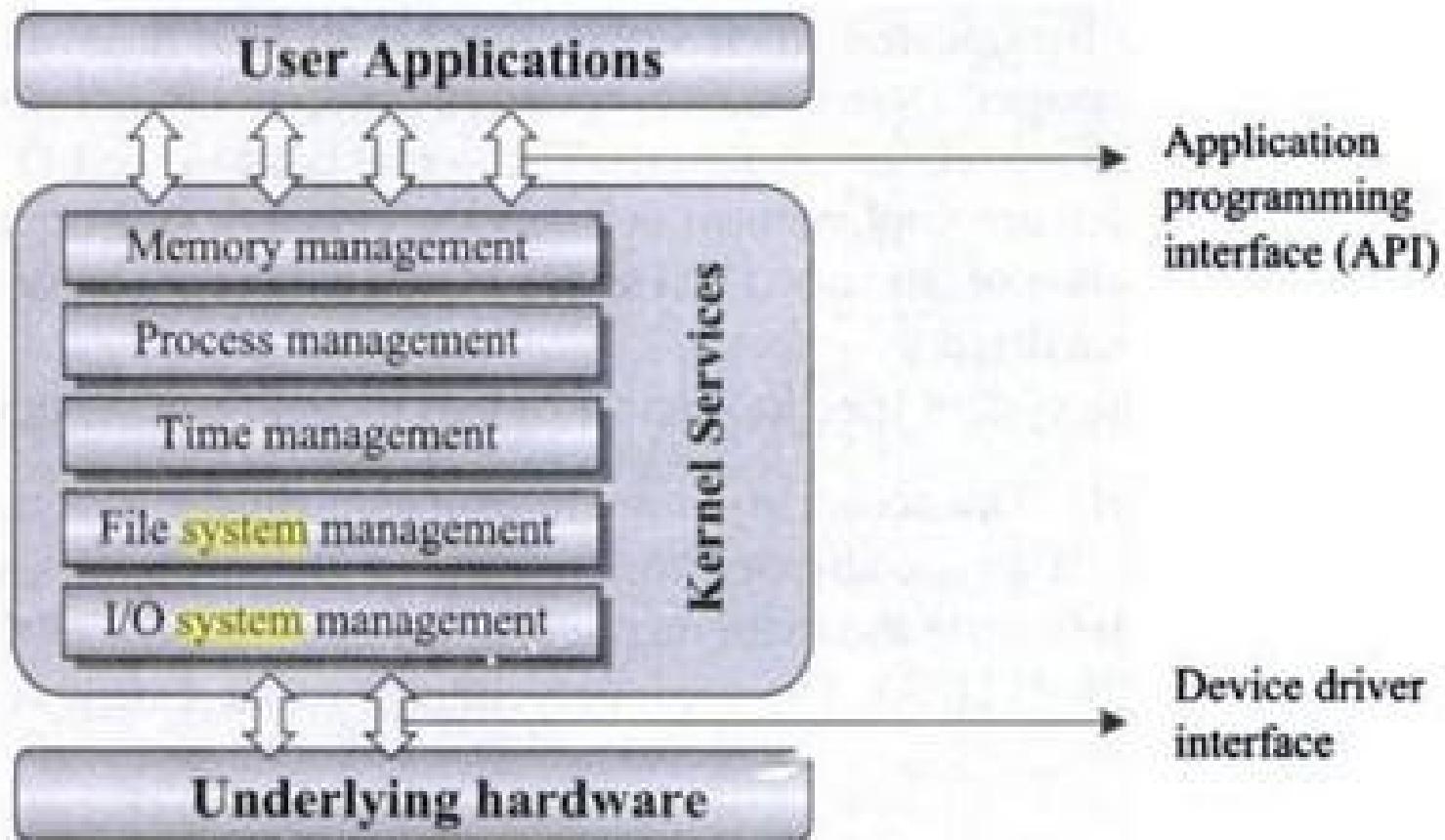
# Operating System Basics

- The Operating System acts as a **bridge** between:
  - = The user applications/tasks (and)
  - = The underlying system resources
  - = Through a set of system functionalities and services
- OS :→
  - = Manages resources
  - = Make them available → to the user applications/tasks on a need basis
- The **primary functions** of an Operating system is :
  - = Make the system convenient to use
  - = Organize and manage the system resources efficiently and correctly



Edit with WPS Office

# OS Architecture



## The Operating System Architecture



Edit with WPS Office

# OS Architecture : The Kernel



- The kernel is the core of the operating system
  - It is responsible for managing the system resources and the communication among the hardware and other system services and the user applications
  - Kernel acts as the abstraction layer between system resources and communication among the hardware and other system services
  - Kernel acts as the abstraction layer between system resources and user applications
- For a general purpose OS, the kernel contains different services like
  - Kernel contains a set of system libraries and services.
  - Process Management
  - Primary Memory Management
  - File System management
  - Primary Memory Management
  - I/O System (Device) Management
  - File System management
  - Secondary Storage Management
  - I/O System (Device) Management
  - Protection
  - Secondary Storage Management
  - Time management
  - Protection
  - Interrupt Handling
  - Time management
  - Interrupt Handling



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

# OS Architecture : The Kernel

## Kernel Space and User Space

- The program code corresponding to the kernel applications/services are kept:
  - In a contiguous area → i.e. (OS dependent) → In primary (working) memory (and)
  - Is protected from the un-authorized access by user
  - Is protected from the un-authorized access
- The memory space at which the kernel code is located is known as 'Kernel Space'
- The memory space at which the kernel code is located
- All user applications are loaded to a specific area of primary memory and this memory area is referred as 'User Space'
- The partitioning of memory into kernel and user space is known as 'Kernel Space'
- All user applications are loaded to a specific area of primary



# OS Architecture : The Kernel Space and User Space

- An operating system with virtual memory support,
  - loads the user applications into its corresponding virtual memory space with demand paging technique
- Most of the operating systems keep:
  - space with demand paging technique
- Most of the operating systems keep:
  - The kernel application code in main memory and it is not swapped out into the secondary memory
- Different approaches for building an Operating System Kernel
  - The kernel application code in main memory and it is not swapped out into the secondary memory
  - Monolithic Kernel
  - Microkernel
- Different approaches for building an Operating System Kernel
  - Monolithic Kernel
  - Microkernel

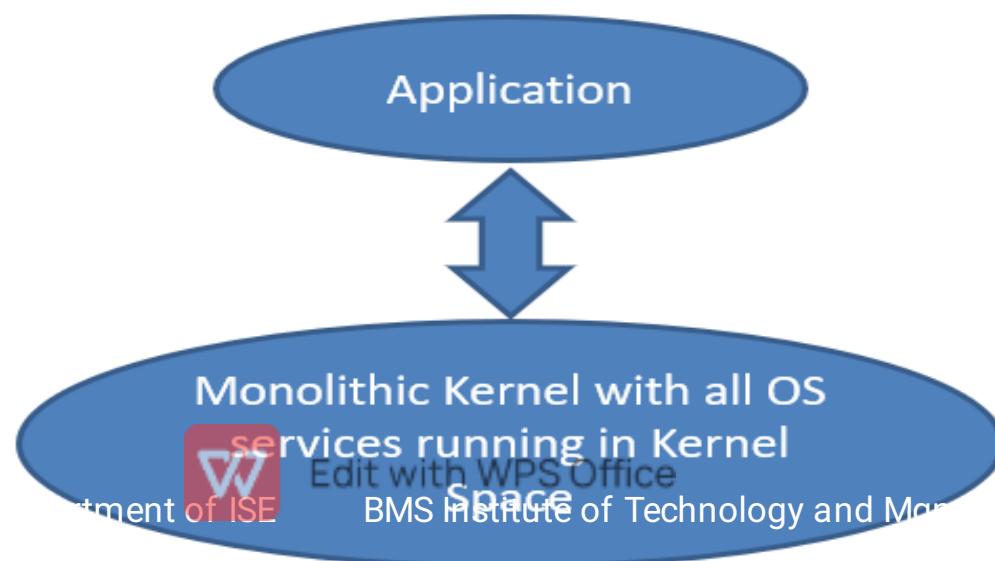




# OS Architecture : The Kernel

## Kernel Monolithic Kernel

- All kernel services run in the kernel space
- All kernel modules run within the same memory space under a single kernel thread
- The internal integration of kernel modules in monolithic kernel architecture.
- The internal integration of kernel modules in monolithic kernel
  - allows the effective utilization of the low-level features of the underlying system





# OS Architecture : The Kernel

## Kernel Monolithic Kernel

- The major drawback of monolithic kernel is that:
  - Any error or failure in any one of the kernel modules → leads to the crashing of the entire kernel application
- Examples of monolithic kernel :
- Examples of monolithic kernel :
  - LINUX, SOLARIS, MS-DOS
  - LINUX, SOLARIS, MS-DOS



Edit with WPS Office

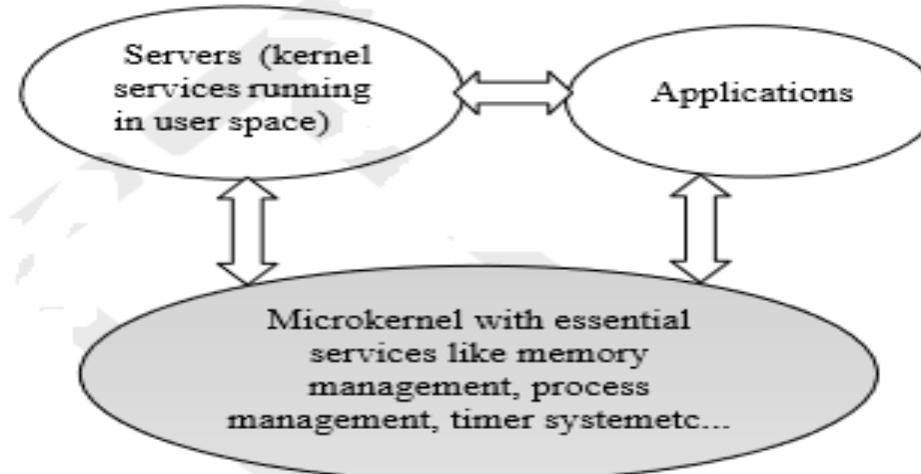
Department of ISE

BMS Institute of Technology and Mgmt



# OS Architecture : The Kernel Microkernel

- The microkernel design incorporates:
  - Only the **essential** set of Operating System services into the kernel
  - Rest of the Operating System services are implemented in programs known as 'Servers which runs in user space'
  - Rest of the Operating System services are implemented in programs known as 'Applications which runs in user space'



- Memory management, process management, timer systems and interrupt handlers are **examples** of essential services, which forms the part of the microkernel



Edit with WPS Office



# OS Architecture : The Kernel

## Kernel Microkernel

- Examples of Microkernel:
  - QNX, Minix 3
- Benefits of Microkernel:
  1. **Robustness:** If a problem is encountered in any services in server →
    - Can be reconfigured and re-started without the need for re-starting the entire OS.
    - Can be reconfigured and re-started without the need for re-starting the entire OS.
  2. **Configurability:** Any services , which run as 'server' application →
    - Can be changed without need to restart the whole system.
  2. **Configurability:** Any services , which run as 'server' application →
    - Can be changed without need to restart the whole system.



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt



# Types of Operating Systems

## Systems

Depending on the:

- Type of kernel and kernel services
- Purpose and type of computing systems where the OS is deployed and the responsiveness to applications where the OS is
- Operating Systems are classified into
  - deployed and the responsiveness to applications
  - 1. General Purpose Operating System (GPOS)
  - 2. Real Time Purpose Operating System (RTOS)
- Operating Systems are classified into
  - 1. General Purpose Operating System (GPOS)
  - 2. Real Time Purpose Operating System (RTOS)



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt



# Types of Operating Systems

## General Purpose Operating System (GPOS)

- Operating Systems, which are **deployed** in general computing systems
- The kernel is more **generalized** and contains all the required services to **execute** generic applications
- Need not be **deterministic** in execution behavior
- May **inject** random delays into application software
  - Thus cause slow responsiveness of an application at unexpected times
- Usually **deployed** in computing systems:
- **Usually deployed** in computing systems:
  - Where deterministic behavior is not an important criterion
  - Where deterministic behavior is not an important criterion
- **Example Systems:** Personal Computer/Desktop system
- **GPOS Examples:** Windows XP/MS-DOS etc



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt



# Types of Operating Systems

## Real Time Operating System (RTOS)

- Operating Systems, which are **deployed** in embedded systems demanding **real-time response**
- Deterministic** in execution behavior. Consumes only known amount of time for kernel applications
- Implements scheduling policies for executing the highest priority task/application always
- Implements scheduling policies for executing the highest priority task/application always
- Implements policies and rules concerning time-critical allocation of a system's resources
- RTOS Examples : Windows CE, QNX, VxWorks , MicroC/OS-II etc.
- RTOS Examples : Windows CE, QNX, VxWorks , MicroC/OS-II etc.



Edit with WPS Office



# Types of Operating Systems

## Real Time Operating System (RTOS)

### Systems (RTOS)

- In RTOS we need to study about:
- In RTOS we need to study about:
  - The Real Time Kernel
  - Hard Real Time
  - Soft Real Time
  - Soft Real Time



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt



# Types of Operating Systems

Real Time Operating System (RTOS): The Real Time Kernel

## Systems

- The Real Time Kernel: The kernel of a Real Time Operating System is referred as **Real Time kernel**.
- The Real Time kernel is **highly specialized** and it contains only the **minimal set of services** required for running the user applications/tasks.
- The basic functions of a Real Time kernel are
  - a) Task/Process management
  - b) Task/Process scheduling
  - c) Task/Process synchronization
  - d) Error/Exception handling
  - e) Memory Management
  - f) Interrupt handling
  - g) Time management



# Types of Operating Systems

## Real Time Operating System (RTOS): The Real Time Kernel

### a) Task/Process Management : Deals with →

- Setting up the memory space for the tasks
- Loading the task's code into the memory space
- Allocating system resources,
- Setting up a Task Control Block (TCB) for the task and task/process termination/deletion.

and

- A Task Control Block (TCB) is used for holding the information corresponding to a task.

- A Task Control Block (TCB) is used for **holding** the information
  - *Task ID*: Task Identification Number corresponding to a task.

- TCB Contains following information:
  - *Task ID*: Task Identification Number

- *Task State*: The current state of the task. (E.g. State= 'Ready' for a task which is ready to execute)
  - *Task ID*: Task Identification Number
  - *Task State*: The current state of the task. (E.g. State= 'Ready' for a task which is ready to execute)

Contd..



# Types of Operating Systems

Real Time Operating System (RTOS) : The Real Time Kernel

## a) Task/Process Management :

- TCB Contains following information:
  - Task Type: Task type. Indicates what is the type for this task. The task can be a hard real time or soft real time or background task.
  - Task Priority: Task priority (E.g. Task priority =1 for task with priority = 1) Priority: Task priority (E.g. Task priority = 1 for task)
  - Task Context Pointer: Pointer for context saving =1 for task
  - Task Memory Pointers: Pointers to the code memory, data memory and stack memory for the task with priority = 1
  - Task System Resource Pointers: Pointers to system resources (semaphores, mutex etc) used by the task
  - Task Memory Pointers: Pointers to the code memory, data memory and stack memory for the task
  - Task Pointers: Pointers to other TCBs (TCBs for preceding, next and waiting tasks)
  - Other Parameters: Other relevant task parameters



# Types of Operating Systems

Real Time Operating System (RTOS) : The Real Time Kernel

## Systems

b) Task/Process Scheduling : Deals with →

- Sharing the CPU among various tasks/processes.
- A kernel application called 'Scheduler' handles the task scheduling.
- Scheduler is nothing but an algorithm implementation,
- Scheduler is nothing but an algorithm implementation,
  - Which performs the efficient and optimal scheduling of tasks to provide a deterministic behavior.

c) Task/Process Synchronization : Deals with →

c) Task/Process Synchronization : Deals with →

- Synchronizing the concurrent access of a resource,
  - Synchronizing the concurrent access of a resource,
    - Which is shared across multiple tasks and the communication between various tasks.
- between various tasks.



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt



# Types of Operating Systems

Real Time Operating System (RTOS): The Real Time Kernel

d) Error/Exception Handling: Deals with →

- Registering and handling the errors occurred/exceptions raised during the execution of tasks.
- Few of the examples are: Insufficient memory, timeouts, deadlocks, deadline missing, bus error, divide by zero, unknown instruction execution etc.
- Errors/Exceptions can happen at :→
  - Kernel level services : Example → Deadlock
  - Task level: Example → Timeout is an example for a task level exception.



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt



# Types of Operating Systems

Real Time Operating System (RTOS) : The Real Time Kernel

e) Memory Management: Deals with →

- The memory allocation time increases depending on: →
    - The size of the block of memory needs to be allocated
    - The state of the allocated memory block (initialized memory block consumes more allocation time than un- initialized memory block)
  - Since predictable timing and deterministic behavior are the primary focus for an RTOS, RTOS achieves this →
  - RTOS generally uses block based memory allocation technique.
- Blocks are fixed size of dynamic memory
- The block is allocated for a task on a need basis.
- The blocks are stored in a 'Free buffer Queue'.

Contd..



Edit with WPS Office



# Types of Operating Systems

Real Time Operating System (RTOS): The Real Time Kernel

## e) Memory Management:

- RTOS kernels assume that the whole design → Is proven correct and protection is unnecessary.
  - Some commercial RTOS kernels allow memory protection as optional
  - The kernel enters a fail-safe mode when an illegal memory access occurs
  - The kernel enters a fail-safe mode when an illegal memory access occurs
- A few RTOS kernels implement Virtual Memory concept for memory allocation if the system supports secondary memory storage (like HDD and FLASH memory)
- A few RTOS kernels implement Virtual Memory concept for memory allocation if the system supports secondary memory storage (like HDD and FLASH memory)

# Real Time Operating System (RTOS): The Real Time Kernel



## f) Interrupt Handling:

- Interrupts inform the processor that an external device or an associated task → Requires immediate attention of the CPU.
- Can Handle Nested Interrupts.
- Interrupts can be either:

### = **Synchronous:**

- Interrupts which occurs in sync with the currently executing task
- Example: Software interrupts
- For synchronous interrupts, the interrupt handler runs in the same context of the interrupting task.
- For synchronous interrupts, the interrupt handler

### = **Asynchronous:**

- Interrupts which occurs at any point of execution of any task, and are not in sync with the currently executing task.
- Example: Hardware interrupts(generated by external devices)
- For asynchronous interrupts, the interrupt handler is usually written as separate task (Depends on OS Kernel implementation) and it runs in a different context.



# Types of Operating Systems

Real Time Operating System (RTOS): The Real Time Kernel

## g) Time Management:

- Accurate time management is essential for providing precise time reference for all applications.
- The time reference to kernel is provided by → a high-resolution Real Time Clock (RTC) hardware chip (hardware timer)
  - The hardware timer is programmed to interrupt the processor/controller at a fixed rate.
  - This timer interrupt is referred as 'Timer tick'
  - This timer interrupt is referred as 'Timer tick'
- The 'Timer tick' is taken as the timing reference by the kernel.
  - The 'Timer tick' interval may vary depending on the hardware timer.
  - Usually the 'Timer tick' varies in the microseconds range
  - The 'Timer tick' interval may vary depending on the hardware timer.
  - Usually the 'Timer tick' varies in the microseconds range

Contd..



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt



# Types of Operating Systems

Real Time Operating System (RTOS): The Real Time Kernel

## g) Time Management:

- Accurate time management is essential for providing precise time reference for all applications.
- The time reference to kernel is provided by → a high-resolution Real Time Clock (RTC) hardware chip (hardware timer)
  - The hardware timer is programmed to interrupt the processor/controller at a fixed rate.
  - This timer interrupt is referred as 'Timer tick'
  - This timer interrupt is referred as 'Timer tick'
- The 'Timer tick' is taken as the timing reference by the kernel.
  - The 'Timer tick' interval may vary depending on the hardware timer.
  - Usually the 'Timer tick' varies in the microseconds range
  - The 'Timer tick' interval may vary depending on the hardware timer.
  - Usually the 'Timer tick' varies in the microseconds range

Contd..



Edit with WPS Office

Department of ECE

BMSS Institute of Technology and Mgmt



# Types of Operating Systems

Real Time Operating System (RTOS): The Real Time Kernel

## Systems

### g) Time Management:

- The time parameters for tasks are expressed as the multiples of the 'Timer tick'
- The System time is updated based on the 'Timer tick'
- If the System time register is 32 bits wide and the 'Timer tick' interval is 1 microsecond, the System time register will reset in **1.19 Hours**
  - $2^{32} * 10^{-6} / (24 * 60 * 60) = 1.19 \text{ Hours}$
- If the System time register is 32 bits wide and the 'Timer tick' interval is 1 millisecond, the System time register will reset in **50 Days**
  - $2^{32} * 10^{-3} / (24 * 60 * 60) = 50 \text{ Days}$

Contd..



Edit with WPS Office

Department of

BMS Institute of Technology and Mgmt



# Types of Operating Systems

- Real Time Operating System (RTOS):** The Real Time Kernel
- g) Time Management: The 'Timer tick' interrupt can be utilized for implementing the following actions. Only few are listed: →
- for implementing the following actions. Only few are listed: →
- 
- Save the current context (Context of the currently executing task)
  - Save the current context (Context of the currently executing task)
  - Update the timers implemented in kernel
  - Activate the periodic tasks, which are in the idle state
  - Invoke the scheduler and schedule the tasks again based on the scheduling algorithm
  - Activate the periodic tasks, which are in the idle state
  - Delete all the terminated tasks and their associated data structures (TCBs)
  - Invoke the scheduler and schedule the tasks again based on the scheduling algorithm
  - Delete all the terminated tasks and their associated data structures (TCBs)



# Types of Operating Systems

## Real Time Operating System (RTOS)

### Hard Real Time :

- A Real Time Operating Systems which strictly adheres to the timing constraints for a task strictly adheres to the timing constraints for a task
  - Missing any deadline may produce catastrophic results including permanent data loss and irrecoverable damages to the system/users
- Must meet the deadlines for a task without any slippage
  - Missing any deadline may produce catastrophic results including permanent data loss and irrecoverable damages to the system/users
- Emphasize on the principle 'A late answer is a wrong answer'
- Most Hard real time system are automatic that is does not contain a 'human in the loop'
  - Example: Air bag control systems and Anti-lock Brake Systems (ABS) of vehicles.
- Emphasize on the principle 'A late answer is a wrong answer'
  - As a rule of thumb,
    - Example: Air bag control systems and Anti-lock Brake Systems (ABS) of vehicles.
    - Hard Real Time Systems does not implement the virtual memory model for handling the memory.
    - This eliminates the delay in swapping in and out the code corresponding to the task to and from the primary memory/Office.
- As a rule of thumb,
  - Hard Real Time Systems does not implement the virtual memory



# Types of Operating Systems

## Real Time Operating System (RTOS)

### Soft Real Time :

- Real Time Operating Systems that does not guarantee meeting deadlines, but, offer the best effort to meet the deadline
- Missing deadlines for tasks are acceptable
  - if the frequency of deadline missing is within the compliance limit of the Quality of Service (QoS)
- Emphasizes on the principle 'A late answer is an acceptable answer, but it could have done bit faster'
- Soft Real Time systems most often have a 'human in the loop (HITL)'
- Example : Automatic Teller Machine (ATM) , An audio video play back system

# 10.3 Tasks, Process and Threads



## In the Operating System context →

- A task is defined as the program in execution
- The related information maintained by the Operating system for the program
- Task is also known as 'Job'.
- A program or part of it in execution is also called a 'Process'
- The terms 'Task', 'job' and 'Process' refer to the same entity in the Operating System context and most often they are used interchangeably
- A process requires various system resources like CPU for executing the process, memory for storing the code corresponding to the process and associated variables, I/O devices for information exchange etc.

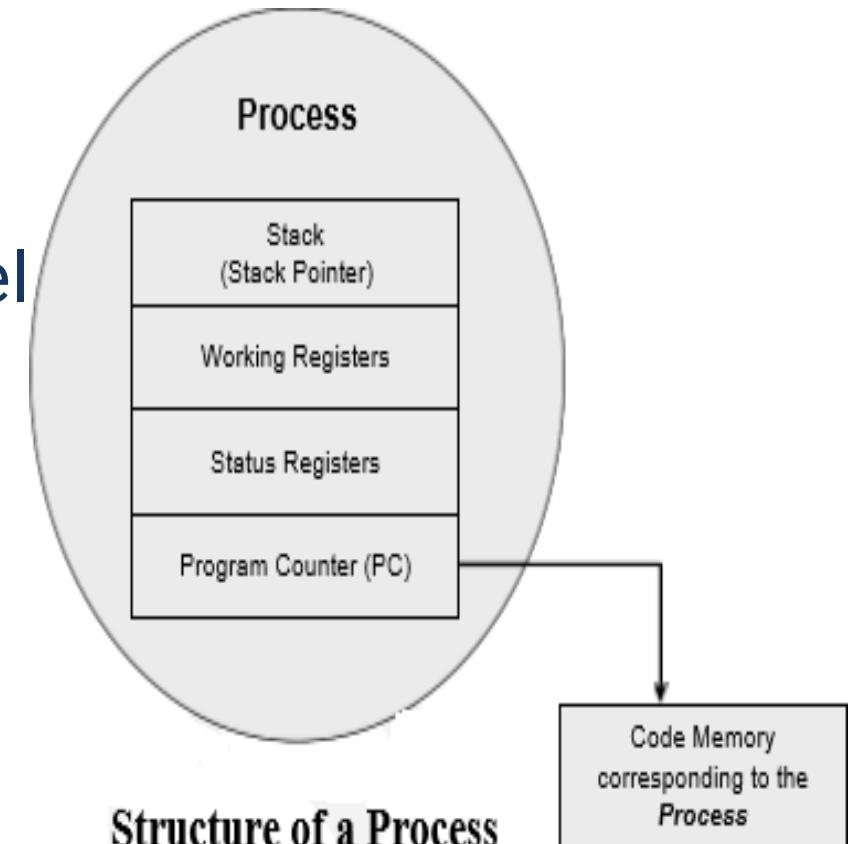
# 10.3 Tasks, Process and Threads:

## Process



### The structure of a Process

- 'Process' leads to concurrent execution (parallelism) of tasks
- concurrent
- Which needs:
  - Efficient utilization of the CPU and other system resources
- Which needs:
  - Concurrent execution is achieved → the sharing of CPU and other system resources
- A Process contains:
  - Registers, Stack, PC, code corresponding to process
- Concurrent execution is achieved → the sharing of CPU among the processes.
- A Process contains
  - Registers, Stack, PC,



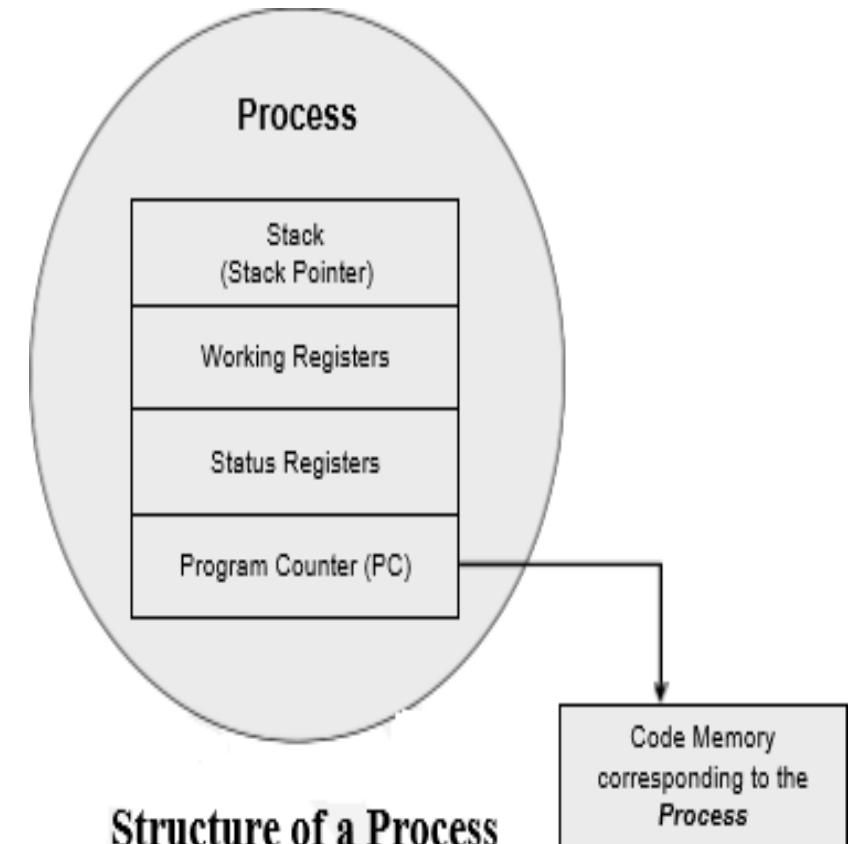
# 10.3 Tasks, Process and Threads: Process



## The structure of a Process

S :

- A process, which inherits all the properties of the CPU.
- A process, which inherits all
  - Can be considered as a virtual processor, awaiting its turn to have its properties switched into the physical processor.
  - Can be considered as a virtual processor, awaiting its turn to have its properties switched into the physical processor. Its properties are mapped to the physical registers of the CPU.
  - When the process gets its turn, its registers and Program



# 10.3 Tasks, Process and Threads:

## Process

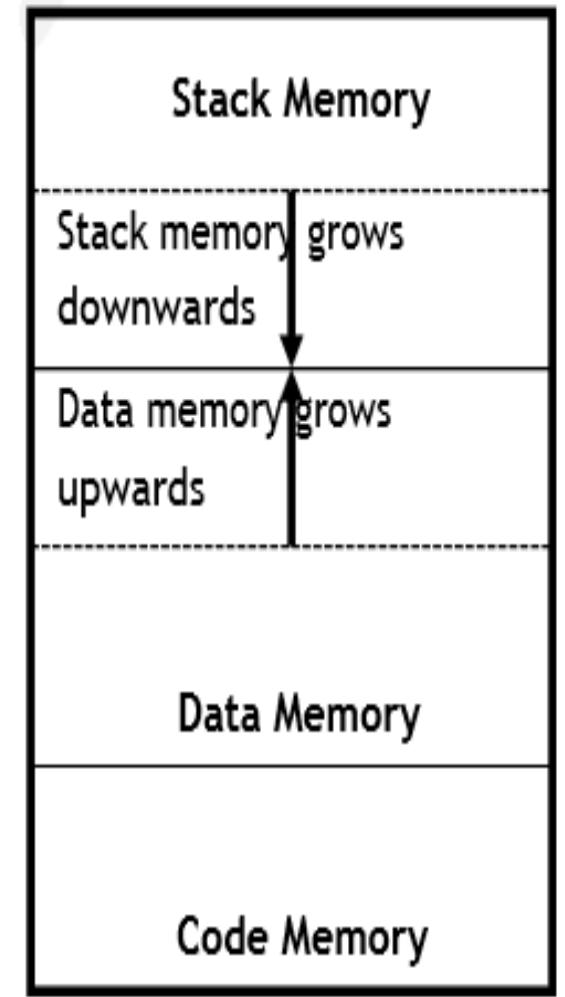


### Memory organization of Processes:

- The memory occupied by the process is segregated into three regions:

- Stack memory
  - Holds all temporary data such as variables local to the process
  - Starts at the highest memory address from the memory area allocated for the process
  - (Depending on the system) Usually starts at the highest memory address
- Data memory
  - Holds all global data for the process
- Code memory
  - Contains the program code (instructions) corresponding to the process

- On loading a process into the main memory, a specific area of memory is allocated for the process



Memory organization of a Process

# 10.3 Tasks, Process and Threads: Process



## Process States & State Transition

- The creation of a process → to its termination is not a single step operation
- The process traverses through a series of states during its transition from the:
  - Newly created state to the terminated state
  - **Newly created state to the terminated state**
- The cycle through which a process changes its state from 'newly created' to 'execution completed' is known as '**Process Life Cycle**'.
- Process Life Cycle is represented through State transition diagram.
- **Process Life Cycle is represented through State transition diagram.**



Edit with WPS Office

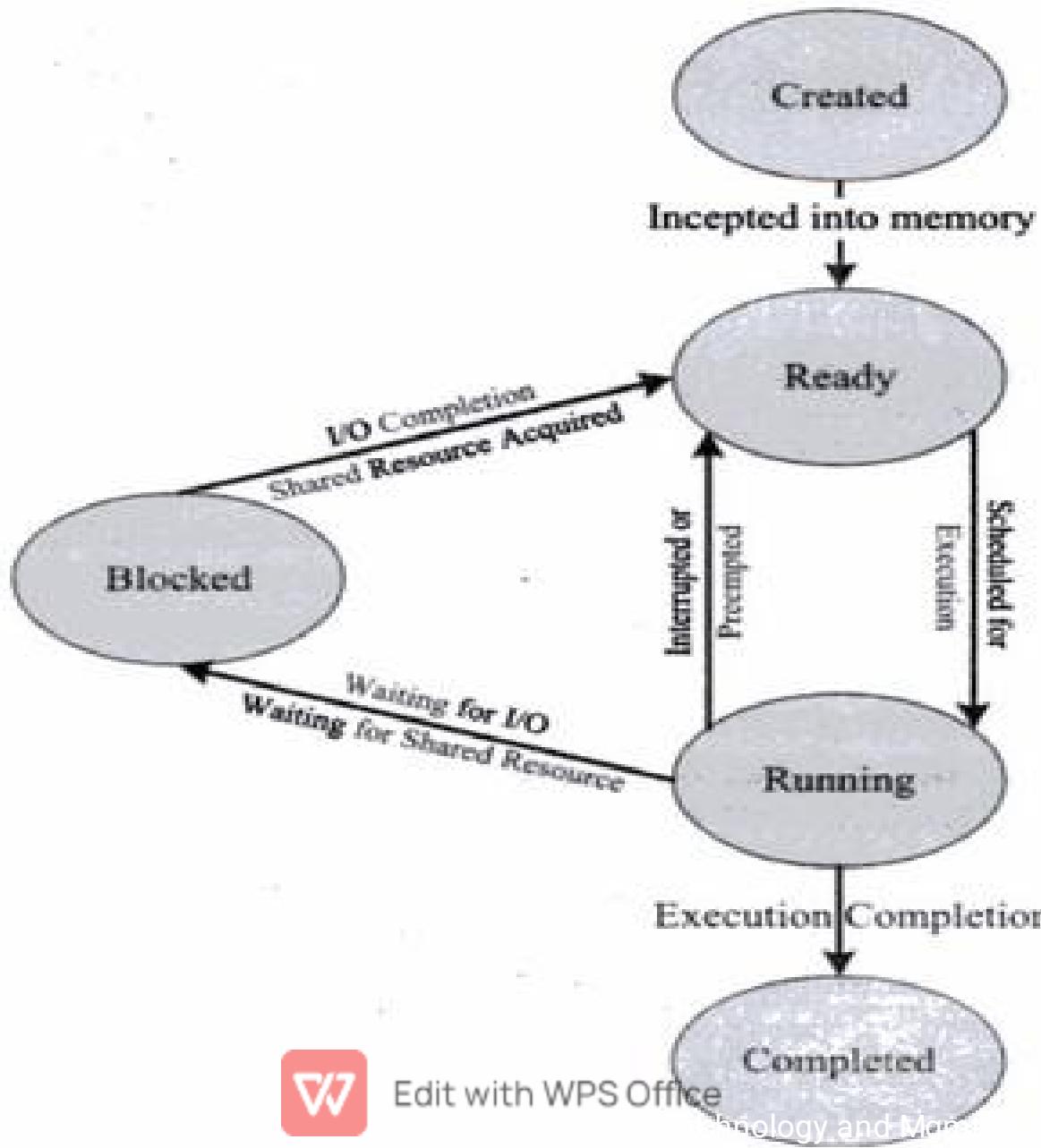
Department of ISE

BMS Institute of Technology and Mgmt

# 10.3 Tasks, Process and Threads: Threads: Process-State



D



Edit with WPS Office

Technology and More

# 10.3 Tasks, Process and Threads: Process



**Process Management:** Deals with →

- Creation of Process
- Setting up memory space
- Loading the process to memory space
- Setting up Process Control Block (PCB) for the process and process termination / deletion.



Edit with WPS Office

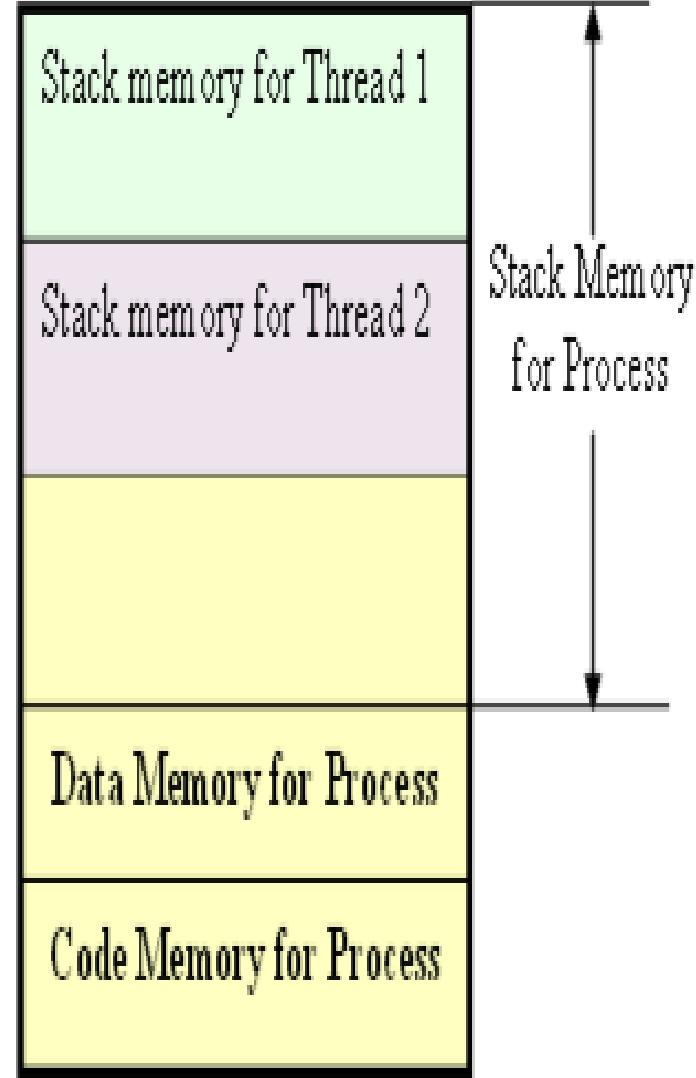
Department of ISE

BMS Institute of Technology and Mgmt

# 10.3 Tasks, Process and Threads: Threads



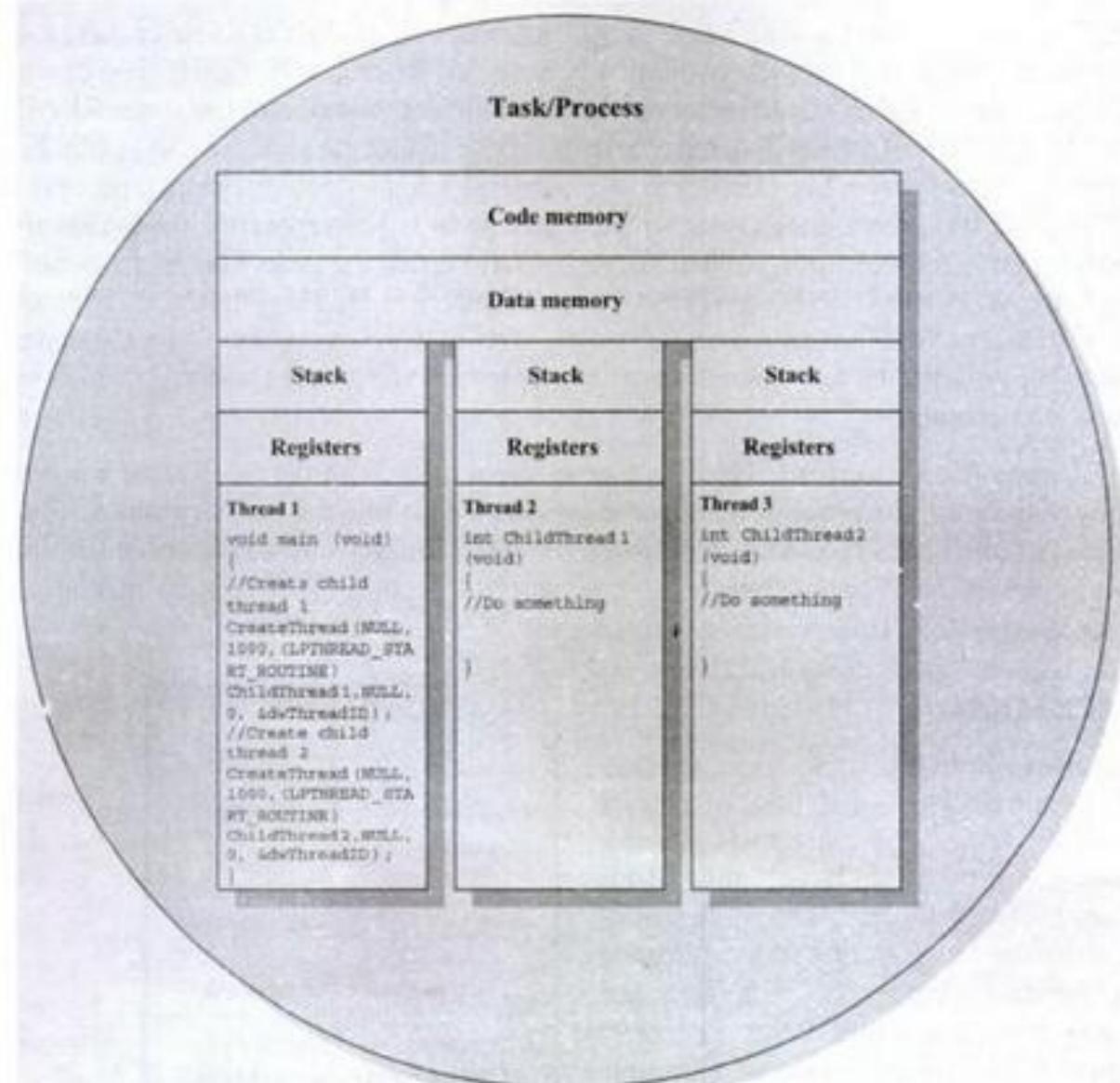
- A thread is the primitive that can execute code.
- A thread is a single sequential flow of control within a process.
- A thread is a single sequential flow of control
- 'Thread' is also known as lightweight process. within a process.
- A process can have many threads of execution.
- 'Thread' is also known as lightweight process.
- Different threads, which are part of a process
  - Share the same address space →
  - Share the data memory, code memory and heap memory area
- A process can have many threads of execution.
  - Share the data memory, code memory and heap memory area
- Different threads, which are part of a process
  - CPU register values
  - Share the same address space →
  - Program Counter (PC)
    - Share the data memory, code memory and heap memory area
  - stack
- Threads maintain their own thread status
  - CPU register values
  - Program Counter (PC)
- Threads maintain their own thread status
  - CPU register values
  - Program Counter (PC)



# 10.3 Tasks, Process and Threads: Threads: Threads-Concept of MultiThreading



- Better memory utilization.
- Since the process is split into different threads, when one thread enters a wait state, the CPU can be utilized by other.
- Hence Efficient CPU utilization. The CPU is engaged all time.
- Speeds up the execution of the process.



# 10.3 Tasks, Process and Threads:

## Threads-Pre-emption(Context Switching)



### User Level Threads :

- User level threads do not have kernel/Operating System support
- They exist solely in the running process.
- Even if a process contains multiple user level threads,
  - The OS treats it as single thread and will not switch the execution among the different threads of it.

### Kernel Level Threads :

- Kernel level threads are individual units of execution
  - Which the OS treats as separate threads.
- The OS interrupts the execution of the currently running kernel thread
  - Switches the execution to another kernel thread



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

# 10.3 Tasks, Process and Threads: Threads-Pre-emption(Context Switching)



As user level threads has to be mapped for kernel level thread for execution following models are used.

- **Many-to-One Model:**
  - Many user level threads are mapped to a single kernel thread.
  - The kernel treats all user level threads as single thread
  - The execution switching among the user level threads happens → happens →
    - when a currently executing user level thread voluntarily blocks itself
  - Example: Solaris Green threads and GNU Portable or relinquishes the CPU.
- **One-to-One Model:**
  - Example : Solaris Green threads and GNU Portable
  - Each user level thread is bonded to a kernel/system level thread.
- **One-to-One Model:**
  - Example : Windows XP/NT/2000 and Linux threads
  - Each user level thread is bonded to a kernel/system level
- **Many-to-Many Model:**
  - In this model many user level threads are allowed to be mapped to many kernel threads.
  - Example : Windows XP/NT/2000 and Linux threads
- **Many-to-Many Model:**
  - Example : Windows NT/2000 with Thread Fiber package
  - In this model many user level threads can be mapped to

# 10.3 Tasks, Process and Threads: Threads Vs Process



Thread	Process
Thread is a single unit of execution and is part of process.	Process is a program in execution and contains one or more threads.
A thread does not have its own data memory and heap memory. It shares the data memory and heap memory with other threads of the same process.	Process has its own code memory, data memory and stack memory.
A thread cannot live independently; it lives within the process.	A process contains at least one thread.
There can be multiple threads in a process. The first thread (main thread) calls the main function and occupies the start of the stack memory of the process.	Threads within a process share the code, data and heap memory. Each thread holds separate memory area for stack (shares the total stack memory of the process).
Threads are very inexpensive to create	Processes are very expensive to create. Involves many OS overhead.
Context switching is inexpensive and fast	Context switching is complex and involves lot of OS overhead and is comparatively slower.
If a thread expires, its stack is reclaimed by the process.	If a process dies, the resources allocated to it are reclaimed by the OS and all the associated threads of the process also dies.

# 10.4 Multiprocessing and Multitasking

## Multitasking



- The ability to execute multiple processes simultaneously.
- Systems which are capable of performing multiprocessing are known as multiprocessor systems
  - Multiprocessor systems possess multiple CPUs and can execute multiple processes simultaneously
- The ability of the Operating System to have multiple programs in memory, which are ready for execution, is referred as multiprogramming.



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

# 10.4 Multiprocessing and Multitasking

## Multitasking



- Ability of an operating system to hold multiple processes in memory and switch the processor (CPU) from executing one process to another process
- Multitasking involves:
  - Context switching
    - The switching of execution context from task to other
  - Context saving
    - When a task/process switching happens, the current context of execution should be saved to (Context saving) retrieve it at a later point of time when needed.
  - Context retrieval
    - During context switching, the context of the task to be executed is retrieved from the saved context list



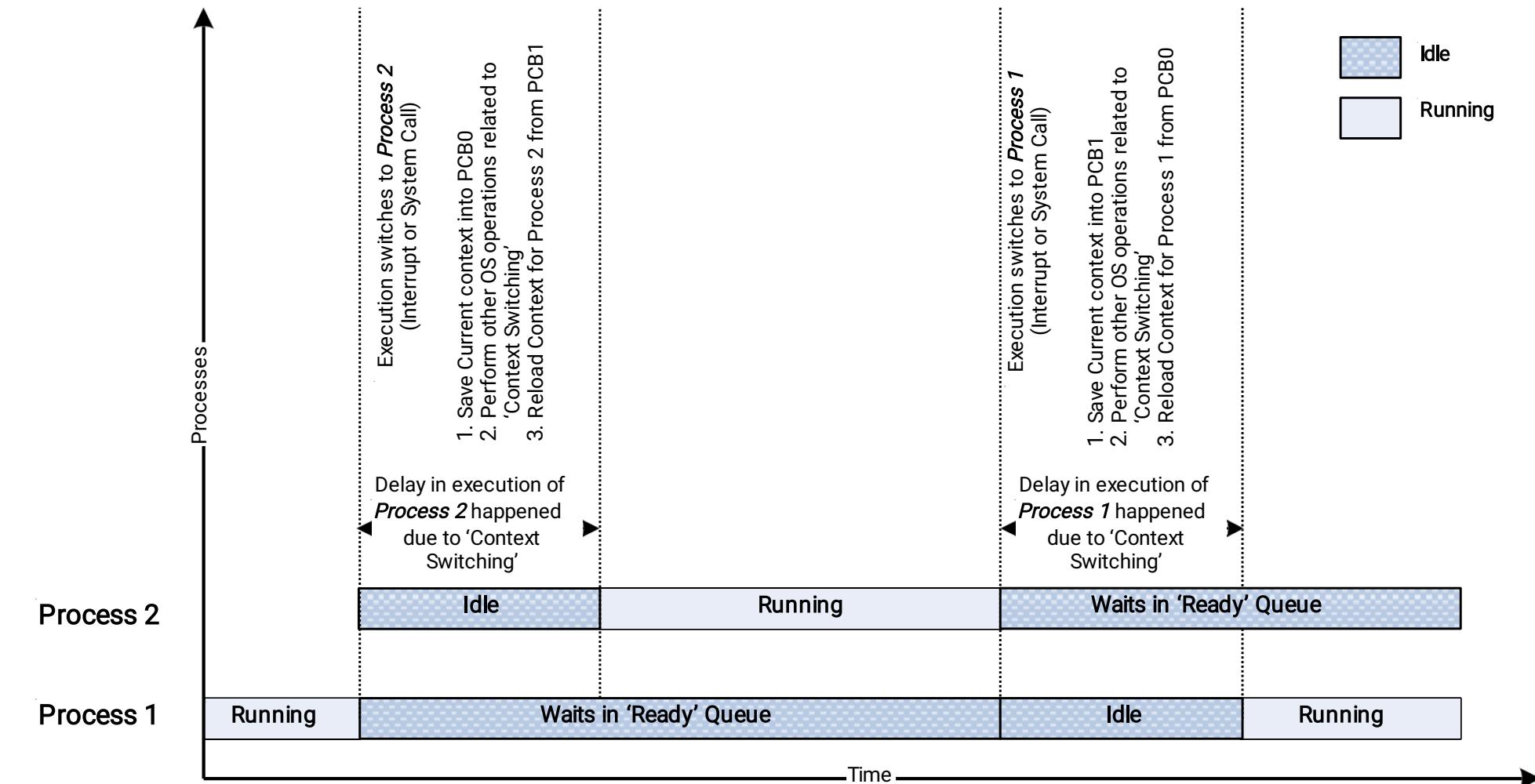
Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

# 10.4 Multiprocessing and Multitasking

## Multitasking: Context Switching





# 10.4 Multiprocessing and Multitasking

## Multitasking

### Types of Multitasking :

#### 1. Co-operative Multitasking:

- It is most primitive form of multitasking
- In which a task/process gets a chance to execute only when the currently executing task/process voluntarily relinquishes the CPU.
- In this method, any task/process can avail the CPU **as much time as it wants.**
- Since this type of implementation involves the **cooperation** of tasks with each other for **getting the CPU time for execution**
- If the currently executing task is **non-cooperative**,
  - the other tasks may have to wait for a **long time** to get the CPU
  - the other tasks may have to wait for a long time to get the CPU



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

# 10.4 Multiprocessing and Multitasking

## Types of Multitasking :

### 2. Preemptive Multitasking:

- Ensures that every task/process gets a chance to execute.
- When and how much time a process gets is dependent on the implementation of the preemptive scheduling.
- When and how much time a process gets is dependent on the implementation of the preemptive scheduling.
- As the name indicates, in preemptive multitasking, the currently running task/process is preempted to give a chance to other tasks/process to execute.
  - The preemption of task may be based on time slots or task/process priority



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

# 10.4 Multiprocessing and Multitasking

## Types of Multitasking :

### 3. Non-Preemptive Multitasking:

- = The process/task, which is currently given the CPU time, is allowed to execute until it terminates or enters the 'Blocked/Wait' state, waiting for an I/O.
- = The co-operative and non-preemptive multitasking differs in their behavior when they are in the 'Blocked/Wait' state.
  - In co-operative multitasking → the currently executing process/task need not relinquish the CPU when it enters the 'Blocked/Wait' state
  - In non-preemptive multitasking the currently executing task relinquishes the CPU when it waits for an I/O.



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

# 10.7 Task Communication



## Communication

- In a ~~multitasking system~~, **multiple tasks/processes run concurrently** and each process may or may not interact between.
- Based on the degree of interaction, the processes /tasks running on an OS are classified as:
  - **Co-operating Processes:** In the co-operating interaction model one process requires the inputs from other processes to complete its execution.
    - **Co-operation through sharing:** Exchanges data through some shared resources.
    - **Co-operation through Communication:** No data is shared between the processes. But they communicate for execution synchronization.
  - **Competing Processes:** The competing processes do not share anything among themselves but they share the system resources. The competing processes compete for the system resources such as file, display device etc



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

# 10.7 Task Communication

## Communication



- IPC refers to the mechanism through which tasks/processes communicate each other
- Implementation of IPC mechanism is OS kernel dependent
- Some important IPC mechanisms adopted by OS kernels are:
  - Shared Memory
    - Global Variables
    - Pipes (Named & Un-named)
    - Memory mapped Objects
  - Message Passing
    - Message Queues
    - Mailbox
    - Mail slot
    - Signals
  - Remote Procedure Calls (RPC)

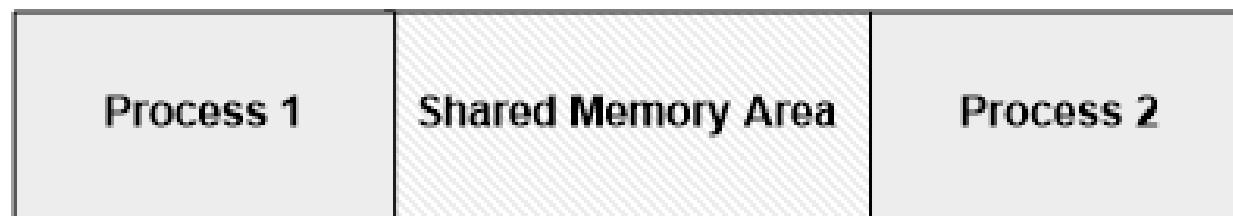
# 10.7 Task Communication

## Communication



### IPC – Shared Memory

- Processes share :
  - some area of the memory to communicate among them
  - Information to be communicated by the process is written to the shared memory area
- Processes which require this information can read the same from the shared memory area
- Same as the real world concept where a 'Notice Board' is used by the college to publish the information for students.



### Concept of Shared Memory



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt



# IPC – Shared Memory :

Pipe: → Used for  
communication  
Pipes

Pipes follow the client-server architecture.

- A process which creates a pipe is known as pipe server and a process which connects to a pipe is known as pipe client.
- It can be:
  1. Unidirectional: → allowing information flow in one direction
    - The process connecting at one end of the pipe to write to the pipe and the process connected at the other end of the pipe to read the data
  2. Bidirectional: → allowing bi-directional information flow.
    - A unidirectional pipe allows, whereas a bi-directional pipe allows both reading and writing at one end





# IPC – Shared Memory :

Pipe: → Are of two  
Types      Pipes

- **Anonymous Pipes:**
  - The anonymous pipes are unnamed.
  - Unidirectional pipes used for data transfer between two processes.
- **Named Pipes:**
  - Named pipe is a named.
  - Unidirectional or bi-directional pipe for data exchange between processes.
  - The process which creates the named pipe is known as pipe server.
  - A process which uses the named pipe is known as pipe client.



## IPC – Shared Memory : Memory Mapped Objects



- It is a shared memory technique
  - For allocating a shared block of memory which can be accessed by multiple process simultaneously.
- For a process:
  - Mapping object is created and physical storage for it is reserved and committed.
  - A process can map the entire committed physical area or a block of it to its virtual address space
- All read and write operation can be performed.
- Any process which wants to share data with other processes
  - Can map the physical memory area of the mapped object to its



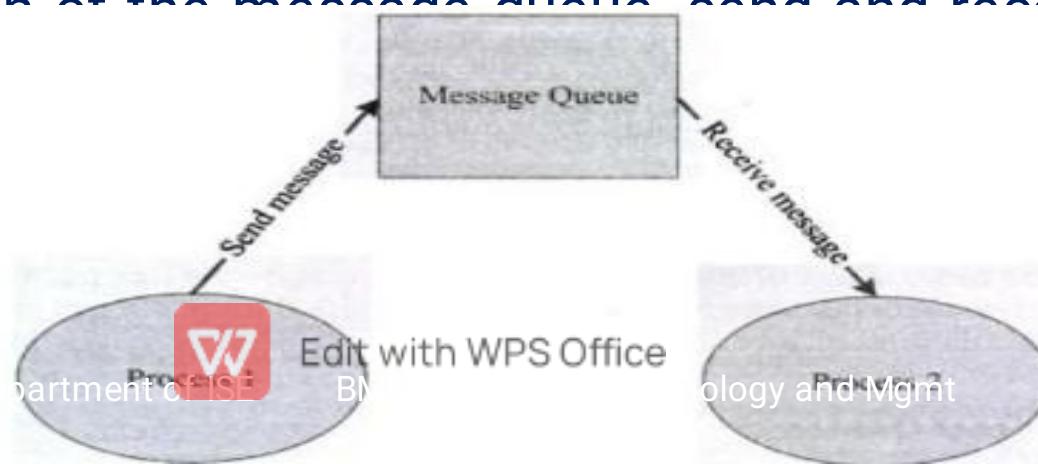
## 10.7 Task Communication

- Message Passing is a **synchronous / asynchronous** information exchange mechanism for Inter Process/ thread **Communication**
- Through shared memory lot of data can be shared: →
  - Whereas only limited amount of information / data is passed through message passing
- Message passing is relatively **fast and free from the synchronization overheads** compared to shared memory.
- Three ways it can be done:
  - Message Queue
  - Mailbox
  - Signalling

# Message Passing : Message Queue



- Process which wants to talk to another process posts the message to a First-In-First-Out (FIFO) queue called 'Message queue',
  - Which stores the messages temporarily in a system defined memory object, to pass it to the desired process
- Messages are sent and received through:
  - **Send:** → Name of the process to which the message is to be sent.
  - **Receive:** → Name of the process from which the message is to be received.
- The messages are exchanged through a message queue
- The implementation of the message queue send and receive methods are **OS kernel dependent**



# 10.7 Task Communication



## Message Passing

- Mailbox is a special implementation of **Mailbox** message queue
- Usually used for one way communication
- Only a **single message** is exchanged through mailbox whereas 'message queue' can be used for exchanging multiple messages
- One task/process creates **mailbox** and the other subscribe to this mailbox for getting message notification
- The implementation of the mailbox is OS kernel dependent

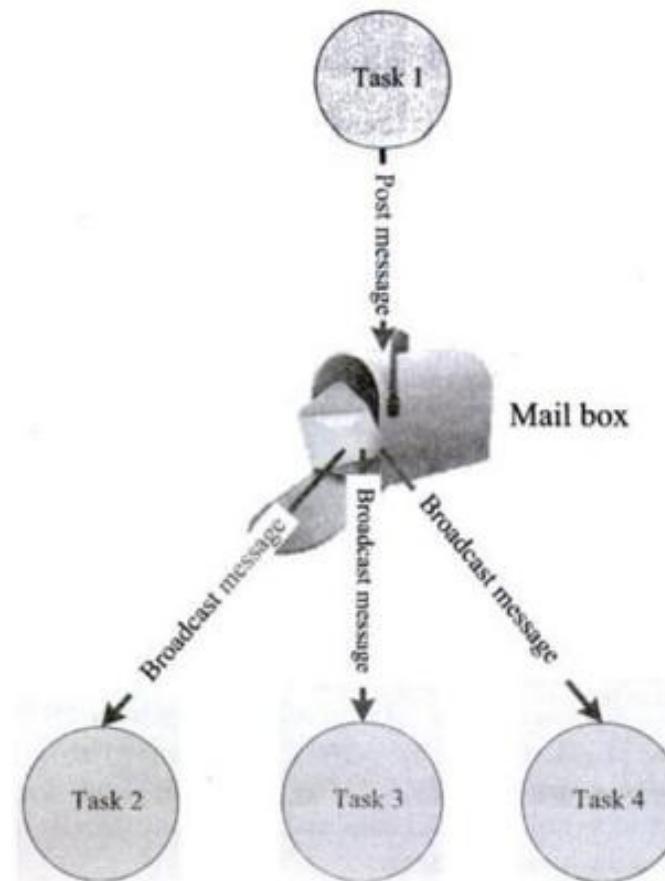


Fig. 10.21 Concept of Mailbox based indirect messaging for IPC



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

## Message Passing :

### Signalling

- An **asynchronous** notification mechanism
- Mainly used for the execution **synchronization** of tasks process/tasks
- Signal do **not carry any data** and are **not queued**
- The implementation of signals is OS **kernel** dependent
- A task/process can **create** a set of signals and register for it
- A task or Interrupt Service Routine (ISR) can signal a 'signal'
- Whenever a specified signal occurs it is handled in a **signal handler** associated with the signal



Edit with WPS Office

Department of ISE

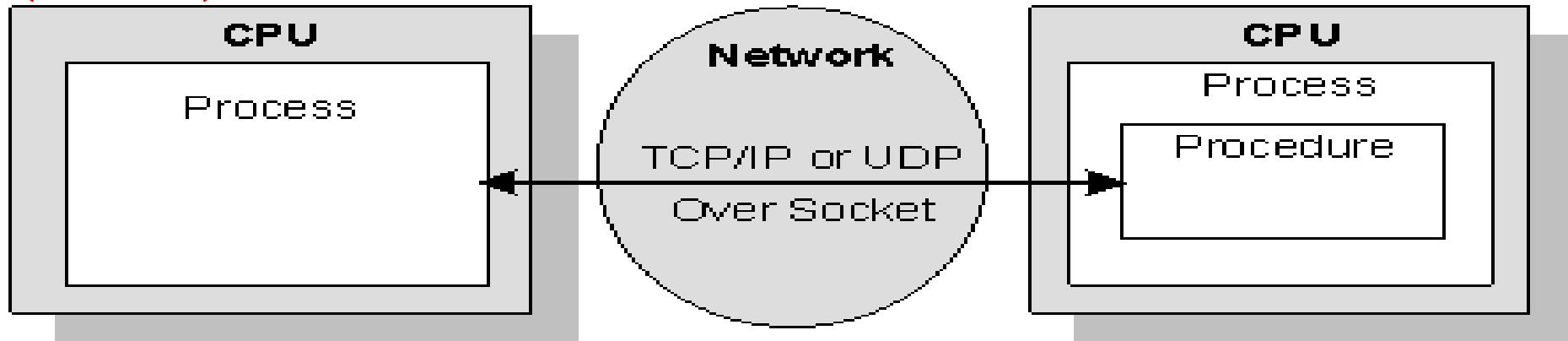
BMS Institute of Technology and Mgmt



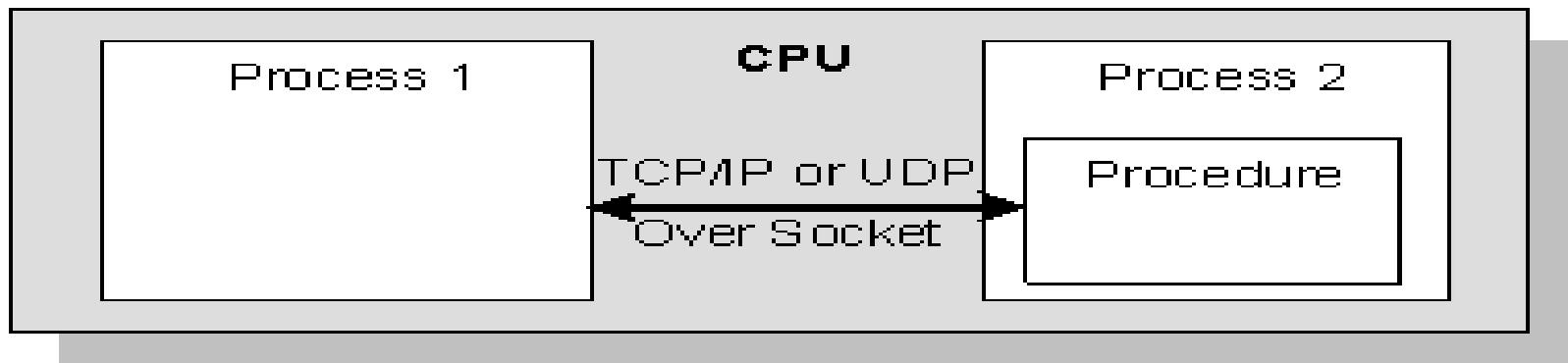
## IPC – Remote Procedure Call

- The IPC mechanism used by a process:  
**(RPC)** To call a procedure of another process running on the same CPU or on a different CPU which is interconnected in a network
- RPC is mainly used for **distributed** applications like client-server applications
- With RPC it is possible to communicate over a **heterogeneous** network
- The CPU/Process containing the procedure which needs to be **invoked** **remotely** is known as **server**
- The CPU/Process which **initiates an RPC request** is known as **client**
- In order to make the RPC communication compatible across all platforms it should stick on to **certain standard formats**
- Interface Definition Language (IDL) defines the interfaces for RPC
- Microsoft Interface Definition Language

## IPC – Remote Procedure Call (RPC)



**Processes running on different CPUs which are networked**



**Processes running on same CPU**

**Concept of Remote Procedure Call (RPC) for IPC**



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

# 10.10 How to Choose an RTOS



- The decision of an RTOS for an embedded design is very critical.
  - A lot of factors need to be analyzed carefully before making a decision on the selection of an RTOS.
- These factors can be either
- These factors can be either
  1. Functional Requirements.
  1. **Functional Requirements.**
  2. Non-functional Requirements.
  2. **Non-functional Requirements.**



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

# 10.10 How to Choose an RTOS Functional Requirements



## 1. Processor support:

- It is **not necessary** that all RTOS's support all kinds of processor architectures.
- It is essential to ensure the processor support by the RTOS

## 2. Memory Requirements:

- The RTOS requires ROM memory for holding the OS files and it is normally stored in a non-volatile memory like FLASH.
- OS also requires working memory RAM for loading the OS service.
- Since embedded systems are constrained, it is essential to evaluate the minimal RAM and ROM requirements for the OS under consideration.

## 1. Real-Time Capabilities:

- It is not mandatory that the OS for all embedded systems need to be Real- Time and all embedded OS's are 'Real-Time' in behavior.
- The Task/process scheduling policies plays an important role in the Real - Time

behavior of an OS

# 10.10 How to Choose an RTOS Functional Requirements



## 4. Kernel and Requirements

- The kernel of the OS may disable interrupts while executing certain services and it may lead to interrupt latency.
- For an embedded system whose response requirements are high, this latency should be minimal.

## 5. Inter process Communication (IPC) and Task Synchronization:

- The implementation of IPC and Synchronization is OS kernel dependent.

## 6. Modularization Support:

- Most of the OS's provide a bunch of features.
- It is very useful if the OS supports modularization where in which the developer can choose the essential modules and re-compile the OS image for functioning.

## 7. Support for Networking and Communication:

- The OS kernel may provide stack implementation and driver support for a bunch of communication interfaces and networking.
- Ensure that the OS under consideration provides support for all the interfaces required by the embedded product.

## 8. Development Language Support



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

# 10.10 How to Choose an RTOS Non-Functional Requirements



## RTOS Non-Functional Requirements

### 1. Custom Developed or Off the Shelf:

- It is possible to go for the complete development of an OS suiting the embedded system needs or use an off the shelf, readily available OS.
- It may be possible to build the required features by customizing an open source OS.
- The decision on which to select is purely dependent on the development cost, licensing fees for the OS, development time and availability of skilled resources.

### 1. Cost:

- The total cost for developing or buying the OS and maintaining it in terms of commercial product and custom build needs to be evaluated before taking a decision on the selection of OS.



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

# 10.10 How to Choose an RTOS Non-Functional Requirements

## Requirements



### 3. Development and Debugging tools Availability:

- The availability of development and debugging tools is a critical decision making factor in the selection of an OS for embedded design.
- Certain OS's may superior in performance, but the availability of tools for supporting the development may be limited.

### 3. Ease of Use:

- How easy it is to use a commercial RTOS is another important feature that needs to be considered in the RTOS selection.

### 4. After Sales:

- For a commercial embedded RTOS, after sales in the form of e-mail, on-call services etc. for bug fixes, critical patch updates and support for production issues etc. should be analyzed thoroughly.



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

# Integration and Testing of Embedded Hardware and Firmware



- 'Embedded Intelligence' is the process of Integrating hardware and firmware by:
  - Embedding the firmware into the target hardware board.
- Loading Firmware mainly deals with:
  - Non-Operating System based Embedded Products:
    - Processor/Controller has built in code memory:
      - Here firmware is loaded into the code memory before the usage.
      - Processor/Controller doesn't have built in code memory:
        - Operating System based Embedded Products:
          - They use Boot Loader.
        - Operating System based Embedded Products:
          - They use Boot Loader.
  - Loading Firmware mainly deals with:
    - Non-Operating System based Embedded Products:
      - Processor/Controller has built in code memory:
        - Here we use external memory like EPROM/FLASH to load firmware.
      - Processor/Controller doesn't have built in code memory:
        - Operating System based Embedded Products:
          - They use Boot Loader.



Edit with WPS Office

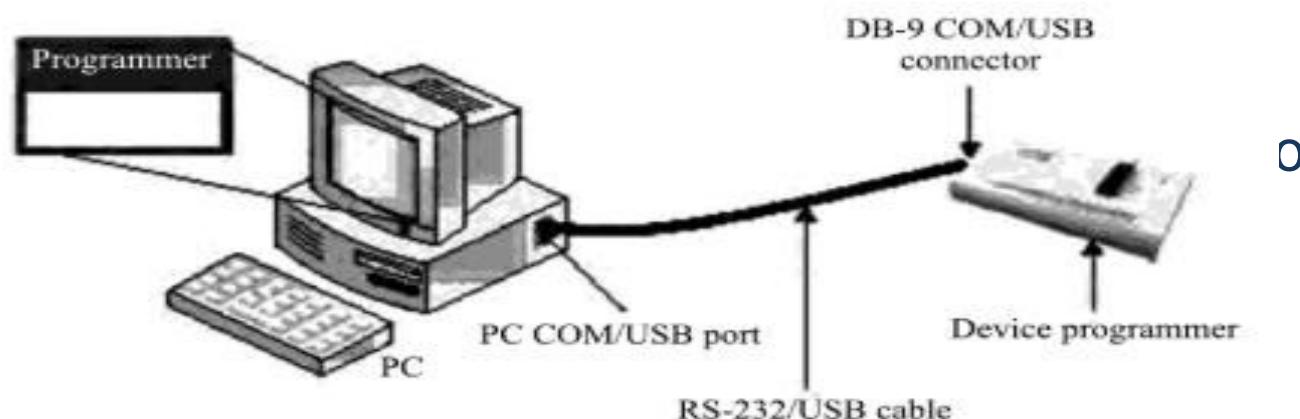
Department of ISE

BMS Institute of Technology and Mgmt

# Integration and Testing of Embedded Hardware and Firmware



- **Out-of-Circuit Programming:**
  - Here the firmware is programmed using a **programming device**.
    - i.e. the device to be programmed is removed from target board.
  - The **programming device** has necessary hardware to generate programming signals required.
  - The **programming device** has necessary hardware to generate programming signals required.
  - This programmer device is interfaced through RS-232/USB/Parallel Port Interface.
  - This programmer device is interfaced through RS-232/USB Port Interface.
  - The programmer device contains ZIF socket with locking pin to hold the device to be programmed.



# Integration and Testing of Embedded Hardware and Firmware



The sequence of operations for embedding the **firmware** with a programmer is listed below.

1. Connect the **programming device** to the specified port of PC (USB/COM port/parallel port)
2. Power up the **device** (Most of the programmers incorporate LED to indicate **Device** power up. Ensure that the power indication LED is ON)
3. Execute the programming utility on the PC and ensure proper connectivity is established between PC and programmer. In case of error, turn off **device** power and try connecting it again
4. Unlock the **ZIF socket** by turning the lock pin
5. Insert the **device** to be programmed into the open socket as per the insert diagram shown on the programmer
6. Lock the **ZIF socket**
7. Select the **device name** from the list of supported devices
8. Load the hex file which is to be **embedded** into the **device**
9. Program the **device** by 'Program' option of utility program
10. Wait till the completion of programming operation (Till busy LED of programmer is off)
11. Ensure that programming is successful by checking the status LED on the programmer (Usually 'Green' for success and 'Red' for error condition) or by noticing the feedback from the utility program
12. Unlock the **ZIF socket** and take the **device** out of programmer



Edit with WPS Office

Department of ISE

BMS Institute of Technology and Mgmt

# Integration and Testing of Embedded Hardware and Firmware



- **In System Programming (ISP):**
  - Here Programming is done 'within the system'
  - It is the most easiest way of embedding firmware, the only pre-requisite is that the target board should have ISP support.
  - Along with target board it requires PC, ISP cable, ISP utility:
    - Chips supporting ISP generates necessary programming signals.
    - Target board is interfaced to PC using Serial port/ Parallel Port/ USB.
    - The communication between target device and ISP utility is through serial protocols:
      - JTAG: Joint Test Action Group.
      - Serial Peripheral Interface (SPI)
    - In order to perform ISP operations the target board should be ISP mode:
      - The device receives commands, data or reprograms of code.
      - The device receives commands, data or reprograms of code.
    - Once ISP operations are complete, the device is re-configured by reset.
    - Once ISP operations are complete, the device is re-configured by reset.

# Integration and Testing of Embedded Hardware and Firmware



- **In Application Programming (IAP):**
  - IAP is a technique used by the firmware running on the target device for modifying a selected portion of code memory.
  - It deals with updating calibration data, look up tables etc. which are stored in code memory.
  - It deals with updating calibration data, look up tables etc. which are stored in code memory.
  - The Boot ROM consists of API instructions required for programming, erasing, reading Flash memory.
  - The Boot ROM consists of API instructions required for programming, erasing, reading Flash memory.
    - Thus end-user applications can perform operations on Flash memory.
    - Here any updating is done just like function call.
  - The Boot ROM is controlled by status bit which decides the access to code memory by Boot ROM/ user code memory:
    - Status bit → Set : Access by Boot ROM
    - Status bit → Cleared : Access by user code memory.
  - The Boot ROM is controlled by **status bit** which decides the access to code memory by Boot ROM/ user code memory.
    - Status bit → Set : Access by Boot ROM

# Integration and Testing of Embedded Hardware and Firmware



- **Use of Factory Programmed chip:**
  - Here the Firmware is embedded to the target processor/controller at the time of Chip Fabrication.
  - at the time of Chip Fabrication.
  - Once the firmware design is over , we can send the design to chip fabricator to embed it into the code memory.
  - to chip fabricator to embed it into the code memory.
  - They are expensive.
  - They are expensive.

# Integration and Testing of Embedded Hardware and Firmware



- **Firmware Loading for Operating System Based Devices:**
  - This embedded system are programmed using ISP technique.
    - This embedded system are programmed using ISP technique.
    - It uses 'Boot Loader' which takes control of:→
      - Embedding Firmware
      - Embedding Firmware
      - Copying OS image into RAM
    - The Boot loader will be pre-loaded into embedded system / can be loaded through JTAG.
    - The Boot Loader deals with :→
      - Initializing the supporting interfaces like UART/I2C, TCP/IP.
      - Initializing the supporting interfaces like UART/I2C, TCP/IP.
      - In network based system, it broad casts the target presence over the network.



**Mrs. Ashwini N**  
Assistant Professor  
Dept.of.ISE  
BMSIT&M, Bengaluru Email:  
ashwinilaxman@bmsit.in