

# EMBEDDED SYSTEMS

**G.PRABHAKAR**  
**Assistant Professor,**  
**Syed Ammal Engineering College.**

# Agenda

## **Embedded System Basics – PART 1**

- Introduction to Embedded systems
- Embedded Processors & their Architectures
- Serial communication
- RTOS Concepts
- Videos of Embedded applications

# Embedded Programming in C- PART 2

- Embedded Software Development Process and Tools
- Embedded programming Demo for PIC Microcontroller & simulation using Proteus & MPLAB IDE
  - How to write a program
  - How to handle the datasheets of the processors
- Embedded programming Demo for MSP430 by using Code Composer Studio
  - How to write a program
  - How to handle the datasheets of the processors

# System Definition



- A way of working, organizing or performing one or many tasks according to a fixed set of rules, program or plan.
- Also an arrangement in which all units assemble and work together according to a program or plan

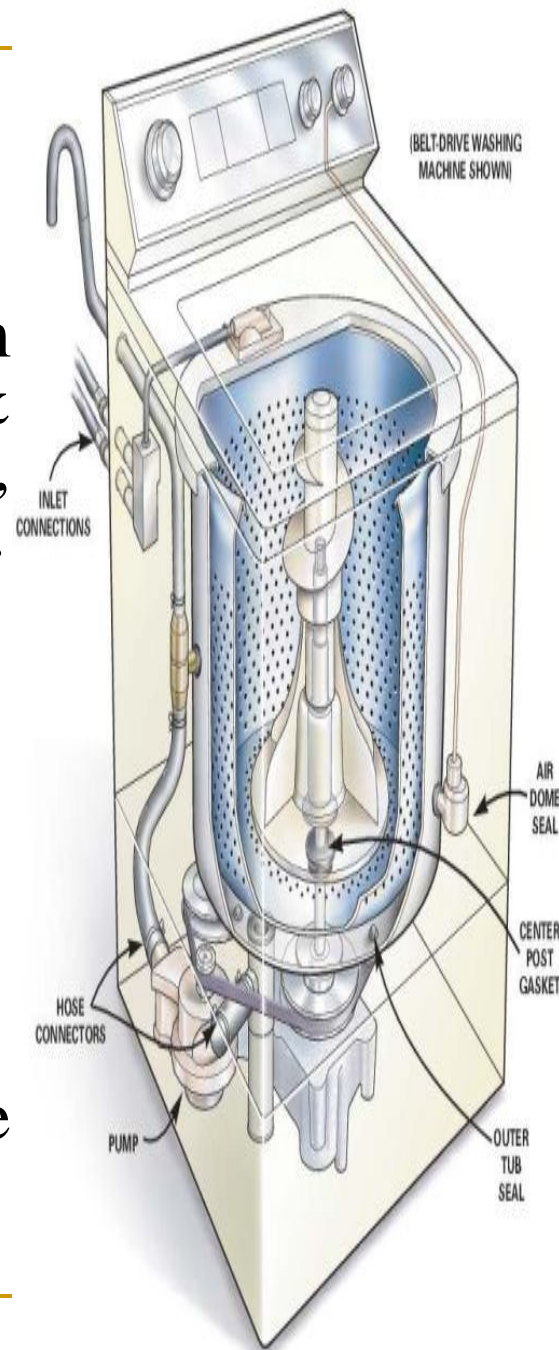
Examples:

- Time display system – A watch
- Automatic cloth washing system- A washing machine



# System Examples

- It is an automatic clothes washing system  
Parts: Status display panel, Switches & Dials, Motor, Power supply & control unit, Inner water level sensor and solenoid valve.
- Process:
  1. Wash by spinning
  2. Rinse
  3. Drying
  4. Wash over by blinking
  5. Each step display the process stage
  6. In case interruption, execute only the remaining



# Embedded Systems- Definition

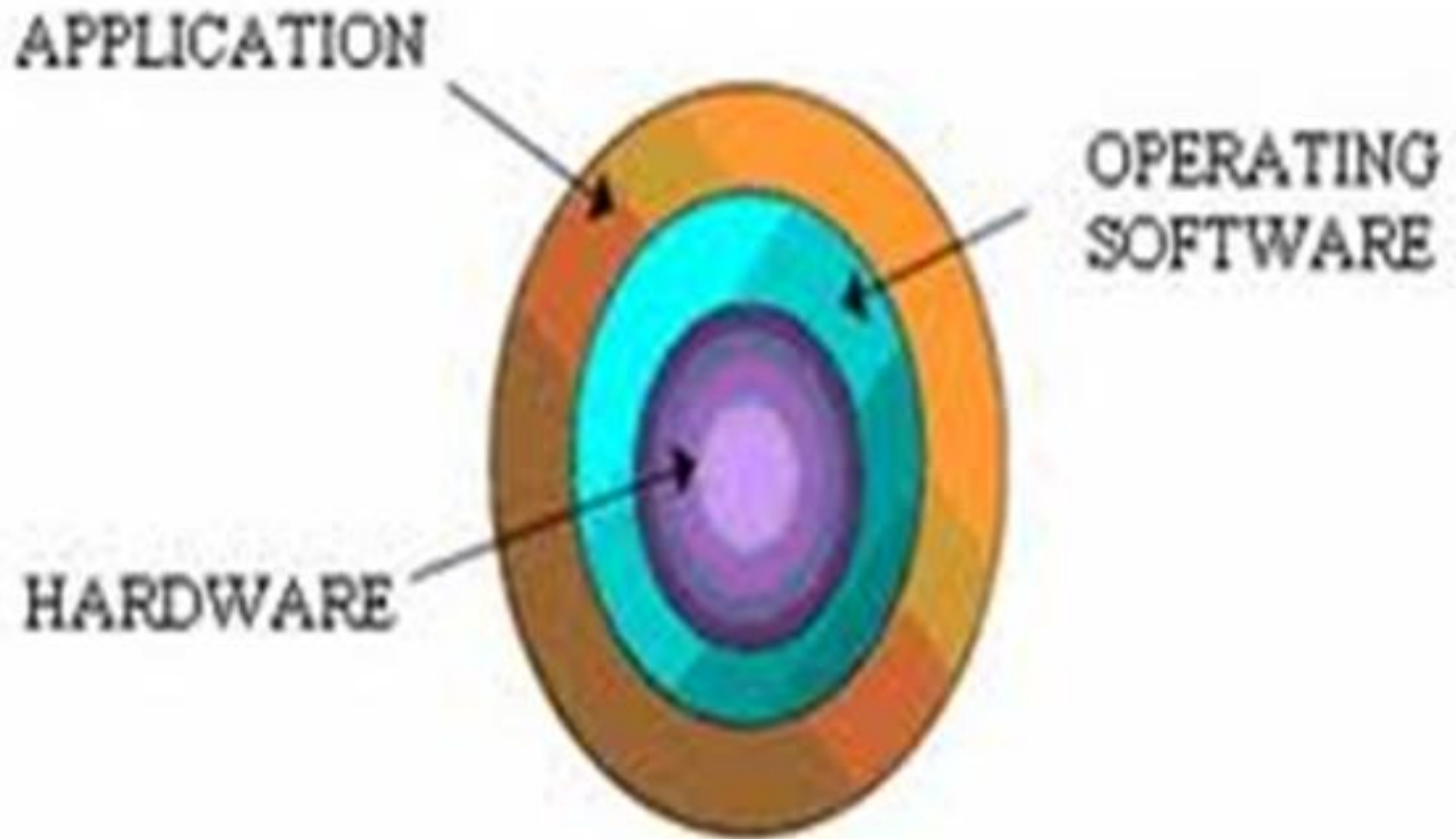
- “An embedded system is a system that has software embedded into computer-hardware, which makes a system dedicated for an application(s) or specific part of an application or product or part of a larger system.”
- It is any device that includes a programmable computer but is not itself intended to be a general purpose computer.”



**SOFTWARE PROGRAM**

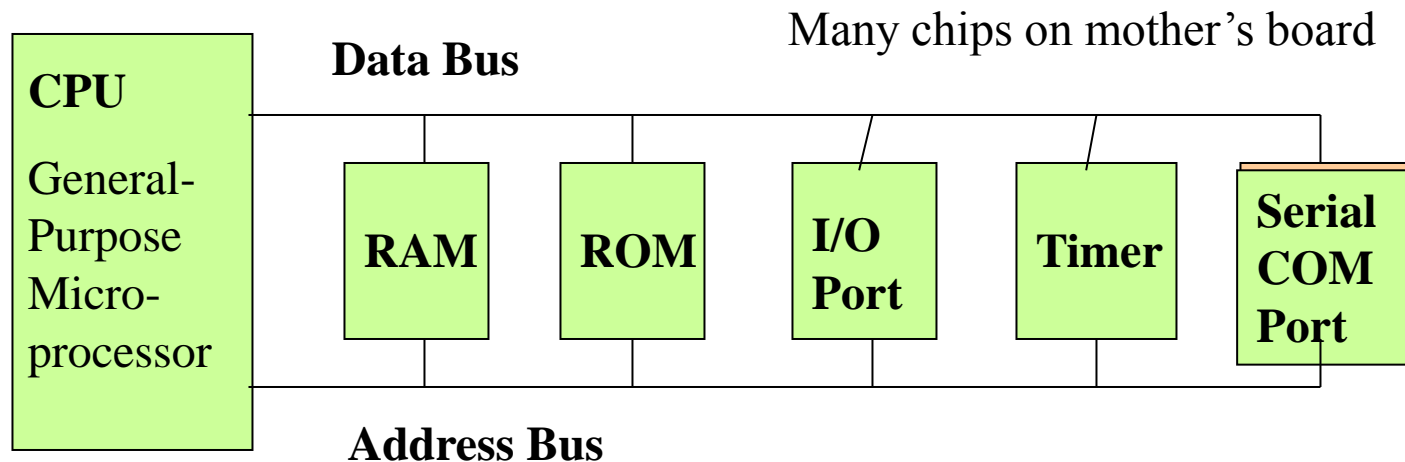
```
#include <16f876a.h>
#use delay (clock=20000000)
#byte PORTB=6
main()
{
  set_tris_b(0);
  portb=255;      //decimal
  delay_ms(1000);
  portb=0x55;     //hexadecimal
  delay_ms(1000);
  portb=0b10101010; //binary
  delay_ms(500);
}
```

# Embedded Systems- Architecture



# Microprocessors

- CPU for Computers
- No RAM, ROM, I/O on CPU chip itself
- Example : Intel's x86, Motorola's 680x0

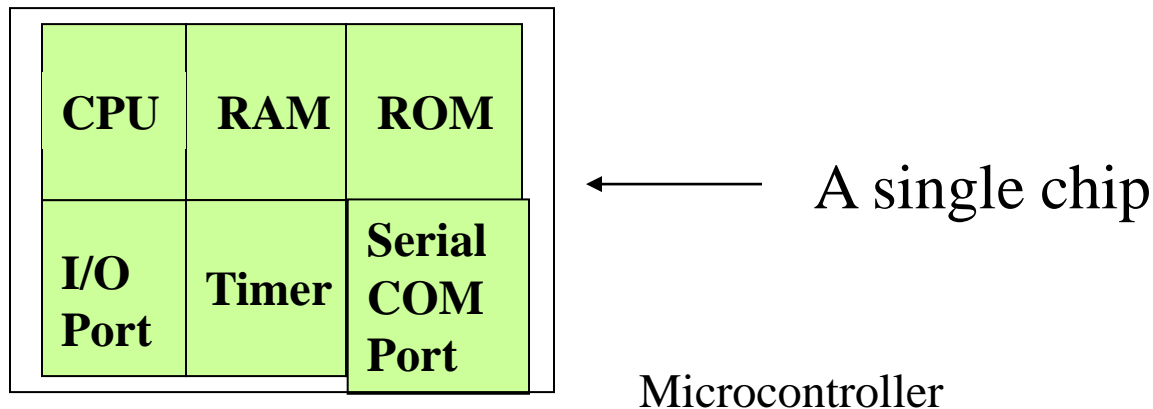


General-Purpose Microprocessor System



# Microcontrollers

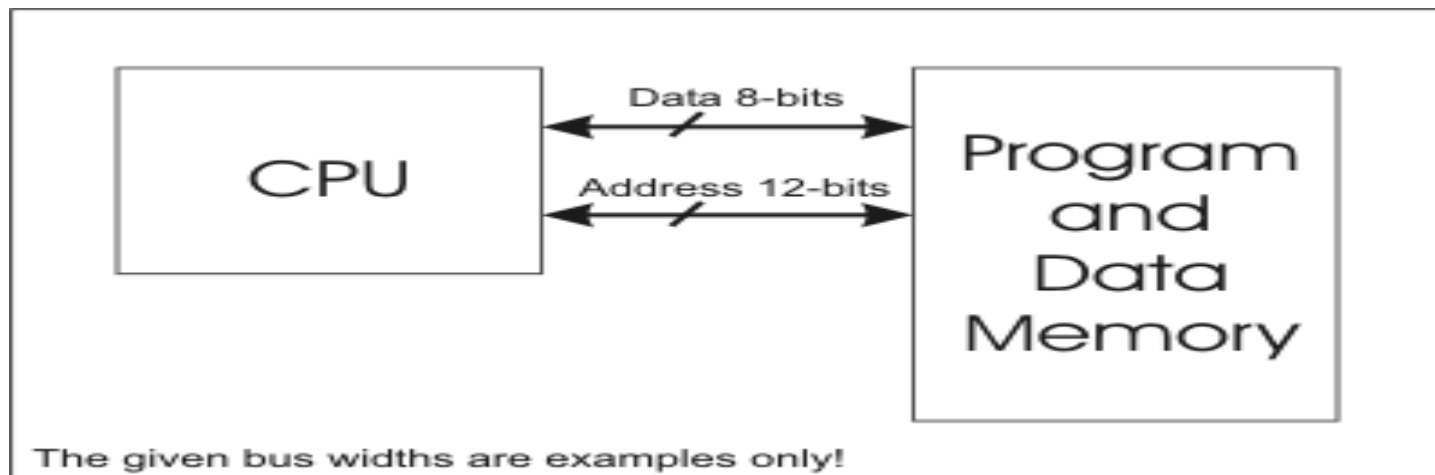
- A smaller computer
- On-chip RAM, ROM, I/O ports...
- Example : Motorola's 6811, Intel's 8051, Zilog's Z8 and PIC 16X



# Processor Architectures

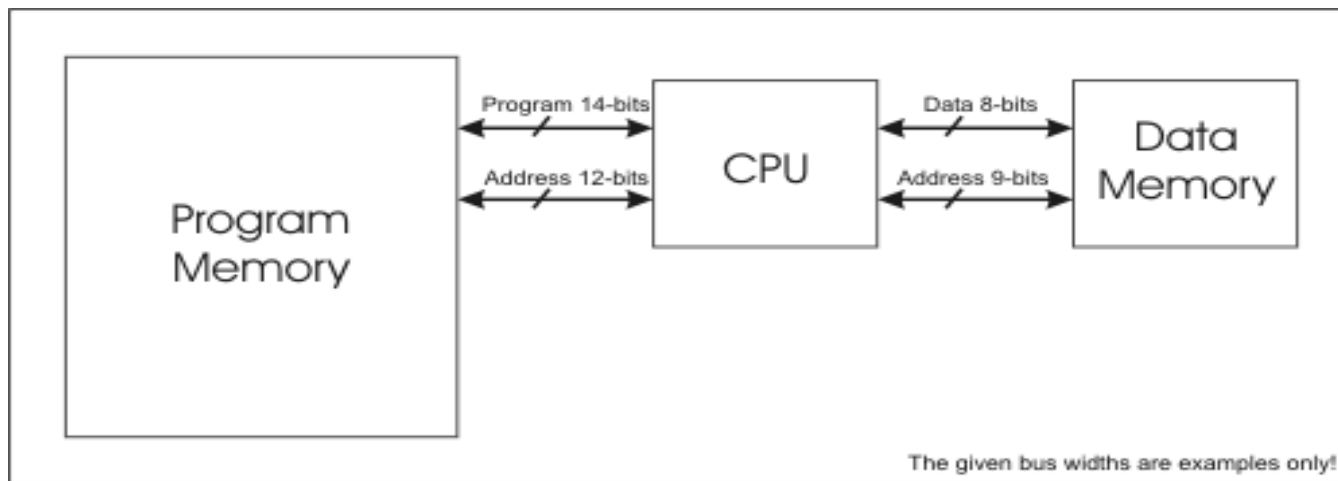
## 1-Von Neumann Architecture

- Single memory & single bus for transferring the data into & out of the CPU.
- Multiplication of two numbers require atleast 3 clk cycles

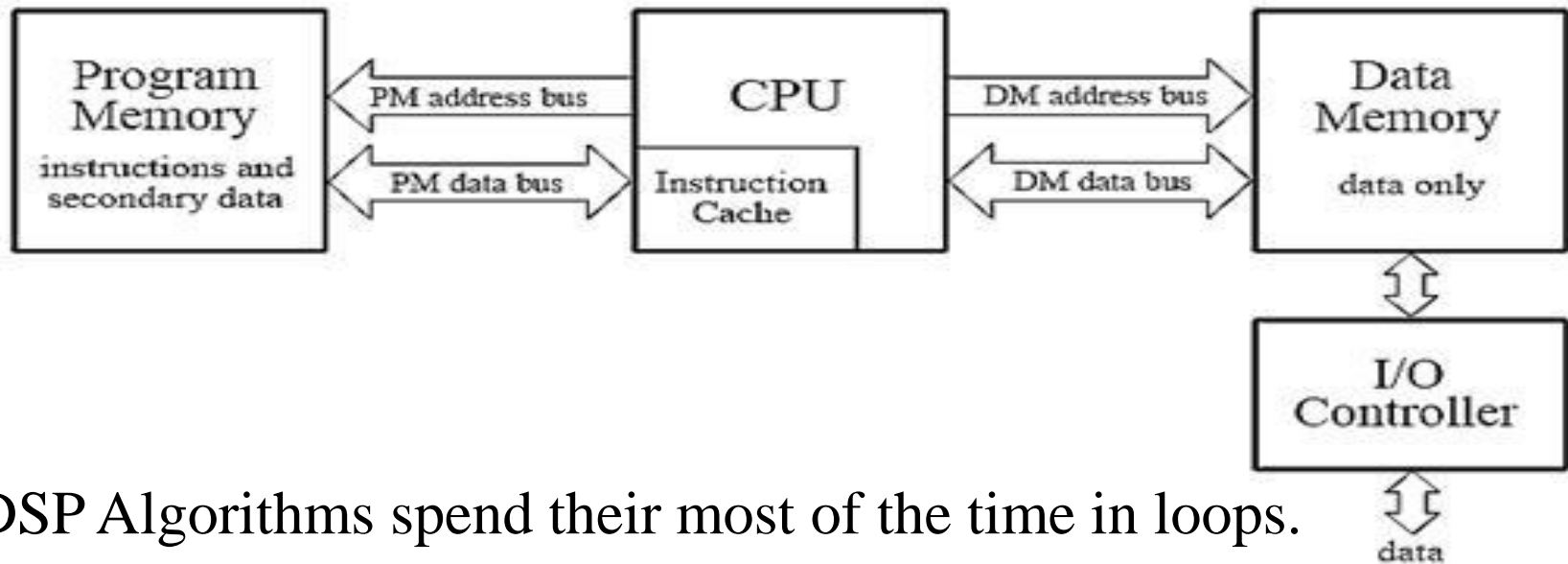


## 2-Harvard Architecture

- Separate memory for data & program with separate buses for each
- Both program instructions & data can be fetched at the same time
- Operational speed is higher than Von Neumann

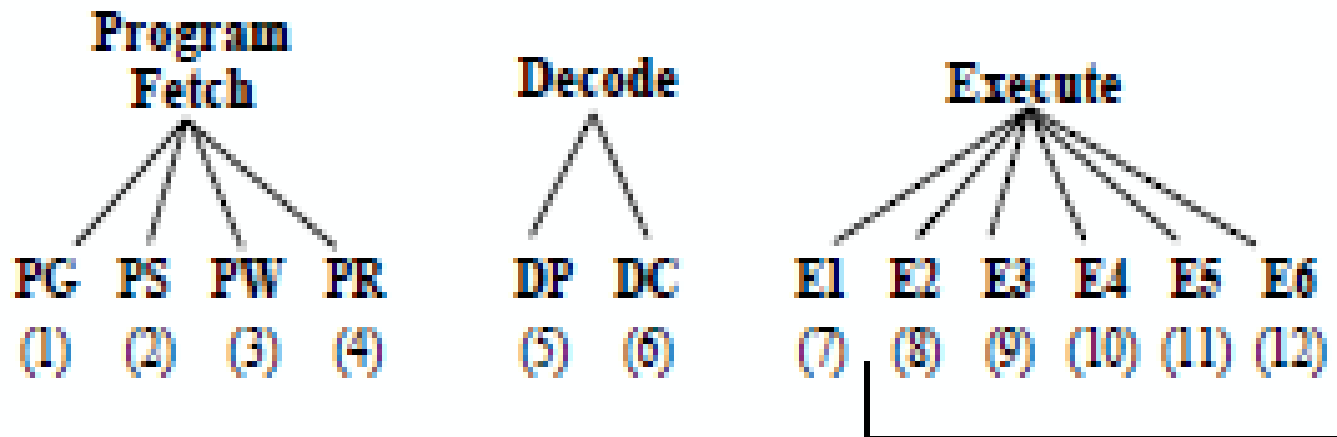


### 3-Super Harvard Architecture (DSP Processors)



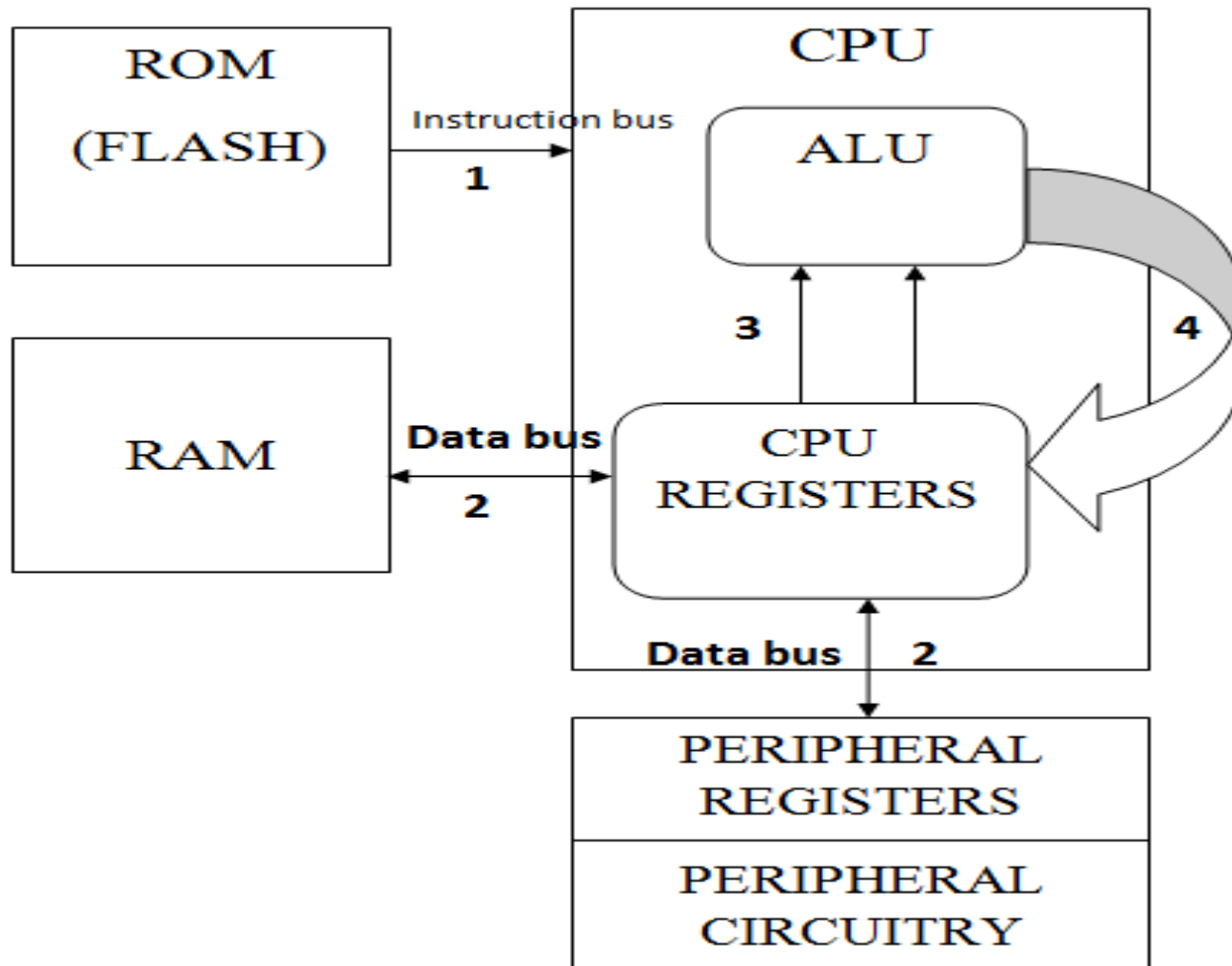
- DSP Algorithms spend their most of the time in loops.  
So instruction Cache is added
- Processing of many “high” frequency signals in real time
- Multiply–Accumulate operation 
$$a \leftarrow a + (b \times c)$$
- Fixed point/ floating point, Matrix operations, convolution , correlation, parallelism etc.  
eg: C5000, C6000 single core processor families

# DSP Pipeline Architecture



C6000 fetch eight 32 bit instructions per clock cycles

# Mechanism in Microcontrollers



# Difference between DSP & $\mu$ C

- DSPs often don't have a flash program memory. They need the software to be 'loaded' into them. Whereas, microcontrollers have a non power off erasable program memory inside, some with EPROM store capabilities.
- DSPs are much faster for integer mathematics operations, whereas many microcontrollers do not have the hardware.
- DSPs are much faster for floating point operations. In microcontrollers, this has to be done in software.
- DSPs are oriented to be an input/output device with 'fast calculating machine'. Microcontrollers are a multi-feature device with several ways of interfacing with the world, however none are the fastest.

- 
- DSPs are not designed to be a 'robust' device. They need a well designed board to work properly. Microcontrollers can work on a Test Board.
  - DSPs are a fast calculator microprocessor, that is very effective for computing calculations and moving data, whereas, microcontrollers are a more flexible device with more features.

Eg: 1) Image, audio & video processing by DSP

2) Data-transmission & control signals by Microcontroller



# Digital Signal Controllers

(Combined Processor Architectures)

Texas Instruments (DSP+ARM processor)

- Davinci- (video streaming applications )
- Multicore processors
- OMAP application processors (Image processing)

Analog Devices

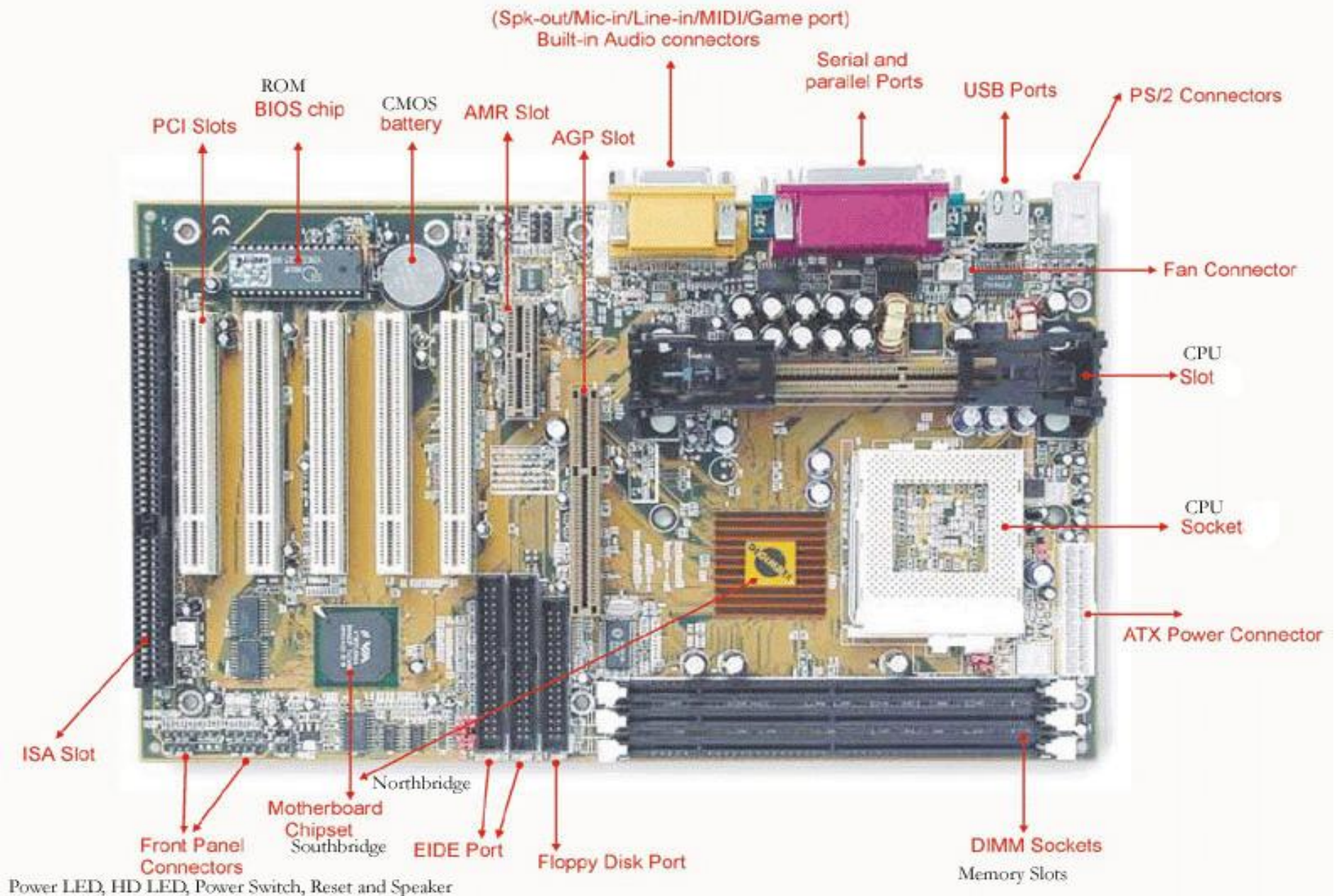
- Black-fin Processor (DSP+ARM)

# How the PC Starts

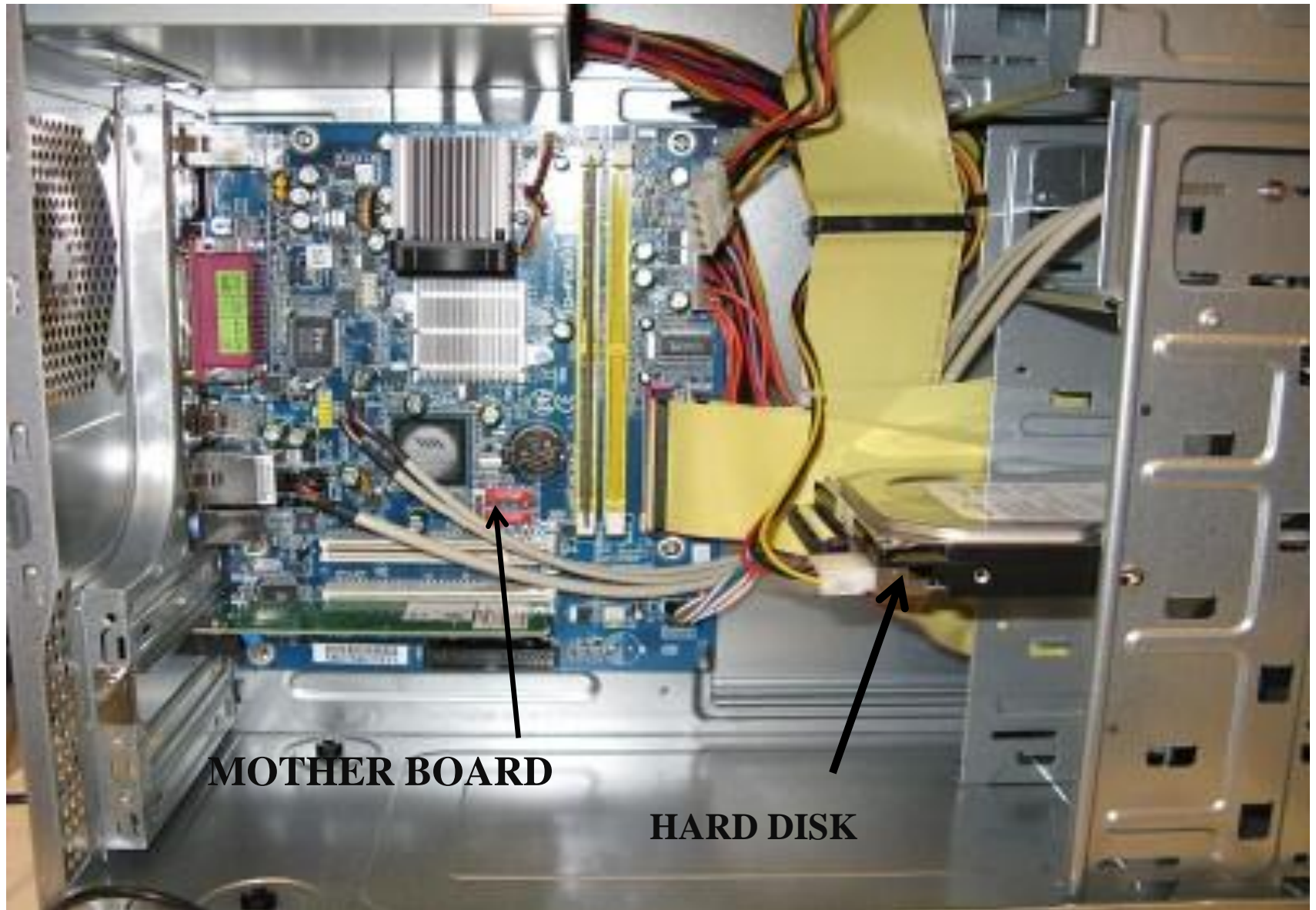
- Microcontroller searches its first instruction to execute
  - BIOS in Flash chip provides the Initial instructions to the  $\mu$ c
- BIOS does several sequences:
1. Check the CMOS Setup for custom settings
    - CMOS setup present in Nonvolatile RAM (NVRAM) or CMOS RAM
    - CMOS RAM & RTC integrated as a Southbridge Chipset, which is powered by CMOS battery (CMOS RAM-64 to 512 bytes capacity)
    - CMOS setup consists of sys time/date, Chipsets status, memory management

2. Load the interrupt handlers and device drivers
  3. Initialize registers and power management
  4. Perform the power-on self-test (POST)
  5. Display system settings
  6. Determine which devices are bootable
  7. Initiate the bootstrap sequence – the order to Boot OS
- Cold boot – restart PC with the power being off
- Warm boot or normal boot- restart PC when the power is ON









# VARIOUS MICROCONTROLLERS

- INTEL

8031,8032,8051,8052,8751,8752, Pentium etc

- PIC

8-bit PIC16, PIC18, 16-bit DSPIC33 / PIC24,  
PIC16C7x

- Motorola

MC68HC11

- ARM – ARM7, ARM9, ARM 11

# Types of Processors

---

# EMBEDDED PROCESSOR

- Special microprocessors & microcontrollers often called, Embedded processors.
- An embedded processor is used when fast processing fast context-switching & atomic ALU operations are needed.
- Examples : ARM 7, INTEL i960, AMD 29050.



# DIGITAL SIGNAL PROCESSOR

- DSP as a GPP is a single chip VLSI unit.
- It includes the computational capabilities of microprocessor and multiply & accumulate units (MAC).
- DSP has large number of applications such as image processing, audio, video & telecommunication processing systems.
- It is used when signal processing functions are to be processed fast.
- Examples : TMS320Cxx, SHARC, Motorola 5600xx

# APPLICATION SPECIFIC SYSTEM PROCESSOR (ASSP)

- ASSP is dedicated to specific tasks and provides a faster solution.
- An ASSP is used as an additional processing unit for running the application in place of using embedded software.
- Examples : IIM7100, W3100A

# APPLICATIONS

- **Household appliances:**  
**Microwave ovens, Television, DVD**
- **Players & Recorders**  
**Audio players**
- **Integrated systems in aircrafts and missiles**
- **Cellular telephones**
- **Electric and Electronic Motor controllers**
- **Engine controllers in automobiles**
- **Calculators**
- **Medical equipments etc...**



TELEVISION

REMOT CONTROL



REFRIGERATORS



ELEVATORS

VIDEO GAMES



SET-TOP BOX



PLANES



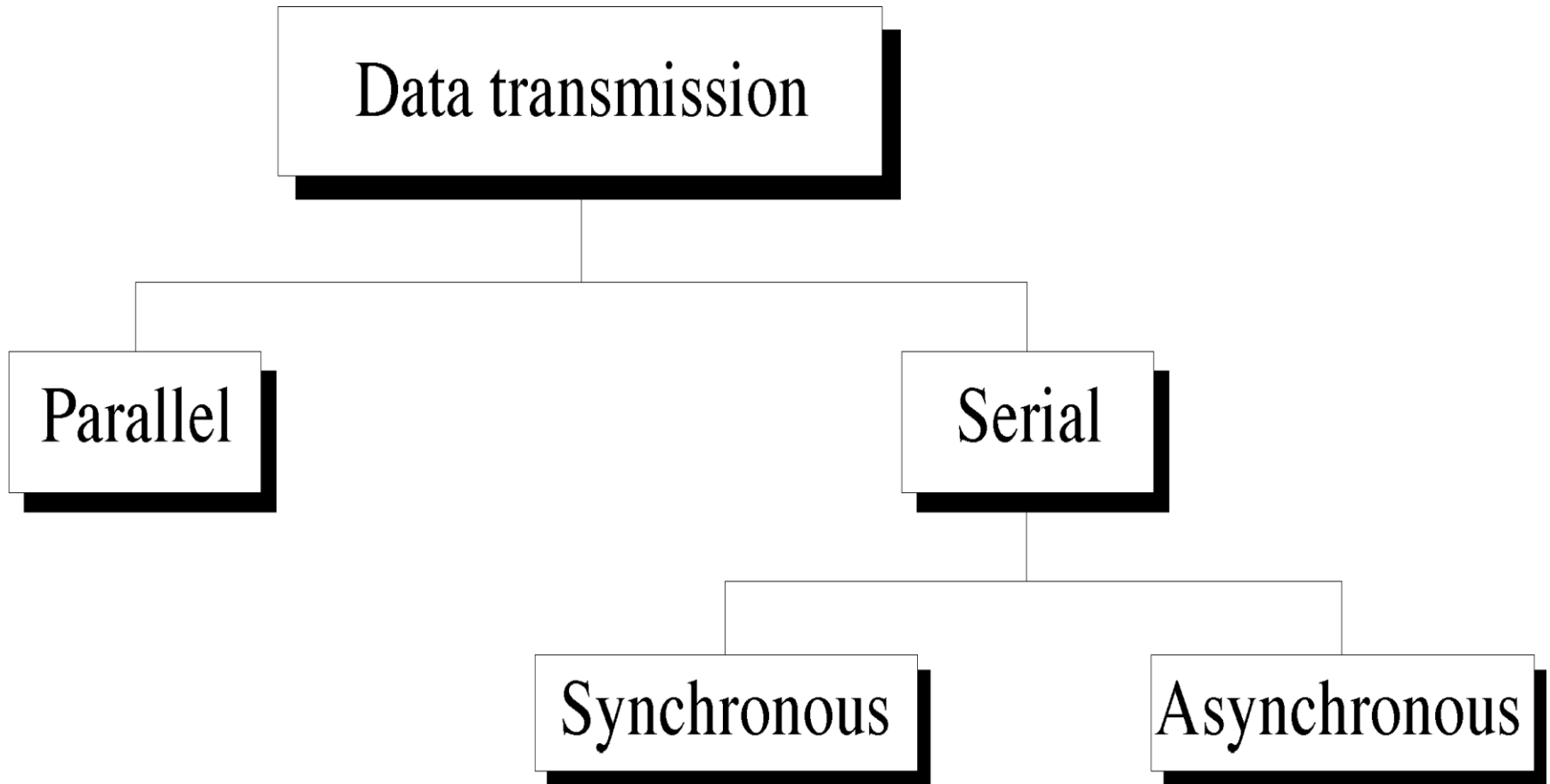
CARS



# Automobiles



2002: Opel Vectra has over 40 sensors (25 types)



# SERIAL COMMUNICATION

---

## SERIAL PORT:

- one bit can communicate & the bits transmitted at periodic intervals generated by a clock.
- Short and long distance communication.

## PARALLEL PORT:

- Multiple bits can communicate over a set of parallel lines at any given instance.
- Communication within the same board.
- Short distance communication.

# Serial communication

Mainly divided into

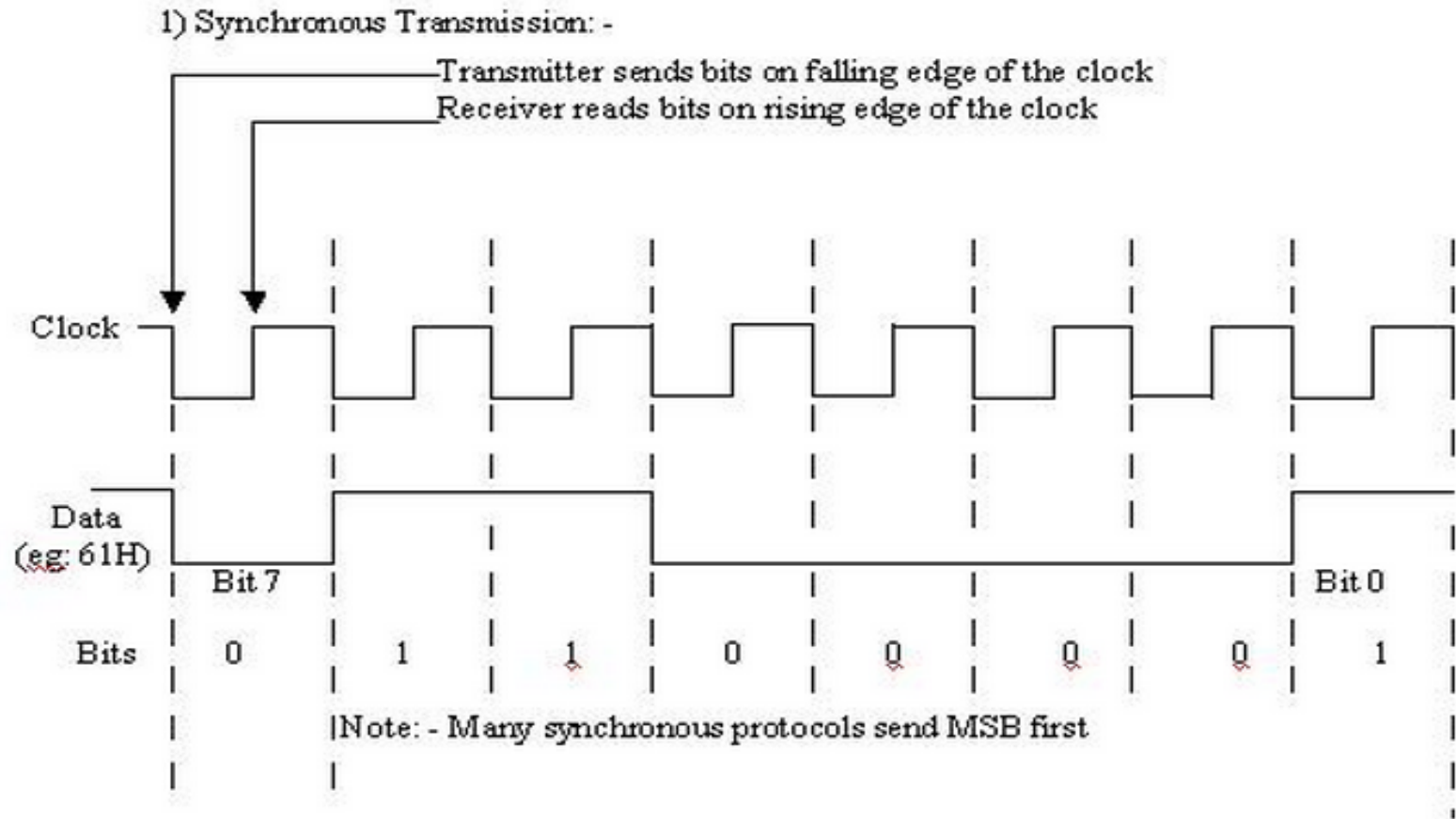
- Synchronous- Data is received or transmitted at constant time intervals with uniform phase differences. Eg: audio inputs, frames sent over a LAN
- Asynchronous- Data is received or transmitted at variable time intervals. Eg: keypad communication, Modem.



# DIFFERENCE

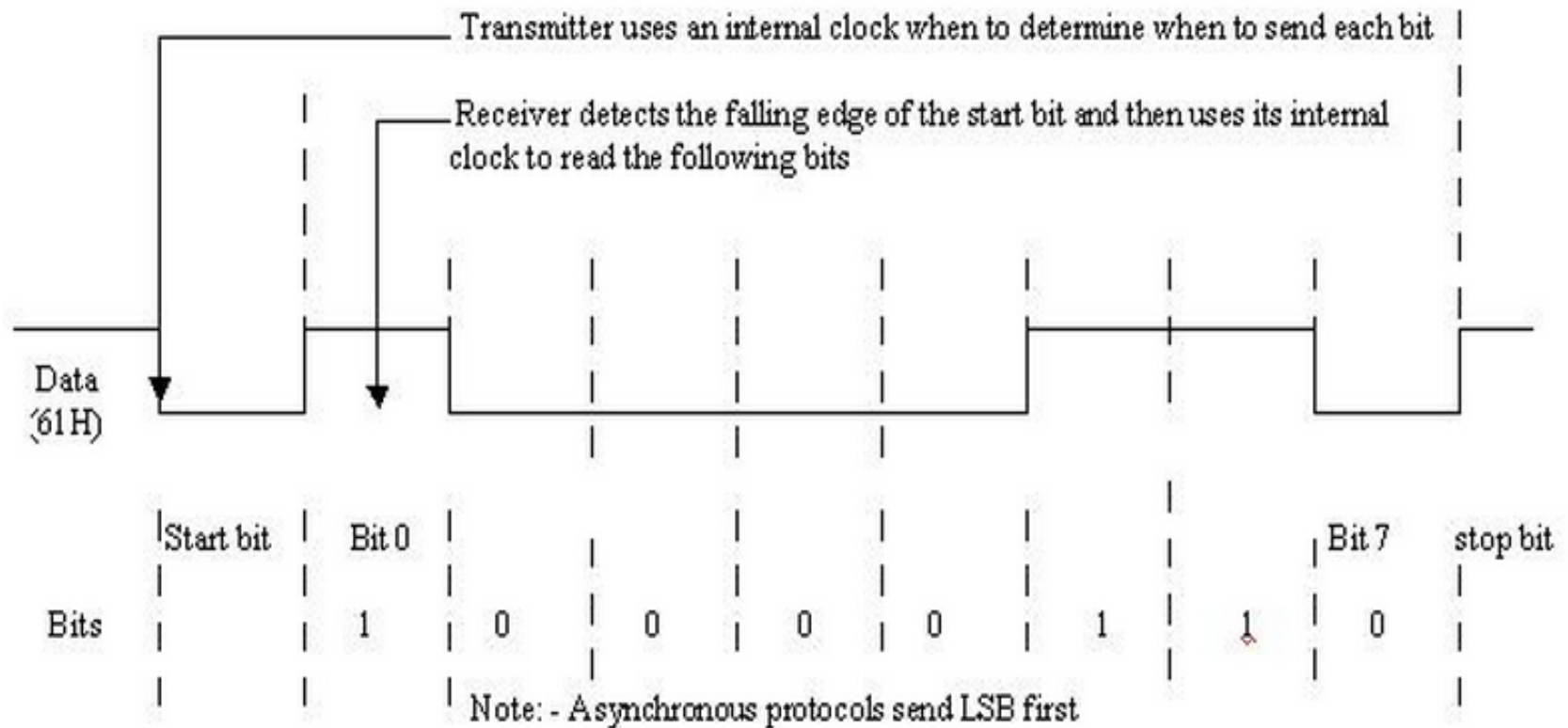
Synchronous	Asynchronous
Constant time intervals	Random time intervals
No Handshaking between Tx & Rx	Handshaking between Tx & Rx
Clock is known to the receiver explicitly	Clock is known to the receiver implicitly
Eg: SPI, SCI, SI and SDIO	Eg: RS232, UART, HDLC etc

# Synchronous Transmission



# Asynchronous Transmission

## 2) Asynchronous Transmission: -

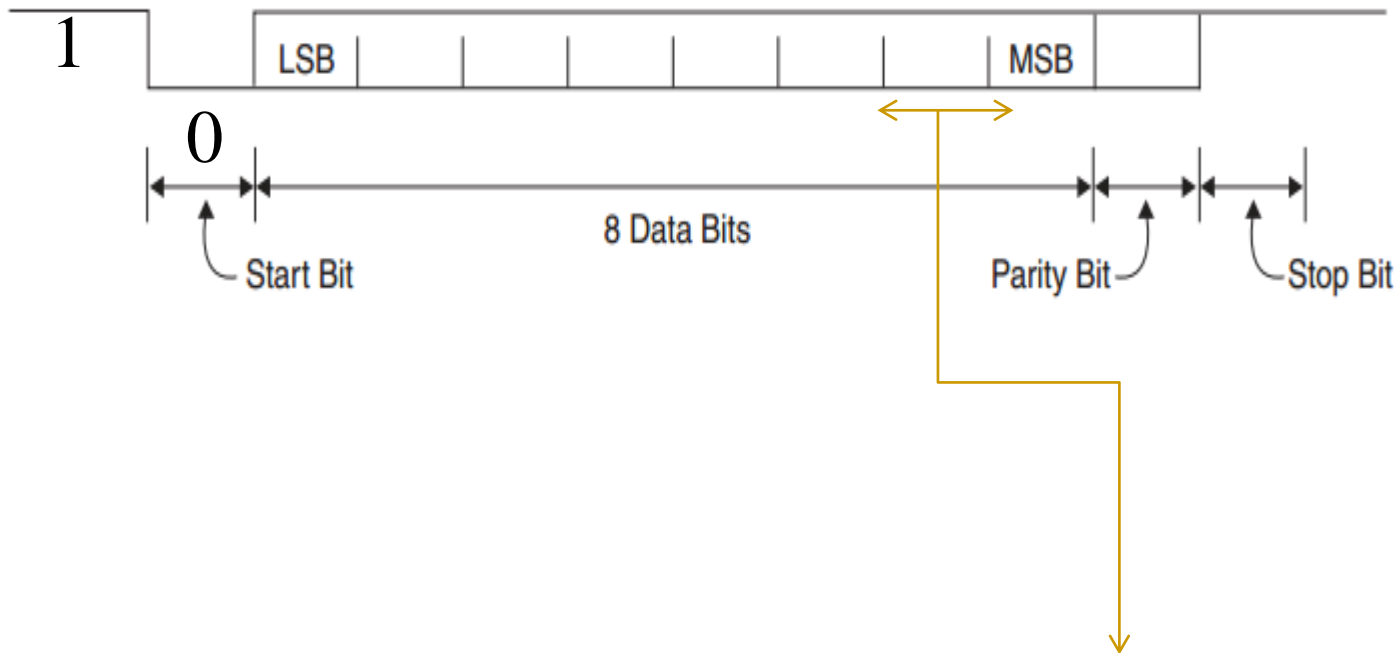


# What is the difference between RS232 and UART?

- RS232 is a specification for serial communications between a DCE and DTE it defines electrical characteristics, the 25-way 'D' connector and the various functions of the various signal lines.
- A UART is a **U**niversal **A**ynchronous **R**eceiver and **T**ransmitter - it is an electronic circuit which handles communication over an asynchronous serial interface - very often an RS232 interface.

# UART Frame format

1 → 0 transition

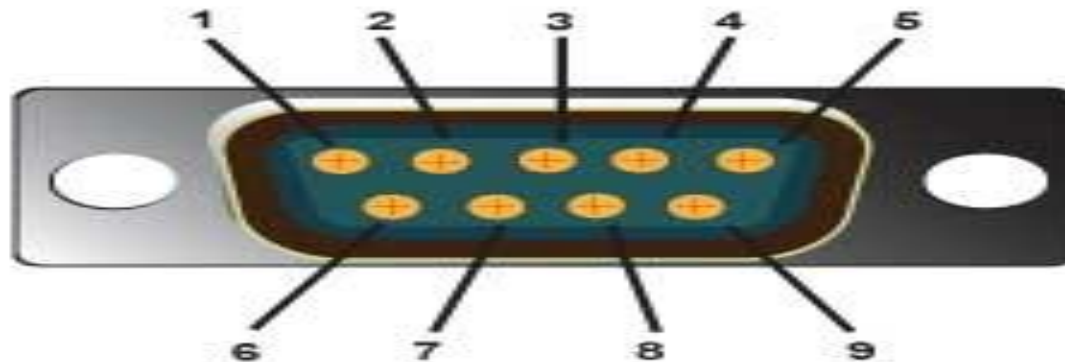


1bit time= 1/baud rate

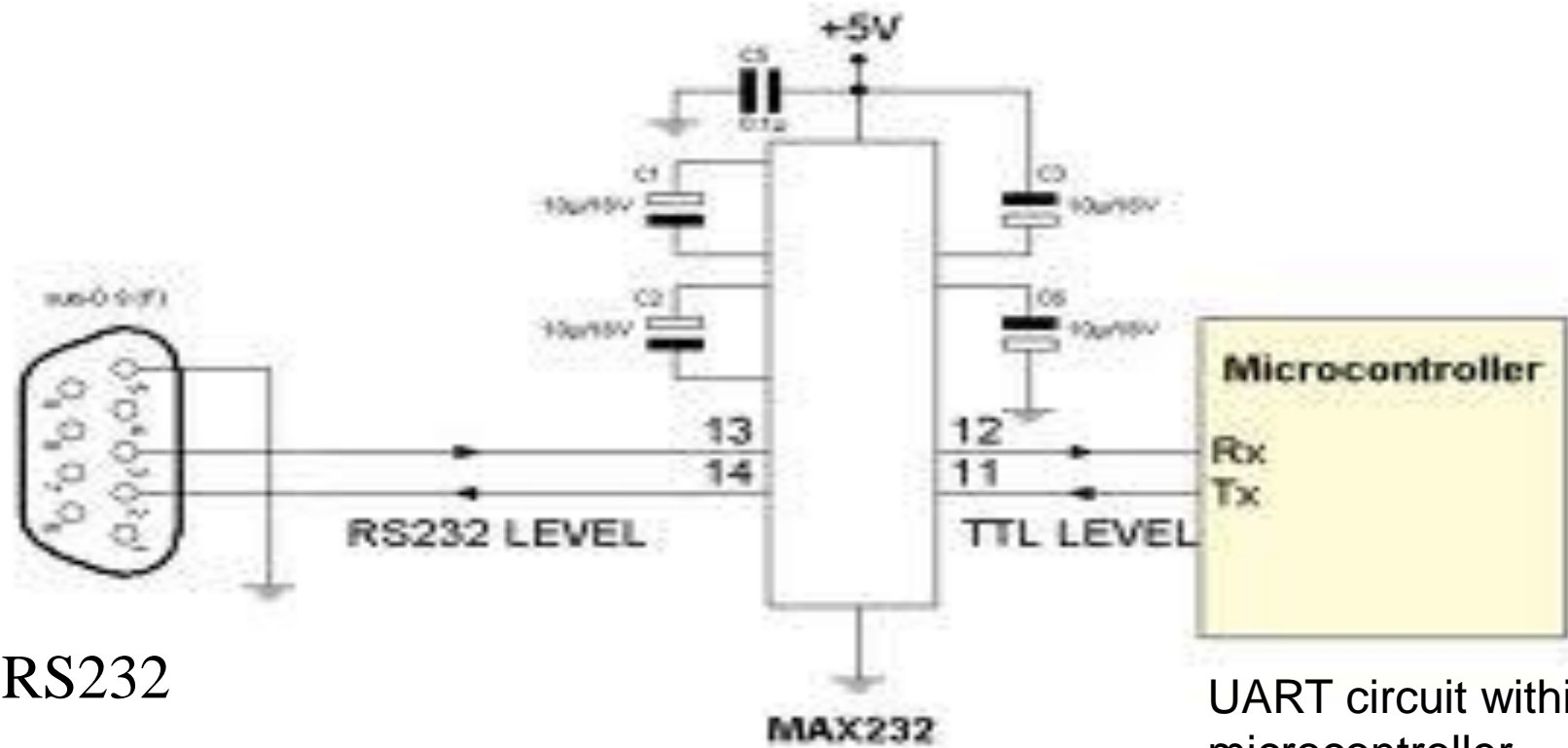
# RS232

- Interfacing signal standard
- Follows asynchronous UART protocol
- Point to point interface

DB-9 Connector Pin Outs



Pin #	Signal	Direction	Description
1	CD	in → computer	Carrier Detect
2	RXD	in → computer	Receive Data
3	TXD	out ← computer	Transmit Data
4	DTR	out ← computer	Data Terminal Ready
5	GND	-	Ground
6	DSR	in → computer	Data Set Ready
7	RTS	out ← computer	Request To Send
8	CTS	in → computer	Clear To Send
9	RI	in → computer	Ring Indicator (Cordless Serial Adapter: optional power input 3.3V-5.0V)



RS232

UART circuit within  
microcontroller

RS232 line type and logic level	RS232 voltage	TTL voltage to/from MAX232
Data transmission (Rx/Tx) logic 0	+3 V to +15 V	0 V
Data transmission (Rx/Tx) logic 1	-3 V to -15 V	5 V
Control signals (RTS/CTS/DTR/DSR) logic 0	-3 V to -15 V	5 V
Control signals (RTS/CTS/DTR/DSR) logic 1	+3 V to +15 V	0 V

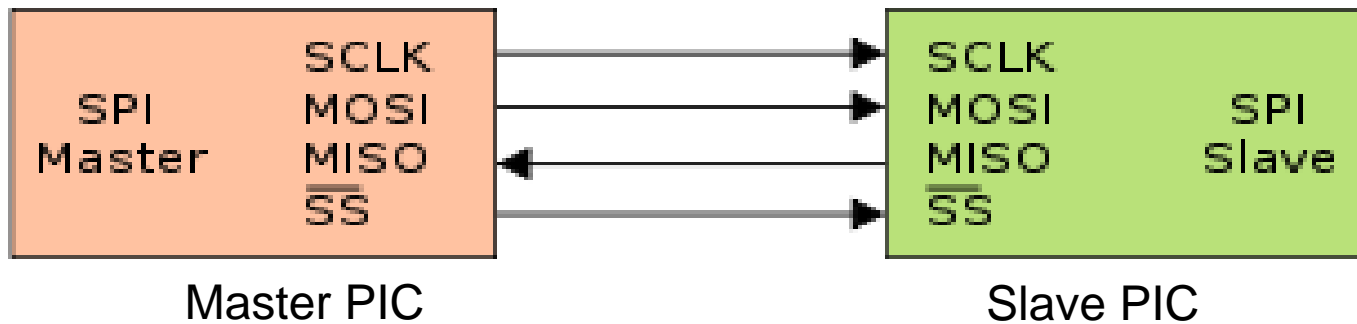
---

# SERIAL COMMUNICATION PROTOCOLS

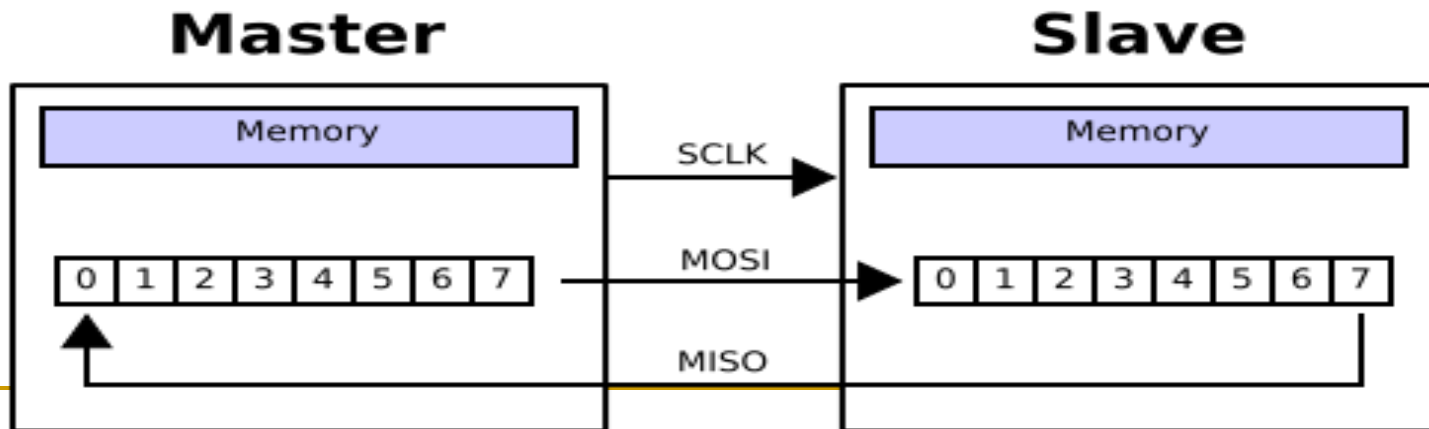


# SPI- Synchronous Peripheral Interface

- Defined by Motorola on the MC68HCxx line of microcontrollers
- Full duplex
- It supports 1) full master mode
  - 2) slave mode(with general address call)
- Addressing each device is not needed
- If SS pin set as 1- Master
- If SS pin set as 0- Slave
- It is called as 3 wire communication
- Allows 8 bit to transmit & receive
- Generally faster than I<sup>2</sup>C, capable of several Mbps
- Bus Arbitration logic is needed

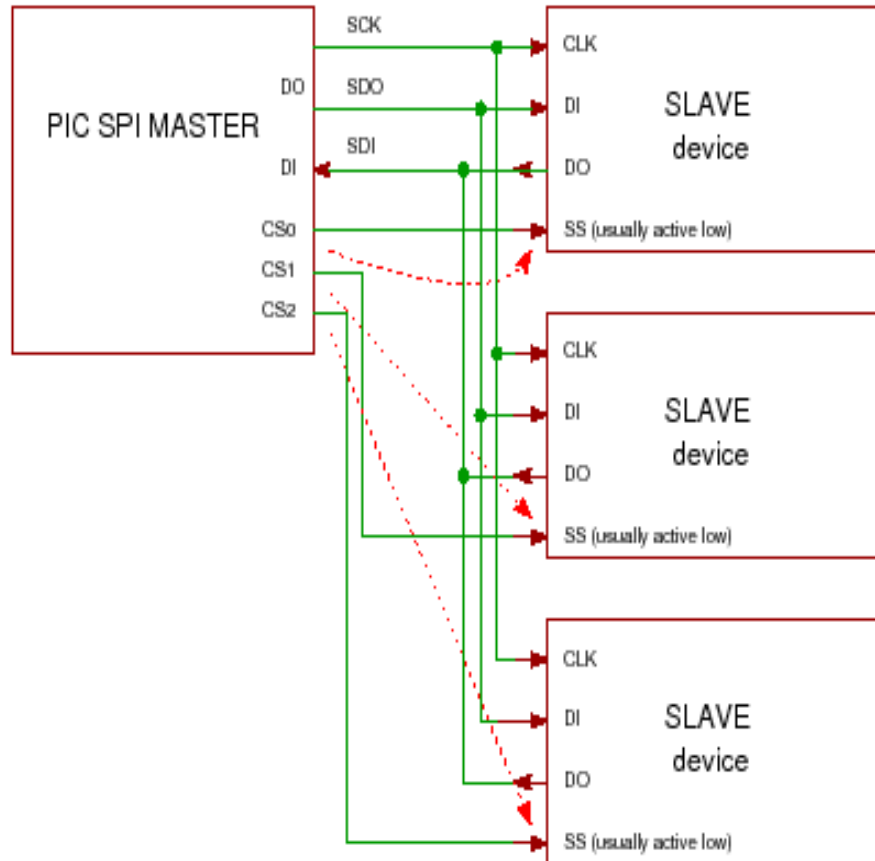


- A master sends a clock signal, and upon each clock pulse it shifts one bit out to the slave, and one bit in, coming from the slave.

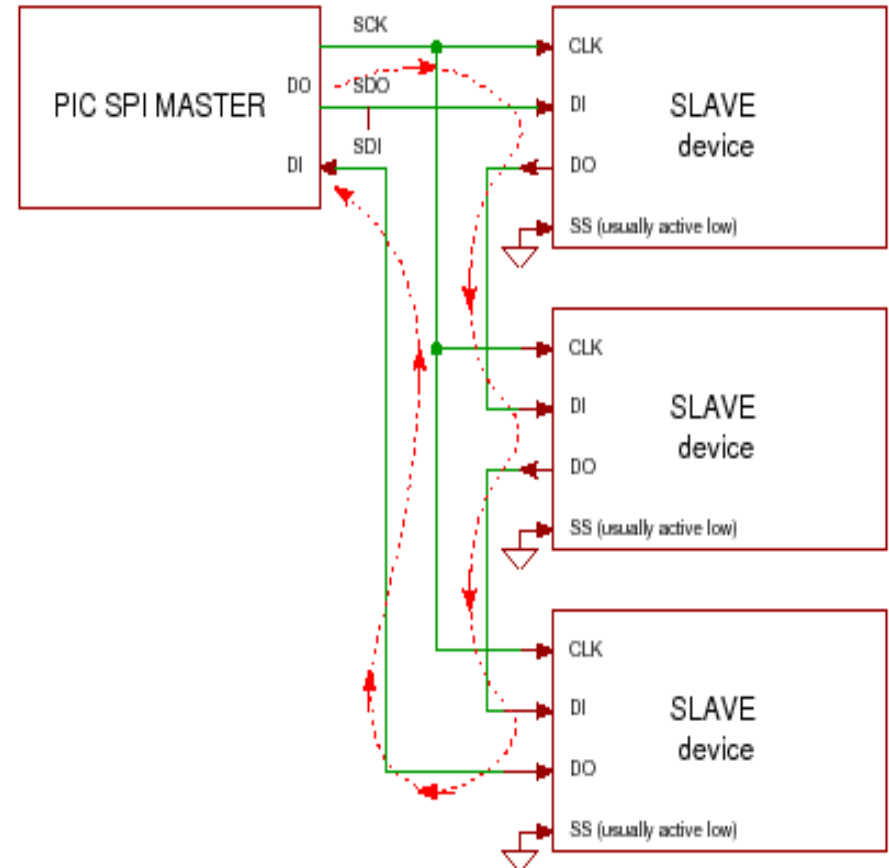


# BUS ARBITRATION

## Using separate chip selects



## Daisy chaining



---

# Applications of SPI

- Like I<sup>2</sup>C, used in EEPROM, Flash, and real time clocks
- Better suited for “data streams”, i.e. ADC converters
- Full duplex capability, i.e. communication between a codec and digital signal processor

# I2C (Inter Integrated Circuit)

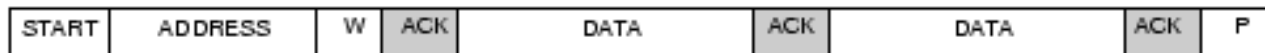
- Developed by Philips Semiconductor for TV sets in the 1980's
- 2-wire serial bus – Serial data (SDA) and Serial clock (SCL)
- Half-duplex, synchronous
- I2c supports
  - 1) master mode
  - 2) slave mode
  - 3) multi master mode
- Addressing is needed (eg: In PIC- MSSP Address Register )
- No chip select or arbitration logic required


# I2C


1. Master sends start condition (S) and controls the clock signal
2. Master sends a unique 7-bit slave device address
3. Master sends read/write bit (R/W) – 0 - slave receive, 1 - slave transmit
4. Wait for/Send an acknowledge bit (A).
5. Send/Receive the data byte (8 bits) (DATA).
6. Expect/Send acknowledge bit (A).
7. Send the STOP bit (P).

# I2C Data Transfer

## Data Transfer from master to slave





 sent by master

 sent by slave

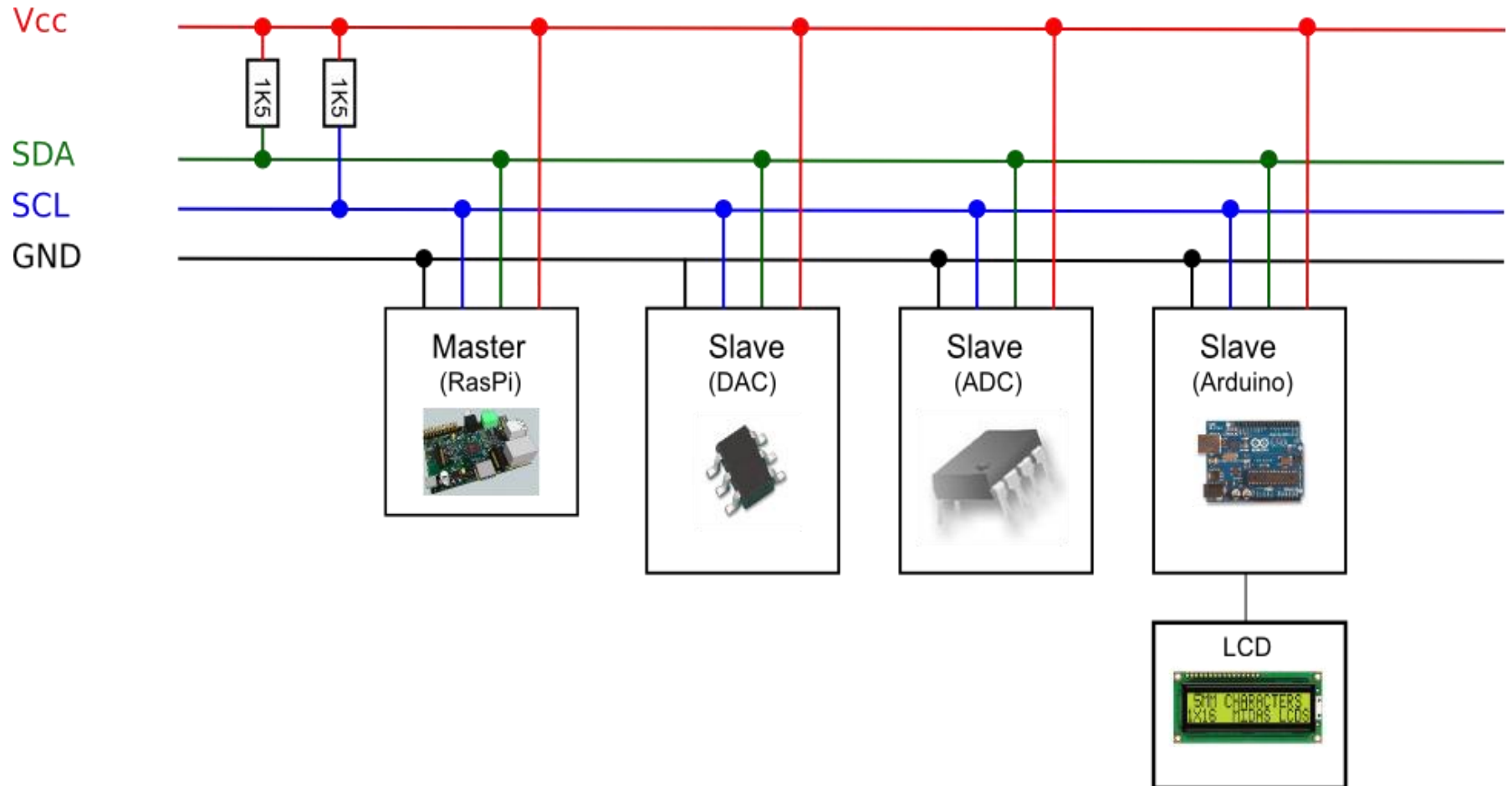
## Data transfer from slave to master



 sent by master

 sent by slave

# Example





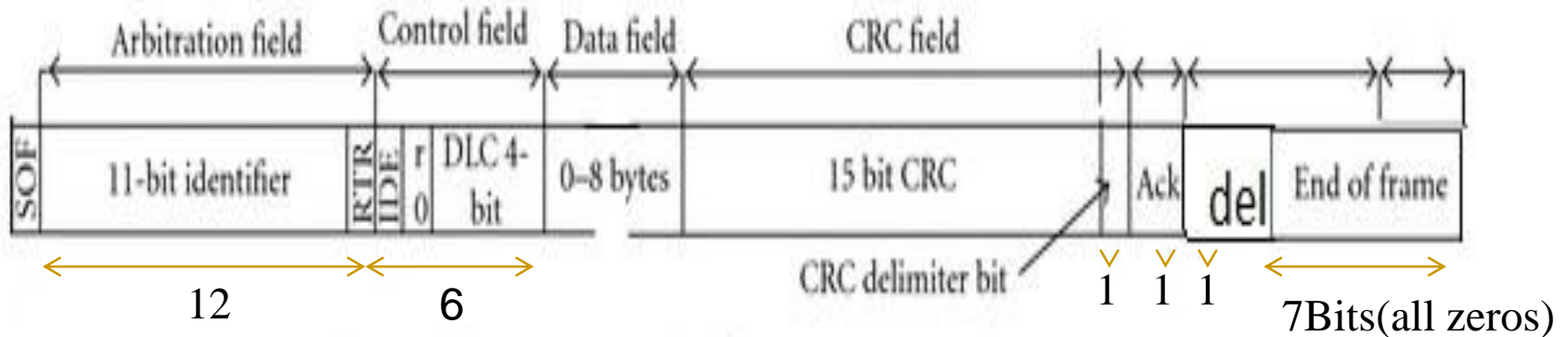
# Applications

- Used as a control interface to signal processing devices that have separate data interfaces, e.g. RF tuners, video decoders and encoders, and audio processors.

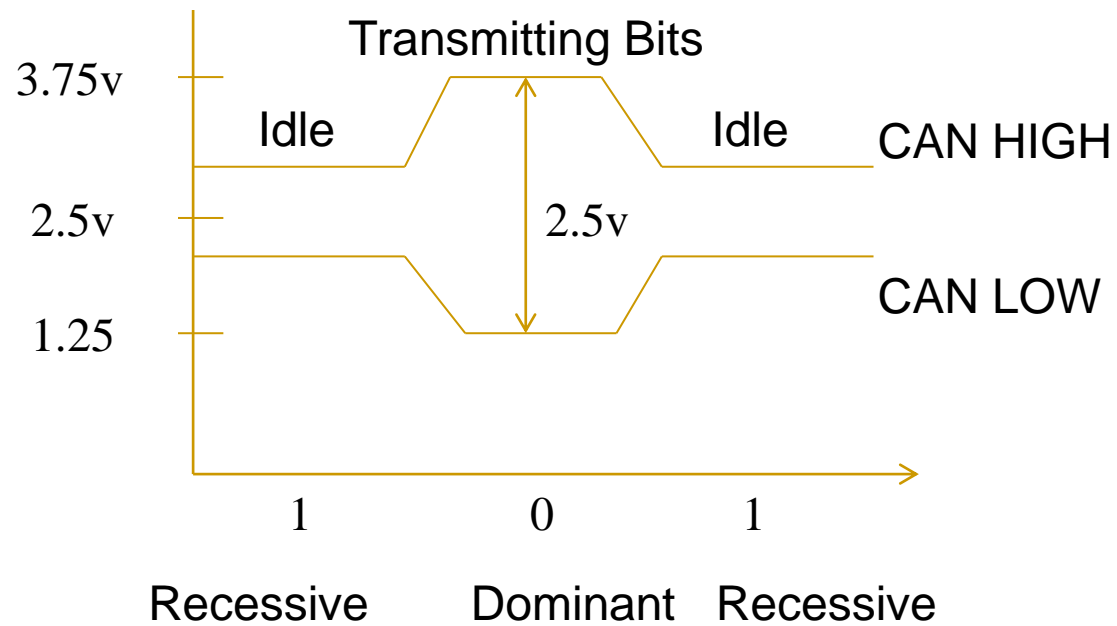
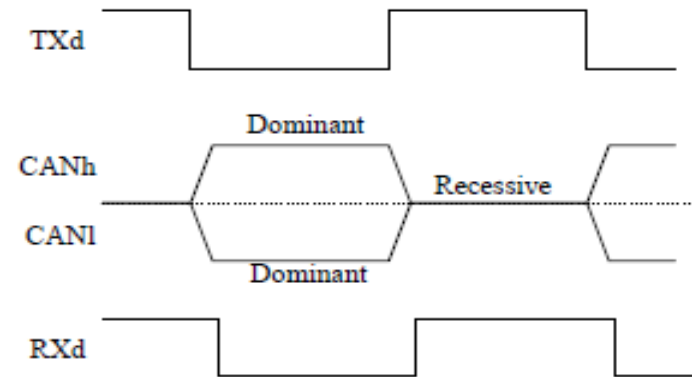
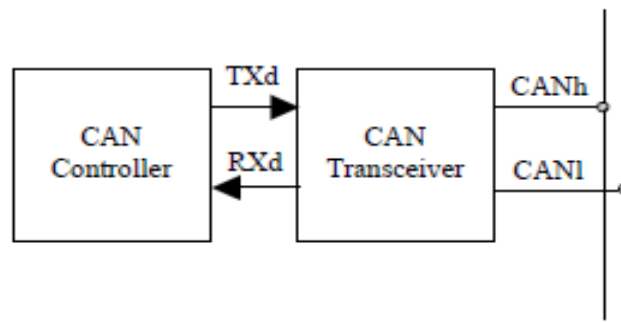
# CAN

- CAN is a serial bus system, which was originally developed for automotive applications in 1980 by BOSCH
- Data link layer protocol internationally standard as ISO 11898-1
- Referred as network of independent controllers
- CAN provides 2 Communication Services:
  - Sending of a message- data frame transmission
  - Requesting of a message- remote transmission request
- Message based protocol not an address based protocol
- Bidirectional and the data rate is 1 Mbps

# CAN DATA FRAME FORMAT



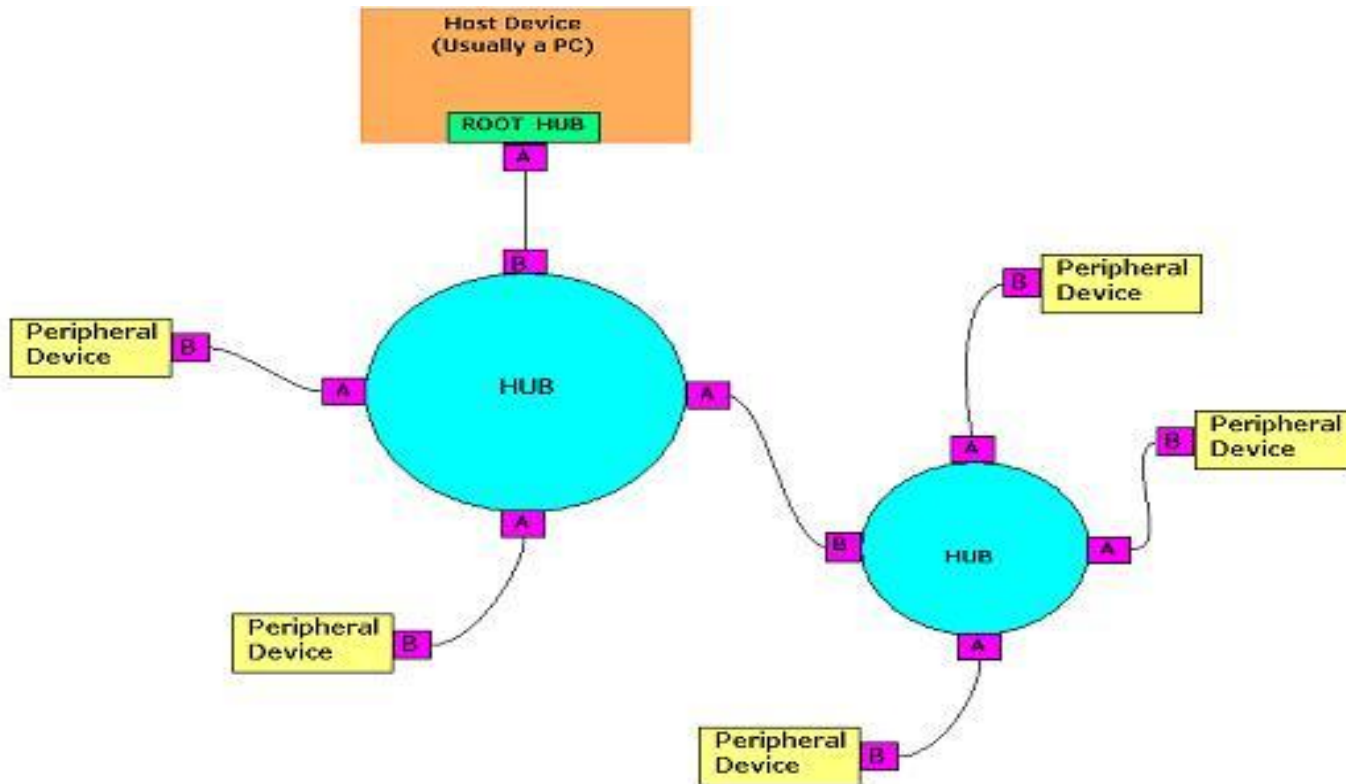
- If RTR sets as 1- packet is a data to the RX
- If RTR sets as 0- packet is a request for the data from the RX



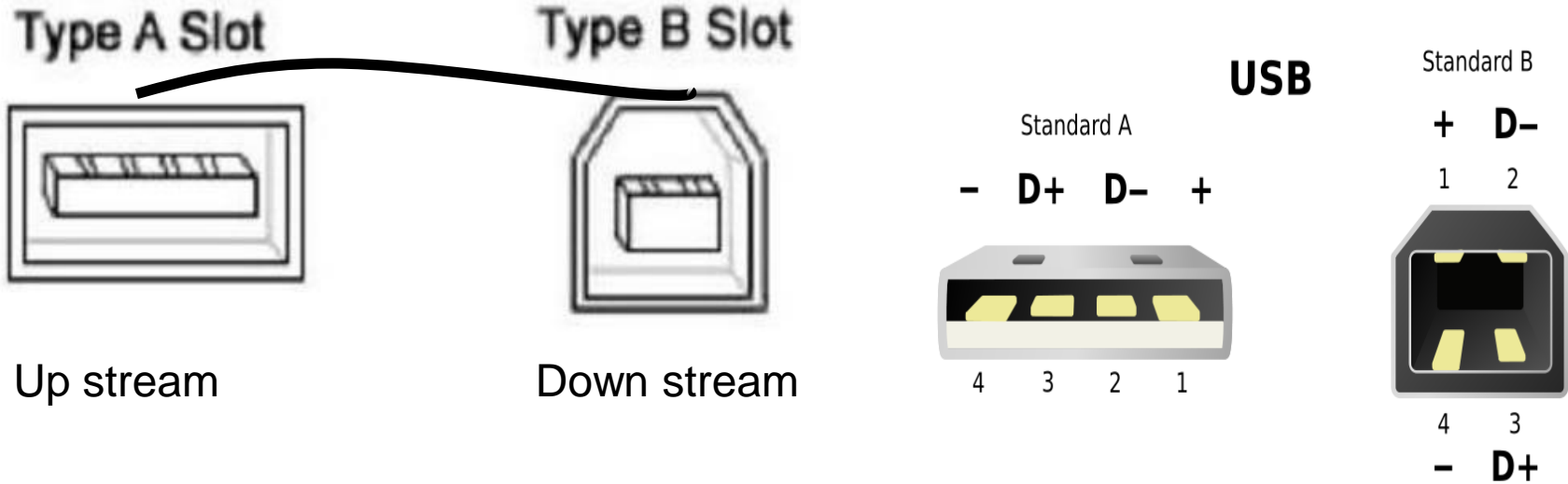
# USB (Universal Serial Bus)

- Bus between host system & number of interconnected peripheral devices
- Addressable bus system
- Hot swapping or hot plugging or plug & play- no Interruption, no reboot, auto configuration.
- Bus powered
- Star topology

# Star topology



## PIN Diagram

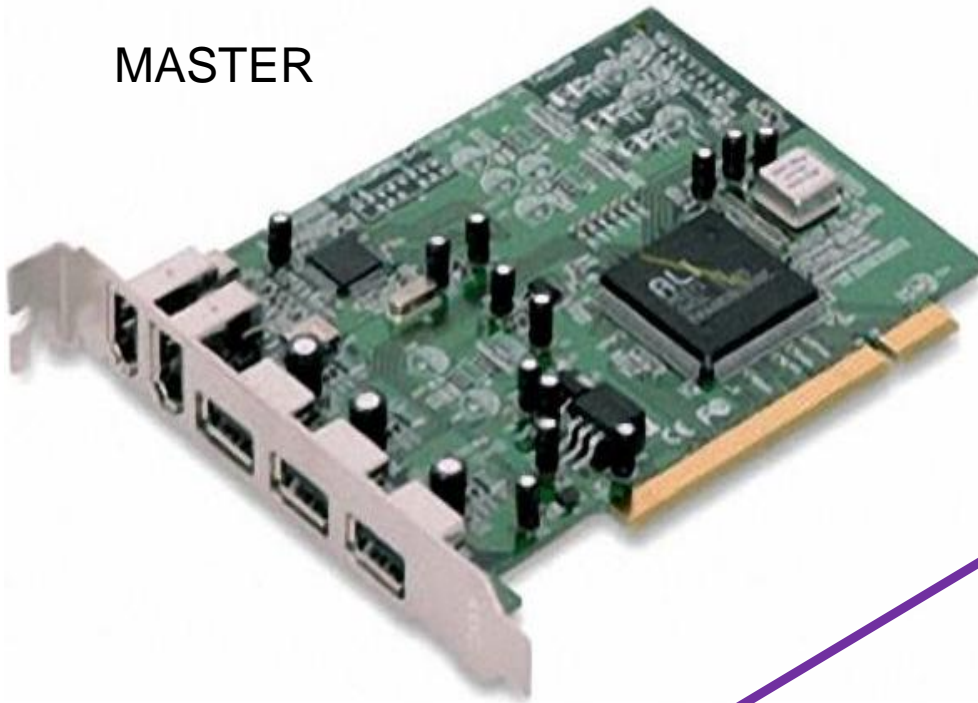


3 standards are :

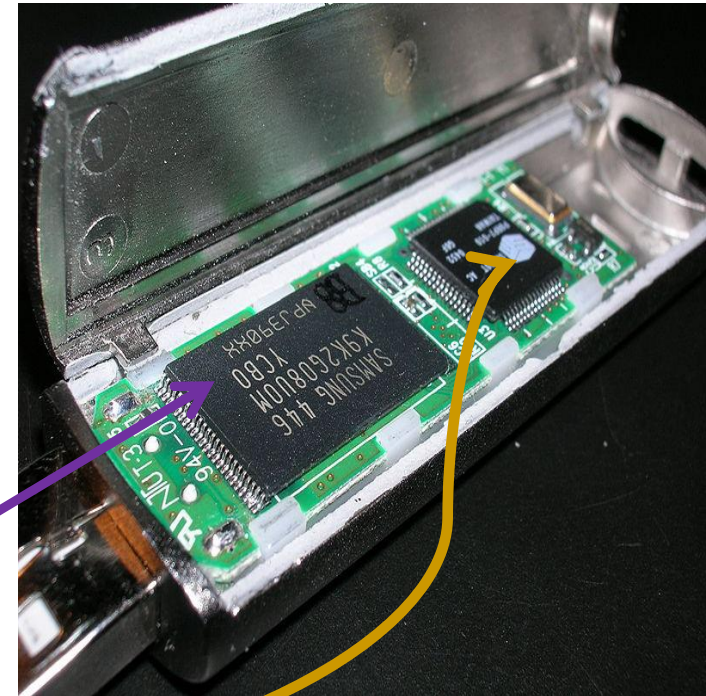
- USB 1.1-1.5 Mbps to 12 Mbps, support up to 127 devices
- USB2.0-1.5, 12 and 480 megabits per second data transfer
- USB 3.0-Super Speed >4.8 Gbits/sec

USB ROOT HUB CONTROLLER or HOST  
INTERFACE CONTROLLER

MASTER



USB STICK



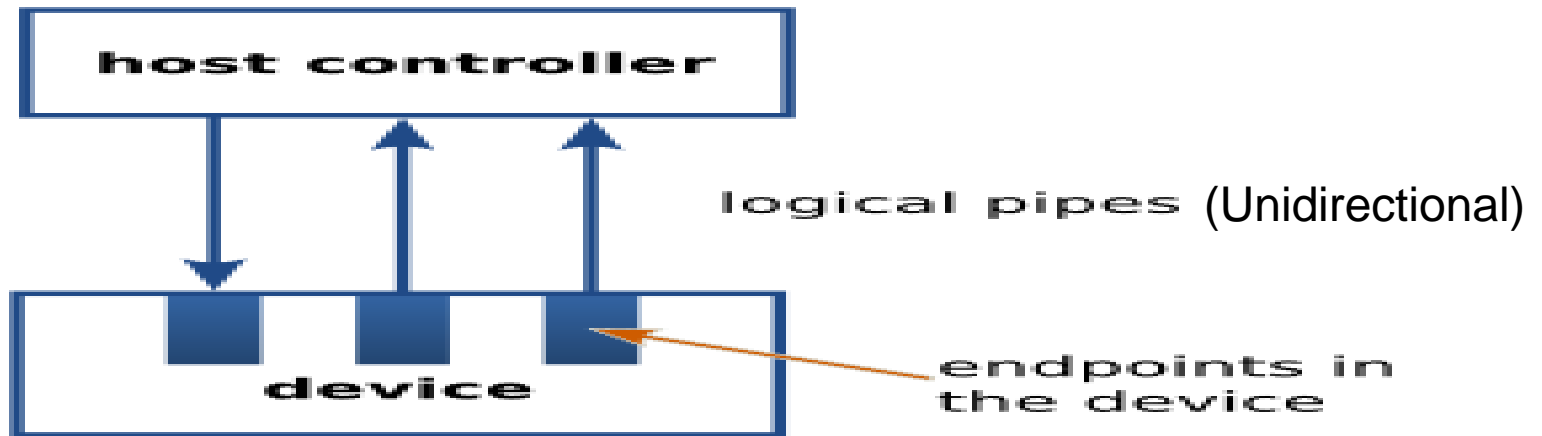
MEMORY

CONTROLLER

SLAVE



- When the Host powers up, Enumeration process starts (host address 0x00)
  - polls each of the Slave devices
  - assigns unique address (7bit address code)
  - finds each device Speed and type of data transfer
  - device drivers are loaded for auto configuration
  - virtual pipes are established for data communication



---

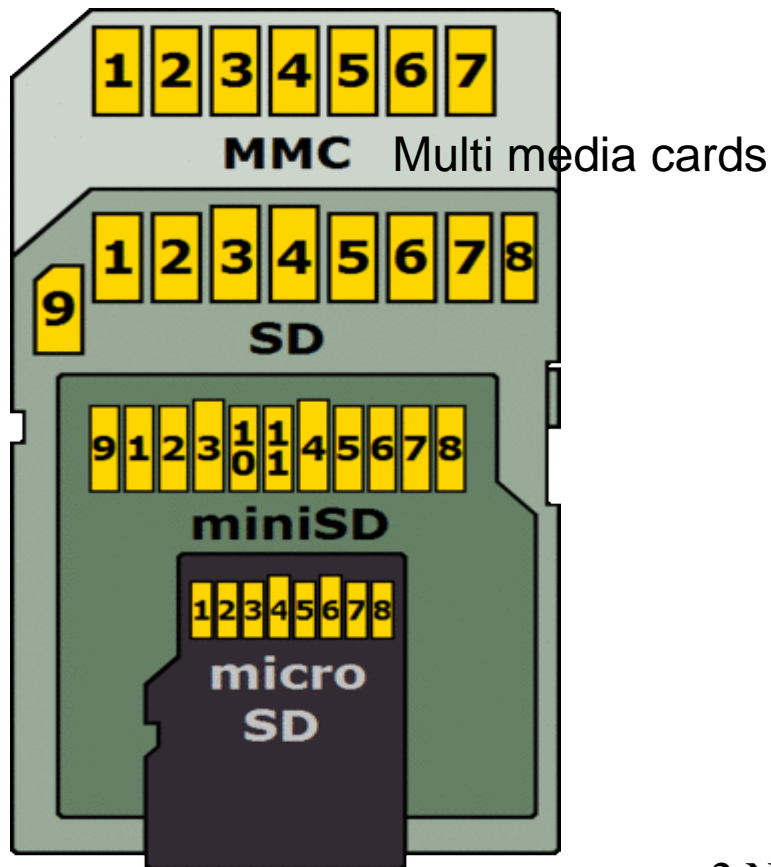
USB can support four data transfer types or transfer mode:

1. Control
2. Isochronous eg: keyboard, mouse
3. Bulk eg: printer
4. Interrupt eg: memory stick

# Secure Digital Card(SDIO)

- Secure Digital (or SD) is a non-volatile FLASH memory card format for use in portable devices, such as mobile phones, digital cameras, GPS navigation devices, and tablet computers.
- **Flash memory** is an electronic non-volatile computer storage device that can be electrically erased and reprogrammed.
- Programmable for communication in SPI mode or 1bit SD mode or 4bit SD

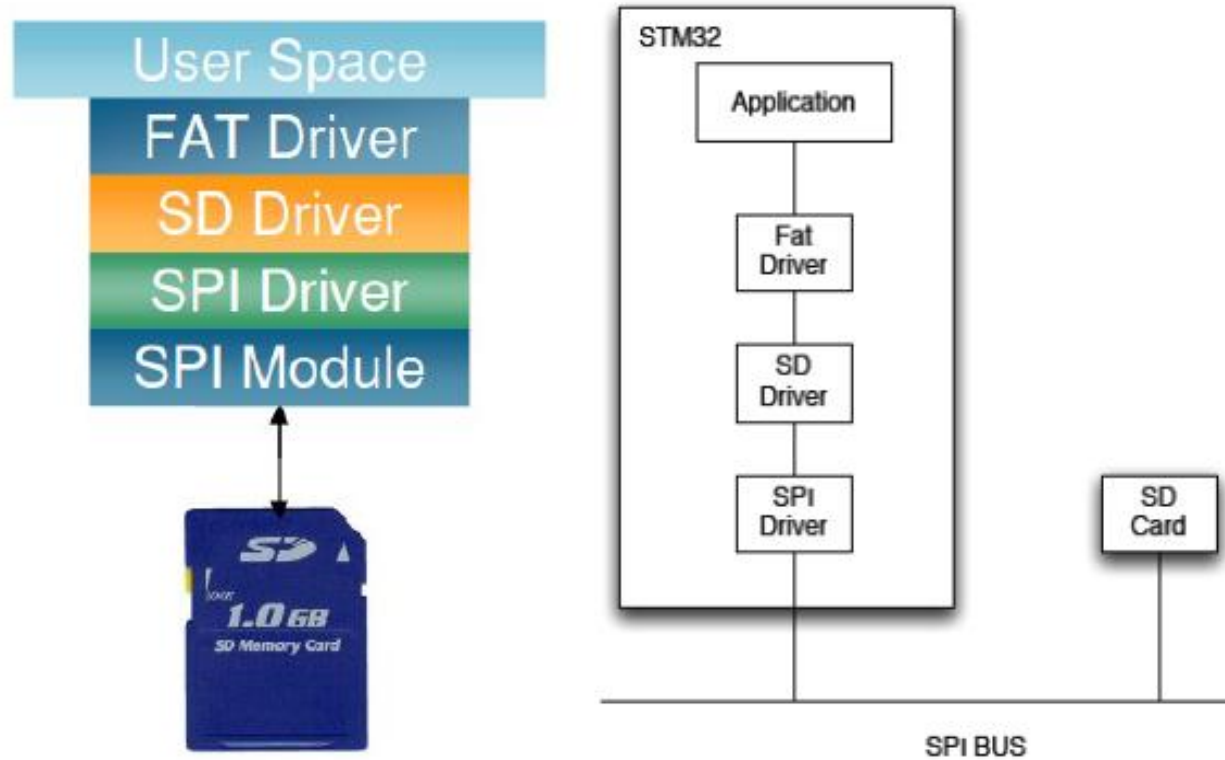
# Secure Digital Card(SDIO)



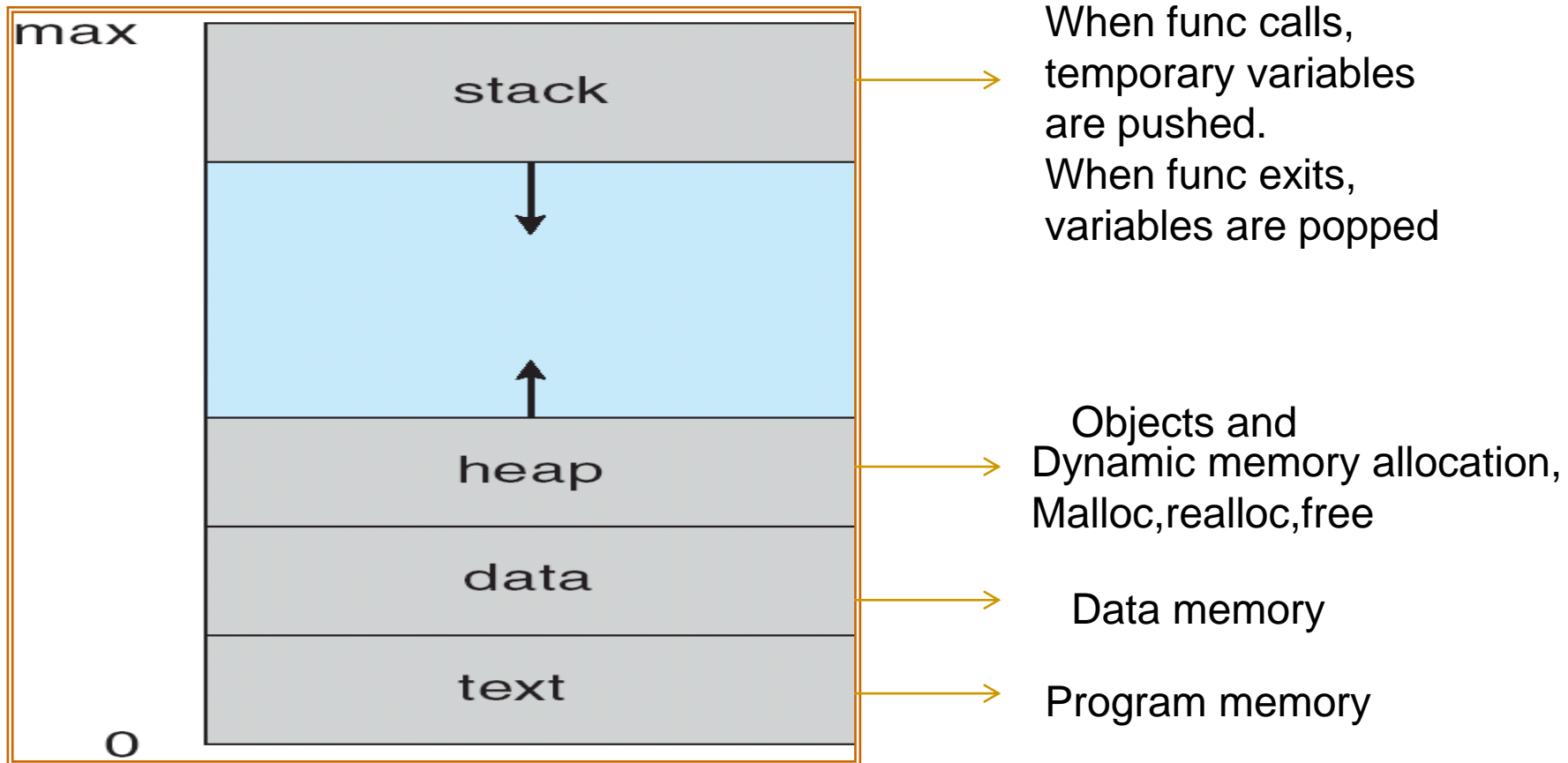
2 NAND CHIPS AND ONE SD CONTROLLER

### SPI Bus Mode

MMC Pin	SD Pin	miniSD Pin	microSD Pin	Name	I/O	Logic	Description
1	1	1	2	nCS	I	PP	SPI Card Select [CS] (Negative Logic)
2	2	2	3	DI	I	PP	SPI Serial Data In [MOSI]
3	3	3		VSS	S	S	Ground
4	4	4	4	VDD	S	S	Power
5	5	5	5	CLK	I	PP	SPI Serial Clock [SCLK]
6	6	6	6	VSS	S	S	Ground
7	7	7	7	DO	O	PP	SPI Serial Data Out [MISO]
	8	8	8	NC nIRQ	. O	. OD	Unused (memory cards) Interrupt (SDIO cards) (Negative Logic)
	9	9	1	NC	.	.	Unused
		10		NC	.	.	Reserved
		11		NC	.	.	Reserved



# Memory Model



- 
- Stack : store temporary variables created by funcs
    - Fast access
    - Managed automatically by CPU
    - Local variables only
    - Limited stack size. (stack overflow will occur)
    - Grows downwards- going to reach its limit
  - Heap : memory allocation can be done by using malloc, realloc & free (dynamic memory allocation)
    - Slow access
    - Managed by the programmer
    - Variables can be accessed globally
    - No limit on memory
    - Grows upwards- its limit increases continuously
-



# Process and Threads

- A process is an instance of a computer program that is being executed. It contains the program code and its current activity

PROCESS	THREADS
Instance of a computer program being executed	Subsets of a process
Independent resources & memory	Shared resources & memory
Context switching is slow	Context switching is fast
Independent	Depends on their own process

# Process Concept

Process – a program in execution

process execution must progress in sequential fashion

A process includes:

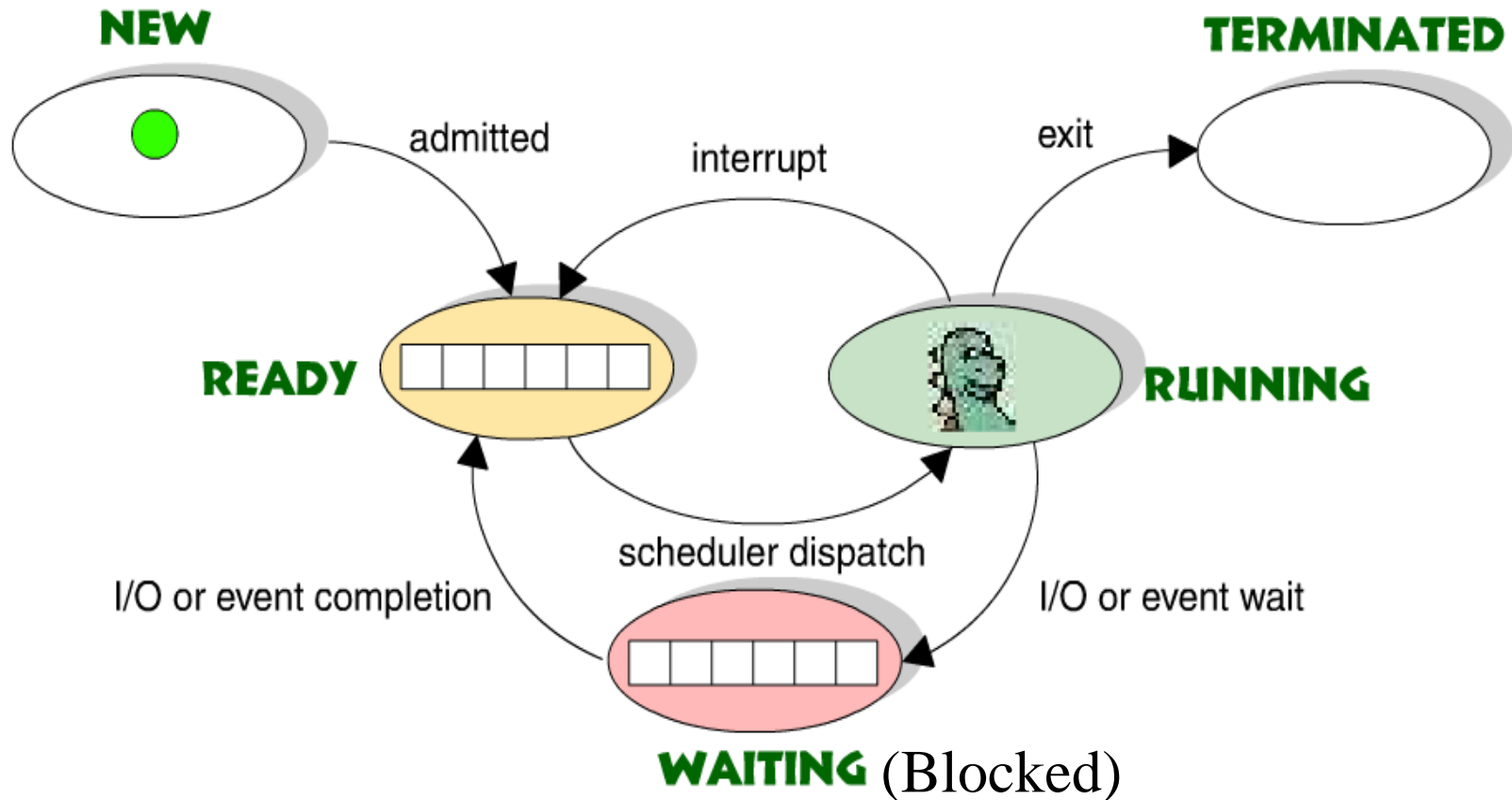
- program counter
  - stack
  - data section
- 
- Task is a similar one to process. But the term TASK is often used in RTOS.
  - Task – a embedded program in execution
  - But in some RTOS, the term PROCESS is also used

# Process State

As a process executes, it changes *state*

- **new:** Create & memory allotted
- **ready:** The process is waiting to run while the high priority tasks are running
- **running:** executing with allotted resources. Some times it preempts by another priority task
- **Waiting(blocked):** The process is waiting for some event to occur
- **terminated:** The process has finished execution & memory gets de-allotted.

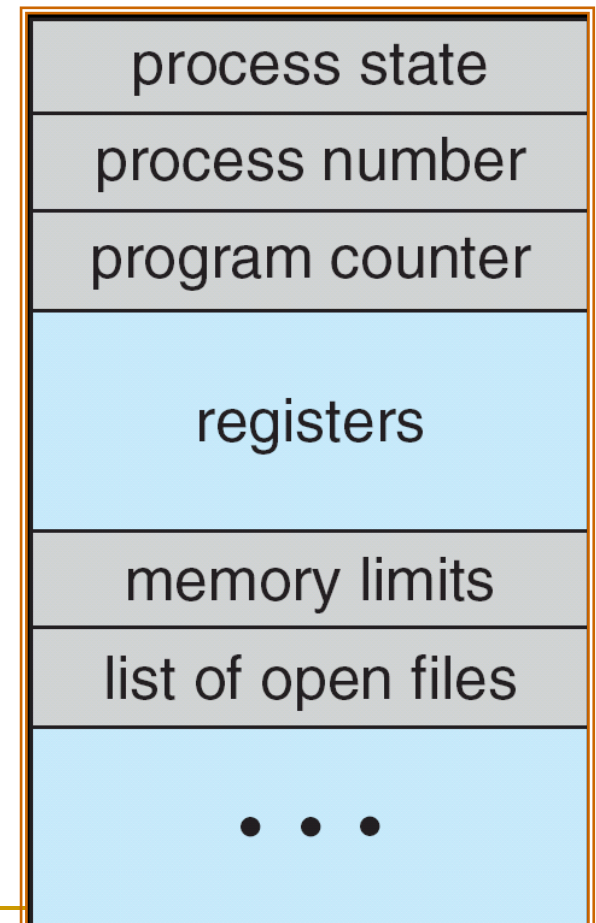
# Diagram of Process State



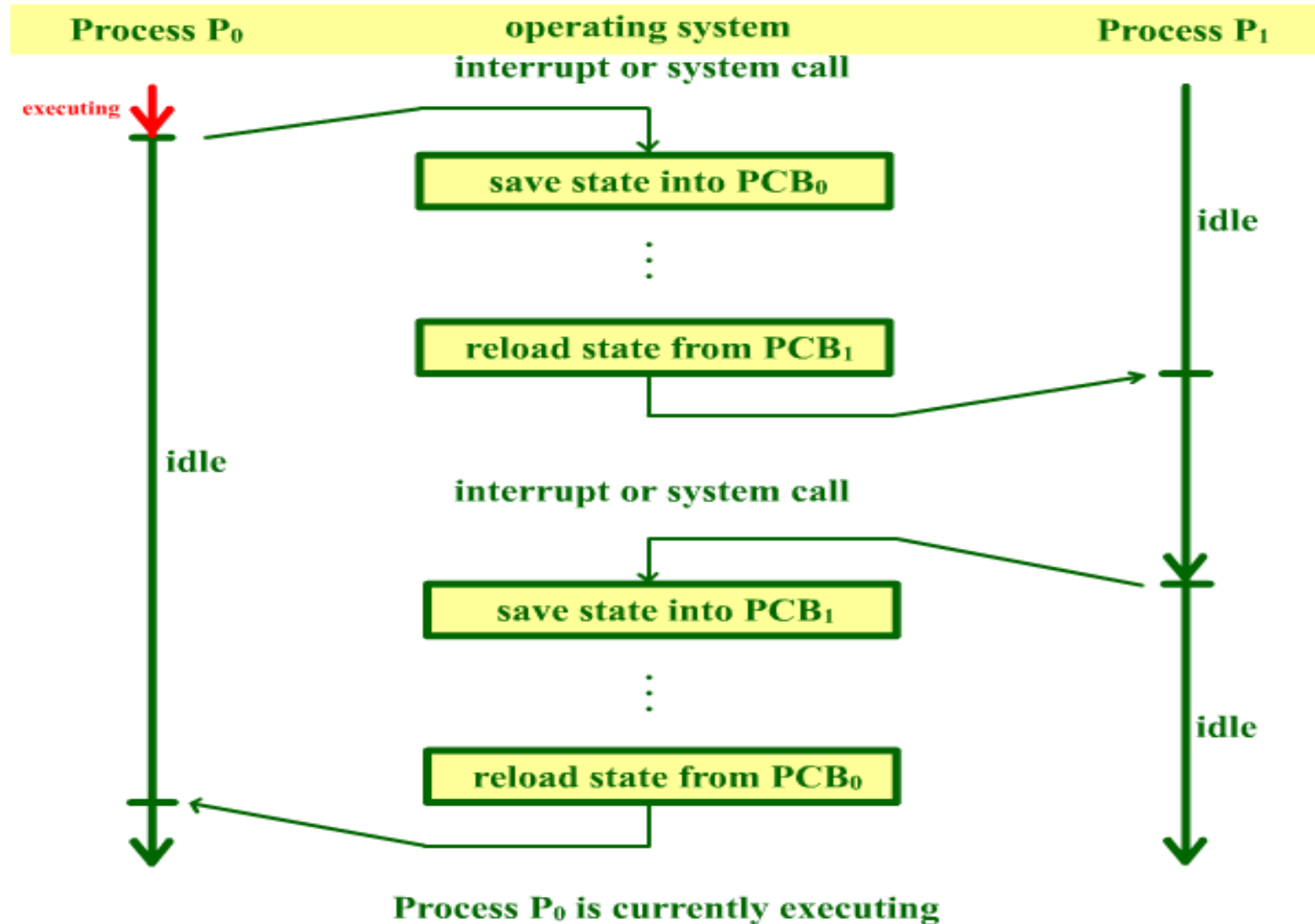
# Process Control Block (PCB)

Information associated with each process

- Process state
- Program counter
- CPU registers
- CPU scheduling information
- Memory-management information
- Accounting information
- I/O status information



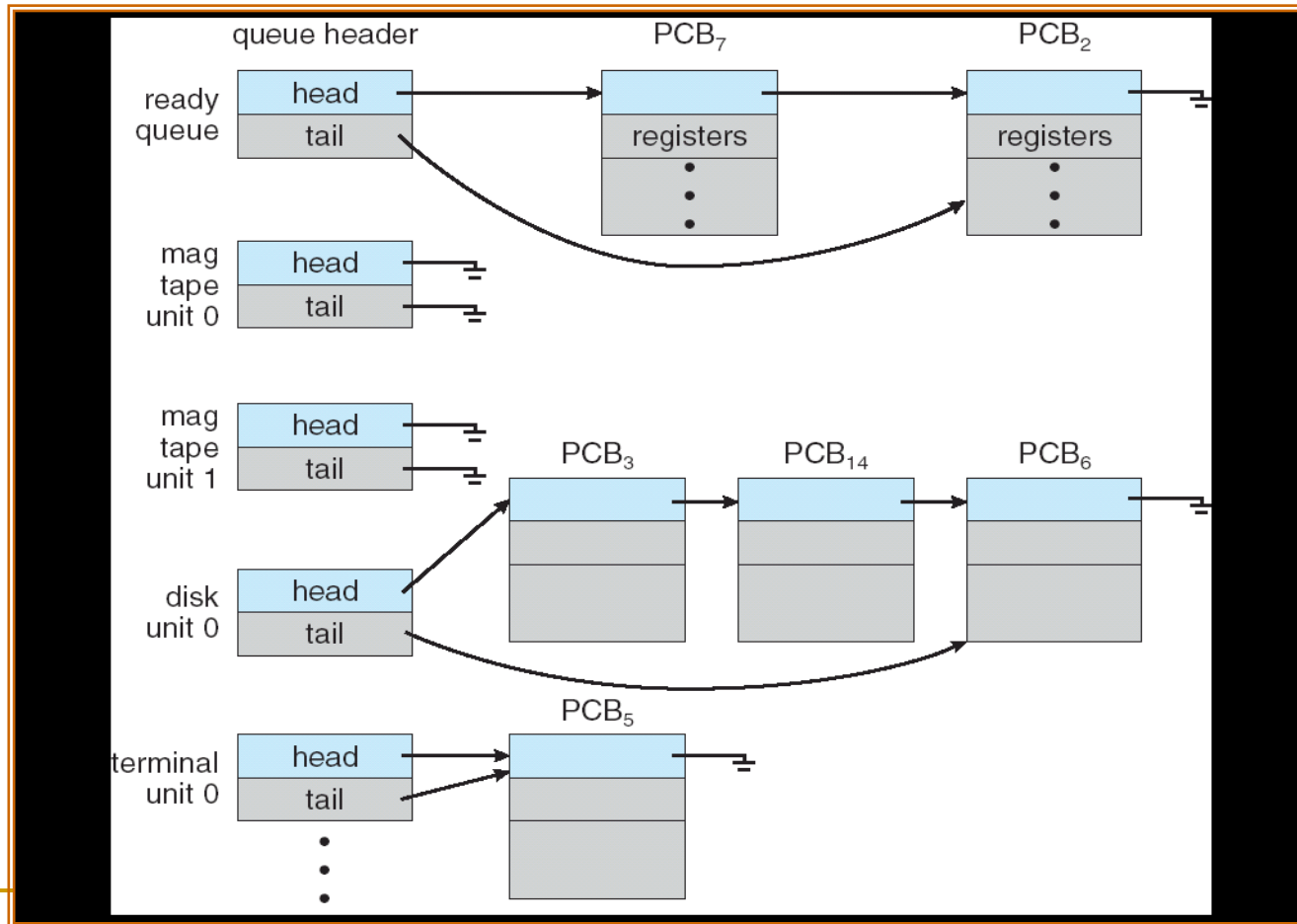
# CPU Switch From Process to Process



# Process Scheduling Queues

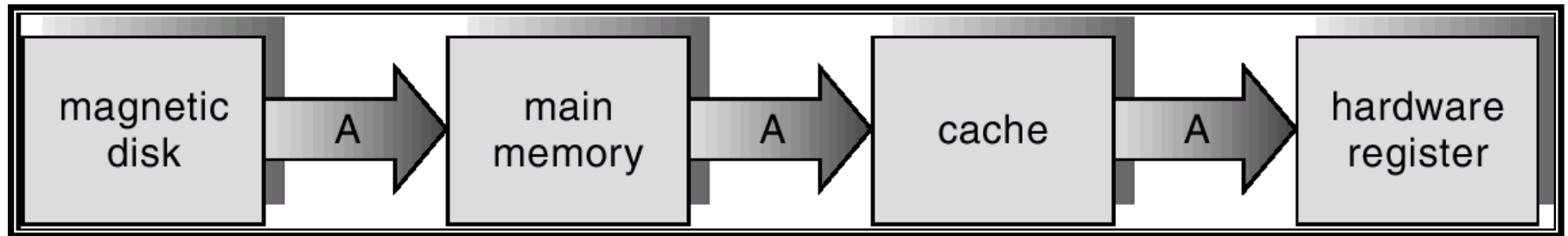
- **Job queue** – set of all processes in the system
- **Ready queue** – set of all processes residing in main memory, ready and waiting to execute
- **Device queues** – set of processes waiting for an I/O device
- Processes migrate among the various queues

# Ready Queue And Various I/O Device Queues





# Migration of 'A' From Disk to Register



---

# Transferring Data Between the CPU and I/O Device

What is Interrupt Mechanism ?

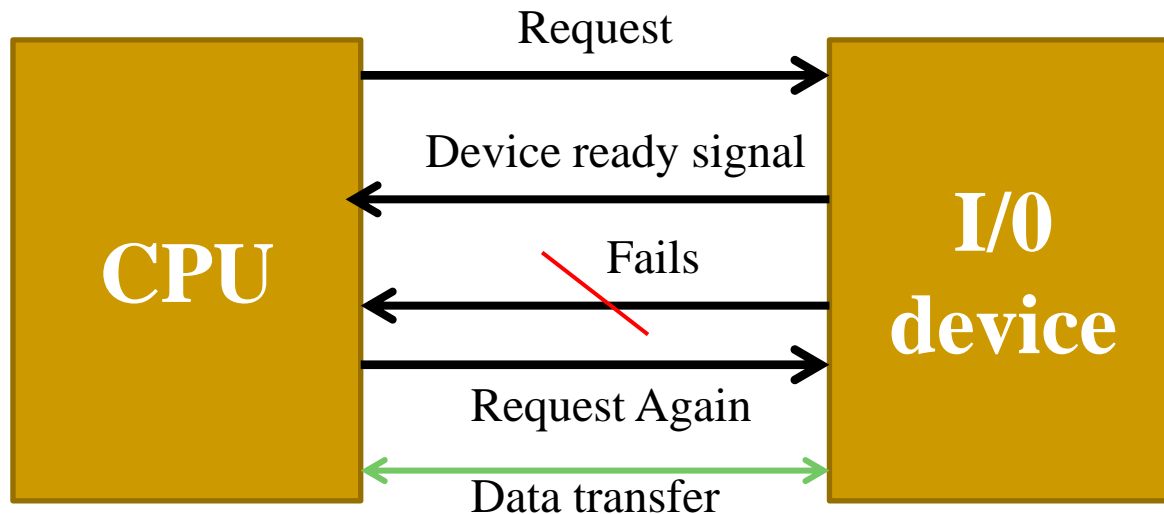
- Interrupts is a mechanism for alleviating the delay caused by this uncertainty and for maximizing system performance.

# Transferring Data Between the CPU and I/O Device

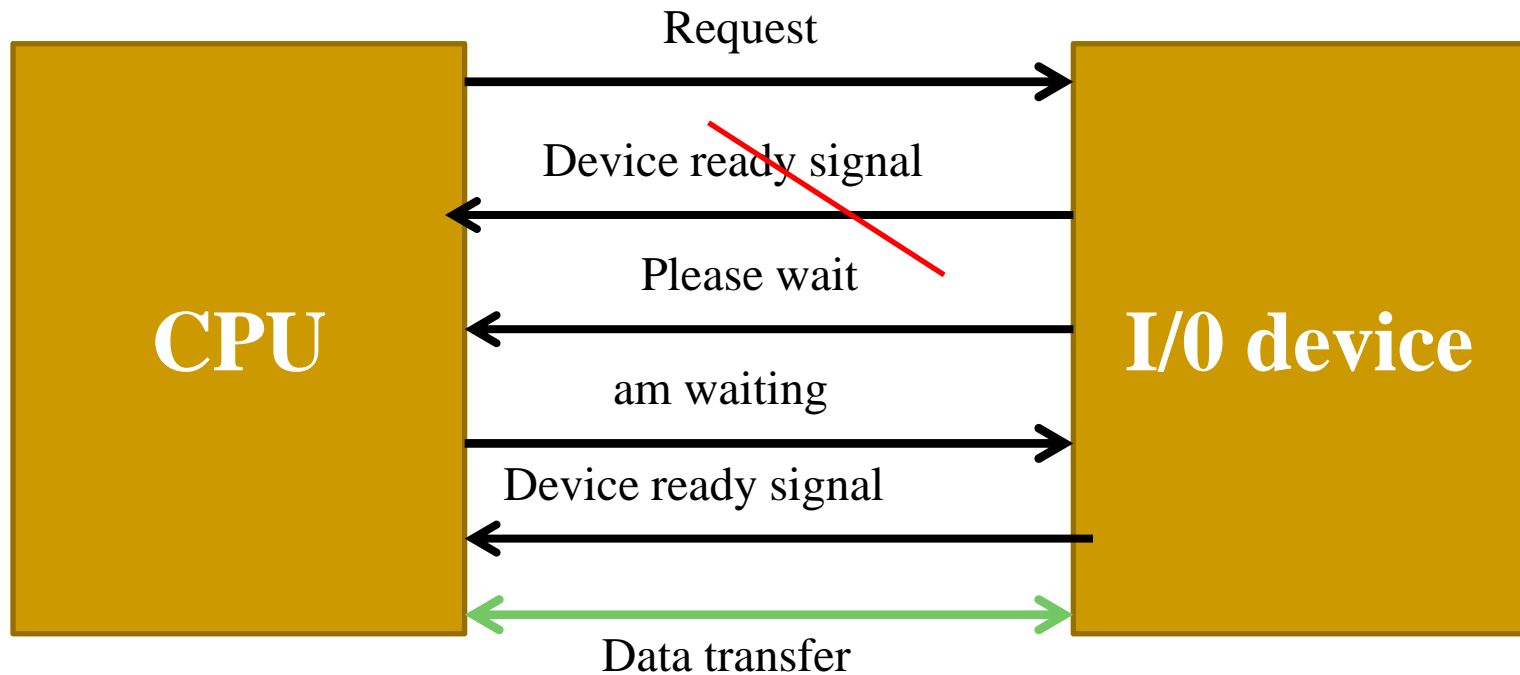
- Polling
- Wait states
- Interrupts

# Polling

- One method used in small system to alleviate the problem of I/O devices with variable delays.
- Repeatedly checking its ports



# wait states

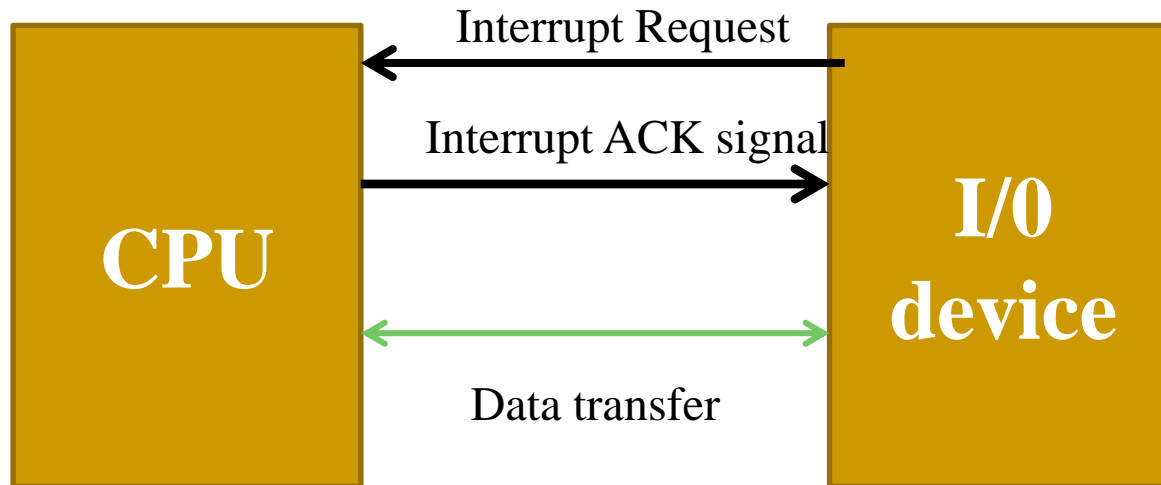


# Disadvantages of Polling & wait states

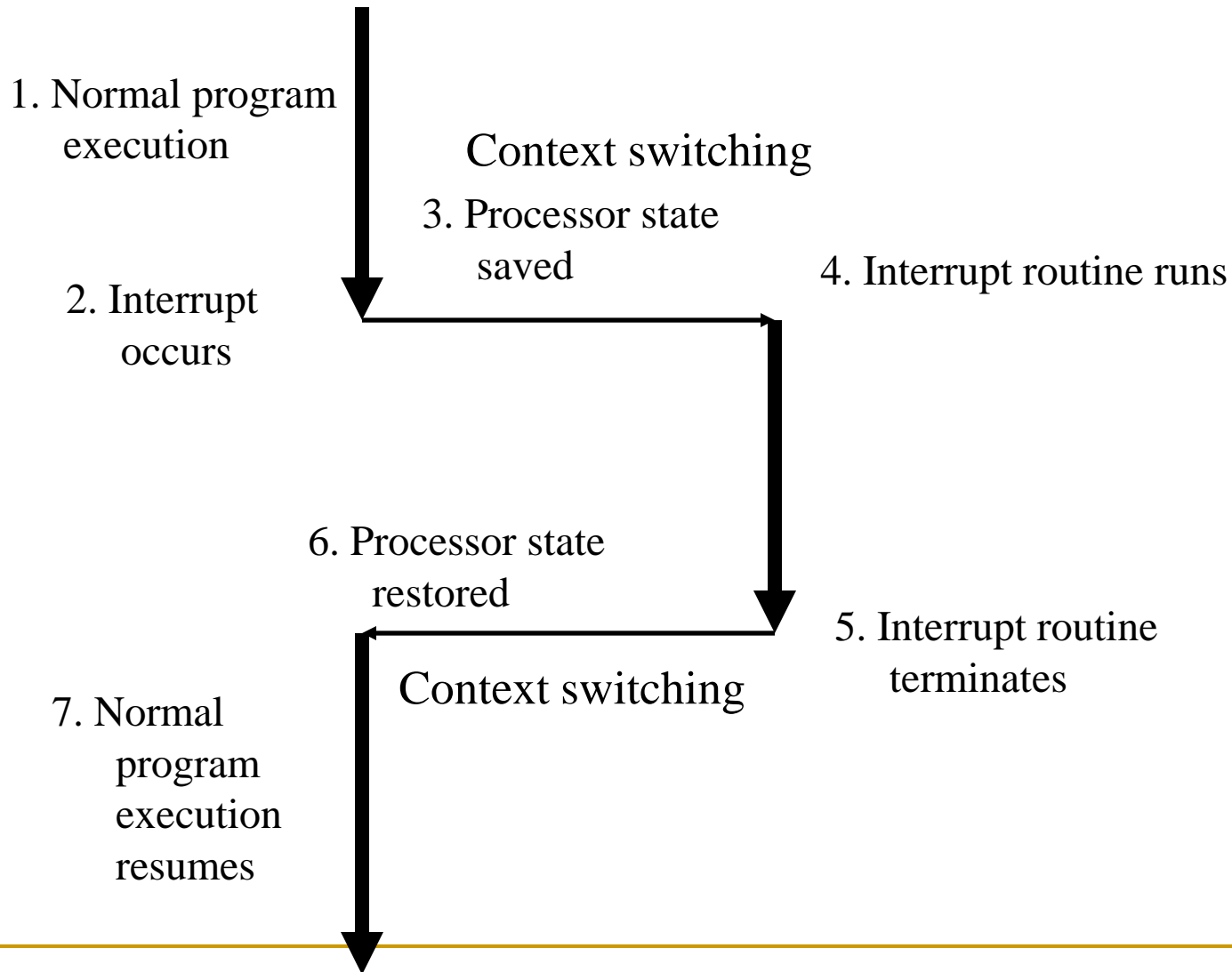
- The CPU does not perform any useful work while waiting for the I/O device to become ready to transfer data. To make use of this wasted CPU time, *interrupts* are developed.

# Interrupts

- Instead of polling the device or entering a wait state, the CPU continuously executing its instructions & performing useful work.
- When the device is ready to transfer data, it sends an interrupts request to the CPU; this is done via a dedicated signal on the control bus.



# Handling an Interrupt





# Shared data problem

- Arises when an interrupt service routine and the task code share the data or between two tasks that share the data.
- In order to solve the problem, we need to make the shared data “atomic”
- A part of the program is said to be “atomic” if it cannot be interrupted by anything



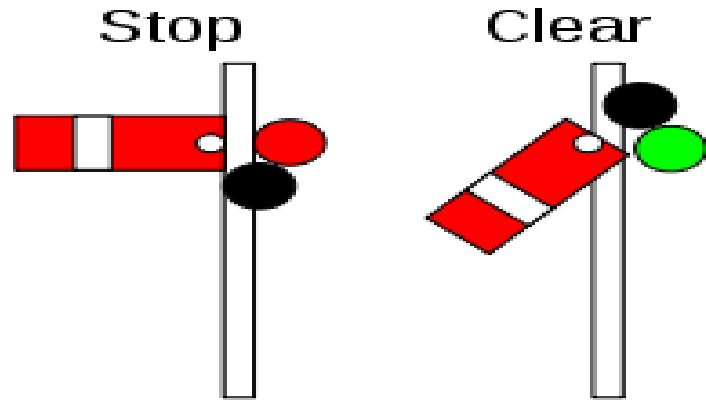
# Semaphores

- Semaphores: Introduced by Dijkstra in 1960s
- Semaphore is a variable or a signal event
- Provides the synchronization mechanism
- Eg: open-close, up-down, pend-post, lock-unlock
- Synchronization mechanism is required at the entry and exit of critical section to ensure exclusive use. This mechanism is **semaphore**.

# Semaphores

- Semaphores have two purposes
  - Mutex: Ensure threads don't access critical section at same time
  - Scheduling constraints: Ensure threads execute in specific order

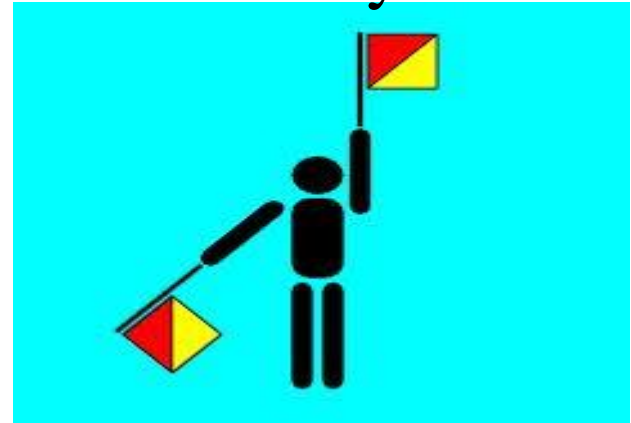
# Real time Semaphores



Railways



Airways



Navy



# Mutual Exclusion (Mutex)

- Mutex is a program or object that is created, when we start the process.
- Any task needs the resource must use the Mutex to lock the resource from other task while it is using the resource.
- If the Mutex is already locked, other tasks are queued.

# Critical section

- In concurrent programming, a critical section is a piece of code that accesses a shared resource that must not be concurrently accessed by more than one thread of execution.

Semaphore taken → critical section → semaphore release

# Conditions for Implementing the Critical Section

- Disable the interrupts, when try to enter into the CS
- Avoid system calls that can cause a context switch while inside the CS.
- Whatever may be, any kind of code must not disturb the CS, until the original thread leaves its CS.
- This approach can be improved by using semaphores.
- Eg: Printer can only be accessed by one process at a time
- But other processes are free to gain the control of CPU & execute other code or CS that are protected by different semaphores.

# Deadlock Characterization

Deadlock can arise if four conditions hold simultaneously.

- **Mutual exclusion:** only one process at a time can use a resource.
- **Hold and wait:** a process holding at least one resource is waiting to acquire additional resources held by other processes.
- **No preemption:** a resource can be released only voluntarily by the process holding it, after that process has completed its task.
- **Circular wait:** there exists a set  $\{P_0, P_1, \dots, P_n\}$  of waiting processes such that  $P_0$  is waiting for a resource that is held by  $P_1$ ,  $P_1$  is waiting for a resource that is held by  $P_2$ , ...,  $P_{n-1}$  is waiting for a resource that is held by  $P_n$ , and  $P_n$  is waiting for a resource that is held by  $P_0$ .



# Pre-emption

- The act of temporarily interrupting a task being carried out by a computer system without requiring its cooperation.
- Preemptive scheduler consists of high priority tasks which has the power to preempt or interrupt any process.
- But kernel functions cannot be preempted

# Embedded Software's:

## IDE- Integrated Development Environment

- **Compiler**→ The compiler turns source code into machine code packaged in object files. (HI-tech compiler, CCS etc)
- **Cross compiler**→A cross-compiler produces object files that will then be linked for the target instead of the computer running the compiler
- A **cross compiler** is a compiler capable of creating executable code for a platform other than the one on which the compiler is running.( eg: GCC for linux)
- **Debugger**→A special program used to find errors (*bugs*) in other programs. A debugger allows a programmer to stop a program at any point and examine and change the values of variables.

- 
- The *Simulator* tries to duplicate the *behaviour* of the device.

The *Emulator* tries to duplicate the *inner workings* of the device.(mimicry)

- **Emulation** is the replacement of a real world device with an model at a well defined interface for the purposes of allowing controlled responses from the emulated real world device.

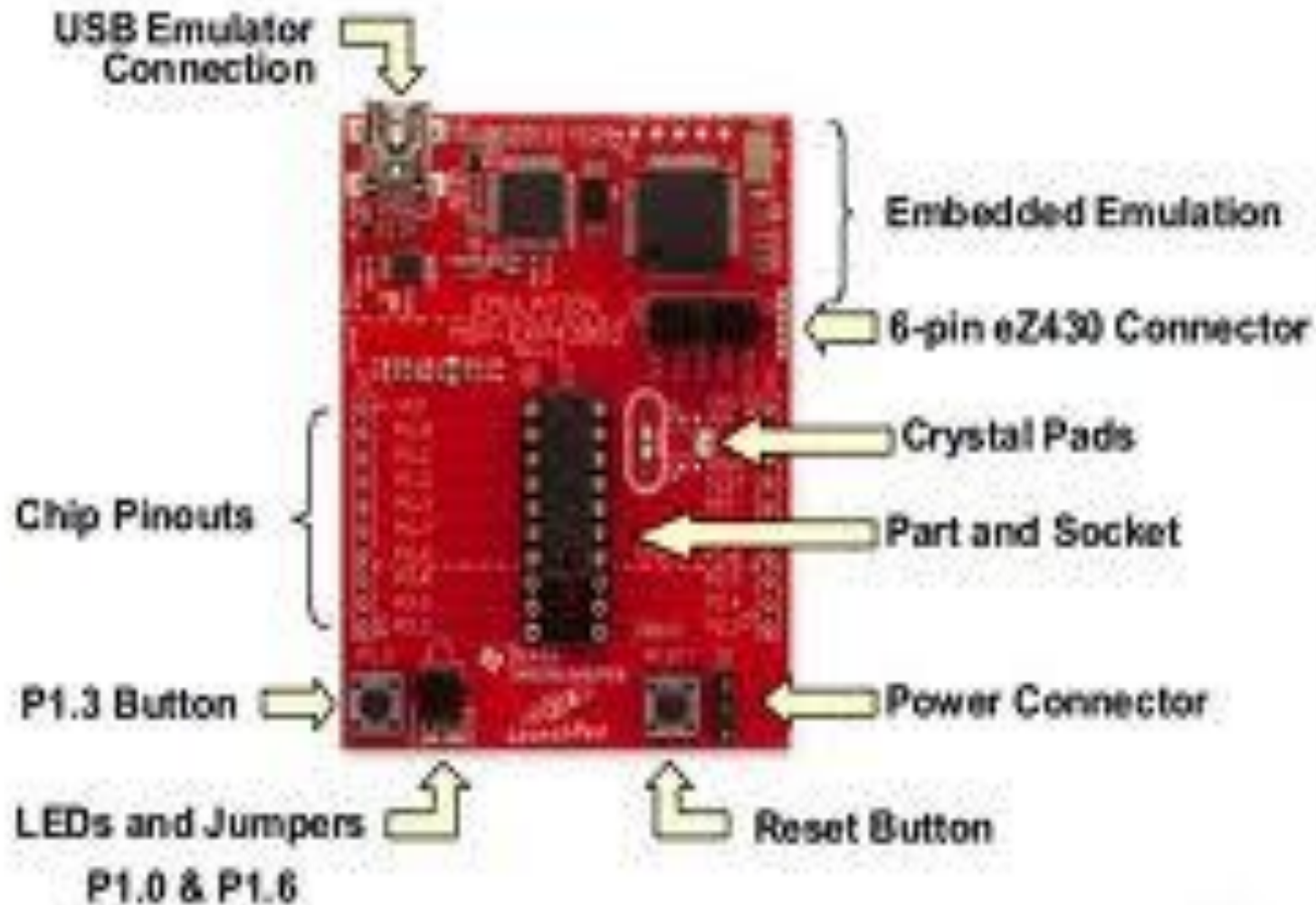
# *MSP430 Launch Pad*



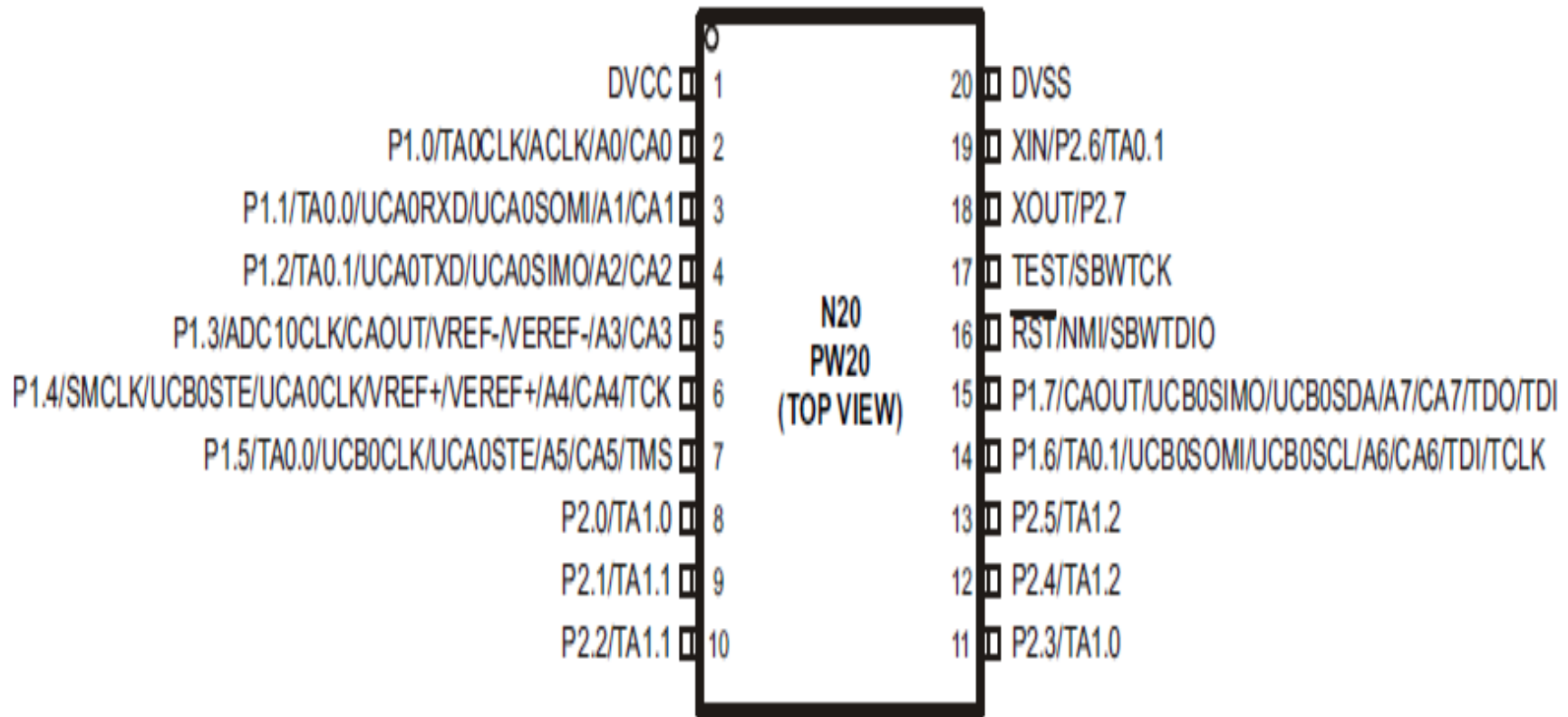
# Introduction to MSP430

- The MSP430 can be used for low powered embedded devices.
- The electric current drawn in idle mode can be less than 1  $\mu\text{A}$ .
- The top CPU speed is 25 MHz. It can be throttled back for lower power consumption.
- Six different low-power modes, which can disable unneeded clocks and CPU.
- Capable of wake-up times below 1 microsecond, allowing the microcontroller to stay in sleep mode longer, minimizing its average current consumption.

# LaunchPad Development Board



# PIN DIAGRAM OF MSP430G2553

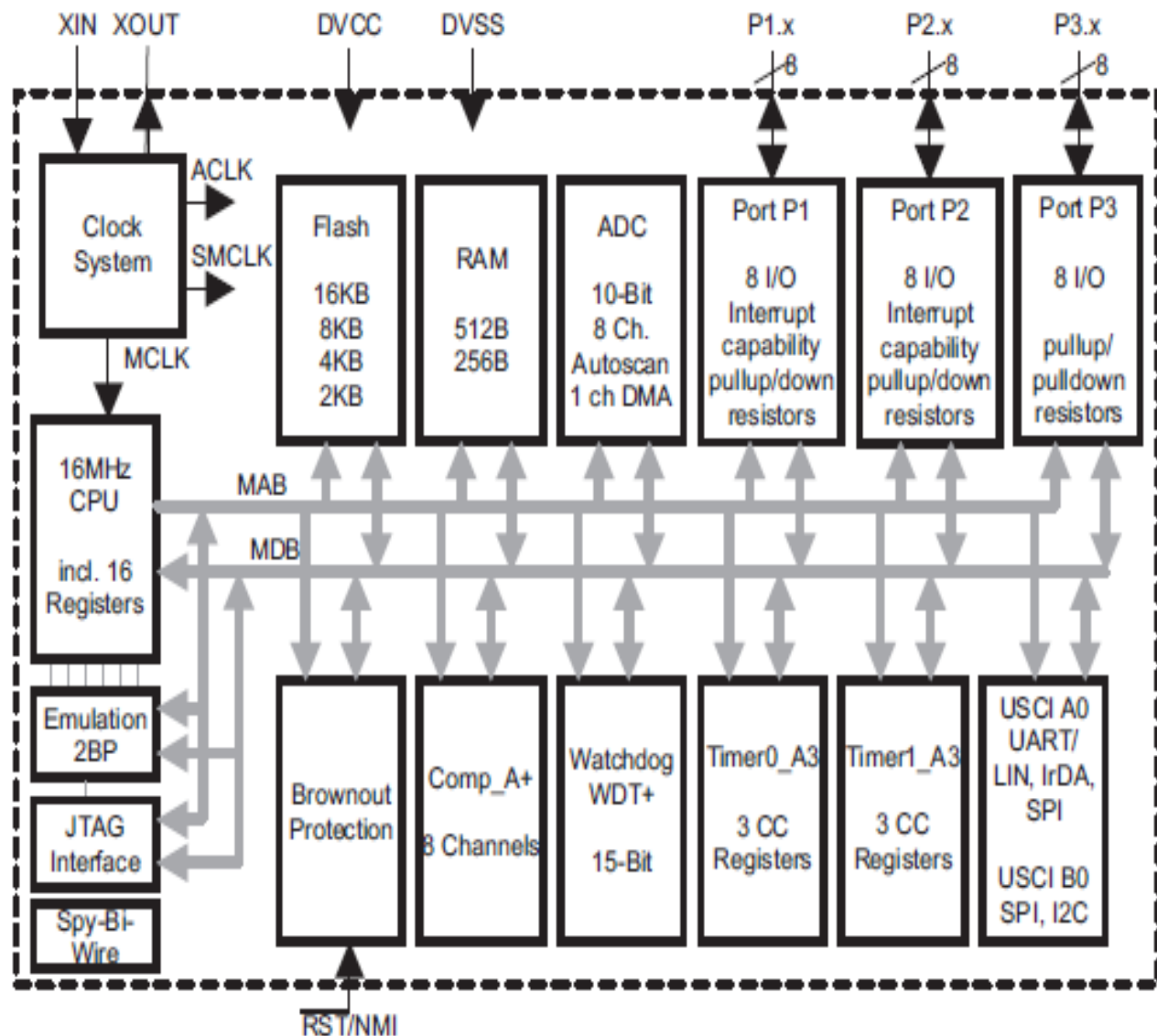


NOTE: ADC10 is available on MSP430G2x53 devices only.

NOTE: The pulldown resistors of port P3 should be enabled by setting P3REN.x = 1.



## Functional Block Diagram, MSP430G2x53



**NOTE:** Port P3 is available on 28-pin and 32-pin devices only.



# Digital IO Control:

Digital I/O is configured by user software

- – Input Register PxIN
- – Output Register PxOUT
- – Direction Register PxDIR
- – Pullup/Pulldown Resistor Enable Register PxREN
- – Function Select Registers PxSEL and PxSEL2
  - Most likely will not need to use these
- – Each pin has interrupt capabilities

# *Input Register PxIN*

- Each bit in each PxIN register reflects the value of the input signal at the corresponding I/O pin when the pin is configured as I/O function.
- Bit = 0: The input is low
- Bit = 1: The input is high

# *Output Registers P<sub>x</sub>OUT*

Each bit in each P<sub>x</sub>OUT register is the value to be output on the corresponding I/O pin when the pin is configured as I/O function, output direction, and the pullup/down resistor is disabled.

- Bit = 0: The output is low
- Bit = 1: The output is high

If the pin's pullup/pulldown resistor is enabled, the corresponding bit in the P<sub>x</sub>OUT register selects pullup or pulldown.

- Bit = 0: The pin is pulled down
- Bit = 1: The pin is pulled up

# *Direction Registers PxDIR*

Each bit in each PxDIR register selects the direction of the corresponding I/O pin, regardless of the selected function for the pin. PxDIR bits for I/O pins that are selected for other functions must be set as required by the other function.

- Bit = 0: The port pin is switched to input direction
- Bit = 1: The port pin is switched to output direction

# *Pullup/Pulldown Resistor Enable Registers PxREN*

Each bit in each PxREN register enables or disables the pullup/pulldown resistor of the corresponding I/O pin. The corresponding bit in the PxOUT register selects if the pin is pulled up or pulled down.

- Bit = 0: Pullup/pulldown resistor disabled
- Bit = 1: Pullup/pulldown resistor enabled

# *Function Select Registers PxSEL and PxSEL2*

- Port pins are often multiplexed with other peripheral module functions. See the device-specific data sheet to determine pin functions. Each PxSEL and PxSEL2 bit is used to select the pin function - I/O port or peripheral module function.

PxSEL2	PxSEL	Pin Function
0	0	I/O function is selected.
0	1	Primary peripheral module function is selected.
1	0	Reserved. See device-specific data sheet.
1	1	Secondary peripheral module function is selected.

# How to enable the pins or bits

- Each pin in port can also acts as a register.
- If the port 1 has 8 pins (p1.0 to p1.7)
- To enable the pins: just set 0 or 1
- For example set P1.1 & P1.6 and form the hexadecimal code

P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0
0	1	0	0	0	0	1	0

4                      2

- Write as 0x42

# Low power modes

Mode	CPU and Clocks Status
Active	CPU is active, all enabled clocks are active
LPM0	CPU, MCLK are disabled, SMCLK, ACLK are active
LPM1	CPU, MCLK are disabled. DCO and DC generator are disabled if the DCO is not used for SMCLK. ACLK is active.
LPM2	CPU, MCLK, SMCLK, DCO are disabled. DC generator remains enabled. ACLK is active.
LPM3	CPU, MCLK, SMCLK, DCO are disabled. DC generator disabled. ACLK is active.
LPM4	CPU and all clocks disabled



- **#define LPM0\_bits (CPUOFF)**
- **#define LPM1\_bits (SCG0+CPUOFF)**
- **#define LPM2\_bits (SCG1+CPUOFF)**
- **#define LPM3\_bits (SCG1+SCG0+CPUOFF)**
- **#define LPM4\_bits (SCG1+SCG0+OSCOFF+CPUOFF)**

- **#define LPM0       \_bis\_SR\_register(LPM0\_bits)       /\* Enter Low Power Mode 0 \*/**
- **#define LPM0\_EXIT   \_bic\_SR\_register\_on\_exit(LPM0\_bits) /\* Exit Low Power Mode 0 \*/**
- **#define LPM1       \_bis\_SR\_register(LPM1\_bits)       /\* Enter Low Power Mode 1 \*/**
- **#define LPM1\_EXIT   \_bic\_SR\_register\_on\_exit(LPM1\_bits) /\* Exit Low Power Mode 1 \*/**
- **#define LPM2       \_bis\_SR\_register(LPM2\_bits)       /\* Enter Low Power Mode 2 \*/**
- **#define LPM2\_EXIT   \_bic\_SR\_register\_on\_exit(LPM2\_bits) /\* Exit Low Power Mode 2 \*/**
- **#define LPM3       \_bis\_SR\_register(LPM3\_bits)       /\* Enter Low Power Mode 3 \*/**
- **#define LPM3\_EXIT   \_bic\_SR\_register\_on\_exit(LPM3\_bits) /\* Exit Low Power Mode 3 \*/**
- **#define LPM4       \_bis\_SR\_register(LPM4\_bits)       /\* Enter Low Power Mode 4 \*/**
- **#define LPM4\_EXIT   \_bic\_SR\_register\_on\_exit(LPM4\_bits) /\* Exit Low Power Mode 4 \*/**

# Led toggling:

```
■ #include <msp430g2553.h>
■ unsigned int i;
■ void main(void)
■ {
■     WDTCTL=WDTPW + WDT HOLD;// watch dog timer disabled
■     P1DIR |= 0X41; // here itself P1OUT = 0100 0001 //led on the pin
                                   will glow
■     while(1)
■     {
■
■         P1OUT ^= 0X41; //0100 0001 Exor 0100 0001= 0000 0000// led
                                   will off for the given delay
■         for(i=0;i<50000;i++);
■     }
■ }
```

# Analog to digital module

- The ADC10 module is a high-performance 10-bit analog-to-digital converter.

Register	Short Form	Register Type	Address	Initial State
ADC10 input enable register 0	ADC10AE0	Read/write	04Ah	Reset with POR
ADC10 input enable register 1	ADC10AE1	Read/write	04Bh	Reset with POR
ADC10 control register 0	ADC10CTL0	Read/write	01B0h	Reset with POR
ADC10 control register 1	ADC10CTL1	Read/write	01B2h	Reset with POR
ADC10 memory	ADC10MEM	Read	01B4h	Unchanged
ADC10 data transfer control register 0	ADC10DTC0	Read/write	048h	Reset with POR
ADC10 data transfer control register 1	ADC10DTC1	Read/write	049h	Reset with POR
ADC10 data transfer start address	ADC10SA	Read/write	01BCh	0200h with POR

Refer the data sheet page no: 558

# Universal Serial Communications Interface (USCI)

- The USCI module is used for serial data communication. The USCI module supports synchronous communication protocols such as SPI (3 or 4 pin) and I2C, an asynchronous communication protocols such as UART, enhanced UART with automatic baudrate detection (LIN), and IrDA. Not all packages support the USCI functionality.
- ✓ USCI\_A0 provides support for SPI (3 or 4 pin), UART, enhanced UART, and IrDA.
- ✓ USCI\_B0 provides support for SPI (3 or 4 pin) and I2C.

---

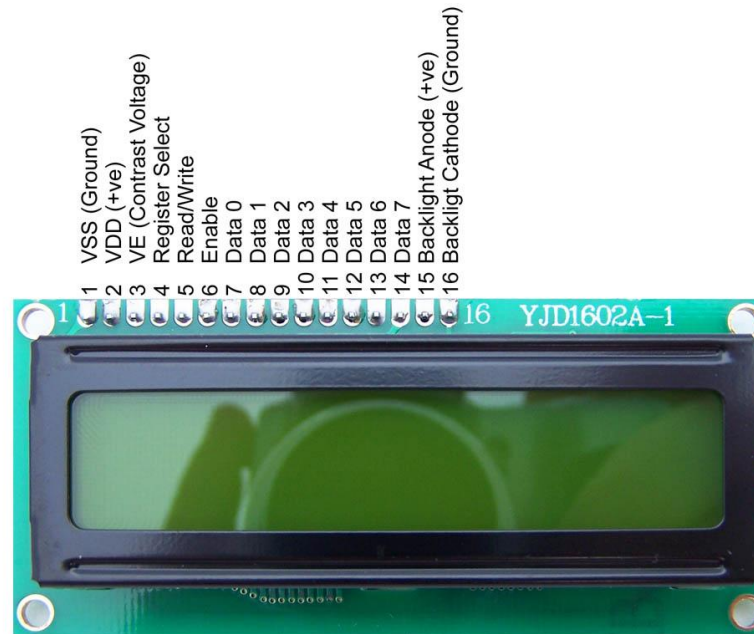
# UART Steps:

1. The I/O pins have to be selected for the UART function
2. A clock with a certain frequency must be sourced to the UART module
3. Enable UART TX or RX or Both
4. Select the byte format as 7- or 8-bit data with odd, even, or non-parity
5. Set the baud generator correctly so as to get a correct baud rate from the clock sourced
6. Enable the module

Table 15-6. USCI\_A0 Control and Status Registers

Register	Short Form	Register Type	Address	Initial State
USCI_A0 control register 0	UCA0CTL0	Read/write	060h	Reset with PUC
USCI_A0 control register 1	UCA0CTL1	Read/write	061h	001h with PUC
USCI_A0 Baud rate control register 0	UCA0BR0	Read/write	062h	Reset with PUC
USCI_A0 baud rate control register 1	UCA0BR1	Read/write	063h	Reset with PUC
USCI_A0 modulation control register	UCA0MCTL	Read/write	064h	Reset with PUC
USCI_A0 status register	UCA0STAT	Read/write	065h	Reset with PUC
USCI_A0 receive buffer register	UCA0RXBUF	Read	066h	Reset with PUC
USCI_A0 transmit buffer register	UCA0TXBUF	Read/write	067h	Reset with PUC
USCI_A0 Auto baud control register	UCA0ABCTL	Read/write	05Dh	Reset with PUC
USCI_A0 IrDA transmit control register	UCA0IRTCTL	Read/write	05Eh	Reset with PUC
USCI_A0 IrDA receive control register	UCA0IRRCTL	Read/write	05Fh	Reset with PUC
SFR interrupt enable register 2	IE2	Read/write	001h	Reset with PUC
SFR interrupt flag register 2	IFG2	Read/write	003h	00Ah with PUC

# 16X2 LCD PIN OUT





# EMBEDDED COMPANIES

- Intel
- Texas Instruments
- Freescale
- Philips
- Samsung
- LG Electronics
- Tata Elxsi/Sasken
- Ittiam Systems
- Infosys/TCS
- HCL
- Technologies/Wipro
- Analog Devices
- Mphasis/BFL
- Symphony
- Sonata Software
- Mistral/eInfochips
- Dexcel Designs
- Robosoft/Yindusoft
- Qualcomm
- Etc.....

# Thank you

