A D PATEL INSTITUTE OF TECHNOLOGY
NEW V V NAGAR

(Affiliated to Gujarat Technological University)

ELECTRONICS AND COMMUNICATION ENGINEERING (ECE) DEPARTMENT

**PROJECT-2**
**REPORT**
**SUBJECT CODE:181105**

# Multi-variable Parameter Interfacing Using RS 485

Submitted by:
Patel Jay R (090010111027)
Pipalia Jaydeep P (090010111035)
Ridham Mayani S (090010111017)

Guided by:
Ami Shah
Assistant Professor,
ADIT

# Acknowledgements:

We are fortunate to have many useful suggestions for our project from faculty Members and student friends, which has helped us in improving performance of project. We have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organizations. We would like to extend our sincere thanks to all of them.

We are highly thankful to Prof. AMI SHAH for their guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing full the project.

We wish to express my sincere thanks to our team members as well as our college faculty for their technical support which helped us to enhance our project performance.

We feel extremely happy to come out with this project, with support of many individuals who have willingly helped us out with their abilities and we are sincerely thankful to them.

| | | |
|---|---|---|
| Jay Patel | Mayani Ridham | Pipalia jaydeep |
| (090010111027) | (090010111017) | (090010111035) |

# Abstract:

System is basically a data acquisition system. Here master and slaves are connected on a bus such that they can communicate. The number of slaves or masters that the bus can support and whether the communication is master slave or multi master depends on the protocol selected.

Taking an example of a coal mine where in there are various chambers, in a coal mine there is always a risk of gas leakage which may lead to accidents if not monitored at regular intervals. A simple solution to this is an acquisition system where in every chamber has a gas sensor and a slave unit that alerts the master of gas leakages, there are multiple slaves connected to the master unit on a single bus. Such a system allows u to precisely locate the chamber that has gas leakage and avoid accidents.

Such a system is also used in automation industries where in different variable parameters need to be monitored in different units located far off and give the results to the central ECU. Can be used to monitor sensors at places which are not physically accessible like a high temperature chamber, poisonous gas chamber etc.

**A. D. Patel Institute of Technology**

**Department of Electronics and Communication Engineering**

**New V. V. Nagar**

## CERTIFICATE

This is to certify that Mr. Jay Patel ID No. 090010111027 of final year Bachelor of Engineering (Electronics and Communication) has satisfactorily completed his project work entitled "**Multi variable parameter interfacing using RS 485**" in academic year 2012-13 for partial fulfillment of the award of Bachelor of Engineering by Gujarat Technological University.

Date:

Prof. Ami Shah
**Project Guide**

Dr. Vishvjit Thakar
**Head of Department**

**A. D. Patel Institute of Technology**

**Department of Electronics and Communication Engineering**

**New V. V. Nagar**

## CERTIFICATE

This is to certify that Mr. Ridham Mayani ID No. 090010111017 of final year Bachelor of Engineering (Electronics and Communication) has satisfactorily completed his project work entitled "**Multi variable parameter interfacing using RS 485**" in academic year 2012-13 for partial fulfillment of the award of Bachelor of Engineering by Gujarat Technological University.

Date:

Prof. Ami Shah
**Project Guide**

Dr. Vishvjit Thakar
**Head of Department**

**A. D. Patel Institute of Technology**

**Department of Electronics and Communication Engineering**

**New V. V. Nagar**

## CERTIFICATE

This is to certify that Mr. Jaydeep Pipalia ID No. 090010111035 of final year Bachelor of Engineering (Electronics and Communication) has satisfactorily completed his project work entitled "**Multi variable parameter interfacing using RS 485**" in academic year 2012-13 for partial fulfillment of the award of Bachelor of Engineering by Gujarat Technological University.

Date:

Prof. Ami Shah
**Project Guide**

Dr. Vishvjit Thakar
**Head of Department**

# Table of Contents

# List of figures and tables

# CHAPTER 1
# INTRODUCTION

## 1.1 Overview:

Our project is based on developing a data acquisition system that basically demands master slave communication, multiple slaves connected to a master. The slaves send the information to the master unit. The system is a wired system and works over long range of up to 1.2 km.

## 1.2 Major work area of our project:

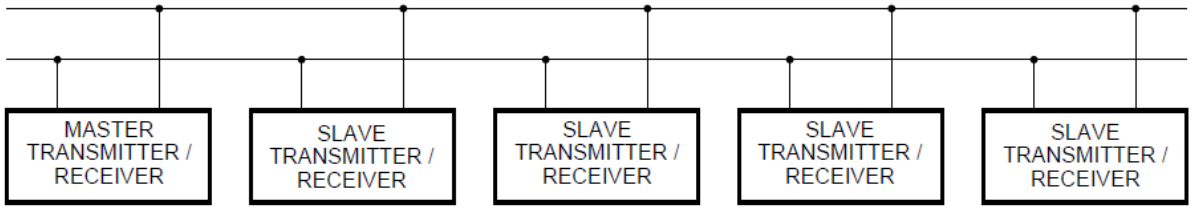
To study different protocols, their practical implementation and finally selecting a protocol that best suits our application.

- RS 232 (UART) protocol.
- I2C protocol.
- SPI-Serial Peripheral Interface protocol.
- CAN-Control Area Network protocol.
- RS 485 advanced version of RS 232.

## 1.3 Detailed description of system:

System is basically a data acquisition system. Here master and slaves are connected on a bus such that they can communicate. The number of slaves or masters that the bus can support and whether the communication is master slave or multi master depends on the protocol selected. Taking an example of a coal mine where in there are various chambers, in a coal mine there is always a risk of gas leakage which may lead to accidents if not monitored at regular intervals. A simple solution to this is an acquisition system where in every chamber has a gas sensor and a slave unit that alerts the master of gas leakages, there are multiple slaves connected to the master unit on a single bus. Such a system allows u to precisely locate the chamber that has gas leakage and avoid accidents. Such a system is also used in automation industries where in different variable parameters need to be monitored in different units located far off and give the results to the central ECU. Can be used to monitor sensors at places which are not physically accessible like a high temperature chamber, poisonous gas chamber etc.

# 1.4 Block Diagram:




System configuration.

**Fig: 1.4.1: System Configuration**

# CHAPTER 2
# Protocols

## 2.1 RS 232 (UART)

### 2.1.1 Introduction:

Information being transferred between data processing equipment and peripherals is in the form of digital data which is transmitted in either a serial or parallel mode. Parallel communications are used mainly for connections between test instruments or computers and printers, while serial is often used between computers and other peripherals. Serial transmission involves the sending of data one bit at a time, over a single communications line. In contrast, parallel communications require at least as many lines as there are bits in a word being transmitted (for an 8-bit word, a minimum of 8 lines are needed). Serial transmission is beneficial for long distance communications, whereas parallel is designed for short distances or when very high transmission rates are required.[3]

### 2.1.2 Standards:

One of the advantages of a serial system is that it lends itself to transmission over telephone lines. The serial digital data can be converted by modem, placed onto a standard voice-grade telephone line, and converted back to serial digital data at the receiving end of the line by another modem. Officially, RS-232 is defined as the "Interface between data terminal equipment and data communications equipment using serial binary data exchange." This definition defines data terminal equipment (DTE) as the computer, while data communications equipment (DCE) is the modem. A modem cable has pin-to-pin connections, and is designed to connect a DTE device to a DCE device.

### 2.1.3 Interfaces:

In addition to communications between computer equipment over telephone lines, RS-232 is now widely used for direct connections between data acquisition devices and computer systems. As in the definition of RS-232, the computer is data transmission equipment (DTE). However, many interface products are not data communications equipment (DCE). Null modem cables are designed for this situation; rather than having the pin- to-pin connections of modem cables, null modem cables have different internal wiring to allow DTE devices to communicate with one another.

### 2.1.4 Cabling Options:

RS-232 cables are commonly available with 4, 9 or 25-pin wiring. The 25-pin cable connects every pin; the 9-pin cables do not include many of the uncommonly used connections; 4-pin cables provide the bare minimum connections, and have jumpers to provide "handshaking" for those devices that require it. These jumpers connect pins 4, 5 and 8, and also pins 6 and 20. The advent of the IBM PC AT has created a new wrinkle in RS-232 communications. Rather than having the standard 25-pin connector, this computer and many new expansion boards for PC's

feature a 9-pin serial port. To connect this port to a standard 25- pin port, a 9-to-25-pin adaptor cable can be utilized, or the user can create his own cable specifically for that purpose.[3]

## 2.1.5 Selecting a Cable:

The major consideration in choosing an RS-232 cable is, what devices are to be connected? First, are you connecting two DTE devices (null modem cable) or a DTE device to a DCE device (modem cable)? Second, what connectors are required on each end, male or female, 25-pin or 9-pin (AT style)? Usually, it is recommended that the user obtain the two devices to be connected, and then determine which cable is required.

## 2.1.6 RS-232 Specifications:

**TRANSMITTED SIGNAL VOLTAGE LEVELS:**
Binary 0: +5 to +15 VDC (called a "space" or "on")
Binary 1: -5 to -15 VDC (called a "mark" or "off")

**RECEIVED SIGNAL VOLTAGE LEVELS:**
Binary 0: +3 to +13 VDC
Binary 1: -3 to -13 VDC

**DATA FORMAT:**
Start bit: Binary 0
Data: 5, 6, 7 or 8 bits
Parity: Odd, even, mark or space (not used with 8-bit data)
Stop bit: Binary 1, one or two bits

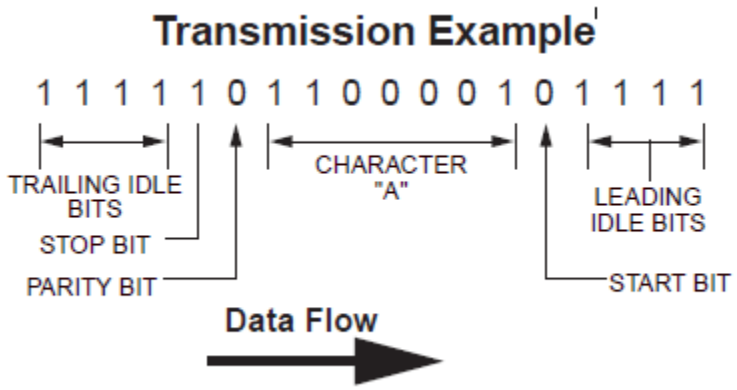



**Fig: 2.1.1: Data Format**

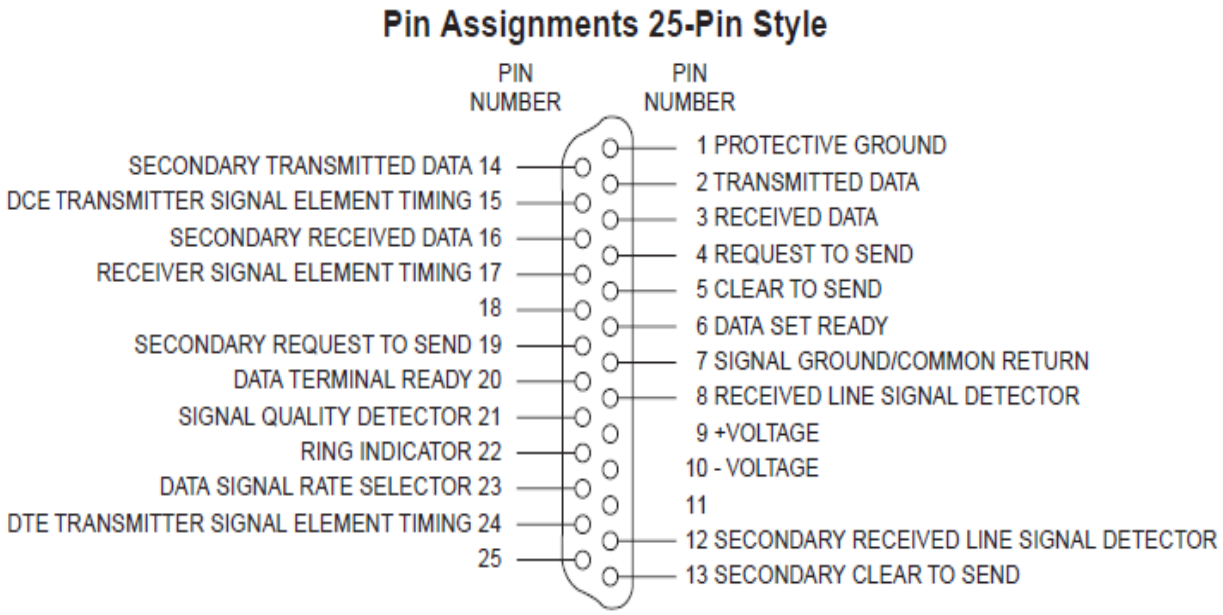



**Fig: 2.1.2: Pin Assignment**

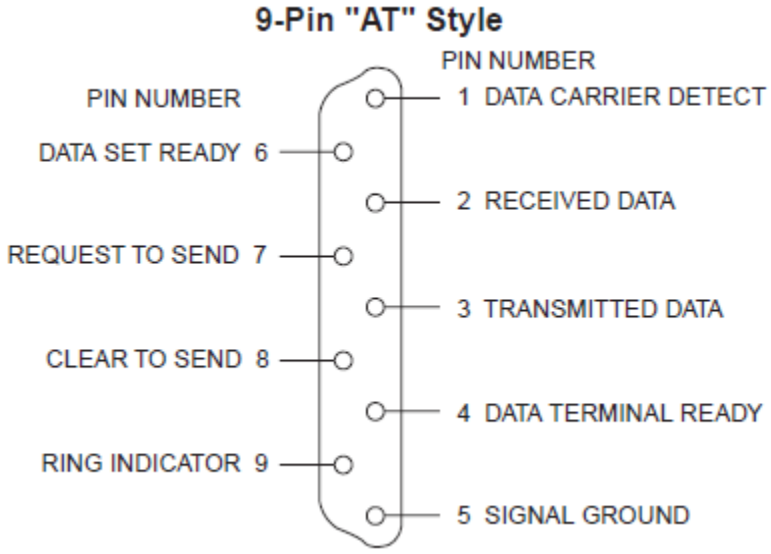



**Fig: 2.1.3: 9-Pin Assignment**

# DTE & DCE

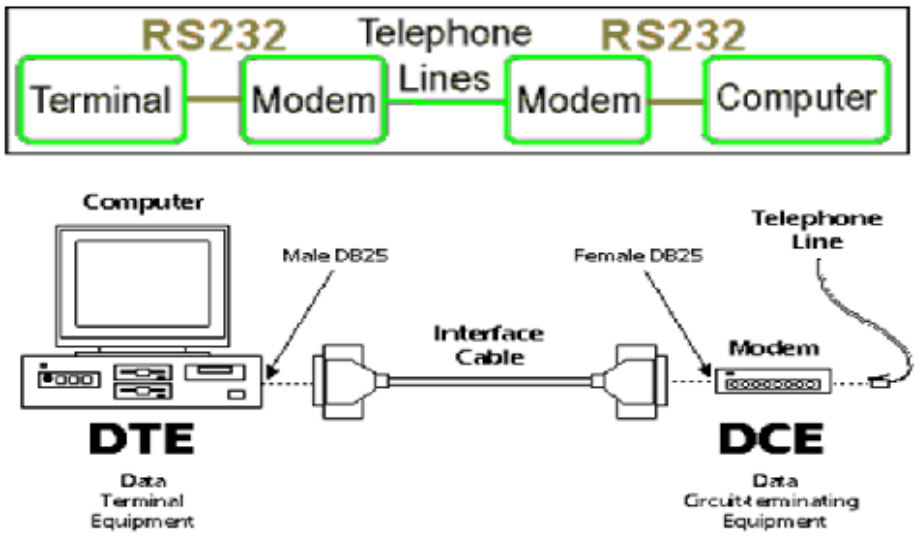
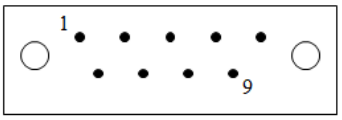



**Fig: 2.1.4 RS232 Connections**

## RS 232 DB-9 PIN CONNECTOR




| | Pin | | Description |
|---|---|---|---|
| 1 | $\overline{\text{DCD}}$ | --- | Data Carrier Detect |
| 2 | RxD | --- | Receiver Data |
| 3 | TxD | --- | Transmitted Data |
| 4 | DTR | --- | Data Terminal Ready |
| 5 | GND | --- | Signal Ground |
| 6 | $\overline{\text{DSR}}$ | --- | Data Set Ready |
| 7 | $\overline{\text{RTS}}$ | --- | Request To Send |
| 8 | $\overline{\text{CTS}}$ | --- | Clear To Send |
| 9 | RI | --- | Ring Indicator |

**Fig: 2.1.5: DB-9 Connector**

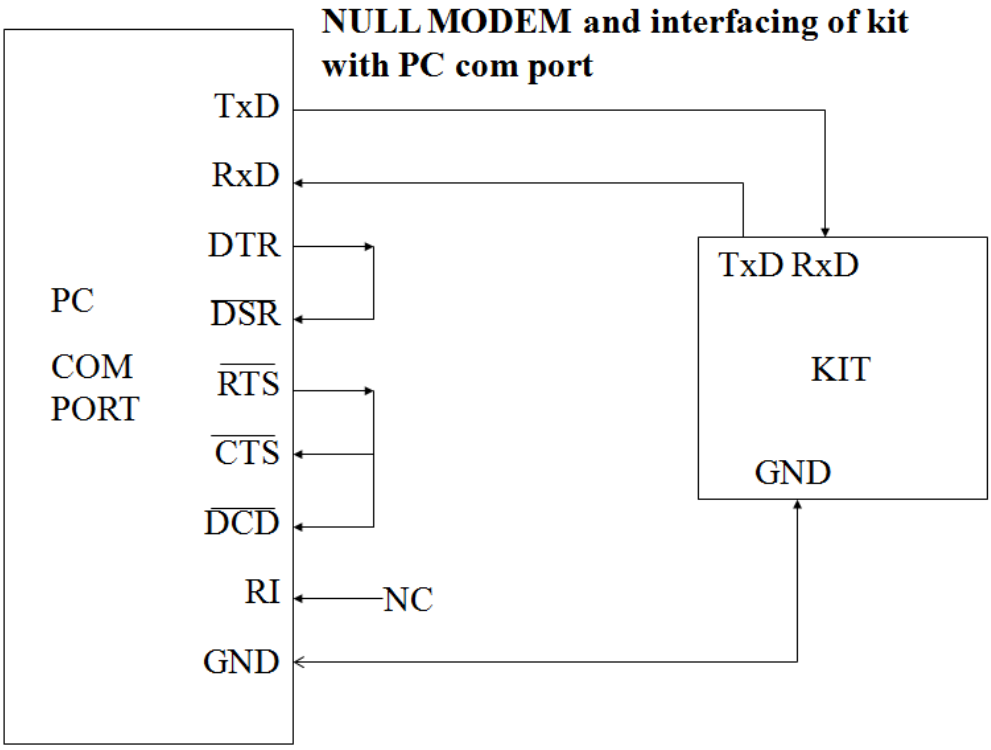



**Fig: 2.1.6 Null Modem Interfacing Configuration**

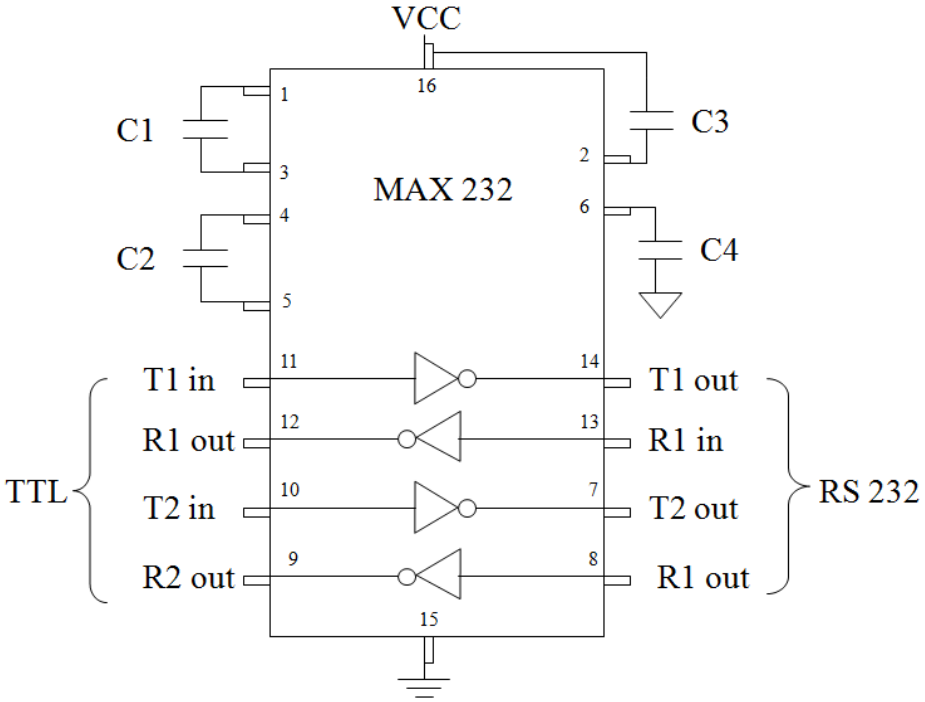



**Fig: 2.1.7: Pin Configuration Of MAX 232**

# 2.2 I2C Protocol

## 2.2.1 Introduction to the I2C-Bus Specification

For 8-bit oriented digital control applications, such as those requiring microcontrollers, certain design criteria can be established:

- A complete system usually consists of at least one microcontroller and other peripheral devices such as memories and I/O expanders
- The cost of connecting the various devices within the system must be minimized
- A system that performs a control function doesn't require high-speed data transfer
- Overall efficiency depends on the devices chosen and the nature of the interconnecting bus structure.

To produce a system to satisfy these criteria, a serial bus structure is needed. Although serial buses don't have the throughput capability of parallel buses, they do require less wiring and fewer IC connecting pins. However, a bus is not merely an interconnecting wire; it embodies all the formats and procedures for communication within the system. Devices communicating with each other on a serial bus must have some form of protocol which avoids all possibilities of confusion, data loss and blockage of information. Fast devices must be able to communicate with slow devices. The system must not be dependent on the devices connected to it, otherwise modifications or improvements would be impossible. A procedure has also to be devised to decide which device will be in control of the bus and when. And, if different devices with different clock speeds are connected to the bus, the bus clock source must be defined. All these criteria are involved in the specification of the I2C-bus.[4]

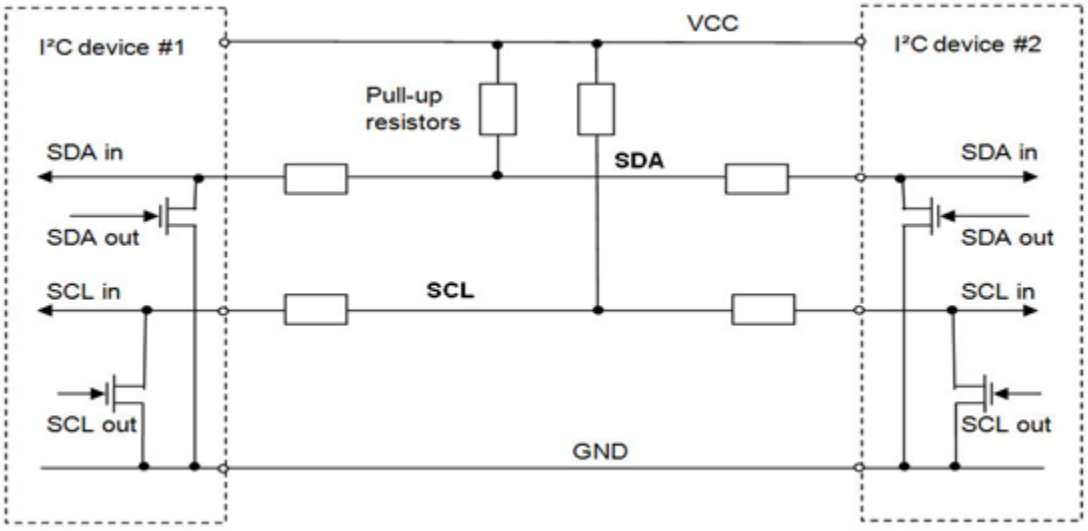



**Fig: 2.2.1: PC bus with 2 devices connected. SDA and SCL are connected to VCC pull up resistor. Each device controls the bus lines outputs with open drain buffer**

## 2.2.2 The I2C-Bus Concept

The I2C-bus supports any IC fabrication process (NMOS, CMOS, bipolar). Two wires, serial data (SDA) and serial clock (SCL), carry information between the devices connected to the bus. Each device is recognized by a unique address (whether it's a microcontroller, LCD driver, memory or keyboard interface) and can operate as either a transmitter or receiver, depending on the function of the device. Obviously an LCD driver is only a receiver, whereas a memory can both receive and transmit data. In addition to transmitters and receivers, devices can also be considered as masters or slaves when performing data transfers (see Table 1). A master is the device which initiates a data transfer on the bus and generates the clock signals to permit that transfer. At that time, any device addressed is considered a slave.

The I2C-bus is a multi-master bus. This means that more than one device capable of controlling the bus can be connected to it. As masters are usually micro-controllers, let's consider the case of a data transfer between two microcontrollers connected to the I2C-bus (see Fig.2). This highlights the master-slave and receiver-transmitter relationships to be found on the I2C-bus. It should be noted that these relationships are not permanent, but only depend on the direction of data transfer at that time.

The transfer of data would proceed as follows:

1) Suppose microcontroller A wants to send information to microcontroller B:

- Microcontroller A (master), addresses microcontroller B (slave)
- Microcontroller A (master-transmitter), sends data to microcontroller B (slave-receiver)
- Microcontroller A terminates the transfer

2) If microcontroller A wants to receive information from microcontroller B:

- Microcontroller A (master) addresses microcontroller B (slave)
- Microcontroller A (master- receiver) receives data from microcontroller B (slave-transmitter)
- Microcontroller A terminates the transfer. Even in this case, the master (microcontroller A) generates the timing and terminates the transfer.

The possibility of connecting more than one microcontroller to the I2C-bus means that more than one master could try to initiate a data transfer at the same time. To avoid the chaos that might ensue from such an event -an arbitration procedure has been developed. This procedure relies on the wired-AND connection of all I2C interfaces to the I2C-bus. If two or more masters try to put information onto the bus, the first to produce a 'one' when the other produces a 'zero' will lose the arbitration. The clock signals during arbitration are a synchronized combination of the clocks generated by the masters using the wired-AND connection to the SCL line Generation of clock signals on the I2C-bus is always the responsibility of master devices; each master generates its own clock signals when transferring data on the bus. Bus clock signals from a master can only be altered when they are stretched by a slow-slave device holding-down the clock line, or by another master when arbitration occurs.[4]

## 2.2.3 General Characteristics

Both SDA and SCL are bi-directional lines, connected to a positive supply voltage via a current-source or pull-up resistor (see Fig.3). When the bus is free, both lines are HIGH. The output stages of devices connected to the bus must have an open-drain or open-collector to perform the wired-AND function. Data on the I2C-bus can be transferred at rates of up to 100 kbit/s in the Standard-mode, up to 400 kbit/s in the Fast-mode, and up to 3.4 Mbit/s in the High-speed mode. The number of interfaces connected to the bus is solely dependent on the bus capacitance limit of 400 pF.

Bit Transmission Rates

- Standard-mode I2C: 0 Hz to 100 KHz
- Fast-mode I2C: 0 Hz to 400 KHz
- SM Bus: 10 KHz to 100 KHz

Standard-mode and fast-mode I2C parts can be mixed as long a 0 Hz to 100 KHz range is used.

I2C Transfers

- 9 bits are sent for each transfer.
- First 8 bits are sent by the transmitter – MSB first to LSB last.
- $9^{th}$ bit is an acknowledge bit sent by the receiver.

Acknowledge Bit

- When no peripheral chip (slave) acknowledges a transfer from the micro (master), this is interpreted as an error.
- When the micro acknowledges a transfer from a slave, this is interpreted as a request for the slave to send more data. No acknowledge from the micro is interpreted as a request by the micro for the slave to stop sending data.

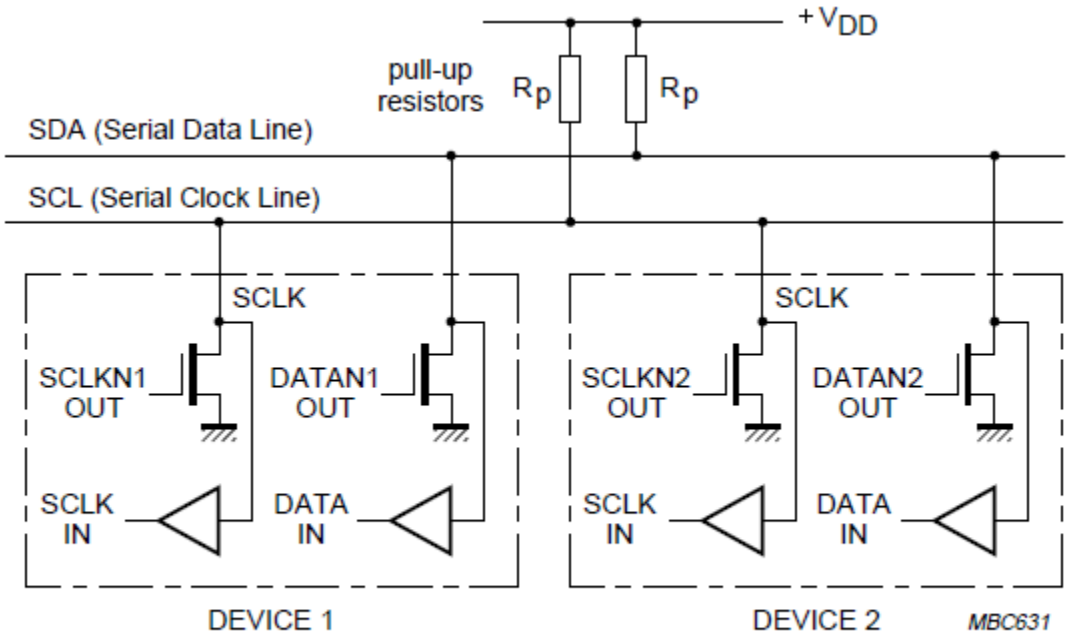
Valid Data and SCL

- Valid data should be on SDA before SCL rises and the valid data should remain on SDA until after SCL falls during the 9 bit transfer.
- Only during two special events (called START and STOP – not part of a 9-bit transfer) should the SDA voltage change while SCL is high.[4]

## 2.2.4 Start and Stop Conditions

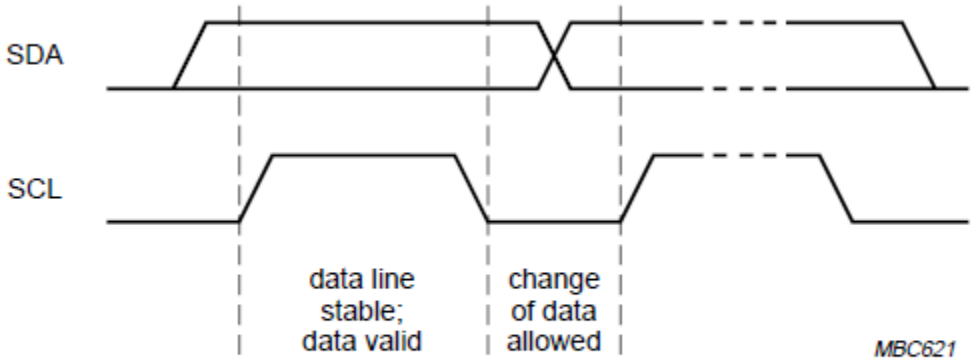
Within the procedure of the I2C-bus, unique situations arise which are defined as START (S) and STOP (P) conditions (see Fig.5). A HIGH to LOW transition on the SDA line while SCL is HIGH is one such unique case. This situation indicates a START condition. A LOW to HIGH transition on the SDA line while SCL is HIGH defines a STOP condition. START and STOP conditions are always generated by the master. The bus is considered to be busy after the START condition. The bus is considered to be free again a certain time after the STOP condition. The bus stays busy if a repeated START (Sr) is generated instead of a STOP condition. In this respect, the START (S) and repeated START (Sr) conditions are functionally identical. For the remainder of this document, therefore, the S symbol will be used as a generic term to represent both the START and repeated START conditions, unless Sr is particularly relevant. Detection of START and STOP conditions by devices connected to the bus is easy if they incorporate the

necessary interfacing hardware. However, microcontrollers with no such interface have to sample the SDA line at least twice per clock period to sense the transition.[4]



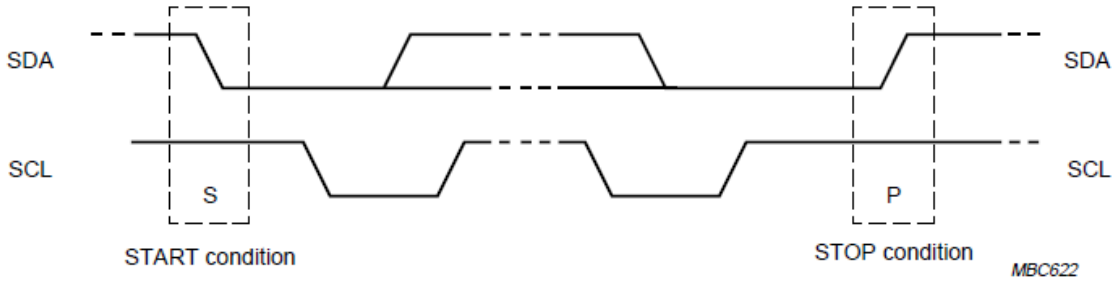

Connection of Standard- and Fast-mode devices to the I²C-bus.

**Fig: 2.2.2: connection of standard and fast mode devices to the I2C bus**




Bit transfer on the I²C-bus.

**Fig: 2.2.3: Bit transfer on the I2C bus**



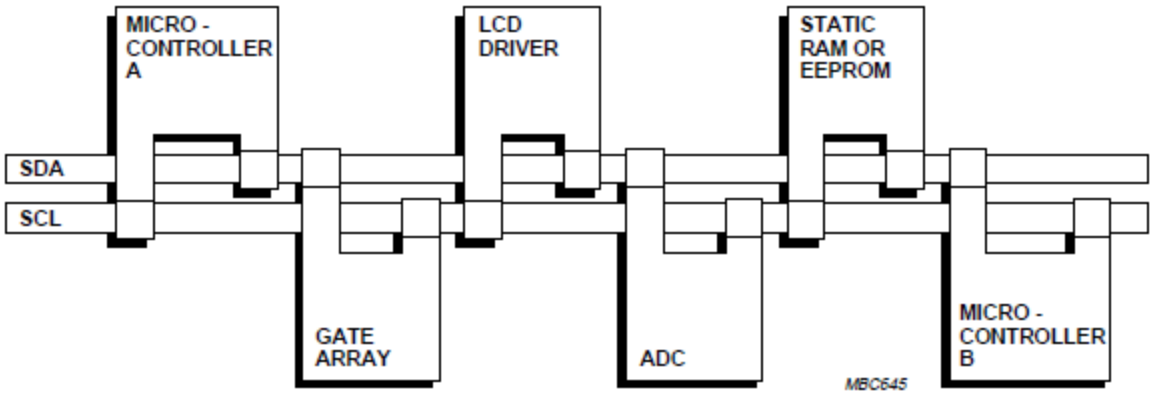

START and STOP conditions.

**Fig: 2.2.4: START and STOP conditions**

**Table 1** Definition of $I^2C$-bus terminology

| TERM | DESCRIPTION |
|---|---|
| Transmitter | The device which sends data to the bus |
| Receiver | The device which receives data from the bus |
| Master | The device which initiates a transfer, generates clock signals and terminates a transfer |
| Slave | The device addressed by a master |
| Multi-master | More than one master can attempt to control the bus at the same time without corrupting the message |
| Arbitration | Procedure to ensure that, if more than one master simultaneously tries to control the bus, only one is allowed to do so and the winning message is not corrupted |
| Synchronization | Procedure to synchronize the clock signals of two or more devices |

**Table 2.2.1 I2C Bus Terminology**




Example of an I²C-bus configuration using two microcontrollers.

**Fig: 2.2.5: configuration using two microcontrollers**

## 2.2.5 Open Drain Outputs

- An open-drain output can either force the output to ground or let the output float, but can not force the output to VDD. Used in all buses.
- A pull-up resistor is a (usually 1KΩ to 100KΩ) resistor connected between VDD and the bus wire. In I2C specification, this resistor is 2.2 KΩ.
- In 89C668 P1.6 and P1.7 are open drain whereas in 8x554 we have to write to certain registers to make the pins open drain.

- Open-collector/open-drain is a circuit technique which allows multiple devices to communicate bi-directionally on a single wire.

- Open-collector/open-drain devices sink (flow) current in their low voltage active (logic 0) state, or are high impedance (no current flows) in their high voltage non-active (logic 1) state. These devices usually operate with an external pull-up resistor that holds the signal line high until a device on the wire sinks enough current to pull the line low. Many devices can be attached to the signal wire. If all devices attached to the wire are in their non-active state, the pull-up will hold the wire at a high voltage. If one or more devices are in the active state, the signal wire voltage will be low.

- The bi-directional nature of an open-collector/open-drain device is what makes this circuit so important in interconnecting many devices on a common line. The I2C Bus and SMBus uses this technique for connecting up to 127 devices.

- Open-drain refers to the drain terminal of a MOS FET transistor. Open-collector is the same concept on a bipolar device.

## 2.2.6 Slave Address

- Each I2C peripheral chip should have a unique *slave address*.
- Slave addresses can be either 7-bit or 10-bit addresses. Almost all systems use 7-bit addressing (so we will not worry about 10-bit addressing here).
- Addresses are assigned by a central authority. Though theoretically 127 chips may be connected in a I2C bus practically the number is far less as it is limited by bus capacitance to a maximum limit of 400pF.

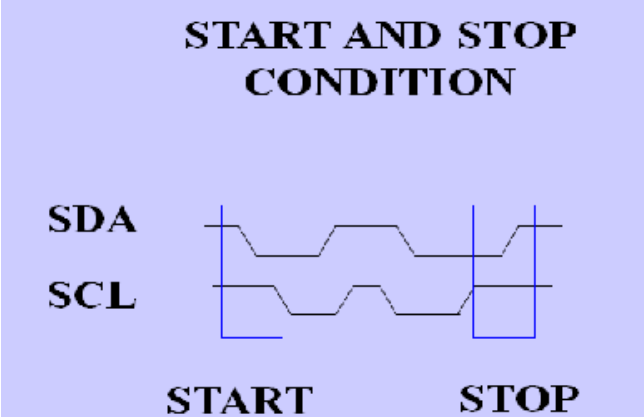



**Fig: 2.2.6: START and STOP conditions**

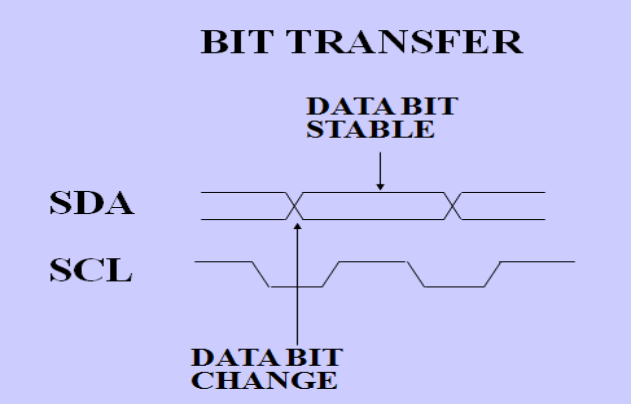



**Fig: 2.2.7: Bit transfer**

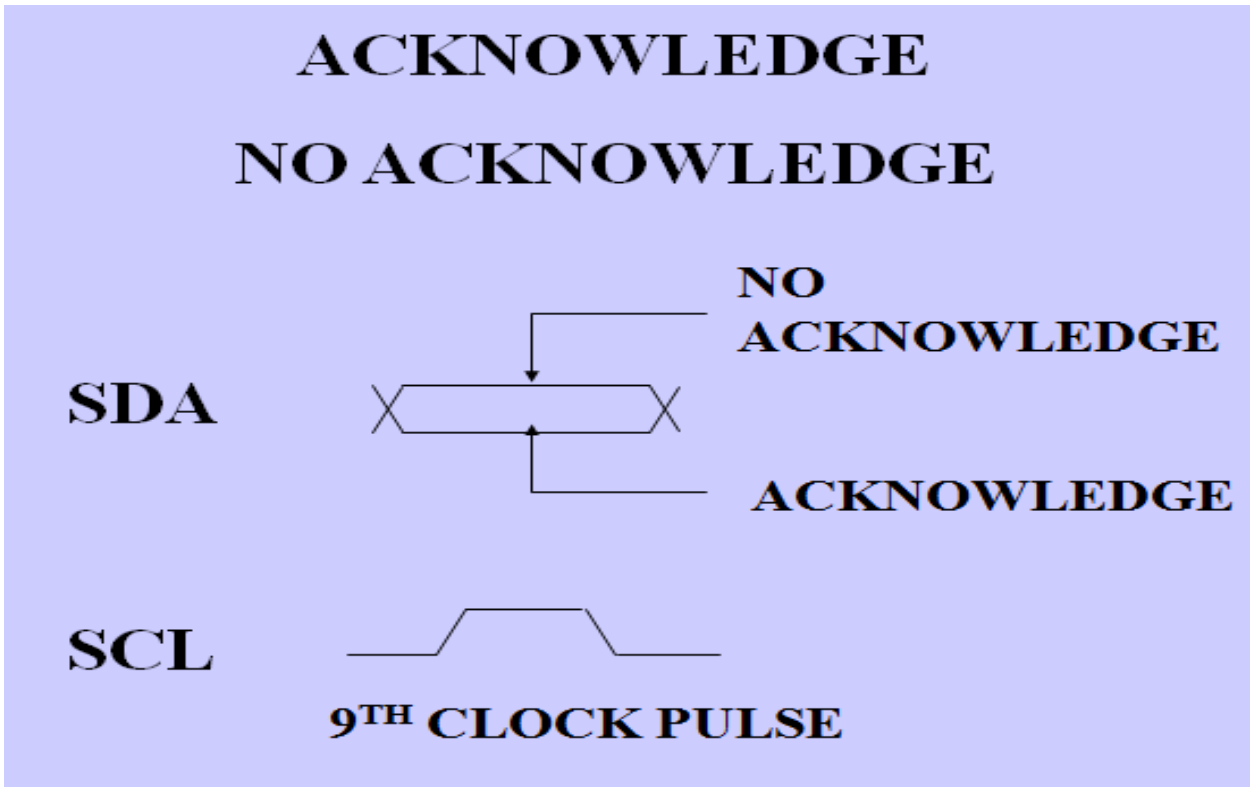



**Fig: 2.2.8: ACK and NO ACK**

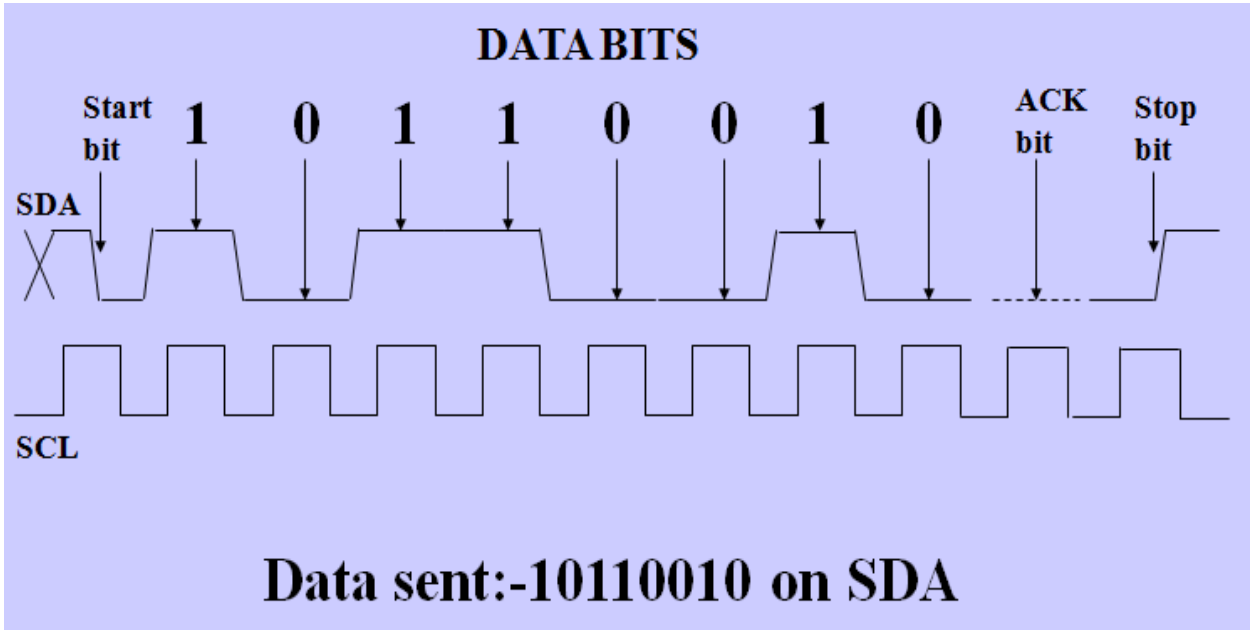



**Fig: 2.2.9: Data sending on SDA**

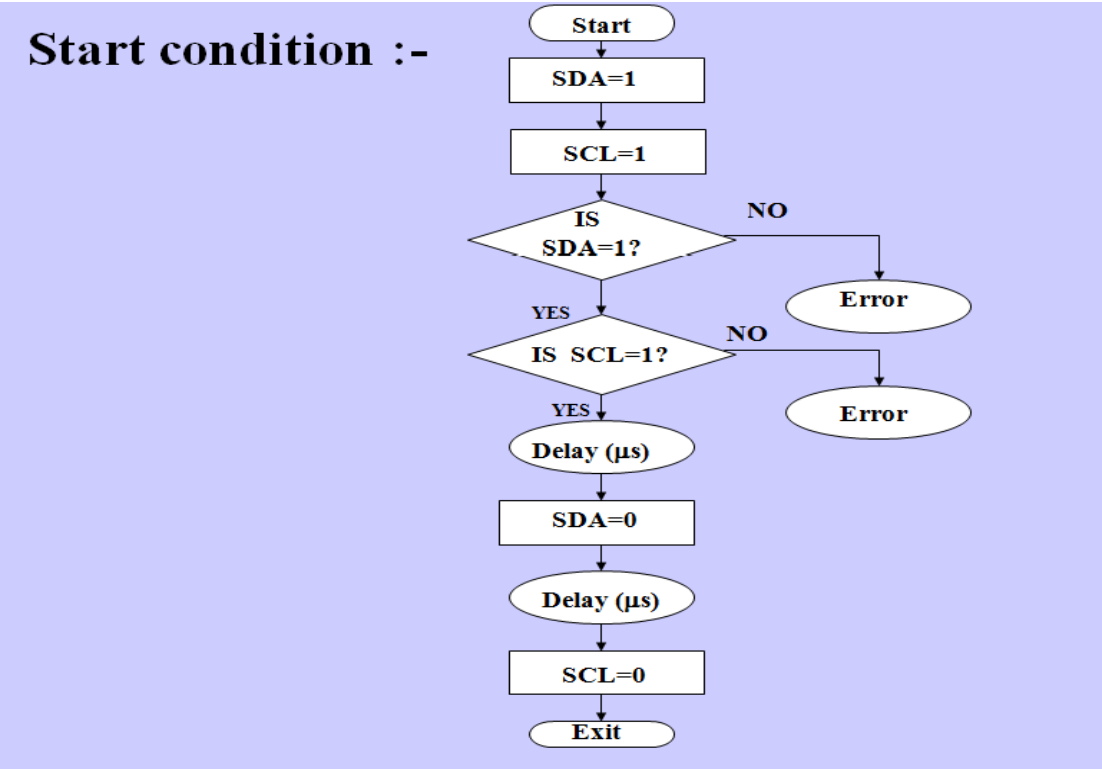



**Fig: 2.2.10: Flow chart of start condition**

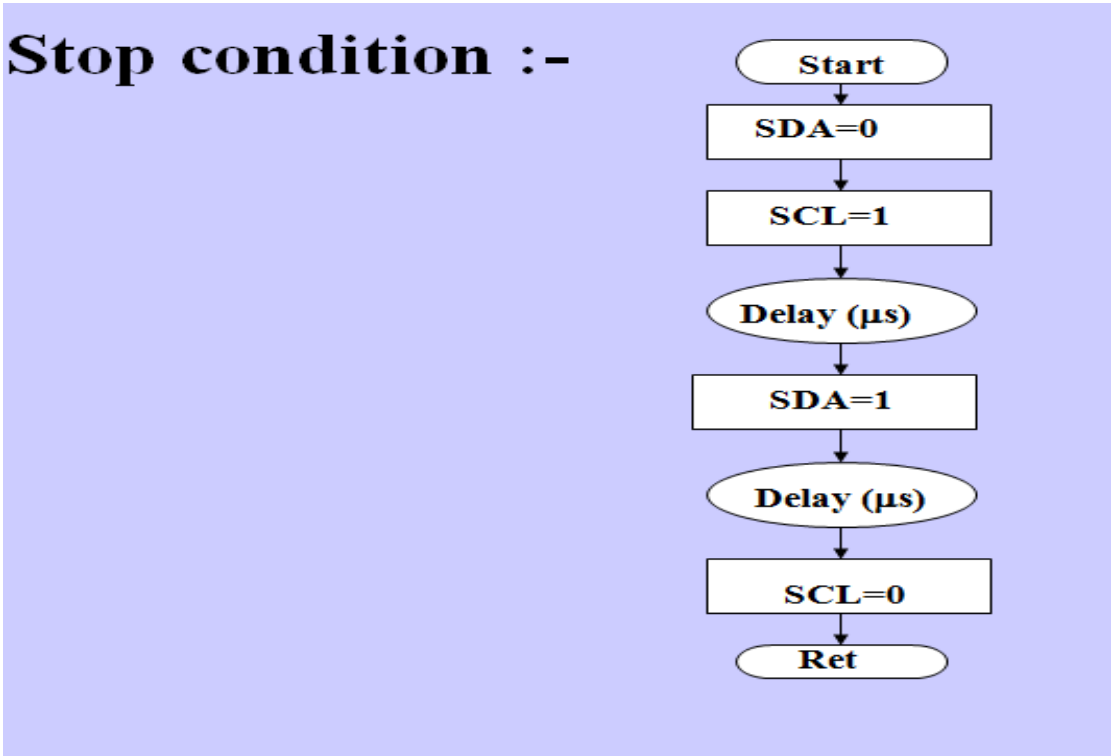



**Fig: 2.2.11: Flow chart of stop condition**

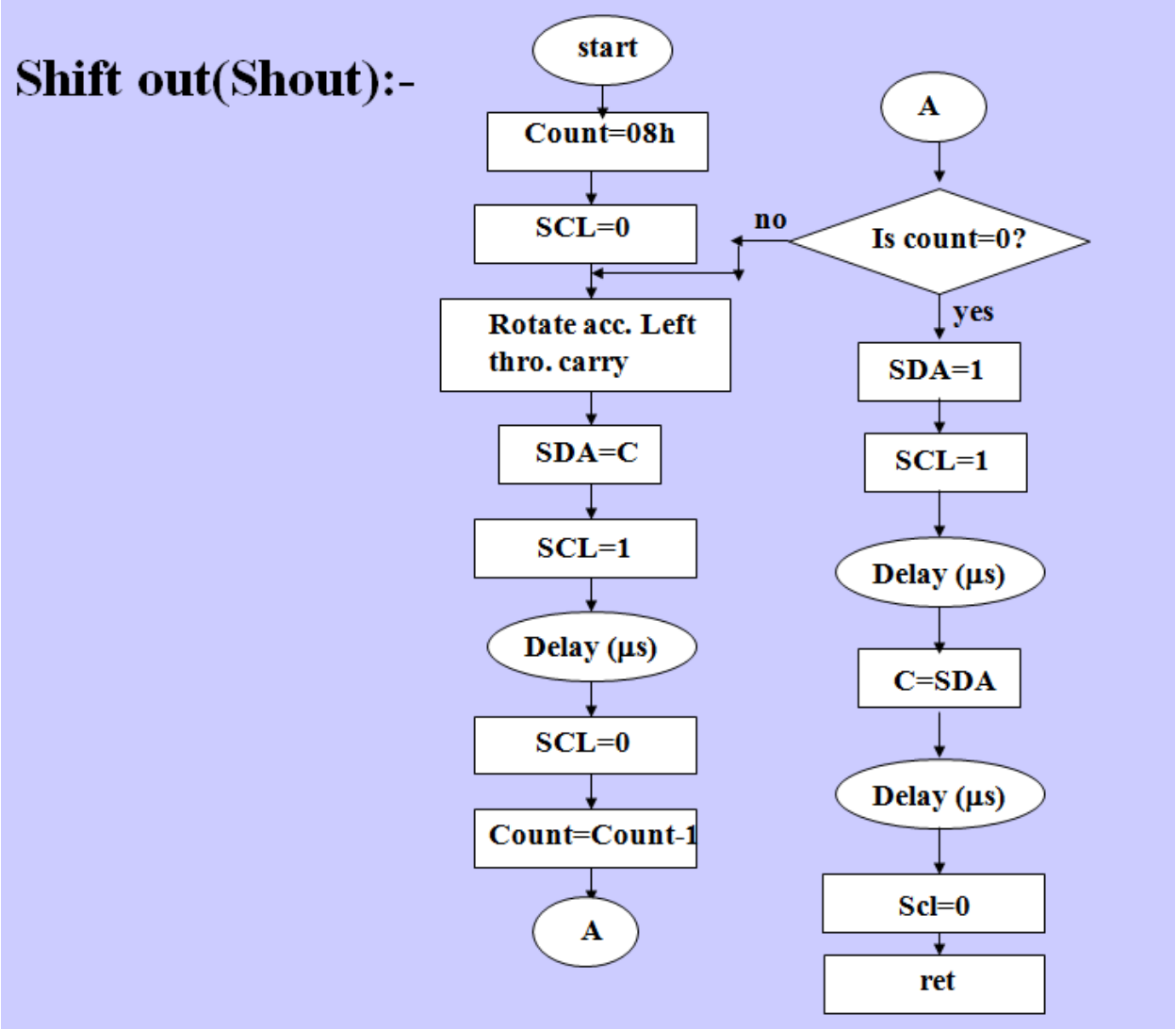



**Fig: 2.2.12: Flow chart of shift out**

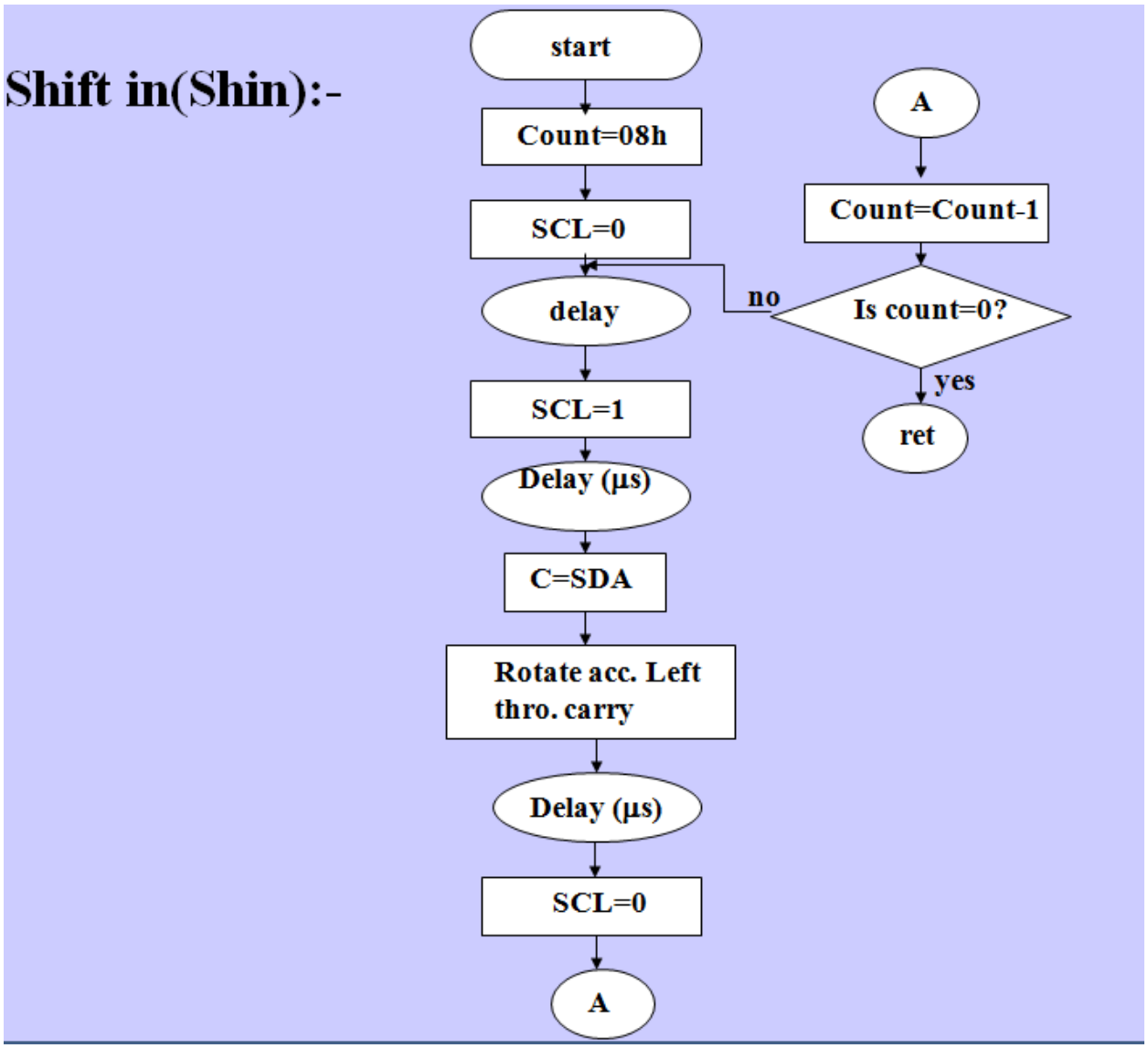



**Fig: 2.2.13: Flow chart of shift in**

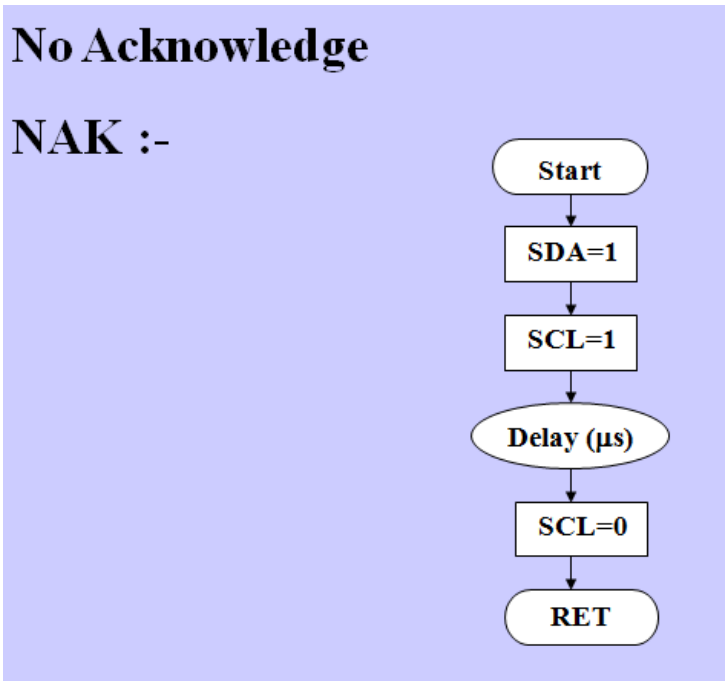



**Fig: 2.2.14: Flow chart of NO ACK**

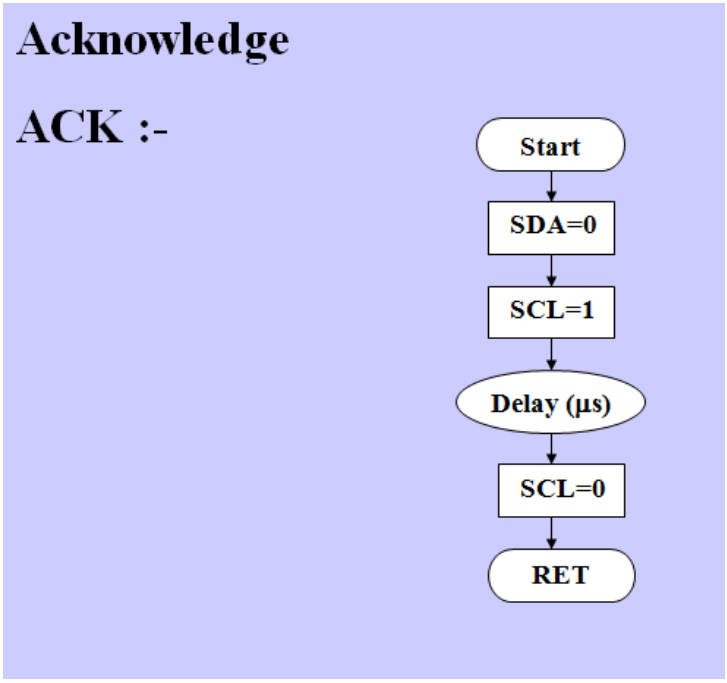



**Fig: 2.2.15: Flow chart of ACK**

# 2.3 SPI Protocol

## 2.3.1 Introduction

SPI is a single-master communication protocol. This means that one central device initiates all the communications with the slaves. When the SPI master wishes to send data to a slave and/or request information from it, it selects slave by pulling the corresponding SS line low and it activates the clock signal at a clock frequency usable by the master and the slave. The master generates information onto MOSI line while it samples the MISO line

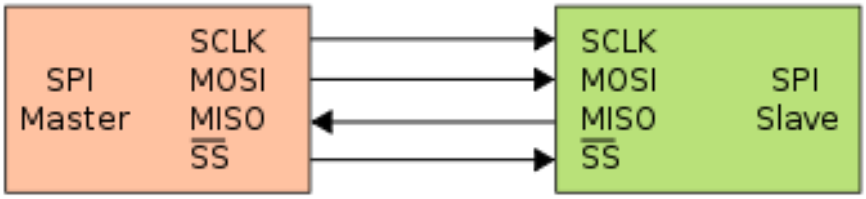



**Fig: 2.3.1: Master Slave connection**

The SPI bus specifies four logic signals:

- SCLK: serial clock (output from master);
- MOSI: master output, slave input (output from master);
- MISO: master input, slave output (output from slave);
- SS: Slave select (active low, output from master).

Alternative naming conventions are also widely used:

- SCLK: SCK, CLK: serial clock (output from master)
- MOSI: SIMO, SDO, DO, DOUT, SI, MTSR: serial data out; data out, serial out, master transmit slave receive
- MISO: SOMI, SDI, DI, DIN, SO, MRST: serial data in; data in, serial in, master receive slave transmit
- SS: nCS, CS, CSB, CSN, nSS, STE: chip select, slave transmit enable (active low, output from master)

The SDI/SDO (DI/DO, SI/SO) convention requires that SDO on the master be connected to SDI on the slave, and vice versa. Chip select polarity is rarely active high, although some notations (such as SS or CS instead of nSS or nCS) suggest otherwise.[5]

## 2.3.2 Operation

The SPI bus can operate with a single master device and with one or more slave devices. If a single slave device is used, the SS pin *may* be fixed to logic low if the slave permits it. Some slaves require the falling edge (high-low transition) of the chip select to initiate an action such as the Maxim MAX1242 ADC, which starts conversion on said transition. With multiple slave devices, an independent SS signal is required from the master for each slave device. Most slave devices have tri-state outputs so their MISO signal becomes high impedance (*logically disconnected*) when the device is not selected. Devices without tri-state outputs can't share SPI bus segments with other devices; only one such slave could talk to the master, and only its chip select could be activated.[5]

## 2.3.3 Data transmission

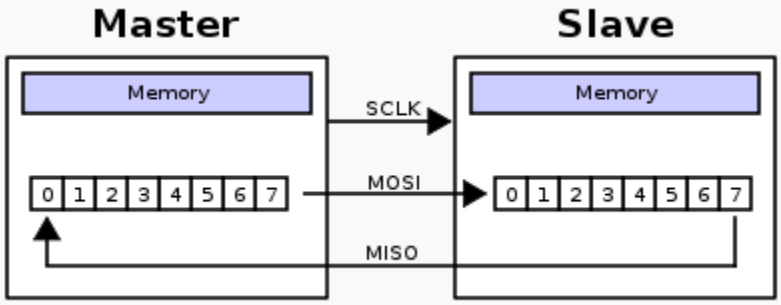



**Fig: 2.3.2: Master Slave Data Transmission**

A typical hardware setup using two shift registers to form an inter-chip circular buffer

To begin a communication, the bus master first configures the clock, using a frequency less than or equal to the maximum frequency the slave device supports. Such frequencies are commonly in the range of 1–100 M Hz.

The master then transmits the logic 0 for the desired chip over chip select line. Logic 0 is transmitted because the chip select line is active low, meaning its *off* state is logic 1; *on* is asserted with logic 0. If a waiting period is required (such as for analog-to-digital conversion), then the master must wait for at least that period of time before starting to issue clock cycles.

During each SPI clock cycle, a full duplex data transmission occurs:

- the master sends a bit on the MOSI line; the slave reads it from that same line
- the slave sends a bit on the MISO line; the master reads it from that same line

Not all transmissions require all four of these operations to be *meaningful* but they do happen.

Transmissions normally involve two shift registers of some given word size, such as eight bits, one in the master and one in the slave; they are connected in a ring. Data is usually shifted out with the most significant bit first, while shifting a new least significant bit into the same register. After that register has been shifted out, the master and slave have exchanged register values. Then each device takes that value and does something with it, such as writing it to memory. If there is more data to exchange, the shift registers are loaded with new data and the process repeats.

Transmissions may involve any number of clock cycles. When there is no more data to be transmitted, the master stops toggling its clock. Normally, it then deselects the slave.

Transmissions often consist of 8-bit words, and a master can initiate multiple such transmissions if it wishes/needs. However, other word sizes are also common, such as 16-bit words for touch screen controllers or audio codecs, like the TSC2101 from Texas Instruments; or 12-bit words for many digital-to-analog or analog-to-digital converters.

Every slave on the bus that hasn't been activated using its chip select line must disregard the input clock and MOSI signals, and must not drive MISO. The master must select only one slave at a time.[5]

## 2.3.4 Clock Polarity and Phase

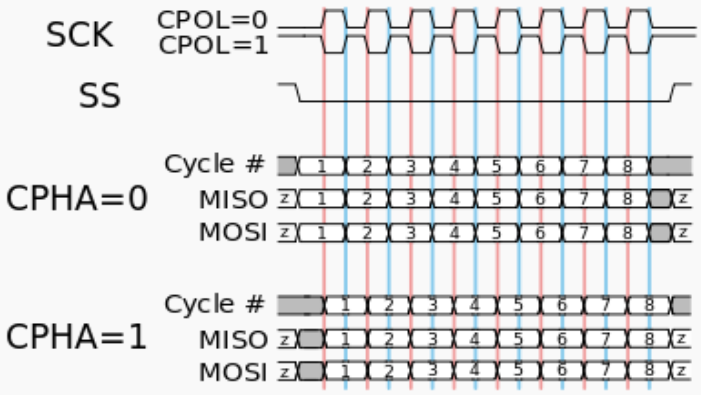



**Fig: 2.3.3: Clock polarity and phase**

Timing diagram above shows clock polarity and phase. The red vertical line represents CPHA=0 and the blue vertical line represents CPHA=1

In addition to setting the clock frequency, the master must also configure the clock polarity and phase with respect to the data. Free scale's SPI Block Guide names these two options as CPOL and CPHA respectively, and most vendors have adopted that convention.

The timing diagram is shown to the right. The timing is further described below and applies to both the master and the slave device.

- At CPOL=0 the base value of the clock is zero
  - For CPHA=0, data is captured on the clock's rising edge (low high transition) and data is propagated on a falling edge (high-low clock transition).
  - For CPHA=1, data is captured on the clock's falling edge and data is propagated on a rising edge.
- At CPOL=1 the base value of the clock is one (inversion of CPOL=0)
  - For CPHA=0, data is captured on clock's falling edge and data is propagated on a rising edge.
  - For CPHA=1, data is captured on clock's rising edge and data is propagated on a falling edge.

That is, CPHA=0 means sample on the leading (first) clock edge, while CPHA=1 means sample on the trailing (second) clock edge, regardless of whether that clock edge is rising or falling. Note that with CPHA=0, the data must be stable for a half cycle before the first clock cycle. For all CPOL and CPHA modes, the initial clock value must be stable before the chip select line goes active.

The MOSI and MISO signals are usually stable (at their reception points) for the half cycle until the next clock transition. SPI master and slave devices may well sample data at different points in that half cycle.

This adds more flexibility to the communication channel between the master and slave. Some products use different naming conventions. For example, the TI MSP430 uses the name UCCKPL instead of CPOL, and its UCCKPH is the inverse of CPHA. When connecting two chips together carefully examine the clock phase initialization values to be sure of using the right settings.[5]

## 2.3.5 Mode Numbers

The combinations of polarity and phases are often referred to as modes which are commonly numbered according to the following convention, with CPOL as the high order bit and CPHA as the low order bit:

| Mode | CPOL | CPHA |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 2 | 1 | 0 |
| 3 | 1 | 1 |

Another commonly used notation represents the mode as a (CPOL, CPHA) tuple; e.g., the value '(0, 1)' would indicate CPOL=0 and CPHA=1

## 2.3.6 Independent slave SPI configuration

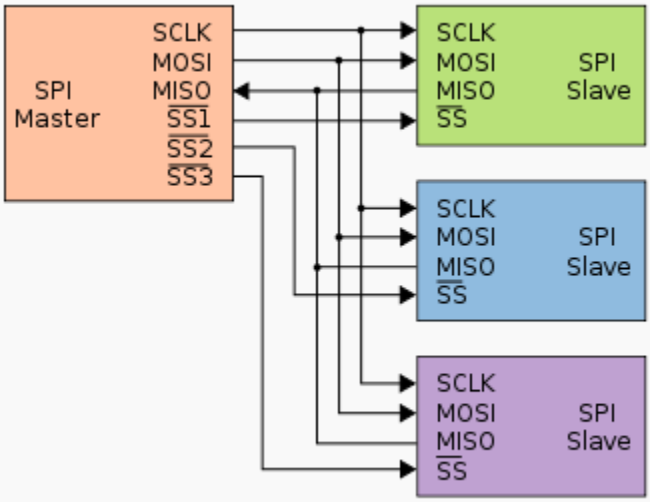



**Fig: 2.3.4: Slave SPI configuration**

Typical SPI bus: master and three independent slaves

In the independent slave configuration, there is an independent chip select line for each slave. This is the way SPI is normally used. Since the MISO pins of the slaves are connected together, they are required to be tri-state pins.

## 2.3.7 Daisy chain SPI configuration

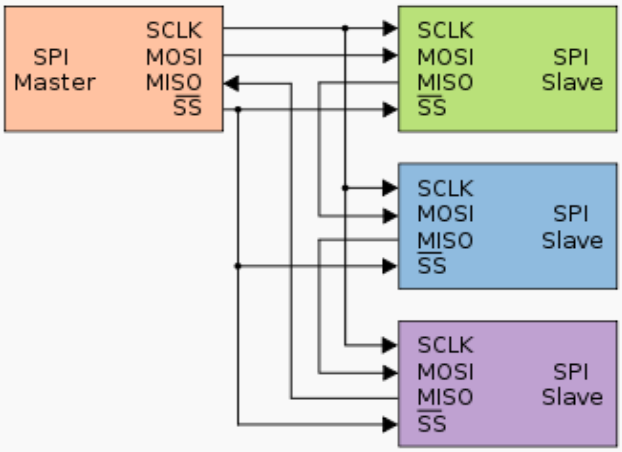



**Fig: 2.3.5: Daisy chain SPI configuration**

Daisy-chained SPI bus: master and cooperative slaves

Some products with SPI bus are designed to be capable of being connected in a daisy chain configuration, the first slave output being connected to the second slave input, etc. The SPI port of each slave is designed to send out during the second group of clock pulses an exact copy of what it received during the first group of clock pulses. The whole chain acts as an SPI communication shift register; daisy chaining is often done with shift registers to provide a bank of inputs or outputs through SPI. Such a feature only requires a single SS line from the master, rather than a separate SS line for each slave.

Applications (discussed later) that require a daisy chain configuration include SGPIO and JTAG.

## 2.3.8 Valid SPI communications

Some slave devices are designed to ignore any SPI communications in which the number of clock pulses is greater than specified. Others don't care, ignoring extra inputs and continuing to shift the same output bit. It is common for different devices to use SPI communications with different lengths, as, for example, when SPI is used to access the scan chain of a digital IC by issuing a command word of one size (perhaps 32 bits) and then getting a response of a different size (perhaps 153 bits, one for each pin in that scan chain).

## 2.3.9 Interrupts

SPI devices sometimes use another signal line to send an interrupt signal to a host CPU. Examples include pen-down interrupts from touch screen sensors, thermal limit alerts from temperature sensors, alarms issued by real time clock chips, SDIO, and headset jack insertions from the sound codec in a cell phone. Interrupts are not covered by the SPI standard; their usage is neither forbidden nor specified by the standard.[5]

## 2.3.10 Pros and cons of SPI

## Advantages

- Full duplex communication
- Higher throughput than I²C or SMBus
- Complete protocol flexibility for the bits transferred
  - Not limited to 8-bit words
  - Arbitrary choice of message size, content, and purpose
- Extremely simple hardware interfacing
  - Typically lower power requirements than I²C or SMBus due to less circuitry (including pull up resistors)
  - No arbitration or associated failure modes
  - Slaves use the master's clock, and don't need precision oscillators
  - Slaves don't need a unique address — unlike I²C or GPIB or SCSI
  - Transceivers are not needed

- ❖ Uses only four pins on IC packages, and wires in board layouts or connectors, much fewer than parallel interfaces
- ❖ At most one unique bus signal per device (chip select); all others are shared
- ❖ Signals are unidirectional allowing for easy Galvanic isolation
- ❖ Not limited to any maximum clock speed, enabling potentially high throughput

## Disadvantages

- Requires more pins on IC packages than I²C, even in the *three-wire* variant
- No in-band addressing; out-of-band chip select signals are required on shared buses
- No hardware flow control by the slave (but the master can delay the next clock edge to slow the transfer rate)
- No hardware slave acknowledgment (the master could be transmitting to nowhere and not knowing it)
- Supports only one master device
- No error-checking protocol is defined
- Generally prone to noise spikes causing faulty communication
- Without a formal standard, validating conformance is not possible
- Only handles short distances compared to RS-232, RS-485, or CAN-bus
- Many existing variations, making it difficult to find development tools like host adapters that support those variations
- SPI does not support hot plugging (dynamically adding nodes).

## 2.3.11 Applications

The board real estate savings compared to a parallel I/O bus are significant, and have earned SPI a solid role in embedded systems. That is true for most system-on-a-chip processors, both with higher end 32-bit processors such as those using ARM, MIPS, or PowerPC and with other microcontrollers such as the AVR, PIC, and MSP430. These chips usually include SPI controllers capable of running in either master or slave mode. In-system programmable AVR controllers (including blank ones) can be programmed using an SPI interface.[4]

Chip or FPGA based designs sometimes use SPI to communicate between internal components; on-chip real estate can be as costly as its on-board cousin.

The full-duplex capability makes SPI very simple and efficient for single master/single slave applications. Some devices use the full-duplex mode to implement an efficient, swift data stream for applications such as digital audio, digital signal processing, or telecommunications channels, but most off-the-shelf chips stick to half-duplex request/response protocols.

SPI is used to talk to a variety of peripherals, such as

- Sensors: temperature, pressure, ADC, touch screens, video game controllers
- Control devices: audio codec's, digital potentiometers, DAC
- Camera lenses: Canon EF lens mount

- Communications: Ethernet, USB, USART, CAN, IEEE 802.15.4, IEEE 802.11, handheld video games
- Memory: flash and EEPROM
- Real-time clocks
- LCD, sometimes even for managing image data
- Any MMC or SD card (including SDIO variant)

For high performance systems, FPGAs sometimes use SPI to interface as a slave to a host, as a master to sensors, or for flash memory used to bootstrap if they are SRAM-based.

JTAG is essentially an application stack for a three-wire SPI flavor, using different signal names: TCK not SCK, TDI not MOSI, TDO not MISO. It defines a state machine (driven by a TMS signal instead of a chip select line), protocol messages, a core command set, the ability to daisy-chain devices in a "scan chain", and how vendors define new commands. The devices in a scan chain are initially treated as a single device, and transitions on TMS update their state machines; once the individual devices are identified, commands may be issued that affect only one device in that scan chain. Different vendors use different JTAG connectors. Bit strings used in JTAG are often long and not multiples of 8 bit words; for example, a boundary scan reports signal state on each of several hundred pins.

SGPIO is essentially another (incompatible) application stack for SPI designed for particular backplane management activities SGPIO uses 3-bit messages.

## 2.4 I²C VS SPI

Let's compare I²C and SPI on several key protocol aspects:

### Bus topology / routing / resources:

I²C needs 2 lines and that's it, while SPI formally defines at least 4 signals and more, if you add slaves. Some unofficial SPI variants only need 3 wires, that is a SCLK, SS and a bi-directional MISO/MOSI line. Still, this implementation would require one SS line per slave. SPI requires additional work, logic and/or pins if a multi-master architecture has to be built on SPI. The only problem I²C when building a system is a limited device address space on 7 bits, overcome with the 10-bits extension.

From this point of view, I²C is a clear winner over SPI in sparing pins, board routing and how easy it is to build an I²C network.

### Throughput / Speed:

If data must be transferred at 'high speed', SPI is clearly the protocol of choice, over I²C. SPI is full-duplex; I²C is not. SPI does not define any speed limit; implementations often go over 10 Mbps. I²C is limited to 1Mbps in Fast Mode+ and to 3.4 Mbps in High Speed Mode – this last one requiring specific I/O buffers, not always easily available.

## Elegance:

It is often said that I²C is much more elegant than SPI, and that this last one is a very 'rough' (if not 'dumb') protocol. Actually, we tend to think the two protocols are equally elegant and comparable on robustness.

I²C is elegant because it offers very advanced features – such as automatic multi-master conflicts handling and built-in addressing management – on a very light infrastructure. It can be very complex, however and somewhat lacks performance.

SPI, on the other hand, is very easy to understand and to implement and offers a great deal of flexibility for extensions and variations. Simplicity is where the elegance of SPI lies. SPI should be considered as a good platform for building custom protocol stacks for communication between ICs. So, according to the engineer's need, using SPI may need more work but offers increased data transfer performance and almost total freedom.

Both SPI and I2C offer good support for communication with low-speed devices, but SPI is better suited to applications in which devices transfer data streams, while I²C is better at multi master 'register access' application.

Used properly, the two protocols offer the same level of robustness and have been equally successful among vendors. EEPROM (Electrically-Erasable Programmable Read-Only Memory), ADC (Analog-to-Digital Converter), DAC (Digital-to-Analog Converter), RTC (Real-time clocks), microcontrollers, sensors, LCD (Liquid Crystal Display) controllers are largely available with I²C, SPI or the 2 interfaces.

# 2.5 CAN Protocol

## 2.5.1CANBusIntroduction:

CAN stands for Controller Area Network i.e. Microcontroller Area Network A BUS is a channel of communication between two or more electronic devices where messages in the form of electrical waveforms can travel in both directions. A CAN BUS is a channel of communication between two or more microcontrollers where messages in the form of electrical waveforms can travel in both direction which obey a highly sophisticated set of rules called the CAN Bus Protocol.[6]

## Importance of CAN

Without a CAN Bus (point to point communication) Wiring harnesses currently in use (complicated, messy & confusing)

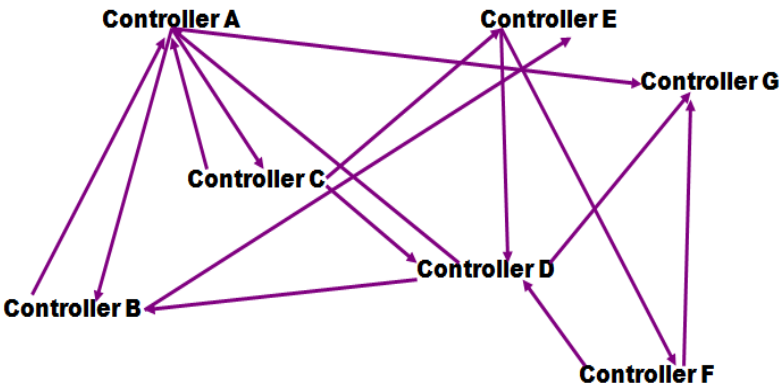



**Fig: 2.5.1: CAN control**

With a CAN Bus: Example of a CAN Bus system with 8 nodes

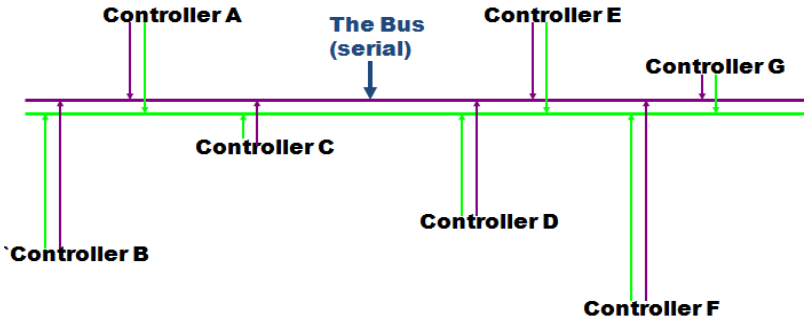



**Fig: 2.5.2: Example of a CAN BUS system with 8 nodes**

CAN was first developed by Robert Bosch GmbH, Germany in 1986 when they were requested to develop a communication system between three ECUs (electronic control units) in vehicles designed by Mercedes. They found that point to point communications was no longer suitable in this situation. The need for a multi-master communication system became imperative. The first CAN silicon was then fabricated in 1987 by Intel.

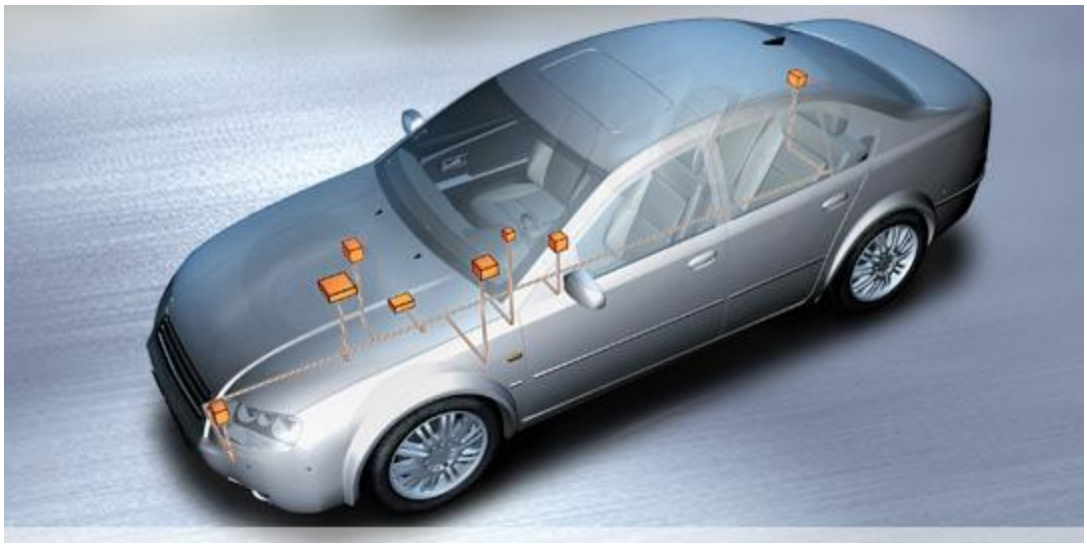


**Fig: 2.5.3: CAN controller used in Auto-Mobiles**

## 2.5.2 Understanding CAN Frame Structure

A CAN Bus system works by sending and receiving messages from node X to node Y. A node is a sensor or actuator connected to a microcontroller and any other device (for ex: ADC) that helps to turn this assembly into a fully functional stand-alone unit.

There are 2 types of Nodes:

- Transmitter: A node originating a message is called a transmitter of that message
- Receiver: A node receiving a message that it is not transmitting itself is called a receiver of a message

Nodes send and receive messages by obeying a highly sophisticated protocol (set of rules) called the CAN Bus protocol.

There are two CAN Bus protocols

- CAN BUS 2.0 A
- CAN BUS 2.0 B

CAN 2.0 A is used with smaller, less complicated systems. CAN 2.0 B is used with larger, more complicated systems. The heart & soul of the CAN Bus protocol deals with a FRAME into which the MESSAGE to be sent or received on the CAN Bus is encoded.[6]

## 2.5.3 CAN Frame Structure

There are only 2 types of Frames

- A Frame that contains a request for data
- A Frame that contains the requested data

The frame that contains a request for a particular data is called the REMOTE TRANSMISSION REQUEST FRAME. The frame that contains the requested data is called the DATA FRAME.

## The DATA FRAME

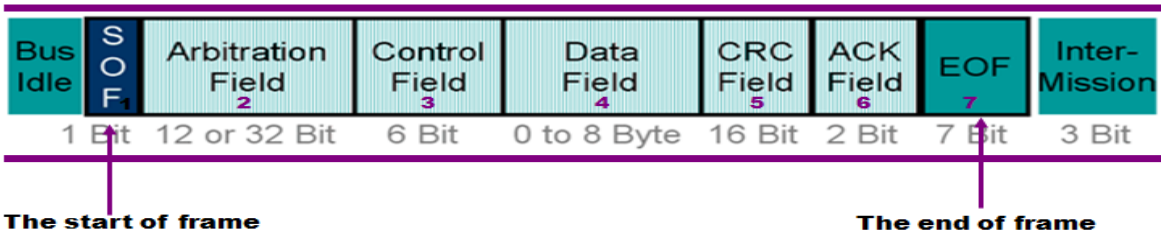



**Fig: 2.5.4: Data frame**

## The RTR Frame: Remote Transmission Request

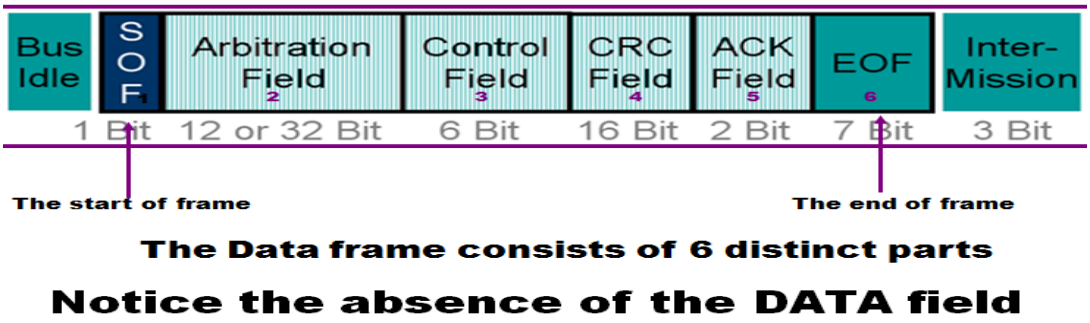



**Fig 2.5.5: RTR frame**

A Frame consists of a large number of bits (0's and 1's). The bits present in a frame are of two types Dominant bit & Recessive bit. Dominant bit represents 0 & Recessive bit represents 1.The Dominant bit has a higher priority than the Recessive bit

## The 7 parts that make up a FRAME

- ❖ SOF – Start of Frame
  - The SOF consists of a single bit
  - This is the MSB i.e. Most Significant bit present in the FRAME
  - This bit is dominant (0) always

❖ Arbitration Field

- The arbitration field is the most important field for the programmer or system designer
- It is also called as the Identifier field as the INDENTITY of the entire frame traveling on the bus is figured out by decoding this field

# Arbitration & Identity

ARBITRATION: is the settling of a dispute between two or more individuals by a third person chosen by them. In CAN, this translates into the settling of a collision between two or more frames sent onto the bus simultaneously (by two or more nodes) with the help of the CAN protocol (third person)

IDENTITY: The unique, individual characteristics by which a person or thing is recognized or known such as Genes, PAN card number, Telephone number etc.

The bits present in the ARBITRATION field can be arranged intelligently by the programmer so that each FRAME is easily distinguished by the unique sequence of bits present in this field. This unique sequence of bits constitutes an IDENTITY. Thus an IDENTITY facilitates the ARBITRATION process with its unique pattern of 0's and 1's.

CAN Bus 2.0 A, has arbitration field that is 11 bits long which can have 2048 different combinations. This means - that with an 11 bit long identifier there can be 2048 unique frames These 2048 unique frames can then be sent onto the bus and each of these frames can then be easily identified by all the nodes present on the bus. How do we get 2048 unique combinations from 11 bits?

Since we follow the Binary format consisting of 0 and 1,

11 bits can give us 2048 combinations as follows

$2^{11} = 2*2*2*2*2*2*2*2*2*2*2 = 2048$

CAN Bus 2.0 B, has an arbitration field that is 29 bits long which can have over 5 million different combinations. This means with a 29 bit long identifier there can be over 5 million unique frames. These 5 million unique frames can then be sent onto the bus and each of these frames can then be easily identified by all the nodes present on the bus. Thus, the bits present in an arbitration field are a carefully selected combination of dominant (0) and recessive (1) bits that have to be decided by the programmer which decides the priority level of the different FRAMES being sent onto the bus and also acts as a unique identifier which helps a node identify any FRAME present on the Bus.

The last bit present in the arbitration field is an RTR bit. In the DATA frame the RTR bit is always dominant (0). In the Remote frame the RTR bit is always recessive (1).[6]

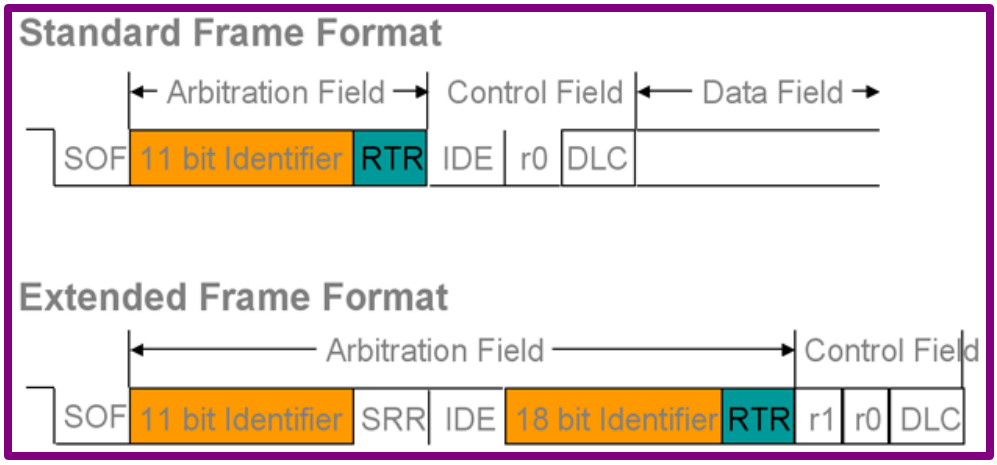



**Fig: 2.5.6: Standard frame format**

## ❖ Control Field

It consists of 6 bits. The first bit is an IDE bit i.e. Identifier Extension bit. This bit is used if CAN Bus 2.0 B with the extended identifier format consisting of 29 bits is desired to be used. The second bit r0 has been reserved for future use and its status is of no use to us. The last 4 bits represent the number of bytes that are present in the next field i.e. the Data field. There can be a minimum of 0 bytes to a maximum of 8 bytes that can be sent as data in a frame

Data Length code in Binary

| Byte | Code in Binary |
|---|---|
| 0 | 0000 |
| 1 | 0001 |
| 2 | 0010 |
| 3 | 0011 |
| 4 | 0100 |
| 5 | 0101 |
| 6 | 0110 |
| 7 | 0111 |
| 8 | 1000 |

## ❖ Data Field

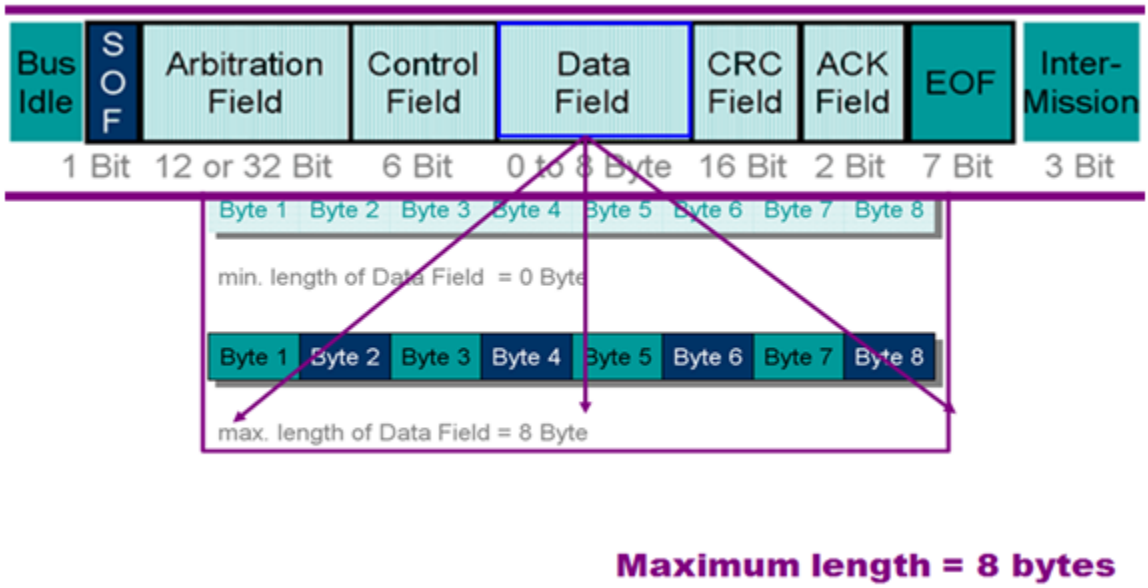



**Fig: 2.5.7: Data field**

## ❖ CRC Field Cyclic Redundancy Check

This field is used to calculate the bit errors present within a CAN frame. The CAN controller does this operation automatically. There are a total of 16 bits present in this field. Of this, the value of the bits in the first 15 fields is automatically calculated. The last bit is called the CRC Delimiter and is always recessive (1).

## ❖ Acknowledgement Field

The acknowledgement field is 2 bits long. It consists of the ACK slot and the ACK Delimiter Acknowledgement Field. The ACK Delimiter as is the case with all Delimiter's found in a frame is always recessive (1). The node that transmits a frame transmits the ACK slot as recessive (1). The node that receives a frame successfully, reports this to the transmitter by sending a dominant bit in the ACK slot (0)

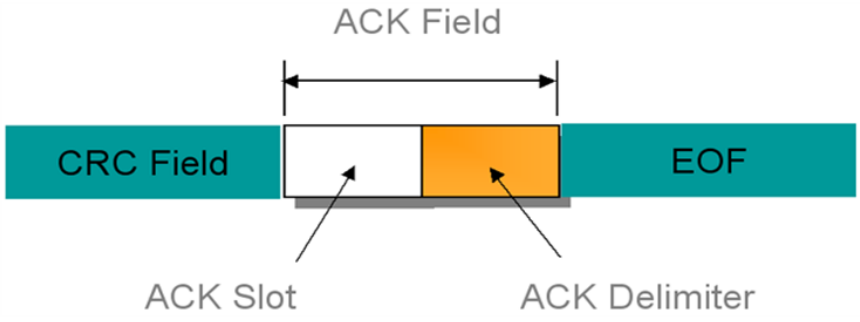



**Fig: 2.5.8: ACK field**

### ❖ End of Frame

The EOF consists of 7 bits. All these bits are recessive (1)

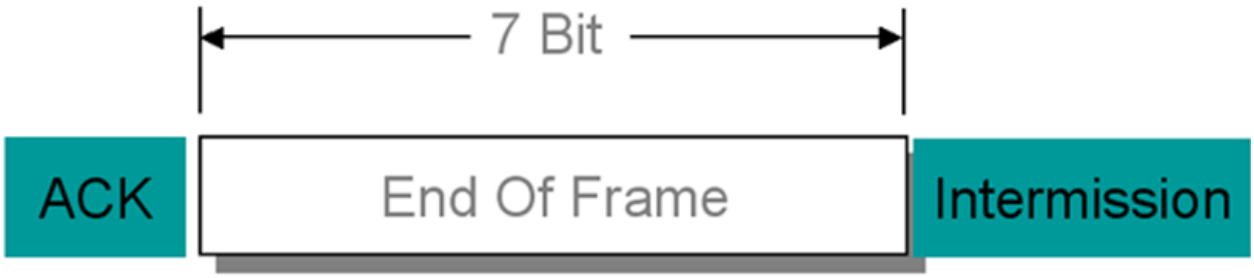



**Fig: 2.5.9: End of frame**

### ❖ Intermission

The Intermission field consists of 3 bits. All these bits are recessive (1), ALWAYS. An intermission field is present in between consecutive DATA frames and RTR frames.

## Bus Idle

When the bus is idle, it is in a recessive state. Thus there will be an infinite series of recessive (1) bits that will be found on the bus in this state.

## Broadcasting a frame onto & Receiving a frame from the Bus.

To broadcast a Frame onto the Bus we need a node that sends a Frame onto the Bus i.e. a Transmitter & we need a node that receives a Frame from the Bus i.e. a Receiver

The transmitter and receiver consists of 3 different blocks

- Local Intelligence
- Frame
- Filter

Local Intelligence consists of the sensor or actuator and the microcontroller functioning intelligently by sending FRAMES onto the bus or receiving FRAMES from the bus as and when instructed to do so.

A Filter can be compared to a Security Guard with a big moustache and an even bigger pot belly standing at the entrance of a large gate. Messages carried by a person can enter this gate ONLY if the security guard is able to verify the identity of the person by checking his ID card.

Similarly in a CAN Bus system a FRAME is allowed to go through the FILTER onto the local intelligence only if the identity of the FRAME present in the Identifier field (also known as the arbitration field) matches the uniquely coded bits present in the FILTER.[6]

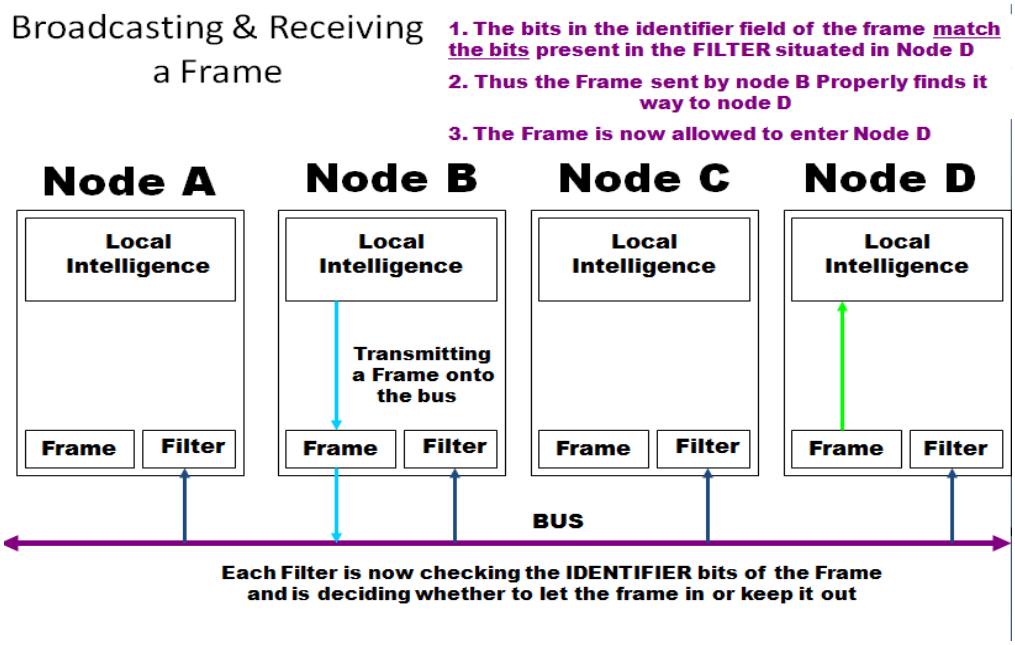



**Fig: 2.5.10: Broadcasting and Receiving frame**

## FRAME Arbitration

For FRAME Arbitration let us assume that three nodes broadcast their respective frames onto the bus simultaneously. Of these 3 frames, finally only 1 frame with the highest priority can travel across the bus. The remaining 2 frames will have to wait for the highest priority frame to reach its destination. After it has reached its destination, the remaining 2 nodes can then begin transmitting their frames on the bus. These 3 nodes will have 3 unique identifiers respectively. These unique identifiers will be present in the arbitration field of the FRAME. In a previous slide it was mentioned that the Dominant bit (0) has a higher priority than the recessive bit (1). This bit priority predetermined by the CAN Protocol will decide which frame wins the arbitration battle being fought on the bus. Let us make up 3 unique identifiers for each node

- Node A Identifier – 11001110001
- Node B Identifier – 11001011011
- Node C Identifier – 11001011001

(Notice that they are 11 bits long)

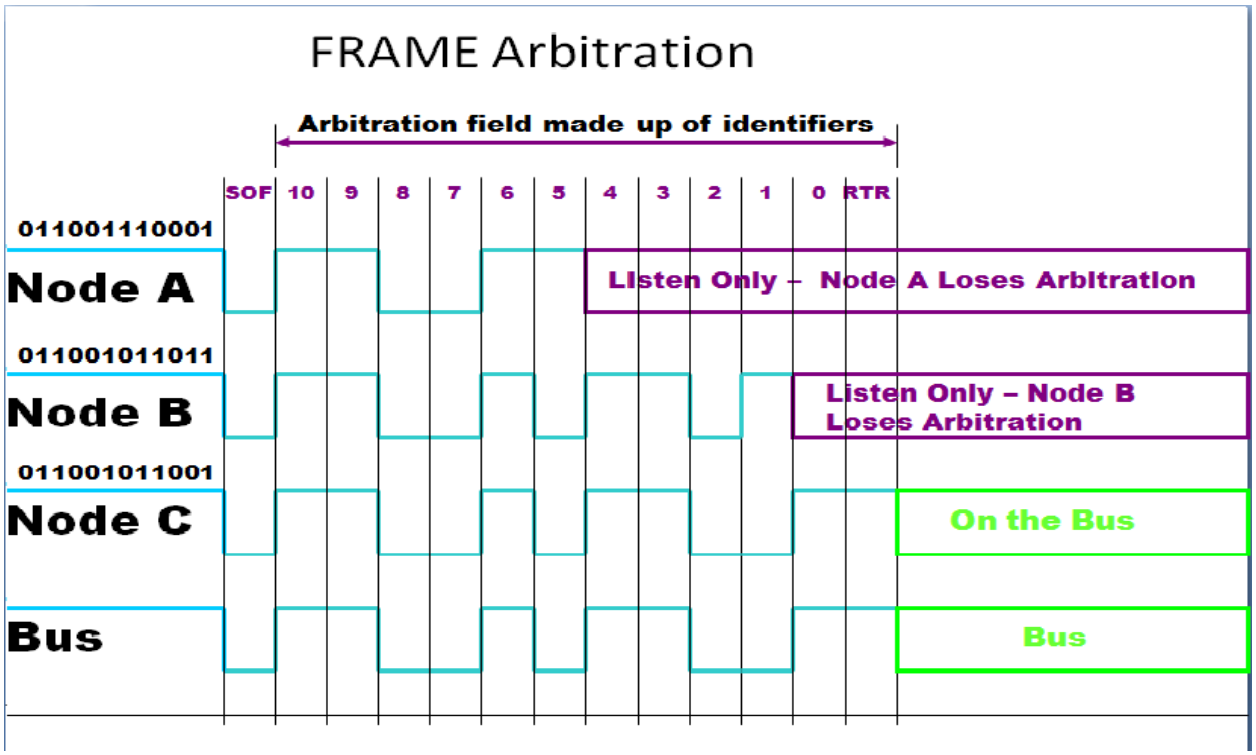



**Fig: 2.5.11: Frame Arbitration**

Node A loses the arbitration battle to Node B & Node C at the 5$^{th}$ bit present in the arbitration field as the 5$^{th}$ bit present in Node A is recessive (1) whereas the 5$^{th}$ bit present in Node B & Node C is dominant (0). As the dominant bit (0) has a higher priority than the recessive bit (1), Node A is forced into LISTEN ONLY mode and Node B & Node C continue the arbitration battle on the bus which is finally won by Node C.

## 2.5.4 CAN Synchronization & Bit Timing

Several different nodes on a bus have their own individual internal clocks running at a frequency that MUST be common across the bus. Let us say that our system has 5 nodes. Each of these 5 nodes has a clock with a frequency of 16 Mhz. Even though each node has a clock that operates at the same 16 Mhz frequency, it is practically impossible that these 5 clocks run in perfect synchronicity with each other. There is bound to be a certain tolerance limit in which the clock operates, for example from 15.99 Mhz to 16.01 Mhz. This deviation of +/- 0.2 Mhz is caused due to practical reasons such as variations in

- temperature
- material properties of the quartz crystal
- manufacturing defects

To take care of these deviations in operating frequencies and to ensure that all the nodes on the bus are in perfect synchronicity with each other we move on to Bit Timing.

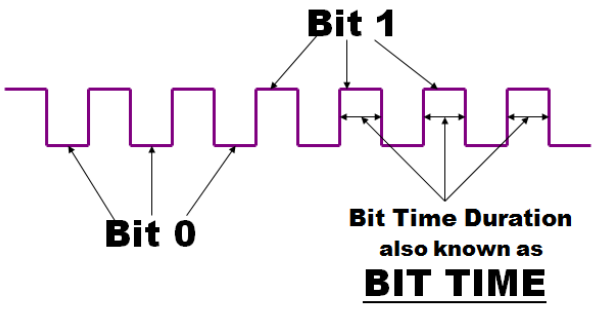



**Fig: 2.5.12: Bit timing**

Bit Time is defined as the time taken by each bit to represent itself. Higher frequencies result in smaller bit time. Lower frequencies result in longer bit time. A Bit Time is divided into 4 Non-overlapping time segments.

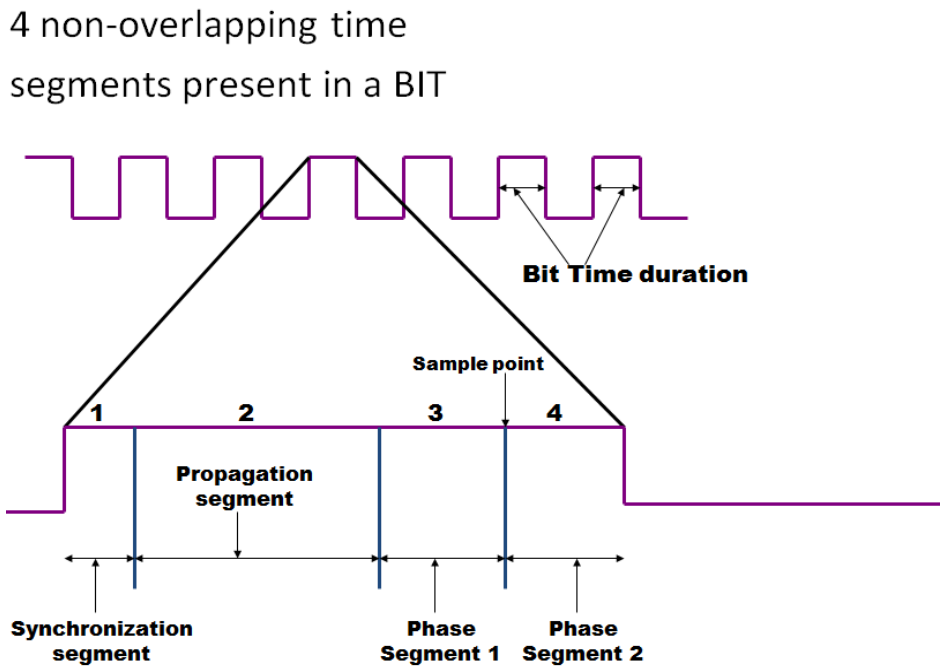



**Fig: 2.5.13: Non-Overlapping time segment**

Each time segment is further divided into a number of smaller units called Time Quanta. A Time Quanta is the smallest unit of time present in a BIT.

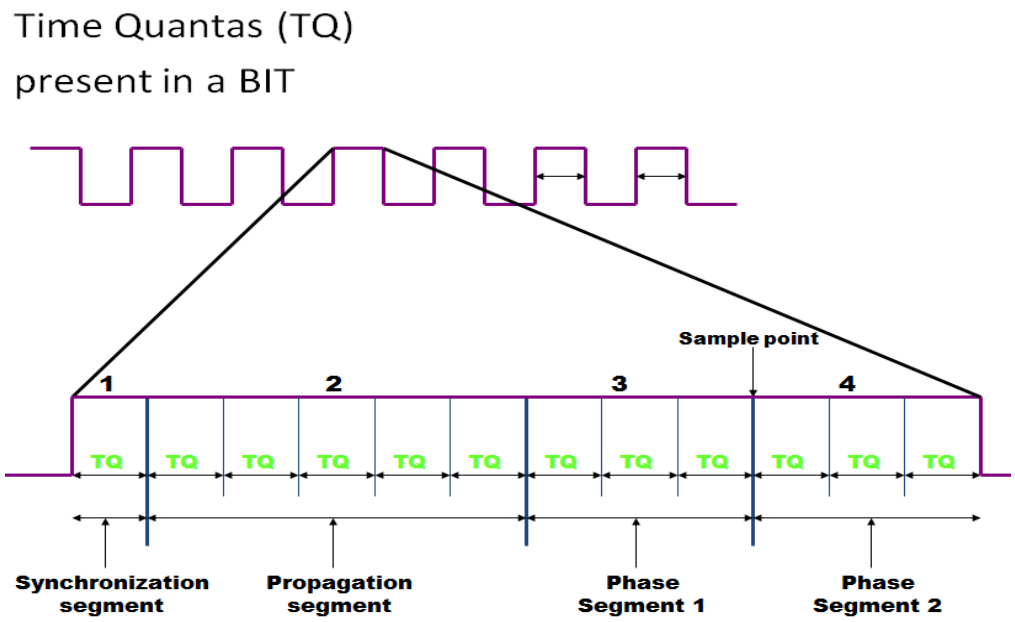



**Fig: 2.5.14: Time Quantas present in a Bit**

The total number of Time Quanta that are present in a BIT must be greater than 8 Time Quanta and less than or equal to 25 Time Quanta.

## Synchronization Segment

This is the first non-overlapping time segment present in a bit. Its duration is 1 Time Quanta always. It is responsible for synchronizing the various CAN nodes on the bus.

## Propagation Segment

This is the second non-overlapping time segment present in a bit. Its duration is programmable to be between 1 to 8 Time Quanta in length. It is responsible for compensating the physical delay times on the nodes and the bus. The physical delay (obstructions in electron flow) is caused by the CAN bus cable, transmitting microcontroller and receiving microcontroller

## Phase Segment 1

This is the third non-overlapping time segment present in a bit. Its duration is programmable to be between 1 to 8 Time Quanta in length. It is responsible for compensating phase errors (mismatches in timing) with the help of an additional component called SJW or Synchronization Jump Width.

## Phase Segment 2

This is the fourth and last of the non-overlapping time segments present in a bit. Its duration is programmable to be between 1 to 8 Time Quanta in length. It is responsible for compensating

phase errors (mismatches in timing) with the help of an additional component called SJW or Synchronization Jump Width.

## Synchronization Jump Width - SJW

The SJW acts like a washer used in mechanical systems. Washers are added or removed from components in a mechanical system to make components fit into each other perfectly. SJW is a software washer. It is implanted into a BIT in such a way that the BIT TIME can compensate for the various shifts in phase (mismatched timing) while receiving a frame from another node. SJW can consist of a minimum of 1 TQ to a maximum of 4 TQ.

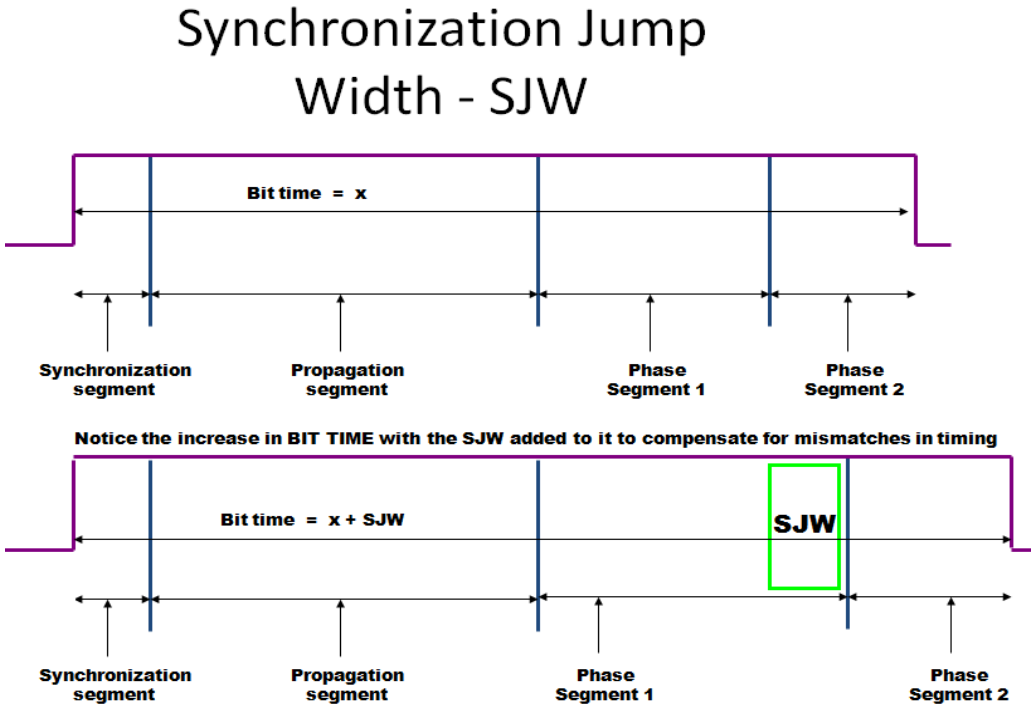



**Fig: 2.5.15: Synchronization Jump Width**

## Sample Point

Its exact location is in-between the end of Phase segment 1 and the beginning of Phase segment 2. This is the point of time at which the BIT is read and interpreted. At this point we know whether the bit is dominant (0) or recessive (1).

Bit Timing on its own does 95% of the work required to synchronize the frames sent by the various nodes present in a bus. But 95% is not good enough. Synchronization between frames sent by the various nodes must reach as close to the 100% mark as possible for proper transmission and reception. This is done with the help of Bit Stuffing.

The CAN Protocol intelligently inserts (stuffs) a bit (s) of opposite polarity into a FRAME, If a continuous series of bits of the same polarity exceed 5 in number this is called bit stuffing.

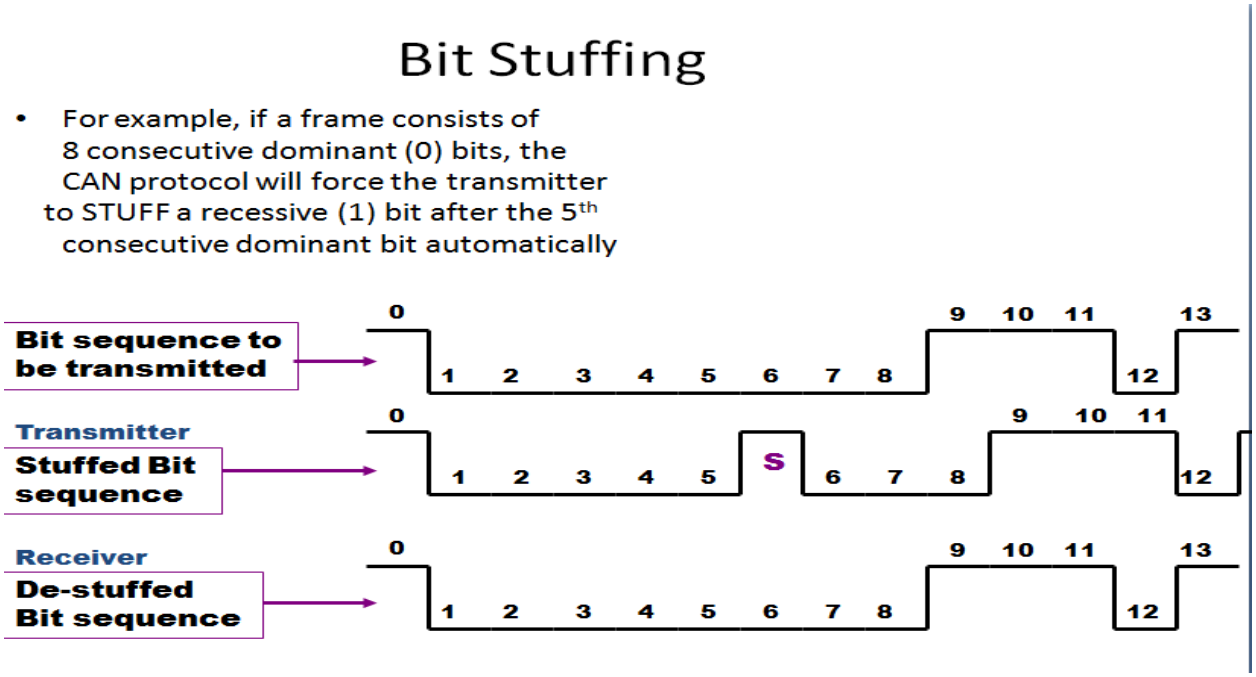



**Fig: 2.5.16: Bit Stuffing**

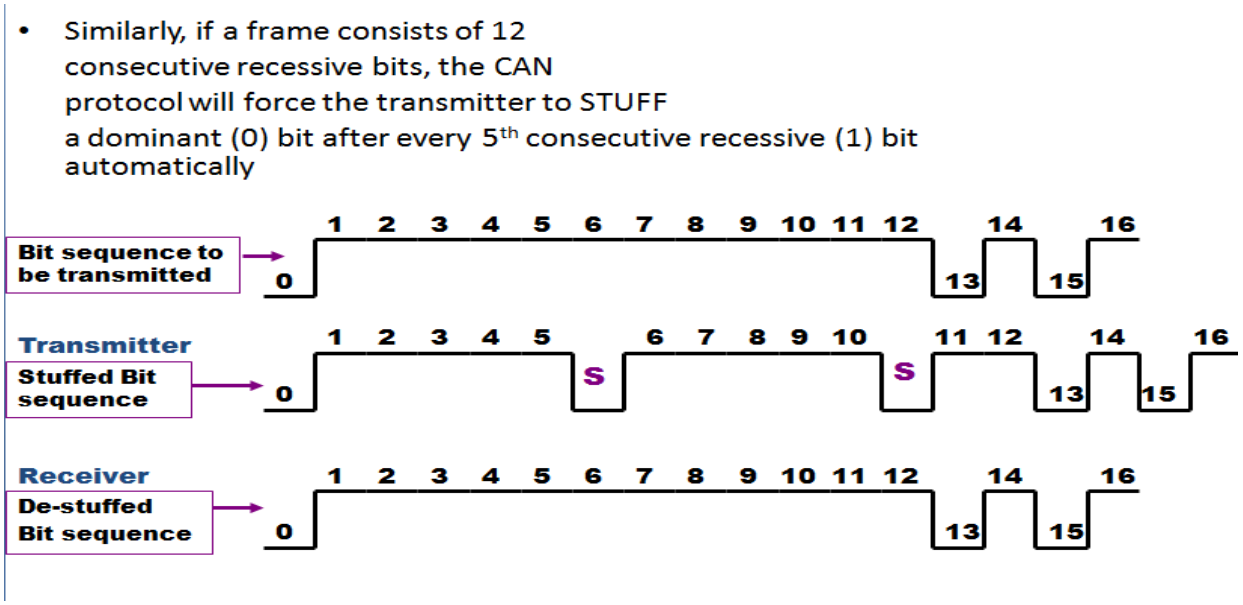
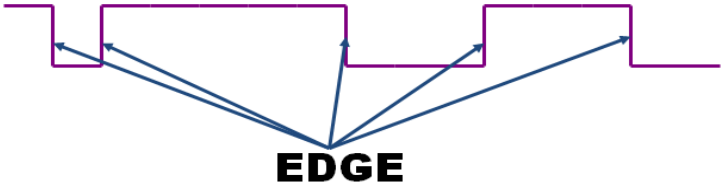



**Fig: 2.5.17: Bit Stuffing**

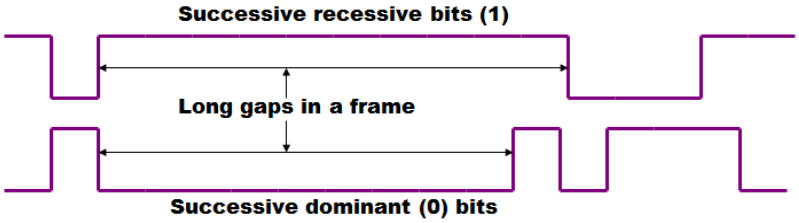
The Node transmitting a FRAME will automatically stuff bits if required. The Node receiving the FRAME will automatically de-stuff the bits thereby returning the FRAME into its original form.

Bit Stuffing is required because CAN follows NZR i.e. Non-Return to Zero coding as against Manchester coding. Remember, we said that a BIT is made up of Time Quanta and each BIT should have a minimum of 8 TQ to a maximum of 24 TQ. We also know that by manipulating TQ's we can negate the phase shifts or mismatches in Timing of FRAMES received from different nodes. All these things mentioned above help to achieve synchronicity between nodes because of which one can transmit and receive frames successfully.

For all this matching & synchronizing business to take place successfully, there is one small requirement without which nothing will work i.e. an edge.




Without an EDGE, there is no reference point and without a reference point there is no way synchronization can take place. So, if a FRAME has a large number of successive bits that are either dominant or recessive, there is a long gap before an edge appears. These long gaps throw the synchronization mechanism into complete confusion. In order to avoid this, an EDGE is introduced after regular intervals.




In CAN an EDGE is introduced, if 5 successive bits of the same polarity are present in a FRAME by inserting a BIT with opposite polarity (BIT STUFFING), to facilitate the smooth functioning of the synchronization process. So now we know how FRAMES are sent and received on the CAN Bus synchronously without sending clock pulses simultaneously with the message as is done with I2C and SPI protocol but by intelligently manipulating the width of each bit (BIT TIME) with the help of small packets of time called TIME QUANTA and with the help of BIT STUFFING.[6]

## 2.5.5 Error Types Handling & Detection

All nodes listen to any message being transmitted onto the bus. The transmitting node too listens to its own message while transmitting it onto the bus.

## Message Validation

The point of time at which a FRAME is taken to be valid is DIFFERENT for the transmitter and receiver of the message.

- Transmitter: A FRAME is valid for a transmitter if there is no error till the end of the FRAME
- Receiver: A FRAME is valid for a receiver if there is no error till the last but one bit of the END OF FRAME.

## CAN Error Detection

- Bit value check by Transmitter
- Bit Stuff checking
- Frame check
- 15 bit Redundancy check
- Acknowledgement check

## CAN Error Types

- Bit error
- Stuff error
- Form error
- CRC error
- Acknowledgement error

## Bit error

A Node that is transmitting a FRAME listens to its own FRAME on the Bus. For example if a node sends a sequence 01100110011 but on the bus it sees 01100100011, a bit error has occurred. An error frame is then sent onto the bus and the FRAME with the Bit error is ignored by all nodes.

## Stuff error

If a FRAME consists of the following bits 0110000000011100110 then according to the bit stuffing rule there should a recessive bit (1) inserted after the $5^{th}$ dominant (0) bit. So the FRAME should be transformed into 01100000100011100110 from 0110000000011100110

BIT STUFFING is done automatically by the CAN Controller but for some reason if it does not happen, a STUFF ERROR occurs. An error frame is then sent onto the bus and the FRAME with the STUFF ERROR is ignored by all nodes

## Form error

The CRC Delimiter, ACK Delimiter and EOF consist of recessive bits (1) ALWAYS. If a dominant bit (0) is detected in any of these, a FORM ERROR has taken place. An Error frame is then sent onto the bus and the FRAME with the FORM ERROR is ignored by all nodes.

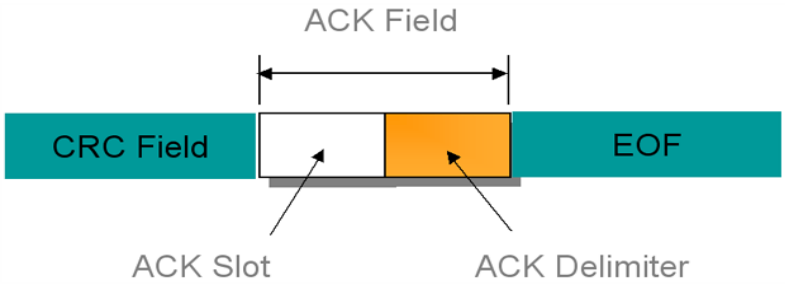



**Fig: 2.5.18: Form Error**

## CRC error (Cyclic Redundancy Check)

The CRC value is 15 bits long. This value is automatically calculated by the transmitting and receiving nodes. If the CRC value of the FRAME calculated by the transmitting node is different from the CRC value of the FRAME calculated by the receiving node a CRC error has occurred. An error frame is then sent onto the bus and the FRAME with the CRC ERROR is ignored by all nodes.

## Acknowledgement error

A transmitting node keeps the value of the ACK slot recessive (1). A receiving node on successful reception of a message changes the value of the ACK slot to dominant (1). If it fails to do this an acknowledgement error has occurred. An error frame is then sent onto the bus and the FRAME with the acknowledgement error is ignored by all nodes.

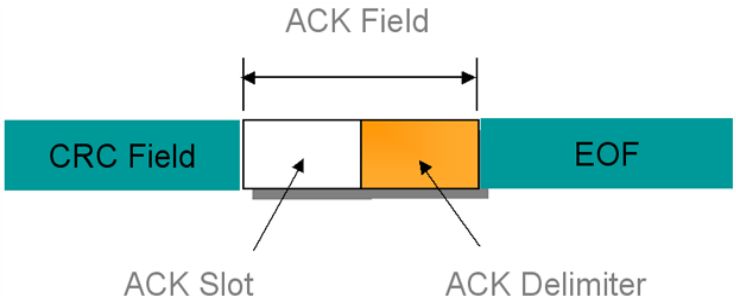



**Fig: 2.5.19: ACK Error**

## Error frame

- It is made up of two fields
  - Error Flag Field
  - Error delimiter Field

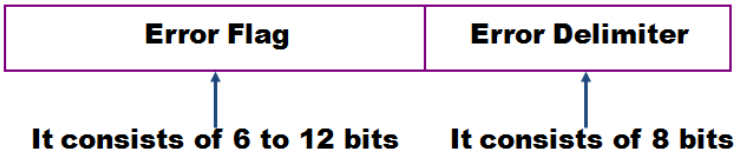



**Fig: 2.5.20: Error Frame**

- An error can occur during transmission
  - A count is kept on these errors by a counter called TEC or Transmit Error Counter
- An error can occur during reception
  - A count is kept on these errors by a counter called REC or Receive Error Counter
- The count value in TEC and REC must be maintained between a certain count limit that is predefined by the CAN Protocol
- Depending upon the count value present in TEC and REC at each node, the Node can be in any of the 3 following states
  - Active Error State
  - Passive Error State
  - Bus Off State

## Active Error State

All the nodes on the bus should be in Active Error State for normal transmission and reception of FRAMES. If any given node has less than 128 counts in it's TEC and REC, it is in Active Error State.

## Passive Error State

If any given node has greater than 127 counts and less than 256 counts in its TEC and REC, it goes into Passive Error State. This is NOT a desirable state for any given NODE to be in. This state is caused due to more than the normal number of errors occurring while transmitting and receiving FRAMES.

Nodes in Passive error state have a lower priority for transmitting frames than nodes in Active error state.

## Bus off State

If any given node has more than 256 counts in its TEC, it goes into in Bus off State. In Bus off state the given node is not allowed to Transmit or Receive any more FRAMES onto the Bus until the reason behind the excessive error generation has been identified and rectified.

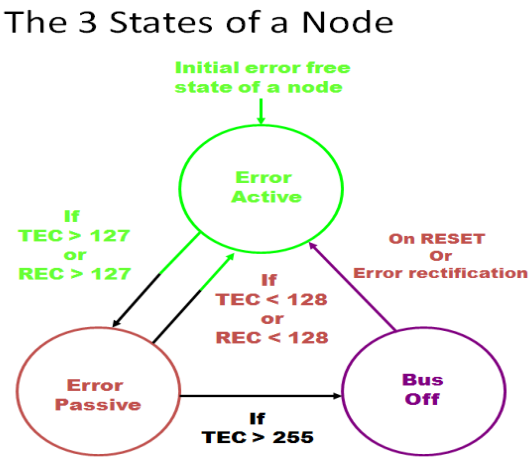



**Fig: 2.5.21: Bus Off State**

- If an error occurs while a node is in Error Active state then
  - The error flag will consist of 6 consecutive dominant (0) bits
  - The error delimiter will consist of 8 consecutive recessive (1) bits


- If an error occurs while a node is in Passive Active state then
  - The error flag will consist of 6 consecutive recessive (0) bits
  - The error delimiter will consist of 8 consecutive recessive (1) bits


## Overload Frame

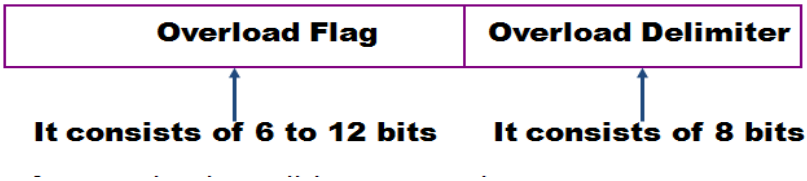



**Fig: 2.5.22: Overload Frame**

It is generated due to 2 conditions:

- If the receiver is not yet ready to receive the next FRAME

- If a node detects a dominant bit during intermission
  - An intermission consists of 3 bits
  - ALL OF THEM ARE RECESSIVE (1) AT ALL TIMES
  - As mentioned earlier, an Intermission frame is present between successive DATA and RTR frames

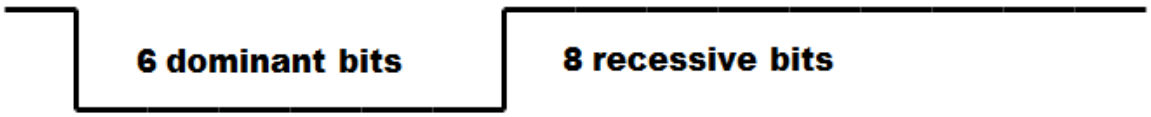
An overload frame is similar to the error frame and is made up of two fields

- Overload Flag Field
- Overload delimiter Field

If an overload condition occurs then

- The overload flag will consist of 6 consecutive dominant (0) bits
- The overload delimiter will consist of 8 consecutive recessive (1) bits

The CAN Protocol generates a maximum of 2 overload frames to delay the next DATA or RTR Frame. If the receiver is still not ready to receive the next frame, an Error Frame is generated




## 2.5.6 Physical Layer of the CAN Bus

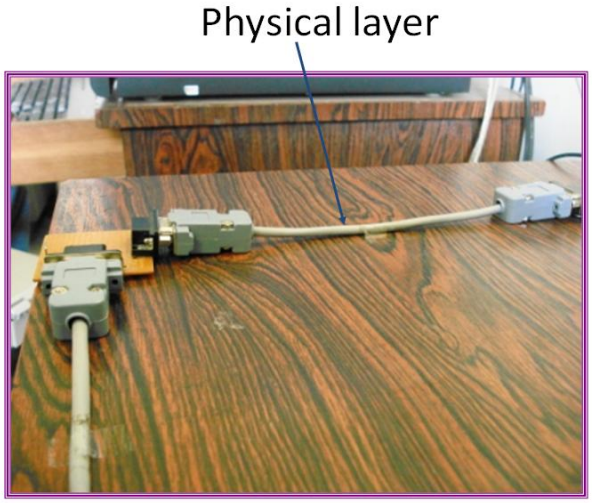



**Fig: 2.5.23: Physical Layer**

Physical Layer consists of a pair of twisted wires that are shielded to reduce electromagnetic interference caused due to noise. One wire carries the CAN High signal. The other wire carries the CAN Low signal. The difference between these signals results in a dominant (0) or recessive bit (1).


**Fig: 2.5.24: Shielded Twisted Pair Wires**

If the difference between the CAN High voltage and CAN Low voltage is less than 0.5 volts, the resulting bit is recessive (1).If the difference between the CAN High and CAN Low voltage is greater than 0.9 volts and less than 3.5 volts, the resulting bit is dominant (0).

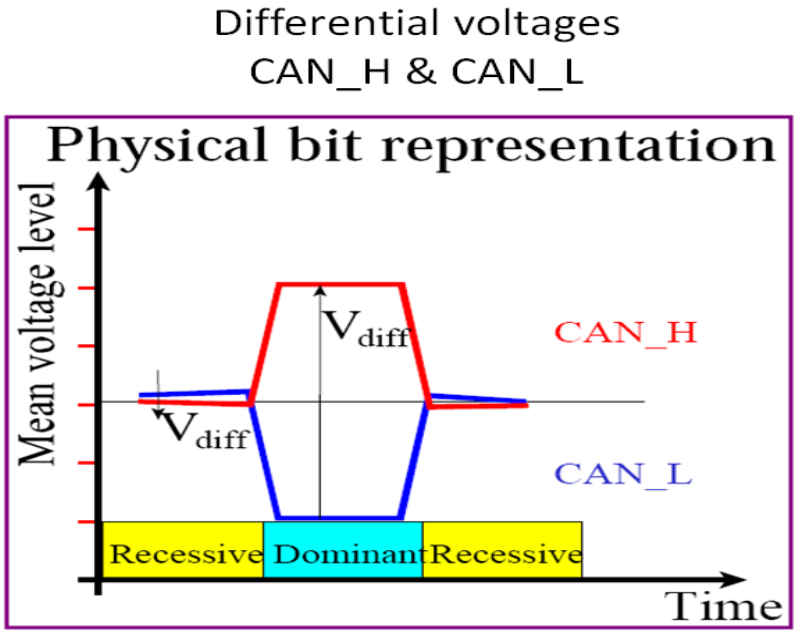



**Fig: 2.5.25: Differential Voltages CAN_H & CAN_L**

CAN Bus Systems are present in harsh, noisy environments where mechanical and electrical disturbances induce unwanted voltages into the wires thereby corrupting the signal and making it

unworthy. To prevent this from happening CAN uses the differential voltage system to figure out whether the BIT is Dominant (0) or Recessive (1).

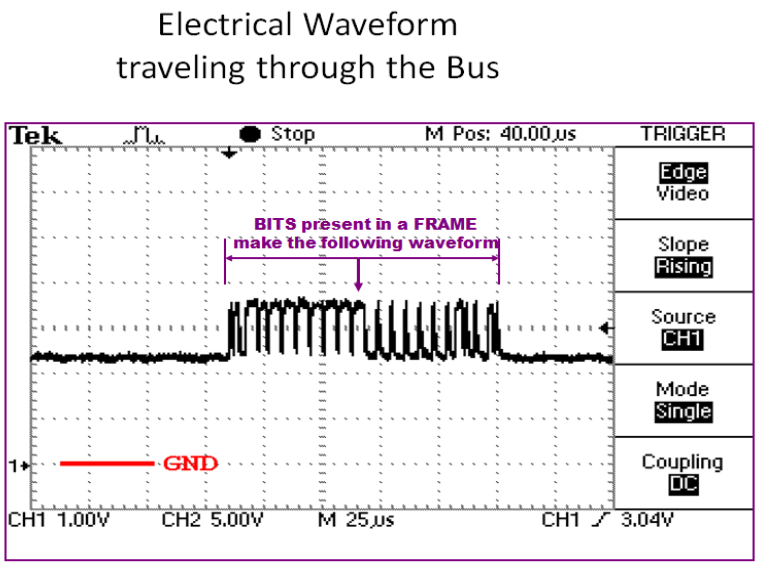



**Fig: 2.5.26: Electrical Waveform Travelling Through the Bus**

# 2.6 RS 485 Protocol

## 2.6.1 Introduction

The RS485 communications standard has been widely adopted for network communications in industrial and scientific applications. It is a very robust system when installed correctly. There are constraints which must be observed during installation to ensure satisfactory network performance. RS-485 is a reliable 2-wire, half-duplex serial communications system. Cables can be up to 4000 feet in length and up to 32 Unit Loads (devices) can be connected to the network. The signals are balanced making it nearly immune to electrical noise. RS-485 Transceiver chips require single +5V supply making them simple to integrate into your project. RS-485 is ideal for applications ranging from simple communication experiments to industrial control systems.[2]

## 2.6.2 Cabling

## Cabling Specifications

RS485 communication cable is constant impedance, shielded, twisted pair cable. Many cable manufacturers supply cable meeting the RS485 standard. The characteristic impedance is generally close to 120 Ohms although other impedances are sometimes available. The lower impedance limit is set by the drive capability of the RS485 chips. The driver chips used by

Sphere are sourced from Maxim, National Semiconductor and Texas Instruments (SN75176/DS75176) and can drive impedances down to approximately 50ohms

Other cabling media such as telephone cable, coaxial cable and non-constant impedance cables such as shielded wire, multi-core wires, building wire and wet string will prove problematical and not provide satisfactory performance.

A typical cable specification is:

| | |
|---|---|
| 1. Cable Code | DCK4702 |
| 2. Cable Description | 2 pair 7/0.2tcw (24AWG) polyethylene insulated aluminum polyester tape screened including a 7/0.2 TCW drain. TCW braided PVC sheathed. EIS RS485 Data Cable. Low voltage, not for mains connection. |
| 3. Construction | |
| 3.1 Conductor | 7 strands of 0.2mm tinned annealed copper to AS1125, drawn from Class 11A "Electrolytic tough pitch copper" to AS 1574.<br>Max DC conductor resistance at 20C 89.0 □ /km |
| 3.2 Insulation | Colored polyethylene to AS1049<br>Nominal Diameter 1.7mm<br>Nominal Wall 0.5mm |
| 3.3 Lay Up | 2 pairs twisted<br>PAIR 1 Black + White<br>PAIR 2 Red + Green |
| 3.4 Screen | Aluminum/Polyester tape over layup plus 7/0.2 TCW Drain |
| 3.4 Braid | Tinned annealed copper wire braid to 80% cover |
| 3.5 Sheath | Colored 4V75 PVC to AS3191<br>Nominal diameter 7.0mm |

| | |
|---|---|
| | Nominal wall 0.76mm |
| 3.6 Typical Electrical Properties | Nominal impedance 120□<br>Nominal capacitance 42pF/m cond to cond<br>Velocity of Propagation 66% |

This cable is manufactured by Tycab and distributed by AIM.

## Topology

RS485 allows multiple network nodes to be connected to a single cable. The cable must be installed to pass close by each node. Stubs (cables joining the node to the cable) or stars (multiple cable segments brought back to a single point) are not permitted.

If stubs or stars are required then a repeater must be installed to drive the stub or star segment.

## Cable Length

The maximum length of a single cable run is 1200 meters.

## 2.6.3 Termination

The cable must be terminated at each end with a terminating resistor equal to the characteristic impedance of the cable. This is to prevent reflections which would disrupt communications. A cable with 120 Ohm characteristic impedance must have a 120 Ohm resistor at each end. A power rating of ¼ Watt is adequate. The terminating resistors should be carbon composition or carbon or metal film. Wire-wound resistors have substantial inductance and should not be used.

The characteristic impedance of the cable is published by the cable manufacturer as part of the cable specifications.

## 2.6.4 Repeaters

Sphere supplies repeater types for different applications.
The standard repeater is used to connect to a stub or to extend cable length
An optically-isolated repeater is used to maintain electrical isolation between cable runs in separate buildings. Note: Electrical isolation is required where separate MEN grounds are installed but are desirable for any data link between buildings.

## 2.6.5 Cable Isolation

The communication cable must not run in cable trays carrying power wiring or in close proximity to power wiring. Current surges in power wiring due to high equipment starting currents or to faults can disrupt communications and may cause damage to Sphere equipment.

## 2.6.6 Ground and Earth Connections, Safety & Voltage Reference

The grounding and earthing connections in RS485 provide two separate functions. The first is related to safety and the second to establish a reference voltage.

The RS485 cable screen must be bonded to the protective earth system of a building at one point only.
The cable screen must be electrically continuous throughout the entire cable run.
The screen of the RS485 cable establishes a ground reference voltage for the RS485 signal conductors. For this reason the cable shield must be connected to the ground reference for each node on the network.
It is not acceptable practice to tie the node ground reference to the building protective earth as this will introduce electrical noise into the system and may lead to equipment damage in the event of electrical fault currents.[1]

## 2.6.7 Installation throughout Multiple Buildings

There are two separate installation procedures depending on the type of electrical earthing system.

### MEN System at one building

In this case the installation may be made as though it exists in a single building. The cable shield will be continuous throughout the installation.

### MEN System in each building

In this case there must be electrical isolation between buildings with different MEN systems. Each building is wired as a separate and complete installation with the cable shield tied to the building protective earth at one point.

Optically isolated repeaters or optical fiber must be used to connect the communications network between buildings.

## 2.6.8 Wiring Details

Figure 1 shows how the RS485 network should be wired between various network nodes. The essential features to note are:

1. All the cable shields are tied together and are tied to a Ground connection (GD or Shield) on EACH node. Without the cable shield connected there will be no ground reference voltage established for the RS485 drivers and receivers. Reliable communications will not be achieved.
2. The cable shield is connected to a building protective earth at ONE POINT. Any point along the shielded cable can be used to attach this protective earth. This earthing is required to ensure that the network cabling cannot float to dangerous voltages.

3. The DP11, or in fact any other node, can be connected at any point along the cable. It does not need to be at one end.
4. Termination resistors must be connected at each end of the cable. The nominal resistance is 120□. These resistors make it appear that the cable continues indefinitely and hence eliminate reflections from the end of the cable. These resistors can be ¼ or ½ Watt resistors.
5. The polarity of the cable is important. For ease of installation and maintenance it is recommended that a consistent color code be adopted throughout the network. An acceptable scheme may be Blue for COM- and White for COM+.
6. Beyond any repeater the cable must be 2-pair with one pair connecting the COM terminals and the second pair connecting the TRX terminals. In figure 2 the 2-pair cable as shown as fatter lines[1]

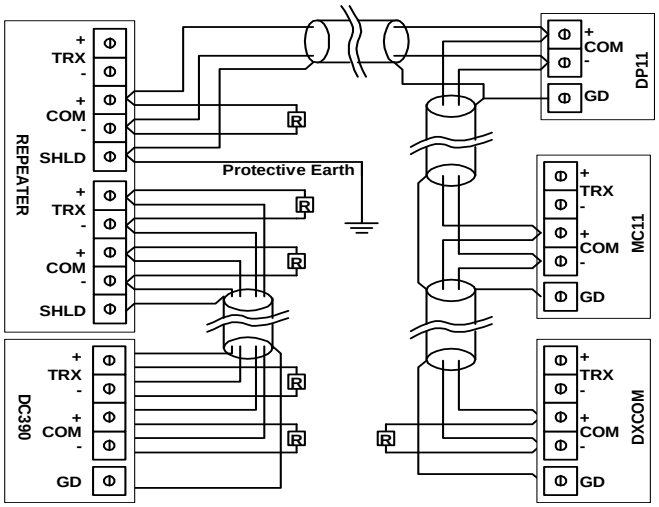



**Fig: 2.6.1: Wiring Details for RS485 Network**

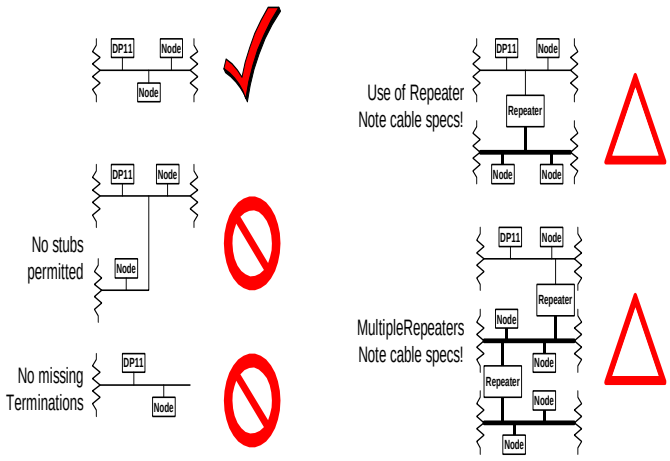



**Fig: 2.6.2: RS 485 Cabling Topologies**

## 2.6.9 RS-232 vs. RS-485

RS-232 is a point-point system with a maximum specified range of 75 feet. Since there is no addressing, only 2 devices can communicate. Speeds of over 115 K Baud is possible. High speeds are often problematic at longer distances but low speeds are quite reliable, even at several times the specified maximum range. Signaling is done using a minimum of 3 Lines (TXD, RXD, GND) and uses ±12VDC levels (0-5 VDC for TTL). RS-232 is not suitable in electrically noisy environments, even with shielded cables. RS-485 is a Point-to-Multipoint (also referred to as multi drop) system that can address up to 32 nodes over a distance of up to 4000 feet.

Data rates of up to 10 M bit/sec. are possible. Only 2 wires are required and inexpensive unshielded twisted-pair can be used. Regardless of which method you use, data is sent by varying the voltage levels on a wire. "TTL RS-232" uses 5 VDC (2.4V to 6.0VDC is valid) to represent a logic 1, and 0V (up to 0.4V is valid) to represent a logic 0. By comparison, True RS-232 uses +12V for logic 0 and -12V for logic 1. RS-485 uses a different approach to indicating logic levels. Instead of measuring voltage levels on one line with reference to ground, it uses the voltage *difference* between 2 lines called "A" and "B". The line that is more positive is assumed to be line B and then less positive is line A. The signal or logic level is the result of subtracting line A from line B. Put another way, if Line B is greater then Line A then logic level is 1; if Line A is greater then Line B, the logic level is 0. If an RS-485 receiver sees line A is 0.2V more positive then line B it is logic 1. If line B is 0.2V more positive then line A it is logic 0. If there is less then 0.2V difference received the logic is undetermined.

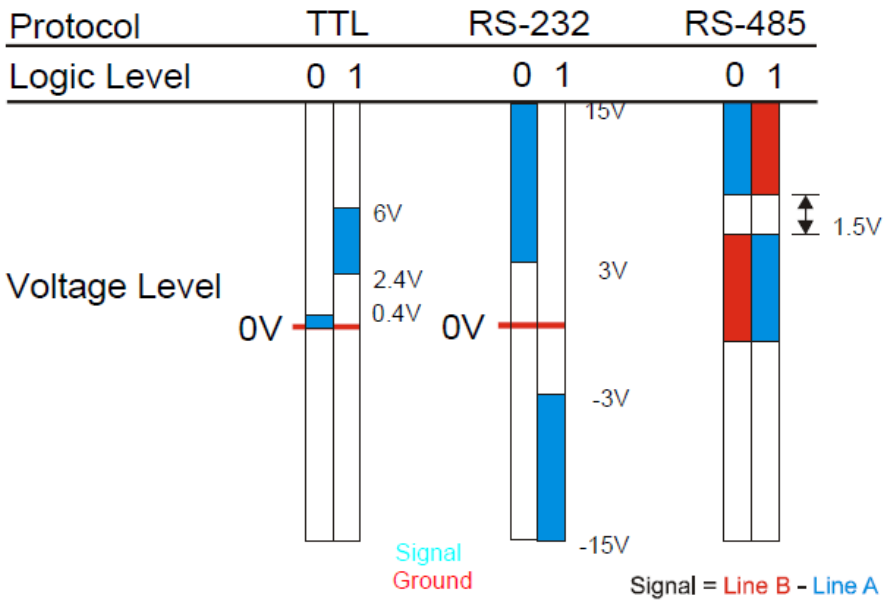
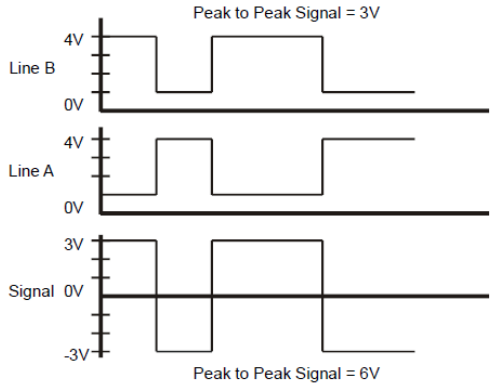



**Fig: 2.6.3: RS-232 VS RS-485**

An RS-485 driver chip only needs to drive a minimum voltage difference of 1.5V. This reduces the requirement for a stable ground, lowers current consumption and reduces component cost.[2]

## Noise

The term "noise" refers to electrical interference that is caused by induced voltages. These voltages result when a varying magnetic field passes through a conductor such as the wire you are using to communicate over. This results in unexpected changes in the voltage levels on the serial lines, and can alter or obliterate your data. TTL RS-232 communications operate over very short distances. The shorter the conductor the less chance there is for induced voltages. True RS-232 deals with moderate distances and environments where induced voltages aren't as likely. RS-485 operates on cable lengths as long as 4000 feet and in environments where induced voltages are common. Noise immunity in the RS-485 protocol is the result of the induced voltage levels being subtracted away. Any noise will be present on both signal lines (A and B). When the receiver chip measures the signals it calculates the difference between the lines -not their absolute voltage levels. This is why RS-485 is referred to as a balanced system. Another inherent advantage is that as each signal line has a peak voltage of 3 Volts, that when you subtract the 2 lines, the resultant signal has a peak-peak voltage of 6 Volts. The larger the voltage used to represent a bit, the less likely it is to be damaged or destroyed by noise.




## Balanced vs. Unbalanced

An unbalanced transmission medium is one in which an induced voltage can affect one line more than another. RS-232 is unbalanced as it references ground and an induced voltage will alter the signal line, but not ground. A balanced transmission medium is one where any induced voltages affect all signal carrying lines in the same way. With RS-485, the information being sent is detected by the difference in voltage on the signal lines; if an induced voltage affects both lines equally, then the net voltage difference between the 2 lines remains unchanged and the information remains intact.

## Protocol or Standard

RS-485 is the common name for the TIA/EIA-485 standard. This a guideline describing the voltage levels for signals, cable impedance, load capacity and other details. It does not specify how to communicate with your equipment; nor does it tell you how to address your nodes, acknowledge receipt or rejection of data, or perform error checking. Since EIA-485 is basically a specification for driver, receiver and transceiver chips, the manufacturer of the equipment needs

to specify cabling, grounding, termination and connectors. RS-485 is a Standard an example of a protocol that can be used over RS-485 is S.N.A.P (described in the examples section).

## Duplex

RS-485 is a half-duplex system, meaning that only one node can transmit at once. A simple example would be a system with two nodes; where Node 1 is transmitting while Node 2 is receiving. Similarly, if Node 2 needs to transmit, Node 1 must switch to receive. In a system where there are multiple nodes, preventing conflicts requires some kind of protocol. The easiest protocol to implement for half duplex communications is called Master/Slave. With this arrangement, one node is designated as the Master and tells all the other Slave nodes when they can transmit. A more complicated version of this is “Token Ring”, where the master is defined as the node that has the token. The Master can transfer the token to another node when needed. Note that both options require that each node have a unique address.[2]

# CHAPTER 3
# Integrated circuit details

## 3.1 MOCROCONTROLLER 89V51RD2

### General description:

The P89V51RD2 is an 80C51 microcontroller with 64 KB Flash and 1024 bytes of data RAM. A key feature of the P89V51RD2 is its X2 mode option. The design engineer can choose to run the application with the conventional 80C51 clock rate (12 clocks per machine cycle) or select the X2 mode (6 clocks per machine cycle) to achieve twice the throughput at the same clock frequency. Another way to benefit from this feature is to keep the same performance by reducing the clock frequency by half, thus dramatically reducing the EMI.

The Flash program memory supports both parallel programming and in serial In-System Programming (ISP). Parallel programming mode offers gang-programming at high speed, reducing programming costs and time to market. ISP allows a device to be reprogrammed in the end product under software control. The capability to field/update the application firmware makes a wide range of applications possible.

The P89V51RD2 is also In-Application Programmable (IAP), allowing the Flash program memory to be configured even while the application is running.

### Capabilities of 8051:

- Internal ROM & RAM
- I/O ports with programmable pins
- Timers & Counters
- Serial data communication

### Features:

- 80C51 Central Processing Unit
- 5 V Operating voltage from 0 to 40 MHz
- 64 KB of on-chip Flash program memory with ISP (In-System Programming) and
- IAP (In-Application Programming)
- Supports 12-clock (default) or 6-clock mode selection via software or ISP
- SPI (Serial Peripheral Interface) and enhanced UART
- PCA (Programmable Counter Array) with PWM and Capture/Compare functions
- Four 8-bit I/O ports with three high-current Port 1 pins (16 mA each)
- Three 16-bit timers/counters
- Programmable Watchdog timer (WDT)
- Eight interrupt sources with four priority levels
- Second DPTR register

- Low EMI mode (ALE inhibit)
- TTL- and CMOS-compatible logic levels

# Block diagram:

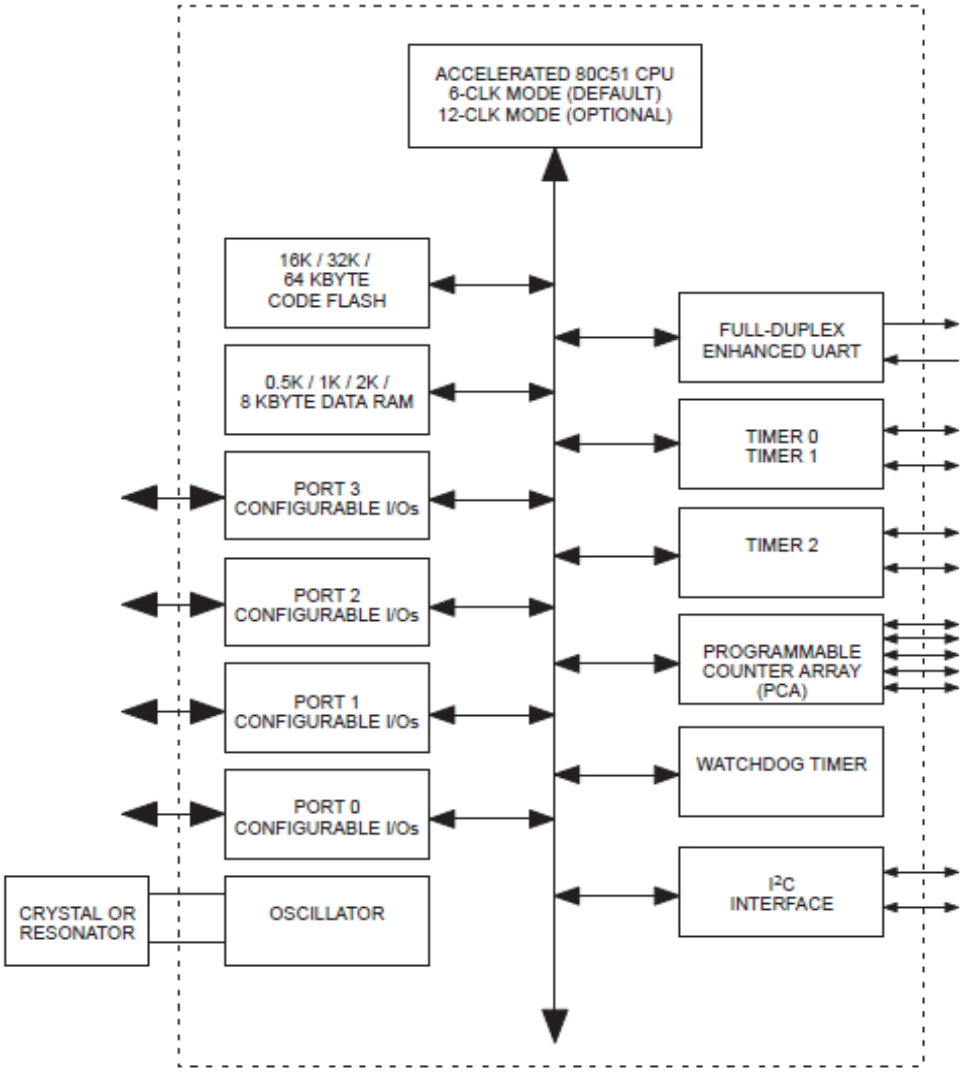



**Fig: 3.1.1 89V51RD2 Block Diagram**

# Hardware description:

## Pin diagram:

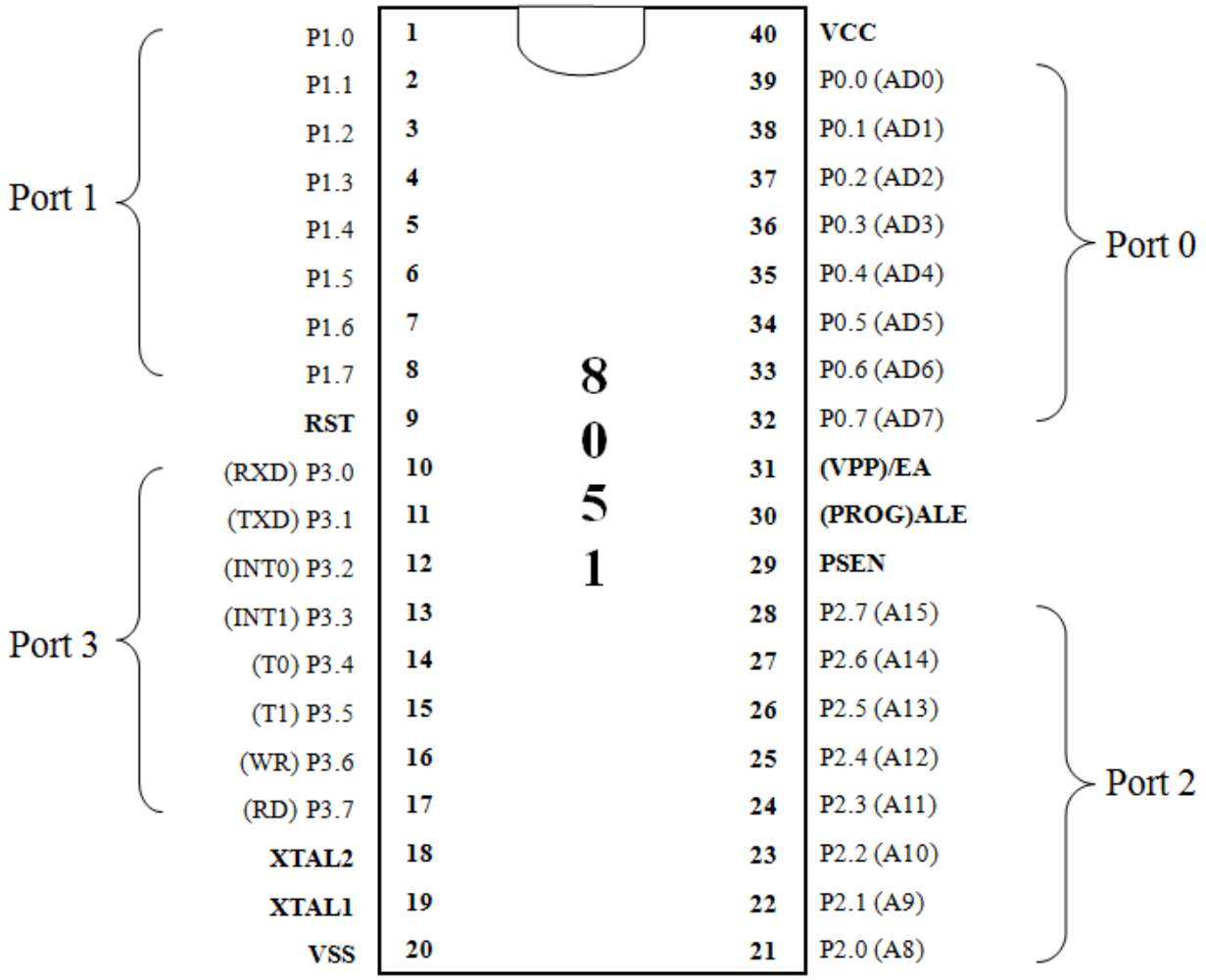



**Fig: 3.1.2 89V51RD2 Pin Diagram**

## Pin description:

| PINS | TYPE | DESCRIPTION |
|---|---|---|
| P0.0 to P0.7 | I/O | **Port 0:** Port 0 is an 8-bit open drain bi-directional I/O port. Port 0 pins that have '1's written to them float, and in this state can be used as high-impedance inputs. Port 0 is also the multiplexed low-order address and data bus during accesses to external code and data memory. In this application, it uses strong internal pull-ups when transitioning to '1's. Port 0 also receives the code bytes during the external host mode programming, and outputs the code bytes during the external host mode verification. External pull-ups are required during program verification or as a general purpose I/O port. |
| P1.0 to P1.7 | I/O with internal pull-up | **Port 1:** Port 1 is an 8-bit bi-directional I/O port with internal pull-ups. The Port 1 pins are pulled high by the internal pull-ups when '1's are written to them and can be used as inputs in this state. As inputs, Port 1 pins that are externally pulled LOW will source current (IIL) because of the internal pull-ups. P1.5, P1.6, P1.7 have high current drive of 16 mA. Port 1 also receives the low-order address bytes during the external host mode programming and verification. |
| P1.0 | I/O | **T2:** External count input to Timer/Counter 2 or Clock-out from Timer/Counter 2 |
| P1.1 | I | **T2EX**: Timer/Counter 2 capture/reload trigger and direction control |
| P1.2 | I | **ECI**: External clock input. This signal is the external clock input for the PCA |
| P1.3 | I/O | **CEX0**: Capture/compare external I/O for PCA Module 0. Each capture/compare module connects to a Port 1 pin for external I/O. When not used by the PCA, this pin can handle standard I/O. |

| | | |
|---|---|---|
| P1.4 | I/O | **SS**: Slave port select input for SPI<br>**CEX1**: Capture/compare external I/O for PCA Module 1 |
| P1.5 | I/O | **MOSI**: Master Output Slave Input for SPI<br>**CEX2**: Capture/compare external I/O for PCA Module 2 |
| P1.6 | I/O | **MISO**: Master Input Slave Output for SPI<br>**CEX3**: Capture/compare external I/O for PCA Module 3 |
| P1.7 | I/O | **SCK**: Master Output Slave Input for SPI<br>**CEX4**: Capture/compare external I/O for PCA Module 4 |
| P2.0 to P2.7 | I/O with internal pull-up | **Port 2**: Port 2 is an 8-bit bi-directional I/O port with internal pull-ups. Port 2 pins are pulled HIGH by the internal pull-ups when '1's are written to them and can be used as inputs in this state. As inputs, Port 2 pins that are externally pulled LOW will source current (IIL) because of the internal pull-ups. Port 2 sends the high-order address byte during fetches from external program memory and during accesses to external Data Memory that use 16-bit address (MOVX@DPTR). In this application, it uses strong internal pull-ups when transitioning to '1's. Port 2 also receives some control signals and a partial of high-order address bits during the external host mode programming and verification |
| P3.0 to P3.7 | I/O with internal pull-up | **Port 3**: Port 3 is an 8-bit bidirectional I/O port with internal pull-ups. Port 3 pins are pulled HIGH by the internal pull-ups when '1's are written to them and can be used as inputs in this state. As inputs, Port 3 pins that are externally pulled LOW will source current (IIL) because of the internal pull-ups. Port 3 also receives some control signals and a partial of high-order address bits during the external host mode programming and verification. |
| P3.0 | I | **RXD**: serial input port |
| P3.1 | O | **TXD**: serial output port |

| | | |
|---|---|---|
| P3.2 | I | **INT0**: external interrupt 0 input |
| P3.3 | I | **INT1**: external interrupt 1 input |
| P3.4 | I | **T0**: external count input to Timer/Counter 0 |
| P3.5 | I | **T1**: external count input to Timer/Counter 1 |
| P3.6 | O | **WR**: external data memory write strobe |
| P3.7 | O | **RD**: external data memory read strobe |
| PSEN | I/O | **Program Store Enable**: PSEN is the read strobe for external program memory. When the device is executing from internal program memory, PSEN is inactive (HIGH). When the device is executing code from external program memory, PSEN is activated twice each machine cycle, except that two PSEN activations are skipped during each access to external data memory. A forced HIGH-to-LOW input transition on the PSEN pin while the RST input is continually held HIGH for more than 10 machine cycles will cause the device to enter external host mode programming. |
| RST | I | **Reset**: While the oscillator is running, a HIGH logic state on this pin for two machine cycles will reset the device. If the PSEN pin is driven by a HIGH-to-LOW input transition while the RST input pin is held HIGH, the device will enter the external host mode; otherwise the device will enter the normal operation mode. |
| EA | I | **External Access Enable**: EA must be connected to VSS in order to enable the device to fetch code from the external program memory. EA must be strapped to VDD for internal program execution. However, Security lock level 4 will disable EA, and program execution is only possible from |

| | | |
|---|---|---|
| | | internal program memory. The EA pin can tolerate a high voltage of 12 V. |
| ALE/ PROG | I/O | **Address Latch Enable:** ALE is the output signal for latching the low byte of the address during an access to external memory. This pin is also the programming pulse input (PROG) for flash programming. Normally the ALE is emitted at a constant rate of 1□ 6 the crystal frequency and can be used for external timing and clocking. One ALE pulse is skipped during each access to external data memory. However, if AO is set to '1', ALE is disabled. |
| XTAL1 | I | **Crystal 1**: Input to the inverting oscillator amplifier and input to the internal clock generator circuits. |
| XTAL2 | O | **Crystal 2:** Output from the inverting oscillator amplifier. |
| VDD | I | **Power supply** |
| VSS | I | **Ground** |

**Table: 3.1.1 Pin Functions**

# Internal data memory

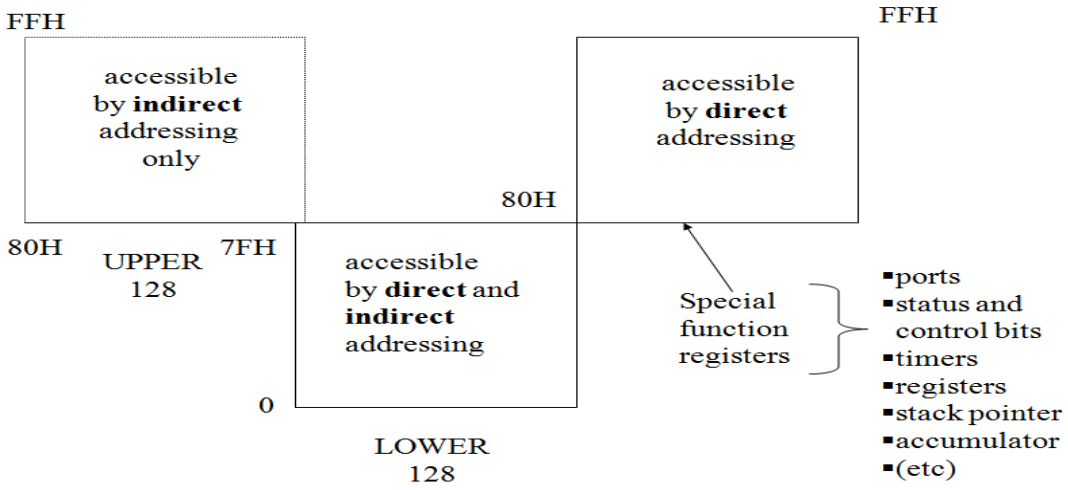



**Fig: 3.1.3 Internal Data Memory**

# The lower 128 bytes of Internal RAM

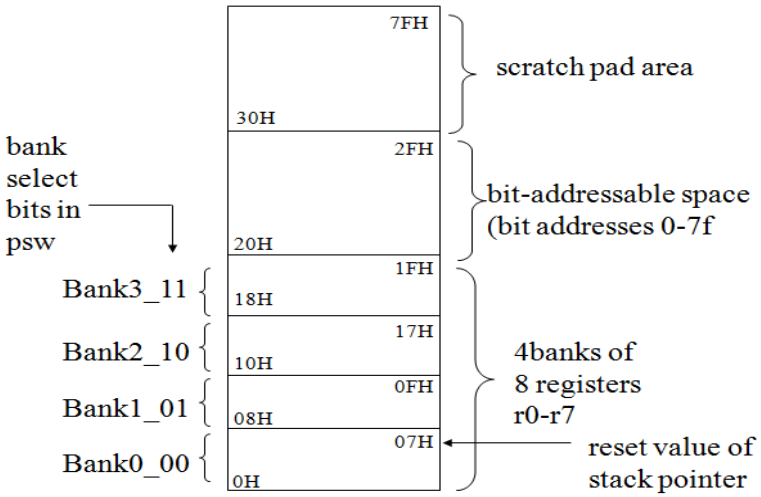



**Fig: 3.1.4 RAM**

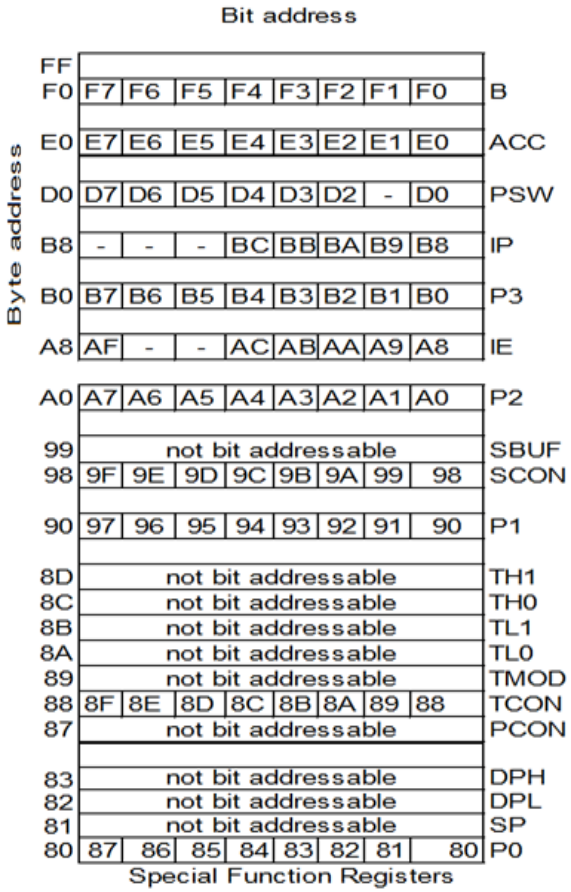



**Fig: 3.1.5 SFR**

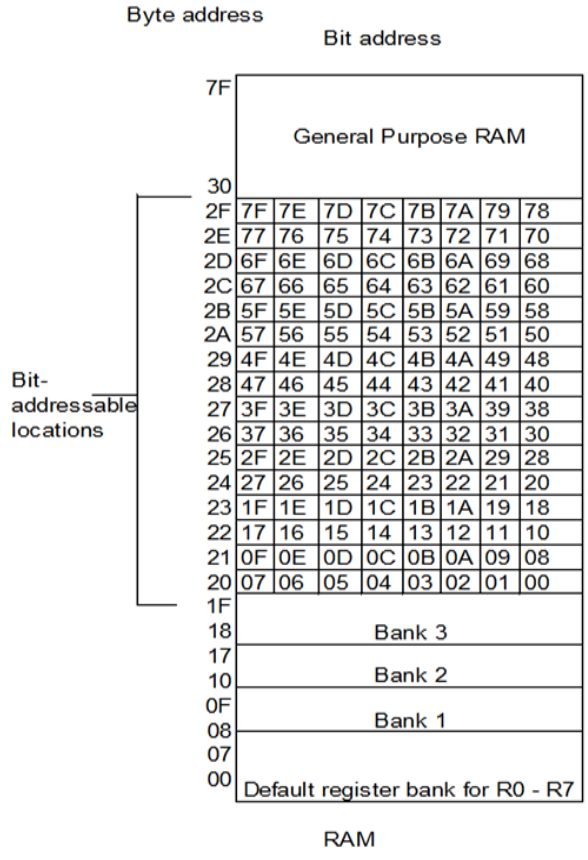



**Fig: 3.1.6 Internal RAM Byte Address**

## Using the on chip oscillator

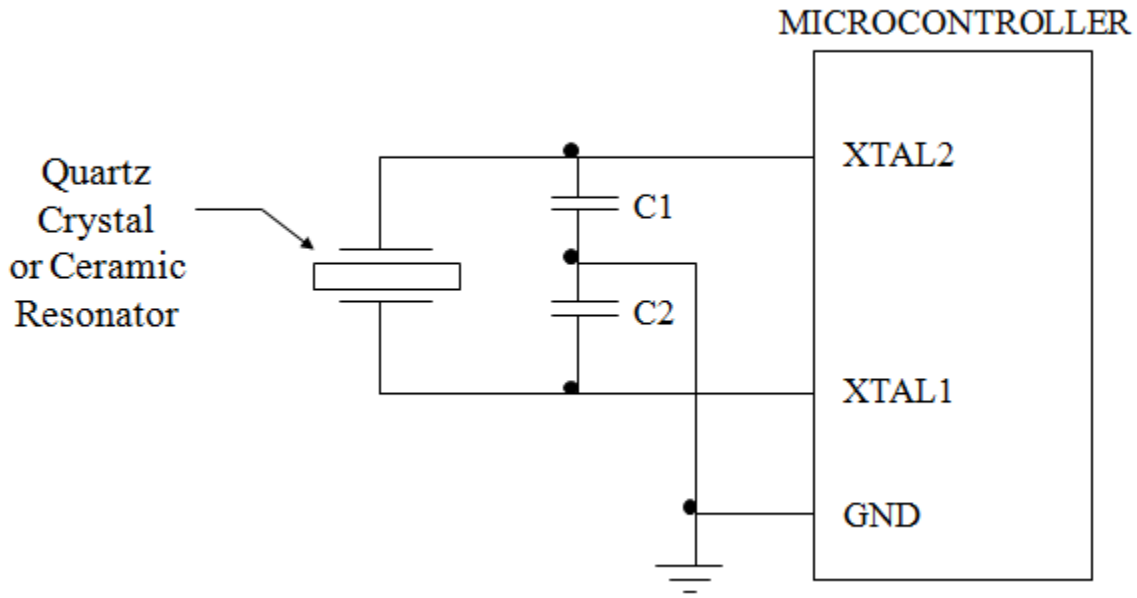



**Fig: 3.1.7 On Chip Oscillator**

## External clock drive configuration

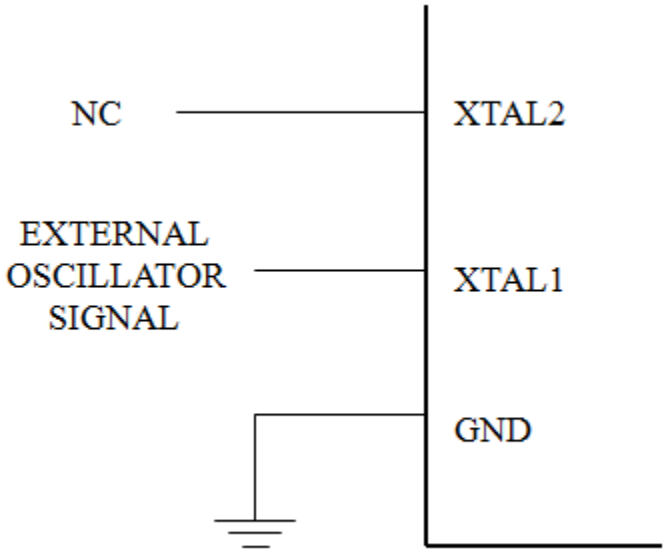



**Fig: 3.1.8 External clock drive**

## Power on reset

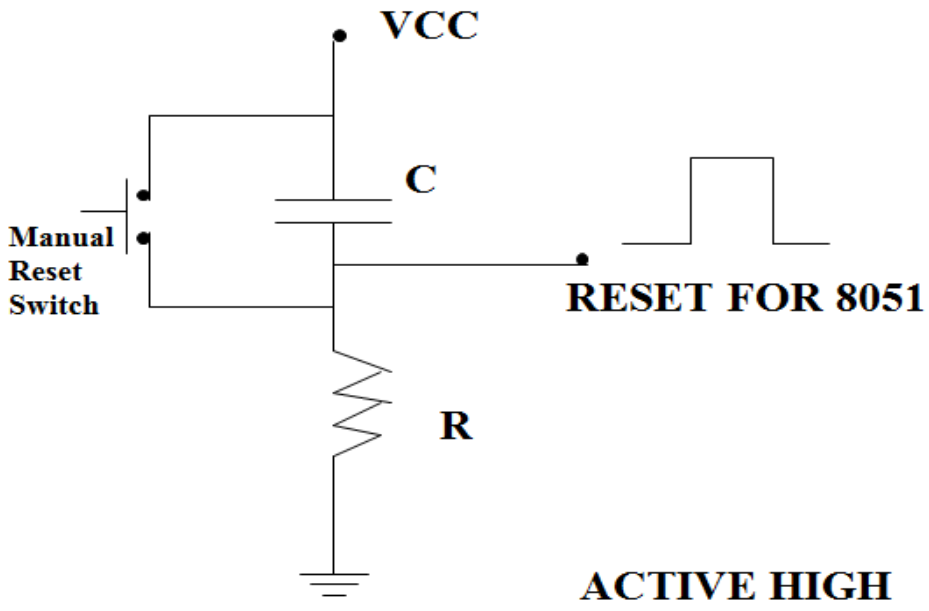



**Fig: 3.1.9 Reset Configuration**

## Regulated power supply using 7805 voltage regulator

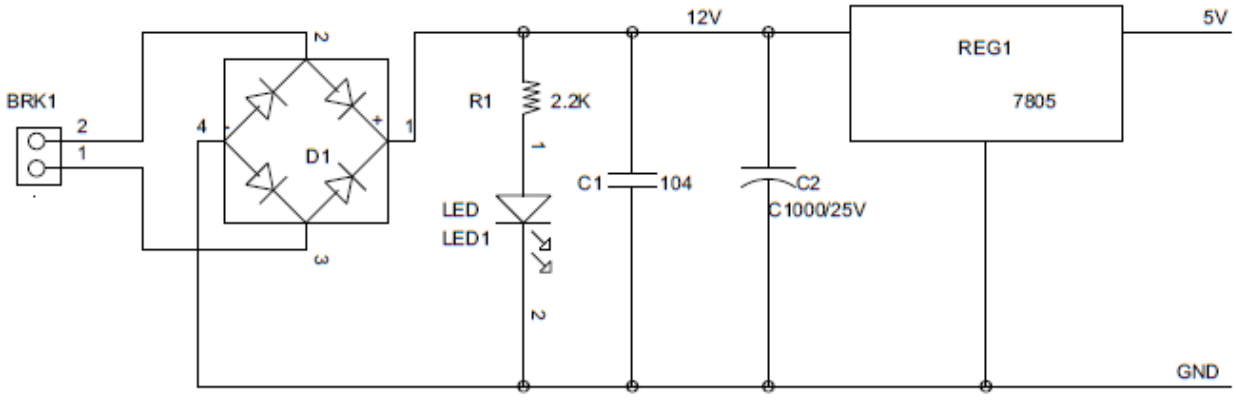



**Fig: 3.1.10 Regulated Power Supply**

## Basic circuit diagram:

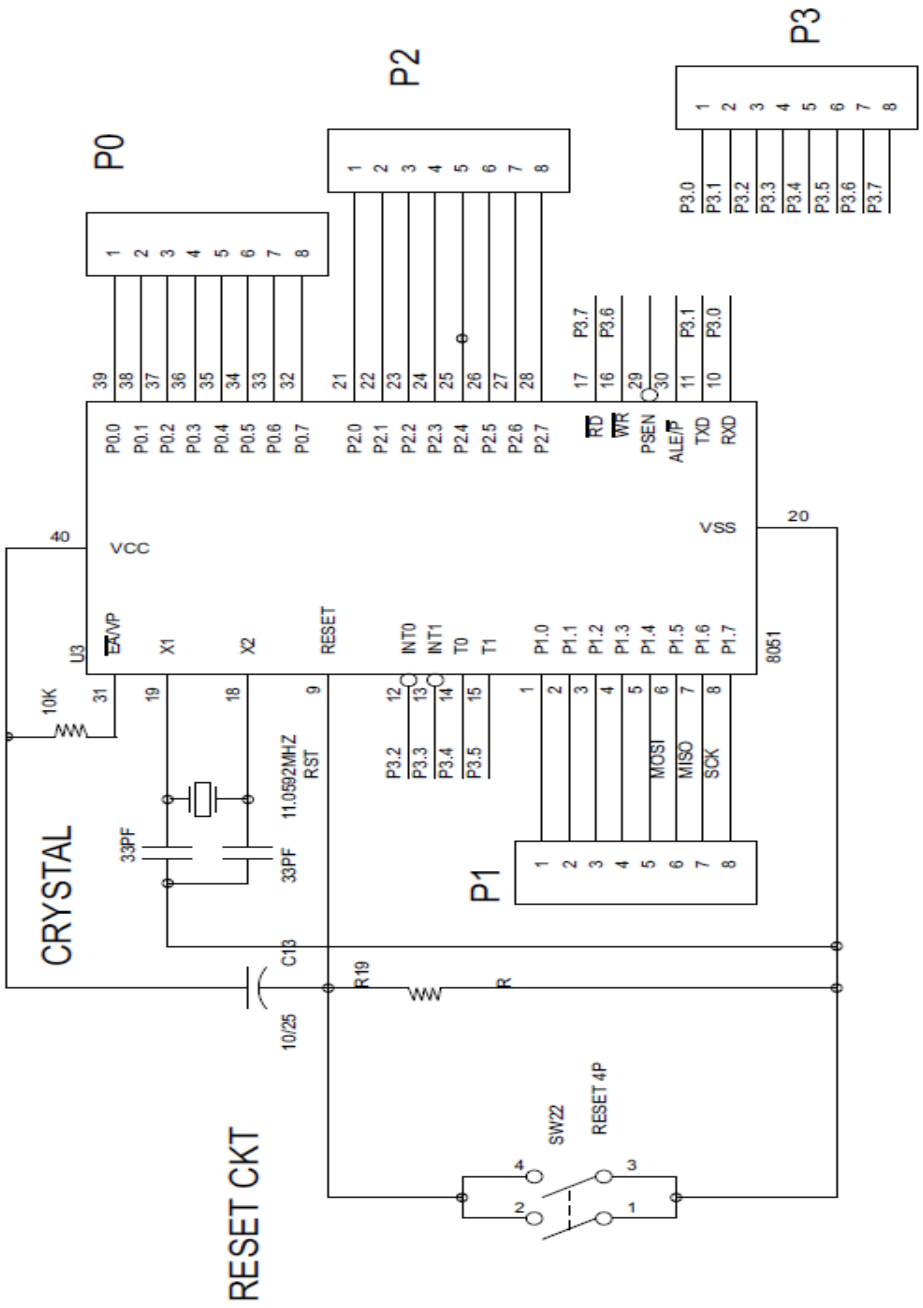



**Fig: 3.1.11 Basic Circuit Diagram of 8051**

Basic configuration required for the working of 89v51rd2. The configuration has on board oscillator and reset circuitry. Power supply to the chip is obtained through voltage regulator

7805. The IC can now be programmed using UART, ISP or external programmer like super-prog or universal programmer. The peripherals can be interfaced and applications can me designed.

# 3.2 7805 -Positive Voltage Regulator

## Description:

The L7800 series of three-terminal positive regulators is available in TO-220, TO-220FP, TO-220FM, TO-3 and D2PAK packages and several fixed output voltages, making it useful in a wide range of applications. These regulators can provide local on-card regulation, eliminating the distribution problems associated with single point regulation. Each type employs internal current limiting, thermal shut-down and safe area protection, making it essentially indestructible. If adequate heat sinking is provided, they can deliver over 1A output current. Although designed primarily as fixed voltage regulators, these devices can be used with external components to obtain adjustable voltage and currents.

## Features:

- Output current to 1.5a
- Output voltages of 5; 5.2; 6; 8; 8.5; 9;10; 12; 15; 18; 24v
- Thermal overload protection
- Short circuit protection
- Output transition SOA protection

## Packages:

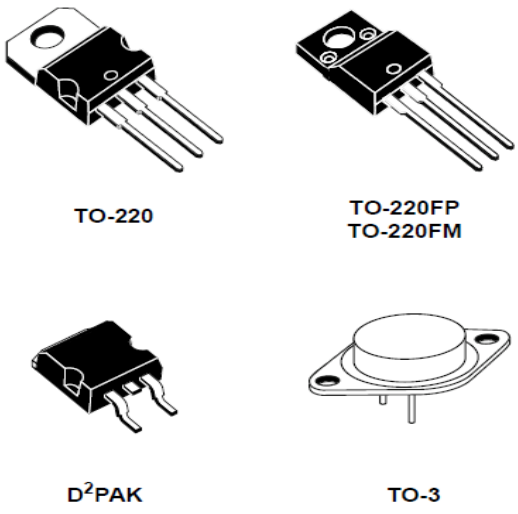



**Fig: 3.2.1 7805 Packages**

## Schematic Diagram:

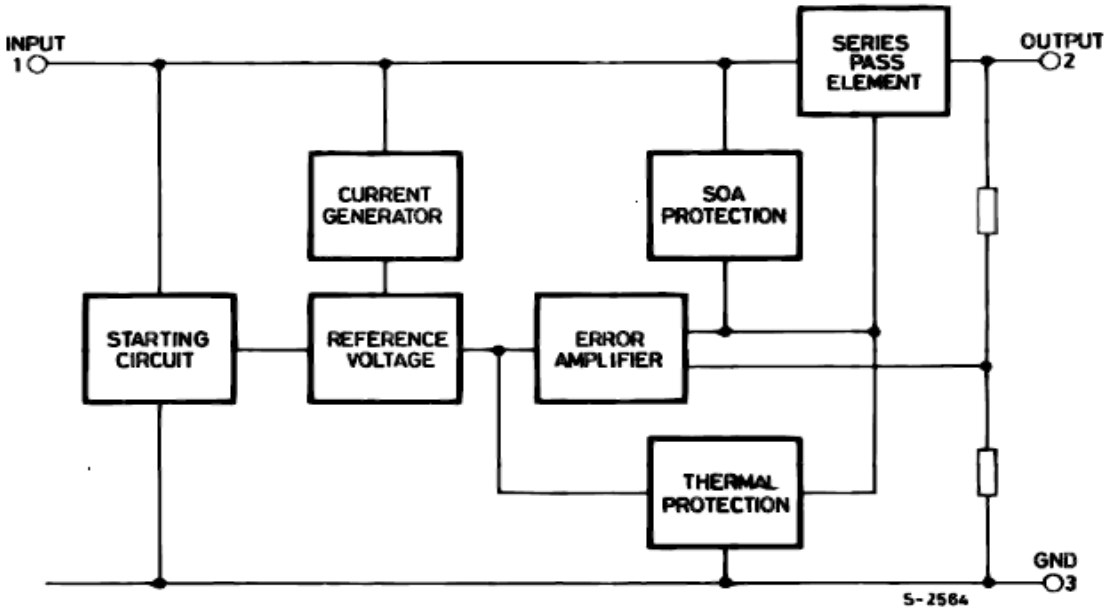



**Fig: 3.2.2 Voltage Regulator Schematic**

## Regulated power supply using 7805 voltage regulator

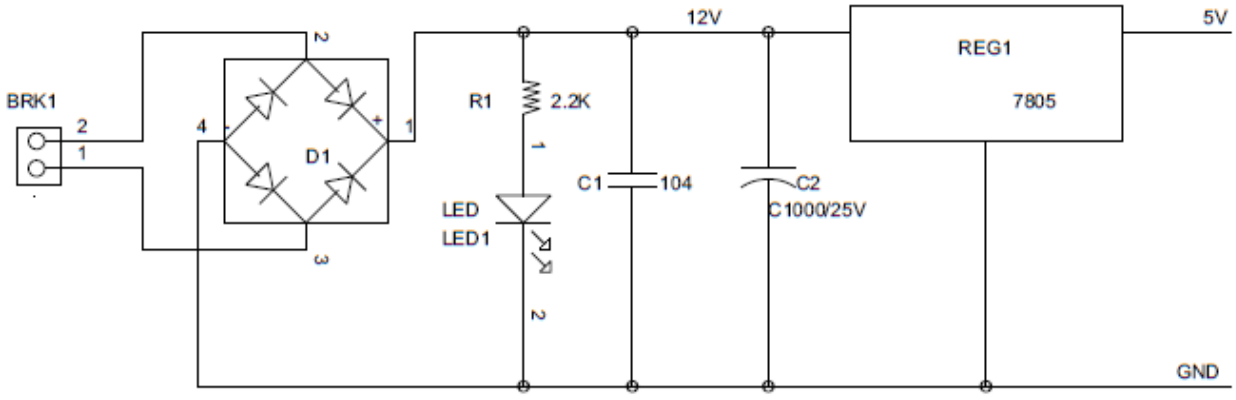



**Fig: 3.2.3 7805 Voltage Regulator**

# 3.3 MAX 232

## Description:

The MAX232 device is a dual driver/receiver that includes a capacitive voltage generator to supply EIA-232 voltage levels from a single 5-V supply. Each receiver converts EIA-232 inputs to 5-V TTL/CMOS levels. These receivers have a typical threshold of 1.3 V and a typical hysteresis of 0.5 V, and can accept ±30-V inputs. Each driver converts TTL/CMOS input levels into EIA-232 levels. The driver, receiver, and voltage-generator functions are available as cells in the Texas Instruments Lin-ASICE library. The MAX232 is characterized for operation from 0°C to 70°C. The MAX232I is characterized for operation from –40°C to 85°C.

## Features:

- Operates With Single 5-V Power Supply
- LinBiCMOSE Process Technology
- Two Drivers and Two Receivers
- ±30-V Input Levels
- Low Supply Current . . . 8 mA Typical
- Meets or Exceeds TIA/EIA-232-F and ITU Recommendation V.28
- Designed to be Interchangeable With Maxim MAX232

## Applications:

- TIA/EIA-232-F Battery-Powered Systems
- Terminals
- Modems
- Computers

## Packages:

Package Options Include Plastic Small-Outline (D, DW) Packages and Standard Plastic (N) DIPs

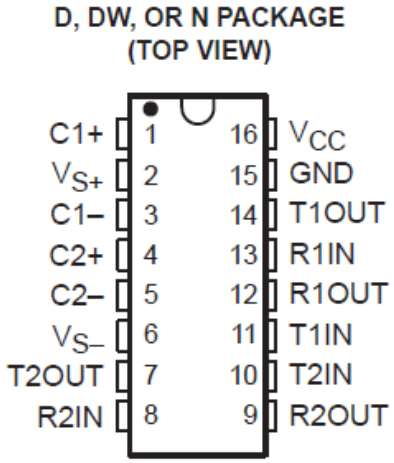



**Fig: 3.3.1 MAX232 Pin Diagram**

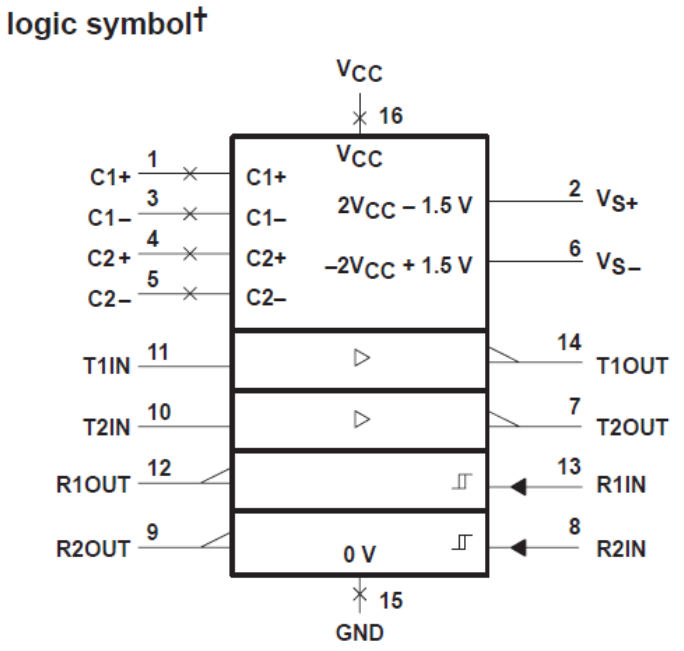



**Fig: 3.3.2 MAX232 Logic Diagram**

# MAX 232 Connections:

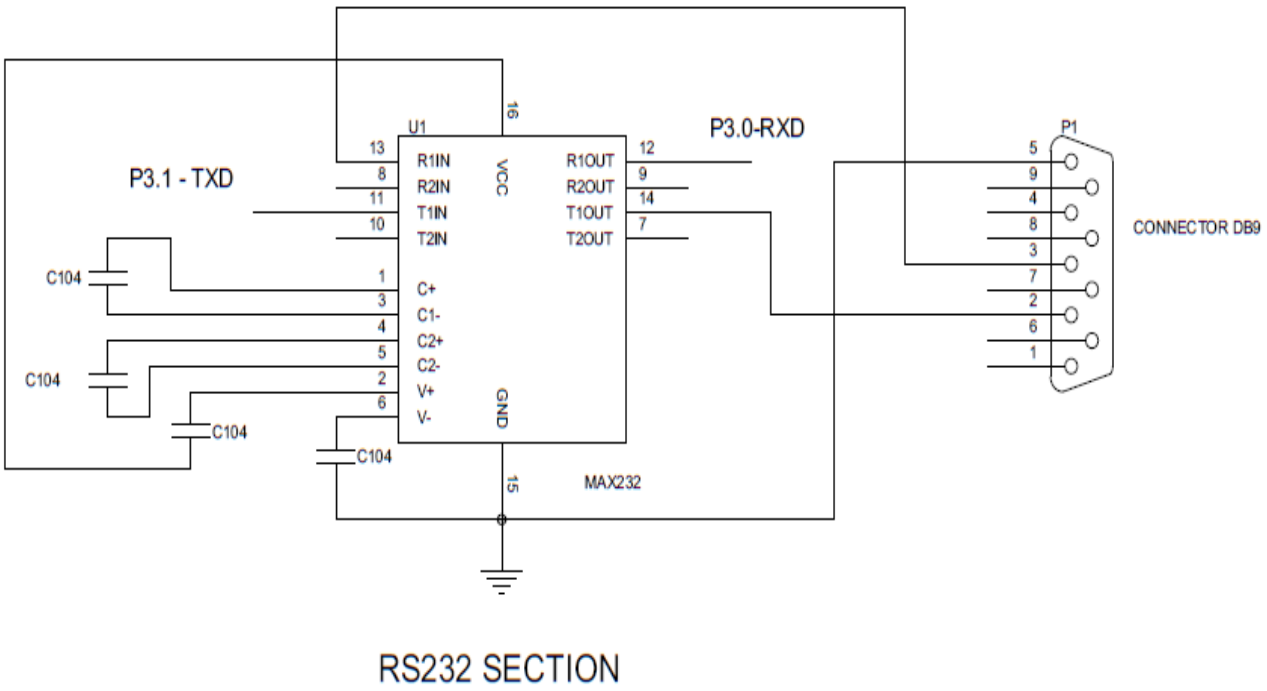



**Fig: 3.3.3 Max232 Connections**

# 3.4 PCF 8591 8-bit A/D and D/A converter

## General Description

The PCF8591 is a single-chip, single-supply low power 8-bit CMOS data acquisition device with four analog inputs, one analog output and a serial I2C-bus interface. Three address pins A0, A1 and A2 are used for programming the hardware address, allowing the use of up to eight devices connected to the I2C-bus without additional hardware. Address, control and data to and from the device are transferred serially via the two-line bidirectional I2C-bus. The functions of the device include analog input multiplexing, on-chip track and hold function, 8-bit analog-to-digital conversion and an 8-bit digital-to-analog conversion. The maximum conversion rate is given by the maximum speed of the I2C-bus.

## Features

- Single power supply
- Operating supply voltage 2.5 V to 6 V
- Low standby current
- Serial input/output via I2C-bus
- Address by 3 hardware address pins
- Sampling rate given by I2C-bus speed
- 4 analog inputs programmable as single-ended or differential inputs
- Auto-incremented channel selection
- Analog voltage range from VSS to VDD
- On-chip track and hold circuit
- 8-bit successive approximation A/D conversion
- Multiplying DAC with one analog output.

## Applications

- Closed loop control systems
- Low power converter for remote data acquisition
- Battery operated equipment
- Acquisition of analog values in automotive, audio and TV applications.

# Block diagram

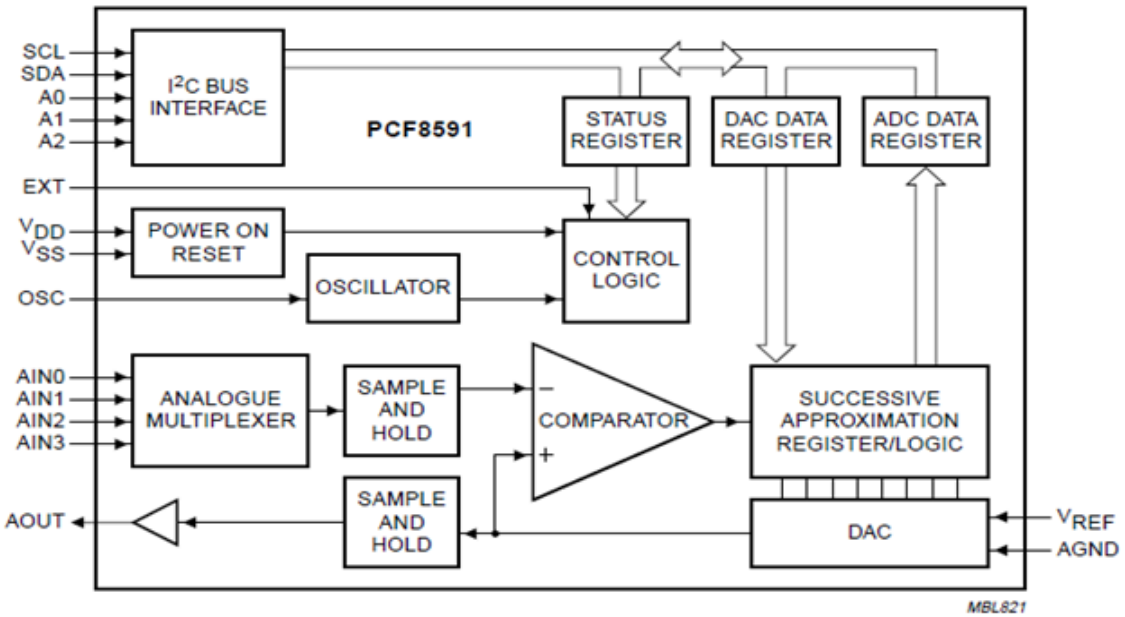
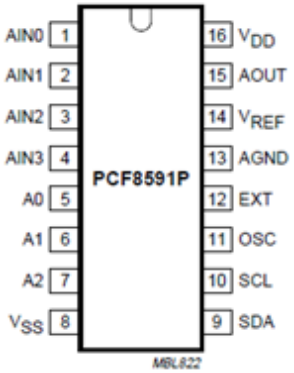



**Fig: 3.4.1 PCF8591 Block Diagram**

# Pin Diagram &Pin Function

| SYMBOL | PIN | DESCRIPTION |
|---|---|---|
| AINO | 1 | analog inputs (A/D converter) |
| AIN1 | 2 | |
| AIN2 | 3 | |
| AIN3 | 4 | |
| A0 | 5 | hardware address |
| A1 | 6 | |
| A2 | 7 | |
| $V_{SS}$ | 8 | negative supply voltage |
| SDA | 9 | $I^2C$-bus data input/output |
| SCL | 10 | $I^2C$-bus clock input |
| OSC | 11 | oscillator input/output |
| EXT | 12 | external/internal switch for oscillator input |
| AGND | 13 | analog ground |
| $V_{REF}$ | 14 | voltage reference input |
| AOUT | 15 | analog output (D/A converter) |
| $V_{DD}$ | 16 | positive supply voltage |




Pinning diagram (DIP16).

**Fig: 3.4.2 Pin Diagram and Pin Functions of PCF8591**

# Functional Description

## Addressing

Each PCF8591 device in an I2C-bus system is activated by sending a valid address to the device. The address consists of a fixed part and a programmable part. The programmable part must be set according to the address pins A0, A1 and A2. The address always has to be sent as the first byte after the start condition in the I2C-bus protocol. The last bit of the address byte is the read/write-bit which sets the direction of the following data transfer.

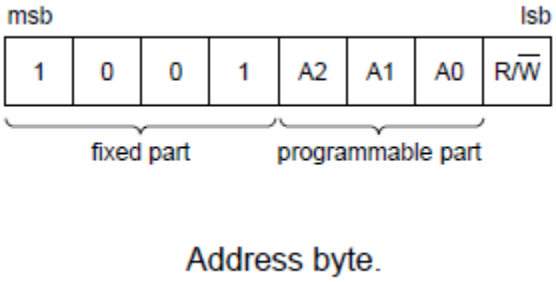
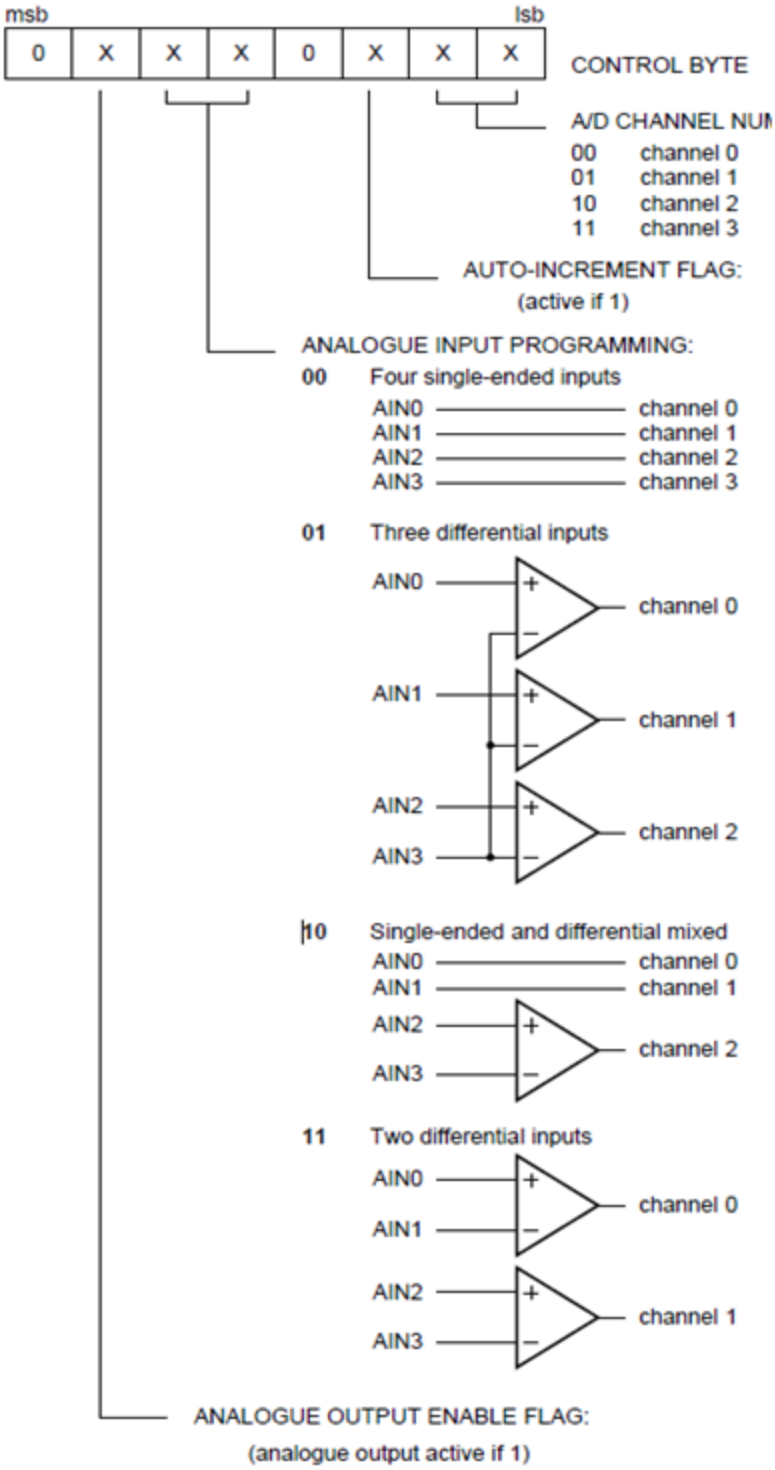



**Fig: 3.4.3 Address Byte of PCF8591**

## Control

The second byte sent to a PCF8591 device will be stored in its control register and is required to control the device function. The upper nibble of the control register is used for enabling the analog output, and for programming the analog inputs as single-ended or differential inputs. The lower nibble selects one of the analog input channels defined by the upper nibble (see Fig.5). If the auto-increment flag is set, the channel number is incremented automatically after each A/D conversion. If the auto-increment mode is desired in applications where the internal oscillator is used, the analog output enable flag in the control byte (bit 6) should be set. This allows the internal oscillator to run continuously, thereby preventing conversion errors resulting from oscillator start-up delay. The analog output enable flag may be reset at other times to reduce quiescent power consumption. The selection of a non-existing input channel results in the highest available channel number being allocated. Therefore, if the auto-increment flag is set, the next selected channel will be always channel 0. The most significant bits of both nibbles are reserved for future functions and have to be set to logic 0. After a Power-on reset condition all bits of the control register are reset to logic 0. The D/A converter and the oscillator are disabled for power saving. The analog output is switched to a high-impedance state.

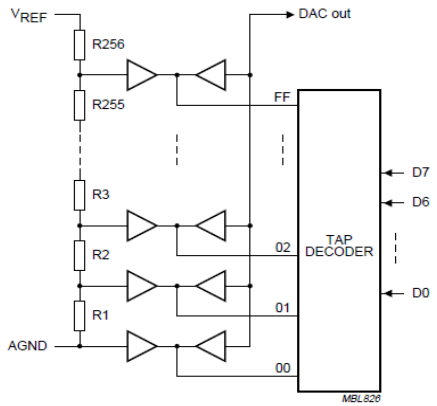



Control byte.

**Fig: 3.4.4 Control Byte of PCF8591**

# D/A conversion

The third byte sent to a PCF8591 device is stored in the DAC data register and is converted to the corresponding analog voltage using the on-chip D/A converter. This D/A converter consist of a resistor divider chain connected to the external reference voltage with 256 taps and selection switches. The tap-decoder switches one of these taps to the DAC output line. The analog output voltage is buffered by an auto-zeroed unity gain amplifier. This buffer amplifier may be switched on or off by setting the analog output-enable flag of the control register. In the active state the output voltage is held until a further data byte is sent. The on-chip D/A converters are also used for successive approximation A/D conversion. In order to release the DAC for an A/D conversion cycle the unity gain amplifier is equipped with a track and hold circuit. This circuit holds the output voltage while executing the A/D conversion.




DAC resistor divider chain.

**Fig: 3.4.5 DAC Resistor Driver Chain**

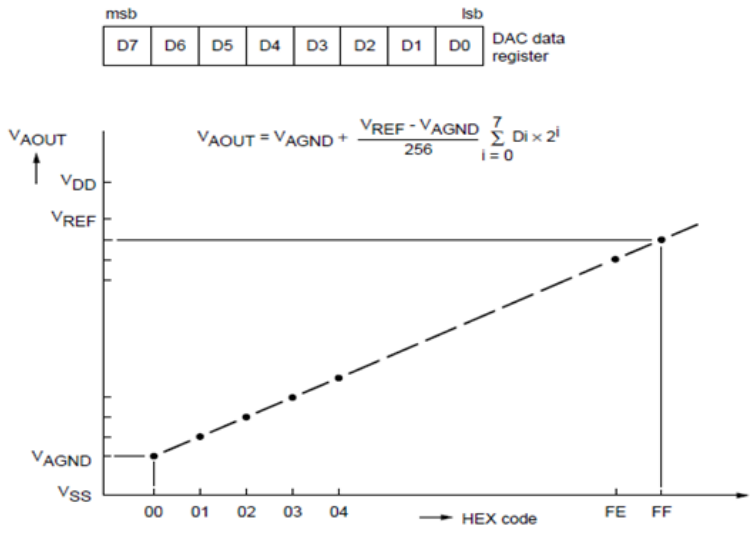



**Fig: 3.4.6 DAC data and DC Conversion Characteristics**

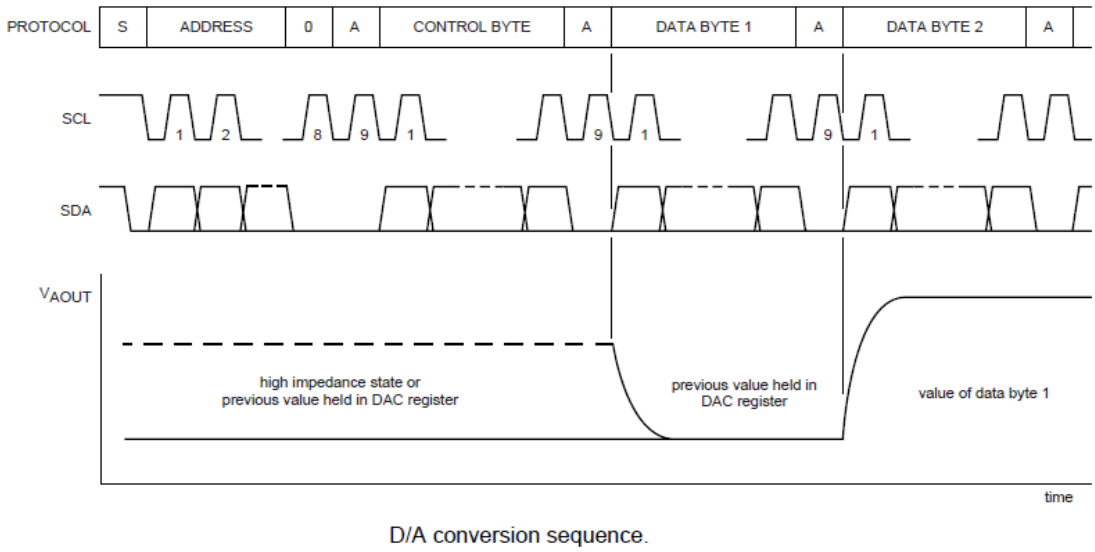
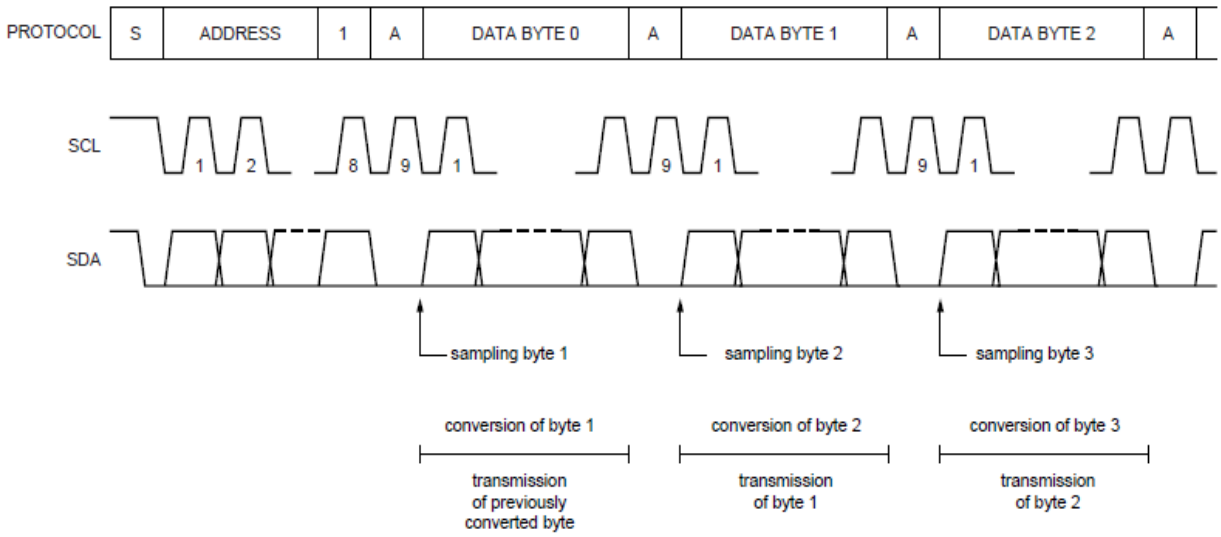



**Fig: 3.4.7 D/A Conversion Sequence**

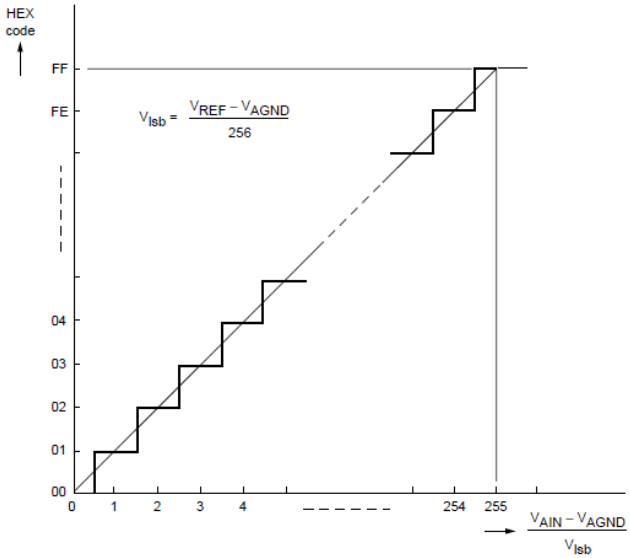
# A/D Conversion

The A/D converter makes use of the successive approximation conversion technique. The on-chip D/A converter and a high-gain comparator are used temporarily during an A/D conversion cycle. A/D conversion cycle is always started after sending a valid read mode address to a PCF8591 device. The A/D conversion cycle is triggered at the trailing edge of the acknowledge clock pulse and is executed while transmitting the result of the previous conversion. Once a conversion cycle is triggered an input voltage sample of the selected channel is stored on the chip and is converted to the corresponding 8-bit binary code. Samples picked up from differential inputs are converted to an 8-bit twos complement code. The conversion result is stored in the ADC data register and awaits transmission. If the auto-increment flag is set the next channel is selected. The first byte transmitted in a read cycle contains the conversion result code of the previous read cycle. After a Power-on reset condition the first byte read is a hexadecimal 80. The maximum A/D conversion rate is given by the actual speed of the I2C-bus.




A/D conversion sequence.

**Fig: 3.4.8 A/D Conversion Sequence**




A/D conversion characteristics of single-ended inputs.

**Fig: 3.4.9 A/D Conversion Characteristics of Single Ended Inputs**

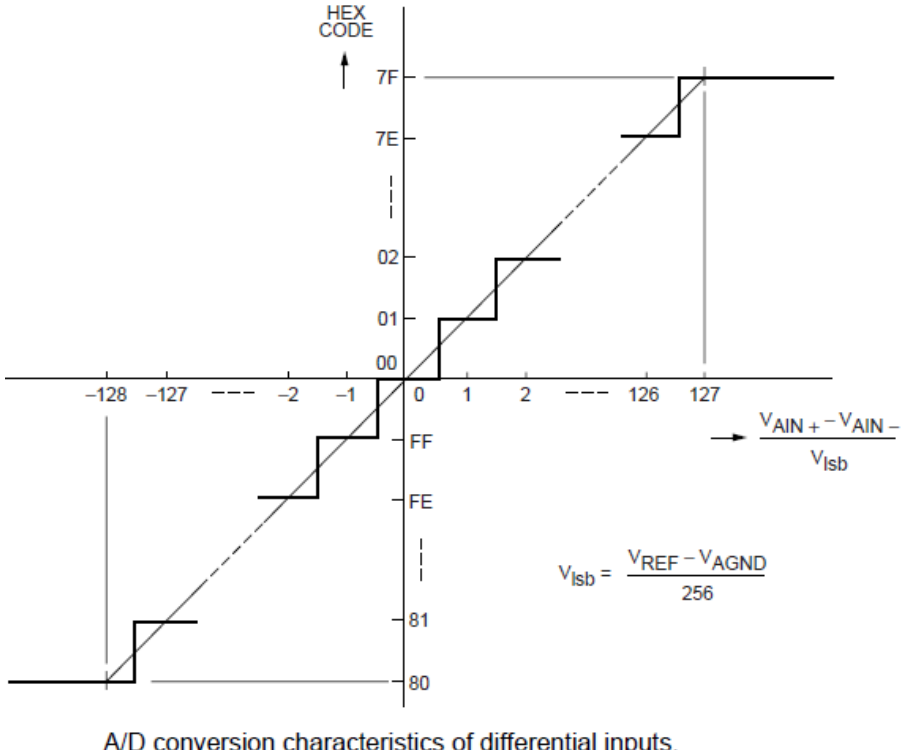



**Fig: 3.4.10 A/D Conversion Characteristics of Differential inputs**

# System configuration

A device generating a message is a ‘transmitter’, a device receiving a message is the ‘receiver’. The device that controls the message is the ‘master’ and the devices which are controlled by the master are the ‘slaves’.

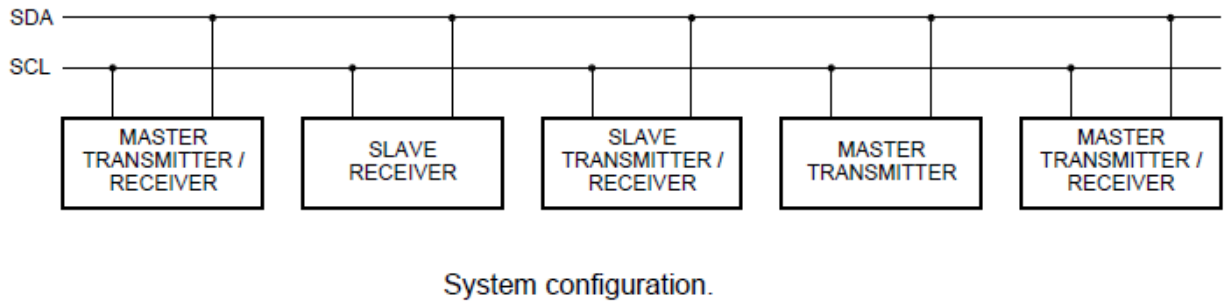
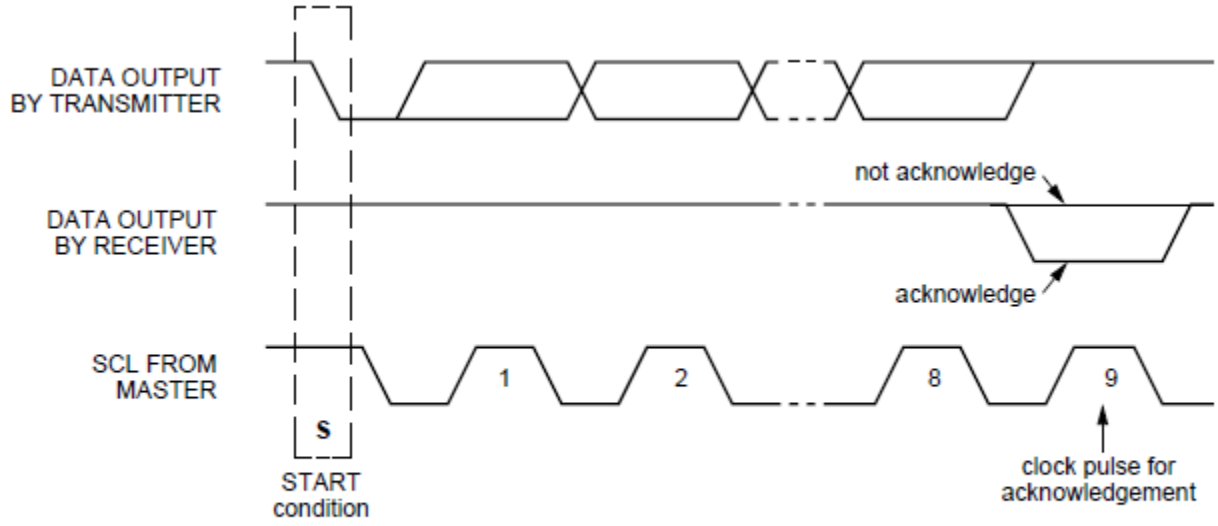



**Fig: 3.4.11 I2C System Configuration**

## Acknowledge

The number of data bytes transferred between the start and stop conditions from transmitter to receiver is not limited. Each data byte of eight bits is followed by one acknowledge bit. The acknowledge bit is a HIGH level put on the bus by the transmitter whereas the master also generates an extra acknowledge related clock pulse. A slave receiver which is addressed must generate an acknowledge after the reception of each byte. Also a master must generate an acknowledge after the reception of each byte that has been clocked out of the slave transmitter. The device that acknowledges has to pull down the SDA line during the acknowledge clock pulse, so that the SDA line is stable LOW during the HIGH period of the acknowledge related clock pulse. A master receiver must signal an end of data to the transmitter by **not** generating an acknowledge on the last byte that has been clocked out of the slave. In this event the transmitter must leave the data line HIGH to enable the master to generate a stop condition.

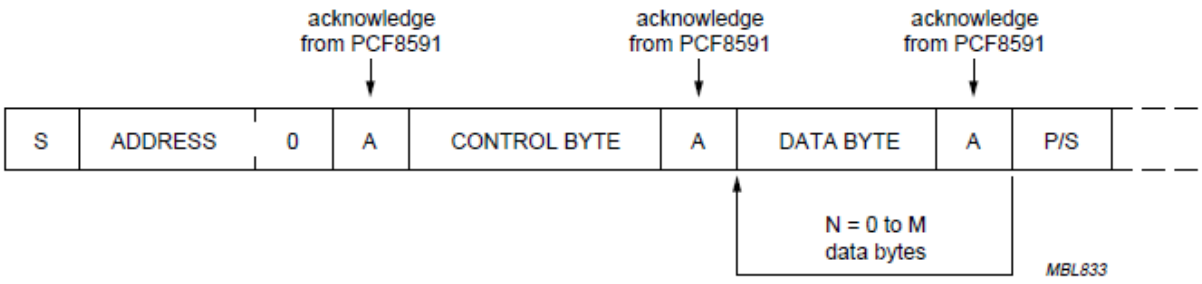



Acknowledgement on the I²C-bus.

**Fig: 3.4.12 Acknowledgement on the I2C bus**

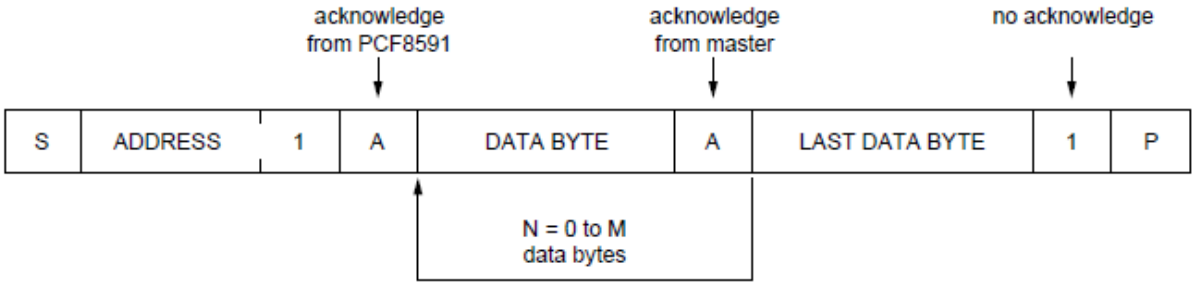
## I2C Bus Protocol

After a start condition a valid hardware address has to be sent to a PCF8591 device. The read/write bit defines the direction of the following single or multiple byte data transfer. For the format and the timing of the start condition (S), the stop condition (P) and the acknowledge bit (A) refer to the I2C-bus characteristics. In the write mode a data transfer is terminated by sending either a stop condition or the start condition of the next data transfer.




Bus protocol for write mode, D/A conversion.

**Fig: 3.4.13 Bus Protocol for Write mode, D/A Conversion**




Bus protocol for read mode, A/D conversion.

**Fig: 3.4.14 Bus Protocol for Read mode, A/D Conversion**

# 3.5 PCF8574 Remote 8-bit I/O expander for I2C-bus

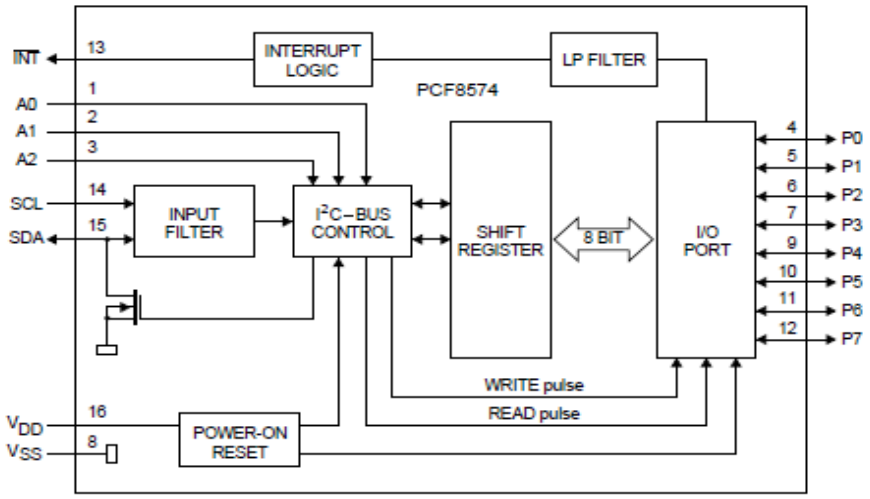
## General Description

The PCF8574 is a silicon CMOS circuit. It provides general purpose remote I/O expansion for most microcontroller families via the two-line bidirectional bus (I2C-bus). The device consists of an 8-bit quasi-bidirectional port and an I2C-bus interface. The PCF8574 has a low current consumption and includes latched outputs with high current drive capability for directly driving LEDs. It also possesses an interrupt line (INT) which can be connected to the interrupt logic of the microcontroller. By sending an interrupt signal on this line, the remote I/O can inform the microcontroller if there is incoming data on its ports without having to communicate via the I2C-bus. This means that the PCF8574 can remain a simple slave device. The PCF8574 and PCF8574A versions differ only in their slave address.

## Features

- Operating supply voltage 2.5 to 6 V
- Low standby current consumption of 10 mA maximum
- I2C-bus to parallel port expander
- Open-drain interrupt output
- 8-bit remote I/O port for the I2C-bus
- Compatible with most microcontrollers
- Latched outputs with high current drive capability for directly driving LEDs
- Address by 3 hardware address pins for use of up to 8 devices (up to 16 with PCF8574A)
- DIP 16, or space-saving SO16 or SSOP20 packages.

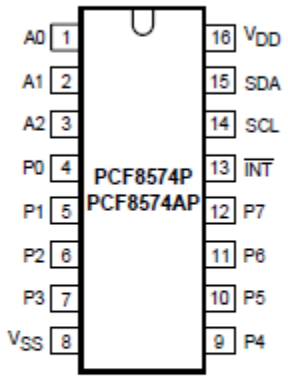
## Block diagram




Block diagram (pin numbers apply to DIP16 and SO16 packages).

**Fig: 3.5.1 PCF 8574 Block Diagram**

# Pin Diagram &Pin Function

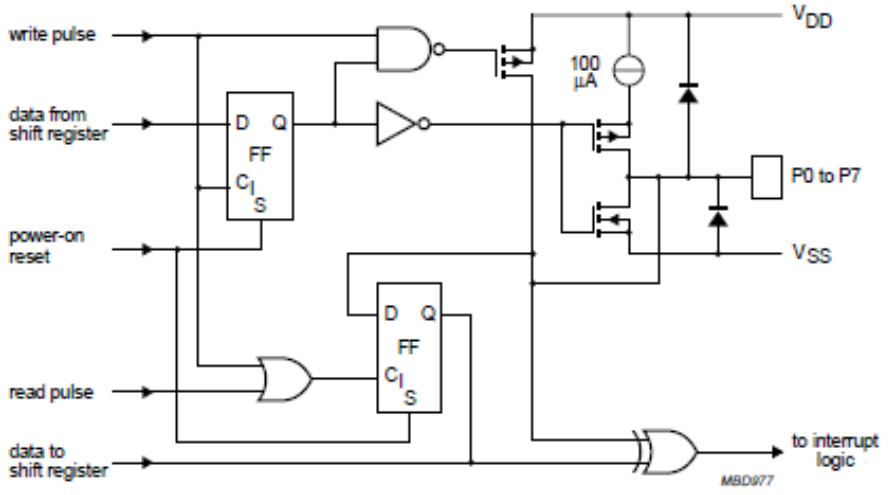



Pin configuration (DIP16).

| SYMBOL | PIN | DESCRIPTION |
|---|---|---|
| A0 | 1 | address input 0 |
| A1 | 2 | address input 1 |
| A2 | 3 | address input 2 |
| P0 | 4 | quasi-bidirectional I/O 0 |
| P1 | 5 | quasi-bidirectional I/O 1 |
| P2 | 6 | quasi-bidirectional I/O 2 |
| P3 | 7 | quasi-bidirectional I/O 3 |
| $V_{SS}$ | 8 | supply ground |
| P4 | 9 | quasi-bidirectional I/O 4 |
| P5 | 10 | quasi-bidirectional I/O 5 |
| P6 | 11 | quasi-bidirectional I/O 6 |
| P7 | 12 | quasi-bidirectional I/O 7 |
| $\overline{INT}$ | 13 | interrupt output (active LOW) |
| SCL | 14 | serial clock line |
| SDA | 15 | serial data line |
| $V_{DD}$ | 16 | supply voltage |

**Fig: 3.5.2 Pin Function and Pin Diagram of PCF8574**

# Functional Description




Simplified schematic diagram of each I/O.

**Fig: 3.5.3 Functional Diagram of PCF8574**

# Addressing

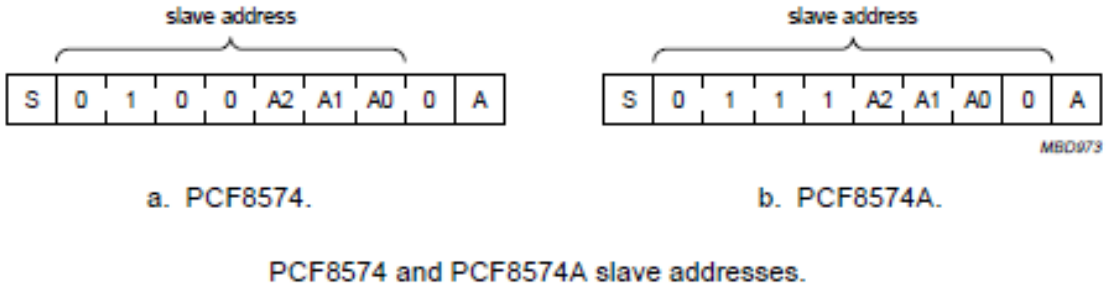



**Fig: 3.5.4 PCF8574 Slave Address**

Each of the PCF8574's eight I/Os can be independently used as an input or output. Input data is transferred from the port to the microcontroller by the READ mode. Output data is transmitted to the port by the WRITE mode.

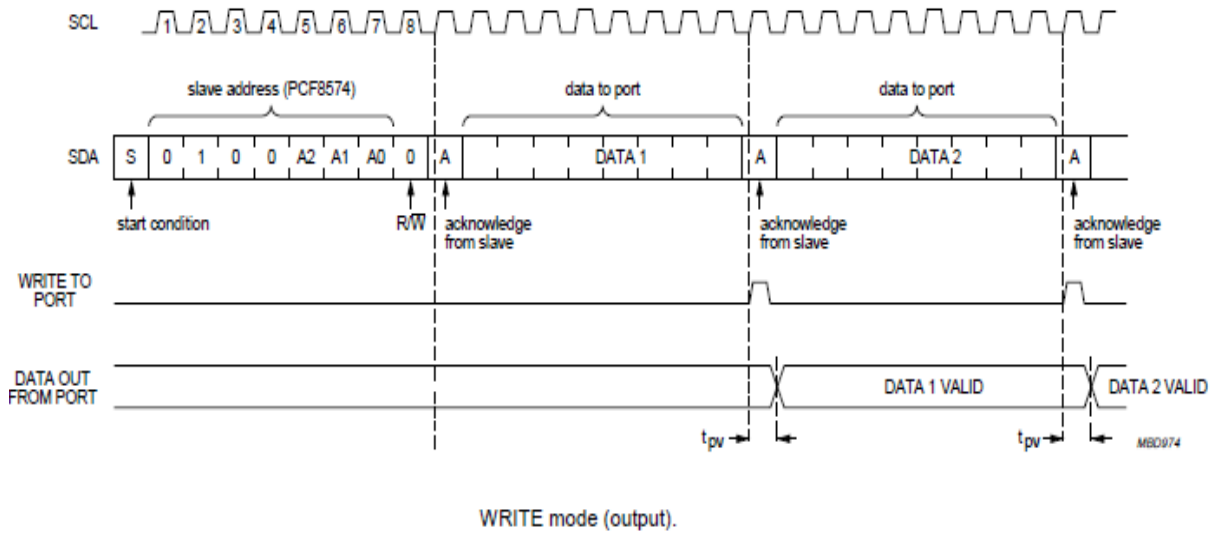
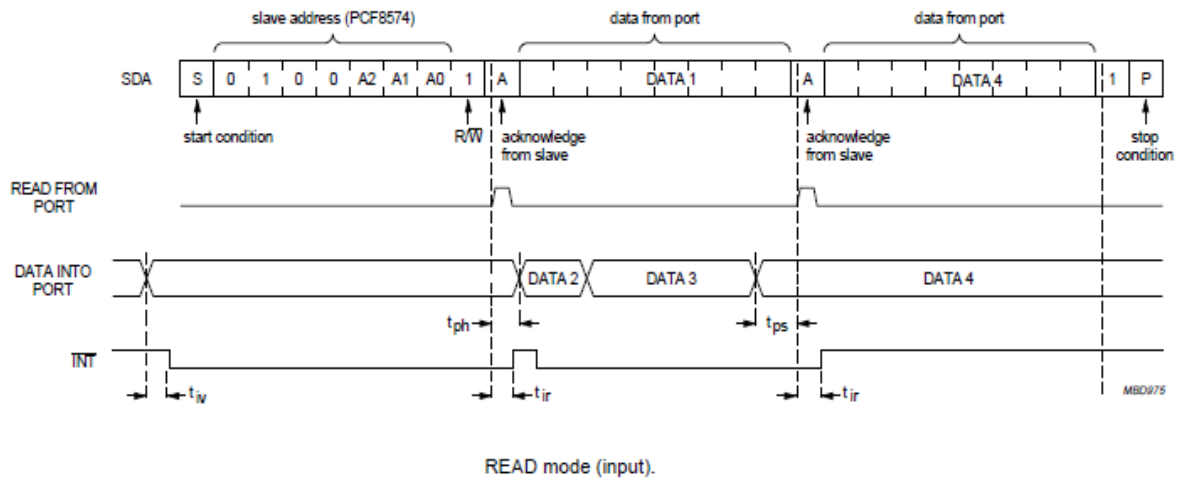



**Fig: 3.5.5 Write Mode PCF8574**




READ mode (input).

**Fig: 3.5.6 Read Mode PCF8574**

A LOW-to-HIGH transition of SDA, while SCL is HIGH is defined as the stop condition (P). Transfer of data can be stopped at any moment by a stop condition. When this occurs, data present at the last acknowledge phase is valid (output mode). Input data is lost.

## System configuration

A device generating a message is a 'transmitter', a device receiving is the 'receiver'. The device that controls the message is the 'master' and the devices which are controlled by the master are the 'slaves'.

## Acknowledge

The number of data bytes transferred between the start and the stop conditions from transmitter to receiver is not limited. Each byte of eight bits is followed by one acknowledge bit. The acknowledge bit is a HIGH level put on the bus by the transmitter whereas the master generates an extra acknowledge related clock pulse. A slave receiver which is addressed must generate an acknowledge after the reception of each byte. Also a master must generate an acknowledge after the reception of each byte that has been clocked out of the slave transmitter. The device that acknowledges has to pull down the SDA line during the acknowledge clock pulse, so that the SDA line is stable LOW during the HIGH period of the acknowledge related clock pulse, set-up and hold times must be taken into account. A master receiver must signal an end of data to the transmitter by **not** generating an acknowledge on the last byte that has been clocked out of the slave. In this event the transmitter must leave the data line HIGH to enable the master to generate a stop condition.

# 3.6 AT24C32 2-Wire Serial EEPROM 32K (4096 x 8)

## General Description

The AT24C32/64 provides 32,768/65,536 bits of serial electrically erasable and programmable read only memory (EEPROM) organized as 4096/8192 words of 8 bits each. The device's cascade able feature allows up to 8 devices to share a common 2- wire bus. The device is optimized for use in many industrial and commercial applications where low power and low voltage operation are essential. The AT24C32/64 is available in space saving 8-pin JEDEC PDIP, 8-pin JEDEC SOIC, 8-pin EIAJ SOIC, and 8-pin TSSOP (AT24C64) packages and is accessed via a 2-wire serial interface In addition, the entire family is available in 2.7V (2.7V to 5.5V) and 1.8V (1.8V to 5.5V) versions.

## Features

- Low-Voltage and Standard-Voltage Operation
    - 2.7 (VCC = 2.7V to 5.5V)
    - 1.8 (VCC = 1.8V to 5.5V)
- Low-Power Devices (ISB = 2 μA at 5.5V) Available
- Internally Organized 4096 x 8, 8192 x 8
- 2-Wire Serial Interface
- Schmitt Trigger, Filtered Inputs for Noise Suppression
- Bidirectional Data Transfer Protocol
- 100 kHz (1.8V, 2.5V, 2.7V) and 400 kHz (5V) Clock Rate
- Write Protect Pin for Hardware Data Protection
- 32-Byte Page Write Mode (Partial Page Writes Allowed)
- Self-Timed Write Cycle (10 ms max)
- High Reliability
    - Endurance: 1 Million Write Cycles
    - Data Retention: 100 Years
- Automotive Grade and Extended Temperature Devices Available
- 8-Pin JEDEC PDIP, 8-Pin JEDEC SOIC, 8-Pin EIAJ SOIC,
- and 8-pin TSSOP Packages

## Pin Diagram &Pin Description

| Pin Name | Function |
|---|---|
| A0 - A2 | Address Inputs |
| SDA | Serial Data |
| SCL | Serial Clock Input |
| WP | Write Protect |

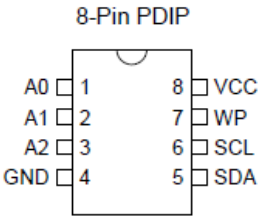



**Fig: 3.6.1 Pin Diagram and Pin Description of AT24C32**

## Block diagram

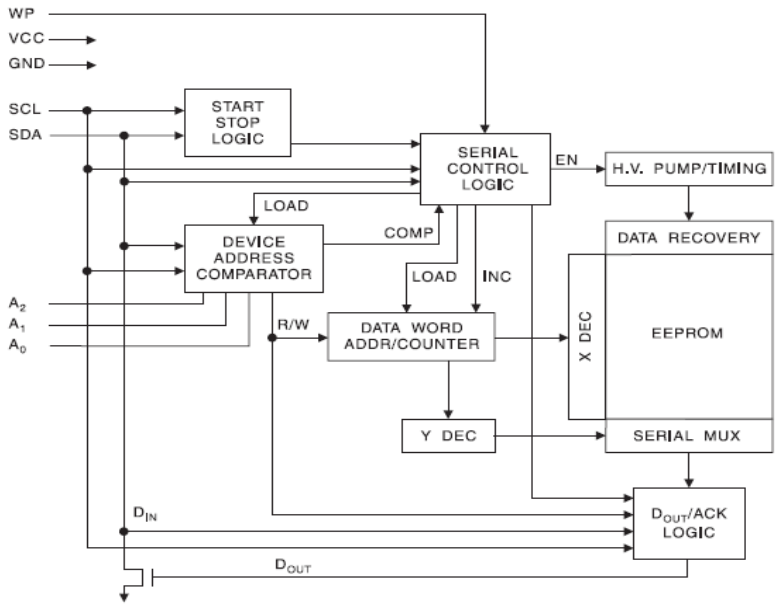



**Fig: 3.6.2 Block Diagram of AT24C32**

## Pin Description

SERIAL CLOCK (SCL): The SCL input is used to positive edge clock data into each EEPROM device and negative edge clock data out of each device.

SERIAL DATA (SDA): The SDA pin is bidirectional for serial data transfer. This pin is open-drain driven and may be wire-O Red with any number of other open-drain or open collector devices.

DEVICE/PAGE ADDRESSES (A2, A1, A0): The A2, A1 and A0 pins are device address inputs that are hard wired or left not connected for hardware compatibility with AT24C16. When the pins are hardwired, as many as eight 32K/64K devices may be addressed on a single bus system (device addressing is discussed in detail under the Device Addressing section). When the pins are not hardwired, the default A2, A1, and A0 are zero.

WRITE PROTECT (WP): The write protect input, when tied to GND, allows normal write operations. When WP is tied high to VCC, all write operations to the upper quadrant (8/16K bits) of memory are inhibited. If left unconnected, WP is internally pulled down to GND.

## Memory Organization

AT24C32/64, 32K/64K SERIAL EEPROM: The 32K/64K is internally organized as 256 pages of 32 bytes each. Random word addressing requires a 12/13 bit data word address.

## Device Operation

CLOCK and DATA TRANSITIONS: The SDA pin is normally pulled high with an external device. Data on the SDA pin may change only during SCL low time periods (refer to Data Validity timing diagram). Data changes during SCL high periods will indicate a start or stop condition as defined below.

START CONDITION: A high-to-low transition of SDA with SCL high is a start condition which must precede any other command (refer to Start and Stop Definition timing diagram).

STOP CONDITION: A low-to-high transition of SDA with SCL high is a stop condition. After a read sequence, the stop command will place the EEPROM in a standby power mode (refer to Start and Stop Definition timing diagram).

ACKNOWLEDGE: All addresses and data words are serially transmitted to and from the EPROM in 8-bit words. The EEPROM sends a zero during the ninth clock cycle to acknowledge that it has received each word.

STANDBY MODE: The AT24C32/64 features a low power standby mode which is enabled: a) upon power-up and b) after the receipt of the STOP bit and the completion of any internal operations.

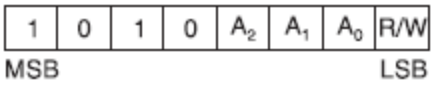
MEMORY RESET: After an interruption in protocol, power loss or system reset, any 2- wire part can be reset by following these steps:
(a) Clock up to 9 cycles, (b) look for SDA high in each cycle while SCL is high and then (c) create a start condition as SDA is high.

## Device Addressing

The 32K/64K EEPROM requires an 8-bit device address word following a start condition to enable the chip for a read or write operation. The device address word consists of a mandatory one, zero sequence for the first four most significant bits as shown. This is common to all 2-wire EEPROM devices. The 32K/64K uses the three device address bits A2, A1, A0 to allow as many as eight devices on the same bus. These bits must compare to their corresponding hardwired input pins. The A2, A1, and A0 pins use an internal proprietary circuit that biases them to a logic low condition if the pins are allowed to float. The eighth bit of the device address is the read/write operation select bit. A read operation is initiated if this bit is high and a write operation is initiated if this bit is low. Upon a compare of the device address, the EEPROM will output a zero. If a compare is not made, the device will return to standby state.

NOISE PROTECTION**:** Special internal circuitry placed on the SDA and SCL pins prevent small noise spikes from activating the device. A low-VCC detector (5-volt option) resets the device to prevent data corruption in a noisy environment.

DATA SECURITY**:** The AT24C32/64 has a hardware data protection scheme that allows the user to write protect the upper quadrant (8/16K bits) of memory when the WP pin is at VCC.




## Write Operations

BYTE WRITE**:** A write operation requires two 8-bit data word addresses following the device address word and acknowledgment. Upon receipt of this address, the EEPROM will again respond with a zero and then clock in the first 8-bit data word. Following receipt of the 8-bit data word, the EEPROM will output a zero and the addressing device, such as a microcontroller, must terminate the write sequence with a stop condition. At this time the EEPROM enters an internally-timed write cycle, tWR, to the nonvolatile memory. All inputs are disabled during this write cycle and the EEPROM will not respond until the write is complete.

PAGE WRITE**:** The 32K/64K EEPROM is capable of 32-byte page writes. A page write is initiated the same way as a byte write, but the microcontroller does not send a stop condition after the first data word is clocked in. Instead, after the EEPROM acknowledges receipt of the first data word, the microcontroller can transmit up to 31 more data words. The EEPROM will respond with a zero after each data word received. The microcontroller must terminate the page write sequence with a stop condition. The data word address lower 5 bits are internally incremented following the receipt of each data word. The higher data word address bits are not incremented, retaining the memory page row location. When the word address, internally generated, reaches the page boundary, the following byte is placed at the beginning of the same page. If more than 32 data words are transmitted to the EEPROM, the data word address will "roll over" and previous data will be overwritten.

ACKNOWLEDGE POLLING**:** Once the internally-timed write cycle has started and the EEPROM inputs are disabled, acknowledge polling can be initiated. This involves sending a start condition followed by the device address word. The read/write bit is representative of the operation desired. Only if the internal write cycle has completed will the EEPROM respond with a zero, allowing the read or write sequence to continue.

## Read Operations

Read operations are initiated the same way as write operations with the exception that the read/write select bit in the device address word is set to one. There are three read operations: current address read, random address read and sequential read.

CURRENT ADDRESS READ**:** The internal data word address counter maintains the last address accessed during the last read or write operation, incremented by one. This address stays valid between operations as long as the chip power is maintained. The address "roll over" during read is from the last byte of the last memory page, to the first byte of the first page. The address "roll over" during write is from the last byte of the current page to the first byte of the same page. Once the device address with the read/write select bit set to one is clocked in and

acknowledged by the EEPROM, the current address data word is serially clocked out. The microcontroller does not respond with an input zero but does generate a following stop condition.

RANDOM READ**:** A random read requires a "dummy" byte write sequence to load in the data word address. Once the device address word and data word address are clocked in and acknowledged by the EEPROM, the microcontroller must generate another start condition. The microcontroller now initiates a current address read by sending a device address with the read/write select bit high. The EEPROM acknowledges the device address and serially clocks out the data word. The microcontroller does not respond with a zero but does generate a following stop condition.

SEQUENTIAL READ**:** Sequential reads are initiated by either a current address read or a random address read. After the microcontroller receives a data word, it responds with an acknowledge. As long as the EEPROM receives an acknowledge, it will continue to increment the data word address and serially clock out sequential data words. When the memory address limit is reached, the data word address will "roll over" and the sequential read will continue. The sequential read operation is terminated when the microcontroller does not respond with a zero but does generate a following stop condition.

# 3.7 Max 485

## DESCRIPTION

The MAX485 is low-power transceivers for RS-485 and RS-422 communication. The IC contains one driver and one receiver. The driver slew rates of the MAX485 is not limited, allowing them to transmit up to 2.5Mbps. These transceivers draw between 120µA and 500µA of supply current when unloaded or fully loaded with disabled drivers. All parts operate from a single 5V supply. Drivers are short-circuit current limited and are protected against excessive power dissipation by thermal shutdown circuitry that places the driver outputs into a high-impedance state. The receiver input has a fail-safe feature that guarantees a logic-high output if the input is open circuit. The MAX485 is designed for half-duplex applications

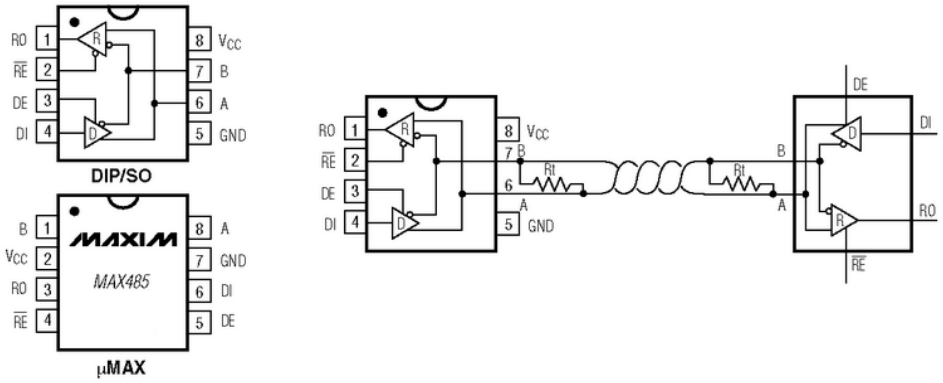



**Fig: 3.7.1 MAX 485 Pin Diagram and Connection**

# FEATURES

- Low Quiescent Current: 300μA
- -7V to +12V Common-Mode Input Voltage Range
- Three-State Outputs
- 30ns Propagation Delays, 5ns Skew
- Operate from a Single 5V Supply
- Allows up to 32 Transceivers on the Bus
- Data rate: 2.5 Mbps
- Current-Limiting and Thermal Shutdown for Driver Overload Protection
- The transmitter outputs and receiver inputs are protected to ±15kV Air ESD

# APPLICATION

- Low-Power RS-485 Transceivers
- Low-Power RS-422 Transceivers
- Level Translators
- Transceivers for EMI-Sensitive Applications
- Industrial-Control Local Area Networks

# PIN DESCRIPTION

| No. | Name | Function |
|---|---|---|
| **1** | **RO** | Receive output: if A > B by 200mV, RO will be high; if A < B by 200mV,<br>RO will be low. |
| **2** | **RE** | Receiver Output Enable. RO is enabled when RE is low; RO is high impedance when RE is high. |
| **3** | **DE** | Driver Output Enable. The driver outputs are enabled when DE is high.<br>They are high impedance when DE is low. If the driver outputs are enabled, the parts function as line drivers. While they are high impedance, they function as line receivers if RE is low. |
| **4** | **DI** | Driver input. A low on DI forces output A low and output B high. Similarly, a high on DI forces output A high and output B low. |
| **5** | **GND** | Ground |
| | | |

| | | |
|---|---|---|
| **6** | **A** | Driver Output and Receiver differential input. |
| **7** | **B** | Driver Output and Receiver differential input. |
| **8** | **VCC** | Positive Supply: 4.75V ≤ VCC ≤ 5.25V |

**Table 3.7.1 Pin Functions of MAX485**

# FUNCTIONAL DIAGRAM

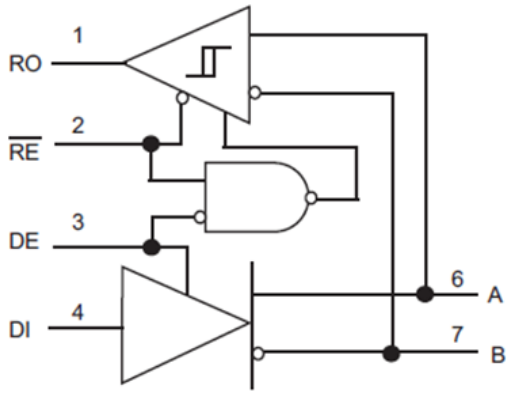



**Fig: 3.7.2 Functional Diagram of MAX485**

# CHAPTER 4
# Result and analysis

Data acquisition system may b wired or wireless. Our application demands a master slave system where in multiple slaves connected to master give information to the master. The system is a wired system so we go into the concept of wired communication protocols.

## 4.1 UART (RS232)

The basic protocol being UART (RS232), implementing UART following conclusions were derived.

- It allows only 1TX and 1RX on each line.
- It is a full duplex communication system speed of about 1Mb/s.
- It operates in single ended mode. Moreover it supports only the physical layer of OSI model. Communication between two 8051 boards and between the board and PC has been studied using UART.

The disadvantage being, it supports only 1 slave at a time and so cannot prove to be useful to our system.

## 4.2 SPI (Serial Peripheral Interface)

Then we go into the concept of SPI (Serial Peripheral Interface). This communication protocol works on 4 wires or pins namely MOSI, MISO, SCK and SS. MOSI- Master Output Slave Input, MISO- Master Input Slave Output, SCK-Serial Clock and SS stands for slave select.

- It supports only master slave communication.
- It supports synchronous mode of operation.
- Generally used for on PCB applications with a data rate of up to 3Mb/s to 10Mb/s.
- It is a full duplex protocol.

However for n slaves connected to the master it requires 3+n pins. This makes the wiring complex and also adds to the cost. Hence proves a bit disadvantageous to our system. Communication between two ECU's using SPI has been studied and implemented.

## 4.3 I2C (Inter Integrated Circuit bus)

The next is the I2C protocol.

- It supports multi master communication unlike SPI that supports only master slave communication.
- It supports both physical as well as data link layer of OSI model.

- Operates in synchronous mode and generally used for small distance communication i.e. on PCB applications having variable data rates of 100kb/s, 400kb/s and 3.4Mb/s witch could be set as per the intended application.
- It communicates on only two pins and can support up to 2^7 devices theoretically.
- It supports half duplex mode of communication.

Communication between ECU and various on board slave IC's like port-expander, ADC-DAC, EPROM etc. has been implemented and studied. Algorithms of I2C implementation on various salve ICs port-expander, ADC-DAC, EPROM are given below.

## 4.3.1 Port Expander Interfacing

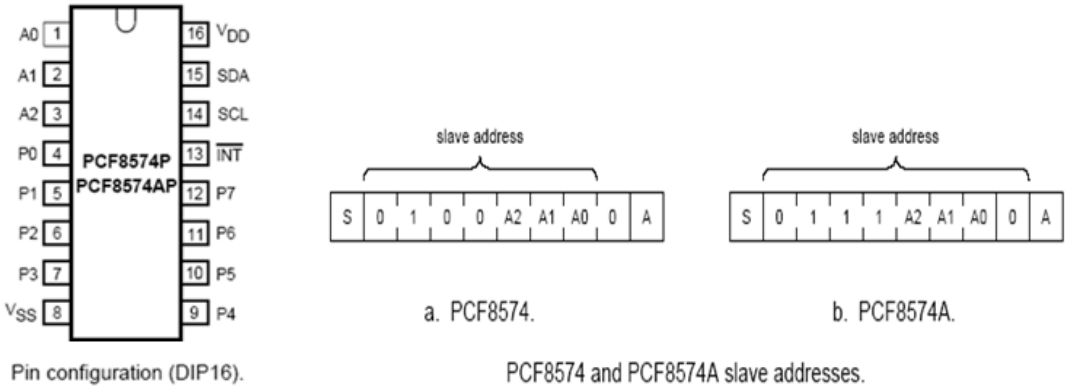



**Fig: 4.3.1 Port Expander Pin Diagram and Address Byte**

## Circuit Diagram

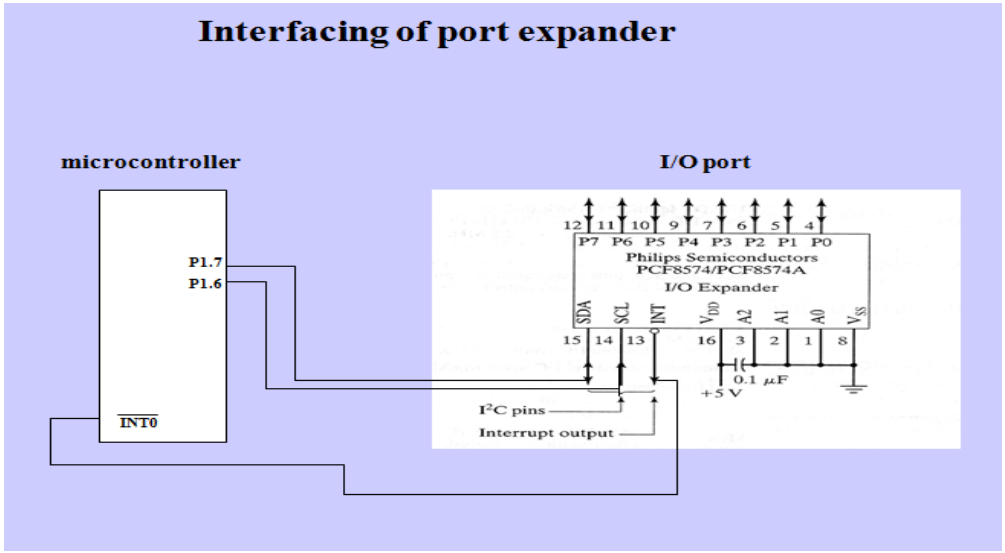
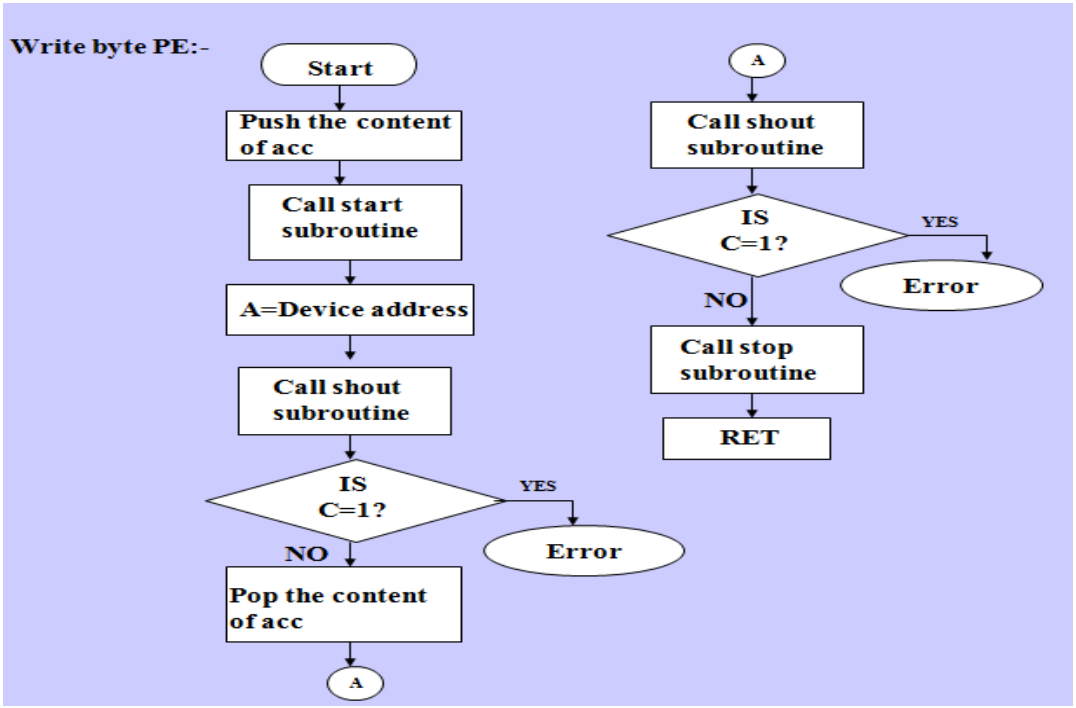
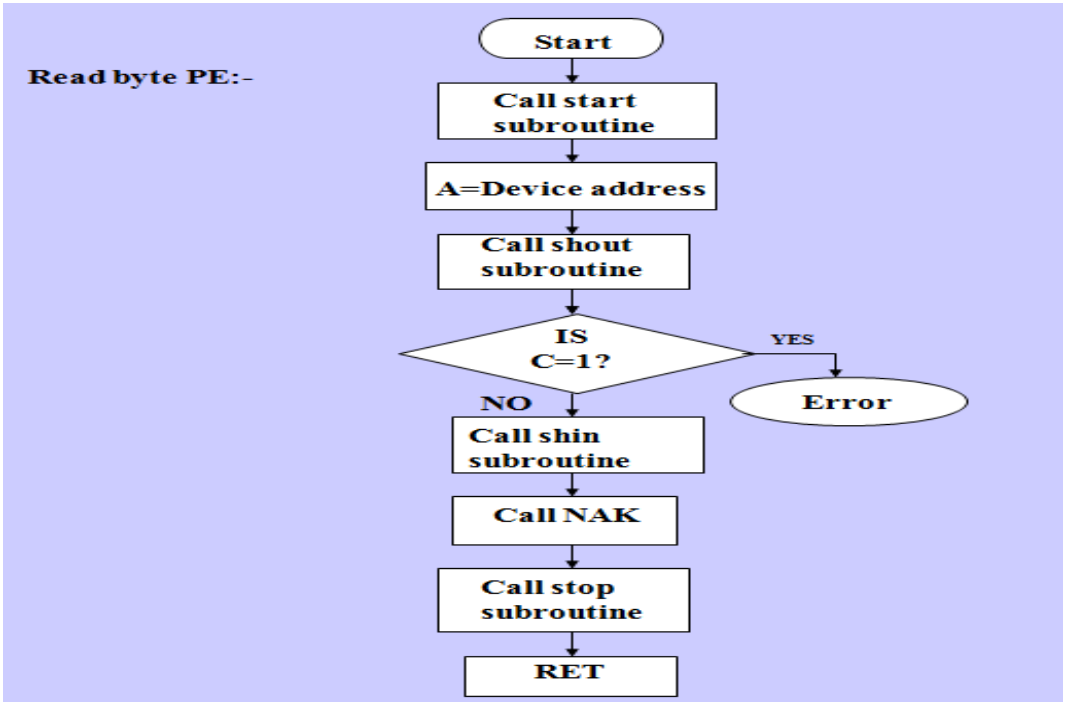



**Fig: 4.3.2 Prot Expander Interfacing Circuit Diagram**

# Procedural Algorithm



## 4.3.2 ADC-DAC Interfacing

## Circuit Diagram

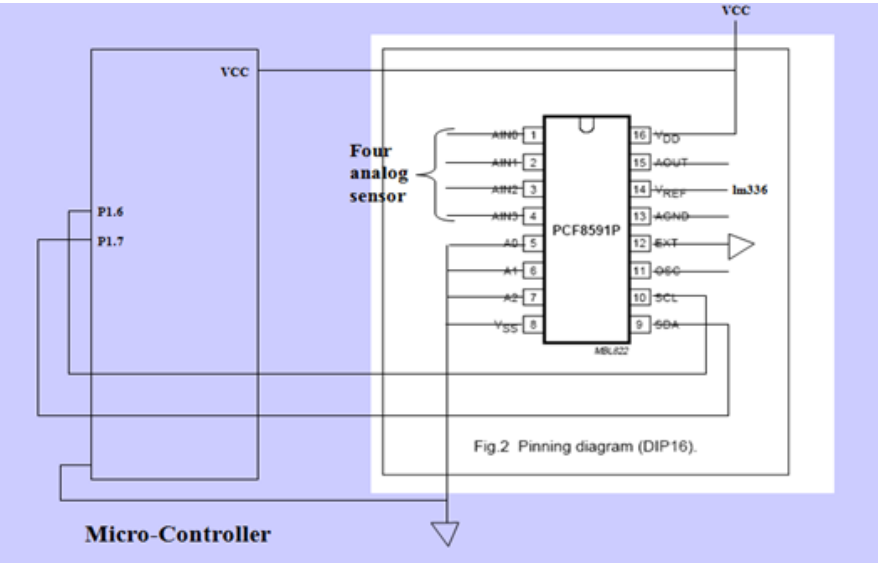
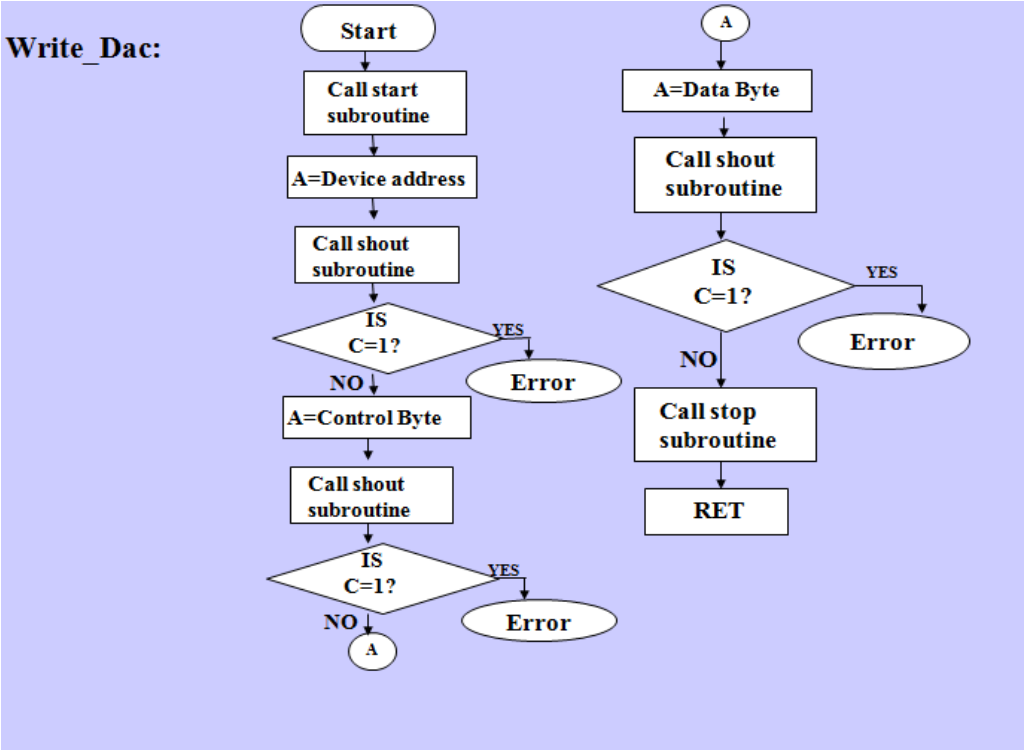



**Fig: 4.3.3 PCF8591 Interfacing Circuit Diagram**

## Procedural Algorithm



## 4.3.3 EEPROM Interfacing

## Circuit Diagram

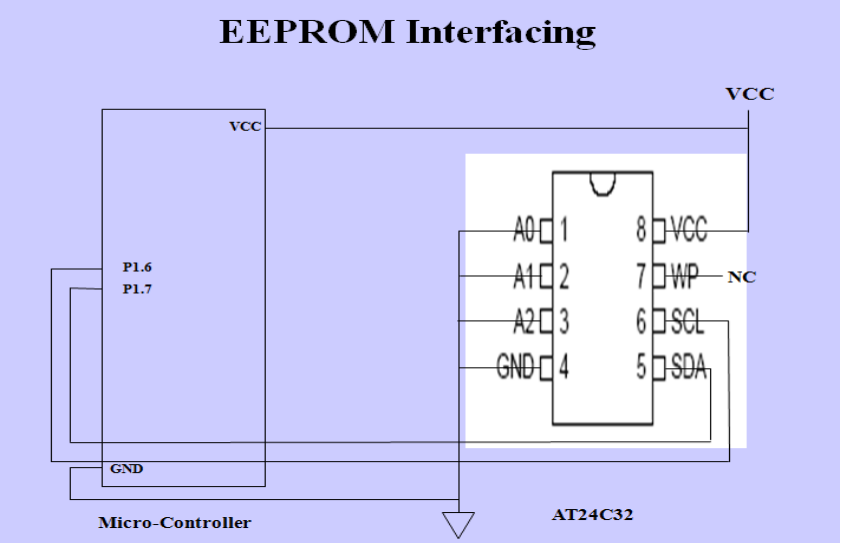
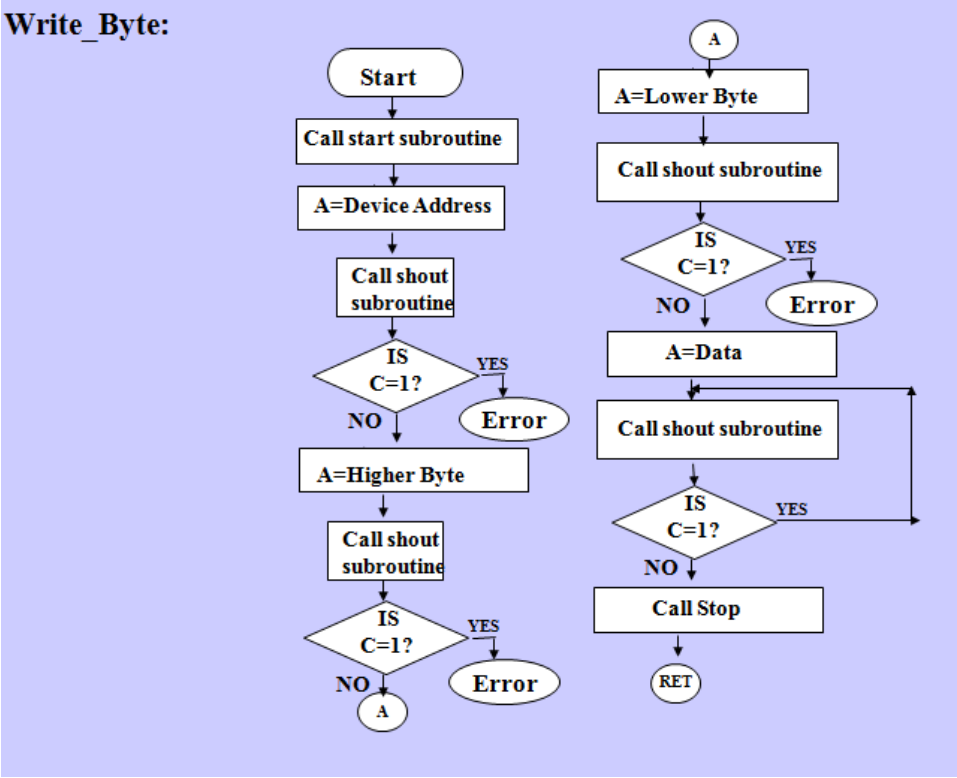
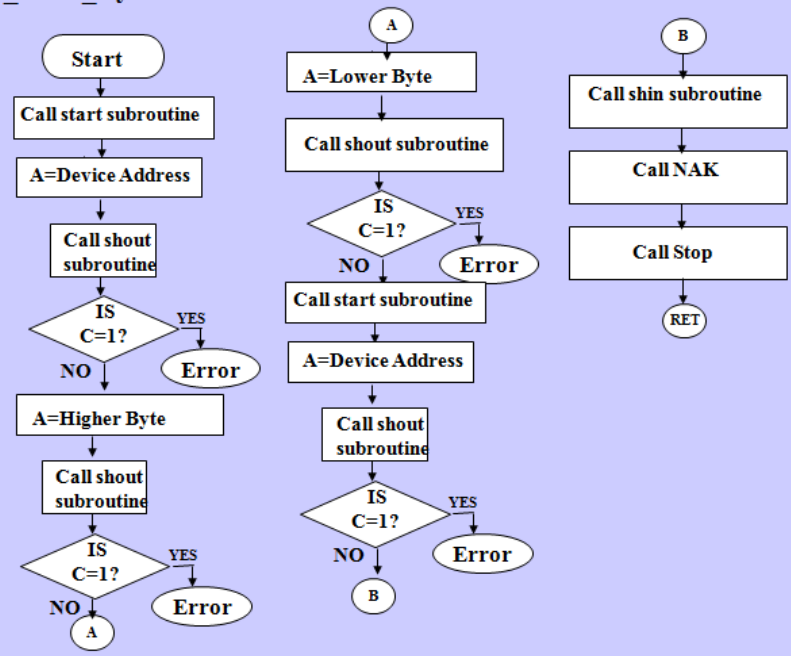



**Fig: 4.3.4 EEPROM Interfacing Circuit Diagram**

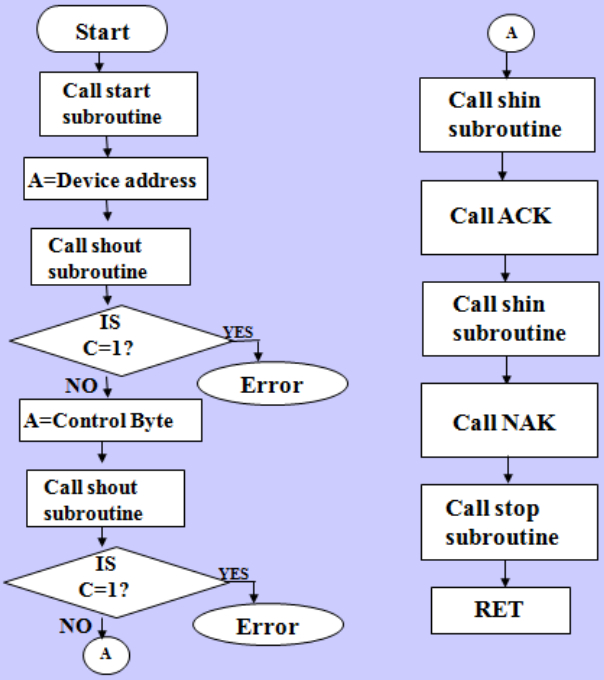
## Procedural Algorithm



**Random_Read_Byte:**




**Read_ADC0:**



# 4.4CAN (Control Area Network)

In order to overcome the disadvantages of above protocols CAN was studied.

- It supports multi master communication operates in asynchronous mode having both physical as well as data link layer.
- Control Area Network can operate over long distances, communication takes place over two wires with data rates of up to 40kb/s to 1Mb/s.[1]
- It has advantages of collision detection and error correction but at the same time has complex hardware and coding structure.

# 4.5 RS 485

Multi master communication is not at all a necessity in the required system. We require multiple slaves connected to the master unit. These slaves give the information to the master. This makes us go into the concepts of the advanced version of RS232 i.e. RS485.

- This protocol also communicates on two wires like CAN and I2C.
- It supports up to 32 slave units on line.
- Gives data rates of up to 10Mb/s and can also communicate over long distances in differential mode of operation unlike RS232 with uses single ended mode of operation.
- It also has the advantage of slave to slave communication where in we bypass the master unit.

We thus select RS485 for the required system. The system implementation will have one master and one slave unit. Communication between them will be demonstrated using RS485 protocol. The ECU used is AREDUINO ATMEGA 8. [2]

## RS485 Transceiver connections

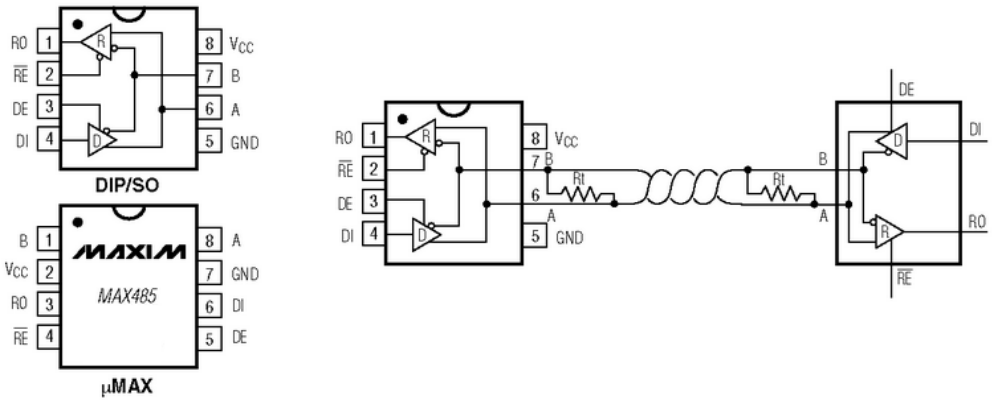



**Fig: 4.5.1 MAX485 Transceiver Connections**

# Circuit Diagram

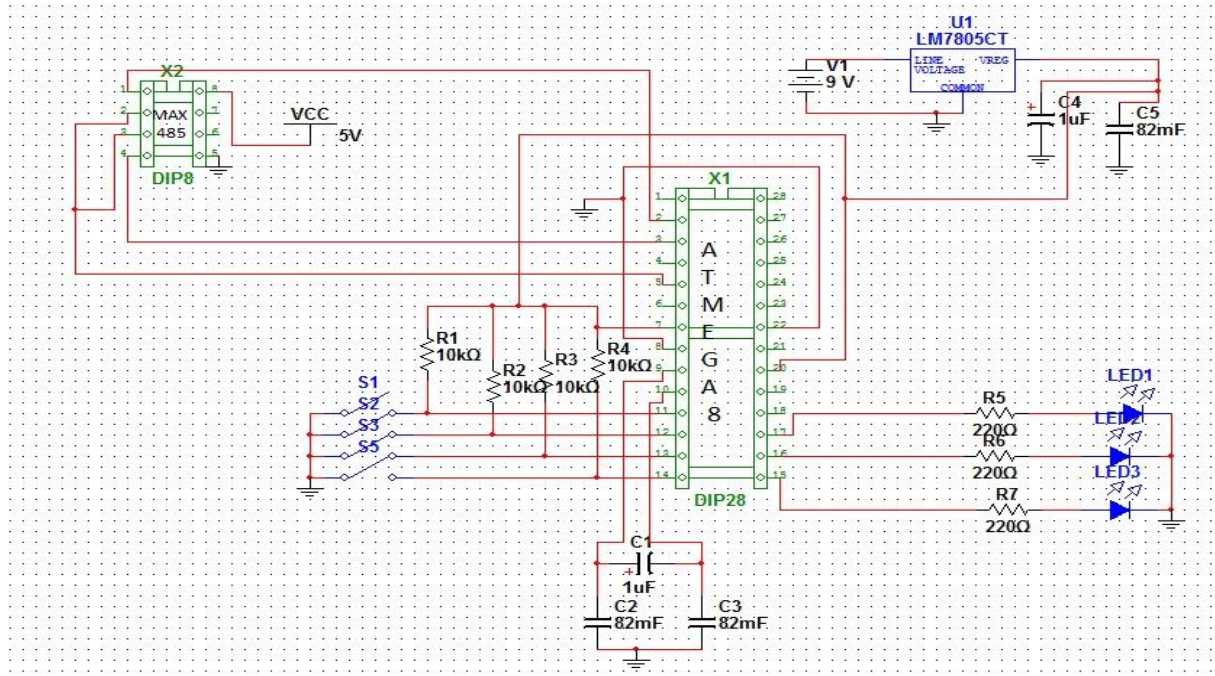



**Fig 4.5.2 Master Unit of RS485**

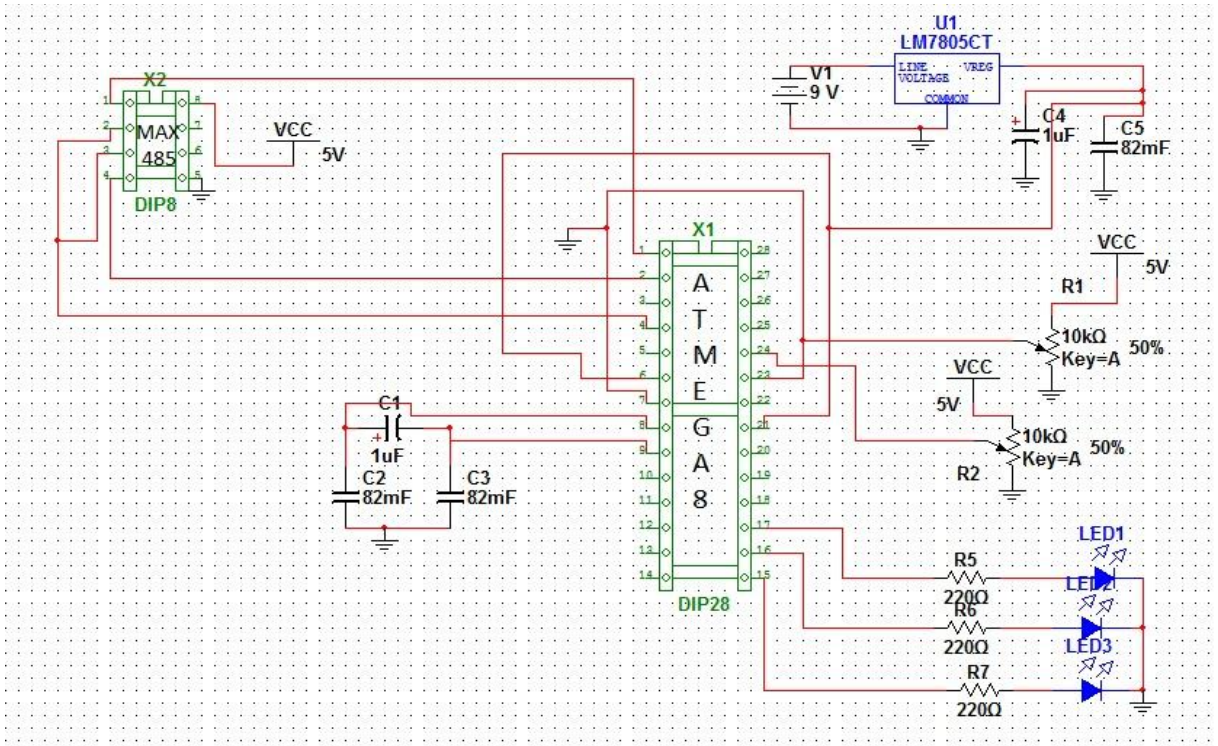



**Fig 4.5.3 Slave Unit of RS485**

# Future work

- Implementing multi master communication.
- Implementing CAN protocol.
- Interfacing multiple slaves on the same bus.
- Communicate between two slaves bypassing the master unit.

# Conclusion

Out of the protocols studied RS485 is best suited for the intended application of data acquisition system. Approximately 32 slaves could be monitored over the established network. These slaves report the master unit whenever they are asked to by the master. The network could be spread over about 1.2 km. The wiring of the system is not complex since all the units are connected over same bus in parallel. However this network supports only master slave communication, for more robust and multi-master communication network CAN protocol could be used.

# References


1. Serial port complete Second Edition: by Jan Axelson's Lakeview Research.
2. Trim-the-fat-off-RS-485-designs.EE Times.2000.
3. Adam Osborne, An introduction to microcomputers volume 1: Basic Concepts
4. Embedded Systems Architecture programming and design Second Edition- Raj Kamal.
5. Serial Peripheral interface by Neil's Log book.
6. Embedded Networking with CAN and CAN open by Olaf Pfeiffer.

# Appendix

- Datasheet  of 89v51rd2
- Datasheet of max 232
- Datasheet of max 485
- Datasheet of ATMEGA 8