Unit IV-PIC Interfacing-II

CCP Modules

- Capture and compare features allow the user to time and control different events. PIC18F has a built-in peripheral module called the CCP (Capture/Compare/PWM) module for this purpose.
- PIC18F has 0 to 5 CCP modules depending on the PIC family it belongs to. The CCP modules are designated as CCP1, CCP2, CCP3 and so on.
- In recent PIC microcontrollers, enhanced PWM functionality and auto-shutdown capability is introduced to produce Enhanced CCP (ECCP) module. This Enhanced CCP (ECCP) module allows better control of DC motor.
- Depending on the PIC18F family, there are 0-3 (ECCP) modules. Table shows the number of CCP and ECCP modules in PIC18F.

PIC18 Microcontroller	No. of CCP modules	No. of ECCP modules
PIC18F2220	2	0
PIC18F4220	1	1
PIC18F452/4520	1	1
PIC18F458/4580	1	1
PIC18F65J10	2	3

Table PIC18F CCP and ECCP modules

CCP and Timers

- Different timers are associated with the CCP module depending on the mode used. Table shows the PIC18 timers for different CCP modes.

CCP Modes	Timer
Capture	1 or 3
Compare	1 or 3
PWM	2

- T3CON register is used to select from Timer1 or Timer3 for the capture and compare modes.

CCP Registers

- Each CCP module has three registers :
  1. CCPxCON (8-bit) : CCP control register
  2. CCPRxH (8-bit) : CCPR high register
  3. CCPRxL (8-bit) : CCPR low register

CCPxCON (8-bit) : CCP Control Register

- shows the CCP control registers for CCP1 module.

-	-	DC1B1	DC1B0	CCP1M3	CCP1M2	CCP1M1	CCP1M0

bit 7 bit 0

bit 7 - 6 Unimplemented : Read as '0'

bit 5 - 4 DCxB1 : DCxB0 : PWM Cycle bit 1 and bit 0

Capture mode :

Unused.

Compare mode :

Unused

PWM code :

These bits are the two LSbs (bit 1 and bit 0) of the 10-bit PWM duty cycle. The upper eight bits (DCxp : DCx2) of the duty cycle are found in CCPRxL

bit 3 - 0 **CCPxM3 : CCPxM0 :** CCPx Mode Select bits

0000 = Capture /Compare /PWM off (resets CCPx module)

0001 = Reversed

0010 = Compare mode, toggle output on match (CCPxIF) bit is set)

0011 = Compare mode, CAN message received (CCP1 only)

0100 = Compare mode, every falling edge

0101 = Compare mode, every rising edge

0110 = Capture mode, every 4^{th} rising edge

0111 = Capture mode, every 16^{th} rising edge

1000 = Compare mode, initialize CCP pin low, on compare match force CCP pin high (CCPIF bit is set)

1001 = Compare mode, initialize CCP pin high, on compare match force CCP pin low(CCPIF bit is set)

1010 = Compare mode, CCP pin is unaffected (CCPIF bit is set)

1011 = Compare mode, trigger event (CCP1IF bit is set ; CCP resets TMR1 or TMR3 and starts an A/D conversion if the A/D module is enabled)

11XX = PWM mode

CCPR High and Low Register

- CCPRxH and CCPRxL form the high byte and the low byte of the 16-bit register. Fig. shows CCPR1 high and low registers. This 16-bit register that can operate as a 16-bit capture register, as a 16-bit compare register or as a PWM Duty Cycle register depending upon the mode selected.

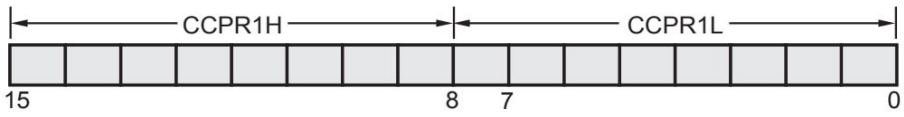
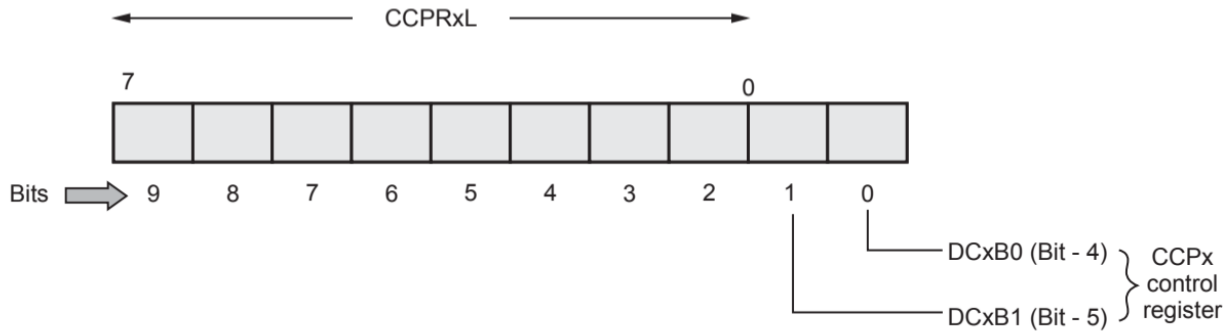


Fig. CCPR1 high and low register

PWM 10-bit Duty Cycle Register

- In PWM mode, 10-bit register is used to decide the duty cycle. This 10-bit is formed using :
  - **(Bit 1 and Bit 0)** lower 2-bits : **DCxB1:DCxB0 :** from CCPx Control Register
  - **(Bit 9 - Bit 2)** upper 8-bits : **CCPRxL register** : Bit 7 - Bit 0



PWM 10-bit duty cycle register

Capture Mode

- This mode provides access to the current state of that timer TMR1 or TMR3 register.
- When capture mode with the bit selection in the CCP1CON register is selected, CCPR1H : CCPR1L captures the 16-bit value of the TMR1 or TMR3 register when an event occurs on pin RC2/CCP1.

- The combination of the four bits (CCP1M3 - CCP1M0) of the control register determines the event :
  - CCP1M3-CCP1M0 : 0100 : Every falling edge
  - CCP1M3-CCP1M0 : 0101 : Every rising edge
  - CCP1M3-CCP1M0 : 0110 : Every 4^{th} rising edge
  - CCP1M3-CCP1M0 : 0111 : Every 16^{th} rising edge
- When a capture is made, the interrupt request flag bit, CCP1IF (PIR registers), is set. It must be cleared in software.

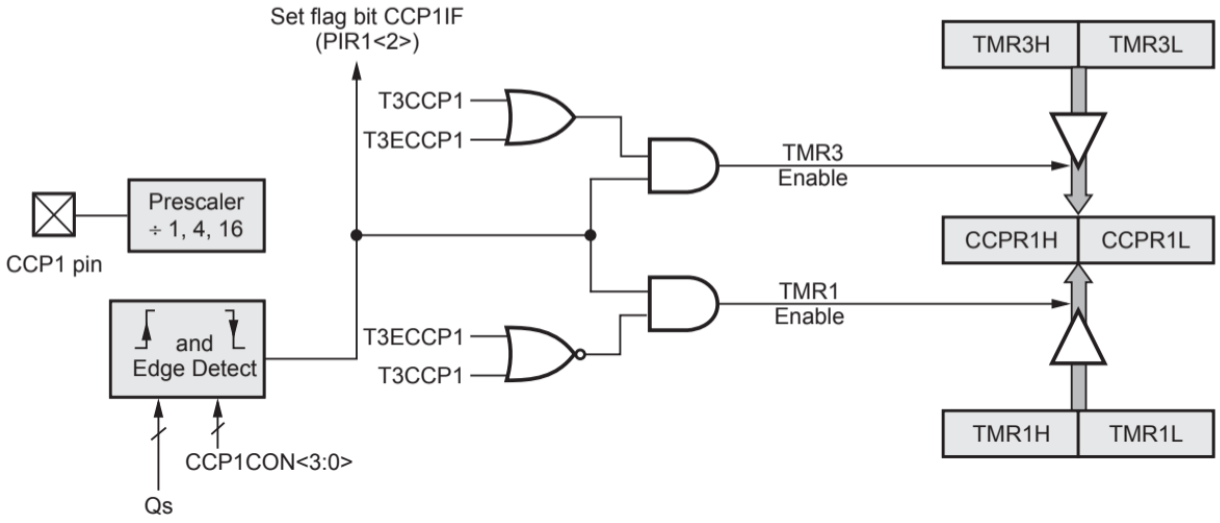
- If another capture occurs before the value in register CCPR1 is read, the old captured value will be lost.
- In capture mode, the RC2/CCP1 pin should be configured as an input.
- The timers used with the capture feature (either Timer1 and/or Timer3) must be running in Timer mode or Synchronized Counter mode. In Asynchronous Counter mode, the capture operation may not work.
- The timer used with each CCP module is selected in the T3CON register.
- Fig. 8.4.1 shows the block diagram of capture mode operation.

bit 6, 3 **T3ECCP1 : T3CCP1 :** Timer3 and Timer1 to CCP1/ECCP1 enable bits

1x = Timer3 is the clock source for compare/capture CCP1 and ECCP1 modules

01 = Timer3 is the clock source for compare/capture ECCP1, Timer1 is the clock source for compare/capture CCP1

00 = Timer1 is the clock source for compare/capture CCP1 and ECC1 modules



Steps for Programming Capture Mode

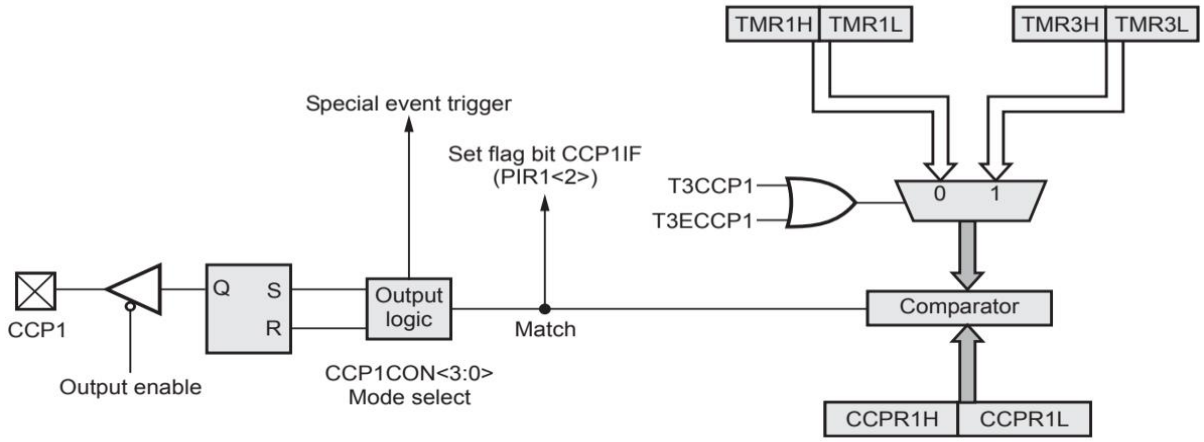
Initialization of PIC18F458 Capture Mode

1. Initialize the CCPCON1 register for capturing the rising or falling edge.
2. Configure the CCP1 pin as an input pin.
3. Configure T3CON register to select either Timer0 or Timer3.
4. Initialize TxCON register to either configure Timer1 or Timer3 depending on step 3.
5. Set CCPR1 and TMR3/TMR1 register to 0.
6. Enable CCP1 interrupt enable flag : CCP1IE = 1.
7. Clear CCP1 interrupt flag : CCP1IF = 0.
8. Wait for interrupt flag. If it is set, read captured value.

Note *'x' is for 1 or 3.*

Compare Mode

- CCP in compare mode is used to generate a waveform of various duty cycles like PWM and also used to trigger an event when the pre-determined time expires. It is also used to generate specific time delay.
- In compare mode, the 16-bit CCPR1 and ECCPR1 register value is constantly compared against either the TMR1 register pair value or the TMR3 register pair value. When a match occurs, the CCP1 pin can have one of the following actions :
  - Driven high
  - Driven low
  - Toggle output (High-to-low or low-to-high)
  - Remains unchanged.
- **CCP1M3:CCP1M0 :** CCP1 module mode select bit. These bits decide which action takes on compare match. At the same time, interrupt flag bit CCP1IF is set.
  - 0010 = Toggle output on match
  - 1000 = Initialize CCP1 pin low, on compare match this pin is set to high.
  - 1001 = Initialize CCP1 pin high, on compare match this pin is set low.
  - 1010 = On compare match generates software interrupt.
  - 1011 = On compare match trigger special event, reset the timer, start ADC conversion.

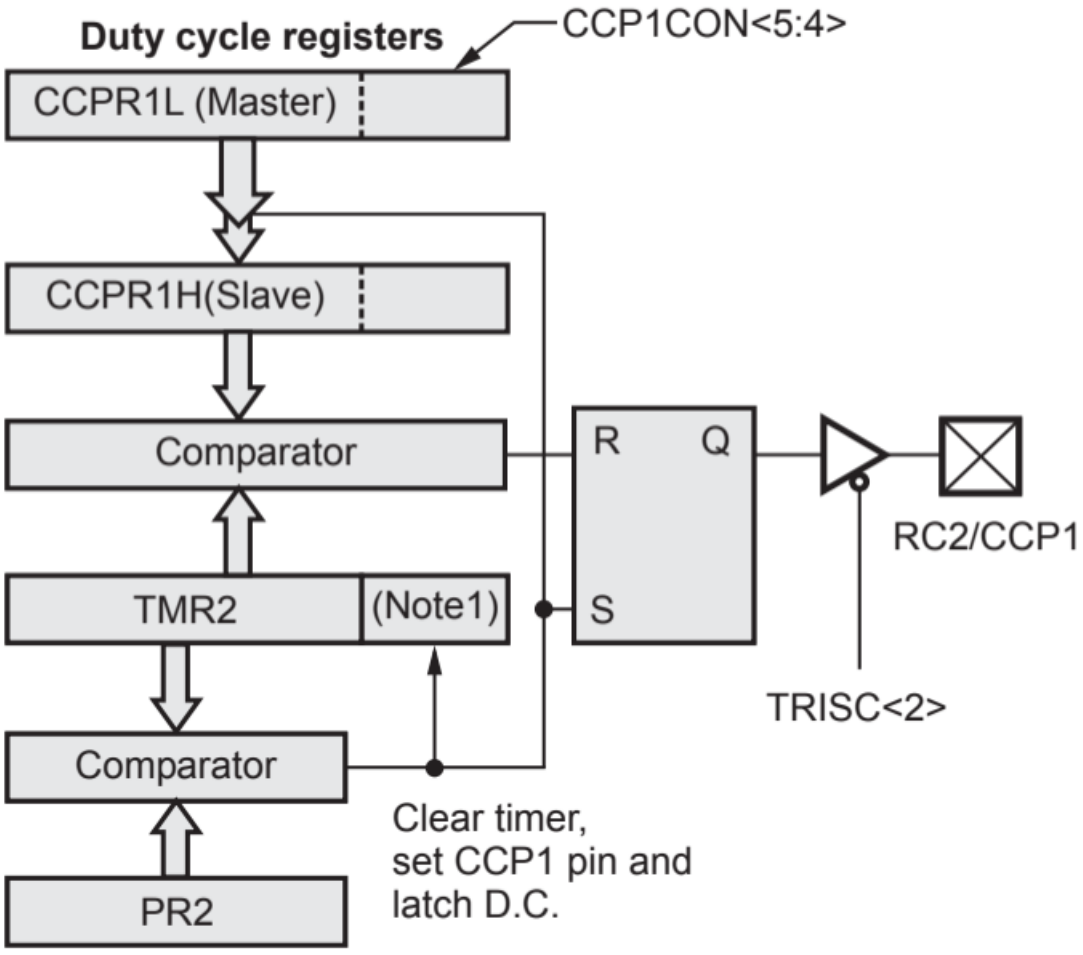


Steps for Programming

1. Set the CCP1 pin as an output.
2. Configure T3CON Register for Timer1 or Timer 3
3. If Timer1 is used then configure T1CON Register also.
4. Configure CCP1CON Register for compare mode.
5. Load desired count in CCPR1 (CCPR1H: CCPR1L) Register.
6. Initialize TMR1 or TMR3 register value.
7. Start Timer
8. Wait for CCP1IF (PIR1<2>) interrupt flag to set.

PWM Mode

- Pulse Width Modulation (PWM) is a digital signal which is most commonly used in control circuitry. This signal is set high and low) in a predefined time and speed. The time during which the signal stays high is called the "**on time**" and the time during which the signal stays low is called the "**off time**".
- **Duty cycle of the PWM :** The percentage of time in which the PWM signal remains HIGH (on time) is called as **duty cycle**.
- **Frequency of a PWM :** The frequency of a PWM signal determines how fast a PWM completes one period. One Period is complete ON and OFF of a PWM signal.
- The PIC18F when configured in Pulse-Width Modulation (PWM) mode, the CCP1 pin produces up to a 10-bit resolution PWM output. This means that for a value of 0 there will be a duty cycle of 0 % and for a value of 1024 (2^{10}) there be a duty cycle of 100 %.
- Since the CCP1 pin is multiplexed with the PORTC data latch, the TRISC<2> bit must be cleared to make the CCP1 pin an output.



PWM Programming steps

- Following steps are needed to setup the PWM module in PIC18F458.
  1. Set the PWM period by writing to the PR2 register.
  2. Set the PWM duty cycle by writing to the CCPR1L register and CCP1CON<5:4> bits.
  3. Configure the CCP1CON module for PWM operation.
  4. Make the CCP1 pin an output by clearing the TRISC<2> bit.
  5. Clear Timer2 register
  6. Set the TMR2 prescale value
  7. Enable Timer2 by writing to T2CON.

1. Set the PWM Period

- The PWM period is set by writing the PR2 register. So the value can be calculated by using this formula.i.e,

$$\text{PWM period} = [(PR2) + 1] \times 4 \times T_{osc} \times N$$

where T_{osc} is the $1/ F_{osc}$

N : TMR2 prescale value. Can be set to 1, 4 or 16 by programming Timer2 control register (T2CON)

$$\therefore \quad PR2 = \frac{\text{PWM period}}{T_{osc} \times 4 \times N} - 1 = \frac{F_{osc}}{\text{PWM frequency} \times 4 \times N} - 1$$

- For example, if we use a 20 MHz clock, the O/P frequency is 2 kHz and N = 4 then

$$PR2 = \frac{20\ \text{MHz}}{2\ \text{kHz} \times 4 \times 4} - 1 = 624$$

2. Set PWM Duty Cycle

- The PWM duty cycle is specified by writing to the CCPRxL register and to the CCPxCON<5:4> bits. Up to 10-bit resolution is available. The CCPRxL contains the eight MSbs and the CCPxCON<5:4> bits contain the two LSbs. This 10-bit value is represented by CCPRxL: CCPxCON<5:4>. The duty cycle will be in the range of 0 to 1024.
- Now, let us see how to set a value for the CCPR1L which decides the duty cycle of a pulse. We know that a duty cycle is some % of the PR2 (period) register. For example, if PR2 is 201 then a 25 % duty cycle of the 201 is given by,

$$(CCPR1L : CCP1CON < 5 : 4 >) = PR2 \times \left(\frac{\text{Duty Cycle}}{100}\right)$$

$$(CCPR1L: CCP1CON < 5 : 4 >) = (201) \times (25/100)$$

$$(CCPR1L: CCP1CON < 5 : 4 >) = 50.25$$

- Here, hexademical equivalent of the integer part of the result i.e., 50 is loaded as 8 most significant bits in CCPR1L and 2 LSB bits in CCP1CON <5:4> are used to represent fraction portion of the result, i.e. 0.25.
- The decimal point portion of the duty cycle count is set

CCP1CON <5:4>		Decimal Point
DC1B2	DC1B1	
0	0	0
0	1	0.25
1	0	0.5
1	1	0.75

Representation decimal point portion

CCPR1L = 50 = 0b00110010 = 0x32

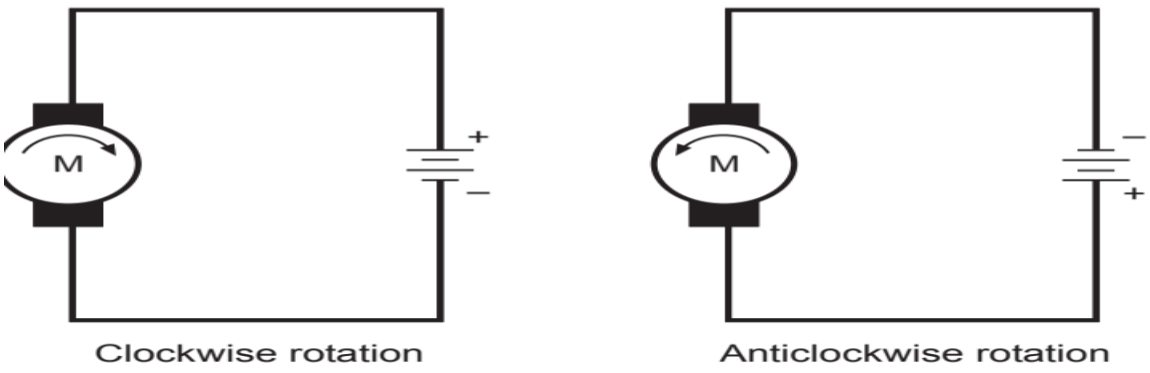
CCP1CON <5:4> = DC1B2 : DC1B1 = 0b01

DC Motor Speed Control with CCP

- Unlike stepper motor, the DC (direct current) motor rotates continuously. It has two terminals positive and negative. Connecting DC power supply to these terminals rotates motor in one direct and reversing the polarity of the power supply reverses the direction of rotation. The speed of the DC motor is measured in Revolutions Per Minute (RPM).
- The speed of the DC motor increases with increase in the supply voltage. However, we can not exceed supply voltage beyond the rated voltage. The speed of the DC motor also depends on the load. At no-load speed is highest. As we increase the load, the speed decreases. The overloading the DC motor can damage it because of excessive heat generated due to high current consumption.

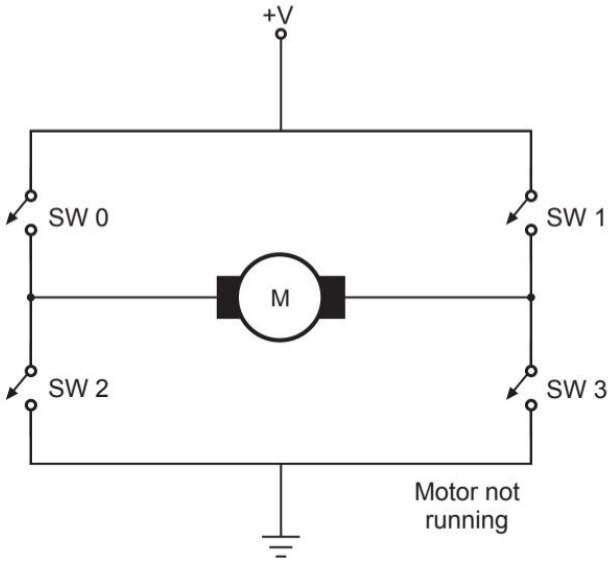
Direction control

- dc motor changes its direction of rotation when the polarity of power supply reverses.



Clockwise rotation Anticlockwise rotation

- By using switches for changing the power supply polarity we can control the direction of the rotation of the DC motor.



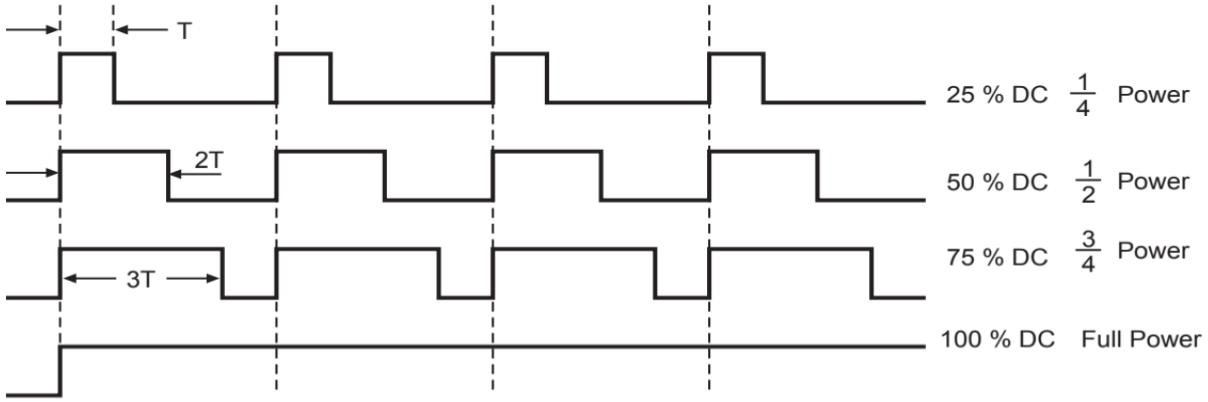
SW3	SW2	SW1	SW0	Motor rotation
ON	OFF	OFF	ON	Clockwise
OFF	ON	ON	OFF	Anticlockwise

Table 8.7.1 Switch configurations

- Any other switch configurations, for example, SW0 and SW2 ON is invalid, because it creates short circuit of the power supply.

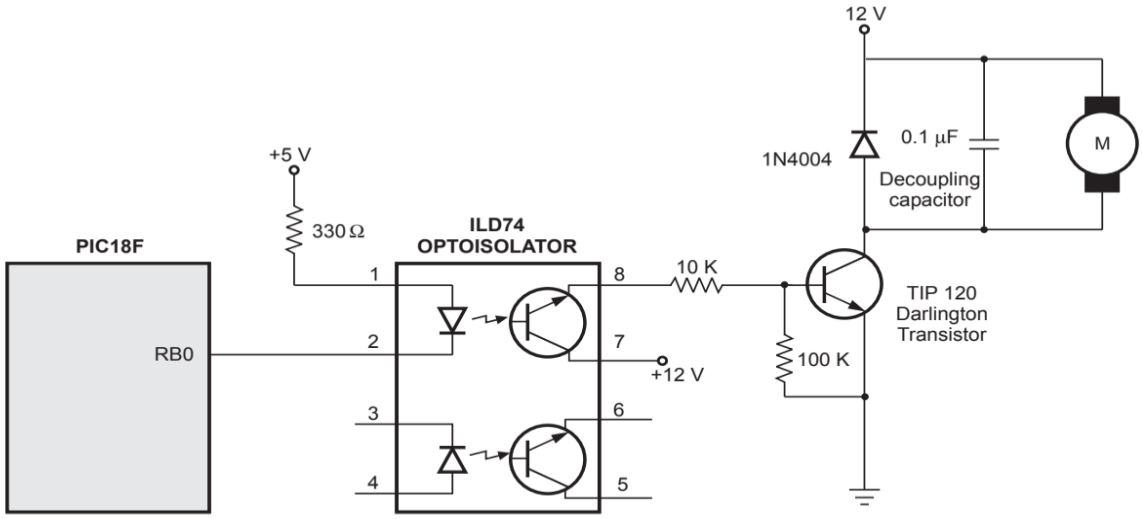
Pulse Width Modulation (PWM)

We know that the speed of the dc motor depends on the applied voltage. The average applied DC voltage and hence the power can be varied using technique called **pulse width modulation.** In this technique, the dc power supply is not a voltage of fixed amplitude; however it is a pulsating DC voltage. By changing pulse width we can change the applied power.

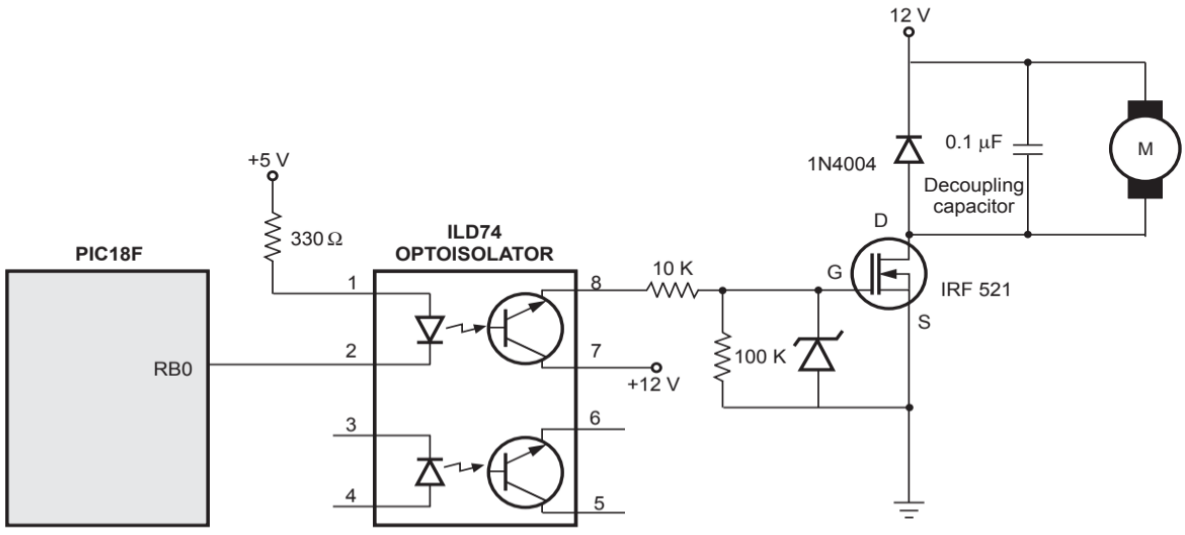


DC Motor Control with Optoisolator

- Fig. shows the dc motor control using a bipolar transistor and MOSFET as a switch. The circuit uses optoisolator which allows the motor and PIC18F to use separate power supplies.
- The circuit shown in the Fig. is also protected from EMI interference produced due to motor brushes. To protect the circuit from EMI interference the decoupling capacitor is connected across the motor.
- As shown in Fig. the port pin RB0 is used to control the switching of DC motor. When RB0 is low, motor is switched ON and when RB0 is high motor is switched OFF.
- In MOSFET circuit, the zener diode keeps the gate voltage below the rated maximum gate voltage for MOSFET.



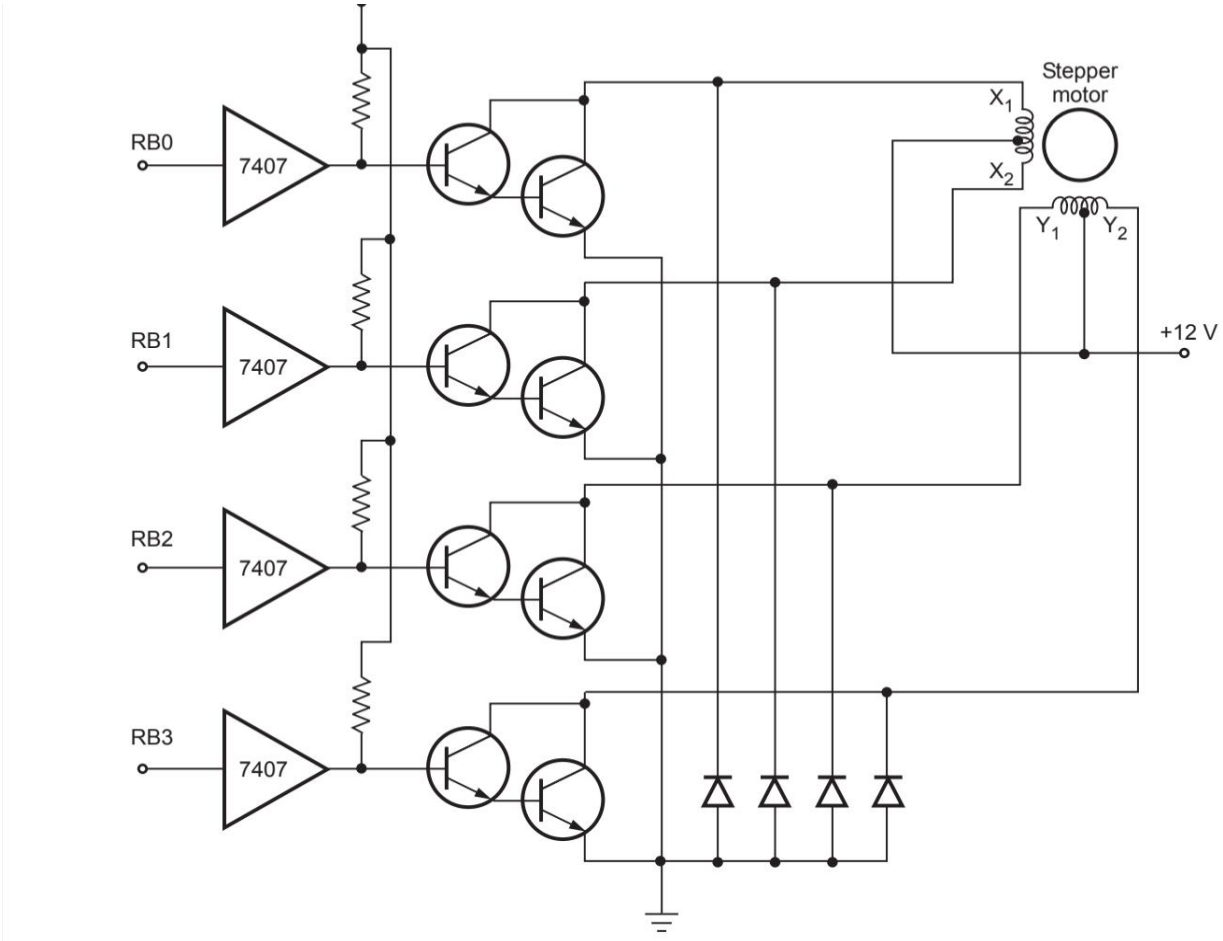
DC motor connection using a bipolar transistor



DC motor connection using a MOSFET

Stepper Motor Interfacing with PIC

- A stepper motor is a digital motor. It can be driven by digital signal. Motor shown in the circuit has two phases, with center-tap winding.
- The center taps of these windings are connected to the 12 V supply. Due to this, motor can be excited by grounding four terminals of the two windings. Motor can be rotated in steps by giving proper excitation sequence to these windings.
- The lower nibble of Port B of the PIC18F is used to generate excitation signals in the proper sequence.
- Particular winding is excited by making corresponding Port pin low. For example, winding X1 can be excited by making RB0 low.
- The given excitation sequence rotates the motor in clockwise direction. To rotate motor in anticlockwise direction we have to excite motor in a reverse sequence.



Step	Y_2	Y_1	X_2	X_1	RB3	RB2	RB1	RB0	CODE
1	1	0	1	0	0	1	0	1	05H
2	1	0	0	1	0	1	1	0	06H
3	0	1	0	1	1	0	1	0	0AH
4	0	1	1	0	1	0	0	1	09H
5	1	0	1	0	0	1	0	1	05H

Basics of Serial communication

- In serial data communication, during transmission parallel data is converted into serial bits using a parallel-in serial-out (PISO) shift register. At the receiver, the serial bits are connected into parallel data by another shift register called serial-in parallel-out (SIPO) shift register.

Classification

- Serial data transmission can be classified on the basis of how transmission occurs.
  1. Simplex 2. Half duplex 3. Full duplex

Simplex

- In simplex, the hardware exists such that data transfer takes place only in one direction. There is no possibility of data transfer in the other direction. A typical example is transmission from a computer to the printer.

Half Duplex

- The half duplex transmission allows the data transfer in both directions, but not simultaneously. A typical example is a walkie-talkie.

Full Duplex

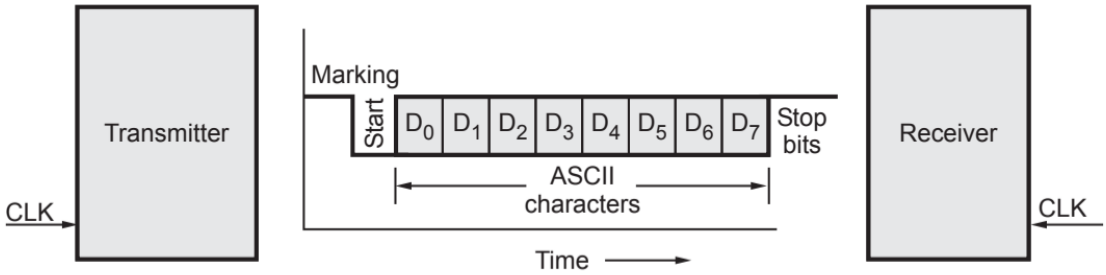
- The full duplex transmission allows the data transfer in both direction simultaneously. The typical example is transmission through telephone lines.

Types of Serial Data Communication

- Serial data communication uses two types of communications
  - Asynchronous serial data communication
  - Synchronous serial data communication

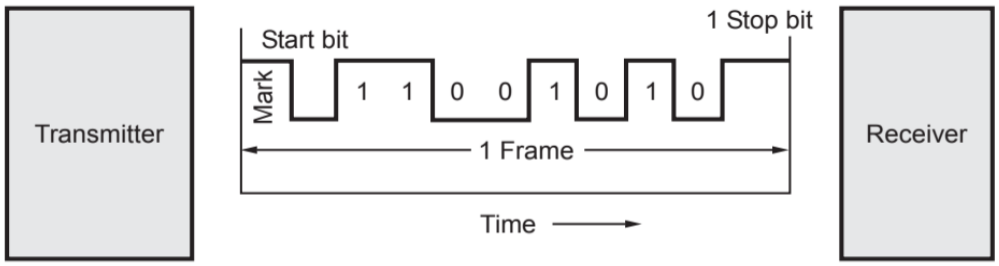
Asynchronous Serial Data Communication

- Fig. shows the transmission format for asynchronous transmission. Asynchronous formats are character oriented. In this, the bits of a character or data word are sent at a constant rate, but characters can come at any rate (asynchronously) as long as they do not overlap. When no characters are being sent, a line stays high at logic 1 called **mark**, logic 0 is called **space**.



Transmission format for asynchronous transmission

- The beginning of a character is indicated by a start bit which is always low. This is used to synchronize the transmitter and receiver. After the start bit, the data bits are sent with least significant bit first, followed by one or more stop bits (active high). The stop bits indicate the end of character. Different systems use 1, 1 1/2 or 2 stop bits. The combination of start bit, character and stop bits is known as **frame.**
- The start and stop bits carry no information, but are required because of the asynchronous nature of data. Fig. illustrates how the data byte CAH would look when transmitted in the asynchronous serial format.

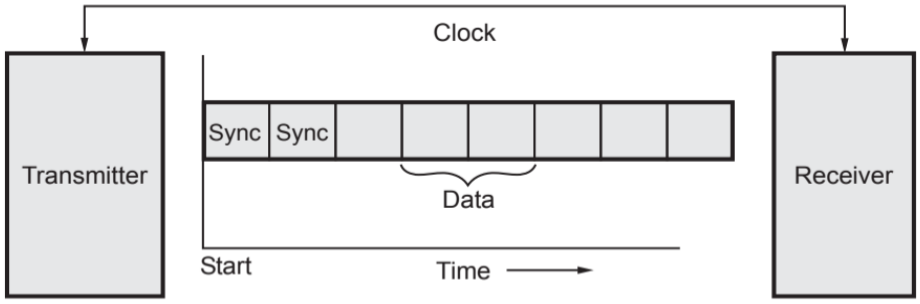


Asynchronous format with data byte CAH

- The data rate can be expressed as bits/sec or characters/sec. The term bits/sec is also called the **baud rate**. The asynchronous format is generally used in low-speed transmission (less than 20 kbits/sec).

Synchronous Serial Data Communication

- The start and stop bits in each frame of asynchronous format represents wasted overhead bytes that reduce the overall character rate. These start and stop bits can be eliminated by synchronizing receiver and transmitter.
- They can be synchronized by having a common clock signal. Such a communication is called **synchronous serial communication**. The Fig. shows the transmission format of synchronous serial communication. In this transmission synchronous bits are inserted instead of start and stop bits.



Synchronous transmission format

Sr. No.	Asynchronous serial communication	Synchronous serial communication
1.	Transmitters and receivers are not synchronized by clock.	Transmitters and receivers are synchronized by clock.
2.	Bits of data are transmitted at constant rate.	Data bits are transmitted with synchronization of clock.
3.	Character may arrive at any rate at receiver.	Character is received at constant rate.
4.	Data transfer is character oriented.	Data transfer takes place in blocks.
5.	Start and stop bits are required to establish communication of each character.	Start and stop bits are not required to establish communication of each character; however, synchronization bits are required to transfer the data block.
6.	Used in low-speed transmissions at about speed less than 20 kbits/sec.	Used in high-speed transmissions.

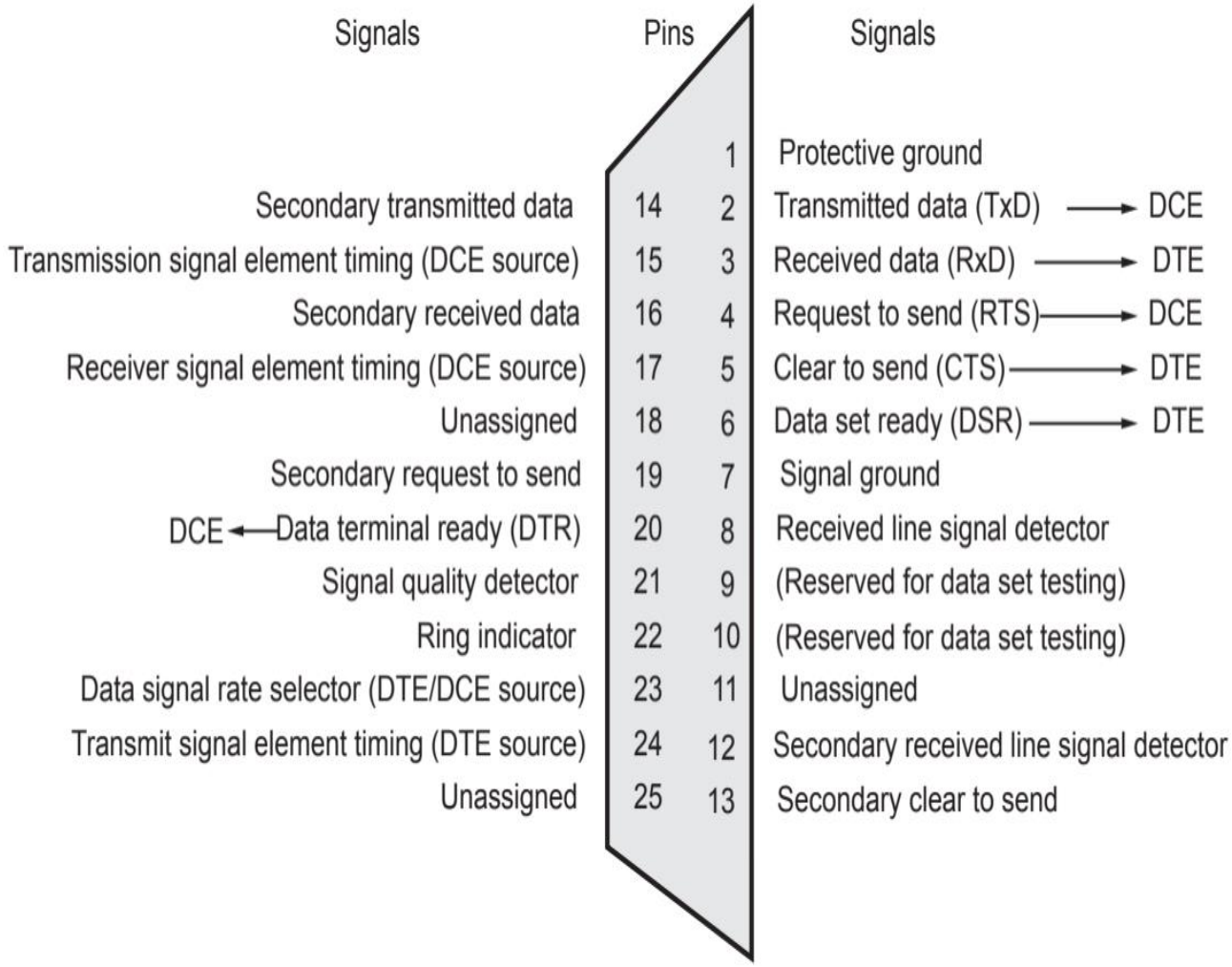
Comparison between asynchronous and synchronous serial data transfer

Baud Rate

- Baud rate is data transmission speed. The rate at which the bits are transmitted (bits/second) is called **baud** or **transfer rate**. The baud rate is the reciprocal of the time to send 1-bit. The common baud rates are multiples of 75 bits per second (bit/s) are typically : 75, 150, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400 bit/s

Study of RS 232

- In response to the need for signals and handshake standards between DTE and DCE, the Electronic Industries Association (EIA) introduced EIA standard RS 232 in 1962. It was revised and named as RS 232C, in 1969 by EIA. It is widely accepted for single ended data transmission over short distances with low data rates.
- This standard describes the functions of 25 signal and handshake pins for serial data transfer. It also describes the voltage levels, impedance levels, rise and fall times, maximum bit rate, and maximum capacitance for these signal lines. RS 232C specifies 25 signal pins and it specifies that the DTE connector should be male, and the DCE connector should be a female. The most commonly used connector should be a female. The most commonly used connector, DB-25P is shown in the



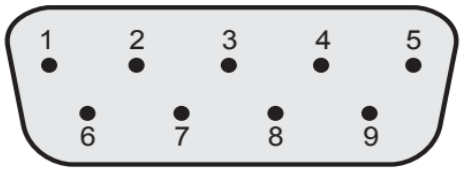
DB-25 P Connector

- The Table shows pins and signals description for RS 232C DB-25P connector.

Pin number	Common name	RS 232C name	Description	Signal direction on DEC
1		AA	Protective ground	
2	TxD	BA	Transmitted data	IN
3	RxD	BB	Received data	OUT
4	$\overline{\text{RTS}}$	CA	Request to send	IN
5	$\overline{\text{CTS}}$	CB	Clear to send	OUT
6		CC	Data set ready	OUT
7	GND	AB	Signal ground (common return)	
8	$\overline{\text{CD}}$	CF	Received line signal detector	OUT
9			(Reserved for data set testing)	
10			(Reserved for data set testing)	
11			Unsigned	
12		SCF	Secondary recd. line sig. detector	OUT
13		SCB	Secondary clear to send	OUT
14		SBA	Secondary transmitted data	IN
15		DB	Transmission signal element timing(DCE source)	OUT
16		SBB	Secondary received data	OUT
17		DD	Receiver signal element timing(DCE source)	OUT
18			Unassigned	
19		SCA	Secondary request to send	IN
20	$\overline{\text{DTR}}$	CD	Data terminal ready	IN
21		CG	Signal quality detector	OUT
22		CE	Ring indicator	OUT
23		CH/CI	Data signal rate selector (DTE/DCE source)	IN/OUT
24		DA	Transmit signal element timing (DTE source)	IN
25			Unassigned	

DB-9P Connector

- Since not all the pins are used in PC cables, IBM introduced the DB-9P version of the serial I/O standard, which uses only 9-pins.



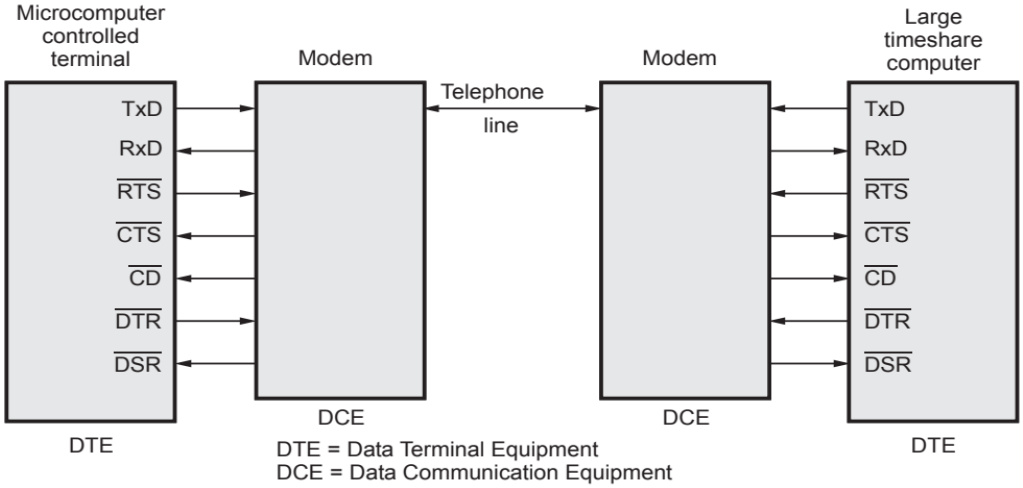
DB-9P connector

- The Table describes the pins for DB-9P connector.

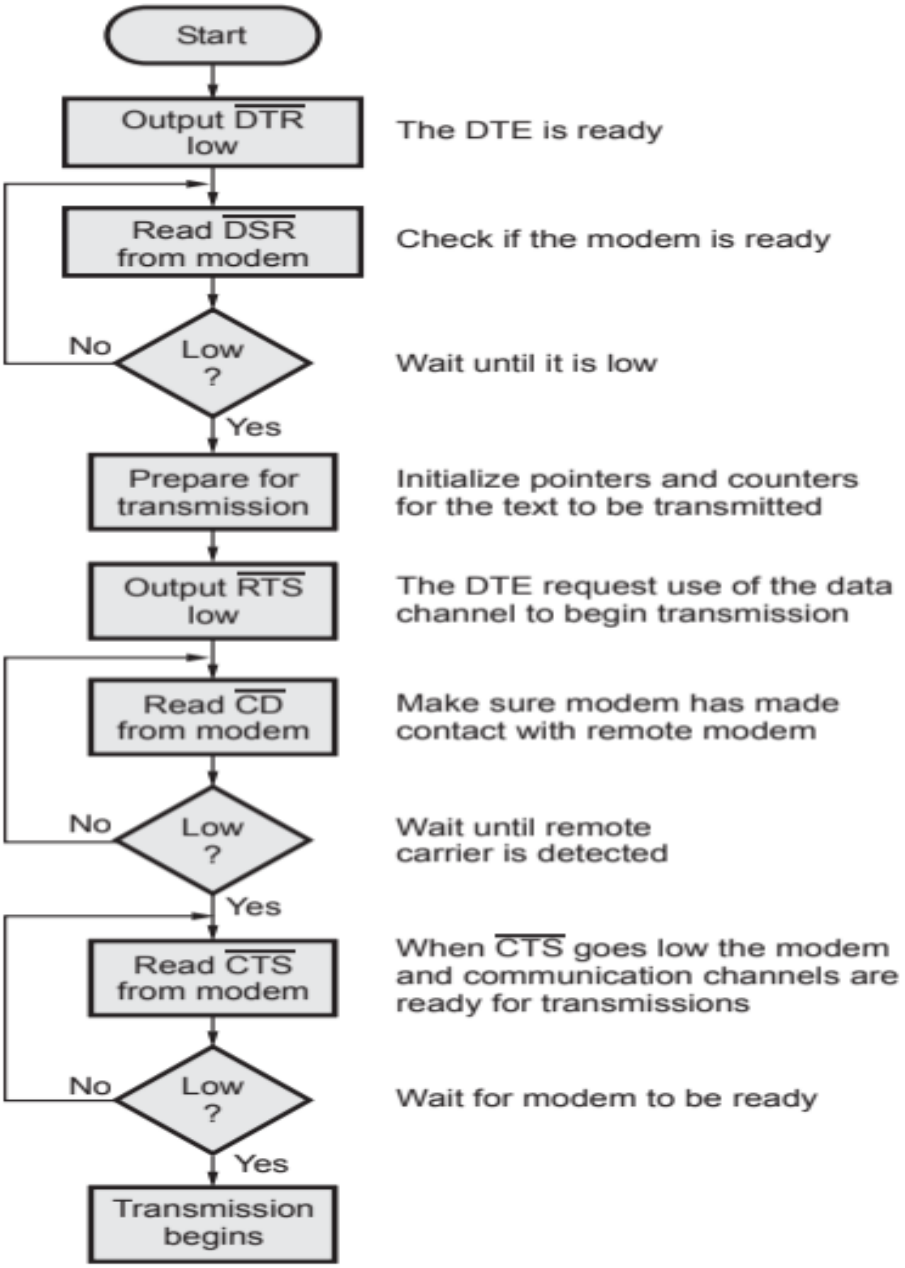
Pin no.	Pin	Description
1	$\overline{\text{DCD}}$	Data carrier detect
2	RxD	Received data
3	TxD	Transmitted data
4	DTR	Data terminal ready
5	GND	Signal ground
6	$\overline{\text{DSR}}$	Data set ready
7	RTS	Request to send
8	CTS	Clear to send
9	RI	Ring indicator

Data Transmission and Reception using RS 232C

- Fig. shows the digital data transmission using modems and standard telephone lines. Modems and other equipment used to send serial data over long distances are known as Data Communication Equipment (DCE). The terminals and computers that are sending or receiving the serial data are referred to as Data Terminal Equipment (DTE).



- After the terminal power is turned on and the terminal runs any self-checks, it asserts the data terminal ready ($\overline{DTR}$) signal to tell the modem it is ready. When modem is ready to transmit or receive data, it asserts the data set ready ($\overline{DTR}$) signal to the terminal.
- After receiving $\overline{DTR}$, the DTE requests use of the data channel by asserting $\overline{RTS}$ signal to begin transmission. The MODEM at the receiving end is then dialed. This answer-mode modem responds with a 2225 Hz carrier frequency. When the modem connected to the sending terminal receives this carrier, it asserts carrier detect ($\overline{CD}$) to the terminal. After proper time interval the modem sends clear-to-send signal ($\overline{CTS}$) indicating modem and communication channels are ready for transmission.



Study of I^2C

- Communication network systems are rapidly growing in size and complexity. These systems have many high speed integrated circuits with critical operating parameters and must provide extremely reliable service with zero down time. To maintain the performance of these systems, adequate environmental monitoring must be performed, so a failure or a data trend leading to a potential failure can be rapidly identified. Furthermore, this monitoring must be performed cheaply to keep system costs low.
- To provide such needs by maximizing hardware efficiency along with providing circuit simplicity, Philips developed a simple bi-directional 2-wire bus for efficient Inter-IC control. This bus is called the **Inter IC (Integrated Circuit) bus** or **I^2C bus**. The I^2C interface employs a comprehensive protocol to ensure reliable transmission and reception of data within connected devices. Here, 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. 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 as a **slave**. Let us see some important features of I^2C bus.

Features of I^2C Bus

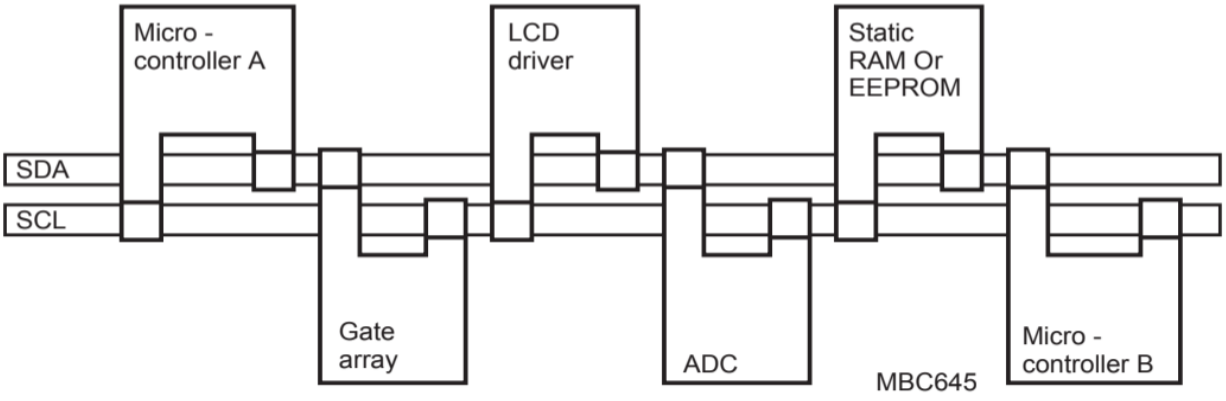
- The important features of the I^2C-bus are :
  - Only two bus lines are required; a serial data line (SDA) and a serial clock (SCL).
  - Each device connected to the bus is software addressable by a unique address and simple master/slave relationships exist at all times; masters can operate as master-transmitters or as master-receivers.
  - It's a true multi-master bus including collision detection and arbitration to prevent data corruption if two or more masters simultaneously initiate data transfer.
  - Serial, 8-bit oriented, bi-directional data transfers can be made at
  - Up to 100 kbit/s in the Standard-mode,
  - Up to 400 kbit/s in the Fast-mode or
  - Up to 3.4 Mbit/s in the High-speed mode
  - On-chip filtering rejects spikes on the bus data line to preserve data integrity.
  - The number of ICs that can be connected to the same bus is limited only by a maximum bus capacitance of 400 pF.

Term	Description
Transmitter	The device that sends the data to the bus.
Receiver	The device that receives the data from the bus.
Master	The device which initiates the transfer, generates the clock and terminates the transfer.
Slave	The device addressed by a master.
Multi-master	More than one master device in a system. These masters can attempt to control the bus at the same time without corrupting the message.
Arbitration	Procedure that ensures that only one of the master devices will control the bus. This ensure that the transfer data does not get corrupted.
Synchronization	Procedure where the clock signals of two or more devices are synchronized.

I^2C BUS Configuration

- Let us see the example of I^2C bus configuration using two microcontrollers. The I^2C-bus is a multi-master bus. This means that more than one device capable of controlling the bus can be connected to it. The Fig. shows the two microcontrollers connected to I^2C-bus. Due to two microcontrollers, there exists master-slave and receiver-transmitter relationships on the I^2C-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 :
- **Case 1 :** 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.
- **Case 2 :** 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 I^2C-bus means that more than one master could try to initiate a data transfer at the same time. To avoid the chaos an arbitration procedure has been developed.



I^2C Signals

Start - High-to-low transition of the SDA line while SCL line is high.

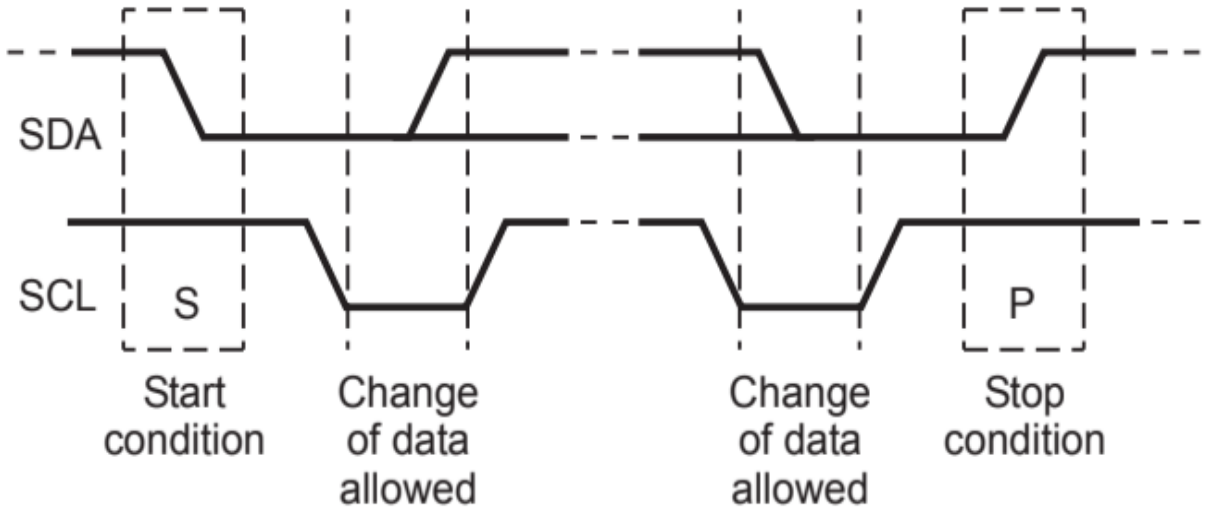
Stop - Low-to-high transition of the SDA line while SCL line is high.

Ack - Receiver pulls SDA low while transmitter allows it to float high.

Data – Transition takes place while SCL is low.

Initiating and Terminating Data Transfer

- During times of no data transfer (idle time), both the clock line (SCL) and the data line (SDA) are pulled high through the external pull-up resistors. The START and STOP conditions determine the start and stop of data transmission. The START condition is defined as a high to low transition of the SDA when the SCL is high. The STOP condition is defined as a low to high transition of the SDA when the SCL is high. Fig. shows the START and STOP conditions. The master generates these conditions for starting and terminating data transfer. Due to the definition of the START and STOP conditions, when data is being transmitted, the SDA line can only change state when the SCL line is low.



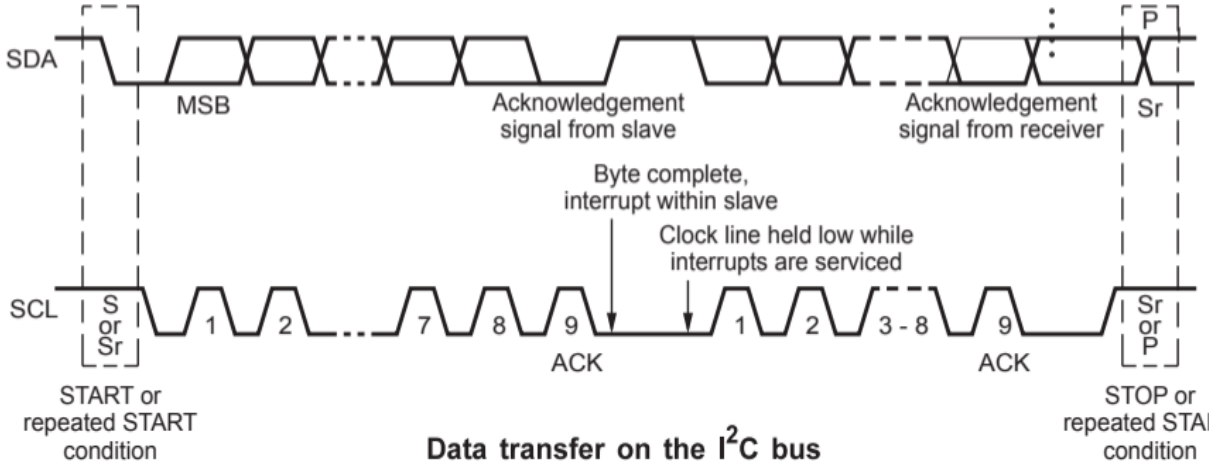
Start and stop conditions

- 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. Thus, hereafter, 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.

Transferring Data

Byte Format

- In the byte format, 8-bit data is put on the SDA line. The number of bytes that can be transmitted per transfer is unrestricted. Each byte has to be followed by an acknowledge bit. Data is transferred with the most significant bit (MSB) first as shown in Fig. If a slave can't receive or transmit another complete byte of data until it has performed some other function, for example servicing an internal interrupt, it can hold the clock line SCL LOW to force the master into a wait state. Data transfer then continues when the slave is ready for another byte of data and releases clock line SCL.

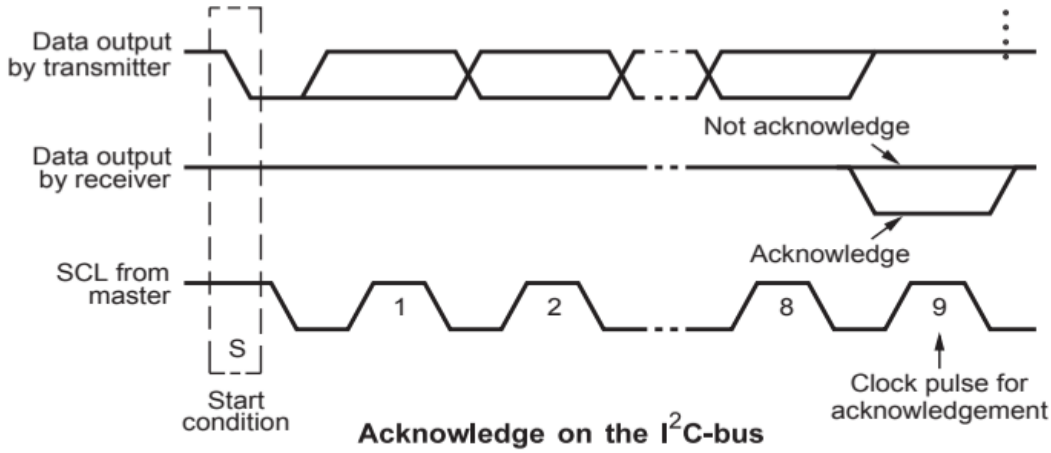


Data transfer on the I^2C bus

- A message which starts with such an address can be terminated by generation of a STOP condition, even during the transmission of a byte. In this case, no acknowledge is generated.

Acknowledge

- Data transfer with acknowledge is compulsory. The acknowledge-related clock pulse is generated by the master. The transmitter releases the SDA line (HIGH) during the acknowledge clock pulse. The receiver must pull down the SDA line during the acknowledge clock pulse so that it remains stable LOW during the HIGH period of this clock pulse as shown in Fig. Of course, set-up and hold times must also be taken into account.
- When a slave doesn't acknowledge the slave address (for example, it's unable to receive or transmit because it's performing some real-time function), the data line must be left HIGH by the slave. The master can then generate either a STOP condition to abort the transfer, or a repeated START condition to start a new transfer.



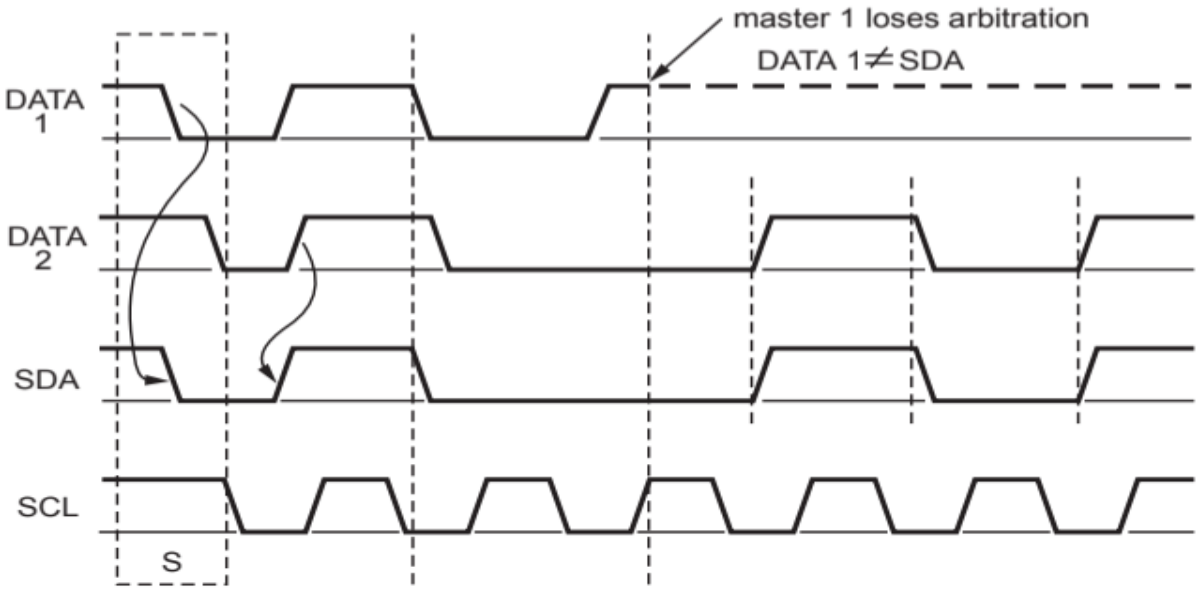
Acknowledge on the I^2C-bus

- If a master-receiver is involved in a transfer, it must signal the end of data to the slave- transmitter by not generating an acknowledge on the last byte that was clocked out of the slave. The slave-transmitter must release the data line to allow the master to generate a STOP or repeated START condition.

Arbitration

- A master may start a transfer only if the bus is free. Two or more masters may generate a START condition within the minimum hold time ($t_{HD;STA}$) of the START condition which results in a defined START condition to the bus. Arbitration takes place on the SDA line, while the SCL line is at the HIGH level, in such a way that

the master which transmits a HIGH level, while another master is transmitting a LOW level will switch off its DATA output stage because the level on the bus doesn't correspond to its own level. This is illustrated in Fig. As shown in the Fig. at point A, master 1 transmits HIGH level and master 2 transmits LOW level. Since SDA line uses wired-AND logic the effective level on the SDA line is LOW. Thus, master 1 level does not match with the SDA level and it switches off its DATA output stage.



Arbitration procedure of two masters

- If more than one masters are trying to address the same device, arbitration continues with comparison of the data-bits if they are master-transmitter, or acknowledge-bits if they are master-receiver. Because address and data information on the I^2C-bus is determined by the winning master, no information is lost during the arbitration process.
- It is important to note that, arbitration isn't allowed between :
  - A repeated START condition and a data bit
  - A STOP condition and a data bit
  - A repeated START condition and a STOP condition.

Note : Slaves are not involved in the arbitration procedure.

Advantages of I^2C

- Good for communication with on-board devices that are accessed occasionally.
- Easy to link multiple devices because of addressing scheme.
- Cost and complexity do not scale up with the number of devices.

Disadvantage of I^2C

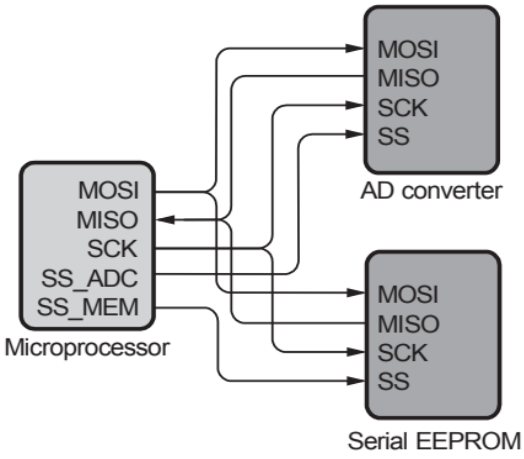
- The complexity of supporting software components can be higher than that of competing schemes (for example, SPI).

Applications of I^2C

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

Study of SPI

- The SPI interface provides bi-directional, synchronous serial communications between microcontrollers and peripherals. It is based upon a master-slave protocol where the master is the device that drives the clock signal. Multiple masters and multiple slaves are allowed on the bus; a simple example, shown below, contains a single master and two slave peripherals. The slave select signals are driven by the master, so that only one slave is accessed at any given time.



One master and two slave peripherals

Features

- The features of SPI protocol are as follows :
  - Defined by Motorola on the MC68HCxx line of microcontrollers.
  - It is a synchronous serial data link operating at full duplex mode.
  - Generally faster than I^2C, capable of 3 Mbps data transfer rate.
  - The SPI dataword size is 8-bits.
  - The SPI provides hold facility which allows the transmitter to suspend data transfer.
  - In SPI data can be transferred in multiple bytes known as blocks or pages.

Standard Naming Convention in SPI

- The standard naming convention used for the SPI signals is as follows:

  Data signals : MOSI : Master data output, slave data input

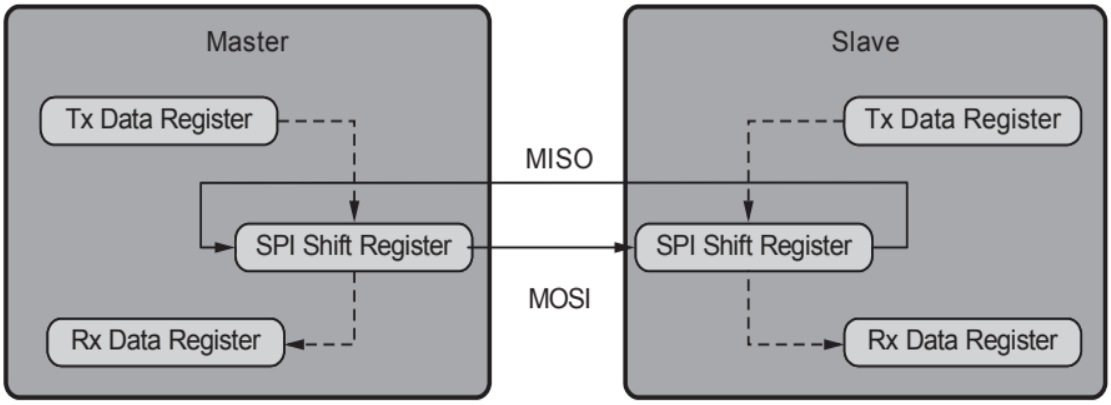
  MISO : Master data input, slave data output

  Control Signals : SS : Slave Select signal (*our example uses two of those,*

SS_ADC and SS_MEM)

SCK : Serial Clock (*always driven by the Master*)

- The transfer of data using an SPI interface is implemented using a large shift register shared between the master and slave devices. Data is clocked IN at the same time it is clocked OUT of the devices (the clock is shared by the two devices). The clock used by SPI is symmetrical, i.e. ON period and OFF period of clock is same. The Fig. shows a high level view of how the internal SPI registers work.



Internal SPI registers

Clock Polarity and Clock Phase

- The SPI interface requires the master and slave devices to be agreed regarding the idle state of the clock signal as well as the way in which the data will be clocked during the SPI transfer.

Clock Polarity : The Clock Polarity (**CPOL**) parameter indicates the level of the clock signal when it is idle.

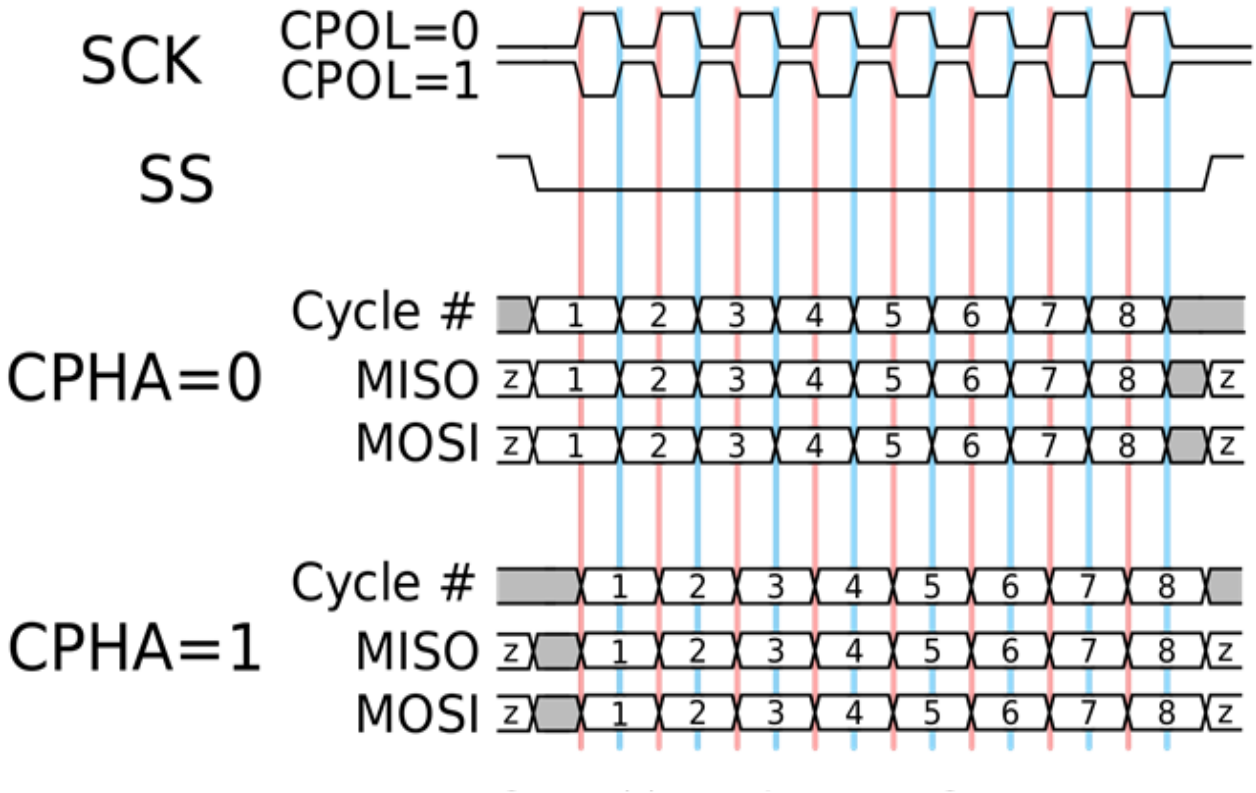
CPOL = 0 Clock idle state is low.

CPOL = 1 Clock idle state is high.

Clock Phase : The Clock Phase (**CPHA**) parameter indicates when the data should be presented and when it should be sampled.

CPHA = 0 The first edge on the SCK line is used to sample the first data bit and the second edge is used to present it (the first bit of the data must be ready before the first edge of the SCK line).

CPHA = 1 The first edge on the SCK line is used to present the data and the second edge to sample it.



Applications

- Like I^2C, used in EEPROM, Flash, and real time clocks.
- Better suited for "data streams", i.e. ADC converters.
- Full duplex capability, hence can be used in communication between a codec and digital signal processor.

SPI Vs I^2C

- For point-to-point, SPI is simple and efficient.
- Less overhead than I^2C due to lack of addressing, plus SPI is full duplex.
- For multiple slaves, each slave needs separate slave select signal hence more effort and more hardware required than I^2C.

UART/USART

- The Universal Synchronous Asynchronous Receiver Transmitter (USART) module is one of the three serial I/O modules incorporated into PIC18FXX8 devices.
- USART is also known as a Serial Communications Interface or SCI.
- It can be configured in the following modes:
  - Asynchronous (full-duplex)
  - Synchronous - **Master** (half-duplex)
  - Synchronous - **Slave** (half-duplex)
- In full-duplex asynchronous mode, it is used to communicate with peripheral devices, such as CRT terminals and personal computers.
- In a half-duplex synchronous system, it is used to communicate with peripheral devices, such as A/D or D/A integrated circuits, serial EEPROMs, etc.
- Registers associated with transmission and reception modes of USART are :
  1. TXREG USART Transmit Register
  2. RCREG : Receive Register
  3. TXSTA : Transmit Status and Control Register
  4. RCSTA : Receive Status and Control Register
  5. SPBRG : Serial Port Baud Rate Generator
  6. PIR1 : Peripheral Interrupt Request Register.

TXREG - Transmit Register and TSR (Transmit Shift Register)

- It is an 8-bit Transmit Buffer register. The data to be transmitted through Tx (RC6) pin is loaded into TxREG register.
- TSR (Transmit Shift Register) is loaded with new data from the TXREG register (if available). Once the TXREG register transfers the data to the TSR register, the TXREG register is empty and flag bit TXIF (PIR1 register) is set.
- TXIF will reset only when new data is loaded into the TXREG register.
- The data from TSR along with start and stop bits are transmitted serially through Tx pin.
- The TSR register is not mapped in data memory, so it is not available to the user.

RCREG : Receive Register

- It is an 8-bit Receive Buffer register. The received through Rx (RC7) pin is loaded into RCREG register.
- On completely receiving the word, the RSR (Receive Shift Register) will transfer the data to the RCREG register.

TXSTA : Transmit Status and Control Register

- Fig. shows the Transmit Status and Control register (TXSTA)
- The TXSTA register is used to select the clock source, mode of transmission, baud rate, and to enable/disable transmission.

CSRC	TX9	TXEN	SYNC	–	BRGH	TRMT	TX9D

bit 7 bit 0

bit 7 **CSRC :** Clock Source Select bit

Asynchronous mode :

Don't care

Synchronous mode :

1 = Master mode (clock generated internally from BRG)

0 = Slave mode (clock from external source)

bit 6 **TX9 :** 9 bit Transmit Enable bit

1 = Selects 9-bit transmission

0 = Selects 8 bit transmission

bit 5 **TXEN :** Transmit Enable bit

1 = Transmit enable

0 = Transmit disabled

Note : SREN/CREN overrides TXEN in Sync mode,

bit 4 **SYNC :** USART Mode select bit

1 = Synchronous mode

0 = Asynchronous mode

bit 3 **Unimplemented :** Read as '0'

bit 2 **BRGH :** High Baud Rate Select bit

Asynchronous mode :

1 = High speed

0 = Low Speed

Synchronous mode :

Unused in this mode.

bit 1 **TRMT :** Transmit Shift Register Status bit

1 = TSR empty

0 = TSR full

bit 0 **TX9D :** 9th bit of Transmit Data

Can be address / data bit or a parity bit.

RCSTA : Receive Status and Control Register

- Fig shows the Receive Status and Control register (RCSTA)
- The RCSTA register is used to enable the single or continuous receive mode and to enable/disable reception.

SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D

bit 7 bit 0

bit 7 **SPEN :** Serial Port Enable bit

1 = Serial port enabled (configures RX/DT and TX/CK pins as serial port pins)

0 = Serial port disabled

bit 6 **RX9 :** 9-bit Receive Enable bit

1 = Selects 9-bit reception

0 = Selects 8-bit reception

bit 5 **SREN :** Single Receive Enable bit

Asynchronous mode :

Don't care

Synchronous mode - Master :

1 = Enables single receive

0 = Disables singles receive (this bit is cleared after reception is complete)

Synchoronous mode : Slave :

Unused in this mode

bit 4 **CREN :** Continous Receive Enable bit

Asynchronous mode :

1 = Enables continous receive

0 = Disables continous receive

Synchronous mode :

1 = Enables continous receive until CREN is cleared (CREN overrides SREN)

0 = Disables continous receive

bit 3 **ADDEN :** Address detect Enable bit

Asynchronous mode 9-bit (RX9 = 1) :

1 = Enables address detection, enables interrupt and load of the receive buffer when RSR<8> is set

0 = Disables address detectionm all bytes are received and ninth bit can be used parity bit

bit 2 **FERR :** Framing Error bit

1 = Framing error (can be updated by reading RCREG register and receive next valid byte)

0 = No framing error

bit 1 **OERR :** Overrun Error bit

1 = Overrun error (can be cleared by clearing bit CREN)

0 = No overrun error

bit 0 RX9D : 9th bit of Received Data

Can be addresss/data bit or parity bit

SPBRG : Serial Port Baud Rate Generator

- The SPBRG register is a bit register used to hold the value which decides the baud rate, i.e., it is used to program the baud rate in both the Asynchronous and Synchronous modes of the USART.
- Given the desired baud rate and FOSC, the nearest integer value for the SPBRG register can be calculated using the formula shown in Table It shows the formula for the computation of the baud rate for different USART modes.

SYNC	BRGH = 0 (Low speed)	BRGH = 1 (High speed)
0	(Asynchronous) Baud Rate = Fosc/(64 (X+1))	Baud Rate = Fosc /(16 (X+1))
1	(Synchronous) Baud Rate = Fosc/(4 (X+1))	NA

Note : X = value in SPBRG (0 to 255)

Table: Baud rate formula

- As shown in Table 5.6.1, bit BRGH (TXSTA register) also controls the baud rate in asynchronous mode. In Synchronous mode, bit BRGH is ignored.
- **For example,** when SYNC = 0 and BRGH = 0,

$$X = \frac{FOSC}{\text{Desired Baud Rate} \times 64} - 1$$

- There are two ways to increase the baud rate of data transfer in the PIC18.
  - Use a higher frequency crystal
  - Make BRGH (Bit2 in the TXSTA register) = 1
- For a Fixed crystal frequency, we can **quadruple the baud rate** by making BRGH = 1.

PIR1 : Peripheral Interrupt Request (PIR) Register1

- The Peripheral Interrupt Request (PIR) registers contain the individual flag bits for the peripheral interrupts.
- Bits 4 and 5 of PIR1 register are used by the USART. They are :
  - RCIF: USART Receive Interrupt Flag bit
  - TXIF: USART Transmit Interrupt Flag bit

PSPIF	ADIF	**RCIF**	**TXIF**	SSPIF	CCP1IF	TMR2IF	TMR1IF

bit 7 bit 0

bit 5 **RCIF** : USART Receive interrupt Flag bit

1 = The USART receive buffer, RCREG is full (cleared when RCREG, is read)

0 = The USART receive buffer is empty

bit 4 **TXIF** : USART Receive interrupt Flag bit

1 = The USART transmit buffer, TXREG is empty (cleared when TXREG is written)

0 = The USART transmit buffer is full

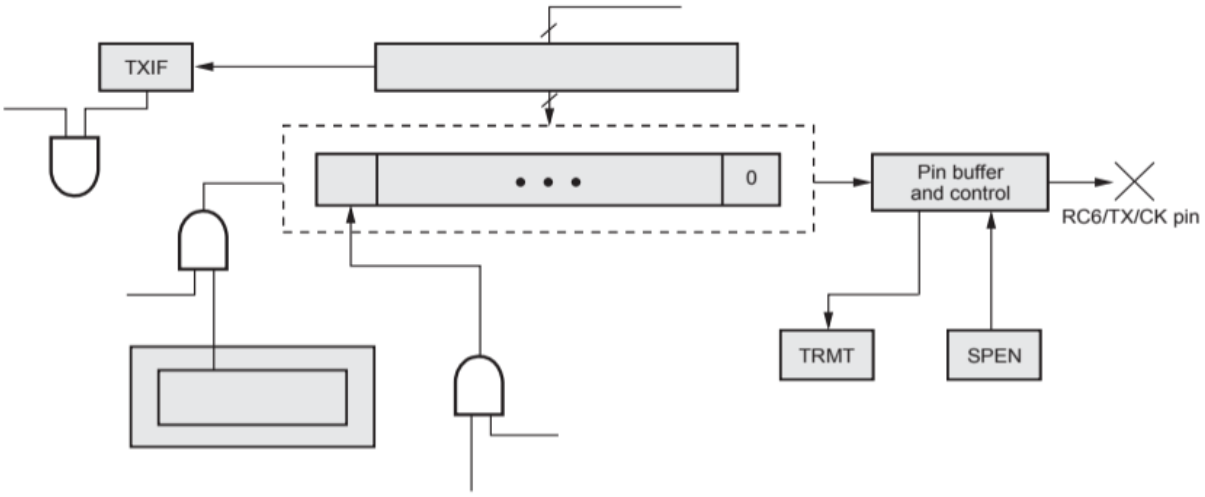
USART Asynchronous Mode

- In this mode, the USART uses standard Non-Return-to-Zero (NRZ) format (one Start bit, eight or nine data bits, and one Stop bit). The most common data format is 8 bits.
- An on-chip dedicated 8-bit Baud Rate Generator can be used to derive standard baud rate frequencies from the oscillator.
- The USART transmits and receives the LSB first. The USART's transmitter and receiver are functionally independent but use the same data format and baud rate.
- The Baud Rate Generator produces a clock, either x16 or x64 of the bit shift rate, depending on the BRGH bit (TXSTA register).
- Parity is not supported by the hardware but can be implemented in software (and stored as the ninth data bit).
- Asynchronous mode is stopped during Sleep. It is selected by making the SYNC bit = 0 (TXSTA register).
- The USART Asynchronous module consists of the following important elements:
  - Baud Rate Generator
  - Sampling Circuit
  - Asynchronous Transmitter
  - Asynchronous Receiver

USART Asynchronous Transmitter

Block Diagram

- Fig shows the block diagram of USART transmitter.
- The heart of the transmitter is the **Transmit (Serial) Shift Register (TSR)**. The TSR register obtains its data from the Read/Write Transmit Buffer register (TXREG).

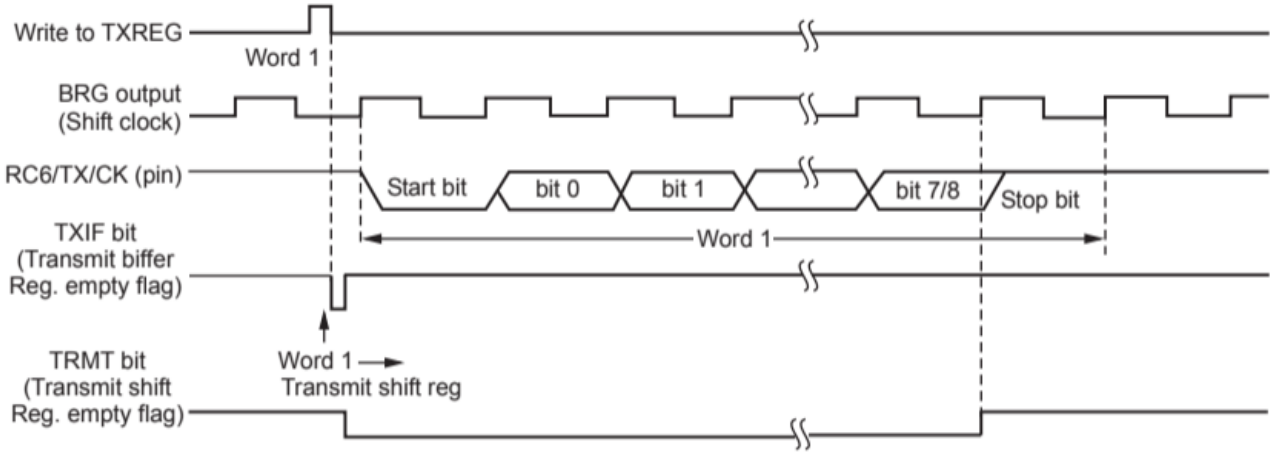


- It is loaded with data in software. The TSR register is not loaded until the Stop bit has been transmitted from the previous load. As soon as the Stop bit is transmitted, the TSR is loaded with new data from the TXREG register (if available).
- Once the TXREG register transfers the data to the TSR register, the TXREG register is empty, and flag bit TXIF (PIR1 register) is set. This interrupt can be enabled/disabled by setting/clearing enable bit TXIE (PIE1 register). It will reset only when new data is loaded into the TXREG register.
- TRMT is a read-only bit that is set when the TSR register is empty. No interrupt logic is tied to this bit, so the user has to poll this bit to determine if the TSR register is empty.
- SPEN bit is set to enable the serial port.
- Tx9D holds the ninth bit to be used in 9-bit transmission.

Steps to follow for setting up an Asynchronous Transmission

1. Initialize the SPBRG register and BRGH bit for the appropriate baud rate.
2. Make SYNC bit = 0 and SPEN bit = 1 to enable the asynchronous serial port
3. If interrupts are desired, set enable bit TXIE.
4. If 9-bit transmission is desired, set transmit bit TX9.
5. Make TXEN = 1 to enable the transmission which will also set bit TXIF.
6. If 9-bit transmission is selected, the ninth bit should be loaded in bit TX9D.
7. Load data to the TXREG register (starts transmission).

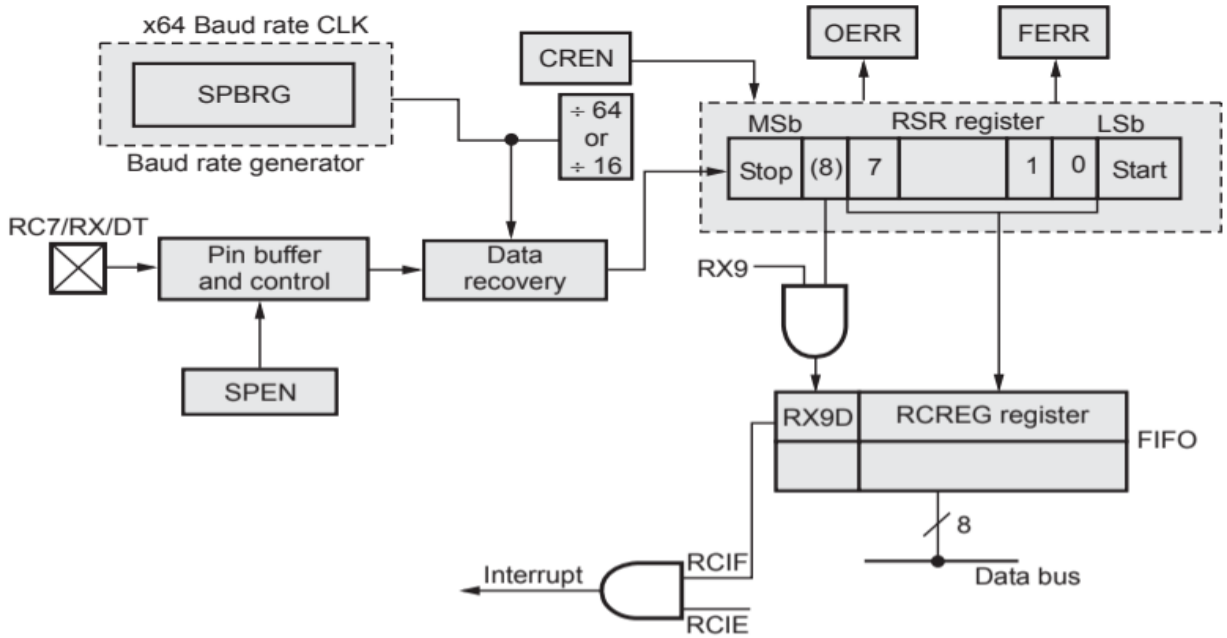
Timing Diagram



Timing diagram for 8/9 bit asynchronous transmission

USART Asynchronous Receiver

Block Diagram



Block diagram of USART receiver

- The data is received on the RC7/RX/DT pin and drives the data recovery block. The data recovery block is actually a high-speed shifter, operating at x16 times the baud rate, whereas the main receive serial shifter operates at the bit rate or at FOSC. This mode would typically be used in RS-232 systems.

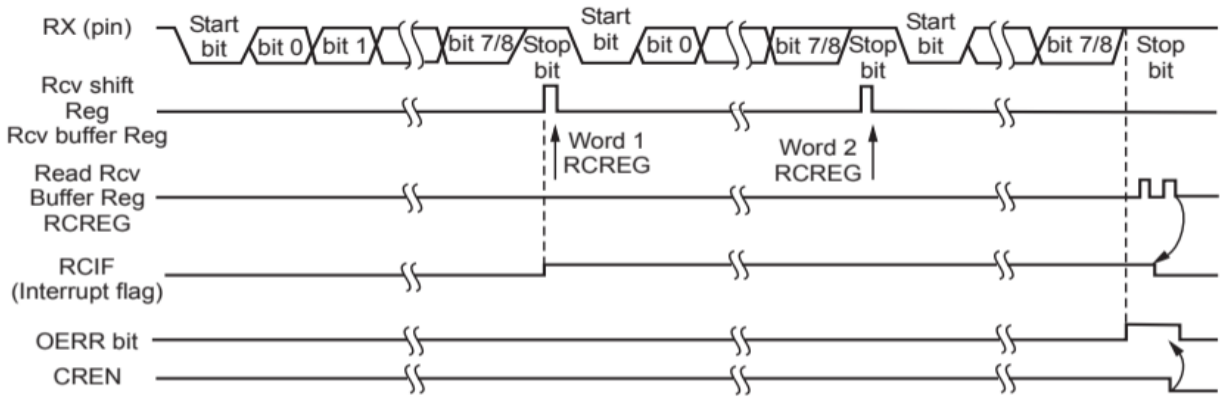
- Once the Asynchronous mode is selected, reception is enabled by setting bit CREN.
- The heart of the receiver is the **Receive Serial Shift Register (RSR)**. After sampling the Stop bit, the received data frame in the RSR is transferred to the RCREG register (if it is empty).
- If the transfer is complete, the flag bit, RCIF, is set. The actual interrupt can be enabled/disabled by setting/clearing the enable bit, RCIE. Flag bit RCIF is a read-only bit which is cleared by the hardware. It is cleared when the RCREG register has been read and is empty.
- The RCREG is a double-buffered register, which means it's a two-level deep FIFO. With the help of it, we can receive two bytes and load them in FIFO.
- OERR bit is set if the third byte is received and RCREG is full.
- FERR bit is set if a low-level stop bit is detected.
- SPEN bit is set to enable the serial port.
- In 9-bit transmission, the ninth bit is stored in Rx9D.

Steps to follow for setting up an Asynchronous Reception

1. Initialize the SPBRG register and BRGH bit for the appropriate baud rate.
2. Make SYNC bit = 0 and SPEN bit = 1 to enable the asynchronous serial port
3. If interrupts are desired, set enable bit RCIE.
4. If 9-bit reception is desired, set bit RX9.
5. Enable the reception by setting bit CREN.
6. Flag bit RCIF will be set when reception is complete and an interrupt will be generated if enable bit RCIE was set.
7. Read the RCSTA register to get the ninth bit (if enabled) and determine if any error occurred during reception.
8. Read the 8-bit received data by reading the RCREG register.
9. If any error occurred, clear the error by clearing enable bit CREN.

Timing Diagram

- Fig. shows the timing diagram for 8/9 bit asynchronous reception.
- **Note :** This timing diagram shows three words appearing on the RX input. The RCREG (receive buffer) is read after the third word, causing the OERR (overrun) bit to be set.



Timing diagram for 8/9 bit asynchronous reception

Setting Up 9-Bit Mode With Address Detect

- This mode would typically be used in RS-485 systems.

Steps to follow for setting up an Asynchronous Reception with Address Detect Enable

1. Initialize the SPBRG register and BRGH bit for the appropriate baud rate.
2. Make SYNC bit = 0 and SPEN bit = 1 to enable the asynchronous serial port
3. If interrupts are required, set the RCEN bit and select the desired priority level with the RCIP bit.
4. Set the RX9 bit to enable 9-bit reception.
5. Set the ADDEN bit to enable address detect.
6. Enable reception by setting the CREN bit.
7. The RCIF bit will be set when reception is complete. The interrupt will be acknowledged if the RCIE and GIE bits are set.
8. Read the RCSTA register to determine if any error occurred during reception, as well as read bit 9 of data (if applicable).
9. Read RCREG to determine if the device is being addressed.
10. If any error occurred, clear the CREN bit.
11. If the device has been addressed, clear the ADDEN bit to allow all received data into the receive buffer and interrupt the CPU.

Serial Communication Programming using Embedded C

Programming PIC18 to Transfer Data Serially

Steps for Programming PIC18 to Transfer Data Serially

1. Load the value in TXSTA register for the desired configuration as shown below

Value	Configuration
20H	Asynchronous mode with 8-bit data frame, low baud rate and transmit enable.
60H	Asynchronous mode with 9-bit data frame, low baud rate and transmit enable.
24H	Asynchronous mode with 8-bit data frame, high baud rate and transmit enable.
64H	Asynchronous mode with 9-bit data frame, high baud rate and transmit enable.

2. Make TX pin of PORTC (RC6) an output pin.
3. Initialize the SPBRG register and BRGH bit for the appropriate baud rate.
4. Set SPEN bit in the RCSTA register to enable the serial port.
5. Load the character byte required to transmit in TXREG register.
6. Monitor the TXIF bit of PIR1 register to make UART ready for the next byte.
7. Repeat steps 5 and 6 to transmit the next character.

Programming PIC18 to Receive Data Serially

Steps for Programming PIC18 to Receive Data Serially

1. Load the value in RCSTA register for the desired configuration as shown below

Value	Configuration
90H	Asynchronous mode with 8-bit data frame and receive enable.
D0H	Asynchronous mode with 8-bit data frame and receive enable.

2. Initialize the SPBRG register and BRGH bit for the appropriate baud rate.
3. Make the RX Pin of PORT (RC7) as input for data to receive.
4. Monitor the RCIF bit = 1 of the PIR1 register to check for the entire character reception.
5. When RCIF is raised, the RCREG register has the byte. So save the contents of the RCREG register.
6. Repeat steps 5 and 6 to receive the next character.

Programming Examples

Write a C program for the PIC18 to transfer letter "A" serially at 9600 baud, continuously. Use 8-bit data and 1-stop bit. Assume XTAL = 10 MHz.

Solution :

```
#include<P18F458.h>
void main(void)
{
    TXSTA = 0x20;          // Asynchronous mode with 8-bit data frame, low baud rate
                           // and transmit enable : TX9 = 0, TXEN = 1, SYNC  = 0,
                                BRGH = 0
    SPBRG = 15;            // 9600 baud rate
    TRISBbits.TRISC6=0;    // configure pin 6 of Port C (Transmitter pin)  as output
    RCSTAbits.SPEN=1;      // Enable serial port
  while(1)
  {
    TXREG = 'A' ;                 // Send character
    while(PIR1bits.TXIF==0);      // Wait until  TXREG register becomes empty
  }
}
```

Write a C program for the PIC18 to transfer the message "YES" serially at 9600 baud, 8-bit data, 1 stop bit. Do this continuously. Assume XTAL = 10 MHz.

Solution :

```
#include<P18F458.h>
void main(void)
{
     TXSTA = 0x20;          // Asynchronous mode with 8-bit data frame, low baud rate
                            // and transmit enable : TX9 = 0, TXEN = 1, SYNC  = 0,
                               BRGH = 0
     SPBRG = 15;            // 9600 baud rate
     TRISBbits.TRISC6=0;    // configure pin 6 of Port C (Transmitter pin)  as output

   RCSTAbits.SPEN=1;       // Enable serial port
  while(1)
  {
   TXREG = 'Y' ;                  // Send character Y
   while(PIR1bits.TXIF==0);       // Wait until  TXREG register becomes empty
   TXREG = 'E' ;                  // Send character E
   while(PIR1bits.TXIF==0);       // Wait until  TXREG register becomes empty
   TXREG = 'S' ;                  // Send character S
   while(PIR1bits.TXIF==0);       // Wait until  TXREG register becomes empty
  }
}
```

Write a C program for the PIC18 to send the two messages "Normal Speed" and "High Speed" to the serial port. Assuming that SW is connected to pin PORTB.0, monitor its status and set the baud rate as follows :
SW = 0, 9600 baud rate
SW = 1, 38400 baud rate
Assume that XTAL = 10 MHz for both cases.

```
#include<P18F458.h>
#define SWBit PORTBbits.RB0      // Switch input
void main(void)
{
      unsigned char z;
      unsigned char Mes1[ ]="Normal Speed";
      unsigned char Mes2[ ]="High Speed";
      TRISBbits.TRISB0=1;     // configure pin 0 of Port B as input
      TXSTA = 0x20;           // Asynchronous mode with 8-bit data frame, low baud rate
                              // and transmit enable : TX9 = 0, TXEN = 1, SYNC  = 0,
                                    BRGH = 0
      SPBRG = 15;             // 9600 baud rate
      RCSTAbits.SPEN=1;       // Enable serial port
      If (SWBit == 0)

   {
        for (z =0; z<12; z++)
        {
             while(PIR1bits.TXIF==0);     // Wait until  TXREG register becomes empty
             TXREG = Mes1[z ] ;           // Send character
        }
   }
   else
   {
        TXSTAbits.BRGH = 1;          //  Select high baud rate (Low baud rate x 4)
        for (z =0; z<10; z++)
        {
             while(PIR1bits.TXIF==0);     // Wait until  TXREG register becomes empty
             TXREG = Mes2[z ] ;           // Send character
        }
   }
while(1)
}
```

****