Automatic Wireless Registration System for Vehicles

Syndicate

NC ALI JAVED
PC MUHAMMAD ABDULLAH
NC MUHAMMAD NAEEM YOUNIS
NC TAHAWAR ALI
NC SHOAIB KHAN

Project Supervisor

Gp Capt (R) Muzaffar Ali

Submitted to the Faculty of Electrical Engineering
National University of Sciences and Technology, Rawalpindi
in partial fulfilment of the requirements of a B.E degree in Telecomm Engineering

April 2007

DEDICATION

DEDICATED TO OUR BELOVED PARENTS AND FRIENDS WHOSE CONSTANT SUPPORT AND APPRECIATION HAS BEEN A SOURCE OF ENCOURAGEMENT FOR US THROUGHOUT.

ACKNOWLEDGEMENTS

We extend our sincere gratitude to all those who helped us complete the project successfully. We thank our internal supervisor Gp Capt (R) Muzaffar Ali whose experienced advice stood as the paragon for us. His kind help and interest and support showed us right through every stage of the project and we were able to align ourselves in the right direction.

DECLARATION

No portion of the work presented in this dissertation has been submitted in support of another award of qualification either at this institution or elsewhere.

ABSTRACT

Automatic wireless registration system for vehicles is one of the latest techniques in the present era of telecommunication. This technique is never being implanted before on commercial level. Although a lot of work is being done in the field of RFID technology using passive tags. We have used an active module in the car comprising Microcontroller and RF Modem. One of the main reasons for working on this idea was the fact that manual compliance monitoring methods are sporadic and usually depend upon other incident detection events, resulting in minimal sampling of the total vehicle population.

The aim of the project is to establish a communication link between the RF Modems using wireless channel. The idea is to successfully implement this technology in lab environment in a cost effective manner using minimum resources. For this purpose microcontrollers are used for the processing, with the input and output through RF modems. The microcontroller code is implemented using Keil and the base server code is implemented using Visual Basic.

Table of Contents

List of Figures

CHAPTER 1

INTRODUCTION

1.1 Electromagnetic Spectrum

Waves, as means of transfer of energy, is a well known phenomenon in physics we can see waves being produced when we throw a stone in water. Water waves are type that we can see. Sound is a type of wave that we cannot see. Like water waves sound waves also need medium to travel through. Sound can travel through air because air is made up of molecules. The molecules carry the sound by bumping into each other, like dominoes knocking each other over. Sound can travel through anything made of molecules-even water.

There is no sound in space due to absence of molecules to transmit sound waves. Electromagnetic waves are unlike sound waves because they do not need molecules to travel this means electromagnetic waves can travel through air and solid materials-but they can also travel through empty spaces. This is why astronauts on the space walks use radios to communicate radio waves are type of electromagnetic waves.

While water waves can be produced mechanically by repeatedly dipping a stick in water and sound waves by vibrations of diaphragm. Electromagnetic waves are formed when an electric field couples with a magnetic field. Magnetic and electric fields of an electromagnetic wave are perpendicular to each other and to the direction of waves.

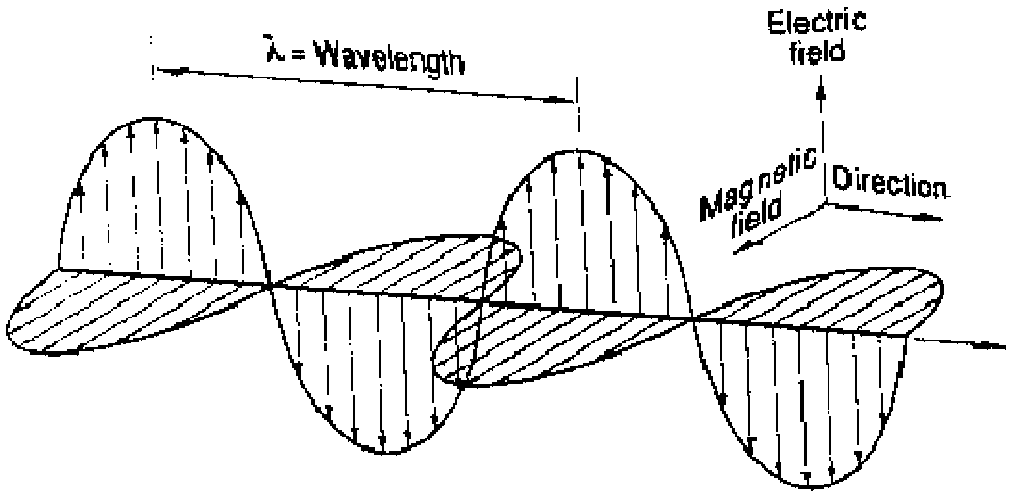


Fig 1.1 Waves

Electromagnetic waves like any other waves have several characteristics. The highest part is referred to as crest and lowest is referred as trough. The horizontal distance between two successive crests or two troughs is called wavelength. The number of crests or troughs passing a given point in a particular wave period is called the wave frequency. As the wave frequency increases, the wavelength decreases. The amount of energy carried depends on wavelength. The shorter the wavelength, the more the energy contained; the long the wavelength, the less the energy contained.

Actually, the electromagnetic spectrum is the range of wavelengths from short wavelengths (below visible light)to long wavelengths(above visible light).electromagnetic radiations travels in a straight line at the fixed rate of speed of light,299,792,458 km/s in a vacuum. No other observable body's rate can exceed the speed of light. The electromagnetic spectrum includes the following arranged in order from the shortest wavelength to the longest wavelength: Gamma rays, X-rays, ultraviolet rays, visible light, infrared rays, microwaves, and radio waves.

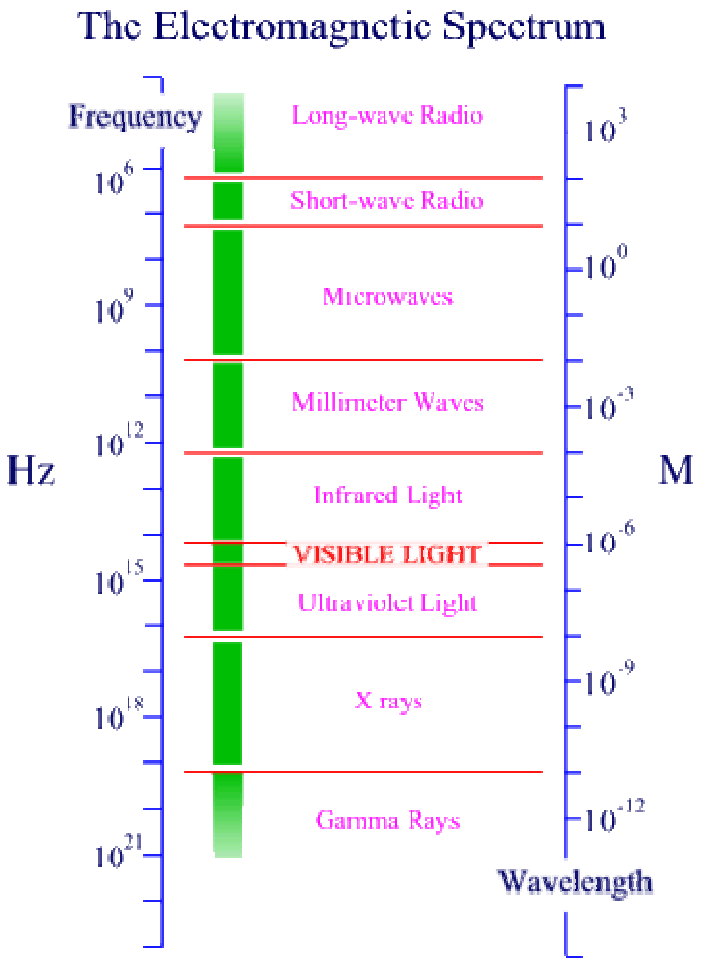


Fig 1.2 Electromagnetic spectrum

Light is the only visible part of spectrum. Visible light makes only small portion of spectrum. It consists of rainbow of colors ranging from red (longest wavelength) to the (shortest wavelength). Gamma rays have wavelengths less than one ten billionth of a meter.

On the other extreme, radio waves have wavelength from one millimeter to several kilometers. The following figure 1.3 also helps us visualize the different waves comprising electromagnetic spectrum.

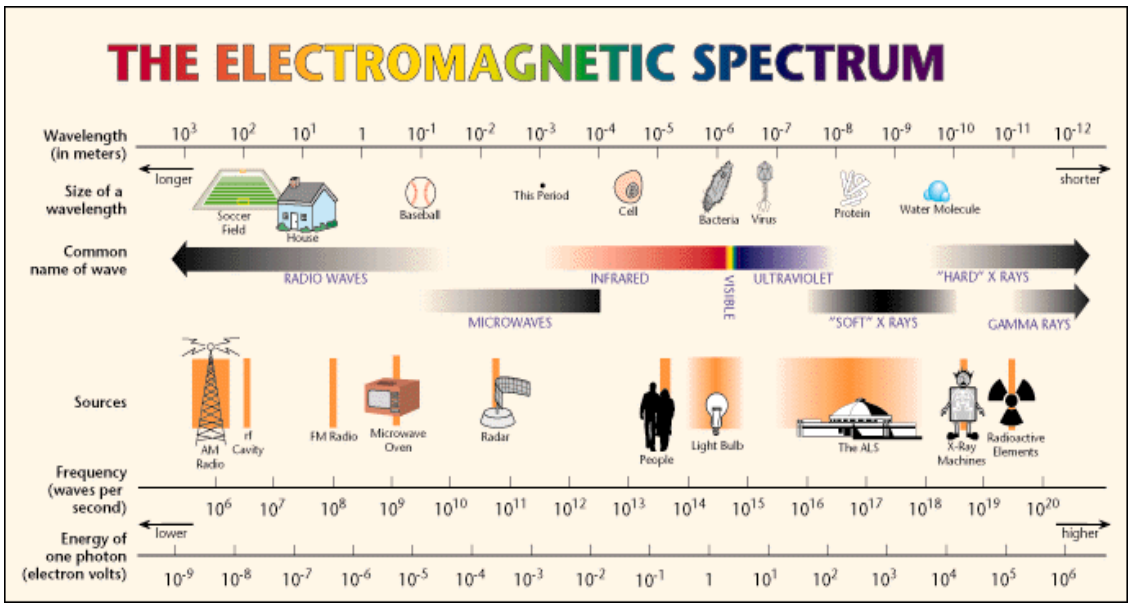


Fig 1.3 Electromagnetic Spectrum

When electrons move, their movement creates a magnetic field. Electromagnetic waves are created by the vibration (movement back and forth or oscillation) of an electric charge. The vibration of atoms creates a wave with both an electric and magnetic component that are both changing the average energy of the vibrating atoms produce heat which as a result, gives off electromagnetic radiations. An electromagnetic radiation is emitted from a matter with a temperature greater than absolute zero. As the temperature increases (atoms vibrate faster), more radiations will be emitted and shorter wavelengths will be developed if the temperature decreases (atoms vibrate slower), fewer radiations will be emitted and longer wavelengths will develop.

1.2 Wireless communication using electromagnetic wave

As the electromagnetic wave is travelling through space it is carrying communication in the form of electromagnetic radiations. In order to retrieve the information to be used by wireless technology a transmitter and a receiver are needed. A transmitter is a component that is used to create a replica of the electromagnetic wave. Once the transmitter has created the wave a receiver, found in most of today's wireless products, accept the waves and decodes the modulated signals. Wireless device can be connected by two different approaches: point-to-point and networked .A point-to-point connection is usually done using infrared. A networked connection uses bridges (base stations) to connect wireless devices to physical networks .An

example of networking is using a wireless network card on a laptop computer. All transmitted information from the electromagnetic wave comes directly from the base station of the network.

1.3 Radio Frequency

Our project uses radio waves as a means of wireless data communication. Radio frequency is a term that refers to alternating current having characteristics such that, it' if the current is input to an antenna, an electromagnetic (EM) field is generated suitable for wireless broadcasting and/or communications. These frequencies cover a significant portion of the electromagnetic radiation spectrum, extending from nine kilo hertz (9 kHz), the lowest allocated wireless communications frequency (it's within the range of human hearing),to thousands of giga hertz (GHz).

The really nice thing about radio waves is that they will make the electrons in a piece of copper wire move this means that they generate electric currents in the wire, and electromagnetic waves generate alternating currents. The electric currents at "radio frequencies" (RF) are used by radio and television transmitters and receivers.

When a radio frequency current is supplied to an antenna, it gives rise to an electromagnetic field that propagates through space. This field is sometimes called an RF field; in less technical jargon it is a "radio Wave". Any RF field has a wavelength that is inversely proportional to the frequency. In the atmosphere or in outer space, if *f* is the frequency in megahertz and *s* is the wavelength in meters, then

$$S = c/f$$

The frequency of an RF signal is inversely proportional to the wavelength of the EM field to which it corresponds. At 9 kHz, the free space wavelength is approximately 33 kilometres (km) or 21 miles (mi). At the highest radio frequencies, the EM wavelength measure approximately one millimetre

(1mm). As the frequency is increased beyond that of the RF spectrum, EM energy takes the form of infrared (IR), visible, ultraviolet (UV), X rays, and gamma rays.

Many types of wireless devices make use o f frequency cordless and cellular telephone, radio and television broadcast stations, satellite communications systems, and two-way radio services all operate in the radio frequency spectrum. Some wireless devices operate at infrared or visible-light frequencies, whose electromagnetic wavelengths are shorter than those of radio frequency fields. Examples include most television –set, remote control boxes, some cordless computer keyboards and mice, and a few wireless hi-fi stereo headsets.

The radio frequency spectrum is divided in to several ranges, or bands. With the exception of the lowest frequency segment, each band represents an increase of frequency corresponding to an order of magnitude (power of 10). The table depicts the eighth bands in the radio frequency spectrum, showing frequency and bandwidth ranges. The super high frequency and extremely high frequency bands are often referred to as the *microwave spectrum.*

Designation	**Abbreviation**	**Frequencies**	**Free-space** **Wavelengths**
Very low frequency	VF	9kHz – 30kHz	33km – 10km
Low frequency	LF	30kHz – 300kHz	10km – 1 km
Medium frequency	MF	300kHz – 3MHz	1km – 100m
High frequency	HF	3Mhz – 30MHz	100m – 10m
Very high frequency	VHF	30MHz – 3MHz	10m – 1 m

Ultra high frequency	UHF	300MHz – 30GHz	1m – 100mm
Super high frequency	SHF	3GHz – 30GHz	100mm – 10mm
Extremely high Frequency	EHF	30GHz – 300GHz	10mm – 1mm

1.4 Brief Project Description

1.4.1 Basic Goal

The project is to design a system for vehicles that would provide various facilities, for example

1. Registration of vehicles
2. Detection of blacklisted vehicles
3. To keep their roaming record from one city to another
4. And to keep up with the security of the environment, etc.

1.4.2 Basic Architecture

The project has been divided into two modules for the implementation of the idea. Main feature of both the modules is that both are active and this is where it differs from the rest of the RFID systems so far implemented practically. The basic architecture of both the modules is described below briefly;

a. Car Module

The car module will be installed in the car and it consists of the following basic three components;

- Microcontroller

- RS-232 level converter
- RF Modem

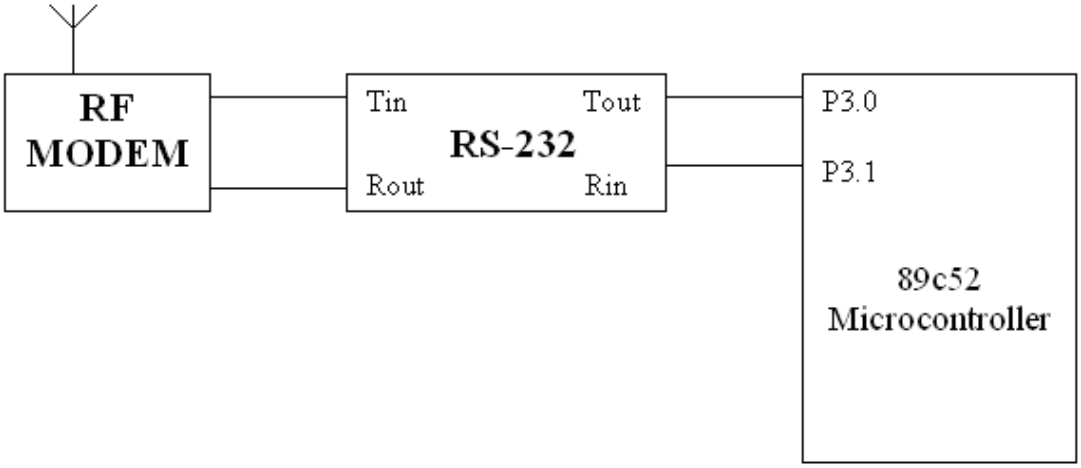


Fig 1.4 Car Module Block Diagram

The microcontroller in the car module is being programmed and also contains the data of the car. RS-232 IC is a level converter and acts as an interface between the microcontroller and RF modem. RF Modem is used for the reception and transmission of data. Prior to use of the RF modem it has also been programmed using its software X-CTU and all the parameters for the wireless communication between RF modems has been set.

b. Base Module

The base module is installed at the point near entry or path to be monitored and acts as a base to the car module. The base module consists of the following components;

- Base server (computer)
- RS-232 level converter
- RF Modem

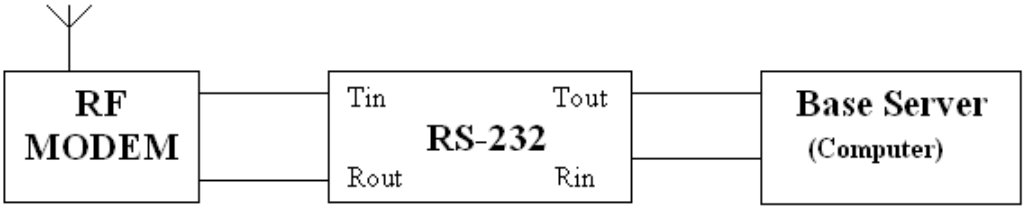


Fig 1.5 Base Module Block Diagram

The base server is the database which contains all the data of the authenticated cars belonging to that campus RS-232 IC is a level converter and acts as an interface between the base server and RF modem. RF Modem is used for the reception and transmission of data. Prior to use of the RF modem it has also been programmed using its software X-CTU and all the parameters for the wireless communication between RF modems has been set.

1.4.3 Basic Working

1. Base Interrogation

The base is continuously transmitting an interrogation message over its range so that if any car has entered the range it must get registered with the base.

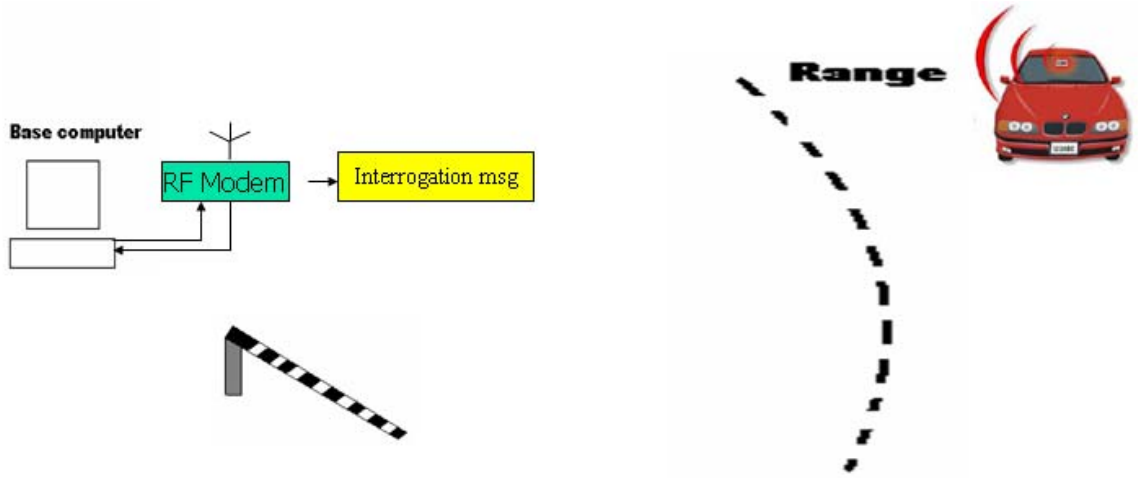


Fig 1.6 Base Interrogation

2. Reply from the Car Module

The car module receives the interrogation message and sends its data stored in the car module (Registration No., Chassis No., Engine No., Owner's Name and Make of the car etc.)

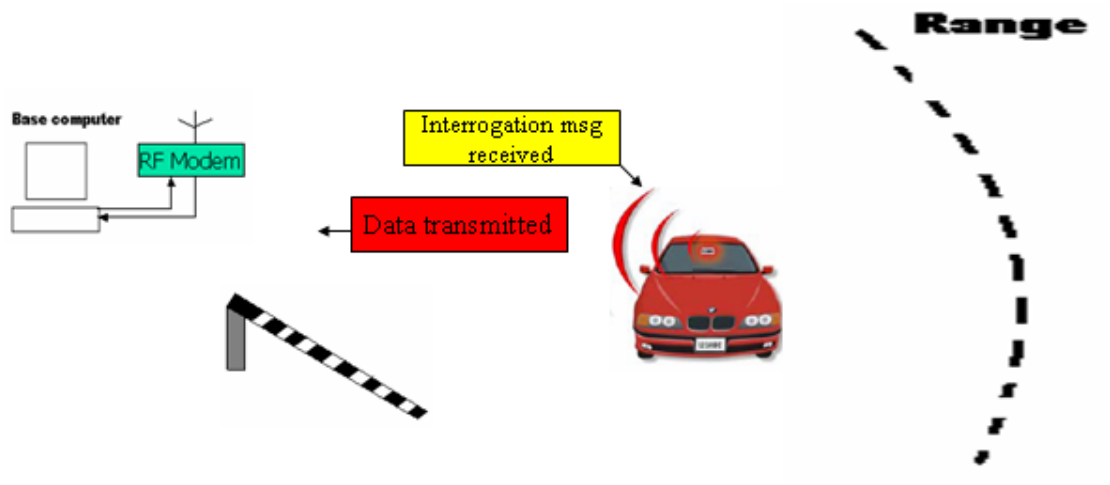


Fig 1.7 Reply from the Car

3. Data received by the Base Module

The base receives the data from the car and checks whether the data is valid. Then it registers the car and sends an authentication message to the car.

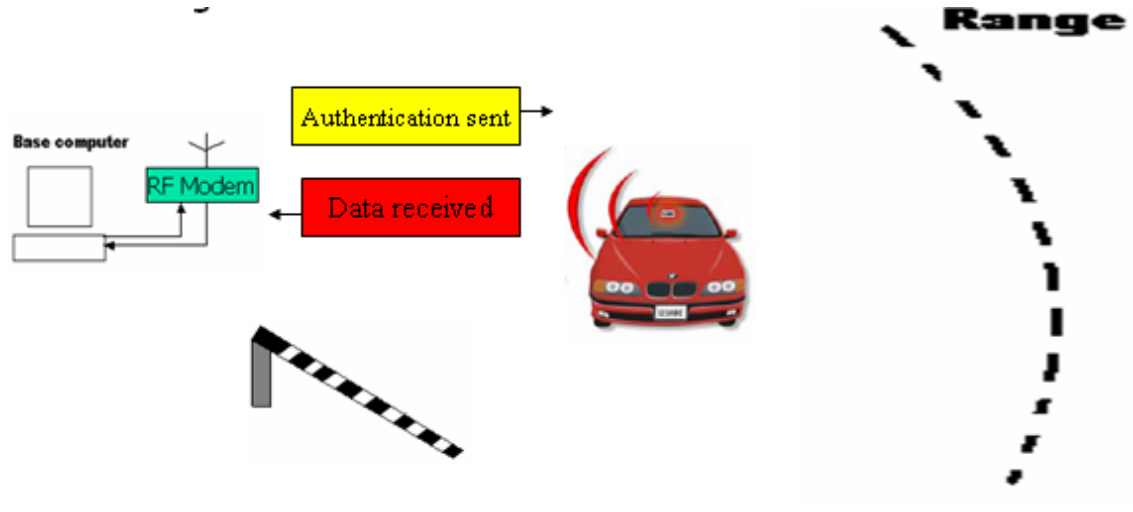


Fig 1.8 Data received by the Base

4. Car module accepts Authentication

The car module receives the authentication and goes into a wait loop till the time it reaches the barrier and sends its notification message.

Fig 1.9 Authentication Accepted

5. Car approaches the base

The car identifies the barrier with the help of an infrared detector ten feet before the barrier. After sensing infrared detection it sends its specific notification message.

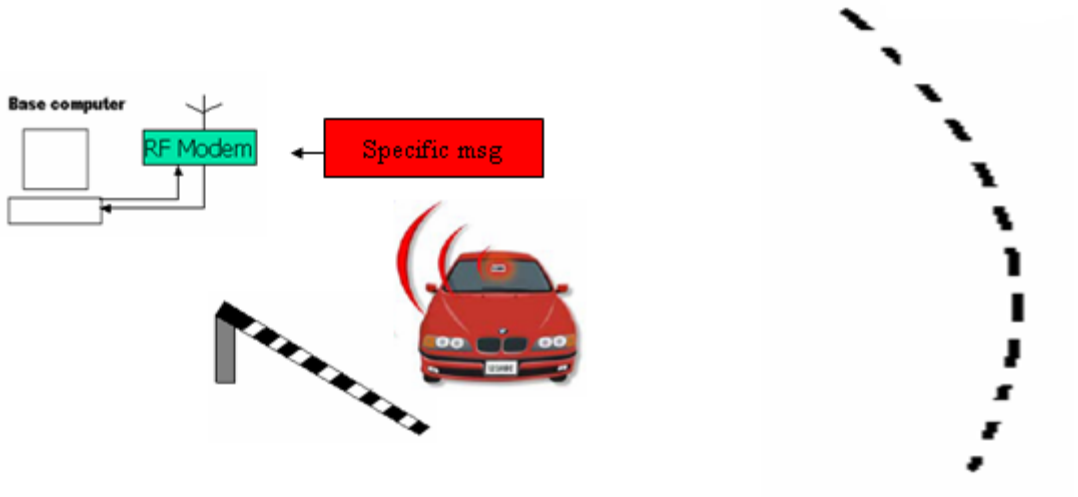


Fig 1.10 Car approaches the barrier

6. Base opens the gate

The base receives that specific message and gives power output at its printer port to control mechanical system such as opening the barrier.

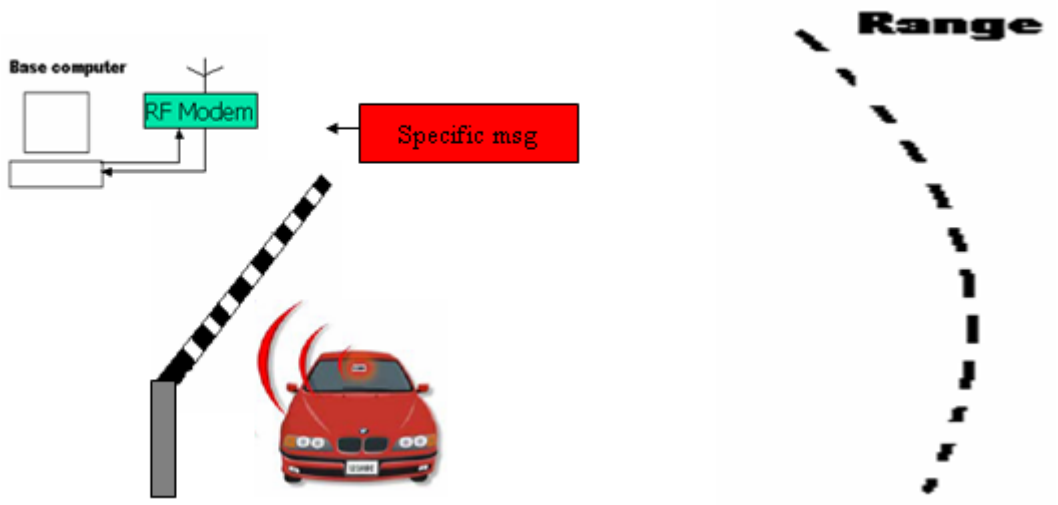


Fig 1.11 Base opens the gate

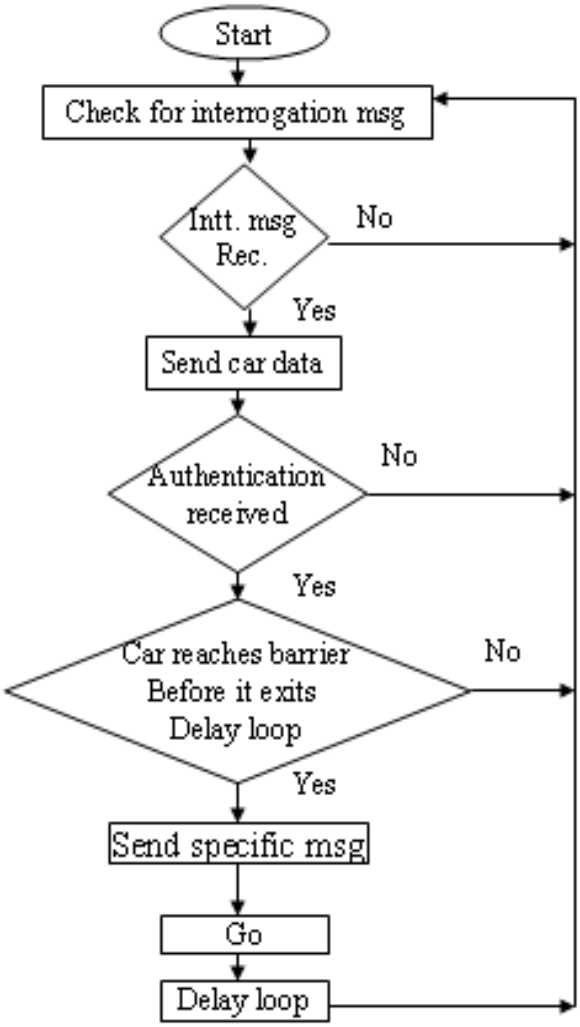
7. Invalid or unregistered car

The base will never open the gate for invalid or unregistered car. If such a car approaches the gate, the base would detect it and ring an alarm.

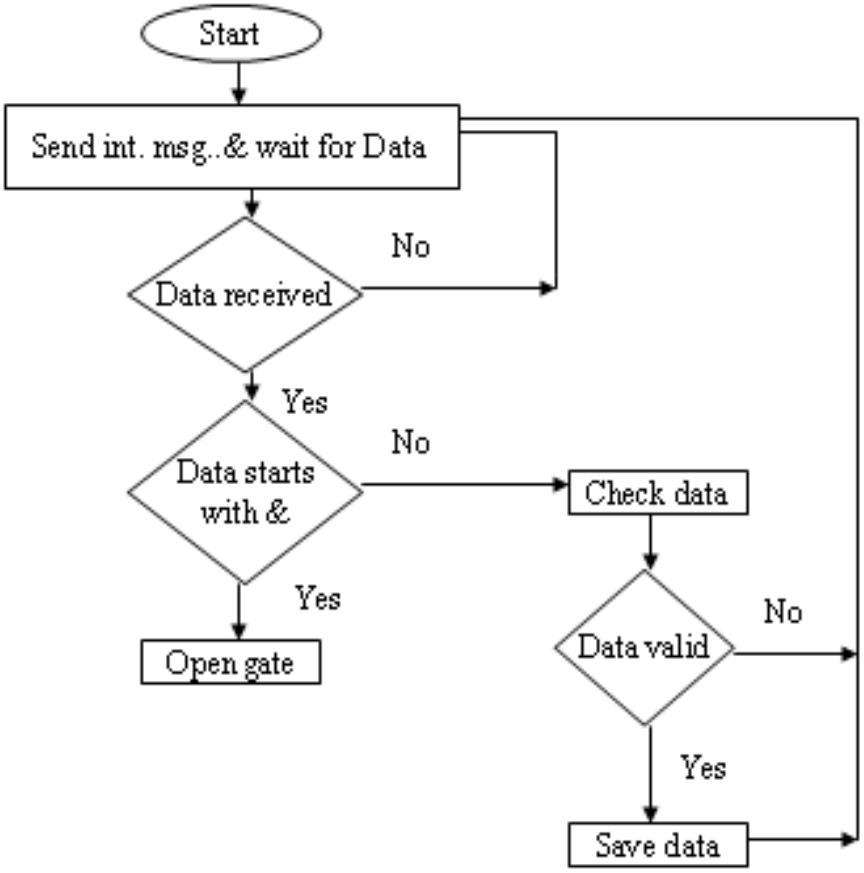
Fig 1.12 Action in case of invalid/ unregistered car

1.4.4 Flow Charts

a. Car Module Operation



b. Base Module Operation


Start
Send int. msg..& wait for Data
Data received
No
Yes
Data starts with &
No
Check data
Yes
Open gate
Data valid
No
Yes
Save data

Chapter 2

Micro controller AT89C52

2.1 Micro controller

The AT89C52 is a low-power, high-performance CMOS 8-bit microcomputer with 8K bytes of Flash programmable and erasable read only memory (PEROM). The device is manufactured using Atmel's high-density non-volatile memory technology and is compatible with the industry-standard 80C51 and 80C52 instruction set and pin out.

The on-chip Flash allows the program memory to be reprogrammed in-system or by a conventional non-volatile memory programmer. By combining a versatile 8-bit CPU with Flash on a monolithic chip, the Atmel AT89C52 is a powerful microcontroller which provides a highly-flexible and cost-effective solution to many embedded control applications.

2.2 Pin Diagram

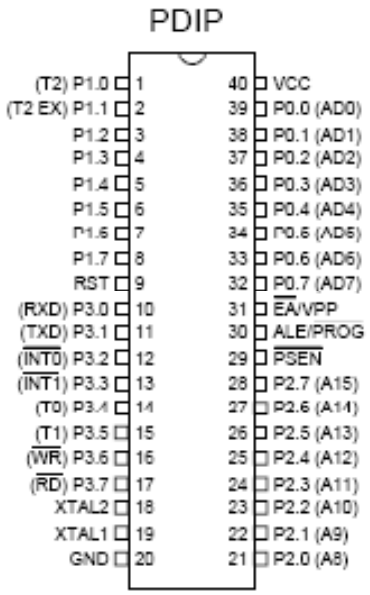


Fig 2.1 Pin diagram

2.3 Block Diagram

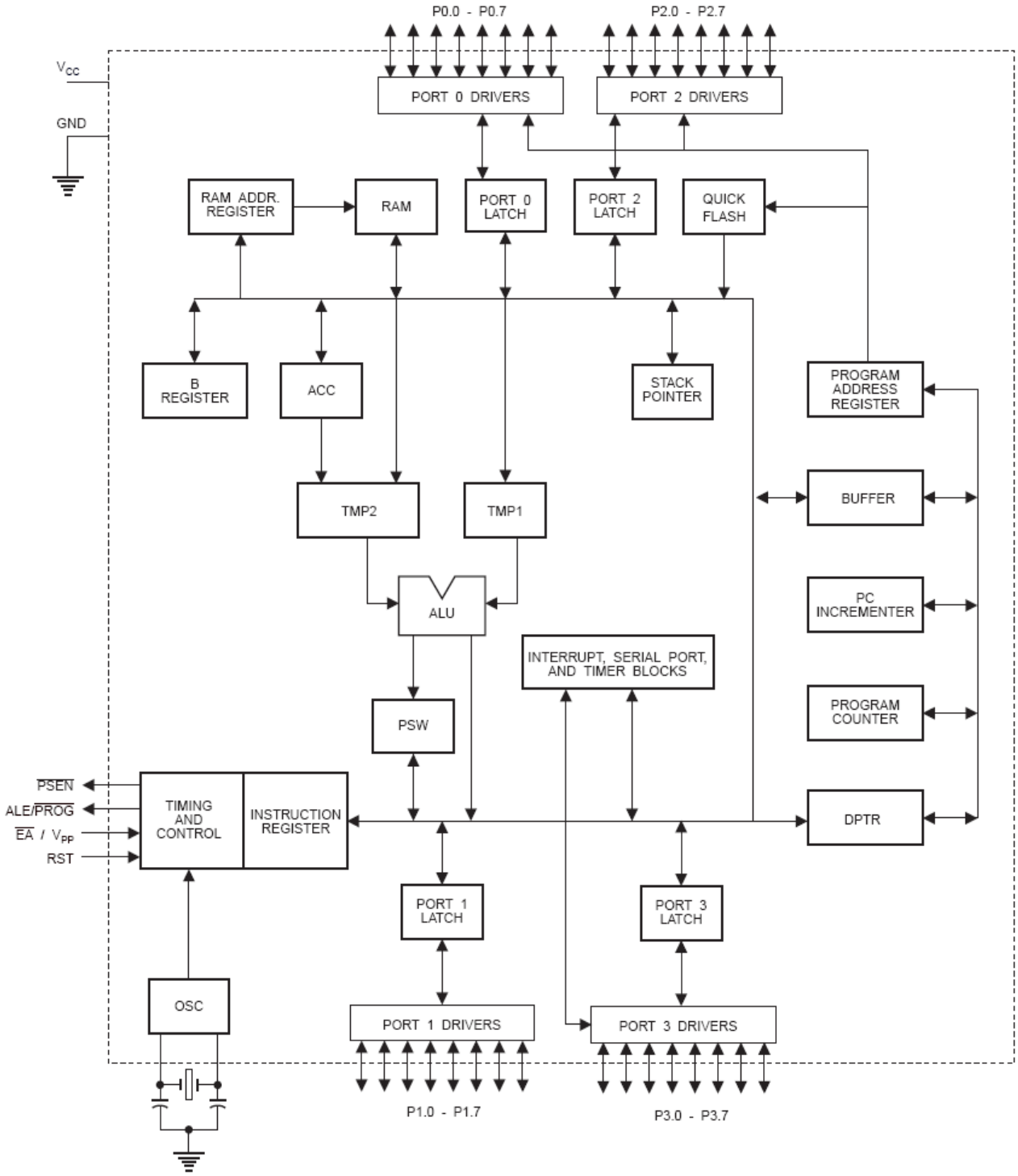


Fig 2.2 Block Diagram

2.4 AT89C52 Description

2.4.1 Features

- Compatible with MCS-51™ Products
- 8K Bytes of In-System Reprogrammable Flash Memory
- Endurance: 1,000 Write/Erase Cycles
- Fully Static Operation: 0 Hz to 24 MHz
- Three-level Program Memory Lock
- 256 x 8-bit Internal RAM
- 32 Programmable I/O Lines
- Three 16-bit Timer/Counters
- Eight Interrupt Sources
- Programmable Serial Channel
- Low-power Idle and Power-down Modes

The AT89C52 provides the following standard features: 8K bytes of Flash, 256 bytes of RAM, 32 I/O lines, three 16-bit timer/counters, a six-vector two-level interrupt architecture, a full-duplex serial port, on-chip oscillator, and clock circuitry. In addition, the AT89C52 is designed with static logic for operation down to zero frequency and supports two software selectable power saving modes. The Idle Mode stops the CPU while allowing the RAM, timer/counters, serial port, and interrupt system to continue functioning. The Power-down mode saves the RAM contents but freezes the oscillator, disabling all other chip functions until the next hardware reset.

2.4.2 Absolute Maximum Ratings

Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of this specification are not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

- Operating Temperature -55 C to +125 C
- Storage Temperature -65 C to +150 C
- Voltage on any Pin with Respect to Ground -1.0 V to +7.0 V
- Maximum Operating Voltage 6.6 V
- DC Output Current 5.0 mA

2.5 Pin Description

VCC

Supply voltage.

GND

Ground.

I. Port 0

Port 0 is an 8-bit open drain bi-directional I/O port. As an output port each pin can sink eight TTL inputs. When 1s are written to port 0 pins, the pins can be used as high-impedance inputs. Port 0 may also be configured to be the multiplexed low-order address/data bus during accesses to external program and data memory. In this mode P0 has internal pull-ups. Port 0 also receives the code bytes

during Flash programming, and outputs the code bytes during program verification. External Pull-ups are required during program verification.

II. Port 1

Port 1 is an 8-bit bi-directional I/O port with internal pull-ups. The Port 1 output buffers can sink/source four TTL inputs. When 1s are written to Port 1 pins, they are pulled high by the internal pull-ups and can be used as inputs. As inputs, Port 1 pins that are externally being pulled low will source current (IIL) because of the internal pull-ups. In addition, P1.0 and P1.1 can be configured to be the timer/counter 2 external count input (P1.0/T2) and the timer/counter 2 trigger input (P1.1/T2EX), respectively, as shown in the following table. Port 1 also receives the low-order address bytes during Flash programming and verification.

III. Port 2

Port 2 is an8-bit bi-directional I/O port with internal pull-ups. The port 2 output buffers can sink/source four TTL inputs. When 1s are written to port 2 pins they are pulled high by the internal pull-ups and can be used as inputs. As inputs, Port 2 pins that are externally being pulled low will source current (IIL) because of the internal pull-ups. Port 2 emits the high-order address byte during fetches from external program memory and during accesses to external data memory that uses 16 bit addresses (MOVX @ DPTR). In this application it uses strong internal pull-ups when emitting 1s. During accesses to external data memory that uses 8-bit addresses (MOVX @ R1), Port 2 emits the contents of the P2 Special Function Register. Port 2 also receives the high-order address bits and some control signals during Flash programming and verification.

IV. Port 3

Port 3 is an 8-bit bi-directional I/O port with internal pull-ups. The Port 3 output buffers can sink/source four TTL inputs. When 1s are written to port 3 pins they are pulled high by the internal pull-ups and can be used as inputs. As inputs, Port 3 pins that are externally being pulled low will source current (IIL) because of the pull-ups. Port 3 also serves the function of various special features of the AT89C52 as listed below.

2.6 Port Pin Alternate Function

- P3.0 RXD (serial input port)
- P3.1 TXD (serial output port)
- P3.2 INT0 (external interrupt 0)
- P3.3 INT1 (external interrupt 1)
- P3.4 T0 (timer 0 external input)
- P3.5 T1 (timer 1 external input)
- P3.6 WR (external data memory write strobe)
- P3.7 RD (external data memory read strobe)

Port 3 also receives some control signals for Flash programming and programming verification.

I. RST Reset Input

A high on this pin for two machine cycles while the oscillator is running resets the device.

II. ALE/PROG

Address Latch Enable output pulse for latching the low byte of the address during accesses to external memory. This pin is also the program pulse input (PROG) during Flash programming. In normal operation ALE is emitted at a constant rate of 1/6 the oscillator frequency, and may be used

for external timing or clocking purposes. Note, however, that one ALE pulse is skipped during each access to external Data Memory. If desired, ALE operation can be disabled by setting bit 0 of SFR location 8EH. With the bit set, ALE is active only during a MOVX or MOVC instruction. Otherwise, the pin is weakly pulled high. Setting the ALE-disable bit has no effect if the micro controller is in external execution mode.

III. PSEN

Program Store Enable is the read strobe to external program memory. When the AT89C51 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.

IV. EA/VPP

External Access Enable, EA must be strapped to GND in order to enable the device to fetch code from external program memory locations starting at 0000H up to FFFFH. Note, however, that if lock bit 1 is programmed, EA will be internally latched on reset. EA should be strapped to VCC for internal program executions. This pin also receives the 12-volt programming enable voltage (VPP) during Flash programming, for parts that require 12-volt VPP designed with static logic for operation down to zero frequency and supports two software selectable power saving modes. The Idle Mode stops the CPU while allowing the RAM, timer/counters, serial port and interrupt system to continue functioning. The Power Down Mode saves the RAM contents but freezes the oscillator disabling all other chip functions until the next hardware reset.

2.7 Oscillator Characteristics

XTAL1 and XTAL2 are the input and output, respectively, of an inverting amplifier, which can be configured for use as an on-chip oscillator, as shown in Figure 2.3. Either a quartz crystal or ceramic resonator may be used. To drive the device from an external clock source, XTAL2 should be left unconnected while XTAL1 is driven as shown in Figure 2. There are no requirements on the duty cycle of the external clock signal, since the input to the internal clocking circuitry is through a divide-by-two flip-flop, but minimum and maximum voltage high and low time specifications must be observed.

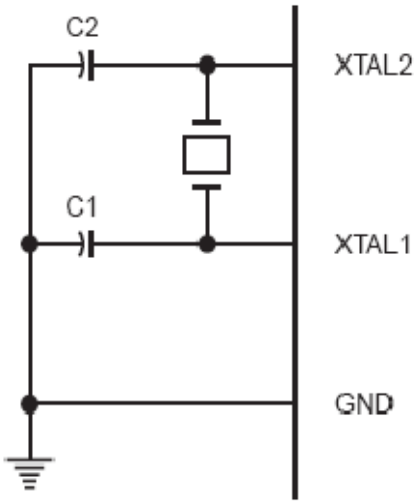


Fig 2.3 Oscillator circuit

2.8 Idle Mode

In idle mode, the CPU puts itself to sleep while all the on-chip peripherals remain active. The mode is invoked by software. The content of the on-chip RAM and all the special functions registers remain unchanged during this mode. The idle mode can be terminated by any enabled interrupt or by a hardware reset. It should be noted that when idle is terminated by a hardware reset, the device normally resumes program execution, from where it left off, up to two machine cycles before the internal reset algorithm takes control. On-chip hardware inhibits access to internal RAM in this

event, but access to the port pins is not inhibited. To eliminate the possibility of an unexpected write to a port pin when Idle is terminated by reset, the instruction following the one that invokes Idle should not be one that writes to a port pin or to external memory.

2.9 Power down mode

In the power-down mode, the oscillator is stopped, and the instruction that invokes power-down is the last instruction executed. The on-chip RAM and Special Function Registers retain their values until the power-down mode is terminated. The only exit from power-down is a hardware reset. Reset redefines the SFRs but does not change the on-chip RAM. The reset should not be activated before VCC is restored to its normal operating level and must be held active long enough to allow the oscillator to restart and stabilize.

2.10 Programming the Flash

The AT89C52 is normally shipped with the on-chip Flash memory array in the erased state (that is, contents =FFH) and ready to be programmed. The programming interface accepts either a high-voltage (12-volts) or a low-voltage (VCC) program enable signal. The low voltage programming mode provides a convenient way to program AT89C52 inside the user's system, while the high voltage programming mode is compatible with conventional third party Flash or EPROM programmers.

The AT89C52 is shipped with either the high-voltage or low-voltage programming mode enabled. The AT89C52 code memory array is programmed byte by byte in either programming mode. To program any non-blank byte in the on-chip Flash Memory, the entire memory must be erased using the Chip Erase Mode.

2.11 Chip Erase

The entire Flash array is erased electrically by using the proper combination of control signals and by holding ALE /PROG low for 10 ms. The code array is written with all “1”s. The chip erase operation must be executed before the code memory can be reprogrammed.

Chapter 3

Serial port and communication standards

3.1 What is a serial port?

The UART serial port (or just "serial port for short") is an I/O (input/output) device.

An I/O is just a way to get data into and out of a computer. There are many types of I/O Devices such as serial ports, parallel ports, and disk drive controllers. Ethernet boards, universal serial buses, etc. most PC's have one or two serial ports. Each has a 9-pin connector (sometimes 25 pins) on the back of the computer. Computer programs can send data (bytes) to the transmit pin (output) and receive bytes from the receive pin (input). The other pins are for control purposes and ground.

The serial port is much more than a connector. It converts the data from parallel to serial and change the electrical representation of the data. Inside the computer, data bit flow in parallel (using many wires at the same time). Serial flow is a stream of bits over a single wire (such as on the transmit and receive pin of the serial connector).for the serial port to create such a slow, it must convert data from parallel (inside the computer)to serial on the transmit pin (and conversely).

Most of the serial port is found in a computer chip (or a part of a chip) known as a UART.

3.2 Pins and Wires

Old pc used 25-pin connector but only about 9 pin were actually used so today most connectors are only 9 pin. Each of the 9 pins usually connects to a wire. Besides the two wires used for transmitting and receiving data, another pin (wire) is signal ground. The voltage on any wire is measures with respect to this ground. Thus the minimum number of wires to use for 2-way transmission of data is 3.

Except that it has been known to work with no signal ground wire but with de-graded performances and sometimes with errors.

There are still more wires which are for control purposes (signalling) only and not for sending bytes. All of these signals could have been shared on a single wire, but instead, there is a separate dedicated wire for every type of signal. Some (or all) of these control wires are called “modem control lines”. Modem control wires are either in the asserted state (on) of +12 volts or in the negated state (off) of-12 volts. One of these wires is to signal the computer to stop sending bytes out the serial ports cable. Conversely, another wire signals the device attached to the serial port to stop sending bytes to the computer. If the attached device is a modem, other wires may tell the modem to hang up the telephone line or tell the computer that a connection has been made or that the telephone line is ringing (someone is attempting to call in).

The serial port (not the USB) is usually a RS-232-C, or EIA-232-D, or EIA-232-E. These three are almost the same thing. The original RS (recommended standard) prefix became EIA (electronic industries association) and later EIA/TIA after EIA merged with TIA (telecommunications industries association). The EIA-232 spec provides also for synchronous (sync) communication but the hardware to support sync is almost always missing on PC’s. The RS designation is obsolete but is still widely used. EIA will be used in this how to. Some documents use the full EIA/TIA designation.

3.3 IO Address & IRQ

Since the computer needs to communicate with each serial port, the operating system must know that each serial port exists are where it is (its I/O address). It also needs to know which wire (IRQ number) the serial port must use to request service from the computer’s central processing unit. It requests service by sending an interrupt on this wire. Thus every serial port device must store I its

non-volatile memory both its I/O address and its interrupt request number: IRQ. For the PCI bus it doesn't work exactly this way since the PCI bus has its own interrupts to IRQ's, it seemingly behaves just as described above except that sharing of interrupts is allowed (2 or more devices may use the same IRQ number).

I/O addresses are not the same as memory addresses. When an I/O addresses is put onto the computers address bus, another wire is energized. This both tells main memory to ignore the address and tells all devices, which have I/O addresses (such as the serial port) to listen to he address to see if it matches the devices. If the address matches, then the I/O device reads the data on the data bus.

3.4 Names: Com1, Com2, etc.

The serial ports are named COM1, COM2, etc. in DOS/windows.

3.5 Interrupts

When the serial port receives a number of bytes (may be set to 1,4,8, or 14) into its FIFO buffer, it signals the central processing unit to fetch them by sending an electrical signal known as an interrupt on a certain wire normally used only by that port. Thus the FIFO waits until it has received a number of bytes and then issues an interrupt.

However, this interrupt will also be sent if there is an unexpected delay while waiting for the next byte to arrive (known as timeout). Thus if the bytes are being received slowly (such as from someone typing on a terminal keyboard) there may be an interrupt issued for every byte received. For some UART chips the rule is like this: if 4 bytes in row could have been received in an interval of time, but none of these 4 shows up, then the port gives up waiting for more bytes and issues an interrupt to fetch the bytes currently in the FIFO. Of course, if the FIFO is empty, no interrupt will be issued.

Each interrupt conductor (inside the computer) has a number (IRQ) and the serial port must know which conductor to use to signal o. for example, ttyS0 normally uses IRQ number 4 known as IRQ4 (or IRQ 4). A list of them and more will be found in “man set serial” (search for “configuring serial ports”). Interrupts are issued whenever the serial ports needs to get the central processing units attention. It’s important to do this in a timely manner since the buffer inside the serial port can hold only 16 incoming bytes. If the central processing unit fails to remove such received bytes promptly, then there will not be any space left for any more incoming bytes and small buffer may overflow (overrun) resulting in a loss of data bytes. There is no flow control to prevent this.

Interrupts are also issued when the serial port has just sent out all of its bytes from its small transmits FIFO buffer out the external cable. It then has space for 16 more outgoing bytes. The interrupt is it notifies the central processing unit of that fact so that it may put more bytes in the small transmit buffer to be transmitted. Also, when a modem control line changes state, an interrupt issued.

The buffers mentioned above all hardware buffers. The serial port also has large buffers in main memory. This will be explained later. Interrupts convey a lot of information but only indirectly. The interrupt itself just tells a chip called interrupt controller that a certain serial port needs attention. The interrupt controller then signals the central processing unit. The central processing unit then runs a special program to service the serial port. That program is called an interrupt service routine (part of the serial driver software).it tries to find out what has happened to serial port then deals with the problem such as transferring bytes from (or to) the serial port’s hardware buffer. This program can easily find out what has happened since the serial port has a registers at I/O addresses known to the serial driver software. These registers contain status information about the serial port. The software read these registers and by inspecting the contents, finds out what has happened and takes appropriate action.

3.6 Data Flow (Speeds)

Data (bytes representing letters, pictures etc) flows into and out of serial port. Flow rates such as (56K (56000) bits/sec) are (incorrectly called) “Speed” but almost everyone say speed instead of “flow rate”. Its important to understand that average speed is often less than the specified speed. Waits (or idle time) result in lower average speed. These waits include long waits or perhaps a second due to flow control. At the other extreme there may be very short waits (idle time) of several microseconds between bytes. If the device on the serial port (such as modem) can’t accept the full serial port speed, then the average speed must be reduced.

3.7 Flow Control

Flow control means the ability to slow down the flow of bytes in a wire. For serial ports this means the ability to stop and then restart the flow without any loss of bytes. Flow control is needed for modems and the other hardware to allow a jump in instantaneous flow rates. The flow control may be from computer to modem or it may be from modem to computer each direction involves three buffers.

1. In the modem
2. In the UART chip (called FIFO) and
3. In main memory managed by the serial driver.

Flow control protects all buffers (except the FIFO) from overflowing.

3.7.1 Symptoms of No flow control

Understanding flow-control theory can be of practical use. The symptom of no flow control is that chunks of data missing from files sent without the benefits of flow control. When overflow happens, often hundreds or even thousands of bytes get lost, and all in contiguous chunks.

3.7.2 Hardware vs. software flow control

If feasible, it's best to use "hardware"2flow control that uses two dedicated "modem control" wires to send the "stop" and "start" signals. Hardware flow control at the serial ports works like this; the two pins, RTS (request to send) and CTS (clear to send) are used. When the computer is ready to receive date it asserts RTS by putting a positive voltage on the RTS pin (meaning "request to send to me"). When the computer is not able to receive any more bytes, it negates RTS by putting a negative voltage on the pin saying: "stop sending to me". The serial cable to another pin on the modem, printer, and terminal connects the RTS pin, etc. this other pins only function is to receive the signal.

For the case of the modem this "other" pin will be the modem RTS pin but for a printer, another pc, or a non modem device, it is usually a CTS pin so a "crossover" or "null modem" cable is required. This cable connects the CTS pin at one end with the RTS pin at the other end (two wires since each end of the cable has a CTS pin). For a modem, a straight-thru cable is used.

For the opposite direction of flow similar scheme is used. For a modem, the CTS pin is used to send the flow control signal to the CTS pin on the PC. For a non- modem, the RTS pins send the signal. Thus modems and non-modems have the role of their RTS and CTS pin interchanged. Some non-modems such as dumb terminals may use other pins for flow control such as the DTR pin instead of RTS.

Software flow control uses the main receive and transmit wires to send the start and stop signals. It uses the ASCII control characters DC1 (start) and DC3 (stop) for this purpose. They are just inserted into the regular stream of data. Software flow control is not only slower in reacting but also doesn't allow the sending of binary data unless special precautions are taken. Since binary

data will likely contain DC1 and DC3, special means must be taken o distinguish between a DC3 that means a flow control stop and a DC3 that is part of the binary code. Likewise for DC1.

3.8 Data flow path Buffers

It has been mentioned that there are three buffers for each direction of flow (3 pair altogether):

1. 16 byte FIFO buffers (in the UART)
2. A pair of larger buffers inside a device connected to the serial port (such as a modem).
3. A pair of buffers (say 8k) in main memory.

When an application programs send bytes to the serial port they first get stashed in the transmit serial port buffer in main memory. The other member of the pair consist of a receive buffer for the opposite direction of byte-flow. Here's an example diagram for the case of browsing the Internet with a browser. Transmit data flow is left to right wile receive flow is right to left. There is a separate buffer for each direction of flow.

Application 8k-byte 16-byte 1k-byte tele-

BROWSER------MEMORY-------FIFO--------MODEM--------PHONE

Program buffer buffer buffer line

For the transmit case, the serial device driver takes out ay 16 bytes from this transmit buffer (in main memory), one byte at a time and puts them into 16 byte transmit buffer in the serial UART

for transmission. Once in that transmit buffer, there is no way to stop them from being transmitted. They are then transmitted to the modem or (other device connected to the serial port) which also has a fair sized (say 1k) buffer. When the device driver (on orders from flow control) stops the flow of outgoing bytes from the computer, what it actually stops is the flow of outgoing bytes from the large transmit buffer in main memory. Even after this has happened and the flow to the modem has stopped, an application program may keep sending bytes to the 8k transmit buffer until it becomes fill. At the same time, the bytes stored in FIFO and MODM continue to be sent out until these buffers empty.

When the memory buffer gets fill, the application program cant send any more bytes to it (a "write" statement in a c_ program blocks) and the application program temporarily stops running and waits until some buffer space becomes available. Thus a flow control "stop" is ultimately able to stop the program that is sending the bytes. Even though this program stops, the compute does not necessarily stop computing since it may switch to running other processes while its waiting at a flow control stop.

The above was little over simplified in three ways. First, some UARTs can do automatic hardware flow control, which can stop the transmission out of the FIFO buffers if needed (not yet supported by Linux). Second, while an application process is waiting to write to the transmit buffer, it could possibly perform other tasks. Third the serial driver (located between the memory buffer and the FIFO) has its own buffer (in main memory)used to process characters.

3.9 Pin Out and signals

PINOUT of the serial port (→ direction is out of PC)

(Note DCD is sometimes labelled CD)

Pin # 9-pin	Pin # 25-pin	Acronym	full name	direction	what-it-may-do/mean
3	2	T*D	transmit data	→	transmit bytes out of PC
2	3	R*D	receive data	←	receive bytes into PC
7	4	RTS	request to send	→	RTS/CTS flow control
8	5	CTS	clear to send	←	RTS/CTS flow control
6	6	DSR	data set ready	←	I'm ready to communicate
4	20	DTR	data terminal ready	→	I'm ready to communicate
1	8	DCD	data career detect	←	modem connected to another
9	22	RI	ring indicator	←	telephone line ringing
5	7	SG	signal ground		

3.10 Signals may have no fixed meaning

Only 3 of the 9pins have a fixed assignment: transmit, receive and signal ground. This is fixed by the hardware and you can't change it. But the other signal lines are controlled by software and may do (and mean) almost anything at all. However they can only be in one of two states: asserted (+12 volts) or negated (-12 volts). Asserted is "on" and Negated is "off". For example, Linux software may command that DTR be negated and the hardware only carries out. This Command and puts –12 volts on the DTR pin. Modem (or other device) that receives the DTR signal may do receive the various things. If a modem has been configured a certain way it will hang up the telephone line when DTR is negated. In other cases it may ignore this signal or do something else when DR is negated (turned off).

It's like this for all the 6 signal lines. The hardware only sends and receives the signals, but what action (if any) they perform up to the Linux software and the configuration/design of devices that

you connect to the serial port. However, most pins have certain functions, which they normally perform but this may vary with the operating system and the device driver configuration. Under Linux, one may modify the source code to make these signal lines behave differently (some people have).

3.11 Cabling Between Serial Ports

A cable from a serial port always connects to another serial port. An external modem or other device that connects to the serial port has a serial port built in to it. For modems, the cable is always straight thru: pin 2 goes to pin 2 etc. the modem is said to be DCE (data communication equipment) and the computer is said to be DTE (data terminal equipment).

Thus for connecting DTE-to-DCE you use straight- thru cable. For connecting DTE-to-DTE you must use a null-modem cable (also called a crossover cable). Her are many ways to wire such cable (see examples in text –terminal –HOWTO subsection: “direct cable connection”)

There are good reasons why it works this way. One reason is that the signals are unidirectional. If 2 pins send a signal out of it (but is unable to receive any signal) then obviously you can’t connect it to pin 2 of the same type of device. If you did, they would both send out signals on the same wire to each other but neither would be able to receive any signal. There are two ways to deal with the situation. One way is to have two different types of equipment where pin 2 of the first type sends the signal to pin 2 of the second type (which receives the signal). That’s the way its done when you connect a PC (DTE) to a modem (DCE). There’s a second way to do this without having two different types of equipment. That’s the way it’s done when you connect 2 PCs together or a PC to a terminal (DTE-to-DTE). The cable used for this is called a null-modem cable since it connects two PCs without use of a modem. A null-modem cable may also be called a cross over cable since

the wires between pin 2 and 3 crosses over each other (if you draw them on a sheet of paper). The above example is for a 25-pin connector but for a 9-pin connector the pin numbers are just the opposite.

The serial pin designations were originally intended for connecting a dumb terminal to a modem. The terminal was DTE (data terminal equipment) and the modem was DCE (data communication equipment). Today the PC is usually used as DTE instead of a terminal (but real terminals may still be used this way). The names of the pins are same on both DTE and DCE. The words: “receive” and “transmit” are from the “point of view” of the PC (DTE). The transmit pin from the PC transmits to the “transmit” pin of the modem (but actually the modem is receiving the data from this pin so from the point of view of the modem it would be a receive pin).

The serial port was originally intended to be used for connecting DTE to DCE, which makes cabling simple: just use a straight thru cable. Thus when one connects a modem one seldom needs to worry about which pin is which. But people wanted to connect DTE to DTE (for example a computer to a terminal) and various ways were found to do this by fabricating various types of special null ode cables. In this case what pin connects to what pin becomes significant.

3.12 RTS/CTS and DTR/DSR Flow Control

This is “hardware” flow control. Flow control was previously explained in the flow control subsection but the pins and voltage signals were not. Only RTS/CTS flow control will be discussed since DTR/DSR flow control works the same way. To get RTS/CTS flow control one needs to either select hardware flow control in an application program.

The when a DTE (such as a PC) wants to stop the flow into it, it negates RTS. Negated “request to send” (-12volts) means, “request not to send to me “ (stop sending). When the PC is ready for more bytes it asserts RTS (+12 volts) and the flow o bytes to it resumes. Flow control signal are always

sent in a direction opposite to the flow of byte that is being controlled equipment (modems) works the same way but sends the stop signal out the CTS pin. Thus its RTS/CTS flow control using 2 lines.

On what pins is this stop signal received? That depends on whether we have a DCE-DTE connection or DTE-DTE connection. For DCE-DTE it's a straight thru connection so obviously the signal is received on the pin with the same name as the pin it's sent out from. Its RTS→RTS (PC to modem) and CTS ←CTS (modem to PC). For DTE-to- DTE the connection is also easy o figure out. The RTS pins always sends and the CTS pin always receives. Assume that we connect two PCs (PC1 and PC2) together via their serial ports. Then its RTS (PC1) →CTS (PC2) and CTS (PC1)←RTS (PC2). In other words RTS and CTS cross over. Such a cable (with other signals crossed over as well) is called a "null modem" cable.

What is sometimes confusing is that there is a original use of RTS where it means about the opposite of the previous explanation above. This original meaning is: I request to send to you. This request was intended to be sent from a terminal (or computer) to a modem which, if it decided to grant the request, would send back an asserted CTS from its CTS pin in to the CTS pin of the computer: You are cleared to send to me. Note that in contrast to the modern RTS/CTS bi-directional flow control, this only protects the flow in one direction: from the computer (or terminal) to the modem. This original use appears to be little used today on modern equipment (including modems).

3.13 The DTR and DSR Pins

Just like RTS and CTS, these pins are paired. For DTE-to-DTE connections they are likely to cross over. There are two ways to use these pins. One way is to use them as a substitute for RTS/CTS flow control. The DTR pin is just like the RTS pin while the DSR pin behaves like the CTS pin.

Although Linux doesn't support DTR/DSR flow control, it can be obtained by connecting the RTS/CTS pins at the PC to the DSR/DTR pins at the device that uses DTR/DSR flow control. DTR slow control is the same as DTR/DSR flow control but its only one-way and only uses the DTR pin at the device. Many text terminals and some printers use DTR/DSR (or just DTR) flow control. In the future, Linux may support DTR/DSR flow control. the software has already been written but its not clear when (or if)it will incorporated into the serial driver.

The normal use of DTR/DSR (not for flow control) is as follows: a device asserting DTR says that it powered on and ready to operate. For a modem, the meaning of a DTR signal from the Pc depends on how the modem is configured. Negating DTR is sometimes called "hanging up" but it does not always do this. One way to hang up (negate DTR) is to set the baud rate to 0using the command "sty 0". Trying to do this from a "foreign "terminal may not work due to thee two-interface problem. See two interfaces at a terminal. For internal modem-serial ports it worked OK with a port using minicom but didn't work if the port was using wvdial.why?

3.14 Introduction to UARTS

UARTs (universal asynchronous receiver transmitter) are serial chips on your PC motherboard (or o an internal modem card). The UART function may also be done on a chip that does other things as well. On older computers like many 486's, the chips were on the disk IO controller card. Still order computer have dedicated serial boards.

The UARTs purpose is to convert bytes from the PC's parallel bus to a serial bit-stream. The cable going out of the serial port is serial and has only one wire for each direction of flow. The serial port sends out a stream of bits, one bit at a time. Conversely, the bit stream that enters the serial port via the external cable is converted to parallel bytes that the computer can understand. UARTs deals with data in byte sized pieces, which Is conveniently also the size of ASCII characters.

So you have a terminal hooked up to your PC. When you type a character, the terminal gives the character to its transmitters (also a UART). The transmitter sends that byte out onto the serial line, one bit at a time, at a specific rate. On the PC end, the receiving UART takes all the bits and rebuilds the (parallel) byte and puts it in a buffer.

Along with converting between serial and parallel, the UART does some other things as a by product (side effect) of its primary task. The voltage used to represent basis also converted (changed). Extra bits (called start and stop bits) are added to each byte before it is transmitted.

See the serial-HOWTO section, “voltage wave shapes” for details. Also, while the flow rate (in bytes/sec) on the parallel bus inside the computer is very high, the flow rate out the UART on the serial port side of it is much lower. The UART has a fixed set of rates (speeds), which it can use, at its serial port interface.

3.14.1 Types of UARTs

There are two basic types of UARTs: dumb UARTS and FIFO UARTS. Dumb UARTs are the 8250, 16450, early 16550, and early 16650. They are obsolete but if you understand how they work it's easy to understand how the modern ones work with FIFO UARTS (late 16550, 16550A, 16c552, late 16650, 16750, and 16C950).

There is some confusion regarding 16550. Early models had a bug and worked properly only as 16450's (no FIFO). Later models with the bug fixed were named 16550A but many manufacturers did not accept the name change and continued calling it a 16550. Most all 16550's in use today are like 16550A's. Linux will report it as being a 16550A even though your hardware manual (or a label note) says it's a 16550. A similar situation exists for the 16650 (only it's worse since the

manufacturer allegedly didn't admit anything was wrong). Linux will report a late 16650 as being a 16650V2. If it reports it as 16650 it is bad news and only is used as if it had a one-byte buffer.

3.15 FIFOs

To understand the differences between dumb and FIFO (First In, First Out queue discipline) first let's examine what happens when a UART has sent or received a byte. The UART itself can't do anything with the data passing thru it, it just receives and sends it. For the original dumb UARTS, the CPU gets an interrupt from the serial device every time a byte has been sent or received. The CPU then moves the received byte out of the UART's buffer and into memory somewhere, or gives the UART another byte to send. The 8250 and 16450 UARTs only have a 1 byte buffer. That means that every time 1 byte is sent or received, the CPU is interrupted. At low transfer rates, this is OK. But, at high transfer rates, the CPU gets so busy dealing with the UART, that is doesn't have time to adequately tend to other tasks. In some cases, the CPU does not get around to servicing the interrupt in time, and the byte is overwritten, because they are coming in so fast. This is called an "overrun" or "overflow". That's where the FIFO UARTs are useful. The 16550A (or 16550) FIFO chip comes with 16 byte FIFO buffers. This means that it can receive up to 14 bytes (or send 16 bytes) before it has to interrupt the CPU. Not only can it wait for more bytes, but the CPU then can transfer all 14 (or more) bytes at a time. This is a significant advantage over the other UARTs, which only have 1 byte buffers. The CPU receives less interrupts, and is free to do other things. Data is not lost, and everyone is happy. Note that the interrupt threshold of FIFO buffers (trigger level) may be set at less than 14. 1, 4 and 8 are other possible choices.

As of late 2000 there was no way the Linux users could set these directly(set serial can’t do it)While many pc’s only have a 16550 with 16 bytes buffers, better UARTS have been larger buffers.

Note that the interrupt is issued slightly before the buffer gets full (at say a "trigger level" of 14 bytes for a 16-byte buffer). This allows room for a few more bytes to be received during the time that the interrupt is being serviced. The trigger level may be set to various permitted values by kernel software. A trigger level of 1 will be almost like a dumb UART (except that it still has room for 15 more bytes after it issues the interrupt).

If you type something while visiting a BBS, the characters you type go out thru the serial port. Your typed characters that you see on the screen are what was echoed back thru the telephone line thru your modem and then thru your serial port to the screen. If you had a 16-byte buffer on the serial port which held back characters until it had 14 of them, you would need to type many characters before you could see what you typed (before they appeared on the screen). This would be very confusing but there is a "timeout" to prevent this. Thus you normally see a character on the screen just as soon as you type it.

The "timeout" works like this for the receive UART buffer: If characters arrive one after another, then an interrupt is issued only when say the 14th character reaches the buffer. But if a character arrives and the next character doesn't arrive soon thereafter, then an interrupt is issued. This happens even though there are not 14 characters in the buffer (there may only be one character in it). Thus when what you type goes thru this buffer, it acts almost like a 1-byte buffer even though it is actually a 16-byte buffer (unless your typing speed is a hundred times faster than normal). There is also "timeout" for the transmit buffer as well.

3.16 Why FIFO buffers are small

You may wonder why the FIFO buffers are not larger. After all, memory is cheap an it wouldn’t cost much more to use buffers in the kilobyte range. The reason is flow control. Flow control stops the flow of data on the serial line when necessary. If the stop signal is sent to serial port, then the

stop request is handled by the software .the serial port hardware knows nothing about this flow control.

If the serial port buffer contains 64 bytes ready to send when it receives a flow control signal to stop sending, it will send out the 64 bytes anyway in violation of the stop request. There is no stopping it since it doesn't know about flow control. If the buffer was large, then many more bytes would be sent in violation of flow control's request to stop.

3.17 UART Model Numbers

Here's a list of some UARTs. TL *is* T*rigger* L*evel*

- 8250, 16450, early 16550: Obsolete with 1-byte buffers
- 16550, 16550A, 16C552: 16-byte buffers, TL=1,4,8,14; 115.2 kbps standard, many support 230.4 or 460.8 kbps
- 16650: 32-byte buffers. 460.8 kbps
- 16750: 64-byte buffer for send, 56-byte for receive. 921.6 kbps
- 16850, 16C850: 128-byte buffers. 460.8 kbps or 1.5 mbps
- 16950
- Hayes ESP: 1k-byte buffers

For V.90 56k modems, it may be a several percent faster with a 16650 (especially if you are downloading large uncompressed files). The main advantage of the 16650 is its larger buffer size as the extra speed isn't needed unless the modem compression ratio is high. Some 56k internal modems may come with a 16650 ??

Non-UART and intelligent multi port boards use DSP chips to do additional buffering and control, thus relieving the CPU even more. For example, the Cyclades Cyclom, and Stallion Easy IO boards

use a Cirrus Logic CD1400 RISC UART, and many boards use 80186 CPUs or even special RISC CPUs, to handle the serial IO.

3.18 Introduction to communication standards

Line drivers and receivers are commonly used to exchange data between two or more points (nodes) on a network. Reliable data communications can be difficult in the presence of induced noise, ground level differences, impedance mismatches, failure to effectively bias for idle line conditions, and other hazards associated with installation of a network.

The connection between two or more elements (drivers and receivers) should be considered a transmission line if the rise and/or fall time is less than half the time for the signal to travel from the transmitter to the receiver. Standards have been developed to insure compatibility between units provided by different manufacturers, and to allow for reasonable success in transferring data over specified distances and/or data rates. The Electronics Industry Association (EIA) has produced standards for RS485, RS422, RS232, and RS423 that deal with data communications. Suggestions are often made to deal with practical problems that might be encountered in a typical network. EIA standards where previously marked with the prefix "RS" to indicate recommended standard; however, the standards are now generally indicated as "EIA" standards to identify the standards organization. While the standards bring uniformity to data communications, many areas are not specifically covered and remain as "gray areas" for the user to discover (usually during installation) on his own.

3.18.1 Single-ended Data Transmission

Electronic data communications between elements will generally fall into two broad categories: single-ended and differential. RS232 (single-ended) was introduced in 1962, and despite rumours for

its early demise, has remained widely used through the industry. The specification allows for data transmission from one transmitter to one receiver at relatively slow data rates (up to 20K bits/second) and short distances (up to 50Ft. @ the maximum data rate). Independent channels are established for two-way (full-duplex) communications. The RS232 signals are represented by voltage levels with respect to a system common (power / logic ground). The "idle" state (MARK) has the signal level negative with respect to common, and the "active" state (SPACE) has the signal level positive with respect to common. RS232 has numerous handshaking lines (primarily used with modems), and also specifies a communications protocol. In general if you are not connected to a modem the handshaking lines can present a lot of problems if not disabled in software or accounted for in the hardware (loop-back or pulled-up). RTS (Request to send) does have some utility in certain applications. RS423 is another single ended specification with enhanced operation over RS232; however, it has not been widely used in the industry.

3.18.2 Differential Data Transmission

When communicating at high data rates, or over long distances in real world environments, single-ended methods are often inadequate. Differential data transmission (balanced differential signal) offers superior performance in most applications. Differential signals can help nullify the effects of ground shifts and induced noise signals that can appear as common mode voltages on a network. RS422 (differential) was designed for greater distances and higher Baud rates than RS232. In its simplest form, a pair of converters from RS232 to RS422 (and back again) can be used to form an "RS232 extension cord." Data rates of up to 100K bits / second and distances up to 4000 Ft. can be accommodated with RS422. RS422 is also specified for multi-drop (party-line) applications where only one driver is connected to, and transmits on, a "bus" of up to 10 receivers.

While a multi-drop "type" application has many desirable advantages, RS422 devices cannot be used to construct a truly multi-point network. A true multi-point network consists of multiple drivers and receivers connected on a single bus, where any node can transmit or receive data. "Quasi" multi-drop networks (4-wire) are often constructed using RS422 devices. These networks are often used in a half-duplex mode, where a single master in a system sends a command to one of several "slave" devices on a network. Typically one device (node) is addressed by the host computer and a response is received from that device. Systems of this type (4-wire, half-duplex) are often constructed to avoid "data collision" (bus contention) problems on a multi-drop network (more about solving this problem on a two-wire network in a moment).

RS485 meets the requirements for a truly multi-point communications network, and the standard specifies up to 32 drivers and 32 receivers on a single (2-wire) bus. With the introduction of "automatic" repeaters and high-impedance drivers / receivers this "limitation" can be extended to hundreds (or even thousands) of nodes on a network. RS485 extends the common mode range for both drivers and receivers in the "tri-state" mode and with power off. Also, RS485 drivers are able to withstand "data collisions" (bus contention) problems and bus fault conditions.

To solve the "data collision" problem often present in multi-drop networks hardware units (converters, repeaters, micro-processor controls) can be constructed to remain in a receive mode until they are ready to transmit data. Single master systems (many other communications schemes are available) offer a straight forward and simple means of avoiding "data collisions" in a typical 2-wire, half-duplex, multi-drop system. The master initiates a communications request to a "slave node" by addressing that unit. The hardware detects the start-bit of the transmission and automatically enables (on the fly) the RS485 transmitter. Once a character is sent the hardware reverts back into a receive mode in about 1-2 microseconds (at least with R.E. Smith converters, repeaters, and remote I/O boards).

Any number of characters can be sent, and the transmitter will automatically re-trigger with each new character (or in many cases a "bit-oriented" timing scheme is used in conjunction with network biasing for fully automatic operation, including any Baud rate and/or any communications specification, example . 9600,N, 8,1). Once a "slave" unit is addressed it is able to respond immediately because of the fast transmitter turn-off time of the automatic device. It is NOT necessary to introduce long delays in a network to avoid "data collisions." Because delays are NOT required, networks can be constructed, that will utilize the data communications bandwidth with up to 100% through put.

Standards have been developed to define how serial ports should work, and how the data exchange should be controlled. The ports we see offered are named for the Electronic Industry Association (EIA) standards they are designed around. RS232, RS422, and RS485 are often listed as options when communication ability is needed

RS232 is the oldest standard, developed by the EIA in the 1970's primarily for use in connecting computers to modems and terminals. They must have done a darned fine job, because it is still the most prolific communication port in existence. RS232 is rated for point-to-point data communication at speeds below 20,000 bits per second (bps or baud) and distances up to 50 feet. The standard also defines a 25-pin D-connector and assigns functions to each pin for control of data flow. Since cable capacitance and electrical noise are the limiting factors for speed and distance, we often exceed the standard's limits by using shielded low-capacitance cable. It's not unusual to see RS232 links running up to 200 feet at 9600 baud. If distances are kept short (5 feet or less), speeds up to 115,000 baud are not uncommon. While lots of control lines are defined, in practice we normally only see either 3 or 5 wires used (2 for transmit/receive, one ground, and sometimes 2 for send/receive control or "handshaking" between the devices).

RS422 was developed in the 1980's to give greater distance and speed to serial data links. By using a differential voltage scheme (RS232 measures all voltages relative to ground, RS422 measures them relative to one another), immunity to electrical noise is very high and much lower voltage levels may be used, too. As a result, this standard is rated for point-to-point data communication at speeds up to 115,000 baud and distances to 4,000 feet. Data flow control is most often accomplished with software rather than hardware, so a typical cabling uses only two twisted pairs of wires. At shorter distances (50 feet), speeds up to 10,000,000 baud are possible. This standard doesn't define a connector, however, so we're at the mercy of the manufacturers when hooking it up!

RS485 was also developed in the 1980's as an extension to the RS422 standard. It has identical ratings, and a bonus! RS485 line drivers have the ability to disconnect themselves from the circuit when they aren't in use, so it is possible to connect up to 32 devices to a single communication link. One device must be a "master," which controls all communication. The other 31 possible devices can receive all data but transmit only when the master allows them to. In this fashion, a small network is created, sort of like an old telephone party line. Some automation companies use RS485 links to connect multiple controllers for sharing of data such as time schedules and other global variables. Each device has an address, and the master must poll the various addresses in sequence to make sure only one device transmits at a time.

3.19 Summary

In summary, RS232 is a pretty fast port for two-way communication. RS422 is a really fast one, and RS485 is a really fast one, which can handle many devices on one line. Remember, too, that they're standards and not laws. Manufacturers will often add features, which surpass the limits we've described. Unlike the warden in Cool Hand Luke, our industry has no problem with "failure to communicate".

CHAPTER 4

Xcite OEM RF MODULES

4.1 Introduction

The XCite OEM RF Module is a drop-in wireless solution that can add RF connectivity to any data system. It transfers a standard asynchronous serial data stream and features the following:

- Continuous data stream of up to 38400 bps (factory-set, RF baud rate)
- Serial Interfacing from 1200 to 57600 bps
- Software selectable between Hopping (FHSS) and Single Frequency Channel Modes
- Approved by the FCC under Part 15 of the FCC Rules and Regulations
- Variable input supply voltage (2.85 – 5.50 VDC)

4.2 Features

Long Range

- Indoor/Urban Range: **Up to 300'** (90 m)
- Outdoor/RF Line-of-sight Range: **Up to 1000'** (300 m) w/ 2.1 dB dipole antenna
- Receiver Sensitivity: **-108 dBm** (9600 Baud), **-104 dBm** (38400 Baud)

Low Power

- Transmit Power Output: **4 mW** [50 mW effective considering excellent receiver sensitivity]
- **55 mA** transmit / **45 mA** receive current consumption
- Power-down current as low as **20 µA**

Advanced Networking & Security (True Peer-to-Peer (no "master" required), Point-to-Point, Point-to-Multipoint, Multidrop)

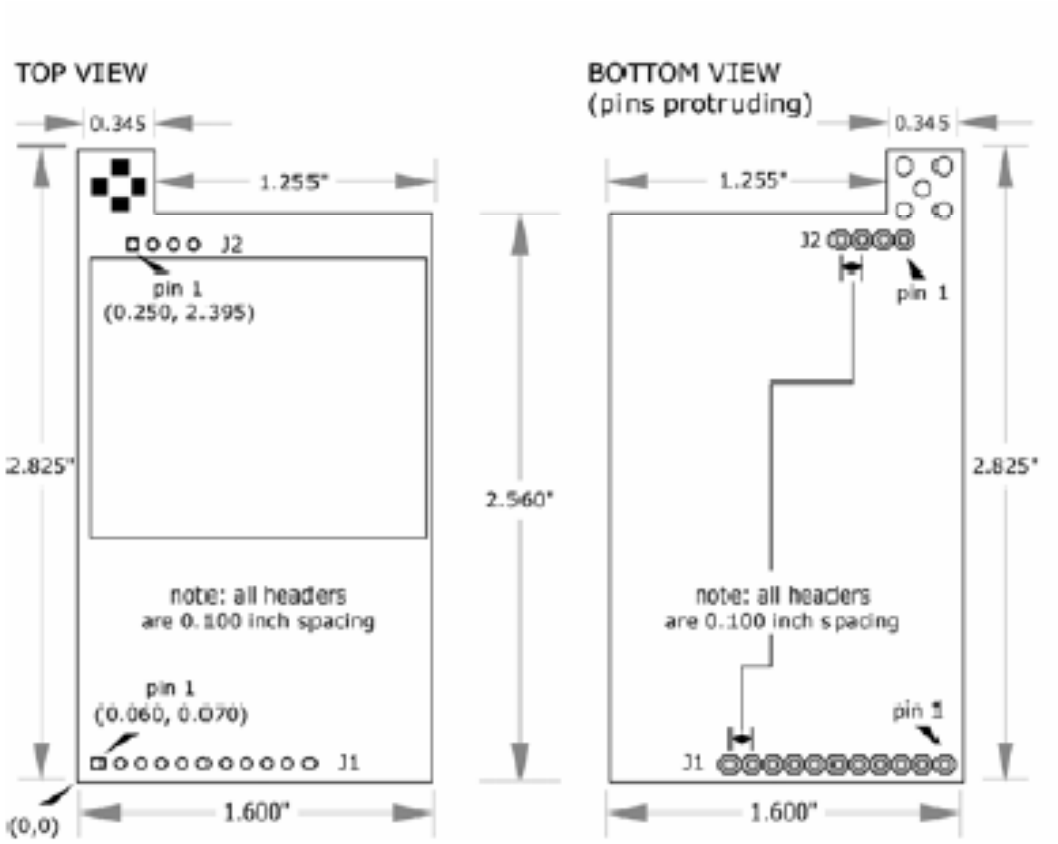
4.3 Specifications

Table on next page;

Specification	9XCite (900 MHz) OEM RF Module	
Performance		
Indoor/Urban Range	up to 300' (90 m)	
Outdoor LOS Range	up to 1000' (300 m) w/ 2.1 dB dipole antenna	
Transmit Power Output	4 mW (6 dBm)	
Interface Data Rate	Software selectable 1200-57600 bps	
Throughput Data Rate	**9600 bps**	**38400 bps**
RF Data Rate (Baud)	10000 bps	41666 bps
Receiver Sensitivity	-108 dBm	-104 dBm
Interfacing, Networking & Security		
Frequency	902-928 MHz	
Spread Spectrum Type	Frequency Hopping, Direct FM modulator	
Single Channel (Optional)	Fixed or User Selectable (up to 12 channels)	
Network Topology	Peer-to-Peer, Point-to-Multipoint, Point-to-Point, Multi-Drop Transparent	
Channel Capacity	Hopping Mode (7 hop sequences share 25 frequencies) or Single Channel Mode (25 single frequency channels)	
Serial Data Interface	CMOS	
Power Requirements		
Supply Voltage	2.85 to 5.50 VDC	
Transmit Current (@ 2.85V)	55 mA	
Receive Current (@ 2.85V)	45 mA	
Receive Current (@ 5V)	55 mA	
Power Down Current	20 µA	
Physical Properties		
Module Board Size	1.6" x 2.825" x 0.35" (4.06 cm x 7.17 cm x 0.89 cm)	
Weight	0.8 oz. (24 g)	
Connector	11-Pin & 4-Pin 0.1" spaced Male Berg-type headers	
Operating Temperature	Commercial (0 to 70° C) or Industrial (-40 to 85° C)	
Antennas		
Type	¼ Wave Monopole, 3" (7.62 cm) integrated wire, 1.9 dBi	
Connector (Optional)	Reverse-polarity SMA (RPSMA)	
Impedance	50 ohms unbalanced	
Certifications (For complete list, visit www.MaxStream.net or call 1-801-765-9885)		
FCC Part 15.247	OUR-9XCITE	
Industry Canada (IC)	4214A-9XCITE	

Table 4.1 RF Modem Specifications

4.4 Mechanical Drawing



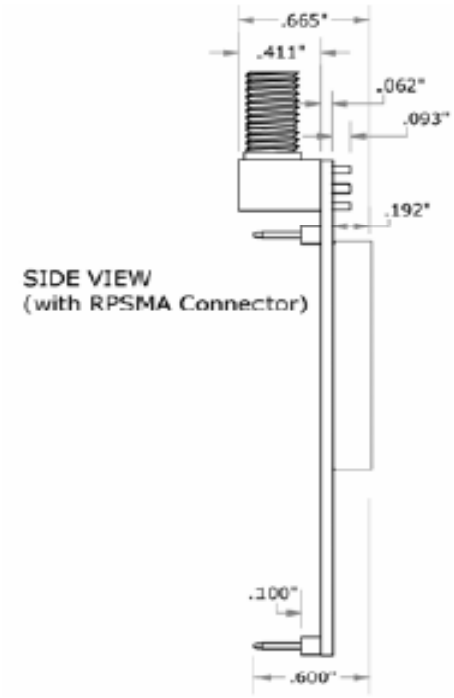


Fig 4.1 Mechanical diagram

4.5 Block Diagram

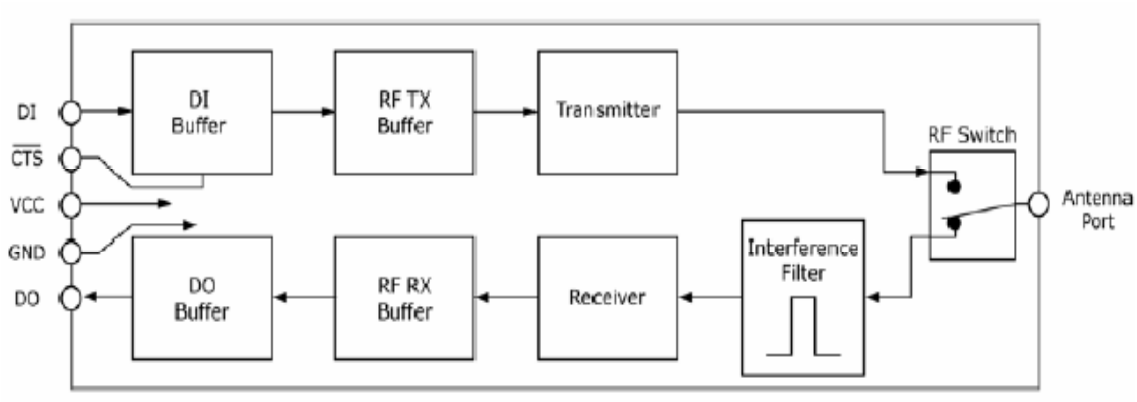


Fig 4.2 Block diagram

MaxStream's proprietary XII™ (Interference Immunity Technology) is contained in the 'Interference Filter' of the block diagram. The filter blocks interference from pagers and cellular.

4.6 Pin Signals

Xcite module pin signals and their functions:

Module Pin	Signal Name	I/O	When Active	Description
1	DO2	O*	low	Clear-to-Send ($\overline{CTS}$) Flow Control
2	DI3	I*	high	Can be used to enter Sleep Modes that consume less power
3	DO (Data Out)	O*	n/a	Serial Data leaving the XCite Module (to the host)
4	DI (Data In)	I	n/a	Serial data entering the XCite Module (from the host)
5	DI2	I**	low	Request-to-Send ($\overline{RTS}$)
6	$\overline{RESET}$	I*	low	Reset Module Parameters
7	DO3	O	high	Receive (RX) LED
8	$\overline{TX}$ / PWR	O	low	$\overline{TX}$ - Asserted during transmission
			high	PWR – Indicates power is on
9	$\overline{CONFIG}$	I***	low	Backup method for entering AT Command Mode. Primary method is with "+++" [See CC Parameter]
10	VCC	I	-	2.85 – 5.50 VDC Regulated
11	GND	-	-	Ground

Table 4.2 Pin signals

* Pin utilizes 10K Ω Pull-Up resistor (already installed in the module)

** Pin utilizes 10K Ω Pull-Down resistor (already installed in the module)

*** Pin utilizes 100K Ω Pull-Up resistor (already installed in the module)

Note: When integrating the XCite Module onto a Host PC Board, all lines that are not used should be left disconnected (floating).

Module Pin	Signal Name
1	reserved
2	GND
3	GND
4	GND

Table 4.3 Pin Signals

The interface signals are available through the 11-pin header. All pins operate on VCC CMOS levels. Five signals commonly used in applications are:

- DI (pin 4 – Data In)
- DO (pin 3 – Data Out)
- VCC (pin 10 – Power)
- GND (pin 11 – Ground)
- DO2-CTS (pin 1 – Clear-to-Send)

The remaining six pins may be used for additional functionality.

4.7 Application Circuit

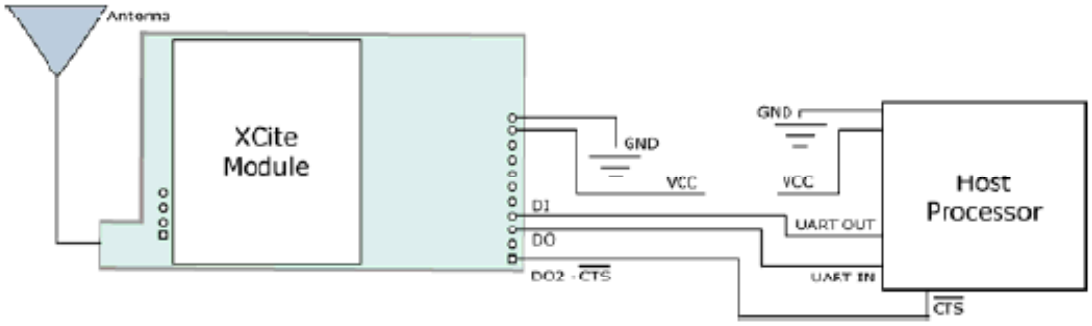


Fig 4.3 Application circuit

4.8 PIN SIGNALS

4.8.1 I/O Pin Signals

Pin 4: DI (Data In)

<Input> Data enters the XCite Module through the DI Pin as an asynchronous serial signal. The signal should idle high when no data is being transmitted.

Each data packet consists of a start bit (low), 8 data bits (least significant bit first) and a stop bit (high). The diagram of a UART data packet [Figure 10] illustrates the serial bit pattern of data shifting into the module. The start and stop bits from the UART signal are not transmitted, but are regenerated on the receiving module.

- **DI Buffer**

Once serial data has entered the XCite Module through the DI Pin, the data is stored in the DI Buffer until it can be transmitted.

Once the first byte of data enters the DI Buffer, the module begins to initialize the RF channel unless RF data is already being received. In the case where the module is receiving RF data, serial data is stored in the DI Buffer. When the DI Buffer is 17 bytes away from being full, the XCite Module de-asserts (high) CTS to signal to the host device to stop sending data. CTS re-asserts once the DI Buffer has at least 35 bytes available.

In addition to CTS hardware flow control, XON/XOFF software flow control can also be implemented [See the FL (Software Flow Control) Command]. In this case, the XCite Module sends XON and XOFF signals in addition to asserting/de-asserting CTS.

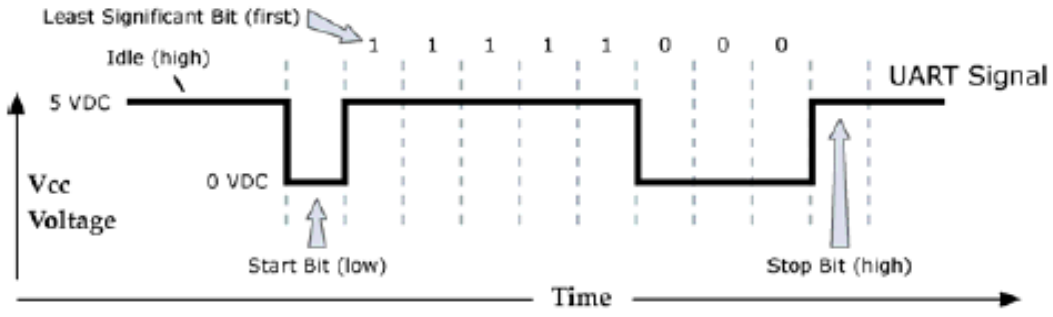


Fig 4.4 UART Signal

Data is packetized for transmission. The packet structure is as follows:

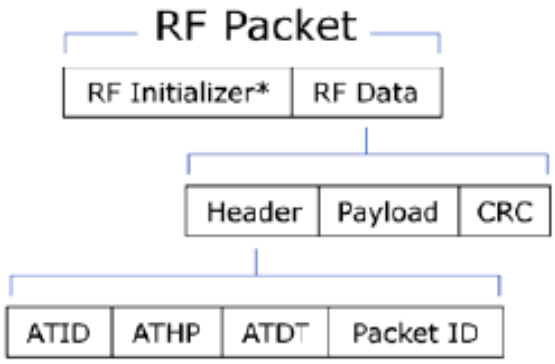


Fig 4.5 RF Packet

Pin 3: DO (Data Out)

<Output> Data from RF (over-the-air) transmission is received through the DO Pin. Received data is checked for errors and addressing, then sent to the DO Buffer before being sent to the host device. This pin utilizes a 10K Ω Pull-Up resistor that is already installed in the module.

DO Buffer

Once incoming RF data is received into the DO Buffer, data is sent out the serial port to a host device. If RTS is enabled for flow control, data will not be sent out the DO Buffer as long as the RTS pin is de-asserted (high). In such a scenario, data could be lost if RTS is de-asserted long enough to allow the DO Buffer to become full. Pins 1 & 2 can also be used for flow control.

4.8.2 Flow Control Pin Signals

- **Pin 1: DO2-CTS (Clear-to-Send)**

<Output> The CTS pin (Clear-to-Send) informs the host device whether or not serial data can be sent to the XCite Module from the host device. When Pin 1 is asserted (low), serial data is permitted to be sent to the XCite Module. In RS-232 applications, Pin 1 is directly associated with the DI Buffer. In RS-485/422 applications, the Pin 1 signal controls the transmit driver (TX enable) on the RS-485 and RS-422 logic converters (on the MaxStream Interface Board or equivalent). Pin 3 utilizes a 10KΩ Pull-Up resistor that is already installed in the module. In some applications, Pin 1 may not need to be observed.

CTS Hardware Flow Control

If the DI Buffer reaches its capacity, either the CTS line or XON / XOFF flow control must be observed to prevent loss of data between the host device and the XCite Module. There are two cases in which the DI Buffer may become full:

- If the serial interface rate is set higher than the default baud rate for the module, the module will receive serial data faster than it is transmitted.
- If the XCite Module is receiving a continuous stream of data or if it is monitoring data on a network, any serial data that arrives on the DI pin is placed in the DI Buffer. This data will be transmitted when the module no longer detects RF data in the network.

XON Software Flow Control

XON/XOFF software flow control can be used (on Pin 3) instead of CTS hardware flow control. [See FL (Software Flow Control) Command for more information]

- **Pin 5: DI2- RTS (Request-to-Send)**

<Input> The Pin 5 (RTS) signal can be configured to enable RTS flow control recognition. Use RT Command to adjust the parameters that control Pin 5 flow control. By default, RTS flow control is not observed. This pin utilizes a 10KΩ Pull-Down resistor already installed in the module.

RTS Hardware Flow Control

If RTS flow control is enabled, no data is sent out the DO pin when RTS is de-asserted (high). If RTS flow control is implemented on the host device, RT Parameter must be set on the XCite Module in order to recognize the RTS signal as a flow control line.

If RTS is asserted (low), all received RF data is placed in the DO Buffer until the line is de-asserted. Once the DO Buffer reaches capacity, any additional received RF data is lost.

XOFF Software Flow Control

XON/XOFF software flow control can be used (on Pin 4) to simulate RTS hardware flow control. [See FL (Software Flow Control) Command for more information]

4.8.3 Remaining Pin Signals

- **Pin 2: DI3-Sleep/Power-Down**

<Input> Pin 2 can be used to transition the XCite Module into a low power-consuming Sleep Mode. If SM = 1 [SM (Sleep Mode) Command], allowing Pin 2 to float high causes the module to enter into a state of minimal power-consumption (until awakened by driving Pin 2 low). [Go to the "Sleep Modes" section for more information.] This pin utilizes a 10K Ω Pull-Up resistor already installed in the module.

- **Pin 6: RESET**

<Input> Pin 6 is almost always high and only low when the radio is reset. Since the OEM module has an onboard reset monitor, this pin can be left disconnected. Pin 6 utilizes a 10KΩ Pull-Up resistor already installed in the module.

- **Pin 7: DO3-RX LED**

<Output> Pin 7 is normally driven low, but is driven high briefly by the radio to indicate RF data reception. This pin can be tied through a resistor to an LED for visual indication.

- **Pin 8: TX/ PWR**

<Output> Pin 8 is normally driven high and can be tied through a resistor to an LED to indicate the following:

- The module has power

- The module is not is sleep mode
- Pin 8 pulses on/off when data is transmitted over-the-air.

➢ **Pin 9: CONFIG**

<Input> When Command Mode cannot be entered using normal procedure [See "Command Mode" section], the CONFIG pin is used to manually enter the module into Command Mode. If Pin 9 is asserted during reset or power-up, the module immediately enters into Command Mode at the module's default baud rate. After the pin is asserted, the serial port baud rate is temporarily set to match the default baud rate of the XCite Module in use. Upon entering into Command Mode, all configured parameters (including baud rate) remain in their saved state unless modified as is described in the "Module Configurations" chapter of this manual. This pin utilizes a 100K Ω Pull-Up resistor already installed in the module.

IMPORTANT: The CONFIG pin is intended as a secondary method for entering Command Mode. The primary method is with a command break sequence. MaxStream reserves the right to change the functionality of the CONFIG pin and recommends using the command break sequence [See "Command Mode" section] for entering Command Mode.

➢ **Pin 10: VCC (power)**

<Input> Pin 10 accepts regulated 5V signals.

➢ **Pin 11: GND (Ground)**

Pin 11 is used for grounding.

4.9 Electrical Characteristics

- **Timing Specifications**

The diagram below shows 9XCite Modules providing an RF Link between hosts.

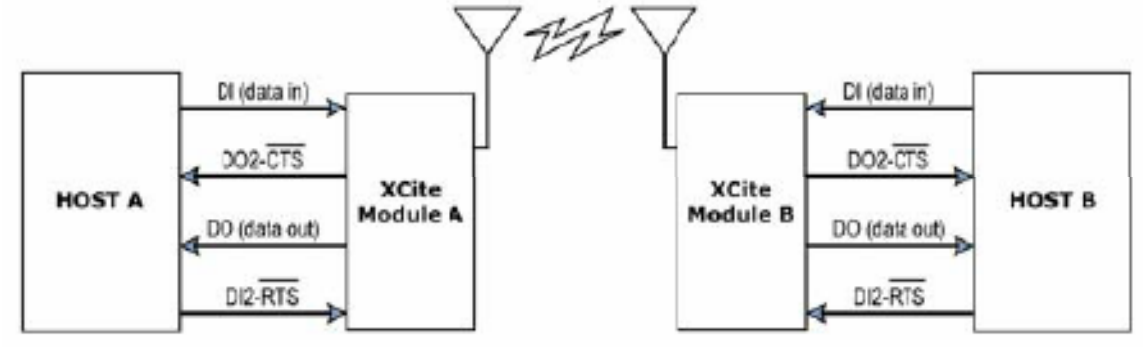


Fig 4.6 RF Link

The RF data flow sequence is initiated when the first byte of data is received by the transmitting XCite Module. As long as XCite Module A is not already receiving RF data, the serial data goes into the DI Buffer, is packetized, and then transmitted to XCite Module B.

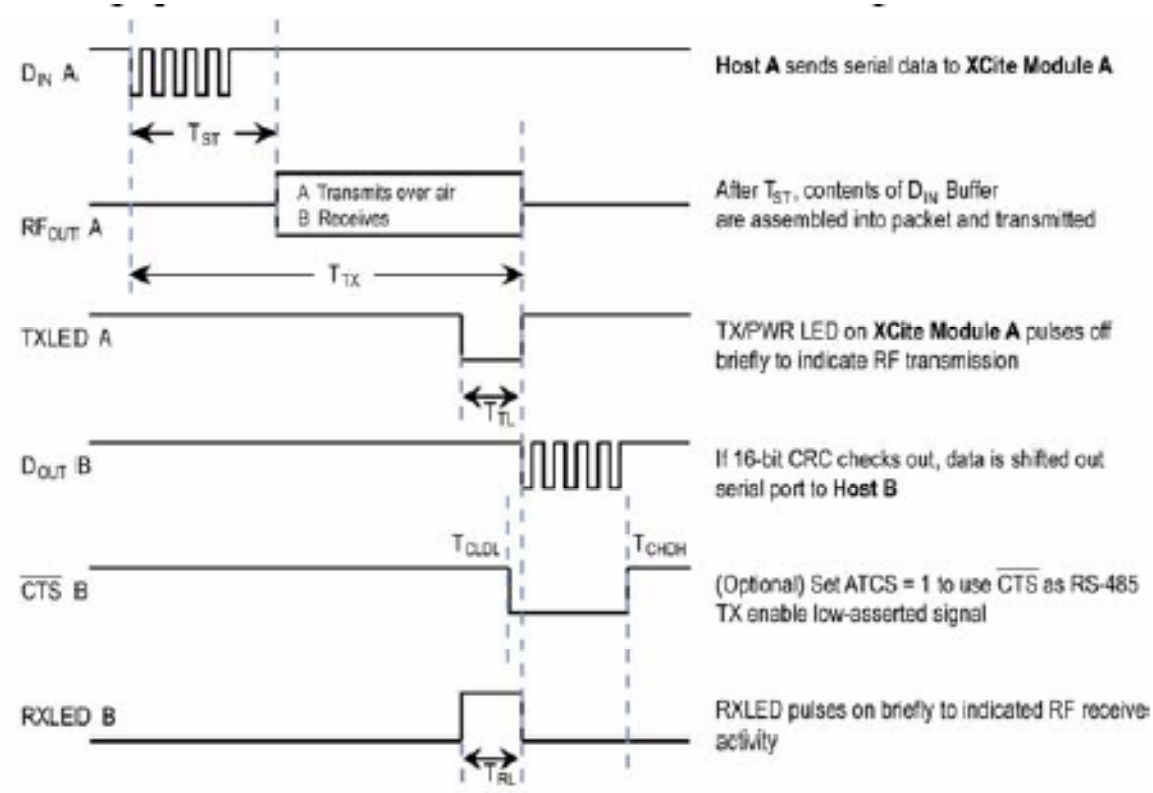


Fig 4.7 RF Data flow sequence

Symbol	Parameter	Condition	Min	Typical	Max	Units
V_{IL}	Input Low Voltage	All input signals	-0.5		0.3 * Vcc	V
V_{IH}	Input High Voltage	All except $\overline{\text{RESET}}$ pin	0.6 * Vcc		Vcc + 0.5	V
V_{IH2}	Input High Voltage	$\overline{\text{RESET}}$ pin *	0.9 * Vcc		Vcc + 0.5	V
V_{OL}	Output Low Voltage	I_{OL} = 20 mA, Vcc = 5V			0.7 0.5	V
V_{OH}	Output High Voltage	I_{OH} = -20 mA, Vcc = 5V	4.0 2.0			V
I_{IL}	Input Leakage Current I/O Pin	Vcc = 5.5V, pin low (absolute value)			3	µA
I_{IH}	Input Leakage Current I/O Pin	Vcc = 5.5V, pin high (absolute value)			3	µA
I_{IL2}		$\overline{\text{CTS}}$, DO, $\overline{\text{RESET}}$		(Vcc – VI) / 10 **		mA
I_{IL3}		$\overline{\text{CONFIG}}$		(Vcc – VI) / 47 **		mA
I_{IH2}		$\overline{\text{RTS}}$		(Vcc – VI) / 10 **		mA

Table 4.1 Symbols and parameters

* Reset pulse must last at least 250 nanoseconds

** VI = the input voltage on the pin

4.10 RF Module Operation

4.10.1 Serial Communications

The XCite OEM RF Module interfaces to a host device through a CMOS-level asynchronous serial port. Through its serial port, the XCite Module can communicate directly with any UART-interfaced or RS-232/485/422 device. [Timing specifications illustrated in Figure]

UART-Interfaced Data Flow

Devices that have a UART interface can connect directly through the pins of the XCite OEM RF Module as is shown in the figure below.

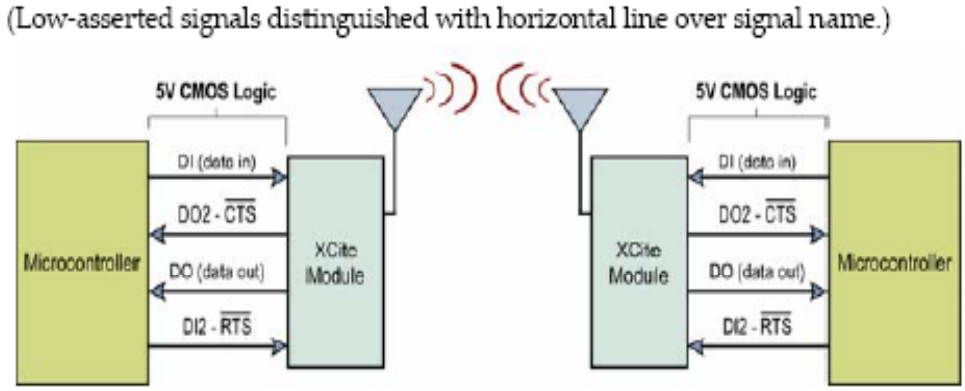


Fig 4.8 UART interfaced structure

RS-232 and RS-485/422 Data Flow

The XCite Module can enable a host device to communicate wirelessly. To transmit, the host device simply sends serial data to the XCite Module pins. The XCite Module then converts the data into FCC-approved RF data. Once transmitted, the RF data can be detected by receiving XCite Modules, checked for integrity and then sent to a receiving device.

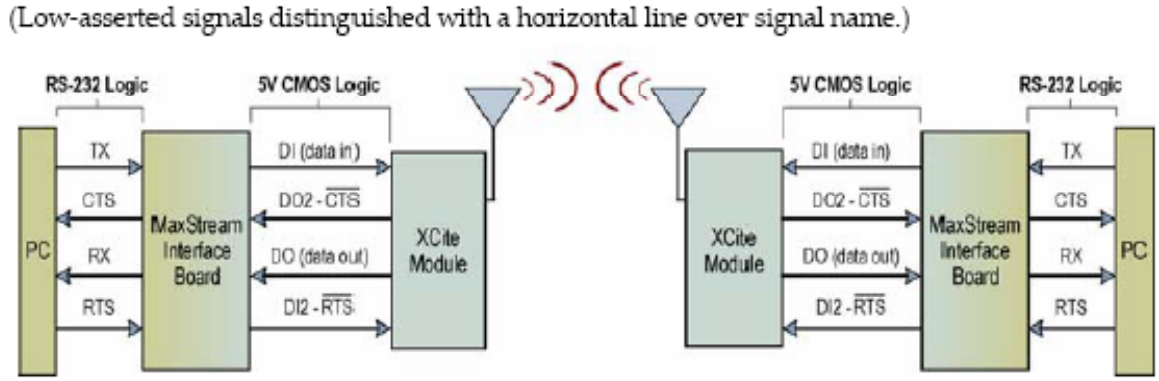


Fig 4.9 RS-232 Data flow

Serial Data

Data enters the MaxStream OEM RF Module through the DI Pin (pin 4) as an asynchronous serial signal. The signal should idle high when no data is being transmitted.

The UART performs tasks (such as timing and parity checking) needed for communication. Serial communication consists of two UARTs which must be configured with compatible parameters (Baud rate, parity, start bits, stop bits, data bits) to have successful communication. Each data packet consists of a start bit (low), 8 data bits (least significant bit first) and a stop bit (high). The following figure illustrates the serial bit pattern of data passing through the module.

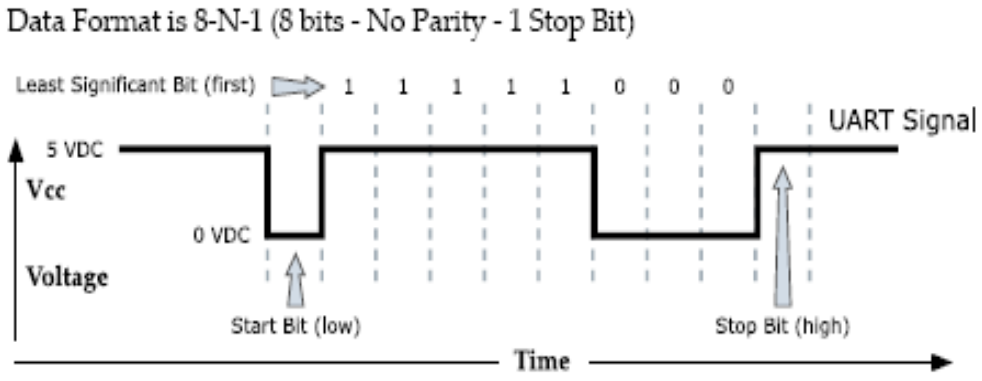


Fig 4.10 Serial data

In the example above, the MaxStream Module transfer 8 bits over-the-air [Selectable using BI (Number of Bits) Parameter]. Start and stop bits of the UART signal are not transmitted over-the-air, but are regenerated by the receiving module.

The MaxStream OEM RF Modules support the following data formats [See NB (Parity) Parameter]:

- 8-none-1, 7-any-1
- 8-even-1

- 8-odd-1
- 8-mark-1, 8-none-2
- 8-space-1

DI (Data In) Buffer

Once serial data has entered the 9XCite Module through the Data In (DI) Pin, the data is stored in the DI Buffer until it can be transmitted.

Once the first byte of data enters the DI Buffer, the module begins to initialize the RF channel (unless RF data is being received). In the case where the module is receiving RF data, the serial (UART) data is stored in the DI Buffer. When the DI Buffer has only 17 bytes of memory left, the XCite Module de-asserts (high) CTS to signal to the host device to stop sending data. CTS re-asserts once the DI Buffer has at least 35 bytes available in memory. Once in the DI Buffer, the data is packetized for transmission. [Packet structures are broken down below.]

4.10.2 Modes of Operation

XCite Radio Modems operate in five modes. The modules operate in one mode at a time.

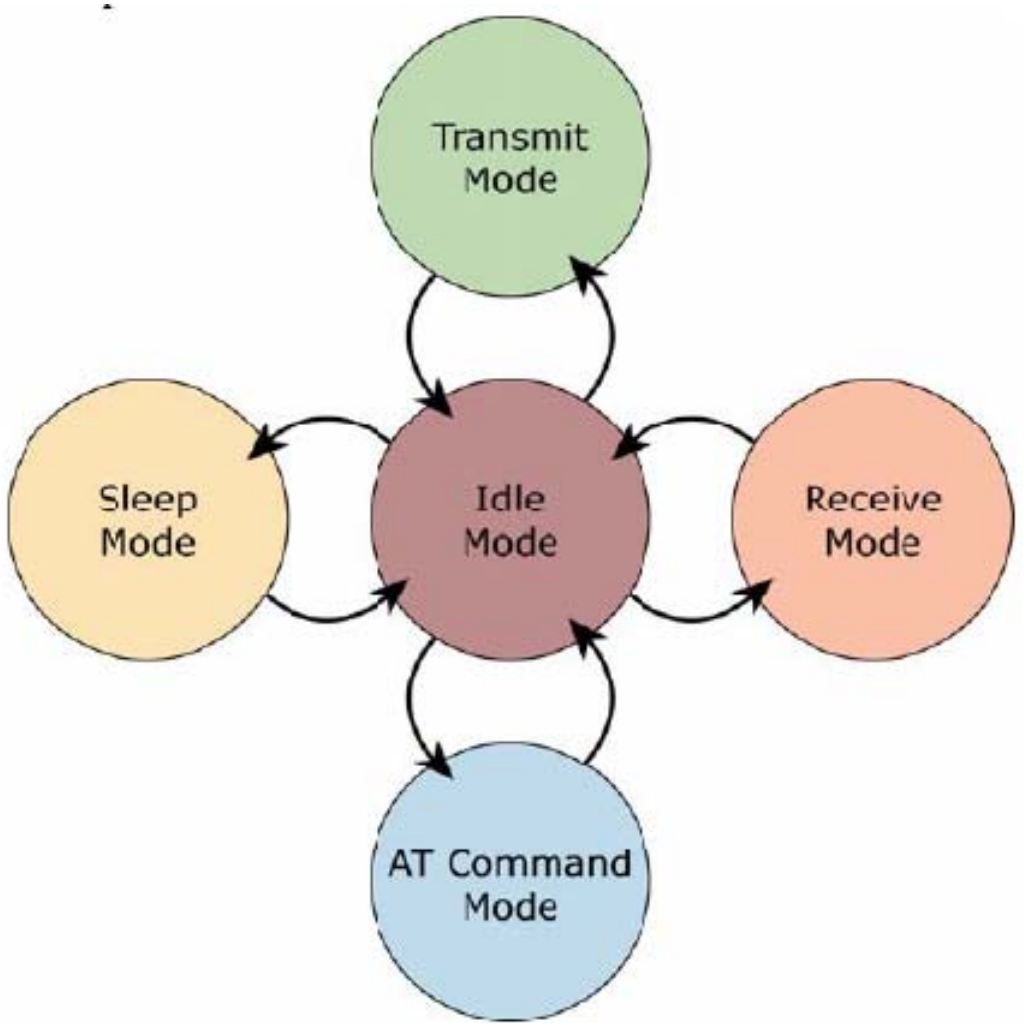


Fig 4.11 Modes of operation

1. Idle Mode

XCite Modules operate in Idle Mode when data is not being transmitted nor received. While in Idle Mode, modules use the same amount of power as they do in RX (Receive) mode. Modules will transition into other modes under any of the following conditions:

1. Serial data is received in the DI Buffer (Module then transitions into Transmit Mode)
2. Valid data is received by the antenna (Module then transitions into Receive Mode)

3. AT Command Mode Sequence is issued (Module then transitions into AT Command Mode)

4. Sleep Mode condition is met (Module then transitions into Sleep Mode)

Modules automatically transition back to Idle Mode once finished responding to these conditions.

2. Transmit Mode

When the first byte of serial data comes through the DI Pin and arrives in the DI Buffer, the module transitions into Transmit Mode. Once in Transmit Mode, the module initializes a communications channel. During channel initialization, incoming serial data accumulates in the DI buffer. After the channel is initialized, data in the DI buffer is grouped into packets (up to 64 bytes in each packet) and is transmitted. The module continues to transmit data packets until the DI buffer is empty. Once transmission is finished, the module returns to Idle Mode. This progression is shown below:

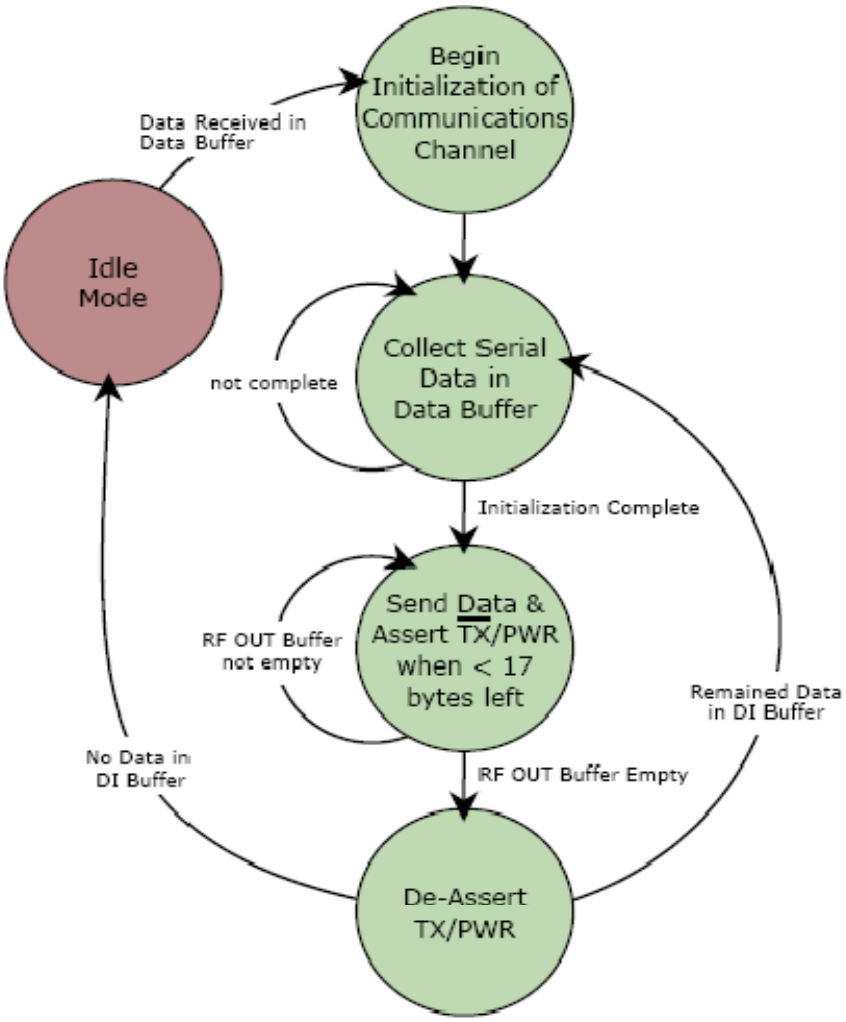


Fig 4.12 Transmit Mode

Cyclic Redundancy Check (CRC)

To verify data integrity and provide built-in error checking, a 16-bit cyclic redundancy check (CRC) is computed for the transmitted data and attached to the end of each data packet before transmission. On the receiving end, the receiver computes the CRC on all incoming data. Received data that has an invalid CRC is discarded.

Transmission Latency

Transmission latency depends on the number of bytes contained in a packet and the baud rate of the module. To reduce latency in the XCite Module, load in a single channel version using the X-CTU Software. Operating in Single Frequency Channel Mode greatly reduces latency.

3. Receive Mode

If a module detects RF transmitted data while operating in Idle Mode, it transitions into Receive Mode to start receiving packets. Once a packet is received, it goes through the receiving-end of a CRC (cyclic redundancy check) to ensure that the data was transmitted without error. If the CRC data bits on the incoming packet are invalid, the packet is discarded. If the CRC is valid, the packet is placed the DO Buffer. This process is shown in the figure below:

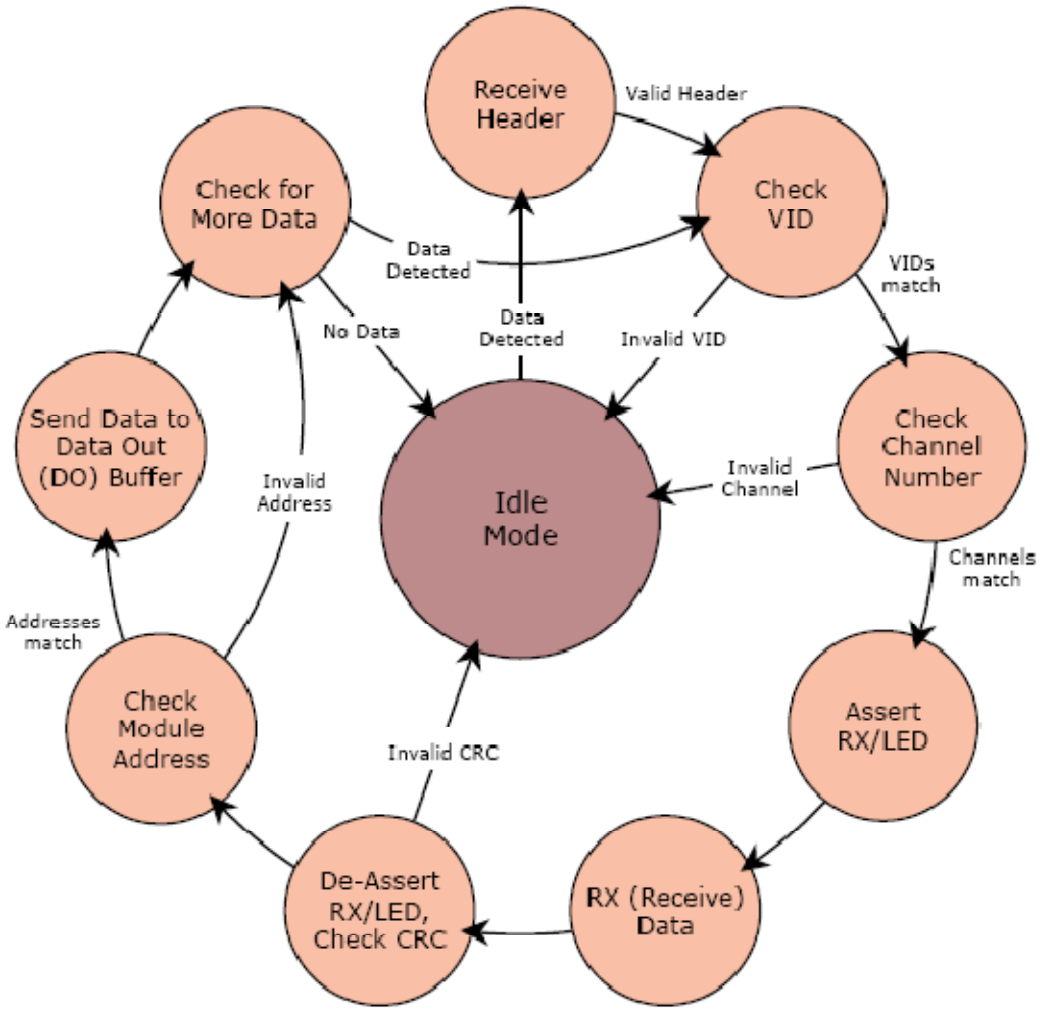


Fig 4.13 Receive Mode

The module returns to Idle Mode after valid data is no longer detected or once an error is detected in the received data. If serial data-to-transmit is stored in the DI buffer while the module is giving precedence to Receive Mode, the data will be transmitted after the module finishes receiving data and returns to Idle Mode.

4. Sleep Modes

Sleep Modes enable the XCite Radio Modem to go into states of low power-consumption when not in use. Any of three Sleep Modes configurations can be used:

1. Host Controlled
2. Wake on RF activity
3. Wake on Serial Port activity

To enter Sleep Mode, one of the following must occur (In addition to SM (Sleep Mode) Command having a non-zero value):

- The radio modem must be idle (no data transmission or reception) for a user-defined period of time [See ST (Time before Sleep) Command]
- The Sleep Pin (Pin 2) is de-asserted

Once in Sleep Mode, the radio modem does not transmit or receive data until it first returns to Idle Mode. The return into Idle Mode is triggered by the de-assertion of Pin 2 or the arrival of a serial byte through Pin 4 (Data In). Sleep Mode is enabled and disabled using SM Command.

The following table lists MaxStream's Sleep Mode configurations and the requirements needed to transition into and out of Sleep Mode:

Sleep Mode Setting	Transition into Sleep Mode	Transition out of Sleep Mode	Related Commands	Power Consumption
Pin Sleep (SM=1)	Microcontroller can shut down and Wake-up modules. Assert (high) Sleep Pin (Pin 2). Note: Module will complete a transmission or reception before activating Pin Sleep.	De-Assert (low) Sleep pin (Pin 2)	SM	20 µA
Serial Port Sleep (SM=2)	Automatic transition into Sleep Mode after user-defined period of inactivity (no transmitting or receiving). Period of inactivity set using ST Command.	When serial byte is received on the DI pin (Pin 4)	SM, ST	1 mA
Cyclic Sleep (SM=3-8)	Transitions into and out of Sleep Mode in cycles (user-selectable wake-up interval of time (½ second to 16 seconds) set by SM Command). The Cyclic Sleep interval time must be shorter than "Wake-up Initializer Timer" (set by LH Command). (Can be forced into Idle Mode using Sleep Pin if PW (Pin Wake-up) Command is issued.)		HT, LH, PW, SM, ST	typically 76 µA (when sleeping)

Table 4.13 Sleep mode configurations

Pin Sleep (SM = 1)

<Lowest Power Configuration> In order to achieve this low-power state, Pin 2 must be asserted (high). The module remains in Pin Sleep until the Sleep pin is de-asserted. The module will complete a transmission or reception before activating Pin Sleep.

After enabling Pin Sleep (SM (Sleep Mode) Parameter = 1), Pin 2 controls whether the XCite Module is active or in Sleep Mode. When Pin 2 is asserted (high), the module transitions to Sleep Mode and remains in its lowest power-consuming state until the Sleep pin is de-asserted. The XCite Module requires 40ms to transition from Sleep Mode to Idle Mode. Pin 2 is only active if the module is setup to operate in this mode; otherwise the pin is ignored. Once in Pin Sleep Mode, Pin 1 (CTS)

is de-asserted (high), indicating that data should not be sent to the module. Pin 8 (PWR) is also de-asserted (low) when the module is in Pin Sleep Mode.

Serial Port Sleep (SM = 2)

Serial Port Sleep is a Sleep Mode setting in which the module runs in a low power state until data is detected on the DI pin.

When Serial Port Sleep is enabled, the module goes into Sleep Mode after a user-defined period of inactivity (no transmitting or receiving of data). This period of time is determined by ST (Time before Sleep) Command. The module returns to Idle Mode once a character is received through the DI pin.

Cyclic Sleep (SM = 3-8)

Cyclic Sleep is the Sleep Mode setting in which the XCite Module enters into a low power state and awakens periodically to determine if any transmissions are being sent.

When Cyclic Sleep settings are enabled, the XCite Module goes into Sleep Mode after a user-defined period of inactivity (no transmission or reception on the RF channel). The user-defined period is determined by ST Parameter. [See ST (Time before Sleep) Parameter]

While the module is in a low-power state, Pin 1 (CTS) is de-asserted (high) to indicate that data should not be sent to the module during this time. When the module awakens to listen for data, Pin 1 (CTS) is asserted and any data received on the DI Pin is transmitted. Pin 8 (PWR) is also de-asserted (low) when the module is in Cyclic Sleep Mode. Pins 1 and 8 are asserted each time the module cycles into Idle Mode to listen for valid data packets and de-asserts when the module returns to Sleep Mode.

The module remains in Sleep Mode for a user-defined period of time ranging from 0.5 seconds to 16 seconds (SM Parameters 3 through 8). After this interval of time, the module returns to Idle Mode and listens for a valid data packet for 100 ms. If the module does not detect valid data (on any frequency), the module returns to Sleep Mode. If valid data is detected, the module transitions into Receive Mode and receives the incoming packets. The module then returns to Sleep Mode after a Period of inactivity that is determined by ST "Time before Sleep" Parameter.

The module can also be configured to Wake-up from cyclic sleep when the SLEEP Pin (Pin 2) is de-asserted (low). To configure a module to operate in this manner, PW (Pin Wake-up) Command must be issued. Once the Pin 2 (Sleep Pin) is de-asserted, the module is forced into Idle Mode and can begin transmitting or receiving data. It remains active until no data is detected for the period of time specified by the ST parameter, at which point it resumes its low-power cyclic state.

Cyclic Scanning

Each RF transmission consists of a wake-up initializer and payload data. The wake-up initializer contains initialization information and all receiving modules must Wake-up during the wake-up initializer portion of data transmission in order to synchronize with the transmitter and receive the data.

The length of the wake-up initializer exceeds the time interval of Cyclic Sleep. The receiver is guara

nteed to detect the wake-up initializer and receive the accompanying payload data.

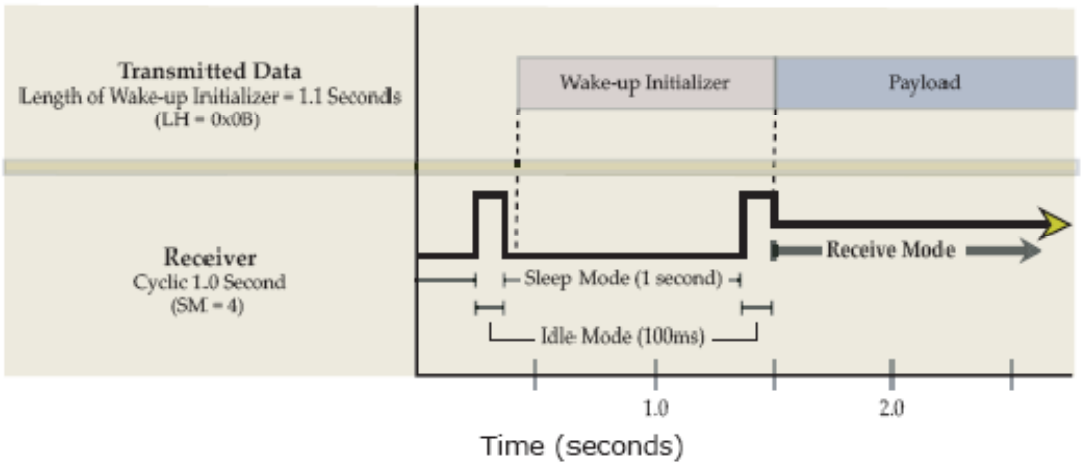


Fig 4.14 Cyclic scanning actual

The length of the wake-up initializer is shorter than the time interval of Cyclic Sleep. this is vulnerable to the receiver waking and missing the wake-up initializer (and therefore also the acco mpanying payload data).

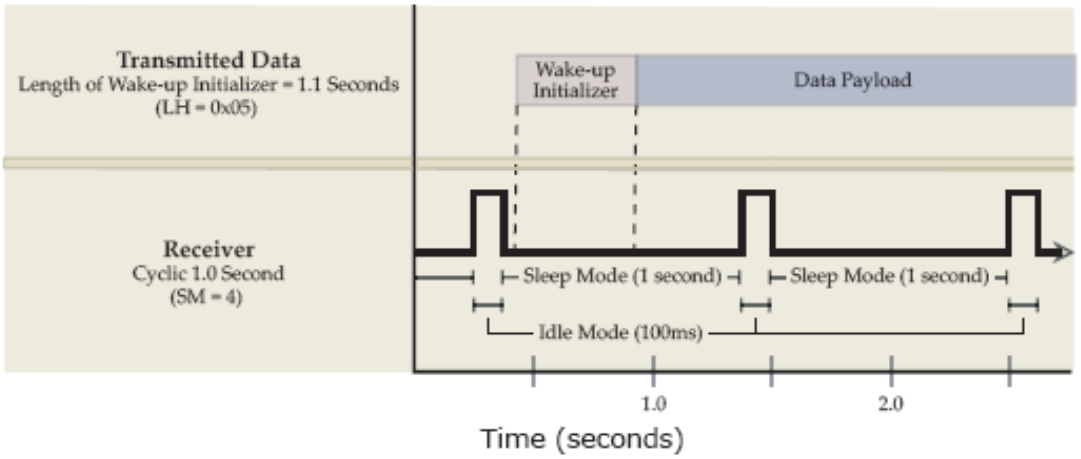


Fig 4.15 Cyclic scanning for error case

The wake-up initializer is sent with the initial transmitted packet after a user-defined period of inactivity (no serial or RF data is sent or received). This period of inactivity is adjusted using HT Command. [See HT (Time before Wake-up Initializer) Parameter] Sending a wake-up initializer

(length slightly exceeds the cyclic sleep time interval) assures that the receiver will detect the new transmission and will be able to receive the accompanying data. If the sleeping module misses the wake-up initializer due to interference and does not respond as expected, a new wake-up initializer can be sent using FH (Force Wake-up Initializer) Command.

5. Command Mode

AT Command Mode provides access to AT-Settable parameters. These parameters extend flexibility in configuring modules to fit specific design criteria such as networking modules. Not all of the parameters in the XCite Module can be adjusted using AT Commands.

Enter AT Command Mode

To issue XCite AT Commands, you must first transition out of Idle Mode and into AT Command Mode.

To enter into AT Command Mode, use any of the following means:

1. Through **X-CTU Software**: When using the X-CTU Software, any of the buttons that read or write modem parameters automatically trigger entrance into AT Command Mode. To successfully enter into AT Command Mode in this manner, the serial data baud rate [BD (Baud Rate) Parameter] of the modem must be equal to that of the PC Serial Com Port [Settable using the "PC Settings" tab of the X-CTU Software.] The X-CTU Software can be downloaded from: www.maxstream.net/support/downloads.php

2. Through **Serial Communications Software** ("X-CTU", "HyperTerminal", "Pro Comm", etc.): When using serial communications software to enter into AT Command Mode, users must send the "AT Command Mode Sequence". The default sequence is as follows:

- No characters sent for 1 second. [Time can be modified using BT (Guard Time Before) Parameter]
- Input three (3) plus characters ("+++") within one (1) second. [Character can be modified using CC (Command Sequence Character) Parameter.]
- No characters sent for one (1) second. [Time can be modified using the AT (Guard Time After) Parameter.]

"AT" & "BT" times must always be observed.

3. Assert (low) the CONFIG pin*, then power the modem off and then on again. (If using the MaxStream Interface Board, power and configuration switches are available to facilitate this process.)

Important: * Never tie the CONFIG pin to the microprocessor.

Configure and Read Module Parameters

Once in AT Command Mode, parameters can be configured and read using AT Commands. Parameters changed while in AT Command Mode must be saved to non-volatile memory using WR (Write) Command for the changes to persist in memory. If WR Command is not issued, the parameters will be reset to their previously stored value the next time the module is powered-up.

The "Modem Configuration" chapter is dedicated to explaining the methods needed to configure and read module parameters. The following sections explain the protocols and syntax required to configure and read module parameters.

Syntax. When using serial communications software, ASCII commands and parameters are not case sensitive. The optional space can be any non-alpha-numeric character and XCite Modules require parameter values be hexadecimal.

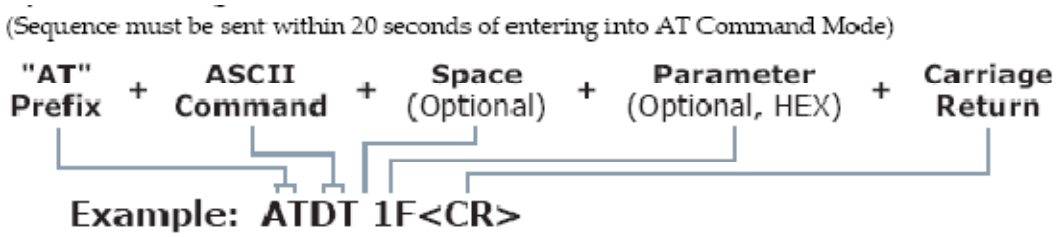


Fig 4.16 Syntax for entering Command

In example above, the "ATDT 1F" sequence would change the modem's networking address to a hexadecimal value of "1F" (decimal 31).

Queries. To query the current value of a particular parameter, send the corresponding AT command without any parameters (followed by a carriage return).

Multiple Commands. Multiple AT commands can be entered on one line with one carriage return at the end of the line. Each command must be delimited by a comma (and an optional space). The "AT" prefix is only sent before the first command and should not be included with subsequent commands in a line.

System Response. After executing a recognized AT command, the module responds with "**OK<CR>**". If an unrecognized command or a command with a bad parameter is received, the module responds with "**ERROR<CR>**." Modified AT values are reset to previous stored values upon module power-down unless the WR (Write) Command was issued to save parameters to non-volatile memory.

4.11 RF Module Configurations

The following versions of the XCite Module are currently available:

- 900 MHz, 9600 Baud (RF data rate), Hopping Channel Mode
- 900 MHz, 9600 Baud, Single Channel mode
- 900 MHz, 38400 Baud, Hopping Channel mode
- 900 MHz, 38400 Baud, Single Channel mode

XCite Modules can operate in both Single Channel and Hopping modes. Mode is selectable using the "Function Set" dropdown list of the "XCite Configuration" tab of the MaxStream-provided X-CTU Software.

The XCite Module is shipped with a unique parameter set in its memory. Parameters within the set are organized under the following categories: AT Commands & Non-AT Settable Parameters.

4.11.1 Command & Parameter Types

AT Commands

AT Commands can be changed at any time by entering AT Command Mode and sending commands to the module. [AT Commands are listed in Table 4..]

AT Commands can be modified using the any of the following means:

- X-CTU Software "Modem Configuration" tab
- X-CTU Software "Terminal" tab
- Terminal software program (such as "HyperTerminal")
- Microcontroller

Non-AT Settable Parameters (X-CTU Software configurable only)

Non-AT Settable Parameters can only be adjusted using the MaxStream-provided X-CTU Software. To modify Non-AT Settable Parameter, connect the module to the serial com port of a PC (interface board is necessary for RS-232 connection) and modify parameter values through the X-CTU Software interface. These parameters enable features that need to be set before the module is used in the field. [Non-AT Settable Parameters are listed in Table 4..]

Non-AT Settable Parameters can only be modified using the following means:

- X-CTU Software "Modem Configuration" tab

4.11.2 Configuration Software

a. X-CTU Software

X-CTU is MaxStream-provided software used to configure XCite Modules. It is the only means that can be used to set all three command parameter types [AT Commands and Non-AT Settable Parameters].

X-CTU Software is organized into the following four tabs:

- **PC Settings** tab - Setup PC serial ports to interface with an XCite Module assembly
- **Range Test** tab – Test XCite Module range
- **Terminal** tab – Configure and read XCite Module parameters using AT Commands
- **Modem Configuration** tab – Configure and read XCite Module parameters

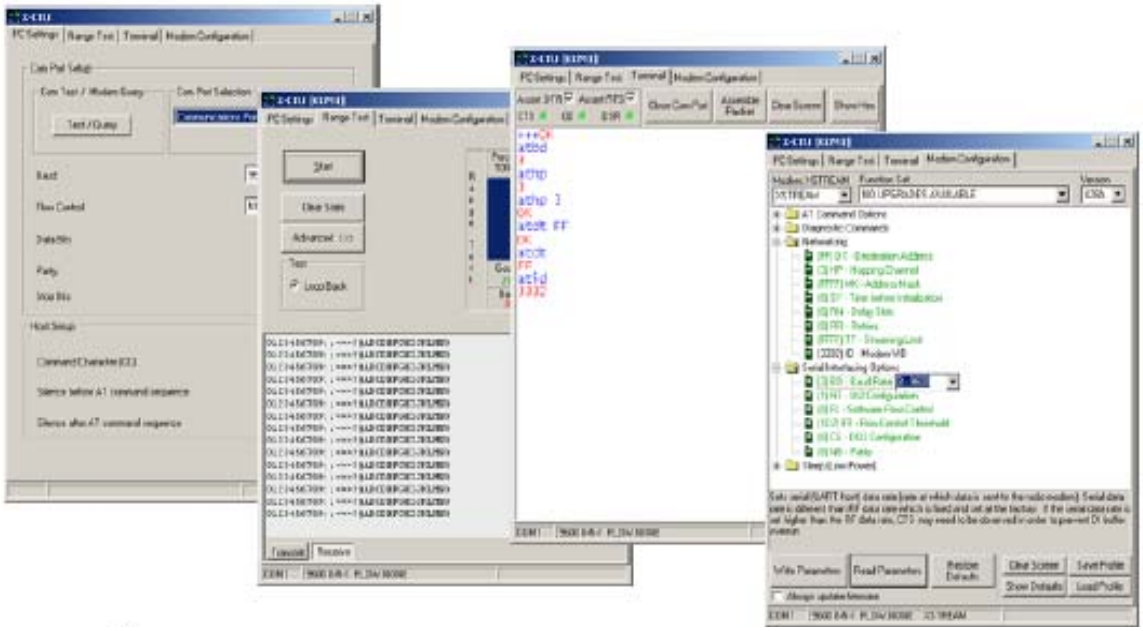


Fig 4.17 X-CTU Software windows

Install X-CTU software

Double-click the "setup_X-CTU.exe" file and follow prompts of the installation screens. This file is located in the ‘software’ folder of the MaxStream CD and also under the ‘Downloads’ section of the following web page: www.maxstream.net/support/downloads.php

Using X-CTU software

In order to use the X-CTU software, a module assembly (an XCite Module mounted to a MaxStream Interface Board) must be connected to the serial port of a PC. The baud rate of the serial port (“PC Settings” tab) must match the baud rate of the module (BD (Baud Rate) Command on the “Modem Configuration” tab).

b. Serial Communications Software

<for AT Commands Only>A terminal program has been built into the X-CTU software. Serial communications software can be used to issue AT Commands, but cannot be used to set Non-AT Settable Parameters. Use the syntax illustrated in the following example when issuing AT Commands. (This example sets the destination address of the module to “0x1F”. WR (Write) Command would also have to be used to save the new value to non-volatile memory.)

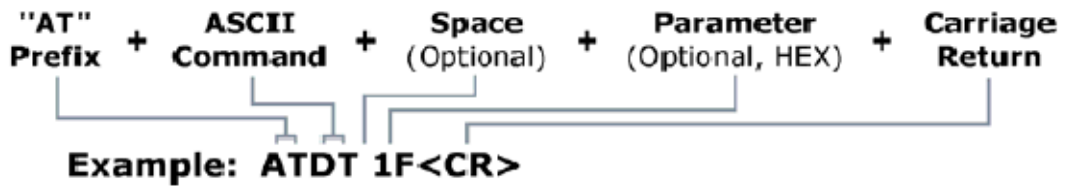


Fig 4.18 Syntax

4.11.3 Command Descriptions

Commands and parameters are listed alphabetically. Parameter types and categories are designated between “< >” symbols. For example: <AT Command: Networking>. “AT Command” is the command/parameter type and “Networking” is the command/parameter category.

- **AT (Guard Time After) Parameter**

<Non-AT Settable Parameter: AT Command Options> AT Parameter is used to set the DI pin silent time that follows the command sequence character (CC Parameter). By default, 1 half of a second (500 milliseconds) must elapse before entering another character. The AT Command Mode Sequence used to enter AT Command Mode is as follows:

- No characters sent for 1 millisecond [BT (Guard Time Before) Parameter]
- Send three plus characters “+++” [CC (Command Sequence Character) Parameter]
- No characters sent for 1 millisecond [AT (Guard Time After) Parameter]

All of the values in this sequence can be adjusted. AT Parameter is used to adjust the period of silence that follows the command sequence character.

Parameter Range: 0x02 – 0xFFFF (x 1 millisecond)

# of bytes returned: 2

Default Parameter Value: 0x1F4 (500 decimal)

Related Commands: BT (Silence before Sequence), CC (Commands Sequence Character)

- **BD (Interface Data Rate) Parameter**

<Non-AT Settable Parameter: Serial Interfacing> BD Parameter allows the user to adjust the UART baud rate and thus modify the rate at which serial data is sent to the module. Baud rates range from 1200 to 57600 baud (bps). The new baud rate does not take effect until CN (Exit AT Command Mode) Command is issued.

Note: If the serial data baud rate is set to exceed the fixed RF data baud rate of the XCite radio modem, flow control may need to be implemented as is described in the "I/O Pin Signals" section of this Manual.

Parameter Range: 0 - 6

# of bytes returned: 1

Default Parameter Value: Set to equal radio modem's fixed RF data rate (baud).

Parameter Value	Configuration
0	1200 Baud (bps)
1	2400
2	4800
3	9600
4	19200
5	38400
6	57600

Fig 4.18 BD Parameter

- **BI (Number of Bits) Parameter**

<Non-AT Settable Parameter: Serial Interfacing> BI Parameter allows the user to define the number of data bits between the start and stop bits. Setting 7 bits and Mark or Space parity (NB Parameter) will result in a setting of 7 bits and no parity.

Parameter Range: 0 - 1

# of bytes returned: 1

Default Parameter Value: 1

Parameter Value	Configuration
0	7 bits
1	8

Fig 4.19 BI Parameter

- **BT (Guard Time Before) Parameter**

<Non-AT Settable Parameter: AT Command Options> BT Parameter is used to set the DI pin silent time that precedes the command sequence character (CC Parameter). By default, 1 half of a second (500 milliseconds) must elapse before entering another character. The AT Command Mode Sequence used to enter AT Command Mode is as follows:

- No characters sent for 1 millisecond [BT (Guard Time Before) Parameter]
- Send three plus characters "+++" [CC (Command Sequence Character) Parameter]
- No characters sent for 1 millisecond [AT (Guard Time After) Parameter]

All of the values in this sequence can be adjusted. AT Command is used to adjust the period of silence that precedes the command sequence character.

Parameter Range: 0 - 0xFFFF (x 1 millisecond)

# of bytes returned: 2

Default Parameter Value: 0x1F4 (500 decimal)

Related Commands: AT (Guard Time After), CC (Commands Sequence Character)

- **CC (Command Sequence Character) Parameter**

<Non-AT Settable Parameter: AT Command Options> CC Parameter is used to adjust the command sequence character used when entering AT Command Mode.

The AT Command Mode Sequence used to enter AT Command Mode is as follows:

- No characters sent for 1 millisecond [BT (Guard Time Before) Parameter]
- Send three plus characters "+++" [CC (Command Sequence Character) Parameter]
- No characters sent for 1 millisecond [AT (Guard Time After) Parameter]

Parameter Range: 0x20 – 0x7F

# of bytes returned: 1

Default Parameter Value: 0x2B (ASCII "+" sign)

Related Parameters: AT (Guard Time After), BT (Guard Time Before)

- **CD (DO3 Configuration) Command**

<AT Command: Serial Interfacing> Used to redefine the RX LED I/O line.

AT Command: CD

Parameter Range: 0 – 5

# of bytes returned: 1

Default Parameter Value: 0

Parameter Value	Configuration
0	RX LED
1	High
2	Low

Fig 4.20 CD Command

- **CN (Exit AT Command Mode) Command**

<AT Command: AT Command Mode Options> CN Command allows users to explicitly exit AT Command Mode and return the radio modem into Idle Mode.

AT Command: CN

- **CS (DO2 Configuration) Command**

<AT Command: Serial Interfacing> CS Command is used to modify the behavior of the CTS signal such that it either provides RS-232 flow control, enables RS-485 transmission / reception or determines RS-422 transmit enable. By default, CTS provides RS-232 flow control. CS Parameter must be adjusted for the module to operate in RS-485/422 environments.

AT Command: CS

Parameter Range: 0 – 4

# of bytes returned: 1

Default Parameter Value: 0

Parameter Value	Configuration
0	Normal
1	RS-485 Enable (low)
2	high
3	RS-485 Enable (high)
4	low

Fig 4.21 CS Command

- **CT (Time before Exit AT Command Mode) Parameter**

<Non-AT Settable Parameter: AT Command Options> AT Command Mode can be exited manually using CN (Exit AT Command Mode) Command or, after a given time of inactivity, the module exits AT Command Mode on its own and return to Idle Mode. CT Command sets the amount of time before AT Command Mode is exited automatically. If no characters are received before this time elapses, the module will return to Idle Mode.

Parameter Range: 0x02 – 0xFFFF [x 100 ms]

# of bytes returned: 2

Default Parameter Value: 0xC8 (20 seconds decimal)

- **DB (Receive Signal Strength) Command**

<AT Command: Diagnostic> DB Parameter returns the receive signal strength (in decibels) of the last received packet. This Parameter is useful in determining range characteristics of the XCite Modules under various conditions.

AT Command: DB

Parameter Range: 0x25 – 0x6A [Read-only]

# of bytes returned: 1

- **DT (Destination Address) Command**

<AT Command: Networking> DT Command is used to set the address of the XCite Radio Modem. XCite Radio Modems use three network layers – the Vendor Identification Number (ATID), Channels (ATHP) and Destination Addresses (ATDT).

DT Command assigns an address to a radio modem that enables it to communicate only with radio modems that have matching addresses. This is similar to interconnecting several PCs under a common hub. All radio modems that share the same destination address can communicate freely with each other. Radio Modems in the same network with a different destination address (than that of the transmitter) will listen to all transmissions to stay synchronized, but will not send any of the data out their serial ports.

AT Command: DT

Parameter Range: 0 - 0xFFFF

# of bytes returned: 2

Default Parameter Value: 0

Related Commands: ID (Modem ID), HP (Channel), MK (Address Mask)

- **FH (Force Wake-up Initializer) Command**

<AT Command: Sleep (Low Power)> FH Command is used to force a Wake-up Initializer to be sent on the next transmission. WR (Write) Command does not need to be issued with FH Command. Use only with cyclic sleep modes active on remote modules.

- **FL (Software Flow Control) Parameter**

<Non-AT Settable Parameter: Serial Interfacing> FL Parameter is used to adjust serial flow control. Hardware flow control is implemented with the XCite Radio Modem as the CTS pin (which

regulates when serial data can be transferred to the radio modem). FL Parameter can be used to allow software flow control to also be enabled. The XON character to use is 0x11 (“17” decimal). The XOFF character to use is 0x13 (“19” decimal)

Parameter Range: 0 – 1

# of bytes returned: 1

Default Parameter Value: 0

Parameter Value	Configuration
0	No Software Flow Control
1	Use Software Flow Control

Fig 4.22 FL Parameter

- **HP (Channel) Command**

<AT Command: Networking> HP Command is used to set the radio modem channel number. A channel is one of three layers of addressing available to the XCite Radio Modem. In order for radio modems to communicate with each other, the modules must have the same channel number since each channel uses a different hopping sequence or single frequency. Different channels can be used to prevent modules in one network from listening to transmissions of another.

The XCite Radio Modem can operate both in Hopping and Single Frequency Channel Modes. Switching between Single Channel and Hopping Modes can only be done only using the “Function Set” dropdown list on the “Modem Configuration” tab of the X-CTU Software.

AT Command: HP

Hopping Channel Range: 0 – 6

Single Frequency Channel Range: 0 - 0x18

# of bytes returned: 1

Default Parameter Value: 0

Related Parameters: DT (Destination Address), ID (Modem ID), MK (Address Mask)

A "**Hopping Channel**" is a channel comprised of a group of frequencies. When in Hopping Channel Mode, the radio modem hops between the frequencies them when transmitting data. This option utilizes FHSS (Frequency Hopping Spread Spectrum) technology. This option helps bolster security in wireless data communications and also makes the system less prone to interference.

The 25 center frequencies available in **Single Frequency Channel Mode** are spaced 300 KHz apart. Since each channel occupies a 500 KHz bandwidth, adjacent channels therefore overlap. If modules are used in the same vicinity but on different channels, the channels used should occupy every other channel at a minimum separation. If channels used on different radio modems can be separated more they should be. This will provide for more isolation and less interference.

Parameter Value	Frequency (MHz)
0x00	910.5
0x01	910.8
0x02	911.1
0x03	911.4
0x04	911.7
0x05	912.0
0x06	912.3
0x07	912.6
0x08	912.9
0x09	913.2
0x0A	913.5
0x0B	913.8
0x0C	914.1
0x0D	914.4
0x0E	914.7
0x0F	915.0
0x10	915.3
0x11	915.6
0x12	915.9
0x13	916.2
0x14	916.5
0x15	916.8
0x16	917.1
0x17	917.4
0x18	917.7

Fig 4.23 Frequency assignment with the command

- **HT (Time before Wake-up Initializer) Parameter**

<Non-AT Settable Parameter: Sleep (Low Power)> If any modules within range are running in a "Cyclic Sleep Setting", a wake-up initializer must be sent by the transmitter for the other radio modems to synchronize to the transmitter [see LH ("Wake-up Initializer Timer") Command]. When a receiving radio modem in Cyclic Sleep wakes, it must detect the wake-up initializer portion of the RF packet in order to synchronize to the transmitter and receive data. HT Parameter sets time period of inactivity (no serial or RF data is sent or received) before a Wake-up Initializer is sent. Base station tracks awake-status of remote radios. HT of base radio should be set shorter than ST (Time before Sleep) of remote radios.

From the receiver perspective, after “HT” time elapses and the ST (Time before Sleep) Parameter is met, the receiver goes into cyclic sleep. Once in cyclic sleep, the radio modem must first detect the wake-up initializer and synchronize to the transmitter before it can receive data. Thus, when time “HT” time elapses, the transmitter then knows it needs to send a long wake-up initializer for all receivers to be able to synchronize to its next transmission. Matching “HT” to the “ST” time on the receiver(s) guarantees that all receivers will detect the next transmission.

Parameter Range: 0 - 0xFFFF [x 100 ms]

# of bytes returned: 2

Default Parameter Value: 0xFFFF (long wake-up initializer will not be sent)

Related Parameters: LH (Wake-up Initializer Timer), SM (Sleep Mode), ST (Time before Sleep)

- **HV (Hardware Version) Command**

<AT Command: AT Command Options> Reads and returns the hardware version of the XCite Module.

AT Command: HV

Parameter Range: 0 – 0xFFFF [Read-only]

# of bytes returned: 2

- **ID (Modem VID) Parameter**

<Non-AT Settable Parameter: Networking> ID Parameter reads and edits the module’s VID. VID is a MaxStream-specific acronym that stands for “Vendor Identification Number”. Modules can only communicate with other modules having the same VID.

Parameter Range: 0 - 0x7FFF (above this range is Read-only)

# of bytes returned: 2

Default Parameter Value: 0x3332

- **LH (Wake-up Initializer Timer) Parameter**

<Non-AT Settable Parameter: Sleep (Low Power)> LH Parameter adjusts the duration of time in which the wake-up initializer is sent. When receiving modules are put into the Cyclic Sleep Mode, they power-down after a period of inactivity (specified by ST (Time before Sleep) Parameter) and will periodically awaken and listen for transmitted data. In order for the receiving modules to initialize with the transmitter, they must detect ~35ms of the wake-up initializer. LH Parameter must be used whenever a receiver is operating in Cyclic Sleep Mode. This lengthens the wake-up initializer to a specific amount of time (in x 100 ms). The long wake-up initializer must be longer than the cyclic sleep time that is determined by SM (Sleep Mode) Command. If the wake-up initializer time were less than the Cyclic Sleep interval, the connection would be at risk of missing the wake-up initializer transmission. The data and figures on the next page illustrate this behavior:

Parameter Range: 0 – 0xFF [x 100 ms]

# of bytes returned: 1

Default Parameter Value: 0x01 (0.1 second)

Related Parameters: HT (Time before Wake-up Initializer), SM (Sleep Mode), ST (Time before Sleep)

The length of the wake-up initializer exceeds the time interval of Cyclic Sleep. The receiver is guaranteed to detect the wake-up initializer and receive the accompanying payload data

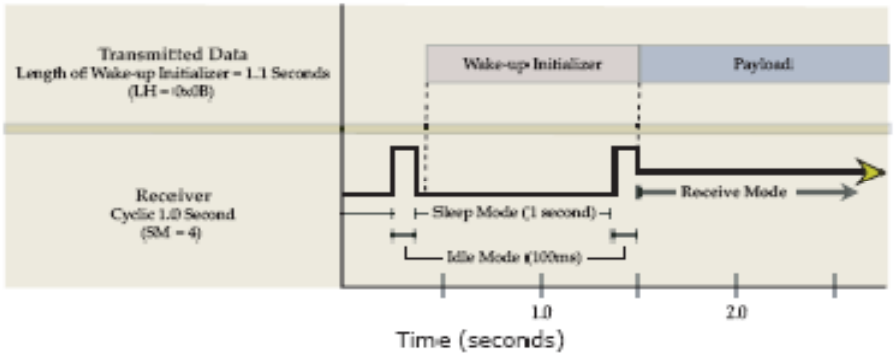


Fig 4.24 LH Parameter original

The length of the wake-up initializer is shorter than the time interval of Cyclic Sleep. This configuration is vulnerable to the receiver waking and missing the wake-up initializer (and therefore also the accompanying payload data).

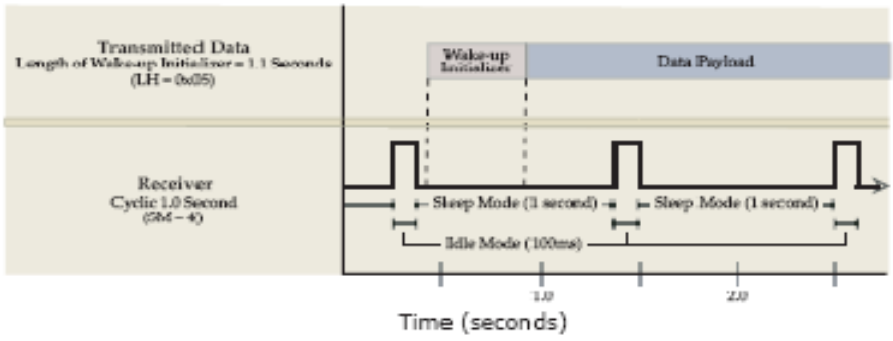


Fig 4.25 LH Parameter error case

- **MK (Address Mask) Command**

<AT Command: Networking> MK Command is used to set the radio modem address mask. All RF packets contain the Destination Address of the transmitting radio modem. When an RF packet is received, the transmitter's Destination Address is logically "ANDed" (bitwise) with the Address Mask of the receiver. The resulting value must match the Destination Address or the Address Mask

of the receiver for the packet to be received and sent out the receiving modem serial port. If the "ANDed" value does not match either the Destination Address or the Address Mask of the receiver, the packet is discarded. (All "0" values are treated as "irrelevant" values and ignored.)

AT Command: MK

Parameter Range: 0 – 0xFFFF

# of bytes returned: 2

Default Parameter Value: 0xFFFF (When set to this value, the Destination Address of the transmitter must exactly match the Destination Address of the receiver.)

Related Commands: DT (Destination Address), HP (Channel)

- **NB (Parity) Parameter**

<Non-AT Settable Parameter: Serial Interfacing> NB Parameter allows parity for the module to be changed. Parity is an error detection method in which a bit (0 or 1) is added to each group of bits so that it will have either an odd number of 1's or an even number of 1's. For example, if parity is odd, then any group of bits that arrives with an even number of 1's must contain an error.

Parameter Range: 0 - 4

# of bytes returned: 1

Default Parameter Value: 0

Parameter Value	Configuration
0	8-bit (no parity) or 7-bit (with any parity)
1	8-bit even parity
2	8-bit odd parity
3	8-bit mark parity
4	8-bit space parity

Fig 4.26 NB Parameter

- **PW (Pin Wake-up) Parameter**

<Non-AT Settable Parameter: Sleep (Low Power)> Under normal operation, a radio modem in Cyclic Sleep Mode cycles from an active state to a low-power state at regular intervals until data is ready to be received. If PW Parameter is set to 1, the SLEEP Pin (Pin 2 of the OEM RF Module) can be used to awaken the module from Cyclic Sleep. If the SLEEP Pin is de-asserted (low), the radio modem will be fully operational and will not go into Cyclic Sleep. Once SLEEP is asserted, the radio modem will remain active for the period of time specified by ST (Time before Sleep) Command, and will return to Cyclic Sleep Mode (if no data is ready to be transmitted). PW Command is only valid if Cyclic Sleep has been enabled using SM Command.

Parameter Range: 0 – 1

# of bytes returned: 1

Default Parameter Value: 0

Related Parameters: SM (Sleep Mode), ST (Time before Sleep)

Parameter Value	Configuration
0	Disable Pin Wake-Up
1	Enable Pin Wake-Up

Fig 4.27 PW Parameter

- **RE (Default Configuration) Command**

<AT Command: AT Command Options> RE Command restores all AT-settable parameters to factory default settings. However, RE Command will not write the default values to non-volatile

memory. Unless the WR (Write) Command is issued after the RE Parameter, the default settings will not be saved in the event of radio modem reset or power-down.

AT Command: RE

Related Command: WR (Write)

- **RT (DI2 Configuration) Parameter**

<Non-AT Settable Parameter: Serial Interfacing> RT Parameter enables RTS Mode.

Parameter Range: 0 - 1

# of bytes returned: 1

Default Parameter Value: 0

Parameter Value	Configuration
0	Disabled
1	Enables RTS Handshaking

Fig 4.28 RT Parameter

- **SB (Stop Bits) Parameter**

<Non-AT Settable Parameter: Serial Interfacing> SB Parameter allows the user set the number of stop bits used in data transmission.

Parameter Range: 0 - 1

# of bytes returned: 1

Default Parameter Value: 0

Parameter Value	Configuration
0	1 stop bit
1	2 stop bits

Fig 4.29 SB Parameter

- **SH (Serial Number High) Command**

<AT Command: AT Command Options> SH Command reads and returns the module serial number high word.

AT Command: SH

Parameter Range: 0 – 0xFFFF [Read-only]

# of bytes returned: 2

Related Command: SL (Serial Number Low)

- **SL (Serial Number Low) Command**

<AT Command: AT Command Options> SL Command reads and reports the module serial number low word.

AT Command: SL

Parameter Range: 0 – 0xFFFF [Read-only]

# of bytes returned: 2

Related Command: SH (Serial Number High)

- **SM (Sleep Mode) Parameter**

<Non-AT Settable Parameter: Sleep Mode (Low Power)> SM Parameter is used to adjust Sleep Mode settings. By default, Sleep Mode is disabled and the radio modem remains continually active. SM Parameter allows the radio modem to run in a lower-power state and be configured in one of eight settings.

Cyclic Sleep settings wake the radio modem after the amount of time designated by SM Command. If the radio modem detects a wake-up initializer during the time it is awake, it will synchronize with the transmitting radio modem and start receiving data after the wake-up initializer runs its duration. Otherwise, it returns to Sleep Mode and continue to cycle in and out of sleep until the wake-up initializer is detected. If a Cyclic Sleep setting is chosen, the ST, LH and HT parameters must also be set as described in the "Sleep Mode" section of this manual.

Parameter Range: 0 - 8

# of bytes returned: 1

Default Parameter Value: 0

Related Parameters: LH (Wake-up Initializer Timer), HT (Time before Wake-up Initializer), PW (Pin Wake-up), ST (Time before Sleep)

Parameter Value	Configuration
0	No Sleep
1	Pin Sleep
2	Serial Port Sleep
3	Cyclic 0.5 seconds
4	Cyclic 1.0 seconds
5	Cyclic 2.0 seconds
6	Cyclic 4.0 seconds
7	Cyclic 8.0 seconds
8	Cyclic 16.0 seconds

Fig 4.30 SM Parameter

- **ST (Time before Sleep) Parameter**

<Non-AT Settable Parameter: Sleep Mode (Low Power)> ST Parameter sets the period of time (in tenths of seconds) in which the radio modem remains inactive before entering into Sleep Mode. For example, if the ST Parameter is set to 0x64 ("100" decimal), the radio modem will enter into Sleep Mode after 10 seconds of inactivity (no transmitting or receiving). This command can only be used if either Cyclic Sleep or Serial Port Sleep Mode settings have been selected using SM (Sleep Mode) Parameter.

Parameter Range: 0x10 – 0xFFFF [x 100 ms]

# of bytes returned: 2

Default Parameter Value: 0x64 ("100" decimal)

Related Parameters: SM (Sleep Mode), LH (Wake-up Initializer Timer), HT (Time before Wake-up Initializer)

- **VR (Firmware Version) Command**

<AT Command: AT Command Options> Reads and returns the currently loaded firmware version of the XCite Radio Modem.

AT Command: VR

Parameter Range: 0 – 0xFFFF [Read-only]

# of bytes returned: 2

- **WR (Write) Command**

<AT Command: (Special)> WR Command writes all configurable parameters to non-volatile memory. Using WR Command saves parameters to the radio modem's persistent memory. (This means that the parameters remain in the radio modem's memory until explicitly overwritten by future uses of WR Command.)

AT Command: WR

4.12 Advanced Networking and Security

4.12.1 Filtration Layers

The XCite Radio Modems utilize three layers of addressing to communicate between radio modems. The network layers are depicted below. Only radio modems with the matching addresses are able to communicate. The main layers of XCite Networking and Addressing are:

- Vendor Identification Number (ATID)
- Channel (ATHP)
- Destination Address (ATDT)

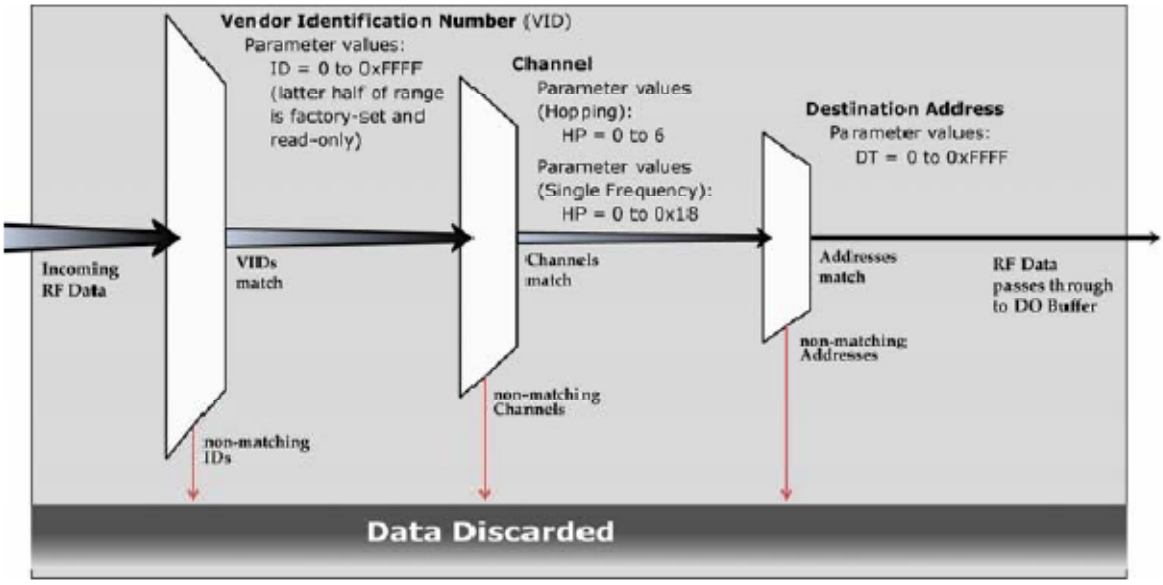


Fig 4.31 Filtration Layers

Each network layer provides a separate layer of filtration. The Vendor Identification Number (VID) provides the first layer of filtration through the ID (Modem ID) Parameter. If the incoming RF data carries a matching VID number, the data continues through to the subsequent Channel and Destination Address layers. The Destination Address is the last network layer and provides the most granular form of filtration. If at any point during the incoming RF data flow the numbers in question do not match, the data is discarded.

XCite Modules and RF Modems are built around a peer-to-peer protocol that inherently supports a multidrop type network (similar to RS-485). In their default state, any XCite radio modem will communicate with any other XCite radio modem in its default state.

- **Vendor Identification Number (ATID)**

The bottom half of the ID (Vendor Identification Number) Parameter range is user-settable. The upper half of the range is factory-set and read-only. The value of the ID Command is called the Vendor Identification Number (VID). A unique VID is available upon special request. The VID is programmed to the XCite Module at the factory and is stored in the module's permanent memory. Only modules with matching VIDs can communicate with each other.

VID addressing ensures that radio modems ignore transmissions and receptions of XCite Radio Modems having a different VID in the same vicinity. To request a unique VID, contact MaxStream to obtain the VID Request Form.

- **Channel (ATHP)**

Channels provide a network layer from which channels can be used for isolation. HP (Channel) Parameter is used to define channel values.

Hopping Channel Mode: HP Parameter value range is 0 through 6

Single Frequency Channel Mode: HP range is 0 through 0x18 (decimal range: 0 – 24)

In "Hopping Channel Mode", each channel utilizes a different pseudo-random hopping sequence to navigate through shared hopping channels. In the event that two modules from different networks collide on a channel, the two modules will jump to separate channels on the next hop. Multiple module pairs can operate in the same vicinity with minimal interference from each other.

- **Destination Address (ATDT) and Address Mask (ATMK)**

XCite Destination Addresses and Masks provide the means to set up global or local addresses for establishing module groups, subnets, etc. The Destination Address network layer provides for more granular isolation of radio modems. The XCite Destination Addresses and Masks can be used to:

- Set up point-to-point and point-to-multipoint network configurations
- Provide greater flexibility in establishing module groups, subnets, etc.

Each radio modem in a network can be configured with a 16-bit Destination Address to establish selective communications within a network. This address is set to one of 65535 values using DT (Destination Address) Command. The default Destination Address is 0.

All radio modems with the same Destination Address can transmit and receive data among themselves. Radio modems having different Destination Addresses still detect and listen to the data (in order to maintain network synchronization); however, the data is discarded data rather than passing on through the DO pin.

XCite Radio Modems are packet based. This means all data shifted into one module is packetized and sent out the antenna port. Because XCite modules use a peer-to-peer architecture, all modules on the same channel (ATHP) will receive the packet and decide whether to pass it to the host or to throw it away. Each transmitted packet contains information about the transmitting module.

Any module that receives a packet will check the address values and decide what to do with the packet. The options are as follows:

- Receive the packet as a global packet
- Receive the packet as a local packet
- Discard the packet

Address Mask

The mask parameter can be used to allow a base module to receive data from a range of addresses. It may also be used to configure "subnets" of modules that communicate in a group together.

See below for the Pseudo 'C' Code that qualifies the Destination Addresses and address masks.

The Pseudo Code uses the bit-wise "AND" operation, "&". This operation is performed bit by bit on each of the 16 bits in the TXDT, RXDT and RXMK parameters.

Bit-wise AND Operation ("&")		
Operand 1	& Operand 2	= Result
0	0	0
0	1	0
1	0	0
1	1	1

Fig 4.32 Bit-wise AND Operation

For example: Hexadecimal: 0x3 & 0x9 = 0x1

The Address Mask can be used as an additional method of facilitating communications between modules. The Address Mask can be set to one of 65535 possible values using MK (Address Mask) Command. The default value of the MK Parameter is 0xFFFF.

All transmitted data packets contain the Destination Address of the transmitting module. When a transmitted packet is received by a module, the Destination Address of the transmitter (contained in the packet) is logically "ANDed" (bitwise) with the Address Mask of the Receiver. If the resulting value matches the Destination Address of the Receiver, or if it matches the Receiver Address Mask, the packet is accepted. Otherwise, the packet is discarded.

Note: When performing this comparison, any "0" values in the Receiver Address Mask are treated as irrelevant and are ignored.

Chapter 5

CAR MODULE

The car module is installed in the vehicle. When the vehicle enters in the range of base, it receives the interrogation message and it replies it with its data

5.1 Architecture

The basic architecture of car has following components

1. Microcontroller (89C52)
2. Max-232 IC
3. RF modem
4. Infrared Receiver

5.1.1 Microcontroller

The car module uses 89C52 microcontroller. The microcontroller contains the data of the vehicle in which it is installed. The vehicle data consists of registration no., chassis no. , engine no., owners name and make of the car.

5.1.2 Max-232 IC

This IC converts voltage levels to and from +5V and +12V. it is placed between microcontroller and RF modem. Microcontroller operates at +5V whereas RF modem communicates through serial port at +12V. So there is a need to convert the signal levels going through RF modem to microcontroller and vice versa. Data coming through RF modem is converted from +12V to +5V. Similarly data coming through microcontroller is converted from +5V to +12V.

5.1.3 RF modem

The RF modem communicates with base via a wireless channel. It operates in ISM 900MHz band. The modulation scheme used is Frequency Hopping Spread Spectrum. The specifications and features of RF modem are given in chapter 4.

5.1.4 Infrared Receiver

The infrared receiver installed in the car module to detect the base. An infrared transmitter is installed at the base. It has range of 10 feet. When the car is at a distance of 10 feet, the IR receiver detects it and tells the car module that it is approaching barrier.

5.2 Working

The car module when enters in the range of the base, it receives an interrogation message. The car module then replies with its data including registration no., chassis no. , engine no., owners name and make of the car. The base receives this data and sends an authentication message to the car module. The car receives this message and goes in the wait loop. When the car reaches the base, the infrared receiver detects the infrared wave and then car module sends a notification message to base to open the barrier.

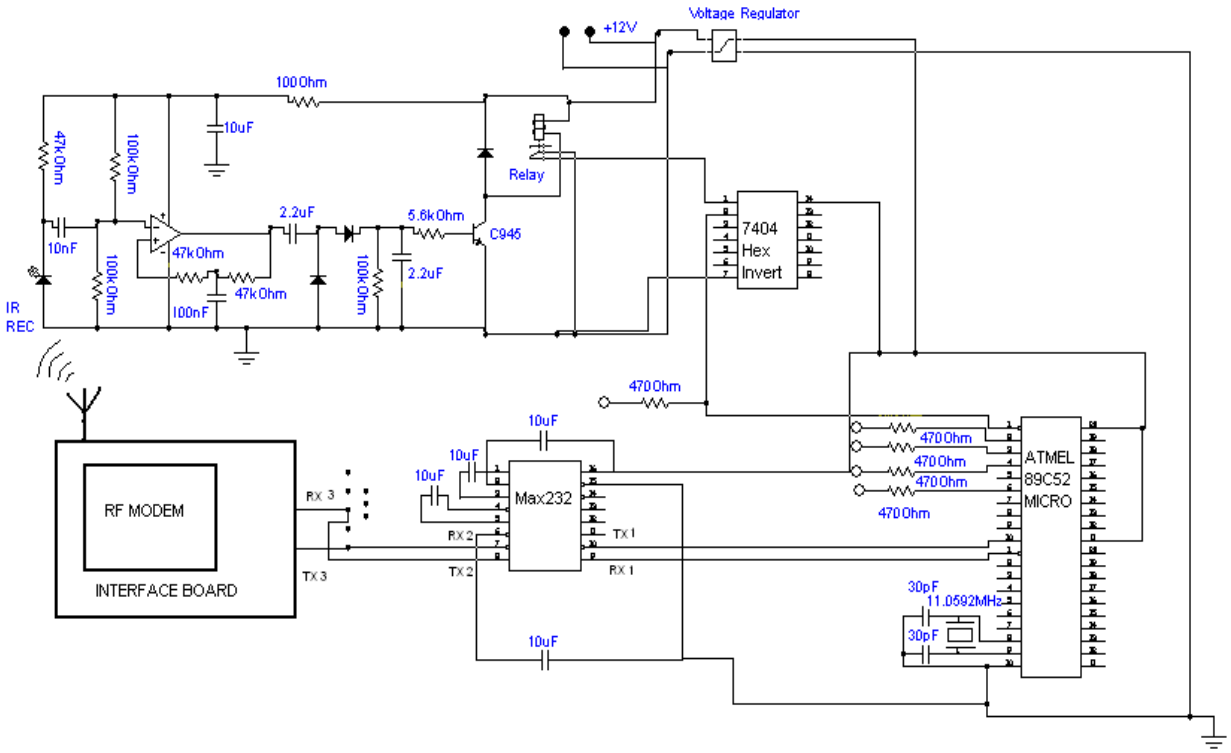


Fig 5.1 Circuit Diagram of Car Module

5.3 Circuit components

- AT89C52 Microcontroller
- Max-232
- 7404 hex converter
- RF modem
- Interface board
- Relay
- Voltage regulator
- Operational amplifier
- 11.0592 Mhz crystal
- 470 ohms

- 47 kohms
- 100 kohms
- 100 ohms
- 5.6 kohms
- 2.2 uF
- 30 pF
- 10 uF
- 100 uF
- 100 nF
- C945 npn transistor
- LEDs

5.4 Flow chart of Car Module

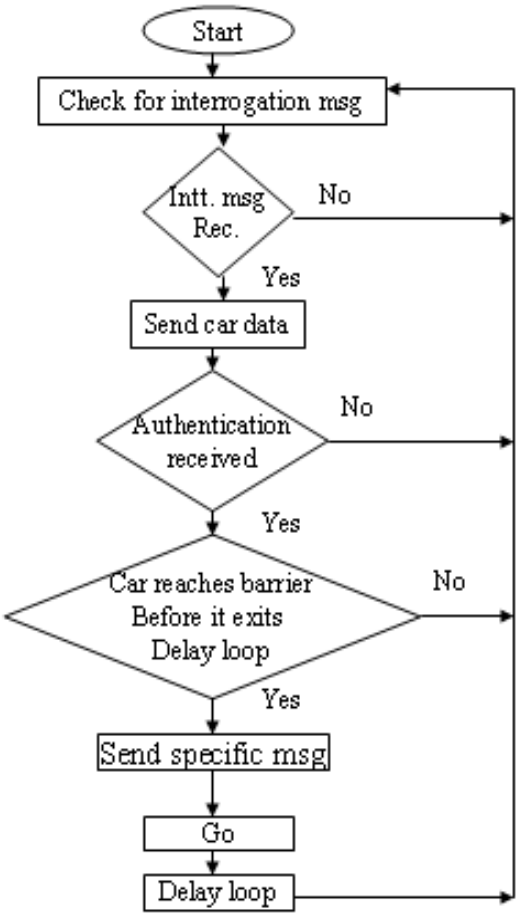


Fig 5.2 Flow Chart for Car Module

5.5 Code for the microcontroller installed in car module

```
/*-------------------------------------------------------------------------

Wireless Registration System for Vehicles; The Car module.

-------------------------------------------------------------------------*/

#include <atmel/at89x52.h>            /* special function register declarations   */
                                      /* for the intended 8051 derivative         */

#include <stdio.h>                    /* prototype declarations for I/O functions */
```

```
/*-------------------------------------------------
The main C function.  Program execution starts
here.
-------------------------------------------------*/
 char A [12];              /* A character array that is used to receive the authentication msg */

 char REG [12]="@OK#RLD 674$";                              /* A character array which contains
                                                            the required authentication msg */

 sbit P10=P1^0;

 char atr;

 char check1;                /* A character that is used to answer the interrogation of base */

 long int c;

 long int i;                 /* Variable for the FOR LOOP */

 unsigned int j;             /* Variable for the FOR LOOP */

 unsigned int x;

 unsigned int y;

 int flag;                   /* Variable that ensures the reception of authentication */

 int port;
```

```
  void main (void)
{

/*-------------------------------------------------

-------------------------------------------------*/

  SCON  = 0x50;                 /* mode 1: 8-bit UART, enable receiver   */
  TMOD |= 0x20;                 /* timer 1 mode 2: 8-Bit reload          */
  TH1   = 0xf4;                 /* reload value 2400 baud                */
  TR1   = 1;                    /* timer 1 run                           */
  TI    = 1;                    /* TI:   set TI to send first char of UART    */



/*-------------------------------------------------
Note that an embedded program never exits (because
there is no operating system to return to).  It
must loop and execute forever.
-------------------------------------------------*/

P0=0xFF;
P1=0xFF;
P2=0xFF;
P3=0xFF;

while(1)
  {
```

```
  port=0;
  flag=0;
  while(flag==0)
   {
   P1=0xFF;
   while (1)
   {
      while (RI!=1);
            check1=SBUF;
      if (check1 == '%')
      {
      P1_1=0;                    /* Toggle P1.1 */
            break;
            }
            RI=0;
   }
     RI=0;
           printf ("*RLD 674,MEHRAN 2006,ALI JAVED,E171234,C432567;");                    /* Sending the car
data */
     P1_2=0;                    /* Toggle P1.2 */


     for (c=0;c<=76800;c++)
           {
     atr=SBUF;
            if (atr == '@')
             {
                     while (SBUF != '$')
```

```
        {for (i=0; i<=11; i++)
    {
    while (RI!=1);
                    A[i]=SBUF;
    RI=0;
                }
   }

  for (i=0; i<=11; i++)
        {if (A[i] == REG[i])
         {if (i==11)
                {c=76800;
       flag=1;
                                P1_3=0;                 /* Toggle P1.3 */
                                }

    }
         else
           {c=76800;
      break;}
         }
        }
        }
}
      for (i=1; i<=360000; i++)
         {
                for (j=1; j<=60; j++)
                 {
```

```
                if ( P10==0)
                        {printf("&RLD 674;");
                         P1_4=0;                    /* Toggle P1.4 */
                                port=1;
                                break;}
                 }

               if (port == 1)
               {
                 for (x=1; x<=60000; x++)
                  {
              for (y=1; y<=120; y++)
                           {}
                        }
                 break;
               }


          }
       }
}
```

Chapter 6

BASE MODULE

The Base Module will be installed at the entry points in a campus or near the barriers in case of Toll Plazas. The function of the base module will be to register the cars in its range, allow them to pass if the cars are valid and trigger the alarms in case of invalid cars.

6.1 Architecture/Components

1. Base Station Computer
2. RF Modem (for details see Chapter 4)
3. Infrared Transmitters (2 in total)
4. Infrared Receiver and Decision Circuit

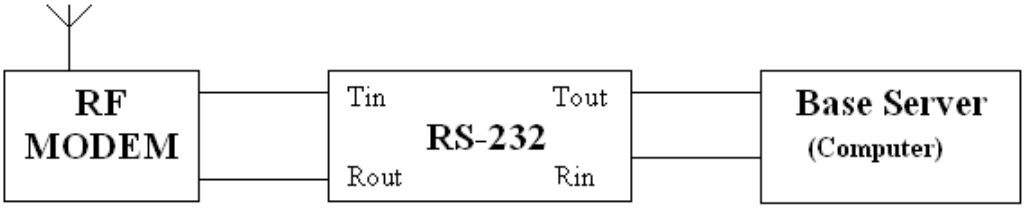


Fig 6.1 Block Diagram of Base Module

Note: We have done the programming of Base Station Computer in Visual Basic provided in the software Visual Studio 2005.

6.2 Working

The bas station computer is programmed to continuously send an interrogation over its range. In case of any response to the interrogation message if the base computer receives any car data, it compares the data with the pre-saved data in its database. If the car is allowed to pass, the base sends an authentication message to the car module. The purpose of the authentication message is to counter erroneous transmission of data. One of the two infrared transmitters will be installed at the entrance in such a way that when the car reaches within 10 ft range of the barrier, it reminds the car module to send its specific notification message. As the notification message is received the registered car will be allowed to pass. The second infrared transmitter and the infrared receiver along with the decision circuit can detect un-registered cars. (For further details on Infrared decision circuit refer to Chapter No. 7)

6.3 Code for base station computer

```
Imports System.IO.Ports
Imports System.Threading

Public Class Form1
    Private serialObject As New SerialPort
    Private thread As Thread
    Dim fileStream As System.IO.FileStream
    Dim writer As System.IO.StreamWriter
    Dim reader As System.IO.StreamReader
    Dim dataRead As Boolean
    Dim readingThread As System.Threading.Thread
```

```
    Dim buffer As String
    Dim ok As Boolean
    Dim charArray(0 To 25) As Char
    Public Declare Sub Out Lib "inpout32.dll" Alias "Out32" (ByVal PortAddress As
Short, ByVal Value As Short)
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

    End Sub

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click

        If Button1.Text.Equals("Start") Then

            serialObject.BaudRate = 2400
            serialObject.DataBits = 8
            serialObject.Parity = Parity.None
            serialObject.PortName = "COM1"
            serialObject.StopBits = StopBits.One
            PictureBox2.Image=
Bitmap.FromFile("C:\\CarPictures\\StartingPic.bmp")
            serialObject.DiscardNull = True
            serialObject.ReadBufferSize = 200
            serialObject.ReadTimeout = 3000
            Out(&H378S, 240)
            serialObject.Open()
            Try
                serialObject.ReadChar()
```

```
        Catch ex As Exception

        End Try
        Timer1.Enabled = True
        Timer2.Enabled = True
        ok = True
        Timer2_Tick(vbNull, e)

        Button1.Text = "Stop"
    Else
        serialObject.Close()
        Timer1.Enabled = False
        Timer2.Enabled = False
        Button1.Text = "Start"
    End If


End Sub

Private Function process()
    Dim stream As System.IO.FileStream
    Dim reader As System.IO.StreamReader
    Dim data As String
    Timer2.Enabled = False

    If (buffer.StartsWith("&")) Then
        data = buffer.Substring(1, buffer.Length - 1)
        If (System.IO.File.Exists("C:\\CarsInRange\\" & data & ".txt")) Then
            Try
```

```
                    Try
                        System.IO.File.Copy("C:\\CarsInRange\\" & data & ".txt",
"C:\\PassingCarsData\\" & data & ".txt", True)
                        System.IO.File.Delete("C:\\CarsInRange\\"   &   data   &
".txt")
                    Catch ex As Exception
                        MsgBox("Could not move" & ex.Message.ToString)
                    End Try
                    updateInRangeCars()
                    currentlyPassing_txt.Text = data
                    PictureBox1.Image   =   Bitmap.FromFile("C:\\CarPictures\\"   &
data & ".jpg")
                    Out(&H378S, 15)
                Catch ex As Exception
                    MsgBox(ex.Message.ToString)
                End Try

            End If




        ElseIf (buffer.StartsWith("*")) Then
            Dim regNo As String
            Dim tempRead As String
            Dim i As Integer
            data = buffer.Substring(1, buffer.Length - 1)
            regNo = data.Substring(0, data.IndexOf(","))
```

```
            stream = New System.IO.FileStream("C:\\AllCarsData\\" & regNo &
".txt", IO.FileMode.Open)
            reader = New System.IO.StreamReader(stream)
            tempRead = reader.ReadLine()
            If tempRead.ToLower().Equals(data.ToLower()) Then
                If System.IO.File.Exists("C:\\AllCarsData\\" & regNo & ".txt")
Then
                    System.IO.File.Copy("C:\\AllCarsData\\" & regNo & ".txt",
"C:\\CarsInRange\\" & regNo & ".txt", True)
                    tempRead = "@OK#" & regNo & "$"
                    charArray = tempRead.ToCharArray()
                    i = 0
                    System.Threading.Thread.CurrentThread.Sleep(100)
                    While Not charArray(i) = "$"
                        serialObject.Write(charArray, i, 1)
                        i = i + 1
                    End While
                    serialObject.Write(charArray, i, 1)
                End If
            End If

            reader.Close()
            stream.Close()
            updateInRangeCars()



        End If

        Timer2.Enabled = True
```

```
    End Function
    Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Timer1.Tick
        ok = False
        Dim files As String()
        Dim i As Integer
        files = System.IO.Directory.GetFiles("C:\\CarsInRange")
        For i = 0 To files.Length - 1
            If (System.DateTime.Now.TimeOfDay.TotalMinutes -
System.IO.File.GetCreationTime(files(i)).TimeOfDay.TotalMinutes) > 4 Then
                System.IO.File.Delete(files(i))
                updateInRangeCars()
            End If
        Next
        ok = True
    End Sub

    Private Sub Timer2_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Timer2.Tick
        If (ok) Then
            serialObject.Write("%")
            Out(&H378S, 240)
            Try
                buffer = serialObject.ReadTo(";")
                process()
            Catch ex As Exception
            End Try
        End If
```

```
    End Sub

    Private Function updateInRangeCars()
        Dim files As String()
        Dim i As Integer
        files = System.IO.Directory.GetFiles("C:\CarsInRange")
        carsInRange_txt.Clear()
        For i = 0 To files.Length - 1

carsInRange_txt.AppendText(files(i).Substring(files(i).LastIndexOf("\")    +    1,
files(i).LastIndexOf(".") - files(i).LastIndexOf("\")) & vbCrLf)
            Application.DoEvents()
        Next
    End Function


End Class
```

6.4 Graphic User Interface

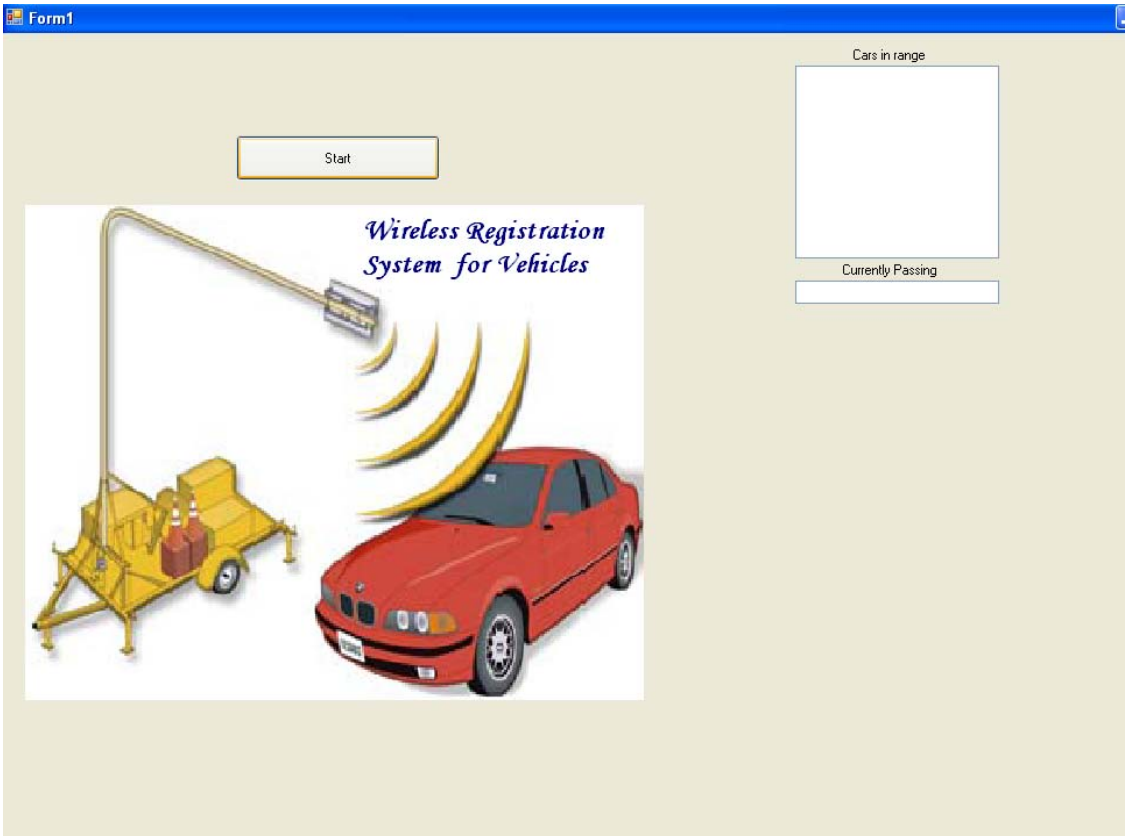


Fig 6.2 Graphical User Interface

6.5 Flowchart for base module

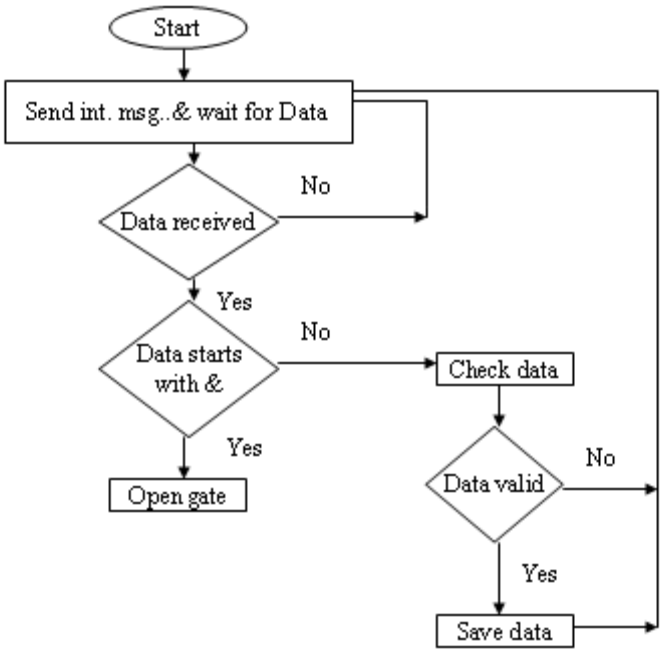


Fig 6.3 Flow Chart of Base module

CHAPTER 7

INFRARED LINE CUTTER SYSTEM

7.1 BASIC OPERATION

This system detects the arrival of unregistered or illegal cars at the entrance of the base station. The transmitter is continuously transmitting infrared waves. The receiver at the base is receiving these waves and through amplifier and rectifier circuits it is fed to the delay which acts as a switch. It gives output high at +5 V. the decision circuit attached with it checks if any car cuts infrared line. The registered car would take less than ten seconds to cross the gate and pass by the infrared line. If anything stays more than twelve seconds the decision circuit will detect some invalid or unregistered car and an alarm will be rung.

7.2 Architecture

The system comprises of two modules:

1. Transmitter circuit
2. Receiver and Decision circuit

7.2.1 Transmitter Circuit:

The transmitter circuit comprises following main components:

- Oscillator
- Coupling Circuit
- Transmitter LED

The oscillator generates a wave of 38 KHz. This wave is fed to the coupling circuit which is further transmitted as an infrared wave via transmitter LED.

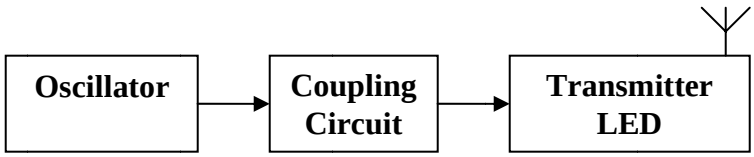


Fig 7.1 Transmitter Circuit (Block Diagram)

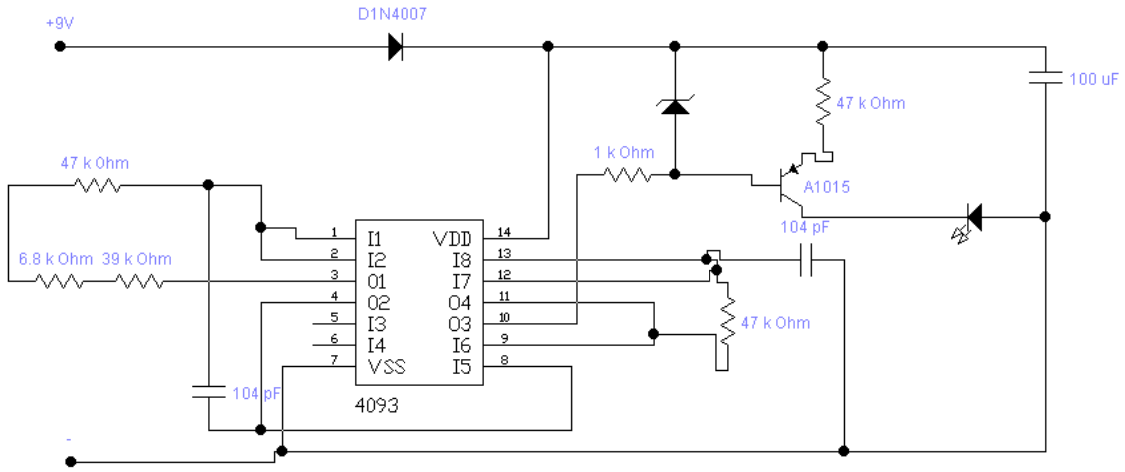


Fig 7.2 Circuit Diagram

7.2.2 Receiver and Decision Circuit:

This circuit comprises of following main components:

- Photo-transistor Diode
- Operational amplifier
- Rectifier circuit
- Relay (Switch)
- Microcontroller (89c52)
- Alarm

The photo transistor diode receives the infrared waves from the transmitter circuit (LOS communication). This signal is amplified through an operational amplifier. Then this signal is converted into a DC signal by the rectifier circuit. After this conversion the DC signal output is fed to the relay (switch). The relay acts as switch and whenever the infrared line is cut, it gives low output to the decision circuit. Decision circuit consists of a microcontroller and an alarm. The microcontroller is programmed in such a way that it detects any obstacle in the path of LOS infrared link and if the obstacle does not clear the path within twelve seconds, it rings an alarm and security is being alerted

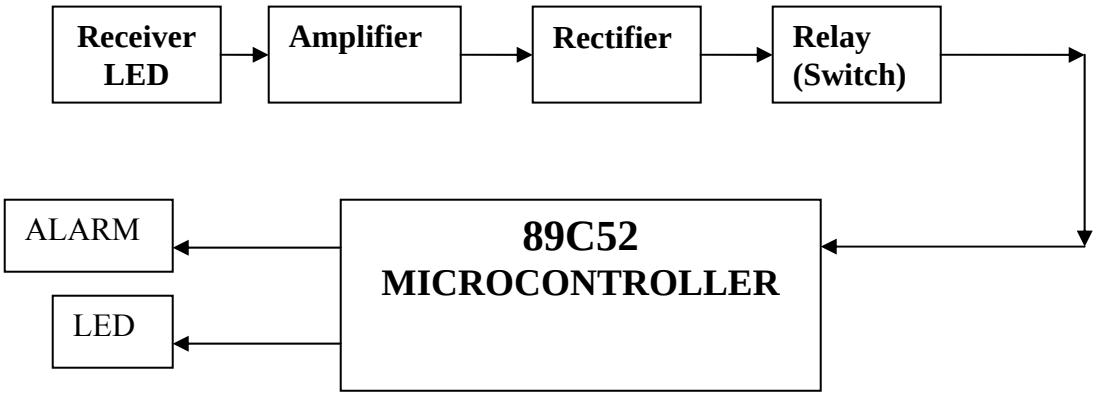


Fig 7.3 Receiver and Decision Circuit (Block Diagram)

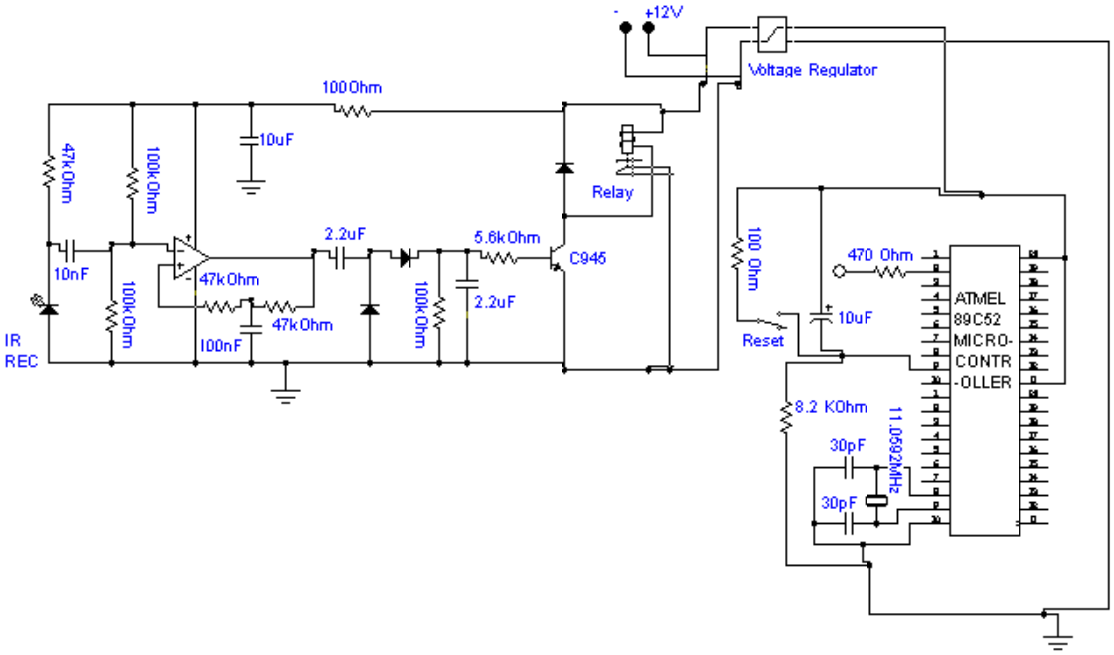


Fig 7.4 Circuit Diagram

APPENDIX A

Maxim 232 Description

General Description

The MAX220–MAX249 family of line drivers/receivers is intended for all EIA/TIA-232E and V.28/V.24 communications interfaces, particularly applications where ±12V is not available.

These parts are especially useful in battery-powered systems, since their low-power shutdown mode reduces power dissipation to less than 5µW. The MAX225,

MAX233, MAX235, and MAX245/MAX246/MAX247 use no external components and are recommended for applications where printed circuit board space is critical.

Applications

- Portable Computers
- Low-Power Modems
- Interface Translation

- Battery-Powered RS-232 Systems

- Multidrop RS-232 Networks

ABSOLUTE MAXIMUM RATINGS—MAX220/222/232A/233A/242/243

Supply Voltage (VCC) .....-0.3V to +6V

V+ (Note 1) .....(VCC - 0.3V) to +14V

V- (Note 1) .....+0.3V to +14V

Input Voltages

TIN.....-0.3V to (VCC - 0.3V)

RIN (Except MAX220) .....±30V

RIN (MAX220).....±25V

TOUT (Except MAX220) (Note 2) .....±15V

TOUT (MAX220).....±13.2V

Output Voltages

TOUT.....±15V

ROUT.....-0.3V to (VCC + 0.3V)

Driver/Receiver Output Short Circuited to GND.....Continuous

Continuous Power Dissipation (TA = +70°C)

16-Pin Plastic DIP (derate 10.53mW/°C above +70°C)..842mW

18-Pin Plastic DIP (derate 11.11mW/°C above +70°C)..889mW

20-Pin Plastic DIP (derate 8.00mW/°C above +70°C) ..440mW

16-Pin Narrow SO (derate 8.70mW/°C above +70°C) ...696mW

16-Pin Wide SO (derate 9.52mW/°C above +70°C).....762mW

18-Pin Wide SO (derate 9.52mW/°C above +70°C).....762mW

20-Pin Wide SO (derate 10.00mW/°C above +70°C)....800mW

20-Pin SSOP (derate 8.00mW/°C above +70°C) 640mW

16-Pin CERDIP (derate 10.00mW/°C above +70°C).....800mW

18-Pin CERDIP (derate 10.53mW/°C above +70°C).....842mW

Operating Temperature Ranges

MAX2_ _AC_ _, MAX2_ _C_ _.....0°C to +70°C

MAX2_ _AE_ _, MAX2_ _E_ _ .....-40°C to +85°C

MAX2_ _AM_ _, MAX2_ _M_ _.....-55°C to +125°C

Storage Temperature Range .....-65°C to +160°C

Lead Temperature (soldering, 10s) (Note 3) .....+300°C

ELECTRICAL CHARACTERISTICS—MAX220/222/232A/233A/242/243

(VCC = +5V ±10%, C1–C4 = 0.1µF, MAX220, C1 = 0.047µF, C2–C4 = 0.33µF, TA = TMIN to TMAX, unless otherwise noted.)

PARAMETER	CONDITIONS		MIN	TYP	MAX	UNITS
RS-232 TRANSMITTERS						
Output Voltage Swing	All transmitter outputs loaded with 3kΩ to GND		±5	±8		V
Input Logic Threshold Low				1.4	0.8	V
Input Logic Threshold High	All devices except MAX220		2	1.4		V
	MAX220: V_{CC} = 5.0V		2.4			
Logic Pullup/Input Current	All except MAX220, normal operation			5	40	µA
	$\overline{SHDN}$ = 0V, MAX222/MAX242, shutdown, MAX220			±0.01	±1	
Output Leakage Current	V_{CC} = 5.5V, $\overline{SHDN}$ = 0V, V_{OUT} = ±15V, MAX222/MAX242			±0.01	±10	µA
	V_{CC} = $\overline{SHDN}$ = 0V	V_{OUT} = ±15V		±0.01	±10	
		MAX220, V_{OUT} = ±12V			±25	
Data Rate				200	116	kbps
Transmitter Output Resistance	V_{CC} = V+ = V- = 0V, V_{OUT} = ±2V		300	10M		Ω
Output Short-Circuit Current	V_{OUT} = 0V	V_{OUT} = 0V	±7	±22		mA
		MAX220			±60	
RS-232 RECEIVERS						
RS-232 Input Voltage Operating Range					±30	V
		MAX220			±25	
RS-232 Input Threshold Low	V_{CC} = 5V	All except MAX243 $R2_{IN}$	0.8	1.3		V
		MAX243 $R2_{IN}$ (Note 4)	-3			
RS-232 Input Threshold High	V_{CC} = 5V	All except MAX243 $R2_{IN}$		1.8	2.4	V
		MAX243 $R2_{IN}$ (Note 4)		-0.5	-0.1	

PARAMETER	CONDITIONS		MIN	TYP	MAX	UNITS
RS-232 Input Hysteresis	All except MAX220/MAX243, V_{CC} = 5V, no hysteresis in SHDN		0.2	0.5	1	V
	MAX220			0.3		
	MAX243			1		
RS-232 Input Resistance	T_A = +25°C (MAX220)		3	5	7	KΩ
			3	5	7	
TTL/CMOS Output Voltage Low	I_{OUT} = 3.2mA			0.2	0.4	V
	I_{OUT} = 1.6mA (MAX220)				0.4	
TTL/CMOS Output Voltage High	I_{OUT} = -1.0mA		3.5	V_{CC} - 0.2		V
TTL/CMOS Output Short-Circuit Current	Sourcing V_{OUT} = GND		-2	-10		mA
	Shrinking V_{OUT} = V_{CC}		10	30		
TTL/CMOS Output Leakage Current	$\overline{SHDN}$ = V_{CC} or $\overline{EN}$ = V_{CC} ($\overline{SHDN}$ = 0V for MAX222), 0V ≤ V_{OUT} ≤ V_{CC}			±0.05	±10	µA
EN Input Threshold Low	MAX242			1.4	0.8	V
EN Input Threshold High	MAX242		2.0	1.4		V
Operating Supply Voltage			4.5		5.5	V
V_{CC} Supply Current ($\overline{SHDN}$ = V_{CC}), figures 5, 6, 11, 19	No load	MAX220		0.5	2	µA
		MAX222/MAX232A/MAX233A/ MAX242/MAX243		4	10	
	3kΩ load both inputs	MAX220		12		
		MAX222/MAX232A/MAX233A/ MAX242/MAX243		15		
Shutdown Supply Current	MAX222/ MAX242	T_A = +25°C		0.1	10	µA
		T_A = 0°C to +70°C		2	50	
		T_A = -40°C to +85°C		2	50	
		T_A = -55°C to +125°C		35	100	
$\overline{SHDN}$ Input Leakage Current	MAX222/MAX242				±1	µA
$\overline{SHDN}$ Threshold Low	MAX222/MAX242			1.4	0.8	V
$\overline{SHDN}$ Threshold High	MAX222/MAX242		2.0	1.4		V
Transition Slew Rate	C_L = 50pF to 2500pF, R_L = 3kΩ to 7kΩ, V_{CC} = 5V, T_A = +25°C, measured from +3V to -3V or -3V	MAX222/MAX232A/MAX233/ MAX242/MAX243	6	12	30	V/µs
		MAX220	1.5	3	30	
Transmitter Propagation Delay TLL to RS-232 (Normal Operation), Figure 1	t_{PHLT}	MAX222/MAX232A/MAX233/ MAX242/MAX243		1.3	3.5	µs
		MAX220		4	10	
	t_{PLHT}	MAX222/MAX232A/MAX233/ MAX242/MAX243		1.5	3.5	
		MAX220		5	10	

PARAMETER	CONDITIONS		MIN	TYP	MAX	UNITS
Receiver Propagation Delay RS-232 to TLL (Normal Operation), Figure 2	t_{PHLR}	MAX222/MAX232A/MAX233/ MAX242/MAX243		0.5	1	µs
		MAX220		0.6	3	
	t_{PLHR}	MAX222/MAX232A/MAX233/ MAX242/MAX243		0.6	1	
		MAX220		0.8	3	
Receiver Propagation Delay RS-232 to TLL (Shutdown), Figure 2	t_{PHLS}	MAX242		0.5	10	µs
	t_{PHLS}	MAX242		2.5	10	
Receiver-Output Enable Time, Figure 3	t_{ER}	MAX242		125	500	ns
Receiver-Output Disable Time, Figure 3	t_{DR}	MAX242		160	500	ns
Transmitter-Output Enable Time ($\overline{SHDN}$ Goes High), Figure 4	tET	MAX222/MAX242, 0.1µF caps (includes charge-pump start-up)		250		µs
Transmitter-Output Disable Time ($\overline{SHDN}$ Goes Low), Figure 4	tDT	MAX222/MAX242, 0.1µF caps		600		ns
Transmitter + to - Propagation Delay Difference (Normal Operation)	tPHLT - tPLHT	MAX222/MAX232A/MAX233/ MAX242/MAX243		300		ns
		MAX220		2000		
Receiver + to - Propagation Delay Difference (Normal Operation)	tPHLR - tPLHR	MAX222/MAX232A/MAX233/ MAX242/MAX243		100		ns
		MAX220		225		

RS 232 Drivers

The typical driver output voltage swing is ±8V when loaded with a nominal 5k. RS-232 receiver and VCC = +5V. Output swing is guaranteed to meet the EIA/TIA-

232E and V.28 specification, which calls for ±5V minimum driver output levels under worst-case conditions.

These include a minimum 3k. load, VCC = +4.5V, and maximum operating temperature. Unloaded driver output voltage ranges from (V+ -1.3V) to (V- +0.5V).

Input thresholds are both TTL and CMOS compatible. The inputs of unused drivers can be left unconnected since 400k. input pull-up resistors to VCC are built in

(except for the MAX220). The pull-up resistors force the outputs of unused drivers low because all drivers invert.

The internal input pull-up resistors typically source 12µA, except in shutdown mode where the pull-ups are disabled. Driver outputs turn off and enter a high-impedance

state—where leakage current is typically microamperes (maximum 25µA)—when in shutdown mode, in three-state mode, or when device power is removed. Outputs can be driven to ±15V. The power supply current typically drops to 8µA in shutdown mode.

The MAX220 does not have pull-up resistors to force the outputs of the unused drivers low. Connect unused inputs to GND or VCC.

The MAX239 has a receiver three-state control line, and the MAX223, MAX225, MAX235, MAX236, MAX240, and MAX241 have both a receiver three-state control

line and a low-power shutdown control. Table 2 shows the effects of the shutdown control and receiver three state control on the receiver outputs.

The receiver TTL/CMOS outputs are in a high-impedance, three-state mode whenever the three-state enable line is high (for the MAX225/MAX235/MAX236/MAX239–

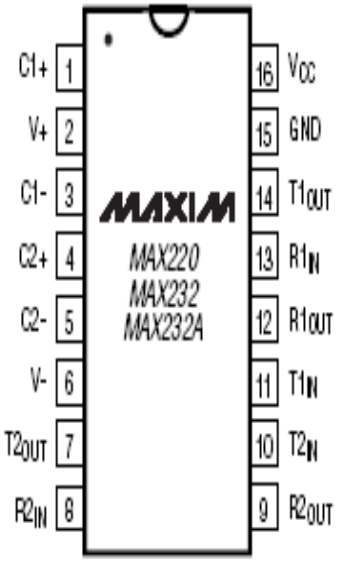
MAX241), and are also high-impedance whenever the shutdown control line is high.

When in low-power shutdown mode, the driver outputs are turned off and their leakage current is less than 1µA with the driver output pulled to ground. The driver output leakage remains less than 1µA, even if the transmitter output is back driven between 0V and (VCC + 6V). Below -0.5V, the transmitter is diode clamped to ground with 1k. series impedance. The transmitter is also zener clamped to approximately VCC + 6V, with a series impedance of 1k.. The driver output slew rate is limited to less than 30V/µs as required by the EIA/TIA-232E and V.28 specifications. Typical slew rates are 24V/µs unloaded and 10V/µs loaded with 3. and 2500pF.

RS-232 Receivers

EIA/TIA-232E and V.28 specifications define a voltage level greater than 3V as a logic 0, so all receivers invert. Input thresholds are set at 0.8V and 2.4V, so receivers

respond to TTL level inputs as well as EIA/TIA-232E and V.28 levels. The receiver inputs withstand an input over voltage up to ±25V and provide input terminating resistors with nominal 5k. values. The receivers implement Type 1 interpretation of the fault conditions of V.28 and EIA/TIA-232E. The receiver input hysteresis is typically 0.5V with a guaranteed minimum of 0.2V. This produces clear output transitions with slow-moving input signals, even with moderate amounts of noise and ringing. The receiver propagation delay is typically 600ns and is independent of input swing direction.



DIP/SO

CAPACITANCE (µF)					
DEVICE	C1	C2	C3	C4	C5
MAX220	4.7	4.7	10	10	4.7
MAX232	1.0	1.0	1.0	1.0	1.0
MAX232A	0.1	0.1	0.1	0.1	0.1

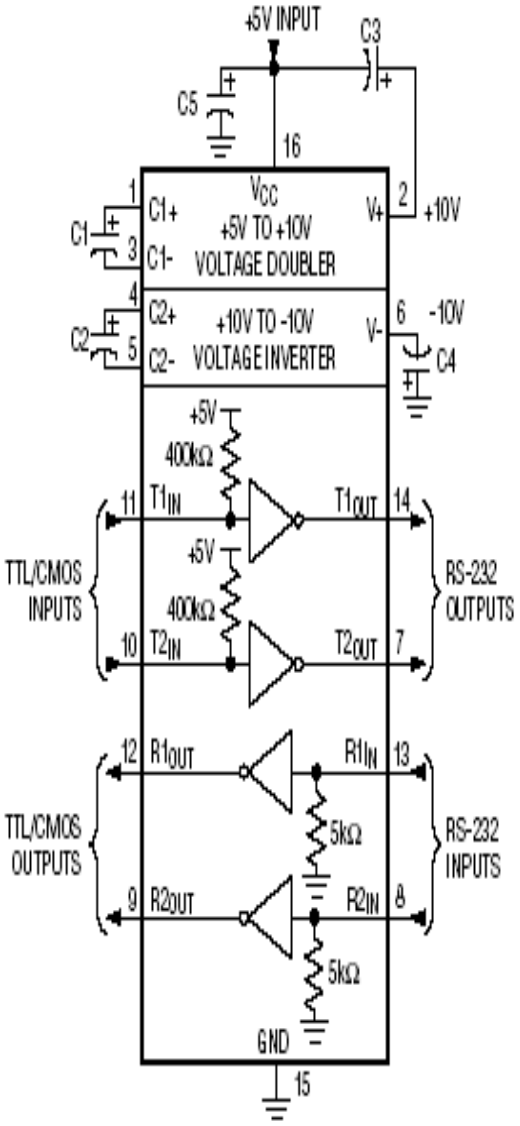


Fig A1 MAX 220/MAX 232/ Pin figuration and typical operating circuit .

APPENDIX B

Voltage Regulator LM 7805

General Description

The LM78XX series of three terminal positive regulators are available in the TO-220 package and with several fixed output voltages, making them useful in a wide range of applications.

Each type employs internal current limiting, thermal shut down and safe operating 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 voltages and currents.

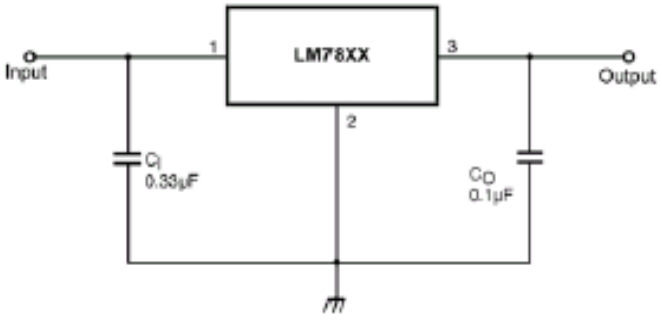


Fig B1 Application circuit

Features

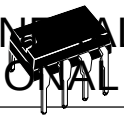
- Output Current up to 1A
- Output Voltages of 5, 6, 8, 9, 12, 15, 18, 24 Volts
- Thermal Overload Protection
- Short Circuit Protection
- Output Transistor Safe Operating Area Protection

UA 741 GENERAL PURPOSE OP AMP DATA SHEET

UA741

GENERAL PURPOSE SINGLE OPERATIONAL AMPLIFIER

- LARGE INPUT VOLTAGE RANGE
- NO LATCH-UP
- HIGH GAIN
- SHORT-CIRCUIT PROTECTION
- NO FREQUENCY COMPENSATION
- REQUIRED
- SAME PIN CONFIGURATION AS THE UA709



N
DIP8
(Plastic Package)

D
SO8
(Plastic Micropackage)

DESCRIPTION

The UA741 is a high performance monolithic operational amplifier constructed on a single silicon chip. It is intented for a wide range of analog applications.

- Summing amplifier
- Voltage follower
- Integrator
- Active filter
- Function generator

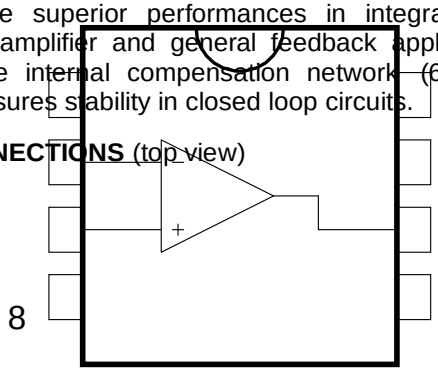
The high gain and wide range of operating voltages provide superior performances in integrator, summing amplifier and general feedback applications. The internal compensation network (6dB/ octave) insures stability in closed loop circuits.

ORDER CODE

Part Number	Temperature Range	Package	
		N	D
UA741C	0°C, +70°C	•	•
UA741I	-40°C, +105°C	•	•
UA741M	-55°C, +125°C	•	•
Example : UA741CN			

N = Dual in Line Package (DIP)
D = Small Outline Package (SO) - also available in Tape & Reel (DT)

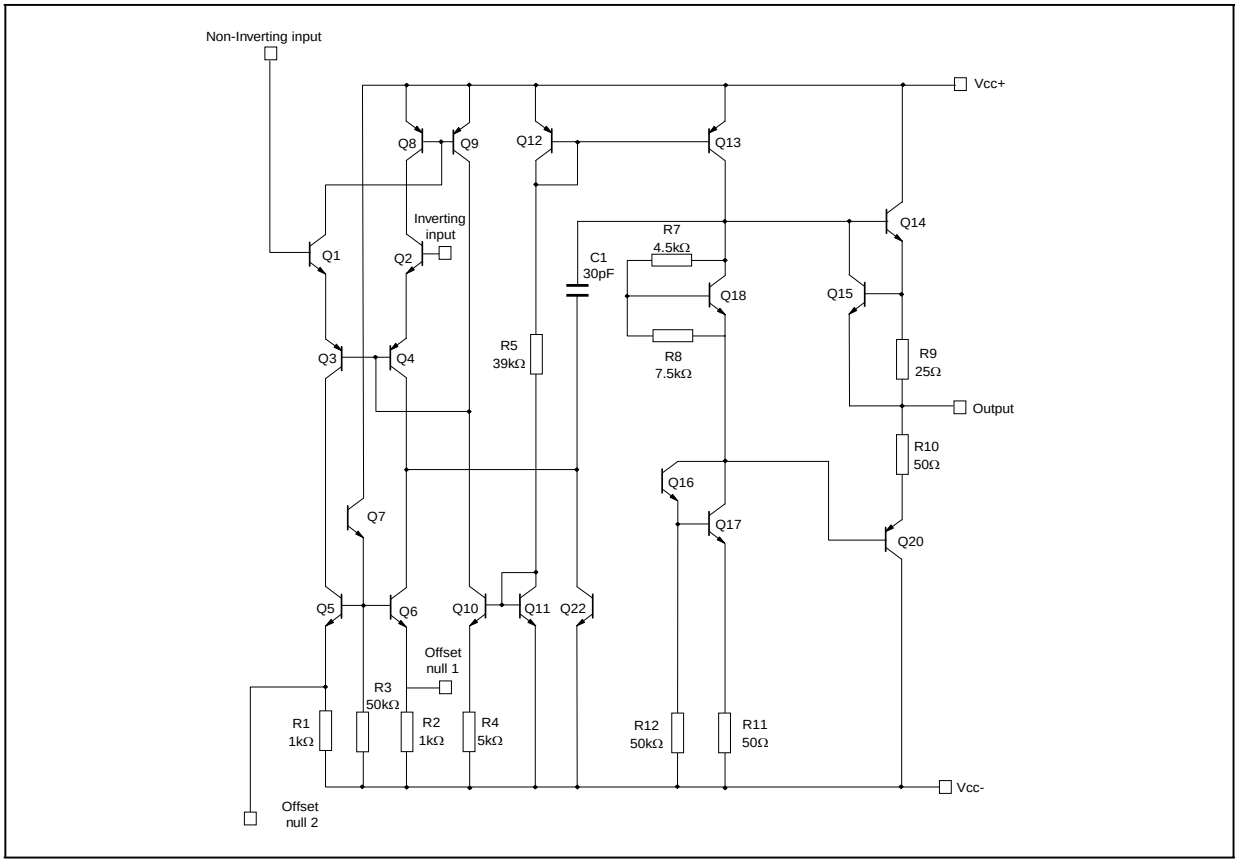
PIN CONNECTIONS (top view)



1 - Offset null 1
2 - Inverting input
3 - Non-inverting input
4 - V_{CC}^-
5 - Offset null 2
6 - Output
7 - V_{CC}^+

5

SCHEMATIC DIAGRAM



ABSOLUTE MAXIMUM RATINGS

Symbol	Parameter	UA741M	UA741I	UA741C	Unit
V_{CC}	Supply voltage	±22			V
V_{id}	Differential Input Voltage	±30			V
V_i	Input Voltage	±15			V
P_{tot}	Power Dissipation 1)	500			mW
	Output Short-circuit Duration	Infinite			
T_{oper}	Operating Free-air Temperature Range	-55 to +125	-40 to +105	0 to +70	°C
T_{stg}	Storage Temperature Range	-65 to +150			°C

1. Power dissipation must be considered to ensure maximum junction temperature (Tj) is not exceeded.

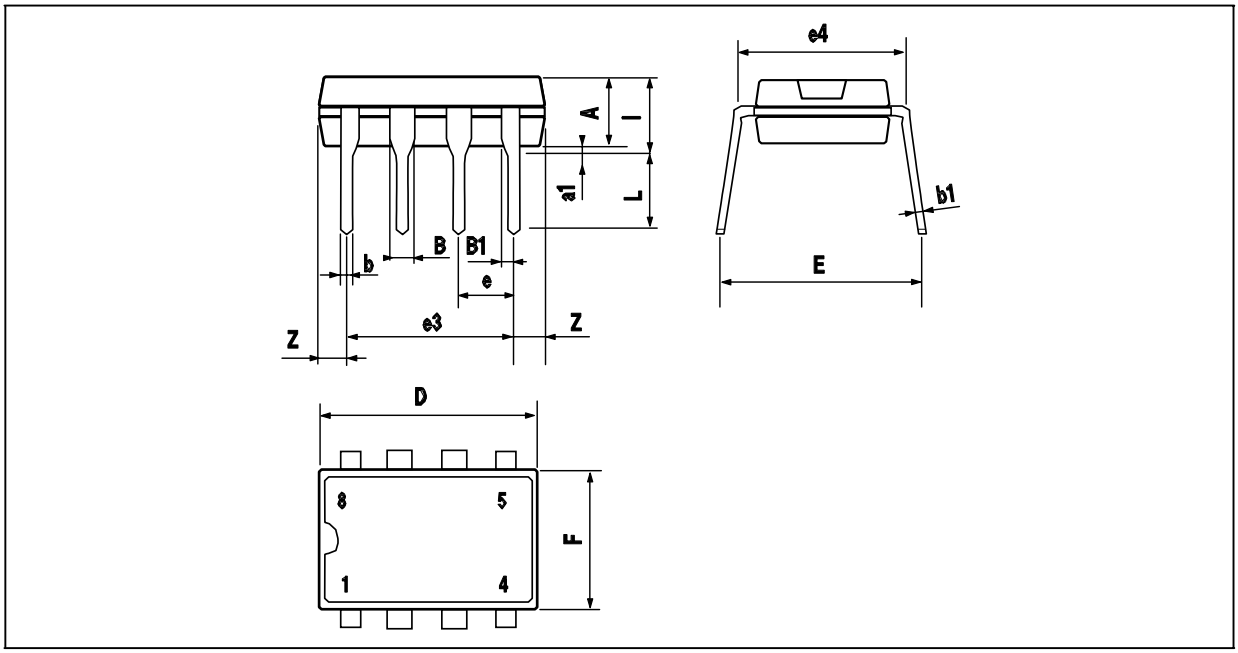
ELECTRICAL CHARACTERISTICS

V_{CC} = ±15V, T_{amb} = +25°C (unless otherwise specified)

Symbol	Parameter	Min.	Typ.	Max.	Unit
V_{io}	Input Offset Voltage ($R_S \le 10k\Omega$) T_{amb} = +25°C $T_{min} \le T_{amb} \le T_{max}$		1	5 6	mV
I_{io}	Input Offset Current T_{amb} = +25°C $T_{min} \le T_{amb} \le T_{max}$		2	30 70	nA
I_{ib}	Input Bias Current T_{amb} = +25°C $T_{min} \le T_{amb} \le T_{max}$		10	100 200	nA
A_{vd}	Large Signal Voltage Gain (V_o = ±10V, R_L = 2kΩ) T_{amb} = +25°C $T_{min} \le T_{amb} \le T_{max}$	50 25	200		V/mV
SVR	Supply Voltage Rejection Ratio ($R_S \le 10k\Omega$) T_{amb} = +25°C $T_{min} \le T_{amb} \le T_{max}$	77 77	90		dB
I_{CC}	Supply Current, no load T_{amb} = +25°C $T_{min} \le T_{amb} \le T_{max}$		1.7	2.8 3.3	mA
V_{icm}	Input Common Mode Voltage Range T_{amb} = +25°C $T_{min} \le T_{amb} \le T_{max}$	±12 ±12			V
CMR	Common Mode Rejection Ratio ($R_S \le 10k\Omega$) T_{amb} = +25°C $T_{min} \le T_{amb} \le T_{max}$	70 70	90		dB
I_{OS}	Output short Circuit Current	10	25	40	mA
$\pm V_{opp}$	Output Voltage Swing T_{amb} = +25°C, R_L = 10kΩ T_{amb} = +25°C, R_L = 2kΩ $T_{min} \le T_{amb} \le T_{max}$, R_L = 10kΩ $T_{min} \le T_{amb} \le T_{max}$, R_L = 2kΩ	12 10 12 10	14 13		V
SR	Slew Rate V_i = ±10V, R_L = 2kΩ, C_L = 100pF, unity Gain	0.25	0.5		V/µs
t_r	Rise Time V_i = ±20mV, R_L = 2kΩ, C_L = 100pF, unity Gain		0.3		µs
K_{ov}	Overshoot V_i = 20mV, R_L = 2kΩ, C_L = 100pF, unity Gain		5		%
R_i	Input Resistance	0.3	2		MΩ
GBP	Gain Bandwith Product V_i = 10mV, R_L = 2kΩ, C_L = 100pF, f =100kHz	0.7	1		MHz
THD	Total Harmonic Distortion f = 1kHz, A_v = 20dB, R_L = 2kΩ, V_o = $2V_{pp}$, C_L = 100pF, T_{amb} = +25°C		0.06		%
e_n	Equivalent Input Noise Voltage f = 1kHz, R_s = 100Ω		23		$\frac{nV}{\sqrt{Hz}}$
∅m	Phase Margin		50		Degrees

PACKAGE MECHANICAL DATA

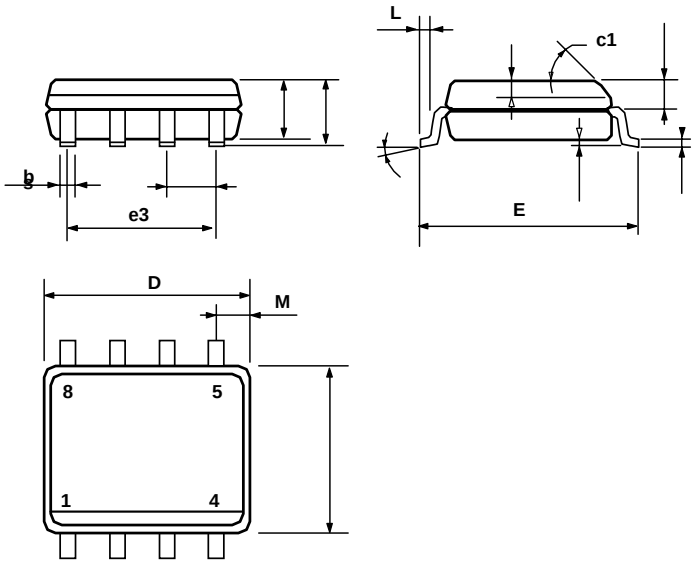
8 PINS - PLASTIC DIP



Dim.	Millimeters			Inches		
	Min.	Typ.	Max.	Min.	Typ.	Max.
A		3.32			0.131	
a1	0.51			0.020		
B	1.15		1.65	0.045		0.065
b	0.356		0.55	0.014		0.022
b1	0.204		0.304	0.008		0.012
D			10.92			0.430
E	7.95		9.75	0.313		0.384
e		2.54			0.100	
e3		7.62			0.300	
e4		7.62			0.300	
F			6.6			0260
i			5.08			0.200
L	3.18		3.81	0.125		0.150
Z			1.52			0.060

PACKAGE MECHANICAL DATA

8 PINS - PLASTIC MICROPACKAGE (SO)



Dim.	Millimeters			Inches		
	Min.	Typ.	Max.	Min.	Typ.	Max.
A			1.75			0.069
a1	0.1		0.25	0.004		0.010
a2			1.65			0.065
a3	0.65		0.85	0.026		0.033
b	0.35		0.48	0.014		0.019
b1	0.19		0.25	0.007		0.010
C	0.25		0.5	0.010		0.020
c1	45° (typ.)					
D	4.8		5.0	0.189		0.197
E	5.8		6.2	0.228		0.244
e		1.27			0.050	
e3		3.81			0.150	
F	3.8		4.0	0.150		0.157
L	0.4		1.27	0.016		0.050
M			0.6			0.024
S	8° (max.)					

HEX INVERTER IC SN 74LS04 DATA SHEET

APPENDIX D

Code for base station computer

```
Imports System.IO.Ports
Imports System.Threading

Public Class Form1
    Private serialObject As New SerialPort
    Private thread As Thread
    Dim fileStream As System.IO.FileStream
    Dim writer As System.IO.StreamWriter
    Dim reader As System.IO.StreamReader
    Dim dataRead As Boolean
    Dim readingThread As System.Threading.Thread
    Dim buffer As String
    Dim ok As Boolean
    Dim charArray(0 To 25) As Char
    Public Declare Sub Out Lib "inpout32.dll" Alias "Out32" (ByVal
PortAddress As Short, ByVal Value As Short)
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

    End Sub

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click

        If Button1.Text.Equals("Start") Then

            serialObject.BaudRate = 2400
            serialObject.DataBits = 8
```

```
            serialObject.Parity = Parity.None
            serialObject.PortName = "COM1"
            serialObject.StopBits = StopBits.One
            PictureBox2.Image=
Bitmap.FromFile("C:\\CarPictures\\StartingPic.bmp")
            serialObject.DiscardNull = True
            serialObject.ReadBufferSize = 200
            serialObject.ReadTimeout = 3000
            Out(&H378S, 240)
            serialObject.Open()
            Try
                serialObject.ReadChar()
            Catch ex As Exception

            End Try
            Timer1.Enabled = True
            Timer2.Enabled = True
            ok = True
            Timer2_Tick(vbNull, e)

            Button1.Text = "Stop"
        Else
            serialObject.Close()
            Timer1.Enabled = False
            Timer2.Enabled = False
            Button1.Text = "Start"
        End If


    End Sub

    Private Function process()
        Dim stream As System.IO.FileStream
        Dim reader As System.IO.StreamReader
        Dim data As String
        Timer2.Enabled = False

        If (buffer.StartsWith("&")) Then
```

```
            data = buffer.Substring(1, buffer.Length - 1)
            If (System.IO.File.Exists("C:\\CarsInRange\\" & data &
".txt")) Then
                Try
                    Try
                        System.IO.File.Copy("C:\\CarsInRange\\" & data
& ".txt", "C:\\PassingCarsData\\" & data & ".txt", True)
                        System.IO.File.Delete("C:\\CarsInRange\\" &
data & ".txt")
                    Catch ex As Exception
                        MsgBox("Could not move" & ex.Message.ToString)
                    End Try
                    updateInRangeCars()
                    currentlyPassing_txt.Text = data
                    PictureBox1.Image =
Bitmap.FromFile("C:\\CarPictures\\" & data & ".jpg")
                    Out(&H378S, 15)
                Catch ex As Exception
                    MsgBox(ex.Message.ToString)
                End Try

            End If



        ElseIf (buffer.StartsWith("*")) Then
            Dim regNo As String
            Dim tempRead As String
            Dim i As Integer
            data = buffer.Substring(1, buffer.Length - 1)
            regNo = data.Substring(0, data.IndexOf(","))
            stream = New System.IO.FileStream("C:\\AllCarsData\\" &
regNo & ".txt", IO.FileMode.Open)
            reader = New System.IO.StreamReader(stream)
            tempRead = reader.ReadLine()
            If tempRead.ToLower().Equals(data.ToLower()) Then
```

```
                If System.IO.File.Exists("C:\\AllCarsData\\" & regNo &
".txt") Then
                    System.IO.File.Copy("C:\\AllCarsData\\" & regNo &
".txt", "C:\\CarsInRange\\" & regNo & ".txt", True)
                    tempRead = "@OK#" & regNo & "$"
                    charArray = tempRead.ToCharArray()
                    i = 0
                    System.Threading.Thread.CurrentThread.Sleep(100)
                    While Not charArray(i) = "$"
                        serialObject.Write(charArray, i, 1)
                        i = i + 1
                    End While
                    serialObject.Write(charArray, i, 1)
                End If
            End If

            reader.Close()
            stream.Close()
            updateInRangeCars()


        End If

        Timer2.Enabled = True

    End Function
    Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Timer1.Tick
        ok = False
        Dim files As String()
        Dim i As Integer
        files = System.IO.Directory.GetFiles("C:\\CarsInRange")
        For i = 0 To files.Length - 1
            If (System.DateTime.Now.TimeOfDay.TotalMinutes -
System.IO.File.GetCreationTime(files(i)).TimeOfDay.TotalMinutes) > 4
Then
                System.IO.File.Delete(files(i))
                updateInRangeCars()
```

```
            End If
        Next
        ok = True
    End Sub

    Private Sub Timer2_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Timer2.Tick
        If (ok) Then
            serialObject.Write("%")
            Out(&H378S, 240)
            Try
                buffer = serialObject.ReadTo(";")
                process()
            Catch ex As Exception
            End Try
        End If
    End Sub

    Private Function updateInRangeCars()
        Dim files As String()
        Dim i As Integer
        files = System.IO.Directory.GetFiles("C:\CarsInRange")
        carsInRange_txt.Clear()
        For i = 0 To files.Length - 1

carsInRange_txt.AppendText(files(i).Substring(files(i).LastIndexOf("\")
+ 1, files(i).LastIndexOf(".") - files(i).LastIndexOf("\")) & vbCrLf)
            Application.DoEvents()
        Next
    End Function


End Class
```

Code for the microcontroller installed in car module

```
/*---------------------------------------------------------------------------
Wireless Registration System for Vehicles; The Car module.
---------------------------------------------------------------------------*/

#include <atmel/at89x52.h>            /* special function register declarations   */
                                      /* for the intended 8051 derivative         */

#include <stdio.h>                    /* prototype declarations for I/O functions */


/*-----------------------------------------------
The main C function.  Program execution starts
here.
-----------------------------------------------*/
 char A [12];              /* A character array that is used to receive the authentication msg */

 char REG [12]="@OK#RLD 674$";          /* A character array which contains
                                           the required authentication msg */

 sbit P10=P1^0;

 char atr;

 char check1;                    /* A character that is used to answer the interrogation of base */

 long int c;

 long int i;                     /* Variable for the FOR LOOP */

 unsigned int j;                 /* Variable for the FOR LOOP */
```

```
  unsigned int x;

  unsigned int y;

  int flag;                        /* Variable that ensures the reception of authentication */

  int port;

  void main (void)
{

/*-------------------------------------------------

-------------------------------------------------*/

  SCON  = 0x50;                    /* mode 1: 8-bit UART, enable receiver   */
  TMOD |= 0x20;                    /* timer 1 mode 2: 8-Bit reload          */
  TH1   = 0xf4;                   /* reload value 2400 baud                */
  TR1   = 1;                      /* timer 1 run                           */
  TI    = 1;                     /* TI:   set TI to send first char of UART    */



/*-------------------------------------------------
Note that an embedded program never exits (because
there is no operating system to return to).  It
must loop and execute forever.
-------------------------------------------------*/

P0=0xFF;
P1=0xFF;
P2=0xFF;
P3=0xFF;

while(1)
  {
```

```
port=0;
flag=0;
while(flag==0)
 {
 P1=0xFF;
 while (1)
 {
    while (RI!=1);
           check1=SBUF;
    if (check1 == '%')
    {
    P1_1=0;                    /* Toggle P1.1 */
           break;
           }
           RI=0;
 }
   RI=0;
          printf ("*RLD 674,MEHRAN 2006,ALI JAVED,E171234,C432567;");                    /*
Sending the car data */
    P1_2=0;                    /* Toggle P1.2 */


    for (c=0;c<=76800;c++)
          {
    atr=SBUF;
           if (atr == '@')
           {
                  while (SBUF != '$')


             {for (i=0; i<=11; i++)
        {
        while (RI!=1);
                        A[i]=SBUF;
        RI=0;
                  }
```

```
    }

  for (i=0; i<=11; i++)
        {if (A[i] == REG[i])
         {if (i==11)
                {c=76800;
       flag=1;
                                P1_3=0;                 /* Toggle P1.3 */
                                }

    }
        else
          {c=76800;
      break;}
        }
       }
       }
}
       for (i=1; i<=360000; i++)
         {
              for (j=1; j<=60; j++)
               {
                if ( P10==0)
                        {printf("&RLD 674;");
                         P1_4=0;                /* Toggle P1.4 */
                                port=1;
                                break;}
               }

             if (port == 1)
             {
               for (x=1; x<=60000; x++)
                {
             for (y=1; y<=120; y++)
                        {}
                    }
               break;
```

```
                    }


                }

    }
```

Glossary

EM	Electro Magnetic
RF	Radio Frequency
FM	Frequency Modulation
UART	Universal Asynchronous Receiver Transmitter
EIA	Electronics Industries Associations
TIA	Telecommunication Industries Associations
PCI	Peripheral Control Interface
RS	Recommended Standard
DTE	Data Terminal Equipment
DCE	Data Communication Equipment
RTS	Request To Send
CTS	Cleared To Send
FIFO	First In First Out
IR	Infra red
TTL	Transistor - Transistor Logic
DC	Direct Current

REFERENCES

Websites

1. http://ece-www.clorado.edu/~ecen4618/lab4.pdf
2. http://www.maxstream.net/
3. http://www. Linuxdocs.org/HOWTOs/Serail-HOWTO-3.html
4. http://www.see.ed.ac.uk/~gaa/DigilabNotes/Digilab/components/node13.html
5. http://www.rs485.com/rs485spec.html
6. http://www.interq.or.jp/japan/se-inoue/e_ic.html
7. http://kele.com/Tech/Signal/SerialPt.html
8. http://searchnetworking.techtarget.com/sDefination.html
9. http://www.cnes.fr.cnesdeu/sommaire/passion/campagne/national/outils/xr2211.pdf
10. http://www.jaycar.com.au/images_uploaded/R2206V1.PDF
11. http://www.geo.mtu.edu/rs/back/spectrum.html
12. http://www.purchon.com/physics/electormagnatic.html
13. http://imager.gsfc.nasa.gov/ems/waves2.html
14. http://www.lbl.gov/MicroWorlds/ALSTool/EMSpec/EMspec2.html
15. http://.users.muohio.edu/grimlejm/physics_of_Wireless.html
16. http://www.casde.unl.edu/vn/tutorial.emag.lgspec2.html

BOOKS

1. R.M.Marston. 110 integrated circuits projects for home constructor. Printed in England by Billings & Sons limited, Guildford, London and Worcester, pp71-85.

2. Lloyd Temes and Mitchel E. Schultz. Electronic communication, 2nd edition McGraw Hill, pp 52-55.

3. Scott Meckenzie 8051 micro controller.

4. Mazidi 8051 micro controller.