
# COLLEGE MONITORING SYSTEM USING SMART CARD

P-1999

**A PROJECT REPORT**

*Submitted by*


| | |
|---|---|
| **G.MAHEEJA** | **71203106029** |
| **S.MIRUNYA** | **71203106030** |
| **G.P. NANDHINI** | **71203106031** |
| **S.JANANI** | **71203106303** |


*In partial fulfillment for the award of the degree*

*of*

**BACHELOR OF ENGINEERING**

**IN**

**ELECTRONICS AND COMMUNICATION ENGINEERING**


**KUMARAGURU COLLEGE OF TECHNOLOGY ,**

**COIMBATORE-641006**



**ANNA UNIVERSITY :CHENNAI 600 025**

**APRIL 2007**

**ANNA UNIVERSITY : CHENNAI 600 025**

## BONAFIDE CERTIFICATE

Certified that this project report **"COLLEGE MONITORING SYSTEM USING SMART CARD"** is the bonafide work of **"G.MAHEEJA, S.MIRUNYA, G.P.NANDHINI ,S.JANANI"** carried out the project work under my supervision.

**SIGNATURE**

Dr.RAJESWARI MARIAPPAN

**HEAD OF THE DEPARTMENT**

Electronics and
Communication Engineeering
Kumaraguru college of Technology
Coimbatore-641006.

**SIGNATURE** 16/4/07

Mrs.M.SHANTHI

**SUPERVISOR**

ASSISTANT PROFESSOR

Electronics and
Communication Engineering
Kumaraguru college of Technology
Coimbatore-641006

The candidates with university register numbers **71203106029, 71203106030, 71203106031,71203106303** was examined by us in the project Viva Voce examination held on 23.04.2007

Internal Examiner

External Examiner

(An ISO 9001 : 2000 Certified Company)
Regd. Office & Factory : 25, Functional Electronic Estate,
Civil Aerodrome Post, Coimbatore - 641 014. Tamilnadu. India
Phone : (91) 422 - 262 7795, 262 7018 Fax : (91) 422 - 262 7659
Email : agtgold@vsnl.com Home Page : www.agtindia.com

# CERTIFICATE

## TO WHOMSOEVER IT MAY CONCERN

This is to certify that

| | |
|---|---|
| **1. Ms. G. MAHEEJA** | **(71203106029)** |
| **2. Ms. S. MIRUNYA** | **(71203106030)** |
| **3. Ms. G.P. NANDHINI** | **(71203106031)** |
| **4. Ms. S. JANANI** | **(71203106303)** |

doing final year **B.E (ELECTRONICS & COMMUNICATION ENGINEERING)** in **Kumaraguru College of Technology, Coimbatore** have done a project under the title **" College Monitoring System Using Smart Card "** in our organization from **03.01.2007** to **11.04.2007.**

They completed the project successfully and their Conduct during this period was excellent.

We wish them the very best for a bright future.

For AGT ELECTRONICS LTD.

AUTHORISED SIGNATORY

PROJECT GUIDE

Date: 11.04.2007

AGT ELECTRONICS LTD.
25, Functional Electronics Estate,
Civil Aerodrome Post,
Coimbatore - 641 014.
www.agtgold.com, agtgold@vsnl.com.

## ACKNOWLEDGEMENT

We take this opportunity to express our profound thanks to our guide **Mrs. M.SHANTHI,ASSISTANT PROFESSOR,** Department of Electronics and Communication Engineering, Kumaraguru College of Technology, Coimbatore, for her valuable guidance and support for the successful completion of the project work.

We wish to record our gratitude to **Mr. K.RAMPRAKASH, PROFESSOR, THE PROJECT COORDINATOR,**Department of Electronics and Communication Engineering, Kumaraguru College of Technology , Coimbatore, for his extended support in the completion of the project.

We take this opportunity to thank, **Dr. RAJESWARI MARIAPPAN, HEAD OF THE DEPARTMENT,** Electronics and Communication Engineering, Kumaraguru College of Technology, Coimbatore, for her extended support in completion of the project.

We acknowledge our thanks to **Dr. JOSEPH.V.THANIKAL, PRINCIPAL,** Kumaraguru College of technology,Coimbatore, for providing an opportunity to take up this project.

Finally, We express our thanks to our parents and our friends for their encouragement in completion of this project work successfully.

# ABSTRACT

Smart card has a variety of applications . In this project , the attendance of staff and students is monitored using the smart card. Every staff and student is issued an individual smart card with a unique password .The card is inserted into the card reader in the classroom during the entry and exit . The password , date and time is stored into the database .The absence of any staff is identified and alternate arrangements are made using SMS facility.
Smart card is designed using I2C chip which is a serial EEPROM memory chip in which datas are written. Writing operation in the smart card is done by programming a Microcontroller 89C52.The password is written in a predefined memory location of the chip.

Card readers are designed for two class rooms and this can be extended to any number of rooms using a buffer.A laser pen is used to detect the number of people entering the room per card and a buzzer is used to indicate if more than one person enters per card.

## TABLE OF CONTENTS


| HAPTER NO. | TITLE | PAGE NO |
|---|---|---|
| | **ABSTRACT** | **v** |
| | **LIST OF TABLES** | **x** |
| | **LIST OF FIGURES** | **x** |
| | **LIST OF ABBREVIATIONS** | **xi** |
| **1** | **INTRODUCTION** | **1** |
| | 1.1 EMBEDDED SYSTEMS | 1 |
| | 1.2 SMART CARD | 2 |
| **2** | **SYSTEM DESIGN** | **3** |
| | 2.1 COMPONENTS USED | 3 |
| | 2.2 BASIC BLOCK DIAGRAM | 4 |
| | 2.3 WORKING | 4 |
| | 2.3.1 Write operation | 4 |
| | 2.3.2 Read operation | 5 |
| | 2.4 CIRCUIT DIAGRAM | 7 |
| **3** | **HARDWARE OVERVIEW** | **8** |
| | 3.1 FEATURES | 8 |
| | 3.2 PIN DESCRIPTION | 9 |
| | 3.3 SPECIAL FUNCTION REGISTERS | 14 |
| | 3.4 OSCILLATOR CHARACTERISTICS | 21 |
| | 3.5 POWER DOWN MODE | 22 |
| | 3.6 PROGRAM MEMORY LOCK BITS | 23 |
| | 3.7 PROGRAMMING THE FLASH | 24 |
| | 3.8 PROGRAMMING ALGORITHM | 25 |
| | 3.9 DATA POLLING | 25 |
| | 3.10 READY/BUSY | 26 |
| | 3.11 PROGRAM VERIFY | 26 |
| | 3.12 CHIP ERASE | 26 |
| | 3.13 READING THE SIGNATURE BYTES | 26 |
| | 3.14 PROGRAMMING INTERFACE | 27 |

| | | |
|---|---|---|
| **4** | **RS232** | **28** |
| | 4.1 RS232 STANDARDS | 28 |
| | 4.2 RS232 PINS | 28 |
| | 4.3 FEATURES OF RS232 | 29 |
| | 4.4 MAX232 | 30 |
| **5** | **SERIAL COMMUNICATION** | **32** |
| | 5.1 SERIAL DATA TRANSFER | 32 |
| | 5.2 ASYNCHRONOUS SERIAL COMMUNICATION | 33 |
| | 5.3 DATA TRANSFER RATE | 33 |
| | 5.4 RXD AND TXD PINS IN AT89C52 | 34 |
| **6** | **LCD DISPLAY** | **35** |
| | 6.1 LCD OPERATION | 35 |
| | 6.2 FUNCTIONAL DESCRIPTION OF LCD | 35 |
| | 6.3 INITIALISATION OF LCD | 36 |
| | 6.4 ADVANTAGES | 37 |
| | 6.5 DISADVANTAGES | 37 |
| **7** | **I2C** | **38** |
| | 7.1 DESIGN | 38 |
| | 7.2 APPLICATIONS | 38 |
| **8** | **SMART CARD** | **40** |
| | 8.1 FEATURES | 40 |
| | 8.2 DESCRIPTION | 40 |
| | 8.3 PIN DESCRIPTION | 42 |
| **9** | **DEVICE OPERATION** | **43** |
| | 9.1 DEVICE ADDRESSING | 45 |
| | 9.2 WRITE OPERATIONS | 46 |
| | 9.3 READ OPERATIONS | 47 |

| | | |
|---|---|---|
| 10 | **POWER SUPPLY** | **50** |
| | 10.1 CIRCUIT DESCRIPTION | 50 |
| | 10.2 TRANSFORMER SECTION | 50 |
| | 10.3 RECTIFIER SECTION | 50 |
| | 10.4 FILTER SECTION | 51 |
| | 10.5 VOLTAGE REGULATOR SECTION | 52 |
| 11 | **KEYPAD** | **54** |
| | 11.1 TYPES | 54 |
| | 11.2 KEYBOARD MICROCOMPUTER | 54 |
| | 11.3 KEY FUNCTION | 55 |
| | 11.4 KEYPAD INTERFACING | 55 |
| | 11.5 BIDIRECTIONAL COMMUNICATIONS | 56 |
| | 11.6 FEATURES | 57 |
| 12 | **KEIL C51 COMPILER BASICS** | **58** |
| | 12.1 8051 MEMORY CONFIGURATION | 58 |
| | 12.2 POSSIBLE MEMORY MODELS | 59 |
| | 12.3 STEP BY STEP PROCEDURE | 60 |
| | 12.4 DSCOPE DEBUGGER | 62 |
| | 12.5 LOADING A PROGRAM | 62 |
| 13 | **RESULTS** | **64** |
| | 13.1 VISUAL BASIC | 64 |
| | 13.2 INITIAL STAGE | 64 |
| | 13.3 PORT INITIALISATION | 65 |
| | 13.4 CELL PHONE CONNECTION | 66 |
| | 13.5 DATABASE UPDATION | 67 |
| | 13.6 INSTANT MESSAGING | 68 |
| | 13.7 DETECTION OF MISUSE | 69 |
| | 13.8 MOBILE DISCONNECTION | 70 |
| | 13.9 DATABASE | 71 |
| 14 | **CODING** | **72** |
| | 14.1 MAIN CODE | 72 |

14.2 DISPLAY CODE 89
14.3 KEYPAD CODE 93
14.4 I2C CODE 94

**15 CONCLUSION 105**

**16 APPENDIX 106**

## LIST OF TABLES

Table 1 Port1 pin alternate functions

Table 2 Port3 pin alternate functions

Table 3 Timer 2 operating modes

Table 4 Interrupt Enable Register

Table 5 Status of external pins during idle and power down modes

Table 6 Lock bit protection modes

Table 7 Top side marking and device signature codes

Table 8 Flash programming modes

Table 9 Pin description for LCD

## LIST OF FIGURES

Figure 2.1 Block diagram

Figure 2.2 Write operation

Figure 2.3 Final circuit diagram

Figure 3.1 Pin diagram

Figure 3.2 AT89C52 Architecture

Figure 3.3 Programming and verifying flash memory

Figure 4.1 RS232 pin layout

Figure 4.2 MAX232 interfacing

Figure 5.1 Communication model

Figure 5.2 Asynchronous serial communication

Figure 5.3 Processor interfacing

Figure 8.1 AT24C64 pin diagram

Figure 8.2 AT24C64 block diagram

Figure 9.1 Timing diagrams
Figure 9.2 Read operation
Figure 10.1 Power supply circuit diagram
Figure 10.2 Rectifier circuit diagram
Figure 11.1 Keypad design module
Figure 11.2 Keypad interfacing with Microcontroller

## LIST OF ABBREVIATIONS

| | |
|---|---|
| I2C | -Inter Integrated Circuit |
| EEPROM | -Electrically Erasable Programmable Read Only Memory |
| LCD | -Liquid Crystal Display |
| LED | -Light Emitting Diode |
| RAM | -Random Access Memory |
| TTL | -Transistor Transistor Logic |
| SCL | -Serial Clock |
| SDA | -Serial Data |
| TXD | -Transmit Data |
| RXD | -Receive Data |

# 1. INTRODUCTION

## 1.1 EMBEDDED SYSTEMS

The embedded system is a dedicated system which is a combination of hardware and software. As the size of the processor goes smaller and cheaper more products have the processor embedded in the system .It basically has a CPU, memory and control register. The memory is the storage device. But this processor cannot function independently because of the simple reason they do not have enough memory and do not have direct control the external device. The external devices that provide support to the processor are DMA channel, ROM, RAM, but etc, to make the processor function. This complete set appears to be very big though they can perform vast tasks.

Then there comes the idea of using system that can do some dedicated tasks at the same speed. In the earlier case reliability, speed and durability was a question. All these were given a single solution with the advent of embedded system. It has a processor, long with DMA channel, ROM, RAM, interrupt, bus control all these were found on a single chip termed microcontroller. So, it proved faster, reliable and efficiency as very high. In processor based system the program where loaded into the hard disk and the processor fetch the instruction whenever needed.

The program was loaded into the memory of the controller chip needed to do a dedicated task. Then looking into the microcontroller ATMEL 89C52 that is used, it belongs to the 8051 and 8052 family. The

advantages over these were they have three timers and flash memory which can be programmed quite a number of times. The combination of software and hardware forming the embedded system prove devastating to the bio-medical instrumentation and security purpose in medical industry. They are portable and can be installed in security purpose.

## CHARACTERISTICS OF EMBEDDED SYSTEMS

1. Sophisticated functionality
2. Real time operation
3. Low manufacturing cost
4. Low power
5. Designed to tight deadlines by small teams

## 1.2 SMART CARD

The widespread use of smart card applications is now running at an all time high. The smart card technology has multifunction flexibility and all functions within one card. Nowadays smart cards are used for the following purposes.

a. Office administration
b. Business
c. Predicting the world market
d. Banking

It can also be used as a remote monitor. Our project is to demonstrate how a smart card can be used to monitor the attendance of the

staffs and students, an instant mailing to the department HOD and seating arrangements during the examinations. In our project the smart cards are used as IDs. Some identification details are stored in this smart card. When the card is scratched, the details in the card are transferred to the microcontroller through card reader. All the details of the users are already stored in the microcontroller. With reference to the smart card details it will retrieve further details about the user.. Then the microcontroller transfers the details of the user to the PC along with the time.

# 2.SYSTEM DESIGN

## 2.1 COMPONENTS USED

The components used in our project are given below

- Micro controller - 89C52
- I2C serial EEPROM - AT 2464
- Interface - MAX 232
- LCD
- KEYPAD
- PC
- BUZZER

## 2.2 BASIC BLOCK DIAGRAM

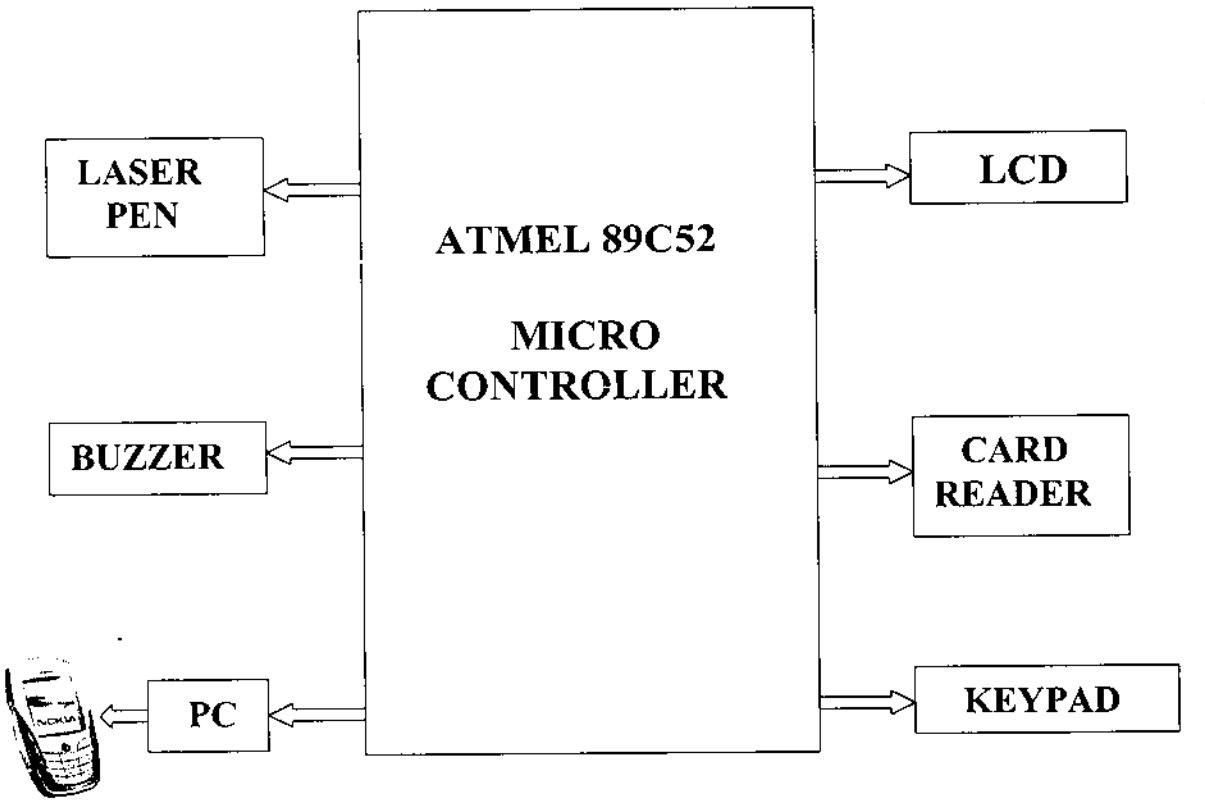



Figure 2.1 BLOCK DIAGRAM

## 2.3 WORKING

The above block diagram represents both for read and write operation.

### 2.3.1 WRITE OPERATION

In write operation the microcontroller 8952 is programmed in such a way that the datas are written in a particular memory location in AT2464.Hence for write operation microcontroller and EEPROM memory components are only used.

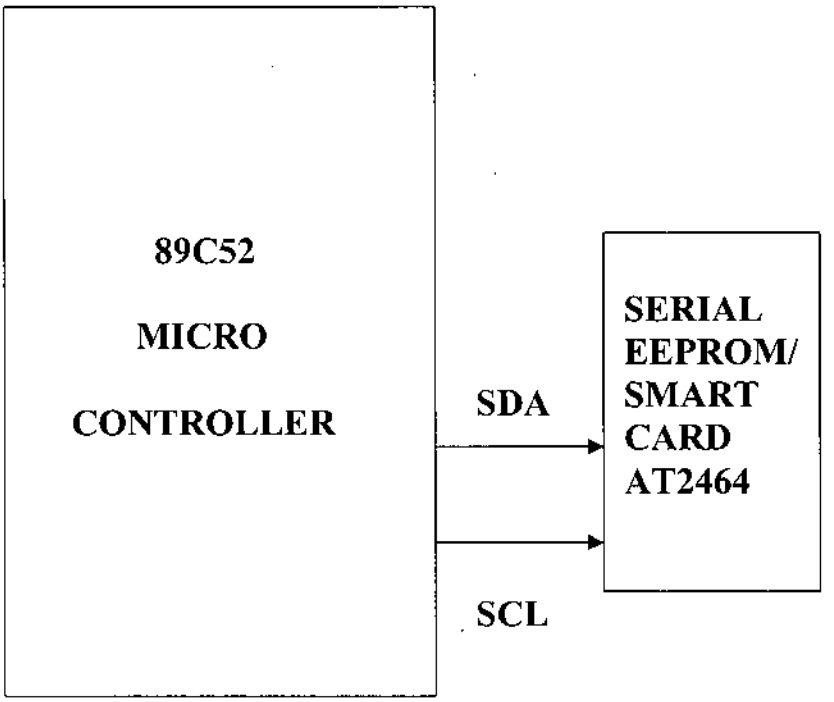



Figure 2.2 write operation

The data from the microcontroller 89C52 are transmitted serially bit by bit through SDA when clock pulse pin is at high state. The number of bytes to be written is given and loop is given which executes write operation for the number specified.

## 2.3.2 READ OPERATION

The read operation consists of all components as in the basic block diagram.

The Microcontroller is programmed is sush a way that it fetches data from the given location. Here the password is stored in the EEPROM. When the user inserts the card the microcontroller displays the instruction to enter the password through keypad the value entered are stored in an array (in

89C52). The value in the array is matched with the one stored in the EEPROM . when the password entered is correct then microcontroller is programmed to display "PASSWORD IS INCORRECT".

When the password entered is right then controller dispays "PASSWORD CORRECT" and enters into the operation of fetching the information. The SDA pin in the EEPROM in the EEPROM is bidirectional, the information from the EEPROM is is taken by sensing the SDA pin. The address byte and the number of bytes written is given , the controller sends the appropriate operation format and the address byte. Then by sensing the bits from the EEPROM the datas are stored as bytes in microcontroller array and then displayed through LCD or PC.

## 2.4 **CIRCUIT DIAGRAM**

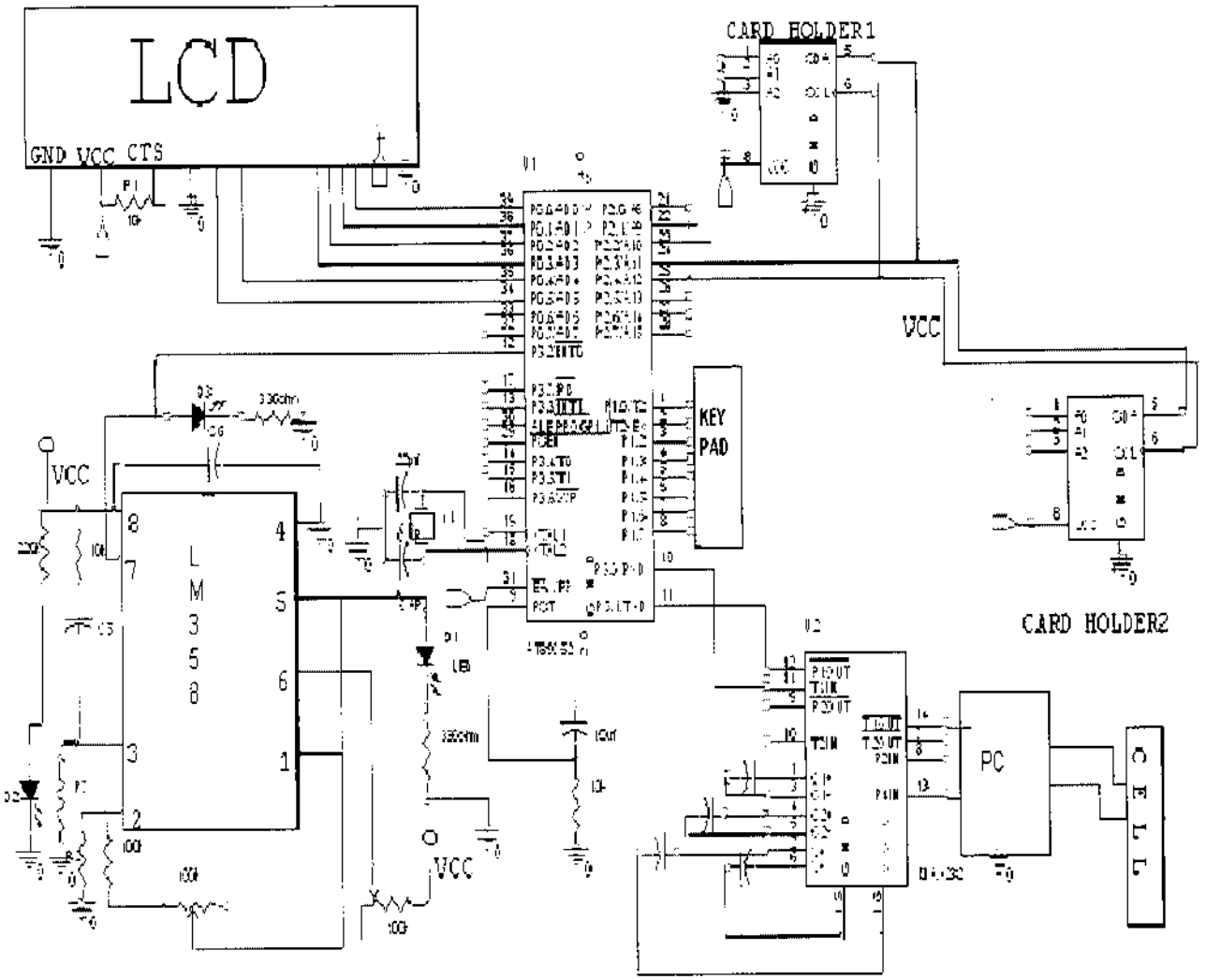



Figure 2.3 final circuit diagram

# 3. HARDWARE OVERVIEW

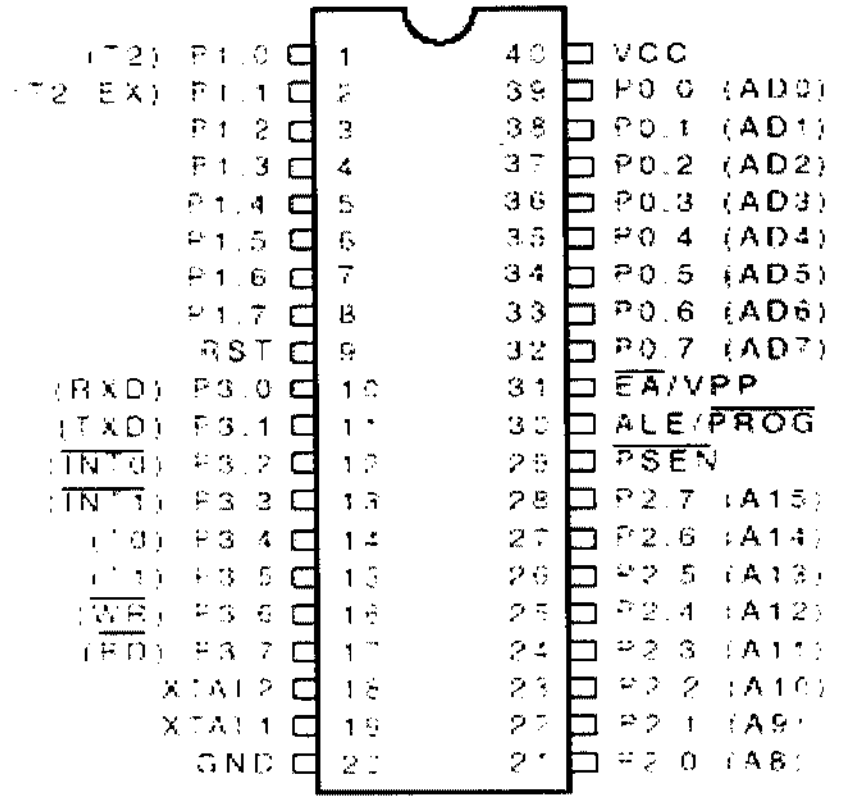
# AT89C52 MICROCONTROLLER

The AT89C52 is a 24 MHz CMOS controller belongs to 8051 families. The basic idea of using this IC in spite other controllers is it suffices our need. Each operation takes only 1 microsecond, which is sufficient for application that we have dealt. It has 8K bytes of flash memory along with 128 bytes on chip RAM. It has two pins XTAL1 and XTAL2 input and output. Either a quartz or ceramic resonator is used. To drive the device from an external clock source, XTAL2 should be left unconnected while XTAL1 is driven. An oscillator of frequency 11.0592 MHz is used. The controller has 4 ports, 32 input output lines and three timers/counters. It has reset pin 9. A high on this pin for two machine cycles while the oscillator is running resets the device.

## 3.1 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

**Figure 3.1 - PIN DIAGRAM**




## 3.2 PIN DESCRIPTION

**VCC** - Supply voltage.

**GND** - Ground.

**PORT 0 (P 0.0 – P 0.7)**

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 can 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.

**PORT 1 (P 1.0 – P 1.7)**

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.

**Table 1**

| **Port Pin Alternate Functions** |
|---|
| P1.0 - T2 (external count input to Timer/Counter 2), clock-out |
| P1.1 - T2EX (Timer/Counter 2 capture/reload trigger and direction control) |

**PORT 2 (P 2.0 – P 2.7)**

Port 2 is an 8-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, Port 2 uses strong internal pull-ups when emitting 1s. During accesses to external data memory that uses 8-bit addresses (MOVX @ RI), 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.

**PORT 3 (P 3.0 – P 3.7)**

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 functions of various special features of the AT89C51, as shown in the following table. Port 3 also receives some control signals for Flash programming and verification.

**Table 2**

| **Port Pin Alternate Functions** |
|---|
| 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) |



**RST**

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

**ALE/PROG**

Address Latch Enable is an 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 microcontroller is in external execution mode.

**PSEN**

Program Store Enable is the read strobe to external program memory. When the AT89C52 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.

**EA/VPP**

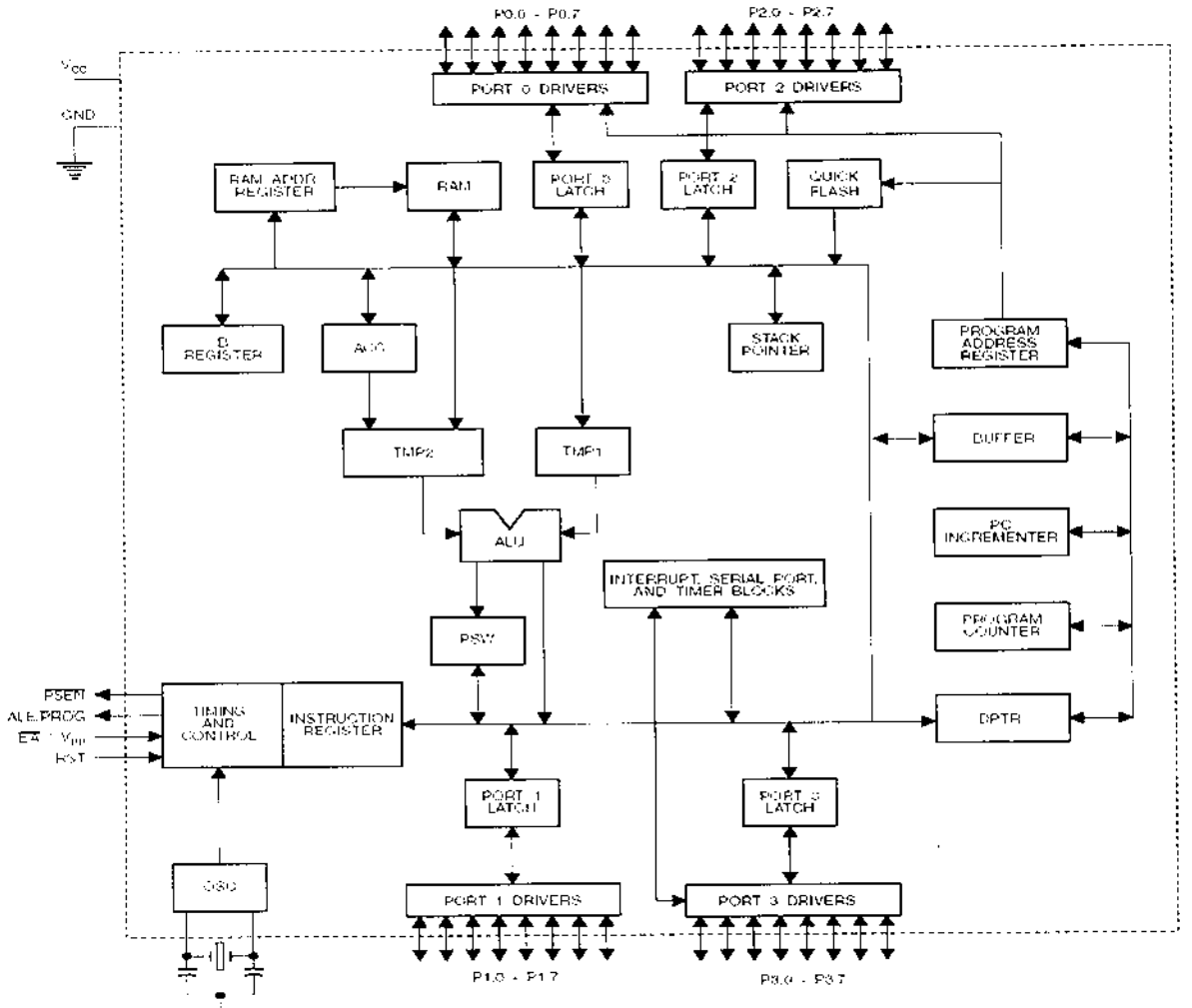
External Access Enable. EA must be strapped to GND in order to enable the device to fetch code from external program memory locations starting at 0000H up to FFFFH. Note, however, that if lock bit 1 is programmed, EA will be internally latched on reset. EA should be strapped to VCC for internal program executions. This pin also receives the 12-volt programming enable voltage (VPP) during Flash programming when 12-volt programming is selected.

**XTAL1 :** Input to the inverting oscillator amplifier and input to the internal clock operating circuit.

**XTAL2 :** Output from the inverting oscillator amplifier.

## AT89C52 ARCHITECTURE

Figure 3.2



## 3.3 SPECIAL FUNCTION REGISTERS

A map of the on-chip memory area called the Special Function Register (SFR). Note that not all of the addresses are occupied, and unoccupied addresses may not be implemented on the chip. Read accesses to these addresses will in general return random data, and write accesses will have an indeterminate effect. User software should not write 1s to these unlisted locations, since they may be used in future products to invoke new features. In that case, the reset or inactive values of the new bits will always be 0.

### TIMER 2 REGISTERS

Control and status bits are contained in registers T2CON and T2MOD for Timer 2. The register pair (RCAP2H, RCAP2L) is the Capture/Reload registers for Timer 2 in 16-bit capture mode or 16-bit auto-reload mode.

### INTERRUPT REGISTERS

The individual interrupt enable bits are in the IE register. Two priorities can be set for each of the six interrupt sources in the IP register.

### DATA MEMORY

The AT89C52 implements 256 bytes of on-chip RAM. The upper 128 bytes occupy a parallel address space to the Special Function Registers. That means the upper 128 bytes have the same addresses as the SFR space but are physically separate from SFR space. When an instruction accesses an internal location above address 7FH, the address mode used in the instruction specifies whether the CPU accesses the upper 128 bytes of RAM or the SFR space. Instructions that use direct addressing access SFR space.

For example, the following direct addressing instruction accesses the SFR at location 0A0H (which is P2). MOV 0A0H, #data instructions that use indirect addressing access the upper 128 bytes of RAM. For example, the following indirect addressing instruction, where R0 contains 0A0H, accesses the data byte at address 0A0H, rather than P2 (whose address is 0A0H). MOV @R0, #data Note that stack operations are examples of indirect addressing, so the upper 128 bytes of data RAM are available as stack space.

**TIMER 0 AND 1**

Timer 0 and Timer 1 in the AT89C52 operate the same way as Timer 0 and Timer 1 in the AT89C51.

**TIMER 2**

Timer 2 is a 16-bit Timer/Counter that can operate as either a timer or an event counter. The type of operation is selected by bit C/T2 in the SFR T2CON (shown in Table 2). Timer 2 has three operating modes: capture, auto-reload (up or down counting), and baud rate generator. The modes are selected by bits in T2CON, as shown in Table 3. Timer 2 consists of two 8-bit registers, TH2 and TL2. In the Timer function, the TL2 register is incremented every machine cycle. Since a machine cycle consists of 12 oscillator periods, the count rate is 1/12 of the oscillator frequency. In the Counter function, the register is incremented in response to a 1-to-0 transition at its corresponding external input pin, T2. In this function, the external input is sampled during S5P2 of every machine cycle. When the samples show a high in one cycle and a low in the next cycle, the count is incremented. The new count value appears in the register during S3P1 of the cycle following the one in which the transition was detected. Since two

machine cycles (24 oscillator periods) are required to recognize a 1-to-0 transition, the maximum count rate is 1/24 of the oscillator frequency. To ensure that a given level is sampled at least once before it changes, the level should be held for at least one full machine cycle.

**Table 3 - Timer 2 Operating Modes**

| RCLK+TCLK | CP/RL2 | TR2 | MODE |
|---|---|---|---|
| 0 | 0 | 1 | 16-Bit/Auto-Reload |
| 0 | 1 | 1 | 16-Bit capture |
| 1 | X | 1 | Baud Rate Generator |
| X | X | 0 | (off) |

## CAPTURE MODE

In the capture mode, two options are selected by bit EXEN2 in T2CON. If EXEN2 = 0, Timer 2 is a 16-bit timer or counter which upon overflow sets bit TF2 in T2CON. This bit can then be used to generate an interrupt. If EXEN2 = 1, Timer 2 performs the same operation, but a 1 to 0 transition at external input T2EX also causes the current value in TH2 and TL2 to be captured into CAP2H and RCAP2L, respectively. In addition, the transition at T2EX causes bit EXF2 in T2CON to be set. The EXF2 bit, like TF2, can generate an interrupt.

**AUTO-RELOAD (UP OR DOWN COUNTER)**

Timer 2 can be programmed to count up or down when configured in its 16-bit auto-reload mode. This feature is invoked by the DCEN (Down Counter Enable) bit located in the SFR T2MOD. Upon reset, the DCEN bit is set to 0 so that timer 2 will default to count up. When DCEN is set, Timer 2 can count up or down, depending on the value of the T2EX pin. Timer 2 automatically starts counting up when DCEN = 0. In this mode, two options are selected by bit EXEN2 in T2CON. If EXEN2 = 0, Timer 2 counts up to 0FFFFH and then sets the TF2 bit upon overflow. The overflow also causes the timer registers to be reloaded with the 16-bit value in RCAP2H and RCAP2L. The values in Timer in Capture Mode RCAP2H and RCAP2L are preset by software. If EXEN2 = 1, a 16-bit reload can be triggered either by an overflow or by a 1 to 0 transition at external input T2EX. This transition also sets the EXF2 bit. Both the TF2 and EXF2 bits can generate an interrupt if enabled. Setting the DCEN bit enables Timer 2 to count up or down. In this mode, the T2EX pin controls the direction of the count. A logic 1 at T2EX makes Timer 2 count up. The timer will overflow at 0FFFFH and set the TF2 bit. This overflow also causes the 16-bit value in RCAP2H and RCAP2L to be reloaded into the timer registers, TH2 and TL2, respectively. A logic 0 at T2EX makes Timer 2 count down. The timer underflows when TH2 and TL2 equal the values stored in RCAP2H and RCAP2L. The underflow sets the TF2 bit and causes 0FFFFH to be reloaded into the timer registers. The EXF2 bit toggles whenever Timer 2 overflows or underflows and can be used as a 17th bit of resolution. In this operating mode, EXF2 does not flag an interrupt.

**BAUD RATE GENERATOR**

Timer 2 is selected as the baud rate generator by setting CLK and/or RCLK in T2CON. Note that the baud rates for transmit and receive can be different if Timer 2 is used for the receiver or transmitter and Timer 1 is used for the other function. Setting RCLK and/or TCLK puts Timer 2 into its baud rate generator mode. The baud rate generator mode is similar to the auto-reload mode, in that a rollover in TH2 causes the Timer 2 registers to be reloaded with the 16-bit value in registers RCAP2H and RCAP2L, which are preset by software.

The baud rates in Modes 1 and 3 are determined by Timer 2's overflow rate according to the following equation. The Timer can be configured for either timer or counter operation. In most applications, it is configured for timer operation (CP/T2 = 0). The timer operation is different for Timer 2 when it is used as a baud rate generator. Normally, as a timer, it increments every machine cycle (at 1/12 the oscillator frequency). As a baud rate generator, however, it increments every state time (at 1/2 the oscillator frequency). The baud rate formula is given below, where (RCAP2H, RCAP2L) is the content of RCAP2H and RCAP2L taken as a 16-bit unsigned integer. It is valid only if RCLK or TCLK = 1 in T2CON. Note that a rollover in TH2 does not set TF2 and will not generate an interrupt. Note too, that if EXEN2 is set, a 1 to 0 transition in T2EX will set EXF2 but will not cause a reload from (RCAP2H, RCAP2L) to (TH2, TL2). Thus when Timer 2 is in use as a baud rate generator, T2EX can be used as an extra external interrupt. Note that when Timer 2 is running (TR2 = 1) as a timer in the baud rate generator mode, TH2 or TL2 should not be read from or written to. Under these conditions, the Timer is incremented every state time, and the results of a read or write may not be accurate. The RCAP2

registers may be read but should not be written to, because a write might overlap a reload and cause write and/or reload errors. The timer should be turned off (clear TR2) before accessing the Timer 2 or RCAP2 registers.

## PROGRAMMABLE CLOCK OUT

A 50% duty cycle clock can be programmed to come out on P1.0 pin. This pin, besides being a regular I/O pin, has two alternate functions. It can be programmed to input the external clock for Timer/Counter 2 or to output a 50% duty cycle clock ranging from 61 Hz to 4 MHz at a 16 MHz operating frequency. To configure the Timer/Counter 2 as a clock generator, bit C/T2 (T2CON.1) must be cleared and bit T2OE (T2MOD.1) must be set. Bit TR2 (T2CON.2) starts and stops the timer. The clock-out frequency depends on the oscillator frequency and the reload value of Timer 2 capture registers (RCAP2H, RCAP2L), as shown in the following equation. In the clock-out mode, Timer 2 roll-overs will not generate an interrupt. This behavior is similar to when Timer 2 is used as a baud-rate generator. It is possible to use Timer 2 as a baud-rate generator and a clock generator simultaneously. Note, however, that the baud-rate and clock-out frequencies cannot be determined independently from one another since they both use RCAP2H and RCAP2L.

## UART

The UART in the AT89C52 operates the same way as the UART in the AT89C51.

## INTERRUPTS

The AT89C52 has a total of six interrupt vectors: two external interrupts (INT0 and INT1), three timer interrupts (Timers 0, 1, and 2), and

the serial port interrupt. Each of these interrupt sources can be individually enabled or disabled by setting or clearing a bit in Special Function Register IE. IE also contains a global disable bit, EA, which disables all interrupts at once. Note that Table 4 shows that bit position IE.6 is unimplemented. In the AT89C51, bit position IE.5 is also unimplemented. User software should not write 1s to these bit positions, since they may be used in future AT89 products. Timer 2 interrupt is generated by the logical OR of bits TF2 and EXF2 in register T2CON. Neither of these flags is cleared by hardware when the service routine is vectored to. In fact, the service routine may have to determine whether it was TF2 or EXF2 that generated the interrupt, and that bit will have to be cleared in software. The Timer 0 and Timer 1 flags, TF0 and TF1, are set at S5P2 of the cycle in which the timers overflow. The values are then polled by the circuitry in the next cycle. However, the Timer 2 flag, TF2, is set at S2P2 and is polled in the same cycle in which the timer overflows.

**Table 4 – Interrupt Enable (IE) Register**

(MSB) (LSB)

| EA | – | ET2 | ES | ET1 | EX1 | ET0 | EX0 |
|---|---|---|---|---|---|---|---|

Enable Bit = 1 enables the interrupt.

Enable Bit = 0 disables the interrupt.

| Symbol | Position | Function |
|---|---|---|
| EA | IE.7 | Disables all interrupts. If EA = 0, no interrupt is acknowledged. If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit. |
| – | IE.6 | Reserved. |
| ET2 | IE.5 | Timer 2 interrupt enable bit. |
| ES | IE.4 | Serial Port interrupt enable bit. |
| ET1 | IE.3 | Timer 1 interrupt enable bit. |
| EX1 | IE.2 | External interrupt 1 enable bit. |
| ET0 | IE.1 | Timer 0 interrupt enable bit. |
| EX0 | IE.0 | External interrupt 0 enable bit. |
| User software should never write 1s to unimplemented bits, because they may be used in future AT89 products. | | |

## 3.4 OSCILLATOR CHARACTERISTICS

XTAL1 and XTAL2 are the input and output, respectively, of an inverting amplifier that can be configured for use as an on-chip oscillator.

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 8. 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. 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. Note that when idle mode 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 mode is terminated by a reset, the instruction following the one that invokes idle mode should not write to a port pin or to external memory.

## 3.5 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.

**Table 5 – Status of External Pins during Idle and Power-down Modes**

| Mode | Program Memory | ALE | PSEN | PORT0 | PORT1 | PORT2 | PORT3 |
|---|---|---|---|---|---|---|---|
| Idle | Internal | 1 | 1 | Data | Data | Data | Data |
| Idle | External | 1 | 1 | Float | Data | Address | Data |
| Power-down | Internal | 0 | 0 | Data | Data | Data | Data |
| Power-down | External | 0 | 0 | Float | Data | Data | Data |

## 3.6 PROGRAM MEMORY LOCK BITS

The AT89C52 has three lock bits that can be left unprogrammed (U) or can be programmed (P) to obtain the additional features listed in the following table.

**Lock Bit Protection Modes**

| Program Lock Bits | | | | Protection Type |
|---|---|---|---|---|
| | LB1 | LB2 | LB3 | |
| 1 | U | U | U | No program lock features. |
| 2 | P | U | U | MOVC instructions executed from external program memory are disabled from fetching code bytes from internal memory, $\overline{EA}$ is sampled and latched on reset, and further programming of the Flash memory is disabled. |
| 3 | P | P | U | Same as mode 2, but verify is also disabled. |
| 4 | P | P | P | Same as mode 3, but external execution is also disabled. |

**Table - 6**

When lock bit 1 is programmed, the logic level at the EA pin is sampled and latched during reset. If the device is powered up without a reset, the latch initializes to a random value and holds that value until reset is activated. The latched value of EA must agree with the current logic level at that pin in order for the device to function properly.

## 3.7 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-volt) or a low-voltage ($V_{CC}$) program enable signal. The Low-voltage programming mode provides a convenient way to program the 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 respective top-side marking and device signature codes are listed in the following table.

**Table 7 - Top-side Marking and Device Signature Codes**

| | $V_{PP}$ = 12V | $V_{PP}$ = 5V |
|---|---|---|
| Top-side Mark | AT89C52<br>xxxx<br>yyww | AT89C52<br>xxxx - 5<br>yyww |

| | $V_{PP}$ = 12V | $V_{PP}$ = 5V |
|---|---|---|
| Signature | (030H) = 1EH<br>(031H) = 52H<br>(032H) = FFH | (030H) = 1EH<br>(031H) = 52H<br>(032H) = 05H |

The AT89C52 code memory array is programmed byte-by-byte in either programming mode. To program any nonblank byte in the on-chip

Flash Memory, the entire memory must be erased using the Chip Erase Mode.

## 3.8 PROGRAMMING ALGORITHM

Before programming the AT89C52, the address, data and control signals should be set up according to the Flash programming mode table and Figure 1 and Figure 2. To program the AT89C52, take the following steps.

1. Input the desired memory location on the address lines.
2. Input the appropriate data byte on the data lines.
3. Activate the correct combination of control signals.
4. Raise EA/VPP to 12V for the high-voltage programming mode.
5. Pulse ALE/PROG once to program a byte in the Flash array or the lock bits. The byte-write cycle is self-timed and typically takes no more than 1.5 ms.
6. Repeat steps 1 through 5, changing the address and data for the entire array or until the end of the object file is reached.

## 3.9 DATA POLLING

The AT89C52 features Data Polling to indicate the end of a write cycle. During a write cycle, an attempted read of the last byte written will result in the complement of the written data on PO.7. Once the write cycle has been completed, true data is valid on all outputs, and the next cycle may begin. Data Polling may begin any time after a write cycle has been initiated.

## 3.10 READY/BUSY

The progress of byte programming can also be monitored by the RDY/BSY output signal. P3.4 is pulled low after ALE goes high during programming to indicate BUSY. P3.4 is pulled high again when programming is done to indicate READY.

## 3.11 PROGRAM VERIFY

If lock bits LB1 and LB2 have not been programmed, the programmed code data can be read back via the address and data lines for verification. The lock bits cannot be verified directly. Verification of the lock bits is achieved by observing that their features are enabled.

## 3.12 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 1s. The chip erase operation must be executed before the code memory can be reprogrammed.

## 3.13 READING THE SIGNATURE BYTES

The signature bytes are read by the same procedure as a normal verification of locations 030H, 031H, and 032H, except that P3.6 and P3.7 must be pulled to a logic low. The values returned are as follows.

(030H) = 1EH indicates manufactured by Atmel

(031H) = 52H indicates 89C52

(032H) = FFH indicates 12V programming

(032H) = 05H indicates 5V programming

**Table 8**

**Flash Programming Modes**

| Mode | | RST | $\overline{PSEN}$ | ALE/$\overline{PROG}$ | $\overline{EA}$/$V_{PP}$ | P2.6 | P2.7 | P3.6 | P3.7 |
|---|---|---|---|---|---|---|---|---|---|
| Write Code Data | | H | L | | H/12V | L | H | H | H |
| Read Code Data | | H | L | H | H | L | L | H | H |
| Write Lock | Bit - 1 | H | L | | H/12V | H | H | H | H |
| | Bit - 2 | H | L | | H/12V | H | H | L | L |
| | Bit - 3 | H | L | | H/12V | H | L | H | L |
| Chip Erase | | H | L | (1) | H/12V | H | L | L | L |
| Read Signature Byte | | H | L | H | H | L | L | L | L |

Note: 1 Chip Erase requires a 10 ms $\overline{PROG}$ pulse.

## 3.14 PROGRAMMING INTERFACE

Every code byte in the Flash array can be written, and the entire array can be erased, by using the appropriate combination of control signals. The write operation cycle is self timed and once initiated, will automatically time itself to completion. All major programming vendors offer worldwide support for the Atmel microcontroller series. Please contact your local programming vendor for the appropriate software revision. Figure3.3

Figure 6. Programming the Flash Memory

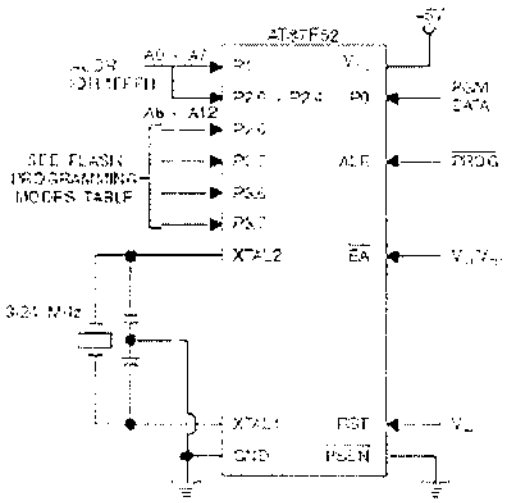
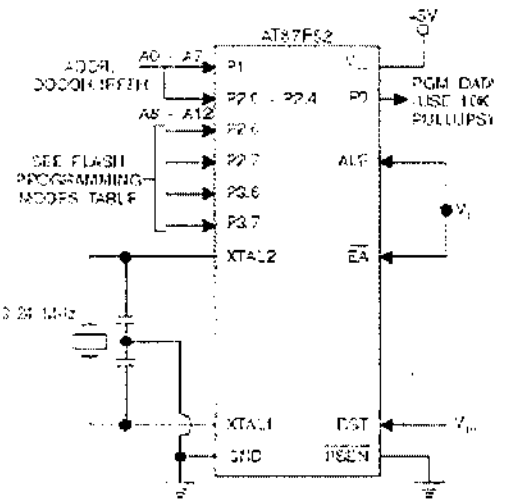



Figure 7.



# 4. RS 232

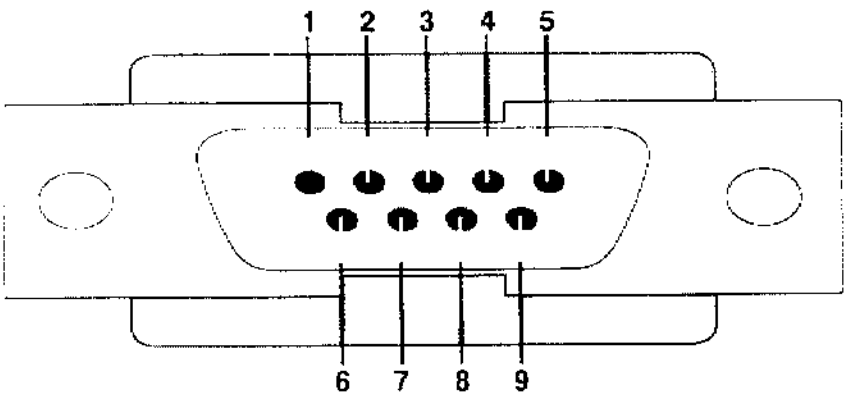
## 4.1 RS 232 STANDARDS

To allow compatibility among data communication equipment made by various manufactures, an interfacing standards called RS 232 are said by electronics industries association (EIA) in 1960. However, since the standards were set longed before the advent of the TTL logic family, its input and output voltages levels are not TTL compatible. In RS232, 1 bit is represented by -3 to -25 volt while a 0 bit is +3 to + 25 Volt, making -3 to +3 undefined. For this reason to connect any RS 232 to a micro controller system we must use voltage converters such as MAX 232 to convert the TTL level to the RS232 voltage level and Vice versa. MAX 232 IC chip are commonly referred to as line drivers

## 4.2 RS 232 PINS

The table provides the pins and the labels for the RS 232 cable, commonly referred to as the DB-25 connector.

**Figure 4.1 – RS232 Pin Layout**




| Pin | Signal | Pin | Signal |
|---|---|---|---|
| 1 | Data Carrier Detect | 6 | Data Set Ready |
| 2 | Received Data | 7 | Request to Send |
| 3 | Transmitted Data | 8 | Clear to Send |
| 4 | Data Terminal Ready | 9 | Ring Indicator |
| 5 | Signal Ground | | |

The standards for RS 232 and similar interfaces usually restrict RS232 to 20 Kbps or less and line length of 15 m or less. RS232 is fair more robust than the traditional limits of 20 Kbps over a 15 m line would imply RS232 is perfectly adequate at speed up to 200 Kbps most RS232 ports on main frame and midrange computers are capable of far higher speeds than their rated 19.2kbps .Usually these 1000 speed parts will run error free at 56kbps and above the 15 m limitation for enable length can be stretched to about 30 m for ordinary cable if well screened and grounded, and about 100m if the cable is 10m capacity.

## 4.3 FEATURES OF RS 232

The essential feature of RS 232 is the signals are carried as signal voltages referred to a common earth and print data to a common earth pin data is transmitted and received on pins 2 and 3 respectively. Data set ready (DSR) is an indication from the data set i.e., the modem DSU/CSU that's it is on. Similarly, DTR indicates to the data set that the DTE is on. Data carrier (DCD) indicates that carrier for transmit data is on. Pins 4 and 5 carry the RTS and CTS signals. In most situations RTS and CTS are constantly go throughout the communication session. How ever when the DTE is connected to a turn carrier on the modem on and off. On a multipoint linear, it is imperative that only one station is transmitting at a line when a station wants to transmit, it raises RTS, the modem turns on carrier.

**The truth table for RS 232**

Signal > +3V=0

Signal < - 3V=1

The output signal level usually swings between +12V and -12V.The "dead area" between +3V and -3V is designed to absorb line noise.

## 4.4 MAX 232

MAX 232 chips are commonly referred to as line drivers. Since the RS232 is not compatible with today's micro processors and micro controllers, we need a line driver (voltage converter) to convert the RS232's signals to TTL voltage levels that will be acceptable to the 8051's TxD and RxD pins. The MAX 232 converts from RS232 voltage levels to TTL voltage levels and viz. one advantage of the MAX 232 chip is that it uses a +5 volt power source, which is the same as the source voltage for the 8051. In other words with a signal +5 V power supply we can power both the 8051 and MAX 232, with no need for the dual power supplies that is common in many holder systems.

The MAX 232 has two sets of line drivers for transferring and receiving data as shown in figure 6. The line drivers used for TxD are called T1 and T2, while the line drivers for RxD designated as R1 and R2. In many applications only one of each is used. For example T1 and R1 are used together for TxD and RxD of the 8051 and the second set is left unused. Notice in MAX 232 that the T1 line driver has a designation of T1 in and T1 out on pin numbers 11 and 14 respectively.

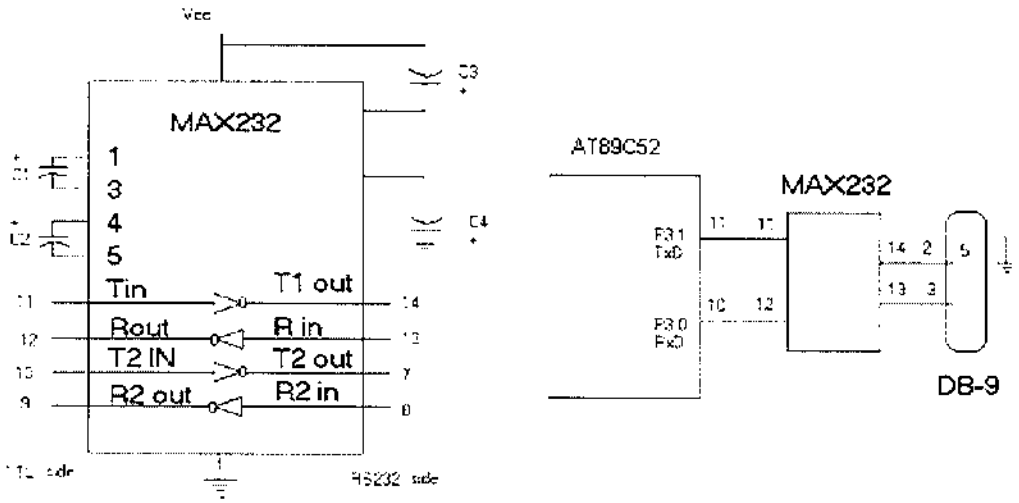
The T1 in pin is the TTL side and is connected to TxD of the micro controller, while T1 out is the RS 232 side that is connected to the RxD pin of the RS 232 DB connector. The R1 line driver has a designation of R1 in and R1 out on pin numbers 13 and 12 respectively.

The R1 in (pin 13) is the RS232 side that is connected to the TxD pin of the RS 232 DB connector and R1 out (pin 12) is the TTL side that is

connected to the RxD pin of the micro controller. See figure notice the null modem connection where RxD for one is TxD or the other.

MAX 232 requires 4 capacitors ranging from 1 to 22 μF. The most widely used value for this capacitor is 22 μF.

Figure 4.2 – MAX232 Interfacing



# 5. SERIAL COMMUNICATION

To transfer to a device located many meters away, the serial data transfer is used. In serial communication, the data is sent one bit at a time. AT89C52 has serial communication capability built into it, thereby making possible the data transfer using only a few wire.

For serial data communication to work, the byte of data must be converted to serial bits using a parallel-in-serial-out shift register, because the microcontroller is sending parallel data. Then it can be transmitted over a single data line. This also means at the receiving end there must be a serial-in-parallel-out shift register. The basic block diagram of serial communication is shown in figure.

**Figure 5.1 – Communication Module**


## 5.1 SERIAL DATA TRANSFER

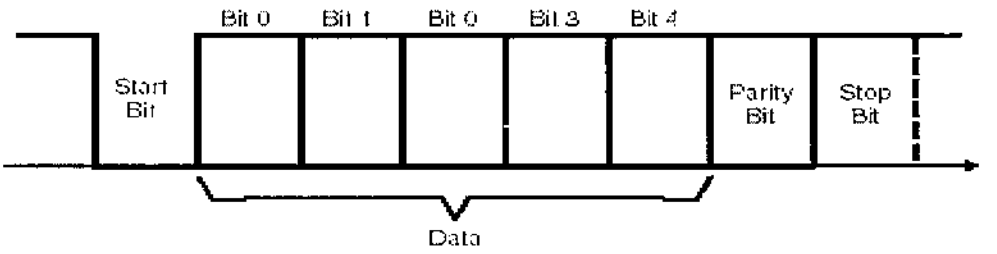
Serial data transfer uses two methods, Asynchronous and Synchronous. The synchronous method transfers a block of data (character) at a time while the asynchronous transfer a single byte at a time. It is possible to write software to use either of these methods, but the programs can be tedious and long. For this reason, there are special IC chips made by many manufactures for serial data communication. These chips are commonly referred to as UART (Universal Asynchronous Receiver

Transmitter) and USART (Universal Synchronous-Asynchronous Receiver Transmitter).

## 5.2 ASYNCHRONOUS SERIAL COMMUNICATION

Asynchronous serial data communication is widely used for character-oriented transmissions, while block-oriented data transfers use the synchronous method. In the asynchronous method, each character is placed in between a start bit and a stop bit. The start bit is always one bit but the stop bit can be one or two bits. The start bit is always a 0 (low) and the stop bit is 1 (High).

**Figure 5.2 - Asynchronous Serial Communication – Timing Diagram**




## 5.3 DATA TRANSFER RATE

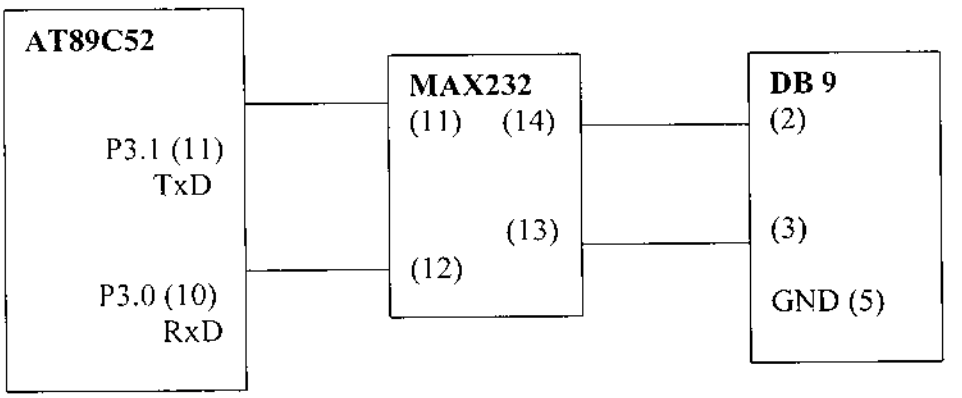
The rate of data transfer in serial data communication is stated in bps (bits per second). Another widely used technology for bps is baud rate. However, the baud rate and bps rates are not necessarily equal. This is due to the fact that baud rate is the modern terminology and is defined as the number of signal changes per second.

The data transfer rate of a given computer system depends on communication ports incorporated into that system. For e.g. the early IBM PC could transfer data at the rate of 100 to 9600 bps. Now the recent computers can transfer data at rate as high as 512 kbps. In asynchronous serial communication the baud rate is generally limited to 100,000 bps.

## 5.4 RxD AND TxD PINS IN AT89C52

The 8952 has two pins that are used specifically for transferring and receiving data serially. These two pins are called TxD and RxD and are part of the port 3 group (P3.0 and P3.1 pin). Pin 11 of the 8952 (P3.1) is assigned to TxD and pin 10 (P3.0) is designated as RxD. These pins are TTL compatible.

**Figure 5.3 – Processor Interfacing**




The MAX 232 converts from Rs232 voltage level to TTL voltage levels, and vice versa. One advantage of MAX232 chip is that it uses a +5V power supply which is the same as the source voltage of 89c52.

# 6. LCD DISPLAY

## 6.1 LCD OPERATION

The LCDs are used widely in all application namely microcontroller instead of LEDs or other multi segment LEDs. This is due to

- The decline prices of LCDs
- The ability to display numbers, characters and graphics. This is in contrast to LEDs, which are limited to numbers and a few characters.
- Incorporation of a refreshing controller into the LCD, thereby, relieving the CPU of the task of refreshing the LCD. In contrast, the LED must be refreshed by the CPU (or in some other way) to keep displaying the data.
- Ease of programming for characters and graphics.

## 6.2 FUNCTIONAL DESCRIPTION OF LCD:

The LCD discussed in this section is LAMPEX 16101, 2x1 lines LCD. There are 16 segments and each segment has 5x7 matrix display. The address of each segments are 80-87 and C0-C7. The data being transferred as 8bit data transfer to the 16bit LCD. LAMPEX 16101 has 16 pins and are described below.

**Table 9 - Pin Description for LCD**

| No of Pin | Symbols | I/O | Description |
|---|---|---|---|
| 1 | Vss | ----- | Ground |
| 2 | Vcc | ----- | +5 V power supply |
| 3 | Vo | ----- | Power supply to control contrast |

| | | | |
|---|---|---|---|
| 4 | RS | I | RS=0 to select command register<br>RS=1 to select data register |
| 5 | R/W | I | R/W=0 for write<br>R/W=1 for read |
| 6 | E | I/O | Enable |
| 7 | DB0 | I/O | The 8-bit data bus |
| 8 | DB1 | I/O | The 8-bit data bus |
| 9 | DB2 | I/O | The 8-bit data bus |
| 10 | DB3 | I/O | The 8-bit data bus |
| 11 | DB4 | I/O | The 8-bit data bus |
| 12 | DB5 | I/O | The 8-bit data bus |
| 13 | DB6 | I/O | The 8-bit data bus |
| 14 | DB7 | I/O | The 8-bit data bus |
| 15 | Lamp+ | | Back ground light |
| 16 | Lamp- | | Back ground light |

## 6.3 INITIALIZATION OF LCD

**CLEAR DISPLAY**

Clears the display and returns the cursor to home position. The busy flag is kept in the Busy state (BF=1) unit initialization ends. The time is 15ms.

**FUNCTION SET**

Sets interface data length (DL), number of display lines (N) and character font (F).

DL =1, 8-bit long interface data

N =0, 2x1 line LCD

**DISPLAY ON/OFF CONTROL**

Sets ON/OFF all display (D), cursor ON/OFF (C), and blink of cursor position character (B).

D=0 display OFF

C=0 cursor OFF

B=0 Blink OFF

**ENTRY MODE SET**

Set the cursor move direction and specifies or not shift the display. These operations are performed during data write and read of RAM.

I/D =1 +1 increment

S =0 no shift

## 6.4 ADVANTAGES

1. Consumes much lesser energy (i.e., low power) when compared to LEDs.
2. Utilizes the light available outside and no generation of light.
3. Since very thin layer of liquid crystal is used, more suitable to act as display elements (in digital watches, pocket calculators, etc...)
4. Since reflectivity is highly sensitive to temperature, used as temperature measuring sensor.
5. Very cheap.

## 6.5 DISADVANTAGES

1. Angle of viewing is limited and external light is must for display.
2. Since not generating its own light and makes use of external display, contrast is poor.
3. Cannot be used under wide range of temperature.

# 7.I²C

**I²C** is a serial computer bus invented by Philips that is used to attach low-speed peripherals to a motherboard, embedded system, or cellphone. The name is an acronym for **Inter-Integrated Circuit** and is pronounced *I-squared-C.*

## 7.1 DESIGN

I²C uses only two bi-directional lines, clock and data, both running at +5 V and pulled high with resistors. The I²C reference design has a 7-bit address space with 16 reserved addresses, so a maximum of 112 nodes can communicate on the same bus. The most common I²C bus modes are the 100 kbit/s *standard mode* and the 10 kbit/s *low-speed mode*, but clock frequencies down to zero are also allowed. Recent revisions of I²C can host more nodes and run faster (400 kbit/s *Fast mode* and 3.4 Mbit/s *High Speed mode*) .

## 7.2 APPLICATIONS

I²C is appropriate for peripherals where simplicity and low manufacturing cost are more important than speed. Common applications of the I²C bus are:

- Accessing NVRAM chips that keep user settings.
- Accessing low speed DACs.
- Accessing low speed ADCs.
- Changing contrast, hue, and color balance settings in monitors.
- Changing sound volume in intelligent speakers.

- Controlling LED displays, like in a cellphone.
- Reading hardware monitors and diagnostic sensors, like a CPU thermostat and fan speed.
- Reading real time clocks.
- Turning on and turning off the power supply of system components.

A particular strength of I²C is that a microcontroller can control a network of device chips with just two general-purpose I/O pins and software.

Peripherals can also be added to or removed from the I²C bus while the system is running, which makes it ideal for applications that require hot swappable components.

Buses like I²C became popular when computer engineers realized that much of the manufacturing cost of an integrated circuit design results from its package size and pin count. A smaller package also usually weighs less and consumes less power, which is especially important in cellphones and portable computing.

# 8. SMART CARD

## 8.1 FEATURES

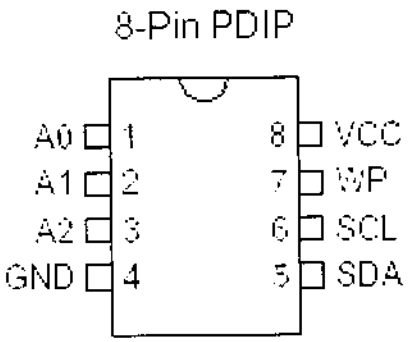
- Low-Voltage and Standard-Voltage Operation
  - 5.0 (VCC = 4.5V to 5.5V)
  - 2.7 (VCC = 2.7V to 5.5V)
  - 2.5 (VCC = 2.5V to 5.5V)
  - 1.8 (VCC = 1.8V to 5.5V)
- Low-Power Devices (ISB = 2 μA @ 5.5V) Available
- Internally Organized 4096 x 8, 8192 x 8
- 2-Wire Serial Interface
- Schmitt Trigger, Filtered Inputs for Noise Suppression
- Bidirectional Data Transfer Protocol
- 100 kHz (1.8V, 2.5V, 2.7V) and 400 kHz (5V) Compatibility
- Write Protect Pin for Hardware Data Protection
- 32-Byte Page Write Mode (Partial Page Writes Allowed)
- Self-Timed Write Cycle (10 ms max)
- High Reliability
  - Endurance: 1 Million Write Cycles
  - Data Retention: 100 Years
  - ESD Protection: >3,000V
- Automotive Grade and Extended Temperature Devices Available
- 8-Pin JEDEC PDIP, 8-Pin and 14-Pin JEDEC SOIC, 8-Pin EIAJ SOIC, and 8-pin TSSOP Packages

## 8.2 DESCRIPTION

The AT24C32/64 provides 32,768/65,536 bits of serial electrically erasable and programmable read only memory (EEPROM) organized as 4096/8192 words of 8 bits each. The device's cascadable feature allows up

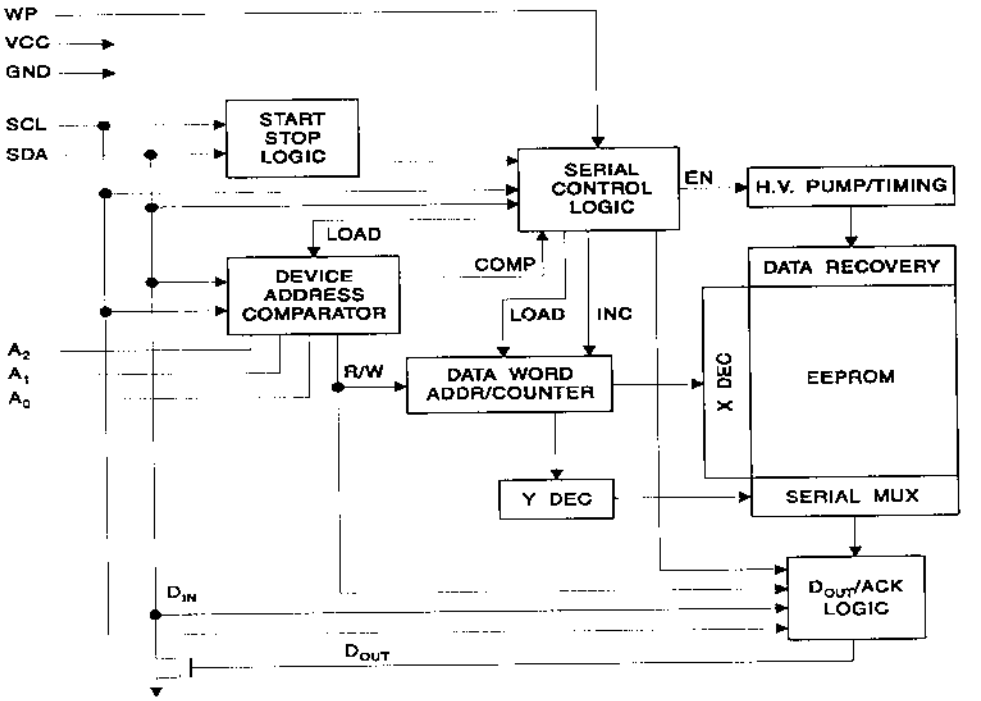
to 8 devices to share a common 2- wire bus. The device is optimized for use in many industrial and commercial applications where low power and low voltage operation are essential. The AT24C32/64 is available in space saving 8-pin JEDEC PDIP, 8-pin and 14-pin JEDEC SOIC, 8-pin EIAJ SOIC, and 8-pin TSSOP packages and is accessed via a 2-wire serial interface. In addition, the entire family is available in 5.0V (4.5V to 5.5V), 2.7V (2.7V to 5.5V), 2.5V (2.5V to 5.5V) and 1.8V (1.8V to 5.5V) versions.

**Figure 8.1 – AT24C64 - Pin Diagram**




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

**Figure 8.2 – AT24C64 – Block Diagram**



## 8.3 PIN DESCRIPTION

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

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

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

**WRITE PROTECT (WP):** The write protect input, when tied to GND, allows normal write operations. When WP is tied high to VCC, all write operations to the upper quadrant (8/16K bits) of memory are inhibited. If left unconnected, WP is internally pulled down to GND. Memory Organization AT24C32/64, 32K/64K SERIAL EEPROM: The 32K/64K is internally organized as 256 pages of 32 bytes each. Random word addressing requires a 12/13 bit data word address.

# 9. DEVICE OPERATION

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

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

**STOP CONDITION:** A low-to-high transition of SDA with SCL high is a stop condition. After a read sequence, the stop command will place the EEPROM in a standby power mode.

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

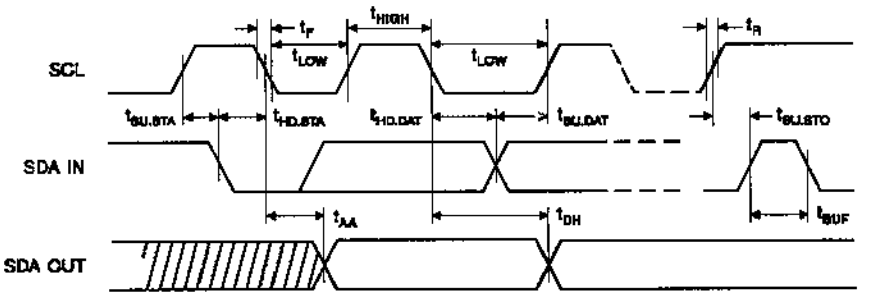
**STANDBY MODE:** The AT24C32/64 features a low power standby mode which is enabled: a) upon power-up and b) after the receipt of the STOP bit and the completion of any internal operations.

**MEMORY RESET:** After an interruption in protocol, power loss or system reset, any 2-wire part can be reset by following these steps:

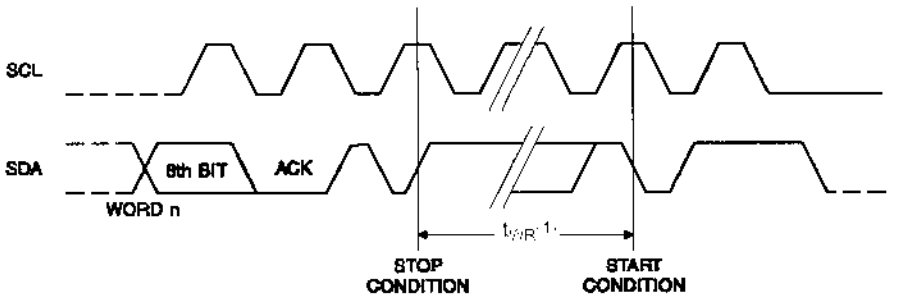
(a) Clock up to 9 cycles, (b) look for SDA high in each cycle while SCL is high and then (c) create a start condition as SDA is high.

## Figure 9.1 – Timing Diagrams

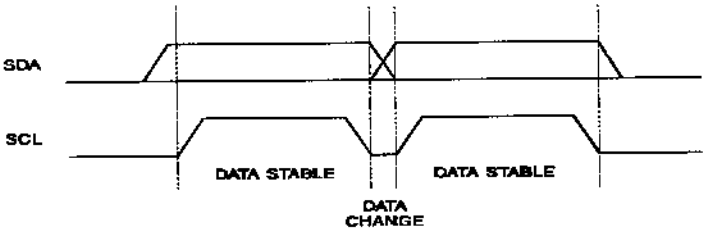
Bus Timing
SCL: Serial Clock, SDA: Serial Data I/O



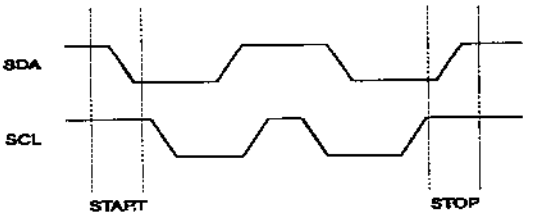

Write Cycle Timing
SCL: Serial Clock, SDA: Serial Data I/O



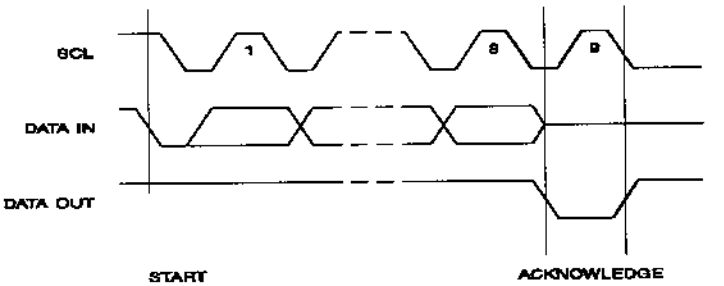
Data Validity




Start and Stop Definition




Output Acknowledge




## 9.1 DEVICE ADDRESSING

The 32K/64K EEPROM requires an 8-bit device address word following a start condition to enable the chip for a read or write operation (refer to Figure 15). The device address word consists of a mandatory one, zero sequence for the first four most significant bits as shown. This is common to all 2-wire EEPROM devices. The 32K/64K uses the three device address bits A2, A1, A0 to allow as many as eight devices on the same bus. These bits must compare to their corresponding hardwired input pins. The A2, A1, and A0 pins use an internal proprietary circuit that biases them to a logic low condition if the pins are allowed to float. The eighth bit of the device address is the read/write operation select bit. A read operation is

initiated if this bit is high and a write operation is initiated if this bit is low. Upon a compare of the device address, the EEPROM will output a zero. If a compare is not made, the device will return to standby state.

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

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

## 9.2 WRITE OPERATIONS

**BYTE WRITE:** A write operation requires two 8-bit data word addresses following the device address word and acknowledgment. Upon receipt of this address, the EEPROM will again respond with a zero and then clock in the first 8-bit data word. Following receipt of the 8-bit data word, the EEPROM will output a zero and the addressing device, such as a microcontroller, must terminate the write sequence with a stop condition. At this time the EEPROM enters an internally-timed write cycle, $t_{WR}$, to the nonvolatile memory. All inputs are disabled during this write cycle and the EEPROM will not respond until the write is complete (refer to Figure 16).

**PAGE WRITE:** The 32K/64K EEPROM is capable of 32-byte page writes. A page write is initiated the same way as a byte write, but the

microcontroller does not send a stop condition after the first data word is clocked in. Instead, after the EEPROM acknowledges receipt of the first data word, the microcontroller can transmit up to 31 more data words. The EEPROM will respond with a zero after each data word received. The microcontroller must terminate the page write sequence with a stop condition (refer to Figure 16).

The data word address lower 5 bits are internally incremented following the receipt of each data word. The higher data word address bits are not incremented, retaining the memory page row location. When the word address, internally generated, reaches the page boundary, the following byte is placed at the beginning of the same page. If more than 32 data words are transmitted to the EEPROM, the data word address will "roll over" and previous data will be overwritten.

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

## 9.3 READ OPERATIONS

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

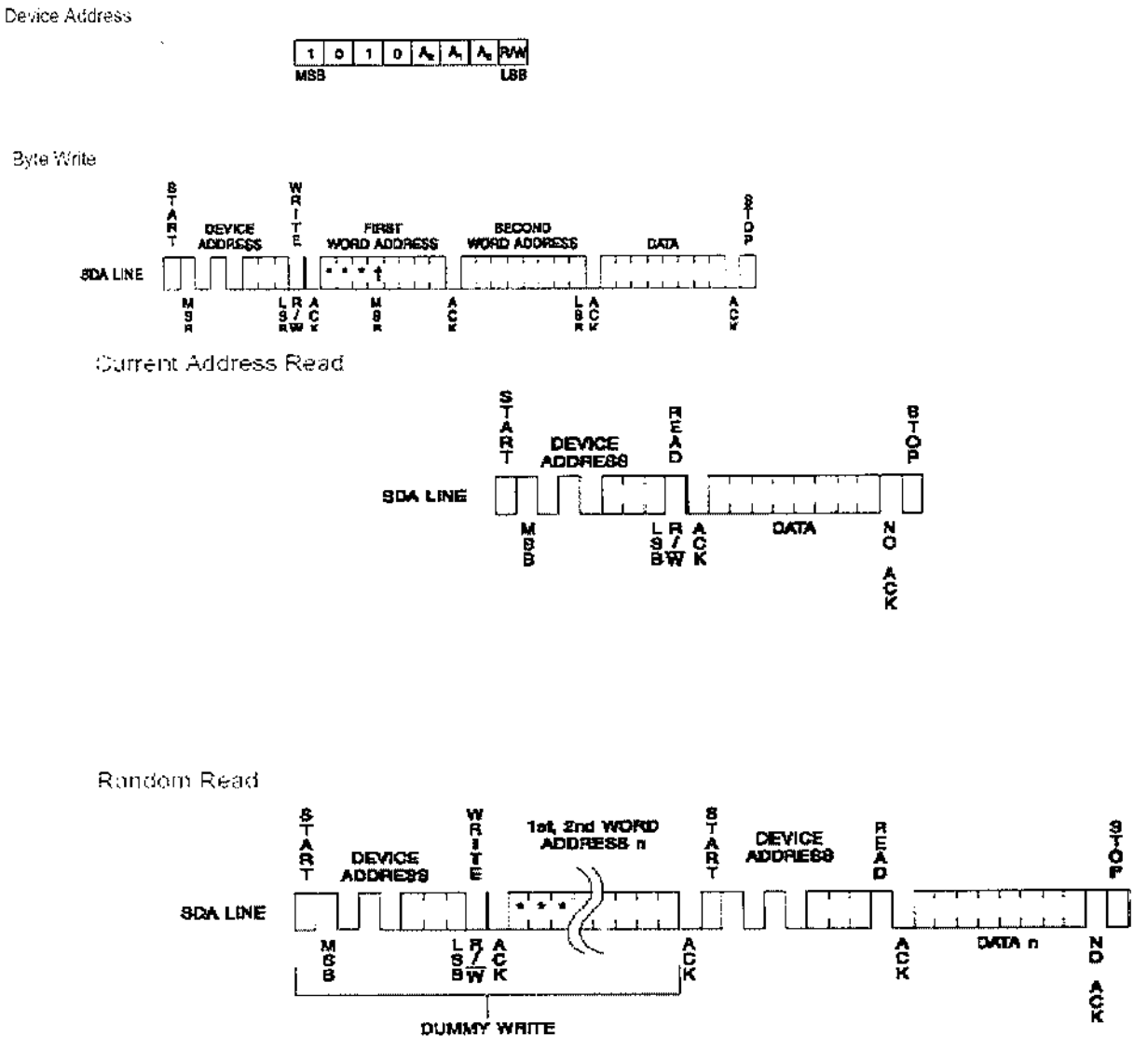
**CURRENT ADDRESS READ:** The internal data word address counter maintains the last address accessed during the last read or write operation, incremented by one. This address stays valid between operations as long as the chip power is maintained. The address "roll over" during read is from the last byte of the last memory page, to the first byte of the first page. The address "roll over" during write is from the last byte of the current page to the first byte of the same page. Once the device address with the read/write select bit set to one is clocked in and acknowledged by the EEPROM, the current address data word is serially clocked out. The microcontroller does not respond with an input zero but does generate a following stop condition (refer to Figure 16).

**RANDOM READ:** A random read requires a "dummy" byte write sequence to load in the data word address. Once the device address word and data word address are clocked in and acknowledged by the EEPROM, the microcontroller must generate another start condition. The microcontroller now initiates a current address read by sending a device address with the read/write select bit high. The EEPROM acknowledges the device address and serially clocks out the data word. The microcontroller does not respond with a zero but does generate a following stop condition (refer to Figure 16).

**SEQUENTIAL READ:** Sequential reads are initiated by either a current address read or a random address read. After the microcontroller receives a data word, it responds with an acknowledge. As long as the EEPROM receives an acknowledgement, it will continue to increment the data word address and serially clock out sequential data words. When the memory address limit is reached, the data word address will "roll over" and the

sequential read will continue. The sequential read operation is terminated when the microcontroller does not respond with a zero but does generate a following stop condition (refer to Figure 16).

## Figure 9.2




Notes

Note: 1. * = DON'T CARE bits

# 10. POWER SUPPLY

## 10.1 CIRCUIT DESCRIPTION

The circuit diagram containing the parts of a power supply and the voltage at various points in the unit is shown. The AC voltage typically 230V is connected to a transformer, which steps down the AC voltage down to the level of desired output .A diode rectifier then provides a rectified voltage that is initially filtered by a simple capacitor filter to produce a DC voltage. This resulting DC voltage usually has some ripple or AC voltage variation. A regulator circuit can use this DC input to provide a DC voltage that not only has much less ripple voltage but also remains the same DC value even if the input DC voltage varies somewhat, or the load connected to the output DC voltage changes. This voltage regulation is usually obtained by using one of popular voltage regulator IC's.

## 10.2 TRANSFORMER SECTION

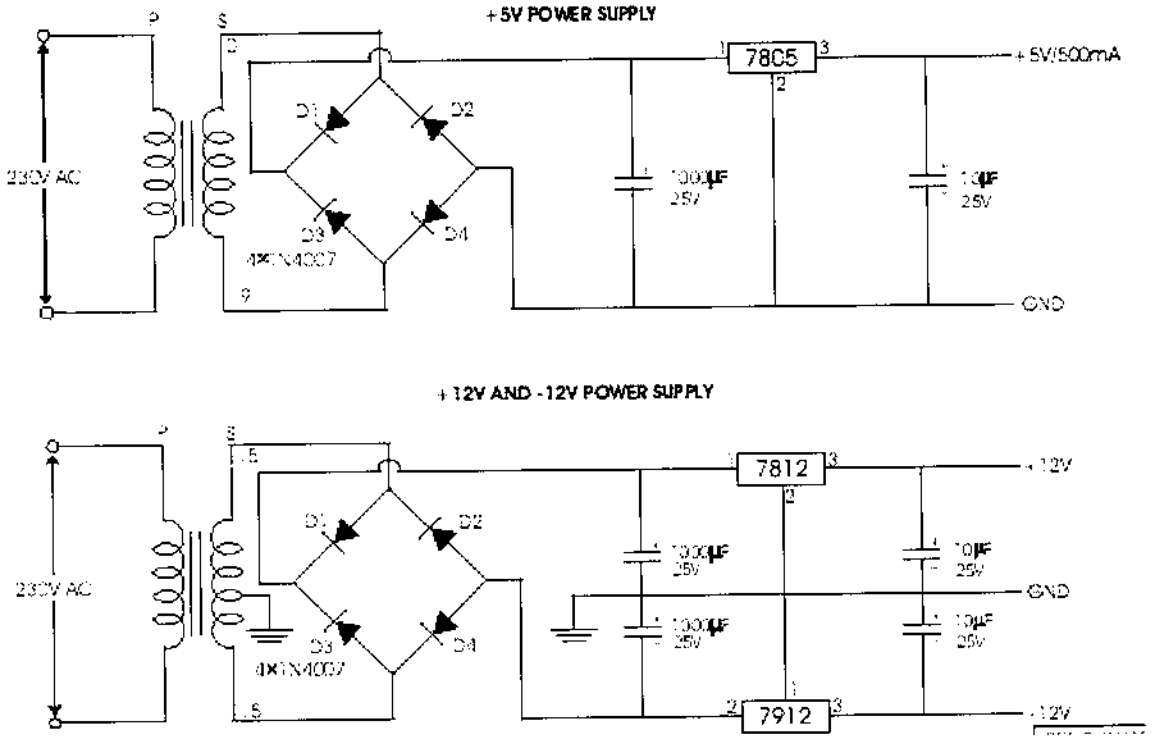
A transformer is a static device in which electric power in one circuit is transformed into electric power of same frequency in another circuit. It provides a decrease or increase in the output section along with a decrease or increase in the current. It works in the principle of mutual induction. It provides isolation to the circuit. The stepped down transformer is used in this section

## 10.3 RECTIFIER SECTION

The process of rectification really means conversion of AC into DC voltage. The DC level obtained from a sinusoidal input can be improved 100% using a process called bridge rectification .it uses 4 diodes in a bridge

configuration. In the bridge opposite two diodes are on in one cycle and other are on in the other cycle. In one cycle D1 and D3 conducts and in the other cycle D2 and D4 is on. Irrespective of the input cycle the output polarity across the load remains same.

**Figure 10.1 POWER SUPPLY – CIRCUIT DIAGRAM**



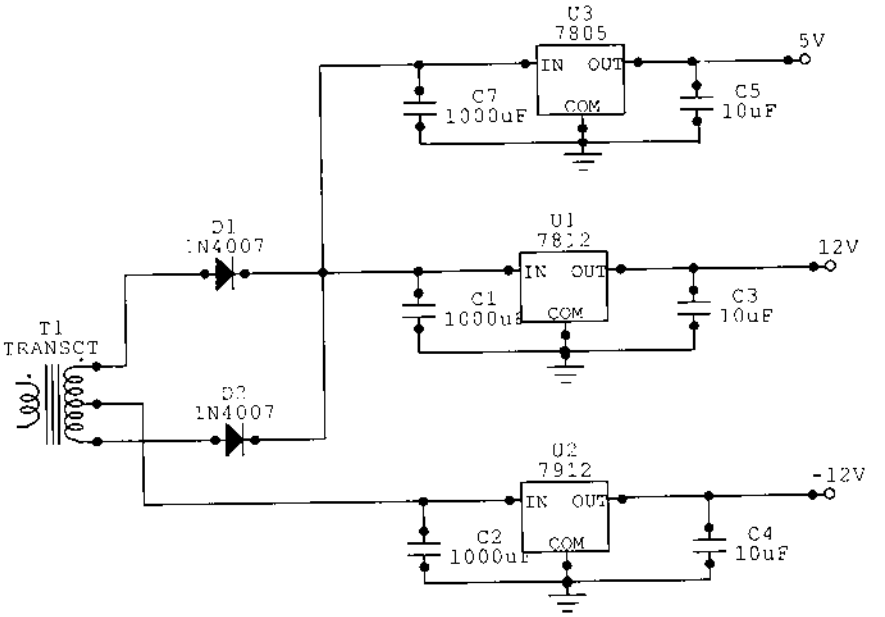

## 10.4 FILTER SECTION

The filter circuit used here is the capacitor filter where a capacitor is connected at the rectifier output, and the DC voltage is obtained at the output, the filter filters the ac components. Still the output contains negligible ripple.

## 10.5 VOLTAGE REGULATOR SECTION

The voltage regulator is a device, which maintains the output voltage constant irrespective of the change in supply variations, load variations and temperature variations. Voltage regulators comprise a class of widely used IC's. Regulator IC units contains the circuitry for reference source, comparator, amplifier, control device and overload protection, all in a single IC A1 though the internal construction. If the IC is some what different for discrete voltage regulator circuits, the external operation is the same. IC units provide regulated output of either positive or negative voltages.

**Figure 10.2 RECTIFIER CIRCUIT DIAGRAM**




A power can be built using a transformer connected to the AC mains. The input voltage will be step down to the desired by choosing the appropriate transformer. They are sent through the rectifier circuit, then through the filter circuits. One important to note here is we to use higher rating capacitor at the output of the bridge rectifier circuit. The reason is

simple because we have high frequency ripples at the output of the transformer.

The series voltage regulators IC's used are 7805, 7812, 7905 and 7912.The 78 series stands for the type of output voltage that is available at the output. These types provide positive output and the 79 series provides negative voltage at the output. The 5 and 12 indicates the voltage level at the output.

# 11. KEYPAD

Keypad is the friendliest input peripheral. Both program and data can be keyed in through it in addition certain commands to software can be given from the keyboard.

The keypad consists of a set of key switches. There is one key switch for each letter, number, symbol etc, much like a typewriter.

When a key is pressed the key switch is activated. The key is an electronic circuit to determine which key has been pressed.

## 11.1 TYPES

- Function key
- Numerical keys
- Cursor control and editing keys
- Alphanumeric keys

## 11.2 KEYBOARD MICROCOMPUTER

The PC keyboard uses a microcomputer chip. The IBM pc keyboard uses Intel 8048 which is the single chip microcomputer with RAM and ROM. The clone's keyboard follows two different schemes. Some keyboard use a microprocessor such as Intel 8051 or Intel 8052 and external support hardware.

1. Scanning the key switch matrix
2. Detecting a key press
3. Debouching
4. Generating a make scan code

5. Buttering of up to 20 key scan codes if the pc is busy and not accepting the scan code.
6. Transmitting the scan code to the pc on serial interface
7. Providing bidirectional serial communication for data and clock line
8. Observing handshake protocol for each scans code transfer
9. Performing power on self test when requested by the pc system
10. Repeating the transmission of make scan code if a key is press down constantly.

## 11.3 KEY FUNCTION

The key switches are connected in a matrix of row and columns each key switch has a fixed set of coordinates.

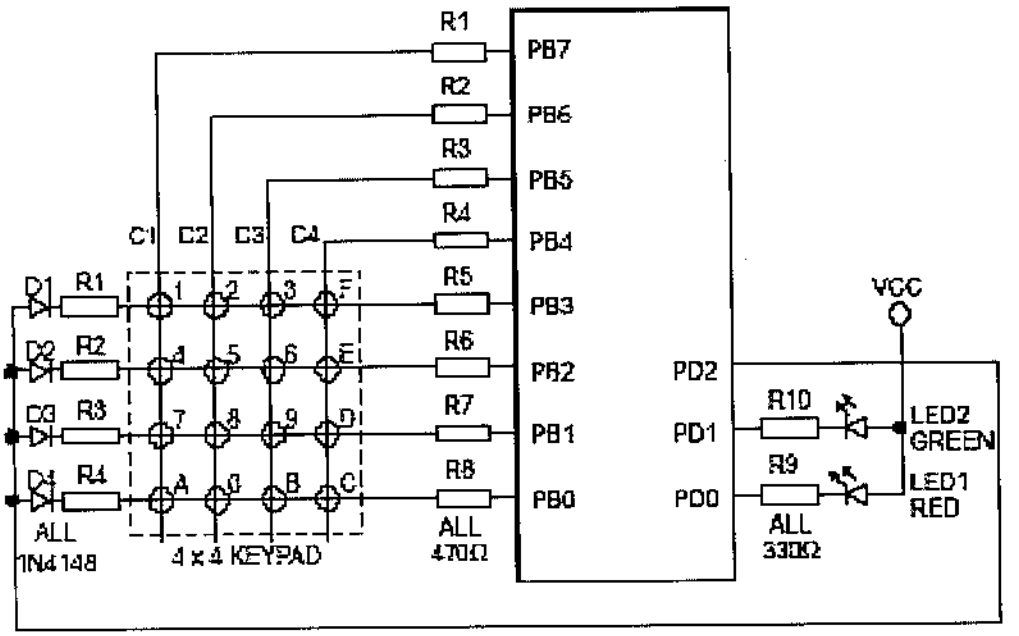
1. Sensing a key depression
2. Encoding
3. Sending the code to computer

## 11.4 KEYPAD INTERFACING

The keypad interface receives the scan code in a serial format. The keypad assembles the series data into a parallel 8 bit scan code and generates interrupt request to the interrupt logic.

The system software follows specification protocol with keypad and microprocessor for data transfer and control sequences. The keypad interface consists of the following function sections-

**Figure 11.1 – Keypad Design Module**




- Serial to parallel converter
- Interrupt generation logic
- Scan code port

The series to parallel converter is enabled by software (Keil C). Once the scan code assembled interrupt request is generated and this interrupt request freezes the shift register. The interrupt series service routine also clears the shift register, so as to prepare the next scan code.

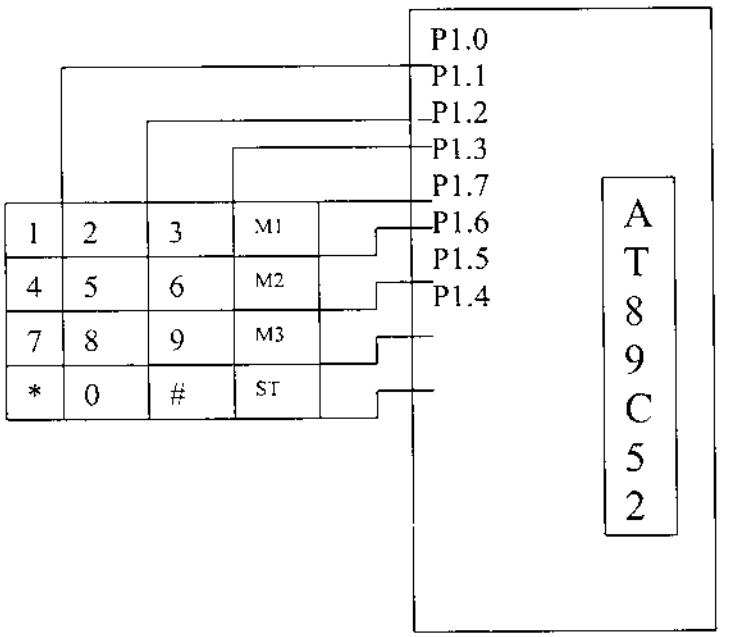
## 11.5 BIDIRECTIONAL COMMUNICATION

The keypad and microprocessor transmits the scan code to the PC systems over a serial interface. There are two bidirectional lines between the keyboard and PC motherboard. One line is the serial data bit and the other line is the clock signal.

The four signals between the PC and the keyboard are as follows-

- +5V DC
- Ground
- Data
- Clock

**Figure 11.2 – Keypad Interfacing with Microcontroller**




## 11.6 FEATURES

- Keypads are organized in a matrix of rows and columns.
- The CPU accesses both the rows and columns through ports.
- Only when a key is pressed a row and column makes a contact else there is no connection between them.

# SOFTWARE OVERVIEW

# 12. KEIL C51 COMPILER BASICS

The Keil C51 compiler has been written to allow the C programmers to get code running quickly on 8051 systems with little or no learning curve. However, to get the best from it, some appreciation of the underlying hardware is desirable. The most basic decision to be made is which memory model to use.

## 12.1 8051 MEMORY CONFIGURATIONS

### PHYSICAL LOCATION OF THE MEMORY SPACES

Perhaps the most initially confusing thing about the 8051 is that there are three different memory spaces, all of which start at the same address. Other microcontrollers, such as the 68HC11, have a single Von Neumann memory configuration, where memory areas are located at the sequential address; regardless of in what device they physically exist.

Within the CPU there is one such, The DATA on-chip RAM. This starts at D: 00(the 'D:' prefix implies DATA segment) and ends at 07Fh (127 decimal) .This RAM can be used for the program variables. It is directly addressable, so that the instructions like 'MOV A, X' are usable. Above 80h the special function registers are located, which are again directly addressable. However, a second memory area exists between 80h and 0FFh which is only indirectly addressable and is prefixed by I: and known as IDATA. It is only accessible via indirect addressing (MOV A, @RI) and effectively overlays the directly addressable SFR area. This constitutes an

extended on-chip RAM area and was added to the ordinary 8051 design when the 8052 appeared. As it is only indirectly addressable, it is best left for stack use, which is by definition, always indirectly addressed via the stack pointer SP. Just to confuse things, the normal directly addressable RAM from 0-80h can also be indirectly addressed by the MOV A,@Ri instruction.

## 12.2 POSSIBLE MEMORY MODELS

With a microcontroller like the 8051, the first decision is which memory model to use. Whereas the PC programmer chooses between TINY, SMALL, MEDIUM, COMPACT, LARGE and HUGE to control how the processor segmentation of the RAM is to used (overcome) , the 8051 user has to decide where the program and data are to reside.

C51 currently supports the following memory configurations

**ROM**

Currently the largest single object file that can be produced is 64K, although up to 1MB can be supported with the BANKED model described below. All compiler output to be directed to EPROM/ROM, constants, look-up tables etc., should be declared as 'code'.

**RAM**

There are three memory models, SMALL, COMPACT and LARGE

**SMALL**

All the variables and parameter-passing segments will be placed in the 8051's internal memory.

**COMPACT**

Variables are stored in paged memory addressed by port 0 and 2. Indirect addressing opposed is used. On-chip registers are still used for locals and parameters.

**LARGE**

Variables are placed in the external memory addressed by @ DPTR. On-Chip registers are still used for locals and parameters.

**BANKED**

Code can occupy up to1MB by using either CPU port pins or memory-mapped latches to page memory above 0xffff. Within each 64KB memory block a COMMON area must be set aside for C library code. Inter-bank function calls are possible.

## 12.3 STEP BY STEP PROCEDURE FOR BUILDING NEW PROJECT IN 'KEIL C' COMPILER

1. Create working subdirectory normally under c:\temp directory for your program files (.a51 files, etc.)
2. Connect EMAC board to COM2 PC port.
3. Go to EE Applications.
4. Go to KEIL PK51
5. Select uVision-51
6. Use the Keil editor for writing your source files: errors are highlighted in your files when you compile or assembly your program
7. Go to project
8. Select NEW Project
9. Choose c:\Temp\your_dir

10. Enter name of the project as .prj file and select OK. This will pop up a window to add source files.
11. Select ADD. This will pop up a window to add files.
12. Select files to be added to project from c:\temp directory.
13. Select ADD after each file to be added
14. Select Close
15. Select Open All.
16. Select SAVE.
17. Select Close
18. Click in the main Window
19. Go to Options
20. Select BL51 Code Banking Linker, then size/Location
21. At Code Address , Enter 8000 (starting location of EMAC program), then OK
22. Go to Project
23. Select Make: Build Project.
24. If errors, fix modules (save them)
25. Click on main window
26. Go to Project
27. Select Make: Update Project. Note a handy feature: no reason to reassemble good files again!
28. A *.m51 file is created after a good make. This is handy for debugging. Open with editor
29. Go to Run
30. Select dScope Debugger

## 12.4 DSCOPE DEBUGGER: (RUNS LINKED PROGRAM)

1. Go to view
2. Select Toolbar, Status Bar, Register Window, Command Window, Debug Window, Memory Window and Toolbox
3. Go to CPU DLL selection and choose mon51.dll
4. Go to file
5. Select Load Object File. This will pop up a window to select an object file: your project file without an extension
6. Go to command window (bottom of screen)
7. Type $=starting address of program and hit ENTER, example $=8000H
8. Select Go or Step Into to execute your program
9. To terminate your program (without breakpoints) you must reset to EMAC board (red push-button switch)
10. See help for information on helpful debugging tools: Watch-points, breakpoints, etc.

## 12.5 LOADING A PROGRAM INTO THE CONTROLLER BOARD

Once the program has been compiled into a HEX file, it needs to download it into the controller board. For this step we need to use a different program called WINISP. It is probably easiest to minimize Keil C at this time.

To load the program into the controller board, we need to follow these steps:

1. The controller board needs to be plugged into the computer. Switch it on (the red LED will come on) and switch it into program mode (green LED will come on). Then push the reset button.
2. The WINISP program can be launched by going to Start Menu, then Programs, then Winisp, then WINISP (or use the shortcut if there is one)
3. Set the WINISP parameters. For Chip, choose 89C51RD2. For Port, choose Com 1 (or whichever com port the board is connected to)
4. Under Misc, click on read. Vector should be FC and Status should be 0. If they are not, change them and click on Write. If they are correct, you do not need to do anything
5. Before we load our program, we need to erase any program that is already in the processor. Do this by clicking on Erase Blocks. In the dialog box, click on the first 8K block (our programs are short), then ERASE! The window should close when its done
6. Now we load the program into WINISP by clicking on Load File. Find the directory with your program, select example. Hex, and click OK.
7. Finally, click on Program Part. This will download the file into the processor

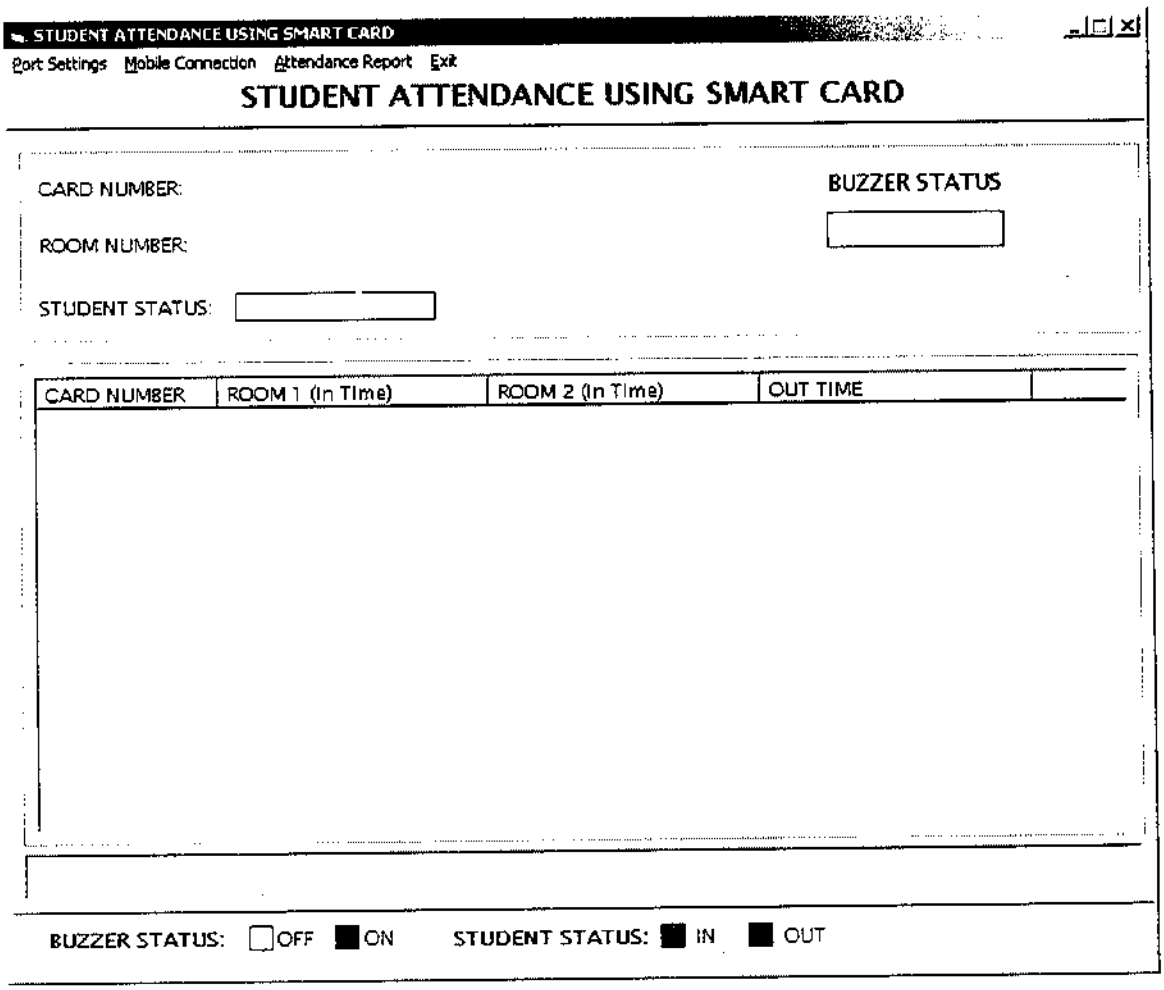
To have the controller board run the program, disconnect the cord to the computer; press the reset button on the back of the display, then switch the board into run mode (the green LED will turn off) and press the reset button.

## 13.1 VISUAL BASIC

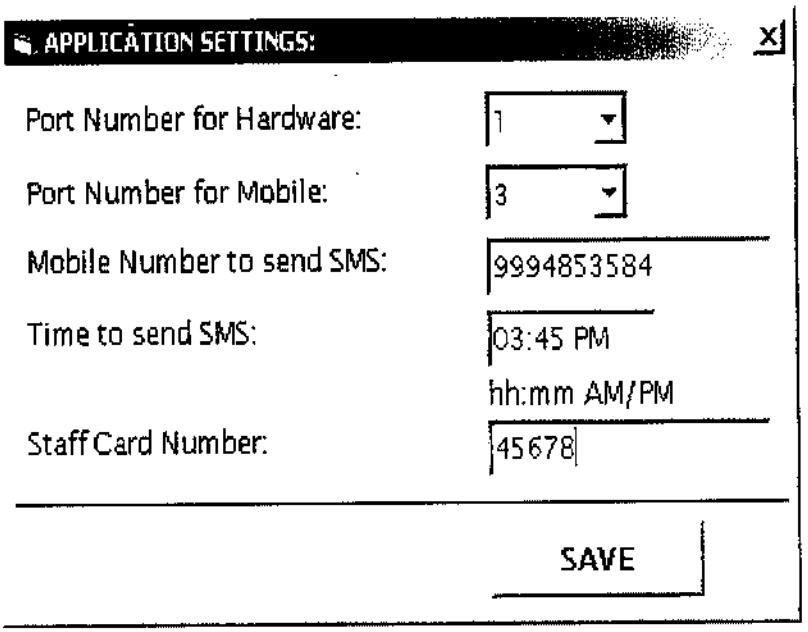
The output of the microcontroller is displayed through PC and LCD. For viewing the results through PC,VISUAL BASIC program is used. A database of the staffs and students are stored and when the password is typed correctly it compares with the value in the database and displays the result.

The COM port is connected to the reader through MAX 232 . The data communication is checked through Hyper Terminal.

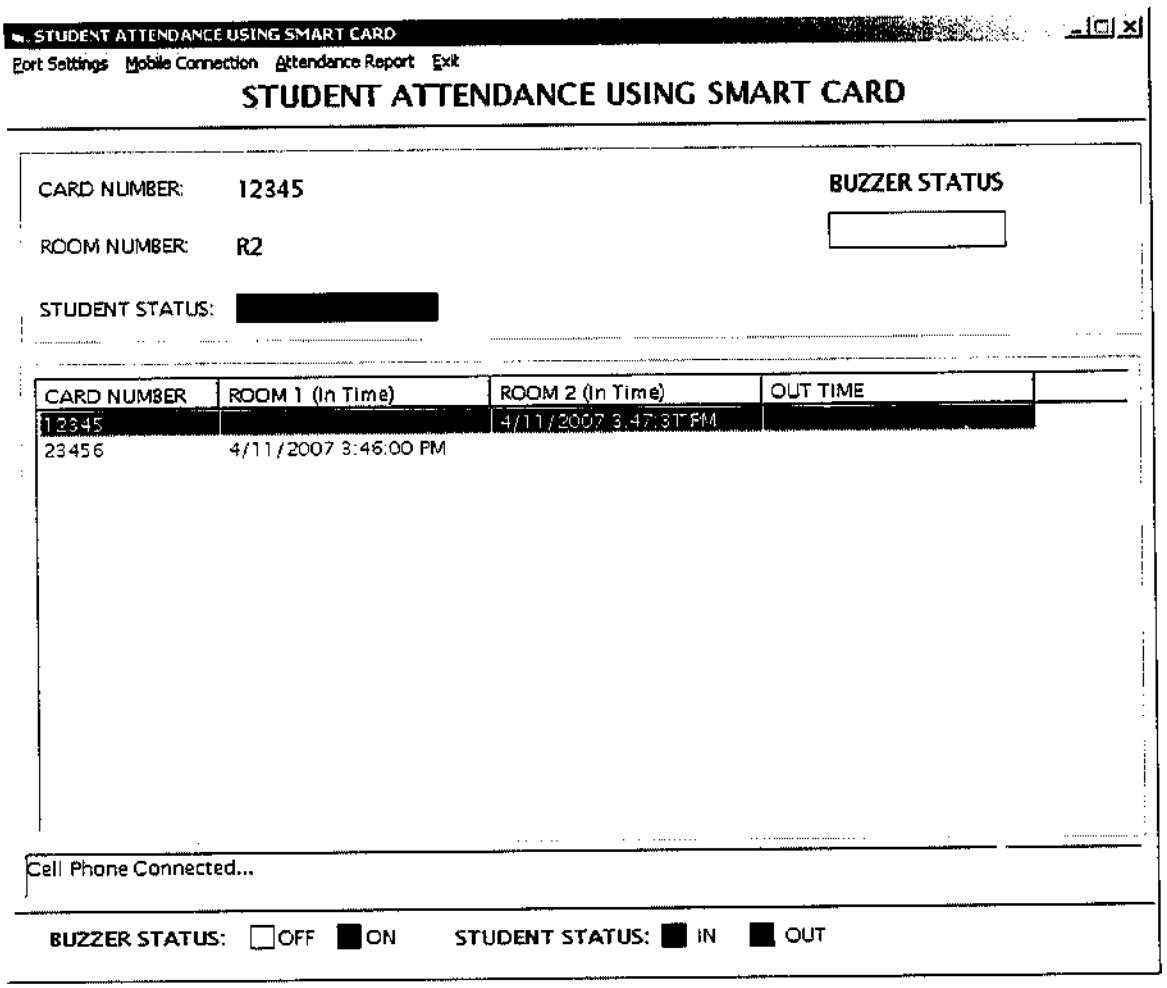
## 13.2 INITIAL STAGE



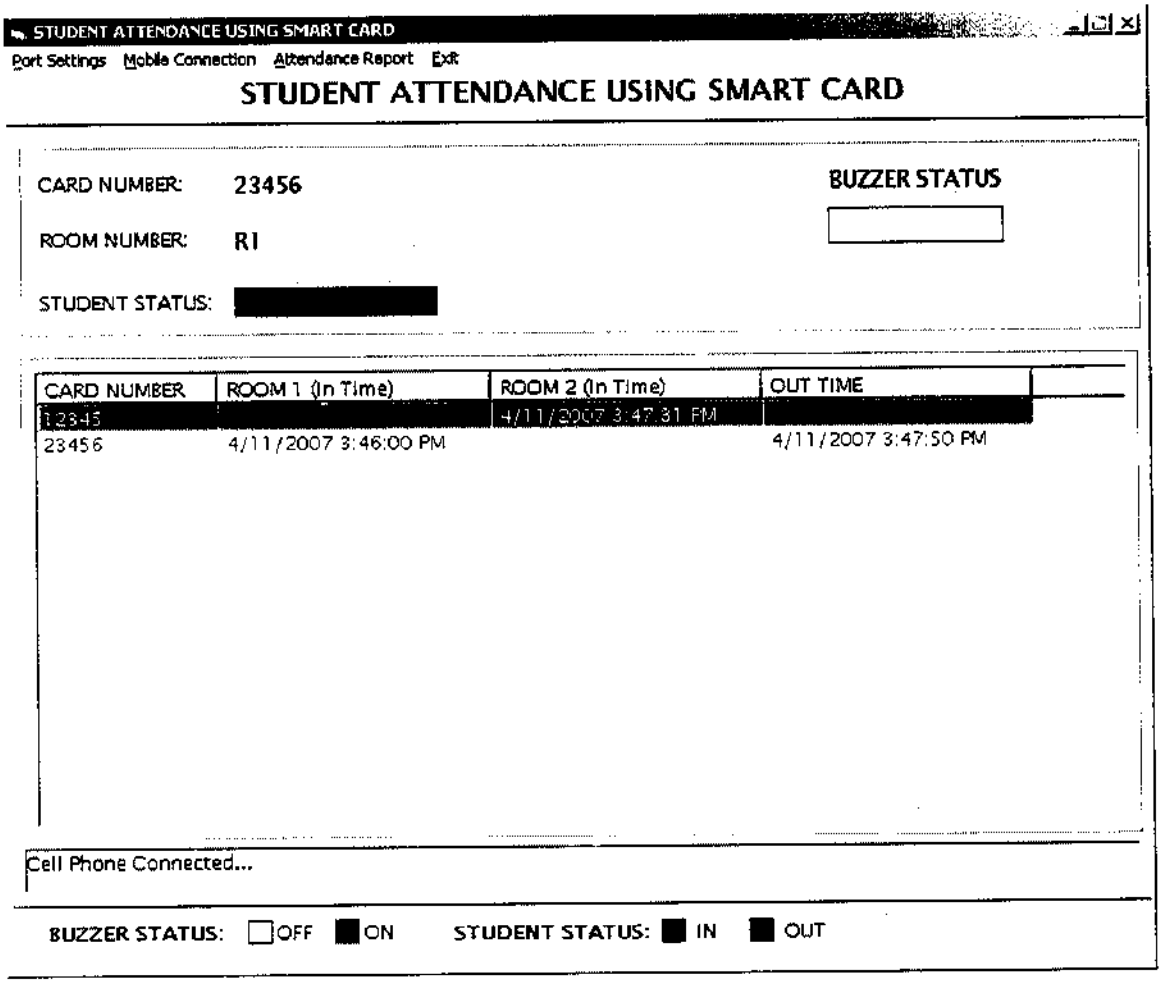
13.3 PORT INITIALISATION

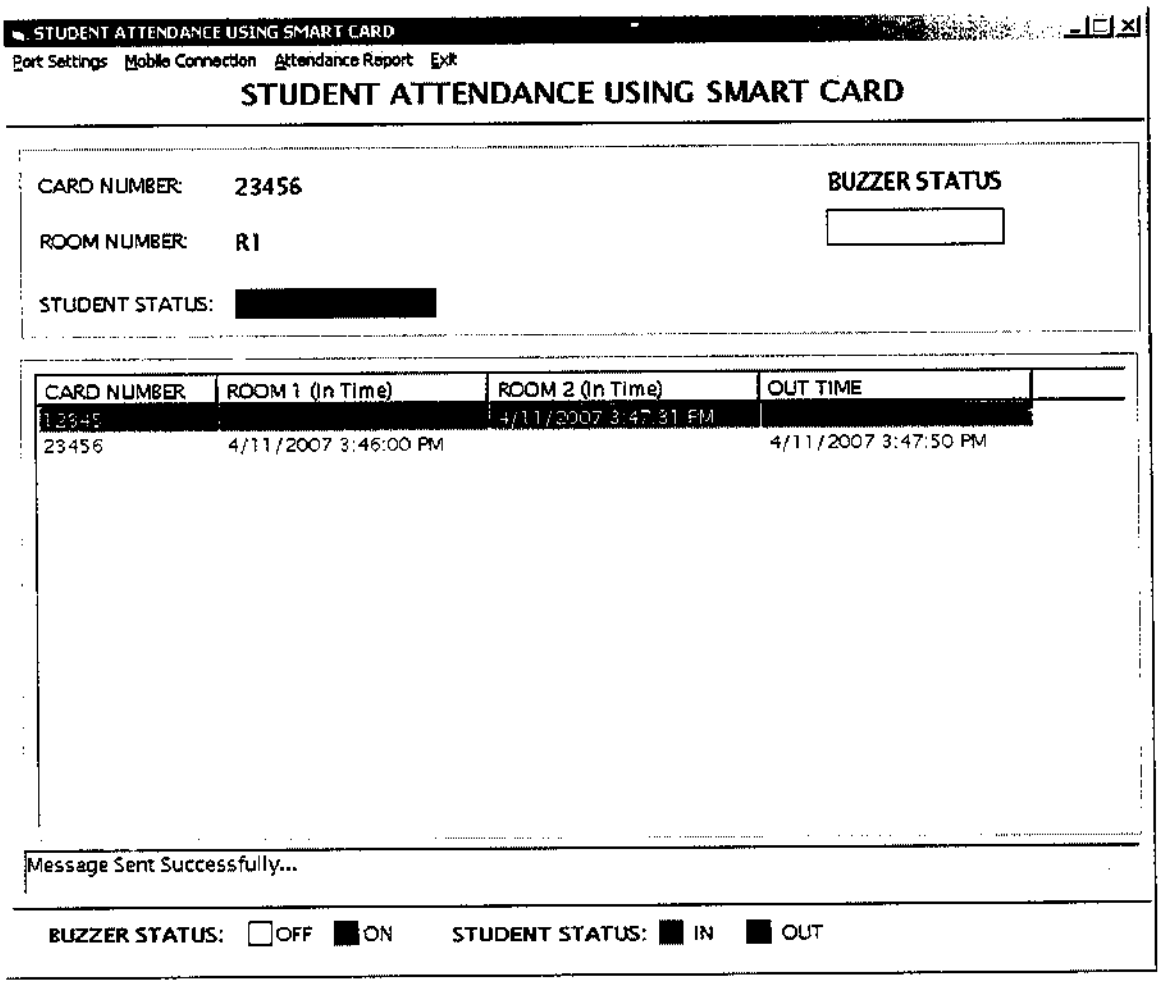


# 13.4 CELL PHONE CONNECTION AND STUDENT ENTRY

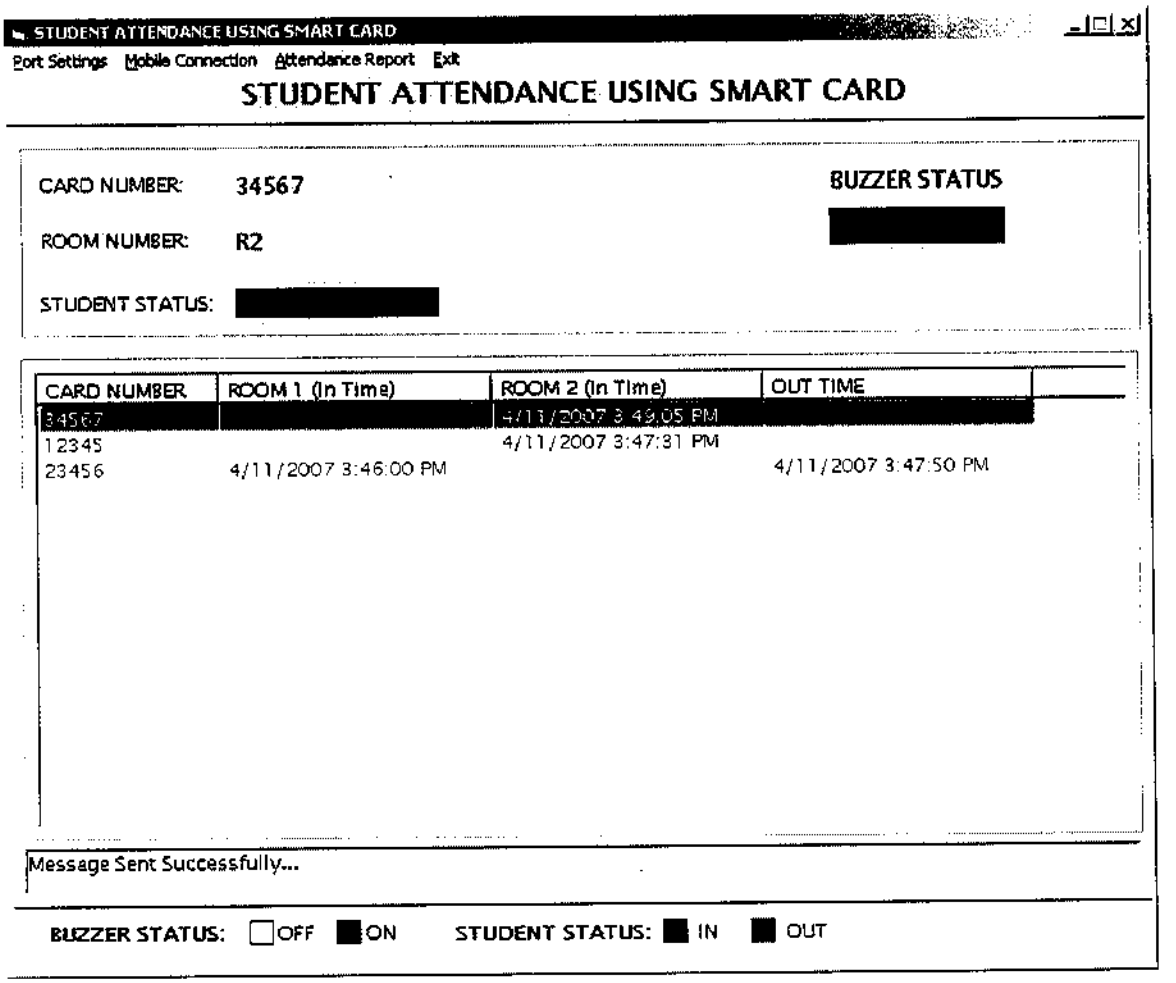


13.5 DATABASE UPDATION AND STUDENT EXIT

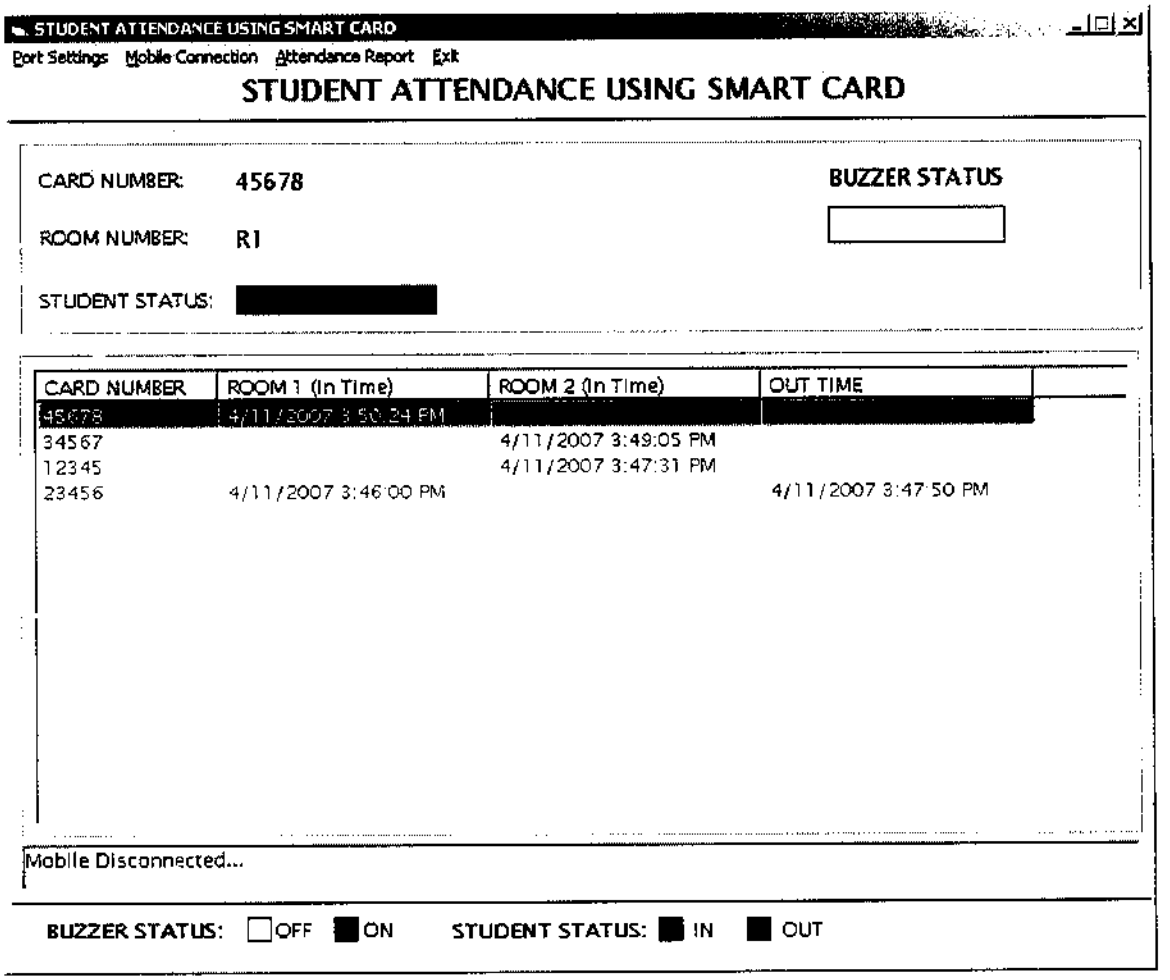



STUDENT ATTENDANCE USING SMART CARD
Port Settings
Mobile Connection
Attendance Report
Exit
STUDENT ATTENDANCE USING SMART CARD
CARD NUMBER:
23456
ROOM NUMBER:
R1
STUDENT STATUS:
BUZZER STATUS
CARD NUMBER
ROOM 1 (In Time)
ROOM 2 (In Time)
OUT TIME
23456
4/11/2007 3:46:00 PM
4/11/2007 3:47:50 PM
Message Sent Successfully...
BUZZER STATUS:
OFF
ON
STUDENT STATUS:
IN
OUT

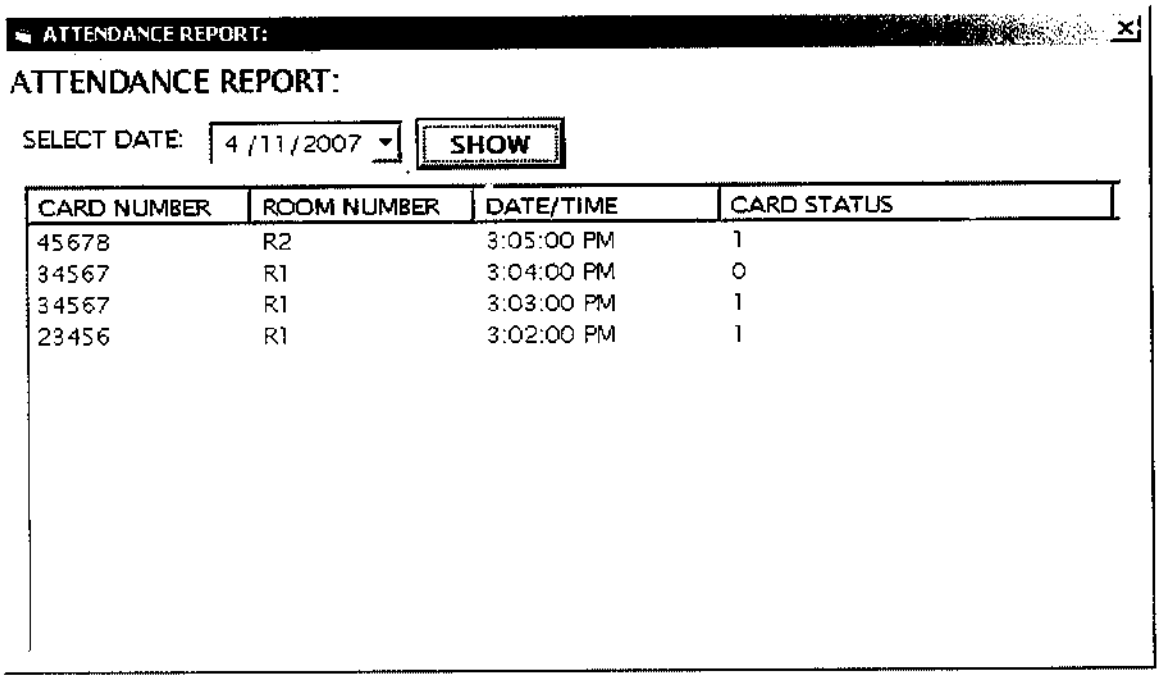
## 13.7 DETECTION OF MISUSE OF THE CARD



## 13.8 MOBILE DISCONNECTION



13.9 DATABASE



# 14. CODING

## 14.1 MAIN CODE

```
#include "student.h"

#include <stdio.h>

#include<string.h>

void serial_init();

void enter_pass();

void sreboot();

void enter();

void exit();

sbit buzzer=P2^2;
sbit sensor=P3^2;

bit
room1_flag,room2_flag,R1_card1_in,R2_card1_in,R1_card2_in,R2_card2_i
n,R1_card3_in,R2_card3_in,R1_card4_in,R2_card4_in;
bit card1,card2,card3,card4,card4,f1_buzzer;
```

```
unsigned char
a[5],i,b[10],c[4],j,k=0,z,Room1[5],Room2[5],j,k,count=0,tx,temp[5];

void isr()interrupt 0
 {
        count++;
        if(count>=3)
            {
             count=0;
             f1_buzzer=1;
            }
 }


 void isr1()interrupt 4
 {

 }

 void serial_init()              //function for serial communication
 {
     EA = 0;
         SCON = 0x40;            //for Tx (serial control)
             TMOD = 0x20;
         TH1  = 0xFD;
         TR1  = 1;               //T1 on
     EX0  = 1;                   //enabling EXT ipt0
```

```
        IT0  = 1;                    //enabling hw ipt0
        ES   = 1;          // enabling si
      EA   = 0;                  //enabling GIE
}

void enter_pass()          //entering password
{
while(1)
 {
   key = get_key();
      if(key != 0 )
      {
     lcd_display('*' , 20+i);
            a[i] = key;
            key = 0;
            if(i++ >= 4)
            {
              i=0;
                  key = 0;

                  break;
            }
        delay(65000);
      }
  }
}
```

```
void check_password()          //function for checking password
{
 while(1)
      {
   if( (strncmp(Room1,a,5) == 0 ) || (strncmp(Room2,a,5)==0))
        {
              clear_lcd();
              lcd_display1("Password ok",0);
              delay(65000);
              delay(65000);
              delay(65000);
              break;
    }

            else
                  {
                   clear_lcd();
                   lcd_display1("Password wrong",0);
           delay(65000);
                   delay(65000);
        delay(65000);
                   clear_lcd();
        lcd_display1("E Ur password",0);
                   enter_pass();
                  }
   }
}
```

```
void enter()
{
        clear_lcd();
   lcd_display1("INSERT CARD",0);
        while(1)
        {
         key=get_key();
         if(key=='D')
          {
           lcd_display1("    Attendance ",0);
       lcd_display1("      System    ",16);
                delay(65000);
                delay(65000);
                delay(65000);
                break;
          }

         for(j=0x00;j<0x05;j++)                    //reading datas
from card(Room1)
           {
             read_data(j);
                 Room1[j] = data_in;
               }

               for(k=0x00;k<0x05;k++)                    //reading
datas from card(Room2)
```

```
    {
      read_data1(k);
          Room2[k] = data_in1;
}
    lcd_display1(" ",16);
        lcd_display1(" ",26);
delay(65000);
    if(Room1[0]=='1')
        {
           data_in=0;
               R1_card1_in=1;
               card1=1;
               room1_flag=1;
               room2_flag=0;
           clear_lcd();
        }
if(Room2[0]=='1')
        {
           data_in1=0;
               R2_card1_in=1;
               card1=1;
               room2_flag=1;
               room1_flag=0;
               clear_lcd();
        }
        if(Room1[0]=='2')
        {
```

```
            data_in=0;
                R1_card2_in=1;
                card2=1;
                room1_flag=1;
            room2_flag=0;
            clear_lcd();
        }
if(Room2[0]=='2')
        {
            data_in1=0;
                R2_card2_in=1;
                card2=1;
                room2_flag=1;
                room1_flag=0;
                clear_lcd();
        }

        if(Room1[0]=='3')
        {
            data_in=0;
                R1_card3_in=1;
                card3=1;
                room1_flag=1;
            room2_flag=0;
            clear_lcd();
        }
if(Room2[0]=='3')
```

```
            data_in1=0;
                R2_card3_in=1;
                card3=1;
                room2_flag=1;
                room1_flag=0;
                clear_lcd();
        }

        if(Room1[0]=='4')
        {
            data_in=0;
                R1_card4_in=1;
                card4=1;
                room1_flag=1;
            room2_flag=0;
            clear_lcd();
        }
if(Room2[0]=='4')
        {
            data_in1=0;
                R2_card4_in=1;
                card4=1;
                room2_flag=1;
                room1_flag=0;
                clear_lcd();
        }
```

```
 if(room1_flag | room2_flag)
       {
          lcd_display1("E Ur password",0);
     enter_pass();
 check_password();
                EA=1;
                delay(30000);
                delay(30000);
                delay(30000);
     EA=0;


// wait for some time to enter  //
        if(f1_buzzer)
    {
          f1_buzzer=0;
              buzzer=0;              //on
              delay(20000);
              buzzer=1;                  //off
              delay(10000);
              buzzer=0;
              delay(20000);
              buzzer=1;
              delay(10000);
```

```
            EA=1;
            TI=1;
            printf("<x>");
            TI=0;
            EA=0;
            buzzer=0;
            delay(20000);
            buzzer=1;
            delay(10000);
            buzzer=0;
            delay(20000);
            buzzer=1;

            delay(10000);
}
        if(room1_flag )
            {

              room1_flag = 0;
              lcd_display1("entered in R1",0);
                  delay(65000);
                  delay(65000);
                  EA=1;
                  TI=1;
                  printf("<R1");
              for(tx=0;tx<5;tx++)
                  printf("%c",Room1[tx]);
```

```
                printf("1>");
                TI=0;
                EA=0;
                break;
            }

        if(room2_flag)
            {
             room2_flag=0;
          lcd_display1("entered in Room2",0);
                delay(65000);
                delay(65000);
                EA=1;
                TI=1;
                printf("<R2");
              for(tx=0;tx<5;tx++)
                printf("%c",Room2[tx]);
                printf("1>");
                TI=0;
                EA=0;
                delay(65000);
         break;
            }
        }


}
```

```
}


void exit()
 {
       clear_lcd();
  lcd_display1("Insert card",0);
       while(1)
       {
        key=get_key();
        if(key=='D')
         {
          lcd_display1("   Attendance ",0);
      lcd_display1("     System    ",16);
              delay(65000);
              delay(65000);
              delay(65000);
              break;
         }
        read_data(0x00);        // first room
        read_data1(0x00);                       //second room
        if(data_in == '1'&& R1_card1_in==1 && card1==1)
              {
               data_in=0;
               R1_card1_in=0;
               card1=0;
               clear_lcd();
```

```
        lcd_display1("Out from R1",0);
        delay(65000);
        EA=1;
        TI=1;
        printf("<R1123450>");
    TI=0;
        EA=0;
        delay(65000);
        break;
       }
       if(data_in1 =='1'&&  R2_card1_in==1 && card1==1)
       {
        data_in=0;
        R2_card1_in=0;
        card1=0;
        clear_lcd();
        lcd_display1("Out from R2",0);
        delay(65000);
        EA=1;
        TI=1;
        printf("<R2123450>");
        TI=0;
        EA=0;
        delay(65000);
        break;
}
  if(data_in == '2'&&  R1_card2_in==1 && card2==1)
```

```
        {
         data_in=0;
         R1_card2_in=0;
         card2=0;
         clear_lcd();
         lcd_display1("Out from R1",0);
         EA=1;
         TI=1;
         printf("<R1234560>");
         TI=0;
         EA=0;
         delay(65000);
    break;
        }

if(data_in1 =='2'&&  R2_card2_in==1 && card2==1)
        {
         data_in=0;
         R2_card2_in=0;
         card2=0;
         clear_lcd();
         lcd_display1("Out from R2",0);
         EA=1;
         TI=1;
         printf("<R2234560>");
         TI=0;
         EA=0;
```

```
        delay(65000);
        break;
}

if(data_in == '3'&&  R1_card3_in==1 && card3==1)
        {
         data_in=0;
         R1_card3_in=0;
         card3=0;
         clear_lcd();
         lcd_display1("Out from R1",0);
         EA=1;
         TI=1;
         printf("<R1345670>");
         TI=0;
         EA=0;
         delay(65000);
    break;
         }

 if(data_in1 =='3'&&  R2_card3_in==1 && card3==1)
        {
         data_in=0;
         R2_card3_in=0;
         card3=0;
         clear_lcd();
         lcd_display1("Out from R2",0);
```

```
        EA=1;
        TI=1;
        printf("<R2345670>");
        TI=0;
        EA=0;
        delay(65000);
        break;
}
      if(data_in == '4'&&  R1_card4_in==1 && card4==1)
      {
       data_in=0;
       R1_card4_in=0;
       card4=0;
       clear_lcd();
       lcd_display1("Out from R1",0);
       EA=1;
       TI=1;
       printf("<R1456780>");
       TI=0;
       EA=0;
       delay(65000);
    break;
      }

  if(data_in1 =='4'&&  R2_card4_in==1 && card4==1)
      {
       data_in=0;
```

```
                R2_card4_in=0;
                card4=0;
                clear_lcd();
                lcd_display1("Out from R2",0);
                EA=1;
                TI=1;
                printf("<R2456780>");
                TI=0;
                EA=0;
                delay(65000);
                break;
        }


    }
}

void main()
{
  lcd_init();
  serial_init();
  lcd_display1("   Attendance ",0);
  lcd_display1("    System   ",16);
  delay(65000);
  delay(65000);
  while(1)
   {
```

```
        key=get_key();
    switch(key)
          {
          case 'A':
                              key = 0;
                              enter();
                        break;
               case 'B':
                              key = 0;
                              exit();
                              break;
               default:
                              lcd_display1("M1 ->Enter      ",0);
                              lcd_display1("M2->Exit        ",16);
                        break;

               }
    }
  }
```

14.2 DISPLAY CODE

```
#include "student.h"

#define display P0
```

```
sbit rs= P2^7;            //register select
sbit rw=P2^6;             //read write
sbit en=  P2^5;           //enable
unsigned char code loc[32]=  {0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,

0x88,0x89,0x8a,0x8b,0x8c,0x8d,0x8e,0x8f,

0xc0,0xc1,0xc2,0xc3,0xc4,0xc5,0xc6,0xc7,

0xc8,0xc9,0xca,0xcb,0xcc,0xcd,0xce,0xcf};



void lcd_enable()
{
        en=0;
        delay(50);
        en=1;
        delay(50);
        en=0;
}

void lcd_init()
  {
    rw=0;
    delay(250);
         cmd_code(0x30);              //Power on delay  (3 times)
    cmd_code(0x30);
```

```
    cmd_code(0x06);                 //Entry mode set
    cmd_code(0x0c);                         // Display on

}

void cmd_code(unsigned char cmd)
 {
  rs=0;
  display=cmd;
  lcd_enable();
 }

 void data_code(unsigned char dat)
 {
       rs=1;
  display=dat;
  lcd_enable();


 }
 void lcd_display(unsigned char m,unsigned char l)
 {
    cmd_code(loc[l]);
    rs=1;
    display=m;
```

```
    lcd_enable();
}

void clear_lcd()
 {
  rs=0;
       display=0x01;
       lcd_enable();
 }

 void lcd_display1(unsigned char *m,unsigned char l)
 {
  cmd_code(loc[l]);
       while(*m)
         {
     rs=1;
      display=*m++;
          lcd_enable();
         }

}

 void  delay(unsigned int d)
{
    unsigned int i;
    for(i=0;i<d;i++);
```

}

14.3 KEYPAD CODE

```
#include "student.h"
#define keyscan P1


unsigned char key;

unsigned char code key_table[16] = {"123A456B789C*0#D"};

unsigned char code ROW[4] = {0x7f,0xbf,0xdf,0xef};

unsigned char code COL[16]= {0x77,0x7b,0x7d,0x7e,
                             0xb7,0xbb,0xbd,0xbe,
                             0xd7,0xdb,0xdd,0xde,
                             0xe7,0xeb,0xed,0xee};

unsigned char get_key(void)
{
   unsigned char i,j,n;
   n = 0;
   for(i=0;i<4;i++)
         {
      keyscan = ROW[i];
```

```
      for(j=n;j<n+4;j++)
        {
          if(keyscan == COL[j])
           return(key_table[j]);
        }
      n += 4;
   }
   return(0);
}
```

14.4 I2C CODE

```
#include "student.h"

 #define      TRUE                    0x01              // Value
representing TRUE
 #define      FALSE                   0x00              // Value
representing FALSE
 #define      HIGH          0x01              // Value representing ON
 #define      LOW                     0x00              // Value
representing OFF

 #define      DELAY_BLINK             1500              // Value for delay
time - blink

 sbit  SDATA          = P3 ^ 7;              // Serial data
 sbit  SCLK         = P3 ^ 6;              // Serial clock
```

```
unsigned char data_in;

void write_data(unsigned char lowerbyte,unsigned char data1)
 {

      i2c_start();                    // Send I2C Start Transfer
      i2c_write(0xa0);                // Send identifier I2C address -
Write
      i2c_write(0x00);                          // Send control
byte to device
      i2c_write(lowerbyte);           // Send voltage to DAC
     i2c_write(data1);                // Send voltage to DAC
      i2c_stop();                     // Send I2C Stop Transfer
 }

unsigned char read_data(unsigned char lowerbyte)
{

      i2c_start();                    // Send I2C Start Transfer
      i2c_write(0xa0);                // Send identifier I2C address -
Write
      i2c_write(0x00);                          // Send control
byte to device
      i2c_write(lowerbyte);           // Send voltage to DAC

      i2c_start();                    // Send I2C Start Transfer
```

```
        i2c_write(0xa1);
                                // Send identifier I2C address - Read
          data_in = i2c_read();                                  // Read the channel
number
        i2c_stop();                                  // Send I2C Stop Transfer

        return data_in;

}

void i2c_start (void)
{
        SDATA = HIGH;                                                  // Set data
line high
        delay_time(50);
        SCLK = HIGH;                                                   // Set clock
line high
        delay_time(50);
        SDATA = LOW;                                                   // Set data
line low (START SIGNAL)
        delay_time(50);
        SCLK = LOW;                                                        // Set
clock line low
}

void i2c_stop (void)
 {
```

```
        unsigned char input_var;

        SCLK = LOW;                                          // Set
clock line low
        delay_time(50);
        SDATA = LOW;                                         // Set data
line low
        delay_time(50);
        SCLK = HIGH;                                         // Set clock
line high
        delay_time(50);
        SDATA = HIGH;                                        // Set data
line high (STOP SIGNAL)
        input_var = SDATA;                                   // Put port
pin into HiZ
 }

void i2c_write (unsigned char output_data)
{
        unsigned char index;

        for(index = 0; index < 8; index++)          // Send 8 bits to the I2C
Bus
        {
                                        // Output the data bit to the I2C Bus
                SDATA = ((output_data & 0x80) ? 1 : 0);
```

```
            output_data <<= 1;                  // Shift the byte by one
bit
            SCLK = HIGH;                        // Clock the data into the
I2C Bus
            delay_time(50);
            SCLK = LOW;
        }

        index = SDATA;                                  // Put data
pin into read mode
        SCLK = HIGH;                            // Clock the ACK from
the I2C Bus
        delay_time(50);
        SCLK = LOW;
}

unsigned char i2c_read (void)
{
        unsigned char index, input_data;

        index = SDATA;                                  // Put data
pin into read mode

        input_data = 0x00;
        for(index = 0; index < 8; index++)              // Send 8 bits to the
I2C Bus
        {
```

```
            input_data <<= 1;                    // Shift the byte by
one bit
            SCLK = HIGH;                         // Clock the data
into the I2C Bus
            input_data |= SDATA;                 // Input the data
from the I2C Bus
            SCLK = LOW;
        }
  return input_data;
}


void delay_time (unsigned int time_end)
{
        unsigned int index;
        for (index = 0; index < time_end; index++);
}

#include "student.h"

 #define      TRUE                0x01          // Value
representing TRUE
 #define      FALSE               0x00          // Value
representing FALSE
 #define      HIGH          0x01          // Value representing ON
 #define      LOW                 0x00          // Value
representing OFF
```

```
#define     DELAY_BLINK          1500              // Value for delay
time - blink

sbit  SDATA1        = P2 ^ 4;              // Serial data
sbit  SCLK1             = P2 ^ 3;                  // Serial clock

unsigned char data_in1;


void write_data1(unsigned char lowerbyte,unsigned char data1)
 {

      i2c_start1();                        // Send I2C Start Transfer
      i2c_write1(0xa0);                        // Send identifier I2C
address - Write
      i2c_write1(0x00);                                // Send control
byte to device
      i2c_write1(lowerbyte);               // Send voltage to DAC
     i2c_write1(data1);                    // Send voltage to DAC
      i2c_stop1();                         // Send I2C Stop Transfer
 }

unsigned char read_data1(unsigned char lowerbyte)
{

      i2c_start1();                        // Send I2C Start Transfer
```

```
        i2c_write1(0xa0);                    // Send identifier I2C
address - Write
        i2c_write1(0x00);                    // Send control
byte to device
        i2c_write1(lowerbyte);               // Send voltage to DAC

        i2c_start1();                        // Send I2C Start Transfer
        i2c_write1(0xa1);
                        // Send identifier I2C address - Read
          data_in1= i2c_read1();             // Read the channel
number
        i2c_stop1();                         // Send I2C Stop Transfer

        return data_in1;

}

void i2c_start1 (void)
{
        SDATA1 = HIGH;                       // Set data
line high
        delay_time(50);
        SCLK1 = HIGH;                        // Set clock
line high
        delay_time(50);
        SDATA1 = LOW;                        // Set data
line low (START SIGNAL)
```

```
        delay_time(50);
        SCLK1 = LOW;                                    // Set
clock line low
}

void i2c_stop1 (void)
 {
        unsigned char input_var;

        SCLK1 = LOW;                                    // Set
clock line low
        delay_time(50);
        SDATA1 = LOW;                                   // Set data
line low
        delay_time(50);
        SCLK1 = HIGH;                                   // Set clock
line high
        delay_time(50);
        SDATA1 = HIGH;                                  // Set data
line high (STOP SIGNAL)
        input_var = SDATA1;                             // Put port
pin into HiZ
 }

void i2c_write1 (unsigned char output_data)
{
        unsigned char index;
```

```
    for(index = 0; index < 8; index++)          // Send 8 bits to the I2C
Bus
    {
                                        // Output the data bit to the I2C Bus
        SDATA1 = ((output_data & 0x80) ? 1 : 0);
       output_data  <<= 1;                      // Shift the byte by one
bit
        SCLK1 = HIGH;                           // Clock the data into the
I2C Bus
        delay_time(50);
        SCLK1 = LOW;
    }

    index = SDATA1;                                     // Put data
pin into read mode
    SCLK1 = HIGH;                           // Clock the ACK from
the I2C Bus
    delay_time(50);
    SCLK1 = LOW;
}

unsigned char i2c_read1 (void)
{
    unsigned char index, input_data;
```

```
        index = SDATA1;                                    // Put data
pin into read mode

        input_data = 0x00;
        for(index = 0; index < 8; index++)                 // Send 8 bits to the
I2C Bus
        {
                input_data <<= 1;                          // Shift the byte by
one bit
                SCLK1 = HIGH;                              // Clock the data
into the I2C Bus
                input_data |= SDATA1;                      // Input the data
from the I2C Bus
                SCLK1 = LOW;
        }
  return input_data;
}


void delay_time1 (unsigned int time_end)
{
        unsigned int index;
        for (index = 0; index < time_end; index++);
}
```

# 15. CONCLUSION

Our project minimizes the amount of paper work done in monitoring the attendance of the students. The smart cards are cheaper than the RF ID's. About 256 rooms can be connected using a single reader. Since our project has been done using 89C52 micro controller, from which readers and writers are designed will short the cost of designing the products. This project can also be used to access the seating arrangement details during examination.

# 16. APPENDIX - DATASHEETS

## tures

v-Voltage and Standard-Voltage Operation
5.0 ($V_{CC}$ = 4.5V to 5.5V)
2.7 ($V_{CC}$ = 2.7V to 5.5V)
2.5 ($V_{CC}$ = 2.5V to 5.5V)
1.8 ($V_{CC}$ = 1.8V to 5.5V)
v-Power Devices ($I_{SB}$ = 2 µA @ 5.5V) Available
rnally Organized 4096 x 8, 8192 x 8
ire Serial Interface
mitt Trigger, Filtered Inputs for Noise Suppression
irectional Data Transfer Protocol
kHz (1.8V, 2.5V, 2.7V) and 400 kHz (5V) Compatibility
te Protect Pin for Hardware Data Protection
Byte Page Write Mode (Partial Page Writes Allowed)
f-Timed Write Cycle (10 ms max)
h Reliability
Endurance: 1 Million Write Cycles
Data Retention: 100 Years
ESD Protection: >3,000V
omotive Grade and Extended Temperature Devices Available
in JEDEC PDIP, 8-Pin and 14-Pin JEDEC SOIC, 8-Pin EIAJ SOIC,
8-pin TSSOP Packages

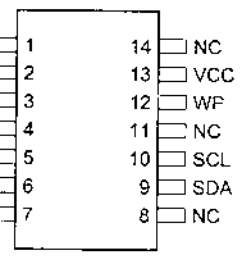
## cription

T24C32/64 provides 32,768/65,536 bits of serial electrically erasable and pro-mable read only memory (EEPROM) organized as 4096/8192 words of 8 bits The device's cascadable feature allows up to 8 devices to share a common 2-us. The device is optimized for use in many industrial and commercial applica-where low power and low voltage operation are essential. The AT24C32/64 is ble in space saving 8-pin JEDEC PDIP, 8-pin and 14-pin JEDEC SOIC, 8-pin SOIC, and 8-pin TSSOP packages and is accessed via a 2-wire serial interface. dition, the entire family is available in 5.0V (4.5V to 5.5V), 2.7V (2.7V to 5.5V), 2.5V to 5.5V) and 1.8V (1.8V to 5.5V) versions.

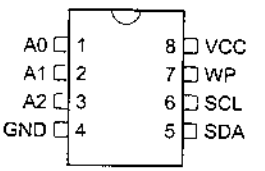
## Configurations

| Name | Function |
|---|---|
| A2 | Address Inputs |
| | Serial Data |
| | Serial Clock Input |
| | Write Protect |

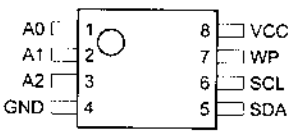
14-Pin SOIC



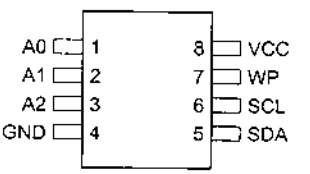

8-Pin PDIP




8-Pin TSSOP




8-Pin SOIC




# 2-Wire Serial EEPROM

**32K (4096 x 8)**

**64K (8192 x 8)**

# AT24C32
# AT24C64

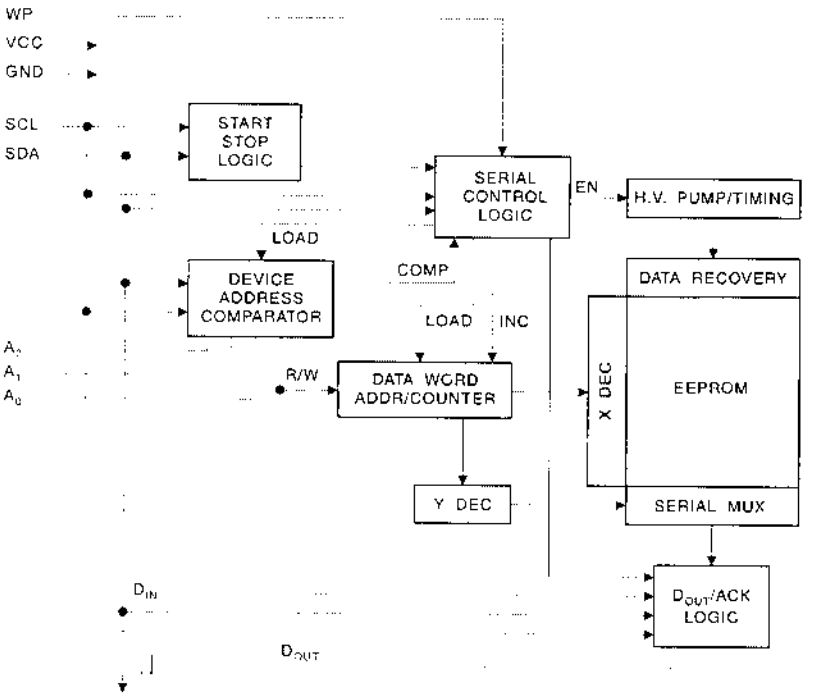
Rev. 0336F–08/98

## solute Maximum Ratings*

| | |
|---|---|
| rating Temperature | -55°C to +125°C |
| age Temperature | -65°C to +150°C |
| age on Any Pin Respect to Ground | -1.0V to +7.0V |
| imum Operating Voltage | 6.25V |
| Output Current | 5.0 mA |

*NOTICE: 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 is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

## ck Diagram




## Description

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

**AL DATA (SDA):** The SDA pin is bidirectional for data transfer. This pin is open-drain driven and may e-ORed with any number of other open-drain or open tor devices.

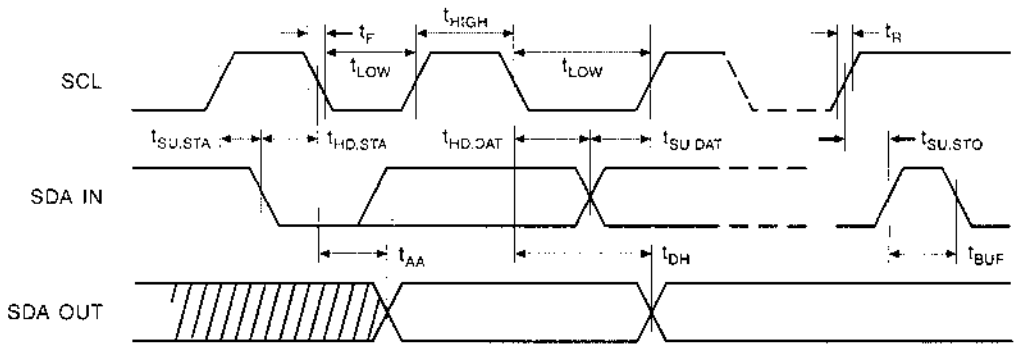
**CE/PAGE ADDRESSES (A2, A1, A0):** The A2, A1 0 pins are device address inputs that are hard wired t not connected for hardware compatibility with C16. When the pins are hardwired, as many as eight 4K devices may be addressed on a single bus system device addressing is discussed in detail under the Device Addressing section). When the pins are not hardwired, the default $A_2$, $A_1$, and $A_0$ are zero.

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

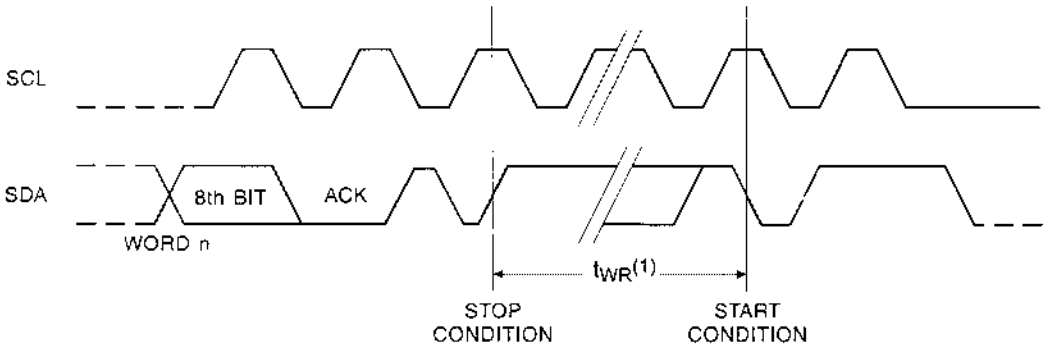
## Memory Organization

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

## Timing
## .: Serial Clock, SDA: Serial Data I/O

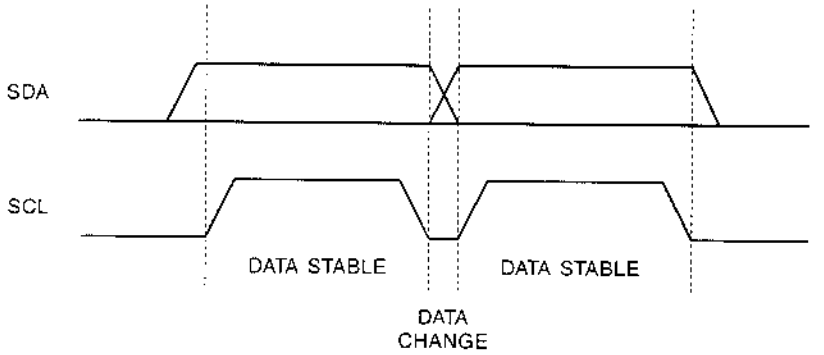



## te Cycle Timing
## .: Serial Clock, SDA: Serial Data I/O

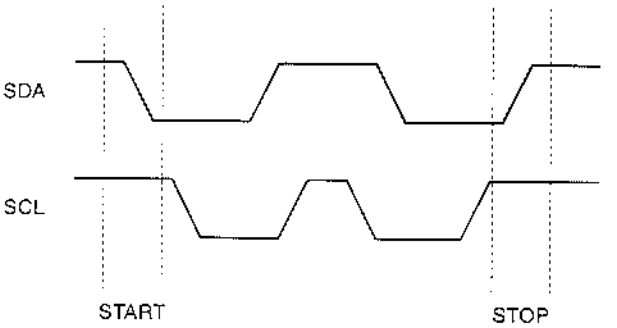



1. The write cycle time $t_{WR}$ is the time from a valid stop condition of a write sequence to the end of the internal clear/write cycle.

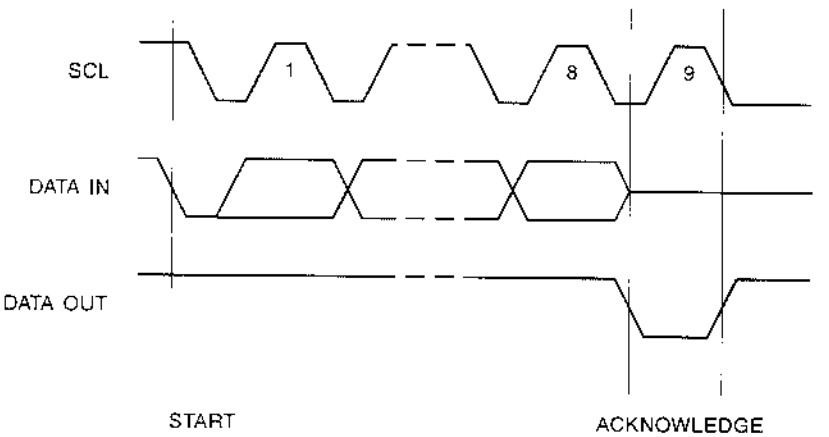
## a Validity




## rt and Stop Definition




## put Acknowledge



## ice Addressing

2K/64K EEPROM requires an 8-bit device address following a start condition to enable the chip for a read te operation (refer to Figure 1). The device address consists of a mandatory one, zero sequence for the ur most significant bits as shown. This is common to ire EEPROM devices.

2K/64K uses the three device address bits A2, A1, A0 w as many as eight devices on the same bus. These ust compare to their corresponding hardwired input The A2, A1, and A0 pins use an internal proprietary that biases them to a logic low condition if the pins lowed to float.

ighth bit of the device address is the read/write opera- elect bit. A read operation is initiated if this bit is high write operation is initiated if this bit is low.

a compare of the device address, the EEPROM will t a zero. If a compare is not made, the device will to standby state.

**E PROTECTION:** Special internal circuitry placed on DA and SCL pins prevent small noise spikes from ating the device. A low-$V_{CC}$ detector (5-volt option) s the device to prevent data corruption in a noisy envi- ent.

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

## te Operations

**WRITE:** A write operation requires two 8-bit data addresses following the device address word and owledgment. Upon receipt of this address, the ROM will again respond with a zero and then clock in st 8-bit data word. Following receipt of the 8-bit data the EEPROM will output a zero and the addressing e, such as a microcontroller, must terminate the write ence with a stop condition. At this time the EEPROM s an internally-timed write cycle, $t_{WR}$, to the nonvolatile ory. All inputs are disabled during this write cycle and EPROM will not respond until the write is complete to Figure 2).

**WRITE:** The 32K/64K EEPROM is capable of 32- page writes.

ge write is initiated the same way as a byte write, but icrocontroller does not send a stop condition after the data word is clocked in. Instead, after the EEPROM owledges receipt of the first data word, the microcon- r can transmit up to 31 more data words. The ROM will respond with a zero after each data word ved. The microcontroller must terminate the page sequence with a stop condition (refer to Figure 3).

The data word address lower 5 bits are internally incremented following the receipt of each data word. The higher data word address bits are not incremented, retaining the memory page row location. When the word address, internally generated, reaches the page boundary, the following byte is placed at the beginning of the same page. If more than 32 data words are transmitted to the EEPROM, the data word address will "roll over" and previous data will be overwritten.

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

## Read Operations

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

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

Once the device address with the read/write select bit set to one is clocked in and acknowledged by the EEPROM, the current address data word is serially clocked out. The microcontroller does not respond with an input zero but does generate a following stop condition (refer to Figure 4).

**RANDOM READ:** A random read requires a "dummy" byte write sequence to load in the data word address. Once the device address word and data word address are clocked in and acknowledged by the EEPROM, the microcontroller must generate another start condition. The microcontroller now initiates a current address read by sending a device address with the read/write select bit high. The EEPROM acknowledges the device address and serially clocks out the data word. The microcontroller does not respond with a zero but does generate a following stop condition (refer to Figure 5).

**SEQUENTIAL READ:** Sequential reads are initiated by either a current address read or a random address read. After the microcontroller receives a data word, it responds with an acknowledge. As long as the EEPROM receives an

## tures

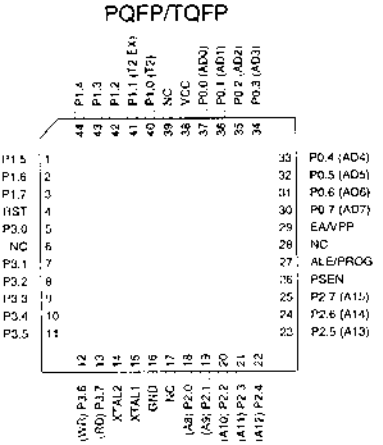
- npatible with MCS-51™ Products
- Bytes of In-System Reprogrammable Flash Memory
- urance: 1,000 Write/Erase Cycles
- y Static Operation: 0 Hz to 24 MHz
- ee-level Program Memory Lock
- x 8-bit Internal RAM
- Programmable I/O Lines
- ee 16-bit Timer/Counters
- ht Interrupt Sources
- grammable Serial Channel
- v-power Idle and Power-down Modes

## scription

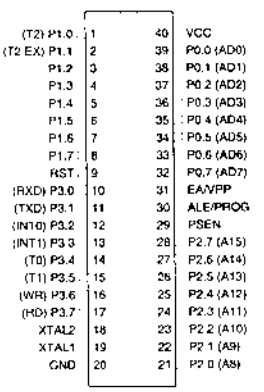
AT89C52 is a low-power, high-performance CMOS 8-bit microcomputer with 8K of Flash programmable and erasable read only memory (PEROM). The device anufactured using Atmel's high-density nonvolatile memory technology and is patible with the industry-standard 80C51 and 80C52 instruction set and pinout. on-chip Flash allows the program memory to be reprogrammed in-system or by a entional nonvolatile memory programmer. By combining a versatile 8-bit CPU Flash on a monolithic chip, the Atmel AT89C52 is a powerful microcomputer n provides a highly-flexible and cost-effective solution to many embedded control cations.

## Configurations

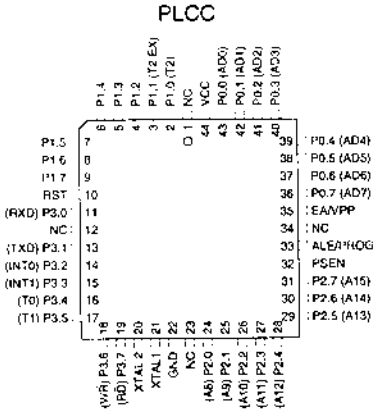
PQFP/TQFP




PDIP




PLCC




# 8-bit Microcontroller with 8K Bytes Flash

# AT89C52

Rev. 0313H–02/00

T89C52 provides the following standard features: 8K of Flash, 256 bytes of RAM, 32 I/O lines, three 16-bit counters, a six-vector two-level interrupt architecture, duplex serial port, on-chip oscillator, and clock cir- In addition, the AT89C52 is designed with static logic peration down to zero frequency and supports two are selectable power saving modes. The Idle Mode the CPU while allowing the RAM, timer/counters, port, and interrupt system to continue functioning. Power-down mode saves the RAM contents but es the oscillator, disabling all other chip functions until ext hardware reset.

## Description

ly voltage.

nd.

**0**

0 is an 8-bit open drain bi-directional I/O port. As an ut port, each pin can sink eight TTL inputs. When 1s vritten to port 0 pins, the pins can be used as high- dance inputs.

0 can also be configured to be the multiplexed low- r address/data bus during accesses to external pro- n and data memory. In this mode, P0 has internal ps.

0 also receives the code bytes during Flash program- g and outputs the code bytes during program cation. External pullups are required during program cation.

**1**

1 is an 8-bit bi-directional I/O port with internal pullups. Port 1 output buffers can sink/source four TTL inputs. n 1s are written to Port 1 pins, they are pulled high by nternal pullups and can be used as inputs. As inputs, 1 pins that are externally being pulled low will source ent ($I_{IL}$) because of the internal pullups.

ddition, P1.0 and P1.1 can be configured to be the r/counter 2 external count input (P1.0/T2) and the r/counter 2 trigger input (P1.1/T2EX), respectively, as vn in the following table.

1 also receives the low-order address bytes during h programming and verification.

| rt Pin | Alternate Functions |
|---|---|
| .0 | T2 (external count input to Timer/Counter 2), clock-out |
| .1 | T2EX (Timer/Counter 2 capture/reload trigger and direction control) |

**Port 2**

Port 2 is an 8-bit bi-directional I/O port with internal pullups. 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 pullups and can be used as inputs. As inputs, Port 2 pins that are externally being pulled low will source current ($I_{IL}$) because of the internal pullups.

Port 2 emits the high-order address byte during fetches from external program memory and during accesses to external data memory that use 16-bit addresses (MOVX @ DPTR). In this application, Port 2 uses strong internal pullups when emitting 1s. During accesses to external data memory that use 8-bit addresses (MOVX @ RI), 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.

**Port 3**

Port 3 is an 8-bit bi-directional I/O port with internal pullups. 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 pullups and can be used as inputs. As inputs, Port 3 pins that are externally being pulled low will source current ($I_{IL}$) because of the pullups.

Port 3 also serves the functions of various special features of the AT89C51, as shown in the following table.

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

| Port Pin | Alternate Functions |
|---|---|
| P3.0 | RXD (serial input port) |
| P3.1 | TXD (serial output port) |
| P3.2 | $\overline{INT0}$ (external interrupt 0) |
| P3.3 | $\overline{INT1}$ (external interrupt 1) |
| P3.4 | T0 (timer 0 external input) |
| P3.5 | T1 (timer 1 external input) |
| P3.6 | $\overline{WR}$ (external data memory write strobe) |
| P3.7 | $\overline{RD}$ (external data memory read strobe) |

**RST**

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

**ALE/$\overline{PROG}$**

Address Latch Enable is an output pulse for latching the low byte of the address during accesses to external memory. This pin is also the program pulse input ($\overline{PROG}$) 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

or clocking purposes. Note, however, that one ALE is skipped during each access to external data y.

ed, ALE operation can be disabled by setting bit 0 of cation 8EH. With the bit set, ALE is active only dur- MOVX or MOVC instruction. Otherwise, the pin is y pulled high. Setting the ALE-disable bit has no f the microcontroller is in external execution mode.

am Store Enable is the read strobe to external pro- memory.

the AT89C52 is executing code from external pro- memory, $\overline{PSEN}$ is activated twice each machine except that two $\overline{PSEN}$ activations are skipped during access to external data memory.

**$\overline{EA}$/VPP**

External Access Enable. $\overline{EA}$ 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, $\overline{EA}$ will be internally latched on reset.

$\overline{EA}$ should be strapped to $V_{CC}$ for internal program executions.

This pin also receives the 12-volt programming enable voltage ($V_{PP}$) during Flash programming when 12-volt programming is selected.

**XTAL1**

Input to the inverting oscillator amplifier and input to the internal clock operating circuit.

**XTAL2**

Output from the inverting oscillator amplifier.

**1.** AT89C52 SFR Map and Reset Values

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| F8H | | | | | | | | 0FFH |
| F0H | B<br>00000000 | | | | | | | 0F7H |
| E8H | | | | | | | | 0EFH |
| E0H | ACC<br>00000000 | | | | | | | 0E7H |
| D8H | | | | | | | | 0DFH |
| D0H | PSW<br>00000000 | | | | | | | 0D7H |
| C8H | T2CON<br>00000000 | T2MOD<br>XXXXXX00 | RCAP2L<br>00000000 | RCAP2H<br>00000000 | TL2<br>00000000 | TH2<br>00000000 | | 0CFH |
| C0H | | | | | | | | 0C7H |
| B8H | IP<br>XX000000 | | | | | | | 0BFH |
| B0H | P3<br>11111111 | | | | | | | 0B7H |
| A8H | IE<br>0X000000 | | | | | | | 0AFH |
| A0H | P2<br>11111111 | | | | | | | 0A7H |
| 98H | SCON<br>00000000 | SBUF<br>XXXXXXXX | | | | | | 9FH |
| 90H | P1<br>11111111 | | | | | | | 97H |
| 88H | TCON<br>00000000 | TMOD<br>00000000 | TL0<br>00000000 | TL1<br>00000000 | TH0<br>00000000 | TH1<br>00000000 | | 8FH |
| 80H | P0<br>11111111 | SP<br>00000111 | DPL<br>00000000 | DPH<br>00000000 | | | PCON<br>0XXX0000 | 87H |

## cial Function Registers

of the on-chip memory area called the Special Func-egister (SFR) space is shown in Table 1.

nat not all of the addresses are occupied, and unoc-l addresses may not be implemented on the chip. accesses to these addresses will in general return n data, and write accesses will have an indetermi-ffect.

software should not write 1s to these unlisted loca-since they may be used in future products to invoke new features. In that case, the reset or inactive values of the new bits will always be 0.

**Timer 2 Registers** Control and status bits are contained in registers T2CON (shown in Table 2) and T2MOD (shown in Table 4) for Timer 2. The register pair (RCAP2H, RCAP2L) are the Capture/Reload registers for Timer 2 in 16-bit capture mode or 16-bit auto-reload mode.

**Interrupt Registers** The individual interrupt enable bits are in the IE register. Two priorities can be set for each of the six interrupt sources in the IP register.r

**2.** T2CON – Timer/Counter 2 Control Registe

T2CON Address = 0C8H — Reset Value = 0000 0000B

Bit Addressable

| Bit | TF2 | EXF2 | RCLK | TCLK | EXEN2 | TR2 | C/$\overline{T2}$ | CP/$\overline{RL2}$ |
|---|---|---|---|---|---|---|---|---|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

| bol | Function |
|---|---|
| | Timer 2 overflow flag set by a Timer 2 overflow and must be cleared by software. TF2 will not be set when either RCLK = 1 or TCLK = 1. |
| 2 | Timer 2 external flag set when either a capture or reload is caused by a negative transition on T2EX and EXEN2 = 1. When Timer 2 interrupt is enabled, EXF2 = 1 will cause the CPU to vector to the Timer 2 interrupt routine. EXF2 must be cleared by software. EXF2 does not cause an interrupt in up/down counter mode (DCEN = 1). |
| K | Receive clock enable. When set, causes the serial port to use Timer 2 overflow pulses for its receive clock in serial port Modes 1 and 3. RCLK = 0 causes Timer 1 overflow to be used for the receive clock. |
| K | Transmit clock enable. When set, causes the serial port to use Timer 2 overflow pulses for its transmit clock in serial port Modes 1 and 3. TCLK = 0 causes Timer 1 overflows to be used for the transmit clock. |
| N2 | Timer 2 external enable. When set, allows a capture or reload to occur as a result of a negative transition on T2EX if Timer 2 is not being used to clock the serial port. EXEN2 = 0 causes Timer 2 to ignore events at T2EX. |
| | Start/Stop control for Timer 2. TR2 = 1 starts the timer. |
| $\overline{2}$ | Timer or counter select for Timer 2. C/$\overline{T2}$ = 0 for timer function. C/$\overline{T2}$ = 1 for external event counter (falling edge triggered). |
| $\overline{RL2}$ | Capture/Reload select. CP/$\overline{RL2}$ = 1 causes captures to occur on negative transitions at T2EX if EXEN2 = 1. CP/$\overline{RL2}$ = 0 causes automatic reloads to occur when Timer 2 overflows or negative transitions occur at T2EX when EXEN2 = 1. When either RCLK or TCLK = 1, this bit is ignored and the timer is forced to auto-reload on Timer 2 overflow. |

## ta Memory

AT89C52 implements 256 bytes of on-chip RAM. The er 128 bytes occupy a parallel address space to the cial Function Registers. That means the upper 128 s have the same addresses as the SFR space but are ically separate from SFR space.

n an instruction accesses an internal location above ress 7FH, the address mode used in the instruction specifies whether the CPU accesses the upper 128 bytes of RAM or the SFR space. Instructions that use direct addressing access SFR space.

For example, the following direct addressing instruction accesses the SFR at location 0A0H (which is P2).

```
MOV 0A0H, #data
```

ctions that use indirect addressing access the upper ytes of RAM. For example, the following indirect ssing instruction, where R0 contains 0A0H, accesses ata byte at address 0A0H, rather than P2 (whose ss is 0A0H).

```
V @R0, #data
```

that stack operations are examples of indirect ssing, so the upper 128 bytes of data RAM are avail- s stack space.

## er 0 and 1

0 and Timer 1 in the AT89C52 operate the same way ner 0 and Timer 1 in the AT89C51.

## er 2

2 is a 16-bit Timer/Counter that can operate as either er or an event counter. The type of operation is ed by bit C/$\overline{T2}$ in the SFR T2CON (shown in Table 2). 2 has three operating modes: capture, auto-reload r down counting), and baud rate generator. The s are selected by bits in T2CON, as shown in Table 3.

2 consists of two 8-bit registers, TH2 and TL2. In the r function, the TL2 register is incremented every ine cycle. Since a machine cycle consists of 12 oscil- periods, the count rate is 1/12 of the oscillator ency.

**3.** Timer 2 Operating Modes

| LK +TCLK | CP/$\overline{RL2}$ | TR2 | MODE |
|---|---|---|---|
| 0 | 0 | 1 | 16-bit Auto-reload |
| 0 | 1 | 1 | 16-bit Capture |
| 1 | X | 1 | Baud Rate Generator |
| X | X | 0 | (Off) |

e Counter function, the register is incremented in onse to a 1-to-0 transition at its corresponding external input pin, T2. In this function, the external input is sampled during S5P2 of every machine cycle. When the samples show a high in one cycle and a low in the next cycle, the count is incremented. The new count value appears in the register during S3P1 of the cycle following the one in which the transition was detected. Since two machine cycles (24 oscillator periods) are required to recognize a 1-to-0 transition, the maximum count rate is 1/24 of the oscillator frequency. To ensure that a given level is sampled at least once before it changes, the level should be held for at least one full machine cycle.

### Capture Mode

In the capture mode, two options are selected by bit EXEN2 in T2CON. If EXEN2 = 0, Timer 2 is a 16-bit timer or counter which upon overflow sets bit TF2 in T2CON. This bit can then be used to generate an interrupt. If EXEN2 = 1, Timer 2 performs the same operation, but a 1-to-0 transition at external input T2EX also causes the current value in TH2 and TL2 to be captured into RCAP2H and RCAP2L, respectively. In addition, the transition at T2EX causes bit EXF2 in T2CON to be set. The EXF2 bit, like TF2, can generate an interrupt. The capture mode is illustrated in Figure 1.

### Auto-reload (Up or Down Counter)

Timer 2 can be programmed to count up or down when configured in its 16-bit auto-reload mode. This feature is invoked by the DCEN (Down Counter Enable) bit located in the SFR T2MOD (see Table 4). Upon reset, the DCEN bit is set to 0 so that timer 2 will default to count up. When DCEN is set, Timer 2 can count up or down, depending on the value of the T2EX pin.

# +5V-Powered, Multichannel RS-232 Drivers/Receivers

## General Description

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

## Features

### Superior to Bipolar

- **Operate from Single +5V Power Supply (+5V and +12V—MAX231/MAX239)**
- **Low-Power Receive Mode in Shutdown (MAX223/MAX242)**
- **Meet All EIA/TIA-232E and V.28 Specifications**
- **Multiple Drivers and Receivers**
- **3-State Driver and Receiver Outputs**
- **Open-Line Detection (MAX243)**

## Ordering Information

| PART | TEMP. RANGE | PIN-PACKAGE |
|---|---|---|
| **MAX220**CPE | 0°C to +70°C | 16 Plastic DIP |
| MAX220CSE | 0°C to +70°C | 16 Narrow SO |
| MAX220CWE | 0°C to +70°C | 16 Wide SO |
| MAX220C/D | 0°C to +70°C | Dice* |
| MAX220EPE | -40°C to +85°C | 16 Plastic DIP |
| MAX220ESE | -40°C to +85°C | 16 Narrow SO |
| MAX220EWE | -40°C to +85°C | 16 Wide SO |
| MAX220EJE | -40°C to +85°C | 16 CERDIP |
| MAX220MJE | -55°C to +125°C | 16 CERDIP |

***Ordering Information continued at end of data sheet.***

**Contact factory for dice specifications.*

## Selection Table

| Part Number | Power Supply (V) | No. of RS-232 Drivers/Rx | No. of Ext. Caps | Nominal Cap. Value (µF) | SHDN & Three-State | Rx Active in SHDN | Data Rate (kbps) | Features |
|---|---|---|---|---|---|---|---|---|
| MAX220 | +5 | 2/2 | 4 | 0.1 | No | — | 120 | Ultra-low-power, industry-standard pinout |
| MAX222 | +5 | 2/2 | 4 | 0.1 | Yes | - | 200 | Low-power shutdown |
| MAX223 (MAX213) | +5 | 4/5 | 4 | 1.0 (0.1) | Yes | ✔ | 120 | MAX241 and receivers active in shutdown |
| MAX225 | +5 | 5/5 | 0 | | Yes | ✔ | 120 | Available in SO |
| MAX230 (MAX200) | +5 | 5/0 | 4 | 1.0 (0.1) | Yes | -- | 120 | 5 drivers with shutdown |
| MAX231 (MAX201) | +5 and +7.5 to +13.2 | 2/2 | 2 | 1.0 (0.1) | No | | 120 | Standard +5/+12V or battery supplies; same functions as MAX232 |
| MAX232 (MAX202) | +5 | 2/2 | 4 | 1.0 (0.1) | No | | 120 (64) | Industry standard |
| MAX232A | +5 | 2/2 | 4 | 0.1 | No | — | 200 | Higher slew rate, small caps |
| MAX233 (MAX203) | +5 | 2/2 | 0 | - | No | -- | 120 | No external caps |
| MAX233A | +5 | 2/2 | 0 | | No | | 200 | No external caps, high slew rate |
| MAX234 (MAX204) | +5 | 4/0 | 4 | 1.0 (0.1) | No | | 120 | Replaces 1488 |
| MAX235 (MAX205) | +5 | 5/5 | 0 | | Yes | | 120 | No external caps |
| MAX236 (MAX206) | +5 | 4/3 | 4 | 1.0 (0.1) | Yes | | 120 | Shutdown, three state |
| MAX237 (MAX207) | +5 | 5/3 | 4 | 1.0 (0.1) | No | — | 120 | Complements IBM PC serial port |
| MAX238 (MAX208) | +5 | 4/4 | 4 | 1.0 (0.1) | No | — | 120 | Replaces 1488 and 1489 |
| MAX239 (MAX209) | +5 and +7.5 to +13.2 | 3/5 | 2 | 1.0 (0.1) | No | | 120 | Standard +5/+12V or battery supplies; single-package solution for IBM PC serial port |
| MAX240 | +5 | 5/5 | 4 | 1.0 | Yes | — | 120 | DIP or flatpack package |
| MAX241 (MAX211) | +5 | 4/5 | 4 | 1.0 (0.1) | Yes | — | 120 | Complete IBM PC serial port |
| MAX242 | +5 | 2/2 | 4 | 0.1 | Yes | ✔ | 200 | Separate shutdown and enable |
| MAX243 | +5 | 2/2 | 4 | 0.1 | No | | 200 | Open-line detection simplifies cabling |
| MAX244 | +5 | 8/10 | 4 | 1.0 | No | | 120 | High slew rate |
| MAX245 | +5 | 8/10 | 0 | - | Yes | ✔ | 120 | High slew rate, int. caps, two shutdown modes |
| MAX246 | +5 | 8/10 | 0 | -- | Yes | ✔ | 120 | High slew rate, int. caps, three shutdown modes |
| MAX247 | +5 | 8/9 | 0 | | Yes | ✔ | 120 | High slew rate, int. caps, nine operating modes |
| MAX248 | +5 | 6/8 | 4 | 1.0 | Yes | ✔ | 120 | High slew rate, selective half-chip enables |
| MAX249 | +5 | 6/10 | 4 | 1.0 | Yes | ✔ | 120 | Available in quad flatpack package |