DESIGN AND CONSTRUCTION OF AN ELECTRONIC SECURITY KEYPAD LOCK

BY

SAMUEL OLAWALE NIYI
2003/15413EE

A THESIS SUBMITTED TO THE DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING, FEDERAL UNIVERSITY OF TECHNOLOGY, MINNA, NIGER STATE.

NOVEMBER, 2008.

DECLARATION

I, Samuel Olawale Niyi, declare that this work was done by me and has never been presented elsewhere for the award of a degree. I also hereby relinquish the copyright to the Federal University of Technology, Minna.

SAMUEL OLAWALE NIYI

(Name of student)

10-11-2008

(Signature and Date)

MR. N. SALAWU

(Name of supervisor)

10/11/08

(Signature and Date)

Engr. Dr. Y. A. ADEDIRAN

(Name of H.O.D)

(Signature and Date)

(Name of External Examiner)

(Signature and Date)

DEDICATION

This project is dedicated to the Most High God who saw me through the course of my study in the university and through this project work and to my parents

Mr and Mrs. E. NIYI

ACKNOWLEDGEMENT

My deepest appreciation and profound gratitude goes to the Almighty God for his amazing grace and who granted me wisdom, strength and guidance during the course of my study.

My gratitude goes to my beloved and caring parents, Mr. and Mrs. E. Niyi for all their support that has motivated me through this path to a successful completion of my first degree programme. I also appreciate the contributions of my siblings; Mr. S Niyi, Mrs. T. Dasilva, Sist. Wemi, Tolu and Gbenga, especially those who have also reached out to me in my times of need.

My profound appreciation also goes to my very understanding project supervisor Mr. N. Salawu for his expertise and guidance, advices and contributions to making my project research, design and construction a success. I will also show my appreciation and admiration of all my lecturers especially those of the Electrical and Computer Engineering department for the impartation of knowledge both academically and morally.

Finally, to my fellowship CACSF, course mates and friends whose presence made the whole experience worth while, I say Adieu and thank you very much and I love you all so dearly.

ABSTRACT

The electronic **Security Keypad Lock Project** is a basic access control system that is interfaced to a reversible electronic door motor for opening and closing the door. The "Code Lock" ability will allow the rightful user to deploy the platform to any property that requires simple password-protection. The Intel microcontroller used ensures low costs. The project Exploits the versatility of the microcontroller and flexibility of implementation of security using software including features which prevent criminals from breaking the password protection such features include password encryption, security keypad lockout after three fail attempts, and wire tapping lockout. Combining these feature makes for more protection; reliable, efficient and effective lock whose limitations are minimal.

TABLE OF CONTENTS

CHAPTER TWO: LITERATURE REVIEW

CHAPTER THREE: DESIGN AND IMPLEMENTATION

CHAPTER FOUR: TEST, RESULT AND DISCUSSION

CHAPTER FIVE: CONCLUSION

LIST OF FIGURES

LIST OF TABLES

CHAPTER ONE

1.0 INTRODUCTION

A **combination lock** is a type of lock in which a sequence of numbers or symbols is used to open the lock. The sequence may be entered using a single rotating dial which interacts with several discs or *cams*, by using a set of several rotating discs with inscribed numerals which directly interact with the locking mechanism, or through an electronic or mechanical keypad. Combination lock requires the correct permutation, not merely the correct combination of digits [1].

Electronic combination locks require the user to enter a numeric sequence on a keypad and facilitate entry through the use of electronic circuitry. The chief advantage of this system is that if used for the door of a large office, each employee can be told the code number without having to supply a key to each person. Electronic combination locks, while generally safe from the attacks on their mechanical counterparts, suffer from their own set of flaws. It is much easier to determine the lock sequence by viewing several successful accesses, since the arrangement of numbers is fixed.

This security keypad lock project is a Standalone system which provides controlled access and security to a single door, working in conjunction with either a mechanical or electric lock. The "Code Lock" ability will allow the rightful user to deploy the platform to any property that requires simple password-protection. Door mounted units will be battery operated, and are the simplest to install. Wall mounted units require mains power with backup battery in case of power failure from mains, and are more aesthetically pleasing and offer a higher level of security [2].

1.1 AIMS AND OBJECTIVE

The objectives of this project are

1. To design a project that is low-end and low-cost system that will focus mainly on affordability and simplicity rather than connectivity or even ease-of-use. Hence this will be a stand-alone system that will not need a special communications interface.
2. Creating a lock whose key can be changed when considered necessary
3. To emphasize the versatility of a microcontroller based project.
4. Meeting the security specifications and requirements of homes, offices and public utilities security standard.

1.2 METHODOLOGY

The project exploits the versatility of the microcontroller (Intel 89C51) and flexibility of implementation of logical operations using software. On initialization, the console request for a user password. When the user enters the password the microcontroller compares it with that stored in the EEPROM. If the password marches the microcontroller will activate the motor driver which controls the door lock. The door will open and wait for a few minutes and then closes. If a wrong password is entered the system will initialize and request for the user password. If a three consecutive wrong password is entered, the microcontroller will implement a keypad defensive lockout which will last for only a few minutes. The console consists of five sub-systems: Power supply unit, keypad, controller, non-volatile memory and feedback/mechanical interface.

Keypad

The properties of the key interface will determine the type of environment the system may be deployed to. For example, the availability of sealed and hardened keypad would allow the system to function in industrial and outdoor settings. In order to process the user's commands in a binary environment, software encoding has been provided so that the microcontroller can translate on/off keystrokes into a digital form.

Controller

The AT89S51 is a low-power, high-performance CMOS 8-bit microcomputer with 4K bytes of Flash Programmable and Erasable Read Only Memory (PEROM). The device is manufactured using Atmel's high density nonvolatile memory technology and is compatible with the industry standard MCS-51™ instruction set and pinout. The on-chip Flash allows the program memory to be reprogrammed in-system or by a conventional nonvolatile memory programmer. By combining a versatile 8-bit CPU with Flash on a monolithic chip, the Atmel AT89C51 is a powerful microcomputer which provides a highly flexible and cost effective solution to many embedded control applications [3].

Feedback and Mechanical Interface

The feedback system is based on a responsive LED system which will affirm every keystroke from the keypad into the microcontroller.

The electro-mechanical interface will be responsible for engaging and disengaging the external locking mechanism. To be more specific, the controller is supposed to provide signaling to the outside device, plus limited power for locking and unlocking.

EEPROM

A non-volatile data storage medium. Electronically Erasable Programable Read Only Memory. Their serial interface, small size, and low power consumption make them very practical as a means to hold serial numbers, manufacturing information, and configuration data [4, 5]. EEPROM will be used to store the passwords.

1.3 SCOPE OF WORK

The project work only covers all involved in constructing an electronic code lock, but in no way does the research attempt to explain the mechanical makeup of the locking device.

1.4 PROJECT OUTLINE

Chapter one contains the general introduction of the project, the scope and objectives, methodology, scope of work. The second chapter contains the literature review/theoretical background of the project. The design and implementation of the report is contained in chapter three. The steps taken in the design were carefully laid out in this chapter. Chapter four contains the tests carried out and their outcome (results) and a brief but concise discussion of the results obtained. Chapter five which is the last chapter contains recommendations and conclusions. It also includes references.

CHAPTER TWO

LITERATURE REVIEW

2.1 HISTORICAL BACKGROUND

The simplest form of lock is the ward locks, which essentially is a bolt containing a notch known as a talon. The bolt is moved backwards or forward by engaging a key in the talon. A back spring attached to the bolt held it in place once it is released by the key. The tumbler or lever lock, similar to the ward lock, contains one or more pieces of metal of different heights known as tumblers, levers or latches which intercepts the bolt and prevents it from being moved until the tumblers are raised or released by the action of an appropriate key [8].

The first mechanical lock is an Egyptian door lock made of wood used about 4000 years ago in Egypt. It was probably created by a number of civilizations at that time. The locks were fastened vertically on the post, the wooden lock contains movable pins that drops by gravity into opening in the cross piece or bolt or locks the door. It was operated by a wooden key with pegs or prongs that raised the number of tumblers sufficiently to clear the bolt so that it could be pulled back [8].

The earliest lock in existence is an Egyptian lock made of wood, found with its key in the ruins of Nineveh in ancient Assyria [9]. In construction, it is the prototype of the modern cylinder lock. Locks and keys are also mentioned in the Old Testament, the Greeks and Romans used locks of simple design. Medieval artisans designed locks of exquisite detail, the perfections and carvings often having no relation to the lock. But they still had a disadvantage of not long lasting as they were wholly made of wood.

New concepts for locking devices were developed in Europe in the 17th century. For example, the Romans fabricated the first metal locks with an improvement by Robert Barson, an Englishman in 1778. With the exception of the development of ward locks, little was done to improve the efficiency and convenience of locks until the late 18th century.

In the year 1861, an American named Linus Yale Jr. invented the Yale lock and he was the first to use a small, flat key in place of a large one [8].

The Yale lock consists essentially of a cylindrical plug placed in an outer basset. The plug is rotated by a key and in turn moves the bolt of the lock by means of a cam. In order to rotate the plug, the inserted key must raise five pins of different sizes into corresponding holes in the plug. If the pins are not raised to the circumference of the plug, the plug cannot be turned.

In the 19th century, ward locks were improved and tumbler or lever locks, pin tumbler or cylinder locks were converted and improved. The most common form of cylinder locks used in the home is the night latch operated by a key from the outside and a knob from the inside. Another type of lock that is increasing in use is the magnetic lock, which is essentially the same as a cylinder lock , except the pins need a suitably magnetized key to bring them into alignment so that they can be tossed [6].

In the 20th century, as machine tools and manufacturing methods became more sophisticated, locks were produced with closer part tolerances, resulting in better security. Subsequent development has focused on mass production, improvement of materials and increasing complexity of the working mechanisms including the increasing use of automatic electronic alarm and safety devices.

Many types of locks appeared in the 20th century. Locks are now either key operated i.e. key locks or keyless.

Narrowing down to the key locks, in the late 20th century, electromechanical locks were developed to trip electrical circuit as in automobile ignitions. Unfortunately, the use of key locks is unreliable with the use of master keys by illegal intruders.

The keyless locks are the most modern locks. They were first invented and made popular in the late 20th century. The keyless locks were remote control lock, security card operated, combination lock and electronic code lock. Of the various types of keyless locks, the electronics code lock is the most versatile and it is incorporated into various safes and vaults [6].

2.2 TYPES AND FUNCTIONS OF LOCKS

There are many types of locks in existence today but the main types are:

1. Mechanical key locks
2. Magnetic locks
3. Electronic locks

Mechanical key locks

These are locks that consist of a bolt that may be slid to and fro or rotated by a key (e.g. padlock). In these types of locks, there are obstacles called wards or tumblers that permit only the right key to be turned on. This is mostly used in doors, gates and windows of houses, stores etc.

Magnetic locks

These are locks that are operated based on the theory of magnetism. These types of locks consist of bolts connected to magnets to ensure that they are locked. The key (which is usually a ferrous metal foil) when inserted pulls the bolts thereby releasing the lock to ensure it is opened (e.g. solenoid). It is mainly used in residential as well as administrative areas (e.g. office) [7].

Electronic locks

These are most sophisticated of all locks. This is because unlike the two other types that require a sort of key to open them, this doesn't require a key so the issue of keys gets missing does not arise. These types of locks are driven basically by electronic means and they are mostly used in industrial areas and areas where high security is needed. Some of the electronic locks are describe below:

1. DNA sensor locks: These are sophisticated electronic locks that stores into their memory, the genetic makeup of the individual such that only individuals that have their DNA computed into its memory would be grant access. This is used in foreign countries in places where high level of security is needed. The DNA locks are expensive to produce.
2. Card sensor locks: These are electronic locks that use codes as keys such that when the card is inserted, it generates voltage by closing the circuit and energizing the relay which then opens the lock. These are used in industrial areas.
3. Electronic key locks: These are electronic locks that are operated by putting the correct code by an external means such as a keypad so that only people with correct code can open the lock.

4. Thumbprint sensor locks: These are electronic locks that use the thumbprints of the user as the key such that it is only individuals whose thumbprints have been stored in its memory would open the lock.

2.3 ELECTRONIC DOOR HARDWARE

Electronic door hardware enables doors to be locked and unlocked by a peripheral device. The device may be a simple electric push button or a motion sensor, or may be an elaborate access control device such as a card reader or digital keypad.

Two important terms to remember when dealing with electronic door hardware are fail safe and fail secure. Both terms are usually addressed in regard to fire/life safety codes and pertain to doors in the path of egress from an occupied space or during a fire emergency. Local fire codes vary by municipality. Security professionals should refer to the authority having jurisdiction when specifying any door locking mechanisms.

A lock for a door that normally provides free egress by simply turning a handle or depressing the exit bar from the secure side does not need to be fail safe. For instance, an electrified crashbar (panic bar) will mechanically unlock its door when pushed regardless of whether the lock is electrically energized or not.

A **Fail Safe Lock** will unlock under any failure condition. The most common failure modes are loss of power and activation of fire detection or protection systems. However, failure of the mechanism itself and any connected control device should also be considered.

A **Fail Secure Lock** will remain locked when the power is lost, fire alarm activates or other failure occurs. A fail secure lock may be used on a door in the path of egress provided that the free egress is available regardless of the state of the lock's power or other control mechanism.

Delayed Egress is designed to delay egress through an opening using panic or fire exit device for 15 seconds. Delayed egress systems are variations of maglocks and electrified crashbar devices, and offer a 15 to 30 second delay to prevent unauthorized exit. This provides security personnel enough time to avert unauthorized exits, while preventing people from being trapped inside. These locking arrangements are ideal for monitored care facilities such as mental institutions. The systems function in the fail-safe mode during a fire alarm or power outage. Security professionals intending to use them should check with their local code officials and/or fire marshals before installing the delayed egress. Some communities require a special variance while some have banned them completely [5].

2.4 TYPES OF ELECTRONIC DOOR HARDWARE

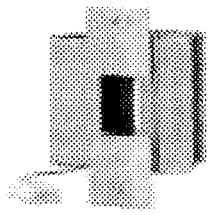

Fig. 2.1: Electric strike [5]

Electric strike provides remote release of a locked door. They allow the door to be opened without retracting the latchbolt. This occurs by the releasing of the electric strike lip (sometimes called a keeper or gate). When the door closes the beveled latchbolt rides

over the lip and falls into the strike pocket. Keeping the knob or handle free (allowing it to turn) on the secure side will permit free egress.

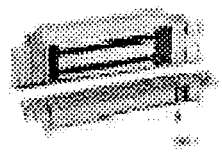

Fig. 2.2: Electromagnetic Lock [5]

Electromagnetic Lock is easy to install. The most popular locks in use today are the electromagnetic or maglocks. Maglocks are available in holding forces from 600 to 1650 pounds. With exception of the shear lock, there are no moving parts, making these locks virtually maintenance free.

Another variation of the maglock is the shear lock. This lock is fully concealed in the header or frame and typically has a holding force of 2,700 pounds.

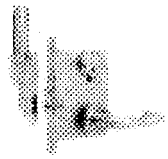

Fig. 2.3: Stairtower Lock [5]

Stairtower Lock As high rise buildings became more common, laws required that electronic control of stairwell doors be able to unlock the stairwell side of the door without unlatching it to prevent people from becoming trapped in a fire situation. This applies to both new construction and retrofit on older multi-story buildings. The mechanism consists of a frame mounted electric solenoid device that does not protrude beyond the frame but controls the dead-locking mortise latch bolt of regular mortise type door lock. Removal of power to the device releases the dead-locking latch bolt mechanism to allow the door handles to operate. One of the lock's advantages is that

drilling the door for electrical wiring, which could negate the door's fire certification, is not required.

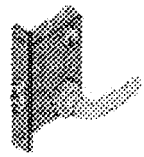

Fig. 2.4: Electric Mortise Lock [5]

Electric Mortise Lock This lock is simply a regular mortise lockset that has been electrified to control the ability to turn the handle. The lock is contained within the door and requires the door to be drilled to allow power wiring to be fed to the hinge side. The cabling must then be fed either through (electric hinge) or around (armored cable loop) the door hinge. The fixed, unsecured side of the lockset is controlled by an access control device (i.e. card reader, keypad) while the secure side handle remains mechanical at all times for unimpeded egress

Some lock manufacturers offer an optional built-in REX micro-switch which automatically shunt any door alarm for a valid egress. If this option is not used then some external REX device such as a REX sensor must be used to shunt door alarms.

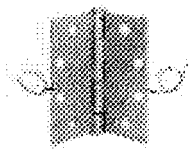

Fig. 2.5: Electric Transfer Hinge [5]

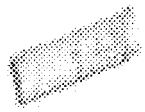

Fig. 2.6: REX Sensor [5]

Electrified Exit Devices (Crashbars)

This lock is simply a regular crashbar that has been electrified to control the ability to open the door from the unsecured side. The device is manufactured either with a rim device, mortise lock device or vertical rod device. The rim mounted device requires little modification to the door and is typical surface mounted. The mortise device is installed in

mullion to accept a latchbolt. Rim and mortise devices require the use of either a transfer device such as feed-through hinge or a power transfer hinge. The cabling must then be fed either through (electric hinge) or around (armored cable loop) the door hinge. The fixed, unsecured side of the lockset is controlled by an access control device (i.e. card reader, keypad) while the secure side remains mechanical at all times for unimpeded egress.

CHAPTER THREE

DESIGN AND IMPLEMENTATION

The electronic security keypad lock system was designed around the following subsystems.

1. 5 – volt system supply voltage.
2. Keypad for data input.
3. 8-bit system controller.
4. 256-byte non-volatile memory (24C02).
5. The reversible motor driver.

3.1 PRINCIPLE OF OPERATION

The block diagram in fig. 3.1 below shows the interrelationship between the subsystems that makeup this project.

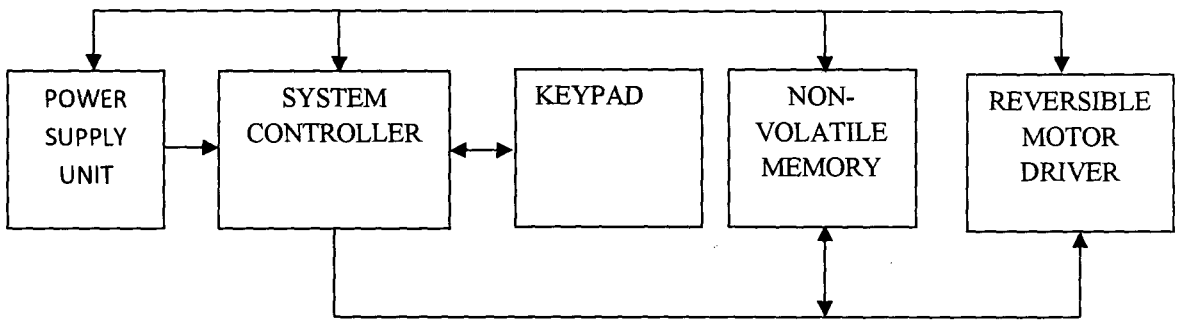

Fig. 3.1: Block diagram of the system

The electronic security keypad lock is a 10 digit combination lock that provides controlled access to a door using a motor driver. The system use a keypad for data input and this data is interpreted by the system form using software. The data is sent to the system controller which carries out validation of the data. The door is opened when a right combination of keys is entered.

This device is programmable. When a password change command is read on the keypad, the system request for old user password. If a right combination is entered, the system then request for a new user password. The new programmed codes are permanently stored in an EEPROM memory section which is remembered even after a power down. All these activities are coordinated by a program written and stored in the microcontroller.

3.2 POWER SUPPLY

The system power supply was derived from the local mains via a 240/12V 0.5A step-down transformer. The transformer was connected to a 4 diode full wave rectifier as depicted in fig 3.2

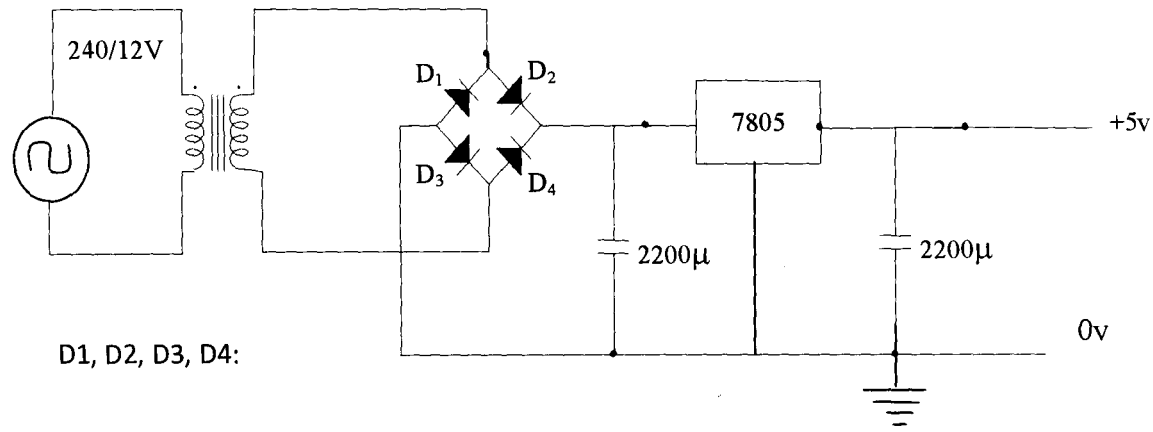

Fig. 3.2: System Power System

The 240V AC from the mains is stepped down by the transformer to 12Vrms

Peak secondary voltage V_{sp}

$$V_{SP} = 12\sqrt{2} \quad = 16.9V \tag{3.1}$$

This voltage is rectified by the bridge rectifier of D1, D2, D3 and D4. The values of IN4001 were chosen since it reverse breakdown rating is many times greater than V_{sp} of 16.9 volts.

For a full wave rectifier making use of silicon diodes, the forward voltage drop for two diode 0.7 X 2 = 1.4 is dropped from the peak secondary voltage.

The rectified dc output $V_{dc} = V_{sp} - 2V_d$, (3.2)

where V_d = Voltage drop across diode.

$V_{dc} = 16.9 - 1.4 = 15.5V$

V_{dc} is filtered to remove ripple voltages. This is achieved by the electrolytic filter capacitor C_1. The allowable ripple factor in the output voltage determines what minimal value of capacitance C_1 should have. For a pure dc supply such as a battery, the ripple factor γ is zero. But it is practically impossible for a rectified ac signal however; it is always desirable that the ripple factor approaches that of dc as much as possible. The ripple factor for a full wave rectified voltage is given by:

$$\gamma = \frac{I_{dc}}{4\sqrt{3}fcV_{ip}} \quad (3.3)$$

$$C = \frac{I_{dc}}{4\sqrt{3}f\gamma V_{ip}} \quad (3.4)$$

Where I_{dc} = load current

f = Frequency of the mains supply voltage

C = capacitance of filter capacitor

V_{ip} = peak full-wave rectified voltage at the filter input

The load current was calculated by summing together the system current drain as follows:

AT89S51 Controller	: 15mA
24C02 Memory	: 3mA
Led	: 20mA peak
Motor driver	: 15mA
	53mA

Using equation (3.1) to calculate the capacitance

$$C = \frac{53 \times 10^{-3}}{4\sqrt{3} \times 50 \times 0.1 \times 15.5} = 0.000098f$$

This value of capacitance is the minimum value required to keep the 5 volt system supply regulated. A value of 2200μF capacitance was used for improved system performance.

The smoothened DC voltage was fed into a 7805 5-volt regulator to keep the terminal voltage of the DC supply constant even when the AC input voltage into the transformer fluctuates. The output from the 5-volt regulator was fed into a second 2200μf capacitor C_2 which acts as a line filter to improve stability and transient response [11, 12].

3.3 KEYPAD

This is the input device. The keypad comprises 16 buttons on a 4X4 matrix, interfaced to the controller over Port 3 as shown in fig 3.3;

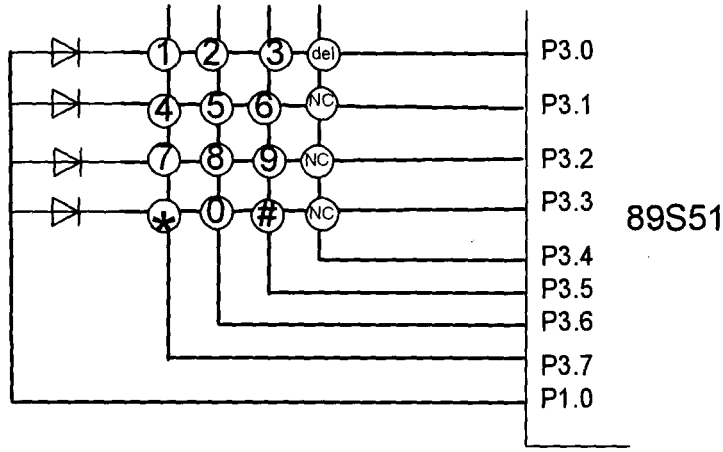

Fig. 3.3: Keypad

The matrix is a modified version of commonly implemented keypad matrix type in that a single bit output was provided along with the eight bits needed for the interfacing. Four IN4148 diodes are used to create the kind of logical relationship (OR gate) required between the single bit output and the eight used for interfacing.

The keypad port (port 3) was initialized with the binary pattern 00001111b at system power up. With no key pressed, the logic input into P1.0 is high. The software loops around this flag to detect key presses. However, if a key is pressed, say 1, P3.0 is low, and so also P1.0. The software enters into a keypad scanning routine to determine the key pressed. The software stores a unique digit for every low row (i.e P3.0 – P3.3). if P3.0 is low, a 0Ch is stored in a temporary register; if P3.1 is low, a 4 is stored; if P3.2 is detected low, a 8 is stored; for P3.3, a 12 is stored.

The software thereafter flips the logic level on port 3 and the value 11110000b is written to the port. Port P1.0 is again checked for a low, if a low is detected on P3.4, a 0 is stored in a second temporary resistor, if P3.8 is low, a 1 is stored, if P3.6, a 2 is stored. For P3.7, a 3 is stored.

The values of temporary registers 1 and temporary register 2 are added together and used as an index to a lookup table in memory. The value corresponding to the pressed key is thus read into a removable key code from flash memory.

3.4 8-BIT SYSTEM CONTROLLER

The 98S51 microcontroller was utilized for system control. The device was loaded with software that defined the system functionality. The software was used to address the keypad and perform the computations needed based on the input parameters.

The microcontroller is interfaced with

1. The keypad
2. The non-volatile memory
3. A 3 led system status indicator.
4. A BA6219B reversible-motor driver.

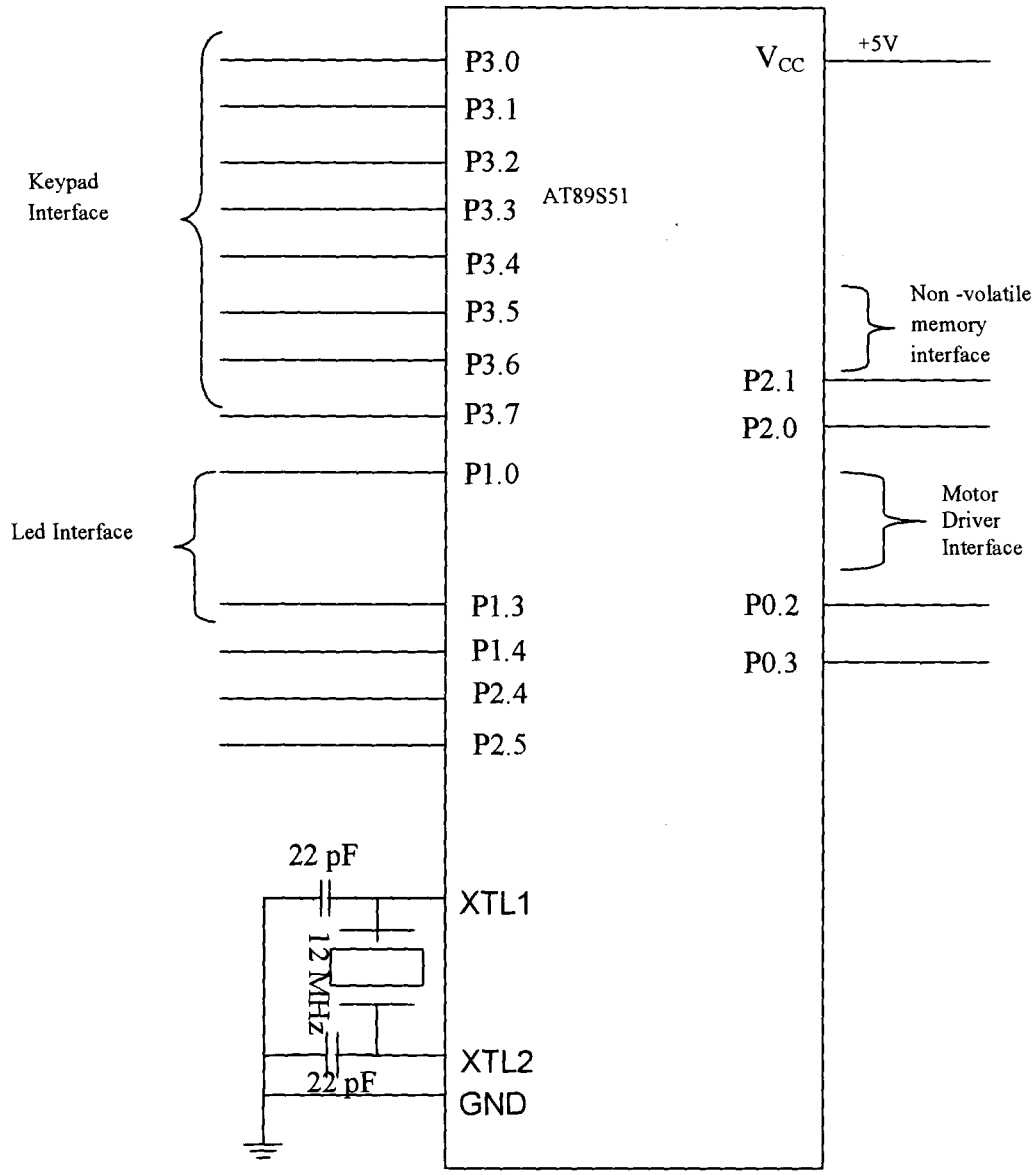

Fig. 3.4: AT89S51 pin interface [3].

The AT89S51 is a low-power, high-performance CMOS 8-bit microcontroller with 4K Bytes of in-system programmable Flash memory [3]. The device is manufactured using Atmel's high-density nonvolatile memory technology and is compatible with the industry-standard 80C51 instruction set and pinout. The on-chip Flash allows the program memory to be reprogrammed in-system or by a conventional nonvolatile memory programmer. By combining a versatile 8-bit CPU with in-system programmable Flash on a monolithic chip, the Atmel AT89S51 is a powerful microcontroller which provides a

powerful microcontroller which provides a highly-flexible and cost-effective solution to many embedded control applications. The AT89S51 provides the following standard features:

- Compatible with MCS-51® Products
- 4K Bytes of In-System Programmable (ISP) Flash Memory
- Endurance: 1000 Write/Erase Cycles
- 4.0V to 5.5V Operating Range
- Fully Static Operation: 0 Hz to 33 MHz
- Three-level Program Memory Lock
- 128 x 8-bit Internal RAM
- 32 Programmable I/O Lines
- Two 16-bit Timer/Counters
- Six Interrupt Sources
- Full Duplex UART Serial Channel
- Low-power Idle and Power-down Modes
- Interrupt Recovery from Power-down Mode
- Watchdog Timer
- Dual Data Pointer
- Power-off Flag
- Fast Programming Time
- Flexible ISP Programming (Byte and Page Mode)

The device was run on a 12MHz crystal. Normally an 8051 compactable microcontroller executes one instruction in 12 clock pulses [13]. Therefore the microcontroller will be able to perform 12,000,000 divided by 12 instructions in one second (1 MIPS i.e. one million instruction per second).

3.5 256-byte EEPROM

An AT24C02A, 2K SERIAL EEPROM, was interfaced to the microcontroller over P2.0 and P2.1. The device was used to store the user 10 digit password and other system configuration data.

The device has a proprietary interface not supported by the microcontroller so a bit-banded software emulation of the interface was implemented for microcontroller-memory interface.

The following sequence of actions is involved in talking to the non-volatile memory.

1. Start condition
2. Data read/write
3. Stop data

The microcontroller implemented the above three via software as included in the Appendix B.

Error indication was incorporated in the software routines to detect non-existent or faulty memory device. Should the controller detect a faulty I^2C bus transaction, the RED led is permanently lit. This can occur during a memory read or write cycle.

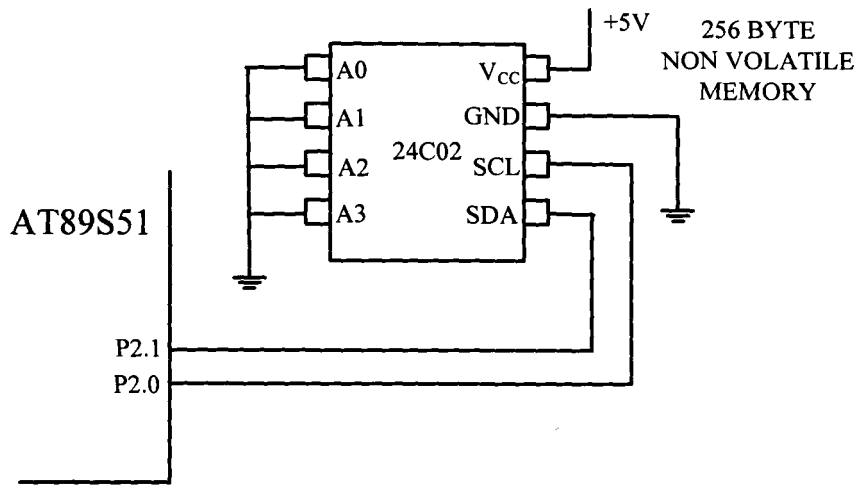

Fig 3.5 Non volatile memory [10]

Table 3.1: AT24C02A Pin Description [10]

PIN NAME	FUNCTION
A0 – A2	ADDRESS INPUTS
SDA	SERIAL DATA
SCL	SERIAL CLOCK INPUT
WP	WRITE PROTECT
NC	NO-CONNECT

The AT24C02A, serial EEPROM is internally organized with 32 pages of 8 bytes each, the 2K requires an 8-bit data word address for random word addressing [10]. The device has the following features;

• Write Protect Pin for Hardware Data Protection

• Low-voltage and Standard-voltage Operation

– 2.7 (VCC = 2.7V to 5.5V)

– 1.8 (VCC = 1.8V to 5.5V)

• Internally Organized 256 x 8 (2K)

• 2-wire Serial Interface

• Bi-directional Data Transfer Protocol

• 100 kHz (1.8V) and 400 kHz (5V) Clock Rate for AT24C02A

• 8-byte Page (2K) Write Mode

• Partial Page Writes are allowed

• Self-timed Write Cycle (10 ms max)

– Endurance: One Million Write Cycles

– Data Retention: 100 Years

3.6 SYSTEM STATUS INDICATOR

Since no text-based user-interactive communication medium was used, LEDS were used to indicate the system status. Three LEDs (green, orange, red) were connected to the controller through ports P1.3, P1.4, P2.5 to reflect the state of the system. A white LED was interfaced to the controller on pin P2.4 to affirm every keystroke on the keypad.

The green LED indicated system ready condition; the orange LED was used to provide password ok indication. A red LED was used as a several purpose error indicator.

The LEDs were runoff the +5 volt system voltage via current limiting resistors of values calculated from the follow expressions [15].

$$R_S = \frac{\text{Vs - Vled}}{\text{I}_{\text{led}}} \tag{3.5}$$

$V_s = +5V$

V_{led} = typically 1.7v

I_{led} = 5mA – 20mA

A continuous led current of 10mA was chosen.

$$R_s = \frac{5 - 1.7}{0.01} = 330\Omega$$

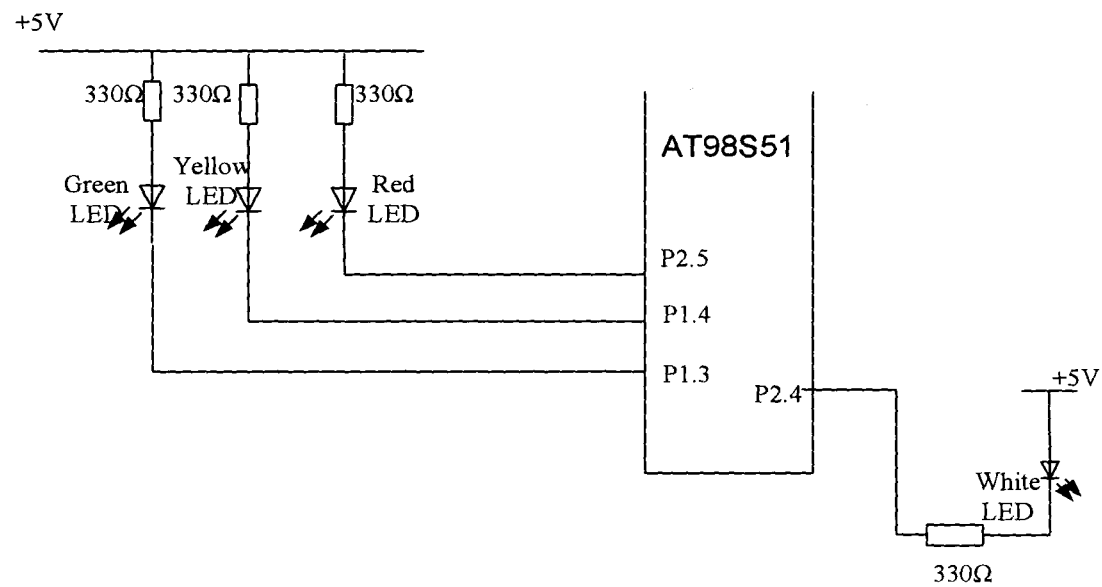

Fig. 3.6: System status indicator

3.7 THE REVERSIBLE-MOTOR DRIVER (BA6219B)

A BA6219B reversible-motor driver is used to interface the controller with a controlled motor for opening and closing a door. The motor driver is interfaced to the controller pin P0.2 and P0.3. The interfacing is done as shown in fig. 3.7 and table 3.2 shows the pins description of the motor driver

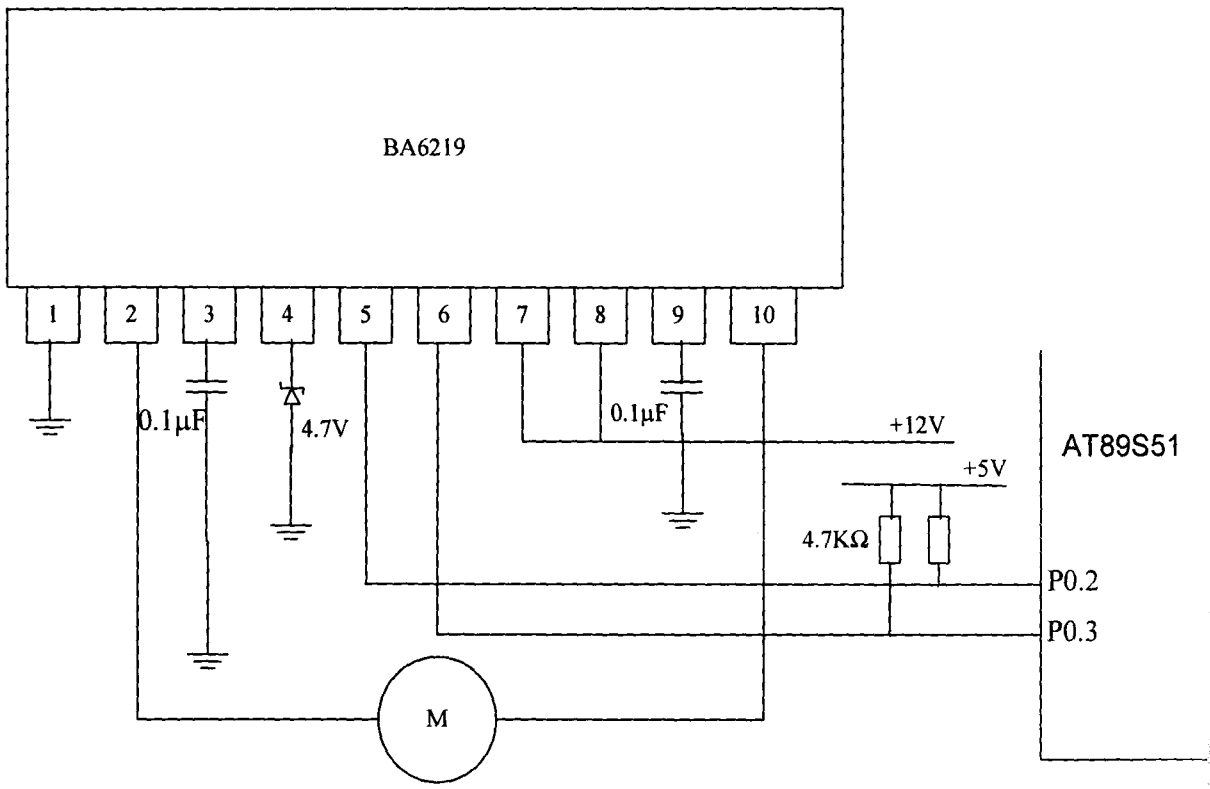

Fig. 3.7: Reversible motor driver

Table 3.2: BA6219B Pin Description [14]

PIN NUMBER	PIN NAME	FUNCTION
1	GND	GND
2	OUT_1	Motot output
3	C_{01}	Capacitor connection pin for preventing both output transistors being turne on at the same time
4	V_R	Output High voltage setting
5	IN_1	Logic input
6	IN_2	Logic input
7	VCC_1	Control circuit power supply
8	VCC_2	Output power supply
9	CD_2	Capacitor connection pin for preventing both output transistors being turned on at the same time
10	OUT_2	Motor output

The BA6219B is a reversible-motor driver suitable for brush motors. Two logic inputs allow four output modes: forward, reverse, idling, and braking. The motor revolving speed can be set arbitrarily by controlling the voltage applied to the motor [14].

Applications

VCRs and cassette tape recorders

Features

1) Large output current. (I_O=2.2A Max.)

2) Built-in thermal shutdown circuit.

3) Built-in output voltage setting pins.

4) Small standby supply current.

3.8 SOFTWARE OVERVIEW

The software is written in assembly language using the 98S51 instruction set. The software was modularized for easy design. The control software executes an endless loop waiting for a key to be pressed. When a key press is detected, the keypad is read. The software expects only two kinds of data from the keypad; a password change command or a password input.

If a password change command (*123#) is read over the keypad, the system demands for the previous password by pulsing the Orange LED three times. If the password is inputted and it matches the stored password, the green led is pulsed three times, prompting the user to input the new password. If the new password is inputted, the software checks to ensure that the inputted strings is between 0-9, otherwise an error is generated , prompting for a keypad re-read. If the keypad data satisfies system requirements, the new password is stored, over-writing the previously stored password, and access is granted via the motor driver which controls a motor that opens the door for about one minute and closes the door.

If a password input command, the keypad data is stored in a temporary location and compared to a lookup table and then validation begins. If the password matched the stored password, the orange led comes on and the door is opened and closed after a few minutes.

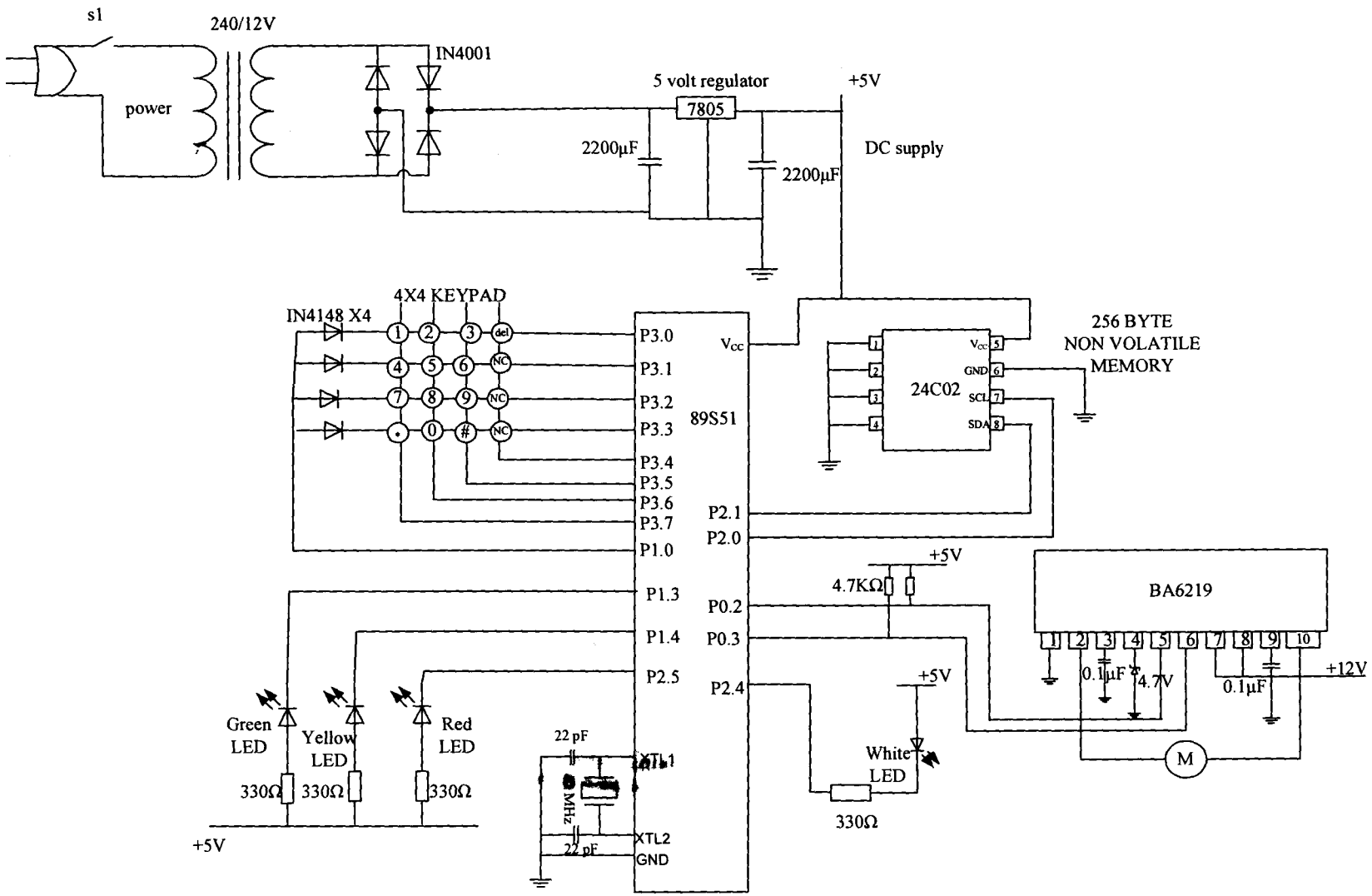

Fig.3.8 ELECTRONIC SECURITY KEYPAD LOCK

CHAPTER FOUR

TESTS, RESULTS AND DISCUSSION

4.1 CONSTRUCTION AND TESTING

With the successful completion of the design of this project, all the necessary components needed for its construction were bought. The construction started with the implementation of the design on a veroboard because of the static sensitive component. This was done in stages. The project work is divided into hardware and software implementation. The hardware implementation stages are; the power supply unit, keypad, the microcontroller, the non-volatile memory and the reversible motor driver.

A digital multimeter was used to ensure that contacts are made and to ensure continuity.

The other stage of the design is the software aspect, which completely defines the logical functionality of the entire system and this is done in assemble language and written into the microcontroller.

4.2 TEST ANALYSIS

Having completed the construction, the electronic security keypad lock is interfaced to the motor of a CDROM via the reversible motor driver for testing. The project was carefully tested as stated below.

1. The continuity and connectivity of the connecting wires and links were taken while not powered, using a digital multimeter.
2. Each module of the construction was tested to ensure accuracy.
3. All components with polarities were tested using their polarities.

4. The circuit was powered and the system was tested for the proper operation of the following features: a password entry command, password change command, keystroke timeout and wrong password.

4.3 DISCUSSION OF RESULTS

Testing is carried out in order to ensure a fault free operation of the system. During the test operations, the correct password is entered and the door opens for one minute and closes. During the password change command test, a wrong old user password is used and the error indicator comes on. During a second test a new user password that is less than 10 digits is used and the error indicator comes. The correct old user password and a 10 digit new user password were entered and the door opens for about one minute and closes. This simply indicates that the new user password has been accepted and the old user pass word is no longer valid. Keystroke timeout error occurs when less than 10 digits is entered or when too much time is used between each key press.

CHAPTER FIVE

CONCLUSION

5.1 PROBLEMS ENCOUNTERED

While carrying out this project work, several problems were encountered. Some of them are:

- The difficulty encountered sourcing for some components which led to the replacing of the relay (which is supposed to drive a solenoid lock) with a motor driver.
- Software development (Coding and debugging) in assembly programming language for the functionality of the controller.
- Irregular power supply
- Wastage of resources, time and energy resulting from damage of some components during soldering.

5.2 RECOMMENDATION

Some suggestions for further improvements on this project include

- This project is designed to be used by just one user since there is only one user account and password, I therefore recommend that as an improvement, multiple user accounts, each with its own password, be developed for the security keypad lock.
- A text-based user-interactive interface (liquid crystal display) can be designed to be used mainly during a password change command and multiple user sign in.

- The keypad console can also be designed to have a wireless interface with the electronic door hardware it is suppose to control.

5.3 CONCLUSION

The design and construction of the electronic security keypad lock was successfully carried out as described. The aims of the project were achieved. A closer look at the project shows that it has relevance in the technological age of ours. The versatility of the microcontroller cannot be overemphasize, with so many advantages like permitting the interfacing many devices and allowing flexibility while reducing the hardware circuitry needed and providing limitless functions implemented in software

With the construction of the electronic security keypad lock, a programmable access control system that can be modified to locked and unlocked any electronic door hardware by changing the mechanical interface is successfully constructed.

REFERENCES

[1] Retrieved from http://www.wikipedia.com (accessed on 16-7-2008)

[2] Retrieved from http://www.geocities.com/digitan000/hardware/5/e5_page.html (accessed on 16-7-2008)

[3] Atmel corporation, ATMEL AT89S51 DATASHEET, corporate headquarters, 2325 orchard parkway san jose, CA 95131, 2001, pp 1-10

[4] Mark Balch, Complete Digital Design - A Comprehensive Guide To Digital Electronics and Computer System Architecture, The Mcgraw Hill Companies, 2003, Pp. 79-81.

[5] Retrieved from http://www.seemgmt.com\typesoflocks (accessed on 19-7-2008)

[6] Retrieved from http://www.historyoflocks.com (accessed on 20-7-2008)

[7] Obilikwu A. John, Undergraduate Project "Design and Construction of Electronic Combination Lock System" October 2006, P. 5 -7. Unpublished

[8] Retrieved from http://www.investors.about.com/library/investors/bllock.html (accessed on 5-8-2008)

[9] Retrieved from http://www.locks.ru/germ/informat/schlagehistory.html (accessed on 2-8-2008)

[10] Atmel corporation, ATMEL AT24C02A DATASHEET, corporate headquarters, 2325 orchard parkway san jose, CA 95131, 2001, pp 1-10

[11] B.L. Theraja and A. K. Theraja, Electrical Technology. 2002 Edition: S. Chand and Company Ltd. India, 2002, pp. 1923-1938.

[12] Retrieved from http://www.fairchild.com (accessed on 23-8-2008)

[13] Vault Information Services, 8051 Tutorial. http://www.8052.com

[14] BA6219B Datasheet search [Internet]; ©2003-2006 Alldatasheet.com, Available at: http://www.alldatasheet.com/ (accessed on 18-8-2008)

[15] Timothy L. Skvarenina, The Power Electronic Handbook, Industrial Electronics Series, CRC Press LLC, 2002, pp 162-170

APPENDIX A

SYSTEM USER GUIDE

OPERATION MANUAL

When the security keypad is powered on, the green led comes on indicating system ready.

1. Enter the 10 digit user password, the orange led will come on indicating correct password and the door will open.
2. If the red led comes on that means an error (wrong password)

Password change

1. Enter *123# (the orange led will flash thrice)
2. Enter old user password (10 digits) (the green led will flash thrice)
3. Enter the new password (the orange light will come on and the door will open indicating that the password has been changed)

TIPS FOR GOOD PASSWORD SECURITY

1. Passwords must be 10 digits long.
2. They should not be: your phone number – these are equivalent to sending out invitations to hackers to break into your system!
3. Do not let anyone watch while you type in a password.
4. Never give out your password. Never.
5. Never write your passwords down and leave them near keypad lock.
6. Passwords should be changed no more frequently than once a month and no less frequently than once every 6 months.
7. Use a password validator or automatic generator if you do not trust yourself to come up with a sound password.

PRECAUTIONS ON PROPER SYSTEM USAGE.

1. Keep the system in a dry and clean environment
2. Make sure that appropriate power is supplied to the system
3. Don't open the system when ON to avoid electric shock.
4. Consult technician in case of system malfunction and consult a software engineer if the control software malfunctions.

APPENDIX B

SOFTWARE

```
INCLUDE 89c51.mc


sounder_dx BIT p2.4
relay_dx BIT p2.2
green_led BIT p1.3
yellow_led BIT p1.4
red_led BIT p2.5
STACK EQU 60h
dATA_rEAD  DATA 33
DAta_2_write DATA 32
clock BIT p2.0
data_out BIT p2.1
address DATA 28
card_Data_buffer DATA 8
keypad_data_buffer DATA 18
prog_mode_header_byte EQU 10
prog_mode_exit_byte EQU 11
error BIT 80
time_out_error BIT 83
chk_pgm_mode BIT 85
chk_password_mode BIT 86
password_valid BIT 87
key_code DATA 34
temp DATA 35
default_Flag BIT 88
format_valid BIT 89
flags DATA 28h
key_in BIT p1.0
row_1 BIT p3.3
row_2 BIT p3.2
row_3 BIT p3.1
row_4 BIT p3.0
col_1 BIT p3.7
col_2 BIT p3.6
col_3 BIT p3.5
col_4 BIT p3.4
last_Address EQU 128
slave_address EQU 0a0h
read_flag EQU 01h
write_flag EQU 00h
key EQU 4fh
format_key EQU 55h
password_retry DATA 38
card_io_Error BIT 92
mode_Sw BIT p2.3
FORMAT_FLAG EQU 55H
delete_flag EQU 15
enter_flag EQU 14
lock_on BIT 94
no_key BIT 95
motor_dx1 BIT p0.3
```

```
motor_dx2 BIT p0.2

;*********************************************************************
                        org 0000h

START:                  MOV sp, #stack
                        CALL long_Delay2
                        CALL init_system_handler
;*********************************************************************
mainloop:               MOV password_Retry,#3

wait_card:              cALL get_format_status_handler
                        JBC card_io_error, flag_error_1
                        JBC format_valid, go_on_1
                        CALL invalid_media_handler
                        SJMP mainloop
;*********************************************************************
go_on_1:                call get_nvm_password_handler
                        JBC card_io_Error,flag_Error_1
                        CALL terminal_ready_handler
                        JB key_in,$
;*********************************************************************
                        CALL get_user_password_handler
                        JBC time_out_error, mainloop
                        CALL get_user_entry_mode_handler
                        JBC card_io_Error,flag_error_1
                        JBC password_valid, go_on_2
                        CALL invalid_password_handler
next_try:               DEC password_retry
                        MOV A, password_retry
                        JNZ wait_Card
                        JBC time_out_error, mainloop
                        CALL invalid_password_handler
                        ACALL long_Delay2
next_try_2:             SJMP MAINLOOP

;*********************************************************************
go_on_2:                CALL access_granted_Handler
                        CALL grant_Access_handler
go_2_loop:              SJMP MAINLOOP

;*********************************************************************
flag_error_1:           CALL card_io_error_handler
                        SJMP $
;*********************************************************************
FLAG_timeout_error:     CALL keypad_timeout_handler
                        SJMP MAINLOOP

;*********************************************************************
write_1_line:           CALL long_delay
                        ACALL long_delay
                        RET
;*********************************************************************
card_io_error_handler:  SETB GREEN_LED
                        SETB YELLOW_LED
                        CLR RED_LED
```

```
                        SJMP write_1_line

;*****************************************************************
terminal_ready_handler: ACALL long_delay
                        ACALL long_Delay
                        CLR green_led
                        SETB yellow_led
                        SETB red_led
                        RET
;*****************************************************************

keypad_timeout_handler: SJMP write_1_line
;*****************************************************************

terminal_busy_handler:  CLR yellow_led
                        SETB red_led
                        SETB green_led
                        ACALL LONG_DELAY
                        JMP write_1_line
;*****************************************************************
access_granted_handler: SETB green_led
                        CLR yellow_led
                        SETB red_led
                        ACALL LONG_DELAY
                        SJMP Write_1_line


;*****************************************************************

grant_access_handler:   CLR motor_dx1
                        SETB motor_dx2
                        ACALL LONG_DELAY
                        ACALL LONG_DELAY
                        ACALL long_Delay
                        SETB motor_dx1
                        CLR motor_dx2
                        ACALL LONG_dELAY
                        ACALL long_delay
                        ACALL long_Delay
                        SETB motor_dx1
                        SETB motor_dx2
                        RET


;*****************************************************************
invalid_media_handler:  CLR GREEN_LED
                        SETB YELLOW_LED
                        CLR RED_LED
                        SJMP write_1_line
;*****************************************************************
invalid_password_handler:CLR red_led
                        SETB green_led
                        SETB yellow_Led
                        SJMP write_1_line
;*********************************************************
get_nvm_password_handler:
```

```
                        MOV R0,#card_data_buffer
                        MOV R1,#10
                        MOV address,#0
get_card_passworD_loop: ACALL read
                        JC card_read_Error
                        ACALL decrypt_data
                        MOV @R0, data_Read
                        INC R0
                        INC address
                        DJNZ R1,get_Card_password_loop
                        CLR C
card_read_Error:        RET
;*********************************************************
write_password:         MOV R0,#keypad_Data_buffer
                        MOV address,#0
write_passworD_loop:    MOV A, @R0
                        CALL encrypt_data
                        MOV data_2_write, A
                        CALL write
                        JB card_io_error, WRITE_IO_eRROR
                        CALL read
                        JB card_io_error, write_io_Error
                        CALL decrypt_data
                        MOV A, data_Read
                        XRL A, @R0
                        JNZ write_io_error
                        INC R0
                        INC address
                        CJNE R0,#keypad_Data_buffer+10, write_password_loop
                        MOV address,#7Eh
                        MOV data_2_write,#0
                        CALL encrypt_Data
                        MOV data_2_Write, A
                        CALL write
                        JB card_io_Error, WRITE_IO_eRROR
                        CLR C
write_io_error:         RET
;*****************************************************************
encrypt_Data:    XRL A, #key
                        XRL A, address
                        DEC address
                        XRL A, address
                        INC address
                        INC address
                        XRL A, address
                        DEC address
                        MOV data_2_Write, A
                        RET

;*********************************************************************
 decrypt_Data:   MOV A,data_read
                        INC address
                        XRL A, address
                        DEC address
                        DEC address
                        XRL A, address
```

```
                    INC address
                    XRL A, address
                    XRL A, #key
                    MOV data_read, A
                    RET

;*********************************************************
get_user_entry_mode_handler:ACALL parse_user_password
                    JB error, exit_chk_pgm
                    JB chk_pgm_mode, chk_program
                    JB chk_password_mode, chk_password
exit_chk_pgm:       CLR passworD_Valid
                    RET

chk_program:        MOV R0,#keypad_Data_buffer
                    MOV A,@R0
                    CJNE A,#prog_mode_header_byte, exit_chk_pgm
                    MOV A,#1
                    INC R0
chk_pgm_loop:       XRL A, @R0
                    JNZ exit_chk_pgm
                    MOV A,@R0
                    NC R0
                    INC A
                    CJNE R0,#keypad_Data_buffer+4,chk_pgm_loop
                    MOV A,#prog_mode_exit_byte
                    XRL A,@R0
                    JNZ exit_chk_pgm
pgm_mode_proper:    ACALL get_old_user_password_handler
                    JB error, exit_chk_pgm
                    ACALL filter_user_password
                    JB error, exit_chk_pgm
                    ACALL compare_password
                    JNB password_Valid, exit_chk_pgm
                    ACALL get_new_user_password_handler
                    JB error, exit_chk_pgm
                    ACALL filter_user_password
                    JB error, exit_chk_pgm
                    ACALL write_password
                    JC exit_here_3
                    CLR C
                    SETB password_valid
exit_here_3:        RET
;*********************************************************
filter_user_password:   MOV R0,#keypad_data_buffer
                    CLR error
loop:               MOV A,@R0
                    CJNE A,#9, chk_2
skip_back_filter_user_password:INC R0
                    CJNE R0,#keypad_Data_buffer+10,loop
                    CLR C
                    RET
chk_2:              JC skip_Back_filter_user_password
                    SETB error
                    CLR C
                    RET
```

```
;*************************************************************

chk_password:           CLR password_valid
                        CALL filter_user_password
                        CLR C
                        JB error, exit_chk_password
                        ACALL compare_password
exit_chk_password:      RET

;*********************************************************
GET_NEW_USER_PASSWORD_HANDLER:MOV C, GREEN_LED
                        CPL GREEN_LED
                        ACALL LONG_DELAY
                        CPL GREEN_LED
                        ACALL LONG_DELAY
                        CPL GREEN_LED
                        ACALL LONG_DELAY
                        CPL GREEN_LED
                        ACALL LONG_DELAY
                        CPL GREEN_LED
                        ACALL LONG_DELAY
                        MOV GREEN_LED, C
                        ACALL READ_KEYPAD
                        RET

;*********************************************************
get_old_user_password_handler:MOV C, YELLOW_LED
                        CPL YELLOW_LED
                        ACALL LONG_DELAY
                        CPL YELLOW_LED
                        ACALL LONG_DELAY
                        CPL YELLOW_LED
                        ACALL LONG_DELAY
                        CPL YELLOW_LED
                        ACALL LONG_DELAY
                        CPL YELLOW_LED
                        ACALL LONG_DELAY
                        MOV YELLOW_LED, C
                        ACALL READ_KEYPAD
                                RET
;*********************************************************
parse_user_password:    CLR error
                        CLR chk_pgm_mode
                        CLR chk_password_mode
                        MOV A,R0
                        CLR C
                        SUBB A,#18
                        JZ EXIT_parse_user_password_1
                        MOV temp,A
                        XRL A,#9
                        JZ exit_parse_user_password_2
                        MOV A, temp
                        XRL A, #5
                        JZ exit_parse_user_password_3
exit_parse_user_password_1: SETB error
                                RET
```

```
exit_parse_user_password_2:     SETB chk_password_mode
                                RET
exit_parse_user_password_3:     SETB chk_pgm_mode
                                RET

;*************************************************************
get_user_password_handler:      ACALL read_keypad
                                CLR no_key
                                RET
;*************************************************************
read_keypad:            CLR error
                        CLR time_out_error
                        MOV R0,#keypad_Data_buffer
t_out_3:                MOV R7,#20
t_out_2:                MOV R6,#0
t_out_1:                MOV R5,#0
wait_key1:              JNB key_in, wait_key
                        DJNZ R5, wait_key1
                        DJNZ R6, t_out_1
                        DJNZ R7, t_out_2
                        SETB time_out_error
                        RET

reload_r7:              MOV R7,#15
reload_r6:              MOV R6,#0
reload_r5:              MOV R5,#0
wait_key:               JNB key_in, go_get_key
                        DJNZ R5, wait_key
                        DJNZ R6, reload_r5
                        DJNZ R7, reload_r6
                        SETB error
                        RET
;*************************************************************
go_get_key:             ACALL read_key_code
                        MOV temp, A
chk_delete:             XRL A, #delete_Flag
                        JZ delete_pressed
                        MOV A, temp
                        MOV @R0, A
                        INC R0
                        CLR sounder_dx
                        MOV R2,#35
again_x:                ACALL dly_2ms
                        ACALL dly_2ms
                        ACALL dly_2ms
                        ACALL dly_2ms
                        DJNZ R2, again_X
                        SETB sounder_dx
                        CJNE R0,#keypad_Data_buffer+10, reload_r7
                        DEC R0
                        CLR error
                        RET
;*************************************************************
delete_pressed:         ACALL long_delay
                                SJMP read_keypad
                                RET
```

```
;*************************************************************
read_key_code:  MOV key_code,#10h
                JB row_1, skip_1
                MOV key_code,#0
skip_1:         JB row_2, skip_2
                MOV key_code,#4
skip_2:         JB row_3, skip_3
                MOV key_code,#8
skip_3:         JB row_4, flip_bits
                MOV key_code,#12
flip_bits:      MOV p3,#11110000b
                ACALL settle_Delay
                JB col_1, skip_5
                MOV temp,#0
skip_5:         JB col_2, skip_6
                MOV temp,#1
skip_6:         JB col_3,skip_7
                MOV temp,#2
skip_7:         JB col_4,compute_key
                MOV temp,#3
compute_key:    MOV p3,#00001111b
                MOV A, key_code
                XRL A,#10h
                JZ no_key
                MOV A, key_code
                ADD A,temp
                MOV DPTR,#xlate_table
                MOVC A,@a+dptr
no_key:         RET

xlate_table:    DB 1,2,3,15,4,5,6,14,7,8,9,13,10,0,11,12
;*********************************************************
settle_delay:   MOV R7,#0
                DJNZ R7,$
                RET
;*********************************************************

compare_password:               MOV R0,#keypad_data_buffer
                                MOV R1,#carD_data_buffer
                                CLR password_valid
compare_loop:           MOV A,@R1
                                XRL A,@R0
                                JNZ error_on_compare
                                INC R0
                                INC R1
                                CJNE R0,#keypad_Data_Buffer+10, compare_loop
                                SETB password_valid
error_on_compare:               RET

;*********************************************************
Get_format_status_handler: CLR format_valid
                                MOV address, #07fh
                                ACALL read
                                JB card_io_Error, exit_Here
                                ACALL decrypt_data
                                MOV A, data_read
```

```
                        XRL A, #format_key
                        JNZ exit_here
                        SETB format_valid

exit_here:              RET

;*********************************************************
init_system_handler:    SETB motor_dx1
                        SETB motor_dx2
                        SETB data_out
                        MOV p3,#00001111b
                        SETB key_in
                        ACALL CLEAR_rAM
                        RET
;*********************************************************
CLEAR_rAM:              MOV R0,#08H
                        CLR A
CLEAR_rAM_LOOP:         MOV @R0, A
                        INC R0
                        CJNE R0,#60H, CLEAR_rAM_LOOP
                        RET

;*********************************************************
dly_2ms:                MOV R7,#0
dly_2ms_lp:             NOP
                        NOP
                        NOP
                        NOP
                        NOP
                        NOP
                        NOP
                        NOP
                        NOP
                        NOP
                        NOP
                        NOP
                        DJNZ R7,dly_2ms_lp
                        RET
;*************************************************
dly_100us:      MOV R7,#100
                DJNZ R7,$
                RET

;*************************************************
long_Delay:             MOV R6,#0
long_delay_lp:  ACALL dly_2ms
                        DJNZ R6, long_delay_lp
                        RET
;*************************************************
dly_5_sec:              MOV R5,#10
loop_t:         ACALL long_Delay
                        DJNZ R5, loop_t
                        RET

;*************************************************
write:                  CLR card_io_error
```

```
                        ACALL i2c_Start
                        MOV A, #slave_Address
                        ORL A, #write_flag
                        CALL write_byte
                        JC write_Abort
                        MOV A, address
                        CALL write_byte
                        JC write_Abort
                        MOV A, data_2_Write
                        CALL write_byte
                        JC write_Abort
                        CLR C
                        CALL i2c_Stop
                        call write_time_out
                        RET

write_abort:        SETB card_io_Error
                        CALL i2c_Stop
                        CALL write_time_out
                        ret
;*********************************************************

read:                   CLR card_io_error
                        CALL i2c_Start
                        MOV A, #slave_Address
                        ORL A, #write_flag
                        CALL write_byte
                        JC read_Abort
                        MOV A, address
                        CALL write_byte
                        JC read_abort
                        CALL i2c_Start
                        MOV A, #slave_Address
                        ORL A, #read_flag
                        CALL write_byte
                        JC read_Abort
                        CALL read_byte
                        MOV data_Read, A
                        CALL no_Ack
                        CLR C
                        CALL I2C_STOP
                        RET
read_Abort:             SETB card_io_Error
                        CALL i2c_Stop
                        RET
;*********************************************

i2c_start:              SETB data_out
                        SETB clock
                        CALL dly_7us
                        CLR data_out
                        LCALL dly_5us
                        CLR clock
                        CALL dly_7us
                        CLR C
                        RET
```

```
;*************************************************
i2c_stop:                       CLR data_out
                                CALL dly_5us
                                SETB clock
                                CALL dly_7us
                                SETB data_out
                                RET
;*************************************************

write_byte:                     MOV R7,#8
write_loop:                     RLC A
                MOV data_out, C
                NOP
                NOP
                SETB clock
                CALL dly_7us
                CLR clock
                CALL dly_7us
                DJNZ R7, write_loop
                SETB data_out
                NOP
                NOP
                NOP
                SETB clock
                NOP
                NOP
                MOV C, data_out
                CLR clock
                RET

;*************************************************
read_byte:      MOV R7,#8
                SETB data_out
read_loop:      NOP
                SETB clock
                CALL dly_7us
                SETB data_out
                NOP
                NOP
                MOV C, data_out
                RLC A
                NOP
                NOP
                CLR clock
                CALL dly_5us
                DJNZ R7, read_loop
                MOV data_Read, A
                RET
;*************************************************
write_time_out: MOV R7,#30
reload:         MOV R6,#250
                        DJNZ R6,$
                        DJNZ R7, reload
                        RET
;*************************************************
```

```
no_Ack:         SETB data_out
                NOP
                NOP
                SETB clock
                NOP
                NOP
                NOP
                CLR clock
                RET
;*************************************************************
dly_7us:        NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                RET
;*************************************************************
dly_5us:        NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                RET
;*********************************************************

long_Delay2:    ACALL long_Delay
                ACALL long_Delay
                ACALL long_Delay
                RET
;*********************************************************

init_nvm:
init_now:                   MOV address,#0
clear_Again:            MOV data_2_write, address
                            MOV A, data_2_Write
                            ACALL encrypt_Data
                            ACALL write
                            JC ACK_ERROR_2
                            INC address
                            MOV R2, address
                            CJNE R2,#last_address, clear_Again
                            MOV Address, #127
                            MOV A, #format_flag
                            MOV data_2_Write, A
                            ACALL encrypt_data
                            MOV data_2_Write, a
                            ACALL write
                            JC ack_error_2
                            MOV A, #80h
                            MOV address,#126
                            ACALL encrypt_Data
                            ACALL write
                            JC ack_error_2
```

```
                        RET
;*****************************************************************
ACK_ERROR_2:    CALL card_io_error_handler
                CALL long_delay2
                CALL long_Delay2
                LJMP START


;*****************************************************************
make_sound:     CLR sounder_dx
                ACALL dly_2ms
                SETB sounder_dx
                RET
```