This page is intentionally left blank

For further details

Contact *sivamskr@gmail.com*

HARDWARE PIRACY LOCK SYSTEM

1. ABSTRACT

The main problem faced by software industry is piracy. We are providing the basic solution for avoiding the piracy by providing an embedded circuit which will be attached to the serial port of a PC. The software will be having a driver which will be checking the embedded hardware repeatedly. If it doesn't get the expected response from the lock it will just halt the software and exits.

We will write a program in the lock which will apply a predefined algorithm on the sent request variable and send the result to the software. The software will apply the reverse algorithm on the result and it should get the original value. The only it will allow the software to run otherwise it will halt the execution.

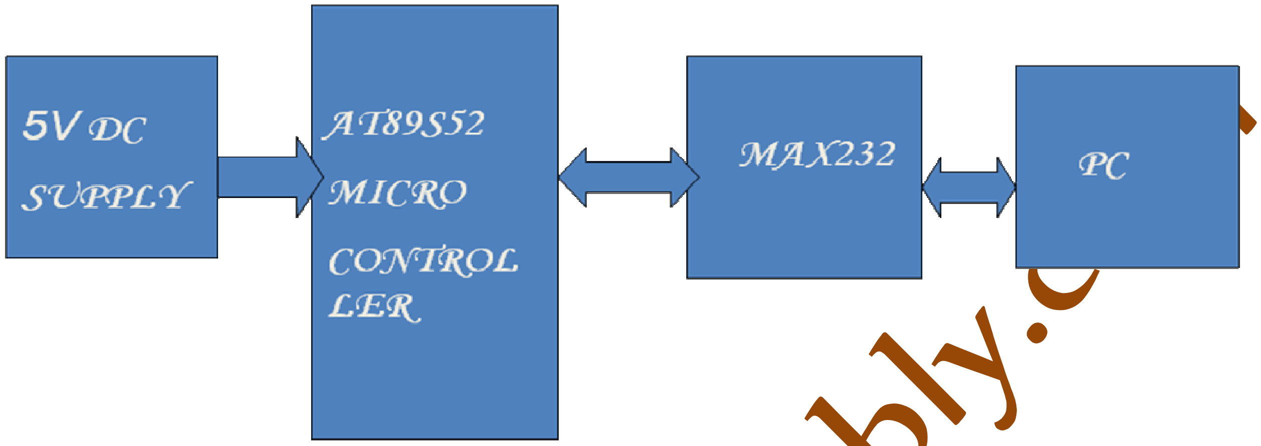
We will provide the driver program libraries to the software vendors so that they can easily incorporate these libraries in their software. This type of implementation avoids not only piracy but also we can prevent usage of the software in more than one system.

2. INTRODUCTION

The basic idea of hardware piracy lock system is to avoid the piracy by providing an embedded circuit is connected to PC. Hard ware lock is committed to maintaining the integrity of software license agreement, it is the most painless, least obtrusive means of assuring compliances with the agreement. Users can make backup copies , load software onto hard disks, run from net work file servers, and have copies of the program installed on multiple machines. Only a device like the hardware lock allows such flexibility and freedom of file duplication while enforcing the license agreement. By protecting the license agreement, we protect our profit margins, which help us grow quickly enough to meet the explosive demand for software and related products. The lock protects legitimate users from the unfair competitive advantage exacted by dishonest users, who would like to run unauthorized copies widespread software piracy otherwise tens to cause price inflation to recoup revenue lost to illegal freebies. By stopping the proliferation of unauthorized copies.

The hard ware lock implementation avoids not only piracy but also we can prevent usage of the software in more than one system.

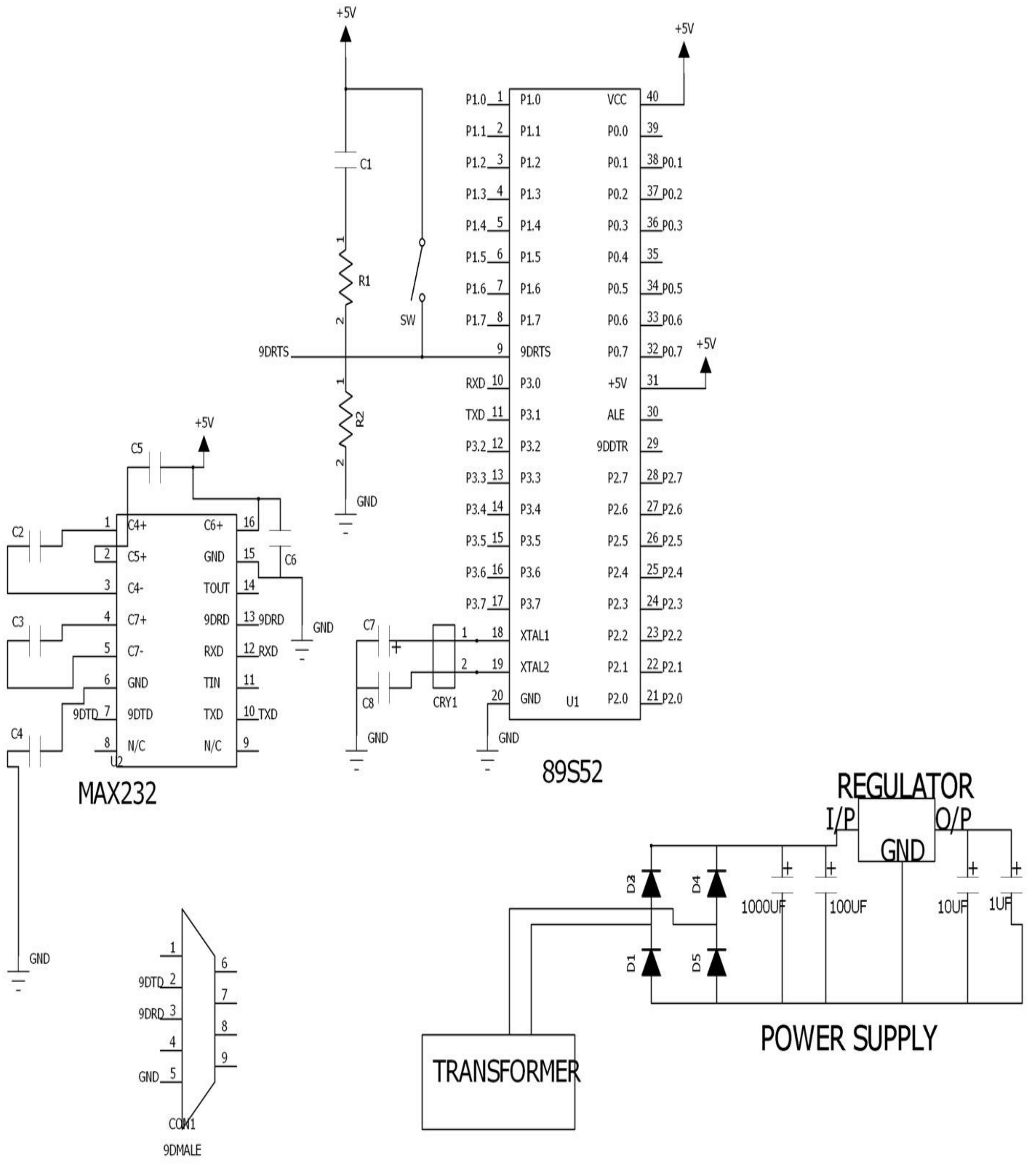
3. DESCRIPTION OF PROJECT



The hardware piracy lock system block diagram is as shown above it consists of power supply, microcontroller, MAX232, PC. From power supply we can give to micro controller and MAX232. Microcontroller is connected to PC through MAX232 and DB9 connecter. The AT89S52 is a low-power, high-performance CMOS 8-bit microcontroller with 8K bytes of in-system programmable Flash memory . The device is manufactured using Atmel's high-density non-volatile memory technology and is compatible with the industry-standard 80C51 instruction set and pin out . The on-chip Flash allows the program memory to be reprogrammed in-system or by a conventional non-volatile memory programmer .The microcontroller receives power supply from power supply unit. By using keil u vision IDE we can write the hard ware programming and dumped in to the microcontroller. It has 4 ports of 8 bit bidirectional (0-7) I/O port with pull-ups. By using any one of the port microcontroller connected to MAX232. It is One of the easiest and most frequently used solutions to the voltage differences for TTL/CMOS and Rs-232 is theMAX-232 chip from MAXIM semiconductors .This chip converts 5-volts TTL/CMOS signal to a minimum of +8 to-8volts.There are several variations of these chips .The two biggest difference are the number of RS-232 interfaces and the use of internal or external capacitors .For our need we will use the basic MAX-232 chip , Each level converter handles the conversion of two TTl/CMOS lines .We will use the first for data transmit & receive and the second for CTS/RTS handshaking.

The MAX-232 is compatible chip as a direct replacement for the TI chip since they all share the same pinouts. Power& ground connectors are not same on all versions of the MAX-232 chips. MAX232 is connected to serial port PC through DB9 connecter. Whenever the soft ware is running in the system The software will be having a driver which will be checking the embedded hardware repeatedly. The hardware coading is existing in the microcontroller, microcontroller gives positive acknowledgement to PC, it allows the running of software. If it doesn't get the expected response from the lock it will just halt the software and exits.

CIRCUIT DIAGRAM:



4. HARDWARE DESCRIPTION

4.1MICROCONTROLLERS

The past three decades have seen the introduction that has radically changed the way in which we analyze and control the world around us. Born of parallel developments in computer architecture and integrated circuit fabrications, the microprocessor or computer on chip first becomes a commercial reality in 1971 with the introduction of the 4 bit 4004 by a small, unknown company by the name of Intel corporation other,well established, semiconductor firms soon followed Intel's pioneering technology so that by the late 1970,s we could choose from a half dozen or so micro processor types.

The 1970's also saw the growth of the number of the personal computers users from a handful of hobbyists and hackers to millions of business, industrial, governmental, defense, and educational and private users now enjoying the advantages of inexpensive computing.

A bye product of microprocessor development was the micro controller. The same fabrications techniques and programming concepts that make possible general-purpose microprocessor also yield the micro controller.

The criteria in choosing micro controller are as follows:

1. Meeting the computing needs of the task at hand efficiently and cost effectively.
2. Availability of software development tools such as compilers, assemblers and debuggers and
3. Wide availability and reliable sources of the micro controller.

Among other considerations in this category are:

a. Speed and packaging
b. Power consumption. This is especially critical for battery-powered products.
c. The amount of RAM and ROM on chip.
d. The number of I/O pins and the timer on the chip.
e. It is easy to upgrade to higher performance or lower power consumption versions.
f. Cost per unit.

Microprocessors and micro controller stems form the same basic idea, microprocessor is a general purpose digital computer central processing unit (CPU) popularly known as memory usually ROM, RAM, "computer on chip". To make a complete micro computer, one must add memory ,usually ROM,RAM Memory decoders , an isolator and number of I/O devices, such as parallel and serial data ports . The design of microcontroller added all these features along with ALU, PC, SP and registers.

The prime use of microprocessor is to read data, perform extensive calculations on that data and store those calculations on a mass storage device or display the results for human use. Like the microprocessor, a microcontroller is a general purpose device, but one that is meant to read data, perform limited calculations on that data and control its environment based on those calculations the prime use of micro controller is to control the operation of a machine using a fixed program that is stored in ROM and that does not change over the life time of the system.

The contrast between a micro controller and a microprocessor is exemplified by the fact that most microprocessors have many operational codes (OPCODES) for moving data from external memory to the CPU. Micro controllers may have one or two. Microprocessor may have one or two types of bit handling instructions; microcontrollers will have many .The microprocessor is concerned with rapid movement of code and data from external addresses to the chip; the micro controller is concerned with rapid movement of bits with in the chip. The micro controller can function as a computer with the addition of no external digital parts; the microprocessor must have many additional parts to be operational.

AT89S52 MICROCONTROLLER

Description: The AT89S52 is a low-power, high-performance CMOS 8-bit microcontroller with 8Kbytes of in-system programmable Flash memory. The device is manufactured using Atmel's high density nonvolatile memory technology and is compatible with the industry standard 80C51 micro controller. 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 AT89S52 is a powerful microcontroller which provides a highly flexible and cost effective solution to many embedded control applications

FEATURES:

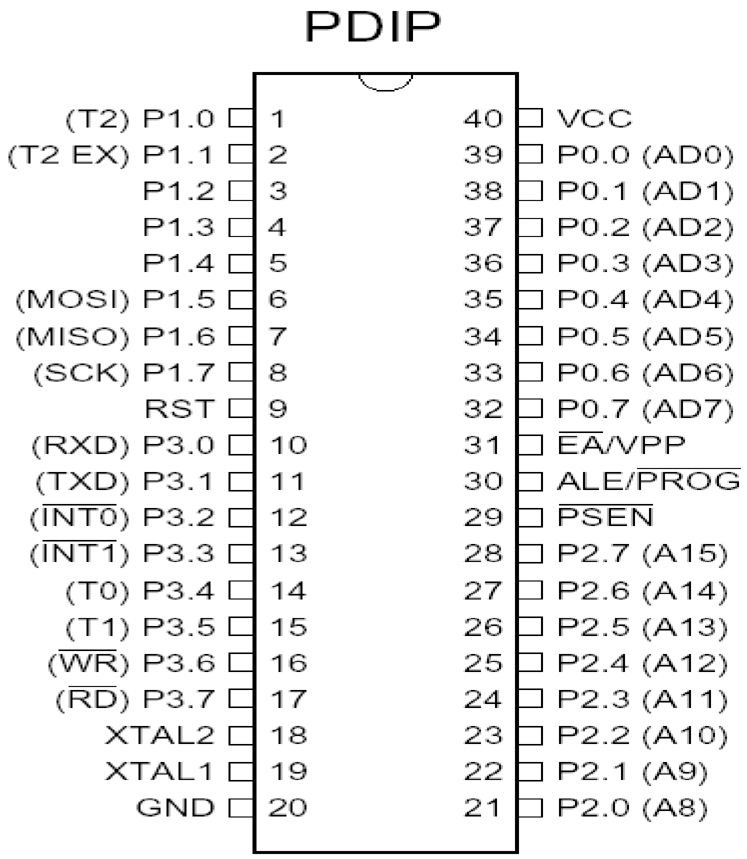
- Compatible with MCS-51 Products
- 8K Bytes of In-System Programmable (ISP) Flash Memory
  – Endurance: 1000 Write/Erase Cycles.

- 4.0V to 5.5V Operating Range
- Fully Static Operation: 0 Hz to 33 MHz
- Three-level Program Memory Lock
- 256K Internal RAM
- 32 Programmable I/O Lines
- Three 16-bit Timer/Counters
- Eight 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

The AT89S52 provides the following standard features: 8K bytes of Flash, 256 bytes of RAM, 32 I/O lines, Watchdog timer, two data pointers, three 16-bit timer/counters, full duplex serial port, on-chip oscillator,and clock circuitry. In addition, the AT89S52 is designed with static logic for operation down to zero frequency and supports two software selectable power saving modes.The Idle Mode stops the CPU while allowing the RAM, timer/counters, serial port, and interrupt system to continue functioning. The Power-down mode saves the RAM contents but freezes the oscillator, disabling all other chip functions until the next interrupt or hardware reset.

PIN DIAGRAM



PIN DESCRIPTION

VCC : Supply voltage.
GND : Ground.
Port 0 : Port 0 is an 8-bit open drain bi-directional I/O port as an output port, each pin can sink eight TTL inputs. When 1s are written to port 0 pins, the pins can be used as high impedance inputs. Port 0 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 : Port 1 is an 8-bit bidirectional 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 pullups 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 low order address bytes during Flash programming and verification.

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)
P1.5	MOSI (used for In-System Programming)
P1.6	MISO (used for In-System Programming)
P1.7	SCK (used for In-System Programming)

Port 2 : 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 use 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 : 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 AT89S52, 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	$\overline{\text{INT0}}$ (external interrupt 0)
P3.3	$\overline{\text{INT1}}$ (external interrupt 1)
P3.4	T0 (timer 0 external input)
P3.5	T1 (timer 1 external input)
P3.6	$\overline{\text{WR}}$ (external data memory write strobe)
P3.7	$\overline{\text{RD}}$ (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. Address Latch Enable (ALE) 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 of1/6 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 duringa 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.

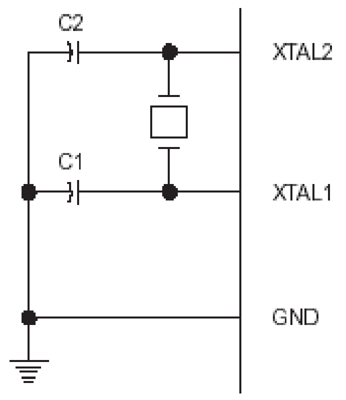
PSEN : Program Store Enable (PSEN) is the read strobe to external program memory .When the AT89S52 is executing code from external program memory , PSEN is activated twice each machine cycle , except that two PSEN activations are skipped during each access to external data memory.

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.

XTAL1 : Input to the inverting oscillator amplifier and input to the internal clock operating circuit.**XTAL2** Output from the inverting oscillator amplifier

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, as shown in Figure 1. Either a quartz crystal or ceramic resonator may be used. To drive the device from an external is driven, as shown in Figure clock source; XTAL2 should be left unconnected while XTAL1 is driven as shown in Figure 2. There are no requirements on the duty cycle of the external clock signal, since the input to the internal clocking circuitry is through a divide-by-two flip-flop, but minimum and maximum voltage high and low time specifications must be observed.



Note: C1, C2 = 30 pF ± 10 pF for Crystals
= 40 pF ± 10 pF for Ceramic Resonators

Figure 1. Oscillator Connections

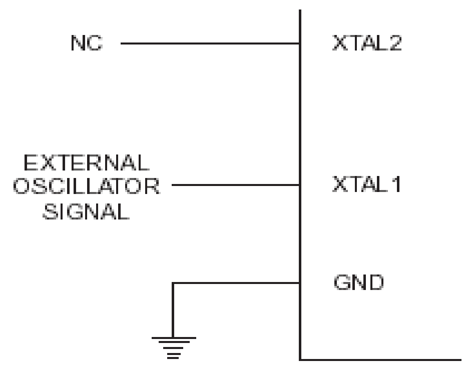
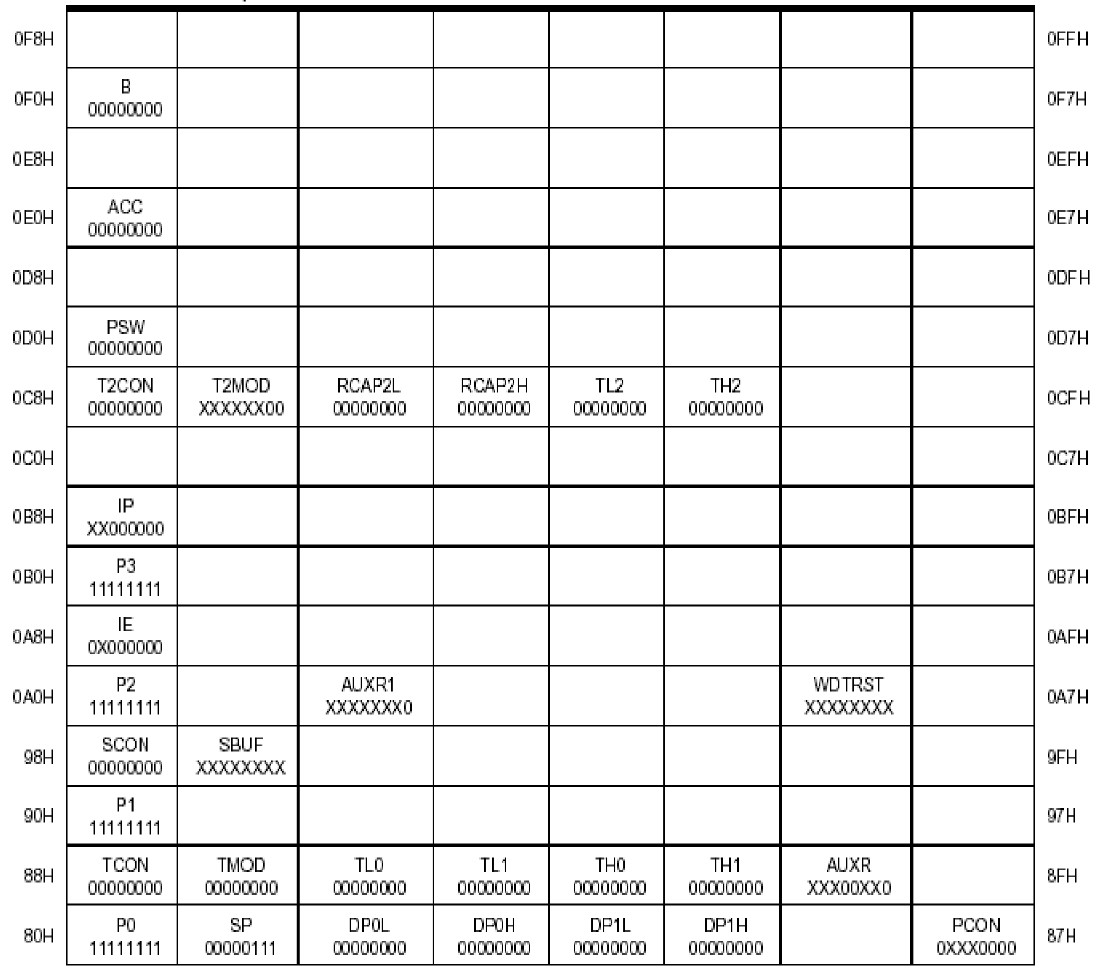


Figure 2. External Clock Drive Configuration

Special Function Register (SFR):-

A map of the on-chip memory area called the Special Function Register (SFR) space is shown in Table 1.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.

Table 1. AT89S52 SFR Map and Reset Values



0F8H									0FFH
0F0H	B 00000000								0F7H
0E8H									0EFH
0E0H	ACC 00000000								0E7H
0D8H									0DFH
0D0H	PSW 00000000								0D7H
0C8H	T2CON 00000000	T2MOD XXXXXX00	RCAP2L 00000000	RCAP2H 00000000	TL2 00000000	TH2 00000000			0CFH
0C0H									0C7H
0B8H	IP XX000000								0BFH
0B0H	P3 11111111								0B7H
0A8H	IE 0X000000								0AFH
0A0H	P2 11111111		AUXR1 XXXXXXX0				WDTRST XXXXXXXX		0A7H
98H	SCON 00000000	SBUF XXXXXXXX							9FH
90H	P1 11111111								97H
88H	TCON 00000000	TMOD 00000000	TL0 00000000	TL1 00000000	TH0 00000000	TH1 00000000	AUXR XXX00XX0		8FH
80H	P0 11111111	SP 00000111	DP0L 00000000	DP0H 00000000	DP1L 00000000	DP1H 00000000		PCON 0XXX0000	87H

Special Function Register (SFR) Memory: -

Special Function Registers (SFR s) are areas of memory that control specific functionality of the 8051 processor. For example, four SFRs permit access to the 8051's 32 input/output lines. Another SFR allows the user to set the serial baud rate, control and access timers, and configure the 8051's interrupt system

REGISTERS: -

The Accumulator:

The Accumulator, as its name suggests is used as a general register to accumulate the results of a large number of instructions. It can hold 8-bit (1-byte) value and is the most versatile register.

The "R" registers:

The "R" registers are a set of eight registers that are named R0,R1. etc up to R7. These registers are used as auxiliary registers in many operations.

The "B" register:

The "B" register is very similar to the accumulator in the sense that it may hold an 8-bit(1-byte)value. The "B" register is only used by two 8051 instructions: MUL AB and DIV AB.

The Data Pointer:

The Data pointer(DPTR) is the 8051's only user-accessible 16 bit(2Bytes) register. The accumulator, "R" registers are all 1-Byte values. DPTR, as the name suggests, is used to point to data. It is used by a number of commands which allow the 8051 to access external memory.

THE PROGRAM COUNTER AND STACK POINTER:

The program counter (PC) is a 2-byte address, which tells the 8051 where the next instruction to execute is found in memory. The stack pointer like all registers except DPTR and PC may hold an 8-bit (1-Byte) value.

ADDRESSING MODES:

An "addressing mode" refers that you are addressing a given memory location. In summary, the addressing modes are as follows, with an example of each:

Each of these addressing modes provides important flexibility.

Immediate Addressing	MOV A, #20 H
Direct Addressing	MOV A, 30 H
Indirect Addressing	MOV A, @R0
Indexed Addressing:	
a. External Direct	MOVX A, @DPTR
b. Code Indirect	MOVC A, @A+DPTR

Immediate Addressing:

Immediate addressing is so named because the value to be stored in memory immediately follows the operation code in memory. That is to say, the instruction itself dictates what value will be stored in memory. For example, the instruction:

MOV A, #20H:

This instruction uses immediate Addressing because the accumulator will be loaded with the value that immediately follows; in this case 20(hexadecimal). Immediate addressing is very fast since the value to be loaded is included in the instruction. However, since the value to be loaded is fixed at compile-time it is not very flexible.

Direct Addressing:

Direct addressing is so named because the value to be stored in memory is obtained by directly retrieving it from another memory location.

For example: MOV A, 30h

This instruction will read the data out of internal RAM address 30(hexadecimal) and store it in the Accumulator. Direct addressing is generally fast since, although the value to be loaded isn't included in the instruction, it is quickly accessible since it is stored in the 8051's internal RAM. It is also much more flexible than Immediate Addressing since the value to be loaded is whatever is found at the given address which may variable. Also it is important to note that when using direct addressing any instruction which refers to an address between 00h and 7Fh is referring to the SFR control registers that control the 8051 micro controller itself.

Indirect Addressing:

Indirect addressing is a very powerful addressing mode which in many cases provides an exceptional level of flexibility. Indirect addressing is also the only way to access the extra 128 bytes of internal RAM found on the 8052. Indirect addressing appears as follows:

MOV A,@R0:

This instruction causes the 8051 to analyze the value of the R0

register. The 8051 will then load the accumulator with the value from

Internal RAM, which is found at the address indicated by R0.

Indirect addressing always refers to Internal RAM; it never refers to an SFR. Thus, in a prior example we mentioned that SFR 99h can be used to write a value to the serial port. Thus one may think that the following would be a valid solution to write the value '1' to the serial port:

```
MOV R0, #99h; Load the address of the serial port
MOV  @R0, #01h; Send 01 to the serial port –WRONG!
```

This is not valid. Since indirect addressing always refers to the internal RAM these two instructions would write the value 01h to internal RAM address 99h on an 8052. On an 8051 these two instructions would produce an undefined result since the 8051 only has 128 bytes of internal RAM.

Indexed Addressing:

Indexed addressing mode is widely used in accessing data elements of look-up table entries located in the program ROM space of the 8051.

Eg: MOVC A, @A+DPTR

The 16-bit register DPTR and register A are used to form the address of the data element stored in on-chip ROM "C" means code.

External Indirect:

External memory can also be accessed using a form of indirect addressing which we can call External Indirect addressing. This form of addressing is usually only used in relatively small projects that have a very small amount of external RAM an example of this addressing mode is:

Eg: MOVX @R0,A .

Memory Organisation:

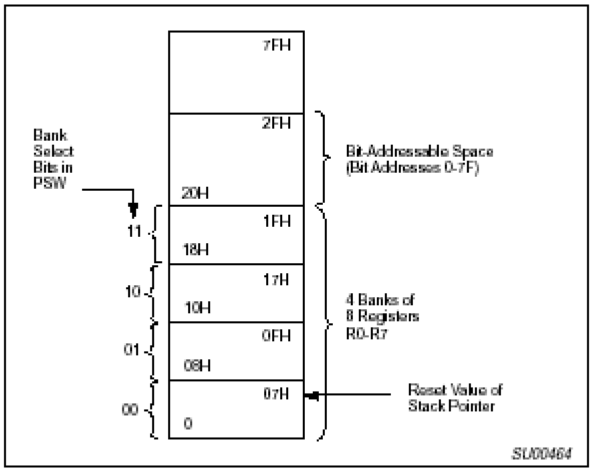

7FH
2FH
20H
1FH
18H
17H
10H
0FH
08H
07H
0
Bank Select Bits in PSW
11
10
01
00
Bit-Addressable Space (Bit Addresses 0-7F)
4 Banks of 8 Registers R0-R7
Reset Value of Stack Pointer
SU00464

Fig: Lower 128 Bytes of Internal RAM

The first 8 bytes (00h-07h)are "register bank 0". By manipulating certain SFRs, a program may choose to use register banks 1, 2 or 3. These alternative register banks are located in internal RAM in address 08h through 1F.

Bit memory also lives and is a part of internal RAM. The 80 bytes remaining of Internal RAM, from address 30h through 7Fh, may be used by user variables that need to be accessed frequently or at high-speed.

This area is also utilized by the micro controller as a storage area for the operating stack. This fact severely limits the 8051's stack since, as illustrated in the memory map, the area reserved for the stack is only 80 bytes-and usually it is less since this 80 bytes has to be shared between the stack and user.

1. External Code memory:

This code (or program) memory that resides off-chip. This is often in the form of an external EPROM. Code memory is the memory that holds the actual 8051 programs that is to be run. This memory is limited to 64K and comes in many shapes and sizes: code memory may be found on-chip, either burned into the micro controller as ROM or EPROM.

Code may also be stored completely off chip in an external ROM or, more commonly, an external EPROM. Flash RAM is also another popular method of storing a program. Various combinations of these memory on chip and code memory off chip in an EPROM.

When the program is stored on chip the 64K maximum is often reduced to 4K, 8K or 16K. This varies depending on the version of the chip that is being used. Each version offers specific capabilities and one of the distinguishing factors from chip to chip is how much ROM/EPROM space the chip has.However, code memory is most commonly implemented as off-chip EPROM. This is especially true in low-cost development systems.

External RAM:

This RAM memory resides off-chip. This is often in the form of standard static RAM or flash RAM. As an obvious opposite of internal RAM, the 8051 also supports what is called External RAM.

As the name suggests, External RAM is any random access memory which is found off-chip. Since the memory is off-chip it is not flexible in terms of accessing, and is also slower. For example, to increment an internal RAM location by 1 requires only 1 instruction and 1 instruction cycle. To increment a 1-byte value stored in external RAM requires 4 instructions and 7 instruction cycles. In this case, external memory is 7 times slower! What External RAM loses in speed and flexibility it gains in quantity. While internal RAM is limited to 128 bytes256 bytes with an 8052), the 8051 supports External RAM up to 64K.

4.2RS-232/UART

RS-232 is a standard developed by Electronics Industry Assosiation (EIA).This is one of the oldest and mostcommunication interfaces. The PC will have two RS-232 ports designated as com1& com2.Most of the microcontrollers have on-chip serial interface.The evaluation to the host system using RS-232. RS-232 is used to connect a DTE(Data Terminal Equipment)to a Data Circuit Terminating Equipment (DCE). A DTE can be a pc a modem, mouse,digitizer or ascanner.RS-232 interface specifies the physical layer interface only.

The specifications describe the physical, mechanical,electrical & procedural characteristics for serial communication . RS-232 is the standard for serial communication i.e. the bits are tr4ansmitted serially. The communication between two devices is in full duplex, i.e. the data transfer can take place in both directions.

RS-232 Connector Configurations:

- 25 pin connector
- 9 pin connector

Port Wiring Notes

```
DB9 Male (Pin Side)              DB9 Female (Pin Side)
DB9 Female (Solder Side)         DB9 Male (Solder Side)
   -------------                 -------------
   \ 1 2 3 4 5 /                   \ 5 4 3 2 1 /
    \ 6 7 8 9 /                     \ 9 8 7 6 /
     ---------                    ---------
```

DB9 Female to DB9 Female Null-Modem Wiring

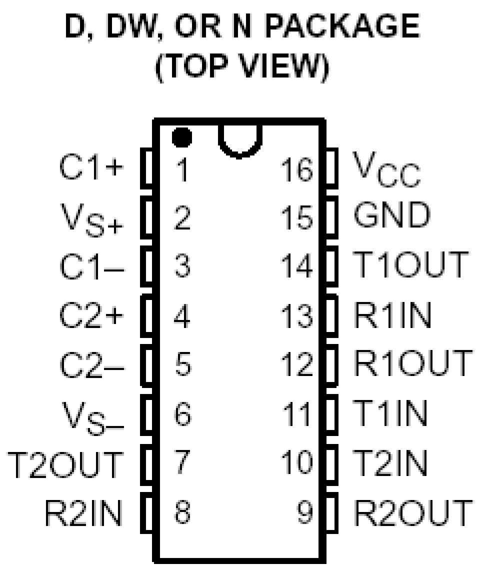
```
 2 | 3 | 7 | 8 | 6&1| 5 | 4
---- ---- ---- ---- ---- ---- ----
 3 | 2 | 8 | 7 | 4 | 5 | 6&1
```

```
9-pin   25-pin  Assignment                From PC
------  ------  ------------------------  ------------
Sheild  1       Case Ground               Gnd
1       8       DCD (Data Carrier Detect)  Input
2       3       RX  (Receive Data)        Input
3       2       TX  (Transmit Data)       Output
4       20      DTR (Data Terminal Ready)  Output
5       7       GND (Signal Ground)       Gnd
6       6       DSR (Data Set Ready)      Input
7       4       RTS (Request To Send)     Output
8       5       CTS (Clear To Send)       Input
9       22      RI  (Ring Indicator)      Input
```

- RTS & DTR are binary outputs that can be manually set and held
- DCD, DSR, CTS, and RI are binary inputs that can be read
- RX & TX can not be set manually and are controlled by the UART
- maximum voltages are between -15 volts and +15 volts
- binary outputs are between +5 to +15 volts and -5 to -15 volts
- binary inputs are between +3 to +15 volts and -3 to -15 volts
- input voltages between -3 to +3 are undefined while output voltages between -5 and +5 are undefined
- positive voltages indicate ON or SPACE, negative voltages indicate OFF or MARK

4.3MAX-232

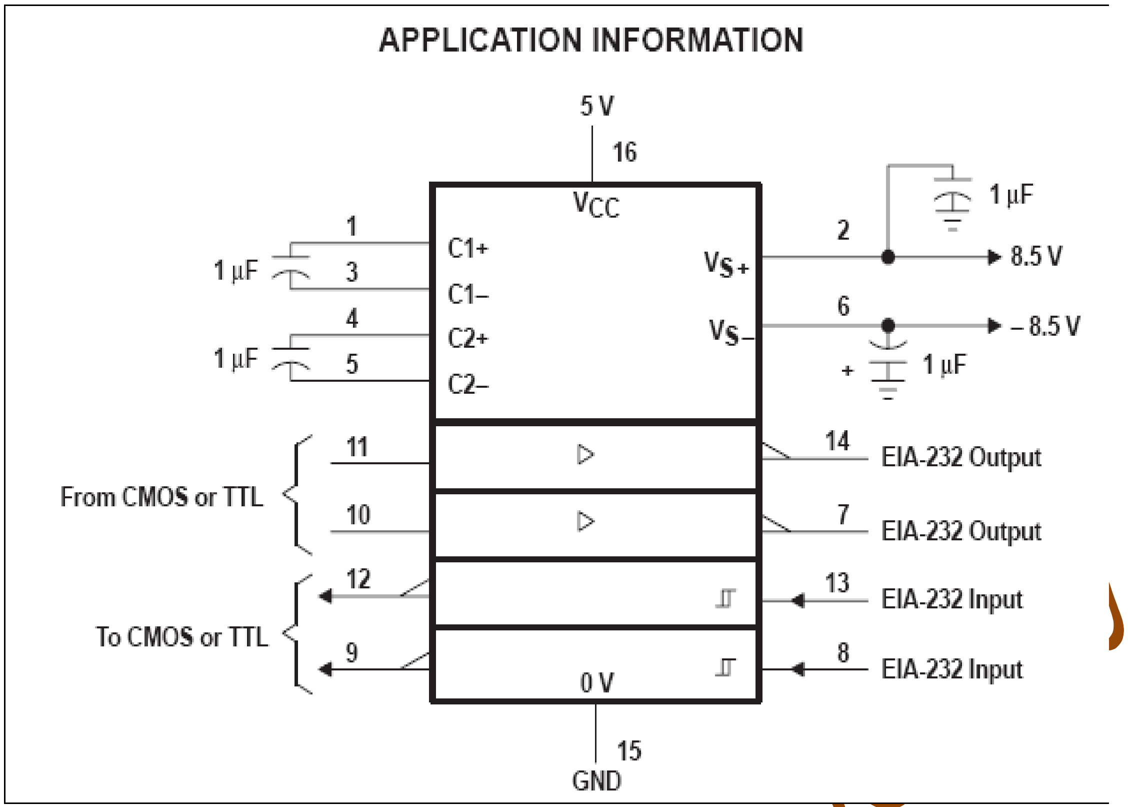
PIN CONFIGURATIONS:



One of the easiest and most frequently used solutions to the voltage differences for TTL/CMOS and Rs-232 is theMAX-232 chip from MAXIM semiconductors .This chip converts 5-volts TTL/CMOS signal to a minimum of +8 to-8volts.There are several variations of these chips .The two biggest difference are the number of RS-232 interfaces and the use of internal or external capacitors .For our need we will use the basic MAX-232 chip , Each level converter handles the conversion of two TTl/CMOS lines .We will use the first for data transmit & receive and the second for CTS/RTS handshaking.

The MAX-232 is compatible chip as a direct replacement for the TI chip since they all share the same pinouts . Power& ground connectors are not same on all versions of the MAX-232 chips.

The MAX-232 chip requires one microfarad capacitors .The MAx-232 is a16-pin chip that has two complete RS-232 transreceivers. These chips work by using the capacitors to boost the signal voltage levels to operate with in the RS-232 signal definitions .This types of circuits is known as charge pump. The capacitor charges and when it reaches its discharge level ,it pumps the higher voltage to the circuits. This chip comes in several different forms,but our purpose we will use the 16-pin dip which is easy to work with and does not require surface mounting.



4.4HARDWARE SOURCE CODE:

```
[.ShellClassInfo]
CLSID={645FF040-5081-101B-9F08-00AA002F954E}
t1isr)();
 void interrupt PORT1INT()  /* Interrupt Service Routine (ISR) for PORT1 */
{
unsigned char  c;
do {
        c = inportb(PORT1 + 5);
   if (c & 1) {
                ch=inportb(PORT1);
               // putchar(ch);
                 buffer[bufferin++]=ch;
                 if(bufferin==1025)
                 bufferin=0;
                 }
 }while (c & 1);
 outportb(0x20,0x20);
}

void main(void)
{
unsigned char c,repflag=0,buf[10],repbuf[20],k=0,okflag=0;
int ran=0,encdata=0,rcvdata=0,j=0,delay1;



 outportb(PORT1 + 1 , 0);       /* Turn off interrupts - Port1 */

 oldport1isr = getvect(INTVECT); /* Save old Interrupt Vector of later
                                   recovery */

 setvect(INTVECT, PORT1INT);     /* Set Interrupt Vector Entry */
                                 /* COM1 - 0x0C */
                                 /* COM2 - 0x0B */
                                 /* COM3 - 0x0C */
                                 /* COM4 - 0x0B */

 /*      PORT 1 - Communication Settings      */

 outportb(PORT1 + 3 , 0x80);  /* SET DLAB ON */
 outportb(PORT1 + 0 , 0x0C);  /* Set Baud rate - Divisor Latch Low Byte */
                              /* Default 0x03 =  38,400 BPS */
                              /*       0x01 = 115,200 BPS */
                              /*       0x02 =  57,600 BPS */
                              /*       0x06 =  19,200 BPS */
                              /*       0x0C =   9,600 BPS */
                              /*       0x18 =   4,800 BPS */
                              /*       0x30 =   2,400 BPS */
 outportb(PORT1 + 1 , 0x00);  /* Set Baud rate - Divisor Latch High Byte */
 outportb(PORT1 + 3 , 0x03);  /* 8 Bits, No Parity, 1 Stop Bit */
 outportb(PORT1 + 2 , 0xC7);  /* FIFO Control Register */
 outportb(PORT1 + 4 , 0x0B);  /* Turn on DTR, RTS, and OUT2 */

 outportb(0x21,(inportb(0x21) & 0xEF));  /* Set Programmable Interrupt Controller */
                                         /* COM1 (IRQ4) - 0xEF  */
                                         /* COM2 (IRQ3) - 0xF7  */
```

```
                                        /* COM3 (IRQ4) - 0xEF  */
                                        /* COM4 (IRQ3) - 0xF7  */

outportb(PORT1 + 1 , 0x01);  /* Interrupt when data received */

       clrscr();
do {

       repflag=0;
       okflag=0;
       j=0;
      ran=rand();
   // printf("\n CHECKING HARDWARE LOCK ");
   // printf("\n SENDING %d",ran);
      encdata=(ran*2-97)/7;
  //  printf("\n EXPECTING %d",encdata);
      sprintf(buf,"%d",ran);
      j=0;
      do{
       outportb(PORT1, buf[j]);
       j++;
       }while(buf[j-1]!=0);
       outportb(PORT1, 0XFE);
       k=0;
      delay(6000);
      do
      {

             if(bufferin!=bufferout)
             {
             c=buffer[bufferout++];
             if(c!=0xfe)
             {
             repbuf[k]=c;
             k++;
             }
             else
             {
             repbuf[k]=0;
             rcvdata=atoi(repbuf);
   //        printf("\n GOT %d",rcvdata);
```

```
                if(rcvdata==encdata)
                okflag=1;
                else
                okflag=0;
                repflag=1;
                }
                }
                if(bufferout==1024)
                bufferout=0;
                  j++;
        } while((j<500)&&(repflag==0));
        if(repflag==0)
        {
        printf("\n HARDWARE LOCK NOT FOUND");
        printf("\n EXITING SOFTWARE       ");
        delay(5000);
        exit(0);
        }
        else
        {
        if(okflag==1)
        {
        okflag=0;
        printf("\n HARDWARE LOCK FOUND SOFTWARE RUNNING");
        }
        else
        {
        printf("\n PYRATED HARDWARE LOCK FOUND SOFTWARE NOT RUNNING");
        }

        }
 }while(!kbhit());
outportb(PORT1 + 1 , 0);      /* Turn off interrupts - Port1 */
outportb(0x21,(inportb(0x21) | 0x10));  /* MASK IRQ using PIC */
                                       /* COM1 (IRQ4) - 0x10  */
                                       /* COM2 (IRQ3) - 0x08  */
                                       /* COM3 (IRQ4) - 0x10  */
                                       /* COM4 (IRQ3) - 0x08  */
setvect(INTVECT, oldport1isr); /* Restore old interrupt vector */
```

}

5. SOFTWARE DESCRIPTION:

5.1 INTRODUCTION:

The C programming language is a general – purpose programming language that provides code efficiency, elements of structured programming , and a rich set of operators . C is not a *big* language and is not designed for any one particular area of application . Its generality combined with its absence of restrictions , makes C a convenient and effective programming solution for a wide variety of software tasks . Many applications can be solved more easily and efficiently with C than with other more specialized languages .

The **C*x*51** Optimizing C Compiler is a complete implementation of the American National Standards Institute (ANSI) standard for the C language . **C*x*51** is not universal C compiler adapted for the 8051 target . It is a ground-up implementation dedicated to generating extremely fast and compact code for the 8051 microprocessor . **C*x*51** provides you with the flexibility of programming in C and the code efficiency and speed of assembly language . Since **C*x*51** is a cross compiler , some aspects of the C Programming language and standard libraries are altered or enhanced to address the peculiarities of a embedded target processor. For optimum support of different 8051 variants , Keil provides the several development tools . A new output file format (OMF2) allows direct support of up to 16MB code and data space . The CX51 compiler is a variant of the C51 compiler that is designed for the new Philips80C51MX architecture

Compiling with the C*x*51Compiler

For compilation we require the following directives. These directives allow you to:

- Direct the **C*x*51** compiler to generate a listing file
- Control the amount of information included in the object file
- Specify optimization level and memory models

Running C*x*51 from the Command Prompt

To invoke the C51 or CX51 compiler , enter **C51** or **CX51** at the command prompt . On this command line , you must include the name of the C source file to be compiled , as well as any other necessary control directives required to compile your source file .
The format for the **C*x*51** command line is:

C51 sourcefile [directives]
CX51 sourcefile [directives]

:
.
.
.

Control Directives

*The **Cx51** compiler offers a number of control directives that you may use to control compilation . Directives are composed of one or more letters or digits and, unless otherwise specified , can be specified after the filename on the command line or within a source file using the **#pragma** directive.*

Directive Categories

Control directives can be divided into three groups: source controls, object controls , and listing controls.

- <u>Source controls</u> define macros on the command line and determine the name of the file to be compiled.
- <u>Object controls</u> affect the form and content of the generated object module (***.OBJ**). These directives allow you to specify the optimizing level or include debugging information in the object file.
- <u>Listing controls</u> govern various aspects of the listing file (***.LST**), in particular its format and specific content.

Function Declarations

The **Cx51** compiler provides a number of extensions for standard C function declarations . These extensions allow you to:

- Specify a function as an interrupt procedure
- Choose the register bank used
- Select the memory model
- Specify reentrancy
- Specify alien (PL/M-51) functions

Bit-addressable Objects:

Bit-addressable objects are objects that can be addressed as words or as bits. Only data objects that occupy the bit-addressable area of the 8051 internal memory fall into this category . The **Cx51** compiler places variables declared with the **bdata** memory type into this bit-addressable area . Furthermore , variables declared with the **bdata** memory type must be global . You may declare these variables as shown below:

int bdata ibase; /* Bit-addressable int */
char bdata bary [4]; /* Bit-addressable array */
The variables **ibase** and **bary** are bit-addressable . Therefore , the individual bits of these variables may be directly accessed and modified . Use the **sbit** keyword to declare new variables that access the bits of variables declared using **bdata**.
For example:
sbit mybit0 = ibase ^ 0; /* bit 0 of ibase */
sbit mybit15 = ibase ^ 15; /* bit 15 of ibase */
sbit Ary07 = bary[0] ^ 7; /* bit 7 of bary[0] */
sbit Ary37 = bary[3] ^ 7; /* bit 7 of bary[3] */

Special Function Registers:
The 8051 family of microcontrollers provides a distinct memory area for accessing Special Function Registers (SFRs). SFRs are used in your program to control timers , counters , serial I/Os , port I/Os , and peripherals . SFRs reside from address 0x80 to 0xFF and can be accessed as bits , bytes , and words . Within the 8051 family, the number and type of SFRs vary . Note that no SFR names are predefined by the **Cx51** compiler. However , declarations for SFRs are provided in include files. The **Cx51** compiler provides you with a number of include files for various 8051 derivatives. Each file contains declarations for the SFRs available on that derivative . The **Cx51** compiler provides access to SFRs with the **sfr** , **sfr16** , and **sbit** data Types . The following sections describe each of these data types .
SFR:
SFRs are declared in the same fashion as other C variables . The only difference is that the data type specified is **sfr** rather than **char** or **int** .
For example:
sfr P0 = 0x80; /* Port-0, address 80h */
sfr P1 = 0x90; /* Port-1, address 90h */
sfr P2 = 0xA0; /* Port-2, address 0A0h */
sfr P3 = 0xB0; /* Port-3, address 0B0h */

SFR16:
Many of the newer 8051 derivatives use two SFRs with consecutive addresses to specify 16-bit values . For example , the 8052 uses addresses 0xCC and 0xCD for the low and high bytes of timer / counter 2 . The **Cx51** compiler provides the **sfr16** data type to access 2 SFRs as a 16-bit SFR.
Access to 16-bit SFRs is possible only when the low byte immediately precedes the

high byte . The low byte is used as the address in the **sfr16** declaration .
For example:
sfr16 T2 = 0xCC; /* Timer 2: T2L 0CCh, T2H 0CDh */
sfr16 RCAP2 = 0xCA; /* RCAP2L 0CAh, RCAP2H 0CBh */

SBIT:
With typica l 8051 applications , it is often necessary to access individual bits within an SFR . The **C*x*51** compiler makes this possible with the **sbit** data type which provides access to bit-addressable SFRs and other bit-addressable objects.

5.2SOURCE CODE:
```
#include<serialheader.h>
#include<reg52.h>
#include<stdio.h>
#include<stdlib.h>


sbit LED1 = P1^5;
sbit LED2 = P1^6;
sbit LED3 = P1^7;


void main(void)
{
unsigned char buf[20],c,repbuf[10],d,i,comflag=0;
int j=0;
LED1=0;
LED2=0;
LED3=0;
serial_init(0xfd);
while(1)
{
LED1=0;
LED2=0;
LED3=0;
i=0;
comflag=0;
do
{


do
{
LED1=1;
d=getc(&c);

}while(d==0);
LED1=0;
if(c!=0xfe)
{
buf[i]=c;
i++;
}
else
{
buf[i]=0;
comflag=1;
j=atoi(buf);
j*=2;
j-=97;
j/=7;
sprintf(repbuf,"%d",j);
put_str(repbuf);
putc(0xfe);
}
}while(comflag==0);
comflag=0;
}

}
```

6. CONCLUTION

In order to avoid the piracy of software a new hardware lock is implemented based on microcontroller in this study. Running of the software completely dependent on the hardware which is existing in the microcontroller. The hardware lock system has following advantages: protection of license agreement, it is not copy protection, Fair compensation, protects users' investments, Realistic pricing, Wide user acceptance, High reliability/low failure rate