STORING DATA ON I²C EEPROM USING AVR MICROCONTROLLER

■ ARUN DAYAL UDAI

Microcontroller-based applications like robotics are often limited by the small storage space of microcontrollers. Implementing applications like artificial intelligence requires a large storage space for pictures, music or text needed. An electrically erasable programmable read-only memory (EEPROM) device, however, may be added to the project as an external storage device.

Here we describe how to store data in an AT24C64 EEPROM and verify the stored data using an ATmega2560 AVR microcontroller.

AT24C64 I²C EEPROM

The AT24C64 is an inter-integrated-circuit (I²C) EEPROM device. The I²C interface is termed as a two-wire interface (TWI) in the context of AVR microcontrollers. I²C is a synchronous serial protocol that allows interconnection of up to 128 different devices using only two bidirectional bus wires: clock (SCL) and data (SDA).

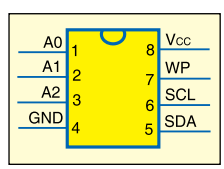

Fig. 1: Pin details of AT24C64 I²C device

Pin details of an AT24C64 chip are shown in Fig. 1. When using only one device, connect address pins A0, A1 and A2 to ground. The pins form different addresses when SDA and SCL are connected to more than one I²C devices (maximum eight devices) in parallel. For a single device, permanently ground WP pin to enable normal write operations. All the devices connected to the bus have individual addresses, and mechanisms for resolving bus contention are inherent in the TWI protocol.

For the algorithm for handling the TWI interface of the AVR microcontroller, you may refer the datasheet which supports TWI communication. This article covers implementation of the TWI communication protocol in AT24C64 with ATmega2560 chip.

ATmega2560 microcontroller

The ATmega2560 from ATMEL is a low-power, 8-bit CMOS microcontroller based on the AVR enhanced RISC architecture. The AVR core combines a rich instruction set with 32 general-purpose working registers. The microcontroller has 256 kB of in-system programmable Flash with read-while-write capabilities, 4kB EEPROM, 8kB SRAM, 86 general-purpose input/output (I/O) lines, real-time counter, six flexible timers/counters with compare modes and pulse-width modulation (PWM), four USARTs, a byte-oriented two-wire serial interface, a 16-channel, 10-bit analogue-to-digital converter (ADC) with optional differential input stage with programmable

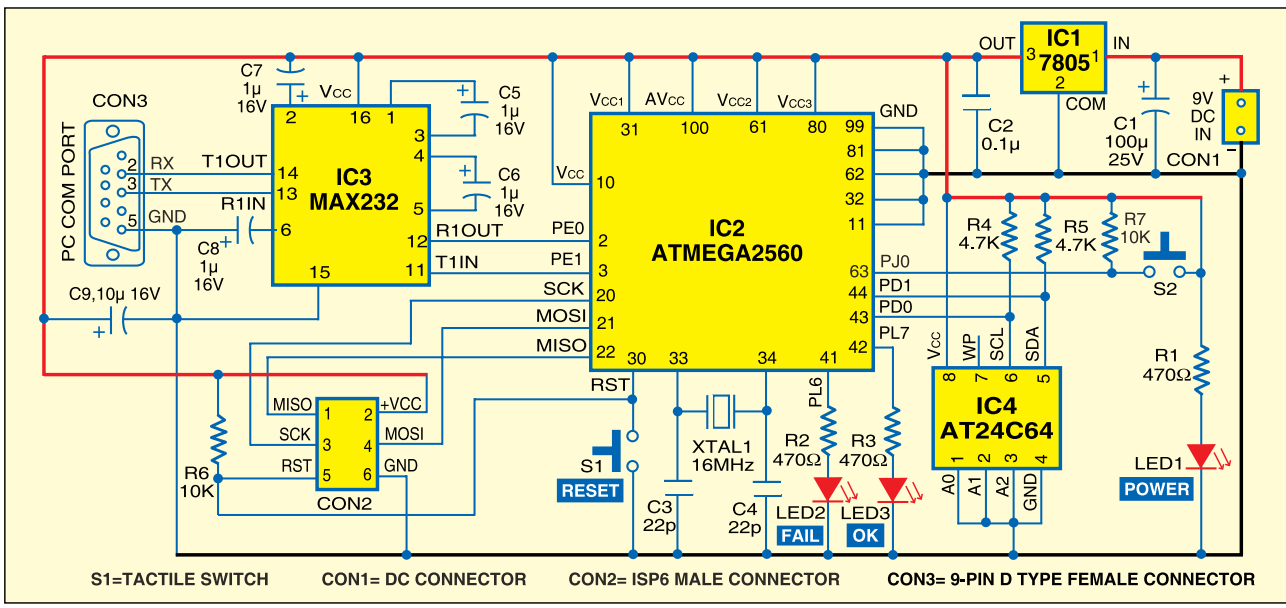

Fig. 2: Storing data on I²C EEPROM using AVR microcontroller

Fig. 3: HyperTerminal new connection

Fig. 4: COM port properties window

gain, programmable watchdog timer with internal oscillator, a serial peripheral interface (SPI) port, IEEE standard 1149.1 compliant JTAG test interface (also used for accessing the on-chip debug system and programming) and six software-selectable power-saving modes.

Circuit description

Fig. 2 shows the circuit for storing data on the AT24C64 I²C EEPROM using an ATmega2560 AVR microcontroller. The circuit is powered by a standard 9V DC source. The 9V DC is converted into 5V DC using a 7805 regulator (IC1). The glowing of LED1 shows the presence of power in the circuit. Regulated 5V powers the circuit including the ATmega2560, MAX232, AT24C64 and ISP6 connector (CON2). CON3 is a 9-pin, D-type COM port connector used to interface with the PC for text

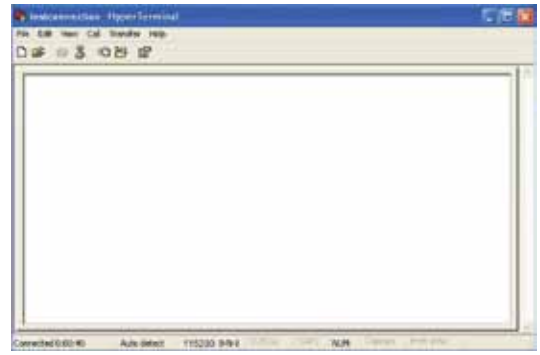

Fig. 5: HyperTerminal output window

file transfer to the AT24C64 through the microcontroller (IC2). Address pins A0, A1 and A2 of AT24C64 are connected to ground and SDA and SCL pins are connected to pins 44 and 43 (PD1 and PD0) of the microcontroller, respectively. Resistors R5 and R4 (4.7K each) are external components connected to SDA and SCL pins of AT24C64, respectively. Pin 8 of AT24C64 is connected to +5V, while its pin 7 (WP) is not used here.

ATmega2560 (IC2) is the main controller in this circuit. A 16MHz crystal oscillator is used to generate the clock pulse. Communication between the microcontroller and the PC is done through MAX232 driver (IC3). Pins 2 and 3 of IC2 are connected to pins 12 and 11 of IC3, respectively. Pins 20, 21 and 22 of IC2 are serial data communication lines connected to ISP6 connector CON2. ISP6 connector is used in programming the microcontroller using STK500 board.

Whenever the program doesn't function properly, you can reset the microcontroller by momentarily pressing reset switch S1.

Software program

The software for the microcontroller is written in 'C' language using the IAR Embedded Workbench integrated development environment. IAR Embedded Workbench is being developed by IAR Systems and ATMEL developers in parallel and hence it generates an optimised code which uses full 'C' coding capabilities of AVR devices. AVR development tools for embedded systems can be downloaded for free from IAR website www.iar.com. For details of IAR Embedded Workbench, you may refer to 'A Beginners' Guide to ATMEL AVR Development' article published in January issue of EFY.

To start the application, click 'IAR Embedded Workbench' icon in 'All Programs' menu of Windows. From the main menu, select Project→Create New Project and start an AVR Studio compatible C project. Generate the Intel hex file for burning the code into Atmega2560 microcontroller as follows: Project→Options→Linker (under Category)→Extra Output→Generate Extra Output (to click)→Override Default (to click)→Type project name with .hex extension→Output Format→Intel-standard (to select). Press 'ok' after all these settings are done.

The source code (I2C_EEPROM.c) to store data on an I²C EEPROM using an AVR microcontroller is given at the end of this article along with comments for instructions. It lets you store 4096 bytes of data in AT24C64 using principles of I²C communication.

Downloading the code into ATmega2560

Download the code into the AVR using appropriate tools. Two options to download the code are:

Using AVR Studio and STK500. To program the ATmega2560, connect the 6-wire cable between ISP connectors of the STK500 board (marked as ISP6) and the target board (marked as ISP con). Connect a serial cable from 'RS232 CTRL' connector on the STK500 board to a COM port on the PC. Now start AVR Studio 4.0 without opening the project file.

From the AVR Studio output window, proceed as follows: Main Menu→Tools→Program AVR→Select AVR Programmer. Press 'Connect...' after selecting STK500 or AVRISP in

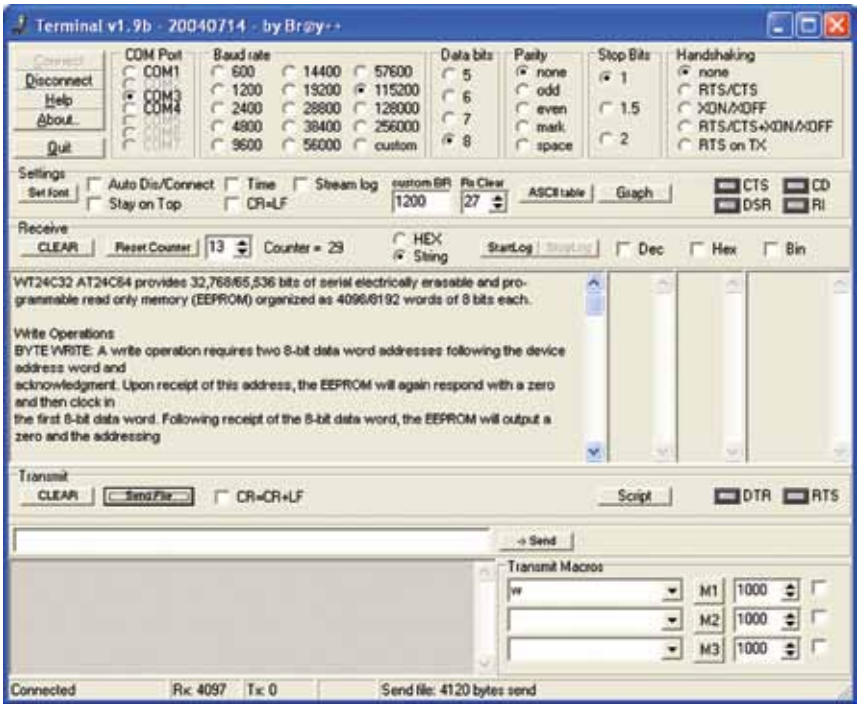

Fig. 6: Terminal v1.9b output window

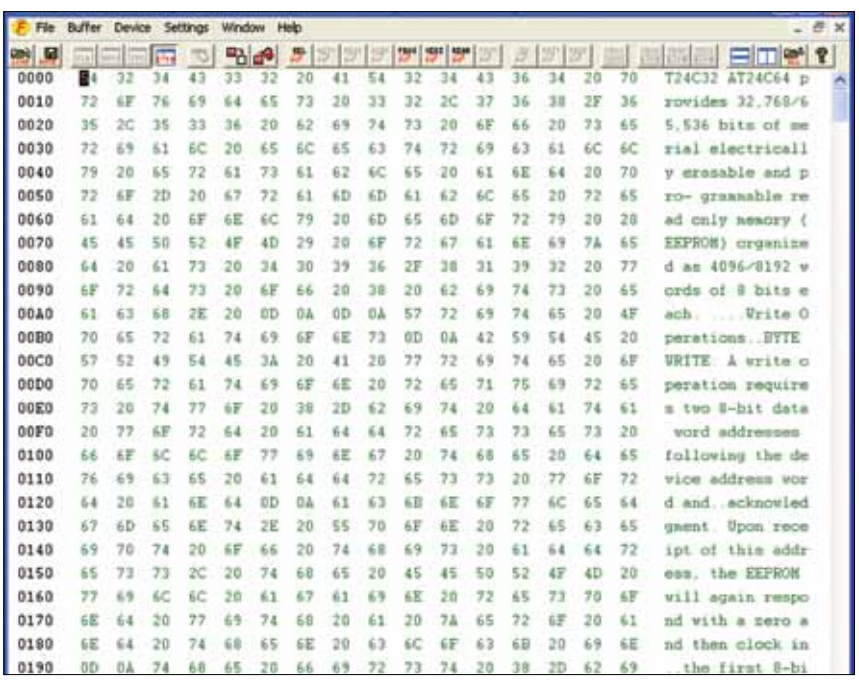

Fig. 7: TopView output text captured from EEPROM device

the platform window and COM port (say, COM1) in the port window. Next, select Atmega2560 as 'Device' and ISP as 'Programming Mode' and browse your project hex file from Project→Debug→Exe folder. Press 'Program' button on the STK500 dialogue box to burn the hex file into your microcontroller.

Boot loader implementation greatly simplifies your work by eliminating the need of a separate programmer again and again. Bootloader code is provided in this month's EFY-CD.

PARTS LIST

Semiconductors:

IC1	- 7805, 5V regulator
IC2	- ATmega2560 microcontroller
IC3	- MAX232 RS-232 driver
IC4	- AT26C64 EEPROM
LED1-LED3	- 5mm light-emitting diode

Resistors (all ¼-watt, ±5% carbon):

R1-R3	- 470-ohm
R4, R5	- 4.7-kilo-ohm
R6, R7	- 10-kilo-ohm

Capacitors:

C1	- 100µF, 25V electrolytic
C2	- 0.1µF ceramic
C3, C4	- 22pF ceramic
C5-C8	- 1µF, 16V electrolytic
C9	- 10µF, 16V electrolytic

Miscellaneous:

CON1	- DC connector
CON2	- ISP 6-pin male connector
CON3	- 9-pin D-type female connector
S1, S2	- Tactile switch

You can burn boot loader code into MCU flash memory using STK500 or any other inexpensive programmer such as Serial Port AVR-01 Programmer from http://www.onlinetps.com/shop/index.php?main_page=index&cPath=65

You can now program the MCU in your application board using software such as AVR-OSP II.

Using AVR-OSP II and AVR application board. AVR-OSP II is an Open Source programmer tool for AVR chips. Download its Version 547 from www.esnips.com/doc/ad-233fbe-7a98-4ebc-9b9c-d2b9f4d5f786/AvrOspII_547. Unzip the contents to a single folder and run AvrOspII.exe. In 'Configurations' tab, set bits per second as 19,200 and select COM port. Now select the protocol as AVR11.

Press switch S2 and reset switch S1 simultaneously. While releasing S1, keep S2 pressed. LED2 connected to pin 41 (PL6) of ATmega2560 should glow to indicate that the board is in programming mode. Release switch S2.

Press 'Auto Detect' in 'Program' window. It should detect the AVR board ATmega2560 on it. Now select the hex file that has been created by your microcontroller code. Enable

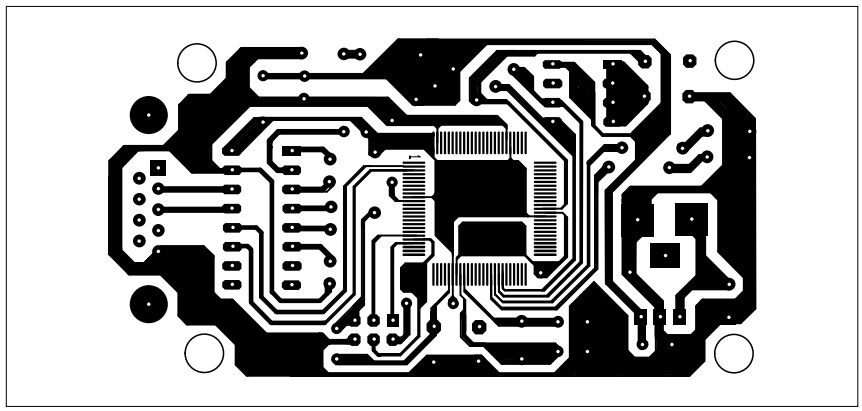

Fig. 8: An actual-size, single-side PCB for storing data on I²C EEPROM using AVR microcontroller

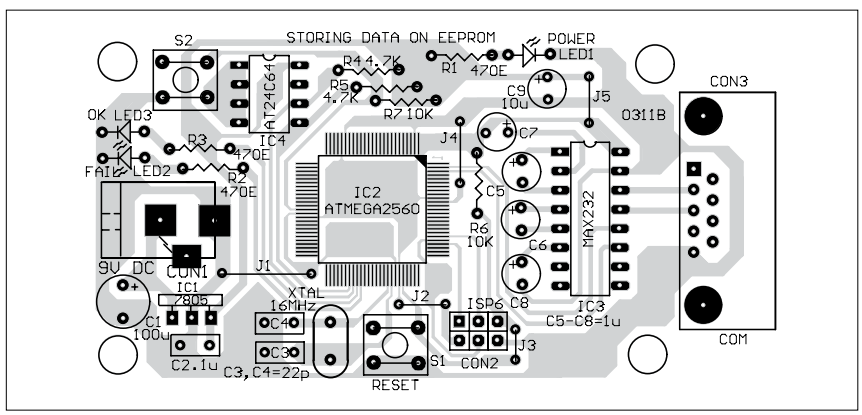

Fig. 9: Component layout for the PCB

'Erase device before programming' and 'Verify device after programming' options. Then press 'Program' to download your hex code into the ATmega2560 microcontroller.

Now you can test the code in HyperTerminal through the serial port after pressing the reset switch.

Setting HyperTerminal for testing

Start the HyperTerminal application from the desktop of your computer. Select the active COM port available on your system. Set HyperTerminal for serial communication as follows:

1. Connect the serial device (microcontroller's serial port) to the COM port of the PC.

2. Launch HyperTerminal from Start→Programs→Accessories→Communications

3. Create a new connection by specifying the name (refer Fig. 3). Set location information as per your requirement if configuring for the first time.

4. Configure the COM port properties (refer Fig. 4) as follows:

Bits per second: 115200
Data bits: 8
Parity: None
Stop bits: 1
Flow control: None

Once the connection is successfully established, you get a screen as shown in Fig. 5.

6. Select 'Send a text file' option from 'Transfer' menu to send the text file to the I²C EEPROM.

In this project, the communication between your PC and EEPROM starts with 'A' character. A text file can be sent to the EEPROM when it starts with 'A' character. The code can be modified for any other character.

Let's consider a 4096-byte long text file starting with 'A' followed by a repeating EFY sequence (A EFYEFYE-FYEFYEFYEFY). Send it by pressing 'Send Text' file option from 'Transfer' menu through HyperTerminal of Windows XP or lower version.

The code in the microcontroller waits for character 'A' from the PC's COM port at 115.2 kbps. As soon as it receives 'A,' it sends character 'W' to your PC as an acknowledgment. Then the rest of the characters are sent as a text file to the EEPROM through HyperTerminal. The text file is written onto the EEPROM, which sends an acknowledgement to the PC after writing is done. The text written on the EEPROM is shown in the HyperTerminal window on the screen. LED2 glows if data transfer fails, while LED3 glows to indicate that data transfer is successful. Sending a 4096-byte text file to the EEPROM and receiving the acknowledgement may take five to 10 seconds.

You can transfer the text file to the EEPROM through COM port using any serial communication software such as Terminal v1.9b, which is freely available on the Internet. Set the communication parameters as in HyperTerminal (refer Fig. 6). Press 'Connect' button followed by 'SendFile.' Now select any 4096-byte long text file that starts with 'A' from your computer and press 'Send' button to send it to EEPROM.

EEPROM verification

In order to ensure that the text is actually stored on the EEPROM, use a programmer such as TopView from Front Line Electronics. Insert the EEPROM in the ZIF socket and run the TopView programmer software. Select Device→Read, to read the data from the EEPROM. Now select EEPROM Buffer→Edit in 'Buffer' menu. You can see the data in the I²C EEPROM as shown in Fig. 7.

Construction

An actual-size, single-side PCB for storing data in the AT24C64 EEPROM using ATmega2560 AVR microcontroller circuit (Fig. 2) is shown in Fig. 8 and its component layout in Fig. 9.

Mount all the components as shown in the PCB layout. The ATmega2560 used in the prototype comes in a 100-pin, surface-mount TQFP package. By using surface-mount devices (SMDs) in your project, you don't have to drill holes in the PCB and the board size will be much smaller. But it may be difficult to handle and solder SMDs without an SMD soldering station. However, with a normal soldering iron, tweezers and some basic knowledge, you can solder SMD components without much problem. You need good eyes, a steady hand and a soldering iron with a small, clean tip.

Make sure that the PCB is clean. You may use copper polish material to remove all kinds of oxidation and acetone (nail polish remover) to remove all kinds of contaminations on the PCB. On the PCB, figure out the correct location for placing the SMD. Fixate the SMD on the top-right corner and the bottom-left corner. Carefully solder SMD pins one by one. SMD components are very sensitive to heat, so allow your SMD to cool down after every step.

For more tips on soldering SMD components, you may refer to http://hem.passagen.se/communication/pcbsmd.html and www.engadget.com/2006/03/07/how-tomake-a-surface-mount-soldering-iron/ websites.

Now connect a 9V DC source to the circuit and switch on the power supply. LED1 should glow to indicate the presence of power supply in the circuit. Now run the HyperTerminal or Terminal software and send the text file (.txt) to the LCD from your computer.

EFY note. The source code of this article has been included in this month's EFY-CD and is also available on efymag.com website.

The author is an assistant professor in Department of Mechanical Engineering at Birla Institute of Technology, Mesra, Ranchi

I2C_EEPROM.C

```
Source code  : I2C_EEPROM.c
/*  I2C Implementation routine for Serial EEPROM chip AT24C32 using
*   ATmega2560 and 16Mhz External Crystal
*/

#include <ioavr.h>
#include <intrinsics.h>

#define START                0x08
#define REP_START   0x10
#define MT_SLA_ACK  0x18
#define MT_DATA_ACK 0x28
#define MR_SLA_ACK  0x40
#define MR_SLA_NACK 0x48
#define MR_DATA_NACK         0x58

#define STATUS_MASK  (1<<TWS7 | 1<<TWS6| 1<<TWS5 | 1<<TWS4 | 1<<TWS3)
#define STATUS (TWSR & STATUS_MASK)

#define SETBIT(ADDRESS, BIT)     (ADDRESS |= (1 << BIT))
#define CLEARBIT(ADDRESS, BIT)   (ADDRESS &= ~(1 << BIT))
#define CHECKBIT(ADDRESS, BIT)   (ADDRESS & (1 << BIT))

#define TRUE  1
#define FALSE 0

#define LED1 PORTL6       //Connected to PORTL
#define LED2 PORTL7       //Connected to PORTL

//Address ranges from 0 - 4095 for 24C32 and 0 - 8191 for 24C64 chips
void EEOpen();
char EEWriteByte(unsigned int address, unsigned char data);
unsigned char EEReadByte(unsigned int address);

void USART0_Init (unsigned int baud);
void USART0_Transmit (unsigned char data);
unsigned char USART0_Receive (void);
void USART0_Receive_String(char *s, int length);

char arr[4096];

__C_task main(void)
{
        unsigned int address;

        //Initialize USART1 at 115.2kbps with 16MHz Crystal
        USART0_Init(8);
        SETBIT(DDRL, PL6);  //Set data direction registers for LEDs
       SETBIT(DDRL, PL7);
       while(USART0_Receive() != 'A');
       USART0_Transmit('W');
        EEOpen();
        //Fill whole eeprom 8KB (8096 bytes) with incomming data over USART
             USART0_Receive_String(arr, 4096);
  //       for(address=0;address<255;address++)
     //                       USART0_Transmit(arr[address]);

        char failed=0;
        for(address=0;address<4096;address++)
        {
                if(EEWriteByte(address, arr[address])==0)
                {
                                //Write Failed

SETBIT(PORTL, LED1);
                                failed=1;
                                break;
                }
        }
        if(!failed)
         SETBIT(PORTL, LED2);
        failed=0;
        for(address=0;address<4096;address++)
                USART0_Transmit(EEReadByte(address));
}
void EEOpen(void)
{
       //Refer: Atmel Application Note, AVR315
        //Set up TWI Module; Setting for 400kHz with 16Mhz Osc.
        TWBR = 12;
                                TWSR &= (~((1<<TWPS1)|(1<<TWPS0)));
}
char EEWriteByte(unsigned int address,unsigned char data)
{
        do
        {
                //Send START Condition
                TWCR=(1<<TWINT)|(1<<TWSTA)|(1<<TWEN);
                //Wait for TWINT flag set
                while(!(TWCR & (1<<TWINT)));
                if((STATUS != START) && (STATUS != REP_START))
                        return FALSE;
                //Write SLA+W to EEPROM @ 00h
                TWDR=0xa0; //1010 000 0
                //Initiate Transfer

TWCR=(1<<TWINT)|(1<<TWEN);
                //Wait Till Done
                while(!(TWCR & (1<<TWINT)));
        }
       while(STATUS != MT_SLA_ACK);
        //Write ADDRH
        TWDR=(address >> 8);
        //Initiate Transfer
        TWCR=(1<<TWINT)|(1<<TWEN);
        //Wait Till Done
        while(!(TWCR & (1<<TWINT)));
        if(STATUS != MT_DATA_ACK)
                return FALSE;

        //Write ADDRL
        TWDR= address;
        //Initiate Transfer
        TWCR=(1<<TWINT)|(1<<TWEN);
        //Wait Till Done
        while(!(TWCR & (1<<TWINT)));
        if(STATUS != MT_DATA_ACK)
                return FALSE;

        //Write DATA
        TWDR=(data);
        //Initiate Transfer
        TWCR=(1<<TWINT)|(1<<TWEN);
        //Wait Till Done
        while(!(TWCR & (1<<TWINT)));
        if(STATUS != MT_DATA_ACK)
                return FALSE;
```

```
        //Put Stop Condition
        TWCR=(1<<TWINT)|(1<<TWEN)|(1<<
TWSTO);
        //Wait for STOP to finish
        while(TWCR & (1<<TWSTO));
        //Wait untill Writing is com-
plete
        __delay_cycles(34000);
//2.125mS failed for 2mS
        //Return TRUE on Success
        return TRUE;
}
unsigned char EEReadByte(unsigned int
address)
{
        unsigned char data;
        //Initiate a Dummy Write Se-
quence to start Random Read
        do
        {
                //Put Start Condition
on TWI Bus
                TWCR=(1<<TWINT)|(1<<T
WSTA)|(1<<TWEN);
                //Wait Till Done
                while(!(TWCR &
(1<<TWINT)));
                if((STATUS != START)
&& (STATUS != REP_START))
                        return
FALSE;
                //Write SLA+W to EE-
PROM @ 00h
                TWDR=0xa0;
//1010 000 0;
                //Initiate Transfer

TWCR=(1<<TWINT)|(1<<TWEN);
                //Wait Till Done
                while(!(TWCR &
(1<<TWINT)));
        }
       while(STATUS != MT_SLA_ACK);
        //Write ADDRH
        TWDR=(address >> 8);
        //Initiate Transfer
        TWCR=(1<<TWINT)|(1<<TWEN);
        //Wait Till Done
        while(!(TWCR & (1<<TWINT)));
        if(STATUS != MT_DATA_ACK)
                return FALSE;
        //Write ADDRL
        TWDR=address;
        //Initiate Transfer
        TWCR=(1<<TWINT)|(1<<TWEN);
        //Wait Till Done
        while(!(TWCR & (1<<TWINT)));
        if(STATUS != MT_DATA_ACK)
                return FALSE;

        //********************** End
dummy write sequence ******************
        //Put Start Condition on TWI
Bus
        TWCR=(1<<TWINT)|(1<<TWSTA)|(1<
<TWEN);
        //Wait Till Done
        while(!(TWCR & (1<<TWINT)));
        if(STATUS != REP_START)
         return FALSE;
        //Write SLA+R to EEPROM @ 00h
        TWDR=0xa1;              //1010
000 1
        //Initiate Transfer
        TWCR=(1<<TWINT)|(1<<TWEN);
        //Wait Till Done
        while(!(TWCR & (1<<TWINT)));
        if(STATUS != MR_SLA_ACK)
         return FALSE;
        //Enable Reception of data by
clearing TWINT
        TWCR=(1<<TWINT)|(1<<TWEN);
        //Wait till done
        while(!(TWCR & (1<<TWINT)));
        if(STATUS != MR_DATA_NACK)
         return FALSE;
        //Read the data
        data=TWDR;
        //Put Stop Condition
        TWCR=(1<<TWINT)|(1<<TWEN)|(1<<
TWSTO);
        //Wait for STOP to finish
        while(TWCR & (1<<TWSTO));
        //Return TRUE
        return data;
}
//Initialize USART1
void USART0_Init(unsigned int baud)
{
  /* Set baud rate */
  UBRR0H = (unsigned char)(baud >> 8);
  UBRR0L = (unsigned char)(baud);
  /* Enable receiver and transmitter */
  UCSR0B = (1<<RXEN0)|(1<<TXEN0);
   /* Set frame format: Asynchronous 8
data, 1 stop bit */
  UCSR0C = (1<<UCSZ01)|(1<<UCSZ00);
}

//Read and Write functions for USART0
void USART0_Transmit (unsigned char
data)
{
  /* Wait for empty transmit buffer */
  while (!CHECKBIT(UCSR0A, UDRE0));
   /* Put data into buffer, transmits
the data */
  UDR0 = data;
}

unsigned char USART0_Receive (void)
{
  /* Wait for data to be received */
  while (!CHECKBIT(UCSR0A, RXC0));
  /* Get and return received data from
buffer */
  return UDR0;
}
void USART0_Receive_String(char *s, int
length)
{
  unsigned char character;
  character = USART0_Receive();
  while (length--)
  {
    *s++ = character;
    character = USART0_Receive();
  }
}
```