

```
/*-----|
| Name : Eeprom.c |
| Purpose: |
| I2C (inter integrated circuit bus) is a 2-wire serial interface standard defined |
| by Philips Semiconductor. This program will demonstrate the basic function, read |
| byte, write byte, sequential read and write page of the i2c eeprom. UMPS simulator |
| will be implemented for this demonstration program. select the device 'I2C Eeprom' |
| in resources, config the setting appropriate and run the program. |
|-----*/
#include "config.h"

/*
 demonstrate the basic function of the i2c eeprom.
*/
void main(void)
{
    unsigned int i;
    unsigned char rcvdata;
    unsigned char databuf[PAGE_SIZE];

    read_Page(0x10,databuf);           // read a page from address 0x10 to 0x16
    for(i=0;i<3000;i++);              // and store it in array databuf[].
    write_Page(0x20,databuf);         // write a page form address 0x20 to 0x26
    for(i=0;i<3000;i++);              // the data is store in array databuf[].
    rcvdata=read_Byte(0x20);          // read a byte in address 0x20
    for(i=0;i<3000;i++);
    write_Byte(0x30,rcvdata);         // write a byte in address 0x30

    while(1);
}
```

```

/*-----
| Name: advanced.c
| Purpose:
| provide the advanced function of the i2c eeeprom. write byte, read byte, write page
| and sequential read.(PS: function read_Byte is call by value, read_Page is call by
| address)
|-----*/
#include "config.h"

/*
 * write a byte to i2c devices.
 */
void write_Byte(unsigned char eeepromAddr, wrData)
{
    sendStart(); // start
    sendSlaveAddress(SLAVE_ADDRESS); // 7 bit slave address
    sendBitIndicateWrite(); // a bit for write mode
    waitAck();
    sendByte(eeepromAddr); // eeeprom address
    waitAck();
    sendByte(wrData); // a data
    waitAck();
    sendStop(); // byte written completed
}

/*
 * write a page to i2c devices, eeepromAddr indicate the starting address for write
 * and eeepromAddr + PAGE_SIZE is the ending address. the data for write is store in
 * array wrbuf[]. the array is pass by address that the caller determined.
 */
void write_Page(unsigned char eeepromAddr, unsigned char wrbuf[PAGE_SIZE])
{
    unsigned int i;
    sendStart(); // start notation
    sendSlaveAddress(SLAVE_ADDRESS); // 7 bit slave address
    sendBitIndicateWrite(); // a bit for write mode
    waitAck();
    sendByte(eeepromAddr); // eeeprom address
    waitAck();
    for(i=0;i<PAGE_SIZE;i++) // send the data for write
    {
        sendByte(wrbuf[i]); // some data
        waitAck();
    }
    sendStop(); // page written completed
}

/*
 * read a byte in a particular address of eeeprom memory, and return a readed value
 * to the caller.
 */
unsigned char read_Byte(unsigned char eeepromAddr)
{
    unsigned char rcvdata;
    unsigned char mask; // variable for getting the reading data

    /* set the ptr of eeeprom address */
    sendStart(); // set the eeeprom location address
    sendSlaveAddress(SLAVE_ADDRESS); // 7 bit address for slave location
    sendBitIndicateWrite(); // bit indicated for write mode to
    waitAck(); // set the memory address of eeeprom
    sendByte(eeepromAddr);
    waitAck();

    /* current address read */
    sendStart();
    sendSlaveAddress(SLAVE_ADDRESS);
    sendBitIndicateRead();
    waitAck();

    mask = 0x80;
    do{ // store 8 bit data
        setSCL; // negative edge clock data out
        if (SDA==1)
            rcvdata |= mask;
    } while(mask >> 1);
}

```

```
        clrSCL;
        mask = mask/2;
    }while (mask>0);
    masterNoAck();

    sendStop();                // read completed
    return rcvdata;
}

/*
random sequential read, the function will read the data from the address 'eepromAddr'
to the eepromAddr + PAGE_SIZE. the readed data will stored in variable rcvbuf[], which
address is passed by the caller. pass by reference, that means the caller variable array
will be changed by the called function.
*/
void read_Page(unsigned char eepromAddr,unsigned char rcvbuf[PAGE_SIZE])
{
    unsigned int i;
    unsigned char mask;                // variable for getting the reading data

    /* set the ptr of eeprom address */
    sendStart();
    sendSlaveAddress(SLAVE_ADDRESS);    // set the eeprom location address
    sendBitIndicateWrite();            // 7 bit address for slave location
    waitAck();                        // bit indicated for write mode to
    sendByte(eepromAddr);              // set the memory address of eeprom
    waitAck();

    /* current address read */
    sendStart();
    sendSlaveAddress(SLAVE_ADDRESS);
    sendBitIndicateRead();
    waitAck();

    // get the data from eeprom and store it to array rcvbuf[], which address is
    // determined by the caller. As this function is call by address
    for(i=0;i<PAGE_SIZE;i++)            // repeat PAGE_SIZE time
    {
        mask = 0x80;
        do{
            setSCL;                    // store 8 bit data
            if (SDA==1)                // negative edge clock data out
                rcvbuf[i] |= mask;
            clrSCL;
            mask = mask/2;
        }while (mask>0);
        if(i==PAGE_SIZE-1)            // if last read, send No ack
            masterNoAck();
        else                            // if not last read, send ack
            masterAck();
    }

    sendStop();                // read completed

    return;
}
```

```
/*-----|
| Name: basic.c |
| Purpose: |
| provide the basic level control for the combined advanced function. Eg, send a start |
| notation, stop notation, send a slave address, byte and acknowledgement. |
|-----*/
#include "config.h"

// a start notation for the i2c slave
void sendStart(void){
    setSDA;
    setSCL;
    clrSDA;
    clrSCL;
}

// a stop notation for the i2c slave
void sendStop(void){
    clrSDA;
    setSCL;
    setSDA;
    clrSCL;
}

// sends one byte of data to a i2c slave
void sendByte(unsigned char b){
    unsigned char mask;
    mask = 0x80;
    do{
        if ( b & mask ){
            setSDA;
        }
        else{
            clrSDA;
        }
        setSCL;
        clrSCL;
        mask = mask/2;
    }while (mask>0);
}

// sends lower 7 bit of data to a i2c slave
void sendSlaveAddress(unsigned char b){
    unsigned char mask;
    mask = 0x80;
    b*=2;
    do{
        if ( b & mask ){
            setSDA;
        }
        else{
            clrSDA;
        }
        setSCL;
        clrSCL;
        mask = mask/2;
    }while (mask>1);
}

// wait until acknowledgment is received from slave
void waitAck(void){
    setSDA;
    setSCL;
    while(SDA);
    clrSCL;
}

// master acknowledge by sending a clr bit to slave
void masterAck(void){
    clrSDA;
    setSCL;
    clrSCL;
}

// master disacknowledge by sending a set bit to slave
```

```
void masterNoAck(void){
    setSDA;
    setSCL;
    clrSCL;
}

// a bit after the slave address, clr indicate i2c write
void sendBitIndicateWrite(void){
    clrSDA;
    setSCL;
    clrSCL;
}

// a bit after the slave address, set indicate i2c read
void sendBitIndicateRead(void){
    setSDA;
    setSCL;
    clrSCL;
}
```

```

/*-----|
| Name: config.h |
| Purpose: |
| config the hardware setting, define the hardware control pin with a notation name |
| select the require lib for the MPU, and define the protocol for the program access |
|-----*/
#include <reg51.h>
#include <stdio.h>

#define setSCL      SCL=1;while (SCL!=1);
#define clrSCL      SCL=0;
#define setSDA      SDA=1;
#define clrSDA      SDA=0;

#define PAGE_SIZE    7                // max no. of page size for read/ write
#define SLAVE_ADDRESS 0x50            // the address of the i2c slave

sbit    SCL = P1^2;                    // i2c clock pin
sbit    SDA = P1^3;                    // i2c data pin

// basic.c
void sendStart(void);
void sendStop(void);
void sendByte(unsigned char);
void sendSlaveAddress(unsigned char b);
void waitAck(void);
void masterAck(void);
void masterNoAck(void);
void sendBitIndicateWrite(void);
void sendBitIndicateRead(void);

// advanced.c
void write_Byte(unsigned char eeepromAddr, eeepromData);
void write_Page(unsigned char eeepromAddr, unsigned char wrbuf[PAGE_SIZE]);
unsigned char read_Byte(unsigned char eeepromAddr);
void read_Page(unsigned char eeepromAddr,unsigned char rcvbuf[PAGE_SIZE]);

```