

```
/*-----|
| Name: 93C46.c
| Purpose:
| A demonstration program to illustrate how to program the 3 wire bus serial EEPROM (93C46 type).
| Device is first enable for write and erase (EWEN).
| This program use 8 bit mode, ORG = 0
|-----*/
#include "config.h"

/*
 * demonstrate the usage of the function of this program
 */
void main(void)
{
    unsigned char rcvdata;
    erase_write_enable();           // enable erase/ write
    write_byte(0x31,0xaa);          // write data to memroy
    rcvdata=read_byte(0x31);        // read data from memory
    erase_byte(0x31);              // erase byte
    write_all(0xcc);                // write all with data
    erase_all();                   // erase all
    write_all(0x99);                // write all with data
    write_byte(0x31,0xaa);          // write data to memory
    erase_write_disable();          // disable erase/ write

    while(1);
}
```

```

/*-----
| Name: advanced.c
| Purpose:
| The AT93C46/56/66 is enabled through the Chip Select pin (CS), and accessed via a 3-wire serial
| interface consisting of Data Input (DI), Data Output (DO), and Shift Clock (SK). Upon receiving a
| READ instruction at DI, the address is decoded and the data is clocked out serially on the data
| output pin DO. The WRITE cycle is completely self-timed and no separate. ERASE cycle is required
| before WRITE. The WRITE cycle is only enabled when the part is in the ERASE/WRITE ENABLE state.
| When CS is brought | Sigh| "following the initiation of a WRITE cycle, the DO pin outputs the
| READY/BUSY status of the part.
|-----*/
#include "config.h"

/*
 * write enable must precede all programming modes
 */
void erase_write_enable(void){
    setCS;                // chip select
    sendStartBit();        // send start bit
    sendOpCode(EWEN);      // op-code
    sendAddr(EWENADDR);    // function code
    clrCS;                // chip disselect
}

/*
 * disable all programming instructions
 */
void erase_write_disable(void){
    setCS;                // chip select
    sendStartBit();        // send start bit
    sendOpCode(EWDS);      // op-code
    sendAddr(EWDSADDR);    // function code
    clrCS;                // chip disselect
}

/*
 * erase memroy location eraddr
 */
void erase_byte(unsigned char eraddr){
    setCS;                // chip select
    sendStartBit();        // send start bit
    sendOpCode(ERASE);      // op-code
    sendAddr(eraddr);      // function code
    clrCS;                // chip disselect
    waitDone();            // wait process complete
}

/*
 * erases all memory locations.
 */
void erase_all(void){
    setCS;                // chip select
    sendStartBit();        // send start bit
    sendOpCode(ERAL);      // op-code
    sendAddr(ERALADDR);    // function code
    clrCS;                // chip disselect
    waitDone();            // wait process complete
}

/*
 * reads data stored in memory, at specified address
 */
unsigned char read_byte(unsigned char rcvaddr){
    unsigned char mask,rcvdata;
    setCS;                // chip select
    sendStartBit();        // send start bit
    sendOpCode(READ);      // op-code
    sendAddr(rcvaddr);     // address

    // I find this part can not use function to impletement instead, and must be written in this
    // form directly. The reason may involves the process while the function return the value to
    // read_byte or the complier problems.
    mask = 0x80;
    do{
        setSK;                // store 8 bit data
        if (DO==1)            // negative edge clock data out
            rcvdata |= mask;
        clrSK;
        mask = mask/2;
    }while(mask);
}

```

```
    }while (mask>0);
    clrCS;                                     // chip disselect
    return rcvdata;
}

/*
 * writes memory location wraddr with byte wrdata
 */
void write_byte(unsigned char wraddr, wrdata){
    setCS;                                     // chip select
    sendStartBit();                           // send start bit
    sendOpCode(WRITE);                         // op-code
    sendAddr(wraddr);                         // address
    sendByte(wrdata);                         // data
    clrCS;                                     // chip disselect
    waitDone();                               // wait process complete
}

/*
 * writes all memory loctions with data wrdata
 */
void write_all(unsigned char wrdata){
    setCS;                                     // chip select
    sendStartBit();                           // send start bit
    sendOpCode(WRAL);                         // op-code
    sendAddr(WRALADDR);                      // function code
    sendByte(wrdata);                         // data
    clrCS;                                     // chip disselect
    waitDone();                               // wait process complete
}
```

```
/*-----
| Name: basic.c
| Purpose:
| The fundamental function of a specific bit pattern for the usage of the advanced.c
| PS: function waitDone() is a critical function. the function may by pass because the reach of the
| processing time. At this conditions, user can increase the valuse of MAXPROCESSTIME defined in
| config.h
|-----*/
#include "config.h"

/*
a start bit of all instruction
*/
void sendStartBit(void){
    clrSK;                                // a high bit before op-code
    setDI;
    setSK;
    clrSK;
}

/*
op-code for the instruction
*/
void sendOpCode(bit a, bit b){
    if(a){                                // first op-code
        setDI;
    }
    else{
        clrDI;
    }
    setSK;                                // toggle a clock
    clrSK;

    if(b){                                // second op-code
        setDI;
    }
    else{
        clrDI;
    }
    setSK;                                // toggle a clock
    clrSK;
}

/*
sends lower 7 bit of address to a serial eeprom
*/
void sendAddr(unsigned char b){
    unsigned char mask;
    mask = 0x80;
    b*=2;                                // shift b left
    do{                                    // send 7 bits
        if ( b & mask ){
            setDI;                                // bit is high
        }
        else{
            clrDI;                                // bit is low
        }
        setSK;                                // toggle a clock
        clrSK;
        mask = mask/2;                            // shift mask right
    }while (mask>1);
}

/*
sends one byte of data to a serial eeprom
*/
void sendByte(unsigned char b){
    unsigned char mask;
    mask = 0x80;
    do{                                    // send 8 bits
        if ( b & mask ){
            setDI;                                // bit is high
        }
        else{
            clrDI;                                // bit is low
        }
        setSK;                                // toggle a clock
        clrSK;
        mask = mask/2;                            // shift mask right
    }
}
```

```
    }while (mask>0);
}

/*
  check the status of serial eeprom. If the eeprom is busy, DO is low. If the eeprom
  is ready, DO is high. This function will wait until the operation is completed, or
  the overflow of the MAXPROCESSTIME which limit the process time, and avoid the dead
  loop of the program. Note that if the processing cycle is too short, the busy ack may
  not be caught. The limitation of the processing time can help to let the next instruction
  to continue carry on without dead loop.
*/
void waitDone(void) {
    unsigned int i;
    setDO;                                // release DO for receive the status
    setSK;                                // a clock pulse
    clrSK;
    setCS;                                // set the chip select
    i=0;
    while (DO&i<MAXPROCESSTIME) {         // wait until the appear of busy state
        setSK;                            // or the overflow of the i
        clrSK;
        i++;
    }
    i=0;
    while (!DO&i<MAXPROCESSTIME) {        // wait until the appear of ready state
        setSK;                            // or the overflow of the i
        clrSK;
        i++;
    }
    clrCS;                                // eeprom operation completed
}
```

```
/*-----|
| Name: config.c
| Purpose:
| The hardware pin setting to connect the program and target. The definition of the preprocessor for
| easier programming and the setting of include marcus.
| PS: If the real hardware design find the advance function can not complete, users can increase the
| value of MAXPROCESSTIME to increase the processing time.
|-----*/
#include <reg51.h>
#include <stdio.h>

/* set or clr of important PIN */
#define setCS      CS=1;while (CS!=1);          // chip select
#define clrCS      CS=0;
#define setSK      SK=1;while (SK!=1);          // clock
#define clrSK      SK=0;
#define setDI      DI=1;while (DI!=1);          // data in
#define clrDI      DI=0;
#define setDO      DO=1;while (DO!=1);          // data out
#define clrDO      DO=0;

/* Op-Code */
#define READ        1,0
#define EWEN        0,0
#define ERASE       1,1
#define WRITE       0,1
#define ERAL        0,0
#define WRAL        0,0
#define EWDS        0,0

/* X8 Addr for special function */
#define EWENADDR    0x60
#define ERALADDR    0x40
#define WRALADDR    0x20
#define EWDSADDR    0x00

/* the maxiumum allowance of the processing time */
#define MAXPROCESSTIME 500

/* Pin definition */
sbit CS = P1^2;          // chip select (ENB)
sbit SK = P1^3;          // clock (HCK)
sbit DI = P1^4;          // data in (DI)
sbit DO = P1^5;          // data out (Dout)

// basic.c
void sendStartBit(void);
void sendOpCode(bit a, bit b);
void sendAddr(unsigned char b);
void sendByte(unsigned char b);
void waitDone(void);

// advance.c
void erase_write_enable(void);
void erase_write_disable(void);
void erase_byte(unsigned char erAddr);
void erase_all(void);
unsigned char read_byte(unsigned char rcvaddr);
void write_byte(unsigned char wraddr, wrdata);
void write_all(unsigned char wrdata);
```