C:\WTP_ROOT\89C51_52_Programmer_7_28_03\89C51_52_Programmer\main.c 07/28/03 00:59:32

```
/*-------------------------------------------------------------------------------+
| MAIN.C : AT89C51/52 PROGRAMMER                                                 |
|(use AT89C52 as a master chip for programmer)                                   |
| WTP Product                                                                    |
+-------------------------------------------------------------------------------*/
#include "config.h"
static char code menu[] =                          // menu of programmer
    "\nAT89C51/52 PROGRAMMER (made by WTP)"
    "\nq   bulk programming(i,e,p,l1,l2,l3)"
    "\ni   chip identification"
    "\ne   chip erase"
    "\nc   read memory"
    "\np   program memory"
    "\nv   verify memory"
    "\nl   write lock bit"
    "\na   send address"
    "\nd   send data"
    "\nw   send write code command"
    "\nr   send read code command"
    "\ng   get return data value"
    "\nt   write all [Test mode]"
    ;

void main(void)
{
    static unsigned char addrL,addrH,hexdata;      // generate variable for address, hex data
    static unsigned char s[1];                     // for two char input
    static unsigned char cmd;                      // for one char input

        uart_init();                               /* setting for serial interface   */
        printf(menu);                              /*  Menu.                         */
        while(1){
        printf( "\n[Enter selection:] " );
        scanf("%c", &cmd);

            switch (cmd){
                case '?':                          // print menu
                    printf(menu);
                    break;
                case 'q':                          // quick mode
                    chip_identification();         // identifify
                    if(AT89C51_identification()|AT89C52_identification())
                    {                              // if programmer supported
                        erase();                   // implement main function
                        program();                 // program
                        verify();                  // verify
                        write_lock('1');           // 1,2,3 lock bit
                        write_lock('2');
                        write_lock('3');
                    }
                    else                           // if not terminate
                    {
                        printf("\nfunction terminate!");
                    }
                    break;
                case 'i':                          // identifiy the chip
                    chip_identification();
                    break;
                case 'e':                          // chip erase
                    erase();
                    break;
                case 'c':                          // read memory
                    read();
                    break;
                case 'p':                          // program memory
                    program();
                    break;
                case 'v':                          // verify memory
                    verify();
                    break;
                case 'l':                          // write lock bit
                    printf("\npress 1 for LB1, 2 for LB2, 3 for LB3 : ");
                    scanf("%c",&cmd);
                    write_lock(cmd);
                    printf("...programmed");
                    break;
```

```
                case 'a':                           // send a address to slave
                    printf("\n....enter a Higher address :");
                    scanf("%c",&s[0]);
                    scanf("%c",&s[1]);
                    addrH=toint(s[0])*16 + toint(s[1]);
                    printf("\n....enter a Lower address  :");
                    scanf("%c",&s[0]);
                    scanf("%c",&s[1]);
                    addrL=toint(s[0])*16 + toint(s[1]);
                    send_addr(addrH,addrL);
                    break;
                case 'd':                           // send a data to slave
                    printf("\n....enter a data value :");
                    scanf("%c",&s[0]);
                    scanf("%c",&s[1]);
                    hexdata=toint(s[0])*16 + toint(s[1]);
                    send_data(hexdata);
                    break;
                case 'g':                           // get the return data from slave
                    printf("\nThe return data :");
                    print_return_data();
                    printf("H");
                    break;
                case 'w':                           // send a write code command to slave
                    write_code_data();
                    break;
                case 'r':                           // send a read code command to slave
                    read_code_data();
                    break;
                case 't':                           // a test mode for writing all the address
                    write_all();
                    break;
            }

        }//End of while
}
```

C:\WTP_ROOT\89C51_52_Programmer_7_28_03\89C51_52_Programmer\hex.c 05/20/80 17:50:08

```
/*-------------------------------------------------------------------------------+
| Name   : hex.c                                                                 |
| purpose:                                                                       |
| This source code for the print out of hex data, hex to ascii conversation is   |
| implemented before printing the hex data on screen that from a slave chip or   |
| direct input parameter.                                                        |
+-------------------------------------------------------------------------------*/
#include "config.h"

/*
    send out the value of P0
*/
unsigned char get_return_data(void)
{
    return P0;
}

/*
    print the value of P0 on screen
*/
void print_return_data(void)
{
    static unsigned char hex_h;
    static unsigned char hex_l;
    hex_l = toasciiHex(get_return_data()&0x0f);         // convert low byte
    hex_h = toasciiHex((get_return_data()&0xf0)/0x0f);  // convert high byte
    printf("%c%c",hex_h,hex_l);                         // print out
}

/*
    print the value of hex data on screen
*/
void print_hexdata(unsigned char hexdata)
{
    static unsigned char hex_h;
    static unsigned char hex_l;
    hex_l = toasciiHex(hexdata&0x0f);                   // convert low byte
    hex_h = toasciiHex((hexdata&0xf0)/0x0f);            // convert high byte
    printf("%c%c",hex_h,hex_l);                         // print out
}

/*
    This function will convert 4 bit data, one hex, to ascii code for print out.
    it is the binary to ascii hex data convertion. Eg.0x01 -> '1',0x0a ->'A' etc.
*/
unsigned char toasciiHex(unsigned char bits_data)
{
    if(bits_data >=0x0f)                                // all invalid data return 'F'
    {
        return 0x46;
    }
    if (bits_data >=0x00 & bits_data <=0x09)            // return '0'-'9'
    {
        bits_data +=0x30;
        return bits_data;
    }
    if (bits_data >=0x0a & bits_data <=0x0f);           // return 'A'-'F'
    {
        bits_data +=0x37;
        return bits_data;
    }
}
```

C:\WTP_ROOT\89C51_52_Programmer_7_28_03\89C51_52_Programmer\command.c 07/28/03 01:05:02

```
/*-------------------------------------------------------------------------------+
| Name   : command.c                                                             |
| purpose:                                                                       |
| The source code for the control of the hardware device, send a address, data,  |
| write command, read command, erase, lock bit and signature command etc.        |
+-------------------------------------------------------------------------------*/
#include "config.h"


/*
    write code data command
*/
void write_code_data(void){
static unsigned int i;

    VPP=VPP12V; // VPP 12V
    ctrlcmd(0,1,1,1);
    ALE=1;
    for (i=0;i<5;i++);                          // i=2 is ok for 9600, if find error
    ALE=0;                                      // may need to increase the value of i
    for (i=0;i<5;i++);
    ALE=1;
    ctrlcmd(0,0,0,0);                           // avoid destrution
}

/*
    read code data command
*/
void read_code_data(void){
static unsigned int i;

    ALE=0;
    for (i=0;i<20;i++);
    ALE=1;
    VPP=VPP5V;  // VPP 5V

    P0 = 0xFF;                                  // put FF before read back
    for(i=0;i<20;i++);
    ctrlcmd(0,0,1,1);
    for(i=0;i<20;i++);

}

/*
    read signature command
*/
void read_signature(void)
{
    static unsigned int i;
    ALE=0;
    for (i=0;i<20;i++);
    ALE=1;
    VPP=VPP5V;  // VPP 5V
    ctrlcmd(0,0,0,1);

    P0 = 0xFF;                                  // put FF before read back
    for(i=0;i<20;i++);
    MT4=0;
    for(i=0;i<20;i++);
}

/*
    chip erase command
*/
void erase(void)
{
    static unsigned int i;
    printf("\nErasing.........");
    VPP=VPP12V; // VPP 12V
    ctrlcmd(1,0,0,0);
    ALE=1;
    ALE=0;
    for(i=0;i<5000;i++);
    ALE=1;
    printf("ok!");
}
```

```
/*
    chip lock bit
*/
void write_lock(unsigned char k)
{
    static unsigned int i;
    VPP=VPP12V; // VPP 12V
    switch(k)
    {
        case '1':
        {
            printf("\nLock Bit-1");
            ctrlcmd(1,1,1,1);
            break;
        }
        case '2':
        {
            printf("\nLock Bit-2");
            ctrlcmd(1,1,0,0);
            break;
        }
        case '3':
        {
            printf("\nLock Bit-3");
            ctrlcmd(1,0,1,0);
            break;
        }
    }
    ALE=1;
    for (i=0;i<20;i++);                             // i put i=20 is larger
    ALE=0;                                          // but it works
    for (i=0;i<20;i++);
    ALE=1;
    ctrlcmd(0,0,0,0);                               // avoid destrution
    printf("....ok!");

}


/*
    send 2 bytes address to P0-P2.3
*/
void send_addr(unsigned char addrH,addrL){
    ctrlcmd(1,1,1,1);                               //  set before send data
    P2=addrH;
    P0=addrL;
    CLK=0;                                          // trigger the D-flip-flop
    CLK=1;
}

/*
    send hex data to P0
*/
void send_data(unsigned char hexdata){
    ctrlcmd(1,1,1,1);                               // set before send data
    P0=hexdata;
}

/*
    4 control pin for programming a chip
*/
void ctrlcmd(unsigned int i,j,k,l){
    MT1=i;
    MT2=j;
    MT3=k;
    MT4=l;
}
```

C:\WTP_ROOT\89C51_52_Programmer_7_28_03\89C51_52_Programmer\uart.c 07/28/03 01:05:36

```
/*-------------------------------------------------------------------------------+
| Name   : uart.c                                                                |
| purpose:                                                                       |
| Functions dedicated to hex file reception from host via the uart.              |
| These functions provide uart initalization, interrupt handling, hex data       |
| analysing and implement instant chip programming                               |
+-------------------------------------------------------------------------------*/

#include "config.h"
static unsigned int rx_continue;            // flag of serial interrupt
static unsigned int verify_implement,verify_pass;
                                            // flag of verify implement,
                                            // and verify pass

/*
    set the serial interface, 8bit data, 9600 bauds, no parity operating mode
*/
void uart_init(void)
{
    SCON = 0x50;
    TMOD = TMOD | 0x20 ;                    // Timer1 in mode 2 & not gated
    TH1 = 253;                              // 9600 bauds @11.0592MHz
    PCON = PCON & 0x00;                     // no double baud rate
    TCON |= 0x40;
    TI=1;

}

/*
    Enable the serial interrupt before sending a hex file to MPU, when the hex file
    transmit  complete disable serial interrupt.
*/
void en_serInt(void)
{
    printf("\nPress Alt s for transmit a hex file");
    rx_continue=1;                          // rx receive on process
    EA=1;                                   // enable global interrupts
    ES=1;                                   // enable serial interrupts
    while(rx_continue);                     // only rx_continue is false
    ES=0;                                   // disable serial interrupts
}

/*
    program memory according the hex data from serial interrupt
*/
void program(void)
{
    printf("\nprogramming....");
    en_serInt();                            // implement serial interrupt
}

/*
    verify memory according the hex data from serial interrupt
*/
void verify(void)
{
    verify_implement=1;                     // set verify implement
    verify_pass=1;                          // set verify pass
    printf("\nverifing.....");
    en_serInt();                            // implement serial interupt
    if(verify_pass)                         // if verify pass is set
        printf("\nverify pass!");
    else                                    // if verify pass is clear
        printf("\nverify NOT pass!");
    verify_implement=0;                     // restore flag, becasue it
    verify_pass=0;                          // attribute is static
}

/*
    Interrupt serivice rountine for serial interrupt.
*/
void uart (void) interrupt 4{
    if (RI == 1)                            /* Processes Rx event only, not Tx */
    {
            process(SBUF);
```

```
        RI = 0;                             // clear interrupt request flag
    }
}


/*
    Decodes the hex data received, analyse which one is the length, address, data
    value and check sum. Extracts data for chip programming or verifying, check sum
    is implemented. When the sign of ending detected, variable rx_continue will clear
    for ending the serial interrupt.
*/
void process(unsigned char rx_data)
{
    static unsigned int ptr;
    static unsigned char length,addrH,addrL,type,hexdata,checksum,nb_byte,calsum;
    static unsigned int error_occur;        // if error, set error_occur

    if(rx_data==':')                        /* start of a new record */
    {

        ptr=1;
        nb_byte=0;                          // restart no. of byte
        calsum=0;                           // restart calsum

        return;
    }

    switch (ptr)
   {
        case 1:                             /* length */
        {
            length=toint(rx_data)*16;       // get the lst digit of the length byte
            ptr=2;
            break;
        }
        case 2:
        {
            length+=toint(rx_data);         // get the 2st digit of the length byte
            calsum+=length;
            ptr=3;
            break;
        }
        case 3:                             /* address */
        {
            addrH=toint(rx_data)*16;        // get the 1st digit of the address byte
            ptr=4;
            break;
        }
        case 4:
        {
            addrH+=toint(rx_data);          // get the 2st digit of the address byte
            calsum+=addrH;
            ptr=5;
            break;
        }

        case 5:
        {
            addrL=toint(rx_data)*16;        // get the 3st digit of the address byte
            ptr=6;
            break;
        }
        case 6:
        {
            addrL+=toint(rx_data);          // get the 4st digit of the address byte
            calsum+=addrL;
            ptr=7;
            break;
        }
        case 7:                             /* data type */
        {
            type=toint(rx_data)*16;         // get the 1st digit of the data type
            ptr=8;
            break;
        }
        case 8:
```

```
        {
            type+=toint(rx_data);           // get the 2st digit of the data type
            calsum+=type;
            ptr=9;
            if(length==0x00)                /* end of hex file */
            {
                if(type==0x01)
                {
                        ptr=11;             // go to check sum
                }
            }
            break;
        }
        case 9:                             /* data value */
        {
            hexdata=toint(rx_data)*16;      // get the 1st digit of one data
            ptr=10;
            break;
        }
        case 10:
        {
            hexdata+=toint(rx_data);        // get the 2st digit of one data
            ptr=9;                          // write one byte record
            if(!error_occur){               // if no error, write code data
                if(!verify_implement)       // if not verify, write
                {
                    write(addrH,addrL+nb_byte,hexdata);
                }
                else                        // if verify, check memory
                {
                    send_addr(addrH,addrL+nb_byte);
                    read_code_data();
                    if(hexdata!=get_return_data())
                        verify_pass=0;      // if not equal, clear verify_pass
                }
            }
            nb_byte++;
            if(addrL+nb_byte-1==0xff){      // lower address overflow, a increment
                addrH++;                    // of higher address
            }
            calsum+=hexdata;                // check sum

            if (length==nb_byte)            // if no data, go to check sum
            {
                ptr=11;
            }

            break;
        }
        case 11:                            /* check sum */
        {
            checksum=toint(rx_data)*16;     // get the 1st digit of the check sum
            ptr=12;
            break;
        }
        case 12:
        {
            checksum+=toint(rx_data);       // get the 2st digit of the check sum
            ptr=0;
            calsum=(~calsum)+1;
            if(checksum !=calsum){
                error_occur=1;              // set the error flag
            }
            if(length==0x00)                /* end of hex file */
            {                               // detect 00000001ff
                if(type==0x01)
                {
                    if(checksum==0xff)
                    {
                        rx_continue=0;      // detected, disable serial interrupt
                        if(!error_occur){   // no error, display ok state
                            if(!verify_implement)
                                printf("\nprogram completed!");
                            else
                                printf("\ntransmit completed!");
                        }
```

```
                        else
                        {
                            printf("\nCheck sum error!");
                        }
                    }
                }
            }

            break;
        }
    }// end of switch

    return;
}
```

C:\WTP_ROOT\89C51_52_Programmer_7_28_03\89C51_52_Programmer\config.h 07/28/03 01:04:18

```
/*-------------------------------------------------------------------------+
| Name   : config.h                                                        |
| purpose:                                                                 |
| Define header file for c files, define dependant definition to enhance program |
| portability and define the control pin for hardware                      |
+-------------------------------------------------------------------------*/

/*_____ I N C L U D E S ____________________________________________________*/
#include <reg51.h>
#include <stdio.h>
#include <ctype.h>

/*_____ M A C R O S ________________________________________________________*/
#define FALSE 0
#define TRUE 1
#define VPP5V 1
#define VPP12V 0

/*_____ H A R D W A R E ____________________________________________________*/

sbit VPP = P3^3;                        // set for 5V, clr for 12V
sbit ALE = P3^4;                        // program enable pulse
sbit MT1 = P3^7;                        // p26
sbit MT2 = P3^6;                        // p27
sbit MT3 = P2^7;                        // p36
sbit MT4 = P3^5;                        // p37
sbit CLK = P3^2;                        // clock of D-flip-flop

/*_____ P R O T O C O L ____________________________________________________*/

// uart.c
void uart_init(void);
void en_serInt(void);
void program(void);
void verify(void);
void process(unsigned char rx_data);

// advance.c
void read(void);
void write(unsigned char addrH,addrL,hexdata);
void chip_identification(void);
unsigned int AT89C51_identification(void);
unsigned int AT89C52_identification(void);
void write_all(void);

// command.c
void write_code_data(void);
void read_code_data(void);
void read_signature(void);
void erase(void);
void write_lock(unsigned char k);
void send_addr(unsigned char addrH,addrL);
void send_data(unsigned char hexdata);
void ctrlcmd(unsigned int i,j,k,l);

// hex.c
unsigned char get_return_data(void);
void print_return_data(void);
void print_hexdata(unsigned char hexdata);
unsigned char toasciiHex(unsigned char bits_data);
```

C:\WTP_ROOT\89C51_52_Programmer_7_28_03\89C51_52_Programmer\advanced.c 07/28/03 00:59:08

```
/*-------------------------------------------------------------------------------+
| Name   : advanced.c                                                            |
| purpose:                                                                       |
| The source code for the advance function of the programmer, read a chip and    |
| generate a intel-Hex80 file, program the entire chip and chip identification   |
| is provided for the advance or main function of the programmer.                |
+-------------------------------------------------------------------------------*/
#include "config.h"

/*
    read chip data from address 0000 to 0fff, this function will generate intel-Hex80
    for the program of another chip, length, address, type, data and check sum field
    is provided.
*/
void read(void)
{
    static unsigned char ptrH,ptrL;
    static int i,j,k;
    static unsigned char checksum;
    ptrH=0x00;                              // restore address, as its attribute
    ptrL=0x00;                              // is static
    printf("\n");
    for(k=0;k<=0x0f;k++)
    {
        for(j=0;j<=0x0f;j++)
        {
            checksum=0x00;                  // add for check sum
            checksum+=0x10;                 // length
            checksum+=ptrH;                 // address
            checksum+=ptrL;
            printf(":10");                  // length
            print_hexdata(ptrH);            // address
            print_hexdata(ptrL);
            printf("00");                   // data type
            for(i=0;i<=0x0f;i++)
            {
                send_addr(ptrH,ptrL);
                read_code_data();
                checksum+=get_return_data();// calculate check sum
                print_return_data();        // screen out
                ptrL++;
            }
            checksum=(~checksum)+1;
            print_hexdata(checksum);
            printf("\n");
        }
        ptrH++;
    }
    printf(":00000001FF");                  // end of hex file
}

/*
    write code data in specific address with desired hex data, defined by slave
    chip P1-P2.3 and P0 for 89c51 or P1-P2.4 and P0 for 89c52
*/
void write(unsigned char addrH,addrL,hexdata){

    send_addr(addrH,addrL);
    send_data(hexdata);
    write_code_data();
}


/*
    this function implemented for testing how long the ale pulse required for programming
    a chip, and see which address the programmer can not writed. By chaning the value of
    ale pulse, it is easily determine what is the valid value for ale pulse.
*/
void write_all(void)
{
    static unsigned char ptrH,ptrL,hexdata;
    static int i,j,k;
    printf("\nChip code write [Test mode]");
    printf("\nThis function will write the hex data same as it lower address value");
    printf("\nwrite hex data in addresss 0000 to 0FFF");
        ptrH=0x00;
```

```
        ptrL=0x00;
        hexdata=0x00;
    for(k=0;k<=0x0f;k++)
    {
        for(j=0;j<=0x0f;j++)
        {
            for(i=0;i<=0x0f;i++)
            {
                write(ptrH,ptrL,hexdata);
                ptrL++;
                hexdata++;
            }
        }
        ptrH++;
    }
}

/*
    determine the chip is AT89C51 or not, if target chip is found, set
    AT89C51_12V_programming, clear if not. set indicate programmer is
    supported.
*/
unsigned int AT89C51_identification(void)
{
    static unsigned char hexdata;
    static unsigned int AT89C51_12V_programming;

    AT89C51_12V_programming=1;
    send_addr(0x00,0x30);                   /* manufacture */
    read_signature();                       // send command
    hexdata=get_return_data();              // get return data
    if(hexdata!=0x1E)
    {
        AT89C51_12V_programming=0;
    }

    send_addr(0x00,0x31);                   /* MPU type */
    read_signature();                       // send command
    hexdata=get_return_data();              // get return data
    if(hexdata!=0x51)
    {
        AT89C51_12V_programming=0;
    }

    send_addr(0x00,0x32);                   /* programming voltage */
    read_signature();                       // send command
    hexdata=get_return_data();              // get return data
    if(hexdata!=0xFF)
    {
        AT89C51_12V_programming=0;
    }

    return AT89C51_12V_programming;
}

/*
    determine the chip is AT89C52 or not, if target chip is found, set
    AT89C52_12V_programming, clear if not. set indicate programmer is
    supported.
*/
unsigned int AT89C52_identification(void)
{
    static unsigned char hexdata;
    static unsigned int AT89C52_12V_programming;

    AT89C52_12V_programming=1;
    send_addr(0x00,0x30);                   /* manufacture */
    read_signature();                       // send command
    hexdata=get_return_data();              // get return data
    if(hexdata!=0x1E)
    {
        AT89C52_12V_programming=0;
    }

    send_addr(0x00,0x31);                   /* MPU type */
    read_signature();                       // send command
```

```
    hexdata=get_return_data();              // get return data
    if(hexdata!=0x52)
    {
        AT89C52_12V_programming=0;
    }

    send_addr(0x00,0x32);                   /* programming voltage */
    read_signature();                       // send command
    hexdata=get_return_data();              // get return data
    if(hexdata!=0xFF)
    {
        AT89C52_12V_programming=0;
    }

    return AT89C52_12V_programming;
}

/*
    This function is implemented to determine whether the programmer is
    supported or not
*/
void chip_identification(void)
{
    printf("\nchip identification....");
    if(AT89C51_identification())                // if AT89C51
    {
        printf("\nAT89C51 found");
        printf("\n12V Programming found");
        printf("\nprogrammer supported!");
    }
    else if(AT89C52_identification())           // if AT89C52
    {
        printf("\nAT89C52 found");
        printf("\n12V Programming found");
        printf("\nprogrammer supported!");
    }
    else                                        // if not supported
    {
        printf("\nNOT AT89C51 or NOT AT89C52 or");
        printf("\nNOT 12V Programming");
        printf("\nprogrammer NOT supported!");
    }
}
```