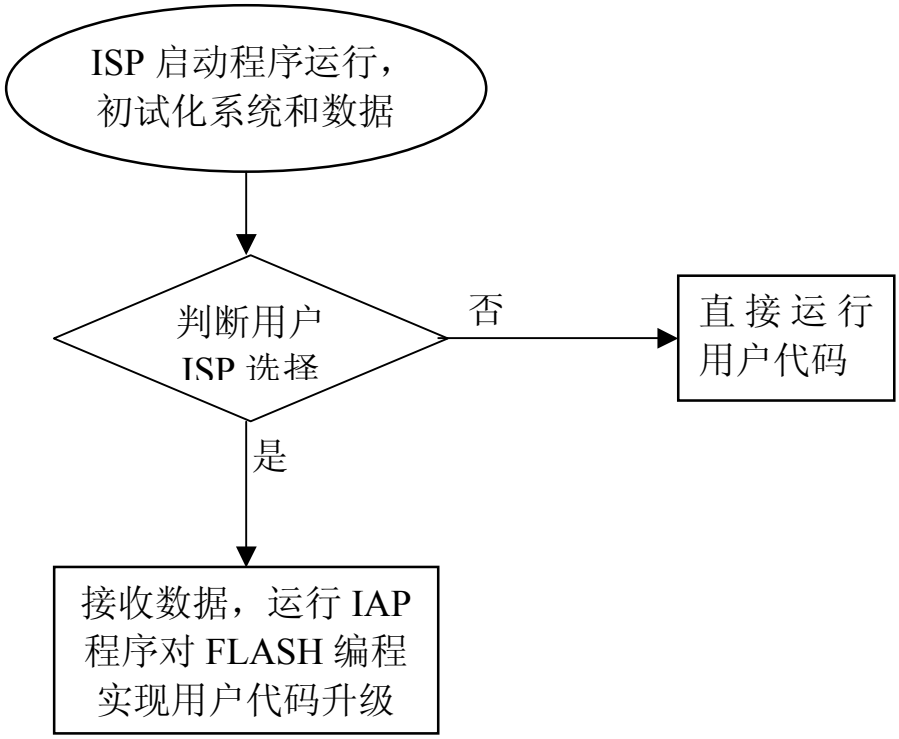
设计自己的 **ISP** 程序

许多应用系统中都需要进行程序代码升级，如果程序代码在外部 FLASH 存储其中，实现程序代码升级可以对外部 FLASH 直接操作，分块擦除和编程。但对于一些系统程序代码在单片机内部的，就要求此单片机支持 IAP(In-Application-Programming)功能。本文即介绍此情况下的 ISP(In-System-Programming)程序设计方法，以及在 SST 和 STC 单片机上的具体实现。

一. **ISP** 实现基本结构

ISP 实现方式有很多种，但大致都遵循以下流程：



其中，判断用户 ISP 选择，一般有以下几中方式：

1. 连接计算机系统，由系统的命令选择

进入用户 ISP 选择判断时，先由单片机发送特定特征数据，然后等待命令数据，如果在一定的时间内，接收到计算机系统发出的选择命令则进入用户代码升级，否则直接跳转用户代码执行。

2. 由用户板上的跳线选择

一般利用单片机空余的端口，设计一个代码升级选择跳线。进入用户 ISP 选择判断时，单片机可以直接根据此端口的状态判断进入用户代码升级还是直接跳转用户代码执行。

3. 由用户板操作功能选择

在用户板的功能菜单或功能组合中允许用户选择代码升级功能，同时，在外部存储器中存放相应的标志。当选择代码升级功能时，在外部存储器中写入特定数据，然后程序复位，进入用户 ISP 选择判断时，判断外部存储器中的数据，如果符合条件则进入用户代码升级，否则直接跳转用户代码执行。

二. IAP 程序设计

目前许多单片机都支持 IAP 功能，一般这些单片机内部都包含两个 FLASH 块，在一块 FLASH 中运行的程序可对另一块 FLASH 进行擦除和重新编程。一般我们都将 ISP 程序存放在容量较小的一块 FLASH 中(block1)，而将用户代码存放在容量较大的一块 FLASH 中(block0)。Block1 中的 ISP 程序对 block0 中的用户代码进行擦除和重新编程。

我们以两种 51 系列兼容的 FLASH 单片机为例，介绍 IAP 的程序设计。一种是 SST89C54，另一种是 STC89C516RD。

1. SST89C54 相关特性：

SST89C54 内部有 20K(16K+4K)程序存储器，统一编址。Block0：0000H～3FFFH；Block1：F000H～FFFFH；Block1 可以选择映射到 0000H 地址开始的 1K/2K/4K 程序区。

2. STC89C516RD 相关特性：

SST89C516RD 内部有 72K(64K+8K)程序存储器。Block0：0000H～FFFFH，Block1 可以选择映射到 0000H 地址开始的 8K 程序区(上电复位确省为地址映射)。

SST 与 STC 单片机的 IAP 操作几乎完全相同，附件 IAP.C 中给出了 IAP 函数的 C 程序源代码。要特别注意的是，Block0_erase 函数中对于 block0 的选择，两种单片机是不同的(正好相反)。

三. ISP 程序到用户代码的切换

在设计中一般都将 ISP 程序设计为上电复位后运行的程序，如果不需要用户代码升级或升级完成后，就要将程序切换到用户代码执行。ISP 程序到用户代码的切换，不同的单片机各不相同。

1. SST89C54 程序区 Block1 到 Block0 的切换实现：

SST89C54 单片机在烧录时，将 ISP 程序写到 Block1，并且烧录映射选择位 RB0/RB1(RE-MAP[1:0])。这样，程序上电复位时，自动将 Block1 映射到 0000H 地址开始的 4K 程序区，进

入 ISP 程序执行。由于 Block1 同时还分配在地址：F000H～FFFFH，因此，编译生成 ISP 程序代码时，设定所有的地址范围都在 F000H～FFFFH。需要切换到用户代码(Block0)运行时，修改 SFCF[7]控制位 VIS，将 Block1 的 0000H 地址映射取消，然后程序跳转到地址 0000H 执行，则开始运行 Block0 中的用户代码程序。

附件 ISP.C 中给出了 ISP 的 C 程序源代码。要注意的是：此程序在 Keil-C 中要建立工程文件，包含 IAP.C 函数以及 STARTUP.A51，并且在 IAP.C 和 ISP.C 中都要去掉 STC 的定义。为了将地址范围设定到 F000H～FFFFH，要将 STARTUP.A51 中程序入口地址由 0 改为 0F000H，如下：

```
                CSEG    AT      0F000H
?C_STARTUP:     LJMP    STARTUP1
```

还要修改编译选项设置 Target 选项卡中 Off-chip Code memory：Start =0xF000；Size=0x1000；还要设置 C51 选项卡中 Interrupt Vectors at address：0xF000。

2. STC89C516RD 程序区 Block1 到 Block0 的切换实现：

STC89C516RD 单片机在烧录时，将 ISP 程序写到 Block1。(注意：并不烧录 SC0/SC1 位)。单片机上电复位时，由于缺省地 Block1 映射到 0000H 地址开始的 8K 程序区，进入 ISP 程序执行。需要切换到用户代码(Block0)运行时，ISP 修改 SFCF[1]

控制位 SWR，产生一个软复位(Software Reset)。由于 SC0 和 SC1 都未烧录，程序软复位后，Block1 将不再映射到 0000H 地址映，则开始运行 Block0 中的用户代码程序。

附件 ISP.C 中给出了 ISP 的 C 程序源代码。要注意的是：此程序在 Keil-C 中要建立工程文件，包含 IAP.C 函数，并且在 IAP.C 和 ISP.C 中都要保留 STC 的定义。

四. 与计算机连接的通讯协议

升级用户代码时，需要与计算机进行通讯，一般采用 RS232 串行通讯，数据协议采用简单协议。此协议参考了 ADuC812 单片机的 ISP 数据协议。(ADuC812 单片机硬件内置 ISP 程序)

1. 复位命令(计算机->单片机)

计算机发送四字节复位命令：21H,5AH,00H,A6H，单片机返回复位信息。

2. 复位信息(单片机->计算机)

复位信息为 25 字节，前三字节为单片机公司特征字符(如：“ADI”“SST”“STC”)，最后一字节为效验和。

3. 数据包格式(计算机->单片机)

计算机发送数据包格式：07H，0EH，长度，数据，效验和(长度与数据的效验和)。

4. 擦除命令

计算机发送数据包，其中数据只有一字节，内容为：字符’A’

或’C’。单片机擦除用户程序区后返回一字节 06H 表示成功；05H 表示失败。

5. 编程命令

计算机发送数据包，其中数据内容：’W’，00H，地址高字节，地址低字节，程序数据。单片机返回一字节 06H 表示成功；05H 表示失败。

6. 运行用户程序

计算机发送数据包，其中数据只有一字节，内容为：字符’U’。单片机返回一字节 06H 表示成功，然后跳转用户程序运行。

附件 Download.C 提供了计算机下载软件的 C 程序源代码。

五. 结束语

根据以上 ISP 程序设计思路和实例，大家可以修改 ISP 初始代码，或者丰富其他的 ISP 功能(如：读功能，口令控制等)，尝试设计自己的 ISP 程序，定可以为自己的系统增色不少。

魏东

tonywei@tom.com

http://bbgrow.126.com

2004-7-25

附件一：IAP.C

```
/*************************************************************************
IAP.C     SST 和 STC 单片机 IAP 操作函数
      魏东(tonywei@tom.com)
      2004.7.23
*************************************************************************/
#define STC          /*定义为 STC 单片机(如果是 SST 单片机则去掉此行)*/
/*************************************************************************
* SFR Memory Addresses
*************************************************************************/
sfr SFCF = 0xB1; /*SuperFlash Configuration*/
sfr SFCM = 0xB2; /*SuperFlash Command*/
sfr SFAL = 0xB3; /*SuperFlash Address Low*/
sfr SFAH = 0xB4; /*SuperFlash Address High*/
sfr SFDT = 0xB5; /*SuperFlash Data*/
sfr SFST = 0xB6; /*SuperFlash Status*/
/*************************************************************************
* MCU IAP Commands
*************************************************************************/
#define SFCM_BE 0x0D; /*block-Erase IAP cmd*/
#define SFCM_SE 0x0B; /*Sector-Erase IAP cmd*/
#define SFCM_VB 0x0C; /*Byte-Verify IAP cmd*/
#define SFCM_PB 0x0E; /*Byte-Program IAP cmd*/

bit iap_error=0;

void block0_erase(void);
void sector_erase(unsigned int);
void byte_program(unsigned int, unsigned char);
unsigned char byte_verify(unsigned int);
unsigned char ready(void);

/*************************************************************************
* Block0-Erase Subroutine
*************************************************************************/
void block0_erase(void)
{
      SFCF = SFCF | 0x40;           /*enable IAP */
#ifdef STC
      SFAH = 0xf0;              /*STC 单片机选择 block0*/
#else
      SFAH = 0x00;              /*SST 单片机选择 block0*/
#endif
      SFDT = 0x55;
      SFCM = SFCM_BE;                  /*issue block erase command */
      if(!ready()) iap_error=1;
}

/*************************************************************************
* Sector-Erase Subroutine
*************************************************************************/
void sector_erase(unsigned short int destAddr)
{
      SFCF = SFCF | 0x40;           /*enable IAP */
```

```
    SFAH = destAddr>>8;          /*load high order address byte*/
    SFAL = destAddr;         /*load low order address byte */
    SFCM = SFCM_SE;                  /*issue sector erase command */
    if(!ready()) iap_error=1;
}

/*************************************************************************
* Byte-Program Subroutine
*************************************************************************/
void byte_program(unsigned short int destAddr, unsigned char dataByte)
{
    SFCF = SFCF | 0x40;          /*enable IAP */
    SFAH = destAddr>>8;          /*load high order address byte*/
    SFAL = destAddr;         /*load low order address byte */
    SFDT = dataByte;         /*load data to be programmed */
    SFCM = SFCM_PB;                  /*issue byte program command */
    if(!ready()) iap_error=1;
}

/*************************************************************************
* Byte-Verify Subroutine
*************************************************************************/
unsigned char byte_verify(unsigned short int destAddr)
{   unsigned char readByte;

    SFCF = SFCF | 0x40;     /*enable IAP */
    SFAH = destAddr>>8;   /*load high order address byte*/
    SFAL = destAddr;   /*load low order address byte */
    SFCM = SFCM_VB;          /*issue byte verify command */
    readByte = SFDT;
    SFCF = SFCF & 0xBF;  /*turn off IAP*/
    SFDT = 0;
    return readByte;
}

/*************************************************************************
* Ready Subroutine
* Purpose: To check if the IAP operation is completed.
* When it is done, turn off IAP configuration.
*************************************************************************/
unsigned char ready()
{   unsigned long TimeOut;

    for(TimeOut=0;TimeOut<100000;TimeOut++)
    {   if ((SFST&4) == 0)           /* Check if IAP is done */
        {                    /* IAP is done */
            SFCF = SFCF & 0xBF;  /* turn off IAP*/
            SFDT = 0;            /* any value other than 0x55 */
            return 1;        /* IAP operation is completed*/
        }
    }
    SFCF = SFCF & 0xBF;          /*turn off IAP*/
    SFDT = 0;                    /*any value other than 0x55*/
    return 0;               /*IAP operation is NOT completed before time out*/
}
```

附件二： ISP.C

```
/*****************************************************************************
SST 或 STC 单片机 ISP 程序
      魏东(tonywei@tom.com) 2004.7.25
      SST89C54 或 STC89C516RD,频率:7.372848MHz
*****************************************************************************/
#include "reg51.h"
/*  对 IAP.C 调用的定义   */
extern bit iap_error;
extern void block0_erase(void);
extern void sector_erase(unsigned int);
extern void byte_program(unsigned int, unsigned char);
extern unsigned char byte_verify(unsigned int);
extern unsigned char ready(void);

//#define STC       /*定义为 STC 单片机*/

sfr WDTC    = 0xC0;
sfr WDTD    = 0x86;
sfr SFCF    = 0xB1;

#define CON_OSC 7372848                         /*  振荡频率  */
#define CON_BPS 256 - ( CON_OSC / 12 / 32 / 6400 )     /* 6400BPS 时间常数  */
#define CON_MSC CON_OSC/360000                          /*  延时  */

/*  变量定义  */
unsigned char com_buf[58];
unsigned int nAddress = 0;    /*  编程器编程地址  */

/*  函数定义  */
unsigned char com_getch(void);          /*  串口接收 1 字符  */
void com_putch(char);               /*  串口发送 1 字符  */
void com_putinfo(void);             /*  串口发送复位信息  */
void delay_ms(unsigned char);           /*  延时(单位:0.1 毫秒) */
void delay_s(unsigned char);        /*  延时(单位:0.01 秒) */

void (* pc_0)(void);                /*  程序地址 0000 */
void goto_pc0(void);                /*  跳转到程序地址 0000 运行  */

/*****************************************************************************/

/*  串口接收 1 字符  */
unsigned char com_getch(void)
{     unsigned char c;

      RI=0;
      while(RI==0);
      c=SBUF;
      return c;
}

/*  串口发送 1 字符  */
```

```
void com_putch(char c)
{
    TI=0;
    SBUF=c;
    while(TI==0);
}

/* 串口发送复位信息 */
void com_putinfo(void)
{
#ifdef STC
    unsigned char code *reset_info = "STC 89C516 101              ";
#else
    unsigned char code *reset_info = "SST 89C54   101              ";
#endif
    unsigned char cr,ci;

    cr=0;
    for(ci=0;ci<24;ci++)
    {   com_putch(reset_info[ci]);
        cr=cr + reset_info[ci];
    }
    cr = 0 - cr;
    com_putch(cr);          /* 发送效验和 */
}
/*****************************************************************************/

/* 延时(单位:0.1 毫秒) */
void delay_ms(unsigned char ci)
{   unsigned char cj;

    while(ci)
    {   for(cj=0;cj<CON_MSC;cj++) ;   /* 延时(12/CON_OSC)*3 */
                        /*   编译为汇编如下：
                        delay_ub:INC      R6        ;1
                        CJNE    R6,delay_ub   ;2
                        */
        ci --;
    }
}

/* 跳转到程序地址 0000 运行 */
void goto_pc0(void)
{   unsigned char ci;

    EA=0;           /* 禁止中断 */
    ci = SFCF;
#ifdef STC
    ci = (ci&0xfc)|2;
    SFCF = ci;    // SFCF[1]从 0 到 1 产生软复位，程序从 0 开始执行，并且 SC0 按照
其设置(从 block0 运行)
    while(1);
#else
    ci = (ci&0xfc)|0x80;
    SFCF = ci;
    pc_0 = 0;
```

```
        pc_0();
    #endif
    }

    /*****************************************************************************/

    void main (void)
    {
        unsigned char cr,ci,cj;
        unsigned int ui;

        PCON=0;
        SCON=0x50; /* 置串口工作方式 1(T1 用作串口波特率发生) */
        TMOD=0x21; /* 置定时器 1 工作方式 2,定时器 0 工作方式 1 */
        TH1=CON_BPS;  /* 波特率时间常数 */
        TL1=CON_BPS;
        TR1=1;        /* 启动定时器 1 */
        EA=0;         /* 禁止中断 */
        TI=1;              /* set TI to send first char of UART   */

        WDTD = 0;
        WDTC = 0;   /* 关看门狗 */
        SFCF |= 0x80;
        for(ci=0;ci<5;ci++) delay_ms(100);     /* 等待 50mS 稳定时间 */
        com_putinfo();                         /* 发送复位信息 */
        ui=0;
        RI=0;
        while(1)
        {   delay_ms(1);
            if(RI)
            {   ci=SBUF;
                RI=0;
                ui=0;
                if((ci==0x21)||(ci==0x07)) break; /* 接收到命令就进入通讯 */
            }
            ui++;
            if(ui>=10000) goto_pc0();    /* 等待 1 秒,跳转到用户程序运行 */
        }
      while(1)
      { iap_error=0;
        if(ci==0x21)
        {   for(ci=0;ci<3;ci++) com_buf[ci]=com_getch();
            if((com_buf[0]==0x5a)||(com_buf[1]==0)||(com_buf[2]==0xa6))
                com_putinfo();             /* 发送复位信息 */
            else com_putch(5);       /* 返回错误标志 */
        }
        else if(ci==0x07)   /* 数据包：07H,0EH,包长度,'W',00H,地址高字节,地址低字节，
程序数据,效验和 */
        {   for(ci=0;ci<2;ci++) com_buf[ci]=com_getch();
            if((com_buf[0]==0x0e)&&com_buf[1])
            {   cj = com_buf[1];
                cr = cj;
                for(ci=0;ci<=cj;ci++)
                {   com_buf[ci] = com_getch();
                    cr += com_buf[ci];
                }
```

```
                    if(cr) com_putch(5);            /* 返回错误标志 */
                    else if((com_buf[0]=='W')&&(com_buf[1]==00)&&(cj>4))
                    {   nAddress = com_buf[2]*256 + com_buf[3];
                        cj -= 4;
                        for(ci=0;ci<cj;ci++)
                        {
    #ifdef STC
                            byte_program(nAddress,com_buf[4+ci]);   /* 写 */
    #else
                            if(nAddress<0x8000) byte_program(nAddress,com_buf[4+ci]);  /*
SST 限制写地址 */
    #endif
                            nAddress ++;
                        }
                        if(iap_error) com_putch(5);  /* 返回错误标志 */
                        else com_putch(6);      /* 返回正确标志 */
                    }
                    else if((com_buf[0]=='A')||(com_buf[0]=='C'))
                    {   block0_erase();             /* 擦除器件 */
                        if(iap_error) com_putch(5);  /* 返回错误标志 */
                        else com_putch(6);      /* 返回正确标志 */
                    }
                    else if(com_buf[0]=='U')
                    {   com_putch(6);       /* 返回正确标志 */
                        goto_pc0();         /* 跳转到用户程序运行 */
                    }
                    else com_putch(5);      /* 返回错误标志 */
                }
                else com_putch(5);      /* 返回错误标志 */
            }
            else com_putch(5);      /* 返回错误标志 */
            ci = com_getch();
        }
    }
```

附件三：DOWNLOAD.C

```
/*********************************************************************
ISP 下载程序
      魏东(tonywei@tom.com) 2004.7.25

Program files must be of INTEL standard HEX format.

This code was compiled using Borland C++ version 5.0 for DOS.

The clock() function is used to determine communication timeouts.
[the tick resolution is assumed to be 1/18th of a second.    It may be
different on other compilers or in memory models other than LARGE]

*********************************************************************/

#include <stdio.h>      /*for reading hex file*/
#include <string.h>    /*for copying parameter strings*/
#include <stdlib.h>    /*for exit() function*/
#include <time.h>       /*for clock() and clock_t (timeouts)*/
#include <conio.h>      /*for reading/writing the UART hardware ports*/
#include <ctype.h>      /*for toupper() function*/

/*********************************************************************/

#define DEFAULTTIMEOUT 5    /*about a quarter of a second*/
#define ERASETIMEOUT 80      /*about 4 seconds*/

#define VER " 1.01 "
#define DATE "25 July,2004    "


/*function prototypes*/

int    reset(int *major, int *minor);
int    download(int rev, char *program);
int    run();

void showTitle(void);
void showInstructions(void);

int    decodeCommandLine(int argc, char *argv[],
                                 int *comport, double *crystal,
                                 char *filename,
                                 int *dontEraseData,
                                 int *autoRun);

int    validateParameters(int comport, double *crystal,
                                   char *filename);

int    openComms(int comport, long baudrate);
int    initcommshardware(int comport, unsigned long int baudrate);
int    dataavail(void);
void charin(unsigned char *ch);
int    clearxmit(void);
```

```
void charout(unsigned char ch);

int    readBlockFromFile(FILE *FptrIn, unsigned long *addr,unsigned char *data);

int    sendPacket(unsigned char *message, int length);
int    sendLoaderPacket(int len, char *data);
int    waitForAck(int timeout);

int    waitForSignature(char *sig, int *major, int *minor);
int    enable(int revision, int dontEraseData);
int    downloadU(char *program);
int    runU(long address);


/*****************************************************************/
/*                        BEGINNING OF MAIN PROGRAM                */
/*****************************************************************/

void main(int argc, char *argv[])
{
   int comport=1;                  /*comport to connect through*/
   long baudrate=9600;             /*communications baud rate*/
   char filename[256]="";          /*name&location of program to download*/
   double crystal=11.059200;       /*crystal frequency of the target*/
   int rMajor, rMinor;             /*rev number of the target firmware*/
   int dontEraseData = 0;          /*default to erasing all target memory*/
   int autoRun=0;                  /*don't automatically run target prog*/

   showTitle();

   decodeCommandLine(argc,argv,
                     &comport,&crystal,
                     filename,
                     &dontEraseData,
                     &autoRun);

   if(!validateParameters(comport,&crystal,filename))
      exit(1);

   /*work out actual baud rate*/
   baudrate=crystal/1152.0;

   openComms(comport,baudrate);

   /*reset the target device*/
   if(reset(&rMajor,&rMinor))
   { rMajor=2;
      if(enable(rMajor,dontEraseData))
      {
         if(download(rMajor,filename))
         {
            if(autoRun)
            {
               run();
            }
         }
      }
```

```
   }
}

/*****************************************************************/
/*                 END OF MAIN PROGRAM , BEGINNING OF FUNCTIONS                */
/*****************************************************************/

/*extract the filename etc. from the command line parameters*/
int    decodeCommandLine(int argc,char *argv[],
                                int *comport,double *crystal,
                                char *filename,
                                int *dontEraseData,
                                int *autoRun)

{
   int i;
   char option;
   for(i=1;i<argc;i++)              /*loop thru all arguments on cmd line*/
                                        /*skip argv[0] as that's the appl. name*/
   {
      if(    (argv[i][0]=='-')       /*see if this argument is a parameter*/
          ||(argv[i][0]=='\\')      /*by checking first char in the string*/
          ||(argv[i][0]=='/'))
      {
         option=toupper(argv[i][1]);
         if(argv[i][2]==':')         /*3rd char in param string must be ';'*/
         {
            switch(option)              /*check the second char to see what*/
                                            /*parameter it is*/
            {
            case 'C':                                                   /*comport parameter*/
               *comport=argv[i][3]-'0';
               break;
            case 'F':                                                   /*crystal frequency*/
               sscanf(&argv[i][3],"%lf",crystal);
               break;
            }
         }
         else if ((option=='H')||(option=='?'))
            showInstructions();
         else if (option=='R')
            *autoRun=1;
      }
      else                                          /*assume argument is a filename*/
         strcpy(filename,argv[i]);
   }
   return 0;
}

/*check that the specified parameters are valid*/
int validateParameters(int comport,double *crystal,
                                char *filename)
{
   if(strlen(filename)==0)
   {
      printf("No Program file specified\n");
      return 0;                                     /*false*/
```

```
    }
    if((comport<1) || (comport>8))
    {
        printf("invalid comport number\n");
        return 0;                                   /*false*/
    }

    if(*crystal<70)
        *crystal*=1000000.0;    /*convert from MHz into Hz*/

    if((*crystal<2000000.0) || (*crystal>70000000.0))
    {
        printf("invalid target crystal frequency\n");
        return 0;                                   /*false*/
    }
    return 1;                                       /*true*/
}

void showTitle(void)
{
    printf("\nDOWNLOAD.EXE: ISP program downloader ver "VER"\n");
    printf(DATE" WeiDong(tonywei@tom.com) .\n");
}

void showInstructions(void)
{
    printf("Program options:\n");
    printf("    a filename must be specified on the command line\n");
    printf("    /C:n        select the com port (default is 1)\n");
    printf("    /F:n.n    select the target crystal freq in MHz (default 11.0592)\n");
    printf("    /R            run the program from the beginning (0000 hex)\n");
    printf("Example:\n");
    printf("    DOWNLOAD filename.hex /c:1 /f:16 /r\n");
}

/*****************************************************************/

/*set up the communications link to the target*/
int openComms(int comport,long baudrate)
{
    int rcvd;
    unsigned char ch;
    printf("Initialising Com%d at %ld baud:",
                comport,baudrate);
    if(!initcommshardware(comport,baudrate))
    {
        printf("Failed.\n");
        return 0;           /*failure*/
    }
    printf("OK!\n");

    if(dataavail()){
        /*need to purge the com channel*/
        rcvd=0;
        printf("Clearing receive buffer:");
        while(dataavail()){           /*see if any data from the target*/
            charin(&ch);                    /*get the data*/
```

```
            printf(".");
            rcvd++;
            if(rcvd>100){
               printf("\nRecieve buffer overflow: terminating\n");
               return 0; /*failure*/
            }
         }
         printf("Cleared\n");
      }
      return 1;                  /*success*/

}

/*attempt to perform a software reset on the target*/
int    reset(int *major,int *minor)
{
   unsigned char data[4];
   char sanatizedSig[100];
   *major=0;
   *minor=0;
   printf("Resetting the target device: ");
   /*Send the rev1 loader reset command*/
   charout('!');
   /*now send the rest of the reset packet*/
   /*the first byte, the '!' has already been sent*/
   data[0]='Z';            /*command code*/
   data[1]=0-'Z';         /*set up the checksum*/
   data[2]=0;               /*command parameter*/
   if(sendPacket(data,3))
      if(waitForSignature(sanatizedSig,major,minor))
      {
         printf("%s reset OK!\n",sanatizedSig);
         return 1;               /*success*/
      }
   printf("Failed\n");
   return 0;                  /*failure*/
}



/*download the program to the target device*/
int download(int rev,char *program)
{
   int result;
   printf("Downloading %s: ",program);
   switch(rev)
   {
   case 0:
      result=1;
      break;
   case 1:
      result=downloadU(program);
      break;
   case 2:
      result=downloadU(program);
      break;
   default:
```

```
      result=2;
      break;
   }
   switch(result)
   {
   case 0:
      printf("Downloaded OK!\n");
      break;
   case 1:
      printf("Error: Target was not reset\n");
      break;
   case 2:
      printf("Error: This version of DOWNLOAD.EXE is not compatible with this target\n");
      break;
   case 3:
      printf("Error: Failed to download\n");
      break;
   case 4:
      printf("Error: File not found\n");
      break;
   }
   return (result==0);
}


/* send the target device a command to commence user code execution
     from a specified address */
int run()
{
   int result;
   printf("Running program from address 0x0000");
   result=runU(0);
   switch(result)
   {
   case 0:
      printf("Run OK!\n");
      break;
   case 1:
      printf("Not reset\n");
      break;
   case 2:
      printf("Error: This version of DOWNLOAD.EXE is not compatible with this target\n");
      break;
   case 3:
      printf("Failed to run\n");
      break;
   }
   return (result==0);
}

/*********************************************************************/

int waitForSignature(char *sig,int *major,int *minor)
{
   int i=0;
   int state=0;
   clock_t starttime;              /*clock variable for determining timeouts*/
```

```
unsigned char ch;
unsigned char checksum=0;
unsigned char data[30];
strcpy(sig,"none");
starttime=clock();              /*get start clock value*/
while(1)                            /*keep trying....*/
{
   if(dataavail())              /*see if any data from the target*/
   {
      charin(&ch);                  /*get the data*/
      data[state]=ch;
      checksum+=ch;
      switch(state) /*see what we are doing....*/
      {                     /*watch for fallthroughs in this switch statement*/
      case 0:             /*waiting for start character ("A")("S")*/
         if((ch=='A')||(ch=='S'))
   {   sig[0]=ch;
            state++; }
         else
            checksum=0;
         break;
      case 1:             /*waiting for second character ("D")("S")("T")*/
         if((ch=='D')||(ch=='S')||(ch=='T'))
   {   sig[1]=ch;
            state++; }
         else
            state=0;
         break;
      case 2:                       /*waiting for third character ("I")("T")("C")*/
         if((ch=='I')||(ch=='T')||(ch=='C'))
   {   sig[2]=ch;
            state++; }
         else
            state=0;
         break;
      case 3:                       /*waiting for device partname*/
      case 4:
      case 5:
      case 6:
      case 7:
      case 8:
      case 9:
      case 10:
      case 11:
      case 12:
      case 13:
      case 14:
      case 15:
      case 16:         /*waiting for hardware id hi byte: ignore*/
      case 17:         /*waiting for hardware id lo byte: ignore*/
      case 18:         /*waiting for firmware id hi byte: ignore*/
      case 19:         /*waiting for firmware id lo byte: ignore*/
      case 20:         /*waiting for age hi byte:              ignore*/
      case 21:         /*waiting for age lo byte:              ignore*/
      case 22:         /*waiting for spare byte:               ignore*/
      case 23:         /*waiting for spare byte:               ignore*/
         state++;
```

```
            break;
         case 24:          /*waiting for checksum*/
            return (checksum==0);
         }
      }
      if(clock()>starttime+150)          /*see if timeout has elapsed*/
         return 0;                       /*timeout failure*/
   }
}

int enable(int revision,int dontEraseData)
{
   int result=0;
   if(revision==1)
      result=1;
   if(revision==2){
      if(dontEraseData){
         printf("Erasing code memory . . ");
         if(sendLoaderPacket(1,"C"))
            result=waitForAck(ERASETIMEOUT);
      }
      else{
         printf("Erasing code and data memory . . ");
         if(sendLoaderPacket(1,"A"))
            result=waitForAck(DEFAULTTIMEOUT);
      }
      if(result)
         printf("OK\n");
      else
         printf("Failed\n");
   }
   return result;
}

int downloadU(char *program)
{
   FILE *pFile;
   unsigned char progData[128];      /*block of data to download*/
   unsigned long progAddr;            /*address to download to*/
   int blockLen;                      /*number of bytes in the block*/
   int result=0;                      /*success*/
               char packet[128];
               int i;

   /*try and open the file*/
   pFile=fopen(program,"r");
   /*check if it opened ok*/
   if(pFile!=NULL)
   {
      while(1)                              /*keep going until all downloaded*/
      {
         /*read a block of data from the file, along with its destination addr*/
         blockLen = readBlockFromFile(pFile,&progAddr,progData);
         if(blockLen>0)
         {
            /*see if we got data ok*/
               result=3;                    /*assume the worst...*/
```

```
                /*build up the program packet*/
                packet[0]='W';                          /*write code memory command*/
                packet[1]=(progAddr>>16)%256;   /*set up the 24bit address*/
                packet[2]=(progAddr>>8) %256;
                packet[3]= progAddr         %256;
                /*copy the program bytes into the packet*/
                for(i=0;i<blockLen;i++)
                   packet[i+4]=progData[i];
                /*send the command as a 'loader' packet*/
                if(sendLoaderPacket(blockLen+4,packet))
                {
                   /*check to see if the target ack's the command*/
                   if(waitForAck(DEFAULTTIMEOUT))
                   {
                      /*print a 'progress dot'*/
                      printf(".");
                      result=0;
                   }
                   else
                      break;    /*break out of the loop*/
                }
             }
             else
                break;              /*break out of the loop*/
          }
          /*close the intel hex file*/
          fclose(pFile);
       }
       else
          result=4;
       return result;
    }

    int runU(long address)
    {
       char data[4];
       /*build the 'run' command packet*/
       data[0]='U';                        /*run from user space command*/
       data[1]=(address>>16)%256;     /*24 bit address*/
       data[2]=(address>>8)%256;
       data[3]= address%256;
       /*send the command as a 'loader' packet*/
       if(sendLoaderPacket(4,data))
       {
          /*check to see if the target ack's the command*/
          if(waitForAck(DEFAULTTIMEOUT))
             return 0;                     /*success*/
       }
       return 3;                           /*failure*/
    }

    /*****************************************************************/

    /*sends a data packet to the target*/
    int sendPacket(unsigned char *message,int length)
    {
       int i;
```

```
   clock_t starttime;              /*clock variable for determining timeouts*/
   starttime=clock();              /*get start clock value*/
   i=0;
   while(i<length)
   {
      if(clearxmit())              /*is uart ready to transmit next byte?*/
      {
         charout(message[i++]);    /*send the byte*/
         starttime=clock();        /*'reset' the timeout*/
      }
      else                                /*uart is not ready*/
         if(clock()>starttime+2)  /*see if timeout has elapsed*/
            return 0;                     /*timeout failure*/
   }
   return 1;
}

/*sends a correctly formated loader command packet to the target*/
int sendLoaderPacket(int len, char *data)
{
   unsigned char packet[128];   /*the loader packet*/
   unsigned char xsum;              /*temp var to calculate the checksum*/
   int i;
   /*set up the 070E header*/
   packet[0]=0x07;
   packet[1]=0x0E;
   /*set up the number of data bytes in the loader packet*/
   packet[2]=len;
   /*copy the data into the packet*/
   for(i=0;i<len;i++)
      packet[3+i]=data[i];
   /*now calculate the checksum of the packet*/
   /*ignore the header, but include the length byte*/
   xsum=len;
   for(i=0;i<len;i++)
      xsum+=packet[i+3];
   packet[len+3]=0-xsum;
   /*send the packet to the target device*/
   return sendPacket(packet,len+4);
}

/*wait for acknowledge from the target device*/
int waitForAck(int timeout)
{
   unsigned char ch;
   clock_t starttime;   /*clock variable for determining the timeouts*/
   starttime=clock();   /*get start clock value*/
   while(1)                  /*keep trying....*/
      if(dataavail())     /*see if any data from the target*/
      {
         charin(&ch);       /*get the data*/
         if(ch==0x06)       /*see if it is an ACK*/
            return 1;         /*success*/
      }
      else
         if(clock()>starttime+timeout)       /*see if timeout has elapsed*/
            return 0;                              /*failed (due to timeout)*/
```

```
}

int readBlockFromFile (FILE *FptrIn,unsigned long *addr,unsigned char *data)
{
   char Record[128];
   int bytecount;
   unsigned char xsum=0;
   int rectype;
   int packetxsum;
   int i;
   int Index=0;
   unsigned char InChar;

/* read a record from file*/
   InChar=fgetc(FptrIn);
   while((InChar!='\n') && ((char)InChar!=EOF))
   {
      Record[Index++]=InChar;
      InChar=fgetc(FptrIn);
   }
   Record[Index] = '\0';

   if((Record[0]==':')&&(Index!=0))
   {
      /*get the number of data bytes in the record*/
      if(sscanf(Record+1,"%2X",&bytecount)!=1) return 0;
      xsum+=bytecount;
      /*get the block start address*/
      if(sscanf(Record+3,"%4lX",addr)!=1) return 0;
      xsum+=(*addr)/256;
      xsum+=(*addr)%256;
      /*get the record type*/
      if(sscanf(Record+7,"%2X",&rectype)!=1) return 0;
      xsum+=rectype;
      /*get the data from the record*/
      for(i=0;i<bytecount;i++)
      {
         if(sscanf(Record+9+i*2,"%2X",data+i)!=1) return 0;
         xsum+=data[i];
      }
      /*get the checksum from the record*/
      if(sscanf(Record+9+bytecount*2,"%2X",&packetxsum)!=1) return 0;
      xsum+=packetxsum;
      /*validate the checksum*/
      if(xsum!=0)
         return 0;
      /*make sure it's a type of records that we can handle*/
      if(rectype==0)                /*only return type 0 records*/
         return bytecount;
      else
         return 0;                  /*fail on any other type of records*/
   }
   return 0;
}

/*****************************************************************/
/*low-level uart communications routines*/
```

```
/*type of error*/
typedef enum {COMMS_NOERR,             /*no error encountered*/
                    COMMS_OVERRUN,         /*serial comunications overrun*/
                    COMMS_PARITY,          /*serial communications parity error*/
                    COMMS_FRAME,           /*serial communications frame error*/
                    COMMS_BREAK,           /*serial communications break*/
                    COMMS_SENDTIMEOUT, /*communications timeout on xmit*/
                    COMMS_WAITTIMEOUT, /*communications timeout on recieve*/
                    COMMS_RECVTIMEOUT    /*communications timeout on recieve*/
                    } commerrors;


/*line control register bit definitions*/

#define LCRWORDLSB 1
#define LCRWORDMSB 2
#define LCRSTOP       4
#define LCRPAR        8
#define LCRPARSEL 16
#define LCRPARONE 32
#define LCRBREAK    64
#define LCRDIV      128

/*modem control register bit definitions*/
#define MCRDTR      1
#define MCRRTS      2
#define MCROUT1     4
#define MCROUT2     8
#define MCRLOOP 16

/*interupt enable register bit definitions*/
#define INTRECV     1
#define INTXMIT     2
#define INTLSTAT 4
#define INTMSTAT 8

/*line status register bit definitions*/
#define DATAREADY    1
#define OVERRUN       2
#define PARITY        4
#define FRAME         8
#define BREAK        16
#define XMITREADY 32
#define TSRE          64

/*modem status register bit definitions*/
#define DELTACTS      1
#define DELTADSR      2
#define TERI          4
#define DELTARLSD    8
#define CTS          16
#define DSR          32
#define RI           64
#define RLSD        128

/*interupt identifiation register definitions*/
```

```
#define INT_NONE    1
#define INT_MSTAT 0
#define INT_THRE    2
#define INT_RDA     4
#define INT_LSTAT 6

/*uart register offsets from base address*/
#define LCONT (combase+3)
#define TXREG (combase)
#define RXREG (combase)
#define MCONT (combase+4)
#define LSTAT (combase+5)
#define MSTAT (combase+6)

/*record of the previous error*/
commerrors lastcommerr=COMMS_NOERR;

/*hardware IO port base address of the uart*/
unsigned int combase;

/*set up the uart*/
int initcommshardware(int comport,unsigned long int baudrate)
{
   unsigned long divisor;
   unsigned char lsb,msb;
   /*calculate the uart hardware base address*/
   if(comport==1)
      combase=0x3F8;
   else if(comport==2)
      combase=0x2F8;
   else
      combase=comport;
   /*calculate the baud rate divisor values*/
   divisor=(115200L / baudrate);
   msb=divisor/256;
   lsb=divisor & 0x00ff;
   printf(".");                    /*display a progress dot*/
   /*set up the uart to access the baud rate divisor registers*/
   outport(LCONT,LCRDIV);
   printf(".");                    /*display a progress dot*/
   /*write the baud rate divisor values*/
   outport(combase,    lsb);
   printf(".");                    /*display a progress dot*/
   outport(combase+1,msb);
   printf(".");                    /*display a progress dot*/
   /*set up the line parameters*/
   outport(LCONT,0x03);                /*no parity, 1 stop, 8 data*/
   printf(".");                    /*display a progress dot*/
   return 1;
}

/*checks to see if the uart line status is ok*/
unsigned char check_status(int *status)
{
   lastcommerr=COMMS_NOERR;            /*assume no error*/
   *status=inport(LSTAT);                /*get the uart line status*/
   if (*status & OVERRUN)              /*check for overrun error*/
```

```
        lastcommerr=COMMS_OVERRUN;
    if (*status & PARITY)                   /*check for parity error*/
        lastcommerr=COMMS_PARITY;
    if (*status & FRAME)                    /*check for frame error*/
        lastcommerr=COMMS_FRAME;
    if (*status & BREAK)                    /*check for break error*/
        lastcommerr=COMMS_BREAK;
    /*return TRUE if there was no error*/
    return(lastcommerr==COMMS_NOERR);
}


int clearxmit(void)
{
    int status;
    /*see if the uart status is ok*/
    if(check_status(&status))
        /*see if the transmit ready bit is set*/
        if (status & XMITREADY)
            return 1;
    return 0;
}

int dataavail(void)
{
    int status;
    /*see if the uart status is ok*/
    if(check_status(&status))
        /*see if the dataready bit is set*/
        if (status & DATAREADY)
            return 1;
    return 0;
}

/*read the byte from the recieve register of the uart*/
void charin(unsigned char *ch)
{
    *ch=inport(RXREG);
}

/*send the byte to the transmision register of the uart*/
void charout(unsigned char ch)
{
    outport(TXREG,ch);
}
```