

MSP430 程序自升级的实现原理及过程

作者: 乔海坤 微控论坛特约 DC 微控论坛版主

更新: 01

在过去有很多公司或个人对于 MSP430 单片机串口自升级技术都当一个技术机密。而 TI 公布的是汇编语言编写的例子, 在理解上也不便。而在网络上也很难得到公开例程, 这样使得部分 MSP430 用户想实现这个功能时极为艰难。

为此, 微控论坛和大家分享如何利用 MSP430 单片机串口自升级的实现实验。希望能够透过这一个实验能给大家在这方面应用的启蒙和参考。如果你想要优化或增加一些特殊功能, 比如增加密码等功能则需要用户去编写了。如果你对 these 程序做了更好的修改优化, 欢迎到微控论坛来与我们一起分享。

实验流程

[1] 先往 MSP430 单片机写入一个自升级引导程序。

[2] 利用 MC430FUT.EXE 软件下载用户应用程序到 MSP430 单片机。



实验工具: 本次实验采用的是微控的 MC430F14+ 开发板和并口仿真器。单片机型号选用的是 MSP430F147。

基础原理:

首先我们要用 MSP430FET 仿真器在 WE430 上下载一段引导程序到 MCU 上。引导程序需要固定在 FLASH 中一个位置上, 且保证不能被擦除。在本例中, 咱们选择固定在 0Xfa00 后的空间。关于这个固定位置, 用户可以灵活一些。只要后面有空间够放一个引导程序就基本可以了。

引导程序工作程序如下:

```
//*****
```

```
// 描述: 引导程序主程序,此程序首先下载到 MSMP430 单片机中
```

```
void main(void)
```

```
{
```

```
    WDTCTL = WDTPW + WDTHOLD;           // 关 WDT
```

```
    if(ResetVectorValid()==1)           // 判断是否已经下载过应用程序
```

```

{
    Application();           // 执行应用程序
}
Update();                 // 执行升级程序
}
//*****

```

主程序中主要判断 FLASH 中是否已存在着有应用程序，如果有应用程序则执行应用程序，否则执行升级程序。这是基本的引导程序功能。当然要控制是否要执行升级程序的办法有很多，比如可以通过 IO 来控制等等。从引导程序进入应用程序很简单，只需要将 PC 值转移到相应的应用程序起始地址位置即可。

如果需要升级程序，进入升级程序后，首先是初始化串口，在本例中是使用串口 0 来实现与上位机的通讯，在这里我采用了查询的方式，这样做是为了简单，在引导程序中尽量不使用中断，如果使用了串口中断，应用程序中也有相同的串口中断，处理会比较麻烦；只需将接收到的数据全部写入到 flash 中相应的位置。

补充知识

在未实验之前，我想为大这有补充一下关于 MSP430 单片机烧录文件.txt 的知识。下面是一个串口调试的 C 例子和生成烧录.txt 文件内容。至于下程序内容，在这里不必详细多讲了，主要实现串口收发实验。

```

#include <msp430x16x.h>
void main(void)
{
    WDTCTL = WDTPW + WDTHOLD;
    P3SEL |= 0x30;
    ME1 |= UTXE0 + URXE0;
    UCTL0 |= CHAR;
    UTCTL0 |= SSELO;
    UBR00 = 0x03;
    UBR10 = 0x00;
    UMCTL0 = 0x4A;
    UCTL0 &= ~SWRST;
    IE1 |= URXIE0;

    for (;;)
    {
        _BIS_SR(LPM3_bits + GIE);
        while (!(IFG1 & UTXIFG0));
        TXBUF0 = RXBUF0;
    }
}

// UART0 RX ISR will for exit from LPM3 in Mainloop
#pragma vector=UART0RX_VECTOR
__interrupt void usart0_rx (void)
{
    _BIC_SR_IRQ(LPM3_bits);
}

```

经 WE430 编译后，生成的文件内容如下：

```
@8000
```

```

31 40 00 0A B0 12 0C 40 B0 12 5E 40 B2 40 80 5A
20 01 F2 D0 30 00 1B 00 F2 D0 C0 00 04 00 F2 D0
10 00 70 00 F2 D0 10 00 71 00 F2 40 03 00 74 00
C2 43 75 00 F2 40 4A 00 73 00 D2 C3 70 00 F2 D0
40 00 00 00 32 D0 D8 00 C2 93 02 00 FD 37 D2 42
76 00 77 00 F7 3F B1 C0 D0 00 00 00 00 13 30 40
62 40 30 40 66 40 FF 3F
@F9F2
56 40
@F9FE
00 40

```

注意: @后面的 8000 表示地址, 表示下面的内容需要写入从 8000 开始的地址中, @F9F2: 0xf9f2 是应用程序的串口中断 0 的中断向量地址, @F9FE 为应用程序的复位向量地址, q 为结束标志。

而在我们往后的更新程序中在 update() 函数就是将收到的以上内容按相应地址写入到 flash 中。

实验前准备

打开 C:\Program Files\IAR Systems\Embedded Workbench Evaluation 5.0\430\config\目录
 以上地址只相对于我的片子而言, 读者的安装目录可能跟我有差别。由于我本次用的是 F147 芯片, 所以我要找到 Ink430F147.xcl 文件, 然后分别复制出来两个文件。并分别将改名为 Ink430F147_FlashUpdate.xcl 用于引导程序工程使用的, 和 Ink430F147_FlashApp.xcl 用于用户应用程序工程用的。这样做的原因是为了方便一会实验修改之用。
 如果是准备好以上的动作, 咱们就开始吧!

引导程序产生

在 WE430 编译引导程序工程之前, 我们需要修改刚才一个 Ink430F147_FlashUpdate.xcl 文件。

[1]修改目的: 首先我们要修改 Ink430F147_FlashUpdate.xcl 文件的部分数据, 如下面的红色所示。原单片机默认程序开始地址为 8000H, 现在由于我们要放置一段引导程序的需要, 那么必须将引导程序放到一个指定的地址去。由原来的 8000H(32768)修改为 FA00H(64000), FA00H 地址开始就是我们引导程序放置的地址。

```

// -----
// Constant data
// -----
-Z(CONST)DATA16_C,DATA16_ID,DIFUNCT=FA00-FFDF //原为 8000-FFDF
-Z(CONST)DATA20_C,DATA20_ID= FA00-FFDF //原为 8000-FFDF

// -----
// Code
// -----
-Z(CODE)CSTART,ISR_CODE= FA00-FFDF //原为 8000-FFDF
-P(CODE)CODE=FA00-FFDF //原为 8000-FFDF

// -----
// Interrupt vectors
// -----

```

```
-Z(CODE)INTVEC=FFE0-FFFF
-Z(CODE)RESET=FFFE-FFFF
```



```
// -----
// The end
// -----
```

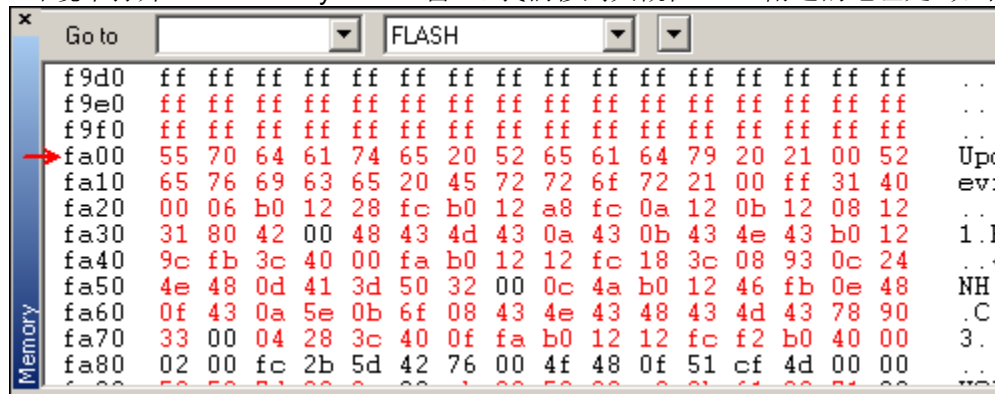
注意:引导程序用的 Ink430F147_FlashUpdate.xcl 文件中的中断向量和复位地址是没有变的,也就是说单片机上电的起始地址和中断向量实质是以引导程序为主的。用户程序才是为被调用的。这是本实验原理的要点,切记!

[2]编译操作:在 WE430 建立引导程序工程,将 FlashUpdate.c 加入工程并进行相关的设置。其中,在 IAR WE430 的项目选择中,设置:Options/Linker/Config/Linker command file/Override default/Ink430F147_FlashUpdate.xcl 文件。

然后进行编译,当编译好后的程序的起始地址就会在 FA00H 地址。那么,我们如何去确认引导程序成功地放置在我们想指定的地方呢?

嗯,问得好。方法如下:

将引导程序用 MSP430FET 仿真器下载到目标板子上,按运行  一下后然后再按  停止下来。我们在 WE430 环境中打开:View/Memory/Flash 窗口,我们移到大概在 FA00 附近的地址处。如下图所示。



从上图可以看出,在红色箭头处是 FA00 地址,对应着代码空间处就是我们上面的引导程序了。这个表明我们定位成功了,呵呵。

MC430FU.c 的源程序如下:

```

/*****
// 关于 MSP430 单片机串口升级实验演示程序----串口升级引导程序
// 原创发布: 微控技术论坛 www.microcontrol.cn
// 发布时间: 2008.12. 编译环境: IAR WE430 V4.1
// 声明:本程序属微控网原创,如需转载或引用请在参考文献中说明 www.microcontrol.cn 微控网
*****/
#include <msp430x14x.h>
#define RESETVECTORADDR_APP 0xF9FE //应用程序复位向量地址

/*****
void InitUart(void);
void EraseFlash(unsigned int addr);
void Application(void);
void Update(void);
unsigned char WriteFlash(unsigned int addr,unsigned char *pdata,
                          unsigned char length);

```

```
unsigned char ReadFlash(unsigned int waddr);

unsigned char ResetVectorValid(void);
unsigned char AsciiToHex(unsigned char cNum);
void uart_send(const unsigned char *data_point);

//*****
// 描述: 引导程序主程序,此程序首先下载到 MSMP430 单片机中
// 输入: 无  返回:无
void main(void)
{
    WDTCTL = WDTPW + WDTHOLD;          //关狗

    if(ResetVectorValid()==1)          // 判断是否已经下载过应用程序
    {
        Application();                 // 执行应用程序
    }
    Update();                           // 执行升级程序
}

//*****
// 描述:应用程序, 将 PC 转移到应用程序的复位向量处
// 输入:无  输出:无
void Application(void)
{
    asm(" mov &0xF9FE, PC;");          // 在 C 中调用汇编指令,实现地址转移
}

//*****
// 描述: 程序更新
// 输入: 无  输出:无
void Update(void)
{
    unsigned int i,j;
    unsigned char RecBuf[50];
    unsigned char WriterBuf[16];
    unsigned char RecCnt=0;
    unsigned char RecTemp=0,RxTemp;
    unsigned long Addr=0;
    unsigned char NumberFlag=0;
    //unsigned int Address;

    InitUart();                          // 初始化串口
    uart_send("Update Ready !");         // 发送确认

/* 在此处根据和上位机软件协议添加擦除 flash 的程序, 由于本人只是验证方法的正确性,
未做此功能 */

/* 下面一段函数为串口数据接收和处理程序 */
    while(1)
    {
```

```
if(IFG1 & URXIFG0)
{
    //MSP430_TX0(RXBUF0);    //将收到的数据发回 PC 机来验证
    RxTemp=RXBUF0;
    RecBuf[RecCnt++]=RxTemp;
    if(RxTemp==0x0A&&RecCnt>0)
    {
        if(RecBuf[0]=='q')    //quit
        {
            //_NOP();
            //return;
            asm(" mov &0xF9FE, PC;");
        }
        else if(RecBuf[0]=='@')    //address
        {
            i=1;
            Addr=0;
            while(1)
            {
                if(RecBuf[i]<'0')
                    break;
                Addr<<=4;
                Addr+=AsciiToHex(RecBuf[i]);
                i++;
            }
            //set address here
        }
        else //number
        {
            RecTemp=0;
            j=0;

            for(i=0; i<50; i++)
            {
                if(RecBuf[i]<'0')
                {
                    if(RecBuf[i]==0x0A)
                    { break;}
                    if(NumberFlag)
                    { NumberFlag=0;
                      WriterBuf[j++]=RecTemp;
                    }
                }
                else if(RecBuf[i]>='0')
                {
                    RecTemp <<= 4;
                    RecTemp += AsciiToHex(RecBuf[i]);
                    NumberFlag=1;
                }
            }
            if(j>0)
            {
                //write data
            }
        }
    }
}
```

```
        WriteFlash(Addr,WriterBuf,j);
        Addr += j;
        j=0;
    }
    NumberFlag=0;
}
RecCnt=0;
RxTemp=0;
}
if(RecCnt>50)
{
    uart_send("Revice Error!");
}
}
}

/*****
// 描述: ASCALL 码转换成字符
// 输入: unsigned char cNum  ASC-II 字符码
// 输出: unsigned char HEX 码
unsigned char AsciiToHex(unsigned char cNum)
{
    if(cNum>='0'&&cNum<='9')
    {
        cNum -= '0';
    }
    else if(cNum>='A'&&cNum<='F')
    {
        cNum -= 'A';
        cNum += 10;
    }
    return cNum;
}

/*****
// 描述:发送一字符串往 PC
// 输入:const unsigned char *data_point 字符串数组
// 输出:无
void uart_send(const unsigned char *data_point)
{
    while(1)
    {
        while(!(IFG1&UTXIFG0)); //query tx ready?
        if(*data_point=='\0')
            break;
        else
        {
            TXBUF0=*data_point;
            data_point++;
        }
    }
}
```

```
}

//*****
// 描述: 检查复位向量地址的内容是否为 0xffff
// 输入: 无
// 输出: 如果复位向量地址内容不为 0xffff,则返回 1.
unsigned char ResetVectorValid(void)
{
    if(ReadFlash(RESETVECTORADDR_APP)==0xff&&
        ReadFlash(RESETVECTORADDR_APP+1)==0xff)
//如果应用程序的复位向量处的内容为 0xffff,表示没应用程序
    {
        return 0;
    }
    return 1;
}

//*****
// 描述: UART0 初始化,9600 位波特率/秒,8 位字符,1 位停止位,不校验.
// 输入: 无 输出: 无
// 说明一下,波特率 9600,时钟源选择为 ACLK.误码率可能会大些.如果用户需要可以将选择
// 其他高频时钟源.这点网友应该要自己修改.选择高频时钟会使波特率的误码率减低.
void InitUart(void)
{
    P3SEL |= BIT4 + BIT5;           // P3.4,5 = USART0 TXD/RXD
    ME1 |= UTXE0 + URXE0;          // 使能 USART0 TXD/RXD
    UCTL0 |= CHAR;                 // 8 位字符
    UTCTL0 |= SSELO;              // UCLK = ACLK
    UBR00 = 0x03;                 // 32k/9600 - 3.41
    UBR10 = 0x00;
    UMCTL0 = 0x4A;                // 波特率调整器设置
    UCTL0 &= ~SWRST;              // 初始化 USART 状态机
}

//*****
// 描述: FLASH 擦除操作
// 输入: 16 位 FLASH 地址
// 输出: 无
void EraseFlash(unsigned int waddr)
{
    _BIC_SR(GIE);                 // 关闭总中断

    FCTL2 = FWKEY + FSSEL0+FN1;    // 选择 DC0 作为 LFASH 操作时钟源,MCLK/2
    FCTL3 = FWKEY;
    FCTL1 = FWKEY + ERASE;         // 擦除操作
    *(unsigned char*)waddr=0;      // 虚拟的擦除段操作
    while(FCTL3 & BUSY);
    FCTL3=FWKEY+LOCK;

    _BIS_SR(GIE);                 // 再次开总中断使能
}
}
```



```

//*****
// 描述: FLASH 写操作
// 输入: unsigned int addr 16 位 FLASH 地址, unsigned char *pdata 数据指针
// 输入: unsigned char length 数据长度
// 输出: unsigned char 错误标志
unsigned char WriteFlash(unsigned int addr,unsigned char *pdata,
                        unsigned char length)
{
    unsigned char ErrorFlag = 0;
    unsigned char i;

    while(FCTL3 & BUSY);

    _BIC_SR(GIE);

    FCTL2 = FWKEY + FSSEL0+FN1;
    FCTL3 = FWKEY; // 清除锁
    FCTL1 = FWKEY + WRT; // 设置 WRT 位为写操作
    for(i=0;i<length;i++)
    {
        *(unsigned char*)addr=*pdata; // 写一个字节
        if(ReadFlash(addr)!=*pdata) // 验证,写比较.正确或错误
        {
            ErrorFlag = 1; // 设置错误标志
        }
        addr++;pdata++;
    }
    FCTL1=FWKEY;
    FCTL3=FWKEY+LOCK;

    _BIS_SR(GIE);

    return ErrorFlag;
}

//*****
// 描述: 读 FLASH 操作
// 输入: unsigned int waddr 16 位地址
// 输出: unsigned char 返回一个字节数据
unsigned char ReadFlash(unsigned int waddr)
{
    unsigned char value;
    while(FCTL3 & BUSY);
    value = *(unsigned char*)waddr;
    return value;
}

//*****
// 描述: 中断向量列表
#pragma vector=0
__interrupt void intec_0(void)

```

```
{
    asm(" br &0xF9E0;");
}

#pragma vector=2
__interrupt void intec_1(void)
{
    asm(" br &0xF9E2;");
}

#pragma vector=4
__interrupt void intec_2(void)
{
    asm(" br &0xF9E4;");
}

#pragma vector=6
__interrupt void intec_3(void)
{
    asm(" br &0xF9E6;");
}

#pragma vector=8
__interrupt void intec_4(void)
{
    asm(" br &0xF9E8;");
}

#pragma vector=10
__interrupt void intec_5(void)
{
    asm(" br &0xF9EA;");
}

#pragma vector=12
__interrupt void intec_6(void)
{
    asm(" br &0xF9EC;");
}

#pragma vector=14
__interrupt void intec_7(void)
{
    asm(" br &0xF9EE;");
}

#pragma vector=16
__interrupt void intec_8(void)
{
    asm(" br &0xF9F0;");
}
```

```
#pragma vector=18
__interrupt void intec_9(void)
{
    asm(" br &0xF9F2;");
}

#pragma vector=20
__interrupt void intec_10(void)
{
    asm(" br &0xF9F4;");
}

#pragma vector=22
__interrupt void intec_11(void)
{
    asm(" br &0xF9F6;");
}

#pragma vector=24
__interrupt void intec_12(void)
{
    asm(" br &0xF9F8;");
}

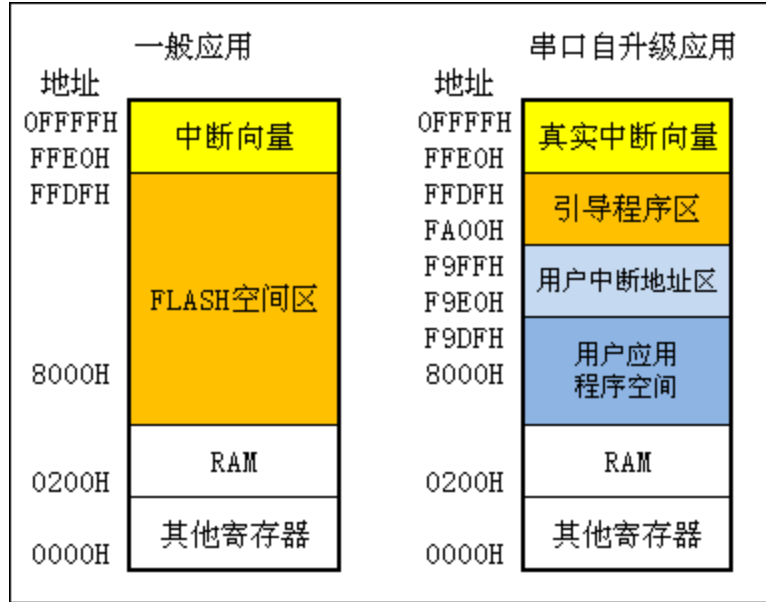
#pragma vector=26
__interrupt void intec_13(void)
{
    asm(" br &0xF9FA;");
}

#pragma vector=28
__interrupt void intec_14(void)
{
    asm(" br &0xF9FC;");
}
```

下载用户应用程序

同样，要修改 Ink430F147_FlashApp.xcl 文件，打开 IAR 的根目录，C:\Program Files\IAR Systems\Embedded Workbench Evaluation 5.0\430\config\Ink430F147_FlashApp.xcl

[1]修改目的:此次修改目的有别于上面的引导程序修改。原为 8000-FFDF 地址现在修改为 8000-F9DF 地址。从地址数上看程序空间变小了，这样可以保证用户应用程序空间不和引导程序空间相冲突。换句话说，将原来的 8000-FFDF 空间分成两部分，一小部分空间用于放置引导程序。一大部分空间用于分给用户应用程序空间使用。



另外，重要一点就是用户的应用程序的中断向量是虚拟出来的，这个地址是由用户修改 `Ink430F147_FlashApp.xcl` 文件分配的(见下面修改部分)。用户的应用程序的中断向量地址是虚拟的，再也不是由原来的 `FFE0-FFFF` 了。而被咱们修改为 `F9E0-F9FF`。当单片机真正接受到中断时，单片机中断先跳去真正的中断向量中，然后再被转移到用户应用程序的虚拟中断向量地址中。修改 `Ink430F147_FlashApp.xcl` 的内容如下：

```
// -----
// Constant data
// -----
-Z(CONST)DATA16_C,DATA16_ID,DIFUNCT=8000-F9DF //原为 8000-FFDF
-Z(CONST)DATA20_C,DATA20_ID=8000-F9DF
// -----
// Code
// -----
-Z(CODE)CSTART,ISR_CODE=8000-F9DF // 原为 8000-FFDF
-P(CODE)CODE=8000-F9DF

// -----
// Interrupt vectors
// -----

-Z(CODE)INTVEC=F9E0-F9FF // 原为 FFE0-FFFF
-Z(CODE)RESET=F9FE-F9FF
```

[2]编译操作：在 WE430 创建用户应用程序工程，并进行相关的选项设置。其中，在 IAR WE430 的工程选项中：

[2-1] 设置：Options/Linker/Config/Linker command file/Override default/`Ink430F147_FlashApp.xcl` 文件。

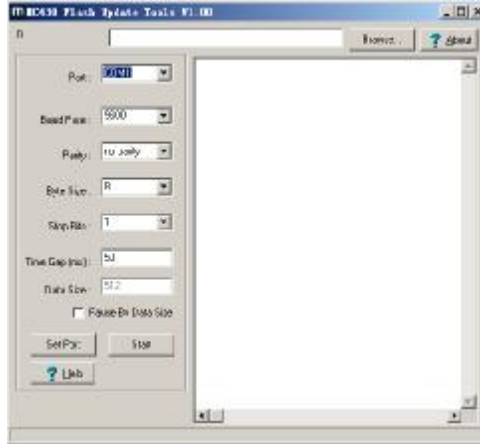
[2-2] 设置：Options/Linker/Output/将输出文件选择为 Other，使其输出为 .txt 文件。

[2-3] 编译：进行工程编译，并自动生成了 .txt 烧录文件在工程的目录中。 .txt 文件是生成在工程文件夹的 `Debug\Exe\` 位置。

[3]升级用户应用程序

打开微控提供的 **MC430FUT.EXE** 上位机软件，将目标板的 UART0 接口(DB9 端子)通过串口延长线连接到电脑的串口。此次例程中升级实验用的是 UART0 口，相信很多朋友能够模仿得到此实验。

关于上位软件 MC430FUT.EXE，基于实验的考虑软件在设计上做得比较简单。电脑串口只做数据输出，并未读回或检验等等动作。所以你必须保证你的目标板子串口硬件是能正常使用的才能实验。



将引导程序下载后，应使 IAR WE430 退出调试模式。接下来是下载应用程序。

在 MC430FUT 先选择要下载的 .txt 烧录文件，然后选择你 PC 相连接的 COM 口。然后按 **Start** 开始下载，如果目标板子 UART0 硬件是正常的话程序会顺利下载到板子中。下载完成后，板子程序就可以执行。

本次实验我们所提供的文件如下。如需要这些文档，请到微控论坛上下载。

- FlashUpdate
- lnk430F147_FlashApp.xcl
- lnk430F147_FlashUpdate.xcl
- MC430FUT.exe
- MSP430_FlashUpdate.pdf

更新日期	描述
2008/12/8	首发版