前 言

本手册简单介绍了利用 MSP430 Flash 系列芯片内部的启动程序载入器(Bootstrap),通过MSP430 的 P1.1 和 P2.2 进行程序下载的方法.

利用这种方法自制的编程器,可以脱机操作、设定编程次数,方便现场操作.

另外,当通过 JTAG 口将程序加密后,还可通过此方法进行程序下载,但不能读出.

本手册是根据 TI 光盘中的 SLAA096.pdf 文件翻译过来(现有最新 SLAA096A).手册中所提及的文档在 TI 的光盘中均可找到,读者若需要相关资料请与我们联系.

在翻译过程中得到本公司诸位同事的大力支持,在此一并表示感谢.本手册翻译的目的主要是方便读者了解、利用 MSP430,由于译者的学识水平有限,翻译中难免有错误和不妥之处,恳请读者批评指正.

译者

2001 年 6 月

启动程序载入器(Bootstrap)在 MSP430F11X-硬件和软件中的应用建议

摘要

由带 Flash Memory 的 MSP430 引申出来的启动程序载入器(Bootstrap)（也叫引导加载）允许在样品、成品、甚至在程序区访问其内置存储器。可以在 Flash Memory(electrically-erasable and programmable memory)中下载或修改代码，在 Flash Memory 或 RAM 中存储标准数据或其它系统相关数据。

本应用手册描述了通过计算机串口（RS-232），访问 MSP430F11X-芯片（F1121，F1101，F112 和 F110 等芯片）的启动程序载入器(Bootstrap)实现程序下载的简单的、低成本的软硬件解决方法。本说明提供的及 C-源代码软件程序允许根据特定的要求来修改。

目录

图形列表

表格列表

1 绪论

由带Flash Memory(electrically-erasable and programmable memory)的MSP430引申出来的启动程序载入器(Bootstrap)允许在几秒钟之内修改数据和程序代码。不再需要使用紫外线（UV）擦除EPROM的时间。控制单元中的部分——启动程序载入器(Bootstrap)，在带Flash Memory的MSP430芯片中实现访问并使用这些特征。报告中阐述了用硬件和软件解决从PC到提供控制访问加载器及简单使用启动程序载入器(Bootstrap)的方法。

2 启动程序载入器(Bootstrap)的基本要素

本章叙述了启动程序载入器(Bootstrap)的基本要素和使用方法。详细请参见：*MSP430F11x (SLAS256)* 或者*MSP430F11x1 (SLAS241)*文档和*MSP430 Bootstrap Loader in the MSP430F1121(SLAA089)*应用手册。

启动程序载入器(Bootstrap)是一种编程方法，它允许通过串行连接和MSP430通讯，甚至在Flash Memory 被完全擦除时。不要同某些数字信号处理器（DSP）的从外部存储器向DSP内部存储器自动装载程序代码或（数据）相混淆。这些编程方法也常被化归为启动程序载入器(Bootstrap)。

2.1 调用启动程序载入器(Bootstrap)

MSP430的启动程序载入器(Bootstrap)不会自动启动——需要对RST/NMI和TEST引脚设置特殊的顺序。当TEST脚下降而RST/NMI为上升沿（图1），复位向量位于Memory地址0FFFEh时，用户程序开始。图2显示了启动启动程序载入器(Bootstrap)所需的顺序。启动顺序的详细资料请见上面所提及文档。

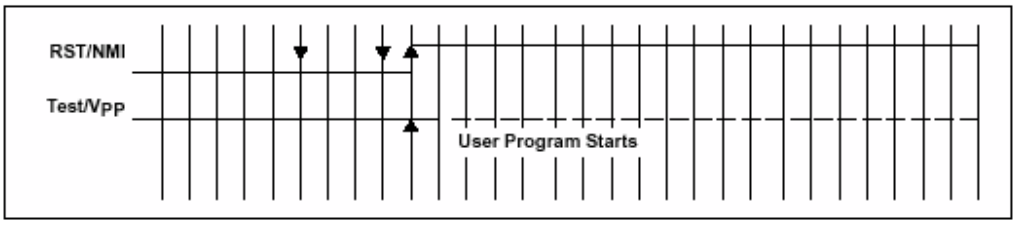


Figure 1. RST/NMI and TEST Sequence to Start User Program

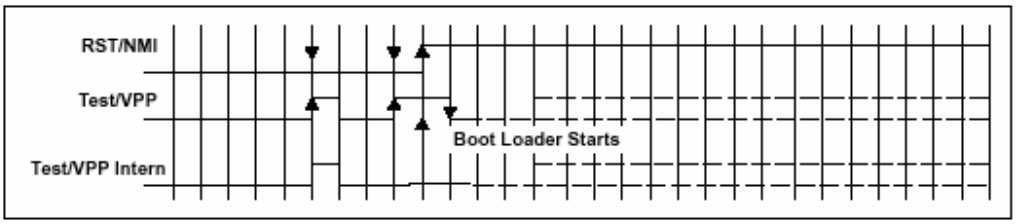


Figure 2. RST/NMI and TEST Sequence to Start Bootstrap Loader

2.2 访问启动程序载入器(Bootstrap)

使用TEST和RST/NMI脚调用启动程序载入器(Bootstrap)后，通讯可以用一个标准的异步串口协议确定。用MSP430的P1.1口（BSLTX）传输数据，P2.2口（BSLRX）接收数据。UART设置见表1。

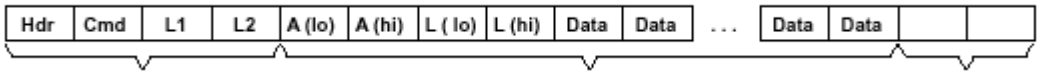
表1.启动程序载入器(Bootstrap)的UART设置

设置	值
波特率	9600波特
数据位	8（二进制）
寄偶位	偶
停止位	1

用于与启动程序载入器(Bootstrap) 的通讯协议来源于一个更复杂的协议；这增加了一些开销。

首先PC机必须发送一个同步字节。如果启动程序载入器(Bootstrap)正确接收该字符，将返回一个确认字节。

同步通讯成功以后，PC机发送一帧包含命令和数据的字符。该帧（见图3）能被分割成报头段、数据段及检验和。数据段总是包含至少4个字节的数据：通常为一个起始地址和长度，但是一些命令可以忽略其内容或不同的意义。在字（2字节）的情况下，字节的顺序总是从低字节到高字节。



报头段 数据段 检验和

Hdr：0X80（始终） **A**：地址（其始）

Cmd：（被请求函数代码） **L**：长度

L1，L2：数据段字节个数 （**lo**）：低字节

（**hi**）：高字节

图3.发送启动程序载入器(Bootstrap)的格式

处理完该格式后，启动程序载入器(Bootstrap)也返回一个确认信号，如果该格式不正确则返回拒绝信号或命令失败信号。如果该命令不允许则发出命令失败信号。

2.3 启动程序载入器(Bootstrap)功能

启动程序载入器(Bootstrap)具有保护和非保护功能。要实现保护功能则必须将程序的所有中断向量表（位于地址范围0FFE0h到0FFFFh之间）发送到启动程序载入器(Bootstrap)。

本节给出了能用到的命令的提要。这些命令及其使用方法在本应用手册中的另一个文档和软件部分有更详细的说明。

2.3.1非保护功能

- 设置口令，能实现保护功能
- 块擦除，完全擦除Flash Memory。访问保护功能的口令为32位（十六进制）。

2.3.2保护功能

- 发送模块，写数据到MSP430的存储器
- 接收模块，从MSP430存储器中读数据
- 擦除段
- 装载程序计数器，开始执行。

表2 启动程序载入器(Bootstrap)功能一览

FUNCTION	CMD	L1=L2	ADDRESS	LENGTH	DATA
Set password	10h	24h	xx	xx	Password
Mass erase	18h	04h	Flash	0A506h	–
Transmit byte	12h	n+4	Start	n	Data (n byte)
Receive byte	14h	04h	Start	n	–
Erase	16h	04h	Segment	0A502h	–
Load PC	1Ah	04h	Start	xx	–

3 硬件描述

在本应用手册中低成本硬件的提出（图4），主要由：一个低压差稳压器(low-dropout voltage regulator)、几个反向器和一个运算放大器，以及一些电阻、电容、二极管等组成。完整元件列在本节后面。

下面详细介绍其功能模块。

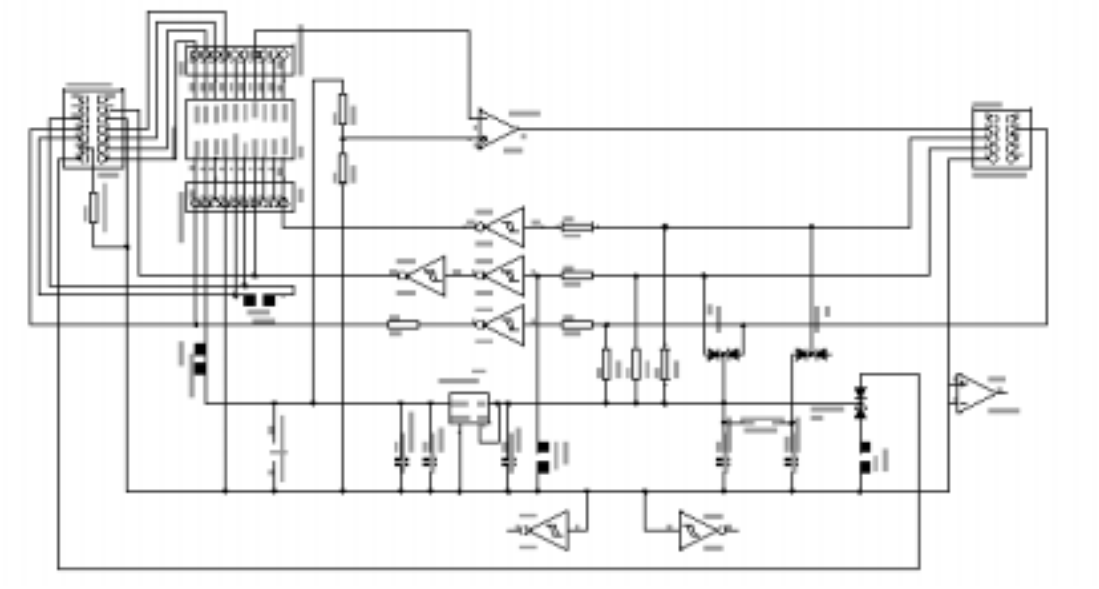


图4 启动程序载入器(Bootstrap)接口示意图

3.1 供电方式

启动程序载入器(Bootstrap)硬件可以通过RS-232接口供电。RS-232信号DTR（串口连接器第7脚）和RST（串口连接器第4脚）通常送给负载电容C1一正向电压，及低压差稳压器(low-dropout voltage regulator)IC1(Texas Instruments TPS76030 or LP2980-3.0, or equivalent 3-V low-dropout regulator).供电。

使用一个相当大的电容器，可以提取一个向Flash Memory编程所需要的高于串口所能提供的短周期电流。

也可以连接一个外部供电电压（Vext，CON4），或者通过JTAG连接器的第2脚向硬件供电。二极管防止反向电流。

3.2 串口界面

表3为用于与启动程序载入器(Bootstrap)通讯的信号。所提及名称的引脚功能是由PC引出。例如，PC经过RxD脚接收数据，反之启动程序载入器(Bootstrap)需要驱动该信号。带有后缀-D的连接器与PC连接器组合的引脚分配显示（由外部看来）。

表3.串口信号和引脚分配

PIN NAME	FULL NAME	SERIAL	3-PIN SUB-D (PC)
RxD	Receive data	3	2
TxD	Transmit data	5	3
DTR	Data terminal ready	7	4
RTS	Request to send	4	7
GND	Ground	9	5

3.2.1 电平移位

简单的、带有施密特触发器的CMOS反向器（IC2）被用于转换RS-232电平（见表4）为3-V CMOS电平

表4.RS-232电平

LOGIC LEVEL	RS-232 LEVEL	RS-232 VOLTAGE LEVEL
1	Mark	–3 V – 15 V
0	Space	3 V – 15 V

依靠过电压保护所选设备，其剩余电压被送到VCC（TI 74HC14 ）或GND（TI 74AHC14）。如果保护二极管导向VCC，则电压稳压器需要补偿过电压。这样74AHC14芯片导向地面（GND）（推荐），而74HC14也正常工作。

为避免过剩电压浪费和保护二极管的损坏，串联电阻器（R1，R2和R3）用于限制输入电流。

运算放大器（IC3）用于产生3-V CMOS电平之外的RS-232电平。正向输入电平设置为VCC/2（通常为1.5-V）。如果负向输入电平超过该电平，输出电平将被导向运算放大器的负向端（标志）。如果电平低于VCC/2,则输出将被导向正向范围（隔离）。

运算放大器的正向供电与稳压器的输入相同。隔离电容（C10）用来产生负向供电电压。该电容由启动程序载入器(Bootstrap)硬件（串口连接器的第5脚）的接收信号控制。

在异步串口通讯期间，停止位和起始位相结合为同步发送器和接收器。当一个数据字节发送后，停止位驱动发送行为已定义的状态，通常为逻辑位1，或在RS-232中标志。这意味着当没有传送数据和电容被控制时，传送行电压为负向。二极管用于防止在传送期间电容放电。

因为用于启动程序载入器(Bootstrap)的通信协议是半双工的（一次一个发送端），所以接收行的电压是负向、稳定的，且能被用于提供运算放大器所需电压。

3.2.2 RST/NMI和TEST引脚的控制

MSP430的RST/NMI和TEST引脚分别由DTR和RTS信号控制。正如已提及的，这两个信号通常必须传送一个正向电压给启动程序载入器(Bootstrap)的硬件。它们也被用于控制RST/NMI和TEST引脚的电平。

RST/NMI和TEST引脚的电压在常规操作期间，分别为逻辑1和逻辑0。为完成这些电平和使用相应的RS-232信号作为供电行，需要对RST/NMI引脚使用两个反向器，TEST引脚使用一个反向器。

允许通过控制RS-232的二极管来防止电容C1放电（RST和DTR）。

3.3 元件清单

表6.附加连接器/衰减器

NAME	DESCRIPTIVE NAME	PAD LAYOUT	COMMENT
CON1	I-measure	SMD 0805	For current measurements
CON2	Quartz	SMD 0805	To connect an optional quartz
CON3	RESET	SMD 0805	To connect an optional reset button
CON4	Vext	SMD 0805	To connect an optional external-power supply

表5.元件清单

PART	VALUE/ PART NUMBER	PACKAGE	COMMENT
C1	33 µF, 16 V	SMD 7243	
C2	100 nF	SMD 0805	
C3	2.2 µF, 6.3 V	SMD 1206	
C4	100 nF	SMD 0805	
C10	33 µF, 16 V	SMD 7243	
D1	BAV70	SOT23	High-speed double diode
D2	BAV70	SOT23	High-speed double diode
D3	BAR43C	SOT23	Schottky
IC0	MSP430F11x	SO20	TI
IC1	TPS76030	SOT23/5	TI
IC2	74AHC14	SO14	TI (alternate: 74HC14 – see text)
IC3	LM358D	SO8	
R1	330 kΩ	SMD 0603	
R2	330 kΩ	SMD 0603	
R3	330 kΩ	SMD 0603	
R4	680 kΩ	SMD 0603	
R5	680 kΩ	SMD 0603	
R6	680 kΩ	SMD 0603	
R7	330 kΩ	SMD 0603	
R8	330 kΩ	SMD 0603	
R9	300 kΩ	SMD 0603	
R10	220 kΩ	SMD 0603	Assemble only for use with programming adapter: forces Vext to 3 V.
JP1	PINHD-1X10	2X05	Access to pins 11–20 of MSP430F11x
JP2	PINHD-1X10	2X05	Access to pins 1–10 of MSP430F11x
JTAG	PINHD-2X7	2X07	
SERIAL	PINHD-2x5	2X05	RS–232 connector (see Table 3)

4 软件描述

本节说明了使用PC的RS-232接口访问启动程序载入器(Bootstrap)的基本顺序。这里给出的代码为使用32-位Windows™ API，在Microsoft Visual C++™5.0环境下调用 （也可工作在Microsoft Visual C++™ 6.0版本下）。该代码最初写于16-位Windows，主要由替换函数名扩展为32-位Windows(Win32™, Windows 95/98™, and Windows NT™)。某些Win32的特征不能继续被移植。用商业库可以取代Windows API界面。

一个详细描述在Win32中的串口通信（*Serial Communications in Win32*）文件，在线网址：http://msdn.microsoft.com/library/techart/msdn_serial.htm 或参照微软开发网络库的VC++在线文件。

详细描述API功能的使用文件请参照Windows软件开发包（SDK）。
附录A有完整的清单。

4.1 全局变量

下面的全局变量使用在整个代码事例中。在Windows SDK文档可以找到有关DCB，COMSTAT和COMMTIMEOUTS类型定义。

```
HANDLE      hComPort;              /* COM-port handle */
DCB         comDCB;                /* COM-port control settings */
COMSTAT comState;                  /* COM-port status information */
COMMTIMEOUTS orgTimeouts;      /* Original COM-port time-out */
```

4.2 RS-232口初始化

程序访问串口RS-RS232时需要请求要使用口（通常为COM1，COM2）的句柄。下面代码可用于这类请求：

```
/* Size of internal WINDOWS–Comm buffer: */
```

```
#define QUEUE_SIZE 512
. . .
char* lpszDevice= ”COM1” /* For example ... *./
. . .
hComPort= CreateFile(lpszDevice, GENERIC_READ | GENERIC_WRITE,
0, 0, OPEN_EXISTING, 0, 0);
if (hComPort == INVALID_HANDLE_VALUE)
{ . . . /* Error! */
}
if (SetupComm(hComPort, QUEUE_SIZE, QUEUE_SIZE) == 0)
{ . . . /* Error! */
}
```

Microsoft, Windows, Win32, Windows NT, Windows CE, and Visual C++ are trademarks of Microsoft Corporation.

大家知道复用输入/输出（I/O）口可以在Win32下完成。意味着系统可以立即返回回调，当操作完成后发出信号通知回调，甚至在I/O操作没有完成时。当涉及到移植性时，复用操作不是一个很好的选择，因为大多数操作系统不支持它，因此它没有用在此程序中，当使用回调函数CreateFile时，相映的变量参数置为零。

收到一个有效的句柄后，通讯口的设置需要被定义和赋值（见表1）。

首先，获得初始设置：

```
if (!GetCommState(hComPort, &comDCB))
{ . . . /* Error! */
}
```

然后，可以修改。启动程序载入器(Bootstrap)的通讯的最重要的设置显示在下面的程序段：

```
comDCB.BaudRate = CBR_9600;          /* Startup Baudrate: 9,6kBaud */
comDCB.ByteSize = 8;
comDCB.Parity = EVENPARITY;
comDCB.StopBits = ONESTOPBIT;
comDCB.fBinary = TRUE;               /* Enable Binary Transmission */
comDCB.fParity = TRUE;               /* Enable Parity Check */
comDCB.fRtsControl = RTS_CONTROL_ENABLE; /* For power supply and TEST */
                                         /* pin control */
comDCB.fDtrControl = DTR_CONTROL_ENABLE;   /* For power supply and RST/NMI */
                                               /* pin control */
. . .
```

最后，修改值赋给端口：

```
if (!SetCommState(hComPort, &comDCB))
{ . . . /* Error! */
}
```

一个Win32的应用在使用通讯口时总是要设置通讯暂停时间；否则，使用默认值或上一级应用的余值。该应用不需要暂停功能。因此，一旦最初设置被保存，暂停完全丧失，可在程序的末尾恢复：

```
/* Save original time-out values: */
GetCommTimeouts(hComPort, &orgTimeouts);
```

```
/* Set Windows time-out values (disable built-in time-outs): */
COMMTIMEOUTS timeouts;
timeouts.ReadIntervalTimeout= 0;
timeouts.ReadTotalTimeoutMultiplier= 0;
timeouts.ReadTotalTimeoutConstant= 0;
timeouts.WriteTotalTimeoutMultiplier= 0;
timeouts.WriteTotalTimeoutConstant= 0;
if (!SetCommTimeouts(hComPort, &timeouts))
{ . . . /* Error! */
}
```

利用清除发送和接收缓冲器来完成初始化的顺序：

```
PurgeComm(hComPort, PURGE_TXCLEAR | PURGE_TXABORT);
PurgeComm(hComPort, PURGE_RXCLEAR | PURGE_RXABORT);
```

一个完整的初始化程序可以在Serial Communication Implementation File（见附录A）中找到：

```
int comInit(LPCSTR lpszDevice, DWORD aTimeout, int aProlongFactor)
```

4.3 调用启动程序载入器(Bootstrap)

RST/NMI和TEST的电平必须按照2.1节的指示来调用启动程序载入器(Bootstrap)。

下面的子程序可以用来控制相应的RS-232队列RST和DTR：

```
void SetRSTpin(BOOL level)
/* Controls RST/NMI pin (0: GND; 1: VCC) */
{
if (level == TRUE)
comDCB.fDtrControl = DTR_CONTROL_ENABLE;
else
comDCB.fDtrControl = DTR_CONTROL_DISABLE;
SetCommState(hComPort, &comDCB);
} /* SetRSTpin */
void SetTESTpin(BOOL level)
/* Controls TEST pin (0: VCC; 1: GND) */
{
if (level == TRUE)
comDCB.fRtsControl = RTS_CONTROL_ENABLE;
else
comDCB.fRtsControl = RTS_CONTROL_DISABLE;
SetCommState(hComPort, &comDCB);
} /* SetTESTpin */
```

调用函数SetRSTpin使RST/NMI 接地为0，反之调用SetTESTpin使TEST供电电源VCC为0。其不同值起决于这些引脚所使用的反向器的个数（见第3章）。

下面的子程序允许使用上面显示的函数来对MSP430启动程序载入器(Bootstrap)的硬件复位。首先，需要控制电容C1对电路板供电。接着，RST/NMI和TEST脚可以根据需要来触发。然后，可以复位MSP430及启动应用程序(invokeBSL=FALSE)。或者用invokeBSL=TRUE调用启动程序载入器(Bootstrap)：

```
void bslReset(BOOL invokeBSL)
{
/* To charge capacitor on boot loader hardware: */
SetRSTpin(1);
SetTESTpin(1);
delay(250);
SetRSTpin(0); /* RST pin: GND */
if (invokeBSL)
{
SetTESTpin(1); /* TEST pin: GND */
SetTESTpin(0); /* TEST pin: Vcc */
SetTESTpin(1); /* TEST pin: GND */
SetTESTpin(0); /* TEST pin: Vcc */
SetRSTpin (1); /* RST pin: Vcc */
SetTESTpin(1); /* TEST pin: GND */
}
else
{
SetRSTpin(1); /* RST pin: Vcc */
}
/* Give MSP430's oscillator time to stabilize: */
delay(250);
/* Clear buffers: */
PurgeComm(hComPort, PURGE_TXCLEAR); PurgeComm(hComPort, PURGE_RXCLEAR);
} /* bslReset */
```

到目前为止，可以获得访问RS-232接口的PC界面和使用当前函数调用启动程序载入器(Bootstrap)。

我们现在准备访问启动程序载入器(Bootstrap)。

4.4 访问启动程序载入器(Bootstrap)

本节描述了访问启动程序载入器(Bootstrap)的基本子程序。表7给出了当前函数的简要说明，在附录A中有详细的清单。常数定义位于清单前部。

表7. 启动程序载入器(Bootstrap)访问函数

SECTIONS	FUNCTIONS
Bootstrap-loader communication header file, bootstrap-loader communication implementation file	bslReset, bslSync, bslTxRx
Serial-communication header file, serial-communication implementation file	comInit, comTxRx, comDone

4.4.1 同步

在调用一个函数的每种格式前PC和MSP430需要同步。因此，下面的参数（0x80）必须由PC发送到MSP430：

```
#define BSL_SYNC 0x80
```

这样，该参数被设为一变量且被函数WriteFile当作通讯口句柄的第一个参数回调。要传送字节的数量必须被指定（这里指定为1）。在这种情况下最后两个参数是不重要的、可以忽略的。

```
/* Send synchronization byte: */
ch = BSL_SYNC;
WriteFile(hComPort, &ch, 1, &NrTx, NULL);
```

如果启动程序载入器(Bootstrap)正确接收该参数，将返回DATA_ACK（0X90）；否则，将返回一个未知参数，因为加载器需要该同步字符去产生同步（例如，串口通讯）。没有参数被传回也是可能的，比如，当MSP430启动程序载入器(Bootstrap)函数没有连接时。下面子程序是用来检测给定字符数在给定时间内收到。用函数ClearCommError来接收通讯口的实际状态。在COMSTAT结构中用cbInQue保存接收字节的数量：

```
int comWaitForData(int count, DWORD timeout)
{
DWORD errors;
int rxCount= 0;
DWORD startTime= GetTickCount();
do
{
ClearCommError(hComPort, &errors, &comState);
} while (((rxCount= comState.cbInQue) < count) &&
(calcTimeout(startTime) <= timeout));
return(rxCount);
}
```

该子程序用于同步函数中等待某个字符。如果字符被接收，可以使用函数ReadFile查询，可在下面的程序段中看到。当接收字符为DATA_ACK时，同步成功。

```
/* Wait for 1 byte; time-out: 100ms */
rxCount= comWaitForData(1, 100);
if (rxCount > 0)
{
ReadFile(hComPort, &ch, 1, &NrRx, NULL);
if (ch == DATA_ACK)
{ return(ERR_NONE); } /* Sync. successful */
}
```

程序 int bslSync()实现预先描述的函数。

4.4.2 传送格式

下面的函数用于发送和接收的格式：

```
int comTxRx(BYTE cmd, BYTE data[], BYTE length)
```

该函数实现TI标准串口协议的功能。仅需一个函数的子集和启动程序载入器(Bootstrap)通讯的支持。使用Win32功能的程序已经描述过。

函数：

```
int bslTxRx(BYTE cmd, WORD addr, WORD len, BYTE blkout[], BYTE blkin[])
```

结合同步和传输的格式，也准备了启动程序载入器(Bootstrap)所需的格式。一个重要的条件是发送到（例如，使用RXBLK命令）启动程序载入器(Bootstrap)或加载器请求（使用RXBLK命令）的字节的个数应该总是相等。

4.5调用启动程序载入器(Bootstrap)函数

使用当前函数bslTxRx调用启动程序载入器(Bootstrap)功能是相当简单的。在函数调用期间根据需要设置参数（见4.7部分例子）。

可用到命令的相应常数定义可以在启动程序载入器(Bootstrap)的通讯头文件中找到（见附录A）。

4.6 释放RS-232口

程序结束后，RS-232口必须用函数：

CloseHandle(hComPort)

释放。否则，其它应用程序不能使用，因为串行通讯口通常一次只允许一个程序正确访问。

例程

int comDone()

提供了一个更完善的解决方法。一直等到所有等待数据传送完，清空所有缓冲器，恢复初始暂停时间设置。

注意：通常切断应用电源来关闭串行通讯口，因为经过串口的相应的串口控制线已丧失。

4.7 一个完整应用事例

本节介绍了用TI-TXT格式向MSP430的flash memory编程的一个小的应用事例开发。完整的示范程序代码列在附录A中的Bootstrap Loader Demonstration Program中。本节使用的变量定义也在那里。

首先，通讯口必须打开。COM-口的名字可以根据需要而改变，或者从命令行获得。

```
if (comInit(”COM1”, DEFAULT_TIMEOUT, 4) != 0)
{ . . . /* Error! */
}
```

然后，启动程序载入器(Bootstrap)被调用:

```
bslReset(1);
```

接下来，使用块擦除命令将flash memory全部擦除：

```
if ((error= bslTxRx(BSL_MERAS, /* Command: Mass Erase */
0xff00, /* Any address within flash memory. */
0xa506, /* Required setting for mass erase! */
NULL, blkin)) != 0)
{ . . . /* Error! */
}
```

当flash memory被擦除，访问启动程序载入器(Bootstrap)保护功能的口令复位。所有寄存器单元被设为0FFh。可以发送相应口令激活保护功能。

```
/* Fill blkout with 0xff */
for (i= 0; i < 0x20; i++)
{
blkout[i]= 0xff;
}
if ((error= bslTxRx(BSL_TXPWORD, /* Command: transmit password */
0xffe0, /* Address of interrupt vectors */
0x0020, /* Number of bytes */
blkout, blkin)) != 0)
{ . . . /* Error! */
/* Special case here: 7(ERR_RX_NAK): Password not accepted! */
}
```

最后，分析TI-TXT文件格式，将数据写入到flash memory并校验。

这里有一个单独的子程序清单，它能够从主程序分析文件、编程、校验FLASH内容中

调回：

```
/* Program: */
if ((error= programFlash(”TEST.TXT”, ACTION_PROGRAM)) != 0)
{ . . . /* Error! */
}
/* Verify: */
if ((error= programFlash(”TEST.TXT”, ACTION_VERIFY)) != 0)
{ . . . /* Error! */
}
```

程序programFlash对给定文件名进行简单分析（文件名来源于命令行），提取数据填充缓冲器，当缓冲器几乎满时调用其它子程序。

如果该数据需要编程，可以使用传送块命令(BSL_TXBLK)发送到启动程序载入器(Bootstrap)。

```
error= bslTxRx(BSL_TXBLK, addr, len, blkout, blkin);
```

利用从启动程序载入器(Bootstrap)（读模块BSL_RXBLK）读出的数据与发送缓冲器中的内容作比较来校验：

```
error= bslTxRx(BSL_RXBLK, addr, len, NULL, blkin);
if (error != 0)
{ . . . /* Cancel! */
}
else
{
for (i= 0; i < len; i++)
{ /* Compare data in blkout and blkin: */
if (blkin[i] != blkout[i])
{
printf(”Verification failed at %x (%x, %x)\n”, addr+i, blkin[i],
blkout[i]);
return(ERR_VERIFY_FAILED); /* Verify failed! */
}
}
} /* for (i) */
```

注意：一个类似的数列可以用于检查擦除的范围。在这种情况下，blkin的内容和擦除模式0xff相比较。

检查成功后，MSP430可以复位，用户程序可以开始执行：

```
bslReset(0);
```

串行通讯口在程序结尾释放

```
comDone();
```

现在，用户已有所有部分的资源去写自己的应用程序，访问MSP430的启动程序载入器(Bootstrap)及根据特殊需要修改。

4.8 错误校验

在示范程序中没有错误校验的装置。当发现错误时编程失败。在某些情况下某些错误校验装置是很有用的。

如果，发送到MSP430的数据格式因为数据未知信号(DATA_NAK)而被拒绝，传送格

式可能简单重复。但是， 错误的数据可能已经被编程到FLASH。一个更复杂的装置，包含校验、擦除和再编程是需要的。

如果，接收格式是不正确的(wrong checksum, inconsistent lengths)，从MSP430发送到接收模块的上一级命令需要重复。

4.9 最初版本的启动程序载入器(Bootstrap)补丁

最初版本的启动程序载入器(Bootstrap)向FLASH编程需要一个小的补丁。该补丁在本节中描述，它的句柄包含在程序BSLDEMO.C (见附录A中*the Bootstrap Loader Demonstration Program* 清单)中。附录A中还能找到所需TI-TXT文件WAROUND.TXT。

示范程序中的补丁句柄可以用删除关闭（启动程序载入器(Bootstrap)将来版本），或注释。见下面：

```
#define WORKAROUND
```

代码部分所需要的工作区由预处理命令包围：

```
#ifdef WORKAROUND
. . .
#endif
```

获得访问被保护的启动程序载入器(Bootstrap)的命令后，需要调回装载器代码的子程序，准备补丁中堆栈的指针。这可以由装入PC地址的启动程序载入器(Bootstrap)来完成：

```
if ((error= bslTxRx(BSL_LOADPC, /* Command: load PC */
0x0C22, /* Address to load into PC */
0, /* No additional data! */
NULL, blkin)) != 0)
{ . . . /* Error! */
}
```

再次调用函数锁存保护命令，口令必须重新发送。

然后补丁能被写到RAM中。支持补丁的TXT文件WROUND.TXT可以在附录A中找到。下载完成后使用flash-programming函数分析文件：

```
programFlash(”WAROUND.TXT”, ACTION_PROGRAM | ACTION_VERIFY);
```

既然这样，在单次发送中使ACTION_PROGRAM 和 ACTION_VERIFY用在一起进行编程和校验补丁。也就是说文件WAROUND.TXT仅被读取一次。注意，补丁必须位于与执行程序相同的位置。

现在已准备好向flash memory编程。如果编程范围扩展，或者位于地址范围0x1000到0xffff（flash memory所在范围）。在发送程序数据格式之前，PC仅仅需要下载补丁起始地址（0x0220）：

```
#ifdef WORKAROUND
if (addr + len >= 0x1000)
{
error= bslTxRx(BSL_LOADPC, /* Command: Load PC */
0x0220, /* Address to load into PC */
0, /* No additional data! */
NULL, blkin);
if (error != 0) return(error);
}
#endif
error= bslTxRx(BSL_TXBLK, addr, len, blkout, blkin);
```

即使发送格式有一个确定的校验核，还是有一个小错误可能影响存储器单元（RAM或外围模块寄存器）。普通的工作区是不允许。这个错误与向flash memory编程的补丁无关；在所有其他情况下能被提供的唯一帮助是：如果发生这种情况时警告。该功能由包含在文件SSP.C中的comRxTx函数（见附录A中*Serial Communication Implementation File*）来完成。全局变量BSLMemAccessWarning允许打开、关闭消息：

```
{
WORD accessAddr= (0x0212 + (checksum^0xffff)) & 0xfffe;
if (BSLMemAccessWarning && (accessAddr < 0x1000))
{
printf(”WARNING: This command might change data ”
”at address %x or %x!\n”,
accessAddr, accessAddr + 1);
}
}
```

5 参考书目

1. *MSP430F11x Mixed Signal Microcontroller* data sheet , literature number SLAS256
2. *MSP430F11x1 Mixed Signal Controller* data sheet, literature number SLAS241
3. Graf, Franz. *Features of the MSP430 Bootstrap Loader in the MSP430F1121*, literature number SLAA089.
4. Denver, Allen. *Serial Communication in Win32*, Microsoft Developer Network (MSDN) Library
5. *Microsoft Win32 Software Development Kit (SDK) Documentation*

附录A 清单

A.1 建立示范程序

为建立示范程序需要编译文件bslcomm.c和bsldemo.c，以及连接生成的目标文件。不需要编译单独的文件ssp.c，因为它直接包含在文件bslcomm.c中。另一个不同的方法，拷贝或粘帖SSP.C的内容到bslcomm.c在下面行的位置中、移动该行：

```
#include ”ssp.c”
```

例如，如果选择*Win32 Console Application*模板,使用Visual C++，创建一个空的新的工程。在工程中包括文件bsldemo.c，bslcomm.c,和bslcomm.h并建立。在这之前，自动包含文件ssp.c 和ssp.h。如果在工程中包含SSP.C文件则创建进程将失败。

A.2 启动程序载入器(Bootstrap)通讯头文件——bslcomm.h

```
/*****************************************************************
*
* Copyright (C) 1999–2000 Texas Instruments, Inc.
*
*****************************************************************/
#ifndef BSLComm__H
#define BSLComm__H
#include ”ssp.h”
/* Transmit password to boot loader: */
#define BSL_TXPWORD 0x10
/* Transmit block to boot loader: */
#define BSL_TXBLK 0x12
/* Receive block from boot loader: */
```

```
#define BSL_RXBLK 0x14
/* Erase one segment: */
#define BSL_ERASE 0x16
/* Erase complete FLASH memory: */
#define BSL_MERAS 0x18
/* Load PC and start execution: */
#define BSL_LOADPC 0x1A
/* Bootstrap loader synchronization error: */
#define ERR_BSL_SYNC 99
#ifdef __cplusplus
extern ”C” {
#endif
extern int BSLMemAccessWarning;
/*————————————————————————————————————*/
void bslReset(BOOL invokeBSL);
/* Applies BSL entry sequence on RST/NMI and TEST/VPP pins
* Parameters: invokeBSL = TRUE: complete sequence
* invokeBSL = FALSE: only RST/NMI pin accessed
*/
/*————————————————————————————————————*/
int bslSync();
/* Transmits Synchronization character and expects to
* receive Acknowledge character
* Return == 0: OK
* Return == 1: Sync. failed.
*/
/*————————————————————————————————————*/
int bslTxRx(BYTE cmd, WORD addr, WORD len,
BYTE blkout[], BYTE blkin[]);
/* Transmits a command (cmd) with its parameters:
* start–address (addr), length (len) and additional
* data (blkout) to boot loader.
* Parameters return by boot loader are passed via blkin.
* Return == 0: OK
* Return != 0: Error!
*/
#ifdef __cplusplus
}
#endif
#endif
/* EOF */
```

A.3 启动程序载入器(Bootstrap)通讯执行文件——bslcomm.c

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

```
* Copyright (C) 1999–2000 Texas Instruments, Inc.
*
*****************************************************************/
#include <windows.h>
#include <string.h>
#include <stdio.h>
#include <fcntl.h>
#include ”bslcomm.h”
#include ”ssp.c”
#define BSL_SYNC 0x80
/* 1: Warning, if access to memory below 0x1000 is possible.
* This can happen due to an error in the first version(s) of
* the bootstrap loader code in combination with specific
* checksum values.
* 0: No Warning.
*/
int BSLMemAccessWarning= 0; /* Default: no warning. */
/*————————————————————————————————*/
void SetRSTpin(BOOL level)
/* Controls RST/NMI pin (0: GND; 1: VCC) */
{
if (level == TRUE)
comDCB.fDtrControl = DTR_CONTROL_ENABLE;
else
comDCB.fDtrControl = DTR_CONTROL_DISABLE;
SetCommState(hComPort, &comDCB);
} /* SetRSTpin */
void SetTESTpin(BOOL level)
/* Controls TEST pin (0: VCC; 1: GND) */
{
if (level == TRUE)
comDCB.fRtsControl = RTS_CONTROL_ENABLE;
else
comDCB.fRtsControl = RTS_CONTROL_DISABLE;
SetCommState(hComPort, &comDCB);
} /* SetTESTpin */
/*————————————————————————————————*/
void bslReset(BOOL invokeBSL)
/* Applies BSL entry sequence on RST/NMI and TEST/VPP pins
* Parameters: invokeBSL = TRUE: complete sequence
* invokeBSL = FALSE: only RST/NMI pin accessed
*
* RST is inverted twice on boot loader hardware
* TEST is inverted (only once)
```

```
* Need positive voltage on DTR, RTS for power–supply of hardware
*/
{
/* To charge capacitor on boot loader hardware: */
SetRSTpin(1);
SetTESTpin(1);
delay(250);
SetRSTpin(0); /* RST pin: GND */
if (invokeBSL)
{
SetTESTpin(1); /* TEST pin: GND */
SetTESTpin(0); /* TEST pin: Vcc */
SetTESTpin(1); /* TEST pin: GND */
SetTESTpin(0); /* TEST pin: Vcc */
SetRSTpin (1); /* RST pin: Vcc */
SetTESTpin(1); /* TEST pin: GND */
}
else
{
SetRSTpin(1); /* RST pin: Vcc */
}
/* Give MSP430's oscillator time to stabilize: */
delay(250);
/* Clear buffers: */
PurgeComm(hComPort, PURGE_TXCLEAR); PurgeComm(hComPort, PURGE_RXCLEAR);
} /* bslReset */
/*————————————————————————————————————————*/
int bslSync()
/* Transmits Synchronization character and expects to
* receive Acknowledge character
* Return == 0: OK
* Return == 1: Sync. failed.
*/
{
BYTE ch;
int rxCount, loopcnt;
const BYTE cLoopOut = 3; /* Max. trials to get synchronization */
DWORD NrTx;
DWORD NrRx;
for (loopcnt=0; loopcnt < cLoopOut; loopcnt++)
{
PurgeComm(hComPort, PURGE_RXCLEAR); /* Clear receiving queue */
/* Send synchronization byte: */
ch = BSL_SYNC;
```

```
WriteFile(hComPort, &ch, 1, &NrTx, NULL);
/* Wait for 1 byte; Timeout: 100ms */
rxCount= comWaitForData(1, 100);
if (rxCount > 0)
{
ReadFile(hComPort, &ch, 1, &NrRx, NULL);
if (ch == DATA_ACK)
{ return(ERR_NONE); } /* Sync. successful */
}
} /* for (loopcount) */
return(ERR_BSL_SYNC); /* Sync. failed */
} /* bslSync */
/*————————————————————————————————————————*/
int bslTxRx(BYTE cmd, WORD addr, WORD len,
BYTE* blkout, BYTE* blkin)
/* Transmits a command (cmd) with its parameters:
* start–address (addr), length (len) and additional
* data (blkout) to boot loader.
* Parameters return by boot loader are passed via blkin.
* Return == 0: OK
* Return != 0: Error!
*/
{
BYTE dataOut[MAX_FRAME_SIZE];
int error;
WORD length= 4;
/* Make sure that len is even, when sending data to BSL: */
if ((cmd == BSL_TXBLK) && ((len % 2) != 0))
{ /* Inc. len and fill blkout with 0xFF
* => even number of bytes to send!
*/
blkout[(len++)]= 0xFF;
}
/* Make sure that len is even, if receiving data from BSL: */
if ((cmd == BSL_RXBLK) && ((len % 2) != 0))
{
len++;
}
if ((cmd == BSL_TXBLK) || (cmd == BSL_TXPWORD))
{
length = len + 4;
}
/* Add necessary information data to frame: */
dataOut[0] = (BYTE)( addr & 0x00ff);
```

```
dataOut[1] = (BYTE)((addr >> 8) & 0x00ff);
dataOut[2] = (BYTE)( len & 0x00ff);
dataOut[3] = (BYTE)((len >> 8) & 0x00ff);
if (blkout != NULL)
{ /* Copy data out of blkout into frame: */
memcpy(&dataOut[4], blkout, len);
}
if (bslSync() != ERR_NONE)
{
return(ERR_BSL_SYNC);
}
/* Send frame: */
error = comTxRx(cmd, dataOut, (BYTE)length);
if (blkin != NULL)
{ /* Copy received data out of frame buffer into blkin: */
memcpy(blkin, &rxFrame[4], rxFrame[2]);
}
return (error);
}
/* EOF */
```

A.4 串口通讯头文件——SSP.h

```
/*************************************************************************
*
* Copyright (C) 1998–2000 Texas Instruments, Inc.
*
************************************************************************/
#ifndef SSP__H
#define SSP__H
#include <windows.h>
#define MODE_SSP 0
#define MODE_BSL 1
/* Error Codes:
*/
/* No Error: */
#define ERR_NONE 0
/* Unspecific error: */
#define ERR_COM 1
/* OpenComm failed: */
#define ERR_OPEN_COMM 2
/* SetCommState failed: */
#define ERR_SET_COMM_STATE 3
/* Synchronization failed: */
#define ERR_SYNC_FAILED 4
/* Unspecific error concerning transmission of command: */
```

```
#define ERR_SEND_COMMAND 5
/* Timeout while receiving header": */
#define ERR_RX_HDR_TIMEOUT 6
/* NAK received: */
#define ERR_RX_NAK 7
/* Command did not send ACK: indicates that it didn't complete correctly: */
#define ERR_CMD_NOT_COMPLETED 8
/* Command failed, is not defined or is not allowed: */
#define ERR_CMD_FAILED 9
/* CloseComm failed: */
#define ERR_CLOSE_COMM 10
/* Header Definitions: */
#define CMD_FAILED 0x70
#define DATA_FRAME 0x80
#define DATA_ACK 0x90
#define DATA_NAK 0xA0
#define QUERY_POLL 0xB0
#define QUERY_RESPONSE 0x50
#define OPEN_CONNECTION 0xC0
#define ACK_CONNECTION 0x40
#define DEFAULT_TIMEOUT 300
#define DEFAULT_PROLONG 10
#define MAX_FRAME_SIZE 256
#define MAX_DATA_BYTES 250
#define MAX_DATA_WORDS 125
#ifdef __cplusplus
extern "C" {
#endif
/*_____________________________________________________________
* Support Subroutines:
*_____________________________________________________________
*/
/*_____________________________________________________________*/
extern DWORD calcTimeout(DWORD startTime);
/* Calculates the difference between startTime and the actual
* windows time (in milliseconds).
*/
/*_____________________________________________________________*/
extern void delay(DWORD time);
/* Delays the execution by a given time in ms.
*/
/*_____________________________________________________________*/
extern int comWaitForData(int count, DWORD timeout);
/* Waits until a given number (count) of bytes was received or a
```

```
* given time (timeout) has passed.
*/
extern void comTxHeader(const BYTE txHeader);
/*_____________________________________________________________________
* Communication Subroutines:
*_____________________________________________________________________
*/
extern int comGetLastError();
/* Returns the error code generated by the last function call to
* a SERCOMM–Function. If this function returned without errors,
* comGetLastError will return zero (errNoError) as well.
*/
/*_____________________________________________________________________*/
#ifdef __cplusplus
extern int comInit(LPCSTR lpszDevice = ”COM1”,
DWORD aTimeout = DEFAULT_TIMEOUT,
int aProlongFactor= DEFAULT_PROLONG);
#else
extern int comInit(LPCSTR lpszDevice,
DWORD aTimeout, int aProlongFactor);
#endif
/* Tries to open the serial port given in ’lpszDevice’ and
* initializes the port and global variables.
* The timeout and the number of allowed errors is multiplied by
* ’aProlongFactor’ after transmission of a command to give
* plenty of time to the micro controller to finish the command.
* Returns zero if the function is successful.
*/
/*_____________________________________________________________________*/
extern int comDone();
/* Closes the used serial port.
* This function must be called at the end of a program,
* otherwise the serial port might not be released and can not be
* used in other programs.
* Returns zero if the function is successful.
*/
#ifdef __cplusplus
}
#endif
#endif
/* EOF */
```

A.5 串口通讯执行文件——SSP.C

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

```
* Copyright (C) 1998–2000 Texas Instruments, Inc.
*
**********************************************************************/
#include <string.h>
#include <stdio.h>
#include <windows.h>
#include ”ssp.h”
/* Global Constants: */
/* Size of internal WINDOWS–Comm–Buffer: */
#define QUEUE_SIZE 512
#define MAX_FRAME_COUNT 16
#define MAX_ERR_COUNT 5
/* Global Variables: */
const unsigned short protocolMode= MODE_BSL;
HANDLE hComPort; /* COM–Port Handle */
DCB comDCB; /* COM–Port Control–Settings */
COMSTAT comState; /* COM–Port Status–Information */
COMMTIMEOUTS orgTimeouts; /* Original COM–Port Time–out */
/* Time in milliseconds until a timeout occurs: */
DWORD timeout = DEFAULT_TIMEOUT;
/* Factor by which the timeout after sending a frame is prolonged: */
int prolongFactor= DEFAULT_PROLONG;
/* Variable to save the latest error (used by comGetLastError): */
int lastError;
BYTE seqNo, reqNo, txPtr, rxPtr;
BYTE rxFrame[MAX_FRAME_SIZE];
DWORD nakDelay; /* Delay before DATA_NAK will be send */
/**********************************************************************/
DWORD calcTimeout(DWORD startTime) /* exported! */
/* Calculates the difference between startTime and the actual
* windows time (in milliseconds).
*/
{
return((DWORD)(GetTickCount() – startTime));
}
/*———————————————————————————————————*/
void delay(DWORD time) /* exported! */
/* Delays the execution by a given time in ms.
*/
{
#ifndef WIN32
DWORD startTime= GetTickCount();
while (calcTimeout(startTime) < time);
#else
```

```
Sleep(time);
#endif
}
/*————————————————————————————————————————*/
WORD calcChecksum(BYTE data[], WORD length)
/* Calculates a checksum of ”data”.
*/
{
WORD* i_data;
WORD checksum= 0;
BYTE i= 0;
i_data= (WORD*)data;
for (i= 0; i < length/2; i++)
{
checksum^= i_data[i]; /* xor–ing */
}
return(checksum ^ 0xffff); /* inverting */
}
/*————————————————————————————————————————*/
int comWaitForData(int count, DWORD timeout) /* exported! */
/* Waits until a given number (count) of bytes was received or a
* given time (timeout) has passed.
*/
{
DWORD errors;
int rxCount= 0;
DWORD startTime= GetTickCount();
do
{
ClearCommError(hComPort, &errors, &comState);
} while (((rxCount= comState.cbInQue) < count) &&
(calcTimeout(startTime) <= timeout));
return(rxCount);
}
/*————————————————————————————————————————*/
int comRxHeader(BYTE *rxHeader, BYTE *rxNum,
DWORD timeout)
{
BYTE Hdr;
DWORD dwRead;
if (comWaitForData(1, timeout) >= 1)
{
ReadFile(hComPort, &Hdr, 1, &dwRead, NULL);
*rxHeader= Hdr & 0xf0;
```

```
*rxNum = Hdr & 0x0f;
if (protocolMode == MODE_BSL)
{ reqNo= 0;
seqNo= 0;
*rxNum= 0;
}
return(ERR_NONE);
}
else
{
*rxHeader= 0;
*rxNum= 0;
return(lastError= ERR_RX_HDR_TIMEOUT);
}
}
/*_____*/
void comTxHeader(const BYTE txHeader)
{
DWORD dwWrite;
BYTE Hdr= txHeader;
WriteFile(hComPort, &Hdr, 1, &dwWrite, NULL);
}
/*****************************************************************/
int comGetLastError()
/* Returns the error code generated by the last function call to
* a SERCOMM–Function. If this function returned without errors,
* comGetLastError will return zero (errNoError) as well.
*/
{ return(lastError); }
/*****************************************************************/
int comInit(LPCSTR lpszDevice, DWORD aTimeout, int aProlongFactor)
/* Tries to open the serial port given in 'lpszDevice' and
* initializes the port and global variables.
* The timeout and the number of allowed errors is multiplied by
* 'aProlongFactor' after transmission of a command to give
* plenty of time to the micro controller to finish the command.
* Returns zero if the function is successful.
*/
{
COMMTIMEOUTS timeouts;
DWORD dwCommEvents;
/* Init. global variables: */
seqNo= 0;
reqNo= 0;
```

```
rxPtr= 0;
txPtr= 0;
timeout= aTimeout;
prolongFactor= aProlongFactor;
hComPort= CreateFile(lpszDevice, GENERIC_READ | GENERIC_WRITE,
0, 0, OPEN_EXISTING, 0, 0);
/* In this application the serial port is used in
* nonoverlapped mode!
*/
if (hComPort == INVALID_HANDLE_VALUE)
{
hComPort= 0;
return (lastError= ERR_OPEN_COMM); /* Error! */
}
if (SetupComm(hComPort, QUEUE_SIZE, QUEUE_SIZE) == 0)
{
CloseHandle(hComPort);
hComPort= 0;
return (lastError= ERR_OPEN_COMM); /* Error! */
}
/* Save original timeout values: */
GetCommTimeouts(hComPort, &orgTimeouts);
/* Set Windows timeout values (disable build–in timeouts): */
timeouts.ReadIntervalTimeout= 0;
timeouts.ReadTotalTimeoutMultiplier= 0;
timeouts.ReadTotalTimeoutConstant= 0;
timeouts.WriteTotalTimeoutMultiplier= 0;
timeouts.WriteTotalTimeoutConstant= 0;
if (!SetCommTimeouts(hComPort, &timeouts))
{
CloseHandle(hComPort);
hComPort= 0;
return (lastError= ERR_OPEN_COMM); /* Error! */
}
dwCommEvents= EV_RXCHAR | EV_TXEMPTY | EV_RXFLAG | EV_ERR;
SetCommMask(hComPort, dwCommEvents);
/* Get state and modify it: */
if (!GetCommState(hComPort, &comDCB))
{
CloseHandle(hComPort);
hComPort= 0;
return (lastError= ERR_OPEN_COMM); /* Error! */
}
comDCB.BaudRate = CBR_9600; /* Startup–Baudrate: 9,6kBaud */
```

```
comDCB.ByteSize = 8;
nakDelay= (DWORD)((11*MAX_FRAME_SIZE)/9.6);
comDCB.Parity = EVENPARITY;
comDCB.StopBits = ONESTOPBIT;
comDCB.fBinary = TRUE; /* Enable Binary Transmission */
comDCB.fParity = TRUE; /* Enable Parity Check */
comDCB.ErrorChar = (char)0xff;
/* Char. w/ Parity–Err are replaced with 0xff
*(if fErrorChar is set to TRUE)
*/
comDCB.fRtsControl = RTS_CONTROL_ENABLE; /* For power supply */
comDCB.fDtrControl = DTR_CONTROL_ENABLE; /* For power supply */
comDCB.fOutxCtsFlow= FALSE; comDCB.fOutxDsrFlow= FALSE;
comDCB.fOutX = FALSE; comDCB.fInX = FALSE;
comDCB.fNull = FALSE;
comDCB.fErrorChar = FALSE;
/* Assign new state: */
if (!SetCommState(hComPort, &comDCB))
{
CloseHandle(hComPort);
hComPort= 0;
return(lastError= ERR_SET_COMM_STATE); /* Error! */
}
/* Clear buffers: */
PurgeComm(hComPort, PURGE_TXCLEAR | PURGE_TXABORT);
PurgeComm(hComPort, PURGE_RXCLEAR | PURGE_RXABORT);
return(lastError= 0);
} /* comInit */
/*****************************************************************/
int comDone()
/* Closes the used serial port.
* This function must be called at the end of a program,
* otherwise the serial port might not be released and can not be
* used in other programs.
* Returns zero if the function is successful.
*/
{
DWORD errors;
DWORD startTime= GetTickCount();
/* Wait until data is transmitted, but not too long... (Timeout–Time) */
do
{
ClearCommError(hComPort, &errors, &comState);
} while ((comState.cbOutQue > 0) &&
```

```
(calcTimeout(startTime) < timeout));
/* Clear buffers: */
PurgeComm(hComPort, PURGE_TXCLEAR | PURGE_TXABORT);
PurgeComm(hComPort, PURGE_RXCLEAR | PURGE_RXABORT);
/* Restore original timeout values: */
SetCommTimeouts(hComPort, &orgTimeouts);
/* Close COM–Port: */
if (!CloseHandle(hComPort))
return(lastError= ERR_CLOSE_COMM); /* Error! */
else
return(lastError= ERR_NONE);
} /* comDone */
/*****************************************************************/
/*———————————————————————————————*/
int comRxFrame(BYTE *rxHeader, BYTE *rxNum)
{
DWORD dwRead;
WORD checksum;
BYTE* rxLength;
WORD rxLengthCRC;
rxFrame[0]= DATA_FRAME | *rxNum;
if (comWaitForData(3, timeout) >= 3)
{
ReadFile(hComPort, &rxFrame[1], 3, &dwRead, NULL);
if ((rxFrame[1] == 0) && (rxFrame[2] == rxFrame[3]))
{
rxLength= &rxFrame[2]; /* Pointer to rxFrame[2] */
rxLengthCRC= *rxLength + 2; /* Add CRC–Bytes to length */
if (comWaitForData(rxLengthCRC, timeout) >= rxLengthCRC)
{
ReadFile(hComPort, &rxFrame[4], rxLengthCRC, &dwRead, NULL);
/* Check received frame: */
checksum= calcChecksum(rxFrame, (WORD)(*rxLength+4));
/* rxLength+4: Length with header but w/o CRC */
if ((rxFrame[*rxLength+4] == (BYTE)checksum) &&
(rxFrame[*rxLength+5] == (BYTE)(checksum >> 8)))
{
return(ERR_NONE);
/* Frame received correctly (=> send next frame) */
} /* if (Checksum correct?) */
} /* if (Data: no timeout?) */
} /* if (Add. header info. correct?) */
} /* if (Add. header info.: no timeout?) */
return(ERR_COM); /* Frame has errors! */
```

```
} /* comRxFrame */
/*————————————————————————————————————————*/
int comTxRx(BYTE cmd, BYTE dataOut[], BYTE length)
/* Sends the command cmd with the data given in dataOut to the
* microcontroller and expects either an acknowledge or a frame
* with result from the microcontroller. The results are stored
* in dataIn (if not a NULL pointer is passed).
* In this routine all the necessary protocol stuff is handled.
* Returns zero if the function was successful.
*/
{
DWORD dwWrite;
DWORD errors;
BYTE txFrame[MAX_FRAME_SIZE];
WORD checksum= 0;
int k= 0;
int errCtr= 0;
int resendCtr= 0;
BYTE rxHeader= 0;
BYTE rxNum= 0;
int resentFrame= 0;
int pollCtr= 0;
/* Transmitting part ————————————————————————————*/
/* Prepare data for transmit */
if ((length % 2) != 0)
{ /* Fill with one byte to have even number of bytes to send */
if (protocolMode == MODE_BSL)
dataOut[length++]= 0xFF; // fill with 0xFF
else
dataOut[length++]= 0; // fill with zero
}
txFrame[0]= DATA_FRAME | seqNo;
txFrame[1]= cmd;
txFrame[2]= length;
txFrame[3]= length;
reqNo= (seqNo + 1) % MAX_FRAME_COUNT;
memcpy(&txFrame[4], dataOut, length);
checksum= calcChecksum(txFrame, (WORD)(length+4));
txFrame[length+4]= (BYTE)(checksum);
txFrame[length+5]= (BYTE)(checksum >> 8);
{
WORD accessAddr= (0x0212 + (checksum^0xffff)) & 0xfffe;
/* 0x0212: Address of wCHKSUM */
if (BSLMemAccessWarning && (accessAddr < 0x1000))
```

```
{
printf(”WARNING: This command might change data ”
”at address %x or %x!\n”,
accessAddr, accessAddr + 1);
}
}
/* Transmit data: */
k= 0;
/* Clear receiving queue: */
PurgeComm(hComPort, PURGE_RXCLEAR | PURGE_RXABORT);
do
{
WriteFile(hComPort, &txFrame[k++], 1, &dwWrite, NULL);
ClearCommError(hComPort, &errors, &comState);
} while ((k < length + 6) && (comState.cbInQue == 0));
/* Check after each transmitted character,
* if microcontroller did send a character (probably a NAK!).
*/
/* Receiving part ————————————————————————————*/
rxFrame[2]= 0;
rxFrame[3]= 0; /* Set lengths of received data to 0! */
do
{
lastError= 0; /* Clear last error */
if (comRxHeader(&rxHeader, &rxNum, timeout*prolongFactor) == 0)
/* prolong timeout to allow execution of sent command */
{ /* => Header received */
do
{
resentFrame= 0;
switch (rxHeader)
{ case DATA_ACK:
if (rxNum == reqNo)
{ seqNo= reqNo;
return(lastError= ERR_NONE);
/* Acknowledge received correctly => next frame */
}
break; /* case DATA_ACK */
case DATA_NAK:
return(lastError= ERR_RX_NAK);
break; /* case DATA_NAK */
case DATA_FRAME:
if (rxNum == reqNo)
if (comRxFrame(&rxHeader, &rxNum) == 0)
```

```
return(lastError= ERR_NONE);
break; /* case DATA_FRAME */
case CMD_FAILED:
/* Frame ok, but command failed. */
return(lastError= ERR_CMD_FAILED);
break; /* case CMD_FAILED */
default:
;
} /* switch */
errCtr= MAX_ERR_COUNT;
} while ((resentFrame == 0) && (errCtr < MAX_ERR_COUNT));
} /* if (comRxHeader) */
else
{ /* => Timeout while receiving header */
errCtr= MAX_ERR_COUNT;
} /* else (comRxHeader) */
} while (errCtr < MAX_ERR_COUNT);
if (lastError == ERR_CMD_NOT_COMPLETED)
{ /* Accept QUERY_RESPONSE as real ACK and correct Seq.–No.: */
seqNo= reqNo;
}
if (lastError == ERR_NONE)
return(lastError= ERR_COM);
else
return(lastError);
} /* comTxRx */
/* EOF */
```

A.6 启动程序载入器(Bootstrap)示范程序——bsldemo.c

```
/*****************************************************************
*
* Copyright (C) 1999–2000 Texas Instruments, Inc.
*
*****************************************************************
*
* Project: MSP430 Bootstrap Loader Demonstration Program
** File: BSLDEMO.C
*
* Description:
* This is the main program of the bootstrap loader
* demonstration.
* The main function holds the general sequence to access the
* bootstrap loader and program/verify a file.
* The parsing of the TI TXT file is done in a separate
* function.
```

```
*
* A couple of parameters can be passed to the program to
* control its functions. For a detailed description see the
* appendix of the corresponding application note.
************************************************************/
#include <string.h>
#include <stdio.h>
#include <windows.h>
#include ”bslcomm.h”
/*-----------------------------------------------------------
* Defines:
*-----------------------------------------------------------
*/
/* The ”WORKAROUND” definition includes code for a workaround
* required by the first version(s) of the bootstrap loader.
*/
#define WORKAROUND
/* If ”DEBUG” is defined, all checked and programmed blocks are
* logged on the screen.
*/
//#define DEBUG
/* Error: verification failed: */
#define ERR_VERIFY_FAILED 98
/* Error: erase check failed: */
#define ERR_ERASE_CHECK_FAILED 97
/* Error: unable to open input file: */
#define ERR_FILE_OPEN 96
/* Mask: program data: */
#define ACTION_PROGRAM 0x01
/* Mask: verify data: */
#define ACTION_VERIFY 0x02
/* Mask: erase check: */
#define ACTION_ERASE_CHECK 0x04
/* Mask: transmit password: */
/* Note: Should not be used in conjunction with any other action! */
#define ACTION_PASSWD 0x08
/* Max. bytes sent within one frame if parsing a TI TXT file.
* ( > 16 and == n*16 and <= MAX_DATA_BYTES!)
*/
const int maxData= 64;
/*-----------------------------------------------------------
* Global Variables:
*-----------------------------------------------------------
*/
```

```
/* Buffers used to store data transmitted to and received from BSL: */
BYTE blkin [MAX_DATA_BYTES]; /* Receive buffer */
BYTE blkout[MAX_DATA_BYTES]; /* Transmit buffer */
struct toDoList
{
unsigned MassErase : 1;
unsigned EraseCheck: 1;
unsigned Program : 1;
unsigned Verify : 1;
unsigned Reset : 1;
unsigned Wait : 1; /* Wait for <Enter> at end of program */
/* (0: no; 1: yes): */
unsigned OnePass : 1; /* Do EraseCheck, Program and Verify */
/* in one pass (TI TXT file is read */
/* only once) */
} toDo;
/*_____________________________________________________________________
* Functions:
*_____________________________________________________________________
*/
int verifyBlk(WORD addr, WORD len, unsigned action)
{
int i= 0;
int error= 0;
if ((action & (ACTION_VERIFY | ACTION_ERASE_CHECK)) != 0)
{
#ifdef DEBUG
printf("Check starting at %x, %i bytes... ", addr, len);
#endif
error= bslTxRx(BSL_RXBLK, addr, len, NULL, blkin);
#ifdef DEBUG
printf("Error: %i\n", error);
#endif
if (error != 0)
{
return(error); /* Cancel, if read error */
}
else
{
for (i= 0; i < len; i++)
{
if ((action & ACTION_VERIFY) != 0)
{
/* Compare data in blkout and blkin: */
```

```
if (blkin[i] != blkout[i])
{
printf(”Verification failed at %x (%x, %x)\n”, addr+i, blkin[i],
blkout[i]);
return(ERR_VERIFY_FAILED); /* Verify failed! */
}
continue;
}
if ((action & ACTION_ERASE_CHECK) != 0)
{
/* Compare data in blkin with erase pattern: */
if (blkin[i] != 0xff)
{
printf(”Erase Check failed at %x (%x)\n”, addr+i, blkin[i]);
return(ERR_ERASE_CHECK_FAILED); /* Erase Check failed! */
}
continue;
} /* if ACTION_ERASE_CHECK */
} /* for (i) */
} /* else */
} /* if ACTION_VERIFY | ACTION_ERASE_CHECK */
return(error);
}
int programBlk(WORD addr, WORD len, unsigned action)
{
int i= 0;
int error= 0;
if ((action & ACTION_PASSWD) != 0)
{
return(bslTxRx(BSL_TXPWORD, /* Command: Transmit Password */
addr, /* Address of interupt vectors */
len, /* Number of bytes */
blkout, blkin));
} /* if ACTION_PASSWD */
/* Check, if specified range is erased: */
error= verifyBlk(addr, len, action & ACTION_ERASE_CHECK);
if (error != 0)
{
return(error);
}
if ((action & ACTION_PROGRAM) != 0)
{
#ifdef WORKAROUND
if (addr + len >= 0x1000)
```

```
{
/* Load PC with 0x0220.
* This will invoke the patched flash programming subroutine.
*/
error= bslTxRx(BSL_LOADPC, /* Command: Load PC */
0x0220, /* Address to load into PC */
0, /* No additional data! */
NULL, blkin);
if (error != 0) return(error);
}
BSLMemAccessWarning= 0; /* Error is removed within workaround code */
#endif
#ifdef DEBUG
printf("Program starting at %x, %i bytes... ", addr, len);
#endif
/* Program block: */
error= bslTxRx(BSL_TXBLK, addr, len, blkout, blkin);
#ifdef DEBUG
printf("Error: %i\n", error);
#endif
#ifdef WORKAROUND
BSLMemAccessWarning= 1; /* Turn warning back on. */
#endif
if (error != 0)
{
return(error); /* Cancel, if error (ACTION_VERIFY is skipped!) */
}
} /* if ACTION_PROGRAM */
/* Verify block: */
error= verifyBlk(addr, len, action & ACTION_VERIFY);
if (error != 0)
{
return(error);
}
return(error);
} /* programBlk */
int programTIText (char *filename, unsigned action)
{
int next= 1;
int error=0;
int linelen= 0;
int linepos= 0;
WORD dataframelen=0;
WORD currentAddr;
```

```
char strdata[128];
FILE* infile;
if ((infile = fopen(filename, ”rb”)) == 0)
{ return(ERR_FILE_OPEN); }
/* Convert data for MSP430, TXT–File is parsed line by line: */
for (next= 1; next>=1; )
{
/* Read one line: */
if ((fgets(strdata, 127, infile) == 0) ||
/* if End Of File or */
(strdata[0] == ’q’))
/* if q (last character in file) */
{ /* => send frame and quit */
if (dataframelen > 0) /* Data in frame? */
{
error= programBlk(currentAddr, dataframelen, action);
dataframelen=0;
}
next=0; /* Quit! */
continue;
}
linelen= strlen(strdata);
if (strdata[0] == ’@’)
/* if @ => new address => send frame and set new addr. */
{
if (dataframelen > 0)
{
error= programBlk(currentAddr, dataframelen, action);
dataframelen=0;
}
sscanf(&strdata[1], ”%lx\n”, &currentAddr);
continue;
}
/* Transfer data in line into blkout: */
for(linepos= 0;
linepos < linelen–3; linepos+= 3, dataframelen++)
{
sscanf(&strdata[linepos], ”%3x”, &blkout[dataframelen]);
/* (Max 16 bytes per line!) */
}
if (dataframelen > maxData–16)
/* if frame is getting full => send frame */
{
error= programBlk(currentAddr, dataframelen, action);
```

```
currentAddr+= dataframelen;
dataframelen=0;
}
if (error != 0) next=0; /* Cancel loop, if any error */
}
fclose(infile);
return(error);
} /* programTIText */
int txPasswd(char* passwdFile)
{
int i;
if (passwdFile == NULL)
{
/* Send ”standard” password to get access to protected functions. */
printf(”Transmit Password...\n”);
/* Fill blkout with 0xff
* (Flash is completely erased, the contents of all Flash cells is 0xff)
*/
for (i= 0; i < 0x20; i++)
{
blkout[i]= 0xff;
}
return(bslTxRx(BSL_TXPWORD, /* Command: Transmit Password */
0xffe0, /* Address of interupt vectors */
0x0020, /* Number of bytes */
blkout, blkin));
}
else
{
/* Send TI TXT file holding interrupt vector data as password: */
printf(”Transmit password file %s...\n”, passwdFile);
return(programTIText(passwdFile, ACTION_PASSWD));
}
} /* txPasswd */
int signOff(int error, BOOL passwd)
{
if (toDo.Reset)
{
bslReset(0); /* Reset MSP430 and start user program. */
}
switch (error)
{
case ERR_NONE:
printf(”Programming completed!\n”);
```

```
break;
case ERR_BSL_SYNC:
printf(”ERROR: Synchronization failed!\n”);
printf(”Device with boot loader connected?\n”);
break;
case ERR_VERIFY_FAILED:
printf(”ERROR: Verification failed!\n”);
break;
case ERR_FILE_OPEN:
printf(”ERROR: Unable to open input file!\n”);
break;
default:
if ((passwd) && (error == ERR_RX_NAK))
/* If last command == transmit password && Error: */
printf(”ERROR: Password not accepted!\n”);
else
printf(”ERROR: Communication Error!\n”);
} /* switch */
if (toDo.Wait)
{
printf(”Press <ENTER> ...\n”); getchar();
}
comDone(); /* Release serial communication port. */
/* After having released the serial port,
* the target is no longer supplied via this port!
*/
if (error == ERR_NONE)
return(0);
else
return(1);
} /* signOff */
/*————————————————————————————————————
* Main:
*————————————————————————————————————
*/
int main(int argc, char *argv[])
{
int error= 0;
int i, j;
char comPortName[10]= ”COM1”; /* Default setting. */
char *filename= NULL;
char *passwdFile= NULL;
/* Default: all actions turned on: */
toDo.MassErase = 1;
```

```
toDo.EraseCheck= 1;
toDo.Program = 1;
toDo.Verify = 1;
toDo.Reset = 1;
toDo.Wait = 0; /* Do not wait for <Enter> at the end! */
toDo.OnePass = 0; /* Do erase check, program and verify */
/* sequential! */
#ifdef WORKAROUND
/* Show memory access warning, if working with bootstrap
* loader version(s) requiring the workaround patch.
*/
BSLMemAccessWarning= 1;
#endif
/*————————————————————————————————————————
* Parse Command Line Parameters ...
*————————————————————————————————————————
*/
if (argc > 1)
{
for (i= 1; i < (argc – 1); i++)
{
switch (argv[i][0])
{
case ’–’:
switch (argv[i][1])
{
case ’c’: case ’C’:
memcpy(comPortName, &argv[i][2], strlen(argv[i])–2);
break;
case ’p’: case ’P’:
passwdFile= &argv[i][2];
break;
case ’w’: case ’W’:
toDo.Wait= 1; /* Do wait for <Enter> at the end! */
break;
case ’1’:
toDo.OnePass= 1;
#ifdef WORKAROUND
printf(”WARNING: One pass programming is discouraged!\n”);
#endif
break;
default:
printf(”ERROR: Illegal command line parameter!\n”);
} /* switch argv[i][1] */
```

```
break; /* '–' */
case '+':
/* Turn all actions off: */
toDo.MassErase = 0;
toDo.EraseCheck= 0;
toDo.Program = 0;
toDo.Verify = 0;
toDo.Reset = 0;
toDo.Wait = 0;
/* Turn only specified actions back on: */
for (j= 1; j < (int)(strlen(argv[i])); j++)
{
switch (argv[i][j])
{
case 'e': case 'E':
/* Erase Flash */
toDo.MassErase = 1;
break;
case 'c': case 'C':
/* Erase Check (by file) */
toDo.EraseCheck= 1;
break;
case 'p': case 'P':
/* Program file */
toDo.Program = 1;
break;
case 'v': case 'V':
/* Verify file */
toDo.Verify = 1;
break;
case 'r': case 'R':
/* Reset MSP430 before waiting for <Enter> */
toDo.Reset = 1;
break;
case 'w': case 'W':
/* Wait for <Enter> before closing serial port */
toDo.Wait = 1;
break;
default:
printf("ERROR: Illegal action specified!\n");
} /* switch */
} /* for (j) */
break; /* '+' */
default:
```

```
printf(”ERROR: Illegal command line parameter!\n”);
} /* switch argv[i][0] */
} /* for (i) */
filename= argv[i];
}
else
{
printf(”ERROR: Filename required!\n”);
return(1);
}
/*————————————————————————————
* Communication with Bootstrap Loader ...
*————————————————————————————
*/
/* Open COMx port (Change COM–port name to your needs!): */
if (comInit(comPortName, DEFAULT_TIMEOUT, 4) != 0)
{
printf(”ERROR: Opening COM–Port failed!\n”);
return(1);
}
bslReset(1); /* Invoke the boot loader. */
if (toDo.MassErase)
{
/* Erase the flash memory completely (with mass erase command): */
printf(”Mass Erase...\n”);
if ((error= bslTxRx(BSL_MERAS, /* Command: Mass Erase */
0xff00, /* Any address within flash memory. */
0xa506, /* Required setting for mass erase! */
NULL, blkin)) != 0)
{
return(signOff(error, FALSE));
}
passwdFile= NULL; /* No password file required! */
}
if ((error= txPasswd(passwdFile)) != 0)
{
return(signOff(error, TRUE)); /* Password was transmitted! */
}
#ifdef WORKAROUND
/* Execute function within bootstrap loader
* to prepare stack pointer for the following patch.
* This function will lock the protected functions again.
*/
printf(”Load PC with 0x0C22...\n”);
```

```
if ((error= bslTxRx(BSL_LOADPC, /* Command: Load PC */
0x0C22, /* Address to load into PC */
0, /* No additional data! */
NULL, blkin)) != 0)
{
return(signOff(error, FALSE));
}
/* Re–send password to re–gain access to protected functions. */
if ((error= txPasswd(passwdFile)) != 0)
{
return(signOff(error, TRUE)); /* Password was transmitted! */
}
printf("Load and verify patch...\n");
/* Programming and verification is done in one pass.
* "WAROUND.TXT" is only read and parsed once.
*/
if ((error= programTIText("WAROUND.TXT", /* File to program */
ACTION_PROGRAM | ACTION_VERIFY)) != 0)
{
return(signOff(error, FALSE));
}
#endif
if (!toDo.OnePass)
{
if (toDo.EraseCheck)
{
/* Parse file in TXT–Format and
* check the erasure of required flash cells.
*/
printf("Erase Check by file %s...\n", filename);
if ((error= programTIText(filename, ACTION_ERASE_CHECK)) != 0)
{
return(signOff(error, FALSE));
}
}
if (toDo.Program)
{
/* Parse file in TXT–Format and program data into flash memory. */
printf("Program %s...\n", filename);
if ((error= programTIText(filename, ACTION_PROGRAM)) != 0)
{
return(signOff(error, FALSE));
}
}
```

```
if (toDo.Verify)
{
/* Verify programmed data: */
printf("Verify %s...\n", filename);
if ((error= programTIText(filename, ACTION_VERIFY)) != 0)
{
return(signOff(error, FALSE));
}
}
}
else
{
unsigned action= 0;
if (toDo.EraseCheck)
{
action |= ACTION_ERASE_CHECK; printf("EraseCheck ");
}
if (toDo.Program)
{
action |= ACTION_PROGRAM; printf("Program ");
}
if (toDo.Verify)
{
action |= ACTION_VERIFY; printf("Verify ");
}
if (action != 0)
{
printf("%s ...\n", filename);
error= programTIText(filename, action);
if (error != 0)
{
return(signOff(error, FALSE));
}
}
}
return(signOff(ERR_NONE, FALSE));
}
/* EOF */
```

A.7 启动程序载入器(Bootstrap)TXT文件补丁——waround.txt

```
@0220
31 40 1A 02 B0 12 2A 0E B0 12 BA 0D 55 42 0B 02
75 90 12 00 11 24 B0 12 84 0E 06 3C B0 12 94 0E
03 3C 21 53 B0 12 8C 0E B2 40 10 A5 2C 01 B2 40
00 A5 28 01 30 40 42 0C 16 42 0E 02 17 42 10 02
```

E2 B2 08 02 14 24 B0 12 10 0F 36 90 00 10 06 28
B2 40 00 A5 2C 01 B2 40 40 A5 28 01 D6 42 06 02
00 00 16 53 17 83 EF 23 B0 12 9C 02 D7 3F B0 12
10 0F 17 83 FC 23 B0 12 9C 02 D4 3F 18 42 12 02
B0 12 10 0F D2 42 06 02 12 02 B0 12 10 0F D2 42
06 02 13 02 38 E3 18 92 12 02 C3 23 E2 B3 08 02
C0 23 30 41
Q

附录B PCB排版建议

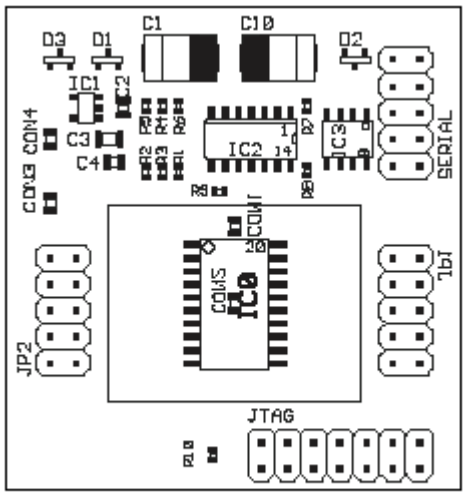
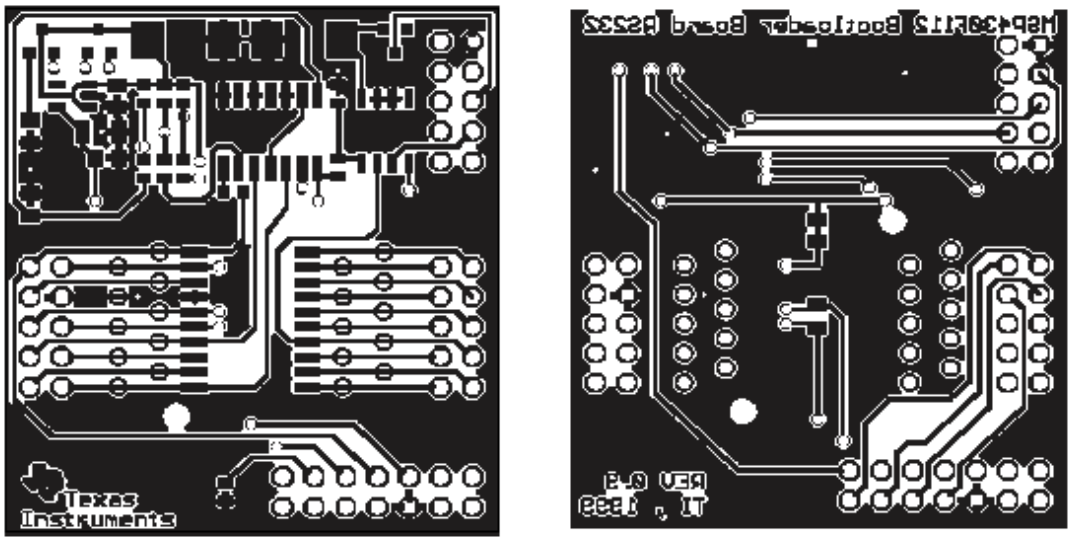


Figure B–1. Component Placement



Top Bottom

Figure B–2. PCB Layout

附录C 示范程序的使用方法

启动程序载入器(Bootstrap)有一个简单的命令行界面。仅需参数为要编程的TI TXT文件名。其它参数是可选的，但必须在TI TXT文件名之前进入。TI TXT文件名必须是最后的参数。

表C-1.命令行参数

PARAMETER	DESCRIPTION	EXAMPLE
–c{COM-port name}	Specifies the COM port to be used (default: COM1).	–cCOM2
–p{TI TXT-file name}	Specifies a TI TXT file containing the actual password to access the bootstrap loader.	–pint_vect.txt
–w	The program will wait after successful programming and reset for the <ENTER> key. The application can run powered via the serial port.	–w
–1	Do erase-check, programming, and verification in one pass (the TI TXT file is read and parsed only once). This option is discouraged with the first version(s) of the bootstrap loader that require the worka-round patch.	–1

除了这些命令行参数外，存在允许控制程序流的参数。（例如，可以校验flash memory的内容。）该参数由是由所发生步骤的说明（仅仅发生这些步骤）而产生的，由+字符引入。表C-2列出了所用到的修饰语。发生的步骤与表C-2中的修饰成分相同。

表C-2程序流修饰成分

MODIFIER	DESCRIPTION
e	Erase complete flash (mass erase)
c	Check erasure
p	Program given file
v	Verify against given file
r	Reset device
w	Wait for <ENTER> at the end

表C-3列出了一些示范程序调用的实例

表C-3调用实例

EXAMPLE	DESCRIPTION
bsldemo test.txt	Erase flash, check erasure, program and verify file test.txt, exit
bsldemo –1 –w –cCOM2 test.txt	Same as above, but the hardware is connected to COM2; erase-check, program, and verify are done in one pass through file test.txt; the program waits for <ENTER> at the end.
bsldemo +vrw –pint_vect.txt test.txt	Use data within file int_vect.txt as password, verify against file test.txt (no erasure or programming), reset MSP430, wait for <ENTER> at the end.
bsldemo +rw –pint_vect.txt test.txt	Reset MSP430 and wait for <ENTER> at the end. Password and file name are also required.