Texas Instruments

Application Report

SLAA452C–October 2010–Revised November 2014

USB Field Firmware Updates on MSP430™ MCUs

MSP430 Applications

1 Introduction

With the advent of USB, end users can perform firmware upgrades in the field by simply attaching the device via USB and executing an application on the host PC. Such an approach has numerous advantages:

- Problems discovered after product release can be fixed.
- Reduced need for high-touch support, because problems can often be solved by instructing the user to upgrade firmware.
- End users have a more positive experience with the product.
- Product returns can be reduced.

The USB solution for MSP430™ microcontrollers has been designed to make this process simple and straightforward. The device contains a USB-based on-chip bootstrap loader, and TI provides a Windows source project for downloading firmware that can be quickly customized. This source can be built with Visual C++ 2008 Express, available from Microsoft at no cost. For the USB support software for MSP430 MCUs, see the MSP430 USB Developers Package (MSP430USBDEVPACK).

This guide is intended for developers creating a basic USB firmware upgrade system. Those wishing to customize the update process or wanting a deeper look at the underlying bootstrap loader can consult *Creating a Custom Flash-Based BSL* (SLAA450) and *MSP430 Programming Via the Bootstrap Loader* (SLAU319).

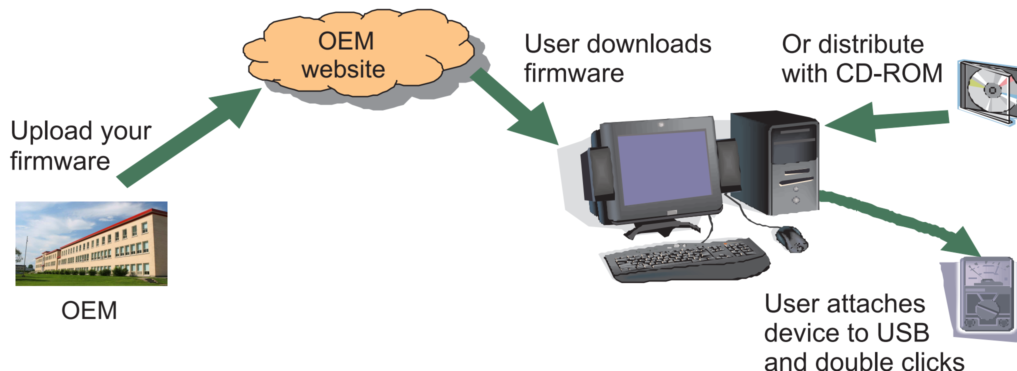

Figure 1.

MSP430 is a trademark of Texas Instruments.
All other trademarks are the property of their respective owners.

Copyright © 2010–2014, Texas Instruments Incorporated

2 MSP430 USB Field Firmware Update Process: Overview

MSP430 firmware upgrade over USB is implemented using the on-chip bootstrap loader (BSL) program. The BSL is located in a dedicated section of flash, outside of main program flash. TI has provided a BSL on almost every MSP430 device derivative, pre-programmed in the factory; but unlike other devices, the BSL on USB-equipped MSP430 devices uses USB as the hardware interface. And whereas previous BSLs are invoked with a special waveform on the I/O pins, the BSL on USB-equipped devices is invoked through one of the means shown in Table 1.

Table 1. Methods By Which the USB BSL Can Be Invoked

Method	When Used
Blank RESET vector If the RESET entry in the vector table is blank (0xFFFF) after a BOR reset (see below), the BSL is invoked.	Initial device production. (After initial programming, the vector is no longer be blank.)
Software If the user application transfers control of execution to the BSL (jump to 0x1000), the BSL is invoked.	Firmware upgrades in the field, initiated by the end user.
Hardware If the PUR pin is held high by external circuitry while a BOR reset occurs on the MSP430, the BSL is invoked. Often this involves the end user pressing a button while re-inserting the battery (depending on the application).	Mostly used as a failsafe, in the rare event that the other two methods cannot be used to recover from an interrupted update session. It can also be used as a primary means of invocation.

The software method is recommended as the primary means of entry when performing field upgrades, because it usually provides the best experience for the end user. The other two methods can serve backup roles for failsafe operation.

Practical ways to implement these in a system are discussed in Section 4. A high-level overview is shown in Figure 2.

Copyright © 2010–2014, Texas Instruments Incorporated

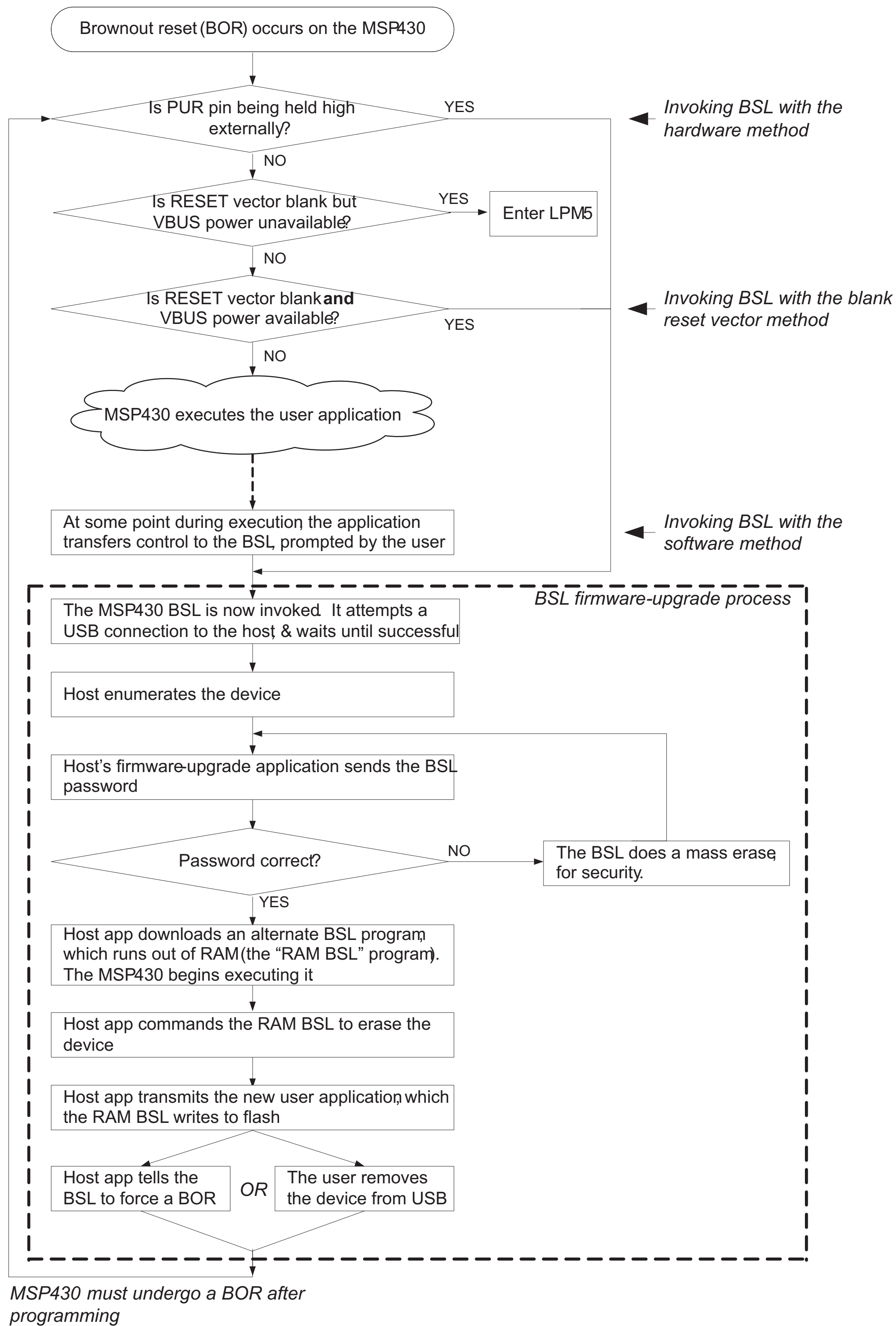

Figure 2. System-Level Description of MSP430 USB Firmware Updates

Copyright © 2010–2014, Texas Instruments Incorporated

The Windows source project accompanying this document performs all the host functions shown in the flowchart in Figure 2 and can be customized as needed. All other actions that the engineer must perform are described in this document.

3 Before Implementation: System Considerations

This section addresses topics the developer should understand before beginning development.

3.1 What Host Drivers or Applications are Required?

No special kernel-mode drivers are required on the host. This is because the MSP430 reports itself as a Human Interface Device (HID), which has built-in Windows support. HID is a great fit for firmware updates, for these reasons:

- It has perhaps the widest native support among host operating systems. In contrast, other update systems might require a kernel driver installation, which adds a risk of the user experiencing problems if something goes wrong.
- It loads silently onto the host, which means that no user intervention is required for "installation" of the device after it is attached.
- HID throughput (64 KB/sec) fits well with MSP430 range of memory sizes, both in existing devices and future ones. Downloading firmware finishes quickly.

HID is easier for the end user, and this can translate to less support and less development cost for the OEM.

A host user-mode application is required to drive the download process. This is what the accompanying source project creates.

3.2 Generation of a Brownout Reset (BOR)

As Section 2 indicated, two of the three invocation methods involve a BOR event. A BOR is also necessary for the BSL to hand control back to the new user application. Although the name BOR includes the word "brownout", an actual power brownout event is only one means by which a BOR can be triggered. BOR is simply a name for the deepest reset an MSP430F5xx device can undergo.

These events can generate a BOR:

- Initial powerup of the device
- An edge on the RST pin (perhaps with a dedicated button)
- Setting the PMMSWBOR bit with software
- Removing the device from USB while the BSL is in control of the MSP430
- A brownout event on DVCC
- Other events [see the SYS Module chapter of the *MSP430x5xx/MSP430x6xx Family User's Guide* (SLAU208)]

BOR events are a critical part of the BSL flow if the invocation method is the hardware or blank RESET vector method. Some user-level events might generate a BOR "automatically"; for example, when the user removes the device from USB while the BSL is in control, or if the device is purely bus-powered and the user attaches it to the host, causing a power-up event. In other cases, the user might need to be given special instructions to cause it; for example, with a special reset button, or by removing the battery prior to attaching it to USB (causing a power-up event when attached). The system designer must ensure the end user does what is necessary to cause the BOR at the proper time.

3.3 VID/PIDs in the USB BSL

Any USB device contains a pair of 16-bit values called a vendor ID (VID) and product ID (PID). The VID is associated with the OEM, and the PID is associated with a product sold by that vendor. A unique combination of a VID and PID allows a USB host to identify one USB product type from another.

Copyright © 2010–2014, Texas Instruments Incorporated

The BSL is considered a separate USB device from any that might exist in the main user application. Therefore, it has a VID/PID. If a USB function is also in the user application, that function has its own VID/PID. Having different VID/PIDs is what allows the host to differentiate whether the user application is in control of the device or the device is making itself available for firmware update.

The VID/PID implemented by the default USB BSL is 0x2047/0x0200. 0x2047 is a TI-owned VID, and 0x0200 is a PID unique to the BSL for the 'F552x/'F550x/'F663x device families.

If it is necessary to change the BSL VID/PID to a different unique value, the change must be made in three locations:

- The host application, built by the Visual Studio project
- The on-chip flash BSL
- The RAM BSL that is downloaded from the host application

The values for the host application are located in the source file Firmware.resx (see Section 6.2.1). The other two items require recompiling the BSL source, which is provided with the application report *Creating a Custom Flash-Based BSL* (SLAA450). After modifying the VID/PID in BSL_Device_File.h, the BSL must be compiled in two different ways: as a flash BSL and as a RAM BSL. The flash BSL image must be downloaded into every target MSP430 device. The RAM BSL image must be pasted into Firmware.resx. When all three components contain the new VID/PID, the system functions in the same way as if the defaults are used.

3.4 BSL Password

When the host attempts to interface with the device's BSL, the BSL requires a password. This is the same password mechanism used in non-USB MSP430 BSL versions. The password consists of the device's memory contents at addresses 0xFFE0 to 0xFFFF that exist at the time of BSL access. This range is a subset of the vector table. For more information on the password and security measures of the BSL, see *MSP430 Programming Via the Bootstrap Loader* (SLAU319). After the correct password is entered, the firmware can be upgraded. If the password is incorrect, the BSL automatically performs a mass erase of the device's entire main flash area.

Some applications store data in flash during normal operation that needs to be preserved after the update. Developers of these applications should always report the correct password to avoid the mass erase. It is recommended to ensure the password never changes during the product lifetime by fixing the locations of the ISRs (whose addresses comprise the BSL password) to consistent locations in memory. If this is not done, then the password will likely change after each revision; allowing this to happen makes it necessary to upgrade the versions in succession (v1, v2, v3, etc.) to ensure that the right password is used each time, if data contents are to be preserved. It is usually difficult to ensure that the user will do this.

Fixing the ISR locations solves this problem. It can be done during development of the MSP430 application using a variety of methods. One is to create a dedicated linker segment for each ISR with a fixed address and assign the ISR functions to those segments.

However, some applications do not need to preserve data after the update, which means that a mass erase causes no harm. These might be better served by reporting a blank (that is, invalid) password. The resulting mass erase means that the host always knows the correct password on the second try: all 0xFFFF. This method is easier than reporting the correct password, because it avoids the planning process described previously.

If the correct password is used, the Visual Studio application automatically uses segment erases that cover only the flash segments needed to store the new application, leaving the rest intact. If the incorrect password is used, it is assumed that a mass erase already took place, and no additional erases are performed.

3.5 Distributing Firmware Updates to the Field

The updates can be distributed via the OEM's website, a CD-ROM, or other methods. To create new host applications for each revision, it is only necessary to replace the firmware image (as described in Section 6.2) in the source project and rebuild it.

It should be noted that the MSP430 firmware within the host application is not encrypted. If encryption is desired, a custom on-chip BSL can be written that decrypts the image after transmission from the host.

Copyright © 2010–2014, Texas Instruments Incorporated

3.6 Interruption of the Upgrade Process

When upgrading firmware in an MCU, there is always a chance that the process could be interrupted, potentially corrupting the code. Updates conducted over USB are more susceptible to this, because the end user is able to remove the USB cable at any point in the process.

The MSP430 firmware update system has two layers of security to assist in recovering from such an event. One is to maximize the time in which the reset vector is blank. The "blank reset vector" method ensures that a device reset under this condition invokes the BSL, allowing recovery simply by re-attempting the upgrade. The host application accompanying this document ensures that the vector table is the first segment erased and the last segment written. The only remaining window of vulnerability is during the erase, which takes approximately 25 ms (see the device data sheet for specifics).

The probability that the user will remove the USB cable during this vulnerability window might be calculated as the time for the erase (approximately 25 ms) divided by the overall update session (for example, 25 seconds). With these assumptions, there would be one unrecoverable session in a thousand interrupted sessions. The vast majority of users will not remove the cable, making the odds more remote. In the same scenario, if 5% of users remove the cable during the upgrade, then about one in 20,000 upgrades will experience a corruption that is unrecoverable by the reset vector method.

In these failed cases, recovery of the device would require the hardware method (or JTAG). The hardware method will always succeed at reprogramming the device in the field, with the tradeoff being the cost of implementing a button. If the odds of irrecoverable failure with the reset vector method are too high, a BSL button should be implemented (see Section 4.5).

3.7 USB PLL Reference Clock

The MSP430 USB module requires a clock to serve as a reference for the USB PLL. The clock can either be generated by the XT2 oscillator by applying a crystal, or a clock source from elsewhere on the board can be applied to XT2 in bypass mode. In the former case, the minimum frequency is 4 MHz; in the latter, it is 1.5 MHz (see the device data sheet for parametric details). The PLL is programmable and can accept a wide variety of frequencies (see the *MSP430x5xx/MSP430x6xx Family User's Guide* (SLAU208) for architectural details).

The on-chip BSL needs to know the reference frequency to configure the PLL. By default, it is able to detect the presence of four frequencies on XT2: 4 MHz, 8 MHz, 12 MHz, or 24 MHz. If it finds one of these, it programs the PLL accordingly. If any other frequency is be used, the BSL must be modified and reprogrammed in every device (see Section 6.1).

3.8 Download Speeds

The download time for even the largest firmware images is short. Example programming times are shown in Table 2.

Table 2. Programming Times

Image Size	Mass Erase	Segment Erases
32 KB	8 seconds	8 seconds
82 KB	15 seconds	18 seconds
180 KB	22 seconds	28 seconds

Notice that speed is somewhat influenced by whether the MSP430 flash is erased by segments or in mass. A mass erase takes the same amount of time as each individual segment erase does, so performing multiple segment erases adds time to the process. However, the difference is quite small. The accompanying Visual Studio project performs segment erases if the BSL password succeeds. If the password fails, it does not attempt segment erases, because it knows that a mass erase has already occurred (see Section 3.4). As a result, using the right or wrong password affects programming times.

Because the MSP430 firmware updates are performed with the HID device class, firmware upgrades have guaranteed bandwidth on the bus. This makes them unaffected by heavy bulk traffic.

Copyright © 2010–2014, Texas Instruments Incorporated

4 Field Firmware Upgrade: System Models

Five ways of implementing firmware upgrade in the field are described. Each represents a different way the end user can experience the update process. They also vary to some degree in the amount of development effort required.

Table 3 shows models for normal update sessions, and Table 4 shows failsafe models (for recovery from an interrupted session). Under most circumstances, one "normal" model should be chosen, the reset vector failsafe model should be planned for, and the upgrade button failsafe model should be considered as described in Section 4.5.

Table 3. Primary Usage Model Options

Application Characteristics	How the User Would Invoke the BSL	Described Where
The user application does not contain a USB function (in other words, USB is only used for firmware update)	BSL is invoked any time the user attaches the device to a USB host	Section 4.1
The user application contains its own USB function (in other words, USB is used for two purposes – in the user application and firmware update)	User accesses the device's user interface to enter a special "update mode"; once in this mode, the BSL is invoked any time the user attaches the device to a USB host	Section 4.2
The user application contains its own USB function, and the above method is not possible, or it is desired to invoke the BSL from host control	With the device already attached to the host for the normal USB function, the user accesses a host application to update firmware	Section 4.3

Table 4. Failsafe Usage Models (For Recovering from an Interrupted Session)

Application Characteristics	How the User Would Invoke the BSL	Described Where
The interruption occurred while the reset vector was blank (true of most interrupted sessions)	User attaches the device to any USB host and then generates a BOR.	Section 4.4
The interruption occurred while the reset vector was not blank (rare case)	User performs an action that pulls the PUR pin high while generating a BOR (probably using one or two dedicated switches)	Section 4.5

Copyright © 2010–2014, Texas Instruments Incorporated

4.1 If the User Application Does Not Contain a USB Function

This is the simplest model. A USB interrupt is generated when the device is attached to a USB host. If there is no other USB function, it is reasonable for the application to interpret this as the user attempting to update firmware. Code can be placed in the USB ISR (if USBVECINT points to a VBUS-on event) to invoke the BSL using the software method:

```
void main(void)
{
  volatile unsigned char fInvokeBSL = 0;

  WDTCTL = WDTPW + WDTHOLD;     // Stop watchdog timer -- not used in this example

  // Activate the sensing of 5V VBUS on the USB cable
  USBKEYPID   =  0x9628;        // Unlock the USB registers
  USBPWRCTL   |= VBONIE;        // Enable the VBUS-on interrupt
  USBKEYPID   =  0x9600;        // Lock the USB registers

  while(1)
  {
    __bis_SR_register(LPM3_bits + GIE); // Go into LPM3; keep ints enabled.

__  disable_interrupt();        // Ensure no application interrupts fire during BSL
    if(fInvokeBSL)
      ((void (*)())0x1000)();   // This sends execution to the BSL.  When execution
                                // returns to the user app, it will be via the reset
  }                             // vector, meaning execution will re-start.
}

#pragma vector=USB_UBM_VECTOR   // USB Interrupt Service Routine
__interrupt void iUsbInterruptHandler(void)
{
  switch (__even_in_range(USBVECINT & 0x3f, USBVECINT_OUTPUT_ENDPOINT7))
  {
    case USBVECINT_PWR_VBUSOn:
      fInvokeBSL = 1;                        // Set flag
      __bic_SR_register_on_exit(LPM3_bits);  // Exit LPM3 when exiting the ISR.
      break;

    default:
      break;
  }
}
```

Note that the BSL is not invoked directly from within the ISR. It is best to return from the ISR, and then invoke the BSL.

Whenever the device is removed from USB while the BSL is in control, the BSL jumps to the reset vector. This puts the user application back into control.

If the user attaches to the USB host errantly, the BSL is invoked, but unless a host application tries to interface with it, the device should not experience any negative effects. Once removed from the host, a BOR reset occurs, and code executes from the beginning. While the device is attached to USB with the default BSL in control, buttons are not monitored, and no display elements are driven, so the device might appear to be "dead" from the user's perspective during this time.

Copyright © 2010–2014, Texas Instruments Incorporated

4.2 If the User Application Does Contain a USB Function

This model is exactly the same as described in Section 4.1, except that the user must put the device into a special mode before a USB attachment triggers a BSL invocation. For example, the user might use a menu system on an LCD display or might be required to press and hold two buttons for five seconds. Once in this mode, any USB attachment causes the application to invoke the BSL using the software method.

The implementation is similar to the method described in Section 4.1, with the addition of code to evaluate the mode. If one of TI's MSP430 USB API stacks are being used, it is recommended to place this code in the handleVbusOnEvent() handler, rather than directly in the USB ISR:

```
BYTE USB_handleVbusOnEvent()
{
  if(inFWUpdateMode)
  {
    fInvokeBSL = 1;                   // Set flag triggering main() to invoke BSL
  }
  else
  {
    if(USB_enable() == kUSB_succeed) //Connect to USB, for user application function
    {
        USB_reset();
        USB_connect();
    }
  }
  return TRUE;
}
```

fInvokeBSL can then be evaluated in the main loop to determine whether or not to jump to 0x1000 in the manner shown in Section 4.1.

4.3 If Host Control is Desired

This model can be used only if there is a USB function in the user application in addition to the USB function that is used for firmware update. It is presumed that a host application program already exists for communication with the user application to carry out this normal USB function. The firmware upgrade process then becomes a feature within this application, rather than being a standalone application.

In this model, the device is already attached to the USB host and, therefore, enumerated under the control of the user application. The following procedure takes place:

1. The user selects a control in the host application program to update the firmware.
2. The host application sends an application-specific command to the device telling it to invoke the BSL.
3. The MSP430 user application recognizes the command. It disconnects from USB, disables general interrupts, then uses the software method to invoke the BSL (jumps to 0x1000).
4. Under BSL control, the device re-enumerates on the USB bus.
5. The host application recognizes that the BSL device has enumerated. It then uses BSL commands to update the firmware in the same manner described previously.
6. The host application issues a BSL command that instructs the device to issue a BOR. This time the device enumerates under control of the new user application.

From the end user's perspective, this entire process is triggered simply by pressing a button or selecting a menu item in the host application program.

Copyright © 2010–2014, Texas Instruments Incorporated

Implementation of this model is more application-specific than the previous methods. It requires that a command be placed within the USB traffic of the user application by which the host application can tell the device to invoke the BSL. This requires changes to both the host and MSP430 applications: the host application must essentially integrate the accompanying Source Project, and the MSP430 application must recognize the command to invoke the BSL.

4.4 Recovering From an Interrupted Update Session When the Reset Vector is Blank

In the majority of interrupted update sessions, the interruptions occurs while the reset vector is blank. This ensures easy recovery (see Section 3.6).

If a BOR occurs while the reset vector is blank and if the device is already attached to an active USB host (5-V VBUS is present), then the BSL is invoked. VBUS must be present when the BSL is invoked, otherwise the BSL enters an invalid state and another BOR is required.

To ensure the BOR happens (see Section 3.2), it might be necessary to provide the end user with special instructions that cause a BOR. For example:

- Remove power from the device, then restore it after the device has been attached to USB (a power-up event generates a BOR).
- Push a button that causes an edge on the MSP430 RST pin (for example, a "paperclip switch" on the back of the device).

The user interface (buttons/display) should not be replied upon for this, because the application firmware is presumed corrupted at this point and unable to process the interface.

If the device receives power from the USB, it might be necessary only to remove the battery and attach to the host. Upon attachment, the device powers up and VBUS is present; the two conditions would be met and, therefore, the BSL would be invoked.

If the device receives power only from a battery and not from USB, the device could be attached to USB and then the battery re-inserted.

If the device has a rechargeable battery, a reset switch is probably required. This is because most such systems do not have removable batteries.

Although unlikely, the interrupted session might result in the reset vector being corrupted (not blank). Under this condition, a reset results in unpredictable execution. In this situation, only the method in Section 4.5 or JTAG can recover the device.

4.5 Recovering From an Interrupted Update Session Using a "Firmware Update Button"

This model always works, no matter the contents of user flash. It can be used as either a backup to one of the other methods or as a primary method of performing updates.

As described in Section 3.6, the probability of a failure that needs this model for recovery is quite low. If the engineer is willing to accept the risk, he or she could choose to omit this method. If absolute recovery is needed, this model can be implemented.

In this model, a button that forces a pullup on the PUR pin is implemented. The user holds this button and then causes a BOR on the MSP430. In this way, it invokes the BSL using the hardware method. This button must be dedicated to this function; it cannot be part of the device's normal user interface. An example might be a "paperclip switch" underneath the battery cover.

While the button is held down, a BOR must be generated, possibly using one of the methods described in Section 4.4.

An example procedure is described below. It assumes an application in which a "firmware upgrade button" is implemented using a paperclip switch in the battery compartment and in which the device is able to operate from the USB host in the absence of a battery:

1. User removes the battery. This also exposes the switch.
2. User keeps the button held down while attaching to USB. Note this might be difficult to do on a small handheld device using two hands.
3. The device enumerates under BSL control; the user can now release the button.

Copyright © 2010–2014, Texas Instruments Incorporated

4. The host updates the firmware in the same manner described previously in this document.

Disadvantages of this model are the cost of the switch and, in some applications, the possibly awkward process for the end user to hold down the button while causing a reset.

An example circuit that can be used to implement this method is shown in Figure 3 (the TS430PN80USB FET target board uses this circuit).

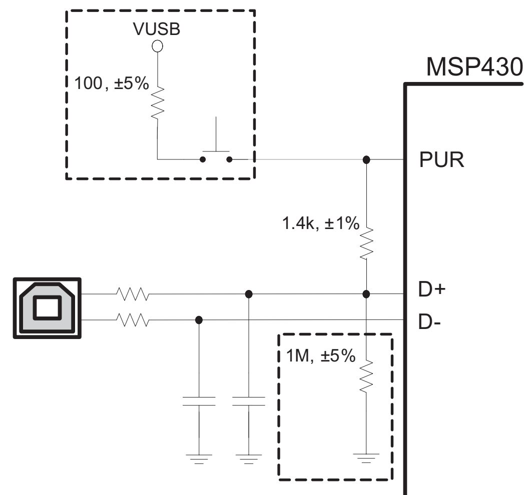

Figure 3. Example Circuit for Hardware Invocation Circuit

Prior to software driving PUR high to signal USB attachment, the 1-MΩ pulldown resistor keeps PUR low when the switch is not being pressed. After software drives PUR high, the pulldown is weak enough not to affect the operation of D+.

Copyright © 2010–2014, Texas Instruments Incorporated

5 MSP430 Firmware Update Source Project

The source project accompanying this document can be built to provide a firmware update host application. It is designed for quick adaptation to most applications, with minimal effort. For the USB support software for MSP430 MCUs, see the MSP430 USB Developers Package (MSP430USBDEVPACK).

This section describes the source project. Section 6 includes specific steps on how to customize it.

5.1 System Requirements

The setup installer file accompanying this document contains both source and object code.

The source project is designed for use with Visual Studio C++ 2008, Full or Express Editions, using the .NET framework. The Express Edition can be obtained from Microsoft at no cost.

When running the application, the .NET framework v3.5 or later must be available on the user's system.

5.2 Using the Program

When opening the firmware update program, the welcome view is shown (see Figure 4).

Figure 4. Welcome View

Copyright © 2010–2014, Texas Instruments Incorporated

The user is asked to accept an end user license agreement (EULA) (see Figure 5).

Figure 5. EULA View

Accepting the license leads to the download view (see Figure 6).

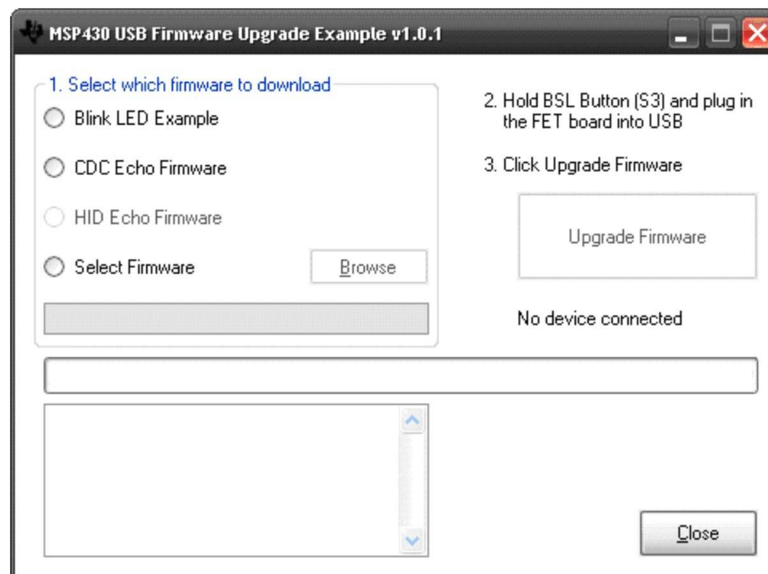

Figure 6. Download View

When the GUI detects the presence of an MSP430-based device under the control of the BSL (that is, a device for which the BSL has been invoked, as defined by its VID/PID) the "Upgrade" button is activated. When the button is clicked, the USB BSL is initiated and the firmware selected with the radio buttons is downloaded. Several default programs are provided, and there is also an option to select an external file. The file selection must be of format msp430-txt, which can be generated by standard MSP430 IDEs (in CCS, this format is called TI-TXT).

Copyright © 2010–2014, Texas Instruments Incorporated

The download view's instructions say to hold down button "S3". This refers to a button on the TS430PN80USB FET target board, which is implemented in the manner discussed in Section 4.5. Because the FET board has no local power source and thus easily generates a BOR upon attachment to USB (a power-on event), and because the BSL button is easily accessible on the FET board, the BSL button approach provides an easy way for the developer to experience the process.

5.3 Source

5.3.1 Project Structure

Two projects are provided, bound within a single Visual Studio 2008 "solution" file.

- BSL_USB_GUI: the Graphical User Interface (GUI) project viewed by the user. It makes calls to the file BSL430.DLL to communicate with the MSP430 device's BSL.
- BSL430_DLL: the project that generates BSL430.DLL at compile time.

The solution file to be opened is BSL_USB_GUI_wBSL430DLL.sln. It contains all necessary preset settings linking the two projects together.

If the user does not want to recompile the BSL430_DLL project, a solution file (BSL_USB_GUI.sln) is provided to recompile the GUI with a prebundled BSL430.DLL file.

Figure 7 shows the expected project view when opening the solution for the first time.

Copyright © 2010–2014, Texas Instruments Incorporated

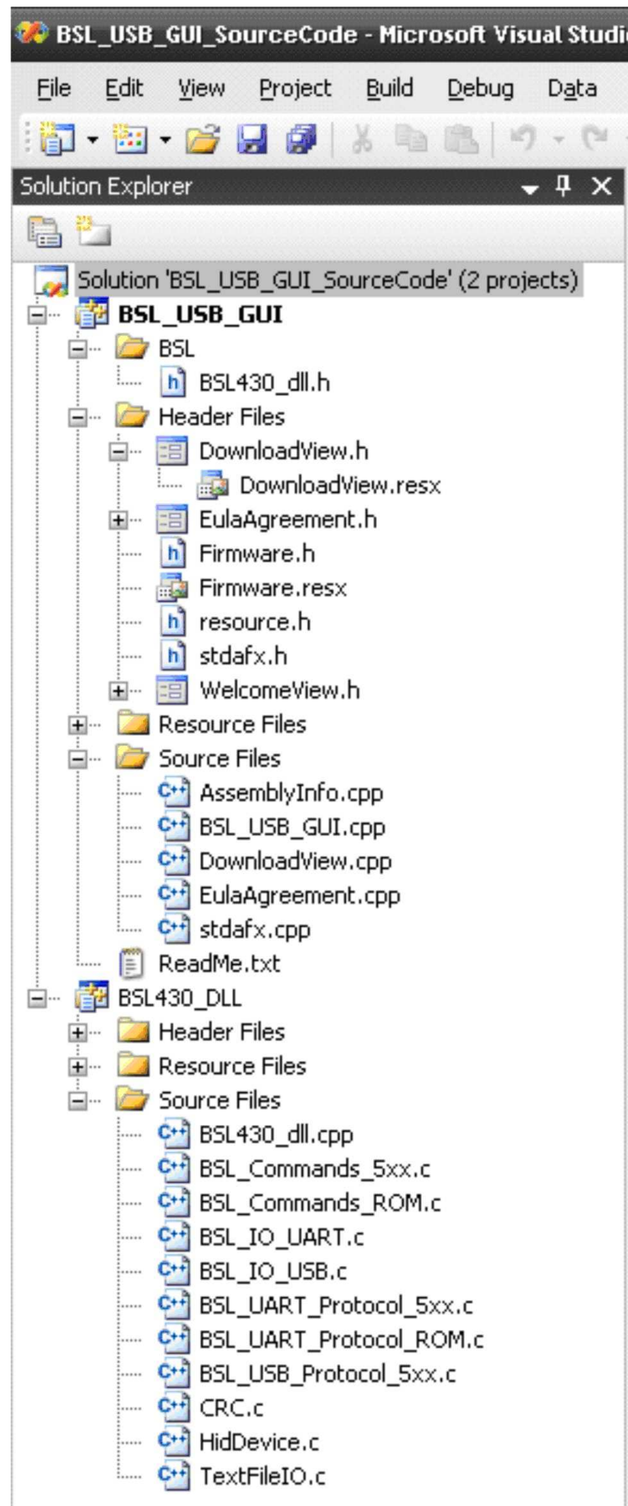

Figure 7. Default Project View Structure

Copyright © 2010–2014, Texas Instruments Incorporated

5.3.2 Selecting Compilation With Full or Express Editions

After opening the solution file, the user must select either Full or Express edition. The only feature not available when using Express is the mechanism for tracking versions of the built application. Selecting the wrong version for Visual Studio Express Edition install generates an error.

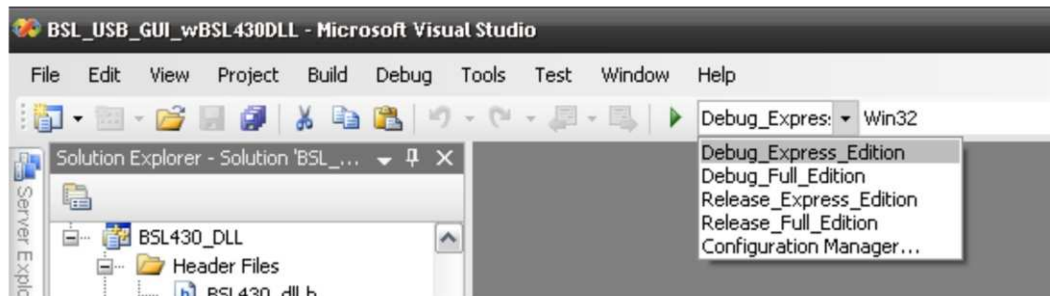

Figure 8. Selecting Compilation With Visual Studio Full or Express Edition

5.3.3 Code Organization

Each view has *.cpp/h files containing code associated with it. Most of the code related to firmware download is in DownloadView.cpp/h. The functions in these files interface with BSL430.DLL to download the firmware.

The download process is threaded separately so that the GUI is always responsive.

The default firmware options (shown as the radio buttons) all have corresponding firmware images contained within the GUI. These are all of format msp430-txt. Also, each option has its own set of functions to download that image, linked to the radio buttons.

The code in this file registers the USB device according to its VID and PID, so that Windows notifies the view if the device is added/removed from the system. It is in this way that the "Upgrade" button is activated/deactivated.

5.3.4 Icons

The icons for this project are stored in app.rc, under the project's "Resource Files" folder.

5.3.5 Version/Company Information

This project also maintains version, company, and product information in app.rc, under VS_VERSION_INFO. This file stores the data that is used by Windows to store a file's properties from version number to manufacturer information.

Version information is also located in AssemblyInfo.cpp. This file contains the managed version of company's information and version number, which are used by the application to display the version number on the GUI.

5.4 Threading

Downloading, verifying, and scanning for new devices can take up to a few seconds, having the potential to make the GUI unresponsive. To prevent this, two threads are maintained. The .NET framework's BackgroundWorker class offers an easy means of doing this. The two threads are:

- backgroundWorker1

  This is the main thread, which downloads BSL firmware into the MSP430. Only one image can be downloaded at a time. It is within the function backgroundWorker1_DoWork()that the radio buttons are evaluated, and a call to the corresponding download function is made.
- backgroundWorkerCheckDevice

  Scans for USB devices connected to the system, based on a specific VID/PID. This thread can be used only if backgroundWorker1 is not busy.

Copyright © 2010–2014, Texas Instruments Incorporated

Because these threads access the same DLL resource, only one can run at a time. Each has start-process, progress-update, and end-thread functions that allow control of each step in the process. The start process thread calls the BSL download functions according to the selected firmware image. The progress update is called inside the thread during the download process to report changes to the progress bar and status box. The end thread re-enables or disables the "Upgrade" button.

To pass variables between threads, it should be properly handled so that no memory access violation occurs.

Copyright © 2010–2014, Texas Instruments Incorporated

6 Putting It in Practice: Implementing USB Device Firmware Updates

Here is a summary of steps required to create an MSP430 USB field firmware update system:

1. Choose a usage model (see Section 4).
2. In the MSP430 user application, place the necessary code to invoke the BSL, according to the chosen usage model.
3. If implementing a "firmware upgrade button", add the required circuitry to the board (see Section 4.5).
4. If necessary, customize the clock frequency used in the MSP430 BSL source code, and download it to each MSP430 device (see Section 6.1).
5. Customize the provided host application source project (see Section 6.2).

Steps 1 to 3 have already been discussed. Steps 4 and 5 are discussed in Section 6.1 and Section 6.1, respectively.

6.1 Customize the XT2 Clock Frequency in the MSP430 BSL Code

If the clock that is applied to the MSP430 device's XT2 oscillator is any frequency other than one of those that the BSL is able to auto-detect (4 MHz, 8 MHz, 12 MHz, or 24 MHz), then it is necessary to customize the value in the BSL and download this new BSL image to each MSP430 device. Without this, USB cannot function properly, and the BSL does not enumerate on the host. This is because XT2 provides the reference clock to the USB PLL.

The process of customizing this is described below. Note that if one of the four auto-detect frequencies is used, this step can be eliminated.

Note that the default BSL does not support XT2 bypass mode, in which the crystal is replaced by an external clock fed into XT2. To use the bypass mode, the clock configuration code in the BSL source must be customized—see the application report *Creating a Custom Flash-Based BSL* (SLAA450).

To modify the XT2 frequency, the BSL source code must be modified and downloaded to each device. The BSL's source has been arranged to make this simple and straightforward for those who are making only this small modification.

The BSL source is available as part of the application note *Creating a Custom Flash-Based BSL* (SLAA450). This source comes in the form of IAR projects; developers not already using IAR can download IAR KickStart from TI's website at no charge. The BSL fits within KickStart's size limitation.

The BSL source has several constants that must be modified to reflect the new values. The constants are located near the top of the file BSL_Device_File.h.

Table 5. Customizing XT2 Frequency in the BSL Source

Constants	Description
SPEED_1 SPEED_2 SPEED_3 SPEED_4	Values related to the BSL's frequency auto-scan feature. The number is equal to the frequency in Hertz. It is recommended to change all four to the target value.
SPEED_1_PLL SPEED_2_PLL SPEED_3_PLL SPEED_4_PLL	Values related to the BSL's frequency auto-scan feature. These values correspond one-for-one with SPEED_x above. The value assigned to these constants should be one from the MSP430 header file of format USBPLL_SETCLK_xx_y. It is recommended to change all four to the target value.

Once the modifications are made, the program can be built and downloaded into a device using IAR and an emulation tool like the FET430UIF FET tool. Once performed, development can proceed as normal with a non-BSL project on this device, and the BSL contents remain updated.

A different approach is needed for programming devices in production. A single msp430-txt image file is needed that contains both the BSL object code and the main application object code. Production programmers can then use this file to update each device. This msp430-txt file must be created manually by concatenating the application to the end of the customized BSL image. The following example demonstrates this process.

Copyright © 2010–2014, Texas Instruments Incorporated

First, edit the XT2 frequency in the BSL's IAR source project as described earlier, and output an msp430-txt file for the BSL. It appears similar to the sample below when viewed with a text editor. Note that the starting address is 0x1000, the location of the BSL in MSP430F5xx memory.

msp430-txt file for the BSL

```
@1000
2C 3C 06 3C FF 3F FF 3F FF 3F FF 3F FF 3F FF 3F
3D 90 AD DE 04 20 3E 90 EF BE 01 20 03 3C 0C 43
.
.
.
0F 4C 0F 5D 03 3C CC 43 00 00 1C 53 0C 9F FB 23
10 01 80 00 76 44 80 00 7A 44 FF 3F
@FFFE
00 44
q
```

Now output the msp430-txt file of the main application. This code is positioned higher in memory.

msp430-txt file for the user application

```
@4400
31 40 00 44 3C 40 00 24 3D 40 01 00 B0 13 60 44
.
.
0F 4C 0F 5D 03 3C CC 43 00 00 1C 53 0C 9F FB 23
10 01 80 00 76 44 80 00 7A 44 FF 3F
@FFE6
18 44
@FFFE
00 44
q
```


Copyright © 2010–2014, Texas Instruments Incorporated

The two should then be combined with a text editor. Open the BSL output and delete anything in the range of the interrupt vectors (0xFF80 to 0xFFFF), as well as the "q" character (which denotes the end of the file). Then, append the application's output. When combined, they should look like:

msp430-txt file for the two combined

```
@1000
2C 3C 06 3C FF 3F FF 3F FF 3F FF 3F FF 3F FF 3F
3D 90 AD DE 04 20 3E 90 EF BE 01 20 03 3C 0C 43
.
.
.
3B 00 00 00 1F 42 44 1C CF 4C 01 00 E2 43 00 1C
34 21 78 56 80 00 3A 17 21 83 B1 40 4C 04 00 00
02 3C B1 53 00 00 81 93 00 00 FB 23 21 53 18 02
B2 40 00 51 CE 03 92 B3 CE 03 FD 2B 10 01 1D 15
10 01
@17F0
FF FF 2A 10 A5 3C 5A C3 FF FF 00 10
@17FC
FF FF FF FF
@4400
31 40 00 44 3C 40 00 24 3D 40 01 00 B0 13 60 44
.
.
.
0F 4C 0F 5D 03 3C CC 43 00 00 1C 53 0C 9F FB 23
10 01 80 00 76 44 80 00 7A 44 FF 3F
@FFE6
18 44
@FFFE
00 44
q
```

The combined file can then be downloaded into the MSP430 device using a production programmer. Note that a "combined" msp430-txt file (application plus BSL) cannot be downloaded with the Visual Studio project. This device is now prepared to function with a USB host firmware upgrade application configured to recognize this VID/PID. Preparing this application is discussed in Section 6.2.

Copyright © 2010–2014, Texas Instruments Incorporated

6.2 Modifying the Source Project

This section assumes basic skills with Microsoft Visual Studio.

The basic steps are shown here, and the sections that follow provide detail to these steps.

1. Replace the "blink LED" MSP430 firmware image with the new target image.
2. Customize the BSL password (see Section 3.4).
3. Customize the EULA agreement.
4. Customize the GUI's look and feel. Customize the display strings, graphics, etc.
5. Build the application.

6.2.1 Steps 1-2: Customize the Firmware and BSL Password

These are accomplished in a single location: Firmware.resx, located within the GUI project. Double-clicking on this resource file shows a table of strings. The contents are shown in Table 6.

Table 6. Firmware.resx Contents

Field	Description
5529_LED_Blink.Text	MSP430 object code for an "LED blink" application in msp430-txt format.
CDC119Echo.Text	MSP430 object code for an "echo" application that enumerates as a CDC device in msp430-txt format. Allows communication with the host via HyperTerminal.
Forced_PUC.Text	MSP430 object code that forces a powerup clear (PUC) reset. This should not be changed.
Forced_BOR.Text	MSP430 object code that forces a brownout reset (POR). This should not be changed.
HID119Echo.Text	MSP430 object code for an "echo" application that enumerates as a HID device in msp430-txt format.
password.Text	The BSL password in msp430-txt format (see Section 3.4).
RAM_BSL.Text	MSP430 object code for the RAM BSL. TI may release new versions of the Source Project that contain newer versions of the RAM BSL. This should not be changed by the developer.
RAM_BSL_VERSION	Versioning for the RAM_BSL field (see previous entry). **NOTE:** If the developer updates the RAM_BSL and setting to a different version, the RAM_BSL_VERSION should also be updated.
USB_VID	The target's vendor ID (VID). The application associates only with a USB device that has this VID.
USB_PID	The target's product ID (PID). The application associates only with a USB device that has this PID.

Assuming that only a single firmware image is to be downloaded, the following is recommended:

1. Paste the contents of the new target application's msp430-txt file into the value field for "5529_LED_Blink.Text".
2. Eliminate the controls from the GUI that allow selection of firmware; for example, the radio buttons.
3. Eliminate every part of the code that references the eliminated controls. As part of this, the function backgroundWorker1_DoWork(), which is called by the "Upgrade" button, should only call DownloadBlinkLED(), not the other download functions.

This approach uses the "blink LED" program as a container for the new target firmware image and eliminates the program's ability to use any image other than this one.

As discussed in Section 3.4, the password can either be left as the blank password or updated to the correct password. The correct password can be obtained from the msp430-txt file of the firmware image already in the device, addresses 0xFFE0 to 0xFFFF. Note that the "@0xFFE0" address header must remain present in the "password.Text" field, followed by all 32 bytes through 0xFFFF.

Copyright © 2010–2014, Texas Instruments Incorporated

6.2.2 Step 4: The EULA

An OEM should use its own end user license agreement text, because the default one has TI-specific phrasing. The text can be found in the file EulaAgreement.resx.

6.2.3 Step 5: Customizing the GUI's Look and Feel

It is up to the OEM's discretion how to package the GUI.

6.3 Building and Distributing the Application

The .NET framework v3.5 or later must exist on any PC running the application. Without it, the application cannot execute. Although most Windows users today already have this installed, some may not. One approach is to create an installer for the firmware upgrade application that ensures the .NET framework is installed. For example, the full version of Visual Studio 2008 is capable of creating an installer that detects whether the necessary .NET framework is present and, if not, downloads it from Microsoft (this feature is not present in Visual Studio Express).

7 MSP430 USB Tool Suite

TI MSP430 provides a complete suite of tools to make USB easy on MSP430 devices. Other members of the suite include:

- MSP430 USB CDC API

  An API for implementing the Communications Device Class, allowing the host application to interface with the device using a COM port.
- MSP430 and Windows USB HID API

  An MSP430 API and Windows API for implementing the Human Interface Device class. It has special features that allow HID to be a great choice for general-purpose use.
- MSP430 USB Descriptor Tool

  Automatically generates USB descriptors for the MSP430 USB API stacks. Its descriptors work the first time, saving the developer's time and reducing the chance for errors.
- MSP430 USB MSC API

  An API for implementing the Mass Storage class

For more information on these tools, see the application note *Starting a USB Design with MSP430 MCUs* (SLAA457), and check with TI for updates.

8 References

1. *MSP430x5xx/MSP430x6xx Family User's Guide* (SLAU208)
2. *Creating a Custom Flash-Based BSL* (SLAA450)
3. *MSP430 Programming Via the Bootstrap Loader* (SLAU319)
4. http://www.ti.com/430usb

Copyright © 2010–2014, Texas Instruments Incorporated

Revision History

Changes from B Revision (May 2011) to C Revision	Page
• Added link to firmware update source project in paragraph that starts "The USB solution for MSP430™ microcontrollers..."	1

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Copyright © 2010–2014, Texas Instruments Incorporated

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products (also referred to herein as “components”) are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers’ products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers’ products and applications, Buyers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of significant portions of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI components or services with statements different from or beyond the parameters stated by TI for that component or service voids all express and any implied warranties for the associated TI component or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards which anticipate dangerous consequences of failures, monitor failures and their consequences, lessen the likelihood of failures that might cause harm and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed a special agreement specifically governing such use.

Only those TI components which TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components which have ***not*** been so designated is solely at the Buyer's risk, and that Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2014, Texas Instruments Incorporated