参考資料

JAJA458

カスタム・フラッシュベース・ブートストラップ・ローダー(BSL)の作成

Lane Westlund MSP430 Tools

概要

MSP430F5xxおよびMSP430F6xxデバイスには、ブートストラップ・ローダー(BSL)をフラッシュ・メモリ内の保護された場所に配置することができます。出荷されるすべてのデバイスには標準的なTI BSLが搭載されていますが、このBSLを消去して、カスタムメイドのBSLを同じ場所にプログラミングすることが可能です。これにより、カスタム通信インターフェイスの使用、スタートアップ(起動)・シーケンス等の可能性が生まれます。このドキュメントの目標は、BSLのメモリの基本について説明するとともに、TIの標準的なBSLソフトウェアについても説明し、各顧客に合ったカスタム・プロジェクトで利用できるようにすることです。

このアプリケーション・レポートには、MSP430G2xデバイスで使用できる小サイズのデモ用BSLも記載されています。エントリ・シーケンスによりコードの更新が開始され、新しいユーザー・コードを送信してフラッシュに保存できるようになります。ステータスの通知用として、1バイトのフィードバックが提供されます。エントリ・シーケンス、データ、フィードバック用としてはTA0ベースのUART通信が使用されます。

このアプリケーション・レポートで説明されている、付随するプロジェクト(資料)およびソース・コードは、 次のURLからダウンロードできます。http://software-dl.ti.com/msp430/msp430_public_sw/mcu/msp430/CustomBSL/latest/index_FDS.html.

目次

1 5xxおよび6xxブートストラップ・ローダーのカスタマイズ 2
2 G2xxブートストラップ・ローダーの生成とカスタマイズ 9

図目次

図 1 デバイスの起動シーケンス 3
図 2 機能ブロック図、MSP430G2x01 9
図 3 デモ用セットアップ 15
図 4 Windows® XPの「デバイス マネージャ」画面例 16
図 5 HTermのCOMポート構成の画面例 16
図 6 View -> Register -> Calibration_Data (CCS) 17
図 7 View -> Memory: 0x10fe and 0x10ff (CCS) 18
図 8 CCS内のファイルにデバイスのメモリ内容を保存する 18

表目次

表 1 ピン割り当て 11
表 2 リンカ・コマンド・ファイル 12
表 3リンカ・ファイルの修正 12
表 4 カスタム・リンカ・ファイルの使用 14
表 5 プロジェクトの設定 14
表 6 ユーザー・アプリケーション 14
表 7 BSLデモのセットアップ 15

1 5xx および 6xx ブートストラップ・ローダーのカスタマイズ

1.1 BSL メモリのレイアウト

BSLメモリは、フラッシュの2キロバイト・セクションです。このフラッシュの具体的な位置は、各デバイス個別のデータ・シートに記載されています。ただし、通常はアドレス0x1000～0x17FF間に存在します。このメモリの各種セクションも保護することが可能です。(MSP430F5xx and MSP430x6xx Family User's Guide (SLAU208)で説明されているように)この保護は、SYSBSLCレジスタにフラグをセットすることでイネーブルになります。

1.1.1 Z 領域

BSLメモリが保護されている場合は、BSLメモリを読み出したり、BSLメモリ外の場所からBSLメモリ内にジャンプしたりすることはできません。これは、BSLを消去からより完全に保護するため、および誤ったBSLの実行を防ぐためです。ただしBSLメモリ空間全体を上記のように保護すれば、意図的にBSLを起動したり、特定のパブリックBSL関数を使用する等のどのような方法を用いても、ユーザーのアプリケーション・コードがBSLを呼び出すことはできなくなります。

Z領域は、制御された方法を使用して、保護されたBSLに公的にアクセスできるようにするために設計された特殊なメモリ・セクションです。Z領域はアドレス0x1000～0x100F間に存在するBSLメモリの1セクションであり、外部アプリケーション・コードからこのセクションにジャンプしたり、このセクションを読み出したりすることが可能です。Z領域は、BSLメモリ内の任意の位置へのジャンプを実行することが可能なゲートウェイとして機能します。デフォルトのTI BSLでは、BSLの先頭へのジャンプおよびBSLのパブリック関数へのジャンプ用にこの領域を使用します。

1.1.2 BSL 用に予約されたメモリ位置

次に示すように、BSLメモリには定義済み値の格納用に予約された固有の位置も用意され、BSLの起動が確実に正しく行われるようになっています。

0x17FC～0x17FF

JTAG Key: すべてのバイトが0x00か0xFFであれば、デバイスはJTAGに自由にアクセスできます。それ以外の値ではJTAGにアクセスできません。JTAG Keyの詳細については、MSP430F5xx and MSP430x6xx Family User’s Guide (SLAU208)を参照してください。

0x17FA

BSL起動ベクタ。BSL呼び出し時に最初に実行される命令のアドレスです。

0x17F8

予約

0x17F6

BSLアンロック・シグネチャ1: 正しくプログラムされたBSLを示すために、このワードを0xC35Aとセットする必要があります。このワードがセットされないとBSLは起動しません。

0x17F4

BSLアンロック・シグネチャ2: 正しくプログラムされたBSLを示すために、このワードを0x3CA5とセットする必要があります。このワードがセットされないとBSLは起動しません。

0x17F2

BSL保護関数ベクタ: BSL保護関数(protect function)の最初の命令のアドレスです。
この関数の詳細については、このアプリケーション・レポートで後述します。

1.2 デバイスの起動シーケンス

電源が投入されると、デバイスでは最初にBSLシグネチャをチェックします。BSLシグネチャに適切な値が存在していれば、デバイスによりBSL保護関数が呼び出されます。BSL保護関数についてはセクション1.2.1で詳細に説明されていますが、主な責務はBSLメモリを保護し、ユーザー・アプリケーションに代わってBSLを起動する必要があるかどうかを通知することです。デバイスの起動シーケンスは、図1のフロー・チャートに従って起こります。

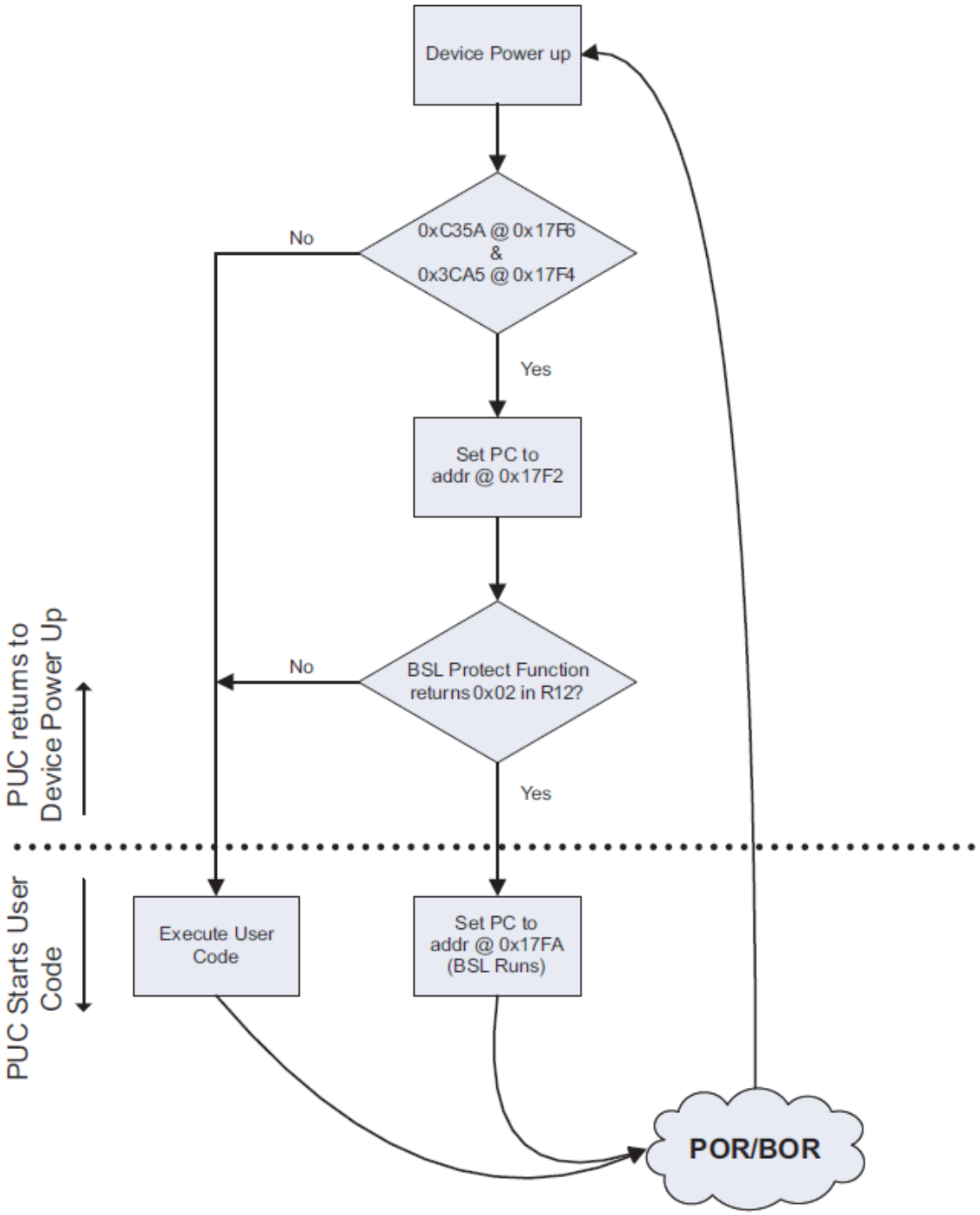

図 1 デバイスの起動シーケンス

1.2.1 BSL 保護機能

BSL保護機能は、BORの後と、ユーザー・コードの実行前に呼び出される機能です。このコードに課された時間的および機能的な制限はありませんが、この関数が戻らない場合は、デバイスが完全に応答しなくなります。さらに、戻り関数での遅延が長すぎると、デバッグ中に問題が発生する可能性があります。BSL保護機能の最も基本的な役割は、BSLメモリの保護、およびBSL保護機能を抜けた後にBSLまたはユーザー・コードを実行する必要があるかどうかの判定という2つの基幹機能(essential functions)を実行することです。

BSL保護機能は、スタック・ポインタがデフォルトの位置にセットされた状態で呼び出されます。デフォルトの位置は使用されるデバイスごとに異なります。スタック・ポインタを変更したり、スタック・ポインタのデータ値を操作したりすると、間違いなくデバイスが応答しなくなります。このような場合は、再プログラミングまでも含めて一切の処置が不可能になります。デバイスによっては、スタック・ポインタ空間(stack pointer space)が非常に限られており、BSL保護機能内でスタック・ポインタを使用しすぎるとメモリ・オーバーフローの原因となる可能性があります。正しいふるまいを確実に実現するには、スタックのアクセスを制限するか、スタック・ポインタを別の場所に移す必要があります。デバイスがBSL保護機能から確実に正しく戻るようにするには、戻る前にスタック・ポインタを回復させる必要があります。

1.2.1.1 BSLメモリの保護

SYSBLSCレジスタのサイズ・ビットと保護ビットをセットすることで行われます。これは、BSLの呼び出しが要求されるかどうかに関係なく行われます。これらの特定ビットの詳細については、MSP430F5xx and MSP430x6xx Family User’s Guide (SLAU208)を参照してください。

1.2.1.2 BSL呼び出しのチェック

BSL_Protect関数では、BSLを起動するか、(存在する場合は)アプリケーション・コードを起動するかを通知する必要もあります。 これは、R12のビットを次のように定義された値にセットすることで行われます。

ビット 0

0: JTAGステートを、JTAG Keyのステートに基づいて判定する必要があることを通知します。
1: JTAG Keyを上書きして、 JTAGをオープン状態に保ちます(主にBSLのデバッグに使用されます)。

ビット 1

0: BSLを起動してはならないことを通知します。
1: BSL起動ベクタの値をPCにロードしてBSLを起動する必要があることを示します。

注意する必要があるのは、BSLを呼び出す必要があるかどうかを判定するための方法が、全面的にBSL保護機能に依存していることです。TIの供給するUART BSLでは、SYSBSLINDビットをチェックして、特定ピンでトグル(切り替え)・シーケンスが発生しているかどうかをチェックします。ただし、TI供給のUSB BSLには、アプリケーション要件に基づくまったく異なる起動基準(startup criteria)があります。USB電力が存在し、デバイスがブランクの場合は、BSLを起動する必要があることがBSL_Protect関数により通知されるため、USBを介してブランクのデバイスをプログラム可能です。ただしカスタムBSLでは、ユーザー・コードを既知のCRC値と照合して正しくプログラムされたユーザー・コードのみが実行を開始するようにする等、想定可能なほとんどすべての基準を使用することが可能です。

1.3 TI 供給の BSL ソフトウェア

各MSP430x5xxおよびMSP430x6xxデバイスには、事前プログラムされたBSLが付属しています。このBSLの使用方法については、 MSP430 Programming Via the Bootstrap Loader (BSL) (SLAU319)で説明されています。このレベルでBSLに精通していることが、以降のセクションの前提となります。TI供給のBSLは、IAR KickStart用に作成され、IARKickStartを使用してコンパイルされています。

1.3.1 ソフトウェアの概要

TI供給のBSLは、きわめてモジュール化の進んだ方法で作成されています。必要なカスタマイズのレベルに応じて、各種のソース・ファイルを再利用したり置換したりすることが可能です。
BSLコードの主なセクションは、次の3つです。

ペリフェラル・インターフェイス

このコード・セクションでは、BSLコア・コマンドの受信と検証を担当します。このことを実現するために、任意のハードウェアやプロトコル(ラッパー・バイト等)を使用できます。実際の送信メカニズムとプロトコルは、BSLのその他の部分(rest)とは関連がありません。重要なのは、このコード・セクションではパケットを受信するよう要求された場合に、自らに送信されたデータをすべて正しく受信したことを認識するまで戻らないということです。BSLはユーザー・メモリの消去とプログラムに使用されるため、ペリフェラル・インターフェイスがフラッシュベースの割り込みベクタを使用することはできません。イベント・ポーリングか、RAMベースの割り込みベクタを使用する必要があります。

コマンド・インタプリタ

このコード・セクションでは、供給されたBSLコア・コマンドのバイトを解釈します。ペリフェラル・インターフェイスがエラーなしで正常にバイトを受信したことを前提にしていますが、必ずしも受信バイトがBSLプロトコルにとって正しいかどうか(例えば、正しくないコマンドが送信されていないかどうか)を前提にしていません。このコードでは、受信されたコア・コマンドをBSLコア・コマンド・リスト(SLAU319参照)に従って解釈し、BSL APIを呼び出して受信コマンドを実行します。

BSL API

このコード・セクションでは、コマンド・インタプリタ～読み出し/書き込み対象メモリ間の抽象化層を提供し、メモリのロック解除(unlocking)、メモリへの書き込み、メモリからの読み出し、CRC(巡回符号検査)という全局面(aspect)を扱います。また、可能な場合は常にセキュリティを扱うコード・セクションでもあります。これにより、コマンド・インタプリタではセキュリティの問題を気遣うことなく、要求の生成と応答の送信を簡素に行うことが可能になります。また、このモデルにより、きわめて安全性の高いカスタムBSLを生成することが可能になります。TIから供給されるセキュリティ検査および対策(security checks and measures)のメリットをあらゆるカスタムBSLが享受できるように、BSL APIはどのようなカスタムBSLにおいても、置き換えは、ほとんど無いセクションであるとみなされています。

1.3.2 ソフトウェア・ファイルの詳細

このセクションでは、すべての関数の詳細を(パラメータや返り値等まで)説明しているわけではありません。この情報は、ヘッダ・ファイルとソース・ファイルの関数コメント(function comments)に記載されています。このセクションの目標は、ソースを一読しただけでは明快に理解できない可能性のある、個々のファイルの重要な担当(responsibilities)を目立たせることです。

1.3.2.1 BSL430_Low_Level_Init.s43

このファイルでは、予約されたメモリ位置、BSL_Protect関数、Z領域内の公的に利用可能な(publically available)関数等の低レベルBSL局面(aspect)を扱います。通常、ここで必要となるカスタマイズは次に挙げるものだけです。

- 異なる条件でBSLを呼び出すためにBSL_Protect関数を変更する。
- 最終的なBSLイメージ用に、0xFFFF以外の値をJTAG keyの位置に書き込む。
- 付加的な関数をZ領域に追加して、ユーザー・コードを介して実行できるようにする。

1.3.2.2 BSL_Device_File.h

このファイルは、ある特定のデバイス派生品(device variations)を抽象化して、すべてのBSLソース・コード・ファイルをデバイス間で同じに扱うために使用されます。デバイスのインクルード・ファイル、ポートの再定義、クロック速度等、すべてのデバイス固有情報が記載されている場所です。さらに、次に示すようなカスタムBSLの定義も記載されています。すべてのBSLにすべての定義が有効であるわけではなく、このファイルに記載のない定義もあることに注意してください。

MASS_ERASE_DELAY

ロック解除時に、ACKを送信する前に使用される遅延値

INTERRUPT_VECTOR_START

BSLパスワードの開始アドレス用の定義

INTERRUPT_VECTOR_END

BSLパスワードの終了アドレス用の定義

SECURE_RAM_START

RAMの先頭(The start of the RAM)であり、安全のためにBSL起動時に上書きされる必要があります。これは通常、最低位のRAM値です。RAMはこのアドレス～スタック・ポインタ間でクリアされます。

TX_PORT_SEL

このポート定義の集まりは、TX/RXポートを個別のポート・ピンから抽象化し、ペリフェラル・インターフェイスのソース・コードを変更しなくてもこれらの値をデバイス間で変更できるようにするために使用されます。

RAM_WRITE_ONLY_BSL

これが定義されると、コンパイルされるBSLのサイズが大幅に小さくなります。この小サイズ版では、デバイス内のRAMへの書き込みと、ロック解除(unlock)、PCの設定、データ受信に必要なコマンドの受信のみが可能になります。スペースを節減し、USBスタックを2kBのBSLメモリに収納するために、USB BSL内で使用されます。RAM書き込み専用(write-only)BSLは、完全なBSLをRAMにロードして、このBSLを起動してさらに通信を行うために使用されます。

RAM_BASED_BSL

これが定義されると、BSL がBSLメモリでなくRAMの外で実行されている場合に必要となる、API内のコード・セクションがインクルードされます。これは主にフラッシュ書き込み時の遅延に対応するためであり、BSLがRAMの外(USB BSL等)で実行されている場合にのみ使用されます。

USB_VID

USB BSL用のベクタIDです。

USB_PID

USB BSL用のプロダクトIDです。

SPEED_x

これは、外付けオシレータ速度を定義するために使用され、USB BSLの起動シーケンスでテストされます。この場合の値はヘルツ単位のそのままの値です(例えば、 24000000は24MHzとなります)。

SPEED_x_PLL

デバイスのヘッダ・ファイルからの、SPEED_x 定義内に記述されているオシレータ速度についての対応するPLL定義です。

1.3.2.3 lnk430FXXXX_BSL_AREA.xcl

カスタム・リンカ・コマンド・ファイルです。プロジェクト・コードの出力をBSLメモリ位置に配置します。
さらに、前述の予約されたBSLメモリ・セクション用の定義も備えています。通常、このファイルを変更する必要はないはずです。BSLリンカ・コマンド・ファイルの種類によっては、リセット・ベクタの出力が意図的に、正しいデバイスRESETベクタ位置以外の場所に配置されます。こうすると、保護されたBSLメモリへのジャンプをデバイスが継続的に試みることになり、その結果リセットが発生して無限リセット・ループを起こします。

1.4 カスタム・ペリフェラル・インターフェイスの生成

既存のBSLソフトウェアは、別の通信インターフェイスへの移植が容易です。これが可能なのは、通信を扱うペリフェラル・インターフェイス・コードと、BSLソフトウェアのその他部分が非常に緩やかに結合しているためです。カスタム・ペリフェラル・インターフェイスを実装するには、ファイルBSL430_PI.hにある次のような関数を定義する必要があります。

1.4.1 PI_init ()

この関数では、次の3つを主に管理します。

- 通信ペリフェラルを初期化して、データ受信開始のステートに入るようにする。
- デバイスのクロック・システム(通常は8MHzですが、他のコードは独立しているため、その他の値でも可能です)を初期化する。
- BSL430_Command_Interpreterの値 "BSL430_ReceiveBuffer" と "BSL430_SendBuffer" を、データ・パケットが格納される位置にセットします。RAM (UART用等の)にある同一バッファでも、ペリフェラル(USB用等)に使用される別の位置でもかまいません。これらのポインタでBSLコア・コマンドの先頭バイト(first byte)をポイントする必要があることに注意してください。したがって、ペリフェラル・インターフェイスのバッファがラッパー・バイト用にさらに大きいバッファをRAMで使用する場合は、これらのポインタでそのバッファ内部をポイントする必要があります。

1.4.2 PI_receivePacket()

BSLの主なループ関数(primary loop function)です。この関数ではコア・コマンド用のすべてのバイトを受信し、その結果に基づいてコア・コマンドに値を返します。返り値の定義は次の通りです。

DATA_RECEIVED

データが正常に受信されました。コマンド自体が有効であるかどうかを検証するためのチェックは行われませんが、バイトはホストが送信したとおりに正しく受信されており、安全に処理できます。

RX_ERROR_RECOVERABLE

パケットの受信中に何らかのエラーが発生しました。パケットは失われましたが、受信関数(receive function)を再度呼び出すことが可能です。

RX_ERROR_REINIT

パケットの受信中にエラーが発生したために、それ以上パケットを受信する前にPI_init() 関数を再度呼び出す必要がある場合のために予約されています。

RX_ERROR_FATAL

パケットの受信中にエラーが発生し、それ以上の通信が不可能になった場合のために予約されています。完全なシステム再起動が必要になります。

PI_DATA_RECEIVED

パケットが受信され、ペリフェラル・インターフェイスによって処理されたことを通知します。PI_receivePacket()関数を再度呼び出すこと以外のアクションは不要です。

1.4.3 PI_sendData(int bufSize)

この関数は、コマンド・インタプリタが送信バッファを応答(reply)で満杯にした時に、コマンド・インタプリタによって呼び出されます。送信するバイト数をペリフェラル・インターフェイス側で認識できるように、バッファ内のデータのサイズがパラメータとして渡されます。

1.5 BSL の開発とデバッグ

1.5.1 開発とテスト

BSLメモリの保護が原因で、BSLコードのデバッグと開発は難しいものになりがちですが、新しいペリフェラル・インターフェイス等の「高レベル」の開発の場合は、ユーザー・コード・フラッシュの外部で実行するアプリケーションとして容易にBSLを開発することが可能です。

- プロジェクト・ビルドからBSL430_Low_Level_Init.s43を削除します (IAR内のファイルを右クリックして "remove from build"を選択)。
- CONFIGディレクトリからリンカ・コマンド・ファイルを使用して、BSL を"FLASH_AREA"に置きます (Project -> Options -> Linker -> Config -> Override default)。
- BSL呼び出しシーケンス中にデバイスをリセットせずに、外部アプリケーション(BSL_Scripter.exe等)を実行します。
- 正しくないパスワードを送信したり、一括消去(mass erase)を起動したりしないでください (IARではRESETベクタを自動的にビルドするため、パスワードはこれに含まれることを忘れないようにしてください)。

BSL_Protect関数の開発とデバッグのために、 シミュレータを使用することも、デバッグ中にBSLメモリ保護ビットをオープン状態にしておくことも可能です。どちらの場合も、"Run to Main" のチェックを外しておく必要があります。

1.5.2 特記事項とヒント

BSLコードは、RAM内の最高位アドレスと最低位アドレスでRAMを使用します。このため、BSLが複数のデバイスをターゲットとする場合は、RAM容量が最小のデバイスをターゲット(IARの場合は Project -> Options -> General Options -> Device)とする必要があります。また、SECURE_RAM_STARTは最高位の共有アドレスにセットする必要があります。

デバイスのBSLは、次に示すバージョンのIARを使用してビルドされています。
MSP430F5438A と CC430F6137: IAR 5.3 KickStart (IAR C/C++ コンパイラ 4.20.1を併用)
MSP430F5529: IAR 4.11C KickStart (IAR C/C++ コンパイラ 4.11Bを併用)

1.5.3 USB BSL 外付けオシレータの周波数

USB BSLではルーチンを使用して、アプリケーションで使用される外付けオシレータの速度を測定します。測定は、外付けクロックの速度を既知の較正済み内部クロックと比較することで行われます。この方法で、デフォルトのBSLを修正せずに、ある特定の外付けオシレータ周波数で使用できるようになります。アプリケーションで他の周波数を使用する場合は、SPEED_x値とそれに対応するSPEED_x_PLL値を変更できます。この2つは、最高速度から最低速度へという順番になっている必要があります。使用する速度が一種類のみの場合でもすべての値を定義する必要がありますが、定義する周波数は同一にできます。

```
//9MHz Example Code
#define SPEED_1          9000000
#define SPEED_1_PLL      USBPLL_SETCLK_9_0
#define SPEED_2          SPEED_1
#define SPEED_2_PLL      SPEED_1_PLL
#define SPEED_3          SPEED_1
#define SPEED_3_PLL      SPEED_1_PLL
#define SPEED_4          SPEED_1
#define SPEED_4_PLL      SPEED_1_PLL
```

上記のコード例に示されているように、既知の周波数を1種類だけ使用する場合でも、計測ループをUSBペリフェラル・インターフェイスに残しておくことが重要となります。水晶発振回路(クリスタル)の起動(startup)を可能にするための遅延にも使用されるためです。

2 G2xx ブートストラップ・ローダーの生成とカスタマイズ

2.1 対象となるシステム仕様(Target System Specification)

このブートストラップ・ローダーは、MSP430G2001(MSP430バリュー・ラインのデバイス・メンバの中では現時点で最小サイズの製品)で使用することを目的として設計されています。ただし、他の G2xxデバイスに合わせて修正することも容易に可能となっています。その場合に必要となる修正は、このアプリケーション・レポートに記載されています。

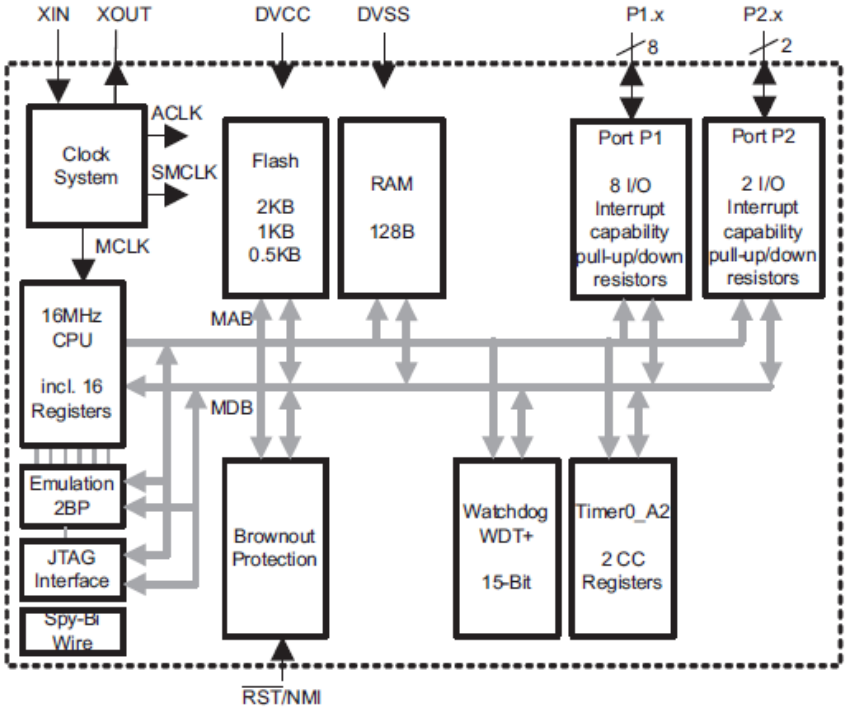

図 2 機能ブロック図、MSP430G2x01

BSLでは、次のようにメモリ・サイズが重要となります。

RAM: 128バイト
フラッシュ: 512バイト、MSP430G2001でのセグメント1つ
Info A: 64バイト－ DCO較正データ2バイト(1 MHzの場合)
Info B, C, D: それぞれ64バイト

2.2 BSL の仕様

このアプリケーション・レポートで紹介されているBSLは、ひとつの単純な要件に従っています。その要件とは「簡素さ」です。それに加えて、デバイスで使用可能なリソースがこのBSLの機能のしかたに大きく影響しています。

注: このBSLを、他の TIのBSL製品と混同しないようにしてください。ROMかフラッシュかを問わず、このBSLは他のMSP430 BSL製品をベースにして作られたわけではありません。

2.2.1 機能性

使用可能なコード・サイズに関しては、ひとつの機能だけが実装されます。そのフローは以降のセクションで説明されます。正確にこの順番でないとコードを実行できないことに注意してください。

2.2.1.1 エントリ・シーケンス

BSLに入ることが可能なのは、デバイスがリセット・イベントから抜けた時、かつBSL_entryピンがLowにプルされている時のみです。

2.2.1.2 同期

エントリ・シーケンスの後、BSLはSYNC文字の待機中となります。SYNC文字は、BSLエントリが偶然によるものではないことを確認するために使用されます。

1バイトが受信されるまで、デバイスは永続的にループし続けることに注意してください。このバイトがSYNC文字と一致すれば、BSLでは動作を継続します。一致しない場合は、BSLによりリセット・イベントが生成されます。

SYNC文字 = 0xBA

2.2.1.3 直前のフラッシュ内容の消去

SYNC文字が正常に受信された後、BSLでは直ちにメイン・フラッシュを消去して、新しいデータに備えます。

ユーザー・コードはメイン・フラッシュに存在していますが、そのコード空間を割り込みベクタ・テーブルと共有しています。そのため、BSLではBSLをポイントしているリセット・ベクタを復元するための処置を直ちにとる必要があります。復元にはフラッシュ・セグメント1つあたり約16msの時間がかかりますが、最低でもその2倍の時間の間、デバイスの電圧を一定に保つ必要があります。個別のデバイスのデータ・シートを参照して、ひとつのデバイスに存在するフラッシュ・セグメントの数を判断してください。

警告(CAUTION)

フラッシュの消去～リセット・ベクタ復元完了までの期間中、システムはロックアウト状態の発生に対して非常に脆弱となっています。ロックアウトが発生した場合は、デバイスに再度アクセスすることはできなくなります。

2.2.1.4 新しいユーザー・データの受信と書き込み

消去が発生した後、BSLではユーザー・データを1バイトずつ受信できるようになります。ハンドシェイクは行われないため、システムの予測するデータのサイズは非常に正確となります。このデータ・サイズは、使用可能なメイン・フラッシュの総サイズから2バイトを引いた値となります。 範囲は最低位のフラッシュ・アドレスからリセット・ベクタまでですが、リセット・ベクタ自体は入りません。

2.2.1.5 データの検証

デーががすべて受信された後、1バイトのみのXORチェックサムがデバイスに送信されます。BSLではこのチェックサムを、書き込まれたフラッシュ・データについて計算したチェックサムと比較します。

チェックサムが一致すれば、BSLではACK文字(0xF8)を返します。チェックサムが一致しない場合は、NACK (0xFE)が返されます。

2.2.2 メモリのフットプリント

このアプリケーション・ノートで使用されているような小サイズのデバイスにブートストラップ・ローダーを実装する際に最も難しい問題になるのが、コードのサイズです。その本質から見て、BSLは常にメモリ内に存在して、ロックアウト状態が発生しないようにする必要があります。ロックアウト状態になると、デバイスにアクセスできなくなります。

BSLのユーザー・コード更新中にBSLコードが変更されないようにするには、ユーザー・コードとは別のフラッシュ・セクションにBSLが常駐している必要があります。MSP430G2001デバイスには使用可能なメイン・フラッシュ・セクションがひとつしかないため、BSLの常駐先は残りの唯一のフラッシュ部分、つまり情報メモリ(総使用可能サイズ254バイト)とする必要があります。

2.2.3 ペリフェラル

MSP430G2xxデバイスの中には、通信モジュールが搭載されていないものもあります。したがって、シンプルなソフトウェア・ベースのインターフェイスが使用されます。通信機構がシンプルなため、非ハンドシェイク型UARTが使用されます。
効率的にこのソフトウェアUARTを実装するために「ボー・レート9600、データ・ビット8個、ストップ・ビット1個、パリティ無し」という設定でTimer_Aと2つのGPIOピンが使用されます。

さらに、RSTピンを介したデバイス・リセットと連動して、1つのGPIOピンを使用してBSL起動シーケンスを開始します。

表 1 ピン割り当て

ピン	機能(関数)	LaunchPadピン
P1.3	BSL_entry	Switch S2(スイッチ S2)
P1.1	UART RxD(データ受信)	Cross connect(相互接続)! TXD(データ送信)
P1.2	UART TxD(データ送信)	Cross connect(相互接続)! RXD(データ受信)
RESET	RESET(リセット)	RESET(リセット)
USB	Backchannel UART and power (バックチャネルUARTと電力)	UST～Host (UST～ホスト間)

2.3 実装

プロジェクトはBSL部と、メインまたはユーザー・アプリケーション部の2つに分割されます。各種ニーズに応じて、C言語版またはアセンブリ言語(ASM)版の ユーザー・コードが入手可能です。

2.3.1 BSL アセンブラ・コード

BSL用のすべてのコードは、MSP430G2xxBSL_CCS(IAR).asmに記載されています。内容は次の通りです。

- リセット割り込みテーブル・ステートメント (エントリ・ポイント)
- BSLコード
- 小サイズのユーザー・アプリケーション例 (デモ目的でのみLEDを点灯)

2.3.1.1 DCO較正データの保存

BSLコードは情報メモリに存在するため、InfoAに格納されているDCOの較正データを保存して復元することが重要となります。試作中は、これを次のように手動で行うことができます。

```
; ------------------------------------------------------------------
; Note that the user needs to ensure that DCO Cal Data is not
; erased during debugging - Read out from InfoA and hardcode
; two bytes in move commands below.
mov.b   #YOUR_DEVICE_VALUE,&DCOCTL     ; Copy from address 0x010FEh
mov.b   #YOUR_DEVICE_VALUE,&BCSCTL1    ; Copy from address 0x010FFh
; Replace YOUR_DEVICE_VALUE with values gathered from actual
; device. Values look like that: 0b3h and 086h
; (CCS Debug -> Debugger -> Loading options: Load symbols only)
; ------------------------------------------------------------------
```

量産(production)段階では、これはさらに容易になります。情報メモリが空のため、書き込みの前に消去する必要がないとみなせるためです。元のデータを使用するには、コードを変更する必要があります。GANG430等、大多数の量産プログラマ(production programmer) でも、フラッシュ・データを「保全する」機能をサポートしています。

```
; ===============================================================
; For Mass production please enable cal data readout directly from
; InfoA. Empty devices should be programmed without erasing flash
; before. If erased first, CAL data is lost. But can be restored
; with FlashPro430 / GangPro430 programmers. www.elprotronic.com
mov.b   &CALDCO_1MHZ,&DCOCTL   ; Set DCO step + modulation
mov.b   &CALBC1_1MHZ,&BCSCTL1  ; Set range
; ===============================================================
```

2.3.1.2 リンカ・コマンド・ファイル

もう一つ重要なBSLのファイルが、コードとデータの場所を定義するリンカ・コマンド・ファイルです。リンカ・コマンド・ファイルはCCSとIARではわずかに異なりますが、「BSLとアプリケーション・コードを置く場所をリンカに伝える」という概念は同じです。

2.3.1.2.1 リンカ・コマンド・ファイルの配置

デフォルトでは、IDEに付随する標準的なリンカ・コマンド・ファイル が選択・使用されます。このBSLプロジェクトの場合は、ファイルにいくらか修正を加える必要があります。

表 2 リンカ・コマンド・ファイル

CCS	IAR
CCSでは自動的にデフォルトのリンカ･コマンド･ファイルをプロジェクト・ディレクトリにコピーします。ファイルには、CCS Project Explorerを使用してアクセス可能です。	IARでは、インストール・パスからデフォルトのリンカ・コマンド・ファイルを直接選択して使用します。通常のパスは次の通りです。 C:\Program Files\IAR Systems\EWB MSP430 5.20\430\config
標準的なファイル名	
lnk_msp430g2001.cmd	lnk430g2001.xcl
元のファイルの保持方法	
CCSではファイルをプロジェクトにコピー済みです	元のファイルをプロジェクト・ディレクトリにコピーして、デフォルトのファイルを保全する必要があります。
カスタマイズされたファイルを明確に区別するために変更された後のファイル名	
lnk_msp430g2001_bsl.cmd	lnk430g2001_bsl.xcl

2.3.1.2.2 リンカ・ファイルの修正

このBSLプロジェクトの場合は、次のような修正をファイルに加える必要があります。

表 3リンカ・ファイルの修正

CCS	IAR
情報メモリ・セグメントがデータ用に使用されなくなっているため、削除することが可能です	
MEMORYセクション内の次の4行を削除するか、コメント化(/* および */)します /*INFOA: origin = 0x10C0, length = 0x0040 */ /*INFOB: origin = 0x1080, length = 0x0040 */ /*INFOC: origin = 0x1040, length = 0x0040 */ /*INFOD: origin = 0x1000, length = 0x0040 */	次の5行を削除するか、コメント化(//)します //-Z(CONST)INFO=1000-10FF //-Z(CONST)INFOA=10C0-10FF //-Z(CONST)INFOB=1080-10BF //-Z(CONST)INFOC=1040-107F //-Z(CONST)INFOD=1000-103F
この使用されなくなったメモリは、BSLと表示(labeled)され、かつコードが含まれる新しいメモリ・ブロック1つに割り当てられます。下記の行が、直前に削除またはコメント化された行の下に追加されます。	
BSL : origin = 0x1000, length = 0x00FE	-P(CODE)BSL=1000-10FD
メモリ領域への内容の割り当て	
SECTIONS部から次の4行を削除します。 /* MSP430 INFO FLASH MEMORY SEGMENTS */ /* .infoA : {} > INFOA */* .infoB : {} > INFOB */ /* .infoC : {} > INFOC */ /* .infoD : {} > INFOD */	上のステートメントですでに行われています。

これらの行の下に、次の1行を追加します。 bsl : {} > BSL /* BSL CODE */	

2.3.1.2.3 IDE がカスタム・リンカ・ファイルを使用するようにする

このBSLプロジェクトの場合は、次のような修正をファイルに加える必要があります。

表 4 カスタム・リンカ・ファイルの使用

CCS	IAR
リンカ・コマンド・ファイルがプロジェクト・パスに存在している場合(これがデフォルト)は、アクションは不要です。	Project -> Options -> Linker -> Config デフォルトのリンカ構成ファイルを、直前に生成したファイルで上書き(Override)します。 xcl ファイルがasmファイルと同じ位置に格納されている場合は、次の用に$PROJ_DIR$ を使用できます。 $PROJ_DIR$\lnk430g2211_bsl.xcl. そうでない場合は、フルパスを提供してファイルをポイントすることが可能です。

2.3.1.3 プロジェクトの設定

IDE自体の中にあるプロジェクトが、BLSのもうひとつの重要な部分です。このプロジェクトをセットアップして、正しいデバイスを使用したり、InfoAを含めすべての情報メモリ位置に対する消去や書き込みアクセスが行えるようにする必要があります。

表 5 プロジェクトの設定

CCS	IAR
ターゲット・デバイスの指定方法	
Project -> Properties -> CCS Build -> General -> Device Variant	Project -> Options -> General Options -> Target -> Device
情報メモリへのアクセス	
Debug Properties -> Target -> MSP430 Properties -> Download Options -> Erase options: メイン・メモリ、情報メモリ、保護対象情報メモリを消去します。	Project -> Options -> Debugger -> FET Debugger -> Download: ロックされたフラッシュ・メモリの消去と書き込みアクセスを可能にし、メイン・メモリと情報メモリを消去します。

2.3.2 ユーザー・アプリケーション

参考の目的で、点滅LEDの例が提供されています。

このプログラムは、他のあらゆるユーザー・アプリケーション同様、BSLを修正しなくてもダウンロードしてデバッグできます。

BSLとユーザー・アプリケーションの間では、プログラムの開始位置以外に相互作用はありません。これは、BSLがこの開始位置にジャンプして、それがコードであるとみなすためです。データがこの位置に格納されている等の場合は、ユーザー・アプリケーションが正しく起動しなくなります。開始位置はデバイスとプログラム言語によって異なります (表6参照)。

表 6 ユーザー・アプリケーション

CCS	IAR
アセンブリ	
ラベリング直後にユーザー・アプリケーションが起動します。	
.text	RSEG コード
コード例	
mainApp_CCS.asm	mainApp_IAR .asm
C	
ユーザー・アプリケーションがmain() 関数として起動します。実際の主要開始位置はcinit() 関数に依存するため、コンパイラとリンカが位置を管理します。	
コード例	
mainApp.c	

2.4 BSL の動作

2.4.1 ハードウェアのセットアップ

デモ用としては、MSP430G2xxがLaunchPadプラットフォームに装着されており、表7に記載の接続が行われることが前提となります。

表 7 BSLデモのセットアップ

ピン	機能(関数)	LaunchPadピン
P1.3	BSL_entry	Switch S2(スイッチ S2)
P1.1	UART RxD(データ受信)	TXD(データ送信) (相互接続)
P1.2	UART TxD(データ送信)	RXD(データ受信) (相互接続)
RESET	RESET(リセット)	RESET(リセット)
USB	(バックチャネルUARTと電力)	UST～Host (UST～ホスト間)

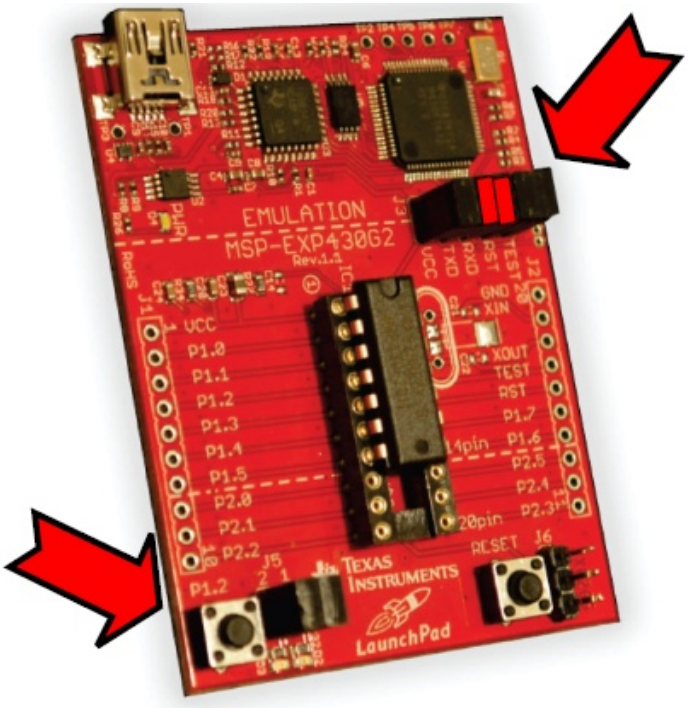

図 3 デモ用セットアップ

2.4.2 ホストへの接続

デモを実行するには、LaunchPadを ホストPCに接続する必要があります。これによりボードに電源が投入され、Spy-Bi-Wire を介したコードのダウンロードがデバッグ用に可能になり、またBSL動作にとって最も重要なことですが、 UARTインターフェイスが提供されます。すべてのLaunchPadのオンボード・エミュレーション回路には自由に使えるアプリケーションUARTが搭載されています。アプリケーションUARTを使用して、USB ～UARTブリッジ～ホストPCという接続が実現されます。

注: 電圧とタイミングの要件が合う限り、上記以外のどんな方法を使用してもデバイスにUARTインターフェイスを提供することは可能です。

2.4.2.1 COMポートの決定

LaunchPadがホストに接続された後、どのCOMポートを割り当てるかを決める必要があります。
Windows® XPのシステムでは「デバイス マネージャ」を使用してCOMポートが決められます。「デバイス マネージャ」はdevmgmt.mscを実行することで起動できます。

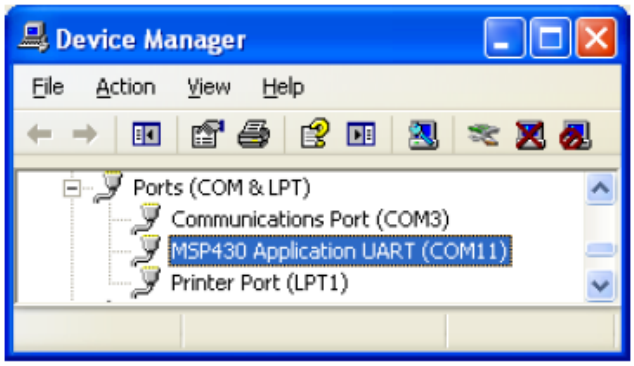

図 4 Windows® XPの「デバイス マネージャ」画面例

「Ports(COM & LPT)」という見出しのセクション内に、LaunchPadが「MSP430 Application UART」として表示されているはずです。
COMポートの番号は、この名前の右側に記載されています。 例えば図4の画面例では「COM11」となっています。

2.4.2.2 COMポートのセットアップ

COMポートは用途の広いインターフェイスであるため、初期化して「ボー・レート9600、データ・ビット8個、ストップ・ビット1個、パリティ無し」という正しい設定にすることが重要です。

上記のことを行うもっとも簡単な方法は、端末プログラム(terminal program)を使用することです。端末プログラムは、このBSLを使用するためにも必要です。このアプリケーション・レポートでは、シンプルでありながら強力なHTermという端末プログラムを使用します。これはhttp://www.der-hammer.info/terminal/hterm.zipからダウンロードでき、このアプリケーション・レポートを執筆している時点では(as of this writing)無償です。また、HTerm はLinuxシステムにも使用できます。

セクション2.4.2.1に記載のCOMポートを選択し、ポート値を設定して、デバイスに接続します。

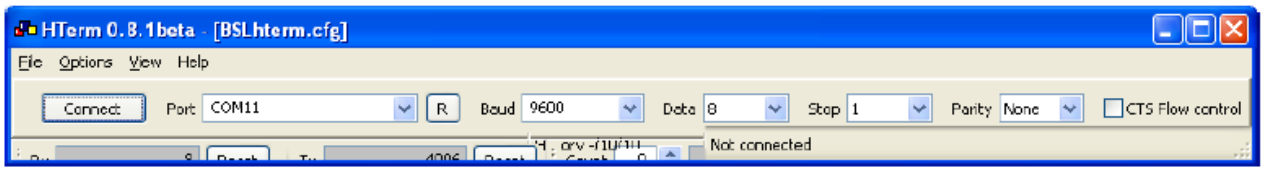

図 5 HTermのCOMポート構成の画面例

注: Htermはこの目的に使用できる多くのプログラムのひとつであり、あくまでこのプロジェクト用のGUIの一例として使用されているにすぎません。

2.4.3 BSL を動作させるための標準的な手順(Sequence)

新しいデータをデバイスのフラッシュに保存するために、BSLが次の手順で使用されます。

1. LaunchPad上のリセット・ボタンを押したままにして、MSP430G2xxをリセット・ステートにします。
2. LaunchPad上のボタンS2 (BSL_entry ピン)を押して、リセット・ボタンを放します。デバイスがBSLモードになったら、S2を放します。
3. SYNC文字をホストからデバイスに送信します。
4. (フラッシュ・セグメント数(NumberOfFlashSegments) × 16 × 2) ms以上待機して、デバイスがフラッシュを消去し、割り込みベクタを復元するための時間を十分に取ります。
5. ユーザー・コードをバイト単位でホストからデバイスに送信します。
6. チェックサムをホストからデバイスに送信します。
7. 応答 (ACK またはNACK) をデバイスから読み出します。
8. 応答(answer)がACKの場合は、BSLの行った強制リセットによりデバイスがすでにアプリケーション・モードになっています。
9. 応答(answer)がNACKまたは無反応の場合は、ACKが受信されるまでこの手順を反復して実行します。

HTermには"Send File" ボタンがあり、これを使用して、バイナリファイルに格納されたデータを送信できます。このドキュメント経由で入手可能なZipアーカイブには、BSLを検証するためのユーザー・コード(ファームウェア)のファイルが入っています。また、数字の形式を気にせずに同期文字を送信するためのSYNC.binファイルも入っています。

2.4.4 BSL を介してダウンロードするための新規コードの生成

2.4.4.1 カスタム・アプリケーションの生成

顧客アプリケーションの開発は常に、仕様の作成、コーディング、デバッグ、テストという手順で行われます。この時点では、プロジェクトにBSLを組み込む(インクルードする)必要はありません。

便宜のため、C言語とアセンブリ言語で作成された2つのプロジェクト・デモが入手可能であり、アプリケーション開発の開始点として使用できます (詳細についてはセクション2.3を参照)。

> **注: 情報メモリはユーザー・コード用には使用できません。デバッグ・オプションの'erase Info memory(情報メモリを消去)' のチェックを外して、この領域が誤って使用されないようにしておくとよいでしょう。**

2.4.4.2 較正データの保存

試作中は、1MHzのDCOの較正データを保持しておくことが重要となります。このデータは、BSL実行中にホストと安定した通信を行うために使用されます。またDCO較正データは、ユーザー・アプリケーション内で定義されたクロック周波数用にも役立ちます。

較正データは、InfoAデータの上書きが発生しないユーザー・アプリケーションのデバッグ中に最も簡単に読み出せます。例えば、図6と図7に示す方法で較正値を取り出せます。

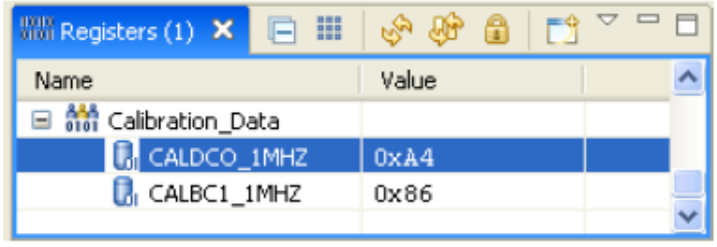

図 6 View -> Register -> Calibration_Data (CCS)

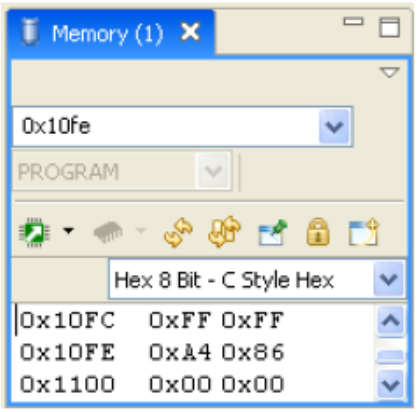

図 7 View -> Memory: 0x10fe and 0x10ff (CCS)

デバッグ中に、これらの値を記録してBSL asm ファイル内で復元することが重要となります。量産(production)の場合は、量産プログラマ製品GANG430を使用する等、別の方法で較正データを保存します。

2.4.4.3 ユーザー・アプリケーション・コードをBSL更新ファイルにする

BSLの更新を意図して、ユーザー・アプリケーションを作っている場合は、 BSLデータを生成することが可能です。

2.4.4.3.1 CCSの使用

IDEを利用すると、これを最も簡単に行うことができます。アプリケーションがデバイスにダウンロードされた後、メモリを再度読み出してファイルに保存できます。

View -> Memoryと選択して出てきた画面で、上向き矢印の付いた小さな緑のICのアイコンをクリックすると、メモリの読み出しと保存を行うことができます(図8参照)。

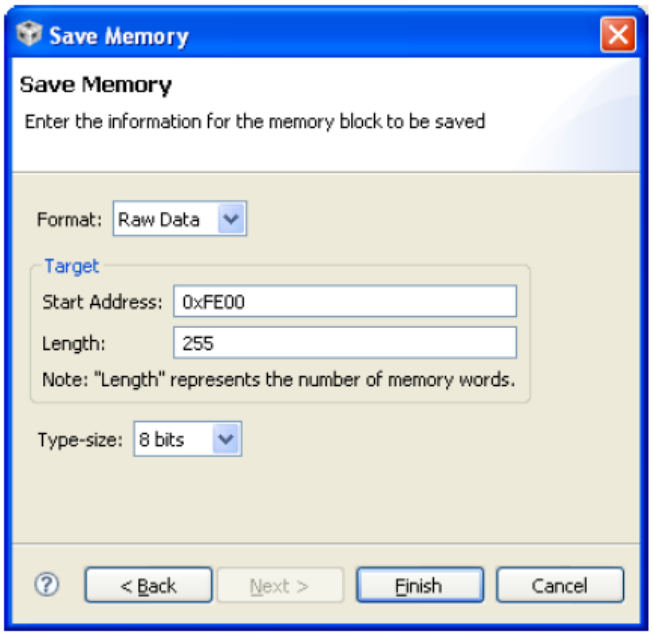

図 8 CCS内のファイルにデバイスのメモリ内容を保存する

このファイルはこれ以降、(HTerm等の)端末プログラムに対する入力ファイルとして使用できます。このファイルには XORチェックサムが含まれていないことに注意してください。XORチェックサムは次のステップで付加されます。

2.4.4.3.2 IARの使用

IARでは(現時点では) TI Text形式またはIntel Hex形式としてのみメモリ領域を保存できるため、必要なメモリ領域を保存した後、変換ユーティリティ(hex2bin等)を使用して出力ファイルを変換する必要があります。

2.4.4.4 XORチェックサムの取得

追加のプログラムを使用せずに正しいチェックサムを取得する最も容易な方法は、デバイスをデバッグ・モードで使用することです。

注: デバッガに接続している場合は、ハードウェア上でリセット・スイッチを使用しないでください。リセット・スイッチではなく、IDEのソフトウェア・リセット・ボタンを使用してください。

2.4.4.4.1 ユーザー・データの送信

正しいエントリ・シーケンスの後、ユーザー・コードの全バイト(メイン・フラッシュ・サイズから2バイトを引いた値)がホストからデバイスに送信されます。これらのバイトには、XORチェックサムは含まれていません。チェックサムは最後のバイトとして送信されます。

2.4.4.4.2 チェックサムの読み出し

ここで、IDEのデバッグ・セッションを一時停止して、デバイス上で計算されたフラッシュのチェックサムを読み出します。チェックサムは、rCHKSUMという識別名(symbolic name)でコア・レジスタR8に置かれています。

2.4.4.4.3 取得したチェックサムの送信

ここで、ターゲットを動かして(一時停止解除)、R8から抽出された値をターゲットに送信します。これによりBSLサイクルが完了し、ACKが受信されることになります。

2.4.4.4.4 データの検証

チェックサムが正しかったかどうか、およびデバイス内のデータが正しいかどうかを検証するには、BSLを介してフラッシュに書かれたデータを読み出し(図8参照)、元のフラッシュ・ファイルと比較する必要があります。

2.4.4.4.5 チェックサムの保存

比較してもエラーが出なければ、チェックサムをファームウェア・ファイルに付加できます。これにより、チェックサムをデバイスごとに計算し直さなくても、複数デバイスへのダウンロードが容易に可能になります。

2.4.5 量産(本稼働)に向けての準備

量産(本稼働)に備えて、デバイスのフラッシュに書き込むための有効なファームウェア・イメージを1つ用意する必要があります。

これを行う簡単な方法は、TI-TXT形式のコードのテキスト版を使用して、BSL TI-TXTの内容をユーザー・アプリケーションにコピー&ペーストすることです。

BSLプロジェクトにユーザー・コードを追加します。この方法を使用すると、BSLとユーザー・コード間の通信のデバッグが可能になるというメリットが得られます。

ご注意

Texas Instruments Incorporated 及びその関連会社（以下総称して TI といいます）は、最新の JESD46 に従いその半導体製品及びサービスを修正し、改善、改良、その他の変更をし、又は最新の JESD48 に従い製品の製造中止またはサービスの提供を中止する権利を留保します。お客様は、発注される前に、関連する最新の情報を取得して頂き、その情報が現在有効かつ完全なものであるかどうかご確認下さい。全ての半導体製品は、ご注文の受諾の際に提示される TI の標準販売契約約款に従って販売されます。

TI は、その製品が、半導体製品に関する TI の標準販売契約約款に記載された保証条件に従い、販売時の仕様に対応した性能を有していることを保証します。検査及びその他の品質管理技法は、TI が当該保証を支援するのに必要とみなす範囲で行なわれております。各デバイスの全てのパラメーターに関する固有の検査は、適用される法令によってそれ等の実行が義務づけられている場合を除き、必ずしも行なわれておりません。

TI は、製品のアプリケーションに関する支援又はお客様の製品の設計について責任を負うことはありません。TI 製部品を使用しているお客様の製品及びそのアプリケーションについての責任はお客様にあります。TI 製部品を使用したお客様の製品及びアプリケーションに関連する危険を最小のものとするため、適切な設計上及び操作上の安全対策は、お客様にてお取り下さい。

TI は、TI の製品又はサービスが使用されている組み合せ、機械装置、又は方法に関連している TI の特許権、著作権、回路配置利用権、その他の TI の知的財産権に基づいて何らかのライセンスを許諾するということは明示的にも黙示的にも保証も表明もしておりません。TI が第三者の製品もしくはサービスについて情報を提供することは、TI が当該製品又はサービスを使用することについてライセンスを与えるとか、保証又は是認するということを意味しません。そのような情報を使用するには第三者の特許その他の知的財産権に基づき当該第三者からライセンスを得なければならない、又は TI の特許その他の知的財産権に基づき TI からライセンスを得て頂かなければならない場合もあります。

TI のデータ・ブック又はデータ・シートの中にある情報の重要な部分の複製は、その情報に一切の変更を加えること無く、且つその情報と関連する全ての保証、条件、制限及び通知と共になされる限りにおいてのみ許されるものとします。TI は、変更が加えられて文書化されたものについては一切責任を負いません。第三者の情報については、追加的な制約に服する可能性があります。

TI の製品又はサービスについて TI が提示したパラメーターと異なる、又は、それを超えてなされた説明で当該 TI 製品又はサービスを再販売することは、関連する TI 製品又はサービスに対する全ての明示的保証、及び何らかの黙示的保証を無効にし、且つ不公正で誤認を生じさせる行為です。TI は、そのような説明については何の義務も責任も負いません。

TI からのアプリケーションに関する情報提供又は支援の一切に拘わらず、お客様は、ご自身の製品及びご自身のアプリケーションにおける TI 製品の使用に関する法的責任、規制、及び安全に関する要求事項の全てにつき、これをご自身で遵守する責任があることを認め、且つそのことに同意します。お客様は、想定される不具合がもたらす危険な結果に対する安全対策を立案し実行し、不具合及びその帰結を監視し、害を及ぼす可能性のある不具合の可能性を低減し、及び、適切な治癒措置を講じるために必要な専門的知識の一切を自ら有することを表明し、保証します。お客様は、TI 製品を安全でないことが致命的となるアプリケーションに使用したことから生じる損害の一切につき、TI 及びその代表者にその全額の補償をするものとします。

TI 製品につき、安全に関連するアプリケーションを促進するために特に宣伝される場合があります。そのような製品については、TI が目的とするところは、適用される機能上の安全標準及び要求事項を満たしたお客様の最終製品につき、お客様が設計及び製造ができるようお手伝いをすることにあります。それにも拘わらず、当該 TI 製品については、前のパラグラフ記載の条件の適用を受けるものとします。

FDA クラス III（又は同様に安全でないことが致命的となるような医療機器）への TI 製品の使用は、TI とお客様双方の権限ある役員の間で、そのような使用を行う際について規定した特殊な契約書を締結した場合を除き、一切認められていません。

TI が軍需対応グレード品又は「強化プラスティック」製品として特に指定した製品のみが軍事用又は宇宙航空用アプリケーション、若しくは、軍事的環境又は航空宇宙環境にて使用されるように設計され、かつ使用されることを意図しています。お客様は、TI がそのように指定していない製品を軍事用又は航空宇宙用に使う場合は全てご自身の危険負担において行うこと、及び、そのような使用に関して必要とされるすべての法的要求事項及び規制上の要求事項につきご自身のみの責任により満足させることを認め、且つ同意します。

TI には、主に自動車用に使われることを目的として、ISO/TS 16949 の要求事項を満たしていると特別に指定した製品があります。当該指定を受けていない製品については、自動車用に使われるようには設計されてもいませんし、使用されることを意図しておりません。従いまして、前記指定品以外の TI 製品が当該要求事項を満たしていなかったことについては、TI はいかなる責任も負いません。

弊社半導体製品の取り扱い・保管について

半導体製品は、取り扱い、保管・輸送環境、基板実装条件によっては、お客様での実装前後に破壊／劣化、または故障を起こすことがあります。

弊社半導体製品のお取り扱い、ご使用にあたっては下記の点を遵守して下さい。

1. 静電気
   - 素手で半導体製品単体を触らないこと。どうしても触る必要がある場合は、リストストラップ等で人体からアースをとり、導電性手袋等をして取り扱うこと。
   - 弊社出荷梱包単位（外装から取り出された内装及び個装）又は製品単品で取り扱いを行う場合は、接地された導電性のテーブル上で（導電性マットにアースをとったもの等）、アースをした作業者が行うこと。また、コンテナ等も、導電性のものを使うこと。
   - マウンタやはんだ付け設備等、半導体の実装に関わる全ての装置類は、静電気の帯電を防止する措置を施すこと。
   - 前記のリストストラップ・導電性手袋・テーブル表面及び実装装置類の接地等の静電気帯電防止措置は、常に管理されその機能が確認されていること。
2. 温・湿度環境
   - 温度：0～40℃、相対湿度：40～85%で保管・輸送及び取り扱いを行うこと。（但し、結露しないこと。）
   - 直射日光があたる状態で保管・輸送しないこと。
3. 防湿梱包
   - 防湿梱包品は、開封後は個別推奨保管環境及び期間に従い基板実装すること。
4. 機械的衝撃
   - 梱包品（外装、内装、個装）及び製品単品を落下させたり、衝撃を与えないこと。
5. 熱衝撃
   - はんだ付け時は、最低限 260℃以上の高温状態に、10 秒以上さらさないこと。（個別推奨条件がある時はそれに従うこと。）
6. 汚染
   - はんだ付け性を損なう、又はアルミ配線腐食の原因となるような汚染物質（硫黄、塩素等ハロゲン）のある環境で保管・輸送しないこと。
   - はんだ付け後は十分にフラックスの洗浄を行うこと。（不純物含有率が一定以下に保証された無洗浄タイプのフラックスは除く。）

以上