MICROCHIP

AN1310

PIC16/PIC18 デバイス用高速シリアル ブートローダ

著者： *E. Schlunder*
Microchip Technology Inc.

はじめに

マイクロチップ・テクノロジー社の強化型フラッシュ マイクロコントローラは、ファームウェアによる自己書き込みが可能です。「ブートローダ」はそのためのファームウェア カーネルを提供します。カーネルはマイクロコントローラに常駐し、ファームウェアのメイン アプリケーションが通常使用する事のないプログラムメモリのごく一部を使用します。

ブートローダ ファームウェアが起動すると、ホスト PC はシリアル プロトコルを使用してマイクロコントローラのアプリケーション ファームウェアの読み出し、書き込み、更新確認を実行できます。アプリケーション ファームウェアの書き込みが完了すると、ブートローダは制御を渡します。次にブートローダが呼び出されるまでの間は、通常のアプリケーションを実行できます。

AN1310 ブートローダの特長

*AN1310「PIC16/PIC18 デバイス用高速シリアル ブートローダ」*の主な特長は次の通りです。

- ファームウェア コードサイズが小さい (ほとんどのデバイスで 450 命令ワード未満)
- 自動的に baud レートをホストと同期
- 柔軟な baud レート - 1,200 bps ～ 3 Mbps、書き込み時間を大幅に短縮
- 16 ビット CRC パケットとフラッシュメモリの検証 - 書き込んだプログラムを低 baud レートでも素早く検証
- 先進の「書き込みプランナ」- 不要な消去 / 書き込みトランザクションを排除
- 「基本デバイス特性」データベース - 広範な PIC16/PIC18 デバイスをサポート
- アプリケーションの再マッピング (オプション) - リンカスクリプトの変更または割り込みサービス ルーチンの再マッピングは不要
- ブートローダの強制リエントリ メカニズム - 起動時の遅延がほとんどなく、I/O ピンまたはアプリケーション ファームウェア コードを追加しなくてもブートローダの再起動が可能
- $\overline{\text{MCLR}}$ リセット制御 (オプション) - ホスト PC アプリケーションによるデバイスの自動リセットをサポートし、信頼性の高いブートローダのリエントリを実現
- クロスプラットフォーム向けに C/C++ で PC ソフトウェアを書き換え、Qt℠ SDK - PC ソフトウェアのソースコードの再コンパイルによって Linux ホストをサポート
- PC ソフトウェアによる単純なシリアル ターミナル アプリケーション モード - ブートローダ ホストとシリアル ターミナル アプリケーションの切り換えにかかる無駄な時間を排除

Note: 上記の特長一覧を確認して別のブートローダが必要と判断された場合、**「その他の参考資料」**を参照してください。

前提条件

シリアル ブートローダを使用する場合、次の前提条件を満たす必要があります。

- コンフィグレーション ビットと PIC® マイクロコントローラのコンパイル / プログラミングに習熟している事
- シリアルポートを実装した開発ボードを PIC デバイスの USART1 RX/TX ピンに接続している事
- シリアルポートまたは USB- シリアル変換アダプタを PC に実装している事
- ブートローダ ファームウェアを初めて PIC デバイスに書き込む際に従来のプログラミング ツール (REAL ICE™ エミュレータ、PICkit™ 3、MPLAB® ICD 3 等) を使用する事
- MPLAB® IDE ソフトウェアをインストールしている事
- AN1310 高速シリアル ブートローダ ソフトウェアをインストールしている事

AN1310 高速シリアル ブートローダ ソフトウェア パッケージ (全ソースコードを含む) は、ウェブサイト www.microchip.com/applicationnotes からダウンロードできます。

1. [アプリケーション ノートの参照] ページが表示されたら、**[Select a Function]** メニューで [Programming & Bootloaders] から [Bootloader] を選択します。
2. **[Search]** ボタンをクリックします。
3. [AN1310] が表示されるまで下方向にスクロールし、[Resource Type] 列の圧縮ファイルアイコンをクリックします。

実装の基本手順

このセクションでは、ブートローダ / アプリケーションの開発に習熟している開発者向けに、ブートローダをセットアップして使用するための 3 つの基本ステップについて説明します。

続くセクション (下記) では、これらの基本ステップの概要と詳細を解説します。

- **「ファームウェアの概要」**
- **「ハードウェアの注意点」**
- **「ブートローダ モードの注意点」**
- **「アプリケーション モードの注意点」**
- **「ソフトウェア設計」**

ステップ 1: ブートローダ ファームウェアのコンパイルとプログラミング

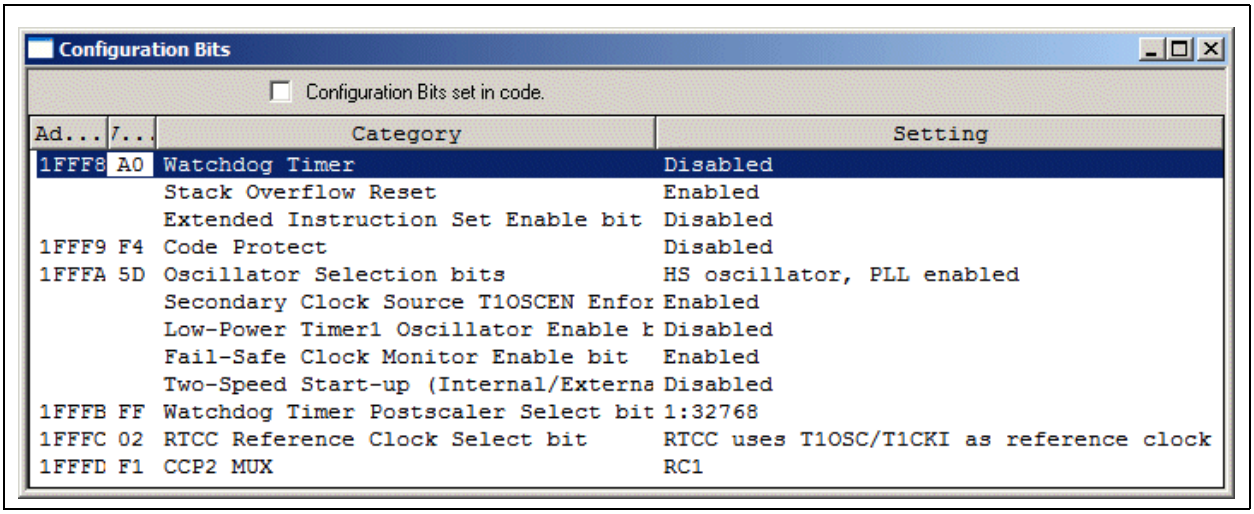
詳細は、次の各項を参照してください。

- **「ファームウェアの概要」**
- **「ハードウェアの注意点」**
- **「ブートローダ モードの注意点」**
- **「ソフトウェア設計」**

既定値では、シリアル ブートローダをインストールすると、以下のパスにブートローダ ファームウェアのソースコードが格納されます。

```
C:\Microchip
Solutions\SerialBootloader AN1310
vX.XX\PICxx Bootloader\
```

図 1: [CONFIGURATION BITS] ダイアログ ボックス



ブートローダ ファームウェアのコンパイルとプログラミングの手順 :

1. MPLAB IDE ソフトウェアで、PIC16 または PIC18 デバイスのブートローダ プロジェクトを開きます。
2. *[Configure] > [Select Devic...]* を選択し、使用する PIC デバイス (*[PIC18F87J90]* 等) を選択します。
3. コンフィグレーション ビットを設定するには、*[Configure] > [Configuration Bits...]* を選択します。

   図 1 に示すダイアログ ボックスが表示されます。
4. 各カテゴリで設定を行います。

   ブートローダを初めて実行するためのビットの推奨設定を表 1 に示します。
5. コンパイルを行い、ブートローダ ファームウェアをマイクロコントローラに書き込みます。

   ブートローダを PIC デバイスに書き込むには、ICSP™ プログラミング ツール (MPLAB® ICD 3 等) またはソケット付きプログラマ (MPLAB PM3 ユニバーサル デバイス プログラマ等) を使用する必要があります。

表 1: コンフィグレーション ビットの推奨設定

カテゴリ	設定
Watchdog Timer(1)	Disabled
Extended Instruction Set Enable bit	Disabled
Oscillator Selection bits	(ハードウェアに応じて選択。一般には高速にするほど baud レートを高くできる)
Fail-Safe Clock Monitor Enable bit	Enabled (選択可能な場合)
Low-Voltage Program (LVP)	Disabled (選択可能な場合)
Table Read-Protect	Disabled (選択可能な場合)

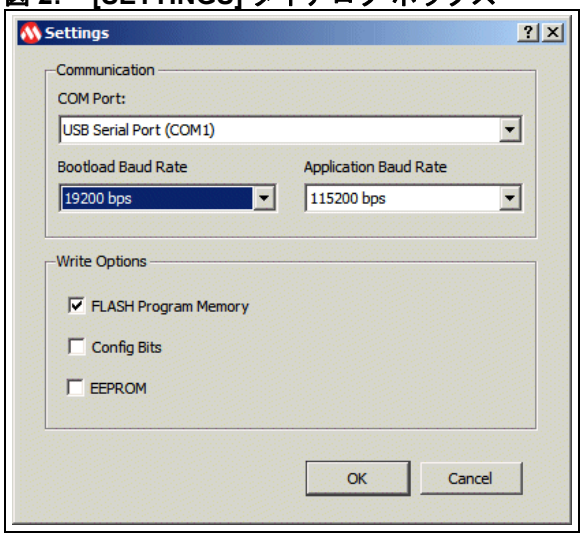
Note 1: ほとんどの PIC デバイスは、起動後にアプリケーション ファームウェアで再びウォッチドッグ タイマを有効化できます。

ステップ 2:
ホストとブートローダの接続

1. シリアル ブートローダ ホスト PC ソフトウェア (AN1310ui.exe) を起動します。
2. 初期設定の場合、*[Program] > [Settings]* を選択して、シリアルポートとブートローダの baud レートを設定します。

   図 2 に示すダイアログ ボックスが表示されます。

図 2: [SETTINGS] ダイアログ ボックス



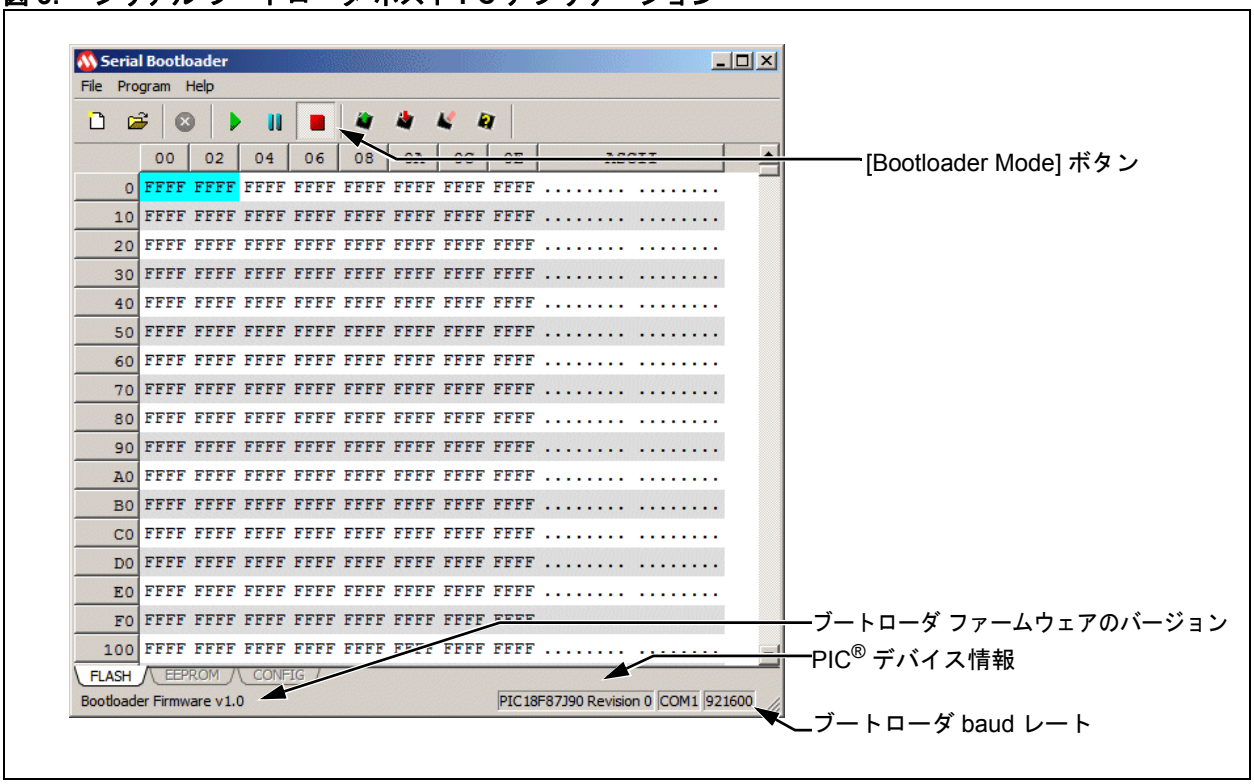
3. [COM Port] フィールドと [Bootloader Baud Rate] フィールドで値を選択し、**[OK]** をクリックします。

   [Bootloader Baud Rate] では、最初は問題なく通信できるように 19.2 kbps 等の適度な速度を選択します。全て正常に動作する事を確認してから標準的でない高速の baud レートを試す事を推奨します。

   このソフトウェアを使うと、書き込んだアプリケーション ファームウェアとシリアル通信を行う事ができます。アプリケーションによっては特定の baud レートが必要なため、[Application Baud Rate] 設定が用意されています。サンプル アプリケーション ファームウェア プロジェクトの場合、115,200 bps の baud レートが推奨されています。
4. PC のシリアルポートを PIC MCU 開発ボードに接続します。
5. 図 3 に示す赤いブートローダ ボタンをクリックするか、F4 キーを押して「ブートローダ」モードに移行します。

   PC ソフトウェアは、指定されたブートローダ baud レートで PIC ブートローダ ファームウェアとの通信を試みます。通信が確立すると、図 3 に示すように、PC にブートローダ ファームウェアのリビジョンと PIC デバイス情報が表示されます。

図 3: シリアル ブートローダ ホスト PC アプリケーション



ステップ 3:
アプリケーション ファームウェアの書き込み

詳細は、以下の各項を参照してください。

- **「アプリケーション モードの注意点」**
- **「ソフトウェア設計」**

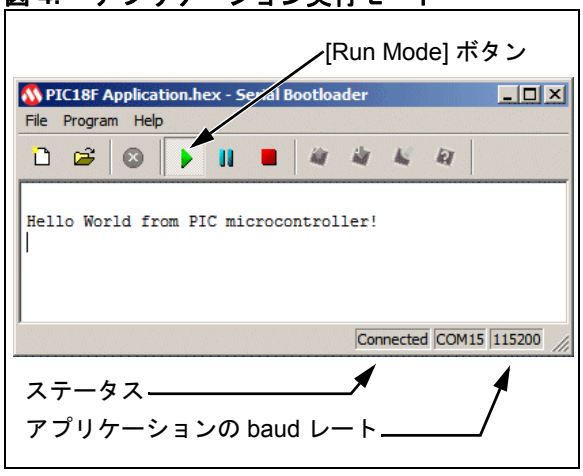
ホスト PIC をブートローダと接続すると、PIC デバイスに対して読み出し、書き込み、消去、検証を実行できます。サンプル アプリケーション ファームウェア プロジェクトは、以下のパスにインストールされます。

```
C:\Microchip Solutions\Serial
Bootloader AN1310 vX.XX\PICxx Application\
```

1. MPLAB IDE ソフトウェアでプロジェクトを開き、使用するデバイスを選択して設定し、アプリケーション ファームウェアをコンパイルします。
2. 生成されたアプリケーション　ファームウェアの HEX ファイルを、ホスト PC のシリアル ブートローダ ソフトウェアで開きます。
3. 赤い下向き矢印ボタンをクリックするか、F6 キーを押して、アプリケーション ファームウェアを PIC デバイスに書き込みます。
4. 緑の [Run Mode] ボタンをクリックするか、F2 キーを押して、アプリケーション ファームウェアを起動します。

   このモードでは、図 4 に示すように、PC ソフトウェアはシンプルなシリアル ターミナル アプリケーションとして動作します。PIC デバイスの USART1 ポート経由でアプリケーション ファームウェアと双方向テキスト通信を実行できます。

図 4:　アプリケーション実行モード



ファームウェアの概要

ファームウェアは、ブートローダ モードとアプリケーション モードの 2 つのモードで動作します。マイクロコントローラのリセット後、ブートローダ起動ルーチンは、ブートローダ コマンド ループに入るか (ブートローダ モード)、またはアプリケーション エントリ ポイント ベクタにジャンプするか (アプリケーション モード) を決定します。

外部からの介入がなく以下のいずれかの条件に該当する場合、自動的にブートローダ モードに移行します。どちらの条件にも該当しない場合、アプリケーション モードに移行します。

- マイクロコントローラにアプリケーション ファームウェア コードが書き込まれていない場合、ブートローダ モードに移行します。
- マイクロコントローラのリセット後、PIC デバイスの RX ピンの論理レベルが Low (RS-232 が「ブレーク」状態) の場合、ブートローダ モードに移行します。

ブートローダ モードには手動もしくはソフトウェア / ハードウェアで自動的に移行する事ができます。

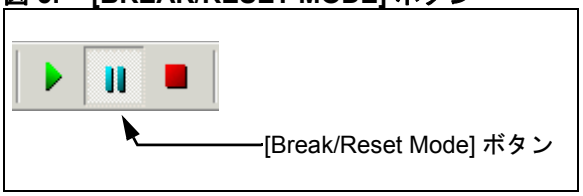
手動のブレーク / リセットによるリエントリ

ホスト PC ソフトウェアから PIC デバイスの RX ピンを RS-232「ブレーク」状態にする事ができます。これにより PIC デバイスの RX ピンは Low に保持され、この時にマイクロコントローラをリセットすると、ブートローダ モードに強制的に移行します。

手動でブートローダ モードに移行するには、下記手順を実行します。

1. 図 5 に示す青い [Break/Reset Mode] ボタンをクリックするか、F3 キーを押します。

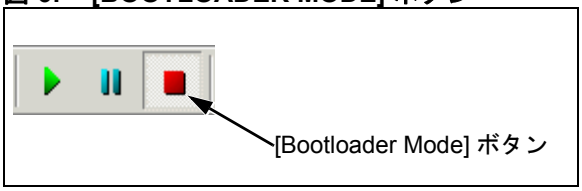
図 5:　[BREAK/RESET MODE] ボタン



2. ブートローダ起動ルーチンを実行するために、次のどちらかを実行して PIC デバイスをリセットします。
   - 開発ボードの $\overline{\text{MCLR}}$ リセットボタン (右図参照) を押します。
   - 一旦電源を切り、再度つなぎます。

   デバイスがリセットされ、ブートローダ起動ルーチンが RX ピンの「ブレーク」要求を検出して、ブートローダ モードに移行します。その際、デバイスにアプリケーション ファームウェアが書き込み済みであっても関係ありません。
3. 図 6 に示す赤い [Bootloader Mode] ボタンをクリックするか、F4 キーを押して、ブートローダに接続します。

図 6:　[BOOTLOADER MODE] ボタン



PC ソフトウェアは、ブートローダ baud レートでブートローダとの通信を試みます。通信が確立すると、図 4 に示したように、PC はブートローダ ファームウェアのリビジョンと PIC デバイス情報を受信します。

ソフトウェアによるブートローダのリエントリ

開発中にアプリケーション ファームウェアを何度も差分更新する場合、前述の手動によるリエントリ手順では手間がかかりすぎる場合があります。簡単な代替手順として、サンプル アプリケーション ファームウェア プロジェクトに実装済みの、ソフトウェアによるシンプルなリエントリ メカニズムを使用する方法があります。

例 1 に、そのメカニズムを示します。

例 1:　ソフトウェアによるブートローダのリエントリ

```
while(1)
{
  if(PIR1bits.RCIF)
  {
     if(RCSTAbits.FERR &&
  !PORTCbits.RC7)
     {
        // receiving BREAK
    // state, soft reboot
    // into Bootloader mode.
        Reset();
     }
  }
  (...)
}
```

このコードは、USART モジュールの状態を常時監視して、フレーミング エラーを検出します。フレーミング エラーが発生すると、コードは RXD ピンの論理レベルが Low に保持されている (ホスト PC が RS-232 ブレークステートを送信してブートローダのリエントリを要求している可能性が高い) かどうかを検証します。アプリケーションは、マイクロコントローラのソフトウェア リセットを開始して、制御をブートローダ起動ルーチンに渡します。

しかし、PIC16 マイクロコントローラにはソフトウェア リセット命令が存在しないため、アプリケーションはアドレス 0h のブートローダ起動ベクタにジャンプします。コールスタックに削除不可能な戻りアドレスが残る事を避けるため、アドレス 0h へのジャンプはアプリケーションの「main()」関数で実行する必要があります。

ハードウェアによるブートローダのリエントリ

ソフトウェアによるリエントリ手順は開発初期には便利ですが、信頼性の高い動作を実現するための手順としては推奨できません。アプリケーション コードにバグがある場合、アプリケーション ファームウェアがロックアップし、次回のコード変更時にブートローダを自動的にリエントリできない可能性があります。

また、アプリケーションによる通常のシリアル通信で実際にフレーミング エラーが発生した場合、意図しないリエントリによってブートローダ モードに移行する可能性があります。その場合、ユーザによる介入がないとアプリケーションを再起動できません。

より信頼性の高い、ハードウェアベースのブートローダ リエントリを行うには、シリアルポートの RTS 信号を PIC デバイスの MCLR リセット信号に接続します。この方法では、**「手動のブレーク / リセットによるリエントリ」**で説明したように、ホスト PC ソフトウェアが自動的にブレーク / リセット信号をアサートできます。

ハードウェアの注意点

図 7 は、シリアル ブートローダの代表的な接続を示します。

RTS 信号を使用した $\overline{\text{MCLR}}$ リセット制御は必須ではありません。しかし、RTS 信号を接続するとホスト PC はシリアルポートの RTS 信号を使用して PIC デバイスの MCLR ピンをプルダウンできます。これにより、ブートローダ モードに移行した時にホストPCソフトウェアから自動的にデバイスをリセットできます。

In-Circuit Serial Programming™ (ICSP™) モードに移行するために MCLR に高電圧 (V_{DD} よりも高い電圧) をかける必要がある PIC デバイスを使用する場合、RTS/MCLR ダイオードが従来の ICSP プログラミング / デバッグの妨げになる可能性があります。この場合、N チャンネル金属酸化膜半導体電界効果トランジスタ (MOSFET) の方が適している場合もあります (図 8 参照)。

一般に、RS-232 トランシーバ チップは、受信する TXD/RTS 信号に対するプルダウン抵抗器を内蔵しています。これにより PIC デバイスは、シリアルポートの DB9 コネクタが接続されていなくても、RX ピンの論理レベルが High (RS-232 が「アイドル」状態) である事を通常通りに確認できます。

図 7: RX ピン回路図

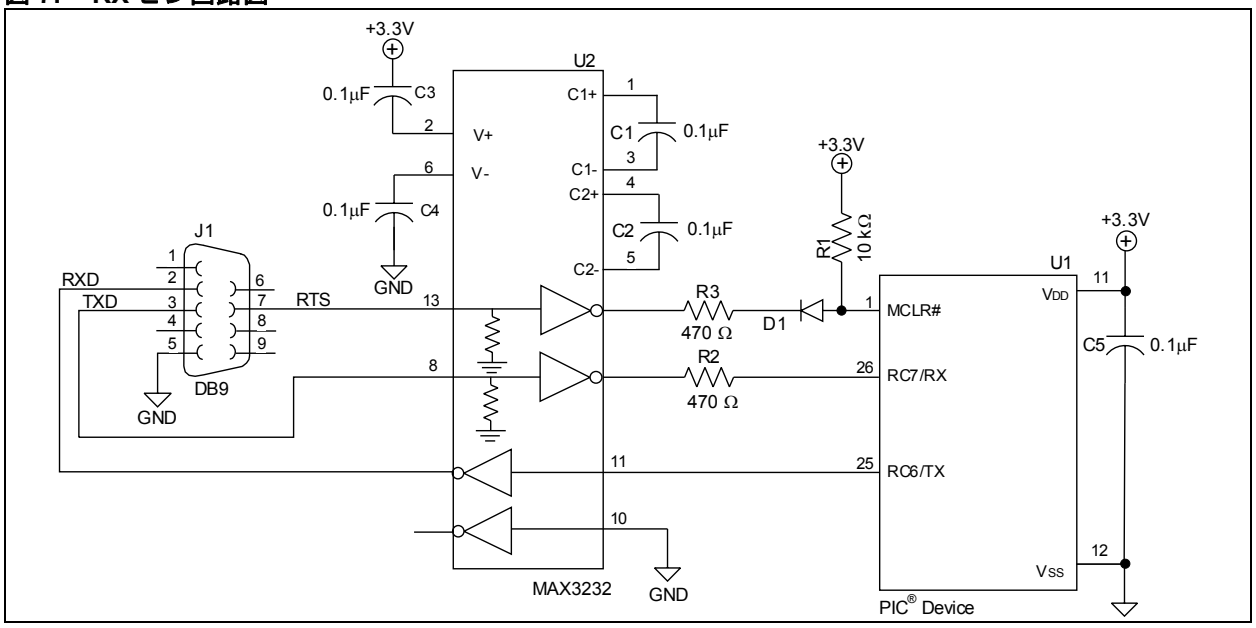
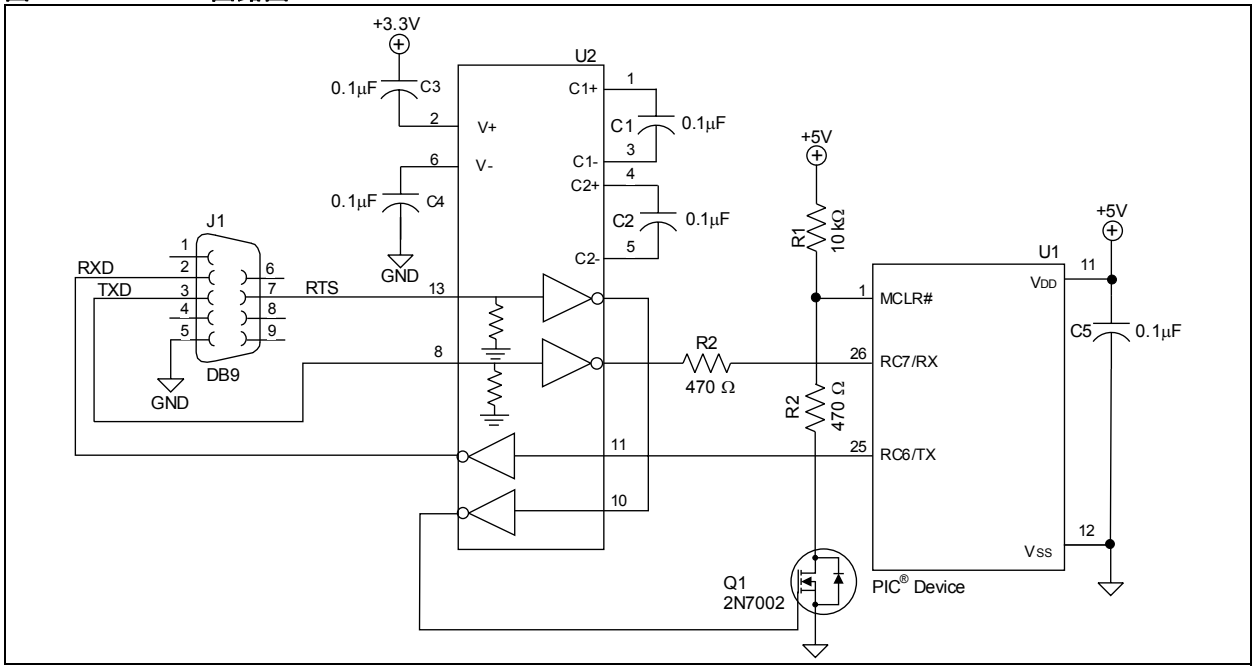


図 8: MOSFET 回路図



アプリケーション モードへの移行

デバイスが確実にアプリケーション モードに移行するように、以下の事を推奨します。

- RX ピンを High にして RS-232 が「アイドル」状態である事を示すために、デバイスを十分長い時間リセット状態に保持する。
- デバイスのリセット中とその直後は、マイクロコントローラにデータを伝送しない。

  受信データが誤って解釈されてRS-232が「ブレーク」状態であると判断され、ブートローダ起動ルーチンによってブートローダ モードに移行する可能性があります。

カスタム RS-232 トランシーバ回路を使用する場合、アイドル状態またはホスト PC のシリアルポートから切断時に、PIC デバイスの RX ピンを確実に VDD までプルアップする必要があります。PIC デバイスの RX ピンがフローティングのままか、またはプルダウンされた場合、誤ってブートローダ モードに移行する可能性があります。

高速 baud レート

ブートローダは、最大 3 Mbps の baud レートで動作可能です。高速 baud レート通信を確実に達成するには、ハードウェアの設計に関していくつか注意点があります。

従来の PC のシリアルポートは、115.2 kbps 以下の baud レートでしか動作しません。USB- シリアル コンバータ メーカーはより高い baud レートでも動作すると主張しますが、baud レートを 500 kbps 以下に制限するレベル変換器回路を組み込んでいる製品もあります。

RS-232 レベル変換器を使用しないで USB- シリアル コンバータ回路を直接 PIC デバイスに接続すると、より信頼性の高い高速動作を実現できます。

ハードウェア設計のもう 1 つの注意点は、PIC デバイスのクロック源の周波数をホストのシリアルポートで使用可能な baud レートと一致させる事です。高速になると、USB- シリアル コンバータが使用できる baud レートは限定される事があります。

例として、表 2 に 2 つの USB- シリアル コンバータで現在使用できる baud レートの一部を示します。詳細は、Prolific Technology Inc. 社 (PL2303) または Future Technology Devices International Limited社(FT232BM) にお問い合わせ頂くか、または使用しているコンバータのデータシートを参照してください。

表 2: USB- シリアルの高速 BAUD レートの例

PL2303	FT232BM
6,000,000	3,000,000
3,000,000	2,000,000
2,457,600	1,500,000
1,228,800	1,411,765
921,600	1,333,333
614,400	1,263,158
460,800	1,200,000
230,400	1,142,857
	1,000,000
	750,000
	500,000
	250,000

PIC デバイスが使用できる baud レートの種類も、使用する FOSC クロック源と PIC デバイスの baud レート計算式によって限定されます (表 3 を参照)。

表 3: PIC® デバイスの BAUD レート計算式

BRG16	BRGH	PIC baud レート
0	1	FOSC/[16 (BRG + 1)]
1	1	FOSC/[4 (BRG + 1)]

ブートローダ ファームウェアのソースコードは、「BRG16 = 1、BRGH = 1」のモードをサポートするマイクロコントローラ デバイスでは自動的にそのモードを使用します。これは最も柔軟性が高いモードで、最も広い範囲の baud レートを使用できます。

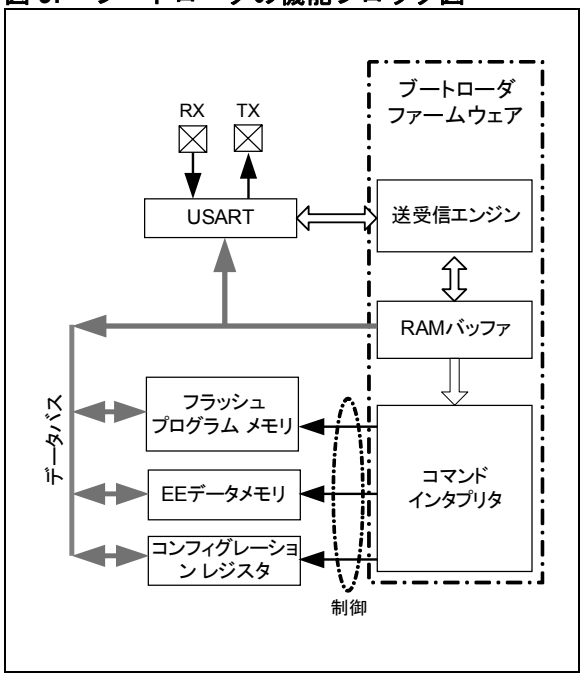
通信の失敗を避けるため、理想としては PIC デバイスとシリアルポートの baud レートを 3% 以内の誤差で一致させる必要があります。ブートローダ ファームウェアとの通信中に CRC エラーを検出すると、ホスト PC アプリケーションは停止してエラーステータスを表示します。

ブートローダ ファームウェアの動作

図9に、ブートローダの基本的な動作設計を示します。

USARTモジュール経由で受信したデータは、オーバーラン エラーの発生を防ぐためにすぐに RAM バッファに格納します。各パケットを受信するたびに、16 ビット CRC を使用してデータの整合性を検証します。

図 9: ブートローダの機能ブロック図



有効な要求パケットを受信すると、コマンド インタプリタがパケット内のコマンド番号を評価して、実行する操作 (消去、書き込み、読み出し、検証) を決定します。要求を実行した後、USART 経由で応答を返します。応答の内容は、タスク完了の肯定応答または (読み込み操作の場合は) デバイスメモリ データです。

ホスト PC アプリケーションは、1 度に複数のパケットを送信する事はできません。パケットを 1 つ送信すると、肯定応答を受信するまでは次のパケットを送信できません。これによってフロー制御が維持されます。

ブートローダ プロトコルの詳細は、**補遺 A19 ページの「ブートローダのプロトコル」**を参照してください。

パイプライン化された読み出し

このブートローダは、AN851 の従来のブートローダの説明と異なり、デバイス読み出し操作で RAM バッファを使用しません。

フラッシュ /EEPROM から読み出されたデータは、直接 USART モジュールに送信されます。この方法の主な利点は以下の通りです。

- 応答パケットサイズがマイクロコントローラの RAM サイズに制限されない。
- デバイスメモリ全体を 1 回の要求トランザクションで読み出す事ができ、トランザクションのオーバーヘッドを最小限に抑えられる。
- プロセッサコアと USART が同時に動作できるため (パイプライン化)、総クロック時間を削減できる。

USART 通信

マイクロコントローラの USART モジュールは、データの送受信に使用します。通信設定は次の通りです。

- 8 データビット
- パリティなし
- 1 スタート / ストップビット
- 可変 baud レート

最初の Start-of-Text 文字 (STX または 0Fh) のビットレートの計測には、自動 baud ルーチンを使用します。以前のブートローダと異なり、全パケットの先頭で自動baudルーチンを実行するわけではありません。baud レートの検出に成功すると、その値はブートローダ コマンド ループが無効な要求または予期しない要求を受信するまでロックされます。

baudレートをロックする事によって、低速な自動baud計算中でもデータを失わずに高 baud レートのデータを受信できます。無効な要求または予期しない要求を受信すると、自動 baud ルーチンを再実行します。

baud レートを切り換える際にブートローダ ファームウェアが誤った baud レートでロックされ、データストリームを受信しても自動 baud ルーチンの再実行がトリガされない場合があります。そのような場合、MCLR リセットにより自動 baud ルーチンを強制的に実行します。

$\overline{\text{MCLR}}$ の制御用に RTS 信号が接続されている場合、上記の状況はホスト PC アプリケーションが MCLR リセットをアサートする事によって自動的に処理できます。

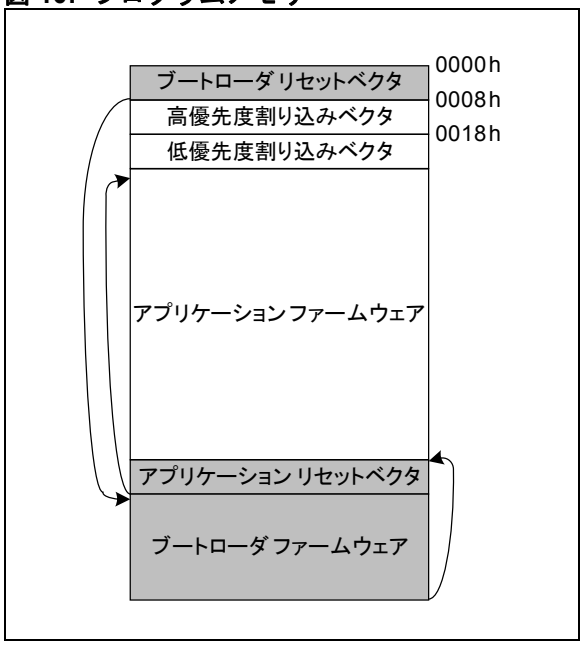
ブートローダ モードの注意点

既定値設定では、ブートローダ ファームウェアはフラッシュ プログラムメモリ空間の最後に格納されます。ブートローダをプログラムメモリ空間の最後に配置する事で、アプリケーション ファームウェアは通常のハードウェア割り込みベクタアドレスで割り込みを処理できます。この設定では、割り込みレイテンシを最小限に抑えられます。

ホスト PC ブートローダ アプリケーションは、リセットベクタ (アドレス 0000h) に最初の命令として「GOTO」を書き込みます。起動時は、この GOTO によってブートローダにジャンプします。アプリケーション ファームウェアの元々のリセットベクタ命令は、メモリのブートローダ ブロックの直前に自動的に移動します。ブートローダ ファームウェアは、ブートローダ モードが要求されていない場合、このアプリケーション リセットベクタを使用してアプリケーションを起動します (図 10 を参照)。

PIC18 デバイスでリセットベクタの自動再配置を行うには、アプリケーション ファームウェアがアドレス 0000h に最初の命令として GOTO 命令を書き込む必要があります。

図 10: プログラムメモリ



PIC16 デバイスでは、プログラムメモリはページ化されています。そのため、far GOTO を実行する前に PCLATH レジスタを読み込む必要があります。ホスト PC ブートローダ アプリケーションは、起動時にブートローダにジャンプするために、最大 4 つの命令をリセットベクタ (アドレス 0000h) に書き込みます。

その書き込みスペースを空けるために、ホスト PC ソフトウェアは自動的にアプリケーション ファームウェアの最初の 4 つの命令をブートローダ ファームウェアの直前に移動します。例 2 に、ソフトウェアによって有効なアプリケーション リセットベクタとして認識される PIC16 far GOTO コードシーケンスを示します。

例 2: PIC16 リセットベクタ

```
    ORG 0
ResetVector:
    movlw       high(FarApplication)
    movwf       PCLATH
    goto        FarApplication

    (...)

FarApplication:
    (...)
```

差分ブートロード

PIC デバイスのプログラミング完了後、ホスト PC アプリケーションはディスク上の HEX ファイルの監視を継続します。アプリケーション ファームウェアを変更して再コンパイルした場合、ホスト PC ブートローダ アプリケーションはすぐにその変更を検出します。これは、ホスト PC アプリケーションによるデバイス ファームウェアの差分更新のトリガになります。

差分更新は、前回書き込まれた HEX ファイルと新しいファイルを比較し、プログラムメモリのどのバイトが変更されたかを判断します。再プログラミングされるのは、変更を含むメモリブロックのみです。大規模なアプリケーション プロジェクトを開発する場合、この仕組みによって大幅な時間の節約と生産性の向上を実現できます。

書き込み保護

デバイスの更新に常にブートローダ ファームウェアを使用できる事を保証するには、フラッシュメモリ領域のブートローダ ブロック (ブートブロック) を書き込み保護するのが賢明な方法です。それには、ソフトウェアによる方法とハードウェアによる方法があります。

ソフトウェアによる書き込み保護

ソフトウェアでブートブロックを書き込み保護するには、ファイル bootconfig.inc で USE_SOFTBOOTWP オプションを有効にします。この機能を有効にすると、ブートローダ ファームウェアは消去操作と書き込み操作のたびに操作内容をチェックし、対象アドレスがブートブロック外である事を確認します。ブートブロック内を対象とする消去 / 書き込み操作はスキップされます。

コンフィグレーション ビットへの書き込みアクセスがある事は非常に強力ですが、危険でもあります。内部オシレータによる動作のためのハードウェアしかない設計で、誤って EC モードを使用するようにコンフィグレーション ビットを変更した場合、その後はブートローダ モードも含めた一切の操作を実行できない可能性があります。

インクルード ファイル bootconfig.inc で USE_SOFTCONFIGWP オプションを使用すると、ソフトウェアでコンフィグレーション ビットを書き込み保護できます。PIC16 デバイスには、コンフィグレーション ビットの書き込みアクセスはありません。

ハードウェアによる書き込み保護

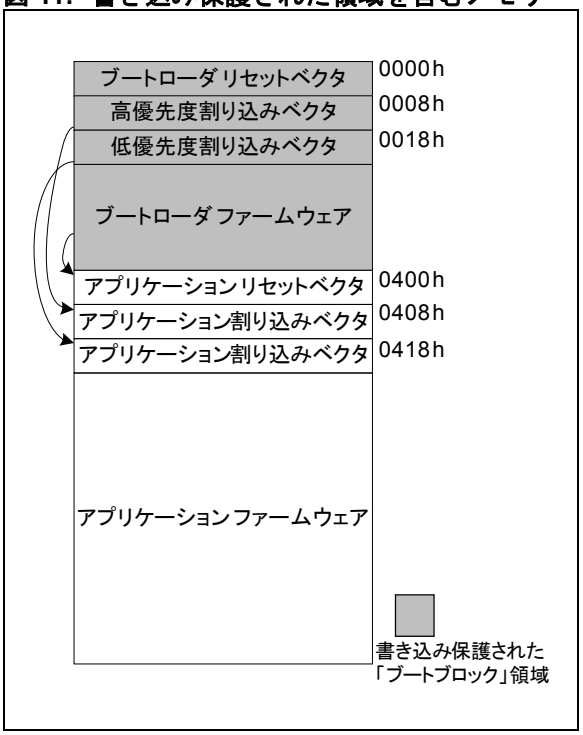
一部の PIC18 デバイス (PIC18F46J11 等) には、フラッシュメモリ領域の最後を書き込み保護できるコンフィグレーション ビットがあります。

しかしほとんどのデバイスは、フラッシュメモリ領域の先頭からの書き込み保護のみを提供します。このようなデバイスの場合、フラッシュメモリのアドレス 0h にブートローダを配置する必要があります。それには、ファイル bootconfig.inc で #define BOOTLOADER_ADDRESS 0 オプションを使用します。

ブートローダをアドレス 0h に配置すると、ブートローダがハードウェアのリセット ベクタアドレスと割り込みベクタアドレスを占有します。このレイアウトが機能するには、アプリケーション ファームウェアがリセットベクタと割り込みベクタを再マッピングする必要があります。ブートローダ ファームウェアは、ハードウェアのリセットイベントと割り込みイベントを、再マッピングされたアドレスのアプリケーション ファームウェアに「転送」します (図 11 を参照)。.

Note:	一部の PIC デバイスは、アプリケーション ファームウェアのプログラムメモリ領域を書き込み保護できるコンフィグレーション オプションを提供します。書き込み保護された領域は、ブートローダで変更できません。 書き込み保護された領域の既存の内容がアプリケーション ファームウェアの HEXファイルの新しいデータと一致しない場合、書き込み保護によってブートローダの書き込み操作が失敗する可能性があります。

図 11: 書き込み保護された領域を含むメモリ



ファイル bootconfig.inc は、ブートローダ ファームウェアがアプリケーション ベクタ AppVector、AppHighIntVector、AppLowIntVector を探す場所を定義します。特定のアプリケーションに合わせてフラッシュメモリの使用方法を最適化するために、これらのアドレスを調整できます。しかし、サンプル アプリケーション ファームウェア プロジェクトとの互換性を維持するために、これらのアドレスを既定値のままにします。

アプリケーション モードの注意点

フラッシュメモリにブートローダ コードを配置する場合、アプリケーション ファームウェアの変更が必要になる可能性があります。この変更を円滑に実行できるように、このアプリケーション ノートでは、全てのコンフィグレーションで「標準設定のまま」動作するソフトウェア メカニズムとサンプル アプリケーション ファームウェア プロジェクトを提供します。

ブートローダ ファームウェアをフラッシュメモリ領域の最後に配置する場合、ホスト PC ブートローダ ソフトウェアによって自動的にリセットベクタが再マッピングされるため、アプリケーション ファームウェアを変更する必要はほとんど (または全く) ありません。割り込みベクタは通常のハードウェア割り込みベクタアドレスでアプリケーション ファームウェアによって処理されるため、この設計ではアプリケーション ファームウェアを変更する必要はありません。このタイプの設計のメモリマップ例については、9 ページの**図 10** を参照してください。

ブートローダ コードをアドレス 0h に配置する場合、アプリケーション ファームウェアは、ハードウェアのリセットベクタと割り込みベクタの新しいアドレスへの再マッピングをサポートする必要があります。再マッピング後の新しいアドレスは、ハードウェア ベクタアドレスを占有するブートローダ ファームウェアによる「転送」で使用されます。このタイプの設計のメモリマップ例については、10 ページの**図 11** を参照してください。

このセクションでは、この両方のタイプの設計でブートローダ ファームウェアと共に動作するようにサンプル アプリケーション ファームウェア プロジェクトを変更した方法について説明します。

Note: サンプル アプリケーション ファームウェア プロジェクトは、PIC デバイス上の UART1 経由で通信します。ブートローダ ファームウェアと異なり、自動 baud コードは含まれません。

コンパイルの前に、クロック周波数と希望する baud レートに基づいてサンプル ソースコードを編集し、適切な baud レート生成器 (BRG) の値を設定します。ソースコードのコメント部分には、一般的に使用されるいくつかの値が記述されています。

MPLAB® IDE C18 アプリケーション

MPLAB IDE C18 コンパイラを使用してアプリケーション ファームウェアを開発する場合、C コンパイラが生成する先頭のリセットベクタ命令は、規定値では「GOTO」です。これにより、コードを変更せずにブートローダを使用できます。

C18 コンパイラは、プログラムメモリ領域の先頭からアプリケーション コードを作成し、断片化を最小限に抑えます。これにより、規定値でブートローダ ファームウェアがプログラムメモリ領域の最後に常駐するため、アプリケーション ファームウェア コードとの競合は発生しません。従って通常は、リンカスクリプトを変更してブートローダ ファームウェア用のプログラムメモリ領域を予約する必要はありません。

PIC18 上の MCC18 用再マッピング済みサンプル アプリケーション

PIC18 上の MCC18 用再マッピング済みサンプル アプリケーション ファームウェアのインストール先は次の通りです。

```
C:\Microchip
Solutions\SerialBootloader AN1310
vX.XX\PIC18 Application\MCC18 Remapped
Application\
```

このプロジェクトの C コードには、アドレス 0h に配置されたブートローダと動作するために必要な全ての変更が適用済みです。この変更は全て C コードで記述されています (例 3 を参照)。

例 3: PIC18 上の再マッピング済みアプリケーション C コード

```
// Prevent application code from
// being written into FLASH
// memory space needed for the
// Bootloader firmware at
// addresses 0 through 3FFh.
#pragma romdata BootloaderProgramMemorySpace = 0x6
const rom char bootloaderProgramMemorySpace[0x400 - 0x6];

// The routine _startup() is
// defined in the C18 startup
// code (usually c018i.c) and
// is usually the first code to be called by a GOTO at the
// normal reset vector of 0.

extern void _startup(void);

// Since the bootloader isn't
// going to write the normal
// reset vector at 0, we have
// to generate our own remapped
// reset vector at the address
// specified in the
// bootloader firmware.
#pragma code AppVector = 0x400
void AppVector(void)
{
    _asm GOTO _startup _endasm
}

// For PIC18 devices the high
// priority interrupt vector is
// normally positioned at
// address 0008h, but the
// bootloader resides there.
// Therefore, the bootloader's
// interrupt vector code is set
// up to branch to our code at
// 0408h.
#pragma code AppHighIntVector = 0x408
void AppHighIntVector(void)
{
    _asm GOTO high_isr _endasm              // branch to the high_isr()
                                            // function to handle priority
                                            // interrupts.
}
#pragma code AppLowIntVector = 0x418
void low_vector(void)
{
    _asm GOTO low_isr _endasm               // branch to the low_isr()
                                            // function to handle low
                                            // priority interrupts.
}
#pragma code                                // return to the default
                                            // code section
```

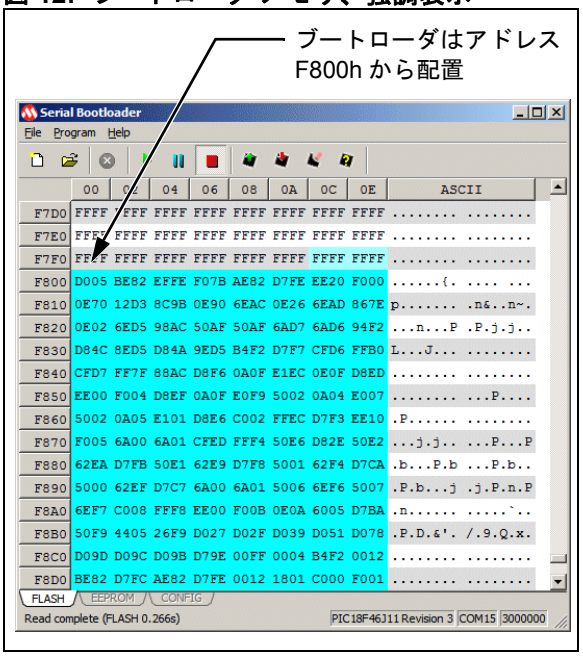
例 3 では、アプリケーション ベクタを再マッピングするためのアセンブリ言語ヘルパーファイル、プロジェクトのビルドオプション、カスタマイズされたリンカスクリプトを使用しません。全てをコードで処理します。

HI-TECH C® アプリケーション

HI-TECH C コンパイラは、しばしばアプリケーション コードを断片化して複数のプログラムメモリ領域に配置するため、ブートローダ ファームウェアとの競合が発生します。アプリケーション コードが同じデバイス プログラム フラッシュメモリを共有するブートローダ ファームウェアと競合するという問題を防ぐには、HI-TECH C プロジェクトを変更して、ブートローダ用のプログラムメモリ領域を予約する必要があります。

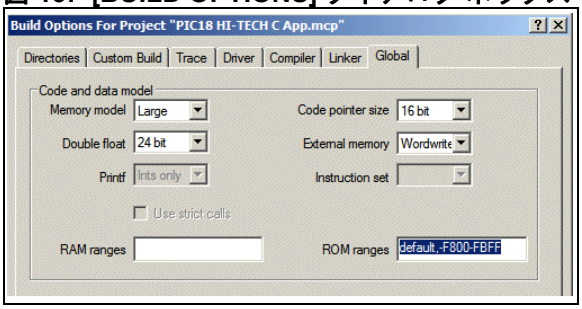
ブートローダ ファームウェアが使用するアドレスは、ホスト PC ブートローダ アプリケーションのフラッシュメモリ表示で確認できます。ブートローダに接続してフラッシュメモリ領域の最後までスクロールダウンすると、ブートローダ用に予約されている共有メモリ領域 (濃い青緑色) を確認できます (図 12 を参照)。

図 12: ブートローダ メモリ、強調表示



HI-TECH C プロジェクトの [Build Options] の [Global] タブで、ブートローダ領域を予約します(図13を参照)。

図 13: [BUILD OPTIONS] ダイアログ ボックス



[ROM ranges] フィールドを以下のように設定します。

```
default,-F800-FBFF
```

「default」は、デバイスのフラッシュ プログラムメモリ領域全体を使用するようにコンパイラに指示します。*マイナス記号付き*のアドレス範囲「-F800-FBFF」は、F800h ～ FBFFh のアドレス範囲をアプリケーション ファームウェアに使用しないようにコンパイラに指示します。

これで、アプリケーション コードとブートローダ コードの競合は発生しなくなります。

PIC16 上の HI-TECH C 用再マッピング済みサンプル アプリケーション

PIC16 デバイス上の HI-TECH C 用再マッピング済みサンプル アプリケーション ファームウェアの既定値のインストール先は以下の通りです。

```
C:\Microchip
Solutions\SerialBootloader AN1310
vX.XX\PIC16 Application\HI-TECH C Remapped
Application\
```

一目見るだけで、このプロジェクトの C コードが一般のアプリケーション コードに非常によく似ている事が分かります。ただし、以下に示すような多少の違いはあります。

- リセットベクタと割り込みベクタの再マッピングを処理するために、アセンブリ言語ヘルパーファイル (`isr.as`) が追加されます。

  このファイルは、ユーザのプロジェクトにコピーする必要があり、本来のリセット ベクタアドレス 0x400 以外のアドレスにカスタマイズできます。アドレス 0x400 は、ブートローダの AppVector 設定と一致する必要があります。

- [Codeoffset] オプションは 404 にリセットされます。

  [Codeoffset] の 16 進数は常に、アプリケーション リセットベクタに 4 を加算した値 (AppVector + 4) である必要があります。これにより、HI-TECH C コンパイラは、リセットベクタと割り込みベクタのコードを、フラッシュメモリ領域で既定値のアドレス 0 よりも後方に生成します。

  これを検証するには、MPLAB IDE でプロジェクトを開き、*[Project] > [Build Options...] > [Project]* を選択して **[Linker]** タブメニューを選択します。

- HI-TECH C コンパイラが RAM の 7Eh と 7Fh を使用しないようにするには、[RAM ranges] で `default,-7E-7F` を設定します。

  ブートローダは、ファイル `isr.as` のアセンブリ言語ヘルパーコードに、アクセスバンク RAM の 7Eh と 7Fh を使用してそれぞれ PCLATH レジスタと WREG レジスタを渡します。使用するアドレス番号は、ブートローダ ファームウェアのソースコードのアドレス定義 PCLATH_TEMP および W_TEMP と一致する必要があります。

  これを検証するには、MPLAB IDE でプロジェクトを開き、*[Project] > [Build Options...] > [Project]* を選択して **[Global]** タブメニューを選択します。

> **Note:** 拡張アーキテクチャ PIC16F デバイスには、割り込み処理中に ISR コンテキストを保存 / 復元する自動ハードウェアが実装されています。従って、拡張コア PIC16 デバイスでは、[RAM ranges] の設定は省略できます。

- HI-TECH C コンパイラがフラッシュメモリ領域のアドレス 0h からブートローダ ファームウェアの最後までの範囲を使用しないように、[ROM ranges] オプションを設定します。

  アプリケーション コードが使用できるフラッシュメモリ容量をできるだけ大きく確保するため、除外する範囲の最後のアドレスはブートローダ ファームウェアと書き込み保護されたブートブロックのサイズに応じて変更できます。

  これを検証するには、前出の **[Global]** タブメニューを参照します。

PIC18 上の HI-TECH C 用再マッピング済みサンプル アプリケーション

PIC18 上の HI-TECH C 用再マッピング済みサンプル アプリケーション ファームウェアの既定値のインストール先は以下の通りです。

```
C:\Microchip
Solutions\SerialBootloader AN1310
vX.XX\PIC18 Application\HI-TECH C Remapped
Application\
```

このプロジェクトのコードは、一般的なアプリケーション コードです。PIC16 デバイスとは異なり、アセンブリ言語ヘルパーファイルは必要ありません。ビルドオプションのみ設定が必要です。

1. *[Project] > [Build Options...] > [Project]* を選択して **[Linker]** タブメニューを選択します。

   [Codeoffset] フィールドには 400 が設定されています。

   HI-TECH C コンパイラがリセットベクタと割り込みベクタのコードを、フラッシュメモリ領域で既定値のアドレス 0h よりも後方に生成するには、このアドレスがブートローダで定義されている AppVector アドレスと等しい事が必要です。

2. **[Global]** タブメニューを選択し、[ROM ranges] フィールドを確認します。

   このオプションは、HI-TECH C コンパイラがフラッシュメモリ領域のアドレス 0h からブートローダ ファームウェアの最後までの範囲を使用しないように設定されています。

> **Note:** ブートローダ ファームウェアと書き込み保護されたブートブロックのサイズによっては、除外する範囲の最後のアドレスを調整して、アプリケーション コード用のフラッシュメモリ容量を増やす事ができます。ここでは、最後のアドレスは `default,-0-3FF` のままにしておきます。

PIC18 デバイスでは、RAM の範囲を除外する必要はありません。PIC18 ブートローダは、アプリケーションの割り込みベクタコードに制御を渡す時に、RAM を使用しないでコンテキスト データを保存します。

ソフトウェア設計

このセクションでは、ホスト PC ソフトウェアと、ブートローダ アプリケーションの書き込みプランニングについて説明します。

ホスト PC ソフトウェア

ホスト PC アプリケーションには、次のコマンドライン オプションを指定できます。

```
Serial Bootloader.exe [/e] [/p] [/v]
[filename.hex]
```

/e – デバイスを消去します。
/p – 指定した HEX ファイルをデバイスに書き込みます。
/v – デバイスと指定した HEX ファイルのデータが一致するかどうかを検証します。

ホスト PC ソフトウェアのビルドに使用したソフトウェア :

- Qt SDK 4.x – 劣等 GPL、変更せずダイナミックにリンク
- QextSerialPort 1.2 – パブリック ドメイン ソフトウェア、本アプリケーション向けに大幅変更
- SQLite 3.6.x – パブリック ドメイン ソフトウェア、変更せずに使用

オープンソースのパブリック ドメイン開発ツール / ライブラリのみを使用する事によって、ソフトウェアを Linux®、Macintosh® 等の他のオペレーティング システム プラットフォームに簡単に移植できます。本書の執筆時点では、このソフトウェアは、Linux 上でコンパイルして動作する事が確認されています。Macintosh でのテストは未実施です。

書き込みプランニング

AN851 で説明している従来のブートローダ アプリケーションでは、全メモリを消去してから新しいデータを全メモリに書き込むという単純なアルゴリズムを使用して新しいファームウェアを書き込んでいました。

PIC18F87J11 等の大容量メモリを搭載したデバイスの場合、フラッシュメモリ全体の消去に約 4 秒かかり、さらに 128 KB の新しいフラッシュデータの転送に 115.2 kbps で約 11 秒、低い baud レートではさらに長い時間がかかります。

新しいアプリケーション ファームウェアを開発する場合、一般的にアプリケーション ファームウェアに必要なメモリ容量よりもはるかに大容量のフラッシュメモリがターゲット PIC マイクロコントローラに搭載されています。従って、従来のブートローダの単純なアルゴリズムでは非常に効率が悪くなる可能性があります。

このシリアル ブートローダには、「書き込みプランニング」という別のアルゴリズムが組み込まれています。アプリケーション ファームウェアの HEX ファイルを解析し、消去と書き込みの 2 つのメモリ領域一覧を作成します。この 2 つの一覧から生成される「書き込みプラン」に従って、ソフトウェアはブートローダ ファームウェア カーネルに消去コマンドと書き込みコマンドを発行します。表 4 に書き込みプランの例を示します。

表 4: 書き込みプランの例

消去一覧	
開始アドレス	**終了アドレス**
1F800h	1F400h
1C00h	0h
20000h	1FC00h
書き込み一覧	
開始アドレス	**終了アドレス**
1FFC0h	20000h
0h	1AC0h
1F7C0h	1F800h

書き込みプランの作成

書き込みプランでは、次の処理が行われています。

1. フラッシュメモリ領域を書き込み一覧に追加します。その際、フラッシュ書き込みブロック境界でアライメントされます。メモリブロックが空 (NOP 等) である事を検出した場合、そのメモリブロックは書き込み一覧に追加しません。
2. 書き込み一覧をコピーして消去一覧を作成します。ただし、メモリアドレス領域は、フラッシュ消去ブロック境界でアライメントされます。

   消去一覧と書き込み一覧を使用すると、書き込み操作中に割り込みが発生した場合のフェイルセーフ動作として、最初に消去するメモリ領域または最初に書き込むメモリ領域を簡単に並べ替える事ができます。

   「J」ファミリデバイスではコンフィグレーション ビットはフラッシュメモリの最後に格納されますが、コンフィグレーション ビットを含むフラッシュ消去ブロックは最後に消去されるようにスケジュールされます。新しいコンフィグレーション ビットを含むフラッシュ書き込みブロックは、その直後の書き込みトランザクションとしてスケジュールされます。

   コンフィグレーション ビットの消去 / 書き込みをスケジュールする事により、コンフィグレーション ビットがデバイス上で空白になる可能性がある時間を最小限に抑えます。
3. フラッシュメモリは、消去ブロックアドレスの降順に消去されます。

   これにより、消去途中で割り込みが発生しても、次回のリセットでブートローダが部分的に消去されたアプリケーション ファームウェアの起動を試みる事はなく、ブートローダ モードに留まります。

書き込みプランの実行

書き込みプランを生成した後、消去一覧を使用して必要な全てのフラッシュメモリを消去し、書き込み一覧を使用して必要なフラッシュメモリを書き込みます。

前回のプログラミングで使用され、新しいファームウェアでは使用されないフラッシュメモリ領域に不要なデータが残らないようにするために、次の追加手順を実行します。

1. デバイス上とホスト PC 上の両方でフラッシュ消去ブロックごとに 16 ビット CCITT CRC を計算して、フラッシュメモリ領域全体を検証します。

   デバイス上のブロックに消去する必要がある不要データが残っていなければ、計算した CRC 値は一致するはずです。
2. ホスト PC はデバイス上で計算した CRC 値とホスト PC 上で計算した CRC 値を比較します。ホスト PC 側の CRC 値と一致させるために消去する必要があるブロックを、新しい消去一覧に追加します。ブロックを消去してもホスト PC 側の CRC 値と一致しない場合、エラーを表示します。
3. 新しい消去一覧を実行して、残っている古いファームウェアの不要データを消去します。

CRC 値を使用すると、全メモリの内容をホスト PC に戻すまたはフラッシュメモリ 全体を消去しなくても、ブートローダが不要データを処理できます。このアルゴリズムは、古いデータが残っているデバイスの再プログラミングにかかる時間を大幅に短縮します。

テーブル読み出し

一部の PIC デバイスには「テーブル読み出し保護」用のコンフィグレーション ビット オプションが用意されており、ブートローダのテーブル読み出し操作が妨げられる可能性があります。

ブートローダ ファームウェアは、次の操作でテーブル読み出しが実行可能である必要があります。

- フラッシュ プログラムメモリの内容の読み出し
- 最終パス消去のために不要データを検出する CRC 値の生成
- フラッシュメモリの内容の正しさを検証するための CRC 値の生成
- 基本デバイス特性データベース情報を検索するためのデバイス番号とリビジョン ID 番号の読み出し

このブートローダはテーブル読み出しを繰り返し使用する必要があるため、「テーブル読み出し保護」コンフィグレーション ビットを有効にする事は推奨しません。

コード保護

一部の PIC デバイスには、「コード保護」コンフィグレーション ビット オプションが用意されています。コード保護の目的は、ICSP™ によるデバイスメモリの内容の読み出しを禁止する事です。

ただし、PIC デバイスがこのブートローダ ファームウェアでプログラミングされている場合、コード保護を簡単に回避できます。このブートローダは、「コード保護」コンフィグレーション ビットが有効かどうかに関係なく、デバイスメモリの内容に対してテーブル読み出しを実行します。そのため、攻撃者はアプリケーションを読み出す事が可能です。

セキュリティが重要なアプリケーションの場合、*AN1157「A Serial Bootloader for PIC24F Devices」* (DS01157) の方が適している可能性があります。AN1157 の暗号化対応バージョンは、マイクロチップ社の販売代理店から注文できます。

新しいデバイスのサポート

機能を拡張した新しいマイクロコントローラのリリースに対応できるように、シリアル ブートローダはホスト PC に SQLite™ データベースを採用しています。PC ソフトウェアは、ブートローダ ファームウェアに接続すると、すぐにデバイス情報を確認します。データベースは、ホスト PC ソフトウェアを再コンパイルしなくても更新できます。

ブートローダ ファームウェアにも同じ適応能力が必要ですが、もちろんデータベースは使用できません。代わりに、インクルード ファイル `DEVICES.INC` を使用して、コンパイル時にデバイス固有情報を提供します。

この方法では、新しいデバイス情報をデータベースとインクルード ファイルに追加するだけで新しいデバイスのサポートできます。ただし、新しいデバイスとコード互換性がある事が前提です。

SQLite™ データベースの変更

既定値では、デバイス データベースに接続するための SQL ソースコードのインストール先は以下の通りです。

```
C:\Microchip
Solutions\SerialBootloader AN1310
vX.XX\Device Database\devices.sql
```

データベースには、現在 DEVICES と CONFIGWORDS の 2 つのテーブルがあります。

新しいデバイスに合わせて SQLite データベースを更新するには、以下の手順を実行します。

1. 例 4 に示すような SQL insert 文を使用して、DEVICES に 1 レコード追加します。

例 4: SQL INSERT 文

```
insert into DEVICES values (
     161, -- Device ID (in decimal)
     4,   -- Family ID (2 for PIC16, 4 for
PIC18)
     'PIC18F8722',
     2,  -- Bytes Per Word (FLASH)
     64, -- Write FLASH Block Size
     64, -- Erase FLASH Page Size
     '0x0', -- Start address of FLASH
     '0x20000', -- End address of FLASH
     '0xF00000', -- Start address of EEPROM
(use 0 if no EEPROM on your device)
     '0xF00400', -- End address of EEPROM
     '0x200000', -- Start address of User ID
(use 0 if no User ID space on your device)
     '0x200008', -- End address of User ID
     '0x300000', -- Start address of Config
Bits
     '0x30000E', -- End address of Config
Bits
     '0x3FFFFE', -- Start address of Device
Id
     '0x400000', -- End address of Device Id
     '0xFFE0', -- Device Id Mask (so that
Revision ID bits are ignored)
     '0x0', -- Start address of GPR
     '0xF60' -- End address of GPR (tells
the PC software how big packets can be
            --  without overflowing the
device buffer RAM)
);
```

2. 新しいデバイスがコンフィグレーション ヒューズを使用する場合、例 5 に示すように、各コンフィグレーション ワードについて CONFIGWORDS テーブルにレコードを追加します。

例 5: CONFIGWORDS への追加

```
insert into CONFIGWORDS values (
     161, -- Device ID
     4,   -- Family ID
     'CONFIG1H', -- Config Name
     3145729, -- Address
     '0x37', -- Default Value
     '0xCF'  -- Implemented Bits
);

insert into CONFIGWORDS values (
     161, -- Device ID
     4,   -- Family ID
     'CONFIG2L', -- Config Name
     3145730, -- Address
     '0xFF', -- Default Value
     '0x1F' -- Implemented Bits
);

(...)
```

これにより、検証操作から未使用のコンフィグレーション ワード ビットをマスクして除外する事ができ、誤って検証エラーが生じる可能性を排除できます。

3. ホスト PC ソフトウェアが実行時に使用するバイナリ データベース ファイル devices.db を再生成します。
   - 次のコマンド プロンプト パスに移動します。

```
C:\Microchip
Solutions\SerialBootloader AN1310
vX.XX\Device Database
```

   - 次のコマンドを実行します。

```
sqlite3.exe -init devices.sql -batch
..\devices.db .quit
```

更新後の devices.db ファイルは、ホスト PC シリアル ブートローダ実行ファイルと同じフォルダに置く必要があります。

DEVICES.INC インクルード ファイルの変更

新しいデバイスに合わせて DEVICES.INC ファイルを更新する手順 :

1. MPLAB IDE で、新しいデバイスに適したブートローダ ファームウェア プロジェクト ファイルを開きます。
2. DEVICES.INC ファイルを開きます。
3. 例 6 に示すように、新しいデバイスに関する #ifdef ブロックを新しく追加します。

   DEVICES.INC ファイルのデバイス番号は、SQL データベース スクリプト (例 5) で使用したデバイス番号と一致する必要があります。

例 6: インクルード ファイルへの追加

```
#ifdef __18F8722
    #define DEVICEID                .161
    #define WRITE_FLASH_BLOCKSIZE   .64
    #define ERASE_FLASH_BLOCKSIZE   .64
    #define END_FLASH 0x20000
    #define END_GPR 0xF60
#endif
```

デバイスデータの簡単な収集方法

上記例で必要だったデバイス番号とファミリ番号は、デバイス データシート、プログラミング仕様、リファレンス マニュアルで調べる事ができます。しかし、マイクロチップ社の「基本デバイス特性」XML ファイルを使用すると、それらの情報をより簡単に取得する事ができます。

これらの XML ファイルは、MPLAB IDE のインストール中にホスト PC 上に圧縮ファイルとして格納され、MPLAB プロセッサ固有ヘッダファイル、MPLAB コンフィグレーション画面、その他の項目の生成に使用されます。[Device Database] ユーティリティ プログラムでは、新しいデバイスの SQL 情報とインクルード情報を生成できます。

ユーティリティ使用手順 :

1. 以下のパスに移動して、XML ファイルを解凍します。

```
C:\Program Files\Microchip\MPLAB IDE\Device\*PIC.zip
```

2. 以下のパスに移動して、「Device Database.exe」を実行します。

```
C:\Microchip Solutions\Serial Bootloader AN1310 vX.XX\
```

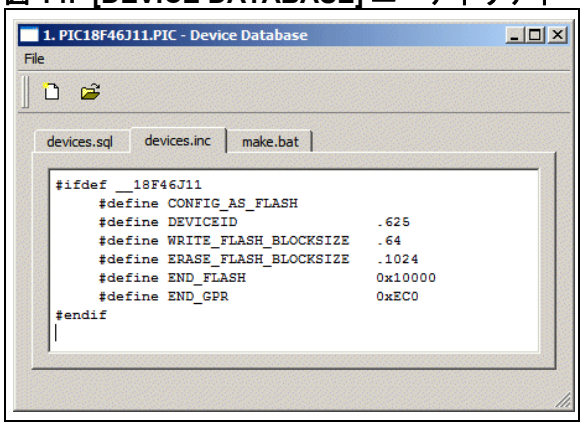
ユーティリティのダイアログ ボックスが表示されます。

3. *[File] > [Open PIC Definitions]* を選択し、使用するデバイスの PIC ファイルを選択します (例 : [PIC18F6520.PIC])。

複数のデバイスの情報を生成するには、Ctrl を押したままで 2 番目以降の PIC ファイル名を選択し、**[Open]** ボタンをクリックします。

図 14 に示すように、**[devices.inc]** タブにデバイス情報が表示されます。

図 14: [DEVICE DATABASE] ユーティリティ



参考資料

Brant Ivey、*AN1157「A Serial Bootloader for PIC24F Devices」*(DS01157)、Microchip Technology Inc.、2008

Ross M. Fosler and Rodger Richey、*AN851「A FLASH Bootloader for PIC16 and PIC18 Devices」*(DS00851)、Microchip Technology Inc.、2002

その他の参考資料

他のブートローダについては、以下のウェブサイトを参照してください。

- USB ブートローダ — http://www.microchip.com/usb
- TCP/IP ブートローダ — http://www.microchip.com/tcpip
- dsPIC® DSC と 32 ビット ブートローダ — http://www.microchip.com/pic32libraries

APPENDIX A: ブートローダのプロトコル

ブートローダは、信頼性が高く、使いやすく、簡単に実装できる基本通信プロトコルを採用しています。

パケット形式

ホスト PC からマイクロコントローラに送信されるデータパケットは全て、以下の基本パケット形式に従います。

```
[<STX>…]<STX>[<DATA>…]<CRCL><CRCH><ETX>[<ETX>]
```

<...> – 1 バイトを表す
[...] – 省略可能または可変個のバイトを表す

最大パケット長は、マイクロコントローラの RAM によって制限されます。マイクロコントローラがバッファ可能な量を超えるデータを送信しないように、ホスト PC ソフトウェアは使用するデバイスを判断し、基本デバイス特性データベースで RAM サイズを調べる必要があります。正常に機能する実装例として、サンプルホスト PC ソフトウェア ソースコード、Device.cpp、maxPacketSize() 関数を参照してください。

ブートローダ ファームウェアからホスト PC に返信される応答データパケットの多くは、同様の基本パケット形式に従います。

```
[<STX>…]<STX>[<DATA>…]<CRCL><CRCH><ETX>
```

最大応答パケット長は、コマンドパケットでホスト PC ソフトウェアが要求したブロック数によって制限されます。

制御文字

特別な意味を持つ 3 つの制御文字があります。そのうち <STX> と <ETX> の 2 つは、上記例で示しました。残りの 1 つは「データリンク拡張」<DLE> (16 進法で 0x05) です。

表 5: 制御文字

制御	16 進数	説明
<STX>	0Fh	テキスト開始 (Start of TeXt)
<ETX>	04h	テキスト終了 (End of TeXt)
<DLE>	05h	データリンク拡張 (Data Link Escape)

<DLE> は、制御文字として解釈可能なバイトを拡張するために使用します。ブートローダは、<DLE> に続くバイトを常にデータとして受け取り、制御文字の前に常に <DLE> を送信します。データペイロード バイトと CRC バイトは、データが制御文字と一致した場合は <DLE> 文字で拡張されます。

<STX>(テキスト開始)制御文字(16 進法で 0x0F)は、自動 baud、フロー制御、パケット フレーミングといった複数の目的で使用されます。

ブートローダ ファームウェアは、初期通信を確立するために、<STX> 文字のパルス幅を計測してホスト PC のデータ転送速度を計算します (自動 baud)。

ホスト PC は <STX> 文字をマイクロコントローラに送信し、マイクロコントローラから <STX> 文字がエコーバックされるまで、それを繰り返します。マイクロコントローラからエコーバックされた<STX>文字を受信するまで、ホスト PC はデータペイロード フィールドの送信を開始してはいけません。<STX> 文字のエコーバックを待たずにすぐにデータを送信した場合、マイクロコントローラ ファームウェアは自動 baud 等のタスクでビジー状態にあるため、データが失われる可能性があります。

<ETX>(テキスト終了)制御文字(16 進法で 0x04)は、パケットの終了を示します。また、ホスト PC ソフトウェアから <ETX> を追加で送信すると、baud レートがロックされたブートローダを強制的に自動 baud ルーチンに戻す事ができます。これは、ホスト PC で変更した baud レートに再同期するための時間を短縮し、信頼性を高める方法として使用できます。

巡回冗長チェック (CRC)

16 ビット CCIT CRC アルゴリズムは、受信したデータがシリアル通信リンクで破損していない事を保証します。16 ビット CRC は、最大 15 ビット長の不正データを数多く検出できますが、単純なチェックサム アルゴリズムはマルチビット エラーの検出では信頼できない可能性があります。

16 ビット CCIT CRC アルゴリズムは、組み込みアプリケーションで一般に使用されている MMC/SD カード SPI プロトコルで指定されている CRC アルゴリズムと同じです。使いやすい無償の CRC 計算機とソースコード生成器を、Thomas Pircher 氏の pycrc ツールから入手できます。

http://www.tty1.net/pycrc/

ブートローダ ファームウェアとホスト PC ソフトウェアは、制御文字を除くデータペイロードの各バイトに対して CRC を計算します。計算した CRC が送信された CRC と一致しない場合、パケットは破損していると認識され、破棄されます。

CRC アルゴリズムを使うと、フラッシュメモリが正しくプログラミングされているかどうかも簡単に検証できます。フラッシュメモリの全バイトを低速のシリアル通信リンク経由で送信するよりもずっと短い時間で、フラッシュメモリのブロックに対して CRC 値を計算できます。

ブートローダ コマンド

ブートローダ情報の読み出し

要求

```
<STX> <0x00> <CRCL><CRCH> <ETX>
```

PIC18 の応答

```
<STX> <BOOTBYTESL><BOOTBYTESH> <VERSIONL><VERSIONH> <COMMANDMASKH> <COMMANDMASKL:FAMILYID>
<STARTBOOTL><STARTBOOTH><STARTBOOTU><0x00> <CRCL><CRCH> <ETX>
```

PIC16 の応答

```
<STX> <BOOTBYTESL><BOOTBYTESH> <VERSIONL><VERSIONH> <COMMANDMASKH> <COMMANDMASKL:FAMILYID>
<STARTBOOTL><STARTBOOTH><STARTBOOTU><0x00> <DEVICEIDL><DEVICEIDH> <CRCL><CRCH> <ETX>
```

詳細

要求パケットの CRC は、分かりやすくするために、このパケットでは常に 0x0000 にします。

- VERSION は、ブートローダ ファームウェアのバージョン番号を定義します。
- BOOTBYTES は、ブートブロックのサイズ (バイト数) を指定します。
- STARTBOOT は、ブートブロックのフラッシュメモリ開始アドレスを指定します。

  ホストPCソフトウェアは、STARTBOOTとBOOTBYTESに基づいて、フラッシュメモリ領域を青緑色で表示します。
- COMMANDMASK は、PIC18 では現在未使用です。

  PIC16 の場合、デバイスにフラッシュ消去コマンドが実装されている場合は、COMMANDMASKH に 0x01 が格納されます。実装されていない場合は 0x00 が格納され、PIC16 デバイスが書き込み時に自動的に消去する事を表します (例 : PIC16F88X ファミリ)。
- FAMILYID は、使用中のマイクロコントローラの種類を表す最下位ニブルの 4 ビット数です。現在使用されているファミリ ID は、PIC16 が **2**、PIC18 が **4** です。
- DEVICEID は、PIC16 デバイスの場合にのみ送信されます。これはコンパイル時に決まる値で、使用されている PIC16FXXX 形式の製品番号を正確に示します。

  PIC18 製品は DEVICEID を送信する必要がありません。これは、ホスト PC アプリケーションが後述のフラッシュメモリ読み出しコマンドを使用して、デバイス ID をアドレス 0x3FFFFE のコンフィグレーション メモリから読み出す事ができるからです。

ホスト PC アプリケーションは、ファミリ ID とデバイス ID を使用して、基本デバイス特性データベースで正しいデバイス情報を調べます。

フラッシュメモリ読み出し

要求

```
<STX> <0x01> <ADDRESSL><ADDRESSH><ADDRESU><0x00> <BYTESL><BYTESH> <CRCH><CRCL> <ETX>
```

応答

```
<STX> [DATA…] <CRCL><CRCH> <ETX>
```

詳細

- ADDRESS は、読み出しを始める最初のフラッシュメモリ アドレスです。
- BYTES は、読み出すバイト数を指定します。

フラッシュメモリ CRC 読み出し

要求

```
<STX> <0x02> <ADDRESSL><ADDRESSH><ADDRESSU><0x00> <BLOCKSL><BLOCKSH> <CRCL><CRCH> <ETX>
```

応答

```
<STX>[<CRC1L><CRC1H>…<CRCnL><CRCnH>]<ETX>
```

詳細

- ADDRESS は、読み出しを始める最初のフラッシュメモリ アドレスです。
- BLOCKS は、生成する 16 ビット CRC ワード数を指定します。各 16 ビット CRC はフラッシュ消去ブロックのサイズ (バイト数) に対して生成されます。

  このコマンドの応答パケットは、データペイロードの CRC が含まれていないという点で、大部分のパケットと多少異なります。そのため、ブートローダ ファームウェアは同時に 2 つの異なる CRC を計算する必要がなく、メモリ使用量の節約と処理時間の短縮を実現できます。

フラッシュメモリ消去

要求

```
<STX> <0x03> <ADDRESSL><ADDRESSH><ADDRESSU><0x00> <PAGESL> <CRCL><CRCH> <ETX>
```

応答

```
<STX> <0x03> <CRCL><CRCH> <ETX>
```

詳細

- ADDRESS は、消去を始める最後のフラッシュメモリ アドレスです。消去はアドレスの*降順*に実行されます。これは、通常の操作と反対です。

  この機能は、ブートローダのフェイルセーフの強化に役立ちます。
- PAGES は、フラッシュメモリのフラッシュ消去ブロックサイズのページ数を指定します。

フラッシュメモリ書き込み

要求

```
<STX> <0x04> <ADDRESSL><ADDRESSH><ADDRESSU><0x00> <BLOCKSL> [<DATA>…] <CRCL><CRCH> <ETX>
```

応答

```
<STX> <0x04> <CRCL><CRCH> <ETX>
```

詳細

- ADDRESS は、書き込みを始めるフラッシュメモリ アドレスです。書き込みはアドレスの昇順に実行されます。
- BLOCKS は、フラッシュメモリのフラッシュ書き込みブロックサイズのチャンク数を指定します。

フラッシュメモリ セルは、次にそのセルに書き込む前に消去する必要があります。

EEPROM 読み出し

要求

```
<STX> <0x05> <ADDRESSL><ADDRESSH><0x00><0x00> <BYTESL><BYTESH> <CRCH><CRCL> <ETX>
```

応答

```
<STX> [DATA…] <CRCL><CRCH> <ETX>
```

詳細

- ADDRESS は、読み出しを始める最初の EEPROM アドレスです。
- BYTES は、読み出すバイト数を指定します。

EEPROM を搭載しないマイクロコントローラは、ダミーペイロード 0x05 を含む応答パケットをただちに送信します。

EEPROM 書き込み

要求

```
<STX> <0x06> <ADDRESSL><ADDRESSH><0x00><0x00> <BYTESL><BYTESH> [<DATA>…] <CRCL><CRCH> <ETX>
```

応答

```
<STX> <0x06> <CRCL><CRCH> <ETX>
```

詳細

- ADDRESS は、書き込みを始める EEPROM アドレスです。書き込みはアドレスの昇順に実行されます。
- BYTES は、EEPROM に書き込むバイト数を指定します。

EEPROM は、次に書き換えを行う前に消去する必要がないため、消去コマンドは存在しません。

EEPROM を搭載しないマイクロコントローラは、何もアクションを実行せず直ちに応答パケットを送信します。

コンフィグレーション ヒューズ書き込み

要求

```
<STX> <0x07> <ADDRESSL><ADDRESSH><0x00><0x00><BYTES> [<DATA>…] <CRCL><CRCH> <ETX>
```

応答

```
<STX> <0x07> <CRCL><CRCH> <ETX>
```

詳細

- ADDRESS は、書き込みを始めるコンフィグレーション アドレスです。書き込みはアドレスの昇順に実行されます。
- BYTES は、書き込むバイト数を指定します。

コンフィグレーション ビットは、次に書き換える前に消去する必要がないため、消去コマンドは存在しません。

アプリケーション ファームウェアの実行

要求

```
<STX> <0x08> <CRCL><CRCH> <ETX>
```

応答

なし

詳細

このコマンドによって、ブートローダはアプリケーション ファームウェアのリセットベクタにジャンプします。アプリケーション ファームウェアが起動するとブートローダはアクティブではなくなるため、応答は送信されません。

マイクロチップ社製デバイスのコード保護機能に関して次の点にご注意ください。

- マイクロチップ社製品は、該当するマイクロチップ社データシートに記載の仕様を満たしています。
- マイクロチップ社では、通常の条件ならびに仕様に従って使用した場合、マイクロチップ社製品のセキュリティ レベルは、現在市場に流通している同種製品の中でも最も高度であると考えています。
- しかし、コード保護機能を解除するための不正かつ違法な方法が存在する事もまた事実です。弊社の理解では、こうした手法はマイクロチップ社データシートにある動作仕様書以外の方法でマイクロチップ社製品を使用する事になります。このような行為は知的所有権の侵害に該当する可能性が非常に高いと言えます。
- マイクロチップ社は、コードの保全性に懸念を抱くお客様と連携し、対応策に取り組んでいきます。
- マイクロチップ社を含む全ての半導体メーカーで、自社のコードのセキュリティを完全に保証できる企業はありません。コード保護機能とは、マイクロチップ社が製品を「解読不能」として保証するものではありません。

コード保護機能は常に進歩しています。マイクロチップ社では、常に製品のコード保護機能の改善に取り組んでいます。マイクロチップ社のコード保護機能の侵害は、デジタル ミレニアム著作権法に違反します。そのような行為によってソフトウェアまたはその他の著作物に不正なアクセスを受けた場合は、デジタル ミレニアム著作権法の定めるところにより損害賠償訴訟を起こす権利があります。

商標

本書は再生紙を使用しています。

ISBN: 978-1-60932-189-5

マイクロチップ社では、Chandler および Tempe (アリゾナ州)、Gresham (オレゴン州) の本部、設計部およびウェハー製造工場そしてカリフォルニア州とインドのデザインセンターが ISO/TS-16949:2002 認証を取得しています。マイクロチップ社の品質システム プロセスおよび手順は、PIC® MCU および dsPIC® DSC、KEELOQ® コード ホッピング デバイス、シリアル EEPROM、マイクロペリフェラル、不揮発性メモリ、アナログ製品に採用されています。さらに、開発システムの設計と製造に関するマイクロチップ社の品質システムは ISO 9001:2000 認証を取得しています。

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
ISO/TS 16949:2002

各国の営業所とサービス

北米

本社
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
技術サポート :
http://support.microchip.com
URL:
www.microchip.com

アトランタ
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

ボストン
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

シカゴ
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

クリーブランド
Independence, OH
Tel: 216-447-0464
Fax: 216-447-0643

ダラス
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

デトロイト
Farmington Hills, MI
Tel: 248-538-2250
Fax: 248-538-2260

ココモ
Kokomo, IN
Tel: 765-864-8360
Fax: 765-864-8387

ロサンゼルス
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608

サンタクララ
Santa Clara, CA
Tel: 408-961-6444
Fax: 408-961-6445

トロント
Mississauga, Ontario,
Canada
Tel: 905-673-0699
Fax: 905-673-6509

アジア / 太平洋

アジア太平洋支社
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon
Hong Kong
Tel: 852-2401-1200
Fax: 852-2401-3431

オーストラリア - シドニー
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

中国 - 北京
Tel: 86-10-8528-2100
Fax: 86-10-8528-2104

中国 - 成都
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

中国 - 重慶
Tel: 86-23-8980-9588
Fax: 86-23-8980-9500

中国 - 香港 SAR
Tel: 852-2401-1200
Fax: 852-2401-3431

中国 - 南京
Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

中国 - 青島
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

中国 - 上海
Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

中国 - 瀋陽
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

中国 - 深圳
Tel: 86-755-8203-2660
Fax: 86-755-8203-1760

中国 - 武漢
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

中国 - 西安
Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

中国 - 厦門
Tel: 86-592-2388138
Fax: 86-592-2388130

中国 - 珠海
Tel: 86-756-3210040
Fax: 86-756-3210049

アジア / 太平洋

インド - バンガロール
Tel: 91-80-3090-4444
Fax: 91-80-3090-4123

インド - ニューデリー
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

インド - プネ
Tel: 91-20-2566-1512
Fax: 91-20-2566-1513

日本 - 横浜
Tel: 81-45-471- 6166
Fax: 81-45-471-6122

韓国 - 大邱
Tel: 82-53-744-4301
Fax: 82-53-744-4302

韓国 - ソウル
Tel: 82-2-554-7200
Fax:82-2-558-5932 または
82-2-558-5934

マレーシア - クアラルンプール
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

マレーシア - ペナン
Tel: 60-4-227-8870
Fax: 60-4-227-4068

フィリピン - マニラ
Tel: 63-2-634-9065
Fax: 63-2-634-9069

シンガポール
Tel: 65-6334-8870
Fax: 65-6334-8850

台湾 - 新竹
Tel: 886-3-6578-300
Fax: 886-3-6578-370

台湾 - 高雄
Tel: 886-7-536-4818
Fax: 886-7-536-4803

台湾 - 台北
Tel: 886-2-2500-6610
Fax: 886-2-2508-0102

タイ - バンコク
Tel: 66-2-694-1351
Fax: 66-2-694-1350

ヨーロッパ

オーストリア - ヴェルス
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

デンマーク - コペンハーゲン
Tel: 45-4450-2828
Fax: 45-4485-2829

フランス - パリ
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

ドイツ - ミュンヘン
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

イタリア - ミラノ
Tel: 39-0331-742611
Fax: 39-0331-466781

オランダ - ドリューネン
Tel: 31-416-690399
Fax: 31-416-690340

スペイン - マドリッド
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

イギリス - ウォーキンガム
Tel: 44-118-921-5869
Fax: 44-118-921-5820

01/05/10