応用記述

AVR947：自己ﾌﾟﾛｸﾞﾗﾐﾝｸﾞ能力を持つ任意MCU用単線ﾌﾞｰﾄﾛｰﾀﾞ

Atmel 8ﾋﾞｯﾄ ﾏｲｸﾛ ｺﾝﾄﾛｰﾗ

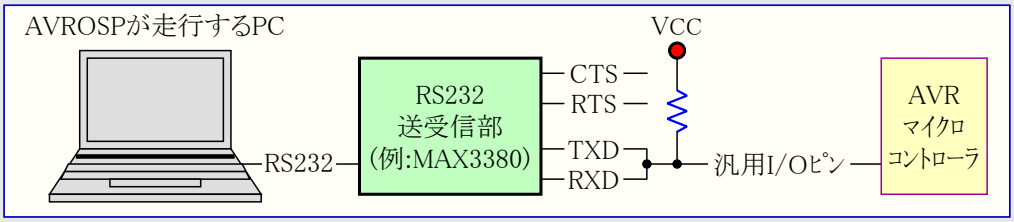
要点

- ﾌﾞｰﾄﾛｰﾀﾞ領域不要
- 通信周辺機能部署不要
- 1つの汎用入力ﾋﾟﾝだけ必要
- (AVROSPに基づいて)提供されるﾌﾞｰﾄﾛｰﾀﾞPCｲﾝﾀｰﾌｪｰｽ ｽｸﾘﾌﾟﾄ
- 容易なﾌﾞｰﾄﾛｰﾀﾞの開始と抜け出し
- 1371ﾊﾞｲﾄの小さなｺｰﾄﾞ使用量
- ﾌﾟﾛｸﾞﾗﾐﾝｸﾞ後のCRC検証
- ﾌﾞｰﾄﾛｰﾀﾞ動作形態でﾃﾞﾊﾞｲｽを休止に置くことが可能
- 現場ﾌｧｰﾑｳｪｱ更新を支援
- ﾌﾞｰﾄﾛｰﾀﾞ ﾌｧｰﾑｳｪｱの上書き保護

序説

この応用記述は独立したﾌﾞｰﾄﾛｰﾀﾞ領域とどんなﾊｰﾄﾞｳｪｱ通信部署もないﾃﾞﾊﾞｲｽのために特に設計されたﾌﾞｰﾄﾛｰﾀﾞを検討します。この設計はTWI,SPI,UART部署のような、どのﾁｯﾌﾟ資源も必要としません。代わりに、1つの汎用入出力ﾋﾟﾝだけを必要とする、(Atmel® 「AVR®274:単線ｿﾌﾄｳｪｱUART」応用記述で記述されるような)ｿﾌﾄｳｪｱでの単線UARTを実装します。これは制限された汎用ﾋﾟﾝ数を持つ安価なAVRﾃﾞﾊﾞｲｽに対して特に有用です。けれども、この設計はﾁｯﾌﾟでの自己ﾌﾟﾛｸﾞﾗﾐﾝｸﾞだけでなく或る種の不揮発性記憶(内部EEPROM)が必要です。この設計では図1.で示されるようにﾌﾞｰﾄﾛｰﾄﾞ(設定)する応用はAVR目的対象に接続されるPCに基づく応用です。

図1. PCとの接続



目次

1. 理屈

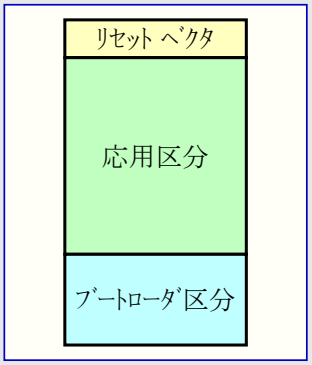
本章はｼｽﾃﾑ構造に沿ってﾏｲｸﾛ ｺﾝﾄﾛｰﾗ ﾌﾞｰﾄﾛｰﾀﾞ実装の詳細な記述を与えます。

1.1. ﾏｲｸﾛ ｺﾝﾄﾛｰﾗ ﾌﾞｰﾄﾛｰﾀﾞて何、そしてそれはどう動くの?

一般的に、ﾏｲｸﾛ ｺﾝﾄﾛｰﾗをﾌﾟﾛｸﾞﾗﾐﾝｸﾞするには目的対象ﾃﾞﾊﾞｲｽを支援する書き込み器が必要です。高価になることを除き、書き込み器はﾏｲｸﾛ ｺﾝﾄﾛｰﾗと書き込み器の間で多すぎる相互接続が必要です。出荷した製品のﾌｧｰﾑｳｪｱ更新はﾌﾟﾛｸﾞﾗﾐﾝｸﾞ ﾎﾟｰﾄの利用不能性のために面倒くさいものです。例え研究室でも、各作業台に対して書き込み器を得ることは実用的でないかもしれず、最良の解決策はﾌﾟﾛｸﾞﾗﾑ ﾒﾓﾘ内のﾌﾞｰﾄﾛｰﾀﾞで既に一旦ﾌﾟﾛｸﾞﾗﾐﾝｸﾞされたﾏｲｸﾛ ｺﾝﾄﾛｰﾗ試供品を提供することです。

図1-1.は応用とﾌﾞｰﾄの区域に分けられたﾌﾗｯｼｭ領域での簡潔な考えを与えます。

図1-1. ﾌﾗｯｼｭ ﾒﾓﾘ割り当て



注: これらの区分に対するｱﾄﾞﾚｽは使用者によるﾌﾞｰﾄﾛｰﾀﾞと応用のﾘﾝｶ ﾌｧｲﾙで定義されます。

注意する別の重要な点は、この構造でのﾘｾｯﾄ事象後、常にﾌﾞｰﾄﾛｰﾀﾞ ﾌﾟﾛｸﾞﾗﾑを始めるためにﾌﾞｰﾄﾛｰﾀﾞによってﾘｾｯﾄ ﾍﾞｸﾀが乗っ取られることです。ﾌﾞｰﾄﾛｰﾀﾞは書き込み器によって一度書かれ、ﾃﾞﾊﾞｲｽをﾌﾟﾛｸﾞﾗﾐﾝｸﾞするためにそれが必要とされる時に必ず応用をﾌﾟﾛｸﾞﾗﾐﾝｸﾞするのがﾌﾞｰﾄﾛｰﾀﾞです。

一旦ﾌﾞｰﾄﾛｰﾀﾞが書かれると、各々のｲﾝﾀｰﾌｪｰｽ(UART)は伝統的な書き込み器の使用なしに残りのﾌﾗｯｼｭ領域を再書き込みするのに使用することができます。この応用記述では更新処理を始めるためにﾌﾞｰﾄﾛｰﾀﾞが単線UARTｲﾝﾀｰﾌｪｰｽとEEPROMで設定されるﾌﾞｰﾄ ﾌﾗｸﾞ($2A)を使用します。ﾘｾｯﾄ後、ﾌﾞｰﾄﾛｰﾀﾞ ﾌｧｰﾑｳｪｱはEEPROMﾌﾗｸﾞの状態を調べ、設定を見つけた場合に形態設定されたﾌﾞｰﾄ領域内のﾌﾟﾛｸﾞﾗﾑが実行を開始し、さもなければ形態設定された応用領域内のﾌﾟﾛｸﾞﾗﾑが実行を開始します。ﾌﾞｰﾄﾛｰﾀﾞは応用領域書き込みと飛び実行のため独立した命令を持ちます。飛びが実行されない場合、例え応用領域が書かれても、応用は決して実行しません。

1.2. ｼｽﾃﾑ構造

この応用記述はAtmel ATmega16HVAﾃﾞﾊﾞｲｽに基づき、検査されたﾌｧｰﾑｳｪｱ設計を使用して単線ﾌﾞｰﾄﾛｰﾀﾞ実装の説明を試みます。この応用記述と共に提供されるｿﾌﾄｳｪｱは以下を含みます。

1.2.1. ﾌﾞｰﾄﾛｰﾀﾞ ﾌｧｰﾑｳｪｱ ｿｰｽ ｺｰﾄﾞとｺﾝﾊﾟｲﾙされたﾊﾞｲﾅﾘ

供給されるﾌﾞｰﾄﾛｰﾀﾞはIAR Embedded Workbenchを使用して書かれ、$3A00で始まる上位ﾌﾗｯｼｭ ｱﾄﾞﾚｽに属すように形態設定されます。これは_..X_FLASH_BASEを_..X_BOOT_STARTで置換し、ﾘﾝｶ ﾌｧｲﾙで_..X_BOOT_STARTを0x3A00に設定することによって行われます。ATmega16HVAのﾘﾝｶ ﾌｧｲﾙ内の以下のｺｰﾄﾞ部分は同じことを行います。

```
// ﾌﾞｰﾄﾛｰﾀﾞ
-Z(CODE)INTVEC=0-(_..X_INTVEC_SIZE-1)
-D_..X_BOOT_START=3A00                          // 14K境界、換言すると最終2Kﾊﾞｲﾄ
-Z(CODE)NEAR_F=_..X_BOOT_START-_..X_FLASH_NEND
-Z(CODE)SWITCH=_..X_BOOT_START-_..X_FLASH_NEND
-Z(CODE)DIFUNCT=_..X_BOOT_START-_..X_FLASH_NEND
-Z(CODE)CODE=_..X_BOOT_START-_..X_FLASH_END
-P(CODE)FAR_F=[_..X_BOOT_START-_..X_FLASH_END]/10000
-Z(CODE)INITTAB=_..X_BOOT_START-_..X_FLASH_END
-Z(CODE)HUGE_F=_..X_BOOT_START-_..X_FLASH_END
-Z(CODE)TINY_ID=_..X_BOOT_START-_..X_FLASH_END
-Z(CODE)NEAR_ID=_..X_BOOT_START-_..X_FLASH_END
-Z(CODE)CHECKSUM#_..X_FLASH_END
```

ﾌﾞｰﾄﾛｰﾀﾞはbootflagとしてEEPROMの最終ﾊﾞｲﾄ、ｱﾄﾞﾚｽ$00FFを使用します。このﾊﾞｲﾄの値が$2Aなら、ﾌﾞｰﾄﾛｰﾀﾞ ﾌﾗｸﾞが設定されたことを意味し、制御はﾌﾞｰﾄﾛｰﾀﾞに行くべきです。このﾌﾗｸﾞが他のどれかの値なら、制御が使用者応用へ行くべきことを意味します。下は最後のﾒﾓﾘ位置に$2Aを書いた(ﾌﾞｰﾄﾛｰﾀﾞで供給される)EEPROM.hexからの最後の行です。

```
:1000F000000000000000000000000000000002AD6
```

ﾌﾞｰﾄﾛｰﾀﾞと供給されるEEPROM2進ﾌｧｲﾙは支援される書き込み器でﾃﾞﾊﾞｲｽ内に書かれなければなりません。

例のﾌﾞｰﾄﾛｰﾀﾞは単線UARTｲﾝﾀｰﾌｪｰｽ用にPC0ﾋﾟﾝを使用します。このﾋﾟﾝとそれの定義は"Common_Define.h"のｺｰﾄﾞの以下の行によって定義されます。

```
#define SWUART_PORT_REG      PORTC
#define SWUART_PIN_REG       PINC
#define SWUART_PIN           PC0
```

1.2.2. 仮の応用ﾌｧｰﾑｳｪｱ ｿｰｽ ｺｰﾄﾞ

供給される仮の応用"16HVADummy"は付随するﾌﾞｰﾄﾛｰﾀﾞと共に使用できる独自応用開発用の参照基準として使用することができる使用者応用例です。提供される目的対象応用はAtmel ATmega16HVAﾃﾞﾊﾞｲｽのPB0ﾋﾟﾝを交互切り替えします。これはｱﾄﾞﾚｽ$0000～$0065に存在し、ﾌﾞｰﾄﾛｰﾀﾞと共に使用されるのに必要とされる全ての変更を持ちます。必要とされる変更は7.2.項で詳細に検討されます。この仮の応用のHEXﾌｧｲﾙはﾌﾞｰﾄﾛｰﾀﾞを使用することによって書くことができます。

1.2.3. 変更されたAVROSPソース コード

PCソフトウェア インターフェースはこのブートローダに命令を送ることが必要とされます。この目的のために(Atmel「AVR911:AVR公開ソース書き込み器」応用記述で提供されるソフトウェアに基づいて)変更されたAVROSPコードがこの応用記述と共に提供されます。このソフトウェアは供給されるブートローダとだけ適合し、最終使用者によって選んだデバイス用に形態設定されることだけが必要です。必要とされるAVROSPの変更については7.4.1.項を参照してください。この一括と共に提供されるAVROSP実行可能UART_PROG.exeは既にATmega16HVAデバイス用に形態設定されています。ソース コードはこの一括で提供されます。ブートロード成功に必要な命令の一覧は後の項で提供されます。

2. 機能

このブートローダは基本的なブートロードを除き、多くの有用な機能を支援します。これらの機能はブートローダPC応用と共にブートローダ ファームウェアで実装されます。このブートローダを使用することで、使用者は以下の付加機能を利用することができます。

2.1. CRCチェックサム

ブートローダはフラッシュとEEPROMの両方の更新に対してCRCチェックサムを実装します。CRC検査はデバイス書き込み後にフラッシュ/EEPROMの内容を読み出すことによって実行され、従って更新成功を保証します。CRC検査はフラッシュ更新とEEPROM更新の命令に統合されます。

2.2. SLEEP命令

このブートローダは未だブートローダ動作形態中にデバイスを休止に置くことができます。デバイスは単線UART上の何れかの活動で起き上がります。この命令は消費電力が第1に重要な設計で非常に有用です。

2.3. 上書き保護

このブートローダはブートローダ ファームウェアとファームウェアでのブート フラグの上書き保護を提供します。これはブートローダが自身を上書きしないことを保証します。

2.4. 現場更新

このブートローダ構造は現場更新性を支援します。この機能は7.3.項で説明されます。

3. ブートローダ ファームウェア設計

先に言及したように、このブートローダ設計は以下の3つの既存応用記述を伴います。

- Atmel AVR112:ブート領域なしデバイス用TWIブートローダ
- Atmel AVR274:単線ソフトウェアUART
- Atmel AVR911:AVR公開ソース書き込み器

これらの部署を深く理解するには各々の応用記述を参照してください。ブートローダ ファームウェア設計は単線UART用に変更されたAVR112に基づきます。AVR112は更にAtmel「AVR109:自己プログラミング」応用記述に基づき、AVR911で説明されるようにAVROSP互換です。AVR112実装だけでなくAVROSPも単線UARTインターフェースに適応させるために変更されます。目的対象AVRデバイスはブートロード(設定)するどんな主装置の必要もなく、PCホストへ直接的に接続します。この接続は図1.で示されます。

UART_PROG.exeと名付けられて変更したAVROSPの実行可能バイナリだけでなくソースもこの一括で提供されます。

この応用記述は以下の4つの部分から成る完全なブートローダ解決策を検討します。

- ブートローダ使用者インターフェース
- 単線UART
- ブートローダ ファームウェア
- 仮の目的対象応用

3.1. ブートローダ使用者インターフェース

ブートローダ応用はAVR911応用記述で説明されるようにAVR公開ソース書き込み器(AVROSP)に基づきます。AVROSPは単線UARTに対して提供するために変更されます。変更されたAVROSPのために予めコンパイルされた実行可能(ファイル)がUART_PROG.exeとして一括と共に提供されます。この実行可能(ファイル)を内部的に呼んでブートローダ命令を実行する既製のバッチ ファイルも提供されます。

3.2. 単線UART

単線UART実装はAVR274応用記述に基づきますが、ポーリングに基づく設計に変更されました。ブートローダがどのシステム割り込みも使用してはならず、故に主応用のために全てのシステム資源を利用可能にするためにポーリング法が選ばれます。この単線UART実装は4MHzシステム クロックで最大38400のボーレートを支援するように改良されています。シシテム クロックでのどの変更もUARTのタイミングを失敗させます。

3.3. ブートローダ ファームウェア

この設計は単線UART用に変更されるAVR112応用記述からブートローダ ファームウェアを取り入れます。AVR112はブート領域を持たないデバイスに対するTWIブートローダを検討します。対照的に、このブートローダはブートロードするために1つの汎用I/Oピンだけが必要で、故にどの通信インターフェースも必要ありません。AVR112設計はTWIに代わって単線UARTの使用に変更されて簡単化されました。更にこの設計はAVR112で検討されたような主装置としての別のMCUの必要がありません。目的対象デバイスはRS232ケーブルとRS232送受信部を使用してホストPCへ直接的に接続されます。AVROSPが走行するPCがブートロード主装置です。

3.4. 実演目的対象応用

この応用記述は先の項で言及された全ての変更を持つ仮の応用"16HVADummy"と共にやって来ます。この仮の応用はそれら自身の独自目的対象応用を作成する使用者のための参照基準ファームウェア設計として扱うことができます。この応用は早急な試作やこのブートローダ構造の試験にも使用することができます。

cstartup.s90、macro.m90とリンカ ファイル(cfgm16hva.xcl)は既に変更されて仮応用に含まれます。s90とm90のファイルはそれ以上の変更の必要がなく、使用者はどんな混乱もなしに複写/貼り付けして新しいプロジェクトにそれらを含めることができます。けれども、リンカ ファイルは独自応用がそれを使用して構築される時毎に使用者応用に従って変更が必要でしょう。必要とされる変更は7.2.項で説明されます。

仮の目的対象応用はPB0を交互切り替えし、この構造はブートローダが成功裏に応用を実行したかを見るために同じピンに視野を置くことによって検査することができます。

4. 一括内容

応用記述と共に供給されるソフトウェア一括は以下の4つのディレクトリを含みます。

- Single Wire UART Bootloader source code (単線ブートローダ ソース コード)

  これはブートローダ ファームウェアの完全なソースを含みます。

- 16HVADummy (ATmega16HVA仮応用)

  これはPB0を交互切り替えするAtmel ATmega16HVAデバイス用の仮の目的対象応用構築です。これはこのブートローダと共に使用されるために目的対象がどう形態設定されるかを実演するのに使用されます。

- Bootloader UI source code (ブートローダ使用者インターフェース ソース コード)

  これはブートローダ命令をブートローダ ファームウェアへ送るAVROSPに基づくPC UI応用ソース コードを含みます。この応用はC++で書かれ、Windows®の実行可能ファイル(UART_PROG.exe)を生成します。

- Executables (実行可能)

  このフォルダはそれらを使用するスクリプトを伴い、上のディレクトリの全てのコンパイルされたバイナリを含みます。これらのファイルの使用で、使用者は全くコンパイルの必要なしにATmega16HVAデバイスでのこの構造を素早く確認することができます。この構造確認後、コード単位部はどのデバイスへも移植することができます。

5. 必要条件

5.1. ハードウェア必要条件

単線UARTブートローダはハードウェア書き込み器とのどんな接続も不要です。けれども、以下が必要です。

1. シリアル ポート(またはUSB-シリアル ブリッジ ケーブルを持つUSBポート)を持つホストPC
2. RS-232送受信部
3. UARTとして使用する汎用I/Oピンでの(代表的に15kΩの)外部プルアップ抵抗。これはUSB-シリアル変換ケーブルを使用する場合の任意選択です。それらのケーブルは組み込みプルアップを持ちます。

図1.で示されるような接続作成後、ブートローダ ハードウェアはコードをダウンロード(書き込み)するための道具の全てです。

5.2. ソフトウェア必要条件

この応用記述と提携するファームウェアは以下のツールを使用してコンパイルされます。

1. IAR™ Embedded Workbench IDE 6.10版
2. DEV C++ 4.9版

6. 手順

何れかのﾌﾞｰﾄﾛｰﾀﾞを持つ時に、このﾌﾞｰﾄﾛｰﾀﾞはﾌﾞｰﾄﾛｰﾀﾞ自身を書くための書き込み器が必要です。このﾌﾞｰﾄﾛｰﾀﾞはﾌｧｰﾑｳｪｱ用の1つとEEPROM用の1つの2つのHEXﾌｧｲﾙから成ります。これらのﾌｧｲﾙのﾌﾟﾛｸﾞﾗﾐﾝｸﾞ(書き込み)後、SELFPRGENﾋｭｰｽﾞをﾌﾟﾛｸﾞﾗﾑ(0)にしてください。単線UARTｲﾝﾀｰﾌｪｰｽを接続し、AVROSPのｺﾝﾊﾟｲﾙされた出力であるUART_PROG.exeを使用してPCからの命令を受け取る準備を整えてください。

この手順は図6-1.の流れ図の形式でもっと明確に説明することができます。流れ図の段階は次項でより明らかになります。

もっと使用者に友好的なﾌﾞｰﾄﾛｰﾀﾞGUIを作るため、複雑で長いﾌﾞｰﾄﾛｰﾀﾞ命令に対して容易な文章覆い部を提供するｽｸﾘﾌﾟﾄ ﾌｧｲﾙがｿﾌﾄｳｪｱ一括に含まれます。これら全てのﾌｧｲﾙがUART_PROG.exeと同じ場所になければならないことは言うまでもありません。

これらﾌｧｲﾙは表6-1.で記述されます。

表6-1. ｽｸﾘﾌﾟﾄ ﾌｧｲﾙ

ﾌｧｲﾙ	説明
SETCOM.CMD	使用すべきCOMﾎﾟｰﾄを設定
START.CMD	ｺﾏﾝﾄﾞ ﾌﾟﾛﾝﾌﾟﾄを開始
UART_COM.CFG	最後に使用したCOMﾎﾟｰﾄ番号を保存するためにSETCOM.CMDによって使用されます。
UARTBL.CMD	ﾌﾞｰﾄﾛｰﾀﾞ命令ｽｸﾘﾌﾟﾄ ﾌｧｲﾙ。覆い部用ｽｸﾘﾌﾟﾄを含み、共に全てのﾌｧｲﾙを持って来ます。
UARTBL_DUMP	命令に対する出力はﾃﾞﾊﾞｯｸﾞのためにこのﾌｧｲﾙへ運ぶ(ﾊﾟｲﾌﾟする)ことができます。

使用者が既存ﾌﾞｰﾄﾛｰﾀﾞにもっと機能/命令の追加を望む以外、これらのﾌｧｲﾙはどんな変更も全く必要ありません。

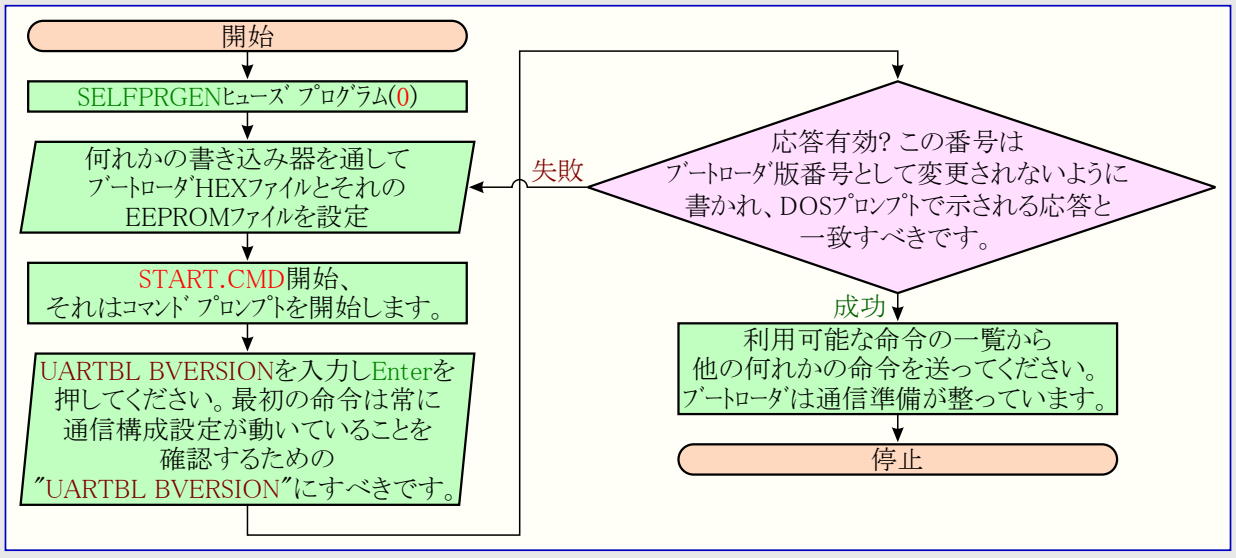
6.1. GUI初期化

初回のﾌﾞｰﾄﾛｰﾀﾞ走行は単線UARTが接続されるCOMﾎﾟｰﾄ番号が必要です。PCのｼﾘｱﾙ ﾎﾟｰﾄを使用する場合、この番号は通常1ですが、USB-ｼﾘｱﾙ変換ｹｰﾌﾞﾙ使用時のCOMﾎﾟｰﾄ番号はﾃﾞﾊﾞｲｽ ﾏﾈｰｼﾞｬを使用して見つけ出さなければなりません。それを行うには"ﾏｲ ｺﾝﾋﾟｭｰﾀ"⇒ﾌﾟﾛﾊﾟﾃｨ⇒ﾊｰﾄﾞｳｪｱ⇒ﾃﾞﾊﾞｲｽ ﾏﾈｰｼﾞｬを右ｸﾘｯｸしてください。ﾃﾞﾊﾞｲｽ ﾏﾈｰｼﾞｬに於いて、COMﾎﾟｰﾄ下でｲﾝｽﾄｰﾙされたUSB-ｼﾘｱﾙ変換器を見つけ出してそれと提携するCOMﾎﾟｰﾄ番号を記録してください。

注: AVROSPは8とそれ以下のCOMﾎﾟｰﾄ番号を支援します。より大きなCOMﾎﾟｰﾄ番号を見つけた場合、それをより小さなCOMﾎﾟｰﾄへ再割り当てしてください。

一旦COMﾎﾟｰﾄ番号が記録されたなら、SETCOM.CMDを開いてそれと同じ番号を入力してください。ｽｸﾘﾌﾟﾄはﾌﾞｰﾄﾛｰﾀﾞ命令を通すためにそのﾎﾟｰﾄを設定します。そこでSTART.CMDをﾀﾞﾌﾞﾙｸﾘｯｸし、命令を送るためのUIの準備を整えます。

図6-1. 手順



6.2. ﾌﾞｰﾄﾛｰﾀﾞ命令

ﾌﾞｰﾄﾛｰﾀﾞは以下の命令を支援します。各命令は同じﾌｫﾙﾀﾞに存在する"START.CMD"と呼ばれるｺﾏﾝﾄﾞ ﾌﾟﾛﾝﾌﾟﾄ実行形式から発行されます。この命令は正しいﾊﾟﾗﾒｰﾀでAVROSP実行可能物を順に呼ぶ覆い部です。このｽｸﾘﾌﾟﾄのより多くの情報はUARTBL_CMDで得られます。

6.2.1. COMSETUP

この命令はCOMﾎﾟｰﾄ番号を構成設定します。

構文: "UARTBL COMSETUP x"

xは単線UARTが接続されるCOMﾎﾟｰﾄ番号です。

6.2.2. BVERSION

この命令はﾌﾞｰﾄﾛｰﾀﾞの版番号を得ます。

構文: "UARTBL BVERSION"

供給されるﾌﾞｰﾄﾛｰﾀﾞ ﾌｧｰﾑｳｪｱで、それは$10です。

6.2.3. FUPDATE

この命令はﾌﾗｯｼｭ ﾒﾓﾘを更新してCRCを確認します。

構文: "UARTBL FUPDATE <ﾌｧｲﾙ名.hex>"

<ﾌｧｲﾙ名.hex>はﾌﾙ ﾊﾟｽを持つ目的対象ﾌﾗｯｼｭHEXﾌｧｲﾙの名前です。

ﾁｪｯｸｻﾑはﾌﾗｯｼｭ ﾒﾓﾘに新しいｲﾒｰｼﾞ書き込みとその内容の読み出し後に確認されます。ﾌﾞｰﾄﾛｰﾀﾞ ﾌｧｰﾑｳｪｱはﾌﾞｰﾄﾛｰﾀﾞ ﾌｧｰﾑｳｪｱの上書き保護のためにﾌﾞｰﾄﾛｰﾀﾞ区分内のどんな書き込みも許しません。

6.2.4. EUPDATE

この命令はEEPROMを更新してCRCを確認します。

構文: "UARTBL EUPDATE <ﾌｧｲﾙ名.hex>"

<ﾌｧｲﾙ名.hex>はﾌﾙ ﾊﾟｽを持つ目的対象EEPROM HEXﾌｧｲﾙの名前です。

ﾁｪｯｸｻﾑはEEPROMに新しいｲﾒｰｼﾞ書き込みとその内容の読み出し後に確認されます。ﾌﾞｰﾄﾛｰﾀﾞ ﾌｧｰﾑｳｪｱはEEPROM位置の上書き保護のためにBOOT_FLAGが格納された場所の位置へのどんな書き込みも許しません。

6.2.5. SLEEP

この命令はﾃﾞﾊﾞｲｽを休止に置きます。

構文: "UARTBL SLEEP"

一旦休止に置かれると、ﾃﾞﾊﾞｲｽは単線UARTに使用されるﾋﾟﾝでの活動がある場合にだけ起き上がります。ﾌｧｰﾑｳｪｱはこの機能のためにUARTﾋﾟﾝでの外部割り込みを使用します。選択したUARTﾋﾟﾝに対して"main.c"ﾌｧｲﾙ内で以下の定義が正しく設定されていることを確実にしてください。

```
/* 外部割り込みﾏｸﾛ。これらはﾃﾞﾊﾞｲｽ依存です。 */
#define INITIALIZE_UART_EXTERNAL_INTERRUPT()     (EICRA |= (1<<ISC01))
//< INT0の下降端が割り込みを生成するように設定
#define ENABLE_EXTERNAL_INTERRUPT()              (EIMSK |= (1<<INT0))
```

6.2.6. EXEC

この命令はﾌﾞｰﾄ ﾌﾗｸﾞを解除してﾃﾞﾊﾞｲｽをﾘｾｯﾄすることによって応用を開始します。

構文: "UARTBL EXEC"

6.3. 段階的手順

使用者は真新しいﾃﾞﾊﾞｲｽを成功裏にﾌﾞｰﾄﾛｰﾄﾞ(設定)するために以下の段階を完了することが必要です。

段階1. – SELFPRGENﾋｭｰｽﾞをﾌﾟﾛｸﾞﾗﾑ(0)してください。

段階2. – 何れかの支援される書き込み器を使用してﾌﾞｰﾄﾛｰﾀﾞ ﾌｧｰﾑｳｪｱとﾌﾞｰﾄﾛｰﾀﾞEEPROMﾌｧｲﾙをﾌﾟﾛｸﾞﾗﾐﾝｸﾞして(書いて)ください。

段階3. – 6.1.項で記述されるように使用者ｲﾝﾀｰﾌｪｰｽを初期化してください。

段階4. – "UARTBL BVERSION"を送って現在のﾌﾞｰﾄﾛｰﾀﾞの版番号が受け取れるかを見てください。

段階5. – ﾌﾗｯｼｭ ﾒﾓﾘを更新するために"UARTBL FUPDATE <ﾌｧｲﾙ名.hex>"命令を送ってください。命令が成功かを知るために調べてください。

段階6. – EEPROMﾌﾟﾛｸﾞﾗﾐﾝｸﾞが必要な場合、EEPROMを更新するために"UARTBL EUPDATE <ﾌｧｲﾙ名.hex>"命令を送ってください。命令が成功かを知るために調べてください。

段階7. – 上の命令の成功裏での完了後、"UARTBL EXEC"実行命令を送ってください。ﾌﾞｰﾄﾛｰﾀﾞはEEPROM内のﾌﾞｰﾄ ﾌﾗｸﾞを解除してｳｫｯﾁﾄﾞｯｸﾞ ﾀｲﾏのﾘｾｯﾄを用いてﾃﾞﾊﾞｲｽをﾘｾｯﾄします。次回のﾃﾞﾊﾞｲｽ始動は目的対象応用で始まります。

6.4. 障害対策

6.4.1. 兆候:BVERSIONへの$FFでの初回応答

予期する返答を受け取るまでBVERSION命令を送ることによって通信構成設定を検査することが推奨されます。時々、ｼｽﾃﾑが深い休止の時にｼｽﾃﾑは最初のBVERSIONに$FFで応答するかもしれません。これはｼｽﾃﾑがﾋﾟﾝでの外部割り込みで丁度起き上がり、最初の命令を取り損なうからです。

6.4.2. 兆候:AVROSPがCOMﾎﾟｰﾄ番号について苦情を言う

AVROSPはCOMﾎﾟｰﾄ1～8を支援し、故にUARTが8よりも大きなCOMﾎﾟｰﾄを割り当てられた場合、AVROSPは誤りを与えます。これはｲﾝﾀｰﾌｪｰｽにより小さなCOMﾎﾟｰﾄ番号を再割り当てすることによって単純に固定化することができます。これはﾃﾞﾊﾞｲｽ ﾏﾈｰｼﾞｬで行うことができます。

6.4.3. 繰り返すCOMﾎﾟｰﾄ時間超過

ｼﾘｱﾙ通信が2回失敗する場合、ﾃﾞﾊﾞｲｽを再始動してｼﾘｱﾙ ｹｰﾌﾞﾙを再接続してください。SETCOM.CMDを走らせて接続を試みるCOMﾎﾟｰﾄ番号を入力してください。これが問題を修正するでしょう。

7. ソフトウェア独自化

この応用記述と共に提供されるｿﾌﾄｳｪｱはAtmel ATmega16HVAﾃﾞﾊﾞｲｽ用に独自化された完全な解決策です。ﾌｧｰﾑｳｪｱ部品の全てはこのﾃﾞﾊﾞｲｽ用に書かれて検査されています。この応用記述は使用者がそれらのどのﾃﾞﾊﾞｲｽの選択に対しても同じ基本構造を設計し得るような例として、この基本構造を使用することを意図します。新しいﾃﾞﾊﾞｲｽに対して、このﾌﾞｰﾄﾛｰﾀﾞ解決策を独自化するには各部品で変更が必要とされます。これらの変更は本章で説明されます。

7.1. ﾌﾞｰﾄﾛｰﾀﾞ ﾌｧｰﾑｳｪｱ

7.1.1. 定義

ﾌﾞｰﾄﾛｰﾀﾞは以下の応用とﾃﾞﾊﾞｲｽ依存定義を含みます。これらの定義は"Commands_Define.h"で注意深く設定されるべきです。

- #define INTVECT_PAGE_ADDRESS 0x00

  これは割り込みﾍﾞｸﾀ表ｱﾄﾞﾚｽの開始位置を定義します。

- #define PAGE_SIZE 128

  これはﾌﾗｯｼｭ ﾒﾓﾘのﾍﾟｰｼﾞの大きさを定義します。

- #define BOOT_PAGE_ADDRESS 0x3A00

  これはﾌﾞｰﾄﾛｰﾀﾞ区分の開始位置を定義します。

- #define TOTAL_NO_OF_PAGES 128

  これは総ﾍﾟｰｼﾞ数を定義します。128ﾊﾞｲﾄのﾍﾟｰｼﾞ容量を持つ16Kﾊﾞｲﾄのﾌﾗｯｼｭ ﾒﾓﾘについては総ﾍﾟｰｼﾞ数は128です。

- #define EEMEM_ADDR_AVERSION 0xFF

  これはEEPROM内のﾌﾞｰﾄ ﾌﾗｸﾞの位置を定義します。この位置はEEPROMの最終位置であるべきです。

- #define BVERSION 0x10

  これはﾌﾞｰﾄﾛｰﾀﾞの版番号を定義します。

- #define CSTARTUP_ADDRESS 0x800

  これはﾌﾞｰﾄﾛｰﾄﾞ成功後にﾌﾞｰﾄﾛｰﾀﾞが飛ぶべきｱﾄﾞﾚｽを定義します。このｱﾄﾞﾚｽは(後で説明される)目的対象応用ﾘﾝｶ ｽｸﾘﾌﾟﾄで定義されるCSTARTUPSEGｱﾄﾞﾚｽと同じです。

- #define BOOT_FLAG 0x2A

  これは設定された場合にｼｽﾃﾑ ﾘｾｯﾄ後にﾌﾞｰﾄﾛｰﾀﾞを開始するﾌﾞｰﾄ ﾌﾗｸﾞの値です。さもなければ目的対象応用が開始されます。

- #define SWUART_PORT_REG PORTC

  これはｿﾌﾄｳｪｱUARTに使用されるﾎﾟｰﾄ出力ﾚｼﾞｽﾀを定義します。

- #define SWUART_PIN_REG PINC

  これはｿﾌﾄｳｪｱUARTに使用されるﾎﾟｰﾄ入力ﾚｼﾞｽﾀを定義します。

- #define SWUART_PIN PC0

  これはｿﾌﾄｳｪｱUARTに使用されるﾋﾟﾝを定義します。

注: EEMEM_ADDR_AVERSIONまたはBOOT_FLAGに行われた何らかの変更があれば、EEMEM_ADDR_AVERSIONによって指定される位置にBOOT_FLAGを書くために、ﾌﾞｰﾄﾛｰﾀﾞEEPROM HEXﾌｧｲﾙがもう一度生成されるべきです。

7.1.2. ﾘﾝｶ ﾌｧｲﾙ変更

提供されるﾌﾞｰﾄﾛｰﾀﾞはAtmel ATmega16HVAに基づき、故にそれは1.2.1.項で説明されたように変更されたこのﾃﾞﾊﾞｲｽの既定ﾘﾝｶ ﾌｧｲﾙに基づきます。このﾌﾞｰﾄﾛｰﾀﾞを別のﾃﾞﾊﾞｲｽへ移転するには選択したﾃﾞﾊﾞｲｽの既定ﾘﾝｶ ﾌｧｲﾙに於いて同様の変更が必要とされます。

7.1.3. システム クロック

提供されるﾌﾞｰﾄﾛｰﾀﾞは4MHzのシステム クロックで走行しなければなりません。目的対象応用がどのシステム クロックでも走行し得るように、ﾌﾞｰﾄﾛｰﾀﾞはそれの初期化ｺｰﾄﾞの一部としてクロック レジスタ設定を変更することによってそのクロックを設定すべきです。例えば、ATmega16HVAに対して、システム クロックは提供されるﾌﾞｰﾄﾛｰﾀﾞ ファームウェアで以下の行のｺｰﾄﾞを追加することによって4MHzに設定されます。

```
CLKPR = (1<<CLKPCE);
CLKPR = 0x01;
```

他のどのﾃﾞﾊﾞｲｽに対しても、このｺｰﾄﾞがシステム クロックを4MHzに設定するための適切なｺｰﾄﾞによって置換されることを確実にしてください。

7.2. 目的対象応用ｿﾌﾄｳｪｱ

ﾌﾞｰﾄﾛｰﾀﾞで使用できるには、応用ﾌｧｰﾑｳｪｱで必要とされるいくつかの形態設定変更があります。標準的にこれらのﾌｧｲﾙはｺﾝﾊﾟｲﾗによってそれらの既定位置から自動的にｲﾝｸﾙｰﾄﾞされますが、変更された版を使用するために、これら3つのﾌｧｲﾙは手動でｲﾝｸﾙｰﾄﾞされなければなりません。ﾌｧｲﾙはこの応用記述と提携するｿﾌﾄｳｪｱ一括で既に提供されます。そのﾌｧｲﾙは以下です。

1. ﾘﾝｶ ﾌｧｲﾙ
2. cstartup.s90ﾌｧｲﾙ
3. macros.m90ﾌｧｲﾙ

これらのﾌｧｲﾙで行われた変更が次に説明されます。

注: 独自ﾘﾝｶ ﾌｧｲﾙを使用するため、ﾌﾟﾛｼﾞｪｸﾄを右ｸﾘｯｸしてOptionsへ行き、Linkerﾀﾌﾞ下で"Override default"ﾁｪｯｸﾎﾞｯｸｽを設定してﾘﾝｶ ﾌｧｲﾙ位置を設定してください。

注: cstartup.s90を追加するため、Add⇒Add filesへ行き、cstartup.s90ﾌｧｲﾙを選択してください。macros.m90ﾌｧｲﾙはcstartup.s90ﾌｧｲﾙ依存でﾌﾟﾛｼﾞｪｸﾄ内に含まれる必要はありません。これは単にcstartup.s90ﾌｧｲﾙと同じﾌｫﾙﾀﾞ内であることが必要です。

7.2.1. ﾌﾞｰﾄﾛｰﾀﾞに使用されるﾌﾗｯｼｭ ﾒﾓﾘを予約するためのﾘﾝｶ ﾌｧｲﾙ設定

応用に於いて、ﾌﾟﾛｸﾞﾗﾑ ﾒﾓﾘの上位側はﾌﾞｰﾄﾛｰﾀﾞ ファームウェア用に予約されることが必要です(図1-1.をご覧ください)。ﾌﾞｰﾄﾛｰﾀﾞ開始ｱﾄﾞﾚｽは固定化され、ﾌﾞｰﾄﾛｰﾀﾞ ファームウェアで変更できないようにｺｰﾄﾞ化されます。利用可能な応用ﾒﾓﾘはﾌﾞｰﾄﾛｰﾀﾞ区分と重複することができず、故に応用ﾘﾝｶ ﾌｧｲﾙで注意深く定義されることが必要です。例えば、与えられたﾌﾞｰﾄﾛｰﾀﾞ ソース ｺｰﾄﾞに於いて、ﾌﾞｰﾄﾛｰﾀﾞは$3A00で始まり、故に応用がｱｸｾｽ可能なﾌﾗｯｼｭ ﾒﾓﾘは、例えそれが16Kﾊﾞｲﾄの(ﾌﾗｯｼｭ ﾒﾓﾘ容量を持つ)ﾃﾞﾊﾞｲｽでも、$39FFで終わらなければなりません。これはﾘﾝｶ ﾌｧｲﾙ内のﾌﾗｯｼｭ終了ｱﾄﾞﾚｽを変更することによって行えます。

例えば、ATmega16HVAﾃﾞﾊﾞｲｽ(16Kﾊﾞｲﾄのﾌﾗｯｼｭ ﾒﾓﾘ容量)について、$3A00～$3FFFのﾌﾞｰﾄﾛｰﾀﾞ区分を予約するためにﾘﾝｶ ﾌｧｲﾙで以下の変更を行うことができます。

```
-D_..X_FLASH_NEND=39FF          /* 近ﾌﾗｯｼｭ ﾒﾓﾘの最後 */
-D_..X_FLASH_END=39FF           /* ﾌﾗｯｼｭ ﾒﾓﾘの最後 */
```

これらの変更はﾘﾝｶに対してﾌﾗｯｼｭ ﾒﾓﾘの最後の1.5Kﾊﾞｲﾄ($3A00～$3FFF)をｱｸｾｽ不可にします。ｺﾝﾊﾟｲﾙされた応用のﾊﾞｲﾅﾘが14.5Kﾊﾞｲﾄよりも大きければ、ﾘﾝｶは誤りを生成します。

7.2.2. ﾌﾞｰﾄﾛｰﾀﾞ ﾌﾗｸﾞに使用されるEEPROM内の1ﾊﾞｲﾄを予約するためのﾘﾝｶ ﾌｧｲﾙ設定

EEPROMﾊﾞｲﾄの1ﾊﾞｲﾄ(最終ﾊﾞｲﾄが望ましい)はEEPROMﾌﾞｰﾄﾛｰﾀﾞ ﾌﾗｸﾞを格納するために予約されることが必要です。

例えば、Atmel ATmega16HVAﾃﾞﾊﾞｲｽに対して、最後のEEPROMﾊﾞｲﾄを予約するためにﾘﾝｶ ﾌｧｲﾙで以下の変更を行うことができます。

```
-D_..X_EEPROM_END=FE            /* EEPROMの最後 */
```

これはﾘﾝｶに対してｱﾄﾞﾚｽ$00FFの最終ﾊﾞｲﾄをｱｸｾｽ不可にします。

7.2.3. 初期化ｺｰﾄﾞ用の新しい区分を作成するためのﾘﾝｶ ﾌｧｲﾙ変更

応用の初期化ｺｰﾄﾞを含む新しい開始区分は応用ﾘﾝｶ ﾌｧｲﾙで作成されることが必要です。

ﾌﾗｯｼｭ ﾒﾓﾘの応用初期化ｺｰﾄﾞのために予め設定されたｱﾄﾞﾚｽを持つために、応用ﾘﾝｶ ﾌｧｲﾙで新しい始動区分が必要とされます。このｱﾄﾞﾚｽはﾌﾞｰﾄﾛｰﾀﾞ ファームウェアで変更できないようにｺｰﾄﾞ化され、ﾌﾞｰﾄﾛｰﾄﾞ成功後にﾌﾞｰﾄﾛｰﾀﾞが飛ぶｱﾄﾞﾚｽです。このｱﾄﾞﾚｽが独立した区分として設定されない場合、初期化ｺｰﾄﾞのｱﾄﾞﾚｽは応用ｺｰﾄﾞの再構築毎で移動されるかもしれません。この区分は標準応用ｺｰﾄﾞ空間の外側で構成設定されなければなりません。

例えば、付随する仮のｺｰﾄﾞに於いて、応用ｺｰﾄﾞは$0000～$0065の長さで、ﾌﾞｰﾄﾛｰﾀﾞは$3A00から始まります。この区分は$0066～$39FF間の何処にでも設定することができます。ﾘﾝｶ ﾌｧｲﾙ内のｺｰﾄﾞの以下の行はｱﾄﾞﾚｽ$0800で新しい区分を構成設定します。これは変更したcstartup.s90ﾌｧｲﾙによって使用されるのと正確に同じ区分名を使用することが重要です。開始ｱﾄﾞﾚｽは使用者によって必要とされるように変更することができます。

```
-Z(CODE)CSTARTUPSEG=800
```

注: 上で言及したように、このｱﾄﾞﾚｽと名称"CSTARTUPSEG"はﾌﾞｰﾄﾛｰﾀﾞと変更したcstartup.s90ﾌｧｲﾙに於いて変更できないようにｺｰﾄﾞ化されるように非常に重要です。この名称は変更されてはなりません。変更した場合、ｱﾄﾞﾚｽは更にﾌﾞｰﾄﾛｰﾀﾞ ファームウェアでも変更が必要です。

7.2.4. cstartup.s90ﾌｧｲﾙ変更

cstartup.s90はCSTARTUPSEGによって定義されるｱﾄﾞﾚｽで初期化ｺｰﾄﾞを置くことを応用に告げるように変更されるｼｽﾃﾑ初期化ﾌｧｲﾙです。cstartup.s90はﾌﾟﾛｼﾞｪｸﾄを右ｸﾘｯｸしてこのﾌｧｲﾙを追加することによって他の何れのﾌｧｲﾙのようにﾌﾟﾛｼﾞｪｸﾄに追加することができます。このﾌｧｲﾙは(提供される)macros.m90ﾌｧｲﾙでの依存性を持ち、ﾌﾟﾛｼﾞｪｸﾄはmacros.m90ﾌｧｲﾙがcstartup.s90ﾌｧｲﾙと共に複写されなければｺﾝﾊﾟｲﾙしません。

7.3. 現場更新可能性のための応用ｿﾌﾄｳｪｱ必要条件

現場項更新可能なﾃﾞﾊﾞｲｽを作るため、応用はﾌﾞｰﾄﾛｰﾀﾞを開始する能力を持たなければなりません。ﾌﾞｰﾄﾛｰﾀﾞはEEPROM郵便箱(boot_flag)に依存して、ｼｽﾃﾑ始動でだけ開始され得るため、EEPROMﾌﾗｸﾞを構成設定するために主応用で以下のｺｰﾄﾞ行が追加され、ｳｫｯﾁﾄﾞｯｸﾞ ﾀｲﾏを用いてﾃﾞﾊﾞｲｽをﾘｾｯﾄすることが必要です。使用者は現場更新が必要とされる時に主応用内からこのｺｰﾄﾞを実行することが必要です。

```
// ﾌｧｰﾑｳｪｱ更新値でEEPORM郵便箱書き込み
Write_EEPROM_byte(EEMEM_ADDR, BOOT_FLAG);
/* ｳｫｯﾁﾄﾞｯｸﾞ ﾘｾｯﾄ動作許可、ｳｫｯﾁﾄﾞｯｸﾞ時間を16msに設定、WDTﾘｾｯﾄ発生まで20ms遅延 */
WDT_ResetTimer();
WDTCSR |= (1<<WDE);
WDT_SetTimeOut( WDTO_16ms );
delay_cycles(20000);    // 1MHzで走行するMCUと仮定
```

EEMEM_ADDRｱﾄﾞﾚｽはﾌﾞｰﾄﾛｰﾀﾞ ﾌｧｰﾑｳｪｱとﾘﾝｶ ﾌｧｲﾙの定義で一貫しているべきﾌﾞｰﾄﾛｰﾀﾞ ﾌﾗｸﾞ ﾊﾞｲﾄのｱﾄﾞﾚｽです。与えられるｿｰｽ ｺｰﾄﾞではこれがEEPROMの最終ｱﾄﾞﾚｽ、換言すると$00FFです。

Write_EEPROM_byte()関数はEEMEM_ADDRによって定義されるｱﾄﾞﾚｽでboot_flagを書きます。提供されるﾌｧｰﾑｳｪｱに於いて、EEMEM_ADDRは$00FFでBOOT_FLAGは$2Aです。Write_EEPROM_byte()の定義はこの資料の範囲外です。

7.4. ﾌﾞｰﾄﾛｰﾀﾞ使用者ｲﾝﾀｰﾌｪｰｽ独自化

提供されるﾌﾞｰﾄﾛｰﾀﾞUI、換言するとAVROSPはAtmel ATmega16HVAﾃﾞﾊﾞｲｽ用に独自化されていますが。どのﾃﾞﾊﾞｲｽへも移転することができます。変更が必要な2つの構成部品があります。最初はAVROSPﾌｧｰﾑｳｪｱで2つ目はUARTBL.CMDです。

7.4.1. AVROSP独自化

AVROSP内の変更は最小で、下で一覧にされます。

1. 無料のDEVC++ｿﾌﾄｳｪｱを使用してﾌﾟﾛｼﾞｪｸﾄを開いてください。
2. AVRDevice.cppﾌｧｲﾙを開き、AVRDeviceｺﾝｽﾄﾗｸﾀで選択したﾃﾞﾊﾞｲｽに対して'flashSize'、'eepromSize'、'pagesize'に正しい値を置いてください。'flashSize'はﾃﾞﾊﾞｲｽ上で利用可能なﾌﾗｯｼｭ ﾒﾓﾘの大きさ、'eepromSize'はﾃﾞﾊﾞｲｽ上で利用可能なEEPROMの大きさ、'pagesize'はﾌﾗｯｼｭ ﾍﾟｰｼﾞの大きさです。
3. ﾌｧｲﾙを保存してﾌﾟﾛｼﾞｪｸﾄをｺﾝﾊﾟｲﾙしてください。生成されたUART_PROG.exeを他のｽｸﾘﾌﾟﾄと同じﾌｫﾙﾀﾞ内に複写してください。

7.4.2. UARTBL.cmd独自化

何れかの文章ｴﾃﾞｨﾀでUARTBL.cmdを開き、"ATmega16HVA"文を新しいﾃﾞﾊﾞｲｽ型式で置き換えてください。ﾌｧｲﾙを保存してください。

Atmel | Enabling Unlimited Possibilities®

Atmel Corporation
1600 Technology Drive
San Jose, CA 95110
USA
TEL (+1)(408) 441-0311
FAX (+1)(408) 487-2600
www.atmel.com

Atmel Asia Limited
Unit 01-5 & 16, 19F
BEA Tower, Millennium City 5
418 Kwun Tong Road
Kwun Tong, Kowloon
HONG KONG
TEL (+852) 2245-6100
FAX (+852) 2722-1369

Atmel Munich GmbH
Business Campus
Parking 4
D-85748 Garching b. Munich
GERMANY
TEL (+49) 89-31970-0
FAX (+49) 89-3194621

Atmel Japan G.K.
141-0032 東京都品川区
大崎1-6-4
新大崎勧業ﾋﾞﾙ 16F
ｱﾄﾒﾙ ｼﾞｬﾊﾟﾝ合同会社
TEL (+81)(3)-6417-0300
FAX (+81)(3)-6417-0370

本応用記述はAtmelのAVR947応用記述(Rev.42034A-10/2012)の翻訳日本語版です。日本語では不自然となる重複する形容表現は省略されている場合があります。日本語では難解となる表現は大幅に意訳されている部分もあります。必要に応じて一部加筆されています。頁割の変更により、原本より頁数が少なくなっています。

必要と思われる部分には( )内に英語表記や略称などを残す形で表記しています。

青字の部分はﾘﾝｸとなっています。一般的に赤字の0,1は論理0,1を表します。その他の赤字は重要な部分を表します。