AN3341

AVR® DA MCU系用基本ブートローダ

序説

著者: Cristian Pop, Iustinian Bujor, Microchip Technology Inc.

この応用記述はAVR® DA MCU系ﾏｲｸﾛ ｺﾝﾄﾛｰﾗ(MCU)が自己ﾌﾟﾛｸﾞﾗﾐﾝｸﾞをどう使うことができるかを記述します。これは外部書き込み器の必要なしにﾌﾗｯｼｭ ﾒﾓﾘに応用ｺｰﾄﾞをﾀﾞｳﾝﾛｰﾄﾞ(書き込み設定)することを使用者に許します。応用例はPython™ｽｸﾘﾌﾟﾄが走行しているPCとUARTｲﾝﾀｰﾌｪｰｽを通して通信するのにAVR128DA48 Curiosity Nano基板を使っています。

無用なﾃﾞｰﾀの転送を避けるため、現在の実装は新しいｲﾒｰｼﾞの機能についてﾌﾞｰﾄﾛｰﾀﾞに通知する構成設定領域をｲﾒｰｼﾞの始めに含みます。この情報に含まれるのはｺｰﾄﾞの大きさで、故に有効なﾃﾞｰﾀだけがﾒﾓﾘに転送され、従ってｱｯﾌﾟﾛｰﾄﾞ(読み出し)時間をかなり減らしています。

提供されるﾌﾞｰﾄﾛｰﾀﾞ例応用とPython™ｽｸﾘﾌﾟﾄは独自のﾌﾞｰﾄﾛｰﾀﾞ応用のための開始点として適当です。下の貯蔵庫の各々はMPLAB XとAtmel Studioの両環境に対してﾌﾞｰﾄﾛｰﾀﾞの例とﾎｽﾄ応用を提供します。

GitHubでMPLAB Xﾌﾟﾛｼﾞｪｸﾄを見てください。
貯蔵庫を閲覧するにはｸﾘｯｸしてください。

GitHubでAtmel Studio解決策を見てください。
貯蔵庫を閲覧するにはｸﾘｯｸしてください。

目次

1. 関連ﾃﾞﾊﾞｲｽ

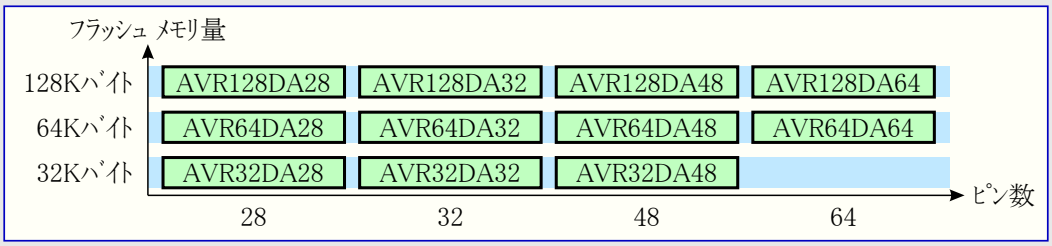
本章はこの文書に関連するﾃﾞﾊﾞｲｽを一覧にします。

1.1. AVR® DA系概要

下図はﾋﾟﾝ数の変種とﾒﾓﾘ量を展開してAVR® DAﾃﾞﾊﾞｲｽを示します。

- これらのﾃﾞﾊﾞｲｽが完全にﾋﾟﾝと機能が互換のため、垂直方向移植はｺｰﾄﾞ変更なしで可能です。
- 左への水平方向移植はﾋﾟﾝ数、従って利用可能な機能を減らします。

図1-1. AVR® DA系概要



異なるﾌﾗｯｼｭ ﾒﾓﾘ量を持つﾃﾞﾊﾞｲｽは一般的にSRAM量も異なります。

2. ﾃﾞﾊﾞｲｽ自己ﾌﾟﾛｸﾞﾗﾐﾝｸﾞ

AVR DAﾃﾞﾊﾞｲｽでのﾌﾗｯｼｭ ﾒﾓﾘとEEPROMに対するｱｸｾｽは以前のmegaAVR®とtinyAVR®のﾃﾞﾊﾞｲｽから変更されてしまっています。これは他のﾃﾞﾊﾞｲｽでﾌﾗｯｼｭ ﾒﾓﾘとEEPROMを書くための既存ｺｰﾄﾞがAVR DAﾃﾞﾊﾞｲｽでの正しい機能に変更されなければならないことを意味します。本章は何が変わったかとｺｰﾄﾞをそれらの変更に適合させる方法を記述します。

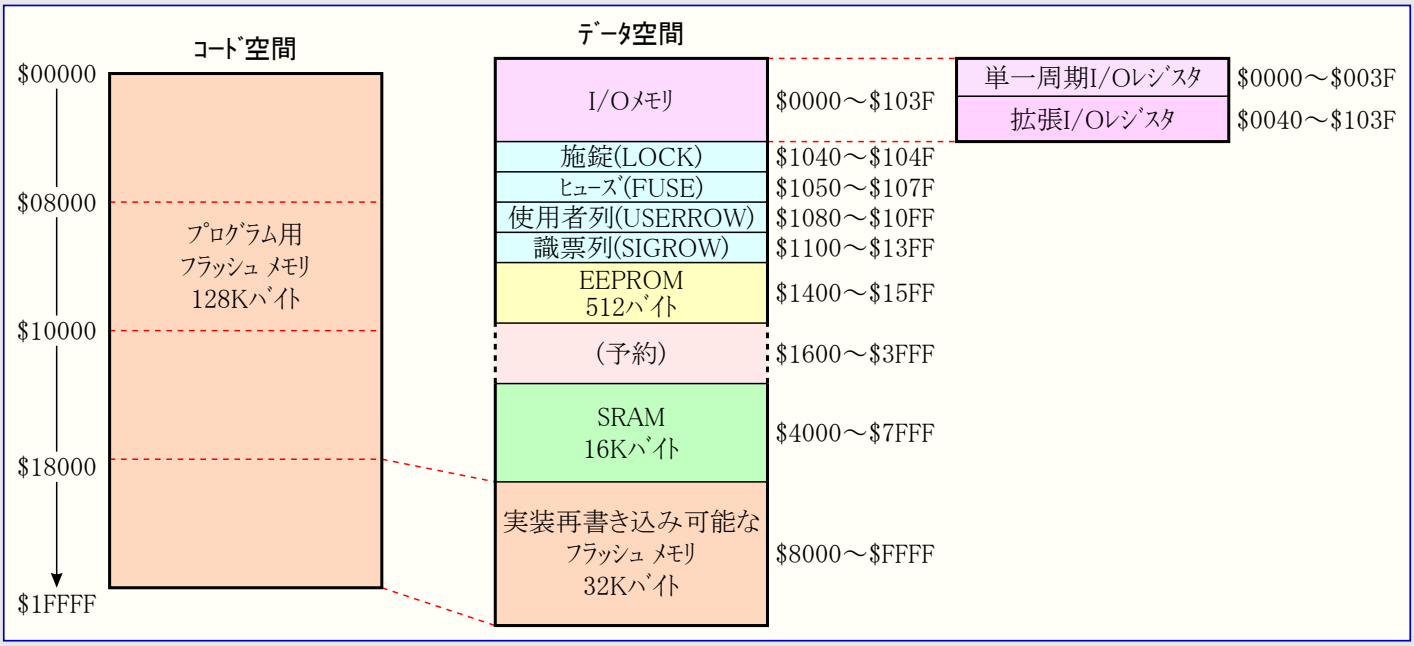
2.1. 変わったこと

AVR DAﾃﾞﾊﾞｲｽでﾌﾗｯｼｭ ﾒﾓﾘはｺｰﾄﾞ空間に割り当てられてLPMとSPMの命令を通して語ｱｸｾｽ可能です。加えて、ﾌﾗｯｼｭ ﾒﾓﾘはCPUﾃﾞｰﾀ空間にも割り当てることができます。これはSRAM、EEPROM、I/Oﾚｼﾞｽﾀと同じｱﾄﾞﾚｽ空間と命令を共用し、ｱｾﾝﾌﾞﾘ言語でLD/ST系命令を使ってｱｸｾｽ可能なことを意味します。

ﾌﾗｯｼｭ ﾒﾓﾘは次のように割り当てられます。

- $0000 (PROGMEM_START)　　：LPM/SPM命令経由でﾌﾟﾛｸﾞﾗﾑ ﾒﾓﾘとしてｱｸｾｽした時
- $8000 (MAPPED_PROGMEM_START)：LD/ST系命令経由でﾃﾞｰﾀ ﾒﾓﾘとしてｱｸｾｽした時

図2-1. AVR DA系用ﾒﾓﾘ配置



ﾃﾞｰﾀ空間内の実装再書き込み可能なﾌﾗｯｼｭ ﾒﾓﾘの大きさは32Kﾊﾞｲﾄです。32Kﾊﾞｲﾄを超えるﾌﾗｯｼｭ ﾒﾓﾘの大きさを持つﾃﾞﾊﾞｲｽについてはﾌﾗｯｼｭ ﾒﾓﾘが32Kﾊﾞｲﾄの塊に分割されます。これらの塊はNVM制御器の制御B(NVMCTRL.CTRLB)ﾚｼﾞｽﾀのﾃﾞｰﾀ空間へのﾌﾗｯｼｭ部分割り当て(FLMAP)ﾋﾞｯﾄ領域を使ってﾃﾞｰﾀ空間に割り当てられます。

図2-2. CTRLBレジスタ

ビット	7	6	5	4	3	2	1	0
	FLMAPLOCK		FLMAP1,0			APPDATAWP	BOOTRP	APPCODEWP
アクセス種別	R/W	R	R/W	R/W	R	R/W	R/W	R/W
リセット値	0	0	1	1	0	0	0	0

● **ビット5,4 – FLMAP1,0 : データ空間へのフラッシュ部分割り当て** (Flash Section Mapped into Data Space)

フラッシュ メモリの(32Kバイトの塊の)どの部分がCPUデータ空間の部分として割り当てられ、LD/ST系命令を通してアクセス可能かを選びます。

このビット領域は構成設定変更保護下ではありません。

値		0 0	0 1	1 0	1 1
名称		SECTION0	SECTION1	SECTION2	SECTION3
割り当てられるフラッシュ メモリの部分 (KB)	32KB版	0～32	0～32	0～32	0～32
	64KB版	0～32	32～64	0～32	32～64
	128KB版	0～32	32～64	64～96	96～128

AVR DAデバイスに比べてtinyAVR®とmegaAVR®のデバイスとの別の主な違いはフラッシュ書き込みアクセスです。以前のtinyAVRとmegaAVRのデバイスではフラッシュ書き込みがページ緩衝部を通して実行される一方で、AVR DAデバイスでは書き込みがST/SPM命令を使ってフラッシュ メモリ位置に直接的に行われます。

注: フラッシュ メモリ位置に書くにはその位置が空白($FF)でなければならず、さもなければ結果は既存の値と新しい値間のAND(論理積)になります。

2.2. ブートローダ コード領域、応用コード領域、応用データ領域

フラッシュ メモリは、ブートローダ コード(BOOT)、応用コード(APPCODE)、応用データ(APPDATA)の3つの領域に分けることができ、各々は可変数の(512バイトの塊の)フラッシュ ページから成ります。

安全性の理由のため、現在コードが実行中のフラッシュ メモリの領域に書くことはできません。APPCODE領域へのコード書き込みはBOOT領域から実行することが必要で、APPDATA領域へのコード書き込みはBOOT領域かAPPCODE領域のどちらかから実行しなければなりません。

ヒューズが各々の領域の大きさを定義します。これを制御するBOOTSIZEとCODESIZEのヒューズがあります。次表はこれらのヒューズがどう領域を構成するかを示します。

図2-3. AVR® DAフラッシュ メモリ割り当て

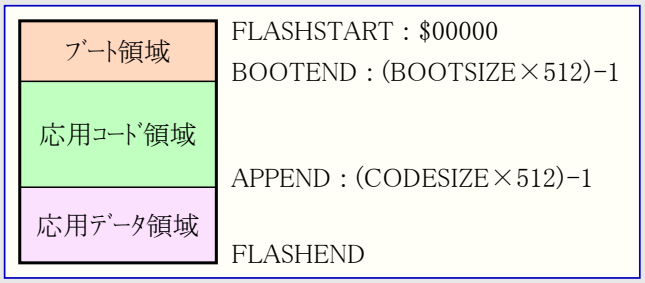


表2-1. フラッシュ領域の構成設定

BOOTSIZE	CODESIZE	BOOT領域	APPCODE領域	APPDATA領域
0	0	0～FLASHEND	-	-
>0	0	0～BOOTEND	BOOTEND～FLASHEND	-
>0	≦BOOTSIZE	0～BOOTEND	-	BOOTEND～FLASHEND
>0	>BOOTSIZE	0～BOOTEND	BOOTEND～APPEND	APPEND～FLASHEND

これらのヒューズがデバイスで意図されるように設定されるのを確実にする良い方法はブートローダ コード プロジェクト内のFUSESマクロを使うことです。これはio.hによってインクルードされるfuse.hで見つけることができます。

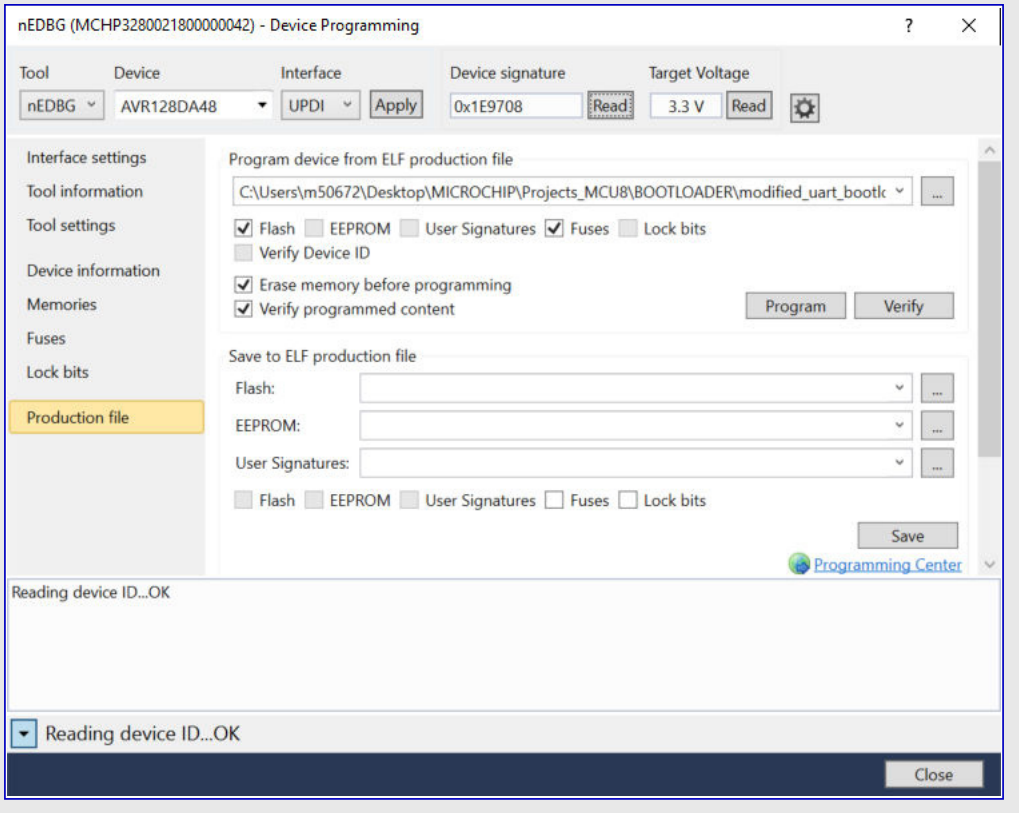
```
#include <avr/io.h>
FUSES = {
.OSCCFG = CLKSEL_OSCHF_gc,                          // 高周波数発振器選択
.SYSCFG0 = CRCSRC_NOCRC_gc | RSTPINCFG_GPIO_gc,     // CRC不許可、RSTピンはGPIO動作
.SYSCFG1 = SUT_64MS_gc,                             // 始動時間64ms
.BOOTSIZE = 0x02,                                   // BOOTの大きさ=0x02×512バイト=1024バイト
.CODESIZE = 0x00                                    // 残り全てのフラッシュ メモリを応用コードとして使用
};
```

注: BOOTSIZEとCODESIZEだけでなく、構造体内の全てのヒューズ バイトが構成設定されなければなりません。これは省略されたヒューズ バイトが$00に設定され、望まない構成設定を起こすかもしれないからです。

プロジェクト内にこのマクロを含めることにより、ヒューズ設定がコンパイルされたファイルに含まれます。フラッシュ メモリと同時にヒューズ設定を書くにはデバイス書き込み中に.hexファイルの代わりに.elfファイルが選ばれなければなりません。

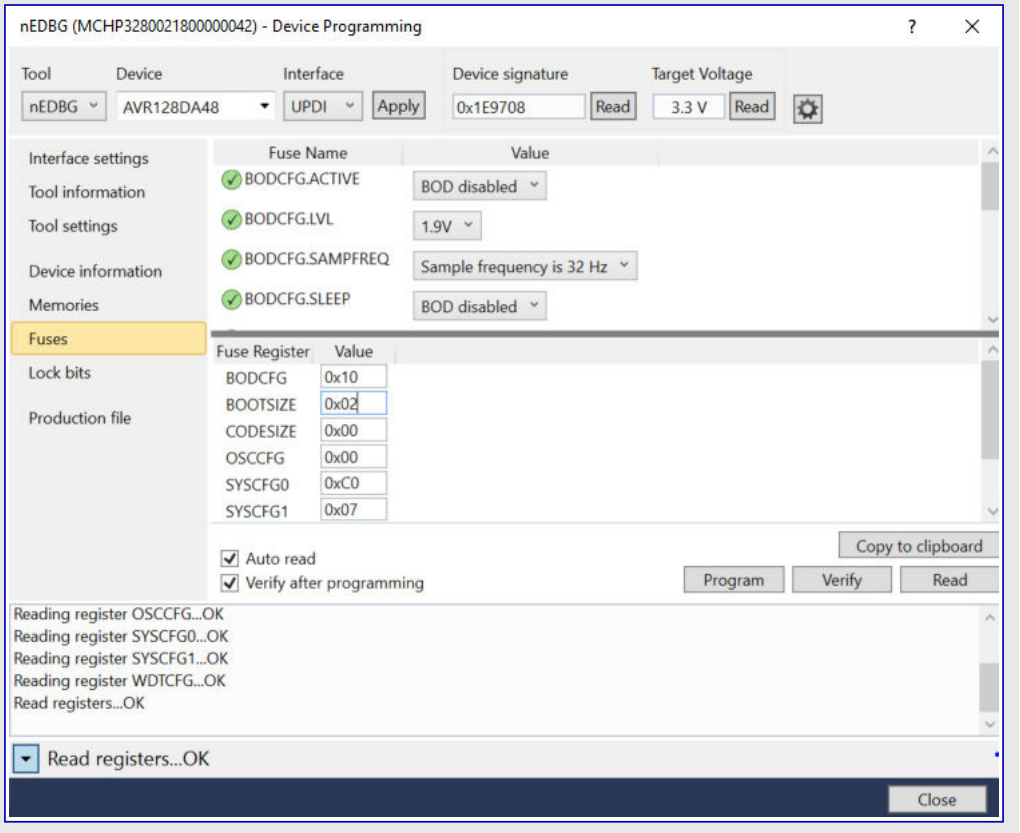
Atmel Studio 7.0での別な方法はDevice Programming(ﾃﾞﾊﾞｲｽ書き込み)からProduction File(製品ﾌｧｲﾙ)を使うことです。

図2-4. Atmel Studio 7.0で製品ﾌｧｲﾙを使うﾋｭｰｽﾞ書き込み



Atmel Studio 7.0に於いて、下図で示されるように、Device Programming(ﾃﾞﾊﾞｲｽ書き込み)(Ctrl+Shift+P)でFuses(ﾋｭｰｽﾞ)を使ってﾋｭｰｽﾞを構成設定することもできます。

図2-5. Atmel Studio 7.0でBOOTSIZEとCODESIZEのﾋｭｰｽﾞを構成設定



2.3. フラッシュ メモリ書き込み

AVR DA系ではフラッシュ メモリ アクセスがNVM制御器を通して扱われます。これはフラッシュ メモリ領域の(32Kバイトの塊の)各部がNVM制御器の制御B(NVMCTRL.CTRLB)レジスタを使うことによってデータ空間に割り当てられ、消去と書き込みの操作が制御A(NVMCTRL.CTRLA)レジスタを使うことによって処理されることを意味します。

現実装のブートローダはフラッシュ メモリ全体をアクセスするのにプログラム空間を使います。アドレス指定はLPM/SPM命令を使って行われます。

フラッシュ メモリはプログラム メモリ空間に割り当てられる時にページ単位で消去されて語の粒度で書かれます。語を書くにはそれぞれの語アドレスを含むページが前もって消去されなければならず、さもなければ結果は既存の値と新しい値間のAND(論理積)になります。

現実装のブートローダは書き込みアドレスが順次増加と仮定し、故に書き込みアドレスが新しいページに入る時にNVMCTRL.CTRLAレジスタ経由でフラッシュ ページ消去(FLPER)指令が送られます。消去操作を実際に開始するため、選んだページに対する偽装書き込みの実行が必要です。

```
if((addr_flash % MAPPED_PROGMEM_PAGE_SIZE) == 0x0000)       // 新しいページ
{
    /* 直前の指令完了待ち */
    while (NVMCTRL.STATUS & NVMCTRL_FBUSY_bm)
    {
        ;
    }
    /* 現在の指令を解消 */
    _PROTECTED_WRITE_SPM(NVMCTRL.CTRLA, NVMCTRL_CMD_NONE_gc);
    /* フラッシュ ページ消去 */
    _PROTECTED_WRITE_SPM(NVMCTRL.CTRLA, NVMCTRL_CMD_FLPER_gc);
    /* 消去操作を開始するため偽装書き込み */
    pgm_word_write(flash_addr, 0x00);
}
```

注: 1つの指令から別の指令への変更は常に指令なし(NOCMD)または操作なし(NOOP)を通して行わなければなりません。さもなければ、NVM制御器の状態(NVMCTRL.STATUS)レジスタの異常(ERROR)ビット領域で指令衝突異常が設定され、現在の指令は実行されません。

書き込み操作は制御A(NVMCTRL.CTRLA)レジスタへのフラッシュ書き込み許可(FLWR)指令書き込みによって開始されます。この指令後、NVMCTRL.CTRLAレジスタに他の指令が書かれない限り、選んだページ位置への複数バイト書き込みが許されます。

```
/* フラッシュ書き込み動作許可 */
_PROTECTED_WRITE_SPM(NVMCTRL.CTRLA, NVMCTRL_CMD_FLWR_gc);

/* 望む値でフラッシュ語書き込み */
pgm_word_write((flash_addr & 0xFFFFFE), data_word);
```

3. ブートローダ応用書き込み

ブートローダは一般的に使用者応用コードの再書き込みを許す短い一纏まりのコードです。新しい応用コードはチップ上の通信インターフェース(UART、I²C、SPI)を使って、または代替のダウンロード チャネル(無線インターフェース、ネットワーク インターフェース、外部メモリ)を使って転送することができます。

現在の例はブートローダ応用をコンパイルするためのAVR GNUコンパイラ集合(GCC)を持つAtmel Studio 7.0またはMPLAB® X 5.30版に対して利用可能です。生成したコードを短くするため、AVR GCC特有コンパイラ/リンカ疑似命令が使われます。

3.1. ブートローダ応用構成設定

AVR GCCによって使われる標準開始ファイルは割り込みベクタ表、AVR CPUとメモリの初期化、main()への無条件分岐を含みます。現実装のブートローダは割り込みを使わず、故に開始ファイルは可能な限りコードを小さく保つために取り去られます。

この場合、main()は呼ばれず、故に関数はデバイスに対する入口の点として開始実行特性に定義されることが必要です。

次のコード断片はAVR GCCコード プロジェクトのコンストラクタ部(.ctors)に置かれ、必要とされる全属性を持つboot()関数の例を示します。

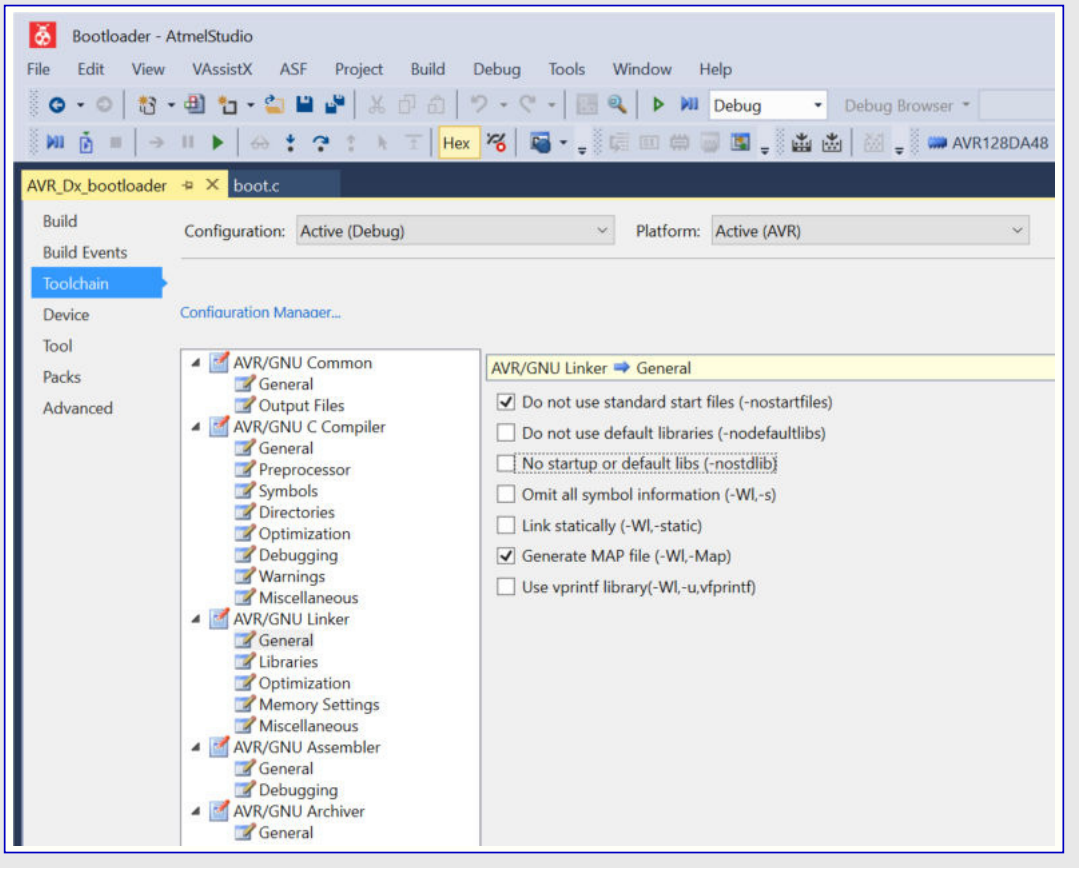
```
__attribute__((naked)) __attribute__((section(".ctors"))) void boot(void){
    /* C支援用システム初期化 */
    asm volatile("clr r1");
    /* ブートローダ コードで置換 */
    while (1)
    {
        ;
    }
}
```

関数はCALL/RET命令を使って呼ばれず、始動で移行されるため、ｺﾝﾊﾟｲﾗはﾌﾟﾛﾛｰｸﾞとｴﾋﾟﾛｰｸﾞの機能を省略するためにnaked属性によって指示されます。より多くの詳細についてはAVR GCC文書をご覧ください。

AVR GCCでの標準開始ﾌｧｲﾙはﾌﾟﾛｼﾞｪｸﾄ ｺﾝﾊﾟｲﾙ時に-nostartfilesﾘﾝｶ ﾌﾗｸﾞを設定することによって禁止されます。

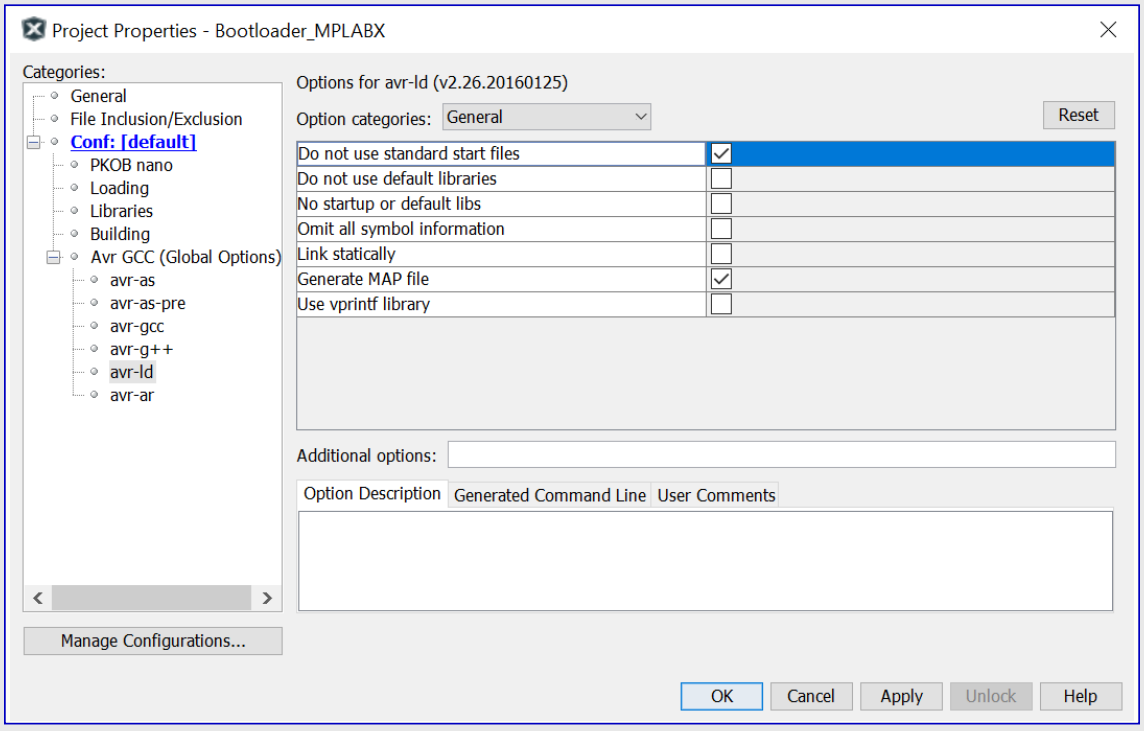
Atmel Studio 7.0では、下図で示されるように、これはProject Properties (Alt+F7)(ﾌﾟﾛｼﾞｪｸﾄ特性(Alt+F7))⇒Toolchain(ﾂｰﾙﾁｪｰﾝ)⇒AVR/GNU Linker(AVR/GNUﾘﾝｶ)⇒General(全般)で見つけることができます。

図3-1. Atmel Studio 7.0での標準ﾌｧｲﾙ禁止



MPLAB Xでは、下図で示されるように、これはFile(ﾌｧｲﾙ)⇒Project Properties(ﾌﾟﾛｼﾞｪｸﾄ特性)⇒Conf(構成設定)⇒Avr GCC(Global Options)(AVR GCC(全体任意選択))⇒avr-ID(AVR-ID)⇒General(全般)で見つけることができます。

図3-2. MPLABでの標準ﾌｧｲﾙ禁止



注: • ﾌﾞｰﾄﾛｰﾀﾞ ﾌﾟﾛｼﾞｪｸﾄはﾋｭｰｽﾞ設定を含めることが必要です。より多くの詳細については「2.2. ﾌﾞｰﾄﾛｰﾀﾞ ｺｰﾄﾞ領域、応用ｺｰﾄﾞ領域、応用ﾃﾞｰﾀ領域」を参照してください。

• 生成されたｺｰﾄﾞはﾋｭｰｽﾞ設定からの結果としてのBOOT領域を超えてはなりません。

3.2. ﾌﾞｰﾄﾛｰﾀﾞとで使うための応用構成設定

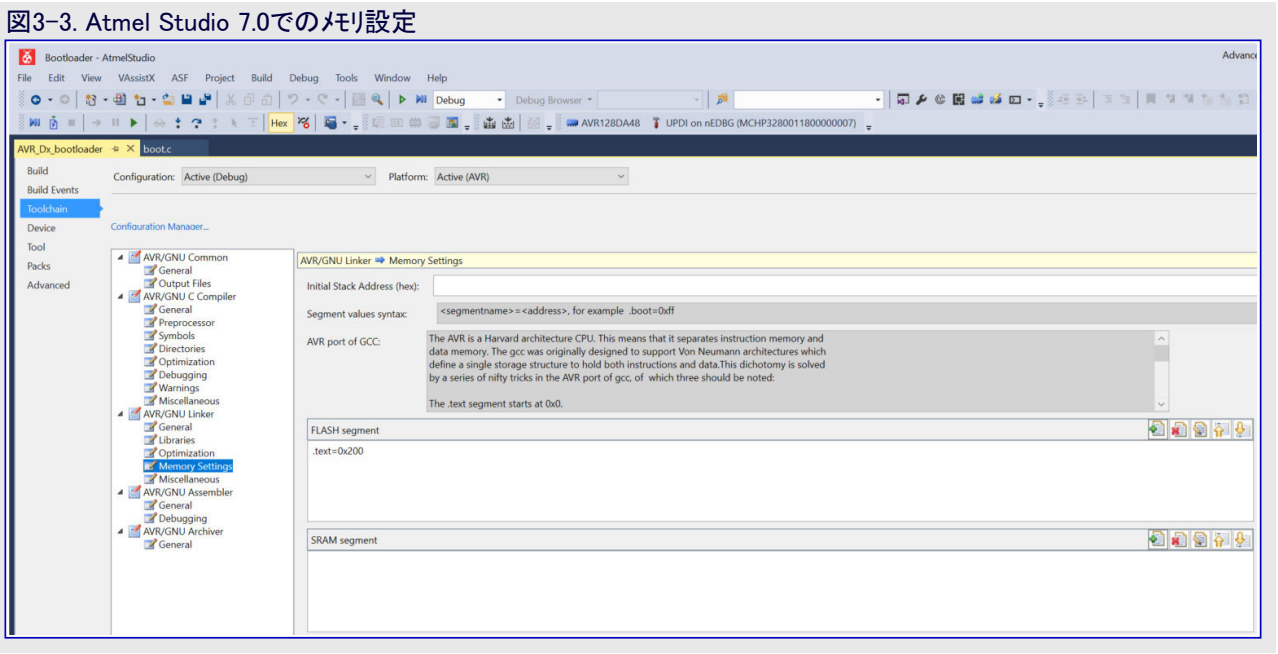
ﾌﾞｰﾄﾛｰﾀﾞが使われると、使用者応用はBOOT領域の後ろに置かれなければなりません。例としてBOOTSIZEﾋｭｰｽﾞ設定$02を使うと、BOOT領域は1024(2×512)ﾊﾞｲﾄの大きさを持ちます。ｺｰﾄﾞ領域が語整列(語で構成)されるため、応用ｺｰﾄﾞの開始ｱﾄﾞﾚｽは$200です。

ｺﾝﾊﾟｲﾙされた応用ｺｰﾄﾞを置くﾌﾗｯｼｭ ﾒﾓﾘ内の場所を知るためのAVR GCCﾘﾝｶ ｽｸﾘﾌﾟﾄについて、.textｺｰﾄﾞ部は応用ｺｰﾄﾞ領域の位置に対応するように構成設定されなければなりません。この再配置は以下のﾘﾝｶ任意選択を利用することによって行われます。

```
-Wl,--section-start=.text=0x200
```

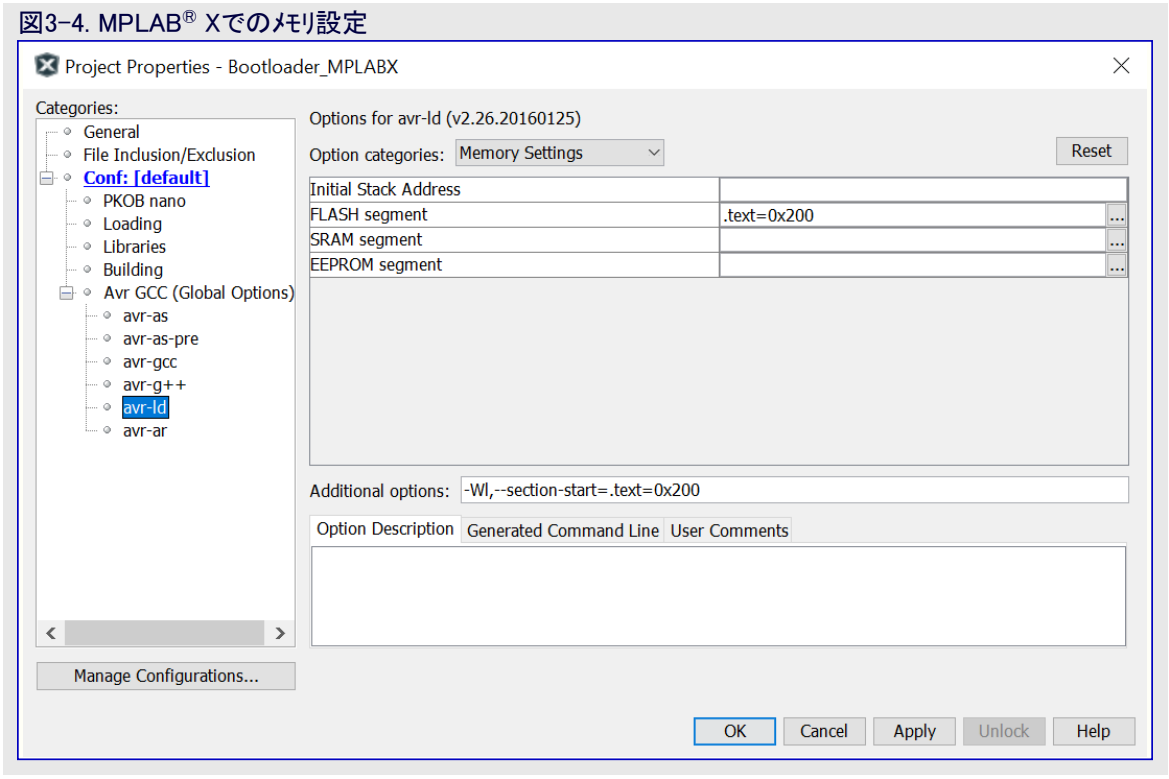
Atmel Studio 7.0では、下図で示されるように、Project Properties (Alt+F7)(ﾌﾟﾛｼﾞｪｸﾄ特性(Alt+F7))⇒Toolchain(ツールチェーン)⇒AVR/GNU Linker(AVR/GNUﾘﾝｶ)⇒Memoy Settings(ﾒﾓﾘ設定)でFLASH区分に.text=0x200を追加することによって再配置を行うことができます。

図3-3. Atmel Studio 7.0でのﾒﾓﾘ設定



MPLAB Xでは、下図で示されるように、File(ﾌｧｲﾙ)⇒Project Properties(ﾌﾟﾛｼﾞｪｸﾄ特性)⇒Conf(構成設定)⇒Avr GCC(AVR GCC)⇒avr-ID(AVR-ID)⇒Memory Settings(ﾒﾓﾘ設定)でFLASH区分に.text=0x200を追加することによって再配置を行うことができます。

図3-4. MPLAB® Xでのﾒﾓﾘ設定



3.3. メモリ保護

行われつつあるアクセスや書き込みからいくつかの領域やフラッシュ メモリの全てを保護するため、利用可能ないくつかの保護の手順があります。禁止することができない保護は「2.2. ブートローダ コード領域、応用コード領域、応用データ領域」で記述されたフラッシュ領域書き込み特権だけです。加えて、安全性を改善するために以下の保護形式を構成設定することができます。

3.3.1. BOOTRPとAPPCODEWP

ブート領域読み込み保護(BOOTRP)と応用コード領域書き込み保護(APPCODEWP)はNVM制御器の制御B(NVMCTRL.CTRLB)レジスタに置かれ、走行時書き込み保護に使われます。

BOOTRPはBOOT領域からの読み込みアクセスとコード実行を防ぎます。このビットはBOOT領域からだけ書くことができ、リセットによってのみ解除(0)することができます。BOOTRPが設定(1)されると、どのBOOT領域読み込みの試みも'0'を返し、どのBOOT領域命令取得もNOP命令を返します。この読み込み保護はビットが書かれた後にBOOT領域を出た時にだけ有効です。

APPCODEWPは応用コード領域への書き込みアクセスを制御します。設定(1)時、この領域を更新するどの試みも、例えこれがBOOT領域から実行されていても、無視されます。このビットはリセットによってのみ解除(0)されます。

3.3.2. 施錠ビット

施錠ビットはヒューズ、フラッシュ メモリ(ブートローダ コード領域、応用コード領域、応用データ領域の全て)、SRAM、EEPROMのアクセスから書き込み器を防ぐことができる独立したヒューズに配置されます。

デバイスを解錠する唯一の方法はチップ消去(CHIPERASE)だけです。(チップ消去時に)応用データは保持されません。

3.4. ブートローダ動作

ブートローダ応用の開始で、MCUがブートローダ動作へ移行するか、または使用者の応用を開始するかを決めるのに物理的なピンの状態を使うことができます。このブート ピンがHighの場合、BOOTRPが許可(1)され実行はAPPCODEへ飛び、さもなければブートローダを開始してUARTからデータを受け取るのを待ちます。

ブートローダ動作移行時、基板上のLEDがONに切り替わり、ページ境界に達する時に(ON/OFFが)切り替わります。

開始後、ブートローダは通信インターフェース経由で受け取る付加標識("INFO")と後続する124バイトのデータを待ちます。この128バイトの緩衝部は転送される応用イメージについての情報を含みます。この情報緩衝部に使われる構造体は次のとおりです。

```
typedef struct
{
    uint32_t start_mark;
    uint32_t start_address;
    uint32_t memory_size;
    uint8_t  reserved[116];
}application_code_info;
```

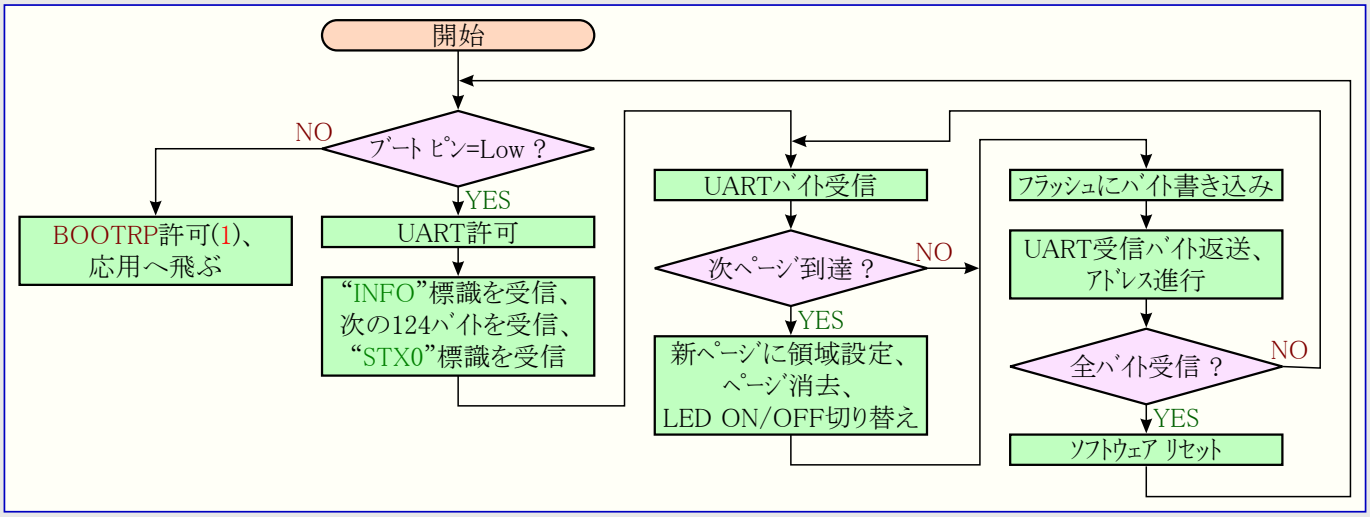
start_markは"INFO"標識を含み、start_addressは使用者応用開始アドレスを含み、memory_sizeはイメージの大きさ(バイト数)を含みます。次の116バイトは将来の実装用に予約されます。

イメージ情報受信後、情報緩衝部の最後で付加標識("STX0")が期待され、その後にブートローダはAPPCODE領域へのデータ書き込みを開始します。

ブートローダは応用コードがフラッシュ メモリ アドレスの増加順でバイト毎に受信されることを期待します。新しいページ境界に達すると、最初にこれが消去され次の書き込みに対して準備されます。イメージ情報緩衝部で示されるバイト数に達すると、ブートローダはソフトウェア リセットを実行して新しい応用を開始します。

次の画像はブートローダ動作の流れ構成図を示します。

図3-5. ブートローダ流れ構成図



4. ホスト応用

ブートローダ環境に於いて、ホストはデバイスに応用コードを送る責任があるシステムです。これは通常、応用コード更新を実行する目的対象に接続することができるコンピュータやCPU、または同じ回路基板上のホストCPUです。

使用者が目的対象デバイスと通信することができる場合、ホスト応用を作る方法には殆ど制限がありません。最も簡単なホスト応用は基本的な命令行インターフェースだけを持つ一方で、より複雑な応用は安全性と高度な構成設定のいくつかの層を持つ画像使用者インターフェース(GUI)を持ちます。

装置が無線接続を持つ場合、無線通信(OTA:Over-The-Air)書き込みも可能です。これはスマートフォン応用からソフトウェア更新機能や、各装置をコンピュータや書き込み器に物理的に接続することなく多くの装置を更新する他の方法を追加することをより易しくします。

4.1. 応用コード形式

マイクロ コントローラのメモリでアップロードされるイメージの大きさを減らすため、アップロードされるファイルにイメージ情報が追加されます。この情報は新しい応用イメージの開始アドレスと大きさを含みます。

マイクロ コントローラでスクリプトによってアップロードされる.binファイルは以下の形式を持ちます。

“INFO” + 開始アドレス + 応用の大きさ + 予約バイト + “STX0” + 応用コード + $FF(ページの最後まで)

標識の“INFO”と“STX0”はそれらの後に各々、情報領域、応用コードが続くことをブートローダに知らせるためにスクリプトによって挿入されます。

情報領域は128バイトの大きさを持ちます。

メモリに書かれる必要がある応用の大きさについてブートローダが知らされるため、アップロード処理が効率化されます。

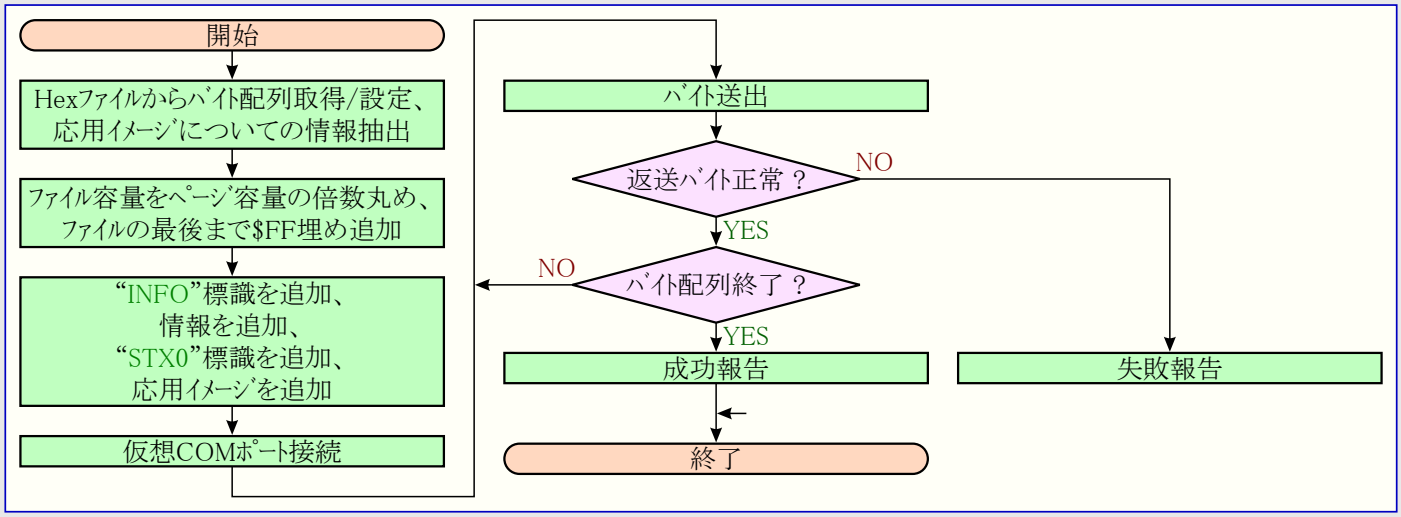
4.2. Python™スクリプト動作

提供されるPythonスクリプトは応用イメージについての応用情報(開始アドレスと大きさ)を含む.hexファイルを.binファイルに変換します。情報領域を書く前に、スクリプトは追加する領域を$FFで満たすことにより、ファイルの大きさをページの大きさの倍数に丸めます。その後、情報領域は「4.1. 応用コード形式」で記述される構造に従って仕上げられて.binファイルの始めに挿入されます。

Pythonスクリプト実行は通信インターフェースを使って生成した.binファイルをアップロードすることによって続けます。この例では、Curiosity Nano基板の組み込みデバッガがマイクロ コントローラとPC間の橋渡しとして使われます。各バイト送信に対し、データ転送が成功したことを知らせるために返答で同じ値が期待されます。

Pythonスクリプトは次の流れ図を持ちます。

図4-1. Python™スクリプト流れ図

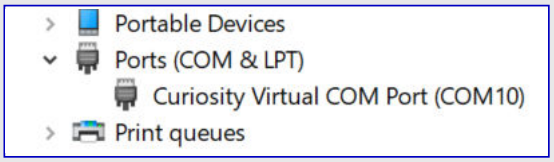


スクリプトを走らせるには以下の引数が必要とされます。

1. アップロードされるべきHexファイル。ファイルが同じフォルダでない場合はパスを含めてください。
2. マイクロ コントローラの最大フラッシュ メモリ容量
3. UART通信に使われる仮想COMポート
   - これはWindows® PCのデバイス マネージャで一覧にされます。AVR128DA48 Curiosity NanoについてはCuriosityVirtual COM Port (COMxxx)として一覧されます。現在の例についてはCOM10ポートが使われています。

   注: 仮想COMポートは基板上のAVR128DA48デバイスのUSARTピン(PC0-TXDとPC1-RXD)に接続されます。

図4-2. Curiosity仮想COMポート



4. 使うボーレート。このブートローダ例によって使われる既定ボーレートは9600です。

COM10ポートに接続されて128Kバイトのフラッシュ メモリ容量を持つAVR Curiosity Nanoに対してスクリプトを走らせるには以下の形式が使われなければなりません。

```
python AVR-DA_uploader.py AVR_Dx_app_example.hex 0x20000 COM10 9600
```

注: スクリプトを開始する前にデバイスをブートローダ動作に置くことを確実にしてください。これはAVR128DA48 Curiosity Nano基板の通電またはリセット中にSW0を押すことによって行われます。

5. 機能拡張

このブートローダ例は基本機能だけを含むブートローダの簡単な実装です。けれども、この実装はいくつかの方法で拡張することができます。本章は可能な拡張のいくつかを紹介します。

5.1. ブートローダ動作移行

物理ピンの状態だけがデバイスをブートローダ動作へ移行する方法ではありません。度々、応用に対してブートローダ更新を起動することが必要です。使用者列(USERROW)やEEPROMに特定の値が書かれた時に更新を起動することが可能です。特定の値が書かれた後、ソフトウェア リセットが発行され、システムはブートローダ動作に移ります。

5.2. 通信インターフェース

ホスト通信に利用可能なインターフェースは最終応用間で違うかもしれません。このブートローダ例は応用コードを受信するのにUARTの基本構成設定を利用している一方で、boot.cでインターフェース原型宣言として使う関数を置換することが必要とされるようにこれは容易に更新することができます。

```
/* インターフェース関数原型宣言 */
#define BOOTLOADER_isRequested()     BUTTON_read()
#define INTERFACE_init()             USART_init()
#define INTERFACE_readByte()         USART_read()
#define INTERFACE_writeByte(a)       USART_write(a)
```

全てのAVR DA系は直列通信に利用可能はハードウェアUSART、TWI、SPIを持ち、入出力ピンを独自のデジタル規約に使うこともできます。

5.3. 割り込み

現実装のﾌﾞｰﾄﾛｰﾀﾞは割り込みを使っていません。ﾌﾞｰﾄﾛｰﾀﾞがもっと複雑な場合、割り込みの使用を必要とし得ます。

ﾘｾｯﾄ後、既定のﾍﾞｸﾀ表の位置はAPPCODE領域の最初の部分です。割り込みﾍﾞｸﾀ表をBOOT領域の最初の部分に再配置することによってﾌﾞｰﾄﾛｰﾀﾞ ｺｰﾄﾞで周辺機能割り込みを使うことができます。それはCPU割り込み制御器の制御A(CPUINT.CTRLA)ﾚｼﾞｽﾀの割り込みﾍﾞｸﾀ選択(IVSEL)ﾋﾞｯﾄを設定(1)することによって行われます。

```
void relocating_vector_table_example(void)
{
    // 割り込みﾍﾞｸﾀ選択ﾋﾞｯﾄを設定(1)、
    // 他の構成設定ﾋﾞｯﾄを守るために読み-変更-書き
    _PROTECTED_WRITE(CPUINT.CTRLA, (CPUINT.CTRLA | CPUINT_IVSEL_bm));
    // 全体割り込み許可
    sei();
}
```

注: ・ﾌﾞｰﾄﾛｰﾀﾞで割り込みが使われる場合、-nostartfilesﾘﾝｶ疑似命令が使われてはなりません。

・割り込みﾍﾞｸﾀ位置がBOOT領域の最初の部分に変更された場合、主応用に移る前にAPPCODE領域の最初の部分に戻すことが必要です。

5.4. ﾃﾞｰﾀ整合性

ﾃﾞﾊﾞｲｽに転送されたｺｰﾄﾞが正しく受信されたことを確実にするため、やって来るﾃﾞｰﾀで巡回冗長検査(CRC:Cyclic Redundancy Check)を使うことができます。これはﾃﾞｰﾀ受信中やｺｰﾄﾞを実行する前に行うことができます。

全てのAVR-DAﾃﾞﾊﾞｲｽはﾌﾗｯｼｭ ﾒﾓﾘ内容を確認するのに使うことができる巡回冗長検査ﾒﾓﾘ走査(CRCSCAN)周辺機能を持ちます。

5.4.1. 機密性

製品とその応用ｺｰﾄﾞが複製、偽造、改ざんされないことを保証するために暗号対応策が必要かもしれません。ﾌﾞｰﾄﾛｰﾀﾞでのCryptoAuthentication™実装はﾎｽﾄとﾃﾞﾊﾞｲｽ間でだけ正当なｺｰﾄﾞを転送することができることを保証します。

より多くの情報についてはMicrochipのCryptoAuthenticationｻｲﾄを訪ねてください。

6. 参考資料

1. AVR128DA28/32/48/64暫定ﾃﾞｰﾀｼｰﾄ
2. AVR128DA48 Curiosity Nano使用者の手引き

7. 改訂履歴

文書改訂	日付	注釈
A	2020年2月	初版文書公開
B	2020年3月	貯蔵庫ﾘﾝｸ更新 最新の商標により、AVR-DAをAVR® MCU DA (AVR-DA)に更新
C	2020年5月	最新の商標により、AVR® MCU DA (AVR-DA)をAVR® DA MCUに、AVR-DAをAVR DAに更新

Microchipｳｪﾌﾞ ｻｲﾄ

Microchipはhttp://www.microchip.com/で当社のｳｪﾌﾞ ｻｲﾄ経由でのｵﾝﾗｲﾝ支援を提供します。このｳｪﾌﾞ ｻｲﾄはお客様がﾌｧｲﾙや情報を容易に利用可能にするのに使われます。利用可能な情報のいくつかは以下を含みます。

- **製品支援** – ﾃﾞｰﾀｼｰﾄと障害情報、応用記述と試供ﾌﾟﾛｸﾞﾗﾑ、設計資源、使用者の手引きとﾊｰﾄﾞｳｪｱ支援資料、最新ｿﾌﾄｳｪｱ配布と保管されたｿﾌﾄｳｪｱ
- **全般的な技術支援** – 良くある質問(FAQ)、技術支援要求、ｵﾝﾗｲﾝ検討ｸﾞﾙｰﾌﾟ、Microchip設計協力課程会員一覧
- **Microshipの事業** – 製品選択器と注文の手引き、最新Microchip報道発表、ｾﾐﾅｰとｲﾍﾞﾝﾄの一覧、Microchip営業所の一覧、代理店と代表する工場

製品変更通知ｻｰﾋﾞｽ

Microchipの製品変更通知ｻｰﾋﾞｽはMicrochip製品を最新に保つのに役立ちます。加入者は指定した製品系統や興味のある開発ﾂｰﾙに関連する変更、更新、改訂、障害情報がある場合に必ず電子ﾒｰﾙ通知を受け取ります。

登録するにはhttp://www.microchip.com/pcnへ行って登録指示に従ってください。

お客様支援

Microchip製品の使用者は以下のいくつかのﾁｬﾈﾙを通して支援を受け取ることができます。

- 代理店または販売会社
- 最寄りの営業所
- 組み込み解決技術者(ESE:Embedded Solutions Engineer)
- 技術支援

お客様は支援に関してこれらの代理店、販売会社、またはESEに連絡を取るべきです。最寄りの営業所もお客様の手助けに利用できます。営業所と位置の一覧はこの資料の後ろに含まれます。

技術支援はhttp://www.microchip.com/supportでのｳｪﾌﾞ ｻｲﾄを通して利用できます。

Microchipﾃﾞﾊﾞｲｽ ｺｰﾄﾞ保護機能

Microchipﾃﾞﾊﾞｲｽでの以下のｺｰﾄﾞ保護機能の詳細に注意してください。

- Microchip製品はそれら特定のMicrochipﾃﾞｰﾀｼｰﾄに含まれる仕様に合致します。
- Microchipは意図した方法と通常条件下で使われる時に、その製品系統が今日の市場でその種類の最も安全な系統の1つであると考えます。
- ｺｰﾄﾞ保護機能を破るのに使われる不正でおそらく違法な方法があります。当社の知る限りこれらの方法の全てはMicrochipのﾃﾞｰﾀｼｰﾄに含まれた動作仕様外の方法でMicrochip製品を使うことが必要です。おそらく、それを行う人は知的財産の窃盗に関与しています。
- Microchipはそれらのｺｰﾄﾞの完全性について心配されているお客様と共に働きたいと思います。
- Microchipや他のどの半導体製造業者もそれらのｺｰﾄﾞの安全を保証することはできません。ｺｰﾄﾞ保護は当社が製品を”破ることができない”として保証すると言うことを意味しません。

ｺｰﾄﾞ保護は常に進化しています。Microchipは当社製品のｺｰﾄﾞ保護機能を継続的に改善することを約束します。Microchipのｺｰﾄﾞ保護機能を破る試みはﾃﾞｼﾞﾀﾙ ﾐﾚﾆｱﾑ著作権法に違反するかもしれません。そのような行為があなたのｿﾌﾄｳｪｱや他の著作物に不正なｱｸｾｽを許す場合、その法律下の救済のために訴権を持つかもしれません。

法的通知

ﾃﾞﾊﾞｲｽ応用などに関してこの刊行物に含まれる情報は皆さまの便宜のためにだけ提供され、更新によって取り換えられるかもしれません。皆さまの応用が皆さまの仕様に合致するのを保証するのは皆さまの責任です。**Microchipはその条件、品質、性能、商品性、目的適合性を含め、明示的にも黙示的にもその情報に関連して書面または表記された書面または黙示の如何なる表明や保証もしません。**Microchipはこの情報とそれの使用から生じる全責任を否認します。生命維持や安全応用でのMicrochipﾃﾞﾊﾞｲｽの使用は完全に購入者の危険性で、購入者はそのような使用に起因する全ての損害、請求、訴訟、費用からMicrochipを擁護し、補償し、免責にすることに同意します。他に言及されない限り、Microchipのどの知的財産権下でも暗黙的または違う方法で許認可は譲渡されません。

商標

品質管理ｼｽﾃﾑ

Microchipの品質管理ｼｽﾃﾑに関する情報についてはhttp://www.microchip.com/qualityを訪ねてください。

日本語© *HERO* 2020.

本応用記述はMicrochipのAN3406応用記述(DS00003341C-2020年5月)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。

世界的な販売とサービス

米国	亜細亜/太平洋	亜細亜/太平洋	欧州
本社 2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel: 480-792-7200 Fax: 480-792-7277 技術支援: http://www.microchip.com/support ｳｪﾌﾞｱﾄﾞﾚｽ: http://www.microchip.com	**ｵｰｽﾄﾗﾘｱ - ｼﾄﾞﾆｰ** Tel: 61-2-9868-6733 **中国 - 北京** Tel: 86-10-8569-7000 **中国 - 成都** Tel: 86-28-8665-5511 **中国 - 重慶** Tel: 86-23-8980-9588 **中国 - 東莞** Tel: 86-769-8702-9880 **中国 - 広州** Tel: 86-20-8755-8029 **中国 - 杭州** Tel: 86-571-8792-8115 **中国 - 香港特別行政区** Tel: 852-2943-5100 **中国 - 南京** Tel: 86-25-8473-2460 **中国 - 青島** Tel: 86-532-8502-7355 **中国 - 上海** Tel: 86-21-3326-8000 **中国 - 瀋陽** Tel: 86-24-2334-2829 **中国 - 深圳** Tel: 86-755-8864-2200 **中国 - 蘇州** Tel: 86-186-6233-1526 **中国 - 武漢** Tel: 86-27-5980-5300 **中国 - 西安** Tel: 86-29-8833-7252 **中国 - 廈門** Tel: 86-592-2388138 **中国 - 珠海** Tel: 86-756-3210040	**ｲﾝﾄﾞ - ﾊﾝｶﾞﾛｰﾙ** Tel: 91-80-3090-4444 **ｲﾝﾄﾞ - ﾆｭｰﾃﾞﾘｰ** Tel: 91-11-4160-8631 **ｲﾝﾄﾞ - ﾌﾟﾈｰ** Tel: 91-20-4121-0141 **日本 - 大阪** Tel: 81-6-6152-7160 **日本 - 東京** Tel: 81-3-6880-3770 **韓国 - 大邱** Tel: 82-53-744-4301 **韓国 - ｿｳﾙ** Tel: 82-2-554-7200 **ﾏﾚｰｼｱ - ｸｱﾗﾙﾝﾌﾟｰﾙ** Tel: 60-3-7651-7906 **ﾏﾚｰｼｱ - ﾍﾟﾅﾝ** Tel: 60-4-227-8870 **ﾌｨﾘﾋﾟﾝ - ﾏﾆﾗ** Tel: 63-2-634-9065 **ｼﾝｶﾞﾎﾟｰﾙ** Tel: 65-6334-8870 **台湾 - 新竹** Tel: 886-3-577-8366 **台湾 - 高雄** Tel: 886-7-213-7830 **台湾 - 台北** Tel: 886-2-2508-8600 **ﾀｲ - ﾊﾞﾝｺｸ** Tel: 66-2-694-1351 **ﾍﾞﾄﾅﾑ - ﾎｰﾁﾐﾝ** Tel: 84-28-5448-2100	**ｵｰｽﾄﾘｱ - ヴｪﾙｽ** Tel: 43-7242-2244-39 Fax: 43-7242-2244-393 **ﾃﾞﾝﾏｰｸ - ｺﾍﾟﾝﾊｰｹﾞﾝ** Tel: 45-4485-5910 Fax: 45-4485-2829 **ﾌｨﾝﾗﾝﾄﾞ - ｴｽﾎﾟｰ** Tel: 358-9-4520-820 **ﾌﾗﾝｽ - ﾊﾟﾘ** Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79 **ﾄﾞｲﾂ - ｶﾞﾙﾋﾝｸﾞ** Tel: 49-8931-9700 **ﾄﾞｲﾂ - ﾊｰﾝ** Tel: 49-2129-3766400 **ﾄﾞｲﾂ - ﾊｲﾙﾌﾞﾛﾝ** Tel: 49-7131-72400 **ﾄﾞｲﾂ - ｶｰﾙｽﾙｰｴ** Tel: 49-721-625370 **ﾄﾞｲﾂ - ﾐｭﾝﾍﾝ** Tel: 49-89-627-144-0 Fax: 49-89-627-144-44 **ﾄﾞｲﾂ - ﾛｰｾﾞﾝﾊｲﾑ** Tel: 49-8031-354-560 **ｲｽﾗｴﾙ - ﾗｰﾅﾅ** Tel: 972-9-744-7705 **ｲﾀﾘｱ - ﾐﾗﾉ** Tel: 39-0331-742611 Fax: 39-0331-466781 **ｲﾀﾘｱ - ﾊﾟﾄﾞﾊﾞ** Tel: 39-049-7625286 **ｵﾗﾝﾀﾞ - ﾃﾞﾙｰﾈﾝ** Tel: 31-416-690399 Fax: 31-416-690340 **ﾉﾙｳｪｰ - ﾄﾛﾝﾊｲﾑ** Tel: 47-72884388 **ﾎﾟｰﾗﾝﾄﾞ - ﾜﾙｼｬﾜ** Tel: 48-22-3325737 **ﾙｰﾏﾆｱ - ﾌﾞｶﾚｽﾄ** Tel: 40-21-407-87-50 **ｽﾍﾟｲﾝ - ﾏﾄﾞﾘｰﾄﾞ** Tel: 34-91-708-08-90 Fax: 34-91-708-08-91 **ｽｳｪｰﾃﾞﾝ - ｲｪｰﾃﾎﾞﾘ** Tel: 46-31-704-60-40 **ｽｳｪｰﾃﾞﾝ - ｽﾄｯｸﾎﾙﾑ** Tel: 46-8-5090-4654 **ｲｷﾞﾘｽ - ｳｫｰｷﾝｶﾞﾑ** Tel: 44-118-921-5800 Fax: 44-118-921-5820