

## AT1886 : AVRGCCでのアセンブリとCの混合

8ビット AVR マイクロコントローラ

## 要点

- Atmel<sup>®</sup> Studio 6での容易なCとアセンブリソースの結合
  - Cはアセンブリルーチン呼び出し可
  - アセンブリはC関数呼び出し可
- Cとアセンブリ間での変数とポインタを渡す。
- 全域変数の共用をCとアセンブリに許す。

## 序説

この応用記述はAtmel Studio 6 IDEを用いてAVRGCCプロジェクトでCとアセンブリの両コードを結合する方法を記述します。この応用記述はCが選択言語で必要または便利のどちらかであるアセンブリ言語が解決策でインクルードされる考え方で書かれます。

Studio 6(と以前の版)は同じプロジェクト内でアセンブリ言語とCファイルの両方での作業を許します。疑問はソフトウェア技術者がアセンブリコードとCコードをどう構成するかです。パラメータ、変数、それとアセンブリコードとCコード間で渡された同様のものはどうですか？

## 目次

---

1. 事前必要条件	3
2. プロジェクト構成	3
3. 関数の視界	3
4. 変数	3
5. レジスタの使い方	3
6. パラメータの渡し方	4
7. 実演の構築と走行の方法	4
8. 必要とされるハードウェア	4
9. ハードウェア構成設定	4
10. 手順	5
11. 参考文献	5
12. 改訂履歴	5

## 1. 事前必要条件

本文書で検討される解決策は以下の技能と技術の基本的な習熟が必要です。もっと学ぶには[11章 参考文献](#)を参照してください。

- Atmel Studio 6
- Atmelのデバugg JTAGICE mk II または JTAGICE 3
- Atmel STK® 600スターキット

## 2. プロジェクト構成

プロジェクトを構成するのに特別な必要条件はありません。Cコードまたはアセンブリのどちらであろうとも、ヘッダファイルとして識別されたファイルはプロジェクトのヘッダファイルフォルダに置かれます。ソースファイルとして識別されたファイルはプロジェクトのソースファイルフォルダに置かれます。

例え厳密に必要とされなくても、全てのアセンブリ言語ソースファイルは“.S”拡張子を持つべきです。これは必要な時に自動的にアセンブラまたはリンカを呼び出すことをCコンパイラ前置処理部に許します。加えて、Cプリプロセッサは抽象的な定数(シンボル定数)の使用の自動的な許諾が実施されます。

ソースファイルの1つはリンカが応用を開始する場所を知るよう、リンカに対して“main”単位部を生成することが必要です。最も一般的なのは“main”と呼ばれる関数を持つCコードです。けれども、“main”と名付けられて(“.global”擬似命令を使って)全域宣言されたサブルーチンを持つアセンブリファイルも“main”と名付けられた単位部を生成します。

単一プロジェクトでのCとアセンブリの組み合わせは応用の必要に応じて様々な疑問を引き起こします。それらの疑問は以下に含まれるかもしれません。

- C関数がアセンブリルーチンを呼ぶことができるように、アセンブリルーチンはCコンパイラに見えるようにどう作ることができますか?
- 同様に、アセンブリルーチンがC関数を呼ぶことができるように、C関数はアセンブリルーチンに見えるようにどう作ることができますか?
- 変数はアセンブリコードにどう渡されますか?
- 変数はC関数にどう渡されますか?
- アセンブリコードとCは同じ全域変数を使うことができますか?

## 3. 関数の視界

C言語関数はアセンブラによって“見える”ためにアセンブリコード内で外部(external)として宣言されることが必要です。

```
.extern my_C_funtion
```

アセンブリ言語ルーチンはCコンパイラで可視であるためにアセンブリコード内で全域(global)として宣言されることが必要です。

```
.global my_assembly_fct
```

加えて、アセンブリ言語ルーチンを呼ぶことを意図されるCファイルは外部であるアセンブリ言語ルーチンの関数原型宣言を持つことが必要です。

```
extern unsigned char my_assembly_fct (unsigned char, unsigned int);
```

## 4. 変数

Cコードとアセンブリコードの両方は独立して変数をアクセスすることができます。実際問題として、Cコードが変数を管理して値または参照のどちらかによってパラメータをアセンブリコードへ渡すのが得策です。[5. レジスタの使い方](#)と[6. パラメータの渡し方](#)の章はレジスタの組がCコンパイラによってどう使われ、パラメータがどう渡されるのかを記述します。

アセンブリとCの両方に対して同じ全域変数をアクセスすることが可能です。そのような変数はCコード内で全域変数であり、アセンブリコード内で外部(external)として宣言されることが必要です。全域が意図される“my\_value”変数を考察してください。Cコードに於いて、それは以下のように、何れかの他の変数のように何れかの関数の外側で宣言されます。

```
unsigned char my_value;
```

アセンブリではそれが以下のようにコード化されます。

```
.extern my_value
```

## 5. レジスタの使い方

Cコードと結合するためのアセンブリ言語ルーチンを書くにはコンパイラがレジスタをどう使うかの知識が必要です。

- R0は一時レジスタでコンパイラが生成したコードによって使われ得ます。このレジスタを使ってC関数を呼ぶアセンブリコードを書く場合、コンパイラがこのレジスタを使うかもしれないので、それを保存して回復することが必要です。
- R1はコンパイラによって常に0を含むと仮定されます。このレジスタを使うアセンブリコードはコンパイラが生成した何れかのコードに戻るまたは呼ぶ前にこのレジスタを解除(=0)すべきです。
- R2~R17,R28,R29はC関数呼び出しがそれらのレジスタを未変化のままにすべきことを意味する“呼び出し保存”レジスタです。これらのレジスタを使うCから呼ばれるアセンブリ言語ルーチンは使うこれらのどのレジスタの内容も保存して回復することが必要です。
- R18~R27,R30,R31はレジスタがどのコードに対しても使用可能を意味する“呼び出し使用”レジスタです。コンパイラが生成したコードは使うこれらのどのレジスタも保存しないため、C関数を呼ぶアセンブリコードはアセンブリコードによって使われるこれらのレジスタのどれをも保存することが必要です。

表5-1.はCとアセンブリ間のレジスタ インターフェースを要約します。

表5-1. Cとアセンブリ間のレジスタ インターフェースの要約

レジスタ	説明	Cから呼ばれるアセンブリコード	Cコードを呼ぶアセンブリコード
R0	一時	使うなら、保存して回復	使うなら、保存して回復
R1	常に0	戻る前に解除(=0)しなければなりません。	呼ぶ前に解除(=0)しなければなりません。
R2~R17	"呼び出し保存"	使うなら、保存して回復	自由に使うことができます。
R28			
R29			
R18~R27	"呼び出し使用"	自由に使うことができます。	使うなら、保存して回復
R30			
R31			

## 6. パラメータの渡し方

固定引数一覧内の引数はR25~R8のレジスタを通して左から右に割り当てられます。全ての引数は偶数のレジスタを使います。これはchar引数が2つのレジスタを消費することに帰着します。レジスタ内に適応する範囲を超えた追加の引数はスタックで渡されます。

変数引数一覧内の引数は右から左への順でスタックに押し込まれます。char引数は2バイトを消費します。

戻り値は戻り値の大きさに依存してR25~R18のレジスタを使います。レジスタとバイト順間の関連は表6-1.で示されます。

表6-1. レジスタとバイト順間の関連

レジスタ	R19	R18	R21	R20	R23	R22	R25	R24
バイト順	バイト7	バイト6	バイト5	バイト4	バイト3	バイト2	バイト1	バイト0

## 7. 実演の構築と走行の方法

実演コードはC内からアセンブリ言語ルーチン呼び出して2つの言語間でパラメータを渡すことを説明するための簡単な関数を含みます。加えて、アセンブリ言語割り込み処理ルーチンが含まれます。

このコードは新しい前置分周器値をこのルーチンに渡してクロック前置分周器を変更する"change\_clock"アセンブリ言語ルーチンと呼ぶことによって始めます。

コードはその後に渡された2つのパラメータを共に加算し、加算の結果をCに返すアセンブリ言語ルーチン呼びます。

コードはその後に周期的な割り込みを生成するためにタイマ/カウンタ0を初期化し、その後にコードは"while"繰り返しに留まります。割り込み発生時、アセンブリ言語割り込み処理ルーチンはポートDのビット0(PD0)を交互切り替えます。

## 8. 必要とされるハードウェア

- Atmel STK600スタータ キット
- (STK600と共に含まれる)STK600-ATmega2560カード
- Atmelのデバッグ(JTAGICE mk II、JTAGICE 3、Atmel AVR Dragon™、など)
- Atmel Studio 6
- (STK600と共に含まれる)10ピンIDCリボン ケーブル

## 9. ハードウェア構成設定

STK600基板上で以下の接続を行ってください。

- STK600上にSTK600-ATmega2560を装着してください。
- ポートD(PORTD)とLEDヘッダ間に10ピン リボン ケーブルを接続してください。
- STK600上でデバイスのJTAGヘッダにデバッグのケーブルを接続してください。

## 10. 手順

1. [www.atmel.com](http://www.atmel.com)からAT1886.zipをダウンロードしてください。
2. 作業ディレクトリ内にファイルを解凍してください。
3. Atmel Studio 6でAVR1886.atstlnプロジェクト ファイルを開いてください。
4. Studioの枠であなたのデバッグ ツールを選んでください。
5. 目的対象デバイス(Target device)としてATmega2560を指定してください。
6. プロジェクトを構築してください。
7. デバッグ動作へ移行してください。
8. 応用を走らせてください。
9. 監視(Watch)ウィンドウに"val\_1"、"val\_2"、"val\_3"を追加してください。
10. mainで"val\_1="指定上に中断点(ブレークポイント)を設定してください。
11. 右のプロセッサ(Processor)タブで、レジスタ(Registers)ウィンドウを拡大してください。
12. 走行(Run)鈕をクリックしてください。コードは"change\_clook"関数を実行してその後中断すべきです。
13. "init\_timer\_0"関数呼び出しに達するまで、次の命令を通して段階実行するために単一段階実行(Single step)機能を使ってください。パラメータがレジスタ経由でCから"add\_two"アセンブリ ルーチンに渡されるのを観察することができます。
14. 一旦"init\_timer\_0"関数に達したなら、(Run)鈕をクリックしてください。これはCを使って計時器を初期化して計時器からの周期的な割り込みを許可します。割り込み処理はアセンブリ言語です。PD0のLED点滅を見るでしょう。

## 11. 参考文献

1. Atmel Studio 6 - [www.atmel.com](http://www.atmel.com)
2. Atmel AVR JTAGICE mk II - [www.atmel.com](http://www.atmel.com)
3. Atmel JTAGICE 3 - [www.atmel.com](http://www.atmel.com)
4. Atmel STK600 - [www.atmel.com](http://www.atmel.com)

## 12. 改訂履歴

文書改訂	日付	注釈
42055A	2012年11月	初版文書公開
42055B	2012年11月	文書はAVR1886からAT1886へ改名



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](http://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  
新大崎勧業ビル 16F  
アトメル ジャパン合同会社  
TEL (+81)(3)-6417-0300  
FAX (+81)(3)-6417-0370

© 2012 Atmel Corporation. 不許複製 / 改訂:42055B-AVR-11/2012

Atmel®、ロゴとそれらの組み合わせ、Enabling Unlimited Possibilities®、AVR®とその他はAtmel Corporationの登録商標または商標またはその付属物です。他の用語と製品名は一般的に他の商標です。

**お断り:** 本資料内の情報はAtmel製品と関連して提供されています。本資料またはAtmel製品の販売と関連して承諾される何れの知的所有権も禁反言あるいはその逆によって明示的または暗示的に承諾されるものではありません。Atmelのウェブサイト位置する販売の条件とAtmelの定義での詳しい説明を除いて、商品性、特定目的に関する適合性、または適法性の暗黙保証に制限せず、Atmelはそれらを含むその製品に関連する暗示的、明示的または法令による如何なる保証も否認し、何ら責任がないと認識します。たとえばAtmelがそのような損害賠償の可能性を進言されたとしても、本資料を使用できない、または使用以外で発生する(情報の損失、事業中断、または利益と損失に関する制限なしの損害賠償を含み)直接、間接、必然、偶然、特別、または付随して起こる如何なる損害賠償に対しても決してAtmelに責任がないでしょう。Atmelは本資料の内容の正確さまたは完全性に関して断言または保証を行わず、予告なしでいつでも製品内容と仕様の変更を行う権利を保留します。Atmelはここに含まれた情報を更新することに対してどんな公約も行いません。特に別の方法で提供されなければ、Atmel製品は車載応用に対して適当ではなく、使用されるべきではありません。Atmel製品は延命または生命維持を意図した応用での部品としての使用に対して意図、認定、または保証されません。

© HERO 2021.

本応用記述はAtmelのAT1886応用記述(doc42055.pdf Rev.42055B-11/12)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識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。