

## AVR1000 : XMEGA用のCコードを書く前に

## 要点

- 命名規定
  - ・レジスタ名
  - ・ビット名
- Cコード名
  - ・ビットと群の遮蔽
  - ・群構成設定の遮蔽
- レジスタのアクセス方法
- 再使用可能な単位部関数の書き方

## 1. 序説

電気製品での高品質と短い開発時間の必要条件が高位プログラミング言語を必要にさせます。主な理由はより良い可搬性と可読性のために高位言語がコードの再使用と保守を容易にすることです。

プログラミング言語の選択だけが可読性と再使用の可能性、良いコード書きの流儀を行うことを保証する訳ではありません。XMEGA<sup>®</sup>の周辺機能、ヘッダファイル、ドライバはこの考えで設計されています。

最も広範囲に使われているAVR<sup>®</sup>マイクロコントローラ用の高位言語はCで、従って本応記述はCプログラミングに集中します。AVRのCコンパイラの殆どを支援するために利用可能なコード例は可能な限りANSI Cで書かれています。少しの例がIAR Embedded Workbench<sup>®</sup>に特化していますが、概念と方法は小変更で他のコンパイラに対して使えます。IAR指定例は明確に記されています。

## 2. XMEGA単位部

AVR XMEGAはAVR CPUコア、SRAM、フラッシュ、EEPROM、いくつかの周辺機能単位部の多数の構成物で構成されます。この構成物は“単位部型(Module type)”と呼ばれます。XMEGAは与えられた単位部型の実物を1つ以上持つことができます。或る単位部型の全ての実物は同じ特性と機能を持ちます。

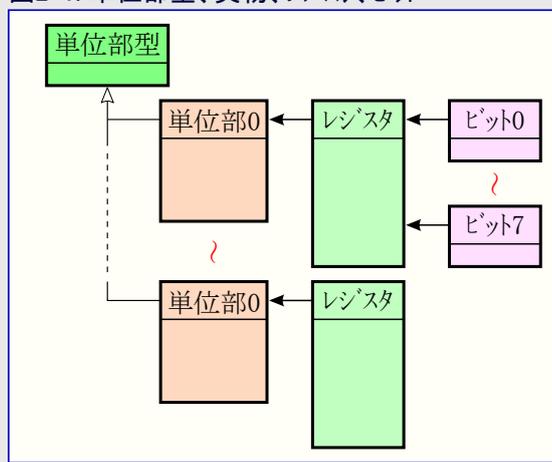
いくつかの単位部型は別の単位部型の下位型で有り得ます。これらは上位型の機能(とレジスタ)の下位型を受け継ぎ、受け継いだ機能の全ては完全に互換です。これは例えばタイマ/カウンタとI/Oポートに適用します。タイマ/カウンタに対する単位部型の下位型は完全なタイマ/カウンタ単位部よりも少ない比較と捕獲のチャネルを持つことを意味します。同様にI/Oポートは8ピンよりも少ないピンを持つかもしれません。

単位部型は“USART”で有り得、一方単位部実物は例えば“USARTC0”で、ここで接尾語“C0”はこの実物が“ポートC”でのUSART番号0であることを示します。単純化のため、これらを区別する必要がなければ、本資料を通じて単位部実物は単位部として参照されます。

各単位部は制御と状態のビットを含むいくつかのレジスタを持ちます。与えられた型の全ての単位部はレジスタの同じ組(または下位の組)を含み、これらのレジスタの全てが制御と状態のビットの同じ組(または下位の組)を含みます。

各単位部はI/Oメモリ割り当て内に固定の基準アドレスを持ち、単位部内に含まれる全てのレジスタは単位部基準アドレスに相対する固定の差分アドレスを持ちます。この方法での各レジスタはI/Oメモリ空間内の絶対アドレスではなく、その差分によって定義される相対アドレスを持ちます。レジスタ差分アドレスは単位部型の全ての実物に対して等しく、ドライバ書きの簡単化した作業は指定型の全単位部に対して使うことができます。

図2-1. 単位部型、実物、レジスタ、ビット



8ビット AVR<sup>®</sup>  
マイクロコントローラ

## 応用記述

本書は一般の方々の便宜のため有志により作成されたもので、Atmel社とは無関係であることを御承知ください。しおりのはじめにでの内容にご注意ください。

Rev. 8075A-02/08, 8075AJ4-05/21

## 2.1. レジスタ命名規則

レジスタは大雑把に制御、状態、データのレジスタに分けて表し、そしてレジスタ命名はこれを反映します。単位部の一般目的制御レジスタはCTRLと名付けられます。単位部内に複数の一般目的制御レジスタが存在する場合、それらはCTRLA, CTRLB, CTRLC、以下同様に名付けられます。これはSTATUSレジスタにも適用します。

特殊機能を持つレジスタについては名前がその機能を反映します。例えば、単位部の割り込みレベルを制御する制御レジスタはINTCTRLと名付けられます。

AVRのデータバス幅が8ビットなので、より大きなレジスタは多数の8ビットレジスタを使って実装されます。16ビットレジスタについては、上位と下位のバイトがそのレジスタ名に各々“H”と“L”を追捕することによってアクセスされます。例えば、16ビットタイマ/カウンタ計数レジスタはCNTと名付けられます。2つのバイトはCNTHとCNTLと命名されます。

16ビットより大きなレジスタについては、バイトが最下位バイトから数値付けされます。例えば、32ビットADC校正レジスタはCALと命名されます。4つのバイトは(最下位バイトから最上位バイトへ)CAL0, CAL1, CAL2, CAL3と名付けられます。

殆どのCコンパイラは複数バイトレジスタに対するアクセスの自動的な扱いを提供します。その場合では、“H”または“L”なしの名前CNTがタイマ/カウンタ計数レジスタに対する16ビットアクセスの実行に使えます。これは32ビットレジスタに対する場合もです。

## 2.2. ビット命名規則

レジスタのビットは個別機能または連帯機能を持つビット群の一部を持ち得ます。個別ビットは単位部の許可ビット、例えばUSARTのENABLEビットで有り得ます。ビット群は2つ以上のビットから成り得、それらが属する単位部の特定構成設定を連帯して選択します。ビット群は $2^n$ 選択までを提供し、ここでのnはビット群のビット数です。USART受信完了割り込みレベルを制御する2つのビット、RXINTLVL1,0はビット群の例です。これら2つのビットは右の選択を提供します。

表2-1. RXINTLVLビットと対応する割り込みレベル選択

RXINTLVL1	RXINTLVL0	割り込みレベル選択
0	0	割り込みOFF
0	1	低レベル割り込み
1	0	中レベル割り込み
1	1	高レベル割り込み

群の一部であるビットは常に数値接尾語を持ちます。群の一部でないビットは決して数値接尾語を持ちません。タイマ/カウンタ制御レジスタDはEVACTとEVSELの2つのビット群を持ちます。これらの群内のビットは数値接尾語を持ち、一方EVDLYはこれがビット群の一部でなく、数値接尾語を持ちません。

表2-2. タイマ/カウンタ制御レジスタD(CTRLD)内のビットに対するビット群とビット名

ビット群	EVACT			-	EVSEL			
	EVACT2	EVACT1	EVACT0	EVDLY	EVSEL3	EVSEL2	EVSEL1	EVSEL0
ビット番号	7	6	5	4	3	2	1	0

## 3. XMEGA用Cコードの書き方

後続項はXMEGA用のCコードの書き方に焦点を当てます。例は異なるXMEGAデバイス間の可搬性と高い可読性のコードの作成方法を示します。例は検証と保守が容易なコードを書く指針にも使えます。

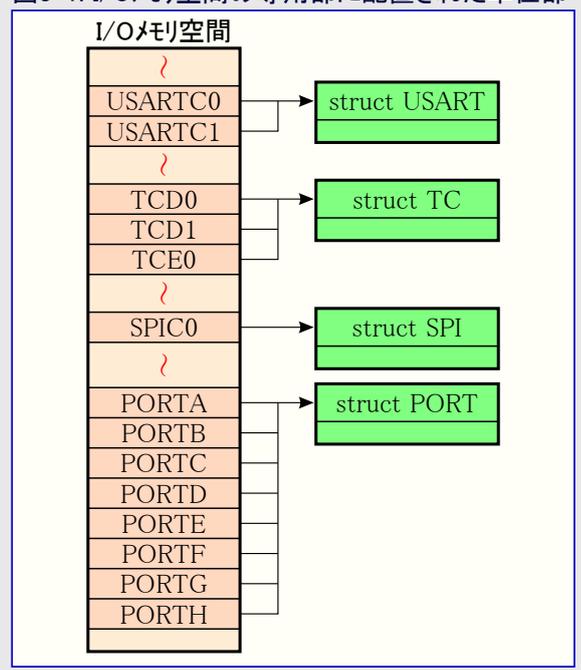
XMEGAの単位部はメモリ空間で専用の連続した塊で配置され、カプセル内の単位群として見る事ができます。これはCコード書き時の単位部がアクセスされる方法を反映しています。単位部はC構造体を使ってカプセル化され、そしてそれには単位部の全レジスタが含まれます。図3-1はこの図式を示します。

いくつかのレジスタが単位部と直接の関連を持たないことに注意してください。構造体が単位部に関連するレジスタに使われるため、これらは構造体でカプセル化されません。

より大きなコード作業に関しては単位部構造が可読性ではなく、コンパイラが単位部ドライバの再使用をできるため、それによって非常に簡潔なコード作成の利点を提供します。これは後でもっと詳細に記述されます。

本資料は命令規則とAVRプログラミングの熟練者が使うのとは異なるアクセス方法を導入しますが、レジスタアクセスの“旧来の方法”も未だヘッダファイルによって支援されていることも留意されるべきです。これはビットレベルにも適用します。

図3-1. I/Oメモリ空間の専用部に配置された単位部



### 3.1. XMEGA ヘッド ファイル

各XMEGAデバイスに対して専用のヘッド ファイルが利用可能です。目的デバイスが(IAR EWAVR用IDEでその使用と仮定する)プロジェクト設定で指定されている場合、コード一覧3-1.で示されるようにデバイス ファイルがインクルードされるなら、IARコンパイラは正しいヘッド ファイルを自動的にインクルードします。

#### コード一覧3-1. IARヘッド ファイルのインクルード

```
#include <ioavr.h>
```

この利点は目的デバイス変更の場合にソースファイル変更の必要がなく、プロジェクト設定の変更だけなことです。

### 3.2. 単位部のレジスタ

I/O割り当ては与えられた周辺機能単位部に関する全レジスタが1つの連続するメモリ部に配置されるように展開されます。違う単位部に属するレジスタは混ぜられません。これは全ての周辺機能単位部をC構造体に構成することを可能にし、ここで構造体のアドレスはその単位部の基準アドレスを定義します。単位部に属する全レジスタがその単位部構造体の要素です。

例は設定可能な多段割り込み制御器(PMIC)単位部です。この単位部に対する構造体宣言はコード一覧3-2.で示され、この例はコード一覧3-3.で使われます。コード一覧3-3.の例はPMICと名付けられたPMIC\_t型との仮定であることに注意してください。これは本資料に於いて後で網羅されます。

#### コード一覧3-2. 単位部構造体宣言

```
typedef struct PMIC_struct {
    unsigned char STATUS;           // 状態レジスタ
    unsigned char INTPRI;          // 割り込み優先権
    unsigned char CTRL;           // 制御レジスタ
} PMIC_t;
```

#### コード一覧3-3. 単位部構造体の使い方

```
unsigned char temp;
temp = PMIC.STATUS;              // tempへ状態レジスタ読み込み
PMIC.CTRL |= PMIC_PMRPE_bm;     // 制御レジスタ内のPMRPEビット設定(1)
```

#### 3.2.1. 単位部構造体内の複数語レジスタ

いくつかのレジスタは16または32ビットを表すために他のレジスタと連結して使われます。例としてコード一覧3-4.で示されるADC構造体宣言を見ることができます。

#### コード一覧3-4. ADC構造体宣言

```
typedef struct ADC_struct {
    unsigned char CH0MUXCTRL;      // チャネル0多重器(MUX)制御レジスタ
    unsigned char CH1MUXCTRL;      // チャネル1多重器(MUX)制御レジスタ
    unsigned char CH2MUXCTRL;      // チャネル2多重器(MUX)制御レジスタ
    unsigned char CH3MUXCTRL;      // チャネル3多重器(MUX)制御レジスタ
    unsigned char CTRLA;           // 制御レジスタA
    unsigned char CTRLB;           // 制御レジスタB
    unsigned char REFCTRL;         // 基準電圧制御レジスタ
    unsigned char EVCTRL;         // 事象制御レジスタ
    WORDREGISTER(CH0RES);          // チャネル0結果レジスタ
    WORDREGISTER(CH1RES);          // チャネル1結果レジスタ
    WORDREGISTER(CH2RES);          // チャネル2結果レジスタ
    WORDREGISTER(CH3RES);          // チャネル3結果レジスタ
    unsigned char reserved_0x10;
    unsigned char reserved_0x11;
    unsigned char CH0INTCTRL;      // チャネル0割り込み制御レジスタ
    unsigned char CH1INTCTRL;      // チャネル1割り込み制御レジスタ
    unsigned char CH2INTCTRL;      // チャネル2割り込み制御レジスタ
    unsigned char CH3INTCTRL;      // チャネル3割り込み制御レジスタ
    unsigned char INTFLAGS;        // 割り込み要求フラグ レジスタ
    WORDREGISTER(CMP);             // 比較値レジスタ
    unsigned char PRESCALER;        // クロック前置分周レジスタ
    unsigned char reserved_0x1A;
    unsigned char reserved_0x1B;
    unsigned char CALCTRL;         // 校正制御レジスタ
    DWORDREGISTER(CAL);            // 校正值レジスタ
} ADC_t;
```

コード一覧3-4.ではADCチャネル結果レジスタCH0RES, CH1RES, CH2RES, CH3RESと比較レジスタCMPが16ビット値です。コード一覧3-5.で示されるWORDREGISTERマクロを使って宣言されます。校正レジスタCALは32ビット値で、コード一覧3-6.で示されるDWORDREGISTERマクロを使って宣言されます。

#### コード一覧3-5. WORDREGISTERマクロ

```
#define WORDREGISTER(regname) ¥
union { ¥
    unsigned short regname; ¥
    struct { ¥
        unsigned char regname ## L; ¥
        unsigned char regname ## H; ¥
    }; ¥
}
```

#### コード一覧3-6. DWORDREGISTERマクロ

```
#define DWORDREGISTER(regname) ¥
union { ¥
    unsigned long regname; ¥
    struct { ¥
        unsigned char regname ## 0; ¥
        unsigned char regname ## 1; ¥
        unsigned char regname ## 2; ¥
        unsigned char regname ## 3; ¥
    }; ¥
}
```

ここで見られるように、WORDREGISTERマクロは上位と下位のバイトに対して各々“H”と“L”の接尾語を使います。WORDREGISTERマクロはバイト順を示すために数値接尾語を使います。16ビットと32ビットの両レジスタはコード一覧3-7.で示されるように、接尾語なしのレジスタ名を使うことによって、16ビット/32ビット動作でアクセスすることができます。

#### コード一覧3-7. 大きさの違うレジスタのアクセス

```
unsigned char tempByte;
unsigned int tempWord;
unsigned long tempDword;

tempByte = ADCA0. CTRLA; // 制御レジスタA読み込み
tempWord = ADCA0. CH0RES; // チャネル0の16ビット結果読み込み
tempDword = ADCA0. CAL; // 32ビット校正值読み込み
```

コード一覧3-7.は単一バイトレジスタCTRLAの読み方、16ビット操作を使う2つのCH0RESH/Lレジスタの読み方、32操作を使う4つのCAL3/2/1/0レジスタの読み方を示します。Cコンパイラは複数バイトレジスタを自動的に扱います。けれども或る状態で、不正を避けるために非分断操作での読み書きが必要とされるかもしれないことに注意してください。この場合は割り込み処理ルーチンが複数バイトアクセスを妨害しないことを保証するために、複数バイトアクセスの間、割り込みが禁止されなければなりません。AVR1306応用記述はXMEGAのタイマ/カウンタ単位部に対して行われるレジスタの非分断アクセス法の例を含みます。

### 3.3. 単位部アドレス

全ての周辺機能単位部の定義はXMEGAに関して利用可能なデバイスヘッダファイルで得られます。単位部のアドレスは利用可能な殆どのCコンパイラと互換にするためにANSI Cで指定されています。コード一覧3-8.はポートAでのADC 0がどう定義されているかを示します。

#### コード一覧3-8. 周辺機能単位部定義

```
#define ADCA (*(volatile ADC_t *) 0x0200)
```

コード一覧3-8.は単位部実物の基準アドレスと一致するメモリ内の絶対アドレスへの直参照位置指示子を使う単位部実物定義法を示します。単位部の位置指示子はXMEGAヘッダファイルで予め定義されており、従ってソースコードでこれらの定義を追加する必要はありません。

### 3.4. ビット遮蔽値とビット群遮蔽値

レジスタのビットは予め定義された遮蔽値、または代わりにビット位置値を使って操作することができます。ビット位置は殆どの作業に対して推奨されません。予め定義されているビット遮蔽値はビットマスクまたはビット群、ビット群マスクまたは短縮して群マスクと呼ばれる、個別ビットに関連したどれかです。

ビット遮蔽値は個別ビットの設定(1)と解除(0)を行う時の両方で使われます。ビット群遮蔽値は主にビット群内の複数ビットを解除(0)する時に使われます。ビット群の部分である複数ビットの設定(1)は3.5項で網羅されます。

#### 3.4.1. ビット遮蔽値

タイマ/カウンタ制御レジスタD(CTRLD)を考察してください。このレジスタのビット群、ビット名、ビット位置、ビット遮蔽値は表3-1.で見ることができます。

表3-1. タイマ/カウンタ制御レジスタD(CTRLD)内のビットに対するビット群、ビット名、ビット遮蔽値

ビット群	EVACT			-	EVSEL			
ビット名	EVACT2	EVACT1	EVACT0	EVDLY	EVSEL3	EVSEL2	EVSEL1	EVSEL0
ビット番号	7	6	5	4	3	2	1	0
ビット遮蔽値	\$80	\$40	\$20	\$10	\$08	\$04	\$02	\$01

ビットの名前がそれらを扱うためにコンパイラに対して固有であることが必要なので、全てのビットはそれらが属する単位部型で接頭語を付けられています。多くの場合で、単位部型名は略名にされています。タイマ/カウンタ単位部に関連する全てのビット定義に関しては、ビット名が"TC\_"の接頭語を付けられています。

ビット遮蔽値とビット位置を区別するために接尾語も追加されます。ビット遮蔽値に対する接尾語は"\_bm"です。従ってEVDLYビットに対するビット遮蔽値名はTC\_EVDLY\_bmです。コード一覧3-9.はビット遮蔽値の代表的な使い方を示します。タイマ/カウンタD0のCTRLDレジスタ内のEVDLYビットが設定(1)され、レジスタ内の他の全てのビットを無変化のままにします。

#### コード一覧3-9. ビット遮蔽値の使い方

```
TCDO. CTRLD |= TC_EVDLY_bm; // ビット遮蔽値指定での設定(1)
```

#### 3.4.2. ビット群遮蔽値

多くの関数がビット群によって制御されています。タイマ/カウンタCTRLDレジスタのEVACT2~0とEVSEL3~0は群化されたビットです。群内のビット値が特定構成設定を選択します。

ビット群内のビット変更時に新しい値を指示する前にビット群を解除することが度々必要とされます。別の方法でそれにするには、設定(1)すべきビットを設定(1)することは不十分で、解除(0)すべきビットを解除(0)することも必要とされます。これを容易に行うためにビット群遮蔽値が定義されています。群遮蔽値はビット群内のビットと同じ名前を使い、"\_gm"の接尾語を付けられています。

コード一覧3-10.は群遮蔽値がビット遮蔽値とどう関連するかを示します。現実には群遮蔽値がヘッダファイルで予め計算されていて、故にコンパイラは何度も何度も同じ定数を計算する必要がありません。

#### コード一覧3-10. 群遮蔽値とビット遮蔽値の関連

```
#define TC_EVACT_gm (TC_EVACT2_bm | TC_EVACT1_bm | TC_EVACT0_bm)
```

ビット群遮蔽値は新しい値を書く前にビット群の旧構成設定を解除することを主に意図されています。コード一覧3-11.はこれがどう行われ得るかを示します。コードはタイマ/カウンタD0のCTRLDレジスタ内のEVACTビット群を解除します。この構成はそれ自身であり有用ではありません。群遮蔽値は代表的に群構成設定遮蔽値と連結して使われ、これは3.5項で網羅されます。

#### コード一覧3-11. 群遮蔽値の使い方

```
TCDO. CTRLD &= ~(TC_EVACT_gm); // 群遮蔽値での群ビット解除
```

### 3.5. ビット群構成設定遮蔽値

望む構成設定にビット群を設定する時に何のビット模様が使われる必要があるかを調査するためにデータシートを調べるのが度々必要とされます。これはコードを読むまたはデバッグする時にも適用されます。可読性を増してビット群での不正なビット設定の可能性を最小とするために、多数の群構成設定遮蔽値が利用可能にされています。

群構成設定遮蔽値の名前は単位部型、群名、構成設定内容と接尾語"\_gc"との連結で、これが群構成設定を示します。例については図3-2.をご覧ください。

図3-2. 群構成設定名合成(例としてUSART受信完了割り込みレベルビットを使用)



図3-2.の群構成設定を調査すると、USART単位部でRXCINTLVLビットに対する構成設定を選択するのにそれが使われているのを見ることができます。この指定群構成設定は高(HI)割り込みレベルを選択します。

受信完了割り込みレベルに対するビット群は2ビットのRXINTLVL1,0から成ります。表3-2.はこのビット群に対して利用可能な群構成設定を示します。

構成設定名は"OFF"、"LO"、"MED"、"HI"です。これらの名前は特定の構成設定遮蔽値が何を構成設定するかを理解するための非常に小さな努力を必要とすることで、コードを書きそして保守することを大変容易にします。

表3-2. RXINTLVLビットと対応する単位部割り込みの割り込みレベル

RXINTLVL1	RXINTLVL0	割り込みレベル	群構成設定遮蔽値
0	0	割り込みOFF	USART_RXCINTLVL_OFF_gc
0	1	低レベル割り込み	USART_RXCINTLVL_LO_gc
1	0	中レベル割り込み	USART_RXCINTLVL_MED_gc
1	1	高レベル割り込み	USART_RXCINTLVL_HIGH_gc

ビット群を新しい構成設定に変更するには、旧構成設定が先に消去されるのを保証するために、ビット群構成設定が代表的にビット群遮蔽値と連結して使われます。コード一覧3-12.はUSART C0受信完了割り込みレベルを中レベルに再構成設定するのに群遮蔽値と構成設定遮蔽値がどう使われ得るかを示します。

#### コード一覧3-12. ビット群構成設定の変更

```
USARTC0. INTCTRL = (USARTC0. INTCTRL & ~USART_RXCINTLVL_gm ) | USART_RXCINTLVL_MED_gc;
```

それにも関わらず、コード一覧3-12.のコードを2つの独立したコード行、ビット群を解除する1つと新しい値を設定する1つに分割する誘惑に駆られるかもしれないことに注意してください。これは推奨されません。INTCTRLレジスタがvolatileとして定義されているので、2つの独立したコード行は1度に代えて2度のINTCTRLレジスタ読み書きを起動します。コードを非効率にすることに加えて、これは周辺機能を予期せぬ状態に置き得ます。

ビット解除のための群遮蔽値の使用は常に必要な訳ではありません。コード一覧3-13.はUSART C0の全ての割り込みレベルが同時にどう構成設定され得るかを示します。受信完了、送信完了、USARTデータレジスタ空の割り込みレベルは各々中(MEDIUM)、なし(OFF)、低(LOW)の割り込みレベルに設定されます。

#### コード一覧3-13. レジスタ内の全構成設定同時設定

```
USARTC0. INTCTRL = USART_RXCINTLVL_MED_gc |
                    USART_TXCINTLVL_OFF_gc |
                    USART_DREINTLVL_LO_gc;
```

### 3.5.1. 群構成設定遮蔽値の列挙

ビット遮蔽値や群遮蔽値のようではなく、ビット構成設定遮蔽値はCの列挙子を使って定義されています。1つの列挙子は各ビット群に対して定義されています。USART TXCINTLVLビットに対する列挙子はコード一覧3-14.で示されます。

#### コード一覧3-14. USART TXCINTLVL列挙子

```
typedef enum {
    USART_RXCINTLVL_OFF_gc = (0x00 << 4);
    USART_RXCINTLVL_LO_gc  = (0x01 << 4);
    USART_RXCINTLVL_MED_gc = (0x02 << 4);
    USART_RXCINTLVL_HI_gc  = (0x03 << 4);
} USART_RXCINTLVL_t;
```

コード一覧3-14.で見えるように、列挙子の名前は単位部型(USART)、ビット群(RXCINTLVL)、接尾語(t)と結び付き、これはデータ型を示します。

列挙子定数の各々はそれ自体で使われる時に大抵通常の数値のように振舞います。けれども、列挙子を使うことはそれが新しいデータ型を作成することの利点を持ちます。USART\_RXCINTLVL\_tは丁度intやcharのように型名として使えます。列挙子定数は整数(integer)として直接使えますが、整数の列挙型割り当てはコンパイラの警告を起動するでしょう。これは書き手の利点として使うことができます。USART単位部に対する受信完了割り込みレベル設定を受け入れる関数を思い浮かべてください。関数が整数型(例えばunsigned char)として割り込みレベルを受け入れる場合、正当または不当などどんな値も関数を通り得ます。関数が代わりにUSART\_RXCINTLVL\_t型のパラメータを受け入れるなら、USART\_RXCINTLVL\_tで予め定義された4つの定数だけがその関数を通ることができます。他の何かの通過はコンパイラの警告に終わります。

コード一覧3-14.の定数が実際のビット位置に移動されていることに注意してください。これは列挙子定数がレジスタに書かれるべき実際の値であることを意味します。追加の移動は必要ありません。

### 3.6. 関数呼び出しと単位部ドライバ

複数の実物を持つ単位部型用のドライバを書くとき、全ての実物が同じレジスタメモリ割り当てを持つ事実は、その単位部型の全ての実物に対して再使用可能なドライバを作成することで利用可能になります。ドライバが適切な単位部実物を指示するポインタ引数を取る場合、そのドライバはこの型の全単位部に対して使うことができます。可搬性を考慮するとき、これは偉大な優位性を表します。

8つのタイマ/カウンタを持つデバイスを考察してください。タイマ/カウンタ単位部を初期化してアクセスするための関数は全ての単位部実物で共用できます。例えば単位部ポインタを関数へ渡す小さな付随負荷があるとは言え、各単位部型の全ての実物に対してコードが再使用されるので、総コード量は頻繁に低減されるでしょう。更に重要なのは、開発時間、保守費用、可搬性がこの方法を使うことによって大いに改善され得ることです。

コード一覧3-15は何れかのタイマ/カウンタ単位部に対してクロック元を選択するために単位部ポインタを使う関数を示します。

コード一覧3-15. 単位部実物ポインタを使う関数の例

```
void TC_ConfigClockSource(volatile TC_t * tc, TC_CLKSEL_t clockSelection)
{
    tc->CTRLA = tc->CTRLA & ~TC_CLKSEL_gm | clockSelection;
}
```

この関数は単位部ポインタの型TC\_tと群構成設定型TC\_CLKSEL\_tの2つの引数を取ります。関数内のコードはtcパラメータを通して提供されるタイマ/カウンタ単位部に対して、新しいクロック選択を設定するためのCTRLAレジスタのアクセスに、タイマ/カウンタ単位部ポインタを使います。

コード一覧3-16は異なるクロック選択で異なるタイマ/カウンタを構成設定するのにコード一覧3-15の関数がどう使われ得るかを示します。

コード一覧3-16. パラメータとして単位部ポインタを取る関数の呼び出し

```
TC_ConfigClockSource (&TCC0, TC_CLKSEL_OFF_gc);
TC_ConfigClockSource (&TCC1, TC_CLKSEL_DIV1_gc);
TC_ConfigClockSource (&TCD0, TC_CLKSEL_DIV2_gc);
TC_ConfigClockSource (&TCD1, TC_CLKSEL_DIV1024_gc);
```

## 4. コード書きの代替法

利便性とプログラミングの流儀変更をAVRプログラミングの熟練者に強いるのを避けるため、構造体を伴わないプログラミングの流儀が未だ使用可能です。それはmegaAVR®に対して使われるビット名形式を使うことも可能です。本項はビット名の使用とレジスタアクセスの代替方法を簡単に記述します。

### 4.1. レジスタ名

単位部構造体を使わないでどのレジスタもアクセスすることが可能です。レジスタを直接参照するには、単位部実物名とレジスタ名を下線(アンダースコア)で連結してください。同じ命名規則がアセンブリ言語プログラミング時に使われます。

例: タイマ/カウンタC0のCTRLAレジスタにアクセスするには、名前としてTCC0\_CTRLAを使ってください。

### 4.2. ビット位置

ビットを設定(1)または解除(0)するのにビット遮蔽値を使うことが可能です。レジスタ内のビット位置はビット位置に対する追加の接尾語"\_bp"で、ビット遮蔽値と同じ名前を使って定義されています。コード一覧4-1はレジスタ構成設定にビット位置がどう使われ得るかを示します。

コード一覧4-1. レジスタアクセス代替コード

```
PORTB_OUT = (1 << PORTB_OUT0_bp); // ポートB出力(PORTB_OUT)のビット0を設定(1)
```

ビット位置定義は互換性の理由でインクルードされています。これらはビット番号を使う命令に対してアセンブリ言語でプログラミングする時にも必要とされます。

## 5. 要約

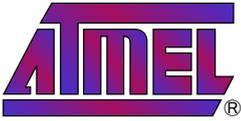
参照用に、ビット構成設定を扱う時に使われる各種接尾語の概要が表5-1で一覧されます。

XMEGAに対してCコードを書くのに提唱された方法を使うことは、必須を意味しませんが、堅強、可搬性、再使用性、高い可読性のコードを保証するために、提供される利点の使用が考慮されるべきです。より大きな企画とより多くの機能のデバイスはより大きな優位性を持ちます。小さな企画に関しては恩恵がないと主張し得ますが、その主張をひっくり返すと不利もまたないと言うことができます。

1つ確かなことがあります。それは良いコードの流儀は常に利点です。

表5-1. 接尾語の概要

接尾語	意味	例
_gm	群遮蔽値	TC_CLKSEL_gm
_gc	群構成設定値	TC_CLKSEL_DIV1_gc
_bm	ビット遮蔽値	TC_CCAEN_bm
_bp	ビット位置	TC_CCAEN_bp



## 本社

### *Atmel Corporation*

2325 Orchard Parkway  
San Jose, CA 95131  
USA  
TEL 1(408) 441-0311  
FAX 1(408) 487-2600

## 国外営業拠点

### *Atmel Asia*

Unit 1-5 & 16, 19/F  
BEA Tower, Millennium City 5  
418 Kwun Tong Road  
Kwun Tong, Kowloon  
Hong Kong  
TEL (852) 2245-6100  
FAX (852) 2722-1369

### *Atmel Europe*

Le Krebs  
8, Rue Jean-Pierre Timbaud  
BP 309  
78054 Saint-Quentin-en-  
Yvelines Cedex  
France  
TEL (33) 1-30-60-70-00  
FAX (33) 1-30-60-71-11

### *Atmel Japan*

104-0033 東京都中央区  
新川1-24-8  
東熱新川ビル 9F  
アトメル ジャパン株式会社  
TEL (81) 03-3523-3551  
FAX (81) 03-3523-7581

## 製品窓口

### ウェブサイト

[www.atmel.com](http://www.atmel.com)

### 技術支援

[avr@atmel.com](mailto:avr@atmel.com)

### 販売窓口

[www.atmel.com/contacts](http://www.atmel.com/contacts)

### 文献請求

[www.atmel.com/literature](http://www.atmel.com/literature)

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のウェブサイトに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、または利益の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。

© Atmel Corporation 2008. 不許複製 Atmel®、ロコとそれらの組み合わせ、AVR®とその他はAtmel Corporationの登録商標、XMEGA®とその他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。

## © HERO 2021.

本応用記述はAtmelのAVR1000応用記述(doc8075.pdf Rev.8075A-02/08)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。