TB3262

AVR1000b: AVR® MCUに対してCコードを書く前に

序説

著者: Cristina Ionescu, Cristian Săbiuţă, Microchip Technology Inc.

この技術概説はより読み易くて再利用可能なｺｰﾄﾞを書くのを助けるｺｰﾄﾞ書きの指針を定義し、AVR®ﾏｲｸﾛ ｺﾝﾄﾛｰﾗ(MCU)を成功裏にﾌﾟﾛｸﾞﾗﾑするのに推奨される手順を提供します。

課せられた高品質と短い開発時間の必要条件のため、高位ﾌﾟﾛｸﾞﾗﾐﾝｸﾞ言語が必要になりました。それらは各ﾏｲｸﾛ ｺﾝﾄﾛｰﾗ基本設計に対して特有の低位命令よりもより良い可搬性と可読性のため、保守と再利用を容易にします。

ﾌﾟﾛｸﾞﾗﾐﾝｸﾞ言語だけでは可読性と再利用は保証されませんが、良いｺｰﾄﾞ書きの流儀は保証されます。従って、AVR MCU周辺機能、ﾍｯﾀﾞ ﾌｧｲﾙ、ﾄﾞﾗｲﾊﾞはこの仮定に従って設計されています。

最も広範囲に使われるAVRﾏｲｸﾛ ｺﾝﾄﾛｰﾗ用高位言語はCで、故に本文書はCﾌﾟﾛｸﾞﾗﾐﾝｸﾞに集中します。殆どのAVR Cｺﾝﾊﾟｲﾗでの互換性を保証するため、本文書でのｺｰﾄﾞ例はANSI Cｺｰﾄﾞ書き基準で書かれています。

本文書はAtmel Studio統合開発環境(IDE:Integrated Development Environment)で開発されたｺｰﾄﾞ例を含みます。殆どのｺｰﾄﾞ例は「5. 更なる段階」章で提供される他のIDEもに適合します。

目次

1. ﾃﾞｰﾀｼｰﾄ単位部基本構造と命名規定

ﾏｲｸﾛ ｺﾝﾄﾛｰﾗに対するCｺｰﾄﾞ書きでの最初の段階はﾌﾟﾛｸﾞﾗﾑを書くのに使われるﾃﾞﾊﾞｲｽのﾃﾞｰﾀｼｰﾄで何の情報形式を見つけることができるかを知って理解することです。ﾃﾞｰﾀｼｰﾄはﾏｲｸﾛ ｺﾝﾄﾛｰﾗの機能、ﾒﾓﾘ、ｺｱと周辺機能、周辺機能単位部の機能的な記述、周辺機能基準ｱﾄﾞﾚｽ、ﾚｼﾞｽﾀの名前とｱﾄﾞﾚｽ、他の機能と電気的な特性についての情報を含みます。

1.1. ﾃﾞｰﾀｼｰﾄの見つけ方

Microchip製品に関連するどの文書も次で見つけることができます。

- Microchipﾃﾞｰﾀｼｰﾄ

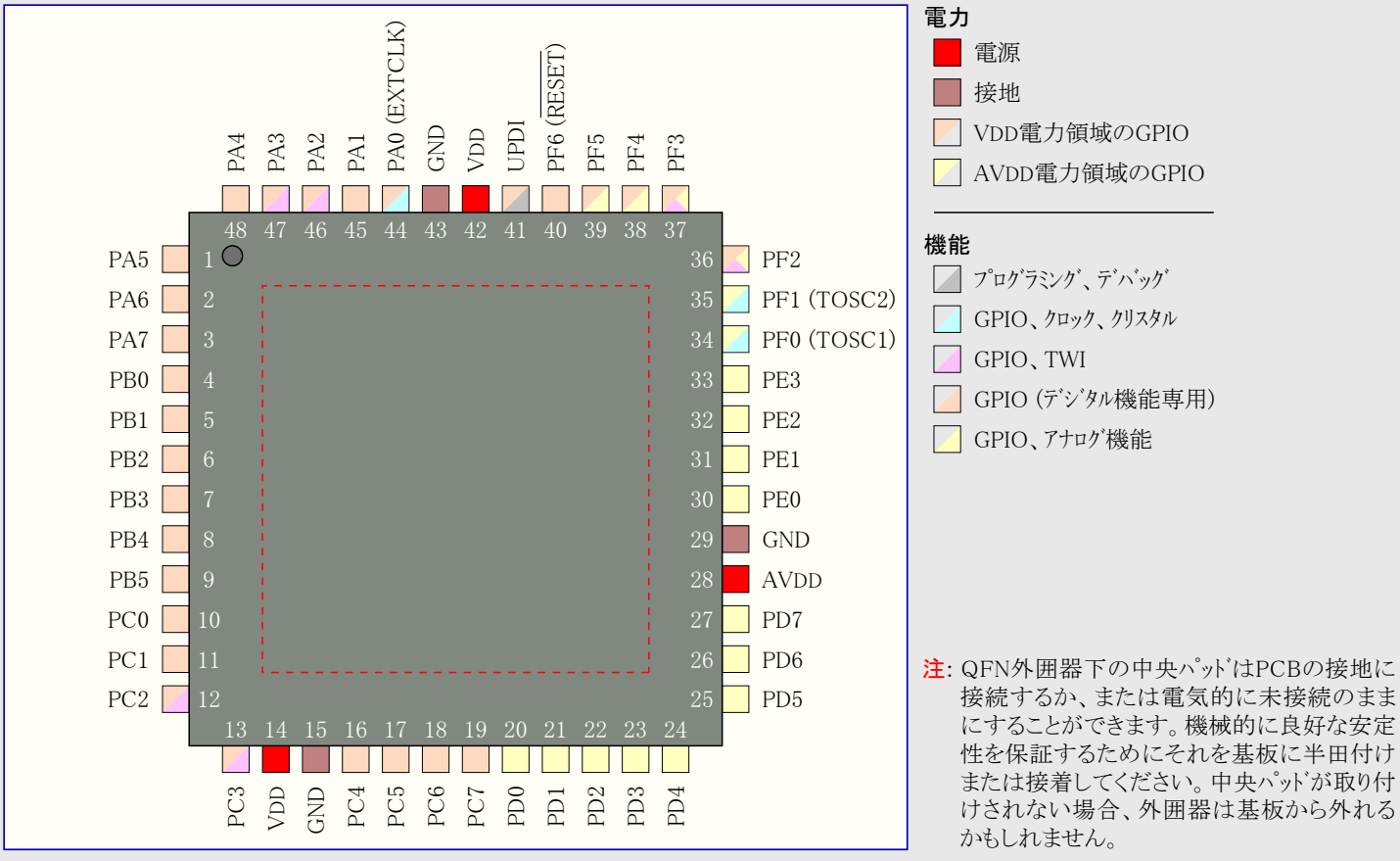
本文書に関係するﾃﾞﾊﾞｲｽ系列に対するﾃﾞﾊﾞｲｽのﾃﾞｰﾀｼｰﾄは以下で見つけることができます。

- ATtiny416概要
- ATtiny212概要
- ATtiny3217概要
- ATtiny1634概要
- ATmega4809概要
- ATmega808概要
- AVR128DA28概要

1.2. ﾋﾟﾝ記述

ﾋﾟﾝ記述はどのﾃﾞｰﾀｼｰﾄでも見つけることができます。ﾋﾟﾝ配置やﾃﾞﾊﾞｲｽに依存する他のどの命名規定も「ﾋﾟﾝ配置」、「ﾋﾟﾝ構成設定」の部分に含まれます。48ﾋﾟﾝのATmega809/1609/3209/4809ﾃﾞﾊﾞｲｽのﾋﾟﾝ配置が図1-1.で示されます。

図1-1. 48ﾋﾟﾝのATmega809/1609/3209/4809のﾋﾟﾝ配置(1)



注: QFN外囲器下の中央ﾊﾟｯﾄﾞはPCBの接地に接続するか、または電気的に未接続のままにすることができます。機械的に良好な安定性を保証するためにそれを基板に半田付けまたは接着してください。中央ﾊﾟｯﾄﾞが取り付けされない場合、外囲器は基板から外れるかもしれません。

各入出力ﾋﾟﾝに対する構成設定可能な機能性はﾃﾞﾊﾞｲｽに応じて、「入出力多重化と考察」章、または「入出力ﾎﾟｰﾄ」章の「代替ﾎﾟｰﾄ機能」項で記述されます。AVR128DA48 Curiosity Nanoのような評価基板が使われる場合、ﾏｲｸﾛ ｺﾝﾄﾛｰﾗのﾋﾟﾝが指定基板でどう割り当てられるかを知ることが必要です。この情報はAVR128DA48 Curiosity Nanoｳｪﾌﾞ頁のAVR128DA48 Curiosity Nano回路図資料で入手可能です。AVR128DA48 Curiosity Nano基板とﾏｲｸﾛ ｺﾝﾄﾛｰﾗ特性を記述する他の文書は同じｳｪﾌﾞ頁で入手可能です。

1.3. 単位部記述

AVRマイクロ コントローラはAVR CPU、SRAM、フラッシュ、EEPROM、単位部型と呼ばれるいくつかの周辺機能単位部で構成されます。この文書を通して、全ての周辺機能単位部は単位部として参照されます。

より新しいAVRマイクロ コントローラ系列は与えられた単位部型の1つまたは複数の実体を持ち得ます。全ての実体は同じ特徴と機能を持ちます。いくつかの単位部型は(同型)他の部分集合でそれらの機能のいくつかを受け継ぎます、受け継がれた機能は各々の単位部型と完全互換です。例えば、計時器の部分型は完全な計時器型よりもより少ない比較と捕獲のチャネルを持ち得ます。

(例えば)単位部型は万能同期/非同期送受信器(USART)で有り得る一方で、単位部実体は例えばUSART0で、ここで接尾語0はこの実体が"USART番号0"であることを示します。単純化のため、区別する必要がない限り、本文書を通して単位部実体は単位部として参照されます。

各単位部はI/Oメモリ割り当て図で固定の基準アドレスを持ち、単位部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変位(差分)アドレスを持ちます。このように、各レジスタはI/Oメモリ空間内の絶対アドレスを持たず、それの変位によって定義される相対アドレスを持ちます。レジスタ変位アドレスは単位部型の全ての実体に対して等しく、特定の型の全ての単位部に使うことができるドライバ書きの作業を簡単化します。周辺機能単位部アドレス割り当てはデータシートで見つけることができ、各周辺機能に対する基準アドレスを示します。

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(または部分集合)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(または部分集合)を含みます。

表1-1.はATtiny804/1604の周辺機能のいくつかの基準アドレスを示します。

図1-2. 単位部型、実体、レジスタ、ビット

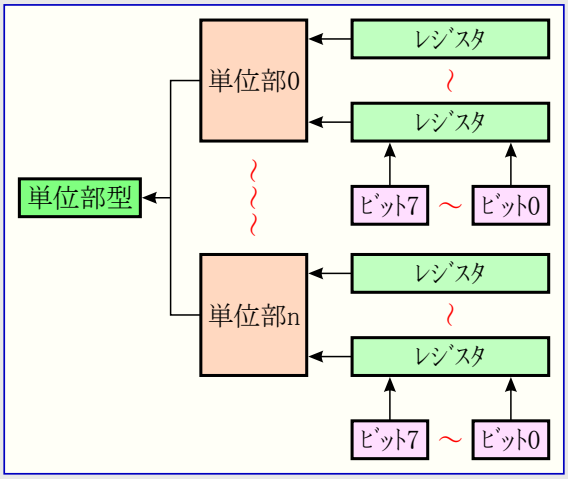


表1-1. 周辺機能単位部アドレス配置(1)

基準アドレス	名称	説明
$0000	VPORTA	仮想ポートA
$0004	VPORTB	仮想ポートB
$001C	GPIO	汎用I/Oレジスタ
$0030	CPU	CPU
$0040	RSTCTRL	リセット制御器
$0050	SLPCTRL	休止制御器
$0060	CLKCTRL	クロック制御器
≀	≀	≀

全ての単位部はその単位部が持つ特徴、単位部の機能的な説明、説明された用語全てと共に或る動作形態を構成設定する方法に於ける具体的な信号と指針を提供する専用部分(章)を持ちます。単位部章の最後には、全てのレジスタの範囲、初期値、それが読み込み可または書き込み可かを含むレジスタ記述部分項があります。全ての構成設定可能/アクセス可能なレジスタのビットの位置も提供します。

全てのレジスタ、それらのアドレス変位、ビット名と位置が各単位部に対して「レジスタ概要」項で記述されます。ADC単位部に対するレジスタ概要が図1-3.で示されます。

図1-3. ADCレジスタ概要(1)

変位	略称	ビット位置	ビット7	ビット6	ビット5	ビット4	ビット3	ビット2	ビット1	ビット0
+$00	CTRLA	7～0	RUNSTDBY					RESSEL	FREERUN	ENABLE
+$01	CTRLB	7～0						SAMPNUM2～0		
+$02	CTRLC	7～0		SAMPCAP	REFSEL1,0			PRESC2～0		
+$03	CTRLD	7～0		INITDLY2～0		ASDV	SAMPDLY3～0			
+$04	CTRLE	7～0						WINCM2～0		
+$05	SAMPCTRL	7～0				SAMPLEN4～0				
+$06	MUXPOS	7～0				MUXPOS4～0				
+$07	予約									
+$08	COMMAND	7～0								STCONV
+$09	EVCTRL	7～0								STARTEI
+$0A	INTCTRL	7～0							WCMP	RESRDY
+$0B	INTFLAGS	7～0							WCMP	RESRDY
+$0C	DBGCTRL	7～0								DBGRUN
+$0D	TEMP	7～0	TEMP7～0							
+$0E～+$0F	予約									
+$10	RES	7～0	RES7～0							
+$11		15～8	RES15～8							
+$12	WINLT	7～0	WINLT7～0							
+$13		15～8	WINLT15～8							
+$14	WINHT	7～0	WINHT7～0							
+$15		15～8	WINHT15～8							
+$16	CALIB	7～0								DUTYCYC

1.4. 命名規則

本項はどの応用開発にも使われるデバイスのデータシートとヘッダ ファイルで見つけることができるレジスタとビットの命名規則を記述します。

1.4.1. レジスタ命名規則

レジスタは制御、状態、データのレジスタに分けられ、レジスタの名前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。同じ単位部に複数の一般目的制御レジスタが存在する場合、それらは接尾辞文字によって識別されます。この場合、制御レジスタは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れら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。

この文書で示されるマイクロ コントローラについてはデータ バス幅が8ビットなので、より大きなレジスタはいくつかの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、そのレジスタ名に各々HとLを追加することによって上位と下位のバイトがアクセスされます。例えば、16ビットA/D結果レジスタはRESと名付けられます。2つのバイトはRESL(RES下位、レジスタの最下位バイト)とRESH(RES上位、レジスタの最上位バイト)と命名されます。同じ名前を持つ複数レジスタを識別する別の方法は番号接尾辞を追加することで、例えば、NVMCTRLのADDRレジスタはAVR DA系統に対して24ビット レジスタです。番号接尾辞を使ってアクセスすることができる3つのレジスタはADDR0、ADDR1、ADDR2です。

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、ADC結果レジスタに対する16ビット アクセスを実行するのにHやL接尾辞なしの名前RESを使うことができます。これは32ビット レジスタに対する場合もです。

加えて、SET/CLR接尾辞を含むレジスタはこれがハードウェアで実装されるため、読み-変更-書き操作を行うことなく、それらのレジスタのビットを設定(1)または解除(0)することを許します。これらのレジスタは対で現れます。CLRレジスタのビットへの論理'1'書き込みは両レジスタの対応するビットを解除(0)する一方で、SETレジスタのビットへの論理'1'書き込みは両レジスタの対応するビットを設定(1)します。例えば、PORT単位部に於いて、データ方向設定(DIRSET)レジスタのビットへの論理'1'書き込みはデータ方向(DIR)とデータ方向解除(DIRCLR)のレジスタの対応するビットも設定(1)します。これらのレジスタは読み込み時に同じ値を返します。(SET/CLRの)両レジスタが同時に書かれた場合、CLRレジスタへの書き込みが優先権を取ります(**訳補**:純粋なプログラム上からは有り得ませんが、ハードウェアでの自動操作との競合としては有り得ます)。

1.4.2. ビットとビット領域の命名規則

以降の殆どの例はA/D変換器(ADC:Analog-to-Digital Converter)単位部に対して提供されます。

レジスタのビットは個別機能または共同機能を持つビット領域の一部を持ち得ます。個別ビットは単位部を許可するビット、例えば、CTRLAレジスタでのADC単位部のENABLEビットで有り得ます。ビット領域は2つ以上のビットから成り得、それらが属する単位部の具体的な構成設定を共同で選びます。ビット領域は2^n選択までを提供し、ここでのnはビット領域のビット数です。

ENABLEビットの位置は**図1-4.**で示され、PRESCビット領域の位置は**図1-5.**で示されます。

1.4.2.1. ビット命名規則

レジスタ構成図で見つかるビット名はビットが構成設定する機能を記述する完全なビット名の略語です。例えば、RUNSTDBYビットはスタンバイ休止動作で動くのを周辺機能に許可することを許します。ENABLEビットを('1'に)構成設定することによってADCを許可することができます。

図1-4. ADC制御Aレジスタに対するビット命名規則(1)

名称 : CTRLA
変位 : +$00
リセット : $00
特質 : -

ビット	7	6	5	4	3	2	1	0
	RUNSTDBY					RESSEL	FREERUN	ENABLE
アクセス種別	R/W	R	R	R	R	R/W	R/W	R/W
リセット値	0	0	0	0	0	0	0	0

1.4.2.2. ビット領域命名規則

同じビット領域内のビットはそのビット領域に相対するビット位置の数値を付加される数値接尾辞を使って参照することができます。例えば、ADC制御Cレジスタの前置分周器ビット領域の最上位ビット(MSb)はPRESC2です。この命名規則はデータシートで指定されませんが、この文書の後ろで記述され、ビット領域の個別レジスタ ビットを扱うためにヘッダ ファイルで使われます。

図1-5. 制御CレジスタでのPRESC構成設定ビット領域(1)

名称 : CTRLC
変位 : +$02
リセット : $00
特質 : -

ビット	7	6	5	4	3	2	1	0
		SAMPCAP	REFSEL1,0			PRESC2~0		
アクセス種別	R	R/W	R/W	R/W	R	R/W	R/W	R/W
リセット値	0	0	0	0	0	0	0	0

PRESCビット領域はビット領域の各値に対して**図1-6.**で示される選択を提供します。

図1-6. ADCクロック信号周波数用前置分周器構成設定(1)

● **ビット2~0 - PRESC2~0 : 前置分周器** (Prescaler)

これらのビットは周辺機能クロック(CLK_PER)からADCクロック(CLK_ADC)への整数分周比を定義します。

値	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
名称	DIV2	DIV4	DIV8	DIV16	DIV32	DIV64	DIV128	DIV256
説明	CLK_PER/2	CLK_PER/4	CLK_PER/8	CLK_PER/16	CLK_PER/32	CLK_PER/64	CLK_PER/128	CLK_PER/256

1.5. 構成設定変更保護(CCP)レジスタ

CCPレジスタは予期せぬ実行からのフラッシュ メモリ自己プログラミングだけでなく、予期せぬ変更からのシステムで重要なI/Oレジスタを保護するのにも使われます。CCP下のレジスタへの書き込みはCPU単位部の一部であるCCPレジスタへの指定識票/鍵書き込み後にだけ可能です。識票の値はデバイスのデータシートの「構成設定変更保護」項で見つけることができます。

保護されたレジスタには次のように各々にそれの鍵がある2つの形式があります。

- 保護されたI/Oレジスタ (鍵/識票はIOREGです。)
- 保護された自己プログラミング (鍵/識票はSPMです。)

構成設定変更保護下のレジスタのいくつかが下表で一覧にされます。

レジスタ名		鍵	機能
CLKCTRL.MCLKCTRLA	クロック制御器 – 主クロック制御A	IOREG	主クロック用クロック元選択とクロック出力を許可
CLKCTRL.MCLKLOCK	クロック制御器 – 主クロック施錠	IOREG	主クロック制御レジスタの施錠を許可
RSTCTRL.SWRR	リセット制御器 – ソフトウェア リセット	IOREG	デバイスへのソフトウェア リセットの印加を許可
CPUINT.CTRLAのIVSEL	CPU割り込み制御器 – CTRLA レジスタの割り込みベクタ選択ビット	IOREG	フラッシュ メモリの応用領域の先頭またはフラッシュ メモリのブート領域の先頭への割り込みベクタ配置を許可
NVMCTRL.CTRLA	不揮発性メモリ制御器 – CTRLAレジスタ	SPM	次の指令の1つの発行を許可 • メモリへのページ緩衝部書き込み • ページ消去 • ページの消去と書き込み • ページ緩衝部消去、など

保護されたI/Oレジスタやビットの変更、または保護された命令の実行はCPUがCCPレジスタへそれらの識票の1つを書いた後にだけアクセス可能です。識票はCCP(CPU.CCP)レジスタの記述で一覧にされます。

コード例は「3.5. 構成設定変更保護(CCP)レジスタへの書き込み」で提供されます。

1.6. ヒューズ

ヒューズはデバイスの構成設定を保持し、不揮発性メモリの一部です。それらはデバイスの始動から利用可能です。それらの値はチップ消去を通しても維持されます。ヒューズはCPUまたはプログラミング インターフェース(例えば、UPDI)によって読むことができますが、書き込み/デバッグ インターフェースを通してだけプログラム(設定)または解除することができます。ヒューズに格納された構成設定値はヒューズ値がCPUによって使えるように始動手順の最後でそれら各々の目的レジスタに複写されます。

ヒューズ概要表はデバイスのデータシートの「メモリ」章の「ヒューズ(FUSE)」項で見つけることができます。ATmega4808/4809データシートから抽出した例が下で示されます。

図1-7. FUSEレジスタ概要

変位	略称	ビット位置	ビット7	ビット6	ビット5	ビット4	ビット3	ビット2	ビット1	ビット0
+$00	WDTCFG	7～0	WINDOW3~0				PERIOD3~0			
+$01	BODCFG	7～0	LVL2~0			SAMPFREQ	ACTIVE1,0		SLEEP1,0	
+$02	OSCCFG	7～0	OSCLOCK						FREQSEL1,0	
+$03	予約									
+$04	予約									
+$05	SYSCFG0	7～0	CRCSRC1,0				RSTPINCFG			EESAVE
+$06	SYSCFG1	7～0						SUT2~0		
+$07	APPEND	7～0	APPEND7~0							
+$08	BOOTEND	7～0	BOOTEND7~0							
+$09	予約									
+$0A	LOCKBIT	7～0	LOCKBIT7~0							

コード例は「3.6. ヒューズの構成設定」で示されます。

2. ヘッダ ファイルでの単位部表現

各AVRデバイスに対して専用のヘッダ ファイルが利用可能です。目的対象デバイスは(使うIDE(MPLAB® X、Atmel Studio、またはIAR EW AVR)の何れかに対して)プロジェクト設定で指定されることが必要で、ヘッダ ファイルは作成したプロジェクトの主(main)ファイルに自動的にインクルードされます。ヘッダ ファイルは下のコードで示されるようにインクルードされます。

```
#include <avr/io.h>
```

必要とされるレジスタと構造体の全ての定義はヘッダ ファイルで見つけることができます。レジスタのアドレスを使う代わりに、デバイス固有ヘッダ ファイルで既に定義されたマクロと構造体の定義を使うことができます。

これは同じ単位部を含み、その単位部に対するヘッダ ファイル定義が同じデバイスで有用です。

2.1. メモリ内の単位部位置

与えられた周辺機能単位部に対する全てのレジスタは1つの連続メモリ部に配置されます。違う単位部に属すレジスタは混合されず、C構造体で全ての周辺機能単位部を構成することを可能にし、実体マクロは下のコードで示されるように定義されます。全ての周辺機能の定義はそのAVRデバイスに対して利用可能なデバイス ヘッダ ファイルで見つかります。単位部用のアドレスは利用可能な殆どのCコンパイラに適合させるためにANSI Cで指定されます。

```
#define PORTMUX         (*(PORTMUX_t *) 0x0200)       /* ポート多重器 */
#define PORTA              (*(PORT_t *) 0x0400)       /* 入出力ポート */
#define PORTB              (*(PORT_t *) 0x0420)       /* 入出力ポート */
#define PORTC              (*(PORT_t *) 0x0440)       /* 入出力ポート */
```

単位部実体定義はメモリ内の絶対アドレスへの間接参照されたポインタを使い、単位部実体基準アドレスと一致します。単位部ポインタはヘッダ ファイルで定義され、従って、ソース コードでそれらの定義を追加することは不要です。

例えば、PORTA単位部の基準アドレスは0x0400です。それのレジスタと予約ビットの全てのを持つ単位部は0x0400～0x0420で利用可能なメモリ空間を持ち、これは10進数で32バイトを意味します。従って、PORT_tはそれのレジスタ(または予約されたビット)全てに対して割り当てられた32バイトを含みます。

2.2. 単位部構造体

2.2.1. 例 – ADC

ADC_t構造体型は下のコードで示されるようにヘッダ ファイルで定義されます。それらがメモリで構成されるようにデータシートで指定された順で単位部のレジスタ全てを含みます。

```
/* A/D変換器 */
typedef struct ADC_struct
{
    register8_t CTRLA;          /* 制御A */
    register8_t CTRLB;          /* 制御B */
    register8_t CTRLC;          /* 制御C */
    register8_t CTRLD;          /* 制御D */
    register8_t CTRLE;          /* 制御E */
    register8_t SAMPCTRL;       /* 採取制御 */
    register8_t MUXPOS;         /* 多重器正選択 */
    register8_t reserved_1[1];
    register8_t COMMAND;        /* 指令 */
    register8_t EVCTRL;         /* 事象制御 */
    register8_t INTCTRL;        /* 割り込み制御 */
    register8_t INTFLAGS;       /* 割り込み要求フラグ */
    register8_t DBGCTRL;        /* デバッグ制御 */
    register8_t TEMP;           /* 一時データ */
    register8_t reserved_2[2];
    _WORDREGISTER(RES);         /* ADC累積器結果 */
    _WORDREGISTER(WINLT);       /* 窓比較器下側閾値 */
    _WORDREGISTER(WINHT);       /* 窓比較器上側閾値 */
    register8_t CALIB;          /* 校正 */
    register8_t reserved_3[1];
} ADC_t;
```

単位部の実体に対するマクロは下のコードで示されるように、その後に構造体型を使ってヘッダ ファイルで定義されます。

```
#define ADC0 (*(ADC_t *) 0x0600)          /* A/D変換器 */
```

従って、特定の単位部レジスタ、例えば、CTRLAレジスタはADC0.CTRLAとしてアドレス指定されます。

2.2.2. 例 – PORT

PORT_t構造体型は下のコードで示されるようにヘッダ ファイルで定義されます。

```
/* 入出力ポート */
typedef struct PORT_struct
{
    register8_t DIR;            /* データ方向 */
    register8_t DIRSET;         /* データ方向設定 */
    register8_t DIRCLR;         /* データ方向解除 */
    register8_t DIRTGL;         /* データ方向切り替え */
```

```
    register8_t OUT;            /* 出力値 */
    register8_t OUTSET;         /* 出力値設定 */
    register8_t OUTCLR;         /* 出力値解除 */
    register8_t OUTTGL;         /* 出力値切り替え */
    register8_t IN;             /* 入力値 */
    register8_t INTFLAGS;       /* 割り込み要求ﾌﾗｸﾞ */
    register8_t reserved_1[6];
    register8_t PIN0CTRL;       /* ﾋﾟﾝ0制御 */
    register8_t PIN1CTRL;       /* ﾋﾟﾝ1制御 */
    register8_t PIN2CTRL;       /* ﾋﾟﾝ2制御 */
    register8_t PIN3CTRL;       /* ﾋﾟﾝ3制御 */
    register8_t PIN4CTRL;       /* ﾋﾟﾝ4制御 */
    register8_t PIN5CTRL;       /* ﾋﾟﾝ5制御 */
    register8_t PIN6CTRL;       /* ﾋﾟﾝ6制御 */
    register8_t PIN7CTRL;       /* ﾋﾟﾝ7制御 */
    register8_t reserved_2[8];
} PORT_t;
```

2.2.3. 単位部構造体での複数ﾊﾞｲﾄ ﾚｼﾞｽﾀ

いくつかのﾚｼﾞｽﾀは16ﾋﾞｯﾄまたは32ﾋﾞｯﾄの値を表すために他のﾚｼﾞｽﾀと連結して使われます。例えば、ATmega4809ﾃﾞﾊﾞｲｽに対してADC結果(RES)、窓比較器下側閾値(WINLT)、窓比較器上側閾値(WINHT)は16ﾋﾞｯﾄ ﾚｼﾞｽﾀで、_WORDREGISTERﾏｸﾛを使って宣言されます。このﾏｸﾛは32ﾋﾞｯﾄ ﾚｼﾞｽﾀに使われる_DWORDREGISTERﾏｸﾛと共に下の一覧で示されます。これらのﾏｸﾛはﾍｯﾀﾞ ﾌｧｲﾙで既に定義されています。

_WORDREGISTERﾏｸﾛはLまたはHの接尾辞追加によってそれの下位側と上位側をｱｸｾｽするためのﾚｼﾞｽﾀ名拡張に使われます。_DWORDREGISTERﾏｸﾛも数値接尾辞追加によって多数ﾊﾞｲﾄ ﾚｼﾞｽﾀの全ﾊﾞｲﾄへのｱｸｾｽを提供します。_WORDREGISTERと_DWORDREGISTERの両定義は下のｺｰﾄﾞで示されます。

```
#ifdef _WORDREGISTER
#undef _WORDREGISTER
#endif
#define _WORDREGISTER(regname) ¥
    __extension__ union ¥
    { ¥
        register16_t regname; ¥
        struct ¥
        { ¥
            register8_t regname ## L; ¥
            register8_t regname ## H; ¥
        }; ¥
    }
#ifdef _DWORDREGISTER
#undef _DWORDREGISTER
#endif
#define _DWORDREGISTER(regname)  ¥
    __extension__ union ¥
    { ¥
        register32_t regname; ¥
        struct ¥
        { ¥
            register8_t regname ## 0; ¥
            register8_t regname ## 1; ¥
            register8_t regname ## 2; ¥
            register8_t regname ## 3; ¥
        }; ¥
    }
```

2.3. ビット遮蔽、ビット群遮蔽、群構成設定遮蔽

単位部の構造体構成を使ってどのレジスタもアクセスすることができます。ADCはデータシートでの単位部の説明特定部分で見つかる手順を使って完全に構成することができます。例えば、ADCを初期化するのに推奨される手順は「ADC – A/D変換器」⇒「機能的な説明」⇒「初期化」で示されます。レジスタのビットはヘッダ ファイルで定義されるビット遮蔽またはビット位置遮蔽を使って操作することができます。予め定義された遮蔽はその場合にそれらがビット遮蔽と呼ばれる個別ビット、またはその場合にそれらがビット群遮蔽または群遮蔽と呼ばれるビット群(ビット領域)に関連するどちらかです。例えば、ADC0単位部はビット遮蔽を使って変換周回を開始するように構成設定して許可することができます。

2.3.1. ビット遮蔽とビット位置遮蔽

ビット遮蔽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。ビット群遮蔽は主にビット領域で複数ビットを解除(0)する時に使われます。例えば、ADC0のCTRLDレジスタのビット領域、ビット名、ビット位置、ビット遮蔽は**表2-1.**で示されます。

表2-1. ビット領域、ビット名、ビット位置、ビット遮蔽

ビット領域	INITDLY2～0			–	SAMPDLY3～0			
ビット名	INITDLY2	INITDLY1	INITDLY0	EVDLY	SAMPDLY3	SAMPDLY2	SAMPDLY1	SAMPDLY0
ビット位置	7	6	5	4	3	2	1	0
ビット遮蔽	$80	$40	$20	$10	$08	$04	$02	$01

コンパイラに対してそれを扱うためにビット名が固有なことを必要とするため、全てのビットはそれらが属す単位部型を前に付けられます。多くの場合、単位部型名は略語にされます。タイマ/カウンタA型に関連する全てのビット定義について、ビット名はTCA_を前付けされます。

ビット遮蔽とビット位置を識別するため、接尾辞も付加されます。ビット遮蔽について接尾辞は_bmで、ビット位置について_bpです。従って、INITDLY0ビットに対するビット遮蔽の名前はADC_INITDLY0_bmで、ビット位置の名前はADC_INITDLY0_bpです。加えて、ヘッダ ファイルは群位置に対する定義も提供します。群位置の接尾辞は_gpで、例えば、INITDLY群位置遮蔽の名前はADC_INITDLY0_gpです。下のコードはそれらがデバイスのヘッダ ファイルで利用可能な時のINITDLYのビット遮蔽、ビット位置、群位置の定義を示します。

```
#define ADC_INITDLY_gp   5              /* 初期化遅延選択群位置 */
#define ADC_INITDLY0_bm  (1<<5)         /* 初期化遅延選択ビット0遮蔽 */
#define ADC_INITDLY0_bp  5              /* 初期化遅延選択ビット0位置 */
#define ADC_INITDLY1_bm  (1<<6)         /* 初期化遅延選択ビット1遮蔽 */
#define ADC_INITDLY1_bp  6              /* 初期化遅延選択ビット1位置 */
#define ADC_INITDLY2_bm  (1<<7)         /* 初期化遅延選択ビット2遮蔽 */
#define ADC_INITDLY2_bp  7              /* 初期化遅延選択ビット2位置 */
```

INITDLY0ビット遮蔽の命名規則例が**図2-1.**で示されます。

図2-1. ビット遮蔽の命名規則

2.3.2. ビット領域遮蔽 (群遮蔽)

ビットの領域によって構成設定される計時器のクロック選択、変換器の前置分周器選択、構成設定可能な注文論理回路の濾波器選択のような多くの機能はビット領域またはビット群として参照されます。群のビットの値が特定構成設定を選びます。

ビット領域を構成設定するための遮蔽はビット群遮蔽または群構成設定遮蔽として参照されます。

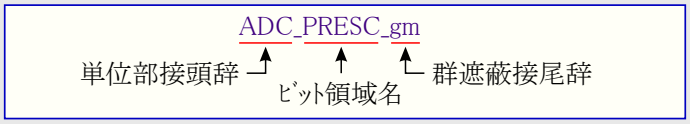
ビット領域のビットを変更する時は望む構成設定に対するビットを設定(1)するだけでは不充分で、新しい値を割り当てる前に古い構成設定でそれらのビットを解除(0)することも必要とされます。これを容易にするため、群遮蔽が定義されます。群遮蔽はビット領域のビットと同じ名前を使い、_gmを後付けされます。

下のコードはADC前置分周器群遮蔽がヘッダ ファイルでどう定義されるかを示します。

```
#define ADC_PRESC_gm     0x07           /* クロック前置分周器群遮蔽 */
```

命名規則例が**図2-2.**で示されます。

図2-2. 群遮蔽の命名規則



ビット群遮蔽は主に新しい値を書く前にそのビット領域の旧構成設定を解除することを意図されます。下のコードはこれがどう行われ得るかを示します。このコードはADC0単位部のCTRLCレジスタのPRESCビット領域を解除します。この構文は構成設定を設定(1)しません。前置分周器構成設定ビットの全てを解除(0)するだけです。具体的な構成設定をリセットするために全てのビット遮蔽を使う必要がなく、この操作には群遮蔽だけが必要なため、これは利点です。群遮蔽は代表的に特定の構成設定を解除するのに群構成設定遮蔽と共に使われます。

```
ADC0.CTRLC &= ~(ADC_PRESC_gm);              /* 群遮蔽を使って前置分周器ビット領域を解錠 */
```

2.3.3. 群構成設定遮蔽と列挙子

これはビット領域を望む構成設定に設定する時に使われるビット様式を調べるためにデータシートを調査するのに度々必要とされます。これはコードを読んだりデバッグしたりする時にも適用されます。可読性を増してビット領域で不正にビットを設定する可能性を最小とするため、ヘッダ ファイルで様々な群構成設定遮蔽が定義されます。各群構成設定遮蔽は特定の群遮蔽に対する構成設定を選びます。

群構成設定遮蔽の名前は単位部型、ビット領域名、構成設定の説明、それとこれが群構成設定であることを示す_gc接尾辞の連結です。ADC前置分周器の例が**図2-3.**で示されます。

図2-3. 群構成設定遮蔽の命名規則

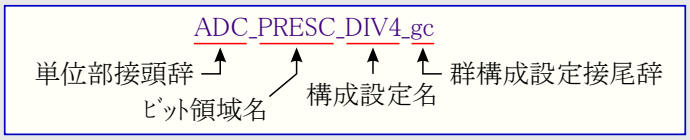


図2-3.で示された群構成設定は周辺機能クロック(CLK_PER)からADCクロックへ4分周の係数で前置分周器を設定します。

ADCの前置分周器ビット領域は分周係数を定義する3ビットから成ります。可能な構成設定名はDIV2、DIV4、DIV8、DIV16、DIV32、DIV64、DIV128、DIV256です。これらの名前は特定遮蔽がどの構成設定を選ぶかを理解するのに非常に小さな努力しか必要としないため、コードを書いて操作するのを非常に容易にします。**表2-2.**はこのビット領域に対して利用可能な構成設定を示します。

表2-2. PRESCビットと対応するビット群構成設定

PRESC2	PRESC1	PRESC0	分周係数	群構成設定遮蔽
0	0	0	CLK_PERを2分周	ADC_PRESC_DIV2_gc
0	0	1	CLK_PERを4分周	ADC_PRESC_DIV4_gc
0	1	0	CLK_PERを8分周	ADC_PRESC_DIV8_gc
0	1	1	CLK_PERを16分周	ADC_PRESC_DIV16_gc
1	0	0	CLK_PERを32分周	ADC_PRESC_DIV32_gc
1	0	1	CLK_PERを64分周	ADC_PRESC_DIV64_gc
1	1	0	CLK_PERを128分周	ADC_PRESC_DIV128_gc
1	1	1	CLK_PERを256分周	ADC_PRESC_DIV256_gc

ビット領域を新しい構成設定に変更するには古い構成設定が先に解除されるのを保証するため、代表的にビット群遮蔽と共に群構成設定が使われます。

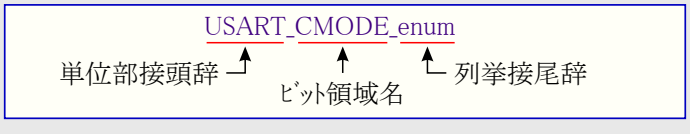
ビット遮蔽や群遮蔽と異なり、ビット群構成設定遮蔽はCの列挙を使って定義されます。各ビット領域に対して1つの列挙が定義されます。USARTのCMODEビット領域用の列挙が下のコードで示されます。

```
typedef enum USART_CMODE_enum
{
    USART_CMODE_ASYNCHRONOUS_gc = (0x00<<6),  /* 非同期動作 */
    USART_CMODE_SYNCHRONOUS_gc = (0x01<<6),   /* 同期動作 */
    USART_CMODE_IRCOM_gc = (0x02<<6),         /* 赤外線通信 */
    USART_CMODE_MSPI_gc = (0x03<<6),          /* 主装置SPI動作 */
} USART_CMODE_t;
```

列挙の名前は単位部型(USART)、ビット領域(CMODE)、接尾辞(_enum)の連結です。

命名規則が**図2-4.**で示されます。

図2-4. 列挙の命名規則



列挙定数の各々はそれ自体が使われる時に通常の定数とそっくりに振る舞います。けれども、列挙型定義の使用は新しいデータ型を作成する利点を持ちます。この例に対するデータ型の名前はUSART_CMODE_tです。USART_CMODE_t変数は整数として直接使うことができますが、列挙型への整数割り当てがコンパイラ警告を起動するでしょう。これはプログラム作成者の利益に使うことができます。

例えば、USART単位部に対する通信動作を設定する関数が整数型(例えば、unsigned char)として通信動作を受け入れる場合、正当または不正な何れの値も関数に渡され得ます。関数が代わりにUSART_CMODE_t型のパラメータを受け入れる場合、USART_CMODE_t列挙内の予め定義された4つの定数だけが関数に渡され得ます。他の何かを渡すと、コンパイラ警告になります。

注: コード一覧内の定数が既にそれらのビット位置に移動されているのに注意することが重要です。列挙定数はレジスタに書かれる実際の値で、追加の移動は必要とされません。

3. AVR®用素のCコード書き

以降の項はAVR用のCコードの書き方に集中します。例は異なるAVRデバイス間で可読で可搬なコードの作成法を記述します。例は確認と保守が容易なコードの書き方の指針としても使うことができます。

マイクロ コントローラの単位部はメモリ空間に於いて専用且つ連続した塊で配置され、カプセル化された単位として見ることができます。単位部はC構造体でカプセル化され、その中に全ての単位部レジスタが含まれます。

本文書はAVRヘッダ ファイルに適合する命名規則とレジスタ アクセス法を紹介し、Cコードで書かれるコードに対する可動性と可搬性を提供します。

3.1. レジスタのビットの設定、解除、読み込み

レジスタのビットの設定(1)と解除(0)は組み込みプログラム書きで使われる基本的な操作でどの応用も基づく技法を表します。

読み-変更-書き操作は非分断操作の類です。これらはメモリ位置を読み、同時にそこへ完全に新しい値または以前の値の一部のどちらかで新しい値を書きます。

それが広い適応性を持つため、ビットの値の読み込みは主に条件式(例えば、if文)と繰り返し式(例えば、while文)での条件として使われます。この技法の一般的な使用事例は割り込み要求フラグのポーリングで、これはビットの値を読んでビットが設定(1)/解除(0)の場合に一式の命令の実行を意味します。

注: 2進数演算の更なる詳細についてはビット単位演算子(Bitwise Operators)を参照してください。

3.1.1. ビット遮蔽を使うレジスタ ビットの設定、解除、読み込み

レジスタのビットはヘッダ ファイルによって提供されるビット遮蔽を使って設定(1)、解除(0)、読み込みができます。

構造体宣言を使うレジスタ アクセス

レジスタの特定ビットを設定(1)するための推奨されるコード書き様式はヘッダ ファイルで宣言された構造体実体とビット遮蔽マクロ定義を使うことです。ビット遮蔽とで2進数OR(論理和)演算子を使うことはそのレジスタ内で作られた他のビット設定がその操作によって影響を及ぼされず、同じに留まることを保証します。

```
ADC0.CTRLA |= ADC_ENABLE_bm;                    /* ADC周辺機能を許可 */
```

レジスタのビットを解除(0)するため、ビット遮蔽の反転された値とレジスタ間に2進数AND(論理積)演算子が適用されます。この操作は他のビット設定を無変化にも保ちます。

```
ADC0.CTRLA &= ~ADC_ENABLE_bm;                   /* ADC周辺機能を禁止 */
```

ADC_ENABLE_bmビット遮蔽は下で示されるようにヘッダ ファイルで定義されます。

```
#define ADC_ENABLE_bm  0x01                     /* ADC許可ビット遮蔽 */
```

ビットの読み込みは様々な応用で必要です。例えば、ADCのRESRDYフラグはADC結果の準備が整うと、ハードウェアによって設定(1)されます。下のコード一覧はこのビットがて設定(1)されているかを調べてその場合に何かの命令を実行する方法を示します。

```
if(ADC0.INTFLAGS & ADC_RESRDY_bm)               /* ADC結果が準備可か調査 */
{
    /* ここに何かの命令を挿入してください。 */
}
```

ビットが解除(0)かを調べて命令を実行する、またはそれが解除(0)に留まる間待つには次のコードを使うことができます。この場合、ADC結果の準備が整うまでこの繰り返し内側の命令が実行されます。

```
while(!(ADC0.INTFLAGS & ADC_RESRDY_bm))
{
    /* ここに何かの命令を挿入してください。 */
}
```

マクロ定義を使うレジスタ アクセス

代わりに、レジスタはヘッダ ファイルのマクロ定義を使ってもアクセスすることができます。下の例はこれらの定義を使ってビットを設定(1)する方法を示します。

```
ADC0_CTRLA |= ADC_ENABLE_bm;                    /* ADC周辺機能を許可 */
```

マクロ定義を使ってビットを解除(0)するのに次のコード例を使うことができます。

```
ADC0_CTRLA &= ~ADC_ENABLE_bm;                /* ADC周辺機能を禁止 */
```

下のコード一覧はこのビットが設定(1)されているかを調べてその場合に何かの命令を実行する方法を示します。

```
if(ADC0_INTFLAGS & ADC_RESRDY_bm)            /* ADC結果が準備可か調査 */
{
    /* ここに何かの命令を挿入してください。 */
}
```

ビットが解除(0)かを調べ、それが解除(0)に留まる間命令を実行するのに次のコードを使うことができます。この場合、ADC結果の準備が整うまでこの繰り返し内側の命令が実行されます。

```
while(!(ADC0_INTFLAGS & ADC_RESRDY_bm))
{
    /* ここに何かの命令を挿入してください。 */
}
```

3.1.2. ビット位置を使うレジスタの設定、解除、読み込み

レジスタのビットを設定(1)、解除(0)、読み込む代替法はビット位置によってで、これもデバイスのヘッダ ファイルによって提供されます。

構造体宣言を使うレジスタ アクセス

構造体宣言を使ってレジスタをアクセスし、ビット位置マクロを使ってビットを設定(1)するのに次のコード例を使うことができます。

```
ADC0.CTRLA |= (1 << ADC_ENABLE_bp);          /* ADC周辺機能を許可 */
```

ビット位置マクロを使ってビットを解除(0)するのに次の例が提供されます。

```
ADC0.CTRLA &= ~(1 << ADC_ENABLE_bp);         /* ADC周辺機能を禁止 */
```

ビットが設定(1)されているかを調べるのに次のコードを使うことができます。

```
if(ADC0.INTFLAGS & (1 << ADC_RESRDY_bp))  /* ADC結果が準備可か調査 */
{
    /* ここに何かの命令を挿入してください。 */
}
```

ビットが解除(0)かを調べ、それが解除(0)に留まる間命令を実行するのに次のコードを使うことができます。この場合、ADC結果の準備が整うまでこの繰り返し内側の命令が実行されます。

```
while(!(ADC0.INTFLAGS & (1 << ADC_RESRDY_bp)))
{
    /* ここに何かの命令を挿入してください。 */
}
```

マクロ定義を使うレジスタ アクセス

マクロ定義を使ってレジスタをアクセスしてビット位置マクロを使ってビットを設定(1)するのに次のコード例を使うことができます。

```
ADC0_CTRLA |= (1 << ADC_ENABLE_bp);          /* ADC周辺機能を許可 */
```

ビット位置マクロを使ってビットを解除(0)するのに次の例が提供されます。

```
ADC0_CTRLA &= ~(1 << ADC_ENABLE_bp);         /* ADC周辺機能を禁止 */
```

ビットが設定(1)されているかを調べるのに次のコードを使うことができます。

```
if(ADC0_INTFLAGS & (1 << ADC_RESRDY_bp))  /* ADC結果が準備可か調査 */
{
    /* ここに何かの命令を挿入してください。 */
}
```

ビットが解除(0)かを調べ、それが解除(0)に留まる間命令を実行するのに次のコードを使うことができます。この場合、ADC結果の準備が整うまでこの繰り返し内側の命令が実行されます。

```
while(!(ADC0_INTFLAGS & (1 << ADC_RESRDY_bp)))
{
    /* ここに何かの命令を挿入してください。 */
}
```

3.2. レジスタ初期化

レジスタを初期化するにはデバイスのデータシートを調査することによって望む構成設定を見つけなければなりません。そして、レジスタの値が望む構成設定に一致するようにレジスタのビットが設定(1)または解除(0)されなければなりません。

レジスタ初期化は殆どの場合レジスタが既知のリセット状態(既定)である時のリセット後にデバイスの初期化の一部として実行されます。

殆どのAVRのレジスタについて、全てのビットとビット領域に対するリセット値は'0'です。レジスタのリセット値はこの例に対して望まれるレジスタのビット構成設定と共に図3-1.で示されるように見つけることができます。

図3-1. ADC制御Aレジスタ – リセット値とビット設定

名称 : CTRLA
変位 : +$00
リセット : $00
特質 : –

ビット	7	6	5	4	3	2	1	0
	RUNSTDBY		COMVMODE	LEFTADJ	RESSEL1,0		FREERUN	ENABLE
アクセス種別	R/W	R	R/W	R/W	R/W	R/W	R/W	R/W
リセット値	0	0	0	0	0	0	0	0

● **ビット7 – RUNSTDBY : スタンバイ時走行** (Run in Standby)

このビットはスタンバイ中にADCが未だ動くかどうかを決めます。

値	0	1
説明	ADCはスタンバイ休止動作で動きません。進行中の変換はADCが休止動作に入る前に終わります。	ADCはスタンバイ休止動作で動きます。

● **ビット5 – CONVMODE : 変換動作形態** (Conversion Mode)

このビットはADCがシングル エンド動作または差動動作で動くかを定義します。

値	0	1
名称	SINGLEENDED	DIFF
説明	ADCは正入力だけが使われるシングル エンド動作で動きます。ADC結果は符号なし値として表されます。	ADCは正と負の両入力が使われる差動動作で動きます。ADC結果は符号付き値として表されます。

● **ビット4 – LEFTADJ : 結果左揃え** (Left Adjust Result)

このビットへの'1'書き込みがADC結果の左揃えを許可します。

● **ビット3,2 – RESSEL1,0 : 分解能選択** (Resolution Selection)

このビット領域はADC分解能を選びます。12ビットから10ビットへ分解能を変えると、変換時間は13.5 CLK_ADC周期から11.5 CLK_ADC周期に減らされます。

値	0 0	0 1	その他
説明	12ビット分解能	10ビット分解能	(予約)

● **ビット1 – FREERUN : 自由走行** (Free Running)

このビットへの'1'書き込みがADCに対して自由走行動作を許可します。最初の変換は指令(ADCn.COMMAND)レジスタの変換開始(STCONV)ビットへ'1'を書くことによって開始されます。

● **ビット0 – ENABLE : ADC許可** (ADC Enable)

値	0	1
説明	ADCは禁止されます。	ADCは許可されます。

レジスタのリセット値が$00の場合、ビット遮蔽またはビット位置で作業する時に読み–変更–書き操作が必要とされず、そのレジスタは単一行で構成設定することができます。

望む構成設定は例えば、以下で有り得ます。

- ADCは許可 – ENABLEビットが'1'
- ADCは差動動作で作動 – CONVMODEビットが'1'
- ADCは10ビット分解能で動作 – RESSELビット領域値が'00'(既定)

この構成設定は下で示されるように、2進数、16進数、10進数のどれかを使ってﾚｼﾞｽﾀに直接、結果の値を書くことによって実装することができます。

```
ADC0.CTRLA = 0b00100001;                /* 2進数 */
ADC0.CTRLA = 0x21;                      /* 16進数 */
ADC0.CTRLA = 33;                        /* 10進数 */
```

けれども、ｺｰﾄﾞの可読性(と潜在的な可搬性)を改善するため、ﾃﾞﾊﾞｲｽ定義を使うことが推奨され、これは間もなく来る部分で示されます。

注: AVRのﾚｼﾞｽﾀについては殆どﾋﾞｯﾄとﾋﾞｯﾄ領域に対するﾘｾｯﾄ値が'0'ですが、例外があります。例えば、USART0制御Cﾚｼﾞｽﾀはﾘｾｯﾄ値'1'のﾋﾞｯﾄ対を持ちます。この場合、ﾋﾞｯﾄのﾘｾｯﾄ値が通常'0'であると言う事実を信頼することなく、望む構成設定を明示的に設定しなければなりません。

図3-2. USART制御Cﾚｼﾞｽﾀ - ﾘｾｯﾄ値

名称 : CTRLC
変位 : +$07
ﾘｾｯﾄ : $03
特質 : -

このﾚｼﾞｽﾀ記述は主装置SPI動作を除く全動作に対して有効です。このﾚｼﾞｽﾀのUSART通信動作(CMODE)ﾋﾞｯﾄ領域が'MSPI'を書かれた時の正確な記述については「制御C(CTRLC) - 主装置SPI動作」ﾚｼﾞｽﾀをご覧ください。

ﾋﾞｯﾄ	7	6	5	4	3	2	1	0
	CMODE1,0		PMODE1,0		SBMODE	CHSIZE2~0		
ｱｸｾｽ種別	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ﾘｾｯﾄ値	0	0	0	0	0	0	1	1

USART0制御Cﾚｼﾞｽﾀは$03のﾘｾｯﾄ値を持ちます。この場合、CHSIZEﾋﾞｯﾄ領域の値を変更せずにそれらのﾋﾞｯﾄまたはﾋﾞｯﾄ領域の1つを初期化するには読み-変更-書き操作が必要とされます。このﾚｼﾞｽﾀは**図3-2.**で示されます。

3.2.1. ﾋﾞｯﾄ遮蔽と群構成設定遮蔽を使うﾚｼﾞｽﾀ初期化

本項はﾋﾞｯﾄ遮蔽と群構成設定遮蔽を使ってADCのCTRLAﾚｼﾞｽﾀを構成設定する推奨方法を提供します。

```
ADC0.CTRLA = ADC_ENABLE_bm              /* ADCを許可 */
           | ADC_CONVMODE_bm            /* 差動変換動作を選択 */
           | ADC_RESSEL_10BIT_gc;       /* 10ﾋﾞｯﾄ変換 */
```

割り当てのﾚｼﾞｽﾀ側でﾋﾞｯﾄ単位OR(|)が存在しないことに注意してください。殆どの場合、ﾃﾞﾊﾞｲｽと周辺機能のﾙｰﾁﾝはこの方法で書かれます。

ADC_RESSEL_enumは下で示される群構成設定遮蔽を含みます。

```
ADC_RESSEL_10BIT_gc = (0x01<<2),        /* 10ﾋﾞｯﾄ動作 */
```

⚠警告 上のﾚｼﾞｽﾀの初期化はCｺｰﾄﾞの単一行で行われなければなりません。次のように書くと、2行目の群構成設定は最初の行で設定(1)されたﾋﾞｯﾄを解除(0)します。

```
ADC0.CTRLA = ADC_ENABLE_bm;
ADC0.CTRLA = ADC_CONVMODE_bm;
ADC0.CTRLA = ADC_RESSEL_10BIT_gc;
```

3行のｺｰﾄﾞを使ってこのｺｰﾄﾞを書く正しい方法が下で示されます。

```
ADC0.CTRLA = ADC_ENABLE_bm;
ADC0.CTRLA |= ADC_CONVMODE_bm;
ADC0.CTRLA |= ADC_RESSEL_10BIT_gc;
```

注: ﾋﾞｯﾄ遮蔽は単一行のｺｰﾄﾞでだけﾋﾞｯﾄを設定(1)することができ、故にﾋﾞｯﾄを'0'に設定するのを必要とするどの構成設定も、それらがそれらのﾘｾｯﾄ値によって正しく構成設定されるため省略することができます。

3.2.2. ﾋﾞｯﾄ位置を使うﾚｼﾞｽﾀ初期化

上で示したのと同じ構成設定は次のようにﾋﾞｯﾄ位置ﾏｸﾛを使うことによって行うことができます。

```
ADC0.CTRLA = (1 << ADC_ENABLE_bp)       /* ADCを許可 */
           | (1 << ADC_CONVMODE_bp)     /* 差動変換動作を選択 */
           | (1 << ADC_RESSEL0_bp)      /* 10ﾋﾞｯﾄ変換 */
           | (0 << ADC_RESSEL1_bp);
```

注: (0 << ADC_RESSEL0_bp)行は単に可読性のために追加されていますが、取り去ることができます。

```
ADC0.CTRLA = (1 << ADC_ENABLE_bp)          /* ADCを許可 */
           | (1 << ADC_CONVMODE_bp)        /* 差動変換動作を選択 */
           | (1 << ADC_RESSEL0_bp)         /* 10ビット変換 */
```

ビット領域を構成設定するのに使うことができる他の位置マクロは群位置遮蔽です。これらは下で示されるように使うことができます。望む構成設定値はビット領域位置(群位置)で移動されなければなりません。

```
ADC0.CTRLA = (1 << ADC_ENABLE_bp)          /* ADCを許可 */
           | (1 << ADC_CONVMODE_bp)        /* 差動変換動作を選択 */
           | (0x01 << ADC_RESSEL_gp);      /* 10ビット変換 */
```

3.3. レジスタのビット領域構成設定変更

本項は様々なヘッダ ファイル定義を使ってレジスタのビット領域を更新する時の考慮を網羅します。RXMODEビット領域が例として使われ、更新は下で示される初期化に例えて行われます。

```
USART0.CTRLB = USART_RXEN_bm               /* 受信部を許可 */
             | USART_TXEN_bm               /* 送信部を許可 */
             | USART_RXMODE_LINAUTO_gc;    /* LIN制限付き自動ボー動作 */
```

以降の下位項は受信部動作(RXMODE)構成設定を標準自動ボー(GENAUTO)動作に変更するコード例を提供します。それらがデバイスのデータシートであるような、このビット領域に対して利用可能な構成設定は下表で示されます。

図3-3. USART0受信部動作構成設定

値	0 0	0 1	1 0	1 1
名称	NORMAL	CLK2X	GENAUTO	LINAUTO
説明	標準USART動作、標準速	標準USART動作、倍速	標準自動ボーレート動作	LIN制限自動ボーレート動作

3.3.1. 群と分構成設定遮蔽を使うレジスタのビット領域構成設定の変更

レジスタのビット領域だけを更新する時は読み-変更-書き操作が使われなければなりません。従って、レジスタのビット領域の構成設定を変更するには先に旧構成設定を解除してその後に新しいものを設定することが推奨されます。

ビット領域に構成設定を設定するにはビット群構成設定遮蔽を使うことが推奨されます。

ビット領域を新しい構成設定に変更する1つの方法が下のコード一覧で示されます。望む構成設定が得られるのを保証するため、先に群遮蔽を使って旧構成設定を解除しなければなりません。

```
/* ビット群構成設定を変更 */
USART0.CTRLB &= (~USART_RXMODE_gm);        /* 旧構成設定を解除 */
USART0.CTRLB |= USART_RXMODE_GENAUTO_gc;   /* 群構成設定遮蔽を使って新しい構成設定を設定 */
```

注: 最初の行(上の2行目)はUSART0のRXMODEを短時間、指定状態('00')に置きます。

群遮蔽マクロは下で示されるようにヘッダ ファイルで定義されます。

```
#define USART_RXMODE_gm 0x06               /* 受信部動作群遮蔽 */
```

USART_RXMODE_enum列挙は下で示される群構成設定遮蔽を含みます。

```
USART_RXMODE_GENAUTO_gc = (0x02<<1),       /* 標準自動ボー動作 */
```

⚠警告 例え1つの行がレジスタを解除し、別の1つの行が望む構成設定を設定する、2つの独立コード行にコードを分けることがより簡単に思われるかもしれませんが、下のコード一覧で示されるように、これを行うには単一行を使うことが推奨されます。

```
/* ビット群構成設定を変更 */
USART0.CTRLB = (USART0.CTRLB & ~USART_RXMODE_gm) | USART_RXMODE_GENAUTO_gc;
```

これらの手順はマイクロ コントローラを予期せぬ状態に置くことを避けるために単一行で実装されなければなりません。

CTRLBレジスタはこのように宣言されます。

```
register8_t CTRLB;
```

register8_t型はvolatile宣言されたデータ型です。

```
typedef volatile uint8_t register8_t;
```

レジスタがvolatileとして定義されるため、2つの異なるコード行はCTRLBレジスタで読みと書きを1回の代わりに2回起動します。コードを非効率にするのに加えて、これは周辺機能を予期せぬ状態にも置き得ます。その理由は2行のコード間で割り込みが起動されると、状況が変えられ、レジスタが未定義状態に留まり得るからです。

3.3.2. ビット遮蔽を使うレジスタのビット領域構成設定の変更

下の例は新しい構成設定を設定するのにビット遮蔽を使ってレジスタのビット領域を更新する方法を示します。単一コード行で、現在の構成設定が解除され、新しい構成設定が設定されます。

```
USART0.CTRLB = (USART0.CTRLB & ~(USART_RXMODE0_bm | USART_RXMODE1_bm))
             | USART_RXMODE1_bm;          /* 標準自動ボー動作 */
```

3.3.3. ビット位置を使うレジスタのビット領域構成設定の変更

下の例は新しい構成設定を設定するのにビット位置を使ってレジスタのビット領域を更新する方法を示します。単一コード行で、現在の構成設定が解除され、新しい構成設定が設定されます。

```
USART0.CTRLB = (USART0.CTRLB & ~((1 << USART_RXMODE0_bp) | (1 << USART_RXMODE1_bp)))
             | (0 << USART_RXMODE0_bp)
             | (1 << USART_RXMODE1_bp);
```

注: (0 << USART_RXMODE0_bp)行は単に可読性のために追加されていますが、取り去ることができます。

```
USART0.CTRLB = (USART0.CTRLB & ~((1 << USART_RXMODE0_bp) | (1 << USART_RXMODE1_bp)))
             | (1 << USART_RXMODE1_bp);
```

3.4. ビット遮蔽と群構成設定遮蔽使用の利点

推奨されたコード書き様式を使う利点は次のとおりです。

- もっと読み易いコードが得られます。群構成設定遮蔽を使うことにより、設定されつつある構成設定が何かを確実に知るでしょう。構成設定名は遮蔽名に含まれています。
- もっと簡潔なコードが得られます。群遮蔽は各ビットにビット遮蔽を使う必要なしにビット領域内の全てのビットの解除を助けします。構成設定遮蔽はビット領域全体を構成設定するのに使うこともできます。
- コードは維持がより容易です。例えば、ビット領域構成設定を変更するには、ビット遮蔽またはビット位置を使って各ビット値を変更する代わりに群構成設定遮蔽名を変更しなければならないだけです。

3.5. 構成設定変更保護(CCP)レジスタへの書き込み

適切なCCP解錠手順に従わずに保護されたレジスタへ書く試みは保護されたレジスタを無変化のままにします。デバイスのデータシートで指定された手順は代表的にCPU.CCPレジスタに識票を書き、その後の4命令内で望む値を保護されたレジスタに書くことを必要とします。

CCP識票は下で示されるようにデバイスのヘッダ ファイルによって提供されます。

```
/* CCP識票選択 */
typedef enum CCP_enum
{
    CCP_SPM_gc = (0x9D<<0),              /* SPM命令保護 */
    CCP_IOREG_gc = (0xD8<<0),            /* I/Oレジスタ保護 */
} CCP_t;
```

次の例はIOREG識票を使ってCCP下のレジスタに書く方法を示します。主クロック前置分周器の分周値が16に設定されます。

```
CCP = CCP_IOREG_gc;                      /* 必要とされる識票をCCPに書き込み */
CLKCTRL.MCLKCTRLB = CLKCTRL_PDIV_16X_gc; /* 前置分周器を16分周に設定 */
```

次の例はSPM識票を使ってCCP下のレジスタに書く方法を示します。

```
CPU.CCP = CCP_SPM_gc;                    /* 必要とされる識票をCCPに書き込み */
NVMCTRL.CTRLA = NVMCTRL_CMD_FLPER_gc;    /* フラッシュ メモリ ページ消去許可 */
```

⚠警告 これらはCCP下のレジスタへ書く推奨方法ではありません。CCPによって保護されたレジスタへ書くには4命令内にソフトウェアが保護されたレジスタに望む値を書かなければなりません。タイミング要件を満たすため、CCPレジスタへの書き込みは多くの場合アセンブリ コードによって処理されます。

従って、保護されたI/Oレジスタ書き込みの推奨方法はccp_write_io関数を使うことによります。この関数を使うには次のヘッダ ファイルがインクルードされなければなりません。

```
#include <avr/cpufunc.h>                 /* 必要とされるヘッダ ファイル */
```

次のｺｰﾄﾞ例は前に示された例と同じ機能を提供しますが、ccp_write_io関数を使っています。

```
/* 前置分周器を16分周に設定 */
ccp_write_io((void *) & (CLKCTRL.MCLKCTRLB), (CLKCTRL_PDIV_16X_gc));
```

代わりに、CCPﾚｼﾞｽﾀに値を書くためにﾏｸﾛが定義されます。これはXMEGA® CCP機構を通して保護され、これはCCPに対して必要とされる時間手順を実装します。

注: CCPﾚｼﾞｽﾀがAVRﾃﾞﾊﾞｲｽのXMEGA系統で導入されたため、_PROTECTED_WRITEと_PROTECTED_WRITE_SPMのﾏｸﾛを含むﾍｯﾀﾞﾌｧｲﾙはxmega.hで、これが下で示されるようにｲﾝｸﾙｰﾄﾞされなければなりません。

```
#include <avr/xmega.h>                    /* 必要とされるﾍｯﾀﾞﾌｧｲﾙ */
```

これらのﾏｸﾛは下のｺｰﾄﾞ例で示されるように望むﾚｼﾞｽﾀへ書くのに使われなければなりません。

```
/* 32kHz内部超低電力発振器を選択 */
_PROTECTED_WRITE(CLKCTRL.MCLKCTRLA, CLKCTRL.MCLKCTRLA | CLKCTRL_CLKSEL_OSCULP32K_gc);
/* ﾍﾟｰｼﾞ書き込み指令 */
_PROTECTED_WRITE_SPM(NVMCTRL.CTRLA, NVMCTRL_CMD_PAGEWRITE_gc);
```

注: XC8ｺﾝﾊﾟｲﾗ(MPLAB X IDE)とGCC(ATmel Stuido 7)の両方がこれらのﾏｸﾛの使用を支援します。

3.6. ﾋｭｰｽﾞの構成設定

ﾋｭｰｽﾞは予め設定されていますが、ﾋｭｰｽﾞﾚｼﾞｽﾀは使用者によって変えることができます。ﾋｭｰｽﾞはそれをﾌﾟﾛｸﾞﾗﾑ(0)することによってだけ変えることができます。けれどもいくつかのﾋｭｰｽﾞ値はﾚｼﾞｽﾀに読み込まれ、それらの値を走行時の間に変更することができます。ﾋｭｰｽﾞは以降の項で記述されるように、Cｺｰﾄﾞを書くことによって構成設定することができます。

3.6.1. XC8構成設定ﾋﾞｯﾄを使うﾋｭｰｽﾞの構成設定

MPLAB X IDEでﾋｭｰｽﾞを構成設定するには構成設定ﾊﾟﾗﾒｰﾀを使うことができます。ｺﾝﾊﾟｲﾗと望むﾃﾞﾊﾞｲｽの構成設定ﾋﾞｯﾄについてのより多くの情報は**図3-4.**で示されるように、ｺﾝﾊﾟｲﾗのﾍﾙﾌﾟ(MPLAB X IDEﾌﾟﾛｼﾞｪｸﾄの計器盤(Dashboard)から青い疑問符(?)をｱｸｾｽすることによって見つけることができます。

図3-4. ｺﾝﾊﾟｲﾗのﾍﾙﾌﾟへのｱｸｾｽ

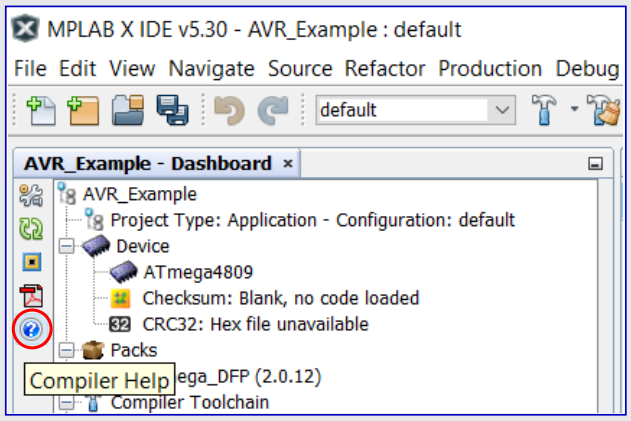
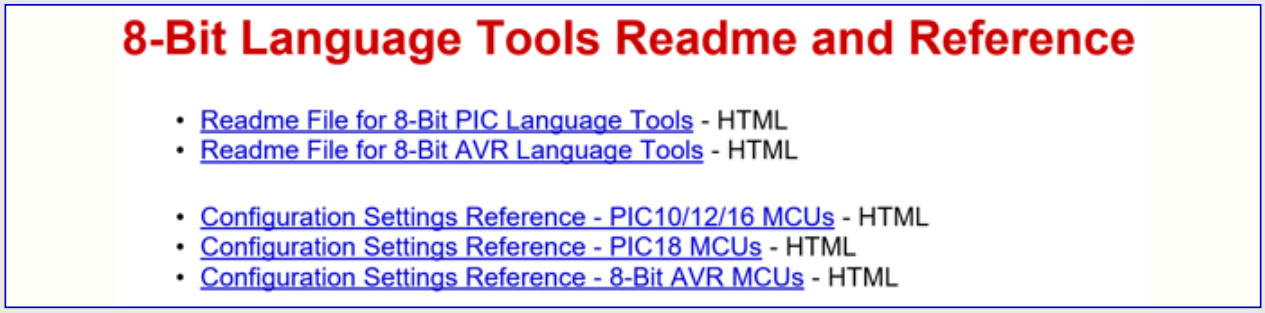


図3-5.で示されるように、支援される全てのﾃﾞﾊﾞｲｽに対する構成設定がConfiguration Settings Reference(構成設定参考文献)⇒8-bit AVR MCUsで示されます。

図3-5. ｺﾝﾊﾟｲﾗのHelp(ﾍﾙﾌﾟ)⇒8-bit Language Tools Readme and Reference(8ﾋﾞｯﾄ言語ﾂｰﾙ注意書きと参考文献)



構成設定指令(pragma)は下で示されるように使うことができます。

```
#pragma config <setting> = <named value | literal constant>
```

次の例はｳｫｯﾁﾄﾞｯｸﾞﾀｲﾏとCRCを禁止して構成設定指令を使って始動時間を8msに構成設定する方法を示します。

```
* ｳｫｯﾁﾄﾞｯｸﾞﾀｲﾏを禁止 */
#pragma config PERIOD = PERIOD_OFF
/* CRCを禁止してﾘｾｯﾄﾋﾟﾝの構成をGPIO(汎用入出力)動作に設定 */
#pragma config CRCSRC = CRCSRC_NOCRC, RSTPINCFG = RSTPINCFG_GPIO
/* 始動時間選択: 8ms */
#pragma config SUT = SUT_8MS
```

3.6.2. AVR® LibCを使うﾋｭｰｽﾞの構成設定

Atmel Studioを使ってﾋｭｰｽﾞを構成設定するには下で示されるようにfuse.hﾍｯﾀﾞ ﾌｧｲﾙがｲﾝｸﾙｰﾄﾞされなければなりません。

```
#include <avr/fuse.h>     /* 必要とされるﾍｯﾀﾞ ﾌｧｲﾙ */
```

次の例はｳｫｯﾁﾄﾞｯｸﾞ構成設定(WDTCFG)ﾋｭｰｽﾞ ﾚｼﾞｽﾀを使ってｳｫｯﾁﾄﾞｯｸﾞ ﾀｲﾏを禁止し、CRCを禁止してｼｽﾃﾑ構成設定0(SYSCFG0)ﾚｼﾞｽﾀを使ってﾘｾｯﾄ ﾋﾟﾝをGPIO動作に設定し、ｼｽﾃﾑ構成設定1(SYSCFG1)ﾚｼﾞｽﾀを使って始動時間を8msに構成設定する方法を示します。

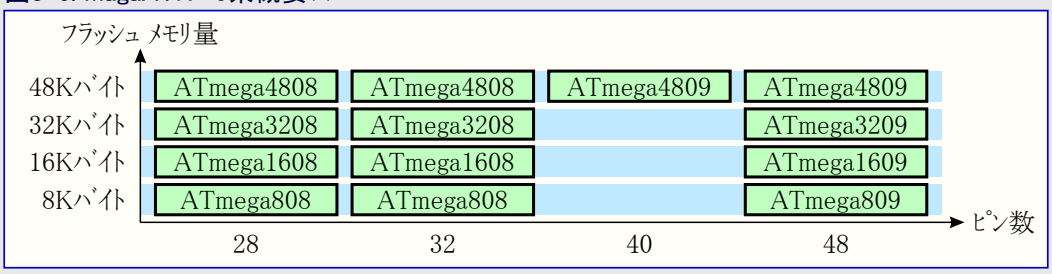
```
FUSES = {
    /* ｳｫｯﾁﾄﾞｯｸﾞ ﾀｲﾏを禁止 */
    .WDTCFG = PERIOD_OFF_gc,
    /* CRCを禁止してﾘｾｯﾄ ﾋﾟﾝの構成をGPIO(汎用入出力)動作に設定 */
    .SYSCFG0 = CRCSRC_NOCRC_gc | RSTPINCFG_GPIO_gc,
    /* 始動時間選択: 8ms */
    .SYSCFG1 = SUT_8MS_gc
};
```

⚠警告 使用者によって初期化されない場合、AVR LibC使用時にﾋｭｰｽﾞは既定値('0')で初期化されます。'0'と違わ(**訳補**:即ち、'1'で)なければならなくて初期化されないﾋｭｰｽﾞがある場合、ﾃﾞﾊﾞｲｽは意図されるように動かないかもしれません。

3.7. 単位部ﾎﾟｲﾝﾀを使う関数呼び出し

複数の実体を持つ単位部型に対してﾄﾞﾗｲﾊﾞを書く時に、全ての実体が同じﾚｼﾞｽﾀ ﾒﾓﾘ割り当てを持つと言う事実は、その単位部型の全ての実体に対してﾄﾞﾗｲﾊﾞを再使用可能にするように活用することができます。ﾄﾞﾗｲﾊﾞが関連する単位部実体を指し示すﾎﾟｲﾝﾀ引数を取るなら、そのﾄﾞﾗｲﾊﾞはこの型の全ての単位部に対して使うことができます。更に、書かれたｺｰﾄﾞは同じ系統のﾃﾞﾊﾞｲｽ間で可搬にすることができます。同じ系統のﾃﾞﾊﾞｲｽ間の互換性の詳細はﾃﾞｰﾀｼｰﾄ系統の「概要」部分で提供され、いくつかの違いが**図3-6.**で示されます。

図3-6. magaAVR® 0系概要(1)



複数のﾀｲﾏ/ｶｳﾝﾀを持つﾃﾞﾊﾞｲｽでは、これらの単位部を初期化してｱｸｾｽする関数は全ての実体に対して同じｺｰﾄﾞ行を複製する代わりに全ての単位部実体によって共用することができます。例え関数に単位部ﾎﾟｲﾝﾀを渡すことでの小さな付随負荷があるとしても、各単位部型の全ての実体に対してｺｰﾄﾞが再利用されるため、総合的なｺｰﾄﾞの大きさは減らされます。更に、開発時間、維持費用、可搬性はこの手法を使うことによって大きく改善することができます。

下のｺｰﾄﾞはどのﾀｲﾏ/ｶｳﾝﾀ単位部に対してもｸﾛｯｸ元を選ぶのに単位部ﾎﾟｲﾝﾀを使う関数を示します。

```
void TC_ConfigClockSource (volatile TCB_t *tc, TCB_CLKSEL_t clockSelection)
{
    tc->CTRLA = (tc->CTRLA & ~TCB_CLKSEL_gm) | clockSelection;
}
```

この関数はTCB_t型の単位部ﾎﾟｲﾝﾀとTCB_CLKSEL_tの群構成設定型の2つの引数を取ります。関数内のｺｰﾄﾞはCTRLAﾚｼﾞｽﾀをｱｸｾｽするのにﾀｲﾏ/ｶｳﾝﾀ単位部ﾎﾟｲﾝﾀを使い、tcﾊﾟﾗﾒｰﾀを通して提供されるｱﾄﾞﾚｽでﾀｲﾏ/ｶｳﾝﾀ単位部に新しいｸﾛｯｸ選択を設定します。下のｺｰﾄﾞは上で記述された関数が違うﾀｲﾏ/ｶｳﾝﾀ実体に異なる構成設定を設定するのにどう使われるかを示します。

```
/* CLKDIV2としてTCB0ｸﾛｯｸ選択を構成設定 */
TCB_ConfigClockSource (&TCB0, TCB_CLKSEL_CLKDIV2_gc);
/* CLKDIV1としてTCB0ｸﾛｯｸ選択を構成設定 */
TCB_ConfigClockSource (&TCB0, TCB_CLKSEL_CLKDIV1_gc);
/* CLKDIV2としてTCB1ｸﾛｯｸ選択を構成設定 */
TCB_ConfigClockSource (&TCB1, TCB_CLKSEL_CLKDIV2_gc);
/* CLKDIV2としてTCB2ｸﾛｯｸ選択を構成設定 */
TCB_ConfigClockSource (&TCB2, TCB_CLKSEL_CLKDIV2_gc);
```

4. コード書き代替法を示す応用例

利便性と旧版のAVRコード書き様式との保守互換性のため、構造体を伴わないプログラム書き様式を使うことも未だ可能です。本章はレジスタ アクセスとビット名使用の代替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ずに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体名その後に下線(_)とレジスタ名を加えて連結してください。同じ命名規則がアセンブリ言語でプログラムを書く時にも使われます。

例えば、タイマ/カウンタB型、実体0のCTRLAレジスタをアクセスするには、構造体を通してアクセスする代わりに、TCB0_CTRLAの名前を使うことができます。

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遮蔽と同じ名前を使って定義されますが、異なる接尾辞です。下のコードはレジスタを構成設定するのにビット位置がどう使われ得るかを示します。

```
PORTB_OUT |= (1 << PIN0_bp);                /* ポートBの出力(OUT)レジスタのビット0を設定(1) */
```

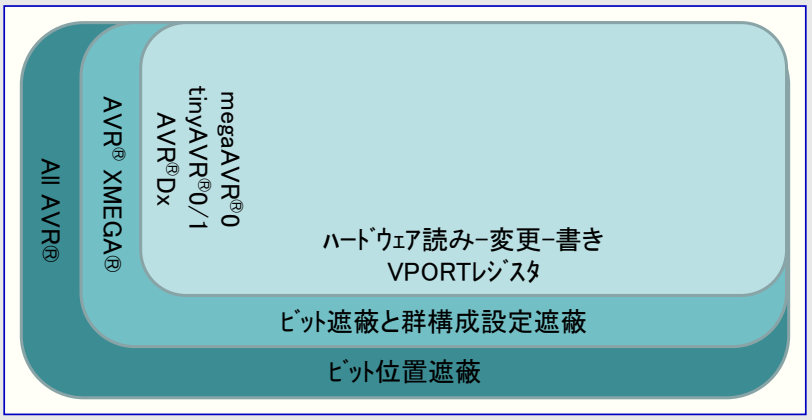
ビット位置定義は互換性の理由のためにインクルードされます。これらはビット番号を使う命令に対してアセンブリ言語でプログラムを書く時にも必要とされます。

4.3. 仮想ポート

仮想ポート(VPORT)レジスタはいくつかのPORTレジスタにビット アクセス可能なI/Oメモリ空間で仮想的に割り当てられることを許します。これが行われると、VPORTレジスタへの書き込みは実際のPORTレジスタへの書き込みと同じになります。これは拡張I/Oメモリ空間に属するPORTレジスタにビット操作命令(SBI/CBI)のようなI/Oメモリ特有命令の使用を許します。利用可能な仮想ポート数は殆どの場合で4または6に制限され、PORT単位部の数よりもより少ない利用可能な仮想ポート数になります。仮想ポート使用の利点はより小さくてより少ない複雑なプログラムとより少ない使用フラッシュ メモリです。

互換性構成図が**図4-1.**で示され、これは特定のAVR製品系統に対してどのコード書き様式が利用可能かを説明します。

図4-1. AVR®系統レジスタ構成設定任意選択



4.4. PORT例

本項は使用者釦を押す時にLEDをONにするようにPORTを構成設定する方法の例を含みます。マイクロ コントローラのどのピンが使用者LEDと使用者釦に配線されるかを識別するには使う基板の回路図が必要です。この例に使われるAVR128SDA48 Curiosity Nano基板についてはLEDがPORTCの6番ピン(PC6)に接続され、釦がPORTCの7番ピン(PC7)に接続されます。これがピンの正しい構成設定であることを確実にするため、基板の回路図を調べなければなりません。

下のコードはビット位置マクロを使ってLEDをONとOFFに切り替えるためのコードをどう書くかを示します。

例4-1. ビット位置を使って釦押下時にLEDをON

```
/* ピンの方向構成設定 */
/* 読み-変更-書き: ソフトウェア */
PORTC.DIR |= (1 << PIN6_bp);                /* 使用者LED用出力 */
PORTC.DIR &= ~(1 << PIN7_bp);               /* 使用者釦用入力 */
PORTC.PIN7CTRL |= (1 << PORT_PULLUPEN_bp);  /* プルアップ構成設定 */
while (1)
{
    if(PORTC.IN & (1 << PIN7_bp))           /* 釦状態調査 */
    {
        /* 釦開放 */
        PORTC.OUT |= (1 << PIN6_bp);        /* LEDをOFF */
```

```
    }
    else
    {
        /* 釦押下 */
        PORTC.OUT &= ~(1 << PIN6_bp);          /* LEDをON */
    }
}
```

注: これは周辺機能構成設定に対してAtmel STARTによって使われるｺｰﾄﾞ書き様式です。
下のｺｰﾄﾞは同じ機能を提供しますが、ﾋﾞｯﾄ遮蔽を使います。

例4-2. ﾋﾞｯﾄ遮蔽を使って釦押下時にLEDをON

```
/* ﾋﾟﾝの方向構成設定 */
/* 読み-変更-書き: ｿﾌﾄｳｪｱ */
PORTC.DIR |= PIN6_bm;                          /* 使用者LED用出力 */
PORTC.DIR &= ~ PIN7_bm;                        /* 使用者釦用入力 */
PORTC.PIN7CTRL |= PORT_PULLUPEN_bm;            /* ﾌﾟﾙｱｯﾌﾟ構成設定 */
while (1)
{
    if(PORTC.IN & PIN7_bm)                     /* 釦状態調査 */
    {
        /* 釦開放 */
        PORTC.OUT |= PIN6_bm;                  /* LEDをOFF */
    }
    else
    {
        /* 釦押下 */
        PORTC.OUT &= ~(PIN6_bm);               /* LEDをON */
    }
}
```

注: これはAVR用ｺｰﾄﾞを生成する時のMCCによって使われるｺｰﾄﾞ書き様式です。
また、下のｺｰﾄﾞで示されるように、読み-変更-書き操作をせずにﾋﾟﾝ値の設定(1)と解除(0)をするのにSETとCLRのﾚｼﾞｽﾀを使うことができます。これは先にﾚｼﾞｽﾀ値を読む代わりに、非分断命令を使って望む値の設定(1)と解除(0)を許します。主な利点はこの動きが割り込まれ得ないことです。3つ(読み-変更-書き)の代わりに1つの命令だけが実行されます。

例4-3. SETとCLRのﾚｼﾞｽﾀを使って釦押下時にLEDをON

```
PORTC.DIRSET = PIN6_bm;                        /* 使用者LED用出力 */
PORTC.DIRCLR = PIN7_bm;                        /* 使用者釦用入力 */
PORTC.PIN7CTRL |= PORT_PULLUPEN_bm;            /* ﾌﾟﾙｱｯﾌﾟ構成設定 */

while (1)
{
    /* 読み-変更-書き: ﾊｰﾄﾞｳｪｱ */
    if(PORTC.IN & PIN7_bm)                     /* 釦状態調査 */
    {
        /* 釦開放 */
        PORTC.OUTSET = PIN6_bm;                /* LEDをOFF */
    }
    else
    {
        /* 釦押下 */
        PORTC.OUTCLR = PIN6_bm;                /* LEDをON */
    }
}
```

出力ﾋﾟﾝの方向を構成設定して値を設定(1)/解除(0)する別の方法は下のｺｰﾄﾞで示されるように仮想ﾎﾟｰﾄを使うことによります。

例4-4. 仮想ﾎﾟｰﾄを使って釦押下時にLEDをON

```
/* ﾋﾟﾝの方向構成設定 */
/* 読み-変更-書き: VPORTﾚｼﾞｽﾀ */
VPORTC.DIR |= PIN6_bm;                         /* 使用者LED用出力 */
```

```
VPORTC.DIR &= ~PIN7_bm;                    /* 使用者釦用入力 */
PORTC.PIN7CTRL |= PORT_PULLUPEN_bm;        /* ﾌﾟﾙｱｯﾌﾟ構成設定 */
while (1)
{
    if(VPORTC.IN & PIN7_bm)                /* 釦状態調査 */
    {
        /* 釦開放 */
        VPORTC.OUT |= PIN6_bm;             /* LEDをOFF */
    }
    else
    {
        /* 釦押下 */
        VPORTC.OUT &= ~PIN6_bm;            /* LEDをON */
    }
}
```

4.5. ADC例

本項はAVR DA系列のﾃﾞﾊﾞｲｽからAVR128DA48ﾃﾞﾊﾞｲｽでADC0単位部に対する簡単な構成設定例を提示します。この制御器を書くのに必要とされる情報は「1. ﾃﾞｰﾀｼｰﾄ単位部基本構造と命名規定」章で言及されたように、ﾃﾞﾊﾞｲｽのﾃﾞｰﾀｼｰﾄで見つけることができます。

ADCを完全に構成設定するのに、ﾃﾞｰﾀｼｰﾄの「ADC – A/D変換器」章の「機能的な説明」項で見つけることができる推奨初期化手順に従うことができます。

ADC周辺機能が構成設定された後、ADC単位部は許可され、変換が開始されます。下のｺｰﾄﾞではﾋﾞｯﾄ位置ﾏｸﾛを使って構成設定が行われます。

例4-5. ﾋﾞｯﾄ位置を使ってADCを構成設定

```
/* ﾋﾞｯﾄ位置ﾏｸﾛを使うADCﾚｼﾞｽﾀ構成設定 */
ADC0.CTRLC |= (1 << ADC_PRESC0_bp) | (1 << ADC_PRESC1_bp); /* 前置分周器ﾋﾞｯﾄをDIV8に構成設定 */
ADC0.CTRLA |= (1<< ADC_ENABLE_bp);         /* ADCを許可 */
ADC0.COMMAND |= (1<< ADC_STCONV_bp);       /* 変換開始 */
```

代わりに、ADCは下のｺｰﾄﾞでのようにﾋﾞｯﾄ遮蔽を使って構成設定することができます。

例4-6. ﾋﾞｯﾄ遮蔽を使ってADCを構成設定

```
/* ﾋﾞｯﾄ遮蔽ﾏｸﾛを使うADCﾚｼﾞｽﾀ構成設定 */
ADC0.CTRLC |= ADC_PRESC0_bm | ADC_PRESC1_bm;  /* 前置分周器ﾋﾞｯﾄをDIV8に構成設定 */
ADC0.CTRLA |= ADC_ENABLE_bm;               /* ADCを許可 */
ADC0.COMMAND = ADC_STCONV_bm;              /* 変換開始 */
```

前置分周器構成設定を変更するのに群構成設定遮蔽を使うことができます。下のｺｰﾄﾞで示されるように元の構成設定が先に解除されなければなりません。

例4-7. 群構成設定遮蔽を使ってADCの前置分周器の構成設定を変更

```
/* 元の構成設定の解除を確実にしてADC前置分周器の構成設定を変更 */
ADC0.CTRLC = (ADC0.CTRLC & ~ADC_PRESC_gm) | ADC_PRESC_DIV4_gc;
```

5. 更なる段階

本章はIDEｲﾝｽﾄｰﾙ指針と利用可能な応用記述に使用者を導く目的を持ちます。

5.1. 応用記述と技術概説の説明

この技術概説の系列はAVRﾏｲｸﾛ ｺﾝﾄﾛｰﾗのmegaAVR® 0系統用の全ての周辺機能を網羅し、この文書で推奨される同じ原則を使います。各技術概説は網羅される使用事例の概要で始まります。各使用事例がその後に開発され、周辺機能を必要とされる構成設定で構成設定するためにﾃﾞｰﾀｼｰﾄをどう使うかを示します。

例えば、「A/D変換器(ADC)での開始に際して」技術概説は周辺機能の概要と、単独変換、自由走行、採取累積器などの異なる動作形態での様々な使用事例でこの周辺機能を使う方法での説明を提供します。

- TB3209 - A/D変換器(ADC)での開始に際して
- TB3211 - ｱﾅﾛｸﾞ比較器(AC)での開始に際して
- TB3213 - 実時間計数器(RTC)での開始に際して
- TB3214 - ﾀｲﾏ/ｶｳﾝﾀB型(TCB)での開始に際して
- TB3215 - 直列周辺ｲﾝﾀｰﾌｪｰｽ(SPI)での開始に際して
- TB3216 - 万能同期/非同期送受信器(USART)での開始に際して
- TB3217 - ﾀｲﾏ/ｶｳﾝﾀA型(TCA)での開始に際して
- TB3218 - 構成設定可能な注文論理回路(CCL)での開始に際して
- TB3229 - 汎用入出力(GPIO)での開始に際して

5.2. 素のAVR開発用関連映像

以下の映像はこの技術概説に対して特に関連します。

- Atmel Studio 7での開始に際して - 第10話 - I/Oｳｨﾝﾄﾞｳと素のﾌﾟﾛｸﾞﾗﾐﾝｸﾞ参考書
- 開始に際して - MPLAB® XでのAVR® - 脈絡的ﾃﾞｰﾀｼｰﾄ ﾍﾙﾌﾟ & AVR割り込み

以下は28部の映像系列で、これは主にﾌﾟﾛｸﾞﾗﾐﾝｸﾞ参考書としてﾃﾞｰﾀｼｰﾄとﾃﾞﾊﾞｲｽのﾍｯﾀﾞ ﾌｧｲﾙを機能的に使って構築します。

- AVR®での開始に際して

5.3. MPLAB® XC8ｺﾝﾊﾟｲﾗ

XCｺﾝﾊﾟｲﾗはどの適切なﾌﾟﾛｼﾞｪｸﾄのｿﾌﾄｳｪｱ開発に対しても包括的な解決策です。MPLAB XC8ｺﾝﾊﾟｲﾗ(2)は8ﾋﾞｯﾄのPIC®とAVRのﾏｲｸﾛ ｺﾝﾄﾛｰﾗを支援し、これは無料で使用制限なしのﾀﾞｳﾝﾛｰﾄﾞとして入手可能です。ﾌﾟﾛ許諾権も利用可能です。ﾌﾟﾛ許諾権を使うことにより、もっと効率的なｺｰﾄﾞを得ます。加えて、今や認定されたXC8機能安全許諾権も利用可能です。

MPLAB X IDEと組み合せると、この前置部はｿｰｽ ｺｰﾄﾞの対応する行に一致する誤りの編集と中断点(ﾌﾞﾚｰｸﾎﾟｲﾝﾄ)、重要な点で変数と構造体を調査するためにCｿｰｽ ｺｰﾄﾞを通す単一段階実行(ｼﾝｸﾞﾙ ｽﾃｯﾌﾟ)を提供します。

MicrochipのMPLAB X IDEのより多くの情報は”MPLAB X IDE使用者の手引き”を探すことにより、使用者の手引きの頁で見つけることができます。

5.4. IDE(MPLAB® X、Atmel Studio、IAR) - 開始に際して

AVRﾏｲｸﾛ ｺﾝﾄﾛｰﾗを書くにはMPLAB X、Atmel Studio、またはIAR Embedded Workbenchのどれかの IDEを使うことができます。

MPLAB X統合開発環境(IDE:Integrated Development Environment)は拡張可能で高度に構成設定可能なｿﾌﾄｳｪｱ ﾌﾟﾛｸﾞﾗﾑです。これはMicrochipのﾏｲｸﾛ ｺﾝﾄﾛｰﾗとﾃﾞｼﾞﾀﾙ信号制御器の殆どに対して見つけて構成設定して開発してﾃﾞﾊﾞｯｸﾞして組み込み設計に適性を与えるのを手助けする強力な道具を組み込みます。MPLAB X IDEはAVR MCUに対する支援を提供します。

実践と映像指導を含み、Atmel Studioに習熟するのに必要とされる全ての情報はこの使用者の手引き「Atmel Studio 7での開始に際して」で提供されます。加えて、ﾌﾟﾛｼﾞｪｸﾄを開発するの必要とされる全てのﾌﾟﾛｼﾞｪｸﾄ構成設定(ﾋｭｰｽﾞ書き込み、発振器校正、ｲﾝﾀｰﾌｪｰｽ設定など)の情報はAtmel Studio 7使用者の手引きで見つけることができます。どれかの基板に対してより多くの例を見つけて開発し、ﾄﾞﾗｲﾊﾞを構成設定、例ﾌﾟﾛｼﾞｪｸﾄを見つけ、ｼｽﾃﾑ ｸﾛｯｸ設定を容易に構成設定するのにｵﾝﾗｲﾝ ｺｰﾄﾞ構成設定ﾂｰﾙのAtmel STARTを使うことができます。より多くの詳細については「Atmel START使用者の手引き」を参照してください。

全ての特徴と共にAVR用IAR Embedded Workstationは「AN3419:AVR®用IAR Embedded Workbench®での開始に際して」応用記述で記述されます。

tinyAVR® 0/1系列、megaAVR 0系列、AVR DAのﾏｲｸﾛ ｺﾝﾄﾛｰﾗ系統はﾌﾟﾛｼﾞｪｸﾄを作成して何かのｽﾀｰﾀ ｷｯﾄを使う方法の情報と共にｵﾝﾗｲﾝで利用可能な専用の「～での開始に際して」系列の手引きも持ちます。これらの手引きの例が下で一覧にされます。

- megaAVR® 0系での開始に際して
- tinyAVR® 1系での開始に際して
- tinyAVR® 0系での開始に際して
- AVR DA系での開始に際して

6. 結び

本文書はAVRマイクロ コントローラのプログラムを書くのに優先されるコード書き様式を紹介します。この文書の経験後、データシートが提供する情報が何の型か、マクロ定義、変数宣言、ヘッダ ファイルによって提供されるデータ型定義も知ります。目標はAVRレジスタとビットに対する命名規則を習熟し、これらのマイクロ コントローラを使うプロジェクトの開発で更なる段階の準備を整えるため、容易な保守性、可搬性、可読性のコード書き様式を使うことです。

本文書は特定データシートでの情報、情報記述、命名規則、AVRマイクロ コントローラ用Cコード書き、コードを書く代替方法、それと最後にプロジェクトの開発での次の段階を網羅します。

Cコードを書くのに示唆された方法を使うことは必須ではありませんが、ここで示された利点が考慮され得ます。より大きなプロジェクトやより多くの機能でデバイスはより大きな利点を持ちます。

7. 参考文献

1. ATmega4808/4809データシート
2. MPLAB® XCコンパイラ
3. AVR Libcライブラリ参考書
4. MPLAB® XC8でのAVR®デバイス
5. AVR® MCU用MPLAB® XC8 Cコンパイラ使用者の手引き
6. Cプログラミング言語の基本
7. Cプログラミングの基本 – 列挙

8. 改訂履歴

文書改訂	日付	注釈
A	2020年5月	初版文書公開
B	2020年6月	表紙の「概要」標題を削除

Microchipウェブ サイト

Microchipはwww.microchip.com/で当社のウェブ サイト経由でのオンライン支援を提供します。このウェブ 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** – データシートと障害情報、応用記述と試供プログラム、設計資源、使用者の手引きとハードウェア支援資料、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、オンライン検討グループ、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、セミナーとイベントの一覧、Microchip営業所の一覧、代理店と代表する工場

製品変更通知サービス

Microchipの製品変更通知サービスは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www.microchip.com/supportでのウェブ サイトを通して利用できます。

Microchipデバイス コード保護機能

Microchipデバイスでの以下のコード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デ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。
- コード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のデ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はそれらのコード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ドの安全を保証することはできません。コード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。

コード保護は常に進化しています。Microchipは当社製品のコード保護機能を継続的に改善することを約束します。Microchipのコード保護機能を破る試みはデジタル 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

デバ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デバ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www.microchip.com/qualityを訪ねてください。

日本語© *HERO* 2021.

本技術概説はMicrochipのTB3262技術概説(DS90003262B-2020年6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: www.microchip.com/support ｳｪﾌﾞｱﾄﾞﾚｽ: www.microchip.com **ｱﾄﾗﾝﾀ** Duluth, GA Tel: 678-957-9614 Fax: 678-957-1455 **ｵｰｽﾁﾝ** TX Tel: 512-257-3370 **ﾎﾞｽﾄﾝ** Westborough, MA Tel: 774-760-0087 Fax: 774-760-0088 **ｼｶｺﾞ** Itasca, IL Tel: 630-285-0071 Fax: 630-285-0075 **ﾀﾞﾗｽ** Addison, TX Tel: 972-818-7423 Fax: 972-818-2924 **ﾃﾞﾄﾛｲﾄ** Novi, MI Tel: 248-848-4000 **ﾋｭｰｽﾄﾝ** TX Tel: 281-894-5983 **ｲﾝﾃﾞｱﾅﾎﾟﾘｽ** Noblesville, IN Tel: 317-773-8323 Fax: 317-773-5453 Tel: 317-536-2380 **ﾛｻﾝｾﾞﾙｽ** Mission Viejo, CA Tel: 949-462-9523 Fax: 949-462-9608 Tel: 951-273-7800 **ﾛｰﾘｰ** NC Tel: 919-844-7510 **ﾆｭｰﾖｰｸ** NY Tel: 631-435-6000 **ｻﾝﾎｾ** CA Tel: 408-735-9110 Tel: 408-436-4270 **ｶﾅﾀﾞ - ﾄﾛﾝﾄ** Tel: 905-695-1980 Fax: 905-695-2078	**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