AVR4030 : ATMELソフトウェア枠組み - 参照の手引き

要点

- 基本構造説明
- コード様式
- 設計様式
- ディレクトリ構造

1. 序説

ATMEL®フトウェア枠組み(略語ASF:ATMEL AVR Software Framework、www.atmel.com/asf)はATMELのmegaAVR®、AVR XMEGA®、AVR UC3、SAMデバイス用の応用構築に対してソフトウェアドライバとライブラリを提供します。それはソフトウェア設計の各種部品を共に開発して固めるのを手助けするように設計されました。これはオペレーティングシステム(OS)内に統合、または独立製品として動作することができます。

この応用記述に於いて、開発者はASFがどう設計され、どの規則が適用され、ASFでコードを使用して開発する方法について読むことができます。

この資料は開始に際しての手引きではありませんが、むしろASFの基礎を成す基本構造を記述します。

2. ソフトウェア インストールと構成設定

2.1. ダウンロード

ASFはATMEL Studio® 6(http://www.atmel.com/atmelstudio)に含まれます。AVR32 StudioとIAR™の使用者に対して、http://www.atmel.com/asfで独立した一括が利用可能です。ATMEL Studio使用者はASFがATMEL Studioに統合されているのでこの一括を必要としません。

2.2. オンラインAPI資料

公式のASFオンライン資料はhttp://asf.atmel.comに置かれます。

2.3. 公開注記

ASF公開注記資料はhttp://www.atmel.com/asfで利用可能で、以下が記述されます。

- 支援されるツール
- 支援されるデバイス
- 新機能
- バグ修正
- 既知の問題

2.4. バグ追跡器

公式のATMELソフトウェア枠組みバグ追跡器はhttp://asf.atmel.com/bugzilla/に置かれます。これはASFに関する全てのバグ報告に利用されるべきです。

2.5. 開始に際して

http://www.atmel.com/asf得られる、ATMELの「AVR4029:ATMELソフトウェア枠組み - 開始に際して」応用記述を参照してください。

3. ASFディレクトリ構造

ATMELソフトウェア枠組みはavr32¥、xmega¥、mega¥、common¥、sam¥、それとthirdparty¥のディレクトリの6つの主な部分に分けられます。これら6つのディレクトリはATMEL AVR UC3基本構造、ATMEL AVR XMEGA基本構造、ATMEL megaAVR基本構造、ATMEL SAM基本構造、それと全基本構造間共通と最後に第3者(社)ライブラリを表します。

ASFルート フォルダにあるものの概要は次の通りです。

avr32¥
common¥
mega¥
sam¥
thirdparty¥
xmega¥

ATMEL

マイクロ コントローラ

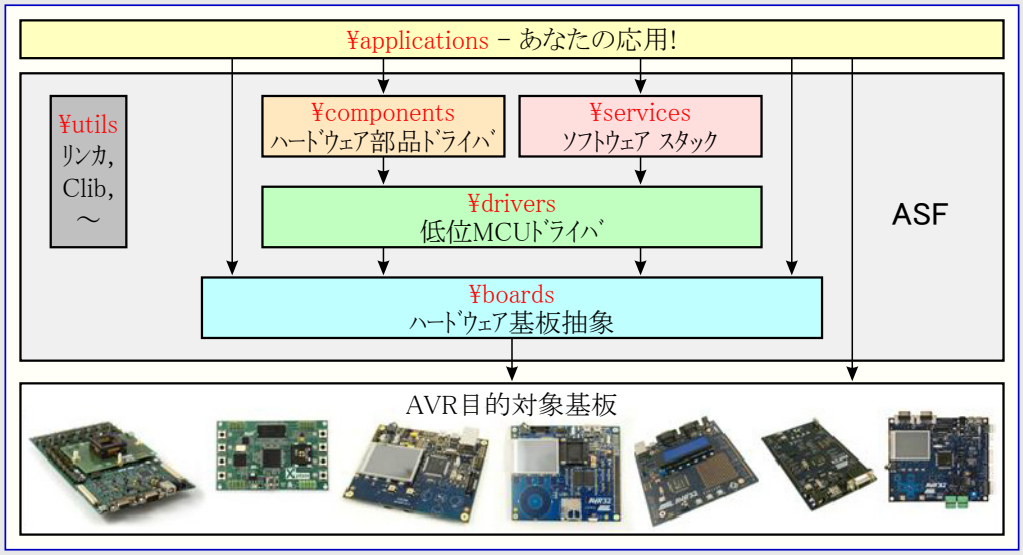
応用記述

Rev. 8432B-03/12, 8432BJ1-12/13

各基本構造(と共通ﾃﾞｨﾚｸﾄﾘ)は多数の副ﾃﾞｨﾚｸﾄﾘに分けられ、これらのﾃﾞｨﾚｸﾄﾘは基板、ﾄﾞﾗｲﾊﾞ、部品、ｻｰﾋﾞｽ、ﾕｰﾃｨﾘﾃｨの種々様々な単位部を含みます。種々様々な単位部が共にどう結び付けられるかの概要については下の一覧と図3-1.をご覧ください。

applications¥
boards¥
components¥
drivers¥
services¥
utils¥

図3-1. ATMELｿﾌﾄｳｪｱ枠組み部署構造



3.1. 基本構造と共通ﾃﾞｨﾚｸﾄﾘ構造

3.1.1. applications¥

このﾃﾞｨﾚｸﾄﾘはｻｰﾋﾞｽ、部品、ﾄﾞﾗｲﾊﾞ単位部に基づいた応用例を提供します。これらの応用はもっと高位で様々な単位部に対して多数の依存性を持つかもしれず、従って、ｳｪﾌﾞ ｻｰﾊﾞｰ、様々なUSB実演、ﾌﾞｰﾄﾛｰﾀﾞ、音響再生器などのような進んだ応用を実演します。

3.1.2. boards¥

このﾃﾞｨﾚｸﾄﾘは与えられた基本構造に対する様々な基板定義を含みます。基板ｺｰﾄﾞは物理的な書き込み、I/O初期化、外部ﾃﾞﾊﾞｲｽの初期化などから基板を上の単位部に抽象化します。基板ｺｰﾄﾞは上の単位部に対して何の基板機能が利用可能かも識別します。

全ての応用によって使用される基板入口点のboard.hﾌｧｲﾙは、それ複数の基本構造間で共用されるため、common¥boards¥board.hに置かれます。

3.1.3. components¥

このﾃﾞｨﾚｸﾄﾘはﾒﾓﾘ(例えば、ATMEL DataFlash®、SDRAM、SRAM、NANDﾌﾗｯｼｭ)、表示器、感知器、無線などのような外部ﾊｰﾄﾞｳｪｱ部品をｱｸｾｽするｿﾌﾄｳｪｱ ﾄﾞﾗｲﾊﾞを提供します。

これが基本構造間で共用される場合、componentsはcommon¥ﾃﾞｨﾚｸﾄﾘに置かれ、さもなければ適切な基本構造ﾃﾞｨﾚｸﾄﾘに置かれます。

3.1.4. drivers¥

各ﾄﾞﾗｲﾊﾞは周辺機能やﾃﾞﾊﾞｲｽ特有機能をｱｸｾｽする低位ﾚｼﾞｽﾀ ｲﾝﾀｰﾌｪｰｽ関数を提供する、driver.cとdriver.hのﾌｧｲﾙから成ります。ｻｰﾋﾞｽ(services)と部品(components)がﾄﾞﾗｲﾊﾞ(drivers)をｲﾝﾀｰﾌｪｰｽします。

3.1.5. services¥

このﾃﾞｨﾚｸﾄﾘはUSBｸﾗｽ、FATﾌｧｲﾙ ｼｽﾃﾑ、基本構造に最適化されたDSPﾗｲﾌﾞﾗﾘ、図画的ﾗｲﾌﾞﾗﾘなどのような、より応用志向のｿﾌﾄｳｪｱを提供します。

これが基本構造間で共用される場合、servicesはcommon¥ﾃﾞｨﾚｸﾄﾘに置かれ、さもなければ適切な基本構造ﾃﾞｨﾚｸﾄﾘに置かれます。

3.1.6. utils¥

このﾃﾞｨﾚｸﾄﾘは多数のﾘﾝｶ ｽｸﾘﾌﾟﾄ ﾌｧｲﾙ、ｼｽﾃﾑ構築用共通ﾌｧｲﾙと、一般用法定義、ﾏｸﾛ、関数を持つC/C++ﾌｧｲﾙを提供します。utils¥ﾃﾞｨﾚｸﾄﾘは特定の基本構造用のﾂｰﾙﾁｪｰﾝ間の違いに関する共通ｲﾝﾀｰﾌｪｰｽを作る方法も提供します。

これらが基本構造間で共用される場合、utilsのｺｰﾄﾞはcommon¥ﾃﾞｨﾚｸﾄﾘに置かれ、さもなければ適切な基本構造ﾃﾞｨﾚｸﾄﾘに置かれます。

3.2. 第3者(社)ﾃﾞｨﾚｸﾄﾘ基本構造

¥thirdpatyﾃﾞｨﾚｸﾄﾘはATMEL株式会社応用記述許諾文以外の各種許諾を持つ全てのｿﾌﾄｳｪｱで作られます。

thirdparty¥ﾃﾞｨﾚｸﾄﾘにあるものの概要は次の通りです。

cyberm¥
freertos¥
qtouch¥
～

thirdparty¥ﾃﾞｨﾚｸﾄﾘ内のこの単位部の各々はthirdparty¥<単位部>¥license.txtで許諾ﾌｧｲﾙを指定されるべきです。

4. ｺﾝﾊﾟｲﾗ支援

ATMELｿﾌﾄｳｪｱ枠組みはｺﾝﾊﾟｲﾗと無関係に用いられるのが狙いで、従ってｺﾝﾊﾟｲﾗ間の様々な違いは基本構造特有ﾍｯﾀﾞ ﾌｧｲﾙにしまい込みます。このﾌｧｲﾙはutils¥compiler.hで各基本構造ﾃﾞｨﾚｸﾄﾘの下に置かれます。

現在、ASFは8ﾋﾞｯﾄと32ﾋﾞｯﾄの両方のAVRとARMに対してGCCとIARを支援します。利用可能な最新のﾂｰﾙﾁｪｰﾝ版が開発に使用されるものであるべきです。

ASFﾂｰﾙでの開始を得るには、http://www.atml.com/asfで得られるATMELの「**AVR4029:ATMELｿﾌﾄｳｪｱ枠組み – 開始に際して**」応用記述を参照してください。

4.1. ATMEL Studio 6

ASFはGNU GCCｺﾝﾊﾟｲﾗに基づき、ATMEL Studio 6内に統合され、より多くの情報についてはhttp://www.atmel.com/atmelstudioを参照してください。

4.2. GNUｺﾝﾊﾟｲﾗ集合

GNU makefileは以下のように全てのASFに対して提供されます。

- 例えば、32ﾋﾞｯﾄAVRﾃﾞﾊﾞｲｽについては、ATMEL EVK1100基板上のATMEL AT32UC3A0512ﾃﾞﾊﾞｲｽに対するGPIO周辺機能ﾊﾞｽﾄﾞﾗｲﾊﾞ例用のGCCﾌﾟﾛｼﾞｪｸﾄ ﾌｧｲﾙが以下に置かれます。

  avr32¥drivers¥gpio¥peripheral_bus_example¥at32uc3a0512_evk1100¥gcc

- 例えば、ATMEL AVR XMEGAﾃﾞﾊﾞｲｽについては、ATMEL AVR Xplained基板上のATMEL ATxmega128A1に対するDMAﾄﾞﾗｲﾊﾞ例用のGCCﾌﾟﾛｼﾞｪｸﾄが以下に置かれます。

  xmega¥drivers¥dma¥example¥atxmega128a1_xplain¥gcc

4.3. IAR Embedded Workbench

IAR Embedded Workbench®作業空間がASFﾌﾟﾛｼﾞｪｸﾄ用に提供されます。

- 例えば、32ﾋﾞｯﾄATMEL AVRﾃﾞﾊﾞｲｽについてはIARﾌﾟﾛｼﾞｪｸﾄ ﾌｧｲﾙが以下に置かれます。

  avr32/drivers/gpio/peripheral_bus_example/ at32uc3a0512_evk1100/iar

- 例えば、AVR XMEGAﾃﾞﾊﾞｲｽについてはIARﾌﾟﾛｼﾞｪｸﾄが以下に置かれます。

  xmega/drivers/dma/example/atxmega128a1_xplain/iar

4.4. ﾂｰﾙﾁｪｰﾝ ﾍｯﾀﾞ ﾌｧｲﾙ

ﾂｰﾙﾁｪｰﾝ ﾍｯﾀﾞ ﾌｧｲﾙがﾊﾞｸﾞなしではないので、それらがﾊﾞｸﾞなしになるのを保証するために、きっといくつかのﾙｰﾁﾝが後続するでしょう。

現在の方法はATMELｿﾌﾄｳｪｱ枠組みに伴って更新されたﾂｰﾙﾁｪｰﾝ ﾍｯﾀﾞ ﾌｧｲﾙを出荷することです。使用者はその後にｺﾝﾊﾟｲﾗが期待したようにﾌｧｰﾑｳｪｱを生成するのを保証すべく、それらでﾂｰﾙﾁｪｰﾝを更新しなければなりません。

4.4.1. ﾊﾞｸﾞ報告

既にﾂｰﾙﾁｪｰﾝが最新のﾍｯﾀﾞ ﾌｧｲﾙ更新を持っていて、開発者が現在のﾂｰﾙﾁｪｰﾝでﾊﾞｸﾞに出会った時は、ﾊﾞｸﾞ報告が作られることが極めて重要です。

4.4.1.1. 定義されたｼﾝﾎﾞﾙに対する一時的な対策

ﾂｰﾙﾁｪｰﾝ ﾍｯﾀﾞ ﾌｧｲﾙの新しい公開を待つ間、ｿｰｽ ｺｰﾄﾞ内で一時的な対策が許されます。対策は代表的に開発内の特定単位部に対するｿｰｽ ｺｰﾄﾞ内で不正な定義を未定義にし、それを正しく定義します。この対策はそれの回避策を宣言する前で記録の行も持たなければならず、ﾍｯﾀﾞ ﾌｧｲﾙで修正された時に取り去られるべきです。

```
//! ¥todo ﾍｯﾀﾞ ﾌｧｲﾙ内のﾊﾞｸﾞに対する回避策を取り去ってください。
#undef DMA_CTRL
#define DMA_CTRL     _SFR_MEM8(0xCAFE)
```

#undef行はﾂｰﾙﾁｪｰﾝ ﾍｯﾀﾞ ﾌｧｲﾙ更新後にｺｰﾄﾞが自動的に誤りにならないのを保証することです。doxygen形式化された資料は、未終了ｺｰﾄﾞについての残りを開発者に与える資料を生成する時の対策ﾎﾟｯﾌﾟｱｯﾌﾟを保証します。

4.4.1.2. 型定義に対する一時的な対策

ﾂｰﾙﾁｪｰﾝ ﾍｯﾀﾞ ﾌｧｲﾙの新しい公開を待つ間、ｿｰｽ ｺｰﾄﾞ内で一意的な対策が許されます。対策はそれが行われるべきとして新しい型定義を定義することですが、型定義名の_t部分の前に接尾子として_tmpfixを追加することによって名前を変えます。

この対策はそれの回避策を宣言する前で記録の行も持たなければならず、ﾍｯﾀﾞ ﾌｧｲﾙで修正された時に取り去られるべきです。

```
//! ¥todo ﾍｯﾀﾞ ﾌｧｲﾙ内のﾊﾞｸﾞに対する回避策を取り去ってください。
typedef struct avr32_dmaca_tmpfix {
    unsigned long         sar0;
    (～)
} avr32_dmaca_tmpfix_t;
```

doxygen形式化された資料は、未終了ｺｰﾄﾞについての残りを開発者に与える資料を生成する時の対策ﾎﾟｯﾌﾟｱｯﾌﾟを保証します。

4.4.2. ﾍｯﾀﾞ ﾌｧｲﾙの更新

ASF内の様々な基本構造はutils¥header_files¥ﾃﾞｨﾚｸﾄﾘ内に置かれたﾍｯﾀﾞ ﾌｧｲﾙ一括を持ちます。同じﾃﾞｨﾚｸﾄﾘ内のreadme.txtは使用者のﾂｰﾙﾁｪｰﾝ ﾍｯﾀﾞ ﾌｧｲﾙを更新する方法を使用者に指示します。ASF貯蔵庫内のﾍｯﾀﾞ ﾌｧｲﾙ一括で直接編集が行われてはなりません。

5. ｺｰﾄﾞ様式

本章はATMELｿﾌﾄｳｪｱ枠組み内の全てのｺｰﾄﾞ部品での使用が必要とされる命名規則とｺｰﾄﾞ様式を含みます。

5.1. 一般命名規則

関数、変数、定数または型の名前がﾃﾞｰﾀｼｰﾄまたは他の仕様書資料から創作する場合、可能な限りそれらが使用される様式に合わせるべきです。

例えば、例えそれがこの頁で指定されるｺｰﾄﾞ書き様式に違反しても、USB構成設定(Setup)要求の要求識別子領域にbRequestと名付けることは、USB 2.0仕様を熟知するどんな人もその領域が何のためかを直ぐに理解するので、それは完全にOKで、まさに好ましくあります。

5.2. 関数と変数の名前

- 関数と変数は全て小文字(a～z)と数字(0～9)を用いて名付けられます。
- 下線'_'は関数と変数名をより論理的な群に分けるのに使用されます。
- 変数名は使用される型名と違わなければなりません(悪例"static name namw[2]")。

5.2.1. 例

```
void this_is_a_function_prototype(void);
```

5.2.2. 理論的根拠

全て小文字の名前は目に優しく、それはCｺｰﾄﾞで見られる非常に一般的な様式です。

5.3. 定数

- 定数は全て大文字(A～Z)と数字(0～9)を用いて名付けられます。
- 下線'_'は定数名をより論理的な群に分けるのに使用されます。
- 列挙定数はこの規則に従うべきです。
- 式から作られる定数は式全体を囲む括弧'()'を持たなければならず、単一値定数はこれを省略できます。

5.3.1. 例

```
#define BUFFER_SIZE              512
#define WTK_FRAME_RESIZE_WIDTH   (WTK_FRAME_RESIZE_RADIUS + 1)

enum buffer_size = {
    BUFFER_SIZE_A = 128,
    BUFFER_SIZE_B = 512,
};
```

5.3.2. 理論的根拠

定数はｺｰﾄﾞのその他に介入すべきではなく、全て大文字の名前がそれを保証します。また、全て大文字の定数は、そのように多くの定数が由来することから、仕様書資料やﾃﾞｰﾀｼｰﾄで非常に一般的です。

式を囲む括弧は予期せぬ評価を避けるために重要です。例えば共に加算される2つの変数を含む定数は、後でｿｰｽ ｺｰﾄﾞ内の変数で乗算されます。

列挙もまた定数で、故にﾌﾟﾘﾌﾟﾛｾｯｻ定数と同じ規則に従ってそれらを持つことは意味を成します。

5.4. 型定義

- stdint.hとstdbool.hの型が利用可能な時に使用されなければなりません。
- 型定義は全て小文字(a～z)と数字(0～9)を用いて名付けられます。
- 下線'_'は名前をより論理的な群に分けるのに使用されます。
- 全ての型定義は末尾の'_t'を持たなければなりません。

5.4.1. 例

```
typedef uint8_t buffer_item_t;
```

5.5. 構造体と共用体

- 構造体と共用体は関数と変数としての命名規則に従います。
- それが本当に必要でなければ、typedefを使用しないでください。typedefは以下の場合にだけOKです。
  ○ 型定義が基本構造依存で、構造体、共用体またはｽｶﾗとして定義することができます。

5.5.1. 例

```
struct cmd {
    uint8_t length;
    uint8_t *payload;
};

union cmd_parser {
    struct cmd cmd_a;
    struct cmd cmd_b;
};
```

5.6. ﾏｸﾛのような機能

- ﾏｸﾛのような機能は関数や変数と同じ命名規則に従います。この方法は後の段階に於いてそれらをｲﾝﾗｲﾝ関数に交換するのが容易です。
- ﾏｸﾛのような機能が可能な場所はdo {} while (0)内に閉塞されるべきです。
- ﾏｸﾛのような機能は1つよりも多いそれらの引数を決してｱｸｾｽしてはなりません。
- ﾏｸﾛ定義それ自身だけでなく全てのﾏｸﾛの引数も、括弧に入れられなければなりません。

5.6.1. 例

```
#define set_io(id) do {       ¥
    PORTA |= (1 << (id));     ¥
} while (0)
```

5.6.2. 理論的根拠

可能な限り正規の関数に合うように振舞うﾏｸﾛのような機能を望みます。これはそれらが単一命令行("void"ﾏｸﾛに対するdo {} while (0)覆い部とこれを保証する値を返すﾏｸﾛに対する周囲を被う括弧)として評価されなければならないことを意味します。

演算子先行に関連するどんな番狂わせも避けるためにﾏｸﾛの引数は括弧に入れられなければならず、それがﾏｸﾛの内側の式で使用されつつあるのに先立って完全に評価されべきことを引数に望みます。また、いくつかのﾏｸﾛ引数の評価が副作用を持つかもしれず、故にﾏｸﾛは一度だけ評価されることを保証しなければなりません(sizeofとtypeofの式は数えません)。

5.7. 字下げ

- 字下げはTAB文字を用いることによって行われます。この方法は異なるｴﾃﾞｨﾀの形態設定がｿｰｽ ｺｰﾄﾞと整列を乱雑にしないことを保証します。
- TAB文字は字下げに関して、式/文の前で使用されなければなりません。
- TABは式/文の後で使用されてはならず、代わりに空白' 'を使用してください。
- TAB文字の大きさ(幅)は固定化されません。とにかく、開発者が充分に大きいTAB文字(幅)を使用することが推奨され、故に可読性が成し遂げられます。更に、大きな字下げは制御部の深い入れ子を避ける、簡単な方法です。

5.7.1. 例

```
enum scsi_asc_ascq {
[TAB]                                   [空白]
    SCSI_ASC_NO_ADDITIONAL_SENSE_INFO          = 0x0000,
    SCSI_ASC_LU_NOT_READY_REBUILD_IN_PROGRESS  = 0x0405,
    SCSI_ASC_WRITE_ERROR                       = 0x0c00,
    SCSI_ASC_UNRECOVERED_READ_ERROR            = 0x1100,
    SCSI_ASC_INVALID_COMMAND_OPERATION_CODE    = 0x2000,
    SCSI_ASC_INVALID_FIELD_IN_CDB              = 0x2400,
    SCSI_ASC_MEDIUM_NOT_PRESENT                = 0x3a00,
    SCSI_ASC_INTERNAL_TARGET_FAILURE           = 0x4400,
};
```

5.7.2. 理論的根拠

TAB文字の大きさは各開発者に対して違えることができます。固定の大きさを押し付けることはできません。最良の可読性を持つために、TAB文字は式と文に先立って行上でだけ使用することができます。式と文の後でTAB文字は使用されてはならず、代わりに空白を使用してください。

字下げについての全体の主眼点は制御部の始まりと終わりの場所を明らかに示すことです。大きな字下げでは、小さな字下げでよりもお互いから様々な字下げ段階を区別することがずっと容易です(2文字字下げでは些細でない関数を理解するのは殆ど不可能です)。

大きな字下げの別な利点は増された段階の入れ子を持つｺｰﾄﾞを書くことが段々難しくなり、故に関数を複数、もっと簡単な単位に分割し、従って可読性を更に改善するための良い動機付けを提供することです。これはその上に遵守されるべき80文字規則が明らかに必要です。

TAB使用がｺｰﾄﾞを正しく整列しなくさせることに関係された場合、継続についての項をご覧ください。

5.8. 文体裁

- ｺｰﾄﾞの1行、文書などは8つの空白のTAB字下げを与えられて、80文字を超えるべきではありません。
- 80文字よりも長い文章行は覆われて2重字下げされるべきです。

5.8.1. 例

```
/* This is a comment which is exactly 80 characters wide for example
showing. */

    dma_pool_init_coherent(&usbb_desc_pool, addr, size,
           sizeof(struct usbb_sw_dma_desc), USBB_DMA_DESC_ALIGN);

#define unhandled_case(value)                                          ¥
    do {                                                             ¥
           if (ASSERT_ENABLED) {                                     ¥
                  dbg_printf_level(DEBUG_ASSERT,                     ¥
                         "%s:%d: Unhandled case value %d¥n",         ¥
                         __FILE__, __LINE__, (value));               ¥
                  abort();                                           ¥
           }                                                         ¥
    } while (0)
```

5.8.2. 理論的根拠

行幅を80文字以下に保つことは、例え小さな画面でも行を中断することなくﾌｧｲﾙの内容を見ることが可能です。それは極端な段階の入れ子の認識も手助けします。一般的に可読性を改善します。

5.9. 空白

・2項と3項の演算子の周囲に空白を置いてください。
・式/文の後では、TAB文字の代わりに空白を使用してください。
・単項演算子の後に空白を置かないでください。
・括弧と式のそれらの内側の間に空白を置かないでください。
・関数呼び出しと関数定義で関数名とﾊﾟﾗﾒｰﾀ間に空白を置かないでください。

5.9.1. 例

```
fat_dir_current_sect
    = ((uint32_t)(dclusters[fat_dchain_index].cluster + fat_dchain_nb_clust - 1)
        * fat_cluster_size) + fat_ptr_data + (nb_sect % fat_cluster_size);
```

5.10. 継続

・継続は単一行に適合しない長い式を中断するのに使用されます。
・継続は字下げ段階に対して余分なTABを追加することによって行われるべきです。

5.10.1. 例

```
static void xmega_usb_udc_submit_out_queue(struct xmega_usb_udc *xudc,
            usb_ep_id_t ep_id, struct xmega_usb_udc_ep *ep)
{
    (～)
}

#define xmega_usb_read(reg) ¥
    mmio_read8((void *)(XMEGA_USB_BASE + XMEGA_USB_##reg))
```

5.10.2. 理論的根拠

余分なTABを使用して継続を字下げすることにより、同一段階と次の字下げ段階の両方でｺｰﾄﾞからの区別が常に容易です。後者は特にif、whileとfor命令行で重要です。

また、整列されるべきものが何も必要ない(それは開始されつつある塊と同じ字下げ段階で終了することで度々引き起こされる)ことにより、TABの大きさ(幅)に拘らず等しく上手く整列するでしょう。

5.11. 注釈

・短い注釈は以下のように使用できます。

```
// 注釈
(～)
```

・長い(複数行)注釈は次のように使用すべきです。

```
/*
 * 複数行に覆われるかもしれない長い注釈
 */
(～)
```

5.12. 中括弧'{ }'

・開始中括弧は関数定義を除き、全ての場合で行の最後に置くべきです。閉鎖中括弧は式と同じ字下げ段階に置きます。
・中括弧内側のｺｰﾄﾞは字下げされます。
・単一行ｺｰﾄﾞ部も中括弧で包まれるべきです。

5.12.1. 例

```
if (byte_cnt == MAX_CNT) {
    do_something();
} else {
    do_something_else();
}
```

5.13. ポインタ宣言

ポインタ宣言時、変数にアスタリスク(*)を繋げてください。

5.13.1. 例

```
uint8_t *p1;
```

5.14. 複合命令行

- 開始中括弧は括弧に入れられた式に直接続いて、行の最後に配置されます。
- 閉鎖中括弧は後続する本体行の始めに配置されます。
- 同じ命令行(例えば、'esle'または'else if'命令行、またはdo/whileでのwhile命令行)のどんな継続も、閉鎖中括弧と同じ行に配置されます。
- 本体は周囲のコードよりも1段階多く字下げされます。
- 'if','else','do','while'と'switch'キーワードは空白によって後続されます。

5.14.1. 例

```
if (byte_cnt == MAX1_CNT) {
    do_something();
} else if (byte_cnt > MAX1_CNT) {
    do_something_else();
} else {
    now_for_something_completely_different();
}

while (i <= 0xFF) {
    ++i;
}

do {
    ++i;
} while (i <= 0xFF);

for (i = 0; i < 0xFF; ++i) {
    do_something();
}

/* Following example shows how to break a long expression. */
for (uint8_t i = 0, uint8_t ii = 0, uint8_t iii = 0;
            (i < LIMIT_I) && (ii < LIMIT_II) && (iii == LIMIT_III);
            ++i, ++ii, ++iii) {
    do_something();
}
```

5.14.2. 理論的根拠

これは標準的なK&R様式です。これは可読性と空間効率の両方です。分離した行での括弧配置は可読性に何も寄与しませんが、沢山の余分な垂直空間を浪費するかもしれません。字下げは視覚的に本体が残りから分離されることを保証します。

5.15. "switch case"命令行

- switch部は他の複合命令行と同じ規則に従います。
- caseラベルはswitchキーワードと同じ字下げ段階です。
- breakは各ラベル内側のコードと同じ字下げ段階です。。
- 各ラベル内側のコードは字下げされます。

5.15.1. 例

```
switch (byte_cnt) {
case 0:
        ～
        break;

case 1:
case 2:
        ～
        break;

default:
        ～
}
```

5.16. ﾌﾟﾘﾌﾟﾛｾｯｻ指示

・#演算子は常に行の始めに置かれなければなりません。

・指示は#後で(必要とされるならば)字下げされます。

5.16.1. 例

```
#if (UART_CONF == UART_SYNC)
# define INIT_CON        (UART_EN | UART_SYNC | UART_PAUSE)
#elif (UART_CONF == UART_ASYNC)
# define INIT_CON        (UART_EN | UART_ASYNC)
#elif (UART_CONF==UART_PCM)
# define INIT_CON        (UART_EN | UART_PCM | UART_NO_HOLE)
#else
# error Unknown UART configuration
#endif
```

5.17. ﾍｯﾀﾞ ﾌｧｲﾙ

5.17.1. ﾍｯﾀﾞ ﾌｧｲﾙのｲﾝｸﾙｰﾄﾞ

ﾍｯﾀﾞ ﾌｧｲﾙのｲﾝｸﾙｰﾄﾞ時に現在のﾌｧｲﾙのﾊﾟｽに関連してｲﾝｸﾙｰﾄﾞされるﾌｧｲﾙに対して""を、ｲﾝｸﾙｰﾄﾞ ﾊﾟｽに関連してｲﾝｸﾙｰﾄﾞされるﾌｧｲﾙに対して<>を使用します。本来、これは""がASF単位部それ自身からｲﾝｸﾙｰﾄﾞされるﾌｧｲﾙ用で、一方<>が他のASF単位部からｲﾝｸﾙｰﾄﾞされるﾌｧｲﾙ用であることを意味します。

例えば、adc.cに於いて次のようにそれに応じてｲﾝｸﾙｰﾄﾞできます。

```
#include <compiler.h>
#include "adc.h"
```

5.17.2. ﾍｯﾀﾞ ﾌｧｲﾙ保護

ｲﾝｸﾙｰﾄﾞ保護は2重ｲﾝｸﾙｰﾄﾞの問題を避けるのに使用されます。単位部ﾍｯﾀﾞ ﾌｧｲﾙ ｲﾝｸﾙｰﾄﾞ保護はMODULE_H_INCLUDEDからでなければなりません。

例えば、adc.hでは次の通りです。

```
#ifndef ADC_H_INCLUDED
#define ADC_H_INCLUDED
    ～
#endif // ADC_H_INCLUDED
```

6. 意匠様式

6.1. 単位部ﾌｧｲﾙ名と配置

- 単位部ﾌｧｲﾙ名は単位部名それ自身と同じであるべきです。
- ﾌｧｲﾙは単位部の後のﾃﾞｨﾚｸﾄﾘ名で群化されるべきです。
- 単位部ﾃﾞｨﾚｸﾄﾘはﾃﾞｨﾚｸﾄﾘ構造定義によって与えられるようにASF内で適切に配置されるべきです。
- 新しい先頭ﾚﾍﾞﾙのﾃﾞｨﾚｸﾄﾘは作成されるべきではありません。
- 共通単位部はcommon¥ﾃﾞｨﾚｸﾄﾘに行くべきで、基本構造特有の単位部はそれの適切な基本構造ﾃﾞｨﾚｸﾄﾘ内に行きます。
  - ○ 共通単位部の基本構造部分は共通単位部と共に群化されるべきです。

6.1.1. 例外

上の規則は常に意味を成さない、または素直な解決策を妨げます。従ってASF保守者が逸脱を是認する限り、上の規則を緩めることが可能です。

加えてconf_*.h、主応用ﾌｧｲﾙなどのような特別なﾌｧｲﾙがこの規則から逸脱するかもしれません。

6.1.2. 例

6.1.2.1. ﾄﾞﾗｲﾊﾞの位置

{avr32, common, ～}¥drivers¥<単位部>¥<単位部>.{c h}
avr32¥drivers¥gpio¥gpio.c

6.1.2.2. 基本構造特有ｻｰﾋﾞｽの位置

{avr32, xmega, ～}¥services¥<単位部>¥<単位部>.{c h}
common¥services¥delay¥delay.c

6.1.2.3. 基本構造特有部分を持つ共通ｻｰﾋﾞｽの位置

common¥services¥<単位部>¥<単位部>.{c h}
common¥services¥<単位部>¥<基本構造>¥<基本構造_単位部>.{c h}

common¥services¥clock¥sysclk.h
common¥services¥clock¥xmega¥xmega_sysclk.h

6.1.2.4. 部品の位置

{avr32, common, ～}¥components¥<単位部>¥<単位部>.{c h}
avr32¥components¥touch¥resistive_touch.c

6.1.2.5. ﾄﾞﾗｲﾊﾞで副構造を持つ時の例外

avr32¥drivers¥usbb¥usbb_device.c
avr32¥drivers¥usbb¥usbb_host.c
avr32¥drivers¥usbb¥usbb_otg.c

6.2. 共通応用ﾌﾟﾛｸﾞﾗﾐﾝｸﾞ ｲﾝﾀｰﾌｪｰｽ

ATMEL ASFはいくつかの共有ｻｰﾋﾞｽと基本構造間の部品を提供しますが、ﾄﾞﾗｲﾊﾞに対する共用ｲﾝﾀｰﾌｪｰｽを提供しません。加えて先頭ﾚﾍﾞﾙのboard.hはutils¥ﾃﾞｨﾚｸﾄﾘ内のｺｰﾄﾞ ﾕｰﾃｨﾘﾃｨの共通部分と共に全ての基本構造間で共用されます。

6.2.1. 共用されるｻｰﾋﾞｽ

共用される全てのｻｰﾋﾞｽはcommon¥services¥ﾃﾞｨﾚｸﾄﾘに配置されます。これらの共通ｻｰﾋﾞｽは全ての基本構造に対して同じｲﾝﾀｰﾌｪｰｽを持ちます。

6.2.2. 共用される部品

共用される部品はcommon¥components¥ﾃﾞｨﾚｸﾄﾘに配置され、ATMEL DataFlashのような外部ﾃﾞﾊﾞｲｽに共通ｲﾝﾀｰﾌｪｰｽを追加するため、代表的に共用ｻｰﾋﾞｽを用います。

6.2.3. 共用されるｺｰﾄﾞ ﾕｰﾃｨﾘﾃｨ

共用されるﾕｰﾃｨﾘﾃｨはcommon¥utils¥ﾃﾞｨﾚｸﾄﾘに配置され、これらのﾕｰﾃｨﾘﾃｨは全ての基本構造に対して同じｲﾝﾀｰﾌｪｰｽを持ちます。代表的な共用ﾕｰﾃｨﾘﾃｨは割り込み制御と標準入出力(stdio)単位部です。

6.3. 類似応用ﾌﾟﾛｸﾞﾗﾐﾝｸﾞ ｲﾝﾀｰﾌｪｰｽ

理想的には全ての単位部が基本構造に渡って同じAPIを持つべきですが、これが常に可能または適用可能ではないため、各単位部は同様のAPIのために努力すべきです。加えて上の階層は適切な時に副単位部を共に繋げることができます。

種々様々な基本構造間で共用される単位部に関し、開発者は可能な限り互換ｲﾝﾀｰﾌｪｰｽのために努力すべきです。互換はここで(全てで必要とされる場合に)妥当な量の結合ｺｰﾄﾞで共用することができるｲﾝﾀｰﾌｪｰｽを記述する方法です。

共用APIの代わりに類似APIを行うことの理由は以下のような多数の理由の1つかもしれません。

- ﾌﾗｯｼｭ ﾒﾓﾘとSRAMの専有空間低減
- 消費電力低減
- 性能改善
- 機能支援改善

いくつかの状況について類似APIは全てで意味を成さないかもしれず、基本構造特有単位部を実装することができます。これは各種基本構造が様々な応用区分に向かって目標化されることによって合理化されます。あなたはATMEL tinyAVR®がﾏﾙﾁﾒﾃﾞｨｱ再生機から圧縮した音楽を接続して復号するのを見ることを予想しないでしょう。

6.3.1. 例

下の例はﾊｰﾄﾞｳｪｱ ﾄﾞﾗｲﾊﾞまたは通信ｻｰﾋﾞｽのような新しい単位部に対して使用される代表的な指針を示します。

6.3.1.1. 初期化

単位部を初期化する推奨方法は下の例で、類似API法が提供される場合にﾊﾟﾗﾒｰﾀは基本構造で違うかもしれません。

```
<単位部>_init(～)
adc_init(adc_t *adc, adc_options_t *options)
```

6.3.1.2. 許可

単位部を許可する推奨方法は下の例で、類似API法が提供される場合にﾊﾟﾗﾒｰﾀは基本構造で変更するかもしれません。

```
<単位部>_enable(～)
adc_enable(adc_t *adc)
```

6.3.1.3. 禁止

単位部を禁止する推奨方法は下の例で、類似API法が提供される場合にﾊﾟﾗﾒｰﾀは基本構造で変更するかもしれません。

```
<単位部>_disable(～)
adc_disable(adc_t *adc)
```

6.3.1.4. 開始

単位部を開始する推奨方法は下の例で、類似API法が提供される場合にﾊﾟﾗﾒｰﾀは基本構造で変更するかもしれません。

```
<単位部>_start(～)
adc_start(adc_t *adc)
```

6.3.1.5. 停止

単位部を停止する推奨方法は下の例で、類似API法が提供される場合にﾊﾟﾗﾒｰﾀは基本構造で変更するかもしれません。

```
<単位部>_stop(～)
adc_stop(adc_t *adc)
```

6.3.1.6. 書き込み

単位部にﾃﾞｰﾀを書く推奨方法は下の例で、類似API法が提供される場合にﾊﾟﾗﾒｰﾀは基本構造で変更するかもしれません。

```
<単位部>_write(～)
adc_write(adc_t *adc, uint16_t value)
```

6.3.1.7. 読み込み

単位部からﾃﾞｰﾀを読む推奨方法は下の例で、類似API法が提供される場合にﾊﾟﾗﾒｰﾀは基本構造で変更するかもしれません。

```
<単位部>_read(～)
adc_read(adc_t *adc, uint16_t *value)
```

6.4. 資料

ATMELｿﾌﾄｳｪｱ枠組みはDoxygen資料化ﾂｰﾙによって読むべく構成された埋め込み文書に基づきます。開放ｿｰｽ企画のDoxygenについてのより多くの情報に関しては、http://www.doxygen.org/を訪ねてください。

Doxygenはｿｰｽ ｺｰﾄﾞ内の注釈部でのそれの入力に基づき、入力ﾌｧｲﾙでの/**と//!のような引き金で発動します。

全てのｿｰｽ ｺｰﾄﾞ ﾌｧｲﾙはATMEL株式会社応用記述許諾文書を持つべきです。加えてﾌｧｲﾙが何に関するものなのかについての幾許かの情報を持つべきです。

注: 既存公開ｿﾌﾄｳｪｱについて、年領域はﾌｧｲﾙが作成された年で始まり、ﾌｧｲﾙが最後に変更された年を一覧にするように更新されます。

- 例えば、2008に作成
  Copyright (C) 2008 Atmel Corporation. All rights reserved.
- 例えば、2008に作成して変更
  Copyright (C) 2008 Atmel Corporation. All rights reserved.
- 例えば、2008に作成して、2009に変更
  Copyright (C) 2008 - 2009 Atmel Corporation. All rights reserved.
- 例えば、2008に作成して、2010に変更
  Copyright (C) 2008 - 2010 Atmel Corporation. All rights reserved.
- 例えば、2008に作成して、2009, 2010, 2012に変更
  Copyright (C) 2006 - 2012 Atmel Corporation. All rights reserved.

生成された資料でより良い出力を提供するために単位部内に資料を一群にすることが推奨されます。これはDoxygen付箋の¥defgroupと¥ingroupによって行われます。

Doxygenは生成した資料の内容と配置を改善するための多数の機能も提供します。更なる詳細を得るためにDoxygenのｳｪﾌﾞｻｲﾄを訪ねてそれらの資料を閲覧してください。

6.4.1. 例

6.4.1.1. 複数行注釈用Doxygen文書開始付箋

```
/**
 * <文書>
 */
```

6.4.1.2. 単一行注釈用Doxygen文書開始付箋

```
//! <文書>
```

6.4.1.3. ﾌｧｲﾙ文書

```
/**
 * ¥file
 *
 * ¥brief AVR XMEGA Direct Memory Access Controller driver
 *
 * Copyright (C) 2011 Atmel Corporation. All rights reserved.
 *
 * ¥page License
 *
 * <Atmel Corporation application note license>
 */
```

6.4.1.4. Dxygen群作成

```
/**
 * ¥defgroup sensible_group_name My module (ABBREVIATED)
 *
 * This is some contents that will show up within this group.
 */
```

6.4.1.5. 文書群に内容追加

```
/**
 * ¥ingroup sensible_group_name
 *
 * ここに書かれた内容は直前に言及したDoxygen付箋で書かれた内容と合併されます。
 * これは多数のﾌｧｲﾙ間で文書を分けることを可能にします。
 */
```

6.5. 即時開始の指針

ASF内のﾄﾞﾗｲﾊﾞはそれらのAPI資料の一部として即時開始の指針を持ちます。これらは1つまたはそれ以上使用する場合にﾄﾞﾗｲﾊﾞを構成設定して使用するのに必要とされるｺｰﾄﾞと活動を段階的な形式で見せて説明します。

即時開始の指針用のdoxygenｺｰﾄﾞは混乱を避けるためにAPIﾌｧｲﾙの最後に置かれます。API資料に於いて、即時開始の指針は記述内の最初のもののﾘﾝｸを示すことによって専用項目で容易にｱｸｾｽ可能にされます。

最も基本的な使用事例は即時開始の指針の最初の頁で示され、一方もっと高度な使用事例は個別副頁で示されます。高度な使用事例へのﾘﾝｸの一覧は各々簡単な説明を伴い、即時開始の指針の主頁の下部で示されます。

各使用の場合について、詳細が最初に示されます。その後にその使用事例での構成設定方法(″Setup steps″)とﾄﾞﾗｲﾊﾞ使用法(″Usage steps″)の2つの項目に従ってください。これらの項目は以下の副項目から成ります。

1. 事前必要条件″Prerequisites″(任意選択)：使用事例に対する事前必要条件の一覧 － 通常、″Setup steps″でだけ必要とされます。使用事例は例えばﾌﾟﾛｼﾞｪｸﾄに手動で追加されなければならない2つ目のﾄﾞﾗｲﾊﾞに依存するかもしれません。例えば、ADCに基づく割り込み使用事例は割り込み管理ﾄﾞﾗｲﾊﾞが必要です。
2. ｺｰﾄﾞ例″Example code″：使用事例で各々のﾄﾞﾗｲﾊﾞを構成設定して使用するのに必要とされる完全なｺｰﾄﾞ。事前必要条件が適切である限り、ｺｰﾄﾞ例はﾎﾞｯｸｽによって動くべきです。
3. 作業の流れ″Workflow″：各々の段階が実行のための複写または活動のためのｺｰﾄﾞの断片を特徴とする、番号付けされた一覧として表される、ｺｰﾄﾞを構築するのに従う段階。どの関連したｺｰﾄﾞも指針を用意に読ませるために段階内で繰り返されます。段階は追加の注目点や注意点、換言すると、使用者が知るのに重要な詳細や説明も持つかもしれません。

6.5.1. 例

以下の副項は即時開始の指針を定義するためのdoxygenｺｰﾄﾞ例を示します。
″Setup steps″と″Usage steps″の項目が同じ方法で定義されるため、それを定義するためのｺｰﾄﾞは前の項目に対してだけ示され、他の項目に対しては単純にｺｰﾄﾞを複写してsetupをusageに置き換えてください。

6.5.1.1. API資料内の即時開始の指針へのﾘﾝｸ

```
/**
 * ¥defgroup some_group 或る単位部(SM)
 *
 * ¥ref some_quickstart をご覧ください。
 *
 * これは或る単位部用のﾄﾞﾗｲﾊﾞです。それは～用の関数を提供します。
～
```

6.5.1.2. 即時開始の指針主頁定義

```
/**
 * ¥page some_quickstart 或る単位部ﾄﾞﾗｲﾊﾞ用即時開始の指針
 *
 * これは使用事例の選択内でﾄﾞﾗｲﾊﾞを形態設定して使用する方法の段階的な指示を持つ、
 * ¥ref some_group ″或る単位部ﾄﾞﾗｲﾊﾞ″用の即時開始の指針です。
 *
 * 使用事例は様々なｺｰﾄﾞ断片を含みます。構成設定用の段階内のｺｰﾄﾞ断片は独自初期化関数内に複写することができ、
 * 一方使用法用の段階は例えば主応用関数内に複写することができます。
 *
～
```

6.5.1.3. 基本使用事例表現

```
 * ¥section some_basic_use_case 基本使用事例
 * この基本使用事例では以下のためにSMが形態設定されます。
 * - 或る形態設定詳細
 * - 2つ目の形態設定詳細
 *
～
```

6.5.1.4. 使用事例段階用項目開始

```
 * ¥section some_basic_use_case_setup 構成設定段階
～
```

6.5.1.5. 使用事例事前必要条件表現(任意選択)

```
 * ¥subsection some_basic_use_case_setup_prereq 事前必要条件
 * 動くためのこの使用事例の構成設定ｺｰﾄﾞについて、以下が行われなければなりません。
 * ﾌﾟﾛｼﾞｪｸﾄへ追加:
 * -# 別の単位部(AM)ﾄﾞﾗｲﾊﾞ
～
```

6.5.1.6. 使用事例ｺｰﾄﾞ表現

```
 * ¥subsection some_basic_use_case_setup_code ｺｰﾄﾞ例
 * conf_sm.hの内容:
 * ¥code
 * #define CONFIG_SM_SOME_FEATURE
 * ¥endcode
 *
 * 応用Cﾌｧｲﾙへ追加:
 * ¥code
 * void some_init(void)
 * {
 *     do_one_thing();
 *     do_something_else();
 *     ～
 * }
 * ¥endcode
～
```

6.5.1.7. 使用事例作業の流れ表現

```
 * ¥subsection some_basic_use_case_setup_flow 作業の流れ
 * -# conf_sm.hが存在し、或る機能に対して形態設定ｼﾝﾎﾞﾙを含むことを保証してください。:
 *    - ¥code #define CONFIG_SM_SOME_FEATURE ¥endcode
 *    - ¥note この形態設定ﾌｧｲﾙはﾄﾞﾗｲﾊﾞによって使用され、使用者によってｲﾝｸﾙｰﾄﾞされるべきではありません。
 * -# 或る機能許可:
 *    - ¥code do_one_thing(); ¥endcode
 *    - ¥note これは使用者が手助けや興味を探すかもしれない詳細です。
 * -# 他のことをする:
 *    - ¥code do_something_else();
 * ～ ¥endcode
 *    - ¥attention これは災難や一般的な間違いを避けるために使用者が注意すべき詳細です。
～
```

6.5.1.8. 高度な使用事例一覧

```
 * ¥section some_use_cases 高度な使用事例
 * SMﾄﾞﾗｲﾊﾞのもっと高度な使用については以下の使用事例をご覧ください。:
 * - ¥subpage some_use_case_1: 強調した機能の一覧
 */
```

6.4.1.9. 高度な使用事例の定義と表現

```
/**
 * ¥page some_use_case_1 使用事例#1
 * この使用事例では以下に対してSMが形態設定されます。:
 * - 最初の詳細
 * - いくつかの他の詳細
～
```

6.6. APIシンボル定義

APIシンボルは使用するには申し分ないのですが、それらにツールチェーンによって提供された値に一致する値を与えてください。

6.6.1. 例

API定義宣言に望まれる方法:

```
#define OSC_MODE_EXTERNAL       AVR32_PM_OSCCTRL0_MODE_EXT_CLOCK
```

可能な時に避ける、API定義宣言に有効でない方法:

```
#define OSC_MODE_EXTERNAL       0x1234567
```

6.7. ハードウェア ドライバ クロック管理

クロック単位部は初期化中に殆どの重要でない部署へのクロックを禁止します。例えば、これはATMEL AVR XMEGAデバイスで電力削減(PR)ビットが全て設定(1)されることを意味します。従ってドライバはこれらをインターフェースする前に、これらのハードウェア部署がクロック駆動されることを保証しなければなりません。

ドライバはドライバ単位部に対する周辺クロックの許可と禁止のためにクロック サービス(clock service)を使用することが望まれます。クロック サービスを通して行うことが可能でない単位部では、勿論ハードウェア レジスタで直接的に動くことは結構です。

クロック サービスは実装と資料の両方がcommon¥services¥clockディレクトリで利用可能です。重要性のある部署関数はsysclk_が前に置かれます。

6.8. ハードウェア ドライバ休止管理

全てのハードウェア ドライバは現在の活動に対してどの休止レベルが適切かについて休止管理部(sleep manager)を更新すべきです。施錠(lock)された休止動作形態の各々がそれに対応する開錠(unlock)呼び出しを持つことが重要です。休止管理部についてのより多くの情報に関してはcommom¥services¥sleepmgr¥ディレクトリをご覧ください。また、部署関数はsleepmgr_が前に置かれます。

6.9. 形態設定ヘッダ ファイル

コンパイル時形態設定可能な単位部について、形態設定は専用のヘッダ ファイルから訂正されなければなりません。単位部用のヘッダ ファイルはconf_<単位部>.hと名付けられます。

利用可能な全ての任意選択を持つ雛形形態設定ヘッダ ファイルは、それが支援するデバイスのどれかに対する雛形プロジェクトへ追加される場合に単位部が構築を中断しないように規定値で満たされ、利用可能にされなければなりません。

6.10. ハードウェア ドライバ割り込みレベル

ドライバ内で使用される割り込み処理部は可能な程度まで形態設定可能な割り込みレベルを持つべきです。割り込みレベルの形態設定は単位部の形態設定ファイル経由で行われるべきです。他の形態設定任意選択とのため、ドライバは何も提供されない場合に健全な既定値を定義しなければなりません。

形態設定シンボルはCONFIG_<部署>_INTLVL形式であるべきです。もっと特定の割り込み形態設定が必要とされる場合、それらはCONFIG_<部署>_<供給元>_INTLVLのようであるべきです。

6.10.1. 例

```
CONFIG_DMA_INTLVL               PMIC_LVL_LOW
CONFIG_USART_DRE_INTLVL         PMIC_LVL_MEDIUM
CONFIG_USART_RXC_INTLVL         PMIC_LVL_HIGH
```

7. 目次

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131
USA
TEL (+1)(408) 441-0311
FAX (+1)(408) 487-2600
www.atmel.com

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

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

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

本応用記述はATMELのAVR4030応用記述(doc8432.pdf Rev.8432B-03/12)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。