AVR4016 : Sensors Xplainedｿﾌﾄｳｪｱ使用者の手引き

要点

- 感知器ﾃﾞﾊﾞｲｽ用ﾊｰﾄﾞｳｪｱ無関係なC言語ｲﾝﾀｰﾌｪｰｽ
- 全ての測定に対する標準単位への変換
- 感知器形式の多様性に対するﾄﾞﾗｲﾊﾞ
- 使用が簡単な形態設定と初期化

1. 序説

この応用記述はATMEL® AVR®ｿﾌﾄｳｪｱ枠組み(ASF:AVR Software Framework)での共通感知器ｻｰﾋﾞｽを紹介します。Sensors Xplainedｿﾌﾄｳｪｱは高位C/C++応用ﾌﾟﾛｸﾞﾗﾐﾝｸﾞ ｲﾝﾀｰﾌｪｰｽ(API)と、ATMELの8ﾋﾞｯﾄと32ﾋﾞｯﾄのAVR XMEGA®とAVR UC3ﾏｲｸﾛ ｺﾝﾄﾛｰﾗ周辺で構築するｼｽﾃﾑでの感知器装置用の2進ﾄﾞﾗｲﾊﾞ ﾗｲﾌﾞﾗﾘから成ります。ASF基板はXplain感知器基板上の感知器の各種組み合わせとAVRﾏｲｸﾛ ｺﾝﾄﾛｰﾗの組み合わせを開発者に許す形態設定定数と実行時初期化呼び出しを含む、ATMEL AVR Xplain MCU評価ｷｯﾄとSensors Xplained付加基板(″上乗せ単位部″)に対する単位部を支援し、応用ｿｰｽ ｺｰﾄﾞへの少しの変更または全く変更なしで自立応用を改めて目的対象とします。SWensors Xplainedｿﾌﾄｳｪｱとで含まれる実演ﾌﾟﾛｼﾞｪｸﾄは自立応用を構築するために感知器API、ﾗｲﾌﾞﾗﾘ、基板支援単位部、ASFﾄﾞﾗｲﾊﾞ、それと形態設定定数と共に導入する方法を例証します。

図1-1. 例のXplain感知器付加基板

2. 概要

ATMEL Sensors XplainedｿﾌﾄｳｪｱはATMEL AVRｿﾌﾄｳｪｱ枠組み(ASF) 2.5またはそれ以降版への共通ｻｰﾋﾞｽ拡張として実装されます。感知器ｿﾌﾄｳｪｱは高位可搬C/C++ API、感知器ﾄﾞﾗｲﾊﾞを含む2進ﾗｲﾌﾞﾗﾘ、感知器形態設定機構、感知器API呼び出しを説明する応用例を含みます。応用は特定の感知器装置に対して詳述するどんなｺｰﾄﾞも必要ありません。代わりに、応用は装置と無関係な規則で感知器と相互作用して、少しの基本的な形態設定定数を用いる感知器とﾏｲｸﾛ ｺﾝﾄﾛｰﾗの各種組み合わせへ再目標化して適切なﾄﾞﾗｲﾊﾞ ﾗｲﾌﾞﾗﾘに対してﾘﾝｸすることができます。

2.1. 共通感知器ｻｰﾋﾞｽ

Sensors Xplained API部分はcommon¥services¥sensorsﾃﾞｨﾚｸﾄﾘでASF 2.x木構造内の共用ｻｰﾋﾞｽとしてｲﾝｽﾄｰﾙされ、APIとﾄﾞﾗｲﾊﾞ ﾍｯﾀﾞ ﾌｧｲﾙ、ASFｻｰﾋﾞｽと目標基板支援単位部用形態設定ﾍｯﾀﾞ ﾌｧｲﾙ、高位ﾕｰﾃｨﾘﾃｨ関数を含みます。

感知器装置ﾄﾞﾗｲﾊﾞとSensors Xplained API内のｲﾝﾀｰﾌｪｰｽ ﾙｰﾁﾝはｿｰｽ ｺｰﾄﾞなしのﾘﾝｸ可能な2進単位部としてだけ配給されます。このAPIのために書かれた応用はthirdparty¥sensors¥libsﾃﾞｨﾚｸﾄﾘ下のﾂｰﾙﾁｪｰﾝ特定副ﾃﾞｨﾚｸﾄﾘで見つけた適切なﾗｲﾌﾞﾗﾘ格納部に備えてﾘﾝｸしなければなりません。加えて、感知器ﾊｰﾄﾞｳｪｱと基盤ｲﾝﾀｰﾌｪｰｽは条件付きｺﾝﾊﾟｲﾙされたASFｻｰﾋﾞｽと基盤ｲﾝﾀｰﾌｪｰｽを広範囲に使用します。結果として、Sensors Xplainedﾗｲﾌﾞﾗﾘを使用する応用を構築する時に追加ASFｿｰｽ ﾌｧｲﾙが必要とされます。これらの依存性はATMEL AVR Studio® 5 ﾌﾟﾛｼﾞｪｸﾄ機能によって管理されます。

ATMEL
マイクロ コントローラ

応用記述

Rev. 8367B-06/11, 8367BJ1-12/13

2.1.1. Sensors Xplained API単位部

common¥services¥sensorsﾃﾞｨﾚｸﾄﾘはSensors Xplained応用ﾌﾟﾛｸﾞﾗﾐﾝｸﾞ ｲﾝﾀｰﾌｪｰｽ(API)を定義するﾍｯﾀﾞ ﾌｧｲﾙとC言語実装ﾌｧｲﾙを含みます。これらは感知器形態設定機構の一部である(以下で記述される)sensor.hﾌｧｲﾙとsensor_platform.cｿｰｽ ﾌｧｲﾙを含みます。感知器APIﾊｰﾄﾞｳｪｱ抽象層(HAL:Hardware Abstraction Layer)は、AVRｿﾌﾄｳｪｱ枠組みﾄﾞﾗｲﾊﾞ、感知器ﾄﾞﾗｲﾊﾞ、目的対象基板の基盤、感知器API間の移転層として働きます。これらの静的な形態設定単位部は様々なﾄﾞﾗｲﾊﾞと基板ｲﾝﾀｰﾌｪｰｽへのｱｸｾｽを提供します。sensor_platform_init()ﾙｰﾁﾝは感知器APIによる使用のために目的対象Xplain評価基板とSensors Xplained基板を初期化する応用による主な走行時機構です。あなたの応用は系初期化中の最初の段階としてsensor_platform_init()を呼ぶべきです。それは他のASFに基づく応用で代表的に作られたboard_init()呼び出しを置き換えます。

common¥services¥sensorsﾃﾞｨﾚｸﾄﾘは他の様々なﾍｯﾀﾞとｿｰｽのﾌｧｲﾙも含みます。これらは感知器ｻｰﾋﾞｽ内で単位部によって使用されます。それらのﾌｧｲﾙをあなたの応用または内部定義に応じて直接的に含めるべきではなく、それらは将来の公開に於いて変更を受けます。

2.1.2. sensor.hﾍｯﾀﾞ ﾌｧｲﾙ

common¥services¥sensorsﾃﾞｨﾚｸﾄﾘはsensor.hﾍｯﾀﾞ ﾌｧｲﾙを含み、これは共通感知器ｻｰﾋﾞｽを使用するのに必要とされる定義を含む主ﾌｧｲﾙです。これらの定義は全てのAPIｲﾝﾀｰﾌｪｰｽに関する関数原型、ﾃﾞｰﾀ構造定義、様々な定数を含みます。sensor.hﾌｧｲﾙは感知器ｲﾝﾀｰﾌｪｰｽを使用するどの応用ｺｰﾄﾞでもｲﾝｸﾙｰﾄﾞされなければなりません。

あなたの応用を作成するのにATMEL AVR Studio 5ﾌﾟﾛｼﾞｪｸﾄ ツールを使用する場合、独自応用を追加するために例応用からか、または感知器部品を選択することのどちらかで、avr.hﾌｧｲﾙはsensor.hをｲﾝｸﾙｰﾄﾞするように自動的に変更されます。この場合、あなたの応用は直接sensor.hをｲﾝｸﾙｰﾄﾞする必要はなく、それは何時ものように単にasf.hをｲﾝｸﾙｰﾄﾞすることができ、そして必要な定義が利用可能になります。

注: この資料の後ろで見つかるｺｰﾄﾞの流れの例はsensor.hを明示的に参照します。sensor.hを含むasf.hの自動生成を使用するのなら、代わりに単純にasf.hをｲﾝｸﾙｰﾄﾞすることができます。

2.1.3. Sensors Xplained module_configﾃﾞｨﾚｸﾄﾘ

common¥services¥sensors¥module_config内に置かれたC言語ﾍｯﾀﾞ ﾌｧｲﾙはSensors Xplainedｿﾌﾄｳｪｱでの使用のための適切な設定を含むASF形態設定ﾌｧｲﾙの参照基準版を提供します。

あなたの応用がAVR Studio 5ﾌﾟﾛｼﾞｪｸﾄ機能で提供された感知器応用例に基づくなら、それらの形態設定の設定はあなたの応用の構築で自動的にｲﾝｸﾙｰﾄﾞされ、あなたの応用内のconfigﾃﾞｨﾚｸﾄﾘに現れるでしょう。

AVR Studio 5のﾌﾟﾛｼﾞｪｸﾄ機能を使用して既存または独自の応用に感知器支援を追加する場合、common¥services¥sensors¥module_config内の対応するﾌｧｲﾙ内で見つかる設定に合うように、(あなたのﾌﾟﾛｼﾞｪｸﾄ内のconfigﾃﾞｨﾚｸﾄﾘに置かれた)様々な形態設定ﾌｧｲﾙ内の初期設定を変更することが必要になるかもしれません。

2.1.4. Sensors Xplainedﾄﾞﾗｲﾊﾞ ﾃﾞｨﾚｸﾄﾘ

common¥services¥sensors¥driver内に置かれたﾌｧｲﾙはSensors Xplainedﾊｰﾄﾞｳｪｱ抽象層によって必要とされる定義を供給します。感知器応用はこれらのﾌｧｲﾙ内の定義が必要ではなく、それらのﾌｧｲﾙをｲﾝｸﾙｰﾄﾞすべきではなく、またはあなたの応用内で定義するそれらのｼﾝﾎﾞﾙのどれをも参照すべきではありません。sensor.hﾌｧｲﾙで指定された定義とAPIﾙｰﾁﾝはｲﾝｽﾄｰﾙされた全ての感知器周辺機能へのｱｸｾｽを提供します。ATMELまたは第三者の感知器ﾄﾞﾗｲﾊﾞ実装はそのどれからもｿｰｽ ｺｰﾄﾞで提供されず、このﾃﾞｨﾚｸﾄﾘ内の全てのﾌｧｲﾙは共通感知器ｻｰﾋﾞｽの将来の版での変更を受けます。ﾃﾞｨﾚｸﾄﾘそれ自身は感知器ｻｰﾋﾞｽによって使用するために新しい感知器ﾄﾞﾗｲﾊﾞを書くそれらの開発者のために木構造に於いて維持されます。

2.1.5. Sensors Xplainedﾄﾞﾗｲﾊﾞ ﾗｲﾌﾞﾗﾘ

感知器ﾄﾞﾗｲﾊﾞとAPI関数はGCCとIAR™ Systemsのﾂｰﾙﾁｪｰﾝ用に構築された静的ﾘﾝｸ ﾗｲﾌﾞﾗﾘからあなたの応用内へﾘﾝｸされなければなりません。ﾗｲﾌﾞﾗﾘは各々、thirdparty¥sensors¥libs¥gccとthirdparty¥sensors¥libs¥iarのﾃﾞｨﾚｸﾄﾘに置かれます。必要な単位部だけがあなたの最終ｼｽﾃﾑ ｲﾒｰｼﾞ内にﾘﾝｸされます。

あなたの応用ﾌﾟﾛｼﾞｪｸﾄを作成するのにAVR Studio 5のﾌﾟﾛｼﾞｪｸﾄ機能を使用する時に、あなたの構築で正しいﾗｲﾌﾞﾗﾘが自動的にｲﾝｸﾙｰﾄﾞされます。

注: ﾗｲﾌﾞﾗﾘで見つかる感知器ﾄﾞﾗｲﾊﾞとAPI関数は2進形式でだけ利用可能です。ｿｰｽ ｺｰﾄﾞは全く提供されません。

ﾗｲﾌﾞﾗﾘ名は支援される目的対象AVRﾏｲｸﾛ ｺﾝﾄﾛｰﾗを示すだけでなく、何れにせよﾗｲﾌﾞﾗﾘはﾃﾞﾊﾞｯｸﾞ目的に使用される構築を目標とする特別なﾌﾗｸﾞと共に構築されます。

thirdparty¥sensors¥libs¥gccﾃﾞｨﾚｸﾄﾘに置かれるGCCﾄﾞﾗｲﾊﾞ ﾗｲﾌﾞﾗﾘは以下の名前の形式を持ちます。

libsensors-$mcu_series-debug.a
libsensors-$mcu_series-release.a

thirdparty¥sensors¥libs¥iarﾃﾞｨﾚｸﾄﾘに置かれるIARﾘﾝｸ ﾗｲﾌﾞﾗﾘは以下の同様の形式を持ちます。

libsensors-$mcu_series-debug.r82
libsensors-$mcu_series-release.r82

両方の場合で、$mcu_seriesは使用される特定の8ﾋﾞｯﾄまたは32ﾋﾞｯﾄのAVRﾏｲｸﾛ ｺﾝﾄﾛｰﾗ型式を識別します。表2-1.は現在利用可能な感知器ﾄﾞﾗｲﾊﾞ ﾗｲﾌﾞﾗﾘを一覧にします。

表2-1. Sensors Xplainedﾗｲﾌﾞﾗﾘ

ﾗｲﾌﾞﾗﾘ名	目的対象MCU	ﾂｰﾙﾁｪｰﾝ
libsensors-at32uc3a3-debug.a	AVR32 UC3-A3	GCC
libsensors-at32uc3a3-release.a	AVR32 UC3-A3	GCC
libsensors-at32uc3a3-debug.r82	AVR32 UC3-A3	IAR
libsensors-at32uc3a3-release.r82	AVR32 UC3-A3	IAR
libsensors-at32uc3a-debug.a	AVR32 UC3-A	GCC
libsensors-at32uc3a-release.a	AVR32 UC3-A	GCC
libsensors-at32uc3a-debug.r82	AVR32 UC3-A	IAR
libsensors-at32uc3a-release.r82	AVR32 UC3-A	IAR
libsensors-at32uc3b-debug.a	AVR32 UC3-B	GCC
libsensors-at32uc3b-release.a	AVR32 UC3-B	GCC
libsensors-at32uc3b-debug.r82	AVR32 UC3-B	IAR
libsensors-at32uc3b-release.r82	AVR32 UC3-B	IAR
libsensors-at32uc3c-debug.a	AVR32 UC3-C	GCC
libsensors-at32uc3c-release.a	AVR32 UC3-C	GCC
libsensors-at32uc3c-debug.r82	AVR32 UC3-C	IAR
libsensors-at32uc3c-release.r82	AVR32 UC3-C	IAR
libsensors-at32uc3l-debug.a	AVR32 UC3-L	GCC
libsensors-at32uc3l-release.a	AVR32 UC3-L	GCC
libsensors-at32uc3l-debug.r82	AVR32 UC3-L	IAR
libsensors-at32uc3l-release.r82	AVR32 UC3-L	IAR

2.2. Sensors Xplained目的対象基板

感知器ｻｰﾋﾞｽAPIﾍｯﾀﾞ、ｿｰｽ、ﾗｲﾌﾞﾗﾘ ﾌｧｲﾙに加えて、全てのATMEL Sensors Xplained応用は目的対象基板支援ｿｰｽ ﾌｧｲﾙと基板特有形態設定ﾌｧｲﾙが必要です。AVRの評価と開発のための基板用の基板支援ﾌｧｲﾙはASF木構造内のavr32¥boardsとxmega¥boardsの副ﾃﾞｨﾚｸﾄﾘに置かれます。ATMEL Sensors Xplained拡張基板用の共通基板支援ﾌｧｲﾙはcommon¥boards¥sensor_xplainedﾃﾞｨﾚｸﾄﾘに置かれます。

それらのﾃﾞｨﾚｸﾄﾘの各々内で定義された個別ﾍｯﾀﾞ ﾌｧｲﾙをｲﾝｸﾙｰﾄﾞするよりも、むしろ応用と基板ｲﾝﾀｰﾌｪｰｽ ｿﾌﾄｳｪｱは単にcommon ¥boards¥board.hﾌｧｲﾙをｲﾝｸﾙｰﾄﾞすべきです。このﾌｧｲﾙは全てのﾌﾟﾛｾｯｻ型式間で共用され、以降の章で検討されるように特定の形態設定定数の値に基づく基板特有定義を公開します。

3. 必要条件

以下はSensors Xplainedｿﾌﾄｳｪｱを使用して自立応用を作成するための最小必要条件です。

- ATMEL AVR Studio 5 (http://www.atmel.com/dyn/products/tools_card.asp?tool_id=17212)
- 32ﾋﾞｯﾄAVRﾂｰﾙﾁｪｰﾝ用ﾍｯﾀﾞ ﾌｧｲﾙ更新を含む、ATMEL AVRｿﾌﾄｳｪｱ枠組み 2.5またはそれ以降の版
- 支援されるATMEL UC3 Xplain評価基板
- 支援されるATML Sensors Xplained拡張基板
- 任意選択: ATMEL AVR32用IAR Embedded Workbench® 3.31またはそれ以降版 (http://www.iar.com)
- 上のﾂｰﾙ(例えば、ATMEL AVR JTAGICE3やATMEL AVRONE!)によって支援されるﾊｰﾄﾞｳｪｱ書き込み器

AVR Studio 5ｿﾌﾄｳｪｱはATMELのｳｪﾌﾞｻｲﾄで入手可能です。開発環境とﾂｰﾙﾁｪｰﾝをｲﾝｽﾄｰﾙするには、それに伴うｲﾝｽﾄｰﾙ指示と応用記述に従ってください。

感知器ﾄﾞﾗｲﾊﾞとAPI関数を含むGCCとIAR Systemの静的ﾘﾝｸ ﾗｲﾌﾞﾗﾘと共に、共通感知器ｻｰﾋﾞｽC/C++ｿｰｽとﾍｯﾀﾞのﾌｧｲﾙは標準AVR Studio 5/ASFｲﾝｽﾄｰﾙに含まれますが、あなたのﾌﾟﾛｼﾞｪｸﾄを作成して形態設定する時に応用内で明示的にｲﾝｸﾙｰﾄﾞされなければなりません。

4. 応用作成

ATMEL AVR Studio 5/ASFｲﾝｽﾄｰﾙは感知器装置を制御して測定ﾃﾞｰﾀを得るためのATMEL Sensors Xplained APIの使用方法を説明する多数の応用例を含みます。ASF下の副ﾃﾞｨﾚｸﾄﾘであるcommon¥applications¥sensorsﾃﾞｨﾚｸﾄﾘに置かれたこれらの応用は感知器APIを使用する応用がXplainﾌﾟﾛｾｯｻ基板とSensors Xplained付加拡張基板の様々な組み合わせに対してどう形態設定されて構築され得るかを説明します。新しい感知器API応用は雛形として実演応用を使用する、または標準的な応用で開始して感知器ｻｰﾋﾞｽと基板支援単位部を追加することによって作成することができます。

4.1. Sensors Xplained応用例

感知器インターフェースがどう使用されるかを説明するためにSensors Xplainedソフトウェアで多数の応用例が含まれます。これらの応用の全てがcommon¥applications¥sensorsディレクトリで見つけられ、同じ基本構築機構と先に記述された基板定義を使用します。他の応用例はあなたの特定のインストールで利用可能かもしれません。

- 慣性感知器実演(inertial_demo)

  この簡単な応用は加速、回転、磁針方位、温度を含む慣性感知器基板からのデータを得ます。データは端末プログラムを使用して表示するために、USB接続経由で接続されたホストPCへ送られます。

- 感知器データ可視器(inertial_visualizer)

  この応用も慣性感知器基板からデータを得ます。データは特殊なパケットに形式化され、特別なATMELデータ可視器応用を用いて表示するために、USB接続経由で接続されたホストPCへ送られます。より多くの情報については「AVR4017:ATMELデータ可視器」応用記述をご覧ください。

- 慣性感知器起き上がり実演(wake_demo)

  この応用は感知器事象が起きた時に低電力休止形態から系を起き上がらせるための感知器処理機構の使い方を実演します。事象は加速度計を使用する運動閾値検出、または回転儀(ジャイロスコープ)からの新しいデータ事象のどちらかで有り得ます。

- 羅針盤感知器校正(compass_calibration)

  この応用は羅針盤(コンパス)/磁力計装置に対する基本的な手動校正手順を実演します。

- 圧力感知器実演(pressure_demo)

  この簡単な応用は圧力感知器基板から大気圧と温度のデータを得ます。データは端末プログラムを使用して表示するために、USB接続経由で接続されたホストPCへ送られます。

4.1.1. 応用例の構築

感知器応用例が選択され、そしてAVR Studio内での他のASF応用と同じ方法で構築してください。以下の段階は、加速度計、回転儀、羅針盤の装置を使用する、慣性感知器実演プロジェクトに基づく新しいプロジェクト作成方法を要約します。

1. AVR Studio 5メニューでFile⇒New⇒New Project...を選択してください。
2. New Example Projectウィンドウに於いて左側のパネルでTechnologyを選択してください。
3. 感知器応用選択を表示するためにTechnology領域の一覧でSensorsを選択してください。
4. 構築したいプロジェクト例を選択し、あなたのATMEL Xplainプロセッサ基板用の慣性感知器実演プロジェクト(例えば、Inertial Sensor Demonstration - UC3-A3 Xplained - AT32UC3A3256)をクリックしてください。OKをクリックしてください。
5. Sensors Xplainedライブラリに含まれるソフトウェアの使用に関する許諾契約を含むSoftware License Agreementウィンドウが現れます。許諾の条件に同意するなら、"I acceplt the license agreement"の箱上でクリックしてFinishを選んでください。
6. 例プロジェクト ファイルがATMEL AVR Studioウィンドウ内に現れます。プロジェクトに対する既定感知器基板があなたが使用するものなら、今や通常のように構築して応用を書き込む(ダウンロードする)ことができます。感知器基板選択の変更が必要な場合、下をご覧ください。

注: ここで要約された慣性感知器応用を含むATMEL Sensors Xplained応用例の殆どはホスト マシンでの仮想通信ポートへのUSB直列I/O接続が必要です。基板構成設定指示に従ってホスト マシン上で適切なドライバをインストールしてください。既定応用直列I/O形態設定は8ビット データ、パリティなし、1停止ビットを使用する115,200bpsで送信します。

4.1.2. 既定感知器基板の変更

各Sensors Xplainedプロジェクトは定義された既定感知器基板部品を持ちます。けれども、感知器基板選択はあなたが使用する実際のハードウェアに基づいて変更することができます。

現在の感知器基板選択を表示または変更するには、AVR Studio内で以下の手順を使用してください。

1. Solution Explorerペインでプロジェクト名を選択してください。
2. AVR Studio 5のメニューで、Project⇒Select Drivers from ASF...を選択してください。
3. 現在の感知器基板選択は右側ペインのSelected Modules一覧内です(例えば、Sensors Xplained Inertial 1基板について、"Sensors - ATAVRSBIN1 Sensor Board (Componebt)"に対する入口になります)。
4. 感知器基板を変更するには、
   - Selected Modulesペインで現在の感知器基板部品を選択してその後にRemove from selectionをクリックしてください。
   - 左側のAvailable Modulesペインで使用を望む新しい感知器基板を見つけて選択し、その後にAdd to selectionをクリックしてください。
   - Finishをクリックしてください。
5. 今や通常の様にあなたのプロジェクトを構築して書き込む(ダウンロードする)ことができます。

4.2. 既存応用への感知器追加

共通感知器サービスはAVR Studioプロジェクト機能を使用することによって些細な変更だけでどの応用にも追加することができます。本項での段階は、あなたが使用するプロセッサ基板に対して適切な使用応用雛形を使用して最初に作成された代表的な応用に関して手順全体を記述します。

最初に、以下の手順を用いてあなたの応用内に感知器支援単位部を追加しなければなりません。

1. Solution Explorerペインでプロジェクト名を選択してください。
2. AVR Studio 5のメニューでProject⇒Select Drivers from ASF...を選択してください。
3. 基本感知器サービスを追加するには、左側のAvailable Modulesペインで Sensors – Sensor Device Stackを選択し、その後にAdd to selectionをクリックしてください。
4. あなたが使用する感知器基板に対する支援を追加するには、左側のAvailable Modulesペインで使用を望む新しい感知器基板を見つけて選択し、その後にAdd to selectionをクリックしてください。
   - 例えば、ATMEL Sensors Xplained inertial 1基板に対する支援を追加するには、Sensors – ATAVRSB1N1 Sensor Board (Component)を選択してください。
5. Finishをクリックしてください。

感知器サービスを初期化するためにあなたの応用も変更しなければなりません。そのようにするには、あなたの応用のmain.cファイルで以下の変更を行ってください。

以下の呼び出し、

```
board_init();
```

を

```
sensor_platform_init();
```

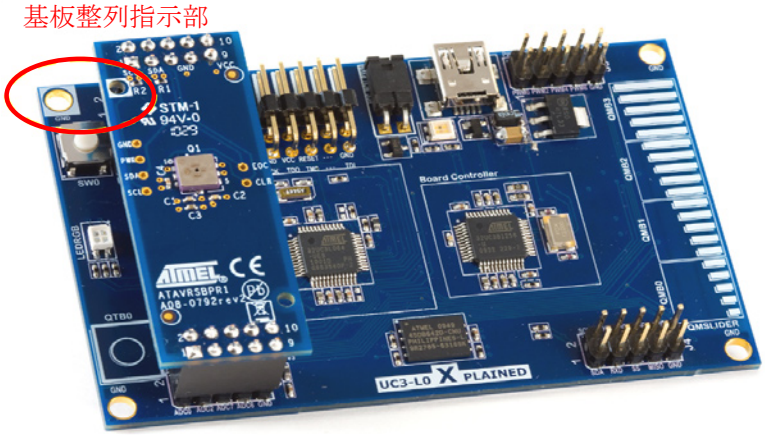
で置き換えてください。

一旦これらの変更が行われると、あなたの応用に通常のSensors Xplained API関数呼び出しの追加を続行することができます。最初にsensor_attach()を使用して1つ以上の感知器装置を初期化することが必要でしょう。その後に6頁の第6.章と7頁の第7.章で記述されるインターフェースと手続きを用いて制御操作とデータ読み込みを実行することができます。

4.3. 感知器基板の付着

図4-1.はSensors Xplained感知器基板がXplainプロセッサ基板にどう付着するかを示します。両基板の装着穴周辺の整列指示部に注意してください。

図4-1. UC3-L0 Xplain評価基板への付着



5. 初期化

5.1. 形態設定

先に記述したように、common¥services¥sensors¥module_configディレクトリはATMEL Sensors Xplained応用プロジェクト例によって使用される様々な形態設定ファイルを含みます。これらはSensors Xplained関数で使用されるための設定が推奨されますが、それらはあなたの応用の必要条件または他のATMEL AVRソフトウェア枠組み(ASF)依存性に基づいて変更されるかもしれません。

感知器インターフェースを使用する、ATMEL AVR Studio 5からの応用例を構築する時に、適切な形態設定ファイルがあなたの応用プロジェクトで自動的にインクルードされ、あなたのプロジェクトのconfigディレクトリ内に現れます。

プロジェクト機能を使用して既存または独自の応用に感知器支援を追加する場合、common¥services¥sensors¥module_config内の対応するファイルで見つかる設定に合うように、(あなたのプロジェクト内のconfigディレクトリ内に置かれた)様々な形態設定ファイル内の初期設定を変更する必要があるかもしれません。

5.2. 感知器初期化

感知器を初期化して後続する使用に対して利用可能にするために、あなたの応用によってsensor_attach()関数が使用されます。あなたの応用は単純に必要とされる感知器の形式を指定して初期化されるべき記述子構造体を提供します。感知器記述子はその後に後で感知器を識別するための関数呼び出し中に使用されます。

sensor_attach()関数の使い方については7頁の第7.章で例ｺｰﾄﾞの流れをご覧ください。

6. 制御ｲﾝﾀｰﾌｪｰｽ

ATMEL Sensors Xplainedｿﾌﾄｳｪｱは可能な時は必ず、装置に無関係で感知器に対して制御と測定の両操作を支援します。本章はあなたの応用に感知器装置の動きの変更を許す様々な感知器制御ｲﾝﾀｰﾌｪｰｽを記述します。

6.1. 感知器の範囲

感知器装置は、測定されつつある物理的な状態のﾚﾍﾞﾙに合わせることを装置の利用可能な出力分解能に許す、多数の測定範囲を度々提供します。装置の感度は装置の尺度一杯の出力範囲が実際に入力ﾚﾍﾞﾙ範囲に対応するように変更されます。従って、与えられた入力ﾚﾍﾞﾙに対する装置からの"生"出力値は範囲設定に基づいて変化するでしょう。

Sensors Xplained感知器ﾗｲﾌﾞﾗﾘ関数は装置の範囲が変更された時にそれらの出力尺度を自動的に調整し、故にあなたの応用へ返される尺度調整された数値は同じで、装置分解能の制限に左右されます。

sensor_set_range()関数はあなたの応用の実行中に感知器範囲を動的に変更するのに使用することができます。この関数は以下の形式を取ります。

```
sensor_set_range (&device, range);
```

ここでのdeviceは装置の装置記述子、rangeは使用されるべき範囲です。range値は装置からの尺度調整された出力が標準に対して使用されるのと同じ単位で表されます(例えば、加速度計に対するmgや圧力感知器に対するPa(ﾊﾟｽｶﾙ))。range値は0から(正または負の)尺度一杯までの測定単位数です。例えば、加速度計の範囲を-2000～+2000mgを網羅するように設定するには、range値が2000です。

rangeに対して指定される値は装置に関して有効な設定に一致しなければならず、さもなければ異常(SENSOR_ERR_PARAMS)が示されます。

有効な範囲設定、既定値などのより多くの情報については15頁の第9.章での個別ﾄﾞﾗｲﾊﾞ記述をご覧ください。

6.2. 採取帯域

感知器装置は一般的に多数の採取周波数または帯域を提供します。これらの各種設定は測定時間と読み取りの安定性(雑音ﾚﾍﾞﾙ)間の二律背反に渡る制御を許します。より短い測定期間(より高い採取周波数)は各測定を得るのに必要とされる時間と電力を減らしますが、測定値は出力値内での"雑音"として現れるより高い変わり易さを示します。

sensor_set_bandwidth()関数はあなたの応用の実行中に感知器の採取帯域を動的に変更するのに使用することができます。この関数は次の形式を取ります。

```
sensor_set_bandwidth (&device, bandwidth);
```

ここでのdeviceは装置の装置記述子、bandwidthは使用されるHzでの周波数です。

bandwidthに指定される値は装置に関して有効な設定に一致しなければならず、さもなければ異常(SENSOR_ERR_PARAMS)が示されます。

有効な帯域周波数、既定値などのより多くの情報については15頁の第9.章での個別ﾄﾞﾗｲﾊﾞ記述をご覧ください。

6.3. 閾値

いくつかの感知器装置は或る事象の検出に対して測定ﾚﾍﾞﾙを確定するのに使用することができる設定可能な閾値を提供します。代表的な例は、感知器が閾値を超える動きを検出する時に、外部的に見ることが可能な事象が生成される設定を動き検出閾値に許す加速度計装置です。

sensor_set_threshold()関数はそのような能力を持つ装置に対してそれらの閾値を設定するのに使用することができます。

```
sensor_set_threshold (&device, type, value);
```

ここでのdeviceは装置の装置記述子、typeは設定される閾値の形式、そしてvalueは新しい閾値です。valueﾊﾟﾗﾒｰﾀは感知器から尺度調整されたﾃﾞｰﾀを読むのに通常使用されるとの同じ単位で表されます(例えば、加速度計装置に対するmg)。

例として、加速度計装置に対して動き検出閾値を500mg(0.5g)に設定するには以下を使用してください。

```
sensor_set_threshold (&accel_dev, SENSOR_THRESHOLD_MOTION, 500);
```

閾値検出は通常、感知器事象と共に使用されます。感知器事象の生成と使用のより多くの情報については12頁の第8.章をご覧ください。

6.4. 校正

代表的に、感知器は正確な測定を提供するために装置毎の同じ水準の校正が必要です。度々、装置の製造中に必要とされる校正だけが実行され(工場校正)、その校正値が装置に於いて内部的に格納されます。他の場合、実際に配備された状態で装置を校正することが必要です。例えば、羅針盤(ｺﾝﾊﾟｽ)/磁力計の装置は代表的に最終製品内に存在する磁界(基板、筐体、電気的接続)に敏感で、それらは正確な読み取りを達成するために相殺されなければなりません。

感知器が配備された環境で明らかに校正が必要な時に、結果の校正値が度々ﾏｲｸﾛ ｺﾝﾄﾛｰﾗ内の不揮発性ﾒﾓﾘに格納されます。

sensor_calibrarte()関数は感知器に対して初期化して校正手順を実行することをあなたの応用に許します。校正手順は感知器装置を指定します。sensor_calibrarte()関数は多段階校正手順が必要な装置を支援するのに入力ﾊﾟﾗﾒｰﾀとして段階数を取ります(例えば、使用者が装置を物理的に操作しなければならない間での測定の連続)。

校正の必要条件と手順のより多くの情報については15頁の第9.章で個別ﾄﾞﾗｲﾊﾞ記述をご覧ください。

6.5. 自己検査

感知器装置は感知器の動作の物理的や電気的な検査を提供する自己検査機能を度々提供します。これらの検査は一般的に結果の解釈そのものとして非常に装置特有です。

sensor_selftest()関数はあなたの応用から感知器の自己検査機能を発動するための機構を提供します。この関数は(もしあれば)失敗形式を示す特定符号と共に、装置検査からの手短な通過(Pass)または失敗(Fail)の結果を示す値を返します。

sensor_selftest()関数は、特定の戻りﾃﾞｰﾀが装置とそれのﾄﾞﾗｲﾊﾞを示す標準的な規則で呼び出し側へ渡し戻されることを、自己検査からのﾃﾞｰﾀに許します。

利用可能な自己検査のより多くの情報については15頁の第9.章で個別ﾄﾞﾗｲﾊﾞ記述をご覧ください。

7. 感知器ﾃﾞｰﾀ読み取り

7.1. 概要

7.1.1. 感知器読み込みｲﾝﾀｰﾌｪｰｽ

ATMEL Sensors Xplainedｿﾌﾄｳｪｱは感知器装置からﾃﾞｰﾀ得て簡単な使用形式で測定を返すための高位関数の組を提供します。各感知器形式は装置からﾃﾞｰﾀを得るための対応する読み込み関数を持ちます。

表7-1.は各感知器形式に対する感知器読み込み関数の要約を含みます。

表7-1. 感知器読み込み関数要約

感知器形式	Sensors Xplained関数	Sensor_data_t領域	測定単位
全て	sensor_device_id()	device.id device.version	
加速度計(X,Y,Z)	sensor_get_acceleration()	axis.x axis.y axis.z	ﾐﾘ ｸﾞﾗﾑ(mg)
羅針盤(ｺﾝﾊﾟｽ)/磁力計	sensor_get_heading()	heading.direction heading.inclination heading.strength	didrection : 北磁極からの角度(0～360°) inclination : 水平からの傾角(-90～+90°) strength : μT(ﾏｲｸﾛ ﾃｽﾗ)での磁界強度 (注: 1ｶﾞｳｽ(G)=100μT)
	sensor_get_field()	axis.x axis.y axis.z	ﾏｲｸﾛ ﾃｽﾗ(μT)
回転儀(ｼﾞｬｲﾛｽｺｰﾌﾟ)	sensor_get_rotation()	axis.x axis.y axis.z	回転角度/秒(°/s)
圧力	sensor_get_pressure()	pressure.value	ﾊﾟｽｶﾙ(Pa)
温度	sensor_get_temperature()	temperature.value	摂氏温度(℃)

注: この章の最後で見つかるｺｰﾄﾞ手順例は明示的にsensor.hを参照します。sensor.hをｲﾝｸﾙｰﾄﾞする自動生成されたasf.hﾌｧｲﾙを使用する場合、代わりにasf.hを単純にｲﾝｸﾙｰﾄﾞすることができます。

7.1.2. 感知器ﾃﾞｰﾀ構造体 – sensor_data_t

感知器ﾃﾞｰﾀ読み取りを返す全てのAPI関数はそのようなsensor_data_tﾃﾞｰﾀ構造体を使って行います。感知器読み込み関数が戻る時に、この構造体は装置からの測定値だけでなく、高粒度の時刻印も含みます。

sensor_data_t構造体はあなたの応用での使用に関してより意味のある名前を提供するためのﾃﾞｰﾀ領域の"別名"定義にC共用体を使用します。特定の関数に対して推奨される領域名参照については表7-1.をご覧ください。

sensor_data_t構造体は感知器読み込み関数が尺度調整された単位か、または生読み取りかのどちらを返すかを指定するために、あなたの応用によって設定される特別な領域も含みます。この領域は感知器読み込み関数を呼ぶ前に設定されるべきです。

sensor_data_t構造体内の最後の領域はﾏｲｸﾛ秒(μs)で表される、経過時間値を提供する高分解能時刻印値です。この領域は内部ATMEL AVRｼｽﾃﾑ ｸﾛｯｸを使用して各感知器読み取り中に更新されます。

7.1.3. 測定単位

ATMEL Sensors Xplained API関数は実世界(通常科学またはSI)単位で感知器の結果を提供します。これらの値は現在の装置設定に基づいて自動的に尺度調整されます。故に装置に対する出力範囲設定が変更される場合、例えば、尺度調整された出力は(各範囲での装置精度の制限に従属させて)同じに留まるでしょう。

多くの感知器読み取りは装置によって直接的に提供されますが、尺度調整やSI単位への他の変換が必要です。他の結果(例えば、磁針方位)は低位感知器読み取りに基づいてSensors Xplained関数によって計算されます。

感知器読み取りの各形式に使用される測定単位については表7-1.をご覧ください。

7.1.4. ″生″値読み取り

例え一般的に感知器ﾃﾞｰﾀに関して尺度調整された値を得るのが好まれるとは言え、感知器からの内部的な″生″値を読むことも可能です。生の値はｼｽﾃﾑ構成設定や校正、または使用されつつある感知器に対して指定する特別な操作に有用で有り得ます。

感知器装置からの生の値を読むには、感知器の読み込み関数(例えば、sensor_get_acceleration()やsensor_get_pressure())を呼ぶ前にsensor_data_tﾃﾞｰﾀ構造体内のscaled領域をfalseに設定してください。

生の値が返される時にそれらは読み込み関数によって変更されず、故に実際の値は装置に対する範囲設定に依存して変わります。けれども、(羅針盤(ｺﾝﾊﾟｽ)/磁力計の感知器に対して使用されるそれらのような)校正相殺は返される値に適用されます。

7.1.5. 時刻印

sensor_data_t構造体内のtimestamp領域は感知器が読まれる時にAVRﾏｲｸﾛ ｺﾝﾄﾛｰﾗの実時間ｸﾛｯｸからのﾏｲｸﾛ秒(μs)で自動的に満たされます。これらの時刻印は複数感知器読み取りの相対ﾀｲﾐﾝｸﾞを決めるのに使用することができます。

感知器読み込み関数が戻る時に、時刻印はsensor_data_t構造体のtimestamp領域から読むことができます。

7.2. 装置IDと版番号

殆どの感知器装置は感知器型式を判断するために読むことができる識別値を提供します。多くの場合、装置版番号も読むことができます。sensor_device_id()関数は感知器装置からそれらの値を読み、実際の感知器読み取りが返すのと同様の方法でsensor_data_t構造体にそれらを返す特別なﾙｰﾁﾝです。この関数は以下の形式を取ります。

```
sensor_device_id (&device, &id_data);
```

ここでのdeviceは感知器の装置記述子、id_dataはIDと版のﾃﾞｰﾀを受け取るsensor_data_t構造体です。

sensor_device_id()関数が戻る時に、装置ID値はsensor_data_t構造体のdevice.id領域から読むことができます。装置版番号はdevice.version領域から読むことができます。どちらかの値が感知器装置によって提供されない場合、対応する領域は0に設定されます。

7.3. 加速

加速度計感知器は代表的に3軸(X,Y,Z)に沿って直線的な加速力を測定します。sensor_get_acceleration()関数は感知器を読んで測定された加速度を返します。この関数は以下の形式を取ります。

```
sensor_get_acceleration (&device, &accel_data);
```

ここでのdeviceは加速度計の装置記述子、accel_dataは加速度ﾃﾞｰﾀを受け取るsensor_data_t構造体です。

尺度調整された加速度測定はﾐﾘ ｸﾞﾗﾑ(mg)で表されます。関数が戻る時に、この値はaccel_data構造体から″axis″領域(axis.x, axis.y, axis.z)を使用して読むことができます。

7.3.1. ｺｰﾄﾞの流れの例

7.3.1.1. 定義と宣言

```
#include ″sensor.h″
sensor_t          accel_dev;      // 装置記述子
sensor_data_t     accel_data;     // 装置からの加速度ﾃﾞｰﾀ
```

7.3.1.2. 感知器初期化

```
sensor_attach (&accel_dev, SENSOR_TYPE_ACCELEROMETER, 0, 0);
```

7.3.1.3. 感知器読み込み

```
accel_data.scaled = true;          // mgで値読み込み指定
sensor_get_acceleration (&accel_dev, &accel_data);
```

7.3.1.4. 応用でのﾃﾞｰﾀ使用

```
int32_t  app_x_value        = accel_data.axis.x;
int32_t  app_y_value        = accel_data.axis.y;
int32_t  app_z_value        = accel_data.axis.z;
uint32_t app_read_time      = accel_data.timestamp;
```

7.4. 回転

回転儀(ｼﾞｬｲﾛｽｺｰﾌﾟ)は代表的に3軸(X,Y,Z)に沿って回転速度を測定します。sensor_get_rotation()関数は感知器を読んで測定した回転速度を返します。この関数は以下の形式を取ります。

```
sensor_get_rotation (&device, &gyro_data);
```

ここでのdeviceは回転儀の装置記述子、gyro_dataは回転ﾃﾞｰﾀを受け取るsensor_data_t構造体です。

尺度調整された回転速度測定は秒当たりの角度で表されます。関数が戻る時に、この値はgyro_data構造体から"axis"領域(axis.x, axis.y, axis.z)を使用して読むことができます。

7.4.1. ｺｰﾄﾞの流れの例

7.4.1.1. 定義と宣言

```
#include "sensor.h"
sensor_t        gyro_dev;       // 装置記述子
sensor_data_t   gyro_data;      // 装置からの回転ﾃﾞｰﾀ
```

7.4.1.2. 感知器初期化

```
sensor_attach (&gyro_dev, SENSOR_TYPE_GYROSCOPE, 0, 0);
```

7.4.1.3. 感知器読み込み

```
gyro_data.scaled = true;            // 角度/秒で値読み込み指定
sensor_get_rotation (&gyro_dev, &gyro_data);
```

7.4.1.4. 応用でのﾃﾞｰﾀ使用

```
int32_t  app_x_value        = gyro_data.axis.x;
int32_t  app_y_value        = gyro_data.axis.y;
int32_t  app_z_value        = gyro_data.axis.z;
uint32_t app_read_time      = gyro_data.timestamp;
```

7.5. 磁針方位

磁気羅針盤(ｺﾝﾊﾟｽ)の殆どで共通する使い方は向きと誘導の応用で使用するために北磁極に相対する方向方位を得ることです。ATMEL Sensors Xplained APIは装置を読んでそのような方位値を計算するためのsensor_get_heading()関数を提供します。この関数は以下の形式を取ります。

```
sensor_get_heading (&device, &compass_data);
```

ここでのdeviceは羅針盤の装置記述子、compass_dataは方位ﾃﾞｰﾀを受け取るsensor_data_t構造体です。

この関数は"heading"領域を使用してsensor_data_t構造体内で3つのﾃﾞｰﾀ値を返します。

最初の値は北磁極から(時計回りで0～360の)角度で表される方向値です。方向値は装置の正のY軸と測定した磁気ﾍﾞｸﾄﾙの水平方向間の角度です。この値はcompass_data.heading.direction領域から読むことができます。

2つ目の値は水平に相対する(-90～+90の)角度で表される傾斜角です。傾斜角は正の値が下方向(大地への)傾斜を示し、負の値が上方向を示す、伝統的な使い方に従います。傾斜角はcompass_data.heading.inclination領域から読むことができます。

3つ目の値はﾏｲｸﾛ ﾃｽﾗ(μT)で表される正味磁界強度(効力)です。この値はcompass_data.heading.strength領域から読むことができます。これは単一の正味磁場強度値で、各軸(X,Y,Z)方向に対して分離した磁場強度の読み取りを達成するには下で検討されるsensor_get_field()関数を使用してください。

注: 磁気羅針盤感知器は通常、実際の装置配備で存在する局所磁界に対して修正するための校正手順が必要です。この校正なしでは方位情報が正確でないでしょう。手動校正手順の例についてはcompass_calibration例応用をご覧ください。

7.5.1. ｺｰﾄﾞの流れの例

7.5.1.1. 定義と宣言

```
#include "sensor.h"
sensor_t        compass_dev;    // 装置記述子
sensor_data_t   compass_data;   // 装置からの方位ﾃﾞｰﾀ
```

7.5.1.2. 感知器初期化

```
sensor_attach (&compass_dev, SENSOR_TYPE_COMPASS, 0, 0);
```

7.5.1.3. 感知器読み込み

```
compass_data.scaled = true;        // 角度とμTで値読み込み指定
sensor_get_field (&compass_dev, &compass_data);
```

7.5.1.4. 応用でのﾃﾞｰﾀ使用

```
int32_t  app_direction      = compass_data.heading.direction;      // 0～360°
int32_t  app_inclination    = compass_data.heading.inclination;    // -90～+90°
int32_t  app_field_strength = compass_data.heading.strength;       // ﾏｲｸﾛ ﾃｽﾗ(μT)
uint32_t app_read_time      = compass_data.timestamp;
```

7.6. 磁界強度

羅針盤(ｺﾝﾊﾟｽ)方位に加えて、局所磁界強度(効力)の多軸測定を得るのに磁気羅針盤感知器を使用することができます。この関数は以下の形式を取ります。

```
sensor_get_field (&device, &mag_data);
```

ここでのdeviceは羅針盤の装置記述子、mag_dataは磁界強度ﾃﾞｰﾀを受け取るsensor_data_t構造体です。

尺度調整された磁界強度測定はﾏｲｸﾛ ﾃｽﾗ(μT)で表されます。関数が戻る時に、この値はmag_data構造体から"axis"領域(axis.x, axis.y, axis.z)を使用して読むことができます。

注: 磁気羅針盤感知器は通常、実際の装置配備で存在する局所磁界に対して修正するための校正手順が必要です。この校正なしでは磁場強度情報が感知器の外部環境を正確に反映しないでしょう。手動校正手順の例についてはcompass_calibration例応用をご覧ください。

7.6.1. ｺｰﾄﾞの流れの例

7.6.1.1. 定義と宣言

```
#include "sensor.h"
sensor_t        compass_dev;    // 装置記述子
sensor_data_t   mag_data;       // 装置からの磁力ﾃﾞｰﾀ
```

7.6.1.2. 感知器初期化

```
sensor_attach (&compass_dev, SENSOR_TYPE_COMPASS, 0, 0);
```

7.6.1.3. 感知器読み込み

```
mag_data.scaled = true;            // ﾏｲｸﾛ ﾃｽﾗ(μT)で値読み込み指定
sensor_get_field (&compass_dev, &mag_data);
```

7.6.1.4. 応用でのﾃﾞｰﾀ使用

```
int32_t  app_x_value        = mag_data.axis.x;
int32_t  app_y_value        = mag_data.axis.y;
int32_t  app_z_value        = mag_data.axis.z;
uint32_t app_read_time      = mag_data.timestamp;
```

7.7. 大気圧

大気圧は気圧感知器を用いて測定されます。sensor_get_pressure()関数は感知器を読んで測定した圧力を返します。この関数は以下の形式を取ります。

```
sensor_get_pressure (&device, &press_data);
```

ここでのdeviceは圧力感知器の装置記述子、press_dataは圧力ﾃﾞｰﾀを受け取るsensor_data_t構造体です。

尺度調整された圧力測定はﾊﾟｽｶﾙ(Pa)で表されます。関数が戻る時に、この値はpressure_data構造体からpressure.value領域を使用して読むことができます。

7.7.1. ｺｰﾄﾞの流れの例

7.7.1.1. 定義と宣言

```
#include "sensor.h"
sensor_t        press_dev;      // 装置記述子
sensor_data_t   press_data;     // 装置からの圧力ﾃﾞｰﾀ
```

7.7.1.2. 感知器初期化

```
sensor_attach (&press_dev, SENSOR_TYPE_BAROMETER, 0, 0);
```

7.7.1.3. 感知器読み込み

```
pressure_data.scaled = true;        // パスカル(Pa)で値読み込み指定
sensor_get_pressure (&press_dev, &press_data);
```

7.7.1.4. 応用でのデータ使用

```
int32_t  app_pressure        = press_data.pressure.value;
uint32_t app_read_time       = press_data.timestamp;
```

7.8. 温度

他の殆どの形式の測定と異なり、温度読み取りは専用の温度感知器、または異なる主感知器機能を持つけれども2次的出力値として温度データを提供することができる装置からのどちらかから来得ます。

専用の温度感知器は一般的に安定で高精度の読み取りを提供します。他の形式の感知器装置からの2次的温度データは代表的に主測定の内部的な温度補償に使用され、それらの温度読み取りは度々かなり荒い精度仕様です。

sensor_get_temparature()関数はこのような測定を支援するどんな感知器からの温度データへのアクセスも許します。2次機能として温度測定を許可するために特別な装置初期化は全く必要とされません。この関数は以下の形式を取ります。

```
sensor_get_temperature (&device, &temp_data);
```

ここでのdeviceは温度測定に使用する感知器装置の装置記述子、temp_dataは温度データを受け取るsensor_data_t構造体です。

必要ならば、温度データは単一応用で複数の感知器装置から得ることができます。sensor_get_temparature()を呼ぶ時に単純に各種の装置記述子を指定してください。

尺度調整された温度測定は摂氏温度で表されます。関数が戻る時に、この値はtemp_data構造体からtemperature.value領域を使用して読むことができます。

7.8.1. コードの流れの例

この例は温度読み取りが専用の温度感知器と回転儀(ジャイロスコープ)感知器の両方からどう得られるかを示します。温度測定を支援する各種形式の装置から温度を読むのに、同等の手順を使用することができます。

7.8.1.1. 定義と宣言

```
#include "sensor.h"
sensor_t           temp_dev;        // 装置記述子
sensor_data_t      temp_data;       // 装置からの温度データ
```

7.8.1.2. 感知器初期化

専用の温度感知器を初期化するには以下を使用してください。

```
sensor_attach (&temp_dev, SENSOR_TYPE_TEMPERATURE, 0, 0);
```

例えそれが温度と別の(主)感知器機能の両方に使用されるとしても、感知器初期化は感知器装置毎に一度行われることだけが必要です。

注: 指定する形式は(2次的な温度機能ではなく)感知器の主機能に対してです。

例えば、(温度感知も支援する)回転儀(ジャイロスコープ)装置に対する以下の初期化関数への単一呼び出しは回転と温度の両測定を許可するのに必要とされるものの全てです。

```
sensor_attach (&gyro_dev, SENSOR_TYPE_GYROSCOPE, 0, 0);
```

その後、後続するsensor_getrotation()またはsensor_get_temperature()のどちらの呼び出し中にも、同じ記述子(gyro_dev)が使用されます。

7.8.1.3. 感知器読み込み

```
temp_data.scaled = true;                          // 摂氏温度で値読み込み指定
sensor_get_temperature (&temp_dev, &temp_data);   // 温度感知器データ
sensor_get_temperature (&gyro_dev, &temp_data);   // 回転儀の温度データ
```

7.8.1.4. 応用でのデータ使用

```
int32_t  app_temperature     = temp_data.temperature.value;
uint32_t app_read_time       = temp_data.timestamp;
```

8. 感知器事象の扱い

要求での測定報告に加えて、殆どの感知器装置はそれらの物理的な周辺を継続的に監視するためのいくつかの機構を提供し、或る基準に合致する、または内部的な条件が起こる時に、外部的に見ることができる事象を生成します。ATMEL Sensors Xplainedｿﾌﾄｳｪｱは各種感知器装置を超えて一貫した規則でこれらの非同期事象を処理するための支援を提供します。

感知器装置は特定の出力ﾋﾟﾝのﾚﾍﾞﾙの変更によって事象を告知します。各感知器装置の出力ﾋﾟﾝはATMEL AVRﾏｲｸﾛ ｺﾝﾄﾛｰﾗの入力ﾋﾟﾝに接続されます。これらの接続は特定の感知器基板とﾌﾟﾛｾｯｻ基板の組み合わせに対してSensors Xplained形態設定が自動的に確立する、ﾊｰﾄﾞｳｪｱ形態設定情報の一部です。AVRﾏｲｸﾛ ｺﾝﾄﾛｰﾗの入力ﾋﾟﾝはその後にﾚﾍﾞﾙ変化時に割り込みを生成するのに使用されます。Sensors Xplainedｿﾌﾄｳｪｱでの事象支援は、使用されつつある実際のﾋﾟﾝや割り込み元への特定の参照なしに、あなたの応用へ一般的に適切な処理部を構成設定して許可することを許します。

一旦それが構成設定されて許可されると、あなたの応用の事象処理ﾙｰﾁﾝは指定された感知器事象が起こる時に必ず呼ばれます。あなたの処理ﾙｰﾁﾝはその後に応用の全体的な状態、新しい感知器ﾃﾞｰﾀの処理、使用者への表示提供などのような何れかの適切な活動を実行することができます。

8.1. 事象処理部の追加

感知器事象処理部はsensor_add_event()関数を用いて追加されます。この関数はsensor_event_desc_t事象記述子構造体のｱﾄﾞﾚｽである単一の入力ﾊﾟﾗﾒｰﾀを取ります。

sensor_event_desc_t構造体は以下を含む事象処理部の動きを指定する様々な領域を含みます。

- 事象を生成する感知器のsensor_t記述子。
- 検出する事象の形式。複数の事象に対して共通処理部を使用するには、複数の事象形式の論理和(OR)を指定してください。
- 処理ﾙｰﾁﾝのｱﾄﾞﾚｽ。
- 処理ﾙｰﾁﾝへの引数(代表的に、sensor_event_desc_t構造体のｱﾄﾞﾚｽ)。
- 事象からの感知器ﾃﾞｰﾀが尺度調整された単位かまたは生のﾃﾞｰﾀのどちらかにされるべきか。
- 事象が初期に許可または禁止のどちらにされるか。

8.2. 事象処理ﾙｰﾁﾝ

事象処理ﾙｰﾁﾝはあなたの応用の一部として作成されます。感知器装置ﾄﾞﾗｲﾊﾞが必要などの内部事象ｻｰﾋﾞｽも実行した後で、この処理部は対応する感知器事象が起こる時に呼ばれます。

事象処理部は以下の形式を取ります。

```
void handler_name (volatile void * in);
```

処理ﾙｰﾁﾝはvoid * inの単一入力ﾊﾟﾗﾒｰﾀを取ります。このﾊﾟﾗﾒｰﾀは現実的にｼｽﾃﾑに事象を追加する時に使用されたsensor_event_desc_t構造体のｱﾄﾞﾚｽです。sensor_event_ desc_t構造体内で、data領域は事象中に得られた実際の感知器ﾃﾞｰﾀを保持するsensor_data_t構造体を含みます。

事象処理部は単一特定事象が起こる時にだけ呼ばれるように定義することができ、若しくは事象の組のどれか1つが起こる時に共通処理部が呼ばれるように定義することができます。処理部への呼び出しを開始した実際の事象を決めるには、事象を示すsensor_event_t形式符号を含むsensor_event_desc_t構造体内の事象領域を調査してください。

注: あなたの事象処理ﾙｰﾁﾝは割り込み処理の一部として、あなたの応用の標準実行から非同期で呼ばれます。処理部が割り込みﾚﾍﾞﾙで実行するので、あなたの処理部が走行する間、他の割り込みは遮蔽されます(ｻｰﾋﾞｽされることを妨げられます)。従って、処理部自体内の処理を最小にすることが必要で、そしてあなたの通常の応用ｺｰﾄﾞで後続する作業を実行するようにあなたの処理部と応用を構成すべきです。これは取られるべき付加活動を始めるために、あなたの主ﾌﾟﾛｸﾞﾗﾑの規則的な周回実行中に調べられるﾌﾗｸﾞまたは状態変数を設定する処理部を持つことによって行うことができます。

8.3. 事象の許可と禁止

事象追加後、それはsensor_enable_event()関数を呼ぶことによって動的に許可することができ、またそれはsensor_disable_event()関数を呼ぶことによって禁止することができます。例え事象が禁止されても、事象処理ﾙｰﾁﾝは定義されたままですが、感知装置設定は事象割り込みが生成されないように変更されます。

8.4. 処理ﾙｰﾁﾝのない事象

通常、あなたは感知器ﾃﾞｰﾀを処理、または他の特別な活動を取ることができるように、指定した感知器事象が起きる時に呼ばれるべき処理ﾙｰﾁﾝを定義します。けれども、あなた自身の処理ﾙｰﾁﾝの提供なしに感知器事象を追加して許可することが可能です。これは感知器が生成する事象割り込みの単純な働きが必要な効果を提供する時に行われ得ます。例えば、ﾏｲｸﾛ ｺﾝﾄﾛｰﾗが低電力休止動作形態に置かれる場合に、加速度計からの動き検出割り込みは更なる必要な処理なしで、ｼｽﾃﾑの起き上がりに使用され得ます。

あなた自身の処理部の提供なしで事象処理を構成設定するには、何時も通りにsensor_add_event()関数を呼びますが、handler領域を0(NULL)に設定してください。

8.5. 事象形式

特定感知器装置に依存して、生成され得る多数の形式の感知器事象があります。各個別感知器によって生成することができる事象のより多くの情報については15頁の第9.章でﾃﾞﾊﾞｲｽﾄﾞﾗｲﾊﾞ記述をご覧ください。

可能な事象形式は以下を含みます。

- SENSOR_EVENT_NEW_DATA – 新しい感知器ﾃﾞｰﾀが利用可能。
- SENSOR_EVENT_MOTION – 運動検出用の設定可能閾値で度々使用される、装置運動が検出された。
- SENSOR_EVENT_LOW_G – 低い引力(即ち自由落下)検出。
- SENSOR_EVENT_HIGH_G – 高い引力(加速)検出。
- SENSOR_EVENT_TAP – 装置で物理的に軽く叩くの(軽打)を検出。
- SENSOR_EVENT_TILT – 装置傾き検出。

8.6. ｺｰﾄﾞの流れの例

8.6.1. 運動検出事象

以下の例は加速度計感知器から生成された運動検出事象の構成設定と使用の方法を示します。この流れ全体は他の感知器や事象形式に対して同等です。

8.6.1.1. 定義と宣言

```
#include "sensor.h"
sensor_t                accel_dev;          // 装置記述子
sensor_event_desc_t     accel_event;        // 事象記述子
sensor_data_t           accel_data;         // 事象処理部からのﾃﾞｰﾀ
```

8.6.1.2. 感知器と事象の初期化

```
sensor_attach (&accel_dev, SENSOR_TYPE_ACCELEROMETER, 0, 0);

accel_event = {
    .sensor       = &accel_dev,             // 加速度計使用
    .event        = SENSOR_EVENT_MOTION,    // 運動検出事象
    .data.scaled  = true,                   // 尺度調整されたﾃﾞｰﾀを返す
    .handler      = accel_handler,          // 処理部のｱﾄﾞﾚｽ
    .arg          = &accel_data,            // ﾃﾞｰﾀを置く場所
    .enabled      = true                    // 事象許可
};

sensor_add_event (&accel_event);            // 事象追加
```

8.6.1.3. 感知器事象処理ﾙｰﾁﾝ

```
void accel_handler (volatile void * in)
{
    // 入力の記述子ｱﾄﾞﾚｽへ位置指示子を設定
    sensor_event_desc_t * const event = (sensor_event_desc_t *) in;

    // ﾃﾞｰﾀ複写 - 注目'引数(arg)'='accel_data'構造体のｱﾄﾞﾚｽ
    *((sensor_data_t *)(event->arg)) = event->data;

    /* 他の要素を行う(応用に対するﾌﾗｸﾞ設定など) */
}
```

8.6.2. 軽打検出事象

以下の例は加速度計感知器から生成された軽打検出事象の構成設定と使用の方法を示します。軽打検出が形態設定可能なﾀｲﾐﾝｸﾞと強度閾値の数に関係するので、初期化は他の事象と多少異なります。sensor_set_tap()関数は軽打検出に影響を及ぼす様々なﾊﾟﾗﾒｰﾀの設定に使用されます。

全ての加速度計装置が軽打検出事象を生成できる訳ではありません。SENSOR_EVENT_TAPが支援される事象形式の1つかどうかを判断するには15頁の第9.章で個別装置ﾄﾞﾗｲﾊﾞをご覧ください。

8.6.2.1. 定義と宣言

```
#include "sensor.h"
sensor_t                accel_dev;          // 装置記述子
sensor_event_desc_t     tap_event;          // 事象記述子
sensor_data_t           tap_data;           // 事象処理部からのﾃﾞｰﾀ
sensor_tap_params_t     tap_params;         // 軽打検出ﾊﾟﾗﾒｰﾀ
```

8.6.2.2. 感知器と事象の初期化

```
sensor_attach (&accel_dev, SENSOR_TYPE_ACCELEROMETER, 0, 0);

tap_params = {
    .count          = 2,                    // 最大2つの軽打検出
    .axes = (SENSOR_TAP_AXIS_X | SENSOR_TAP_AXIS_Y | SENSOR_TAP_AXIS_Z),  // 3軸全てで軽打掲出
    .threshold_min  = 0,                    // 既定最小強度を使用
    .threshold_max  = 0,                    // 既定最大強度を使用
    .total_time     = 400,                  // 軽打検出持続時間 400ms
    .tap_time_min   = 5,                    // 各軽打は≧5msでなければなりません。
    .tap_time_max   = 50,                   // 各軽打は≦50msでなければなりません。
    .between_time   = 300,                  // 軽打間空隙≦300ms
    .ignore_time    = 100                   // 軽打間空隙≧100ms
};

sensor_set_tap (&accel_dev, &tap_params);   // 軽打ﾊﾟﾗﾒｰﾀ設定

tap_event = {
    .sensor         = &accel_dev,           // 加速度計使用
    .event          = SENSOR_EVENT_TAP,     // 軽打検出事象
    .data.scaled    = true,                 // 尺度調整されたﾃﾞｰﾀを返す
    .handler        = tap_handler,          // 処理部のｱﾄﾞﾚｽ
    .arg            = &tap_data,            // ﾃﾞｰﾀを置く場所
    .enabled        = true                  // 事象許可
};

sensor_add_event (&accel_event);            // 事象追加
```

8.6.2.3. 感知器事象処理ルーチン

```
void tap_handler (volatile void * in)
{
    // 入力の記述子ｱﾄﾞﾚｽへ位置指示子を設定
    sensor_event_desc_t * const event = (sensor_event_desc_t *) in;

    // ﾃﾞｰﾀ複写 - 注目'引数(arg)'='tap_data'構造体のｱﾄﾞﾚｽ
    *((sensor_data_t *)(event->arg)) = event->data;

    // 軽打検出数調査
    if (event->data.tap.count == 1) {       // 単一軽打
        /* 単一軽打検出用の活動を行ってください。 */

    } else if (event->data.tap.count == 2) // 2重軽打
        /* 2重軽打検出用の活動を行ってください。 */
    }

    // 軽打が検出された軸を調査
    switch (event->data.tap.axis) {
        case SENSOR_TAP_AXIS_X:
            /* X軸軽打検出用の活動を行ってください。 */
            break;
        case SENSOR_TAP_AXIS_Y:
            /* Y軸軽打検出用の活動を行ってください。 */
            break;
        case SENSOR_TAP_AXIS_Z:
            /* Z軸軽打検出用の活動を行ってください。 */
            break;
    }

    // (軸上での)軽打方向を調査
    switch (event->data.tap.direction) {
        case SENSOR_TAP_DIRECTION_POS:
            /* 正方向軽打検出用の活動を行ってください。 */
            break;
        case SENSOR_TAP_DIRECTION_NEG:
            /* 負方向軽打検出用の活動を行ってください。 */
            break;
    }
}
```

9. 感知器装置ﾄﾞﾗｲﾊﾞ

ATMEL Sensors Xplaiendｿﾌﾄｳｪｱは各種感知器装置に渡って一貫して変わらない高位ｲﾝﾀｰﾌｪｰｽの組を提供するように設計されています。けれども、異なるﾊｰﾄﾞｳｪｱ装置は結局異なる機能と設定を提供し、故にいくつかの制御ｲﾝﾀｰﾌｪｰｽは特定装置ﾄﾞﾗｲﾊﾞに依存します。

本章は支援される各装置に対するﾄﾞﾗｲﾊﾞ依存能力を要約します。

9.1. AKM AK8975羅針盤(ｺﾝﾊﾟｽ)/磁力計

9.1.1. 範囲

AK8957装置は各種測定範囲を提供せず、故に変更は全く不可能です。

9.1.2. 採取周波数/帯域

AK8957装置は各種採取周波数を提供せず、故に変更は全く不可能です。

9.1.3. 校正

殆どの羅針盤(ｺﾝﾊﾟｽ)/磁力計のように、AK8957は正確な測定を提供するために校正が必要です。Sensors Xplainedｿﾌﾄｳｪｱは基本的にあなたのｼｽﾃﾑに於ける(金属部品などの近くのための)一定の磁気変位(ｵﾌｾｯﾄ)を修正するのに使用することができる手動校正法を含みます。この校正法は校正処理の複数の段階間で位置変更されることを装置に求めます。

3段階手動校正手順の使用例についてはcompass_calibration応用をご覧ください。この応用は基板を2者選択で、指定位置(平坦、180°回転または反転)へ移動してﾌﾟﾛｾｯｻ基板上の釦を押すことを使用者に要求します。このような3つの測定後、羅針盤(ｺﾝﾊﾟｽ)の3軸を修正することができます。

9.1.4. 自己検査

AK8957装置は感知器装置に既知のﾊﾞｲｱｽを加えて感知器読み取りの結果が期待する範囲内であることを確認する、基本的な自己検査を提供します。SENSOR_TEST_DEFAULT形式符号でsensor_selftest()関数を使用してください。感知器が正しく動作するなら、関数はtrueを返します。検査失敗の場合、関数はfalseを返します。

9.1.5. 事象

AK8957装置は以下のような事象を生成することができます。

- SENSOR_EVENT_NEW_DATA

9.2. Bosch BMA150加速度計

9.2.1. 範囲

BMA150装置はﾐﾘ ｸﾞﾗﾑ(mg)で表される以下のような範囲設定を提供します。

- 2000 (±2g)
- 4000 (±4g)
- 8000 (±8g)

既定設定は4000mg(±4g)です。

9.2.2. 採取周波数/帯域

BMA150装置はﾍﾙﾂ(Hz)で表される以下のような採取周波数設定を提供します。

- 25
- 50
- 100
- 190
- 375
- 750
- 1500

既定設定は1500Hzです。

9.2.3. 校正

BMA150装置は校正の必要がありません。

9.2.4. 自己検査

BMA150装置は感知器装置に既知のﾊﾞｲｱｽを加えて感知器読み取りの結果が期待する範囲内であることを確認する、基本的な自己検査を提供します。SENSOR_TEST_DEFAULT形式符号でsensor_selftest()関数を使用してください。感知器が正しく動作するならば、関数はtrueを返します。検査失敗の場合、関数はfalseを返します。

9.2.5. 事象

BMA150装置は以下のような事象を生成することができます。

- SENSOR_EVENT_MOTION
- SENSOR_EVENT_HIGH_G
- SENSOR_EVENT_LOW_G
- SENSOR_EVENT_NEW_DATA

9.3. Bosch BMP085圧力感知器

9.3.1. 範囲

BMP085装置は各種測定範囲を提供せず、故に変更は全く不可能です。

9.3.2. 採取周波数/帯域

BMP085装置は各種採取周波数を提供せず、故に変更は全く不可能です。

9.3.3. 校正

BMP085装置は校正の必要がありません。

9.3.4. 自己検査

BMP085装置は自己検査機能を提供しません。

9.3.5. 事象

BMP085装置は以下のような事象を生成することができます。

- SENSOR_EVENT_NEW_DATA

9.4. Honeywell HMC5883L羅針盤(ｺﾝﾊﾟｽ)/磁力計

9.4.1. 範囲

HMC5883L装置はﾏｲｸﾛ ﾃｽﾗ(μT)で表される以下のような範囲設定を提供します。

- 90 (±0.9G(ｶﾞｳｽ))
- 139 (±1.3G)
- 190 (±1.9G)
- 250 (±2.5G)
- 400 (±4.0G)
- 470 (±4.7G)
- 560 (±5.6G)
- 810 (±8.1G)

既定設定は130μT(±1.3G)です。

9.4.2. 採取周波数/帯域

HMC5883L装置はﾍﾙﾂ(Hz)で表される以下のような採取周波数設定を提供します。

- 1 (実際は0.75)
- 2 (実際は1.5)
- 3
- 8 (実際は7.5)
- 15
- 30
- 75

既定設定は15Hzです。

9.4.3. 校正

殆どの羅針盤(ｺﾝﾊﾟｽ)/磁力計のように、HMC5883Lは正確な測定を提供するために校正が必要です。ATMLE Sensors Xplainedｿﾌﾄｳｪｱは基本的にあなたのｼｽﾃﾑに於ける(金属部品などの近くのための)一定の磁気変位(ｵﾌｾｯﾄ)を修正するのに使用することができる手動校正法を含みます。この校正法は校正処理の複数の段階間で位置変更されることを装置に求めます。

3段階手動校正手順の使用例についてはcompass_calibration応用をご覧ください。この応用は基板を2者選択で、指定位置(平坦、180°回転または反転)へ移動してﾌﾟﾛｾｯｻ基板上の釦を押すことを使用者に要求します。このような3つの測定後、羅針盤(ｺﾝﾊﾟｽ)の3軸を修正することができます。

9.4.4. 自己検査

HMC5883L装置は感知器装置に既知のﾊﾞｲｱｽを加えて感知器読み取りの結果が期待する範囲内であることを確認する、基本的な自己検査を提供します。SENSOR_TEST_DEFAULT形式符号でsensor_selftest()関数を使用してください。感知器が正しく動作するなら、関数はtrueを返します。検査失敗の場合、関数はfalseを返します。

9.4.5. 事象

HMC5883L装置は以下のような事象を生成することができます。

- SENSOR_EVENT_NEW_DATA

9.5. Invensense IMU-3000回転儀(ｼﾞｬｲﾛｽｺｰﾌﾟ)/運動処理器

9.5.1. 範囲

IMU-3000装置は±角度/秒(°/s)で表される以下のような範囲設定を提供します。

- 250
- 500
- 1000
- 2000

既定設定は±2000°/s)です。

9.5.2. 採取周波数/帯域

IMU-3000装置はヘルツ(Hz)で表される以下のような採取周波数設定を提供します。

- 5
- 10
- 20
- 42
- 98
- 188
- 256
- 2100

既定設定は256Hzです。

9.5.3. 校正

IMU-3000装置は校正の必要がありません。

9.5.4. 自己検査

IMU-3000装置は自己検査機能を提供しません。

9.5.5. 事象

IMU-3000装置は以下のような事象を生成することができます。

- SENSOR_EVENT_NEW_DATA

9.6. Invensense ITG-3200回転儀(ｼﾞｬｲﾛｽｺｰﾌﾟ)

9.6.1. 範囲

ITG-3200装置は±2000°/sの1つの動作範囲だけを提供します。

9.6.2. 採取周波数/帯域

ITG-3200装置はヘルツ(Hz)で表される以下のような採取周波数設定を提供します。

- 5
- 10
- 20
- 42
- 98
- 188
- 256
- 2100

既定設定は256Hzです。

9.6.3. 校正

ITG-3200装置は校正の必要がありません。

9.6.4. 自己検査

ITG-3200装置は自己検査機能を提供しません。

9.6.5. 事象

ITG-3200装置は以下のような事象を生成することができます。

- SENSOR_EVENT_NEW_DATA

9.7. Kionix KXTF9加速度計

注: Kionix KXTF9ﾄﾞﾗｲﾊﾞはKionix KXTI9型加速度計も支援します。

9.7.1. 範囲

KXTF9装置はﾐﾘ ｸﾞﾗﾑ(mg)で表される以下のような範囲設定を提供します。

- 2000 (±2g)
- 4000 (±4g)
- 8000 (±8g)

既定設定は4000mg(±4g)です。

9.7.2. 採取周波数/帯域

KXTF9装置はヘルツ(Hz)で表される以下のような採取周波数設定を提供します。

- 13 (実際は12.5)
- 25
- 50
- 100
- 200
- 400
- 800

既定設定は200Hzです。

9.7.3. 校正

KXTF9装置は校正の必要がありません。

9.7.4. 自己検査

KXTF9装置は自己検査機能を提供しません。

9.7.5. 事象

KXTF9装置は以下のような事象を生成することができます。

- SENSOR_EVENT_MOTION
- SENSOR_EVENT_NEW_DATA
- SENSOR_EVENT_TAP
- SENSOR_EVENT_TILT

10. 資料改訂履歴

10.1. 改訂A、01/11

暫定/β公開用初版

10.2. 改訂B、06/11

(ATMEL AVR Studio 5とASF2.5での公開のために)ソフトウェア1.1版用更新

11. 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AVR4016応用記述(doc8367.pdf Rev.8367B-06/11)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。