ミニ四駆制御基板用ソフトウェア開発ガイド

目次

[ソフトウェア開発環境概要 6](#_Toc415103926)

[ソフトウェアコンポーネント 6](#_Toc415103927)

[システムソフトウェアとユーザーランド 7](#_Toc415103928)

[SDKディレクトリ構成(TBD) 8](#_Toc415103929)

[ベアメタル開発向け参考情報 8](#_Toc415103930)

[その他開発者向け情報 8](#_Toc415103931)

[開発手順 10](#_Toc415103932)

[開発環境の準備 10](#_Toc415103933)

[サンプルのビルド方法 10](#_Toc415103934)

[基板へのソフトウェア書込の方法 11](#_Toc415103935)

[ハードウェアプログラマを利用した書込み方法 11](#_Toc415103936)

[USB（シリアルポート）経由での書込み 12](#_Toc415103937)

[システムソフトウェア 15](#_Toc415103938)

[システムソフトウェアとは 15](#_Toc415103939)

[ユーザーランドソフトウェアとの関係性 15](#_Toc415103940)

[システムソフトウェアが提供する機能 15](#_Toc415103941)

[システムソフトウェアAPIリファレンス 16](#_Toc415103942)

[systemServGetSystemBatteryVoltage 17](#_Toc415103943)

[systemServGetMini4wdBatteryVoltage 18](#_Toc415103944)

[systemDebugLed 19](#_Toc415103945)

[ユーザランドソフトウェア 20](#_Toc415103946)

[ユーザランドソフトウェアとは 20](#_Toc415103947)

[ユーザランドソフトウェア関数要求仕様 22](#_Toc415103948)

[void initialize\_userland(void) 23](#_Toc415103949)

[void updateUserland(void) 24](#_Toc415103950)

[uint8\_t onSensorInterrupt(void \*sensor) 25](#_Toc415103951)

[void onTimerInterrupt(uint32\_t tick) 25](#_Toc415103952)

[void onSw0Pressed(void) 25](#_Toc415103953)

[void onSw0Repeat(void) 26](#_Toc415103954)

[void onSw0Released(void) 26](#_Toc415103955)

[void onSw1Pressed(void) 26](#_Toc415103956)

[void onSw1Repeat(void) 27](#_Toc415103957)

[void onSw1Released(void) 27](#_Toc415103958)

[void onSw2Pressed(void) 27](#_Toc415103959)

[void onSw2Repeat(void) 28](#_Toc415103960)

[void onSw2Released(void) 28](#_Toc415103961)

**注意事項**

本SDKでは以下のオープンソースソフトウェアを静的あるいは動的に利用しています。　ただし本SDKへの組み込みにあたって追加修正等の措置を行っている可能性がある為、オリジナルの挙動を保証するものであはりません。

* FatFs
  + http://elm-chan.org/fsw/ff/00index\_j.html

# ソフトウェア開発環境概要

## ソフトウェアコンポーネント

　ミニ四駆制御基板SDK（以下、単にSDK）は、ミニ四駆制御基板において制御可能なデバイスの操作を容易化したデバイスドライバ、割込処理や処理のフレームワークを提供するシステムソフトウェア、ユーザが実際に制御を行う為に開発するユーザランドソフトウェアの3つのソフトウェアコンポーネントから成ってます。

　デバイスドライバはUART通信、GPIO、I2C、アナログ‐デジタルコンバータ（以下ADC）、周期タイマ等マイコンのハードウェアに依存する処理をC言語の関数として定義した関数ライブラリとして提供されます。　通常ユーザランドから呼び出す必要はありませんが、システムソフトウェアはデバイスドライバを直接使用しています。デバイスドライバは通常関数呼出によって機能を提供しますが、割込処理についてはC言語の関数ポインタを用いたコールバックによって割込み発生を通知する機能を有しています。

　システムソフトウェアはエントリーポイント(通常main関数)を含むソフトウェアで、制御のフレームワークを提供します。マイコン上で発生した割込は一旦全てシステムソフトウェアが受けます。制御基板上の各デバイスは、ユーザランドの処理が呼ばれるよりも前にシステムソフトウェアによって初期化が行われます。そのほか、microSDカードからパラメータセットを読み込むと言った処理もここで提供されます。システムソフトウェアは通常、ユーザランド側に定義されたユーザ関数を呼び出しながら動作します。これはソフトウェア一連の処理はシステムソフトウェアが流れを管理している事を意味しています。一部の機能については、ユーザランドソフトウェアからの関数呼出と言う形で提供されています。

　ユーザランドソフトウェアはC言語の関数と言う形でユーザ自信が定義するものです。初期状態のSDKには空のユーザランドソフトウェアが含まれています。定義すべき関数はuserland.hに定義されています。ユーザランドソフトウェアは通常、非同期にシステムソフトウェアから呼び出される関数セットとして定義されます。例えばSDKではユーザランド側にonTimerInterruptという関数を定義する事を求めていますが、これはシステムソフトウェアが管理している周期タイマーの周期で呼び出される関数になります。ユーザランドでは、システムソフトウェアおよびデバイスドライバが提供するAPI関数を呼び出す事が可能です。システムソフトウェアが提供するAPI関数はユーザランド側のどこから呼び出しても安全に設計されていますが、デバイスドライバが提供するAPI関数は呼び出すタイミングによってはハングアップや意図しない動作を引き起こす可能性があります。この為、ユーザランドからのデバイスドライバAPI関数の使用は推奨されません。

図 1ソフトウェア階層構造

## システムソフトウェアとユーザーランド

　SDKで用意されているソフトウェアモデルでは、システムソフトウェアが適当なタイミングでユーザランド側に定義された関数を呼び出すことで動作します。ユーザランド側で用意すべき関数は次の通りです。個々の関数が呼び出されるタイミングについては、「ユーザランドソフトウェア」の項目を参照してください。

* initialize\_userland ユーザランドの初期化関数。最初に呼ばれる関数
* updateUserland ユーザランドのメイン処理
* onSensorInterrupt モーションセンサのデータが更新された時の処理
* onTimerInterrupt タイマーによって一定間隔毎に実行される処理
* onSw0Pressed スイッチ0(SW0)が押下時に実行される処理
* onSw0Repeat スイッチ0(SW0)が押下中繰り返し実行される処理
* onSw0Released スイッチ0(SW0)が放された時に実行される処理
* onSw1Pressed スイッチ1(SW1)が押下時に実行される処理
* onSw1Repeat スイッチ1(SW1)が押下中繰り返し実行される処理
* onSw1Released スイッチ1(SW1)が放された時に実行される処理
* onSw2Pressed スイッチ2(SW2)が押下時に実行される処理
* onSw2Repeat スイッチ2(SW2)が押下中繰り返し実行される処理
* onSw2Released スイッチ2(SW2)が放された時に実行される処理

　システムソフトウェアは、ユーザランドソフトウェアに定義された全ての関数（以降、ユーザランド関数）が有限時間内に処理を完了して戻ってくる事を期待しています。これは全てのユーザランド関数は長時間ビジーループすべきでないと言う意味です。特にonSensorInterrupt関数はモーションセンサのデータが更新されると、割込ハンドラの中から呼ばれる関数です。モーションセンサは通常800Hz～1600Hzでデータを取得しているので、onSensorInterrupt関数は最悪でも625マイクロ秒以内に処理を終える必要があります。

## SDKディレクトリ構成(TBD)

　SDKのディレクトリ構成と各ディレクトリに含まれるファイルの概要を説明します

（TBD）

## ベアメタル開発向け参考情報

　SDKが提供しているフレームワークはデバイスドライバやシステムソフトウェアと言った抽象化レイヤを多く含み関数呼び出し等のコストが比較的多く必要です。この為HWの性能をフルに生かす事が構造上難しい場合があります。デバイスドライバやシステムソフトウェアを含むSDK全体での最適化でも解決できない場合、SDKを使わずにベアメタル開発を行う事も選択肢の1つです。SDKに含まれるデバイスドライバに適用されるライセンスは再利用を制限しない為、ドライバの中の必要な部分だけを切り出して利用する事も可能です。

## その他開発者向け情報

#### ハードウェア情報

　開発基板に搭載されている部品の中で、ソフトウェアでの制御が必要になるCPU及びモーションセンサの情報です。

* Atmel Xmega E Series (オンボードCPU)
  + 製品情報
    - http://www.atmel.com/tools/xmega-e5xplained.aspx
  + ハードウェアマニュアル
    - http://www.atmel.com/Images/Atmel-42084-XMEGA-E5-Xplained-Hardware-Users-Guide\_Application-Note\_AT02667.pdf
* LSM330DLS(6軸モーションセンサ)
  + 製品情報
    - http://www.st-japan.co.jp/web/jp/catalog/sense\_power/FM89/SC1448/PF252427
  + データシート
    - http://www.st-japan.co.jp/st-web-ui/static/active/jp/resource/technical/document/datasheet/DM00037200.pdf

#### FatFs

SDKではファイルシステムとしてChaN氏制作のFatFsを利用しています。FatFsに関する各種情報は本ドキュメントの範囲外としています。FatFsのAPIに関する情報等は公式Webページに記載されているAPIリファレンスを参照してください。

* FatFs公式Webページ
  + http://elm-chan.org/fsw/ff/00index\_j.html
* FatFsアプリケーションノート
  + http://elm-chan.org/fsw/ff/ja/appnote.html

# 開発手順

　ここではSDKを用意した状態から、実際にバイナリファイルをビルトし、制御基板に書き込むまでの手順を説明します。

## 開発環境の準備

　本SDKは開発環境としてAVR Studio 6を想定しています。リリース時点で動作確認を行っているバージョンは6.2.1153です。Atmelの公式Webサイト(http://www.atmel.com/microsite/atmel\_studio6/)からダウンロードが可能です。

## サンプルのビルド方法

　SDKディレクトリの中にあるサンプルプロジェクト(TBD ディレクトリパスを後で決める)をAVR Studio 6で開きます。メニューバーの[Build] → [Build Solution] を選択します。

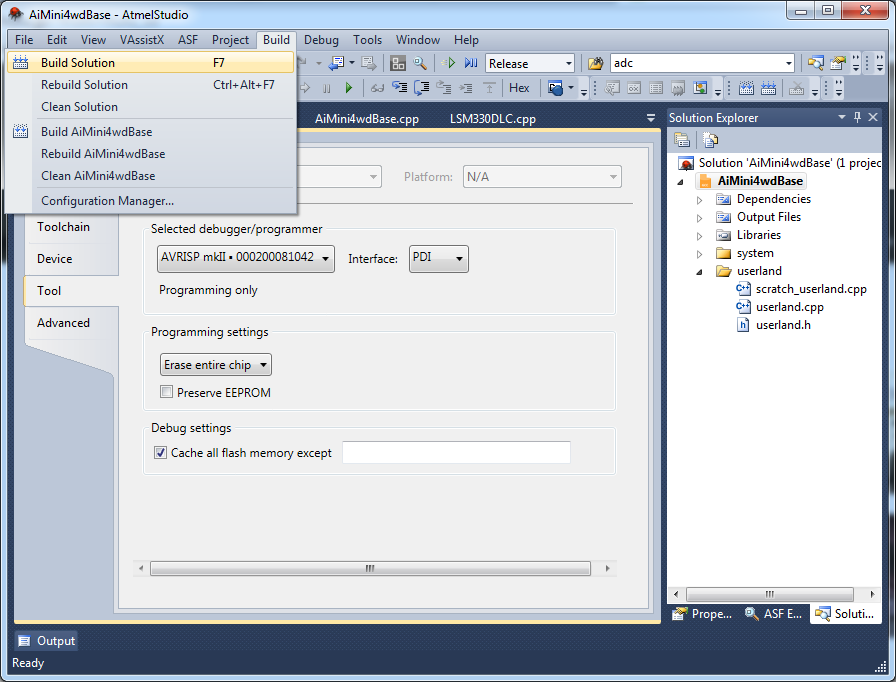


図 2 AVR Studioでのビルド方法

OutputウィンドにBuild Successed. と表示されていればビルドは成功です。ビルド成果物は、プロジェクトファイルのあるディレクトリのDebug又はReleaseディレクトリ内に生成されます。生成されるディレクトリはビルド時のコンフィギュレーションに依存します。詳しくはAVR StudioのHelpをご参照ください

## 基板へのソフトウェア書込の方法

　基板へのソフトウェア書込みには2つの方法があります。

### ハードウェアプログラマを利用した書込み方法

　1つ目はAVR用のプログラマを用いた方法です。SDKリリース時点で動作確認が取れているプログラマはAVRISP mkIIとJTACICE mkII の2機種です。何れの機種でもPDI接続で接続します。接続時の基板側のピン配置は、「ミニ四駆制御基板ハードウェアマニュアル」ドキュメントを参照してください。この方法で書き込むには以下の操作を行います。

1. 基板とプログラマを接続
2. AVR StudioのSolution Explorer からプロジェクトのプロパティを選択

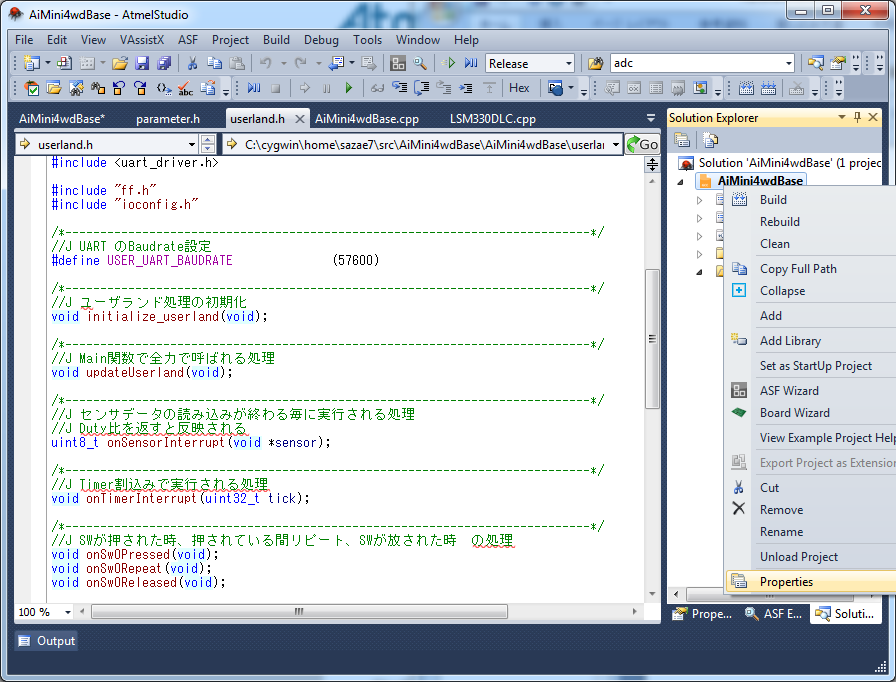


図 3 Solution Explorerでプロジェクトのプロパティを選択

1. プロパティのToolタブのSelected debugger/programmer ので使用するデバイスを選択

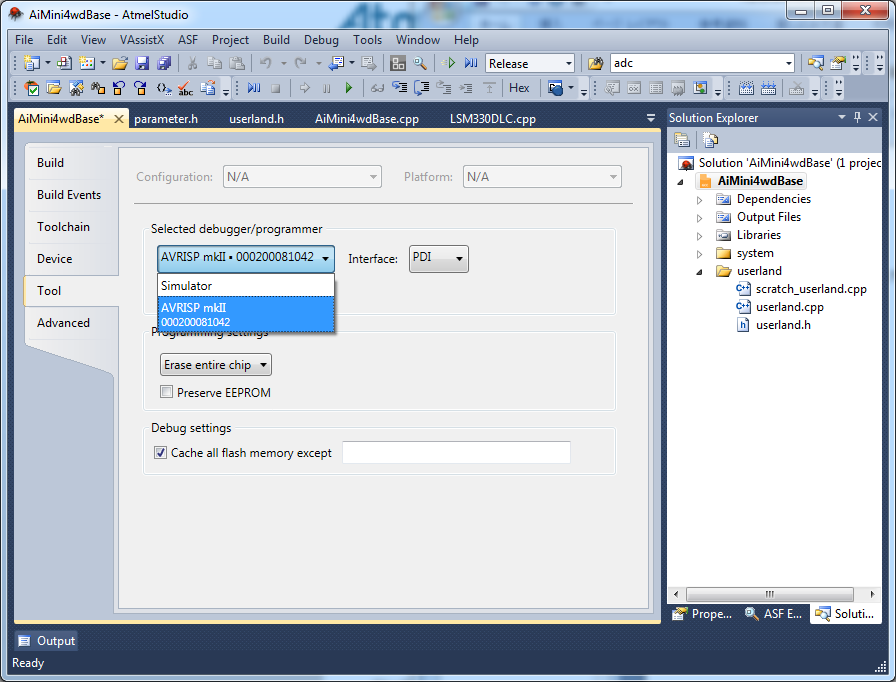


図 4 プロパティのToolタブで使用するツールを選択

1. 制御基板の電源を投入
2. AVR Studioのメニューバー [Debug]→ [Start Debugging and Break] 又は、[Debug]→ [Start Without Debugging] を選択

なお、ハードウェアプログラマの使用方法等についてはAtmelの公式Webページ又は製品付属のマニュアルにより詳しい記載があるので合わせてご参照ください。

### USB（シリアルポート）経由での書込み

　制御基板側にブートローダー及びアップデート機能に対応したシステムソフトウェアが書込み済みであれば、基板上のmicro USBコネクタとPCを接続してソフトウェアの書き込みが可能です。SDKに含まれるシステムソフトウェアはユーザ側が意図的に機能を削除しない限り、この機能に対応した状態でリリースされています。この方法で書き込むには、avedude というソフトウェアを入手する必要があります。最新版は公式Webサイト(http://www.nongnu.org/avrdude/)から入手可能です。

　USB経由での書き込みを行うには以下の操作を行います。

1. コマンドプロンプトを立ち上げ、avedude.exeのあるディレクトリに移動
2. 制御基板の電源をOFFに設定
3. SW0を押下
4. SW0を押下したまま、制御基板とPCをUSBケーブルで接続（Power LEDが点灯）
5. コマンドプロンプトから次のコマンドを打つ

avrdude -p atxmega32e5 -P <COMポート名> -b 115200 -c avr109 -D -U flash:w:<ファイル名>

<COMポート名>

制御基板と繋がっているPC側のUSBポートが認識しているCOMポート名

<ファイル名>

書き込むべきファイル名

例) COMポート名COM10を使って、AiMini4wdBase.hexを書き込む場合、以下のようになります。

avrdude -p atxmega32e5 -P COM10 -b 115200 -c avr109 -D -U flash:w:AiMini4wdBase.hex

1. 書込み終了

書込み終了後べリファイ結果が表示されます。正常に終了していれば以下のように表示されます。

avrdude: verifying ...

avrdude: 26094 bytes of flash verified

書込み開始（上記手順5）から終了（上記手順6）までのコンソールログのキャプチャは以下のようになります。ただしこれはAvrdudeのバージョンによって挙動が異なるため、永続的にこの結果を保証するものではありません。

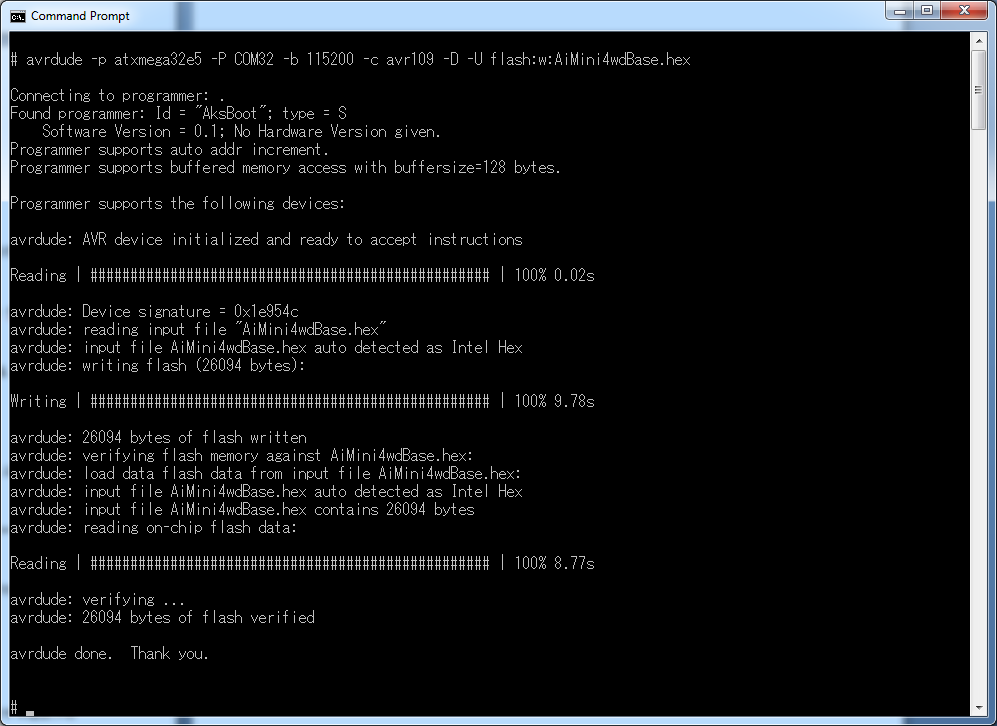


図 5 avrdude による書き込み時のコンソールログ

このモードを使う場合には、不要不急出ない限りシステムソフトウェア側の実装を変更しない事をお勧めします。また、AVRISP等のツールを使ってBootloader領域を書き換えた場合この機能が使用不能になる可能性があります。その場合には、AVRISP mkIIやJTAGICE mkII等を用いて、SDKに含まれるブートローダバイナリ(bootloader\_xmega.hex)をBootloader領域に書き込んでください。

# システムソフトウェア

## システムソフトウェアとは

　システムソフトウェアはミニ四駆制御を行う為のフレームワークををユーザランド（ユーザが自身で開発する領域）に提供する事を目的として実装されているソフトウェアセットです。

## ユーザーランドソフトウェアとの関係性

　システムソフトウェアとユーザランドソフトウェアの関係は基本的にはシステムソフトウェアがユーザランドに定義された関数を呼ぶという関係になっています。例えばモーションセンサから新しいセンサデータが取得された時点でonSensorInterrupt()が呼び出されます。

　このように、システム側から非同期に呼び出される処理をコールバック処理と呼びます。ユーザランドで定義すべきコールバック処理の一覧は「ソフトウェア開発環境概要」を参照してください。また個々のコールバック処理が発生するタイミングについては「ユーザランドソフトウェア」を参照してください。

　このほかに、システム側から何らかの情報を取得する為に、API関数も用意されています。API関数の詳細については、「システムソフトウェアAPIリファレンス」を参照してください。

## システムソフトウェアが提供する機能

　システムソフトウェアがAPI関数として提供する機能の一覧を示します。個々の機能については「システムソフトウェアAPIリファレンス」。

* 制御基板用バッテリの電圧取得
* ミニ四駆駆動用バッテリ（単3乾電池2本）の電圧取得
* デバッグ用LEDの点灯・消灯
* Micro SD上のFATファイルシステム
  + FatFsを組み込むことで実現しています

# システムソフトウェアAPIリファレンス

　システムソフトウェアが提供するAPI関数について解説します。ファイルシステム（FatFS）関連のAPI関数については、FatFsの公式Webページをご参照ください。

## systemServGetSystemBatteryVoltage

#### 概要

制御基板用バッテリの電圧を取得します。制御基板用バッテリが切れると制御そのものが継続不能になります。通常、システムソフトウェアはバッテリが枯渇しそうになると警告としてチャージLEDを点滅させます。一方、走行中にバッテリが枯渇しそうになった場合、不要不急のファイルアクセスを減らす等の工夫でバッテリ持続時間を若干伸ばす事が可能です。

#### 引数

なし

#### 戻り値

uint16\_t システム用バッテリ電圧

この戻り値はシステム用バッテリの電圧をミリボルトであらわした結果を返します。通常リチウムポリマーあるいはリチウムイオンバッテリを使用した場合下限は3600mV、上限は4200mV程度です。

#### 注意事項

　この関数はシステムソフトウェア側でキャッシュされたシステム用バッテリ電圧を返す為、呼び出し時にアナログ・デジタル変換を待つと言った挙動は行いません。この為ユーザがデバイスドライバを操作してバッテリ電圧を取得するよりも高速にバッテリ電圧を取得できます。

## systemServGetMini4wdBatteryVoltage

#### 概要

　ミニ四駆駆動用バッテリ（単3乾電池2本）の電圧を取得します。

#### 引数

なし

#### 戻り値

uint16\_t ミニ四駆用バッテリ電圧

この戻り値はミニ四駆用バッテリの電圧をミリボルトであらわした結果を返します。新品単3乾電池2本であれば、3000mV から3400mV 程度の電圧があります。ただし、走行中は電圧降下が起こっている為、2000mVを割る事もあります。

#### 注意事項

　この関数はシステムソフトウェア側でキャッシュされたミニ四駆用バッテリ電圧を返す為、呼び出し時にアナログ・デジタル変換を待つと言った挙動は行いません。この為ユーザがデバイスドライバを操作してバッテリ電圧を取得するよりも高速にバッテリ電圧を取得できます。

## systemDebugLed

#### 概要

デバッグ用LEDの点灯・消灯を行います

#### 引数

USER\_LED\_INDEX ledIndex

USER\_LED\_MODE on\_off

ledIndex で指定したLEDをon\_offで指定した状態に変更します。

ledIndexには以下のいずれかの値を設定してください。

|  |  |
| --- | --- |
| Definition/Value | Description |
| USER\_LED\_0 | デバッグLED0 |
| USER\_LED\_1 | デバッグLED1 |

on\_offには以下のいずれかの値を設定してください。

|  |  |
| --- | --- |
| Definition/Value | Description |
| USER\_LED\_ON | LED点灯状態 |
| USER\_LED\_OFF | LED消灯状態 |

#### 戻り値

なし

#### 注意事項

# ユーザランドソフトウェア

## ユーザランドソフトウェアとは

　ユーザランドソフトウェアは、制御基板を使用するユーザ自身が定義すべき挙動を記述したものです。SDKでは、ユーザランドソフトウェアのスケルトン（空関数）が定義されています。各種サンプルでは、個々のサンプルが提示する挙動が実装されています。

　ユーザランドソフトウェアで定義されるべき関数（以降、ユーザランド関数）は以下の通りです。

* initialize\_userland ユーザランドの初期化関数。最初に呼ばれる関数
* updateUserland ユーザランドのメイン処理
* onSensorInterrupt モーションセンサのデータが更新された時の処理
* onTimerInterrupt タイマーによって一定間隔毎に実行される処理
* onSw0Pressed スイッチ0(SW0)が押下時に実行される処理
* onSw0Repeat スイッチ0(SW0)が押下中繰り返し実行される処理
* onSw0Released スイッチ0(SW0)が放された時に実行される処理
* onSw1Pressed スイッチ1(SW1)が押下時に実行される処理
* onSw1Repeat スイッチ1(SW1)が押下中繰り返し実行される処理
* onSw1Released スイッチ1(SW1)が放された時に実行される処理
* onSw2Pressed スイッチ2(SW2)が押下時に実行される処理
* onSw2Repeat スイッチ2(SW2)が押下中繰り返し実行される処理
* onSw2Released スイッチ2(SW2)が放された時に実行される処理

何れの関数も、システムソフトウェアから非同期に呼び出される可能性がある事に留意すべきです。グローバル変数等を複数のユーザランド関数で使う場合、予期しないタイミングで書き変わる可能性があります。　ユーザランド関数の中でも、onから始まる名前の関数はCPUの割込みコンテクストから呼び出されます。これらの関数は長時間処理が行われる事を期待していない為、可能な限り早く処理を終えるべきです。

　ユーザランド関数がシステムソフトウェアから呼び出されるイメージを図に示します。図からわかるとおり、非割込みコンテクスト（通常コンテクスト）から呼ばれるユーザランド関数はinitialize\_userlandとupdateUserlandの2つだけです。ファイルIO等の長時間CPUを使用するような処理はこの2つの中に実装されるべきです。

図 6　ユーザランド関数呼び出しイメージ

# ユーザランドソフトウェア関数要求仕様

　ここではユーザランドソフトウェアとして実装されるべきユーザランド関数の要求仕様を説明します。SDKが提供するフレームワークを使用する際には、必ず全ての関数を実装する必要があります。ユーザが処理を希望しない場合には関数の処理開始直後にreturnするような構造の関数（空関数）を用意してください。

## void initialize\_userland(void)

#### 概要

　ユーザランド側の定常処理に入る前の初期化処理を行います。各種デバイスの初期化が終了した後に呼ばれる為、ファイルシステムを含む各種デバイスは使用する事が可能です。この関数は、定常処理が始まる前に1回だけ呼び出されます。

#### 引数

なし

#### 戻り値

なし

#### 注意点

　既にこの時点で割込み処理は始まっている為、onから始まるユーザランド関数（割込みコンテクストから呼ばれる関数）が呼び出される可能性があります。初期化が完了するまでこれらの処理を行いたくない場合には、ユーザランド側でケアする必要があります。

## void updateUserland(void)

#### 概要

　システムソフトウェアの定常処理から呼び出される関数です。システムソフトウェは自身が実行すべき処理が無い限りは可能な限りの回数この関数を呼び出します。この関数は内部でビジーループ又はそれに類する処理をを持つ事が許容されます。

#### 引数

なし

#### 戻り値

なし

#### 注意点

　この関数は制御関係の比較的重い処理（平滑化や推定処理など）を行う事を前提に設計されています。この関数の中にいる限り、システムソフトウェアがアップデートするべき情報（バッテリ電圧など）はアップデートされません。

## uint8\_t onSensorInterrupt(void \*sensor)

#### 概要

　モーションセンサから新しいデータが読み込まれるたびに発生する割込み処理の中から呼ばれる関数です。センサデータの取得レートは400Hz(TBD)です。この割込みタイマー割込みよりも正確な周期性を持つ為、システムソフトウェアはこの関数から戻り値としてモーターのDuty比を受け取ります。

#### 引数

*void \*senror* センサーデータへのポインタ

この引数は将来への拡張性を担保する為に特定の型のポインタではなくvoid \*を渡します。現時点でSDKでは、このポインタは

typedef struct {

uint16\_t x;

uint16\_t y;

uint16\_t z;

} user3AXIS\_uint16;

typedef struct {

user3AXIS\_uint16 acceleration;

user3AXIS\_uint16 angularRate;

} USER\_SENSOR\_DATA\_SET;

で定義される、6軸のモーションデータのヘッドを示しています。渡ってくる加速度の範囲は±8G、角速度の範囲は2000dps(TBD)です。

#### 戻り値

uint8\_t モーターのPWM制御値(Duty比、0～100）

システムソフトウェアはonSensorInetrrupt()の戻り値を使って、モーターをPWMで制御します。制御値が0の時（Duty比0）、モーターは完全に停止しています。一方、100を返した場合モーターは最大のパワーで回ります。ミニ四駆の速度を制御する場合、この関数の戻り値を変更する事で実現可能です。

#### 注意点

## void onTimerInterrupt(uint32\_t tick)

#### 概要

#### 引数

#### 戻り値

#### 注意点

## void onSw0Pressed(void)

#### 概要

#### 引数

#### 戻り値

#### 注意点

## void onSw0Repeat(void)

#### 概要

#### 引数

#### 戻り値

#### 注意点

## void onSw0Released(void)

#### 概要

#### 引数

#### 戻り値

#### 注意点

## void onSw1Pressed(void)

#### 概要

#### 引数

#### 戻り値

#### 注意点

## void onSw1Repeat(void)

#### 概要

#### 引数

#### 戻り値

#### 注意点

## void onSw1Released(void)

#### 概要

#### 引数

#### 戻り値

#### 注意点

## void onSw2Pressed(void)

#### 概要

#### 引数

#### 戻り値

#### 注意点

## void onSw2Repeat(void)

#### 概要

#### 引数

#### 戻り値

#### 注意点

## void onSw2Released(void)

#### 概要

#### 引数

#### 戻り値

#### 注意点