# Uzume ユーザーズ・ガイド

*（Kirishima 対応版 0.1.1）*

*最終更新: Jan/9/2012*

Suikan / Uzume Project

# 1. はじめに

この文書は、オープンソース組み込みオーディオ・プラットフォーム Uzume のユーザーズ・ガイドである。

Uzume プラットフォームは、ユーザーが安価なハードウェアで気軽にオーディオ・信号処理の実験を行えることを目標として開発されている。この文書では、Uzume のターゲットとして NXP 社の LPCxpresso 1768/1769 評価基板を用いたオーディオ・ボード、Kirishima を選び、これを使った開発、実験のための手順を説明する。

この文書を読むことで、開発環境の構築、独自オーディオ・アルゴリズムの実装、性能評価の方法を理解することができる。

## 1.1. Uzume プラットフォーム

Uzume プラットフォームは前述の通り、安価なハードウェアで気軽にオーディオ信号処理の実験を行えることを目標として開発されている。

想定するユーザーは DSP への知識が薄いが、信号処理を試してみたいといった人々であり、ギター・エフェクタやシンセサイザを自作するアナログ回路設計者なども含む。

Uzume プラットフォームが提案されたのは 2010 年の夏であったが、最初の試作コードがコミットされたのは 2011 年の秋になってからであった。

現在、オーディオ信号処理の実験を行うメジャーな方法としては以下の三つの方法が考えられる。

1. PC 上のシミュレータを使う
2. 汎用 DSP の評価ボードを使う
3. 専用 DSP の評価ボードを使う

PC 上のシミュレータはもっとも敷居が低く、入手も容易である。アマチュアが使えるシミュレータとしては SciLab があり、マルチプラットフォームでありフリーである。この方法はアルゴリズムの開発には向いているが、最大の欠点としてリアルタイム性を欠くことを挙げられる。

汎用 DSP の評価ボードとしては ADI や TI のボードがある。これらについては価格が四万円程度することと、心理的な障壁が問題になると思われる。TI のボードについては十分な日本語資料があるが、アマチュアにとっては心理的な障壁の高さは無視できないであろう。

専用 DSP ボードとしては ADI の ΣDSP や TI の TAS がある。これらは場合によっては低価格なボードも存在するが、資料が少なく、かつ、日本語の資料が少ない。また、柔軟性に欠けることも問題である。

Uzume はこれらのギャップを埋める位置にある。

- 入手が容易なハードウェアを用いる
- 汎用マイコンを使う

- 高性能な DSP へのスケールアップ・パスを持つ
- 日本語の資料を用意する

Uzume は初めからオーディオ信号処理用のプラットフォームとして開発されているため、覚えなければならないことが比較的少ない。一方で低価格汎用マイコンを使うため、心理的障壁も少なく入手も容易である。

## 1.2. Kirishima ボード

Kirishima ボードは Uzume を動かすために設計された最初のボードである。裸の LSI ではなく、評価ボードを活用することで組み立てやすくなるよう設計してある。

- マイコンボード LPCxpresso 1768/1769
- CODEC ボード MMCODEC01
- インターフェースボード UB232R

このうちもっとも入手が困難なのはポーランド製の MMCODEC01 であるが、基本的には WEB ショッピングであるので、誰にでも入手できるという点はクリアしている。部品代の総計は MMCODEC01 の送料を含めても 1 万 5 千円程度である。

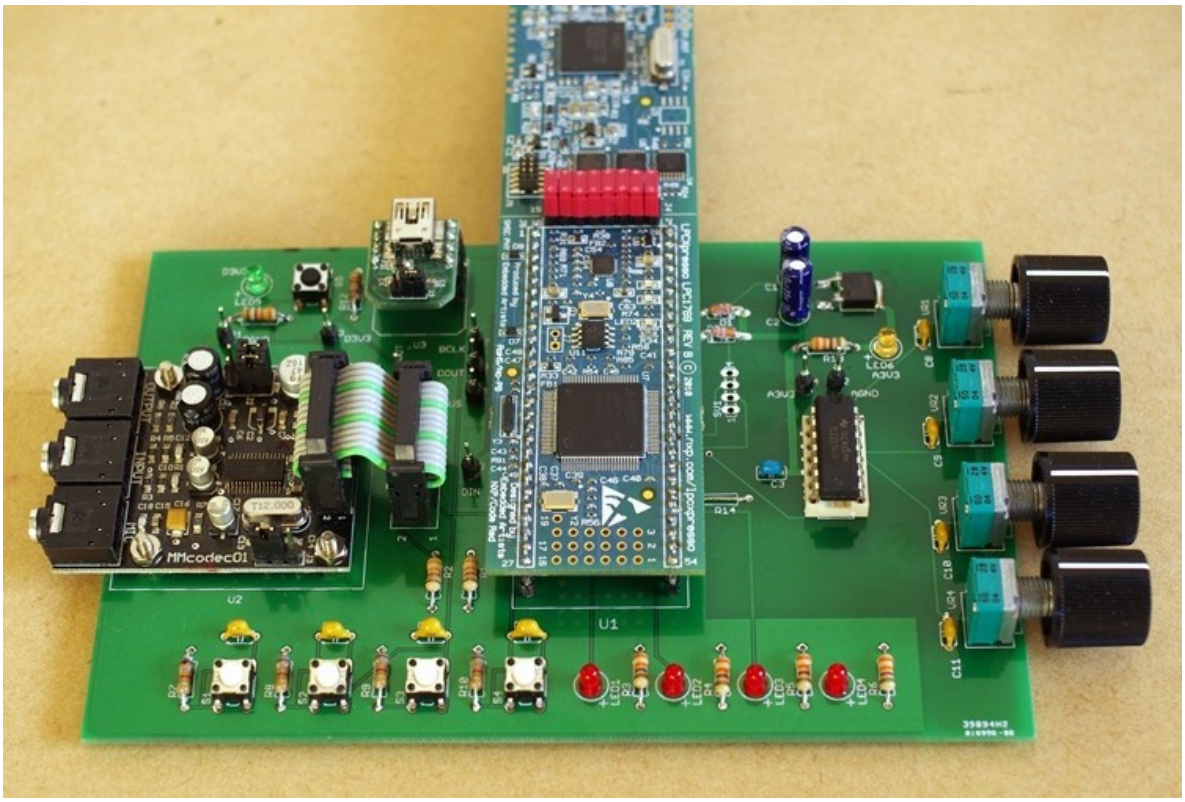

*図 1:Kirishima*

## 1.3.ドキュメントおよびプログラムの入手

Uzume のドキュメントおよびプログラムは、プロジェクト・ページからダウンロードできる。

- http://sourceforge.jp/projects/uzume/

## 1.4.関連情報

# 2. 開発環境の構築

Kirimshima 用の Uzume は、CodeRed 社の LPCXpresso 向け IDE によって開発されており、ユーザー・アプリケーションも同じ IDE で開発する。

以下ではこの IDE を中心とした開発環境の構築について解説する。OS は Ubuntu である。

## 2.1. Ubuntu のインストール

開発環境は Ubuntu 10.04 LTS である。Ubuntu のダウンロードページから 10.04 LTS のイメージをダウンロードしてインストールして使う。なお、イメージは 32 ビット版（PC (Intel x86) desktop CD）を使う。

Ubuntu のインストール方法は非常に多くの解説がネット上に存在するので、ここでは特に説明しない。なお、実機のほか VMware Player にインストールしても使うことができる。

## 2.2. 必須パッケージのインストール

Ubuntu のインストールが終了したら、端末ウインドウから以下のコマンドを実行して、必須パッケージをインストールする。このコマンドはインストールのほか、kermit の設定ファイルも作成している。

```
sudo apt-get install doxygen g++ ckermit libboost-all-dev lv
echo set line /dev/ttyUSB0 > ~/.kermrc
echo set speed 57600 >> ~/.kermrc
echo set parity none >> ~/.kermrc
echo set flow-control none >> ~/.kermrc
echo set carrier-watch off >> ~/.kermrc
```

## 2.3. FTDI ドライバのインストール

Uzume プラットフォームは、オーディオ用であるため、UART 接続は必ずしも必要ではない。しかし、接続できるようにしておけば、デバッグなどに便利である。

Kirishima ボードの UART は、FTDI 社の UB232R モジュールを使用している。これは USB—シリアル変換モジュールであるため、FTDI ドライバが必要になる。FTDI ドライバは Ubuntu の標準品ではなく、FTDI 社が提供しているものをつかう。これは、LPCxpresso 1768 の IDE が、Ubuntu の標準ドライバと衝突するためである。

FTDI 社のドライバは以下のページからダウンロードできる。Libftdi2xx をダウンロードし、説明に従ってインストールする。

- http://www.ftdichip.com/Drivers/D2XX.htm

## 2.4. IDE のインストール

IDE(統合開発環境)には、CodeRed IDE の LPCXpresso 版を使用する。これは Eclipse ベースの開発環境である。インストール方法については公式サイト等を参照のこと。

- http://www.code-red-tech.com/lpcxpresso.php

なお、この文書を執筆した時点で動作を確認したのは ver 3.x 系列のみである。Ver 4.x 系列で不具合がでるようなら、IDE ver 3.x 系列を試していただきたい。

## 2.5. Eclox のインストール

Uzume for LPCxpresso が利用している TOPPERS/ASP for LPC や、そのサブシステムは、Doxygen によるドキュメントを利用している。Uzume を使う上でこれらのドキュメントは必須ではないが、API ドキュメントとして参照できれば便利である。

Eclox は Doxygen を Eclipse から使うためのプラグインであり、パラメータ設定のためのウィザードや、ワンクリックでドキュメントを生成するためのメニュー・プラグインを持っている。

Eclox をインストールするには IDE の Help メニューから Install New Software...を選択する。図 3 のように Install ダイアログが現れるので、Work with フィールドに http://download.gna.org/eclox/update を入力して、エンターキーをおす。

しばらくすると、下のリストが”pending”から”eclox”に切り替わるので、チェックを入れて Next ボタンを押す。あとは、メッセージにしたがってダイアログを進めていけばインストールが終わる。

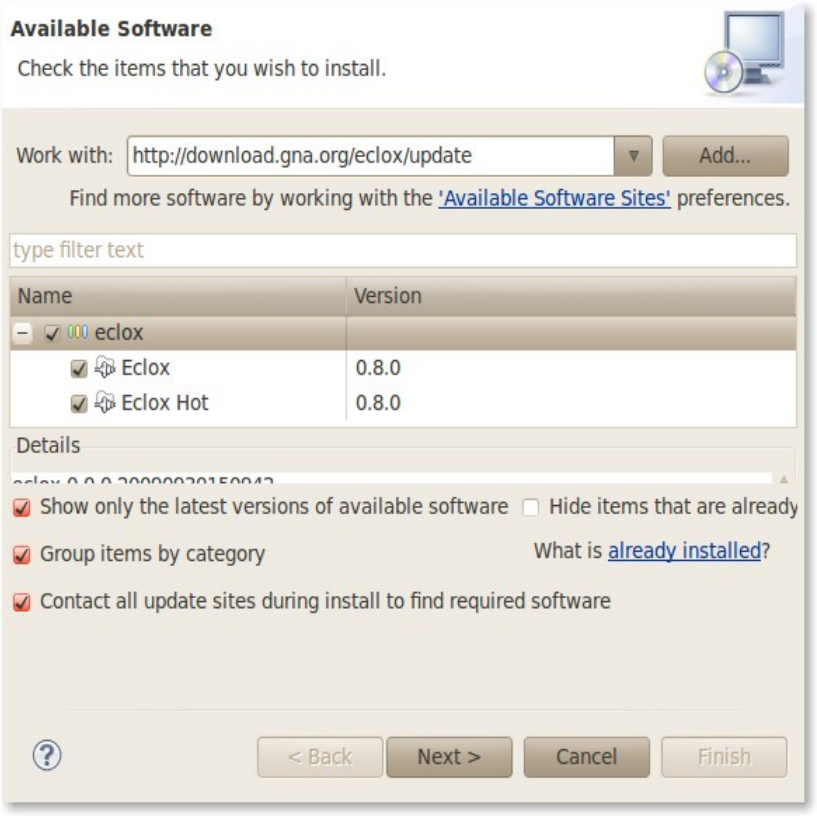


*図 2:Eclox のインストール*

## 2.6. Subversive のインストール

（Uzume を使用するだけであれば、この節は読み飛ばしても良い）

Uzume を使用するだけならばリリース済みファイルをダウンロードすればよい。しかしながら、最新版のソースを使いたい場合や、Uzume の開発に参加したい場合には Subversion レポリトリにアクセスすることになる。リポジトリとは、Uzume のソースコードを保存している一種のデータベスである。リポジトリを IDE に登録すれば常に自分の開発環境をリポジトリと同期した状態で開発をすすめることができる。

Subversive は Subversion レポジトリにアクセするための Eclipse プラグインである。インストールしておけば、CodeRed IDE からレポジトリにアクセスできるようになる。

Subversive をインストールするには IDE の Help メニューから Install New Software...を選択する。図 3 のように Install ダイアログが現れるので、Work with フィールドに http://download.eclipse.org/releases/galileo/ を入力して、エンターキーをおす。

しばらくすると、下のリストに大量のプラグインが表示されるので、リストの直上のフィルタフィールドに”sub”と入力して絞り込みを行う。Subversive SVN Team Provider にチェックを入れて Next ボタンを押す。あとは、メッセージにしたがってダイアログを進めていけばインストールが終わる。

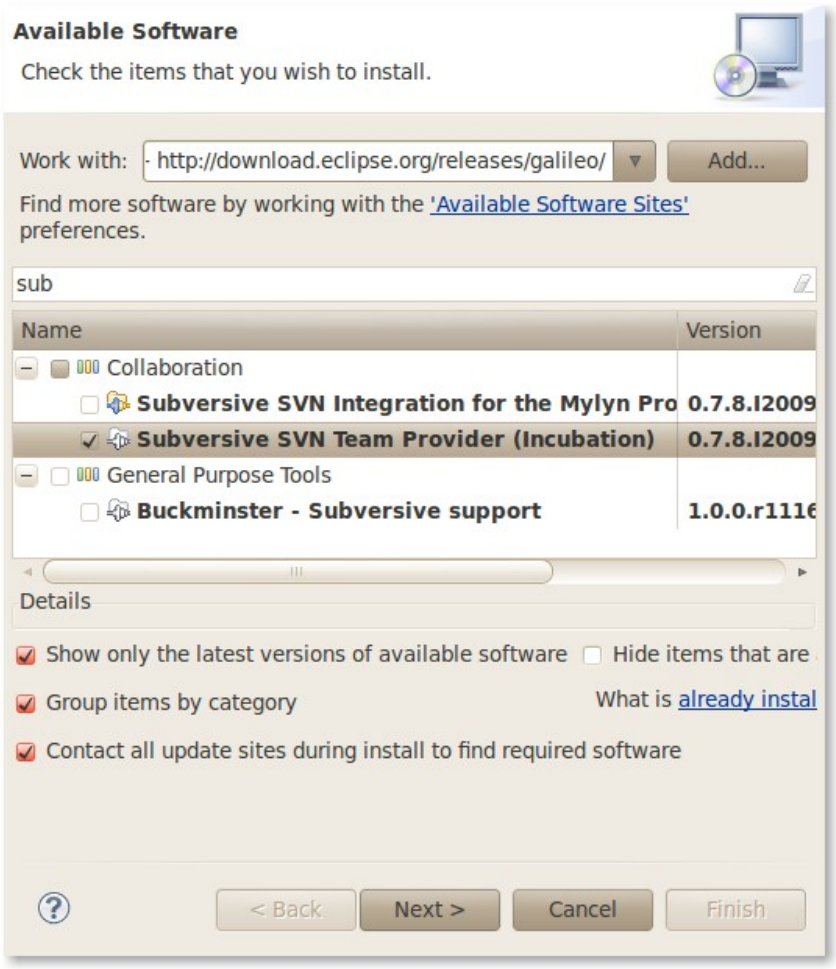


*図 3:Subversive のインストール*

## 2.7. Subversion レポジトリの登録

（Uzume を使用するだけであれば、この節は読み飛ばしても良い）

Subversive のインストールが終われば、Uzume のリポジトリの登録を行う。

リポジトリを登録するには、Window メニュー→ Open Perspective → Other... を選ぶ。するとパースペクティブの一覧が表示されるので、SVN Repository Exploring を選ぶ（図 4）。

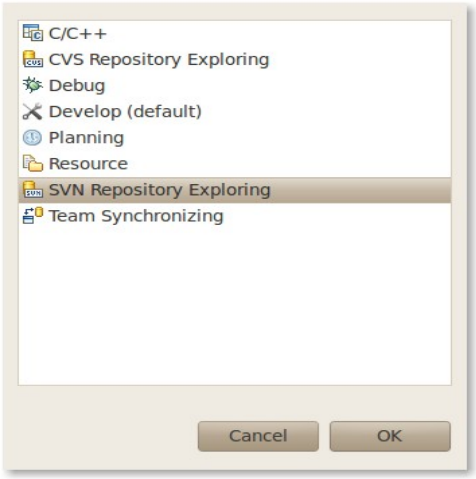


*図 4:レポジトリを選ぶ*

SVN Repository Exploring を初めてひらくと、Subversion Connector のインストール画面が現れる（図 5）。2011 年11月時点では、SVN Kit 1.3.0 を選ぶ。

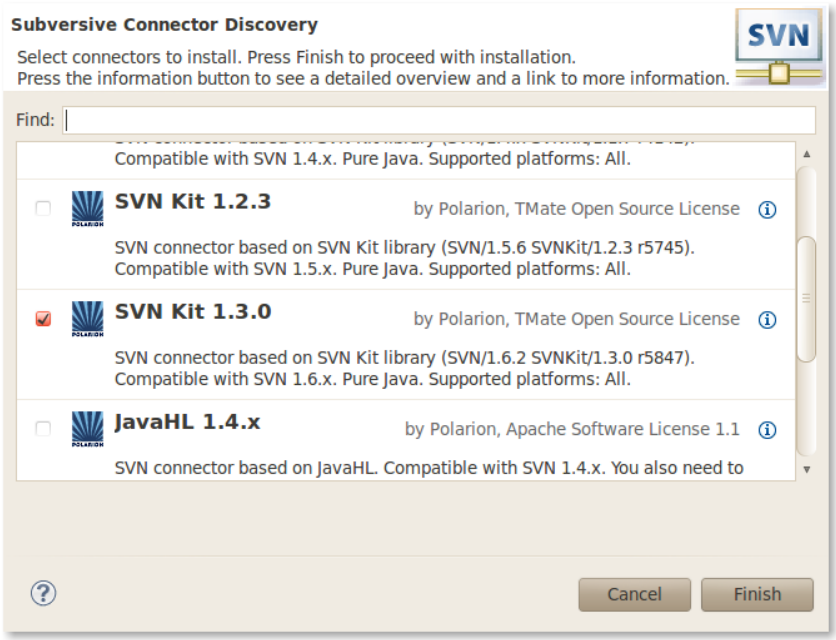


*図 5:SVN コネクタの選択*

SVN コネクタをインストールしたら、ようやく SVN レポジトリを登録できる。SVN Repository Explorer をもう一度開き、File メニュー→New→ Other ... を選ぶ。ダイアログが現れる。SNV→Repository Location を選ぶとリポジトリの登録ダイアログが現れる（図 6）。

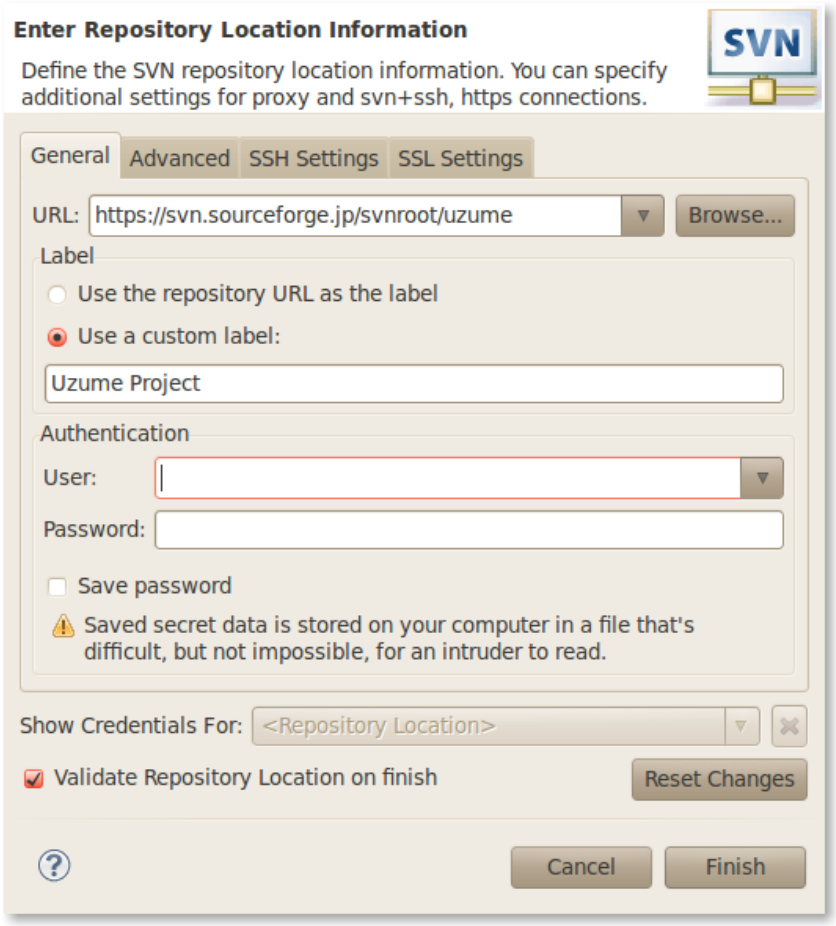


*図 6: リポジトリの登録*

このダイアログには、最低 URL だけ登録すればいい。URL は、開発者の場合コミットするために https を指定する。単に最新のコードを手に入れたいだけならば http で指定してもいい。

ラベルは任意である。指定しなければ URL がラベルになる。このラベルは人間が読むためのものなので分かりやすければなんでもいい。

Authentication も任意である。登録する場合は、sourceforge へのログオン情報を記録する。

Finish ボタンを押すと、IDE が登録作業を開始する。新しいサイトを信用してもよいか問われるので Trust ボタンを押して信用するよう指示する。

## 2.8. プロジェクトのチェックアウト

（Uzume を使用するだけであれば、この節は読み飛ばしても良い）

SVN Repository Exploring にリポジトリを登録すると、簡単にプロジェクトをチェックアウトしてワークスペースで使えるようになる。チェックアウトとは、リポジトリ内部のリソースをローカルの作業領域にコピーすることである。

チェックアウトをするには、リポジトリ・エクスプローラパネルを開く(図 7)。チェックアウトしたいリソースを左クリックし、コンテキストメニューから”Check Out”を実行すれば終わりである。

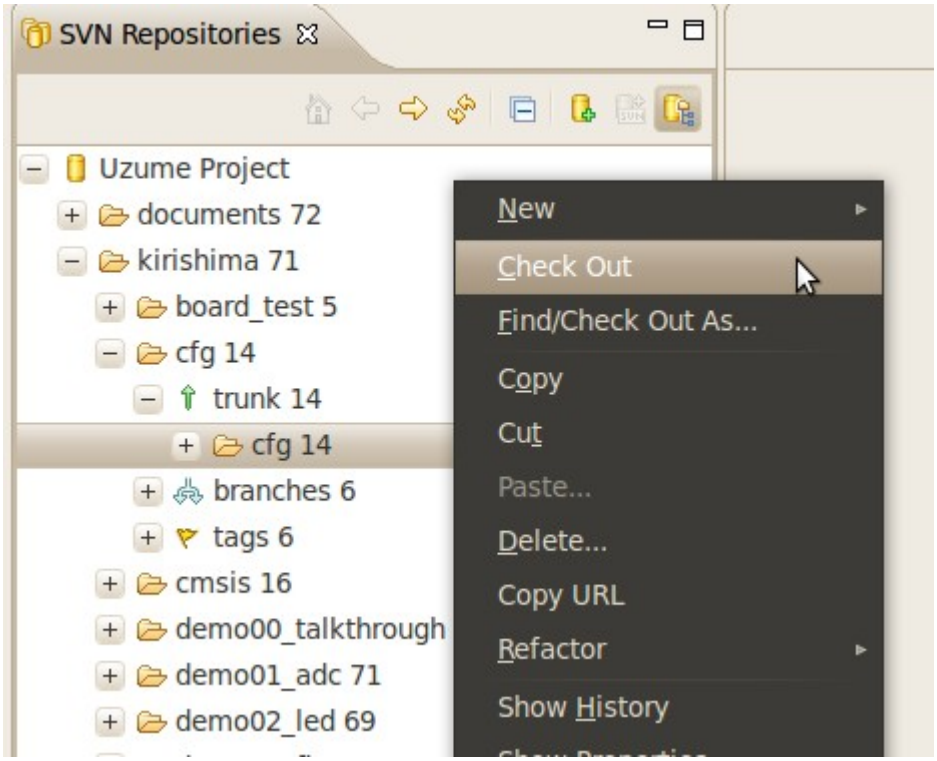


*図 7: リポジトリのチェックアウト*

Uzume プロジェクトでは、プロジェクトの最新コードはリポジトリの中に次のような規則で格納されている。

/基板名/プロジェクト名/trunk/プロジェクト名/

例えば Kirishima 基板用の demo01_adc プロジェクトの最新版は /kirishima/demo01_adc/trunk/demo01_adc/ である。したがって、これをチェックアウトすることで、ワークスペースに最新のコードがコピーされる。

なお、Kirishima 基板用のアプリケーションをビルドするには以下のプロジェクトが必要である。

- demo00_talkthrough : アプリケーションのスケルトンとして使えるサンプルプログラム。
- uzume : Uzume プラットホーム本体のライブラリ
- subsystems : 下位のユーティリティルーチン群
- toppersasp : uzume と subsystems が使用するインクルードファイル群
- cmsis : ARM/NXP の CMSIS ライブラリ
- cfg : TOPPERS/ASP のコンフィギュレータ

# 3. Uzume のインストール

開発環境の準備が整ったら、次は Uzume プラットフォームのインストールである。

Uzume はソースコードの形式で配布されているが、Kirishima で使用している LPCXpresso 1768/1769 の場合は、CodeRed IDE が読み込むことのできるプロジェクトとして配布されている。以下では、このソースコードを入手し、ビルド、実行する手順を説明する。

## 3.1. Uzume の入手

Uzume のソースコードは、プロジェクトのリリース・ページから取得できる。

ソースコードは tar.gz 形式で配布されており、複数のプロジェクトを含んでいる。ダウンロード後、ソースコードは展開せずにとっておき、次節に説明する方法でインポートする。

以下の説明では Uzume_Kirishima_20111124.tar.gz を使用する。日にちの部分が違っても特に記さない限り注意しなくていい。

## 3.2. ワークスペースへのインポート

Uzume を使うには IDE のワークスペースへソースコードをインポートしなければならない。

インポートに当たっては、IDE によって空のワークスペースを用意しておく。空のワークスペースがなければ IDE によって作っておく。

インポートを行うには、Resource Explorer あるいは Develop Explore から右クリックを行い、コンテキストメニューの Import を選ぶ。すると、ダイアログが開くので、Existing Project into Workspace を選択する(図 8)。Archive File を選ぶと失敗するので注意する。

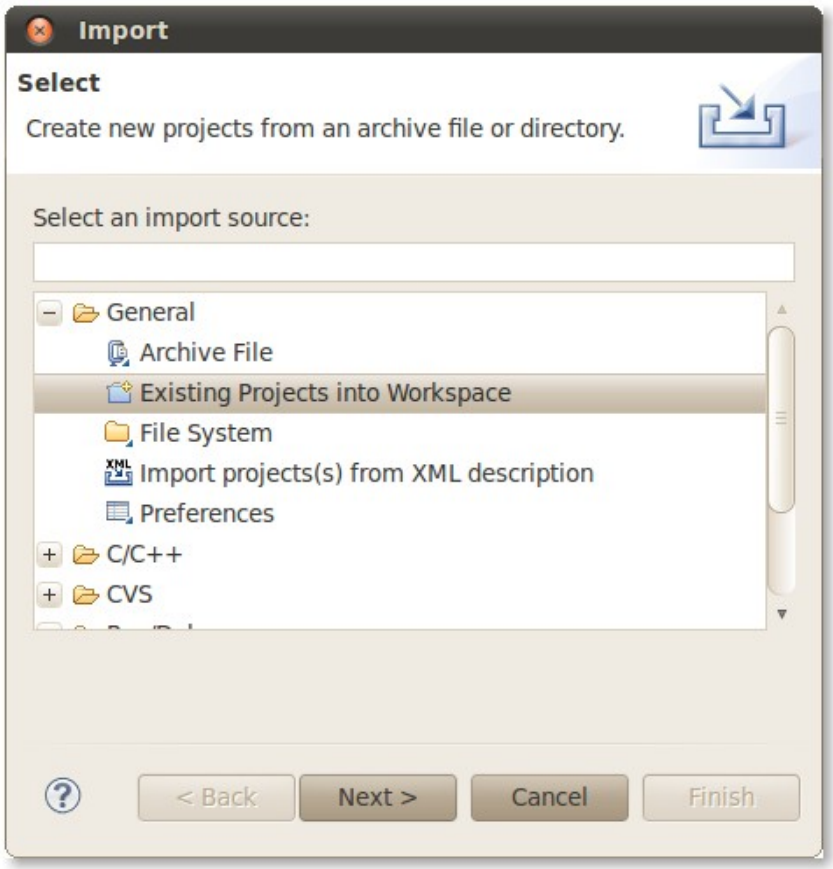


*図 8:インポート方法の指定*

次に現れるダイアログで、tar.gz ファイルを指定すると、tar.gz ファイル内部のプロジェクトが表示される。これらをすべて選択し、インクルードを行う。

インポート元のファイルには複数のプロジェクトが入っています。特に理由がない限り、すべてのプロジェクトをインポートしてください（図 9）。

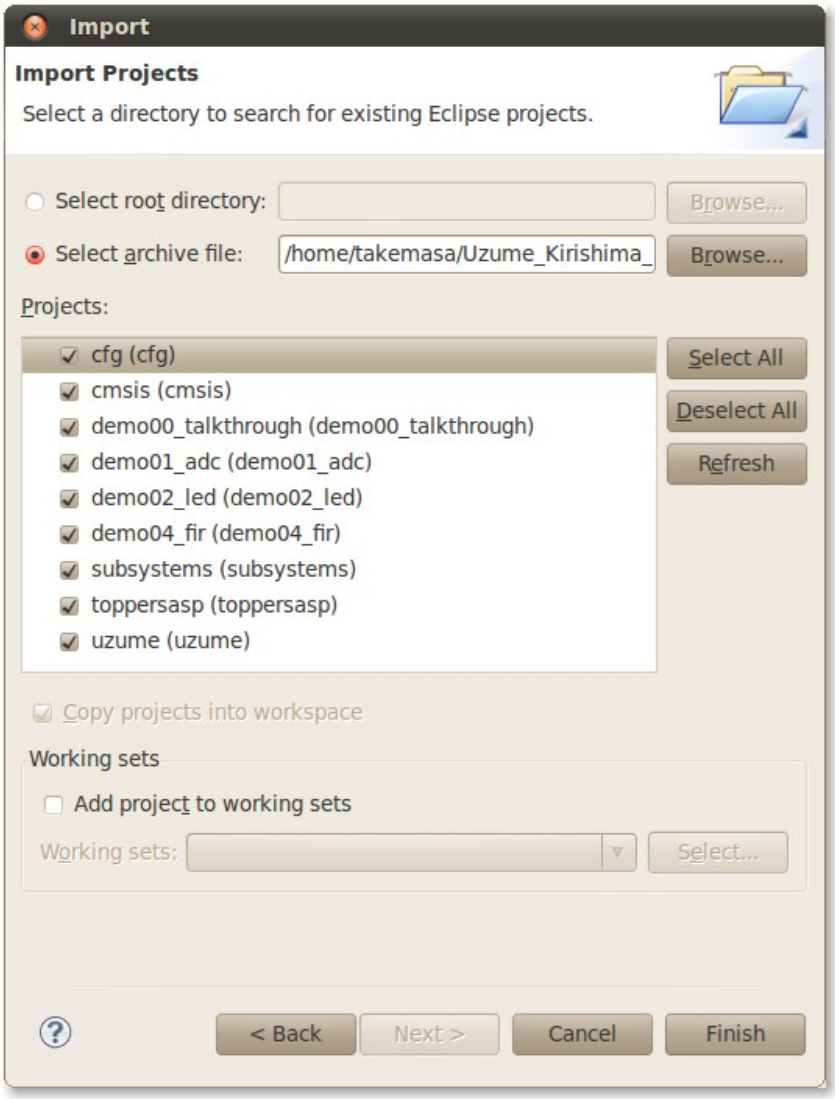


*図 9:読み込むコードの選択*

### 3.3. ビルド

ビルドを行うには、インポートしたプロジェクトの中から、アプリケーション・プロジェクトを選択する。ここでは、demo00_talkthrough を選択する。次に、右クリックでメニューに現れる Build Project を実行する。これで全部である。必要な関連プロジェクトのビルドが自動的に行われ、リンクが終了するとビルドの完了である。

ビルドについては、クイック・パネルの Build Uzme を実行してもよい。

Uzume フレームワークの各プロジェクトは、依存性に基づいて連携している。そのため、アプリケーションをビルドすると、必要に応じて他のプロジェクトも自動的にビルドされる。手作業で一つ一つビルドする必要はない。

## 3.4. 実行

ビルドしたコードの実行は、プロジェクトのなかの demo00_talkthrough Debug.launch を右クリックし、コンテキスト・メニューの中から Debug... → Debug As … → Demo00_talkthrough.Debug を選べばよい。自動的にデバッガが立ち上がり、ターゲットにプログラムを書き込んだあと、最初のブレークポイントで停止する。

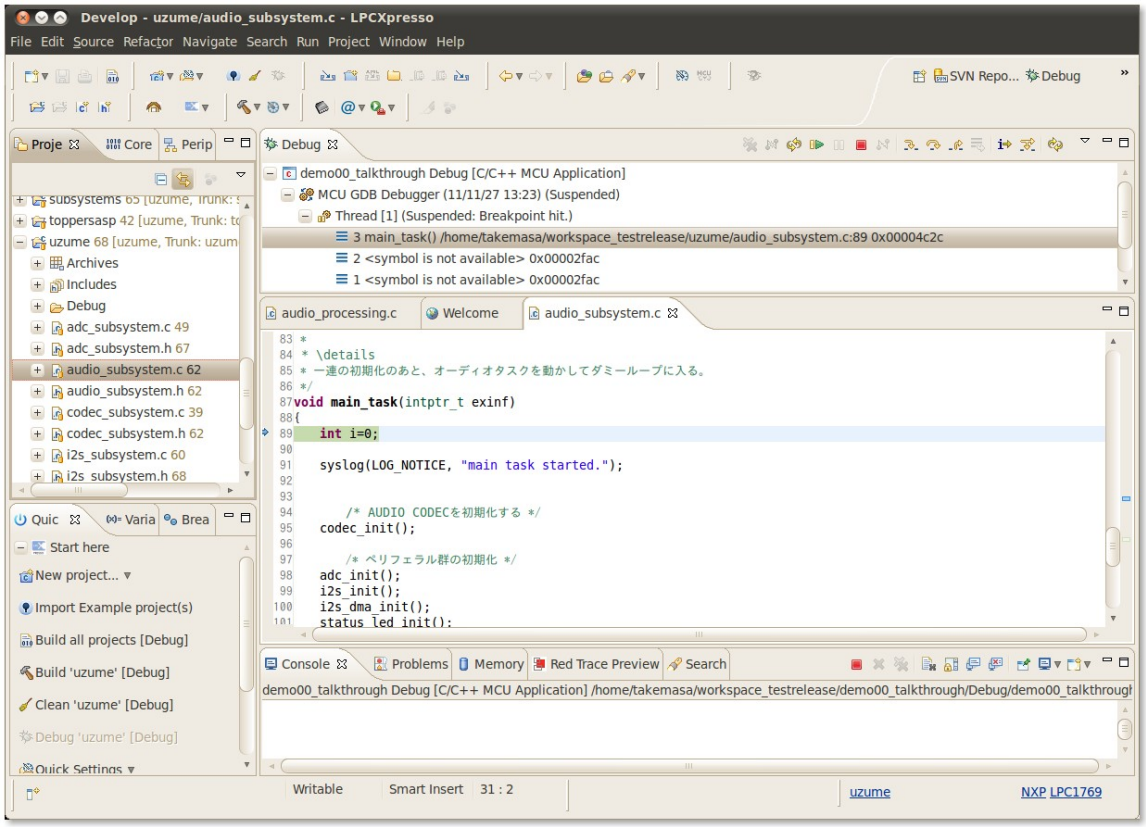


*図 10: デバッグ画面*

実行すると、Kirishima の Input オーディオジャックから入力した音が、Outpu オーディオジャックから出力される。

デバッガの使い方は IDE としては標準的なものである。Eclipse のデバッガに関する資料は書籍にもネット文書にも豊富にあるのでそれらを参考にするとよい。

## 3.5. 各プロジェクトについて

Uzume のフレームワークはいくつかの要素から成り立っており、それぞれ独立したプロジェクトである。以下にそれぞれのプロジェクトの概要を説明する。

### 3.5.1. demo##_XXXX

サンプル・アプリケーションである。LPC1768/1769 用の実行コードはこのアプリケーションが生成する。ユーザー・アプリケーションを開発するときには、サンプル・アプリケーションを改造する。

実際には信号処理用コールバック群だけがソースコードとして含まれ、Uzume の核となるタスク群は外にある。サブディレクトリに TOPPERS/ASP カーネルを持っているが、これは現時点でカーネルのライブラリ化に手間取っているためである。

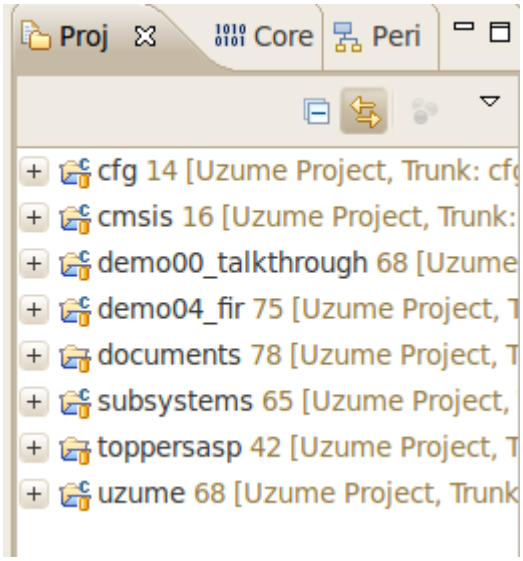


*図 11: プロジェクト一覧*

### 3.5.2. uzume

Uzume フレームワーク本体をライブラリ化したプロジェクト。

ペリフェラルの初期化、DMA の起動、ダブルバッファの管理、ADC タスクの実行などはすべてこのフレームワーク内部で行う。

### 3.5.3. subsystems

I2C API を提供するライブラリ・プロジェクト。

TOPPERS/ASP for LPC プロジェクトの成果物をほぼそのまま利用している。変更したのはコメントで、Doxygen ドキュメントを Uzume に整合させるための変更である。

### 3.5.4. toppersasp

TOPPERS/ASP カーネル 1.7.0 のソースコード。

uzume および subsystems プロジェクトが参照するためのインクルードファイル群としてここにおいてある。将来的にはカーネル自身をライブラリ化してアプリケーション・プロジェクトからカーネルコードを追い出す予定である。その場合、このプロジェクトがライブラリになる。

#### 3.5.5. cmsis

ARM/NXP の CMSIS ライブラリ。

CMSIS が提供する関数群をライブラリ化したものである。CMSIS ver 2.0 には、i2c のコールバック関数を呼び出さないバグがあるため、これを修正している。なお、このライブラリは TOPPERS/ASP for LPC でビルドしたものをそのまま使用している。

#### 3.5.6. cfg

TOPPERS/ASP のコンフィギュレーター。

X86 をターゲットするプロジェクトであり、生成コードは Linux 上で動作する。コンフィギュレータはタスクやセマフォ、割り込みサービスルーチンをアプリケーションに追加したい場合に使用する。通常は使用しなくてもよい。

### 3.6. Doxygen 文書を生成する

Eclox をインストールしているならば、Doxygen 文書の生成は簡単である。

IDE の Resource Explorer から、プロジェクト名を右クリックし、メニューの中から Build Documents を選択する。生成された文書は HTML サブディレクトリに保存される。

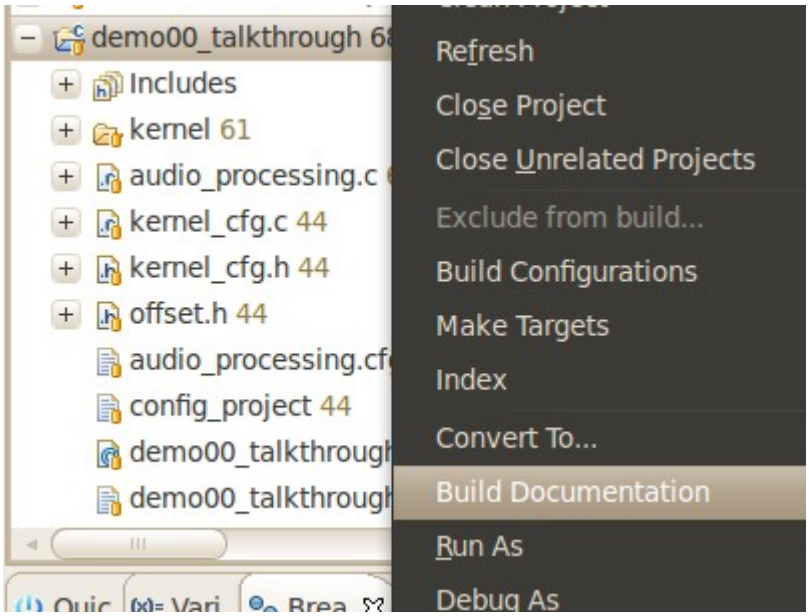


*図 12:ドキュメントの生成*

# 4. 自分自信のアプリケーションを書く

この章では Uzume プラットフォームに独自のユーザー・アルゴリズムを実装する方法を説明する。

Uzume はユーザー・アルゴリズムをシンプルな形で実装することが可能であり、わずかな記述でオーディオ信号処理を行うことができる。この章では、オーディオ信号処理の方法を解説した後、ボリュームやスイッチといったインターフェースの使い方を説明する。

なお、Uzume API Reference に使用可能な API の一覧があるので参考にされたい。

## 4.1. Uzume プログラミングの概要

Uzume フレームワークは簡単な手続きでオーディオ信号処理の実験ができるように構築されている。簡単な処理なら、関数をひとつ書き替えるだけで実装できる。

アプリケーション開発時にユーザーが書き換えなければならないファイルは audio_processing.c である。このファイルには Uzume のコールバック関数群のスケルトンが収めてあり、ユーザーはこれを書き換えるだけで信号処理を実装できる。

アプリケーション開発時には新しいアプリケーション・プロジェクトを作るのではなく、既存のでも・プロジェクトを書き換えることを推奨する。Uzume は多くのサブシステムを使っているうえに、TOPPERS/ASP はインクルード・パスが入り組んでいることからプロジェクトの設定は複雑である。そのため、新しく作るのは労力が掛り過ぎる。

デモ・アプリケーションに自分が好きな名前をつけたい時には、プロジェクト・エクスプローラからプロジェクトを右クリックし、コンテキスト・メニューの Rename を実行する。注意すべき点として、コピー＆ペーストではうまくいかないことが挙げられる。IDE が、内部のインクルード・パス設定などをプロジェクト名に合わせて変更するのは、Rename を行った時だけに限られる。

この方法では、デモ・アプリケーションをひとつ潰してしまう。アプリケーション開発時もデモ・アプリケーションが欲しい場合には、改めてそのプロジェクトをファイルからインポートすれば良い。

Uzume フレームワークは性能評価機構を実装することで、ユーザー信号処理が MIPS をどの程度消費するのか測定することもできる。

スイッチ・コールバックは現時点では未実装であるが、将来的には audio_processing.c の中にコールバックのスケルトンが配置される予定である。

## 4.2. 信号処理アルゴリズムの実装

信号処理アルゴリズムは、process_audio.c の中の audio_processin()関数に実装する（図 13）。

process_audio()関数は Uzume の中心となるコールバック関数で、オーディオ信号の DMA 転送が終了するたびにフレームワークから呼び出される。引数して与えられるのは input[2][]と、output[2][]の二つの

配列へのポインタである。いずれも 2 次元配列で 2 行からなる。左右チャンネルデータはそれぞれの行に分けられて格納する。

行のインデックスとしては LCH、RCH がはじめから用意されているのでそれを使えばいい。LCH を使えば左チャンネルのデータにアクセスでき、RCH を使えば右チャンネルのデータにアクセスできる。たとえば、input[LCH][3]は入力バッファ中の 3 番目の左データにアクセスできる。列のインデックスは C 言語同様 0 から始まり、UZME_BLOCK_SIZE-1 までである。

audio_processing.c

```
32void process_audio( const AUDIO_SAMPLE input[2][UZUME_BLOCK_SIZE], AUDIO_SAMPLE out
33{
34    int i;
35
36        /*
37         * 入力データはinput[][]に格納されている。input[LCH][]が、左チャンネル、
38         * input[RCH][]が右チャンネルである。それぞれのチャンネルにはUZUME_BLOCK_SIZE個の
39         * データが格納されている。各データは2の補数符号付き左詰め形式の整数
40         * （つまり1.15形式固定小数点数）である。
41         *
42         * 出力データの形式も同じである。
43         */
44    for ( i = 0; i< UZUME_BLOCK_SIZE; i++ )
45    {
46            // 単なる入出力コピー
47        output[LCH][i] = input[LCH][i];
48        output[RCH][i] = input[RCH][i];
49    }
50}
51
```

*図 13:process_audio 関数*

UZUME_BLOCK_SIZE マクロは、uzume.h で宣言されるマクロで、一度の DMA 転送で転送されるサンプル数（左右を合わせて1と数える）を示す。Kirishima 版の Uzume は 48kHz サンプルであるので、UZUME_BLOCK_SIZE が 48 ならば、1m 秒ごとに DMA 完了割り込みが入り、この関数が呼ばれる。

プログラマが audio_processing()で最低限やらなければならないことは、input[][]から output[][]へのデータコピーである。データコピーだけしかしなければ、図 13 同様、入力から出力へのデータをコピーするだけの TalkThrough プログラムになる。

コピーを行う際に入力データに処理を施せば、それがオーディオ信号処理になる。例えば、入力データを符号に合わせて最大振幅にさせる処理をすれば、一種のエフェクタになる（歪系の処理はエイリアスが帯域内に落ちてくるので注意が必要である）。

### 4.3. ポーリングによる VR の読み取り

Uzume フレームワークはポーリングによる VR の読み取りを提供する。

VR の値はは ADC 読み取り API から読み込むことができる。読み取りに使用する関数は adc_read()で、引数には VR の番号をあたえる。VR の番号は UZUME_VR1 から始まり、uzume.h で宣言されている。

Kirishima 実装では、adc_read()の返り値、すなわち ADC_SAMPLE 型には signed short 型であり、すなわち 16bit 整数である。Uzume の API リファレンスでは、ADC_SAMPLE 型には ADC の読み取り値を左詰めして格納することになっているすなわち、Uzume では ADC の値を符号付き固定小数点数として扱う。

言い換えると、C 言語としては ADC_SAMPLE 型は 0 から 32767 までの値を取る。しかし、Uzume はこれを 0 から 0.99997 までの値としてみるよう要請している。

固定小数点同士の加算は溢れが生じない限り整数型と同じであり、C 言語としての加算を行うことができる。しかし、乗算の場合、整数とは小数点位置が違うため、固定小数点数に整数乗算を行うと、結果のシフトが必要になる。

簡単には、符号付き 16bit 固定小数点数同士の乗算の場合、整数乗算のあとに結果を右に 15bit シフトすることで正しい結果を得ることができる。demo00_adc プロジェクトにはその例が示されている。

型や API に関する詳細は demo01_adc プロジェクト、および、API リファレンスを参照のこと。

## 4.4. ポーリングによるスイッチ状態の読み取り

## 4.5. 排他処理について

# 5. 内部構造とカスタム化、性能評価

この章では補足情報として Uzume の内部構造と、そのカスタム化、さらには性能評価 API について説明する。

## 5.1. 内部構造

## 5.2. ブロック・サイズの変更

## 5.3. タスクの一覧と優先順位

## 5.4. 割り込みの扱い

## 5.5. 独自タスクを追加する場合の指針

## 5.6. 性能評価 API

# 6. その他

## 6.1. 連絡先

Uzume への要望やバグ報告は http://sourceforge.jp/projects/uzume/ まで連絡されたい。

# 7. 文書履歴

### 7.1. 2011/11/11

オリジナル

### 7.2. 2012/1/9

IDE のバージョン情報を追加。