Uzume API Reference

Rev ~~0.9~~0.91

2011/Nov~~/4~~27 ~~20~~

Suikan / Uzume project

目次

1 はじめに

この文書は、Uzume プロジェクトで使用するフレームワークの API およびデータ型の仕様を定義する。なお、特定のプラットホームではなく、プロジェクトが対応する複数のプラットホームに共通の事項を扱う。

1.1 Uzume プロジェクトについて

Uzume プロジェクトは、低価格のオーディオ・信号処理プラットホームを提供するオープンソース・プロジェクトである。このプロジェクト低価格のマイコン、あるいは DSP に対して 48kHz、ステレオのオーディオ・プラットホームを提供し、信号処理に関心のあるプログラマが容易に実験が行える環境を整えることを目的としている。

Uzume プロジェクトは Sourceforge のホスティングサービスを利用している。URL は http://sourceforge.jp/projects/uzume である。

1.2 この文書の最新版の取得方法

この文書は Uzume プロジェクトにて配布している。最新の正規リリース版は常にプロジェクトのダウンロード・ページから入手できる。

2 データ型

2.1 オーディオ・サンプル型

オーディオ・サンプル型は AUDIO_SAMPLE として宣言される。AUDIO_SAMPLE の実体は、プラットホームによって異なる。AUDIO_SAMPLE 型は uzume.h で宣言される。

2.1.1 浮動小数点型の場合

AUDIO_SMAPLE 型を浮動小数点型として実装する場合、サンプル・データは(1.0, -1.0] の区間の単精度浮動小数点数として表現される。

2.1.2 32bit 整数の場合

AUDIO_SMAPLE 型を32 bit 整数型として実装する場合、サンプル・データは(2^31, -(2^31)]の区間の符号付き整数として表現される。これはオーディオ・データを右詰左詰めした表現である。

2.1.3 16bit 整数型の場合

AUDIO_SMAPLE 型を 16bit 整数型として実装する場合、サンプル・データは(2^15, -(2^15)]の区間の符号付き整数として表現される。これはオーディオ・データを右詰左詰めした表現である。

2.2 ADC サンプル型

ADC サンプル型は ADC_SAMPLE として宣言される。ADC_SAMPLE の実体は、プラットホームによって異なる。ADC_SAMPLE 型は uzume.h で宣言される。

2.2.1 浮動小数点型の場合

ADC_SMAPLE 型を浮動小数点型として実装する場合、サンプル・データは(1.0, 0] の区間の単精度浮動小数点数として表現される。

2.2.2 32bit 整数型の場合

ADC_SMAPLE 型を32 bit 整数型として実装する場合、サンプル・データは(2^31, 0]の区間の符号付き整数として表現される。これは ADC データを右詰左詰めした表現である。

2.2.3 16bit 整数型の場合

ADC_SMAPLE 型を32 bit 整数型として実装する場合、サンプル・データは(2^15, 0]の区間の符号付き整数として表現される。これは ADC データを右詰左詰めした表現である。

2.3 DAC サンプル型

DAC サンプル型は ADC_SAMPLE として宣言される。DAC_SAMPLE の実体は、プラットホームによって異なる。DAC_SAMPLE 型は uzume.h で宣言される。

2.3.1 浮動小数点型の場合

DAC_SMAPLE 型を浮動小数点型として実装する場合、サンプル・データは(1.0, 0] の区間の単精度浮動小数点数として表現される。

2.3.2 32bit 整数型の場合

DAC_SMAPLE 型を32 bit 整数型として実装する場合、サンプル・データは(2^31, 0]の区間の符号付き整数として表現される。これは DAC データを右詰左詰めした表現である。

2.3.3 16bit 整数型の場合

DAC_SMAPLE 型を32 bit 整数型として実装する場合、サンプル・データは(2^15, 0]の区間の符号付き整数として表現される。これは DAC データを右詰左詰めした表現である。

3 API

3.1 ADC 関数

ADC API 関数を使うには、ヘッダ・ファイル uzume.h をインクルードする。関数はスレッド・セーフである。

3.1.1 int adc_get_value()

返り値

ADC_SAMPLE

引数

int adc_channel

解説

引数 adc_channel で表される ADC の入力チャンネルからの信号を読み取る。読み取った数値は ADC_SAMPLE 型として返り値に与えられる。adc_channel には 0 オリジンの正の整数をあたえる。adc_channel の上限はプラットホーム依存である。

3.2 I2C 関数

I2C API 関数は TOPPERS/ASP for LPC プロジェクトで定義した API と同じ関数プロトタイプをもつ。関数を使うには、ヘッダ・ファイル uzume.h をインクルードする。関数はすべてスレッド・セーフである。

3.2.1 int i2c_master_write()

返り値

正常に終了したなら 0 を返す。

引数

int peripheral

int slave

const unsigned char write_data[]

int write_count

解説

I2C コントローラ・ペリフェラルからスレーブに対して書き込みを行う。関数は内部でブロックされ、スレーブへの書き込みが終了したら戻ってくる。

引数 peripheral は MCU 内部の I2C コントローラ・ペリフェラルの番号である。たとえば、MCU が I2C0、I2C1、I2C2 の三つの I2C コントローラ・ペリフェラルをもつとき、それぞれを使用する際に peripheral に渡すべき値は 0, 1, 2 である。MCU が一つしか I2C コントローラ・ペリフェラルをもたない場合には、この引数は無視されるが、互換性の見地から 0 を渡すことが推奨される。

引数 slave は7ビットで表す I2C スレーブ・アドレスである。アドレスは右詰左詰めの int 型で表す。

引数 write_data[]は、スレーブに送信すべきデータ列である。

引数 write_count はスレーブに送信すべきデータの個数である。単位は BYTE である。

3.2.2 int i2c_master_read()

返り値

正常に終了したなら 0 を返す。

引数

`int` peripheral

`int` slave

`unsigned char` read_data[]

`int` read_count

解説

I2C コントローラ・ペリフェラルからスレーブに対して読み込みを行う。関数は内部でブロックされ、スレーブからの読み込みが終了したら戻ってくる。

引数 peripheral は MCU 内部の I2C コントローラ・ペリフェラルの番号である。たとえば、MCU が I2C0、I2C1、I2C2 の三つの I2C コントローラ・ペリフェラルをもつとき、それぞれを使用する際に peripheral に渡すべき値は 0, 1, 2 である。MCU が一つしか I2C コントローラ・ペリフェラルをもたない場合には、この引数は無視されるが、互換性の見地から 0 を渡すことが推奨される。

引数 slave は7ビットで表す I2C スレーブ・アドレスである。アドレスは右詰左詰めの int 型で表す。

引数 read_data[]は、スレーブから読み込んだデータを格納する配列である。

引数 read_count はスレーブから読み込むべきデータの個数である。単位は BYTE である。

3.2.3 `int` i2c_master_write_read()

返り値

正常に終了したなら 0 を返す。

引数

`int` peripheral

`int` slave

`const unsigned char` write_data[]

`int` write_count

`unsigned char` read_data[]

`int` read_count

解説

I2C コントローラ・ペリフェラルからスレーブに対して書き込みを行う。関数は内部でブロックされ、スレーブへの書き込みが終了したら戻ってくる。

引数 peripheral は MCU 内部の I2C コントローラ・ペリフェラルの番号である。たとえば、MCU が I2C0、I2C1、I2C2 の三つの I2C コントローラ・ペリフェラルをもつとき、それぞれを使用する際に peripheral に渡すべき値は 0, 1, 2 である。MCU が一つしか I2C コントローラ・ペリフェラルをもたない場合には、この引数は無視されるが、互換性の見地から 0 を渡すことが推奨される。

引数 slave は7ビットで表す I2C スレーブ・アドレスである。アドレスは右詰左詰めの int 型で表す。

引数 write_data[]は、スレーブに送信すべきデータ列である。

引数 write_count はスレーブに送信すべきデータの個数である。単位は BYTE である。

引数 read_data[]は、スレーブから読み込んだデータを格納する配列である。

引数 read_count はスレーブから読み込むべきデータの個数である。単位は BYTE である。

3.3 ステータス LED 関数

API を使用するには、uzume.h をインクルードする。

Uzume のステータス LED は 0 から始まる LED 番号をもっている。ステータス LED の制御はすべてこの LED 番号によっておこなう。関数はすべてスレッド・セーフである。

3.3.1 void status_led_on()

返り値

void

引数

int led

解説

引数 led で指定されたステータス LED を点灯して戻ってくる。

3.3.2 void status_led_off()

返り値

void

引数

int led

解説

引数 led で指定されたステータス LED を消灯して戻ってくる。

3.3.3 void status_led_set()

返り値

void

引数

int led

int state

解説

引数 led で指定されたステータス LED の状態を設定する。引数 stete の値が 0 なら消灯、非0なら点灯である。

3.3.4 int status_led_get()

返り値

int

引数

int led

解説

引数 led で指定されたステータス LED の状態を取得する。消灯状態であれば返り値は 0、点灯状態であれば返り値は非0である。

4 コールバック

4.1 オーディオ・コールバック

オーディオ・コールバック関数は audio_processing.c の中に置かれているユーザー向け関数である。この関数はオーディオ・フレームワークの中から定期的に呼び出される。この中に信号処理アルゴリズムを記述する。

オーディオ・コールバック関数は、タスク・コンテキストで呼び出される。

4.1.1 void process_audio( )

返り値

void

引数

const AUDIOSAMPLE input[2][AUDIOBUFSIZE/2]

AUDIOSAMPLE output[2][AUDIOBUFSIZE/2]

~~int count~~

説明

引数として与えられる input[][]は、オーディオ・コーデックから入力されるステレオ信号を格納している。input[LCH][]が左チャンネル、input[RCH][]が右チャンネルである。一回の呼び出しで処理すべきサンプル数は引数 ~~count~~UZUME_BLOCKSIZE で与えられる。

処理結果は output[LCH][]および output[RCH][]に格納する。process_audio()関数から戻ると、オーディオ・フレームワークは output[][]のデータをオーディオ・コーデックから出力する。

4.1.2 void init_audio(void)

返り値

なし

引数

なし

説明

この関数は Uzume フレームワークの初期化時に一度だけ呼び出される。呼び出しのコンテキストはタスク・コンテキストであり、かならず process_audio()よりも先に呼び出される。

process_audio()で使用する資源の初期化などに、必要に応じて使用する。

4.2 スイッチ・コールバック

スイッチ・コールバック関数は audio_processing.c の中に置かれているユーザー向け関数である。この関数はプッシュボタン・スイッチの状態が変わるたびに呼び出される。

スイッチ・コールバック関数はタスク・コンテキストで呼び出される。タスクの優先順位はオーディオ・コールバック関数のタスク優先順位より低い。

Uzume のスイッチは 0 から始まるスイッチ番号をもっている。スイッチ状態の通知はすべてこのスイッチ番号によっておこなう。

4.2.1 void button_down( )

返り値

void

引数

int switch

説明

引数 switch の番号に対応するスイッチが押下された事をアプリケーションに通知する。トグル・スイッチやシーソー・スイッチの場合には信号が L に遷移した場合にこのコールバック関数が呼び出される。

4.2.2 void button_up( )

返り値

void

引数

int switch

説明

引数 switch の番号に対応するスイッチが押下状態から戻された事をアプリケーションに通知する。トグル・スイッチやシーソー・スイッチの場合には信号が L に遷移した場合にこのコールバック関数が呼び出される。

5 優先順位

5.1 タスクの優先順位

Uzume フレームワークのタスク優先順位は以下のようになっている。上から下に優先順位が下がっていく。

- audio_task()
- adc_task()
- switch_task()
- logtask()
- main_task()

logtask()は、シリアル・ポートからのログ出力用タスクである。

5.2 割り込みの優先順位

割り込みの優先順位は規定しない。

Uzume フレームワークはオーディオ帯域で動作し、かつ、オーディオ信号は DMA 経由で取り扱う。そのため、割り込みの優先順位は大きな問題にならないはずである。

割り込み優先順位が大きな問題になる場合は、割り込みの処理時間が長すぎる事が考えられる。その場合、割り込みハンドラの再設計が必要である。

6 文書履歴

6.1 Rev 0.9 2011/Nov/4

最初のリリース。

6.2 Rev 0.91 2011/Nov/27

- process_audio()関数から count 引数を削除。配列を作るときどに変数でサイズを指定するのは都合が悪いので、UZUME_BLOCK_SIZE 定数を使うように変更した。
- init_audio()を追加。
- データ型の右詰め、左詰めに関する記述の誤りを修正した。