KOBANZAME SDK

Version 1.40

Software Developers Kit for
“KOBANZAME” Blackfin Evaluation Board

目次

1 はじめに

1.1 このドキュメントについて

このドキュメントは KOBANZAME Blackfin ボード上で開発を行うソフトウェア開発者に向けてかかれたものである。C 言語の知識はすでにあるものとして書かれている。

文中の URL は 2010 年 1 月 5 日現在有効な URL である。

このドキュメントは以下の SDK　Version に対応している

- 1.40

1.2 KOBANZAME SDK について

KOBANZAME SDK とは、AnalogDevices 社製 DSP BF-533 の評価基板である KOBANZAME 用のソフトウェア・デベロッパーズ・キット(SDK)である。この SDK には RTOS(uITRON) / Filesystem / Device Driver が含まれている。

1.3 KOBANZAME(評価基板)について

KOBANZAME とは株式会社ジェイパーソンが製造・販売する AnalogDevices 社 BF-533 の評価基板である。BF-533 の他に、2Mbyte の SPI フラッシュ、32MByte の SDRAM、2ch ステレオのオーディオ入出力、加速度センサ、Micro SD-Card Slot を搭載している。
詳しくはこちらのページを参照のこと
http://kobanzame.j-person.com/bf533/

1.4 開発環境

VDSP++で使用する場合、この SDK は以下の開発環境で動作確認をする

- Windows XP SP2
- Visual DSP++ 5.0 Update7
- HPUSB-ICE

gcc で使用する場合、この SDK は以下の開発環境で動作確認をする

- Windows XP SP2
- VMware + UBUNTU8.04LTS

2 ライセンス

各々のソフトウェアのライセンス条文を守ること。基本的にすべてのソフトウェアは無償で提供されている。

2.1 RTOS 共通部 ... TOPPERS/JSP 1.4.3

µITRON4.0 準拠の RTOS。ライセンス形態は TOPPERS ライセンスに準拠。
こちらを参照のこと
http://www.toppers.jp/license.html

2.2 RTOS 依存部 ... TOPPERS/JSP for Blackfin

上記 TOPPERS の Blackfin ターゲット依存部を酔漢さんが設計された Toppers for Blackfin Release 3.1.1 を使用している。
http://sourceforge.jp/projects/toppersjsp4bf/
こちらも TOPPERS ライセンスに準拠する。

2.3 Filesystem … FatFs

Filesystem にえむるさんが設計された FatFs(R0.07e)を使用している。
http://elm-chan.org/fsw/ff/00index_j.html
ライセンスについてはこちらを参照すること。(最後に記載されている)
http://elm-chan.org/fsw/ff/ja/appnote.html

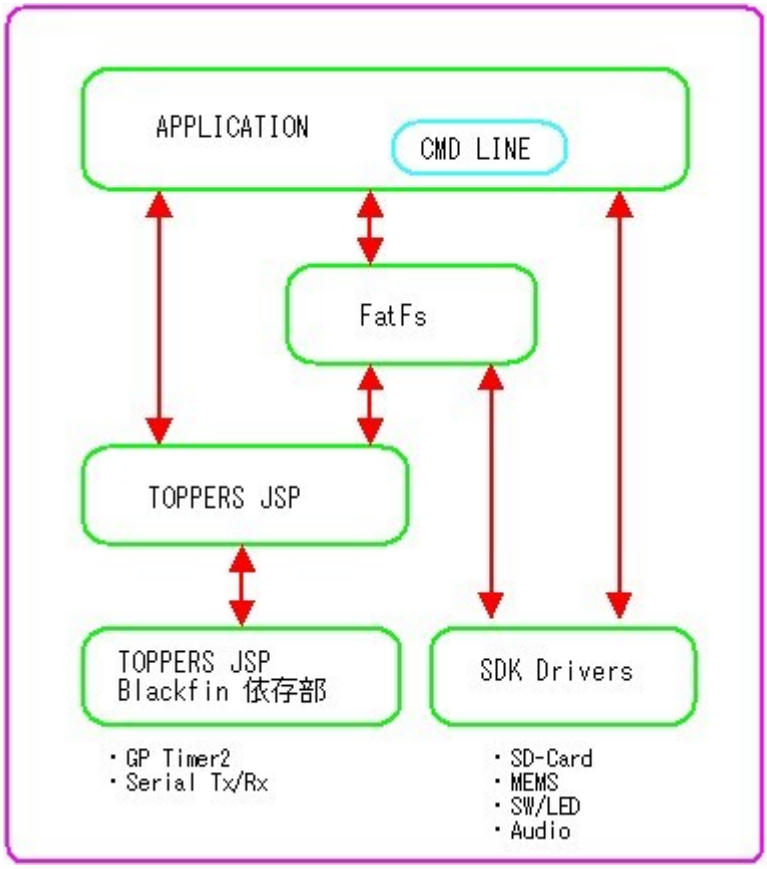
2.4 その他の SDK 付属ソフトウェア

ユーザー責任の下、商用・非商用にかかわらず無償で使用または配布することができる。ただしソースコードを配布する場合 Header 部分はそのまま記載をする必要がある。

3 ソフトウェア概要

3.1 ソフトウェアブロック図

KOBANZAME SDK のソフトウェアブロック図は以下の通りである



ユーザーは、SDK が提供するドライバー、または μITRON や Filesystem を通じてハードウェアにアクセスができる。

また、ユーザーは Command Line を使用することができる。これはシリアルを用い、UNIX の端末のようなインターフェイスを提供するアプリケーションである。ユーザーが独自にコマンドの登録ができる。(ただし動的にはできない)。デフォルトでは、Filesystem にアクセスする ls/ cd/ pwd などのコマンドがあり、または Device Driver にアクセスできる各種のコマンドも用意されている。

3.2 フォルダ・ファイル構成

SDK のフォルダ・ファイル構成を下記に示す。SDK 以外は省略している。

root

+cfg	…	cfg.exe を作成するためのプロジェクト
+doc	…	ドキュメント
+fatfs	…	Filesystem fatfs
+Flash Programmer	…	SPI Flash の Writer
+jsp	…	ROTS TOPPERS JSP と Blackfin 依存部
+kzsdk	…	KOBANZAME SDK
-cmdline.c	…	Command Line
-def_mma7455l.h	…	MMA7455L(MEMS)のレジスタ定義
-fscmd.c	…	Filesystem 用の Command
-fs_support.c	…	Filesystem 用のサポートコード
-inclist.txt	…	Project の include List。Geninc.exe を使用する
-kobanazame.h	…	SDK 標準インクルードファイル
-kz_malloc.c	…	HEAP for Multi-task
-kzconfig.h	…	SDK の USER Configuration ファイル
-kzdev.c	…	Device Driver : 全般
-kzdev.h	…	Device Driver : 全般
-kzdev_audio.c	…	Device Driver : Audio
-kzdev_codec.c	…	Device Driver : Audio Codec
-kzdev_mems.c	…	Device Driver : MEMS
-kzdev_mmc.c	…	Device Driver : MMC / SD-Card (SDHC 対応)
-kzdev_spi.c	…	Device Driver : SPI
-kzdsk.dpj	…	SDK Library 作成 VDSP++用プロジェクト
-kzdsp.h	…	Audio 用 Q26 演算ライブラリ
-kzstdio.c	…	stdio のラッピング関数
-kzstdio.h	…	stdio のラッピング関数
-kzprivate.h	…	SDK Private 関数定義
-kzversion.asm	…	SDK Version
-kzversion.h	…	SDK Version
-serial_io.c	…	Serial Input/Output 関数定義
-target_def.h	…	開発環境定義
-winmmc.c	…	Windows 用 MMC Driver
+lib	…	KOBANZAME SDK で使用する Library
-kzlib.c	…	SDK Library を使うためのコード
-makelib.dpg	…	Library を作るための VDSP++用プロジェクトグループ
+maketool	…	Make で使用する TOOL
+project	…	各 project の下のプロジェクト

+exsample	...	sample Program (VDSP++ & gcc)
+main	...	SDK Main Project
+jsp_sample1	···	JSP Sample1 が動作するプロジェクト
+sdkproject	···	SDK のメインプロジェクト
-config.bat	···	Sdkproject.cfg から kernel_cfg.h/kernel_id.h を作成するための MS-DOS 用バッチファイル。
-inclist.txt	···	Project の Include List。Geninc.exe を使用する
-kernel_cfg.c	···	Sdkproject.cfg から自動生成される。RTOS のコンフィグレーションファイル。
-l1only.ldf	···	リンカ指定コマンド。すべて L1 に配置される設定。
-main_tsk.c	···	メイン関数
-main_tsk.h	···	メイン関数用のインクルードファイル
-sdkproject.cfg	···	RTOS のコンフィギュレーションスクリプト
-sdkproject.dpj	···	VDSP++用のプロジェクト

3.3 SDK 共通 include file

すべての KOBANZAME SDK の機能は以下の include file をインクルードすること。

```
#include “KOBANZAME.h”
```

4 SDK API一覧

“KOBANZAME.h”をインクルードすることで使用可能になる。それぞれの詳しい仕様は後述の詳細仕様を参照すること。

4.1 Device Driver 関連 API

APIs	**概要**
KZSTATUS_t KzSwStart(KzSwCbk_t cbk);	SW Detect を開始する。Callback 関数に Sw の変化が通知される。
KZSTATUS_t KzSwStop(void);	SW Detect を停止する。
KZSTATUS_t KzLedOn(KzLED_t nLedNo);	LED を点灯させる
KZSTATUS_t KzLedOff(KzLED_t nLedNo);	LED を消灯させる
KZSTATUS_t KzLedBlink(KzLED_t nLedNo,int nOnTime, nOffTime);	LED を点滅させる。それぞれ On Time/Off Time を msec 単位で設定できる。
KZSTATUS_t KzMemsStart(KzMemsCbk_t cbk);	MEMS を開始する。Callback 関数に変化が通知される
KZSTATUS_t KzMemsStop(void);	MEMS を停止する
KZSTATUS_t KzAudioStart(KzAudioCbk cbk, long lSampleRate, int nBlocks, int nChannels);	Audio を開始する。Callback 関数で Audio の処理を行う。サンプルレートと、AudioBlock 単位の設定ができる。Channel 数の指定ができる(1～ 4)
KZSTATUS_t KzAudioStop(void);	Audio を停止する
KZSTATUS_t KzFilesystemStart(void);	Filesystem をスタートさせる。

4.2 Command Line 関連 API

コマンドラインは UNIX 等のように動的に実行プログラムを追加することはできない。すべてプログラム上で静的に追加しなくてはならない。

APIs	**概要**
KZSTATUS_t KzCmdlineStart(void);	Command Line を開始する。Command Line が終了するまで関数は戻らない。
KZSTATUS_t KzCmdlineExit(void);	Command Line を終了させる
KZSTATUS_t KzCmdlineAdd(const KzCmdResigt_t *exe);	Command を一つ登録する
KZSTATUS_t KzCmdlineAddMany(const KzCmdRegist_t *exes);	Command を複数登録する
void KzAddCmdFilesystem(void);	Filesystem 関連 Command を追加する
void KzAddCmdDeviceDriver(void);	Device Driver 関連 Command を追加する

4.3 stdio.h 準拠関数

“stdio.h”で定義されている一部の関数や定義が”KOBANZAME.h”をインクルードすることで、使用可能になる。実際にはリネームを行うことで実現している。標準入出力先は Serial になり、ファイルは fatfs 経由で SD-Card にアクセスされる。これらの詳細は他の書籍等を参照すること。

APIs	リネーム
FILE* Kz_fopen(const char *filename, const char *mode);	fopen
int Kz_fclose(FILE *fp);	fclose
int Kz_fseek(FILE *fp, long offset, int origin);	fseek
size_t Kz_fread(void *ptr, size_t size, size_t nmemb, FILE *stream):	fread
size_t Kz_fwrite(const void *ptr, size_t size, size_t nmemb,FILE *stream);	fwrite
char* Kz_fgets(char *s, int n, FILE *fp);	fgets
int Kz_fputs(const char *s, FILE *fp);	fputs
int Kz_fgetc(FILE *fp);	fgetc
int Kz_fputc(int c, FILE *fp);	fputc
int Kz_fscanf(FILE *fp,const char *format,...);	fscanf
int Kz_fprintf(FILE *fp,const char *format,...);	fprintf
int Kz_printf(const char *format,...);	printf
int Kz_scanf(const char *format,...);	scanf
char* Kz_gets(char *s);	gets
int Kz_puts(const char *s);	puts
int Kz_vprintf(const char *format, va_list arg);	vprintf
int Kz_vfprintf(FILE *fp,const char *format, va_list arg);	vfprintf
int Kz_getchar(void);	getchar
int Kz_putchar(int c);	putchar
int Kz_getc(FILE *fp);	getc
int Kz_putc(int c, FILE *fp);	putc

4.4 Audio DSP 演算ライブラリ

Q26 Audio DSP 演算ライブラリ

APIs	概要
int KaDspAdd(int a, int b)	a + b
int KzDspSub(int a, int b)	a - b
int KzDspMpy(int a, int b)	a * b
int KzDspMac(int *a, int *b, int n)	sum(a * b)
int KzDspSat(int a)	Saturation(a) limit [-1,1]
int KzDspAbs(int a)	Absolute(a)
int KzDspShl(int a, int s)	a << s … Shift left
int KzDspShr(int a, int s)	a >> s … Shift right
int KzDspNeg(int a)	-a
float KzDspTof(int a)	Q26 to float
int KzDspToi(float a)	float to Q26

4.5 バージョン関連 API

APIs	概要
int KzGetVersionMajor(void);	SDK の Major Version を取得
int KzGetVersionMinor(void);	SDK の Minor Version を取得
int KzGetBuildIdx(void);	SDK の Build Index Number を取得
const char *KzGetBuildDate(void);	SDK の Build Date をストリングで取得
const char *KzGetCopyright(void);	SDK のコピーライトをストリングで取得

5 SDK API 詳細

“KOBANZAME.h”をインクルードすることで使用可能になる API 詳細を以下に示す。

5.1 Device Driver 関連

5.1.1 KzSwStart

【概要】

SW Detect を開始する。Callback 関数に Sw の変化が通知される。

【書式】

KZSTATUS_t KzSwStart(KzSwCbk_t cbk);

【引数】

KzSwCbk_t cbk … Switch の状態が変化したときにドライバーから呼ばれるコールバック関数。

KzSwCbk_t は以下に定義されている
typedef void (*KzSwCbk_t)(KZSW_t nSw, BOOL bOn);
nSw は Switch No で eKZSW0 or eKZSW1 が入る。 bOn は TRUE なら押された状態、FALSE なら離された状態に変化した。

【戻り値】

KZ_OK … 成功
KZ_ERR … 失敗 （すでに switch を開始している）

【詳細】

Switch を開始する。引数のコールバック関数がドライバーより変化があったときだけ呼ばれる。Switch の状態にかかわらず開始時には必ず Swich の数だけコールバック関数が呼ばれる。

【サンプルコード】

```
static void sw_callback( KZSW_t nSw, BOOL bOn )
{
	if( nSw == eKZSW0 ) {
		if( bOn ) {
		/* Switch 0 が押されたときの処理 */
		} else {
		/* Switch0 が離されたの処理 */
		}
	} else if( nSw == eKZSW1 ) {
		if( bOn ) {
		/* Switch 1 が押されたときの処理 */
		} else {
		/* Switch1 が離されたの処理 */
		}
	}
}

void main(void)
{
	/* Switch Detect Start */
	KzStartSw( sw_callback ):

	/* 何らかの処理 */

	/* Switch Detect Stop */
	KzStopSw();
}
```

5.1.2 KzStopSw

【概要】

SW Detect を停止する。

【書式】

KZSTATUS_t KzStopSw(void);

【引数】

なし

【戻り値】

KZ_OK … 成功
KZ_ERR … 失敗 （すでに switch が停止している）

【詳細】

Switch を停止する。コールバック関数が呼ばれなくなる。

【サンプルコード】

KzStartSw を参照

5.1.3 KzLedOn

【概要】

LED を点灯させる

【書式】

KZSTATUS_t KzLedOn(KzLED_t nLedNo);

【引数】

KzLED_t nLedNo … LED ナンバー。eKZLED0 or eKZLED1。

【戻り値】

KZ_OK … 成功
KZ_ERR … 失敗 （nLedNo の値が範囲を超えている）

【詳細】

LED を点灯させる。

【サンプルコード】

```
void main(void)
{
	/* LED0 を点灯 */
	KzLedOn( eKZLED0 );

	/* LED1 を点灯 */
	KzLedOn( eKZLED1 );
}
```

5.1.4 KzLedOff

【概要】

LED を消灯させる

【書式】

KZSTATUS_t KzLedOff(KzLED_t nLedNo);

【引数】

KzLED_t nLedNo … LED ナンバー。eKZLED0 or eKZLED1。

【戻り値】

KZ_OK … 成功
KZ_ERR … 失敗 （nLedNo の値が範囲を超えている）

【詳細】

LED を消灯させる。

【サンプルコード】

```
void main(void)
{
	/* LED0 を消灯 */
	KzLedOff( eKZLED0 );

	/* LED1 を消灯 */
	KzLedOff( eKZLED1 );
}
```

5.1.5 KzLedBlink

【概要】

LED を点滅させる。それぞれ On Time/Off Time を msec 単位で設定できる。

【書式】

KZSTATUS_t KzLedBlink(KzLED_t nLedNo, int nOnTime, int nOffTime);

【引数】

KzLED_t nLedNo … LED ナンバー。eKZLED0 or eKZLED1。
int nOnTime … ブリンクの点灯時間。単位は ms。
int nOffTime … ブリンクの消灯時間。単位は ms。

【戻り値】

KZ_OK … 成功
KZ_ERR … 失敗 （nLedNo の値が範囲を超えている）

【詳細】

LED をブリンクさせる。点灯時間・消灯時間を設定できる

【サンプルコード】

```
void main(void)
{
        /* LED0 を  点灯時間 1(s) 消灯時間 0.5(s) でブリンク */
        KzLedBlink( eKZLED0, 1000, 500 );
}
```

5.1.6 KzMemsStart

【概要】

MEMS を開始する。Callback 関数に変化が通知される

【書式】

KZSTATUS_t KzMemsStart(KzMemsCbk_t cbk);

【引数】

KzMemsCbk_t cbk … MEMS の値を取得したときに呼ばれるコールバック関数。

KzMemsCbk_t は以下の定義
typedef void (*KzMemsCbk_t)(int x, int y, int z);

【戻り値】

KZ_OK … 成功
KZ_ERR … 失敗 （MEMS がすでにスタート状態）

【詳細】

MEMS (Freescale MMA7455L)をスタートさせる。MEMS は加速度センサーで、基板の傾きによって x,y,z 方向の値を得ることができる。MEMS から値を取得された後に、コールバック関数がドライバーよりコールされ、ユーザーに通知される。

【サンプルコード】

```
static void mems_callback( int x, int y, int z )
{
	/* MEMS 値を取得したときのコード */
}

void main(void)
{
	KZSTATUS_t stat;

	/* MEMS のスタート */
	stat = KzMemsStart( mems_callback );

	if( stat != KZ_OK ) {
		/* ERROR 時の処理*/
	}
}
```

5.1.7 KzMemsStop

【概要】

MEMS を停止する。

【書式】

KZSTATUS_t KzMemsStop(void);

【引数】

なし

【戻り値】

KZ_OK … 成功
KZ_ERR … 失敗 （MEMS がすでに停止状態）

【詳細】

MEMS (Freescale MMA7455L)を停止する。

【サンプルコード】

```
static void mems_callback( int x, int y, int z )
{
	/* MEMS 値を取得したときのコード */
}

void main(void)
{
	KZSTATUS_t stat;

	/* MEMS のスタート */
	stat = KzMemsStart( mems_callback );

	/* 何らかの処理 */

	/* MEMS の停止 */
	stat = KzMemsStop();

}
```

5.1.8 KzAudioStart

【概要】

Audio をスタートする。サンプリングレートとブロック数の指定ができる。ただ現在はサンプルレートは 48kHz, ブロック数は8の倍数で 8～32 までの数、チャンネル数は 1～4 までとなっている。

【書式】

KZSTATUS_t KzAudioStart(KzAudioCbk cbk, long lSampleRate, int nBlocks, int nChannels);

【引数】

KzAudioCbk_t cbk … オーディオ処理をするコールバック関数

KzAudioCbk_t は以下の定義
typedef void (*KzAudioCbk_t)(int *in, int *out, int nBlocks , nChannels);
in には Q6.26 フォーマット で L0/R0/L1/R1 の順に nChannels の数だけクラスタリングされ nBlocks 分のサンプルデータが格納される。 out に nBlocks 分のサンプルデータを in と同じフォーマットと並びで格納する必要がある。

long lSampleRate … オーディオサンプルレート。現在は 48000 固定

int nBlocks … 一回で処理をするオーディオブロック。 1 ~ 144 まで。 この値を大きくすればするほど、大きなブロック処理が可能で、RTOS のオーバーヘッドなどを小さくすることができる。ただし、オーディオ入力が出力まで到達する時間（レイテンシーと呼ばれる）は大きくなる。現在は 8～32 の間の数でかつ8の倍数でなくてはならない。

int nChannels … Block の中に入っている Audio Channel の数。1～4まで。ステレオ処理でよいなら2を指定する。

【戻り値】

KZ_OK … 成功
KZ_ERR … 失敗 （Audio がすでにスタート状態もしくは値が範囲を超えている）

【詳細】

Audio をスタートさせる。Blackfin の SPORT と Audio Codec(AnalogDevicees AD1836A) の設定を lSampleRate の周波数で行い、受信データが用意されたらコールバック関数で

処理をする。自動的にダブルバッファーになる。そのためコールバック関数の時間的制約は次の受信データが用意できるまでに関数を終了させることが必要。

【サンプルコード】

```
static void audio_callback( int *in, int *out, int nBlock, int nChannels )
{
	/*  Echo Back */
	memcpy( out, in, nBlocks * sizeof(int) * nChannels );
}

void main(void)
{
	KZSTATUS_t stat;

	/* 48kHz/32block/4channel でオーティオのスタート */
	stat = KzAudioStart( audio_callback, 48000, 32 , 4);

	if( stat != KZ_OK ) {
		/* ERROR 時の処理*/
	}
}
```

5.1.9 kzAudioStop

【概要】

Audio 停止する。

【書式】

KZSTATUS_t KzAudioStop(void);

【引数】

なし

【戻り値】

KZ_OK … 成功
KZ_ERR … 失敗 （Audio がすでに停止状態）

【詳細】

Audio を停止する。

【サンプルコード】

```
void main(void)
{
	KZSTATUS_t stat;


	/* 何らかの処理 */

	/* Audio の停止 */
	stat = KzAudioStop();

}
```

5.1.10 KzFilesystemStart

【概要】

ファイルシステムをスタートする

【書式】

KZSTATUS_t KzFilesystemStart(void);

【引数】

なし

【戻り値】

KZ_OK … 成功
KZ_ERR … 失敗 （SD-Card Error)

【詳細】

ファイルシステムをスタートする。この関数を呼ぶとSD-Cardが初期化されマウントされる。
成功したときはKZ_OKを返す。SD-Cardは動的な抜き差しに対応していない。

【サンプルコード】

```
void main(void)
{
	KZSTATUS_t stat;

	/* Filesystem のスタート */
	stat = KzFilesystemStart();

	if( stat != KZ_OK ) {
		/* ERROR 時の処理*/
	}
}
```

5.2 Command Line 関連

5.2.1 KzCmdlineStart

【概要】

コマンドラインをスタートさせる

【書式】

KZSTATUS_t KzCmdlineStart(void);

【引数】

なし

【戻り値】

KZ_OK … 成功
KZ_ERR … 失敗

【詳細】

コマンドラインをスタートさせる。この関数はコマンドラインが終了するまで戻らない。

【サンプルコード】

```
void main(void)
{
        KZSTATUS_t stat;

        /* コマンドラインのスタート */
        stat = KzCmdlineStart();

        /* ここにはコマンドライン終了まで実行されない */

}
```

5.2.2 KzCmdlineExit

【概要】

コマンドラインを停止する

【書式】

KZSTATUS_t KzCmdlineExit(void);

【引数】

なし

【戻り値】

KZ_OK … 成功
KZ_ERR … 失敗

【詳細】

コマンドラインを停止する。コマンドライン実行関数で呼ぶのが望ましい。他のタスクからも呼ぶことができるが、実際に停止するのはキー判定の時になるため、すぐにはコマンドは終了しない。exit コマンドはこの関数を呼ぶのみの処理になっている。

【サンプルコード】

```
/*  コマンド実行関数 */
static void print_and_exit( int argc, char* argv[] )
{
        printf("Command Line Exit\n");

        /*  コマンドラインを終了 */
        KzCmdlineExit();

        return 0;
}
```

5.2.3 KzCmdlineAdd

【概要】

Command Line に実行関数を1つ登録する。

【書式】

KZSTATUS_t KzCmdlineAdd(const KzCmdResigt_t *exe);

【引数】

登録する関数と実行コマンド、ヘルプが記述されている構造体を引数にもつ。
KzCmdRegist_t は以下のように宣言されている。

```
typedef struct {
    KzFnCmd_t       mfnExe;     /* 登録する関数 */
    const char*     msCmd;      /* 実行コマンドの文字列*/
    const char*     msHelp;     /* ヘルプの文字列*/
} KzCmdRegist_t;
```

KzFnCmd_t は以下のように宣言されている

```
typedef int (*KzFnCmd_t)(int argc, char *argv[] );
```

登録できるコマンドの最大数はデフォルトで以下に設定されている。この数値はユーザーが自由に設定可能である。この数を超えたコマンド数を登録しようとするとエラーが起こる。

```
kzconfig.h
#define KZCMDLINE_MAX_COMMANDS (32)
```

【戻り値】

KZ_OK … 成功
KZ_ERR … 失敗 （Command 数が最大数を越した）

【詳細】

Command Line に関数を一つ登録する。引数にもつ構造体が Command Line に登録され、以降コマンドとして実行可能になる。help コマンドで表示される文字列を msHelp に入れる。通常は KzCmdlineStart() を呼ぶ前にこの関数を呼び登録するが、後に呼んでも構わない。

【サンプルコード】

```
/*
 * Hello World プロジェクト
 *       “hello”コマンドで実行される
 */

int hello_exe(int argv, char *argv[] )
{
        printf(“Hello World\n”);
        return 0;
}


void main(void)
{
        KzCmdRegist regist;
        regist.mfnExe = hello_exe;          /* 登録する関数 */
        regist.msCmd = “hello”;             /* 実行コマンドの文字列*/
        regist.msHelp = “Show Hello”;       /* ヘルプの文字列 */

        /* 上記のコマンドを登録する */
        KzCmdlineAdd( &regist );

        /* コマンドラインスタート */
        KzCmdlineStart();
}
```

以下　コマンドライン

```
> help
      :
    hello  … Show Hello
      :

> hello
Hello World
>
```

5.2.4 KzCmdlineAddMany

【概要】

Command Line に実行関数を複数登録する。

【書式】

KZSTATUS_t KzCmdlineAddMany(const KzCmdRegist_t *exes);

【引数】

exes ZERO ターミナルされた、KzCmdRegist_t の配列ポインタ。

【戻り値】

KZ_OK … 成功
KZ_ERR … 失敗 （Command 数が最大数を越した）

【詳細】

Command Line に実行関数を複数登録する。KzCmdlineAdd の複数版である。

【サンプルコード】

```
/*
 *  hello1 と hello2 を登録する
*/

/*  hello1 の実行関数 */
int hello_exe_1(int argc, char *argv[] )
{
        printf(“Hello World[%d]\n”, 1 );
        retrun 0;
}

/* hello2 の実行関数 */
int hello_exe_2(int argc, char *argv[] )
{
        printf(“Hello World[%d]\n”, 2 );
        retrun 0;
}

const KzCmdRegist_t cmds[] = {
        {       hello_exe_1,   “hello1”,       “Show Hello1”       },
        {       hello_exe_2,   ”hello2”,       ”Show Hello2”       },
        {       0,             0,              0                   } /* Zero Terminater*/
};

void main(void)
{
        /* 複数のコマンドを一気に登録する */
        KzCmdLineAddMany( cmds );

        /* コマンドラインスタート */
        KzCmdlineStart();
}
```

以下　コマンドライン

```
> help
      :
   hello1  … Show Hello1
   hello2  … Show Hello2
      :

> hello1
Hello World[1]
>
```

5.2.5 KzAddCmdFilesystem

【概要】

コマンドラインに Filesystem 関連コマンドを追加する

【書式】

void KzAddCmdFilesystem(void);

【引数】

なし

【戻り値】

KZ_OK … 成功
KZ_ERR … 失敗 （Audio がすでに停止状態）

【詳細】

コマンドラインに Filesystem 関連コマンドを追加する。KzFilesystemStart を呼び成功を確認した後にこの関数を呼ぶことが望ましい。また KzCmdlineStart() でコマンドラインをスタートさせる前に呼ぶことも望ましい。

この関数を呼ぶと、コマンドに以下の10個のコマンドが登録される。

ls / pwd / cd / mkdir / rmdir / cp / rm / mv / dump / more

【サンプルコード】

```
void main(void)
{
	/* Filesystem スタート */
	if( KzFilesystemStart() == KZ_OK ) {
		/* Filesystem が成功したのでコマンドを追加する */
		 KzAddCmdFilesystem();
	}

	/* コマンドラインスタート */
	KzCmdlineStart();

}
```

5.2.6 KzAddCmdDeviceDriver

【概要】

コマンドラインに Device Driver 関連コマンドを追加する。

【書式】

void KzAddCmdDeviceDriver(void);

【引数】

なし

【戻り値】

KZ_OK … 成功
KZ_ERR … 失敗 （Audio がすでに停止状態）

【詳細】

コマンドラインに Device Driver 関連コマンドを追加する。KzCmdlineStart() でコマンドラインをスタートさせる前に呼ぶことが望ましい。

この関数を呼ぶと、コマンドに以下の4個のコマンドが登録される.。

swled / wink / mems / echo / sdcard

【サンプルコード】

```
void main(void)
{
	/* DeiveDriver コマンド登録 */
	KzAddCmdDeviceDriver();

	/* コマンドラインスタート */
	KzCmdlineStart();

}
```

5.3　　バージョン関連 API

5.3.1 KzGetVersionMajor

【概要】

SDK の Major Version を取得する

【書式】

int　　KzGetVersionMajor(void);

【引数】

なし

【戻り値】

SDK の Major Version

【サンプルコード】

```
void main(void)
{

        printf("KOBANZAME SDK Version   : %d.%02d(%04d)\n",KzGetVersionMejor(),
                KzGetVersionMinor(), KzGetBuildIdx() );
        printf("           Build Date: %s\n", KzGetBuildDate() );
        printf("           Copyright : %s\n", KzGetCopyright() );
}
```

実行例。コマンドライン

```
KOBANZAME SDK Version : 1.00(004)
                    Build Date : Jan  3 2010/18:33:19
                     Copyright : KOBANZAME SDK Project
```

5.3.2 KzGetVersionMinor

【概要】

SDK の Minor Version を取得する

【書式】

int KzGetVersionMinor(void);

【引数】

なし

【戻り値】

SDK の Minor Version

【サンプルコード】

KzGetVersionMajor を参照すること

5.3.3 KzGetBuildIdx

【概要】

SDK の Build Index を取得する

【書式】

int KzGetBuildIdx(void);

【引数】

なし

【戻り値】

SDK の Build Index

【サンプルコード】

KzGetVersionMajor を参照すること

5.3.4 KzGetBuildDate

【概要】

SDK の Build Date を取得する

【書式】

const char *KzGetBuildDate(void);

【引数】

なし

【戻り値】

SDK の Build Date

【サンプルコード】

KzGetVersionMajor を参照すること

5.3.5 KzGetCopyright

【概要】

SDK の Copyright の文字列を取得する

【書式】

const char *KzGetCopyright(void);

【引数】

なし

【戻り値】

SDK の Copyright の文字列

【サンプルコード】

KzGetVersionMajor を参照すること

6 コマンドライン

KzCmdlineStart()が呼ばれるプログラムを起動し、KOBANZAME のシリアルを PC 等に接続をすると、コマンドラインを使用することができる。

6.1 シリアルの設定

- ボーレート 57600 bps
- データ 8bit
- パリティ none
- ストップ 1bit
- フロー制御 none

6.2 サンプルコードのコマンドライン一覧

サンプルコードを実行しすると、デフォルトで以下のコマンドを使用することができる。これらのコマンドはユーザーが独自に追加することも可能である。ただし、動的な追加には対応していない。プログラムのコンパイル時にソースコードにコマンドを追加する必要がある。

6.2.1 ファイルシステム操作関連

KzAddCmdFilesystem() を呼ぶことで追加ができる。
UNIX のコマンドと似ているが、オプション等すべての機能に対応しているわけではない。

コマンド	備考
cd	カレントディレクトリの変更
ls	カレントディレクトリのリスト表示
pwd	カレントディレクトリの場所を出力
cp	ファイルのコピー
mkdir	ディレクトリの作成
mv	ファイルの移動
rm	ファイルの削除
rmdir	ディレクトリの削除
dump	ファイルのダンプ表示
more	ファイルのテキスト表示

6.2.2 ドライバー操作関連

KzAddCmdDeviceDriver() を呼ぶことで追加ができる。

コマンド	備考
swled	Switch /LED のサンプルプログラム。押された場所の LED が点灯する。key 入力で終了する。
wink	LED0 と LED1 が交互に点滅する
mems	MEMS のサンプルプログラム。XZY の値がされ続ける。key 入力で終了する。
echo	Audio のサンプルプログラム。A/D に入力されたデータがそのまま D/A に出力される。Key 入力で終了する。
sdcard	SD-Card のテスト。10MByte の Write/Read テストを行う。Card によっては時間がかかる場合があるので注意すること。

6.3 リダイレクト機能

本コマンドラインプログラムはリダイレクト機能に対応している。これは stdout に表示されるものを任意のファイルに書き出すことができる。[command] > [filename] という書式になる。

例:
mems の出力を mems.txt に書く

```
mems > mems.txt
```

7 Audio DSP 演算ライブラリ

KOBANZAME SDK には Audio 用信号処理に使用できる演算ライブラリーが実装されている。32bit 固定小数で処理される。ここではその詳細を説明する。

7.1 Q26 Format

32bit 固定小数の Audio DSP 演算ライブラリの基本フォーマットである。int 型を使用する。一般的には Q6.26 と呼とも呼ばれるもので、32bit のうち小数部が 26bit、整数が 5bit、符号が 1bit のフォーマットである。int 型ではあるが、仮想的に小数点が 26bit の場所にある。

32bit int 型:	o	ooooo	oooooooooo-oooooooooo-oooooo
	符号	整数(5bit)	小数(26bit)

整数が 5bit あるので 表現できる値の範囲は -32 ～ +32 となる。

例

1.0 を Q26 で表現すると 0x0400-0000 となる
同様に
-1.0 は 0xFC00-0000
0.23 は 0x0400-0000 x 0.23 = 0xEB851E

である

7.2 サポートされる演算

加算

```
int KzDspAdd(int a,int b)
ans = a + b
```

減算

```
int KzDspSub(int a,int b)
ans = a – b
```

乗算

```
int KzDspMpy(int a,int b)
ans = a * b
```

積和

```
int KzDspMac(int *a,int *b, int n)
ans += a * b
```

飽和

int KzDspSat(int a)

最大 1.0 、最小 -1.0 に丸めこむ

ans = a >1.0? 1.0 : a<-1.0? -1.0 : a;

絶対値

int KzDspAbs(int a)

ans = | a |

左シフト

int KzDspShl(int a)

ass = a << s (マイナスの値は右シフトになる)

右シフト

int KzDspShr(int a)

ans = a >> s (マイナスの値は左シフトになる)

符号反転

int KzDspNeg(int a)

ans = -a

Q26 から float に型変化

float KzDspTof(int a)

float から Q26 に型変化

int KzDspToi(float a)

7.3 C 言語の演算との相違点

C 言語でも加算や減算などは容易にできる。C 言語での関数を使うわかりに、Audio DSP 演算ライブラリを使うことで以下のことが自動で行われる。

- 飽和演算
- 小数点の考慮

7.3.1 飽和演算

例えば以下の C 言語での加算を考えてみる。

int ans = a + b;

いまここで a と b にそれぞれ 0x4000-0000 が代入されているとする。この処理が実行された後、ans には 0x8000-0000 が代入される。 ここで 0x4000-0000 は Q26 フォーマットの

16.0 のことで、0x8000-0000 は -32.0 のことである。C 言語では 16.0 + 16.0 = -32.0 という結果になってしまう。これは C 言語の加算の値は循環（ラップアラウンド）するのでこのような結果になってしまう。

Audio の処理においてほとんどの場合は循環させるより飽和（クリップ）を使用する。Audio DSP 演算ライブラリーを使えば最大値をオーバーした場合は最大値に値が飽和する。上記の C 言語の例では

```
int ans = KzDspAdd( a, b );
```

とし、上記同様 a 及び b に 0x4000-0000 が代入されているとすると、ans は飽和処理が行われ正の最大数 0x7FFF-FFFF(約+32.0) になる。負の値も同様に 0x8000-0000(-32.0)に飽和処理が行われる。

通常の CPU ではこれらの処理は比較演算を使用し処理を行うことになり CPU パワーの消費が大きいが、飽和演算をハードウェアでサポートしている DSP の場合は少ないパワーで飽和処理を行うことができる。ただし、一般的な DSP はこれらの機能を使うにはプログラマがアセンブラなどで特殊な操作をしないといけない。このため KOBANZAME SDK ではこれら特殊処理を関数に隠蔽してユーザーに提供している。すべての処理をインラインで記述しており、非常に高速に飽和演算の処理ができる。

7.3.2 小数点の考慮

特に乗算処理でのことだが、Q26 では小数点が違うため乗算演算後にビットシフト等で桁あわせを行わないといけない。また C 言語の整数乗算では 32bit 同士は結果が 64bit 長になり一般的には 64bit 型(long long 型)を使用することになる。

これらにより一般的な CPU で Q26 の演算を行おうとすると以下の演算となる。また加えて飽和処理も実際には加わる。

```
long long tmp = a * b;   /* tmp は Q10.54 format */
ans = (int)( tmp >> (54-27) );   /* ただし飽和演算を省略している */
```

これに飽和演算処理も加わり一般的な CPU では非常に時間がかかる処理になってしまう。Blackifn DSP ではこれらの処理が高速にできる乗算器がハードウェアで実装されている。ただしアセンブラで記述しないと使用ができない。そこで KOBANZAME SDK ではユーザーがすぐに使用できるよう、インラインアセンブラを用い乗算を実装している。

この乗算は数サイクルのうちに実行ができるが、演算精度が 20bit になる。これは演算精度を落とす代わりに処理を軽くすることを目指しているからである。Audio 処理では 20bit の演算精度があれば十分な場合がほとんどである。

8 改訂履歴

ver1.00 … 2010/01/05 新規作成
ver1.10 … 2010/01/11 SDHC 対応 / kzprivate.h 追加 / sdcard コマンド追加
ver1.20 … 2010/01/16 DSP Audio 演算ライブラリ追加
ver1.30 … 2010/01/23 Audio 用 API 変更。チャンネル数の指定が可能
ver1.40 … 2010/02/27 gcc 対応