UTDS ドライバ version 1.17 取扱説明書

株式会社 創夢
第二開発部

平成 21 年 5 月 15 日

目 次

1 はじめに

本文書は、日本通信機株式会社製の USB デバイス「時刻同期式データサンプラーカード」を、**2.6 系の Linux Kernel** にて動作させるために作成したドライバソフトウェアについて記述したものです。以下、時刻同期式データサンプラーカードドライバソフトウェアのことを単に **UTDS ドライバ**と呼びます。

現在の UTDS ドライバの最新バージョンは version 1.17 で、この版では、Linux kernel-2.6.26.1 に対応しています。本書では、カーネルバージョン表記上の煩雑さを避けるため、以下の記法を用います。適宜本文を読み替えてください。

表記	置換文字列
<KERNELVERSION>	2.6.26.1
<KERNELRELEASE>	2.6.26-1-686

2 動作確認環境

- **Debian GNU/Linux 5.0 (kernel 2.6.26-1-686) i386** プラットフォーム[1] [2]
- サンプリングデータをバイナリ形式にてローカルハードディスクに保存するテストにおいて、サンプリング設定 128Mbps まで動作を確認。

3 ディレクトリ/ファイル構成

tar+gzip によって作成されたアーカイブを解凍したディレクトリ/ファイルの構成は、以下のようになっています。

```
vlbi-usb-linux/
   |
   +Makefile
   +doc/
   |  |
   |  +Makefile 取扱説明書作成用の Makefile
   |  |
   |  +manual.pdf 取扱説明書 (PDF 形式)
   |  |
   |  +manual.ps 取扱説明書 (PostScript 形式)
   |  |
   |  +manual.tex 取扱説明書 (LaTeX ソース)
   |
   +driver/
   |  |
   |  +Makefile UTDS ドライバビルド用の Makefile
   |  |
   |  +utds.c UTDS ドライバソースファイル
```

[1] 日本通信機様から後指示の頂いた開発環境である Debian GNU/ Linux 5.0 (kernel 2.6.26-1-686) 以外の環境での動作は未確認です。

[2] マルチプロセッサシステムにおける動作、および、ターゲットサンプラーを同一 PC に複数装着した場合の動作には対応していません。

```
|
+examples/
|   |
|   +Makefile サンプルプログラムビルド用 Makefile
|   |
|   +*.c サンプルプログラム (C ソース)
|
+include/
|   |
|   +tdsio.h UTDS ドライバ I/O コントロールコマンドファイル
|   |
|   +tdssdh.h 時刻同期式データサンプラーのサンプリングデータの
| ヘッダフォーマット定義ファイル
|
+libtds/
    |
    +Makefile ライブラリビルド用 Makefile
    |
    +libtds.c ライブラリソースファイル
```

4 ローダブルモジュール

4.1 Linux カーネルソースツリーの準備

1. Linux カーネルソースツリーの展開 UTDS ドライバのカーネルローダブルモジュールのコンパイルには、コンパイル済みの Linux カーネルソースツリーが必要です。システムで稼働中の Linux カーネルソースツリーがデフォルトのソースパスに存在している場合は、カーネルソースツリーを改めて準備する必要はありませんので、この手順はスキップしてください。

   デフォルトのカーネルソースパス: /lib/modules/<KERNELRELEASE>/build/

   デフォルトのカーネルソースパスに Linux カーネルツリーが存在しない場合は、Linux kernel-<KERNELVERSION> のオリジナルソースを入手し、作業用のディレクトリに展開してください。展開場所は上記のデフォルトのカーネルソースパス以外の任意の場所でも構いません。

   ```
   # cd (some-work-directory)
   # bzip2 -dc linux-<KERNELVERSION>.tar.bz2  | tar xvf -
   ```

2. Lunix カーネルのコンパイルデフォルトのカーネルソースパス以外の場所に Linux

カーネルソースを展開した場合は、以下の手順にしたがって、カーネルのコンフィグレーションとコンパイル行なってください。

```
# cd (some-work-directory)/linux-<KERNELVERSION>
# make oldconfig
# make all
```

4.2 ローダブルモジュールの作成

1. vlbi-usb-linux.tar.gz を展開して vlbi-usb-linux に移動する。

```
# tar xvfz vlbi-usb-linux.tar.gz
# cd vlbi-usb-linux/
```

2. モジュールを構築、インストールする。この時ライブラリも同時にビルド、インストールされます。コンパイル済のカーネルがデフォルトのカーネルソースパス以外にある場合は、環境変数 KERNELDIR にてパス名を指定してください。

```
# export KERNELDIR=(some-work-directory)/linux-<KERNELVERSION>
# make
# make install
```

4.3 ローダブルモジュールのロード

1. デバイスノードを作成する。デバイスファイルは、/dev/utds0 となります。もし Linux 上で udevd が動作している場合はデバイスノードを作成する必要はありません。

```
# cd /dev
# mknod -m 666 utds0 c 180 222
```

2. モジュールをロードする。

```
# insmod /lib/modules/<KERNELRELEASE>/kernel/drivers/usb/misc/utds.ko
```

5 インストールするファイル

make install を実行する事によって次のファイルをそれぞれの場所にインストールします。

```
tds.ko /lib/modules/<KERNELRELEASE>/kernel/drivers/usb/misc/utds.ko
tdsio.h /usr/include/sys/tdsio.h
tdssdh.h /usr/include/sys/tdssdh.h
libtds.so /usr/local/lib/libtds.so
```

6 システムコールインタフェース

6.1 サポートしているシステムコール

1. open()
2. close()
3. read()
4. ioctl()

6.2 システムコール使用例

以下に、各システムコールの使用例を示す。特に記述していない点に関しては、**Linux Kernel 2.6 系**標準のシステムコールの仕様に準じている。

6.2.1 open()

デバイスをオープンし、ファイルディスクリプタを返す。引数にはデバイス名、オープンモードを指定する。当デバイスは読み出し専用であるので、オープンモードは**O_RDONLY**を指定する。

(使用例)

```
int fd;


fd = open("/dev/utds0", O_RDONLY);
```

6.2.2 close()

デバイスをクローズする。引数には 6.2.1 の open() で返されたファイルディスクリプタを指定する。

(使用例)

```
close(fd);
```

6.2.3 read()

オープンしたデバイスから、データを読み込む。引数には 6.2.1 の open() で返されたファイルディスクリプタ、データ読み込み用バッファのアドレス、そのバッファサイズ (バイト数) を指定する。実際に読み込まれたデータサイズ (バイト数) が返される。

(使用例)

```
int nread;
unsigned int buffer[1024];
int nbyte = sizoef(buffer);


nread = read(fd, buffer, nbyte);
```

6.2.4 ioctl()

デバイスの各種操作を行なう。第 1 引数には 6.2.1 の open() で返されたファイルディスクリプタ、第 2 引数には行なうべき操作、必要に応じて第 3 引数以降にその操作ごとに必要なパラメータを指定する。

(使用例)

```
int status, parameter;
```

```
status = ioctl(fd, TDSIO_XXX, &parameter);
```

6.3 ioctl 使用例

6.3.1 初期化 — TDSIO_INIT

デバイスドライバ内パラメータを初期化する。第 2 引数には **TDSIO_INIT** を指定する。この操作を実行すると、以下の状態になる。

- チャンネル数設定がなされていない状態 (6.3.8 参照)
- ビット数設定がなされていない状態 (6.3.9 参照)
- フィルタ周波数設定がなされていない状態 (6.3.10 参照)
- サンプリング周波数設定がなされていない状態 (6.3.11 参照)
- 1PPS 同期がなされていない状態 (6.3.3 参照)
- 時刻設定がなされていない状態 (6.3.4 参照)
- エラー情報をリセットした状態 (6.3.27 参照)

(使用例)

```
ioctl(fd, TDSIO_INIT);
```

6.3.2 サンプラー FIFO クリア — TDSIO_BUFFER_CLEAR

サンプラー上の FIFO をクリアする。第 2 引数には **TDSIO_BUFFER_CLEAR** を指定する。

(使用例)

```
ioctl(fd, TDSIO_BUFFER_CLEAR);
```

6.3.3 1PPS 同期 — TDSIO_SYNC_1PPS

外部 1PPS 入力信号への同期を行なう。第 2 引数には **TDSIO_SYNC_1PPS** を指定する。外部 1PPS 入力信号が検出できない場合には、エラーが発生する。

(使用例)

```
ioctl(fd, TDSIO_SYNC_1PPS);
```

6.3.4 時刻設定 — TDSIO_SET_TIME

サンプラーに時刻を設定する。第 2 引数には **TDSIO_SET_TIME** を、第 3 引数には設定すべき時刻データを保持する変数 (32bit) のアドレスを指定する。時刻の形式は、下位 17 ビット (0bit-16bit) に 00:00:00 からの通算秒で、18bit-27bit(9bit 幅) に 1 月 1 日からの通算日数で指定する。[3]

簡易マクロとして、以下を用意している。

マクロ	機能
TDS_MAKE_TIME(day, sec)	day(1 月 1 日からの通算日数) と sec(00:00:00 からの通算秒) から、適切な値を生成する。

(使用例)

```
/* 2000/09/22 14:39:28 を指定する場合 */
unsigned int time;
unsigned int day;
unsigned int sec;

day = 31 + 29 + 31 + 30 + 31 + 30 + 31 + 31 + 22;
sec = ((14 * 60) + 39) * 60 + 28;
time = TDS_MAKE_TIME(day, sec);
ioctl(fd, TDSIO_SET_TIME, &time);
```

[3] デバイスレジスタへの書き込み時に指定する時刻と同じ形式

6.3.5 時刻取得 — TDSIO_GET_TIME

サンプラーが保持する時刻を取得する。第2引数には **TDSIO_GET_TIME** を、第3引数には取得した時刻データを格納する変数 (32bit) のアドレスを指定する。取得される時刻データの形式は、6.3.4 時刻設定における形式と同様。

簡易マクロとして、以下を用意している。

マクロ	機能
TDS_GET_SEC(arg)	引数 arg から 00:00:00 からの通算秒数部分を取り出す。
TDS_GET_SEC_CARRY(arg)	引数 arg から通算秒数のキャリービットを取り出す。結果は 0 または 1 になる。
TDS_GET_DAYS(arg)	引数 arg から 1 月 1 日からの通算日数部分を取り出す。
TDS_GET_DAYS_CARRY(arg)	引数 arg から通算日数のキャリービットを取り出す。結果は 0 または 1 になる。

(使用例)

```
unsigned int time;
unsigned int day;
unsigned int sec;

ioctl(fd, TDSIO_GET_TIME, &time);
day = TDS_GET_DAYS(time);
sec = TDS_GET_SEC(time);
```

6.3.6 年情報設定 — TDSIO_GET_YEAR

サンプラーに年情報を設定する。第2引数には **TDSIO_GET_YEAR** を、第3引数には、設定する年データを格納する変数 (32bit) のアドレスを指定する。

(使用例)

```
unsigned int year;

ioctl(fd, TDSIO_SET_YEAR, &year);
```

6.3.7 年情報取得 — TDSIO_GET_YEAR

サンプラーが保持している年情報を取得する。第2引数には**TDSIO_GET_YEAR**を、第3引数には、取得した年データを格納する変数(32bit)のアドレスを指定する。

(使用例)

```
unsigned int year;


ioctl(fd, TDSIO_GET_YEAR, &year);
```

6.3.8 チャンネル数設定 — TDSIO_SET_CHMODE

サンプリングを行なうチャネル数を設定する。第2引数には**TDSIO_SET_CHMODE**を、第3引数にはチャンネル数設定情報を保持する変数(32bit)のアドレスを指定する。

- チャネル数指定

チャネル数	指定マクロ
1ch	TDSIO_SAMPLING_1CH
4ch	TDSIO_SAMPLING_4CH

(使用例)

```
/* 4ch を設定する場合 */
unsigned int chmode;


chmode = TDSIO_SAMPLING_4CH;
ioctl(fd, TDSIO_SET_CHMODE, &chmode);
```

6.3.9 ビット数設定 — TDSIO_SET_RESOLUTIONBIT

サンプリングを行なうビット数を設定する。第2引数には**TDSIO_SET_RESOLUTIONBIT**を、第3引数にはビット数設定情報を保持する変数(32bit)のアドレスを指定する。

- データ幅指定

ビット数	指定マクロ
1bit	TDSIO_SAMPLING_1BIT
2bit	TDSIO_SAMPLING_2BIT
4bit	TDSIO_SAMPLING_4BIT
8bit	TDSIO_SAMPLING_8BIT

(使用例)

```
/* 1bit を設定する場合 */
unsigned int resolutionbit;


resolutionbit = TDSIO_SAMPLING_1BIT;
ioctl(fd, TDSIO_SET_RESOLUTIONBIT, &resolutionbit);
```

6.3.10 フィルタ周波数設定 — TDSIO_SET_FILTER

サンプリングを行なうフィルタ周波数を設定する。第2引数には**TDSIO_SET_FILTER**を、第3引数にはフィルタ周波数設定情報を保持する変数 (32bit) のアドレスを指定する。

- フィルタ周波数指定

フィルタ周波数	指定マクロ
16M	TDSIO_SAMPLING_16M
8M	TDSIO_SAMPLING_8M
4M	TDSIO_SAMPLING_4M
2M	TDSIO_SAMPLING_2M
Thru	TDSIO_SAMPLING_THRU

(使用例)

```
/* Thru を設定する場合 */
unsigned int filter;


filter = TDSIO_SAMPLING_THRU;
ioctl(fd, TDSIO_SET_FILTER, &filter);
```

6.3.11 サンプリング設定 — TDSIO_SET_FSAMPLE

サンプリングを行なうサンプリング周波数を設定する。第2引数には**TDSIO_SET_FSAMPLE**を、第3引数にはサンプリング周波数設定情報を保持する変数(32bit)のアドレスを指定する。

- 周波数指定

サンプリング周波数	指定マクロ
40kHz	TDSIO_SAMPLING_40KHZ
100kHz	TDSIO_SAMPLING_100KHZ
200kHz	TDSIO_SAMPLING_200KHZ
500kHz	TDSIO_SAMPLING_500KHZ
1MHz	TDSIO_SAMPLING_1MHZ
2MHz	TDSIO_SAMPLING_2MHZ
4MHz	TDSIO_SAMPLING_4MHZ
8MHz	TDSIO_SAMPLING_8MHZ
16MHz	TDSIO_SAMPLING_16MHZ
32MHz	TDSIO_SAMPLING_32MHZ

(使用例)

```
/* 1MHz を設定する場合 */
unsigned int fsample;


fsample = TDSIO_SAMPLING_1MHZ;
ioctl(fd, TDSIO_SET_FSAMPLE, &fsample);
```

6.3.12 チャンネル数設定取得 — TDSIO_GET_CHMODE

現在のチャンネル数設定を取得する。第2引数には**TDSIO_GET_CHMODE**を、第3引数には取得した設定情報を格納する変数(32bit)のアドレスを指定する。得られるチャンネル数設定情報は、6.3.8のチャンネル数設定時に指定するものと同じ形式になっている。

(使用例)

```
unsigned int chmode;
```

```
ioctl(fd, TDSIO_GET_CHMODE, &chmode);
if (chmode == TDSIO_SAMPLING_4CH) {
        /* 4CH */
}
```

6.3.13 ビット数設定取得 — TDSIO_GET_RESOLUTIONBIT

現在のビット数設定を取得する。第2引数には**TDSIO_GET_RESOLUTIONBIT**を、第3引数には取得した設定情報を格納する変数(32bit)のアドレスを指定する。得られるビット数設定情報は、6.3.9のビット数設定時に指定するものと同じ形式になっている。

(使用例)

```
unsigned int resolutionbit;

ioctl(fd, TDSIO_GET_RESOLUTIONBIT, &resolutionbit);
if (resolutionbit == TDSIO_SAMPLING_4BIT) {
        /* 4Bit */
}
```

6.3.14 フィルタ周波数設定取得 — TDSIO_GET_FILTER

現在のフィルタ周波数設定を取得する。第2引数には**TDSIO_GET_FILTER**を、第3引数には取得した設定情報を格納する変数(32bit)のアドレスを指定する。得られるフィルタ周波数設定情報は、6.3.10のフィルタ周波数設定時に指定するものと同じ形式になっている。

(使用例)

```
unsigned int filter;

ioctl(fd, TDSIO_GET_FILTER, &filter);
```

```
if (filter == TDSIO_SAMPLING_THRU) {
        /* Thru */
}
```

6.3.15 サンプリング周波数設定取得 — TDSIO_GET_FSAMPLE

現在のサンプリング周波数設定を取得する。第 2 引数には **TDSIO_GET_FSAMPLE** を、第 3 引数には取得した設定情報を格納する変数 (32bit) のアドレスを指定する。得られるサンプリング周波数設定情報は、6.3.11 のサンプリング周波数設定時に指定するものと同じ形式になっている。

(使用例)

```
unsigned int fsample;

ioctl(fd, TDSIO_GET_FSAMPLE, &fsample);
if (fsample == TDSIO_SAMPLING_4MHZ) {
        /* 4MHz */
}
```

6.3.16 ステータスの取得 — TDSIO_GET_STATUS

サンプラーのステータス情報を取り出す。第 2 引数には **TDSIO_GET_STATUS** を、第 3 引数にはステータス情報を格納すべき変数 (32bit) のアドレスを指定する。[4]

(使用例)

```
unsigned int stat;

ioctl(fd, TDSIO_GET_STATUS, &stat);
```

[4] 得られるステータス情報はサンプラーの持つレジスタから得られる値そのものになる。

6.3.17 サンプリング開始 — TDSIO_SAMPLING_START

サンプリング開始を指示する。第 2 引数には **TDSIO_SAMPLING_START** を指定する。

(使用例)

```
ioctl(fd, TDSIO_SAMPLING_START);
```

6.3.18 サンプリング停止 — TDSIO_SAMPLING_STOP

サンプリング停止を指示する。第 2 引数には **TDSIO_SAMPLING_STOP** を指定する。なお、ホスト PC 上においてデバイスを open() しているプロセスがすべて終了した場合には、サンプリングは自動的に停止される。

(使用例)

```
ioctl(fd, TDSIO_SAMPLING_STOP);
```

6.3.19 DC オフセット設定 — TDSIO_SET_DCOFFSET

現在の DC オフセット値を設定する。第 2 引数には **TDSIO_SET_DCOFFSET** を、第 3 引数には設定する情報を格納する構造体 (struct tdsio) のアドレスを指定する。DC オフセット (1CH) の値を取得したい場合は、struct tdsio のメンバ key に 0 を、2CH の値を取得したい場合は 1 を設定する。

(使用例)

```
struct tdsio tdsio;


tdsio.key = 0;          /* DCoffset 1 */
tdsio.value = 0x10;
ioctl(fd, TDSIO_SET_DCOFFSET, &tdsio);
```

6.3.20 DC オフセット取得 — TDSIO_GET_DCOFFSET

現在の DC オフセット値設定を取得する。第 2 引数には **TDSIO_GET_DCOFFSET** を、第 3 引数には取得した設定情報を格納する構造体 (struct tdsio) のアドレスを指定する。DC オフセット (1CH) の値を取得したい場合は、struct tdsio のメンバ key に 0 を、2CH の値を取得したい場合は 1 を設定する。

(使用例)

```
struct tdsio tdsio;


tdsio.key = 0;          /* DCoffset 1 */
ioctl(fd, TDSIO_GET_DCOFFSET, &tdsio);

tdsio.key = 1;          /* DCoffset 2 */
ioctl(fd, TDSIO_GET_DCOFFSET, &tdsio);
```

6.3.21 サンプラー FIFO リセット — TDSIO_BUFFER_RESET

サンプラー上の FIFO をリセットする。第 2 引数には **TDSIO_BUFFER_RESET** を指定する。

(使用例)

```
ioctl(fd, TDSIO_BUFFER_RESET);
```

6.3.22 1PPS 入力状態チェック — TDSIO_GET_1PPSINPUT

サンプラーへの 1PPS 入力状態をチェックする。第 2 引数には **TDSIO_GET_1PPSINPUT** を指定する。

(使用例)

```
unsigned int pps;
```

```
ioctl(fd, TDSIO_GET_1PPSINPUT, &pps);
if (pps == 0) {
        /* no 1PPS Input */
}
```

6.3.23 基準信号入力状態チェック — TDSIO_GET_REFINPUT

サンプラーへの基準信号入力状態をチェックする。第2引数には**TDSIO_GET_REFINPUT**を指定する。

(使用例)

```
unsigned int ref;

ioctl(fd, TDSIO_GET_REFINPUT, &ref);
if (ref == 0) {
        /* no Ref Input */
}
```

6.3.24 5/10MHz 入力信号判定 — TDSIO_GET_10MHZINPUT

サンプラーへの基準信号入力が5MHzか10MHzかをチェックする。5MHzの場合は1が、10MHzの場合は0が設定される。第2引数には**TDSIO_GET_10MHZINPUT**を指定する。

(使用例)

```
unsigned int rhz;

ioctl(fd, TDSIO_GET_10MHZINPUT, &rhz);
if (rhz == 0) {
```

```
        /* 10MHz */
}
```

6.3.25 AUX データ設定 — TDSIO_SET_AUX

AUX データを設定する。第 2 引数には **TDSIO_SET_AUX** を、第 3 引数には AUX データを格納する構造体 (struct tdsauxio) のアドレスを指定する。メンバ auxsize に AUX データのサイズを、メンバ auxfield に AUX データを設定する。auxfield に設定できるデータは最大 256 バイトである。

(使用例)

```
struct tdsauxio tdsauxio;

tdsauxio.auxsize = 10;
memcpy(tdsio.auxfield, "TEST AUX!", tdsauxio.auxsize);
ioctl(fd, TDSIO_SET_AUX, &tdsauxio);
```

6.3.26 AUX データ取得 — TDSIO_GET_AUX

AUX データを取得する。第 2 引数には **TDSIO_GET_AUX** を、第 3 引数には AUX データを格納する構造体 (struct tdsauxio) のアドレスを指定する。取得後、メンバ auxsize に AUX データのサイズが、メンバ auxfield に AUX データが取得できる。

(使用例)

```
struct tdsauxio tdsauxio;
char buf[256];

ioctl(fd, TDSIO_GET_AUX, &tdsauxio);
memcpy(buf, tdsio.auxfield, tdsauxio.auxsize);
```

6.3.27 エラー情報取得 — TDSIO_GET_ERROR

デバイス操作において発生したエラー情報を取得する。第2引数には**TDSIO_GET_ERROR**を、第3引数にはエラー情報を格納すべき変数 (32bit) のアドレスを指定する。
ホスト PC がブートした直後、あるいは、6.3.1 初期化が実行された場合に、エラー情報がリセットされる。それ以降、エラーが発生した場合に、その種別に関する情報が保持され、さらに次のエラーが発生した場合には、エラー情報が上書きされる。従って、エラー情報取得を行なった場合に得られる値は、最も最近に発生したエラーの種別を示すことになる。表 1 に示すエラー種別を定義している。[5]

(使用例)

```
int err;


ioctl(fd, TDSIO_GET_ERROR, &err);
```

6.3.28 レジスタ読みだし — TDSIO_REGIO_READ

任意のレジスタの値を読み出す。第 2 引数には **TDSIO_REGIO_READ** を、第 3 引数にはレジスタの値を格納する構造体 (struct tdsio_regio) のアドレスを指定する。読み出し後、メンバ address にレジスタアドレスが、メンバ value に現在の値が格納される。

(使用例)

```
struct tdsio_regio regio;


ioctl(fd, TDSIO_REGIO_READ, &regio);
printf("address=0x%02x, value=0x%02x\n", regio.address, regio.value);
```

[5] vlbi-usb-linux/include/tdsio.h 参照

表 1: エラー種別

エラー値	エラー種別
0	エラーなし。(正常)
1	以下の分類に当てはまらないエラー。
2	サポートしていない操作あるいはパラメータを指定した。
3	サンプリング開始状態で行なうべき操作を、サンプリングが開始されていない状態で指定した。
4	サンプリング停止状態で行なうべき操作を、サンプリングが開始されている状態で指定した。
5	サンプリング設定 (周波数/データ幅/チャネル数) がなされた状態で行なうべき操作を、サンプリング設定されていない状態で指定した。
6	ボード上の FIFO がオーバーフローした。
7	外部 1PPS 入力信号が検出できなかった。
8	外部 10MHz 入力信号が検出できなかった。
9	外部時刻入力信号が検出できなかった。
10	1PPS 同期がなされた後で行なうべき操作を、1PPS 同期されていない状態で指定した。
11	時刻同期 (あるいは時刻設定) がなされたあとで行なうべき操作を、時刻同期 (あるいは時刻設定) されていない状態で指定した。
12	他のプロセス等が read() を実行している状態で、さらに read() を行なおうとした。
13	PCI ボード用の拡張ボードが接続されていないエラー。
14	DMA バッファが無くなった。
15	ユーザランドとの I/O 処理中でエラーになった。
16	DMA 処理がエラーになった。
17	データサンプラーから一定時間内に応答が無かった。
18	ワード単位でアラインされていないデータを検出した。
19	サンプリングデータ内で想定した位置にヘッダが見つからなかった。
20	サンプリング開始時のゴミデータの中にもヘッダと認識可能なデータが存在した。

6.3.29 バージョン情報読みだし — TDSIO_GET_VERSION

ドライバのバージョン情報値を読み出す。第 2 引数には **TDSIO_GET_VERSION** を、第 3 引数にはバージョン情報値を格納する変数のアドレスを指定する。バージョン情報はメジャー、マイナー、リビジョン番号の三つの値からなる。それぞれの番号を取り出すには次のマクロを使用する。

TDS_VERSION_MAJOR()	メジャー番号を取り出すマクロ
TDS_VERSION_MINOR()	マイナー番号を取り出すマクロ
TDS_VERSION_REVISION()	リビジョン番号を取り出すマクロ

(使用例)

```
uint32_t version;


ioctl(fd, TDSIO_GET_VERSION, &version);
printf("version=%02d.%02d.%02d\n", TDS_VERSION_MAJOR(version),
    TDS_VERSION_MINOR(version), TDS_VERSION_REVISION(version));
```

7 サンプルプログラム

サンプルプログラムとして vlbi-usb-linux/example 以下にある sample.c を参照して下さい。

8 ライブラリ

システムコールを抽象化するためのライブラリを作成しています。ライブラリはドライバモジュールと同時にビルド、インストールされています。

8.1 使用時の注意

ライブラリを使用する場合には、以下のようにする必要があります。

- **<sys/tdsio.h>** をインクルードする。
  ソースファイルに
  #include <sys/tdsio.h>
  と記述する。

- libtds をリンクする。
  リンク時オプションに**-ltds** を指定する。

8.2 ライブラリ関数の使用

各ライブラリ関数の使用法は、以下の通りです。UTDS ドライバは一部の関数をサポートしていません。それらは 9260 データサンプラー用 TDS ドライバへの互換性として残してあります。

1. TdsOpen()

   デバイスをオープンする。

引数 1	**char ***(文字列)	デバイス名を指定する。NULL を指定すると、デフォルトで**"/dev/tds0"** が使用される。
戻り値	**tdsdev_t ***	この変数の詳細は、利用者側で関知している必要はない。オープンに失敗した場合には NULL が返される。この戻り値は、以下、すべてのライブラリ関数で引数として利用する。

2. TdsClose()

   デバイスをクローズする。

引数 1	**tdsdev_t ***	1 の TdsOpen() で返された値を指定する。
戻り値	**int**	成功すると 0 が返される。

3. TdsInit()

   ドライバ内変数を初期化する。

引数 1	**tdsdev_t ***	1 の TdsOpen() で返された値を指定する。
戻り値	**int**	成功すると 0 が返される。失敗すると 0 以外の値が返される。

4. TdsClrBuffer()

   サンプラー上の FIFO を初期化する。

引数 1	**tdsdev_t ***	1 の TdsOpen() で返された値を指定する。
戻り値	**int**	成功すると 0 が返される。失敗すると 0 以外の値が返される。

5. TdsSync1pps()

   外部 1PPS 入力信号同期を行なう。

引数 1	**tdsdev_t ***	1 の TdsOpen() で返された値を指定する。
戻り値	**int**	成功すると 0 が返される。失敗すると 0 以外の値が返される。

6. TdsSyncTime()

外部時刻入力信号同期を行なう。

引数 1	**tdsdev_t ***	1 の TdsOpen() で返された値を指定する。
戻り値	**int**	成功すると 0 が返される。失敗すると 0 以外の値が返される。

7. TdsSetTime()

サンプラーに時刻を設定する。

引数 1	**tdsdev_t ***	1 の TdsOpen() で返された値を指定する。
引数 2	**tds_time_t ***	時刻を保持している変数のアドレスを指定する。NULL を指定すると、OS 側から時刻を取得して使用する。
戻り値	**int**	成功すると 0 が返される。失敗すると 0 以外の値が返される。

tds_time_t の型定義は以下の通り。

```
struct tds_time {
    int sec;
    int sec_carry;
    int day;
    int day_carry;
};

typedef struct tds_time tds_time_t;
```

sec	00:00:00 からの通算秒数。
sec_carry	秒のキャリー。 キャリーがあれば 1、なければ 0 であるが、時刻設定の場合には、このフィールドの値は無視される。
day	1 月 1 日からの通算日数。
day_carry	day のキャリー。キャリーがあれば 1、なければ 0 であるが、時刻設定の場合には、このフィールドの値は無視される。

8. TdsGetTime()

サンプラーが保持している時刻を取得する。

引数 1	**tdsdev_t ***	1 の TdsOpen() で返された値を指定する。
引数 2	**tds_time_t ***	時刻を格納する変数のアドレスを指定する。
戻り値	**int**	成功すると 0 が返される。失敗すると 0 以外の値が返される。

tds_time_t の型定義に関しては、7 を参照。

9. TdsSetYear()

   サンプラーに年情報を設定する。

引数 1	**tdsdev_t ***	1 の TdsOpen() で返された値を指定する。
引数 2	**unsigned int ***	年情報を保持している変数のアドレス指定する。
戻り値	**int**	成功すると 0 が返される。失敗すると 0 以外の値が返される。

10. TdsGetYear()

    サンプラーが保持している年情報を取得する。

引数 1	**tdsdev_t ***	1 の TdsOpen() で返された値を指定する。
引数 2	**unsigned int ***	年情報を格納すべき変数のアドレス指定する。
戻り値	**int**	成功すると 0 が返される。失敗すると 0 以外の値が返される。

11. TdsSetSampling()[6]
    サンプリング設定を行なう。

引数 1	**tdsdev_t ***	1 の TdsOpen() で返された値を指定する。
引数 2	**tds_samp_t ***	サンプリング設定値を保持している変数のアドレス指定する。
戻り値	**int**	成功すると 0 が返される。失敗すると 0 以外の値が返される。

tds_samp_t の型定義は以下の通り。

```
struct tds_samp {
    int freq;
```

[6] 9270 ではサポートしていない

```
    int unit;    /* 'k' or 'M' */
    int width;
    int channel;
};

typedef struct tds_samp tds_samp_t;
```

freq, unit	サンプリング周波数を指定する。 周波数 | freq | unit 40kHz | 40 | 'k' 100kHz | 100 | 'k' 200kHz | 200 | 'k' 500kHz | 500 | 'k' 1MHz | 1 | 'M' 2MHz | 2 | 'M' 4MHz | 4 | 'M' 8MHz | 8 | 'M' 16MHz | 16 | 'M'
width	サンプリングビット幅を指定する。 ビット幅 | width 1 | 1 2 | 2 4 | 4 8 | 8
channel	サンプリングチャネル数を指定する。 チャネル数 | channel 1 | 1 4 | 4

12. TdsGetSampling()[7]

サンプラーのサンプリング設定情報を取得する。

引数 1	**tdsdev_t ***	1 の TdsOpen() で返された値を指定する。
引数 2	**tds_samp_t ***	サンプリング設定値を格納すべき変数のアドレス指定する。
戻り値	**int**	成功すると 0 が返される。失敗すると 0 以外の値が返される。

tds_samp_t の定義については、**??**を参照。

[7] 9270 ではサポートしていない

13. TdsSetSampling2()

サンプリング設定を行なう。

引数 1	**tdsdev_t ***	1 の TdsOpen() で返された値を指定する。
引数 2	**int ***	サンプリング周波数設定値を保持している変数のアドレス指定する。
引数 3	**int ***	フィルタ周波数設定値を保持している変数のアドレス指定する。
引数 4	**int ***	ビット数設定値を保持している変数のアドレス指定する。
引数 5	**int ***	チャンネル数設定値を保持している変数のアドレス指定する。
戻り値	**int**	成功すると 0 が返される。失敗すると 0 以外の値が返される。

14. TdsGetSampling2()

サンプラーのサンプリング設定情報を取得する。

引数 1	**tdsdev_t ***	1 の TdsOpen() で返された値を指定する。
引数 2	**int ***	サンプリング周波数設定値を格納すべき変数のアドレス指定する。
引数 3	**int ***	フィルタ周波数設定値を格納すべき変数のアドレス指定する。
引数 4	**int ***	ビット数設定値を格納すべき変数のアドレス指定する。
引数 5	**int ***	チャンネル数設定値を格納すべき変数のアドレス指定する。
戻り値	**int**	成功すると 0 が返される。失敗すると 0 以外の値が返される。

15. TdsSetDcoffset()

サンプラーに DC オフセット値情報を設定する。

引数 1	**tdsdev_t ***	1 の TdsOpen() で返された値を指定する。
引数 2	**int ***	DC オフセット値を設定すべきチャンネルの番号を保持している変数のアドレス指定する。
引数 3	**unsigned int ***	DC オフセット値を保持している変数のアドレス指定する。
戻り値	**int**	成功すると 0 が返される。失敗すると 0 以外の値が返される。

16. TdsGetDcoffset()

サンプラーの DC オフセット値情報を取得する。

引数 1	**tdsdev_t ***	1 の TdsOpen() で返された値を指定する。
引数 2	**int ***	取得する DC オフセット値のチャンネルの番号を保持している変数のアドレス指定する。
引数 3	**unsigned int ***	DC オフセット値を格納する変数のアドレス指定する。
戻り値	**int**	成功すると 0 が返される。失敗すると 0 以外の値が返される。

17. TdsStatus()

サンプラーが保持しているステータス情報を取得する。

引数 1	**tdsdev_t ***	1 の TdsOpen() で返された値を指定する。
引数 2	**unsigned int ***	ステータス情報を格納すべき変数のアドレス指定する。得られるステータス情報は、サンプラーのレジスタ値そのものである。
戻り値	**int**	成功すると 0 が返される。失敗すると 0 以外の値が返される。

18. TdsStart()

サンプリングを開始する。

引数 1	**tdsdev_t ***	1 の TdsOpen() で返された値を指定する。
戻り値	**int**	成功すると 0 が返される。失敗すると 0 以外の値が返される。

19. TdsStop()

サンプリングを停止する。

引数 1	**tdsdev_t ***	1 の TdsOpen() で返された値を指定する。
戻り値	**int**	成功すると 0 が返される。失敗すると 0 以外の値が返される。

20. TdsGetData()

サンプラーからダブルワード (4 バイト) のデータ読み込む。

引数 1	**tdsdev_t ***	1 の TdsOpen() で返された値を指定する。
引数 2	**unsigned int ***	データを格納すべき変数のアドレスを指定する。
戻り値	**int**	成功すると 0 が返される。失敗すると 0 以外の値が返される。

21. TdsRead()

サンプラーから指定数のデータ読み込む。

引数 1	**tdsdev_t ***	1 の TdsOpen() で返された値を指定する。
引数 2	**int ***	読み込むべきデータのバイト数を保持している変数のアドレスを指定する。このアドレスには、実際に読み込んだデータのバイト数が格納される。
引数 3	**unsigned char ***	データを格納すべき領域 (配列など) の先頭アドレスを指定する。
戻り値	**int**	成功すると 0 が返される。失敗すると 0 以外の値が返される。

22. TdsSetAux()

サンプリングデータのヘッダに付加する AUX データを設定する。

引数 1	**tdsdev_t ***	1 の TdsOpen() で返された値を指定する。
引数 2	**int ***	設定する AUX データのバイト数を保持している変数のアドレスを指定する。
引数 3	**char ***	設定する AUX データを保持している領域 (配列など) の先頭アドレスを指定する。
戻り値	**int**	成功すると 0 が返される。失敗すると 0 以外の値が返される。

23. TdsSetAuxFile()

    指定したファイルから読み込んだ AUX データを、サンプリングデータのヘッダに付加する AUX データとして設定する。

引数 1	**tdsdev_t ***	1 の TdsOpen() で返された値を指定する。
引数 2	**char ***	設定する AUX データとして読み込むファイルへのパスの文字列を保持している領域 (配列など) の先頭アドレスを指定する。
戻り値	**int**	成功すると 0 が返される。失敗すると 0 以外の値が返される。

24. TdsGetAux()

    サンプリングデータのヘッダに付加する AUX データを取得する。

引数 1	**tdsdev_t ***	1 の TdsOpen() で返された値を指定する。
引数 2	**int ***	設定している AUX データのバイト数を格納する変数のアドレスを指定する。
引数 3	**char ***	設定している AUX データを格納する領域 (配列など) の先頭アドレスを指定する。
戻り値	**int**	成功すると 0 が返される。失敗すると 0 以外の値が返される。

25. TdsGetAuxFile()

    指定したファイルに、設定した AUX データを書き出す。

引数 1	**tdsdev_t ***	1 の TdsOpen() で返された値を指定する。
引数 2	**char ***	設定した AUX データを書き出すファイルへのパスの文字列を保持している領域 (配列など) の先頭アドレスを指定する。
戻り値	**int**	成功すると 0 が返される。失敗すると 0 以外の値が返される。

26. TdsError()

エラー情報を取得する。

引数 1	**tdsdev_t ***	1 の TdsOpen() で返された値を指定する。
引数 2	**int ***	取得したエラー情報を格納すべき変数のアドレスを指定する。
引数 3	**char ***	エラーに対応するメッセージを格納すべきアドレスを指定する。
戻り値	**int**	成功すると 0 が返される。失敗すると 0 以外の値が返される。

27. TdsGetVersion()

ドライバとライブラリのバージョン情報値を取得する。

引数 1	**tdsdev_t ***	1 の TdsOpen() で返された値を指定する。
引数 2	**int ***	ドライバのバージョン情報を格納すべき変数のアドレスを指定する。
引数 3	**int ***	ライブラリのバージョン情報を格納すべき変数のアドレスを指定する。
戻り値	**int**	成功すると 0 が返される。失敗すると 0 以外の値が返される。

バージョン情報はメジャー、マイナー、リビジョン番号の三つの値からなる。それぞれの番号を取り出すには次のマクロを使用する。

TDS_VERSION_MAJOR()	メジャー番号を取り出すマクロ
TDS_VERSION_MINOR()	マイナー番号を取り出すマクロ
TDS_VERSION_REVISION()	リビジョン番号を取り出すマクロ

8.3 ライブラリ関数を使用したサンプルプログラム

サンプルプログラムとして vlbi-usb-linux/example 以下にある libcall.c を参照して下さい。