TDS ドライバ 取扱説明書

株式会社 創夢
第二開発部

平成 14 年 3 月 31 日

1 はじめに

本文書は、日本通信機株式会社製の PCI ボード「時刻同期式データサンプラーカード」を、**FreeBSD 4.3 にて動作させるために作成したドライバについて記述したものです。**

2 動作確認環境

- **FreeBSD 4.3-RELEASE i386 プラットフォーム**[1]
- **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 1MHz/1bit/4ch 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。

```
tds_driver/
   |
   +dev/
   |   |
   |   +MAKEDEV /dev/MAKEDEV(TDS ドライバ対応版)
   |
   +usr/
       |
       +share/
       |   |
       |   +doc/ ドキュメントディレクトリ
       |   |   |
       |   |   +tds/
       |   |       |
       |   |       +manual.ps 取扱説明書 (PostScript 形式)
       |   |       |
       |   |       +manual.tex 取扱説明書 (LaTeX ソース)
       |   |
       |   +examples/
       |       |
       |       +tds/
       |           |
       |           +*.c サンプルプログラム (C ソース)
       |
       +src/ ソースディレクトリ
           |
           +lib/
           |   |
           |   +Makefile
           |   |
           |   +libtds/
           |       |
           |       +Makefile
```

[1] マルチプロセッサシステムにおける動作、および、ターゲットボードを同一 PC に複数装着した場合の動作は未確認です。

```
|     |
|     +libtds.c ライブラリソースファイル
|
+sys/
   |
   +conf/
   |  |
   |  +files.i386
   |  |
   |  +majors
   |
   +i386/
   |  |
   |  +conf/
   |     |
   |     +NITSUKI カーネルコンフィグレーションファイル
   |                                          サンプル
   +modules/ ローダブルモジュール用ディレクトリ
   |  |
   |  +Makefile
   |  |
   |  +tds/
   |     |
   |     +Makefile
   |
   +pci/
   |  |
   |  +tds.c TDS ドライバソースファイル
   |  |
   |  +tdsvar.h TDS ドライバソースファイル
   |
   +sys/
      |
      +tdsio.h TDS ドライバソースファイル
```

4 カーネルへの組み込み

1. **カーネルへの静的組み込み**

   (a) **/usr/src/sys/pci ディレクトリに、以下のファイルをコピーする。**

   ```
   tds_driver/usr/src/sys/pci/tds.c
   ```

   ```
   tds_driver/usr/src/sys/pci/tdsvar.h
   ```

   (b) **/usr/src/sys/sys ディレクトリに、以下のファイルをコピーする。**

   ```
   tds_driver/usr/src/sys/sys/tdsio.h
   ```

(c) **/usr/src/sys/conf/files.i386 ファイルに以下を追加する。**

```
pci/tds.c                       optional        tds
```

(d) **/usr/src/sys/conf/majors ファイルに以下を追加する。**

```
222   tds       Time Sync Data Sampler (Nitsuki)
```

(e) **/usr/src/sys/i386/conf ディレクトリにあるカーネルコンフィグレーション ファイルに以下を追加する。**

```
device          tds
```

(f) **通常のカーネル再構築を行ない、カーネルを置き換える。**[2]

```
# cd /usr/src
# make buildkernel KERNCONF=MYKERNEL
# make installkernel KERNCONF=MYKERNEL
```

(g) **/dev/MAKEDEV を以下のファイルと置き換える。**

```
tds_driver/dev/MAKEDEV
```

(h) **デバイスノードを作成する。**

```
# cd /dev
# sh MAKEDEV tds0
```

デバイスファイルは、/dev/tds0 となります。

(i) **再起動する。**

[2] この例ではカーネルコンフィグレーションファイルを **MYKERNEL としている。**

2. ローダブルモジュールの作成

第 1 節に記述した他に FreeBSD 4.3 でサポートされているローダブルモジュールによる組み込み方法を記します。一部、操作が重複していますが、第 1 節ですでに行なった場合は再度行なう必要はありません。

(a) /usr/src/sys/pci ディレクトリに、以下のファイルをコピーする。

```
tds_driver/usr/src/sys/pci/tds.c
tds_driver/usr/src/sys/pci/tdsvar.h
```

(b) /usr/src/sys/sys ディレクトリに、以下のファイルをコピーする。

```
tds_driver/usr/src/sys/sys/tdsio.h
```

(c) /usr/src/sys/conf/files.i386 ファイルに以下を追加する。

```
pci/tds.c                       optional        tds
```

(d) /usr/src/sys/conf/majors ファイルに以下を追加する。

```
222   tds       Time Sync Data Sampler (Nitsuki)
```

(e) /usr/src/sys/modules/Makefile を以下のファイルと置き換える。

```
tds_driver/usr/src/sys/modules/Makefile
```

(f) /usr/src/sys/modules/tds ディレクトリを作成する。

(g) 上記 (f) で作成したディレクトリに、以下のファイルをコピーする。

```
tds_driver/usr/src/sys/modules/tds/Makefile
```

(h) モジュールを構築、インストールする。

```
# cd /usr/src/sys/modules/tds
# make
# make install
```

(i) **/dev/MAKEDEV を以下のファイルと置き換える。**

```
tds_driver/dev/MAKEDEV
```

(j) **デバイスノードを作成する。**

```
# cd /dev
# sh MAKEDEV tds0
```

デバイスファイルは、/dev/tds0 となります。

(k) **モジュールをロードする。**

```
# kldload tds
```

5 システムコールインタフェース

5.1 サポートしているシステムコール

1. **open()**
2. **close()**
3. **read()**
4. **ioctl()**
5. **write()**[3]

[3] エントリポイントは用意しているが、実際に write() を行なうと、ドライバがエラーを返すようにしている。

5.2 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、FreeBSD 標準のシステムコールの仕様に準じている。

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。なお、別プロセス等により open() 操作が行なわれている場合に、さらに open() を行なおうとすると、デバイスビジーエラーが発生する。

   (使用例)

   ```
   int fd;

   fd = open("/dev/tds0", O_RDONLY);
   ```

2. close()

   デバイスをクローズする。引数には 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);
   ```

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 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。なお、別プロセス等により read() 操作が行なわれている場合に、さらに read() を行なおうとすると、デバイスビジーエラーが発生する。

（使用例）

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

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

4. ioctl()

   デバイスの各種操作を行なう。第 1 引数には 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。

   (a) **初期化**

   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。

   - **サンプリング設定がなされていない状態 (4h 参照)**
   - **1PPS 同期がなされていない状態 (4c 参照)**
   - **時刻同期がなされていない状態 (4d 参照)**
   - **時刻設定がなされていない状態 (4e 参照)**
   - **エラー情報をリセットした状態 (4m 参照)**
   - **FIFO オーバーフローカウンタをリセットした状態 (4o 参照)**

   （使用例）

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

   (b) **デバイス FIFO クリア**

   ボード上の 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);
```

(c) **1PPS 同期**

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);
```

(d) **時刻同期**

外部時刻入力信号への同期を行なう。第 2 引数には TDSIO_SYNC_TIME を指定する。外部時刻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_TIME);
```

(e) **時刻設定**

ボードに時刻を設定する。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。[4]
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。

[4] 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

(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);
```

(f) **時刻取得**

ボード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、4e 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);
```

(g) **年情報取得**

ボード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。[5]

(使用例)

```
unsigned int year;

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

(h) **サンプリング設定**

サンプリングを行なう周波数/データ幅/チャネル数を設定する。第 2 引数には TDSIO_SET_SAMPLING を、第 3 引数にはサンプリング設定情報を保持する変数 (32bit) のアドレスを指定する。

サンプリング情報は、周波数指定、データ幅、チャネル数を表すマクロの論理和の形式で指定する。

[5] ただし、ボード自身ではこの機能が未実装であるので、この操作を行なうと、ドライバがエラーを返すようにしている。

- **周波数指定**

サンプリング周波数	指定マクロ
40kHz	TDS_SAMPLING_40KHZ
100kHz	TDS_SAMPLING_100KHZ
200kHz	TDS_SAMPLING_200KHZ
500kHz	TDS_SAMPLING_500KHZ
1MHz	TDS_SAMPLING_1MHZ
2MHz	TDS_SAMPLING_2MHZ
4MHz	TDS_SAMPLING_4MHZ
8MHz	TDS_SAMPLING_8MHZ
16MHz	TDS_SAMPLING_16MHZ

- **データ幅指定**

サンプリングデータ幅	指定マクロ
1bit	TDS_SAMPLING_1BIT
2bit	TDS_SAMPLING_2BIT
4bit	TDS_SAMPLING_4BIT
8bit	TDS_SAMPLING_8BIT

- **チャネル数指定**[6]

サンプリングチャネル数	指定マクロ
1ch	TDS_SAMPLING_1CH
4ch	TDS_SAMPLING_4CH

(使用例)

```
/* 1MHz/1bit/4ch を設定する場合 */
unsigned int samp;

samp = TDS_SAMPLING_1MHZ | TDS_SAMPLING_1BIT | TDS_SAMPLING_4CH;

ioctl(fd, TDSIO_SET_SAMPLING, &samp);
```

(i) **サンプリング設定取得**

現在のサンプリング設定を取得する。第2引数にはTDSIO_GET_SAMPLING 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

[6] 4ch サンプリングは現行ボードではサポートされていないので、デバイスドライバがエラーを返すようにしている。

する。得られるサンプリング設定情報は、4h のサンプリング設定時に指定するものと同じ形式になっている。

簡易マクロとして、以下を用意している。

マクロ	機能
TDS_GET_SAMP_FREQ(arg)	引数 arg から周波数部を取り出す。
TDS_GET_SAMP_WIDTH(arg)	引数 arg からビット幅部を取り出す。
TDS_GET_SAMP_CH(arg)	引数 arg からチャネル数部を取り出す。

(使用例)

```
unsigned int samp;
unsigned int freq;
unsigned int width;
unsigned int ch;

ioctl(fd, TDSIO_GET_SAMPLING, &samp);

freq = TDS_GET_SAMP_FREQ(samp);
width = TDS_GET_SAMP_WIDTH(samp);
ch = TDS_GET_SAMP_CH(samp);

if (freq == TDS_SAMPLING_4MHZ) {
/* 4MHz */
}
```

(j) ステータスの取得

ボード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。[7]

[7] 得られるステータス情報はボードの持つレジスタから得られる値そのものになる。

(使用例)

```
unsigned int stat;

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

(k) **サンプリング開始**

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);
```

(l) **サンプリング停止**

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);
```

(m) **エラー情報取得**

ボード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、4a 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。

エラー種別は以下を定義している。[8]

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。

(使用例)

```
int err;

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

[8] tds_driver/usr/src/sys/sys/tdsio.h 参照

(n) **FIFO オーバーフロー検知許容回数設定**

ボード上の FIFO オーバーフローの許容回数を設定する。この設定値までの回数であれば、FIFO オーバーフローを検知してもサンプリングを継続する。この設定値を越えると自動的にサンプリングを停止する。

第 2 引数には TDSIO_SET_ALLOW_OVERFLOW を、第 3 引数には指定する回数を保持している変数 (32bit) のアドレスを指定する。

デフォルトではサンプリング中に 16 回以上オーバーフローが検出された場合にサンプリングを停止するようにしている。

(使用例)

```
int count = 128;

ioctl(fd, TDSIO_SET_ALLOW_OVERFLOW, &count);
```

(o) **FIFO オーバーフロー回数取得**

ドライバが実際にボード上の FIFO オーバーフローを検知した回数を取得する。このカウンタ値は、4a の初期化操作によってリセットされる。

第 2 引数には TDSIO_GET_OVERFLOW_COUNT を、第 3 引数には取得した回数を格納すべき変数 (32bit) のアドレスを指定する。

(使用例)

```
int count;

ioctl(fd, TDSIO_GET_OVERFLOW_COUNT, &count);
```

6　サンプルプログラム

以下に、サンプルプログラムを示します。ここでは、8MHz/1bit/1ch のサンプリングを行ない、そのデータをバイナリのままファイルに書き出しています。

```
#include <stdio.h>
#include <errno.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/tdsio.h>


int main(void)
{
    char dev[] = "/dev/tds0";
    char outfile[] = "./tds.data";
    int fd_in;
    int fd_out;
    mode_t ofilemode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;
    int rv;
    u_int32_t sampling;
    u_int32_t buf[BUFSIZ * 2];


    /* デバイスのオープン */
    fd_in = open(dev, O_RDONLY);
    if (fd_in == -1) {
        perror("device open() error ");
        return -1;
    }

    /* 出力ファイルのオープン */
    fd_out = open(outfile, O_WRONLY | O_CREAT | O_TRUNC, ofilemode);
    if (fd_out == -1) {
        perror("output file open() error");
        goto end;
    }

    /* サンプリング設定 */
    sampling = TDS_SAMPLING_1MHZ | TDS_SAMPLING_1BIT | TDS_SAMPLING_4CH;
    rv = ioctl(fd_in, TDSIO_SET_SAMPLING, &sampling);
    if (rv == -1) {
        perror("ioctl() sampling set failed ");
        goto end;
    }
```

```
else {
    printf("ioctl() sampling set succeed\n");
}

/* デバイスFIFOクリア */
rv = ioctl(fd_in, TDSIO_BUFFER_CLEAR);
if (rv == -1) {
    perror("ioctl() FIFO clear failed ");
    goto end;
}
else {
    printf("ioctl() FIFO clear succeed\n");
}

/* 1PPS同期 */
rv = ioctl(fd_in, TDSIO_SYNC_1PPS);
if (rv == -1) {
    perror("ioctl() 1pps sync failed ");
    goto end;
}
else {
    printf("ioctl() 1pps sync succeed\n");
}

/* 時刻同期 */
rv = ioctl(fd_in, TDSIO_SYNC_TIME);
if (rv == -1) {
    perror("ioctl() time sync failed ");
    goto end;
}
else {
    printf("ioctl() time sync succeed\n");
}

/* サンプリング開始 */
rv = ioctl(fd_in, TDSIO_SAMPLING_START);
if (rv == -1) {
    perror("ioctl() start sampling failed ");
    goto end;
}
else {
    printf("ioctl() start sampling succeed\n");
}

/* データの読み込み */
{
```

```
        int c = 0;
        while (c < BUFSIZ * BUFSIZ) {
            int nbyte = sizeof(buf);

            rv = read(fd_in, buf, nbyte);
            if (rv == -1) {
                perror("read() failed ");
                goto end;
            }
            /* データの書き出し */
            rv = write(fd_out, buf, rv);
            c ++;
        }
    }

    /* サンプリング停止 */
    rv = ioctl(fd_in, TDSIO_SAMPLING_STOP);
    if (rv == -1) {
        perror("ioctl() stop sampling failed ");
        goto end;
    }
    else {
        printf("ioctl() stop sampling succeed\n");
    }

    /* 出力ファイルのクローズ */
    close(fd_out);

 end:
    /* デバイスのクローズ */
    close(fd_in);

    return rv;
}
```

7 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。

7.1 インストール

ライブラリのインストールは、以下のように行ないます。

1. /usr/src/lib/libtds ディレクトリを作成する。

2. 1 で作成したディレクトリに tds_driver/usr/src/lib/libtds ディレクトリ以下のファイルをすべてコピーする。

```
tds_driver/usr/src/lib/libtds/libtds.c
tds_driver/usr/src/lib/libtds/Makefile
```

3. /usr/src/lib/Makefile ファイルを tds_driver/usr/src/lib/Makefile に置き換える。

4. ライブラリを構築、インストールする。

```
# cd /usr/src/lib/libtds
# make
# make install
```

7.2 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。

7.3 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。

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()

   ボード上の 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()

ボードに時刻を設定する。

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()

ボード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. TdsGetYear()[9]

ボード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。**

10. TdsSetSampling()

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

typedef struct tds_samp tds_samp_t;
```

[9] 現在この機能はボード自身に実装されていないため、この関数は必ず失敗する。

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

11. TdsGetSampling()

ボード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。**

12. TdsStatus()

ボード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、ボード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。

13. 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。

14. 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。

15. TdsGetData()

    ボードからダブルワード (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。

16. TdsRead()

    ボード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。

17. 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。

7.4 ライブラリ関数を使用したサンプルプログラム

```
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/tdsio.h>


int main(void)
{
    tdsdev_t *tdsdev;
    tds_samp_t sampling;
    char outfile[] = "./tds.data";
    int fd;
    mode_t ofilemode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;
    int rv;
    unsigned char buf[BUFSIZ * 2];
```

```
/* デバイスのオープン */
tdsdev = TdsOpen(NULL);
if (tdsdev == NULL) {
    perror("TdsOpen() error ");
    return -1;
}

/* 出力ファイルのオープン */
fd = open(outfile, O_WRONLY | O_CREAT | O_TRUNC, ofilemode);
if (fd == -1) {
    perror("output file open() error");
    goto end;
}

/* サンプリング設定 */
sampling.freq = 16;
sampling.unit = 'M';
sampling.width = 1;
sampling.channel = 1;
rv = TdsSetSampling(tdsdev, &sampling);
if (rv != 0) {
    perror("TdsSetSampling() failed ");
    goto end;
}

/* デバイス FIFO クリア */
rv = TdsClrBuffer(tdsdev);
if (rv != 0) {
    perror("TdsClrBuffer() failed ");
    goto end;
}

/* 1PPS 同期 */
rv = TdsSync1pps(tdsdev);
if (rv != 0) {
    perror("TdsSync1pps() failed ");
    goto end;
}

/* 時刻設定 (OS から時刻を取得し、設定する) */
rv = TdsSetTime(tdsdev, NULL);
if (rv != 0) {
    perror("TdsSetTime() failed ");
    goto end;
}
```

```
    /* サンプリング開始 */
    rv = TdsStart(tdsdev);
    if (rv != 0) {
        perror("TdsStart() failed ");
        goto end;
    }

    /* データの読み込み */
    {
        int c = 0;
        while (c < BUFSIZ * BUFSIZ) {
            int nbyte = sizeof(buf);

            rv = TdsRead(tdsdev, &nbyte, buf);
            if (rv != 0) {
                perror("TdsRead() failed ");
                goto end;
            }
            /* データの書き出し */
            rv = write(fd, buf, nbyte);
            c ++;
        }
    }

    /* サンプリング停止 */
    rv = TdsStop(tdsdev);
    if (rv != 0) {
        perror("TdsStop() failed ");
        goto end;
    }

    /* 出力ファイルのクローズ */
    close(fd);

 end:
    /* デバイスのクローズ */
    TdsClose(tdsdev);

    return rv;
}
```