T-Engine Forum Specification

TEF040-S215-01.00.00/ja
2004-2-24

T-Engine デバイスドライバインタフェースライブラリ仕様

Number: TEF40-S215-01.00.00/ja
Title: T-Engine デバイスドライバインタフェースライブラリ仕様
Status: [ ] Working Draft, [ ] Final Draft for Voting, [X] Standard
Date: 2004/01/28
2004/02/24 Voted

目 次

1. はじめに

本仕様書では、T-Kernel/SM デバイス管理仕様に基づくデバイスドライバを開発する際に利用可能な、T-Engine デバイスドライバインタフェースライブラリの仕様に関して説明している。

本ライブラリを利用することで、T-Kernel 上で動作するデバイスドライバの開発が容易になる。

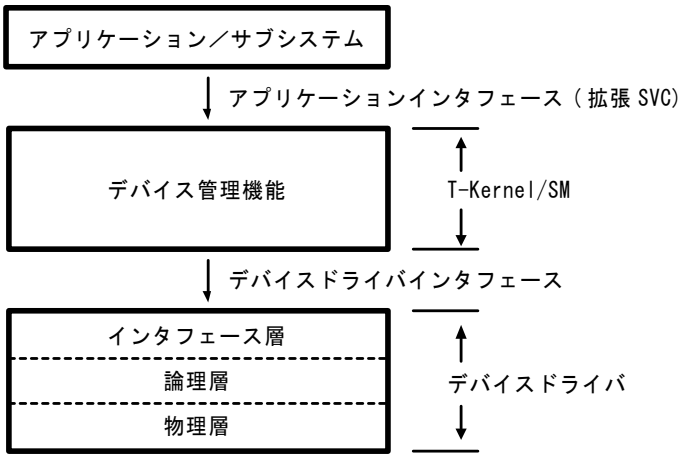
なお本ライブラリは T-Kernel 仕様には含まれない。

2. デバイスドライバ I/F ライブラリ仕様

2.1 デバイスドライバ I/F 層とは

デバイスドライバは、インタフェース層・論理層・物理層の 3 つのレイヤで構成することにより、保守や異なるハードウエア間の移植が行いやすくなる。インタフェース層は T-Kernel のデバイス管理機能とのインタフェース処理部分、論理層は各デバイスに共通でハードウエアコントローラに依存しない部分、そして物理層は実際にハードウエアコントローラを制御する部分である。

このうち、インタフェース層は大概のデバイスドライバで共通化できることが多い。デバイスドライバ作成者の負担を少しでも減らし、余計なオブジェクトコードの増加を防ぐために、共通化可能な部分をまとめたのがデバイスドライバ I/F 層（ドライバ I/F）である。



（T-Kernel 仕様書より引用）

2.2 デバイスドライバ I/F 層の種類

現在、ドライバ I/F は 2 種類用意されている。

- 単純デバイスドライバ I/F 層（単純ドライバ I/F）
  すべての処理を待ちに入ることなく即座に処理できる、ごく単純なデバイス向け（例: RTC 等のレジスタベースの物）

- 汎用デバイスドライバ I/F 層（汎用ドライバ I/F）
  要求順に処理する一般的なデバイス全般で使用し、特に処理を中断する必要があるデバイス向け（例: RS-232C 等）

大きな違いとして、汎用ドライバ I/F は入出力に対応する操作を実行するためのタスクを作成する必要があるのに対し、単純ドライバ I/F は入出力に対応する操作は全て関数として記述するだけで済むことが挙げられる。
また、汎用ドライバ I/F を使用した場合はデバイスに対する入出力操作を中断するための関数を定義することができるが、単純ドライバ I/F では入出力操作の中断は行えない（不定期の待ちに入ることはできないため、当然と言える）という点も異なっている。

これらのドライバ I/F 層は、あくまでデバイスドライバの作成を容易にするためのライブラリであり、必ず使用しなければいけないというものではない。デバイスの性質や性能上などの点から、上記のモデルでは対応できないような場合は、個別に最適な実装を行うことで構わない。

2.3 各種デバイスドライバ I/F 層に共通する注意事項

ドライバ I/F に定義されている関数を使用する上での注意事項は、以下の通りである。

- 特に明記されたものを除き、タスク独立部およびディスパッチ禁止中・割込禁止中にドライバ I/F の関数を呼び出してはならない。

- ドライバ I/F へ登録する各種処理関数は処理を速やかに行い、決して不定期な待ちに入ってはならない。

- ドライバ I/F へ登録する各種処理関数は要求タスクのコンテキストとなり、準タスク部として実行される。よって、

  - タスク優先度等を変更した場合、処理関数から戻る前に元に戻す
  - 処理関数が動作しているタスクのタスク番号を常に意識する
  - スタックの消費量を意識する

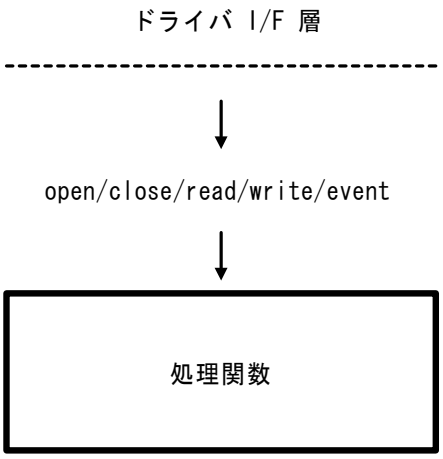
  という点に注意する必要がある。しかし、各種処理関数の呼び出しは排他制御が行われるため、これらの関数が同時にされることはない（汎用ドライバ I/F の abort 処理関数に関してはこの限りでない）。

- ドライバ I/F は T-Kernel のデバイス管理機能の下で動作するため、各種構造体なども T-Kernel のデバイス管理機能の定義を使用する。

本章では、T-Kernel 側で定義されている構造体や値に関しては（T-Kernel）とだけ記述することにし、詳細に関しては記述しない。必要に応じ、 T-Kernel 仕様書のデバイス管理機能の項を参照すること。

2.4 単純デバイスドライバ I/F 層 <driver/sdrvif.h>

不定期の待ちに入ることの無い、ごく単純なデバイス向けのデバイスドライバを作成する際に使用する。
以下の図のように、デバイスドライバは要求を処理する関数群として実装される。



2.4.1 デバイスの登録

```
ER SDefDevice( SDefDev *ddev, T_IDEV *idev, SDI *sdi )
```

ddev:　デバイスドライバ登録情報へのポインタ
idev:　デバイス初期情報（T-Kernel）を格納する領域へのポインタ
sdi:　ドライバ I/F アクセスハンドルを格納する領域へのポインタ

戻り値: = 0　　正常終了
　　< 0　　エラー

ddev の登録情報にしたがって、デバイスを登録する。idev にデバイス初期情報が返されるが、idev = NULL とした場合は返されない。同時に、単純ドライバ I/F の操作に必要なドライバ I/F アクセスハンドルを sdi に返す。

以下に、 SDI および SDefDev 構造体を示す。

```
typedef struct SimpleDriverInterface *  SDI;

typedef struct {
        VP      exinf;                  /* 拡張情報 */
        UB      devnm[L_DEVNM+1];       /* 物理デバイス名(T-Kernel) */
        ATR     drvatr;                 /* ドライバ属性(T-Kernel) */
```

```
        ATR     devatr;                 /* デバイス属性(T-Kernel) */
        INT     nsub;                   /* サブユニット数(T-Kernel) */
        INT     blksz;                  /* ブロックサイズ(T-Kernel) */

        ER  (*open )( ID devid, UINT omode, SDI );
        ER  (*close)( ID devid, UINT option, SDI );
        INT (*read )( ID devid, INT start, INT size, VP buf, SDI );
        INT (*write)( ID devid, INT start, INT size, VP buf, SDI );
        INT (*event)( INT evttyp, VP evtinf, SDI );
} SDefDev;
```

exinf には任意の値が設定でき、SDI_exinf() （後述）によって参照できる。

devnm, drvatr, devatr, nsub, blksz に指定する値に関しては T-Kernel 仕様書のデバイス管理機能の項を参照。

drvatr には以下の値が指定可能である。(T-Kernel)

```
#define TDA_OPENREQ     0x0001  /* 毎回オープン/クローズ */
```

open, close, read, write, event は、それぞれ T-Kernel 側の以下の呼び出しに対応する、デバイスドライバ側の処理関数群である。

デバイスドライバ側	T-Kernel 側
open に定義した関数	tk_opn_dev()
close に定義した関数	tk_cls_dev()
read に定義した関数	tk_rea_dev()
write に定義した関数	tk_wri_dev()
event に定義した関数	サスペンド・レジューム時や USB マネージャ・PC カードマネージャからのイベントが発生した際に実行される。

read/write で定義した入出力処理関数の戻り値は、入出力した結果のサイズまたはエラーとする。また、read/write に渡される、buf で指定された領域はドライバ I/F によって領域の検査(ChkSpace)が済んでいる。

2.4.2 デバイスの更新

```
ER SRedefDevice( SDefDev *ddev, SDI sdi )
```

ddev:　デバイスドライバ登録情報へのポインタ
sdi:　ドライバ I/F アクセスハンドル

戻り値: = 0　正常終了
< 0　エラー

ddev の登録情報にしたがって、SDI のデバイス登録情報を更新する。ただし、デバイス名 (devnm) を変更してはならない。
更新の場合、物理デバイス ID は変更されない。

2.4.3 デバイスの抹消

```
ER SDelDevice( SDI sdi )
```

sdi:　ドライバ I/F アクセスハンドル

戻り値: = 0　正常終了
< 0　エラー

sdi に指定したドライバ I/F アクセスハンドルを持つ、デバイスドライバの登録を抹消する。

2.4.4 各種情報の取得

```
ID SDI_devid( SDI sdi )
VP SDI_exinf( SDI sdi )
const SDefDev* SDI_ddev( SDI sdi )
```

sdi:　ドライバ I/F アクセスハンドル

戻り値: SDI_devid():　物理デバイス ID (T-Kernel)
SDI_exinf():　現在登録されているデバイスドライバ登録情報のうち、exinf に登録されている値
SDI_ddev():　現在登録されているデバイスドライバ登録情報へのポインタ

各種情報を取得する。

・SDI_devid() は、USB マネージャや PC カードマネージャ等にイベント処理関数を実行させる場合に必要な、物理デバイス ID を取得する場合に使用する。

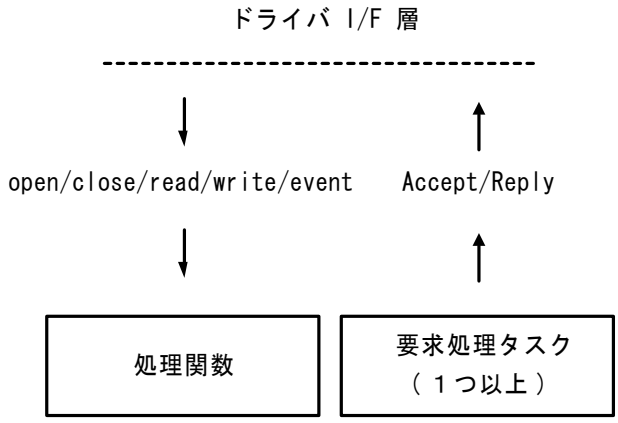
・SDI_exinf() は、SDefDevice() や SRedefDevice() 等で登録したデバイスドライバ登録情報のうち、exinf の値のみを取り出す。
・SDI_exinf() は、SDefDevice() や SRedefDevice() 等で登録したデバイスドライバ登録へのポインタを取り出す。

これらの関数は、タスク独立部およびディスパッチ禁止中・割込禁止中も呼び出すことができる。

2.5 汎用デバイスドライバ I/F 層 <driver/gdrvif.h>

要求順に処理する必要のある、一般的なデバイスを対象としたデバイスドライバを作成する際に使用する。

デバイスドライバは一つ以上の要求処理タスクを持ち、非同期に要求を処理する。



2.5.1 デバイスの登録

ER GDefDevice(GDefDev *ddev, T_IDEV *idev, GDI *gdi)
ddev: デバイスドライバ登録情報へのポインタ
idev: デバイス初期情報（T-Kernel）を格納する領域へのポインタ
gdi: ドライバ I/F アクセスハンドルを格納する領域へのポインタ

戻り値: = 0 正常終了
< 0 エラー

ddev の登録情報にしたがって、デバイスを登録する。idev にデバイス初期情報が返されるが、idev = NULL とした場合は返されない。同時に、汎用ドライバ I/F の操作に必要なドライバ I/F アクセスハンドルを gdi に返す。

以下に、GDI および GDefDev 構造体を示す。

```
typedef struct GeneralDriverInterface * GDI;

typedef struct {
        VP      exinf;                  /* 拡張情報 */
        UB      devnm[L_DEVNM+1];       /* 物理デバイス名(T-Kernel) */
        UH      maxreqq;                /* 要求キューイング数 */
        ATR     drvatr;                 /* ドライバ属性(T-Kernel) */
        ATR     devatr;                 /* デバイス属性(T-Kernel) */
        INT     nsub;                   /* サブユニット数(T-Kernel) */
        INT     blksz;                  /* ブロックサイズ(T-Kernel) */

        ER  (*open )( ID devid, UINT omode, GDI );
        ER  (*close)( ID devid, UINT option, GDI );
        ER  (*abort)( T_DEVREQ *devreq, GDI );
        INT (*event)( INT evttyp, VP evtinf, GDI );
} GDefDev;
```

exinf には任意の値が設定でき、GDI_exinf() （後述）によって参照できる。

maxreqq はキューイングする要求の最大数を指定する。1 以上の値を指定し、それ未満の値は指定しないこと。

devnm, drvatr, devatr, nsub, blksz に指定する値に関しては T-Kernel 仕様書のデバイス管理機能の項を参照。

drvatr には以下の値が指定可能である。(T-Kernel)

```
#define TDA_OPENREQ     0x0001  /* 毎回オープン/クローズ */
#define TDA_LOCKREQ     0x8000  /* アドレス空間のロックが必要*/
#define TDA_LIMITEDREQ  0x4000  /* 要求種別毎にキューイング数を制限*/
```

※TDA_LOCKREQ が指定された場合、入出力バッファ(T_DEVREQ.buf)の
領域はドライバ I/F によってロック(常駐化)される。

※TDA_LIMITEDREQ が指定された場合、要求種別(TDC_READ、TDC_WRITE)ごとに
要求のキューイング数を最大キューイング数(maxreqq)の約半数に制限する。
この場合、maxreqq には 2 以上を指定する必要がある。
TDA_LIMITEDREQ が指定されていなければ、要求の種別に関係なくキューが一
杯になるまでキューイングされる。
TDA_LIMITEDREQ は、読み込みと書き出しを非同期に並行して処理するような

デバイスにおいて、一方の要求でキューが満杯になり他方の要求が受け付けられなくなることを防ぎたいような特殊な場合に使用する。

open, close, abort, event は、それぞれ T-Kernel 側の以下の呼び出しに対応する、デバイスドライバ側の処理関数群である。

デバイスドライバ側	T-Kernel 側
open に定義した関数	tk_opn_dev()
close に定義した関数	tk_cls_dev()
abort に定義した関数	tk_cls_dev()他、デバイスへの入出力操作を中断する必要があった時に実行される
event に定義した関数	サスペンド・レジューム時や USB マネージャ・PC カードマネージャからのイベントが発生した際に実行される

abort 処理関数に関しては、他の要求処理関数に比べて以下の違いがある。

- 他の処理関数の実行中であっても呼び出される場合がある（ただし、abort 関数自体が複数同時に呼び出されることはない）。
- 使用できる汎用ドライバ I/F の関数に制限がある（使用できるのは、GDI_devid(), GDI_exinf(), GDI_ddev(), GDI_SendCmd() のみ）。

abort 処理関数が中断させる、現在入出力中の要求は devreq で示される。

2.5.2 デバイスの更新

ER GRedefDevice(GDefDev *ddev, GDI gdi)

ddev: デバイスドライバ登録情報へのポインタ
gdi: ドライバ I/F アクセスハンドル

戻り値: = 0 正常終了
< 0 エラー

ddev の登録情報にしたがって、GDI のデバイス登録情報を更新する。ただし、デバイス名 (devnm) および要求キューイング最大数 (maxreqq) を変更してはならない。

※更新の場合、物理デバイス ID は変更されない。
※要求受け付けキューに溜まっている、まだ受け付けられていない要求は全てアボートされる。

2.5.3 デバイスの抹消

```
ER GDelDevice( GDI gdi )
```

gdi:　ドライバ I/F アクセスハンドル

戻り値: = 0　正常終了
< 0　エラー

gdi に指定したドライバ I/F アクセスハンドルを持つ、デバイスドライバの登録を抹消する。

2.5.4 入出力要求の受け付け

```
INT GDI_Accept( T_DEVREQ **devreq, INT acpptn, TMO tmout, GDI gdi );
```

devreq: デバイスドライバ入出力要求パケットへのポインタを格納する領域を示すポインタ
acpptn: 受け付ける要求のタイプ
tmout:　タイムアウト時間 (ms)
gdi:　ドライバ I/F アクセスハンドル

戻り値: = 0　受け付けた要求のパターン（要求のタイプにより変化する）
< 0　エラー

要求受け付けキューから要求を１つ取り出す。キューに要求がなければ、要求が来るまで待ちに入る。

acpptn には、受け付ける要求のタイプ(TDC_READ/TDC_WRITE)またはユーザーコマンド（後述）のパターンで、DEVREQ_ACPPTN() で得た値を OR で指定する。

```
/* cmd = TDC_READ || TDC_WRITE || ユーザーコマンド(16～23) */
#define DEVREQ_ACPPTN(cmd)      ( 1 << (cmd) )

#define DRP_READ        DEVREQ_ACPPTN(TDC_READ)
#define DRP_WRITE       DEVREQ_ACPPTN(TDC_WRITE)
#define DRP_NORMREQ     ( DRP_READ|DRP_WRITE )  /* 通常要求 */
#define DRP_USERCMD     0x00ff0000              /* ユーザーコマンド */
```

また固有データに対する要求と属性データに対する要求を個別に受け付けること(受付待ち拡張機能)もできる。

```
#define DRP_ADSEL       0x00000100   /* 固有・属性データ個別指定 */
```

```
#define DRP_DREAD       (DRP_ADSEL | DEVREQ_ACPPTN(TDC_READ) )
#define DRP_DWRITE      (DRP_ADSEL | DEVREQ_ACPPTN(TDC_WRITE) )
#define DRP_AREAD       (DRP_ADSEL | DEVREQ_ACPPTN(TDC_READ  + 8) )
#define DRP_AWRITE      (DRP_ADSEL | DEVREQ_ACPPTN(TDC_WRITE + 8) )

#define DRP_REQMASK     (DRP_ADSEL | DRP_NORMAREQ | (DRP_NORMREQ << 8))
```

DRP_DREAD　　固有データ読み込み
DRP_DWRITE　固有データ書き出し
DRP_AREAD　　属性データ読み込み
DRP_AWRITE　属性データ書き出し
これらを OR で組み合わせて指定する。

※ DRP_DREAD |DRP_AREAD は DRP_READ と等価である。
※ DRP_DWRITE|DRP_AWRITE は DRP_WRITE と等価である。
※ これら固有・属性データ個別指定と、DRP_READ、DRP_WRITE を同時に組み合わせて使用することはできない。

要求の受け付けは、通常要求(TDC_READ/TDC_WRITE)が、ユーザーコマンドより優先されるが、通常要求とユーザーコマンドを同時に受け付ける場合もある。
戻り値や *devreq の内容は、受け付けた要求によって以下のようになる。

・通常要求の場合
　受け付けた要求のタイプを示すパターンを戻り値に返す。
　*devreq に受け付けた要求を返す。
・ユーザーコマンドの場合
　受け付けたユーザーコマンドを示すパターンを戻り値に返す。
　複数種類のユーザーコマンドが溜まっていた場合には、acpptn で指定したすべてのユーザーコマンドを一括して受け付け、それらを OR したパターンが返される。
　*devreq には NULL を返す。
・通常要求とユーザーコマンドを同時に受け付けた場合
　受け付けた通常要求とユーザーコマンドの両方を OR したパターンを戻り値に返す。
　*devreq には受け付けた通常要求を返す。
・タイムアウトまたはエラーの場合
　戻り値にはエラーコードを返す。タイムアウトの場合 E_TMOUT となる。
　*devreq は不定となる。

※ 受け付けた要求のパターンは、acpptn に指定した形式で戻り値に返す。つまり、受け付け待ち拡張機能を使用した場合には、受け付けた要求のパターンも固有データに

対する要求と属性データに対する要求が個別に示される。この場合 DRP_ADSEL も設定される。

tmout には、タイムアウト時間をミリ秒単位で指定する。TMO_POL、TMO_FEVR も指定できる。

受け取った要求(T_DEVREQ)の exinf を変更してはいけない。

buf の領域の検査(ChkSpace)は、ドライバ I/F で行ってあるが、タスク空間は必要に応じ tk_set_tsp(devreq->tskspc) 等を行って切り替える必要がある。

ユーザーコマンドに対する応答(GDI_Reply)は不要である。

一般的には１つの要求を受け付け、処理し、結果を返した後に、次の要求を受け付ける。しかし、複数の要求を受け付け、同時に処理してもよい。
複数の要求を同時に処理する場合、複数の要求処理タスクがそれぞれ GDI_Accept() を行い並行して処理してもよいし、１つの処理タスクが複数回 GDI_Accept() を行い同時に処理してもよい。
また、要求を受け付けた順序と、結果を返す順序は同じでなくても構わない。

2.5.5 入出力要求完了の応答

```
void GDI_Reply( T_DEVREQ*, GDI gdi );
```

devreq: 入出力処理が完了したデバイスドライバ入出力要求パケットを示すポインタ
gdi: ドライバ I/F アクセスハンドル

戻り値: 無し

GDI_Accept() で受け取った要求の処理結果を返す。
GDI_Accept() を行ったタスクと GDI_Reply() を行うタスクは、同一でなくてもよい。

2.5.6 入出力要求タスクに対するユーザーコマンドの発行

```
ER GDI_SendCmd( INT cmd, GDI gdi )
```

gdi: ドライバ I/F アクセスハンドル
cmd: ユーザーコマンド（16 ～ 23 の範囲の値で指定すること）

戻り値: = 0 正常終了
< 0 エラー

cmd で指定したユーザーコマンドを発行する。

発行したユーザーコマンドは、GDI_Accept() によって受け付けられる。
GDI_Accept() によって受け付けられていないユーザーコマンドが溜まっていた場合、同一のコマンドが複数個溜まることはない。同一のコマンドを何回発行しても１回分しか溜まることはなく、１回の GDI_Accept()で溜まったコマンドは消失する。

なお、GDI_SendCmd() はユーザーコマンドを発行するのみで、GDI_Accept()によって受け付けられるのを待つことなく戻る。

ユーザーコマンドは、一般的には GDI_Accept() の待ちを任意の時点で解除する目的に使用するためのものである。

2.5.7 各種情報の取得

```
ID GDI_devid( GDI gdi )
VP GDI_exinf( GDI gdi )
const GDefDev* GDI_ddev( GDI gdi )
```

gdi: ドライバ I/F アクセスハンドル

戻り値: GDI_devid(): 物理デバイス ID (T-Kernel)
GDI_exinf(): 現在登録されているデバイスドライバ登録情報のうち、exinf に登録されている値
GDI_ddev(): 現在登録されているデバイスドライバ登録情報へのポインタ

各種情報を取得する。

- GDI_devid() は、USB マネージャや PC カードマネージャ等にイベント処理関数を実行させる場合に必要な、物理デバイス ID を取得する場合に使用する。
- GDI_exinf() は、GDefDevice() や GRedefDevice() 等で登録したデバイスドライバ登録情報のうち、exinf の値のみを取り出す。
- GDI_exinf() は、GDefDevice() や GRedefDevice() 等で登録したデバイスドライバ登録へのポインタを取り出す。

これらの関数は、タスク独立部およびディスパッチ禁止中・割込禁止中も呼び出すことができる。