リアルタイム APIトレーニングテキスト

アドバンスドコース　DDC1553B

2009/12/01 第3版
コンカレント日本株式会社
プロフェッショナルサービス部

BU-65572i/72v とBU-65570M でのソフトウェアは、実行時ライブラリーを使ってください。ユーザは直接レジスタやメモリをアクセスしてはなりません。

ユーザは、実行時ライブラリーを使って、1553のバス操作のすべてのアスペクトをコントロールすることができます。 それぞれのプログラムはカードをリセットして、メッセージとフレームを作って、そして次に実行させる基本的なスケルトン構成に従います。

あなたは続くサンプルコードでこの形式を見るでしょう。 下のプログラム例は RT と BC の両方をセットアップします。

#include <ts_drv.h>

ts_drv.h ファイルはプログラムに必ず含められなくてはなりません。

アプリケーションが必要とするファンクションの#defines ストラクチャとプロトタイプ宣言は、このヘッダファイルがすべてを供給します。同様に、このヘッダーファイルは、“C”のファンクショナリティに必要なすべてのスタンダードヘッダーファイルを含みます。

このヘッダファイルへのパスはコンパイラにセットされなくてはなりません。

RedHawk では、/usr/include に存在します。同様にプログラムを構築するとき、 -ltstsim がメイクファイルに含められなくてはなりません。

以下に、典型的な Makefile を示します。

```
CC=gcc
INCLUDEDIR=/usr/include

CFLAGS=-Wall -I$(INCLUDEDIR) -g -ltstsim -lpthread

all:        sample
sample:     sample.c
            $(CC) $(CFLAGS) sample.c -o sample
```

下に定義される変数は BC 、 RT と Monitor のオペレーションプログラムによって使われます。実行時ライブラリーによって使われる変数の多くは、ヘッダーファイルで定義されています。

```
MESSAGE Message;
Device_p pCrd;
CMD Cmd1, Cmd2;
INJ_ERR InjErr;
U16BIT Frame[2];
RT_DEFS rt;
DRV_CONFIG cfg;
U16BIT err;
U16BIT data[32],i,j;
S16BIT MTFilter[32][32];

int main ()
{
        printf("Resetting card...¥n");
        if(err = ddcResetCard(&pCrd,&cfg,0))
        {
                ddcPrintErrorMessage(pCrd,err,"reset_card");
                exit (1);
        }
        printf("Card reset successful!¥n");
```

「ddcResetCard」ファンクションはプログラムによって呼び出される最初のRTLファンクションです。このファンクションはハードウェアをリセットして、基本的な環境設定をダウンロードして、そしてハードウェアの自己診断を行ないます。 このファンクションは、 Error_t の戻り値を持っています。この戻り値はプログラムの命令が正しいことを保証するためにすべての関数呼び出しがないかチェックしなくてはなりません。 もしエラー状態が存在するなら、ファンクションからの戻り値は戻り値とストリングを印刷するために ddcPrintErrorMessage ルーチンに引き渡してください。

CMD ストラクチャは、メッセージのためにコマンドを作成するために正しい文字で記入されなくてはなりません。 すべてのメッセージがコマンドを必要とします。

下の例は、リモートターミナル 2 (.tadr) サブアドレス 1 (.subadr).からリモートターミナルアドレス 1 (.tadr) サブアドレス 1 (.subadr)へ 5 ワード(.wcnt)を送信(.t_r= 0)するコマンドを作成します。BC→RT 、RT→BC とモードコードメッセージは単一のコマンド定義で、RT→RT のみ複数のコマンドの定義が必要です。

この例の場合では、 cmd1 と cmd2 は RT-RT メッセージなので2つのコマンド定義が必要です。

下の例は RT-RT メッセージの例です。

```
Cmd1.wcnt = 5;      /* RT が 5 ワード受信 */
Cmd1.subadr = 1;    /* RT サブアドレスは 1 */
Cmd1.t_r = 0;       /* RT が受信(0) 5 ワード */
Cmd1.tadr = 1;      /* RT アドレスは 1 */

Cmd2.wcnt = 5;      /* RT が 5 ワード送信 */
Cmd2.subadr = 1;    /* RT サブアドレスは 1*/
Cmd2.t_r = 1;       /* RT が送信(1) */
Cmd2.tadr = 2;      /* RT アドレスは 2 */
```

それぞれのメッセージは、エラーを検出するための機能を持っています。

これらのエラーはパルス幅タイミングのようなワードカウントとエンコーディングエラーを含みます。

もしエラータイプが E_NONE にセットされるなら、他のエントリは無視されます。

```
InjErr.error = E_NONE;
InjErr.sub_error_1 = 0;
InjErr.sub_error_2 = 0;
InjErr.sub_error_3 = 0
```

メッセージは、メッセージストラクチャを定義する事によって作られます。 メッセージストラクチャは定義されたコマンド(Cmd1 と Cmd2)を結合して、そして、アクティブなバス、タイミング、データテーブルとともに、エラー(InjErr)を挿入したりします。 このストラクチャはメッセージを送るために必要な情報のすべてを含んでいます。

```
Message.bus = BUS_A;
Message.comm_type = RT_RT;
Message.cmd_1 = Cmd1;
```

```
Message.cmd_2 = Cmd2;
Message.time_to_next_message = 1000;
Message.data_table_no = 1;
Message.first_intermessage_routine = NO_OPERATION;
Message.second_intermessage_routine = NO_OPERATION;
Message.inj_error_ptr = &InjErr;
```

このメッセージストラクチャは、ddcDef_message ルーチンに呼び出しで、登録されます。

```
ddcDef_message (pCrd, 1, &Message );
```

メッセージが定義された後、それらは U16BIT フレームアレイに入力されます。 このアレイは1つのフレーム時間に処理されるであろうメッセージのすべてのためにインデックスを持つでしょう。

他のフレームシンボルが同じくマイナーなフレームとフレームを定義するために使われます。それぞれのメッセージが ddcDef_message 呼び出しにおいて割り当てられた番号にのフレームアレイの中に定義されます。 この番号は、実際に送信されるフレームにおける位置とは無関係です。 関数は、END_OF_MAJOR_FRAME や END_OF_MINOR_FRAME 等の実行されるフレームシンボルを含めて、メッセージの数を確立します。

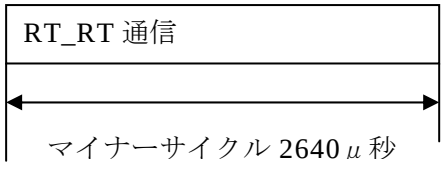
```
Frame[0] = 1;
Frame[1] = END_OF_MAJOR_FRAME;
ddcDef_frame (pCrd, 2/*個*/, Frame );
```

2個

フレームが1度、X回、あるいは永久に、繰り返されるかもしれません。 毎回フレーム点始めから終わりへと、それは指定された時間で実行されます。

それが フレーム完了に使う時間の量は、マイナーフレーム時間にマイナーフレームの総数を掛けた時間です。 この例では、マイナーフレームは、 ddcDefMinorFrameTime ファンクションで2640μ秒に設定され、ただ1つ存在します。

```
ddcDefMinorFrameTime (pCrd, 2640 );
```



同様に RT コマンド用のストラクチャが存在します。

この場合の RT コマンドは、エラー検出なし、リターンステータス 0x0000 と単純です

```
InjErr.error = E_NONE;
InjErr.sub_error_1 = 0;
InjErr.sub_error_2 = 0;
InjErr.sub_error_3 = 0;
rt.inj_error = &InjErr;
rt.basic_status = 0x0000;
```

BU-65572i/72v とBU-65570M はバスで1、あるいは31の RT のすべてとして機能することができます。 ファンクション ddcDef_emulate_rt() は、どの RT がソフトウェアによってエミュレートされるかを定義します。 RT が ddcDef_emulate_rt()で定義された後、 構造体 rt をパラメータとする ddcDef_rt()関数で rt の応答コマンドを定義します。
そして、 RT 用のデータテーブル ddcDef_rt_map にマップすることができます。

```
ddcDef_emulate_rt (pCrd, 1, YES);        //RT1 をエミュレートする
ddcDef_rt(pCrd,1, &rt);                  //RT1 の応答コマンドを定義
ddcDef_rt_map(pCrd,1,RECEIVE,1,1);       //RT1 の受信応答としてテーブル1を使用する
```

この例では RT - RT 通信をしていますから、 RT2を設定してください。（そのために2番目のパラメータが2になる） ddcDef_rt_map を RT 2で呼び出す部分を除いて、この設定は RT 1と同じです、そして3番目のパラメータは送信です。

```
InjErr.error = E_NONE;
InjErr.sub_error_1 = 0;
InjErr.sub_error_2 = 0;
InjErr.sub_error_3 = 0;
rt.inj_error = &InjErr;
rt.basic_status = 0x0000;
ddcDef_emulate_rt (pCrd, 2, YES);        //RT2 をエミュレートする
ddcDef_rt(pCrd,2, &rt);                  //RT2 の応答コマンドを定義
ddcDef_rt_map(pCrd,2,TRANSMIT,1,1);      //RT2 の送信応答としてテーブル2を使用する
```

次にはデータテーブルを RT 1と RT 2にセットアップしなければなりません。関数 ddcDef_table_size() は、指定されたデータテーブル（2番目の仮引数 － 1）の、大きさ（3番目の仮引数 － 5）を設定します。 そして ddcWrite_data() を呼び出し、データテーブル1にデータを実際に書き込みます。

```
ddcDef_table_size(pCrd,1,5);
data[0]=0x0000; data[1]=0x0001; data[2]=0x0002;
data[3]=0x0003; data[4]=0x0004;
ddcWrite_data(pCrd,1,data,5,1);
```

セットアップデータテーブル2。

```
ddcDef_table_size(pCrd,2,5);
data[0]=0x0000; data[1]=0x0001; data[2]=0x0002;
data[3]=0x0003; data[4]=0x0004;
ddcWrite_data(pCrd,2,data,5,1);
```

次の設定は、モニターです。

最初の初期化は、MTFilter 配列に すべて3をセットします。

このファンクションはセットアップにスタックファイルを使用します。 1553B カードからホストに配信される割り込みのマスクビットをセットし、 応答するビットにフラグを付けます

RT はモニターがデータを取り込むときセットされます。 同様にバスが選択されます。

```
for (i=0; i<32; i++)
        for (j=0; j<32; j++)
                MTFilter[i][j]= 3;
ddcDef_monitor_stack(pCrd, CYCLIC_STACK);
ddcDef_int_mask(pCrd, 0xffff);
ddcDef_mon_exception_status(pCrd, 0x07ff);
ddcCapture_event(pCrd,CAPTURE_IMMEDIATE,0x0000,0x0000);
ddcSelect_bus(pCrd,YES,NO);
for(i=0; i<32; i++)
{
        for(j=0; j<32; j++)
        {
            if(MTFilter[i][j] & 0x1)
                ddcSelect_message(pCrd,i,TRANSMIT,j);
            else
                ddcDeselect_message(pCrd,i,TRANSMIT,j);
             if (MTFilter[i][j] & 0x2)
                ddcSelect_message(pCrd,i,RECEIVE,j);
             else
                ddcDeselect_message(pCrd,i,RECEIVE,j)
        }
}
ddcRun_mon(pCrd);
```

RT と BC を使用するために次のルーチンを走らせてください。

BC は1つのフレームのメッセージを100回走らせるでしょう。

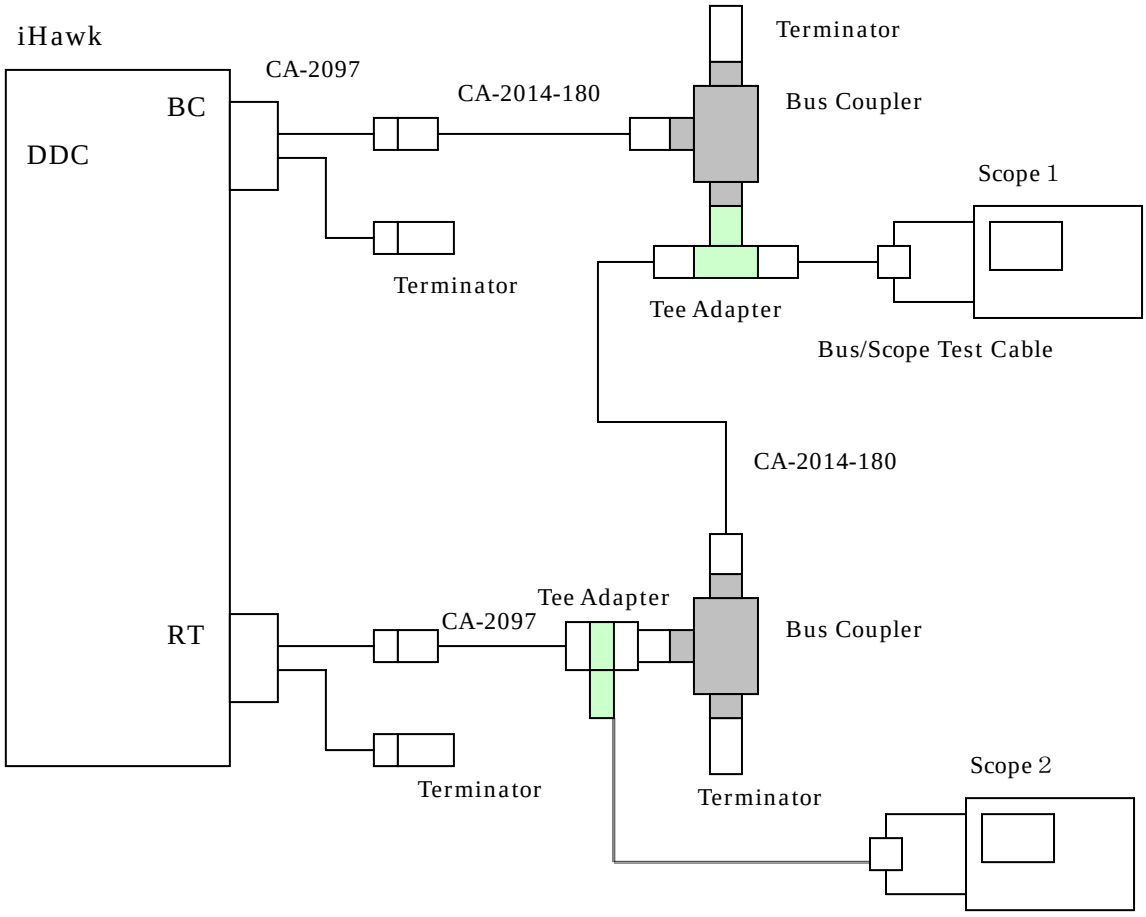
```
ddcRun_rt(pCrd);
ddcRun_bc (pCrd, 1, 100);
```

プログラムが完了したとき、プログラムを止めてください。

```
printf("¥nPress any key to Halt the Card¥n");
getch();
ddcHaltIdea(pCrd);
ddcShutDownIdea(&pCrd);
return 0;
}
```

アンプの設定とカップリング

```
//デバイスリセット
if( (nErr = ddcResetCard(&pDev, &stConfig, uhLogDev)) )
{
        ddcPrintErrorMessage(pDev, nErr, "ddcResetCard");
        return -1;
}
// カップリング　トランス有り、内蔵ターミネータオフ
if( (nErr= ddcSetCoupling(pDev, COUPLING_TRANSFORMER,TERMINATION_NONE)) )
{
        ddcPrintErrorMessage(pDev, nErr, "SetCoupling");
        exit(1);
 }
//アンプの設定
ddcSetAmp(pDev,0xFFF);
```



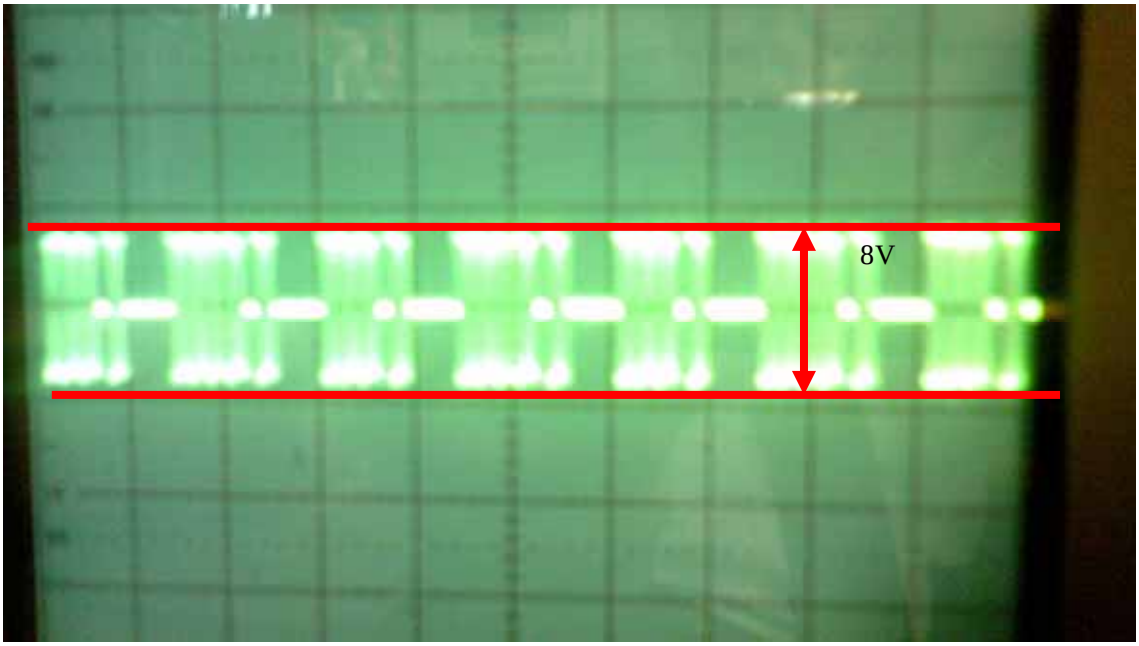
ディフォルトのアンプゲイン値　0x320.(ddcReadCoupling()を使用)

0x320=　800/4096 * 21　= 4.1V

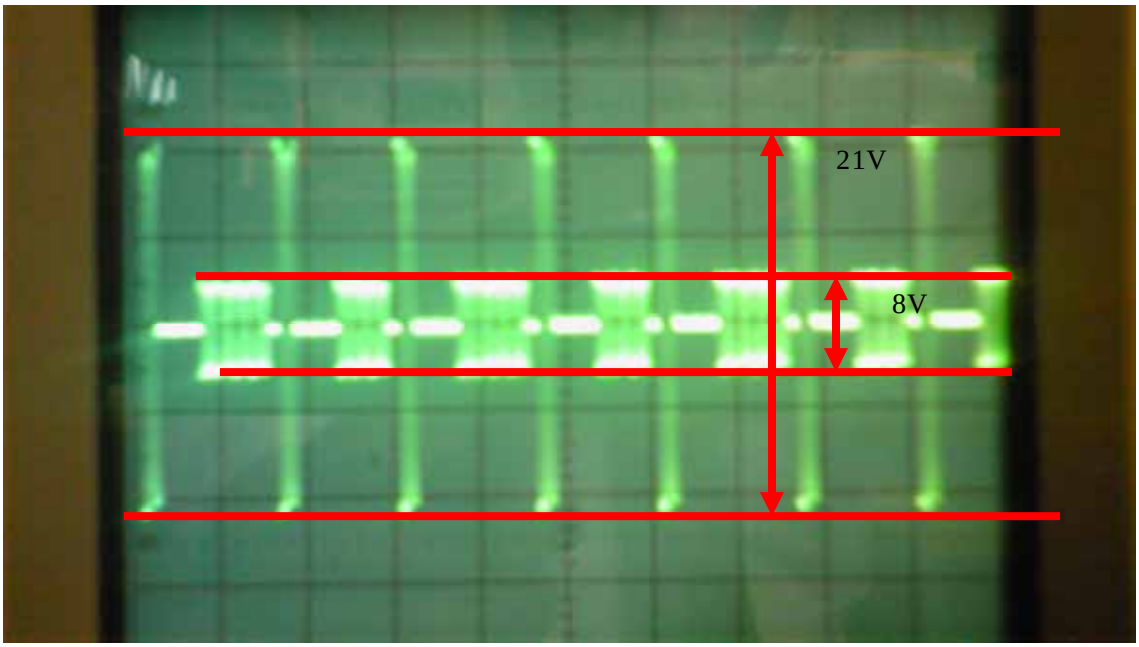
この電圧は、Scope2　側

オシロスコープ２側が２１Ｖ 時、メインバス側（オシロスコープ１側）の電圧は８Ｖになります。
下記波形は、BC,RT 側とも：ddcSetAmp(devp,0xFFF);のものです。

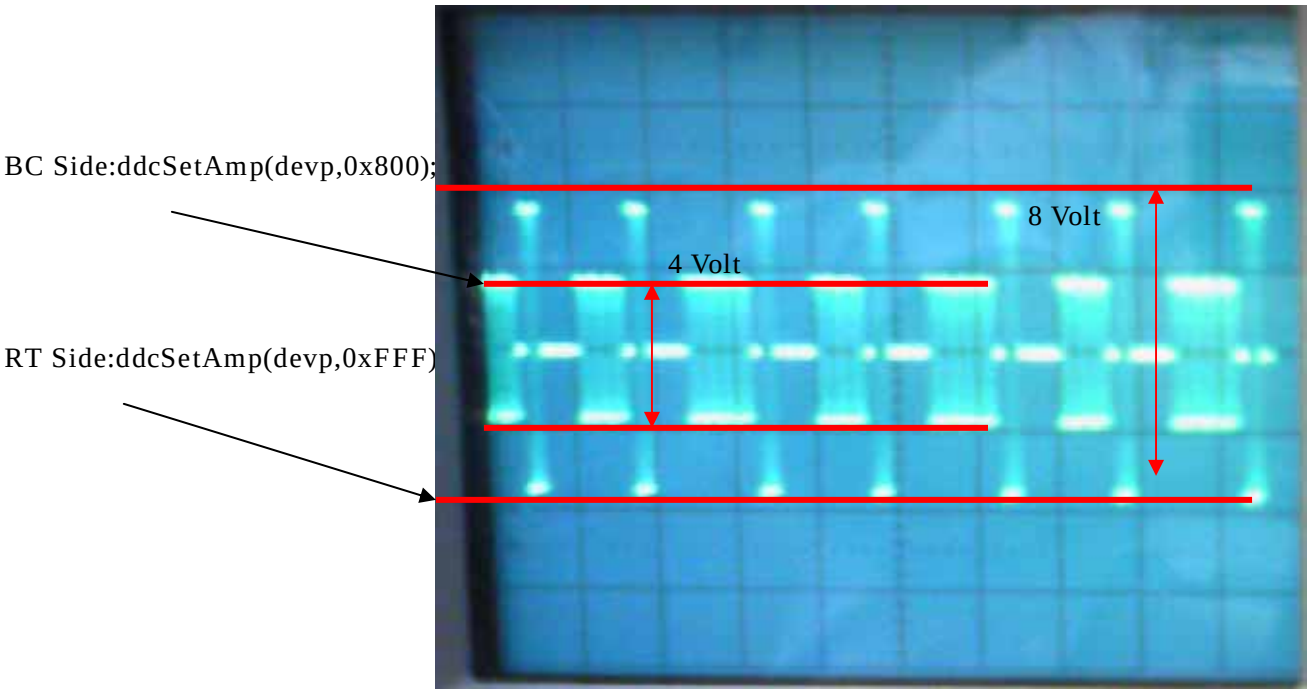
オシロスコープ１の波形



オシロスコープ２の波形



本線側の電圧は、transformer coupled 時、ピークで、1.0V から 14.0V であれば規格範囲です。



本線側の電圧ですが、BC 側のアンプレンジを固定し、RT 側のアンプレンジを小さくしていくと、値が 0x29D（測定時 1.0V）が通信限界で、これ以下になると通信エラーが発生し始めます。
また、A バス以外に B バスをイネーブルにすると、レベルが下がるため、ゲインは最大を選択することが良いようです。

RT 送受信をダブルバッファにする。

非同期にアプリケーションが1553のバスアクセスデータテーブルをアクセスする時、競合を避けるためダブルバッファリングを使用します。もしデータテーブルがダブルバッファに入れられるなら、それがデータを読む前に、そしてそれがデータを書いた後、ホストはテーブルポインタを交換します。 ルーチン *ddcDefDataBuffering()*はサイズ ＝ 0をデータテーブル1から250にそしてサイズ ＝ 32をテーブル0と251に割り当てます。 さらに、データテーブルポインタに適切な値を割り当てることによって、メモリを割り当てます。 結果として、もしユーザがデータテーブルの大きさを明確にしたいなら、*ddcDefTableSize()*ルーチンで再定義されなくてはなりません。

注意：フレームはサイズ ＝ 0を割り当てられるので、再定義されなくてはなりません。

```
TA = RT アドレス(0 から 31)
SA = RT サブアドレス(1から 31)
TR = 0::受信　1:送信
WC =  ワードカウント(1 から 32)
TD =  テーブル ID(1  から 1023)
InjErr.error            = E_NONE;
InjErr.sub_error_1 = 0;
InjErr.sub_error_2 = 0;
InjErr.sub_error_3 = 0;
RT.inj_error       = &InjErr;
RT.basic_status = 0x0000;
RT.dbca                 = NO;
if( (Err = ddcDef_data_buffering(Dev, DOUBLE)) ) {
            ddcPrintErrorMessage(Dev, Err, "ddcDef_data_buffering");return -1;
}
if( (Err = ddcDef_emulate_rt(Dev, TA, YES)) )       {
            ddcPrintErrorMessage(Dev, Err, "ddcDef_emulate_rt");return -1;
}
if( (Err = ddcDef_rt(Dev, TA, &RT)) )     {
            ddcPrintErrorMessage(Dev, Err, "ddcDef_rt");return -1;
}
if( (Err = ddcDef_rt_map(Dev, TA, TR, SA, TD)) )   {
            ddcPrintErrorMessage(Dev, Err, "ddcDef_rt_map");return -1;
}
if( (Err = ddcDef_table_size(Dev, TD, WC)) )          {
            ddcPrintErrorMessage(Dev, Err, "ddcDef_table_size");return -1;
}
if(TR == 受信){
      if( (Err = ddcDef_table_routine(Dev, TD, RT_INT_AFTER_DATA, NO_OPERATION)) )
      {
            ddcPrintErrorMessage(Dev, Err, "ddcDef_table_routine");return -1;
      }
}else{
      if( (Err = ddcDef_table_routine(Dev, TD, NO_OPERATION, NO_OPERATION)) )
      {
            ddcPrintErrorMessage(Dev, Err, "ddcDef_table_routine");return -1;
      }
}
```

割込み操作

割り込みは、指定した BC メッセージ、 RT データテーブル毎、あるいはモニター環状バッファが1/3フルになった場合に生成されます。

BC / RT あるいはモニタにおいて、プログラム割込みが要求されるとき、RTLは割り込み処理を行い、ユーザによって（*ddcSetMonEvent()*あるいは *ddcSetBcrtEvent()*を使って）定義されたイベントハンドラー呼び出します。

ユーザ関数 *mon_event_handler()*は引数を持っていません。 RTL割込みハンドラはユーザ関数 *bcrt_event_handler()*に、以下の2つの引数を渡します。

U16BIT vector_type　－ 割込みベクトルの最初の16bit長のデータ。

U16BIT vector_parameter　－ 割込みベクトルの2番目の16bit長のデータ

モニタ割り込み

モニタは2つのタイプの割り込みを要求することができます:

－ それぞれのメッセージ後の割り込み

－ モニター環状バッファの1/3が満たされた（およそ 2K ワード）。

たった1つのモニター割込み種別を使用することもできます。; INTERRUPT MASK の14と15ビットで使用している割り込みをイネーブル/ディセーブルにします。

もしモニター割り込みを使用していて、それを *ddcHaltMon()*あるいは *ddcHaltIdea()*によって停止した場合にも、モニターは同様に割り込みを出します。 この割り込みは読み取り、カードの環状バッファの「テールエンド」を使用済みにすることが出来ます（*ddcReadMonStack()*と *ddcReadMonStackParteach()*参照）。

モニターが割り込みを求めるたびに、それはカウンターを増加させます。 もし類似のカウンターをユーザが増加させるなら、その値は失われた割り込みを検出するためにカードのカウンターと比較することができます。

BC(Bus Controller)と RT(Remote Terminal)割り込み.

BC / RT 割り込みはメッセージあるいはデータテーブルと関連づけられたメッセージ間に出されます。 それぞれの割込みはユーザに透過的に、巡回待ち行列にプッシュされた2ワードのベクトルを伴って要求されます。 キューは最高64の割り込みベクトルを持つことができます;それで、ホストはすぐに割り込みリクエストに応答を出すように要求されません。 ユーザルーチンは、キューのベクタ毎に一度だけ呼び出されます。

割込みベクトルは下記の様に定義されます:

Table 1. Interrupt Vector

SUB-TYPE	TYPE
PARAMETER	

Table 2. RT Interrupt Types

TYPE	SUB-TYPE	PARAMETER	EVENT
00	table number	Command	ME=0 transmit / receive
01	mode code	Command	ME=0 mode command
02	table number	Command	ME=1 transmit / receive
03	mode code	Command	ME=1 mode command
04-7F	mode code		RESERVED

注意: ME ビットは最後の状態(すなわち、モードコマンド SEND LAST STATUS に応えて送られるであろうステータス)からとられます

Table 3. BC Interrupt Types

TYPE	SUB-TYPE	PARAMETER	EVENT
80	message number	0000	successful message
81	message number	Error code	Communication error
82	message number	Status	bit set in status
83	FC	N/A	frame entry=SKIP
83	FD	N/A	frame entry=BREAK_POINT
83	FE	N/A	frame entry=END_OF_MAJOR_FRAME
83	FF	N/A	frame entry=END_OF_MINOR_FRAME
84-FF			RESERVED

Table 4. BC Error Codes

ERROR CODE	ERROR TYPE
0000	no error
0002	inverse data sync
0004	invalid data
0008	gap between data
0010	no response
0020	invalid status
0080	wrong TADR in status

割込みマスク

割込みマスクは BC / RT とモニター共有でメモリに存在する16ビットワードです。 マスクのそれぞれのビットが、「1」にセットされるとき、ユニークなインタラプトタイプを抑制します。

Table 5. Interrupt Mask

MASK BIT	INTERRUPT TYPE
0	00
1	01
2	02
3	03
4	80
5	81
6	82
7	83
8	XX
9	XX
10	XX
11	XX
12	XX
13	XX
14	1/3 of Circular Buffer (Monitor)
15	End of Msg (Monitor)

Intermessage ルーチン

それぞれの intermessage ギャップの間に、 BC / RT はユーザによって指定された2つの INTERMESSAGE ルーチンを実行します。

これらのルーチンはRTLとそれらに指定した以下のものから選ばれます:

－ メッセージが BC によって発行された

－ データテーブルがエミュレートされた RT によってアクセスされた

－ モードコマンドがエミュレートされた RT によって実行された

呼び出しルーチンの割り当ては *ddcDefMessage()*、*ddcDefTableRoutine()*と *ddcDefModeRoutine()*によって行われます。 もしメッセージがテスタ / シミュレーターカード内部で行われるなら、(すなわち、エミュレートされた BC とエミュレートされた RT の間に)、データテーブルと結び付けられるルーチンは無視されるでしょう。

同様に、もしカードが RT-to-RT 通信の RT の両方をエミュレートし BC をエミュレートしないなら、ただ送信 RT だけが intermessage ルーチンを実行します。

intermessage ルーチンはリセットプロセスの間にカードにダウンロードされます。 以下のテーブルが、intermessage ルーチンのリストです。

Table 13. Intermessage Routines

INDEX	FUNCTION
1	NO OPERATION
2	RETRY CURRRENT MESSAGE ON ALTERNATE BUS
3	RETRY CURRENT MESSAGE AND REMAIN ON ALTERNATE BUS
4	RETRY ON SAME BUS
5	INTERRUPT ON END OF MESSAGE
6	INTERRUPT ON FRAME SYMBOL
7	SET SERVICE REQUEST BIT IN STATUS
8	RESET SERVICE REQUEST BIT IN STATUS
9	INTERRUPT AFTER ACCESSING TX/RX DATA TABLE
10	INTERRUPT AFTER MODE COMMAND
11	INTERRUPT AFTER TX/RX COMMAND TEMPLATE MATCH
12	INTERRUPT AFTER MODE COMMAND TEMPLATE MATCH
13	TIME-TAG (STORE RTC IN A CIRCULAR QUEUE)
14	RESERVED
15	RETRY ON SAME BUS AND THEN ON ALTERNATE BUS
16	SET STATUS BIT IN STATUS
17	RESET STATUS BIT IN STATUS
18	SET OUTPUT TRIGGER
19	RESET OUTPUT TRIGGER
20	WAIT FOR INPUT TRIGGER
21	RESERVED
22	NO RESPONSE ON BOTH BUSES
23	SET BUSY BIT IN STATUS
24	RESET BUSY BIT IN STATUS
25	SET BUSY AND RESET SRQ IN STATUS
26	SET SRQ AND RESET BUSY IN STATUS
27-30	RESERVED
31	SKIP NEXT MESSAGE
32	SET_DISCRETE_0
33	SET_DISCRETE_1
34	SET_DISCRETE_2
35	SET_DISCRETE_3
36	RESET_DISCRETE_0
37	RESET_DISCRETE_1
38	RESET_DISCRETE_2
39	RESET_DISCRETE_3
40-43	RESERVED
44	SKIP NEXT MESSAGE ONCE
45-46	RESERVED
47	BLOCK_DATA_BC
48	BLOCK_DATA_RT
49-50	RESERVED
51	SKIP_NEXT_MESSAGE_ONCE_EX

詳細はData Device Corporation BU-69068 Manual ＡＰＰＥＮＤＩＸ Cを参照してください。

RT イベントハンドラーの登録

```
if( (Err = ddcSetBCRTEventEx(Dev, event_handler)) )
{
        ddcPrintErrorMessage(Dev, Err, "ddcSetBCRTEventEx");return -1;
}
if( (Err = ddcDef_int_mask_bcrt(Dev, 0x3F00U)) )
{
        ddcPrintErrorMessage(Dev, Err, "ddcDef_int_mask_bcrt");return -1;
}
```

RT イベントハンドラーの定義(Table 2. RT Interrupt Types を記述した関数)

```
Error_t event_handler(Device_p Crd, S16BIT Type, S16BIT SubType, S16BIT Param)
{
    CMD     *Cmd;
    S16BIT  TableID;
    S16BIT  TA, SA, WC,TR;
    U16BIT  ModeCode;

    Cmd = (CMD *)&Param;
    TA      = Cmd->tadr;        //アドレス
    SA      = Cmd->subadr;      //サブアドレス
    WC      = Cmd->wcnt;        //ワードカウント
    TR      = Cmd->t_r;         //1:送信  0:受信
    switch(Type)
    {
    case 0x00:// ME=0 transmit/receive
            TableID = SubType;
            if(TR==0)
            {//受信処理
                    ddcRead_data(Dev, TableID, &RecvBuff[TableID][0], WC);
            }
            else
            {//送信処理
                    ddcWrite_data(Dev, TableID, &SendBuff[TableID][0], WC, 1);
            }
            break;
    case 0x01:// ME=0 mode command
            ModeCode = (U16BIT)SubType;
            break;
    case 0x02:// ME=1 transmit/receive
            TableID = SubType;
            break;
    case 0x03:// ME=1 mode command
            ModeCode = (U16BIT)hSubType;
            break;
    default:
            break;
    }
    return 0;
}
```

BC イベントハンドラーの登録

```
if( (Err = ddcSetBCRTEventEx(Dev, event_handler)) )
{
        ddcPrintErrorMessage(Dev, Err, "ddcSetBCRTEventEx");
        exit(1);
}
if( (Err = ddcDef_int_mask_bcrt(Dev, 0x0F00U)) )
{
        ddcPrintErrorMessage(Dev, Err, "ddcDef_int_mask_bcrt" );
        exit(1);
}
```

BC **イベントハンドラーの定義**(Table 3. BC Interrupt Types **を記述した関数**)

```
Error_t event_handler(Device_p Crd, S16BIT Type, S16BIT SubType, S16BIT Param)
{
    MESSAGE         Message;
    S16BIT          MessageID;
    S16BIT          TA, SA, WC;
    Error_t         Err;
    static char     *MessTypeName[] =
    {
        "BC-RT", "RT-BC", "MODE", "RT-RT", "NOP"
    };

    switch(Type)
    {
    case 0x80:          // successful message
        break;
    case 0x81:          // communication error
        MessageID = SubType;
        if( (Err = ddcRead_message(Crd, MessageID, &Message)) )
        {
            ddcPrintErrorMessage(Crd, Err, "read_message");
            return Err;
        }
        TA = Message.cmd_1.tadr;
        SA = Message.cmd_1.subadr;
        WC = Message.cmd_1.wcnt;
        if(Message.cmd_1.t_r == 0)//メッセージのコマンドが受信つまり BC->RT,RT->RT
        {
          printf("[Mess %d] %s TA=%d RECV SA=%d WC=%d   << ErrorCode=%04x >>¥n",
          MessageID, MessTypeName[(int)Message.comm_type], TA, SA, WC,
           (U16BIT)Param);
          printf("      RX Status=%04x¥n", Message.rx_status);
        }
        else//メッセージのコマンドが送信つまり RT->BC
        {
          printf("[Mess %d] %s TA=%d XMIT SA=%d WC=%d   << ErrorCode=%04x >>¥n",
          MessageID, MessTypeName[(int)stMessage.comm_type], TA, SA, WC,
         (U16BIT)Param);
          printf("      TX Status=%04x¥n", Message.tx_status);
        }
        break;
    case 0x82:          // bit set in status
        hMessageID = SubType;
        if( (Err = ddcRead_message(Crd, MessageID, &Message)) )
        {
```

```
                ddcPrintErrorMessage(Crd, Err, "read_message");
                return Err;
            }
            TA = stMessage.cmd_1.tadr;
            SA = stMessage.cmd_1.subadr;
            WC = stMessage.cmd_1.wcnt;
            if(Message.cmd_1.t_r == 0) //受信
            {
              printf("[Mess %d] %s TA=%d RECV SA=%d WC=%d  << Status=%04x >>¥n",
              MessageID, MessTypeName[(int)Message.comm_type], TA, SA, WC,
              (U16BIT)Param);
              printf("      RX Status=%04x¥n", Message.rx_status);
            }
            else//送信
            {
              printf("[Mess %d] %s TA=%d XMIT SA=%d WC=%d  << Status=%04x >>¥n",
              MessageID, MessTypeName[(int)Message.comm_type], TA, SA, WC,
               (U16BIT)Param);
              printf("      TX Status=%04x¥n", Message.tx_status);
            }
            break;
    case 0x83:
            switch(hSubType)
            {
            case 0xfc:
                printf("[SKIP]¥n");
                break;
            case 0xfd:
                printf("[BREAK_POINT]¥n");
                break;
            case 0xfe:
                printf("[END_OF_MAJOR_FRAME]¥n");
                break;
            case 0xff:
                printf("[END_OF_MINOR_FRAME]¥n");
                break;
            default:
                printf("[unknown event] type=%04x subtype=%04x param=%04x¥n",
                    (U16BIT)Type, (U16BIT)SubType, (U16BIT)Param);
                break;
            }
            break;
    default:
            printf("[unknown event] type=%04x subtype=%04x param=%04x¥n",
                (U16BIT)Type, (U16BIT)SubType, (U16BIT)Param);
            break;
    }
    return 0;
}
```

BC と RT のイベントハンドラープログラム例

bc.c

```
#include <unistd.h>                  /* For 'sleep()' */
#include <tstsim/ts_drv.h>/* Tester / Simulator header */

Error_t EventHandler(Device_p Crd, S16BIT Type, S16BIT SubType, S16BIT Param)
{
     U16BIT Err;
        MESSAGE Message;
        S16BIT  MessageID;
        S16BIT  TA, SA, WC, TR , TI;
        S16BIT  TA2, SA2, WC2, TR2;
     S16BIT RBuff[32],TBuff[32],*Data;
     static char     *TRNmae[] = { "RECEIVE ","TRANSMIT",};//BC 側からみると反対になる
     static char     *MessTypeName[] =
     {
               "BC-RT", "RT-BC", "MODE", "RT-RT", "NOP"
     };
     static S16BIT count=0x0000;
     register int i;

        switch(Type)
        {
        case 0x80:      // successful message
               MessageID = SubType;
               if( (Err = ddcRead_message(Crd, MessageID, &Message)) )
               {
                       ddcPrintErrorMessage(Crd, Err, (S8BIT *)"read_message");
                       return Err;
               }
               TA  = Message.cmd_1.tadr;
               SA  = Message.cmd_1.subadr;
               WC  = Message.cmd_1.wcnt;
               TR  = Message.cmd_1.t_r;
               TA2 = Message.cmd_2.tadr;
               SA2 = Message.cmd_2.subadr;
               WC2 = Message.cmd_2.wcnt;
               TR2 = Message.cmd_2.t_r;
               TI  = Message.data_table_no;      /* データテーブル番号*/

               printf("Success [Mess %d] %s TA1=%d %s SA1=%d WC1=%d ",
               MessageID, MessTypeName[(int)Message.comm_type], TA,TRNmae[TR], SA, WC);
               if (Message.comm_type==RT_RT)
               {
                         printf("TA2=%d %s SA2=%d WC2=%d ",TA2,TRNmae[TR2], SA2, WC2);
               }
               if (Message.comm_type == TRANSMIT)
               {
                         Data = RBuff;
                        if( (Err = ddcRead_data(Crd, TI, &RBuff[0],WC)) )
                        {
                                ddcPrintErrorMessage(Crd, Err, (S8BIT *)"ddcRead_data");
                        }
                         printf("TI=%d <<RData[0]=0x%04X >>¥n",TI, (U16BIT)Data[0]);
               }
               else if (Message.comm_type == RECEIVE)
               {
                         count++;
                         Data = TBuff;
                         for(i=0; i<32; i++) TBuff[i] = count;
                        if( (Err = ddcWrite_data(Crd, TI, &TBuff[0], WC, 1)) )
                        {
                                ddcPrintErrorMessage(Crd, Err, (S8BIT *)"ddcWrite_data");
                        }
                         printf("TI=%d <<TData[0]=0x%04X >>¥n",TI, (U16BIT)Data[0]);
               }
               else
               {
                         printf("TI=%d¥n",TI);
               }
     break;
```

```
        case 0x81:      // communication error
                MessageID = SubType;
                if( (Err = ddcRead_message(Crd, MessageID, &Message)) )
                {
                        ddcPrintErrorMessage(Crd, Err, (S8BIT *)"read_message");
                        return Err;
                }
                TA = Message.cmd_1.tadr;
                SA = Message.cmd_1.subadr;
                WC = Message.cmd_1.wcnt;
                TR = Message.cmd_1.t_r;
                TA2 = Message.cmd_2.tadr;
                SA2 = Message.cmd_2.subadr;
                WC2 = Message.cmd_2.wcnt;
                TR2 = Message.cmd_2.t_r;

                printf("Error   [Mess %d] %s TA1=%d %s SA1=%d WC1=%d ",
                MessageID, MessTypeName[(int)Message.comm_type], TA,TRNmae[TR], SA, WC);
                if (Message.comm_type==RT_RT)
                {
                          printf("TA2=%d %s SA2=%d WC2=%d",TA2,TRNmae[TR2], SA2, WC2);
                }
                printf(" << ErrorCode=%04x >>¥n", (U16BIT)Param);
        break;
        case 0x83:
                switch(SubType)
                {
                case 0xfc:
                      printf("[SKIP]¥n");
                        break;
                case 0xfd:
                      printf("[BREAK_POINT]¥n");
                        break;
                case 0xfe:
                      printf("[END_OF_MAJOR_FRAME]¥n");
                        break;
                case 0xff:
                      printf("[END_OF_MINOR_FRAME]¥n");
                        break;
                default:
                        printf("[unknown event] type=%04x subtype=%04x param=%04x¥n",
                                (U16BIT)Type, (U16BIT)SubType, (U16BIT)Param);
                        break;
                }
                break;

        default:
                printf("[unknown event] type=%04x subtype=%04x param=%04x¥n",
                        (U16BIT)Type, (U16BIT)SubType, (U16BIT)Param);
                break;
        }
    return 0;
}

int main()
{
    MESSAGE   Message;
    INJ_ERR   InjErr;
    U16BIT              Frame[10];
    DRV_CONFIG          cfg;
    Error_t   Err;
    S16BIT              i, data[32];
    Device_p  Dev;
    U8BIT               Channel;

    Channel = 0;
    printf("Resetting card...¥n");
    if( (Err = ddcResetCard(&Dev, &cfg, Channel)) ){
              ddcPrintErrorMessage(Dev, Err, (S8BIT *)"reset_card");
              exit(1);
    }
    printf("Card reset successful!¥n");
```

```
    /* Set the callback function. */

    if( (Err = ddcSetBCRTEventEx(Dev, EventHandler)) )
    {
              ddcPrintErrorMessage(Dev, Err, (S8BIT *)"set_mon_evnt");
              exit(3);
    }
    printf("Setting up BC ...¥n");

    /* Set termination for loop back with no card connections. */
    if( (Err = ddcSetCoupling(Dev, COUPLING_TRANSFORMER, TERMINATION_HALF)) ){
              ddcPrintErrorMessage(Dev, Err, (S8BIT *)"SetCoupling");
              exit(1);
    }
    if  ( (Err=ddcSetAmp(Dev,0xFFF) ) )
    {
              ddcPrintErrorMessage(Dev, Err, (S8BIT *)"SetAmp");
              exit(1);
    }

/*  Define Message 1 */
    Message.bus = BUS_A;
    Message.comm_type = RT_RT;
    Message.cmd_1.wcnt = 31;
    Message.cmd_1.tadr = 1;         /* RT Address is 1 */
    Message.cmd_1.subadr = 1;       /* RT Sub address is 1 */
    Message.cmd_1.t_r = RECEIVE;    /* should be 0 for a RECEIVE command */
    Message.cmd_2.wcnt = 31;
    Message.cmd_2.tadr = 2;         /* RT Address is 2 */
    Message.cmd_2.subadr = 1;       /* RT Sub address is 1 */
    Message.cmd_2.t_r = TRANSMIT;   /* should be 1 for a TRANSMIT command */
    Message.time_to_next_message = 100;
    Message.data_table_no = 1;      /* データテーブルは 1*/
    Message.last_data_table_no  = 1;/* データテーブルは 1*/
    Message.first_intermessage_routine = INT_ON_END_OF_MESSAGE;
    Message.second_intermessage_routine = NO_OPERATION;
    Message.inj_error_ptr = &InjErr;
    InjErr.error = E_NONE;
    InjErr.sub_error_1 = 0;
    InjErr.sub_error_2 = 0;
    InjErr.sub_error_3 = 0;
    if( (Err = ddcDef_message (Dev, 1, &Message )) ){
              ddcPrintErrorMessage(Dev, Err, (S8BIT *)"def_message");
              exit(1);
    }

/*  Define Message 2 */
    Message.bus = BUS_A;
    Message.comm_type = RT_RT;
    Message.cmd_1.wcnt = 31;
    Message.cmd_1.tadr = 2;                     /* RT Address is 2 */
    Message.cmd_1.subadr = 1;       /* RT Sub address is 1 */
    Message.cmd_1.t_r = RECEIVE;
    Message.cmd_2.wcnt = 31;
    Message.cmd_2.tadr = 1;                     /* RT Address is 1 */
    Message.cmd_2.subadr = 1;       /* RT Sub address is 1 */
    Message.cmd_2.t_r = TRANSMIT;
    Message.time_to_next_message = 100;
    Message.data_table_no = 2;      /* データテーブルは 2*/
    Message.last_data_table_no  = 2;            /* データテーブルは 2*/
    Message.first_intermessage_routine = INT_ON_END_OF_MESSAGE;
    Message.second_intermessage_routine = NO_OPERATION;
    Message.inj_error_ptr = &InjErr;
    InjErr.error = E_NONE;
    InjErr.sub_error_1 = 0;
    InjErr.sub_error_2 = 0;
    InjErr.sub_error_3 = 0;
    if( (Err = ddcDef_message (Dev, 2, &Message )) ){
              ddcPrintErrorMessage(Dev, Err, (S8BIT *)"def_message");
              exit(1);
    }

/*  Define Message 3. Only changed values are listed */
```

```
    Message.bus = BUS_A;
    Message.comm_type = RECEIVE;
    Message.cmd_1.wcnt = 31;
    Message.cmd_1.tadr = 3;                    /* RT Address is 3 */
    Message.cmd_1.subadr = 1;       /* RT Sub address is 1 */
    Message.cmd_1.t_r = RECEIVE;
    Message.cmd_2.wcnt = 0;
    Message.cmd_2.tadr = 0;
    Message.cmd_2.subadr = 0;
    Message.time_to_next_message = 100;
    Message.data_table_no = 3;      /* データテーブルは 3*/
    Message.last_data_table_no  = 3;          /* データテーブルは 3*/
    Message.first_intermessage_routine = INT_ON_END_OF_MESSAGE;
    Message.second_intermessage_routine = NO_OPERATION;
    Message.inj_error_ptr = &InjErr;
    InjErr.error = E_NONE;
    InjErr.sub_error_1 = 0;
    InjErr.sub_error_2 = 0;
    InjErr.sub_error_3 = 0;
    if( (Err = ddcDef_message (Dev, 3, &Message )) ){
              ddcPrintErrorMessage(Dev, Err, (S8BIT *)"def_message");
              exit(1);
    }

/*  Define Message 4. Only changed values are listed */
    Message.bus = BUS_A;
    Message.comm_type = TRANSMIT;
    Message.cmd_1.wcnt = 31;
    Message.cmd_1.tadr = 4;                    /* RT Address is 4 */
    Message.cmd_1.subadr = 1;       /* RT Sub address is 1 */
    Message.cmd_1.t_r = TRANSMIT;
    Message.cmd_2.wcnt = 0;
    Message.cmd_2.tadr = 0;
    Message.cmd_2.subadr = 0;
    Message.time_to_next_message = 100;
    Message.data_table_no = 4;      /* データテーブルは 4*/
    Message.last_data_table_no  = 4;          /* データテーブルは 4*/
    Message.first_intermessage_routine = INT_ON_END_OF_MESSAGE;
    Message.second_intermessage_routine = NO_OPERATION;
    Message.inj_error_ptr = &InjErr;
    InjErr.error = E_NONE;
    InjErr.sub_error_1 = 0;
    InjErr.sub_error_2 = 0;
    InjErr.sub_error_3 = 0;
    if( (Err = ddcDef_message (Dev, 4, &Message )) ){
              ddcPrintErrorMessage(Dev, Err, (S8BIT *)"def_message");
              exit(1);
    }
      Message.comm_type                   = NOP;
      Message.first_intermessage_routine  = INT_FRAME_SYMBOL;
      Message.second_intermessage_routine = NO_OPERATION;

      if( (Err = ddcDef_message(Dev, END_OF_MINOR, &Message)) )
      {
              ddcPrintErrorMessage(Dev, Err, (S8BIT *)"ddcDef_message");
              exit(1);
      }

      if( (Err = ddcDef_message(Dev, END_OF_MAJOR, &Message)) )
      {
              ddcPrintErrorMessage(Dev, Err, (S8BIT *)"ddcDef_message");
              exit(1);
      }




/*  Set up BC Frame using messages defined above */
    Frame[0] = 1;        /* RT->RT */
```

```
      Frame[1] = END_OF_MINOR;
      Frame[2] = 2;        /* RT->RT */
      Frame[3] = END_OF_MINOR;
      Frame[4] = 3;        /* BC->RT */
      Frame[5] = END_OF_MINOR;
      Frame[6] = 4;        /* RT->BC */
      Frame[7] = END_OF_MAJOR;

      if( (Err = ddcDef_frame ( Dev, 8, Frame )) ){
                ddcPrintErrorMessage(Dev, Err, (S8BIT *)"def_frame");
                Err = ddcShutDownIdea( &Dev );
                exit(1);
      }

      if( (Err = ddcDef_minor_frame_time ( Dev, 1000 )) ){        // 1msec
                ddcPrintErrorMessage(Dev, Err, (S8BIT *)"def_minor_frame_time");
                exit(1);
      }

      for(i=0; i<4; i++){
                if( (Err = ddcDef_table_size (Dev, i+1, 32)) ){
                          ddcPrintErrorMessage(Dev, Err, (S8BIT *)"def_table_size");
                          Err = ddcShutDownIdea( &Dev );
                          exit(1);
                }
      }
      for(i=0; i<32; i++)
                data[i] = 0x1111;
      ddcWrite_data(Dev, 1, data, 32, 1);

      for(i=0; i<32; i++)
                data[i] = 0x2222;
      ddcWrite_data(Dev, 2, data, 32, 1);

      for(i=0; i<32; i++)
                data[i] = 0x3333;
      ddcWrite_data(Dev, 3, data, 32, 1);

      for(i=0; i<32; i++)
                data[i] = 0x4444;
      ddcWrite_data(Dev, 4, data, 32, 1);

         if( (Err = ddcDef_int_mask_bcrt(Dev, 0x3F00U)) )
         {
                  ddcPrintErrorMessage(Dev, Err, (S8BIT *)"ddcDef_int_mask_bcrt" );
                  exit(1);
         }

/*    Run BC Frame twice */
      if( (Err = ddcRun_bc (Dev, 1, 2)) ){//メジャーフレームを２回実行する
                ddcPrintErrorMessage(Dev, Err, (S8BIT *)"run_bc");
                Err = ddcShutDownIdea( &Dev );
                exit(1);
      }

/*    delay until BC finishes */
      //sleep(1);
      /* Wait for user keypress to quit. */
      while(getchar() == 0);
      if( (Err = ddcRead_message(Dev, 1, &Message)) ){
                ddcPrintErrorMessage(Dev, Err, (S8BIT *)"read_message");
                Err = ddcShutDownIdea( &Dev );
                exit(1);
      }
     printf("Message1 Rx Status = %4.4x, Detected error = %d¥n", Message.rx_status, Message.det_error);
      if( (Err = ddcRead_message(Dev, 2, &Message)) ){
                ddcPrintErrorMessage(Dev, Err, (S8BIT *)"read_message");
                Err = ddcShutDownIdea( &Dev );
                exit(1);
      }
     printf("Message2 Tx Status = %4.4x, Detected error = %d¥n", Message.tx_status, Message.det_error);

      /* Write the application level buffer to file. */
```

```
    if( (Err = ddcShutDownIdea( &Dev )) ){
              ddcPrintErrorMessage(Dev, Err, (S8BIT *)"shut_down_idea");
              exit(1);
    }
    printf("Card shut down successful, terminating program.¥n");
    return 0;
}
```

rt.c

```
#include <unistd.h>                  /* For 'sleep()' */
#include <tstsim/ts_drv.h>/* Tester / Simulator header */

Error_t EventHandler(Device_p Crd, S16BIT Type, S16BIT SubType, S16BIT Param)
{
     U16BIT Err;
        CMD     *Cmd;
        S16BIT  TA, SA, WC, TR,TI;
     U16BIT  MC;
     S16BIT RBuff[32],TBuff[32];
     static S16BIT count=0x0000;
     register int i;


        Cmd = (CMD *)&Param;
        TA    = Cmd->tadr;
        SA    = Cmd->subadr;
        WC    = Cmd->wcnt;
        TR    = Cmd->t_r;
     printf("Type=0x%02x sub=0x%02x TA=%d TR=%d SA=%d WC=%d", Type, SubType, TA, TR, SA, WC);
        switch(Type)
        {
                case 0x00:      // ME=0 transmit/receive
                           TI = SubType;
                           if(TR == RECEIVE)
                           {
                                     if( (Err = ddcRead_data(Crd, TI, &RBuff[0],WC)) )
                                     {
                                     ddcPrintErrorMessage(Crd, Err, (S8BIT *)"ddcRead_data");
                                     }
                                     printf(" <<RData[0]=0x%04X >>¥n", (U16BIT)RBuff[0]);
                           }
                           else
                           {
                                     count--;
                                     for(i=0; i<32; i++) TBuff[i] = count;
                                     if( (Err = ddcWrite_data(Crd, TI, &TBuff[0], WC, 1)) )
                           {
                                     ddcPrintErrorMessage(Crd, Err, (S8BIT *)"ddcWrite_data");
                                     }
                                     printf(" <<TData[0]=0x%04X >>¥n", (U16BIT)TBuff[0]);
                           }
                break;
                case 0x01:      // ME=0 mode command
                           MC = (U16BIT)SubType;
                           if(TR == RECEIVE)
                           {
                           }
                           else
                           {
                           }
                           printf("¥n");
                break;
                case 0x02:      // ME=1 transmit/receive
                           TI = SubType;
                           if(TR == RECEIVE)
                           {
                           }
                           else
                           {
                           }
                           printf("¥n");
                break;
                case 0x03:      // ME=1 mode command
                           MC = (U16BIT)SubType;
                           printf("¥n");
                break;
                default:
                           printf("¥n");
                break;
```

```
	}
	return 0;
}

int main()
{
	INJ_ERR	InjErr;
	RT_DEFS	rt;
	DRV_CONFIG		cfg;
	Error_t	Err;
	S16BIT		i,j, data[32];
	Device_p	Dev;
	U8BIT		Channnel;

	Channnel = 1;
	printf("Resetting card...¥n");
	if( (Err = ddcResetCard(&Dev, &cfg, Channnel)) ){
		ddcPrintErrorMessage(Dev, Err, (S8BIT *)"reset_card");
		exit(1);
	}
	printf("Card reset successful!¥n");

	  if( (Err = ddcDef_data_buffering(Dev, /*SINGLE*/ DOUBLE )) )
	  {
		ddcPrintErrorMessage(Dev, Err, (S8BIT *)"ddcDef_data_buffering");
		return -1;
	  }

	printf("Setting up RTs...¥n");


	/* Set termination for loop back with no card connections. */
	if( (Err = ddcSetCoupling(Dev, COUPLING_TRANSFORMER, TERMINATION_HALF)) ){
		ddcPrintErrorMessage(Dev, Err, (S8BIT *)"SetCoupling");
		exit(1);
	}
	if  ( (Err=ddcSetAmp(Dev,0xFFF) ) )
	{
		ddcPrintErrorMessage(Dev, Err, (S8BIT *)"SetAmp");
		exit(1);
	}

/*	Set up RT1 */
	InjErr.error = E_NONE;
	InjErr.sub_error_1 = 0;
	InjErr.sub_error_2 = 0;
	InjErr.sub_error_3 = 0;

	rt.inj_error = &InjErr;
	rt.basic_status = 0x0000;
	rt.dbca = NO;

	if( (Err = ddcDef_emulate_rt( Dev, 1, YES)) ){
		ddcPrintErrorMessage(Dev, Err, (S8BIT *)"def_emulate_rt");
		Err = ddcShutDownIdea( &Dev );
		exit(1);
	}
	if( (Err = ddcDef_rt(Dev,/*TA*/ 1, &rt)) ){
		ddcPrintErrorMessage(Dev, Err, (S8BIT *)"def_rt");
		Err = ddcShutDownIdea( &Dev );
		exit(1);
	}
	if( (Err = ddcDef_rt_map(Dev, /*TA*/1, RECEIVE, /*SA*/1, /*TB*/1)) ){//受信用にテーブルが１つ
		ddcPrintErrorMessage(Dev, Err, (S8BIT *)"def_rt_map");
		Err = ddcShutDownIdea( &Dev );
		exit(1);
	}
	if( (Err = ddcDef_rt_map(Dev, /*TA*/1, TRANSMIT, /*SA*/1, /*TB*/2)) ){//送信用にテーブルが１つ
		ddcPrintErrorMessage(Dev, Err, (S8BIT *)"def_rt_map");
		Err = ddcShutDownIdea( &Dev );
		exit(1);
	}
```

```
/*   Set up RT2 */
     if( (Err = ddcDef_emulate_rt(Dev, /*TA*/ 2, YES)) ){
               ddcPrintErrorMessage(Dev, Err, (S8BIT *)"def_emulate_rt");
               Err = ddcShutDownIdea( &Dev );
               exit(1);
     }
     if( (Err = ddcDef_rt(Dev,/*TA*/ 2, &rt)) ){
               ddcPrintErrorMessage(Dev, Err, (S8BIT *)"def_rt");
               Err = ddcShutDownIdea( &Dev );
               exit(1);
     }
     if( (Err = ddcDef_rt_map(Dev, /*TA*/2, RECEIVE, /*SA*/1, /*TB*/3)) ){
               ddcPrintErrorMessage(Dev, Err, (S8BIT *)"def_rt_map");
               Err = ddcShutDownIdea( &Dev );
               exit(1);
     }
     if( (Err = ddcDef_rt_map(Dev, /*TA*/2, TRANSMIT, /*SA*/1, /*TB*/4)) ){
               ddcPrintErrorMessage(Dev, Err, (S8BIT *)"def_rt_map");
               Err = ddcShutDownIdea( &Dev );
               exit(1);
     }
/*   Set up RT3 */
     if( (Err = ddcDef_emulate_rt(Dev, /*TA*/ 3, YES)) ){
               ddcPrintErrorMessage(Dev, Err, (S8BIT *)"def_emulate_rt");
               Err = ddcShutDownIdea( &Dev );
               exit(1);
     }
     if( (Err = ddcDef_rt(Dev,/*TA*/ 3, &rt)) ){
               ddcPrintErrorMessage(Dev, Err, (S8BIT *)"def_rt");
               Err = ddcShutDownIdea( &Dev );
               exit(1);
     }
     if( (Err = ddcDef_rt_map(Dev, /*TA*/3, TRANSMIT, /*SA*/1, /*TB*/5)) ){
               ddcPrintErrorMessage(Dev, Err, (S8BIT *)"def_rt_map");
               Err = ddcShutDownIdea( &Dev );
               exit(1);
     }
     if( (Err = ddcDef_rt_map(Dev, /*TA*/3, RECEIVE, /*SA*/1, /*TB*/6)) ){
               ddcPrintErrorMessage(Dev, Err, (S8BIT *)"def_rt_map");
               Err = ddcShutDownIdea( &Dev );
               exit(1);
     }
/*   Set up RT4 */
     if( (Err = ddcDef_emulate_rt(Dev, /*TA*/ 4, YES)) ){
               ddcPrintErrorMessage(Dev, Err, (S8BIT *)"def_emulate_rt");
               Err = ddcShutDownIdea( &Dev );
               exit(1);
     }
     if( (Err = ddcDef_rt(Dev,/*TA*/ 4, &rt)) ){
               ddcPrintErrorMessage(Dev, Err, (S8BIT *)"def_rt");
               Err = ddcShutDownIdea( &Dev );
               exit(1);
     }
     if( (Err = ddcDef_rt_map(Dev, /*TA*/4, TRANSMIT, /*SA*/1, /*TB*/7)) ){
               ddcPrintErrorMessage(Dev, Err, (S8BIT *)"def_rt_map");
               Err = ddcShutDownIdea( &Dev );
               exit(1);
     }
     if( (Err = ddcDef_rt_map(Dev, /*TA*/4, RECEIVE, /*SA*/1, /*TB*/8)) ){
               ddcPrintErrorMessage(Dev, Err, (S8BIT *)"def_rt_map");
               Err = ddcShutDownIdea( &Dev );
               exit(1);
     }

/*   Set up Data Tables */
     for(i=0; i<8; i++)
     {
               if( (Err = ddcDef_table_size (Dev, i+1, 32)) ){
                         ddcPrintErrorMessage(Dev, Err, (S8BIT *)"def_table_size");
                         Err = ddcShutDownIdea( &Dev );
                         exit(1);
               }
               for(j=0; j<32; j++) data[j] = i+1;   //テーブル番号を初期値に設定
```

```
                ddcWrite_data(Dev, i+1, data, 32, 1);//SINGLEバッファなら１度で良い
                ddcWrite_data(Dev, i+1, data, 32, 1);//DOUBLEバッファなら２度書き出す必要がある
                if( (Err = ddcDef_table_routine(Dev, i+1, RT_INT_AFTER_DATA, NO_OPERATION)) )
                {
                        ddcPrintErrorMessage(Dev, Err, (S8BIT *)"ddcDef_table_routine");
                        return -1;
                }
        }

        /* Set the callback function. */
        if( (Err = ddcSetBCRTEventEx(Dev, EventHandler)) )
        {
                ddcPrintErrorMessage(Dev, Err, (S8BIT *)"set_rt_evnt");
                exit(3);
        }

           if( (Err = ddcDef_int_mask_bcrt(Dev, 0x3F00U)) )
           {
                   ddcPrintErrorMessage(Dev, Err, (S8BIT *)"ddcDef_int_mask_bcrt");
                   return -1;
           }

           if( (Err = ddcRun_rt(Dev)) )
           {
                   ddcPrintErrorMessage(Dev, Err, (S8BIT *)"ddcRun_rt");
                   return -1;
           }

           printf("RT start¥n¥n");

        sleep(1);
        /* Wait for user keypress to quit. */
        while(getchar() == 0);

        if( (Err = ddcShutDownIdea( &Dev )) ){
                ddcPrintErrorMessage(Dev, Err, (S8BIT *)"shut_down_idea");
                exit(1);
        }
        printf("Card shut down successful, terminating program.¥n");
        return 0;
}
```

BC と RT のイベントハンドラープログラム結果

ｂｃ側

```
[root@ihawk ddc.sample]# ./bc
Resetting card...
Card reset successful!
Setting up BC ...
Success [Mess 1] RT-RT TA1=1 RECEIVE  SA1=1 WC1=31 TA2=2 TRANSMIT SA2=1 WC2=31 TI=1
[END_OF_MINOR_FRAME]
Success [Mess 2] RT-RT TA1=2 RECEIVE  SA1=1 WC1=31 TA2=1 TRANSMIT SA2=1 WC2=31 TI=2
[END_OF_MINOR_FRAME]
Success [Mess 3] BC-RT TA1=3 RECEIVE  SA1=1 WC1=31 TI=3 <<TData[0]=0x0001 >>
[END_OF_MINOR_FRAME]
Success [Mess 4] RT-BC TA1=4 TRANSMIT SA1=1 WC1=31 TI=4 <<RData[0]=0x0007 >>
[END_OF_MAJOR_FRAME]
Success [Mess 1] RT-RT TA1=1 RECEIVE  SA1=1 WC1=31 TA2=2 TRANSMIT SA2=1 WC2=31 TI=1
[END_OF_MINOR_FRAME]
Success [Mess 2] RT-RT TA1=2 RECEIVE  SA1=1 WC1=31 TA2=1 TRANSMIT SA2=1 WC2=31 TI=2
[END_OF_MINOR_FRAME]
Success [Mess 3] BC-RT TA1=3 RECEIVE  SA1=1 WC1=31 TI=3 <<TData[0]=0x0002 >>
[END_OF_MINOR_FRAME]
Success [Mess 4] RT-BC TA1=4 TRANSMIT SA1=1 WC1=31 TI=4 <<RData[0]=0xFFFD >>
[END_OF_MAJOR_FRAME]

Message1 Rx Status = 0800, Detected error = 0
Message2 Tx Status = 0800, Detected error = 0
Card shut down successful, terminating program.
```

RT2→RT1

RT1→RT2

初期化の次に書き出したデータ

ｒｔ側

```
Resetting card...
Card reset successful!
Setting up RTs...
RT start

Type=0x00 sub=0x04 TA=2 TR=1 SA=1 WC=31 <<TData[0]=0xFFFF >>
Type=0x00 sub=0x01 TA=1 TR=0 SA=1 WC=31 <<RData[0]=0x0004 >>
Type=0x00 sub=0x02 TA=1 TR=1 SA=1 WC=31 <<TData[0]=0xFFFE >>
Type=0x00 sub=0x03 TA=2 TR=0 SA=1 WC=31 <<RData[0]=0x0002 >>
Type=0x00 sub=0x06 TA=3 TR=0 SA=1 WC=31 <<RData[0]=0x3333 >>
Type=0x00 sub=0x07 TA=4 TR=1 SA=1 WC=31 <<TData[0]=0xFFFD >>
Type=0x00 sub=0x04 TA=2 TR=1 SA=1 WC=31 <<TData[0]=0xFFFC >>
Type=0x00 sub=0x01 TA=1 TR=0 SA=1 WC=31 <<RData[0]=0xFFFF >>
Type=0x00 sub=0x02 TA=1 TR=1 SA=1 WC=31 <<TData[0]=0xFFFB >>
Type=0x00 sub=0x03 TA=2 TR=0 SA=1 WC=31 <<RData[0]=0xFFFE >>
Type=0x00 sub=0x06 TA=3 TR=0 SA=1 WC=31 <<RData[0]=0x0001 >>
Type=0x00 sub=0x07 TA=4 TR=1 SA=1 WC=31 <<TData[0]=0xFFFA >>

Card shut down successful, terminating program.
```

1回目の TRANSMIT で書き出した 0xFFFF データを 2 回目で受信

バッファ番号 4 が初期値なのは、RT2 の TRANSMIT

同様に RT1 の TRANSMIT

RT2→RT1

RT1→RT2

BC→RT3

2 回目の RT4→BC データ

1 回目

RT2→RT1

RT1→RT2

BC→RT3

3 回目の RT4→BC データ

2 回目

初期化時に 0x3333 を書き出している

RT2→RT1 0x0004	RT1→RT2 0x0002	BC→RT3 0x3333	RT4→BC 0x0007

初期化時に 0x0007 を書き出している

初期化時の RT 側のデータは、テーブル番号

BC 送信後割込み時に 0x0001 を書き出している

RT2→RT1 0xFFFF	RT1→RT2 0xFFFE	BC→RT3 0x0001	RT4→BC 0xFFFD

RT 送信後割込み時に 0xFFFD を書き出している

割り込み時 RT 側のデータは、デクリメント