

MMC/SD アクセスユニット

デモプログラム解説

目次

| | |
|-----------------------------------|----|
| □注意事項..... | 1 |
| 限定保証..... | 1 |
| 免責事項..... | 1 |
| 本資料について..... | 2 |
| 本資料ご利用に関する注意事項..... | 2 |
| デモプログラムで使用した CPU ボード..... | 2 |
| MMC/SD アクセスユニットの CPU..... | 2 |
| MMC/SD アクセスユニットの接続例..... | 2 |
| デモプログラムの動作概要..... | 3 |
| デモプログラムの構成..... | 4 |
| デモプログラムにおける RAM メモリ使用量..... | 5 |
| デモプログラムの説明..... | 5 |
| ●デモプログラム 1 の説明..... | 5 |
| ●デモプログラム 1 の実行結果..... | 6 |
| ●デモプログラム 1 のメインプログラム..... | 6 |
| ●デモプログラムにおける主要な処理内容..... | 10 |
| ○ポートの初期化..... | 10 |
| ○シリアルポートの初期化..... | 10 |
| ○UART0(パソコン)からの 1 行入力..... | 11 |
| ○入力コマンド列の切り分け..... | 12 |
| ○コマンド列のパラメータチェック..... | 14 |
| ○コマンド列の送信..... | 15 |
| ○応答文字列の受信..... | 15 |
| ○パソコンへの応答内容送信..... | 16 |
| コマンド/応答処理の例..... | 17 |
| 1) ファイルのリード (FRED コマンド)..... | 17 |
| 2) ファイルのライト (FWRT コマンド)..... | 17 |
| 3) ディレクトリ項目の読みだし (RDIR コマンド)..... | 18 |
| 終わりに..... | 19 |

□注意事項

本書を必ずよく読み、ご理解された上でご利用下さい

- 本書は株式会社北斗電子製 MMC/SD アクセスユニット本体の使用方法及び付属ソフトについて説明するものであり、ユーザシステムは対象ではありません。
- MMC/SD アクセスユニットは弊社評価用 CPU ボード（BaseBoard シリーズ、HSB シリーズ）に接続することで簡単に動作致します。対応する評価用 CPU ボードをお持ちでない場合は北斗電子製ボードをご利用頂くと便利です。
- 本製品を使用して発生した不具合について弊社は一切の責任を負いません。組み込み用途で使用する場合は十分に検証を行いお客様の責任においてご使用下さい。
- 医療用機器など誤動作により人体や財産への危険が起こりうる機器では使用しないで下さい。
- MMC/SD アクセスユニットのデザイン・機能・仕様は予告なく変更する場合があります。本書の図は実物と異なる場合もあります。
- 本書及び製品は著作権及び工業所有権によって保護されており、全ての権利は弊社に帰属します。本書の無断複写・複製・転載はできません。
- 弊社は安全にご利用頂く為に検討・対策を行っておりますが、潜在的な危険・誤使用については全てを予見できません。本書に記載されている警告が全てではありませんので、お客様の責任で理解・判断し正しく安全にご利用下さい。
- MMC/SD アクセスユニット本体の価格、又は仕様（本書含む）は予告無く変更される場合があります。

弊社 Base Board シリーズ※1、HSB シリーズ※2 CPU ボード(一部を除く)の 20 ピンソケットに直結して使用する事ができます

※1 RAM 容量の少ない一部の CPU ボードでは、動作致しません

※2 一部の CPU ボードでは、動作致しません

限定保証

弊社は MMC/SD アクセスユニットが頒布されているご利用条件に従って製造されたもので、材料・仕上げに欠陥がないことを保証致します。MMC/SD アクセスユニットの保証期間は購入頂いた日から 1 年間です。

免責事項

- 火災・地震・第三者による行為その他の事故により MMC/SD アクセスユニットに不具合が生じた場合
- お客様の故意・過失・誤用・異常な条件でのご利用によって MMC/SD アクセスユニットに不具合が生じた場合
- MMC/SD アクセスユニット及び付属品へのご利用方法に起因した損害が発生した場合
- お客様によって MMC/SD アクセスユニット及び付属品へ改造・修理がなされた場合

弊社は特定の目的・用途に関する保証や特許侵害に対する保証等、本保証条件以外のは明示・黙示に拘わらず一切保証致しません。また、直接的・間接的損害金もしくは欠陥製品や製品の使用方法に起因する損失金・費用には一切責任がありません。損害の発生についてあらかじめ知らされていた場合でも保証致しません。

MMC/SD アクセスユニットは「現状」で販売されているものであり、使用に際してはお客様がその結果に一切の責任を負うものとします。弊社は使用または使用不能から生ずる損害に関して一切責任を負いません。保証は最初の購入者であるお客様ご本人にのみ適用され、お客様が転売された第三者には適用されません。よって転売による第三者またはその為になすお客様からのいかなる請求についても責任を負いません。

本資料について

この資料は、他の CPU ボードから、(株)北斗電子製 MMC/SD アクセスユニット SDC23 を用いてアプリケーションプログラムを作成する場合の理解を得るために、CD に収録したデモプログラムの解説しております。MMC/SD アクセスユニット SDC23 を用いて、実際のアプリケーションプログラム作成する場合の参考としてご利用下さい。

本資料ご利用に関する注意事項

MMC/SD アクセスユニット SDC23 (以下、本アクセスユニット) は他の CPU ボードとシリアルインタフェース接続し調歩同期通信によるデータアクセスを行います。

本資料では、(株)北斗電子製の CPU ボード BB48S8C0 と CPU ボードを搭載した MMC/SD アクセスユニット SDC23 (以下、本アクセスユニット) をシリアルインタフェース接続した場合の CPU ボード BB48S8C0 側のプログラムについて解説しています。他の CPU ボードと接続した場合には、プログラムの変更が必要となりますが、汎用のシリアルインタフェースを使用しているため、シリアルインタフェースの初期化処理など一部のプログラム変更によって、容易に同様の動作を得る事ができます。

概略を理解するものであり、この資料のみで本アクセスユニットの動作全てを理解出来るようには記載しておりませんので予めご了承願います。

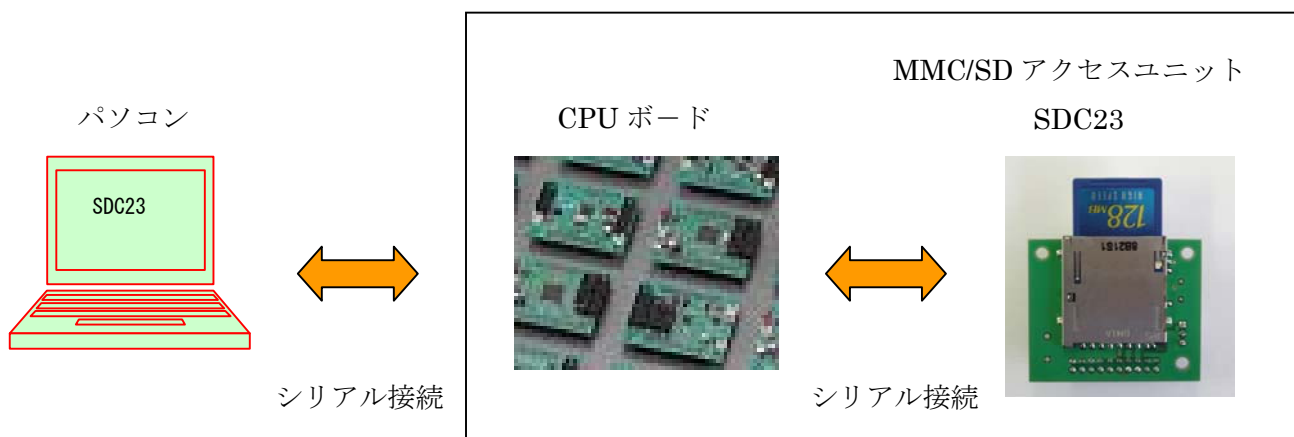
デモプログラムで使用した CPU ボード

ボード名 弊社 R8Ctiny Base Board シリーズ BB48S8C20 ボード型名 BB48A227JF
(搭載 CPU R5F21227JFP 内蔵 ROM48KB、内蔵 RAM 2.5KB)

MMC/SD アクセスユニットの CPU

CPU の型名 : R5F2123CJFP (内蔵 ROM 128KB (+2KB)、内蔵 RAM 6KB)

MMC/SD アクセスユニットの接続例



- * 通常は CPU ボードと本アクセスユニットの組み合わせ（枠内）での利用することが多いと考えられるが本解説では、本アクセスユニットの動作の理解を容易とするため、ターミナルソフトを起動したパソコンと CPU ボードを非同期シリアル接続（RS232C レベル）、CPU ボードと本アクセスユニット間を非同期シリアル接続（TTL レベル）した場合の CPU ボード側のプログラムについて解説する。
- * CPU ボードと MMC/SD アクセスユニット SDC23 は非同期シリアルでクロス接続します。MMC/SD アクセスユニット SDC23 の通信速度は 38400、19200、9600bps から選択する（ユニット側の半田付けにより選択する、半田付けをしない場合は 38400bps が選択される）。・・・MMC/SD アクセスユニット SDC23 取扱説明書を参照

J3 3 ピンコネクタ用

| | |
|-----|---|
| RXD | 3 |
| GND | 2 |
| TXD | 1 |

(数字はピン番号)

J2 20 ピンコネクタ

| | |
|-----|------------------------|
| GND | 2,4,6,8,10,12,14,16,18 |
| RXD | 15 |
| TXD | 17 |

(数字はピン番号)

SDC23 の通信速度選択

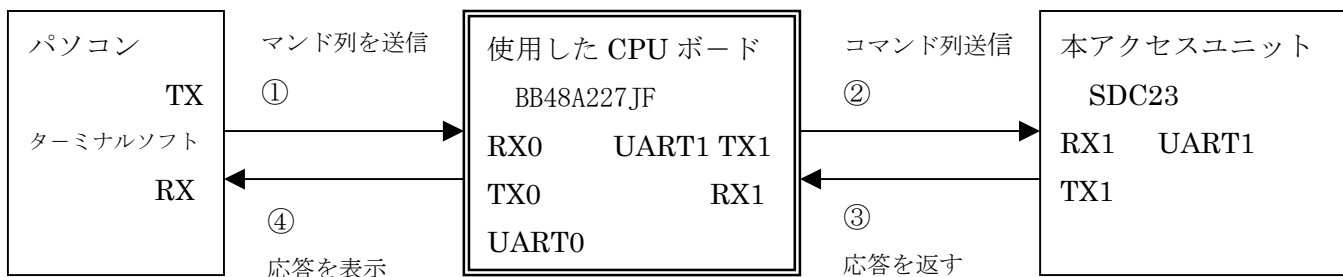
| | | |
|-----------|-----------|----------|
| JP2(P0_7) | JP1(P0_6) | |
| ショート | オープン | 19200bps |
| オープン | ショート | 9600bps |
| 上記以外 | | 38400bps |

* J3 コネクタは未実装

(ショートは半田付けで行なう)

デモプログラムの動作概要

デモプログラムを搭載



- * デモプログラムを CPU ボード（BB48A227JF）で動作させ、パソコンのターミナルソフト（ハイパーターミナルなど）起動させる。CPU ボード（BB48A227JF）はパソコンのキーボードから打ったコマンド列を受信すると、コマンド名やパラメータ数、パラメータのタイプなどをチェックする。チェックの後、コマンド列を本アクセスユニットに送信すると、本アクセスユニットからの応答を得る。CPU ボード（BB48A227JF）はその応答に対応したメッセージをパソコンに送信する。パソコンのターミナルソフトがこのメッセージを表示する。
- * コマンド列のチェックは本アクセスユニット側でも行っている。しかし、コマンド列の記述が正しくないと有効とならないため、本デモプログラムでは、コマンド列を大まかにチェックした後で、本アクセスユニットにコマンド列を送信している。
- * 本アクセスユニットはコマンド列を受けると（受信すると）、受けたコマンドに対応した処理を行なった後に応答を返します。しかし、1つのコマンド列の送信に対して1回の応答とは限らず、コマンドの種類より、CPU ボードとハンドシェイクして複数回に分けての応答を返します。詳細については添付 CD に収録したデモプログラムのソースおよび MMC/SD アクセスユニットの SDC23API コマンド解説.pdf をご覧下さい。

<CPU の種類により異なる処理>

- 1) ボードの初期化処理
- 2) シリアルポートの初期化処理

<CPU ボードの種類を問わない処理>

- 1) パソコンから送信されたコマンド列を受信する (UART0 からの 1 行入力)。
- 2) コマンド列をコマンド名とパラメータに切り分ける (コマンド列の切り分け)。
- 3) 切り分けた最初の文字列はコマンド名かどうか判定する。
- 4) コマンド名に対応した必要パラメータ数、数字か文字かなどのパラメータのタイプのチェックする (コマンド解析)。
- 5) コマンド列を MMC/SD アクセスユニットへ送信する (UART1 への 1 行出力)。
- 6) MMC/SD アクセスユニットからの応答を受信する (UART1 からの 1 行入力など)。
- 7) パソコンへの応答を送信する (UART0 への 1 行出力)。

デモプログラムの構成

デモプログラムは R8C Tiny シリーズの CPU を搭載した、弊社製の BB48S8C20 ボード (型名 BB48A227JF) 用のものです。

添付の CD には 2 つのデモのソースプログラムが以下のホルダに収納されています。

デモプログラム 1 DEMOPRG¥SRC¥SRC1
 デモプログラム 2 DEMOPRG¥SRC¥SRC2

<ヘッダファイル>

- *テストに使用した CPU における IO アドレスの定義 io_23.h
 - *配列や構造体の上限を記述 max_val.h
 - *簡易 printf 関数用のヘッダファイル hprint.h
 - *コマンド (コマンド名、パラメータ数、コマンド識別番号) の構造体定義 cmd_keytbl.h
 - *cmd_keytbl.h で記述したコマンド識別番号の定義 val_keytbl.h
- 注 : cmd_keytbl.h、val_keytbl.h はデモプログラム 2 で使用

<C 言語、アセンブラ言語のプログラムファイル>

- *デモプログラム 1 のメインプログラム testdemo1.c
 CPU 初期化、シリアル初期化、メインプログラム
 - *デモプログラム 2 のメインプログラム testdemo2.c
 CPU 初期化、シリアル初期化、メインプログラム
 - *割り込みベクタ定義、スタートアップルーチンの例 reset.a30
 - *簡易 printf 関数 hprint.c
 - *シリアル読み込み、シリアル書込みに関する入出力関数 debugsci.c
 - *コマンド・パラメータ列の分離および解析を行なう関数 cmd_analyze.c
- 注 : cmd_analyze.c はデモプログラム 2 で使用

<その他のファイル>

- *簡易 printf 関数 (hprint.c) の実装方法および使い方の説明 hprint_mem.txt
- * デモプログラム 1 コンパイル用のバッチファイル build.bat
- * デモプログラム 1 リンク用のコマンドファイル command.sub

注 : 以上のファイルのうち、アセンブラファイル reset.a30 は使用するコンパイラによって記述が異なる場合があります。テストに使用したコンパイラはルネサスのホームページサイトからダウンロードしたものです。コンパイラのバージョンを以下に示す。

M16C/60,/30,/20,/10,/Tiny,R8C/Tiny Series Compiler V.5.44 Release 00

Copyright (C) 2001,2008 Renesas Technology Corp.

and Renesas Solutions Corp. All rights reserved.

デモプログラムにおける RAM メモリ使用量

CD に収録したデモプログラム 2 では、DATA 領域で約 730 バイトの RAM メモリを使用しています。スタック領域の使用量は動的に変化しますが、ファイル `hprint.c` に記述してある簡易 `printf` 関数を呼び出した時に最も多く使用する。その使用量は約 200 バイトとなります。スタック領域の使用量を少なくしたい場合には、この関数を用いないで同等の処理を行なう必要があります。

デモプログラムの説明

デモプログラム 1 は「ファイルを新しく作成して、そのファイルにデータを書込み、書き込んだデータを読み出す」というファイルアクセスの最も基本となる操作を行なうプログラム例である。

デモプログラム 2 は本アクセスユニットで用意してある各コマンドの使い方についてのプログラム例を示したものである。

デモプログラム 1 の説明を次に示す。デモプログラム 2 については CD に収録されているプログラムソースをご覧ください。

●デモプログラム 1 の説明

デモプログラム 1 では以下の 1) から 7) の処理を順に行い、それぞれのところで、CPU ボード (アプリケーション) 側から本アクセスユニット側へコマンド文字列を送信し、本アクセスユニット側からの応答コードを受信して、そのコードに応じた処理を行っている。

<デモプログラム 1 のメインプログラムにおけるコマンド送信内容>

- 1) ファイルオブジェクト 1 を用いて、書込みモードでオープンして TEST1.TXT という名前のファイルを作成する。....."FOPN 1 TEST1.TXT WA¥r"
- 2) ファイル TEST1.TXT へ予め用意したバッファのデータを 40 バイト書く。
....."FWRT 1 40¥r"
- 3) このファイルを閉じる。"FCLS 1¥r"
- 4) ファイル TEST1.TXT 読み込みモードでのオープンする。
....."FOPN 1 TEST1.TXT RX"
- 5) ファイル TEST1.TXT から 30 バイト読む "FRED 1 30¥r"
- 6) さらに 20 バイト読む "FRED 1 20¥r"
- 7) ファイルを閉じる "FCLS 1¥r"

上記の” “内の文字列はアクセスユニット SDC23 へ送信するコマンド文字列である。コマンド文字列を送信すると、アクセスユニット SDC23 から応答文字列を送信してくる。応答は 1 回とは限らず、その内容に応じたデータのやりとりをアクセスユニット SDC23 と行なう。このデモプログラムの例では、ファイル TEST1.TXT には 40 バイトしか書いていない (ファイルサイズが 40 バイト) ので、最初に 30 バイト読んだあとに 20 バイト読もうとしても、実際に読み出されるのは 10 バイトとなる。

なお、デモ用書き込みデータとして予め次ぎの `const char` 型のデータを用意した。

```
testdata[]="0123456789ABCDEFGHIJKLMN OPQRSTUVWXYZabcdefghijklmnopqrstu  
vwxyz";
```

デモプログラム 1 の実行結果とメインプログラムの内容を「●デモプログラム 1 の実行結果」に示す。

●デモプログラム 1 の実行結果

hit anykey

```
program start
オープン 正常終了
```

書き込みバイト数 = 0040

クローズ 正常終了

オープン 正常終了

```
読み出しバイト数 =0030
読み出しバイトデータ
0123456789ABCDEFGHIJKLMNQRST
```

```
読み出しバイト数 =0010
読み出しバイトデータ
UVWXYZabcd
```

クローズ 正常終了

```
program end
```

●デモプログラム 1 のメインプログラム

デモプログラム 1 のメインプログラム部分を次に示す。testdemo1.c の一部

```
void main( void )
{
    int i,j,k,len;

    init_io_data(); //ボードの初期化处理
    sci0_init() ; //シリアルポートの初期化处理 UART0
    sci_init() ; //シリアルポートの初期化处理 UART1

    Hprintf("¥r¥n hit anykey¥r¥n");
    sci0_read() ;// 1char get
    Hprintf("¥r¥n program start¥r¥n");

/* ファイルの作成する
   書き込みモードでオープン
   FOPN          ファイルをオープン
   パラメータ 1   オブジェクト番号  1 or 2 or 3
   パラメータ 2   ファイル名
   パラメータ 3   オ-プンモ-ド
*/

    strcpy(linebuf,"FOPN 1 TEST1.TXT WA");
    hprintf("%s¥r",linebuf) ;//コマンドをアクセスユニットに送信
                                // アクセスユニットからの応答を得る。
    i=getline(swork0) ; // i:応答文字列の長さ (<CR>コードも含む)
    if (strcmp(swork0,"00¥r")==0) { //正常の場合
        Hprintf("オープン 正常終了¥r¥n¥r¥n");
    }
}
```

```

    }
    else { //正常でない場合
        Hprintf("オープン エラー終了¥r¥n¥r¥n");
        goto exit;
    }
/* ファイル TEST1.TXT にテストデータ testdata を書く
FWRT   ファイルにデータを書き込みます
パラメータ 1       オブジェクト番号  1 or 2 or 3
パラメータ 2       書き込むバイト数
*/
    strcpy(linebuf,"FWRT 1 40") ;
    hprintf("%s¥r",linebuf) ; //コマンドをアクセスユニットに送信
                                // アクセスユニットからの応答を得る。
    i=getline(swork0)           ; // i:応答文字列の長さ (<CR>コードも含む)

    if ( swork0[0]=='N' &&  swork0[1]=='N'  && swork0[2]==13 ) { //正常 "NN¥r"
        strcpy(workbuf,testdata) ; // 予め送信するバッファにデータを入れる
                                // この例ではテストデータを使用
        put_nbyte(workbuf,1,40) ; // send 40 byte >> SDC23
        i=getline(swork0)         ; // 終了コード normal:"00¥r"
        if ( swork0[0]=='0' &&  swork0[1]=='0'  && swork0[2]==13 ) { // normal
            i=getline(swork0)     ; // 書き込みバイト数
            Hprintf("書き込みバイト数 = ") ; disp_out0(i,swork0)    ;
            Hprintf("¥r¥n");
        }
        else {
            Hprintf("エラーコード = ") ; disp_out0(i,swork0)    ;
            Hprintf("¥r¥n");
            goto exit1;
        }
    }
    else { //正常でない "NN¥r" 以外
        disp_out0(i,swork0)      ;
        goto exit;
    }
/*   ファイルをクローズする   */
    strcpy(linebuf,"FCLS 1") ;
    hprintf("%s¥r",linebuf) ; // コマンドをアクセスユニットに送信
                                // アクセスユニットからの応答を得る。
    i=getline(swork0)           ; // i:応答文字列の長さ (<CR>コードも含む)
    if ( swork0[0]=='0' &&  swork0[1]=='0'  && swork0[2]==13 ) { //"00¥r"
        Hprintf("クローズ 正常終了¥r¥n");
        Hprintf("¥r¥n");
    }
    else {
        Hprintf("クローズ エラー終了¥r¥n");
        Hprintf("¥r¥n");
        goto exit;
    }
}

/*   ファイル TEST1.TXT の内容を読む   */
// 既存のファイルを読み込みモードでオープン
    strcpy(linebuf,"FOPN 1 TEST1.TXT RX");
    hprintf("%s¥r",linebuf) ; //コマンドをアクセスユニットに送信
                                // アクセスユニットからの応答を得る。

```

```

i=getline(swork0)      ;// i:応答文字列の長さ (<CR>コードも含む)
if (strcmp(swork0,"00¥r")==0) {
    Hprintf("オープン 正常終了¥r¥n");
    Hprintf("¥r¥n");
}
else {
    Hprintf("オープン エラー終了¥r¥n");
    Hprintf("¥r¥n");
    goto exit;
}
/* ファイル TEST1.TXT の内容を読む */
//FRED   ファイルからデータを読み出し
//パラメータ 1      オブジェクト番号   1 or 2 or 3
//パラメータ 2      読み出すバイト数

strcpy(linebuf,"FRED 1 30");
hprintf("%s¥r",linebuf); //コマンドをアクセスユニットに送信
                        // アクセスユニットからの応答を得る。
i=getline(swork0)      ;// i:応答文字列の長さ (<CR>コードも含む)
if (i>3) { // 正常 4桁の 10進数
    swork0[4]=0        ;// cut <CR> toru バイトカウント get
    len=ad_toi(swork0) ;
    getline(swork0)    ;// <CR>までのコードを入力
        if ( strcmp(swork0,"nnnnn",5)==0) {"nnnnn"<CR>
            sci_write('*') ;// 1文字送信 put UART1 同期をとる
        }
    i=0;
    if (len>0) i=get_nbyte(workbuf,1,len) ;//len バイト読み出し

    Hprintf("読み出しバイト数 =%04d¥r¥n",i) ;//i:読み出しバイト数
    if (i > 0) {
        Hprintf("読み出しバイトデータ¥r¥n");

        for (k=0;k<i;k++) {
            if (isprint((int) workbuf[k])){//プリント文字
                sci0_write(workbuf[k]); //disp read data
            }
            else sci0_write('.');//非プリント文字
        }
        Hprintf("¥r¥n");
        Hprintf("¥r¥n");
    }
}
else {
    Hprintf("ファイル読み出し エラー終了 ") ;
    disp_out0(i,swork0) ;
    Hprintf("¥r¥n");
    goto exit1;
}
/* さらに 20 バイト読む */
// この例では、ファイルを作成した時 40 バイトしか書いていないので、
// 先に 30 バイトを読んでいるので、20 バイトを指定しても、読み出せるのは
// 残りの 10 バイトである。
strcpy(linebuf,"FRED 1 20");
hprintf("%s¥r",linebuf); //コマンドをアクセスユニットに送信

```

```

// アクセスユニットからの応答を得る。
i=getline(swork0) ;// i:応答文字列の長さ (<CR>コードも含む)
if (i>3) { // 正常 4桁の10進数
    swork0[4]=0 ;// cut <cr> toru バイトカウント get
    len=ad_toi(swork0) ;
    getline(swork0) ;
    if ( strncmp(swork0,"nnnnn",5)==0 ) {
        sci_write('*') ;// 1文字送信 put UART1 同期をとる
    }
    i=0;
    if (len>0) i=get_nbyte(workbuf,1,len) ;
    Hprintf("読み出しバイト数 =%04d¥r¥n",i);
    if (i > 0) {
        Hprintf("読み出しバイトデータ¥r¥n");

        for (k=0;k<i;k++) {
            if (isprint((int) workbuf[k])){//プリント文字
                sci0_write(workbuf[k]); //disp read data
            }
            else sci0_write('.');//非プリント文字
        }
        Hprintf("¥r¥n");
        Hprintf("¥r¥n");
    }
}
else {
    Hprintf("ファイル読み出し エラー終了¥r¥n") ;
    disp_out0(i,swork0) ;
    Hprintf("¥r¥n");
}
exit1:
/* ファイルをクローズする */
strcpy(linebuf,"FCLS 1") ;
hprintf("%s¥r",linebuf); // コマンドをアクセスユニットに送信
// アクセスユニットからの応答を得る。
i=getline(swork0) ;// i:応答文字列の長さ (<CR>コードも含む)
if ( swork0[0]=='0' && swork0[1]=='0' && swork0[2]==13 ) { //"00¥r"
    Hprintf("クローズ 正常終了¥r¥n");
}
else {
    Hprintf("クローズ エラー終了¥r¥n");
}
exit:
Hprintf("¥r¥n program end¥r¥n");

return ;
}

```

●デモプログラムにおける主要な処理内容

以下にデモプログラム 1 およびデモプログラム 2 における主要な処理内容を説明する。

○ポートの初期化 (デモプログラム 1、デモプログラム 2 で使用)

```
//■void init_io_data(void)      (testdemo1.c 、 testdemo2.c に記述)
void init_io_data(void)
{
    /* ポート Pi 方向レジスタ */
    *PD1 = 0xdf;    // bit5:RXD0 in(0)    bit4:TXD0 out(1)
                  // other bit    out(1)
    *PD6 = 0x7f;    // bit7:RXD1 in(0) bit6:TXD1 out(1)
                  // other bit    out(1)
    *PUR0=0x08;    //P1_4 ~P1_7 のプルアップ
    *P1 = 0xff;
    *P6 = 0xff;
}

```

○シリアルポートの初期化 (デモプログラム 1、デモプログラム 2 で使用)

使用している 2 つのシリアルポートの初期化を行います。

```
//■void sci0_init(void)        (testdemo1.c 、 testdemo2.c に記述)
void sci0_init(void) //UART0
{ volatile int i ;
    *U0MR = 0x05;    //8 ビット ノンパリティ 1ストップビット 内部クロック使用
                  /*パリティなし(b6=0)、内部クロック使用(b3=0) */
    *U0C0 = 0x00;    //クロック=f1(b1=0,b0=0)、TxD=CMOS 出力(b5=0)
    // *U0BRG = 0x81; /*転送レートを 9600bps(20MHz 時)に設定*/
    // *U0BRG = 0x7f; /*転送レートを 9600bps(19.6608MHz 時)に設定*/
    // *U0BRG = 0x40; /*転送レートを 19200bps(20MHz 時)に設定*/
    // *U0BRG = 0x3f; /*転送レートを 19200bps(19.6608MHz 時)に設定*/
    *U0BRG = 0x1f;  /*転送レートを 38400bps(19.6608MHz 時)に設定*/
    for(i=0;i<10000; i++);
    *U0C1 = 0x05;    /*送受信を許可 RE(b2=1) TE(b0=1) */
    for(i=0;i<10000; i++);
}

```

```
//■void sci0_init(void)        (testdemo1.c 、 testdemo2.c に記述)
void sci_init(void) // UART1
{
    volatile int i ;
    *U1MR = 0x05; ; //8 ビット ノンパリティ 1ストップビット 内部クロック使用
    *U1C0 = 0x00;    //クロック=f1(b1=0,b0=0)、TxD=CMOS 出力(b5=0)
    // *U1BRG = 0x81; /*転送レートを 9600bps(20MHz 時)に設定*/
    // *U1BRG = 0x7f; /*転送レートを 9600bps(19.6608MHz 時)に設定*/
    // *U1BRG = 0x40; /*転送レートを 19200bps(20MHz 時)に設定*/
    // *U1BRG = 0x3f; /*転送レートを 19200bps(19.6608MHz 時)に設定*/
    *U1BRG = 0x1f;  /*転送レートを 38400bps(19.6608MHz 時)に設定*/
    for(i=0;i<10000; i++);
    *U1SR = 0x03; //UART1 を使用する場合 0x03
    *PMR = 0x10; //ポート CLK1/TXD1/RXD1 切り替え P6_6:TXD1 P6_7:RXD1
    *U1C1 = 0x05; //UART1 の送受信を許可 RE(b2=1) TE(b0=1)
    for(i=0;i<10000; i++);
}

```

UART0(パソコン)からの1行入力 (デモプログラム2で使用)

パソコンのキーボードで打たれた文字をバッファに格納する (指定した長さあるいは制限長さまで)。

CR (0x0D) あるいは LF(0x0A)を受信すると NULL 文字に変えて処理を終了する。英小文字は大文字に変換される。

戻り値は 入力された文字数 (CR コード、LF コードを除く)
指定長さを超えた場合はエラー (999) とする。

```
/* 通信文字の取得とエコーバック 最大 MAXLEN 文字*/
int get_nchar0(unsigned char *buf,int len) // UART0
// (testdemo2.c で使用、cmd_analyze.c に記述)
{//bufの大きさは (MAXLEN+1) あるいは (len+1) 以上を必要とする。
    int j;
    unsigned int k;
    j=0;
    while(1) {
        k=(unsigned int)sci0_read(); /* UART0 1 char get */
        if (k >= 'a' && k <= 'z') { /* to upper case 大文字に変換*/
            k -= 0x20;
        }
        if ((k=='\r') || (k=='\n')) { // CR または LF コードで終わりを判定
            break;
        }
        else if (k==0x08) { // バックスペース処理
            if (j==0) continue;
            j--;
            sci0_write(0x08);
            sci0_write(' ');
            sci0_write(0x08);
            continue;
        }
        else {
            buf[j]=(k & 0xff); // 受信データをバッファにセット
            sci0_write(k & 0xff); //エコーバック
            j++; //格納ポインタインクリメント
            if ((j>MAXLEN) || (j>len)) { //制限長さを超えた場合はエラー終了
                return 999;
            }
        }
    }
    buf[j] = 0x00; // 終端コード CR または LF は NULL に変えて格納します
    sci0_write('\r');
    sci0_write('\n');
    return j;
}
```

○入力コマンド列の切り分け (デモプログラム2で使用)

入力コマンド列をスペース区切りで切り分け、各パラメタの文字列とパラメータ数を得る。

```

/* int cmd_anal(char *line,char**param) */
/* line :分離対象の文字列 param[0]..param[n]:分離された文字列 */
/* 最大分離数は #define MAXPARAM1 xx 等で予め定義しておく max_val.h */
/* 文字列 line をスペース' 'を区切りとして分ける */
/* 文字に続くスペース' 'は¥0に変わる */
/* ORIGINAL の文字列を残す場合には コピーしてから呼び出す。 */
/* 戻り値 切り分けられた文字列の個数 */
// (testdemo2.c で使用、cmd_analyze.c に記述)

```

```

int cmd_anal(char *line,char**param)
{
    int linepos,pcnt,cnt_flag,c;
    linepos=0;
    pcnt=0;param[0]=0;
    cnt_flag=1; /* カウントフラグを ON にする */
    while (1) {
        c = line[linepos++] ; /* 文字列の頭から順に文字を取りだしポインタを進める*/
        if (c==0x00) return 0; /* line の長さが 0 or スペースだけ */
        if (c==' ') continue ; /* 最初のスペースはカット*/
        else break ;
    }
    linepos-- ; /* ''以外が見つかったのでポインタを1つ戻す */
    param[0]=&line[linepos] ; /* この時の文字位置のアドレスを第一パラメータのポインタとする */
    while (line[linepos] !=0) { /* 文字列の終端に達するまで解析を進める */

        if(line[linepos]==' ') { /* パラメータの後ろの' 'は ¥0 に */
            line[linepos]=0;
            pcnt++; /* パラメータカウンタのインクリメント*/
            if (pcnt>=MAXPARAM1) { /* 最大分離数に達した場合はここで打ち切り*/
                cnt_flag=0; /* カウントフラグを OFF にする */
                break ;
            }
            linepos++; /* ポインタを進める */
            while(line[linepos]==' ') {
                linepos++; /* ''が続く場合文字列のポインタを進める*/
            }
            /* ''以外の文字がきてループを抜ける */
            if (line[linepos] ==0) {
                /* 文字列の終端を見つけたならここで打ち切り*/
                cnt_flag=0; /* カウントフラグを OFF にする */
                break ;
            }
            else { /* 次のパラメータの最初の文字を検出 */
                param[pcnt]=&line[linepos];
                /* この時の文字位置のアドレスをパラメータのポインタとする*/
                cnt_flag=0; /* カウントフラグを OFF にする */
            }
        }
        else { /* ''以外 (文字 or ¥0) がくる場合ポインタを進める*/
            linepos++;
        }
    }
}

```

```
        cnt_flag=1;    /* カウントフラグを ON にする */
    }
}
if(cnt_flag==1)      /* カウントフラグを ON の時 */
    pcnt++;
return pcnt;
}
```

例1 line="ABCD 123 XYZ "; として
この関数 int cmd_anal(char *line,char**pram) を実行すると
戻り値は 切り分けた個数 3 となり
pram [0] は"ABCD" , pram [1] は"123" pram [2] は"XYZ" へのポイン
タとなる。

○コマンド列のパラメータチェック (デモプログラム2で使用)

最初のパラメータ文字列であるコマンド名がコマンドテーブル (添付 CD に収録したヘッダファイル cmd_keytbl.h) に登録されているか否かをチェックする。

コマンドテーブルは、次の形式の構造体の配列で定義

```
struct keytbl { // val_keytbl.h の中で記述
    const char keyp[6] ; //コマンド名
    short pcnt      ; // 必要とするパラメータ数 (コマンド名を除く)
    short cmdval    ; // コマンド名の識別番号
};
```

```
const struct keytbl kresvtbl [] = {
"DMY0" , 0 , LDUMY0 , // 0
"DMY1" , 0 , LDUMY1 , // 1
"MDIR" , 1 , LMKDIR , // 2
"ODIR" , 1 , LOPDIR , // 3
    途中省略
"HELP" , 0 , LHELP , //45
"PEND" , 99 , LPEND
}
```

/* 対象文字列 str が予約テーブル (コマンドテーブル) に登録されていればその添え字の番号を返す。登録されていなければ 99 を返す 得られた添え字の番号から*/

// (testdemo2.c で使用、cmd_analyze.c に記述)

```
int key_check(char *str)
{
    int j;
    j=1;
    while(1) {
        if (strcmp(str,kresvtbl[j].keyp)==0) { //コマンド名が一致した
            break ;
        }
        j++;
        if (j>KEY_MAX) return 99; // コマンドテーブルにはない
    }
    return j; //コマンドテーブルの j 番目で一致した (0 番目が最初)
}
```

例 1 : “MDIR NEWDIR”<CR>

(NEWDIR という名前のディレクトリを作成するコマンド列)

この文字列がパソコンのキーボード入力から入力し、CPU ボードが受信して、パラメータ解析 (関数 int cmd_anal(char *line,char**pram) を実行) すると、

param[0]は”MDIR” param [1] は”NEWDIR” となり 戻り値のパラメータ数は2 (コマンド名も含めて) となる。この後、関数 int key_check(char *str)の引数 str にコマンド名の param[0] : ”MDIR”を与えて実行すると、コマンドテーブルの中を検索して配列の添え字の値 2 を返す。この添え字の値から必要とするパラメータ数 (コマンド名を除く) 1 とコマンド名の識別番号 LMKDIR (2) を得る。

この後は、CPU ボードから本アクセスユニットへコマンド列を送信して、本アクセスユニットからの応答を待にはいり、応答コードを受けてそのコードに応じた処理を行なう。この例では正常動作の場合だと、文字列“MDIR NEWDIR”<CR> を本アクセスユニットに送信し、”00”<CR> (正常の場合) という応答文字列を本アクセスユニットから受け取ります。

○コマンド列の送信（デモプログラム1、デモプログラム2で使用）

本アクセスユニットへ param[1] のコマンド列の送信形式（送信文字列）は
関数 int cmd_anal(char *line, char**param) を実行した場合
必要パラメータ数が3個の場合 “コマンド文字列 param[1] param[2] param[3]<CR>
必要パラメータなしの場合 “コマンド文字列”<CR>

<CR> : 16進のコード 0x0D

となる。

コマンド文字列以外のパラメータの数は0個（なし）から3個まで、コマンド文字列は param[0] としている。デモプログラム用に出力先を UART1、UART0 とする printf 形式の関数（簡易 printf）を作成した（hprint.c, hprint.h, hprint_mem.txt）。

デモプログラムでは、この関数を用いて本アクセスユニットへコマンド列を送信した。

int hprintf(const char *fmt, ...); // UART1 本アクセスユニットへコマンド列送信

< パソコン側へは int Hprintf(const char *fmt, ...); // UART0 >

CPU ボードから本アクセスユニットへのコマンド列送信の例：

“MDIR NEWDIR”<CR>の場合 param[0]は”MDIR”、param[1]は”NEWDIR”

となり、下記の関数を用いてコマンド列の送信を行なう。

hprintf(“%s %s¥r”, param[0], param[1]); // 本アクセスユニットへのコマンド列送信

○応答文字列の受信（デモプログラム1、デモプログラム2で使用）

コマンド列を送信すると、応答文字列が本アクセスユニットから’¥r’（0x0D）を終端とする文字列が応答します。CPU ボードは受信用のバッファを指定して’¥r’（0x0D）までの受信文字列をバッファに格納する。

// (testdemo1.c、testdemo2.c に記述)

```
int getline(char *buf)
{
    int c;    int i;
    i=*U1RB  ;// DUMMY READ
    i= 0;
    buf[0]=0;
    while (1) {
        c= sci_read();
        buf[i]=c;
        i++;
        if ((c=='¥r') || (c=='¥n')) {
            //本アクセスユニットからは¥r (0x0D)
            break;
        }
        if (i>=MAXREAD_COUNT) {
            break;
        }
    }
    buf[i] = 0x00;
    return i;
}
```

○パソコンへの応答内容送信（デモプログラム 1、デモプログラム 2 で使用）

CPU ボードは受信した本アクセスユニットからの応答コードの内容に応じて、次に示す関数のいずれかあるいは組み合わせてパソコンへ文字列を出力します。

// (testdemo1.c 、testdemo2.c で使用)

// (hprint.c に記述)

int Hprintf(const char *fmt, ...) ; // UART0 へ書式付き出力

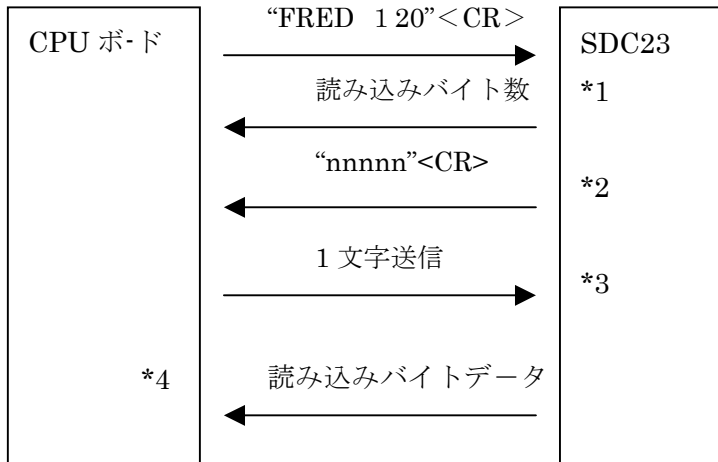
void sci0_write(unsigned char str) ; // UART0 へ 1 文字出力

int disp_out0(int num ,char *str) ; // UART0 へ文字列 str の str[0]から num バイト出力

コマンド/応答処理の例

1) ファイルのリード (FRED コマンド)

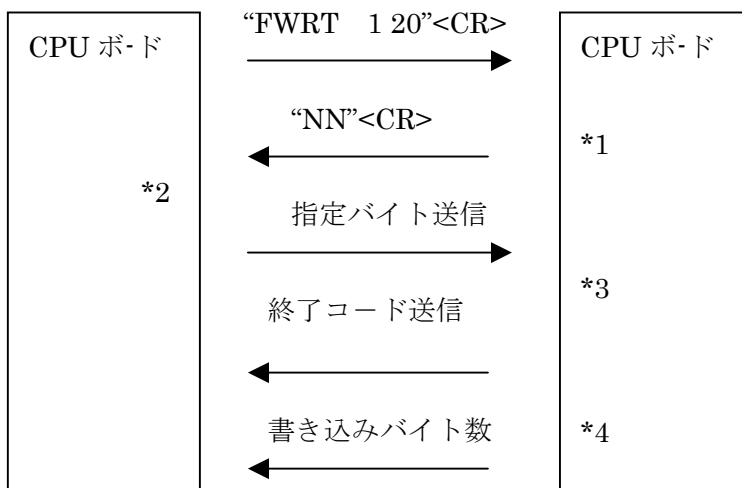
コマンド形式: "FRED 1 20" <CR> 、"FRED 2 40H" <CR> など



- *1: パラメータチェック、ファイル読み込み正常の場合: 4桁10進文字列+<CR>の読み込みバイト数送信
エラーの場合: 2桁の16進文字列+<CR>送信して終了
- *2: 同期用の応答"nnnn"<CR>送信
- *3: 読み込む準備ができたなら1文字送信
- *4: 読み込んだバイト数分のデータを送信

2) ファイルのライト (FWRT コマンド)

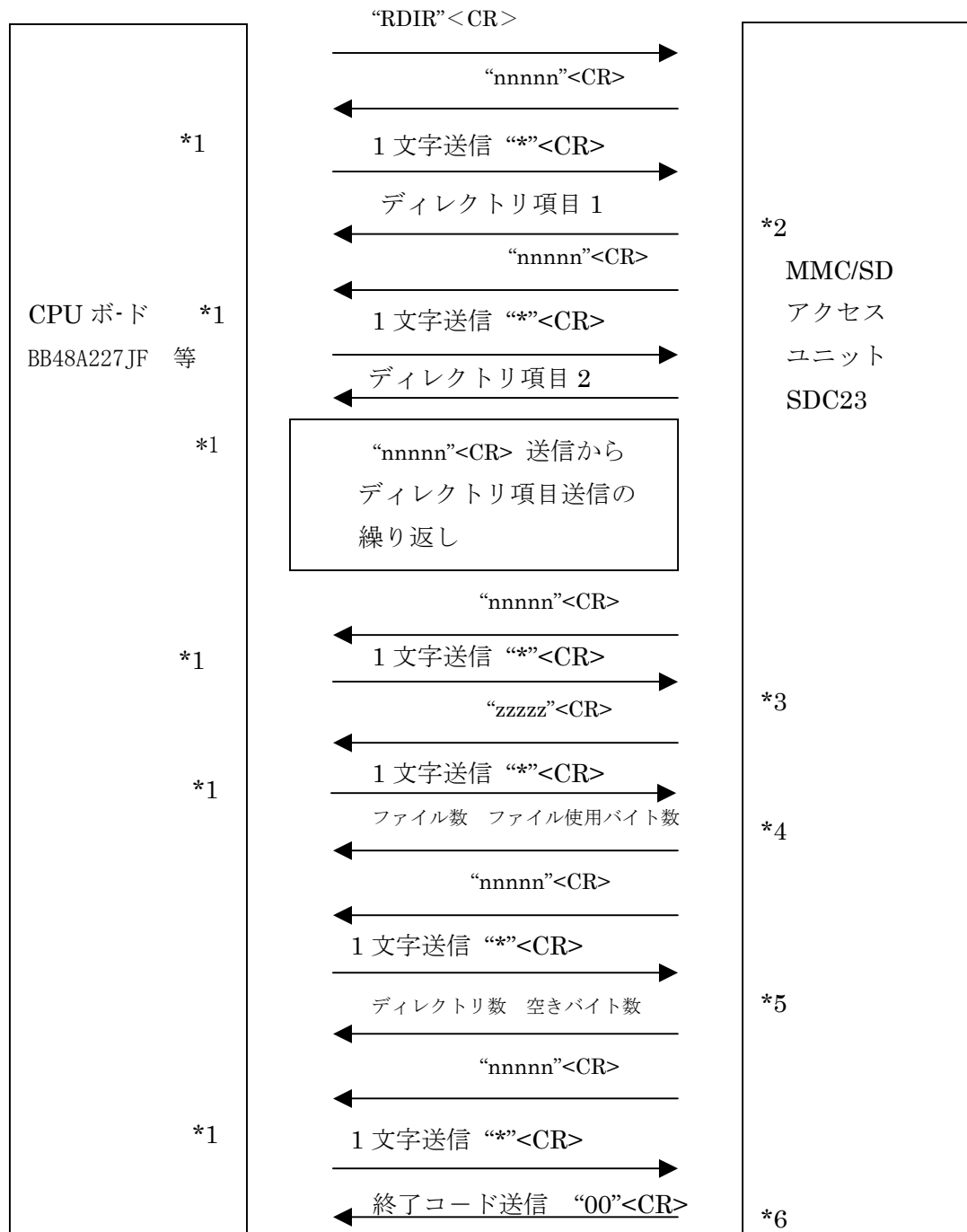
コマンド形式: "FWRT 1 20" <CR> 、"FWRT 2 40H" <CR> など



- *1: パラメータチェックがOKの場合 "NN"<CR>を送信、OKでない場合は16進2桁のエラーコード+<CR> ("AA"<CR>など)を返して処理を終了する。
- *2: 指定したバイト数分のデータを送信 (この例では20バイト)
- *3: 終了コードは正常の場合は"00"<CR>、これ以外はエラー
- *4: 実際に書き込んだバイト数を送信 (10進4桁の文字+<CR>) エラーの時は"0000"<CR>

3) ディレクトリ項目の読みだし (RDIR コマンド)

コマンド形式：“RDIR”<CR>、“RDIR /”<CR>、“RDIR /DIR1/SUBDIR”<CR> など
 RDIR コマンドはパラメータ 0 個の場合は対象とするディレクトリはルートディレクトリです。読み出すディレクトリ項目数は不定のため、データの受け渡しを確認しながら項目を読み出し、全部読んだ後に対象ディレクトリにおけるファイルの数とその使用バイト数、ディレクトリの数、全体の空きバイト数を読みだす処理を行う。



*1: CPU ボード側は SDC23 側から”nnnnn”<CR> あるいは “zzzzz”<CR>のデータを受けた場合に 1 文字送信する。SDC23 側はこの 1 文字を受けてから次の応答データを送る。SDC23 側は CPU ボード側から’0’から’9’の数字の文字データを受けた場合は途中の処理をジャンプして*6の終了コード送信を行い、処理を終える。

*2: ディレクトリ項目は SDC23 がスペースで区切られた次の形式 (属性 日付 時間 サイズ 名前<CR>) で送信する。

Ex “----A 2008/05/14 03:23 13200 TEST1.TXT “<CR>

*3: 指定ディレクトリにディレクトリ項目がない (空) かさらなるディレクトリ項目がない場合には、SDC23 は 6 文字のコード“zzzzz”<CR>を送信する。

*4: CPU ボードが、上記のコードを受信した後、SDC23 に 1 文字送信すると、SDC23 は指定ディレクトリにおける 10 進 4 桁のファイルの数とそれらのファイルが使用している 10 進 10 桁のバイトサイズを次の形式で送信する。

Ex “0004 0000013292”<CR> その後、”nnnnn”<CR>を送信する。

*5 CPU ボードから SDC23 が 1 文字受信すると SDC23 は、10 進 4 桁の指定ディレクトリにおけるディレクトリ数と 10 進 10 桁の空き容量を送信する。

Ex “0002 0127483904”<CR> さらに ”nnnnn”<CR>を送信する。

*6: SDC23 は 1 文字を受信した後、16 進 2 桁の終了コードを送信して処理を終了する。

* SDC23 は DIR1 コマンド (最初のディレクトリ項目の読み出し)、DIRN (次ぎのディレクトリ項目の読み出し) も類似の処理をしている (これらのコマンドは上記に示した *4、*5 の処理は行なわない。)

DIR1 コマンドでは指定ディレクトリにおける最初のディレクトリ項目を読み出す。

DIRN コマンドでは DIR1 コマンドで指定したディレクトリにおけるにおいて 2 つ目以降のディレクトリ項目を読み出す。

終わりに

本アクセスユニットを用いて実際のアプリケーションプログラムを作成する場合は、CD に収録したプログラムソースと、SDC23API コマンド解説.pdf を参照して下さい。なお、このプログラムソースは弊社 R8Ctiny Base Board シリーズ BB48S8C20 ボード型名 BB48A227JF 用に作成したものでありますが、CPU ボードの初期化、シリアルインタフェースの初期化処理についての修正を加える事により、他の CPU ボードでも容易に動作させることが可能です。


MMC/SD アクセスユニット デモプログラム解説

©2008-2009 北斗電子 Printed in Japan 2008 年 12 月 11 日初版発行 REV.1.0.1.0 (090324)

発行 株式会社  URL:<http://www.hokutodenshi.co.jp>

お問い合わせは e-mail: support@hokutodenshi.co.jp(サポート用) order@hokutodenshi.co.jp(ご注文用)

TEL 011-640-8800 FAX 011-640-8801 〒060-0042 札幌市中央区大通西 16 丁目 3 番地 7

19 株式会社  MMC/SD アクセスユニット デモプログラム解説書