管理番号	C06621A-Z080B

アプリケーションノート

品　名 Graphic Operation Panel

型　式 GOP-4000/5000 シリーズ

製品改良の為、予告無く記載内容を変更する場合があります。最終設計に際しましては、納入仕様書をお取り寄せくださいます様、お願いします。

初版作成日	本書作成日	デバイス製造1課			品質保証	
		承認	確認	担当	承認	確認
2008 年 7 月 30 日	2014 年 8 月 12 日	藤岡	藤本	島田		
備考						

株式会社 石井表記

管理番号	C06621A-Z080B

改定履歴表

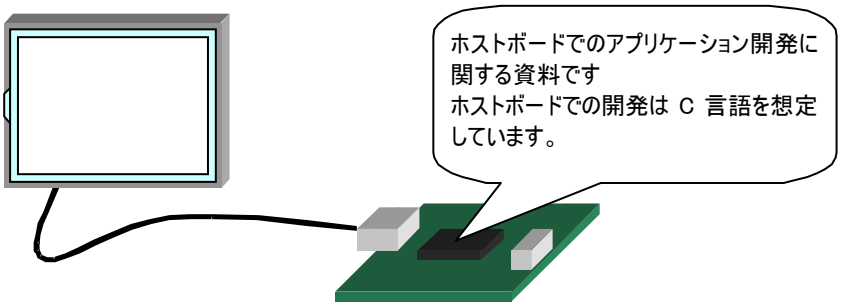
改定番号	改定年月日	改定内容	担当	承認
ー	2008 年 7 月 30 日	初版	島田	藤井(隆)
A	2012 年 8 月 21 日	Goplib サンプル追加 GOP5000 を対応機種に追加	島田	藤岡
B	2014 年 8 月 12 日	Goplib、ファイル転送コマンド追加およびサンプル追加 サンプルの動作対象から VisualStudio6 を除外	島田	藤岡

備考

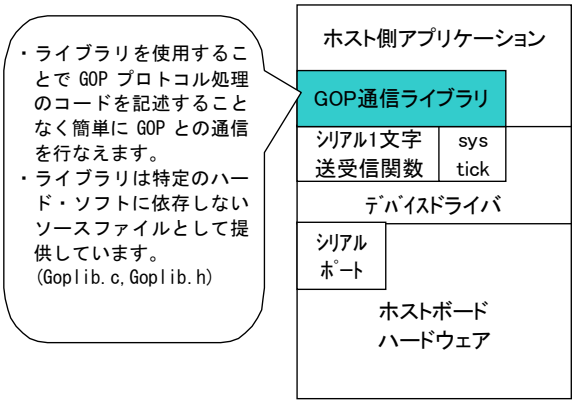
目次

1.本書について

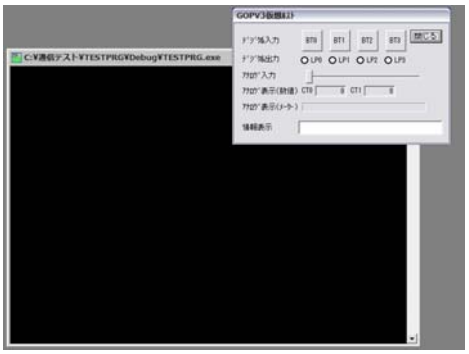
本書では 2 章で GOP-4000/5000 シリーズとホストマイコンを使用してシステムを構築する場合のホスト側ソフトウェアの開発に関する事例およびサンプルソースを記述しています。
3 章ではトレンドグラフの使用法について、マクロから使用する場合とホストから使用する場合の使用方法を記述しています。



ホスト側アプリケーションを作成するために、GOP との通信処理を行うための関数群(GOP 通信ライブラリ)を用意しており、本書ではそれを使用しての説明となっております。



本書では手軽に GOP との通信を行うホストアプリケーションをご評価いただけるけるよう PC(Windows)上で動作する仮想ホストを用意しております。



仮想ホストは VisualStudio C++(2008 以降 ExpressEdtion でも可)にてビルドすることが出来ます。

2.ホストアプリケーション構築例

2. 1 サンプルのファイル構成

フォルダ中に含まれるファイルは以下のとおりです

```
GOP4000 HOST 編サンプルソース
 │ APP1.et4 サンプル 1 用の GOP 画面データ
 │ APP2.et4 サンプル 2 用の GOP 画面データ
 │   :
 │ APP10.et4 サンプル 10 用の GOP 画面データ
 │ TREND.et4 トレンドサンプル用の GOP 画面データ
 └host_source
      │  Goplib.c     GOP 通信用のライブラリソース
      │  Goplib.h     GOP 通信用のライブラのヘッダ
      │  V3AppNote.sln   VC2008 用のソリューション
      │
      ├─APP1          サンプルアプリ 1 用のフォルダ
      │     APP1.c          サンプルアプリ 1 用の C ソース
      │     APP1.vcproj     VC2008 用プロジェクトファイル
      │
      ├─APP2          サンプルアプリ 2 用のフォルダ
      │     APP2.c          サンプルアプリ 2 用の C ソース
      │     APP2.vcproj     VC2008 用プロジェクトファイル
      │      :
      │      :
      ├─APP9          サンプルアプリ 9 用のフォルダ
      │     APP9.c          サンプルアプリ 9 用の C ソース
      │     APP9.vcproj     VC2008 用プロジェクトファイル
      ├─APP10          サンプルアプリ 9 用のフォルダ
      │     APP10.c          サンプルアプリ 9 用の C ソース
      │     APP10.vcproj     VC2008 用プロジェクトファイル
      │     PIC
      │       ├B000.JPG       サンプル用の画像ファイル
      │       │  :
      │       └B003.JPG
      │
      ├─TREND         トレンドサンプル用のフォルダ
      │     TREND.c          トレンドサンプル用の C ソース
      │     TREND.vcproj     VC2008 用プロジェクトファイル
      │
      └─dummysys     Windows 上での実習用仮想ホストソース
            dummyhard.c      仮想ホスト提供機能の実装
            dummyhard.h      仮想ホスト提供機能のヘッダ
            icon1.ico        アイコンリソース
            main.c           仮想ホストのメイン
            resource.h       仮想ホストリソースヘッダ
            ring.c           通信リングバッファ実装
            ring.h           通信リングバッファヘッダ
            sci.c            仮想ホストの通信機能の実装
            sci.h            仮想ホストの通信機能のヘッダ
            Script1.rc       仮想ホストのリソース
```

上記中の Goplib.c が GOP 通信ライブラリのソースになります。このファイルをお客様のホストのプロジェクトに読込み、ホストのソース上で Goplib.h をインクルードすることで GOP 通信ライブラリを使用出来ます。

2. 2 GOP通信ライブラリについて

GOPと通信するホストアプリケーションの作成する時に、GOPとの通信コマンド処理を簡素化するために通信処理用のコードをまとめたライブラリソースを用意しています。(以下 GopLib)
上記ライブラリソースを作成するアプリケーションのプロジェクトに追加し、下項で説明する API を使用することにより簡単に GOP を使ったシステムを作成出来ます。

尚、GOP 通信ライブラリの使用にあたり、GOP 通信ライブラリについては石井表記として動作の保証･修正の義務、動作不具合による損害の補償等は負わないものとしますので、動作、内容等十分検証した上、使用者側の責任においてご使用下さい。

2. 2. 1 使用方法

(1)ホスト側であらかじめ用意する必要のあるAPI

GopLib の使用にあたり、下記仕様に準じた関数をあらかじめホスト側で準備しておく必要があります。

シリアル 1 文字受信関数
定義
int <GETCHAR>(void)
仕様
シリアル受信バッファから 1 文字を受け取ります。
受け取るべきデータがない場合は EOF(-1)を返します。
実装方法は受信割り込みにより受信バッファにバッファリングする必要があります。
関数実行時、直接ポートを見に行く実装の場合、受信データの取りこぼしが発生する可能性があり正常に動作しません。

シリアル 1 文字送信関数
定義
void <PUTCHAR>(char)
仕様
1 文字をシリアルポートへ出力します。
実装方法は送信バッファを介しても、直接ポートに出力してもどちらでもかまいません。

経過時間取得関数
定義
int <TIMECOUNT>()
仕様
タイマ割り込み等で一定間隔でカウントアップを行い、経過時間を ms 単位に換算した値を返します。
最小間隔については特に規定しません。

※<GETCHAR><PUTCHAR><TIMECOUNT>の名称については任意で可です。

また ANSI 標準ライブラリとして以下のヘッダーファイルを使用します。
strncpy.h
strncat.h
strncmp.h
strlen.h
math.h ※GOPLIB.c の#define USEATOF をコメントアウトすると不要です。
その場合 GOP のメモリリンク出力で F 型メモリが処理が出来ません。

(2)設定パラメーター

ホスト側の環境により使用メモリ容量調整のためいくつかのパラメータを調整出来ます。

GopLib.h 内の define 定義で以下のパラメータを設定します。

LINEBUFSIZ	1 行として受け取る文字の最大長さを指定します。
TEXTDATAMAXLEN	T 型メモリの値として設定される最大の文字列長を設定します。
MSGMAXLEN	通信出力で出力される最大の文字列長を指定します。
TIMEOUT	応答確認あり時のタイムアウト時間を設定します。

2.2.2 API説明

(1) GopInitLib

定義
void GopInitLib(int (*getcharctor)(),void (*putcharctor)(char),int (*timecount)());
動作
GopLib の初期化を行ないます。
引数
getcharctor：ホスト側で用意したシリアル１文字受信関数のアドレス。
putcharctor：ホスト側で用意したシリアル１文字送信関数のアドレス。
timecount：ホスト側で用意した経過時間取得関数のアドレス。

(2) GopSendCommand

定義
void GopSendCommand(char *msg);
動作
msg で指定された文字列を stx,etx を付加し GOP へ送信します。
引数
msg：送信する文字列。

(3) GopWrite_?

定義
void GopWrite_?(short addr,<type：?に対応したメモリタイプ> value);
?には b,B,w,W,l,L,F,T の内いずれかを指定します。
type には上記に型に対応し unsigned char,char,unsigned short,short,
unsigned long,long,float,char *となります。
※GopWrite_?はマクロ定義です。
動作
指定のアドレスに指定の型でデータを書込みます。
引数
addr：書込み先の GOP のアドレス。
value：書込むデータ。
使用例
GopWrite_w(0x0000,100);
w0000 に 100 を書込みます。

(4) GopRead_?

定義
int GopRead_?(short addr, <type：?に対応したメモリタイプ>*buf,int timeout)
?については GopWrite_?と同様です。
※GopRead_?はマクロ定義です。
動作
指定のアドレスのデータを読込み buf で指定される領域に記憶します。
引数
addr：読込先の GOP のアドレス。
buf：読込んだデータを格納する領域へのポインタ。
timeout：応答タイムアウト時間の設定。
戻り値
1：読込み成功。
0：読込み失敗。
使用例
unsigned short val;
GopRead_w(0x0000,&val,300);
w0000 の値を読込んで val に保存します。

(5) GopSetFuncTbl

定義
```
void GopSetFuncTbl(pGOP_FUNCTBL tbl,int count);
```
動作
GOP 側からの自動送信コマンドのメッセージのハンドラ関数を登録します。
引数
tbl：メッセージと対応するハンドラ関数を定義した GOP_FUNCTBL 型配列へのアドレス。
count：tbl の登録件数。
GOP_FUNCTBL 構造体について
メンバ
key：メッセージを表す文字列。
func： 対応するハンドラ関数へのポインタ。
func は以下のインターフェースを持ちます。
void func(void)
使用例
```
void hoge1(){
...
}
void hoge2(){
...
}

const GOP_FUNCTBL[] tbl={
 {"HOGE1",hoge1},
 {"HOGE2",hoge2},
};
main(){
....
GopSetFuncTbl(tbl,2);
....
while(1) {
     .....
     GopCheckMsg();//次項参照
}
}
```
と定義すると GOP から"HOGE1"というメッセージが送信されると hoge1 が呼ばれます。

(6) GopCheckMsg

定義
```
char *GopCheckMsg();
```
動作
シリアルの受信を監視し、必要に応じハンドラ関数を呼び出します。
戻り値
受信した文字列へのポインタ。
受信無し時は NULL を返します。
使用例
```
main(){
.....
while(1){
//処理のﾒｲﾝﾙｰﾌﾟ
GopCheckMsg();
}
```
GOP からの自動送信を使用する場合は、メインの処理ループで呼ぶ様にして下さい。

(7) SetResponseMode

定義
void SetResponseMode(int i);
動作
通信の応答確認の動作の有効/無効を設定します。
この設定は GOP 実機の設定とあわせてご使用下さい。
GOP の設定とアンマッチのときは正常に動作しません。
引数
i＝0(FALSE)以降の動作で応答確認処理を行いません。
i≠0(TRUE)以降の動作で応答確認処理を行います。

(8) GopBlockRead

定義
int GopBlockRead(int addr,char *dataaddr,int size);
動作
GOP からブロック読み出します。
引数
addr：読み出す GOP の先頭アドレス。
dataaddr：読込んだデータを格納するホスト側のバッファの先頭アドレス。
size：読込むサイズ。
戻り値
0：読込み失敗。
1：読込み成功。

(9) GopBlockWrite

定義
void GopBlockWrite(int addr,char *dataaddr,int size);
動作
GOP へブロック書込みします。
引数
addr：書込む GOP の先頭アドレス。
dataaddr：書込むデータを格納するホスト側のバッファの先頭アドレス。
size：書込むサイズ。

(10) GopSetErrFunc

定義
void GopSetErrFunc(void (*func)(int,void *));
動作
通信エラー発生時のエラーハンドラを登録します。
通信エラー発生時登録したハンドラ関数が呼ばれます。
引数
func：エラーハンドラ関数。
エラーハンドラ関数は以下のインターフェースを持ちます。
void func(int errno,void *param)
※通信エラーが発生すると GopResetErr を実行するまで GOP との通信が行えなくなります。
※エラーハンドラ関数について
発生したエラーは引数 error で判別します。エラーに対応した処理を行いエラーを解除(GopResetErr 関数を実行)して下さい。
引数 error の値
ERR_TRAFFIC (0) ：直前に送ったメモリリードでない別のメモリの値が返信された。
param の値：誤って返信されたコマンド文字列(char*)でキャスト)。
ERR_TIMEOUT(1) ：コマンドに対する応答が一定時間無い場合。
param の値：NULL
ERR_BLOCK(2) ：WB/RB コマンドで想定外のコマンド返信があった。
param の値：誤って返信されたコマンド文字列(char*)でキャスト)。

(11) GopSendCommandWithRes

定義
```
char * GopSendCommandWithRes(char *msg,char *retbuf,int timeout);
```
動作
msg で指定された文字列を stx,etx を付加し GOP へ送信します。
送信後 GOP からの返信を待ちます。
※返信が無いコマンドの送信に本コマンドは使用しないで下さい。
引数
msg：送信する文字列
retbuf ：返信文を格納するバッファのアドレスの先頭
timeout：返信文のタイムアウト時間
戻り値
受信したデータ(STX,ETX 削除済み)へのポインタ。受信データ取得できなかった場合 NULL を返します。

(12) GopSetMemTbl

定義
```
void GopSetMemTbl(pGOP_FUNCTBL tbl,int count);
```
動作
GOP 側からの自動送信メモリ出力のメッセージのハンドラ関数を登録します。
引数
tbl：メッセージと対応するハンドラ関数を定義した GOP_LINKTBL 型配列へのアドレス。
count：tbl の登録件数。
GOP_LINKTBL 構造体について
メンバ
memname：メモリ名を表す文字列。
valaddr：受信したメモリの値を格納するための領域のアドレス。
valsize：受信するメモリの最大サイズ。メモリが T 型のとき必要。それ以外は 0 で可。
func ：対応するハンドラ関数へのポインタ。
func は以下のインターフェースを持ちます。
void func(GOP_LINKTBL*param)
使用例
```
void L0000_rcv(GOP_LINKTBL *mem){
   ...
}
long val_L0000;
unsigned short val_w0004
char val_T0010[16];
void mem_rcv(GOP_LINKTBL *mem){
   if(strcmp(mem->memname,"L0000")==0){
        ...
   }else if(strcmp(mem->memname,"w0004")==0)
        ...
   }else if(strcmp(mem->memname,"T0004")==0)
        ...
   }
}

const GOP_MEMTBL[] tbl={
   {"L0000",(void *)&val_L0000,0,mem_rcv},
   {"w0004",(void *)&val_w0004,0,mem_rcv},
   {"T0010",(void *)&val_T0010,16,mem_rcv},
};
main(){
.  ...
   GopSetMemTbl(tbl,3);
   ....
   while(1) {
        .....
        GopCheckMsg();//次項参照
   }
}
```

GOP 側で OUTPUT L0000 を実行すると"AL0000=1234"というメッセージが送信されます。
ホスト側でこのメッセージを受信すると val_L0000 に 1234 が格納され、mem_rcv が呼ばれます。mem_rcv 中で mem->memname で受信したメモリを判定し処理を振り分けます。
※テーブルでメモリ毎に別のハンドラ関数を指定すればハンドラ関数内での処理の振り分けは不要です。

(13) GopSetXRcvFunc

定義
```
void GopSetXRcvFunc(void (*func)(void));
```
動作
GOP 起動時の出力コマンド X 受信時のハンドラ関数を設定します。
本ハンドラ関数を登録しなくても X コマンドに対する x コマンドの返信は行なわれます。
引数
func：X のハンドラ関数。
ハンドラ関数は以下のインターフェースを持ちます。
```
void func(void)
```

(14) GopResetErr

定義
```
void GopResetErr();
```
動作
GOP との通信エラーが発生すると Goplib は通信の出力を禁止します。
本関数を実行することで禁止状態を解除します。

(15) GopUpload

定義
```
int GopUpload(char *filename,char *buf,int size);
```
動作
メモリ上のファイルイメージデータを GOP の RAMDISK にファイルとして転送します。
※GOP-5000 シリーズには(16)の GopFastUpload 関数を使用したほうが高速に転送出来ます。
引数
filename：GOP の RAMDISK に保存時のファイル名。
buf：ファイルイメージ格納アドレスの先頭アドレス。
size：ファイルイメージのデータサイズ。
戻り値
転送失敗 0
転送成功 1
※本 API を処理中にホスト側の動作を中断した場合、GOP が転送モードのままの状態となる可能性がありますのでそうならないようご使用に注意して下さい。
また、このような状況となった場合'GopSendCommand("ULE");'を実行することで転送モードから回復できる場合があります。

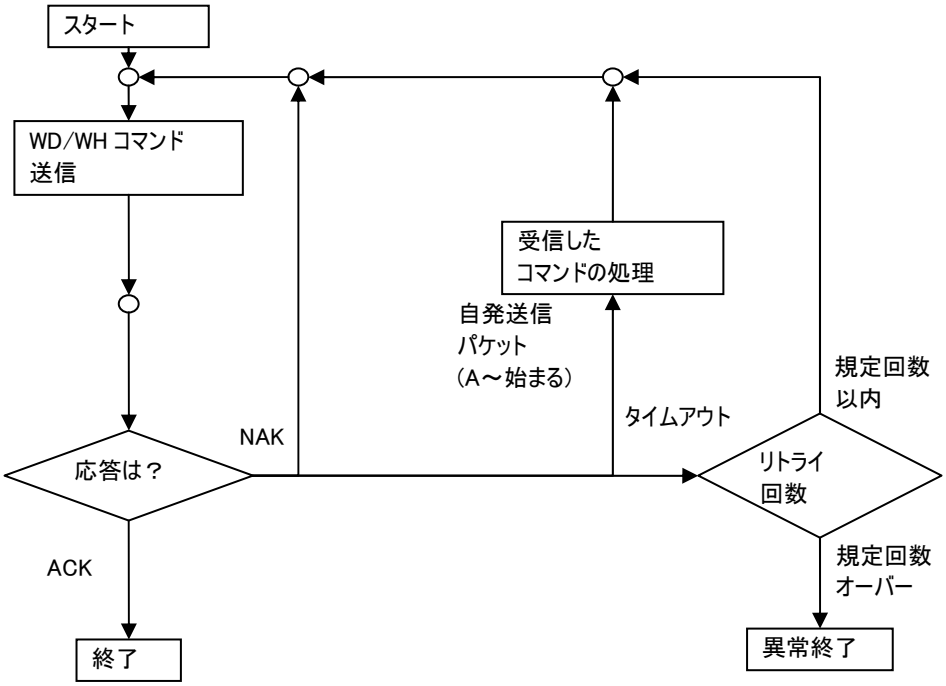
(16) GopFastUpload

定義
```
int GopFastUpload(char *filename,char *buf,int size);
```
動作
メモリ上のファイルイメージデータを GOP の RAMDISK にファイルとして転送します。
※機能としては(15)GopUpload と同様ですが、より高速に転送出来ます。
※本関数は GOP-5000 シリーズ専用となります。
引数
filename：GOP の RAMDISK に保存時のファイル名。
buf：ファイルイメージ格納アドレスの先頭アドレス。
size：ファイルイメージのデータサイズ。
戻り値
転送失敗 0
転送成功 1
※本 API を処理中にホスト側の動作を中断した場合、GOP が転送モードのままの状態となる可能性がありますのでそうならないようご使用に注意して下さい。

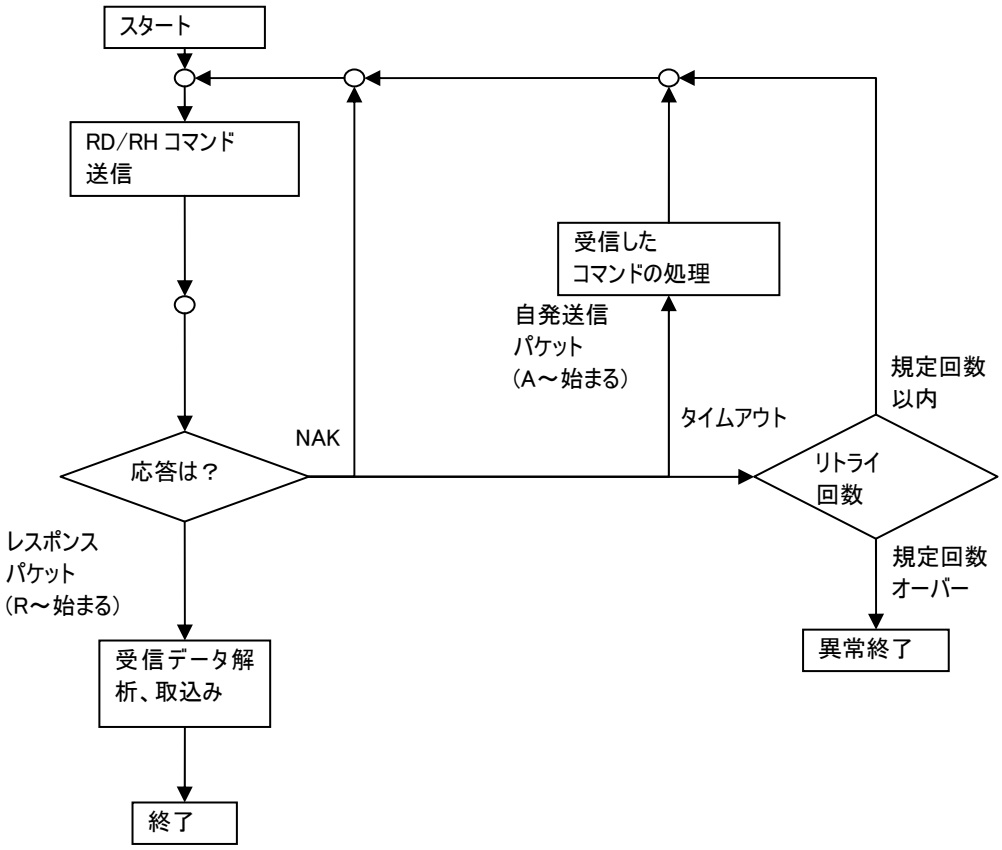
管理番号	C06621A-Z080B

2. 2. 3　参考資料：Goplib内の処理フロー及びGOP側の処理フロー概略

(1)WD/WHコマンド処理フロー

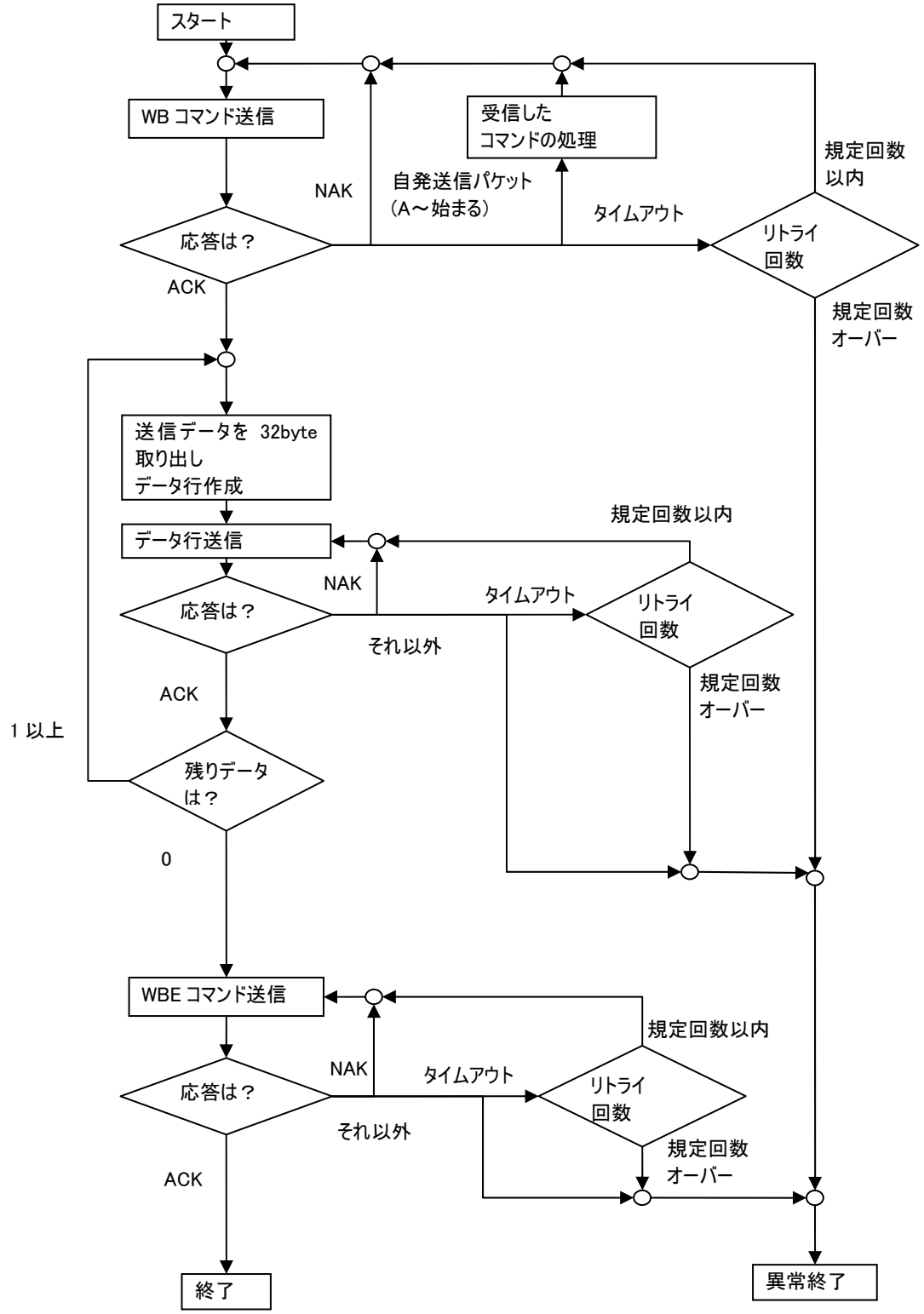


(2)RD/RHコマンド処理フロー



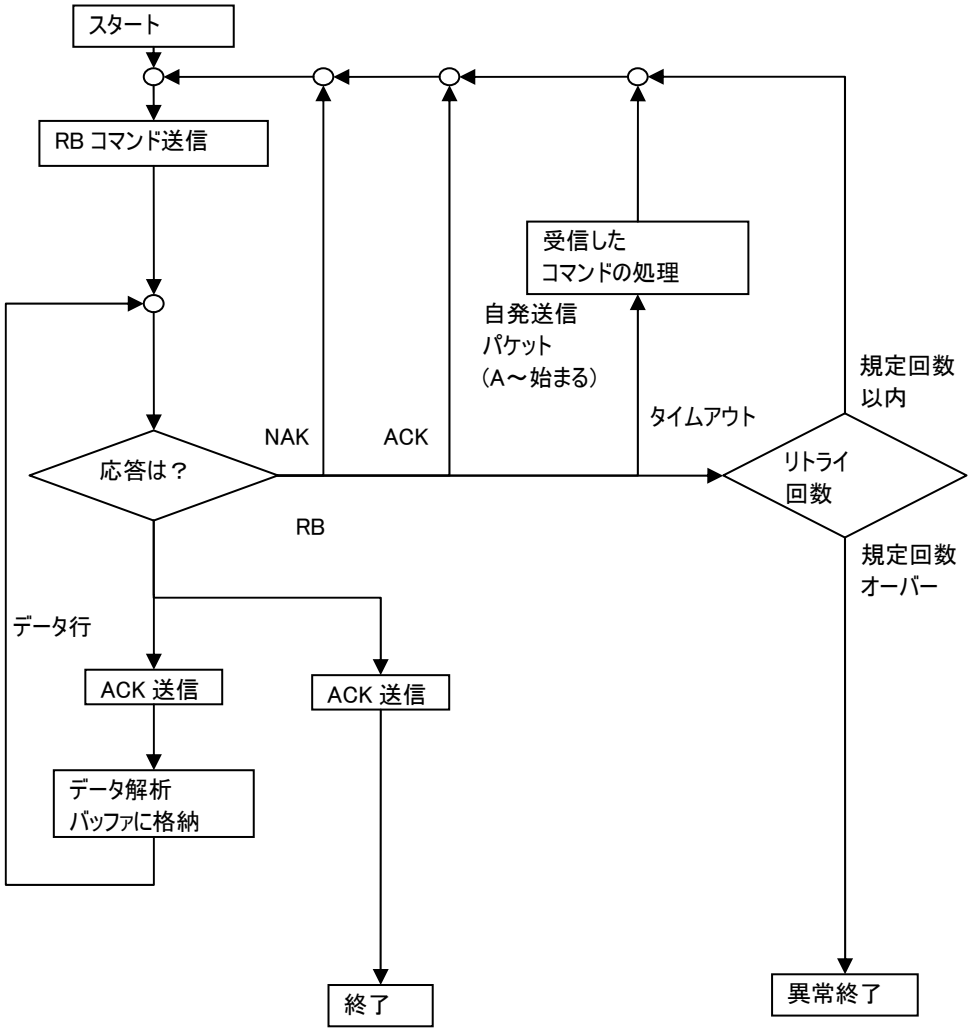
管理番号	C06621A-Z080B

(3)WBコマンド処理フロー



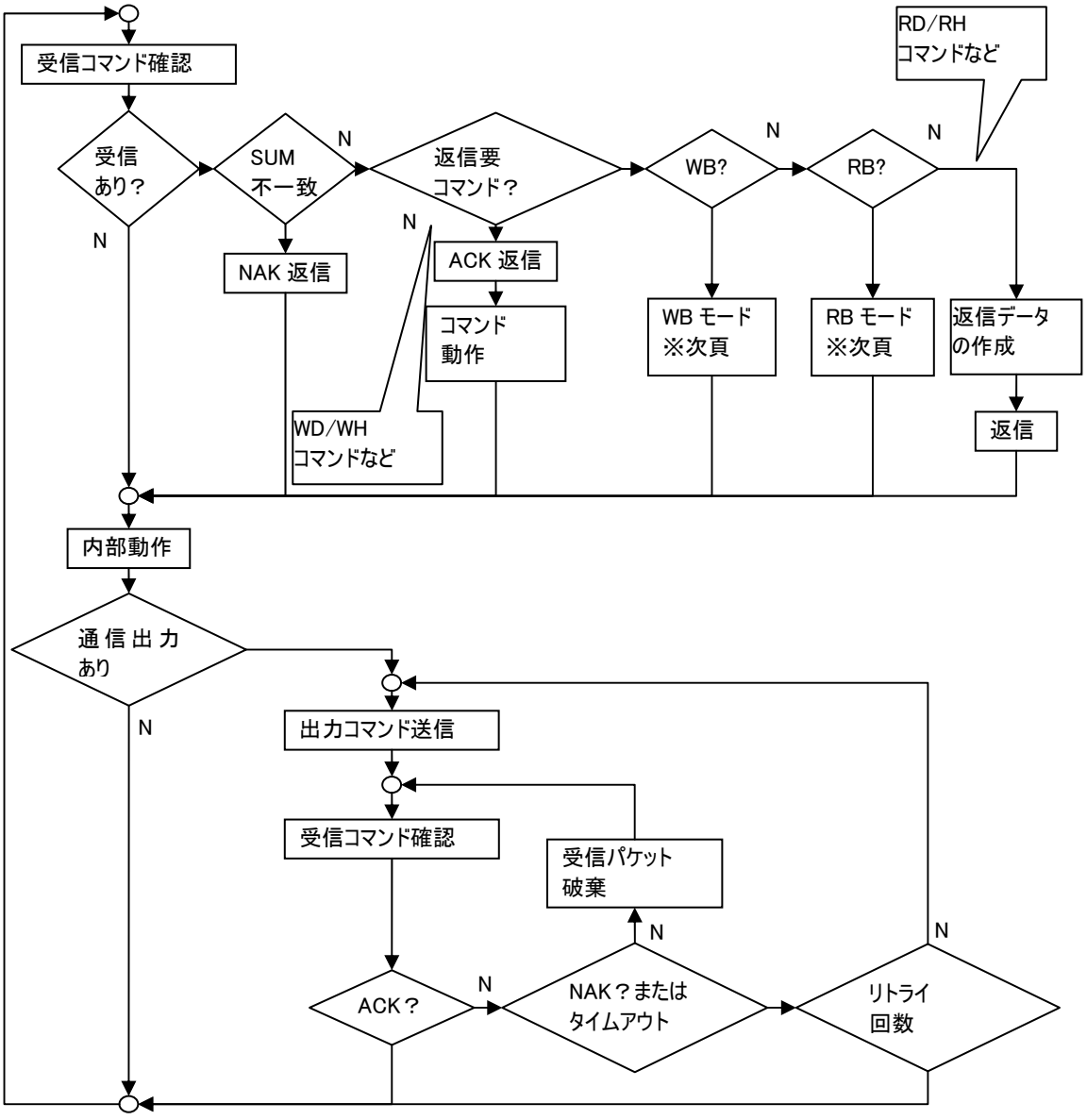
管理番号	C06621A-Z080B

(4)RBコマンド処理フロー

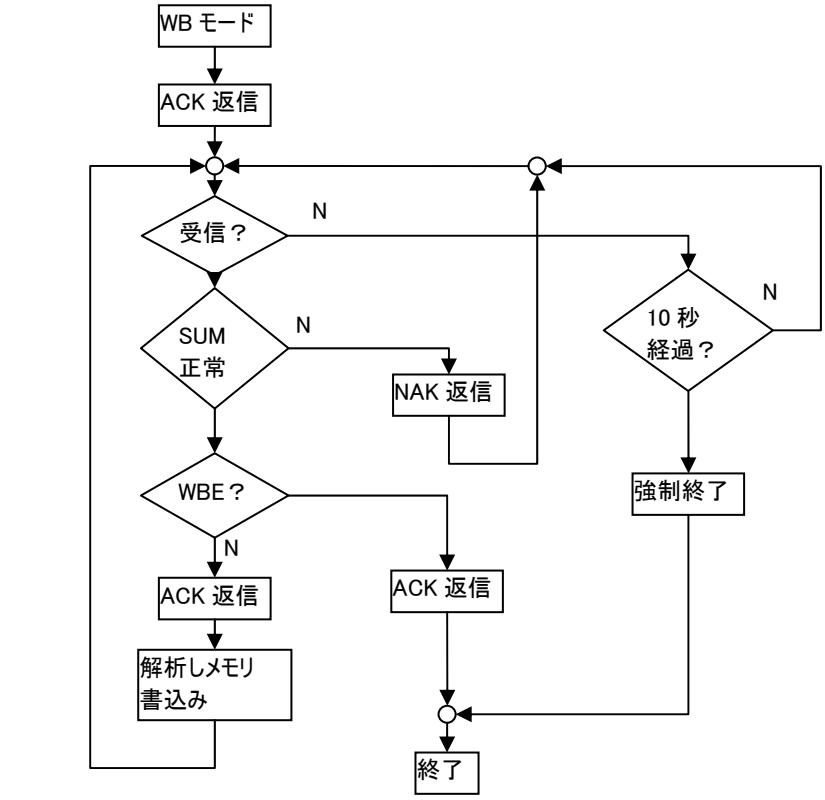


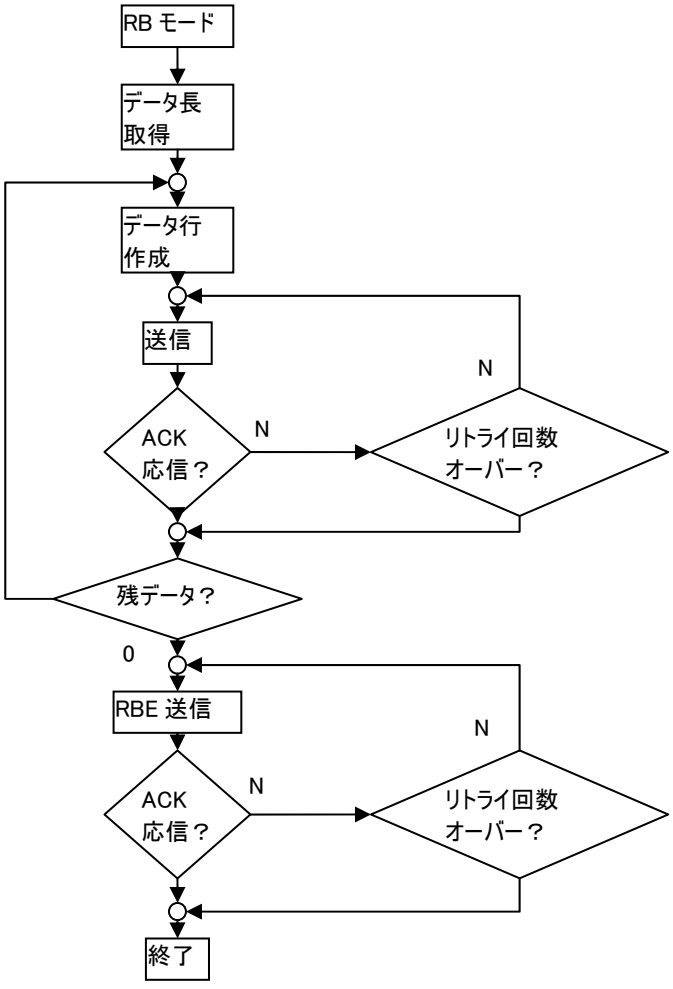
管理番号	C06621A-Z080B

(5)GOP側簡易通信処理フロー



管理番号	C06621A-Z080B

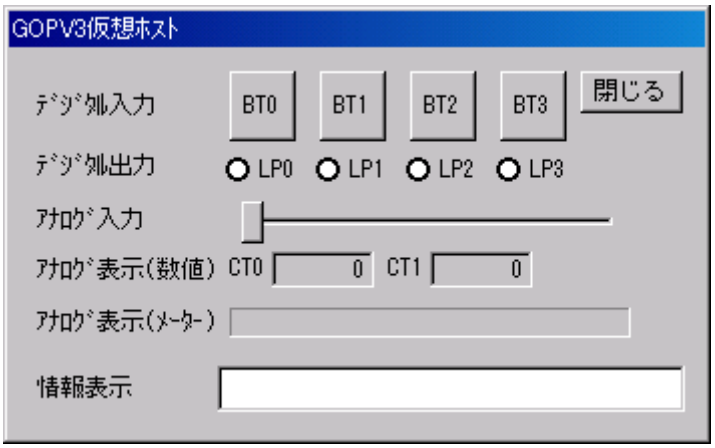




2. 3 仮想ホスト

GOP のホストアプリケーションを作成するに当たり、ホストのプラットホームが必要ですが、このアプリケーションノートでは Windows 上で動作するボードマイコンのシュミレータ(以下仮想ホスト)を使用し説明します。
仮想ホストは VisualStudio6、VisualStudio2008ExpressEdition でのビルド環境を用意しています。

2. 3. 1 仮想ホストの仕様



【仮想ホスト外観】

ホスト側のインターフェースとして
・デジタル入力デバイスとしてボタンを 4 つ配置
・デジタル出力デバイスとしてランプを 4 つ配置
・アナログ入力デバイスとしてスライダーを 1 つ配置(値は 0～255)
・アナログ出力デバイスとして数値表示カウンタを 2 つ、レベルメーターを1つ配置(値は 0～255)
・その他情報入出力用としてエディットボックスをひとつ配置しています。
仮想ホストの制御は次項の API を使用し行います。仮想ホストは main が呼ばれた時点でシリアル通信が使用可となっています。シリアルポートはデフォルトで GOP シミュレーター(SIM)を使用します。
変更する場合は dummyhard.h の#define USEPORT COM1 の行の COM1 を任意のポート番号に変更して下さい。
※使用可能なポートは SIM および COM1～COM5 までです。

2. 3. 2 仮想ホストのAPI

(1) commGetc

定義
```
int commGetc();
```
動作
シリアルポート(厳密には受信バッファ)から一文字受信します。
戻り値
受信した文字の文字コード。
受信すべきデータがない場合は EOF(-1)を返します。

(2) commPutc

定義
```
void commPutc(char c);
```
動作
シリアルポートへ一文字送信します
引数
c：送信する文字

(3) TimeCount

定義
int TimeCount();
動作
システム時間の取得します
戻り値
PC 起動開始後からの経過時間を ms 単位で返します

(4) Wait

定義
void Wait(int i);
動作
指定時間待機します

(5) SetTimerFunc

定義
void SetTimerFunc(void (*t)());
動作
1ms ﾀｲﾏ割り込みハンドラの登録
引数
t：void 型関数へのポインタ
使用例
```
void hoge(){
}
...
main(){
SetTimerFunc(hoge);
....
```
とすると、1ms 間隔で hoge が呼ばれます。

(6) GetSlider

定義
int GetSlider();
動作
ｱﾅﾛｸﾞ入力値を取得
戻り値
0～255 の整数

(7) SetLevel

定義
void SetLevel(int i);
動作
レベルメーターに値をセット
引数
i：0～255 の整数

(8) SetCT

定義
void SetCT(int val,int no);
動作
数値カウンターに値をｾｯﾄ
引数
val：:ｾｯﾄ値
no：ｾｯﾄするｶｳﾝﾀ

(9) SetInfo
定義
void SetInfo(unsigned char *msg);
動作
情報表示内容を設定
引数
msg：表示する文字列

(10) GetInfo
定義
char *GetInfo(char *buf,int length);
動作
情報表示内容を取得
引数
buf：格納領域
length：最大サイズ
戻り値
取得した文字列へのポインタ

(11) SetLamp
定義
void SetLamp(int state);
動作
ﾃﾞｼﾞﾀﾙ出力の設定
引数
state：ランプの点灯状態
state の bit0～3 が LP0 から 3 に対応
ﾋﾞｯﾄがたっていると ON
使用例
SetLamp(5)
LP0 と LP2 が点灯

(12) GetLamp
定義
int GetLamp();
動作
ﾃﾞｼﾞﾀﾙ出力状態の取得
戻り値
ランプの点灯状態
※値については⑪SetLamp と同様

(13) GetBtn
定義
int GetBtn();
動作
ﾃﾞｼﾞﾀﾙ入力状態の取得
戻り値ボタンの押下状態
戻り値の bit0～3 が BT0～3 に対応
使用例
BT0 が押下時
GetBtn()は 1 を返す
BT2 が押下時
GetBtn()は 4 を返す

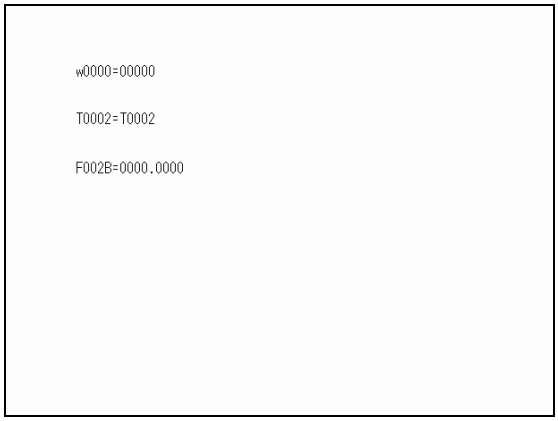
2. 4 サンプルアプリケーション

(1) ホストからGOPのメモリに値を書込む

動作仮想ホストの BT0～3 のいずれかを押下すると GOP のメモリに以下を書込みます。

w0000 に 123
T0002 に"GOP のテストです"
F002b に 123.456

GOP 画面データ APP1.et4



ホストプログラム APP1.c

```
1://仮想ホストの定義ファイル
2:#include "../dummysys/dummyhard.h"
3://GopLib の定義ファイル
4:#include "../goplib.h"
5:
6:void main(){
7:     //GOP ライブラリの初期化
8:     GopInitLib(commGetc,commPutc,TimeCount);
9:     //GOP をリセットします
10:    GopSendCommand("UC");
11:    Wait(2000);
12:    while(!GetBtn());//ボタンが何か押されるまで待つ
13:    GopWrite_w(0x0000,123);
14:    GopWrite_T(0x0002,"GOP のテストです");
15:    GopWrite_F(0x002b,123.456);
16:}
```

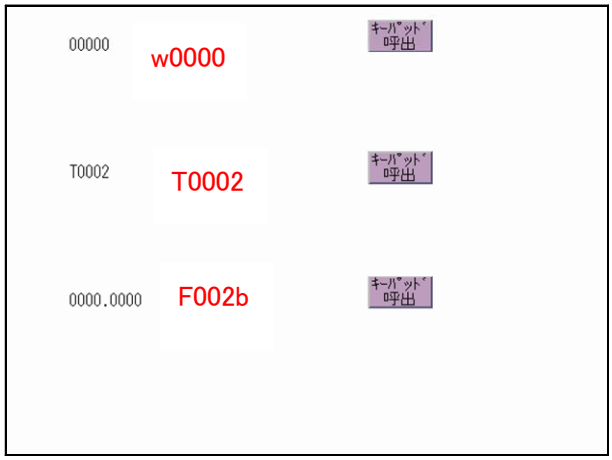
説明

2 行目： 仮想ホストの定義ファイルを読込みます。
4 行目： GopLib の定義ファイルを読込みます。
8 行目： GopLib にホスト側で用意してある、1 文字送受信、時間計測関数を設定します。
10 行目： ホストプログラム開始にあわせ GOP を初期化するため GOP にリセットコマンドを送信しています。
11 行目： GOP の立ちあがり待ちのため 2 秒程度 wait を入れています。
12 行目： ホスト側のボタンが押下されるまで待機します。
13 行目： GOP の 0000 番地に w 型でデータを書込みます。
14 行目： GOP の 0002 番地に T 型でデータを書込みます。
15 行目： GOP の 002b 番地に F 型でデータを書込みます。

(2) ホストからGOPのメモリから値を読込む

動作 GOP の画面上で w0000,T0002,F002b の値をキーパッドを使用し編集します。
仮想ホスト側で BT0 を押すと w0000、BT1 を押すと T0002、BT2 を押すと F002b の値を読込み情報表示欄に取得した内容を表示します。

GOP 画面データ APP2.et4



ホストプログラム APP2.c

```
1://仮想ホストの定義ファイル
2:#include "../dummysys/dummyhard.h"
3://GopLib の定義ファイル
4:#include "../goplib.h"
5:
6://デモとして sprintf を使用するため stdio.h を呼び込みます
7:#include <stdio.h>
8:void main(){
9:    unsigned short w_value;
10:   float f_value;
11:   char t_value[TEXTDATAMAXLEN],temp[100];
12:   //GOP ライブラリの初期化
13:   GopInitLib(commGetc,commPutc,TimeCount);
14:   //GOP をリセットします
15:   GopSendCommand("UC");
16:   Wait(2000);
17:   while(1){
18:       switch(GetBtn()){
19:       case 0x01:
20:            if(GopRead_w(0x0000,&w_value,3000)){
21:                sprintf(temp,"w0000 は%d です",w_value);
22:            }else strcpy(temp,"読込み失敗");
23:            SetInfo(temp);
24:            break;
25:       case 0x02:
26:            if(GopRead_T(0x0002,t_value,3000)){
27:                sprintf(temp,"T0002 は¥"%s¥"です",t_value);
28:            }else strcpy(temp,"読込み失敗");
29:            SetInfo(temp);
30:            break;
31:       case 0x04:
```

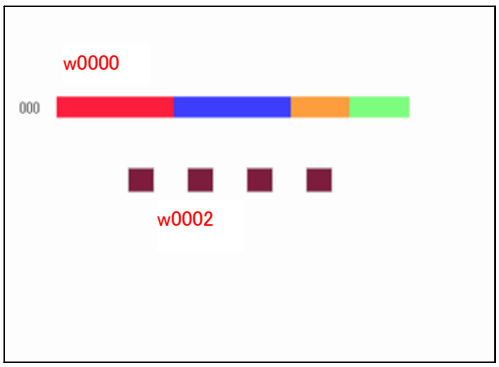
```
32:             if(GopRead_F(0x002b,&f_value,3000)){
33:                 sprintf(temp,"F002b は%f です",f_value);
34:             }else strcpy(temp,"読込み失敗");
35:             SetInfo(temp);
36:             break;
37:         }
38:         Wait(10);
39:     }
40:}
```

説明

10 行目： w 型メモリのデータ読込み領域として unsigned short 型で変数(w_value)を取得します。
11 行目： F 型メモリのデータ読込み領域として float 型で変数(f_value)を取得します。
12 行目： T 型メモリのデータ読込み領域として char 型の配列で変数(t_value)を取得します。
18 行目： 仮想ホストのボタン押下状態を取得し、取得値に応じ処理を振り分けます。
19 行～： BT0 が押された時の処理です。
20 行目： GOP の w0000 を読出し w_value に格納するため GopRead_w 関数を呼び出します。
引数として w_value のアドレスを渡しため&w_value を指定します。
GopRead_w が成功すると w_value には GOP の w0000 の値が格納されます。
temp に読込んだ値を文字列化し、情報表示エディットボックスに表示します。
受信タイムアウト等が発生すると GopRead_w は失敗します。
25 行目： GOP の T0002 を読出し t_value 配列にデータを格納します。基本的には 20 行目と同じですが、T 型メモリの格納先は配列になり t_value 自体がアドレスを表しますので&t_value とする必要はありません。
31 行目： F002b を f_value に読込みます。動作は 20 行目と同じです。

(3) ホストからGOPのメモリに一定間隔で値を書込む

動作仮想ホストのスライダーの値をリアルタイムに GOP のバーメーターとカウンタ(w0000)に
仮想ホストのボタン押下状態を GOP のランプ(w0002)にリアルタイムに反映します。
画面データ-APP3.et4



ホストプログラム APP3.c

```
1://仮想ホストの定義ファイル
2:include "../dummysys/dummyhard.h"
3://GopLib の定義ファイル
4:#include "../goplib.h"
5:
6:void main(){
7:    int oldSendTime=0;
8:    //GOP ライブラリの初期化
9:    GopInitLib(commGetc,commPutc,TimeCount);
10:   //GOP をリセットします
11:   GopSendCommand("UC");
12:   Wait(2000);
13:   while(1){
14:       if(TimeCount()>oldSendTime+50){//前回送信から 20ms 以上経過確認
15:           GopWrite_w(0x0000,GetSlider());
16:           GopWrite_w(0x0002,GetBtn());
17:           oldSendTime=TimeCount();
18:       }
19:       Wait(10);
20:   }
21:}
```

説明

8 行目： 前回送信した時刻を記憶します。

14 行目： 前回送信から 50ms 以上経過したか確認しています。
GOP 側が処理できる通信コマンドは限りがあり、間をおかずコマンド送信しつづけると処理が間に合わず、GOP が暴走する恐れがあります。
ホストは GOP にコマンド送信する場合、大幅な描画変更が行われない動作と仮定して 10ms 以上の送信間隔をあけることが望ましいです。
今回このサンプルでは 50ms 間隔で 2 コマンドずつ送信するよう処理しています。
尚コマンド送信密度を減らす方法として、前回値と今回値が違う時のみ値を送る等の方法もありますが今回のサンプルでは入れていません。

15 行目： 仮想ホストのスライダーの値を読み取り w0000(バーメーター)に書込みます。

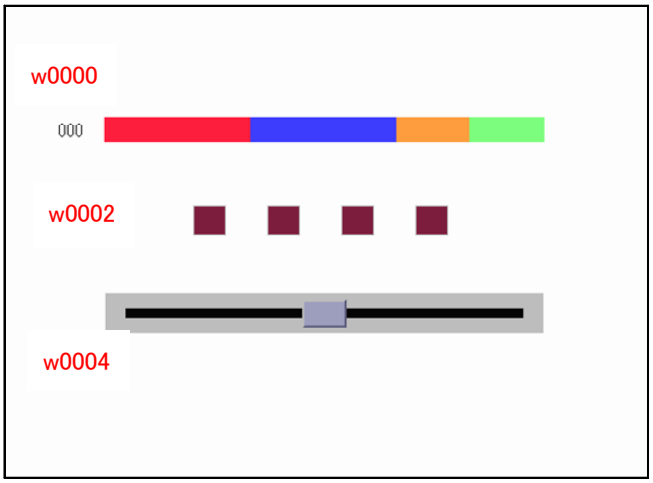
16 行目： 仮想ホストのボタンの値を読み取り w0002(ランプ)に書込みます。

17 行目： 前回送信時刻を更新します。

(4) ホストからGOPのメモリに一定間隔で値を書込む

動作前述の動作に加え、GOP 側でスライダーを配置し(w0004)このメモリの値を仮想ホストが一定間隔で読み取り仮想ホストの数値表示とレベルメーターに反映します。

画面データ APP4.et4



ホストプログラム APP4.c

```
1://仮想ホストの定義ファイル
2:#include "../dummysys/dummyhard.h"
3://GopLib の定義ファイル
4:#include "../goplib.h"
5:
6:void main(){
7:    int oldSendTime=0;
8:    unsigned short w_value;
9
10:   //GOP ライブラリの初期化
11:   GopInitLib(commGetc,commPutc,TimeCount);
12:   //GOP をリセットします
13:   GopSendCommand("UC");
14:   Wait(5000);
15:   while(1){
16:       if(TimeCount()>oldSendTime+50){
17:           GopWrite_w(0x0000,GetSlider());
18:           GopWrite_w(0x0002,GetBtn());
19:           if(GopRead_w(0x0004,&w_value,500)){
20:               SetLevel(w_value);
21:               SetCT(w_value,0);
22:           }
23:           oldSendTime=TimeCount();
24:       }
25:       Wait(10);
26:   }
27:}
```

説明

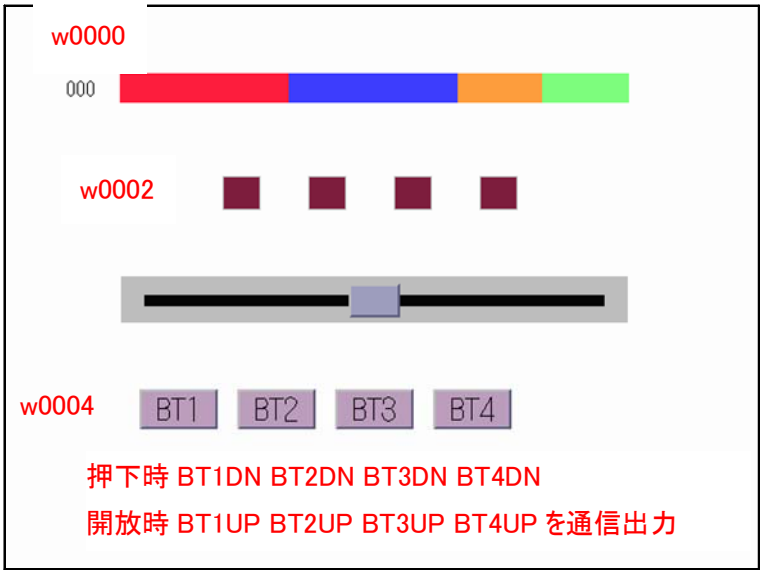
19 行目： w0004 の値を読込みます。読込みが成功するとレベルメーターとカウンタに値を表示します。

(5) GOPのボタン押下イベントをホストで取り込む

動作前項に加え、GOP でのボタン押下、開放を通信出力でイベント通知しホスト側でのそのイベントを取得しランプに表示反映します。

※本項の動作は 3. 4 のサンプルと同様の方法で簡素に実現することは可能ですが、イベント受信ハンドラの動作説明のため以下のような方法で実現します。

画面データ APP5.et4



ホストプログラム APP5.c

```
1://仮想ホストの定義ファイル
2:#include "../dummysys/dummyhard.h"
3://GopLib の定義ファイル
4:#include "../goplib.h"
5:
6:void fnBT1DN(){
7:    SetLamp(GetLamp()|1);
8:}
9:void fnBT2DN(){
10:   SetLamp(GetLamp()|2);
11:}
12:void fnBT3DN(){
13:   SetLamp(GetLamp()|4);
14:}
15:void fnBT4DN(){
16:   SetLamp(GetLamp()|8);
17:}
18:void fnBT1UP(){
19:   SetLamp(GetLamp()&~1);
20:}
21:void fnBT2UP(){
22:   SetLamp(GetLamp()&~2);
23:}
24:void fnBT3UP(){
25:   SetLamp(GetLamp()&~4);
26:}
27:void fnBT4UP(){
28:   SetLamp(GetLamp()&~8);
```

```
29:}
30:
31:GOP_FUNCTBL functbl[8]={
32:    {"BT1DN",fnBT1DN},
33:    {"BT2DN",fnBT2DN},
34:    {"BT3DN",fnBT3DN},
35:    {"BT4DN",fnBT4DN},
36:    {"BT1UP",fnBT1UP},
37:    {"BT2UP",fnBT2UP},
38:    {"BT3UP",fnBT3UP},
39:    {"BT4UP",fnBT4UP},
40:};
41:
42:void main(){
43:    int oldSendTime=0;
44:    unsigned short w_value;
45:    //GOP ライブラリの初期化
46:    GopInitLib(commGetc,commPutc,TimeCount);
47:    //GOP をリセットします
48:    GopSendCommand("UC");
49:    Wait(5000);
50:    GopSetFuncTbl(functbl,8);
51:    while(1){
52:        if(TimeCount()>oldSendTime+50){
53:            GopWrite_w(0x0000,GetSlider());
54:            GopWrite_w(0x0002,GetBtn());
55:            if(GopRead_w(0x0004,&w_value,500)){
56:                SetLevel(w_value);
57:                SetCT(w_value,0);
58:            }
59:            oldSendTime=TimeCount();
60:        }
61:        GopCheckMsg();
62:        Wait(10);
63:    }
64:}
```

説明

6 行目～ 29 行目: "BT1DN"などのメッセージを受け取った時の動作を行う関数を定義します。内容は該当するランプを点灯状態にします。
各ボタンの押下または開放時の動作を行う関数を定義します。

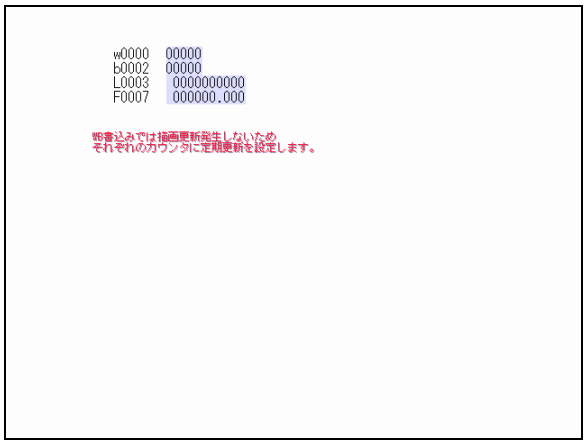
31 行目～ 40 行目: GOP が自動送信する文字列とそれに対応する関数の対応表を GOP_FUNCTBL 型の配列として定義します。

50 行目: 上記で定義した対応表を GopLib 側に通知します。

61 行目: 受信コマンドの解析を行います。受信コマンドとして上で定義したコマンドを受け取るとそれに対応した関数が起動します。

(6) WB/RBコマンドを使用したブロック書込み･読込み

GOP の連続したメモリ領域に WB コマンドでデータを書込み、そのデータを RB コマンドで読込みます。
WB で書込みを行うため、ホスト側のメモリに GOP のメモリ配置と同じ構成のメモリバッファを作成しそこにデータを書込み、GopLib の GopBlockWrite 関数を使用してデータを書込みます。
続いて GopBlockRead 関数を使用し書込んだデータを読み出し内容を比較します。
画面データ APP6.et4



ホストプログラム APP6.c

```
01 : //仮想ホストの定義ファイル
02 : #include "../dummysys/dummyhard.h"
03 : //GopLib の定義ファイル
04 : #include "../goplib.h"
05 : #include <stdlib.h>
06 :
07 : void Rb(unsigned short w0000,unsigned char b0002,long L0003,float F0007){
08 :     char buf[11],*d;
09 :     unsigned short r_w0000;
10 :     unsigned char r_b0002;
11 :     long r_L0003;
12 :     float r_F0007;
13 :     //buf に RB コマンドで GOP メモリイメージを読込み
14 :     GopBlockRead(0x0000,buf,11);
15 :     //buf のデータを読み、書込んだデータと比較
16 :     //CPU アライメント規制のため直接 buf をキャストするのではなく
17 :     //該当する型の変数のアドレスにコピーし指定型で読みます。
18 :     d=(char *)&r_w0000;
19 :     *(d+0)=buf[0];
20 :     *(d+1)=buf[1];
21 :     if(r_w0000!=w0000){
22 :           SetInfo("READDATA 不一致");
23 :           return;
24 :     }
25 :     d=(char *)&r_b0002;
26 :     *(d+0)=buf[2];
27 :     if(r_b0002!=b0002){
28 :           SetInfo("READDATA 不一致");
29 :           return;
30 :     }
31 :     d=(char *)&r_L0003;
32 :     *(d+0)=buf[3];
33 :     *(d+1)=buf[4];
34 :     *(d+2)=buf[5];
35 :     *(d+3)=buf[6];
36 :     if(r_L0003!=L0003){
37 :           SetInfo("READDATA 不一致");
```

```
38 :          return;
39 :     }
40 :     d=(char *)&r_F0007;
41 :     *(d+0)=buf[7];
42 :     *(d+1)=buf[8];
43 :     *(d+2)=buf[9];
44 :     *(d+3)=buf[10];
45 :     if(r_F0007!=F0007){
46 :          SetInfo("READDATA 不一致");
47 :          return;
48 :     }
49 : }
50 :
51 : void Wb(unsigned short w0000,unsigned char b0002,long L0003,float F0007){
52 :     char buf[11],*d;
53 :     //buf に GOP メモリの配置の通りにデータを並べます
54 :     d=(char *)&w0000;
55 :     buf[0]=*(d+0);
56 :     buf[1]=*(d+1);
57 :     d=(char *)&b0002;
58 :     buf[2]=*(d+0);
59 :     d=(char *)&L0003;
60 :     buf[3]=*(d+0);
61 :     buf[4]=*(d+1);
62 :     buf[5]=*(d+2);
63 :     buf[6]=*(d+3);
64 :     d=(char *)&F0007;
65 :     buf[7]=*(d+0);
66 :     buf[8]=*(d+1);
67 :     buf[9]=*(d+2);
68 :     buf[10]=*(d+3);
69 :     //buf を WB コマンドで書込み
70 :     GopBlockWrite(0x0000,buf,11);
71 : }
72 :
73 : void main(){
74 :     //GOP ライブラリの初期化
75 :     int oldSendTime=0;
76 :     unsigned short w0000=0;
77 :     unsigned char b0002=0;
78 :     long L0003=0;
79 :     float F0007=0.0;
80 :     GopInitLib(commGetc,commPutc,TimeCount);
81 :     while(1){
82 :          if(TimeCount()>oldSendTime+500){
83 :              unsigned char b;
84 :
85 :              oldSendTime=TimeCount();
86 :              Wb(w0000,b0002,L0003,F0007);
87 :              Rb(w0000,b0002,L0003,F0007);
88 :              w0000++;
89 :              b0002++;
90 :              L0003++;
91 :              F0007+=0.1;
92 :          }
93 :     }
94 : }
```

説明

7 行目～ 49 行目：GOP から RB コマンドで受信したデータと引数で与えられた書込まれているはずの値を比較する関数です。メモリを任意のバッファに読込み、その値を指定の型の変数領域にコピーし読込みます。
なお変数領域にコピーの際ホスト CPU の仕様(エンディアン)によりコピー順を入れ替える必要があります。
ホスト CPU がリトルエンディアンの場合サンプルどおりで OK です。
ホスト CPU がビッグエンディアンの場合、以下のようにコピー順を逆にして下さい。

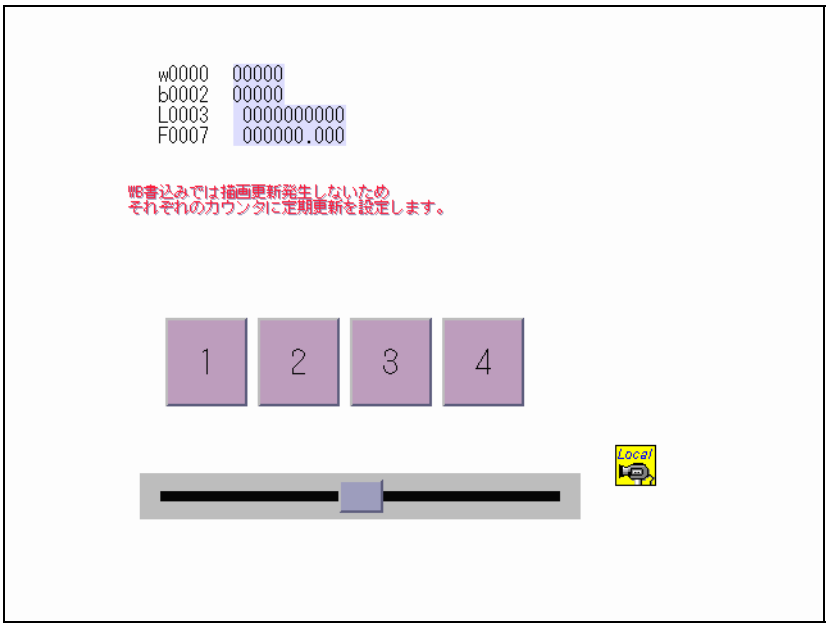
```
59 : d=(char *)&L0003;
60 : buf[3]=*(d+0);   →   60 : buf[3]=*(d+3);
61 : buf[4]=*(d+1);        61 : buf[4]=*(d+2);
62 : buf[5]=*(d+2);        62 : buf[5]=*(d+1);
63 : buf[6]=*(d+3);        63 : buf[6]=*(d+0);
```

51 行目～ 71 行目：GOP に WB コマンドでデータを書込む関数です。引数で与えられたデータを GOP 上のメモリ配置を模したバッファ上に書込み、このバッファを GopBlockWrite 関数で書込みます。
バッファに書込む際、ホストがビックエンディアンの場合エンディアンを入れ替えて下さい。

73 行～ メインループ。
500ms 毎に WB でデータを書き、RB で読込んだデータを比較します。

(7) メモリの通信出力を受けての動作

GOP の通信出力で任意文字列でなく、メモリ値を出力します。
APP6 のデータをベースに、ボタンを追加しボタンを押すと b0010 に1～4 がセットされ通信出力されます。
またスライダーも追加し b0011 を操作し、監視オブジェクトで b0011 の値を出力します。
ホスト側は APP6 の動きに加え、受信した b0010 値の応じてランプを点灯し b0011 の値をレベルメータで表示します。
画面データ-APP7.et4



ホストプログラム APP7.c

```
001 : //仮想ホストの定義ファイル
002 : #include "../dummysys/dummyhard.h"
003 : //GopLib の定義ファイル
004 : #include "../goplib.h"
005 : #include <stdlib.h>
006 :
007 : void Rb(unsigned short w0000,unsigned char b0002,long L0003,float F0007){
    :(省略)
049 : }
050 :
051 : void Wb(unsigned short w0000,unsigned char b0002,long L0003,float F0007){
    :(省略)
071 : }
072 : unsigned char b0010,b0011;
073 : //b0011 受信時の処理
074 : void b0011_handler(GOP_LINKTBL *mem){
075 :   SetLevel(b0011);
076 : }
077 : //b0010 受信時の処理
078 : void b0010_handler(GOP_LINKTBL *mem){
079 :   switch(b0010){
080 :       case 1:
081 :       SetLamp(1);
082 :       break;
083 :       case 2:
084 :       SetLamp(2);
085 :       break;
086 :       case 3:
087 :       SetLamp(4);
088 :       break;
089 :       case 4:
```

```
090 :       SetLamp(8);
091 :       break;
092 :
093 :   }
094 : }
095 : GOP_LINKTBL tbl[]={
096 :   {"b0010",&b0010,0,b0010_handler},
097 :   {"b0011",&b0011,0,b0011_handler},
098 : };
099 : void main(){
100 :   //GOP ライブラリの初期化
101 :   int oldSendTime=0;
102 :   unsigned short w0000=0;
103 :   unsigned char b0002=0;
104 :   long L0003=0;
105 :   float F0007=0.0;
106 :   GopInitLib(commGetc,commPutc,TimeCount);
107 :
108 :   GopSetMemTbl(tbl,2);
109 :   while(1){
110 :       if(TimeCount()>oldSendTime+500){
111 :           unsigned char b;
112 :           printf("%dms¥n",TimeCount()-oldSendTime);
113 :           oldSendTime=TimeCount();
114 :           Wb(w0000,b0002,L0003,F0007);
115 :           Rb(w0000,b0002,L0003,F0007);
116 :           w0000++;
117 :           b0002++;
118 :           L0003++;
119 :           F0007+=0.1;
120 :       }
121 :       GopCheckMsg();
122 :   }
123 : }
```

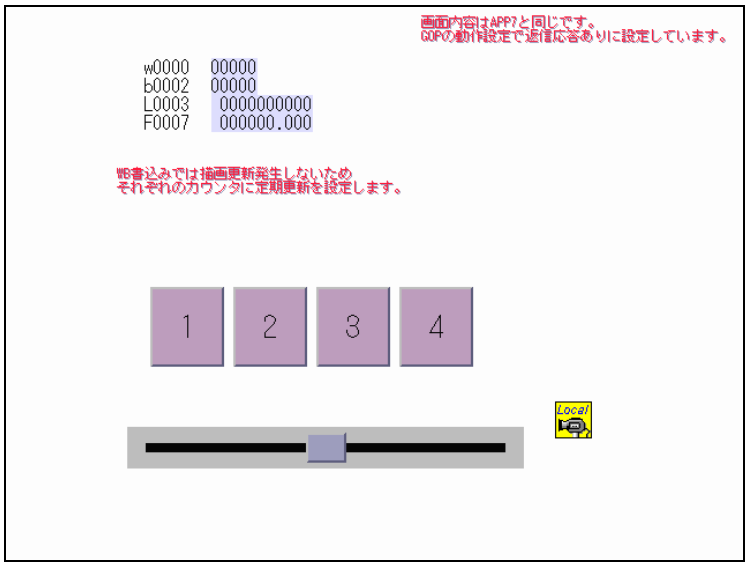
説明

APP6.c に以下を加えます。

72 行目 : b0010,b0011 受け取り用のグローバル変数を用意します。
73 行目～ b0011 受け取り時の処理。受け取った値をレベルメータに表示します。
75 行目 :
77 行目～ b0010 受け取り時の処理。受け取った値に応じたランプを点灯させます。
94 行目 :
95 行目～ メモリとハンドラのリンクテーブルを定義します。
98 行目 :
105 行目 上で定義したリンクテーブルを Goplib にセットします。
121 行目 受信コマンドのチェック・解析を行います。

(8) 応答確認とエラーハンドル

APP7 の画面データを、応答確認ありで通信します。
応答確認ありで Goplib を使用するには SetResponsMode 関数で設定するだけで OK です。
応答確認ありの場合、断線などの通信エラーも検出可能です。通信中ケーブルを抜くと通信エラーをハンドルし"通信エラー"のメッセージを表示します。
画面データ-APP8.et4(APP7 の通信設定を変更)



ホストプログラム APP8.c

```
001 :
002 : //仮想ホストの定義ファイル
003 : #include "../dummysys/dummyhard.h"
004 : //GopLib の定義ファイル
005 : #include "../goplib.h"
006 : #include <stdlib.h>
007 :
008 : void Rb(unsigned short w0000,unsigned char b0002,long L0003,float F0007){
    :(省略)
050 : }
051 :
052 : void Wb(unsigned short w0000,unsigned char b0002,long L0003,float F0007){
    :(省略)
072 : }
073 : unsigned char b0010,b0011;
074 : //b0011 受信時の処理
075 : void b0011_handler(GOP_LINKTBL *mem){
    :(省略)
077 : }
078 : //b0010 受信時の処理
079 : void b0010_handler(GOP_LINKTBL *mem){
    :(省略)
094 : }
095 : GOP_LINKTBL tbl[]={
    :(省略)
098 : };
099 : void err(int errno,void *param){
100 :   SetInfo("通信エラー");
101 :   GopResetErr();
102 : }
103 : void main(){
104 :   //GOP ライブラリの初期化
105 :   int oldSendTime=0;
106 :   unsigned short w0000=0;
107 :   unsigned char b0002=0;
108 :   long L0003=0;
```

```
109 :   float F0007=0.0;
110 :   GopInitLib(commGetc,commPutc,TimeCount);
111 :   SetResponseMode(1);
112 :   GopSetErrFunc(err);
113 :   GopSetMemTbl(tbl,2);
114 :   while(1){
115 :       if(TimeCount()>oldSendTime+500){
116 :           unsigned char b;
117 :           printf("%dms¥n",TimeCount()-oldSendTime);
118 :           oldSendTime=TimeCount();
119 :           Wb(w0000,b0002,L0003,F0007);
120 :           Rb(w0000,b0002,L0003,F0007);
121 :           w0000++;
122 :           b0002++;
123 :           L0003++;
124 :           F0007+=0.1;
125 :       }
126 :       GopCheckMsg();
127 :   }
128 : }
```

説明

APP7.c に以下を加えます。

99 行目～　エラーハンドル関数を定義します。

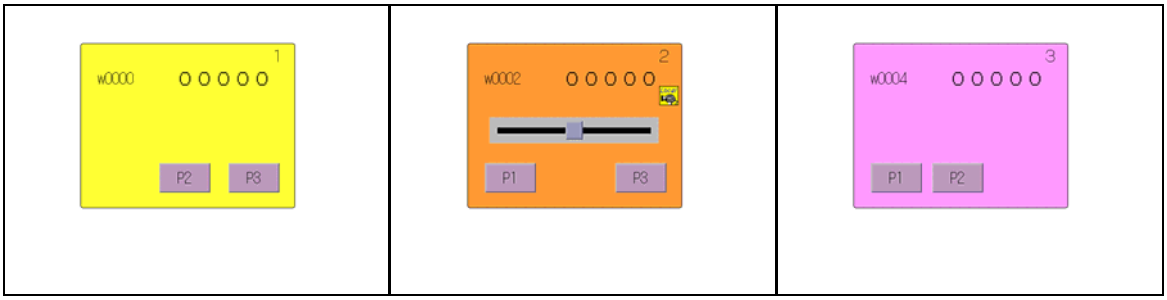
102 行目：　エラー発生のメッセージを表示し、Goplib のエラー状態を解除します。

111 行目：　Goplib を応答確認ありモードに設定します。

112 行目：　上で定義したエラーハンドル関数を Goplib に登録します。

(9) 起動時コマンドのハンドル・ページ遷移

ページ1，2，3を用意します。それぞれ異なるメモリをカウンタで表示します。
ページ遷移は GOP からコマンドを送信し、それを受けてホストがページ移動を行います。
GOP は起動時コマンド出力を行うよう設定します。
GOP が再起動した場合、ホストが管理している表示ページに同期させるように処理します。
Page2 ではスライダーを配置し値の変化でレベルメータを変化させます。
画面データ-APP9.et4



ホストプログラム APP9.C

```
01 : //仮想ホストの定義ファイル
02 : #include "../dummysys/dummyhard.h"
03 : //GopLib の定義ファイル
04 : #include "../goplib.h"
05 : #include <stdlib.h>
06 :
07 : unsigned char b0010;
08 : //b0010 受信時の処理
09 : void b0010_handler(GOP_LINKTBL *mem){
10 :    SetLevel(b0010);
11 : }
12 : GOP_LINKTBL tbl[]={
13 :    {"b0010",&b0010,0,b0010_handler},
14 : };
15 :
16 : static page=1;
17 : void P1(){
18 :    page=1;
19 :    GopWrite_w(0xf000,page);
20 : }
21 : void P2(){
22 :    page=2;
23 :    GopWrite_w(0xf000,page);
24 : }
25 : void P3(){
26 :    page=3;
27 :    GopWrite_w(0xf000,page);
28 : }
29 : void X(){
30 :    Wait(510);  //x 返信後 500ms 受信禁止のため GOP が受信可能になるまで待機
31 :    GopWrite_w(0xf000,page);
32 : }
33 :
34 : GOP_FUNCTBL ftbl[]={
35 :    {"MOVE_P1",P1},
36 :    {"MOVE_P2",P2},
37 :    {"MOVE_P3",P3},
38 : };
39 : void main(){
40 :    //GOP ライブラリの初期化
41 :    int oldSendTime=0;
42 :    unsigned short w0000=0,w0002=0,w0004=0;
43 :    GopInitLib(commGetc,commPutc,TimeCount);
44 :    GopSetMemTbl(tbl,1);
```

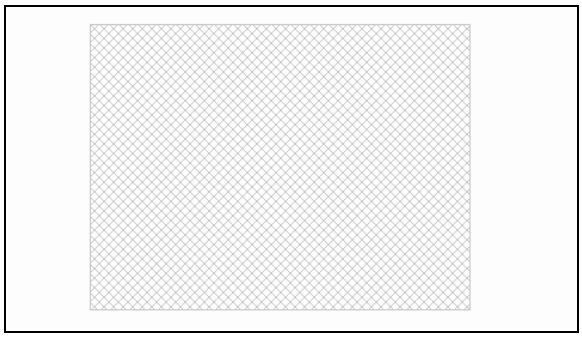
```
45 :    GopSetFuncTbl(ftbl,3);
46 :    GopSetXRcvFunc(X);
47 :    while(1){
48 :        if(TimeCount()>oldSendTime+50){
49 :            oldSendTime=TimeCount();
50 :            switch(page){
51 :                case 1:
52 :                    GopWrite_w(0x0000,w0000);
53 :                    break;
54 :                case 2:
55 :                    GopWrite_w(0x0002,w0002);
56 :                    break;
57 :                case 3:
58 :                    GopWrite_w(0x0004,w0004);
59 :                    break;
60 :
61 :            }
62 :            w0000+=1;
63 :            w0002+=2;
64 :            w0004+=3;
65 :        }
66 :        GopCheckMsg();
67 :    }
68 : }
69 :
```

説明

8 行目～ b0010 受信時の処理およびリンクテーブルの定義。
14 行目:
16 行目: 表示ページ管理変数(page)の定義。
17 行目～ コマンドボタン押下のハンドラ関数。page の値を更新し、GOP のシステムメモリ
28 行目: PAGE(wF000)に page の値をセット。
29 行目～ 起動コマンド受信時の動作。X 返信は Goplib 内で行います。GOP が受信可能になる
32 行目: まで待った後、page を GOP の PAGE に書込み。
34 行目～ 出力コマンドとハンドラ関数のリンクテーブル定義。
38 行目:
46 行目: 起動コマンド受信時の動作を Goplib に登録。
47 行目～ メインループ。
50ms 毎に現在表示中のページに応じたメモリを通信出力。

(10) ホストから画像ファイル(JPEG)の転送と表示

ホストの持つ JPEG 画像ファイルを GOP に転送し、キャンバスオブジェクトにマクロで描画します。



キャンバスオブジェクトの動作は以下となります。

①リンクメモリを b0000 に設定。b0000 に 1 が書込まれると記述したマクロに基づき描画を行ないます。

②キャンバスオブジェクトは以下のマクロを記述しています。

```
LOAD_IMAGE #0 #0 #320 #240 $0:IMAGE.JPG
```

RAMDISK 上の JPEG ファイル(IMAGE.JPG)を描画します。

なお、ボーレートは転送の高速化のため 115200bps で設定します。

ホスト側は以下の動作となります。

①転送する JPEG ファイルをメモリ上に読込み。

②GopFastUpload 関数※で①をファイル名"IMAGE.JPG"で GOP に転送します。

※GOP-4000 シリーズは GopUpload 関数を使用します。

③GopWrite_b 関数で b0000 に 1 を書込みます。

以下のサンプルでは仮想ホスト上のボタン bt0 を押すごとに以下の 4 枚の画像を順に転送します。


```
01:#include <stdio.h>
02://仮想ホストの定義ファイル
03:#include "../dummysys/dummyhard.h"
04://GopLib の定義ファイル
05:#include "../goplib.h"
06://エラーハンドル用
07:void err(int ni,void *param)
08:{
09:  GopResetErr();
10:}
11:void main(){
12:  int framenum=0;
13:  //GOP ライブラリの初期化
14:  GopInitLib(commGetc,commPutc,TimeCount);
15:  GopSetErrFunc(err); //エラーハンドラの登録
16:  PortSetRate(115200);//通信レートセット
17:  SetResponseMode(1);//応答モード設定
18:  while(1){
19:      int bt=GetBtn();
20:      if(bt){//ボタンが押された
21:          char fstr[256];
22:          FILE *fp;
23:          while(GetBtn());//ボタンが離されるまで待つ
```

管理番号	C06621A-Z080B

```
24:             //転送元ファイル名の生成
25:             sprintf(fstr,"PIC¥¥B%03d.JPG",framenum);
26:             fp=fopen(fstr,"rb");//転送元ファイルを開く
27:             if(fp!=NULL){
28:                 //ファイルオープン成功
29:                 char *buf;
30:                 long size;
31:                 //ファイルサイズ取得。ファイル末尾へ SEEK しファイルポインタを取得。
32:                 fseek(fp,0,SEEK_END );
33:                 size=ftell(fp);
34:                 //取得後は先頭位置に SEEK
35:                 fseek(fp,0,SEEK_SET );
36:                 //取得したサイズに応じてメモリ確保
37:                 buf=(char *)malloc(size);
38:                 //取得したメモリにファイルイメージ読込み
39:                 fread(buf,1,size,fp);
40:                 //ファイル転送
41:                 GopFastUpload("IMAGE.JPG",buf,size);
42:                 //GopUpload("IMAGE.JPG",buf,size);//GOP-4000 シリーズはこちらを使用
43:                 //メモリの開放
44:                 free(buf);
45:                 //ファイルクローズ
46:                 fclose(fp);
47:                 //キャンバス描画軌道のため b0000 に 1 を書込み
48:                 GopWrite_b(0x0000,1);
49:                 //読込みビットマップ番号を進める
50:                 framenum=(framenum+1)%4;
51:             }
52:         }
53: }
54:}
```

説明

上記サンプル中のコメントを参照下さい。

3. GOP-4000 トレンドグラフ

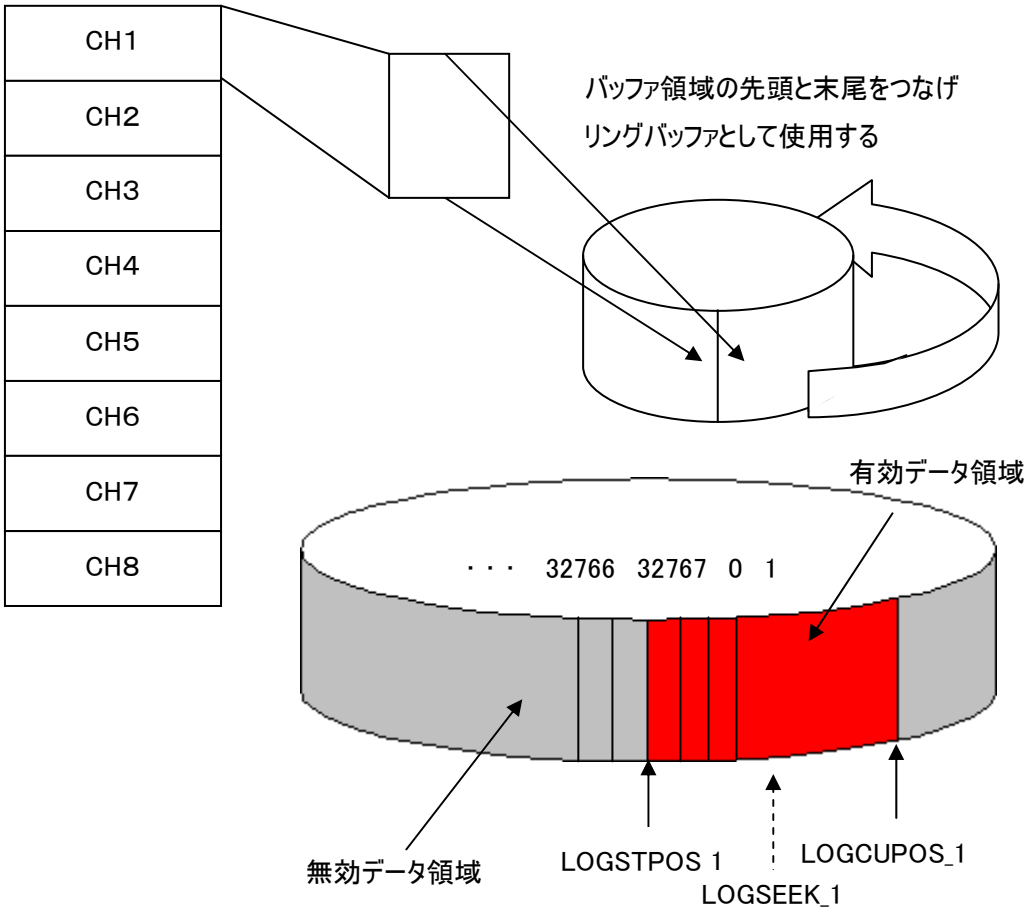
3. 1 トレンドグラフのしくみ

トレンドグラフはトレンド用のバッファ領域に書かれたデータをグラフ化し表示するオブジェクトです。

3. 1. 1 バッファメモリ

バッファは 8 チャンネルあり、トレンド専用の領域に割り当てられています。
各 CH ごとの領域は 64KByte で 1 データは w 型(2byte)ですので最大 32768 個のデータを記憶出来ます。
データの並びと、実際のアドレスとの関係は以下のようになります。
格納アドレス=[CH 毎のバッファの先頭アドレス]+[n(番目)]×2(Byte)
上記の格納アドレスに w 型で書込みます。
バッファのサイズは上記のとおり有限ですので、連続してデータをバッファにためていく場合、上記領域をリングバッファとして使用します。その際のバッファイメージを下記に示します。

CH1 を例とした時のバッファイメージ



バッファメモリの操作は以下のシステムメモリでのみ行います。

・LOGSTPOS_n(n は 1～8 CH)
　機能　有効バッファメモリの先頭(最古データ)を示します。
　　バッファデータが満杯時に LOGPUT_n に書込まれた時、自動で+1 されます。

・LOGCUPOS_n(n は 1～8 CH)
　機能　有効バッファメモリの最後尾(最新データ)を示します。
　　LOGPUT_n にデータが書込まれ、且つ SEEK 無効※1 の時、自動で+1 されます。

・LOGPUT_n(n は 1～8 CH)
　機能　LOGSEEK_n の位置にデータを書込みます。
　　WD/WH コマンドおよび MOV マクロで書込んで下さい。
　　(w 型以外でアクセスしないで下さい。)

・LOGGET_n(n は 1～8 CH)
　機能　LOGSEEK_n 位置のデータ値を取得します。
　　RD/RH コマンドおよび MOV マクロで読込んで下さい。
　　(w 型以外でアクセスしないで下さい。)

・LOGSEEK_n(n は 1～8 CH)
　機能　常に次データの位置を示します。
　　SEEK 有効時※は LOGPUT_n による書込み時、また LOGGET_n による読込み時に自動で+1 されます。
　　SEEK 無効時※は LOGPUT_n による書込み時 LOGCUPOS_n の値になります。

※SEEK の有効無効とは
有効データの範囲は変更せずに、範囲内のデータを上書きしたい場合に、SEEK を使用します。
LOGSTPOS_n <= LOGSEEK_n < LOGCUPOS_n の場合、SEEK 有効となります。
上記範囲外の時、書込みは SEEK 無効となります。

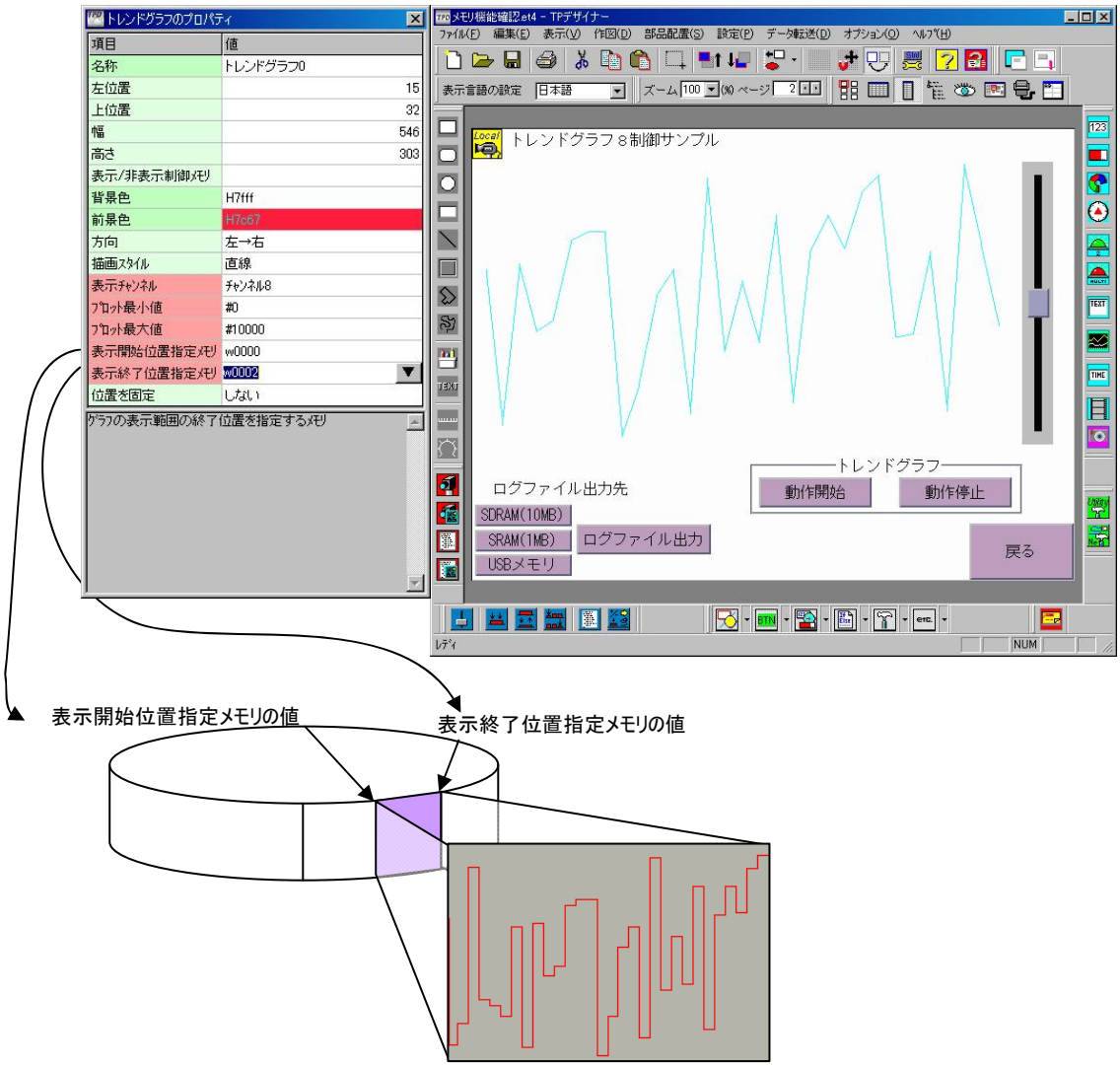
LOGSTPOS_n～LOGCUPOS_n 間のデータは有効データとしてトレンドグラフにプロットされます。
LOGSTPOS_n、LOGCUPOS_n、LOGSEEK_n とも 0～32767 を以外の設定は禁止です。

・CH1 バッファメモリにデータを入れる SEEK 動作例
　LOGSTPOS_1=0LOGCUPOS_1=0LOGSEEK_1=0
　↓　この時、LOGPUT_1 に 1,2,3,4,5 と 5 回データを入れる。
　↓　バッファメモリの状態は[1][2][3][4][5]...以降無効
　↓　この時点で LOGCUPOS_1 と LOGSEEK_1 は自動で増加し以下のようになります。
　LOGSTPOS_1=0LOGCUPOS_1=5LOGSEEK_1=5
　↓　この時、LOGSEEK_1=2 とし、LOGPUT_1 に 10,11,12,13,14 と入れる。
　↓　すると[2],[3],[4]のデータが入っているバッファメモリを[10],[11],[12]と上書きします。
　↓　LOGSEEK_1 が LOGCUPOS_1 に追いつくと更新書込に戻ります。
　↓　バッファメモリの状態は[1][2][10][11][12][13][14]...以降無効
　LOGSTPOS_1=0LOGCUPOS_1=7LOGSEEK_1=7 となります。

3. 1. 2 トレンドグラフ

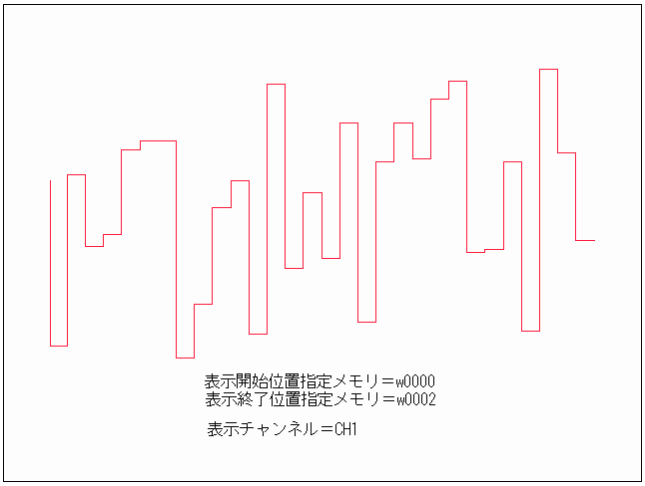
トレンドグラフはバッファメモリのうち一部をグラフ化し表示します。
※有効バッファメモリ間＝グラフ表示範囲ではありません。
表示する範囲は、表示開始位置指定メモリと表示終了位置指定メモリで指定します。
トレンドグラフの描画は LOGPUT_n にデータが入るタイミングで更新します。

使用例）
表示開始位置指定メモリに w0000，表示終了位置指定メモリに w0002 を指定し w0000 に 0、w0002 に 100 をセットすると、バッファの 0～100 番目の有効データをグラフ化します。



3. 2 トレンドグラフの表示手順

3. 2. 1 表示手順



上記のような設定のトレンドエリアにデータプロットする場合、以下の手順でメモリを設定します。

①トレンドバッファメモリの初期化
LOGCUPOS_1,LOGSTPOS_1,LOGSEEK_1 を 0 に設定します。

②w0000,w0002 にプロットエリアの指定
プロットする幅を 100 とした場合 w0000 に 0,w0002 に 100 を設定します。

③バッファメモリにデータを書込む
LOGPUT_1 に w 型で書込みます。先頭アドレス+LOGCUPOS_1*2(Byte)の位置にデータが書込まれます。

④GOP-4000 内部演算により LOGCUPOS_1 と LOGSEEK_1 が1増加
増加した結果が 32768 以下になるように実際には+1 した値と 32768 との剰余をセットします。
トレンドグラフの描画更新はこのタイミングで行われます。

データをプロットし続けていくと、表示領域をオーバーします。
この場合、判定方法は LOGCUPOS_1 が w0002 より大きくなったら表示範囲オーバーとなります。
このときにプロット位置を先頭に戻すか、表示画面をスクロールさせるかの処理が必要です。

・これまでのデータを有効にしたままプロット位置を 0 に戻して上書きしていく場合 LOGSEEK_1 だけを 0 に戻します。
・これまでのデータを無効にしプロット位置を 0 に戻す場合 LOGCUPOS_1 だけを 0 に戻します。
・そのままデータを入力し続け表示領域をスクロールさせる場合表示画面をスクロールさせるには以下のような処理が必要です。

⑤w0000 と w0002 をスクロールする量だけインクリメントする。
ここでも 32767 を越えないように実際にはインクリメントした結果と 32768 の剰余をセットします。

⑥一度バッファが一杯になった以降は、最古データが移動します。(LOGSTPOS_1=1+LOGCUPOS_1)

3. 2. 2 マクロを使った動作例

上記の処理をマクロで記述すると以下のようになります。
動作は 1 秒間隔で w0010 の値をサンプリングしグラフにプロットします。

[ページ表示時実行マクロ]

```
;バッファの初期化を行います
;①の処理
MOV LOGSTPOS_1 #0
MOV LOGCUPOS_1 #0
MOV LOGSEEK_1 #0
;②の処理
MOV w0000 #0
MOV w0002 #100
```

[監視オブジェクトのマクロ リンクメモリ:SECOND 比較条件の無効化:する]

```
;バッファに w0010(サンプリング対象メモリ)の値を書込み
MOV LOGPUT_1 w0010
IF LOGCUPOS_1=0002 THEN
    ;⑤の処理
    ;ここでは 1 プロット分進めています
    EXPR w0000=(w0000+#1)%#32768
    EXPR w0002=(w0002+#1)%#32768
ENDIF
```

3. 2. 3 トレンドログのファイル出力機能

GOP-4000 はファイルシステムを搭載していますので、トレンドのログデータをファイル出力する事が可能です。

・トレンドファイル作成の手順
使用するシステムメモリは以下の通りです。
CURRENT_DRIVE ファイルを書込むドライブを選択します。
0揮発性 RAM ドライブ 10MB
1不揮発性 RAM ドライブ 1MB
2挿入している USB メモリ
LOGSAVE 書込むトレンドのチャンネル番号

[マクロ例]
```
MOV CURRENT_DRIVE #0
MOV LOGSAVE #1
```

上記操作で、ドライブ 0 のルートディレクトリに trendn.csv(n はチャンネル番号)ファイルが作成されます。
USB ケーブルを PC に繋ぐ事などで作成したファイルを取得出来ます。
CSV ファイルには LOGSTPOS_n～LOGCUPOS_n 間のデータが順に入ります。

3. 2. 4 基本的な使用方法

ここでは、最低限トレンドグラフの表示を行う為に必要な手順を示します。
TP-Designer を開き、適当な場所にトレンドグラフオブジェクトを配置して下さい。

プロパティシートの空欄を埋めて下さい。

プロット最小値
グラフ化する数値の取りうる最低値を入力
プロット最大値
グラフ化する数値の取りうる最大値を入力
表示開始位置指定メモリ
0 を入力
表示終了位置指定メモリ
100 を入力

この設定で、画面データを転送します。
チャンネル1を指定していますので、LOGPUT1 メモリに値を入力していきます。
値を受け付けると画面左より値がグラフ化されて表示されます。

この設定の場合、データは100個分まで表示出来ます。
それ以降は使用形態によって変わりますのでマクロプログラムによって制御して下さい。

3. 2. 5 ホストプログラムからの使用例

ホスト側からデータを送信する場合は以下のようになります。

動作としては 100ms 毎にスライダーの状態をサンプリングし GOP に送信しグラフを描画します。

```
//仮想ホストの定義ファイル
#include "../dummysys/dummyhard.h"
//GopLib の定義ファイル
#include "../goplib.h"
void main(){
    int oldSendTime=0;//直前の送信時刻を記憶
    //ホスト側で LOGCUPOS_1,LOGSTPOS_1,表示開始位置指定メモリ(w0000),
    //表示終了位置指定メモリ(w0002)の値管理用変数を以下のように取得
    int cupos=0//LOGCUPOS_1 値管理用
       ,stpos=0//LOGSTPOS_1 値管理用
       ,startpos=0//w0000 値管理用
       ,endpos=100;//w0002 値管理用
    int sf=0;//リングバッファオーバー確認フラグ
    //GOP ライブラリの初期化
    GopInitLib(commGetc,commPutc,TimeCount);
    //GOP をリセットします
    GopSendCommand("UC");
    Wait(5000);
    //①の処理
    GopWrite_w(0xf104,stpos);
    GopWrite_w(0xf106,cupos);
    GopWrite_w(0xf108,0);
    //②の処理
    GopWrite_w(0x0000,startpos);
    GopWrite_w(0x0002,endpos);
    while(1){
        if(TimeCount()>oldSendTime+100){//100ms 間隔でコマンド描画
            //③の処理
            //ホストのスライダーの値を取得しデータとして書込み
            GopWrite_w(0xF100,GetSlider()*100/255);
            //④の処理
            cupos=(cupos+1)%32768;
            //書込み位置が表示範囲を超えたかどうかの判定
            if(cupos>endpos){
                //⑤の処理
                startpos=(startpos+1)%32768;
                GopWrite_w(0x0000,startpos);
                endpos=(endpos+1)%32768;
                GopWrite_w(0x0002,endpos);
                //⑥の処理
                //バッファ一巡したかの確認
                if(endpos<startpos)sf=1;
                //一巡した以降であれば BUF0_STPOS もインクリメント
                if(sf==1){
                    stpos=(stpos+1)%32768;
                    GopWrite_w(0xf104,stpos);
                }
            }
            oldSendTime=TimeCount();
        }
        Wait(10);
    }
}
```