



アプリケーションノート

品 名 Graphic Operation Panel V3

型 式 ISD-001(BM) V3/ISD-002 V3/ISD-202 V3
 GOP-32V

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ていただきます様、お願いします。

初版作成日	本書作成日	技術部			品質保証	
		承認	確認	担当	承認	確認
2005 年 5 月 6 日	2007 年 7 月 7 日	平坂	藤井(伸)	藤井(隆)	浅野	海野
備考						

管理番号

C04691A-Y005B

改定履歴表

改定番号	改定年月日	改定内容	担当	承認
－	2005 年 5 月 6 日	初版	藤井(隆)	藤井(伸)
－	2005 年 5 月 23 日	量産版初版	藤井(隆)	藤井(伸)
A	2005 年 8 月 9 日	2-4(3)、(4)項 CP、APD コマンドの制限による変更	藤本	藤井(隆)
B	2007 年 7 月 7 日	GOP 機能追加による記述変更 ホスト編 GOP シミュレータをデフォルトに記述変更	島田	藤井(伸)
備考				

目次

1. 画面設計	5
1-1 オブジェクトの複数選択による設定	5
1-1-1 複数選択設定: 大きさの統一	6
1-1-2 複数選択設定: メモリ設定	6
1-2 オブジェクトの整列	7
1-3 オブジェクトのコピー/貼り付け	8
1-3-1 同一ページへのオブジェクトコピー/貼り付け	8
1-3-2 異なるページへのオブジェクトコピー/貼り付け	9
1-3-3 複数起動時でのコピー	9
1-4 パーツの書き出しと読み込み	10
1-4-1 パーツの書き出し	10
1-4-2 パーツの読み込み	11
1-5 クリップボードからのビットマップデータ読み込み	12
1-6 クリップボードからのビットマップフォントデータ読み込み	15
1-7 グループ化	16
1-7-1 オブジェクトのグループ化	16
1-7-2 グループ化した個々のオブジェクトを編集	16
1-7-3 不可視設定	17
1-8 オブジェクトの階層	19
1-9 動的オブジェクトの背景描画について	20
1-10 再描画リンク機能	21
2. マクロ	24
2-1 注意事項	24
2-2 マクロ作成に関して	24
2-3 サンプルマクロ(基礎編)	25
2-4 サンプルマクロ(応用編)	29
3. ホストアプリケーション構築例	38
3-1 GOP 通信ライブラリについて	38
3-1-1 使用方法	38
(1) ホスト側であらかじめ用意する必要のある API	38
(2) 設定パラメーター	40
3-1-2 API 説明	40
① GopInitLib	40
② GopSendCommand	40
③ GopWrite_?	40
④ GopRead_?	41
⑤ GopSetFuncTbl	41
⑥ GopCheckMsg	42
3-2 仮想ホスト	43
3-2-1 仮想ホストの仕様	43
3-2-2 仮想ホストの API	43
① commGetc	43
② commPutc	43
③ TimeCount	44
④ Wait	44
⑤ SetTimerFunc	44
⑥ GetSlider	44
⑦ SetLevel	44
⑧ SetCT	44
⑨ SetInfo	44
⑩ GetInfo	45

⑪SetLamp	45
⑫GetLamp	45
⑬GetBtn	45
3-3 サンプルアプリケーション	46
3-3-1 ホストから GOP のメモリに値を書込む	46
3-3-2 ホストから GOP のメモリから値を読み込む	47
3-3-3 ホストから GOP のメモリに一定間隔で値を書込む	50
3-3-4 ホストから GOP のメモリに一定間隔で値を書込む	51
3-3-5 GOP のボタン押下イベントをホストで取り込む	53
4. トレンドグラフ	56
4-1 トレンドグラフのしくみ	56
4-1-1 バッファメモリ	56
4.1-2 トレンドグラフ	58
4.2 トレンドグラフの表示手順	59
4.2-1 表示手順	59
4.2-2 マクロを使った動作例	60
4.2-3 ホストプログラムからの使用例	62

1. 画面設計

1-1 オブジェクトの複数選択による設定

オブジェクトを複数選択した場合、オブジェクトの共通項目がプロパティシートに表示されます。共通項目を入力することにより、全てのオブジェクトに設定内容が反映されますので、同様の内容にする場合に効率良く設定が行えます。

項目	値
左位置	
上位置	
幅	
高さ	32
スタイル	色指定
枠の表示	する
枠線の幅	2
枠色(左上)	H66
枠色(右下)	H49
押下時表示変更	しない
メモリで未知背景の制御	しない
背景制御あり	
枠内色	H62
枠内色(マップ)	(未指定)
押下時枠内色	H62
押下時枠内色(マップ)	(未指定)
メモリで未知制御を行う	しない
ラベルあり	
ラベル色制御あり	
ラベル	ジャンプ
ラベルサイズ	標準12
ラベル色	H00
押下時ラベル変更	しない
押下時ラベル	
押下時ラベル色	H00
配置	センター
インターロック制御を行う	しない
インターロックあり	
インターロック条件比較演算	
インターロック状態の表示	標準(ラベルエンボス化)
インターロック時表示色	HFF
インターロック時表示色(マップ)	(未指定)
音を鳴らす	鳴らす
長押し時間の設定	0
表示リスト	通常
位置を固定	しない
移動先ページ	0

大きさや位置関係など

スタイルや色関係など

制御内容やメモリ関係など

ジャンプ

ジャンプ

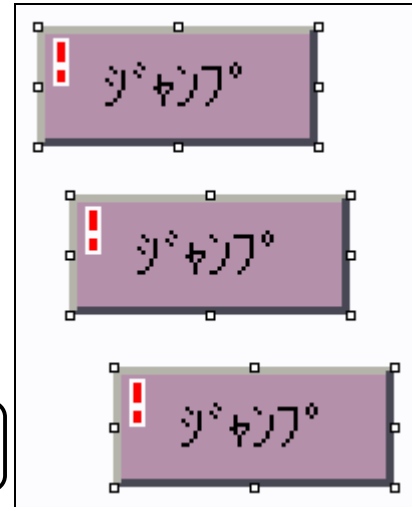
ジャンプ

【オブジェクトの複数選択とプロパティシート】

1-1 複数選択設定: 大きさの統一

複数選択した状態で幅と高さを入力すると、オブジェクト全てが設定された大きさに統一されます。

位置	
幅	70
高さ	30
スタイル	色指定



1-2 複数選択設定: メモリ設定

リンクメモリが同一の場合、複数選択を行い、メモリアドレスを入力することにより全てのボタンに反映されます。

位置で固定	しない
リンクメモリ	0000
セット値	



NOTE

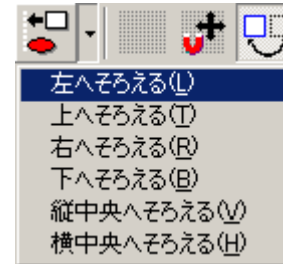
- その他にも複数選択時には位置やスタイル、動作等の設定が可能です。
- 異なるオブジェクトの場合でも、共通の設定項目であれば設定が可能です。
- 設定内容が異なる情報をもったオブジェクトを選択していくと、プロパティシートの当該項目は空白となります。

1-2 オブジェクトの整列

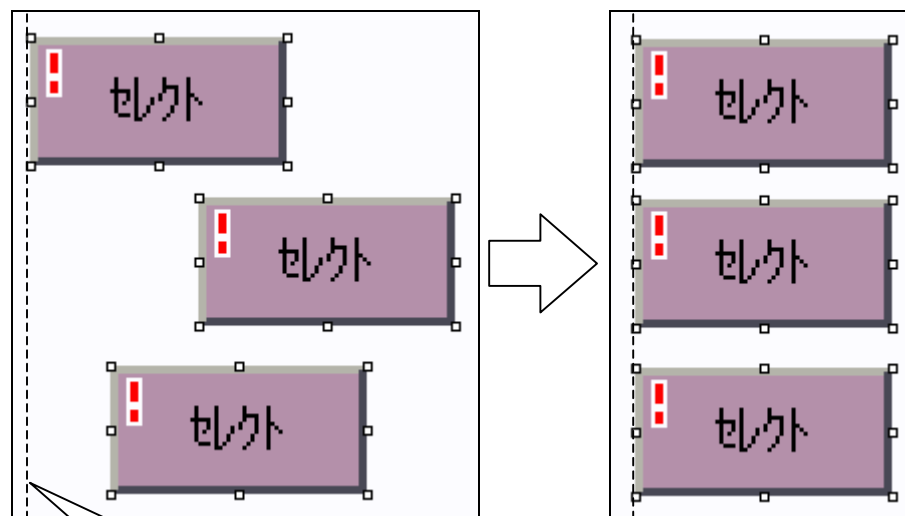
オブジェクトを作画エリアに無造作に配置しても、オブジェクトの整列機能を使えば、簡単にオブジェクトの整列が行えます。オブジェクトの整列はメニューバーの『編集』⇒『整列』の中、または下記アイコンから実行できます。



【メニューバーからの実行】



【アイコンからの実行】

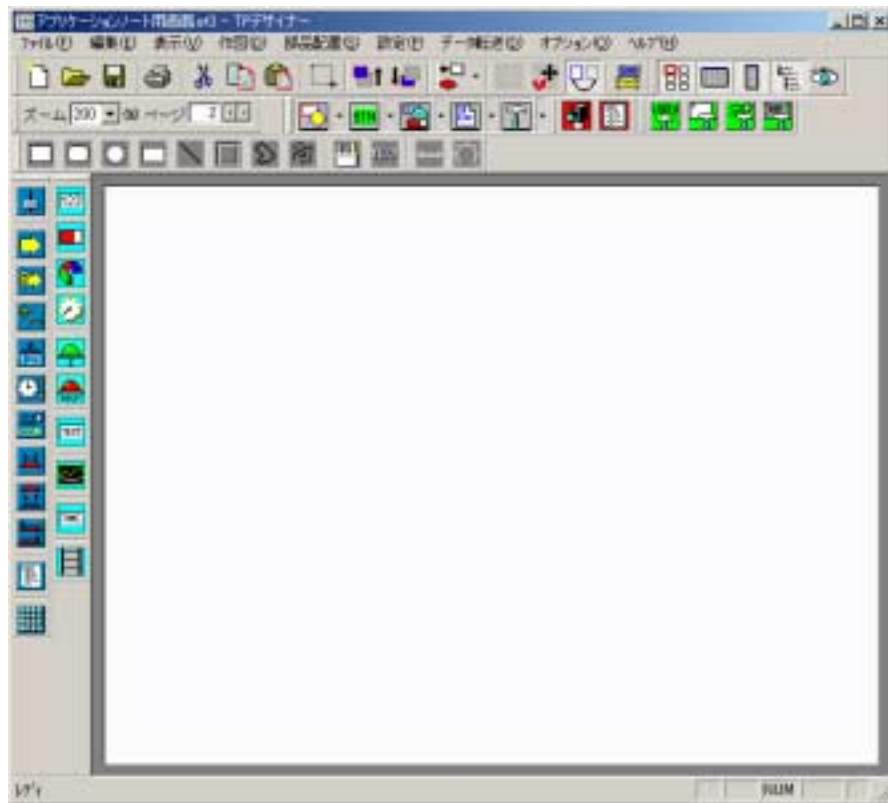


左に揃えるを選択
すると、このライン
に整列します。

1-3 オブジェクトのコピー/貼り付け

1-3-1 同一ページへのオブジェクトコピー/貼り付け

オブジェクトを選択し、ctrl+C やメニューバーから『編集』⇒『コピー』、右クリックによるメニューリストから『コピー』を選択すると、オブジェクトをコピーします。作画ウィンドウがアクティブな状態で ctrl+V やメニューバーから『編集』⇒『貼り付け』、右クリックによるメニューリストから『貼り付け』を選択すると、作画エリア上にオブジェクトを貼り付けることができます。

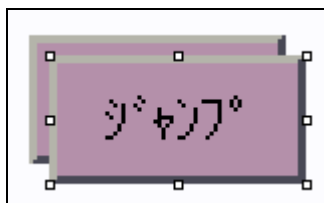


【作画ウィンドウ アクティブ状態】



【作画ウィンドウ 非アクティブ状態】

同一ページへオブジェクトをコピーし貼り付けた場合は、ベースのオブジェクトの右下へ5ドットの位置へ貼り付けられます。



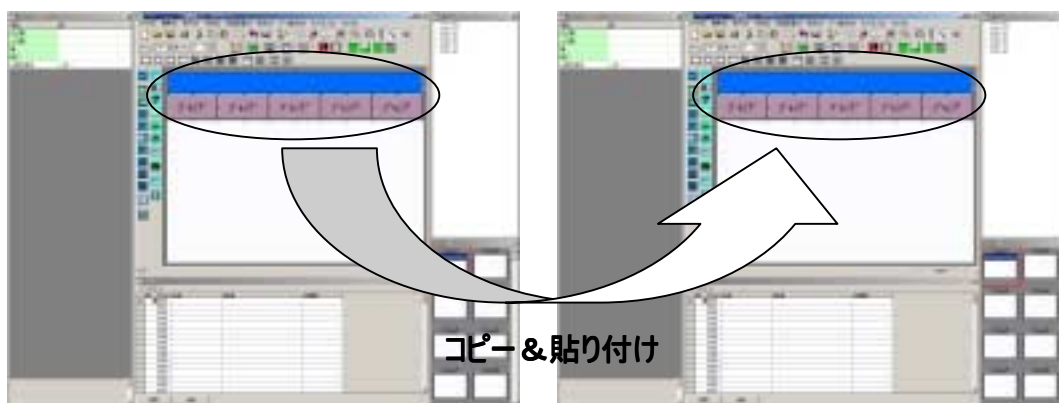
【同一ページでのオブジェクトコピー/貼り付け】

1-3-2 異なるページへのオブジェクトコピー/貼り付け

異なるページへオブジェクトを貼り付けた場合は、同一座標に貼り付けがされます。よって、イメージを崩す事なく、ページ作成を行うことができます。

1-3-3 複数起動時でのコピー

TP-Designer を複数起動し、別ファイル間のコピー/貼り付けが可能です。既存データから新規作成データなどから、コピー/貼り付けを行うことにより、新規にオブジェクトを作成する手間が省けます。



コピー&貼り付け

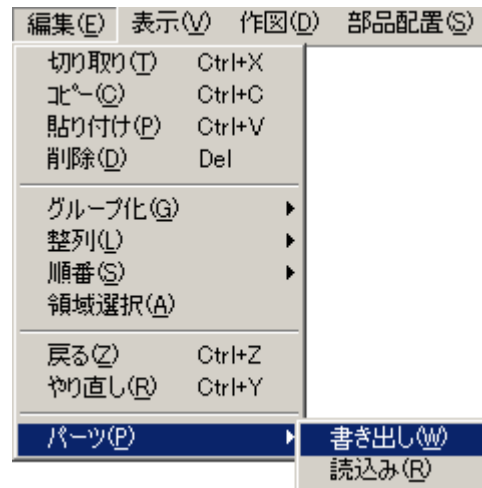
NOTE

- オブジェクトをコピーし、作画ウィンドウがアクティブな状態となっていない時にオブジェクトの貼り付けを行っても、実行はされません。作画ウィンドウをクリックし、アクティブな状態で貼り付けを行ってください。
- ボタンやランプなどのオブジェクトはプロパティシートに表示されている設定項目全てを引き継いで貼り付けられます。複数起動時でのコピー/貼り付けでは取得したメモリとの衝突にご注意ください。

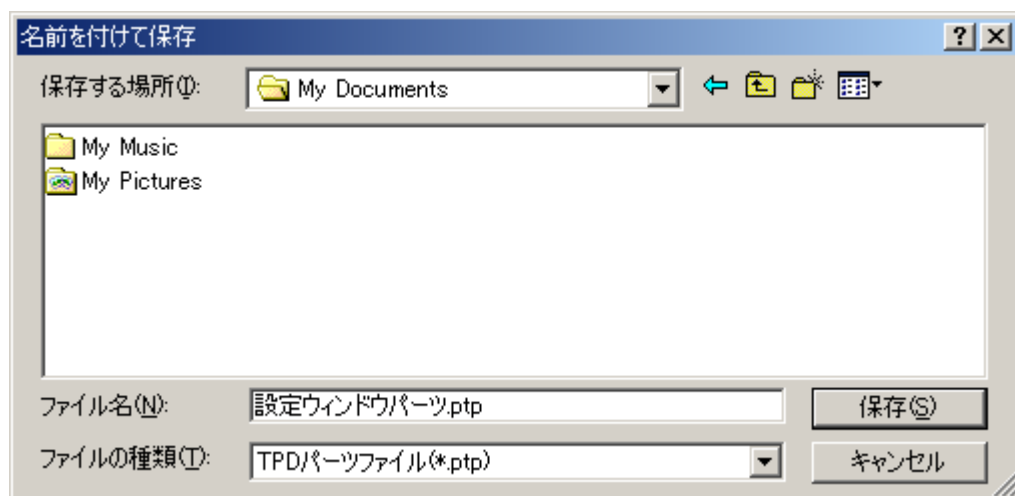
1-4 パーツの書き出しと読み込み

1-4-1 パーツの書き出し

TP-Designer V3 で作成したオブジェクトを別のパーツファイルとして書き出すことができます。書き出しは書き出しをしたいオブジェクトを選択した状態でメニューリストの『編集』⇒『パーツ』⇒『書き出し』を選択しファイル名を入力することにより実行されます。



【メニューバー 書き出し】



【ファイル名を指定し保存】

ファイル名を指定し保存をすると、拡張子.ptp がついたパーツファイルが作成されます。

NOTE

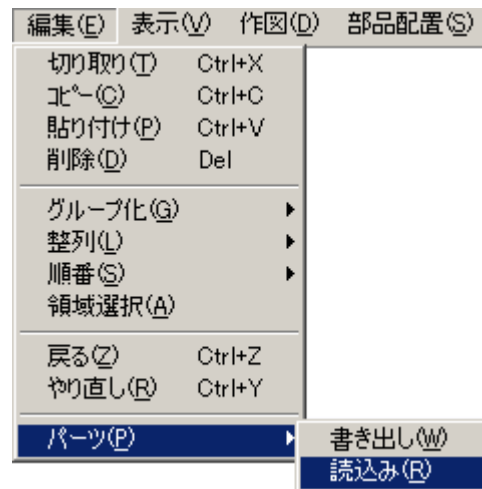
- 書き出されたパーツファイルは位置/スタイル情報やメモリ情報など全て書き出されています。
使いまわしするパーツはメモリ情報等をクリアした状態で書き出すことにより読み込み時にメモリの衝突が起こりません。

管理番号

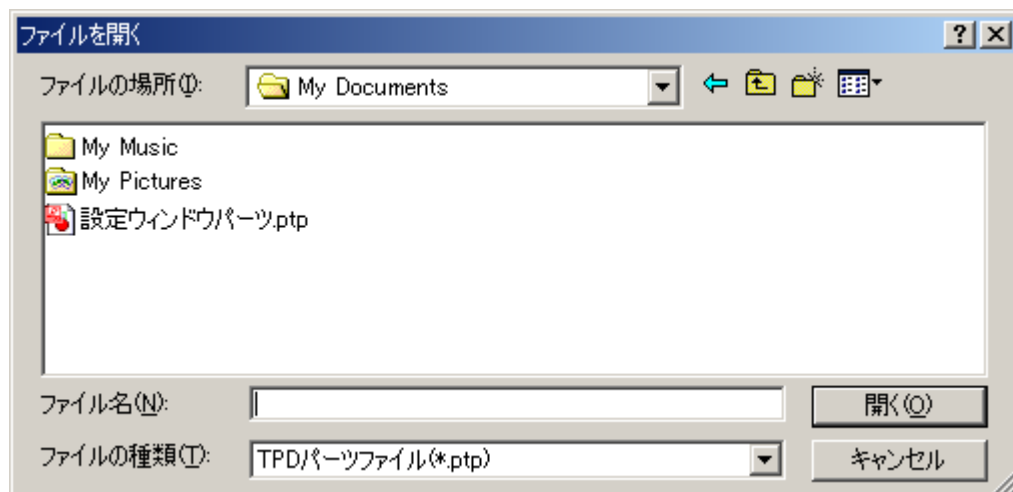
C04691A-Y005B

1-4-2 パーツの読み込み

TP-Designer V3 で作成したパーツファイルを読み込むことができます。読み込みメニューリストの『編集』⇒『パーツ』⇒『読み込み』を選択し、読み込みたいファイルを選択/開くことにより実行されます。



【メニューバー 読み込み】



【ファイルを選択し開く】

拡張子.ptp がついたパーツファイルを選択し開きボタンを押すと、作画ウインドウに読み込まれます。

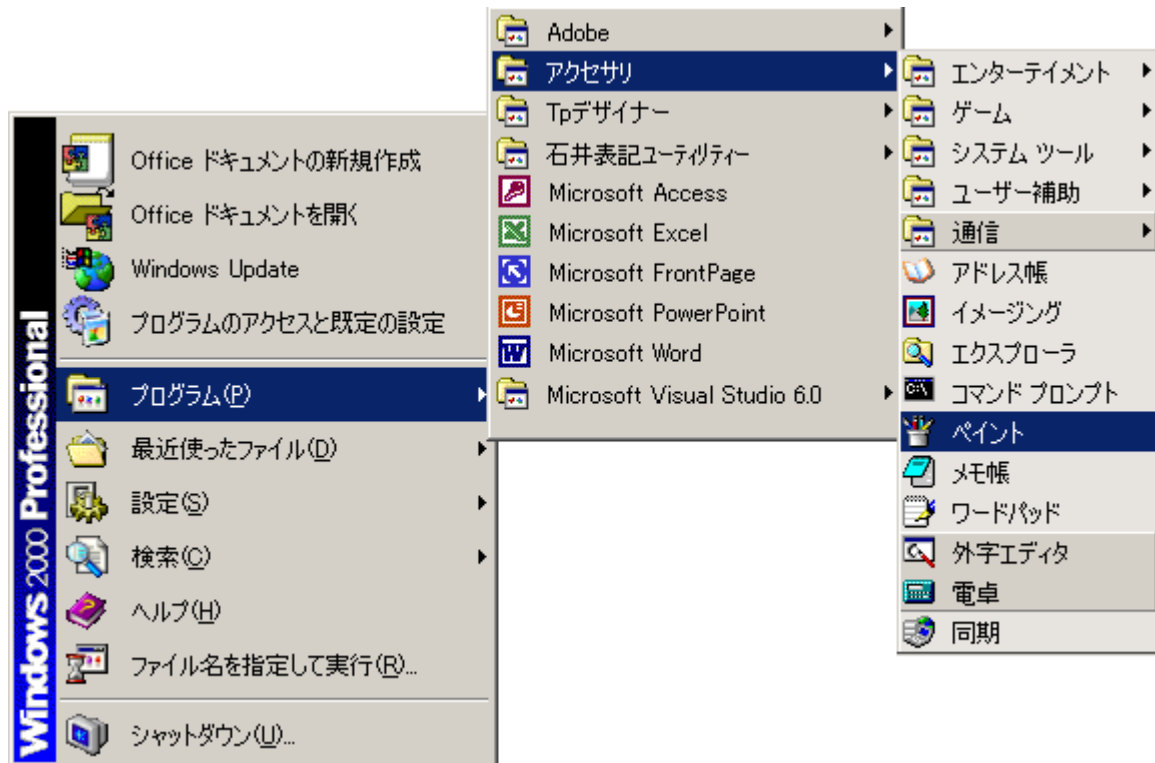
NOTE

- 読み込みしたパーツファイルは位置/スタイル情報やメモリ情報など全て反映され読み込まれます。
メモリ情報等の衝突には十分注意して下さい。
- 読み込みしたパーツファイルにはページ情報はありませので、現在表示している作画ウインドウに読み込まれます。

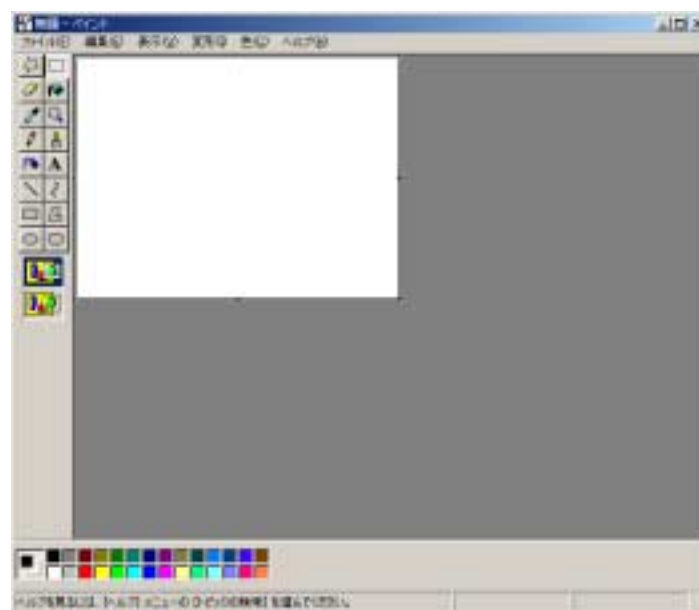
1-5 クリップボードからのビットマップデータ読み込み

Windows®標準ソフト『ペイント』でビットマップデータを作成し、クリップボードを経由することにより、簡単に TP-DesignerV3 へ読み込ませることができます。

※ペイントに限らず、その他の画像編集ソフト等からでも、同様の作業ができますが、コピーをした場合、クリップボードに貼り付けられるソフトであることを前提とします。

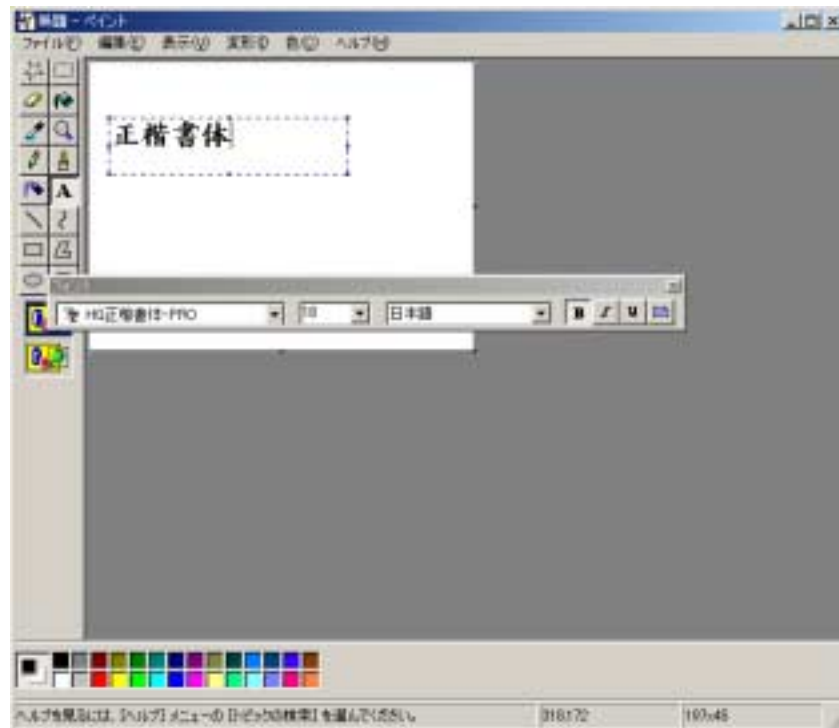


【ペイントの起動】



【Windowx® ペイント】

綺麗に見せたい文字などはビットマップで作成し、クリップボード経由で取り込みます。まず、ペイント上で、以下の様に文字を作成します。

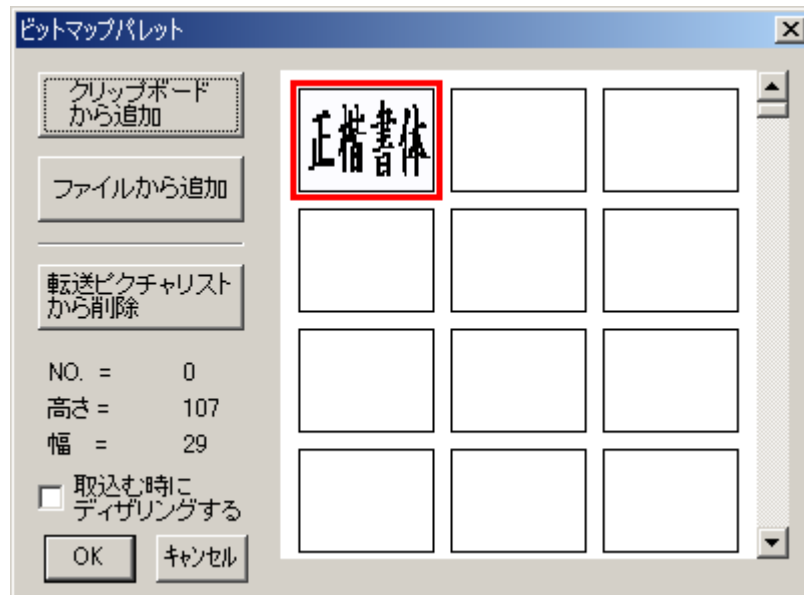


【文字の作成】

作成した文字を領域選択し、コピー (Ctrl + C) します。



TP-DesignerV3 のビットマップパレットを表示し、『クリップボードから追加』ボタンを押すと、先ほど作成したビットマップデータが読み込まれます。



【TP-Designer V3 ビットマップパレット】

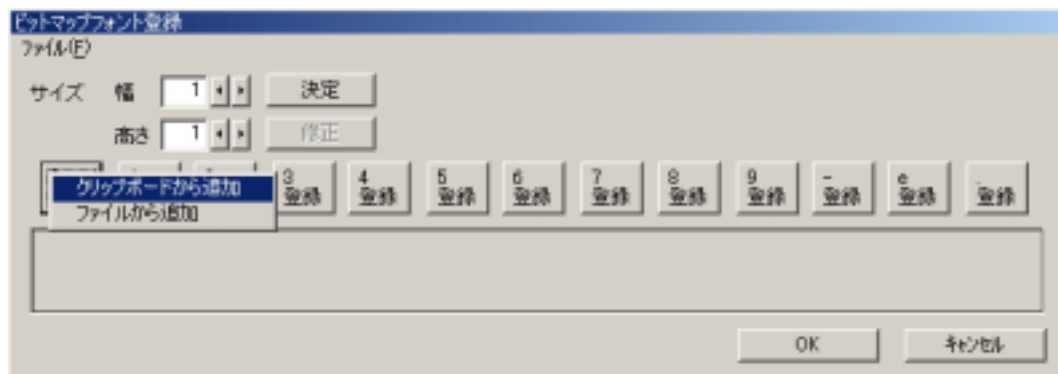
NOTE

○ビットマップデータはユーザーが作成された物をご使用ください。著作権により保護されたビットマップデータを無断で使用することは、法律で禁止されています。ビットマップデータは、ユーザー責任により、よくご確認の上、ご使用ください。

尚、ビットマップデータやその他画像データ等による著作権のトラブルが発生した場合、弊社は一切責任を負いません。

1-6 クリップボードからのビットマップフォントデータ読み込み

前項と同様にペイントソフトを使って数字を作成します。作成した数字をクリップボードにコピーし、ビットマップフォント登録ウィンドウから、数字を個々に登録していきます。



【ビットマップフォント登録ウィンドウ】

NOTE

○ビットマップデータはユーザーが作成された物をご使用ください。著作権により保護されたビットマップデータを無断で使用することは、法律で禁止されています。ビットマップデータは、ユーザー責任により、よくご確認の上、ご使用ください。

尚、ビットマップデータやその他画像データ等による著作権のトラブルが発生した場合、弊社は一切責任を負いません。

1-7 グループ化

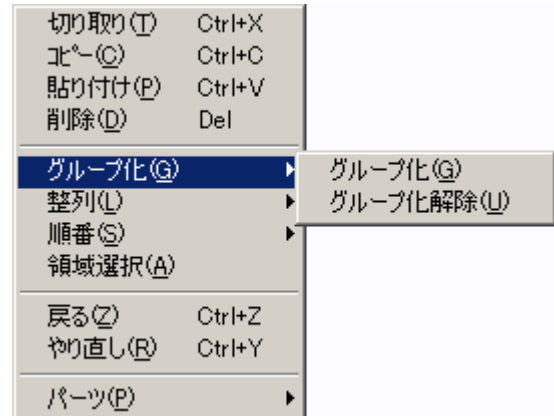
1-7-1 オブジェクトのグループ化

複数のオブジェクト同士をグループ化すれば、設計途上で配置したオブジェクト同士の位置が変わったりすることがありません。また、数の多いオブジェクトはグループ化することにより、操作スピードの向上が図れます。

グループ化したいオブジェクトを選択し、メニューリストまたはマウスの右クリックによるリストの中からグループ化を選択することにより、実行されます。



【メニューバー】



【マウス右クリック】

1-7-2 グループ化した個々のオブジェクトを編集

グループ化したオブジェクトはクリックするとグループ化した1つのオブジェクトになってしまう為、個々のオブジェクトの編集ができません。編集を行うには、オブジェクトツリービュー内より編集したいオブジェクトを選択し、プロパティシート上から編集してください。



【オブジェクトツリービュー】

1-7-3 不可視設定

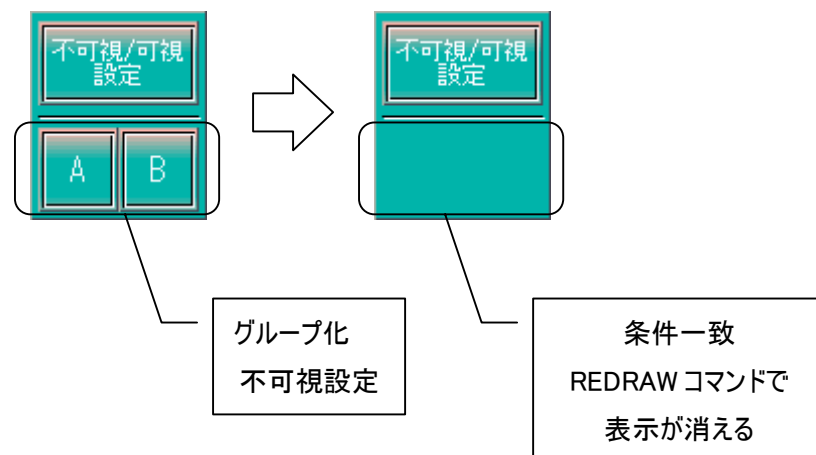
グループ化を行うと、グループとなったオブジェクト全体をメモリの状態により表示/非表示の制御が行えます。たとえオブジェクト1つであっても、グループ化することにより、この機能を使う事ができます。

項目	値
名称	グループ52
左位置	186
上位置	87
幅	80
高さ	42
可視設定メモリ	b002B
不可視設定値	1
位置を固定	しない

【グループ化状態のプロパティシート】

可視設定メモリ(数値型メモリ)を設定し、不可視設定値を入力します。メモリリンクした可視設定メモリの値が、不可視設定値と同じになった場合、グループ化されたオブジェクトが表示されません。不可視設定値が可視設定メモリの値と異なる場合は表示されます。

但し、表示/非表示を行うには、コマンドにより REDRAW をかける必要があります。REDRAW(再描画)をかけない場合は、表示に変化はありません。



REDRAW コマンドは監視オブジェクトを使って実行します。可視設定メモリの変化を監視し、変化があった場合、REDRAW が実行される仕組みにします。不可視設定されたオブジェクトが別ページにある場合は、REDRAW を行う必要がありません。ページ表示時に表示/非表示の状態が反映されます。

管理番号

C04691A-Y005B



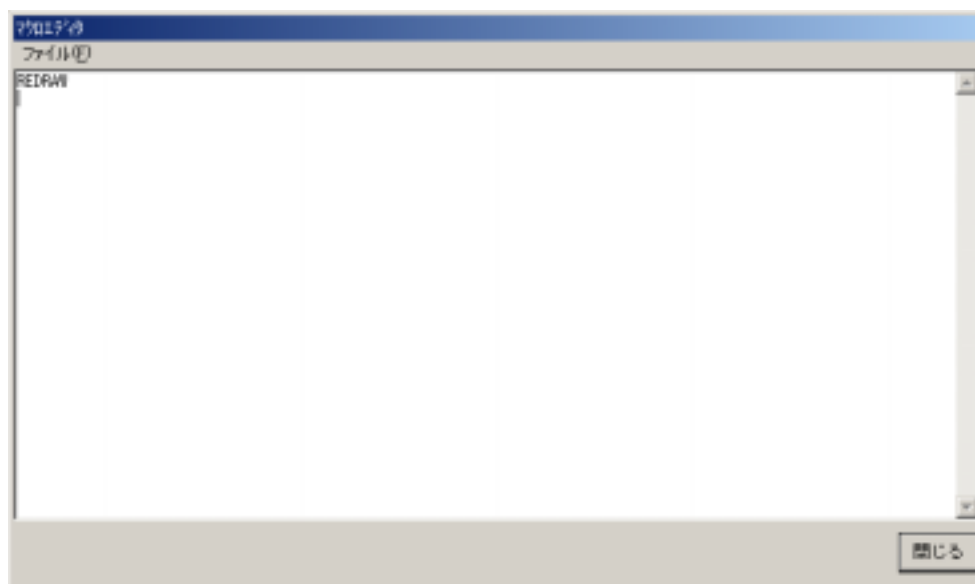
位置で固定	しない
リンクメモリ	b002B
条件	=(等しい)
比較値	
マスク	&HFFFFFFF
適用範囲	表示中
動作	REDRAW..
比較条件の無効化	する

【監視オブジェクトのプロパティシート】

監視オブジェクト

監視オブジェクトのリンクメモリは可視設定メモリと同一にします。また、比較値は入力せず、「比較条件の無効化」を「する」に設定すれば、条件比較を行わず、メモリに変化があれば、マクロの動作を行うようになります。

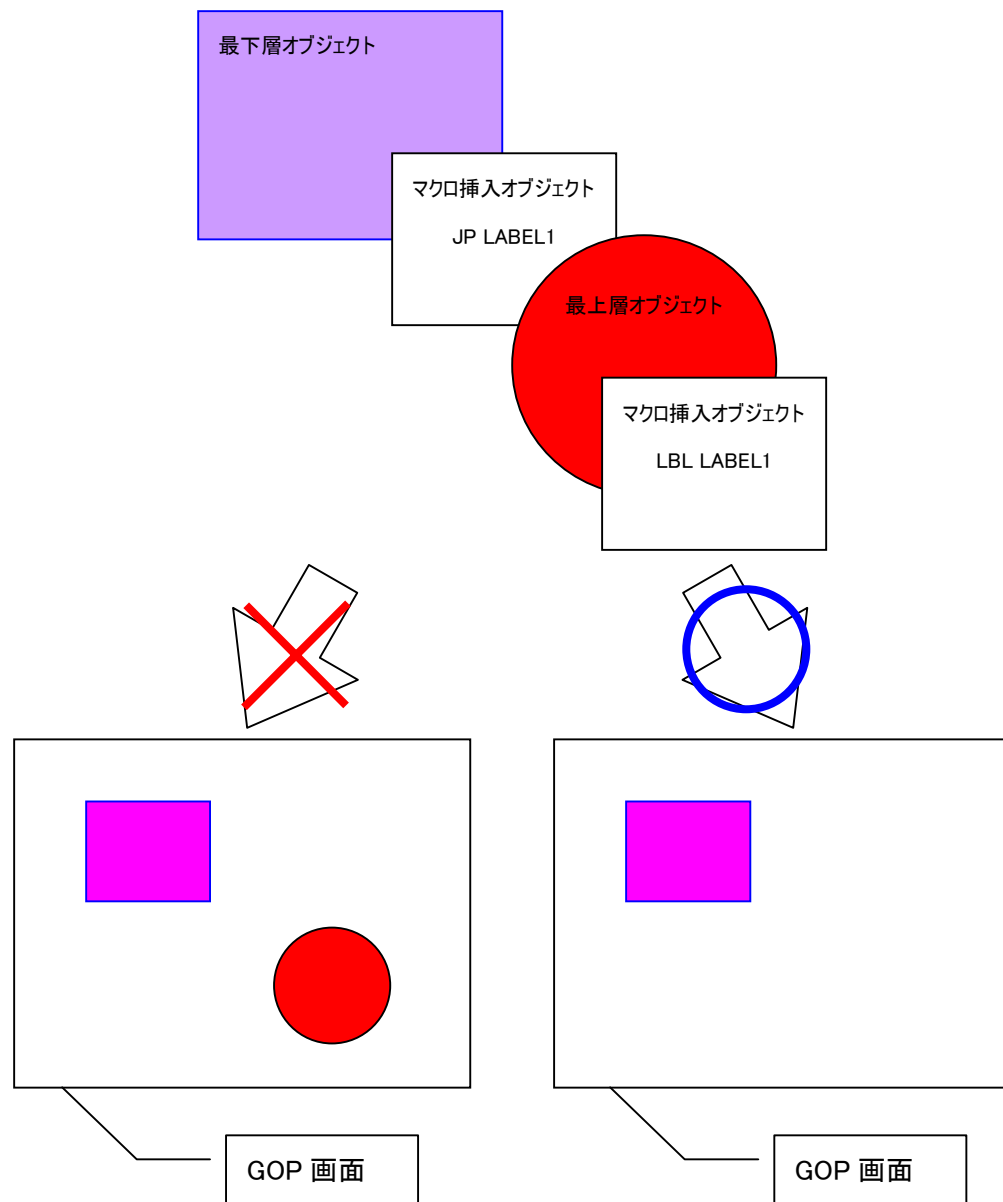
マクロのプログラムは以下の通りです。



1-8 オブジェクトの階層

TP-Designer で作成したオブジェクトの階層は非常に重要です。GOP に画面データを転送し、表示する際、表示は一気に行われますが、細かく言えば、オブジェクトの一番下の階層から順に実行されます。

初期化オブジェクトの組み合わせにより下記の用な画面データを作成した場合の例をご確認ください。

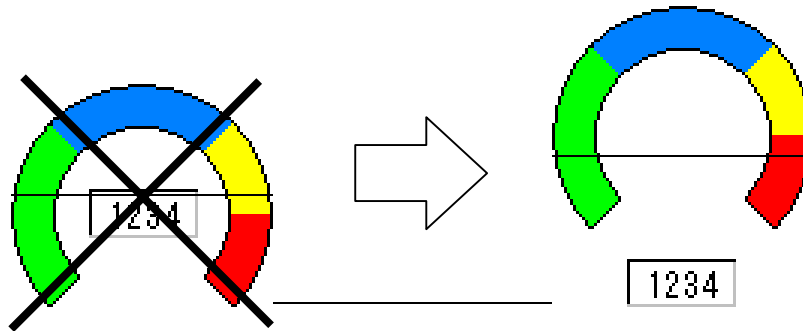


初期化オブジェクトのマクロが実行され、LABEL1 にジャンプしますので、以降のオブジェクトが実行されません。初期化オブジェクトの位置には十分注意してください。

1-9 動的オブジェクトの背景描画について

動的オブジェクトの背景は透明設定できません。よって、背景を透明設定しビットマップや絵柄がある場所に配置したりすることはできません。

但し、アナログメーターのみ、背景が再描画されますので、ビットマップや絵柄がある場所に配置しても、背景に影響がありません。



リングメーター全てが背景色で再描画されますので、中央に配置しているカウンタ等はリングメーターが動作すると消えてしまいます。リングメーターのオブジェクトより外側にオブジェクトを配置する様にしてください。

※再描画リンク機能により動的オブジェクトの重ね合わせもできるようになりました。

1-10 再描画リンク機能

オブジェクトを重ね合わせて画面をデザインする場合、従来ではメータの上にカウンタなどを配置すると



メータのメモリが変化するとメーターのみ描画されカウンタの表示が消えてしまい、重ねて配置できませんでした。



再描画リンク機能は、動的オブジェクトが描画を行うことで影響を受けるオブジェクトを同時に再描画する機能です。

上記の場合、メータを描画するタイミングで一緒にカウンタも描画を行うよう動作します。



再描画リンクの設定を行うには次の2つの設定が必要です。

- ①自身が描画を行うことにより他のオブジェクトの表示に影響を与えるオブジェクト(上記の場合はメータ)のプロパティでグループ内オブジェクト再描画リンクの設定を“する”に設定



- ②描画により影響を受ける他のオブジェクト(上記の場合はカウンタ)とグループ化します。

※描画により影響を受けるオブジェクトは本来自動で処理すべき物ですが、現行の GOP の処理能力上、グループ化により画面設計者より明示的に範囲を指定していただく必要があります。

再描画リンクにより以下のような部品が簡単に作成できます。

- ①ランプ付きボタン



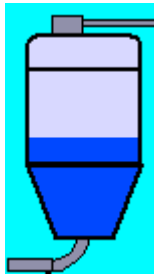
- 1) ボタンを配置します。
- 2) ボタンの上にランプを配置します。
- 3) ボタンの再描画リンクをするに設定します。
- 4) ボタンとランプをグループ化します。

- ②カウンタ付きメーター

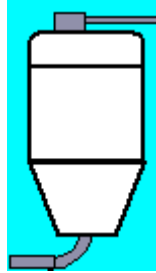


- 1) メータを配置します。
- 2) メータの上にカウンタを配置します。
- 3) メータの再描画リンクをするに設定します。
- 4) メータとカウンタをグループ化します。

③タンク残量計



1)タンクのビットマップを準備します。



2)メータを配置します。



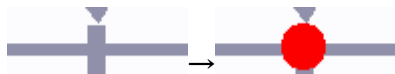
3)メータの再描画リンクをするに設定します。

4)メータの上にビットマップを配置します。

5)ビットマップを透明色サポートをするに設定し、タンク部の色(白:HFF)を透明色に指定します。

6)メータとビットマップをグループ化します。

④透明ランプ



1)背景になる画像を配置します。

※背景になる画像がない場合はランプの大きさ以上でページ背景色と同じ色の四角形を配置して下さい。

2)ランプを背景の上に配置します。

3)ランプの OFF 時の色を透明または、OFF 時のビットマップを未指定にします。

4)ランプの再描画リンクをするに設定します。

5)背景とランプをグループ化します。

※背景は、背景の再描画により他のオブジェクトに影響が出ない大きさにしてください。

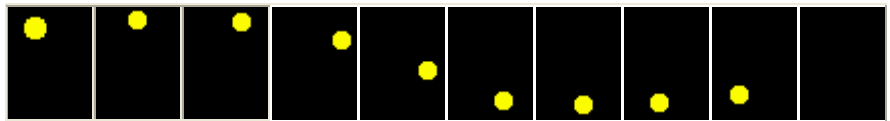
⑤透過アニメーションロゴ



1)ロゴになるビットマップを用意します



2)アニメーションのビットマップを用意します



3)ロゴのビットマップを配置します

4)アニメーションを配置します。

5)アニメーションの透明色サポートをするに設定し、透明色を黒(H00)に設定します。

6)アニメーションの再描画リンクをするに設定します

7)ロゴとアニメーションをグループ化します

※ロゴはアニメーションより大きくなるように設定し、アニメーションがロゴの領域からはみ出ないように配置して下さい。

2. マクロ

2-1 注意事項

記載のサンプルマクロは GOP V3 のマクロのプログラム例として記述するものであり、参考としてお取り扱い頂きますようお願いいたします。本サンプルマクロを使用したことにより生じた損害について弊社は一切責任を負いません。

本サンプルマクロは何れも一通りデバッグ済みですが、実際のご利用に当たっては必ず動作確認の上、貴社システム全体で十分に評価を行ない、お客様の責任において適用可否を判断してください。尚、弊社は、適用可否に対する責を負いませんので予めご了承ください。

2-2 マクロ作成に関して

マクロに使用するコマンド類は、TP-DesignerV3 取扱説明書の『マクロ』項に記載の内容に基づいて作成してください。また、マクロの記述に誤りがある場合は、GOP へ画面転送時にエラーが表示されます。その際に表示されるコメントは、エラーが発生している直前の定型文が表示されます。よって、マクロ内にこの定型文を細かく配置することにより、エラーの位置の限定が容易にできるようになります。

定型文 ;!で始まるコメント文

サンプルマクロ(応用編)項に記載のサンプルマクロは LBL で定義しており、共通サブルーチンなどに記述し、実行したい部分で SUB コマンド(サブルーチンコール)により呼び出すことにより使用することができます。また、サンプル中で使用している変数は重複定義はできませんので重複の場合は変数名を変更してください。

2-3 サンプルマクロ(基礎編)

ここでは基本的なコマンドの使用方法に関して記述します。

(1) w 型メモリに定数を代入

説明 w00F0 に 1 を代入します。
代入値が定数のため数値の前に#を付加します。

```
MOV w00F0 #1
```

(2) w 型メモリに 16 進数表記の定数を代入

説明 w00F0 に 16 進数表記の 1a を代入します。
代入値が定数のため数値の前に#を付加します。
16 進数をあらわす&H を付加します。

```
MOV w00F0 #&H1a
```

(3) w 型メモリに取得変数を代入

説明 w00F2 を TPD にて取得した変数を代入します。

```
;w 型で変数を取得する  
ADIM NUMBER w  
MOV w00F0 NUMBER
```

(4) w 型メモリに w 型メモリを代入

説明 w00F2 を w00F0 に代入します。

```
MOV w00F0 w00F2
```

(5) T 型メモリに格納される 10 進数の数値を表す文字列を数値に変換

説明 T011E に格納されている 10 進数表記の文字列を数値に変換し w00F0 に代入します。

【注意】

代入対象メモリが F 型以外の場合は小数点は無視されます。

12.34 → 1234

```
MOV w00F0 T011E
```

(6) w 型メモリに定数を加算

説明 w00F0 に 10 を加算します。

```
ADD w00F0 #10
```

(7) w 型メモリから定数を減算

説明 w00F0 から 10 を減算します。

```
DEL w00F0 #10  
;ADD w00F0 #-10 でも可
```

(8) w 型メモリに w 型メモリを掛算

説明 w00F0 に w00F2 を掛算します。

MUL w00F0 w00F2

(9) w 型メモリを定数で割算

説明 w00F0 を 10 で割算します。

DEV w00F0 #10

(10) w 型メモリを定数で剰余算

説明 w00F0 を 10 で割った余りを代入します。

MOD w00F0 #10

(11) w 型メモリを w 型メモリで論理和

説明 w00F0 を w00F2 で論理和をとります。

OR w00F0 w00F2

(12) w 型メモリを w 型メモリで論理積

説明 w00F0 を w00F2 で論理積をとります。

AND w00F0 w00F2

(13) w 型メモリを w 型メモリで排他的論理和

説明 w00F0 を w00F2 で排他的論理和をとります。

XOR w00F0 w00F2

(14) w 型メモリを論理反転

説明 w00F0 を論理反転する。

NOT w00F0

(15) T 型メモリに文字列をコピー

説明 T00F4 に文字列“石井表記”をコピーする

文字列のため先頭に\$を付加する。

CP コマンドの文字数は T 型メモリの取得領域を指定することにより
オーバーフローを防止します。(例では 21byte)

CP 21 T00F4 \$石井表記

(16) T 型メモリに T 型メモリを追記

説明

T00F4 に T0109 を追記します。

APD コマンドの文字数は追記される T 型メモリの取得領域を指定することによりオーバーフローを防止します。(例では 21byte)

```
APD 21 T00F4 T0109
```

(17) T 型メモリの一文字削除

説明

T00F4 を一文字削除します。

```
BS T00F4
```

(18) w 型メモリの数値を文字列化し、T 型メモリに代入

説明

w00F0 の数値を文字列に変換し、T0109 に代入します。

小数点の位置を指定することも可能です。小数点の扱いに関しては TP-DesignerV3 の取扱説明書を参照してください。

```
FMT 6 0 3 T0109 w00F0
```

(19) 数値比較を用いたサンプルマクロ

説明

w00F0 が 10 より大きく、w00F2 が 16 進数表記の 100 より大きいときに w00F0 と w00F2 を加算して I00EC に代入します。条件を満たさないときには I00EC に w00F0 を代入します。

```
IF w00F0>#10 AND w00F2>#&H100 THEN  
    EXPR I00EC=w00F0+w00F2  
ELSE  
    MOV I00EC w00F0  
ENDIF
```

(20) 文字列比較を用いたサンプルマクロ

説明

T00F4 が“石井表記”なら w00F0 に 10 を代入します。それ以外の場合には、w00F0 に w00F2 を代入します。

```
CMP = T00F4 $石井表記  
IF bf040=#1 THEN  
    MOV w00F0 #10  
ELSE  
    MOV w00F0 w00F2  
ENDIF
```

(21) 間接参照を用いたサンプルマクロ 1

説明 w00DE の値と 16 進数表記の E0 を加算した値をアドレスとする w 型のメモリに w00F0 を代入します。

w00DE の値を 2 加算すると型メモリの指定が可能です。

w00DE=0 のとき代入されるメモリ w00E0

w00DE=2 のとき代入されるメモリ w00E2

w00DE=4 のとき代入されるメモリ w00E2

となります。

【注意】

間接参照で使用するメモリ(ここでは w00DE)は w 型以外は指定できません。

```
MOV w(00DE+&H00E0) w00F0
```

(22) 間接参照を用いたサンプルマクロ 2

説明 前項で紹介したマクロでは w00DE を 2 ずつ変化させる必要がありますが、1 ずつの変化で同様のことを行う場合下記のようにします。

w00DE=0 のとき代入されるメモリ w00E0

w00DE=1 のとき代入されるメモリ w00E2

w00DE=2 のとき代入されるメモリ w00E2

となります。

【注意】

間接参照で使用するメモリ(ここでは w00DE)は w 型以外は指定できません。

```
;間接で使用する w 型メモリを取得
ADIM SAMPLE_ADDR w
;メモリを計算する
EXPR SAMPLE_ADDR=(w00DE*#2)+##H00E0
MOV w(SAMPLE_ADDR) w00F0
```

2-4 サンプルマクロ(応用編)

(1) ページ番号を指定しないで次のページにジャンプするマクロ

説明 現在のページ番号を格納するメモリを取得し、加算を行った結果を PAGE メモリに代入します。実行結果、次のページにジャンプします。

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;          次ページにページジャンプする
;;          ページ番号の仮置きメモリの取得
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;ページ格納用メモリ
ADIM PageNo w
LBL NEXT_PAGE
;255 ページ以上にならないように制限する
IF PAGE<#255 THEN
;現在のページ番号をページ格納用メモリにコピー
MOV PageNo PAGE
;ページ格納用メモリに1を加算
ADD PageNo #1
;PAGE メモリに新しいページ番号を代入
MOV PAGE PageNo
ENDIF
END

```

(2) タイマーを使用しカウントアップするマクロ

説明 ページ表示時(マクロ挿入オブジェクト)にダウンカウントタイマーを指定の値にセットします。監視オブジェクトにてダウンカウントタイマーが 0 のときのトリガにてメモリの値を加算し、再度指定の値をセットします。(ダウンカウントタイマのため、値をセットしないとトリガが発生しません。)

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;          特定のページで 800ms ごとに w0029 をインクリメントする
;;          100 になると 0 の戻し、ループする
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;サンプルタイマーの初期化
;下記 INI_SAMPLE をマクロ挿入オブジェクトにて呼び出す
;マクロ挿入オブジェクトはページの最前面に配置
LBL INIT_SAMPLETIMER
;TIMER1 に 8 を代入 (800ms ダウンカウント)
MOV TIMER1 #8
END

;下記 SAMPLETIMER を監視オブジェクトにて呼び出す
;比較条件を TIMER=0 とする
LBL SAMPLETIMER
;w0029 の値が 100 より小さいとき 1 加算
IF w0029<#100 THEN
ADD w0029 #1
;w0029 の値が 100 以上のとき 0 を代入
ELSE
MOV w0029 #0
ENDIF
;ダウンカウンタに値をセット
MOV TIMER1 #8
END

```

(3)シリアル 2 送信マクロ<通信サブルーチンを参照>

説明 シリアル 2 に T0000 の内容を送信します。

フォーマットは<STX>T0000=****<ETX><CSM><CRLF>となります。T0000 の値を上記フォーマットに基づいて送信します。送信は、送信文字列を SND2(Tf200)にコピーすることにより送信されます。

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;      シリアル 2 にメモリ (T0000)の内容を送信する
;;      フォーマット
;;      <STX>T0000=****<ETX><CSM><CRLF>
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
LBL SENDSERIAL2
;フォーマットに必要な文字列サブルーチンを呼び出す
SUB INI_SENDSERIAL2
;送信バッファに " STX " を追記
CP 100 SENDBUF STX
CP 99 SENDBUF STX
;送信バッファにアドレス名を追記
APD 100 SENDBUF $T0000=
APD 99 SENDBUF $T0000=
;送信バッファに指定アドレスの中身を追記
APD 100 SENDBUF T0000
APD 99 SENDBUF T0000
;送信バッファに " ETX " を追記
APD 100 SENDBUF ETX
APD 99 SENDBUF ETX
;チェックサムを取得するためチェックサム取得関数にバッファのアドレスを渡す
MOV GETBUF_ADDR &SENDBUF
;チェックサム取得関数を実行
SUB GET_CSM
;送信バッファにチェックサム取得関数実行結果を追記
APD 100 SENDBUF SUM_TXT
APD 99 SENDBUF SUM_TXT
;送信バッファに " CRLF " を追記
APD 100 SENDBUF CRLF
APD 99 SENDBUF CRLF
;送信メモリに送信バッファの内容を代入
CP 100 Tf200 SENDBUF
CP 99 Tf200 SENDBUF
END

```

(4) シリアル 2 受信マクロ

説明 トリガが立つと RCV2 の受信文字列を RCV2BUF にコピーし、STX 無い場合は、エラー処理を行います。ある場合は、コマンド部とチェックサム部に分割し、コマンドのチェックサムを計算します。計算したチェックサムと送信されたチェックサムの照合を行い、不一致の場合はエラー処理を行います。

使用方法は監視オブジェクトにて RCV2_FLAG=1 をトリガとし、実行するように画面を作成し、RCV2CMD に<コマンド>部が格納されます。

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;      シリアル 2 で受信を行う
;;      送信フォーマットは GOP のものと様とする
;;      フォーマット
;;      <STX><コマンド><ETX><CSM><CRLF>
;;      STX がない場合、チェックサムが適合しない場合は
;;      エラーとします。
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
ADIM i_RCV2 w
ADIM j_RCV2 w
ADIM t_RCV2 w
ADIM RCV2_STEP b
ADIM tmp_RCV2 w
;受信用バッファメモリ
ADIM RCV2BUF T 100
;受信用バッファアドレス
ADIM RCV2BUF_ADDR w
;コマンド部格納メモリ
ADIM RCV2CMD T 100
;コマンド格納メモリアドレス
ADIM RCV2CMD_ADDR w
;チェックサム格納メモリ
ADIM RCV2SUM T 3
;チェックサム格納メモリアドレス
ADIM RCV2SUM_ADDR w
;エラーフラグ
ADIM ERROR_FLAG b
LBL GETRCV2
;変数の初期化
    MOV RCV2_STEP #0
    MOV j_RCV2 #0
    MOV ERROR_FLAG #0
;バッファ、チェックサム部、コマンド部のアドレスを取得
    MOV RCV2BUF_ADDR &RCV2BUF
    MOV RCV2SUM_ADDR &RCV2SUM
    MOV RCV2CMD_ADDR &RCV2CMD
;RCV2 の内容をバッファにコピー
    CP 100 RCV2BUF RCV2
    CP 99 RCV2BUF RCV2
;STX がない場合、エラーフラグを立てて関数を終了
    IF b(RCV2BUF_ADDR) != #2 THEN
        MOV ERROR_FLAG #1
        JP GETRCV2_END
    ENDIF
;コマンド部を取得する
    FOR i_RCV2=#1 TO #100 STEP #1
;STX 以降、ETX までを 1byte づつ取得する
        EXPR t_RCV2 = RCV2BUF_ADDR + i_RCV2
        IF RCV2_STEP = #0 THEN
            EXPR tmp_RCV2 = RCV2CMD_ADDR + j_RCV2
;ETX の場合の動作
            IF b(t_RCV2) = #3 THEN
                EXPR tmp_RCV2 = RCV2CMD_ADDR + j_RCV2
;文字列の終了コードを代入
                MOV b(tmp_RCV2) #0
;次のステップへ
                MOV RCV2_STEP #1
;変数の初期化
                MOV j_RCV2 #0
            ELSE
                MOV b(tmp_RCV2) b(t_RCV2)
                ADD j_RCV2 #1
            ENDIF
        ELSE
;チェックサム部の取得

```

管理番号

C04691A-Y005B

```

                                EXPR tmp_RCV2 = RCV2SUM_ADDR + j_RCV2
                                MOV b(tmp_RCV2) b(t_RCV2)
                                ADD j_RCV2 #1
;CR にてループを抜ける
                                IF b(t_RCV2) = #13 THEN
;文字列の終了コードを代入
                                EXPR tmp_RCV2 = RCV2SUM_ADDR + j_RCV2
                                MOV b(tmp_RCV2) #0
                                EXIT
                                ENDIF
                                ENDIF
                                NEXT
;チェックサム取得のため、GETBUF_ADDR にバッファのアドレスを代入
                                MOV GETBUF_ADDR RCV2BUF_ADDR
;チェックサム取得関数実行
                                SUB GET_CSM
;取得したチェックサムと受信したチェックサムを比較
                                CMP = RCV2SUM SUM_TXT
;不一致の場合、エラーフラグを立てて関数を終了
                                IF CMPFLG!=#1 THEN
                                    MOV ERROR_FLAG #1
                                    JP GETRCV2_END
                                ENDIF
                                LBL GETRCV2_END
;受信フラグをクリア
                                MOV RCV2_FLAG #0
END
```


(5) リアル 2 受信のコマンド処理マクロ<通信サブルーチンを参照>

説明 シリアル 2 にて受信した内容をコマンド処理し、実行します。

受信フォーマットは<STX><WD T**** *~~~><ETX><CSM><CRLF>ですが、サブルーチン(GETRCV2)によりコマンド列<WD T**** *~~~>部のみが渡されます。このコマンド列を 1byte ずつ比較しセパレータのスペースおよび文字列の終端により、コマンド部、メモリ部、データ部に分解し、処理を行っています。

但し、エラーが発生している場合は、何も処理を行いません。

エラー条件 サブルーチンで記載

- ・<STX>が無い
- ・<CSM>不一致

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;      シリアル 2 受信のコマンド処理
;;      送信フォーマットは GOP の"WD"コマンドで
;;      T 型メモリに文字列を書き込む
;;      フォーマット
;;      <STX><WD T**** *~~~><ETX><CSM><CRLF>
;;      *はそれぞれ任意アドレスと任意文字列を示す
;;      RCV_FLAG=1 をトリガとした監視オブジェクトを作成し
;;      SUB GETRCV2 → SUB COMMANDRCV2 を実行する
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
ADIM i_CMD w
ADIM j_CMD w
ADIM t_CMD w
ADIM tmp_CMD w
ADIM CMD_STEP b
;コマンド部
ADIM CMD0 T 2
;コマンド部アドレス
ADIM CMD0_ADDR w
;メモリ部格納
ADIM CMD1 T 6
;メモリ部アドレス
ADIM CMD1_ADDR w
;データ部
ADIM CMD2 T 100
;データ部アドレス
ADIM CMD2_ADDR w
;書き込みアドレス
ADIM ADDRESS w
LBL COMMANDRCV2
;エラーでないときの処理
IF ERROR_FLAG != #0 THEN
;コマンド部、メモリ部、データ部のアドレスを取得
MOV CMD0_ADDR &CMD0
MOV CMD1_ADDR &CMD1
MOV CMD2_ADDR &CMD2

;使用する変数の初期化
MOV CMD_STEP #0
MOV j_CMD #0

;コマンド分割
FOR i_CMD=#0 TO #100 STEP #1
;分割するため、1byte ごとにアドレスを進め、データを分割
EXPR t_CMD = RCV2CMD_ADDR + i_CMD
IF CMD_STEP=#0 THEN
EXPR tmp_CMD = CMD0_ADDR + j_CMD

;スペースまでをコマンド部に格納
IF b(t_CMD)=#&H20 THEN

;文字列の終端文字を代入
MOV b(tmp_CMD) #0

;変数の初期化
MOV j_CMD #0

;次のステップへ
MOV CMD_STEP #1

ELSE
MOV b(tmp_CMD) b(t_CMD)
EXPR j_CMD = j_CMD + #1
ENDIF
ENDIF
ELSEIF CMD_STEP=#1 THEN
EXPR tmp_CMD = CMD1_ADDR + j_CMD

;スペースまでをメモリ部に格納
IF b(t_CMD)=#&H20 THEN

;文字列の終端文字を代入
MOV b(tmp_CMD) #0

```

管理番号

C04691A-Y005B

```

;変数の初期化
;次のステップへ
MOV j_CMD #0
MOV CMD_STEP #2
ELSE
MOV b(tmp_CMD) b(t_CMD)
ADD j_CMD #1
ENDIF
ELSE
EXPR tmp_CMD = CMD2_ADDR + j_CMD
IF b(t_CMD)=#0 THEN
MOV b(tmp_CMD) #0
MOV j_CMD #0
MOV CMD_STEP #0
EXIT
ELSE
MOV b(tmp_CMD) b(t_CMD)
ADD j_CMD #1
ENDIF
ENDIF
NEXT
;コマンド処理
;コマンド比較 ( "WD" かどうか )
CMP = CMD0 $WD
;コマンド部が "WD" の時の処理
IF CMPFLG = #1 THEN
;メモリ部のアドレス番号を 1byte ごと取り出す
FOR i_CMD=#1 TO #100 STEP #1
EXPR t_CMD = CMD1_ADDR + i_CMD
IF b(t_CMD) = #0 THEN
EXIT
ENDIF
MOV CHAR b(t_CMD)
;取り出したアドレス番号を数値 (10 進数) に変換
SUB CHARTONUM
;桁上げ
MUL ADDRESS #16
;1byte 分の数値を加算する
ADD ADDRESS NUM
NEXT
;数値化した T 型のアドレス番号にデータ部をコピー
CP 81 T(ADDRESS) CMD2
ENDIF
ENDIF
END

```

(6)通信部サブルーチン

① STX を生成するマクロ

説明 使用する場合は STX で呼び出す。

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; STX 関数
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;STX 用のメモリ
ADIM STX T 2
;STX のアドレス
ADIM STX_ADDR w
LBL SET_STX
    MOV STX_ADDR &STX
    MOV b(STX_ADDR) #2
    MOV b(STX_ADDR+1) #0
END

```

② ETX を生成するマクロ

説明 使用する場合は ETX で呼び出す。

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; ETX 関数
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;ETX 用のメモリ
ADIM ETX T 2
;ETX のアドレス
ADIM ETX_ADDR w
LBL SET_ETX
    MOV ETX_ADDR &ETX
    MOV b(ETX_ADDR) #3
    MOV b(ETX_ADDR+1) #0
END

```

③ CRLF を生成するマクロ

説明 使用する場合は CRLF で呼び出す。

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; CRLF 関数
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;CRLF 用のメモリ
ADIM CRLF T 3
;CRLF 用のメモリ
ADIM TXTADDR w
LBL SET_CRLF
    MOV TXTADDR &CRLF
    MOV b(TXTADDR) #13
    MOV b(TXTADDR+1) #10
    MOV b(TXTADDR+2) #0
END

```

④ 送信初期化マクロ

説明 送信バッファ用のメモリを取得し、必要な関数をコールする。
この場合は、上記の①～③をコールしています。

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; 送信初期化関数
;; 最初に一度実行するように画面に組み込む
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;送信バッファ用のメモリ
ADIM SENDBUF T 100
LBL INI_SENDSERIAL2
    SUB SET_STX
    SUB SET_ETX
    SUB SET_CRLF
END

```

⑤ チェックサムを取得するマクロ

説明 GETBUF_ADDR により取得されるアドレスの 2 番目から (STX を除くため) ETX までのチェックサムを計算します。チェックサムの計算は、XOR とし、最終的に 1 byte のチェックサムデータを 16 進数表記の文字列データに変換しています。変換したデータは SUM_TXT に格納されます。

使用方法は取得したいアドレスを GETBUF_ADDR に代入し、関数を実行し、結果を SUM_TXT にて取得します。

```

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;      チェックサム関数
;;      STX から ETX までのチェックサムを取得
;;      GETBUF_ADDR に取得するアドレスを代入して実行
;;      SUM_TXT にて文字列化したチェックサムを返す
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
ADIM i_CSM w
ADIM t_CSM w
;チェックサムメモリ
ADIM SUM b
;チェックサム上位メモリ
ADIM SUM1 b
;チェックサム下位メモリ
ADIM SUM2 b
;チェックサムテキストメモリ
ADIM SUM_TXT T 3
;チェックサム計算用テキストメモリ
ADIM SUM_TMPTXT T 2
;バッファアドレスメモリ
ADIM GETBUF_ADDR w
LBL GET_CSM
;チェックサムの初期化
MOV SUM #0
;チェックサム取得
;取得は STX を除く 2 番目から ETX までとします
FOR i_CSM=#1 TO #100 STEP #1
    EXPR t_CSM = GETBUF_ADDR + i_CSM
    IF b(t_CSM) = #3 THEN
        EXIT
    ENDIF
;1byte ごとに XOR
XOR SUM b(t_CSM)
NEXT
;チェックサムのテキスト化
;16 進数表記の文字列に変換
EXPR SUM1 = SUM / #16
EXPR SUM2 = SUM % #16
IF SUM1>=#0 AND SUM1<#10 THEN
    FMT 1 0 3 SUM_TXT SUM1
ELSEIF SUM1=#10 THEN
    CP 1 SUM_TXT $a
ELSEIF SUM1=#11 THEN
    CP 1 SUM_TXT $b
ELSEIF SUM1=#12 THEN
    CP 1 SUM_TXT $c
ELSEIF SUM1=#13 THEN
    CP 1 SUM_TXT $d
ELSEIF SUM1=#14 THEN
    CP 1 SUM_TXT $e
ELSE
    CP 1 SUM_TXT $f
ENDIF
IF SUM2>=#0 AND SUM2<#10 THEN
    FMT 1 0 3 SUM_TMPTXT SUM2
ELSEIF SUM2=#10 THEN
    CP 1 SUM_TMPTXT $a
ELSEIF SUM2=#11 THEN
    CP 1 SUM_TMPTXT $b
ELSEIF SUM2=#12 THEN
    CP 1 SUM_TMPTXT $c
ELSEIF SUM2=#13 THEN
    CP 1 SUM_TMPTXT $d
ELSEIF SUM2=#14 THEN
    CP 1 SUM_TMPTXT $e
ELSE
    CP 1 SUM_TMPTXT $f
ENDIF

```

```
;文字列化したチェックサムを SUM_TXT に格納
APD 10 SUM_TXT SUM_TMPTXT
END
```

⑥ 16 進数表記の 1 文字を 10 進数数値に変換するマクロ

説明 使用法は CHAR に変換したい文字列を代入し、関数を実行。
結果を NUM にて取得します。

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; 16 進数表記の 1 文字を 10 進数数値に変換する
;; CHAR に変換する文字を代入して実行
;; NUM にて数値化したデータを返す
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;変換する文字列用メモリ
ADIM CHAR b
;変換後の数値用メモリ
ADIM NUM b
LBL CHARTONUM
; '0' ~ '9' の変換
IF CHAR>=#&H30 AND CHAR<=#&H39 THEN
    EXPR NUM = CHAR - #&H30
; 'a' ~ 'f' の変換
ELSEIF CHAR>=#&H61 AND CHAR <= #&H66 THEN
    EXPR NUM = CHAR - #87
; 'A' ~ 'F' の変換
ELSEIF CHAR>=#&H41 AND CHAR <= #&H46 THEN
    EXPR NUM = CHAR - #55
ENDIF
END
```

3. ホストアプリケーション構築例

3-1 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できます。

現状の制限事項は以下の通りです。

- ・通信仕様として応答確認なし、ハードウェアフロー制御なしのみサポート
- ・ブロックリード/ライト未サポート
- ・GOP 側からのメモリ値の自動送信未サポート
- ・メモリ読み込みの遅延取得未サポート

※遅延取得 メモリ読み込み時通信応答を待つが、応答待ちが許されないアプリケーションを作成する場合読み込みコマンド送信後すぐにルーチンを抜け、読み込みが完了した場合コールバックで通知する仕組み。

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ください。

3-1-1 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 文字を受け取る

受け取るべきデータがない場合は EOF(-1)を返す。

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

実装について、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よい。

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す。最小間隔については特に規定しない。

※<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関数を使用します。

```
strncpy
```

```
strncat
```

```
strncmp
```

```
strlen
```

(2)設定パラメーター

ホスト側の環境により仕様メモリ容量調整のためいくつかのパラメータを調整できます。

GopLib.h 内の define 定義で以下のパラメータを設定します。

LINEBUFSIZE	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
SENDBUFSIZE	GopLib が使用する送信バッファのサイズを指定します (現状未使用)
TIMEOUT	応答確認あり時のタイムアウト時間を設定します (現状未使用)

3-1-2 API 説明

①GopInitLib

定義	void GopInitLib(int (*getcharctor)(),void (*putcharctor)(char),int (*timecount)());		
動作	GopLib の初期化		
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

②GopSendCommand

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		
引数	msg	:	送信する文字列

③GopWrite_?

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き込みます。		
引数	addr	:	書込み先の GOP のアドレス
	value	:	書き込むデータ

使用例

```
GopWrite_w(0x0000,100);
w0000 に 100 を書き込みます
```


④GopRead_?

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

※GopRead_?はマクロ定義です

動作 指定のアドレスのデータを読み込み buf で指定される領域に記憶します

引数 addr : 読み先の GOP のアドレス

buf : 読み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: 読み込み成功

0: 読み込み失敗

使用例

```
unsigned short val;
```

```
GopRead_w(0x0000,&val,300);
```

w0000 の値を読み込んで val に保存します

⑤GopSetFuncTbl

定義 void GopSetFuncTbl(pGOP_FUNC_TBL 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の登録

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_FUNC_TBL 型
配列へのアドレス

count : tbl の登録件数

GOP_FUNC_TBL 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 void 型で引数は無し

使用例

```
void hoge1(){
```

```
...
```

```
}
```

```
void hoge2(){
```

```
...
```

```
}
```

```
const GOP_FUNC_TBL 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 が呼ばれる。

⑥GopCheckMsg

定義 char *GopCheckMsg();

動作 シリアルを受信を監視し、必要に応じハンドラ関数を呼び出します。

使用例

```
main(){
```

```
.....
```

```
while(1){
```

```
//処理のメインループ
```

```
GopCheckMsg();
```

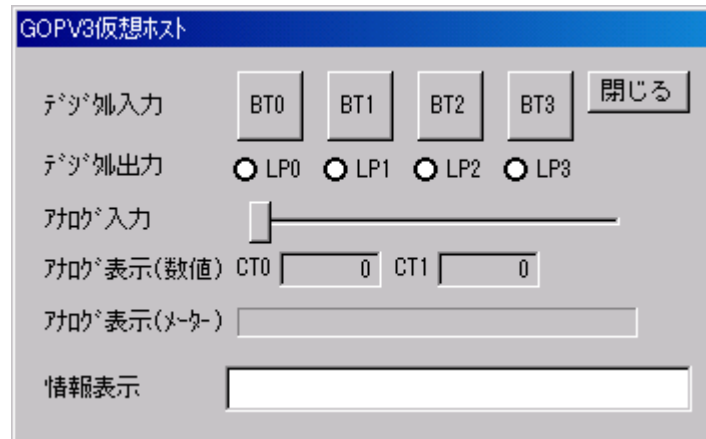
```
}
```

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。

3-2 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、ホストのプラットフォームが必要ですが、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ミュレータ(以下仮想ホスト)を使用し説明いたします。仮想ホストは VisualStudio6 でのビルド環境を用意しています。

3-2-1 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ください。

※使用可能なポートは SIM および COM1～COM5 までです

3-2-2 仮想ホストの API

①commGetc

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。

②commPutc

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

③TimeCount

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

④Wait

定義 void Wait(int i);

動作 指定時間待機します

⑤SetTimerFunc

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。

⑥GetSlider

定義 int GetSlider();

動作 アナログ入力値を取得

戻り値 0～255 の整数

⑦SetLevel

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(範囲を超えた場合の動作は保証しません)

⑧SetCT

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

⑨SetInfo

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

⑩GetInfo

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

⑪SetLamp

定義 void SetLamp(int state);

動作 デジタル出力の設定

引数 state : ランプの点灯状態
state の bit0～3 が LP0 から 3 に対応
ビットがたっていると ON

使用例

SetLamp(5)
LP0 と LP2 が点灯

⑫GetLamp

定義 int GetLamp();

動作 デジ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

⑬GetBtn

定義 int GetBtn();

動作 デジ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

3-3 サンプルアプリケーション

3-3-1 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 w0000 に 123

T0002 に "GOP のテストです"

F002b に 123.456

を書き込む

GOP 画面データ APP1.et3

```
w0000=00000  
T0002=テキストボックス0  
F002B=0000.0000
```

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み込みます。
- 4 行目: GopLib の定義ファイルを読み込みます
- 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 の立ち上がり待ちのため 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 型でデータを書き込みます
- 14 行目: GOP の 0002 番地に T 型でデータを書き込みます
- 15 行目: GOP の 002b 番地に F 型でデータを書き込みます

3-3-2 ホストから GOP のメモリから値を読み込む

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 の値を読み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.et3

w0000	00000	キーパッド 呼出
T0002	テキストボックス0	キーパッド 呼出
F002b	0000.0000	キーパッド 呼出

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,300)){

21: sprintf(temp,"w0000 は%d です",w_value);

22: }else strcpy(temp,"読み込み失敗");

23: SetInfo(temp);

24: break;

25: case 0x02:

26: if(GopRead_T(0x0002,t_value,300)){

27: sprintf(temp,"T0002 は¥"%s¥"です",t_value);

28: }else strcpy(temp,"読み込み失敗");

29: SetInfo(temp);

30: break;

31: case 0x04:

32: if(GopRead_F(0x002b,&f_value,300)){

33: sprintf(temp,"F002b は%f です",f_value);

34: }else strcpy(temp,"読み込み失敗");

35: SetInfo(temp);


```
36:                break;
37:            }
38:    }
39:}
```

説明

10 行目 : w 型メモリのデータ読み領域として 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 行から : BT0 が押された時の処理です

20 行目 : GOP の w0000 を読み出し 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 に読み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。

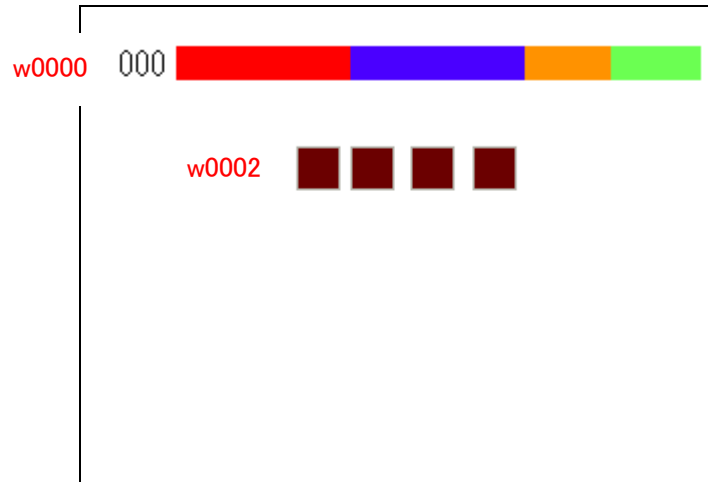
25 行目 : GOP の T0002 を読み出し t_value[] 配列にデータを格納します。基本的には 20 行目と同じですが、T 型メモリの格納先は配列になり t_value 自体がアドレスを表しますので &t_value とする必要はありません。

31 行目 : F002b を f_value に読み込みます。動作は 20 行目と同じです。

3-3-3 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.et3



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:   }
20:}

```

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(パーメーター)に書き込みます。

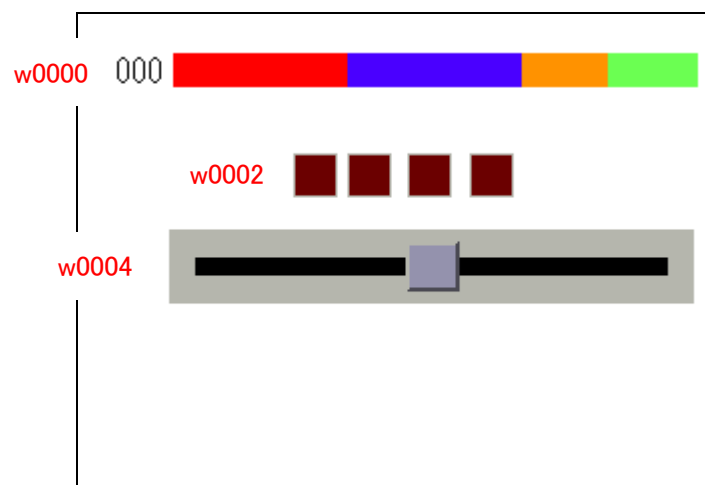
16 行目 : 仮想ホストのボタンの値を読み取り w0002(ランプ)に書き込みます。

17 行目 : 前回送信時刻を更新します

3-3-4 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.et3



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:    //GOP ライブラリの初期化
10:   GopInitLib(commGetc,commPutc,TimeCount);
11:   //GOP をリセットします
12:   GopSendCommand("UC");
13:   Wait(2000);
14:   while(1){
15:       if(TimeCount()>oldSendTime+50){
16:           GopWrite_w(0x0000,GetSlider());
17:           GopWrite_w(0x0002,GetBtn());
18:           if(GopRead_w(0x0004,&w_value,300)){
19:               SetLevel(w_value);
20:               SetCT(w_value,0);
21:           }
22:           oldSendTime=TimeCount();
23:       }
24:   }
25:}
```

説明

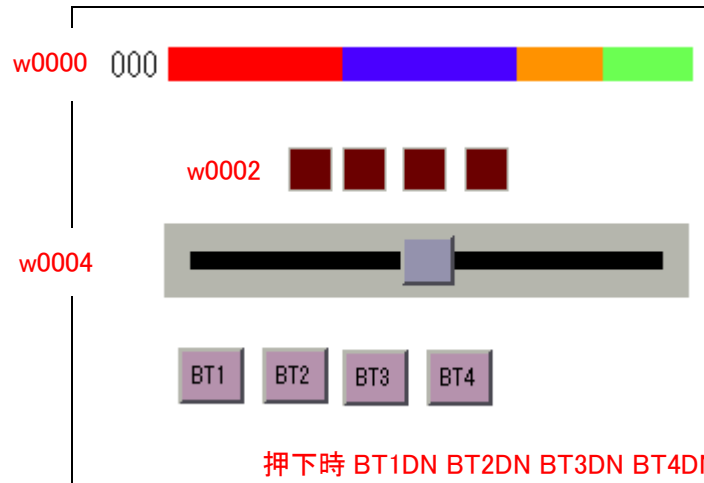
18 行目 : w0004 の値を読み込みます。読み込みが成功するとレベルメーターとカウンタに値を表示します。

3-3-5 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する。

※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だが、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.et3



押下時 BT1DN BT2DN BT3DN BT4DN

開放時 BT1UP BT2UP BT3UP BT4UP を通信出力

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(2000);
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,300)){
56:                    SetLevel(w_value);
57:                    SetCT(w_value,0);
58:                }
59:                oldSendTime=TimeCount();
60:            }
61:    GopCheckMsg();
62:    }
63:}
```

説明

6 行目から: “BT1DN” 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

29 行目まで: 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 行目から: 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_FUNC_TBL 型の配列して定義します。

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: で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。

4. トレンドグラフ

4-1 トレンドグラフのしくみ

4-1-1 バッファメモリ

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。

バッファは 8ch あり、GOP メモリの以下の領域に割り当てられています。

ch0	a000～
ch1	a800～
ch2	b000～
ch3	b800～
ch4	c000～
ch5	c800～
ch6	d000～
ch7	d800～

ch ごとの領域は 2048byte で 1 データは w 型(2byte)ですので最大 1024 のデータを記憶できます。

たとえば ch1 の 1 番目のデータは wa000,2 番目のデータは wa002 になります。

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

上記の格納アドレスを w 型で書き込みます。

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。

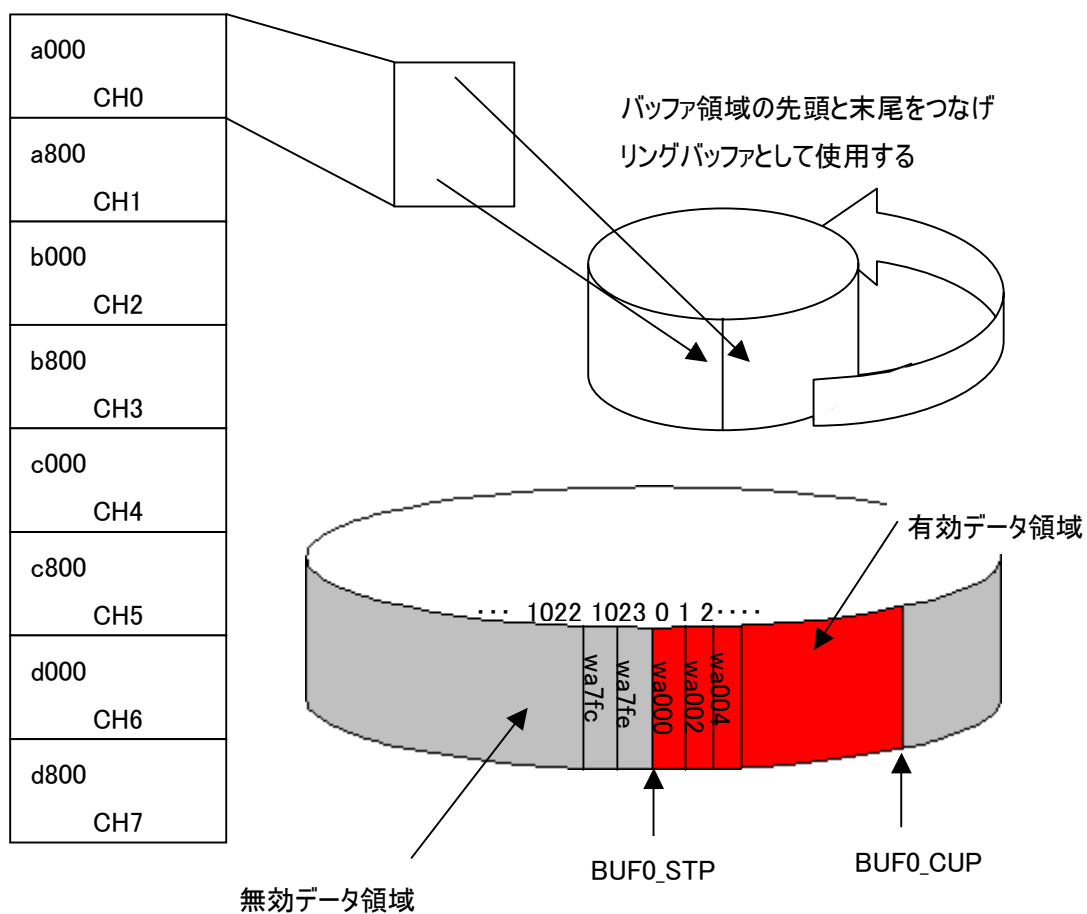
リングバッファの先頭(最も古いデータ)位置を BUF_n_STPOS(n は 0～7)、リングバッファの最後尾(最も新しいデータ)が記憶される位置を BUF_n_CUPOS というシステムメモリで管理します。

BUF_n_STPOS～BUF_n_CUPOS 間のデータは有効データとしてトレンドにプロットされます。

BUF_n_STPOS、BUF_n_CUPOS とも 1024 以上の値を設定してはいけません。

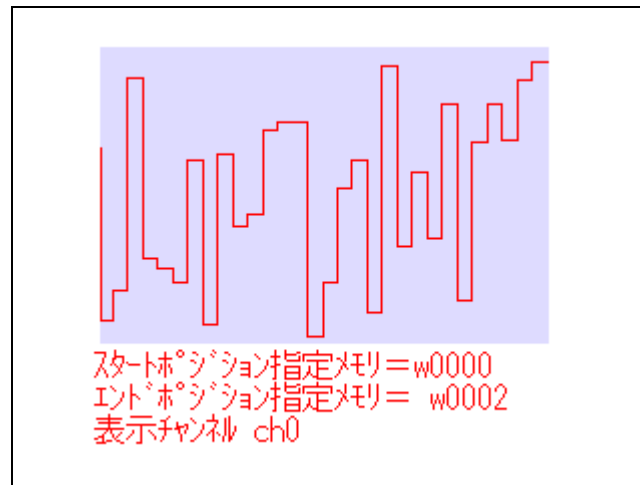
管理番号

C04691A-Y005B



4.2 トレンドグラフの表示手順

4.2-1 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 BUF0_CUPOS, BUF0_STPOS の初期化

BUF0_CUPOS, BUF0_STPOS ともに 0 に設定します。

② w0000, 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。

データをプロットするには以下の手順で行います。

③ バッファ領域にデータを書き込む

書き込むアドレスは $\text{Ha000} + \text{BUF0_CUPOS} \times 2$ の位置に w 型で書き込みます。

④ BUF0_CUPOS をインクリメントします。

インクリメントした結果が 1024 以下になるように実際にはインクリメントし 1024 で割ったあまりを BUF0_CUPOS にセットするようにします。

トレンドの描画はこの時点で行われます。

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。

判定方法は BUF0_CUPOS をインクリメントした結果が w0002 より大きくなったら表示範囲オーバです。

このときにプロット位置を先頭に戻すか、スクロールさせるかの処理が必要です。

プロット位置を先頭に戻す場合は BUF0_CUPOS を 0 に戻すだけでよいです。

スクロールさせるには以下のような処理が必要です。

⑤ w0000 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 1024 をオーバーしないように

実際にはインクリメントした結果を 1024 で割ったあまりをセットします。

- ⑥一度バッファが一杯になった以降は、BUF0_STPOS もスクロール量だけインクリメントします。

バッファが一杯になったかの判定は、フラグ用のメモリを確保し、初期化で 0 にしておきます。

- ①の処理結果 w0002 より w0000 が大きくなった場合バッファ満杯で表示範囲がリングの先頭に戻ったと判断できるのでここでフラグを 1 にセットします。

フラグが 1 であれば BUF0_STPOS をインクリメントするようにします。ここでも 1024 以下にするため 1024 で割ったあまりをセットします。

※この処理を行わないとバッファが満杯になった場合、最初にしたデーターが、プロットされる等の表示異常が発生します。

- ⑦トレンドの再描画のため、TREND_REDRAW コマンドを実行します。

ホストからトレンドを制御する場合は、直接 TREND_REDRAW コマンドを実行できないため、画面データ側で w0000(または w0002)にリンクした監視オブジェクトを作成しておき、w0000 の値変化があった場合 TREND_REDRAW を実行するようマクロ作成しておきます。

4.2-2 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 BUF0_STPOS #0

MOV BUF0_CUPOS #0

;②の処理

MOV w0000 #0

MOV w0002 #100

;⑥のバッファ満杯になったか確認用のフラグ用メモリの確保

ADIM scflug b

;⑥のフラグの初期化

MOV scflug #0

[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]

;書き込みアドレス計算用のメモリを確保

ADIM addr w

;③の処理

;書き込みアドレスの計算

EXPR addr=BUF0_CUPOS*#2+#&ha000

;バッファに w0010(サンプリング対象メモリ)の値を書き込み

MOV w(addr) w0010

;④の処理

EXPR BUF0_CUPOS=(BUF0_CUPOS+#1) % #1024

;書き込み位置が表示範囲を超えたかどうかの判定

IF BUF1_CUPOS>w0002 THEN

;⑤の処理

EXPR w0000=(w0000+#60)%#1024

EXPR w0002=(w0002+#60)%#1024

;⑥の処理

;バッファ一巡したかの確認

IF w0000>w0002 THEN

EXPR scflag=#1

ENDIF

;一巡した以降であれば BUF0_STPOS もインクリメント

IF scflag=#1 THEN

EXPR BUF0_STPOS=(BUF0_STPOS+#60) % #1024

ENDIF

;⑦のトレンド再描画

TREND_REDRAW

ENDIF

4.2-3 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ようになります。

動作としては 50ms 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 BUF0_CUPOS,BUF0_STPOS,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          //BUF0_CUPOS 値管理用
            ,stpos=0          //BUF0_STPOS 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(2000);
    //①の処理
    GopWrite_w(0xf400,stpos);
    GopWrite_w(0xf402,cupos);
    //②の処理
    GopWrite_w(0x0000,startpos);
    GopWrite_w(0x0002,endpos);
    while(1){
        if(TimeCount()-oldSendTime+50){ //50ms 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き込み
            GopWrite_w((cupos*2+0xa000),GetSlider()*100/255);
            //④の処理
            cupos=(cupos+1)%1024;
            GopWrite_w(0xf402,cupos);
            //書き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+60)%1024;
                GopWrite_w(0x0000,startpos);
                endpos=(endpos+60)%1024;
                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+60)%1024;
                    GopWrite_w(0xf400,stpos);
                }
            }
            oldSendTime=TimeCount();
        }
    }
}
```