



モータ制御開発及び

組込み支援システム

*S i m t r o l - m*

ユーザーズ・マニュアル

# 関 数 編

— 関数リファレンス —

2007-2

株式会社 昭和電業社

---

---

# 目次

はじめに	1
1、関数とブロック・ダイアグラムの関係	2
1, 1、ブロックと関数の対応	2
1, 2、ブロックの入出力端子と変数との対応	3
1, 3、ブロック・データと関数データの対応	4
1, 4、ブロック・ダイアグラムの作成法の概要	6
1, 4, 1、ブロックの接続操作の概要	6
1, 4, 2、ブロック（関数）の設定操作の概要	8
2、関数群と各関数	9
2, 1、Source 群ブロック／関数	9
10001: STEP (ステップ信号発生器)	10
10002: CONST (定数信号器)	10
10003: SIN (正弦波発生器)	11
10004: SQUARE (方形波発生器)	12
10005: TRIANG (三角波発生器)	13
10006: NOISE (ノイズ信号発生器)	14
10007: RAMP (ランプ信号発生器)	15
10008: 3PHASE (平衡3相交流信号発生器)	16
10009: OPWAVE (任意波形発生器)	17
10010: SINGEN	18
2, 2、Display 群ブロック／関数	19
20001: SCOPE1 (1入力型オシロスコープ)	20
20002: BAR1 (1入力型バーグラフ)	22
20003: NUM1 (1入力型数値表示)	24
20004: SCOPE2 (2入力型オシロスコープ)	25
20005: SCOPE4 (4入力型オシロスコープ)	27

---

---

## 2, 3、連続群ブロック／関数・・・・・・・・・・・・・・ 29

30001 : SUB (減算器)・・・・・・・・・・・・・・	30
30002 : GAIN (増幅器／定数乗算器)・・・・・・・・・・・・・・	31
30003 : RK4 (一次遅れ要素：ルンゲ・クッタ4次法)・・・・・・・・・・・・・・	32
30004 : SUM4 (4入力加算減算器)・・・・・・・・・・・・・・	33
30005 : ADD (加算器)・・・・・・・・・・・・・・	34
30006 : EULER (一次遅れ要素：オイラー法)・・・・・・・・・・・・・・	35
30007 : RK2 (一次遅れ要素：ルンゲ・クッタ2次法)・・・・・・・・・・・・・・	36
30008 : RK42 (二次遅れ要素：ルンゲ・クッタ4次法)・・・・・・・・・・・・・・	37
30009 : LEADLAG (進み遅れ関数)・・・・・・・・・・・・・・	38
30010 : PID (PID制御器)・・・・・・・・・・・・・・	39
30011 : LIMIT (リミット)・・・・・・・・・・・・・・	40
30012 : ADC (A-Dコンバータ)・・・・・・・・・・・・・・	41
30013 : DAC (D-Aコンバータ)・・・・・・・・・・・・・・	42
30014 : MAT22 (マトリクス演算)・・・・・・・・・・・・・・	43
30015 : RMAT22 (逆マトリクス演算)・・・・・・・・・・・・・・	44
30016 : PID	
(PID型制御器：測定値微分方式PID制御)・・・・・・・・・・・・・・	45
30017 : IPD	
(IPD型制御器：測定値比例微分方式PID制御)・・・・・・・・・・・・・・	46
30018 : PI (PI制御器)・・・・・・・・・・・・・・	47
30019 : CTRL_P (P制御器)・・・・・・・・・・・・・・	48
30020 : CTRL_I (I制御器)・・・・・・・・・・・・・・	49
30021 : CTRL_D (D制御器)・・・・・・・・・・・・・・	50
30022 : DCM (DCモータ)・・・・・・・・・・・・・・	51
30023 : MAT23 (マトリクス演算)・・・・・・・・・・・・・・	52
30024 : MAT32 (マトリクス演算)・・・・・・・・・・・・・・	53
30025 : UVW2DQ (3相→d q変換)・・・・・・・・・・・・・・	54
30026 : UV2DQ (2相(3相)→d q変換)・・・・・・・・・・・・・・	55
30027 : DQ2UVW (d q→3相変換)・・・・・・・・・・・・・・	56
30028 : AB2DQ ( $\alpha\beta$ →d q変換)・・・・・・・・・・・・・・	57
30029 : DQ2AB (d q→ $\alpha\beta$ 変換)・・・・・・・・・・・・・・	58
30030 : AB2UVW ( $\alpha\beta$ →3相変換)・・・・・・・・・・・・・・	59
30031 : UVW2AB (3相→ $\alpha\beta$ 変換)・・・・・・・・・・・・・・	60
30032 : PIDARW	
(アンチ・リセットワインドアップPID制御器)・・・・・・・・・・・・・・	61
30033 : BLDCM	
(ブラシレスDCモータ：モータパラメータ)・・・・・・・・・・・・・・	62
30034 : BLDCM1	
(ブラシレスDCモータ：Flux値)・・・・・・・・・・・・・・	63
30035 : DCTRL	
(ブラシレスDCモータの非干渉化：モータパラメータ)・・・・・・・・・・・・・・	64
30036 : DCTRL1	
(ブラシレスDCモータの非干渉化：Flux値)・・・・・・・・・・・・・・	65
31002 : GAIN (増幅器／定数乗算器)・・・・・・・・・・・・・・	66
31004 : 2/3入力加算減算器・・・・・・・・・・・・・・	67

---

---

## 2, 4、I/O群ブロック／関数 . . . . . 6 8

4 0 0 0 1 : ADC 1 3 6 0 0 (1 3 6 0 0用A-Dコンバータ) . . . . .	70
4 0 0 0 2 : DAC 1 3 6 0 0 (1 3 6 0 0用D-Aコンバータ) . . . . .	71
4 0 0 0 3 : PWM 1 3 6 0 0 (1 3 6 0 0用PWM) . . . . .	72
4 0 0 0 4 : ENC 1 3 6 0 0 (1 3 6 0 0用エンコーダ①) . . . . .	73
4 0 0 0 5 : BLDCM 1 3 6 0 0 (1 3 6 0 0用ブラシレスDCモータ) . . . . .	74
4 0 0 0 6 : BLDC I 1 3 6 0 0 (1 3 6 0 0用整数型ブラシレスDCモータ) . . . . .	75
4 0 0 0 7 : SQW 1 3 6 0 0 (1 3 6 0 0用120度矩形波駆動出力) . . . . .	77
4 0 0 0 8 : ENC 2 1 3 6 0 0 (1 3 6 0 0用エンコーダ②) . . . . .	78
4 0 0 0 9 : SQWL 1 3 6 0 0 (1 3 6 0 0用120度矩形波駆動出力 遅れ制御付) . . . . .	79
4 0 0 1 5 : BLDC C 1 3 6 0 0 (1 3 6 0 0用ブラシレスDCモータ 電流制限付) . . . . .	80

## 2, 5、離散群ブロック／関数 (該当関数なし) . . . . . 8 2

## 2, 6、非線形群ブロック／関数 . . . . . 8 2

6 0 0 0 1 : INVPEND (倒立振子: スコープ表示) . . . . .	83
6 0 0 0 2 : SW (切替スイッチ) . . . . .	85
6 0 0 0 3 : > (比較>) . . . . .	85
6 0 0 0 4 : < (比較<) . . . . .	86
6 0 0 0 5 : ≥ (比較≥) . . . . .	86
6 0 0 0 6 : ≤ (比較≤) . . . . .	87
6 0 0 0 7 : = (比較=) . . . . .	87
6 0 0 0 8 : AND (論理積演算) . . . . .	88
6 0 0 0 9 : OR (論理和演算) . . . . .	88
6 0 0 1 0 : NOT (論理否定演算) . . . . .	89
6 0 0 1 1 : RS (RSフリップフロップ) . . . . .	89
6 0 0 1 2 : TIM (オンディレータイマー) . . . . .	90
6 0 0 1 3 : HS (ハイ・セレクト) . . . . .	91
6 0 0 1 4 : LS (ロー・セレクト) . . . . .	91

## 2, 7、整数群ブロック／関数 . . . . . 9 2

7 0 0 0 1 : INT_CONST (整数定数信号器) . . . . .	93
7 0 0 0 2 : ROUND (四捨五入演算) . . . . .	94
7 0 0 0 3 : INT_SUB (整数減算) . . . . .	94
7 0 0 0 4 : INT_ADD (整数加算) . . . . .	95
7 0 0 0 5 : INT_I (整数型積分制御器) . . . . .	95
7 0 0 0 6 : INT_STEP (整数ステップ信号器) . . . . .	96
7 0 0 0 7 : FLOAT (浮動小数点型変換) . . . . .	96
7 0 0 0 8 : INT_P (整数型比例動作) . . . . .	97

---

---

7 0 0 0 9 : U V 2 D Q I ( 整数型 3 相 → d q 変換 ) . . . . .	97
7 0 0 1 0 : I N T _ P I ( 整数型比例積分動作 ) . . . . .	98

## 2, 8、M a t h 群ブロック／関数 . . . . . 9 9

8 0 0 0 1 : M U L T ( 掛け算器 ) . . . . .	100
8 0 0 0 2 : D I V ( 割り算器 ) . . . . .	100
8 0 0 0 3 : S Q R T ( 平方根 ) . . . . .	101

## 付録

・ブロック ( 関数 ) 名索引 . . . . .	付 1
----------------------------	-----

---

## ● はじめに

S i m t r o l - m をご利用いただき誠にありがとうございます。

本マニュアルでは、S i m t r o l - m に搭載されている、各関数ブロックの使い方と対応する各関数の内容について、説明いたします。

S i m t r o l - m では対象のシステムの、データの入出力の関係に着目し、ブロック・ダイアグラムの形式でシステムの入力と出力を関連付けて、様々な処理を可能とします。ブロック・ダイアグラムを構成する各ブロックも、それぞれ入力と出力を持っており、かつ、対応する関数をもっています。ブロック・ダイアグラムでは各ブロックは線でつながれており、それぞれの入出力を関連づけられております。すなわち、ブロック・ダイアグラムのブロックを関数、ブロック間を繋ぐ線を関数の入出力変数、と考えることができます。

それはそのまま、プログラム言語における関数と変数に対応可能です。以上のことを踏まえ、S i m t r o l - m ではブロックの機能ごとに関数を定義し、そのままモジュール形式でプログラミングすることで、各ブロックに対応した関数機能をブロック単位のオブジェクトとして実現いたしました。

各ブロックには対応する関数が一つあります。ブロックに入力されたデータを対応する関数で処理して出力し、次のブロックの入力とします。その繰り返しで処理を実行します。

以下、S i m t r o l - m のライブラリに搭載されている各ブロック及び、対応する関数について、説明します。

\* S i m t r o l - m の操作及び基本事項については、別紙「S i m t r o l - m ユーザーズ・マニュアル 入門編」などをご参照下さい。

---

## 1、関数とブロック・ダイアグラムの関係

### 1, 1、ブロックと関数の対応

画面要素のブロックは、それぞれが対応するC言語の関数を持っております。

インタープリタ処理によるシミュレーション処理時には、各ブロックに対応する関数がブロック・ダイアグラムの実行順序に基づいて実行されていきます。

ブロック・ダイアグラムからコンパイラ処理を実行すれば、ブロック・ダイアグラム上に定められた実行順序に基づき、ブロックごとに対応する関数のソース・コードが出力され、ブロック・ダイアグラムの内容に準拠したC言語のソース・コードとして出力されます。

このとき、

ブロックがエディタ画面上に追加されれば、対応する関数及び、その関数に関する処理が追加されることになります。

同様にブロックがエディタ画面から削除されますと、対応する関数及び、その関数に関する処理が削除されることになります。

対応する関数とそのデータを含め、ブロック・ダイアグラム上のブロックごとに独立したオブジェクトとして扱うため、ブロックの追加／削除に関して、データの干渉による不祥事は基本的に生じません。

上記の処理は基本的に自動設定されます。

## 1, 2、ブロックの入出力端子と変数との対応

ブロックの入力端子は対応する関数の入力変数に、ブロックの出力端子は対応する関数の出力変数に、各々対応しております。

ブロックの出力端子と他のブロックの入力端子とを線で繋ぎ、ブロック・ダイアグラムを構成すれば、出力端子側のブロックに対応する関数の出力を、入力端子側のブロックの対応する関数へ入力したことになります。

ここで、ブロックの端子間を結ぶ線を変数とみなし、ブロックの出力端子から出力された値＝関数の出力値を、変数（線）に代入し、他のブロックの入力端子＝関数の入力へ直接、代入しているものとみなせば、ブロック・ダイアグラムによるブロック相互の関係を、ブロックに対応する関数への入出力の関係に変換することが可能です。

（出力端子）－（線）－（入力端子）という組み合わせで、一つの変数として動作しているとみなすこともできます。（図 1－1 参照）

上記の意味で、ブロックに対応する関数機能を指す場合など、ブロックのことを関数と呼ぶ場合がありますので、ご注意ください。

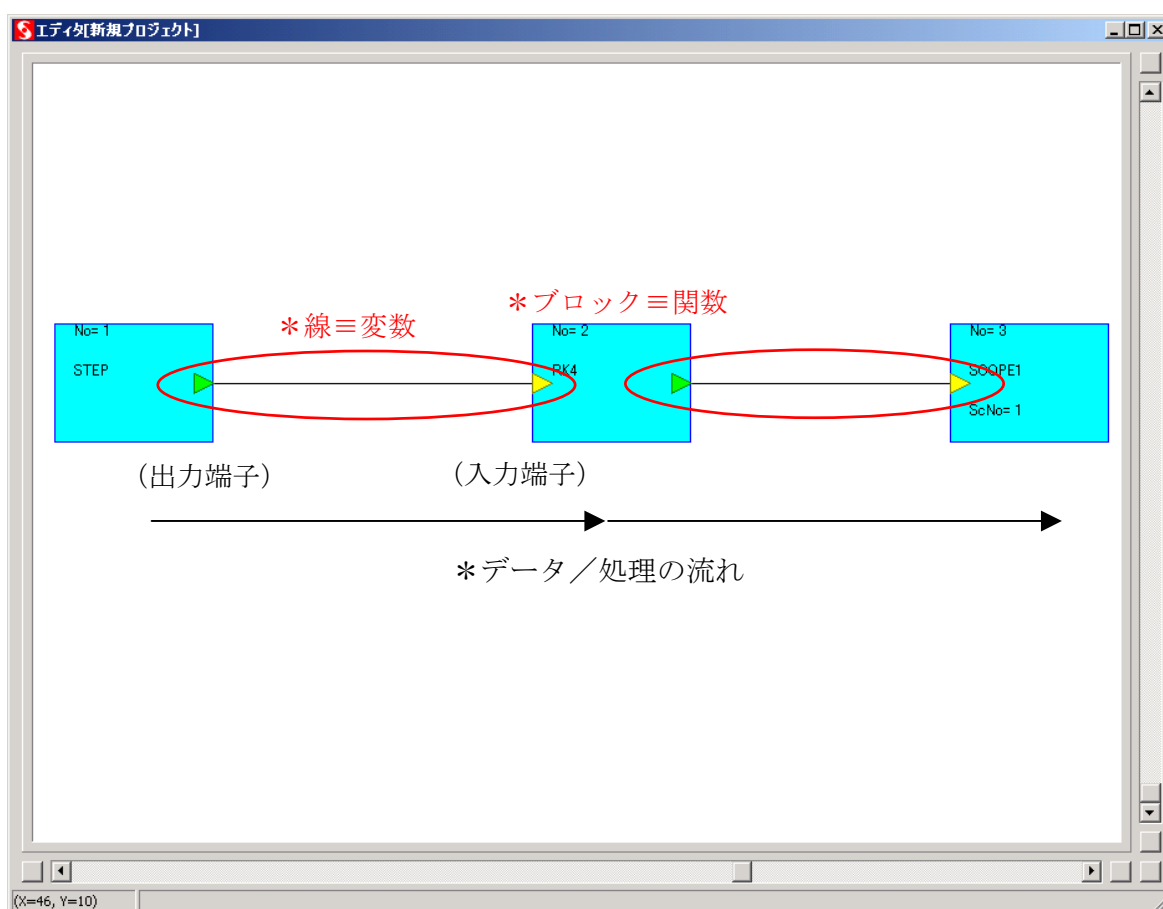


図 1－1：ブロック・ダイアグラム上の入出力端子と線の関係

（前提：線はブロックの出力端子を始点とし、他のブロックの入力端子を終点とすること。）

Simtrol-m の実際の関数ブロックでは、ブロックの出力端子ごとに別の変数を設定します。また、同一の入力端子へ複数の線を結線することを禁止して、ブロックの入力端子を関数の入力変数と 1 対 1 に明確に対応つけております。各々の入力端子に対応するデータ形式のブロックの出力端子とを線で結ぶ必要があります。



## 1, 3、ブロック・データと関数データの対応

各ブロックで設定されたプロパティ・データなどは、そのまま対応する C 言語の関数のデータとなります。実例で示します。

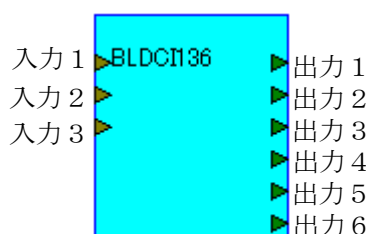
(実例)

### ・BLDCI13600ブロック

：KENTAC13600用整数型ブラシレスDCモータの関数ブロックの場合。

(ブラシレスDCモータをシミュレーションするI/O関数のブロックです。

詳細は40006：BLDCI13600の項をご参照ください。)



### \* BLDCI13600ブロックの入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	V d : d 軸電圧	[V]	int 型	
2	V q : q 軸電圧	[V]	int 型	
3	T L : 負荷トルク	[N・m]	int 型	

### \* BLDCI136000ブロックの出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	i d : d 軸電流	[A]	int 型	
2	i q : q 軸電流	[A]	int 型	
3	$\omega m$ : モータの回転角速度	[rad/sec]	int 型	
4	$\theta m$ : モータの回転角	[rad]	int 型	
5	Tm : モータの発生トルク	[N・m]	int 型	
6	$\theta s$ : モータの回転角積算値	[rad]	int 型	

### \* BLDCI136000ブロックのプロパティ・データ

No.	プロパティ・データ名	単位	データ型	備考
1	Ra : 巻線抵抗	[ $\Omega$ ]	float 型	
2	Ld : d 軸インダクタンス	[H]	float 型	
3	Lq : q 軸インダクタンス	[H]	float 型	
4	J : 慣性モーメント	[kg・m <sup>2</sup> ]	float 型	
5	D : 粘性摩擦係数	[N・m・s/rad]	float 型	
6	K : モータパラメータ	[Vs/rad]または [Nm/A]	float 型	
7	P : 極対数	—	float 型	
8	Vs : 電圧スケール	[V]	float 型	
9	Ts : トルクスケール	[N・m]	float 型	
10	Is : 電流スケール	[A]	float 型	
11	Pe : エンコーダの1週パルス数	[p/r]	float 型	1000/2000
12	Dp : エンコーダのカウント方向	—	float 型	0 : 正方向/ 1 : 逆方向

---

\* B L D C I 1 3 6 0 0 0 ブロックの対応関数 : fn\_bldci13600()

```
void fn_bldci13600(int_vd, int_vq, int_TL, float *bldcparam, float *work,
                  int *int_work, int *bldcmout0, int *bldcmout1, int *bldcmout2,
                  int *bldcmout3, int *bldcmout4, int *bldcmout5, int *err);
```

以下に fn\_bldci13600() 関数の入出力変数と B L D C I 1 3 6 0 0 ブロックの各データとの対応を示します。

①関数の引数 (ブロックの入力変数)

```
int int_vd // d 軸電圧 // (-10000~10000 が-Vs~ Vs[V]に相当)
int int_vq // q 軸電圧 // (-10000~10000 が-Vs~ Vs[V]に相当)
int int_TL // 負荷トルク // (-10000~10000 が-Ts~ Ts[V]に相当)
```

②プロパティ・データ領域各値

```
float bldcparam[0] // Ra : 巻線抵抗 [Ω]
float bldcparam[1] // Ld : d 軸インダクタンス [H]
float bldcparam[2] // Lq : q 軸インダクタンス [H]
float bldcparam[3] // J : 慣性モーメント [Kgm2]
float bldcparam[4] // D : 粘性摩擦係数 [Nm・s/rad]
float bldcparam[5] // K : モータパラメータ [Vs/rad]または[Nm/A]
float bldcparam[6] // P : 極対数
float bldcparam[7] // Vs : 電圧スケール [V]
float bldcparam[8] // Ts : トルクスケール[Nm]
float bldcparam[9] // Is : 電流スケール[A]
float bldcparam[10] // Pe : エンコーダの一周パルス数(1000 または 2000) [p/r]
float bldcparam[11] // Dp : エンコーダのカウント方向 (0:正方向 1:逆方向)
```

③作業領域各値\*

```
float work[0] // 途中結果 (id)
float work[1] // 途中結果 (iq)
float work[2] // 途中結果
float work[3] // 途中結果 ( $\omega_m$ )
float work[4] // 途中結果 ( $\omega_e$ )
float work[5] // 途中結果 ( $\theta_m$ )
float work[6] // 途中結果 ( $\theta_s$ )
```

④関数の戻り値 (ブロックの出力変数)

```
int bldcmout[0] // id : d 軸電流 (-10000~10000 が-Is~ Is[A]に相当。)
int bldcmout[1] // iq : q 軸電流 (-10000~10000 : -Is~ Is[A])
int bldcmout[2] //  $\omega_m$  : モータの回転角速度 (-4000~4000 :  $-2\pi/dt \sim 2\pi/dt$  [rad/sec])
int bldcmout[3] //  $\theta_m$  : モータの回転角 [rad] (-4000~4000 :  $-2\pi \sim 2\pi$  [rad])
int bldcmout[4] // Tm : モータ発生トルク [Nm] (-10000~10000 : -Ts~ Ts[V])
int bldcmout[5] //  $\theta_s$  : モータの回転角積算値 [rad] (-4000~4000 :  $-2\pi \sim 2\pi$  [rad])
```

前頁のブロック各値と上記の C 言語関数の入出力変数とを比較してみると、ブロックの入力端子が関数の引数 (入力変数) に、ブロックの出力端子が関数の戻り値 (出力変数) に、それぞれ対応しているのが分かります。またブロックのプロパティ・データが、関数のプロパティ・データ領域として確保されているのが分かります。つまり、各ブロックに与えられたデータ値が対応する関数の引数にもなっている訳です。

\* 作業領域 : w o r k [] とは、必要な場合に特定の C 言語関数上に定義される演算用の領域です。ユーザ側では設定等を行なう必要はありません。


## 1, 4、ブロック・ダイアグラムの作成法の概要




### 1, 4, 1 ブロックの接続操作の概要




ブロックを関連つけて、ブロック・ダイアグラムを作成するために必要な、エディタ画面でのブロック相互間の結線操作について、概略を説明します。

操作の詳細については、別紙「Simtrol-m ユーザーズ・マニュアル 入門編」などをご参照ください。

ブロック間の結線操作)

まず、エディタ画面の動作モードを【接続線】モードに切り替えます。

ブロックの出力端子：／をリックしてから、他のブロックの入力端子：をクリックしますと、下図のようにブロックの端子の間が結線されます。

なお、入力端子：をクリックしてから、出力端子：／をクリックしても同じです。  
また出力端子・入力端子ともに、結線されていない端子は濃い色で表示されておりますので、結線済みの端子と見分けることができます。


- ・出力端子の表示：結線前  → 結線後 .
- ・入力端子の表示：結線前  → 結線後 .

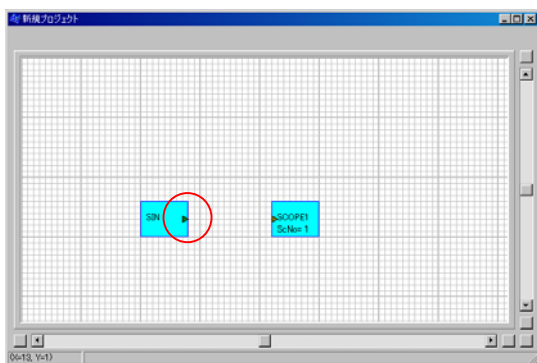
注意) 結線済みの入力端子：に重複して他の線を結線することはできません。

また、ウィンドウが異なるエディタ画面上のブロック同士を結線することはできません。


次にブロックとブロックとの間の結線の手順（一例）を示します。

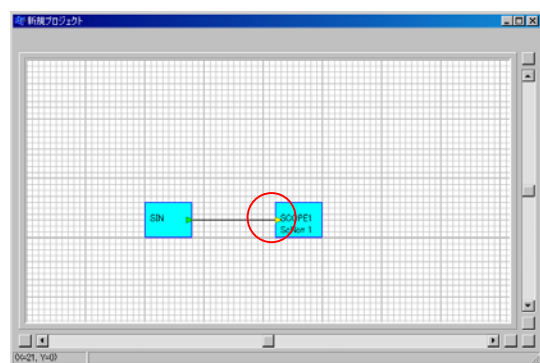


①動作モード：【接続線】モードを選択。



(図：②)

②ブロックの出力端子：をクリック。

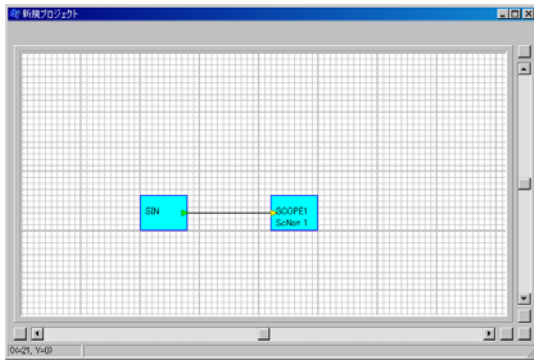


(図：③)



③結線先のブロックの入力端子：をクリック。

(次頁へ続く)



(前頁より続く)





(図 : ④)

④出力端子・入力端子の色が変化します。(  → 、 →  )

同時にブロック間が線で結ばれます。＝ブロック同士が接続されます。

なお、ブロックの入力端子 :  /  には、複数の線を接続することはできません。

従って、既に接続されている状態の入力端子 :  には、線を接続されることはできません。

接続済みの入力端子 :  への接続を変えたい場合は、そこに接続されている線を一度

削除し、接続を解除された上で (端子が  →  へ変化します。)、改めて接続して下さい。

この処理は自動化されています。(確認の警告メッセージが表示されます。)

## 1, 4, 2、ブロック（関数）の設定操作の概要

ブロック・ダイアグラムを作成する場合、関数（ブロック）を線で関連つけるだけでなく、関数自体の設定も必要になります。

ブロック・ダイアグラム上でブロックを選択（ダブル・クリック）し、「ブロック内容」なる画面を呼び出します。

ブロック内容

ブロックNo.=3 GAIN  
ブロック図での表示  
GAIN 初期化

名称  
増幅器(定数乗算器)

説明  
増幅器(定数乗算器)  
出力値 = 入力値・gain、  
gain: 増幅率(利得)

プロパティ  
gain: 増幅率(利得): 実数値 = 1 初期化

入力端子名  
input: 入力 接続先から取得 初期化

出力端子名  
output: 出力 初期化

実行ディレイサイクル  
1

OK キャンセル

図1－2：[ブロック内容] 画面

上記図1－2の表示内容中、**プロパティ**の各項は、ブロックごとに固有のパラメータ項目です。基本的にユーザにより設定される項目です。

その他、ブロック内容画面でのユーザによる可変・設定項目として、**ブロック図での表示**（エディタ画面上に表示される個々のブロックの名称）、**入力端子名**（各端子の名称を変更可能）、**出力端子名**（各端子度の名称を変更可能）、**実行ディレイサイクル**の各項があります。

パラメータ・データの具体的な設定方法は、別紙「Simtrol-m ユーザーズ・マニュアル 入門編」などをご参照ください。

## 2、関数群と各関数

### 2、1、Source 群ブロック／関数

各種の信号を発生させるためのブロック群です。

各ブロックのプロパティ設定により、関数にパラメータ・データを与えて動作させます。外部から与えられる入力変数は必要なく、発生した信号を出力するための出力変数のみとなります。

したがって、ブロックにも入力端子はなく、出力端子のみとなります。(一部例外を除く。)

- ・ ライブラリ画面上では、「1:Source」のタブをクリックして呼び出します。
- ・ メニュー画面からは、「ライブラリ」－「Source」より、関数を選択します。
- ・ 基本的にブロックのプロパティ・データの設定により関数が動作します。  
プロパティ・データの設定には、ご注意ください。



Source 群(ライブラリ表示)

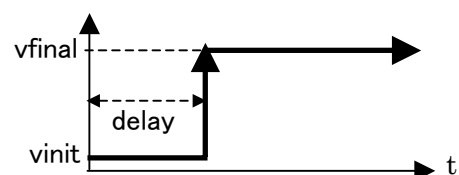
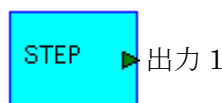
Source 群の関数ブロックに対応するC言語関数は、以下の形式になります。  
(一部例外あり)

```
void fn_***(float *propaty, float *output, int *err);
```

- ・ 関数名の接頭語「fn\_」はSimtrol\_mの関数共通の接頭語です。
- ・ \*propaty は、ブロックのプロパティ設定データの配列を示します。
- ・ 通常は、変数のポインタ渡し形式などを利用して、引数 float \*output を出力用変数として用います。関数はvoid型で定義され、値を持ちません。
- ・ int \*err は各関数共通のエラー処理用変数として定義されており、初期値=0とします。

以下、Source群の各ブロック／関数を解説いたします。

10001: STEP (ステップ信号発生器)



動作：出力の初期値を  $v_{init}$  とし、 $delay$  [sec] 経過後、  
値  $v_{final}$  に出力を切り替えます。  
切替後は  $v_{final}$  を常時出力します。

入力端子／データ：なし

出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	output : 出力	—	float 型	

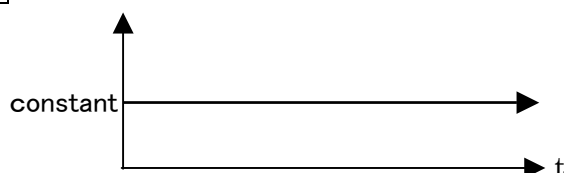
プロパティ・データ

No.	プロパティ・データ名	単位	データ型	備考
1	delay : 切り替え時間	[sec]	float 型	
2	vinit : 出力初期値	—	float 型	
3	vfinal: 切り替え後の出力値	—	float 型	

対応関数

```
void fn_step (float *propaty, float *output, int *err);
```

10002: CONST (定数信号器)



動作：定数値  $constant$  を常時出力する。

入力端子／データ：なし。

出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	output : 出力	—	float 型	

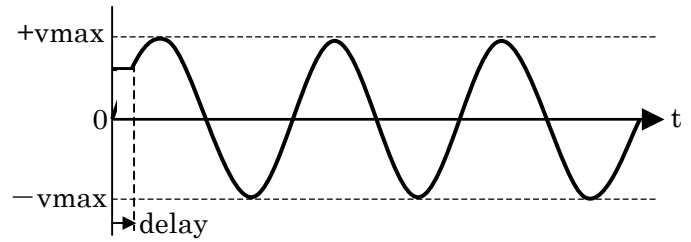
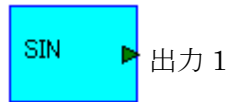
プロパティ・データ

No.	プロパティ・データ名	単位	データ型	備考
1	constant : 定数	—	float 型	

対応関数

```
void fn_const (float *propaty, float *output, int *err);
```

10003 : SIN (正弦波発生器)



動作

正弦波信号を出力する。

出力 $= (vmax) \cdot \sin (2 \pi \cdot (frequency) \cdot (t - delay) + phase)$ 。

vmax : 最大値、frequency : 周波数、delay : 遅延時間、phase : 初期位相。

但し、遅延時間中( $t \leq delay$ )は、

初期位相 phase による出力値 $= (vmax) \cdot \sin (phase)$ を保持する。

入力端子／データ : なし。

出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	output : 出力	—	float 型	

プロパティ・データ

No.	プロパティ・データ名	単位	データ型	備考
1	frequency : 周波数	[H z]	float 型	
2	phase : 初期位相	[rad]	float 型	
3	vmax : 出力最大値	[V]	float 型	0 以上の値
4	delay : 遅延時間	[sec]	float 型	

対応関数

```
void fn_sin (float *propaty, float *output, int *err);
```

- $\pi$  : 円周率値=3.14159 を使用して処理を実行します。
- 位相／角度の単位はラジアン[rad]です。



10004 : SQUARE (方形波発生器)



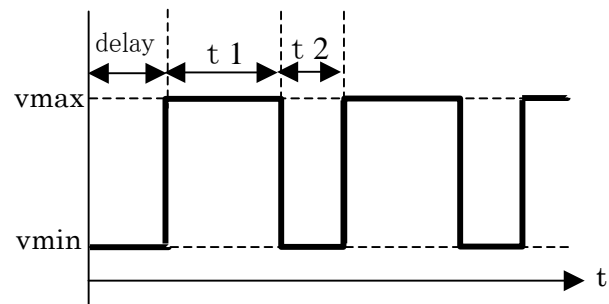
動作：方形波を出力する。

出力最大値：vmax、最大値保持時間：t1

出力最小値：vmin、最小値保持時間：t2

波形周期：t1 + t2、デューティ比：t1 / (t1 + t2)

遅延時間：delay、t ≤ delay の間は、出力=vmin（最小値）を保持する。



入力端子／データ：なし。

出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	output : 出力	—	float 型	

プロパティ・データ

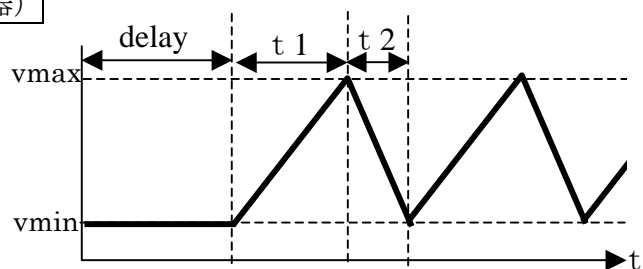
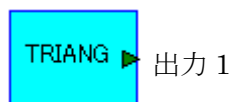
No.	プロパティ・データ名	単位	データ型	備考
1	frequency : 周波数	[Hz]	float 型	1 / (t1 + t2)
2	vmin : 出力最小値(最小電圧)	[V]	float 型	
3	vmax : 出力最大値(最大電圧)	[V]	float 型	
4	duty : デューティ比	[%]	float 型	t1 / (t1 + t2)
5	delay : 遅延時間	[Sec]	float 型	

対応関数

void fn\_square(float \*propaty, float \*output, int \*err) ;

- ・周波数値 frequency = 1 / (t1 + t2)。
- ・周波数値 frequency = 0 は入力禁止。(検出した場合、エラー扱いとなります。)

1 0 0 0 5 : TRIANG (三角波発生器)



動作：三角波を出力する。

波形最小値：vmin、波形上昇時間：t1。

波形最大値：vmax、波形下降時間：t2。

波形周期：t1+t2、デューティ比：t1 / (t1+t2)、

遅延時間：delay、 $t \leq \text{delay}$  の間は出力 = vmin(最小値)を保持。

入力端子／データ：なし。

出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	output：出力	—	float 型	

プロパティ・データ

No.	プロパティ・データ名	単位	データ型	備考
1	frequency：周波数	[Hz]	float 型	$1 / (t_1 + t_2)$
2	vmin：出力最小値(最小電圧)	[V]	float 型	
3	vmax：出力最大値(最大電圧)	[V]	float 型	
4	duty：デューティ比	[%]	float 型	$t_1 / (t_1 + t_2)$
5	delay：遅延時間	[Sec]	float 型	

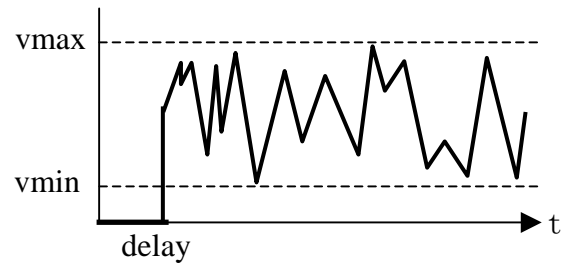
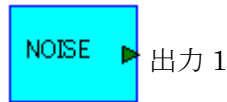
対応関数

void fn\_triangular(float \*propaty, float \*output, int \*err);

- 周波数値 frequency =  $1 / (t_1 + t_2)$ 。
- 周波数値 frequency = 0 は入力禁止。(検出した場合、エラー扱いとなります。)

---

1 0 0 0 6 : NOISE (ノイズ信号発生器)



動作：ランダムなノイズ信号を出力する。  
但し、最小値：vmin、最大値：vmax とする。  
遅延時間：delay、 $t \leq \text{delay}$  の間は、出力 = 0 とする。

入力端子／データ：なし。

出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	output：出力	—	float 型	

プロパティ・データ

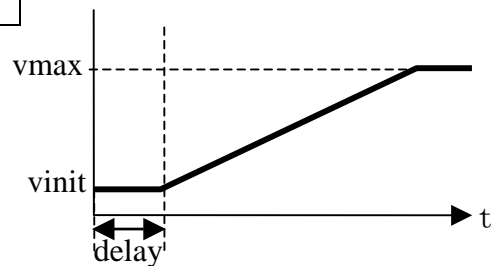
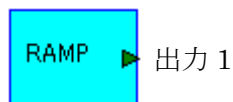
No.	プロパティ・データ名	単位	データ型	備考
1	(未使用)	—	—	予約領域
2	vmin：出力最小値(最小電圧)	[V]	float 型	
3	vmax：出力最大値(最大電圧)	[V]	float 型	
4	delay：遅延時間	[sec]	float 型	

対応関数

void fn\_noise(float \*propaty, float \*output, int \*err) ;

- ・ノイズ信号の発生周期はインタープリタの実行刻み時間値 d t に等しくなります。

1 0 0 0 7 : RAMP (ランプ信号発生器)



動作：ランプ信号を出力します。

出力 =  $v_{\text{initial}} + \text{gradient} \cdot (t - \text{delay})$ 。

$v_{\text{initial}}$ :初期値、gradient:ランプの勾配値。

但し、 $v_{\text{max}}$ : 最大値とし、 $\pm v_{\text{max}}$  で出力リミットとします。

リミットに到達後はその値を出力維持します。

遅延時間:delay、 $t \leq \text{delay}$ の間は、出力= $v_{\text{initial}}$ (一定)とします。

\* ランプ信号：時間に比例した一定勾配で上昇／下降を続ける信号のこと。

入力端子／データ：なし。

出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	output : 出力	—	float 型	

プロパティ・データ

No.	プロパティ・データ名	単位	データ型	備考
1	Gradient : 勾配	[V/sec]	float 型	初期値 : 1
2	vinitial : 初期値	[V]	float 型	
3	delay : 遅延時間	[sec]	float 型	
4	vmax : 最大値(リミット値)	[V]	float 型	初期値 : 1E20

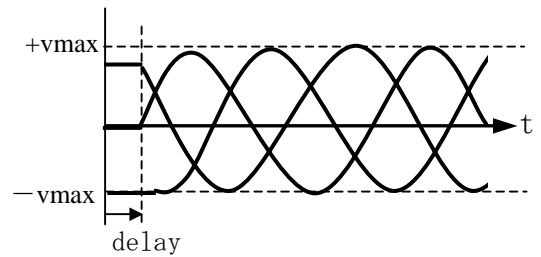
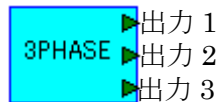
対応関数

```
void fn_ramp(float *propaty, float *output, int *err);
```

- マイコン・ボードへのプログラムの移植時などは、 $v_{\text{max}}$ : 最大値 (リミット値) にご注意ください。

(対象とするシステムが取り扱い可能な範囲の数値を設定して下さい。)

# 1 0 0 0 8 : 3 PHASE (平衡 3 相交流信号発生器)



動作：(正弦波) 平衡 3 相交流信号を出力する。

U 相出力 =  $v_{\max} \cdot \sin(2\pi \cdot \text{frequency} \cdot (t - \text{delay}) + \text{phase})$ 。

V 相出力 =  $v_{\max} \cdot \sin(2\pi \cdot \text{frequency} \cdot (t - \text{delay}) + \text{phase} - (2/3)\pi)$ 。

W 相出力 =  $v_{\max} \cdot \sin(2\pi \cdot \text{frequency} \cdot (t - \text{delay}) + \text{phase} - (4/3)\pi)$ 。

但し、遅延時間：delay、 $t \leq \text{delay}$  の間は、各相とも初期位相値を定数出力する。

入力端子／データ：なし。

出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	output1：出力U相	—	float 型	U相
2	output2：出力V相	—	float 型	V相
3	output3：出力W相	—	float 型	W相

プロパティ・データ

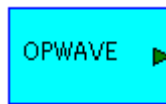
No.	プロパティ・データ名	単位	データ型	備考
1	frequency：周波数	[H z]	float 型	
2	phase：初期位相	[rad]	float 型	
3	vmax：出力最大値	[V]	float 型	0 以上の値
4	delay：遅延時間	[Sec]	float 型	

対応関数

void fn\_threephase (float \*propaty, float \*u, float \*v, float \*w, int \*err) ;

- ・ 平衡 3 相交流につき、周波数・位相・最大値は各相共通です。
- ・ 出力端子が 3 本あります。上から U 相、V 相、W 相の端子です。

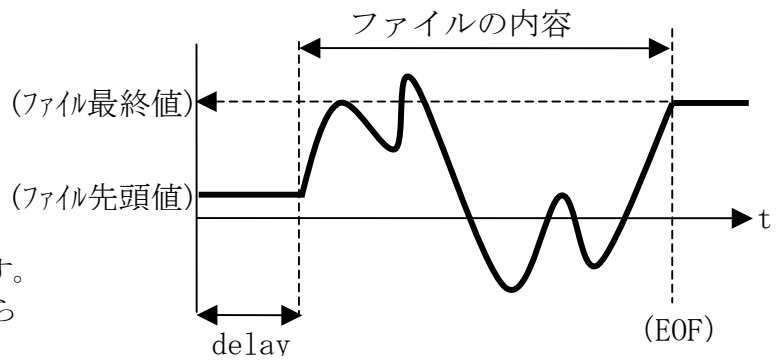
# 10009 : OPWAVE (任意波形発生器)



出力 1

## 動作

ファイルから数値データを順次読み出し、その値を出力／表示します。ファイルの最後 (EOF) に到達したら最後の値を保持します。遅延時間 : delay、 $t \leq \text{delay}$  の間はファイルの先頭値を出力します。



- ・ 波形データとして読み込むファイルは[.csv]形式のみ可。  
エディタ画面の「ブロック内容」設定画面上の[任意波形設定]ボタンより、ファイルを選択／設定します。(OPWAVEブロック独自の機能です。)

入力端子／データ : なし。

## 出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	output : 出力	—	float 型	

## プロパティ・データ

No.	プロパティ・データ名	単位	データ型	備考
1	delay : 遅延時間	[sec]	float 型	
2	OpNo : 波形 No (任意波形 No)	—	float 型	ユーザ変更不可

- ・ OPNo : 波形 No 値はユーザによる設定はできません。  
(OPWAVE ブロック生成の度に自動割付。)

## 対応関数

void fn\_optionalwave (float \*propaty, float \*output, int \*err) ;

- ・ 基本的にインタープリタ処理のみに適用可能な関数です。
- ・ 波形データ・ファイルはブロックごとに一つのみ、指定できます。  
複数個のデータ・ファイルを同じブロックで同時に使用することはできません。

---

1 0 0 1 0 : S I N G E N



#### 動作

入力値の  $\sin$  /  $\cos$  の値をそれぞれ求めます。  
(正弦波への変換／ベクトル分解などに使用可能です。)

#### 入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	入力値 $\theta$	([rad])	float 型	被変換値

#### 出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	出力 $\cos$ : $\cos(\theta)$	—	float 型	
2	出力 $\sin$ : $\sin(\theta)$	—	float 型	

プロパティ・データ：なし。

#### 対応関数

```
void fn_singen(float th, float *outcos, float *outsin, int *err);
```

- ・単独では動作しません。入力端子へのデータ（被変換値）入力が必要です。
- ・入力値をラジアン[r a d]とみなして変換します。
- ・出力端子 1（上）が  $\cos$ 、出力端子 2（下）が  $\sin$  です。

---

## 2, 2、D i s p l a y 群ブロック／関数

入力信号を画面上に出力するブロック群です。インタープリタ処理時にブロック・ダイアグラム上で信号を観測するために用います。

入力された値に対する処理であるため、入力変数及び入力変数への値入力が必要となります。一方、出力値は画面に表示されるだけで、他のブロック（関数）への入力としては利用されないため、出力端子はありません。

- ・ ライブラリ画面上では、「2:Display」のタブをクリックして呼び出します。
- ・ メニュー画面からは、「ライブラリ」－「D i s p l a y」のサブメニューを選択します。

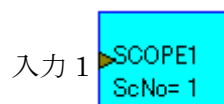


D i s p l a y 群(ライブラリ表示)

以下、D i s p l a y 群の各ブロック／関数を解説いたします。



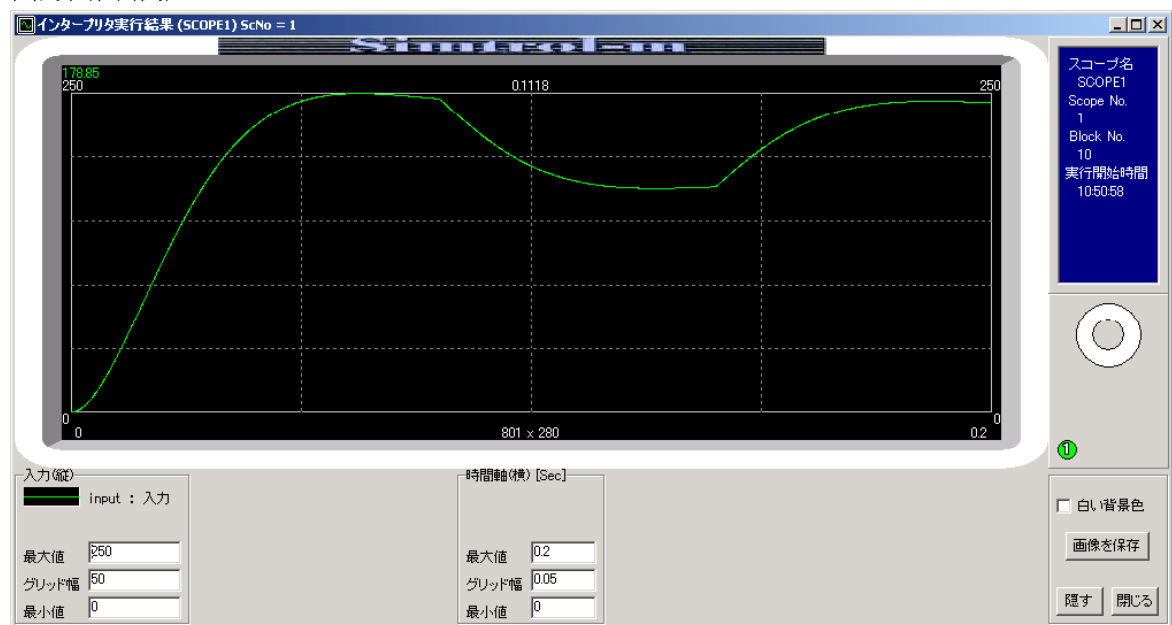
## 20001 : SCOPE1 (1入力型オシロスコープ)



### 動作

float 型 1 入力型のオシロスコープです。入力されたデータを専用の画面にプロットして、連続的に表示出力します。時間変化に対するデータの変化が観測可能です。

### 出力画面(例)



(Y 軸 (縦軸) : 入力データ / 横軸 : 時間軸)

### 入力端子 / データ

端子番号	端子名 (default)	単位	データ型	備考
1	input : 入力	—	float 型	

出力端子 / データ : なし。

プロパティ・データ

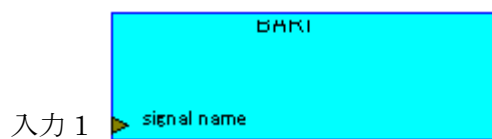
N o.	プロパティ・データ名	単位	データ型	備考
1	Tmin：画面時間軸最小値	[sec]	float 型	0 のとき自動設定
2	Tmax：画面時間軸最大値	[sec]	float 型	0 のとき自動設定
3	Tgrid：画面時間軸目盛り刻み	[sec]	float 型	0 のとき自動設定
4	ymin：画面Y軸最小値	—	float 型	0 のとき自動設定
5	ymax：画面Y軸最大値	—	float 型	0 のとき自動設定
6	ygrid：画面Y軸目盛り刻み	—	float 型	0 のとき自動設定
7	ScopeNo：スコープ番号	—	float 型	変更不可
8	buffsize:処理用バッファサイズ	—	float 型	変更不可

対応関数

```
void fn_scope1 (float input, float *propaty, int *counter, int dataSize, float *data,
               int *err );
```

- ScopeNo:スコープ番号値及び buffsize:処理用バッファサイズは自動で設定されます。  
ユーザによる設定は不用です。(ユーザによる値の変更もできませんのでご注意下さい。)
- 画面表示に必要な、時間軸／Y軸の最大値／最小値及び、各目盛り刻み（グリッド）の値はそれぞれに 0（ゼロ）を設定することで、自動設定（既定）になります。
- プロパティの画面時間軸最小値に負の値（-1 [sec]など）を設定された場合の動作は保証しません。
- 実数値であれば基本的にどのような数値でも入力可能ですが、値が設定の範囲を超えた場合は、画面表示がされません。その他設定によって、画面に測定結果が表示されない場合があります。

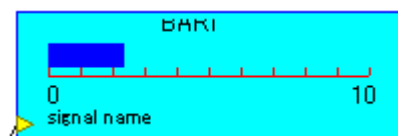
## 20002 : BAR1 (1入力型バーグラフ)



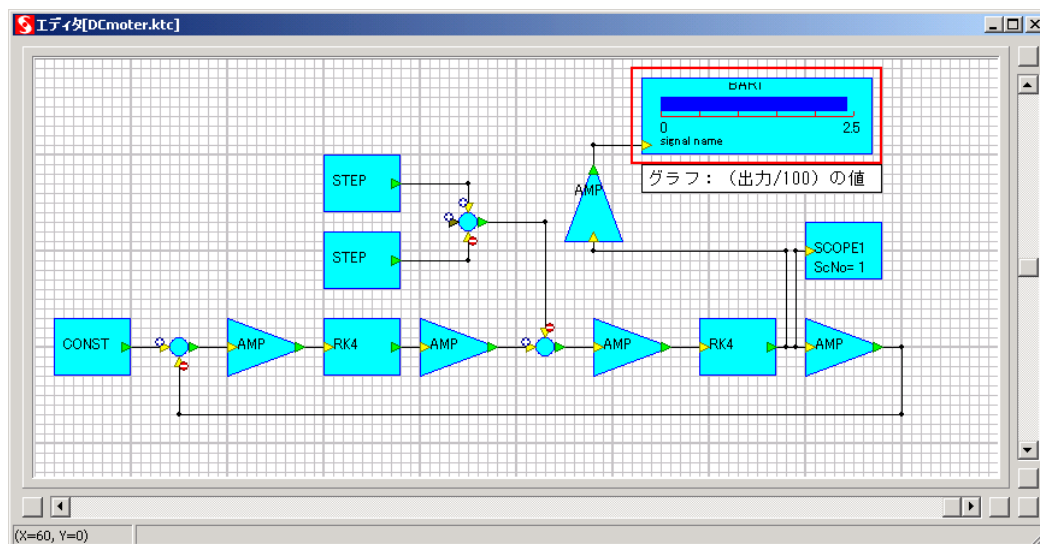
### 動作

ブロック・ダイアグラム（エディタ）上の任意の箇所を指定し、入力値をバーグラフに変換して表示します。表示はエディタ画面上で実施します。

インタープリタの実行に同期して、リアルタイムにバーグラフを表示します。



（動作時：ブロック単体）



（BAR1ブロックの使用例）

### 入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	Signal name	—	float 型	

出力端子／データ：なし。

### プロパティ・データ

No.	プロパティ・データ名	単位	データ型	備考
1	ymin : y 軸最小値	—	float 型	
2	ymax : y 軸最大値	—	float 型	
3	ygrid : y 軸目盛り刻み	—	float 型	0 値禁止

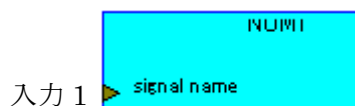
---

対応関数

```
void fn_bar1 (float input1, float *propaty, int *err );
```

- ブロックに表示される入力端子名は可変です。(既定値は「Signal name」)  
[ブロック内容]画面から設定して下さい。

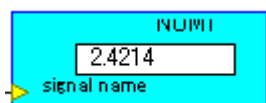
20003 : NUM1 (1入力型数値表示)



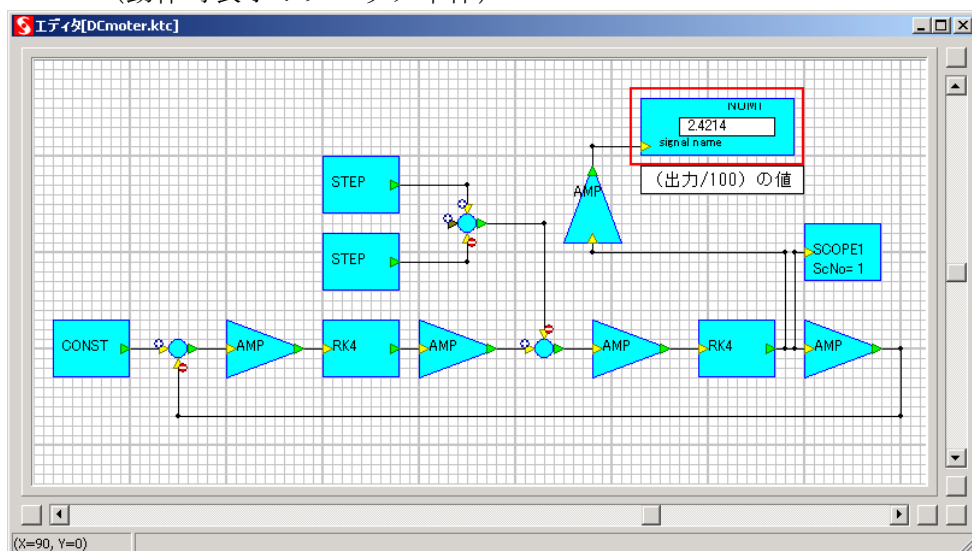
#### 動作

ブロック・ダイアグラム (エディタ) 上の任意の箇所を指定し、入力値を直接、数値にて表示します。表示はエディタ画面上で実施します。

インタープリタの実行に同期して、接続された箇所の値をリアルタイムに数値表示します。表示される数値は有効数字5桁の実数値となります。



(動作時表示 : ブロック単体)



(NUM1ブロックの使用例)

#### 入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	Signal name	—	float 型	

出力端子／データ : なし

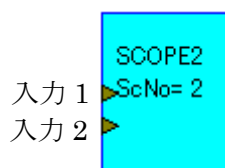
プロパティ・データ : なし

#### 対応関数

```
void fn_num1 (float input1, int *err ) ;
```

- ・ブロックに表示される入力端子名は可変です。(既定値は「Signal name」)  
[ブロック内容]画面から設定して下さい。

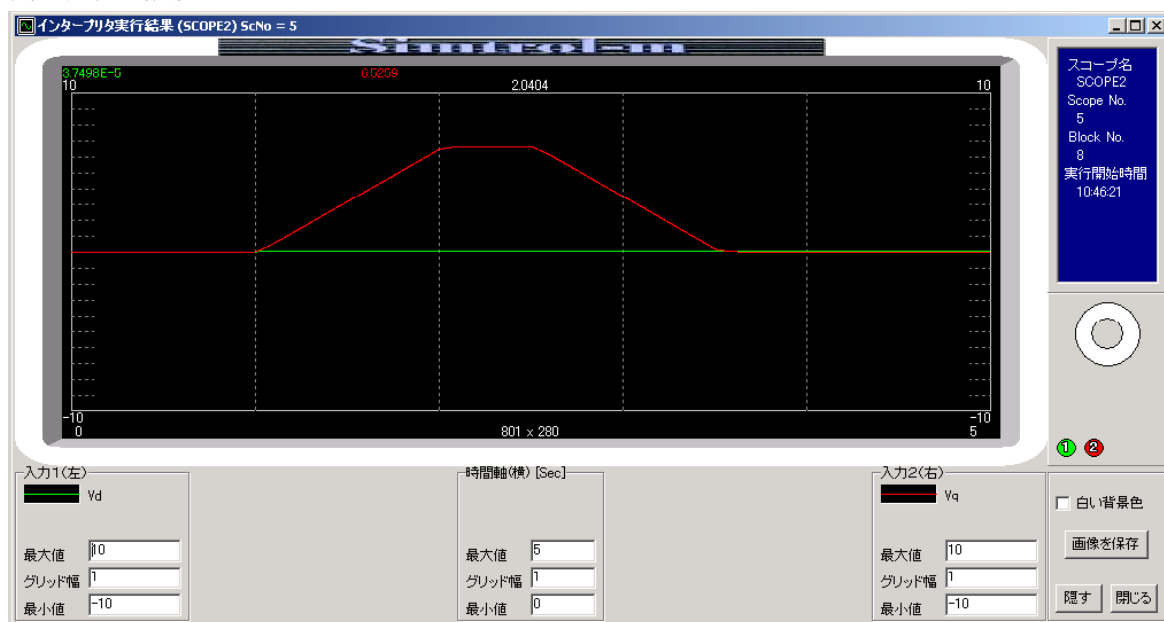
## 20004 : SCOPE 2 (2入力型オシロスコープ)



### 動作

2入力型のオシロスコープです。入力データを専用画面に表示出力します。  
 時間変化に対するデータの変化を観測します。1入力型のスコープとしても使用できます。  
 2入力ともに時間軸は共通です。  
 (入力データ間で異なる時間軸または時間刻みを使用する観測はできません。)

### 出力画面 (例)



(横軸：時間軸 (共通) / Y 軸 (縦軸)：入力データ値)  
 (左 Y 軸グリッド：入力端子 1 のデータ用 / 右 Y 軸グリッド：入力端子 2 のデータ用)

### 入力端子 / データ

端子番号	端子名 (default)	単位	データ型	備考
1	Input : 入力 1	—	float 型	画面左 Y 軸
2	Input : 入力 2	—	float 型	画面右 Y 軸

出力端子 / データ：なし。

プロパティ・データ

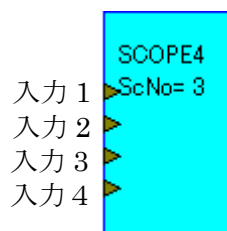
No.	プロパティ・データ名	単位	データ型	備考
1	Tmin: 画面時間軸最小値	[sec]	float 型	0 のとき自動設定
2	Tmax: 画面時間軸最大値	[sec]	float 型	0 のとき自動設定
3	Tgrid: 画面時間軸目盛り刻み	[sec]	float 型	0 のとき自動設定
4	ymin1: 左 Y 軸最小値	—	float 型	0 のとき自動設定
5	ymax1: 左 Y 軸最大値	—	float 型	0 のとき自動設定
6	ygrid1: 左 Y 軸目盛り刻み	—	float 型	0 のとき自動設定
7	ymin2: 右 Y 軸最小値	—	float 型	0 のとき自動設定
8	ymax2: 右 Y 軸最大値	—	float 型	0 のとき自動設定
9	ygrid2: 右 Y 軸目盛り刻み	—	float 型	0 のとき自動設定
10	ScopeNo: スコープ番号	—	float 型	変更不可
11	buffsize: データ数 (1 入力あたり)	—	float 型	変更不可

対応関数

```
void fn_scope2 (float input1, float input2, float *propaty, int *counter, int dataSize,
               float *data, int *err);
```

- ScopeNo: スコープ番号値及び buffsize: 処理用バッファサイズは自動で設定されます。ユーザによる設定は不用です。(ユーザによる値の変更もできません。)
- 画面表示に必要な、時間軸/Y 軸の最大値/最小値及び、各目盛り刻み (グリッド) の値はそれぞれに 0 (ゼロ) を設定することで、自動設定 (既定) になります。
- プロパティの画面時間軸最小値に負の値 (-1 [sec] など) を設定された場合の動作は保証しません。
- 実数値であれば基本的にどのような数値でも入力可能ですが、値が設定の範囲を超えた場合は、画面表示がされません。その他設定によって、画面に測定結果が表示されない場合があります。

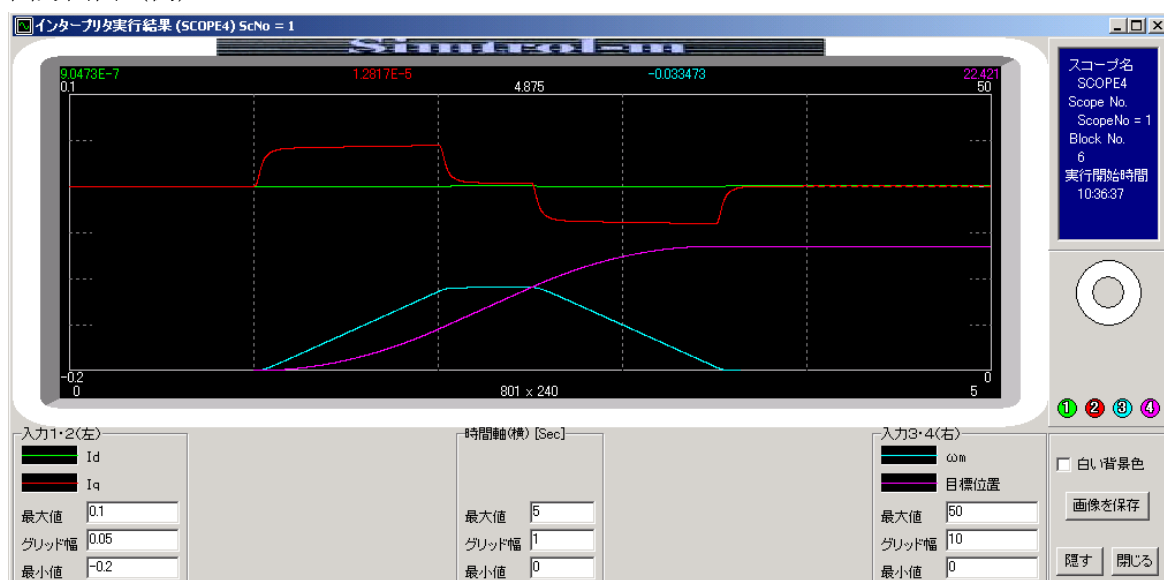
## 20005 : SCOPE 4 (4入力型オシロスコープ)



### 動作

4入力型のオシロスコープです。入力データを専用画面に表示出力します。  
時間変化に対するデータの変化を観測します。1入力～3入力型のスコープとしても使用できます。  
4入力とも時間軸は共通です。(入力データ間で異なる時間軸を使用する観測はできません。)

### 出力画面 (例)



(横軸：時間軸 (共通) / Y 軸 (縦軸)：入力データ値)  
(左 Y 軸グリッド：入力 1・入力 2 のデータ用  
/ 右 Y 軸グリッド：入力 3・入力 4 のデータ用)

### 入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	input : 入力 1	—	float 型	画面左 Y 軸
2	input : 入力 2	—	float 型	画面左 Y 軸
3	input : 入力 3	—	float 型	画面右 Y 軸
4	input : 入力 4	—	float 型	画面右 Y 軸

出力端子／データ：なし。



プロパティ・データ

No.	プロパティ・データ名	単位	データ型	備考
1	Tmin : 画面時間軸最小値	[sec]	float 型	0 のとき自動設定
2	Tmax : 画面時間軸最大値	[sec]	float 型	0 のとき自動設定
3	Tgrid : 画面時間軸目盛り刻み	[sec]	float 型	0 のとき自動設定
4	ymin1 : 左 Y 軸最小値	—	float 型	0 のとき自動設定
5	ymax1 : 左 Y 軸最大値	—	float 型	0 のとき自動設定
6	ygrid1 : 左 Y 軸目盛り刻み	—	float 型	0 のとき自動設定
7	ymin2 : 右 Y 軸最小値	—	float 型	0 のとき自動設定
8	ymax2 : 右 Y 軸最大値	—	float 型	0 のとき自動設定
9	ygrid2 : 右 Y 軸目盛り刻み	—	float 型	0 のとき自動設定
10	ScopeNo : スコープ番号	—	float 型	変更不可
11	buffsize: データ数 (1 入力あたり)	—	float 型	変更不可

対応関数

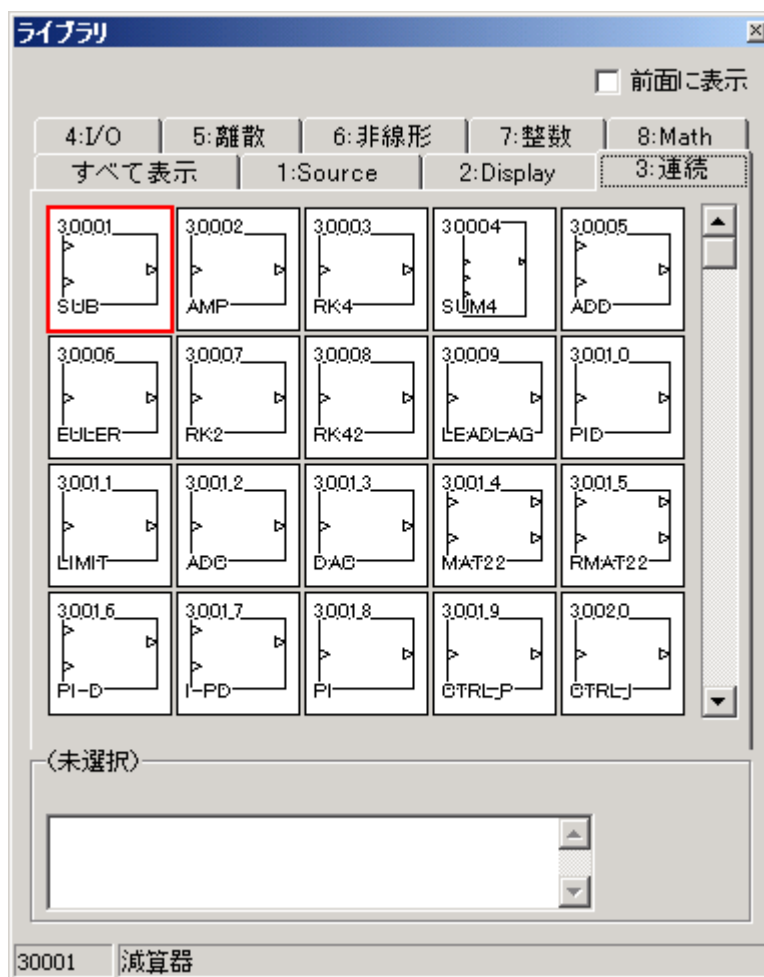
```
void fn_scope4 ( float input1, float input2, float input3, float input4,
                float *propaty, int *counter, int dataSize, float *data,
                int *err );
```

- ・入力データ 4 個に対し、Y 軸グリッドは左右の 2 軸のみです。
- ・入力 1・入力 2 (左 Y 軸) と入力 3・入力 4 (右 Y 軸) とが各々共通の Y 軸グリッドを使用します。
- ・ScopeNo: スコープ番号値及び buffsize: 処理用バッファサイズは自動で設定されます。ユーザによる設定は不用です。(ユーザによる値の変更もできません。)
- ・画面表示に必要な、時間軸／Y 軸の最大値／最小値及び、各目盛り刻み (グリッド) の値はそれぞれに 0 (ゼロ) を設定することで、自動設定 (既定) になります。
- ・プロパティの画面時間軸最小値に負の値 (-1 [sec] など) を設定された場合の動作は保証しません。
- ・実数値であれば基本的にどのような数値でも入力可能ですが、値が設定の範囲を超えた場合は、画面表示がされません。その他設定によって、画面に測定結果が表示されない場合があります。

## 2, 3、連続群ブロック／関数

線形システムの伝達関数及び、各種演算処理を実行するブロック群です。

伝達関数または演算処理のブロックですので、入力と出力とが共に必要になり、ブロックにも入力端子と出力端子が必ずあります。基本的にブロックの入力端子から入力されたデータを対応する関数で処理し、結果を出力端子に出力します。ブロックの入力端子＝対応する関数の入力変数に、正しい種類のデータを入力しなければ、正常な動作はできません。特にブロックに入力端子が複数個ある場合は、各データの対応にご注意ください。



連続群（ライブラリ表示）

連続群のブロックに対応する関数は、以下の形式になります。（一部例外を除く。）

```
void fn_***(float input, float *propaty, float *work, float *output, int *err)
```

- ・関数名の接頭語「fn\_」はSimtrol\_mの関数共通の接頭語です。
- ・\*propatyは、ブロックのプロパティ設定データの配列を示します。
- ・外部に処理領域が必要な関数の場合、関数特有の処理領域を外部変数として確保し、引数：float \*work（ポインタ）でそのアドレスを渡します。
- ・通常は、変数のアドレス渡し機能を利用して、引数 float \*output（ポインタ）を出力用変数として用います。関数自体は値を持ちません。
- ・int \*errは各関数共通のエラー処理用変数として定義されており、初期値＝0とします。

以下、連続群の各ブロック／関数を解説いたします。

---

3 0 0 0 1 : SUB (減算器)
-----------------------



動作

出力値 = 入力値 1 - 入力値 2 (減算)。

入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	input1 : 入力 1	—	float 型	
2	input2 : 入力 2	—	float 型	

出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	output : 出力	—	float 型	

プロパティ・データ：なし。

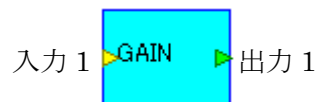
対応関数

```
void fn_sub (float input1, float input2, float *output , int *err);
```

- ・演算項（入力 1 - 入力 2）の順序に御注意下さい。
- ・浮動小数点型による減算なので、数値の精度（「桁落ち」など）にご注意ください。

---

3 0 0 0 2 : G A I N (増幅器／定数乗算器)
---------------------------------



動作

出力値 = 入力値・G a i n。 G a i n : 増幅率 (利得)。

入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	input : 入力	—	float 型	

出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	output : 出力	—	float 型	

プロパティ・データ

N o .	プロパティ・データ名	単位	データ型	備考
1	Gain : 増幅率 (利得)	—	float 型	実数値

対応関数

```
void fn_amp (float xin, float *propaty, float *xout, int *err);  
propaty[0] = Gain // 増幅率(利得)
```

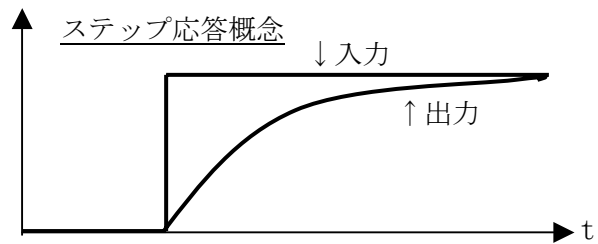
- ・定数の乗算器としても使用できます。
- ・プロパティ・データ G a i n の値は実数値で指定して下さい。( d B 値不可)
- ・浮動小数点演算につき、演算前後の数値の精度にご注意ください。

---

3 0 0 0 3 : R K 4 (一次遅れ要素: ルンゲ・クッタ 4 次法)



動作



ルンゲ・クッタ 4 次法による一次遅れ要素です。

伝達関数  $G(s) = 1 / (1 + T a u \cdot s)$ 。  $T a u$  : 時定数。

入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	input : 入力	—	float 型	

出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	output : 出力	—	float 型	

プロパティ・データ

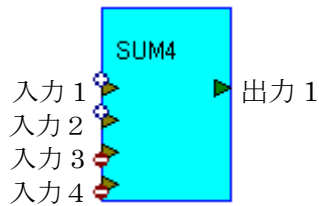
N o .	プロパティ・データ名	単位	データ型	備考
1	Tau : 時定数	[sec]	float 型	

対応関数

```
void fn_rk4(float xin, float *propaty, float *xout, int *err) ;
```

---

3 0 0 0 4 : SUM4 (4入力加算減算器)



#### 動作

出力 = 土入力1 土入力2 土入力3 土入力4。  
(入力端子毎に加算(+)/減算(-)を設定し、合計値を出力します。  
加算/減算はプロパティ・データにより、端子ごとに指定します。)  
・設定の状態は、ブロック上の各端子の脇に土の記号で表示されます。

#### 入力端子/データ

端子番号	端子名 (default)	単位	データ型	備考
1	input1 : 入力 1	—	float 型	
2	input2 : 入力 2	—	float 型	
3	input3 : 入力 3	—	float 型	
4	input4 : 入力 4	—	float 型	

#### 出力端子/データ

端子番号	端子名 (default)	単位	データ型	備考
1	output : 出力	—	float 型	

#### プロパティ・データ

No.	プロパティ・データ名	単位	データ型	備考
1	input1 : 入力 1	—	float 型	+ (加算): 0、- (減算): 1
2	input2 : 入力 2	—	float 型	+ (加算): 0、- (減算): 1
3	input3 : 入力 3	—	float 型	+ (加算): 0、- (減算): 1
4	input4 : 入力 4	—	float 型	+ (加算): 0、- (減算): 1

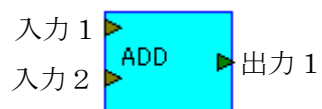
#### 対応関数

```
void fn_sum4 (float input1, float input2, float input3, float input4,  
             float *propaty, float *xout, int *err);
```

- ・浮動小数点演算で、かつ、取り扱う数値の数が多いので、数値の精度や「桁落ち」などの事象にご注意ください。
- ・プロパティ・データの設定では、指定値 (0、1) 以外の数値は使用しないで下さい。

---

3 0 0 0 5 : ADD (加算器)



動作

出力値 = 入力値 1 + 入力値 2 (加算)。

入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	input1 : 入力 1	—	float 型	
2	input2 : 入力 2	—	float 型	

出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	Output : 出力	—	float 型	

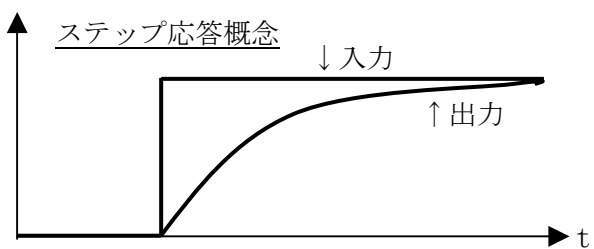
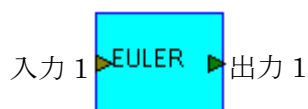
プロパティ・データ : なし。

対応関数

```
void fn_add (float input1, float input2, float *output , int *err);
```

- ・ 浮動小数点演算につき、演算時の数値の精度にご注意ください。

### 3 0 0 0 6 : EULER (一次遅れ要素：オイラー法)



#### 動作

オイラー法による一次遅れ要素。

伝達関数  $G(s) = 1 / (1 + T a u \cdot s)$ 。  $T a u$  : 時定数。

#### 入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	input : 入力	—	float 型	

#### 出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	output : 出力	—	float 型	

#### プロパティ・データ

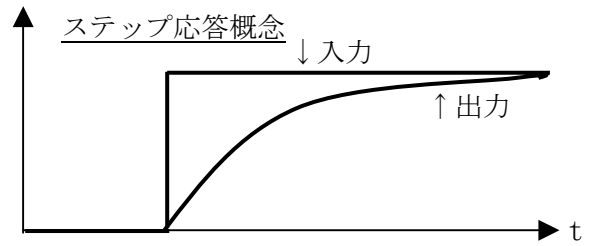
No.	プロパティ・データ名	単位	データ型	備考
1	Tau : 時定数	[sec]	float 型	

#### 対応関数

```
void fn_euler(float input1, float *propaty, float *output, int *err);
```



3 0 0 0 7 : R K 2 (一次遅れ要素：ルンゲ・クッタ 2 次法)



動作

ルンゲ・クッタ 2 次法による一次遅れ要素。

伝達関数  $G(s) = 1 / (1 + T a u \cdot s)$ 。  $T a u$  : 時定数。

入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	input : 入力	—	float 型	

出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	output : 出力	—	float 型	

プロパティ・データ

N o .	プロパティ・データ名	単位	データ型	備考
1	Tau : 時定数	[sec]	float 型	

対応関数

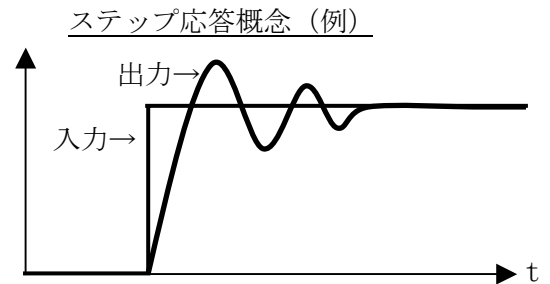
```
void fn_rk2(float xin, float *propaty, float *xout, int *err);
propaty[0] = tau ; //時定数[sec]
```

30008 : RK42 (二次遅れ要素 : ルンゲ・クッタ 4 次法)



#### 動作

ルンゲ・クッタ 4 次法による、二次遅れ要素です。  
伝達関数  $G(s) = 1 / (b - a \cdot s + s^2)$ 。



#### 入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	input : 入力	—	float 型	

#### 出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	output : 出力	—	float 型	

#### プロパティ・データ

No.	プロパティ・データ名	単位	データ型	備考
1	a : 係数 a	—	float 型	
2	b : 係数 b	—	float 型	

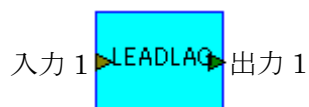
#### 対応関数

```
void fn_rk4_2ndo(float input, float *propaty, float *work, float *output,  
                 int *err) ;
```

・二次遅れ系は様々な伝達関数の様式がありますが、本関数をご利用の際は、  
上記の伝達関数の様式に換算の上、ご利用ください。

---

30009:LEADLAG (進み遅れ関数)
------------------------



#### 動作

システムの安定補償のため、位相進み遅れ補償を行なう関数ブロックです。

伝達関数  $G(s) = (a_0 + a_1 * s) / (b_0 + b_1 * s)$ 。

#### 入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	input : 入力	—	float 型	

#### 出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	output : 出力	—	float 型	

#### プロパティ・データ

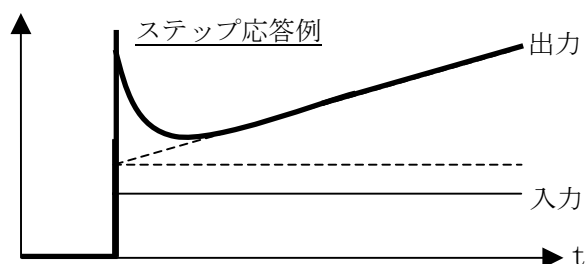
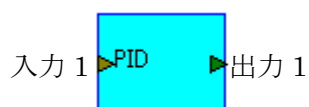
No.	プロパティ・データ名	単位	データ型	備考
1	a 0 : 係数 a 0	—	float 型	
2	a 1 : 係数 a 1	—	float 型	
3	b 0 : 係数 b 0	—	float 型	
4	b 1 : 係数 b 1	—	float 型	0 値禁止

#### 対応関数

```
void fn_leadlag (float input1, float *propaty, float *work, float *output, , int *err) ;
```

- ・ プロパティ・データの係数 b 1 値には 0 は設定不可です。未使用の場合は 1 を設定してください。
- ・ 係数 a 0 / a 1 が位相進み側（微分係数）、係数 b 0 / b 1 が位相遅れ側（積分係数）です。

30010:PID (PID制御器)



#### 動作

制御の比例定数  $k_p$  / 積分定数  $k_i$  / 微分定数  $k_d$  を指定し、入力の変化（偏差）に対するPID制御を実行します。

#### 入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	input : 入力	—	float 型	

#### 出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	output : 出力	—	float 型	

#### プロパティ・データ

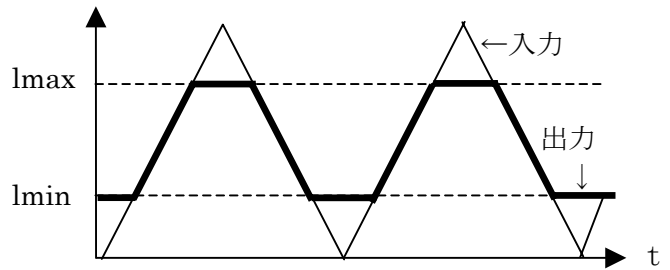
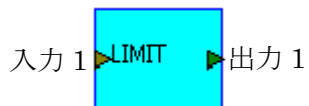
No.	プロパティ・データ名	単位	データ型	備考
1	Kp : 比例定数	—	float 型	
2	Ki : 積分定数	—	float 型	
3	Kd : 微分定数	—	float 型	

#### 対応関数

```
void fn_pid(float input, float *propaty, float *work, float *output, int *err);
```

- 基本型のPID制御器です。P/I/D各制御要素毎に定数の指定が可能です。
- 利用しない制御要素は、定数の値=0に指定して下さい。

# 3 0 0 1 1 : L I M I T (リミット)



## 動作

出力を指定範囲内に収めます。

(入力値が指定範囲内(最小値：  $lmin$  ～最大値：  $lmax$  )の値か否か、検証します。)

- ・ 指定範囲内の入力→入力値をそのまま出力します。
- ・ 指定範囲外の入力→指定の最小値／最大値を出力します。

## 入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	input : 入力	—	float 型	

## 出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	output : 出力	—	float 型	

## プロパティ・データ

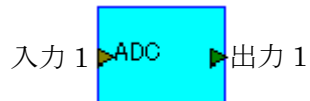
No.	プロパティ・データ名	単位	データ型	備考
1	$lmax$ : 最大値	—	float 型	
2	$lmin$ : 最小値	—	float 型	

## 対応関数

```
void fn_limit(float input1, float *propaty, float *output, int *err);
```

---

3 0 0 1 2 : ADC (A-Dコンバータ)
----------------------------



#### 動作

アナログ信号をデジタル信号に変換します。1 0 0 本（0～9 9）のチャンネルが利用できます。

#### 入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	input : 入力	—	float 型	アナログ信号

#### 出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	output : 出力	—	float 型	デジタル信号 (2 進数)

#### プロパティ・データ

N o .	プロパティ・データ名	単位	データ型	備考
1	adc_ch : チャンネル番号	—	float 型	0 ～ 9 9
2	bits : ビット数	—	float 型	ビット数 (分解能)
3	xmin : 最小値	—	float 型	A D 変換最小値 (入力側)
4	xmax : 最大値	—	float 型	A D 変換最大値 (入力側)
5	ct : 変換時間	[sec]	float 型	

#### 対応関数

```
void fn_adc(float input1, float *propaty, float *output, int *err);
```

- ・A D 変換最大値  $x_{max}$  と A D 変換最小値  $x_{min}$  に同じ値を入力しないで下さい。
- ・入力値を  $((x_{max} - x_{min}) / 2^{bits})$  単位で分割し、昇順に  $0 \sim (2^{bits}) - 1$  の 2 進数値 (扱いは 1 0 進整数) を割り当てます。  
変換時間  $ct$  ごとにそのときの入力値に応じた 2 進数値を出力します。出力された値はそのまま 1 0 進数の実数として扱います。

---

3 0 0 1 3 : DAC (D-Aコンバータ)
----------------------------



#### 動作

ディジタル信号をアナログ信号に変換します。

#### 入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	input : 入力	—	float 型	

#### 出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	output : 出力	—	float 型	

#### プロパティ・データ

No.	プロパティ・データ名	単位	データ型	備考
1	bits : ビット数	—	float 型	D A コンバータのビット数
2	xmin : 最小値	—	float 型	D A 変換処理の最小値
3	xmax : 最大値	—	float 型	D A 変換処理の最大値

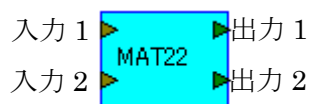
#### 対応関数

```
void fn_adc(float input1, float *propaty, float *output, int *err);
```

- ・プロパティにおいて、変換の最大値  $x_{max}$  / 最小値  $x_{min}$  は必ず指定して下さい。
- ・出力値 = 入力値 \*  $((x_{max} - x_{min}) / (2^{bits})) + x_{min}$ 、となります。

---

3 0 0 1 4 : MAT 2 2 (マトリクス演算)
-------------------------------



#### 動作

2行2列のマトリクスによる演算です。

(出力) (マトリクス) (入力)  
|data0| = |xin[11] xin[12]| |input0|  
|data1| |xin[21] xin[22]| |input1|

#### 入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	入力 : yin0	—	float 型	
2	入力 : yin1	—	float 型	

#### 出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	出力 : data0	—	float 型	
2	出力 : data1	—	float 型	

#### プロパティ・データ

No.	プロパティ・データ名	単位	データ型	備考
1	xin[11]	—	float 型	行列要素
2	xin[12]	—	float 型	行列要素
3	xin[21]	—	float 型	行列要素
4	xin[22]	—	float 型	行列要素

#### 対応関数

```
void fn_matrix22 (float input0, float input1, float *propaty,  
                  float *data0, float *data1, int *err);
```

・プロパティ各値が関数で使用するマトリクス要素の設定になります。入力の順序にご注意ください。



---

3 0 0 1 5 : R M A T 2 2 (逆マトリクス演算)



#### 動作

指定のマトリクス（2行2列）の逆マトリクスを求め、演算する関数です。  
逆マトリクスを求めた後、入力値に対しマトリクス演算を実行します。  
（マトリクスの要素はプロパティで指定します。）

（出力）      （マトリクス）      （入力）  
$$\begin{bmatrix} \text{data0} \\ \text{data1} \end{bmatrix} = \begin{bmatrix} \text{xin}[11] & \text{xin}[12] \\ \text{xin}[21] & \text{xin}[22] \end{bmatrix}^{-1} \begin{bmatrix} \text{input0} \\ \text{input1} \end{bmatrix}$$

#### 入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	入力 : yin0	—	float 型	
2	入力 : yin1	—	float 型	

#### 出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	出力 : data0	—	float 型	
2	出力 : data1	—	float 型	

#### プロパティ・データ

No.	プロパティ・データ名	単位	データ型	備考
1	xin[11]	—	float 型	行列要素
2	xin[12]	—	float 型	行列要素
3	xin[21]	—	float 型	行列要素
4	xin[22]	—	float 型	行列要素

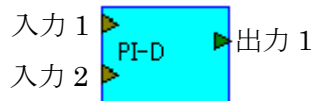
#### 対応関数

```
void fn_rmatrix22 (float input0, float input1, float *propaty,  
                  float *data0, float *data1, int *err);
```

- ・ プロパティ各値が関数で使用するマトリクス要素の設定になります。入力の順序にご注意ください。
- ・ 逆マトリクス演算につき、行列式 :  $(\text{xin}[11] * \text{xin}[22]) - (\text{xin}[12] * \text{xin}[21]) = 0$  となる値は設定禁止です。

---

3 0 0 1 6 : P I - D ( P I - D型制御器 : 測定値微分方式 P I D制御)
------------------------------------------------------



#### 動作

P I - D型制御器

制御の比例定数  $k_p$  / 積分定数  $k_i$  / 微分定数  $k_d$  を指定し、  
P I - D型制御 (測定値微分方式 P I D制御) を実行する関数です。

#### 入力端子 / データ

端子番号	端子名 (default)	単位	データ型	備考
1	pi : P I 入力	—	float 型	(目標値)
2	d : D 入力	—	float 型	(測定値)

#### 出力端子 / データ

端子番号	端子名 (default)	単位	データ型	備考
1	output : 出力	—	float 型	

#### プロパティ・データ

No.	プロパティ・データ名	単位	データ型	備考
1	Kp : 比例定数	—	float 型	
2	Ki : 積分定数	—	float 型	
3	Kd : 微分定数	—	float 型	

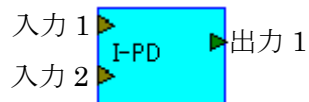
#### 対応関数

```
void fn_pi_d(float pi, float d, float *propaty, float *work, float *xout, int *err);
```

- ・ 入力変数 : pi が P I 入力 (目標値)、d が D 入力 (測定値) になります。

---

3 0 0 1 7 : I - P D ( I - P D型制御器 : 測定値比例微分方式 P I D制御)
--------------------------------------------------------



#### 動作

##### I - P D型制御器

制御の比例定数  $k_p$  / 積分定数  $k_i$  / 微分定数  $k_d$  を指定し、I - P D型制御 (測定値比例微分方式 P I D制御) を実行する関数です。

#### 入力端子 / データ

端子番号	端子名 (default)	単位	データ型	備考
1	i: I 入力	—	float 型	(目標値)
2	pd: P D入力	—	float 型	(測定値)

#### 出力端子 / データ

端子番号	端子名 (default)	単位	データ型	備考
1	output : 出力	—	float 型	

#### プロパティ・データ

No.	プロパティ・データ名	単位	データ型	備考
1	Kp : 比例定数	—	float 型	
2	Ki : 積分定数	—	float 型	
3	Kd : 微分定数	—	float 型	

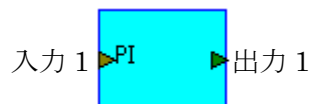
#### 対応関数

```
void fn_i_pd(float i, float pd, float *propaty, float *work, float *xout, int *err);
```

- ・入力変数 : i が I 入力 (目標値)、pd が P D入力 (測定値) になります。

---

3 0 0 1 8 : P I ( P I 制御器)
----------------------------



#### 動作

##### P I 制御器

制御の比例定数  $k_p$  / 積分定数  $k_i$  を指定し、P I 制御を実行する関数です。

#### 入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	input : 入力	—	float 型	

#### 出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	output : 出力	—	float 型	

#### プロパティ・データ

N o .	プロパティ・データ名	単位	データ型	備考
1	Kp : 比例定数	—	float 型	
2	Ki : 積分定数	—	float 型	

#### 対応関数

```
void fn_pi(float input, float *propaty, float *work, float *output, int *err);
```

---

3 0 0 1 9 : CTRL__P (P制御器)
----------------------------



#### 動作

P制御器（比例要素）

制御の比例定数  $k_p$  を指定し、P制御（比例制御）を実行する関数です。  
比例要素としても使えます。

#### 入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	input : 入力	—	float 型	

#### 出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	output : 出力	—	float 型	

#### プロパティ・データ

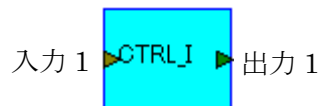
No.	プロパティ・データ名	単位	データ型	備考
1	Kp : 比例定数	—	float 型	

#### 対応関数

```
void fn_ctrl_p(float input, float *plopatty, float *output, int *err);
```

---

3 0 0 2 0 : CTRL_I (I 制御器)
----------------------------



#### 動作

I 制御器 (積分要素)

制御の比例定数  $k_i$  を指定し、I 制御 (積分制御) を実行する関数です。  
積分要素としても使えます。

#### 入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	input : 入力	—	float 型	

#### 出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	output : 出力	—	float 型	

#### プロパティ・データ

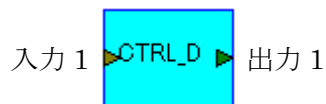
No.	プロパティ・データ名	単位	データ型	備考
1	$k_i$ : 積分定数	—	float 型	

#### 対応関数

```
void fn_ctrl_i(float input, float *plopatty, float *work, float *output, int *err);
```

---

3 0 0 2 1 : CTRL_D (D制御器)
---------------------------



#### 動作

D制御器 (微分要素)

制御の比例定数  $k_d$  を指定し、D制御(微分制御)を実行する関数です。

微分要素としても使えます。

#### 入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	input : 入力	—	float 型	

#### 出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	output : 出力	—	float 型	

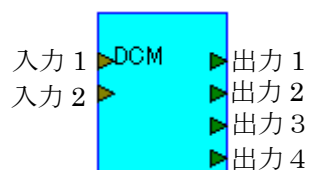
#### プロパティ・データ

No.	プロパティ・データ名	単位	データ型	備考
1	$k_d$ : 微分定数	—	float 型	

#### 対応関数

```
void fn_ctrl_i(float input, float *plopaty, float *work, float *output, int *err);
```

### 3 0 0 2 2 : DCM (DCモータ)



#### 動作

DCモータをシミュレーションする関数です。

プロパティ・データでモータの各定数（電機子抵抗[Ω]/モータパラメータなど）を指定します。モータの電機子電圧[V]と負荷トルク[N・m]を入力し、電機子電流[A]/回転角速度[rad/sec]などを求め、出力します。

#### 入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	Va : 電機子電圧	[V]	float 型	
2	TL : 負荷トルク	[N・m]	float 型	

#### 出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	ia : 電機子電流	[A]	float 型	
2	$\omega_m$ : モータの回転角速度	[rad/sec]	float 型	
3	$\theta_m$ : モータの回転角	[rad]	float 型	
4	Tm : モータの発生トルク	[N・m]	float 型	

#### プロパティ・データ

No.	プロパティ・データ名	単位	データ型	備考
1	Ra : 電機子巻線抵抗	[Ω]	float 型	0 禁止
2	La : 電機子インダクタンス	[H]	float 型	
3	J : 慣性モーメント	[kg・m <sup>2</sup> ]	float 型	
4	D : 粘性摩擦係数	[N・m・s/rad]	float 型	0 禁止
5	K : モータパラメータ	[V s /rad] または [N・m/A]	float 型	

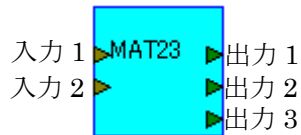
#### 対応関数

```
void fn_dcm(float Va, float TL, float *propaty, float *work, float *xout1,
            float *xout2, float *xout3, float *xout4, int *err);
```

- Ra : 電機子巻線抵抗[Ω]及びD:粘性摩擦係数[Nm・s/rad]の値には、ゼロ値：0を指定しないで下さい。



### 3 0 0 2 3 : MAT 2 3 (マトリクス演算)



#### 動作

(出力) (マトリクス) (入力)

|data0| = |xin[11] xin[12]| |yin0|

|data1| |xin[21] xin[22]| |yin1|

|data2| |xin[31] xin[32]|

(2入力)→(3出力)変換操作のマトリクス演算です。

プロパティ・データでマトリクスの要素値を指定します。

#### 入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	入力 : yin0	—	float 型	
2	入力 : yin1	—	float 型	

#### 出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	出力 : data0	—	float 型	
2	出力 : data1	—	float 型	
3	出力 : data2	—	float 型	

#### プロパティ・データ

No.	プロパティ・データ名	単位	データ型	備考
1	xin[11]	—	float 型	行列要素
2	xin[12]	—	float 型	行列要素
3	xin[21]	—	float 型	行列要素
4	xin[22]	—	float 型	行列要素
5	xin[31]	—	float 型	行列要素
6	xin[32]	—	float 型	行列要素

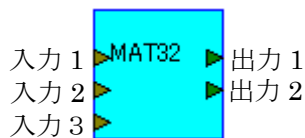
#### 対応関数

```
void fn_matrix23 (float yin0, float yin1, float *propaty, float *data0, float *data1,
                  float *data2, int *err);
```

- ・ プロパティ各値が行列要素の設定になります。マトリクスの配列にご注意ください。

---

### 3 0 0 2 4 : MAT 3 2 (マトリクス演算)



#### 動作

(出力)            (マトリクス)            (入力)

$$\begin{bmatrix} \text{data0} \\ \text{data1} \end{bmatrix} = \begin{bmatrix} \text{xin}[11] & \text{xin}[12] & \text{xin}[13] \\ \text{xin}[21] & \text{xin}[22] & \text{xin}[23] \end{bmatrix} \begin{bmatrix} \text{yin0} \\ \text{yin1} \\ \text{yin2} \end{bmatrix}$$

(3 入力)→(2 出力)変換操作のマトリクス演算です。  
プロパティ・データでマトリクスの要素値を指定します。

#### 入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	入力 : yin0	—	float 型	
2	入力 : yin1	—	float 型	
3	入力 : yin2	—	float 型	

#### 出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	出力 : data0	—	float 型	
2	出力 : data1	—	float 型	

#### プロパティ・データ

No.	プロパティ・データ名	単位	データ型	備考
1	xin[11]	—	float 型	行列要素
2	xin[12]	—	float 型	行列要素
3	xin[13]	—	float 型	行列要素
4	xin[21]	—	float 型	行列要素
5	xin[22]	—	float 型	行列要素
6	xin[23]	—	float 型	行列要素

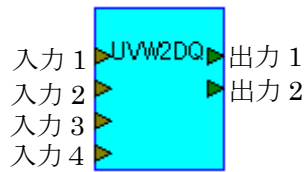
#### 対応関数

```
void fn_matrix32 (float yin0, float yin1, float yin2, float *propaty,  
                 float *data0, float *data1, int *err);
```

- ・ プロパティ各値が配列要素の設定になります。マトリクスの配列にご注意ください。

---

3 0 0 2 5 : U V W 2 D Q ( 3 相 → d q 変換 )



動作

d - q 変換

3 相交流 (U・V・W 相) 系 → d - q 座標系 (角速度回転座標系) 変換のブロックです。  
平衡 3 相交流波と位相の値 (定数) を入力端子に加えて d - q 変換し、出力します。

入力端子 / データ

端子番号	端子名 (default)	単位	データ型	備考
1	U : 3 相 U 相	—	float 型	
2	V : 3 相 V 相	—	float 型	
3	W : 3 相 W 相	—	float 型	
4	$\theta$ : 初期位相	[rad]	float 型	

出力端子 / データ

端子番号	端子名 (default)	単位	データ型	備考
1	d : 出力 d 軸	—	float 型	
2	q : 出力 q 軸	—	float 型	

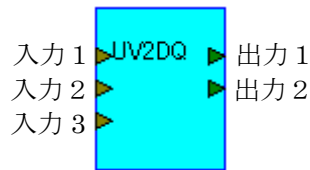
プロパティ・データ : なし。

対応関数

```
void fn_uvw2dq(float u, float v, float w, float th, float *data0, float *data1, int *err);
```

---

3 0 0 2 6 : UV2DQ (2相(3相)→d q 変換)
-----------------------------------



#### 動作

2相からのd-q変換を行なうブロックです。

3相交流(U相・V相)→d-q座標変換。(3相側の入力は2相のみを使用します。)

平衡3相交流の2相と位相の値(定数)を入力端子に加えてd-q変換し、出力します。

#### 入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	U : 3相U相	—	float 型	
2	V : 3相V相	—	float 型	
3	$\theta$ : 初期位相	[rad]	float 型	

#### 出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	d : 出力d軸	—	float 型	
2	q : 出力q軸	—	float 型	

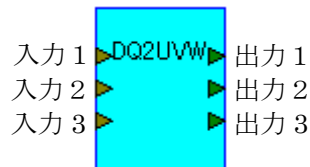
プロパティ・データ : なし。

#### 対応関数

```
void fn_uv2dq(float u, float v, float th, float *data0, float *data1, int *err);
```

---

3 0 0 2 7 : DQ2UVW (d q → 3 相変換)



#### 動作

d - q 逆変換。

d - q 座標系から 3 相交流 (U・V・W 相) に変換するブロックです。

d 軸 / q 軸の波形と位相を入力し、3 相交流に変換して出力します。

#### 入力端子 / データ

端子番号	端子名 (default)	単位	データ型	備考
1	d : d 軸	—	float 型	
2	q : q 軸	—	float 型	
3	$\theta$ : 初期位相	[rad]	float 型	

#### 出力端子 / データ

端子番号	端子名 (default)	単位	データ型	備考
1	U : 出力 3 相 U 相	—	float 型	
2	V : 出力 3 相 V 相	—	float 型	
3	W : 出力 3 相 W 相	—	float 型	

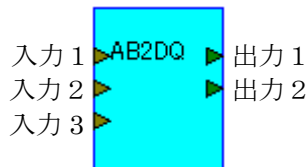
プロパティ・データ : なし。

#### 対応関数

```
void fn_dq2uvw(float d, float q, float th, float *data0, float *data1, float *data2,
               int *err);
```

---

3 0 0 2 8 : A B 2 D Q (  $\alpha$   $\beta$   $\rightarrow$  d q 変換 )



#### 動作

$\alpha$  -  $\beta$  座標系  $\rightarrow$  d - q 座標系変換。

$\alpha$  -  $\beta$  座標系（静止 2 相交流座標系）から d - q 座標系への変換を行なうブロックです。

$\alpha$  軸 /  $\beta$  軸の波形と位相を入力し、d 軸 / q 軸の波形に変換して出力します。

#### 入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	$\alpha$ : $\alpha$ 軸	—	float 型	
2	$\beta$ : $\beta$ 軸	—	float 型	
3	$\theta$ : 初期位相	[rad]	float 型	

#### 出力端／データ

端子番号	端子名 (default)	単位	データ型	備考
1	d : 出力 d 軸	—	float 型	
2	q : 出力 q 軸	—	float 型	

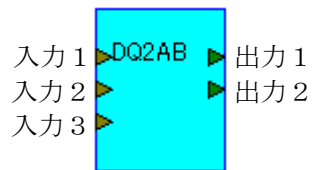
プロパティ・データ：なし。

#### 対応関数

```
fn_ab2dq(float a, float b, float th, float *data0, float *data1, int *err);
```

---

3 0 0 2 9 : D Q 2 A B ( d q → α β 変換)



動作

d - q 座標系 → α - β 座標系変換

d - q 座標系から α - β 座標系への変換を行なうブロックです。

d 軸 / q 軸の波形と位相を入力し、α 軸 / β 軸の波形に変換して、出力します。

入力端子 / データ

端子番号	端子名 (default)	単位	データ型	備考
1	α : α 軸	—	float 型	
2	β : β 軸	—	float 型	

出力端子 / データ

端子番号	端子名 (default)	単位	データ型	備考
1	d : 出力 d 軸	—	float 型	
2	q : 出力 q 軸	—	float 型	

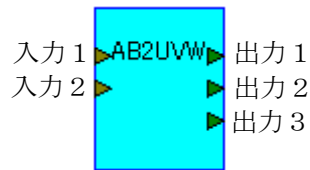
プロパティ・データ : なし。

対応関数

```
void fn_dq2ab(float d, float q, float th, float *data0, float *data1, int *err);
```

---

3 0 0 3 0 : A B 2 U V W ( $\alpha$   $\beta \rightarrow 3$ 相変換)



#### 動作

$\alpha - \beta$  座標系  $\rightarrow$  3 相交流 (U・V・W 相) 変換

$\alpha - \beta$  座標系から 3 相交流 (U・V・W 相) 系への変換を行なうブロックです。

$\alpha$  軸 /  $\beta$  軸の波形を入力し、3 相交流の波形に変換して、出力します。

#### 入力端子 / データ

端子番号	端子名 (default)	単位	データ型	備考
1	$\alpha$ : $\alpha$ 軸	—	float 型	
2	$\beta$ : $\beta$ 軸	—	float 型	

#### 出力端子 / データ

端子番号	端子名 (default)	単位	データ型	備考
1	U : 出力 3 相 U 相	—	float 型	
2	V : 出力 3 相 V 相	—	float 型	
3	W : 出力 3 相 W 相	—	float 型	

プロパティ・データ : なし。

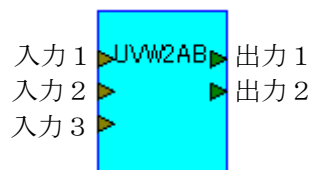
#### 対応関数

```
void fn_dq2uvw(float d, float q, float th, float *data0, float *data1, float *data2,
               int *err);
```



---

3 0 0 3 1 : U V W 2 A B ( 3 相 → $\alpha$ $\beta$ 変換 )
-------------------------------------------------------



#### 動作

3 相交流 (U・V・W 相) →  $\alpha$  -  $\beta$  座標系変換 ( $\alpha$  -  $\beta$  変換)

3 相交流 (U・V・W 相) 系 →  $\alpha$  -  $\beta$  座標系への変換を行なうブロックです。

平衡 3 相交流信号を入力端子に直接加えて、 $\alpha$  軸 /  $\beta$  軸の出力波形に変換して出力します。

#### 入力端子 / データ

端子番号	端子名 (default)	単位	データ型	備考
1	U : 3 相 U 相	—	float 型	
2	V : 3 相 V 相	—	float 型	
3	W : 3 相 W 相	—	float 型	

#### 出力端子 / データ

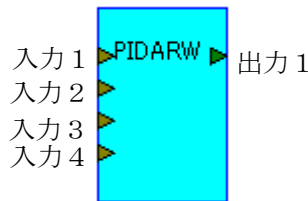
端子番号	端子名 (default)	単位	データ型	備考
1	$\alpha$ : 出力 $\alpha$ 軸	—	float 型	
2	$\beta$ : 出力 $\beta$ 軸	—	float 型	

#### 対応関数

```
void fn_uvw2ab(float u, float v, float w, float *data0, float *data1, int *err);
```

---

3 0 0 3 2 : P I D A R W (アンチ・リセットwindアップP I D制御器)
---------------------------------------------------



#### 動作

アンチ・リセットwindアップP I D制御器(微分先行型)

出力リミット／不感帯を設け、P I D制御の特性を改善した方式をシミュレーションします。

#### 入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	p v : 測定値	—	float 型	
2	s v : 設定値	—	float 型	
3	mvmin : 操作量最小値	—	float 型	
4	mvmax : 操作量最大値	—	float 型	

#### 出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	output : 出力	—	float 型	

#### プロパティ・データ

No.	プロパティ・データ名	単位	データ型	備考
1	k p : 比例定数	—	float 型	
2	T i : 積分時間	[sec]	float 型	
3	T d : 微分時間	[sec]	float 型	
4	p o l : 極性	—	float 型	
5	d b : 不感帯	—	float 型	

T i (積分時間) : 積分動作の出力が偏差 (設定値－測定値) と同じ値に達するまでの時間。  
値が小さいほど、積分動作が強く動作します。

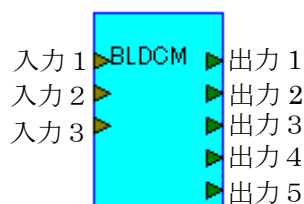
(T i = 0 はオン・オフ制御の状態です。)

T d (微分時間) : 一定時間で変化する偏差が微分動作の出力と等しくなるまでの時間。  
値が大きいほど、微分動作が強く動作します。

#### 対応関数

```
void fn_pidarw ( float pv, float sv, float mvmin, float mvmax, float *propaty,  
                float *work, float *mv, int *err ) ;
```

### 3 0 0 3 3 : B L D C M ( ブラシレス D C モータ : モータパラメータ )



#### 動作

##### ブラシレス D C モータ

モータパラメータ値その他の値を与え、ブラシレス D C モータの動作をシミュレーションします。各モータ特定の値をプロパティ・データとして与え、入力電圧値と負荷トルク値から、モータの回転角速度／発生トルク等を入力します。

#### 入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	V d : d 軸電圧	[V]	float 型	
2	V q : q 軸電圧	[V]	float 型	
3	T L : 負荷トルク	[N・m]	float 型	

#### 出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	i d : d 軸電流	[A]	float 型	
2	i q : q 軸電流	[A]	float 型	
3	$\Omega_m$ : モータの回転角速度	[rad/sec]	float 型	
4	$\Theta_m$ : モータの回転角	[rad]	float 型	
5	T m : モータの発生トルク	[N・m]	float 型	

#### プロパティ・データ

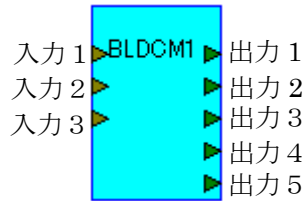
N o .	プロパティ・データ名	単位	データ型	備考
1	Ra : 巻線抵抗	[ $\Omega$ ]	float 型	0 不可
2	Ld : 電機子インダクタンス	[H]	float 型	
3	Lq : 電機子インダクタンス	[H]	float 型	
3	J : 慣性モーメント	[k g・m <sup>2</sup> ]	float 型	
4	D : 粘性摩擦係数	[N・m・s/rad]	float 型	0 不可
5	K : モータパラメータ	[V s /rad] または [N・m/A]	float 型	
6	P:極対数	—	float 型	0 以下不可

#### 対応関数

```
void fn_bldcm(float vd, float vq, float TL, float *bldcmparam, float *work,
              float *bldcmout0, float *bldcmout1, float *bldcmout2, float *bldcmout3,
              float *bldcmout4, int *err);
```

・巻線抵抗 Ra／粘性摩擦係数 D／極対数 P の値には、0（ゼロ）を入力しないで下さい。

### 3 0 0 3 4 : B L D C M 1 ( ブラシレス D C モータ : F l u x 値 )



#### 動作

ブラシレス D C モータ

Flux 値その他のパラメータ値を与え、ブラシレス D C モータの動作をシミュレーションします。

各モータ特定の値をプロパティ・データとして与え、入力電圧値と負荷トルク値から、モータの回転角速度／発生トルク等を求め、出力します。

#### 入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	V d : d 軸電圧	[V]	float 型	
2	V q : q 軸電圧	[V]	float 型	
3	TL : 負荷トルク	[N・m]	float 型	

#### 出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	i d : d 軸電流	[A]	float 型	
2	i q : q 軸電流	[A]	float 型	
3	$\omega m$ : モータの回転角速度	[rad/sec]	float 型	
4	$\theta m$ : モータの回転角	[rad]	float 型	
5	Tm : モータの発生トルク	[N・m]	float 型	

#### プロパティ・データ

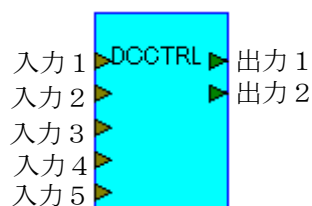
N o .	プロパティ・データ名	単位	データ型	備考
1	Ra : 巻線抵抗	[ $\Omega$ ]	float 型	0 不可
2	Ld : 電機子インダクタンス	[H]	float 型	
3	Lq : 電機子インダクタンス	[H]	float 型	
3	J : 慣性モーメント	[k g ・ m ^ 2]	float 型	
4	D : 粘性摩擦係数	[N ・ m ・ s / rad]	float 型	0 不可
5	Phi : Flux 値	—	float 型	
6	P : 極対数	—	float 型	0 以下以下

#### 対応関数

```
void fn_bldcm1(float vd, float vq, float TL, float *bldcmparam, float *work,
               float *bldcmout0, float *bldcmout1, float *bldcmout2, float *bldcmout3,
               float *bldcmout4, int *err);
```

・巻線抵抗 R a ／粘性摩擦係数 D ／極対数 P の値には、0 (ゼロ)を入力しないで下さい。  
また、極対数 P は 1 以上の整数値で指定して下さい。

### 3 0 0 3 5 : D C C T R L (ブラシレスDCモータの非干渉化：モータパラメータ)



#### 動作

##### ブラシレスDCモータの非干渉化制御

ブラシレスDCモータのd-q軸電流制御時に、d-q軸間での速度起電力による干渉を除去するため、供給されるd軸電圧/q軸電圧を、回転速度/d軸電流/q軸電流で補償する制御を実施するブロックです。制御のパラメータとしてモータパラメータ値を用います。

通常はブラシレスDCモータ・ブロック (3 0 0 3 3 : B L D C M) とともに用います。必要な入力データはモータ・ブロックの出力データをフィード・バックして使用します。

#### 入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	v d : d 軸電圧	[V]	float 型	
2	v q : q 軸電圧	[V]	float 型	
3	i d : d 軸電流	[A]	float 型	
4	i q : q 軸電流	[A]	float 型	
5	$\omega m$ : モータの回転角速度	[rad/sec]	float 型	

#### 出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	Vdc : 非干渉化された d 軸電圧	[V]	float 型	
2	Vqc : 非干渉化された q 軸電圧	[V]	float 型	

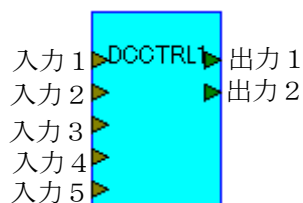
#### プロパティ・データ

No.	プロパティ・データ名	単位	データ型	備考
1	Ra : 巻線抵抗	[ $\Omega$ ]	float 型	(未使用)
2	Ld : 電機子インダクタンス	[H]	float 型	
3	Lq : 電機子インダクタンス	[H]	float 型	
3	J : 慣性モーメント	[ $kg \cdot m^2$ ]	float 型	(未使用)
4	D : 粘性摩擦係数	[ $N \cdot m \cdot s / rad$ ]	float 型	0 不可
5	K : モータパラメータ	[ $V s / rad$ ] または [ $N \cdot m / A$ ]	float 型	
6	P : 極対数	—	float 型	

#### 対応関数

```
void fn_dcctrl(float vd, float vq, float id, float iq, float omega, float *bldcmparam,
               float *bldcmout0, float *bldcmout1, int *err);
```

### 30036 : DCTRL1 (ブラシレスDCモータの非干渉化 : Flux 値)



#### 動作

##### ブラシレスDCモータの非干渉化制御

ブラシレスDCモータのd-q軸電流制御時に、d-q軸間での速度起電力による干渉を除去するため、供給されるd軸電圧/q軸電圧を、回転速度/d軸電流/q軸電流で補償する制御をシミュレーションするブロックです。制御のパラメータとしてFlux値を用います。

通常は30034 : BLDCM1ブロックとともに用います。必要な入力データはモータ・ブロックの出力データをフィード・バックして使用します。

#### 入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	v d : d 軸電圧	[V]	float 型	
2	v q : q 軸電圧	[V]	float 型	
3	i d : d 軸電流	[A]	float 型	
4	i q : q 軸電流	[A]	float 型	
5	$\omega m$ : モータの回転角速度	[rad/sec]	float 型	

#### 出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	Vdc : 非干渉化された d 軸電圧	[V]	float 型	
2	Vqc : 非干渉化された q 軸電圧	[V]	float 型	

#### プロパティ・データ

No.	プロパティ・データ名	単位	データ型	備考
1	Ra : 巻線抵抗	[ $\Omega$ ]	float 型	(未使用)
2	Ld : 電機子インダクタンス	[H]	float 型	
3	Lq : 電機子インダクタンス	[H]	float 型	
3	J : 慣性モーメント	[ $kg \cdot m^2$ ]	float 型	(未使用)
4	D : 粘性摩擦係数	[ $N \cdot m \cdot s/rad$ ]	float 型	0 不可
5	Phi : Flux 値	—	float 型	
6	P : 極対数	—	float 型	0 以下禁止

#### 対応関数

```
void fn_dctrl1(float vd, float vq, float id, float iq, float omega, float *bldcparam,
               float *bldcmout0, float *bldcmout1, int *err);
```

・巻線抵抗Ra／粘性摩擦係数D／極対数Pの値には、0 (ゼロ)を入力しないで下さい。  
また、極対数Pは1以上の整数値で指定して下さい。

---

3 1 0 0 2 : G A I N (増幅器／定数乗算器)
---------------------------------



#### 動作

出力値 = 入力値 ・ G a i n。 G a i n : 増幅率 (利得)。

#### 入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	input : 入力	—	float 型	

#### 出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	output : 出力	—	float 型	

#### プロパティ・データ

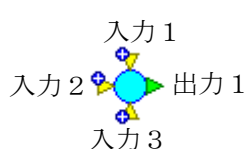
N o .	プロパティ・データ名	単位	データ型	備考
1	Gain : 増幅率 (利得)	—	float 型	実数値

#### 対応関数

```
void fn_amp (float xin, float *propaty, float *xout, int *err);
```

- ・ G a i n の値は実数値で指定して下さい。(d B 値不可)
- ・ 浮動小数点演算につき、演算前後の数値の精度にご注意ください。

# 3 1 0 0 4 : 2 / 3 入力加減演算器



(使用例)

入力 1 : 非表示 (未使用)  
 入力 2 : 減算(-)  
 入力 3 : 加算(+)

## 動作

2 入力 / 3 入力の加算・減算を実行します。

出力 =  $\pm$  入力 1  $\pm$  入力 2  $\pm$  入力 3。

(入力端子毎に加算(+)/減算(-)/(非表示)を設定し、実行します。)

## 入力端子 / データ

端子番号	端子名 (default)	単位	データ型	備考
1	input1 : 入力 1	—	float 型	
2	input2 : 入力 2	—	float 型	
3	input3 : 入力 3	—	float 型	

## 出力端子 / データ

端子番号	端子名 (default)	単位	データ型	備考
1	output : 出力	—	float 型	

## プロパティ・データ

No.	プロパティ・データ名	単位	データ型	備考
1	input1 : 入力 1 (上)	—	float 型	加算: 0、減算: 1、非表示: -1。
2	input2 : 入力 2 (左)	—	float 型	加算: 0、減算: 1、非表示: -1。
3	input3 : 入力 3 (下)	—	float 型	加算: 0、減算: 1、非表示: -1。

## 対応関数

```
void fn_sum3 (float input1, float input2, float input3, float *propaty, float *xout,
             int *err);
```

- ・ プロパティ・データ値を(-1)に設定すると、その端子は画面上に表示されなくなります。ご注意ください。
- ・ 端子に線が結線された状態で、その端子のプロパティ・データ値を(-1)に設定すると、端子は非表示状態に変わり、線の結線は解除されます。プロパティ・データ値を変更して、端子を再度表示しても、端子への線の結線は復元されませんので、ご注意ください。
- ・ 各端子の位置は、ライブラリ画面上での標準表示に基づくものです。ブロックを画面上で回転させる際には端子の位置にご注意ください。



## 2, 4、I/O群ブロック／関数

ターゲット・ボードの制御用ソフトウェアとして不可欠な部分にボードへの各種データの入出力インターフェース（I/O機能）の制御に関わる部分があります。但し、この部分はシステムのハードウェアの構成や機能に依存する割合が高いため、ライブラリ形式の関数として共通化することが困難な面があります。

Simtrol-mでは、特にモータ制御／組込み処理用のターゲット・ボードとして当社が推奨する、KENTAC13600ボード・シリーズ\* に用いることができる形式のI/O制御用関数を作成し、ライブラリに“I/O群”として登録致しました。

なお、これらI/O群の関数は、ターゲット・ボード上でのI/O機能のみでなく、Simtrol-mのエディタ／インタープリタ／コンパイラによる直接の処理機能も備えております。

また、特にI/O群の関数は、パソコンでのシミュレーションの実行時とマイコン・ボード上に実装して制御に使用する場合とでは、同じ関数でも処理動作が異なる場合がありますので、ご注意ください。

特に本章では関数の使用状態を、「シミュレーション時」（パソコン上でのインタープリタによるシミュレーション実行時）、及び「コントロール時」（ターゲット・ボード上での各種処理の実行時）に分けて考えます。

\* KENTAC13600ボード・シリーズの概要（詳細は別紙「Simtrol-m ユーザーズ・マニュアル 応用・実践編などをご参照ください。）

CPU：SH2 70855（32ビット／64MHz）。

A/D：内部10bit／8ch。

D/A：シリアル12ビット／4ch。

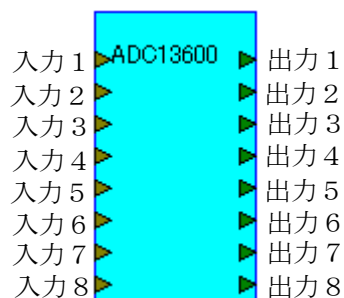
メモリ：ROM512kB、RAM16kB。



I/O群（ライブラリ表示）

- 
- \* ブロック／関数の名称に含まれる「13600」は、特にKENTAC13600ボード用のブロック／関数であることを示します。
  - \* I／O群の関数は直接、ハード・ウェアの制御処理を取り扱うので、入出力データの数が多くなっています。
  - \* 「KENTAC」は株式会社 昭和電業社の登録商標です。

4 0 0 0 1 : ADC 1 3 6 0 0 (1 3 6 0 0 用 A-D コンバータ)



#### 動作

1 3 6 0 0 用の A-D コンバータです。

KENTAC 1 3 6 0 0 ボードに内臓の、分解能 1 0 bit / 入出力 8 チャンネルの A-D コンバータです。

アナログ入力の変換最小値  $X_{min}$  を出力最小値 : 0000000000 (2 進) = 0、

アナログ入力の変換最大値  $X_{max}$  を出力最大値 : 1111111111 (2 進) = 1 0 2 3 とみなし、その範囲のアナログ入力に比例する出力を求めます。

出力値 =  $(1 0 2 4 * (\text{アナログ入力値} - X_{min}) / (X_{max} - X_{min})) - 1$ 。

但し、アナログ入力値 <  $X_{min}$  のとき出力 = 0。

アナログ入力値 >  $X_{max}$  のとき出力 = 1 0 2 3。

- ・ シミュレーション時は入力端子のアナログ量をデジタル量に変換し、出力端子に出力します。
- ・ コントロール時は AD ポートから入力した信号を出力端子に出力します。  
この場合は入力端子とプロパティ・データは使用しません。

#### 入力端子 / データ

端子番号	端子名 (default)	単位	データ型	備考
1	input1 : 入力 1	—	float 型	
2	input2 : 入力 2	—	float 型	
3	input3 : 入力 3	—	float 型	
4	input4 : 入力 4	—	float 型	
5	input5 : 入力 5	—	float 型	
6	input6 : 入力 6	—	float 型	
7	input7 : 入力 7	—	float 型	
8	input8 : 入力 8	—	float 型	

#### 出力端子 / データ

端子番号	端子名 (default)	単位	データ型	備考
1	output1 : 出力 1	—	float 型	
2	output2 : 出力 2	—	float 型	
3	output3 : 出力 3	—	float 型	
4	output4 : 出力 4	—	float 型	
5	output5 : 出力 5	—	float 型	
6	output6 : 出力 6	—	float 型	
7	output7 : 出力 7	—	float 型	
8	output8 : 出力 8	—	float 型	

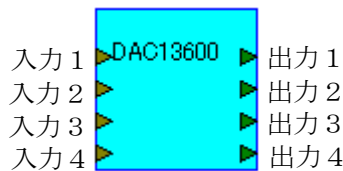
プロパティ・データ

No.	プロパティ・データ名	単位	データ型	備考
1	Xmin : 変換最小値	—	Float 型	
2	Xmax : 変換最大値	—	Float 型	
3	Ct : 変換時間	[sec]	Float 型	

対応関数

```
void fn_adc13600 (float input1, float input2, float input3, float input4,
                  float input5, float input6, float input7, float input8,
                  float *propaty, float *work,
                  float *output1, float *output2, float *output3, float *output4,
                  float *output5, float *output6, float *output7, float *output8,
                  int *err);
```

4 0 0 0 2 : DAC 1 3 6 0 0 (1 3 6 0 0 用 D-A コンバータ)



動作

1 3 6 0 0 用の分解能 1 2 bit / 入出力 4 チャンネルの D-A コンバータです。  
 DA コンバータの出力 = ((入力値 / 分解能) \* (出力範囲の大きさ)) + (出力最小値) より、  
 出力値 = ((各入力値 / 4 0 9 6) \* (出力最大値 X m a x - 出力最小値 X m i n)) +  
 (出力最小値 X m i n) です。

- ・ シミュレーション時は入力端子のデジタル量をアナログ信号に変換して、出力端子に出力します。
- ・ コントロール時は入力端子の信号を D A ポートに出力します。  
出力端子とプロパティは使用しません。

入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	input1 : 入力 1	—	float 型	
2	input2 : 入力 2	—	float 型	
3	input3 : 入力 3	—	float 型	
4	input4 : 入力 4	—	float 型	

出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	output1 : 出力 1	—	float 型	
2	output2 : 出力 2	—	float 型	
3	output3 : 出力 3	—	float 型	
4	output4 : 出力 4	—	float 型	

プロパティ・データ

No.	プロパティ・データ名	単位	データ型	備考
1	Xmin : 出力最小値	—	float 型	
2	Xmax : 出力最大値	—	float 型	

対応関数

```
void fn_dac13600 (float input1, float input2, float input3, float input4, float *propaty,  
float *output1, float *output2, float *output3, float *output4, int *err);
```

4 0 0 0 3 : PWM 1 3 6 0 0 ( 1 3 6 0 0 用 PWM )



動作

PWM制御を行なう関数です。

- ・ シミュレーション時は入力端子の信号を出力端子にそのまま出力します。
- ・ コントロール時は入力端子の信号をパルスのデューティ比 (0～1000. 0%) に変換し、PWM出力します。このときには出力端子は使用しません。

入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	input1 : 入力 1	—	float 型	
2	input2 : 入力 2	—	float 型	
3	input3 : 入力 3	—	float 型	

出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	output1 : 出力 1	—	float 型	
2	output2 : 出力 2	—	float 型	
3	output3 : 出力 3	—	float 型	

プロパティ・データ : なし。

対応関数

```
void fn_pwm13600 (float input1, float input2, float input3, float *output1,  
float *output2, float *output3, int *err);
```

4 0 0 0 4 : ENC 1 3 6 0 0 (1 3 6 0 0 用エンコーダ①)



#### 動作

- ・シミュレーション時は入力端子の信号を出力端子にそのまま出力します。
- ・コントロール時はエンコーダから検出した情報を出力するので、入力端子は使用しません。

float propaty[0] Pe : エンコーダの一周パルス数(1000 または 2000) [p/r]

float propaty[1] Dp : エンコーダのカウント方向 (0:正方向/1:逆方向)

#### 入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	input1 : 入力 1	—	float 型	
2	input2 : 入力 2	—	float 型	
3	input3 : 入力 3	—	float 型	

#### 出力端子／データ名

端子番号	端子名 (default)	単位	データ型	備考
1	$\omega m$ : モータの回転角速度	[rad/sec]	float 型	
2	$\theta m$ : モータの回転角	[rad]	float 型	
3	Rdy : エンコーダ準備完了 (0:ゼロ位置未検出 1:準備完了)	[rad/sec]	float 型	モータの回転角速度を検出

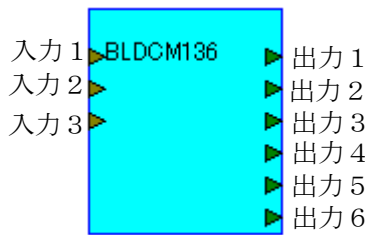
#### プロパティ・データ

No.	プロパティ・データ名	単位	データ型	備考
1	Pe : エンコーダの 1 週パルス数	[ p / r ]	float 型	1000/2000
2	Dp : エンコーダのカウント方向	—	float 型	0 : 正方向 / 1 : 逆方向

#### 対応関数

```
void fn_enc13600 (float input1,float input2,float input3,float *propaty, float
*outoutput1,float *outoutput2,float *outoutput3, int *err);
```

# 4 0 0 0 5 : B L D C M 1 3 6 0 0 ( 1 3 6 0 0 用 ブラシレス D C モータ )



## 動作

この関数は、ブラシレスDCモータをシミュレーションする関数です。モータパラメータ値を利用します。

具体的にはユーザが設定したモータパラメータ値は `bldcmparam` という配列に格納され、利用されます。また、計算結果は `bldcmout` という配列に格納されます。

シミュレーション時はブラシレスDCモータをシミュレーションします。  
コントロール時は  $V_d$ ,  $V_q$  を3相交流 (UVW) に変換し、デューティ比 (0~1000.0%) として、PWM出力します。

シミュレーション時は  $V_s$  は処理しません。

コントロール時は  $V_d$ ,  $V_q$ ,  $V_s$ ,  $P$ ,  $I_d$ ,  $I_q$ ,  $\omega_m$ ,  $\theta_m$ ,  $P_e$ ,  $D_p$  のみ処理し、

$T_L$ ,  $R_a$ ,  $L_d$ ,  $L_q$ ,  $J$ ,  $D$ ,  $K$ ,  $T_m$  は処理しません。

・PWM: パルス幅変調 (Pulse Width Modulation)。

## 入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	$V_d$ : d 軸電圧	[V]	float 型	
2	$V_q$ : q 軸電圧	[V]	float 型	
3	$T_L$ : 負荷トルク	[N・m]	float 型	

## 出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	$i_d$ : d 軸電流	[A]	float 型	
2	$i_q$ : q 軸電流	[A]	float 型	
3	$\Omega_m$ : モータの回転角速度	[rad/sec]	float 型	
4	$\Theta_m$ : モータの回転角	[rad]	float 型	
5	$T_m$ : モータの発生トルク	[N・m]	float 型	
6	$\theta_s$ : モータの回転角積算値	[rad]	float 型	

## プロパティ・データ

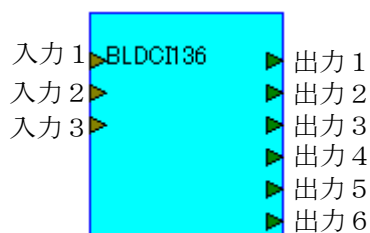
No.	プロパティ・データ名	単位	データ型	備考
1	$R_a$ : 巻線抵抗	[ $\Omega$ ]	float 型	0 不可
2	$L_d$ : d 軸インダクタンス	[H]	float 型	
3	$L_q$ : q 軸インダクタンス	[H]	float 型	
4	$J$ : 慣性モーメント	[kg・m <sup>2</sup> ]	float 型	
5	$D$ : 粘性摩擦係数	[N・m・s/rad]	float 型	0 不可
6	$K$ : モータパラメータ	[Vs/rad] または [Nm/A]	float 型	
7	$P$ : 極対数	—	float 型	0 以下は不可
8	$V_s$ : 電圧スケール	[V]	float 型	
9	$P_e$ : エンコーダの1週パルス数	[p/r]	float 型	1000/2000
10	$D_p$ : エンコーダのカウント方向	—	float 型	0 : 正方向 / 1 : 逆方向

- ・巻線抵抗  $R_a$  / 粘性摩擦係数  $D$  / 極対数  $P$  の値には、0 (ゼロ) を入力しないで下さい。  
また、極対数  $P$  は 1 以上の整数値で指定して下さい。

対応関数

```
void fn_bldcm13600(float vd, float vq, float TL, float *bldcmparam, float *work,
float *bldcmout0, float *bldcmout1, float *bldcmout2, float *bldcmout3,
float *bldcmout4, float *bldcmout5, int *err);
```

4 0 0 0 6 : B L D C M 1 3 6 0 0 ( 1 3 6 0 0 用 整数型 ブラシレス D C モータ )



動作

KENTAC 1 3 6 0 0 用 整数型 ブラシレス D C モータ ( B L D C M ) 関数

ブラシレス D C モータのシミュレーションを整数型で実現した関数です。  
シミュレーション時はブラシレス D C モータをシミュレーションします。  
コントロール時は  $\text{int\_Vd}$  (d 軸電圧),  $\text{int\_Vq}$  (q 軸電圧) を U V W ( 3 相交流 ) に変換し、  
デューティ比 ( 0 ~ 100. 00% ) として、PWM 出力します。  
コントロール時は  $V_d, V_q, P, I_d, I_q, \omega_m, \theta_m$  のみ処理します。  
 $TL, R_a, L_d, L_q, J, D, K, T_s, V_s, I_s$  は処理しません。  
モータパラメータは `bldcmparam` という配列に格納され、利用されます。  
また、計算結果は `bldcmout` という配列に格納され、出力されます。

注意事項)

この関数を使うときは E N C 1 3 6 0 0 ブロック ( `fn_enc13600` ) または P W M 1 3 6 0 0  
ブロック ( `fn_pwm13600` ) は使用できません。(動作不定)

入力端子 / データ

端子番号	端子名 (default)	単位	データ型	備考
1	$V_d$ : d 軸電圧	[V]	Int 型	-10000 ~ 10000 が - $V_s$ ~ $V_s$ [V] に相当
2	$V_q$ : q 軸電圧	[V]	Int 型	-10000 ~ 10000 が - $V_s$ ~ $V_s$ [V] に相当
3	$TL$ : 負荷トルク	[N・m]	Int 型	-10000 ~ 10000 が - $T_s$ ~ $T_s$ [N・m] に相当



## 出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	i d : d 軸電流	[A]	int 型	-10000～10000 が <sup>3</sup> -Is～ Is[A]相当 ・ bldcmout[0]
2	i q : q 軸電流	[A]	int 型	-10000～10000 が <sup>3</sup> -Is～ Is[A]相当 ・ bldcmout[1]
3	$\omega_m$ : モータの回転角速度	[rad/sec]	int 型	-4000～4000 が <sup>3</sup> $-2\pi/dt \sim$ $2\pi/dt$ [rad/sec] ・ bldcmout[2]
4	$\theta_m$ : モータの回転角	[rad]	int 型	-4000～4000 が <sup>3</sup> $-2\pi/dt \sim$ $2\pi/dt$ [rad/sec] ・ bldcmout[3]
5	Tm : モータの発生トルク	[N・m]	int 型	-10000～10000 が <sup>3</sup> -Ts～ Ts[N・m] ・ bldcmout[4]
6	$\theta_s$ : モータの回転角積算値	[rad]	int 型	-4000～4000 が <sup>3</sup> $-2\pi \sim 2\pi$ [rad] ・ bldcmout[5]

- ・ 出力値は配列 bldcmout [] に格納します。
- ・ d t は処理の刻み時間。(時間軸方向の処理単位幅)

## プロパティ・データ

No.	プロパティ・データ名	単位	データ型	備考
1	Ra : 巻線抵抗	[ $\Omega$ ]	float 型	0 不可
2	Ld : d 軸インダクタンス	[H]	float 型	
3	Lq : q 軸インダクタンス	[H]	float 型	
4	J : 慣性モーメント	[k g・m <sup>2</sup> ]	float 型	
5	D : 粘性摩擦係数	[N・m・s/rad]	float 型	
6	K : モータパラメータ	[Vs/rad]または [Nm/A]	float 型	
7	P : 極対数	—	float 型	0 以下は不可
8	Vs : 電圧スケール	[V]	float 型	
9	Ts : トルクスケール	[N・m]	float 型	
10	Is : 電流スケール	[A]	float 型	
11	Pe : エンコーダの1週パルス数	[p/r]	float 型	1000/2000
12	Dp : エンコーダのカウント方向	—	float 型	0 : 正方向/ 1 : 逆方向

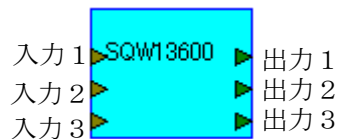
- ・ 巻線抵抗 R a /粘性摩擦係数 D /極対数 P の値には、0 (ゼロ)を入力しないで下さい。  
また、極対数 P は1以上の整数値で指定して下さい。

## 対応関数

```
void fn_bldci13600(int_vd, int_vq, int_TL, float *bldcmparam, float *work,
int *int_work, int *bldcmout[0], int *bldcmout[1], int *bldcmout[2], int *bldcmout[3],
int *bldcmout[4], int *bldcmout[5], int *err);
```

---

4 0 0 0 7 : S Q W 1 3 6 0 0 ( 1 3 6 0 0 用 1 2 0 度 矩 形 波 駆 動 出 力 )
-------------------------------------------------------------------



#### 動作

ブラシレスDCモータの3相矩形波駆動用波形を生成し、出力します。

具体的には、ホールセンサの信号により、120度矩形波方式でブラシレスDCモータを駆動制御するハードウェアを表わす関数です。

- ・シミュレーション時は入力端子の信号を出力端子にそのまま出力します。
- ・コントロール時は入力端子の信号をデューティ比(0~100.00%)として、PWM出力するので出力端子は使用しません。

#### 入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	input1 : 入力 1	—	float 型	
2	input2 : 入力 2	—	float 型	
3	input3 : 入力 3	—	float 型	

#### 出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	output1 : 出力 1	—	float 型	
2	output2 : 出力 2	—	float 型	
3	output3 : 出力 3	—	float 型	

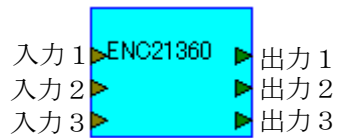
プロパティ・データ : なし。

#### 対応関数

```
void fn_sqw13600 (float input1,float input2,float input3, float *output1,float *output2,float *output3, int *err);
```

---

4 0 0 0 8 : ENC 2 1 3 6 0 0 (1 3 6 0 0用エンコーダ②)
------------------------------------------------



#### 動作

- ・シミュレーション時は入力端子の信号を出力端子にそのまま出力します。
- ・コントロール時はエンコーダから検出した情報を出力端子に出力します。  
入力端子は使用しません。

#### 入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	input1 : 入力 1	—	float 型	
2	input2 : 入力 2	—	float 型	
3	input3 : 入力 3	—	float 型	

#### 出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	output1 : 出力 1	—	float 型	
2	output2 : 出力 2	—	float 型	
3	output3 : 出力 3	—	float 型	

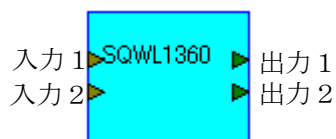
プロパティ・データ : なし。

#### 対応関数

```
void fn_enc213600 (float input1, float input2, float input3, float *output1,  
                  float *output2, float *output3, int *err);
```

---

4 0 0 0 9 : S Q W L 1 3 6 0 0 ( 1 3 6 0 0 用 1 2 0 度 矩 形 波 駆 動 出 力 遅 れ 制 御 付 )
-------------------------------------------------------------------------------



#### 動作

ブラシレスDCモータの3相矩形波駆動用波形を生成し、出力します。

具体的には、ホールセンサの信号により、120度矩形波方式でブラシレスDCモータを駆動制御するハードウェアを表わす関数です。位相制御機能付です。

- ・シミュレーション時は入力端子の信号を出力端子にそのまま出力します。このときは出力端子2の信号は処理しません。
- ・コントロール時は入力端子1の信号をデューティ比(0~100.00%)として、PWM出力するため、出力端子は使用しません。

- ・この関数はENC13600 (fn\_enc13600) との共用は不可です。

#### 入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	Duty : duty 比	[%]	float 型	0~100
2	Lag : 遅れ進み設定値	[deg]	float 型	-30~+30 (+で遅れ)

#### 出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	Duty : duty 比	[%]	float 型	0~100
2	$\omega e$ : 電気角速度	[rad/sec]	float 型	

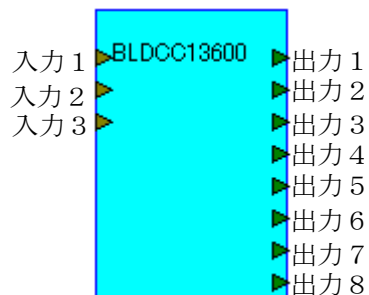
プロパティ・データ : なし。

#### 対応関数

```
void fn_sqwl13600 (float input1, float LL, float *output1, float *output2, int *err);
```

---

4 0 0 1 5 : B L D C C 1 3 6 0 0 ( 1 3 6 0 0 用 ブラシレス D C モータ 電流制御付)
--------------------------------------------------------------------



#### 動作

電流制御機能付きの 1 3 6 0 0 用 ブラシレス D C モータ関数です。

- ・ シミュレーション時はブラシレス D C モータをシミュレーションします。
- ・ コントロール時は電流入力値：  $I_{ds}$  /  $I_{qs}$  を目標値として  
モータの d 軸電流  $I_d$  / q 軸電流  $I_q$  をコントロールし、出力します。

#### 入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	$I_{ds}$ : d 軸電流設定値	[A]	float 型	
2	$I_{qs}$ : q 軸電流設定値	[A]	float 型	
3	TL : 負荷トルク	[N・m]	float 型	

#### 出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	$i_d$ : d 軸電流	[A]	float 型	
2	$i_q$ : q 軸電流	[A]	float 型	
3	$\omega_m$ : モータの回転角速度	[rad/sec]	float 型	
4	$\theta_m$ : モータの回転角	[rad]	float 型	
5	$T_m$ : モータ発生トルク	[N・m]	float 型	
6	$\theta_s$ : モータの回転角積算値	[rad]	float 型	
7	$v_d$ : d 軸電圧	[V]	float 型	
8	$v_q$ : q 軸電圧	[V]	float 型	

プロパティ・データ

№.	プロパティ・データ名	単位	データ型	備考
1	Ra : 巻線抵抗	[Ω]	float 型	0 不可
2	Ld : d 軸インダクタンス	[H]	float 型	
3	Lq : q 軸インダクタンス	[H]	float 型	
4	J : 慣性モーメント	[K g ・ m <sup>2</sup> ]	float 型	
5	D : 粘性摩擦係数	[Nm ・ s /rad]	float 型	0 不可
6	K : モータパラメータ	[Vs/rad] または [Nm/A]	float 型	
7	P : 極対数	—	float 型	0 以下は不可
8	Pe : エンコーダの一周パルス数	[p /r]	float 型	1000 または 2000
9	Dp : エンコーダのカウント方向	—	float 型	0 : 正方向 1 : 逆方向
10	Kp : 比例定数	—	float 型	
11	Ti : 積分時間	[sec]	float 型	
12	Rs : リミットスイッチ検出処理	—	float 型	0:無し/1:有り

・巻線抵抗 R a / 粘性摩擦係数 D / 極対数 P の値には、0 (ゼロ) を入力しないで下さい。  
また、極対数 P は 1 以上の整数値で指定して下さい。

対応関数

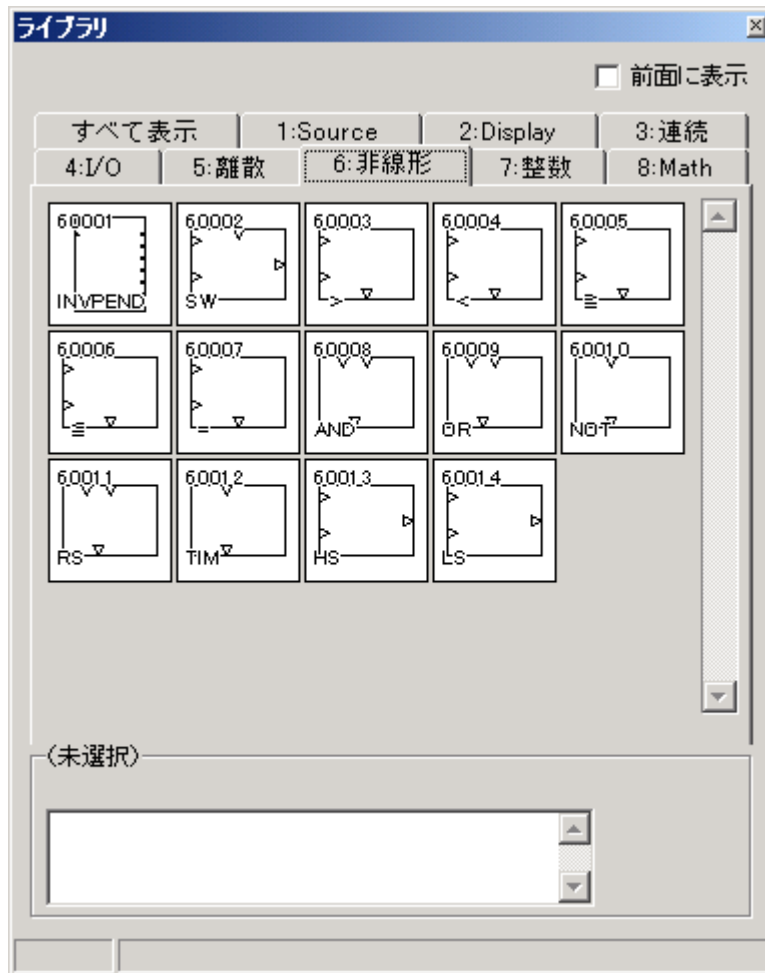
```
void fn_bldcc13600( float Ids, float Iqs, float TL, float *propaty, float *work1,
float *id, float *iq, float *ωm, float *θm, float *Tm, float *θs, float *vd,
float *vq, int *err );
```

## 2, 5、離散群ブロック／関数（該当関数なし）

## 2, 6、非線形群ブロック／関数

非線形なシステム要素に関するブロック群です。

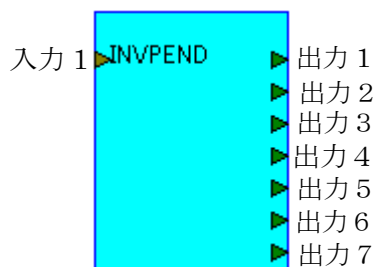
主に条件判定／論理演算を関数としてまとめたものです。特に他のブロックと組み合わせ、スイッチング素子として用いることで、ブロック・ダイアグラム、インタープリタ処理及びコンパイラによるC言語モジュールなどの処理の流れをより確実なものにすることが可能です。



非線形群（ライブラリ表示）

---

6 0 0 0 1 : I N V P E N D (倒立振り子:スコープ表示)



#### 動作

倒立振り子の関数ブロックです。

左右に自由に運動する台車の上に、高さRの棒振り子（棒の一端をフリー・ジョイントで台車に接続したもの）を1本立てます。

その棒振り子に対し、倒立振り子（棒振り子が倒れる方向を検出し、逆方向に台車を加速することで、棒振り子が倒れることを防ぎ、連続的に棒振り子を立たせる機構）のシミュレーションを実行します。

（棒振り子には、重力と台車の運動方向と同軸方向以外の力は加わらないものとします。）

具体的な動作は次のようになります。

台車と棒振り子の質量を無視すれば、台車の加速度 $\alpha$ と棒振り子の角速度 $\omega$ との間に、次の式が成立します。

$$d\omega/dt = -(g/R) * \sin(\theta) + (\alpha/R) * \cos(\theta) \dots \textcircled{1}$$
  $g = 9.80665$ （重力加速度）  
 $\textcircled{1}$ 式にて棒振り子の角加速度 $d\omega/dt$ を求め、時間 $t$ で積分して棒振り子の角速度 $\omega$ と振り子角度 $\theta$ を求めます。

同様に $\alpha$ を積分して、（各時点での）台車速度 $v$ と台車位置 $x$ を求めます。

台車位置 $X$ とその時点の振り子の角度 $\theta$ 、振り子半径（棒の高さ） $R$ より、棒の頂点（錘の位置）の $x$ 座標 $x_m$ 、 $y$ 座標 $y_m$ は、それぞれ、

$x_m = x + R * \cos(\theta) \dots \textcircled{2}$ 、

$y_m = R * \sin(\theta) \dots \textcircled{3}$  にて、求められます。

I N V P E N Dブロックでは、台車加速度 $\alpha$ を入力データとし、棒の高さ $R$ をプロパティ・データ（定数）として与え、その他の各値を順次計算して出力します。

算出した台車位置と棒振り子の頂点座標（錘位置）を元に、算出した倒立振り子の状態をリアルタイムにスコープ表示処理します。

#### 入力端子／データ

端子番号	端子名（入力信号名）	単位	データ型	備考
1	$\alpha$ : 台車加速度	[m/sec <sup>2</sup> ]	float 型	



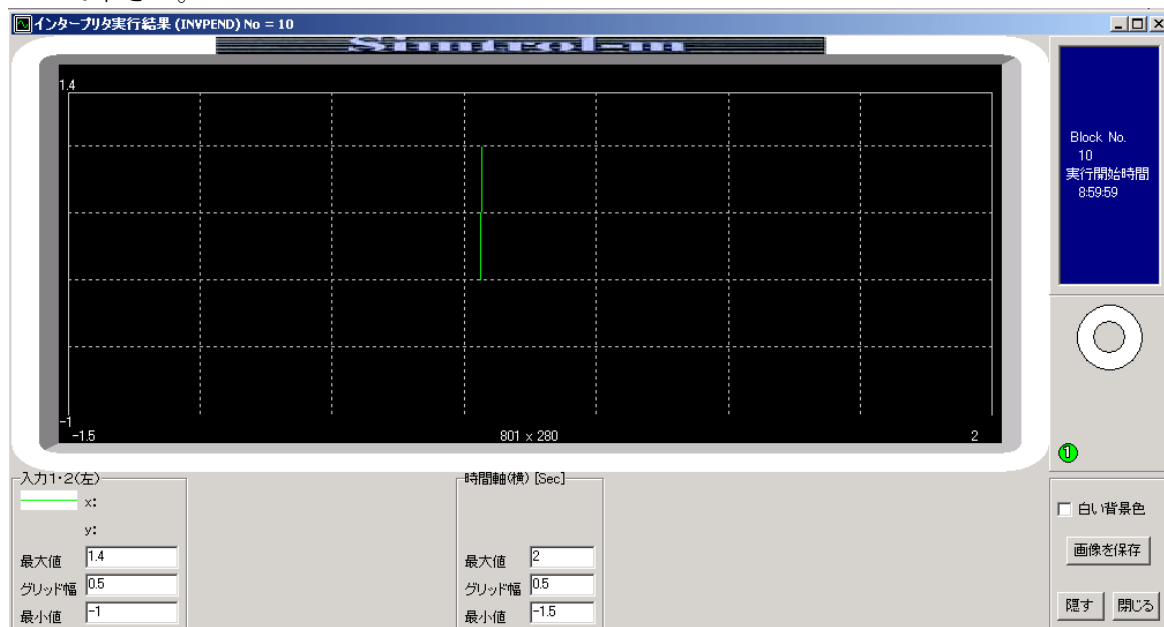
出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	$\theta$ : 振子角度	[rad]	float 型	
2	$\omega$ : 振子角速度	[rad/s]	float 型	
3	$d\omega/dt$ : 振子角加速度	[rad/s <sup>2</sup> ]	float 型	
4	x : 台車位置	[m]	float 型	
5	v : 台車速度	[m/s]	float 型	
6	xm : 錘位置 x 座標	[m]	float 型	振子頂点
7	ym : 錘位置 y 座標	[m]	float 型	振子頂点

プロパティ・データ

No.	プロパティ・データ名	単位	データ型	備考
1	xmin : x 軸最小値		float 型	スコープ設定
2	xmax : x 軸最大値		float 型	スコープ設定
3	xgrid : x 軸目盛り刻み		float 型	0 不可
4	ymin : y 軸最小値		float 型	スコープ設定
5	ymax : y 軸最大値		float 型	スコープ設定
6	ygrid : y 軸目盛り刻み		float 型	0 不可
7	M : 質量	[kg]	float 型	(未使用)
8	R : 振子半径 (高さ)	[m]	float 型	0 以下禁止
9	$\theta_s$ : $\theta$ 初期値	[rad]	float 型	

- ・ 振子半径値Rには、0 以下の値を設定しないで下さい。
- ・ 振子半径値Rに過大／過小な値を設定されますと、スコープ画面での棒振子の様態の観測が困難になります。
- ・ x 軸／y 軸ともに目盛り刻み値 (x g r i d / y g r i d) には、0 (ゼロ) を設定しないで下さい。

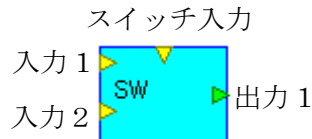


(実行表示例：スコープ画面)

対応関数 : void fn\_invpndvoid fn\_invpnd (float alfa, float \*propaty, float \*th,  
float \*omega,float \*domega,float \*x,float \*v,float \*xm,float \*ym, int \*err ) ;

---

6 0 0 0 2 : SW (切替スイッチ)



動作

出力の切替スイッチ。

sw : スイッチ入力。

sw が 0 のとき出力値=入力 1。

sw が 0 以外のとき出力値=入力 2。

入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	input1 : 入力 1	—	float 型	
2	input2 : 入力 2	—	float 型	
3	sw: スイッチ入力	—	float 型	

出力端子／データ

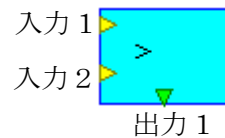
端子番号	端子名 (default)	単位	データ型	備考
1	output : 出力	—	float 型	

プロパティ・データ:なし。

対応関数

```
void fn_sw( float input1, float input2, float sw, float *output, int *err ) ;
```

6 0 0 0 3 : > (比較>)



動作

入力 1 > 入力 2 のとき出力値 = 1。

入力 1 ≤ 入力 2 のとき出力値 = 0。

入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	input1 : 入力 1	—	float 型	
2	input2 : 入力 2	—	float 型	

出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	output : 出力	—	float 型	1, 0 いずれかの値のみ

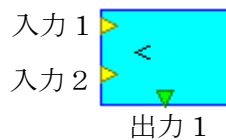
プロパティ・データ:なし。

対応関数

```
void fn_cg( float input1, float input2, float *output, int *err ) ;
```

---

6 0 0 0 4 : < (比較<)



動作

入力 1 < 入力 2 のとき出力値 = 1。

入力 1 ≥ 入力 2 のとき出力値 = 0。

入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	input1 : 入力 1	—	float 型	
2	input2 : 入力 2	—	float 型	

出力端子／データ

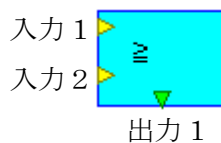
端子番号	端子名 (default)	単位	データ型	備考
1	output : 出力	—	float 型	1, 0 いずれかの値のみ

プロパティ・データ:なし。

対応関数

```
void fn_cl( float input1, float input2, float *output, int *err );
```

6 0 0 0 5 : ≥ (比較≥)



動作

入力 1 ≥ 入力 2 のとき出力値 = 1。

入力 1 < 入力 2 のとき出力値 = 0。

入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	input1 : 入力 1	—	float 型	
2	input2 : 入力 2	—	float 型	

出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	output : 出力	—	float 型	1, 0 いずれかの値のみ

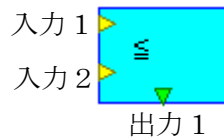
プロパティ・データ:なし。

対応関数

```
void fn_cge( float input1, float input2, float *output, int *err );
```

---

6 0 0 0 6 :  $\leq$  (比較 $\leq$ )



動作

入力 1  $\leq$  入力 2 のとき出力値 = 1。

入力 1 > 入力 2 のとき出力値 = 0。

入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	input1 : 入力 1	—	float 型	
2	input2 : 入力 2	—	float 型	

出力端子／データ

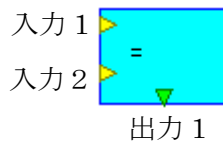
端子番号	端子名 (default)	単位	データ型	備考
1	output : 出力	—	float 型	1, 0 いずれかの値のみ

プロパティ・データ:なし。

対応関数

```
void fn_cle( float input1, float input2, float *output, int *err ) ;
```

6 0 0 0 7 : = (比較=)



動作

入力 1 = 入力 2 のとき出力値 = 1。

入力 1  $\neq$  入力 2 のとき出力値 = 0。

入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	input1 : 入力 1	—	float 型	
2	input2 : 入力 2	—	float 型	

出力端子／データ

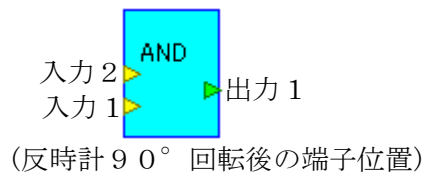
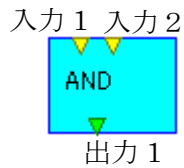
端子番号	端子名 (default)	単位	データ型	備考
1	output : 出力	—	float 型	1, 0 いずれかの値のみ

プロパティ・データ:なし。

対応関数

```
void fn_ce( float input1, float input2, float *output, int *err ) ;
```

---

**6 0 0 0 8 : AND (論理積演算)****動作**

- ・入力 1 / 入力 2 の値のどちらか一つでも 0 (ゼロ値) なら、出力値 = 0。
- ・入力 1 / 入力 2 の値がどちらも 1 なら、出力値 = 1。

**入力端子／データ**

端子番号	端子名 (default)	単位	データ型	備考
1	input1 : 入力 1	—	float 型	1, 0 いずれかの値のみ
2	input2 : 入力 2	—	float 型	1, 0 いずれかの値のみ

**出力端子／データ**

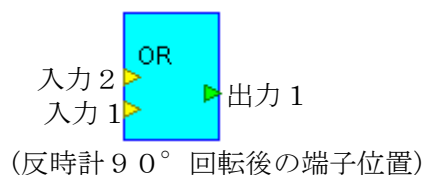
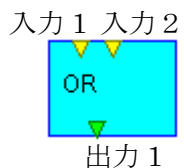
端子番号	端子名 (default)	単位	データ型	備考
1	output : 出力	—	float 型	1, 0 いずれかの値のみ

プロパティ・データ: なし。

**対応関数**

```
void fn_and( float input1, float input2, float *output, int *err ) ;
```

---

**6 0 0 0 9 : OR (論理和演算)****動作**

- ・入力 1 / 入力 2 の値がともに 0 (ゼロ値) のとき、出力値 = 0。
- ・入力 1 / 入力 2 の値のどちらか一つでも 1 なら、出力値 = 1。

**入力端子／データ**

端子番号	端子名 (default)	単位	データ型	備考
1	input1 : 入力 1	—	float 型	1, 0 いずれかの値のみ
2	input2 : 入力 2	—	float 型	1, 0 いずれかの値のみ

**出力端子／データ**

端子番号	端子名 (default)	単位	データ型	備考
1	output : 出力	—	float 型	1, 0 いずれかの値のみ

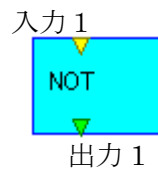
プロパティ・データ: なし。

**対応関数**

```
void fn_or( float input1, float input2, float *output, int *err ) ;
```

---

6 0 0 1 0 : NOT (否定)



動作

- ・入力値 = 0 のとき、出力値 = 1。
- ・入力値 = 1 のとき、出力値 = 0。

入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	input : 入力	—	float 型	1, 0 いずれかの値のみ

出力端子／データ

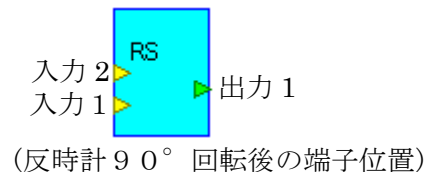
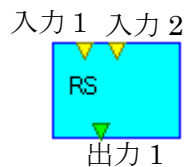
端子番号	端子名 (default)	単位	データ型	備考
1	output : 出力	—	float 型	1, 0 いずれかの値のみ

プロパティ・データ : なし。

対応関数

void fn\_not( float input1, float input2, float \*output, int \*err ) ;

6 0 0 1 1 : RS (RSフリップフロップ)



動作

RSフリップ・フロップ。

入力 1 : set 値／入力 2 : reset 値として、下表の動作を実行します。

(動作表)

set 値	1	1	0	0
reset 値	1	0	1	0
出力値	0	1	0	出力値を保持

入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	set : セット	—	float 型	1, 0 いずれかの値のみ
2	reset : リセット	—	float 型	1, 0 いずれかの値のみ

出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	output : 出力	—	float 型	1, 0 いずれかの値のみ

プロパティ・データ : なし。

---

対応関数

```
void fn_rs( float set, float reset, float *output, int *err );
```

6 0 0 1 2 : T I M ( オンディレータイマー )
----------------------------------



動作

- ・ (入力値 = 1) の状態が、タイマー設定時間以上経過したら出力値 = 1。
- ・ (入力値 = 0) のとき、出力値 = 0 とします。

入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	input : 入力	—	float 型	1, 0 いずれかの値のみ

出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	output : 出力	—	float 型	1, 0 いずれかの値のみ

プロパティ・データ

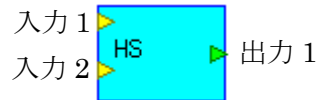
No.	プロパティ・データ名	単位	データ型	備考
1	Tset : タイマー設定時間	[sec]	float 型	

対応関数

```
void fn_tim( float input, float *propaty, float *work, float *output, int &err );
```

---

6 0 0 1 3 : H S (ハイ・セレクト)



動作

入力値のハイ・セレクト (H i g h ・ S e l e c t) 動作です。

- ・ 入力値 1  $\geq$  入力値 2 のとき、出力値 = 入力 1。
- ・ 入力値 1 < 入力値 2 のとき、出力値 = 入力 2。

注意事項) 入力値の単位を合わせて下さい。

入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	input1 : 入力 1	—	float 型	
2	input2 : 入力 2	—	float 型	

出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	output : 出力	—	float 型	

プロパティ・データ:なし。

対応関数

```
void fn_hs( float input1, float input2, float *output, int *err );
```

6 0 0 1 4 : L S (ロー・セレクト)



動作

入力値のロー・セレクト (L o w ・ S e l e c t) 動作です。

- ・ 入力値 1  $\leq$  入力値 2 のとき、出力値 = 入力 1。
- ・ 入力値 1 > 入力値 2 のとき、出力値 = 入力 2。

注意事項) 入力値の単位を合わせて下さい。

入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	input1 : 入力 1	—	float 型	
2	input2 : 入力 2	—	float 型	

出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	output : 出力	—	float 型	

プロパティ・データ:なし。

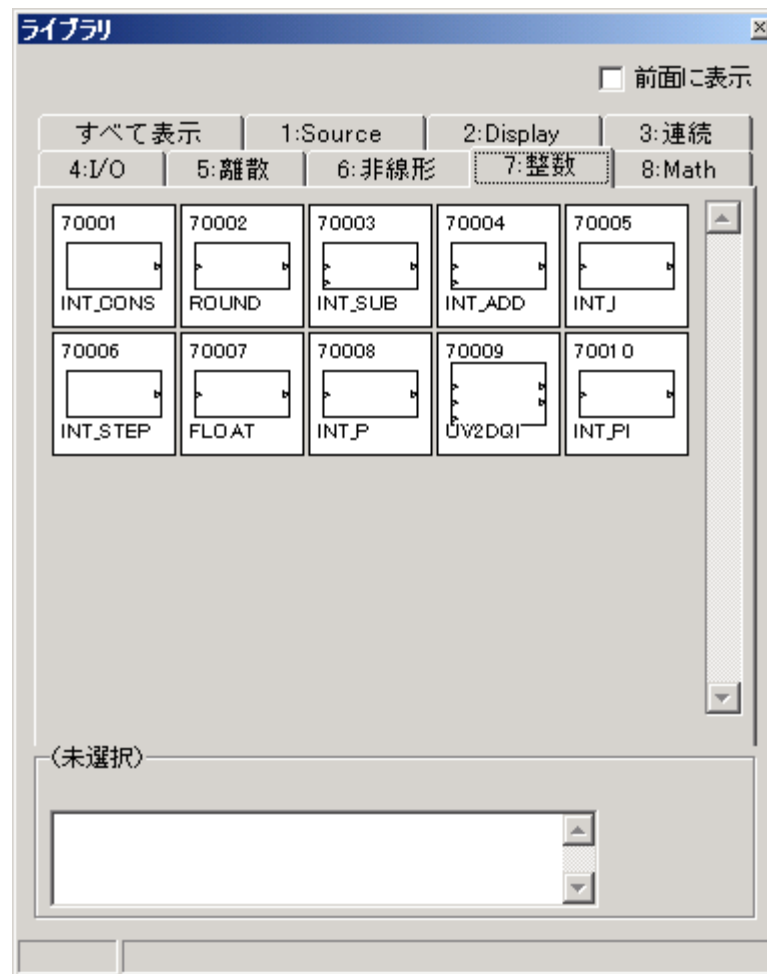
対応関数

```
fn_ls( float input1, float input2, float *output, int *err );
```



## 2. 7、整数群ブロック／関数

代表的な関数ブロックを選択して、入出力データ及び対応関数の内部演算を整数化することで、動作速度の向上及びメモリの節約を狙った関数ブロック群です。



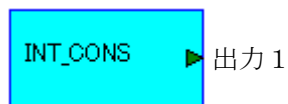
整数群（ライブラリ表示）

### 注意事項)

- ブロック・ダイアグラム上で整数型（`int`型）の出力端子は浮動小数点型（`float`型）の入力端子に接続できないので、`float`型のデータ入力により動作するブロック、特に`Display`群のブロック類（スコープ類／数値表示など）は、直接接続して使用することはできません。  
70007の`FLOAT`ブロックにより、データを浮動小数点型（`float`型）に変換の上、ご利用ください。
- 同様に、この群のブロックの入力端子（`int`型）と、他の`float`型の出力端子を直接、接続することはできません。

---

7 0 0 0 1 : I N T \_ C O N S (整数定数信号器)



#### 動作

定数 constant (実数値: float 型) を初期化時に整数に変換し、常時出力します。  
変換時には四捨五入して整数型(int 型)に変換します。

入力端子／データ：なし。

出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	output : 出力	—	int 型	

プロパティ・データ

No.	プロパティ・データ名	単位	データ型	備考
1	Constant : 定数	—	float 型	

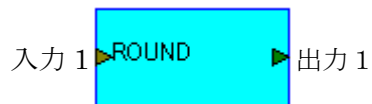
- ・プロパティの定数値 (実数値) と出力値 (整数値) の間に、数値の丸め誤差が生じます。

#### 対応関数

```
void fn_int_const (float *propaty, int *int_work, int *output, int *err);
```

---

7 0 0 0 2 : R O U N D (四捨五入演算)



動作

入力値(float 型)を四捨五入して、整数値(int 型)に変換して出力します。

入力端子／データ

端子番号	端子名 (入力信号名)	単位	データ型	備考
1	input : 浮動小数点入力	—	float 型	

出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	output : 整数出力	—	int 型	

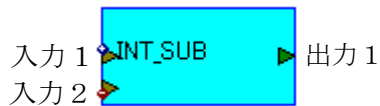
プロパティ・データ : なし。

- ・ float 型と int 型とでは、扱える値の範囲が異なります。ご注意ください。

対応関数

```
void fn_round(float input, int *output, int *err);
```

7 0 0 0 3 : I N T \_ \_ S U B (整数減算)



動作

出力 = 入力 1 - 入力 2。(整数型減算)

入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	Input1 : 整数入力 1	—	int 型	
2	Input2 : 整数入力 2	—	int 型	

出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	output : 整数出力	—	int 型	

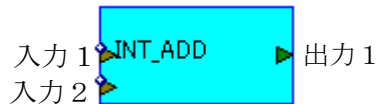
プロパティ・データ : なし。

対応関数

```
void fn_int_sub (int xin1, int xin2, int *xout ,int *err);
```

---

7 0 0 0 4 : I N T \_ A D D (整数加算)



動作

出力=入力 1 + 入力 2。(整数型加算)

入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	input1 : 整数入力 1	—	int 型	
2	input2 : 整数入力 2	—	int 型	

出力端子／データ

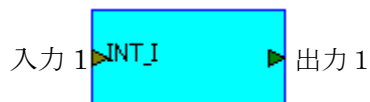
端子番号	端子名 (default)	単位	データ型	備考
1	output : 整数出力	—	int 型	

プロパティ・データ：なし。

対応関数

```
void fn_int_add (int input1, int input2, int *output ,int *err);
```

7 0 0 0 5 : I N T \_ I (整数型積分制御器)



動作

積分時間  $T_i$  を指定して、積分動作を実施する。(積分時間  $T_i$  はプロパティ・データで指定。)

入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	input : 整数入力	—	int 型	

出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	output : 整数積算出力	—	int 型	

プロパティ・データ

N o .	プロパティ・データ名	単位	データ型	備考
1	$T_i$ : 積分時間	[sec]	float 型	0 禁止
2	xmin : 出力最小値	—	float 型	
3	xmax : 出力最大値	—	float 型	

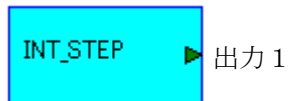
- ・ 積分時間  $T_i$  : 積分動作の出力が偏差 (設定値－測定値) と同じ値に達するまでの時間。値が小さいほど、積分動作が強く動作します。
- ・ 積分時間  $T_i$  の値には 0 を設定しないで下さい。

対応関数

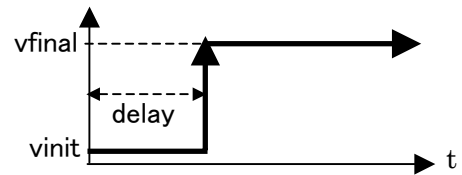
```
void fn_int_ctrl_i(int input1, float *propaty, int *int_work, int *output, int *err);
```

---

---

**7 0 0 0 6 : I N T \_ S T E P (整数ステップ信号器)**

動作  
出力の初期値を *vinit* とし、  
*delay* 秒後に、*vfinal* に出力を切り替えます。



入力端子／データ：なし

出力端子／データ

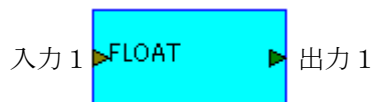
端子番号	端子名 (default)	単位	データ型	備考
1	output : 整数出力	—	int 型	

プロパティ・データ

N o .	プロパティ・データ名	単位	データ型	備考
1	delay : 切り替え時間	[sec]	float 型	
2	vinit : 出力初期値	—	float 型	
3	vfinal: 切り替え後の出力値	—	float 型	

対応関数

```
void fn_int_step (float *propaty, int *int_work, int *output, int *err);
```

**7 0 0 0 7 : F L O A T (浮動小数点型変換)**

動作  
整数型 (int 型) を浮動小数点型 (float 型) に変換します。

入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	input : 整数入力	—	int 型	

出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	output : 浮動小数点出力	—	float 型	

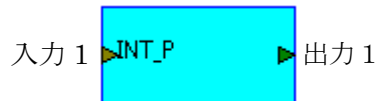
プロパティ・データ：なし。

対応関数

```
void fn_float(int input, float *output, int *err);
```

---

7 0 0 0 8 : I N T \_ P (整数型比例動作)



動作

整数型の比例動作(P)制御器。

入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	xin : 入力	—	int 型	

出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	xout : 出力	—	int 型	

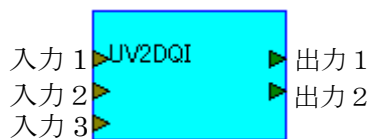
プロパティ・データ

N o .	プロパティ・データ名	単位	データ型	備考
1	kp : 比例定数	—	float 型	0 禁止

対応関数

void fn\_int\_p(int xin,float \*propaty, int \*work,int \*xout,int \*err);

7 0 0 0 9 : U V 2 D Q I (整数型3相→d q変換)



動作

3相交流(U相・V相)→d-q座標変換(3相側の入力は2相のみを使用:整数型)

入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	U : 3相U相	—	int 型	
2	V : 3相V相	—	int 型	
3	$\theta$ : 位相	(0-4000/0-2 $\pi$ [rad])換算	int 型	0-4000の整数値で指定。

出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	d : 出力d相	—	int 型	
2	q : 出力q相	—	int 型	

プロパティ・データ : なし。

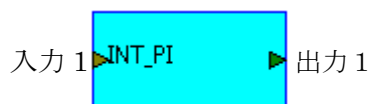
対応関数

void fn\_uv2dqi(int u, int v, int th, int \*d, int \*q, int \*err);

- ・入力側の交流正弦波信号の型を i n t 型にして下さい。

---

7 0 0 1 0 : I N T \_ P I (整数型比例積分要素)



動作

Ti : 積分時間[sec]/Kp : 比例定数などを指定し、整数型の P I 制御を実施します。

入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	xin : 入力	—	int 型	±10,000 まで

出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	xout : 出力	—	int 型	

プロパティ・データ

N o .	プロパティ・データ名	単位	データ型	備考
1	Ti : 積分時間	[sec]	float 型	0 禁止
2	Xmin : 出力最小値	—	float 型	±10,000 まで
3	Xmax : 出力最大値	—	float 型	±10,000 まで
4	Kp : 比例定数	—	float 型	

T i : 積分時間 : 積分動作の出力が偏差 (設定値－測定値) と等しい値に到達するまでの時間。  
この値が小さいほど、積分動作は強く動作します。

注意 1 : Ti は[インタープリタの実行刻み時間 dt] \* 100,000 よりも小さい値を設定する  
して下さい。Ti > (dt\*100000) の場合、積分動作は保障されません。  
また、0 を指定しないで下さい。

注意 2 : 入力(xin)の値は±10,000 でリミットされます。

注意 3 : xmin,xmax は±10,000 の範囲内で設定し、xmin < xmax であるようにして下さい。

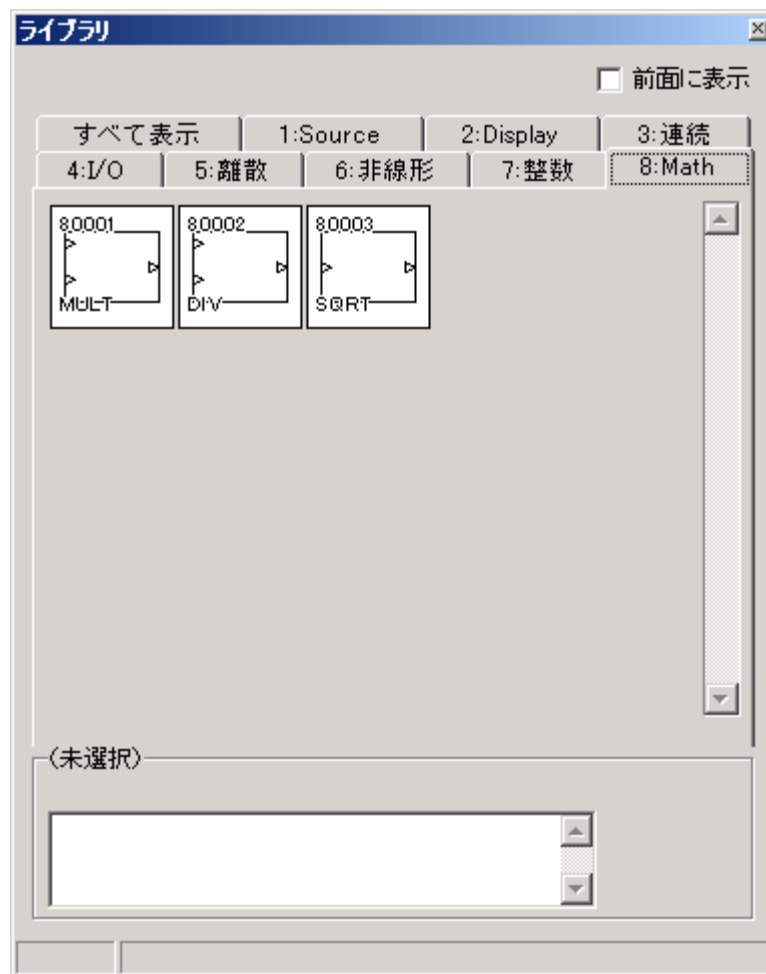
対応関数

```
void fn_int_pi(int xin, float *propaty, int *int_work, int *xout, int *err);
```

---

## 2, 8、Math群ブロック／関数

特に数学的に高度な演算処理を実行する関数ブロック群です。



Math群（ライブラリ表示）



---

8 0 0 0 1 : M U L T (掛け算器)
----------------------------



#### 動作

入力された値同士を乗算して、出力します。

出力 = 入力 1 \* 入力 2。

#### 入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	input1 : 入力 1	—	float 型	
2	input2 : 入力 2	—	float 型	

#### 出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	output : 出力	—	float 型	

プロパティ・データ:なし。

#### 対応関数

void fn\_mult( float input1, float input2, float \*output, int \*err ) ;

8 0 0 0 2 : D I V (割り算器)
--------------------------



#### 動作

出力値 = 入力値 1 / 入力値 2

入力 2 が 0 : ゼロ値のとき、出力は前回値を保持します。

#### 入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	input1 : 入力 1	—	float 型	
2	input2 : 入力 2	—	float 型	

#### 出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	output : 出力	—	float 型	

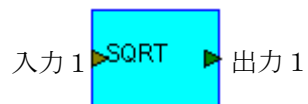
プロパティ・データ:なし。

#### 対応関数

void fn\_div( float input1, float input2, float \*output, int \*err ) ;

---

8 0 0 0 3 : S Q R T (平方根)



#### 動作

入力値の平方根（1/2 乗）を求め、出力します。

入力値が負（－）の値の場合、出力は前回値を保持します。

#### 入力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	input1 : 入力 1	－	float 型	負数（－）不可

#### 出力端子／データ

端子番号	端子名 (default)	単位	データ型	備考
1	output : 出力	－	float 型	

プロパティ・データ:なし。

#### 対応関数

```
void fn_sqrt( float input1, float input2, float *output, int *err ) ;
```

---

\* S i m t r o l - m のライブラリに搭載されている関数を、予め定められた範囲を超えて使用することを禁止致します。

\* 「S i m t r o l」は株式会社 昭和電業社が商標登録を申請中です。

以上

株式会社 昭 和 電 業 社

〒299-0111 千葉県市原市姉崎 745-2

TEL : 0436-61-4616

HP : <http://www.k-sd.co.jp/>

Mail : [kentac@k-sd.co.jp](mailto:kentac@k-sd.co.jp)

---

## 付録

### －ブロック（関数）名索引（1／4）－

#### ●記号

=（比較：＝）（非線形群：60007）	87
<（比較：<）（非線形群：60004）	86
>（比較：>）（非線形群：60003）	85
≤（比較：≤）（非線形群：60006）	87
≥（比較：≥）（非線形群：60005）	86
2／3入力加算減算器（連続群：31004）	67
3 P H A S E（平衡3相交流信号発生器）（S o u r c e群：10008）	16

#### ●A

A B 2 D Q（ $\alpha\beta \rightarrow dq$ 変換）（連続群：30028）	57
A B 2 U V W（ $\alpha\beta \rightarrow 3$ 相変換）（連続群：30030）	59
A D C（A－Dコンバータ）（連続群：30012）	41
A D C 1 3 6 0 0（1 3 6 0 0用A－Dコンバータ）（I／O群：40001）	70
A D D（加算器）（連続群：30005）	34
A N D（論理積演算）（非線形群：60008）	88

#### ●B

B A R 1（1入力型バググラフ）（D i s p l a y群：20002）	22
B L D C C 1 3 6 0 0 （1 3 6 0 0用ブラシレスDCモータ 電流制限付）（I／O群：40015）	80
B L D C I 1 3 6 0 0 （1 3 6 0 0用整数型ブラシレスDCモータ）（I／O群：40006）	75
B L D C M（ブラシレスDCモータ：モータパラメータ）（連続群：30033）	62
B L D C M 1（ブラシレスDCモータ：F l u x値）（連続群：30034）	63
B L D C M 1 3 6 0 0 （1 3 6 0 0用ブラシレスDCモータ）（I／O群：40005）	74

#### ●C

C O N S T（定数信号器）（S o u r c e群：10002）	10
C T R L _ D（D制御器）（連続群：30021）	50
C T R L _ I（I制御器）（連続群：30020）	49
C T R L _ P（P制御器）（連続群：30019）	48

#### ●D

D A C（D－Aコンバータ）（連続群：30013）	42
D A C 1 3 6 0 0（1 3 6 0 0用D－Aコンバータ）（I／O群：40002）	71
D C C T R L （ブラシレスDCモータの非干渉化：モータパラメータ）（連続群：30035）	64
D C C T R L 1 （ブラシレスDCモータの非干渉化：F l u x値）（連続群：30036）	65

---

－ブロック（関数）名索引（2／）－

DCM（DCモータ）（連続群：30022）	51
DIV（割り算器）（Math群：80002）	100
DQ2AB（ $dq \rightarrow \alpha\beta$ 変換）（連続群：30029）	58
DQ2UVW（ $dq \rightarrow 3$ 相変換）（連続群：30027）	56

●E

ENC13600（13600用エンコーダ①）（I/O群：40004）	73
ENC213600（13600用エンコーダ②）（I/O群：40008）	78
EULER（一次遅れ要素：オイラー法）（連続群：30006）	35

●F

FLOAT（浮動小数点型変換）（整数群：70007）	96
----------------------------	----

●G

GAIN（増幅器／定数乗算器）（連続群：30002, 31002）	31, 66
-----------------------------------	--------

●H

HS（ハイ・セレクト）（非線形群：60013）	91
-------------------------	----

●I

INT__ADD（整数加算）（整数群：70004）	95
INT__CONST（整数定数信号器）（整数群：70001）	93
INT__I（整数型積分制御器）（整数群：70005）	95
INT__P（整数型比例動作）（整数群：70008）	97
INT__PI（整数型比例積分動作）（整数群：70010）	98
INT__STEP（整数ステップ信号器）（整数群：70006）	96
INT__SUB（整数減算）（整数群：70003）	94
INVPEND（倒立振子：スコープ表示）（非線形群：60001）	83
I-PD （I-PD型制御器：測定値比例微分方式PID制御）（連続群：30017）	46

●L

LEADLAG（進み遅れ関数）（連続群：30009）	38
LIMIT（リミット）（連続群：30011）	40
LS（ロー・セレクト）（非線形群：60014）	91

●M

MAT22（マトリクス演算）（連続群：30014）	43
MAT23（マトリクス演算）（連続群：30023）	52
MAT32（マトリクス演算）（連続群：30024）	53
MULT（掛け算器）（Math群：80001）	100

---

－ブロック（関数）名索引（3／）－

●N

NOISE（ノイズ信号発生器）（S o u r t h群：10006）	14
NOT（論理否定演算）（非線形群：60010）	89
NUM1（1入力型数値表示）（D i s p l a y群：20003）	24

●O

OPWAVE（任意波形発生器）（S o u r t h群：10009）	17
OR（論理和演算）（非線形群：60009）	88

●P

PI（PI制御器）（連続群：30018）	47
PI-D（PI-D型制御器：測定値微分方式PI D制御）（連続群：30016）	45
PID（PID制御器）（連続群：30010）	39
PIDARW （アンチ・リセットwindアップPID制御器）（連続群：30032）	61
PWM13600（13600用PWM）（I／O群：40003）	72

●R

RAMP（ランプ信号発生器）（S o u r c e群：10007）	15
RK2（一次遅れ要素：ルンゲ・クッタ2次法）（連続群：30007）	36
RK4（一次遅れ要素：ルンゲ・クッタ4次法）（連続群：30003）	32
RK42（二次遅れ要素：ルンゲ・クッタ4次法）（連続群：30008）	37
RMAT22（逆マトリクス演算）（連続群：30015）	44
ROUND（四捨五入演算）（整数群：70002）	94
RS（RSフリップフロップ）（非線形群：60011）	89

●S

SCOPE1（1入力型オシロスコープ）（D i s p l a y群：20001）	20
SCOPE2（2入力型オシロスコープ）（D i s p l a y群：20004）	25
SCOPE4（4入力型オシロスコープ）（D i s p l a y群：20005）	27
SIN（正弦波発生器）（S o u r c e群：10003）	11
SINGEN（S o u r c e群：10010）	18
SQRT（平方根）（M a t h群：80003）	101
SQUARE（方形波発生器）（S o u r c e群：10004）	12
SQW13600（13600用120度矩形波駆動出力）（I／O群：40007）	77
SQWL13600 （13600用120度矩形波駆動出力 遅れ制御付）（I／O群：40009）	79
STEP（ステップ信号発生器）（S o u r c e群：10001）	10
SUB（減算器）（連続群：30001）	30
SUM4（4入力加算減算器）（連続群：30004）	33
SW（切替スイッチ）（非線形群：60002）	85

---

－ブロック（関数）名索引（4／4）－

●T-----

T I M（オンディレータイマー）（非線形群：60012）	90
T R I A N G（三角波発生器）（S o u r c e 群：10005）	13

●U-----

U V 2 D Q（2相（3相）→d q 変換）（連続群：30026）	55
U V 2 D Q I（整数型3相→d q 変換）（整数群：70009）	97
U V W 2 A B（3相→ $\alpha \beta$ 変換）（連続群：30031）	54
U V W 2 D Q（3相→d q 変換）（連続群：30025）	60