

ファンクションブロック/ ストラクチャードテキスト 導入ガイド

The screenshot shows the CX-Programmer software interface. The main window is titled 'sample33 - CX-Programmer'. The interface includes a menu bar, a toolbar, and a project tree on the left. The project tree shows a project named '新規プロジェクト' (New Project) with a sub-project '新規PLC1[CS1G-H] モニタモード' (New PLC1[CS1G-H] Monitor Mode). The main editor area is divided into two panes. The left pane shows a ladder logic diagram for a DVD thickness selection function block. The right pane shows a structured text editor with the following code:

```

(*アクチュエータ制御FBのインスタンスであるワーク移動 制御
ワーク移動(右方向動作入力, 左方向動作入力, LS右, LS左, ア
(*リミットスイッチ開閉回数カウンタ*)
IF 前サイクルLS = FALSE and LS右 = TRUE THEN
    LS_ON回数 = LS_ON回数+1;
END_IF;
前サイクルLS = LS右; (*LS右の状態を前サイクルLSにコピー*)
FOR I = 0 TO 49 DO
    Value[i] = SORT( SIN(Xvalue[i]) + COS(Yvalue[i]));
END_FOR;

```

The bottom of the interface shows a table with the following columns: 名称 (Name), アドレス (Address), データ型/表示形式 (Data Type/Display Format), FB用途 (FB Usage), 値 (Value), and コメント (Comment).

名称	アドレス	データ型/表示形式	FB用途	値	コメント
工程A DVD厚さ選別 DVD厚さ判定 入力3	H536	REAL (単精度浮動)	1: 入力	+0.0000	入力値3
工程A DVD厚さ選別 DVD厚さ判定 上限	H538	REAL (単精度浮動)	1: 入力	+0.0000	閾値の上限値
工程A DVD厚さ選別 DVD厚さ判定 下限	H540	REAL (単精度浮動)	1: 入力	+0.0000	閾値の下限値
工程A DVD厚さ選別 DVD厚さ判定 結果	H531	BOOL (On/Off 接点)	0: 出力	0	OK/NG判定フラグ

The bottom status bar shows 'F1キーを押すとヘルプが表示されます' (Press F1 key to display help), '新規PLC1 (シミュレータ接続中) - モニタモード' (New PLC1 (Simulator connection) - Monitor mode), '0.0 ms' (0.0 ms), and 'モニター同期 回路 2 (1, 0) - 90%' (Monitor sync Circuit 2 (1, 0) - 90%).

はじめに

- ・本製品をご使用になる前に、「CX-Programmerオペレーションマニュアル ファンクションブロック編/ストラクチャードテキスト編(SBCA-338)」、「CX-Programmerオペレーションマニュアル(SBCA-337)」の「はじめに」、「安全上のご注意」、「安全上の要点」を必ずお読みください。
- ・本書は、本製品の基本的な操作を説明しています。詳しい操作や機能については、本製品のヘルプやマニュアルを参照してください。
- ・PDFファイルで提供されているマニュアルを表示するには、アドビ システムズ社が配布しているAdobe Reader(無償)がパソコンにインストールされていることが必要です。
- ・PDFファイルで提供されているマニュアルは、本製品のインストール後に、デスクトップ上のスタートメニューから表示させることができます。
- ・本書で使用している画面は開発中のものであり、製品とは異なることがあります。また予告無く変更することがあります。
- ・本書に記載されている各社の商品名・サービス名・機能名およびロゴは、各社の商標または登録商標です。
- ・本書では、各社の商標または登録商標に(R)、TMの記号を表示していません。
- ・本書では、各社の製品名を省略して記載することがあります。
- ・本書は、Microsoft Corporation のガイドラインに従って画面写真を使用しています。

第1章 Smart FBライブラリ

1. ファンクションブロックとは	1-1
2. 簡単なファンクションブロック化の例	1-2
3. Smart FBライブラリの概要	1-3
3-1. Smart FBライブラリの効果	1-3
3-2. Smart FBライブラリの例	1-4
3-3. Smart FBライブラリの構成要素	1-6
3-4. Smart FBライブラリの品揃えと入手方法	1-7

第2章 Smart FBライブラリを使ってみましょう

1. 作成するプログラムの説明	2-1
1-1. プログラムの仕様	2-1
1-2. FB部品の仕様	2-1
1-3. 入力するプログラム	2-2
2. プロジェクトの新規作成とPLC機種／CPU形式の設定	2-3
3. メインウィンドウについて	2-4
4. 部品ファイルのインポート	2-5
5. プログラムの作成	2-6
5-1. a接点の入力	2-6
5-2. インスタンスの入力	2-7
5-3. パラメータの入力	2-7
6. プログラムのエラーチェック（コンパイル）	2-9
7. オンライン接続	2-10
8. 導通モニター1	2-11
9. 導通モニター2 パラメータの現在値変更	2-12
10. オンラインエディット	2-13

第3章 Smart FB部品をカスタマイズしてみましょう

1. 作成するプログラムの説明	3-1
1-1. 変更仕様	3-1
1-2. ライブラリの変更内容	3-1
2. FB部品のコピー	3-2
3. ファンクションブロック内変数の追加	3-3
4. ファンクションブロックラダーの変更	3-4
4-1. 接点の入力	3-4
4-2. 変数の使用状況の確認	3-5
5. PLCへの転送	3-6
6. 動作確認	3-6
7. ファンクションブロックのオンラインエディット	3-7

第4章 ST言語を使用してみましょう

1. ST言語とは	4-1
2. 作成するプログラムの説明	4-1
3. ST定義FBの作成	4-2
4. ファンクションブロック内変数の登録	4-3
5. STプログラムの入力	4-4
6. プログラムの作成とエラーチェック	4-5
7. プログラムの転送	4-6
8. 動作状態のモニタ	4-7
参考: IF-THEN-ELSE-END, IFを用いたSTプログラムの例	4-8
参考: 文字列型変数を用いたSTプログラムの例	4-9

第5章 応用編(FBによるプログラムの部品化)

1. 概要	5-1
2. プログラム開発の進め方	5-1
3. アプリケーション例	5-1
4. プログラム設計の進め方	5-2
5. FB定義の入力	5-9
シミュレーション機能でのステップ実行	5-14
6. FB定義のライブラリ化	5-20
7. メインプログラムの入力	5-21
8. メインプログラムのデバッグ	5-22
補足事項	
不要になったファンクションブロックの削除方法／ファンクションブロック用メモリの割り付けについて／便利な機能のご紹介	

第6章 応用編(ST言語でタスクプログラムを作成する)

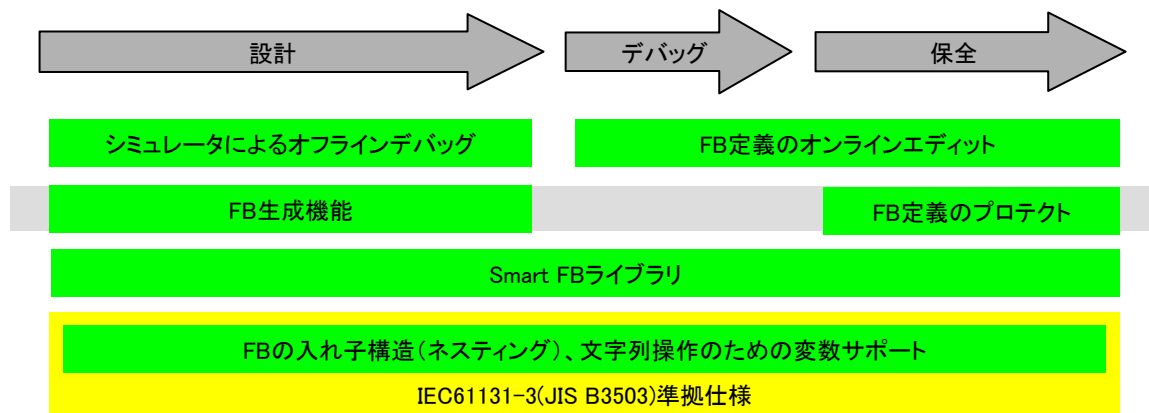
付録. ST(Structured Text)使用例	付録
----------------------------	----

はじめに

本書は、CX-Programmerで利用できる、Smart FBライブラリの使用方法およびファンクションブロック(FB)の作成方法のヒントをまとめたものです。

オムロンのFBの特徴

オムロンのFBはラダー言語とST(ストラクチャードテキスト)言語で表現でき、IEC 61131-3規格(JIS B3503)に準拠しながら、ユーザ様の装置の設計・デバッグ効率、保全性の向上を追及した機能を提供します。



PLCプログラム開発工程と機能の関係図

Smart FBライブラリ

Smart FBライブラリはオムロン製のPLC用ユニット、FAコンポのつながり親和性向上をねらった、ファンクションブロックの形式の部品集です。PLC用ユニット、FAコンポの基本的な機能を使うためのラダープログラムの作成が不要になりますので、ユーザ様は装置の機能をいかに実現するか、という本来の仕事に時間を費やすことができます。

FB定義のオンラインエディット

運転中にFB定義の内容を変更できるため、デバッグ時に短時間にFB定義を修正できます。また、24時間稼働の設備でも安心してFBを使用することができます。

入れ子構造(ネスティング)

オムロンのFBの入れ子構造(ネスティング)は単にプログラミングできるようにしているだけでなく、状況に応じた画面遷移や構造をツリー状で表示することで、わかりやすく・ストレスのない操作を行うことができます。

FB定義のプロテクト

プロジェクトファイル上に割り当てられたファンクションブロック定義にパスワードを設定することで目的に応じたプロテクトを行い、プログラムのノウハウの流出、不用意な変更、改ざん等に対する防止策を施すことができます。

シミュレータによるオフラインデバッグ

PLCプログラムの動作を机上で検証することができるため、プログラムの品質の前倒し確認が可能です。ラダー、ST共にパソコンアプリケーション感覚で実行可能です。

文字列操作のための変数サポート

ST言語で文字列データの操作を行う機能として、単に文字列型変数のサポートのみでなく、文字列データの入出力となる通信用の命令(関数)もあわせて強化しました。そのため、通信コマンド送信・受信のためのプログラム作成が簡単です。

FB生成機能

既存のPLCプログラムを再利用して簡単にFB化することが可能です。

第1章

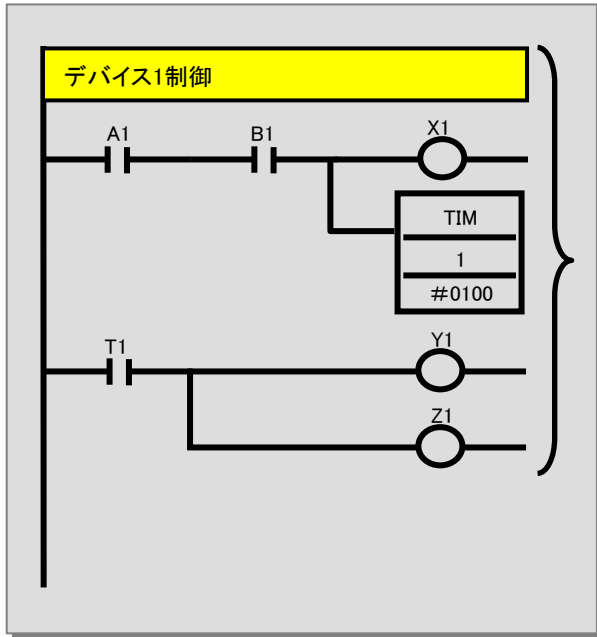
Smart FBライブラリ

Function Block

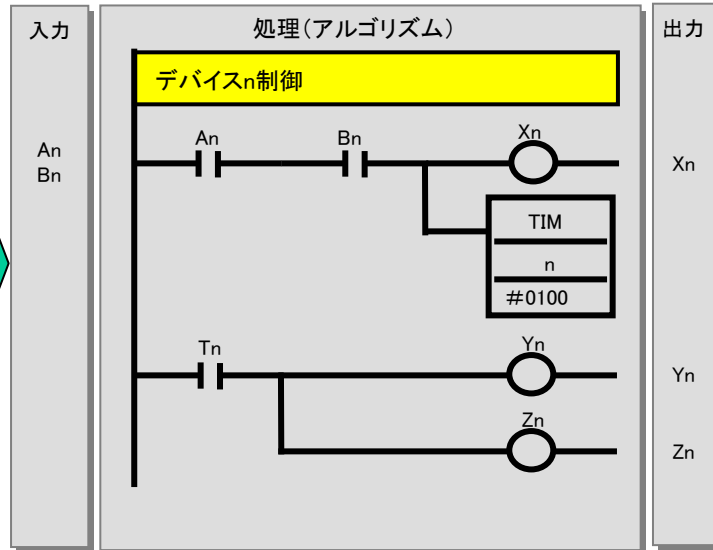
1. ファンクションブロックとは

「ファンクションブロック」とは、ある定型な処理（機能）を1つのかたまり（ブロック）としてあらかじめ定義しておき、ユーザは、それをプログラムに配置して、入出力を設定するだけで使用することができる、プログラム要素（テンプレート）です。標準的な複数回路をファンクションブロックとして作成/ファイル化しておけば、それを配置して、入出力にパラメータを設定するだけで、簡単に、複雑な回路を再利用することができます。

装置Aのラダー図の一部

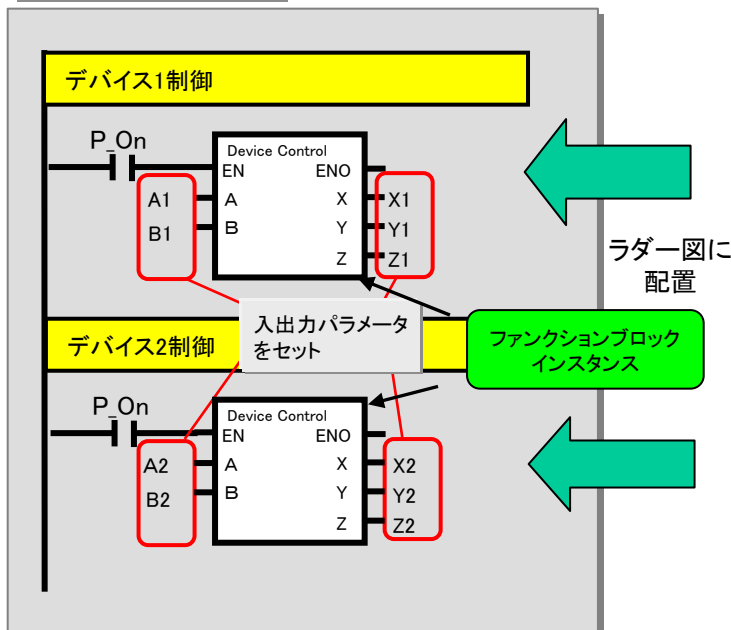


入力・出力を分けると・・・

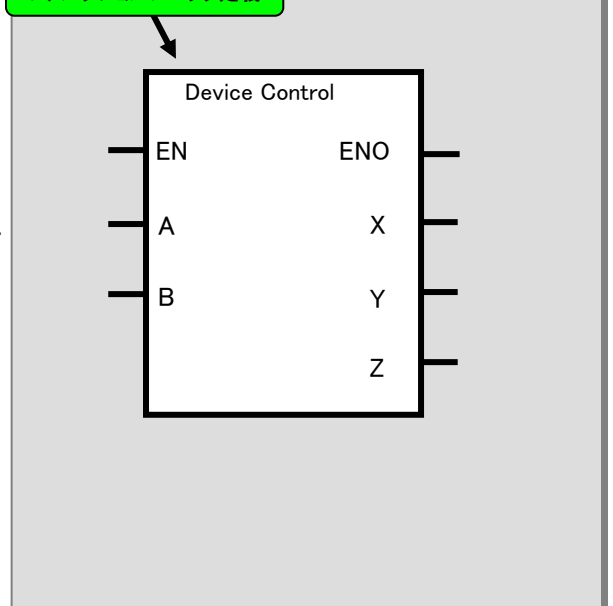


テンプレート化

装置A' のラダー図の一部



ファンクションブロック定義

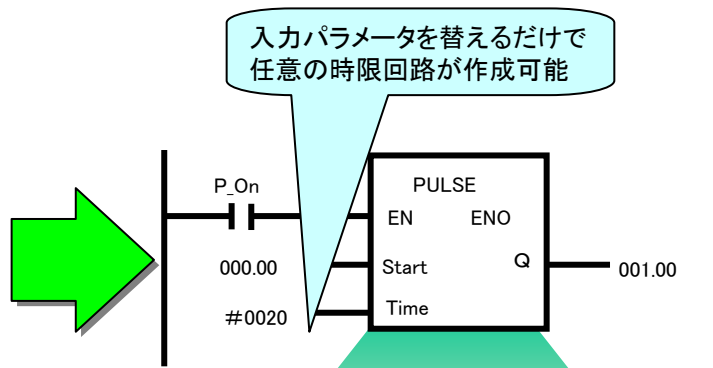
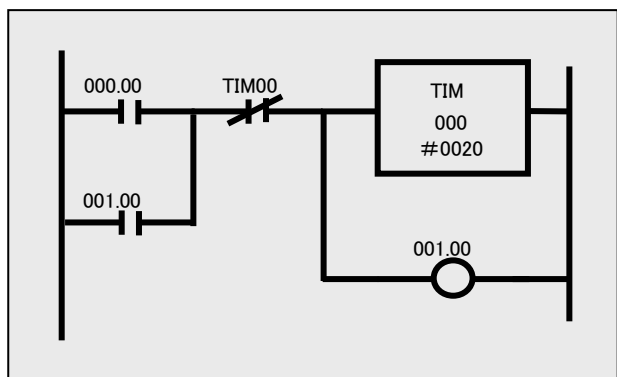


ファンクションブロック定義・・・ロジック（アルゴリズム）および入出力インタフェースを定義したものです。ファンクションブロック定義においては、アドレスは決まっています。
ファンクションブロックインスタンス・・・ファンクションブロック定義をラダープログラムで使用する際の呼び出し命令です。ファンクションブロックインスタンスをラダープログラムに配置した時点でファンクションブロック定義内で使用するアドレスが決まります。使用するファンクションブロックのメモリ（アドレス）は、ファンクションブロックインスタンス毎に別々に割り付けられて実行されるため、1つのファンクションブロック定義を複数のインスタンスで呼び出しても問題は発生しません。

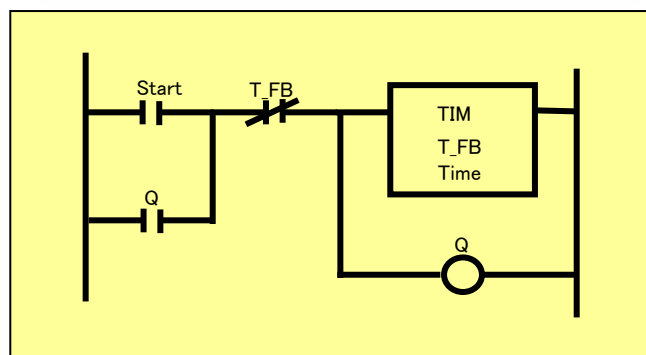
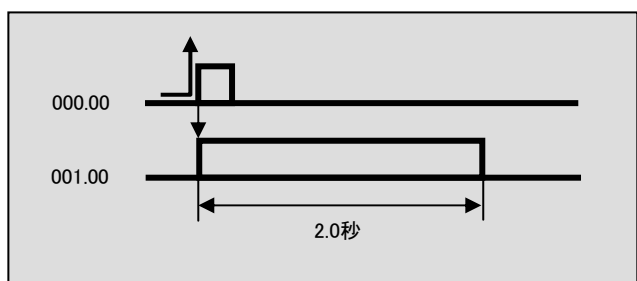
2. 簡単なファンクションブロック化の例

たとえば、時限回路をファンクションブロック化し、ラダー回路で利用すると以下ようになります。
ラダー回路では出力がOFFになるまでの時間を変更するにはタイマ命令の設定値を変更する必要があります。
ファンクションブロック化すると、変更したいパラメータを変更するだけで任意の時限回路とすることができます。

ラダー回路



タイムチャート



その他、既存のラダープログラムを元にファンクションブロックを生成する機能も準備しております。
詳細は「便利な機能のご紹介 既存のラダープログラムを元にFBを作成する機能」をご参照ください。

3. Smart FBライブラリの概要

Smart FBライブラリとは、オムロンがファンクションブロックの形態でユーザー様にご提供する、オムロン製のPLC用ユニット、FAコンポの機能をより簡単に PLCプログラム上で使用するための部品集です。

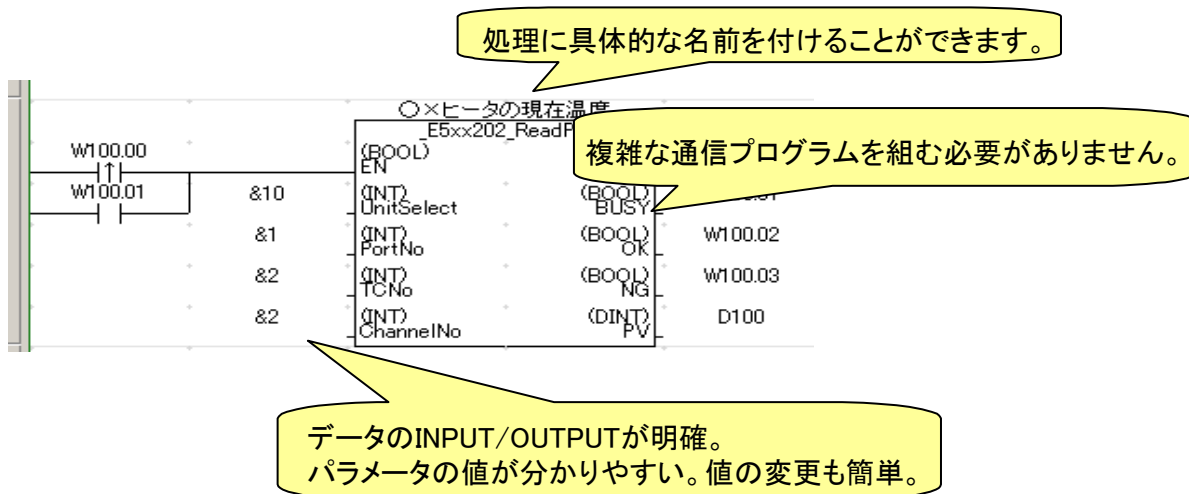
3-1. Smart FBライブラリの効果

Smart FBライブラリはオムロン製のPLC用ユニット、FAコンポのつなぎ親和性向上をねらった、ファンクションブロックの形式の部品集です。お使いいただくことで以下の効果があります。

- (1) PLC用ユニット、FAコンポの基本的な機能を使うためのラダープログラムの作成が不要。
ユーザー様は装置の機能をいかに実現するか、という本来の仕事に時間を費やすことができます。
- (2) 簡単に使える
目的の機能を動作させるためのファンクションブロックを読み込み、ラダープログラムにインスタンス(呼び出し命令)を入力し、入力・出力となるアドレス(パラメータ)をセットするだけで動作プログラムが完成します。
- (3) 動作確認が不要
オムロンで動作確認された部品を使用することができます。ユーザー様による PLC用ユニット、FAコンポを動作させるためのプログラムのデバッグが不要です。
- (4) わかりやすい
ファンクションブロックには機能を示す名前(ボディ名)、ユーザープログラムの該当箇所における名前(インスタンス名)がつきます。処理に具体的な名前をつけることができます。

インスタンス(ファンクションブロック呼び出し命令)はデータの入力・出力を意味するパラメータを持っています。一時受けのリレー、演算途中のテンポラリのアドレスは表示しませんので、パラメータの値がわかりやすくなっています。また、パラメータの変更も1箇所ですむため、デバッグ時の微調整なども簡単に行えます。

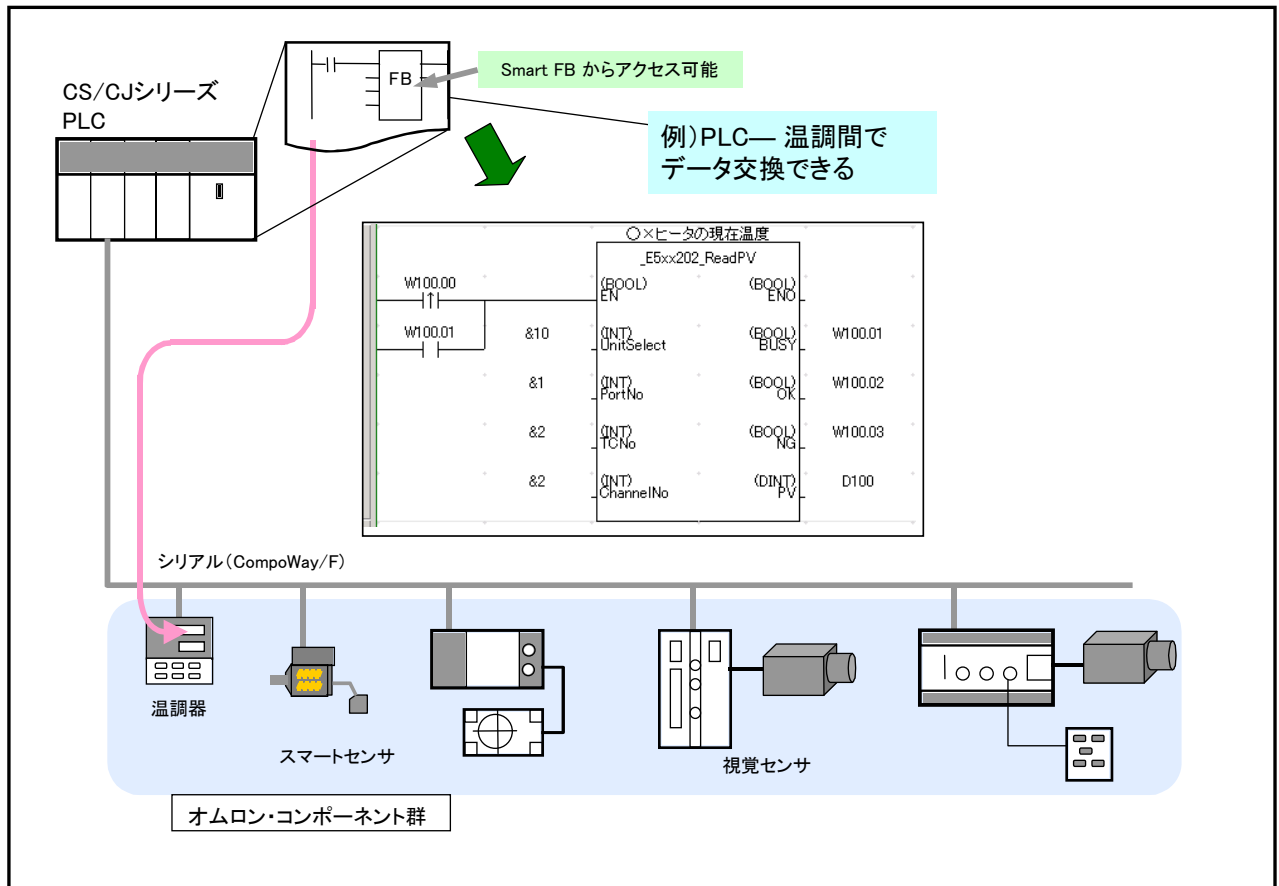
ラダープログラムでインスタンスを使う場合、ファンクションブロックの内部の処理は表示しないため、ラダープログラムをシンプルにすることができます。
- (5) 将来の拡張性
Smart FBライブラリは将来にわたってラダープログラムとファンクションブロック間のインタフェースを変更しません。将来PLC用ユニット、FAコンポをより高機能なものに変更する場合、新しいユニットに対応したファンクションブロックに差し替えるだけでユニットが動作するようになります。



3-2-1. Smart FBライブラリの例1

PLCラダープログラムから、オムロン製コンポーネントを、簡単に制御できます。

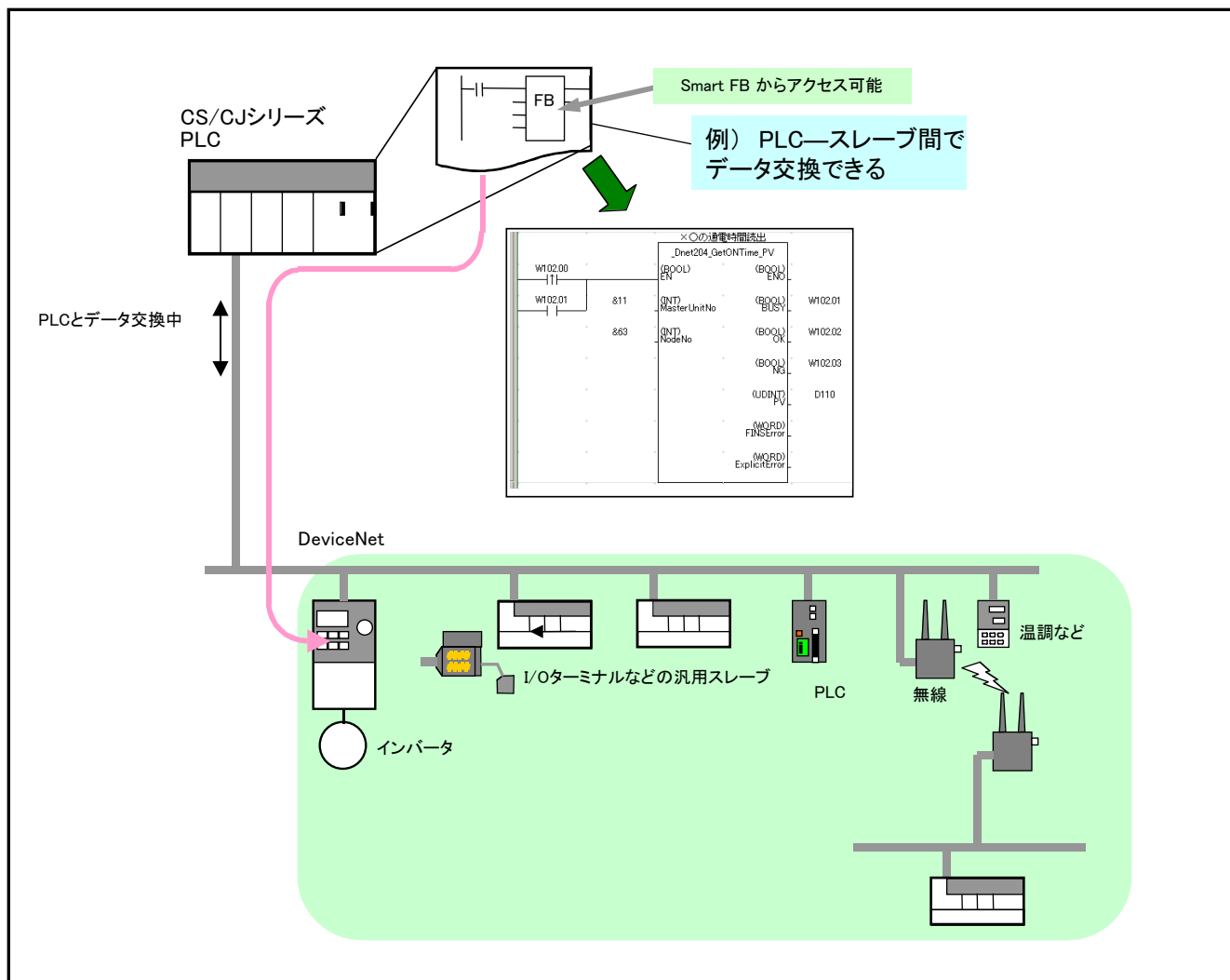
・RS-232C/485レベルでローコストの通信が可能



3-2-2. Smart FBライブラリの例2

・フィールドネットワークDeviceNetレベルで高機能の通信が可能です。

例)簡単に、PLCで、DeviceNetスレーブとデータ交換が可能。

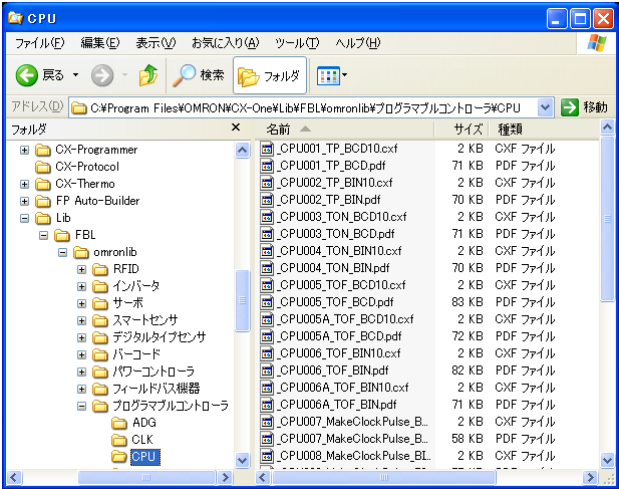


3-3. Smart FBライブラリの構成要素

Smart FBライブラリは以下の要素で構成されます。

3-3-1. 部品ファイル

部品ファイルとは、PLC用ユニット、FAコンポの機能毎に準備されたラダーファンクションブロックで作成されたファイルです。
拡張子は .CXF で、ラダー図で記述されています。
Smart FBライブラリは「_」(アンダーバー)で始まるファイル名となっています。
Smart FBライブラリをパソコンにインストールすると、PLC用ユニット、FAコンポの種類毎にフォルダで分類された部品ファイルがコピーされます。



3-3-2. ライブラリリファレンス

ライブラリリファレンスとは、部品ファイル毎に部品ファイルの動作仕様、入力・出力パラメータの仕様を記述したものです。ファイルフォーマットは PDF 形式です。
Smart FBライブラリを使用する場合、ライブラリリファレンスを参照して使用する部品ファイルの選定、入力・出力パラメータの設定、動作確認を行います。

E5xx 202	現在値読出 E5xx202_ReadPV
機能概要	指定した温度調節器の指定チャンネルの現在値を読み出します。
シンボル	起動トリガ 処理中フラグ シリアルポート番号 温度調節器ユニット番号 チャンネル番号 (BO0L) EN (INT) UnitSelect (INT) UnitNo (INT) PortNo (INT) YCNr (BO0L) END (BO0L) BUSY (BO0L) OK (BO0L) NG (DINT) PV 接続ユニット指定 シリアルポート番号 温度調節器ユニット番号 チャンネル番号 処理中フラグ 正常終了 異常終了 現在値
ファイル名	WLib\FDL\Japanese\Kornonlib\温度調節器\E5202ReadPV10.cxf WLib\FDL\Japanese\Kornonlib\温度調節器\E5202ReadPV10.cxf WLib\FDL\Japanese\Kornonlib\温度調節器\E5202ReadPV10.cxf
対象形式	ESAR/ESER/ES2N/ESCN/ESCN-U
使用条件	■外部接続 ・1対1接続できます。 ・シリアルコミュニケーションユニット(SCU)/シリアルコミュニケーションポート(SCB)と接続する場合、シリアルポートの設定を温度調節器の通信仕様(Compomay/F)と合わせてください。 ・シリアルコミュニケーションユニット(SCU)/シリアルコミュニケーションポート(SCB)はバージョン1.2以上を使用してください。 ・CPUユニットのRS232Cポートに繋ぐ場合は、PCシステム設定「RS232C通信ポート設定(Compomay/F)」は温度調節器の通信仕様(Compomay/F)と合わせてください。 ■設定 PCシステム設定「FB内通信命令共通設定」 ・通信命令レスポンス監視時間(初期値2秒) ・再送回数(初期値3回) ■使用共通資源 ・通信ポート(内部論理ポート)
機能説明	起動トリガをONにすると、温度調節器の指定チャンネルの現在値を読み出します。
FB使用上の注意	・本FBは数サイクルにまたがって処理を行います。出力変数BUSYによって、処理が実行中か否かを確認できます。 ・処理を終えると、OKまたはNGが1サイクルだけONします。これらのフラグを参照して、FBの処理完了を通知してください。
ENの入力条件	上記のように「起動トリガ」の立ち上がりでFBの出力であるBUSYの論理和をENに接続させます。
制限事項	・ENへの入力には必ず立ち上がり微分(1)を使用してください。 ・入力変数の取り得る範囲が機種・形式によって異なります。使用する機種・形式に応じた値をセットしてください。
出力変数	・本処理は、複数サイクルに渡って実行する必要があるため、処理完了まで実行されるように、出力変数BUSYを、入力変数ENへの入力条件のOR回路に、必ず挿入してください(「シンボル」の項参照)。 ・出力変数BUSYをFBでON/OFFしないでください。

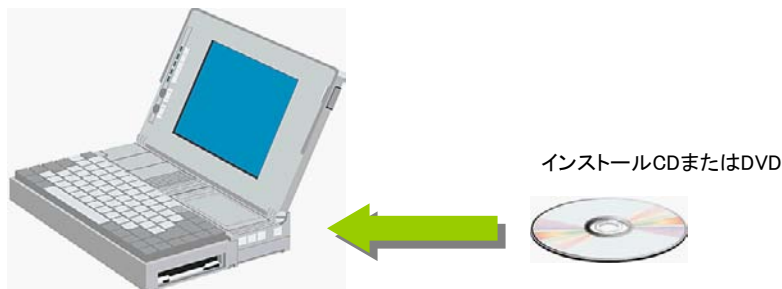
3-4. Smart FBライブラリの品揃えと入手方法

3-4-1. Smart FBライブラリの品揃え

種類	対象機器
コンポーネント	温度調節器、スマートセンサ、IDセンサ、視覚センサ、二次元バーコードリーダ など
PLC	CPUユニット、メモリカード、高機能ユニット (Ethernet、Controller Link、DeviceNetユニット、温度調節ユニット など)
メカトロ	位置制御、インバータ、サーボモータドライバ など
便利ツール	カレンダー、データ比較、デバッグ、パルス変換 など

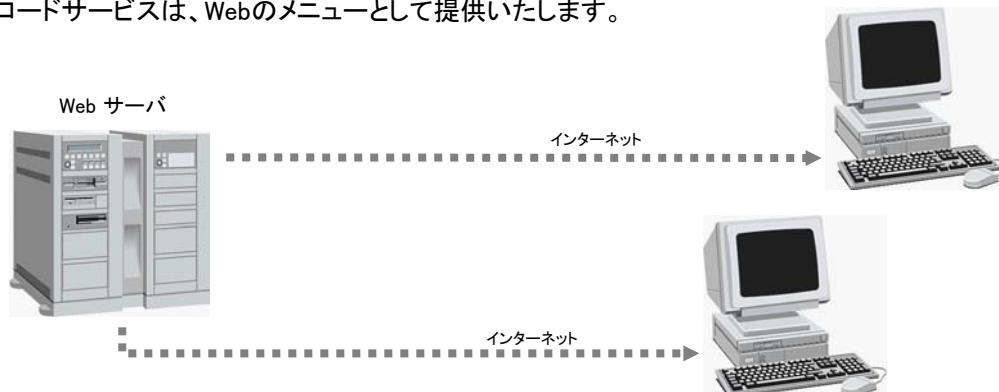
3-4-2. CX-OneインストールCDまたはDVDからの入手

Smart FBライブラリはCX-OneのインストールCDまたはDVDに同梱されています。
CX-Oneのインストール中にインストールを選択することができます。



3-4-3. Webサーバからの入手

Smart FBライブラリはWebサーバにより最新版のファイルを提供いたします。
新しいユニット用のライブラリ、機能アップをしたライブラリなど順次整備していきます。
ダウンロードサービスは、Webのメニューとして提供いたします。



第2章

**Smart FBライブラリを
使ってみましょう。**

プログラムの
説明プロジェクトの
新規作成ライブラリの
ロードプログラムの
作成プログラムの
チェック

1. 作成するプログラムの説明

ここでは、BCD型ON/OFF時間指定クロックパルスファンクションブロックをラダープログラムで使用する手順を例に、Smart FB部品の使用方法を説明します。

1-1. プログラムの仕様

プログラムは、以下の仕様で作成します。

接点 0.00 でパルスの生成開始。

コイル 1.00 にパルスを出力。

パルスのON間隔は、D100に格納された値。

パルスのOFF間隔は、固定値で 2秒。

1-2. FB部品の仕様

BCD型ON/OFF時間指定クロックパルス 部品は、以下の仕様です。

CPU 007	BCD 型 ON/OFF 時間指定クロックパルス _CPU007_MakeClockPulse_BCD
------------	--

機能概要	ON 時間 /OFF 時間を指定したクロックパルスを生成し、ENO に出力します。
シンボル	常時 ONP On On 時間(単位 100ms) Off 時間(単位 100ms) CPU007_MakeClockPulse_BCD (BOOL) EN (WORD) OnTime (WORD) OFFTime (BOOL) ENO

ファイル名	\\\\lib\\\\FBL\\\\Japanese\\\\Yomron\\\\lib\\\\プログラム\\\\コントローラ\\\\CPU\\\\CPU007_MakeClockPulse_BCD10.xaf
対象形式	CPU ユニット形 CS1+H CJ1+H CJ1M
使用条件	■PLC のプロパティ ・PLC のプロパティ「タイム/カウンタ現在値更新方式」が BCD であることが必要です。そうでない場合は、コンパイルエラーとなります。 「タイム/カウンタ現在値更新方式」は CX-Programmer 内「PC のプロパティ」にて設定します。

使用共通資源	・タイム
機能説明	「Off 時間(100ms 単位)」の間、ENO を OFF とした後、 「On 時間(100ms 単位)」の間、ENO を ON とし、それを繰り返します。

EN の入力条件	EN には常時 ONP On)を接続してください。
制限事項	・入力変数の値が範囲外の場合、ENO を OFF し、FB の内容は実行しません。
入力変数	・入力変数「On 時間(100ms 単位)」・「Off 時間(100ms 単位)」は BCD 値#0000～9999 の範囲で使用してください。範囲外の場合は出力 ENO が OFF となります。

使用例	下記ラダーでは、接点 A が 5 秒 ON-3 秒 OFF を繰り返します。 常時 ONP On On 時間(単位 100ms) 50 Off 時間(単位 100ms) 30 CPU007_MakeClockPulse_BCD (BOOL) EN (WORD) OnTime (WORD) OFFTime (BOOL) ENO
-----	---

関連 FB	PLC のプロパティ「タイム/カウンタの現在値更新方式」にしたがって FB を選択してください。 「タイム/カウンタ現在値更新方式」が BIN のとき FB「BIN 型 ON/OFF 時間指定クロックパルス」(CPU008 MakeClockPulse BIN) 「タイム/カウンタ現在値更新方式」が BCD のとき FB「BCD 型 ON/OFF 時間指定クロックパルス」(CPU007 MakeClockPulse BCD)
-------	---

■変数テーブル					
【INPUT】(入力変数)					
名称(和文)	変数名	データ型	初期値	有効範囲	説明
EN	EN	BOOL			I(ON): FB を起動する O(OFF): FB を起動しない
On 時間	OnTime	WORD		#0000～ #9999	ON 時間を 100ms 単位で指定します。 例えば、3 秒のときは#30 と指定します。
Off 時間	OFFTime	WORD		#0000～ #9999	OFF 時間を 100ms 単位で指定します。 例えば、3 秒のときは#30 と指定します。

【OUTPUT】(出力変数)

名称(和文)	変数名	データ型	有効範囲	説明
ENO	ENO	BOOL		ON 時間 OnTime の間 ON し、OFF 時間 OFFTime の間 OFF します。

プログラムの
説明

プロジェクトの
新規作成

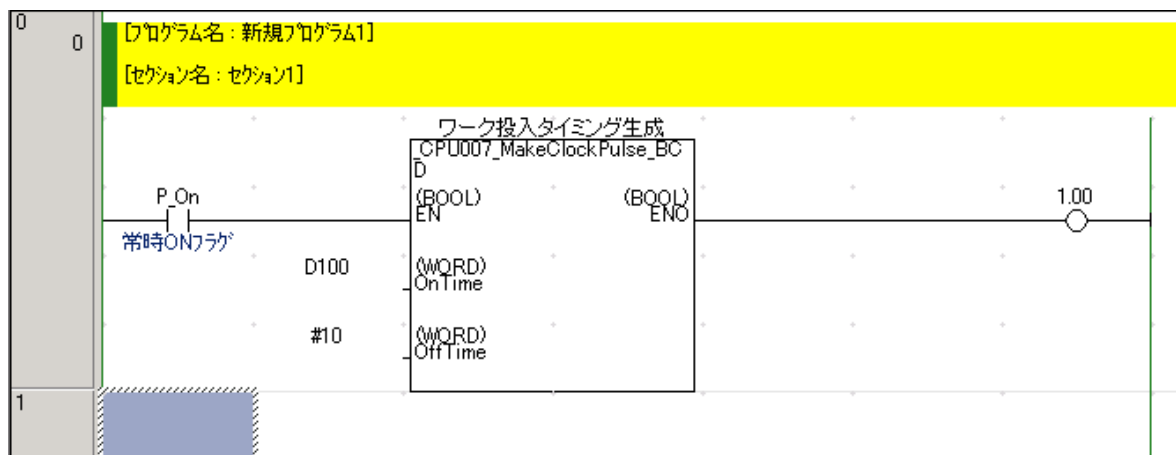
ライブラリの
ロード

プログラムの
作成

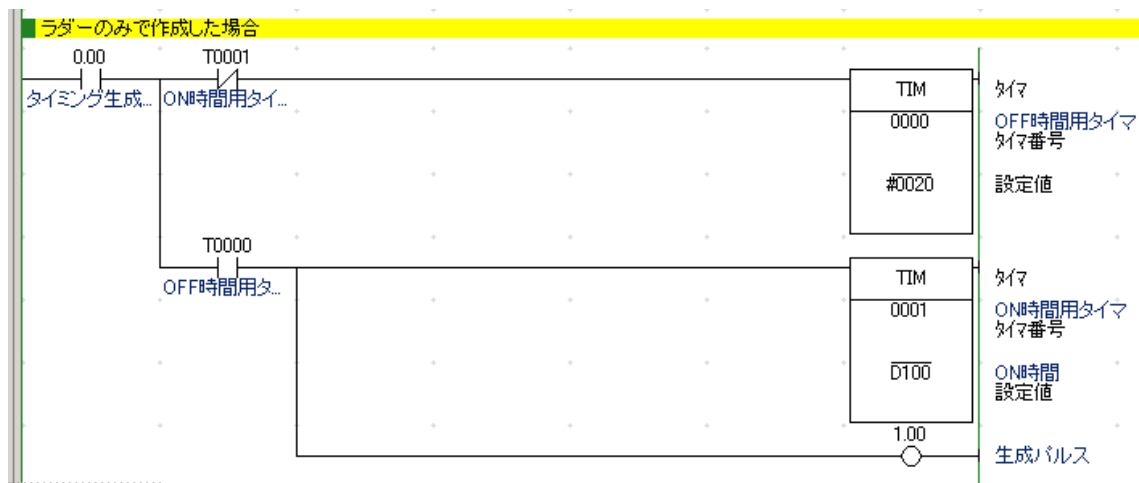
プログラムの
チェック

1-3. 入力するプログラム

以下のようなプログラムを作成します。



【参考】ラダープログラムで作成すると以下のようなプログラムになります。

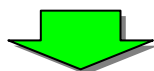
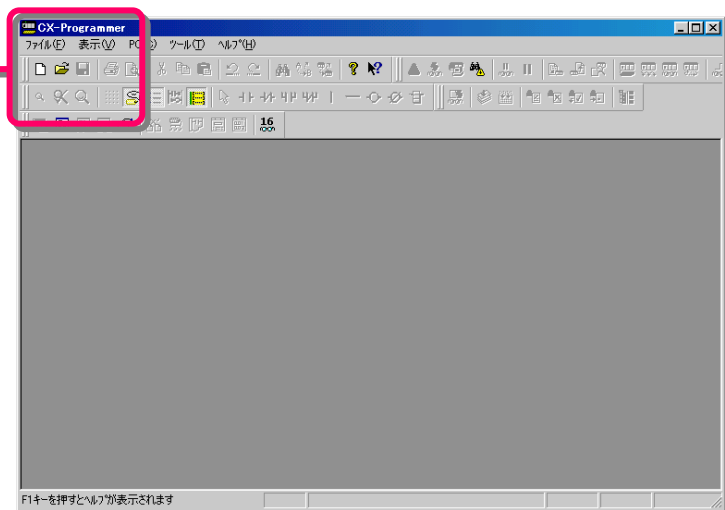


プログラムの
説明プロジェクトの
新規作成ライブラリの
ロードプログラムの
作成プログラムの
チェック

2. プロジェクトの新規作成とPLC機種／CPU形式の設定

CX-Programmerの新規プロジェクト作成ボタンをクリックします。

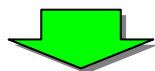
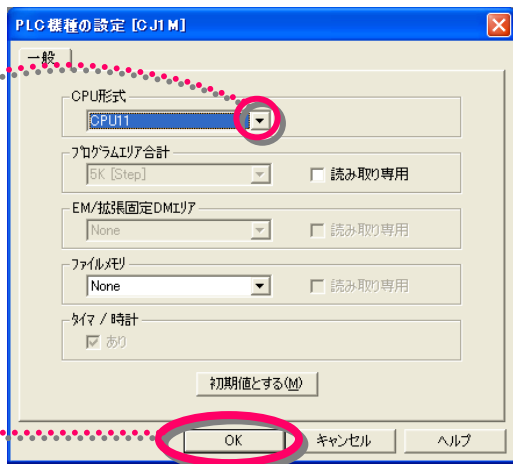
をクリック

▼ を左クリックして
PLC機種を選択します。

！ファンクションブロックを使用するためには、
以下のPLC機種から選択して下さい。
CJ2H、CJ2M、CJ1H-H、CJ1G-H、CJ1M、
CS1H-H、CS1G-H、CP1H、CP1L、CP1L-E

設定(S)...

マウスを左クリック

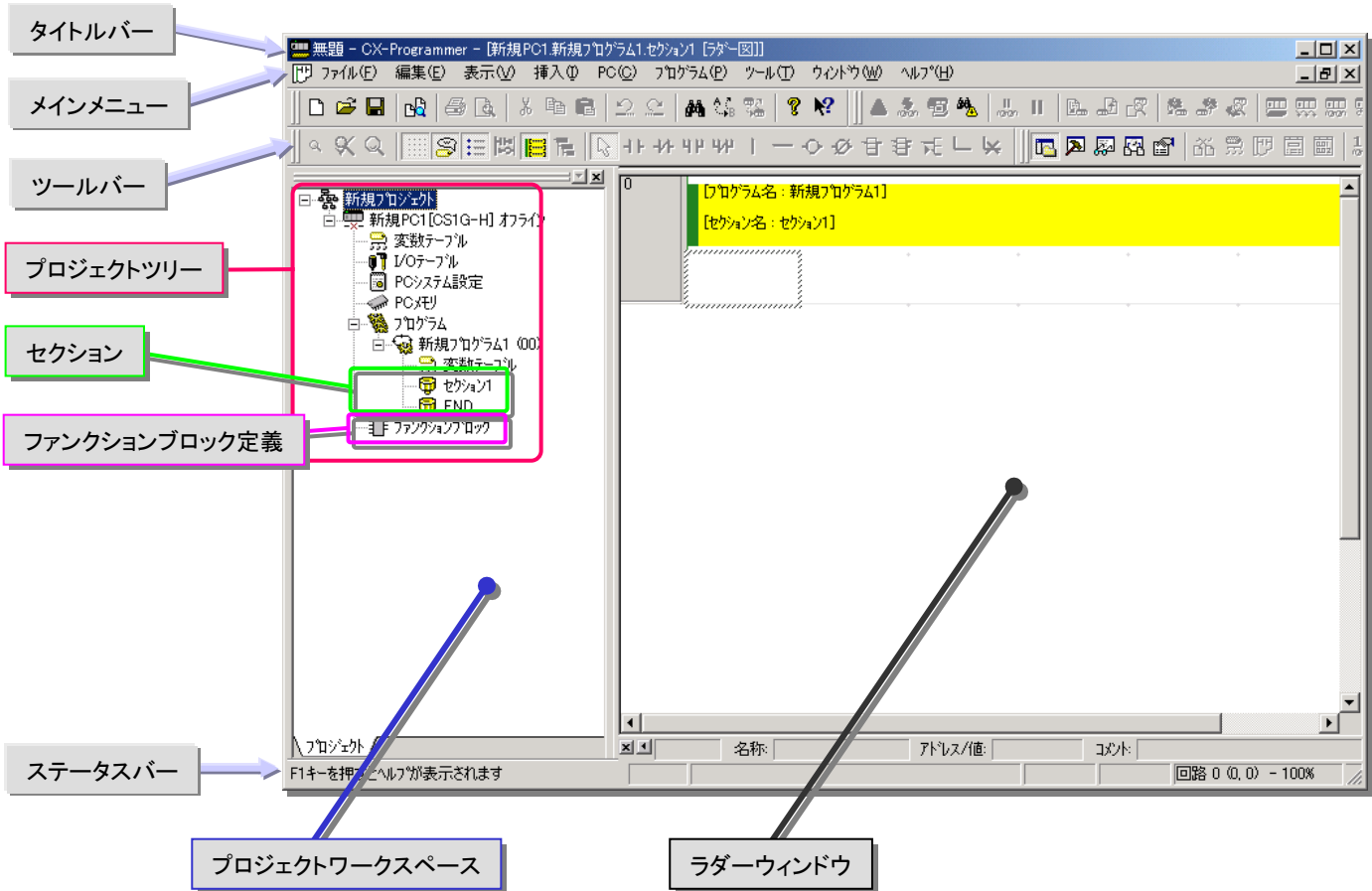
▼ を左クリックしてCPU
型を選択します。

OK

[OK]をクリックして
CPUの形式を確定します。

3. メインウィンドウについて

メインウィンドウの各部の機能について説明します。



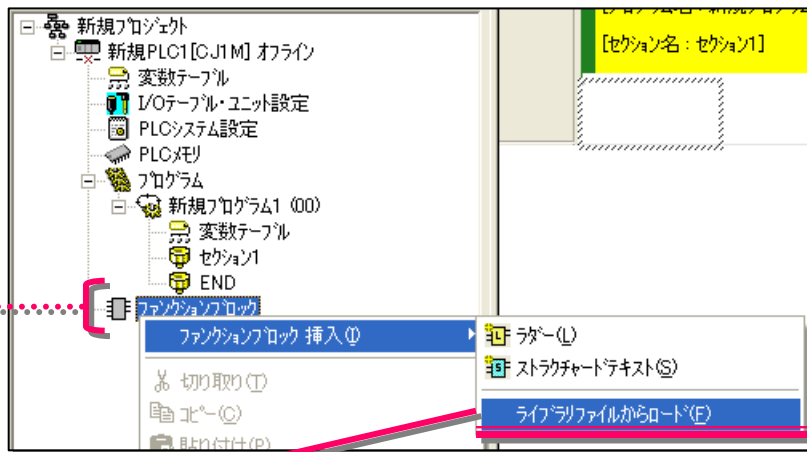
名 称	内 容・機 能
タイトルバー	CX-Programmerで作成したデータの保存ファイル名を表示。
メインメニュー	メニューを選択します。
ツールバー	アイコンのクリックで機能を選択。[表示]→[ツールバー]で表示するツールバーを選択。ドラッグするとグループごとに表示位置を変更できます。
セクション	1つのプログラムを任意の数に分割して、作成／表示することが出来ます。
プロジェクトワークスペース プロジェクトツリー	プログラムや各種のデータを管理します。異プロジェクト間、プロジェクト内において要素単位でドラッグ&ドロップによるデータのコピーが出来ます。
ラダーウィンドウ	ラダープログラムの作成、編集を行う画面です。
ファンクションブロック定義	ファンクションブロック定義が表紙されるツリーです。 アイコンを選択して、コピー、削除などができます。 ・Smart FB部品の場合  と表示します。 ・ユーザ定義ファンクションブロックの場合、ラダーは  、STの場合  と表示します。
ステータスバー	PCネーム、オフライン／オンライン、アクティブセルの位置等の情報を表示。

プログラムの
説明プロジェクトの
新規作成ライブラリの
ロードプログラムの
作成プログラムの
チェック

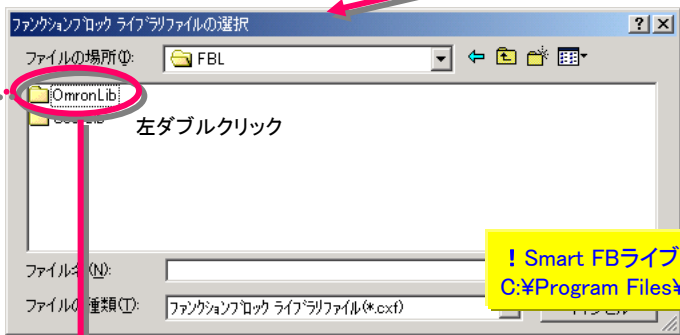
4. 部品ファイルのインポート

マウスカーソルをファンクションブロック定義アイコンに合わせ、マウス右クリックでメニューを表示します。
メニューのファンクションブロック挿入を選択した後、ライブラリファイルを選択します。

マウス右クリック
→ファンクションブロック挿入
→ライブラリファイルからロード



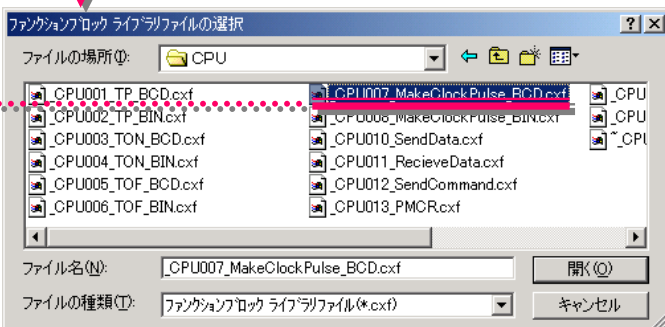
マウス左ダブルクリックで
[OmronLib]
[プログラマブルコントローラ]
[CPU]
を順に選択します。



ファンクションブロックライブラリ
選択ダイアログで目的とするライブ
ラリファイルを選択します。

! Smart FBライブラリのデフォルトのインストール先は
C:\Program Files\Omron\CX-One\Lib\FBLです。

マウス左クリックで
_CPU007_MakeClockPulse_BCD.cxf
を選択



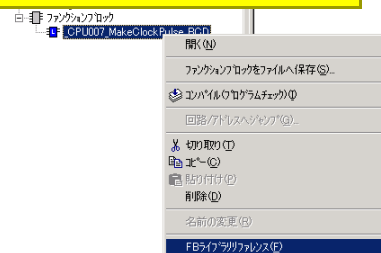
[開く]ボタンを
マウス左クリック

_CPU007_MakeClockPulse_BCD という
ファンクションブロック定義が登録されます。

! 登録されたSmart FB部品を選択、ポップ
アップメニューで[FBライブラリリファレンス]
を選択すると、ライブラリリファレンスファイル
を表示して、Smart FB部品の仕様を容易に確認
することができます。

ファンクションブロック定義

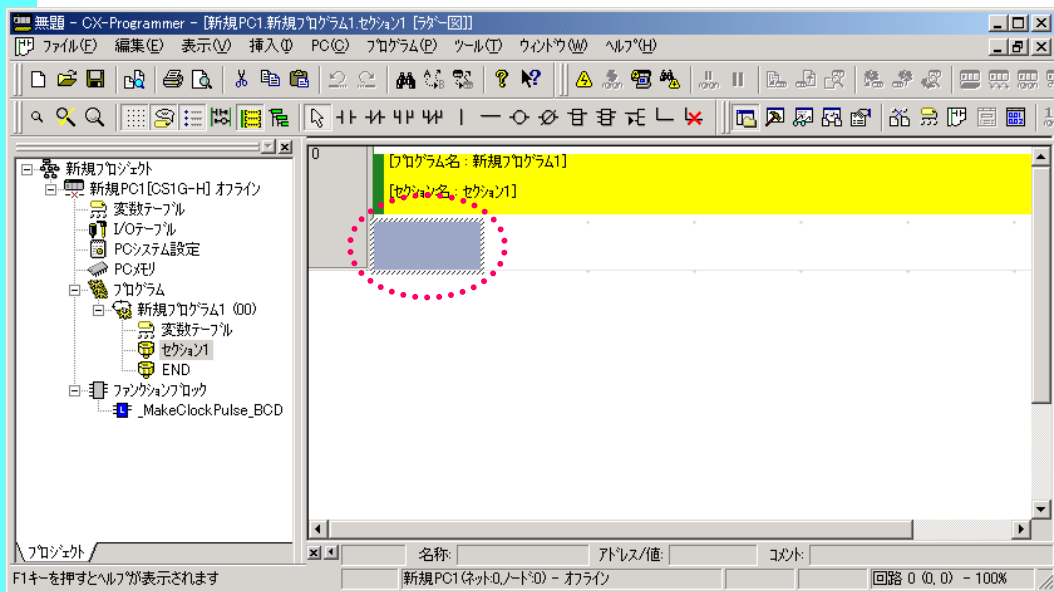
_CPU007_MakeClockPulse_BCD



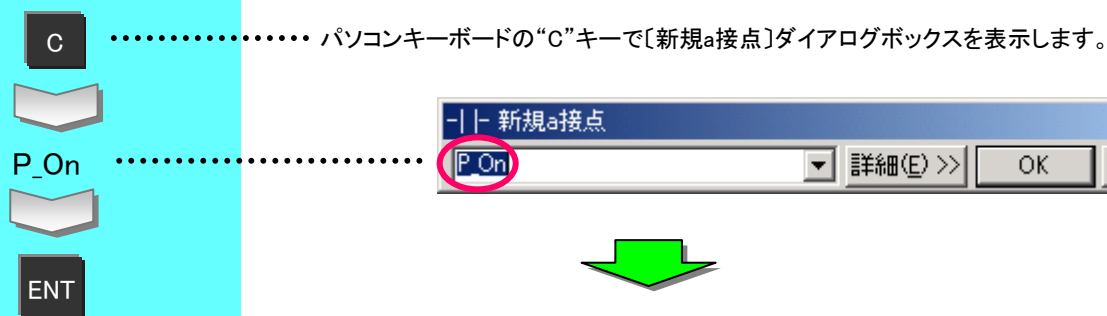
プログラムの
説明プロジェクトの
新規作成ライブラリの
ロードプログラムの
作成プログラムの
チェック

5. プログラムの作成

カーソルがラダーウィンドウ内の左上にあることを確認してプログラミングを開始します。



5-1. a接点の入力



入力した命令の削除方法

- ・命令の上にカーソルを移動してDELキー
- ・命令の右にカーソルを移動してBSキー

“P_On”とは常時ONを意味するシステム定義変数です。
アドレスを入力した場合、アドレスの上位桁の0は省略して表示されます。
また、チャンネル番号とリレー番号の間に[.]ピリオドが表示されます。

プログラムの
説明

プロジェクトの
新規作成

ライブラリの
ロード

プログラムの
作成

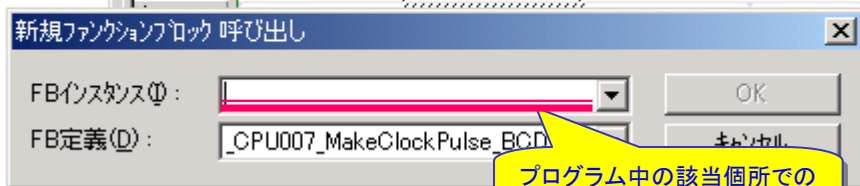
プログラムの
チェック

5-2. インスタンスの入力

F

..... パソコンキーボードの“F”キーで[新規ファンクションブロック 呼び出し]ダイアログボックスを表示します。

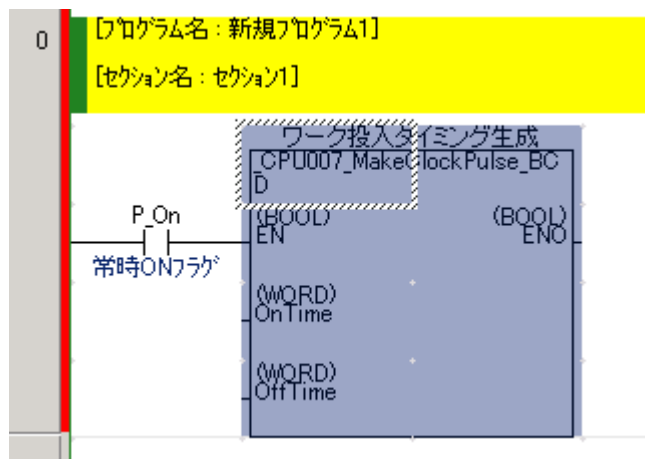
FBインスタンス名称を入力
[ワーク投入タイミング生成]



プログラム中の該当個所での
処理名を入力します。

ENT

..... 「ワーク投入タイミング生成」という名前のファンクションブロックインスタンスを表示します。



5-3. パラメータの入力

P

または

ENT

アドレスを入力します。

[d100]

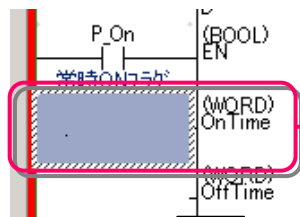
ENT

IOコメントを入力します。

[ON時間]

ENT

2-7



カーソルをパラメータの入力を行いたい
入力変数の左側に移動します。



入力パラメータ OnTimeの
アドレスD100が決まりました。



プログラムの
説明

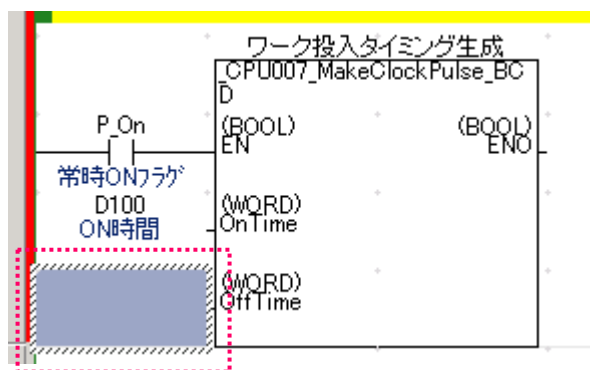
プロジェクトの
新規作成

ライブラリの
ロード

プログラムの
作成

プログラムの
チェック

残りのパラメータを入力します。

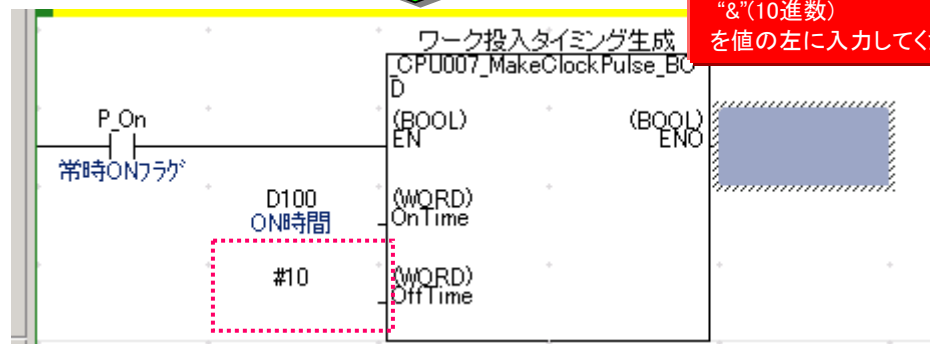


新規パラメータ

#10

詳細(E) >> OK キャンセル

定数の入力時は必ず
"#" (16進数/BCD値)
または
"&" (10進数)
を値の左に入力してください。



-(-) 新規出力

1.00

詳細(E) >> OK キャンセル

-(-) 新規出力

コメントの編集 (1/1): 1.00

1.00 生成パルス

OK キャンセル



P または ENT

#10

ENT

O

1.00

ENT

〔生成パルス〕

ENT

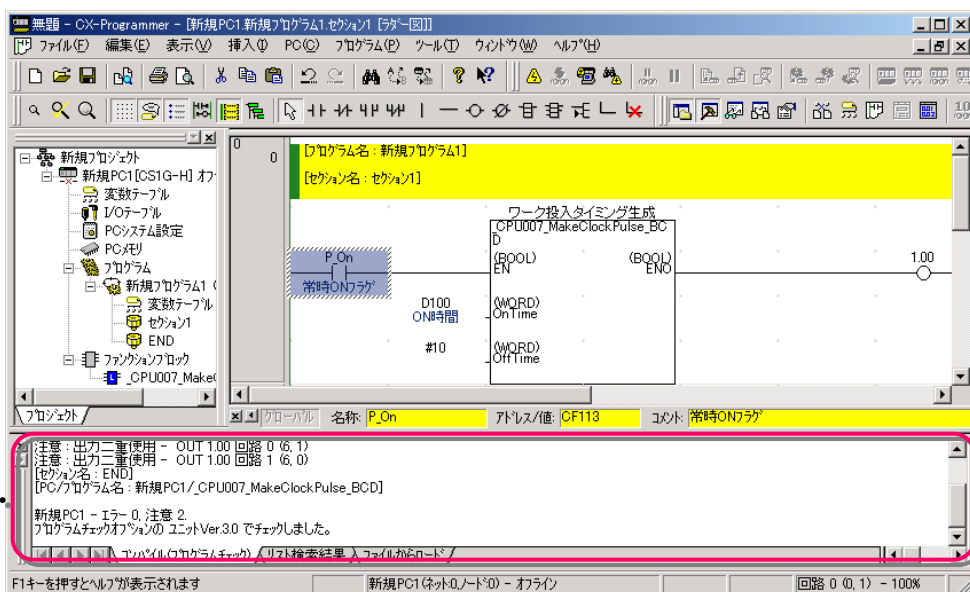
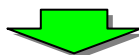
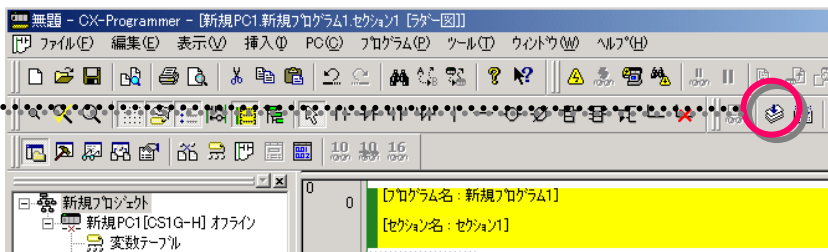
プログラムの
説明プロジェクトの
新規作成ライブラリの
ロードプログラムの
作成プログラムの
チェック

6. プログラムのエラーチェック（コンパイル）

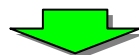
プログラムを転送する前にエラーのチェックを行います。



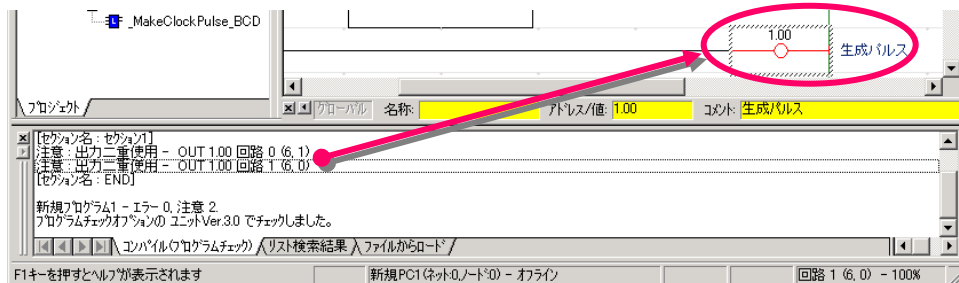
をクリック



プログラムにエラーがある場合、アウトプットウィンドウにエラー内容とアドレスが表示されます。



表示されているエラー情報をダブルクリックすると、ラダーウィンドウのカーソルがエラー箇所へジャンプし、エラー回路が赤字で表示します。



エラー箇所を修正します。

- ・アウトプットウィンドウはプログラムチェックを行うと自動的に開きます。
- ・JキーまたはF4キーでエラー箇所へジャンプすることが出来ます。
- ・アウトプットウィンドウはEscキーで閉じることが出来ます。

オンライン
～転送

導通モニタ

オンライン
エディット

7. オンライン接続

CX-Programmerでは、ご使用方法に合わせて3種類の接続方法をご用意しています。



・通常のオンライン接続。新規プロジェクト作成時に指定した機種・方法でPCと接続します。



・自動オンライン接続。1キーで相手PLC機種を自動認識して接続→PCのプログラム等の吸い上げを一括して行います。



・シミュレータ接続。1キーでCX-Simulatorに接続します。(別途シミュレータのインストールが必要です。)

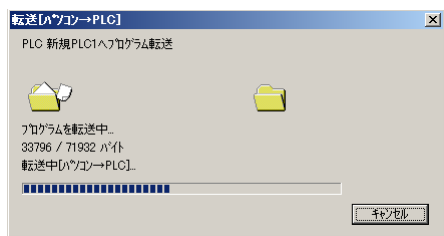
今回はシミュレータ接続で、オンライン/ デバッグ機能を紹介します。
(別途CX-Simulatorのインストールを行って下さい。)



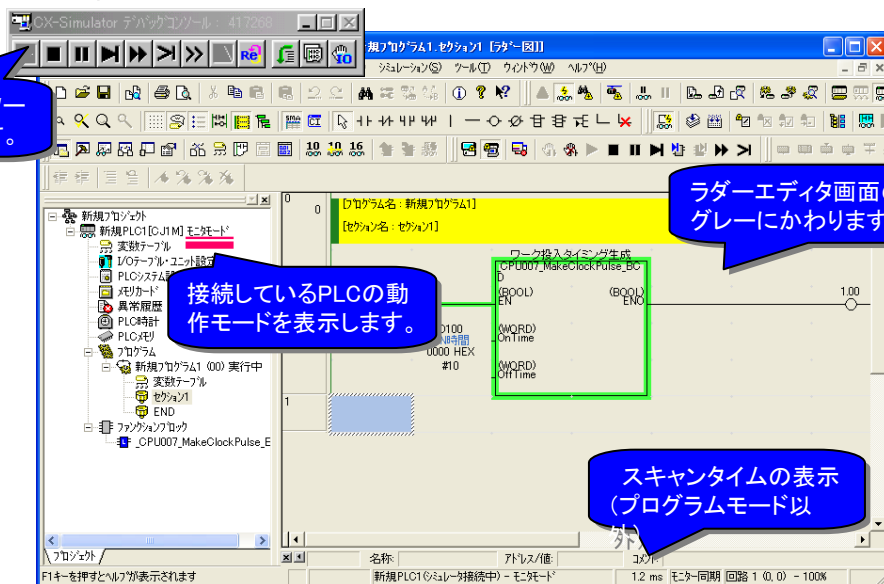
をクリック



プログラムの転送が始まります。



オンライン時の画面



CX-SimulatorのコンソールBoxが表示されます。

接続しているPLCの動作モードを表示します。

ラダーエディタ画面の背景がグレーにかわります。

スキャンタイムの表示
(プログラムモード以

オンライン
～転送

導通モニタ

オンライン
エディット

8. 導通モニター1

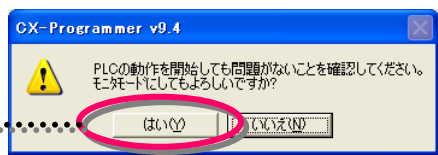
PLC(シミュレータ)を
モニタモードにします。

をクリック



[はい]をクリック

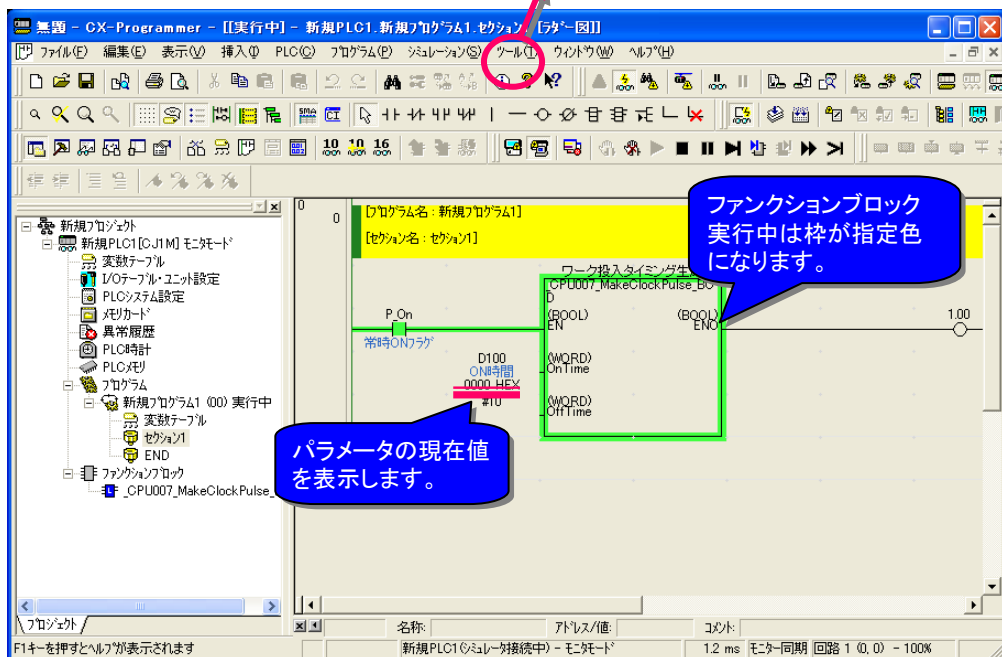
接点、コイル等のON/OFF状態をモニタします。



容量の大きいプログラムの導通モニタを行うと画面のスクロール速度が遅くなることがあります。
その際は下記のアイコンをクリックして導通モニタを一時解除→見たいアドレスまで画面をスクロール→導通モニタモードに戻す。という操作を行うと便利です。



…導通モニタの ON/OFF



オンライン
～転送

導通モニタ

オンライン
エディット

9. 導通モニター2 パラメータの現在値変更

導通モニタにて、接点・チャネルなどの現在値変更を行います。

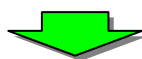
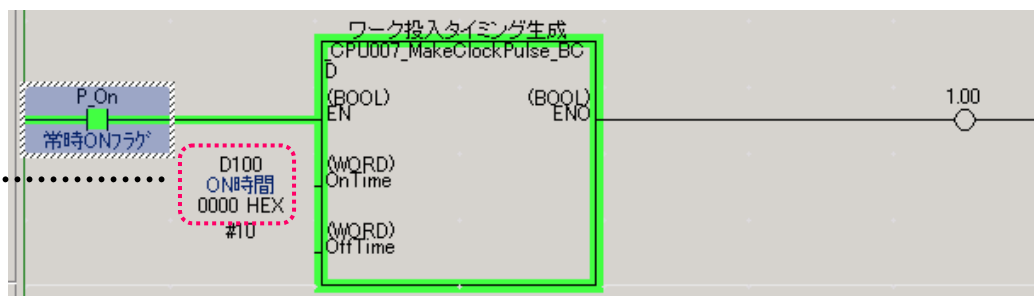
カーソルを入力パラメータ
D100に移動します。

マウスを右クリックし
プルダウンメニューから
[セット/リセット(S)]
→[設定値(V)]
を選択。

または
マウスの左ボタンを
ダブルクリック

または

ENT



新しい値を設定

アドレス(A): D100

データ型(D): UINT

値(V): #0020

#0 ~ #FFFF (16H)

設定

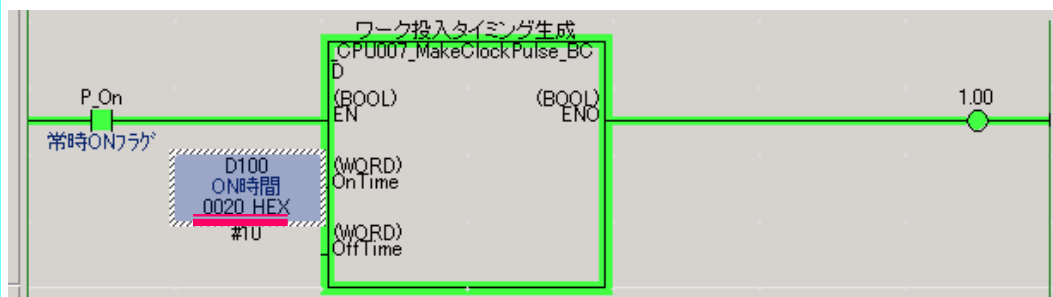
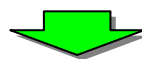
キャンセル

入力パラメータの現在値変更を行います。



[設定]をクリック

定数の入力時は必ず
“(16進数/BCD値)または
“(10進数)を値の左に入力
してください。



オンライン
～転送

導通モニタ

オンライン
エディット

10. オンラインエディット

修正したい回路に、
カーソルを移動します。……

マウスのドラッグ&ドロップで
複数の回路を一括で指定する
ことも可能です。

【プログラム】
→【オンラインエディット】
→【開始】を選択します。

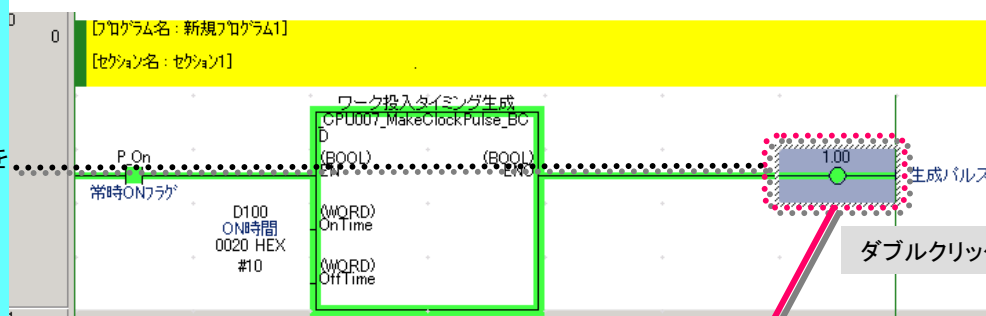
ショートカット【Ctrl】+【E】

変更したい接点の上にカーソルを
移動してマウスの左ボタンを
ダブルクリックします。

修正したい接点番号（回路例では 4.11）を入力します。

【プログラム】
→【オンラインエディット】
→【変更を転送】
を選択します。

ショートカット
【Ctrl】+【Shift】+【E】

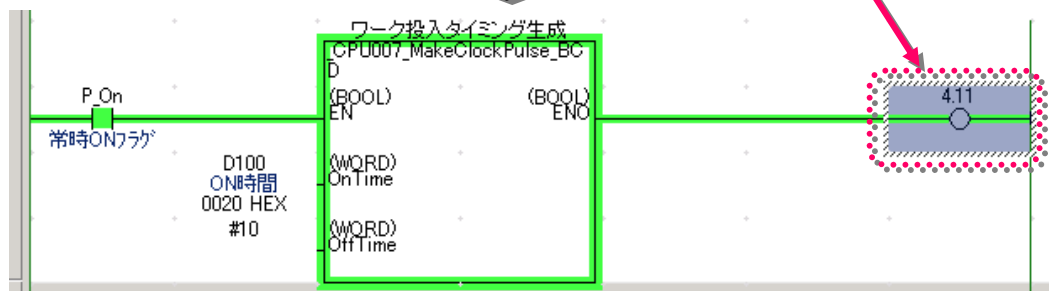


-() 出力の編集

4.11

詳細(E) X

OK キャンセル



終了

第3章

**Smart FB部品を
カスタマイズしてみましょう。**

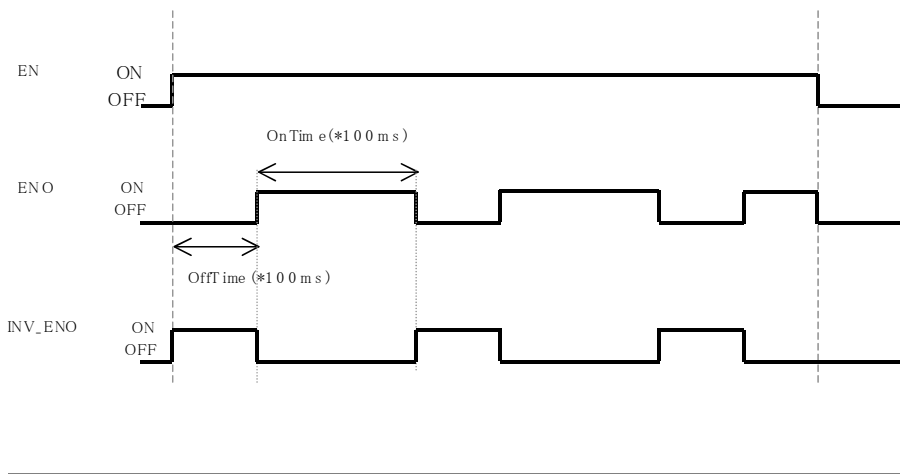
1. 作成するプログラムの説明

ここでは、第1章で説明に用いた、BCD型ON/OFF時間指定クロックパルス部品をカスタマイズする手順を例に説明します。

1-1. 変更仕様

BCD型ON/OFF時間指定クロックパルスファンクションブロックは、「Off時間(100ms単位)」の間、ENOをOFFとした後、「On時間(100ms単位)」の間、ENOをONとし、それを繰り返します。

今回の変更は、INV_ENOという出力パラメータを追加し、ENOを反転した値を出力します。



1-2. ライブラリの変更内容

上記の変更仕様を満たすために、以下の追加・変更をライブラリに対して行います。

1. 出力変数“INV_ENO”の追加。
2. ENOの反転出力をINV_ENOに出力。

お願い

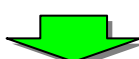
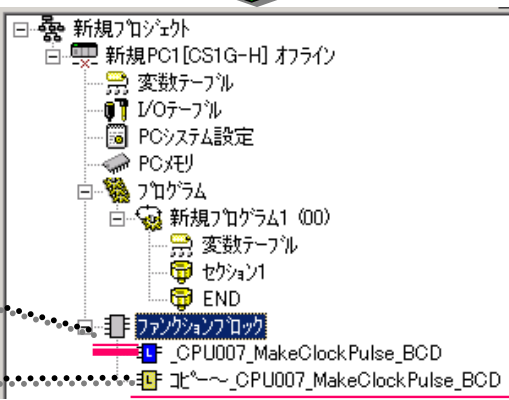
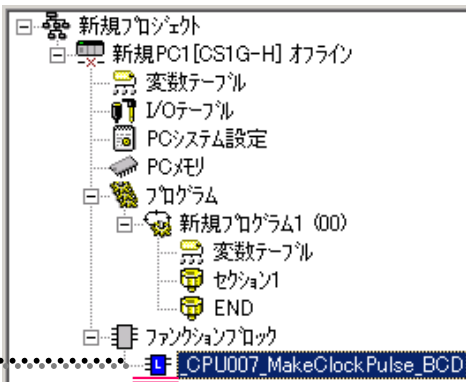
とくに、お客様が、オムロン提供のSmart FB部品を改造してご使用になる場合、改造前に「CX-Programmer オペレーション マニュアル ファンクションブロック編/ストラクチャードテキスト編」をよく読んで、ファンクションブロック機能の仕様内容を十分ご理解ください。

また、改造後は、ファンクションブロック定義としての動作確認を十分行った上で、本運転に入るようにしてください。

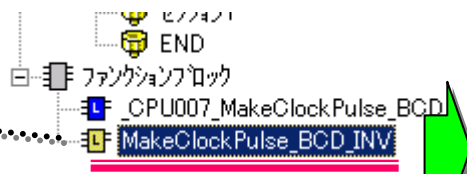
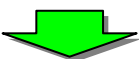
なお、お客様がSmart FB部品をもとに改造した場合、そのファンクションブロック定義の動作に関して、当社として保証することはできません。また、それに関するお問い合わせに対応することはできませんので、あらかじめご了承ください。

2. FB部品のコピー

第1章の説明で用いた、BCD型ON/OFF時間指定クロックパルスファンクションブロック
(FB定義名: _CPU007_MakeClockPulse_BCD)をインポートします。

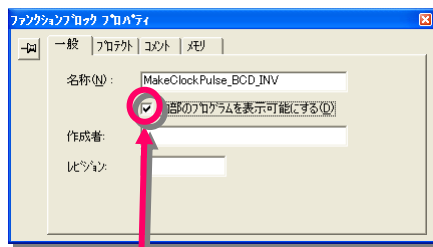


FB定義名を変更します。



ご注意
“_”で始まるファンクションブロック定義は
ユーザは作成できません。
必ず“_”以外の文字で始まる名前にしてください。

FB定義を表示可能に変更します。



マウスカーソルでクリックしチェックマークをつけます。

マウスカーソルをオムロンFB
ライブラリ定義アイコン 
に合わせ、マウス右クリック
→コピー(C)



マウスカーソルをファンクションブ
ロックアイコン  に合わせ、マ
ウス右クリック→貼り付け(V)

FB部品がコピーされました。



マウスカーソルをコピーされた
ファンクションブロック
アイコン  に合わせ、
マウス右クリック
→名前の変更(R)
[MakeClockPulse_BCD_INV]
と入力します。



マウスカーソルをコピーされた
ファンクションブロック
アイコンに合わせ、
マウス右クリック→プロパティ(O)

または

ALT + ENT

3. ファンクションブロック内変数の追加

ファンクションブロックラダーエディタを開きます。

変数テーブル

ファンクションブロックラダー
エディタが開きます。

マウカーソルをファンクション
ブロックアイコン  に合わせ、
マウスの左ボタンをダブルクリッ
ク

Smart FB部品は同様の方法でラダー図を参照できますが、編集はできません。

ラダー入力画面

変数テーブル

名称	データ型	AT	初期値	保持	コメント
ENO	BOOL		FALSE		ファンクションブロック 実行結果フラグ (
内部変数	入力変数	出力変数	外部変数		

マウスカーソルを変数テーブル
の出力変数タブに合わせマウス
を左クリック

マウスを右クリックし、プルダウンメニューから変数追加(I)を選択

追加する変数名称を入力

新規変数

名称(N): INV_ENO

データ型(D): BOOL

用途(U): O: 出力

初期値(V): FALSE ☐ 保持(R)

コメント(C): ENOの反転出力

OK

キャンセル

詳細(A)...

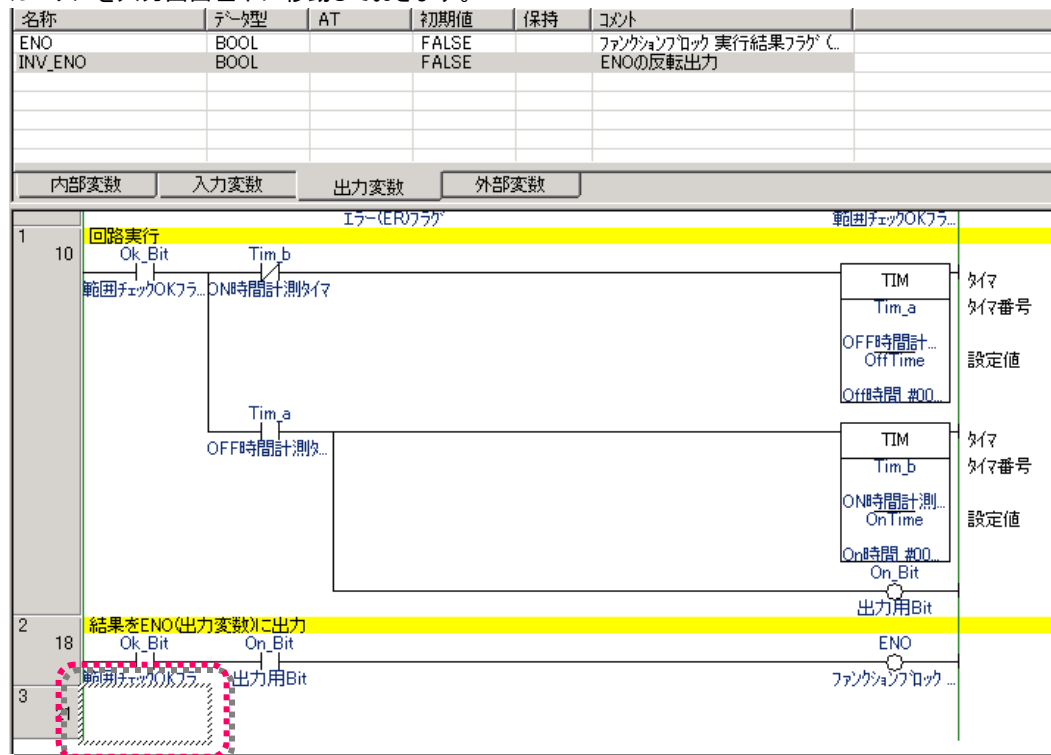
ビットデータなので
BOOLを選択

入力した変数がテーブルに正しく書き込まれているか確認します。

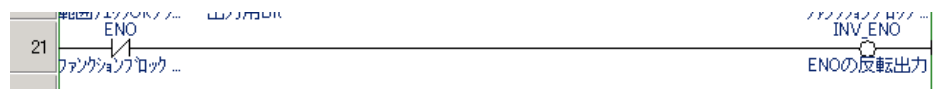
名称	データ型	AT	初期値	保持	コメント
ENO	BOOL		FALSE		ファンクションブロック 実行結果フラグ (
INV ENO	BOOL		FALSE		ENOの反転出力

4. ファンクションブロックラダーの変更

ファンクションブロックラダー入力画面にてラダーを追加します。
カーソルを入力画面左下に移動しておきます。

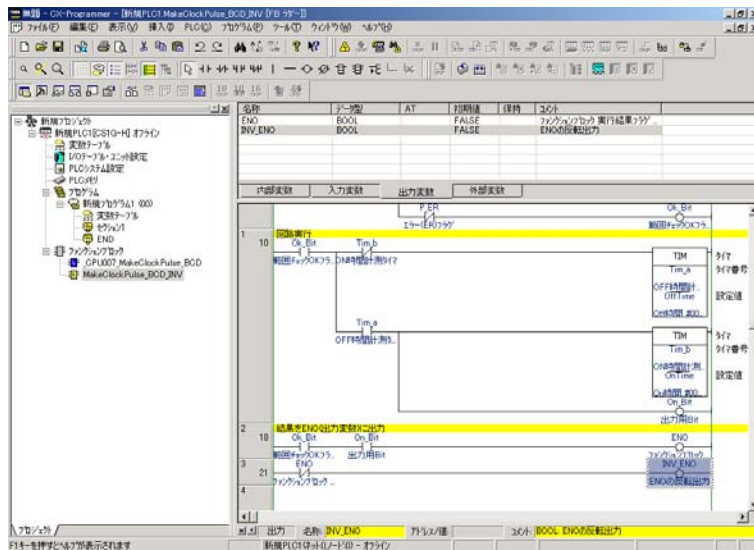


4-1. 接点の入力



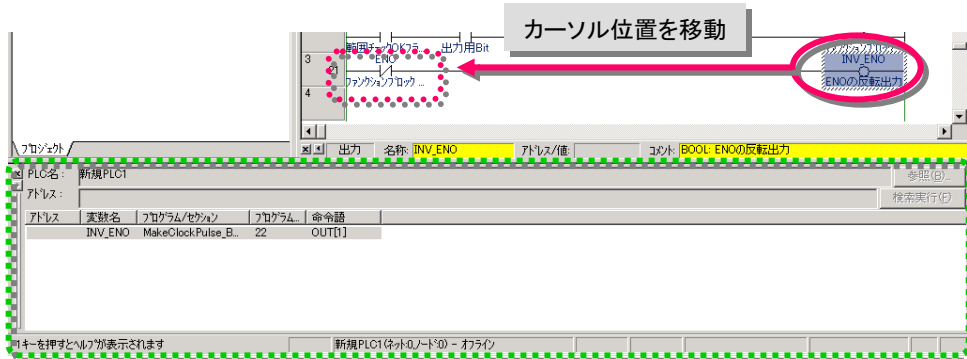
4-2. 変数の使用状況の確認

メインのラダープログラムと同様に、クロスリファレンスポップアップを用いて変数の使用状況の確認を行います。



クロスリファレンスポップアップ
を表示させます。

Alt + 4



マウスカーソルでクロス
リファレンスポップアップの
LDNOT を選択。

↑

変数ENOがステップNo.20の出力コイルでも使用されていることがわかります。



FBラダーエディタのカーソルがステップNo.20の出力コイルに移動します。

5. PLCへの転送

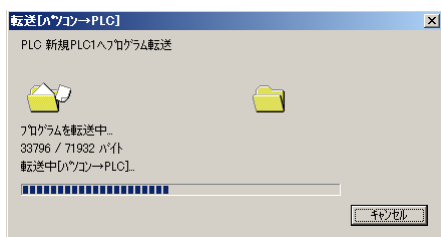
Smart FB ライブラリのカスタマイズを行ったファンクションブロック定義を使用したプログラムをPLCに転送します。



をクリック



プログラムの転送が始まります。



6. 動作確認

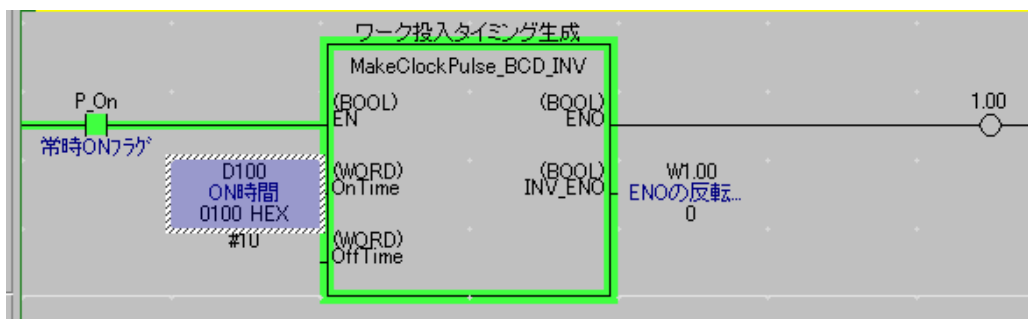
ファンクションブロックのパラメータに指定したD100(ON時間)を変更しながら、プログラムの動作確認・デバッグを行います。

マウスを右クリックし
プルダウンメニューから
「セット/リセット(S)」
→「設定値(V)」
を選択。

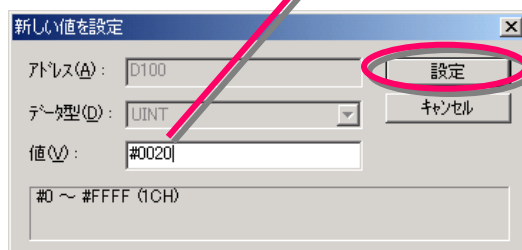
または
マウスの左ボタンを
ダブルクリック

または

ENT



入力パラメータの現在値変更を行います。



「設定」をクリック

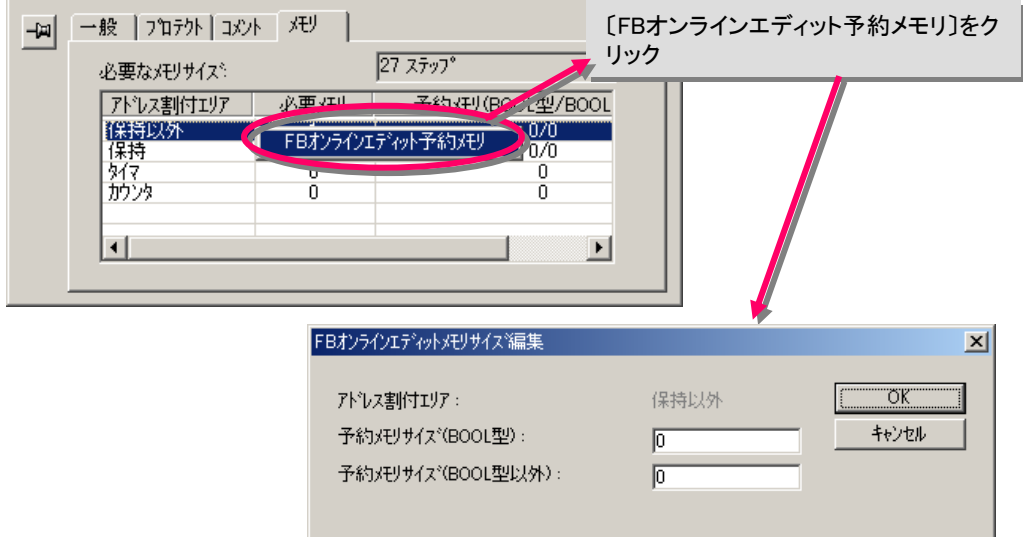
定数の入力時は必ず
“#”(16進数/BCD値)または
“&”(10進数)を値の左に入力
してください。

7. ファンクションブロックのオンラインエディット

FBオンラインエディットで変数(内部変数)を追加する場合は、事前にオフラインのファンクションブロックプロパティの[メモリ]タブの設定にて内部変数を追加するためのメモリを確保する必要があります。

オンラインエディットしたいファンクションブロック定義を選択し、右クリックのプルダウンメニューから「プロパティ」を選択。

オンラインエディット時に追加したい変数のエリアを選択し、右クリックのプルダウンメニューから「FBオンラインエディット予約メモリ」を選択。



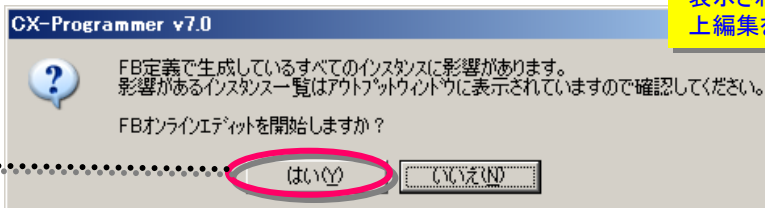
ファンクションブロック定義のオンラインエディットを行います。

オンラインエディットしたいファンクションブロック定義を選択し、右クリックのプルダウンメニューから「FBオンラインエディット」→「開始」を選択。



ファンクションブロック定義のオンラインエディットはユニットバージョン4.0以上の実機PLCのみで使用可能です。CX-Simulatorでは使用できません。

FB定義が複数箇所から呼び出されている可能性がありますので、アウトプットウィンドウに表示される使用状況を確認の上編集を開始してください。



「はい」をクリック。

回路の修正を行います。



オンラインエディットで
回路の修正を行ったあとで
マウスを右クリックし
プルダウンメニューから
[FBオンラインエディット]
→[変更を転送]
を選択。

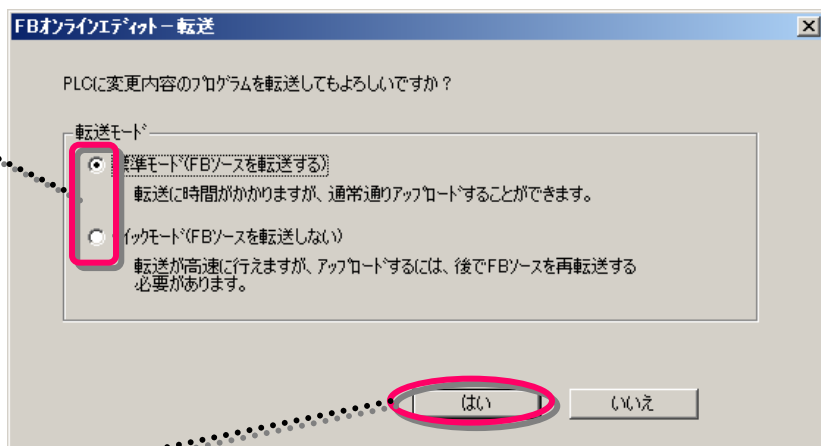
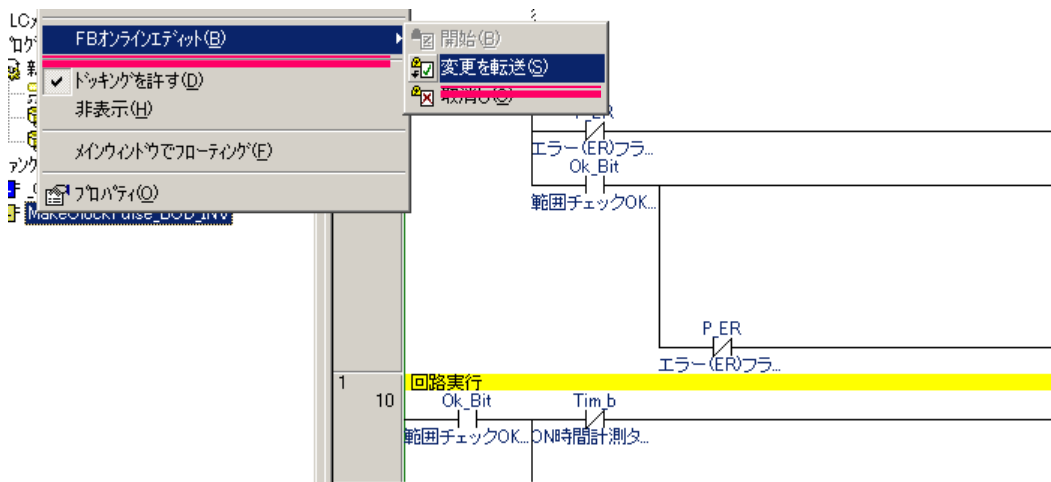
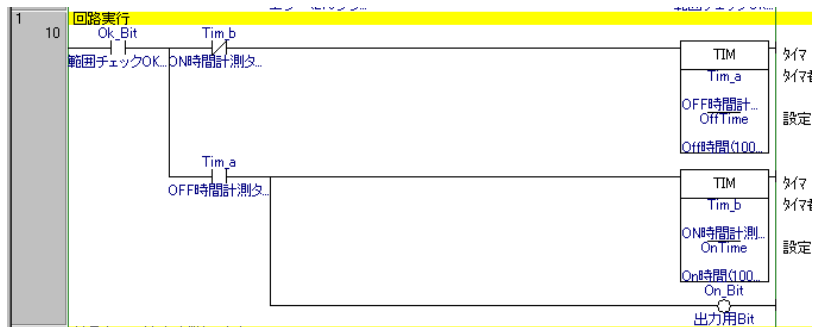


オンラインエディットの
転送モードを選択します。

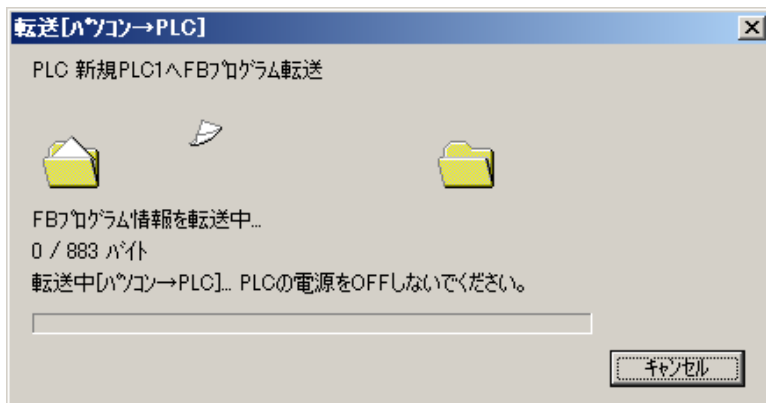
- ・標準モード:FBソース情報を
含めて転送します。
- ・クイックモード:FBソースなし
で転送します。



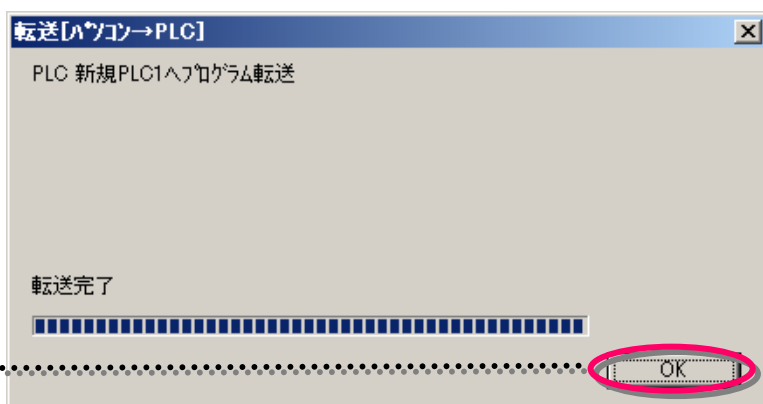
「はい」をクリック。



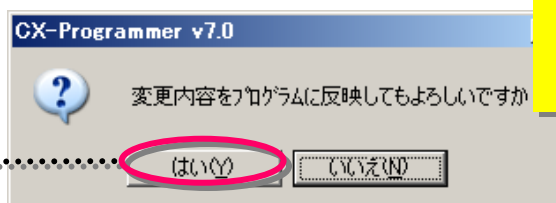
FB定義情報が転送されます。



〔OK〕をクリックします。



〔はい〕をクリックします。



サイクルタイムが延びても影響がないことを確認してから、〔はい〕をクリックしてください。
入力信号を読み取れないことがあります。

終了

第4章

ST言語を使用してみましょう。

プログラムの
説明FB定義の
新規作成

変数の入力

FB内プログラム
の作成ラダープログラムの
作成 & チェック

1. ST言語とは

ST(構造化テキスト)言語とは、IEC 61131-3(JIS B3503)で規定されている、工業制御(おもにPLC)用の高級言語です。IF-THEN-ELSE, FOR/WHILE ループなどの制御構文をもち、算術演算の関数(SIN, LOGなど)も使用できますので、数値演算処理に適します。CX-Programmer でサポートしているST言語は、IEC 61131-3(JIS B3503)に準拠しています。

CX-Programmer で使用可能な算術演算関数は以下です。

サイン (SIN), コサイン (COS), タンジェント (TAN), アークサイン (ASIN), アークコサイン (ACOS), アークタンジェント (ATAN), 平方根 (SQRT), 絶対値 (ABS), 対数 (LOG), 自然対数 (LN), 自然指数 (EXP), べき乗 (EXPT), 剰余 (MOD)

```
(* Initial Settings *)
XMT[1] := 2;
XMT[2] := 7;
N := 2;

(* CRC16 *)
CRCTMP := 16#FFFF;
FOR I := 1 TO N DO
  CRCTMP := CRCTMP XOR XMT[I];
  FOR J := 1 TO 8 DO
    CT := CRCTMP AND 1;
    IF CRCTMP < 0 THEN
      CH := 1;
      CRCTMP := CRCTMP AND 16#7FFF; (* CRCTMP & 0x7FFF *)
    ELSE
      CH := 0;
    END_IF;
    UINT_CRCTMP := WORD_TO_UINT(CRCTMP) / 2;
    CRCTMP := UINT_TO_WORD(UINT_CRCTMP);
    IF CH = 1 THEN
      CRCTMP := CRCTMP OR 16#4000; (* CRCTMP OR 0x4000 *)
    END_IF;
    IF CT = 1 THEN
      CRCTMP := CRCTMP XOR 16#A001; (* CRCTMP XOR 0xA001 *)
    END_IF;
  END_FOR;
END_FOR;

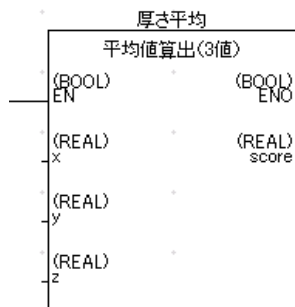
IF CRCTMP < 0 THEN
  CL := 1;
  CRCTMP := CRCTMP AND 16#7FFF; (* CRCTMP & 0x7FFF *)
ELSE
  CL := 0;
END_IF;

C_1 := CRCTMP AND 16#FF; (* CRCTMP & 0xFF *)
CRCTMP := CRCTMP AND 16#7F00; (* CRCTMP & 0x7F00 *)
UINT_CRCTMP := WORD_TO_UINT(CRCTMP) / 256;
C_2 := UINT_TO_WORD(UINT_CRCTMP);
```

参考:IEC61131 規格は、プログラマブルコントローラ(PLC)の国際標準規格として International Electro-technical Commission(IEC) が定めたものです。7つの Part に分かれており、この中の Part3 プログラム言語(IEC 61131-3)で、PLC のプログラミングについて規定しています。JIS B3503は、IEC 61131-3を翻訳し日本工業規格としたものです。

2. 作成するプログラムの説明

ここでは、装置の中でワークの厚さの3箇所を測定を行うためのファンクションブロック内のプログラムをSTで作成します。



変数には、格納されるデータの型を設定する必要があります。
REAL型とは、32bit長の実数で、
 $-3.402823 \times 10^{38} \sim -1.175494 \times 10^{-38}$ 、0、
 $+1.175494 \times 10^{-38} \sim +3.402823 \times 10^{38}$
 が表現できます。

FB定義名 平均値算出(3値)

入力変数 **X**(REAL型), **Y**(REAL型), **Z**(REAL型)

出力変数 **score**(REAL型)

STプログラム定義 **score** := (**X** + **Y** + **Z**) / 3.0;

変数への代入は“:=”と表現します。

式の右端は“;”を入力します。

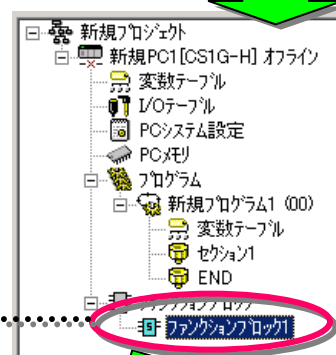
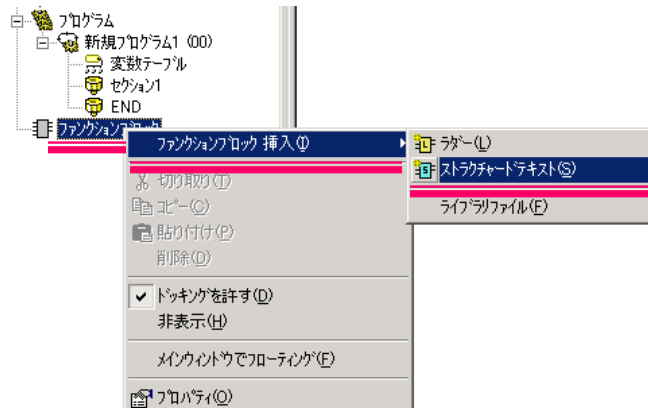
プログラムの
説明FB定義の
新規作成

変数の入力

FB内プログラム
の作成ラダープログラム
の作成 & チェック

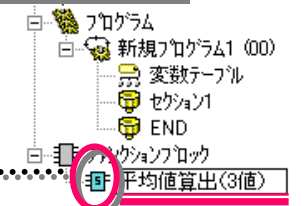
3. ST定義FBの作成

ST定義FBを作成します。



新規FBが生成されました。

FB定義名を変更します

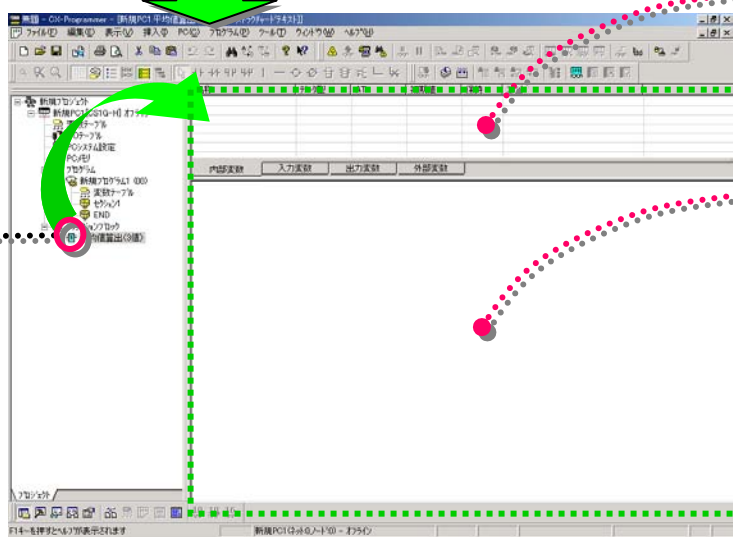


ご注意
“_”で始まるファンクションブロック定義は
ユーザは作成できません。
必ず“_”以外の文字で始まる名前にしてください。

マウスカーソルを追加された
ファンクションブロックアイコン
[平均値算出(3値)]に合わせ、マウス右クリック
→名前の変更(R)
[平均値算出(3値)]と入力
します。

ファンクションブロック
STエディタを開きます。

マウスカーソルをファンクション
ブロックアイコン [平均値算出(3値)]に合わせ、
マウス左ダブルクリックして、
ファンクションブロックSTエディ
タを開きます。



変数テーブル

ST入力画面

プログラムの
説明

FB定義の
新規作成

変数の入力

FB内プログラム
の作成

ラダープログラム
の作成 & チェック

4. ファンクションブロック内変数の登録

変数テーブルを選択します。

名称	データ型	AT	初期値	保持	コメント
EN	BOOL		FALSE		ファンクションブロック 実行制御フラグ

マウスカーソルを変数テーブル
の入力変数タブに合わせマウス
を左クリックします。

プルダウンメニューから
変数追加(I)を選択します。

名称(N)
データ型(D)
コメント(C)
を入力します。

新規変数

名称(N): x

データ型(D): REAL

用途(U): I: 入力

初期値(V): 0

保持(R): ☐

コメント(C): 入力値1

OK

キャンセル

詳細(A)...

追加する変数名称を入力

実数データなので
REALを選択

適切なコメントを入力

上記の作業を繰り返して、入力変数 x, y, z 出力変数scoreを入力します。

名称	データ型	AT	初期値	保持	コメント
EN	BOOL		FALSE		ファンクションブロック 実行制御フラグ
x	REAL		0.0		入力値1
y	REAL		0.0		入力値2
z	REAL		0.0		入力値3

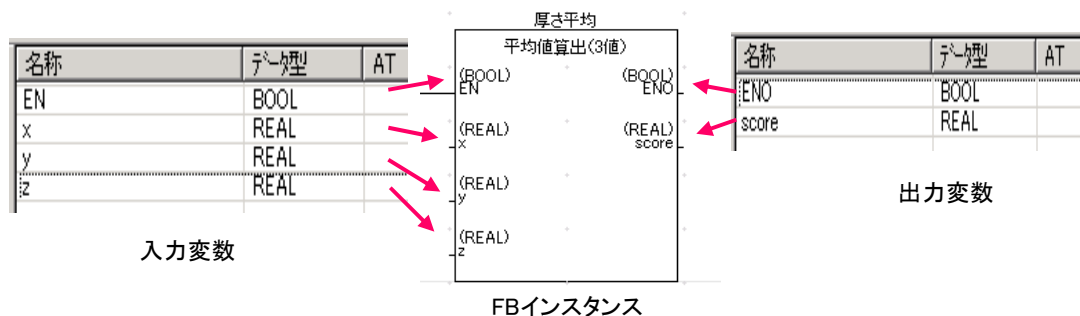
入力変数

名称	データ型	AT	初期値	保持	コメント
ENO	BOOL		FALSE		ファンクションブロック 実行結果フラグ(ファン...
score	REAL		0.0		平均値

出力変数

ご参考: 変数テーブル内では変数のコピー & ペーストが可能です。

ご参考: 変数の順序は、FBインスタンスでのパラメータの順序になります。
変数の順序を変更したい場合は、変数名称をマウスで選択し、
ドラッグ & ドロップ操作により可能です。



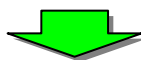
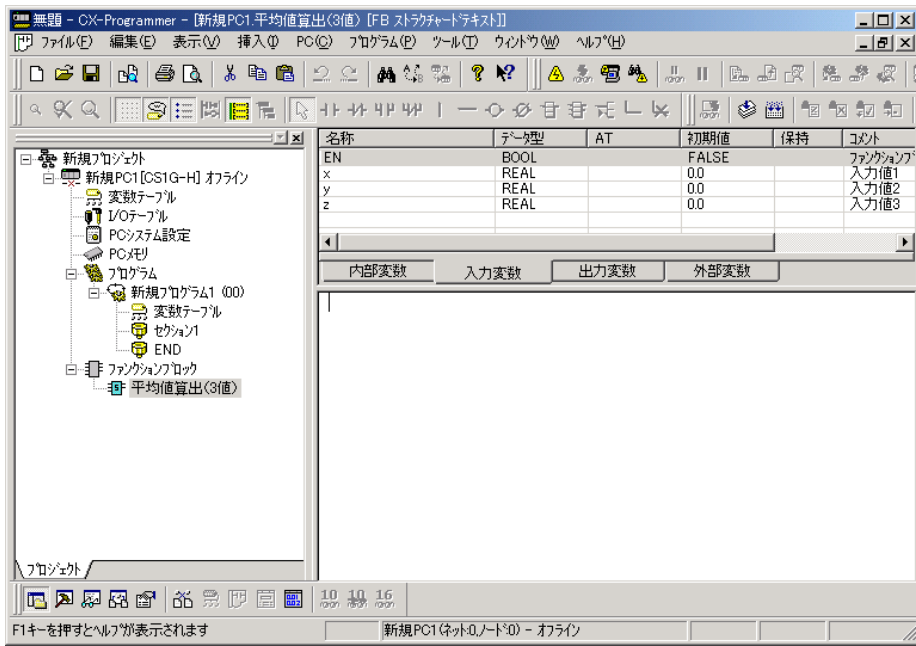
プログラムの
説明FB定義の
新規作成

変数の入力

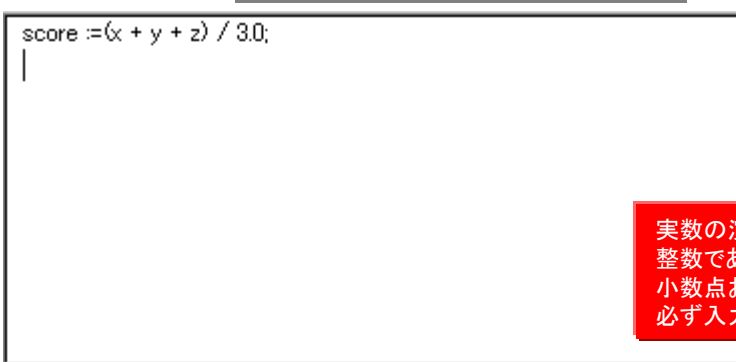
FB内プログラム
の作成ラダープログラムの
作成 & チェック

5. STプログラムの入力

ファンクションブロックSTエディタのST入力画面を選択します。



score := (x + y + z) / 3.0; を入力します。



実数の演算式の場合、定数が整数であっても“3.0”のように小数点および小数点第1位を必ず入力してください。

ご参考: STプログラム内には自由にコメントをつけることが可能です。
文字列の先頭と終端にそれぞれ、(*,*) を入力することでコメントとして扱われます。
プログラムの変更履歴、処理の説明の記述などに活用できます。

(* 作成 04.5.21 設計課 鈴木 *)

score := (x + y + z) / 3.0;

下の階層へ(D)

切り取り(T)

コピー(C)

貼り付け(P)

検索(F...)

置換(R...)

コメントアウト(O)

STヘルプ(H)

ご参考: STエディタにおけるポップアップメニューにて[STヘルプ]を選択すると、STの制御構文の例を示したヘルプピックヘジャンプすることができます。

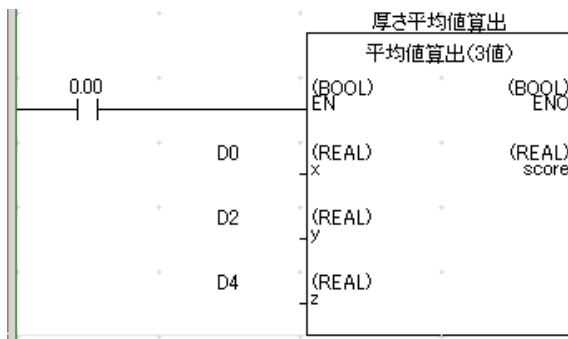
プログラムの
説明FB定義の
新規作成

変数の入力

FB内プログラム
の作成ラダープログラム
の作成&チェック

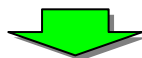
6. プログラムの作成とエラーチェック

以下のラダープログラムを入力します。
 インスタンス名: 厚さ平均値算出
 入力パラメータ: D0, D2, D4
 出力パラメータ: D6

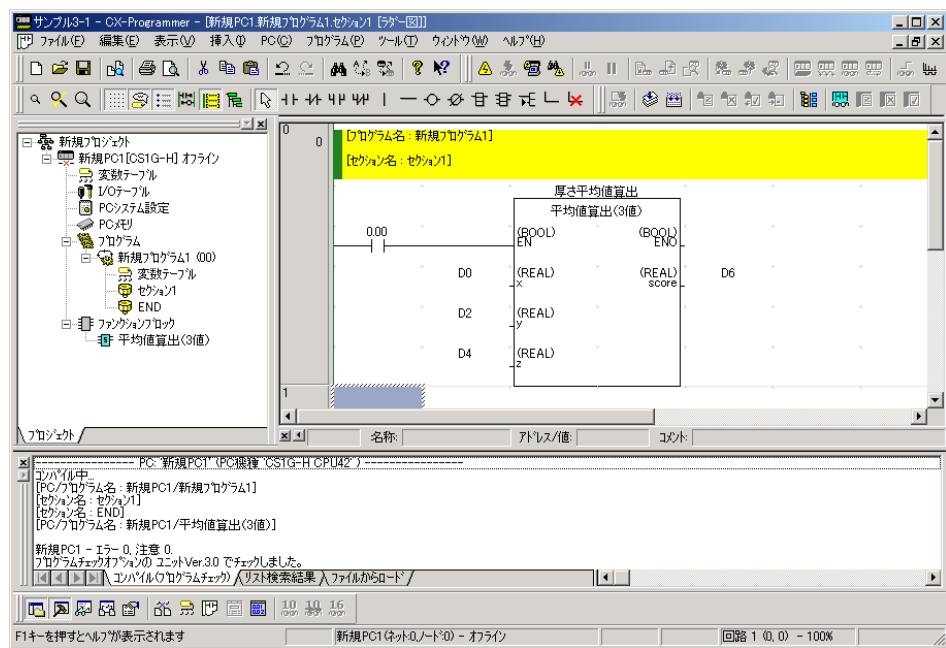


FBインスタンスにカーソルがあるとき、
 [Shift]+[F]キーを押すことにより該当
 するファンクションブロック定義にジャンプ
 できます。

インスタンスの入力方法は、
 2-7ページを参考にしてください。
 ファンクションブロックがラダーでも
 STでも入力方法は同じです。



プログラムを転送する前にエラーのチェックを行います。



エラーチェックの方法は、2-9ページを参考にしてください。
 ファンクションブロックがラダーでもSTでもチェック方法は同じです。

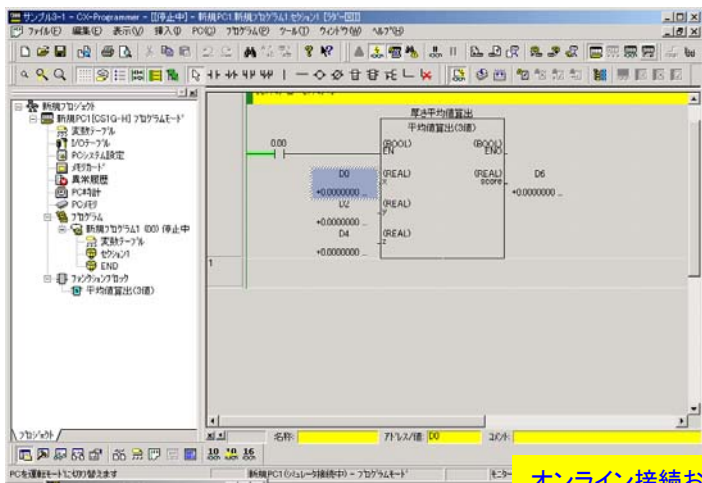
インスタンスの入力後でもファンクションブロック内の変数の追加・変更は可能ですが、
 パラメータが変わったことを通知するためにインスタンスを含む回路の右母線が赤色に
 変わります。
 このような場合、そのインスタンスをマウスカーソルで選択し、プルダウンメニューの
 ファンクションブロック呼び出しの更新(U)を実行してください。

プログラムの
転送

導通モニタ

7. プログラムの転送

CX-Simulatorとオンライン接続し、プログラムを転送します。



オンライン接続およびプログラムの転送の方法は、2-10ページを参考にしてください。

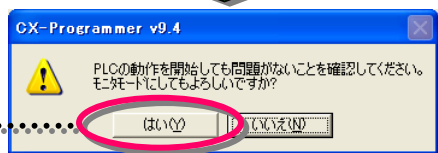
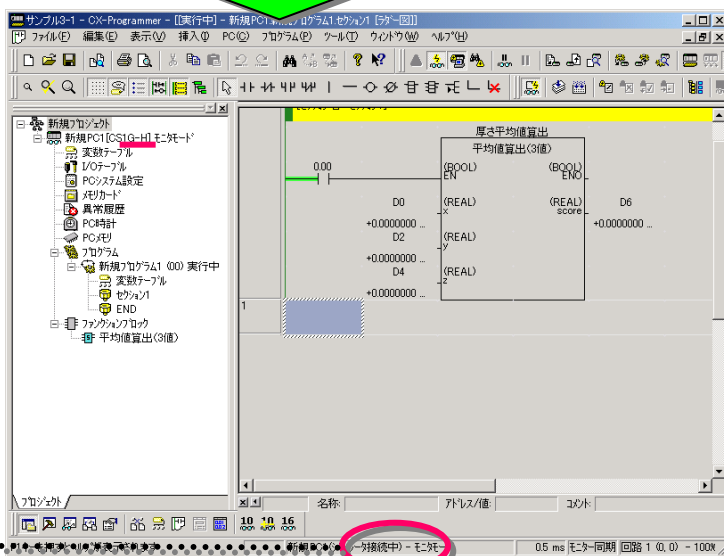
PLC(シミュレータ)を
モニタモードにします。

をクリック

接点、コイル等の ON/OFF状態をモニタします。



[はい]をクリック

PLC(シミュレータ)が
モニタモードになった
ことを、確認します。

プログラムの
転送

導通モニタ

8. 動作状態のモニタ

インスタンスのパラメータの値をウォッチウィンドウでモニタします。

ウォッチウィンドウを表示
させます。

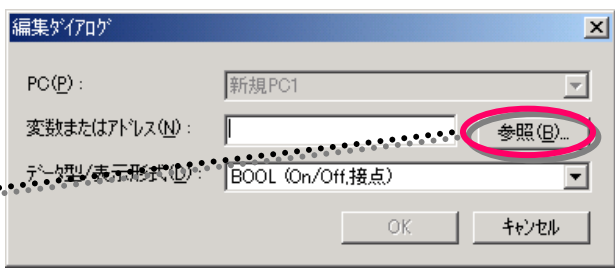
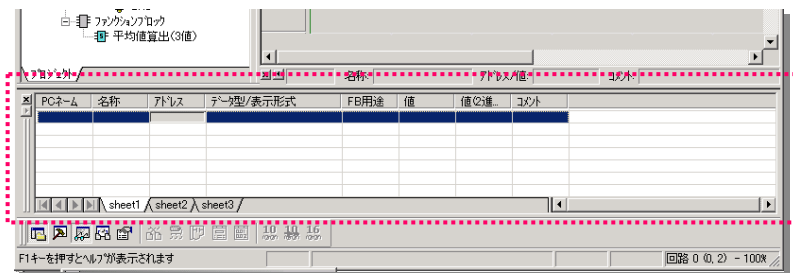
Alt + 3

編集ダイアログを開きます。

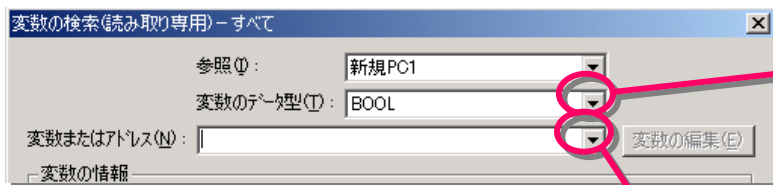
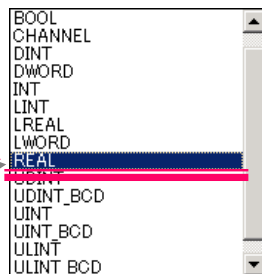
ENT

参照(B) ボタンをマウス
左クリック

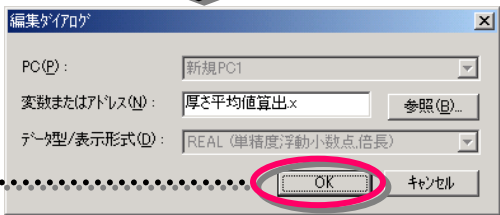
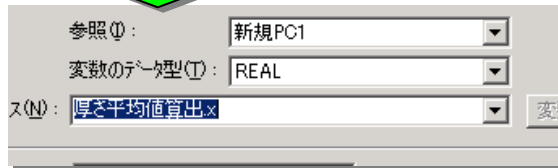
▼ ボタンを、マウス左クリックして、プルダウンメニューから参照したい変数の「変数のデータ型」「変数またはアドレス」をそれぞれ選択します。

OK ボタンをマウス
左クリック

REAL(32bit実数)を選択



厚さ平均値算出 x を選択



デバッグ時に内部変数をモニタする場合
このような操作でウォッチウィンドウに個別に
登録する他、一括で登録することができます。
詳しくは、5-13ページ「5-8. Watch Window
への一括登録」を参照してください。
FBがラダーの場合は導通モニタが可能です。
詳しくは、5-11ページ「5-5. 動作確認-1」を
参照してください。

PCネーム	名称	アドレス	データ型/表示形式	FB用途	値	値の進数	コスト
新規PC1	厚さ平均値算出 x	H522	REAL (単精度浮動小数...	I: 入力	+0.00000000...	+0.00000000 ...	入力値1

参考:IF-THEN-ELSE-END_IFを用いたSTプログラムの例

4-7ページまでのFB定義の例で算出された平均値が、上限値／下限値の範囲にあるか否かチェックを行うためのファンクションブロックをSTで作成すると以下ようになります。

FB定義名:判定結果出力

入力変数: score(REAL型)、setover(REAL型)、setunder(REAL型)

出力変数: OK (BOOL型)、overNG(BOOL型)、underNG(BOOL型)

STプログラム:

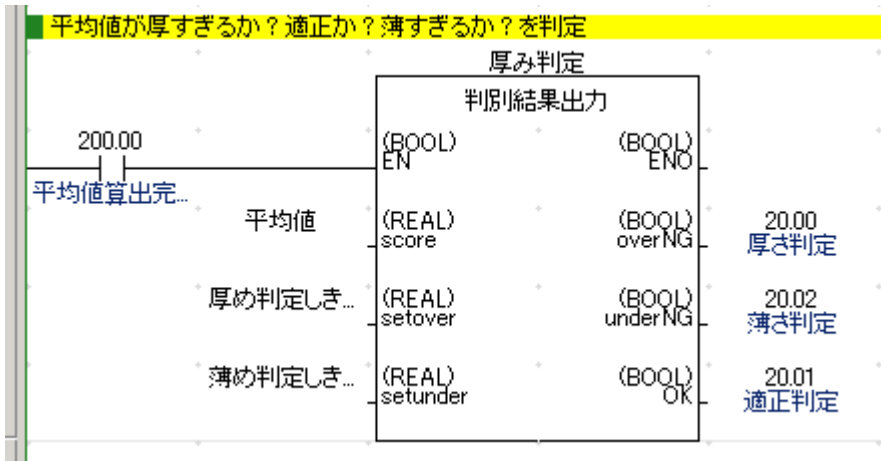
```
IF score > setover THEN      (* score > setover ならば *)
    underNG := FALSE;        (* underNG を OFF *)
    OK := FALSE;             (* OK を OFF *)
    overNG := TRUE;          (* overNG を ON *)

ELSIF score < setunder THEN  (* score =< setover で、score < setunder ならば *)
    overNG := FALSE;         (* overNG を ON *)
    OK := FALSE;             (* OK を OFF *)
    underNG := TRUE;         (* underNG を ON *)

ELSE                          (* setover > score > setunder ならば *)
    underNG := FALSE;        (* underNG を OFF *)
    overNG := FALSE;         (* overNG を OFF *)
    OK := TRUE;              (* OK を OFF *)

END_IF;                      (* IF 分の終了 *)
```

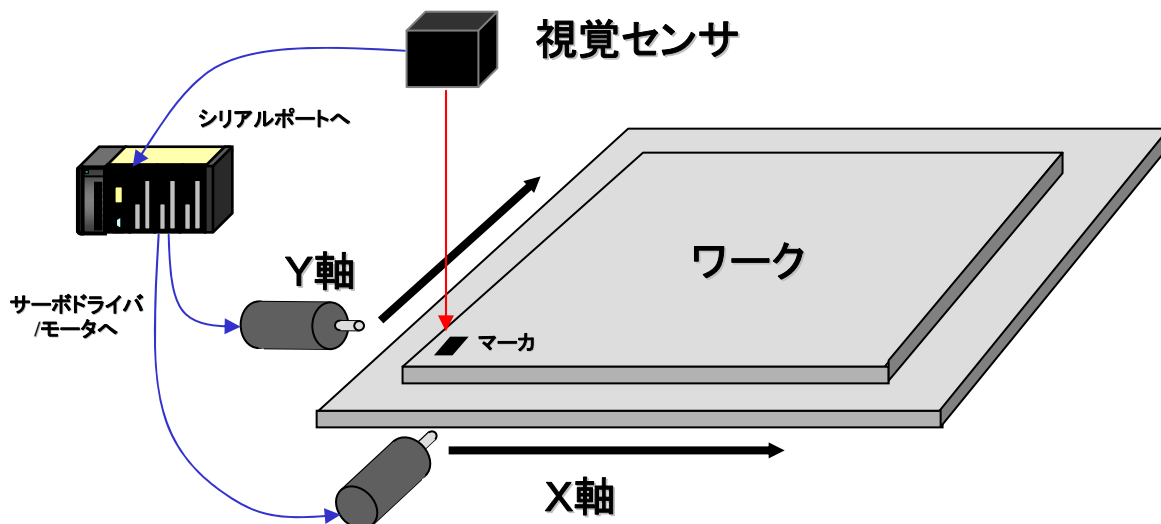
FBインスタンスの表示例(インスタンス名は「厚み判定」としています)



参考:文字列型変数を用いたSTプログラムの例

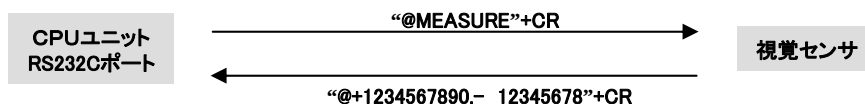
1. アプリケーション例

視覚センサによりワークの位置を検出し、サーボモータでXY軸の位置制御を行うアプリケーションを想定します。

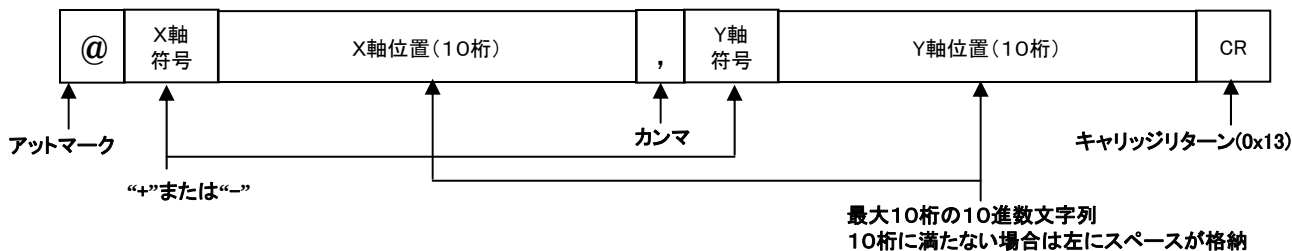


2. 視覚センサとのインタフェース

視覚センサとはCPUユニットのRS232Cポートを介し以下の通信メッセージを送受信します。



CPUユニットのRS232Cポートから“MEASURE”+CR(0x13)を送信すると、視覚センサは受信後、以下のフォーマットの文字列データとして送信します。



3. FB(ST)で作成する範囲(動作仕様)

FBで作成する範囲としては、以下とします。

X座標、Y座標をNCユニットの指令値としてそれぞれ2ワードのデータをセットします。

視覚センサからワークの現在値(X座標、Y座標)情報を含むデータをシリアル通信で受信します。

視覚センサから受信したデータを解析し、ワークの現在値(X座標、Y座標)情報を取得します。

ワークの目標値と現在値(X座標、Y座標)情報の差をNCユニットの指令値として出力します。

4. FB(ST)のプログラム

要件を満たすSTプログラムとしては以下のとおりです。

変数テーブル

変数タイプ	名称	データ型	AT	初期値	保持	コメント
内部変数	bSending	BOOL		FALSE		送信中フラグ
内部変数	nSendStatus	INT		0		送信状態ステータス(1: 送信可能, 2: 送信中, 3: 送信完了)
内部変数	SndEnableCPUPort	BOOL	A392.05			内蔵上位リンクポート送信レディフラグ
内部変数	EndRecvCPUPort	BOOL	A392.06			内蔵上位リンクポート受信完了フラグ
内部変数	strXYPosition	STRING(30)				視覚センサから読み取った文字列
内部変数	strXPos	STRING(15)				X軸現在値 テンポラリ変数
内部変数	strYPos	STRING(15)				Y軸現在値 テンポラリ変数
内部変数	nLen	INT		0		受信データ分解用のテンポラリ変数
内部変数	nCommaPos	INT		0		受信データ分解用カンマ位置
入力変数	bStartFlag	BOOL		FALSE		送信開始フラグ
入力変数	nXTargetPos	DINT		100		X軸目標値
入力変数	nYTargetPos	DINT		100		Y軸目標値
出力変数	nXDiff	DINT		0		X軸指令値
出力変数	nYDiff	DINT		0		Y軸指令値

STプログラム

```

(* 視覚センサから位置情報を読み込み、NCユニットへの指令値を生成する。
視覚センサから読込まれる文字列フォーマット: '(X座標)(区切り文字)(Y座標)'
      X座標 : 符号 + 最大10桁
      Y座標 : 符号 + 最大10桁
      区切り文字 : カンマ
例: '+1234567890,-654321' (X軸、Y軸の桁数は可変。) *)
(* 読出開始トリガの検出 *)
IF ( bStartFlag AND NOT(bBusy) ) THEN
    nStatus := 1;
    (* 既に読出中のときは実行しない*)
END_IF;

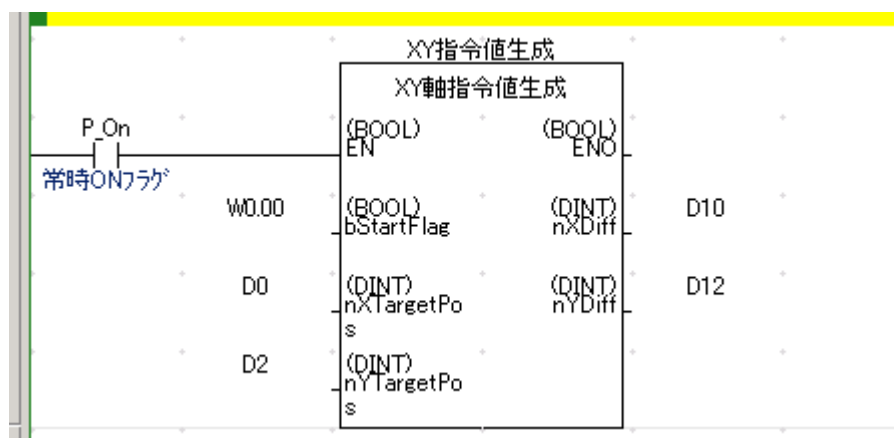
(* 読出処理*)
CASE nStatus OF
    1: (* バーコードリーダーへ読取指示*)
        IF SndEnableCPUPort = TRUE THEN
            (* RS232Cポートが送信可能状態ならば送信 *)
            bBusy := TRUE; (* 視覚センサ読出中フラグをON*)
            TXD_CPU('MEASURE'); (* "1回計測"コマンドを発行する*)
            nStatus := 2;
        END_IF;
    2: (* バーコードリーダーから読取データ取得 *)
        IF EndRecvCPUPort = TRUE THEN
            (* 受信完了フラグがONしたときに*)
            RXD_CPU(strXYPosition, 25); (* 受信データをstrXYPositionに読込む *)
            nStatus := 3;
        END_IF;
    3: (* 読出後処理*)
        (* 視覚センサから読み取った文字列をX座標・Y座標に分解する *)
        nLen := LEN(strXYPosition); (* 文字列の長さ*)
        nCommaPos := FIND(strXYPosition, ','); (* 区切り文字の位置*)
        strXPos := LEFT(strXYPosition, nCommaPos - 1);
        (* X座標の文字列を抽出*)
        strYPos := MID(strXYPosition, nCommaPos + 1, nLen - nCommaPos);
        (* Y座標の文字列を抽出*)
        (* 文字列を数値に変換し、指令値を算出 *)
        nXDiff := nXTargetPos - STRING_TO_DINT(strXPos);
        (* 指令値 := 目標値 - 現在値*)
        nYDiff := nYTargetPos - STRING_TO_DINT(strYPos);
        (* 指令値 := 目標値 - 現在値*)
        nStatus := 0;
        bBusy := FALSE; (* 視覚センサ読出中フラグをOFF*)
END_CASE;

```

5. ラダープログラムでの使用例

ラダープログラムでの使用例は以下のとおりです。

X軸、Y軸の目標値をD0、D2にセットして接点W0.0をONすると、FB内で通信を行い、D10、D12に指令値が出力されます。



第5章

応用編（FBによるプログラムの部品化）

Function Block

プログラムの
設計FB定義の
入力・デバッグFB定義の
ライブラリ化メインプログラムの
入力メインプログラムの
デバッグ

1. 概要

この章では、ファンクションブロックを用いたアプリケーションの例を用いて、ユーザプログラムの部品化の方法について説明します。

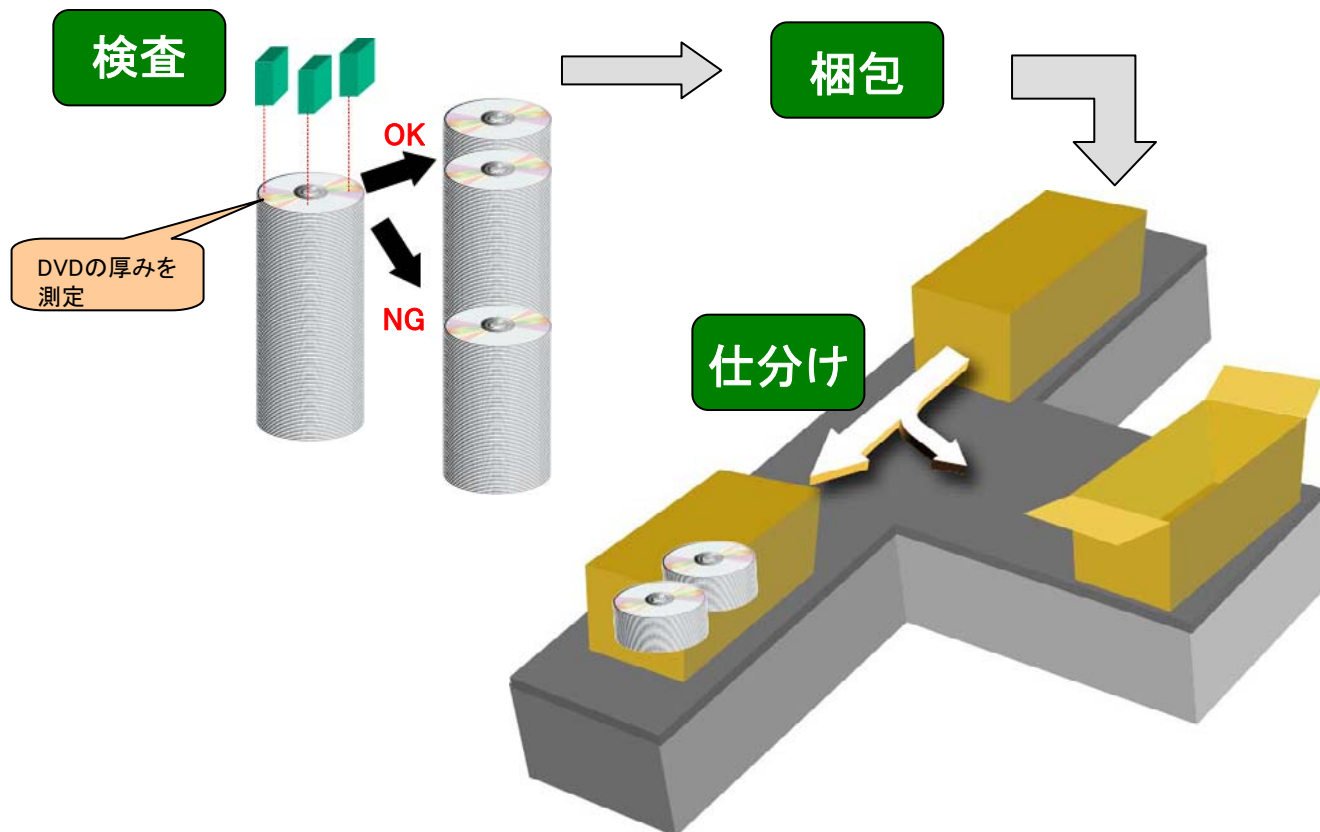
2. プログラム開発の進め方

ユーザプログラムを、部品化を進めながら作成していく場合、本章での例「3. アプリケーション例」においては一般的に以下のような流れとなります。特に、プログラムの設計工程で十分な検討を行う必要があります。

- (1)プログラムの設計
- (2)部品の作成
 - (2-1)FB部品の入力
 - (2-2)FB部品のデバッグ
 - (2-3)FB部品のライブラリ化(ファイルセーブ)
- (3)アプリケーションでの部品の使用
 - (3-1)部品をインポート
 - (3-2)プログラムで部品を使用する
 - (3-3)プログラムでデバッグする
- (4)立ち上げ

3. アプリケーション例

アプリケーションの例として、DVD検査装置を用います。
主に、検査工程、梱包工程、仕分け工程に分かれます。



プログラムの
設計FB定義の
入力・デバッグFB定義の
ライブラリ化メインプログラムの
入力メインプログラムの
デバッグ

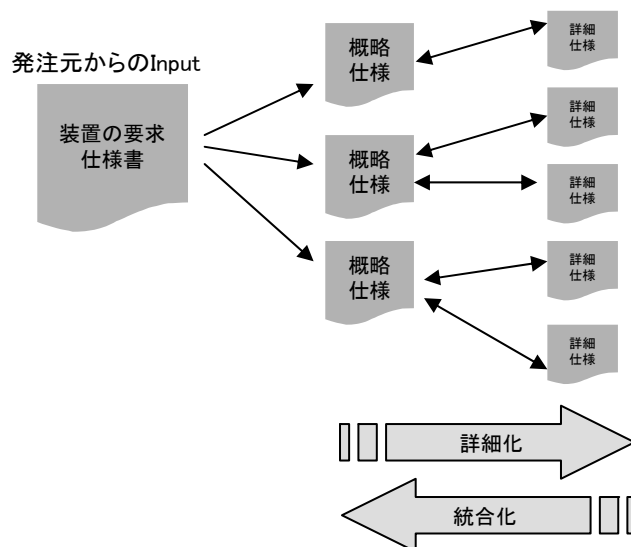
4. プログラム設計の進め方

アプリケーションは、さまざまな要求仕様を装置のハードウェアおよびソフトウェア(プログラム)を用いて実現し、その要求仕様を組み合わせることで実現します。

これより、前記のアプリケーション例を用い、どのような考え方でプログラムの設計を進めていくかを説明します。

4-1. 設計工程の概要

右のイメージのように、仕様の詳細化と統合化を繰り返して仕様の分割・整理を行っていきます。



4-2. 要求仕様の抽出

アプリケーション例の要求仕様を書き出すと以下のようになります。

DVD検査装置の概要(要求仕様)

要求1. ローダからDVDが挿入される。

要求2. DVDの厚さを3点測定する。測定した厚さの平均値を求め、閾値の範囲内であれば、良品のストックへ、範囲外であれば不良品のストックへ仕分けする。

要求3. 良品のDVDをケースに詰める。

要求4. ケース詰めされたDVDを紙箱に詰める。

要求5. 紙箱を2種類に選別する。選別部のアクチュエータに隣接するリミットスイッチの寿命判断のため開閉回数をカウントする。

要求6. その他の要求

※説明を簡単にするために、本書では装置の一部の処理(下線部)に焦点をあてて検討していきます。

プログラムの
設計FB定義の
入力・デバッグFB定義の
ライブラリ化メインプログラムの
入力メインプログラムの
デバッグ

4-3. 仕様の詳細化と類似処理の抽出

仕様を詳細化していくと、仕様のあちこちに類似の処理または汎用的に使えるような処理が見えてきます。

アクチュエータ制御(類似の処理の例)

この例では、良品・不良品の仕分けのためのシリンダ制御と、紙箱の選別のためのアクチュエータの制御は同じものとあつかうことができます。これらの処理の要件を挙げると以下ようになります。

- ・双方向の動作を行うためのアクチュエータを2つ持ち、それぞれの入力条件により動作する。
- ・各方向の動作にはインタロックをかける。
- ・タイマによりアクチュエータの動作時間を監視し、一定時間を超えた場合はエラー信号をONする。
- ・動作のリセットを行う入力信号を持つ。

平均値_閾値チェック(汎用的に使える処理の例)

新規作成時には1度しか使わなくとも、将来使える可能性のある処理を抽出します。この例では、DVDの厚さを3点測定した結果から平均値を求め、閾値内か否かをチェックする処理を挙げます。この処理の要件を挙げると以下ようになります。

- ・3つの値の平均値を求める。
- ・平均値が閾値の上限、下限の範囲内に収まっているか否かを比較する。

これらの要件が、それぞれの部品の仕様の元となります。部品の名前を、それぞれ「アクチュエータ制御」FB、「平均値_閾値チェック」FBとします。

4-3-1. 部品の仕様書の作成について

部品は、繰り返し再利用を行うことでプログラム開発の生産性を向上させることができます。再利用を簡単に行うためには、部品の内部を見なくても簡単に入出力仕様、動作仕様がわかるような仕様書を作成することやコメントを入力しておくことが重要です。

Smart FBライブラリのライブラリリファレンスをご参考に記述しておくことをお勧めします。

プログラムの
設計FB定義の
入力・デバッグFB定義の
ライブラリ化メインプログラムの
入力メインプログラムの
デバッグ

4-3-2. 各FBの作成例

「アクチュエータ制御」FB —— シーケンス制御向けの処理であるため、ラダーで記述します。

[入力変数]

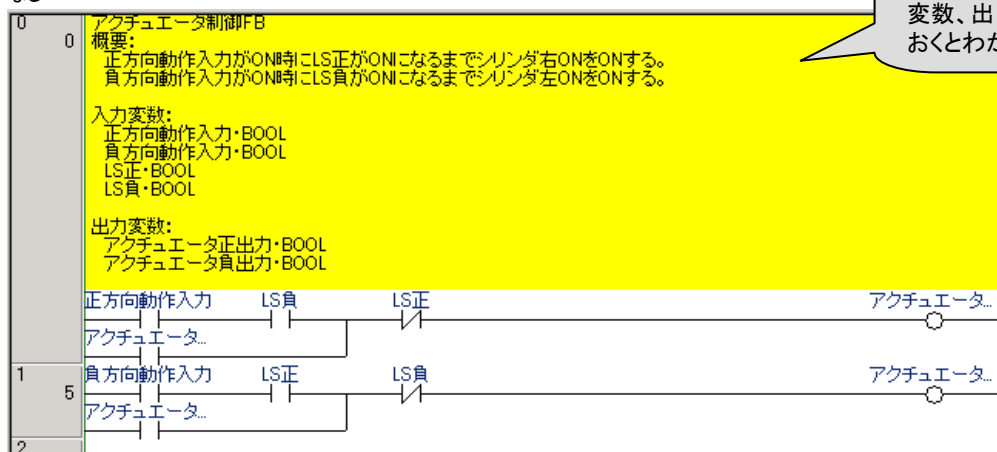
名称	データ型	AT	初期値	保持	コメント
EN	BOOL		FALSE		ファンクションブロック 実行制御フラグ
正方向動作入力	BOOL		FALSE		正方向動作入力
負方向動作入力	BOOL		FALSE		負方向動作入力
LS正	BOOL		FALSE		正方向リミットスイッチ
LS負	BOOL		FALSE		負方向リミットスイッチ

[出力変数]

名称	データ型	AT	初期値	保持	コメント
ENO	BOOL		FALSE		ファンクションブロック 実行結果フラグ (ファ...
アクチュエータ正出力	BOOL		FALSE		アクチュエータ正方向出力
アクチュエータ負出力	BOOL		FALSE		アクチュエータ負方向出力

[内部変数]

なし



行コメントに動作の概要、入力変数、出力変数などを書いておくとうわかりやすくなります。

「平均値・閾値チェック」FB —— 数値演算、比較の処理であるため、STで記述します。

[入力変数]

名称	データ型	AT	初期値	保持	コメント
EN	BOOL		FALSE		ファンクションブロック 実行制御フラグ
入力1	REAL		0.0		入力値1
入力2	REAL		0.0		入力値2
入力3	REAL		0.0		入力値3
上限	REAL		0.0		閾値の上限値
下限	REAL		0.0		閾値の下限値

[出力変数]

名称	データ型	AT	初期値	保持	コメント
ENO	BOOL		FALSE		ファンクションブロック 実行結果フラグ (ファ...
結果	BOOL		FALSE		OK/NG判定フラグ

[内部変数]

名称	データ型	AT	初期値	保持	コメント
平均値	REAL		0.0		

(* 3値 平均値算出 + 閾値判断*)

```

平均値 := (入力1 + 入力2 + 入力3) / 3.0;          (*入力3値の和を3で除算*)
IF ((平均値 <= 上限) AND (平均値 >= 下限)) THEN  (*平均値が上限・下限の間に入っているか比較*)
  結果 := TRUE;
ELSE
  結果 := FALSE;
END_IF;

```

ご参考: ラダー／ST共にFB名称および変数名は、作成時に使用する機能に特化した名前はず、できるだけ一般的な名前をすることをお勧めします。

ネスティングを行う場合は、内部変数に FUNCTION BLOCK(FB)型の変数を宣言することで、その変数名をインスタンスとして使用することができます。

プログラムの
設計FB定義の
入力・デバッグFB定義の
ライブラリ化メインプログラムの
入力メインプログラムの
デバッグ

4-4-2. 既存部品に機能を追加ーワーク移動制御_LSONカウント

要求5の「紙箱を2種類に選別する。選別部のアクチュエータに隣接するリミットスイッチの寿命判断のためリミットスイッチの開閉回数をカウントする。」は、「アクチュエータ制御」にて入力となるリミットスイッチのOFF⇒ONの回数をカウントすることで実現できます。この部品を「ワーク移動制御_LSONカウント」FBと呼びます。以下に作成するFBの例を示します。

[入力変数]

名称	データ型	AT	初期値	保持	コメント
EN	BOOL		FALSE		ファンクションブロック 実行制御フラグ
右方向動作入力	BOOL		FALSE		アクチュエータを右方向に移動する条...
左方向動作入力	BOOL		FALSE		アクチュエータを左方向に移動する条...
LS右	BOOL		FALSE		アクチュエータ右方向のリミットスイッチ
LS左	BOOL		FALSE		アクチュエータ左方向のリミットスイッチ
リセット	BOOL		FALSE		リセットON⇒OFFでリミットスイッチの...

[出力変数]

名称	データ型	AT	初期値	保持	コメント
ENO	BOOL		FALSE		ファンクションブロック 実行結果フラグ(ワ...
アクチュエータ右ON	BOOL		FALSE		アクチュエータ右方向動作指令
アクチュエータ左ON	BOOL		FALSE		アクチュエータ左方向動作指令
LS_ON回数	LINT		0		

[内部変数]

名称	データ型	AT	初期値	保持	コメント
前サイクルLS ワーク移動	BOOL FB [アクチュエータ制御]		FALSE		
(*ワーク移動制御 + リミットスイッチ開閉回数カウント *)					
(*作成者: 工機部 山田、作成日: 2005-01-10 *)					
(*リセットON⇒OFFでリミットスイッチの開閉回数をリセットする*)					
IF リセット = TRUE THEN					
前サイクルLS := FALSE;					
END_IF;					
(*アクチュエータ制御FB のインスタンスであるワーク移動 を呼び出す*)					
ワーク移動(右方向動作入力, 左方向動作入力, LS右, LS左, アクチュエータ右ON, アクチュエータ左ON);					
(* リミットスイッチ開閉回数カウント *)					
IF 前サイクルLS = FALSE and LS右 = TRUE THEN					
LS_ON回数 := LS_ON回数+1;					
END_IF;					
前サイクルLS := LS右; (* LS右の状態を前サイクルLSにコピー(次サイクルで比較するために) *)					

STからラダーFBをコールしています

STからFB(ファンクションブロック)をコールする場合の表記方法

呼び出されるFB: MyFB

呼び出されるFBの入出力変数:

入力: Input1, Input2

出力: Output1, Output2

ST内で宣言されたMyFBのインスタンス: MyInstance

ST内でFBに渡す入出力変数:

入力: STInput1, STInput2

出力: STOutput1, STOutput2

上記の場合、STからFBのインスタンスを呼び出す表記としては、

MyInstance(Input1 := STInput1, Input2 := STInput2, Output1 => STOutput1, Output2 => STOutput2);
となります。

入出力変数をすべて記述する場合は、呼び出される側の変数の記述と代入演算子を省略できます。

MyInstance(STInput1, STInput2, STOutput1, STOutput2);

呼び出される側の変数の記述と代入演算子を記述することで一部の入出力変数のみを記述することができます。

MyInstance(Input1 := STInput1, Output2 => STOutput2);

プログラムの
設計FB定義の
入力・デバッグFB定義の
ライブラリ化メインプログラムの
入力メインプログラムの
デバッグ

4-5. プログラム全体の記述

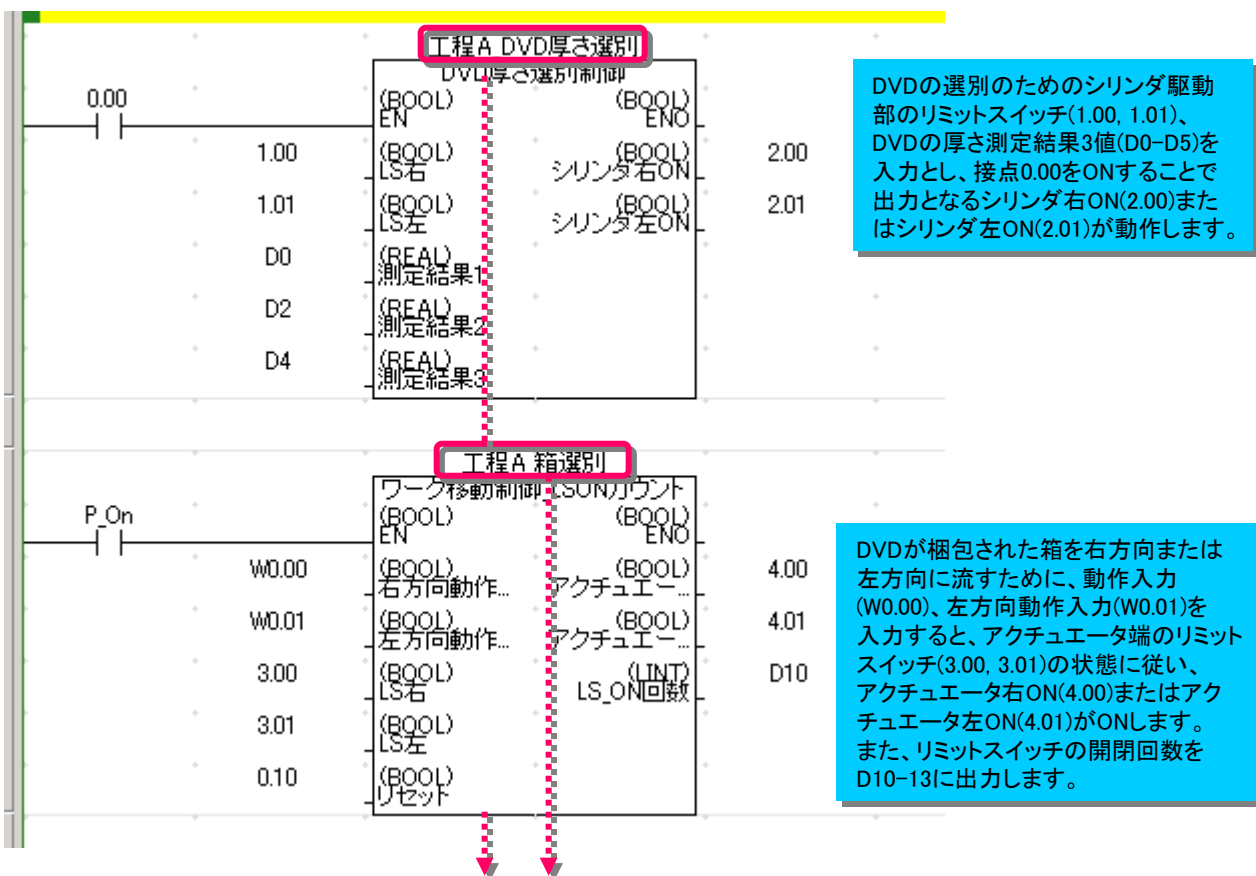
ここまで検討してきた部品 (FB) をプログラムとして動作させるためには、メインのラダープログラムから統合された部品を呼び出す回路を作成します。

※ ここでは要求2と要求5に限定します。

[グローバル変数]

名称	データ型	アドレス / 値	ラック位置	冊
工程A_箱選別	FB [ワーク移動制御_LSONカウント]	N/A [自動]		
工程A_DVD厚さ選別	FB [DVD厚さ選別制御]	N/A [自動]		

※FBを使用するためのインスタンス変数以外は省略しています。



インスタンス名を「工程A***」としている理由

アプリケーション例では明示していませんでしたが、インスタンスの使用場所(または用途)をインスタンス名の一部とすることで、たとえば、工程Bが増設された場合でも、ファンクションブロックを新たに登録する必要はなく、「工程B***」というインスタンスをプログラムに記述し、必要なパラメータを設定するだけで工程B用のプログラムが作成できます。オムロンのファンクションブロックの特徴として、1つのFBに複数のインスタンスを持つことができる、というものがありますが、FBの定義(アルゴリズム)を動作実績があるものを使うことで、アドレスの割り付けのみで同じ動作をするプログラムを作成することができます。

プログラムの
設計

FB定義の
入力・デバッグ

FB定義の
ライブラリ化

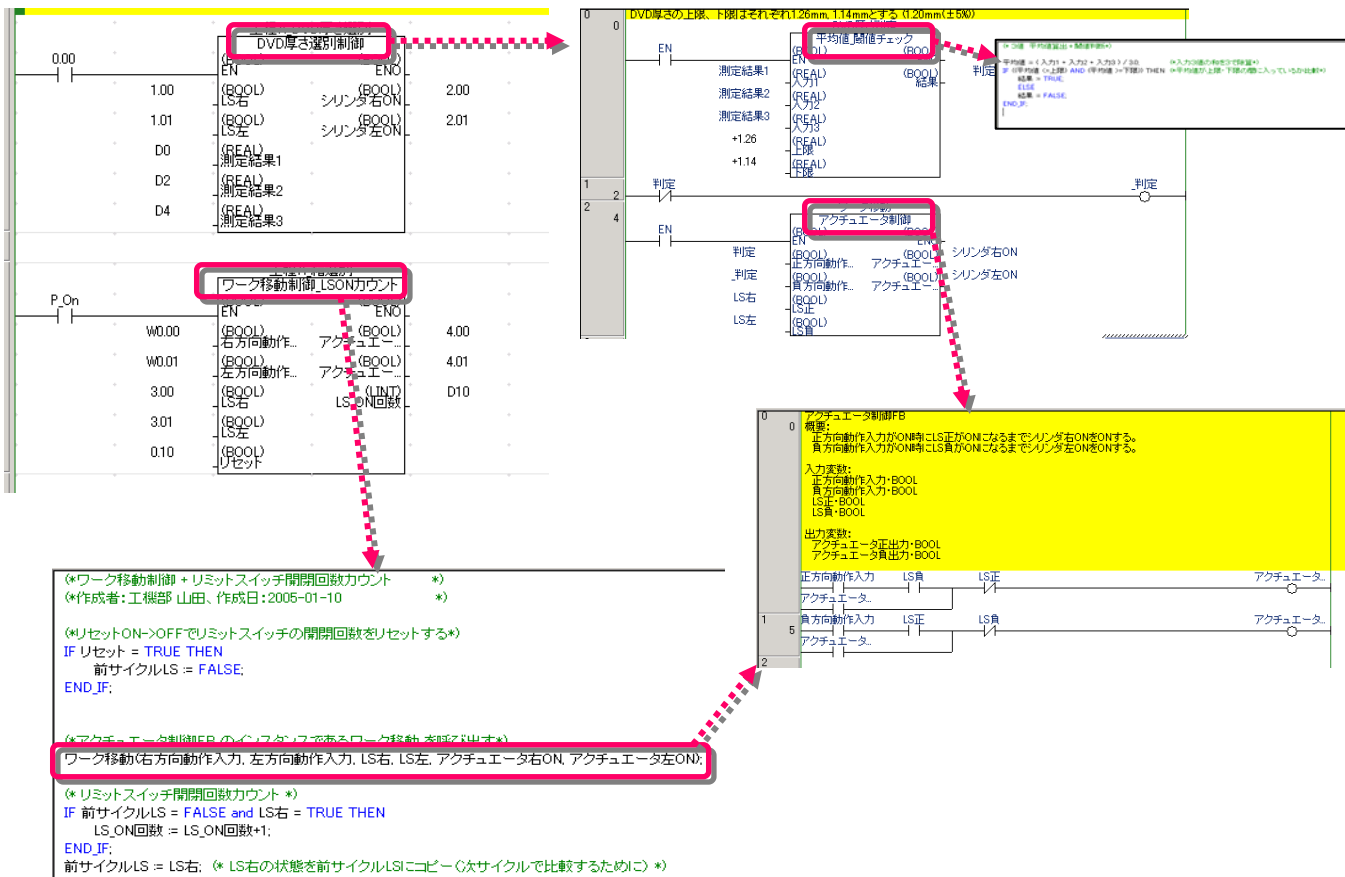
メインプログラムの
入力

メインプログラムの
デバッグ

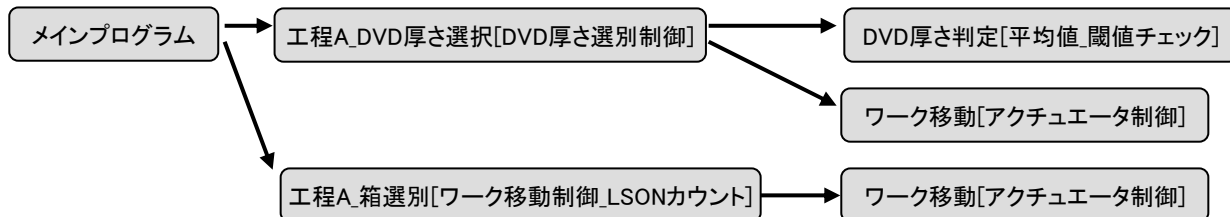
4-5-1. プログラム全体の構造

ここで作成した部品(ファンクションブロック)を含め、プログラム全体の構造を確認します。

[メインプログラム]



インスタンス名、FB名で整理すると以下のようになります。(FB名は[]で表現)



構造化されたプログラムでは、特に下位の部品(FB)の変更を行う場合、デバッグ時など処理の流れを把握する必要のある場合など、部品間の親子関係、部品の共有関係を理解することが重要になります。プログラム全体の構造を図などわかりやすい形で設計の資料として作成することをお勧めします。

CX-Programmer では [Alt]+[5]キーを押すことで「FBインスタンスビュー」を表示し、FBにより構造化されたソフトウェア構造を容易に把握することができます。また、FBインスタンスに割り付けられているアドレスを確認することもできます。

名称(N)	データ型(D)	アドレス(A)	コメント(C)
EN	BOOL	H531.00	ファンクションブロック 実行...
下限	REAL	H540	閾値の下限値
上限	REAL	H538	閾値の上限値
入力1	REAL	H532	入力値1
入力2	REAL	H534	入力値2
入力3	REAL	H536	入力値3

プログラムの
設計FB定義の
入力・デバッグFB定義の
ライブラリ化メインプログラムの
入力メインプログラムの
デバッグ

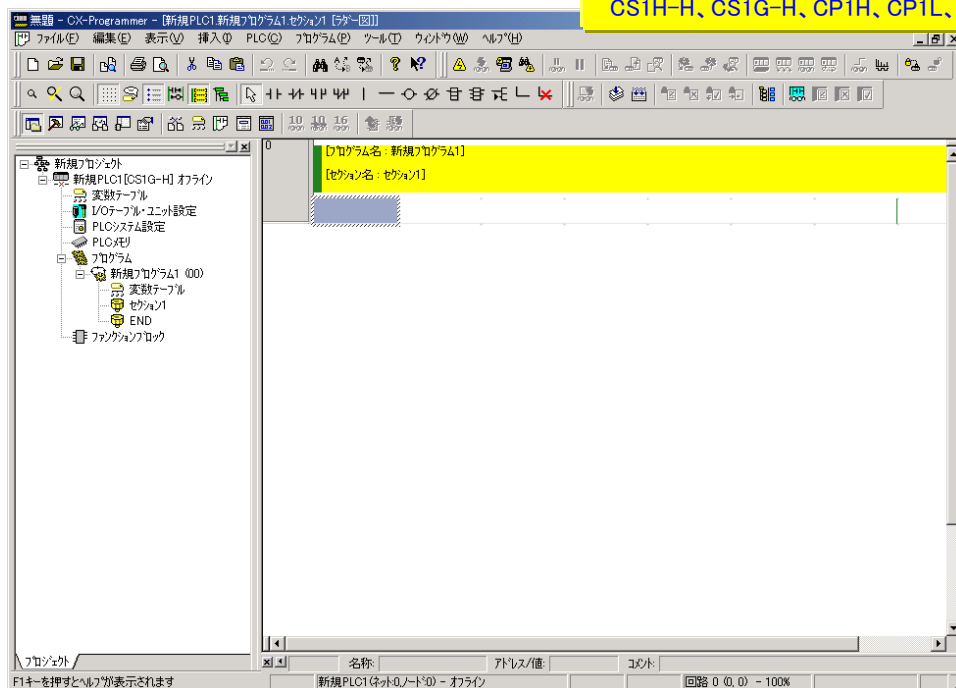
5. FB定義の入力

ここからは、実際に設計を行ったプログラムの入力、デバッグのための操作を説明します。
新規プロジェクトを作成し、5-4ページの「アクチュエータ制御」FBを入力します。

5-1. プロジェクトの新規作成をPLC機種/CPU形式の設定

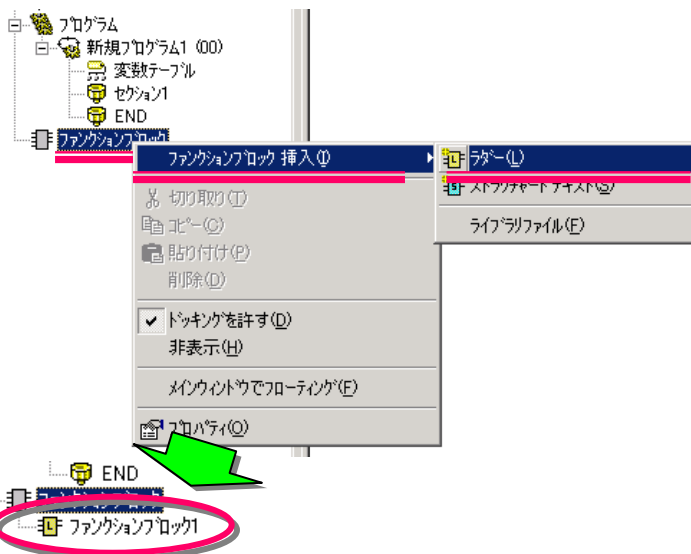
2-3ページを参考にして、プロジェクトの新規作成を行います。

！ファンクションブロックを使用するためには、
以下のPLC機種から選択して下さい。
CJ2H、CJ2M、CJ1H-H、CJ1G-H、CJ1M、
CS1H-H、CS1G-H、CP1H、CP1L、CP1L-E



5-2. ラダー定義FBの作成

ラダー定義FBを作成します。



マウскарソルをファンクション
ブロックアイコンに合わせ、
マウス右クリック
→ファンクションブロック挿入(I)
→ラダー(L)
を選択します。

新規FBが生成されました。.....

プログラムの
設計

FB定義の
入力・デバッグ

FB定義の
ライブラリ化

メインプログラムの
入力

メインプログラムの
デバッグ

5-3. FBラダープログラムの入力

FB定義名を変更します。

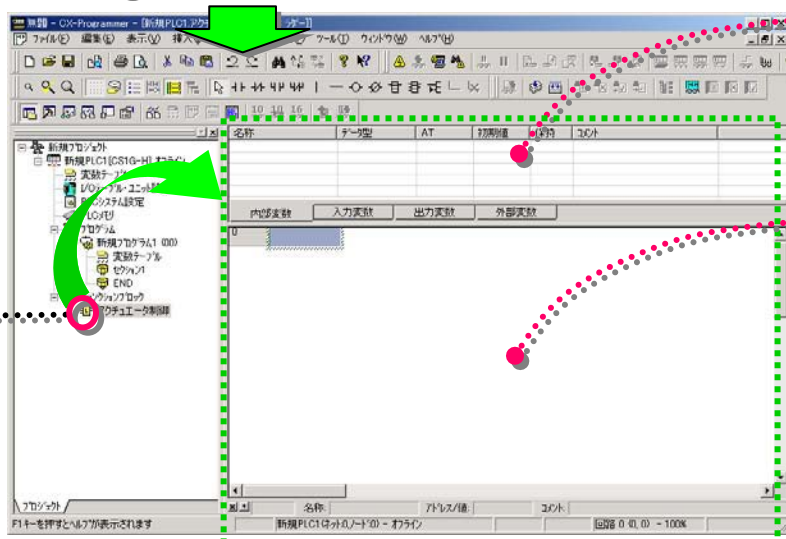


ご注意

“_”で始まるファンクションブロック定義はユーザは作成できません。必ず“_”以外の文字で始まる名前にしてください。

マウスカーソルをコピーされたファンクションブロックアイコンに合わせ、マウス右クリック→名前の変更(R)
[アクチュエータ制御]と入力します。

FBラダーエディタを開きます。



変数テーブル

ラダー入力画面

マウスカーソルをファンクションブロックアイコンに合わせ、マウス左ダブルクリックして、ファンクションブロックSTエディタを開きます。

変数テーブルを選択し、ファンクションブロック内変数の登録を行います。
登録する変数は、5-4ページの「アクチュエータ制御」FBの変数全てです。

ご参考: 変数の順序は、FBインスタンスでのパラメータの順序になります。
変数の順序を変更したい場合は、変数名称をマウスで選択し、ドラッグ&ドロップ操作により可能です。

ラダー入力画面を選択し、ラダープログラムを入力します。
入力するラダープログラムは、5-4ページの「アクチュエータ制御」FBの回路全てです。

ご参考: FBラダーエディタでもメインのラダーエディタと同様に回路の入力が可能ですが、FB内でのアドレスの入力は無効となりますのでご注意ください。

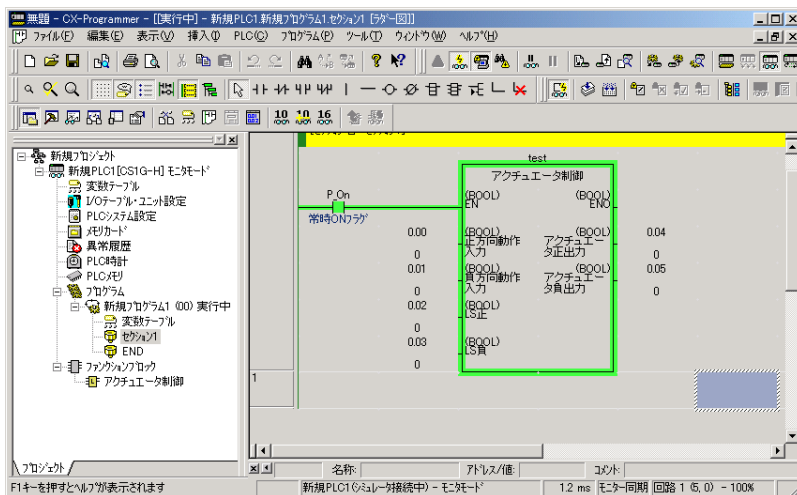
ご参考: 行コメントに変数一覧を入力する場合、変数テーブルの変数を選択し、コピーすることができます。入力の効率化にご活用ください。

プログラムの
設計FB定義の
入力・デバッグFB定義の
ライブラリ化メインプログラムの
入力メインプログラムの
デバッグ

5-4. プログラムの転送

CX-Simulatorとオンライン接続し、プログラムを転送後、PLC(シミュレータ)をモニタモードにします。

オンライン接続およびプログラムの転送の方法は、2-10ページを参考にしてください。

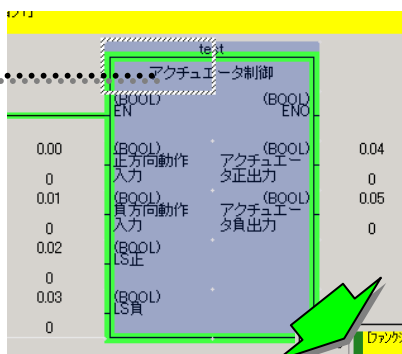


5-5. 動作確認 - 1

メインラダー上のFB呼び出し命令のパラメータの現在値を変更し、「アクチュエータ制御」FBの動作を確認します。

まず、アクチュエータ制御FBのインスタンスをモニタします。

カーソルをFB呼び出し命令内に移動し、マウスダブルクリックまたは、 ボタンをクリックします。



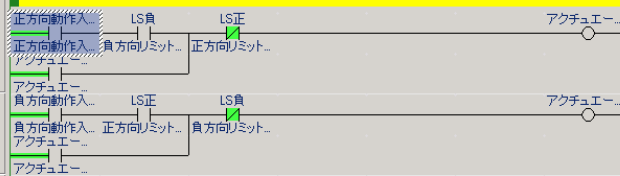
ファンクションブロック名: アクチュエータ制御

【インスタンス名: test】

アクチュエータ制御FB
概要:
正方向動作入力値がON時にLS正がONになるまでシリンダ正出力をONする。
負方向動作入力値がON時にLS負がONになるまでシリンダ負出力をONする。

入力変数:
正方向動作入力: BOOL
負方向動作入力: BOOL
LS正: BOOL
LS負: BOOL

出力変数:
アクチュエータ正出力: BOOL
アクチュエータ負出力: BOOL



FBラダーのインスタンス(アドレスが割り付けられている状態)が導通モニタされます。

プログラムの
設計

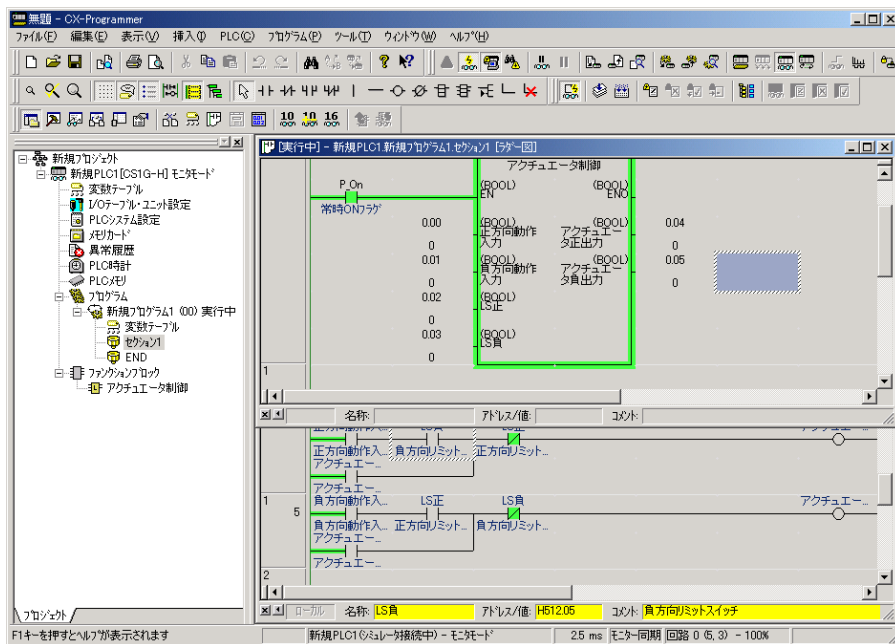
FB定義の
入力・デバッグ

FB定義の
ライブラリ化

メインプログラムの
入力

メインプログラムの
デバッグ

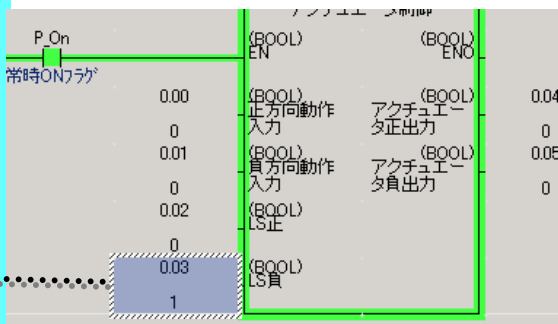
メインラダーとFBインスタンス(メインラダーから呼び出されているFBラダー)を同時に表示し、メインラダー内のFB呼び出し命令のパラメータの現在値を変更しながら動作を確認します。



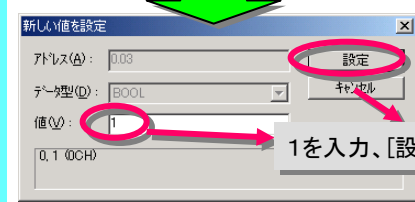
5-6. 動作確認-2

FB呼び出し命令のパラメータに以下のような入力を与え、期待する出力を得ることを確認します。この例では下記(1)のみ示しますが、実際は全ての条件の組み合わせに対する検証が必要です。

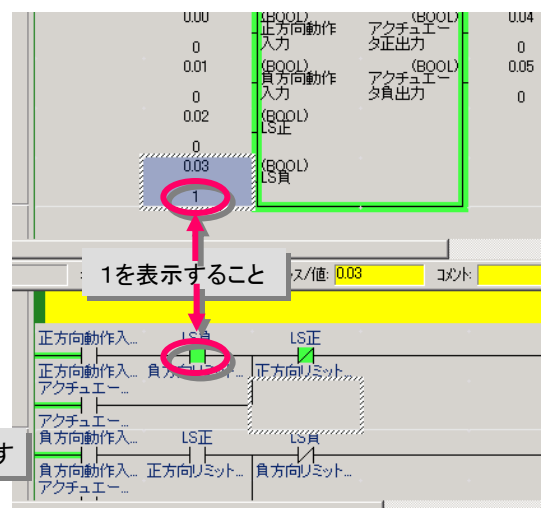
- (1)初期状態:0.03をONする。=>0.04, 0.05がOFFであること。FBインスタンスラダーモニタ画面が対応した状態になること。
- (2)アクチュエータ正方向動作-1:0.00をON=>0.04がONすること。FBインスタンスラダーモニタ画面が対応した状態になること。
- (3)アクチュエータ正方向動作-2:0.03をOFF=>0.04がON、0.05がOFFであること。FBインスタンスラダーモニタ画面が対応した状態になること。
- (4)アクチュエータ正方向動作-3:0.02をON=> 0.04がOFF、0.05がOFFであること。FBインスタンスラダーモニタ画面が対応した状態になること。



カーソルを0.03へ移動し、
[ENT]キーを押します。



1を入力、[設定]ボタンを押す



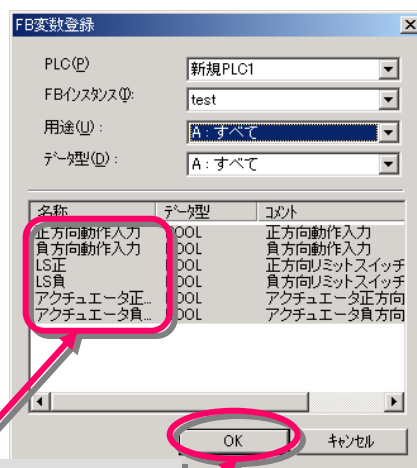
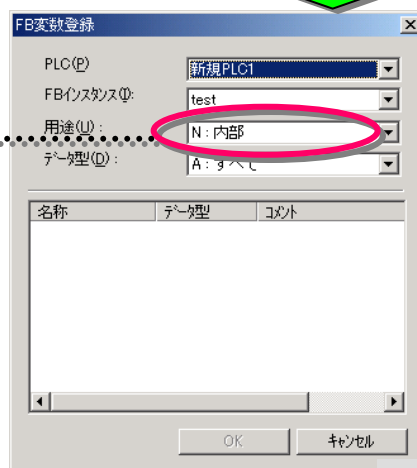
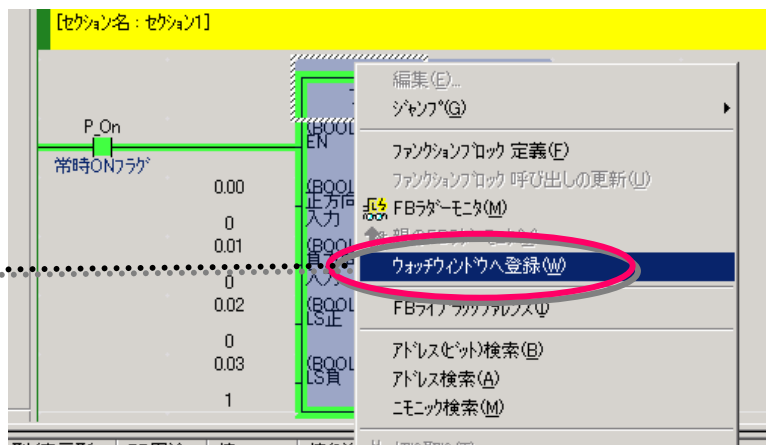
プログラムの
設計FB定義の
入力・デバッグFB定義の
ライブラリ化メインプログラムの
入力メインプログラムの
デバッグ

5-7. 他のFB定義の入力・デバッグ

ここまでは、「アクチュエータ制御」FBの入力、デバッグ方法について説明しましたが、その他のFB定義も同様に入力、十分なデバッグを実施してください。

5-8. Watch Windowへの一括登録

デバッグの手段としてFBラダーモニタの他、FBインスタンスのアドレスをWatch Windowへ一括して登録する機能があります。

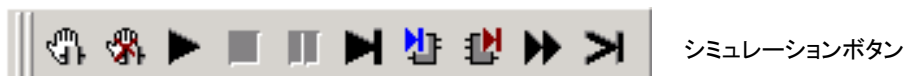


5-9. シミュレーション機能でのステップ実行

シミュレーション機能のブレークポイント設定、ステップ実行機能を用いると、回路の任意の位置でシミュレーション実行を一旦停止し、プログラムの実行途中での演算状況を簡単に確認することができます。

5-9-1. シミュレーションボタンの説明

シミュレーション機能を使用するためにツールバーの以下のボタンを使用します。ここでは、各ボタンの機能の説明をします。



	ブレークポイント設定 (F9 キー)	シミュレーション実行中に停止させたい箇所(ラダー、ST)を選択し、このボタンを押すと、赤色の印が表示されます。
	ブレークポイント解除	ブレークポイント設定ボタンで設定されたブレークポイント(赤色の印)を削除します。
	実行 (F8キー)	ユーザプログラムを実行します。運転モードはモニターモードになります。
	停止	ユーザプログラムの実行を停止します。運転モードはプログラムモードになります。
	一次停止	ユーザプログラムの実行をカーソル位置で一次停止します。
	ステップ実行 (F10キー)	ユーザプログラムを1ステップ実行します。 ラダーの場合は1命令、STの場合は1行実行します。
	ステップイン実行 (F11キー)	ユーザプログラムを1ステップ実行します。 カーソル位置がFB呼び出し命令の場合、呼び出されるFBインスタンス(ラダーまたはST)に遷移します。
	ステップアウト実行 (Shift + F11キー)	ユーザプログラムを1ステップ実行します。 カーソル位置がFBインスタンスの場合、呼び出し元のFB呼び出し命令に遷移します。
	連続ステップ実行	ユーザプログラムをステップ実行しますが、一定間隔をおいて自動的に連続してステップ実行します。
	スキャン実行	ユーザプログラムを1スキャン(1サイクル)実行します。

プログラムの
設計

FB定義の
入力・デバッグ

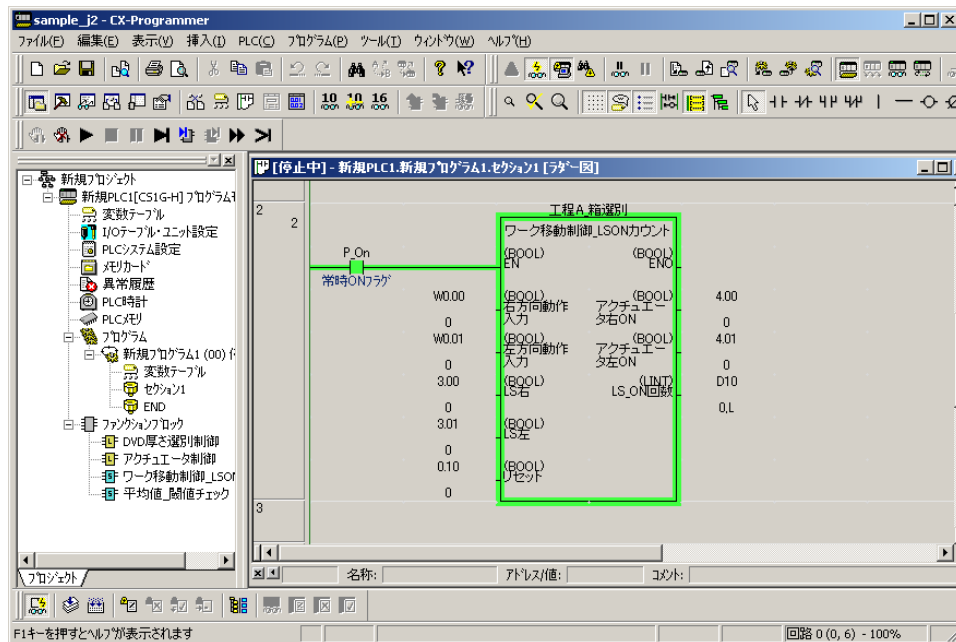
FB定義の
ライブラリ化

メインプログラムの
入力

メインプログラムの
デバッグ

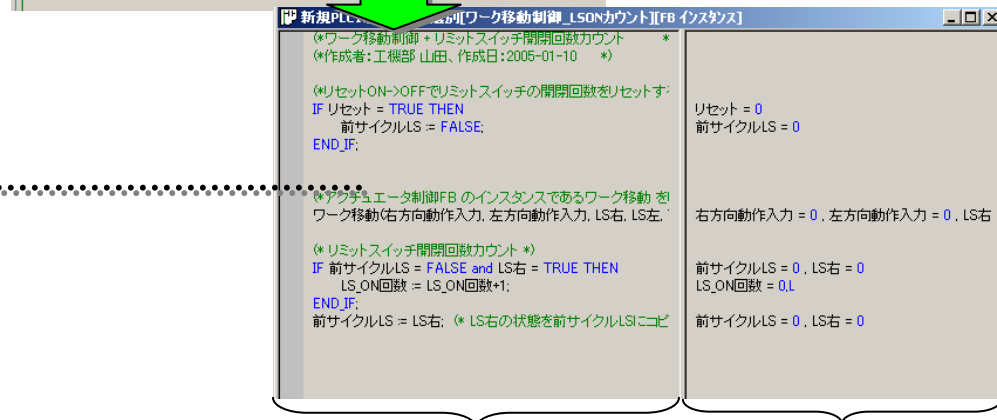
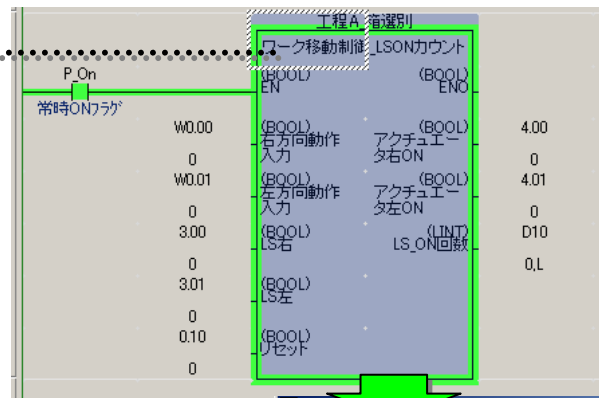
5-9-2. ブレークポイントの設定とステップ実行

ここでは、シミュレーション機能を活用して「ワーク移動制御_LSONカウント」FBのデバッグを行う例で説明します。



運転モードをモニタにします。

「ワーク移動制御_LSONカウント」FBのインスタンスを表示します。



カーソルをFB呼び出し命令内に移動し、マウスダブルクリックまたは、 ボタンをクリックします。

FB STのインスタンス(アドレスが割り付けられている状態)でプログラムと対応する変数の現在値がモニタされます。

プログラムの
設計

FB定義の
入力・デバッグ

FB定義の
ライブラリ化

メインプログラムの
入力

メインプログラムの
デバッグ

FB呼び出し命令の入力パラメータに現在値を設定し実行状態を確認していきます。具体的には以下のケースを設定します。

右方向動作入力: ON
左方向動作入力: OFF
LS右: OFF
LS左: ON
リセット: OFF

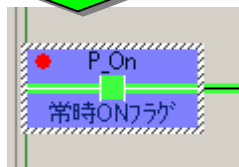
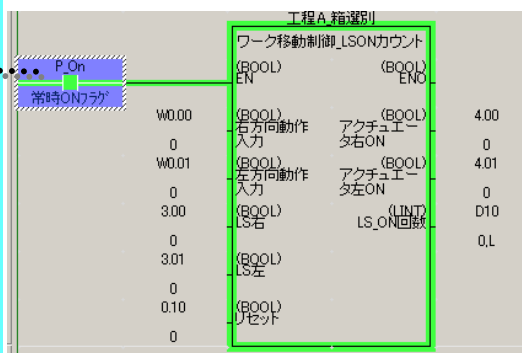
この場合、出力としては、以下が期待されます。

アクチュエータ右ON: ON
アクチュエータ左ON: OFF
LS_ON回数: 1

カーソルをFB呼び出し命令左の入力接点に移動し、



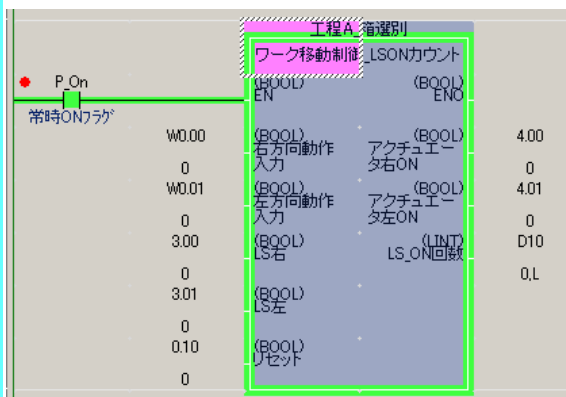
ボタンをクリックします。



プログラムがブレークポイントの位置で停止します。



ボタンをクリックします。



ブレークポイントの入力接点の実行を行い、その次のステップであるFB呼び出し命令で実行が停止します。



プログラムの
設計FB定義の
入力・デバッグFB定義の
ライブラリ化メインプログラムの
入力メインプログラムの
デバッグ

[←][↓][↓]を押します。

ENT

1

ENT

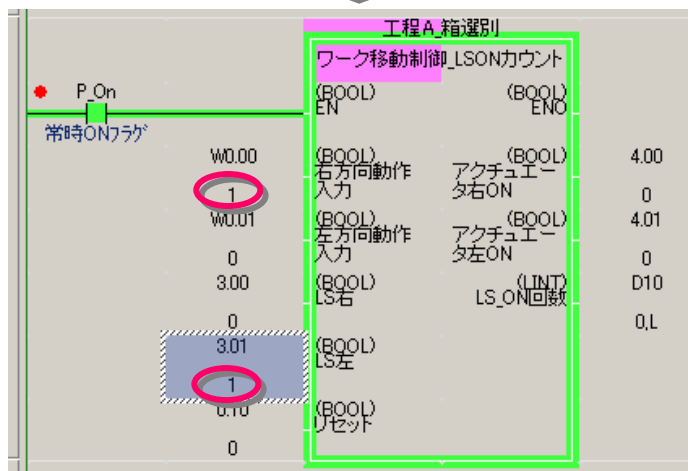
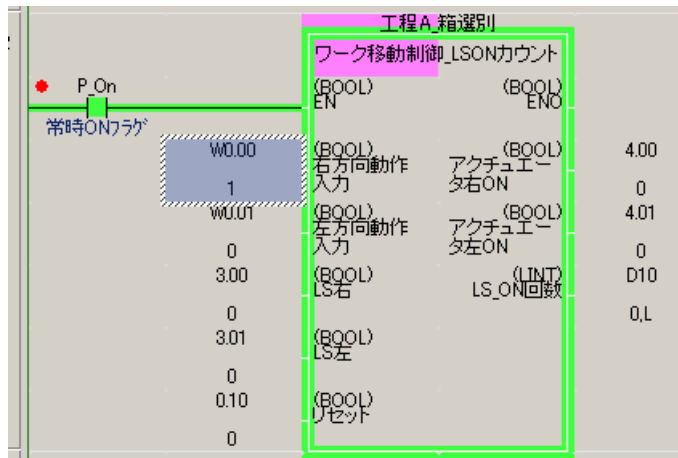
[↓][↓][↓]を押します。

1

ENT

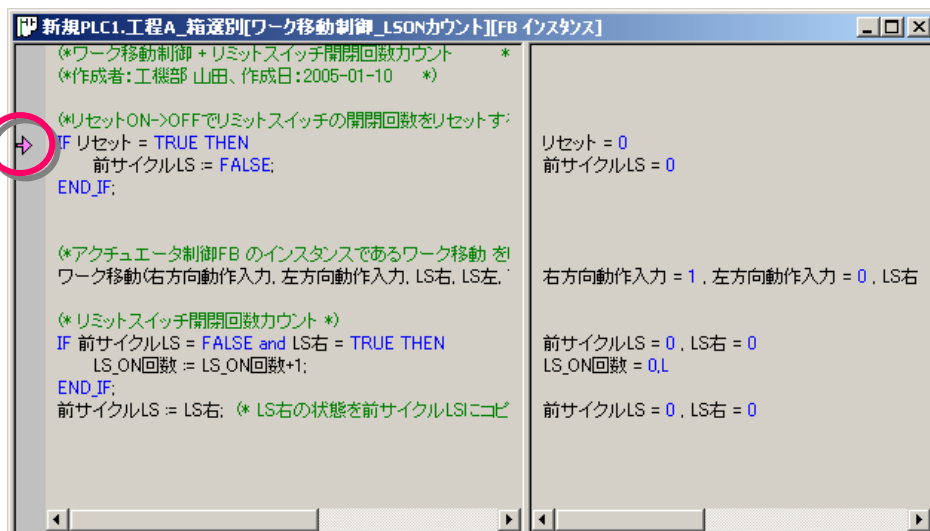
ボタンをクリックします。

FB呼び出し命令の入力パラメータの「右方向動作入力」をON、「LS左」をONします。



必要な入力パラメータの設定を行いました。

STモニタでの実行位置



呼び出し先のSTプログラムの先頭行に実行位置が遷移します。

プログラムの
設計

FB定義の
入力・デバッグ

FB定義の
ライブラリ化

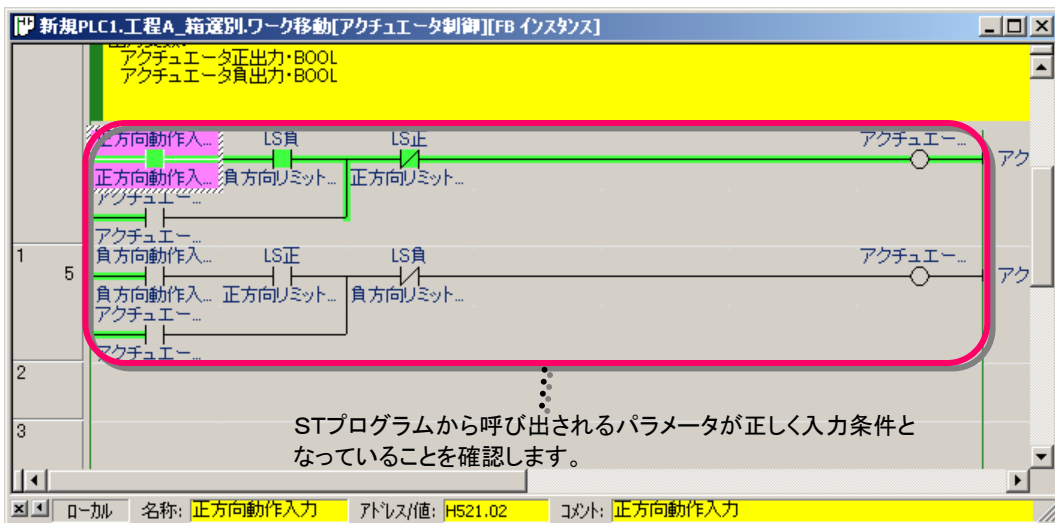
メインプログラムの
入力

メインプログラムの
デバッグ

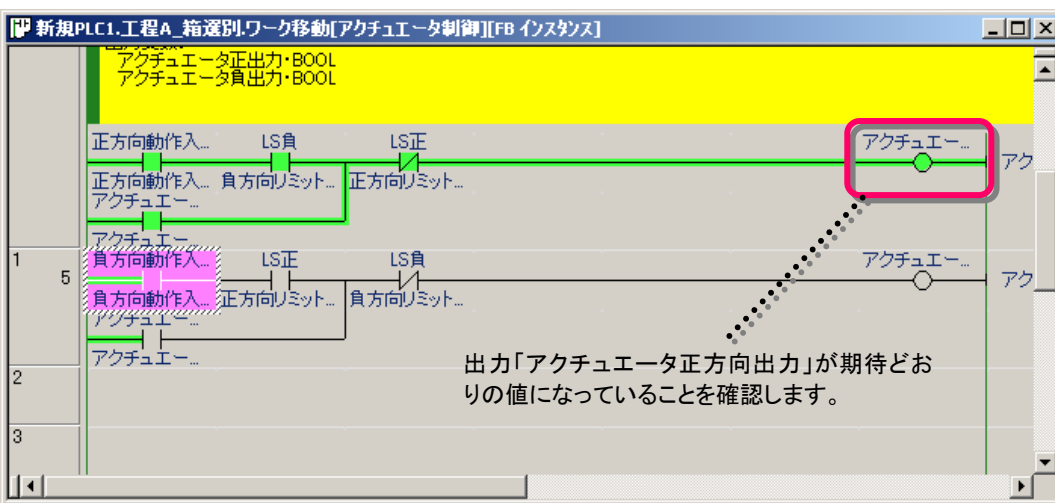
ボタンを2度クリックします。

(*アクチュエータ制御FB のインスタンスである)
ワーク移動(右方向動作入力, 左方向動作入力, 右方向動作入力 = 1, 左方向動作入力 = 0, LS右 = 0, LS左 = 1, アクチュエータ

STプログラムから呼び出されるFBラダープログラムに遷移します。



ボタンを5度クリックします。



ボタンをクリックします。

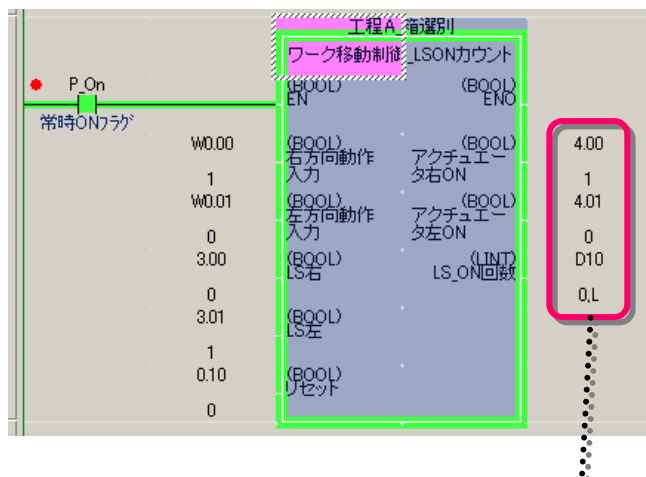
ここで出力が確認できたので、一旦呼び出し元のSTプログラムに戻ります。

(*アクチュエータ制御FB のインスタンスである)
ワーク移動(右方向動作入力, 左方向動作入力, 右方向動作入力 = 1, 左方向動作入力 = 0, LS右 = 0, LS左 = 1, アクチュエータ右ON = 1, アクチュエータ左ON = 0

呼び出し元のSTプログラムのモニタ画面で先ほどの回路の演算結果が正しく反映されていることを確認します。

プログラムの
設計FB定義の
入力・デバッグFB定義の
ライブラリ化メインプログラムの
入力メインプログラムの
デバッグ

ボタンをクリックします。

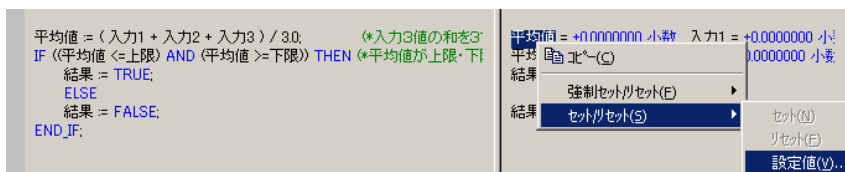


呼び出し元のラダープログラムに移ります。

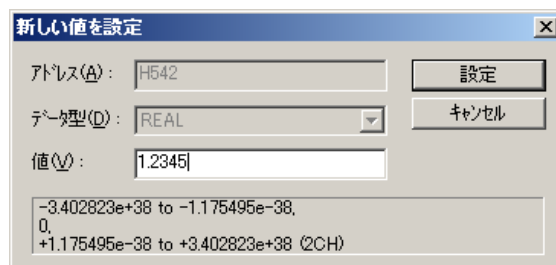
出力パラメータがが正しく反映されていることを確認します。

ワンポイント

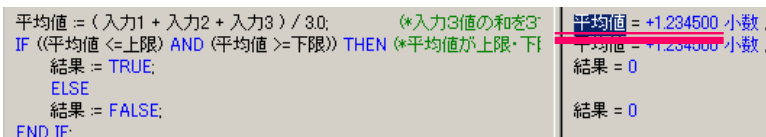
STプログラムの現在値変更は以下の操作で行うことができます。



変更したい変数をマウスマウスカーソルで選択し、マウス右ボタンクリック => セット/リセット => 設定値 を選択。



任意の値を入力、[設定]ボタンを押す。



プログラムの
設計

FB定義の
入力・デバッグ

FB定義の
ライブラリ化

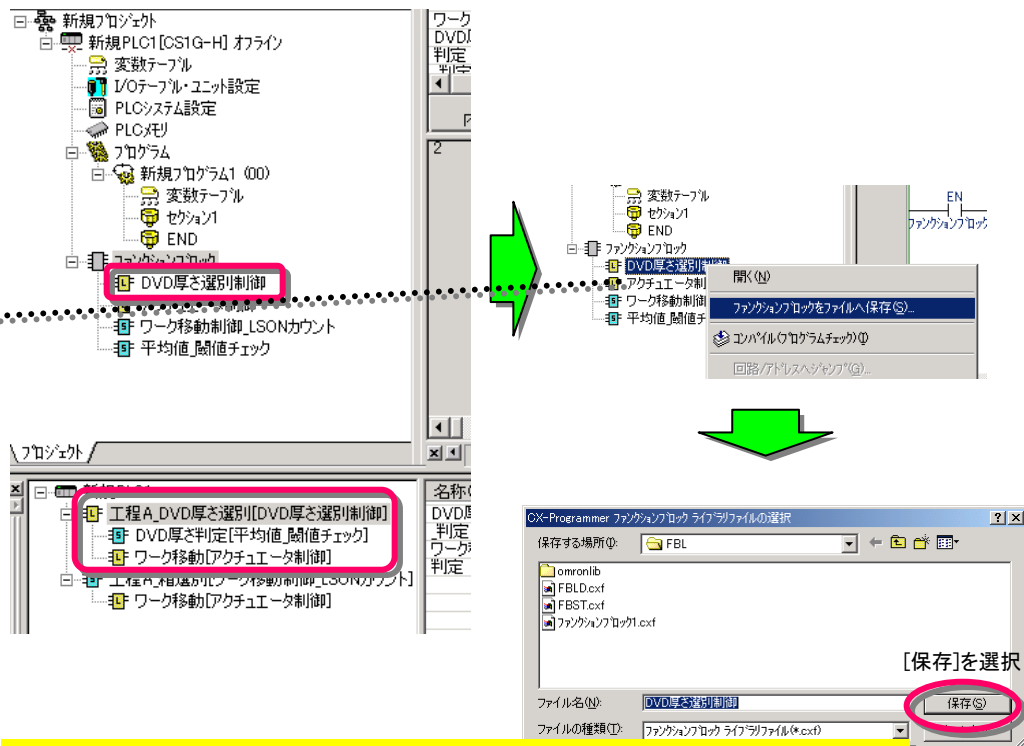
メインプログラムの
入力

メインプログラムの
デバッグ

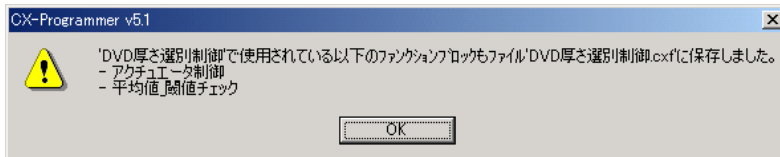
6. FB定義のライブラリ化

動作確認が出来たFB定義を流用できるようにライブラリ化（ファイル化）します。
プロジェクトワークスペースとFBインスタンスビューで階層を確認した上で、ファイル化したい対象のFB定義を決めます。ここでは「DVD厚さ選別制御」FBとします。

「DVD厚さ選別制御」FBを選択し、マウス右ボタンクリックで開くメニューから[ファンクションブロックをファイルへ保存]を選択



デフォルトのセーブ先のフォルダは C:\Program Files\Omron\CX-One\FBL です。
CX-Programmer オプション設定の「FBライブラリの格納場所」で設定変更できます。
Smart FBライブラリが omronlib フォルダにあります。
Userlib\FBL など分類しやすいフォルダを作成して保管してください。



セーブされるFB定義がFBを呼び出している場合、呼び出されるFB定義も含めてセーブされます。プロジェクトへの読み出し時にもセーブ時の呼び出し関係は保持した形でプロジェクトに取り込まれます。
セーブされたFB定義が1本のファイルになるため、管理が容易になります。

プログラムの
設計FB定義の
入力・デバッグFB定義の
ライブラリ化メインプログラムの
入力メインプログラムの
デバッグ

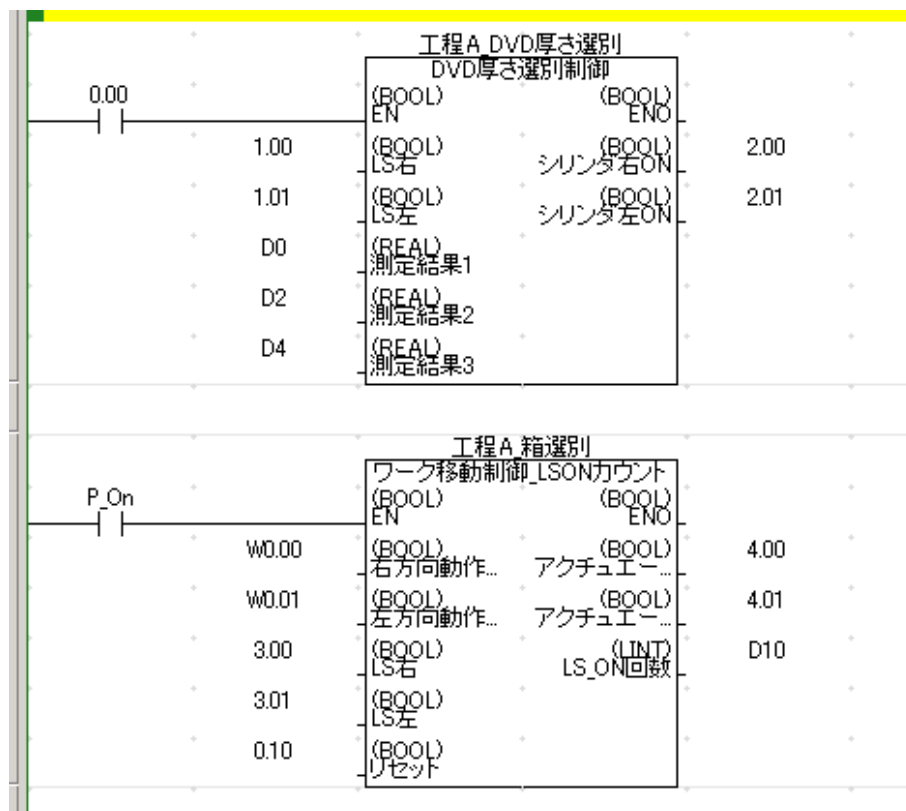
7. メインプログラムの入力

デバッグ済みのFB定義を含むプロジェクトファイルにメインプログラムを追加します。入力するプログラムは、5-7ページに示しているものです。

[グローバル変数]

名称	データ型	アドレス / 値	ラック位置	月
工程A_箱選別	FB [ワーク移動制御_LSONカウント]	N/A [自動]		
工程A_DVD厚さ選別	FB [DVD厚さ選別制御]	N/A [自動]		

※FBを使用するためのインスタンス変数以外は省略しています。



プログラムの入力の方法は2-6ページから2-9ページを参考に入力してください。

プログラムの
設計FB定義の
入力・デバッグFB定義の
ライブラリ化メインプログラムの
入力メインプログラムの
デバッグ

8. メインプログラムのデバッグ

メインプログラムのデバッグは、以下の観点のデバッグとなります。

- ・FBと関連のないプログラム箇所
- ・FBへの入力パラメータとなる値が関連しているプログラム箇所
- ・FBからの出力パラメータを参照しているプログラム箇所

本書のアプリケーション例におけるメインプログラムは上記の箇所がありませんので、以降の説明は省略します。

不要になったファンクションブロックの削除方法

不要になったファンクションブロック定義を削除する場合、ラダープログラム上のインスタンス(呼び出し命令)を削除しただけでは削除できません。

これはインスタンス定義がグローバル変数テーブルに登録されているためです。

このような場合、F7キーによるプログラムチェックを実行するとプログラムチェック結果に未使用のインスタンス定義が表示されますので、この結果を元に変数テーブル上のインスタンス定義の削除ののち、ファンクションブロック定義の削除が簡単にできます。

使用しないファンクションブロック定義、ファンクションブロックインスタンスもCPUユニットにUMの一部として転送されますので、不要なファンクションブロック定義、ファンクションブロックインスタンスは転送前に削除することをお勧めします。

コンパイル(プログラムチェック)実行

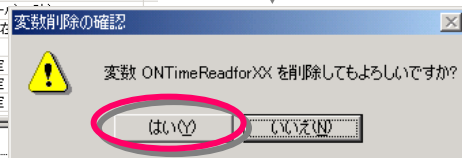
F7 キー



コンパイル(プログラムチェック)結果

名前	変数タイプ	変数値 / 値	変数位置	用途 (ユニット)
ONTimeReadforXX	FB [Dnet204...	N/A [自動]		
P_0.2s	BOOL	CF103	ワーク	0.2秒加算パルス
P_0.1s	BOOL	CF100	ワーク	0.1秒加算パルス
P_0.2s	BOOL	CF101	ワーク	0.2秒加算パルス
P_1min	BOOL	CF104	ワーク	1分加算パルス
P_1s	BOOL	CF102	ワーク	1秒加算パルス
P_AER	BOOL	CF011	ワーク	アクセスエラー(AER)フラグ
P_CIO	WORD	A450	ワーク	CIOエリア指定
P_CY	BOOL	CF004	ワーク	キラー(CY)フラグ
P_Cycle_Time_Error	BOOL	A401.08	ワーク	サイクルタイムオーバー
P_Cycle_Time_Value	UDINT	A264	ワーク	サイクルタイム現在
P_DM	WORD	A460	ワーク	DMエリア指定
P_EM0	WORD	A461	ワーク	EM0エリア指定
P_EM1	WORD	A462	ワーク	EM1エリア指定
P_EM2	WORD	A463	ワーク	EM2エリア指定

Del キー



マウス左ボタンでダブルクリック

マウス左クリック

ファンクションブロック定義が削除可能になります

ファンクションブロック用メモリの割り付けについて

ファンクションブロックを実行するためには、ファンクションブロックインスタンス毎にメモリを割り付ける必要があります。

CX-Programmerは必要なメモリを下の設定ダイアログの内容に基づき自動的に割り付けます。

(PLCメニュー → メモリ割付 → ファンクションブロック/SFC メモリ → ファンクションブロック/SFC アドレス割付)

エリアとしては保持以外(非保持)、保持、タイマ、カウンタの4種類が必要です。作成するアプリケーションに応じて、このダイアログで設定を変更してください。

●設定変更時の注意点

保持以外・保持エリアを変更する場合、使用するCPU高機能ユニット/高機能ユニット用の割付エリアと重複しないように設定してください。

●ファンクションブロック専用エリア

保持以外・保持エリアとして、CS1/CJ1-H/CJ1M ユニットバージョン3.0以上のCPUユニットは従来の保持リレーエリア(H)を拡張しファンクションブロック専用エリア(H512-H1535)を設けました。デフォルトはこのエリアが対象となっています。このエリアはラダー命令のオペランドとしては指定できませんのでご注意ください。

FBインスタンスエリア	先頭アドレス	最終アドレス	サイズ
保持以外	H512	H1407	896
保持	H1408	H1535	128
タイマ	T3072	T4095	1024
カウンタ	C3072	C4095	1024

便利な機能のご紹介

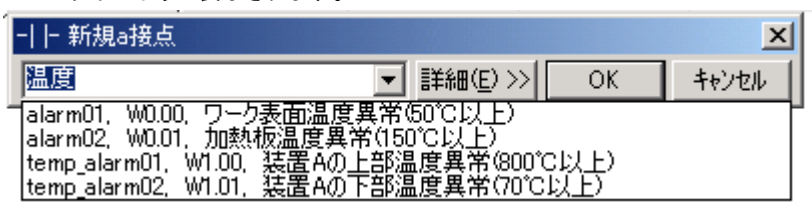
命令のオペランド入力時の自動サーチ&リスト表示

命令のオペランド入力時に、変数名またはI/Oコメントの、自動サーチ&リスト表示が可能です。
入出力接点のオペランド入力時、ある文字列を入力すると、その文字列が変数名またはI/Oコメントに含まれる、アドレスI/Oコメント、および変数名-アドレスI/Oコメントの一覧が、ドロップダウンリストに自動的に表示されます。それを選択することで、オペランドの入力が可能です。
これにより、入力済みの変数/I/Oコメントを再度入力するときの入力効率が向上します。

例) 変数/アドレス入力欄に「温度」と入力



右の▼をクリック、または F4キーを押すと、変数名またはI/Oコメントに「温度」を含む全ての変数/アドレスが以下のように表示されます。



たとえば、「temp_alarm01, W1.00, 装置Aの上部温度異常(800℃以上)」を選択することで、簡単に変数名「temp_alarm01」を入力できます。



FBのプロテクト機能

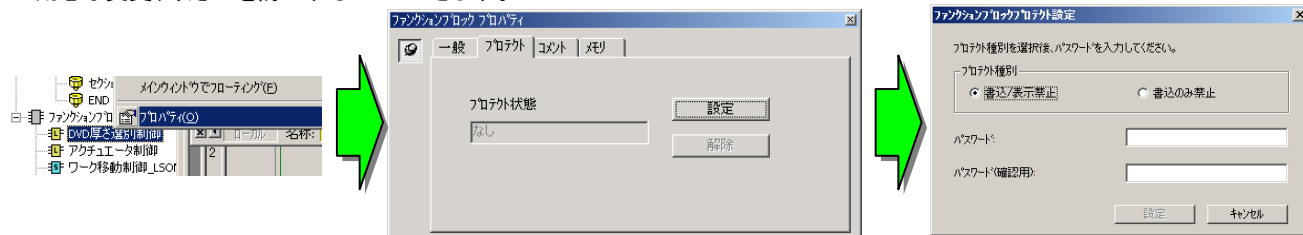
プロジェクトファイル上に割り当てられたファンクションブロック定義にパスワードを設定することで目的に応じたプロテクトを行い、プログラムのノウハウの流出、不用意な変更、改ざん等に対する防止策を施すことができます。

● 書込/表示禁止

プロテクト種別「書込/表示禁止」を設定すると、該当ファンクションブロック定義の内部を表示することができなくなります。ファンクションブロック定義に本プロテクト種別のパスワードプロテクトを設定することで、プログラム上のノウハウの流出を防止することができます。

● 書込のみ禁止

プロテクト種別「書込のみ禁止」を設定すると、該当ファンクションブロック定義の内部を表示することはできますが書き込み・変更ができなくなります。ファンクションブロック定義に本プロテクト種別のパスワードプロテクトを設定することで、プログラムの不用意な変更、改ざんを防止することができます。

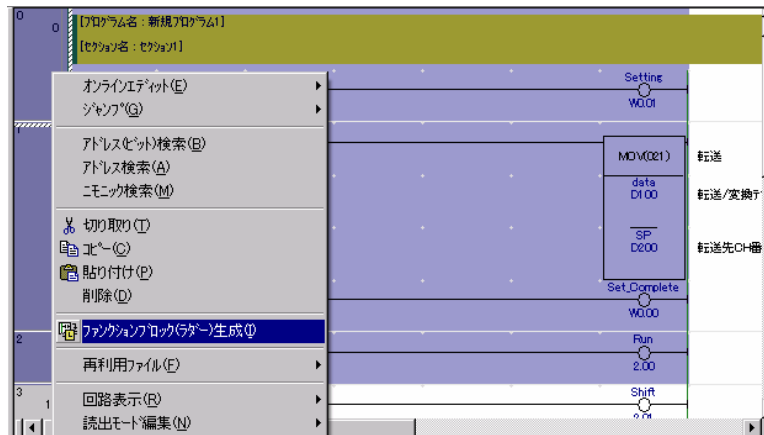


便利な機能のご紹介

既存のラダープログラムを元にFBを作成する機能

動作実績のあるプログラムを元に簡単にFBを作成することができます。
これにより、FBによるプログラムの資産化を加速することができます。

FBにしたい回路を選択し、
マウス右クリック



[ファンクションブロック(ラダー)生成]を選択

[FB変数割付]ダイアログボックスが表示されます。



必要に応じて、回路内で使用されている変数／アドレスの用途(内部変数／入力変数／出力変数／入出力変数)の変更を行います。変数を選択し、ポップアップメニューの[用途変更]から変更します。



ご参考: 回路内で使用されているアドレスに変数が存在しない場合、“AutoGen_”で始まる変数が自動で付加されます。

FBはプログラムで呼び出すときは、パラメータが変数名で表示されますので、少なくとも入力変数・出力変数・入出力変数は、わかりやすい変数名に変更されることをお勧めします。
変更方法は、[FB変数割付]ダイアログにて変更したいアドレスをダブルクリックすることにより表示される、[編集]ダイアログで変更可能です。

FB実数割付

内部変数	入力変数	出力変数	入出力変数		
アドレス	変数名	型	配列サイズ	AT指定	I/Oコメント
0.01	Limit	BOOL	0	なし	
1.00	Error	BOOL	0	なし	
W0.01	Setting	BOOL	0	なし	
0.00	Start	BOOL	0	なし	
W0.00	Set Comp...	BOOL	0	なし	
D100	data	WORD	0	なし	
run	cp	WORD	n	なし	

[OK]を選択

OK キャンセル

ファンクションブロック生成

FB定義名: ファンクションブロック1

コメント:

OK キャンセル

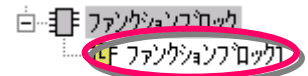
「FB定義名」、「コメント」を入力し、[OK]ボタンをクリックします。

ラダー中に生成したFB呼び出し命令を挿入する場合、[はい]を選択します。

CX-Programmer v6.1

? 生成元ラダー回路の位置にファンクションブロック呼び出しを挿入しますか?

はい(Y) いいえ(N)



FB定義が生成されました。

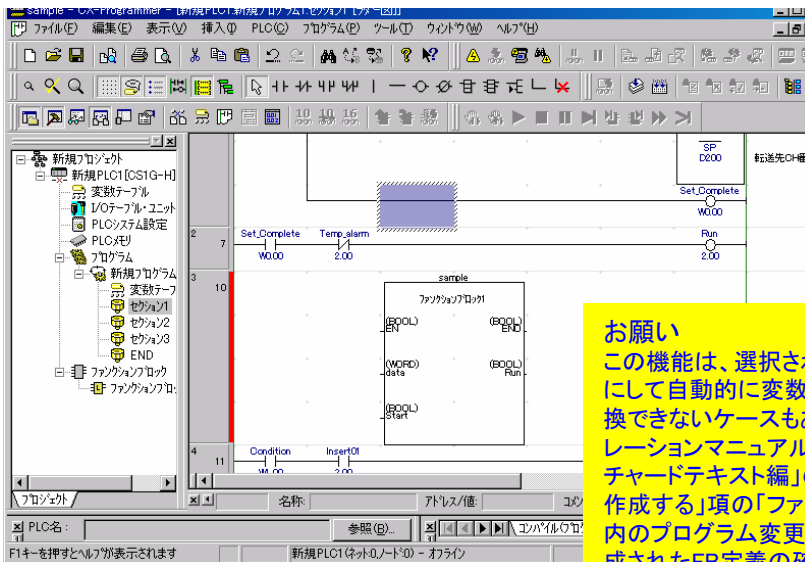
新規ファンクションブロック 呼び出し

FBインスタンス名:

FB定義(D): ファンクションブロック1

OK キャンセル

FBインスタンス名を入力し、[OK]ボタンを押します。



お願い
この機能は、選択された回路中のアドレスの使用状況を元にして自動的に変数の用途を決定していますが、自動で変換できないケースもありますので、「CX-Programmer オペレーションマニュアル ファンクションブロック編/ストラクチャードテキスト編」の「3-2-3 ファンクションブロック定義を作成する」項の「ファンクションブロック定義作成前に、回路内のプログラム変更が必要なケース」を参照いただき、生成されたFB定義の確認をいただくと共に、動作確認を十分に行った上で、本運転に入るようにしてください。

FB呼び出し命令がラダープログラムに挿入されます。

プログラムの
説明

タスクST
新規作成

変数の入力

STプログラム
新規作成

第6章 応用編：ST言語でタスクプログラムを作成する

CX-ProgrammerのタスクプログラムをST言語で作成することができます。制御対象に適した言語の選択肢が、SFC、ラダー、STに広がりました。

Structured Text (ST:ストラクチャードテキスト) 言語は、IEC61131-3 (JIS B3503) で規定されている工業制御用のテキストベースの高級言語です。CX-Programmer では、ファンクションブロックでのSTの使用に加えて、タスクプログラムでSTを使用することができます。

※ファンクションブロックでのSTの使用方法については、4-1ページを参照してください。

IF-THEN-ELSE、FOR/WHILEループなどの制御や、SIN/COSなどの算術演算関数を使用する数値演算を、実アドレスを使用して手軽に行うことができます。

実アドレスを使用して手軽に数値演算プログラムを作成する場合はタスクプログラムでSTを使用、プログラムの再利用を重視する場合はファンクションブロックでSTを使用する、といった使い分けができます。

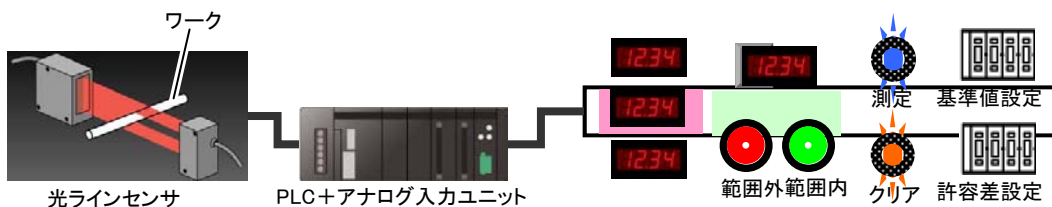
※タスク(task)とは、プログラム実行時の最小単位で、SYSMAC CS1/CJ1シリーズCPUの機能です。タスク分割制御を行うことで、非アクティブのタスクを休止状態にし、サイクルタイムを短縮できます。

1. 作成するプログラムの説明

ここでは、次のような平均値を求めるプログラムを例に、作成手順を説明します。

ワークの直径を3箇所測定し、直径の平均値を求めます。平均値が許容範囲内であれば、緑ランプが、許容範囲外であれば赤ランプが点灯します。

ここでは、ワークの平均値を求め、その値が許容範囲内にあるかどうかを判定するプログラムをSTで作成します。



```
(* 平均値計算 *)
average:=(thickness1+thickness2+thickness3)/3.0; (* average =平均値、thickness1～3 =測定値1～3 *)

(* 平均値の判定 *)
IF flag = 3 THEN
  IF average < criterion-margin THEN
    red_lamp:=TRUE;
  ELSEIF average > criterion+margin THEN
    red_lamp:=TRUE;
  ELSE
    green_lamp:=TRUE;
  END_IF;
END_IF;
```

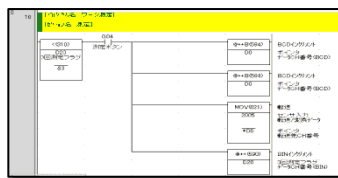
(* flag =3回測定フラグ *)
 (* criterion =基準値、margin =許容範囲 *)
 (* red_lamp =赤ランプ(許容範囲外) *)
 (* green_lamp =緑ランプ(許容範囲内) *)

その他のプログラムはラダーで作成します。

①測定値の初期化、ワーク直径の許容範囲をセットします。



②測定値をセットします。



③7セグメントLEDに、測定値、平均値を表示します。



プログラムの
説明

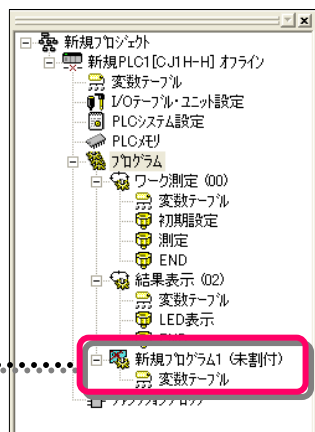
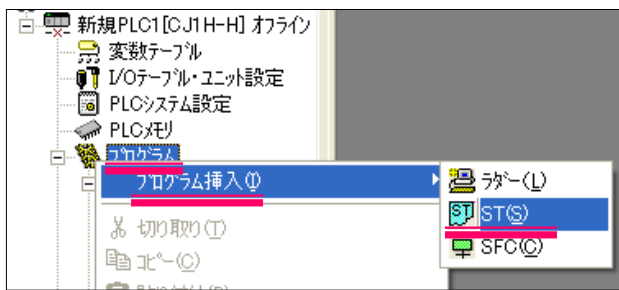
タスクST
新規作成

変数の入力

STプログラム
新規作成

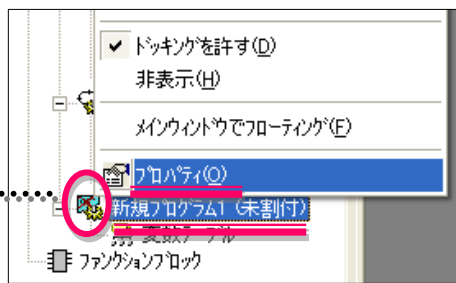
2. タスクSTの作成

タスクSTを作成します。



新規STが生成されました。

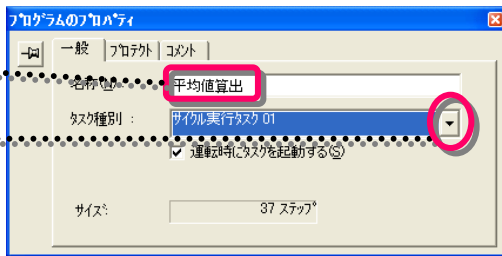
STプログラム名を変更し、
タスクに割り付けます。



カーソルを作成したSTプロ
ラムアイコンに合わせ、
マウスを右クリック
→プロパティ
ダイアログボックスが表示さ
れます。

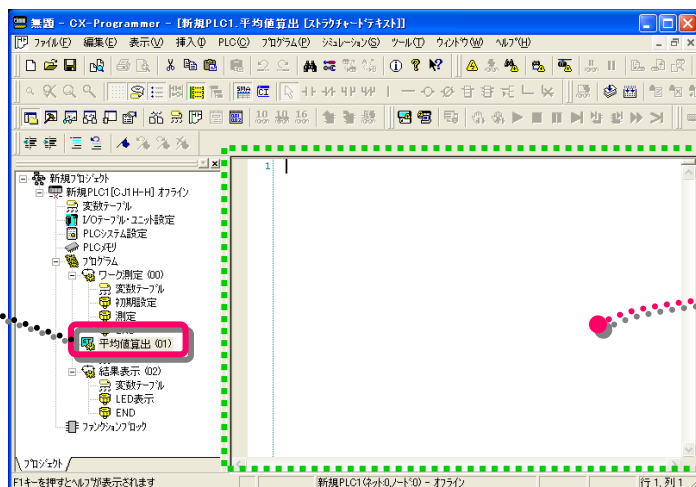
STプログラム名を入力します。
→「平均値算出」

また、タスク種別をプルダウ
ンメニューから選択します。
→「サイクル実行タスク01」



ご参考：
サイクル実行タ
スクは、毎サイクル
実行されます。

STエディタを開きます。



カーソルをSTプログラムアイコン
に合わせ、ダブルクリック
STエディタが開きます。

STエディタ

プログラムの
説明タスクST
新規作成

変数の入力

STプログラム
新規作成

3. STプログラムの変数登録

STプログラムで使用する変数を登録します。

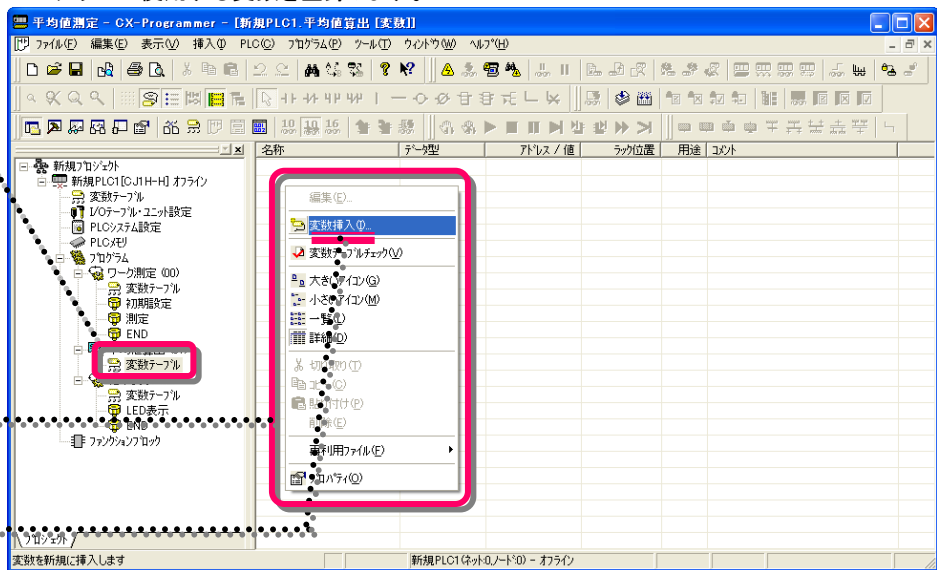
変数テーブルを開きます。

カーソルを変数テーブルアイコン
に合わせ、ダブルクリック
変数テーブルが開きます。

新規変数を作成します。

カーソルを変数テーブル上に
合わせ、マウスを右クリック

プルダウンメニューから

→変数挿入
[新規変数]ダイアログボック
スが表示されます。変数の、
名称
データ型
アドレスまたは値
コメント
を入力します。

The '新規変数' dialog box is shown. It has four main input fields: '名称(N):' (Name), 'データ型(D):' (Data Type), 'アドレスまたは値(A):' (Address or Value), and 'コメント(M):' (Comment). The 'average' variable is being created with data type 'REAL' and address 'D8'. The comment is '平均測定値' (Average measurement value). The 'OK' button is highlighted.

変数名を入力します。

データ型を選択します。
・実数データは、REAL
・接点は、BOOL
・整数は、INT

アドレス/値を入力します。

変数を説明する
コメントを入力します。

最後に[OK]ボタンをクリックします。

上記の作業を繰り返して、変数をすべて入力します。

名称	データ型	アドレス / 値	ラック位置	用途	コメント
red_lamp	BOOL	15.00		ワーク	赤ランプ(許容範囲外)
green_lamp	BOOL	15.01		ワーク	緑ランプ(許容範囲内)
thickness1	REAL	D2		ワーク	測定値1
thickness2	REAL	D4		ワーク	測定値2
thickness3	REAL	D6		ワーク	測定値3
average	REAL	D8		ワーク	平均測定値
criterion	REAL	D10		ワーク	基準値
margin	REAL	D12		ワーク	許容範囲
flag	INT	D20		ワーク	3回測定フラグ

ご参考:

変数登録の際、「アドレス自動割付」機能を使うと、ファンクションブロックでのSTの使用と同じように、実アドレスを気にせず変数の登録を行うことができます。「アドレス自動割付」の使い方については、「CX-Programmerオペレーションマニュアル」を参照してください。

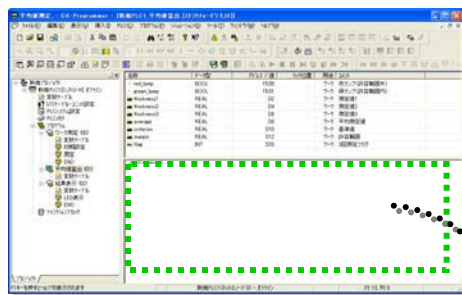
プログラムの
説明タスクST
新規作成

変数の入力

STプログラム
新規作成

4. STプログラムの入力

再びSTエディタを開きます。



プログラムを入力します。

```

(* 平均値計算 *)
average:=(thickness1+thickness2+thickness3)/3.0; (* average =平均値、thickness1~3 =測定値1~3 *)

(* 平均値の判定 *)
IF flag = 3 THEN
  IF average < criterion-margin THEN
    red_lamp:=TRUE;
  ELSEIF average > criterion+margin THEN
    red_lamp:=TRUE;
  ELSE
    green_lamp:=TRUE;
  END_IF;
END_IF;

```

(* flag =3回測定フラグ *)
 (* criterion =基準値、margin =許容範囲 *)
 (* red_lamp =赤ランプ(許容範囲外) *)
 (* green_lamp =緑ランプ(許容範囲内) *)

ご参考:
STプログラム内にはコメント
(* *)を付け、理解しやすく
します。

代入文では、左辺(変数)に右辺(式、変数、または定数)を代入します。

この式では、平均値を計算しています。3回の測定値を足し、3で割った値をaverageに代入します。また、この式では定数「3」はaverageの型(REAL)に合わせ、「3.0」と入力します。

```
average := ( thickness1 + thickness2 + thickness3 )/3.0;
```

IF文では、条件が成立したとき、文を実行します。成立しないとき、ELSEIF以降の条件が成立したら、その文を実行します。いずれも成立しないとき、ELSE以降の文を実行します。

この式では、測定を3回行ったとき、平均値の判定を行います。平均値が許容範囲外の場合、赤ランプを点灯します。許容範囲内の場合、緑ランプを点灯します。

```

IF flag = 3 THEN
  IF average < criterion-margin THEN
    red_lamp := TRUE;
  ELSEIF average > criterion+margin THEN
    red_lamp := TRUE;
  ELSE
    green_lamp := TRUE;
  END_IF;
END_IF;

```

これで、STプログラムの入力は完了です。残りの処理をラダープログラムで作成し、F7キーを押してプログラムのエラーチェック(コンパイル)を行います。プログラムの作成がすべて完了したら、PLCにオンライン接続し、通常のプログラム転送を行います。

F7

付録. ST(Structured Text)使用例

IFステートメント使用例

```
IF expression1 THEN statement-list1
[ ELSIF expression2 THEN statement-list2 ]
[ ELSE statement-list3 ]
END_IF;
```

expression (数式) 1、2 ともに、ブールの値を評価します。*statement-list*(ステートメントリスト)は、*a:=a+1*; や *b:=3+c*; などのステートメントを複数個集めたリストです。

expression1 が真の場合、キーワード IFは*statement-list 1*を実行します。ELSIF が存在し、*expression 1* が偽で*expression 2* が真の場合、IFは*statement-list 2*を実行します。ELSE が存在し、*expression1*または*expression 2* が偽の場合、IFは*statement-list 3*を実行します。*statement-list 1*、*statement-list 2*または *statement-list 3*を実行した後、コントロールは END_IFの次のステートメントに移動します。

IFステートメント内にステートメント ELSIF が複数存在することがありますが、ELSE ステートメントは1つのみです。

IF ステートメントは別の IF ステートメント内にネスティングすることができます(例5参照)。

例 1

```
IF a > 0 THEN
  b := 0;
END_IF;
```

この例では、変数 *a* が0より大きい場合、変数 *b* は0に決まります。

変数 *a* が0以下の場合、変数 *b* では何も実行されず、コントロールは END_IF の次のプログラムステップに移動します。

例 2

```
IF a THEN
  b := 0;
END_IF;
```

この例では、変数 *a* が真の場合、変数 *b* は0に決まります。

変数 *a* が偽の場合、変数 *b* では何も実行されず、コントロールは END_IF の次のプログラムステップに移動します。

例 3

```
IF a > 0 THEN
  b := TRUE;
ELSE
  b := FALSE;
END_IF;
```

この例では、変数 *a* が0より大きい場合、変数 *b* はTRUE(1)に決まり、コントロールは END_IF の次のプログラムステップに移動します。

変数 *a* が0以下の場合、変数 *b* では何も実行されず、コントロールは ELSE の次のステートメントに移動し、次の変数 *b* が FALSE(0)に決まります。

その後コントロールは END_IF の次のプログラムステップに移動します。

例 4

```
IF a < 10 THEN
  b := TRUE;
  c := 100;
ELSIF a > 20 THEN
  b := TRUE;
  c := 200;
ELSE
  b := FALSE;
  c := 300;
END_IF;
```

この例では、変数 *a* が10未満の場合、変数 *b* は真の値(1)に決まり、変数 *c* は100に決まります。その後コントロールは END_IF の次のプログラムステップに移動します。

変数 *a* が10以上の場合、コントロールは ELSE_IF に移動し、変数 *a* が20より大きい場合は、変数 *b* は真の値(1)に、変数 *c* は200に決まります。その後コントロールは END_IF の次のプログラムステップに移動します。

変数 *a* が10から20の間にある場合(上記の条件で IF とELSE_IF 両方が偽の場合)、コントロールは ELSE に移動し、変数 *b* は偽の値(0)に、変数 *c* は300に決まります。その後コントロールは END_IF の次のプログラムステップに移動します。

IFステートメント使用例

例 5

```
IF a THEN
  b := TRUE;
ELSE
  IF c>0 THEN
    d := 0;
  ELSE
    d := 100;
  END_IF;
  d := 400;
END_IF;
```

この例(IF...THENステートメントのネスティング例)では、変数 a が真の値(1)の場合、変数 b は真の値(1)に決まり、コントロールは END_IF の次のプログラムステップに移動します。

変数 a が偽の値(0)の場合、変数 b では何も実行されず、コントロールは ELSE の次のステートメントに移動します(この例では、もう1つのステートメントである IF ... THEN に移動し、このステートメントは例3に記述した通りに実行されますが、サポートするIEC61131-3 のステートメントを使用することがあります。)

記述した IF ... THEN ステートメントを実行後、変数 d は400に決まります。

その後コントロールは END_IF の次のプログラムステップに移動します。

WHILEステートメント使用例

```
WHILE expression DO
  statement-list;
END_WHILE;
```

WHILE *expression*(数式)は、ブールの値を求めます。*statement-list*(ステートメントリスト)は、ステートメントを複数個集めたリストです。

キーワード WHILE は、*expression* が真である限り、繰り返し *statement-list* を実行します。*expression* が偽になると、コントロールは END_WHILE の次のステートメントに移動します。

例 1

```
WHILE a < 10 DO
  a := a + 1;
  b := b * 2.0;
END_WHILE;
```

この例では、WHILE の値を評価し、その値が真の場合(つまり変数 a が10未満の場合)、ステートメントリスト(a:=a+1;とb:=b * 2.0;)が実行されます。ステートメントリスト実行後、コントロールは WHILE の始めに戻ります。変数 a が10未満である限り、この工程を繰り返します。

変数 a が10以上の場合、ステートメントリストは実行されず、コントロールは END_WHILE の次のプログラムステップに移動します。

例 2

```
WHILE a DO
  b := b + 1;
  IF b > 10 THEN
    a:= FALSE;
  END_IF;
END_WHILE;
```

この例では、WHILE の値を評価し、その値が真の場合(つまり変数 a が真の場合)、ステートメントリスト(b:=b+1; と IF ステートメント... THEN)が実行されます。ステートメントリスト実行後、コントロールは WHILE の始めに戻ります。変数 a が真である限り、この工程を繰り返します。

変数 a が偽の場合、ステートメントリストは実行されず、コントロールは END_WHILE の次のプログラムステップに移動します。

例 3

```
WHILE (a + 1) >= (b * 2) DO
  a := a + 1;
  b := b / c;
END_WHILE;
```

この例では、WHILE の値を評価し、その値が真の場合(つまり変数 a+1 が変数 b×2 以上の場合)、ステートメントリスト(a:=a+1;とb:=b/c;)が実行されます。ステートメントリスト実行後、コントロールは WHILE の始めに戻ります。WHILE が真である限り、このプロセスを繰り返します。WHILE が偽の場合、ステートメントリストは実行されず、コントロールは END_WHILE の次のプログラムステップに移動します。

WHILEステートメント使用例

例 4

```
WHILE (a - b) <= (b + c) DO
  a := a + 1;
  b := b * a;
END_WHILE;
```

この例では、WHILE の値を評価し、その値が真の場合（つまり変数 $a-b$ が変数 $b+c$ 以下の場合）、ステートメントリスト（ $a:=a+1$; と $b:=b * a$;）が実行されます。ステートメントリスト実行後、コントロールは WHILE の始めに戻ります。WHILE が真である限り、この工程を繰り返します。WHILE が偽の場合、ステートメントリストは実行されず、コントロールは END_WHILE の次のプログラムステップに移動します。

REPEATステートメント使用例

REPEAT

```
statement-list;
UNTIL expression
END_REPEAT;
```

REPEAT *expression* (数式) は、ブールの値を評価します。statement-list (ステートメントリスト) は、ステートメントを複数個集めたリストです。

キーワード REPEAT は、*expression* が偽である限り、繰り返し *statement-list* を実行します。*expression* が真になると、コントロールは END_REPEAT の次のステートメントに移動します。

例 1

```
REPEAT
  a := a + 1;
  b := b * 2.0;
UNTIL a > 10
END_REPEAT;
```

この例では、ステートメントリスト（ $a:=a+1$; と $b:=b * 2.0$;）が実行されます。ステートメントリスト実行後、UNTIL の値を評価し、その値が偽の場合（つまり変数 a が10以下の場合）、コントロールは REPEAT の始めに戻り、ステートメントリストが再度実行されます。UNTIL が偽である限り、この工程を繰り返します。UNTIL が真の場合（つまり変数 a が10より大きい場合）、コントロールは END_REPEAT の次のプログラムステップに移動します。

例 2

```
REPEAT
  b := b + 1;
  IF b > 10 THEN
    a := FALSE;
  END_IF;
UNTIL a
END_REPEAT;
```

この例では、ステートメントリスト（ $b:=b+1$; と IF ステートメント...THEN）が実行されます。ステートメントリスト実行後、UNTIL の値を評価し、その値が偽の場合（つまり変数 a が偽の場合）、コントロールは REPEAT の始めに戻り、ステートメントリストが再度実行されます。UNTIL が偽である限り、この工程を繰り返します。UNTIL が真の場合（つまり変数 a が真の場合）、コントロールは END_REPEAT の次のプログラムステップに移動します。

例 3

```
REPEAT
  a := a + 1;
  b := b / c;
UNTIL (a + 1) >= (b * 2)
END_REPEAT;
```

この例では、ステートメントリスト（ $a:=a+1$; と $b:=b/c$;）が実行されます。ステートメントリスト実行後、UNTIL の値を評価し、その値が偽の場合（つまり変数 $a+1$ が変数 $b \times 2$ 未満の場合）、コントロールは REPEAT の始めに戻り、ステートメントリストが再度実行されます。UNTIL が偽である限り、この工程を繰り返します。UNTIL が真の場合（つまり変数 $a+1$ が変数 $b \times 2$ 以上の場合）、コントロールは END_REPEAT の次のプログラムステップに移動します。

例 4

```
REPEAT
  a := a + 1;
  b := b * a;
UNTIL (a - b) <= (b + c)
END_REPEAT;
```

この例では、ステートメントリスト（ $a:=a+1$; と $b:=b * a$;）が実行されます。ステートメントリスト実行後、UNTIL の値を評価し、その値が偽の場合（つまり変数 $a-b$ が変数 $b+c$ より大きい場合）、コントロールは REPEAT の始めに戻り、ステートメントリストが再度実行されます。UNTIL が偽である限り、この工程を繰り返します。UNTIL が真の場合（つまり変数 $a-b$ が変数 $b+c$ 以下の場合）、コントロールは END_REPEAT の次のプログラムステップに移動します。

```
FOR control variable := integer expression1 TO integer expression2 [ BY integer expression3 ] DO
    statement-list;
END_FOR;
```

FOR *control variable* (制御変数)は整数の変数となります。FOR *integer expression* (整数式)は、制御変数と同じタイプの整数変数を評価します。*statement-list*(ステートメントリスト)は、ステートメントを複数個集めたリストです。

キーワード FOR は、*control variable* が *integer expression 1* から *integer expression 2* の範囲内にある限り、*statement-list* を繰り返し実行します。BY が存在する場合、*control variable* は *integer expression 3* によってインクリメントされ、存在しない場合は、デフォルトで1つインクリメントされます。*statement-list* の呼び出しを実行するたびに、*control variable* がインクリメントされます。*control variable* が *integer expression 1* から *integer expression 2* の範囲外になると、コントロールは END_FOR の次のステートメントに移動します。

FOR ステートメントは別の FOR のステートメント内にネスティングすることができます。

例 1

```
FOR a := 1 TO 10 DO
    b := b + a;
END_FOR;
```

この例では、FOR を最初に評価し、変数 a を値1に初期化します。次に変数 a の値を FOR ステートメントの値 TO と比較し、その値が10以下だった場合は、ステートメントリスト (b:=b+a;) を実行します。変数 a は1つインクリメントされ、コントロールは FORステートメントの始めに戻ります。変数 a を再び TO と比較し、その値が10以下だった場合は、再びステートメントリストを実行します。変数 a の値が10より大きくなるまでこの工程を繰り返します。その後コントロールは END_FOR の後に続くプログラムステップに移動します。

例 2

```
FOR a := 1 TO 10 BY 2 DO
    b := b + a;
    c := c + 1.0;
END_FOR;
```

この例では、FOR を最初に評価し、変数 a を値1に初期化します。次に変数 a の値を FOR ステートメントの値 TO と比較し、その値が10以下だった場合は、ステートメントリスト (b:=b+a; と c:=c+1.0;) を実行します。変数 a は2つインクリメントされ、コントロールは FOR ステートメントの始めに戻ります。変数 a を再び TO と比較し、その値が10以下だった場合は、再びステートメントリストを実行します。変数 a の値が10より大きくなるまでこの工程を繰り返します。その後コントロールは END_FOR の後に続くプログラムステップに移動します。

例 3

```
FOR a := 10 TO 1 BY -1 DO
    b := b + a;
    c := c + 1.0;
END_FOR;
```

この例では、FOR を最初に評価し、変数 a を値10に初期化します。次に変数 a の値を FOR ステートメントの値 TO と比較し、その値が1以上だった場合は、ステートメントリスト (b:=b+a; と c:=c+1.0;) を実行します。変数 a は1つデクリメントされ、コントロールは FOR ステートメントの始めに戻ります。変数 a を再び TO と比較し、その値が1以上だった場合は、再びステートメントリストを実行します。変数 a の値が1未満になるまでこの工程を繰り返します。その後コントロールは END_FOR の後に続くプログラムステップに移動します。

例 4

```
FOR a := b + 1 TO c + 2 DO
    d := d + a;
    e := e + 1;
END_FOR;
```

この例では、FOR を最初に評価し、変数 a を変数 b+1 の値に初期化します。また、変数 c+2 の値に対して FOR ステートメントの値 TO を評価します。次に変数 a の値を TO の値と比較し、a が TO 以下だった場合は、ステートメントリスト (d:=d+a; と e:=e+1;) を実行します。変数 a は1つインクリメントされ、コントロールは FOR ステートメントの始めに戻ります。変数 a を再び TO と比較し、a が TO 以下だった場合は、再びステートメントリストを実行します。変数 a の値が TO の値より大きくなるまでこの工程を繰り返します。その後コントロールは END_FOR の後に続くプログラムステップに移動します。

例 5

```
FOR a := b + c TO d - e BY f DO
  g := g + a;
  h := h + 1.0;
END_FOR;
```

この例では、FOR を最初に評価し、変数 a を変数 b+c の値に初期化します。また、変数 d-e の値に対して FOR ステートメントの値 TO を評価します。次に変数 a の値を TO の値と比較し、変数 f の値が正数かつ変数 a の値が TO の値以下だった場合は、ステートメントリスト (g:=g+a; と h:=h+1.0;) を実行します。

変数 f の値が負数かつ変数 a の値が TO の値以上だった場合、ステートメントリスト (g:=g+a; と h:=h+1.0;) を実行します。変数 a は変数 f の値によってインクリメントまたはデクリメントされ、コントロールは FOR ステートメントの始めに戻ります。変数 a を再び TO と比較し、上記の条件に当てはまる場合は、再びステートメントリストを実行します。

変数 a の値が TO の値より大きくなる (変数 f が正数の場合) か、TO 未満になる (変数 f が負数の場合) までこの工程を繰り返します。その後コントロールは END_FOR の後に続くプログラムステップに移動します。

CASEステートメント使用例

```
CASE expression OF
  case label1 [, case label2] [ .. case label3 ] : statement-list1;
[ ELSE
  statement-list2 ]
END_CASE;
```

CASE expression (数式) は整数値を評価します。statement-list (ステートメントリスト) は、ステートメントを複数個集めたリストです。case level (ケースレベル) は 0、1、+100、-2 など、有効な整数値とします。

キーワード CASE は expression を評価し、最初の expression と一致する値を持つ case level に対応する statement-list を実行します。実行後、コントロールは END_CASE の次のステートメントに移動します。case level のうち、expression と一致する値がなく、かつ ELSE コマンドが存在する場合、キーワード ELSE に対応する statement-list を実行します。キーワード ELSE が存在しない場合、コントロールは END_CASE の次のステートメントに移動します。

CASE ステートメント内に、複数の異なる case level ステートメント (とその statement-list) が存在することはありますが、ELSE ステートメントは 1 つだけとなります。

演算子、を使用して、同一 statement-list 内にある複数の case level を一覧表示できます。

演算子 .. は case level の範囲を表します。CASE expression がその範囲内にある場合、対応する statement-list が実行されます。例えば、case level 1..10: a:=a+1; で、CASE expression が 1 以上 10 未満の場合、a:=a+1 を実行します。

例 1

```
CASE a OF
  2 : b := 1;
  5 : c := 1.0;
END_CASE;
```

この例では、CASE ステートメントの値を評価し、その後 CASE ステートメントの比較対象値 (ここでは 2 と 5) それぞれと比較します。

変数 a の値が 2 の場合、ステートメントリスト (b:=1;) を実行します。その後コントロールは END_CASE の後に続くプログラムステップに戻ります。

変数 a の値が 5 の場合、ステートメントリスト (c:=1.0;) を実行します。その後コントロールは END_CASE の後に続くプログラムステップに移動します。

変数 a の値が CASE ステートメントのどの比較対象値とも一致しない場合、コントロールは END_CASE の後に続くプログラムステップに移動します。

例 2

```
CASE a + 2 OF
  -2 : b := 1;
  5 : c := 1.0;
ELSE
  d := 1.0;
END_CASE;
```

この例では、CASE ステートメントの値を評価し、その後 CASE ステートメントの比較対象値 (ここでは -2 と 5) それぞれと比較します。

変数 a+2 の値が -2 の場合、ステートメントリスト (b:=1;) を実行します。その後コントロールは END_CASE の後に続くプログラムステップに移動します。変数 a+2 の値が 5 の場合、ステートメントリスト (c:=1.0;) を実行します。その後コントロールは END_CASE の後に続くプログラムステップに移動します。変数 a+2 の値が -2 でも 5 でもない場合、ELSE にあるステートメントリスト

(d:=1.0;) を実行します。その後コントロールは END_CASE の後に続くプログラムステップに移動します。

CASEステートメント使用例

例 3

```
CASE a + 3 * b OF
  1, 3 :   b := 2;
  7, 11 : c := 3.0;
ELSE
  d := 4.0;
END_CASE;
```

この例では、CASE ステートメントの値を評価し、その後CASE ステートメントの比較対象値(ここでは1、3、7、11)それぞれと比較します。

変数 a + 3x変数 b の値が1または3の場合、ステートメントリスト (b:=2;)を実行します。その後コントロールはEND_CASE の後に続くプログラムステップに移動します。

変数 a + 3x変数 b の値が7または11の場合、ステートメントリスト (c:=3.0;)を実行します。その後コントロールは END_CASE の後に続くプログラムステップに移動します。

変数 a + 3x変数 b の値が1、3、7、11のどれでもない場合、ELSE にあるステートメントリスト (d:=4.0;)を実行します。その後コントロールは END_CASE の後に続くプログラムステップに移動します。

例 4

```
CASE a OF
-2, 2, 4 :   b := 2;
           c := 1.0;
6..11, 13 : c := 2.0;
1, 3, 5 :   c := 3.0;
ELSE
  b := 1;
  c := 4.0;
END_CASE;
```

この例では、CASE ステートメントの値を評価し、その後CASE ステートメントの比較対象値それぞれと比較します。ここでは (-2か2か4)と (6から11あるいは13)と (1か3か5)です。

変数aの値が -2または2または4場合、ステートメントリスト (b:=2; と c:=4.0)を実行します。その後コントロールは END_CASE の後に続くプログラムステップに移動します。

変数 a の値が6、7、8、9、10、11、13のどれかの場合、ステートメントリスト (c:=2.0;)を実行します。その後コントロールは END_CASE の後に続くプログラムステップに移動します。

変数 a の値が1、3、5のどれかの場合、ステートメントリスト (c:=3.0;)を実行します。その後コントロールはEND_CASE の後に続くプログラムステップに移動します。

変数aの値が上記のどの値でもない場合、ELSE にあるステートメントリスト (b:=1;と c:=4.0)を実行します。その後コントロールは END_CASE の後に続くプログラムステップに移動します。

EXITステートメント使用例

```
WHILE expression DO
    statement-list1;
    EXIT;
END_WHILE;
statement-list2;
```

```
REPEAT
    statement-list1;
    EXIT;
UNTIL expression
END_REPEAT;
statement-list2;
```

```
FOR control variable := integer expression1 TO integer expression2 [ BY integer expression3 ] DO
    statement-list1;
    EXIT;
END_FOR;
statement-list2;
```

statement-list(ステートメントリスト)は、ステートメントを複数個集めたリストです。

キーワード **EXIT** は反復構文(**WHILE**, **REPEAT**, **FOR** ステートメント)の中でのみ使用可能で、反復ループを中断し次のステートメントに移動することができます。反復ループの中で *statement-list 1* を実行後キーワード **EXIT** が実行されると直ちに *statement-list 2* から実行が継続されます。

例 1

```
WHILE a DO
    IF c = TRUE THEN
        b:=0;EXIT;
    END_IF;
    IF b > 10 THEN
        a:= FALSE;
    END_IF;
END_WHILE;
d:=1;
```

WHILE ループを実行中に、最初の IF 条件が真(変数 c が真)である場合、ステートメントリスト(b := 0; and EXIT;)が実行されます。キーワード EXIT が実行されると、WHILE ループは中断されEND_WHILE の次のステートメント(d := 1;)から処理が継続されます。

例 2

```
a:=FALSE;
FOR i:=1 TO 20 DO
    FOR j:=0 TO 9 DO
        IF i>=10 THEN
            n:=i * 10+j;
            a:=TRUE;EXIT;
        END_IF;
    END_FOR;
    IF a THEN EXIT; END_IF;
END_FOR;
d:=1;
```

FOR ループを実行中に、内側の FOR ループ内の IF ステートメントの条件が真(i>=10 が真)である場合、ステートメントリスト(n := i * 10 + j;; a := TRUE;; EXIT;)が実行されます。キーワード EXIT が実行されると、内側の FOR ループは中断されEND_FOR の次の IF ステートメントから処理が継続されます。この IF ステートメントの条件が真(変数 a の値が真)である場合、キーワード EXIT が実行され、外側の FOR ループが中断され END_FOR の次のステートメント(d := 1;)から処理が継続されます。

RETURNステートメント使用例

```
statement-list1;  
RETURN;  
statement-list2;
```

statement-list(ステートメントリスト)は、ステートメントを複数個集めたリストです。

キーワード **RETURN** はファンクションブロック内部の処理を途中で(*statement-list 1*実行後に)終了し、*statement-list 2*を実行することなくファンクションブロックの呼び出しもとのプログラムへ戻ります。

例 1

```
IF a_1 * b > 100 THEN  
    c := TRUE; RETURN;  
END_IF;  
IF a_2 * (b + 10) > 100 THEN  
    c := TRUE; RETURN;  
END_IF;  
IF a_3 * (b + 20) > 100 THEN  
    c := TRUE;  
END_IF;
```

1番目または2番目の IF ステートメントが真(“a_1 * b” が100より大、または“a_2 * (b + 10)” が100より大)の場合、ステートメント(c := TRUE; RETURN;)が実行されます。キーワード RETURN が実行されるとファンクションブロック内部の処理を終了し、ファンクションブロックの呼び出しもとのプログラムへ戻ります。

配列の例

variable name [*subscript index*]

配列とは、変数などの集まりのことです。配列のサイズは FB 変数テーブル上で定義することができます。

各変数へのアクセスは、配列サブスクリプト演算子[]を使って実行することができます。

subscript index(配列インデックス)を使用して、配列中にある特定の変数にアクセスすることができます。配列インデックスは、正数値、整数式、整数の変数、うちのどれかになります。サブスクリプト・インデックスはゼロベースの値となります。サブスクリプト・インデックス値が0(ゼロ)の場合は先頭の変数にアクセスし、1の場合は2つ目の変数にアクセスします。

警告

配列インデックスが整数式または整数の変数の場合、その結果得られる配列インデックスが、配列中の有効なインデックス範囲内にあることを確認する必要があります。無効なインデックスを使って配列にアクセスすることを避ける必要があります。変数配列オフセット使用時の安全なコード記述については、例5を参照してください。

例 1

```
a[0] := 1;  
a[1] := -2;  
a[2] := 1+2;  
a[3] := b;  
a[4] := b+1;
```

この例では、変数 a は要素を5つ持つ配列で、データ型は INT とします。変数 b のデータ型も INT とします。実行すると、配列中の最初の要素が1に設定されます。2番目の要素は-2、3番目の要素は3(1+2)、4番目の要素は変数 b の値、5番目の要素は変数 b + 1の値に設定されます。

例 2

```
c[0] := FALSE;  
c[1] := 2 > 3;
```

この例では、変数 c は要素を2つ持つ配列で、データ型は BOOL とします。実行すると、配列中の最初の要素が FALSE、2番目の要素も FALSE(2 > 3 = FALSE)に設定されます。

配列の例

例 3

```
d[9]:= 2.0;
```

この例では、変数 d は要素を10個持つ配列で、データ型は REAL とします。実行すると、配列中の最後の要素(10番目の要素)が2.0に設定されます。

例 4

```
a[1] := b[2];
```

この例では、変数 a と b は同じデータ型の配列とします。実行すると、変数 a の2番目の要素が、変数 b の3番目の要素の値に設定されます。

例 5

```
a[b] := 1;
```

```
a[b+1] := 1;
```

```
a[(b+c) * (d-e)] := 1;
```

注意: 整数の変数と整数式を使用して配列にアクセスするため、実際のインデックス値はランタイムまで分かりません。そのためユーザは、そのインデックスが配列aの有効な範囲内にあることを確認する必要があります。安全策として、配列インデックスが有効であることをチェックする方法などがあります。

```
f := (b+c) * (d-e);
```

```
IF (f > 0) AND (f < 5) THEN
```

```
  a[f] := 1;
```

```
END_IF;
```

変数fのデータ型はINTです。

例 6

```
a[b[1]]:= c;
```

```
a[b[2] + 3]:= c;
```

ある配列要素の式を別の配列要素式の中で使う例を表しています。

標準関数一覧

関数種類	
数値演算関数	絶対値、三角関数などの数値演算関数
文字列関数	文字列抽出や結合などの標準文字列関数
データ型変更関数	変更元のデータ型_TO_変更先データ型(変数名)
数値文字列変更関数	変更元のデータ型_TO_STRING(変数名) STRING_TO_変更先データ型(変数名)
データシフト関数	ビットシフト、ビット回転などのデータシフト関数
データ制御関数	上下限リミット制御などの制御関数
データ選択関数	最大値/最小値、マルチプレクサなどのデータ選択関数

■ 数値演算関数

関数名	引数のデータ型		戻り値のデータ型	内容	例
ABS(引数)	INT DINT LINT UINT UDINT ULINT REAL LREAL		INT DINT LINT UINT UDINT ULINT REAL LREAL	絶対値 引数	a:=ABS(b) (* 変数 b の絶対値を変数 a に格納 *)
SQRT(引数)	REAL LREAL		REAL LREAL	平方根 $\sqrt{\text{引数}}$	a:= SQRT (b) (* 変数 b の平方根を変数 a に格納 *)
LN(引数)	REAL LREAL		REAL LREAL	自然対数 LOG _e 引数	a:= LN (b) (* 変数 b の自然対数を変数 a に格納 *)
LOG(引数)	REAL LREAL		REAL LREAL	常用対数 LOG ₁₀ 引数	a:= LOG (b) (* 変数 b の常用対数を変数 a に格納 *)
EXP(引数)	REAL LREAL		REAL LREAL	引数 自然指数 e	a:= EXP (b) (* 変数 b の自然指数を変数 a に格納 *)
SIN(引数)	REAL LREAL		REAL LREAL	正弦 SIN 引数	a:= SIN (b) (* 変数 b の SIN を変数 a に格納 *)
COS(引数)	REAL LREAL		REAL LREAL	余弦 COS 引数	a:= COS (b) (* 変数 b の COS を変数 a に格納 *)
TAN(引数)	REAL LREAL		REAL LREAL	正接 TAN 引数	a:= TAN (b) (* 変数 b の TAN を変数 a に格納 *)
ASIN(引数)	REAL LREAL		REAL LREAL	逆正弦 SIN ⁻¹ 引数	a:= ASIN (b) (* 変数 b の ASIN を変数 a に格納 *)
ACOS(引数)	REAL LREAL		REAL LREAL	逆余弦 COS ⁻¹ 引数	a:= ACOS(b) (* 変数 b の ACOS を変数 a に格納 *)
ATAN(引数)	REAL LREAL		REAL LREAL	逆正接関数 TAN ⁻¹ 引数	a:= ATAN(b) (* 変数 b の ATAN を変数 a に格納 *)
EXPT(<底部>,<指数部>)	底部	REAL LREAL	REAL LREAL	指数 底部指数部	a:= EXPT (b,c) (* 変数 b を底部、変数 c を指数部とした指数を変数 a に格納 *)
	指数部	INT DINT LINT UINT UDINT ULINT REAL LREAL			
MOD(<除算対象データ>,<除算データ>)	除算対象データ	INT UINT UDINT ULINT DINT LINT	INT UINT UDINT ULINT DINT LINT	剰余	a := MOD(b, c) (* 変数 b を変数 c で割った剰余を変数 a に格納*)
	除算データ	INT UINT UDINT ULINT DINT LINT			

注: 数値演算関数では、引数と同じデータ型が返ります。したがって、関数の戻り値を代入する変数は、引数と同じデータ型とすることが必要です。

■ 文字列関数

以下の関数は CS/CJ シリーズユニットVer.4.0 以降、およびCJ2 シリーズのCPU ユニットで使用することができます。

関数名	引数のデータ型		戻り値のデータ型	内容	例
LEN(文字列)	文字列	STRING	INT	文字列長の抽出	a:= LEN (b) (* 変数 b の文字列長を変数 a に格納 *)
LEFT(<抽出対象文字列>, <抽出文字数>)	抽出対象文字列	STRING	STRING	左からの文字列抽出	a:= LEFT(b,c) (* 変数 b の文字列の左端から変数 c で指定した文字数分だけ変数 a に格納 *)
	抽出文字数	INT UINT			
RIGHT(<抽出対象文字列>, <抽出文字数>)	抽出対象文字列	STRING	STRING	右からの文字列抽出	a:= RIGHT(b,c) (* 変数 b の文字列の右端から変数 c で指定した文字数分だけ変数 a に格納 *)
	抽出文字数	INT UINT			
MID(<抽出対象文字列>, <抽出文字数>, <抽出位置>)	抽出対象文字列	STRING	STRING	文字列の抽出	a:= MID(b,c,d) (* 変数 b の文字列の変数 d で指定した位置から変数 c で指定した文字数分だけ変数 a に格納 *)
	抽出文字数	INT UINT			
	抽出位置	INT UINT			
CONCAT(<結合文字列 1>, <結合文字列 2>, ...) *最大 31 個まで指定可能	結合文字列	STRING	STRING	文字列の結合	a:= CONCAT(b,c,...) (* 変数 b、変数 c の文字列を結合して変数 a に格納 *)
INSERT(<挿入対象文字列>, <挿入文字列>, <挿入位置>)	挿入対象文字列	STRING	STRING	文字列の挿入	a:= INSERT(b,c,d) (* 変数 b の変数 d で指定した位置に変数 c の文字列を挿入して変数 a に格納 *)
	挿入文字列	STRING			
	挿入位置	INT UINT			
DELETE(<削除対象文字列>, <削除文字数>, <削除位置>)	削除対象文字列	STRING	STRING	文字列の削除	a:= DELETE(b,c,d) (* 変数 b の変数 d で指定した位置から変数 c で指定した文字数分だけ削除して変数 a に格納 *)
	削除文字数	INT UINT			
	削除位置	INT UINT			
REPLACE(<置換対象文字列>, <置換文字列>, <置換文字数>, <置換位置>)	置換対象文字列	STRING	STRING	文字列の置換	a:= REPLACE(b,c,d,e) (* 変数 b の変数 e で指定した位置から変数 d で指定した文字数分だけ変数 c で置換して変数 a に格納 *)
	置換文字列	STRING			
	置換文字数	INT UINT			
	置換位置	INT UINT			
FIND(<検索対象文字列>, <検索文字列>)	検索対象文字列	STRING	INT	文字列の検索	a:= FIND(b,c) (* 変数 b の文字列から変数 c で指定された文字列を検索し、最初に見つかった文字位置を変数 a に格納。見つからなかった場合は変数 a に 0 を格納。*)
	検索文字列	STRING			

■ データ型変更関数

ST 言語内で、以下のデータ型変更関数を使用可能です。

● 構文:

変更元のデータ型_TO_変更先データ型(変数名)

例) REAL_TO_INT(C)

変数 C のデータ型を、REAL 型からINT 型へ変更します。

変数 C の値の小数点以下は、五捨五入して整数に丸めます。五捨五入とは下の表のような処理のことです。

小数部の値	処理	例	
0.5 より小さい	切り捨て	1.49 → 1	-1.49 → -1
0.5	1 の位の値が偶数: 切り捨て 1 の位の値が奇数: 切り上げ	0.5 → 0 1.5 → 2 2.5 → 2 3.5 → 4	-0.5 → 0 -1.5 → -2 -2.5 → -2 -3.5 → -4
0.5 より大きい	切り上げ	1.51 → 2	-1.51 → -2

● データ型の組み合わせ

データ型の変更元 (FROM) と変更先 (TO) の組み合わせの可否は以下のとおりです。

○: 可能、×: 不可

TO FROM	BOOL	INT	DINT	LINT	UINT	UDINT	ULINT	WORD	DWORD	LWORD	REAL	LREAL	BCD_ WORD	BCD_ DWORD
BOOL	×	×	×	×	×	×	×	×	×	×	×	×	×	×
INT	×	×	○	○	○	○	○	○	○	○	○	○	○	○
DINT	×	○	×	○	○	○	○	○	○	○	○	○	○	○
LINT	×	○	○	×	○	○	○	○	○	○	○	○	×	×
UINT	×	○	○	○	×	○	○	○	○	○	○	○	○	○
UDINT	×	○	○	○	○	×	○	○	○	○	○	○	○	○
ULINT	×	○	○	○	○	○	×	○	○	○	○	○	×	×
WORD	×	○	○	○	○	○	○	×	○	○	×	×	×	×
DWORD	×	○	○	○	○	○	○	○	×	○	×	×	×	×
LWORD	×	○	○	○	○	○	○	○	○	×	×	×	×	×
REAL	×	○	○	○	○	○	○	×	×	×	×	○	×	×
LREAL	×	○	○	○	○	○	○	×	×	×	○	×	×	×
WORD_ BCD	×	○	○	×	○	○	×	×	×	×	×	×	×	×
DWORD_ BCD	×	○	○	×	○	○	×	×	×	×	×	×	×	×

■ 数値文字列変更関数

ST 言語内で、以下の数値文字列変更関数を使用可能です。

● 構文:

変更元のデータ型_TO_STRING(変数名)

例) INT_TO_STRING(C)

変数 C のデータ型を、INT 型からSTRING 型へ変更します。

STRING_TO_変更先のデータ型(変数名)

例) STRING_TO_INT(C)

変数 C のデータ型を、STRING 型からINT 型へ変更します。

● データ型の組み合わせ

データ型の変更元 (FROM) と変更先 (TO) の組み合わせの可否は以下のとおりです。

○:可能、×:不可

TO FROM	BOOL	INT	DINT	LINT	UINT	UDINT	ULINT	WORD	DWORD	LWORD	REAL	LREAL	STRING
BOOL	×	×	×	×	×	×	×	×	×	×	×	×	×
INT	×	×	○	○	○	○	○	○	○	○	○	○	○
DINT	×	○	×	○	○	○	○	○	○	○	○	○	○
LINT	×	○	○	×	○	○	○	○	○	○	○	○	×
UINT	×	○	○	○	×	○	○	○	○	○	○	○	○
UDINT	×	○	○	○	○	×	○	○	○	○	○	○	○
ULINT	×	○	○	○	○	○	×	○	○	○	○	○	×
WORD	×	○	○	○	○	○	○	×	○	○	×	×	○
DWORD	×	○	○	○	○	○	○	○	×	○	×	×	○
LWORD	×	○	○	○	○	○	○	○	○	×	×	×	×
REAL	×	○	○	○	○	○	○	○	○	○	×	×	×
LREAL	×	○	○	○	○	○	○	○	○	○	×	×	×
STRING	×	○	○	×	○	○	×	○	○	×	×	×	×

■ データシフト関数

関数名	第 1 引数のデータ型	第 2 引数のデータ型	戻り値のデータ型	内容	例
SHL(<シフト対象データ>, <ビット数>)	BOOL, WORD, DWORD, LWORD	INT, UINT, UDINT, ULINT, DINT, LINT	BOOL, WORD, DWORD, LWORD	ビット文 字列を左へ n ビットシフトします。シフトしたとき、右にはゼロが入ります。	a := SHL(b, c) (*ビット文字列 b を左へ c ビットシフトした結果を a に格納*)
SHR(<シフト対象データ>, <ビット数>)	BOOL, WORD, DWORD, LWORD	INT, UINT, UDINT, ULINT, DINT, LINT	BOOL, WORD, DWORD, LWORD	ビット文 字列を右へ n ビットシフトします。シフトしたとき、左にはゼロが入ります。	a := SHR(b, c) (*ビット文字列 b を右へ c ビットシフトした結果を a に格納*)
ROL(<回転対象データ>, <ビット数>)	BOOL, WORD, DWORD, LWORD	INT, UINT, UDINT, ULINT, DINT, LINT	BOOL, WORD, DWORD, LWORD	ビット文 字列を左へ n ビット回転します。	a := ROL(b, c) (*ビット文字列 b を左へ c ビット回転した結果を a に格納*)
ROR(<回転対象データ>, <ビット数>)	BOOL, WORD, DWORD, LWORD	INT, UINT, UDINT, ULINT, DINT, LINT	BOOL, WORD, DWORD, LWORD	ビット文 字列を右へ n ビット回転します。	a := ROR(b, c) (*ビット文字列 b を右へ c ビット回転した結果を a に格納*)

■ データ制御関数

関数名	第 1 引数のデータ型	第 2 引数のデータ型	第 3 引数のデータ型	戻り値のデータ型	内容	例
LIMIT(<下限リミットデータ>, <入力データ>, <上限リミットデータ>)	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	入力データが上下限リミットデータの範囲内かどうかにより、出力データを制御します。	a := LIMIT(b, c, d) (*c < b のとき b を、b ≤ c ≤ d のとき c を、d < c のとき d を、a に格納*)

■ データ選択関数

関数名	第 1 引数のデータ型	第 2 引数のデータ型	第 3 引数のデータ型*1	戻り値のデータ型	内容	例
SEL(< 選択条件>, < 選択対象データ 1>, < 選択対象データ 2>)	BOOL	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	2つのデータから選択条件で指定された1つのデータを選択します。	a := SEL(b, c, d) (*c と d のうち b が偽ならば c を、真ならば d を a に格納する*)
MUX(<抽出条件>, < 抽出対象データ 1>, <抽出対象データ 2>, ...)	INT, UINT, UDINT, ULINT, DINT, LINT	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	最大 30 個のデータから抽出条件で指定された1つのデータを選択します。	a := MUX(b, c, d,...) (*c、d、...の中から b+1 番目のデータを a に格納します*)
MAX(<対象データ 1>, <対象データ 2>, <対象データ 3>, ...) *2	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	最大 31 個のデータから、最大値を選択します。	a := MAX(b, c, d,...) (*c、d、...の中から最大値を a に格納します。*)
MIN(< 対象データ 1>, <対象データ 2>, <対象データ 3>, ...) *2	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	BOOL, INT, UINT, UDINT, ULINT, DINT, LINT, WORD, DWORD, LWORD, REAL, LREAL	最大 31 個のデータから、最小値を選択します。	a := MIN(b, c, d,...) (*c、d、...の中から最小値を a に格納します。*)

*1: MUX は第3 引数以降、最大で第31 引数(最大30 個の抽出対象データ)まで指定可能です。

*2: MAX、MIN は、第1 引数から第31 引数(最大31 個の対象データ)の間で引数を指定可能です。

オムロン拡張関数

関数種類	
メモ리카ード処理関数	メモ리카ードへ書き込む関数
通信処理関数	文字列を送受信する関数
角度変換関数	角度とラジアンを変換する関数
タイマ/カウンタ関数	様々なタイマ/カウンタを実行する関数

■メモ리카ード処理関数

以下の関数はCS/CJシリーズユニットVer.4.0以降、およびCJ2シリーズのCPUユニットで 사용할 수 있습니다。

関数名	引数のデータ型		戻り値のデータ型	内容	例
WRITE_TEXT(<書出文字列>,<ディレクトリ名+ファイル名>,<区切り文字>,<パラメータ>)	書出文字列	STRING	—	メモ리카ードへの文字列書込み	WRITE_TEXT (a,b,c,d) (* 変数aで指定した文字列を変数bで指定したディレクトリにテキストファイルとして保存する。変数dが0の時は追加書込み、1の時はファイルを新規作成して書き込む *)
	ディレクトリ名+ファイル名	STRING			
	区切り文字	STRING			
	パラメータ	INT UINT WORD			

■通信処理関数

以下の関数はCS/CJシリーズユニットVer.4.0以降、およびCJ2シリーズのCPUユニットで使用することができます。

関数名	引数のデータ型		戻り値のデータ型	内容	例
TXD_CPU(<送信文字列>)	送信文字列	STRING	—	文字列の送信 (CPUユニットのRS-232Cポート)	TXD_CPU(a) (* 変数aで指定した文字列をCPUユニットのRS-232Cポートから送信する *)
TXD_SCB(<送信文字列>,<シリアルポートNo.>)	送信文字列	STRING	—	文字列の送信 (SCBのシリアルポート)	TXD_SCB(a,b) (* 変数aで指定した文字列を変数bで指定したSCBのシリアルポートから送信する *)
	シリアルポートNo	INT UINT WORD			
TXD_SCU(<送信文字列>,<指定SCUユニット番号>,<シリアルポートNo.>,<内部論理ポートNo.>)	送信文字列	STRING	—	文字列の送信 (SCUのシリアルポート)	TXD_SCU(a,b,c,d) (* 変数aで指定した文字列を変数bで指定したユニット番号のSCUの変数cで指定したシリアルポートから送信する。変数dは内部論理ポート番号です *)
	指定SCUユニット番号	INT UINT WORD			
	シリアルポートNo	INT UINT WORD			
	内部論理ポートNo	INT UINT WORD			
RXD_CPU(<受信文字列格納先>,<受信文字数>)	受信文字列格納先	STRING	—	文字列の受信 (CPUユニットのRS-232Cポート)	RXD_CPU(a,b) (* CPUユニットのRS-232Cポートに受信したデータを変数bの文字数分の文字列として変数aに出力する *)
	受信文字数	INT UINT WORD			
RXD_SCB(<受信文字列格納先>,<受信文字数>,<シリアルポートNo.>)	受信文字列格納先	STRING	—	文字列の受信 (SCBのシリアルポート)	RXD_SCB(a,b,c) (* 変数bで指定したSCBのシリアルポートに受信したデータを変数cの文字数分の文字列として変数aに出力する *)
	受信文字数	INT UINT WORD			
	シリアルポートNo	INT UINT WORD			
RXD_SCU(<受信文字列格納先>,<受信文字数>,<指定SCUユニット番号>,<シリアルポートNo.>,<内部論理ポートNo.>)	受信文字列格納先	STRING	—	文字列の受信 (SCUのシリアルポート)	RXD_SCU(a,b,c,d,e) (* 変数cで指定したユニット番号のSCUの変数dで指定したシリアルポートに受信したデータを変数bの文字数分の文字列として変数aに出力する。変数eは内部論理ポート番号です *)
	受信文字数	INT UINT WORD			
	指定SCUユニット番号	INT UINT WORD			
	シリアルポートNo	INT UINT WORD			
	内部論理ポートNo	INT UINT WORD			

■角度変換関数

以下の関数はCS/CJシリーズユニットVer.4.0以降、およびCJ2シリーズのCPUユニットで使用することができます。

関数名	引数のデータ型	戻り値のデータ型	内容	例
DEG_TO_RAD (引数)	REAL LREAL	REAL LREAL	角度→ラジアン変換	a:= DEG_TO_RAD(b) (* 変数b(角度)をラジアン に変換して変数aに格納 *)
RAD_TO_DEG (引数)	REAL LREAL	REAL LREAL	ラジアン→角度変換	a:= RAD_TO_DEG(b) (* 変数b(ラジアン)を角度 に変換して変数aに格納 *)

■タイマ/カウンタ関数

以下の関数は、CJ2 シリーズのCPU ユニットで使用することができます。

関数名	引数のデータ型		戻り値のデータ型	内容	例
TIMX(<実行条件>, <タイマアドレス>,<タイマ設定値>)	実行条件	BOOL	—	名称:100msタイマ 動作:減算式ONディレー 100ms単位のタイマの動作をします。	TIMX(a, b, c) (*実行条件aが成立した時点で、タイマアドレスb、設定値cのTIMXタイマの実行を開始します。*)
	タイマアドレス	TIMER			
	タイマ設定値	UINT			
TIMHX(<実行条件>,<タイマアドレス>,<タイマ設定値>)	実行条件	BOOL	—	名称:10msタイマ(高速タイマ) 動作:減算式ONディレー 10ms単位のタイマの動作をします。	TIMHX(a, b, c) (*実行条件aが成立した時点で、タイマアドレスb、設定値cのTIMHXタイマの実行を開始します。*)
	タイマアドレス	TIMER			
	タイマ設定値	UINT			
TMHHX(<実行条件>,<タイマアドレス>,<タイマ設定値>)	実行条件	BOOL	—	名称:1msタイマ(超高速タイマ) 動作:減算式ONディレー 1ms単位のタイマの動作をします。	TMHHX(a, b, c) (*実行条件aが成立した時点で、タイマアドレスb、設定値cのTMHHXタイマの実行を開始します。*)
	タイマアドレス	TIMER			
	タイマ設定値	UINT			
TIMUX(<実行条件>,<タイマアドレス>,<タイマ設定値>)	実行条件	BOOL	—	名称:0.1msタイマ 動作:減算式ONディレー 0.1ms単位のタイマの動作をします。	TIMUX(a, b, c) (*実行条件aが成立した時点で、タイマアドレスb、設定値cのTIMUXタイマの実行を開始します。*)
	タイマアドレス	TIMER			
	タイマ設定値	UINT			
TMUHX(<実行条件>,<タイマアドレス>,<タイマ設定値>)	実行条件	BOOL	—	名称:0.01msタイマ 動作:減算式ONディレー 0.01ms単位のタイマの動作をします。	TMUHX(a, b, c) (*実行条件aが成立した時点で、タイマアドレスb、設定値cのTMUHXタイマの実行を開始します。*)
	タイマアドレス	TIMER			
	タイマ設定値	UINT			
TTIMX(<実行条件>,<リセット入力>,<タイマアドレス>,<タイマ設定値>)	実行条件	BOOL	—	名称:積算タイマ 動作:積算式ONディレー 0.1s単位のタイマの動作をします。	TTIMX(a, b, c, d) (*実行条件aがONの間、タイマアドレスc、設定値dのTTIMXタイマを実行します。リセット入力bがONの場合に、タイマ現在値、アップフラグがリセットされます。*)
	リセット入力	BOOL			
	タイマアドレス	TIMER			
	タイマ設定値	UINT			
CNTX(<カウンタ入力>,<リセット入力>,<カウンタアドレス>,<カウンタ設定値>)	カウンタ入力	BOOL	—	名称:カウンタ 動作:減算カウンタの動作をします。	CNTX(a, b, c, d) (*カウンタ入力aの立ち上がりごとに、カウンタアドレスc、カウンタ設定値dのCNTXカウンタを実行します。リセット入力bがONのとき、カウンタ現在値、アップフラグがリセットされます。*)
	リセット入力	BOOL			
	カウンタアドレス	COUNTER			
	カウンタ設定値	UINT			

関数名	引数のデータ型		戻り値のデータ型	内容	例
CNTRX(<加算カウント>, <減算カウント>, <リセット入力>, <カウンタアドレス>, <カウンタ設定値>)	加算カウント	BOOL	—	名称: 可逆カウンタ 動作: 加減算カウンタの動作をします。	CNTRX(a, b, c, d, e) (*カウンタアドレスd、カウンタ設定値eのCNTRXを実行します。加算カウント入力aの立ち上がりで現在値を加算し、減算カウント入力bの立ち上がりで減算します。リセット入力cがONのとき、カウンタ現在値、アップフラグがリセットされます。*)
	減算カウント	BOOL			
	リセット入力	BOOL			
	カウンタアドレス	COUNTER			
	カウンタ設定値	UINT			
TRSET(<実行条件>, <タイマアドレス>)	実行条件	BOOL	—	名称: タイマリセット 動作: 指定タイマをリセットします。	TRSET(a, b) (*実行条件aがONのときにタイマアドレスbのタイマをリセットします。*)
	タイマアドレス	TIMER			

本誌には主に機種のご選定に必要な内容を掲載し、ご使用上の注意事項等は掲載しておりません。
ご使用上の注意事項等、ご使用の際に必要な内容につきましては、必ずユーザーズマニュアルをお読みください。

- 本誌に記載の標準価格はあくまで参考であり、確定されたユーザ購入価格を表示したものではありません。本誌に記載の標準価格には消費税が含まれておりません。
- 本誌に記載されているアプリケーション事例は参考用ですので、ご採用に際しては機器・装置の機能や安全性をご確認の上、ご使用ください。
- 本誌に記載のない条件や環境での使用、および原子力制御・鉄道・航空・車両・燃焼装置・医療機器・娯楽機械・安全機器、その他人命や財産に大きな影響が予測されるなど、特に安全性が要求される用途への使用をご検討の場合は、定格・性能に対し余裕を持った使い方やフェールセーフ等の安全対策へのご配慮をいただくとともに、当社営業担当者までご相談いただき仕様書等による確認をお願いします。
- 本製品の内、外国為替及び外国貿易法に定める輸出許可、承認対象貨物(又は技術)に該当するものを輸出(又は非居住者に提供)する場合は同法に基づく輸出許可、承認(又は役務取引許可)が必要です。

オムロン株式会社 インダストリアルオートメーションビジネスカンパニー

- 製品に関するお問い合わせ先
お客様相談室

クイック オムロン
 **0120-919-066**

携帯電話・PHS・IP電話などではご利用いただけませんので、下記の電話番号へおかけください。

電話 **055-982-5015** (通話料がかかります)

■営業時間：8:00～21:00 ■営業日：365日

- FAXやWebページでもお問い合わせいただけます。

FAX **055-982-5051** / www.fa.omron.co.jp

- その他のお問い合わせ

納期・価格・サンプル・仕様書は貴社のお取引先、または貴社担当オムロン販売員にご相談ください。
オムロン制御機器販売店やオムロン販売拠点は、Webページでご案内しています。

オムロン制御機器の最新情報をご覧ください。

www.fa.omron.co.jp

緊急時のご購入にもご利用ください。

オムロン商品のご用命は