86LDmicro ユーザーマニュアル

86Duino IDE 502

対象ソフトウェア：86LDmicro 2.0.0

プラットフォーム：QEC-M-02（Vortex86EX2）

IDE：86Duino Coding 502 Preview（20261105）

対象読者：エンジニア／技術者

バージョン：v2.0

発行日：2026 年 11 月

改訂履歴

バージョン	日付	主な変更点
v2.0	2026-11	86LDmicro 2.0.0 初版：IEC 61131-3（LD、ST、SFC）、EtherCAT、Motion86、86HMI、オンラインモニター

目次

改訂履歴 ii
目次 iii
第 1 章　はじめに 1
1.1　本マニュアルについて 1
1.2　86LDmicro について 4
1.3　86Duino と QEC-M-02 7
1.4　LDmicro 1.1 との主な違い 8
第 2 章　環境のインストール 10
2.1　システム要件 10
2.2　86Duino Coding 502 のインストール 11
2.3　86LDmicro ツールのインストール 13
2.4　ハードウェア接続と COM ポート 15
2.5　86LDmicro の起動 19
2.6　オプション 25
第 3 章　インターフェースの概要 27
3.1　メインウィンドウとステータスバー 27
3.2　メインメニュー 29
3.3　ツールバー 39
3.4　プロジェクトツリー 40
3.5　POU エディターとサイドパネル 44
3.6　ツールウィンドウ 47
3.7　タブ、左右分割、ズーム、検索 51
第 4 章　プロジェクトと設定 56
4.1　プロジェクトのファイル構成 56
4.2　プロジェクト設定 58
4.3　タスク 60
4.4　デバイス 63
4.5　グローバル変数とデータ型 66
4.6　I/O マッピング 70
4.7　POU とユーザーコードの追加 75
第 5 章　クイックスタート：白紙からアップロードまで 78
5.1　ステップ 1：86Duino IDE を開き、sketch を作成し、ボードとシリアルポートを選択する 79
5.2　ステップ 2：IDE から 86LDmicro を開き、新しいプロジェクトを作成する 80
5.3　ステップ 3：新しいプロジェクトを確認する：プロジェクトツリーとラダー図 Main 81
5.4　ステップ 4：GVL で変数を宣言する 82
5.5　ステップ 5：a 接点 Xstart を挿入する 83
5.6　ステップ 6：b 接点 Xstop とコイル Rrun を直列に追加する 84
5.7　ステップ 7：Rrun の接点を並列に追加して自己保持回路を作る 85

5.8 ステップ 8：ネットワーク 2 を追加：TON タイマーで 2 秒遅れてランプを点灯 86
5.9 ステップ 9：I/O マッピング：変数を QEC-M-02 の物理 I/O に接続する 88
5.10 ステップ 10：チェック（Shift+F7）とシミュレーション（F5） 90
5.11 ステップ 11：ビルド、コンパイル、アップロード（F8） 93
5.12 ステップ 12：オンラインモニター（Ctrl+F5） 95

第 6 章 実践例 97

6.0 共通原則（最初にお読みください） 97
6.1 起動／停止の自己保持 102
6.2 遅延消灯（TOF） 105
6.3 梱包カウンター（CTU＋比較） 107
6.4 流れる表示灯（交互点滅） 110
6.5 ST：注水制御 112
6.6 SFC：シーケンス制御 115
6.7 C/C++：Serial1 でメッセージを送信 119

第 7 章 ラダー図命令リファレンス 121

7.1 編集操作 122
7.2 接点とコイル 129
7.3 エッジとパルス 131
7.4 タイマー 133
7.5 カウンター 135
7.6 比較 136
7.7 演算と転送 137
7.8 IEC ファンクションブロック（CALL）と ST 命令 140
7.9 プログラムフロー 142
7.10 I/O、シリアルポート、その他 143
7.11 モーション制御命令 145

第 8 章 ST と SFC 147

8.1 ST エディター 147
8.2 ST 構文クイックリファレンス 152
8.3 ユーザー定義の FUNCTION と FUNCTION_BLOCK 155
8.4 SFC エディター 159
8.5 ステップ、トランジション、分岐 162
8.6 SFC の実行セマンティクス 169

第 9 章 シミュレーションとデバッグ 171

9.1 シミュレーション 171
9.2 Simulation I/O 175
9.3 ウォッチ、書き込み、強制 177
9.4 Trace 180
9.5 Cross reference と検索 181
9.6 よくあるエラーメッセージ 184

第 10 章 ビルド、アップロード、オンラインモニター 187

10.1 チェック、ビルド、コンパイル、アップロード 187
10.2 生成されるファイル 191
10.3 シリアルモニター 193
10.4 オンラインモニター 194
10.5 Logbook とウォッチドッグ 198
10.6 86HMI リソースのアップロード 201

第 11 章 EtherCAT 203

11.1 概念 203
11.2 全体の流れ 205
11.3 EtherCAT Configurator 213
11.4 ENI の読み込みと変数のマッピング 231
11.5 SubDevice ページ 235
11.6 PDO のオンライン値、SDO と CoE 238
11.7 トラブルシューティング 242

第 12 章 Motion86 モーション制御 243

12.1 概念 243
12.2 マシンの作成 244
12.3 マシンの設定 247
12.4 軸の設定 249
12.5 マシンの試運転 252
12.6 単軸の試運転 254
12.7 PLCopen MC ファンクションブロック 256
12.8 M86 マシンファンクションブロック 262
12.9 安全機構 266

第 13 章 86HMI ヒューマンマシンインターフェース 269

13.1 86HMI を有効にする 269
13.2 ウィジェットのバインド 275
13.3 ウィジェットと操作方法 280
13.4 テキスト、トレンド、アラーム、ページ切り替え 284
13.5 シミュレーションとオンライン 287
13.6 リソースのアップロードと言語 289

第 14 章 C/C++ と 86Duino の統合 291

14.1 ユーザーコードファイル 291
14.2 PLC 変数へのアクセス 293
14.3 フック関数 296
14.4 例 297
14.5 sketch の .ino 301
14.6 注意事項 303

第 15 章 インポート、AI、コマンドラインとプロジェクト保護 304

15.1 LDmicro 1.1 プロジェクトのインポート 304
15.2 PLCopen XML のインポート 307

15.3 AI アシスタント 309
15.4 MCP 314
15.5 コマンドラインツール 316
15.6 プロジェクト保護 318

第 16 章 よくある質問 320
付録 A LDmicro 1.1 との対照 328
付録 B ショートカットキー 330

B.1 メインウィンドウ（どのタブでも） 330
B.2 検索バーとプロジェクトツリー 331
B.3 ラダー図エディター 331
B.4 SFC エディター 333
B.5 コードエディター（ST、GVL、データ型、C/C++、sketch） 333
B.6 AI アシスタント 333

付録 C 変数の命名と型 334

C.1 命名の慣習 334
C.2 命名規則 334
C.3 データ型と範囲 335
C.4 定数と TIME の書き方 336

お問い合わせ 337

第 1 章　はじめに

1.1　本マニュアルについて

本マニュアルは **86LDmicro 2.0.0** の完全なユーザーマニュアル（バージョン v2.0）です。86Duino Coding 502 Preview（20261105）で 86LDmicro をインストール・起動し、IEC 61131-3 のラダー図（LD）、ストラクチャードテキスト（ST）、シーケンシャルファンクションチャート（SFC）および C/C++ で PLC プログラムを作成して、PC 上でシミュレーションし、QEC-M-02 コントローラーにアップロードしてオンラインでモニターする方法を説明します。EtherCAT、Motion86 モーション制御、86HMI ヒューマンマシンインターフェースにもそれぞれ 1 章を設けています。

本マニュアルは日本語で説明し、主な操作画面のスクリーンショットには 86LDmicro 2.0.0 の英語インターフェースを使用しています。画面上の名称は英語表示のまま記載します（例：ウォッチ（Watch）、シミュレーション I/O（Simulation I/O）、タスク（Tasks）、データ型（Data types））。章の構成は次のとおりです。

章	内容	対象
第 1〜2 章	はじめに、環境のインストール（IDE、86LDmicro、ハードウェア接続、起動、オプション）	初めて使う方
第 3〜4 章	インターフェースの概要（各メニュー、ツールバー、プロジェクトツリー、ツールウィンドウ）、プロジェクトと設定（タスク、デバイス、変数、I/O マッピング）	初めて使う方、リファレンス
第 5 章	クイックスタート：空の sketch からアップロードとオンラインモニターまで、全 12 ステップ	初めて使う方
第 6 章	実習例：自己保持、遅延消灯、カウント、流れるランプ、ST、SFC、C/C++	練習
第 7〜8 章	ラダー図命令リファレンス、ST と SFC	プログラム作成
第 9〜10 章	シミュレーションとデバッグ、ビルド・アップロードとオンラインモニター	デバッグ、オンライン運用
第 11〜13 章	EtherCAT、Motion86 モーション制御、86HMI ヒューマンマシンインターフェース	高度な機能
第 14〜15 章	C/C++ と 86Duino の統合；インポート、AI、コマンドラインとプロジェクト保護	高度な機能
第 16 章、付録	よくある質問；LDmicro 1.1 との対照、ショートカットキー、変数の命名と型	リファレンス

***ヒント：コントローラーがなくても試すことができます。2.2、2.3 節のインストールが終わったら、そのまま第 5 章のステップ 1～10 を行い（ステップ 1 のシリアルポートはひとまず省略できます）、PC 上で最初のプログラムを作成・チェックしてシミュレーションしてください。**

QEC コントローラーを接続するときに、2.4 節のハードウェア接続と第 5 章のステップ 11、12（ビルドとアップロード、オンラインモニター）を行います。

1.1.1 本マニュアルの読み方と表記

- **太字**：画面上のボタン、メニュー項目、フィールド名。例：**OK** をクリックします。
- 等幅フォント：入力する文字列、変数名、ファイル名、フォルダー。例：`Xstart`、`data\86ld`。
- 「Project > Add a PROGRAM (ST)...」：メニューを順にクリックします。ショートカットキーは Ctrl+B、Shift+F7 のように表記します。
- `<IDE フォルダー>`：86Duino IDE のあるフォルダー（`86duino.exe` がある階層）。`<sketch>`：86Duino sketch のフォルダー。
- **ヒント**、**注意**、**重要**、**安全**、**新機能** の枠：補足説明、間違えやすい点、安全上の注意事項。

サンプルプロジェクト：第 5 章のクイックスタートプロジェクト `MyPlc`、第 6 章の各サンプル、および総合サンプル `tools\example\ex30_dispense`。

*ヒント：最新のドキュメントは、86LDmicro の「Help」メニューから開くオンラインドキュメントサイト https://www.qec.tw/86duino/86ldmicro/ を参照してください。オフラインのドキュメントは `<IDE フォルダー>\tools\LDmicro\tool\docs` にあります。

1.2　86LDmicro について

86LDmicro 2.0 は、ICOP が 86Duino QEC シリーズコントローラー向けに開発した PLC 開発環境です。LDmicro 1.1 のラダー図の操作感を受け継ぎつつ、IEC 61131-3 のプロジェクト構成（POU、タスク、グローバル変数、データ型）に基づいて再設計されています。PLC に慣れたエンジニアはすぐに使いこなせ、86Duino ユーザーも PLC の方法でマシンを制御できます。

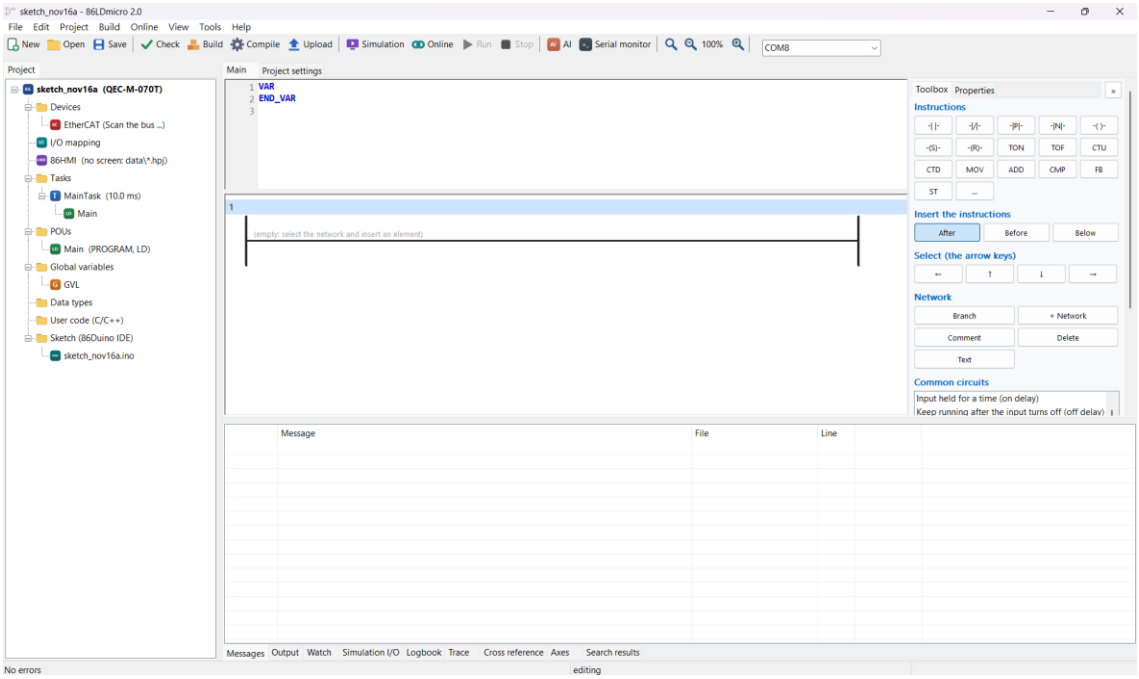


図 1-0　86LDmicro 2.0.0 インターフェース

1 行ずつ順に実行される一般的な C プログラムとは異なり、PLC プログラムは「スキャン周期」で実行されます。

> 各周期：すべての入力を読み込む（ローカル `DI`、`EtherCAT` 入力 `PDO` …）
> → タスク（`Task`）内の順序で各 `PROGRAM`（`LD`／`ST`／`SFC`）を実行
> → 出力変数を `DO`、`EtherCAT` 出力 `PDO` に書き込む
> → 次の周期を待つ（デフォルトでは `MainTask` は `10 ms` ごと）

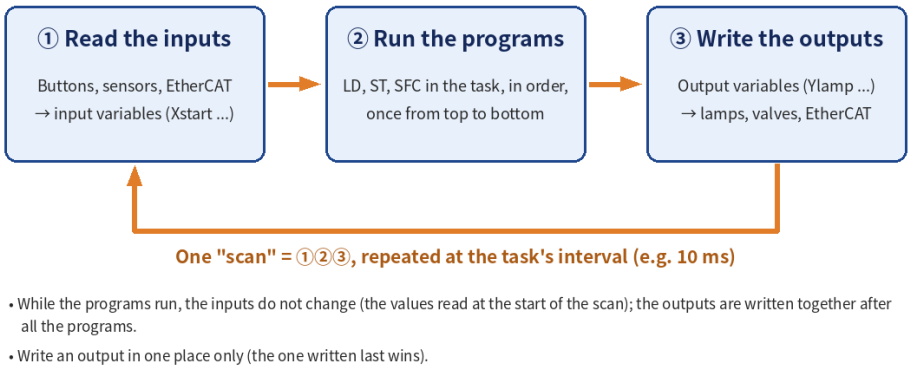


図 1-1　1 回のスキャン周期：入力の読み込み → プログラムの実行 → 出力の書き込み（概略図）

1.2.1　システム構成

86LDmicro はコントローラー上でプログラムを解釈実行するのではなく、PLC プロジェクト全体を 86Duino の C++ sketch に変換し、86Duino IDE のコンパイラーでコンパイルしてアップロードします。PC とコントローラーは 1 本の USB ケーブルで接続し、アップロードとオンラインモニターはこのケーブルを共用します。

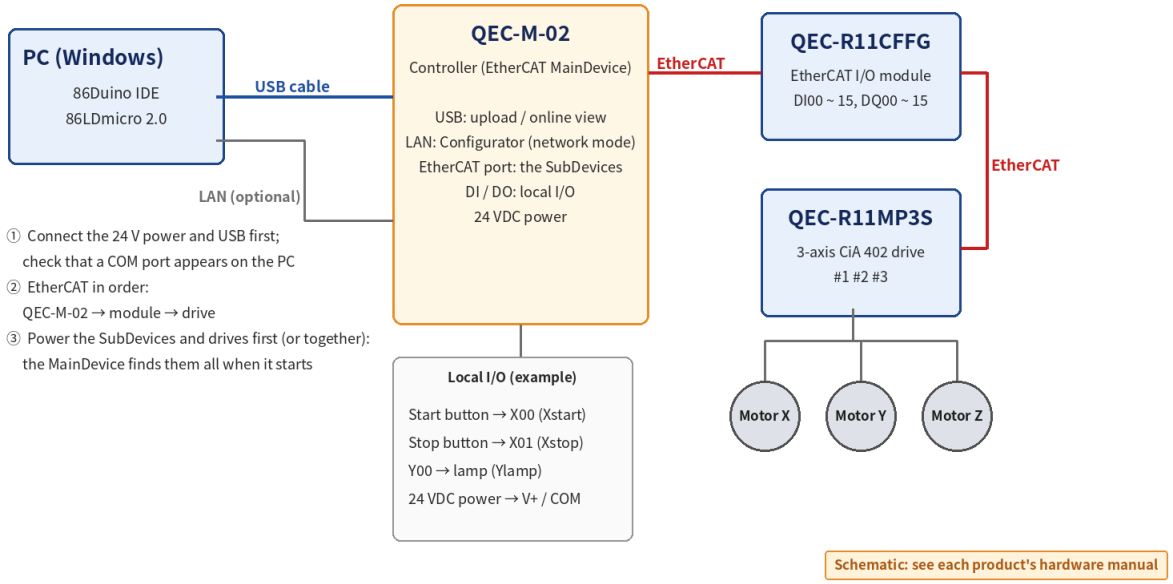


図 1-2　典型的なシステム：PC（86Duino IDE＋86LDmicro）を USB で QEC-M-02 に接続し、EtherCAT で I/O モジュールとドライブをデイジーチェーン接続（概略図）

ステップ	担当	成果物
1. 作成	ユーザーが 86LDmicro で LD／ST／SFC／C/C++ を編集し、デバイスと I/O を設定	`<sketch>\data\86ld\` 以下のプロジェクトファイル（テキストファイル）
2. ビルド	86LDmicro（Ctrl+B）	sketch 内の `plc86.cpp`、`plc86.h`、`plc86_rt.cpp`、`plc86_ecat.cpp` など
3. コンパイル	86Duino IDE の arduino-builder（F7）	コントローラーの実行ファイル
4. アップロード	USB 経由の v86dude（F8）	プログラムを QEC に書き込んで再起動
5. 実行	QEC 上の 86LD run-time	タスク周期でプログラムをスキャンし、I/O、EtherCAT、Motion86 を読み書き
6. モニター	同じ USB ケーブル経由の 86LDmicro（Ctrl+F5）	リアルタイム値、パワーフロー、強制、Logbook

最終的に C++ プログラムになるため、PLC プログラムは 86Duino／Arduino のライブラリ（EtherCAT、Motion86、Modbus、Ethernet、Serial……）と組み合わせて使用でき、C/C++ コードから PLC 変数を直接読み書きすることもできます（第 14 章）。

1.2.2　主な機能

分類	機能
プログラミング言語	ラダー図（LD）、ストラクチャードテキスト（ST）、シーケンシャルファンクションチャート（SFC）、C/C++ ユーザーコード；FUNCTION、FUNCTION_BLOCK の自作が可能
プロジェクト構成	タスク（周期／フリーラン／EtherCAT 周期）、グローバル変数リスト（GVL）、データ型（構造体、列挙型、配列）、RETAIN（停電保持）
I/O と通信	QEC ローカル DI／DO、EtherCAT（ENI）、86EVA、Modbus RTU／TCP、Ethernet、シリアルポート
モーション制御	Motion86 マシン（CiA 402 ドライブ CSP、最大 9 軸）、PLCopen MC_* ファンクションブロック、M86_* パスと G-code、試運転ページ
デバッグ	PC 上での完全なシミュレーション（I/O、タイマー、軸を含む）、オンラインモニター、書き込み／強制、パワーフロー、Trace、Logbook、シリアルモニター
EtherCAT ツール	組み込みの EtherCAT Configurator、SubDevice ページ（PDO のリアルタイム値、書き込み／強制）、CoE オブジェクトディクショナリ
ヒューマンマシンインターフェース	86HMI の画面バインド、インタラクション設定、シミュレーションウィンドウ、リソースのアップロード
その他	I/O 表の Excel インポート／エクスポート、プロジェクトのパスワード保護、左右並べての編集、繁体字中国語／英語／日本語インターフェース、AI アシスタントと MCP、コマンドラインツール、LDmicro 1.1 および PLCopen XML のインポート

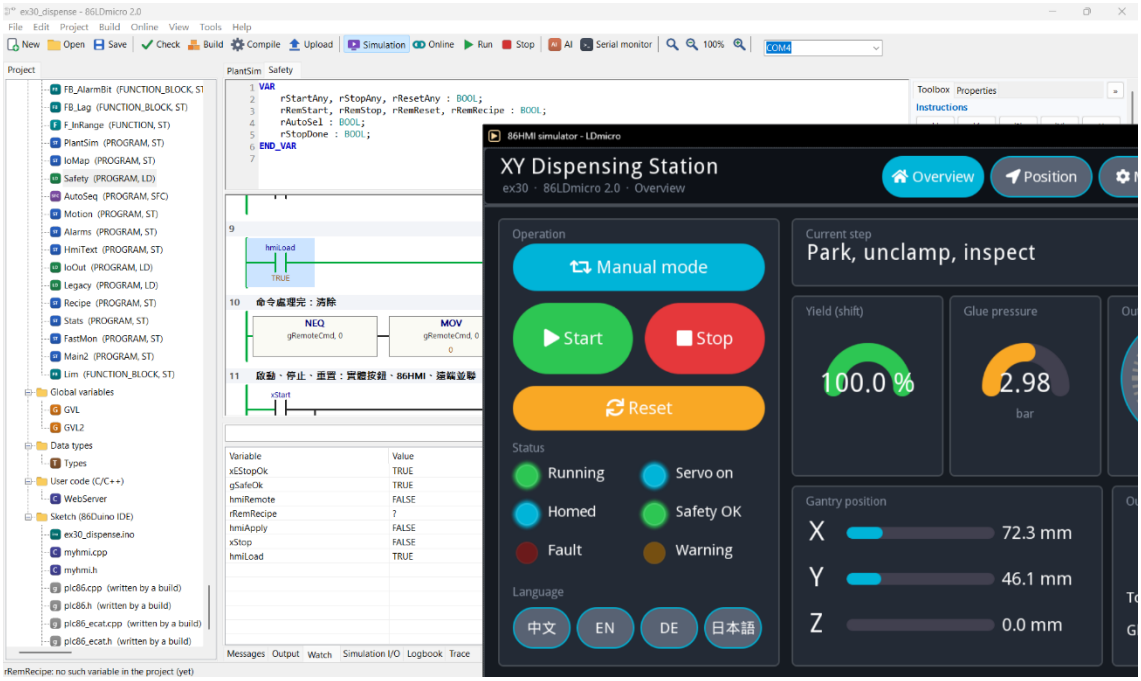


図 1-3　86LDmicro 2.0：プロジェクトツリー、ラダー図、ツールウィンドウ（シミュレーション中）

1.3　86Duino と QEC-M-02

86Duino は、DMP が x86 プロセッサー（Vortex86 シリーズ）をベースに開発した、Arduino 互換構文の開発プラットフォームです。QEC シリーズは 86Duino をコアとする産業用コントローラーで、QEC-M-02 は EtherCAT メインデバイスでもあります。86LDmicro が生成するプログラムは 86Duino sketch そのもので、86Duino Coding IDE でコンパイルして QEC 上で実行されます。



図 1-4　QEC-M-02 コントローラー

項目	仕様／説明
プロセッサー	DMP Vortex86EX2（32 ビット x86）
開発環境	86Duino Coding 502 Preview（20261105）、Arduino 互換構文；86LDmicro 2.0.0 は IDE の `tools\LDmicro\tool` にインストールされます
ローカル I/O	DI 16 点、DO 16 点；86LDmicro でのチャネルは `LocalIO.X00`〜`X17`（入力）、`LocalIO.Y00`〜`Y17`（出力）で、8 進数で番号付けされます
EtherCAT	EtherCAT メインデバイスポート：QEC EtherCAT モジュール（例：QEC-R11CFFG）と CiA 402 ドライブ（例：QEC-R11MP3S）をデイジーチェーン接続
USB	プログラムのアップロード、オンラインモニター、シリアルモニター、EtherCAT Configurator（USB モード）；PC 上では COM ポートとして表示されます（例：「QEC M02 on COM4」）
LAN	EtherCAT Configurator ネットワークモード（デフォルト 192.168.100.1）、Modbus TCP、Ethernet
シリアルポート	COM1〜COM3（RS-232／485）：Modbus RTU、ユーザーコードの Serial1〜Serial3
電源	24 VDC
86LDmicro との統合	IDE の「Tools > 86LDmicro」と IDE ツールバーの 86LDmicro ボタン（2.5 節）
その他のコントローラー	プロジェクト設定の「Controller」では QEC-M-01、QEC-M-043T、QEC-M-070T、QEC-M-090T、QEC-M-150T、および IDE 内のその他の QEC ボードも選択できます（4.2 節）

***安全：**端子の位置、シンク／ソース配線、電圧および電流の定格は、必ず QEC-M-02 ハードウェアマニュアルに従ってください。

1.4 LDmicro 1.1 との主な違い

以前の 86Duino IDE には LDmicro 1.1（ユーザーマニュアル LDmicro UM v1.6.3）が付属していました。86LDmicro 2.0 はラダー図の操作と 1.x の命令を引き継いでいますが、プロジェクト構成、設定方法、デバッグツールは異なります。

項目	LDmicro 1.1	86LDmicro 2.0（本マニュアル）
起動	86Duino IDE の「Tools」メニュー	IDE の「Tools > 86LDmicro」または IDE ツールバーの 86LDmicro ボタン；単独で実行することもできます（2.5 節）
プログラム構成	1 つのラダー図（＋ST ブロック）	複数の POU（PROGRAM、FUNCTION_BLOCK、FUNCTION）をタスク（Task）でスケジュールして実行
言語	LD、一部の ST	LD、ST、SFC、C/C++ ユーザーコード
変数	VAR LIST；名前の接頭辞（X、Y、R、T、C…）で型が決まる	グローバル変数リスト（GVL）＋POU のローカル変数、IEC の型（BOOL、INT、REAL、TIME、STRING、構造体、配列…）
スキャン周期	Settings → General の PLC Cycle Time	タスク（Task）の周期；複数のタスクを持てます（4.3 節）
設定	Settings メニュー：General／Modbus／Virtual Arduino	プロジェクト設定、デバイス、I/O マッピングページ（第 4 章）
I/O の割り当て	IO List の Pin Name	I/O マッピングページ：ローカル、EtherCAT、86EVA、Modbus、ピン；Excel へのエクスポート／インポートが可能（4.6 節）
コード生成	Code Generation ボタン：`ladder.h`、`myplc.cpp`、`myplc.h` を生成し、IDE に戻ってコンパイル・アップロード	ビルド（Ctrl+B）：`plc86.cpp`、`plc86.h` など；コンパイル（F7）、アップロード（F8）は 86LDmicro 内で完結（第 10 章）
.ino での呼び出し	`setupPlc()`／`loopPlc()`	`plc86_setup()`／`plc86_loop()`（インポート時に自動置換）
EtherCAT	86EVA 仮想デバイス（86EVA Configurator）	86EVA、またはプロジェクト独自のメインデバイス（ENI）＋組み込み EtherCAT Configurator＋SubDevice ページ＋CoE（第 11 章）
モーション制御	VirtualServo／パルス	Motion86 マシン（CiA 402 CSP、最大 9 軸）、PLCopen MC ファンクションブロック、試運転ページ（第 12 章）
シミュレーション	Simulate メニュー：Simulation Mode（F7）、Start Real-Time Simulation（F8）、Single Cycle（Space）	プロジェクト全体のシミュレーション（F5）（I/O、タイマー、軸、86HMI シミュレーションウィンドウを含む）、速度 x1／x10（第 9 章）
デバッグ	オンラインモニター	オンラインモニター、書き込み／強制、パワーフロー、Trace、Logbook、Cross reference、プロジェクト内検索

項目	LDmicro 1.1	86LDmicro 2.0（本マニュアル）
プロジェクトファイル	`data\myLD.ld`（単一ファイル）	`data\86ld\` 内のテキストファイル（git でバージョン管理可能、4.1 節）
その他	—	プロジェクト保護、AI アシスタントと MCP、コマンドラインツール、PLCopen XML のインポート（第 15 章）

LDmicro 1.1 のプロジェクトはそのままインポートできます。sketch に `data\myLD.ld` がある場合、IDE から 86LDmicro を開くとインポートするかどうか確認されます（2.5 節）。「File > Import a LDmicro 1.x program or a PLCopen XML」（15.1 節）を使うこともできます。元の `myLD.ld` は変更されません。

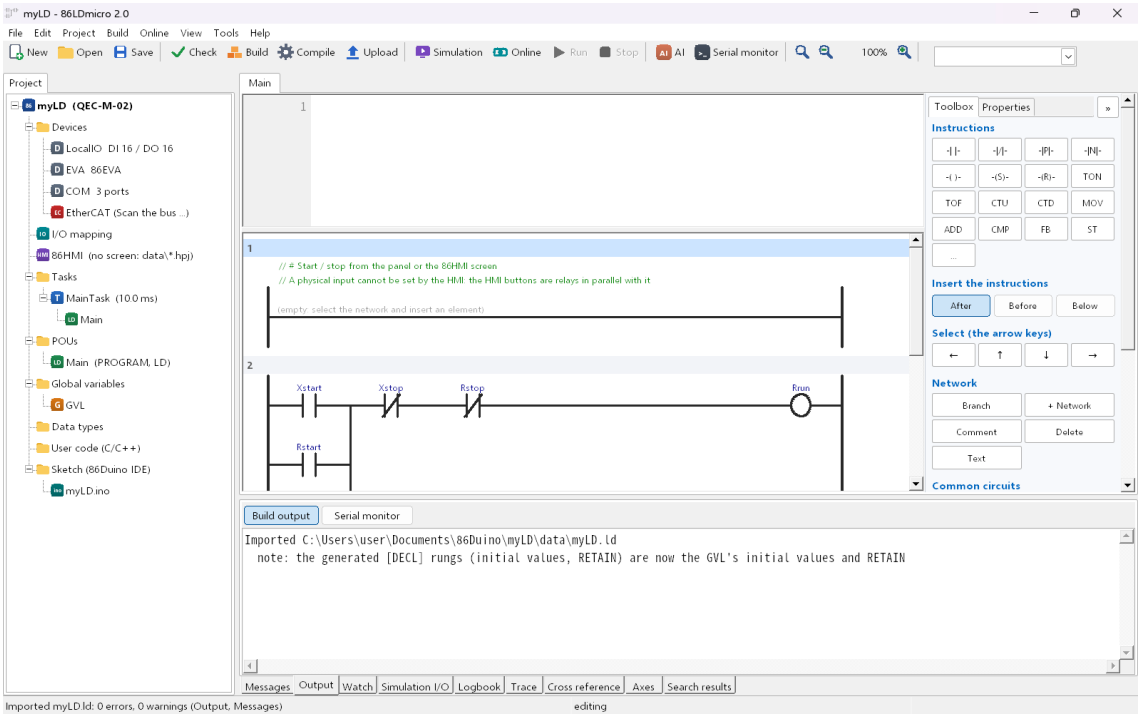


図 1-5　インポート後の 1.1 プロジェクト：ラダー図は POU になり、変数は GVL に入る

> ***ヒント：**1.x のタイマー、カウンターなどの命令は 1.x の意味（周期数でカウント）を保持します。新しいプログラムでは IEC ファンクションブロックを使用することをお勧めします。例：`CALL(t1, IN := @, PT := T#1s, Q => @)`（7.8 節）。
>
> LDmicro 1.1 に戻す方法は、2.3 節の最後の説明を参照してください。

第 2 章　環境のインストール

本章では、プログラムを書き始める前の準備として、86Duino IDE と 86LDmicro のインストール、コントローラーの接続と COM ポートの確認、IDE からの 86LDmicro の起動、および「Options」のコンパイル環境と言語の設定について説明します。

2.1　システム要件

項目	要件
オペレーティングシステム	Windows 10／11（64 ビット）
86Duino IDE	86Duino Coding 502 Preview（20261105）、DMP-ECAT EtherCAT ライブラリと新しい Motion86（9 軸）を含む
86LDmicro	86LDmicro 2.0.0 リリース版（zip）、IDE の `tools\LDmicro\tool` に展開（2.3 節）
コントローラー	QEC-M-02（その他の QEC ボードはプロジェクト設定で選択、4.2 節）
接続	データ通信可能な USB ケーブル（アップロード、オンラインモニター、EtherCAT Configurator で共用）；Configurator をネットワーク経由で使う場合は LAN ケーブルも必要
オプション	Microsoft Edge（組み込み EtherCAT Configurator 画面用）、Python 3（AI の MCP 機能用、15.4 節）

*__注意：__86Duino IDE は 502 Preview **20261105** 版を使用してください。それより古い IDE には CoE オブジェクトディクショナリで使う EtherCAT 関数がないか、旧版の Motion86 しかないため、EtherCAT やモーション制御のプロジェクトをビルドするとコンパイルに失敗することがあります。新しい Motion86 だけを古い IDE にコピーしてもコンパイルに失敗します。

2.2 86Duino Coding 502 のインストール

86LDmicro は 86Duino IDE のコンパイラー（arduino-builder）、ライブラリ、アップロードツール（v86dude）を使用するため、先に IDE をインストールします。

1. QEC 公式サイトから 86Duino Coding 502 Preview（20261105）を入手します。

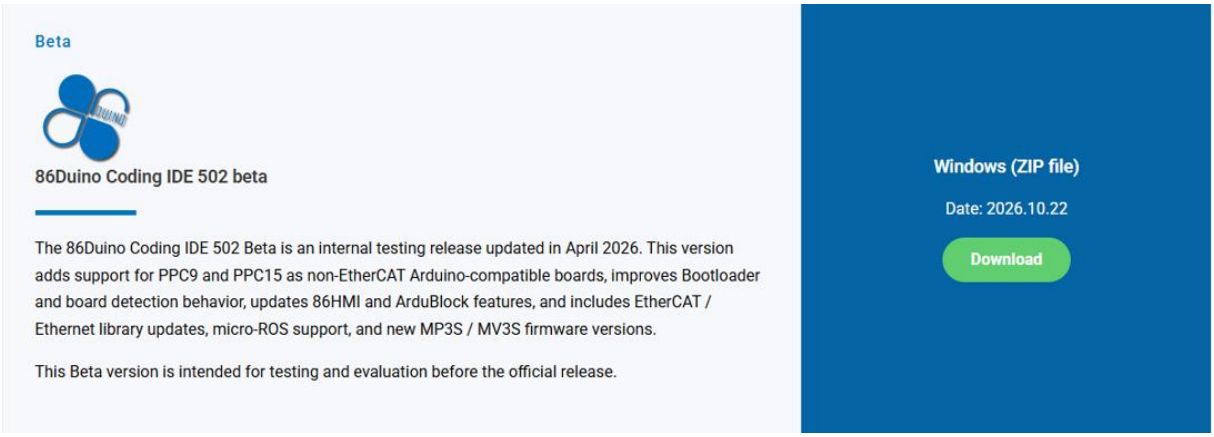


図 2-1　QEC サイトの 86Duino IDE 502 ダウンロードリンク

2. ファイルをローカルのフォルダーに展開します。例：`C:\86Duino_Coding_502_Preview_20261105`。展開後のフォルダーには `86duino.exe`、`hardware`、`tools` などがあります。本マニュアルではこのフォルダーを `<IDE フォルダー>` と呼びます。

3. 86duino.exe をダブルクリックして IDE を開きます。図 2-2　86Duino IDE フォルダー：86duino.exe、arduino-builder.exe、drivers、hardware、tools

名稱	修改日期	類型	大小
drivers	2026/11/16 下午 02:00	檔案資料夾	
examples	2026/11/16 下午 02:00	檔案資料夾	
hardware	2026/11/16 下午 02:01	檔案資料夾	
hermes	2026/11/16 下午 02:06	檔案資料夾	
java	2026/11/16 下午 02:06	檔案資料夾	
lib	2026/11/16 下午 02:07	檔案資料夾	
libraries	2026/11/16 下午 02:07	檔案資料夾	
tools	2026/11/16 下午 02:07	檔案資料夾	
tools-builder	2026/11/16 下午 02:07	檔案資料夾	
86duino.exe	2026/11/16 下午 02:00	應用程式	69 KB
86duino_debug.exe	2026/11/16 下午 02:00	應用程式	67 KB
86duino-cli.exe	2026/11/16 下午 02:00	應用程式	36,715 KB
arduino.l4j.ini	2026/11/16 下午 02:00	組態設定	1 KB
arduino_debug.l4j.ini	2026/11/16 下午 02:00	組態設定	1 KB
arduino-builder.exe	2026/11/16 下午 02:00	應用程式	23,401 KB
libusb0.dll	2026/11/16 下午 02:07	應用程式擴充	43 KB
msvcp100.dll	2026/11/16 下午 02:07	應用程式擴充	412 KB
msvcr100.dll	2026/11/16 下午 02:07	應用程式擴充	753 KB
README.md	2026/11/16 下午 02:07	MD 檔案	7 KB
revisions.txt	2026/11/16 下午 02:07	文字文件	97 KB

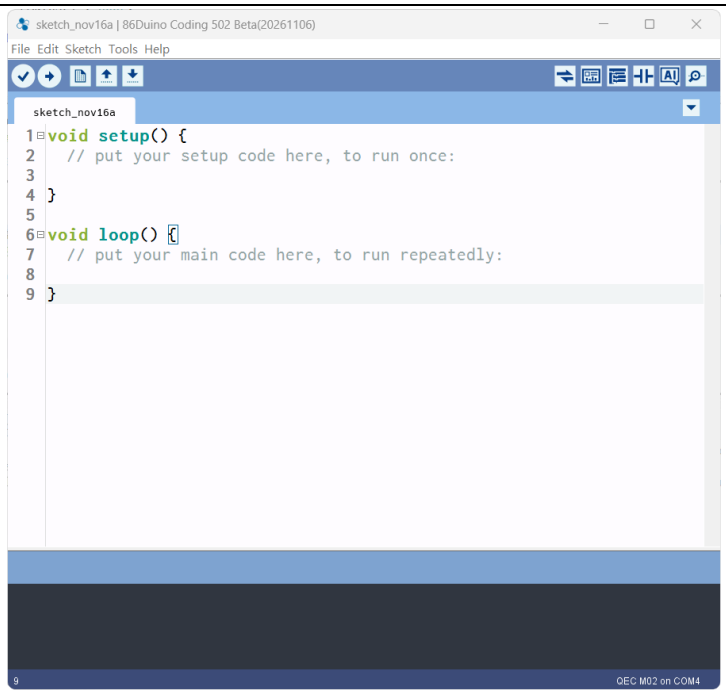


図 2-3　86Duino Coding 502 の起動後の画面（新しい空の sketch）

86LDmicro のプロジェクトは sketch フォルダー（<sketch>\data\86ld）に保存されるため、sketch を先に保存しておく必要があります。「File > Open...」で既存の sketch を開くか、「File > New」の後に「Save As...」で名前を付けて保存してください。IDE に付属のサンプル（examples）は、自分の sketch として保存し直してからでないと 86LDmicro を開けません。

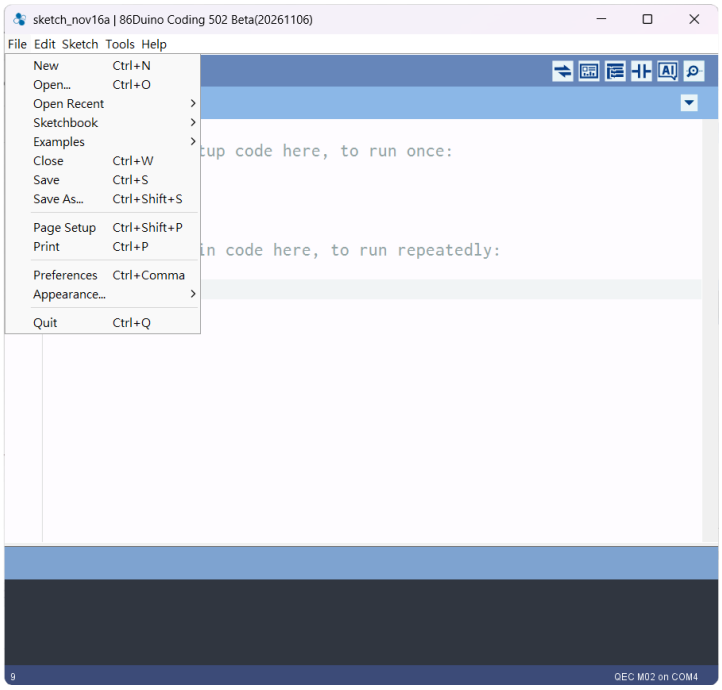


図 2-4　86Duino IDE の「File」メニュー：New、Open、Save As

2.3 86LDmicro ツールのインストール

86LDmicro 2.0.0 は 1 つの zip ファイル（例：86LDmicro_2.0.0_v1r2.zip）としてリリースされます。zip の最上位は tool フォルダーで、その中身を <IDE フォルダー>\tools\LDmicro\tool に配置します。お使いの 86Duino IDE に 86LDmicro 2.0.0 がすでに付属している場合（<IDE フォルダー>\tools\LDmicro\tool\VERSION.txt の内容が 2.0.0 2026-11-11）は、本節を省略できます。

```
<IDE フォルダー>\tools\LDmicro\tool\
    LDmicro.jar          IDE の「Tools > 86LDmicro」と IDE ツールバーのボタン
    VERSION.txt          バージョンと日付：2.0.0 2026-11-11（問題を報告するときに添付してください）
    bin\                 86LDmicro.exe（本体）、86ldmicro-cli.exe（コマンドライン、15.5 節）
    configurator\        EtherCAT Configurator（第 11 章）
    docs\                オフラインドキュメント：USER_GUIDE.md、USER_CODE_API.md、ETHERCAT_MOTION.md、
ai\、LICENSE.txt
    hmisim\              86HMI シミュレーションウィンドウ（13.5 節）
    mcp\                 AI アシスタント用の MCP server（15.4 節）
```

2.3.1 インストール手順

1. 86Duino IDE を閉じます。
2. （推奨）<IDE フォルダー>\tools\LDmicro フォルダー全体をコピーしてバックアップしておきます。後で LDmicro 1.1 に戻すときに使います。
3. ダウンロードした zip ファイルを右クリック > 「プロパティ」を選び、「全般」タブで「許可する」にチェックを入れて（この項目がある場合）「OK」をクリックします。これで、展開した実行ファイルが Windows にブロックされなくなります。
4. zip を <IDE フォルダー>\tools\LDmicro に展開します。zip 内の tool フォルダーが <IDE フォルダー>\tools\LDmicro\tool になります。同じ名前のファイルがある場合は「ファイルを置き換える」を選択します。フォルダーの階層を 1 つ余分に作らないでください（誤った例：tools\LDmicro\86LDmicro_2.0.0_v1r2\tool）。
5. <IDE フォルダー>\tools\LDmicro\tool に上記の LDmicro.jar、VERSION.txt、bin、configurator、docs、hmisim、mcp があり、VERSION.txt の内容が 2.0.0 2026-11-11 であることを確認します。
6. 86Duino IDE を再度開きます。「Tools」メニューに 86LDmicro が表示されれば、インストールは完了です（2.5 節）。

2.3.2 LDmicro 1.1 に戻す

86Duino IDE を閉じ、<IDE フォルダー>\tools\LDmicro をインストール前にバックアップしたフォルダーに戻してから、IDE を再度開きます。86LDmicro 2.0 のプロジェクトは sketch の data\86ld に保存され、インポート時にも 1.1 の data\myLD.ld は変更されないため、同じ sketch に 2 種類のプロジェクトを共存させることができます。

2.4　ハードウェア接続と COM ポート

PC と QEC-M-02 は USB ケーブルで接続します。アップロード、オンラインモニター、シリアルモニターはすべて、このケーブルの PC 上の COM ポートを使用します。

接続	接続元	接続先	用途
USB	PC	QEC-M-02 の USB ポート	プログラムのアップロード、オンラインモニター、シリアルモニター、Configurator（USB モード）
EtherCAT	QEC-M-02 の EtherCAT ポート	最初のサブデバイスの IN、以降は OUT → 次の IN	I/O モジュール（例：QEC-R11CFFG）、CiA 402 ドライブ（例：QEC-R11MP3S）
ローカル I/O	QEC-M-02 の端子台	ボタン、センサー、表示灯、リレー	`LocalIO` の DI／DO（例：X00、Y00）
LAN（オプション）	PC	QEC-M-02 の LAN ポート	Configurator ネットワークモード（デフォルト 192.168.100.1）、Modbus TCP
RS-232／485（オプション）	外部機器	COM1～COM3	Modbus RTU、ユーザーコードの Serial1～3

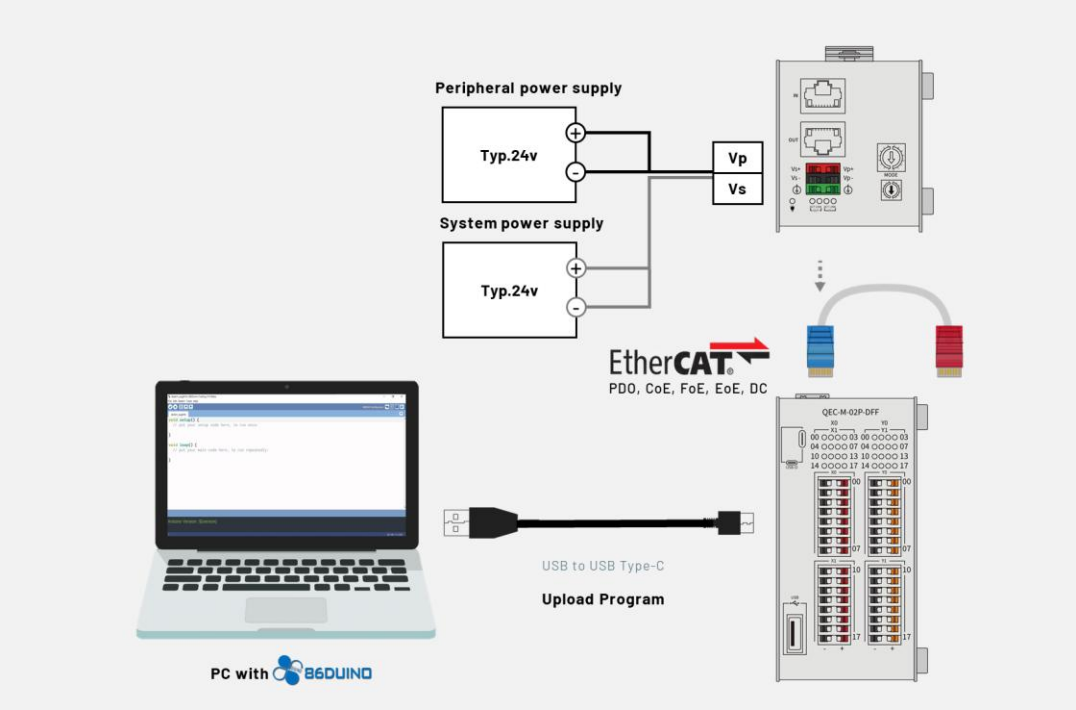


図 2-5　QEC と PC のハードウェア接続（概略図）

*安全：必ず電源を切ってから配線してください。誘導性負荷（リレー、電磁弁）にはサージ吸収素子を付けてください。

非常停止は、ハードウェアでドライブの電源または STO を遮断する必要があります。PLC プログラムだけに頼らないでください。

2.4.1 COM ポートの確認

1. QEC-M-02 に電源を入れ、USB ケーブルで PC に接続します。
2. Windows の「スタート」を右クリック > **デバイス マネージャー** を選びます。
3. 「**ポート (COM と LPT)**」を展開し、新しく表示された「USB シリアル デバイス (COMx)」を見つけて COM 番号を控えます（下図では COM4）。USB ケーブルを抜き差ししたときに消えたり現れたりするものがコントローラーです。

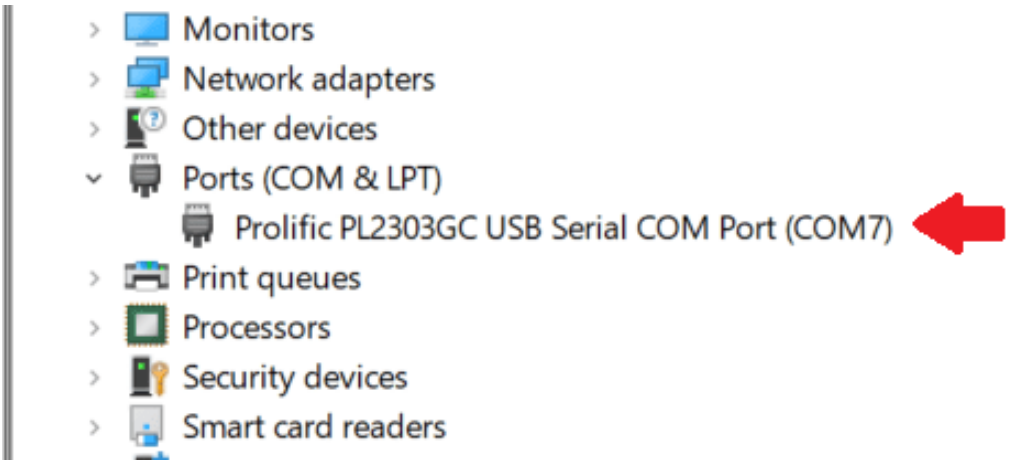


図 2-6　デバイス マネージャー：「ポート (COM と LPT)」の下の USB シリアル デバイス (COM4)（概略図）

*ヒント：新しい COM ポートが見つからない場合は、USB ケーブルを交換する（データ通信可能なもの）、別の USB ポートに挿す、またはコントローラーに電源が入っていることを確認してください。
黄色の感嘆符が表示される場合は、86Duino IDE に付属のドライバー（`<IDE フォルダー>\drivers`）をインストールしてから、もう一度抜き差ししてください。

2.4.2 IDE でのボードとシリアルポートの選択

1. 86Duino IDE で「**Tools > Board**」をクリックし、**QEC M02** を選択します。

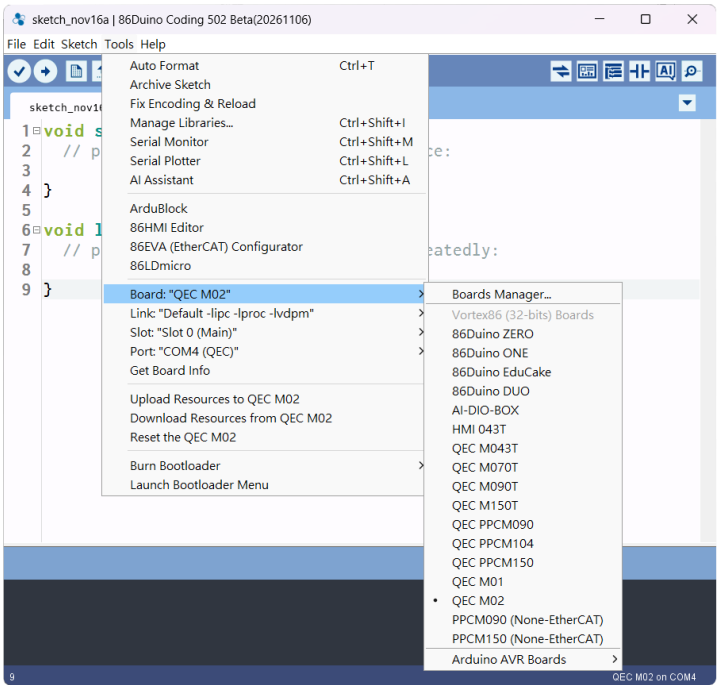


図 2-7　「Tools > Board」で QEC M02 を選択；同じメニューに「86LDmicro」がある

2. 「**Tools > Port**」をクリックし、上で確認した COM ポート（例：COM4）を選択します。IDE の右下に「QEC M02 on COM4」と表示されます。

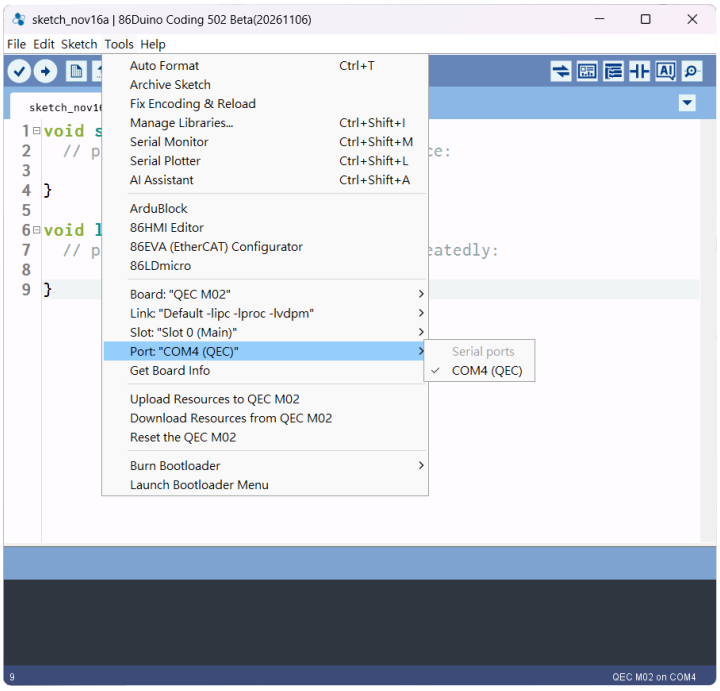


図 2-8　「Tools > Port」で COM4 を選択（概略図）

IDE から 86LDmicro を起動すると、IDE で選択したボードとシリアルポートがそのまま使われます（2.5 節）。86LDmicro では、ツールバー右側のシリアルポート欄から変更することもできます（3.3 節）。

#	初回接続のチェック	確認方法
1	コントローラーに電源が入っている	電源ランプが点灯している
2	PC が COM ポートを認識している	デバイス マネージャーに USB シリアル デバイス (COMx) が表示される
3	IDE のボードとポートが正しい	IDE ウィンドウの右下に「QEC M02 on COMx」と表示される
4	他のプログラムが COM ポートを使用していない	IDE のシリアルモニターウィンドウやその他のターミナルプログラムを閉じる
5	86LDmicro でアップロード・接続できる	第 5 章クイックスタートのプログラムのアップロードに成功し、Ctrl+F5 の後にステータスバーに「ONLINE COMx: RUN」と表示される

2.5 86LDmicro の起動

86Duino IDE から起動することをお勧めします。86LDmicro はその sketch のプロジェクトを開き（または作成し）、その IDE のコンパイラーと IDE で選択したシリアルポートを自動的に使用します。

2.5.1 86Duino IDE から起動する（推奨）

1. IDE で sketch を開きます（または新規作成して保存します）。
2. 「Tools > Board」が QEC ボード、「Tools > Port」が正しい COM ポートであることを確認します（2.4 節）。
3. 「**Tools > 86LDmicro**」をクリックするか、IDE ツールバー右側の 86LDmicro ボタン（マウスを重ねると「86LDmicro」と表示）をクリックします。

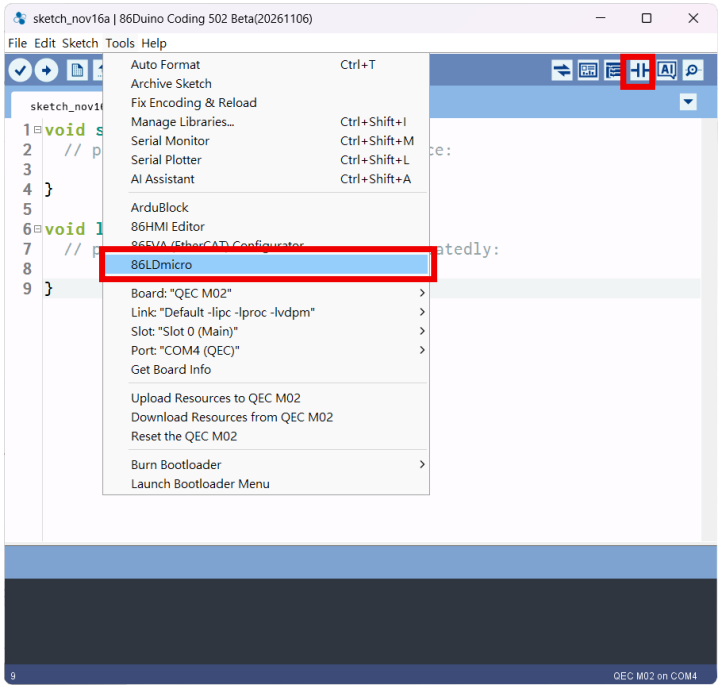
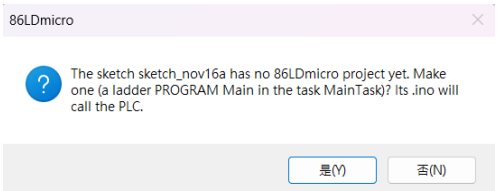
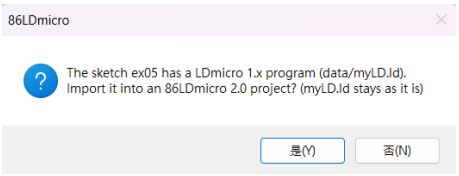
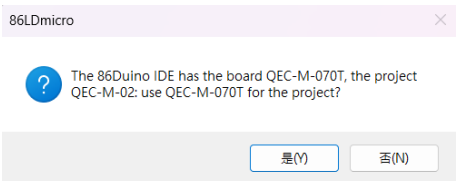


図 2-9　IDE ツールバーと右側の 86LDmicro ボタン

86LDmicro は sketch の状態に応じて自動的に処理します。

sketch の状態	86LDmicro の動作	86LDmicro ウィンドウ
プロジェクトがない	「The sketch … has no 86LDmicro project yet. Create one?」と確認されます。**Yes** をクリックすると新しいプロジェクトが作成されます（ラダー図 PROGRAM `Main`、タスク `MainTask` で実行）	 図 2-10　sketch にまだプロジェクトがないときの確認

sketch の状態	86LDmicro の動作	86LDmicro ウィンドウ
LDmicro 1.1 の `data\myLD.ld` がある	86LDmicro 2.0 プロジェクトとしてインポートするかどうか確認されます。**Yes** をクリックするとインポートされます（`myLD.ld` は変更されません、15.1 節）	 図 2-11　sketch に LDmicro 1.1 のプログラム（data/myLD.ld）があるときの確認
2.0 プロジェクトがすでにある（`data\86ld\project.86ld`）	そのまま開きます。IDE で選択したボードとプロジェクトのボードが異なる場合は、プロジェクトで IDE のボードを使うかどうか確認されます	 図 2-12　IDE で選択したボードとプロジェクトが異なるときの確認
同じ sketch で 86LDmicro がすでに開いている	そのウィンドウを最前面に表示	

新しいプロジェクトを作成した後の 86LDmicro メインウィンドウは次のとおりです。プロジェクトツリーにはすでにタスク `MainTask`（10 ms）、ラダー図 `Main`、グローバル変数リスト `GVL` があり、`Main` が自動的に開きます。各領域の説明は第 3 章を参照してください。

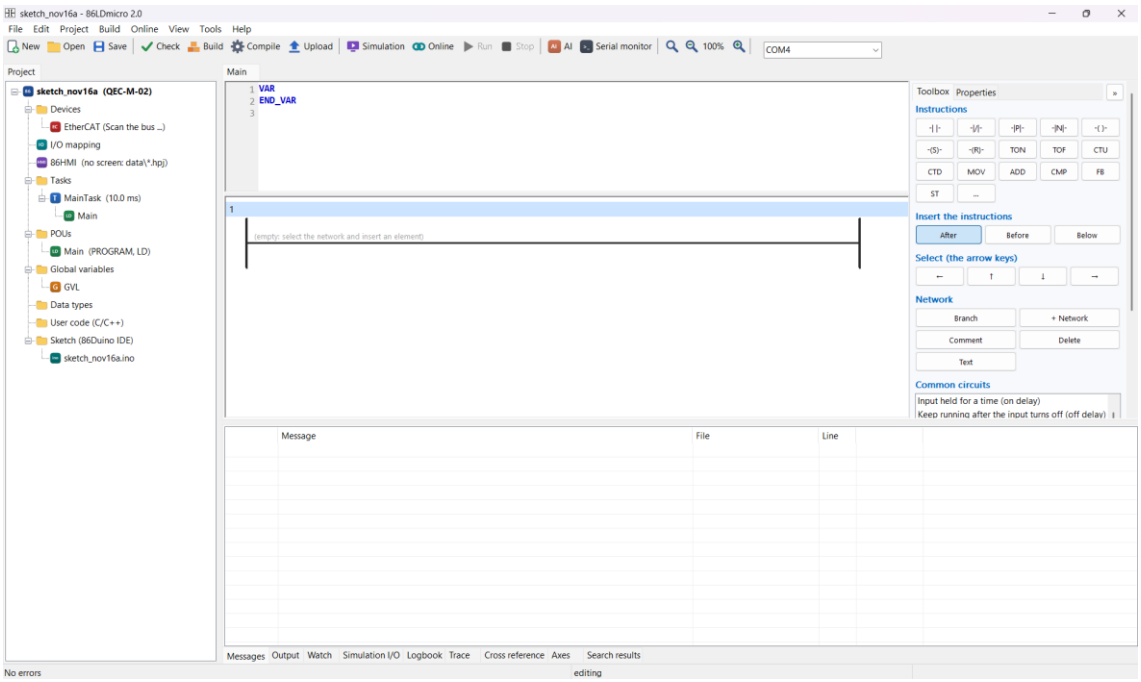


図 2-13　IDE から新しいプロジェクトを作成した後の 86LDmicro：MainTask、Main、GVL が作成済み

- `.ino` には `#include "plc86.h"`、`plc86_setup()`、`plc86_loop()` が追加され（1.1 の `setupPlc()`／`loopPlc()` は置き換えられます）、それ以外のコードは変更されません。

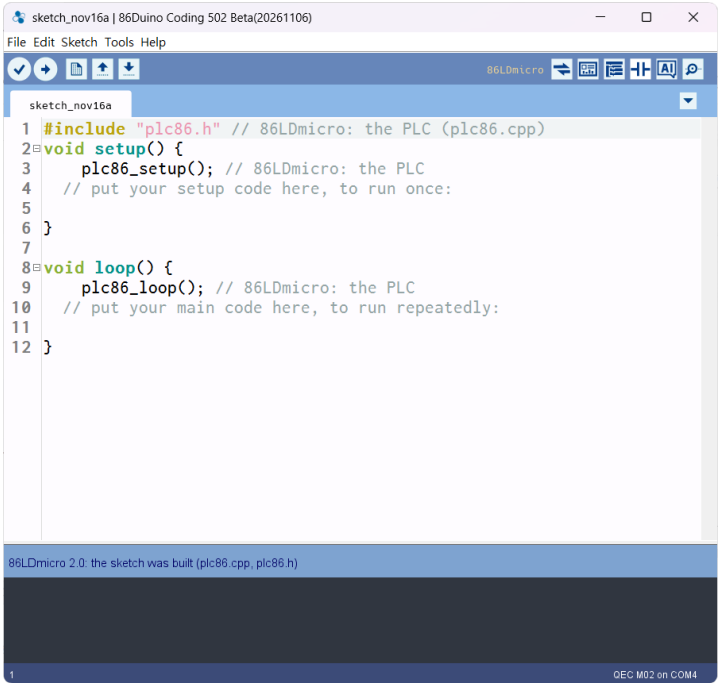


図 2-14　.ino に 86LDmicro のコードを追加

- ビルド後、86LDmicro は IDE に sketch の再読み込みを指示し、IDE に `plc86.cpp` などのタブが表示されます。以降は IDE で直接コンパイル・アップロードすることもできます。これらの自動生成ファイルは手動で変更しないでください（ビルドのたびに書き直されます）。

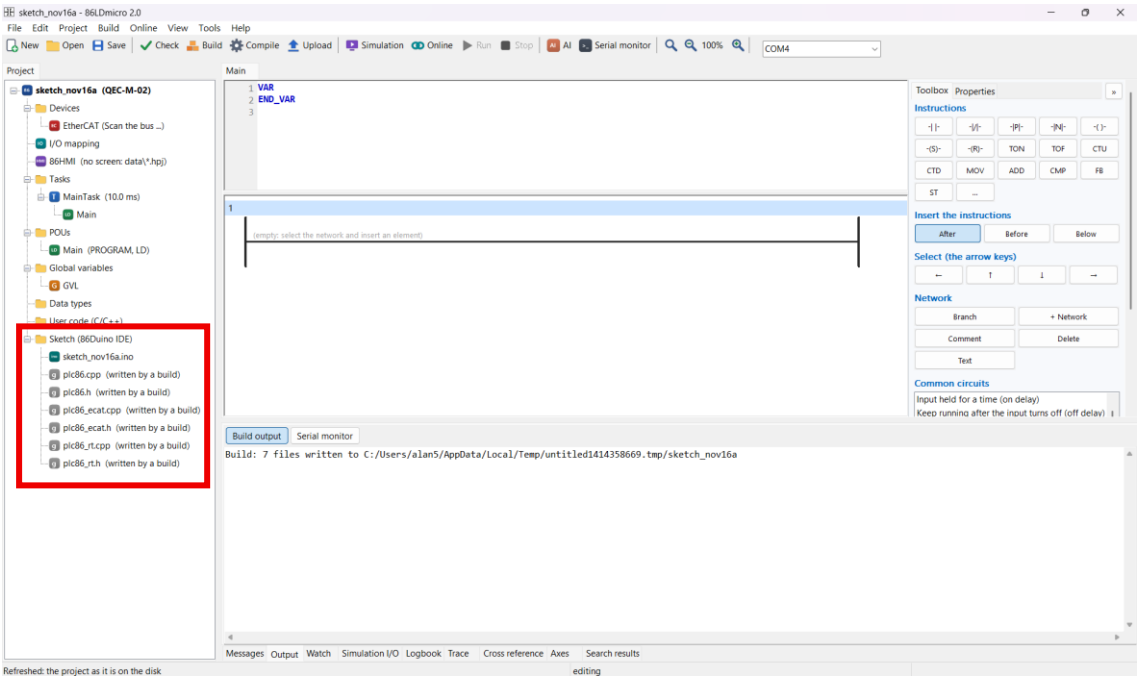


図 2-15　86LDmicro のビルドで生成された plc86 などのファイル

ビルド後、86LDmicro の Sketch (86Duino IDE) に `plc86.cpp` などのタブがすぐに表示されない場合は、View ツールバーの Refresh (Ctrl + R) をクリックします。

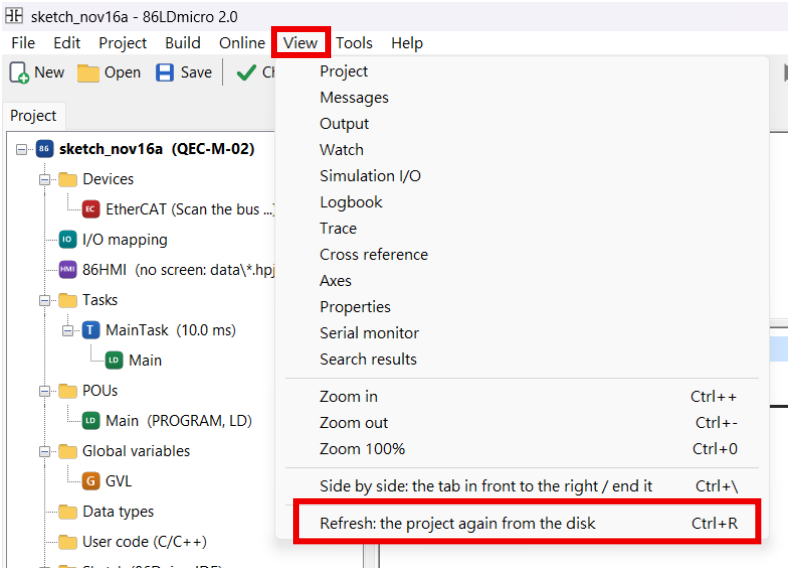


図 2-16　Refresh（Ctrl+R）をクリックすると、Sketch に plc86.cpp などのファイルが表示される

- ボードが QEC ボードでない場合、IDE 下部のステータスバーに「86LDmicro 2.0: choose a QEC board (Tools > Board)」と表示され、86LDmicro は起動しません。sketch が IDE のサンプルフォルダーにある場合は「save the example as a sketch of your own first (File > Save As)」と表示されます。



図 2-17　ビルド後の 86Duino IDE：plc86.cpp、plc86.h、plc86_ecat.cpp などのタブ

***注意：**IDE で未保存の `.ino` の変更は、86LDmicro のビルド後の再読み込みで上書きされます。先に IDE で保存するか、86LDmicro のプロジェクトツリーの「Sketch (86Duino IDE)」で `.ino` を編集してください（3.4 節）。

2.5.2 単独で起動する

次のファイルを直接実行することもできます：`<IDE フォルダー> \tools\LDmicro\tool\bin\86LDmicro.exe`。

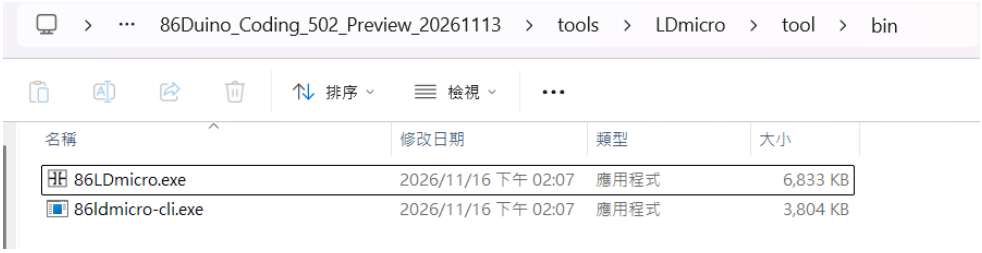


図 2-18　単独で起動した、プロジェクト未オープンの 86LDmicro のパス

単独で起動すると、前回開いていたプロジェクトが自動的に開きます。ない場合、プロジェクトツリーに「No project: File > New, Open or Import」と表示されます。

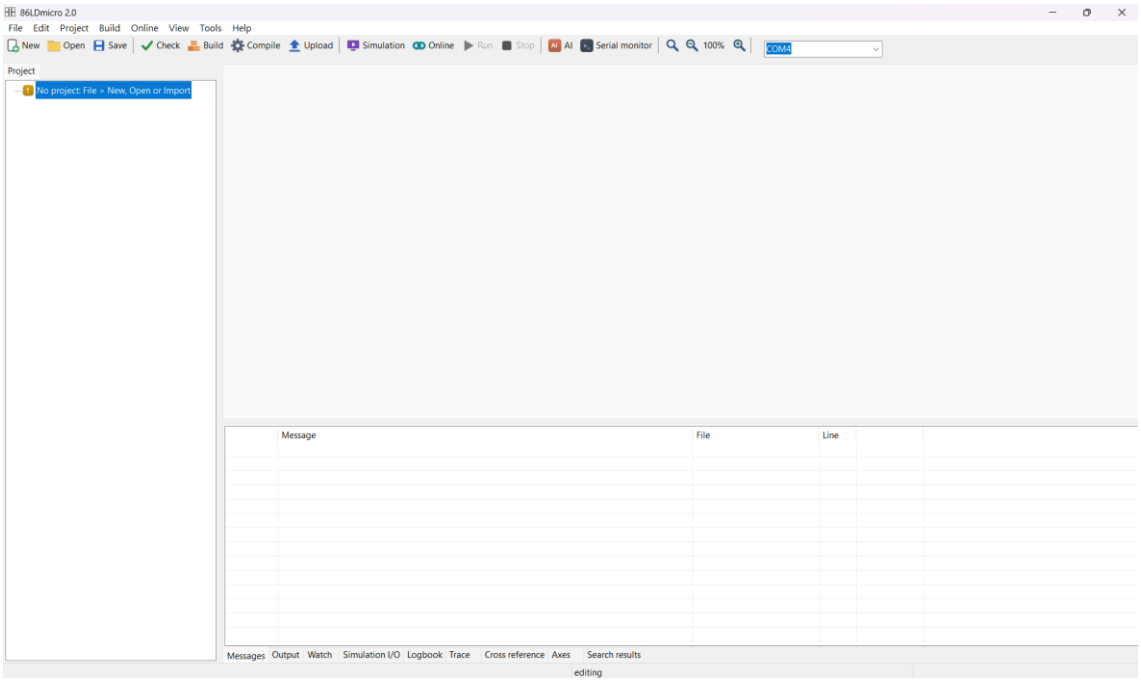
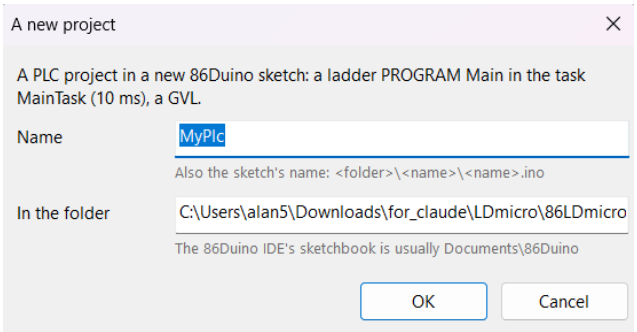


図 2-19　単独で起動した、プロジェクト未オープンの 86LDmicro

やりたいこと	操作
新しいプロジェクトを作成	「File > New project...」（Ctrl+N）：名前（sketch の名前にもなります）と場所（デフォルトは ドキュメント`\86Duino`、つまり IDE の sketchbook）を入力し、**OK** をクリックします。`<場所>\<名前>\<名前>.ino` とプロジェクトが作成され、ラダー図 `Main`、タスク `MainTask`（10 ms）、`GVL` が含まれます  図 2-20　「File > New project...」：名前は sketch の名前にもなる

やりたいこと	操作
プロジェクトを開く	「File > Open project...」（Ctrl+O）：`data\86ld\project.86ld` または sketch の `.ino` を選択
インポート	「File > Import a LDmicro 1.x program or a PLCopen XML (.ld, .xml)...」、または `.ld`／`.xml` ファイルを 86LDmicro ウィンドウにドラッグ＆ドロップ（第 15 章）
最近のプロジェクト	「File > Recent projects」に最近開いた 8 つのプロジェクトが表示されます

*__重要：__単独で起動して初めて使うときは、まず「Tools > Options...」で 86Duino IDE フォルダーを指定してください（2.6 節）。指定しないと、チェック、ビルド、シミュレーションしかできず、コンパイルとアップロードはできません。

2.6 オプション

「**Tools > Options...**」では、この PC 上の 86LDmicro のコンパイル環境、インターフェース言語、アップロード後の接続方法を設定します（プロジェクト自体の設定は「Project settings」、4.2 節）。プロジェクト設定ページの **Options...** ボタンでも同じウィンドウが開きます。

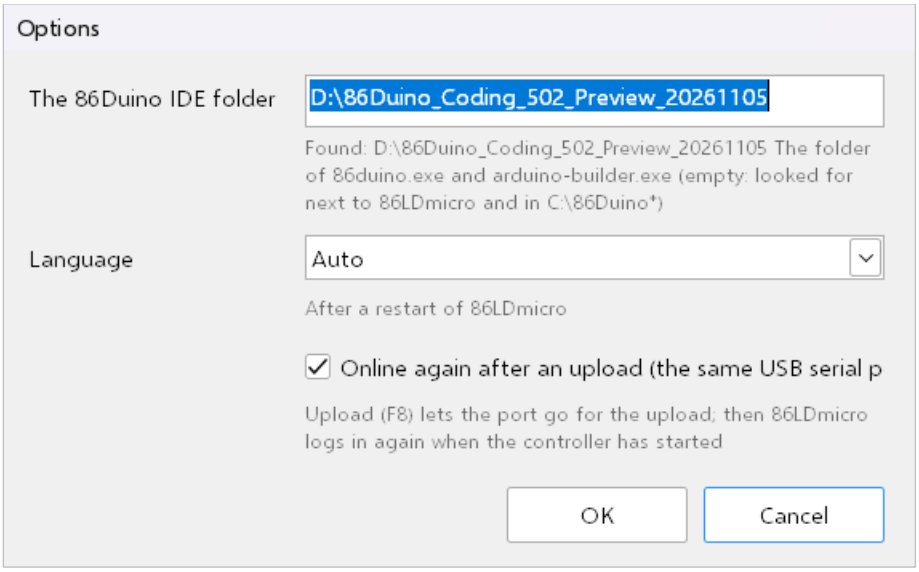


図 2-21　Options：86Duino IDE フォルダー、言語、アップロード後の自動再接続

フィールド	説明
The 86Duino IDE folder	コンパイル環境：`86duino.exe` と `arduino-builder.exe` があるフォルダー。下に「Found: …」または「Not found yet.」と表示されます。空欄の場合は 86LDmicro と同じ場所と `C:\86Duino*` を検索します。IDE から起動すると、その IDE のフォルダーに自動的に変更されます。入力したフォルダーが 86Duino IDE でない場合、OK をクリックすると「There is no 86Duino IDE in this folder」と表示されます
Language	Auto（Windows に従う）、English、繁體中文、日本語；**86LDmicro の再起動後に有効**
Online again after an upload (the same USB serial port, as in LDmicro 1.1)	デフォルトでチェックされています。アップロード（F8）時にまずシリアルポートを解放し、コントローラーの起動後に 86LDmicro が自動的にオンラインモニターに再ログインします

2.6.1 コンパイル環境の選択

- **PC に複数の 86Duino IDE がある場合**：「Tools > 86LDmicro」をクリックした IDE のコンパイラーとライブラリが使用されます。単独で起動した場合は「Options」で指定したフォルダーが使用されます。
- **コントローラー（ボード）**：プロジェクト設定で選択します（4.2 節）。86LDmicro はそれに基づいて IDE 内の FQBN（例：`86duino:x86:86DuinoQECM02`）を見つけます。特別な場合は、プロジェクト設定の「FQBN (optional)」で直接指定できます。

- **ライブラリ**：その IDE の `hardware\86duino\x86\libraries` にあるライブラリ（Ethercat、Motion86、Modbus……）を使用します。EtherCAT と Motion86 のプロジェクトでは、502 Preview 20261105 に付属のライブラリを使用してください。
- IDE が見つからない場合、コンパイルまたはアップロード時に「The 86Duino IDE is not found: Tools > Options > the 86Duino IDE folder」と表示されます。

2.6.2　インターフェース言語

「Language」で選択し、86LDmicro を再起動すると切り替わります。

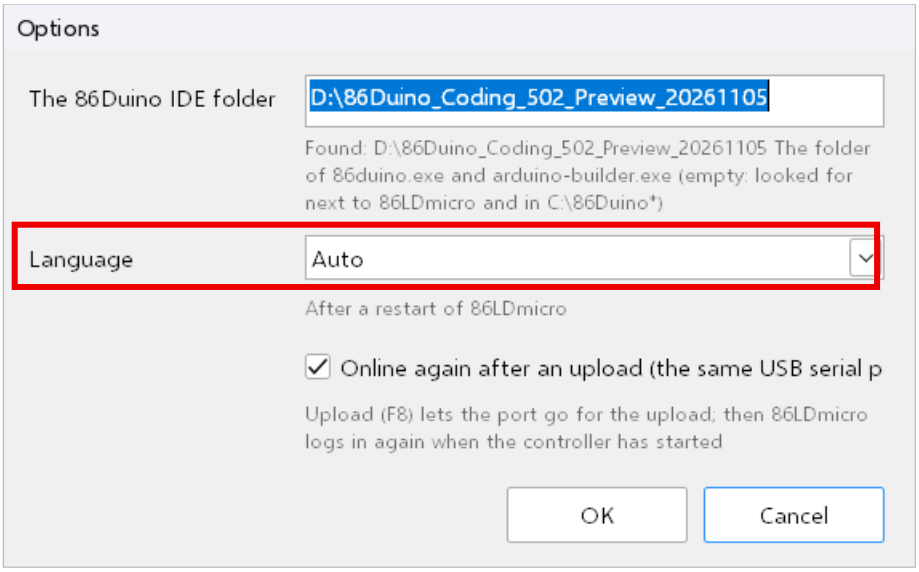


図 2-22　Options：86Duino IDE フォルダー、言語、アップロード後の自動再接続

メニューの位置は 3 つの言語ですべて同じです。本マニュアルのスクリーンショットは英語インターフェースを使用しています。

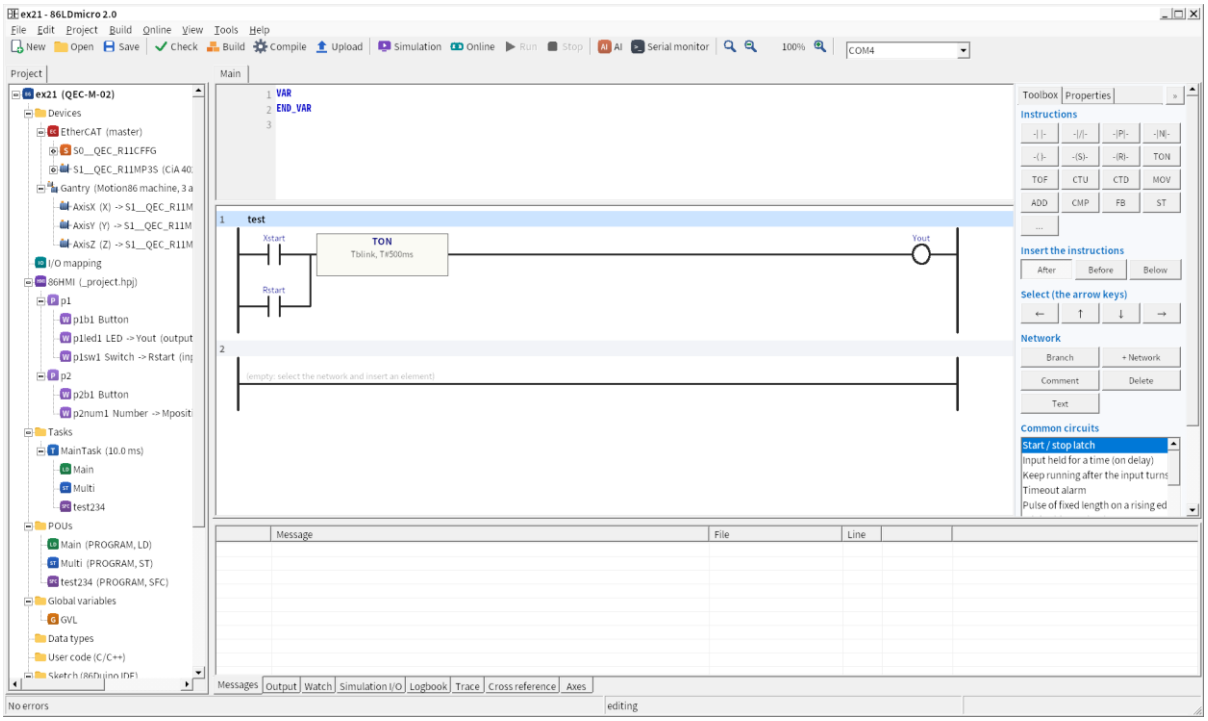


図 2-23　English を選択したときのインターフェース（メニューの位置は繁体字中国語と同じ）

第 3 章　インターフェースの概要

本章では、86LDmicro のメインウィンドウを順に紹介します。各メニューのすべての項目、ツールバー、プロジェクトツリー、POU エディターとサイドパネル、下部のツールウィンドウ、およびタブ、左右並べ表示、ズーム、検索です。各機能の詳しい操作は後の章で説明し、表中に節番号で示します。

3.1　メインウィンドウとステータスバー

メインウィンドウは上から順に、メニューバー、ツールバー（右側にシリアルポート欄）があり、中段の左がプロジェクトツリー、中央がドキュメント領域（POU や設定ページごとに 1 つのタブ）、右が POU のサイドパネルです。その下にツールウィンドウ、最下部にステータスバーがあります。

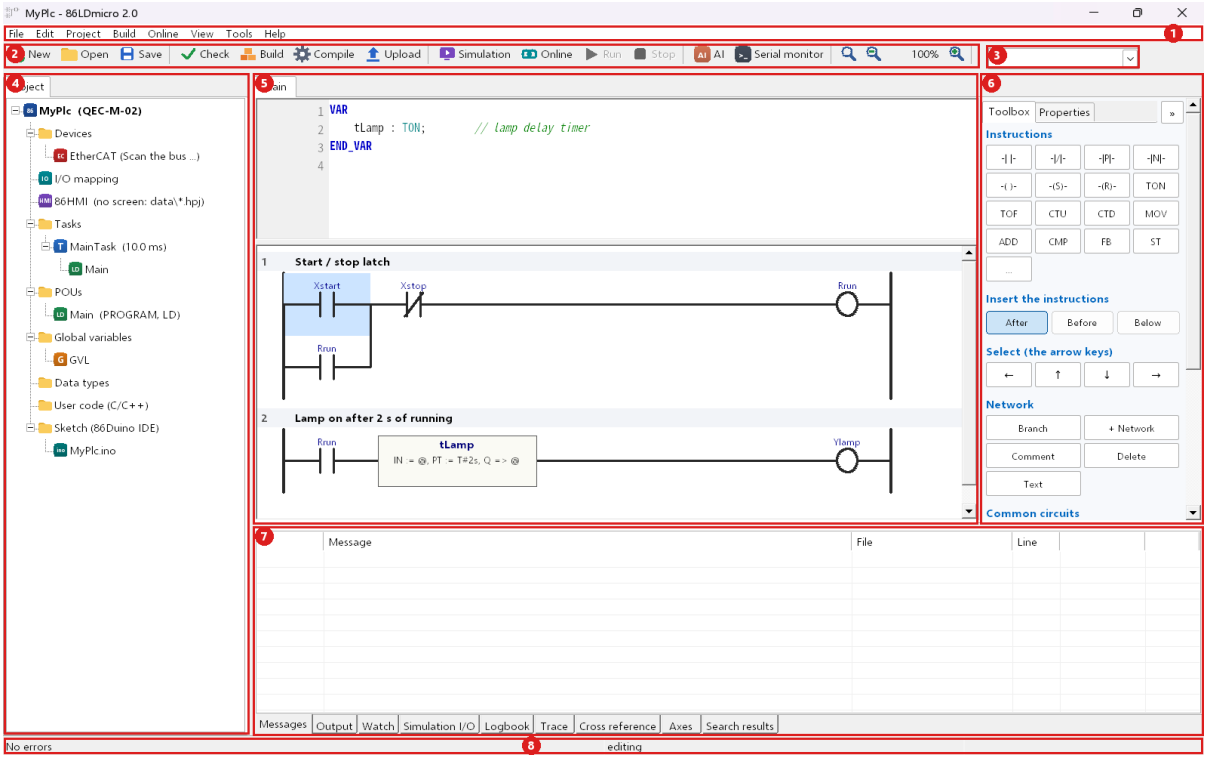


図 3-1　86LDmicro メインウィンドウの各領域

番号	領域	説明
1	メニューバー	File、Edit、Project、Build、Online、View、Tools、Help（3.2 節）
2	ツールバー	よく使うコマンド（3.3 節）；ウィンドウが狭くなると自動的にアイコンのみ、または 2 段表示になります
3	シリアルポート	アップロード、オンラインモニター、シリアルモニター、USB 版 Configurator が使用する COM ポート
4	プロジェクトツリー	プロジェクトのすべての内容；ダブルクリックで開き、右クリックでそのノード専用のメニューが表示されます（3.4 節）

番号	領域	説明
5	ドキュメント領域	POU、設定ページごとに 1 つのタブ；左右に並べて表示できます（3.7 節）
6	サイドパネル	LD、ST、SFC のタブでのみ表示：「Toolbox」と「Properties」（3.5 節）
7	ツールウィンドウ	Messages、Output、Watch、Simulation I/O、Logbook、Trace、Cross reference、Axes（試運転）、Search results（3.6 節）
8	ステータスバー	メッセージ、モード、スキャン情報（下表参照）

タイトルバーには「プロジェクト名 - 86LDmicro 2.0」と表示されます。プロジェクトに未保存の変更があると、名前の後に * が付きます。

ステータスバーは左から右へ 3 つの欄に分かれています。

位置	内容	例
左	直前の操作の結果、チェックで見つかったエラー数	No errors／No errors, 2 warnings／3 errors, 0 warnings／Saved
中央	現在のモード	Editing、Compiling ...、SIMULATION: RUN、SIMULATION: STOP、ONLINE COM4: RUN、ONLINE COM4: STOP
右	シミュレーション中またはオンライン時のスキャン情報	シミュレーション：「1520 scans, 15 s」；オンライン：「scan 230 us (max 410)」（最初のタスク）



図 3-2　シミュレーション中のステータスバー：No errors、SIMULATION: RUN、スキャン回数と時間

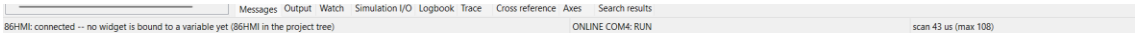


図 3-3　オンライン時のステータスバー：ONLINE COM4: RUN、スキャン時間

3.2　メインメニュー

メインメニューには File、Edit、Project、Build、Online、View、Tools、Help の 8 つのメニューがあります。メニュー名の後の括弧内の文字は Alt と組み合わせて使えます。例えば Alt+F で「File」が開きます。ショートカットキーはどのタブでも有効です。シミュレーション中やオンライン時に使えない項目はグレー表示になります。以下の各表では、メニューごとにすべての項目を上から順に示します（項目名の括弧内の文字は省略します）。

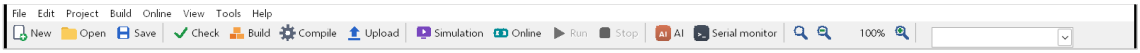


図 3-4　メニューバーとツールバー

3.2.1　File

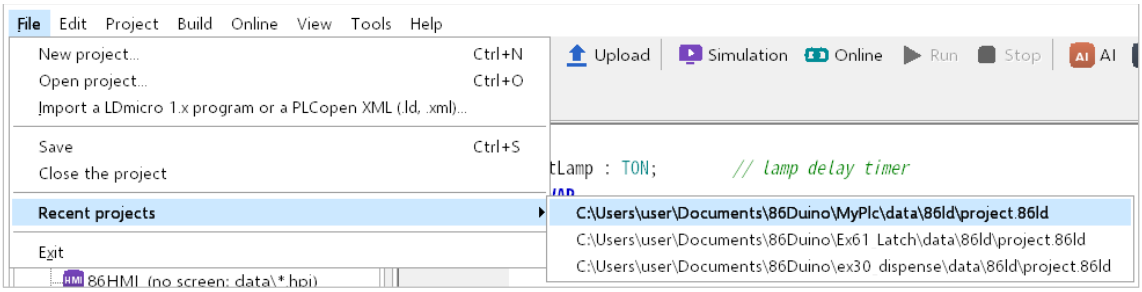


図 3-5　「File」メニュー

項目	ショートカット	用途
New project...	Ctrl+N	新しい 86Duino sketch にプロジェクトを作成：名前、場所（2.5 節）
Open project...	Ctrl+O	プロジェクトを開く：`project.86ld` または sketch の `.ino` を選択
Import a LDmicro 1.x program or a PLCopen XML (.ld, .xml)...	—	LDmicro 1.x の `.ld` または PLCopen XML を新しいプロジェクトとしてインポート（15.1、15.2 節）；ファイルをウィンドウにドラッグ＆ドロップすることもできます
Save	Ctrl+S	プロジェクトのすべての変更を保存（86LDmicro で編集した sketch ファイルを含む）；ステータスバーに「Saved」と表示
Close the project	—	プロジェクトを閉じる；未保存の変更がある場合は「Save the changes of the project?」と確認
Recent projects ▸	—	最近開いた 8 つのプロジェクト。クリックすると開きます。ない場合は「(none)」と表示
Exit	—	86LDmicro を終了（未保存の変更がある場合は先に確認）

3.2.2 Edit

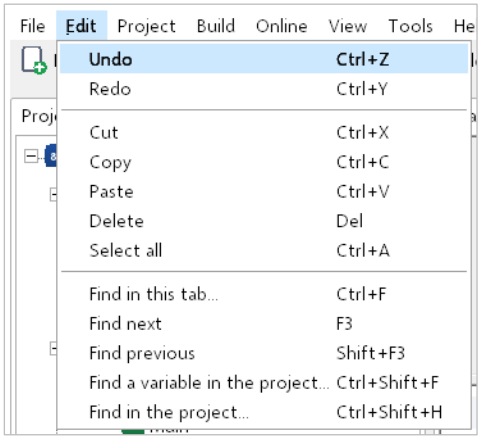


図 3-6 「Edit」メニュー

項目	ショートカット	用途
Undo	Ctrl+Z	現在のタブの直前の変更を元に戻す
Redo	Ctrl+Y	元に戻した変更をやり直す
Cut	Ctrl+X	コード：選択したテキスト；ラダー図：選択した要素、または要素を選択していない場合はネットワーク全体
Copy	Ctrl+C	同上、コピー
Paste	Ctrl+V	貼り付け；ラダー図では選択した位置に貼り付け
Delete	Del	選択したテキスト、要素、ネットワークを削除
Select All	Ctrl+A	現在のタブのすべてのテキストを選択
Find in this tab...	Ctrl+F	現在のタブの上部に検索バーを開く（3.7 節）
Find next	F3	次の一致箇所へ移動
Find previous	Shift+F3	前の一致箇所へ移動
Find a variable in the project...	Ctrl+Shift+F	変数名を入力すると、その変数が読み取り・書き込み・宣告されている位置が「Cross reference」に一覧表示されます（9.5 節）
Find in the project...	Ctrl+Shift+H	プロジェクト全体から任意のテキストを検索し、結果を「Search results」に一覧表示（3.7、9.5 節）

3.2.3 Project

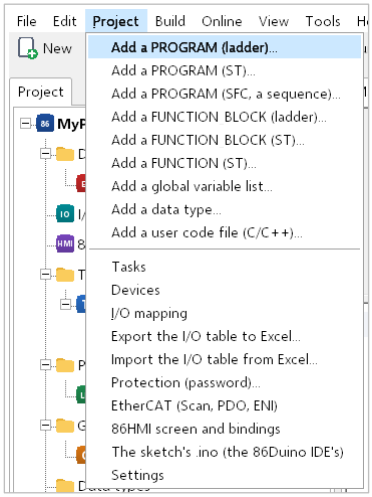


図 3-7 「Project」メニュー

項目	ショートカット	用途
Add a PROGRAM (ladder)...	—	ラダー図の PROGRAM を追加；最初のタスクに自動的に追加されます（4.7 節）
Add a PROGRAM (ST)...	—	ST の PROGRAM を追加（第 8 章）
Add a PROGRAM (SFC, a sequence)...	—	SFC の PROGRAM を追加（8.4 節）
Add a FUNCTION_BLOCK (ladder)...	—	ラダー図のファンクションブロックを追加
Add a FUNCTION_BLOCK (ST)...	—	ST のファンクションブロックを追加（8.3 節）
Add a FUNCTION (ST)...	—	ST のファンクションを追加（8.3 節）
Add a global variable list...	—	GVL を 1 つ追加（4.5 節）
Add a data type...	—	データ型ファイル（構造体、列挙型、配列）を 1 つ追加（4.5 節）
Add a user code file (C/C++)...	—	`.cpp86` ユーザーコードを追加（第 14 章）
Tasks	—	Tasks ページを開く（4.3 節）
Devices	—	Devices ページを開く（4.4 節）
I/O mapping	—	I/O mapping ページを開く（4.6 節）
Export the I/O table to Excel...	—	変数と I/O マッピングを .xlsx または .csv で保存（4.6 節）
Import the I/O table from Excel...	—	.xlsx または .csv から変数を宣言し、I/O マッピングを設定（4.6 節）
Protection (password)...	—	パスワードでファイルを読み取り専用または非表示に設定（15.6 節）
EtherCAT (Scan, PDO, ENI)	—	EtherCAT ページを開く（第 11 章）
86HMI screen and bindings	—	86HMI ページを開く（第 13 章）

項目	ショートカット	用途
The sketch's .ino (the 86Duino IDE's)	—	sketch の `.ino` を 86LDmicro で開いて編集（14.5 節）
Settings	—	プロジェクト設定ページを開く（4.2 節）

3.2.4 Build

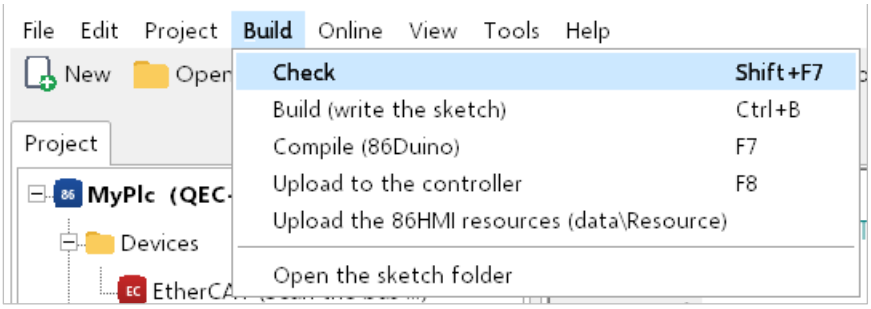


図 3-8 「Build」メニュー

項目	ショートカット	用途
Check	Shift+F7	PLC プログラムをコンパイルするだけで、エラーと警告を（「Messages」ウィンドウに）一覧表示し、ファイルは書き出さない（10.1 節）
Build (write the sketch)	Ctrl+B	先に保存してから、PLC プログラムを sketch 内の `plc86.cpp`、`plc86.h` などに書き出す（10.1、10.2 節）
Compile (86Duino)	F7	ビルド後に 86Duino IDE のコンパイラーでコンパイルし、経過を「Output」に表示（アップロードはしない）
Upload to the controller	F8	ビルド、コンパイルして、ツールバーで選択した COM ポートにアップロード（10.1 節）
Upload the 86HMI resources (data\Resource)	—	86HMI の画像、フォントなどのリソースだけをアップロードし、プログラムは再コンパイルしない（10.6 節）
Open the sketch folder	—	エクスプローラーで sketch フォルダーを開く

*ヒント：コンパイルまたはアップロードの実行中は、「Compile」「Upload」「Upload the 86HMI resources」「Login to the controller」が完了までグレー表示になります。

3.2.5 Online

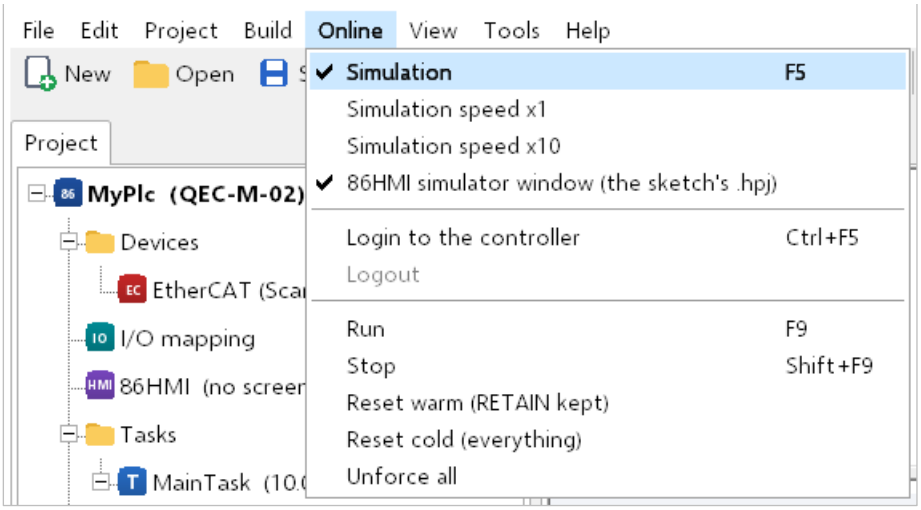


図 3-9 「Online」メニュー（シミュレーション中）

項目	ショートカット	用途
Simulation	F5	プロジェクト全体を PC 上で実行；もう一度クリックするとシミュレーションを終了。シミュレーション中はチェックが付く（9.1 節）
Simulation speed x1	—	シミュレーションを実時間で実行
Simulation speed x10	—	シミュレーション時間を 10 倍速で進める（タイマーなどはシミュレーション時間で動作）
86HMI simulator window (the sketch's .hpj)	—	チェックすると、シミュレーション時に 86HMI シミュレーションウィンドウも開く（13.5 節）
Login to the controller	Ctrl+F5	ツールバーの COM ポート経由でコントローラーにログインし、オンラインモニター；オンライン中にもう一度クリックすると切断（10.4 節）
Disconnect	—	オンラインモニターを終了（オンライン時のみ使用可能）
Run	F9	PLC（シミュレーションまたはコントローラー）の実行を開始
Stop	Shift+F9	プログラムの実行を停止；入出力は通常どおり更新され、出力は最後の値を保持；すべての Motion86 マシンが停止（10.4 節）
Reset warm (RETAIN kept)	—	確認後に再スタート：変数は初期値に戻り、RETAIN 変数は保持
Reset cold (everything)	—	確認後に再スタート：すべての変数（RETAIN を含む）が初期値に戻る
Unforce all	—	強制中のすべての変数を解除（9.3 節）

「Run」から「Unforce all」までは、シミュレーション中またはオンライン時のみ使用でき、編集中はグレー表示です。

3.2.6 View

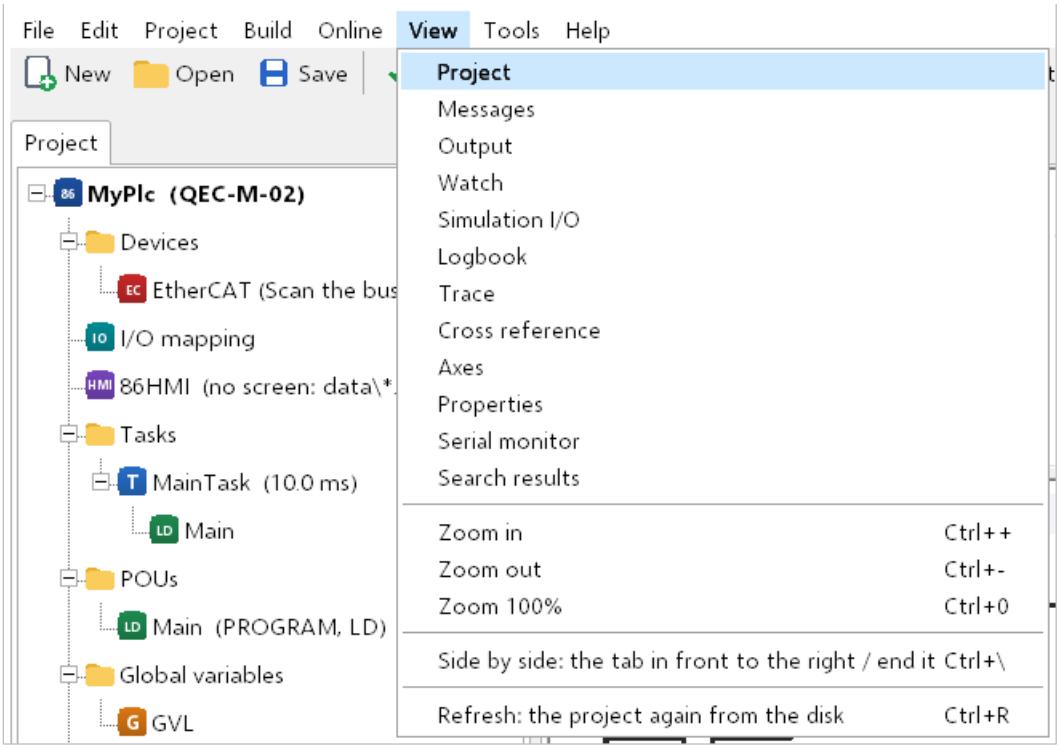


図 3-10 「View」メニュー

項目	ショートカット	用途
Project	—	プロジェクトツリーを表示（3.4 節）
Messages	—	「Messages」ウィンドウを表示：チェックとビルドのエラー、警告
Output	—	「Output」ウィンドウにビルド出力を表示
Watch	—	「Watch」ウィンドウを表示（9.3 節）
Simulation I/O	—	「Simulation I/O」ウィンドウを表示（9.2 節）
Logbook	—	「Logbook」を表示（10.5 節）
Trace	—	「Trace」を表示（9.4 節）
Cross reference	—	「Cross reference」を表示（9.5 節）
Axes	—	「Axes」ウィンドウを表示（12.5、12.6 節）
Properties	—	サイドパネルの「Properties」タブに切り替え（3.5 節）
Serial Monitor	—	「Output」ウィンドウをシリアルモニターに切り替え（10.3 節）
Search results	—	「Search results」を表示（3.7 節）
Zoom In	Ctrl++	インターフェース全体を拡大（3.7 節）
Zoom Out	Ctrl+-	インターフェース全体を縮小
Zoom 100%	Ctrl+0	100% に戻す

項目	ショートカット	用途
Side by side: the tab in front to the right / end it	Ctrl+\	手前のタブを右側に移動して並べて表示；並べて表示中の場合は並べて表示を終了（3.7 節）
Refresh: the project again from the disk	Ctrl+R	プロジェクト全体をディスクから再読み込みし、シリアルポートの一覧を更新（下記の説明を参照）

Refresh（**Ctrl+R**）は、外部エディター、コマンドライン、AI／MCP でプロジェクトファイルが変更された後に使用します。POU、GVL、データ型、ユーザーコード、`project.86ld`、sketch、86HMI をすべて再読み込みし、開いていたタブを復元して、ステータスバーに「Refreshed: the project as it is on the disk」と表示します。未保存の変更がある場合は、先に保存するかどうか確認されます（Yes：保存してから読み込む；No：変更を破棄；Cancel：再読み込みしない）。シミュレーション中またはオンライン時は「Stop the simulation / go offline first」と表示されます。

3.2.7 Tools

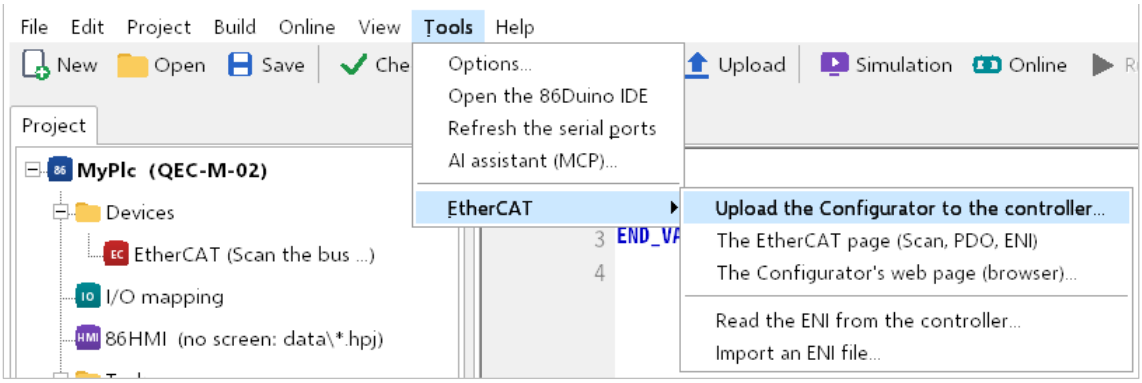


图 3-11 「Tools」メニューと EtherCAT サブメニュー

項目	ショートカット	用途
Options...	—	86Duino IDE フォルダー、言語、アップロード後の自動再接続（2.6 節）
Open the 86Duino IDE	—	86Duino IDE を開き、プロジェクトの `.ino` を読み込む
Refresh the serial ports	—	ツールバーのシリアルポート欄の COM ポートを一覧し直す（コントローラーを接続した後に使用）
AI assistant (MCP)...	—	「AI Assistant」タブを開く：質問、コードの挿入、AI サービスの設定、Claude Desktop（MCP）の設定（15.3、15.4 節）
EtherCAT ▸ Upload the Configurator to the controller...	—	EtherCAT Configurator をコントローラーにアップロード（11.3 節）
EtherCAT ▸ The EtherCAT page (Scan, PDO, ENI)	—	EtherCAT ページを開く（「Project > EtherCAT」と同じ）
EtherCAT ▸ The Configurator's web page (browser)...	—	ブラウザーで Configurator の Web ページを開く（ネットワークモード）
EtherCAT ▸ Read the ENI from the controller...	—	コントローラーから ENI を読み戻し、サブデバイスと PDO を作成（11.4 節）
EtherCAT ▸ Import an ENI file...	—	PC 上の ENI ファイルからサブデバイスと PDO を作成（11.4 節）

3.2.8 Help

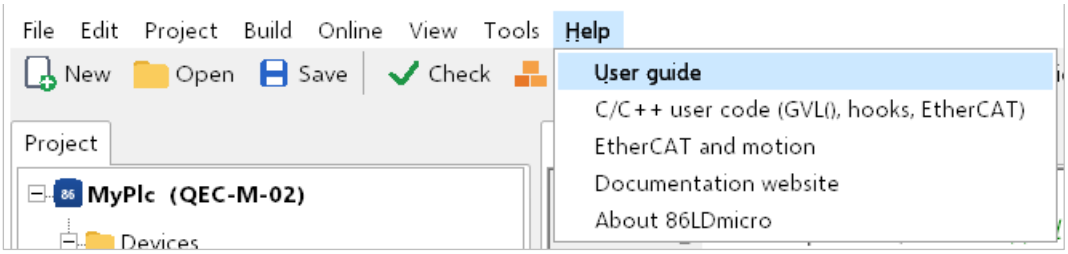


図 3-12 「Help」メニュー

項目	ショートカット	用途
User guide	—	オンラインドキュメントサイトを開く
C/C++ user code (GVL(), hooks, EtherCAT)	—	オンラインドキュメントサイトを開く
EtherCAT and motion	—	オンラインドキュメントサイトを開く
Documentation website	—	オンラインドキュメントサイト https://www.qec.tw/86duino/86ldmicro/ を開く
About 86LDmicro	—	バージョンと著作権情報

最初の 4 項目はいずれもブラウザーで https://www.qec.tw/86duino/86ldmicro/ を開きます。オフラインのときは、`<IDE フォルダー>\tools\LDmicro\tool\docs` にドキュメントとユーザーマニュアルの PDF があります（日本語版は `docs\ja`）。

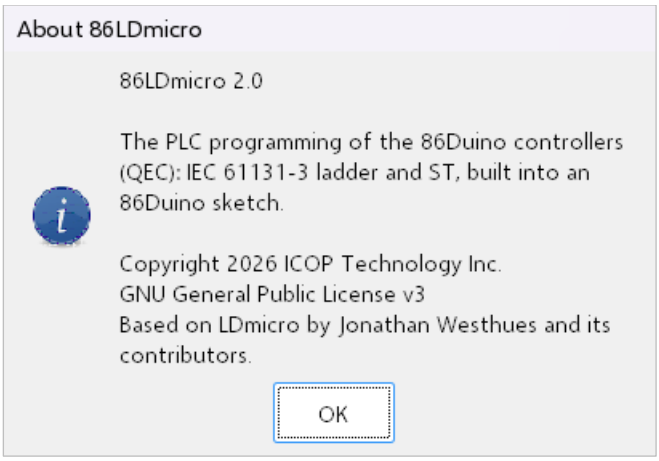


図 3-13 「About 86LDmicro」

3.3 ツールバー

ツールバーには最もよく使うコマンドが集められており、右側にシリアルポート欄があります。ボタンにマウスを重ねると説明が表示されます。

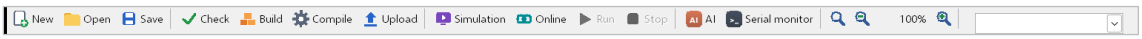


図 3-14　ツールバーとシリアルポート欄

ボタン	ショートカット	機能
New／Open／Save	Ctrl+N／Ctrl+O／Ctrl+S	プロジェクトの新規作成、プロジェクトを開く、保存
Check	Shift+F7	PLC プログラムをコンパイルするだけで、エラーを一覧表示
Build	Ctrl+B	sketch ファイルを書き出す
Compile	F7	86Duino IDE でコンパイル
Upload	F8	コンパイルしてコントローラーにアップロード
Simulate	F5	シミュレーションの開始／終了；シミュレーション中はボタンが押された状態のまま
Online	Ctrl+F5	コントローラーへのログイン／切断；オンライン中はボタンが押された状態のまま
RUN／STOP	F9／Shift+F9	シミュレーション中またはオンライン時に実行／停止
AI	—	「AI Assistant」タブを開く（15.3 節）
Serial monitor	—	「Output」ウィンドウをシリアルモニターに切り替え、Serial.print の出力を見る（10.3 節）
虫眼鏡	Ctrl+F	Find in this tab
Zoom out／100%／Zoom in	Ctrl+-／Ctrl+0／Ctrl++	インターフェース全体をズーム；中央に現在の倍率が表示され、クリックすると 100% に戻る
シリアルポート欄	—	コントローラーの COM ポート；一覧から選ぶか直接入力します。空欄のままアップロードまたはログインすると、先に「The controller's serial port」を確認されます

ウィンドウが狭くなると、ツールバーは次の順に変化します。まず Compile、Upload、Simulate、Online だけが文字を表示し、その他はアイコンのみになります。さらに狭くなるとすべてアイコンのみになり（文字はマウスを重ねたときの説明になります）、必要に応じて 2 段に分かれます。

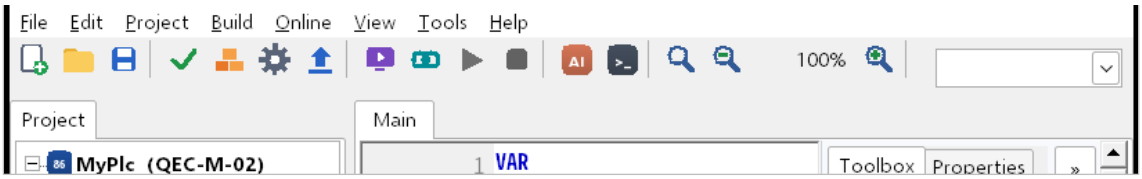


図 3-15　ウィンドウが狭いときのツールバー：アイコンのみ、2 段表示

3.4 プロジェクトツリー

プロジェクトツリー（左側の「Project」ペイン）には、プロジェクトのすべての内容が一覧表示されます。ノードを**ダブルクリック**するとそのページやエディターが開き、**右クリック**するとそのノード専用のメニューが表示されます。プロジェクトを初めて開いたときはすべて展開され、以降は展開状態が記憶されます。

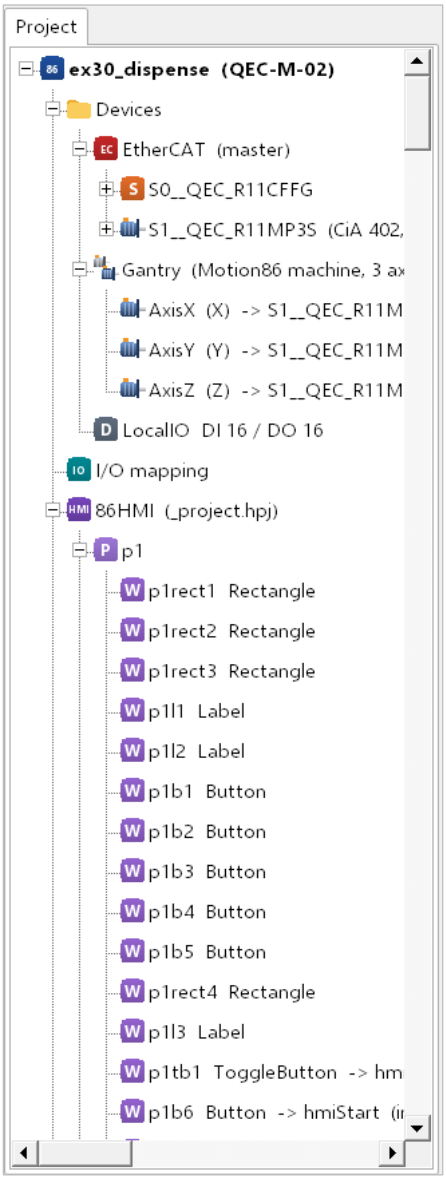


図 3-16　プロジェクトツリー（ex30_dispense）：デバイス、I/O マッピング、86HMI、タスク、POU、グローバル変数、データ型、ユーザーコード、Sketch

ノード	内容	ダブルクリック
プロジェクト名（コントローラー）	例：「MyPlc (QEC-M-02)」	プロジェクト設定ページを開く（4.2 節）
Devices	各デバイス；EtherCAT メインデバイスの下にはサブデバイス、ドライブ、PDO 項目、Motion86 マシンの下には軸があります。EtherCAT がない場合は「EtherCAT (Scan the bus ...)」があります	Devices ページ（4.4 節）；サブデバイスは SubDevice ページを開き、PDO 項目は変数にマッピングし、マシンと軸はそれぞれのページを開く（第 11、12 章）
I/O mapping	変数 ↔ チャネル	I/O mapping ページを開く（4.6 節）

ノード	内容	ダブルクリック
86HMI	`.hpj` のページとウィジェット；ウィジェットの後にバインドされた変数を表示；画面がない場合は「no screen: data*.hpj」と表示	ウィジェットをダブルクリックするとバインドフォームが開く（第 13 章）
Tasks	各タスク（周期または種類）と、その中で順に実行される PROGRAM	タスクをダブルクリックすると設定が開きます（4.3 節）。プログラムをダブルクリックするとエディターが開きます
POU	すべての PROGRAM／FUNCTION_BLOCK／FUNCTION（例：「Main (PROGRAM, LD)」）。保護されている場合は read only／hidden と表示されます	エディターを開きます
Global variables、Data types	GVL、データ型ファイル	エディターを開きます（4.5 節）
User code (C/C++)	`.cpp86` ファイル	エディターを開きます（第 14 章）
Sketch (86Duino IDE)	sketch の `.ino`、`.cpp`、`.h`。`plc86*` には「written by a build」と表示されます	86LDmicro 内で開いて編集します

ファイルに未保存の変更があると、ノード名の後ろに * が付きます。主なノードの右クリックメニューは次のとおりです。

ノード	右クリックメニュー
プロジェクト名	Project settings...／Build (write the sketch)／Compile (86Duino)／Upload to the controller／Simulate／Login to the controller／Open the sketch folder／Open the 86Duino IDE／Close the project
Devices	The devices page／Add a device ▸（localio、eva、com、modbus、ethernet、modbus-server、ethercat、motion86）
個々のデバイス	Edit...／Remove／The devices page／I/O mapping...（Motion86 マシンには「The Axes panel (commissioning)」もあります）
EtherCAT メインデバイス	The EtherCAT page／Upload the Configurator to the controller...／Open the Configurator (RD's web page)／Read the ENI from the controller...／Import an ENI file... ／Edit...／Remove
PDO エントリー	Link to a variable...／Unlink／Add to watch／Cross reference／The EtherCAT page
I/O mapping	Open／Add a link...
Tasks	Add a task...／The tasks page
個々のタスク	Edit the task...／Add a PROGRAM (ladder)...／Add a PROGRAM (ST)...／Remove the task／Run a PROGRAM in this task ▸
タスク内のプログラム	Open／Earlier in the task／Later in the task／Out of this task
POUs（フォルダー）	Add a PROGRAM（ladder／ST／SFC）、Add a FUNCTION_BLOCK（ladder／ST）、Add a FUNCTION（ST）

ノード	右クリックメニュー
個々の POU	Open／Rename...／Cross reference／Remove from the project／Run in the task ▸（PROGRAM のみ。現在所属しているタスクにチェックが付きます）
Global variables、Data types、User code	フォルダー：Add a global variable list...／Add a data type...／Add a user code file (C/C++)...。ファイル：Open／Remove from the project（データ型には「Rename...」もあります）
Sketch (86Duino IDE)	The sketch's .ino (the 86Duino IDE's)／A new .h / .cpp in the sketch...／Open the sketch folder／Open the 86Duino IDE

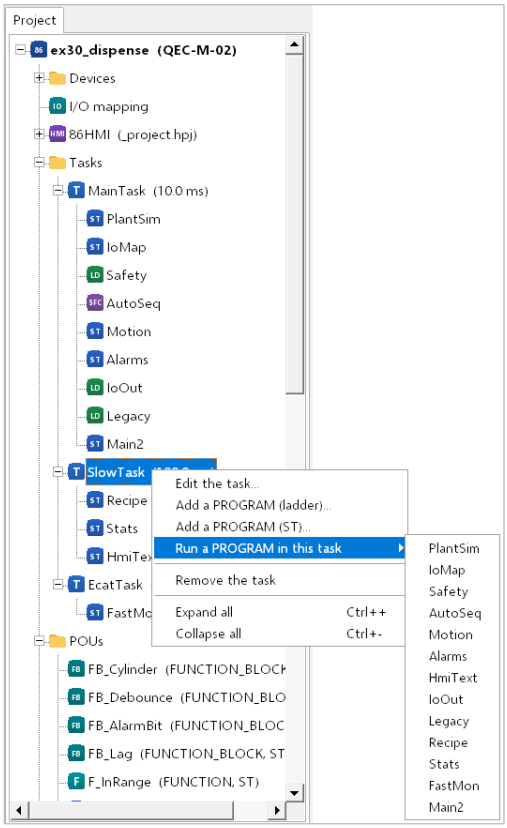


図 3-17　タスクノードの右クリックメニュー：「Run a PROGRAM in this task」サブメニューにすべての PROGRAM が表示されます

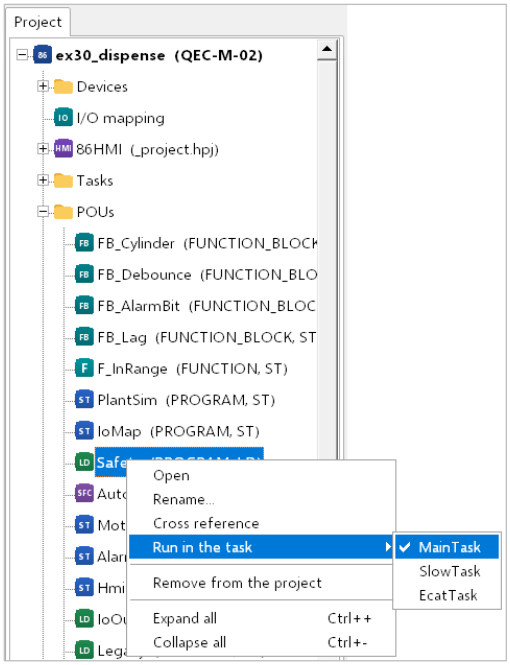


図 3-18　POU の右クリックメニュー：Open、Rename、Cross reference、Remove from the project、Run in the task

- どの右クリックメニュー（空白部分の右クリックを含む）にも、一番下に **Expand all**、**Collapse all** があります。プロジェクトツリーにフォーカスがあるときは **Ctrl++**／**Ctrl+-** も使えます。
- 「Remove from the project」はファイルをプロジェクトから外すだけで、ファイルは `data\86ld` に残ります（実行前に確認されます）。必要な場合は手動で削除してください。
- PDO エントリーまたは 86HMI ウィジェットを選択すると、サイドパネルの「Properties」にその変数と値が表示されます。

3.5 POU エディターとサイドパネル

プロジェクトツリーで POU をダブルクリックすると、ドキュメント領域にそのエディターが開きます。LD と SFC のエディターは、上部が宣言領域（`VAR ... END_VAR`。ST と同じコードエディターで、構文の色分けと補完があります）、下部が図です。ST エディターはプログラム全体のテキストです。LD、ST、SFC のタブにはいずれも右側にサイドパネルがあり、上部に「**Toolbox**」「**Properties**」の 2 つのタブがあります。右上の **»** ボタンでパネルを折りたたみ／展開できます。

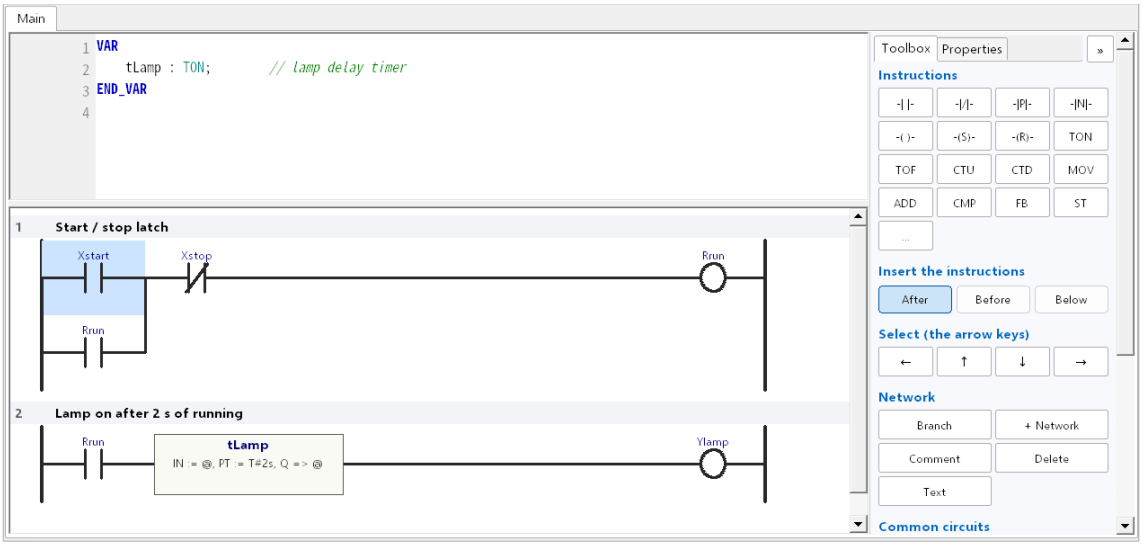


図 3-19　ラダー図エディター：上部に宣言領域、中央にネットワーク、サイドパネルに「Toolbox」

エディター	「Toolbox」の内容	詳細
ラダー図（LD）	Instructions（接点、コイル、TON、TOF、CTU、CTD、MOV、ADD、CMP、FB、ST。「...」で全命令を一覧表示）、Insert the instructions（After、Before、Below）、Select (the arrow keys)（←↑↓→）、Network（Branch、+ Network、Comment、Del、Text）、Common circuits（Start / stop latch などのテンプレート）	第 7 章
ストラクチャードテキスト（ST）	ST snippets（IF、CASE、FOR、タイマーなど）。ダブルクリックで挿入します	8.1 節
シーケンシャルファンクションチャート（SFC）	Chart：+ Step、Alternative、Parallel、+ Branch、Jump、Edit、Del、Text	8.4 節

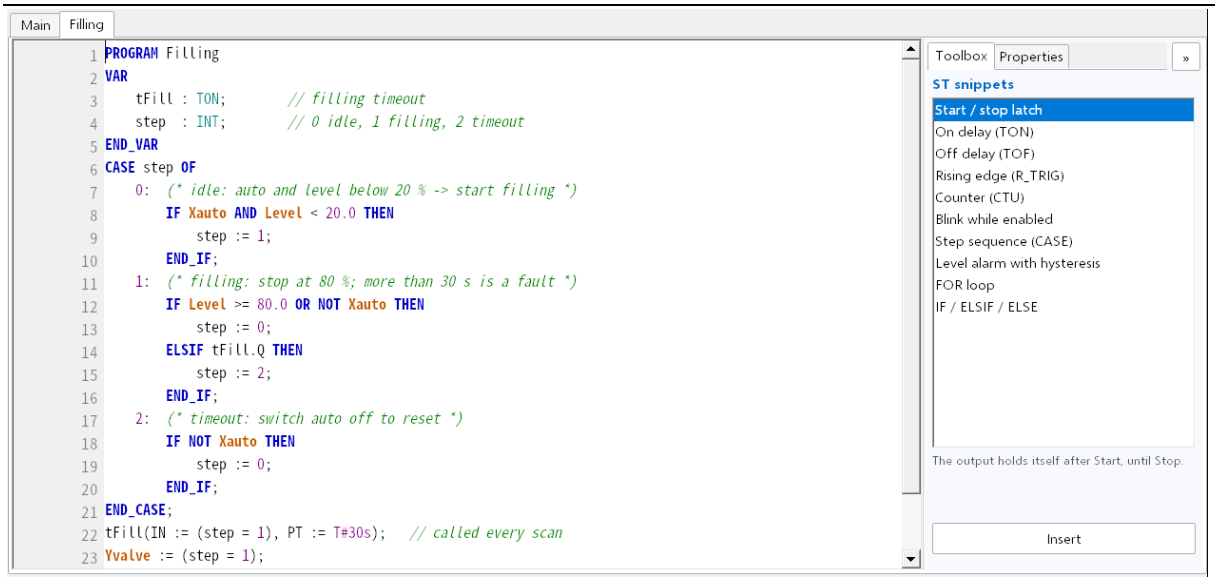


図 3-20　ST エディターと右側の「ST snippets」

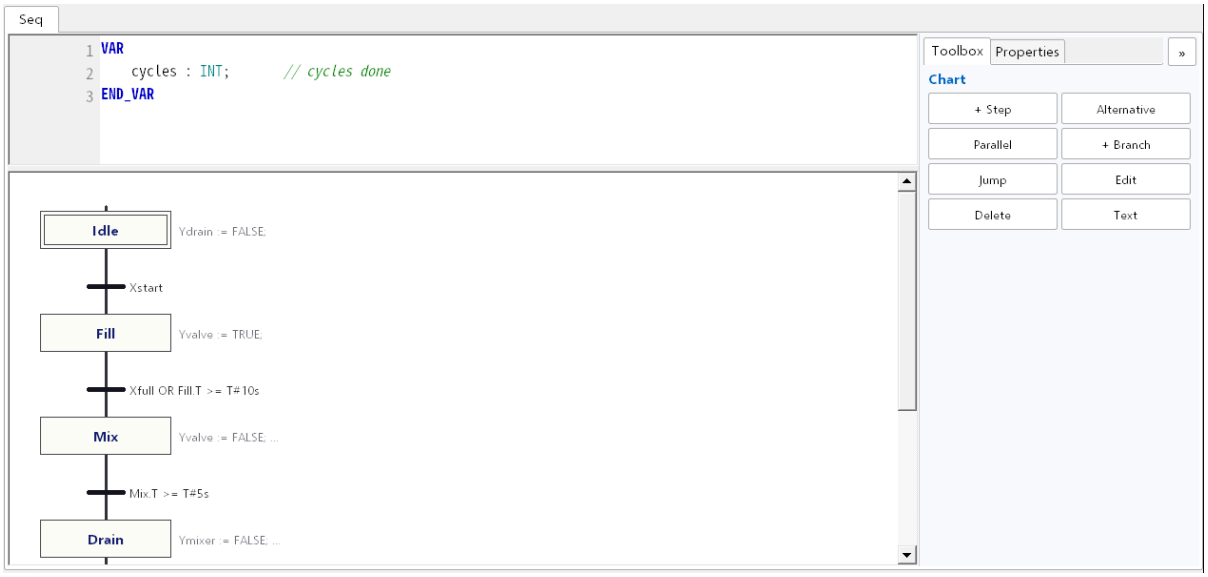


図 3-21　SFC エディター：上部に宣言領域、中央にチャート、右側に「Chart」ツール

「**Properties**」タブには、選択したラダー図の要素、Watch の行、EtherCAT エントリー、86HMI ウィジェットの変数について、型、宣言場所、初期値、コメント、チャンネル（I/O マッピング）、86HMI バインド、およびシミュレーション中またはオンライン中の値が表示されます。何も選択していないときは「Select a ladder element, a row of the watch, an EtherCAT entry or an 86HMI widget: its variable, value, I/O and 86HMI are here.」と表示されます。

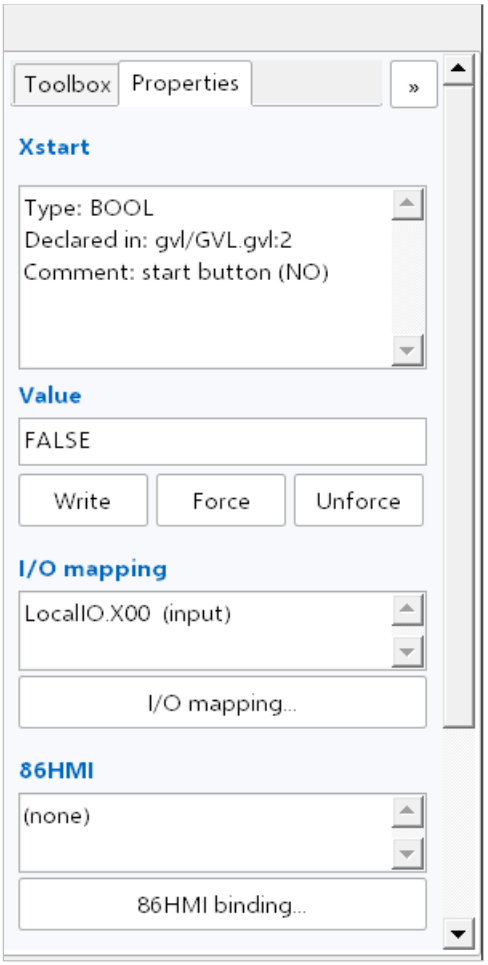


図 3-22　サイドパネルの「Properties」：選択した要素の変数、型、I/O マッピング、値

ラダー図の要素、コード内の変数、Watch の行のいずれを右クリックしても、同じ変数メニューが表示されます（変数 `Rrun` の例）：Write Rrun...、Force Rrun...、Unforce（この 3 項目はシミュレーション中またはオンライン中のみ）、I/O mapping of Rrun...、86HMI binding of Rrun...、Add Rrun to the watch window、Add to the trace、Where is Rrun used?、Its declaration、Properties。

*ヒント：ラダー図と SFC では Ctrl+T でテキストモード（POU 全体の .lad／.sfc テキスト）に切り替わり、「Back to the diagram」でグラフィカル編集に戻ります（7.1 節、8.4 節）。

3.6　ツールウィンドウ

下部のツールウィンドウはタブで切り替えます。閉じたウィンドウは「View」メニューから再表示できます。タブを右クリックして「**Float (a window of its own)**」を選ぶと独立したウィンドウになり（たとえば 2 台目のモニターへドラッグできます）、フローティングウィンドウを閉じると元の位置に戻ります。領域間の区切り線をドラッグするとサイズを調整できます。

ウィンドウ	用途	詳細
Messages	チェックとビルドのエラー、警告（メッセージ、ファイル、行）。ダブルクリックでエラーの要素または行へジャンプします。右クリック：Go to、Fix with AI (copies the question)、Copy the message	9.6 節
Output	上部で「Build output」と「Serial monitor」を切り替えます：ビルド、コンパイル、アップロードの経過と Serial.print の出力内容	10.1 節、10.3 節
Watch	上部に変数名を入力して **Add** をクリックします。変数、値、型を表示します。BOOL はダブルクリックで切り替え、右クリックで書き込み、強制、Trace への追加、Watch からの削除ができます。**Unforce all**	9.3 節
Simulation I/O	シミュレーション中、各 I/O マッピングの実際の点（チャンネル、変数、入力／出力）を一覧表示します。ダブルクリックでデジタル入力の切り替えやアナログ値の入力ができます	9.2 節
Logbook	コントローラー（またはシミュレーター）のイベント：起動、EtherCAT 状態、エラー、ウォッチドッグ、強制解除	10.5 節
Trace	Watch 内の変数を右クリックして「Add to the trace」を選びます（最大 8 個）。100 ms ごとにサンプリングし、直近 1 分間を表示します。右クリック：Pause、Clear the samples、Remove all variables、Copy as CSV	9.4 節
Cross reference	変数が各 POU で読み出し、書き込み、宣言されている場所（Ctrl+Shift+F）	9.5 節
Axes	全軸のサーボオン（励磁）状態、位置、ステータス。直接コマンドを送ることもできます	12.5 節、12.6 節
Search results	プロジェクト内検索（Ctrl+Shift+H）の結果。ダブルクリックでその行へジャンプします	3.7 節、9.5 節

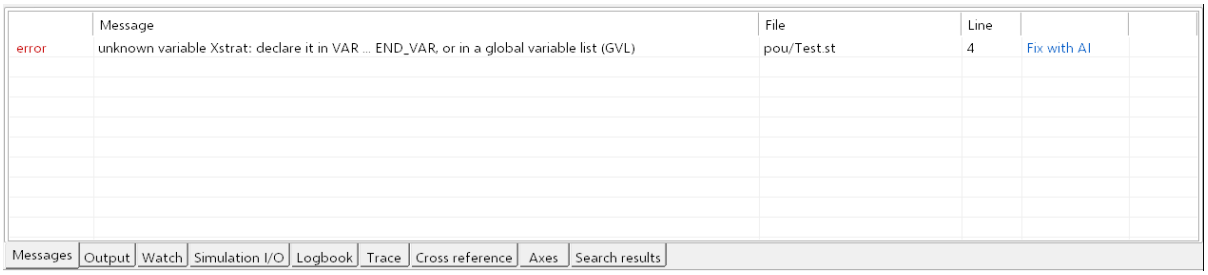


図 3-23　「Messages」ウィンドウ：1 行に 1 つのエラー。ダブルクリックでその要素へジャンプ

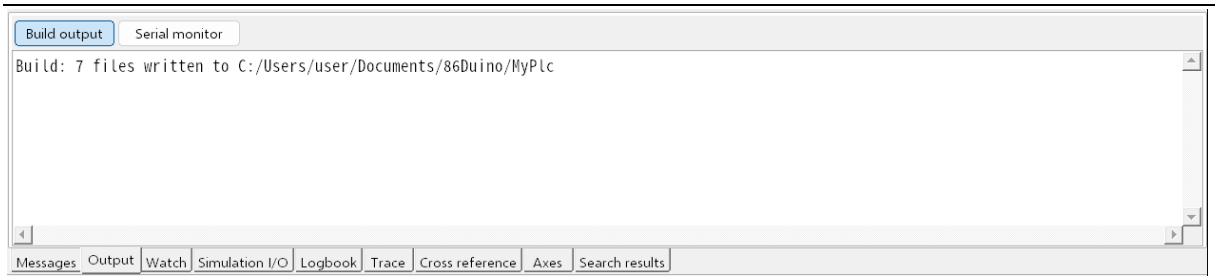


図 3-24　「Output」ウィンドウの「Build output」：ビルド、コンパイル、アップロードの経過

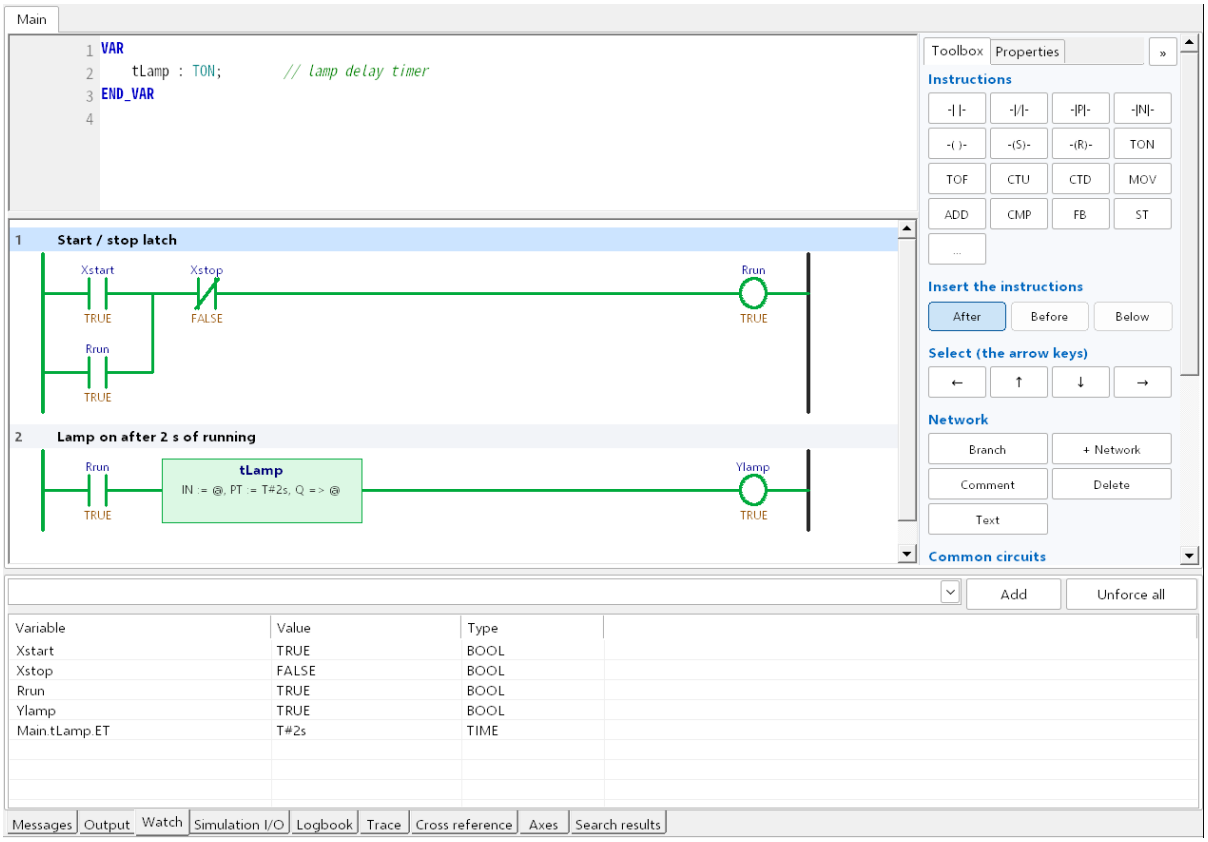


図 3-25　「Watch」ウィンドウ：変数を追加すると値がリアルタイムに表示されます（書き込み、強制も可能）

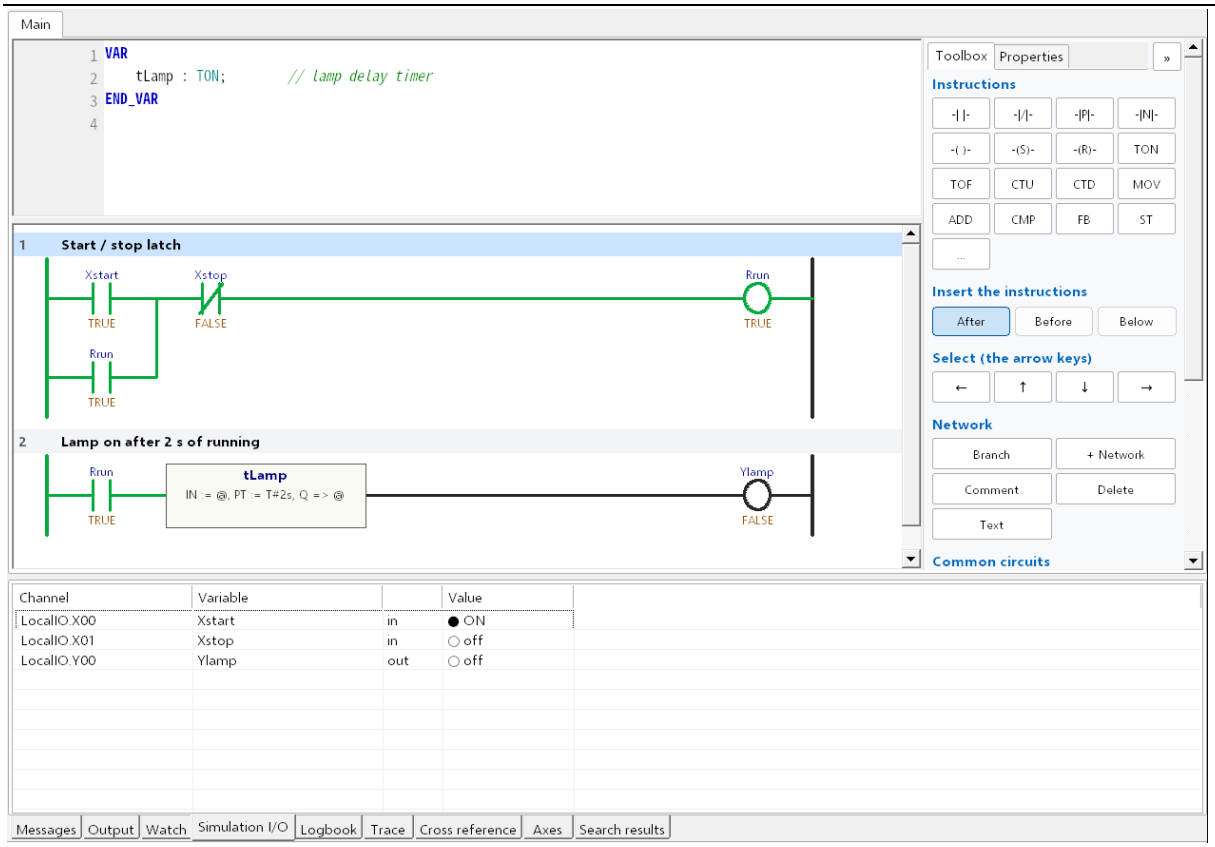


図 3-26 「Simulation I/O」：LocalIO.X00 などの実際の点

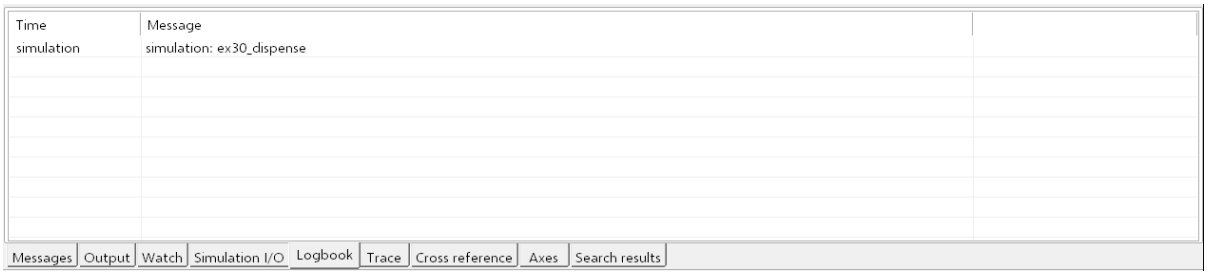


図 3-27 「Logbook」：EtherCAT メインデバイスの起動メッセージとプログラムの実行開始

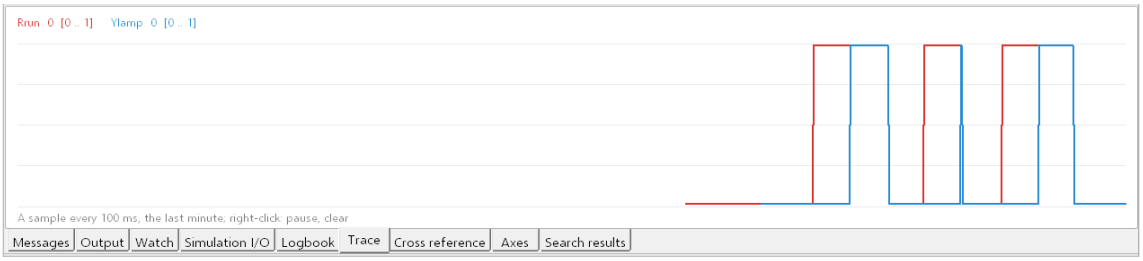


図 3-28 「Trace」：Watch から追加した変数の波形

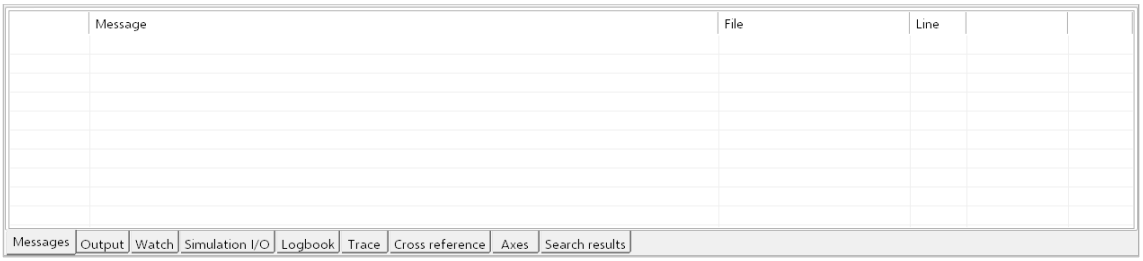


図 3-29 「Cross reference」：変数 Rrun が読み出し、書き込み、宣言されている場所

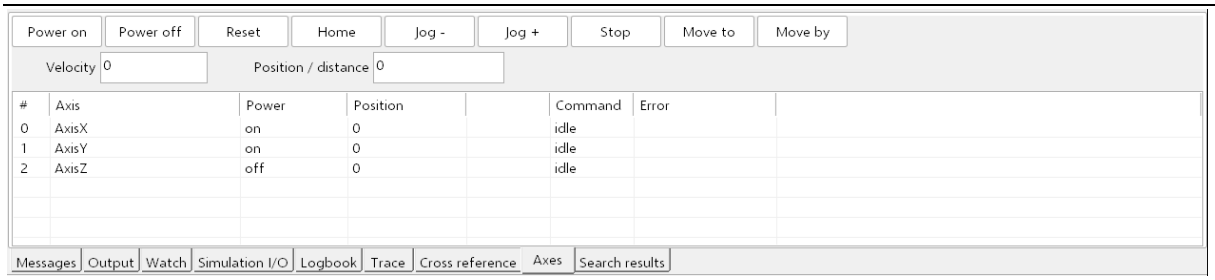


図 3-30　「Axes」ウィンドウ

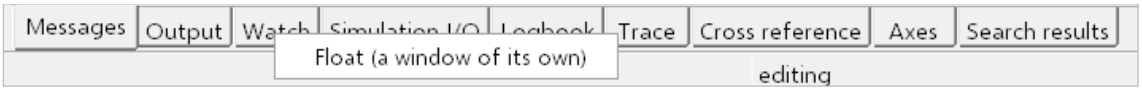


図 3-31　ツールウィンドウのタブの右クリックメニュー：Float (a window of its own)

3.7　タブ、左右分割、ズーム、検索

3.7.1　タブ

- POU、GVL、設定ページごとに 1 つのタブが開きます。**Ctrl+Tab**／**Ctrl+Shift+Tab** でタブを切り替え、**Ctrl+W** またはタブ上での**マウスの中ボタン**クリックで閉じます。
- タブの右クリックメニュー：Close（Ctrl+W）、Close the others、Close all、Split: move to the right side（すでに左右分割しているときは「Move to the right side」／「Move to the left side」）、End the split (all on the left)。
- 開いているタブと分割の状態はプロジェクトごとに記憶され、次回プロジェクトを開いたときに復元されます。

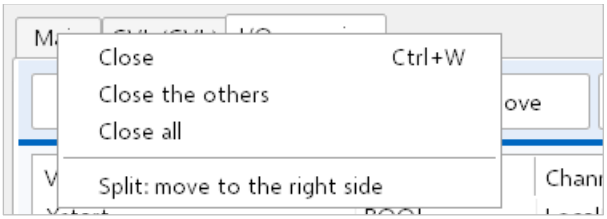


図 3-32　タブの右クリックメニュー

3.7.2　左右分割

1. タブを右クリックして「**Split: move to the right side**」を選ぶか、タブをドキュメント領域の右側 1/3 の位置へドラッグするか、「View > Side by side」（**Ctrl+**）を使います。タブが 2 つ以上開いている必要があります。
2. 中央の区切り線はドラッグできます。タブは右クリックの「Move to the left side／Move to the right side」またはドラッグで左右間を移動できます。
3. クリックした側が操作対象の側になります（上部の青い線で表示）。保存、検索、Ctrl+Tab はその側に対して働きます。シミュレーション中やオンライン中は、両側の値が同時に更新されます。
4. 片側のタブをすべて閉じるか、右クリックで「End the split (all on the left)」を選ぶと、1 列表示に戻ります。

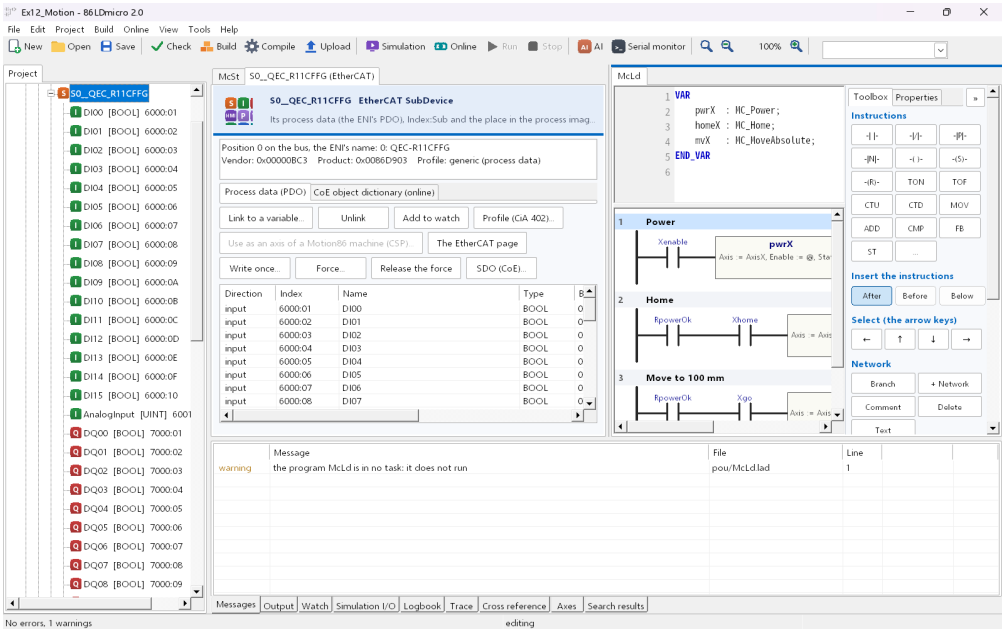


図 3-33　左右分割：左にサブデバイスのページ、右にラダー図

3.7.3　ズーム

Ctrl+マウスホイール、**Ctrl++**／**Ctrl+-**／**Ctrl+0**、またはツールバーの Zoom out、Scale、Zoom in ボタンを使います。コード、ラダー図、設定ページ、プロジェクトツリー、ツールウィンドウがまとめてズームされます。倍率は記憶され、次回起動時にも使われます。

3.7.4 Find in this tab（Ctrl+F）

1. **Ctrl+F**（またはツールバーの虫眼鏡、「Edit > Find in this tab...」）を押すと、タブの上部に検索バーが表示されます。先にテキストを選択してから押すと、そのテキストがすぐに検索されます。
2. 検索するテキストを入力します。必要に応じて「Match case」にチェックを入れます。右側に「何件目／全何件」が表示され、見つからないときは「Not found」と表示されます。
3. **Enter**、**F3**、▼ で次を検索、**Shift+Enter**、**Shift+F3**、▲ で前を検索、**Esc** または ✕ で閉じます。ラダー図では、一致する接点、コイル、ネットワークタイトルが順に選択されます。

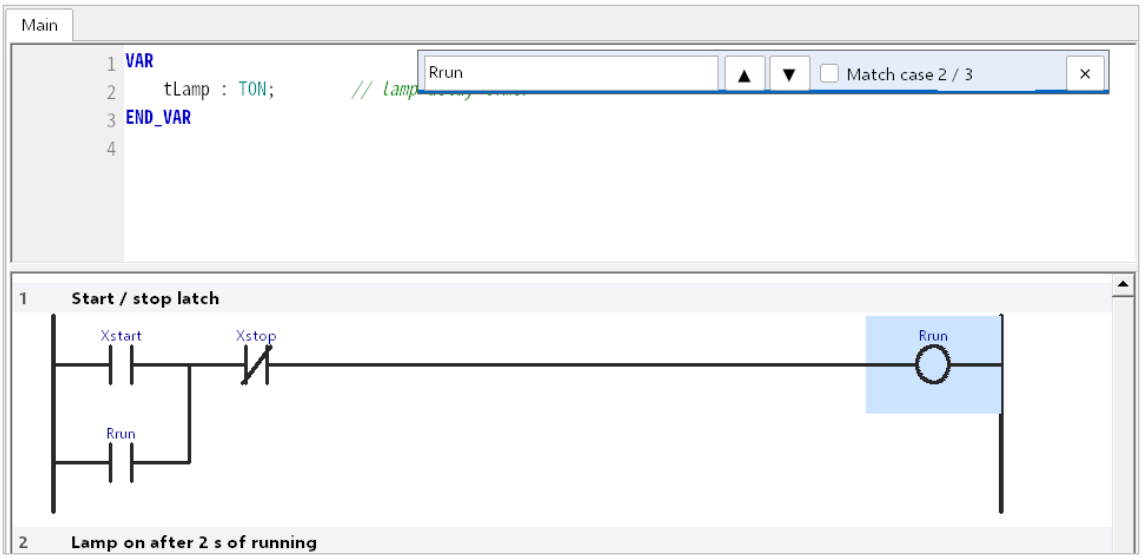


図 3-34　検索バー：ラダー図で Rrun を検索（3 件中 1 件目）

3.7.5 プロジェクト内の変数検索（Find a variable in the project、Ctrl+Shift+F）とプロジェクト内検索（Find in the project、Ctrl+Shift+H）

機能	検索対象	結果	用途
Find in this tab（Ctrl+F）	現在のタブのテキストまたはラダー図の要素	検索バー	1 つのプログラム内で順に探す
Find a variable in the project（Ctrl+Shift+F）	1 つの変数の読み出し、書き込み、宣言の場所	Cross reference	変数の書き込み元を確認する
Find in the project（Ctrl+Shift+H）	プロジェクト全体の任意のテキスト：すべての POU（ネットワークタイトル、コメントを含む）、GVL、データ型、ユーザー C/C++、sketch の .ino	Search results	コメント、文字列、ファンクション、名前がはっきりしないものを探す



図 3-35 「Find a variable」：変数名を入力

1. **Ctrl+Shift+H** を押すと、下部が「Search results」に切り替わります（先にプログラム内のテキストを選択してから押すと、そのテキストがすぐに検索されます）。
2. テキストを入力し、**Enter** を押すか **Find** をクリックします。必要に応じて「Match case」または「Whole word」にチェックを入れます（たとえば `Pump` を検索するときに `PumpSpeed` を表示しないため）。右上に見つかった件数が表示されます。
3. 行を**ダブルクリック**する（または選択して Enter を押す）と、そのファイルが開いてその行へジャンプします。ST と C/C++ ではその行が選択され、ラダー図ではそのネットワークまでスクロールして選択されます。

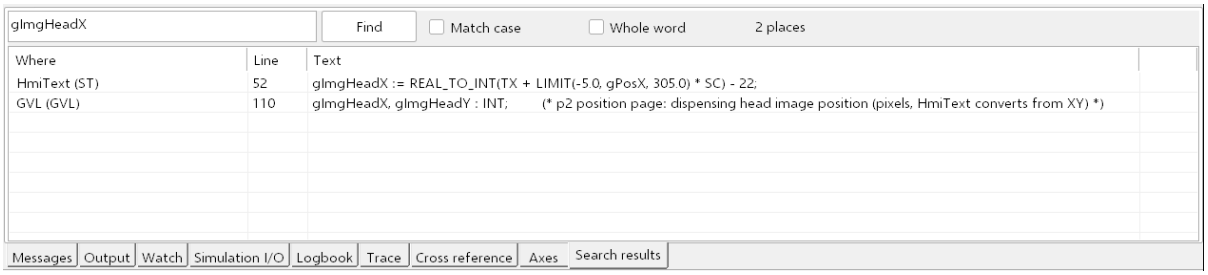


図 3-36 「Search results」：2 件見つかった例（ST プログラムとグローバル変数リスト）

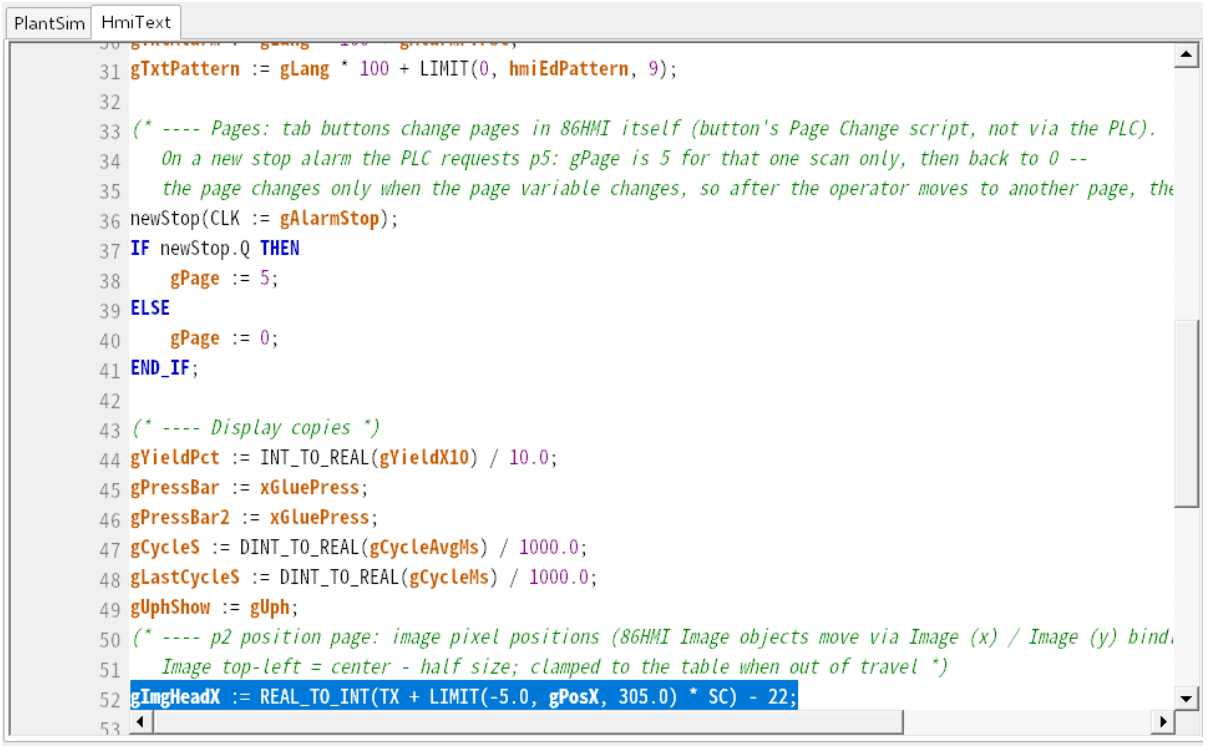


図 3-37　行をダブルクリック：その ST プログラムが開いて該当行が選択される

*ヒント：検索は探すだけで、置換はしません。変数名を変更するには、結果から 1 つずつ開いて修正し、修正後に「Check」（Shift+F7）で漏れがないことを確認してください。

外部でファイルを変更したあとは、**Ctrl+R**（View > Refresh）を押してディスクからプロジェクトを再読み込みします（3.2.6 節）。

第 4 章　プロジェクトと設定

この章では、86LDmicro プロジェクトを構成するファイルと、プロジェクト設定、タスク、デバイス、グローバル変数とデータ型、I/O マッピングの設定方法を説明し、最後に POU とユーザーコードの追加方法を説明します。

4.1　プロジェクトのファイル構成

86LDmicro のプロジェクトは、所属する 86Duino sketch のフォルダーに置かれます。すべて UTF-8 のテキストファイルなので、git などのツールでそのままバージョン管理できます。第 5 章の `MyPlc` を例にします（既定の場所は ドキュメント`\86Duino`）。

```
ドキュメント\86Duino\MyPlc\
    MyPlc.ino                       plc86_setup()／plc86_loop() を呼び出す（存在しない場合のみ 1 回書き出し、以降はユーザーが管理）
    plc86.h、plc86.cpp              PLC プログラム（ビルドで生成、毎回上書き。手動で編集しないでください）
    plc86_rt.h、plc86_rt.cpp        86LD run-time（ビルドで生成）
    plc86_ecat.h、plc86_ecat.cpp    EtherCAT（ビルドで生成）
    myhmi.h、myhmi.cpp              86HMI Editor の「Code Generation」が書き出す（86HMI 画面がある場合）
    data\
        *.hpj                       86HMI 画面（ある場合）
        Resource\                   86HMI の画像、フォントなどのリソース（ある場合）
        86ld\
            project.86ld            プロジェクト設定（JSON）：コントローラー、タスク、デバイス、I/O マッピング、バインド
            pou\                    POU：*.lad（ラダー図）、*.st（ST）、*.sfc（SFC）
            gvl\                    グローバル変数リスト *.gvl
            dut\                    データ型 *.dut
            usercode\               ユーザー C/C++ *.cpp86
            ethercat\               インポートした ENI のコピー、CoE オブジェクトディクショナリのキャッシュ
            build\                  plc86.symbols.json（オンラインモニター用のシンボルテーブル）、コンパイル出力
```

新規作成したプロジェクトには `MyPlc.ino`、`project.86ld`、`pou\Main.lad`、`gvl\GVL.gvl` しかありません。その他のフォルダーは、対応するファイルの追加、ENI の読み込み、ビルドのときに作成されます。新規プロジェクトの `project.86ld` は次のとおりです。

```
{
  "format": "86LDmicro project",
  "version": 1,
  "name": "MyPlc",
  "target": {
    "board": "QEC-M-02",
    "mcu": "Vortex86EX2"
  },
  "tasks": [
```

```
    {
      "name": "MainTask",
      "kind": "cyclic",
      "intervalUs": 10000,
      "priority": 1,
      "programs": ["Main"]
    }
  ],
  "pous": ["pou/Main.lad"],
  "gvls": ["gvl/GVL.gvl"],
  "duts": []
}
```



図 4-1　sketch フォルダー：.ino、ビルドで生成された plc86* ファイル、data\86ld

- `plc86*` ファイルと `data\86ld\build` はビルドのたびに再生成されるため、バージョン管理に含めなくてもかまいません。
- `.ino` はユーザーのファイルです。86LDmicro は存在しない場合に 1 回だけ書き出し、その後は `plc86_setup()`、`plc86_loop()` を呼び出していることだけを確認します（14.5 節）。
- プロジェクトから外したファイルは削除されず、`data\86ld` に残ります。必要な場合は手動で削除してください。
- これらのテキストファイルは、外部エディター、コマンドライン、AI／MCP でも変更できます。変更後は 86LDmicro で **Ctrl+R**（View > Refresh）を押して再読み込みします（3.2.6 節）。ラダー図 `.lad` のテキスト形式は 7.1 節を参照してください。

4.2 プロジェクト設定

プロジェクトツリーでプロジェクト名（一番上のノード）をダブルクリックするか、「Project > Settings」を選ぶと、「Project settings」ページが開きます。ページには、プロジェクト名、フォルダー、コントローラー、オンラインモニター、86Duino IDE の場所が表示されます。上部には 3 つのボタンがあります。

ボタン	機能
Edit...	「Project settings」ダイアログを開きます（下記参照）
Options...	「Options」を開きます（86Duino IDE フォルダー、言語。2.6 節）
Open the folder	エクスプローラーで sketch フォルダーを開きます

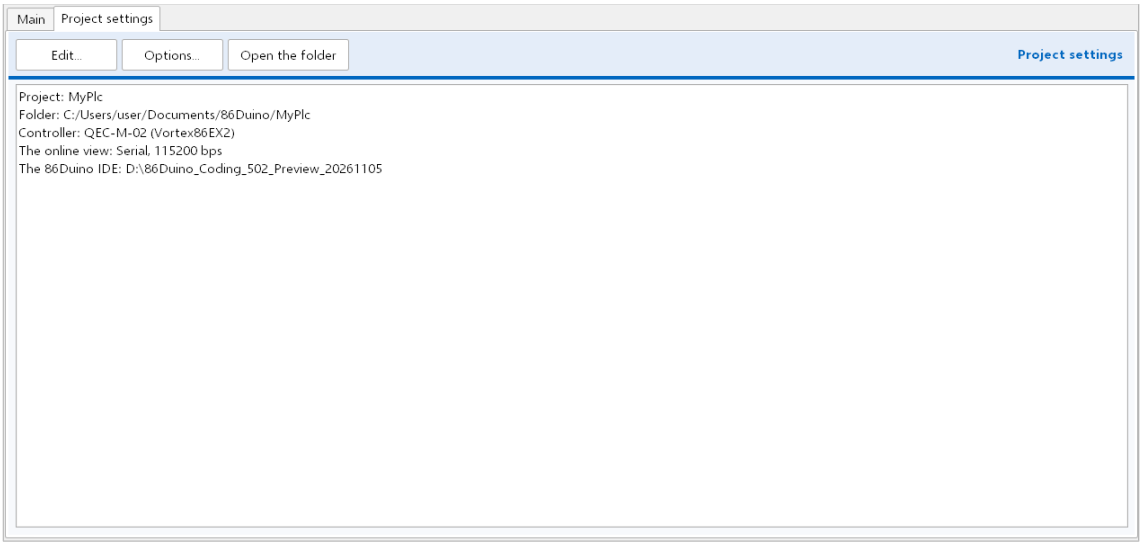


図 4-2　「Project settings」ページ：プロジェクト、フォルダー、コントローラー、オンラインモニター、86Duino IDE

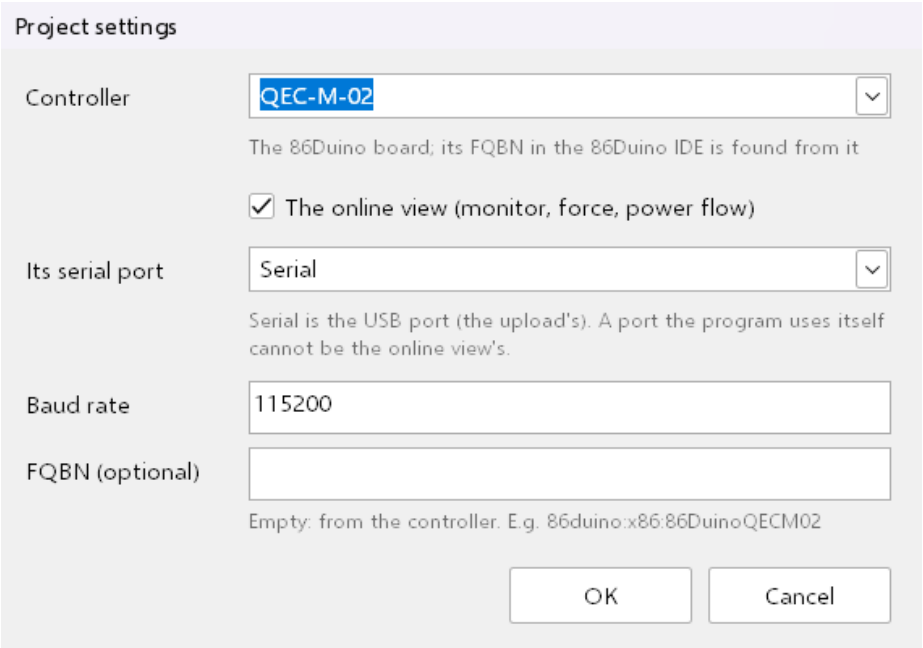


図 4-3　「Project settings」ダイアログ：コントローラー、オンラインモニター、シリアルポート、ボーレート、FQBN

項目	既定値	説明
Controller	QEC-M-02	コンパイル対象（86Duino IDE の FQBN）とローカル I/O を決めます。一覧には QEC-M-02、QEC-M-01、QEC-M-043T、QEC-M-070T、QEC-M-090T、QEC-M-150T と、86Duino IDE の boards.txt にあるその他の QEC ボード（PPC シリーズなど。IDE が新しいバージョンであれば自動的に表示されます）があります
The online view (monitor, force, power flow)	オン	チェックを入れると、アップロードするプログラムにオンラインモニター機能が組み込まれます（10.4 節）
Its serial port	Serial	オンラインモニター用のポート：`Serial` は USB（アップロード用のポート。推奨）です。Serial1、Serial2、Serial3、SerialCOM も選択できます。プログラム自身が使用するシリアルポートはオンラインモニター用ポートにできません
Baud rate	115200	86LDmicro はオンライン接続時にこのボーレートを自動的に使用します。変更後は再ビルドとアップロードが必要です
FQBN (optional)	空欄	空欄のときはコントローラーから自動的に決まります。特殊なボードでは直接指定できます（例：`86duino:x86:86DuinoQECM02`）

*注意：sketch の `.ino` が `plc86_setup()` の後で再び `Serial.begin()` を呼び出すと、オンラインモニターのボーレート設定が上書きされ、接続に失敗します（10.4 節）。

86Duino IDE から起動したときに、IDE で選択したボードがプロジェクトのコントローラーと異なる場合、86LDmicro は IDE のボードに切り替えるかどうかを確認します（2.5 節）。

4.3　タスク

タスクは、PROGRAM をいつ、どの順序で実行するかを決めます。プロジェクトツリーで「Tasks」をダブルクリックする（または「Project > Tasks」を選ぶ）とタスクページが開きます。タスクごとに 1 行で、タスク、種類、周期、ウォッチドッグ、プログラムが表示されます。上部のボタンは **Add...**、**Edit...**、**Remove**、**Up**、**Down** です。行をダブルクリックすると **Edit...** と同じです。

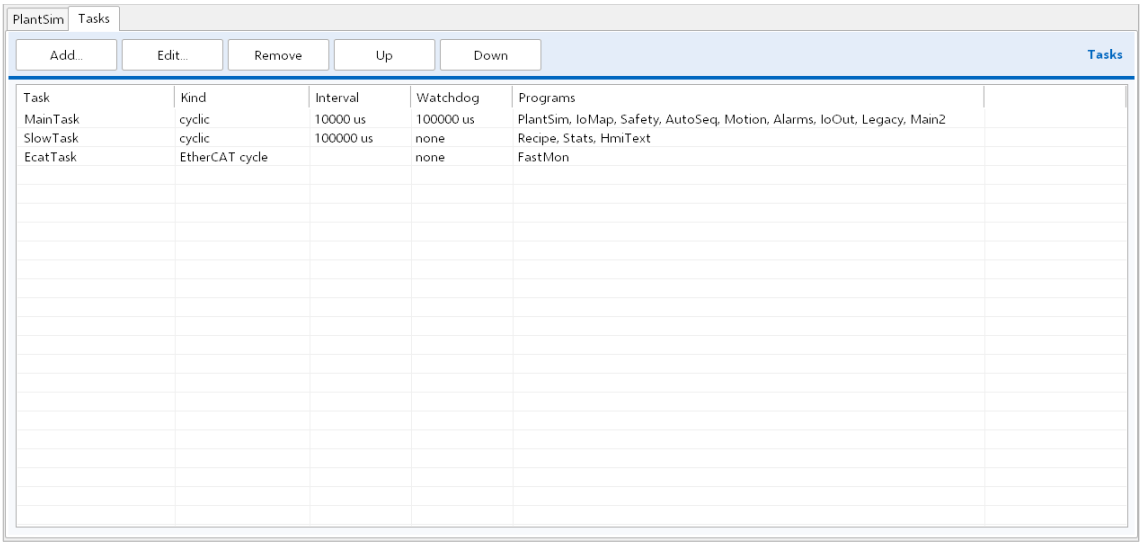


図 4-4　タスクページ：タスク、種類、周期、ウォッチドッグ、プログラム

項目	説明
Name	英字、数字、_（例：`MainTask`）
Kind	cyclic、freewheel、EtherCAT cycle（下表参照）
Interval (us)	cyclic タスクの実行間隔。100 us～10 s（100～10000000 を入力）。新規プロジェクトの MainTask は 10000（10 ms）です
Watchdog (us)	1 回のスキャンがこの時間を超えると PLC を停止します。0 は使用しないことを意味します（下記参照）
Programs, in order	カンマ区切りの PROGRAM 名。順に実行されます。項目の下にプロジェクト内のすべての PROGRAM が表示されます

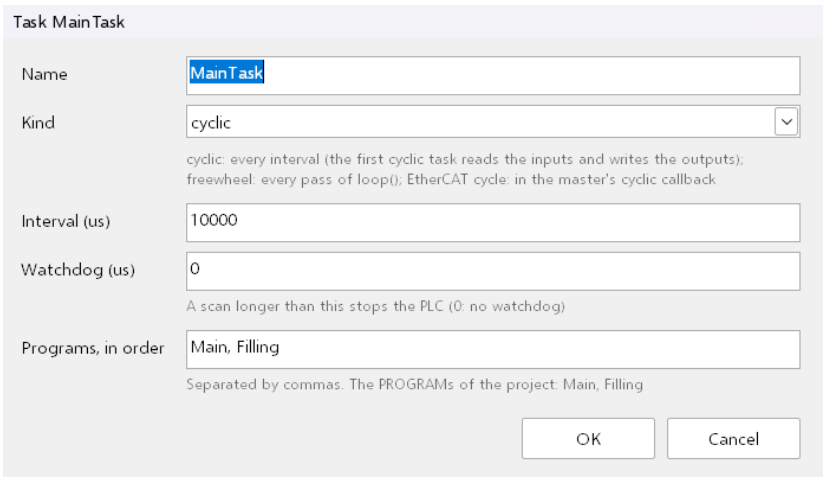


図 4-5　タスクの設定：名前、種類、周期、ウォッチドッグ、プログラム（実行順）

種類	実行タイミング	用途
cyclic	「周期」ごとに 1 回実行	一般的な PLC ロジック。**最初の cyclic タスク**がプログラムの前にすべての入力を読み込み、後にすべての出力を書き出します
freewheel	loop() の 1 回ごと	バックグラウンドの低優先度処理
EtherCAT cycle	EtherCAT メインデバイスの周期コールバック（86EVA または EtherCAT デバイス）	EtherCAT と同期する必要がある制御。他のタスクに割り込むため、短く保ってください

- プロジェクトには **cyclic** タスクが少なくとも 1 つ必要です。**EtherCAT cycle** タスクは最大 1 つです。
- 1 つの PROGRAM は 1 つのタスクにしか入れられません（同じタスクに 2 回入れることもできません）。そうでない場合はチェック時にエラーになります。FUNCTION_BLOCK と FUNCTION はタスクに入れず、PROGRAM から呼び出します。
- 「Project > Add a PROGRAM」で作成したプログラムは、最初のタスクの末尾に自動的に追加されます。
- プロジェクトツリーでも操作できます：タスクノードを右クリック >「Run a PROGRAM in this task ▸」、タスク内のプログラムを右クリック >「Earlier in the task」「Later in the task」「Out of this task」、POU を右クリック >「Run in the task ▸」（3.4 節）。
- cyclic タスクの 1 回のスキャンが周期を超えると、コントローラーの Logbook に「task <タスク名>: a scan took … us (the interval is … us)」が記録されます（初回と、その後 1000 回ごとに 1 回）。

4.3.1 ウォッチドッグ

ウォッチドッグは、遅すぎるプログラムを見つけるための機能です（スキャンタイムアウトのチェック）。コントローラー上でのみ、cyclic タスクに対してのみ判定します（シミュレーションでは判定しません）。

- 判定する時間は、1 回のスキャンにおけるプログラム自体の実行時間です。EtherCAT 周期割り込み、EEPROM への書き込み（RETAIN）、モーション制御のメールボックス、86HMI 画面の更新にかかる時間は含まれません。
- 各タスクの最初の 3 回のスキャンは判定しません（起動時の RETAIN の読み込み、初回の初期値設定など）。
- 設定時間を超えると、PLC は STOP になります（STOP を押したのと同じで、プログラムは実行されなくなり、出力は最後の値を保持し、すべての Motion86 マシンが停止してジョグも終了します。10.4 節）。Logbook には、タスク名、そのスキャンの時間、最も遅いプログラムが記録されます（例：「MainTask watchdog: STOP, 120 ms (3 other), AutoSeq 110 ms」）。
- オンライン中は、86LDmicro の「Output」に「The PLC is in STOP: a scan took longer than the task's watchdog (Project > Tasks). RUN (F9) starts it again.」と表示され、ステータスバーに「The controller's watchdog stopped the PLC: see Output」と表示されます。
- 原因を取り除いたら **RUN**（F9）を押して再実行します。対処方法：各スキャンの処理を減らす（ループや文字列処理を複数回に分ける）か、ウォッチドッグ時間を長くします。

> ***ヒント：**ウォッチドッグ時間は通常のスキャン時間よりも十分に長く設定してください（例：周期 10 ms のタスクで 100000 us）。異常を検出するためだけに使います。通常のスキャン時間は、オンラインモニター中にステータスバーに表示されます（3.1 節）。

4.4 デバイス

デバイスは、プロジェクトで使用するハードウェアと通信です：ローカル I/O、EtherCAT、86EVA、シリアルポート、Modbus、ネットワーク、Motion86 マシン。プロジェクトツリーで「Devices」をダブルクリックする（または「Project > Devices」を選ぶ）とデバイスページが開きます。デバイスごとに 1 行（デバイス、型、設定の概要）で、上部のボタンは **Add...**、**Edit...**、**Remove** です。

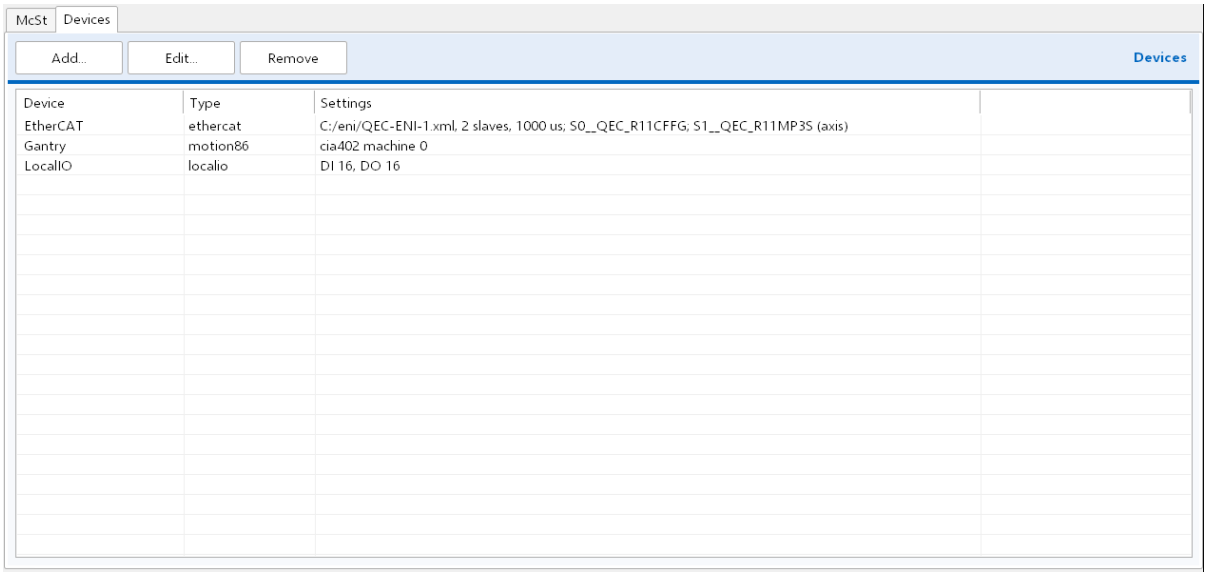


図 4-6　デバイスページ：localio（DI 16, DO 16）

1. **Add...** をクリックします（またはプロジェクトツリーの「Devices」を右クリック >「Add a device ▸」で種類を直接選択します）。
2. 「Add a device」の「Type」で種類を選び、**OK** をクリックします。
3. 続いてそのデバイスの設定フォームが表示されます。項目ごとに 1 行で（チェックボックスはオン／オフ、数値と文字列は直接入力）、リスト形式の項目（シリアルポートの一覧、Modbus サブデバイスなど）は JSON テキストで編集します。**OK** をクリックすると追加されます。

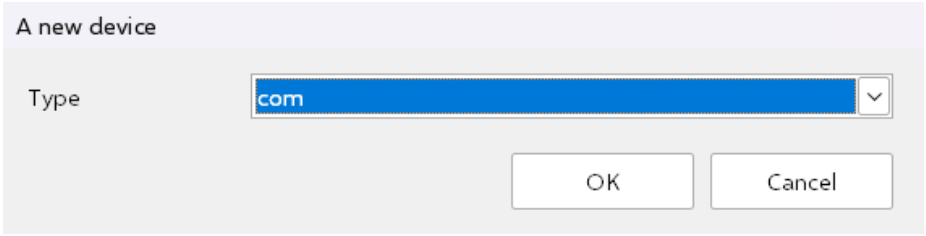


図 4-7　「Add a device」：型の選択

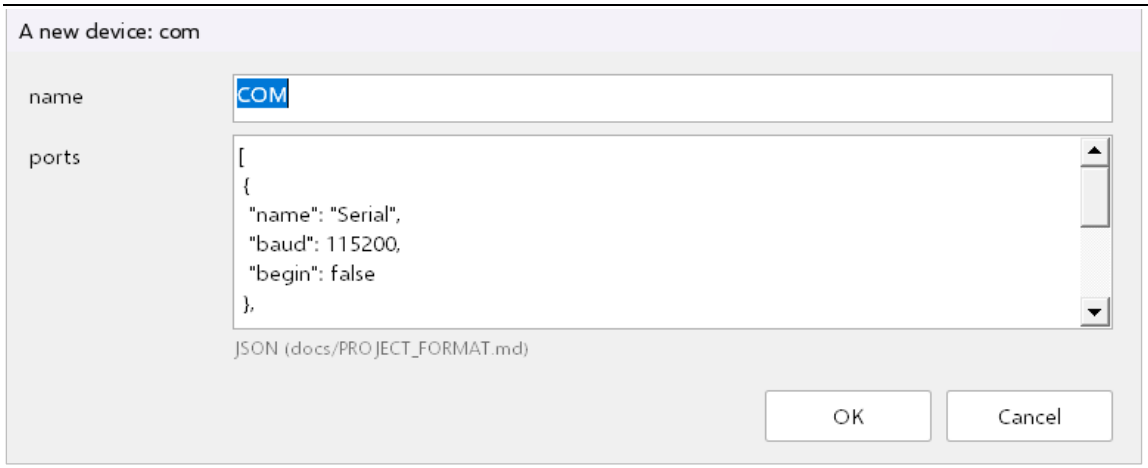


図 4-8　com デバイスの設定フォーム：名前と ports（JSON）

型	説明	追加時の既定の内容
localio	QEC ローカル DI／DO 点数（QEC-M-02：16／16）	名前 LocalIO、di 16、do 16
eva	86EVA 仮想デバイス（LDmicro 1.x の EtherCAT 方式）	enable、io 16、analog 4、decimal 0、servo 0、serial 0
com	シリアルポート：順に COM0、COM1…（UART 命令用）	Serial 115200（begin しない）、Serial1 9600
modbus	Modbus RTU／TCP メインデバイス：バスとサブデバイスの一覧	バス 0：RTU、COM1、有効。サブデバイスの一覧は空
ethernet	ネットワーク設定（static 固定 IP／dhcp）	static、192.168.1.100／255.255.255.0、ゲートウェイと DNS は 192.168.1.1
modbus-server	Modbus TCP サブデバイス：変数を SCADA／HMI に公開して読み書きさせます	on、port 502、unit 1、vars は空
ethercat	EtherCAT メインデバイス：通常は ENI を読み込んで作成します。サブデバイスと PDO は ENI から取得します（第 11 章）	eni QEC-ENI.xml、port 0、cycleUs 1000。ほかに mode（auto／sync／freerun）と master（plc／sketch）があります
motion86	Motion86 マシンと軸：「A new Motion86 machine」を開きます（第 12 章）	—

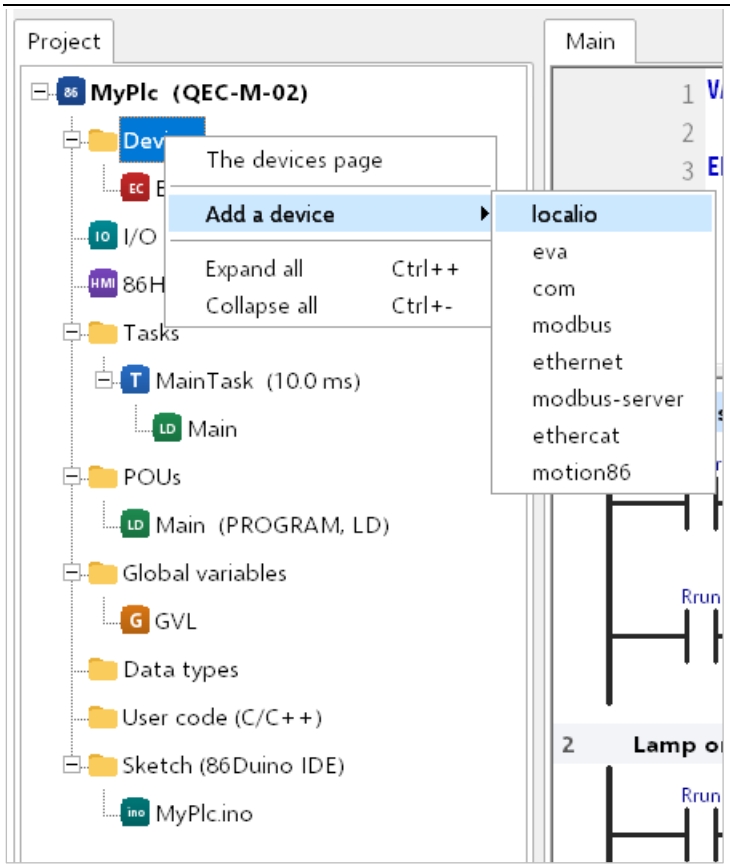


図 4-9　プロジェクトツリーの「Devices」の右クリックメニュー：Add a device ▸ に 8 種類の型が表示されます

ヒント：新規プロジェクトに localio デバイスがなくても、QEC-M-02 のローカル I/O のチャンネル `LocalIO.X00`〜`X17`、`LocalIO.Y00`〜`Y17`（既定は 16／16 点）を使用できます。

EtherCAT デバイスの mode：auto＝CiA 402 サーボがある場合は DC 同期、それ以外は free run。sync＝DC 同期（失敗時は free run）。master：plc＝86LDmicro がメインデバイスを起動（plc86_ecat.cpp）。sketch＝ユーザーの .ino が自分で begin、attach、start を行い、完了後に `plc86_ecOk = true` を設定します。

4.5　グローバル変数とデータ型

86LDmicro では、すべての変数は宣言してから使用する必要があります。宣言されていない名前はラダー図で赤い背景で表示され、チェック時にエラーとして表示されます。**グローバル変数リスト（GVL）**で宣言した変数は、すべての POU、86HMI、C/C++ から参照できます。POU の VAR で宣言した変数は、その POU でのみ使用できます。

4.5.1　グローバル変数リスト（GVL）

プロジェクトツリーで「Global variables > GVL」をダブルクリックするとエディターが開きます。1 行に 1 つの変数を宣言し、Ctrl+S で保存します。エディターには構文の色分けと補完があり、// の後ろまたは (* *) の中はコメントです（補完リストと Properties に表示されます）。

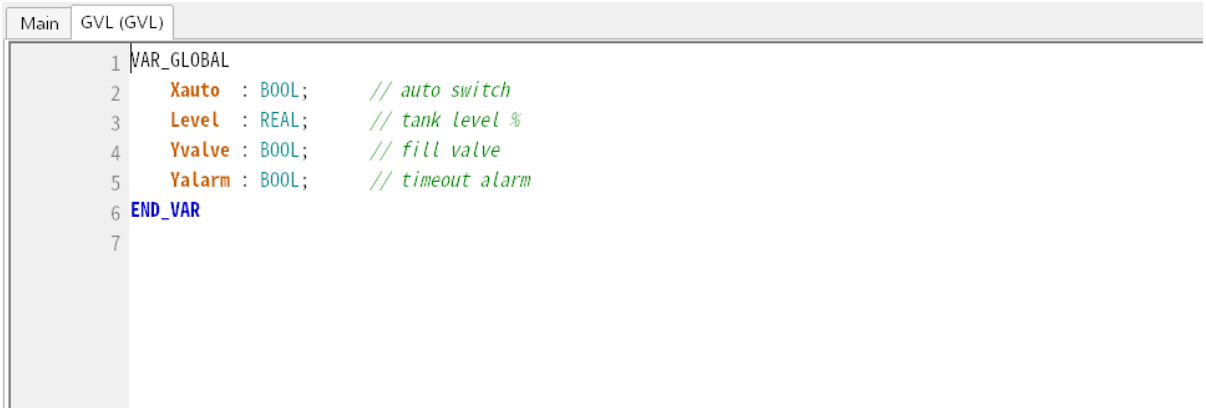


図 4-10　GVL エディター：1 行に 1 つの変数、// の後ろはコメント

```
VAR_GLOBAL
    Xstart   : BOOL;                    // コメントは補完リストと Properties に表示される
    Speed    : REAL := 12.5;            // 初期値
    Recipe   : ARRAY[1..5] OF INT;      // 配列（インデックス 1〜5）
    Axis1    : Axis_t;                  // ユーザー定義の構造体（データ型、下記参照）
    Mode     : Mode_t;                  // ユーザー定義の列挙型
END_VAR
VAR_GLOBAL RETAIN                       // 停電保持：EEPROM に保存され、再通電後に復元
    TotalRuns : DINT;
END_VAR
VAR_GLOBAL CONSTANT                     // 定数：プログラムでは変更できない
    MAX_SPEED : REAL := 100.0;
END_VAR
```

セクション	説明
VAR_GLOBAL	通常のグローバル変数。:= 値 で初期値を設定します（省略時は 0／FALSE／空文字列）
VAR_GLOBAL RETAIN	停電保持：値はコントローラーの EEPROM に保存され、再通電後に復元されます。ウォームリスタートでは保持され、コールドリスタートではクリアされます（3.2.5 節）

セクション	説明
VAR_GLOBAL CONSTANT	定数：:= 値 で値を指定します。プログラムで書き込むとチェック時にエラーになります

*注意：RETAIN は代入のたびにすぐ書き込まれるわけではありません。コントローラーは約 1 秒ごとにチェックし、変化があったときだけ新しいスナップショットを（スキャンの合間に）分割して EEPROM に書き込み、すべて書き終えた時点で有効になります。新しいスナップショットを書き終える前に電源が切れた場合、再通電時には前回書き終えた値に復元されます。EEPROM には書き込み回数の上限があります。常に変化する変数（カウンター、タイマー値、アナログ値）を RETAIN にしないでください。保存が必要な場合は、必要なとき（停止時、レシピ変更時）にだけ RETAIN 変数へコピーします。

用途に応じて複数の GVL に分けることができます（「Project > Add a global variable list...」、例：IO、Recipe）。変数名はすべての GVL を通じて重複できません。

4.5.2 POU のローカル変数

POU の宣言領域では、VAR（ローカル変数。ファンクションブロックのインスタンスは通常ここに置きます。例：tLamp : TON;）と VAR_TEMP（一時変数：呼び出しのたびに初期値から始まり、保持されません）を使います。FUNCTION と FUNCTION_BLOCK にはさらにパラメーター VAR_INPUT、VAR_OUTPUT、VAR_IN_OUT があります（PROGRAM にはパラメーターがありません）。サポート範囲の詳細は 8.2.1 節を参照してください。他の POU や Watch ウィンドウからは、プログラム名.変数名 の形でローカル変数を読み出します（例：Main.tLamp.ET）。

*ヒント：FUNCTION_BLOCK と FUNCTION は、基本型のグローバル変数と定数（例：MAX_SPEED）を読み出すことができます。配列型や構造体型のグローバル変数はその中で直接アクセスできないため、VAR_INPUT で渡してください（渡されるのはコピーです）。配列や構造体を変更させる場合は、呼び出し側 ST プログラムのローカル変数を VAR_IN_OUT で渡し、呼び出し後にグローバル変数へコピーし戻します（8.2.1 節）。

4.5.3 基本データ型

型	範囲／説明	型	範囲／説明
BOOL	TRUE／FALSE	REAL	32 ビット浮動小数点
SINT／USINT	8 ビット符号付き／符号なし	LREAL	64 ビット浮動小数点
INT／UINT	16 ビット符号付き／符号なし	TIME	時間（例：`T#1s500ms`、`T#100ms`
DINT／UDINT	32 ビット符号付き／符号なし	STRING	文字列（例：`'hello'`
LINT／ULINT	64 ビット符号付き／符号なし	BYTE／WORD／DWORD／LWORD	ビット列（8／16／32／64 ビット）

4.5.4 データ型（構造体、列挙型、配列）

「Project > Add a data type...」で `.dut` ファイルを作成し、`TYPE ... END_TYPE` の中で構造体、列挙型、名前付き配列を定義すると、GVL や POU で型として使用できます。

```
TYPE
    Axis_t : STRUCT
        pos   : REAL;
        speed : REAL;
        ready : BOOL;
    END_STRUCT;
    Mode_t : (Idle, Auto, Manual);      // 列挙型
    Recipe_t : ARRAY[1..5] OF INT;      // 名前付き配列
END_TYPE
```

プログラムでは `Axis1.speed`、`Recipe[2]`、`Mode := Auto;` のようにアクセスします。データ型を追加したときの既定の内容は `x : INT;` だけを持つ構造体で、そのまま書き換えられます。

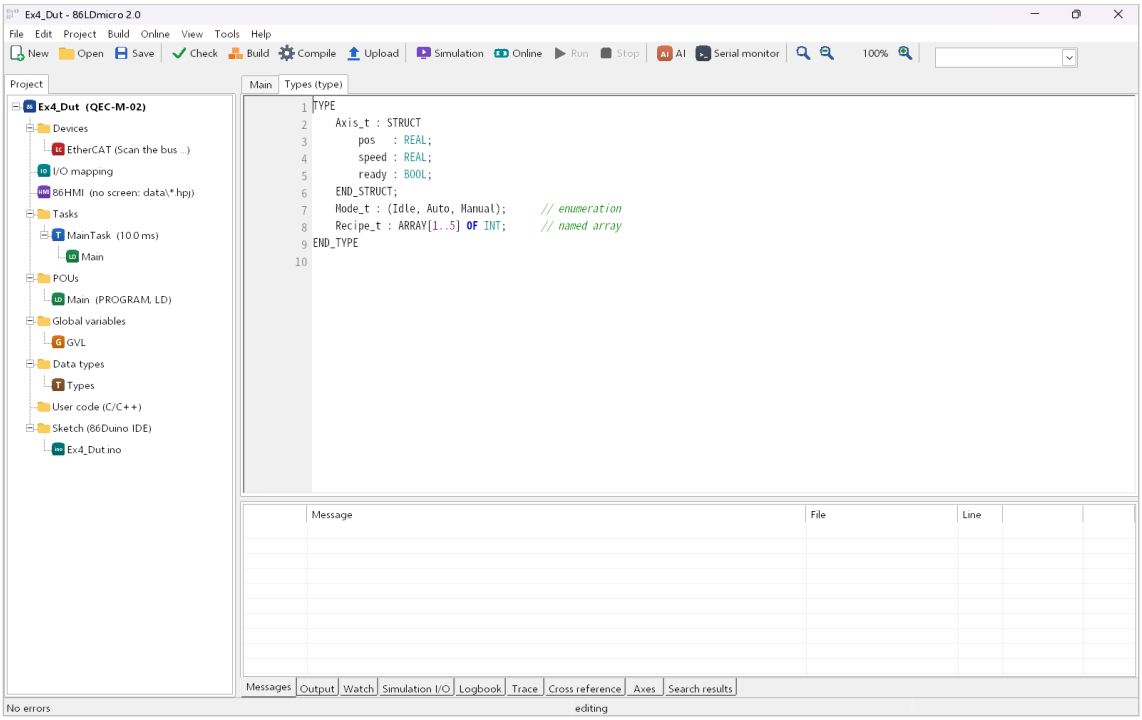


図 4-11　データ型エディター：構造体、列挙型、名前付き配列

4.5.5 変数の追加方法

方法	操作	変数の置き場所
1. GVL を直接編集	プロジェクトツリーで「Global variables > GVL」をダブルクリックし、名前 : 型; を入力	グローバル
2. POU の宣言領域	LD／ST／SFC タブ上部の `VAR ... END_VAR` に入力	ローカル（その POU でのみ使用可能）
3. I/O マッピング	I/O マッピングの追加時に「Variable」欄に新しい名前を入力（4.6 節）	最初の GVL に自動的に追加
4. Common circuits	Common circuits を挿入するときのフォーム内の新しい名前（7.1 節）	入力／出力／内部ビットは GVL、タイマーなどはその POU の VAR に置かれます
5. 86HMI バインド	ウィジェットをバインドするときに「Variable」欄に新しい名前を入力（13.2 節）	GVL に自動的に追加
6. Excel からのインポート	I/O マッピングページの「Import from Excel...」（4.6 節）	指定した GVL（RETAIN は RETAIN セクションへ）
7. AI アシスタントによるコード挿入	AI が回答したコードで「Insert the code」をクリック（15.3 節）	VAR ブロック内の新しい変数が自動的に宣言されます

*ヒント：命名規則：`X` で始まる名前は入力、`Y` は出力、`R` は内部ビット、`T` はタイマー、`C` はカウンター、`M` は数値です（LDmicro 1.1 と同じ）。I/O をマッピングするときに一目で区別できます。名前に使えるのは英字、数字、_ のみで、英字で始める必要があります。大文字と小文字は区別されず、IEC の予約語（付録 C）は使用できません。

4.6 I/O マッピング

PLC プログラムは変数を読み書きするだけです。変数と実際の点（ローカル I/O、EtherCAT PDO、86EVA、Arduino ピン、Modbus）との対応は「I/O mapping」ページで設定するため、プログラムは配線を意識する必要がありません。プロジェクトツリーで「I/O mapping」をダブルクリックする（または「Project > I/O mapping」を選ぶ）と開きます。

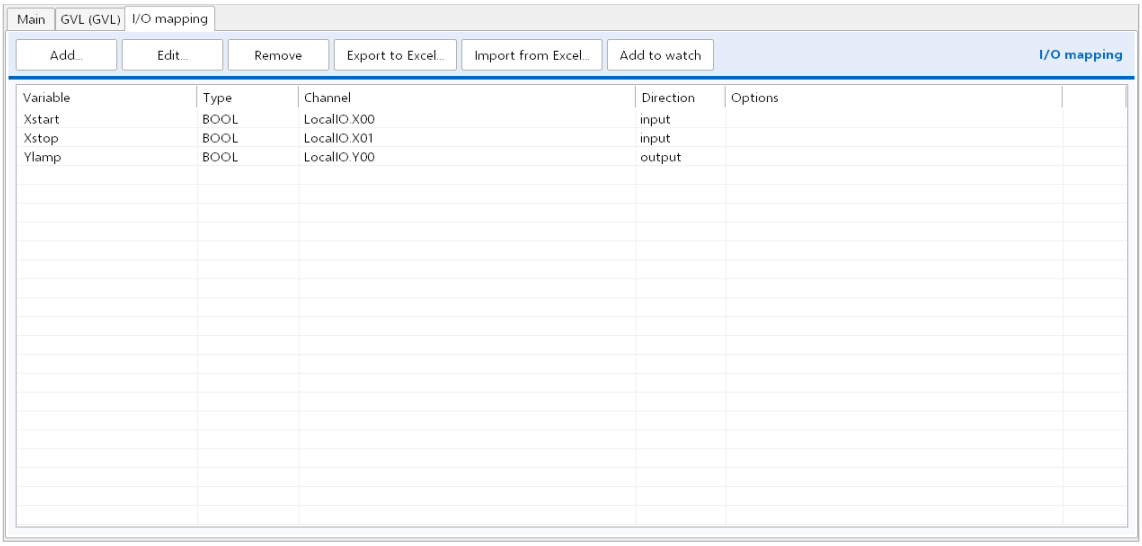


図 4-12　I/O マッピングページ：変数、型、チャンネル、方向、オプション。上部に 6 つのボタン

ボタン	機能
Add...	マッピングを追加します（「A new link」ウィンドウ）
Edit...	選択したマッピングを変更します。行をダブルクリックしても同じです
Remove	選択したマッピングを削除します（変数自体は削除されません）
Export to Excel...	変数とマッピングを .xlsx または .csv に保存します
Import from Excel...	.xlsx または .csv から変数を宣言し、マッピングを設定します
Add to watch	選択した行（Ctrl／Shift で複数選択可）の変数を「Watch」ウィンドウに追加します

4.6.1 I/O マッピングの追加

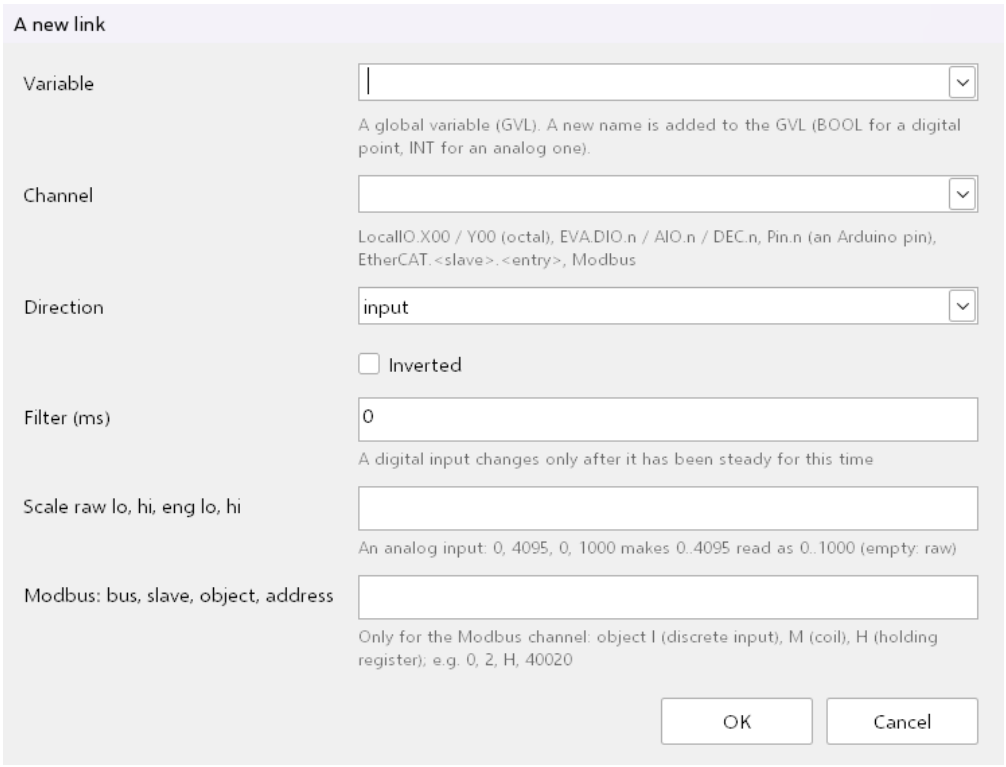


図 4-13 「A new link」：変数、チャンネル、方向、反転、フィルター、スケーリング、Modbus

項目	説明
Variable	グローバル変数（ドロップダウンに GVL の変数が表示されます）。新しい名前を入力すると最初の GVL に自動的に追加されます。デジタル点は BOOL、アナログ点は INT、EtherCAT エントリーはその PDO の型になり、チャンネル名がコメントになります
Channel	`LocalIO.X00`〜`X17`／`Y00`〜`Y17`（ローカル、8 進数）、`EVA.DIO.n`／`AIO.n`／`DEC.n`、`Pin.n`（Arduino ピン）、`EtherCAT.<`サブデバイス`>.<`エントリー`>`、`Modbus`。ドロップダウンに使用可能なチャンネルが表示されます
Direction	入力／出力。LocalIO と EtherCAT の方向はチャンネルで決まり、その他はこの項目に従います
Invert	デジタル点を反転します（例：b 接点で配線したボタン）
Filter (ms)	デジタル入力は、この時間安定して保持された後に変化します（チャタリング除去）
Scale raw lo, hi, eng lo, hi	アナログ入力用。例：`0, 4095, 0, 1000` は 0..4095 を 0..1000 に換算します。空欄の場合は生の値です
Modbus: bus, slave, object, address	Modbus チャンネルのみ：オブジェクト I（ディスクリート入力）、M（コイル）、H（保持レジスター）。例：`0, 2, H, 40020`（先に modbus デバイスを追加する必要があります。4.4 節）

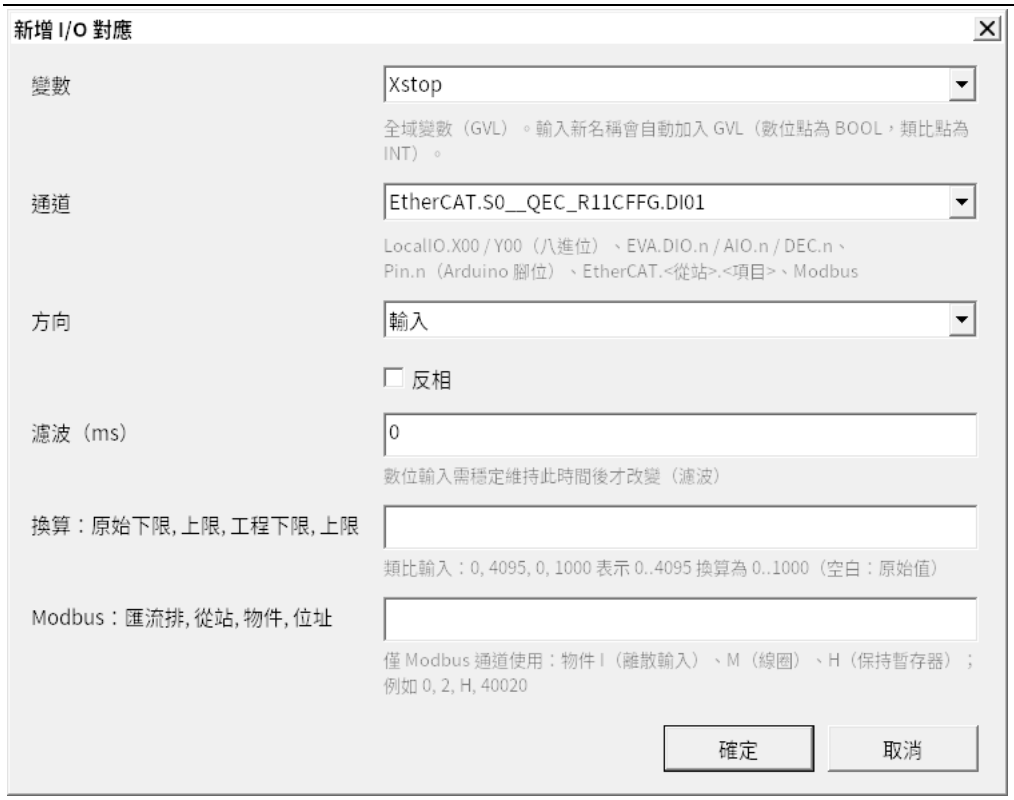


図 4-14　EtherCAT チャンネルのマッピング：Xstop → EtherCAT.S0__QEC_R11CFFG.DI01

EtherCAT のチャンネルは ENI を読み込んだ後に表示されます（第 11 章）。プロジェクトツリーの PDO エントリーを右クリックして「Link to a variable...」を選ぶこともできます。

4.6.2　Watch への追加

I/O マッピングページで 1 行または複数行を選択し（Ctrl／Shift で複数選択）、上部の **Add to watch** をクリックするか右クリックで **Add to watch** を選ぶと、それらの変数が「Watch」ウィンドウに追加され、ステータスバーに「N variables added to the watch window」と表示されます。シミュレーション中やオンライン中は、そのまま値を確認したり強制したりできます。右クリックメニューには **Add...**、**Edit...**、**Remove** もあります。

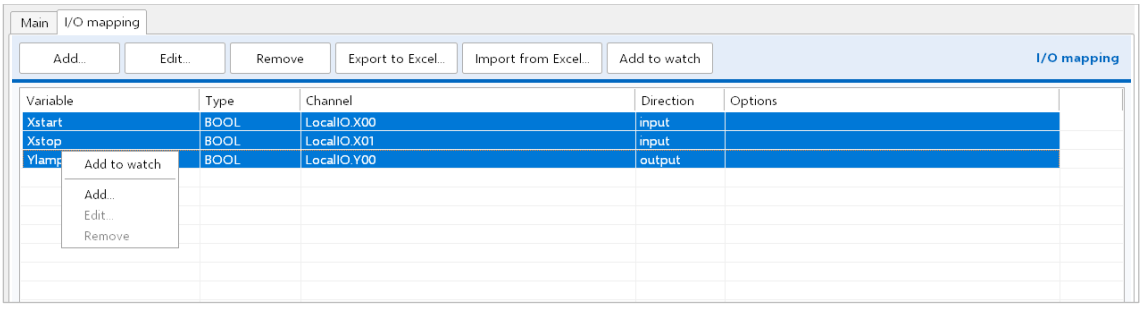


図 4-15　I/O マッピングページ：3 行を選択して右クリック >「Add to watch」

4.6.3　I/O 表の Excel インポート／エクスポート

I/O が多い場合は、Excel で計画してから一括でインポートできます。

1. I/O マッピングページで **Export to Excel...**（または「Project > Export the I/O table to Excel...」）をクリックし、.xlsx（.csv UTF-8 も可）で保存します。保存後、自動的に Excel で開きます。
2. 「IO」ワークシートには変数ごとに 1 行があり、「Channels」ワークシートには使用可能なすべてのチャンネルと現在マッピングされている変数が表示されます。
3. Excel で計画または変更し（列は下表参照）、保存します。
4. **Import from Excel...**（または「Project > Import the I/O table from Excel...」）をクリックしてファイルを選択します。見出し行は中国語でも英語でもよく、列の順序も自由です。「N rows: declare the new variables in the GVL and set their I/O mapping (the links of these variables are replaced)?」を確認して **Yes** をクリックします。
5. 完了すると結果の概要（追加・変更された変数とスキップされた行）が表示されます。元に戻すには、保存せずにプロジェクトを開き直します。

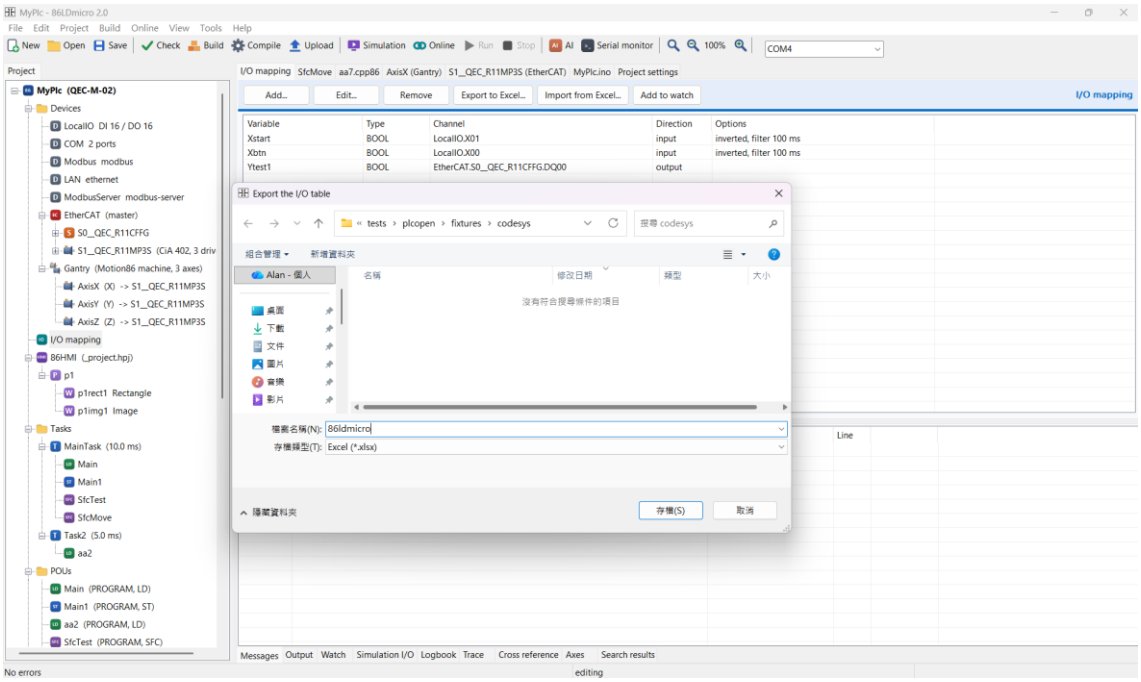


図 4-16　エクスポートした I/O 表（Excel）：「IO」ワークシート

列	説明
変数	変数名（必須）
型	BOOL、INT、REAL…。空欄の場合、新しい変数は BOOL になり、既存の変数は変更されません
初期値、コメント	宣言に書き込まれます
チャンネル	「A new link」と同じ。空欄はこの変数のマッピングを削除することを意味します
方向、反転、フィルター ms、スケーリング（生の下限, 生の上限, 工学値の下限, 工学値の上限）	「A new link」と同じ
停電保持（RETAIN）	新しい変数を GVL の RETAIN セクションに置きます
GVL	新しい変数を置く GVL（空欄：最初の GVL）
その他（JSON）	その他のマッピング属性（例：Modbus の bus、slave、object、address）

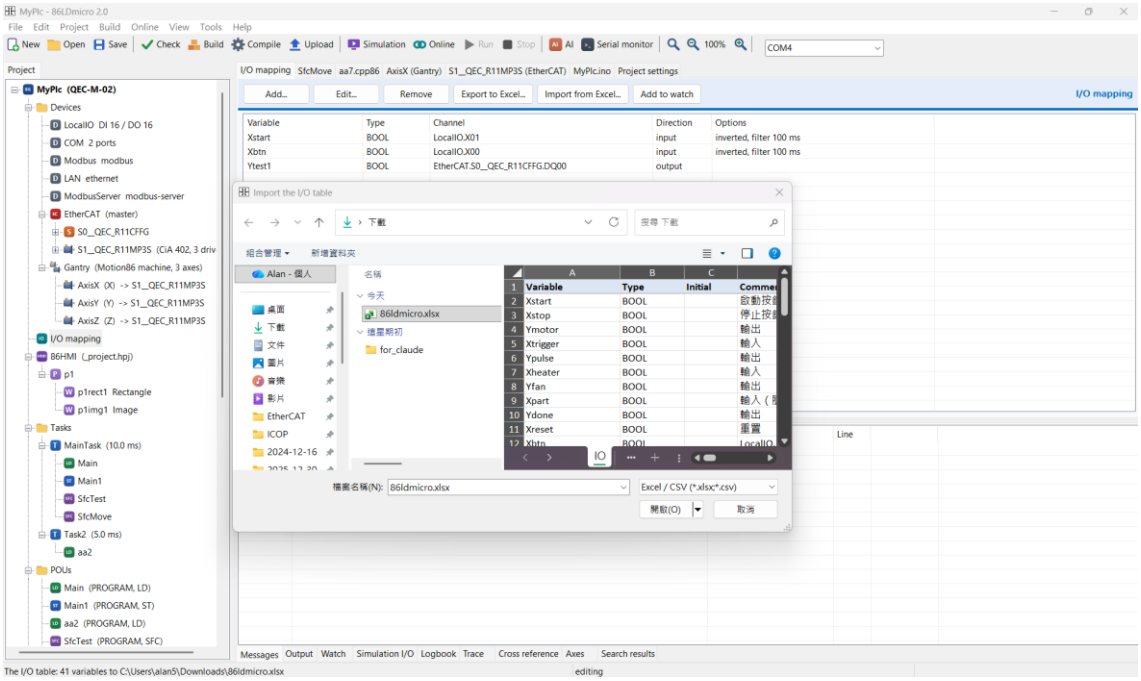


図 4-17　Excel からインポートする前の確認

*ヒント：インポートは編集中にのみ使用できます。シミュレーション中やオンライン中は「Stop the simulation / go offline to edit」と表示されます。

表内に同じ変数が 2 回出現した場合は最初の行だけが使われ、名前が不正な行はスキップされます。いずれも結果の概要に表示されます。コマンドライン：`86ldmicro-cli iotable export|import <プロジェクト> <ファイル>`（15.5 節）。

4.7 POU とユーザーコードの追加

「**Project**」メニュー（またはプロジェクトツリーの該当ノードの右クリックメニュー）から追加します。名前を入力して **OK** をクリックすると、新しいファイルがすぐに開きます。名前に使えるのは英字、数字、_ のみで、英字で始める必要があり、プロジェクト内の他の POU、GVL、データ型、ユーザーコードと同じ名前は使用できません。

コマンド	作成されるファイル	既定の名前	説明
Add a PROGRAM（ladder／ST／SFC）...	`pou\<名前>.lad／.st／.sfc`	Program	新しい PROGRAM は最初のタスクに自動的に追加されます
Add a FUNCTION_BLOCK（ladder／ST）...	`pou\<名前>.lad／.st`	MyFb	内部状態を持つ再利用可能なファンクションブロック（8.3 節）
Add a FUNCTION (ST)...	`pou\<名前>.st`	MyFunction	メモリーを持たないファンクション（8.3 節）
Add a global variable list...	`gvl\<名前>.gvl`	GVL2	空の `VAR_GLOBAL ... END_VAR`
Add a data type...	`dut\<名前>.dut`	MyType	構造体のテンプレート（4.5 節）
Add a user code file (C/C++)...	`usercode\<名前>.cpp86`	MyCode	説明とフック関数のテンプレート付き（第 14 章）

***重要：**新しい PROGRAM の既定の名前 `Program` は IEC の予約語 PROGRAM と同じです（名前は大文字と小文字を区別しません）。そのまま OK をクリックすると、チェック時にエラーになります。`Filling`、`Conveyor` など意味のある名前に変更してください。

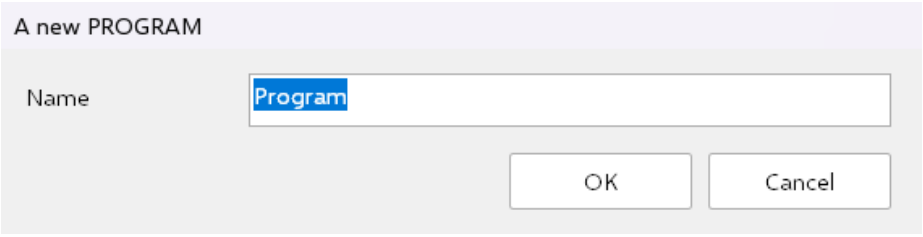


図 4-18 「A new PROGRAM」：名前の入力

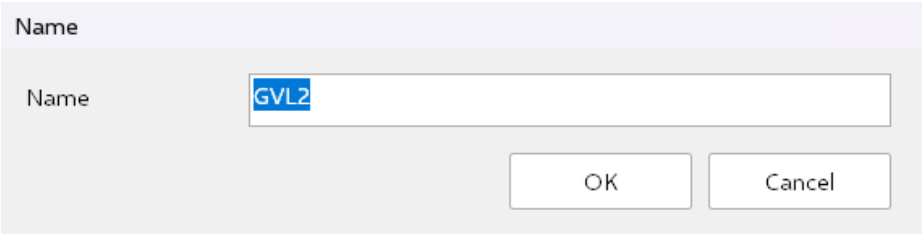


図 4-19 「A new global variable list」：名前の入力（既定は GVL2）

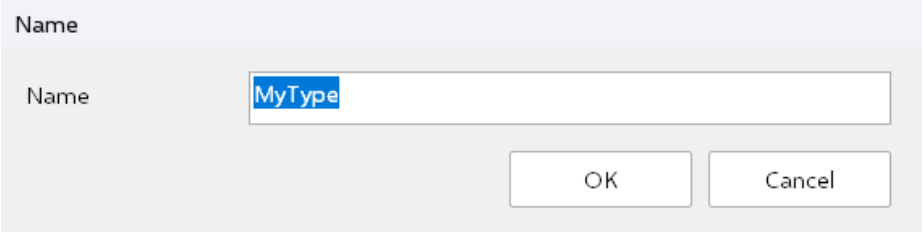


図 4-20　「A new data type」：名前の入力

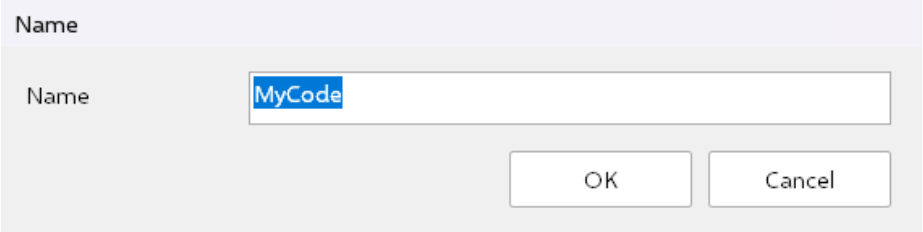


図 4-21　「A new user code file (C/C++)」：名前の入力

新しいファイルの内容はそのまま編集できるテンプレートです。たとえば ST の PROGRAM と FUNCTION_BLOCK は次のとおりです。

```
PROGRAM Filling
VAR
END_VAR

END_PROGRAM

FUNCTION_BLOCK MyFb
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR
END_VAR

END_FUNCTION_BLOCK
```

FUNCTION のテンプレートは `FUNCTION MyFunction : INT`（INT を返し、最終行は `MyFunction := 0;`）です。

SFC のテンプレートは、初期ステップ `Init` と、1 秒後に戻るステップ `Step1` です（8.4 節）。

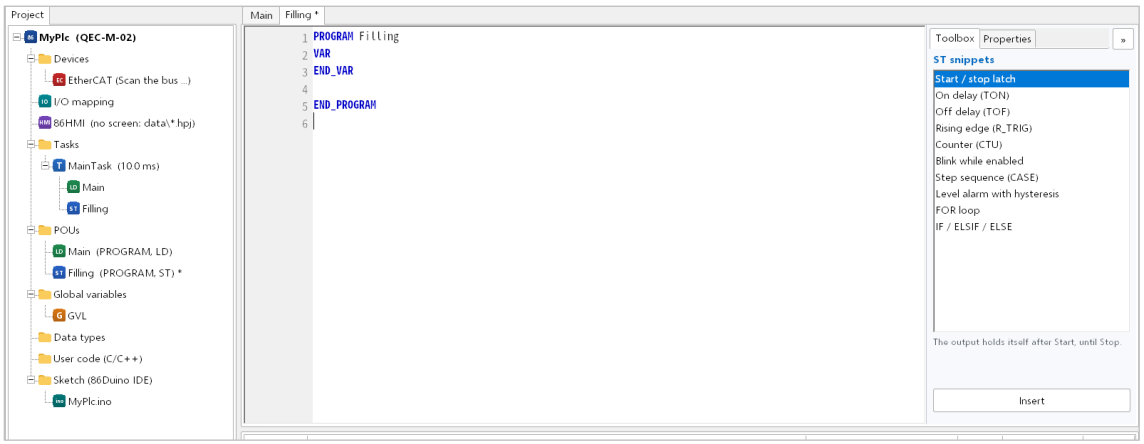


図 4-22　新しい ST PROGRAM Filling：MainTask に追加され、開いた状態

4.7.1 ユーザーコード（C/C++）

ユーザーコードファイル（.cpp86）は PLC と一緒に plc86.cpp にコンパイルされます。新しいファイルの先頭には説明があります：PLC 変数は GVL(名前)（グローバル変数）、ST(プログラム.変数)（プログラムの変数）、FUN(名前)(引数)（プロジェクトの ST FUNCTION）と書きます。また、フック関数 plc86_user_setup、plc86_user_before、plc86_user_after、plc86_user_loop などのテンプレートが付いています。詳細は第 14 章を参照してください。

Main | SendMsg.cpp86

```
1  // SendMsg: when Nmsg changes, send one line on Serial1
2
3  void plc86_user_setup(void) {
4      Serial1.begin(9600);              // COM1, 9600 bps
5  }
6
7  // every pass of loop(): non-blocking work only (no delay())
8  void plc86_user_loop(void) {
9      static int32_t sent = 0;          // number already sent
10     int32_t n = GVL(Nmsg);
11     if(n != sent) {
12         sent = n;
13         Serial1.print("Button pressed, count = ");
14         Serial1.println(n);
15     }
16 }
17
```

図 4-23　ユーザーコードエディター：フック関数と GVL() による PLC 変数へのアクセス

- POU の名前変更：プロジェクトツリーで POU を右クリック >「Rename...」。タスク内のプログラム名も一緒に更新されます。
- 削除：右クリック >「Remove from the project」。ファイルは data\86ld に残ります。
- sketch 独自の .h／.cpp：プロジェクトツリーの「Sketch (86Duino IDE)」を右クリック >「A new .h / .cpp in the sketch...」（14.5 節）。

第 5 章　クイックスタート：白紙からアップロードまで

以下の 12 のステップで、最小限ですが完全な流れを示します：86Duino IDE で sketch を作成し、86LDmicro を開いてプロジェクトを作成し、変数を宣言し、「起動／停止の自己保持＋2 秒遅れで表示灯を点灯」のラダー図を描き、QEC-M-02 のローカル I/O にマッピングし、まず PC 上でシミュレーションしてから、ビルドしてコントローラーにアップロードし、オンラインでモニターします。

完成後の動作：起動ボタン（X00）を 1 回押すと、内部ビット `Rrun` が TRUE になって自己保持します。`Rrun` が 2 秒間続くと表示灯 `Ylamp`（Y00）が点灯します。停止ボタン（X01）を押すと、`Rrun` と表示灯の両方が消えます。

変数	型	I/O マッピング	用途
`Xstart`	BOOL	`LocalIO.X00`（入力）	起動ボタン（a 接点）
`Xstop`	BOOL	`LocalIO.X01`（入力）	停止ボタン（a 接点で配線し、プログラムでは b 接点を使用）
`Rrun`	BOOL	—（内部ビット）	運転中（自己保持）
`Ylamp`	BOOL	`LocalIO.Y00`（出力）	運転表示灯
`tLamp`	TON（IEC ファンクションブロック）	—	表示灯の 2 秒遅延（Main の VAR で宣言）

*__重要：__ラダー図を描いたら、必ず「I/O mapping」で `Xstart`、`Xstop`、`Ylamp` を実際の `LocalIO.X00`、`X01`、`Y00` にマッピングしてください（ステップ 9）。マッピングされていない変数はメモリー上のビットにすぎません。チェックは通りますが（入力がマッピングされていない場合、ビルドでは警告が出るだけです）、アップロード後に実際の I/O は動作しません。

*__ヒント：__始める前に第 2 章を済ませておいてください：86Duino Coding 502 と 86LDmicro がインストールされ、QEC-M-02 が USB ケーブルで PC に接続されており、その COM ポートがわかっていること（2.4 節）。

5.1　ステップ 1：86Duino IDE を開き、sketch を作成し、ボードとシリアルポートを選択する

86Duino Coding 502 を開きます。「**File > New**」（Ctrl+N）で新しい sketch を作成し、「**File > Save As...**」（Ctrl+Shift+S）で `MyPlc` として保存します。sketch の名前が、後で作る 86LDmicro プロジェクトの名前になります。

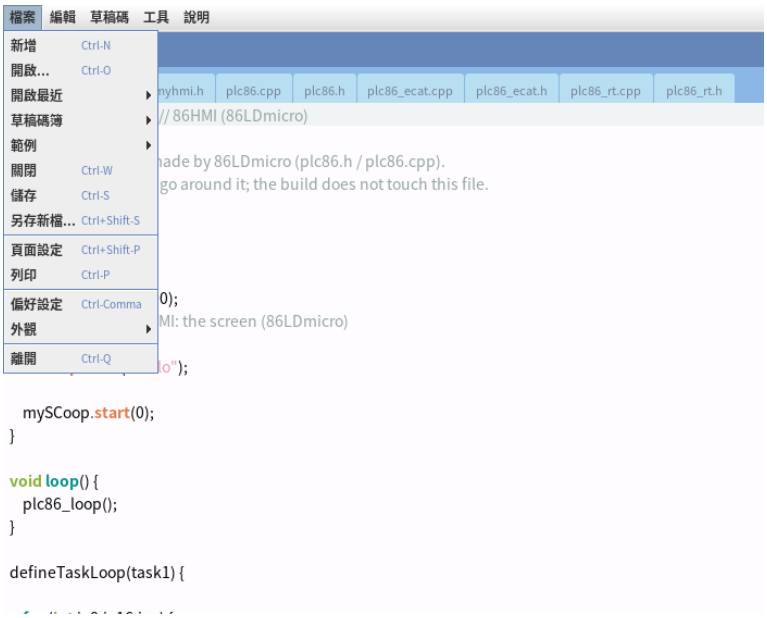


図 5-1　86Duino IDE の「File」メニュー：New、Save As

「**Tools > Board**」で **QEC M02** を選び、「**Tools > Serial Port**」でコントローラーの COM ポートを選びます。IDE ウィンドウの右下に「QEC M02 on COMx」と表示されます。

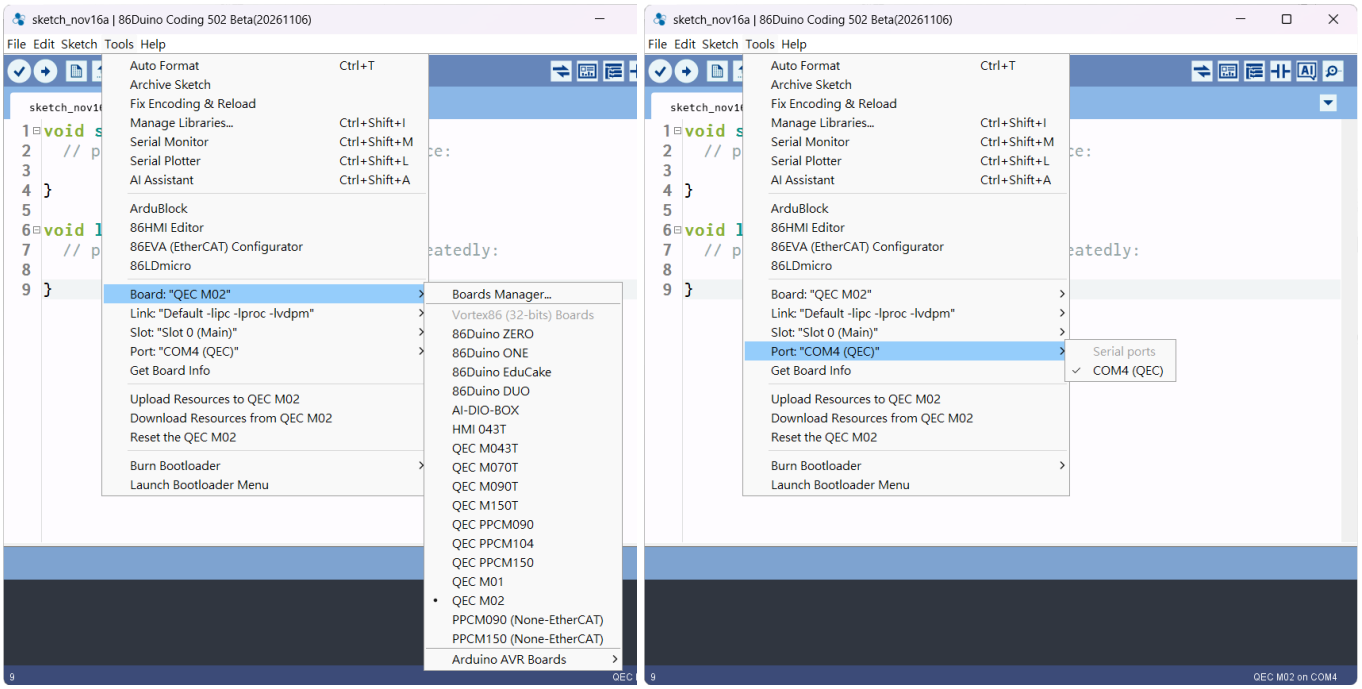


図 5-2 & 図 5-3　「Tools > Board」で QEC M02 を選択、「Tools > Serial Port」でコントローラーの COM ポートを選択

5.2　ステップ 2：IDE から 86LDmicro を開き、新しいプロジェクトを作成する

「**Tools > 86LDmicro**」を選ぶか、IDE ツールバー右側の **86LDmicro** ボタンをクリックします。この sketch にはまだ 86LDmicro プロジェクトがないため確認ウィンドウが表示されます。**Yes** をクリックして新しいプロジェクトを作成します。

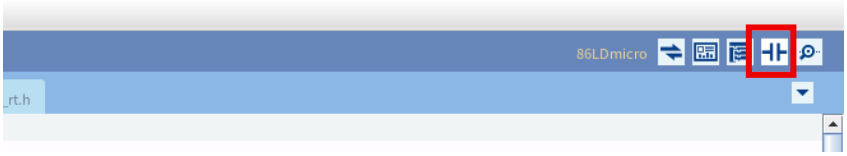


図 5-4　IDE ツールバー右側の 86LDmicro ボタン

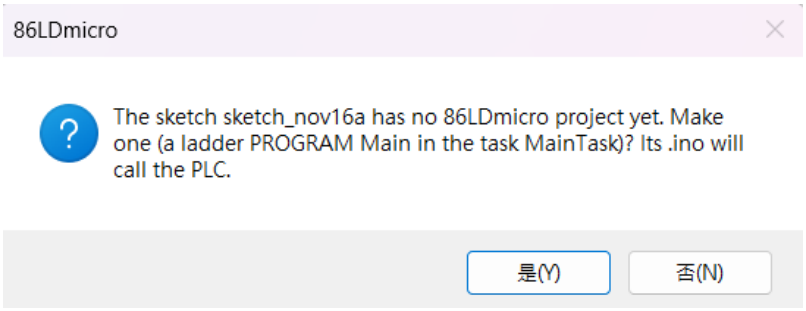


図 5-5　sketch にまだ 86LDmicro プロジェクトがないときの確認

- IDE から開いた場合、86LDmicro はその IDE のコンパイラーと IDE で選択したシリアルポートを自動的に使用するため、別途設定する必要はありません。
- `.ino` には `#include "plc86.h"`、`plc86_setup()`、`plc86_loop()` が自動的に追加され、その他のコードは変更されません。
- 86LDmicro を単独で起動することもできます。「**File > New project...**」（Ctrl+N）を選び、名前 `MyPlc` と場所（既定は ドキュメント`\86Duino`）を入力して **OK** をクリックします（2.5 節）。

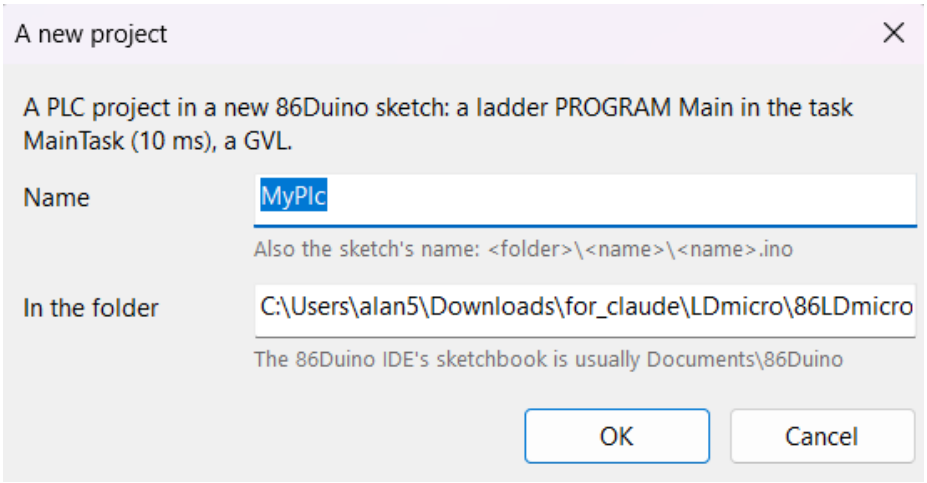


図 5-6　単独で起動したときの「A new project」：名前は sketch の名前にもなります

5.3　ステップ 3：新しいプロジェクトを確認する：プロジェクトツリーとラダー図 Main

新しいプロジェクトには次の項目がすでに作成されており、ラダー図 `Main` が自動的に開きます。

プロジェクトツリーのノード	内容
Devices	コントローラーのデバイス（ローカル I/O、EtherCAT など。4.4 節）
I/O mapping	変数と実際のチャンネルの対応表（ステップ 9）
Tasks > MainTask（10.0 ms）	10 ms ごとに実行され、`Main` を実行します
POUs > Main（PROGRAM, LD）	ラダープログラム。現時点では空のネットワーク 1 が 1 つあるだけです
Global variables > GVL	グローバル変数リスト（ステップ 4）
Sketch (86Duino IDE)	sketch の `.ino`

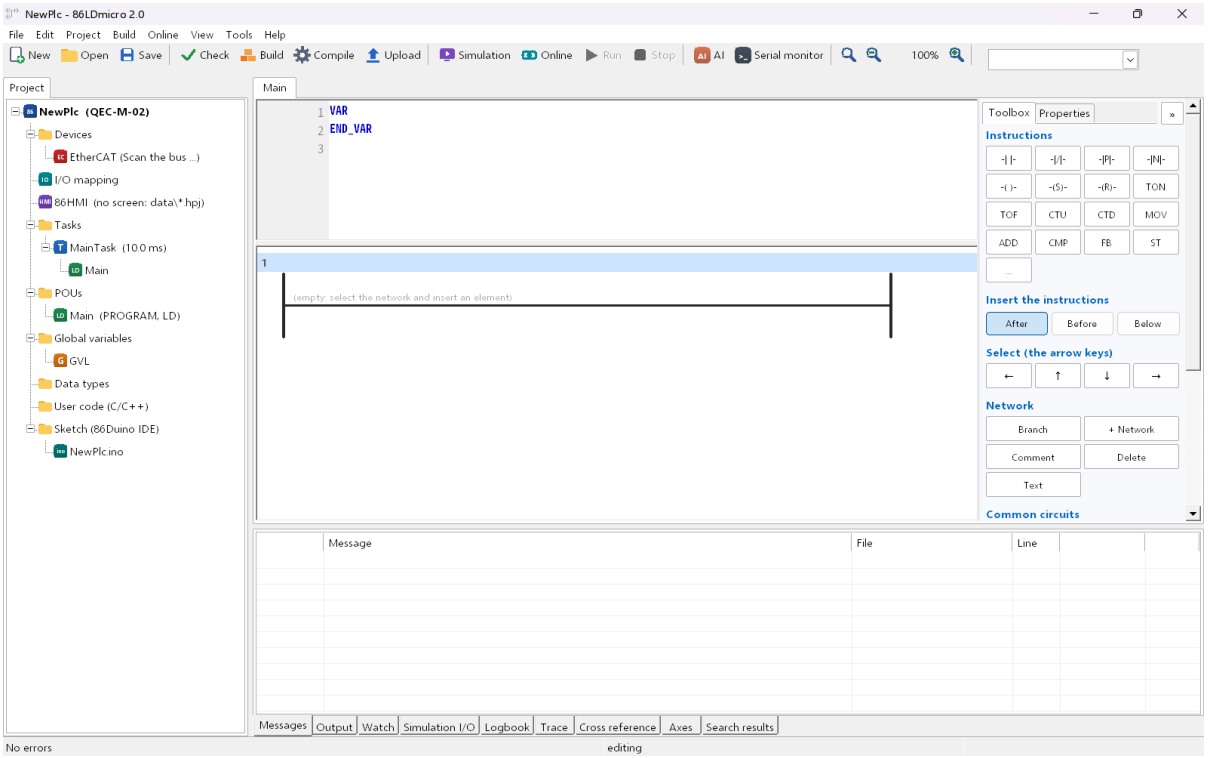


図 5-7　新しいプロジェクト：Devices、I/O mapping、Tasks、POUs（Main）、GVL が作成済み

ラダー図タブは 3 つの部分に分かれています：上部は**宣言領域**（`VAR ... END_VAR`。この POU 自身の変数とファンクションブロックのインスタンスを置きます）、中央は**ネットワーク**、サイドパネルは **Toolbox**（「Instructions」「Insert the instructions」「Select (the arrow keys)」「Network」「Common circuits」）です。各領域の詳細は 3.5 節を参照してください。

5.4　ステップ 4：GVL で変数を宣言する

86LDmicro 2.0 では、すべての変数を先に宣言して型を指定する必要があります。プロジェクトツリーで「Global variables > GVL」をダブルクリックし、GVL の内容全体（VAR_GLOBAL と END_VAR を含む）を次の内容に置き換えて、Ctrl+S で保存します。

```
VAR_GLOBAL
    Xstart : BOOL;      // 起動ボタン（a 接点）
    Xstop  : BOOL;      // 停止ボタン（a 接点で配線）
    Rrun   : BOOL;      // 運転中
    Ylamp  : BOOL;      // 運転表示灯
END_VAR
```

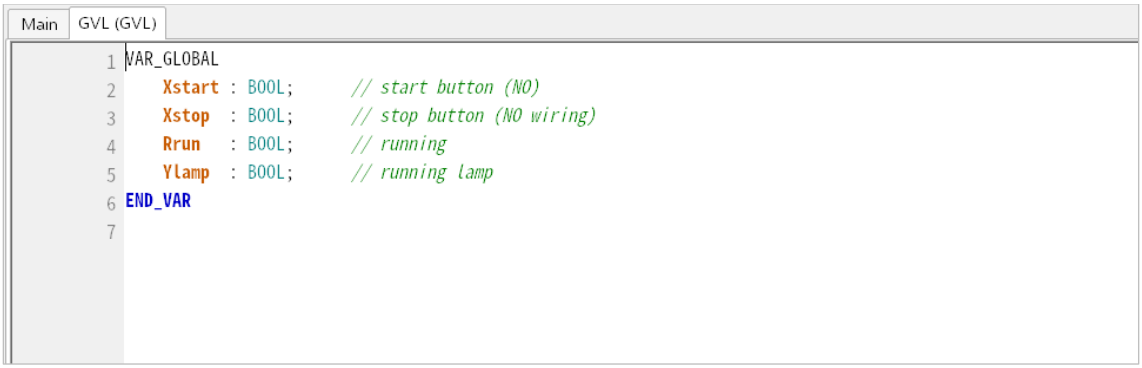


図 5-8　GVL：Xstart、Xstop、Rrun、Ylamp を宣言

*ヒント：命名規則：X で始まる名前は入力、Y は出力、R は内部ビットです（LDmicro 1.1 と同じ）。I/O をマッピングするときに一目で区別できます。名前はあくまで慣例で、型は宣言で決まります。
ラダー図でまだ宣言されていない名前を使うと、その要素は赤い背景でエラー表示されます（6.0 節）。宣言の構文とデータ型については 4.5 節を参照してください。

5.5　ステップ 5：a 接点 Xstart を挿入する

プロジェクトツリーで「**POUs > Main**」をダブルクリックしてラダー図に戻ります。サイドパネルの「**Insert the instructions**」で **After** が選択されていることを確認します。

1. ネットワーク 1（空のネットワーク）をクリックして選択します。
2. サイドパネルの「**Instructions**」にある **-| |-**（a 接点）をクリックするか、キーボードの **C** を押します。
3. 表示されたウィンドウで、「Variable」欄のドロップダウンに宣言済みのすべての変数が表示されます。`Xstart` を選び、**OK** をクリックします。

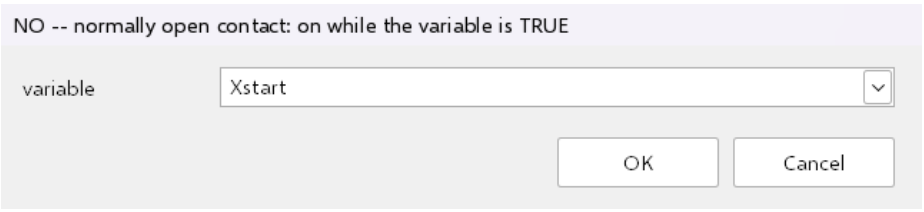


図 5-9　a 接点（NO）のパラメーターウィンドウ：変数の選択



図 5-10　ネットワーク 1 に a 接点 Xstart が表示される（青い背景は現在選択中であることを示します）

5.6　ステップ 6：b 接点 Xstop とコイル Rrun を直列に追加する

1. `Xstart` が選択されたまま、**-|/|-**（b 接点、キー **N**）をクリックし、変数に `Xstop` を選んで **OK** をクリックします。新しい要素は選択中の要素の後ろ、つまり**直列**に挿入されます。
2. **-( )-**（コイル、キー **L**）をクリックし、変数に `Rrun` を選んで **OK** をクリックします。コイルは自動的にネットワークの右側（右母線の横）に配置されます。



図 5-11　コイル（OUT）のパラメーターウィンドウ

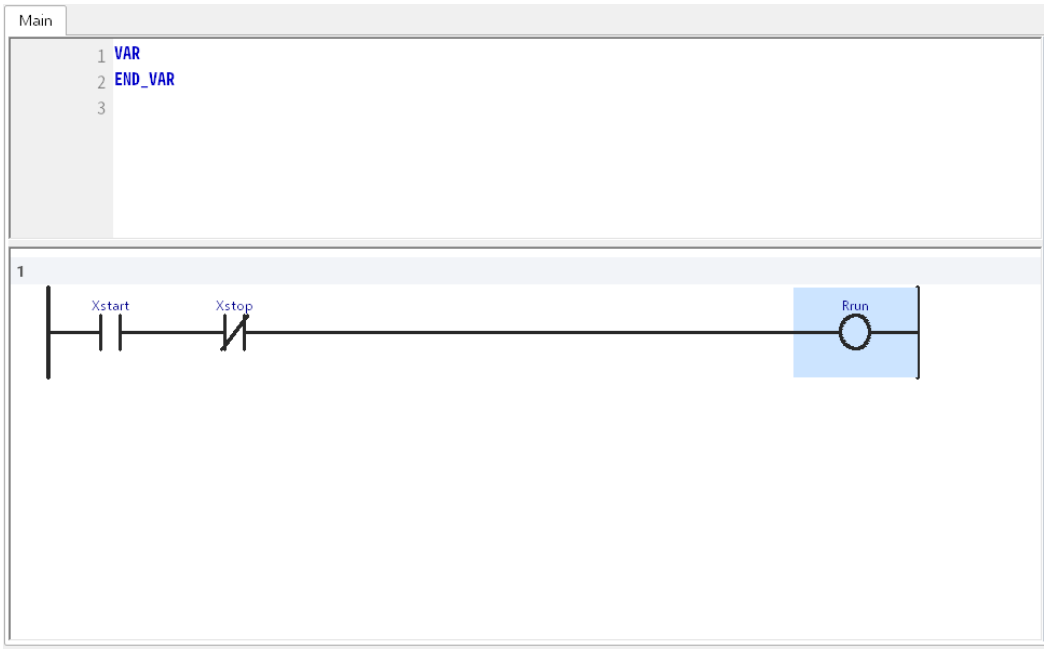


図 5-12　Xstart と Xstop を直列にしてコイル Rrun を駆動

*ヒント：停止ボタンを b 接点で配線する場合（断線時に停止を押したのと同じになるため、より安全）は、プログラムでは a 接点 `NO(Xstop)` を使うか、ステップ 9 の I/O マッピングで「Invert」にチェックを入れます。

5.7　ステップ 7：Rrun の接点を並列に追加して自己保持回路を作る

起動ボタンを離した後は Rrun 自身の接点で導通を維持する必要があるため、Xstart の下に Rrun の a 接点を並列に追加します。

1. サイドパネルの「Select (the arrow keys)」の ←（またはキーボードの ←）を 2 回押して、選択を Xstart に移動します。
2. 「Insert the instructions」で **Below** をクリックします。
3. **-| |-**（または **C**）をクリックし、変数に Rrun を選んで **OK** をクリックします。Rrun の接点が Xstart の下に並列に配置されます。
4. 「Insert the instructions」を **After** に戻します。

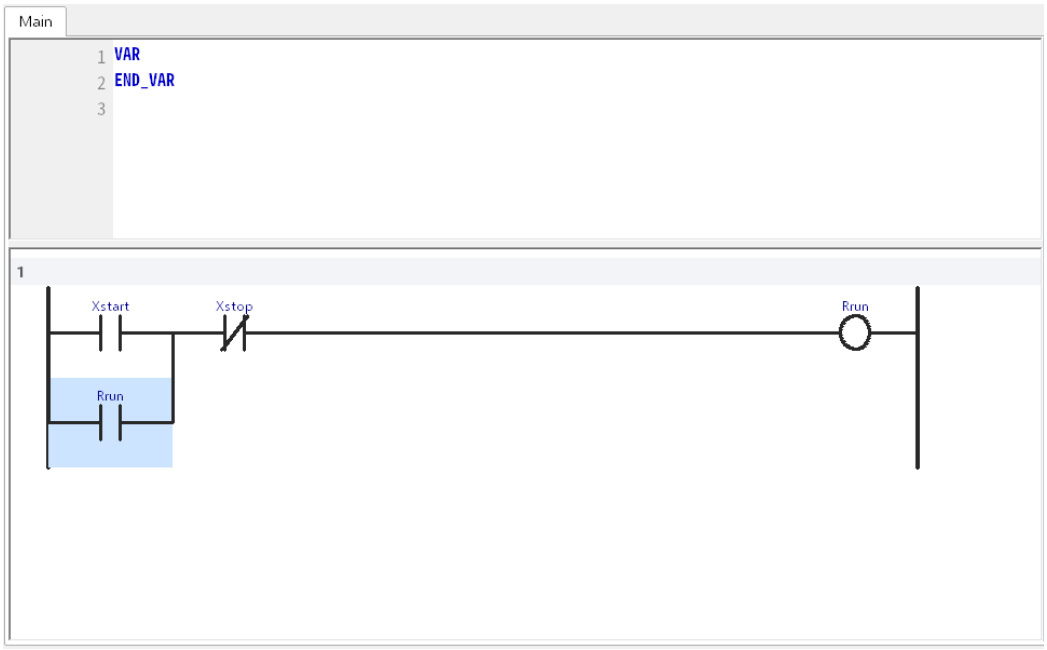


図 5-13　Xstart の下に Rrun を並列に配置：Xstart を押すと Rrun が自己保持し、Xstop を押すと解除

*ヒント：ショートカット：Xstart を選択した後、キーボードの **B** を押すだけで、その下に a 接点が並列に追加されます（続けて変数 Rrun を入力）。「Insert the instructions」を切り替える必要はありません。

5.8　ステップ 8：ネットワーク 2 を追加：TON タイマーで 2 秒遅れてランプを点灯

まず `Main` 上部の宣言エリアで IEC の TON タイマーのインスタンスを宣言します。`VAR` と `END_VAR` の間に `tLamp : TON;` と入力します。

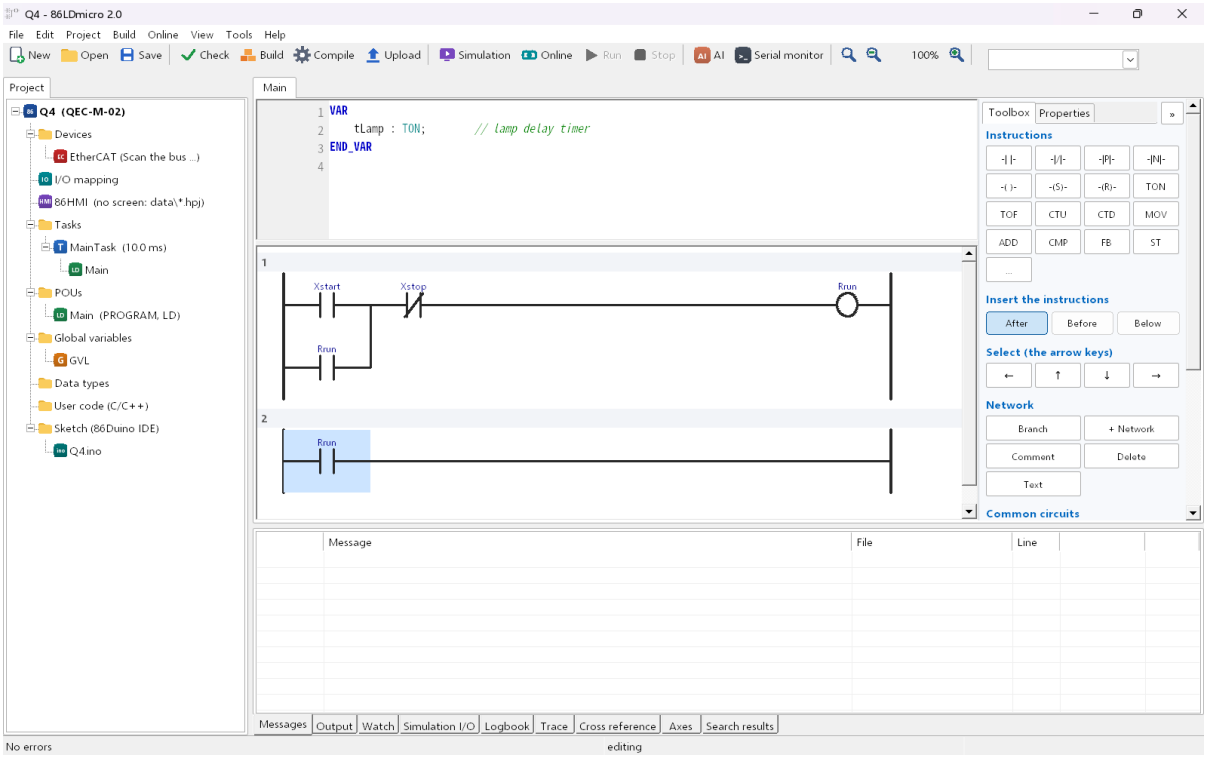


図 5-14　宣言エリアに tLamp : TON（ランプ遅延タイマー）を追加

1. ネットワーク 1 のタイトル行（番号 1 の行）をクリックしてネットワークを選択し、サイドパネルの「Network」の **+ Network**（または **Ctrl+Enter**）をクリックすると、下にネットワーク 2 が追加されて選択されます。
2. **-| |-**（**C**）をクリックし、変数に `Rrun` を選びます。
3. **FB**（ファンクションブロック、キー **K**）をクリックし、「Element」欄に `CALL(tLamp, IN := @, PT := T#2s, Q => @)` と入力して **OK** をクリックします。`@` はラングのパワーフローを表します。左から入ってくるパワーフローが IN に接続され、Q の出力が右へのパワーフローになります。
4. **-( )-**（**L**）をクリックし、変数に `Ylamp` を選びます。

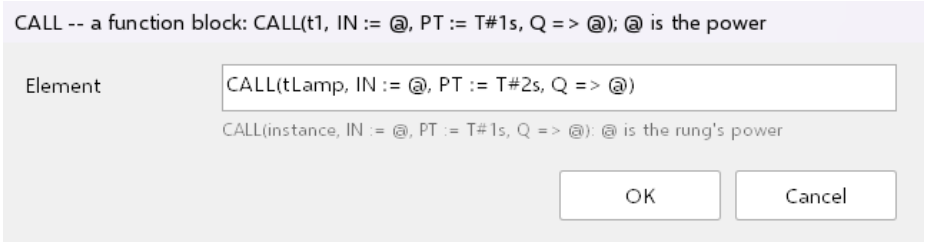


図 5-15　ファンクションブロック（CALL）のパラメーター：IN にパワーフロー、PT は 2 秒、Q の出力をパワーフローに戻す

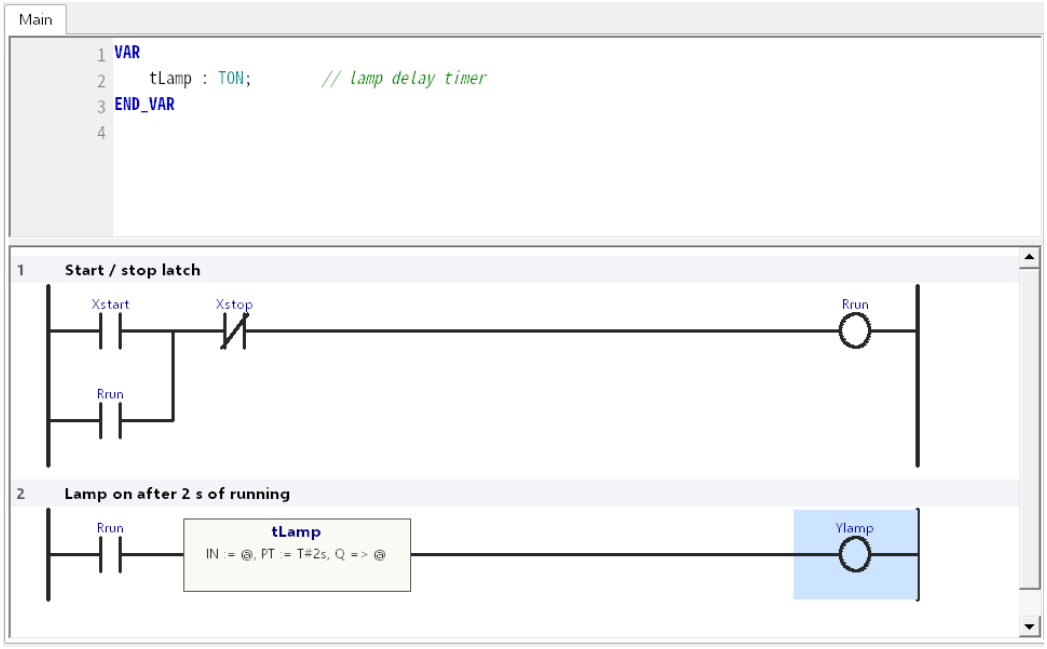


図 5-16　完成した 2 つのネットワーク：Rrun が 2 秒続くと Ylamp が点灯

最後にネットワークにタイトルを付けます。ネットワーク 1 のタイトル行をダブルクリックし、「Title」に 起動／停止の自己保持 と入力して **OK** をクリックします。ネットワーク 2 には 運転 2 秒後にランプ点灯 と入力します。ウィンドウ下部の「Body (text)」が、このネットワークの .lad テキストです。

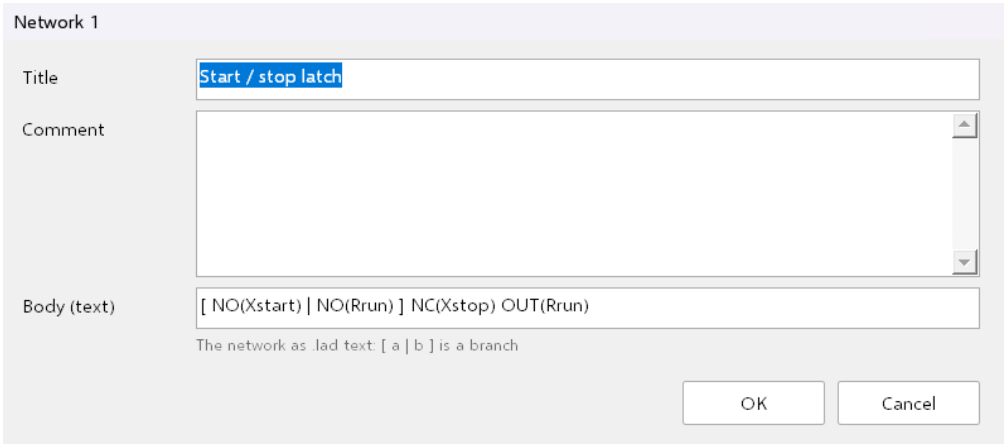


図 5-17　ネットワークのタイトル、コメント、内容（.lad テキスト）

サイドパネルの「Network」の **Text**（Ctrl+T）をクリックすると、Main 全体の .lad テキストが表示されます。次と同じになっているはずです（**Back to the diagram** をクリックすると戻ります）。

```
PROGRAM Main
VAR
    tLamp : TON;        // ランプ遅延タイマー
END_VAR
NETWORK '起動／停止の自己保持'
    [ NO(Xstart) | NO(Rrun) ] NC(Xstop) OUT(Rrun)
NETWORK '運転 2 秒後にランプ点灯'
    NO(Rrun) CALL(tLamp, IN := @, PT := T#2s, Q => @) OUT(Ylamp)
END_PROGRAM
```

5.9　ステップ 9：I/O マッピング：変数を QEC-M-02 の物理 I/O に接続する

現時点では、`Xstart`、`Xstop`、`Ylamp` はメモリー上の変数にすぎません。コントローラーの物理点に割り当てないと、アップロード後にボタンとランプは動作しません。

1. プロジェクトツリーで「**I/O mapping**」をダブルクリックし、**Add...** をクリックします。
2. 「Variable」で `Xstart`、「Channel」で `LocalIO.X00` を選び、**OK** をクリックします（ローカル I/O の方向はチャンネルで決まり、X は入力です）。
3. もう一度 **Add...** をクリック：`Xstop` を `LocalIO.X01` に割り当てます。
4. もう一度 **Add...** をクリック：`Ylamp` を `LocalIO.Y00`（出力）に割り当てます。
5. **Ctrl+S** を押して保存します。

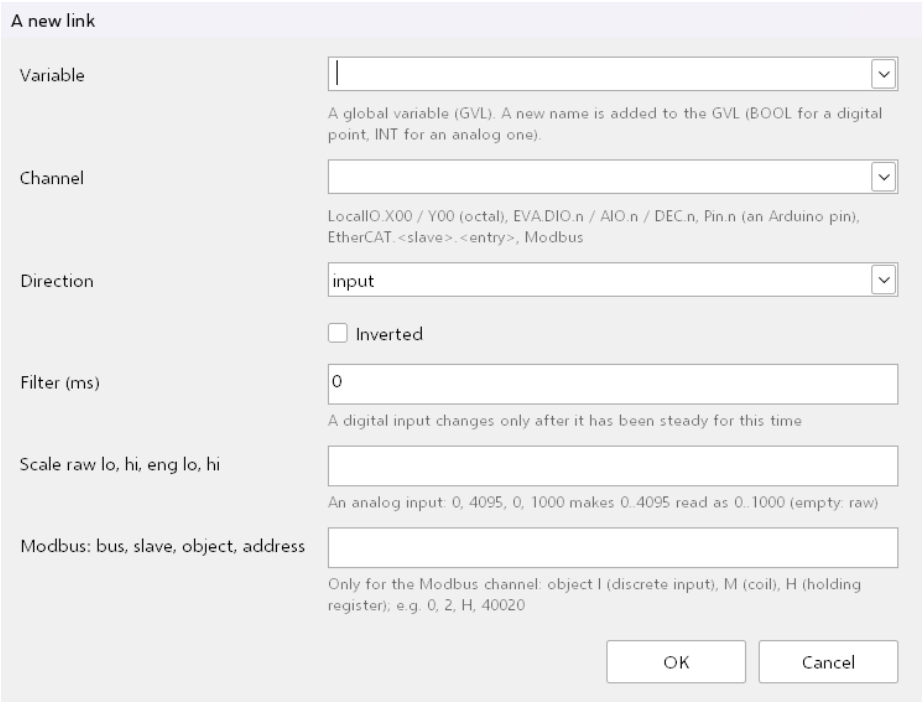


図 5-18　I/O マッピングの追加：変数、チャンネル、方向、反転、フィルター、スケーリング

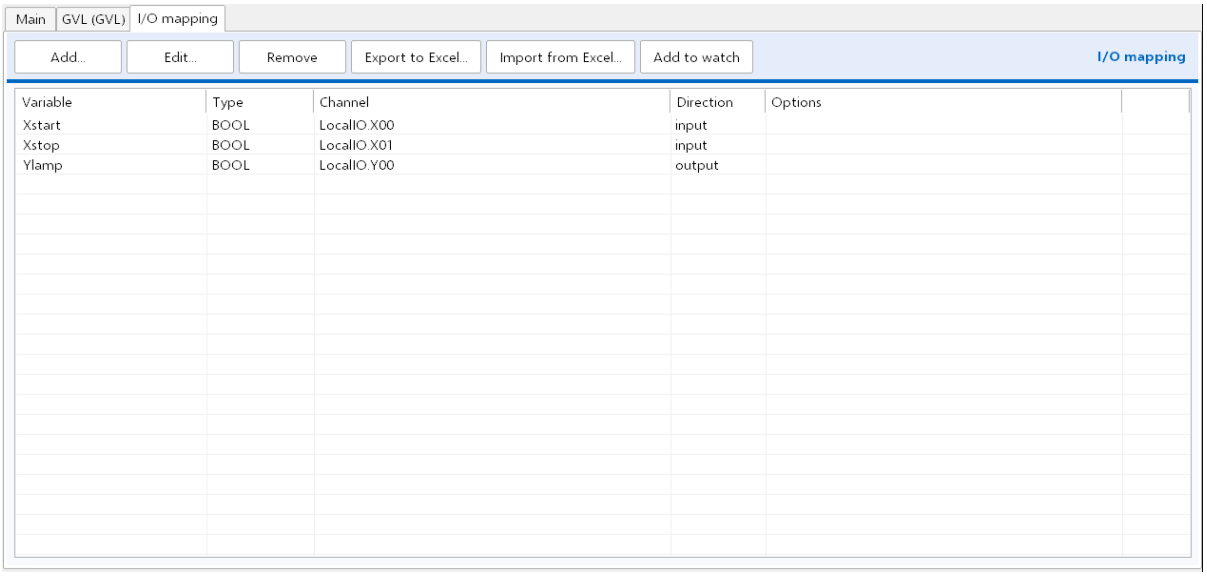


図 5-19　I/O マッピングページ：1 行が変数とチャンネルの 1 つの対応

- QEC-M-02 のローカル I/O のチャンネルは `LocalIO.X00`〜`X17`（入力）と `LocalIO.Y00`〜`Y17`（出力）で、**8進数**で番号が付けられています（X07 の次は X10）。
- `Rrun` と `tLamp` は内部変数なので、マッピングは不要です。
- ラダー図の変数を右クリックして「**I/O mapping of Xstart...**」を選び、マッピングウィンドウを直接開くこともできます。EtherCAT モジュールのチャンネルについては第 11 章、I/O マッピングの詳細については 4.6 節を参照してください。

5.10　ステップ 10：チェック（Shift+F7）とシミュレーション（F5）

アップロードの前にパソコン上でプログラムのロジックを確認しておくと、「書き込み、テスト、再修正」の時間を大幅に節約できます。

1. ツールバーの **Check**（Shift+F7）をクリックします。エラーは下部の「**Messages**」ウィンドウに一覧表示され、ダブルクリックするとエラーのある要素にジャンプします。ステータスバーに「No errors」と表示されたら次に進みます。
2. ツールバーの **Simulate**（F5）をクリックします。ステータスバーに「SIMULATION: RUN」とスキャン回数が表示され、パソコン上でプログラムの実行が始まります。
3. 下部の「**Simulation I/O**」ウィンドウに切り替えます。各 I/O マッピングの物理点が一覧表示されます。`LocalIO.X00` を 1 回ダブルクリック（ON）し、もう 1 回ダブルクリック（OFF）して、起動ボタンを 1 回押す操作をシミュレートします。
4. ラダー図を確認します。パワーフローのある線分が緑色になり、要素の下に TRUE／FALSE が表示されます。`Rrun` が TRUE になって保持され、`tLamp` のブロックが計時を始め、2 秒後に `Ylamp` が TRUE になり、「Simulation I/O」の `LocalIO.Y00` が ON と表示されます。
5. `LocalIO.X01` を 2 回ダブルクリック（ON、OFF）して停止ボタンを押す操作をシミュレートすると、`Rrun` と `Ylamp` が FALSE に戻ります。
6. もう一度 **Simulate**（F5）をクリックするとシミュレーションが終了し、編集状態に戻ります。

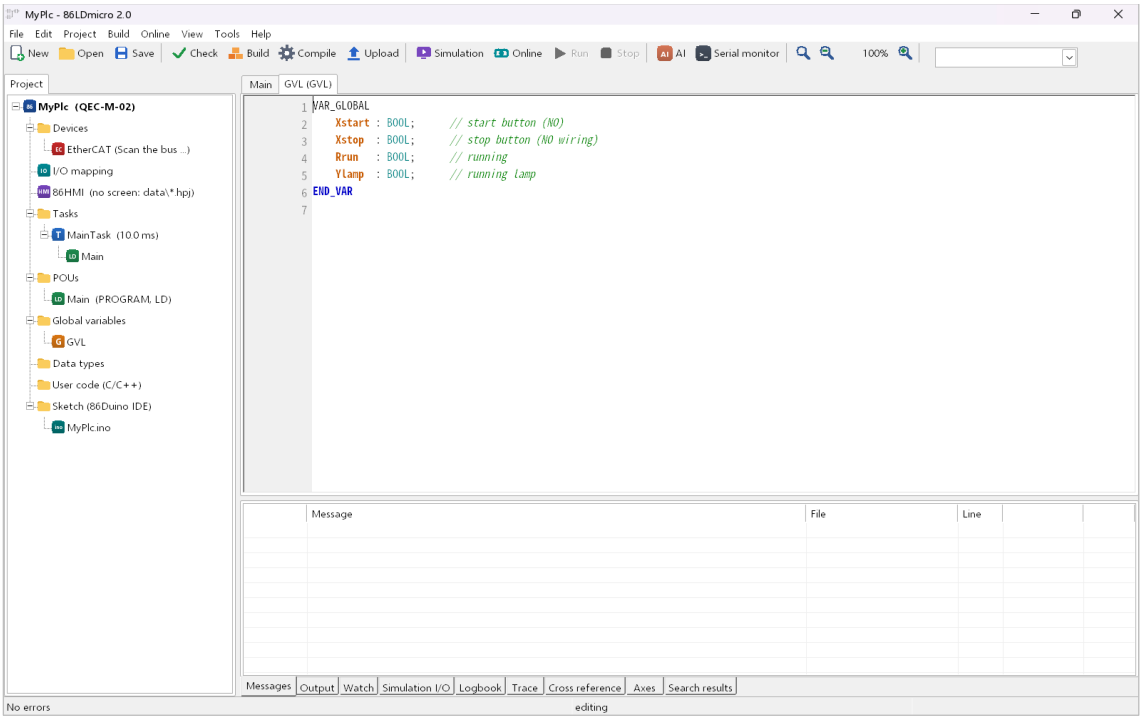


図 5-20　チェック完了：Messages ウィンドウにエラーがなく、ステータスバーに「No errors」と表示

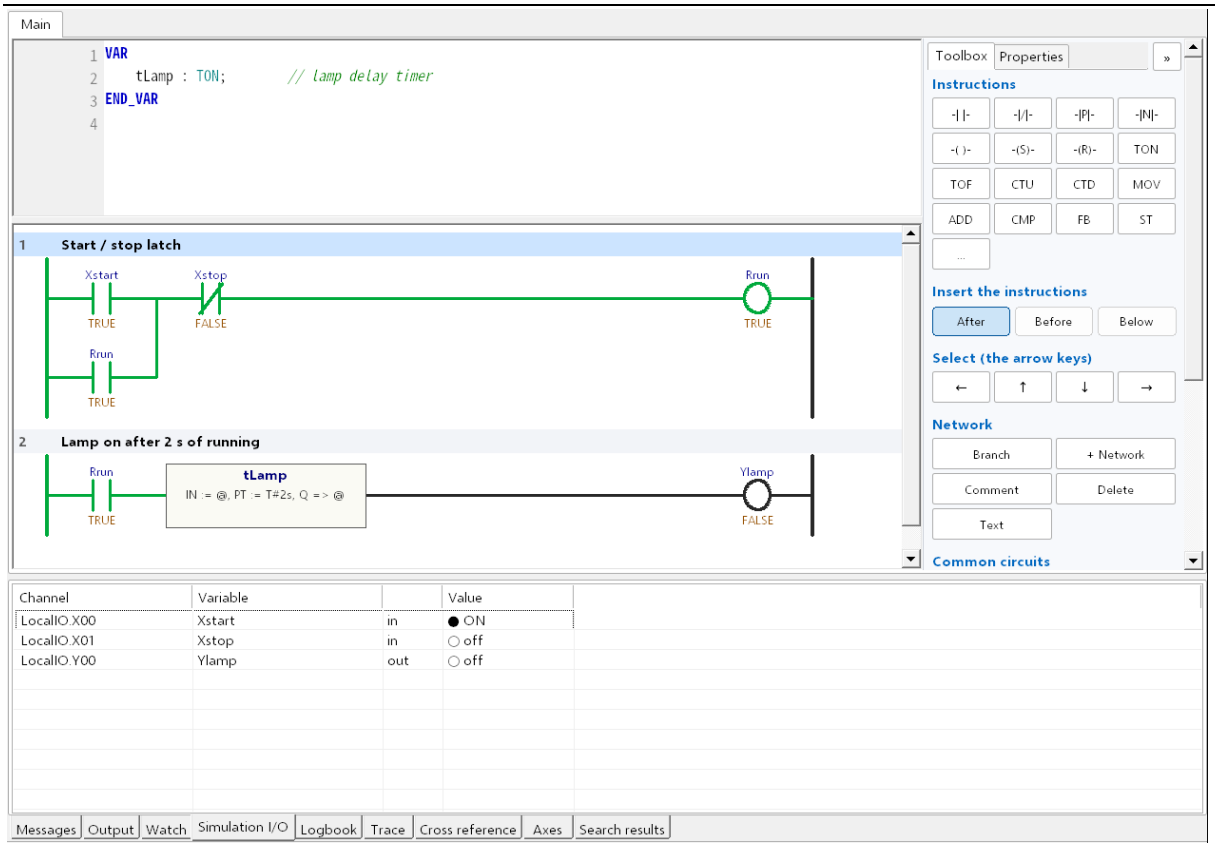


図 5-21　シミュレーション開始：「Simulation I/O」に LocalIO.X00（Xstart）などの物理点を表示

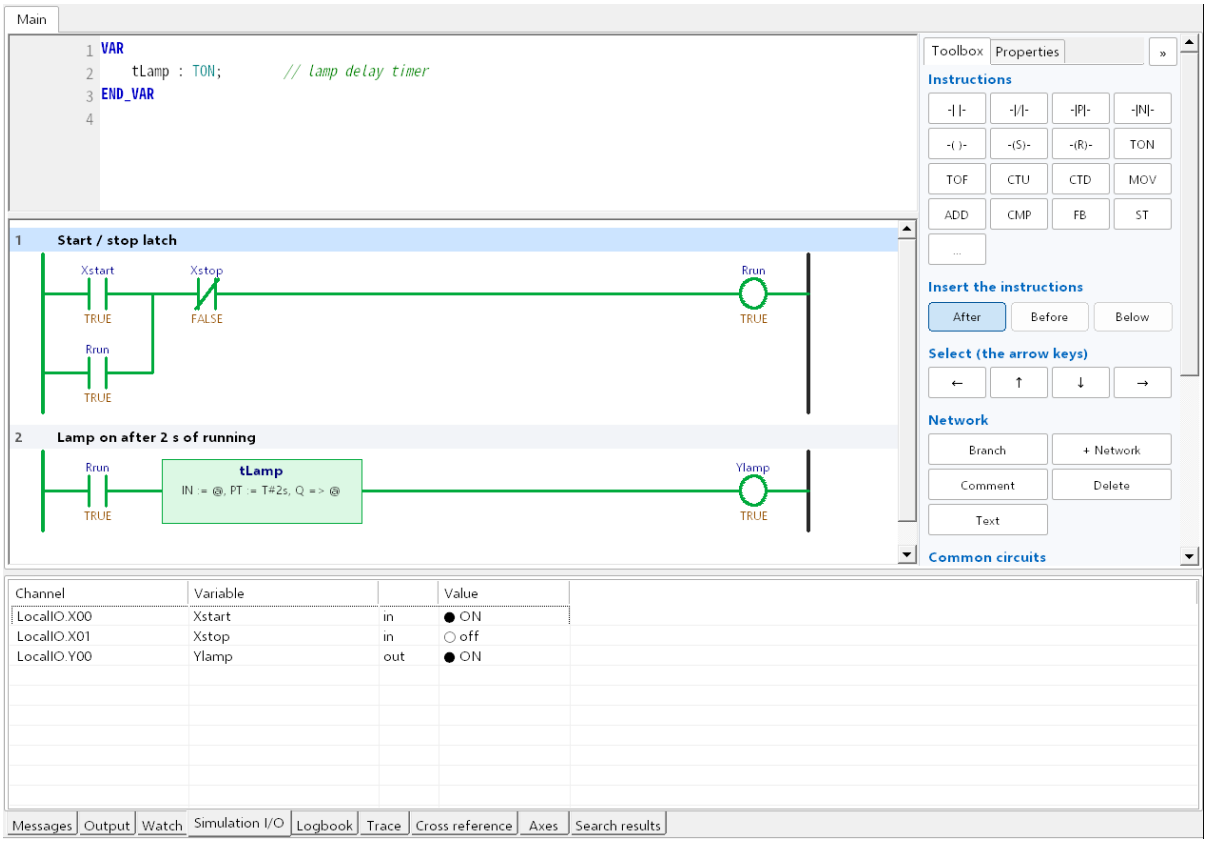


図 5-22　シミュレーション中：緑色はパワーフロー、タイマーブロックに経過時間を表示、LocalIO.Y00 は ON

複数の変数を同時に見たいときは、「**Watch**」ウィンドウに切り替え、上部に変数名（例：`Rrun`、`Main.tLamp.ET`）を入力して **Add** をクリックすると、リアルタイムの値が表示されます（9.3 節）。

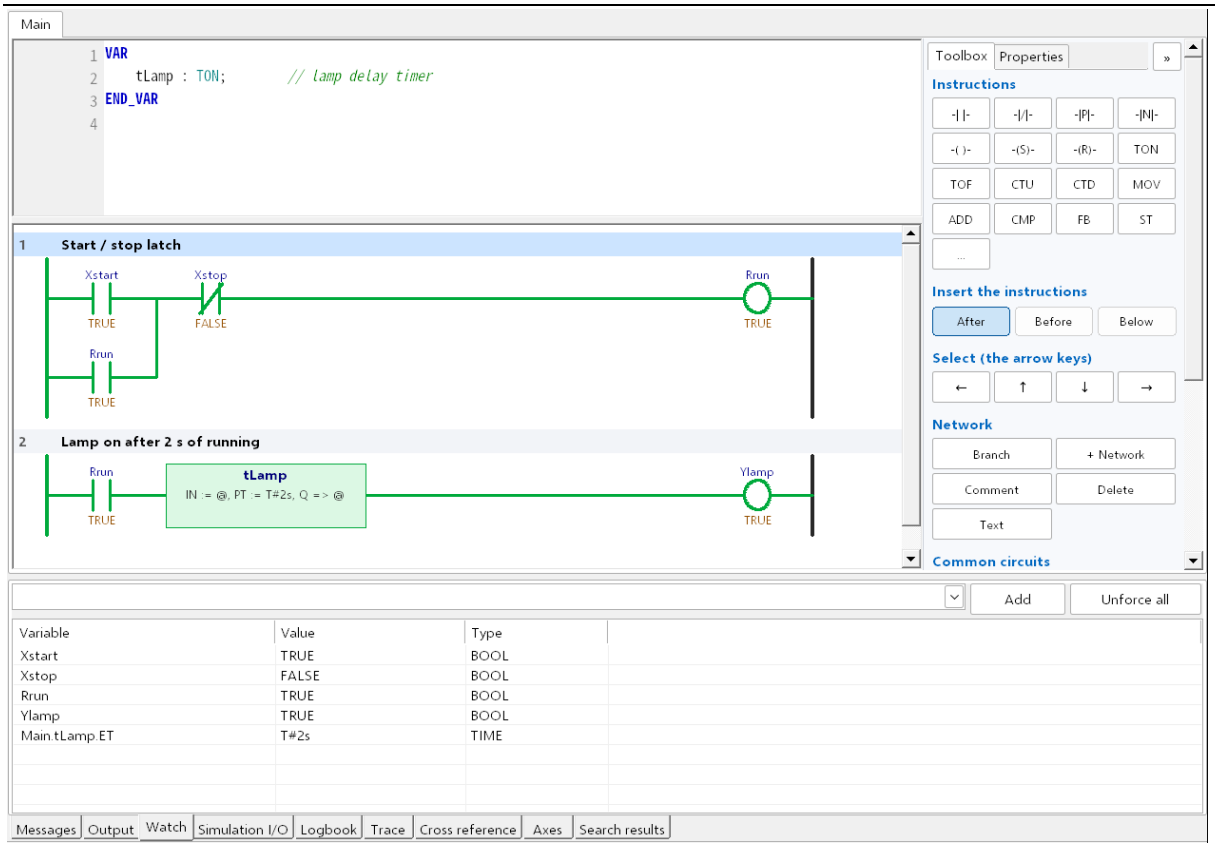


図 5-23　Watch ウィンドウ：追加した変数の値をリアルタイムに表示

*注意：物理入力にマッピングされた変数（例：`LocalIO.X00` に接続された `Xstart`）は、シミュレーション中は「Simulation I/O」によって決まります。Watch で直接書き込んでも、次のスキャンで入力値に上書きされます。入力は「Simulation I/O」で切り替えてください（9.2 節）。

5.11　ステップ 11：ビルド、コンパイル、アップロード（F8）

1. QEC-M-02 が USB ケーブルでパソコンに接続されていることを確認し、ツールバー右側のシリアルポートのドロップダウンリストでコントローラーの COM ポートを選択します（IDE から開いた場合は、IDE で選択したポートが自動的に設定されています）。
2. ツールバーの **Upload**（F8）をクリックします。86LDmicro は順に**ビルド**（プロジェクトを sketch の `plc86.cpp` などのファイルに書き出す）、**コンパイル**（86Duino IDE のコンパイラーを使用）、**アップロード**を行います。その過程は下部の「**Output**」ウィンドウの「**Build output**」に表示されます。
3. 出力には順に「Build: … files written to …」、「== Compiled …」、「== Uploading to COMx ...」と進捗が表示され、最後に「== Uploaded」と表示されます。コントローラーは自動的に再起動し、PLC プログラムの実行を開始します。

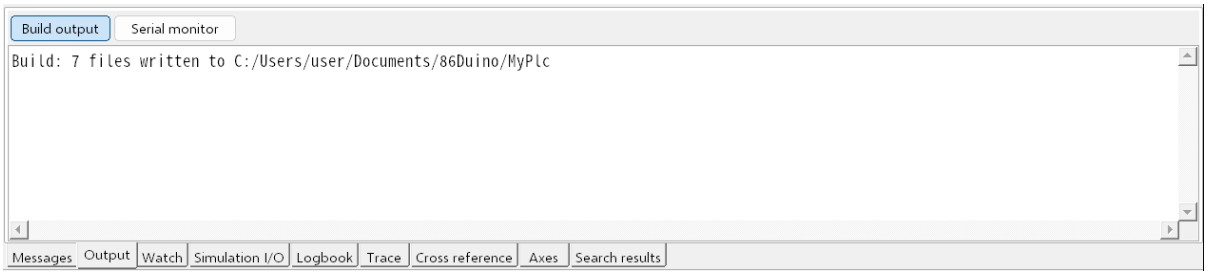


図 5-24　Output ウィンドウの「Build output」：ビルド、コンパイル、アップロードの過程はすべてここに表示

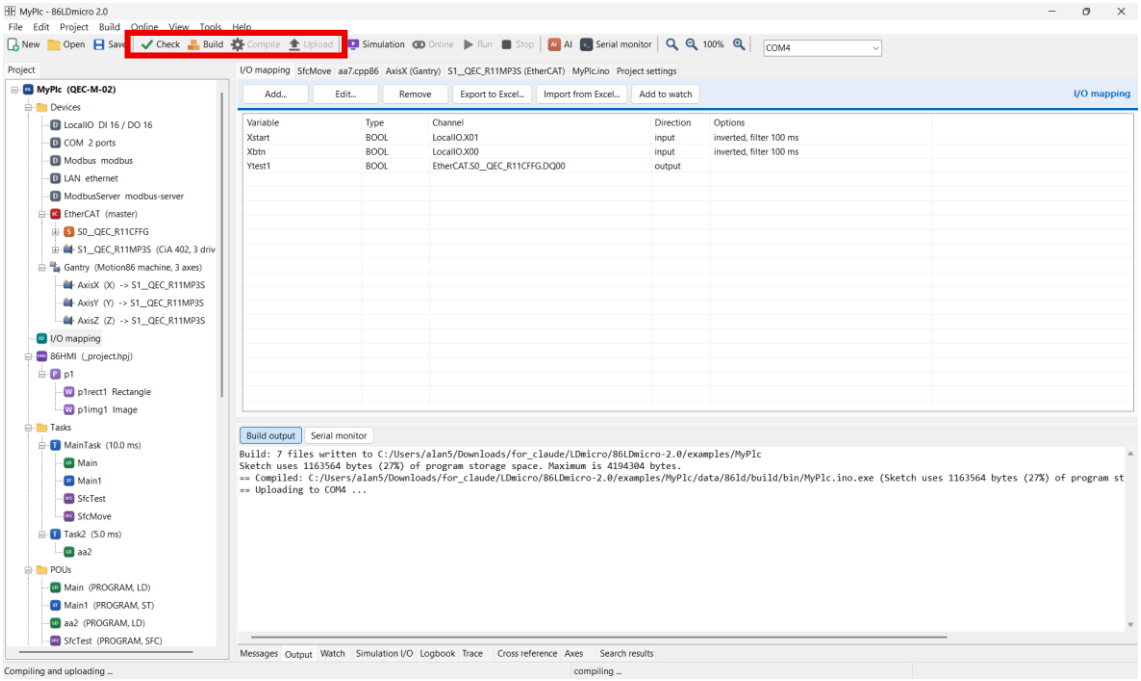


図 5-25　アップロード中

ビルド後、IDE から開いた 86LDmicro は IDE に sketch の再読み込みを通知します。IDE に `plc86.cpp`、`plc86.h` などのタブが表示されます。以後は IDE で直接コンパイル、アップロードすることもできます。

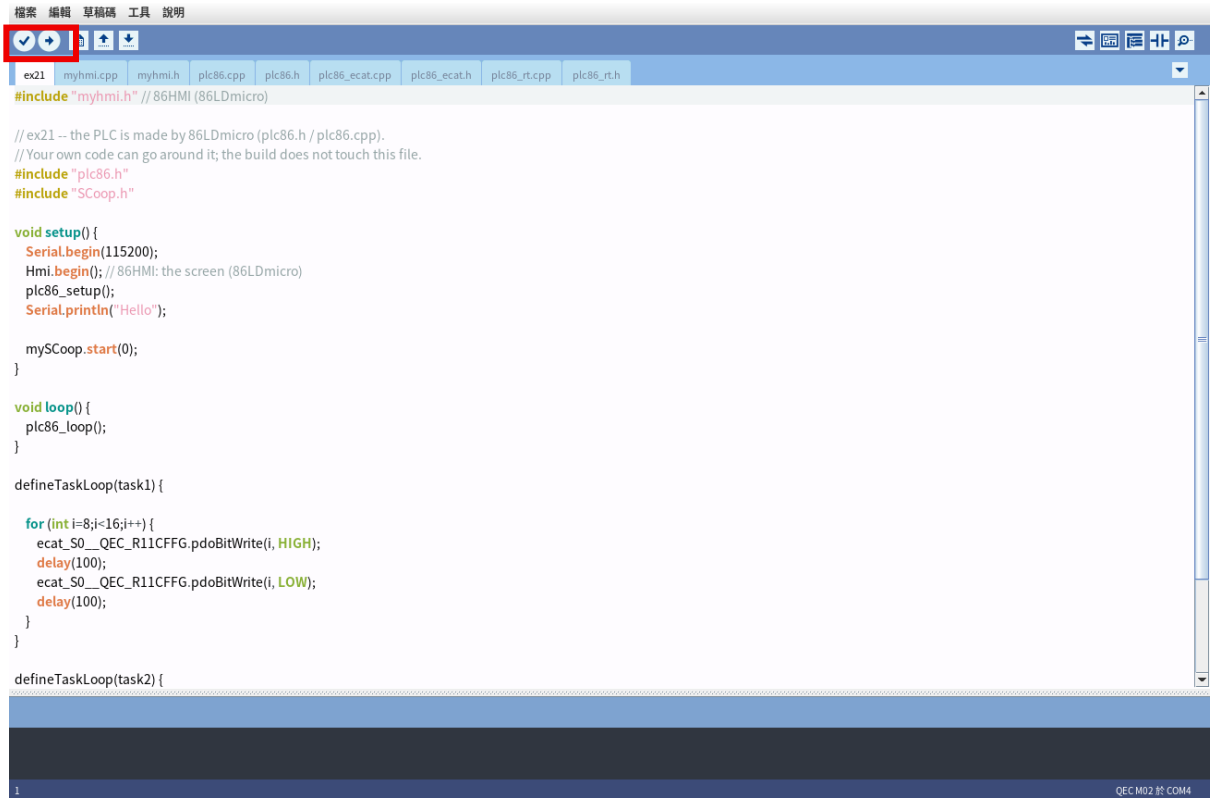


図 5-26　ビルド後の 86Duino IDE：sketch に plc86.cpp、plc86.h などのタブが追加

***注意：**IDE で保存していない `.ino` の変更は、ビルド後の再読み込みで上書きされます。先に IDE で保存してください。

アップロードが「Could not open COMx」で失敗した場合は、同じ COM ポートを使用している他のプログラム（例：IDE のシリアルモニター）を閉じてから、もう一度 F8 を押します。その他のメッセージと対処方法は 10.1 節を参照してください。

5.12　ステップ 12：オンラインモニター（Ctrl+F5）

1. アップロード後、86LDmicro は自動的にオンラインになります（「Tools > Options」の「Online again after an upload」、既定でオン）。または、ツールバーの **Online**（Ctrl+F5、「Online > Login to the controller」）をクリックします。
2. ステータスバーに「ONLINE COMx: RUN」とスキャン時間が表示され、ラダー図にはコントローラー上のリアルタイムのパワーフローと値が表示されます。
3. 実際の起動ボタン（X00）を押すと、`Rrun` が TRUE になり、2 秒後に Y00 のランプが点灯します。停止ボタン（X01）を押すと消灯します。
4. 終了するときは「**Online > Logout**」を選択します。

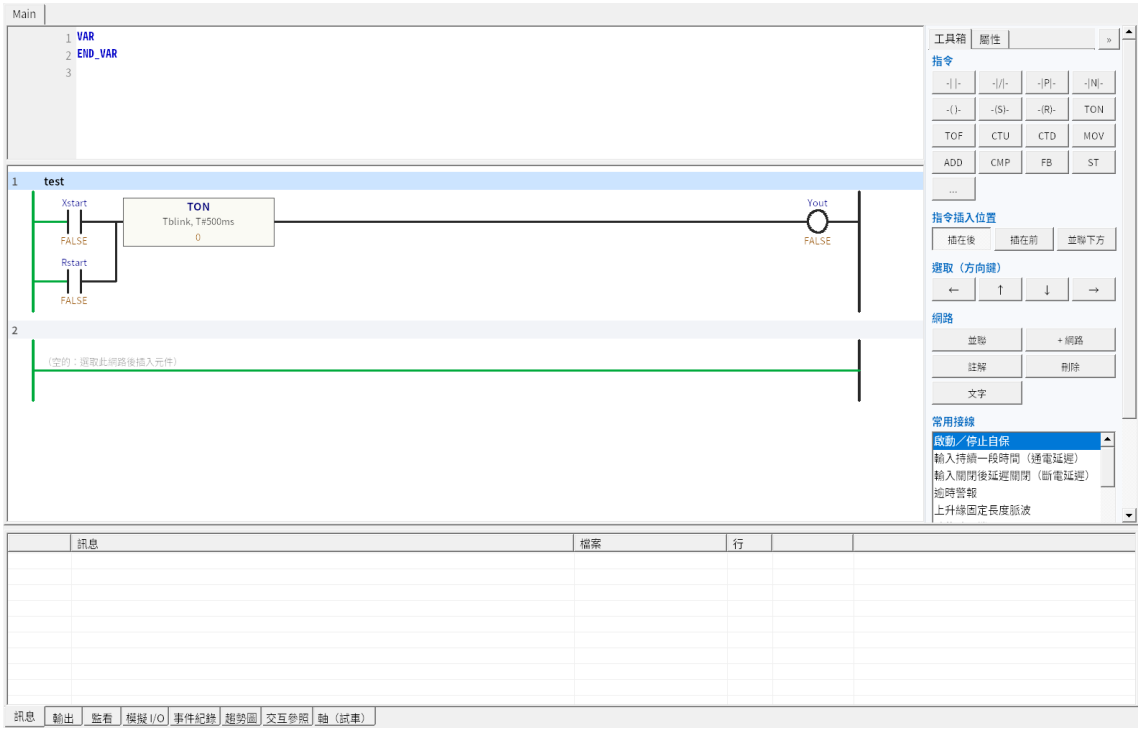


図 5-27　オンラインモニター：ラダー図にコントローラーのリアルタイムの状態を表示

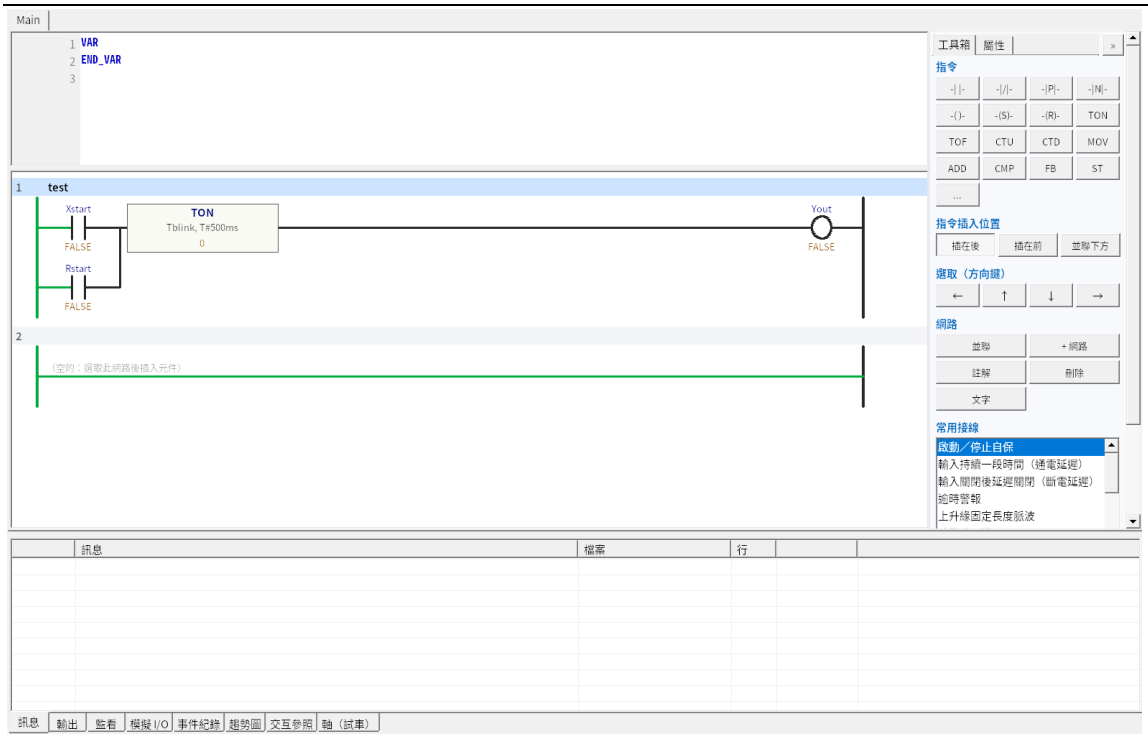


図 5-28　オンライン時のステータスバー：ONLINE COMx: RUN とスキャン時間

「Watch」に変数を追加すると、BOOL はダブルクリックで切り替え、右クリックで強制できます（9.3 節）。オンラインモニター、RUN／STOP、Logbook などの詳細は 10.4 節を参照してください。

> ***ヒント：**完成です！第 6 章では同じ流れで、よく使う例をさらに紹介します（TOF、カウント、点滅、ST、SFC、C/C++）。第 7 章ではラダー図の全命令を一覧にしています。

第 6 章　実践例

前章では「起動／停止の自己保持＋遅延ランプ」で一連の流れを一通り説明しました。本章では、よく使う例をさらに 7 つ紹介します。各例は同じ形式で構成されています：**シナリオ → 変数表 → プログラム（.lad／ST／SFC／C++ テキスト）→ 詳細手順表 → 期待される結果 → 画面。**

各例はそれぞれ独立した sketch（名前は各節に記載）で、流れは第 5 章と同じです：プロジェクトの作成 → 変数の宣言 → プログラムの作成 → I/O マッピング → チェックとシミュレーション → アップロード。プログラムのテキストはラダー図のテキストモード（Ctrl+T）に直接貼り付けることも、手順表に従って Toolbox から 1 つずつ挿入することもできます。

> ***重要：**変数表の「I/O マッピング」欄は推奨値です。実際の配線に合わせて調整してください。X…／Y… の各変数は「I/O mapping」で物理チャンネルに割り当てる必要があります（第 5 章のステップ 9）。割り当てないと、アップロード後に I/O が動作しません。

6.0　共通原則（最初にお読みください）

以下の 4 つの原則は本章のすべての例に共通するもので、LDmicro 1.1 から移行する際に最も考え方の切り替えが必要な点でもあります。これを理解しておけば、コンパイルや実機での動作でつまずくことはほとんどありません。

6.0.1　原則 ①：変数は必ず先に宣言し、型は宣言で決まる

LDmicro 1.1 は名前の接頭辞（X、Y、R、T、C）と要素ウィンドウの Source から変数を自動的に作成していましたが、86LDmicro 2.0 は IEC 61131-3 に従い、**すべての変数を先に宣言して型を指定する必要があります。**

名前の接頭辞は命名の慣習にすぎず、変数の種類を決めるものではありません。

宣言する場所	書き方	使用できる範囲	用途
グローバル変数リスト GVL（プロジェクトツリーの「Global variables」）	`VAR_GLOBAL ... END_VAR`	すべての POU、I/O マッピング、86HMI、C/C++（`GVL.名前`）	入力、出力、プログラム間で共有するフラグと数値
POU 上部の宣言エリア	`VAR ... END_VAR`	この POU のみ（ほかの場所からは `プログラム.変数` で読み取る）	ファンクションブロックのインスタンス（`tLamp : TON;`）、1.x タイマー、ステップ番号などのローカル変数
GVL の保持領域	`VAR_GLOBAL RETAIN ... END_VAR`	GVL と同じ	電源断後も保持する値（例：累計回数）

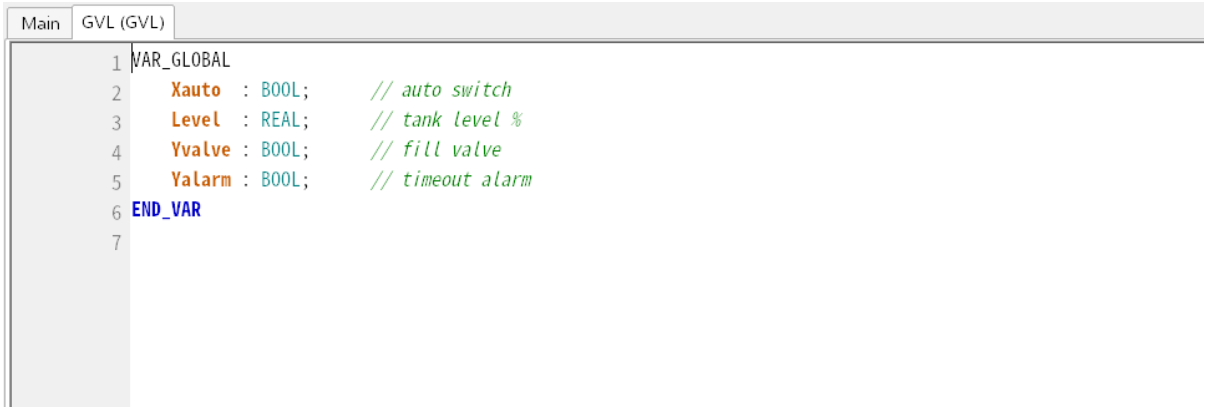


図 6-1　GVL エディター：1 行に 1 つの変数、型はコロンの後、// の後はコメント

ラダー図で宣言されていない名前（例：入力ミスの `Xstrat`）を使うと、要素が**赤い背景**で表示されます。**Check**（Shift+F7）をクリックすると、「Messages」ウィンドウに「unknown variable Xstrat: declare it in VAR ... END_VAR, or in a global variable list (GVL)」と表示されます。その行をダブルクリックすると要素にジャンプします。

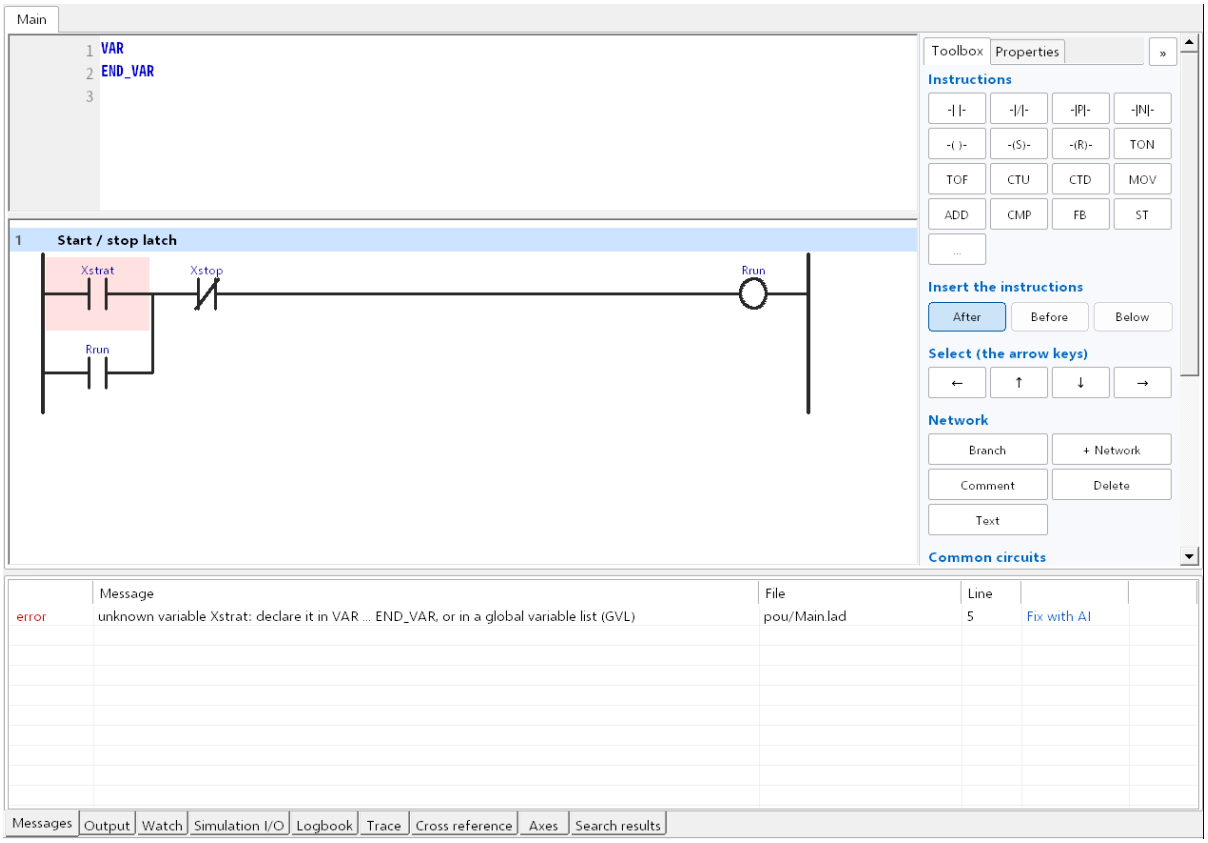


図 6-2　未宣言の変数：要素は赤い背景、ビルド出力に「Build: the project has errors」と表示

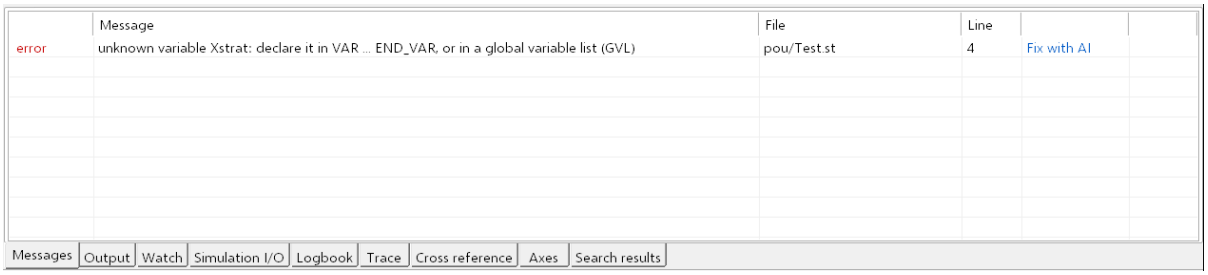


図 6-3　「Messages」ウィンドウ：1 行に 1 つのエラー、ダブルクリックで該当要素にジャンプ

6.0.2　原則 ②：I/O は「I/O mapping」を通してのみ物理点に接続される

名前が `X00` や `Xstart` だからといって、変数が物理入力になるわけではありません。「I/O mapping」でチャンネル（例：`LocalIO.X00`）を指定した変数だけが、各スキャンの開始時に物理入力を読み込み、スキャンの終了時に物理出力を書き出します。

変数の種類	I/O マッピングは必要？	説明
物理入力（ボタン、センサー）	必要	`LocalIO.X00`〜`X17`（8 進数）または EtherCAT モジュールの入力に割り当て
物理出力（ランプ、電磁弁、モーター）	必要	`LocalIO.Y00`〜`Y17` または EtherCAT モジュールの出力に割り当て
アナログ量（液位、温度）	必要（アナログチャンネル）	マッピングウィンドウの「Scaling」で生の値を工学単位に変換（4.6 節）
内部フラグ、数値（`R…`、カウント値）	不要	メモリー上のみ
ファンクションブロックのインスタンス、1.x のタイマー／カウンター変数	不要	POU の VAR で宣言すれば十分

ラダー図の接点が BOOL のグローバル変数を読み取っているのに、その変数に I/O マッピングがなく、どのプログラムや 86HMI からも書き込まれていない場合、ビルド時に「... is only read here: it has no I/O mapping and nothing writes it -- it is always FALSE」という警告が表示され、マッピング漏れを知らせます。

> *ヒント：検証の近道：プログラムが完成したら、まず **Check**（Shift+F7）→ **Simulate**（F5）を実行し、「Simulation I/O」で入力チャンネルをダブルクリックして、出力チャンネルとラダー図のパワーフローが期待どおりかを確認します。シミュレーションで問題がなくなってからアップロードすれば、「書き込み、テスト、再修正」の時間を大幅に節約できます。

6.0.3　原則 ③：1 つの変数は 1 か所からだけ書き込む。スキャンとネットワークの順序に注意

PLC は「入力の読み込み → プログラムを順に実行 → 出力の書き出し」を 1 回のスキャンとし、タスクの周期で繰り返します（新しいプロジェクトでは 10 ms ごと）。同じ POU のネットワークは**上から下へ**実行され、同じタスク内のプログラムはタスクで設定した順序で実行されます（4.3 節）。

- 同じ変数を 2 つの `OUT` コイルで駆動しないでください。後のネットワークが前の結果を上書きします。たとえば、ネットワーク 1 が `NO(Xstart) OUT(Ymotor)`、ネットワーク 2 が `NO(Xstop) OUT(Ymotor)` の場合、`Xstart` が TRUE でも、`Ymotor` はネットワーク 2 によって決まります（FALSE）。複数の条件が必要な場合は、それらを**同じ**コイルの前に並列または直列に配置します。
- `SET`／`RST` を組で使う場合は例外で、後のネットワークに書いたほうが優先されます（たとえば停止優先にするには、`RST` を `SET` の後に置きます）。
- ある変数をどのプログラムが書き込んでいるかを調べるには、変数を右クリックして「**Where is <変数> used?**」を選ぶか、「Cross reference」（Ctrl+Shift+F、9.5 節）を使用します。
- ラダー図、ST、C/C++ は GVL の変数を共有できます。この場合も「書き込むのは 1 か所」のルールを守り、ほかの場所では読み取りだけにします（例 6.7）。

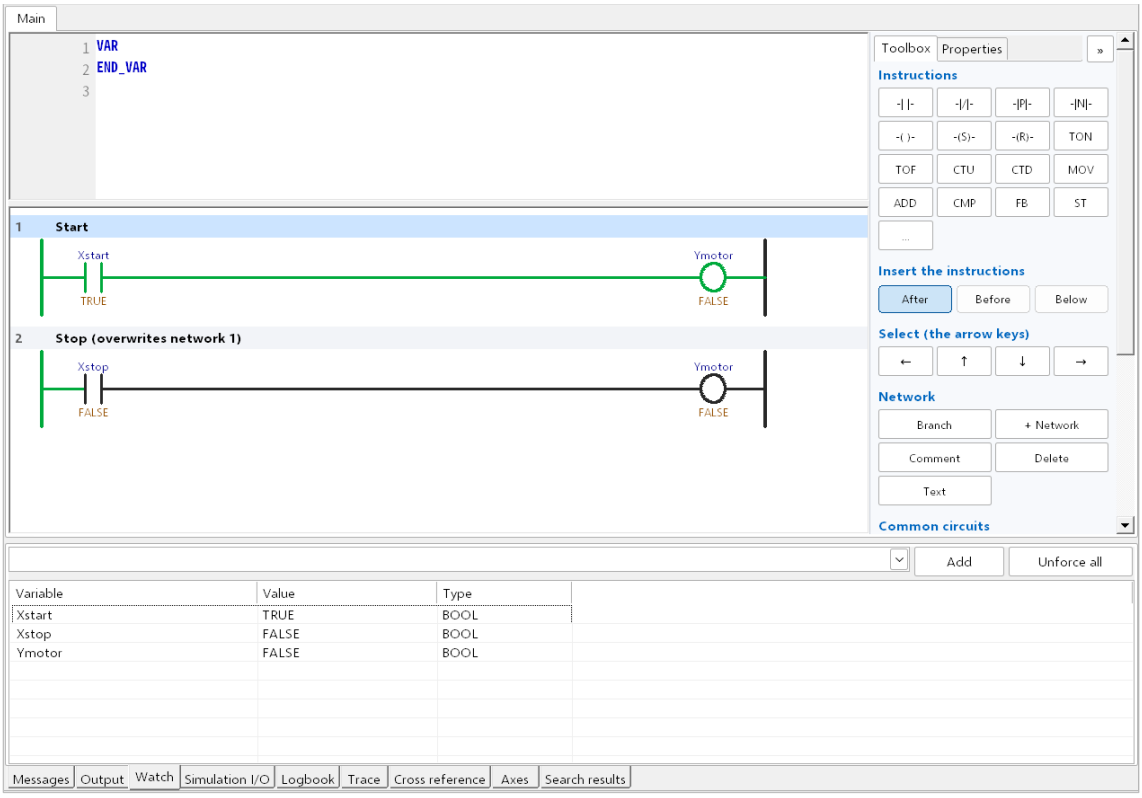


図 6-4　2 つの OUT が同じ変数を駆動：後のネットワークが前の結果を上書き

6.0.4 原則 ④：タイマーとカウンター：IEC ファンクションブロックまたは 1.x ブロック

86LDmicro 2.0 には 2 種類のタイマーとカウンターがあり、混在させて使用できます。

	IEC ファンクションブロック（推奨）	**LDmicro 1.x 互換ブロック**
種類	TON、TOF、TP、CTU、CTD、CTUD、R_TRIG、F_TRIG、SR、RS	TON、TOF、RTO、RTL、TCY、THI、TLO、CTU、CTD、CTC、CTR…
挿入方法	**FB**（キー K）：`CALL(インスタンス, IN := @, PT := T#2s, Q => @)`	Toolbox の **TON**（O）、**TOF**（F）、**CTU**（U）、**CTD**（D）、その他は「...」から
宣言	インスタンスを VAR で宣言：`tLamp : TON;`	タイマー／カウンター変数を VAR で宣言：`t1 : DINT;`、`c1 : INT;`
時間	実時間で、タスクの周期とは無関係	PLC 周期でカウント（遅延 ÷ 周期、整数化）、1.1 と同じ
結果の読み取り	メンバーはどこからでも読み取り可能：`NO(tLamp.Q)`、`CMP(cBox.CV >= 10)`、ST では `tLamp.ET`	カウント変数の値を読み取る（例：`GEQ(c1, 10)`）

LDmicro 1.1 ではタイマーやカウンターを直接接点として使えず、まず `R_…` 内部リレーを駆動して「橋渡し」する必要がありました。2.0 では、IEC ファンクションブロックの出力（`.Q`、`.CV`、`.ET`）がそのまま変数なので、どのネットワークの接点や比較でも直接使用できます。

6.1 起動／停止の自己保持

最も典型的な PLC 回路で、リレーの自己保持に相当します。2 つのボタンで 1 つの出力を制御し、起動を押した後はボタンを離しても出力は ON のまま保持され、停止を押すまで続きます。

シナリオ：起動と停止の 2 つのボタンでモーター（またはランプ）を制御します。起動を 1 回押すと運転が続きます。sketch 名：`Ex61_Latch`。

変数	型	I/O マッピング	用途
`Xstart`	BOOL	`LocalIO.X00`	起動ボタン（a 接点）
`Xstop`	BOOL	`LocalIO.X01`	停止ボタン（a 接点で配線）
`Ymotor`	BOOL	`LocalIO.Y00`	モーター／運転ランプ

プログラム（Main、.lad テキスト）

```
PROGRAM Main
VAR
END_VAR
NETWORK '起動／停止の自己保持'
    // Xstart を押すと起動して自己保持、Xstop を押すと停止
    [ NO(Xstart) | NO(Ymotor) ] NC(Xstop) OUT(Ymotor)
END_PROGRAM
```

詳細手順

#	操作
1	第 5 章のステップ 1〜3 に従って sketch `Ex61_Latch` と新しいプロジェクトを作成します。
2	「Global variables > GVL」をダブルクリックし、`Xstart`、`Xstop`、`Ymotor`（BOOL）を宣言して、Ctrl+S で保存します。
3	`Main` を開き、ネットワーク 1 をクリックして選択します。**C** → 変数 `Xstart` → OK、**N** → `Xstop`、**L** → `Ymotor` の順に押します。
4	← を 2 回押して `Xstart` を選択し、**B**（下に a 接点を並列に追加）→ 変数 `Ymotor` → OK。
5	ネットワーク 1 のタイトル行をダブルクリックし、タイトルに 起動／停止の自己保持、コメントに `Xstart` を押すと起動して自己保持、`Xstop` を押すと停止 と入力します。
6	「I/O mapping」で追加：`Xstart` → `LocalIO.X00`、`Xstop` → `LocalIO.X01`、`Ymotor` → `LocalIO.Y00`。
7	Shift+F7 でチェック。F5 でシミュレーション：「Simulation I/O」で `LocalIO.X00` を 2 回ダブルクリック（ON、OFF）し、続いて `LocalIO.X01` を 2 回ダブルクリックして、自己保持と停止を確認します。
8	F8 でアップロード。

期待される結果：X00 を 1 回押すと Y00 = ON になって保持され、X01 を 1 回押すと Y00 = OFF になります。2 つのボタンを同時に押し続けると Y00 = OFF です（停止優先）。

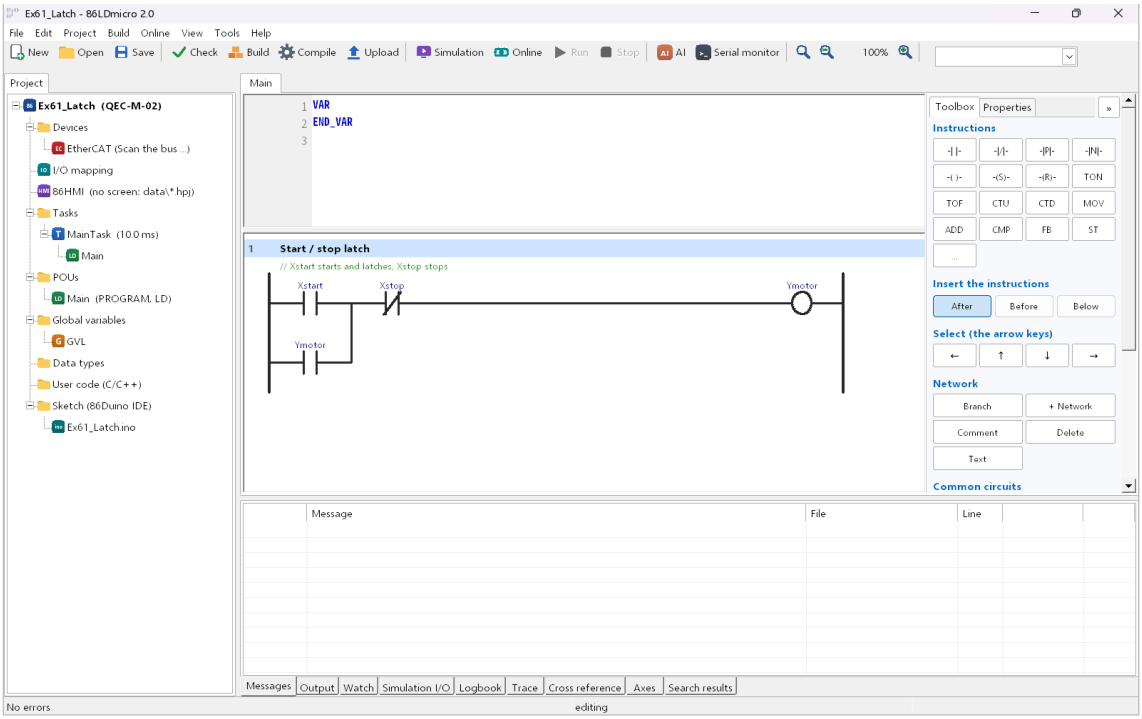


図 6-5　6.1 のラダー図：Xstart と自己保持接点 Ymotor を並列にし、Xstop と直列にして Ymotor を駆動

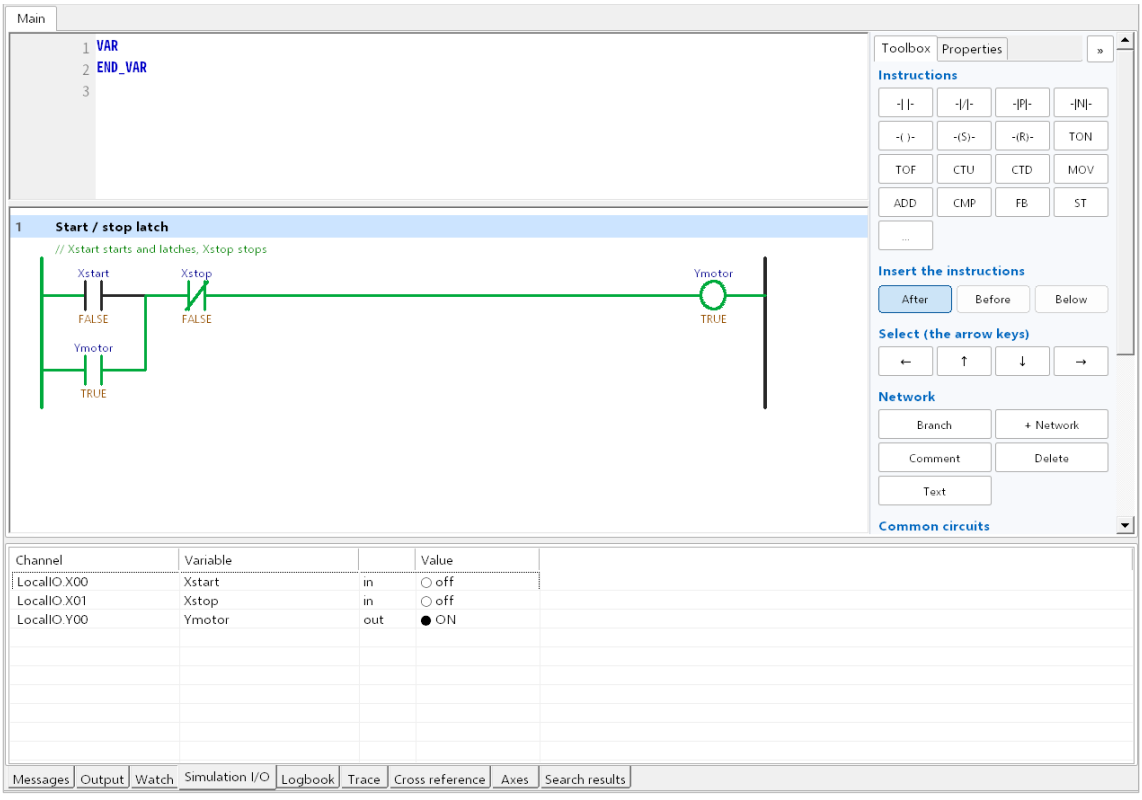


図 6-6　6.1 のシミュレーション：起動ボタンを離しても Ymotor は TRUE のまま（自己保持）

別の方法：Common circuits。サイドパネルの「Common circuits」で **Start / stop latch** を選び、**Insert** をクリックして、起動ボタン、停止ボタン、出力の名前を入力すると、同じネットワークが挿入され、新しい変数が GVL に自動的に宣言されます。

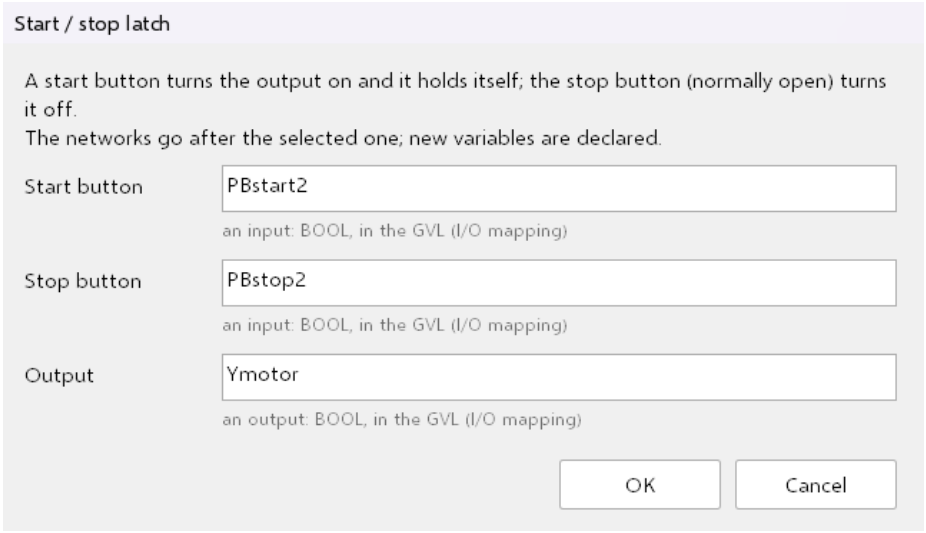


図 6-7　Common circuits の「Start / stop latch」：ボタンと出力の名前を入力

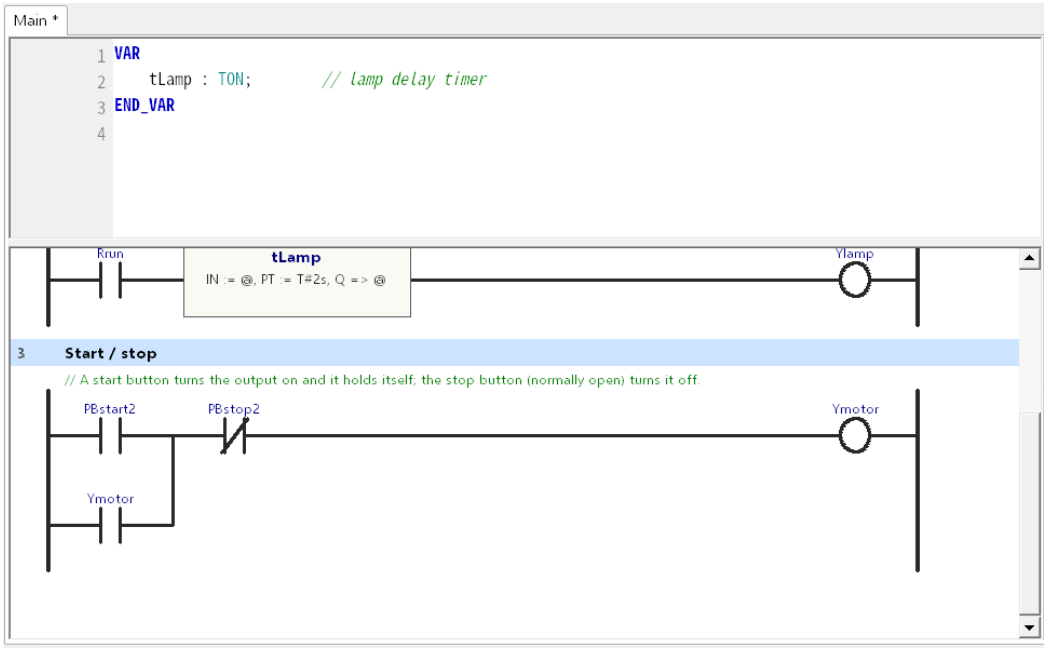


図 6-8　挿入後：テンプレートが作成した自己保持ネットワーク。新しい変数は GVL に自動的に追加済み

バリエーション：SET／RST。同じ機能は、セットコイルとリセットコイルを使って 2 つのネットワークで書くこともできます。RST を後に書くので、2 つのボタンを同時に押したときは停止が優先されます。

```
NETWORK '起動：セット'
    NO(Xstart) SET(Ymotor)
NETWORK '停止：リセット（後に書くので停止優先）'
    NO(Xstop) RST(Ymotor)
```

6.2 遅延消灯（TOF）

スイッチを入れるとすぐにランプが点灯し、スイッチを切った後もさらに 5 秒間点灯してから消灯します。廊下の照明、トイレの照明、会議室の照明によく使われます。

シナリオ：スイッチ ON → ランプはすぐに ON、スイッチ OFF → 5 秒遅れてランプ OFF。sketch 名：`Ex62_Tof`。

変数	型	I/O マッピング	用途
`Xsw`	BOOL	`LocalIO.X00`	照明スイッチ
`Ylight`	BOOL	`LocalIO.Y00`	照明ランプ
`tOff`	TOF（IEC ファンクションブロック、Main の VAR）	—	オフディレー 5 秒

プログラム（Main、.lad テキスト）

```
PROGRAM Main
VAR
    tOff : TOF;          // オフディレー 5 秒
END_VAR
NETWORK '遅延消灯'
    // スイッチ ON ですぐ点灯、スイッチ OFF 後さらに 5 秒点灯
    NO(Xsw) CALL(tOff, IN := @, PT := T#5s, Q => @) OUT(Ylight)
END_PROGRAM
```

詳細手順

#	操作
1	sketch `Ex62_Tof` と新しいプロジェクトを作成し、GVL で `Xsw`、`Ylight`（BOOL）を宣言します。
2	POU `Main` を開き、上部の宣言エリアの `VAR` と `END_VAR` の間に `tOff : TOF;` と入力します。
3	ネットワーク 1 をクリックして選択し、**C** → 変数 `Xsw`。
4	キー **K**（FB）を押す →「Element」に `CALL(tOff, IN := @, PT := T#5s, Q => @)` と入力 → OK。
5	**L** → 変数 `Ylight`。
6	「I/O mapping」：`Xsw` → `LocalIO.X00`、`Ylight` → `LocalIO.Y00`（`tOff` はマッピング不要）。
7	F5 でシミュレーション：`LocalIO.X00` をダブルクリックして ON にすると、`Ylight` はすぐに TRUE になります。もう一度ダブルクリックして OFF にし、「Watch」に `Main.tOff.ET` を追加して経過時間を確認します。5 秒後に `Ylight` が FALSE になります。
8	F8 でアップロード。

期待される結果：`Xsw` = 1 のとき `Ylight` はすぐに 1 になります。`Xsw` が 1 から 0 になった後、`Ylight` は 5 秒間 1 を保持してから 0 になります（`tOff.ET` は 0 から `T#5s` までカウント）。5 秒以内にスイッチを再び入れると、計時がやり直しになり、ランプは点灯し続けます。

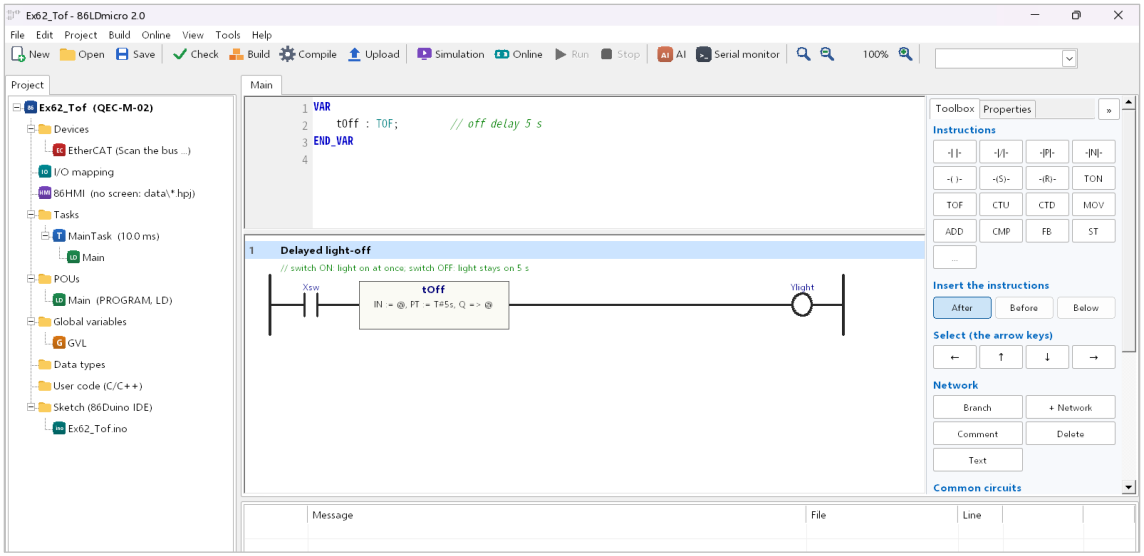


图 6-9　6.2 のラダー図：Xsw → TOF ファンクションブロック tOff（PT := T#5s）→ Ylight

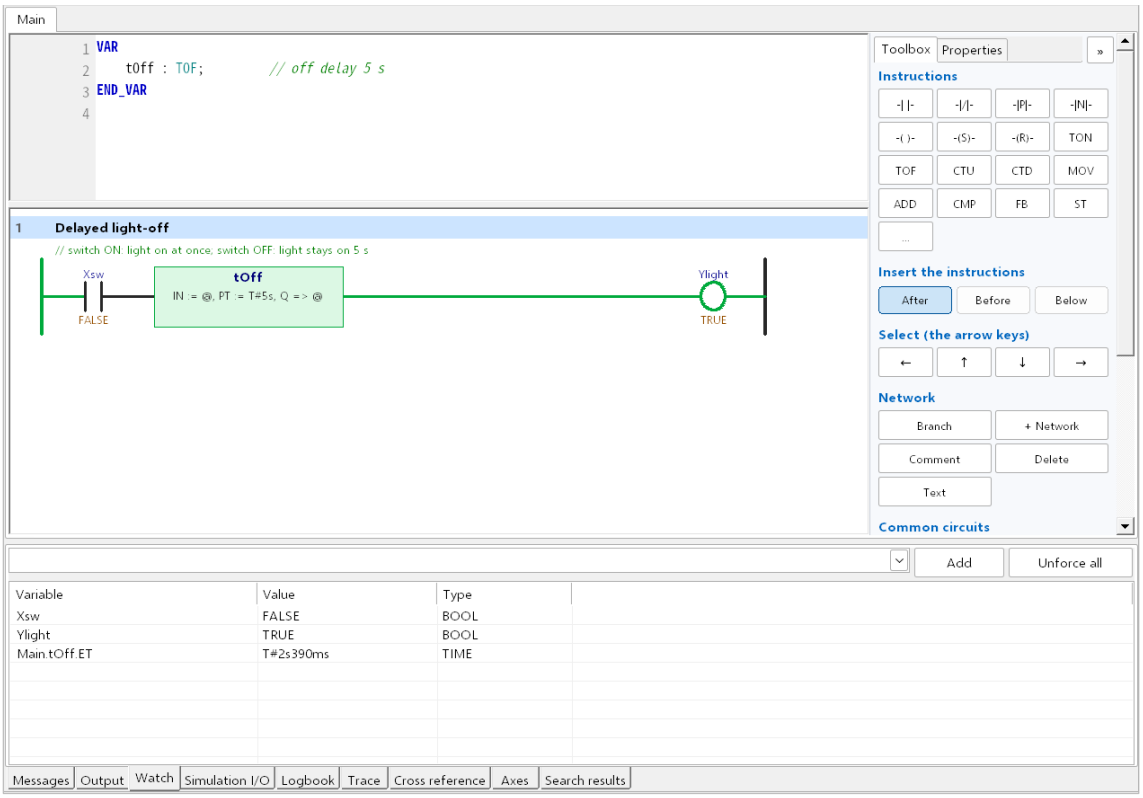


图 6-10　6.2 のシミュレーション：スイッチは OFF、TOF が計時中で、Ylight はまだ TRUE

*ヒント：バリエーション 1：1 回押すと N 秒点灯（自動消灯ボタン）：`tOff : TOF` を `tOn : TP` に替えて、`NO(Xsw) CALL(tOn, IN := @, PT := T#5s, Q => @) OUT(Ylight)` と書きます。TP は入力の立ち上がりで、ボタンを押す長さに関係なく、固定 5 秒のパルスを出力します。

*バリエーション 2：1.x ブロック：Toolbox の **TOF**（F）をクリックして `NO(Xsw) TOF(t1, T#5s) OUT(Ylight)` と書き、VAR で `t1 : DINT;` を宣言します（PLC 周期で計時、7.4 節）。

6.3 梱包カウンター（CTU＋比較）

センサーが製品を 1 つ検出するたびに 1 を加算し、10 個たまったら箱詰め機構を 1 秒間駆動して、カウントを自動的にゼロに戻して計数を続けます。この例では、比較命令でカウント値を読み取り、内部フラグ `Rdone` で後続のネットワークを動かす方法を示します。

シナリオ：生産ラインのカウントで、1 箱 10 個、自動箱詰めと自動リセットを繰り返します。sketch 名：`Ex63_Counter`。

変数	型	I/O マッピング	用途
`Xsensor`	BOOL	`LocalIO.X00`	製品センサー
`Ybox`	BOOL	`LocalIO.Y00`	箱詰め機構
`Count`	INT	—	現在のカウント（Watch や 86HMI で表示可能）
`Rdone`	BOOL	—	10 個到達のフラグ
`cBox`	CTU（Main の VAR）	—	製品カウンター
`tBox`	TP（Main の VAR）	—	箱詰め 1 秒パルス

プログラム（Main、.lad テキスト）

```
PROGRAM Main
VAR
    cBox : CTU;          // 製品カウント
    tBox : TP;           // 箱詰め 1 秒パルス
END_VAR
NETWORK '製品のカウント'
    // センサーの立ち上がりごとに 1 加算、Rdone が TRUE のときリセット
    NO(Xsensor) CALL(cBox, CU := @, R := Rdone, PV := 10, CV => Count)
NETWORK '比較：10 個到達'
    CMP(Count >= 10) OUT(Rdone)
NETWORK '箱詰め 1 秒'
    NO(Rdone) CALL(tBox, IN := @, PT := T#1s, Q => @) OUT(Ybox)
END_PROGRAM
```

詳細手順

#	操作
1	sketch `Ex63_Counter` と新しいプロジェクトを作成し、GVL で `Xsensor`、`Ybox`、`Rdone`（BOOL）と `Count`（INT）を宣言します。
2	`Main` の宣言エリアに `cBox : CTU;` と `tBox : TP;` を入力します。
3	ネットワーク 1：**C** → `Xsensor`、**K** →「Element」に `CALL(cBox, CU := @, R := Rdone, PV := 10, CV => Count)` と入力。
4	Ctrl+Enter でネットワーク 2 を追加：**E**（条件接点 CMP）→「Condition」に `Count >= 10` と入力、**L** → `Rdone`。

#	操作
5	Ctrl+Enter でネットワーク 3 を追加：**C** → `Rdone`、**K** → `CALL(tBox, IN := @, PT := T#1s, Q => @)`、**L** → `Ybox`。
6	「I/O mapping」：`Xsensor` → `LocalIO.X00`、`Ybox` → `LocalIO.Y00`（`Count`、`Rdone`、`cBox`、`tBox` はマッピング不要）。
7	F5 でシミュレーション：「Simulation I/O」で `LocalIO.X00` を繰り返しダブルクリック（ON、OFF で製品 1 個）し、合計 10 回行います。「Watch」に `Count` を追加してカウントを確認します。
8	F8 でアップロード。

動作説明

1. `Xsensor` の立ち上がりごとに `cBox` が 1 加算され、`Count` もそれに合わせて更新されます。
2. `Count` が 10 になると、ネットワーク 2 の条件 `Count >= 10` が成立し、`Rdone` = TRUE になります。
3. ネットワーク 3：`Rdone` の立ち上がりで `tBox`（TP）が起動し、`Ybox` = TRUE が固定 1 秒間続きます。
4. 次のスキャンで、ネットワーク 1 の `R := Rdone` がカウントをゼロに戻し（`Count` = 0）、ネットワーク 2 の `Rdone` は FALSE に戻ります。`Ybox` は TP によって 1 秒が終わるまで維持されます。
5. 次の箱のカウントを繰り返します。

期待される結果：パルスを 10 回受け取るごとに Ybox が 1 秒間通電し、カウントはゼロに戻って再開します。前提条件：箱詰めの 1 秒間に次の 10 個がそろわないこと（TP はパルス中に新しいトリガーを受け付けないため、次の箱の Rdone は無視されます）。

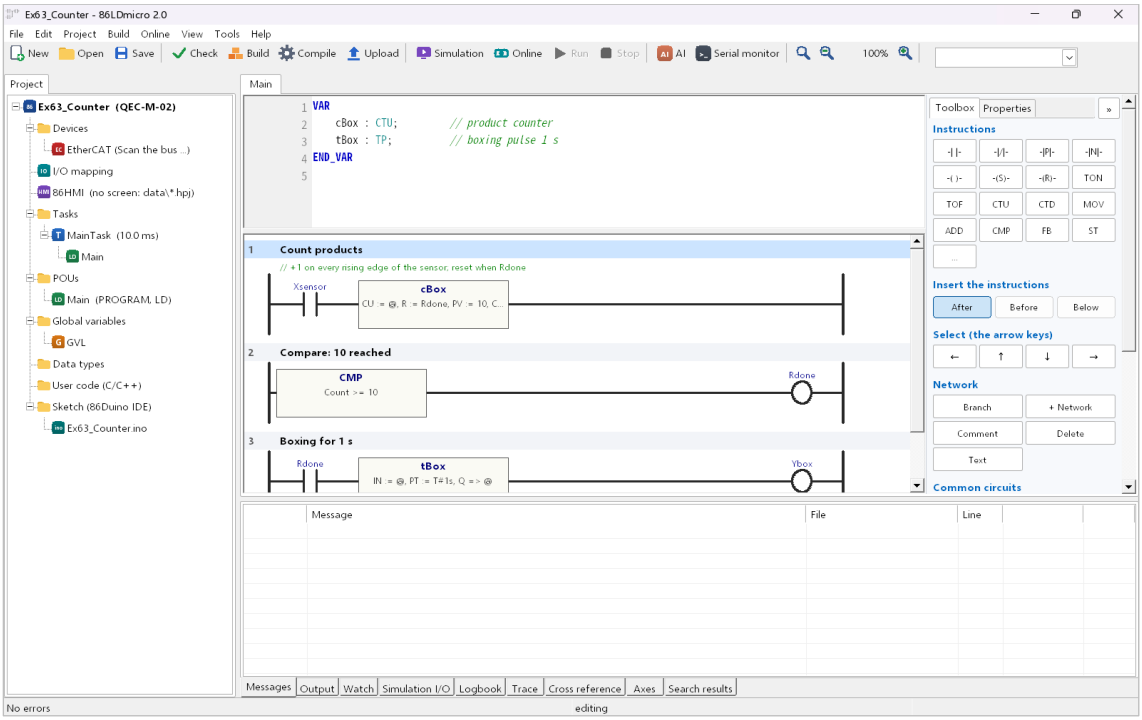


図 6-11　6.3 のラダー図：CTU のカウント、CMP の比較、TP の箱詰めパルスの 3 つのネットワーク

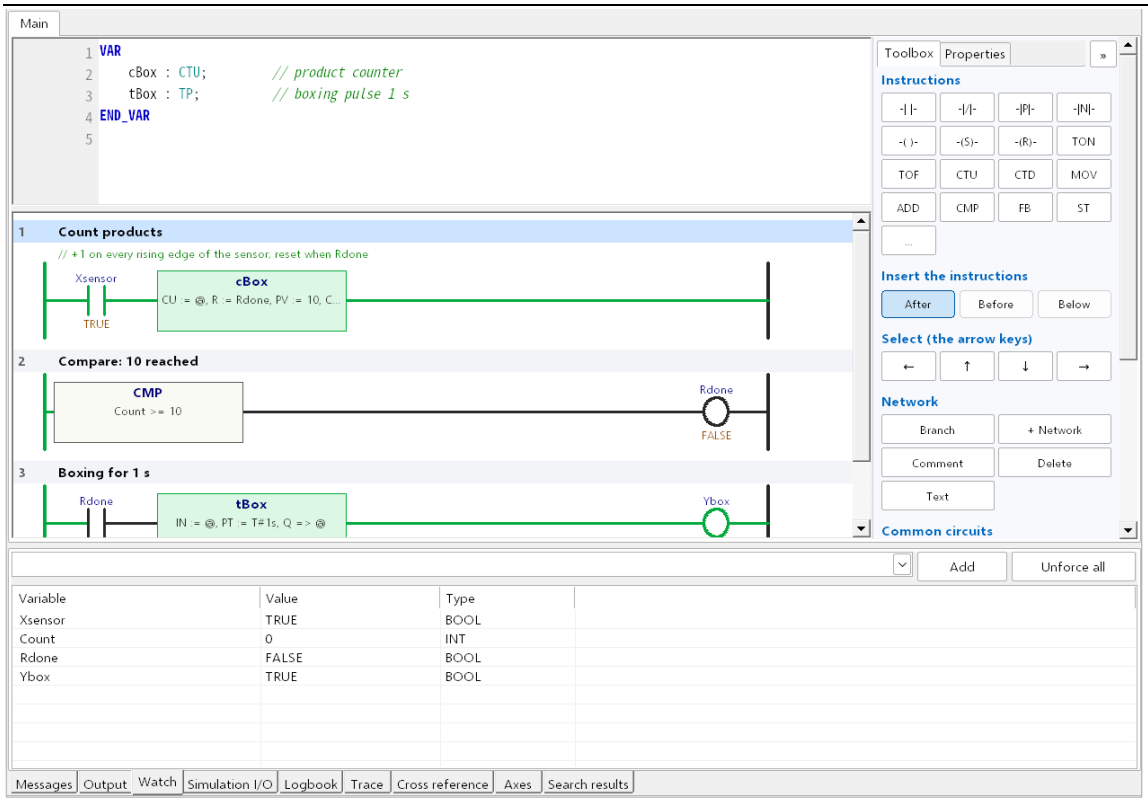


図 6-12　6.3 のシミュレーション：10 個目の製品の後に Ybox が通電し、Count はゼロに戻っている

*ヒント：比較ではファンクションブロックの出力を直接読み取ることもできます：`CMP(cBox.CV >= 10)`、または接点 `NO(cBox.Q)`（CV が PV に達すると TRUE）。

センサー信号にチャタリングがある場合は、I/O マッピングで「Filter (ms)」を設定するか、Common circuits の「Debounce an input」を使用します。

***注意：製品が非常に速く流れてくる場合は、ビジーインターロック（例：箱詰めの Ybox の間はコンベヤーを停止）を追加するか、箱詰め待ちの数量を累積する書き方に変更して、1 箱分の取りこぼしを防いでください。**

カウントがゼロに戻るスキャンにちょうどセンサーの立ち上がりがあると、その製品はカウントされません（CTU の R が優先）。すべての製品を確実にカウントする必要がある場合は、上記の累積する書き方を使用してください。

6.4 流れる表示灯（交互点滅）

2 つのランプを 500 ms で交互に点滅させます。一方が点灯しているときはもう一方が消灯し、500 ms ごとに同時に切り替わります。この例では LDmicro 1.x 互換のサイクルタイマー **TCY** で方形波を作って内部ビット `Rstate` に格納し、2 つのネットワークでそれぞれ a 接点と b 接点を使って 2 つのランプを駆動します。

シナリオ：スイッチが ON のとき、2 つのランプが逆相で交互に点滅し、スイッチが OFF のときは両方とも消灯します。sketch 名：`Ex64_Blink`。

変数	型	I/O マッピング	用途
`Xrun`	BOOL	`LocalIO.X00`	点滅スイッチ
`Rstate`	BOOL	—	方形波の状態（ネットワーク間で共有）
`Yled1`	BOOL	`LocalIO.Y00`	状態が `Rstate` と同相
`Yled2`	BOOL	`LocalIO.Y01`	状態が `Rstate` と逆相
`tClock`	DINT（Main の VAR）	—	TCY のタイマー変数 [1.x]

プログラム（Main、.lad テキスト）

```
PROGRAM Main
VAR
    tClock : DINT;      // TCY タイマー [1.x]
END_VAR
NETWORK '500 ms 方形波'
    NO(Xrun) TCY(tClock, T#500ms) OUT(Rstate)
NETWORK 'ランプ 1：方形波と同相'
    NO(Xrun) NO(Rstate) OUT(Yled1)
NETWORK 'ランプ 2：方形波と逆相'
    NO(Xrun) NC(Rstate) OUT(Yled2)
END_PROGRAM
```

詳細手順

#	操作
1	sketch `Ex64_Blink` と新しいプロジェクトを作成し、GVL で `Xrun`、`Rstate`、`Yled1`、`Yled2`（BOOL）を宣言します。
2	`Main` の宣言エリアに `tClock : DINT;` を入力します。
3	ネットワーク 1：**C** → `Xrun`、サイドパネルの「Instructions」の **...** →「Boxes」→ **TCY** を選び、「timer」に `tClock`、「delay」に `T#500ms` を入力（「adjust」は空欄）→ OK、**L** → `Rstate`。
4	Ctrl+Enter でネットワーク 2 を追加：**C** → `Xrun`、**C** → `Rstate`、**L** → `Yled1`。
5	Ctrl+Enter でネットワーク 3 を追加：**C** → `Xrun`、**N** → `Rstate`、**L** → `Yled2`。
6	「I/O mapping」：`Xrun` → `LocalIO.X00`、`Yled1` → `LocalIO.Y00`、`Yled2` → `LocalIO.Y01`。
7	F5 でシミュレーション：`LocalIO.X00` をダブルクリックして ON にし、`LocalIO.Y00`、`Y01` が 500 ms ごとに交互に ON／OFF することを確認します。

#	操作
8	F8 でアップロード。

期待される結果：`Xrun` = 1 のとき、`Yled1` と `Yled2` は互いに逆の状態で約 500 ms ごとに切り替わります（周期は約 1 秒）。`Xrun` = 0 のときは両方とも 0 です。

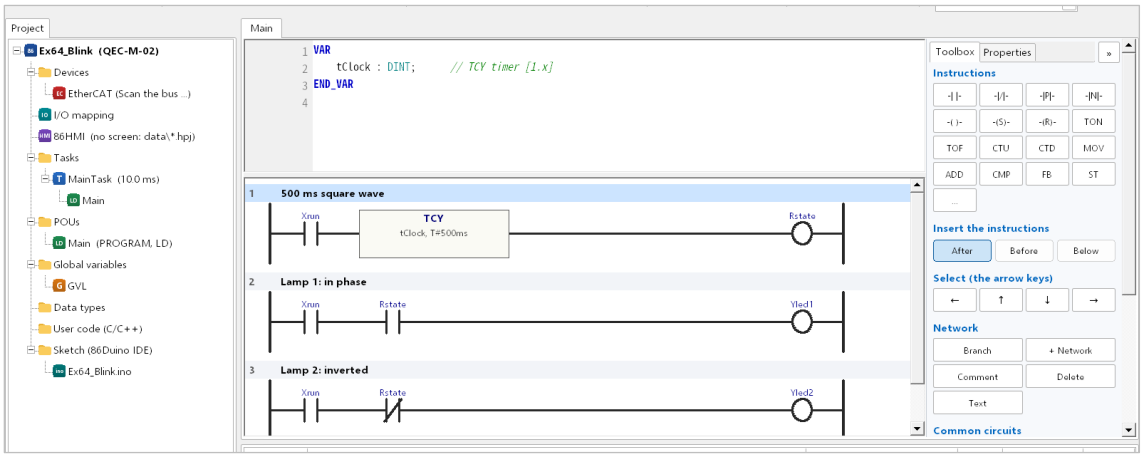


図 6-13　6.4 のラダー図：TCY が方形波 Rstate を作り、2 つのネットワークが NO／NC で 2 つのランプを駆動

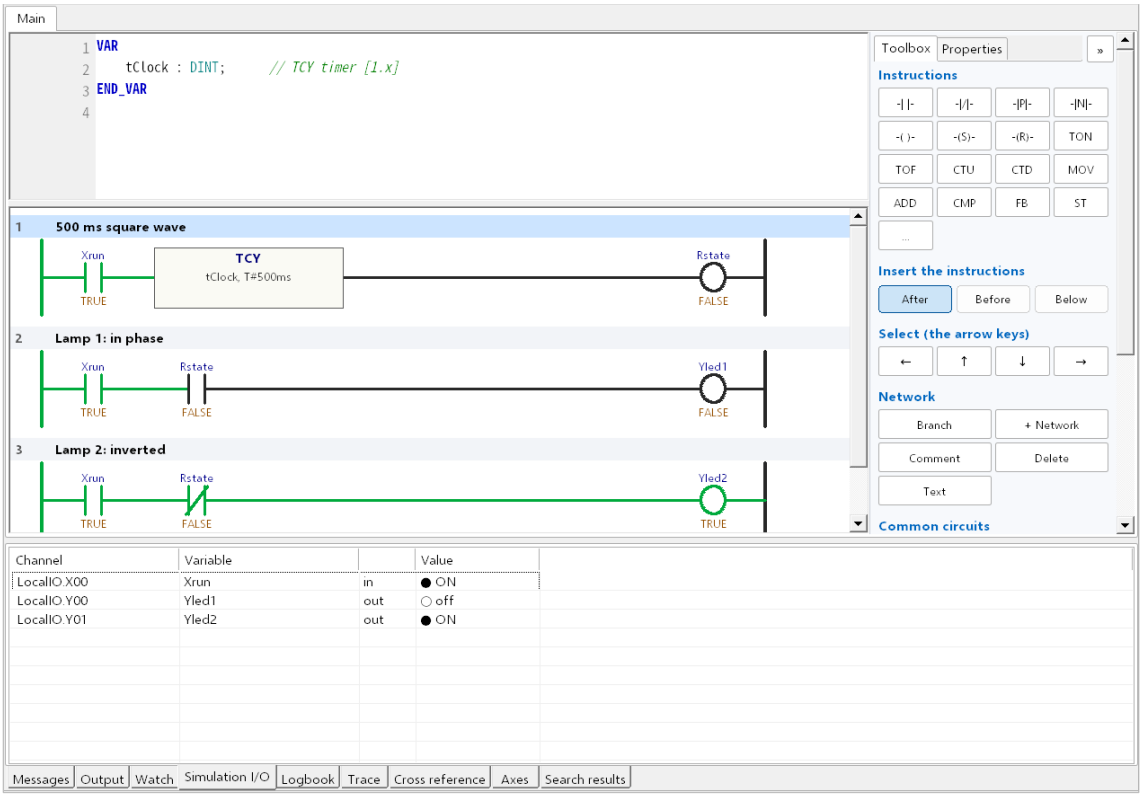


図 6-14　6.4 のシミュレーション：Yled1 と Yled2 が交互に点灯・消灯

*ヒント：TCY などの 1.x タイマーは PLC 周期でカウントします。タスクの周期が 10 ms の場合、`T#500ms` は 50 周期です。遅延がタスクの周期より短い場合は計時できません（チェックで警告が表示されます）。1.x 命令を使わない方法もあります。サイドパネルの「Common circuits」の **Blink while enabled** では点灯時間と消灯時間を個別に設定できます。また、ST で TON を使って点滅用のファンクションブロックを自作することもできます（8.3 節）。

6.5 ST：注水制御

ストラクチャードテキスト（ST）で、ステップのある注水制御を書きます。自動スイッチが入っていて液位が 20 % 未満のときにバルブを開いて注水し、80 % でバルブを閉じます。注水が 30 秒を超えても 80 % に達しない場合は異常とみなして警報を出力します。ST は条件が多い、計算がある、ステップがあるロジックに適しています。

シナリオ：水槽の自動給水で、タイムアウト警報付きです。自動スイッチを切ると停止し、警報もリセットされます。sketch 名：`Ex65_Fill`。

変数	型	I/O マッピング	用途
`Xauto`	BOOL	`LocalIO.X00`	自動注水スイッチ
`Level`	REAL	アナログ入力（ハードウェアによる、4.6 節）	液位（%）
`Yvalve`	BOOL	`LocalIO.Y00`	注水バルブ
`Yalarm`	BOOL	`LocalIO.Y01`	注水タイムアウト警報
`tFill`	TON（Filling の VAR）	—	注水タイムアウト 30 秒
`step`	INT（Filling の VAR）	—	0 待機、1 注水、2 タイムアウト

プログラム（Filling、ST）

```
PROGRAM Filling
VAR
    tFill : TON;        // 注水タイムアウト
    step  : INT;        // 0 待機、1 注水、2 タイムアウト
END_VAR
CASE step OF
    0:  (* 待機：自動で液位が 20 % 未満なら注水開始 *)
        IF Xauto AND Level < 20.0 THEN
            step := 1;
        END_IF;
    1:  (* 注水：80 % で停止、30 秒を超えたら異常 *)
        IF Level >= 80.0 OR NOT Xauto THEN
            step := 0;
        ELSIF tFill.Q THEN
            step := 2;
        END_IF;
    2:  (* タイムアウト：自動スイッチを切るとリセット *)
        IF NOT Xauto THEN
            step := 0;
        END_IF;
END_CASE;
tFill(IN := (step = 1), PT := T#30s);   // 毎スキャン呼び出す
Yvalve := (step = 1);
Yalarm := (step = 2);
END_PROGRAM
```

詳細手順

#	操作
1	sketch `Ex65_Fill` と新しいプロジェクトを作成し、GVL で `Xauto`、`Yvalve`、`Yalarm`（BOOL）と `Level`（REAL）を宣言します。
2	「Project > Add a PROGRAM (ST)...」を選び、名前に `Filling` と入力 → OK。新しい PROGRAM は最初のタスク MainTask に自動的に追加され、ST エディターが開きます。
3	上記のプログラムを入力します（変数名の最初の数文字を入力すると補完候補が表示されます）。**Ctrl+T** で自動整形し、Ctrl+S で保存します。空の `Main` はそのまま残してかまいません。
4	「I/O mapping」：`Xauto` → `LocalIO.X00`、`Yvalve` → `LocalIO.Y00`、`Yalarm` → `LocalIO.Y01`。`Level` はシミュレーションで直接書き込めるよう、いったんマッピングしないでおきます。実機では実際のアナログ入力チャンネルにマッピングし、「Scaling」を設定します（例：生の値 0〜4095 を 0〜100 に変換）。
5	Shift+F7 でチェック。F5 でシミュレーション：「Simulation I/O」で `LocalIO.X00` をダブルクリックして ON にし、「Watch」に `Level`、`Filling.step`、`Yvalve` を追加します。`Level` をダブルクリックして `10` を入力 → `Yvalve` = TRUE、`85` を入力 → `Yvalve` = FALSE。
6	次に `Level` を `10` に変えて 30 秒待つと、`Filling.step` = 2、`Yalarm` = TRUE になります。`LocalIO.X00` を OFF にすると警報がリセットされます。
7	F8 でアップロード。

期待される結果：自動スイッチが ON で液位が 20 % 未満のとき、注水バルブが開き、液位が 80 % に達すると閉じます。注水が 30 秒続いても 80 % に達しない場合は、バルブを閉じて警報を出力し、自動スイッチが OFF になるまで続けます。

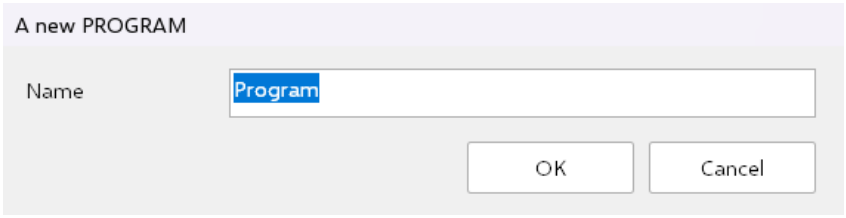


図 6-15　「Project > Add a PROGRAM (ST)...」：プログラム名を入力

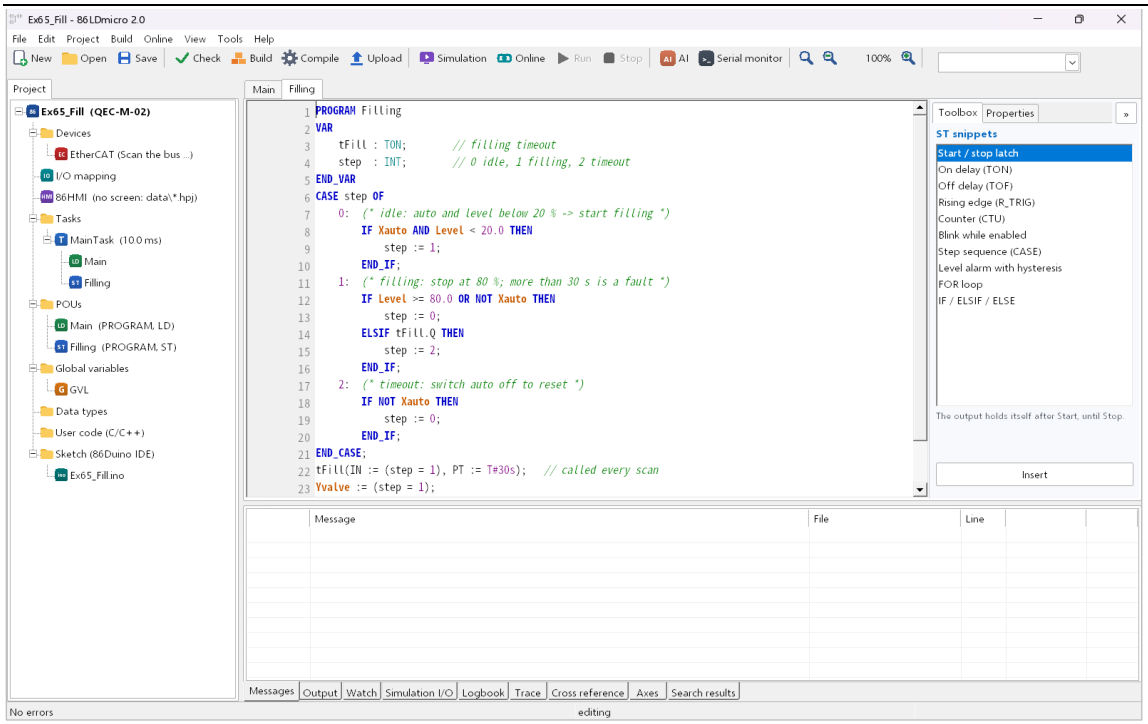


図 6-16　6.5 の ST プログラム Filling：CASE によるステップと TON のタイムアウト

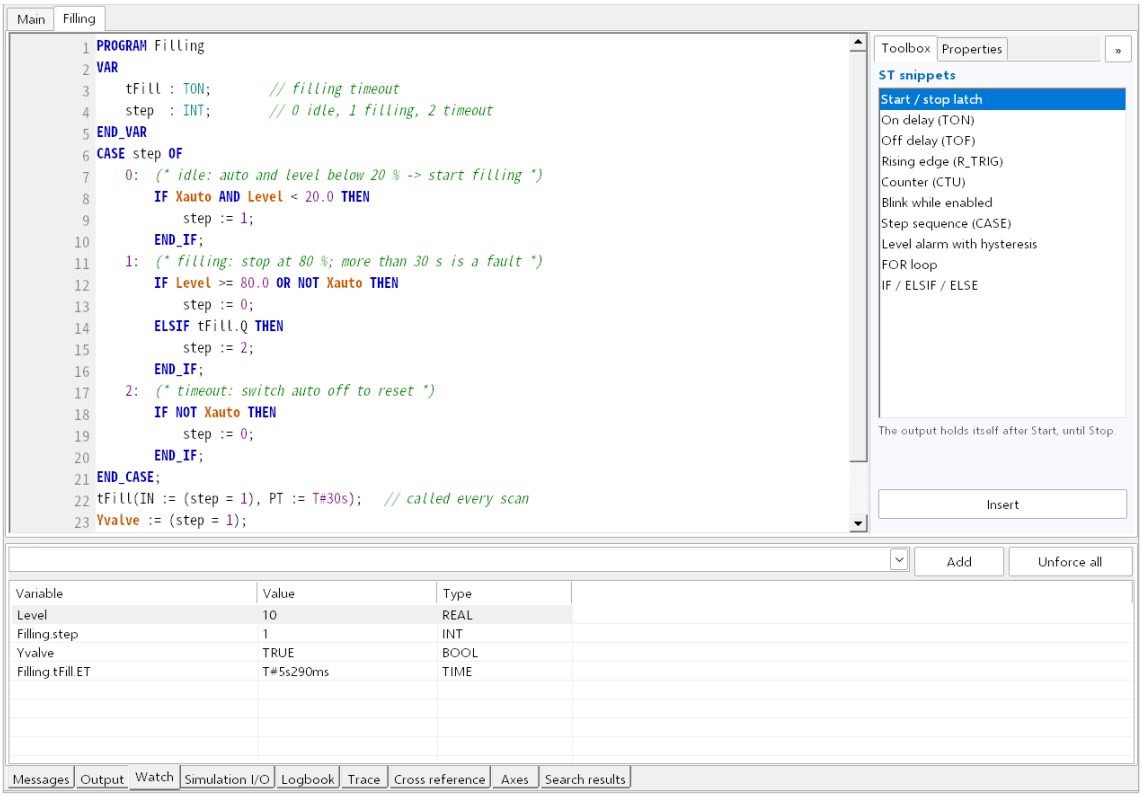


図 6-17　6.5 のシミュレーション：Level = 10、step = 1、注水バルブが開いている

*ヒント：タイマーが時間を積算するには、ファンクションブロックのインスタンスを毎スキャン呼び出す必要があります。この例では `tFill(...)` を CASE の外に置き、`IN := (step = 1)` で呼び出しています。ステップ 1 を離れると IN が FALSE になり、計時は自動的にゼロに戻ります。ST の構文は 8.2 節を参照してください。

6.6　SFC：シーケンス制御

シーケンシャルファンクションチャート（SFC）で「給水 → 攪拌 → 排水」の固定の流れを記述します。SFC は**ステップ**と**遷移条件**で流れを表し、アクティブなステップだけが動作を実行するため、マシンの自動運転の流れに適しています。

シナリオ：起動を押すと、給水バルブが開き、満水位になるか 10 秒経過するまで給水します。続いて 5 秒間攪拌し、3 秒間排水します。完了すると待機に戻り、次の起動を待ちます。sketch 名：`Ex66_Seq`。

変数	型	I/O マッピング	用途
`Xstart`	BOOL	`LocalIO.X00`	起動ボタン
`Xfull`	BOOL	`LocalIO.X01`	満水位スイッチ
`Yvalve`	BOOL	`LocalIO.Y00`	給水バルブ
`Ymixer`	BOOL	`LocalIO.Y01`	攪拌モーター
`Ydrain`	BOOL	`LocalIO.Y02`	排水バルブ
`cycles`	INT（Seq の VAR）	—	完了回数

プログラム（Seq、.sfc テキスト）

```
PROGRAM Seq
VAR
    cycles : INT;       // 完了回数
END_VAR
SFC
STEP Idle INITIAL
    Ydrain := FALSE;
END_STEP
TRANSITION Xstart
STEP Fill
    Yvalve := TRUE;
END_STEP
TRANSITION Xfull OR Fill.T >= T#10s
STEP Mix
    Yvalve := FALSE;
    Ymixer := TRUE;
END_STEP
TRANSITION Mix.T >= T#5s
STEP Drain
    Ymixer := FALSE;
    Ydrain := TRUE;
END_STEP
TRANSITION Drain.T >= T#3s
STEP Done
    cycles := cycles + 1;
END_STEP
TRANSITION TRUE
JUMP Idle
END_SFC
END_PROGRAM
```

詳細手順

#	操作
1	sketch `Ex66_Seq` と新しいプロジェクトを作成し、GVL で `Xstart`、`Xfull`、`Yvalve`、`Ymixer`、`Ydrain`（BOOL）を宣言します。
2	「Project > Add a PROGRAM (SFC, a sequence)...」を選び、名前を `Seq` → OK。新しいチャートには、すでに `Init` → `TRUE` → `Step1` → `Step1.T >= T#1s` → `Init` へジャンプ、があります。上部の宣言エリアに `cycles : INT;` を入力します。
3	`Init` をダブルクリック：名前を `Idle` に変え、「Action」に `Ydrain := FALSE;` を入力 → OK（このステップへの JUMP も一緒に名前が変わります）。
4	`Idle` の後の遷移条件をダブルクリック：「Condition (ST)」に `Xstart` を入力します。
5	ステップ `Step1` をダブルクリック：名前 `Fill`、アクション `Yvalve := TRUE;`。その後の遷移条件をダブルクリックし、`Xfull OR Fill.T >= T#10s` に変更します。
6	この遷移条件を選択し、サイドパネルの「Chart」の **+ Step**（Ins）をクリックすると、その後ろにステップと `TRUE` の遷移条件が追加されます。新しいステップをダブルクリック：名前 `Mix`、アクション `Yvalve := FALSE; Ymixer := TRUE;`。遷移条件を `Mix.T >= T#5s` に変更します。
7	同様に最後の遷移条件を選択して **+ Step** をクリック：ステップ `Drain`（アクション `Ymixer := FALSE; Ydrain := TRUE;`）、遷移条件 `Drain.T >= T#3s`。
8	もう一度 **+ Step** をクリック：ステップ `Done`（アクション `cycles := cycles + 1;`）、遷移条件は `TRUE` のままにします。その後ろは既存の「Idle へジャンプ」です。
9	「I/O mapping」：変数表に従って X00、X01、Y00〜Y02 をマッピングします。
10	F5 でシミュレーション：`LocalIO.X00` を 2 回ダブルクリック（ON、OFF）し、アクティブなステップ（緑色）が順に移動していく様子と、右側に表示されるアクティブ時間を確認します。Fill の間に `LocalIO.X01` をダブルクリックして満水位をシミュレートすることもできます。
11	F8 でアップロード。

期待される結果：X00 を 1 回押すと：`Yvalve` ON（最長 10 秒、X01 が ON になると早めに終了）→ `Ymixer` が 5 秒間 ON → `Ydrain` が 3 秒間 ON → `Idle` に戻り、`Ydrain` OFF、`cycles` に 1 加算。

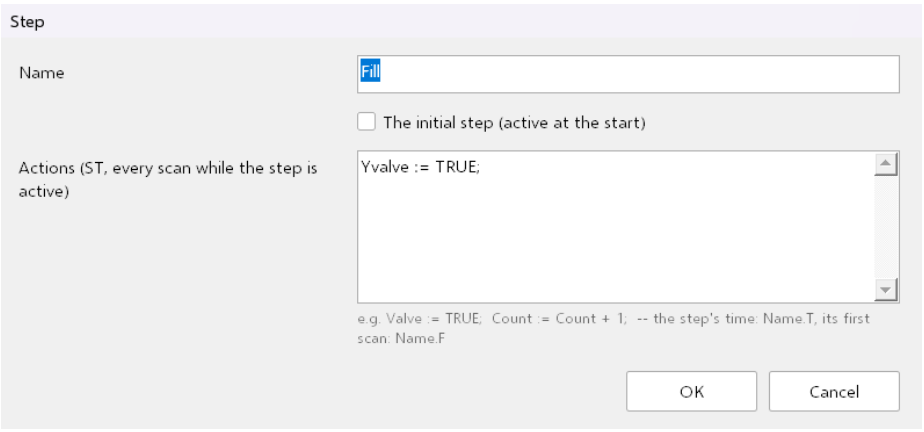


図 6-18　ステップの設定ウィンドウ：名前、初期ステップ、アクション（ST）



図 6-19　遷移条件の設定ウィンドウ：条件（ST）とコメント

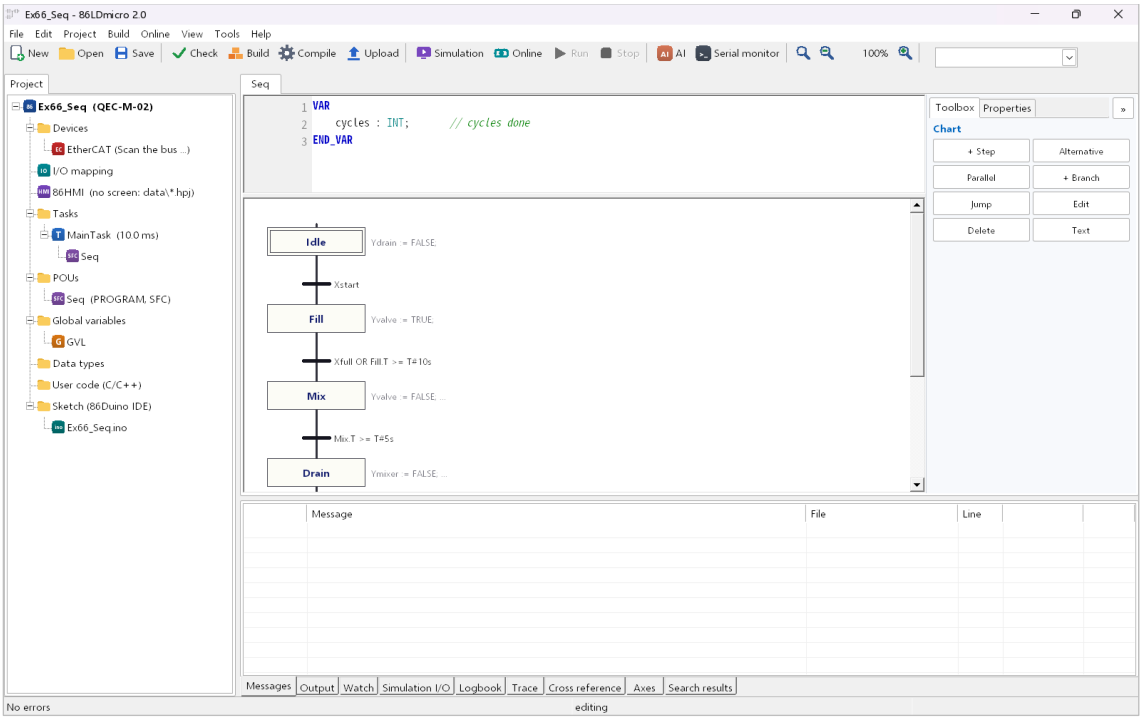


図 6-20　6.6 の SFC チャート：Idle → Fill → Mix → Drain → Done → Idle へジャンプ

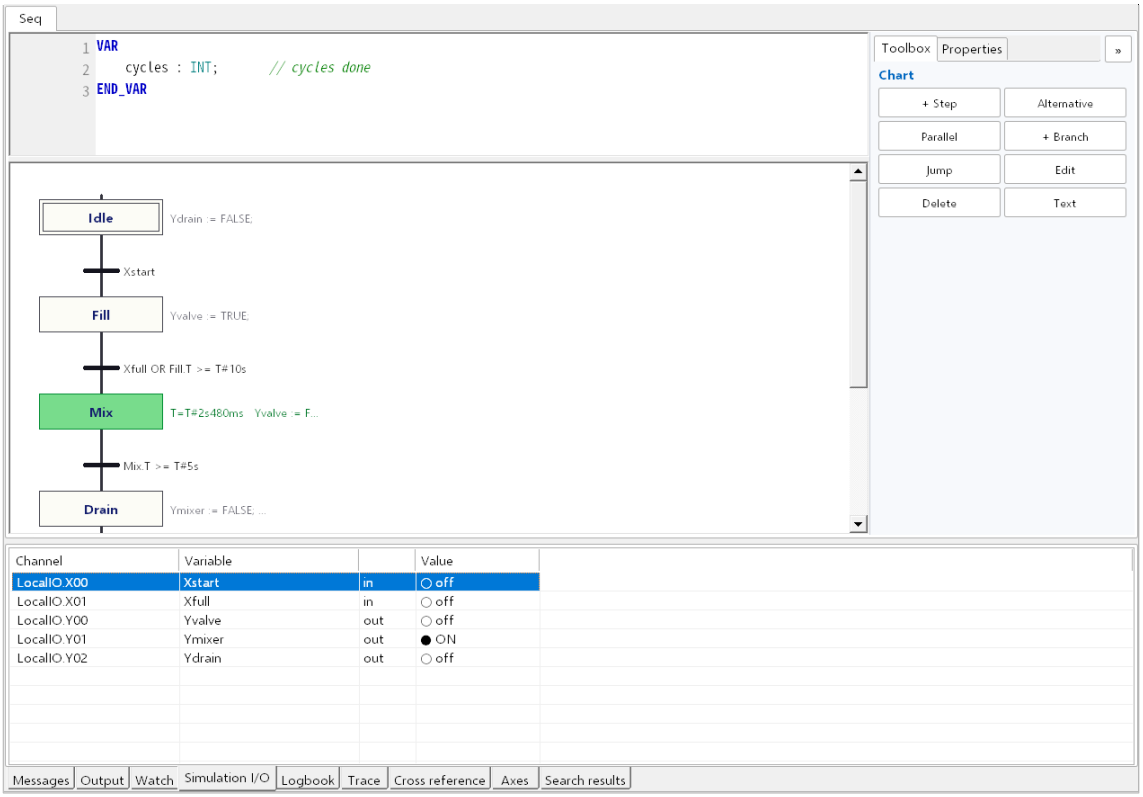


図 6-21　6.6 のシミュレーション：Mix ステップがアクティブ（緑色）、右側にアクティブ時間を表示

*ヒント：SFC のステップで設定した出力は**保持**されます。オフにするには、後のステップで `:= FALSE` と書きます（この例では Mix で給水バルブを閉じ、Drain で撹拌を止め、Idle で排水バルブを閉じています）。「Chart」の **Text**（Ctrl+T）をクリックして上記の .sfc テキストを直接入力し、**Back to the diagram** をクリックすることもできます。分岐、ジャンプ、実行のセマンティクスについては 8.4〜8.6 節を参照してください。

6.7 C/C++：Serial1 でメッセージを送信

ボタンが押されたときに、C/C++ のユーザーコードが Serial1（コントローラーの COM1）から 1 行のテキストを送信します。送信先は、たとえばバーコードプリンター、上位コンピューター、別の機器などです。ラダー図がボタンの判定を、C/C++ が通信を担当し、両者は GVL の変数を共有します。

シナリオ：ボタンを 1 回押すごとに、Serial1 から 9600 bps で Button pressed, count = N を送信します。

sketch 名：Ex67_Serial。

変数	型	I/O マッピング	用途
Xbtn	BOOL	LocalIO.X00	送信ボタン
Nmsg	DINT	—	ボタンを押した回数（メッセージ番号）、ラダー図だけが書き込む

プログラム 1（Main、.lad テキスト）：ボタンの立ち上がりで Nmsg に 1 を加算します。

```
PROGRAM Main
VAR
END_VAR
NETWORK '1 回押す：メッセージ番号に 1 加算'
    PT(Xbtn) ADD(Nmsg, Nmsg, 1)
END_PROGRAM
```

プログラム 2（ユーザーコード SendMsg.cpp86）：Nmsg が変化したら 1 行のメッセージを送信します。

```
// SendMsg：Nmsg が変化したら Serial1 から 1 行のメッセージを送信

void plc86_user_setup(void) {
    Serial1.begin(9600);               // COM1、9600 bps
}

// loop() の各周回：ブロックしない処理だけを行う（delay() は使わない）
void plc86_user_loop(void) {
    static int32_t sent = 0;           // 送信済みの番号
    int32_t n = GVL(Nmsg);
    if(n != sent) {
        sent = n;
        Serial1.print("Button pressed, count = ");
        Serial1.println(n);
    }
}
```

詳細手順

#	操作
1	sketch Ex67_Serial と新しいプロジェクトを作成し、GVL で Xbtn : BOOL; と Nmsg : DINT; を宣言します。
2	Main を開き、ネットワーク 1：**P**（立ち上がり接点 PT）→ Xbtn、**A**（ADD）→「dest」に Nmsg、「a」に Nmsg、「b」に 1 → OK。

#	操作
3	「Project > Add a user code file (C/C++)...」を選び、名前を `SendMsg` → OK。テンプレートには各フック関数がすでに記載されています。`plc86_user_setup()` と `plc86_user_loop()` を上記の内容に変更し、使わない `plc86_user_before()`、`plc86_user_after()` を削除して、Ctrl+S で保存します。
4	「I/O mapping」：`Xbtn` → `LocalIO.X00`。
5	Shift+F7 でチェック。F5 でシミュレーション：「Simulation I/O」で `LocalIO.X00` を繰り返しダブルクリックすると、「Watch」の `Nmsg` が 1 回押すごとに 1 増えます。
6	COM1 をパソコンに接続し（例：USB－RS-232／485 変換アダプター）、F8 でアップロードします。
7	下部の「Output」で **Serial monitor** に切り替え、アダプターのシリアルポートとボーレート `9600` を選んで **Connect** をクリックします。X00 を 1 回押すごとに 1 行のメッセージを受信します。

期待される結果：X00 を 1 回押すごとに `Nmsg` が 1 増え、`Serial1` から 1 行 `Button pressed, count = 1`、`= 2...` が送信されます。

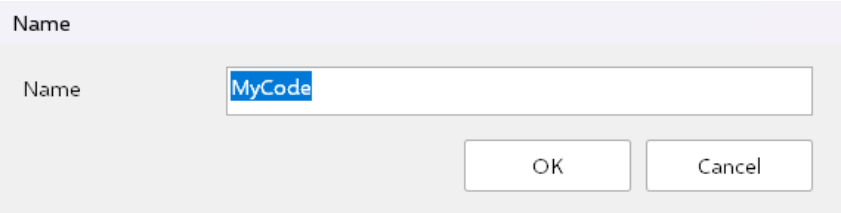


図 6-22　「Project > Add a user code file (C/C++)...」：ファイル名を入力

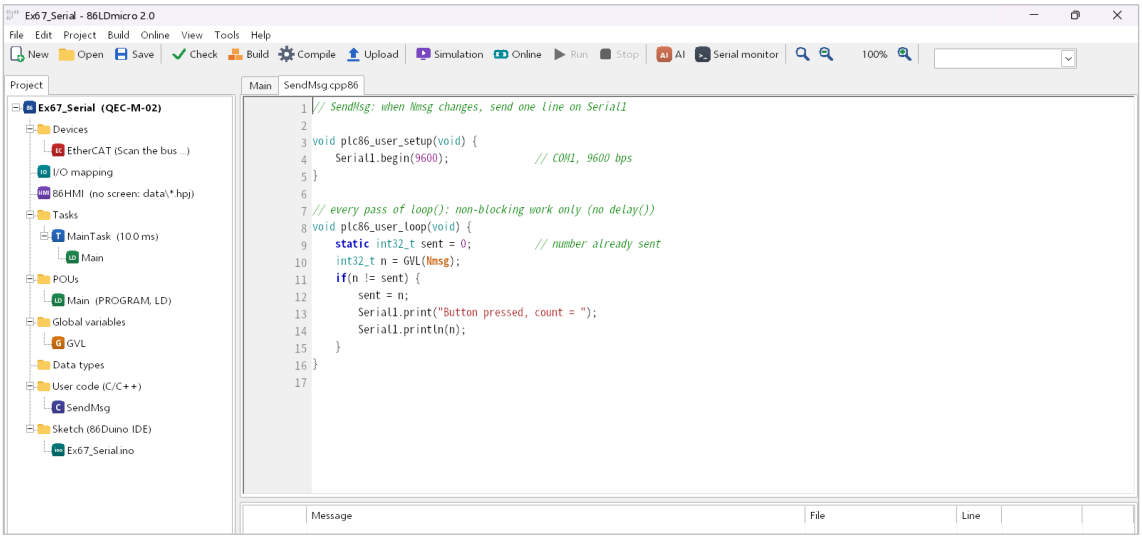


図 6-23　6.7 のユーザーコード SendMsg.cpp86：plc86_user_setup() と plc86_user_loop()

***注意：**USB の `Serial` はアップロードとオンラインモニターに使用されます。外部機器との通信には `Serial1`〜`Serial3` を使用してください。

C/C++ では、PLC の変数は `GVL(Nmsg)` と書く必要があり、名前を直接書くことはできません。

`plc86_user_loop()` は `loop()` のたびに実行されるので、`delay()` は使用しないでください。

この例ではラダー図が `Nmsg` に書き込み、C/C++ は読み取るだけなので、6.0 の原則 ③ に従っています。フック関数とその他の書き方については第 14 章を参照してください。

第 7 章　ラダー図命令リファレンス

86LDmicro 2.0 のラダー図には全部で 90 の命令があります（コマンドライン `86ldmicro-cli instructions` で一覧表示されるものと同じ）。本章では用途別に 10 のグループに分けて 1 つずつ説明します。各節には概要、メニューまたはパラメーターウィンドウの画面、および命令表が含まれます：**命令**（.lad での名前）、**.lad の書き方**（例。テキストモードにそのまま貼り付け可能）、**キー**（要素またはネットワークを選択した後に直接押すキー）、**説明**。

- **[1.x]** の付いた命令は LDmicro 1.x 互換の命令です。動作は 1.1 と同じで、タイマーは PLC 周期でカウントします。新しいプログラムでは IEC ファンクションブロック（7.8 節）の使用をお勧めします。
- パラメーターには変数、定数、ST の式を指定できます。時間は `T#500ms`、`T#2s` などの IEC 時間定数で書いてください。
- Toolbox にボタンがない命令は、サイドパネルの「Instructions」の **...** をクリックし、「Contacts」、「Coils」、「Boxes」、「Program flow」、「Function blocks」のサブメニューから選択します（メニューには各命令の説明が表示されます）。
- どの命令も、挿入するとパラメーターウィンドウが開き、パラメーターごとに 1 つの欄があります。「variable」の欄では、宣言済みの変数をドロップダウンから選択できます。

7.1 編集操作

ラダー図タブの上部は宣言エリア（`VAR ... END_VAR`）、中央はネットワーク、右側のサイドパネルは Toolbox です。この節では、要素とネットワークの挿入、選択、移動、削除の方法、およびテキストモードと Common circuits について説明します。

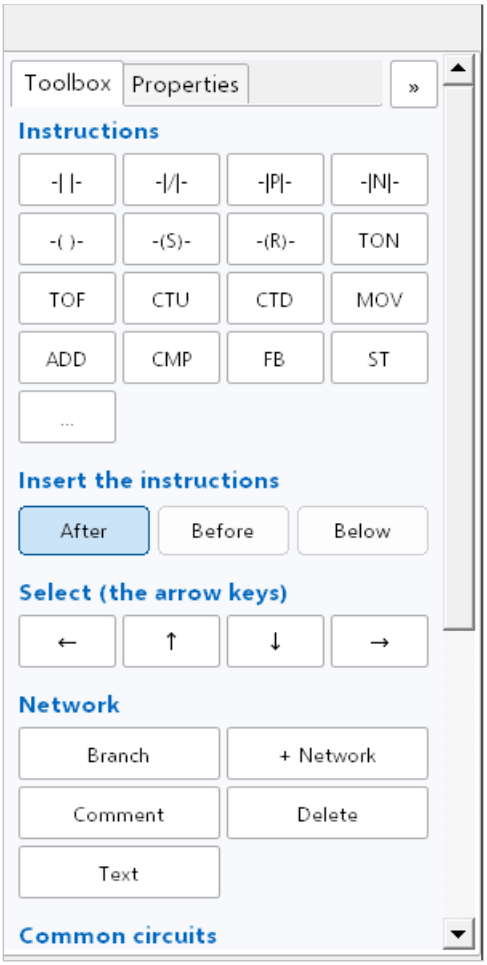


図 7-1　ラダー図のサイドパネルの Toolbox：Instructions、Insert the instructions、Select (the arrow keys)、Network、Common circuits

Toolbox の区画	内容
Instructions	よく使う要素のボタン。マウスを重ねると命令の説明とキーが表示されます。**...** はすべての命令をカテゴリー別に一覧表示します
Insert the instructions	**After**：選択した要素の後に挿入（直列。ネットワーク全体を選択しているときはコイルの前）。**Before**：選択した要素の前に挿入。**Below**：選択した要素の下に並列分岐を追加
Select (the arrow keys)	←→：左右の要素。↑↓：上下の要素、または前／次のネットワーク
Network	**Branch**（選択した要素の下に並列分岐を追加。B キーと同じ）、**+ Network**、**Comment**（ネットワークのタイトルとコメント）、**Delete**、**Text**（POU 全体を .lad テキストで編集。Ctrl+T）
Common circuits	既製のネットワークテンプレート：項目を選択して **Insert** をクリック（またはダブルクリック）し、名前を入力すると挿入されます

7.1.1 要素の挿入

まず要素（挿入位置の基準になります）またはネットワーク（ネットワークのタイトルバーをクリック）を選択し、次のいずれかの方法で挿入します。

- **ボタン**：「Toolbox」の「Instructions」にあるボタンをクリックします。**Shift** を押しながらボタンをクリックすると、選択した要素の前に挿入されます。
- **キー**：下表のキーを直接押します。**Shift+キー** で選択した要素の前に挿入し、**B** で選択した要素の下に a 接点を並列に追加します。
- 「**...**」**メニュー**：**...** をクリックし、カテゴリ別に任意の命令を選択します。

キー	命令	キー	命令	キー	命令
C	a 接点 NO	S	セットコイル SET	U	アップカウンター CTU [1.x]
N	b 接点 NC	R	リセットコイル RST	D	ダウンカウンター CTD [1.x]
P	立上りエッジ接点 PT	O	オンディレイ TON [1.x]	M	転送 MOV
L	コイル OUT	F	オフディレイ TOF [1.x]	A	加算 ADD
E	条件接点 CMP	K	ファンクションブロック CALL（FB）	T	ST ボックス
B	下に並列の a 接点	Shift+キー	選択した要素の前に挿入		

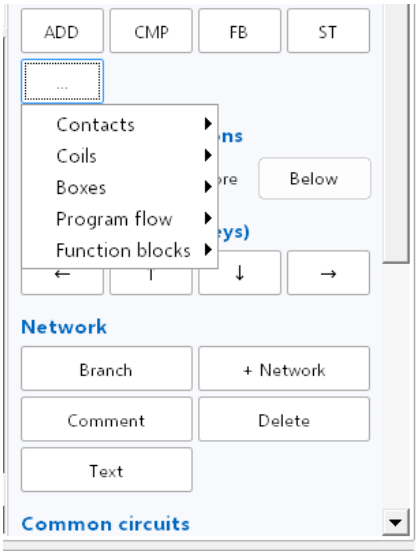


図 7-2 「...」はすべての命令をカテゴリ別に表示：Contacts、Coils、Boxes、Program flow、Function blocks

挿入位置は「Insert the instructions」で決まります。たとえば `Xstart` を選択した状態で b 接点を挿入すると、次のようになります。

Insert the instructions	**結果（.lad）**
After	`NO(Xstart) NC(Xstop)`
Before（または Shift+N）	`NC(Xstop) NO(Xstart)`
Below	`[ NO(Xstart) | NC(Xstop) ]`

コイル類（OUT、SET、RST…）は常にネットワークの右側に置かれます。ネットワークの末尾に連続する複数のコイルはすべて右母線の横に接続され、同じパワーフローで駆動されます。

7.1.2　選択、変更、移動、削除

操作	方法
要素を選択	要素をクリック、または ← → ↑ ↓
ネットワークを選択	ネットワークのタイトルバー（番号の行）をクリック。Home／End で最初／最後のネットワークを選択
要素のパラメーターを変更	要素をダブルクリック、または選択して Enter
要素を移動	Ctrl+←／Ctrl+→：同じ直列内の左右の要素と位置を入れ替え
ネットワークを移動	ネットワークを選択して Ctrl+↑／Ctrl+↓
削除	Del（または Backspace）、または「Network」の **Delete**
切り取り、コピー、貼り付け	Ctrl+X／Ctrl+C／Ctrl+V（要素またはネットワーク全体）
元に戻す、やり直し	Ctrl+Z／Ctrl+Y

要素を右クリックすると、Edit、Cut、Copy、Paste、Delete のほかに、その変数のコマンドがあります：**I/O mapping of <変数>...**、**86HMI binding of <変数>...**、**Add <変数> to the watch window**、**Add to the trace**、**Where is <変数> used?**、**Its declaration**、**Properties**。シミュレーション中またはオンライン時は、さらに **Write**、**Force**、**Unforce** があります。

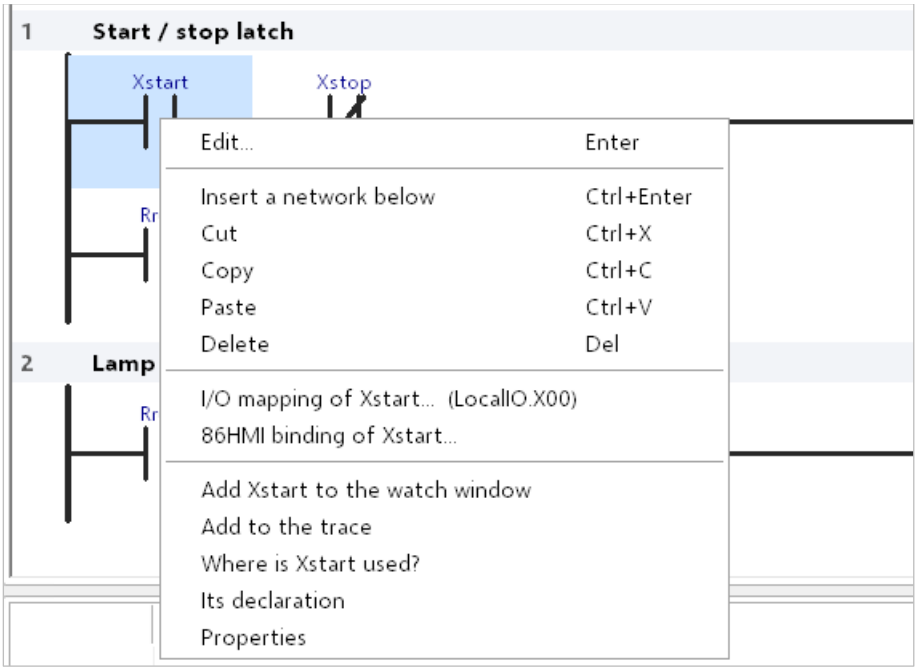


図 7-3　要素の右クリックメニュー：編集、ネットワーク、クリップボード、および変数の I/O マッピング、ウォッチ、Cross reference

***注意：**シミュレーション中またはオンライン時はラダー図を編集できません（ステータスバーに「Stop the simulation / go offline to edit」と表示されます）。このとき接点またはコイルをダブルクリックすると、その変数の値が切り替わります。

7.1.3 ネットワーク

操作	方法
ネットワークを追加	「Network」の **+ Network** または Ctrl+Enter：選択したネットワークの後に挿入。Ctrl+Shift+Enter：前に挿入
タイトルとコメント	ネットワークのタイトルバーをダブルクリック、右クリックの「Network title and comment...」、または「Network」の **Comment**
ネットワークを削除	ネットワークを選択して Del、または「Network」の **Delete**
空のネットワーク	「(empty: select the network and insert an element)」と表示されます。選択すると挿入できます

ネットワークのウィンドウには 3 つの欄があります：**Title**（ネットワークの上に表示）、**Comment**（複数行。タイトルの下に表示）、**Body (text)**（このネットワークの .lad テキスト。直接修正したり、ロジックをまとめて貼り付けたりでき、OK をクリックしたときに構文がチェックされます）。

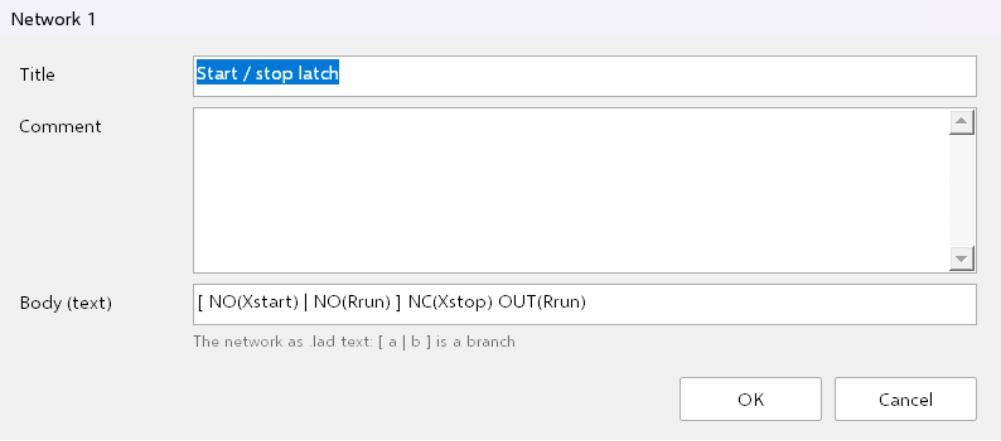


図 7-4　ネットワークのタイトル、コメント、本体（.lad テキスト）

7.1.4　テキストモード

「Network」の **Text**（または Ctrl+T）をクリックすると、POU 全体が .lad テキスト編集に切り替わります。大量の修正、コピー＆ペースト、バージョン管理での差分比較に便利です。上部の **Back to the diagram** をクリックすると書式がチェックされます。ファイルに構文エラーがある場合は、開いた時点でテキストモードのままになり、エラーの位置が示されます。



図 7-5　テキストモード：POU 全体の .lad テキスト。上部に「Back to the diagram」

```
PROGRAM Main
VAR
    tLamp : TON;
END_VAR
NETWORK '起動／停止の自己保持'
    // コメント
    [ NO(Xstart) | NO(Rrun) ] NC(Xstop) OUT(Rrun)
NETWORK
    NO(Rrun) CALL(tLamp, IN := @, PT := T#2s, Q => @) OUT(Ylamp)
END_PROGRAM
```

- `PROGRAM`／`FUNCTION_BLOCK` で始まり、対応する `END_…` で終わります。宣言部の書き方は ST と同じです。
- `NETWORK 'タイトル'` でネットワークが始まり、その後の `//` 行はコメントです。
- ネットワークは空白で区切られた**直列**の要素です。`[ a | b | c ]` は**並列**分岐で、各分岐はそれ自体が直列でもよく、さらに並列を入れ子にすることもできます。
- 要素は `命令(パラメーター, ...)` と書きます。パラメーターのない命令は名前だけを書きます（例：`OSR`、`MCR`）。ファンクションブロックのパラメーターは `名前 := 値`、出力は `名前 => 出力先` と書き、`@` はパワーフローを表します。

7.1.5 Common circuits

サイドパネルの「Common circuits」には既製のネットワークテンプレートがあります。項目を選択し（下に説明が表示されます）、**Insert** をクリック（またはダブルクリック）して、フォームに変数名と時間を入力すると、選択したネットワークの後にネットワークが挿入され、新しい変数は自動的に宣言されます（入力、出力、内部ビットは GVL に、タイマーとカウンターはこの POU に宣言）。

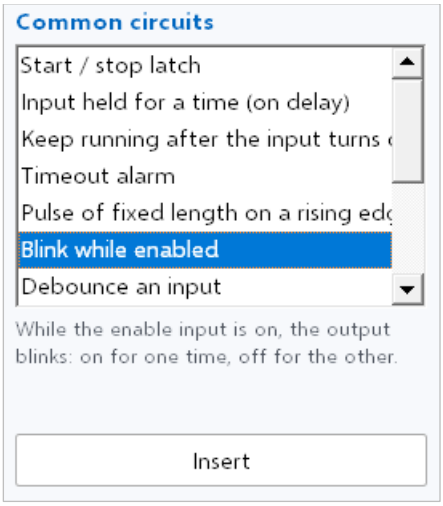


図 7-6　Common circuits の一覧：項目を選択すると下に説明が表示され、「Insert」をクリック

テンプレート	生成されるネットワーク（.lad、名前は既定値）
Start / stop latch	`[ NO(Xstart) | NO(Ymotor) ] NC(Xstop) OUT(Ymotor)`
Input held for a time (on delay)	`NO(Xsensor) TON(Tconfirm, T#2000ms) OUT(Yconfirmed)`
Delayed off after the input goes off (off delay)	`NO(Xheater) TOF(Tafter, T#5000ms) OUT(Yfan)`
Timeout alarm	`NO(Rstep) NC(Xdone) TON(Ttimeout, T#10000ms) SET(Yalarm)`；`NO(Xreset) RST(Yalarm)`
Pulse of fixed length on a rising edge	`PT(Xtrigger) SET(Ypulse)`；`NO(Ypulse) TON(Tpulse, T#500ms) RST(Ypulse)`
Blink while enabled	2 つの TON が交互に出力を SET／RST し、イネーブルが OFF のとき RST
Debounce an input	`NO(Xswitch) TON(Tdb_on, T#50ms) SET(Rswitch)`；`NC(Xswitch) TON(Tdb_off, T#50ms) RST(Rswitch)`
Count to N	`NO(Xpart) CTU(Cparts, 10) OUT(Ydone)`；`NO(Xreset) RES(Cparts)`
Forward / reverse with interlock	正転・逆転それぞれに自己保持ネットワークを 1 つずつ設け、互いの b 接点でインターロック
Level alarm with hysteresis	`GRT(Alevel, 3000) SET(Yhigh)`；`LES(Alevel, 2800) RST(Yhigh)`
One button on / off (toggle)	`NO(Xbutton) TOGGLE(Ylight)`
Square wave (cyclic timer)	`NO(Xrun) TCY(Tcycle, T#1000ms) OUT(Yclock)`
IEC timer block (TON)	`NO(Xstart) CALL(tonStart, IN := @, PT := T#1000ms, Q => @) OUT(Ydelayed)`

7.2 接点とコイル

接点は BOOL 値を読み取り、パワーフローを通すかどうかを決めます。コイルはパワーフローに従って BOOL 変数に書き込みます。接点の変数には、GVL または VAR の BOOL 変数のほか、ファンクションブロックの BOOL 出力（例：`NO(tLamp.Q)`）も使えます。

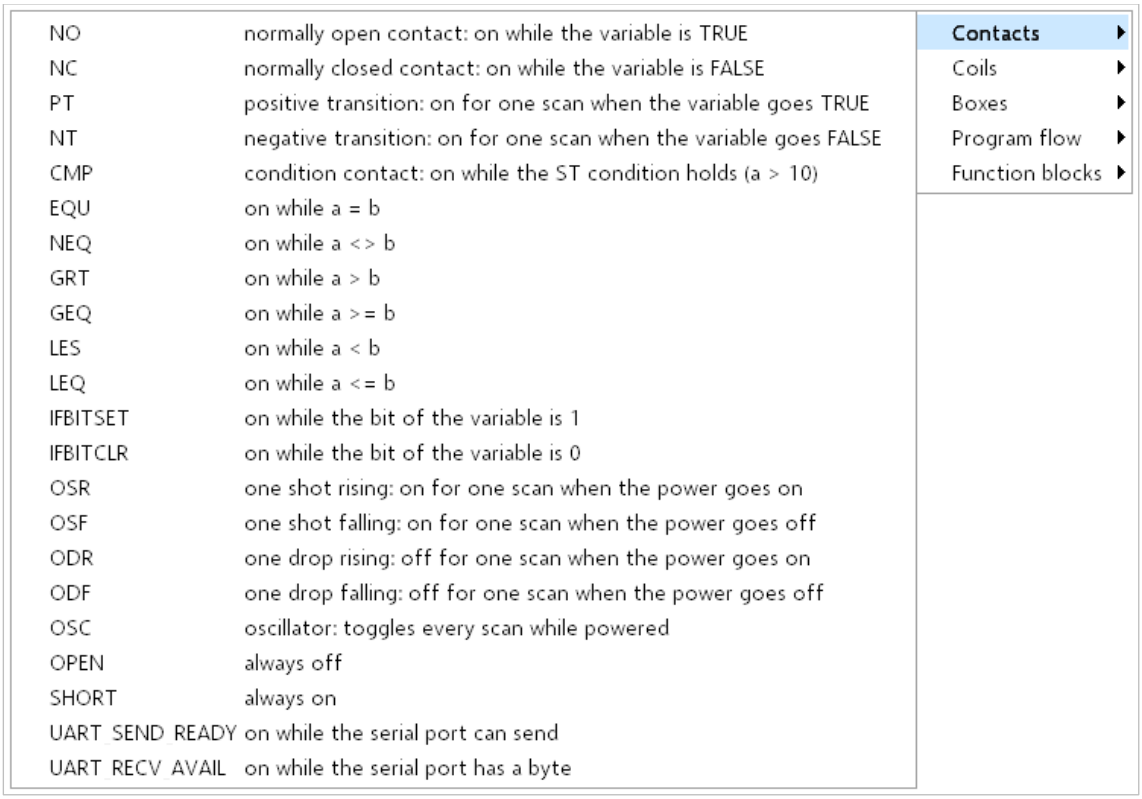


図 7-7 「...」>「Contacts」：すべての接点命令

命令	.lad 記述	キー	説明
NO	`NO(Xstart)`	C	a 接点：変数が TRUE のとき導通
NC	`NC(Xstop)`	N	b 接点：変数が FALSE のとき導通
OPEN	`OPEN`	—	常に開（例：回路の一部を一時的に切り離す）
SHORT	`SHORT`	—	常に導通（例：条件を一時的にスキップする）
OUT	`OUT(Ylamp)`	L	コイル：変数がパワーフローに従う（パワーフローありで TRUE、なしで FALSE）
OUTN	`OUTN(Yready)`	—	反転コイル：変数はパワーフローの反転
SET	`SET(Yrun)`	S	セットコイル：パワーフローがあると TRUE にして保持（パワーフローがなければ変化しない）
RST	`RST(Yrun)`	R	リセットコイル：パワーフローがあると FALSE にして保持
TOGGLE	`TOGGLE(Ylight)`	—	トグルコイル：パワーフローの立上りエッジごとに変数を反転 [1.x]

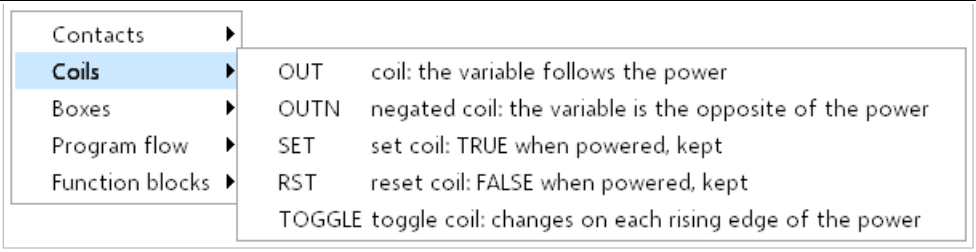


図 7-8 「...」>「Coils」：OUT、OUTN、SET、RST、TOGGLE

*ヒント：1 つの変数は 1 つの OUT コイルだけで駆動します。複数の条件が必要なときは、それらを同じコイルの前に並列または直列に接続します（6.0 節の原則 ③）。SET／RST を対で使う場合は、後ろに書いたネットワークが優先されます。

7.3　エッジとパルス

エッジ命令は状態が変化したその 1 スキャンだけ導通し、「1 回押すと 1 回動作する」処理（カウント、1 加算、データの送信）に使います。PT、NT は変数を検出し、OSR、OSF、ODR、ODF は左から入ってくるパワーフローを検出します。

命令	.lad 記述	キー	説明
PT	PT(Xstart)	P	立上りエッジ接点：変数が FALSE から TRUE になったとき 1 スキャン導通
NT	NT(Xstart)	—	立下りエッジ接点：変数が TRUE から FALSE になったとき 1 スキャン導通
OSR	NO(Xa) OSR	—	立上りワンショット：パワーフローが導通したとき 1 スキャン導通 [1.x]
OSF	NO(Xa) OSF	—	立下りワンショット：パワーフローが遮断されたとき 1 スキャン導通 [1.x]
ODR	NO(Xa) ODR	—	立上りオフ：パワーフローが導通したとき 1 スキャン遮断 [1.x]
ODF	NO(Xa) ODF	—	立下りオフ：パワーフローが遮断されたとき 1 スキャン遮断 [1.x]
OSC	NO(Xrun) OSC OUT(Yclk)	—	発振器：通電中はスキャンごとに切り替わる（周期は 2 スキャン）[1.x]

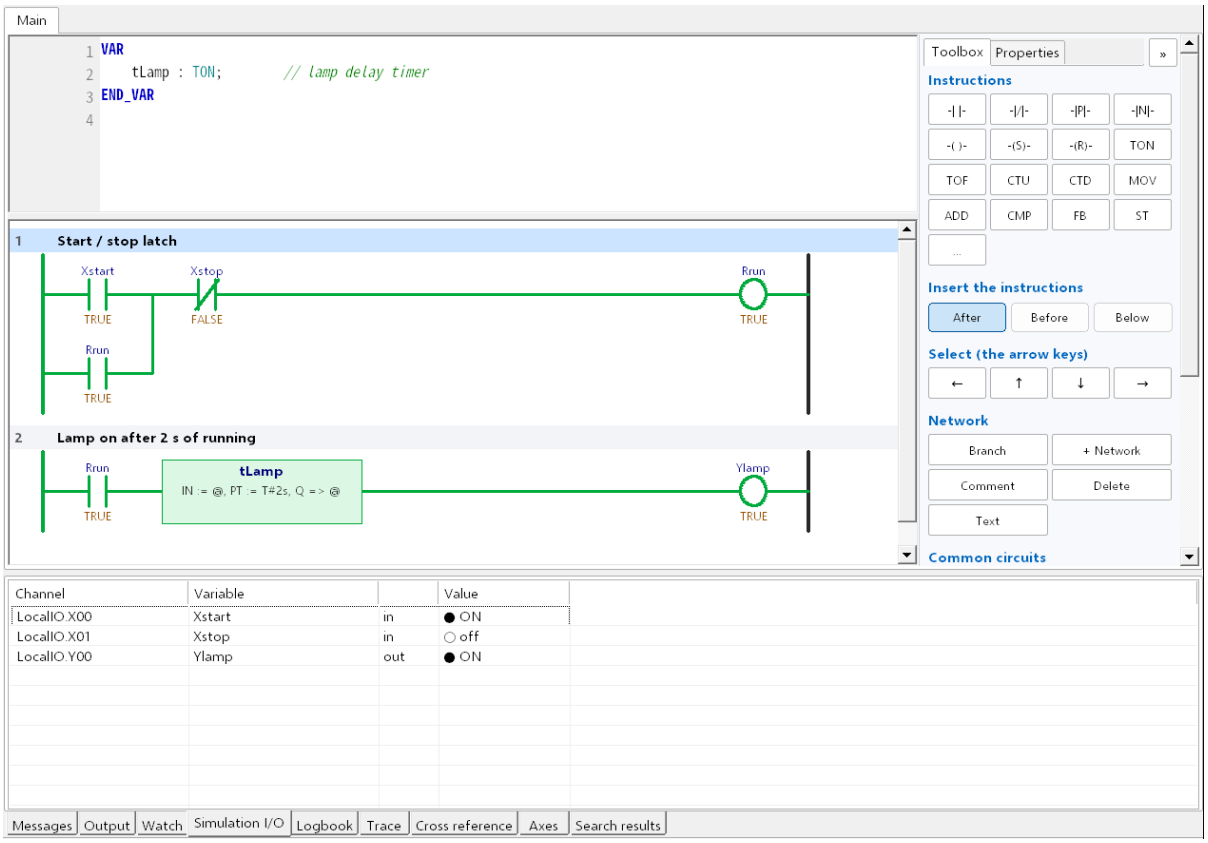


図 7-9　エッジ接点の例（ネットワーク 3）：PT(Xstart) を 1 回押すごとに ADD が Count に 1 を加算

パルス出力：次の 1.x 命令はデジタル出力にパルス列を生成します。出力パラメーターは、I/O マッピングで物理出力に割り付けた BOOL 変数です。加減速が必要な位置決め制御には、代わりに Motion86 の軸を使用してください（第 12 章）。

命令	.lad 記述	キー	説明
NPULSE	`PT(Xgo) NPULSE(100, 1000, Ypls)`	—	パワーフローの立上りエッジで出力に N 個のパルスを送出。パラメーター：数、周波数（Hz）、出力変数 [1.x]
NPULSE_OFF	`NO(Xstop) NPULSE_OFF`	—	パワーフローがあると N パルスを停止 [1.x]
PULSER	`NO(Xa) PULSER(10, 10, 0, pc, Ypls)`	—	コイル変数にパルス列を出力。パラメーター：P1、P0（ON、OFF の長さ）、加速、カウンター、コイル [1.x]
STEPPER	`NO(Xb) STEPPER(stp, Nsteps, 1, 1, 0, Ystep)`	—	ステッピングモーターのパルス。パラメーター：名前、最大値（ステップ数）、P、n、カーブ、コイル。加速テーブルは非対応（n > 1 かつカーブ > 0 のときエラー。NPULSE または Motion86 を使用してください）[1.x]

*ヒント：IEC のエッジ検出ファンクションブロック `R_TRIG`、`F_TRIG` は CALL で使用します。例：`NO(Xa) CALL(rStart, CLK := @, Q => @) ADD(Count, Count, 1)`（7.8 節）。

7.4 タイマー

タイマーには 2 種類あります：**IEC ファンクションブロック**（TON、TOF、TP。実時間で計時し、CALL で挿入。7.8 節）と **1.x タイマーボックス**（本節の表）です。1.x タイマーの最初のパラメーターはタイマー変数（`DINT` などの整数として宣言し、経過した PLC 周期数を記録）で、PLC 周期を数えて計時します：ディレイ ÷ タスク周期（整数に切り捨て）。

- **Delay**：`T#500ms`、`T#2s` のように書きます。単位のない数値は 1.x と同様にマイクロ秒とみなされ、チェック時に警告が出ます。ディレイがタスク周期より短い場合は計時できません（警告）。
- **Adjust**（省略可）：ディレイに加算する PLC 周期数（負の値も可）。1.x と同じです。
- TON、TOF、TCY、THI、TLO は入力が変化すると自動的にリセットされます。RTO、RTL は時間を積算するため、`RES` でクリアします（7.5 節）。

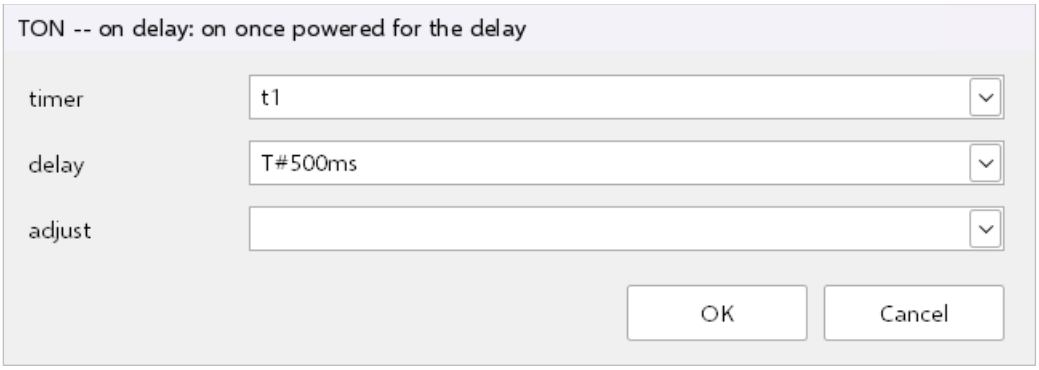


図 7-10　1.x TON のパラメーターウィンドウ：タイマー、ディレイ、調整

命令	.lad 記述	キー	説明
TON	`NO(Xa) TON(t1, T#500ms) OUT(Ya)`	O	オンディレイ：通電がディレイ時間に達すると導通 [1.x]
TOF	`NO(Xa) TOF(t2, T#1s) OUT(Ya)`	F	オフディレイ：遮断後もディレイ時間だけ導通 [1.x]
RTO	`NO(Xa) RTO(t3, T#2s) OUT(Yb)`	—	積算オンディレイ：通電時間を積算し、ディレイに達すると導通（RES でクリア）[1.x]
RTL	`NO(Xa) RTL(t4, T#2s) OUT(Yb)`	—	積算オフディレイ：遮断時間を積算 [1.x]
TCY	`NO(Xrun) TCY(t5, T#500ms) OUT(Yclk)`	—	サイクリックタイマー：通電中、ディレイ時間ごとに ON／OFF を切り替え [1.x]
THI	`NO(Xa) THI(t6, T#1s) OUT(Yp)`	—	立上りエッジでディレイ長のパルスを出力 [1.x]
TLO	`NO(Xa) TLO(t7, T#1s) OUT(Yp)`	—	立下りエッジでディレイ長の Low パルスを出力 [1.x]
T2CNT	`NO(Xa) T2CNT(Dcycles, T#1s)`	—	ディレイを PLC 周期数に換算して変数に格納 [1.x]

IEC タイマーファンクションブロック（推奨）：インスタンスを VAR で宣言し（例：`tLamp : TON;`）、**FB**（K）で挿入します。

ファンクションブロック	.lad 記述	動作
TON	`NO(Xa) CALL(tLamp, IN := @, PT := T#2s, Q => @)` `OUT(Ya)`	IN が PT の間 TRUE を保つと Q = TRUE。IN が FALSE になると Q と ET は直ちにゼロに戻る
TOF	`NO(Xa) CALL(tOff, IN := @, PT := T#5s, Q => @)` `OUT(Yb)`	IN が TRUE の間 Q = TRUE。IN が FALSE になった後も Q は PT の間保持
TP	`NO(Xa) CALL(tPulse, IN := @, PT := T#1s, Q => @)` `OUT(Yc)`	IN の立上りエッジで Q = TRUE を PT の間保持。その間の IN の変化は影響しない

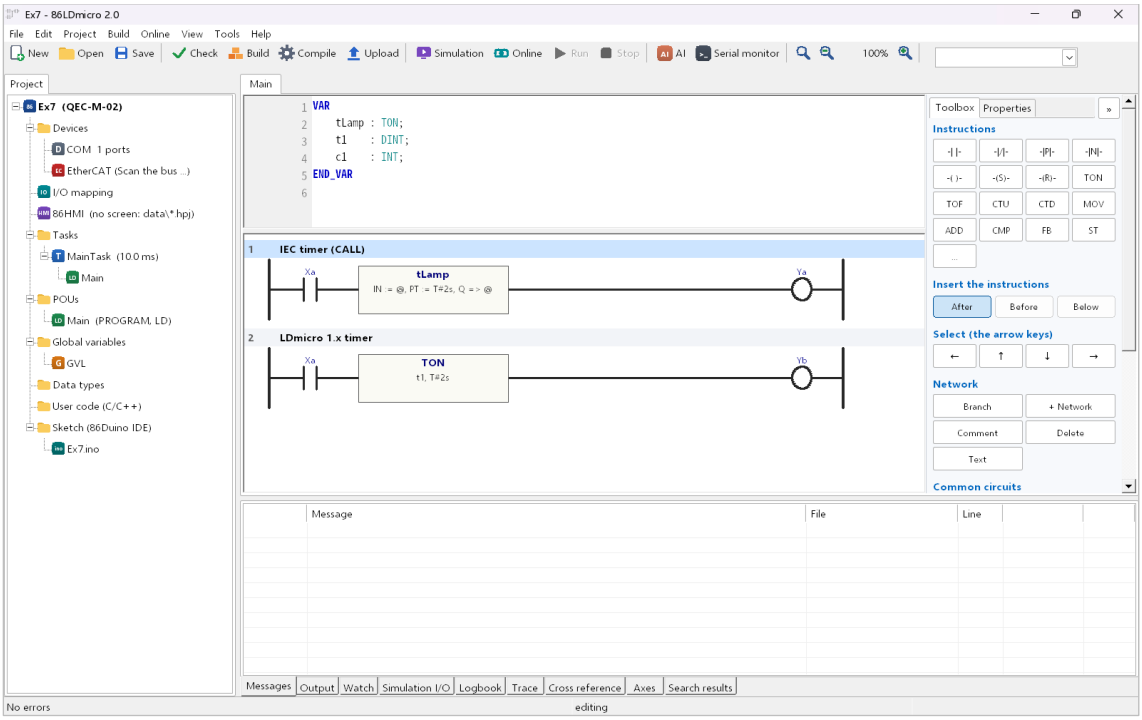


図 7-11　シミュレーション中の IEC TON（上）と 1.x TON（下）：ファンクションブロックはパラメーターを、1.x ボックスはカウント済みの周期数を表示

7.5　カウンター

1.x カウンターの最初のパラメーターはカウンター変数（INT などの整数として宣言）で、その値が現在のカウント値であり、比較命令でそのまま使えます。IEC カウンターファンクションブロック（CTU、CTD、CTUD）は 7.8 節を参照してください。

- パラメーター：**Counter**（変数）、**Max value**／**Min value**、**Initial value**（省略可、既定値 0）、**Input**（省略可。カウントするタイミング：RISING（既定、立上りエッジ）、FALLING（立下りエッジ）、HIGH（通電中の各スキャン）、LOW（遮断中の各スキャン））。
- RES はタイマー、カウンター、変数を 0 にクリアします。RES はボックスで、ネットワークの右端に置きます。

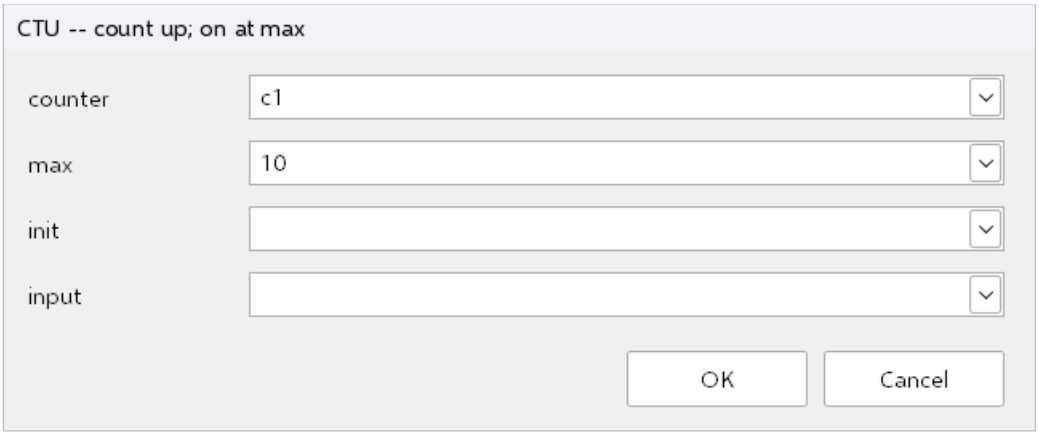


図 7-12　1.x CTU のパラメーターウィンドウ：カウンター、最大値、初期値、入力

命令	.lad 記述	キー	説明
CTU	NO(Xa) CTU(c1, 10) OUT(Ya)	U	アップカウント。カウント値 ≥ 最大値で導通 [1.x]
CTD	NO(Xb) CTD(c2, 0, 10) OUT(Yb)	D	ダウンカウント（この例では初期値 10）。最小値より大きい間導通 [1.x]
CTC	NO(Xb) CTC(c3, 9, 0, FALLING)	—	循環カウンター：カウント値が最大値を超えると初期値に戻り、戻ったその 1 回だけ導通（この例では立下りエッジでカウント）[1.x]
CTR	NO(Xb) CTR(c4, 0, 9)	—	循環ダウンカウンター：カウント値が最小値を下回ると初期値に戻り、戻ったその 1 回だけ導通 [1.x]
RES	NO(Xreset) RES(c1)	—	タイマー／カウンター／変数を 0 にリセット [1.x]

7.6 比較

比較命令はすべて**接点**で、条件が成立すると導通します。**CMP**（条件接点、E キー）の使用を推奨します。ST の比較式を直接書くことができ、任意の型（INT、REAL、TIME、ファンクションブロックのメンバー…）を比較でき、AND／OR を組み合わせることもできます。1.x の EQU～LEQ も引き続き使用できます。

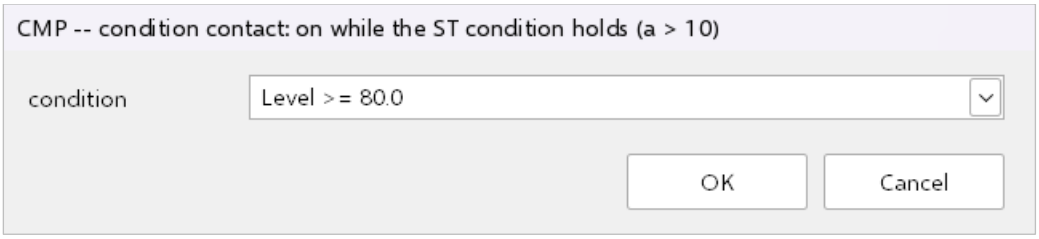


図 7-13　CMP 条件接点のパラメーターウィンドウ：ST の条件を入力

命令	.lad 記述	キー	説明
CMP	`CMP(Level >= 80.0) OUT(Yhigh)`	E	条件接点：ST の条件が成立すると導通。例：`Count = 5、tLamp.ET > T#1s、(a > 10) AND NOT b`
EQU	`EQU(Count, 10)`	—	a = b のとき導通 [1.x]
NEQ	`NEQ(Count, 0)`	—	a <> b のとき導通 [1.x]
GRT	`GRT(Level, 5)`	—	a > b のとき導通 [1.x]
GEQ	`GEQ(Level, 5)`	—	a >= b のとき導通 [1.x]
LES	`LES(Level, 90)`	—	a < b のとき導通 [1.x]
LEQ	`LEQ(Level, 90)`	—	a <= b のとき導通 [1.x]
IFBITSET	`IFBITSET(Nbits, 3)`	—	変数のビット n が 1 のとき導通（ビット 0 が最下位ビット）[1.x]
IFBITCLR	`IFBITCLR(Nbits, 0)`	—	変数のビット n が 0 のとき導通 [1.x]

7.7 演算と転送

演算命令と転送命令はすべて**ボックス**で、パワーフローがあると演算を 1 回実行します（つまり毎スキャン実行されます。1 回だけ実行したいときは、前に PT などのエッジ接点を置きます）。最初のパラメーターは結果の格納先変数です。四則演算でいずれかのオペランドが REAL の場合は浮動小数点で計算し、格納先が整数の場合は最も近い整数になります。複雑な計算には ST ボックス（7.8 節）または ST プログラムを推奨します。

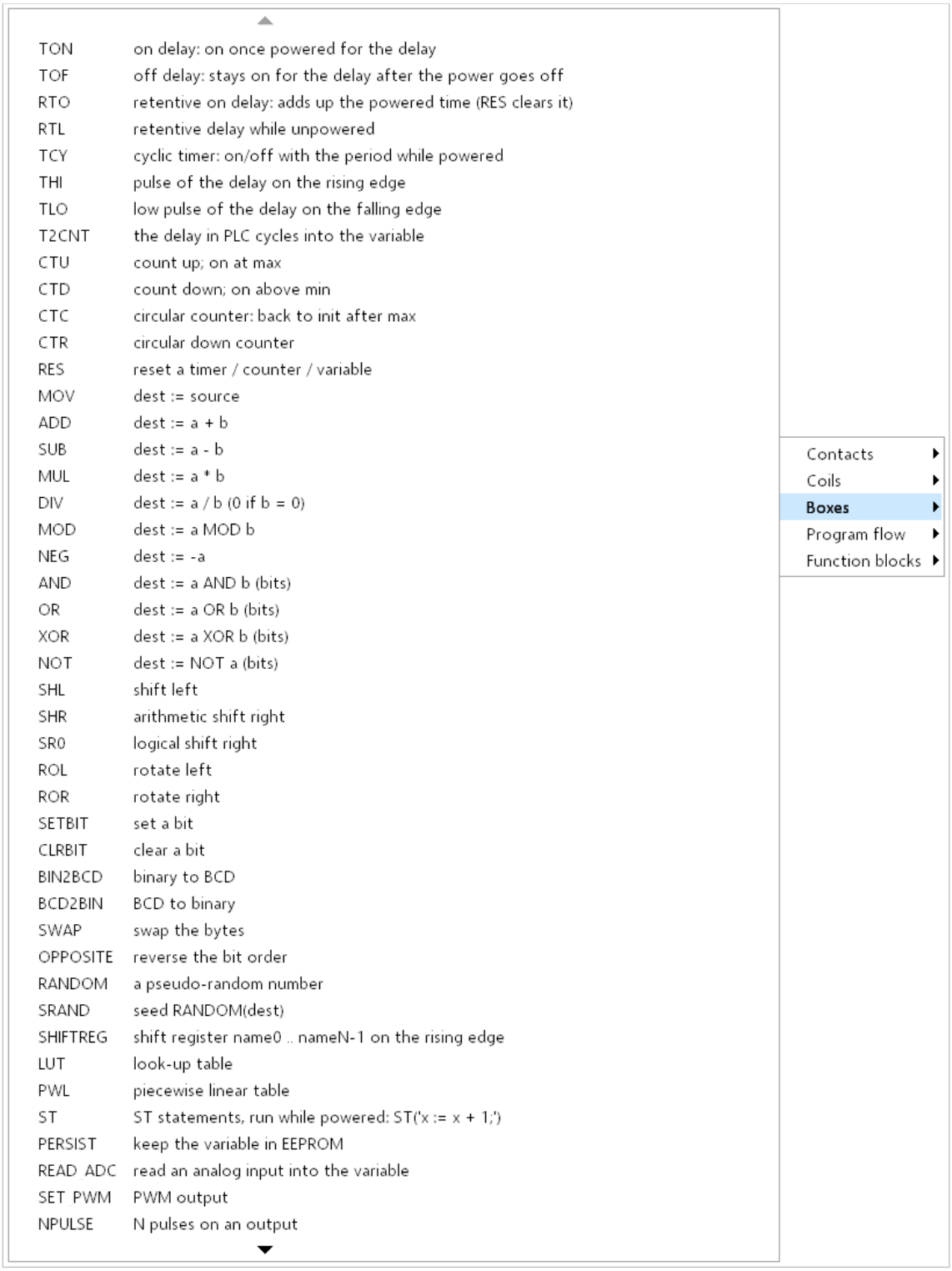


図 7-14 「...」>「Boxes」：タイマー、カウンター、演算、転送、I/O のボックス

7.7.1 転送と四則演算

命令	.lad 記述	キー	説明
MOV	`MOV(Val, 100)`	M	格納先 := 転送元
ADD	`ADD(Count, Count, 1)`	A	格納先 := a + b
SUB	`SUB(Val, Val, 1)`	—	格納先 := a - b
MUL	`MUL(Val, Val, 2)`	—	格納先 := a * b
DIV	`DIV(Val, Val, 3)`	—	格納先 := a / b（整数を 0 で割ると結果は 0）
MOD	`MOD(Val, Val, 7)`	—	格納先 := a MOD b（剰余）
NEG	`NEG(Val, Val)`	—	格納先 := -a

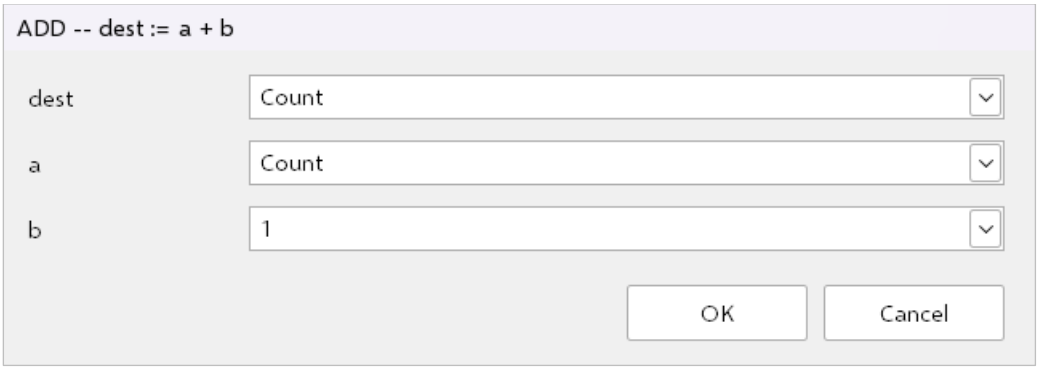


図 7-15　ADD のパラメーターウィンドウ：格納先、a、b

7.7.2 ビット演算

命令	.lad 記述	キー	説明
AND	`AND(Nbits, Nbits, 16#0F)`	—	格納先 := a AND b（ビットごと）[1.x]
OR	`OR(Nbits, Nbits, 1)`	—	格納先 := a OR b（ビットごと）[1.x]
XOR	`XOR(Nbits, Nbits, 2)`	—	格納先 := a XOR b（ビットごと）[1.x]
NOT	`NOT(Nbits, Nbits)`	—	格納先 := NOT a（ビットごとの反転）[1.x]
SHL	`SHL(Nbits, Nbits, 1)`	—	n ビット左シフト [1.x]
SHR	`SHR(Nbits, Nbits, 1)`	—	算術右シフト（符号を保持）[1.x]
SR0	`SR0(Nbits, Nbits, 1)`	—	論理右シフト（左側を 0 で埋める）[1.x]
ROL	`ROL(Nbits, Nbits, 1)`	—	左ローテート [1.x]
ROR	`ROR(Nbits, Nbits, 1)`	—	右ローテート [1.x]
SETBIT	`SETBIT(Nbits, 2)`	—	変数のビット n を 1 にセット [1.x]
CLRBIT	`CLRBIT(Nbits, 5)`	—	変数のビット n を 0 にクリア [1.x]

7.7.3 変換、乱数、テーブル

命令	.lad 記述	キー	説明
BIN2BCD	`BIN2BCD(Nbcd, Val)`	—	バイナリから BCD へ変換 [1.x]
BCD2BIN	`BCD2BIN(Val, Nbcd)`	—	BCD からバイナリへ変換 [1.x]
SWAP	`SWAP(Nraw, Val)`	—	バイトを入れ替え [1.x]
OPPOSITE	`OPPOSITE(Nraw, Val)`	—	ビット順を反転 [1.x]
RANDOM	`RANDOM(Val)`	—	擬似乱数を格納先に格納 [1.x]
SRAND	`SRAND(Val, 1234)`	—	RANDOM のシードを設定 [1.x]
SHIFTREG	`NO(Xclk) SHIFTREG(sreg, 4)`	—	シフトレジスター：パワーフローの立上りエッジで `sreg0→sreg1→…→sreg3` の順にシフト。`sreg0〜sreg3` は自分で宣言する（同じ型）[1.x]
LUT	`LUT(Nout, Nidx, 10, 20, 40, 80)`	—	ルックアップテーブル：格納先 := 「インデックス」番目の値（インデックスは 0 から）[1.x]
PWL	`PWL(Nout, Nraw, 0, 0, 100, 50, 200, 200)`	—	折れ線テーブル：(x0, y0)、(x1, y1)… の折れ線に沿ってインデックスを格納先に換算。x は昇順、値は整数で、値が大きすぎるとチェック時に小さくするよう提示される [1.x]

7.8 IEC ファンクションブロック（CALL）と ST 命令

CALL は任意のファンクションブロックをネットワークに配置します：IEC 標準ファンクションブロック、PLCopen／Motion86 のモーション制御ファンクションブロック（7.11 節）、およびプロジェクト内で自作した FUNCTION_BLOCK（8.3 節）です。ファンクションブロックのインスタンスは、先に VAR（または GVL）で宣言しておきます。

1. 宣言部でインスタンスを宣言します。例：`tLamp : TON;`、`cParts : CTU;`、`rStart : R_TRIG;`。
2. 要素またはネットワークを選択し、**FB**（K）をクリックします。
3. 「Element」欄に `CALL(`インスタンス`,` パラメーター `:=` 値`,` 出力 `=>` 出力先`)` と入力し、OK をクリックします。

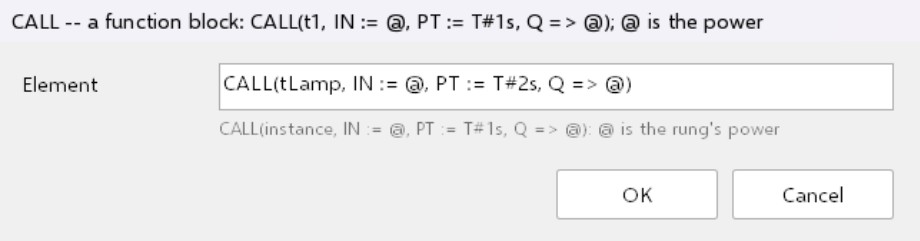


図 7-16　CALL のパラメーターウィンドウ：「Element」欄に CALL 全体を入力

- `IN := @`：左から入ってくるパワーフローをこの入力に接続します。`Q => @`：この出力が右へのパワーフロー（後続の要素へ）になります。
- その他の入力には変数または定数を接続でき（`R := Xreset`、`PV := 10`）、出力は変数に格納できます（`CV => Count`）。使わないパラメーターは省略してかまいません。
- ファンクションブロックの出力は、ほかのネットワークで直接読み取ることもできます：`NO(tLamp.Q)`、`CMP(cParts.CV >= 5)`。

ファンクションブロック	入力	出力	.lad 記述例
TON／TOF／TP	IN, PT	Q, ET	`CALL(tLamp, IN := @, PT := T#2s, Q => @)`
CTU（CTU_DINT）	CU, R, PV	Q, CV	`CALL(cParts, CU := @, R := Xreset, PV := 10, Q => @, CV => Count)`
CTD（CTD_DINT）	CD, LD, PV	Q, CV	`CALL(cDown, CD := @, LD := Xload, PV := 5, Q => @)`
CTUD（CTUD_DINT）	CU, CD, R, LD, PV	QU, QD, CV	`CALL(cUD, CU := @, CD := Xdown, R := Xreset, PV := 10, QU => @)`
R_TRIG／F_TRIG	CLK	Q	`CALL(rStart, CLK := @, Q => @)`
SR（セット優先）	S1, R	Q1	`CALL(sr1, S1 := @, R := Xstop, Q1 => @)`
RS（リセット優先）	S, R1	Q1	`CALL(rs1, S := @, R1 := Xstop, Q1 => @)`

ST ボックス：パワーフローがあると ST の文を実行します（**ST** をクリックまたは T キー）。文字列中の改行は `$N` と書きます。ラダー図の中でちょっとした計算をするのに適しています。長いロジックは ST プログラムとして書いてください（第 8 章）。

命令	.lad 記述	キー	説明
CALL	`NO(Xa) CALL(tLamp, IN := @, PT := T#2s, Q => @) OUT(Ya)`	K	ファンクションブロック：@ はパワーフロー
ST	`NO(Xa) ST('Count := Count + 1; Level := Count * 2;')`	T	ST の文。パワーフローがあると実行

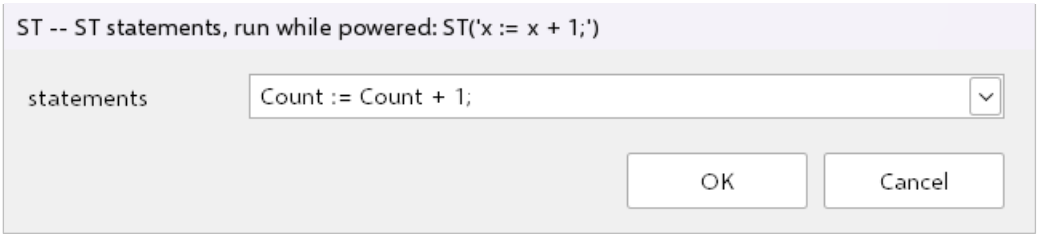


図 7-17　ST ボックスのパラメーターウィンドウ：ST の文を入力

*ヒント：「...」>「Function blocks」サブメニューには CALL しかありません。どのファンクションブロックを挿入するかは、CALL の最初のパラメーター（インスタンス）の型で決まります。

7.9　プログラムフロー

プログラムフロー命令はネットワークの実行順序を変更します。LDmicro 1.x と同じです。一般的なプログラムではほとんど必要ありません。ステップに分かれた処理には SFC（8.4 節）を、繰り返し使うロジックには FUNCTION_BLOCK を使うことを推奨します。

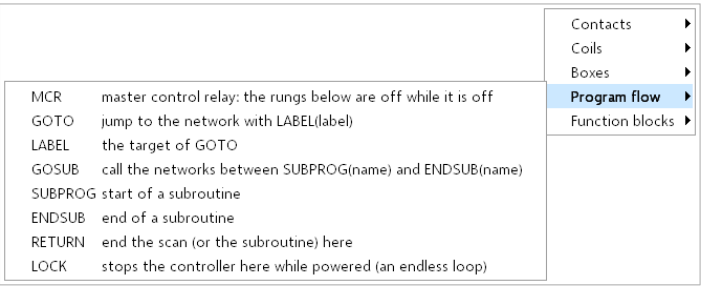


図 7-18　「...」>「Program flow」：MCR、GOTO、LABEL、GOSUB、SUBPROG、ENDSUB、RETURN、LOCK

命令	.lad 記述	キー	説明
MCR	`NO(Xauto) MCR ... MCR`	—	マスターコントロールリレー：条件が OFF のとき、次の `MCR` までその下のネットワークのパワーフローをすべて遮断 [1.x]
GOTO	`NO(Xskip) GOTO(skip)`	—	`LABEL(skip)` を含むネットワークにジャンプ（ネットワーク番号も可）[1.x]
LABEL	`LABEL(skip)`	—	GOTO のジャンプ先。ネットワークの先頭に置く [1.x]
GOSUB	`NO(Xa) GOSUB(Blink)`	—	呼び出し：`SUBPROG(Blink)` と `ENDSUB(Blink)` の間のネットワーク [1.x]
SUBPROG	`SUBPROG(Blink)`	—	サブルーチンの開始（単独のネットワーク）[1.x]
ENDSUB	`ENDSUB(Blink)`	—	サブルーチンの終了（単独のネットワーク）[1.x]
RETURN	`NO(Xb) RETURN`	—	ここで今回のスキャンを終了（サブルーチン内では呼び出し元に戻る）[1.x]
LOCK	`NO(Xa) NO(Xb) NO(Xc) LOCK`	—	通電するとコントローラーをここで停止（無限ループ）[1.x]

```
NETWORK '自動モードのときだけ次の 2 つのネットワークを実行'
    NO(Xauto) MCR
NETWORK
    NO(Xb) OUT(Ya)
NETWORK
    NO(Xc) OUT(Yb)
NETWORK 'マスターコントロール終了'
    MCR
NETWORK
    NO(Xa) GOSUB(Blink)
NETWORK
    SUBPROG(Blink)
NETWORK
    NO(Xa) OUTN(Yc)
NETWORK
    ENDSUB(Blink)
```

***安全：**`LOCK` は通電するとコントローラーを無限ループで停止させ、スキャンを行わなくなります。1.x プログラムの移行用にのみ残されているため、新しいプログラムでは使用しないでください。

7.10 I/O、シリアルポート、その他

本節では LDmicro 1.x のハードウェア関連命令を説明します。新しいプロジェクトでは、I/O は通常「I/O mapping」（4.6 節）で直接扱い、シリアル通信には C/C++ のユーザーコード（第 14 章）を推奨します。以下の命令は主に 1.1 のプログラムの移行用です。

- **COM ポートパラメーター**：`COM0`、`COM1`… と書き、プロジェクトの「Devices > COM」にあるシリアルポート（`Serial`、`Serial1`…）に順に対応します。`Serial`（USB）はアップロードとオンラインモニター用に予約されているため、外部機器との通信には `Serial1`〜`Serial3` を使用してください。
- **アナログと PWM**：`READ_ADC` の変数は I/O マッピングでアナログ入力に割り付ける必要があります。`SET_PWM` のピンパラメーターは、I/O マッピングで `Pin.n`（Arduino のピン）に割り付けた変数です。

命令	.lad 記述	キー	説明
READ_ADC	`NO(Xa) READ_ADC(Nadc)`	—	アナログ入力を変数に読み込む（変数は I/O マッピングでアナログ入力に割り付ける。例：`Pin.n`）[1.x]
SET_PWM	`NO(Xa) SET_PWM(Nduty, 1000, Npwm)`	—	PWM 出力。パラメーター：デューティ比、周波数、ピン（`Pin.n` に割り付けた変数）、分解能（省略可）[1.x]
PERSIST	`NO(Xa) PERSIST(Count)`	—	変数を EEPROM に保存：電源投入後の最初のスキャンで読み戻し、以後は値が変化したときに書き込む [1.x]
UART_SEND	`PT(Xb) UART_SEND(Val, COM1)`	—	シリアルポートから変数を送信。パラメーター：変数、COM、バイト数（省略可）、待機（省略可）[1.x]
UART_RECV	`UART_RECV_AVAIL(COM1)` `UART_RECV(Val, COM1)`	—	シリアルポートから 1 バイトを変数に受信（通常は UART_RECV_AVAIL の後に置く）。パラメーター：変数、COM、バイト数（省略可）、待機（省略可）[1.x]
UART_SEND_READY	`UART_SEND_READY(COM1)`	—	接点：シリアルポートが送信可能なとき導通 [1.x]
UART_RECV_AVAIL	`UART_RECV_AVAIL(COM1)`	—	接点：シリアルポートにデータがあるとき導通 [1.x]
FMTSTR	`PT(Xc) FMTSTR(Count, 'Count = \5\r\n', COM1)`	—	パワーフローの立上りエッジで書式付き文字列をシリアルポートに送信（1 スキャンに 1 文字）。`\1`〜`\5` は変数の桁数、`\-5` は符号付き [1.x]
STRING	`NO(Xa) STRING(sMsg, Count, 'N=%d')`	—	書式付き文字列を STRING 変数に格納（C の printf 書式）[1.x]

命令	.lad 記述	キー	説明
DELAY	`NO(Xa) DELAY(10)`	—	ビジーウェイト（マイクロ秒）。スキャン時間が長くなる [1.x]

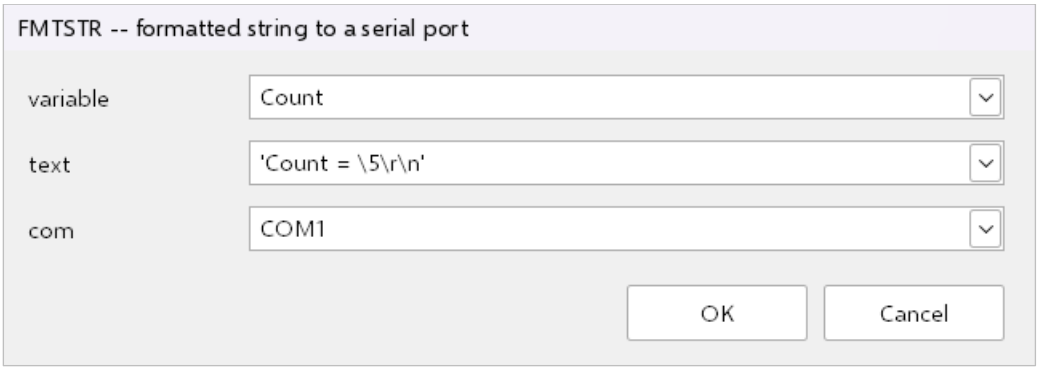


図 7-19　FMTSTR のパラメーターウィンドウ：変数、テキスト、COM

***注意：** `PERSIST` による EEPROM（フラッシュメモリー）への書き込みには寿命の制限があるため、あまり変更しない設定値にのみ使用してください。電源断後も保持する一般の変数は `VAR_GLOBAL RETAIN` で宣言してください。

`DELAY` はスキャン中にビジーウェイトするため、スキャン時間が長くなり、ウォッチドッグが作動する可能性があります（10.5 節）。

7.11 モーション制御命令

ラダー図でのモーション制御には 2 つの書き方があります。

- **PLCopen MC と Motion86 のファンクションブロック**（推奨）：`MC_Power`、`MC_MoveAbsolute`、`MC_Home...`、およびマシンの `M86_*` を **CALL** で挿入し、パワーフローを Enable／Execute に接続します。`Axis :=` には軸名を、`Machine :=` にはマシン名を指定します（第 12 章）。
- **1.x の MC 要素**：`MC(機能, 軸, a1, a2, a3, a4, 完了, ビジー, エラー)`。LDmicro 1.1 から移行したプログラム用です。軸は 86EVA サーボにバインドした変数で、使わないパラメーターには - を指定します。

```
VAR
    pwrX  : MC_Power;
    homeX : MC_Home;
    mvX   : MC_MoveAbsolute;
END_VAR
NETWORK 'サーボオン'
    NO(Xenable) CALL(pwrX, Axis := AxisX, Enable := @, Status => @) OUT(RpowerOk)
NETWORK '原点復帰'
    NO(RpowerOk) NO(Xhome) CALL(homeX, Axis := AxisX, Execute := @, Done => @) OUT(RhomeDone)
NETWORK '100 mm へ移動'
    NO(RpowerOk) NO(Xgo) CALL(mvX, Axis := AxisX, Execute := @, Position := 100.0, Velocity := 50.0,
Done => @) OUT(RmoveDone)
```

命令	.lad 記述	キー	説明
CALL（MC_*／M86_*）	`NO(Xgo) CALL(mvX, Axis := AxisX, Execute := @, Position := 100.0, Velocity := 50.0, Done => @)`	K	PLCopen／Motion86 のファンクションブロック。ファンクションブロックの一覧とパラメーターは 12.7、12.8 節を参照
MC	`PT(Xgo) MC(MC_MoveAbsolute, Naxis, 100, 50, -, -, Rdone, Rbusy, Rerr)`	—	CiA 402 サーボ用の PLCopen モーションボックス（LDmicro 1.x）。機能は MC_Power、MC_MoveAbsolute、MC_MoveRelative、MC_MoveVelocity、MC_Home、MC_Stop、MC_Halt、MC_Reset、MC_ReadActualPosition、MC_ReadStatus、MC_ReadAxisError。使わないパラメーターには - を指定 [1.x]

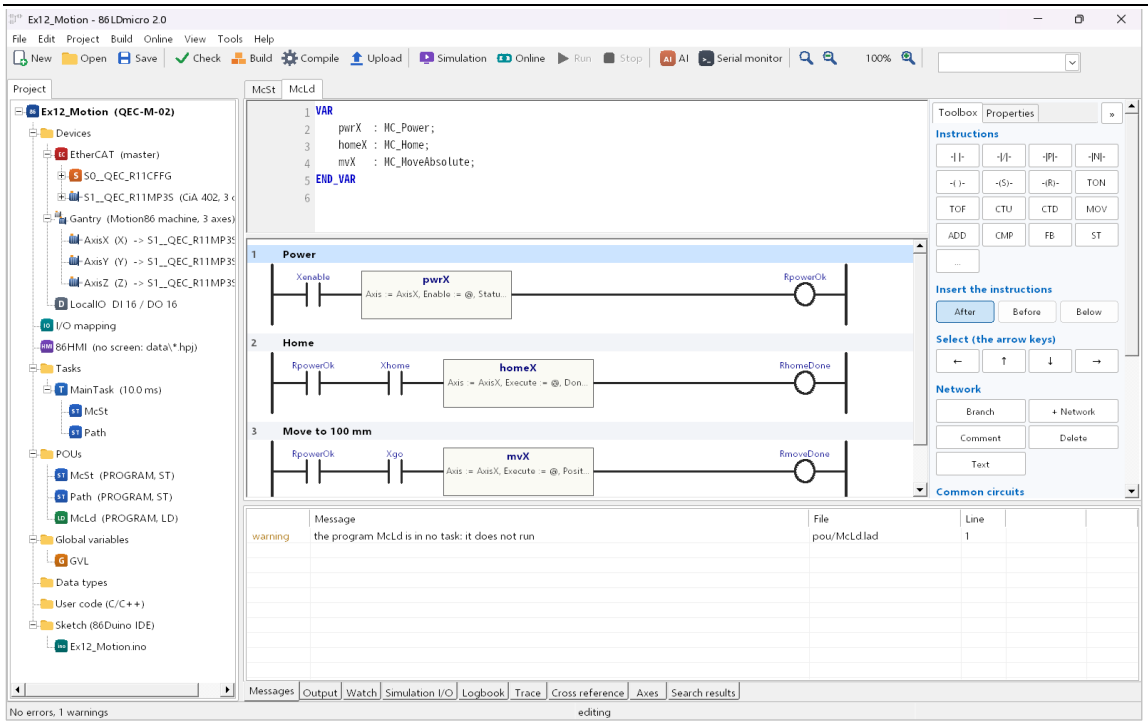


図 7-20　ラダー図の PLCopen ファンクションブロック：MC_Power、MC_Home、MC_MoveAbsolute

*安全：Execute 型のファンクションブロック（MoveAbsolute、Home…）は、Execute の**立上りエッジ**でコマンドを送出します。もう一度送るには、先にパワーフローを遮断してください。

ファンクションブロックのインスタンスは毎スキャン呼び出す必要があります（ネットワークを MCR や GOTO でスキップしないでください）。そうしないと Done／Busy が更新されません。初めて試運転するときは、まずマシンページと軸ページの「Online」で、低速で方向とリミットを確認してください（12.5、12.6 節）。

第 8 章　ST と SFC

本章では、ストラクチャードテキスト（ST）とシーケンシャルファンクションチャート（SFC）のエディターと構文を説明します。ST は計算、比較、ループ、データ処理に適し、SFC は順番にステップを踏む処理に適しています。どちらも IEC 61131-3 の言語で、ラダー図（LD）と同じプロジェクト、同じタスクに置いて順に実行できます。完全な例は 6.5 節（ST：注水制御）と 6.6 節（SFC：シーケンス制御）を参照してください。

8.1　ST エディター

ST プログラムは「Project > Add a PROGRAM (ST)...」で作成します。プロジェクトツリーの POU を右クリックして追加することもできます。名前を入力して **OK** をクリックすると、ファイル `pou/<名前>.st` がすぐに開き、新しい PROGRAM は自動的に最初のタスクに追加されます。

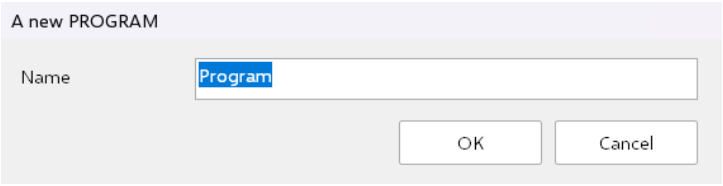


図 8-1　Add a PROGRAM (ST)：プログラム名を入力

ST エディターは 2 つの領域に分かれています。左側はプログラムのテキスト（宣言部 VAR … END_VAR とプログラム本体は同じファイルに書きます）、サイドパネルは「Toolbox」の **ST snippets** です。

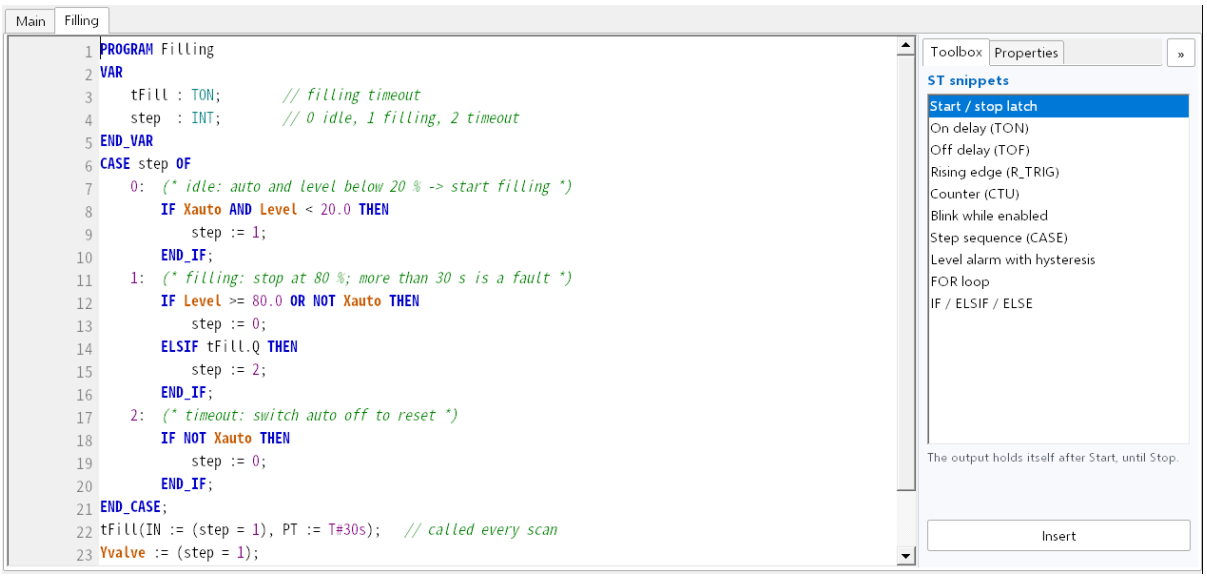


図 8-2　ST エディター：左側がプログラムのテキスト、右側が「ST snippets」

8.1.1 構文の色分け

エディターは構文に従って自動的に色分けするため、入力ミスを見つけやすくなります。

色	内容	例
青の太字	キーワード	IF、THEN、END_IF、VAR、CASE
青緑	データ型	BOOL、INT、REAL、TON
緑の斜体	コメント	(* … *)、// …
濃い赤	文字列	'G1 X10 Y10'
紫	数値、時間定数	100、16#FF、T#2s
黒	変数名、その他	Level、tFill

エラーのある箇所には下線が引かれ、マウスを下線の上に置くとエラーの説明が表示されます。**Check**（Shift+F7）をクリックすると、エラーは「Messages」ウィンドウにも一覧表示されます（9.6 節）。



図 8-3　エラーの下線：マウスを置くとエラーの説明を表示

8.1.2　入力補完とパラメーターヒント

変数やキーワードを入力すると補完リストが自動的に開き、一致する名前とその型、出所（GVL またはローカル変数）、コメントが表示されます。

- 名前の**最初の 2 文字**を入力すると自動的に開きます。`.`（例：`tFill.` で Q、ET などのメンバーを表示）、`#`、`(` を入力したときも開きます。
- ↑↓（または PageUp／PageDown）で選択し、**Enter** または **Tab** で挿入、**Esc** で閉じます（同じ語では再び自動的に開きません）。
- ファンクションブロックやファンクションの `(` または `,` を入力すると、そのパラメーターヒント（例：`TON(IN, PT)`）が表示され、`)` または `;` を入力すると消えます。
- コメントや文字列の中では補完は開きません。

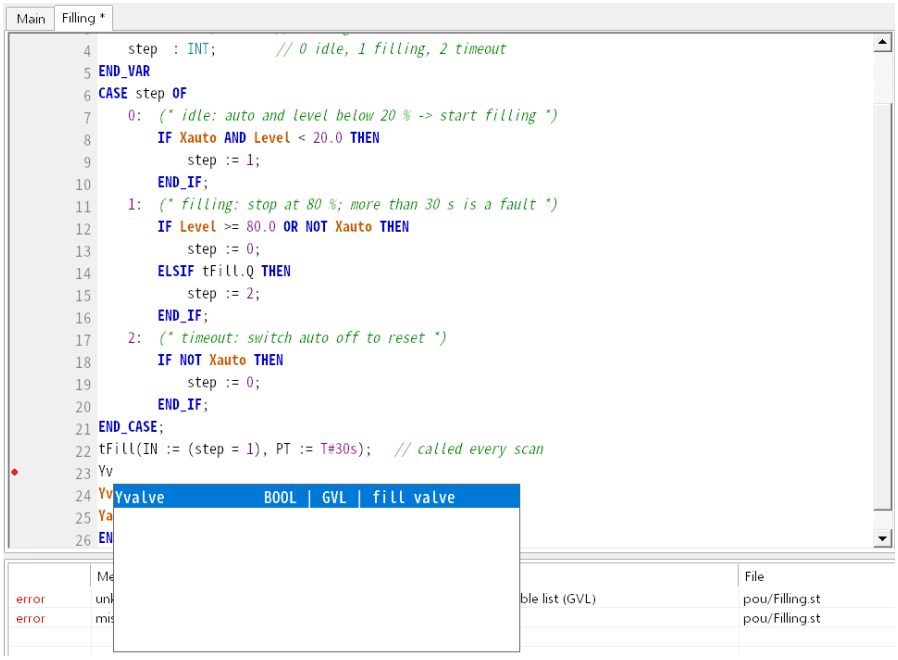


図 8-4　入力補完：Yv と入力すると Yvalve（BOOL、GVL、コメント）を表示

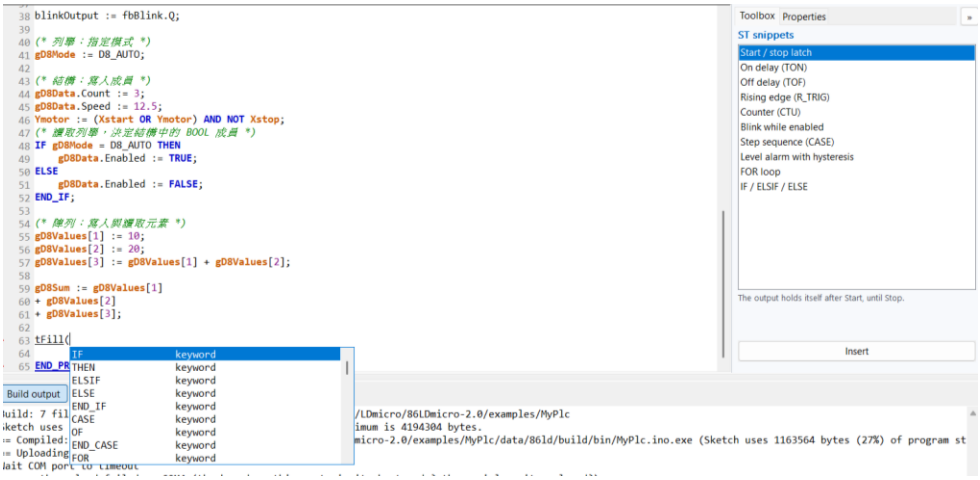


図 8-5　パラメーターヒント：tFill(と入力すると TON の入力を表示

> ***注意：**86LDmicro 2.0.0 では、**Ctrl+Space で補完リストは開きません。**名前の最初の 2 文字（または `.`）を入力すると、リストが自動的に表示されます。

8.1.3 ST snippets

サイドパネルの **ST snippets** には、よく使う書き方のテンプレートがあります。スニペットを選択し（下に説明が表示されます）、**Insert** をクリック（またはダブルクリック）してフォームを開き、変数名を入力して **OK** をクリックすると、コードがカーソル位置に挿入されます。スニペットが必要とするファンクションブロックのインスタンス（例：TON）はプログラムの VAR に、新しいグローバル変数は GVL に自動的に追加されます。

スニペット	用途
Start / stop latch	起動を押すと出力が自己保持し、停止を押すと解除
On delay (TON)	入力が設定時間 ON を続けると出力が ON
Off delay (TOF)	入力が OFF になった後、出力は設定時間だけ保持
Rising edge (R_TRIG)	入力が OFF から ON になるたびに 1 回実行
Counter (CTU)	立上りエッジをカウントし、設定値で出力を ON。リセット可能
Blink while enabled	イネーブルの間、出力は半周期ごとに反転
Step sequence (CASE)	ステートマシン：各ステップが完了すると次のステップへ移る
Level alarm with hysteresis	上限を超えると ON、下限を下回ったときだけ OFF
FOR loop	i を 1 から N まで繰り返し実行
IF / ELSIF / ELSE	Conditional statements

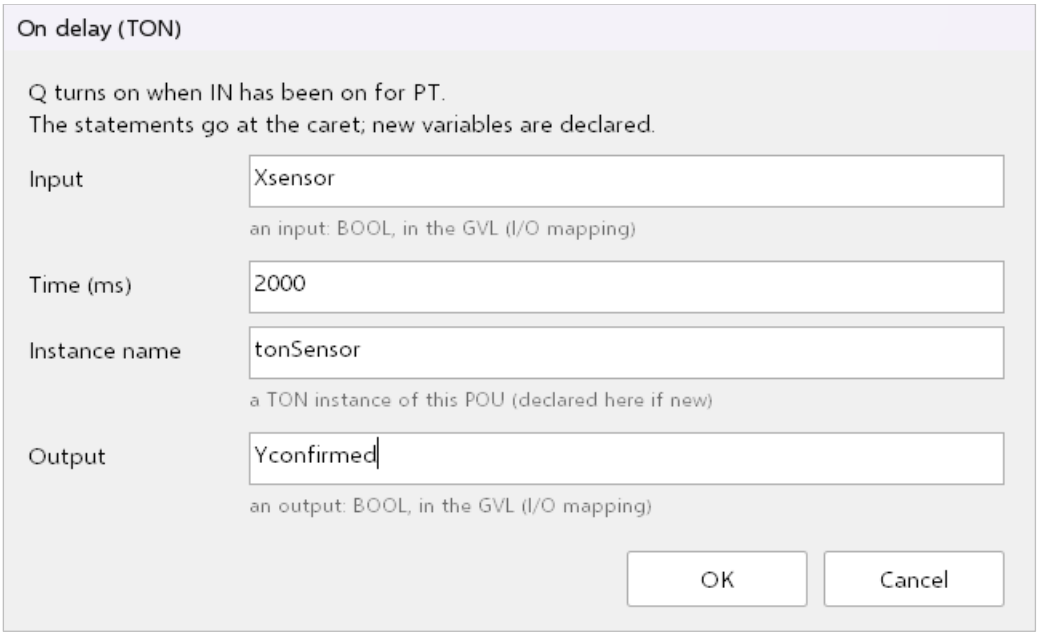


図 8-6 「On delay (TON)」スニペットの挿入：入力、時間、インスタンス名、出力を入力

8.1.4　自動整形、値の表示、その他の操作

操作	キー	説明
自動整形	Ctrl+T	VAR、IF／ELSIF／ELSE、CASE、FOR／WHILE／REPEAT のインデントを整え直す。ステータスバーに「Formatted (Ctrl+T); Ctrl+Z takes it back」と表示
インデント／インデント解除	Tab／Shift+Tab	空白を 4 つ挿入。Shift+Tab で行頭の空白を最大 4 つ削除
自動インデント	Enter	新しい行は前の行のインデントを引き継ぎ、THEN、DO、OF などの後は 1 段深くなる
元に戻す／やり直し	Ctrl+Z／Ctrl+Y	—
検索	Ctrl+F、F3、Shift+F3	Find in this tab（3.7 節）
変数のメニュー	変数を右クリック	Add to watch、Add to the trace、「Where is <変数> used?」、Its declaration、I/O mapping、86HMI binding。シミュレーション中またはオンライン時は、さらに Write、Force

マウスを変数の上に置くと型が表示されます。**シミュレーション中またはオンライン時**は現在の値も表示されるため、変数をウォッチに追加する必要はありません。

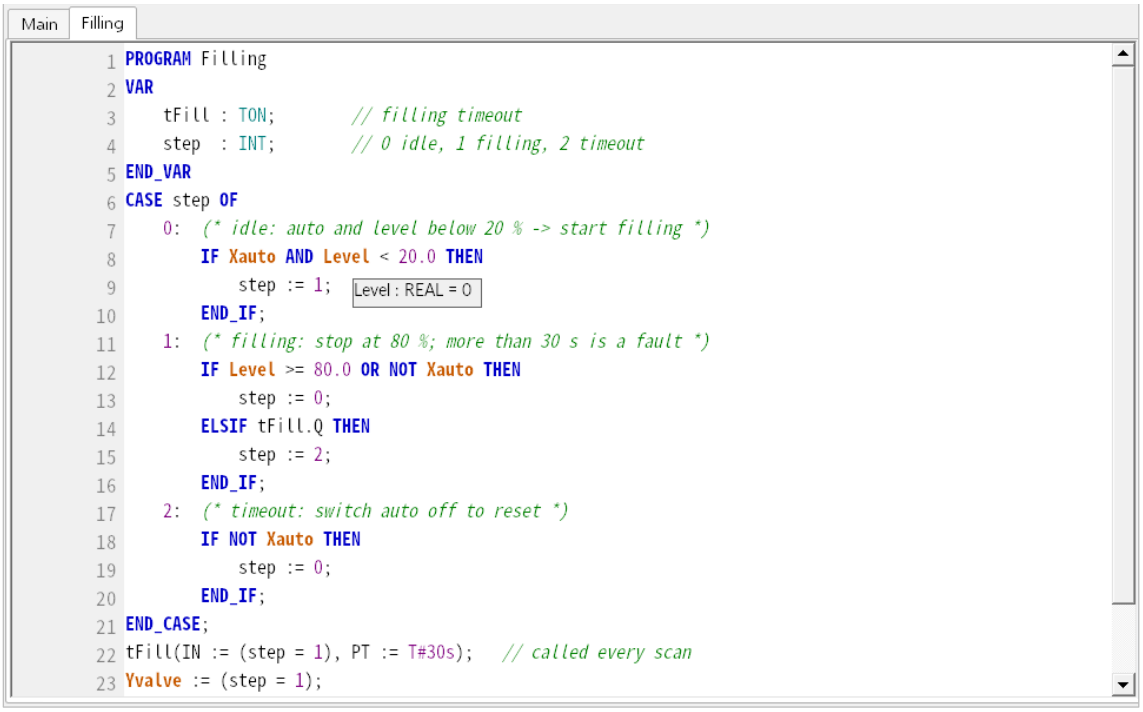


図 8-7　シミュレーション中：マウスを変数の上に置くと型と現在の値を表示

8.2 ST 構文クイックリファレンス

下表は 86LDmicro でよく使う ST 構文（IEC 61131-3）をまとめたものです。表中の書き方はすべて 86LDmicro 2.0.0 で確認済みです。これらは断片なので、貼り付けてチェックするときは、関係する変数を宣言したプログラムの中に置いてください（例：4.5 節の Recipe、Axis1）。

8.2.1 プログラム構造と宣言

```
PROGRAM 名前                    // または FUNCTION_BLOCK 名前、FUNCTION 名前 : 型
VAR                             // ローカル変数（PROGRAM、FUNCTION_BLOCK の値は次のスキャンまで保持）
    tFill : TON;                // ファンクションブロックのインスタンス
    step  : INT := 0;           // 初期値を指定可能
    buf   : ARRAY[0..9] OF INT; // 配列
    s     : STRING[40];         // 文字列（最大長）
END_VAR
VAR CONSTANT
    MAXN : INT := 10;           // 定数
END_VAR
(* プログラム本体 *)
END_PROGRAM
```

ブロック	使用先	説明
VAR … END_VAR	すべて	ローカル変数
VAR CONSTANT	すべて	定数。書き込み不可
VAR_TEMP	すべて	一時変数：呼び出しのたびに初期値から始まり、次回まで保持されない
VAR_INPUT	FUNCTION、FUNCTION_BLOCK	入力パラメーター
VAR_OUTPUT	FUNCTION、FUNCTION_BLOCK	出力（ファンクションブロック：呼び出し後に インスタンス.名前 で読み取る。ファンクション：呼び出し時に 名前 => 変数 で受け取る）
VAR_IN_OUT	FUNCTION、FUNCTION_BLOCK	渡された変数そのもので、直接書き換えることができる。渡すのは呼び出し側 ST プログラムのローカル変数でなければならない（定数、式、グローバル変数は不可）
VAR_GLOBAL	グローバル変数リスト（GVL）	プロジェクト全体で共有する変数（4.5 節）

PROGRAM では VAR_INPUT、VAR_OUTPUT、VAR_IN_OUT を宣言できません（チェック時にエラーになります）。上表は 86LDmicro 2.0.0 で実際にサポートされている範囲です。

8.2.2 文

項目	書き方
代入	`x := y + 1;`
コメント	`(* 複数行 *)`、`// 1 行`
条件分岐	`IF a THEN ... ELSIF b THEN ... ELSE ... END_IF;`
多分岐	`CASE step OF 0: ...; 1, 2: ...; 10..20: ...; ELSE ... END_CASE;`
FOR ループ	`FOR i := 1 TO 10 BY 2 DO ... END_FOR;`
WHILE ループ	`WHILE n > 0 DO ... END_WHILE;`
REPEAT ループ	`REPEAT ... UNTIL n >= 3 END_REPEAT;`
ループを抜ける	`EXIT;`
今回の実行を終了	`RETURN;`（PROGRAM は今回のスキャンでそれ以降を実行しない）
ファンクションブロックの呼び出し	`t1(IN := start, PT := T#2s);  IF t1.Q THEN ... END_IF;`
ファンクションの呼び出し	`y := LIMIT(0.0, x, 100.0);  n := REAL_TO_INT(r);`
配列、構造体	`buf[i] := 0;  Recipe[2] := 10;  Axis1.pos := 10.0;`

8.2.3 演算子と定数

分類	書き方
算術	`+  -  *  /  MOD`（整数を 0 で割ると結果は 0）
比較	`=  <>  <  <=  >  >=`
論理（BOOL）とビット（WORD など）	`AND  OR  XOR  NOT`
整数	`100`、`-5`、`16#FF`（16 進数）、`2#1010`（2 進数）
実数	`1.5`、`1.5E3`
時間	`T#2s`、`T#1s500ms`、`T#100ms`
ブール、文字列	`TRUE`、`FALSE`、`'テキスト'`

8.2.4 よく使うファンクションとファンクションブロック

分類	名前
数学	ABS、SQRT、SIN、COS、EXPT、TRUNC
選択と制限	MIN、MAX、LIMIT、SEL、MUX
ビット	SHL、SHR、ROL、ROR
文字列	LEN、CONCAT、LEFT、RIGHT、MID、FIND
型変換	<型>_TO_<型>。例：`INT_TO_REAL`、`REAL_TO_DINT`、`INT_TO_STRING`、`STRING_TO_REAL`
タイマー	TON（オンディレイ）、TOF（オフディレイ）、TP（パルス）
カウンター	CTU、CTD、CTUD
エッジ	R_TRIG（立上りエッジ）、F_TRIG（立下りエッジ）
フリップフロップ	SR（セット優先）、RS（リセット優先）
モーション制御	MC_*（PLCopen、12.7 節）、M86_*（Motion86 マシン、12.8 節）

*ヒント：タイマーが時間を積算するには、ファンクションブロックのインスタンス（例：TON）を**毎スキャン呼び出す**必要があります。呼び出しを IF や CASE の中に書くと、条件が成立しないスキャンでは更新されません。確実な書き方は、IF／CASE の外で呼び出し、条件を入力にすることです。例：`tFill(IN := (step = 1), PT := T#30s);`。

8.3 ユーザー定義の FUNCTION と FUNCTION_BLOCK

繰り返し使う処理は独自の FUNCTION または FUNCTION_BLOCK として書き、組み込みのファンクションやファンクションブロックと同じように ST、LD、SFC から呼び出せます。

種類	メモリー	作成方法	使用方法
FUNCTION	インスタンスの状態を持たない：呼び出すたびに最初から計算する。計算や換算に適する（パラメーターだけで計算する場合、同じパラメーターからは同じ結果が得られる）	「Project > Add a FUNCTION (ST)...」	`y := Scale(raw, 0.0, 100.0);`
FUNCTION_BLOCK	あり：インスタンスごとに内部変数（例：タイマー）を保持	「Project > Add a FUNCTION_BLOCK (ST)...」または「Add a FUNCTION_BLOCK (ladder)...」	先にインスタンス `bl : Blinker;` を宣言してから `bl(...)`

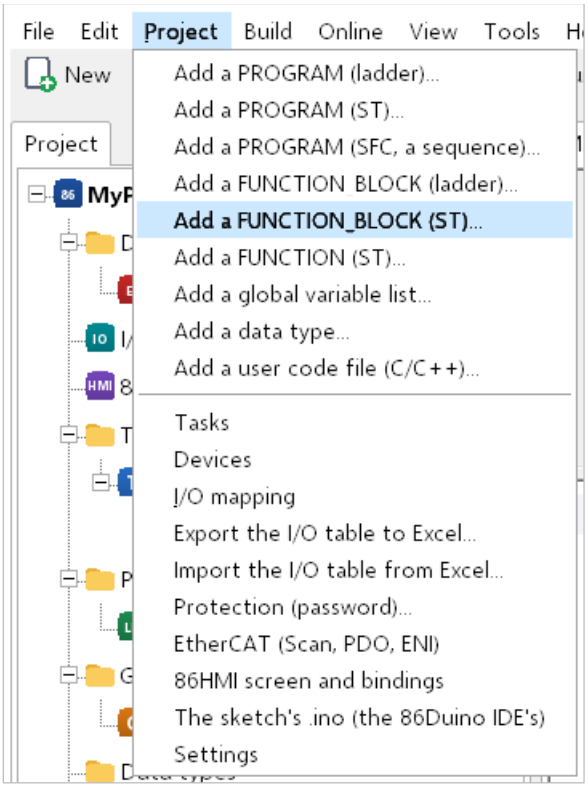


図 8-8 「Project」メニュー：Add a FUNCTION_BLOCK (ST) と Add a FUNCTION (ST)

8.3.1 例：アナログ換算と点滅

シナリオ：アナログ入力の生の値 `AnalogIn`（0〜4095）を液位のパーセント値 `Level` に換算します。運転中（`Rrun`）は、表示灯 `Yblink` を 0.5 秒ごとに点灯・消灯させます。換算は FUNCTION `Scale`、点滅は FUNCTION_BLOCK `Blinker` として書き、ST プログラム `Panel` から呼び出します。

変数	型	I/O マッピング	用途
Rrun	BOOL	—（ほかのプログラムが書き込む）	運転中
AnalogIn	INT	アナログ入力（4.6 節）	アナログの生の値 0〜4095
Level	REAL	—	液位（%）
Yblink	BOOL	デジタル出力	点滅灯

FUNCTION Scale（`pou/Scale.st`）：ファンクション名がそのまま戻り値で、ファンクション内でそれに代入します。

```
FUNCTION Scale : REAL                  // アナログ換算：0〜4095 → lo〜hi
VAR_INPUT
    raw : INT;
    lo, hi : REAL;
END_VAR
Scale := lo + (hi - lo) * INT_TO_REAL(raw) / 4095.0;
END_FUNCTION
```

FUNCTION_BLOCK Blinker（`pou/Blinker.st`）：内部の TON と出力 `Out` はインスタンスごとに個別に保持されます。

```
FUNCTION_BLOCK Blinker                 // 点滅：メモリーを持つファンクションブロック
VAR_INPUT
    Enable : BOOL;
    Half   : TIME;
END_VAR
VAR_OUTPUT
    Out : BOOL;
END_VAR
VAR
    t : TON;
END_VAR
t(IN := Enable AND NOT t.Q, PT := Half);   // 半周期ごとに反転
IF t.Q THEN
    Out := NOT Out;
END_IF;
IF NOT Enable THEN
    Out := FALSE;
END_IF;
END_FUNCTION_BLOCK
```

PROGRAM Panel（pou/Panel.st、タスクに入れて実行）：

```
PROGRAM Panel
VAR
    bl : Blinker;                       // ファンクションブロックのインスタンス
END_VAR
Level := Scale(AnalogIn, 0.0, 100.0);  // 0〜4095 → 0〜100 %
bl(Enable := Rrun, Half := T#500ms);
Yblink := bl.Out;
END_PROGRAM
```

同じファンクションブロックは、ラダー図でも **CALL** で使用できます（7.8 節）。次のネットワークでは別のインスタンス `bl2` を宣言し、パワーフローを Enable に接続し、Out でコイル `Ylamp2` を駆動します。

```
PROGRAM Main
VAR
    bl2 : Blinker;
END_VAR
NETWORK 'ユーザー定義ファンクションブロック：点滅'
    NO(Rrun) CALL(bl2, Enable := @, Half := T#250ms, Out => @) OUT(Ylamp2)
END_PROGRAM
```

#	操作
1	「Project > Add a FUNCTION (ST)...」で名前を `Scale` にし、上のプログラムを入力します。
2	「Project > Add a FUNCTION_BLOCK (ST)...」で名前を `Blinker` にし、上のプログラムを入力します。
3	GVL で Rrun、AnalogIn、Level、Yblink（Ylamp2）を宣言します。
4	「Project > Add a PROGRAM (ST)...」で名前を `Panel` にし、上のプログラムを入力します（自動的に最初のタスクに追加されます）。
5	**Check**（Shift+F7）をクリックし、「Messages」にエラーがないことを確認します。
6	**Simulate**（F5）をクリックし、Watch ウィンドウに Rrun、AnalogIn、Level、Yblink を追加します。AnalogIn に 2048、Rrun に TRUE を書き込みます。

期待される結果：AnalogIn = 2048 のとき Level は約 50.01 になります。Rrun が TRUE のとき Yblink は約 0.5 秒ごとに反転します（ラダー図の Ylamp2 は約 0.25 秒ごとに反転）。Rrun が FALSE のときはどちらも消灯します。

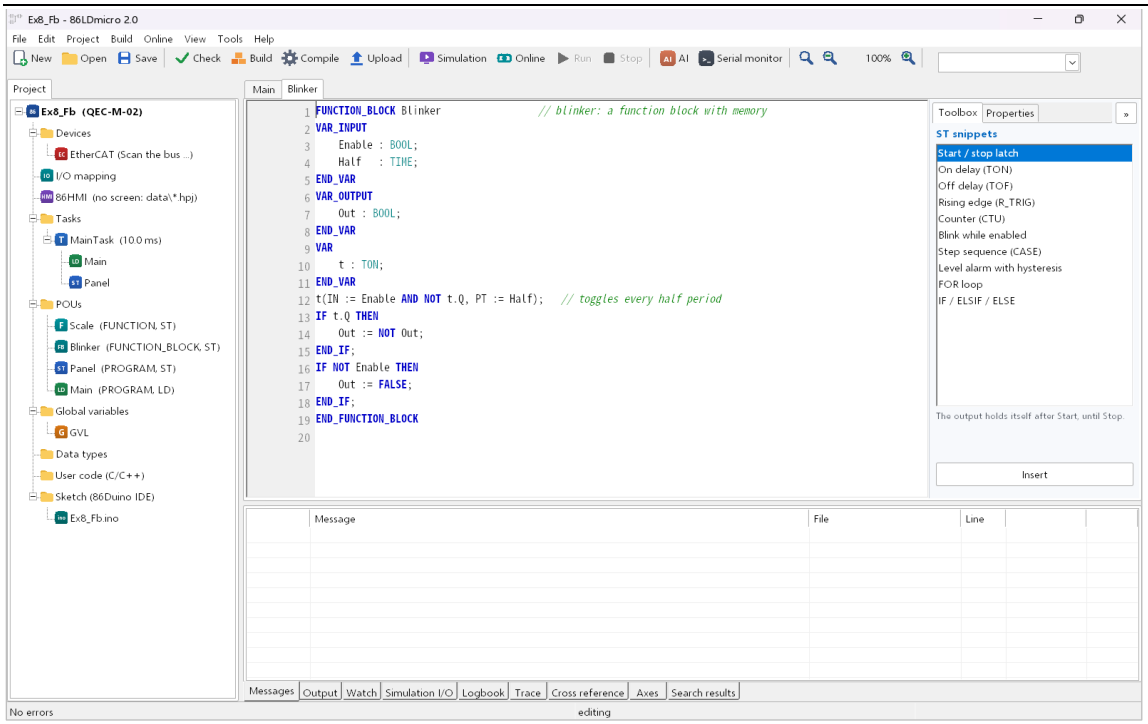


図 8-9　FUNCTION_BLOCK Blinker の編集画面

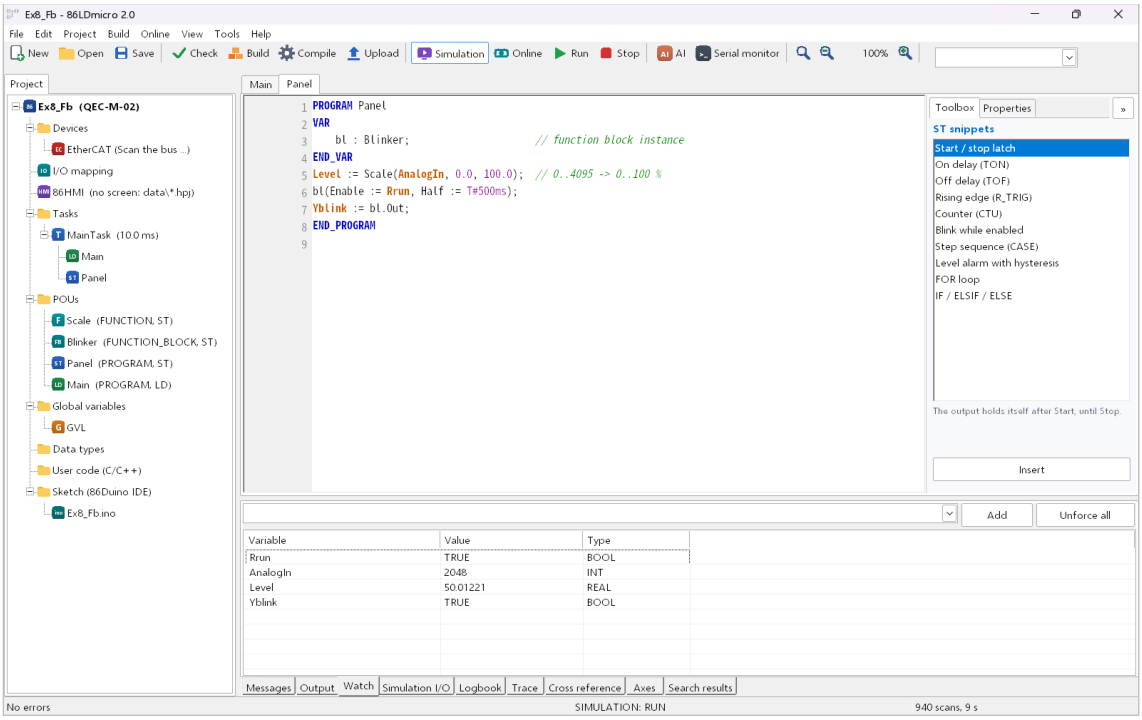


図 8-10　シミュレーション中：Level は 50.01、Yblink は点滅

***ヒント：**FUNCTION のローカル変数は呼び出しのたびに初期値から始まります（保持されません）。タイマー、カウンター、エッジ検出など、メモリーを持つ機能が必要な場合は FUNCTION_BLOCK として書いてください。FUNCTION と FUNCTION_BLOCK はタスクには入れず、それを呼び出すプログラムが実行のタイミングを決めます。

8.4 SFC エディター

SFC は**ステップ**（Step）と**トランジション**（Transition）で処理の流れを記述します。アクティブなステップだけがそのアクションを実行し、トランジションの条件が成立すると次のステップへ移ります。マシンの自動シーケンスやレシピ手順などに適しています。

「Project > Add a PROGRAM (SFC, a sequence)...」またはプロジェクトツリーの POU の右クリックで作成します。ファイルは `pou/<名前>.sfc` で、LD や ST と同様にタスクに入れて実行します。新しい SFC には簡単なチャート（初期ステップ Init → ステップ Step1 → 1 秒後に Init へ戻る）がすでに入っており、そのまま修正できます。

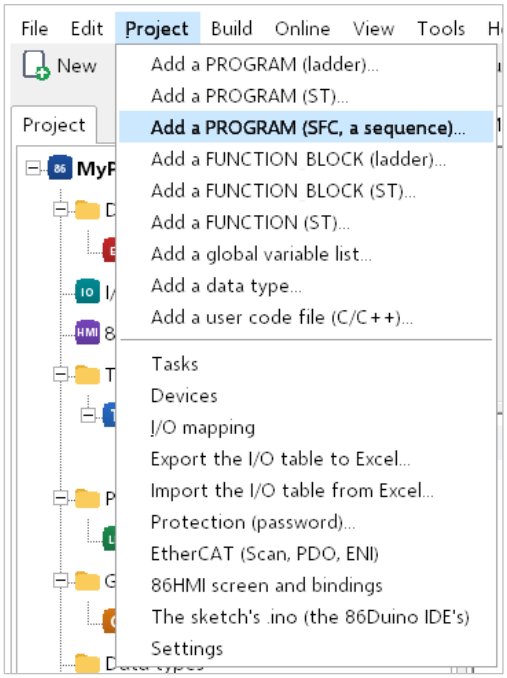


図 8-11　「Project > Add a PROGRAM (SFC, a sequence)...」

SFC エディターは 3 つの領域に分かれています。上部が**宣言部**（VAR … END_VAR）、中央が**チャート**、サイドパネルが「**Chart**」ツールボタンです。

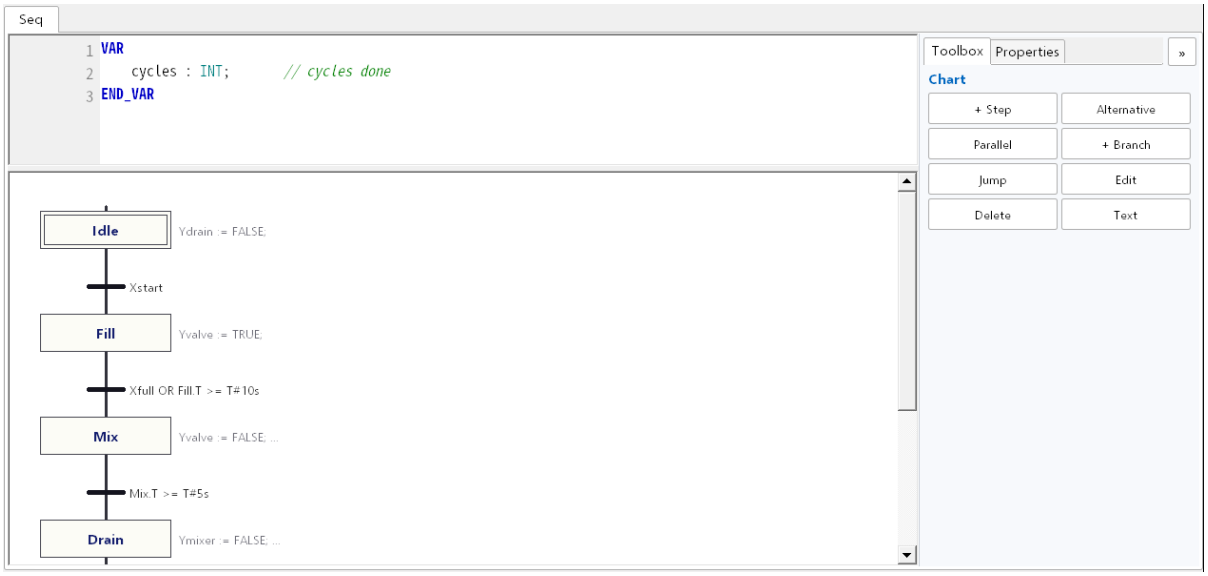


図 8-12　SFC エディター：上部に宣言部、中央にチャート、右側に「Chart」ツール

8.4.1 「Chart」ツール

ボタン	キー	機能
+ Step	Ins	選択箇所の後にトランジションとステップを追加
Alternative	—	ステップを選択してクリック：ステップの後で分岐。左から右へ評価し、最初に成立したトランジションが有効
Parallel	—	トランジションを選択してクリック：すべての分岐を同時に実行し、次のトランジションで合流
+ Branch	—	選択した選択分岐／並列分岐にもう 1 本分岐を追加
Jump	—	分岐またはチャートの最後のトランジションの後に置く：あるステップへ戻る（またはジャンプする）
Edit	Enter、ダブルクリック	ステップの名前とアクション、トランジションの条件を編集
Delete	Del	選択項目を削除（ステップはそのトランジションも含めて削除）
Text	Ctrl+T	チャートを .sfc テキストで表示・編集。「Back to the diagram」をクリックすると構文をチェック

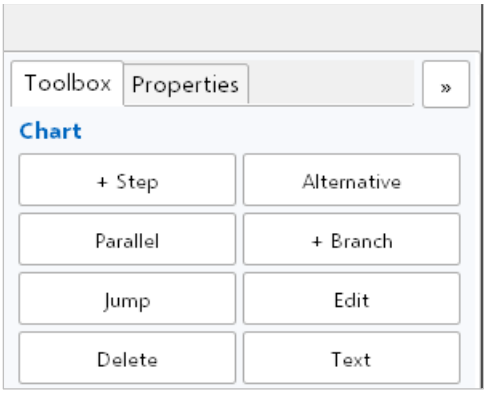


図 8-13　サイドパネルの「Chart」ツールボタン

チャートを右クリックしても同じコマンドがあります。その他の操作：

- 方向キー：↑↓ で同じシーケンスの前後の項目を、← → で隣の分岐を選択します。
- Ctrl+Z／Ctrl+Y：元に戻す／やり直し。
- シミュレーション中またはオンライン時はチャートを変更できません（先にシミュレーションを停止するか、接続を切断してください）。
- プロジェクトがパスワードで保護されている場合、SFC は読み取り専用になります（15.6 節）。

8.4.2　テキストモード

Text（Ctrl+T）をクリックすると、チャート全体の .sfc テキストを直接編集できます（書式は 8.5.4 を参照）。大量の修正や、ほかの SFC からのコピーに便利です。**Back to the diagram** をクリックすると構文がチェックされ、エラーがあると通知されてテキストモードのままになります。

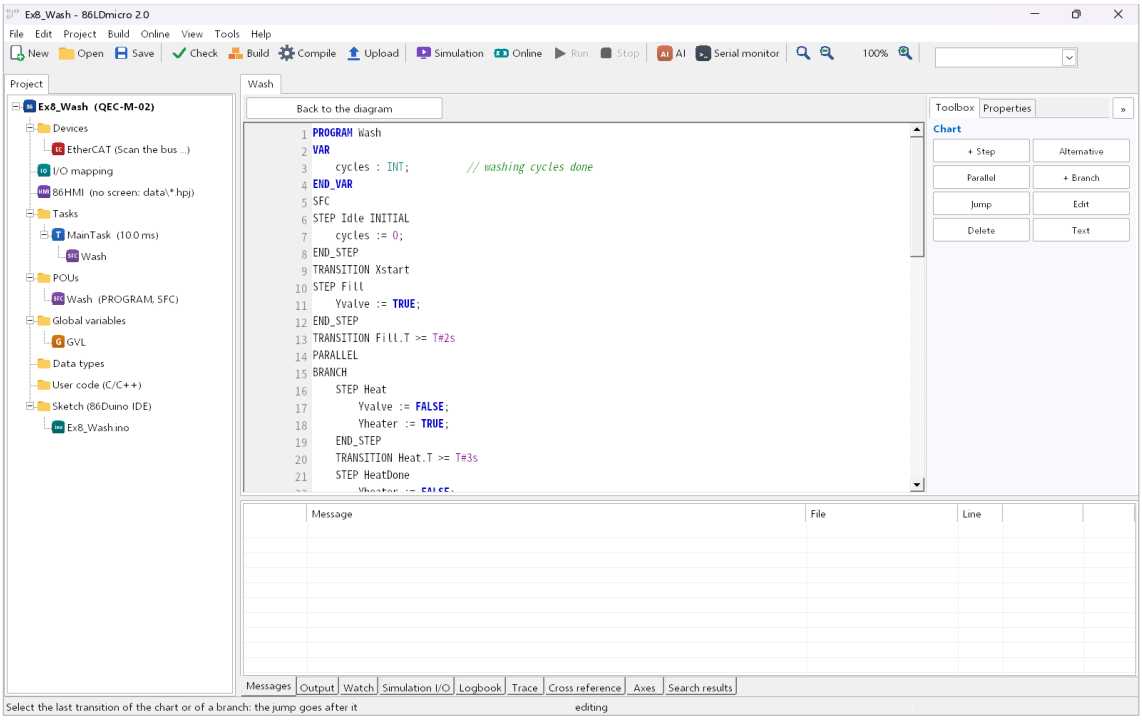


図 8-14　SFC のテキストモード：チャート全体の .sfc テキスト

8.5　ステップ、トランジション、分岐

8.5.1　ステップ

ステップを選択して **Edit**（Enter またはダブルクリック）をクリックすると、「Step」ウィンドウが開きます。

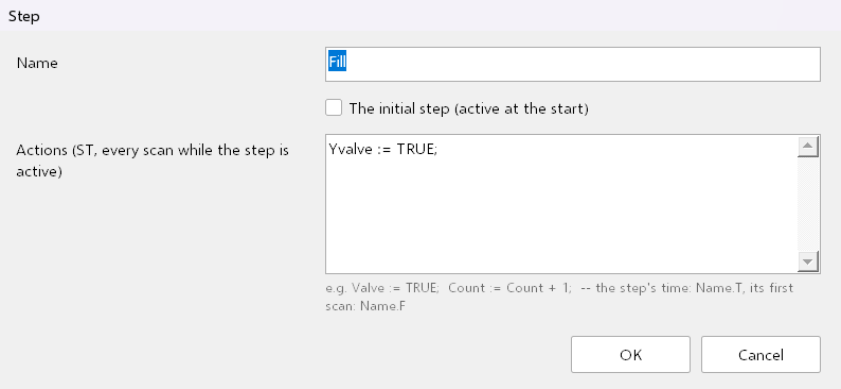


図 8-15　ステップ：名前、初期ステップ、アクション

欄	説明
Name	英字、数字、_。名前を変更すると、そのステップへのジャンプ（JUMP）も追従して変わる
The initial step (active at the start)	PLC の実行開始時にアクティブになるステップで、二重枠で表示される。1 つのチャートに 1 つだけ
Actions (ST, every scan while the step is active)	ST の文。例：`Valve := TRUE;  Count := Count + 1;`。アクションの 1 行目がチャートのステップの右側に表示される

8.5.2　トランジション

トランジションを選択して **Edit** をクリックすると、「Transition」ウィンドウが開きます。条件は ST の式で、`TRUE`、BOOL 変数、または結果が BOOL になる任意の式を指定します。

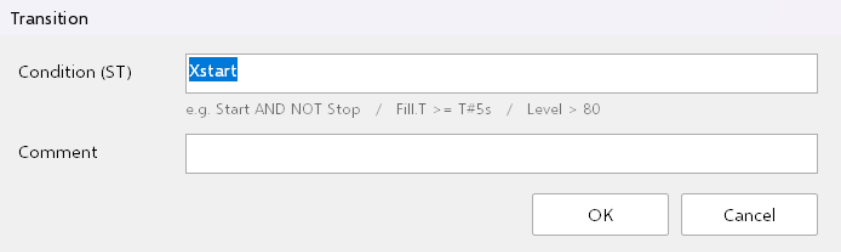


図 8-16　トランジション：条件（ST）とコメント

条件	意味
`Start AND NOT Stop`	起動が押され、かつ停止が押されていない
`Fill.T >= T#5s`	ステップ Fill がアクティブになって 5 秒経過
`Level > 80`	液位が 80 を超えた
`mv.Done`	ファンクションブロック（例：MC_MoveAbsolute）が完了
`TRUE`	直ちに遷移（次のスキャン）

8.5.3　分岐とジャンプ

構造	作成方法	実行方法
選択分岐（ALTERNATIVE）	ステップを選択 → **Alternative**。各分岐はトランジションで始まる。**+ Branch** でさらに 1 本追加	左から右へ評価し、**最初に成立した**トランジションの分岐が有効になる（左が優先）
並列分岐（PARALLEL）	トランジションを選択 → **Parallel**。各分岐はステップで始まる。**+ Branch** でさらに 1 本追加	すべての分岐が**同時に**開始する。各分岐の最後のステップがアクティブになり、かつ次のトランジションが成立すると合流する
ジャンプ（JUMP）	分岐またはチャートの最後のトランジションを選択 → **Jump...** で、ジャンプ先のステップを選択	トランジションが成立すると、指定したステップへジャンプする（戻ってループを作ることもできる）

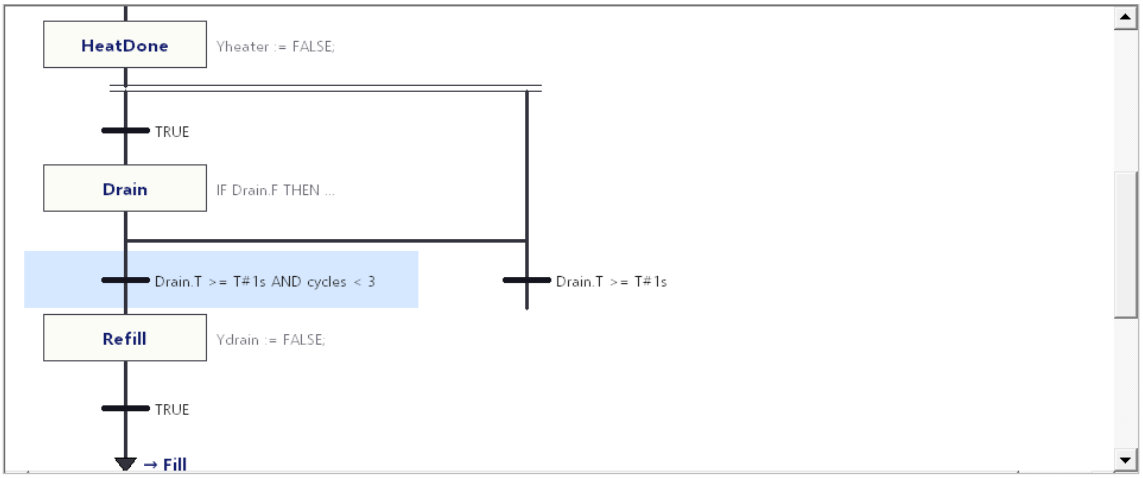


図 8-17　選択分岐：Drain の後に 2 つの経路。左は Fill へ戻り、右は Done へ進む

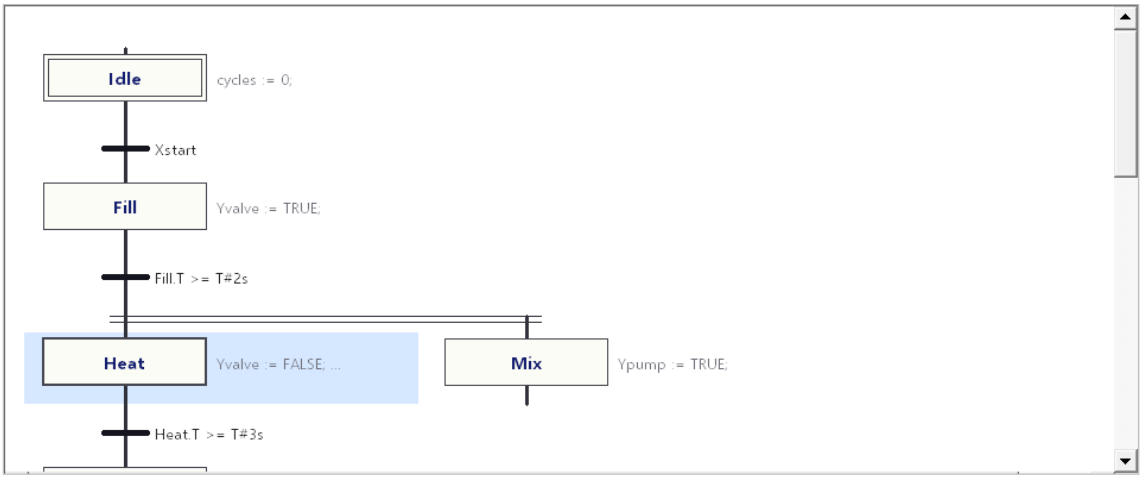


図 8-18　並列分岐：Heat と Mix を同時に実行し、TRUE のトランジションで合流

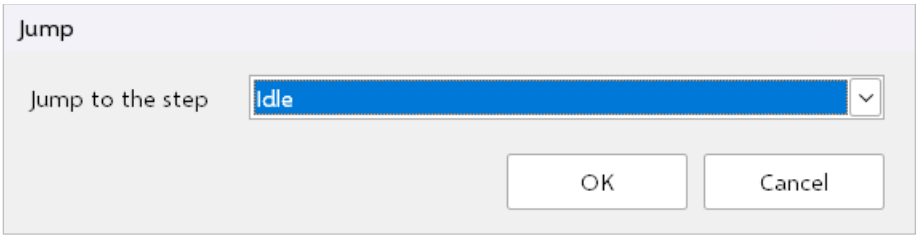


図 8-19　ジャンプ：ジャンプ先のステップを選択

構造が正しくない場合（例：2 つのステップが直接つながっている、並列分岐がトランジションで始まっている）、エラーは下の「Messages」ウィンドウに一覧表示されます（エラー内容、ファイル、行番号）。ダブルクリックするとその行へ移動します。

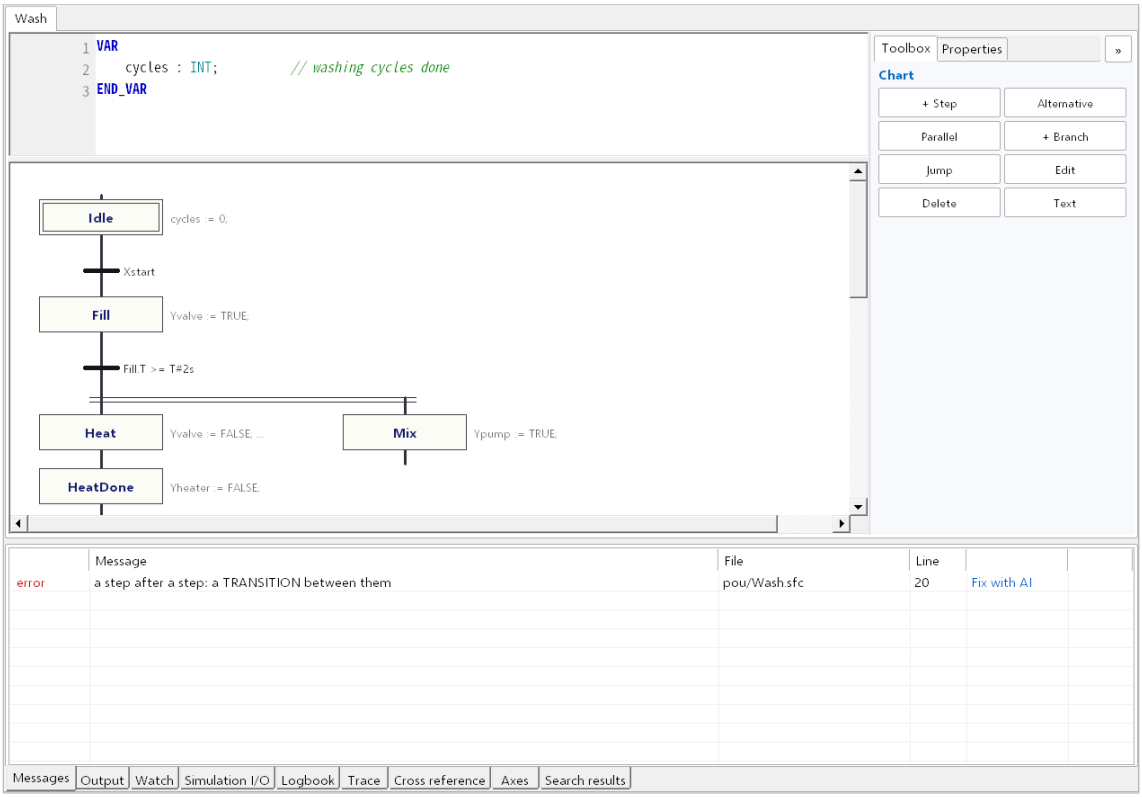


図 8-20　構造エラー：下の「Messages」にエラー、ファイル、行番号を表示

8.5.4　例：洗浄工程（選択分岐と並列分岐を含む）

シナリオ：スイッチ `Xstart` をオンにすると、給水 2 秒 → 加熱 3 秒と撹拌を同時に実行 → 排水 1 秒、を行います。3 回繰り返すと完了灯が点灯し、スイッチをオフにすると待機に戻ります。

変数	型	I/O マッピング	用途
Xstart	BOOL	デジタル入力	起動スイッチ
Yvalve	BOOL	デジタル出力	給水弁
Yheater	BOOL	デジタル出力	ヒーター
Ypump	BOOL	デジタル出力	撹拌ポンプ
Ydrain	BOOL	デジタル出力	排水弁
Ydone	BOOL	デジタル出力	完了灯
cycles	INT	—（Wash のローカル変数）	完了した洗浄回数

SFC エディターで **Text**（Ctrl+T）をクリックし、以下の内容を貼り付けてから **Back to the diagram** をクリックします。

```
PROGRAM Wash
VAR
    cycles : INT;          // 完了した洗浄回数
END_VAR
SFC
STEP Idle INITIAL
    cycles := 0;
END_STEP
TRANSITION Xstart
STEP Fill
    Yvalve := TRUE;
END_STEP
TRANSITION Fill.T >= T#2s
PARALLEL
BRANCH
    STEP Heat
        Yvalve := FALSE;
        Yheater := TRUE;
    END_STEP
    TRANSITION Heat.T >= T#3s
    STEP HeatDone
        Yheater := FALSE;
    END_STEP
BRANCH
    STEP Mix
        Ypump := TRUE;
    END_STEP
END_PARALLEL
TRANSITION TRUE
STEP Drain
    IF Drain.F THEN
        cycles := cycles + 1;
    END_IF;
    Yvalve := FALSE;
    Ypump := FALSE;
    Ydrain := TRUE;
END_STEP
ALTERNATIVE
BRANCH
    TRANSITION Drain.T >= T#1s AND cycles < 3
```

```
    STEP Refill
        Ydrain := FALSE;
    END_STEP
    TRANSITION TRUE
    JUMP Fill
BRANCH
    TRANSITION Drain.T >= T#1s
END_ALTERNATIVE
STEP Done
    Ydrain := FALSE;
    Ydone := TRUE;
END_STEP
TRANSITION NOT Xstart
STEP Reset
    Ydone := FALSE;
END_STEP
TRANSITION TRUE
JUMP Idle
END_SFC
END_PROGRAM
```

テキスト	対応する図形
`STEP 名前 [INITIAL] … END_STEP`	ステップとそのアクション。INITIAL は初期ステップ
`TRANSITION 条件`	トランジション
`ALTERNATIVE BRANCH … BRANCH … END_ALTERNATIVE`	選択分岐（各分岐は TRANSITION で始まる）
`PARALLEL BRANCH … BRANCH … END_PARALLEL`	並列分岐（各分岐は STEP で始まる）
`JUMP ステップ`	ジャンプ
`SFC … END_SFC`	チャートの開始と終了（END_PROGRAM の前）

#	操作
1	「Project > Add a PROGRAM (SFC, a sequence)...」で名前を `Wash` にします。
2	GVL で Xstart、Yvalve、Yheater、Ypump、Ydrain、Ydone（BOOL）を宣言します。
3	**Text**（Ctrl+T）をクリックし、内容全体を上のテキストで置き換えて、**Back to the diagram** をクリックします。
4	**Check**（Shift+F7）をクリックし、エラーがないことを確認します。
5	**Simulate**（F5）をクリックし、Watch ウィンドウで Xstart に TRUE を書き込んで、チャートと出力を観察します。

期待される結果：Fill がアクティブになり、Yvalve が ON になります。2 秒後に Heat と Mix が同時にアクティブになります（Heat で Yvalve を OFF、Yheater と Ypump が ON）。3 秒後に Heat が終了して HeatDone に入り（Yheater OFF）、並列分岐は Drain で合流します（Ypump OFF、Ydrain ON、cycles に 1 加算）。1 秒後、cycles が 3 未満なら左の分岐で Fill に戻り、3 回目は右の分岐で Done に進みます（Ydone ON）。Xstart に FALSE を書き込むと Idle に戻ります。全体で約 18 秒かかります。

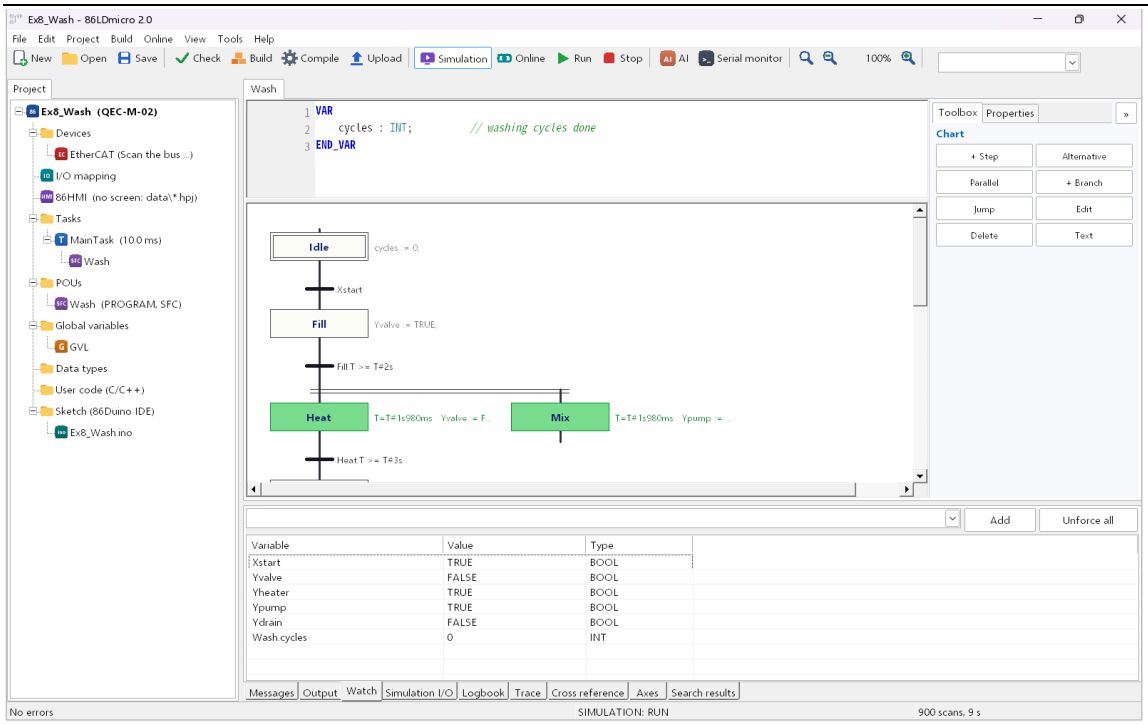


図 8-21　シミュレーション中の Wash：アクティブなステップは緑色で、右側にアクティブになってからの時間を表示

8.6　SFC の実行セマンティクス

SFC が各スキャンで何をしているかを理解すると、予測どおりに動く処理を書くことができます。

8.6.1　各スキャンの処理順序

1. **スキャン開始時**のステップの状態で、すべてのアクティブなステップの後にあるトランジションを評価します。
2. 成立したトランジションを発火させます：前のステップを抜け、次のステップをアクティブにします（同じステップの後の選択分岐では**左が優先**）。
3. すべてのアクティブなステップのアクションを実行します。

したがって、1 スキャンで進むのは最大 1 ステップです。`TRUE` のトランジションは次のスキャンで通過します。

8.6.2　ステップフラグ

書き方	型	意味	使用先
ステップ`.X`	BOOL	ステップがアクティブ	同じ SFC の条件とアクション。例：`Fill.X`
ステップ`.T`	TIME	ステップがアクティブになってからの時間（抜けるとゼロに戻る）	タイムアウト、遅延遷移。例：`Fill.T >= T#5s`
ステップ`.F`	BOOL	ステップがアクティブになった後の**最初のスキャン**で TRUE	1 回だけ行う動作。例：カウント `IF Drain.F THEN … END_IF;`
プログラム`.`ステップ`_X`	BOOL	**ほかの POU** からステップがアクティブかどうかを読み取る	例：`Wash.Done_X`。Watch ウィンドウでもこの書き方を使う

8.6.3　出力は保持される

アクションで設定した変数は、ほかのコードで変更されるまでその値を保持します（CODESYS の通常のアクションの書き方と同様です）。ステップを抜けても、86LDmicro はそれを自動的に FALSE に戻しません。オフにする必要がある場合は、後続のステップで := FALSE と書きます（上の例では、Heat ステップの最初で Yvalve を OFF にし、Drain ステップで Ypump を OFF にしています。オフにしないと、加熱・撹拌の間も給水弁が開いたままになります）。

> ***ヒント：**ファンクションブロック（例：MC_MoveAbsolute）をステップのアクションに置くと、ステップを抜けた後は呼び出されなくなります。次のステップでその Execute を FALSE に戻しておくと、次回そのステップに戻ったときに新しい立上りエッジが得られます。常に呼び出す必要があるファンクションブロック（例：MC_Power）は、別の ST または LD プログラムで常時呼び出すことを推奨します（12.7 節）。

8.6.4　シミュレーションとオンラインでの表示

シミュレーション（F5）またはオンライン（Ctrl+F5）中は、アクティブなステップが**緑色**で表示され、右側にアクティブになってからの時間が表示されます。Watch ウィンドウには プログラム.ステップ_X を追加できます。



図 8-22　シミュレーション中：アクティブなステップは緑色で、右側にアクティブになってからの時間が表示される

第 9 章　シミュレーションとデバッグ

86LDmicro にはシミュレーターが内蔵されており、コントローラーを接続しなくても PC 上でプロジェクト全体を実行し、入力を操作して出力を確認できます。この章では、シミュレーション、Simulation I/O、ウォッチと強制、Trace、Cross reference と検索、およびよくあるエラーメッセージについて説明します。ウォッチ、強制、Trace の操作は、**オンライン**（コントローラーに接続した状態、10.4 節）でもまったく同じです。

9.1　シミュレーション

ツールバーの **Simulate**（F5）をクリックするか、「Online > Simulate」を選択するとシミュレーションが始まります。もう一度 F5 を押すと終了します。シミュレーションはプロジェクト全体を実行します。すべてのタスク、I/O プロセスイメージ、タイマー、シリアルポート、EEPROM、軸、Motion86 マシンが、コントローラー上と同じように動作します。

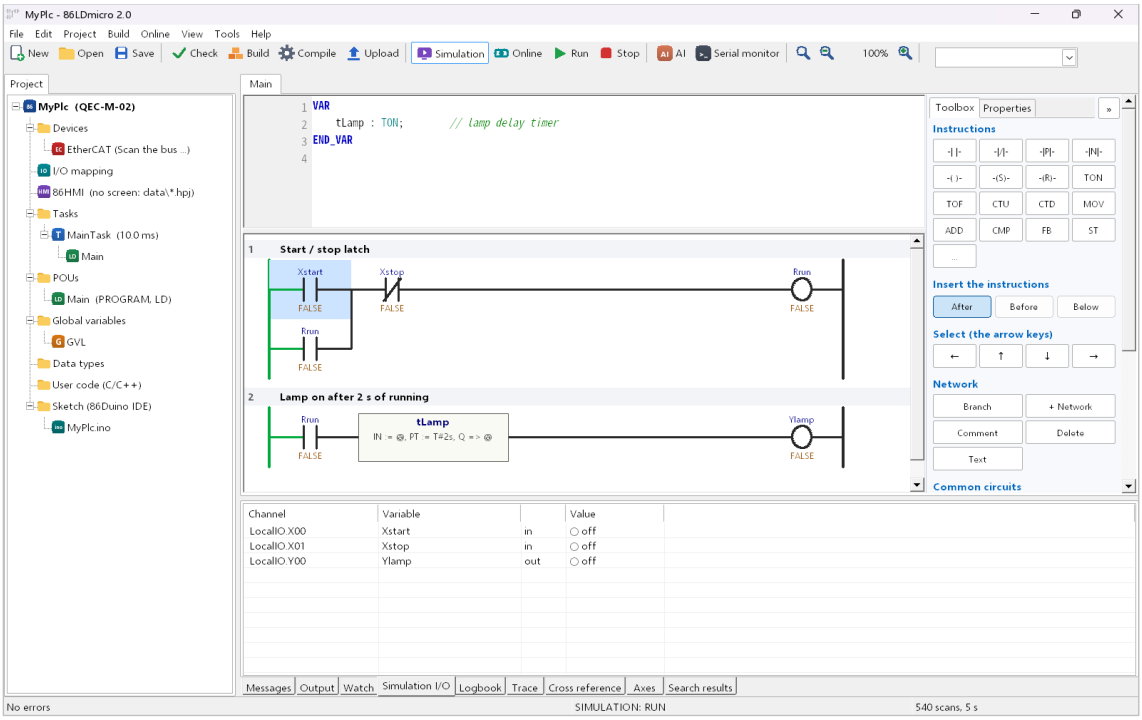


図 9-1　ツールバーの「Simulate」ボタン（押された状態）とステータスバーの「SIMULATION: RUN」

9.1.1　シミュレーション中の画面

エディター	シミュレーション中の表示	操作
ラダー図（LD）	緑色の線はパワーフローを示します。要素の下に変数値が表示され、タイマーブロックには経過時間が表示されます	接点またはコイルをダブルクリックすると、その変数の TRUE／FALSE が切り替わります
ST	変数にマウスポインターを合わせると、型と現在値が表示されます	変数を右クリック：Write、Force、Add to watch
SFC	アクティブなステップは緑色で、右側にアクティブになってからの時間が表示されます	—
マシンページ、軸ページ、Axes（試運転）	軸の位置、速度、状態	サーボオン、ジョグ、移動、原点復帰（12.5、12.6 節）
86HMI シミュレーターウィンドウ	スケッチに 86HMI 画面がある場合は自動的に開きます	PC 上で画面のウィジェットを直接クリックします（13.5 節）

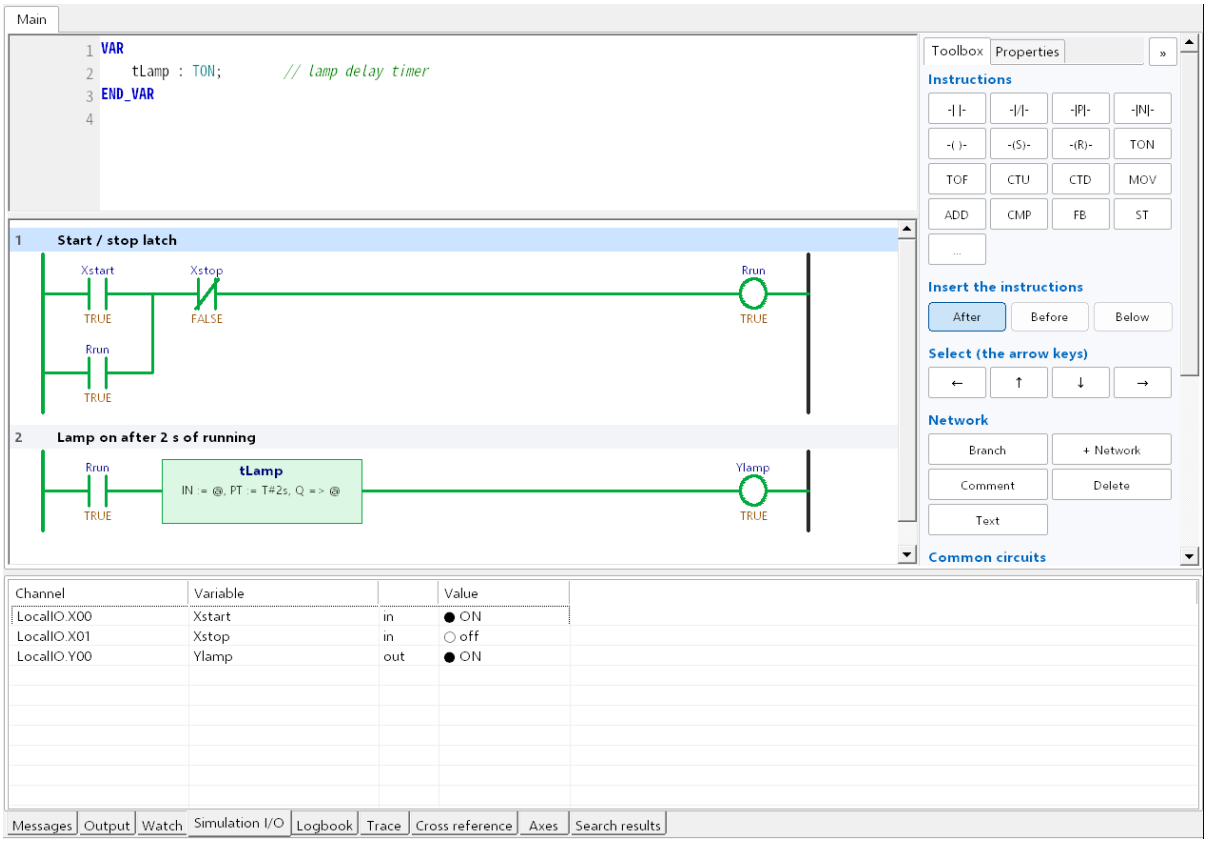


図 9-2　シミュレーション中のラダー図：緑色はパワーフロー、タイマーブロックには経過時間が表示される

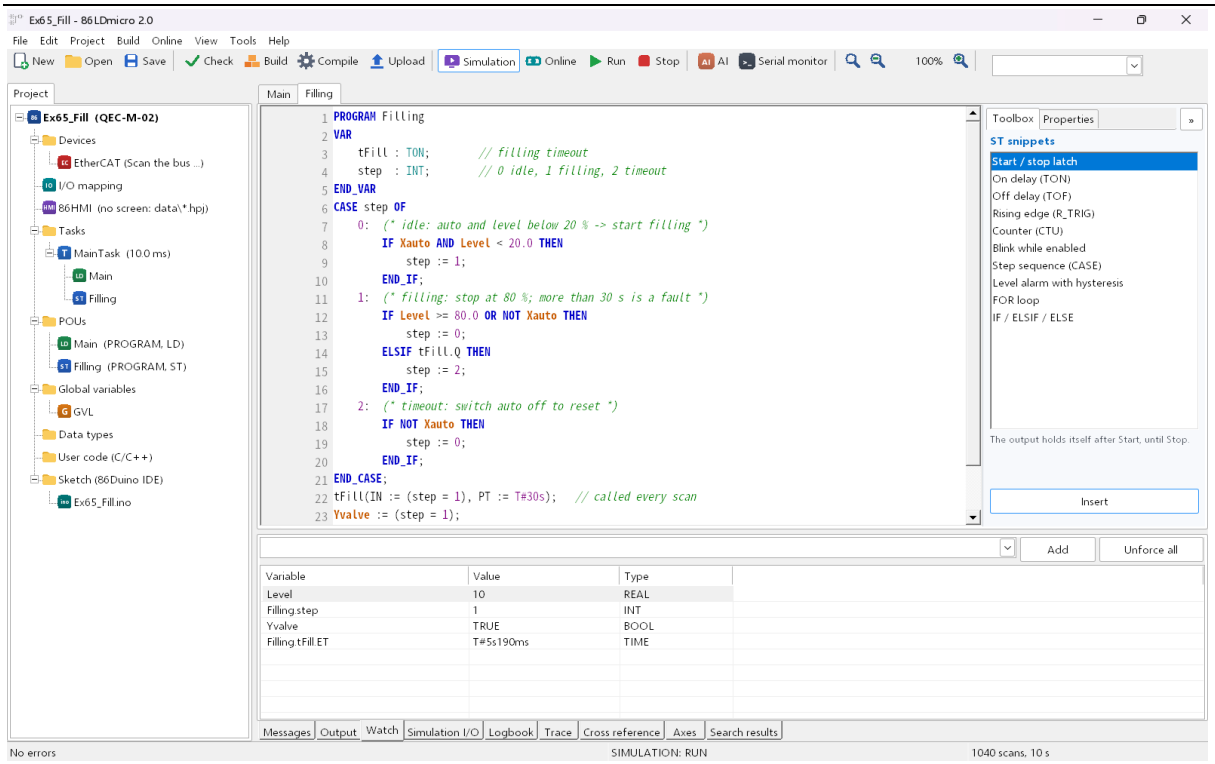


図 9-3　シミュレーション中の ST プログラムと Watch ウィンドウ

> ***注意：**シミュレーション中はプログラムを変更できません。変更するときは、まず F5 を押してシミュレーションを終了し、変更後にもう一度 F5 を押して再開してください（シミュレーションは初期値から始まります）。

9.1.2　シミュレーション速度と RUN／STOP

「Online」メニューにあるシミュレーション関連のコマンド：

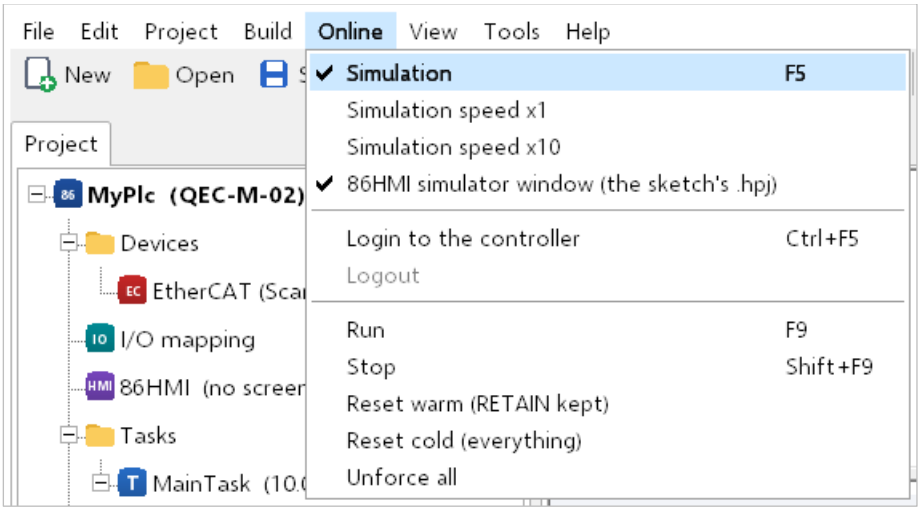


図 9-4　「Online」メニュー：Simulate、シミュレーション速度、Run／Stop、ウォームリスタート／コールドリスタート

項目	ショートカットキー	用途
Simulate	F5	シミュレーションの開始／終了
Simulation speed x1	—	実時間で実行（既定）
Simulation speed x10	—	時間が 10 倍速で進みます。タイマーも軸の移動も速くなるため、長時間かかる工程のテストに便利です
86HMI simulator window (the sketch's .hpj)	—	86HMI シミュレーターウィンドウを開く／閉じる（13.5 節）
Run	F9	プログラムの実行を続けます
Stop	Shift+F9	プログラムの実行を停止します（ステータスバーに「SIMULATION: STOP」と表示）
Reset warm (RETAIN kept)	—	変数は初期値に戻り、RETAIN 変数は保持されます
Reset cold (everything)	—	すべての変数（RETAIN を含む）が初期値に戻ります
Unforce all	—	すべての強制を解除します（9.3 節）

*ヒント：シミュレーション中にプログラムの実行エラー（配列のインデックスが範囲外など）が発生するとシミュレーションは停止（STOP）し、原因が「Logbook」に記録されます（例：`STOP: Main line 12: …`）。プログラムを修正してから、もう一度シミュレーションしてください。

9.2 Simulation I/O

下部ツールウィンドウの「**Simulation I/O**」タブには、各 I/O に割り当てられた物理点（4.6 節）が一覧表示されます。ローカル I/O と EtherCAT の PDO エントリーを含み、チャンネル、変数、方向、現在値が表示されます。

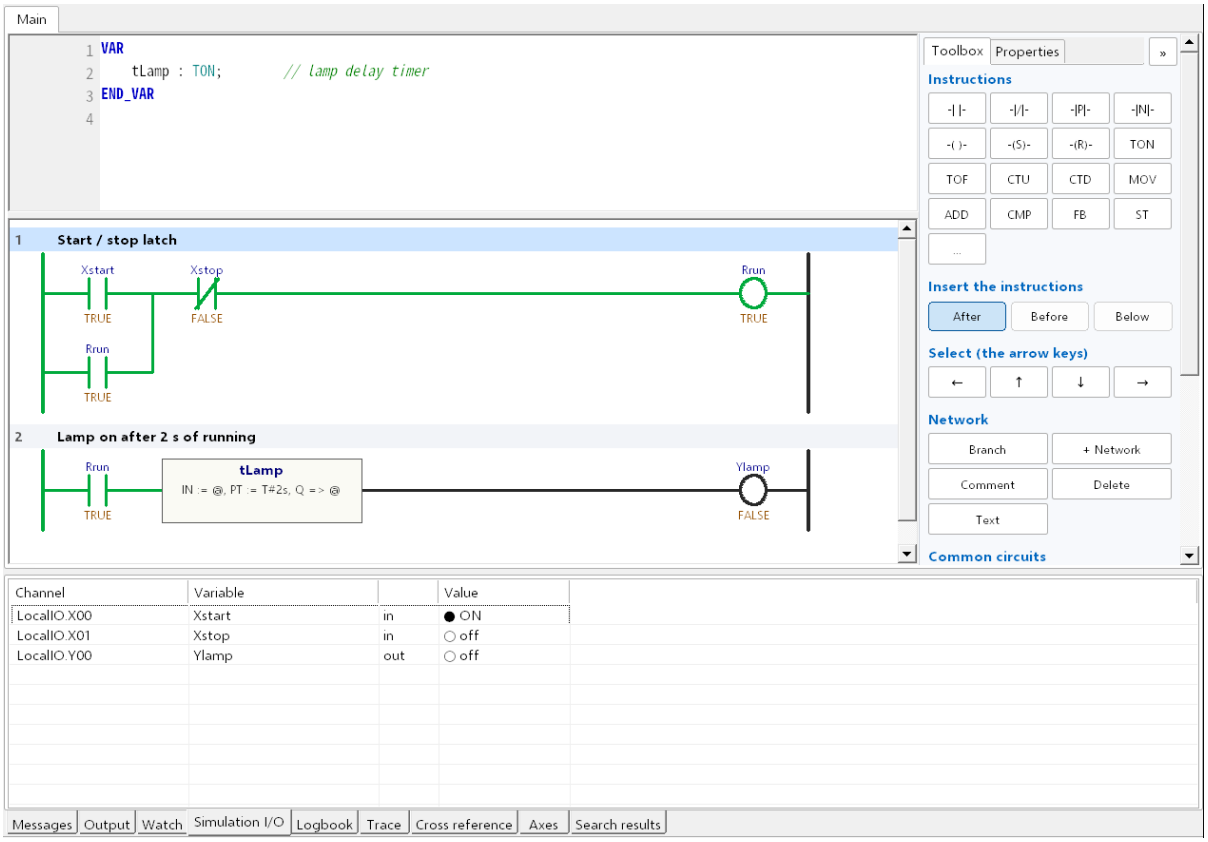


図 9-5　Simulation I/O：LocalIO.X00（Xstart）などの物理点

物理点	操作
デジタル入力	ダブルクリックで ON／OFF を切り替えます。実際のボタンを押したり離したりするのと同じです
アナログ入力	ダブルクリックして数値を入力します
出力（デジタル、アナログ）	表示のみ。値はプログラムが決めます

I/O 割り当ての反転、フィルター、スケーリングは、シミュレーションでもコントローラーと同じように処理されます。

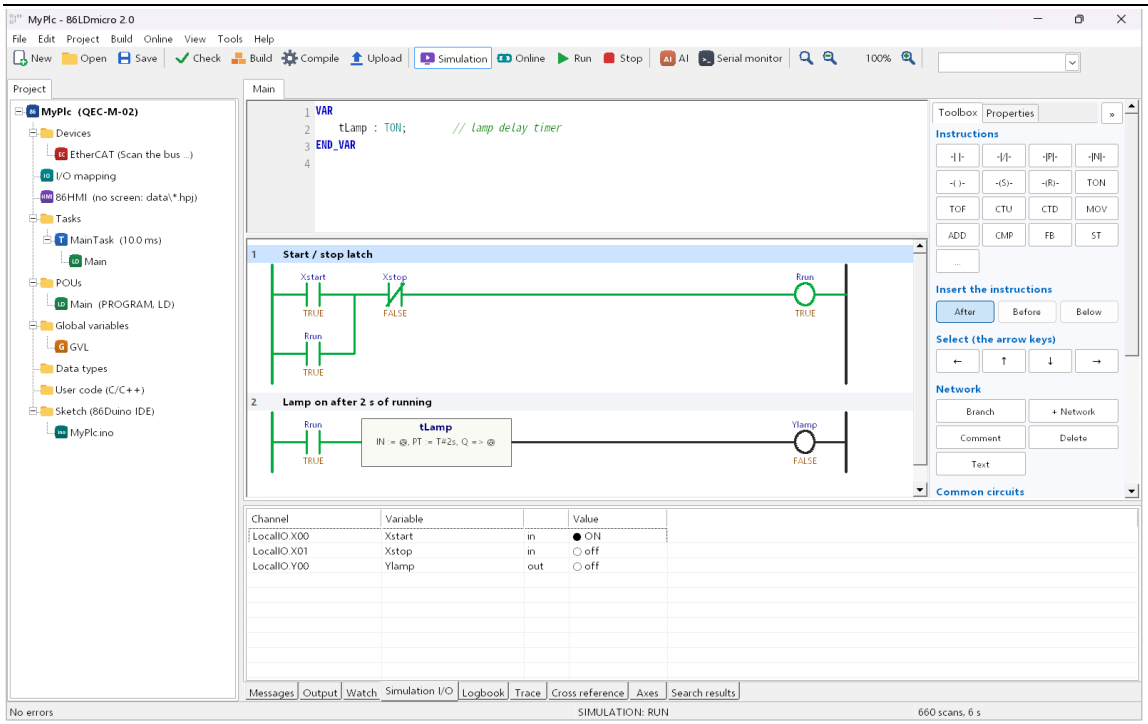


図 9-6　LocalIO.X00 をダブルクリックした後：Xstart が ON になり、Rrun が自己保持する

***注意：**物理入力に割り当てられた変数（例：`LocalIO.X00` に接続された `Xstart`）は、シミュレーション中は「Simulation I/O」によって値が決まります。Watch ウィンドウで直接書き込んでも、次のスキャンで入力値に上書きされます。「Simulation I/O」で切り替えるか、強制（9.3 節）を使用してください。

9.3　ウォッチ、書き込み、強制

「**Watch**」ウィンドウには、選択した変数とそのリアルタイムの値が一覧表示され、書き込みや強制もできます。値の表示と書き込みは、シミュレーション中またはオンライン中のみ可能です。

9.3.1　変数の追加

- Watch ウィンドウ上部の入力欄に変数名を入力し（ドロップダウンにすべての変数が表示されます）、**Add** をクリックします。プログラム.変数（例：`Wash.cycles`）、インスタンス.メンバー（例：`Panel.bl.Out`）、配列要素、SFC の プログラム.ステップ_X を入力できます。
- または、プログラム内の変数を右クリックして「Add <変数> to the watch window」を選択します。
- 行を選択して Del キーを押すか、右クリックして「Remove from the watch」を選択すると、その変数が削除されます。

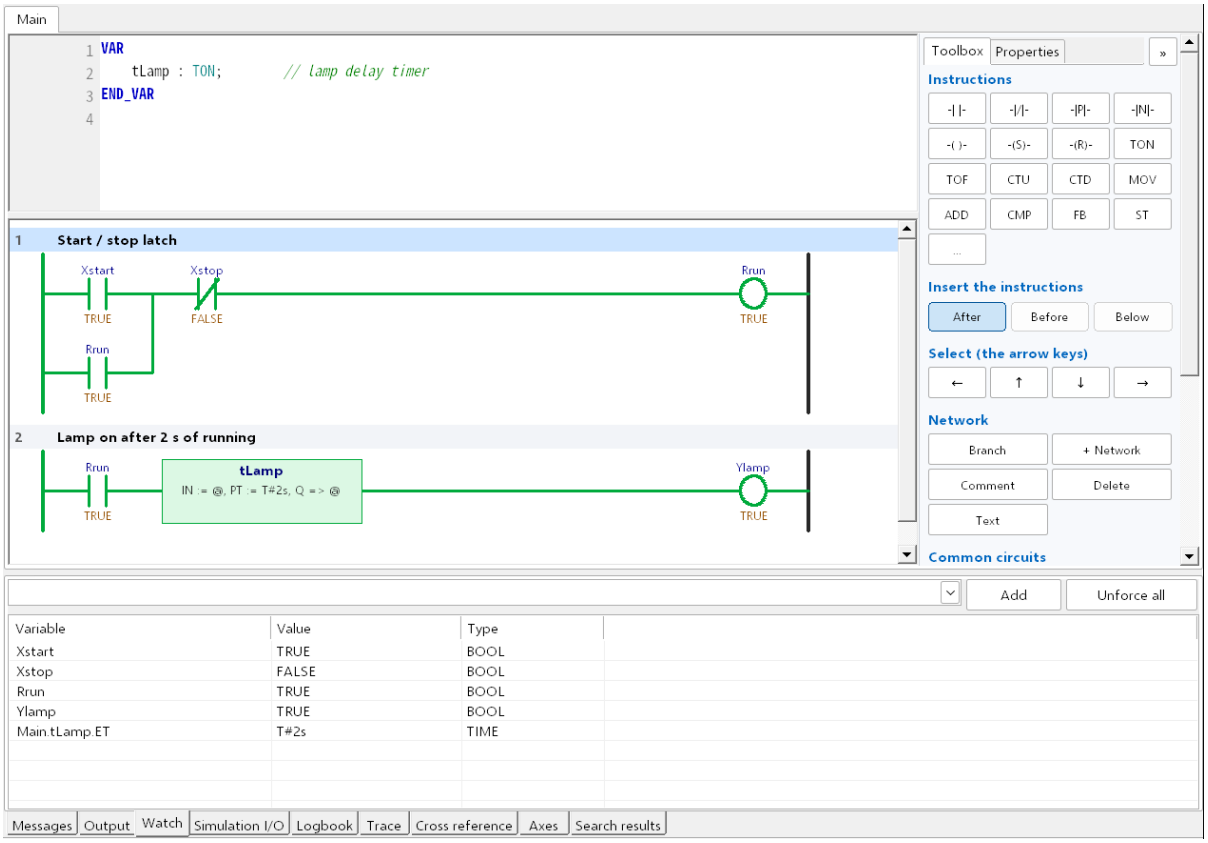


図 9-7　Watch ウィンドウ：変数を追加すると値がリアルタイムで表示される

9.3.2　書き込みと強制

機能	操作方法	効果
Write（書き込み）	Watch ウィンドウの行をダブルクリックします。BOOL は TRUE／FALSE が直接切り替わり、その他の型は数値を入力します。または右クリックして「Write <変数>...」を選択します	**1 回**だけ書き込みます。その後もプログラムが値を変更できます
Force（強制）	右クリックして「Force <変数>...」を選択し、数値を入力します	スキャンごとに同じ値を書き込みます。値の後ろに `[F]` が表示されます
Unforce（強制解除）	右クリックして「Unforce」を選択	この変数の強制を解除します
Unforce all（すべての強制を解除）	Watch ウィンドウ右上の **Unforce all** ボタン、または「Online > Unforce all」	すべての変数の強制を解除します

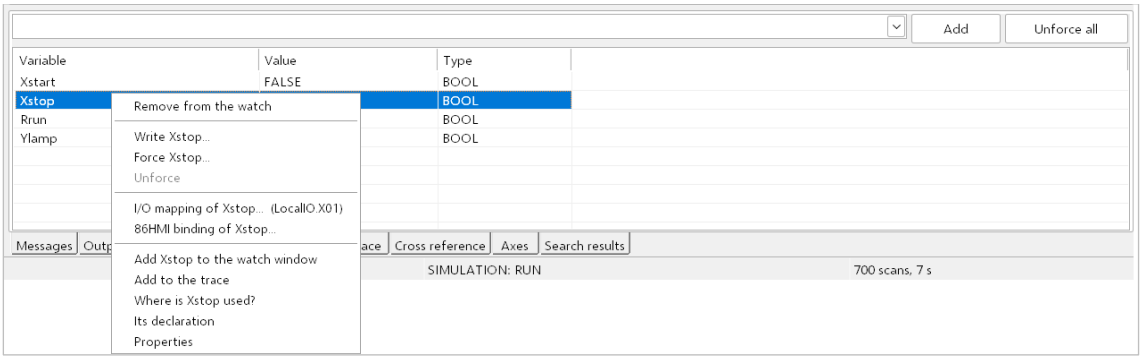


図 9-8　Watch ウィンドウの右クリックメニュー：Write、Force、Unforce、Add to the trace など

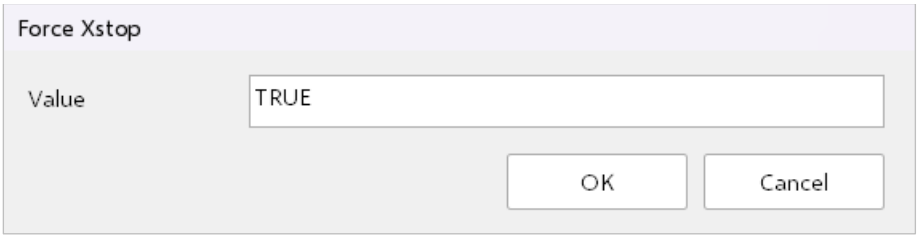


図 9-9　Xstop の強制：保持する値を入力する

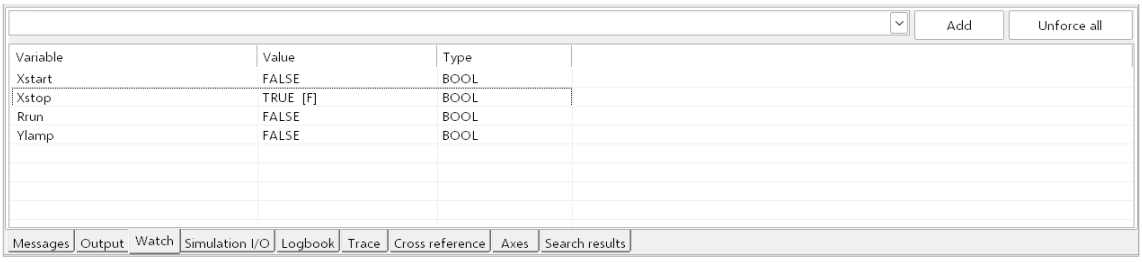


図 9-10　強制中の変数は値の後ろに [F] が表示される

9.3.3 強制値が書き込まれるタイミング

強制値は、CODESYS と同じく、スキャンごとに **2 回**書き込まれます。

1. 入力の読み込み後、プログラムの実行前：プログラムが読むのは強制値です。
2. プログラムの実行後、出力の書き出し前：物理出力に書き出されるのは強制値です。

あるプログラムがスキャン中にその変数へ自分で書き込むと、同じスキャン内でそのプログラムの**後に実行される**プログラムは、強制値ではなくプログラムが書き込んだ値を読みます。強制値で上書きし直されるのはスキャンの最後です。したがって、次のようになります。

- 強制**が**最も効果的なのは、**I/O 割り当てによって書き込まれる入力変数**（物理入力）と**出力変数です。**
- プログラムがスキャンごとに書き直す中間変数は、強制すると Watch ウィンドウには強制値が表示されますが、他のプログラムがその値を読めるとは限りません。
- 強制しても「効かない」「すぐに戻る」ときは、Cross reference（9.5 節）でその変数に書き込んでいるプログラムを調べてください。

*ヒント：例：`tools\example\ex30_dispense` のデモモード（`gDemo = TRUE`）では、最初のプログラム PlantSim が `sim*` を書き直し、さらに IoMap が `x*` を書き直すため、これらの変数を強制しても上書きされます（1 サイクルで元に戻ります）。強制を使って工程をテストするときは、まずデモモードをオフにし（`gDemo = FALSE`。これを強制してもかまいません）、それから物理入力 `diEntry`、`diDoorClosed` などを強制してください。

*安全：強制すると出力はプログラムの制御を受けなくなります。実機では、現場の安全を確認してから操作してください。コントローラーとの接続が 3 秒間途切れると、コントローラーはすべての強制を自動的に解除します。

9.4 Trace

「**Trace**」ウィンドウは変数の変化を曲線で描画します。タイミング、アナログ値、軸位置の観察に適しています。

1. Watch ウィンドウ（またはプログラム内）の変数を右クリックして「**Add to the trace**」を選択します。最大 8 個の変数を追加できます。
2. 「Trace」タブが自動的に前面に表示されます。100 ms ごとにサンプリングし、直近 1 分間を表示します。
3. Trace 上で右クリック：**Pause**、**Clear the samples**、**Remove all variables**、**Copy as CSV**（Excel に貼り付けて分析）。

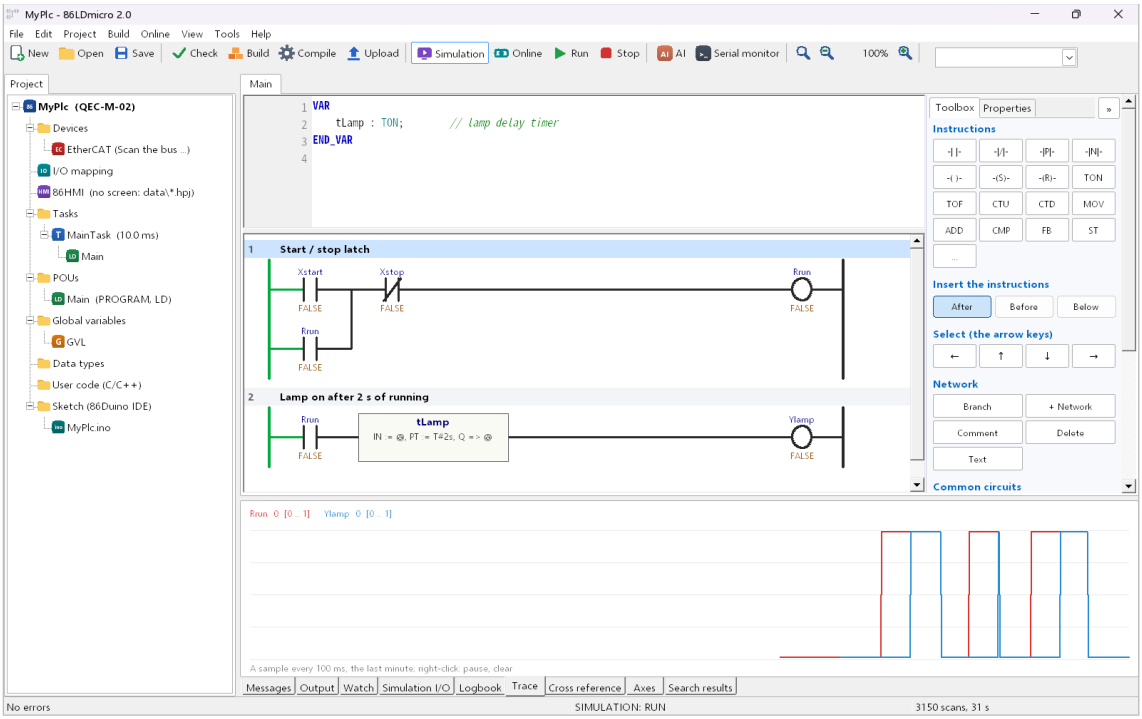


図 9-11　Trace：Rrun と Ylamp の変化

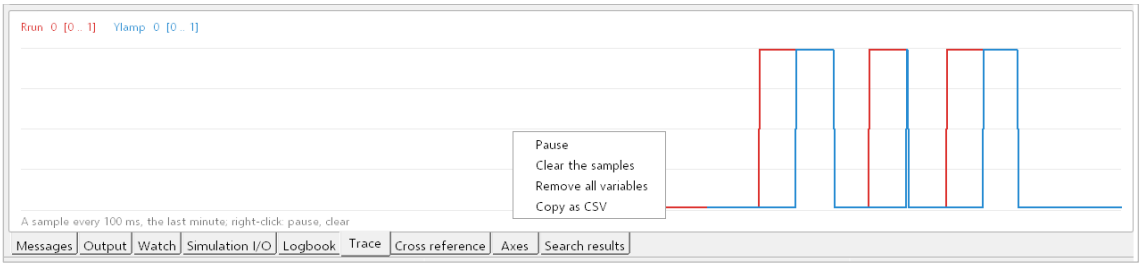


図 9-12　Trace の右クリックメニュー

9.5 Cross reference と検索

プロジェクトが大きくなると、「この変数はどこで使われているか」「この文字列はどのプログラムにあるか」を探す必要がよく出てきます。86LDmicro には 3 つの検索方法があります。

機能	ショートカットキー	検索対象	適した用途
Find in this tab	Ctrl+F（F3／Shift+F3 で次／前）	現在のタブの文字列またはラダー要素	1 つのプログラム内を順に検索する
Find a variable in the project（Cross reference）	Ctrl+Shift+F	変数のあらゆる場所での読み取り、書き込み、宣告	変数に書き込んでいる箇所を確認する
Find in the project	Ctrl+Shift+H	プロジェクト全体の任意の文字列（コメント、.ino を含む）	コメント、文字列、関数、名前がはっきりしないものを探す

9.5.1 Cross reference（Ctrl+Shift+F）

1. **Ctrl+Shift+F**（「Edit > Find a variable in the project...」）を押し、「Find a variable」ウィンドウに名前を入力して **OK** をクリックします。変数を右クリックして「Where is <変数> used?」を選択することもできます。
2. 下部の「**Cross reference**」タブに、すべての箇所が一覧表示されます。ファイル、行、用途（**Declarations**、**Read**、**Write**）とその行の内容が表示され、ステータスバーに合計件数が表示されます。
3. 行をダブルクリックすると、そのファイルが開き、該当行にジャンプします。

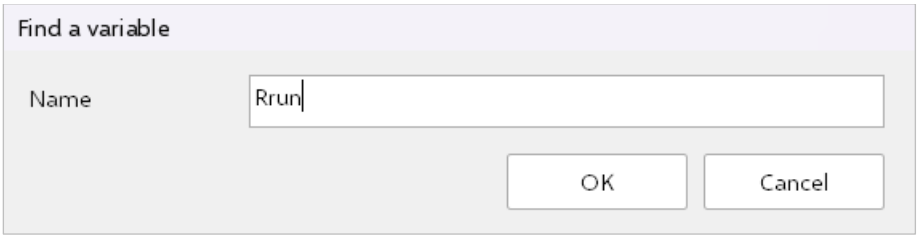


図 9-13 「Find a variable」ウィンドウ

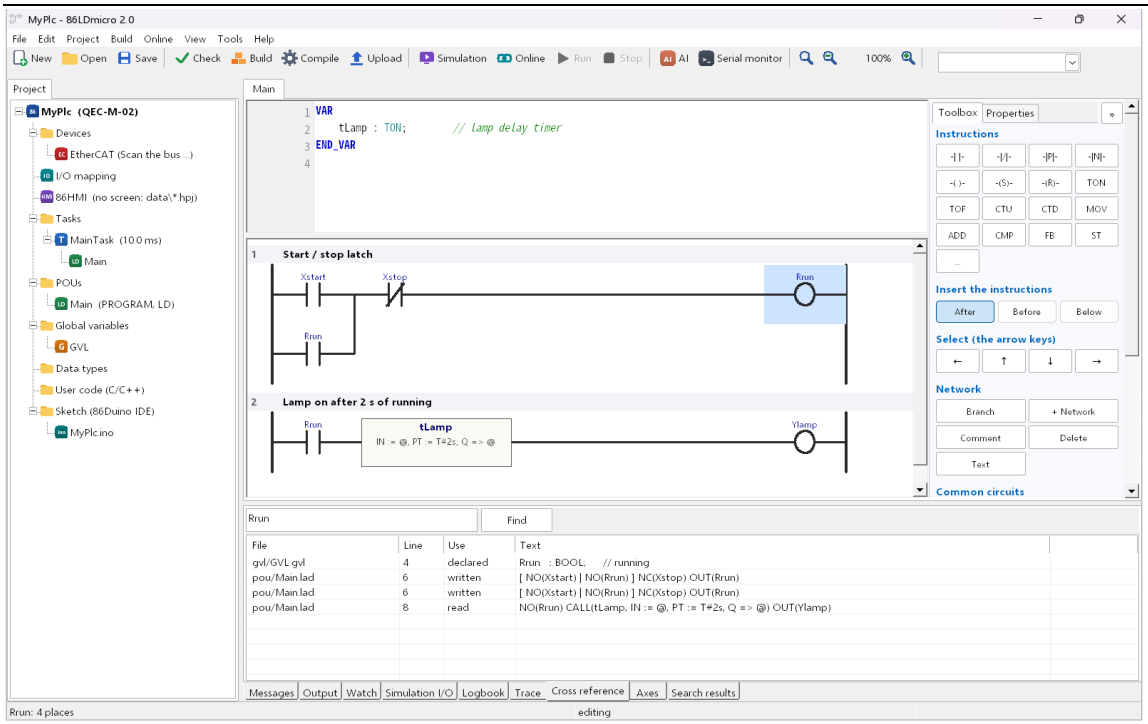


図 9-14　Cross reference：Rrun の宣言、読み取り、書き込みの位置

*ヒント：用途が「Write」の箇所とは、:= の左辺、またはラダー図の OUT、SET、RST、MOV などです。1 つの変数に 2 つ以上のプログラムが書き込む場合、最後に実行されたプログラムがスキャン終了時の値を決めます。

9.5.2　プロジェクト内の検索（Ctrl+Shift+H）

すべての POU（ラダー図のネットワークタイトル、コメント、要素、ST、SFC）、グローバル変数リスト、データ型、ユーザーコード（C/C++）、およびスケッチの .ino を検索します。

1. **Ctrl+Shift+H**（「Edit > Find in the project...」）を押します。下部ツールウィンドウが「**Search results**」に切り替わります。先にプログラム内の文字列を選択してから押すと、その文字列が直接検索されます。
2. 検索する文字列を入力し、**Enter** を押すか **Find** をクリックします。必要に応じて「**Match case**」または「**Whole word**」をオンにします（例：`Pump` を検索するときに `PumpSpeed` を表示しない）。
3. 結果は 1 行に 1 箇所ずつ、位置（POU 名と言語）、行番号、その行の内容が表示されます。右上に合計件数が表示されます。
4. 行を**ダブルクリック**する（または選択して Enter を押す）と、そのファイルが開き、該当行にジャンプします。ST と C/C++ ではその行が選択され、ラダー図ではそのネットワークまでスクロールして選択されます。

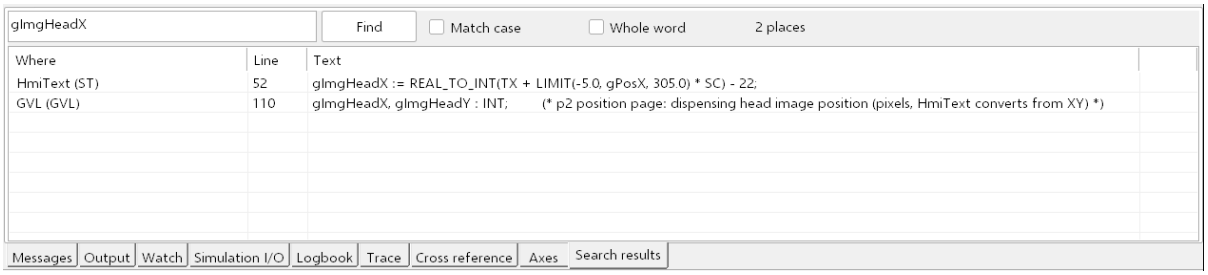


図 9-15　「Search results」：gImgHeadX を検索すると 2 箇所が見つかる（ST プログラムとグローバル変数リスト）

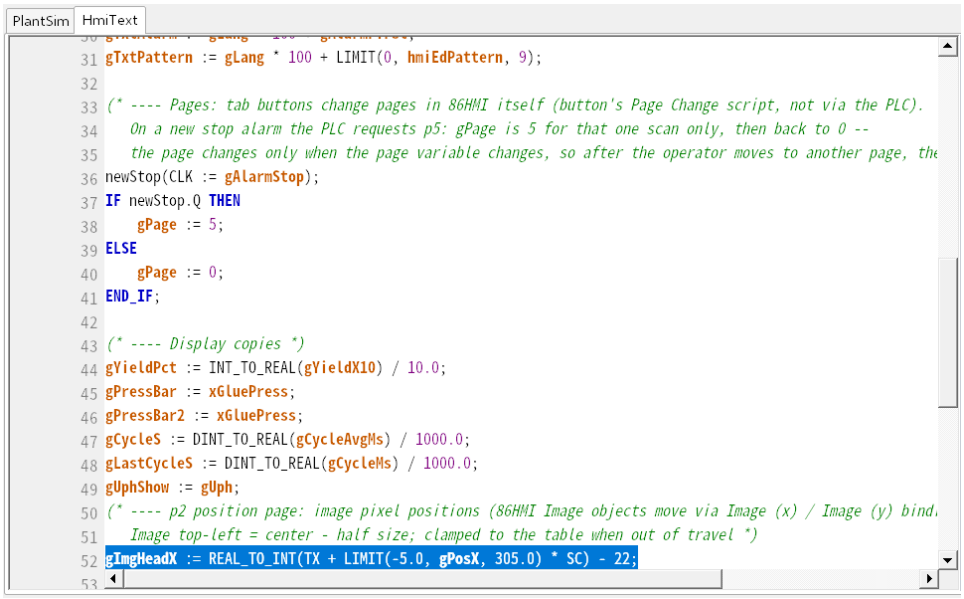


図 9-16　最初の行をダブルクリック：HmiText が開き、51 行目が選択される

> ***ヒント：**検索は探すだけで、置換はしません。変数名を変更するときは、結果を 1 つずつ開いて修正し、最後に **Check**（Shift+F7）をクリックして漏れがないことを確認してください。コマンドラインでも検索できます：`86ldmicro-cli search <プロジェクト> <文字列> [--case] [--word]`（15.5 節）。

9.6　よくあるエラーメッセージ

Check（Shift+F7）をクリックするかビルドすると、「**Messages**」ウィンドウにエラー（必ず修正が必要）と警告（確認を推奨）が一覧表示されます。エラーのある要素は、ラダー図では**赤い背景**で、ST では下線で示されます。

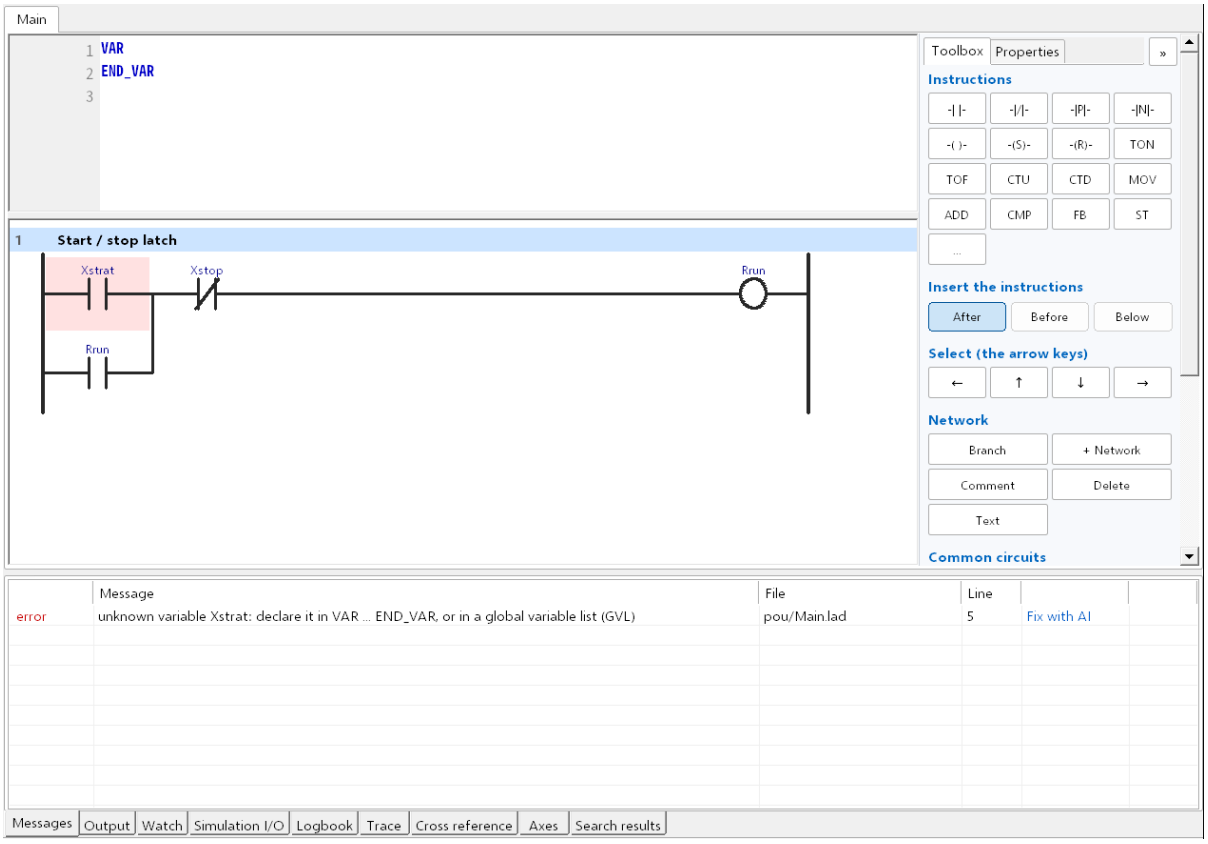


図 9-17　変数名の入力ミス：要素の背景が赤くなり、ビルド出力に「Build: the project has errors (Messages)」と表示される

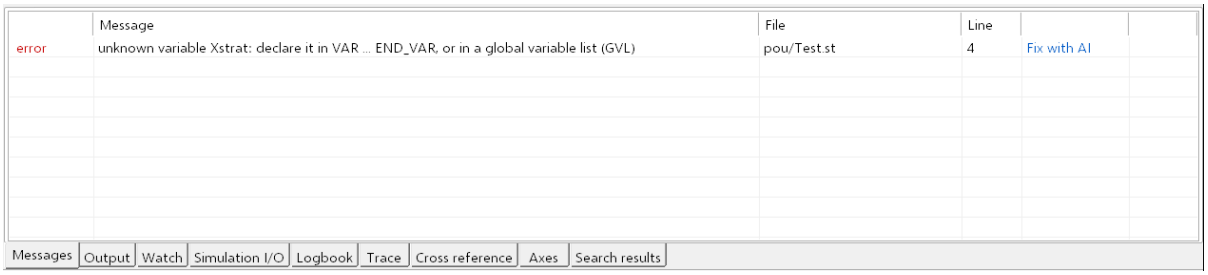


図 9-18　「Messages」ウィンドウ：1 行に 1 つのエラー。ダブルクリックで該当要素にジャンプ

1. 下部の「Messages」タブに切り替えます。
2. エラーの行をダブルクリックすると、エラーのある要素またはプログラム行にジャンプして選択されます。
3. 修正したら、エラーがなくなるまでもう一度 **Check**（Shift+F7）をクリックします。
4. 必要に応じて、その行の **Fix with AI** をクリックすると、エラーとプログラムが AI アシスタントに送られます（15.3 節）。

9.6.1 コンパイルメッセージ一覧

よくあるメッセージを次の表に示します（引用符内の名前は実際のプログラムによって異なります）。

メッセージ（抜粋）	原因	対処
unknown variable Xstrat: declare it in VAR ... END_VAR, or in a global variable list (GVL)	名前の入力ミス、または未宣言	スペルを修正するか、GVL／VAR で宣言します
REAL is not BOOL	数値を接点や条件として使っている	比較式を書く：`x <> 0`、`CMP(x > 10)`
expected a statement, but found '1' (END_IF missing)	ST の IF／FOR／CASE に終わりがない	END_IF／END_FOR／END_CASE を追加します
missing ';'	ST の文末にセミコロンがない	示された位置の後に **;** を追加します
expected ')', but found ';'	括弧の対応が取れていない	括弧を確認します
unknown function block instance tLamp	CALL で使っているインスタンスが宣言されていない	宣言する：`tLamp : TON;`
TON has no input PTT (inputs: IN, PT)	ファンクションブロックのパラメーター名が誤っている	メッセージに示されたパラメーター名を使います
Count is declared twice in the GVL	変数が重複して宣言されている	どちらか一方を削除します
the value 100000 does not fit INT	定数が型の範囲を超えている	DINT に変更するか、数値を修正します
unknown variable AxisQ	軸名が誤っている（MC ファンクションブロック）	マシンページにある軸名を使います（例：AxisX）
task MainTask runs Ghost, which is not a PROGRAM of the project	タスク内の POU の名前が変更された、または削除された	タスク設定で削除するか、選択し直します（4.3 節）
a step follows a step (SFC)	2 つのステップの間にトランジションがない	間にトランジションを挿入します
Warning: Xauto is only read here: it has no I/O mapping and no program writes it -- it is always FALSE	入力変数の割り当て忘れ	I/O 割り当てページで割り当てを追加する（4.6 節）か、意図したものであることを確認します
警告：`the program Test is in no task: it does not run`（英語のメッセージ）	新しく追加した PROGRAM がタスクに入っていない	タスク設定で追加します（4.3 節）

9.6.2 アップロード、接続、実行時のメッセージ

メッセージ／現象	対処
Could not open COMx (in use by another program, such as the 86Duino IDE's Serial Monitor?)	そのポートを使用しているプログラムを閉じてから、もう一度アップロードします（10.1 節）
The controller runs another program (or another version of this one)	もう一度アップロード（F8）してから接続します（10.4 節）
Online COMx: no answer	プロジェクト設定で「The online view」が有効になっていることを確認します。プログラムで USB の Serial を使用しないでください。もう一度アップロードします
シリアルモニター：`EtherCAT: begin failed`	EtherCAT ケーブルがメインデバイスの EtherCAT ポートに接続されていない、またはサブデバイスに電源が入っていない（11.7 節）
シリアルモニター：`EtherCAT: the ENI … cannot be used`	ENI がメインデバイスにアップロードされていない、または実際のネットワークと一致しない。もう一度 Scan し、ENI をエクスポートしてアップロードします（11.7 節）
シリアルモニター：`EtherCAT: no slave`	EtherCAT ケーブルが接続されていない、サブデバイスに電源が入っていない、または IN ではなく OUT に接続されている
Logbook：`MainTask watchdog: STOP, …`	スキャン時間がウォッチドッグの設定を超え、PLC が STOP になった（10.5 節）
Logbook：ドライブが Operation Enabled から外れた	ドライブのアラームコードを確認します。原因を取り除いた後、MC_Reset、続いて MC_Power を実行します（12.9 節）

その他の問題と対処方法については、第 16 章を参照してください。

第 10 章　ビルド、アップロード、オンラインモニター

シミュレーションでプログラムに問題がないことを確認したら、ビルド、コンパイルして QEC コントローラーにアップロードし、オンラインモニターでコントローラー上のリアルタイムの状態を確認できます。この章では、これらのコマンド、ビルドで生成されるファイル、シリアルモニター、オンラインモニター、Logbook とウォッチドッグ、および 86HMI のリソースだけをアップロードする方法を順に説明します。

10.1　チェック、ビルド、コンパイル、アップロード

4 つのコマンドは「Build」メニューとツールバーにあり、左から右へ 1 つずつ処理が増えます。アップロードは先にビルドとコンパイルを行い、コンパイルは先にビルドを行います。

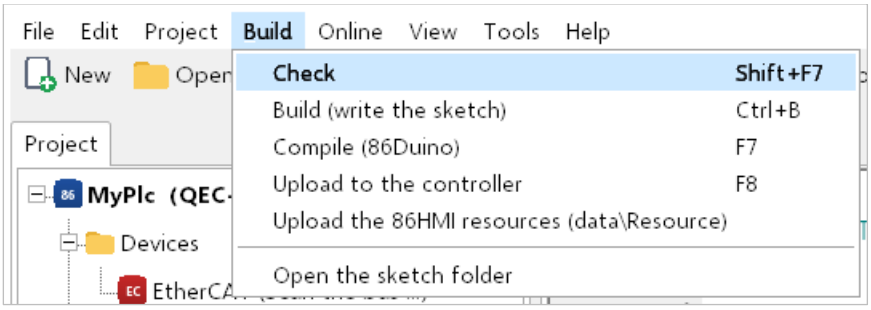


図 10-1　「Build」メニュー

項目	ショートカットキー	用途	必要なもの
Check	Shift+F7	PLC プログラムだけをコンパイルし、エラーと警告を「Messages」に表示します。ファイルは書き込みません	—
Build (write the sketch)	Ctrl+B	プロジェクトをスケッチの `plc86.cpp` などのファイルとして書き出します（10.2 節）。IDE から起動した場合は、IDE に再読み込みを通知します	—
Compile (86Duino)	F7	ビルド後、86Duino IDE の arduino-builder でコンパイルします。C/C++ のエラーは .cpp86 の行番号を指します	86Duino IDE（2.6 節）
Upload to the controller	F8	コンパイル後、v86dude で USB 経由でアップロードします。スケッチに `data\Resource` がある場合は一緒にアップロードします	IDE、USB ケーブル、ツールバーで選択したシリアルポート
Upload the 86HMI resources (data\Resource)	—	86HMI の画像、フォント、画面だけをアップロードします。コンパイルもプログラムの変更も行いません（10.6 節）	IDE、USB ケーブル、シリアルポート
Open the sketch folder	—	エクスプローラーでスケッチフォルダーを開きます	—



図 10-2　ツールバー：Check、Build、Compile、Upload とシリアルポートの選択

処理の経過は下部の「Output」ウィンドウの「**Build output**」に表示され、エラーは「**Messages**」に表示されます（9.6 節）。

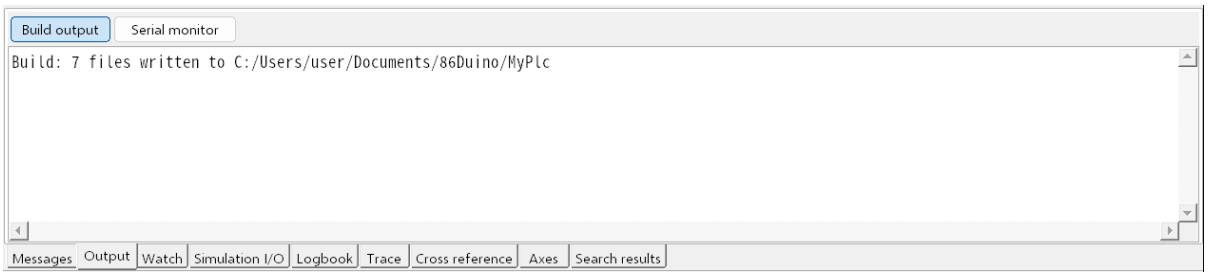


図 10-3　「Build output」：ビルド、コンパイル、アップロードの経過はすべてここに表示される

10.1.1　アップロードの手順

1. USB ケーブルで PC と QEC コントローラーを接続し、コントローラーに電源を入れます。
2. ツールバー右側のシリアルポート一覧で、コントローラーの COM ポートを選択します（例：COM4。見つからないときは「Tools > Refresh the serial ports」を選択します。2.4 節）。
3. **Upload**（F8）をクリックします。「Build output」に「Build: … files written …」、「== Compiled…」、「== Uploading to COM4 ...」と進行状況が順に表示され、最後に「== Uploaded」と表示されます。
4. コントローラーが再起動し、プログラムの実行を開始します。「Tools > Options」で「Online again after an upload」がオンになっていて、アップロード前にオンラインだった場合は、自動的に再接続します（2.6 節）。

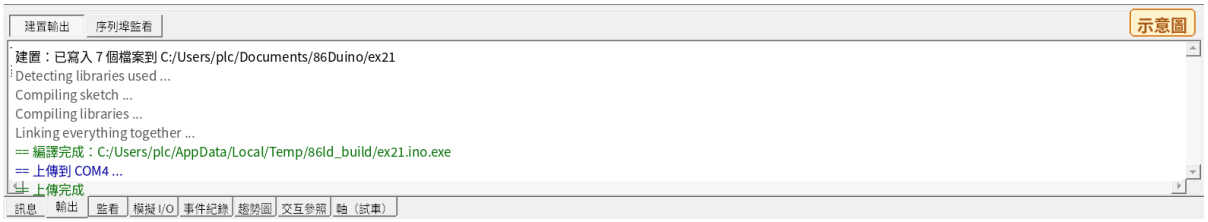


図 10-4　アップロード成功：Build → Compiled → Uploading to COM4 → Uploaded（概略図）

10.1.2　アップロードに失敗したとき

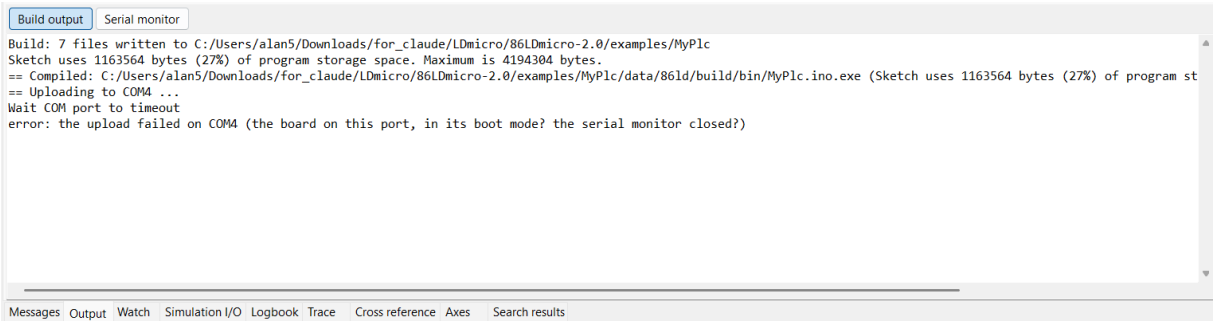


図 10-5　アップロード失敗：COM4 を開けない（シリアルポートが他のプログラムに使用されている）

出力のメッセージ	原因	対処
Could not open COMx (in use by another program…)	86Duino IDE のシリアルモニターやターミナルプログラムが同じポートを開いている	そのプログラムを閉じてから、もう一度 F8 を押します
COMx not found	USB ケーブルが抜けている、ポートの選択が誤っている、またはコントローラーに電源が入っていない	COM ポートを確認し（2.4 節）、選択し直します
The 86Duino IDE is not found	86LDmicro を単独で起動し、IDE フォルダーが設定されていない	「Tools > Options」で 86Duino IDE フォルダーを指定します（2.6 節）
コンパイルエラー（.cpp のファイル名と行番号）	ユーザーコード（C/C++）またはライブラリのエラー	メッセージをダブルクリックして .cpp86 の該当行にジャンプし、修正します（第 14 章）
Build: the project has errors	PLC プログラムにエラーがある（背景が赤い要素）	「Messages」タブを確認します（9.6 節）
アップロードの途中でタイムアウトする	USB ケーブルやハブが不安定、またはコントローラーが再起動中	PC の USB ポートに直接接続し、もう一度アップロードします

10.1.3　1 本の USB ケーブルの 4 つの用途

QEC の USB ケーブルは、PC 上では 1 つのシリアルポート（ツールバーで選択した COM ポート）です。このポートは次の 4 つの用途に使われますが、**同時に使えるのは 1 つだけ**です。86LDmicro が自動的にポートを明け渡し、再接続します。

用途	使用するもの	説明
アップロード	v86dude（F8）	プログラムをコントローラーに書き込みます。アップロード中は 86LDmicro が自動的にシリアルポートを明け渡し（オンライン中は先にオフラインにします）、完了後に自動的に再接続します
オンラインモニター	86LDmicro（Ctrl+F5）	プロジェクト設定で「The online view」が有効で、シリアルポートが `Serial` の場合、ウォッチ、強制、パワーフローはすべてこのケーブルを通ります
Serial.print	ユーザープログラム	プログラムが `Serial.print()` で出力した文字列。オンライン中はウォッチのパケットの間に挟まれて送られ、「Serial monitor」に自動的に表示されます
Configurator（USB）	EtherCAT ページ	USB モードの EtherCAT Configurator もこのケーブルを使います（3 Mbps）。アップロード中やオンライン中は自動的に一時停止します（第 11 章）

*__注意：__ユーザーコード（C/C++）で外部機器と通信するときは `Serial1`〜`Serial3`（コントローラーの COM1〜3）を使用してください。USB の `Serial` はアップロードとオンラインモニター用に確保されています。`plc86_user_ethercat()` の中では出力（print）しないでください。
プログラムが `Serial.print` で USB に出力しているのに PC がそのポートを読んでいない（86LDmicro が未接続、シリアルモニターが開いていない）場合、バッファーがいっぱいになると `Serial.print` が待ち続け、PLC のスキャン全体が止まってしまいます。本番運転用のプログラムでは USB に出力しないか、先に `Serial.availableForWrite()` を確認してください。

10.2 生成されるファイル

ビルドはプロジェクト全体（LD、ST、SFC、GVL、I/O 割り当て、デバイス、86HMI のバインド）を C++ に変換してスケッチフォルダーに書き込み、86Duino IDE のコンパイラーでコンパイルします。これらのファイルは**ビルドのたびに再生成されるので、手作業で編集しないでください。**

ファイル	生成元	内容
`plc86.cpp`、`plc86.h`	ビルド（Ctrl+B）	PLC プログラム本体：すべての POU、グローバル変数、I/O 処理、タスクスケジューリング。`plc86_setup()`／`plc86_loop()`
`plc86_rt.cpp`、`plc86_rt.h`	ビルド	ランタイム：オンラインモニターのプロトコル、強制、RUN／STOP、リスタート、RETAIN、Logbook、トレースのサンプリング、ウォッチドッグ
`plc86_ecat.cpp`、`plc86_ecat.h`	ビルド	EtherCAT メインデバイス（ENI、サブデバイス、PDO 割り当て、Motion86 マシン）。EtherCAT デバイスがない場合は空のファイル
`<sketch>.ino`	新規プロジェクト作成時に作成	次を呼び出します：`plc86_setup()`、`plc86_loop()`。独自のコードを追加でき、ビルドでは PLC に必要な部分（例：`#include "plc86.h"`、`Hmi.begin();`）だけが補われます
`data\86ld\build\plc86.symbols.json`	ビルド	シンボルテーブル：オンラインモニターがコントローラー上のプログラムとの照合や変数の検索に使用します
`data\86ld\build\bin\`	コンパイル（F7）	コンパイルの中間ファイルと実行ファイル（`<sketch>.ino.exe`）
`myhmi.cpp`、`myhmi.h`、`data\Resource\`	86HMI Editor の「Code Generation」	86HMI の画面とリソース（第 13 章）。ビルドでは生成されません

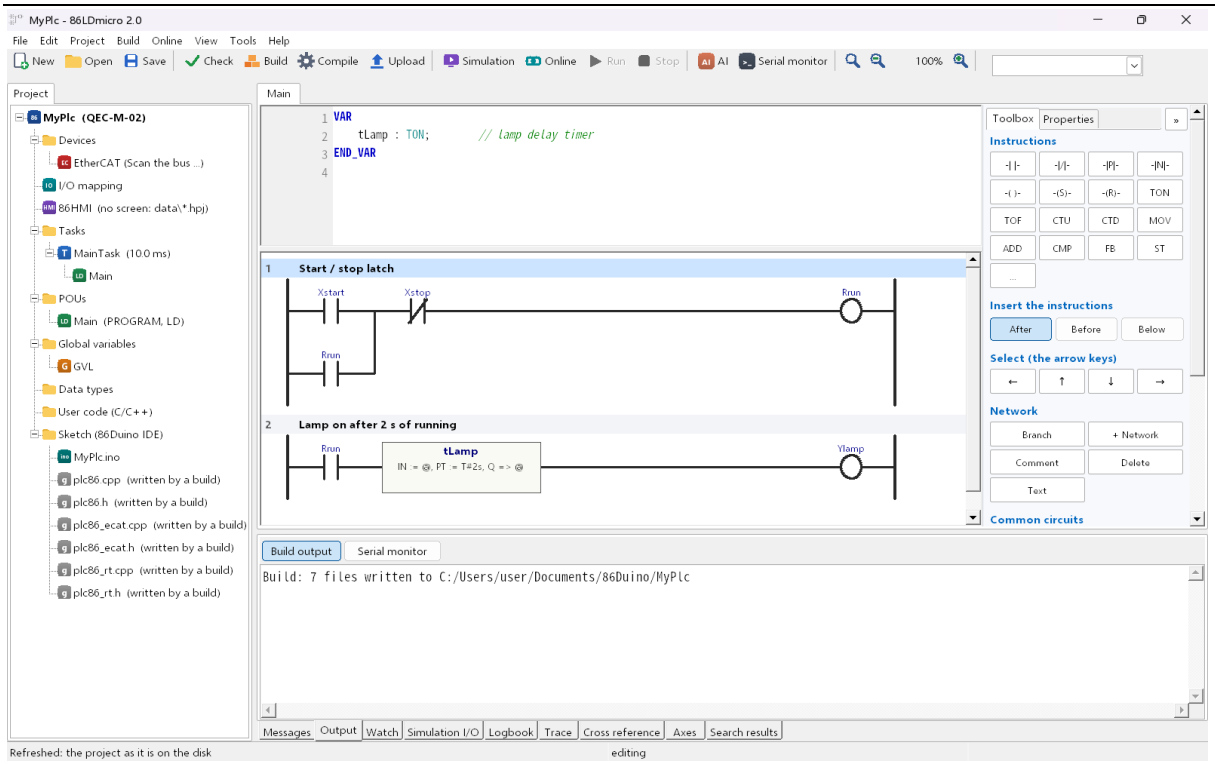


図 10-6　プロジェクトツリーの「Sketch (86Duino IDE)」：ビルドで生成されたファイルには「(written by a build)」と表示され、ビルド出力には「7 files written」と表示される

プロジェクトツリーの「Sketch (86Duino IDE)」の下から、.ino とこれらのファイルを直接開いて確認できます。86Duino IDE でもタブとして表示されます。

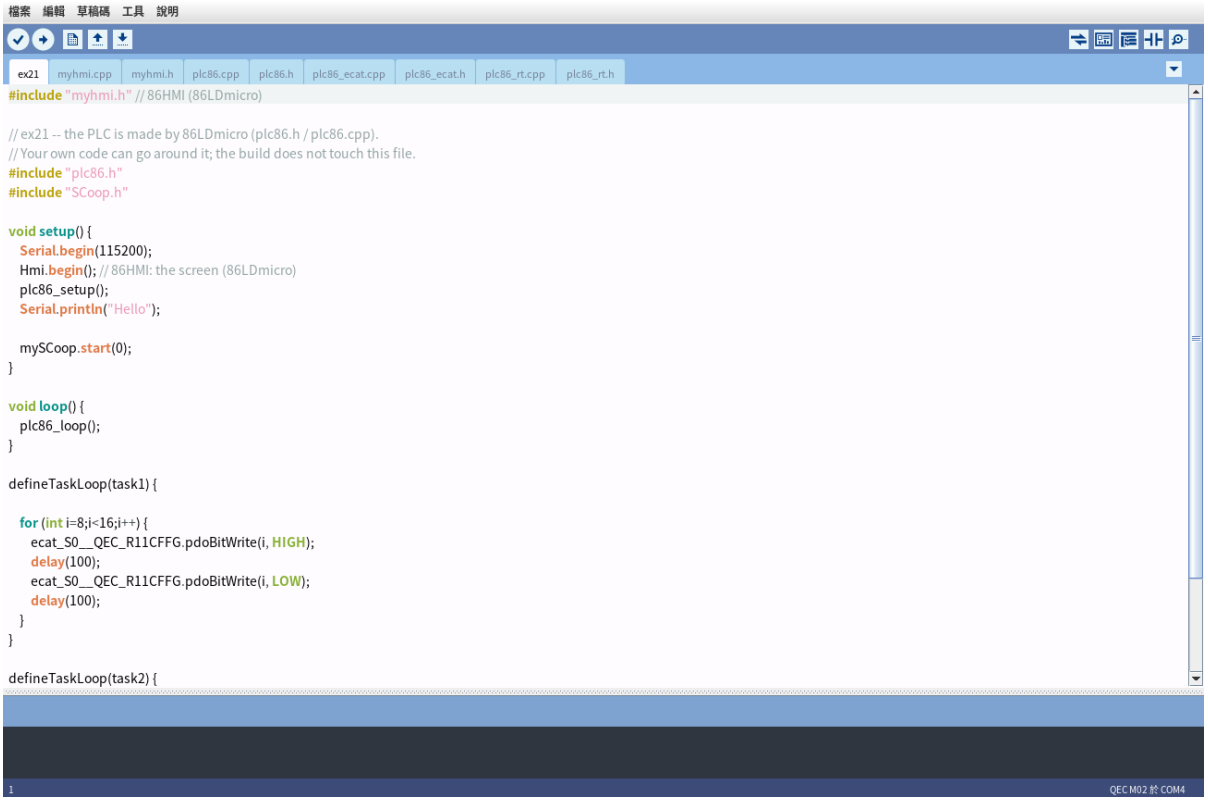


図 10-7　ビルド後の 86Duino IDE：plc86.cpp、plc86.h、plc86_ecat.cpp などのタブ

> ***ヒント：**`plc86*` ファイルと `data\86ld\build` はバージョン管理（git など）の対象から外してかまいません。必要なときに再ビルドすれば生成されます。

10.3　シリアルモニター

「Serial monitor」には、プログラムの `Serial.print()` の出力や EtherCAT メインデバイスの状態など、コントローラーがシリアルポートに出力した文字列が表示されます。ツールバーの **Serial monitor** をクリックするか、下部の「Output」ウィンドウ上部で「Serial monitor」に切り替えます。

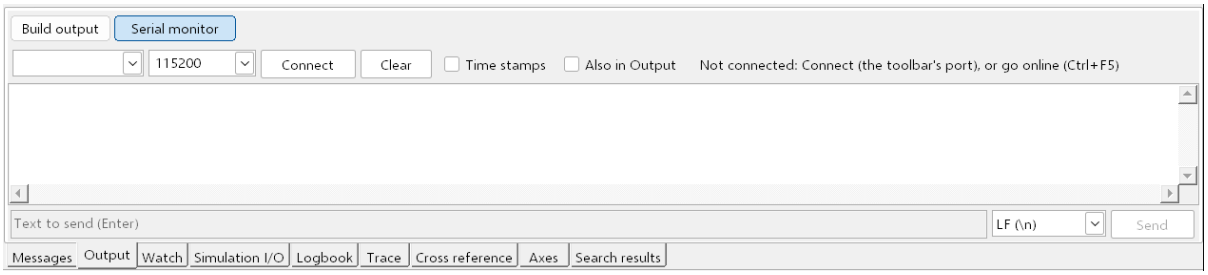


図 10-8　シリアルモニター：ポートの選択、ボーレート、接続、タイムスタンプ、送信

状況	表示方法
オンライン中（Ctrl+F5）	プログラムの `Serial.print()` の内容がオンライン接続を通じて自動的に表示されます。別途接続する必要はありません。オンライン中は送信できません
オフライン時	シリアルポート（空欄＝ツールバーのポート。USB-RS-232／485 変換器を選択して Serial1～3 を見ることもできます）とボーレートを選択し、**Connect** をクリックします。終了するときは **Disconnect** をクリックします
アップロード中	自動的にシリアルポートを明け渡し、アップロードが完了してコントローラーが再起動した後に自動的に再接続します

- **Time stamps**：各行の先頭に受信時刻を付けます。
- **Also in Output**：各行が「Build output」にも表示されます。
- 下部の入力欄に文字列を入力し、Enter を押すか **Send** をクリックします（改行文字を選択できます）。**Clear** で表示をクリアします。

EtherCAT デバイスがある場合は、メインデバイスの状態もここに出力されるため、オンラインにしなくても確認できます。例：

```
EtherCAT: running, 2 slaves, 1000 us, sync, ENI C:/eni/QEC-ENI-1.xml
```

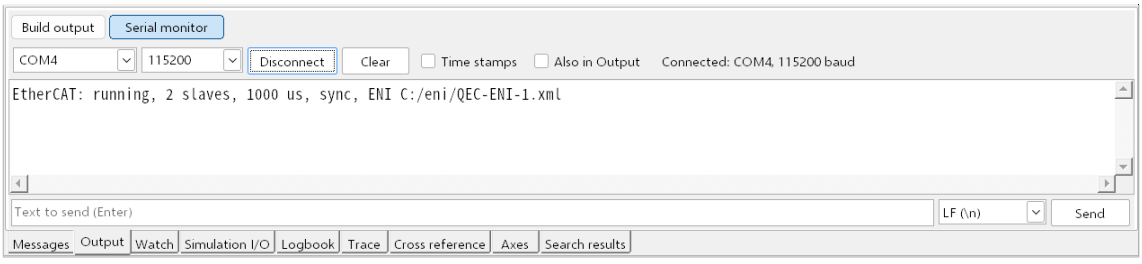


図 10-9　シリアルモニターに表示された EtherCAT メインデバイスの状態

10.4　オンラインモニター

オンラインモニターでは、86LDmicro をコントローラーに接続し、コントローラーで実行中のプログラムを PC 上で確認できます。パワーフロー、変数値、SFC のステップが表示され、書き込み、強制、RUN／STOP も行えます。

10.4.1　準備

「Project > Settings」の「**The online view (monitor, force, power flow)**」は既定でオンになっており、使用するシリアルポートは `Serial`（USB、アップロードと同じケーブル）、ボーレートの既定値は 115200 です。これらの設定を変更したら、再ビルドしてアップロードしてください（4.2 節）。

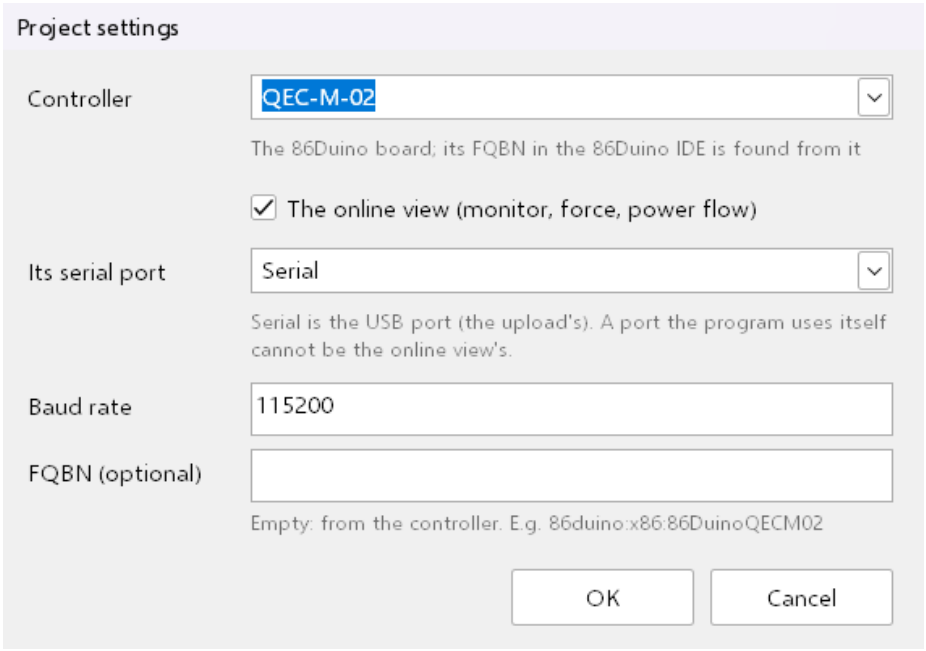


図 10-10　プロジェクト設定：オンラインモニター、使用するシリアルポート、ボーレート

10.4.2　接続

1. プログラムをアップロードします（F8）。
2. ツールバーの **Online** をクリックするか、「Online > Login to the controller」（Ctrl+F5）を選択します。
3. 接続すると、ステータスバーに「ONLINE COM4: RUN」と、スキャン時間、最大値、スキャン回数、タイムアウト回数が表示されます。ラダー図と SFC にはリアルタイムのパワーフローと値が表示されます。
4. 終了するときは「Online > Logout」を選択するか、ツールバーの **Online** をもう一度クリックします。

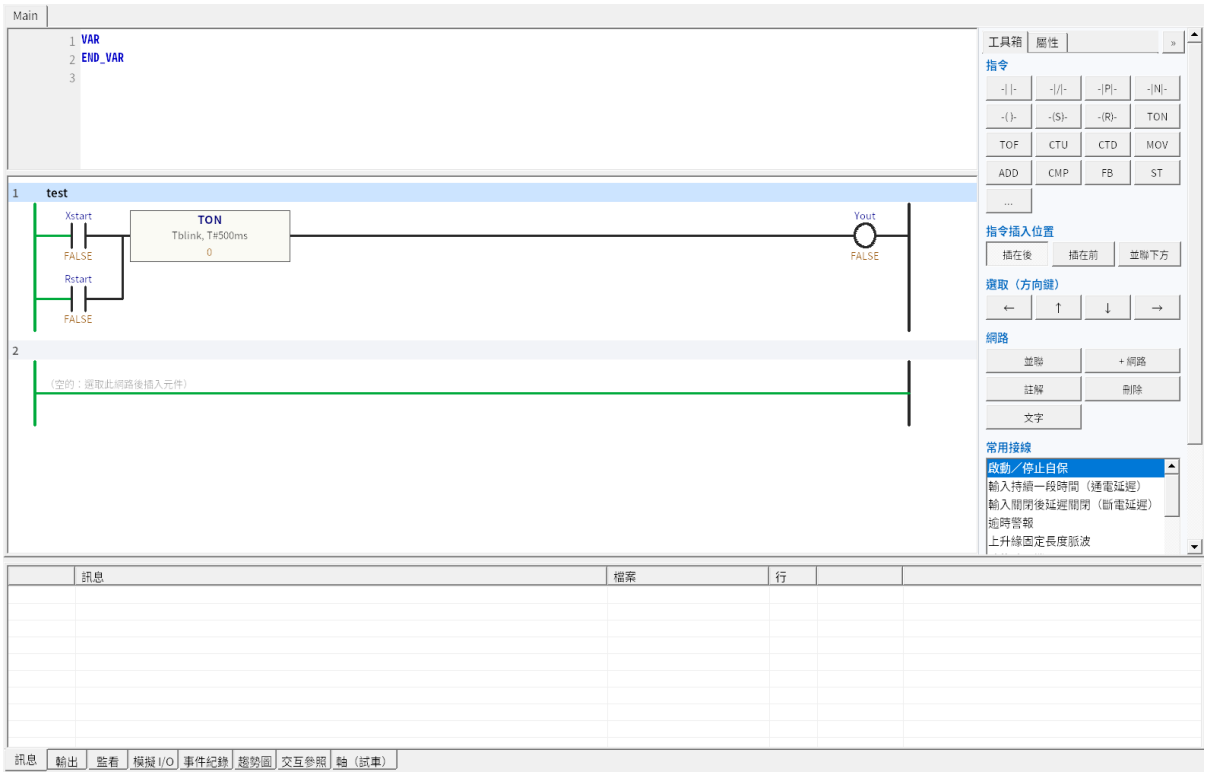


図 10-11　オンラインモニター：ラダー図にコントローラー上のリアルタイムのパワーフローが表示される



図 10-12　オンライン中のステータスバー：ONLINE COM4: RUN とスキャン時間

コントローラーで実行中のプログラムが今回ビルドしたものでない場合、86LDmicro は接続を拒否し、「The controller runs another program (or another version of this one)… Generate the code, upload it, then go online」と表示します。もう一度アップロード（F8）してください。



図 10-13　コントローラー上のプログラムが今回のビルドと異なるときのメッセージ

10.4.3　オンライン中の操作

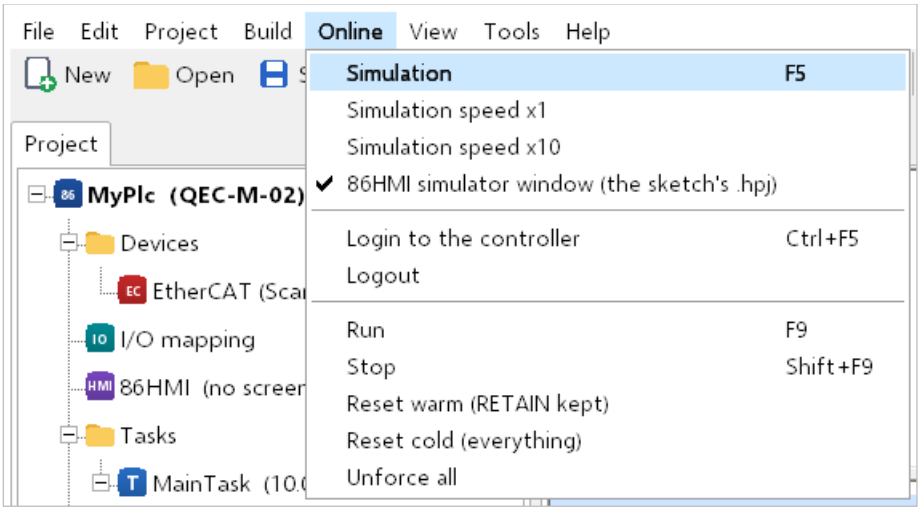


図 10-14　オンライン中の「Online」メニュー

項目	ショートカットキー	用途
Login to the controller	Ctrl+F5	オンラインモニターを開始します
Logout	—	オンラインモニターを終了します（コントローラーは実行を続けます）
Run	F9	プログラムの実行を続けます
Stop	Shift+F9	プログラムの実行を停止します。I/O と EtherCAT は通常どおり更新され、出力は最後の値を保持します。すべての Motion86 マシンが停止し、ジョグも終了します
Reset warm (RETAIN kept)	—	変数は初期値に戻り、RETAIN 変数は保持され、すべての強制が解除されます。RUN／STOP の状態は変わりません（RUN 中はプログラムが初期値から再開します）
Reset cold (everything)	—	すべての変数（RETAIN を含む）が初期値に戻ります
Unforce all	—	すべての強制を解除します

*安全：STOP はプログラムを実行しないだけで、**安全停止ではありません。**出力は最後の値を保持します。装置の非常停止はハードウェアの安全回路で実現してください。

ウォッチ、書き込み、強制、Trace の操作はシミュレーションと同じです（9.3、9.4 節）。

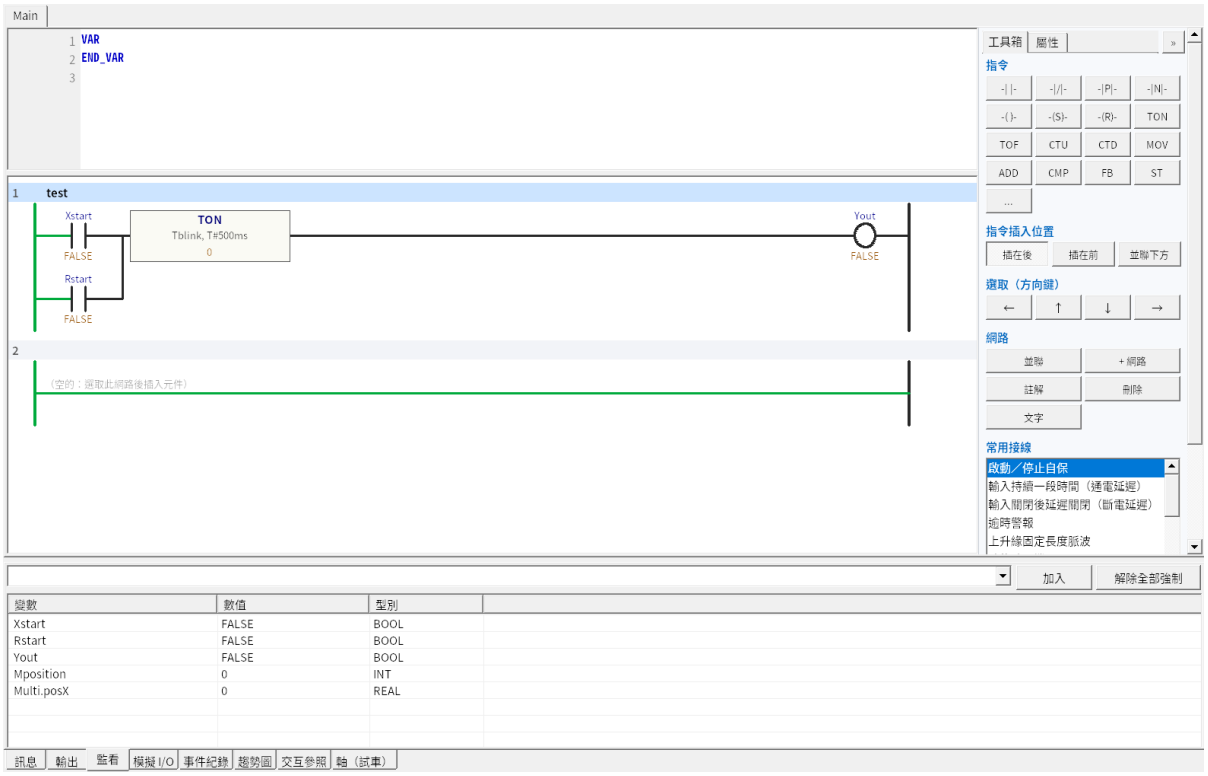


図 10-15　オンライン：ラダー図と Watch ウィンドウにコントローラー上の値が表示される

- **パワーフローと値**：ラダー図の緑色の線と要素の下の値、SFC の緑色のステップ、ST では変数にマウスポインターを合わせたときの値。
- **Watch ウィンドウ**：ダブルクリックで 1 回書き込み。右クリックで強制（[F] を表示）、強制解除、Trace への追加。
- **EtherCAT**：SubDevice ページに各 PDO エントリーのリアルタイムの値が表示されます（11.6 節）。
- **軸**：マシンページ、軸ページ、「Axes」ウィンドウでコントローラー上の軸を操作します（第 12 章）。
- **Logbook**：コントローラーの logbook（10.5 節）。

***注意：**接続が 3 秒間途切れると（USB ケーブルを抜いた、PC がスリープした、など）、コントローラーはすべての強制を自動的に解除します。

10.5 Logbook とウォッチドッグ

10.5.1 Logbook

「**Logbook**」ウィンドウ（下部ツールウィンドウのタブ）には、コントローラーの logbook が表示されます。直近 32 件のイベントが、時刻（コントローラー起動後の秒数）とメッセージ付きで表示されます。シミュレーション中はシミュレーターの記録が表示され、時刻欄には「Simulation」と表示されます。

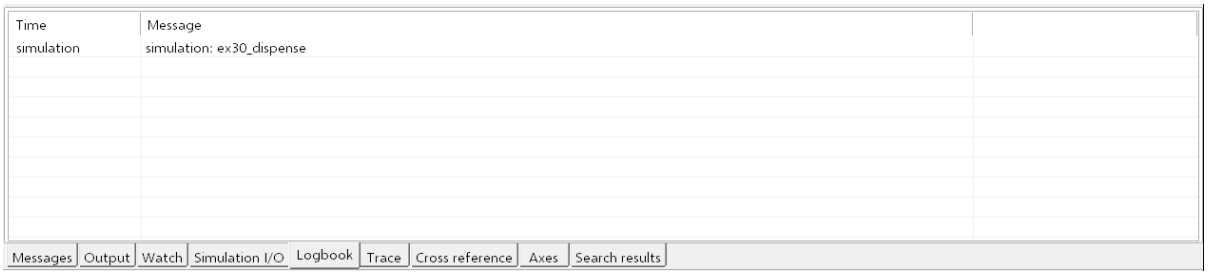


図 10-16　Logbook：EtherCAT メインデバイスの起動メッセージとプログラムの実行開始

イベント	例
電源投入、プログラム開始	`start: MyPlc (build 1A2B3C4D)`（括弧内は今回のビルドの識別子）
RETAIN	`RETAIN: the values kept are loaded`、`RETAIN: nothing stored (first start, or RETAIN changed): initial values`
RUN／STOP（PC から）	`RUN (from the PC)`、`STOP (from the PC)`
ウォームリスタート／コールドリスタート	`warm restart: the variables to their initial values, RETAIN kept`、`cold restart: every variable to its initial value`
接続の切断	`the PC is gone: the forces are released`
Motion86	`Motion86: an axis' drive left operation enabled: the machine stopped (MC_Reset, MC_Power again)` など（12.9 節）
スキャンのタイムアウト	`task MainTask: a scan took 12500 us (the interval is 10000 us)`（初回と 1000 回ごとに 1 件記録）
ウォッチドッグ	`MainTask watchdog: STOP, 152 ms (3 other), Motion 140 ms`
EtherCAT	`EtherCAT: running, 2 slaves, 1000 us, sync, …`、`EtherCAT: begin failed`、`no slave N`
シミュレーションでの実行エラー	`STOP: Main line 12: …`（シミュレーションのみ）

10.5.2　ウォッチドッグ

ウォッチドッグはスキャンのタイムアウトチェックです。周期タスクごとに「Watchdog (us)」を設定でき、1回のスキャンでのプログラムの時間がこの値を超えると PLC は停止（STOP）します。0 はウォッチドッグを使用しないことを表します。プロジェクトツリーの「Tasks」の下でタスクをダブルクリックすると設定できます（4.3 節）。

> ***注意：ウォッチドッグが時間を判定するのは、各スキャンの終了時、プログラムが戻った後です。プログラムがスキャン中に無限ループに入って戻らなくなった場合、ウォッチドッグはそれを中断できません。独立したハードウェアウォッチドッグではなく、装置の安全回路の代わりにもなりません。**
> 稼働中の装置で無限ループをテストしないでください。ウォッチドッグが PLC を停止することを確認するには、有限の長いループを使用できます。

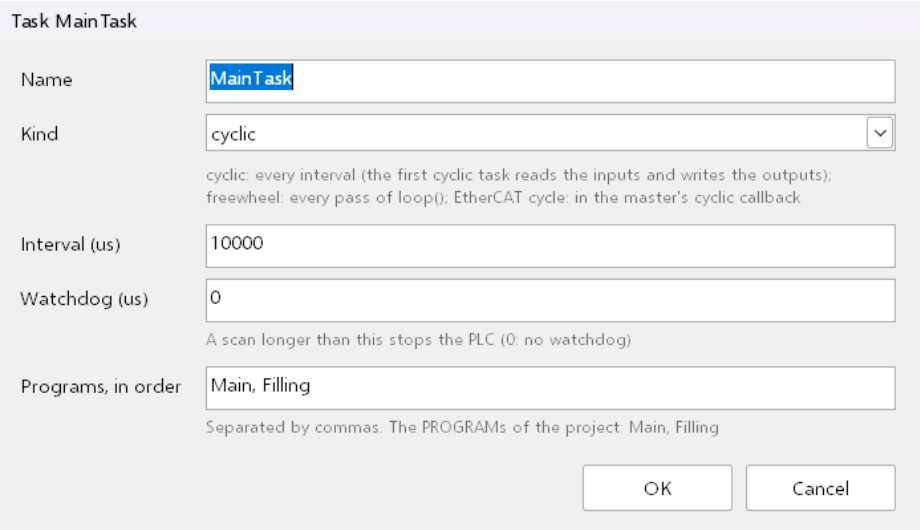


図 10-17　タスクの設定：周期と Watchdog (us)

コントローラーによるウォッチドッグの判定方法：

- 測定するのは、そのタスクの **1 回のスキャンにおけるプログラム自体の時間**です。スキャン全体の時間から、プログラムに属さない時間、すなわち EEPROM の書き込み（PERSIST。86Duino の EEPROM はときどきフラッシュセクターを消去する必要があり、数百 ms 停止することがあります）と EtherCAT 周期割り込み（Motion86、EtherCAT タスク）を差し引いたものです。
- RETAIN の書き込みはスキャンとスキャンの間に行われ、86HMI 画面の更新は最初のタスクのスキャン後に実行されるため、どちらもスキャン時間には**含まれません**。
- 電源投入後の**最初の 3 スキャン**は判定しません（RETAIN の読み込み、変数の初回初期化のため）。
- 設定値を超えた場合：PLC は STOP になり（プログラムは実行されなくなり、出力は最後の値を保持）、すべての Motion86 マシンが停止し、Logbook にウォッチドッグの記録が 1 件書き込まれます。
- シミュレーターはウォッチドッグを判定しません。コントローラー上で確認してください。

Logbook のウォッチドッグ記録の形式：

```
MainTask watchdog: STOP, 152 ms (3 other), Motion 140 ms
```

フィールド	意味
`MainTask`	時間超過したタスク
`152 ms`	今回のスキャンの合計時間
`(3 other)`	そのうち EEPROM の書き込みや EtherCAT 割り込みなど、プログラム以外の時間（ms）。判定には含まれません
`Motion 140 ms`	今回のスキャンで**最も遅かったプログラム**とその時間。まずこのプログラムから原因を探します

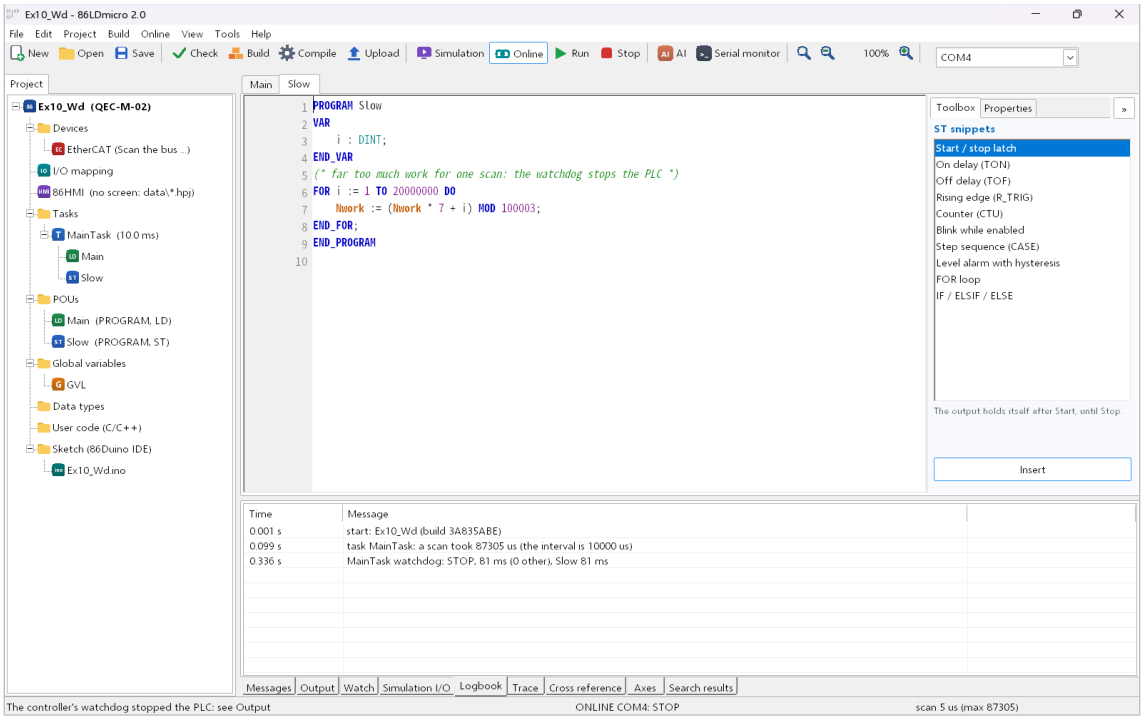


図 10-18　Logbook のウォッチドッグ記録

対処方法	説明
スキャンごとの演算を減らす	長いループを複数のスキャンに分散します（例：1 スキャンあたり 100 件を処理）。必要なときだけ計算します
より遅いタスクに移す	10 ms ごとに実行する必要のないプログラム（統計、レシピ、文字列処理）を 100 ms のタスクに入れます。これは実行頻度を下げるだけで 1 回のスキャン時間は短くならないため、1 回が長すぎる場合はやはり複数のスキャンに分散する必要があります
ウォッチドッグ時間を長くする	スキャン時間が妥当であることを確認したうえで、ウォッチドッグを通常の最大スキャン時間の数倍に設定します
再開する	原因を取り除いた後、RUN（F9）を押すか、もう一度アップロードします

*ヒント：オンライン中のステータスバーには、各スキャンの時間と最大値（「scan … ms (max …)」）、およびタイムアウト回数が表示されます。ウォッチドッグとタスク周期を決める目安にしてください。

10.6　86HMI リソースのアップロード

86HMI Editor で「Code Generation」を行うと、`myhmi.cpp`／`myhmi.h` のほかに、画面で使う画像、フォント、画面データがスケッチの `data\Resource` に書き出されます。これらの**リソース**はプログラムとは別にコントローラー上に保存されます（第 13 章）。

状況	操作
初めてのアップロード、またはプログラムと画面の両方を変更した	**Upload**（F8）をクリックします。プログラムと `data\Resource` が一緒にアップロードされます
画像、フォント、画面の外観だけを変更し、PLC プログラムは変更していない	**Upload the 86HMI resources** をクリックします。リソースだけを送信し、コンパイルは行わず、コントローラー上のプログラムは変わりません（F8 より高速）

1. ツールバーで COM ポート（例：COM4）を確認します。
2. 「Build > Upload the 86HMI resources (data\Resource)」を選択するか、86HMI ページの **Upload resources** をクリックします。
3. 「Build output」に「== Uploading the resources to COM4: …\data\Resource ...」と v86dude の進行状況が表示され、最後に「== Resources uploaded」と表示されます。
4. オンライン中だった場合は、アップロード前に自動的にオフラインになり、完了後に自動的に再接続します。

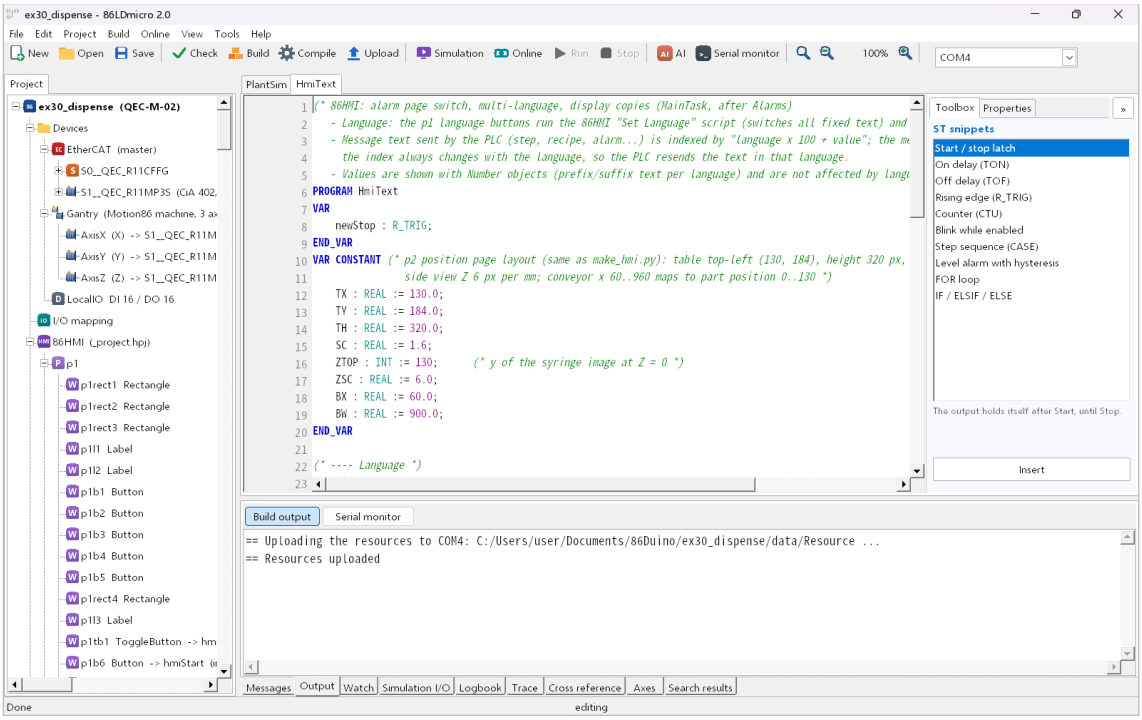


図 10-19　86HMI リソースのアップロード：ビルド出力にアップロードの進行状況と「Resources uploaded」が表示される

*ヒント：この機能は 86Duino IDE の「ツール > QEC M02 へリソースファイルをアップロード」と同じです。「No resources to upload」と表示された場合は、スケッチに `data\Resource` がないか、空です。先に

86HMI Editor で .hpj を開き、「Code Generation」をクリックしてください。コマンドライン：`86ldmicro-cli upload-res <プロジェクト> <COM>`（15.5 節）。

第 11 章　EtherCAT

QEC-M-02 は EtherCAT メインデバイスです。86LDmicro 2.0 のプロジェクトは独自の EtherCAT メインデバイスを持つことができます。内蔵の EtherCAT Configurator でネットワークをスキャンし、サブデバイスを設定して ENI をエクスポートし、その ENI をプロジェクトに読み込んで各 PDO エントリーを変数に割り当てると、プログラムからリモート I/O やサーボドライブを直接読み書きできます。この章では `tools\example\ex30_dispense`（QEC-R11CFFG I/O モジュール＋QEC-R11MP3S 3 軸ドライブ）を例に説明します。

11.1　概念

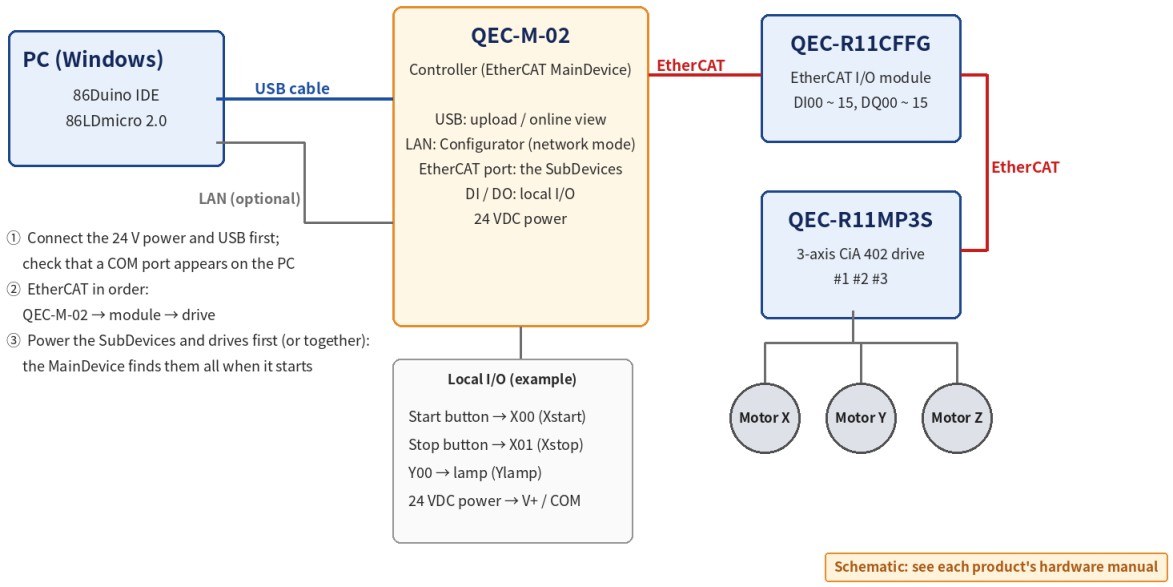


図 11-1　典型的なシステム：PC は USB で QEC-M-02 に接続し、EtherCAT で I/O モジュールとドライブを数珠つなぎに接続する（概略図）

用語	説明
メインデバイス（MainDevice）	QEC-M-02 コントローラー。EtherCAT の周期ごとにすべてのサブデバイスとデータを交換します
サブデバイス（SubDevice）	EtherCAT ネットワーク上の I/O モジュールやサーボドライブなど。例：QEC-R11CFFG、QEC-R11MP3S
PDO（プロセスデータ）	周期ごとに交換されるデータ。RxPDO はメインデバイスからサブデバイスへ（出力）、TxPDO はサブデバイスからメインデバイスへ（入力）送られます
SDO／CoE	非周期のパラメーター読み書き（オブジェクトディクショナリー、Index:Sub）。例：ドライブのパラメーター
ESI	サブデバイスのメーカーが提供する記述ファイル（.xml）：オブジェクトディクショナリー、PDO、アイコン、モジュール定義

用語	説明
ENI	ネットワーク構成ファイル（.xml）：どのサブデバイスがあるか、どの PDO を交換するか、DC 同期、起動パラメーター。コントローラーの `C:/eni/` に保存されます
Configurator	QEC の EtherCAT 設定ツール（Web インターフェース）：Scan、Configure、PDO、DC、CoE、Export ENI
DC（分散クロック）	すべてのサブデバイスを同じタイミングで動作させます。サーボドライブ（CSP）では使用を推奨します

86LDmicro は EtherCAT の 2 つの使用方法に対応しています。

方法	説明	使用する場面
EtherCAT ネットワーク（ENI）	プロジェクト独自のメインデバイス。ネットワークは ENI で記述します。任意の EtherCAT サブデバイス、CiA 402 ドライブ、Motion86 CSP に対応します	2.0 で推奨する方法（本章）
86EVA	1.x の方法：86EVA ツールが `myeva.h／cpp` を生成し、メインデバイスと仮想 I/O を管理します	1.x のプロジェクトを引き続き使用する場合

***注意：**2 つの方法は同時に使用できません（どちらも同じメインデバイスを動かすため）。ENI に切り替えるときは、.ino から 1.x の `#include "myeva.h"` と `EVA.begin()` を削除し、スケッチフォルダーから `myeva.cpp／myeva.h` を移動してください（ビルド時に警告が出ます）。

11.2 全体の流れ

空のネットワークから、プログラムが PDO を読み書きできるようになるまで、次の手順を順に行います。最初の 5 ステップは Configurator（11.3 節）で行い、その後 86LDmicro に戻ります。

ステップ	場所	作業内容	節
1	86LDmicro の EtherCAT ページ	接続方法を選択し、**Upload Configurator** をクリックします（1 回のみ）	11.3.1
2	Configurator	（推奨）サブデバイスの ESI をアップロードして読み込む	11.3.5
3	Configurator	**Scan** でサブデバイスをスキャン → **Configure**	11.3.3
4	Configurator	必要に応じて PDO、DC、Startup、Slots を設定し、**Start** でテストする	11.3.6
5	Configurator	**Export ENI** → **Upload** でコントローラーの `C:/eni/` **に保存する**	11.3.5
6	86LDmicro	**Take it into the project**：サブデバイスと PDO をプロジェクトに読み込む	11.4
7	86LDmicro	PDO を変数に割り当てる。CiA 402 ドライブには Motion86 マシンを作成する	11.4、第 12 章
8	86LDmicro	PLC プログラムをビルドしてアップロードする（Configurator と置き換わる）。メインデバイスの状態を確認する	11.4.4
9	86LDmicro の SubDevice ページ	オンラインで PDO の値を確認、書き込み／強制、SDO、CoE オブジェクトディクショナリー	11.5、11.6

11.2.1　ステップ 1：EtherCAT ページを開き、Configurator をアップロードする

「Project > EtherCAT (Scan, PDO, ENI)」を選択するか、プロジェクトツリーの「Devices > EtherCAT」をダブルクリックします。接続方法を選択して **Upload Configurator** をクリックします。

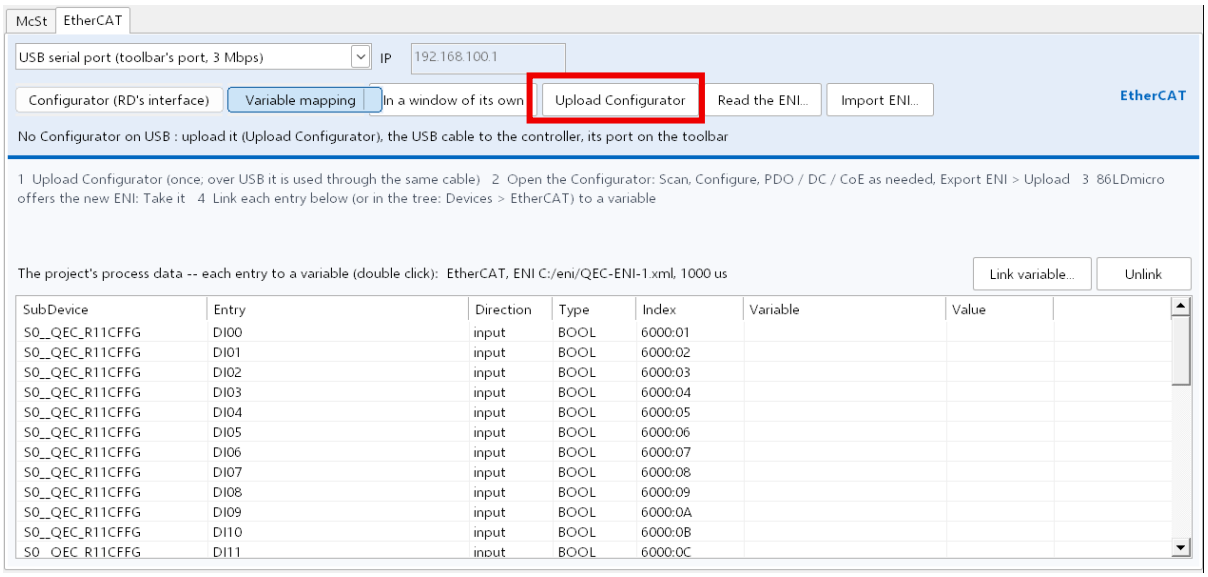


図 11-2　EtherCAT ページ：接続方法、Configurator (RD's interface)、Variable mapping、Upload Configurator、Read the ENI、Import ENI

11.2.2　ステップ 2：タブ内で Configurator を開く

Configurator (RD's interface) をクリックすると、Configurator の Web ページが EtherCAT ページ内に直接表示されます。

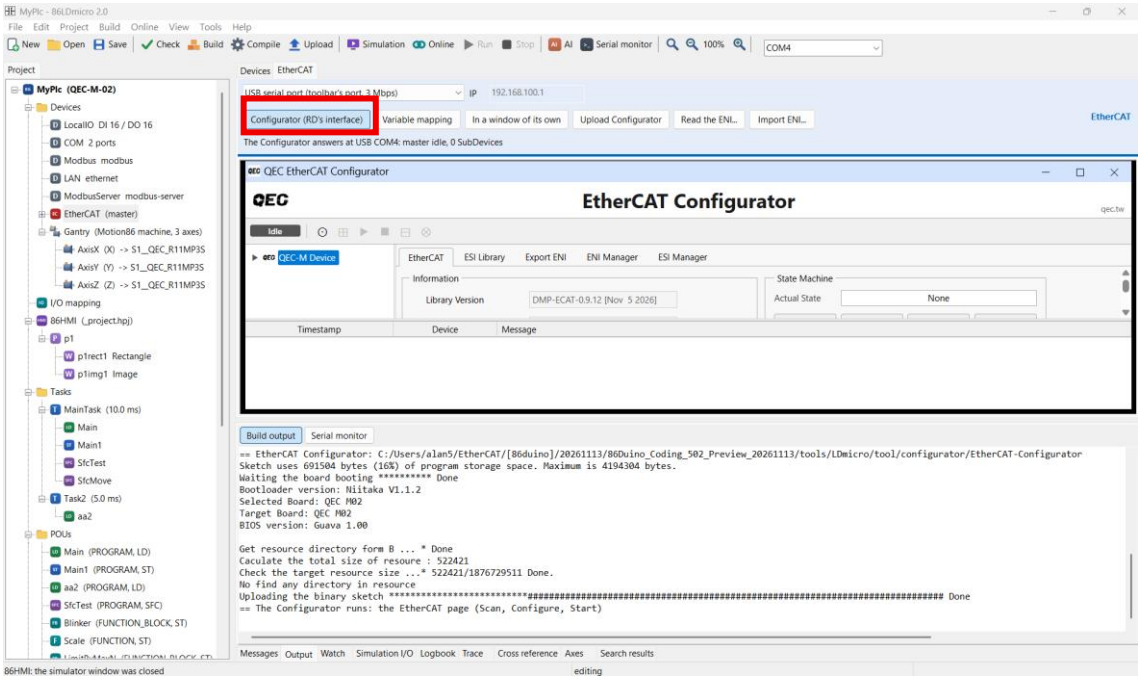


図 11-3　EtherCAT ページに埋め込まれた Configurator の Web ページ（USB モード）

In a window of its own をクリックすると、Configurator が独立したページで表示されます。

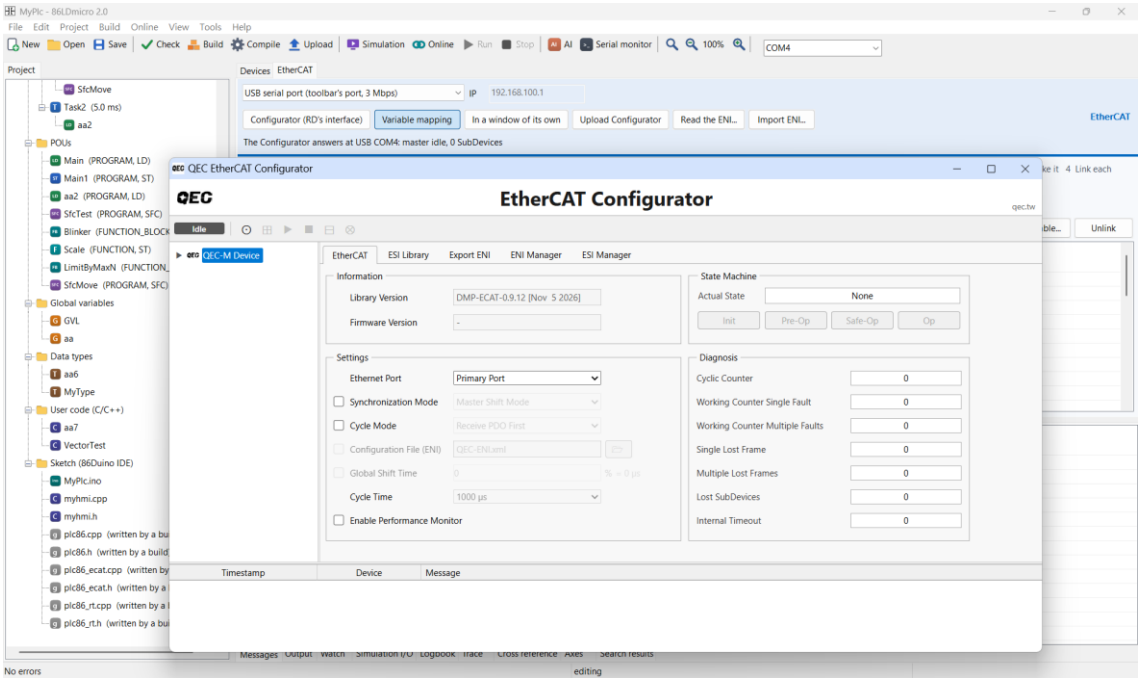


図 11-3-1　独立ページで表示した Configurator の Web ページ（USB モード）

11.2.3　ステップ 3、4：Scan、Configure、Start でテスト

Configurator で **Scan** をクリックすると、左側のデバイスツリーにすべてのサブデバイスが一覧表示されます。



図 11-4-1　ステータスラベル Idle（未スキャン）

Configure をクリックして PDO を読み込みます。



図 11-4-2　ステータスラベル Init

Start をクリックすると、すべてのサブデバイスが OP 状態になります。



図 11-4-3　ステータスラベル Config（Configure 完了）

すべてのサブデバイスが OP 状態になります。



図 11-4-4　ステータスラベル Run（周期交換中）

左側のデバイスツリーにすべてのサブデバイスが一覧表示されます。

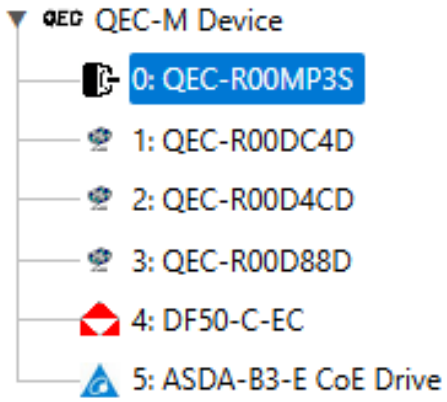


図 11-4-5　Scan 後：デバイスツリーに各サブデバイスが表示される

サブデバイスの **Online** タブでリアルタイムの値を確認します。

EtherCAT | DC | Process Data | Startup | CoE | E²PROM | ESC Access | Online | CiA 402

RxPDO (Outputs)

Index	Type	Name	Value
0x6040:00	UINT16	Controlword	0x0000 (0)
0x607A:00	INT32	Target position	0
0x60FF:00	INT32	Target velocity	0
0x6060:00	UINT8	Modes of operation	0x00 (0)
0x6840:00	UINT16	Controlword	0x0000 (0)
0x687A:00	INT32	Target position	0
0x68FF:00	INT32	Target velocity	0
0x6860:00	UINT8	Modes of operation	0x00 (0)
0x7040:00	UINT16	Controlword	0x0000 (0)
0x707A:00	INT32	Target position	0
0x70FF:00	INT32	Target velocity	0
0x7060:00	UINT8	Modes of operation	0x00 (0)

TxPDO (Inputs)

Index	Type	Name	Value
0x6041:00	UINT16	Statusword	0x0008 (8)
0x6064:00	INT32	Position actual value	0
0x606C:00	INT32	Velocity Actual Value	0
0x60E4:01	INT32	First additional position	0
0x60FD:00	UINT32	Digital inputs	0x00070000 (4...
0x6061:00	UINT8	Modes of operation disp...	0x00 (0)
0x6841:00	UINT16	Statusword	0x0008 (8)
0x6864:00	INT32	Position actual value	0
0x686C:00	INT32	Velocity Actual Value	0
0x68E4:01	INT32	First additional position	0
0x68FD:00	UINT32	Digital inputs	0x00070000 (4...
0x6861:00	UINT8	Modes of operation disp...	0x00 (0)
0x7041:00	UINT16	Statusword	0x0008 (8)
0x7064:00	INT32	Position actual value	0
0x706C:00	INT32	Velocity Actual Value	0
0x70E4:01	INT32	First additional position	0
0x70FD:00	UINT32	Digital inputs	0x00070000 (4...

図 11-4-6　Start 後：サブデバイスの Online タブにリアルタイムの PDO 値が表示される

11.2.4　ステップ 5：ENI をエクスポートしてコントローラーにアップロードする

「**QEC-M Device > Export ENI**」タブをクリックします。

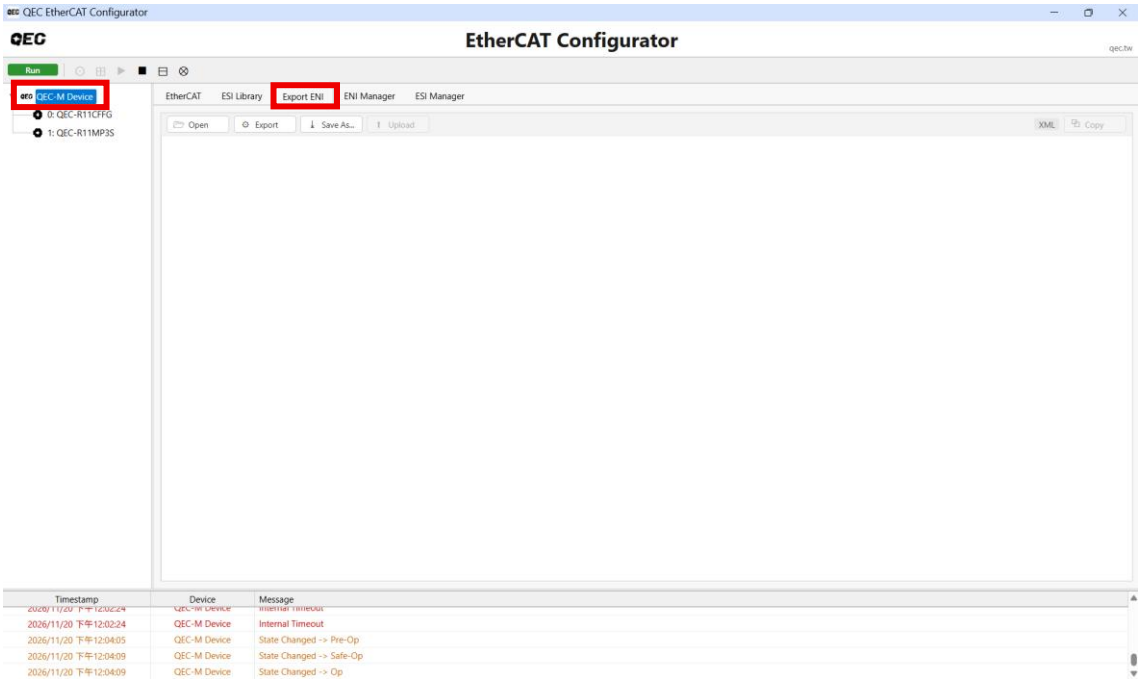


図 11-5-1　Export ENI タブ

Export をクリックして ENI を生成します。

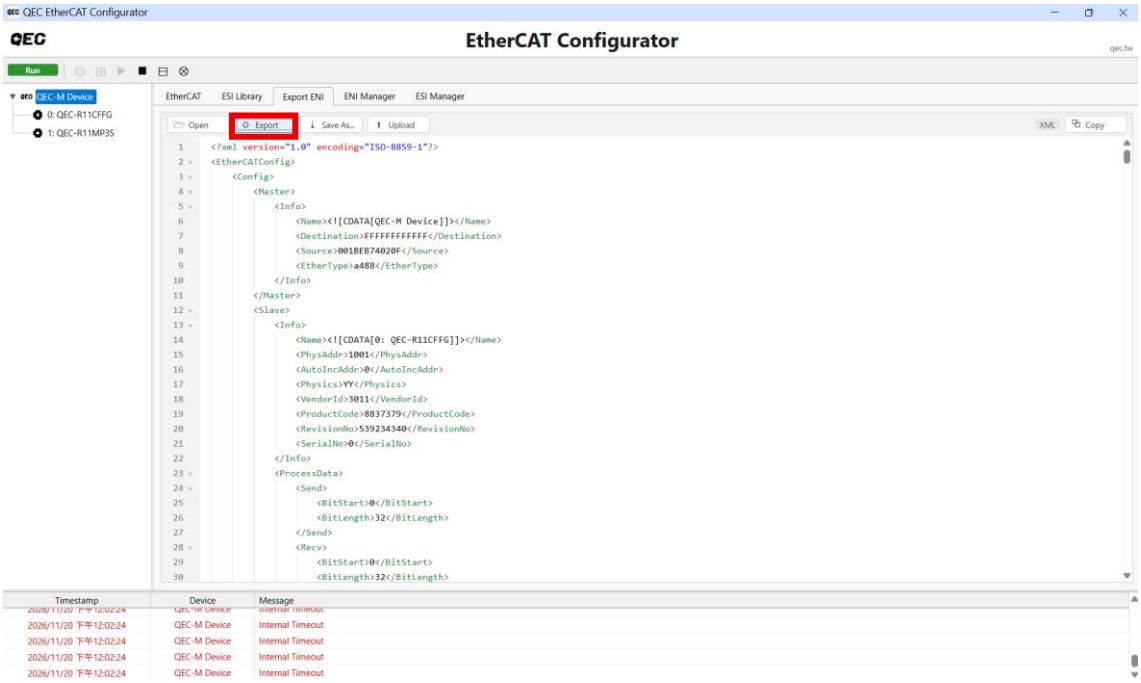


図 11-5-2　Export をクリックして ENI を生成

続いて **Upload** をクリックし、ファイル名を入力してアップロードします。

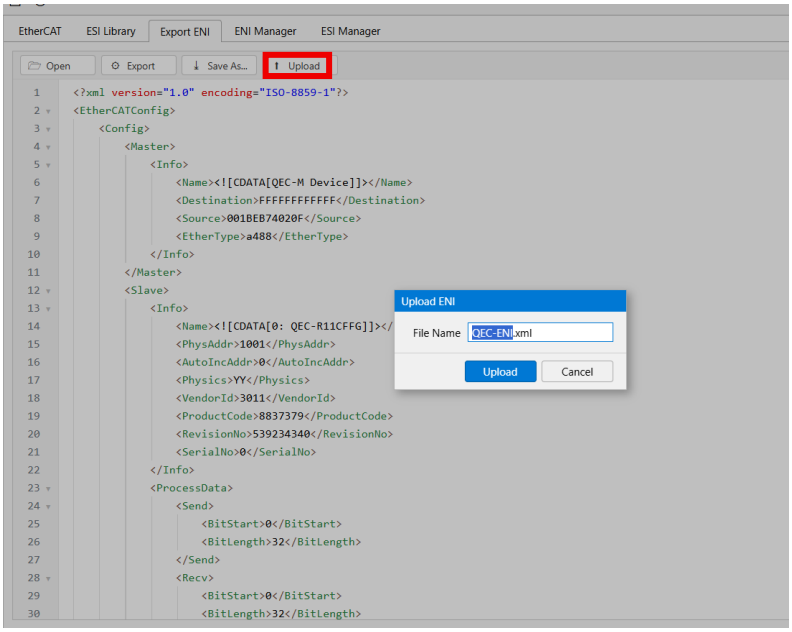


図 11-6　Upload ENI：ファイル名を入力し、コントローラーの C:/eni/ に保存する

11.2.5　ステップ 6、7：プロジェクトに読み込み、変数を割り当てる

86LDmicro に戻ると、コントローラー上で新しい ENI が検出されたときにメッセージが表示されるので、**Take it into the project** をクリックします。左側のプロジェクトツリーの EtherCAT カテゴリーに、ENI に対応するサブデバイス名が表示されます。

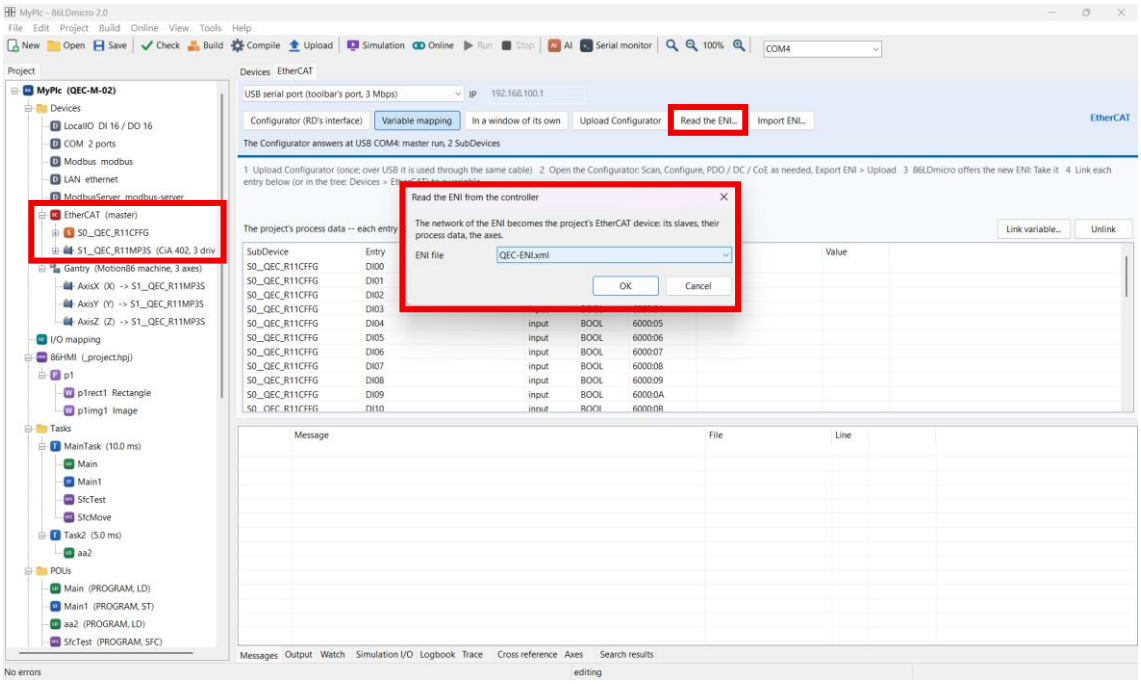


図 11-7-1　コントローラー上で新しい ENI を検出：Take it into the project

続いて「Variable mapping」で各 PDO エントリーを変数に割り当てます。

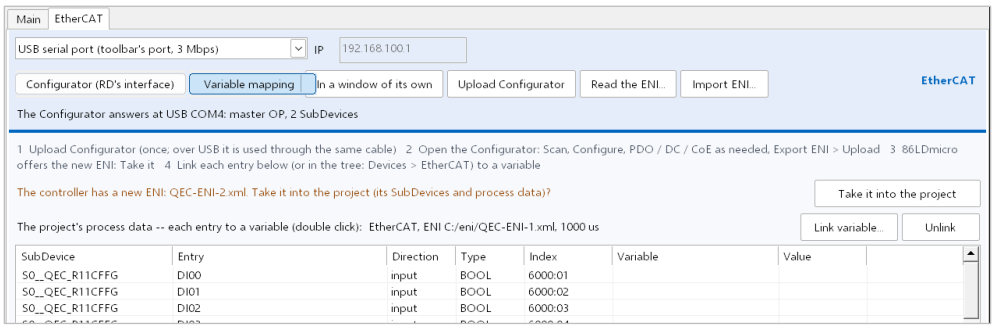


図 11-7-2　コントローラー上で新しい ENI を検出：Take it into the project

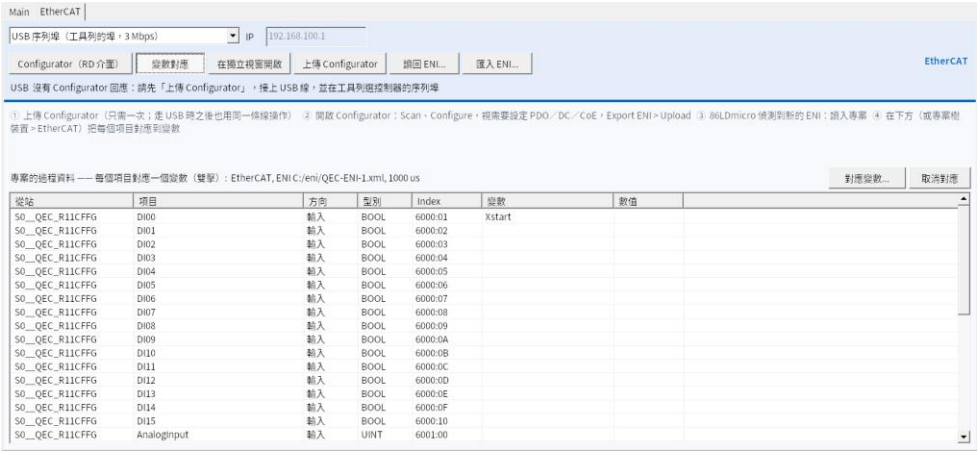


図 11-8　「Variable mapping」：PDO エントリーごとに 1 行。割り当て済みのものには変数が表示される

11.2.6　ステップ 8、9：PLC プログラムをアップロードして確認する

Upload（F8）をクリックします。

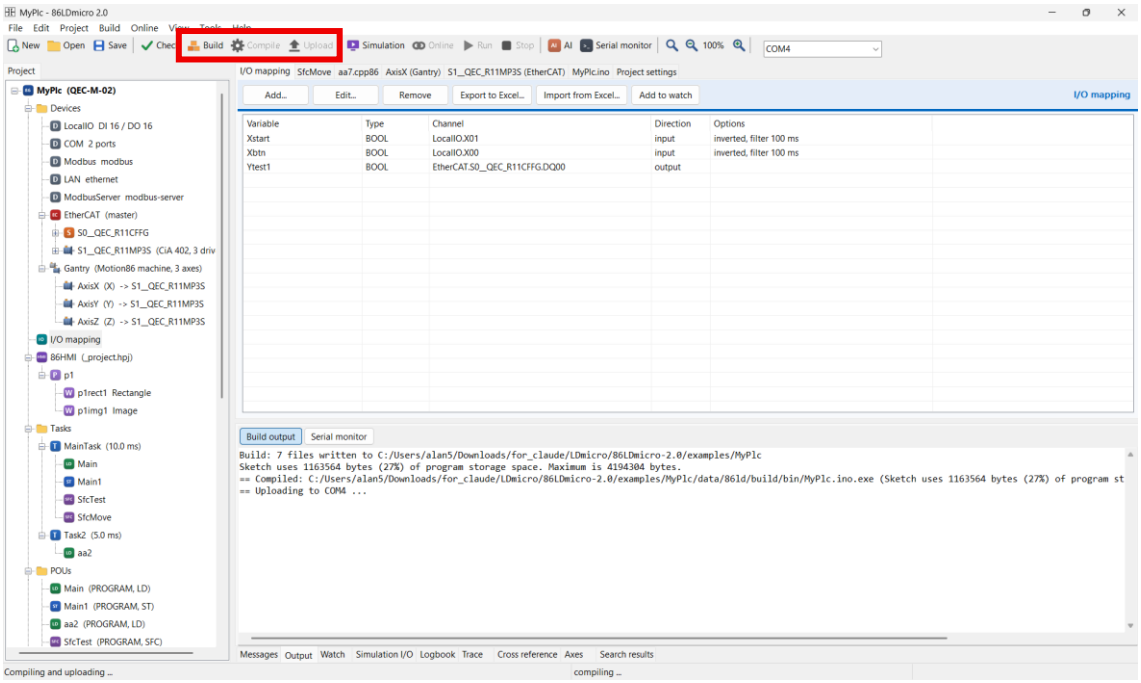


図 11-9-1　アップロード

PLC プログラムが Configurator と置き換わり、電源投入時にメインデバイスは ENI で起動します。「Serial monitor」を開いてメインデバイスの状態を確認し、オンラインにしてから SubDevice ページで PDO のリアルタイムの値を確認します。

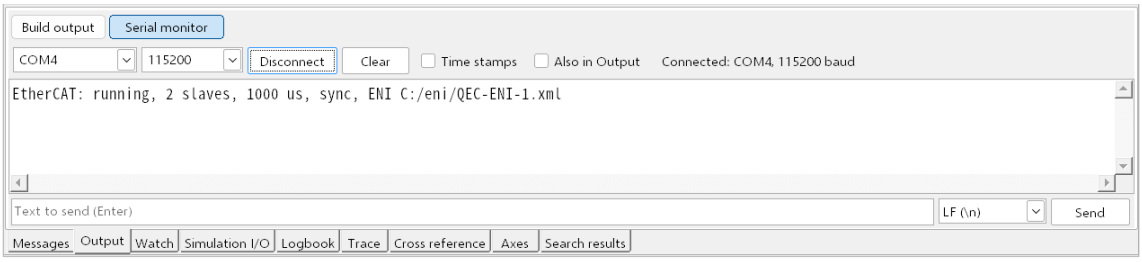


図 11-9　シリアルモニター：EtherCAT: running, 2 slaves, 1000 us, sync

11.3 EtherCAT Configurator

EtherCAT Configurator は QEC コントローラー上で動作する設定プログラムで、Web ページで操作します。86LDmicro はこれをコントローラーにアップロードし、Web ページを EtherCAT ページ内に直接埋め込みます。ネットワークの設定が完了したら ENI をエクスポートし、86LDmicro に渡します。

11.3.1 アップロードと起動

接続方法	説明
USB serial port (toolbar's port, 3 Mbps)	既定。アップロードやオンラインモニターと同じ USB ケーブルを使います。86LDmicro が PC 上で Web 画面を提供し、API だけがシリアルポート経由でコントローラーに送られます
Ethernet (the controller's IP)	PC をコントローラーの LAN ポートに接続し、コントローラーの IP（既定値 `192.168.100.1`）を入力します

1. EtherCAT ページで接続方法を選択します（USB の場合は、ツールバーでコントローラーの COM ポートを選択します）。
2. **Upload Configurator** をクリックします（1 回のみ）。86LDmicro が Configurator プログラムをコンパイルしてアップロードします。これはコントローラー上の PLC プログラムを**一時的に置き換えます**。「Tools > EtherCAT > Upload the Configurator to the controller...」でも実行できます。
3. アップロードが完了すると、ページは Configurator が応答するかどうかとメインデバイスの状態を自動的に検出します。例：「The Configurator answers at COM4: master Idle, 0 SubDevices」。
4. **Configurator (RD's interface)** をクリックすると、Configurator の Web ページがこのタブ内に表示されます（Microsoft Edge で埋め込み）。**In a window of its own** をクリックすると独立したウィンドウに切り離せます（2 台目のモニターにドラッグするなど）。**Variable mapping** をクリックすると PDO 一覧に戻ります。

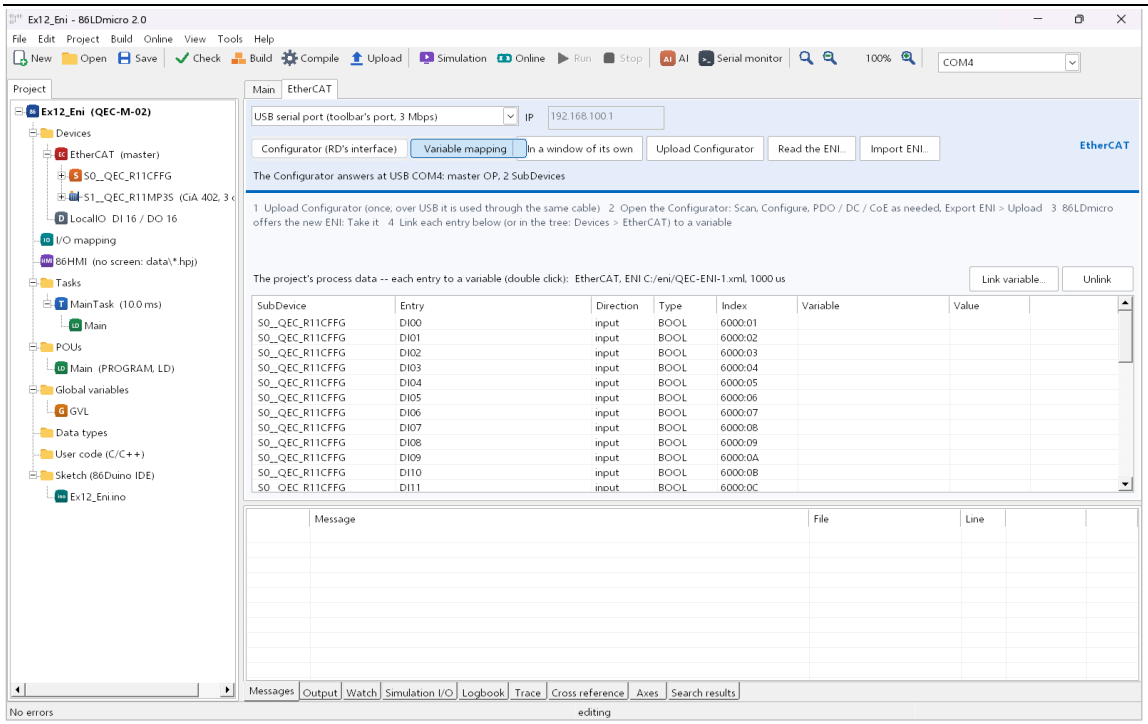


图 11-10　Upload Configurator：ビルド出力の進行状況と、完了後にページに表示される応答メッセージ

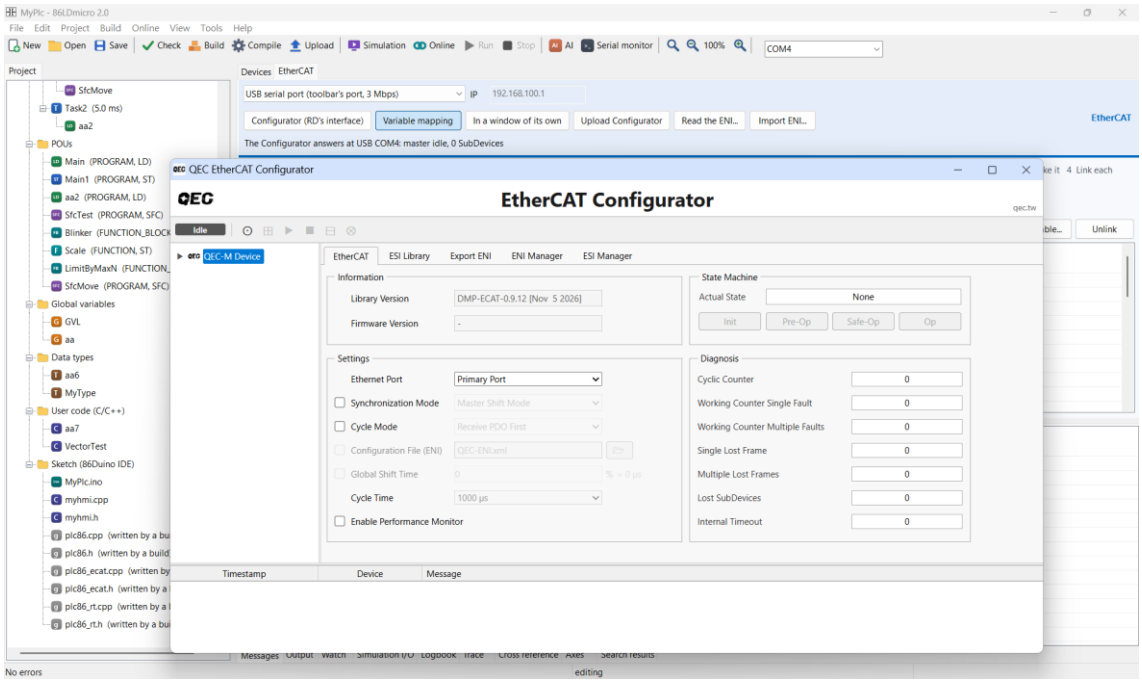


图 11-11　「In a window of its own」：Configurator 独立したウィンドウで表示

***ヒント：**USB モードでは、アップロードやオンラインモニターの間は Configurator の接続が自動的に一時停止します。EtherCAT ページが前面にないときもシリアルポートを占有しません（2 秒間アイドルで解放）。

ネットワークモードでは、ブラウザーで開くこともできます：「Tools > EtherCAT > The Configurator's web page (browser)...」。

11.3.2　画面

Configurator の画面は、上から順に 5 つの領域に分かれています。

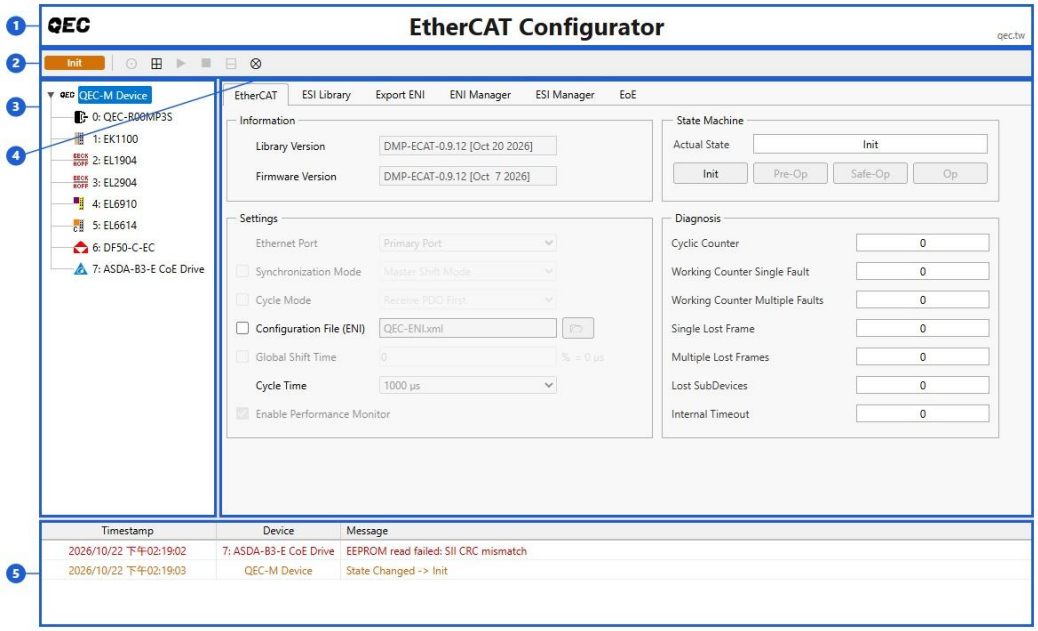


図 11-12　Configurator の画面：① タイトル ② メインツールバー ③ デバイスツリー ④ タブの内容 ⑤ ログ

領域	内容
① タイトル	QEC のロゴとタイトル
② メインツールバー	メインデバイスの状態と 6 つの操作ボタン：Scan、Configure、Start、Stop、Deconfigure、Disconnect
③ デバイスツリー	メインデバイス（QEC-M Device）とスキャンされたサブデバイス
④ タブの内容	デバイスツリーで選択したノードの設定タブ（タブはノードの種類によって異なります）。タブが多すぎる場合は、マウスホイールで左右にスクロールするか、右側の ▾ をクリックしてすべてのタブを一覧表示します
⑤ ログ	操作ログとシステムイベント（11.3.7）

11.3.3　メインツールバーと状態

ツールバー左側のステータスラベルには、メインデバイスの現在の状態が表示されます：**Idle**、**Init**、**Config**、**Run**、**Bootstrap**。各ボタンは対応する状態のときだけクリックでき、それ以外のときはグレー表示になります。

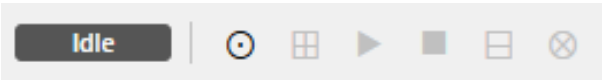


図 11-13　ステータスラベル Idle（未スキャン）

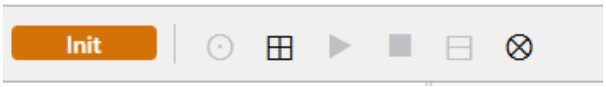


図 11-13-1　ステータスラベル Init



図 11-14　ステータスラベル Config（Configure 完了）



図 11-15　ステータスラベル Run（周期交換中）

ボタン	アイコン	説明
Scan	⊙	バス上のすべてのサブデバイスをスキャンし、デバイスツリーにノードを作成します
Configure	⊞	ENI ファイル（またはオンライン構成）に従って各サブデバイスを設定し、PDO データを読み込みます
Start	▶	周期的なプロセスデータ交換を開始し、サブデバイスが Op になります
Stop	■	周期的な交換を停止し、Safe-Op に戻ります
Deconfigure	⊟	構成を元に戻し、CiA 402 の軸オブジェクトを解放して Init に戻ります
Disconnect	⊗	バスとの接続を切断し、デバイスツリーをクリアします

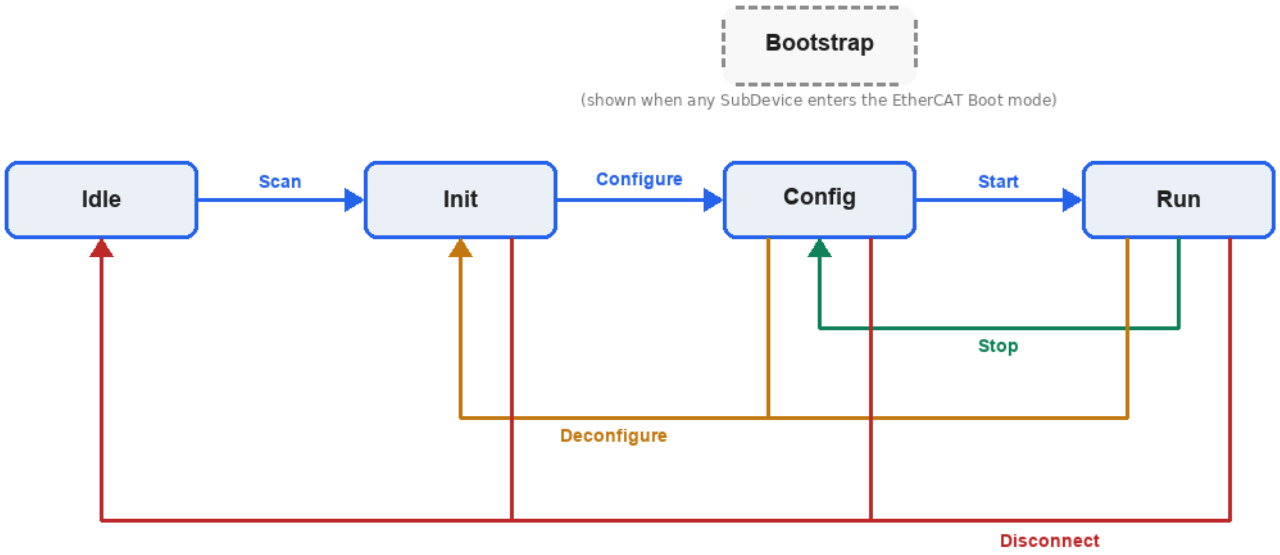


図 11-16　メインデバイスの状態遷移：四角は状態、矢印は押したボタン

デバイスツリーの **QEC-M Device** を右クリックしても、同じ 6 つの操作を実行できます。操作を開始すると進行状況ウィンドウ（例：Scan Network）が表示され、`Step:` 現在のステップ / 全ステップ、プログレスバー、処理中のデバイスが表示されます。完了（または失敗）すると自動的に閉じます。

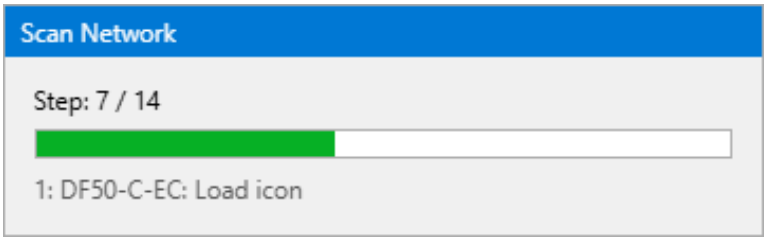


図 11-17　進行状況ウィンドウ

11.3.4　デバイスツリー

- Scan 前のノードは **QEC-M Device**（メインデバイス）の 1 つだけです。
- Scan に成功すると、サブデバイスごとに 1 つのノードができ、番号：サブデバイス名 という名前になります（例：`0: QEC-R11CFFG`、`1: QEC-R11MP3S`）。
- ノードをクリックすると、中央にその設定タブが表示されます。サブデバイスを右クリックして Rename を選択すると名前を変更できます。

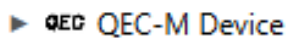


図 11-18　Scan 前のデバイスツリー：QEC-M Device のみ

11.3.5　QEC-M Device（メインデバイス）のタブ

QEC-M Device を選択すると、EtherCAT、ESI Library、Export ENI、ENI Manager、ESI Manager の各タブが表示されます。EoE に対応したサブデバイスがある場合は EoE タブが、パフォーマンス監視を有効にして Run 状態のときは Performance タブが追加されます。

EtherCAT タブ：左半分は Information（ライブラリとファームウェアのバージョン）と Settings、右半分は State Machine と Diagnosis（診断カウンター）です。

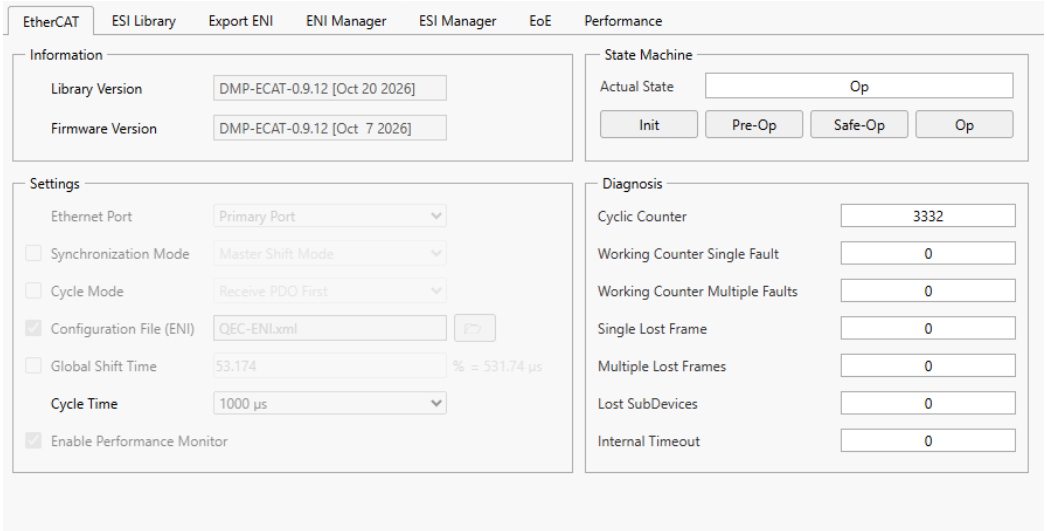


図 11-19　QEC-M Device の EtherCAT タブ：設定、ステートマシン、診断カウンター

設定	説明	変更できるとき
Ethernet Port	Primary Port／Secondary Port／Cable Redundancy	Idle
Synchronization Mode	Master Shift Mode／Bus Shift Mode（Scan 時に使用）	Idle
Cycle Mode	Receive PDO First／Send PDO First	Idle
Configuration File (ENI)	オンにしてパスを入力するか 🗁 をクリックし、コントローラー上の ENI を選択します。Configure 時に使用します	Init
Global Shift Time	パーセントで入力し、横に換算した µs が表示されます（Start 時に使用）	Config
Cycle Time	125～10000 µs	Config
Enable Performance Monitor	オンにすると、Run 状態で Performance タブが表示されます	Idle

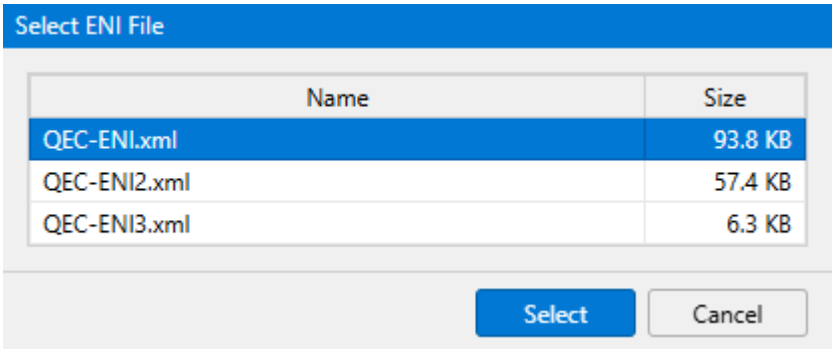


図 11-20　Select ENI File：コントローラー上の ENI ファイルを選択する

State Machine の `Init`／`Pre-Op`／`Safe-Op`／`Op` ボタンで、すべてのサブデバイスを強制的にその状態にできます。Diagnosis には Cyclic Counter、Working Counter エラー、失われたフレーム、失われたサブデバイスなどのカウントが表示されます。

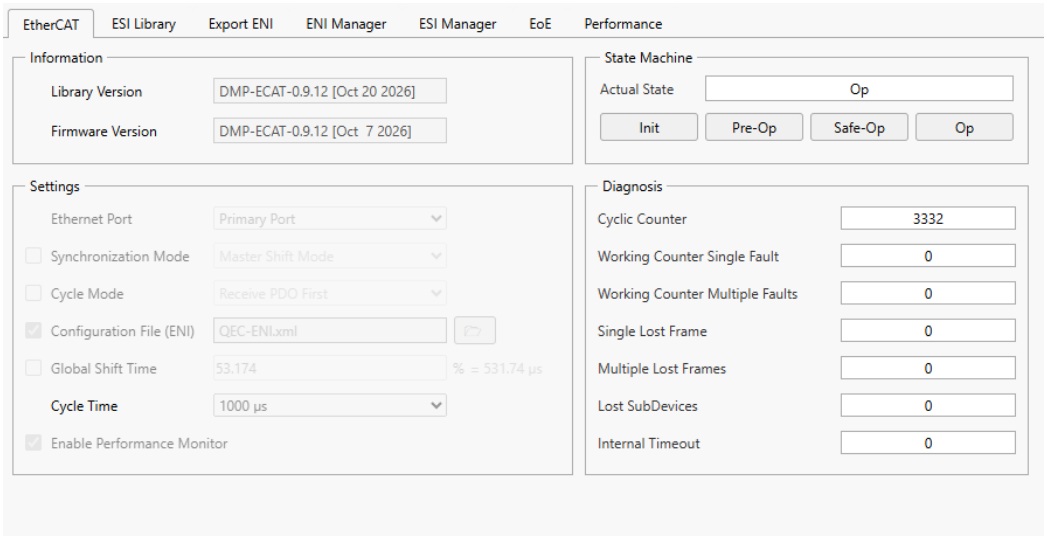


図 11-20-1　QEC-M Device の EtherCAT タブ：設定、ステートマシン、診断カウンター

ESI Manager と ESI Library タブ：ESI を読み込んでおくと、Scan 時にオブジェクトディクショナリー、サブデバイスのアイコン、Startup コマンド、Slots のモジュール定義を取得できます（読み込みを推奨）。コントローラーは電源投入時に ESI を自動的には読み込み**ません**。

1. **ESI Manager** タブ：Upload to Remote で **Browse** をクリックしてローカルの ESI ファイル（.xml）を選択し、**Upload** をクリックしてコントローラーにアップロードします。アップロード中は進行状況ウィンドウが表示されます。
2. **ESI Library** タブ：ESI Tree の空白部分を右クリックして **Load ESI** を選択します（Idle 状態）。読み込み中は Loading ESI ウィンドウが表示されます。
3. ツリー構造は Vendor → Group → Device（モジュールの場合は Vendor → Modules → Module）です。Device をクリックすると、Product Code、PDO 数、Slots、Mailbox、DS402 Channels などを確認できます。

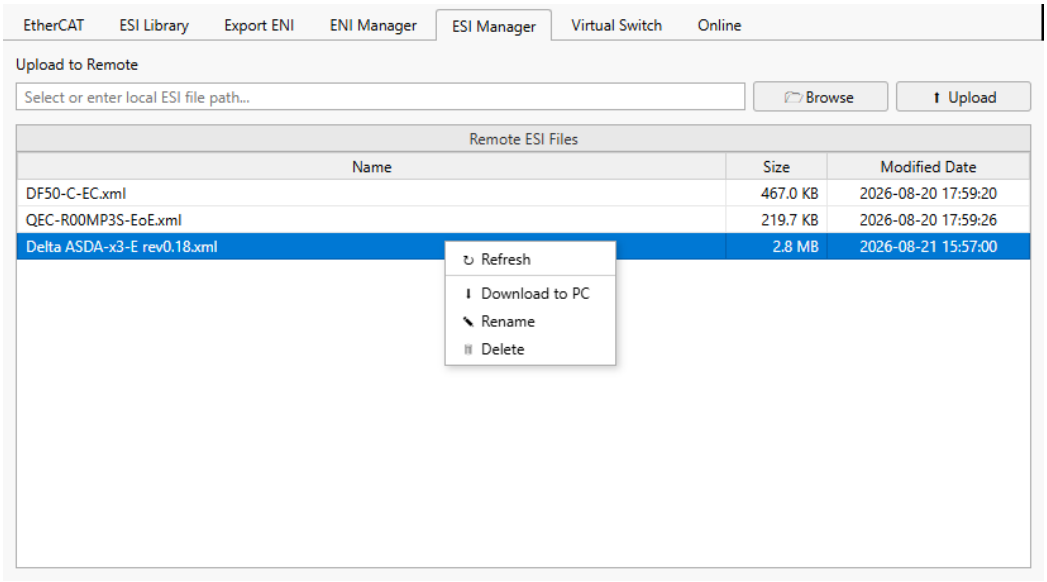


図 11-21　ESI Manager：コントローラー上の ESI ファイル

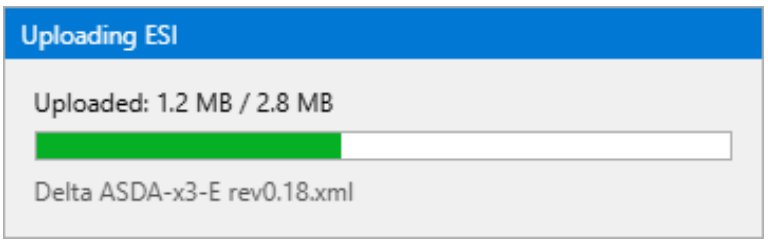


図 11-22　アップロードの進行状況ウィンドウ

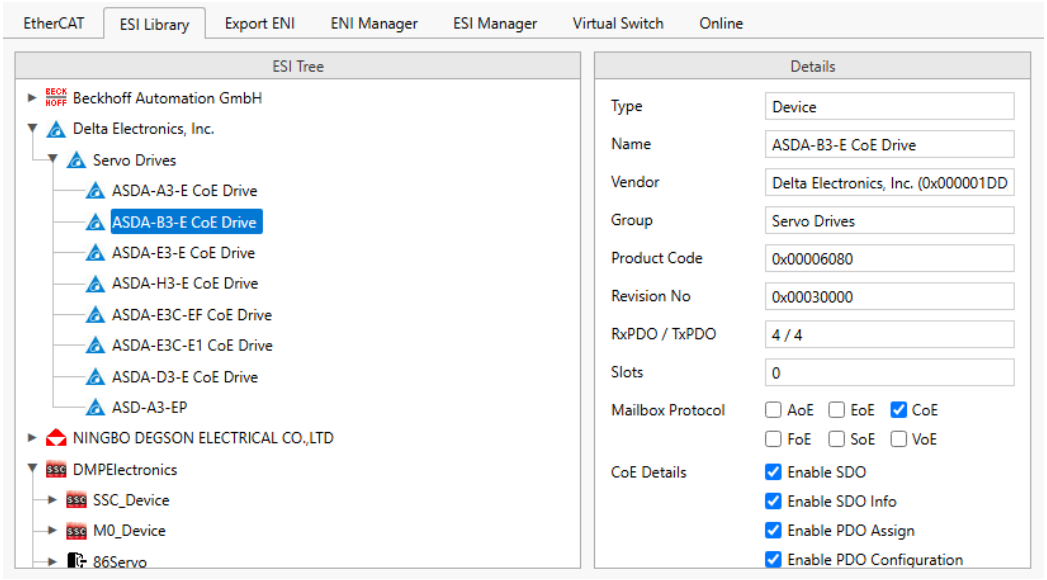


図 11-23　ESI Library：読み込まれた ESI

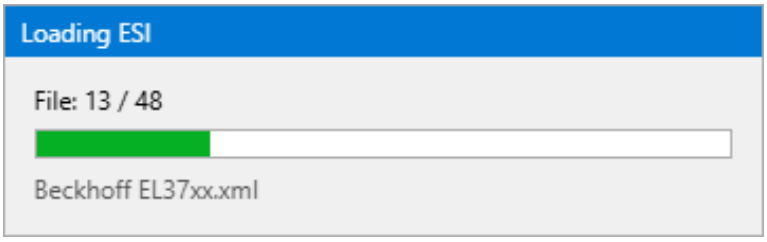


図 11-24　Loading ESI：読み込みの進行状況

*ヒント：ESI Manager で ESI をアップロード、名前変更、削除した後は、ESI Library で **Unload ESI** を実行してから **Load ESI** を実行しないと反映されません。

Export ENI タブ：現在の構成から ENI を生成し、コントローラーにアップロードします（11.2.4）。

ボタン	説明
Open（🗁）	コントローラー上の ENI ファイルを選択して表示します
Export（⚙）	現在のデバイスツリーのデータ（Slots、Process Data、Startup、DC など）から完全な ENI XML を生成します
Save As...（⭳）	表示中の XML をパソコン上に `QEC-ENI.xml` として保存します
Upload（⬆）	ファイル名を入力してコントローラー（`C:/eni/`）にアップロードします
Copy（⧉）	表示中の XML をコピーします

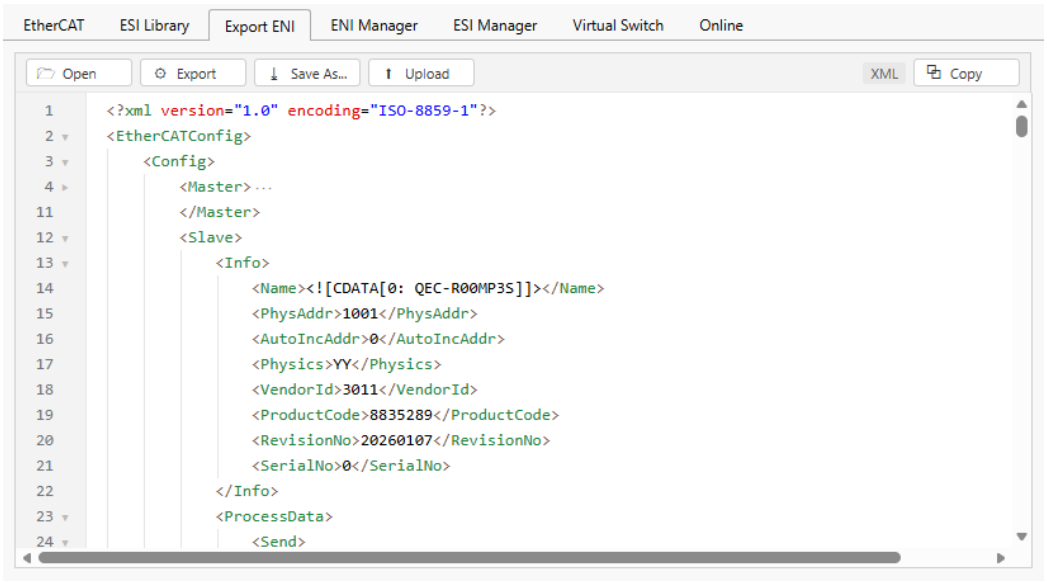


図 11-25　Export ENI タブ：生成された ENI XML

ENI Manager タブ：コントローラー上の ENI ファイルを管理します。表は名前、サイズ、日付で並べ替えられます。右クリックで Refresh、Download to PC、Rename、Delete を実行できます。

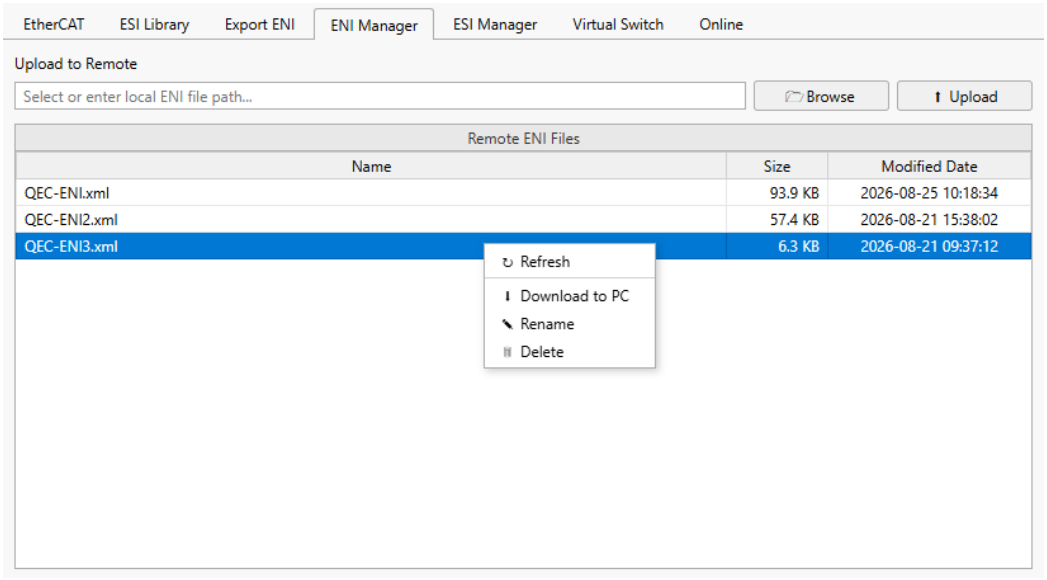


図 11-26　ENI Manager：コントローラー上の ENI ファイル

EoE タブ：EoE（Ethernet over EtherCAT）に対応したサブデバイスがある場合に表示されます。チェックを入れて仮想スイッチに追加し、MAC、IP、サブネットマスク、ゲートウェイ、DNS を設定して Set をクリックすると書き込まれます（Config／Run 状態）。

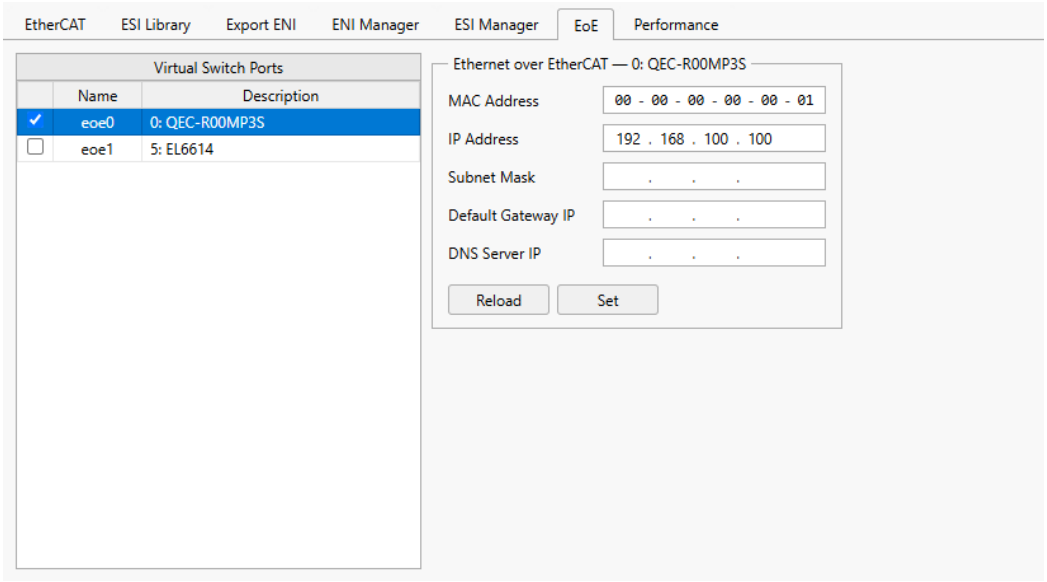


図 11-27　EoE タブ：仮想スイッチとサブデバイスのネットワーク設定

Performance タブ：EtherCAT タブで Enable Performance Monitor にチェックを入れ、メインデバイスが Run のときに表示され、500 ms ごとに更新されます。各周期の各段階（Receive PDO、User Application、Send PDO、Process Acyclic）の最小／平均／最大時間、実際の周期と Callback Jitter、および周期のタイムラインと直近約 60 秒のジッターグラフを表示します。

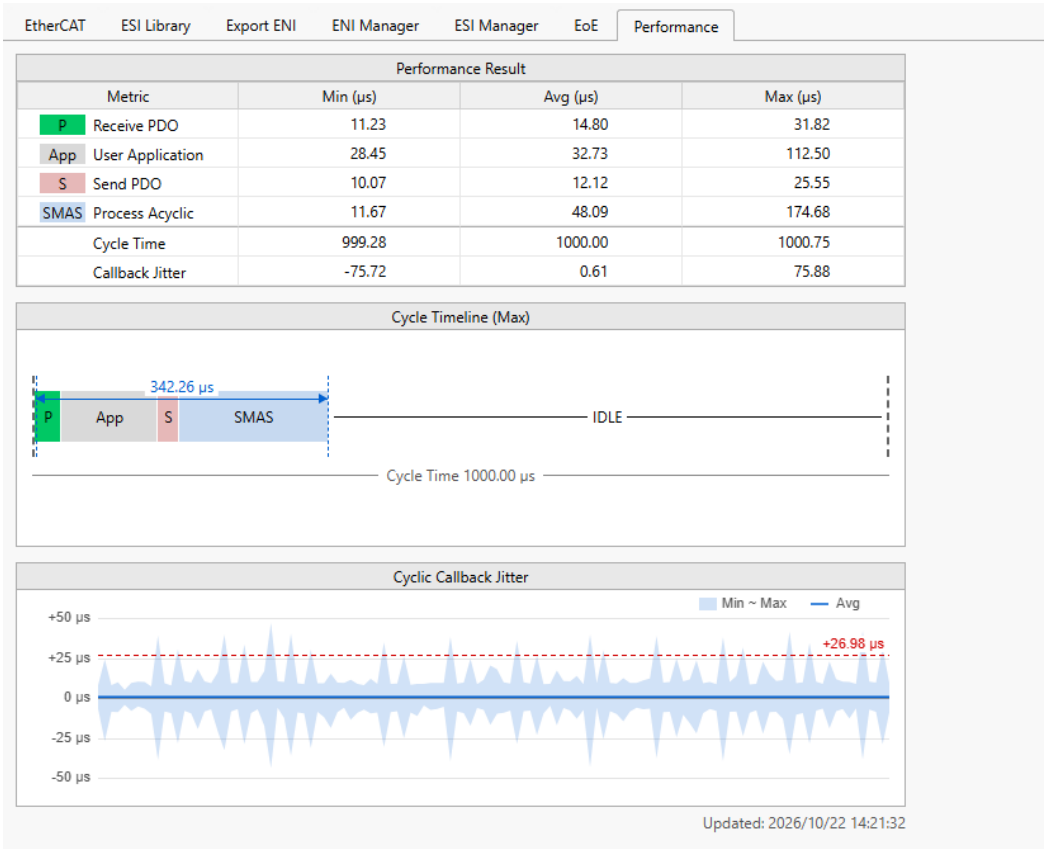


図 11-28　Performance タブ：周期の各段階の時間とジッター

11.3.6 サブデバイス（SubDevice）のタブ

サブデバイスを選択すると、EtherCAT、DC、E²PROM、ESC Access、Online のタブが表示されます。サブデバイスによっては、Slots（ESI に Slots が定義されている場合）、Process Data／Startup／CoE（CoE 対応の場合）、CiA 402（CiA 402 対応で、メインデバイスが Config／Run の場合）も表示されます。

タブ	用途
EtherCAT	サブデバイス情報（名前、Vendor ID、Product Code、Revision、Serial、DS402 Channels、対応する ESI ファイル）、Mailbox プロトコル、ステートマシン（Init／Pre-Op／Safe-Op／Op、Bootstrap）、FoE によるファイルの読み書き
DC	分散クロック：Enable SYNC0、SYNC0 の周期倍率と Shift Time、SYNC1。Start 時に書き込まれます（ENI 使用時は ENI の設定が優先されます）
Slots	モジュール式サブデバイス：Slot を右クリックして Insert でモジュールを選択すると、Startup コマンドが自動生成されます
Process Data	PDO マッピング：交換する RxPDO（0x1C12）／TxPDO（0x1C13）にチェックを入れ、右クリックで Insert／Delete／Move／Edit を実行します。右側には選択した PDO の内容が表示されます
Startup	Pre-Op → Safe-Op 時にサブデバイスへ書き込む SDO コマンド（Index、CA、Data、Comment）。ENI を使用しない場合は Start 時に書き込まれ、エクスポートした ENI にも出力されます
CoE	オブジェクトディクショナリ：Refresh、Upload、Download、Download (CA)。ソースは Load from ESI／Load from SDO Info で切り替えられます
E²PROM	Station Alias、ESI からの EEPROM の生成・読み取り・書き込み、Hex エディター
ESC Access	ESC レジスタ（上級者向け）：読み取り、書き込み、ビットフィールドのデコード
Online	Run 時に 200 ms ごとに更新される PDO のリアルタイム値。右クリックで RxPDO（出力）を変更できます
CiA 402	ドライブのテスト：軸の選択、ステートマシンの Enable／Disable、サーボ設定、Mode of Operation、実際値、各モードの操作パネル

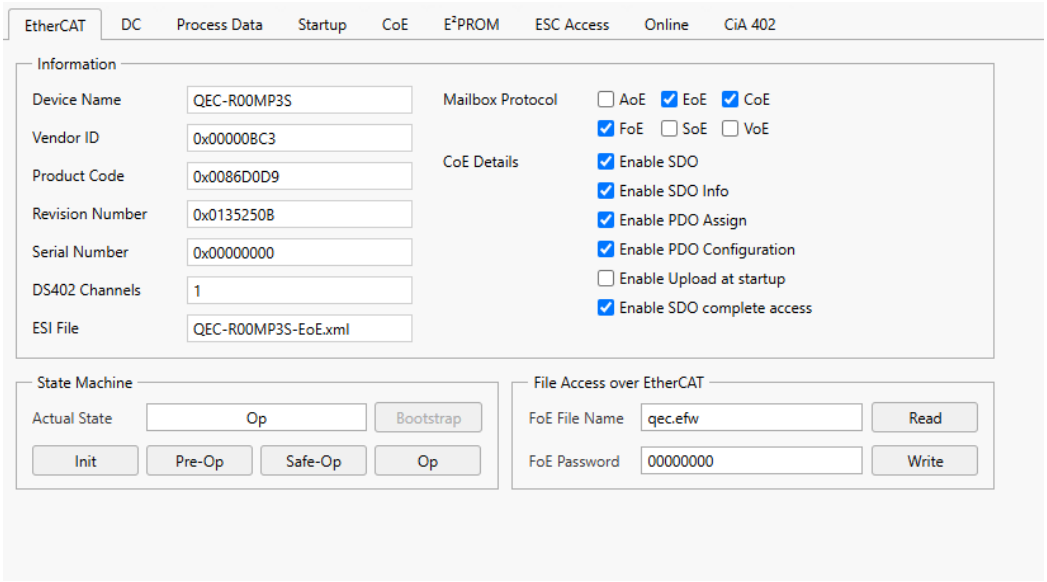


图 11-29　サブデバイスの EtherCAT タブ：情報、ステートマシン、FoE

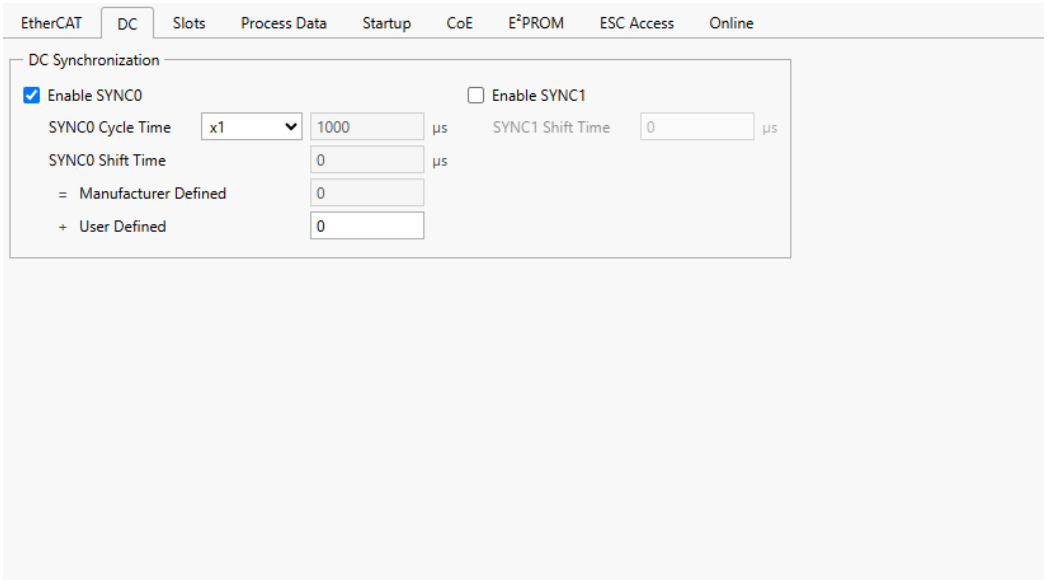


图 11-30　DC タブ：SYNC0／SYNC1 の設定

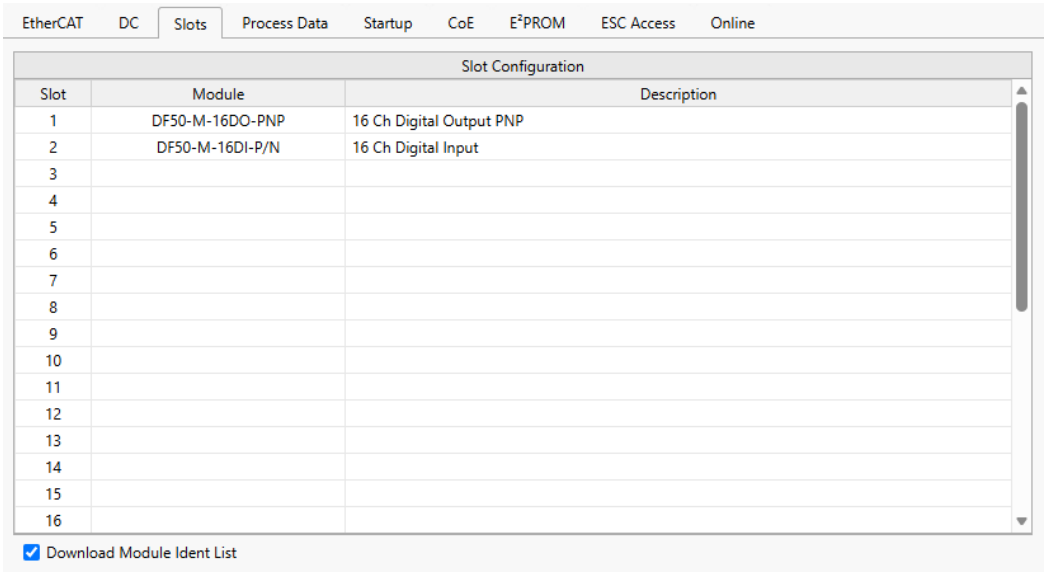


图 11-31　Slots 标签：模块式子设备的 Slot 和模块

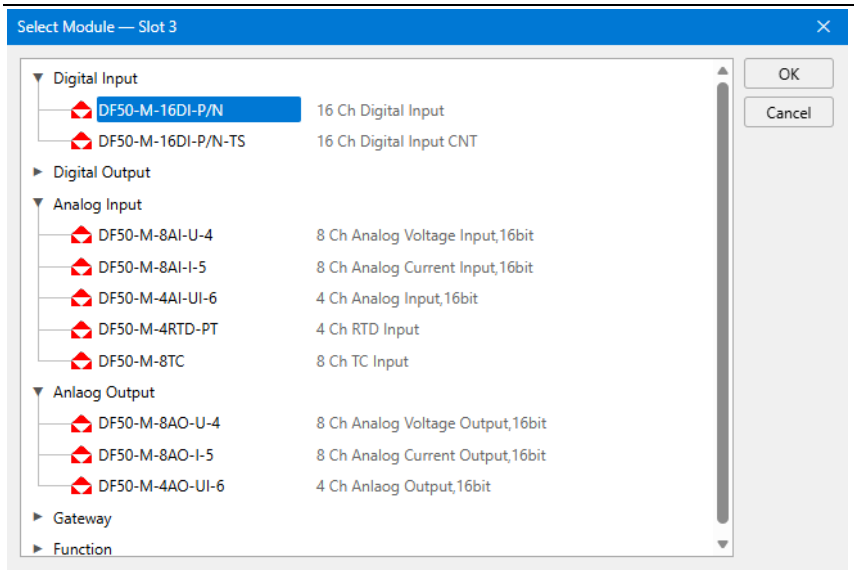


図 11-32　Select Module：Slot に挿入するモジュールの選択

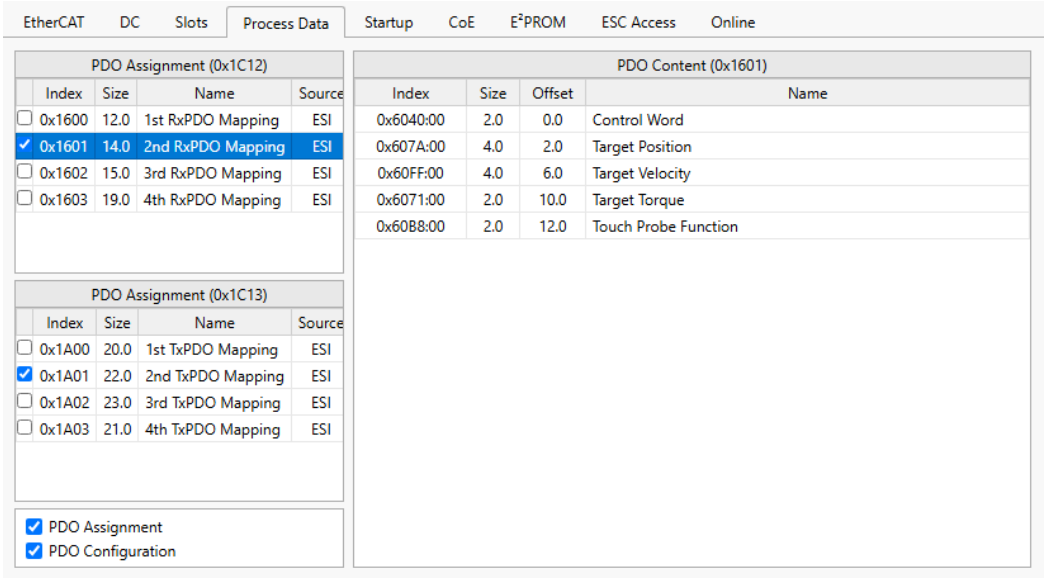


図 11-33　Process Data タブ：左は PDO の割り当て、右は選択した PDO のマッピング内容

Process Data タブの `Source` 列は PDO の出所を示します：`ESI`（読み込んだ ESI または Slots に挿入したモジュール）、`SII`（サブデバイスの EEPROM）、`SDO`（Configure 時に SDO で読み取ったもの）、`User`（ユーザーが追加したもの）。固定または必須の項目は保護されており、削除できません。

*重要：Motion86 の CSP モードを使用する場合（第 12 章）、CiA 402 ドライブの PDO には、出力の 6040h（Controlword）、607Ah（Target position）と、入力の 6041h（Statusword）、6064h（Position actual value）が必要です。多軸ドライブの第 n 軸は、アドレスに 800h×(n−1) を加えます。

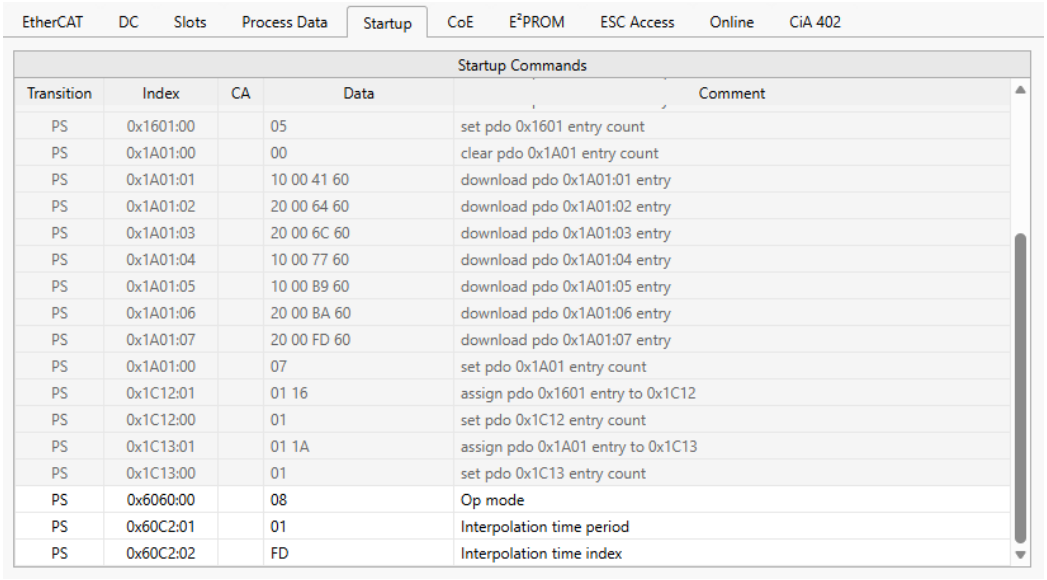


図 11-34　Startup タブ：起動時に書き込む SDO コマンド

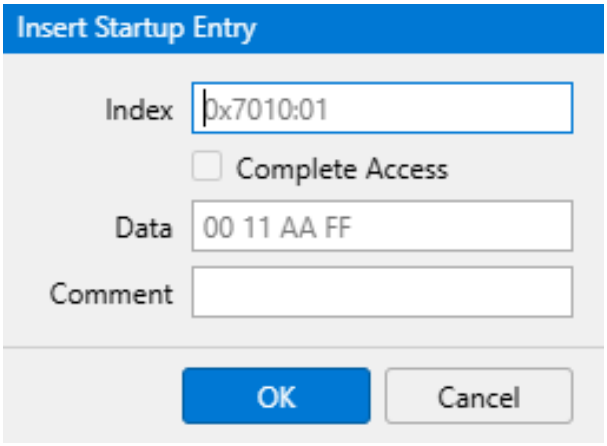


図 11-35　Insert/Edit Startup Entry：Startup コマンドの追加

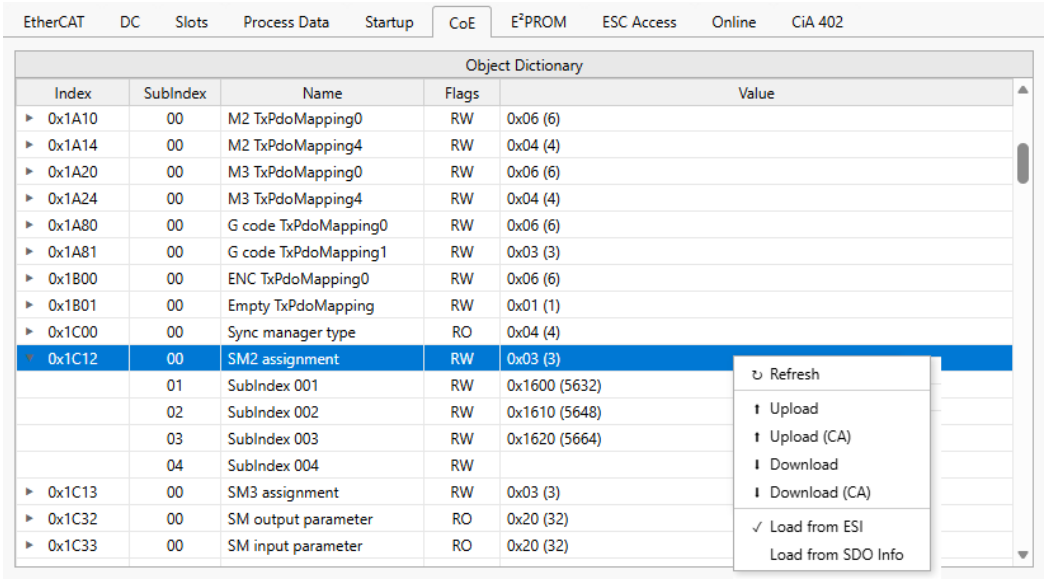


図 11-36　CoE タブ：オブジェクトディクショナリ

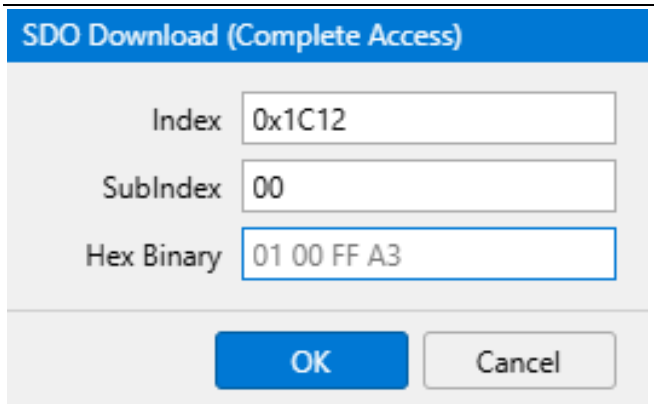


図 11-37　SDO Download (Complete Access)：オブジェクト全体を 16 進数で書き込み

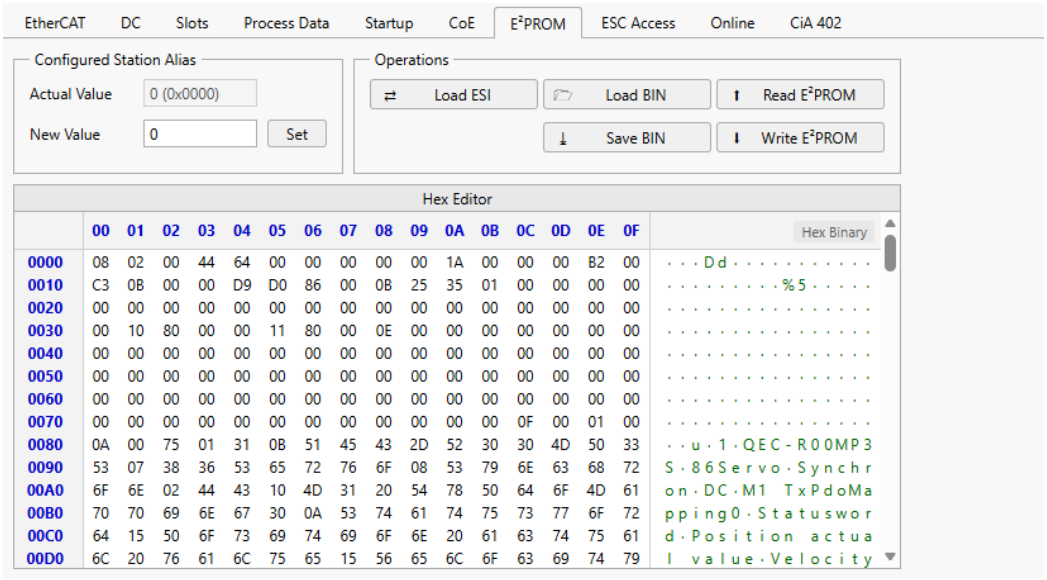


図 11-38　E²PROM タブ：Station Alias、EEPROM 操作、Hex エディター

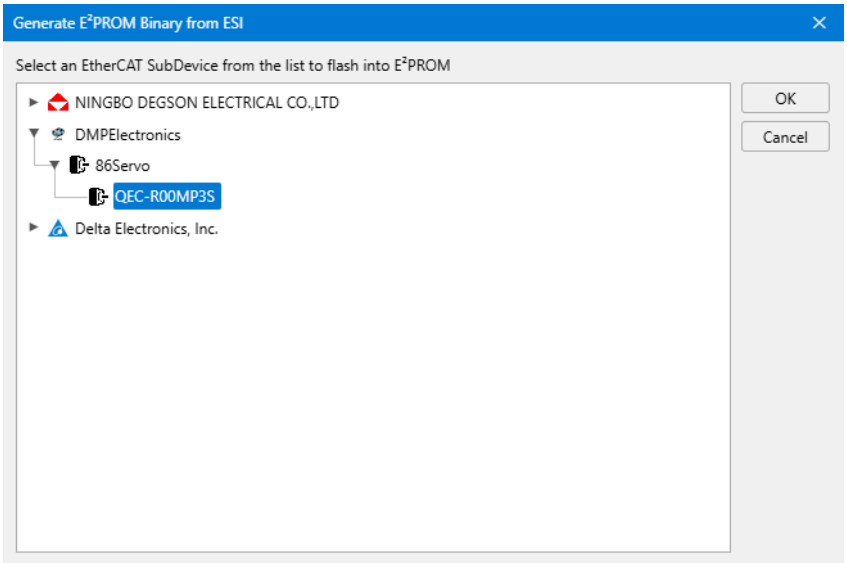


図 11-39　Generate E²PROM Binary from ESI：ESI から EEPROM の内容を生成

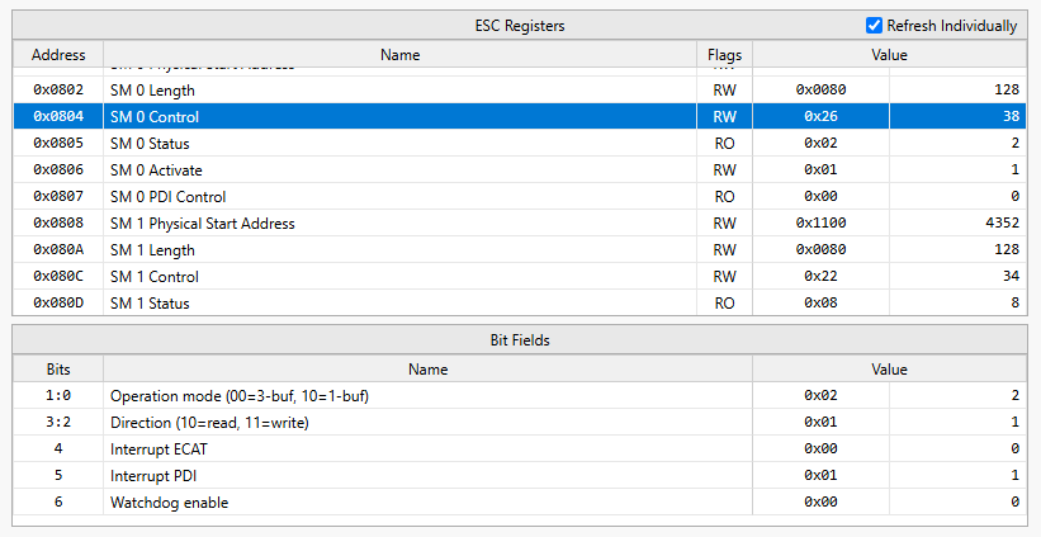


図 11-40　ESC Access タブ：ESC レジスタ

CiA 402 タブ：プログラムを書かずにドライブをテストできます。推奨手順：軸を選択 → Mode of Operation を設定 → **Enable** をクリックしてサーボオン → そのモードのパネルにパラメータを入力して **Run** をクリック → パネルの **Stop** をクリック、または直接 **Disable**。

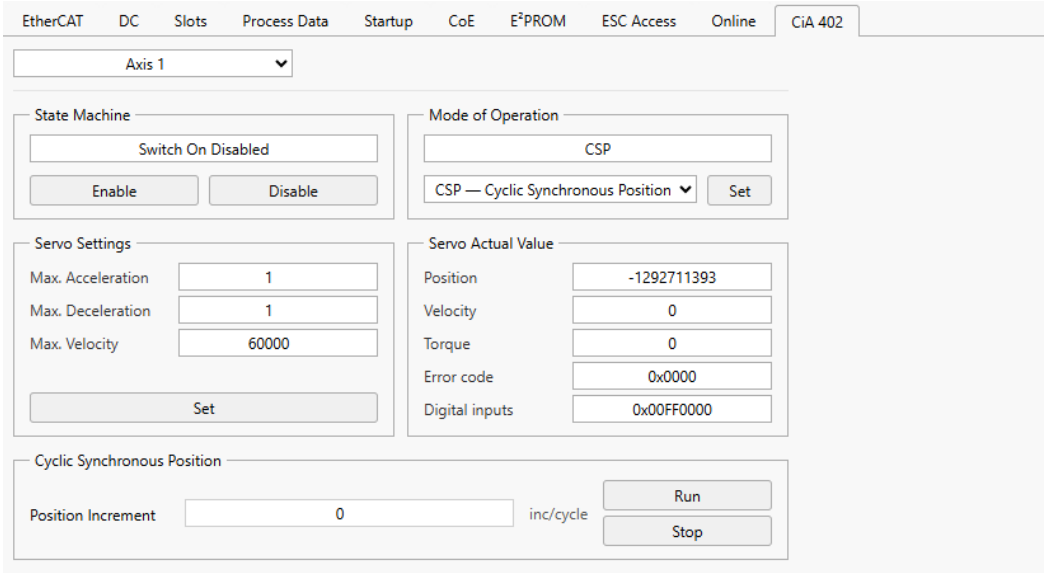


図 11-41　CiA 402 タブ：ステートマシン、サーボ設定、モード、実際値

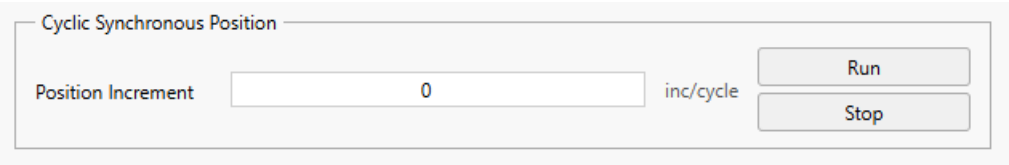


図 11-42　CiA 402 タブの Cyclic Synchronous Position（CSP）パネル

モード	名称
PP	Profile Position
PV	Profile Velocity
TQ	Profile Torque
HM	Homing
CSP	Cyclic Synchronous Position（Motion86 はこのモードを使用）
CSV	Cyclic Synchronous Velocity
CST	Cyclic Synchronous Torque

***安全：**CiA 402 タブはモーターを直接回転させます。テスト前に、機構が自由に動けること、非常停止が有効であることを確認し、まず低速で行ってください。

11.3.7　ログパネル

画面下部のログパネルには Timestamp、Device、Message が表示されます。各操作の成功・失敗と、メインデバイスが Idle でない間に 500 ms ごとに読み出されるシステムイベントです。行をクリックし（Shift で複数選択、Ctrl+A で全選択）、右クリックで Copy または Clear を実行します。

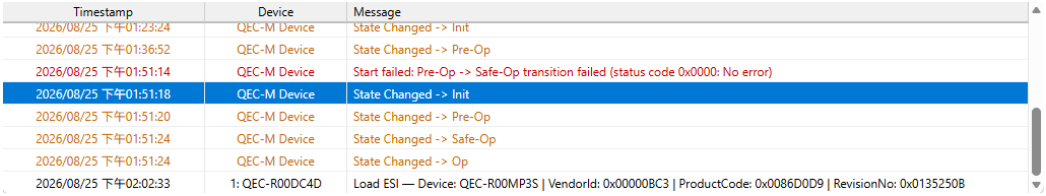


図 11-43　ログパネル

11.3.8　設定手順のまとめ

1. （推奨）ESI Manager で ESI をアップロードし、ESI Library を右クリックして Load ESI を実行します。
2. QEC-M Device の EtherCAT タブで Ethernet Port、Synchronization Mode、Cycle Mode を確認します。
3. **Scan** をクリックし、デバイスツリーにすべてのサブデバイスが表示されるまで待ちます。
4. （任意）既存の ENI を使用する場合は、EtherCAT タブで Configuration File (ENI) にチェックを入れてファイルを選択します。それ以外の場合は、各サブデバイスの Slots／Process Data／Startup で直接設定します。
5. **Configure** をクリックし、各サブデバイスの PDO と CoE が正しく読み込まれたことを確認します。必要に応じて Cycle Time、Global Shift Time、DC を調整します。
6. **Start** をクリックし、Online タブで値を確認します。CiA 402 ドライブは CiA 402 タブでテストできます。
7. Export ENI → **Upload**（例：`QEC-ENI-1.xml`）。
8. 最後に **Stop** → **Deconfigure** の順に実行します。

11.4　ENI の読み込みと変数のマッピング

11.4.1　ENI の読み込み

- **自動**：EtherCAT ページがコントローラー上に新しい、または更新された ENI を検出すると、「The controller has a new ENI: … Take it into the project (its SubDevices and process data)?」と表示されます。**Take it into the project** をクリックします。
- **Read the ENI...**：コントローラーから読み込みます（「Tools > EtherCAT > Read the ENI from the controller...」からも実行できます）。
- **Import ENI...**：パソコン上の ENI ファイルを選択します（例：Configurator の Save As... で保存したファイル）。
- コマンドライン：`86ldmicro-cli eni <プロジェクト> <eni.xml> [コントローラー上の名前]`（15.5 節）。

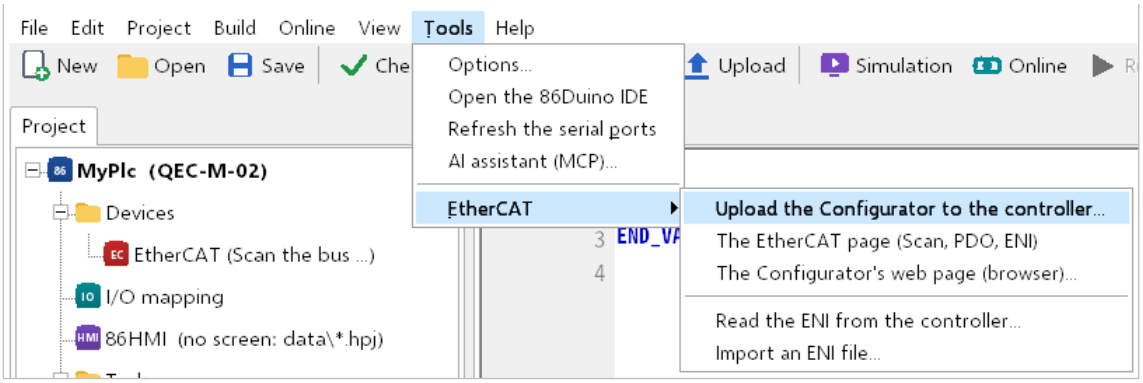


図 11-44　「Tools > EtherCAT」サブメニュー

読み込み後、サブデバイスと各 PDO 項目がプロジェクトツリーの「Devices > EtherCAT」の下に表示され、ENI のコピーが `data\86ld\ethercat\` に保存されます。CiA 402 ドライブはモーターのアイコンで表示され、多軸ドライブ（例：QEC-R11MP3S）は軸ごとに 1 つのノードになります。

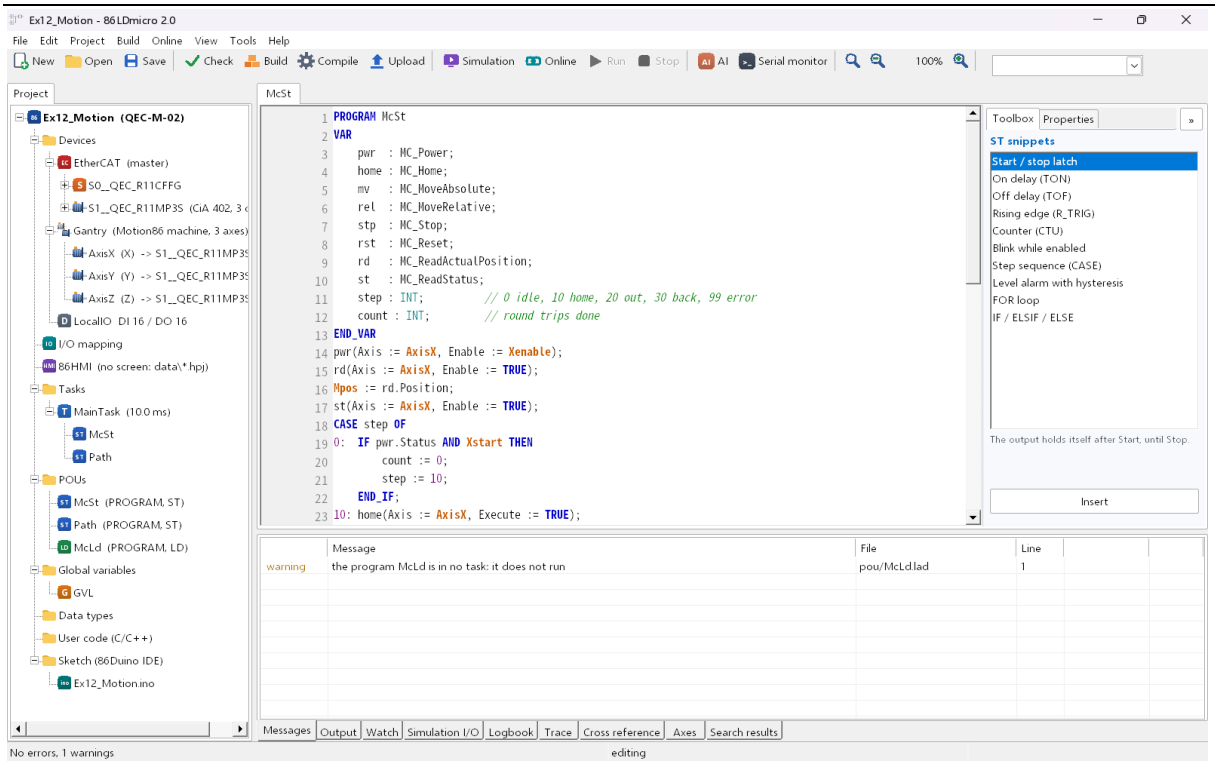


図 11-45　ENI 読み込み後のプロジェクトツリー：S0__QEC_R11CFFG（I/O モジュール）、S1__QEC_R11MP3S（3 軸ドライブ）

- サブデバイス名の形式は `S<位置>__<型式>` です。例：`S0__QEC_R11CFFG`。
- 同じサブデバイスに同名の PDO 項目がある場合（例：多軸ドライブ）、2 番目、3 番目は自動的に「名前 #2」「名前 #3」に改名されます。Index（例：`6864:00`）で指定することもできます。
- PDO に 6040h／6041h を持つサブデバイスは、メーカーを問わず自動的に CiA 402 ドライブとして扱われます。SubDevice ページの **Profile (CiA 402)...** で手動指定することもできます（11.5 節）。
- ENI を再度読み込んでも、作成済みのマッピング、マシン、手動指定した Profile は保持されます。

11.4.2　PDO を変数にマッピングする

プログラムで読み書きするには、PDO 項目を変数にマッピングする必要があります。方法は次の 3 つです。

- EtherCAT ページで **Variable mapping** をクリックし、行を選択して **Link variable...** をクリック（またはダブルクリック）します。**Unlink** で解除します。
- プロジェクトツリーの「Devices > EtherCAT」で、サブデバイスの下にある PDO 項目をダブルクリックします。
- SubDevice ページで行を選択し、**Link to a variable...** をクリックします（11.5 節）。

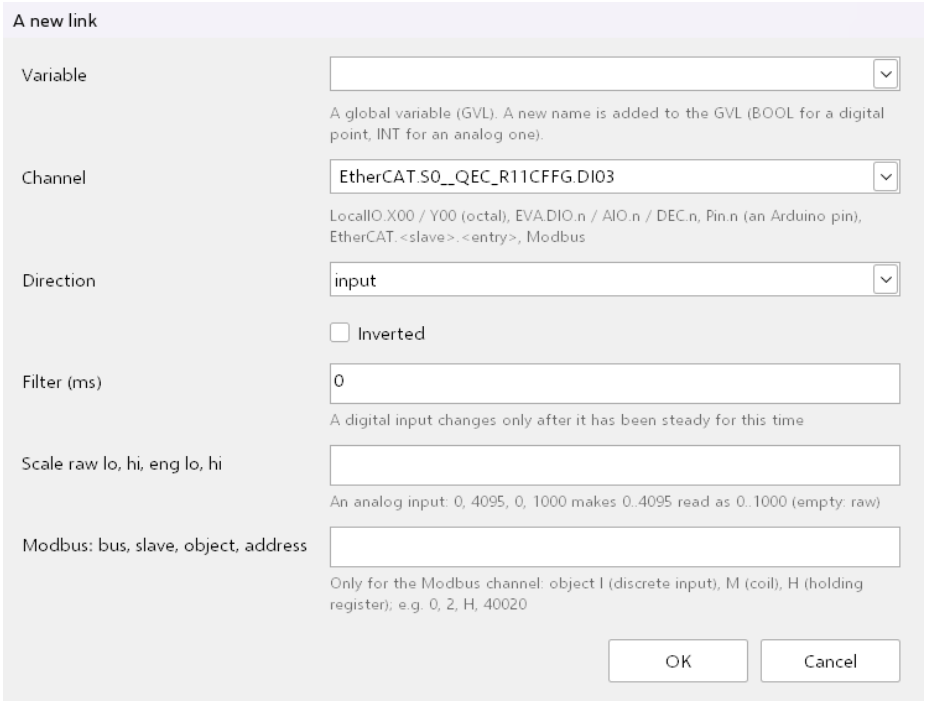
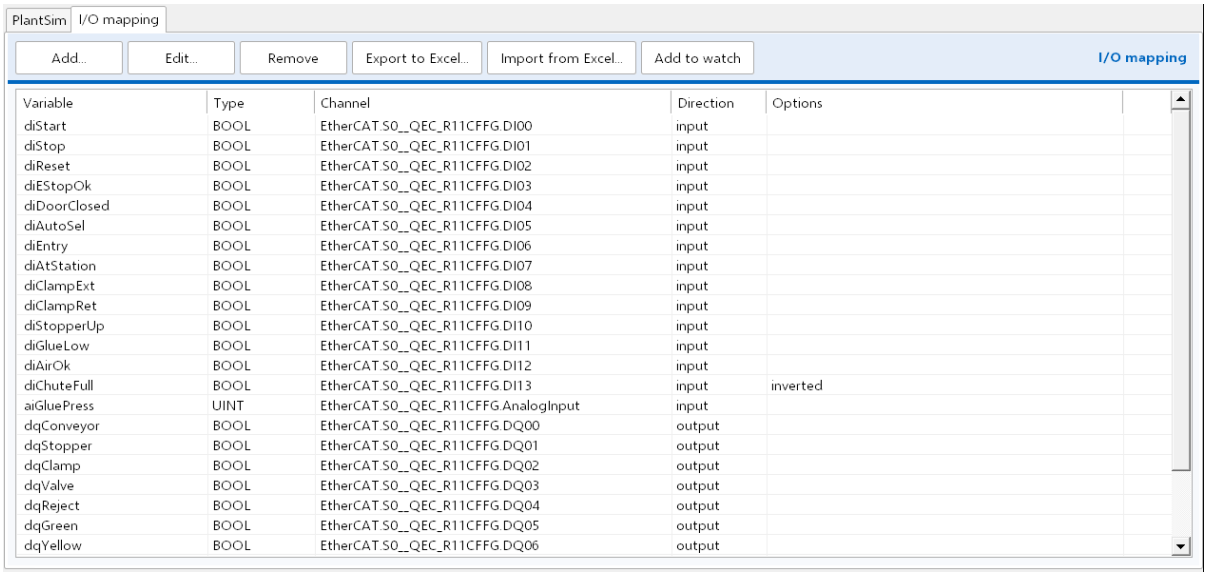


图 11-46　Link variable：チャンネルは入力済み（例：EtherCAT.S0__QEC_R11CFFG.DI02）。変数を入力または選択します

チャンネル名は `EtherCAT.<サブデバイス>.<PDO 項目>` です。新しい変数名を入力すると、変数は自動的に GVL に追加されます（型は PDO に従います）。マッピングが完了すると、「I/O mapping」ページにもこれらのマッピングが表示され（4.6 節）、変数は LD、ST、SFC、C/C++（GVL(名前)）で直接使用できます。



Variable	Type	Channel	Direction	Options
diStart	BOOL	EtherCAT.S0__QEC_R11CFFG.DI00	input	
diStop	BOOL	EtherCAT.S0__QEC_R11CFFG.DI01	input	
diReset	BOOL	EtherCAT.S0__QEC_R11CFFG.DI02	input	
diEStopOk	BOOL	EtherCAT.S0__QEC_R11CFFG.DI03	input	
diDoorClosed	BOOL	EtherCAT.S0__QEC_R11CFFG.DI04	input	
diAutoSel	BOOL	EtherCAT.S0__QEC_R11CFFG.DI05	input	
diEntry	BOOL	EtherCAT.S0__QEC_R11CFFG.DI06	input	
diAtStation	BOOL	EtherCAT.S0__QEC_R11CFFG.DI07	input	
diClampExt	BOOL	EtherCAT.S0__QEC_R11CFFG.DI08	input	
diClampRet	BOOL	EtherCAT.S0__QEC_R11CFFG.DI09	input	
diStopperUp	BOOL	EtherCAT.S0__QEC_R11CFFG.DI10	input	
diGlueLow	BOOL	EtherCAT.S0__QEC_R11CFFG.DI11	input	
diAirOk	BOOL	EtherCAT.S0__QEC_R11CFFG.DI12	input	
diChuteFull	BOOL	EtherCAT.S0__QEC_R11CFFG.DI13	input	inverted
aiGluePress	UINT	EtherCAT.S0__QEC_R11CFFG.AnalogInput	input	
dqConveyor	BOOL	EtherCAT.S0__QEC_R11CFFG.DQ00	output	
dqStopper	BOOL	EtherCAT.S0__QEC_R11CFFG.DQ01	output	
dqClamp	BOOL	EtherCAT.S0__QEC_R11CFFG.DQ02	output	
dqValve	BOOL	EtherCAT.S0__QEC_R11CFFG.DQ03	output	
dqReject	BOOL	EtherCAT.S0__QEC_R11CFFG.DQ04	output	
dqGreen	BOOL	EtherCAT.S0__QEC_R11CFFG.DQ05	output	
dqYellow	BOOL	EtherCAT.S0__QEC_R11CFFG.DQ06	output	

图 11-47　I/O mapping ページ：EtherCAT チャンネルとローカルチャンネルを一緒に表示

*ヒント：プログラムが読み取る入力変数にマッピングがない場合、ビルド時に「Messages」に「… is only read here: it has no I/O mapping and nothing writes it -- it is always FALSE」という警告が表示されます。

11.4.3　EtherCAT デバイスの設定

プロジェクトツリーの「Devices > EtherCAT」を右クリック >「Edit」（またはデバイスページで EtherCAT を選択して「Edit」をクリック）すると、メインデバイスの実行方式を調整できます：

設定	説明
mode	`auto`：CiA 402 サーボがある場合は DC 同期、それ以外は free run（デフォルト）。`sync`：DC 同期（失敗時は free run）。`freerun`
master	`plc`：86LDmicro が生成したプログラムがメインデバイスを起動します（デフォルト、`plc86_ecat.cpp`）。`sketch`：ユーザーの .ino で begin、attach、start を行い、完了後に `plc86_ecOk = true` を設定します
cycleUs	EtherCAT の周期（μs）、デフォルトは 1000

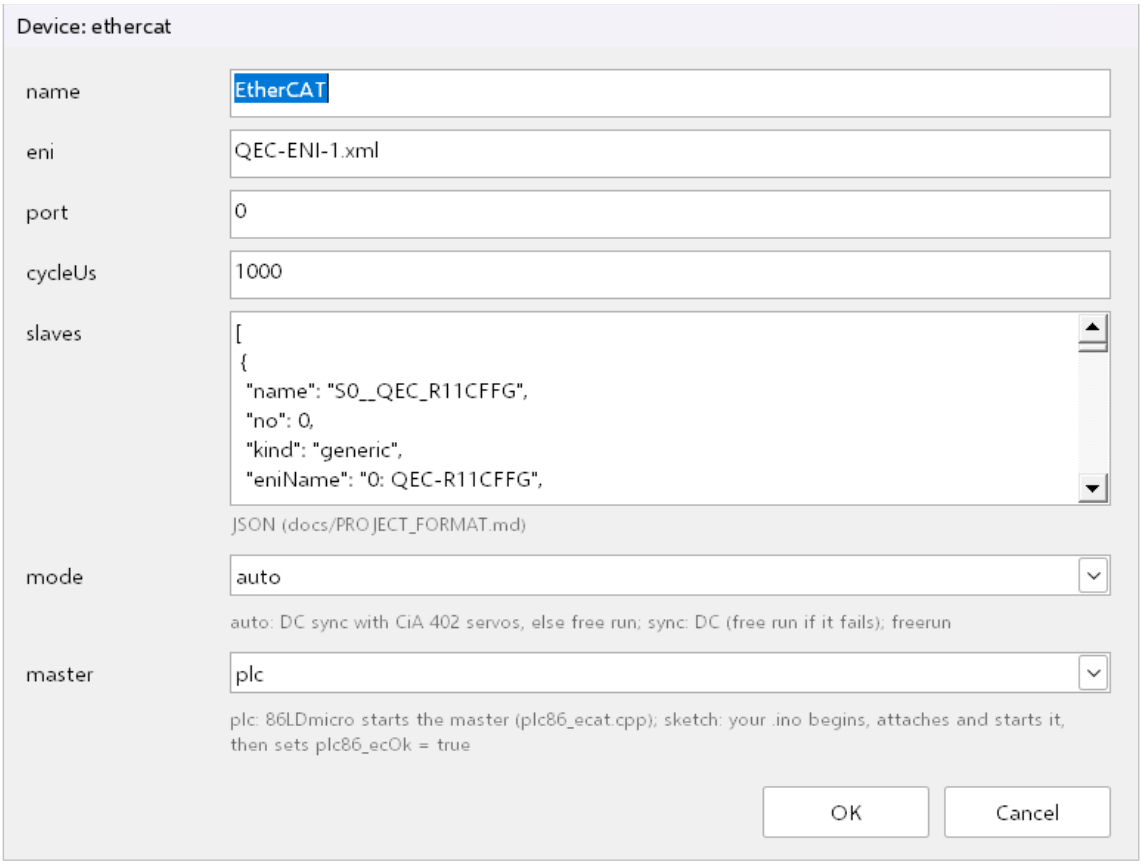


図 11-48　EtherCAT デバイスの設定

11.4.4　ビルド、アップロードとメインデバイスの確認

1. **Upload**（F8）をクリックします。PLC プログラムが Configurator と置き換わり、起動時にメインデバイスが `C:/eni/<ファイル名>` で起動し、各サブデバイスが attach され、ドライブのモードが設定されてから周期交換が始まります。
2. 「Serial monitor」を開きます（オンライン不要、10.3 節）。メインデバイスの状態が 5 秒ごとに表示されます（正常動作時は起動後に 3 回のみ、失敗時は表示し続けます）。例：`EtherCAT: running, 2 slaves, 1000 us, sync, ENI C:/eni/QEC-ENI-1.xml`、または失敗の原因。
3. オンライン時は、「Logbook」にもメインデバイスの状態が記録されます（10.5 節）。

11.5 SubDevice ページ

プロジェクトツリーの「Devices > EtherCAT」の下にあるサブデバイスをダブルクリックすると、その SubDevice ページが開きます。上部にはバス上の位置、ENI での名前、ベンダーコード／製品コード、Profile、コード内のオブジェクト名が表示され、下部は「**Process data (PDO)**」と「**CoE object dictionary (online)**」の 2 つのタブに分かれています。

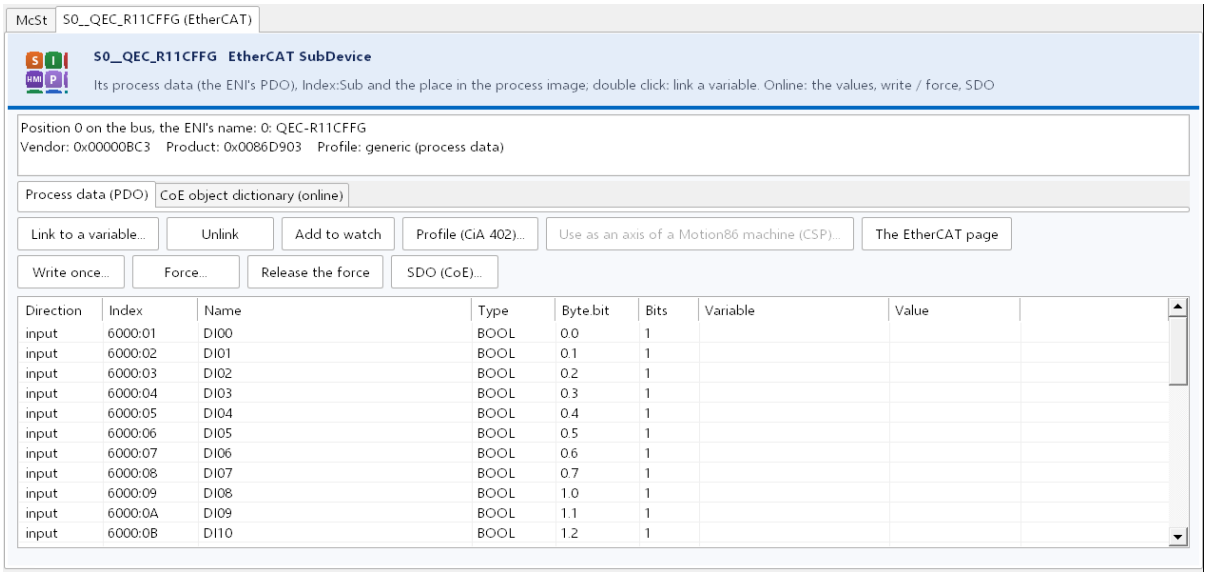


図 11-49　QEC-R11CFFG の SubDevice ページ：各 PDO 項目の方向、Index、名前、型、Byte.bit、ビット数、変数

ボタン	説明
Link to a variable...／Unlink	選択した PDO 項目を変数にマッピング、または解除します
Add to watch	マッピングされた変数を Watch ウィンドウに追加します
Profile (CiA 402)...	Profile（Auto／CiA 402／Generic）とドライブ数を手動で指定します（サードパーティ製ドライブ用）
Use as an axis of a Motion86 machine (CSP)...	選択したドライブを Motion86 マシンに追加します（12.2 節）
The EtherCAT page	EtherCAT ページに戻ります
Write once...／Force...／Unforce	オンライン：PDO 出力を直接書き込み、または強制します（11.6 節）
SDO (CoE)...	オンライン：単一のオブジェクトを読み書きします（11.6 節）

CiA 402 ドライブの SubDevice ページ下部には、各ドライブ（#1、#2、#3…）に対応する Motion86 軸も表示されます：

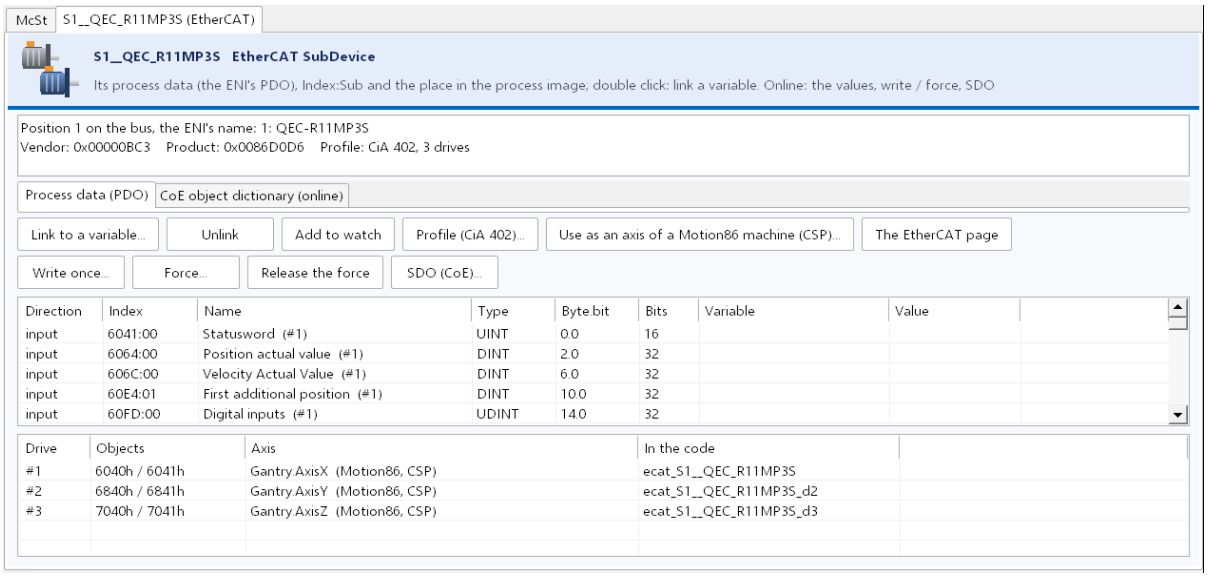


図 11-50　QEC-R11MP3S（3 軸ドライブ）：下部に各ドライブに対応する Motion86 軸を表示

サードパーティ製ドライブの PDO に標準の 6040h／6041h がない場合は、**Profile (CiA 402)...** をクリックして手動で指定します：

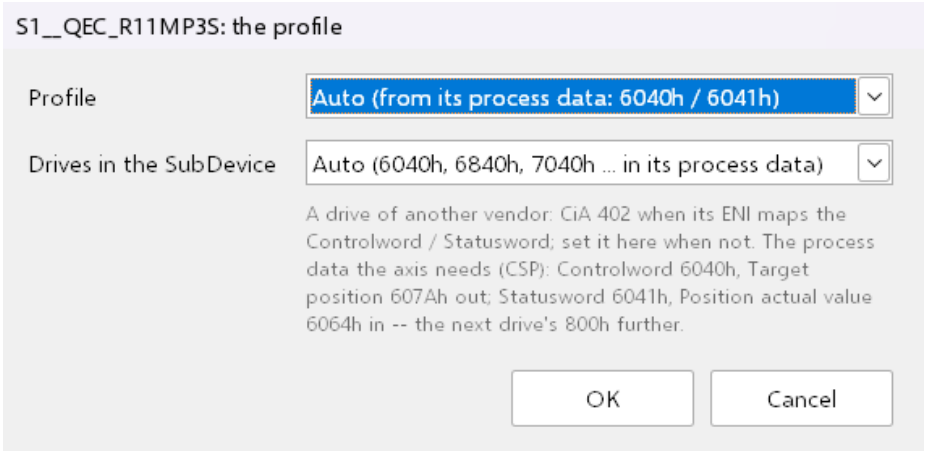


図 11-51　Profile の設定：Auto、CiA 402、Generic と「Drives in the SubDevice」

SubDevice ページはプログラムと左右に並べて表示できるので（「View > Side by side: the tab in front to the right / end it」、Ctrl+\、3.7 節）、PDO の値とプログラムを同時に確認できます：

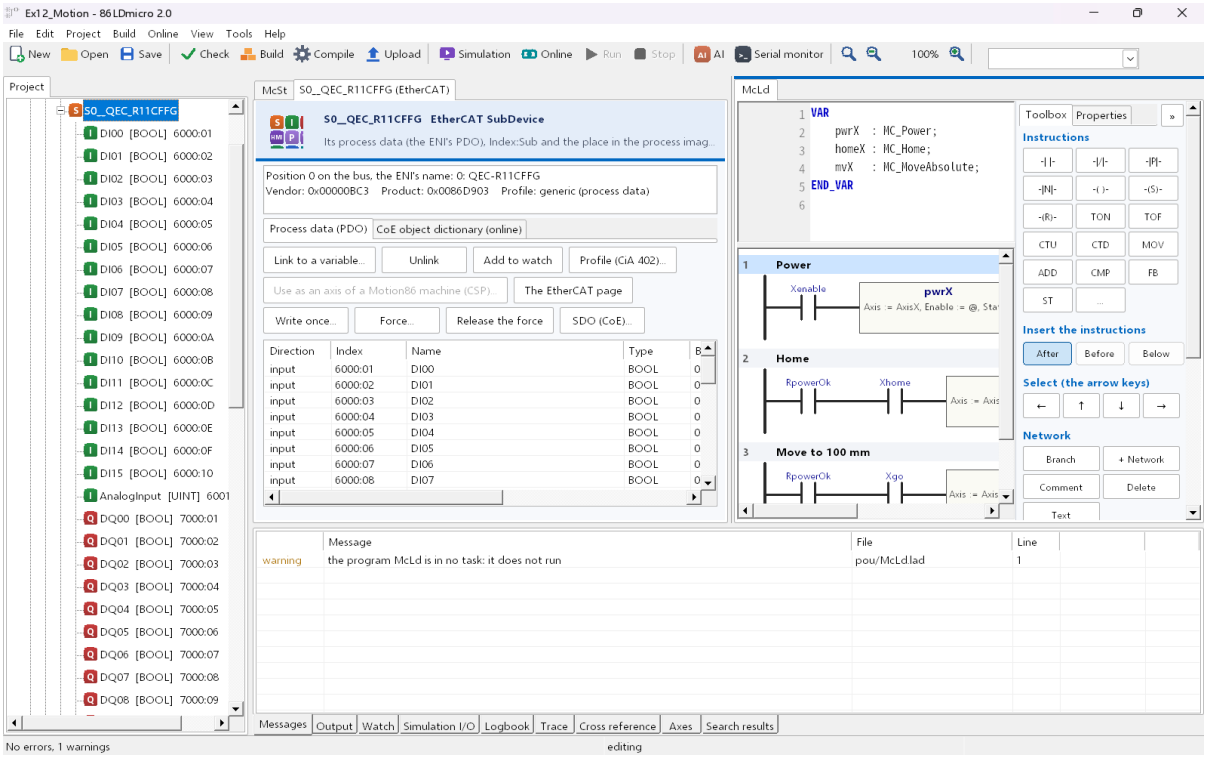


図 11-52　左右に並べて表示：左側に SubDevice ページ、右側にラダー図

11.6　PDO のオンライン値、SDO と CoE

11.6.1　PDO のオンライン値

オンライン接続後（Ctrl+F5）、SubDevice ページのすべての PDO 項目（変数へのマッピングの有無を問わず）に現在値が表示されます。コントローラーは 100 ms ごとに更新し、ページは 0.5 秒ごとに再表示されます。REAL は浮動小数点値を表示し、WORD／DWORD は 16 進数も併記されます。

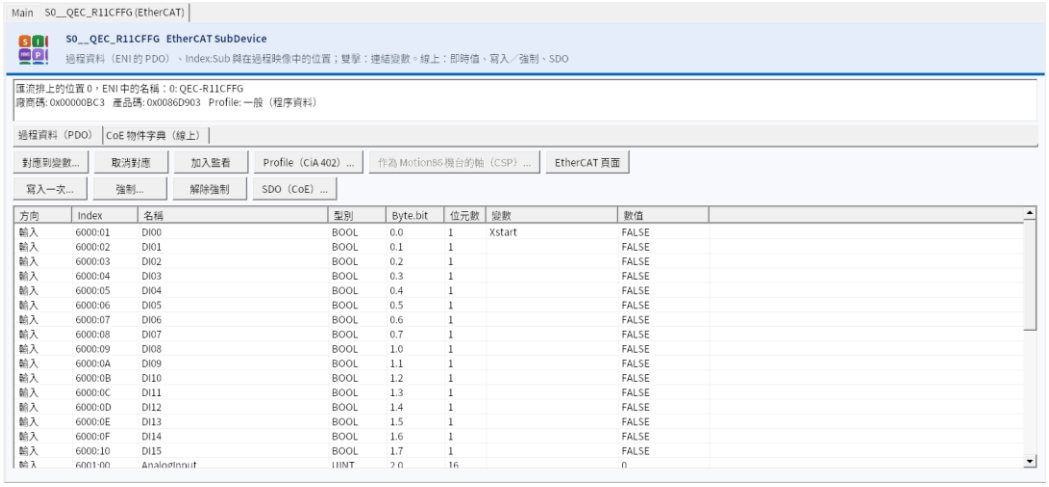


図 11-53　オンライン：各 PDO 項目にリアルタイム値を表示

11.6.2　1 回書き込みと強制

機能	説明
Write once...	1 回だけ書き込み、以後は書き込みません。プログラムからは引き続き変更できます
Force...	「Unforce」を実行するか、接続が 3 秒間切断されるまで、スキャンごとに書き込みます。強制中は [F] が表示されます
Unforce	選択した項目の強制を解除します

- 変数にマッピングされた項目では、書き込み・強制の対象は**その変数**です（シミュレーション中も可能）。変数にマッピングされていない**出力**は、プロセスデータに直接書き込まれます（オンライン時のみ）。
- **手動で書き込めないもの**：入力（サブデバイスが書き込みます。プログラムをテストするには、変数をマッピングしてその変数を強制してください）、および CiA 402 ドライブの運動オブジェクト（6040h、607Ah、60FFh、6060h など。軸が制御します）。

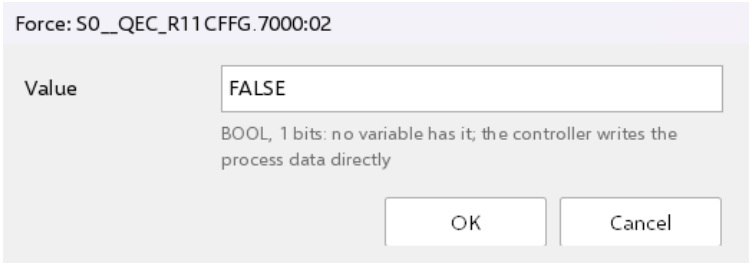


図 11-54　変数にマッピングされていない出力（DQ01）の強制：コントローラーがプロセスデータに直接書き込み

*安全：出力を強制すると、設備がプログラムの制御を受けなくなります。現場の安全を確認してから操作してください。

11.6.3　SDO（CoE）：単一オブジェクトの読み書き

PDO 項目を選択して **SDO (CoE)...** をクリックし（その Index が入力されます）、動作（読み取りまたは書き込み）、Index:Sub（16 進数、例：`1018:01`）、サイズ（1／2／4／8 バイト）、値（書き込み用：10 進数、または 0x で始まる 16 進数）を指定します。失敗時は abort code とその説明が表示されます。

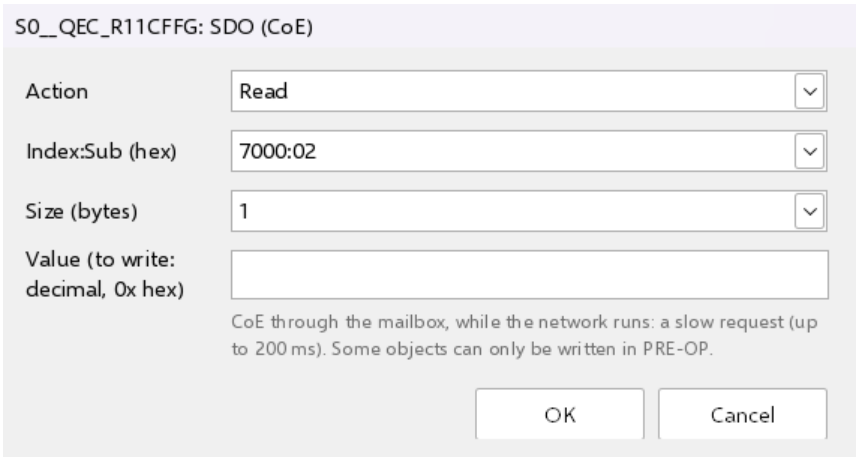


図 11-55　SDO（CoE）：単一オブジェクトの読み取りまたは書き込み

11.6.4　CoE オブジェクトディクショナリ（オンライン）

TwinCAT の CoE - Online と同様に、ドライブのすべてのパラメータを参照・変更できます：

1. オンライン接続後、SubDevice ページの **CoE object dictionary (online)** タブに切り替え、**Read the object dictionary** をクリックします。SDO Info によりサブデバイスから完全な一覧（名前、型、アクセス権）を取得してプロジェクトに保存し、次回開いたときはそのまま表示されます。
2. **Read all values** をクリックします。または **Auto refresh** にチェックを入れて間隔（0.5／1／2／5 秒）を選択すると、画面に表示されている行が継続的に更新され、値が変化すると背景が一時的に黄色になります。
3. 上部の **Filter** で Index または名前により絞り込みます（例：`6098`、`Homing`）。
4. RW 項目をダブルクリック（または **Write...** をクリック）してパラメータを変更します。書き込み後すぐに読み返されます。

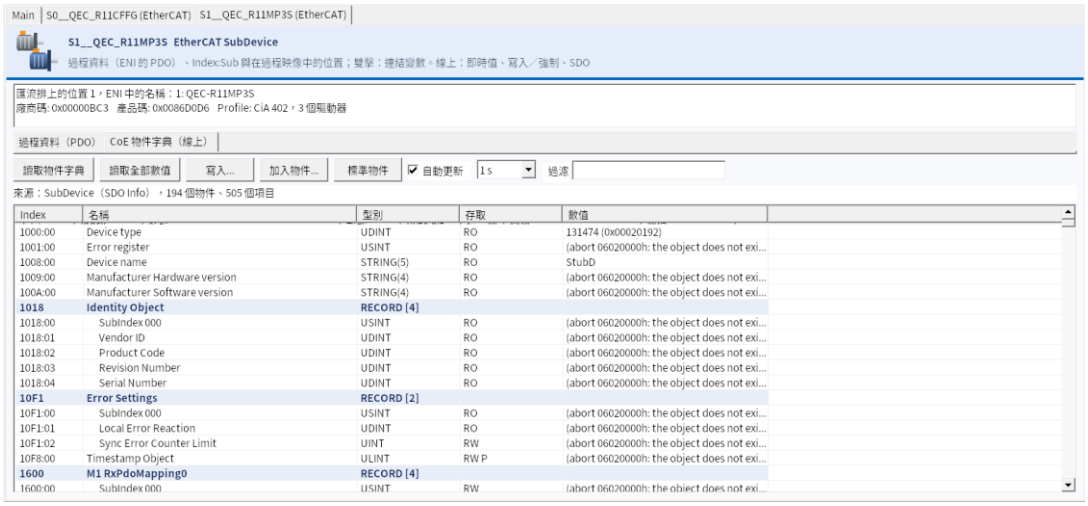


図 11-56　CoE オブジェクトディクショナリ（オンライン）：QEC-R11MP3S のオブジェクト、型、アクセス、値

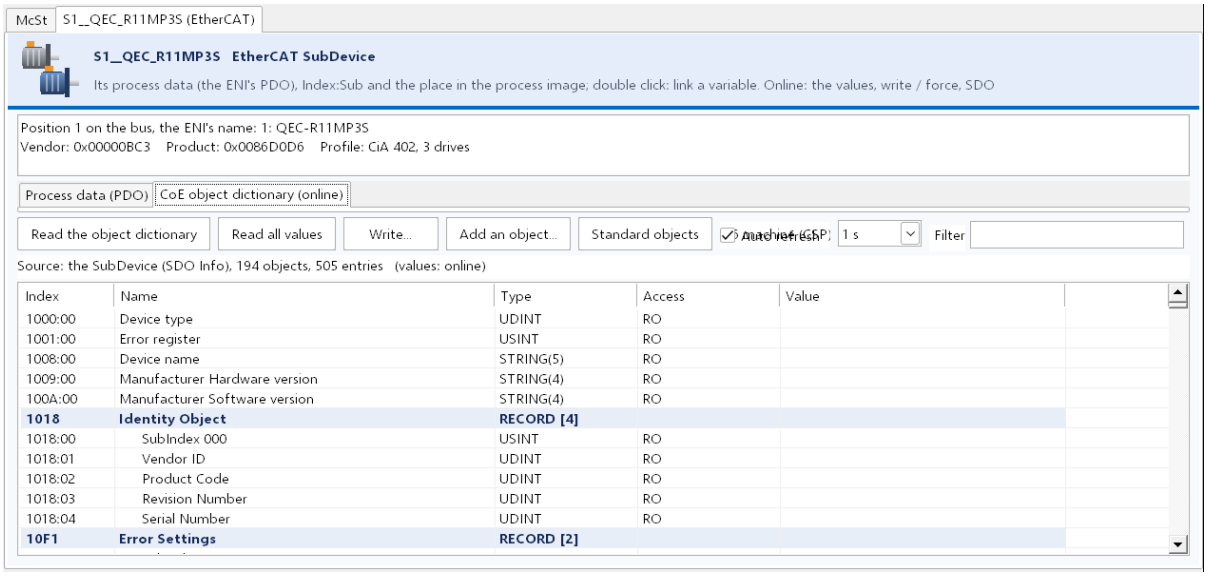


図 11-57　オフライン時：プロジェクトに保存されたオブジェクト一覧を表示（値の表示には接続が必要）

- アクセス列：RO は読み取り専用、RW は読み書き可能。「(PRE-OP)」は PRE-OP 状態でのみ書き込み可能、「P」は PDO にマッピング可能であることを示します。
- サブデバイスが SDO Info に対応していない場合は、代わりに**標準オブジェクト**（1000h、1018h、CiA 402 の主要オブジェクト）と PDO 項目が表示されます。**Add an object...** でその他の Index を手動で追加できます。
- 読み書きはコントローラーのスキャンの合間にノンブロッキングで行われるため、**PLC のスキャンが止まることはありません**。
- 主な abort code：06020000h オブジェクトが存在しない、06010002h 読み取り専用、06090030h 値が範囲外、08000022h 現在の状態では不可（PRE-OP でのみ書き込み可能）。
- コマンドライン：`86ldmicro-cli online <プロジェクト> <ポート> sdo …`、`… coe <SubDevice>`（15.5 節）。

11.7 トラブルシューティング

EtherCAT が動作しない場合は、まず「Serial monitor」を開いて（オンライン不要）メインデバイスの状態を確認し、次の表に従って対処してください：

状況	原因と対処
EtherCAT ページに Configurator の応答がないと表示される	Configurator がまだアップロードされていないか、PLC プログラムに置き換えられています。**Upload Configurator** をクリックしてください。USB の場合はツールバーで選択したポートがコントローラーのポートであることを、ネットワークの場合はパソコンがコントローラーの LAN ポートに接続されていることと IP を確認してください
`EtherCAT: begin failed`	EtherCAT ケーブルがメインデバイスの EtherCAT ポートに接続されていない、またはサブデバイスに電源が入っていない。.ino がまだ 1.x の `EVA.begin()` を呼び出している（11.1 節）
`the ENI … cannot be used`	コントローラーにプロジェクトの ENI（`C:/eni/…`）がない、または実際のネットワークと一致しないため、メインデバイスがスキャンしたネットワークを使用している。再度 Scan し、ENI をエクスポートしてアップロードし、プロジェクトに読み戻してください
`no slave N`	サブデバイスの数または順序が ENI と一致しない、またはケーブルがサブデバイスの IN ではなく OUT に接続されている。配線を確認し、再度 Scan して ENI をエクスポートし、読み戻してください
入力が常に FALSE	変数がマッピングされていない（ビルド時に「Messages」で警告されます）。または、その PDO が ENI の割り当てに含まれていない（Process Data タブでチェックを入れて再エクスポートしてください）
EtherCAT がまったく動作しない	1.x の `EVA.begin()` と `myeva.*` を削除します。EtherCAT デバイスの master が `plc` であることを確認します（11.4.3）
CSP ドライブが動かない	PDO に 6040h／607Ah／6041h／6064h が必要です（11.3.6）。CoE オブジェクトディクショナリで Statusword とエラーコード 603Fh を確認します。マシンと軸の設定は第 12 章を参照してください
Configurator をアップロードしたら PLC プログラムが消えた	正常な動作です。Configurator は一時的に PLC プログラムと置き換わります。設定完了後に **Upload**（F8）をクリックしてください
アップロード中やオンライン時に Configurator が切断される	USB モードではアップロードとオンラインモニターが優先され、Configurator は自動的に一時停止します。完了後に EtherCAT ページに戻ると再開します

第 12 章　Motion86 モーション制御

86LDmicro は TwinCAT 方式の「マシン＋軸」で EtherCAT サーボドライブを管理します。本章では、Motion86 のマシンと軸の作成と設定、プログラムを書かずに行う試運転、PLCopen MC ファンクションブロックと M86 マシンファンクションブロックによるモーションプログラムの作成、および安全機構について説明します。始める前に第 11 章を完了し、CiA 402 ドライブがプロジェクトツリーの「Devices > EtherCAT」に表示されるようにしてください。

12.1　概念

Motion86 マシンは Motion86 ライブラリの軸グループで、最大 **9 軸**（X、Y、Z、U、V、W、A、B、C）を持てます。各**軸**は EtherCAT の CiA 402 ドライブ 1 台で、**CSP（サイクリック同期位置）**モードで制御されます。Motion86 は EtherCAT の周期ごとに各軸の目標位置を計算し、ドライブに送ります。サンプル `tools\example\ex30_dispense` のマシン **Gantry** は、QEC-R11MP3S の 3 軸（AxisX、AxisY、AxisZ）を使用します。

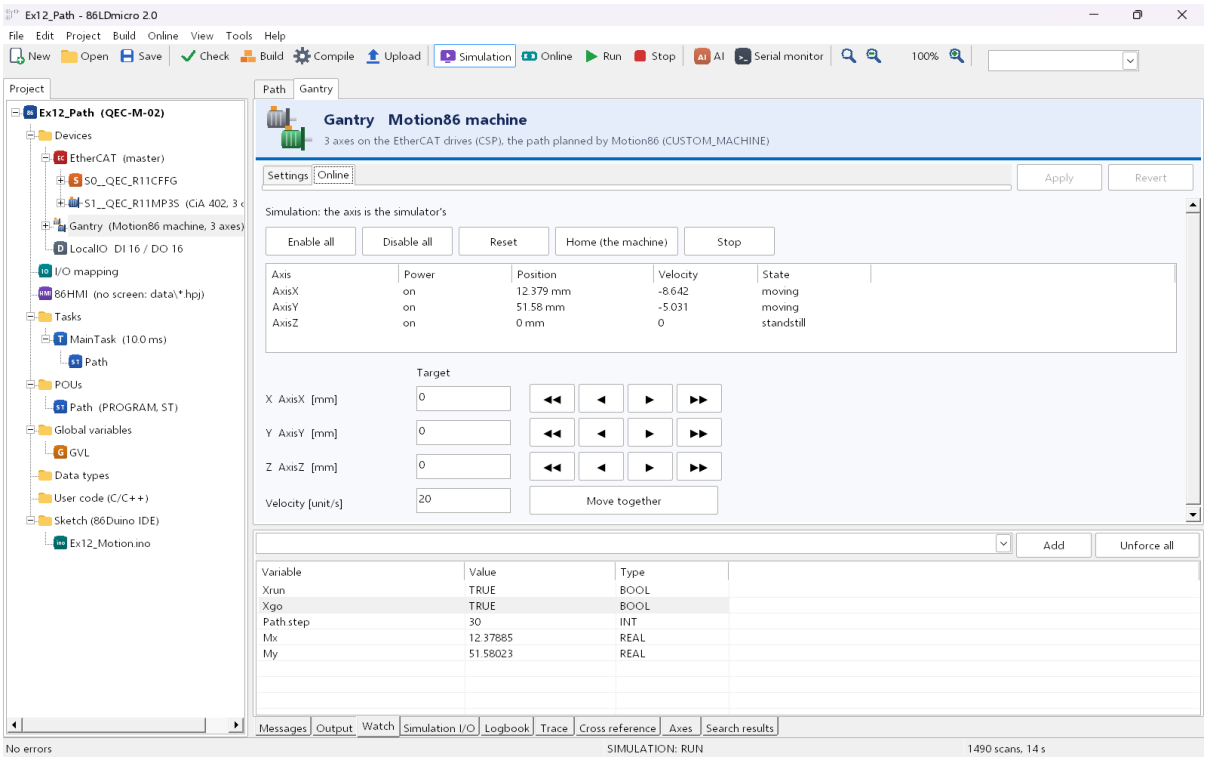


図 12-1　シミュレーションで移動する Gantry マシンの 3 軸：プロジェクトツリーのマシンと軸、マシンページの「Online」

用語	説明
マシン	一緒に計画される軸のグループ。プログラムでは `Machine := Gantry` で指定します
軸	ドライブ 1 台（または仮想軸）。軸名はプログラム内の定数です（例：`Axis := AxisX`）
軸文字	マシンの第 1〜9 軸は順に X、Y、Z、U、V、W、A、B、C です。G-code と M86_* ファンクションブロックはこれらの文字で軸を指定します
ユーザー単位	プログラム内の位置は mm、deg などの単位、速度は unit/s で表します。軸の PPU によってドライブのカウントに換算されます
PPU	単位あたりのパルス数 = 1 回転あたりのパルス数 ÷ 1 回転あたりの移動量。例：6400 counts/rev、ボールねじのリード 10 mm → 640 counts/mm
補間（Interpolated）	マシン全体で 1 本の経路（直線、円弧、G-code）。全軸が同時に到達します
独立（Independent）	軸ごとに個別に計画され、各軸が同時にそれぞれの移動を実行できます
CSP	Cyclic Synchronous Position：メインデバイスが周期ごとに目標位置（607Ah）を送り、ドライブが実際位置（6064h）を返します

12.2 マシンの作成

次の 3 つの方法のいずれでも「A new Motion86 machine」ウィンドウが開きます：

- プロジェクトツリーの「Devices」を右クリック >「Add a device」> `motion86`。
- プロジェクトツリーの CiA 402 ドライブ（モーターのアイコン）を右クリック >「Use as an axis of a Motion86 machine」。
- ドライブの SubDevice ページで **Use as an axis of a Motion86 machine (CSP)...** をクリックします（11.5 節）。

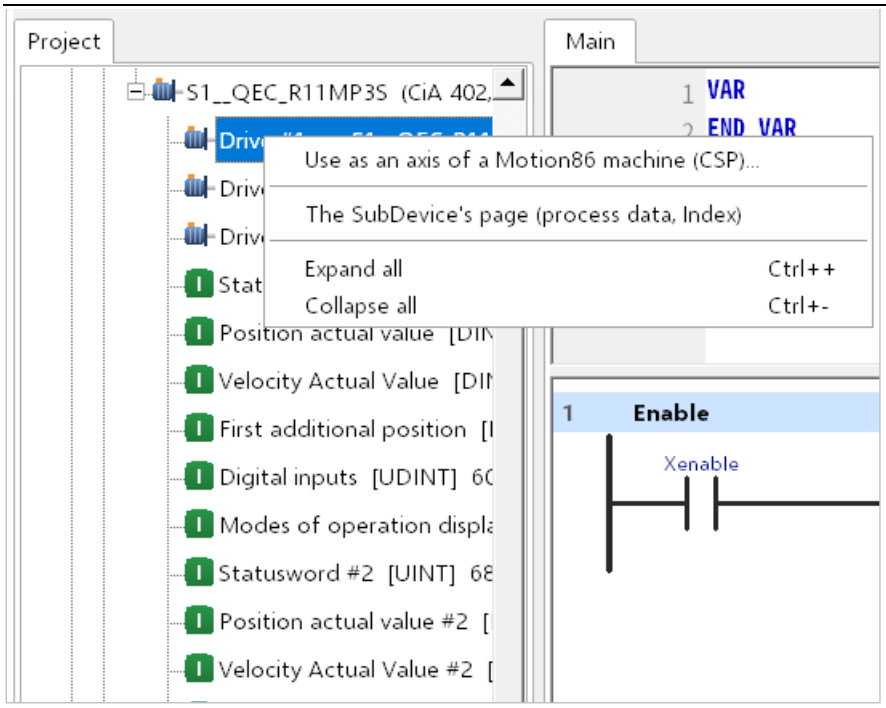


図 12-2　ドライブを右クリック：Use as an axis of a Motion86 machine

1. マシン名を入力します（例：`Gantry`。英字、数字、_ を使用でき、他のデバイスや軸と同じ名前は使えません）。
2. 軸数（1〜9）を選択します。使用可能な CiA 402 ドライブが順に X、Y、Z、U、V、W、A、B、C 軸になります。使用可能なドライブがない場合はまず**仮想軸**として作成し、後で軸ページの「Drive」でドライブを選択します。
3. **OK** をクリックします。プロジェクトツリーにマシンと各軸（モーターのアイコン）が表示され、軸の後ろに軸文字と対応するドライブが表示されます（例：`AxisX (X) -> S1__QEC_R11MP3S #1`）。

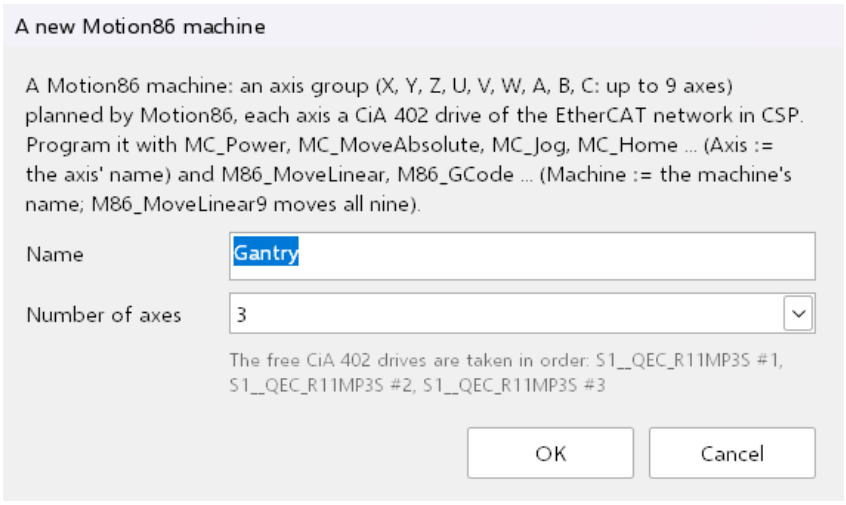


図 12-3　A new Motion86 machine：名前と軸数（1〜9）

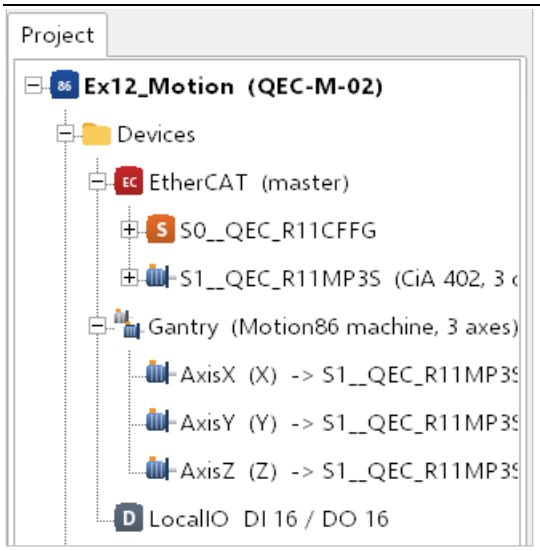


図 12-4　プロジェクトツリー：マシン Gantry と 3 つの軸。軸の後ろに対応するドライブを表示

*ヒント：軸名とマシン名はプログラム内の定数（DINT）です（例：`Axis := AxisX`、`Machine := Gantry`）。名前を変更したらプログラムも修正し、**Check**（Shift+F7）をクリックして修正漏れを見つけてください。

12.3　マシンの設定

プロジェクトツリーのマシンをダブルクリックすると、マシンページの「**Settings**」タブが開きます。変更後は **Apply** をクリックします（プロジェクトに保存されます。コントローラーに反映されるのはビルドしてアップロードした後です）。**Revert** で変更を破棄します。

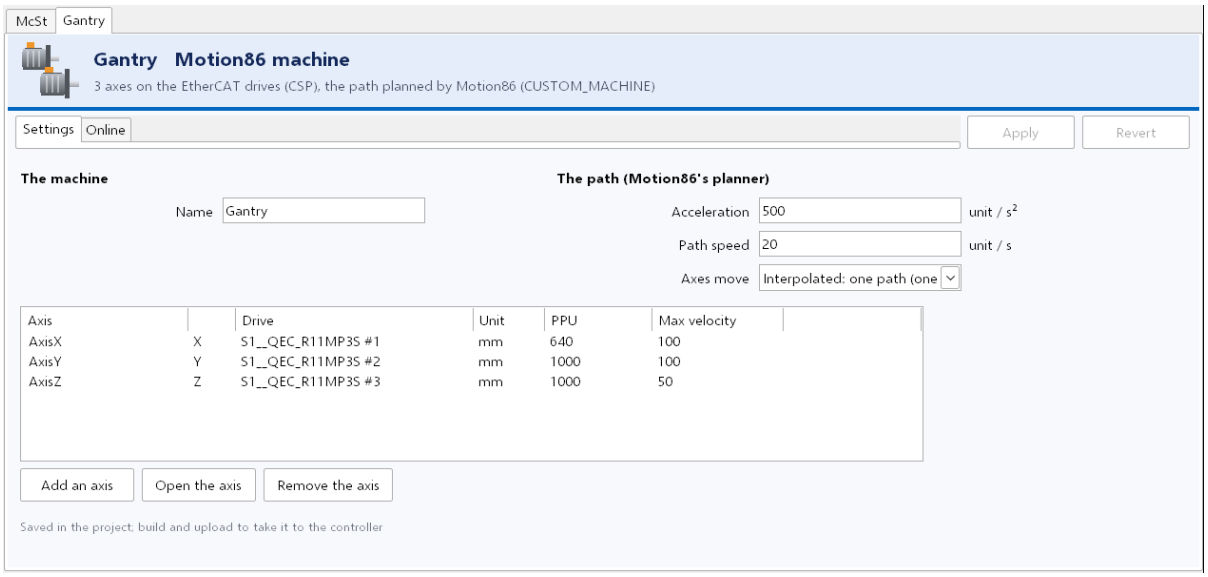


図 12-5　マシンページの「Settings」：名前、経路加速度、経路速度、軸の動作方式、軸リスト

項目	説明
Name	マシン名。プログラムでは `Machine := Gantry`
Acceleration（unit/s²）	Motion86 プランナーの経路加減速度。MC_* の Acceleration／Deceleration はこの値で決まります
Path speed（unit/s）	M86_MoveLinear の Feedrate が 0 のときに使用されます
Axes move	**Interpolated**：単一経路（Motion86 マシン 1 台）。同じマシンの移動は順に待ち行列に入ります。**Independent**：各軸が個別に動作し（軸ごとに Motion86 マシン 1 台）、同時に移動できます
軸リスト	軸、文字、ドライブ、単位、PPU、最大速度。**Add an axis**、**Open the axis**、**Remove the axis** で操作します（ドライブは EtherCAT ネットワークに残ります）

***注意：**EtherCAT ドライブ上の Motion86 マシンは最大 **3 台**です。独立モードでは軸ごとに 1 台を使用するため、プロジェクト全体で独立軸は最大 3 軸（または補間マシン 1 台＋独立軸 2 軸）です。3 軸を超えるマシンには **Interpolated** モードを使用してください（1 台のマシンで 9 軸まで扱えます）。上限を超えると「Apply」とビルドでエラーが表示されます。独立モードでは M86_MoveArcXY／M86_GCode は実行されません。

12.3.1　多軸マシン（最大 9 軸）

1 台のマシンには 1〜9 軸を設定でき、順に X、Y、Z、U、V、W、A、B、C に対応します。プロジェクトツリーと軸リストには各軸の文字が表示されます。

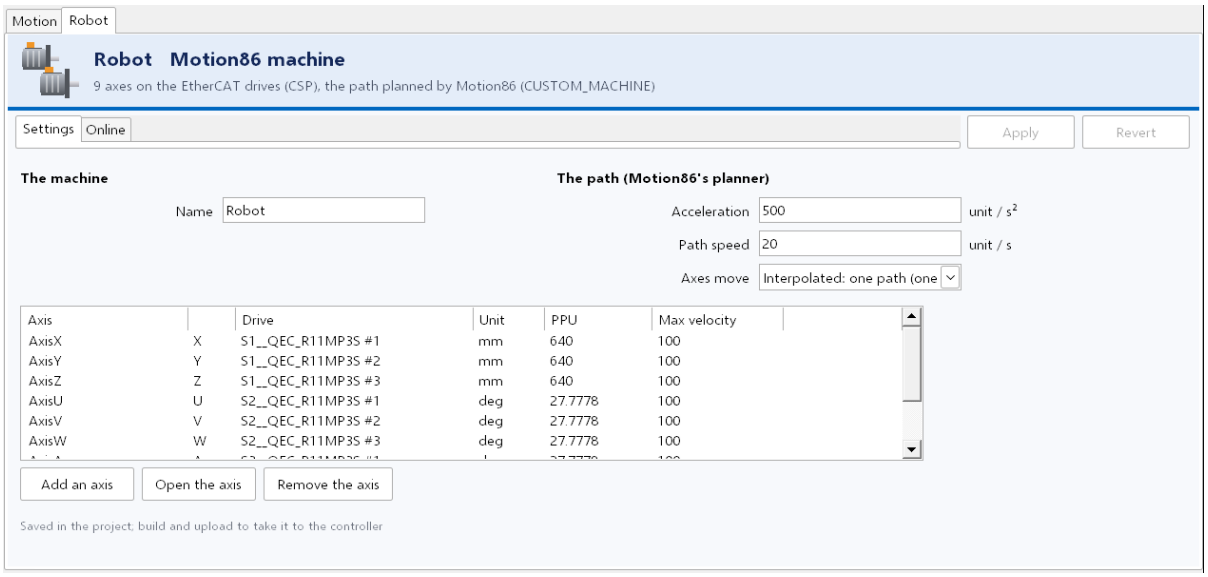


図 12-6　9 軸マシン Robot の「Settings」：軸リストの 2 列目が軸文字（X〜C）。リストは上下にスクロールできます

1. マシン作成時に「Number of axes」で 9 を選択します。またはマシンページで **Add an axis** をクリックし、1 軸ずつ追加します（9 軸まで）。
2. 各軸をダブルクリックし、軸ページの「Drive」で CiA 402 ドライブを選択して、単位、1 回転あたりのパルス数、1 回転あたりの移動量を設定します（12.4 節）。直線軸は mm、回転軸は deg を使用します。
3. 「Axes move」は **Interpolated** のままにして **Apply** をクリックし、ビルドしてアップロードします。

12.4　軸の設定

プロジェクトツリーの軸をダブルクリックすると、軸ページの「**Settings**」タブが開きます。変更後は **Apply** をクリックします。

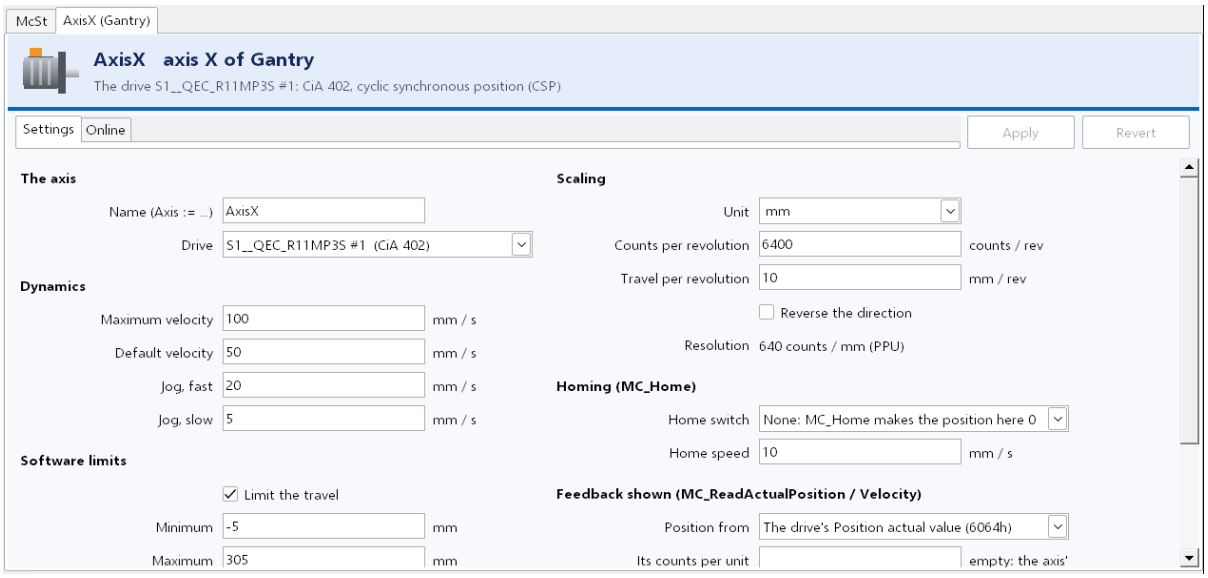


図 12-7　軸ページの「Settings」：ドライブ、ダイナミクス、ソフトウェアリミット、スケーリング、原点復帰

区分	項目	説明
Axis	Name (Axis := ...)	プログラム内の軸名
Axis	Drive	例：`S1__QEC_R11MP3S #1`。多軸ドライブでは #1／#2／#3 を選択します。仮想軸（ドライブなし、位置＝指令値）も選択できます
Dynamics	Max velocity、Default velocity	unit/s。MC_* の Velocity が 0 のときは Default velocity を使用します
Dynamics	Jog, fast／Jog, slow	軸ページのジョグボタンと MC_Jog（Velocity が 0 の場合）の速度
Software limits	Limit the travel、Minimum、Maximum	チェックを入れると、範囲外への移動は拒否されます
Scaling	Unit、Counts per revolution、Travel per revolution	Resolution（PPU）が自動計算されて表示されます
Scaling	Reverse the direction	モーターの方向が座標と逆の場合にチェックを入れます
Homing (MC_Home)	Home switch	**none**：MC_Home が現在位置を 0 に設定します。**The drive's home switch (60FDh bit 2)**、または **a BOOL of the GVL**
Homing (MC_Home)	Home speed	原点スイッチを探す速度（スイッチから離れるときはその 1/10）

***注意：シミュレーターはソフトウェアリミットを適用しません。シミュレーション中は MC_MoveAbsolute などの指令がリミットを超えても通常どおり移動します。コントローラー向けにビルドした場合、リミットは Motion86 の位置リミットで処理されます（「Limit the travel」の最小値、最大値に従います）。**

リミットを超えたときのコントローラーの実際の反応は、実機で低速・短距離で確認してから、それに基づいてエラー処理を設計してください。

12.4.1　表示するフィードバック

「**Feedback shown (MC_ReadActualPosition / Velocity)**」区分では、軸ページに表示される位置と速度、および MC_ReadActualPosition／MC_ReadActualVelocity で読み取る値の取得元を指定します：

項目	説明
Position from	デフォルト：**The drive's Position actual value (6064h)**。EtherCAT ネットワーク内の任意の TxPDO 項目（例：QEC-R11MP3S のエンコーダー位置）に変更できます。表示値 = 項目値 ÷ `Its counts per unit` + `Its offset`
Its counts per unit	選択した項目の 1 単位あたりのカウント数。**空欄 = 軸の設定を使用**（この軸の PPU。Reverse the direction を含む）。負の数を入力すると反転します
Its offset	表示値に加えるオフセット。単位は軸の単位と同じです
Velocity from	デフォルト：**The axis' (Motion86, from the moves)**。TxPDO 項目（例：606Ch 実際速度）に変更できます。MC_ReadActualVelocity = 項目値 ÷ `Its counts/s per unit/s`
Its counts/s per unit/s	速度項目の換算。空欄 = PPU

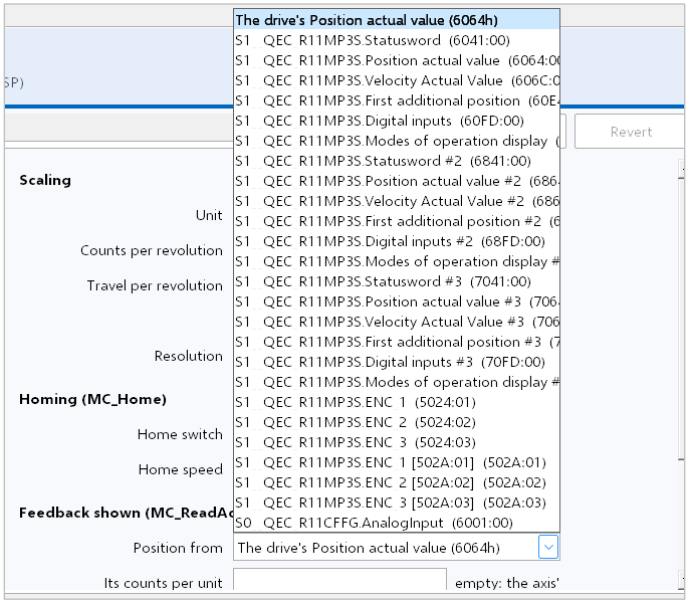


図 12-8　軸ページ「Settings」の「Feedback shown」：位置の取得元として TxPDO 項目を選択

*__重要：__位置／速度の取得元が__影響するのは表示__と __MC_ReadActualPosition／MC_ReadActualVelocity__ だけです。CSP 制御、位置決め完了の判定、原点復帰、ソフトウェアリミット、M86_ReadPosition、マシンページの座標は引き続きドライブの 6064h を使用します。

MC_Home、MC_SetPosition は外部項目の読み取り値を変更しません。原点復帰後も、表示されるフィードバック値はゼロになりません（86LDmicro 2.0.0 の制限）。必要に応じて「Its offset」で合わせてください。

外部エンコーダー（例：60E4h）の分解能は通常モーターと異なるため、「Its counts per unit」にはそのエンコーダーの 1 単位あたりのカウント数を入力してください。空欄の場合はモーターの PPU が使われ、換算が誤っていると値が拡大されたり飛んだりして見えます。

シミュレーション時は EtherCAT の値がないため、ドライブ（シミュレーター）の位置が表示されます。選択した項目が ENI に存在しなくなった場合は、ビルド時に警告が表示され、ドライブの値が表示されます。

12.5　マシンの試運転

シミュレーション（F5）またはオンライン接続（Ctrl+F5）の後、マシンページの「**Online**」タブに切り替えると、プログラムを書かずにマシン全体を操作できます。シミュレーション中でもオンラインでもない場合は、ページに「Offline: start the simulation (F5) or go online (Ctrl+F5) to try the axis」と表示されます。

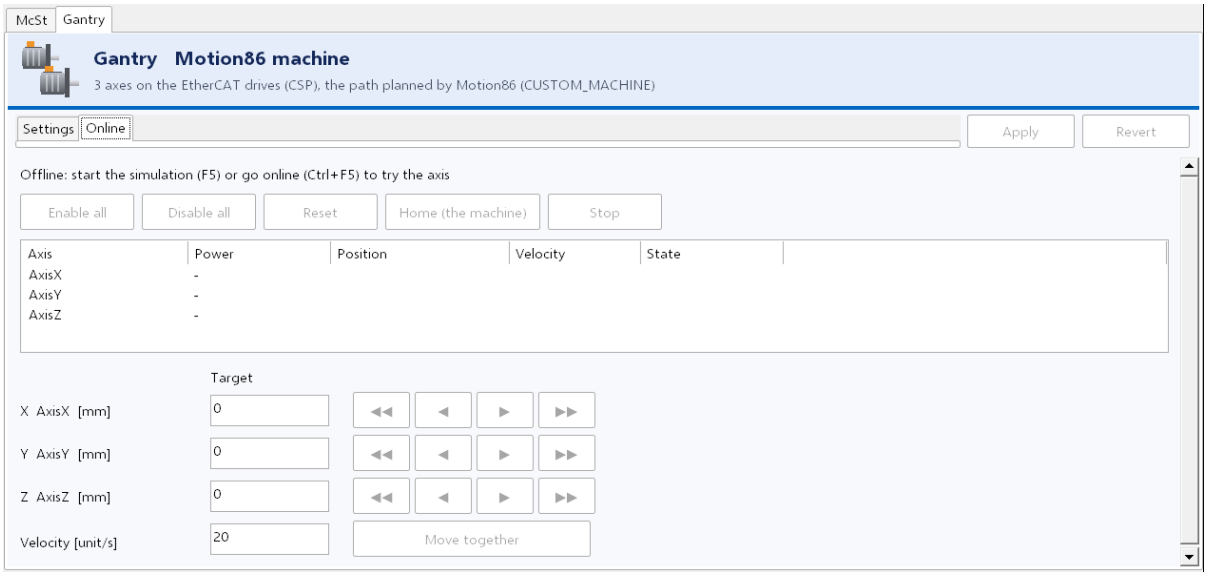


図 12-9　マシンページの「Online」タブ：上部に各軸の状態、下部に目標、速度、ジョグ

ボタン／項目	説明
Enable all	全軸のドライブが Operation Enabled になり、「Power」列に ON と表示されます
Disable all	停止してから全ドライブのサーボをオフにします
Jog ◀◀ ◀ ▶ ▶▶	軸ごとに 1 行：高速後退、低速後退、低速前進、高速前進。**押している間だけ動き、離すと停止します**
Target、Velocity、Move together	各軸の「Target」欄に位置、「Velocity」欄に unit/s を入力し、**Move together** をクリックします。補間モードでは各軸が 1 回の直線移動を行って同時に到達し、独立モードでは各軸が同時に動き出してそれぞれ目標位置に到達します
Home (the machine)	原点スイッチが設定された軸が順に原点を探します。スイッチのない軸は位置が変わりません
STOP	全軸を即時停止します
Reset	エラーをクリアします

1. **Enable all** をクリックし、各軸の「Power」が ON であることを確認します。
2. **低速ジョグ**（◀ ▶）で各軸の方向を確認します。
3. 「Target」に X 80、Y 40、Z 20、「Velocity」に 20 を入力し、**Move together** をクリックします。
4. 終了したら **Disable all** をクリックします。

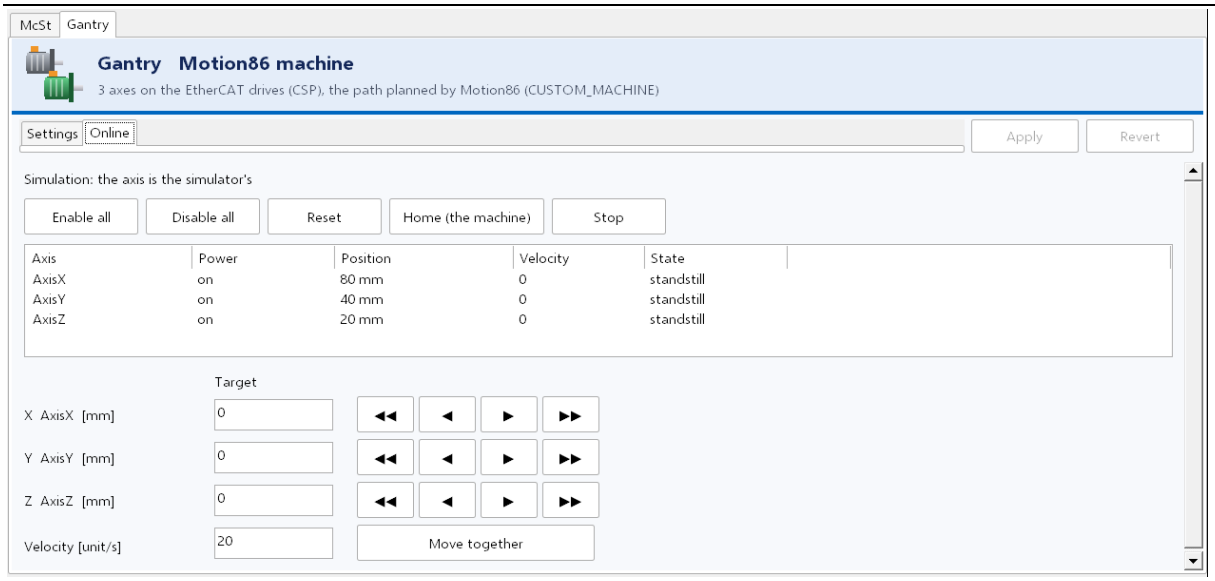


図 12-10　Gantry のオンライン：3 軸がサーボオンし、X 80、Y 40、Z 20 mm に移動

3 軸を超える場合は軸ごとに目標とジョグボタンの行があり、行数は軸数に応じて増えます（配置はウィンドウサイズによって異なります）。画面に収まらない場合は、マウスホイールまたは右側のスクロールバーで下にスクロールし、「Velocity」と Move together を表示します：

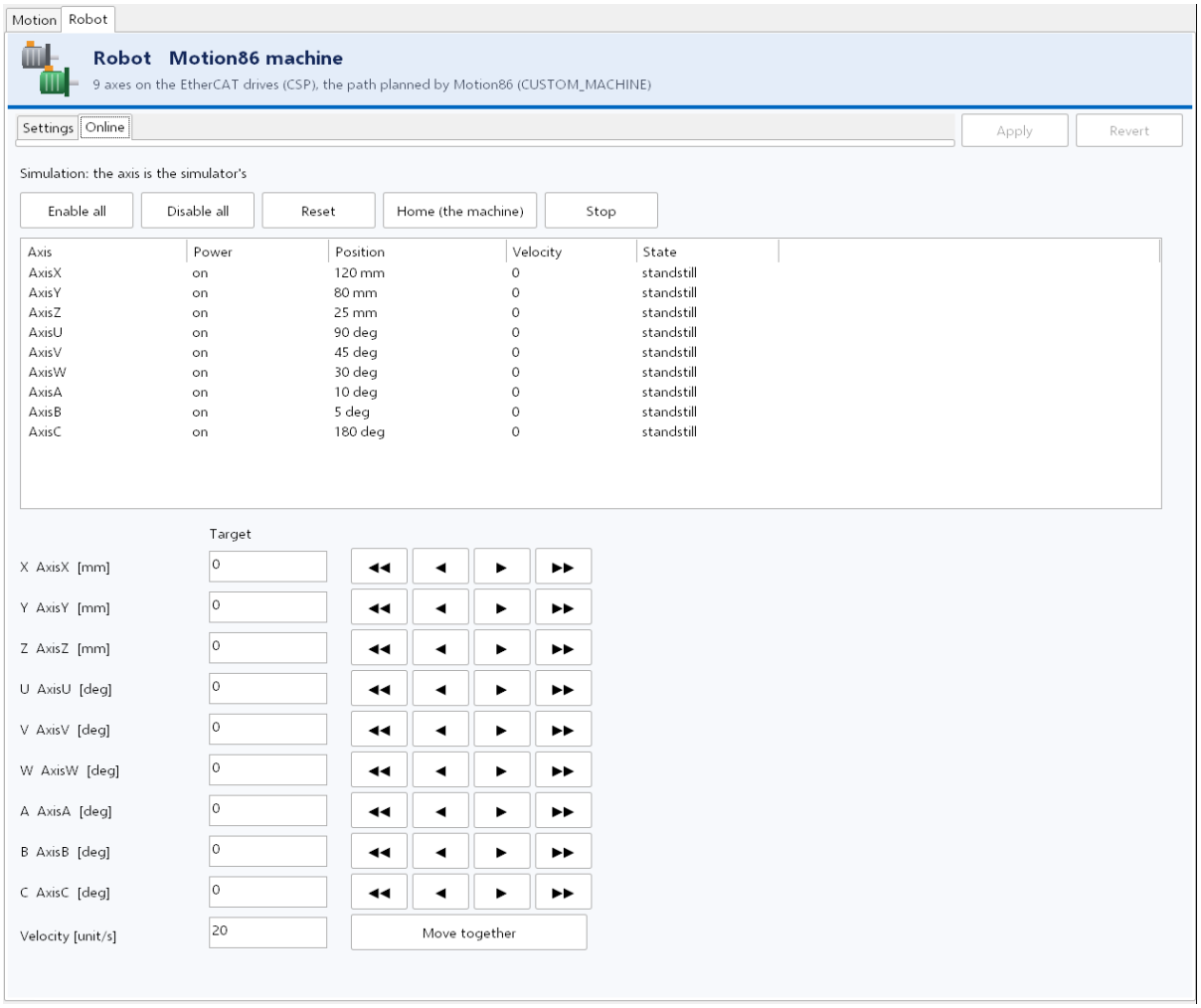


図 12-11　9 軸マシンの「Online」：上部に各軸の状態、下部に各軸の目標、ジョグ、「Move together」

12.6　単軸の試運転

軸ページの「**Online**」タブでは単一の軸を操作でき、位置、速度、追従誤差、指令、直近 30 秒のトレンドが表示されます：

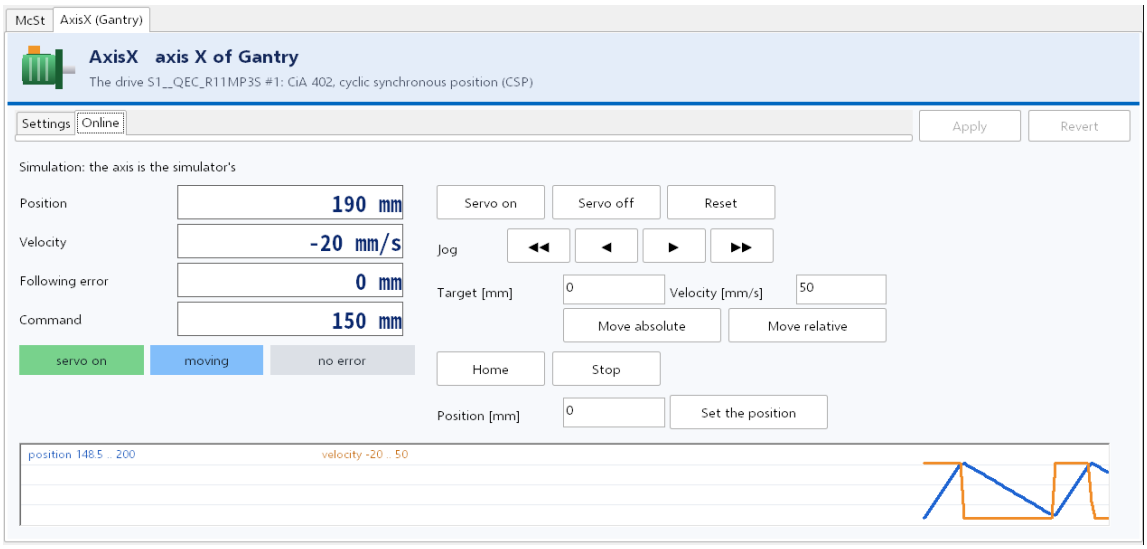


図 12-12　軸ページの「Online」：位置、速度、追従誤差、指令、状態ランプ、サーボオン、ジョグ、移動、原点、位置設定、トレンドグラフ

ボタン／項目	説明
Enable／Disable／Reset	ドライブのサーボオン／オフ。Reset でエラーをクリアします
状態ランプ	サーボオン、停止中／移動中、エラー
Jog ◀◀ ◀ ▶ ▶▶	押している間だけ動きます（200 ms ごとに指令を更新し、コントローラーは 600 ms 受信がないと自動的に停止します）
Move absolute／Move relative	目標（または距離）と速度を入力します
Home	軸の原点スイッチの設定に従います（12.4 節）
STOP	即時停止します
Set the position	現在位置を入力値に設定します。ドライブは動きません（MC_SetPosition と同じ）
Trend	位置（青）と速度（オレンジ）、直近 30 秒。シミュレーション中またはオンライン時に表示されます

12.6.1 Axes ウィンドウ（試運転）

「View > Axes」には、プロジェクトのすべての軸のサーボオン状態、位置、指令状態が表示され、直接サーボオン、ジョグ、移動、原点復帰、STOP を行えます。Motion86 軸をダブルクリックするとその軸ページが開きます。プロジェクトツリーのマシンを右クリック >「The Axes panel (commissioning)」でも開けます。

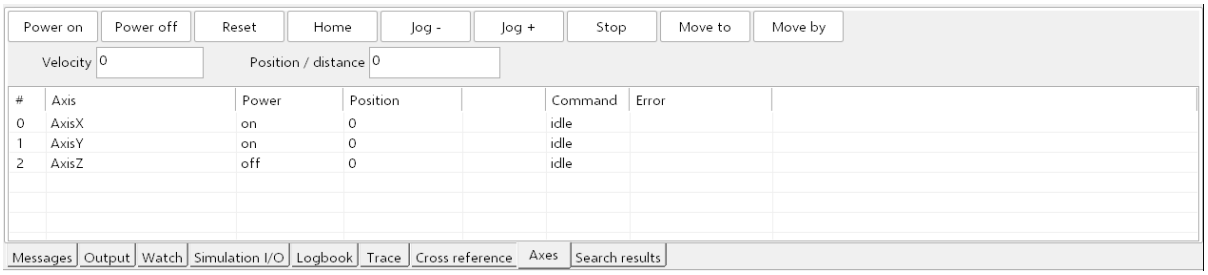


図 12-13　Axes ウィンドウ：全軸の状態と操作

12.7 PLCopen MC ファンクションブロック

プログラムでは PLCopen の MC_* ファンクションブロックで**単一の軸**を制御します。ST、SFC では直接呼び出し、ラダー図では **CALL** で挿入します（7.8、7.11 節）。

12.7.1 基本ルール

- `Axis :=` には軸名（例：`AxisX`）を指定します。軸名は DINT 定数です。
- 位置と距離は軸の単位（mm など）、速度は unit/s で、パラメータの型はすべて **REAL** です。Velocity が 0 の場合は軸のデフォルト速度を使用します。
- Acceleration／Deceleration はマシンの加速度で決まります（渡された値は互換性のために残されています）。
- ファンクションブロックのインスタンスは VAR で宣言し、**毎スキャン呼び出す**必要があります（少なくとも Execute が変化するときと Done を待つ間は呼び出してください）。
- 補間モードでは、同じマシンの MC_* の移動は順に待ち行列に入ります。複数軸を同時に直線移動させるには M86_MoveLinear を使用してください（12.8 節）。

12.7.2 ファンクションブロックの共通動作

種類	トリガー	出力
Execute 型（MoveAbsolute、MoveRelative、MoveVelocity、Home、Stop、Halt、Reset、SetPosition）	`Execute` の**立ち上がりエッジ**でコマンドを送ります。再度送るには、まず Execute を FALSE に戻します	`Busy`（実行中）→ `Done`（完了）または `Error`（`ErrorID`）。新しいコマンドに置き換えられた場合は `CommandAborted`。Done／Error は Execute が TRUE の間保持されます
Enable 型（Power、ReadActualPosition、ReadActualVelocity、ReadStatus、ReadAxisError）	`Enable` が TRUE の間、継続して有効	`Valid`（MC_Power では `Status`）と読み取った値
MC_Jog	`JogForward`／`JogBackward` が TRUE の間移動します（両方が TRUE、または両方が FALSE の場合は停止）	`Busy`、`Done`、`Error`

12.7.3 ファンクションブロック一覧

ファンクションブロック	入力	出力	説明
MC_Power	Axis, Enable	Status, Valid, Error, ErrorID	サーボオン。Status が TRUE ならサーボオン済み
MC_MoveAbsolute	Axis, Execute, Position, Velocity, Acceleration, Deceleration	Done, Busy, Active, CommandAborted, Error, ErrorID	絶対位置へ移動
MC_MoveRelative	Axis, Execute, Distance, Velocity, Acceleration, Deceleration	同上	指定距離を移動
MC_MoveVelocity	Axis, Execute, Velocity, Acceleration, Deceleration	InVelocity, Busy, Active, CommandAborted, Error, ErrorID	MC_Stop まで指定速度で移動し続けます
MC_Jog	Axis, JogForward, JogBackward, Velocity, Acceleration, Deceleration	Done, Busy, CommandAborted, Error, ErrorID	ジョグ（継続して呼び出す必要があります）
MC_Home	Axis, Execute, Method, Velocity, Acceleration	Done, Busy, Active, CommandAborted, Error, ErrorID	原点復帰（軸の原点スイッチの設定に従います）
MC_Stop	Axis, Execute, Deceleration	Done, Busy, CommandAborted, Error, ErrorID	停止（即時停止）
MC_Halt	Axis, Execute, Deceleration	Done, Busy, Active, CommandAborted, Error, ErrorID	一時停止（即時停止）
MC_Reset	Axis, Execute	Done, Busy, Error, ErrorID	エラーをクリア
MC_SetPosition	Axis, Execute, Position	Done, Busy, Error, ErrorID	現在位置を Position に設定（ドライブは動きません）
MC_ReadActualPosition	Axis, Enable	Valid, Busy, Error, ErrorID, Position	現在位置（REAL）
MC_ReadActualVelocity	Axis, Enable	Valid, Busy, Error, ErrorID, Velocity	現在速度
MC_ReadStatus	Axis, Enable	Valid, Busy, Error, ErrorID, ErrorStop, Disabled, Stopping, Homing, StandStill, DiscreteMotion, ContinuousMotion, SynchronizedMotion	軸の状態
MC_ReadAxisError	Axis, Enable	Valid, Busy, Error, ErrorID, AxisErrorID	軸のエラーコード

12.7.4 例：単軸のサーボオン、原点復帰、3 回の往復

シナリオ：Gantry の X 軸。サーボオンスイッチ Xenable をオンにした後、Xstart を押すと、X 軸が原点復帰し、200 mm へ移動してから 50 mm 戻る往復を 3 回行った後、150 mm で停止します。Xstop で軸を停止して待機状態に戻ります（再度 Xstart を押すと原点復帰からやり直します）。エラーが発生するとエラーのステップで停止して Yerror が点灯し、Xreset を押すとリセットされて待機状態に戻ります。Xstart を押し続けても 1 回だけ実行されます。

変数	型	I/O マッピング	用途
Xenable	BOOL	デジタル入力	サーボオンスイッチ
Xstart	BOOL	デジタル入力	起動ボタン
Xstop	BOOL	デジタル入力	停止ボタン
Xreset	BOOL	デジタル入力	エラーリセットボタン
Ybusy	BOOL	デジタル出力	動作中表示灯
Yerror	BOOL	デジタル出力	エラー表示灯
Mpos	REAL	—	X 軸の現在位置（mm）
step、count	INT	—（McSt のローカル変数）	ステップ、往復済み回数

```
PROGRAM McSt
VAR
    pwr  : MC_Power;
    home : MC_Home;
    mv   : MC_MoveAbsolute;
    rel  : MC_MoveRelative;
    stp  : MC_Stop;
    rst  : MC_Reset;
    rd   : MC_ReadActualPosition;
    st   : MC_ReadStatus;
    ae   : MC_ReadAxisError;
    startEdge : R_TRIG;      // Xstart の新たな押下
    resetEdge : R_TRIG;      // Xreset の新たな押下
    resetReq  : BOOL;        // MC_Reset を要求済み
    step  : INT;             // 0 待機、10 原点、20 往路、30 復路、90 停止中、99 エラー
    count : INT;             // 往復済み回数
    lastError : DINT;        // 最後のエラーの ErrorID
END_VAR
pwr(Axis := AxisX, Enable := Xenable);
rd(Axis := AxisX, Enable := TRUE);
Mpos := rd.Position;
st(Axis := AxisX, Enable := TRUE);
ae(Axis := AxisX, Enable := TRUE);
startEdge(CLK := Xstart);
resetEdge(CLK := Xreset);

(* 1. 移動中はエラーと停止を優先：現在のコマンドを終了 *)
IF step >= 10 AND step <= 30 THEN
    IF home.Error OR mv.Error OR rel.Error OR st.ErrorStop THEN
        IF home.Error THEN lastError := home.ErrorID;
        ELSIF mv.Error THEN lastError := mv.ErrorID;
        ELSIF rel.Error THEN lastError := rel.ErrorID;
```

```
        ELSE lastError := ae.AxisErrorID;     (* 軸自体のみが ErrorStop に移行 *)
        END_IF;
        step := 99;
    ELSIF Xstop THEN
        step := 90;
    END_IF;
END_IF;

(* 2. ステップ *)
CASE step OF
0:  IF pwr.Status AND startEdge.Q AND NOT Xstop THEN   (* 新たな押下、かつ停止が押されていない *)
        count := 0;
        step := 10;
    END_IF;
10: IF home.Done THEN step := 20; END_IF;
20: IF mv.Done THEN step := 30; END_IF;
30: IF rel.Done THEN
        count := count + 1;
        IF count < 3 THEN step := 20; ELSE step := 0; END_IF;
    END_IF;
90: IF stp.Error THEN
        lastError := stp.ErrorID;
        step := 99;
    ELSIF stp.Done AND NOT Xstop THEN    (* 停止済み：待機に戻る。再度 Xstart で原点復帰から開始 *)
        step := 0;
    END_IF;
99: IF resetEdge.Q THEN resetReq := TRUE; END_IF;   (* オペレーターによるリセット *)
    IF rst.Error THEN                    (* リセット失敗：エラーを記録。再度 Xreset を押せる *)
        lastError := rst.ErrorID;
        resetReq := FALSE;
    ELSIF rst.Done THEN
        resetReq := FALSE;
        IF NOT st.ErrorStop THEN step := 0; END_IF;
    END_IF;
END_CASE;

(* 3. コマンド：Execute は自分のステップでのみ TRUE。ステップを離れるとコマンドを終了し、
      そのステップに戻ったときに新たな立ち上がりエッジとなる *)
home(Axis := AxisX, Execute := step = 10);
mv(Axis := AxisX, Execute := step = 20, Position := 200.0, Velocity := 50.0);
rel(Axis := AxisX, Execute := step = 30, Distance := -50.0, Velocity := 20.0);
stp(Axis := AxisX, Execute := step = 90);
rst(Axis := AxisX, Execute := resetReq AND step = 99);
Ybusy := step >= 10 AND step <= 30;
Yerror := step = 99;
END_PROGRAM
```

#	操作
1	第 11 章に従って ENI を読み込み（例：ex30_dispense の `data\86ld\ethercat\QEC-ENI-1.xml` をインポート）、12.2 節に従って 3 軸マシン Gantry（AxisX、AxisY、AxisZ）を作成します。
2	GVL で Xenable、Xstart、Xstop、Xreset、Ybusy、Yerror（BOOL）と Mpos（REAL）を宣言します。
3	「Project > Add a PROGRAM (ST)...」で名前を `McSt` とし、上記のプログラムを入力します。
4	**Check**（Shift+F7）をクリックし、エラーがないことを確認します。
5	**Simulate**（F5）をクリックします。Watch ウィンドウに Xenable、Xstart、Mpos、Ybusy、McSt.step、McSt.count を追加します。

#	操作
6	Xenable に TRUE を書き込み、次に Xstart に TRUE を書き込みます（約 1 秒後に FALSE に戻します）。AxisX の軸ページの「Online」を開いて観察します。

期待される結果：X 軸が原点復帰し（原点スイッチがない場合は現在位置を 0 に設定）、50 mm/s で 200 mm へ移動（約 4 秒）した後、20 mm/s で 150 mm へ戻ります。3 回繰り返した後 150 mm で停止し、McSt.step は 0 に戻り、Ybusy が消灯します。Xstart を押し続けても、完了後に自動的に再実行されることはありません。Xstop を押している間は Xstart を押しても起動しません。移動中に Xstop に TRUE を書き込むと軸が停止し、step は 90 を経て 0 に戻ります。Xstop を離して再度 Xstart を押すと、原点復帰からやり直します。

***注意：エラー時はプログラムがステップ 99 で停止し（Yerror 点灯、lastError に ErrorID を記録）、Xreset を押すまでリセットされません。リセットに成功するとステップ 0 に戻ります。エラーは移動ステップでのみ判定するため、99 に戻り続けることはありません。リセットに失敗した場合（rst.Error）は ErrorID を記録し、再度 Xreset を押して再試行できます。**

シミュレーターは軸のエラーを発生させず、ソフトウェアリミットも適用しません（12.4 節）。エラーとリセットの流れは、実機で制御可能な方法（例：低速、短距離）で確認し、事前に停止と非常停止が有効であることを確認してください。

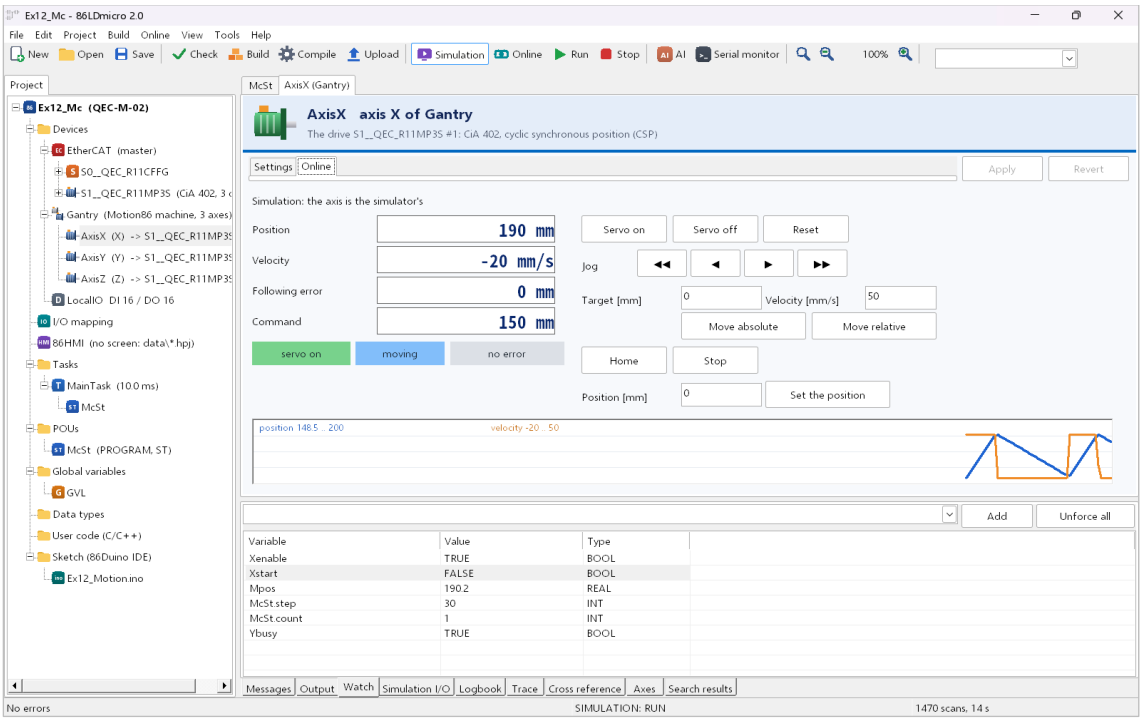


図 12-14　シミュレーション中の McSt：Watch ウィンドウの step、count、Mpos と AxisX 軸ページのトレンド

***ヒント：**SFC で MC を使用する場合、ステップを離れるとそのアクションは実行されなくなります。次のステップで前の移動の Execute を FALSE に戻しておくと、次にそのステップに戻ったときに新たな立ち上

がりエッジになります。MC_Power は別の ST または LD プログラムで常時呼び出すことをお勧めします（8.6 節）。

12.8 M86 マシンファンクションブロック

M86_* ファンクションブロックは**マシン全体**を制御します。複数軸同時の直線補間、円弧補間と G-code に対応します。`Machine :=` にはマシン名を指定します。

ファンクションブロック	入力	出力	説明
M86_Power	Machine, Enable	Status	マシンの全軸をサーボオン
M86_MoveLinear	Machine, Execute, X, Y, Z, Feedrate	Done, Busy, Error	X、Y、Z への直線補間（Feedrate は unit/min、0 ならマシンの経路速度を使用）。U〜C 軸は動きません
M86_MoveLinear9	Machine, Execute, X, Y, Z, U, V, W, A, B, C, Feedrate	Done, Busy, Error	マシンの全軸（最大 9 軸）で 1 本の直線経路を移動し、同時に到達します。軸数が少ない場合、余った入力は使用されません
M86_MoveArcXY	Machine, Execute, CX, CY, X, Y, Clockwise, Feedrate	Done, Busy, Error	XY 平面の円弧（中心 CX、CY、終点 X、Y）
M86_Home	Machine, Execute	Done, Busy, Error	マシン全体の原点復帰
M86_GCode	Machine, Execute, Line	Done, Busy, Error	G-code を 1 行実行します（例：`'G1 X50 Y0 F3000'`）。記述のない軸は動きません
M86_Stop	Machine, Execute, Emergency	Done	マシンを停止
M86_ReadPosition	Machine, Enable	Valid、X〜C、Moving	各軸の現在位置と移動中かどうか
M86_Override	Machine, Ratio	—	速度オーバーライド（0〜2）

- `Feedrate` の単位は **unit/min** です（例：1200 ＝ 20 mm/s）。0 はマシンの経路速度を使用することを意味します。
- `Execute` が FALSE から TRUE に変化したときに 1 回だけ移動を送ります。再度移動するには、まず Execute を FALSE に戻します。
- M86_* の座標は MC_* と同じです（MC_SetPosition のオフセットを含む）。G-code 文字列はマシン座標です。M86_MoveLinear、M86_MoveLinear9、M86_MoveArcXY は実行前に絶対座標（G90）に切り替えます。M86_GCode はモードを変更しないため、一度 G91 を送ると以降の G-code は相対座標で実行されます。先に G90 を送ってください。
- 独立モードのマシンでは M86_MoveArcXY と M86_GCode は実行されません（12.3 節）。

12.8.1 例：直線、円弧、G-code

シナリオ：Gantry をサーボオン（Xrun）した後、Xgo を押すと、XY が (50, 30) へ直線移動し、中心 (25, 30) で反時計回りに半円を描いて (0, 30) へ移動し、さらに G-code で (10, 10) へ移動します。Mx、My に現在位置が表示されます。

変数	型	I/O マッピング	用途
Xrun	BOOL	デジタル入力	マシンのサーボオン
Xgo	BOOL	デジタル入力	経路の実行
Mx、My	REAL	—	現在の X、Y（mm）

```
PROGRAM Path
VAR
    mp  : M86_Power;
    lin : M86_MoveLinear;
    arc : M86_MoveArcXY;
    gc  : M86_GCode;
    rp  : M86_ReadPosition;
    step : INT;
END_VAR
mp(Machine := Gantry, Enable := Xrun);
rp(Machine := Gantry, Enable := TRUE);
CASE step OF
0:  IF mp.Status AND Xgo THEN step := 20; END_IF;
20: lin(Machine := Gantry, Execute := TRUE, X := 50.0, Y := 30.0, Z := 0.0, Feedrate := 1200.0);
    IF lin.Done THEN lin(Machine := Gantry, Execute := FALSE); step := 30; END_IF;
30: arc(Machine := Gantry, Execute := TRUE, CX := 25.0, CY := 30.0, X := 0.0, Y := 30.0,
        Clockwise := FALSE, Feedrate := 600.0);                    (* 半円：(50,30) → (0,30) *)
    IF arc.Done THEN arc(Machine := Gantry, Execute := FALSE); step := 40; END_IF;
40: gc(Machine := Gantry, Execute := TRUE, Line := 'G1 X10 Y10 F3000');   (* 前の lin、arc で G90 に戻っている：絶対座標 *)
    IF gc.Done THEN gc(Machine := Gantry, Execute := FALSE); step := 0; END_IF;
END_CASE;
Mx := rp.X;
My := rp.Y;
END_PROGRAM
```

#	操作
1	12.7.4 のプロジェクト（マシン Gantry は補間モード）を使用し、GVL で Xrun、Xgo（BOOL）と Mx、My（REAL）を宣言します。
2	「Project > Add a PROGRAM (ST)...」で名前を Path とし、上記のプログラムを入力します。タスクの設定から McSt を削除します（2 つのプログラムで同時に Gantry を制御しないでください）。
3	**Check** をクリックしてから **Simulate**（F5）をクリックします。Watch ウィンドウに Xrun、Xgo、Mx、My、Path.step を追加します。
4	Xrun に TRUE を書き込み、次に Xgo に TRUE を書き込みます。Gantry のマシンページの「Online」を開いて観察します。

期待される結果：(50, 30) への直線移動に約 3 秒（20 mm/s）、(0, 30) への半円に約 8 秒（10 mm/s）かかり、G-code で (10, 10) へ移動した後、Path.step が 0 に戻ります。Xgo が TRUE のままだと、同じ経路が再度実行されます。

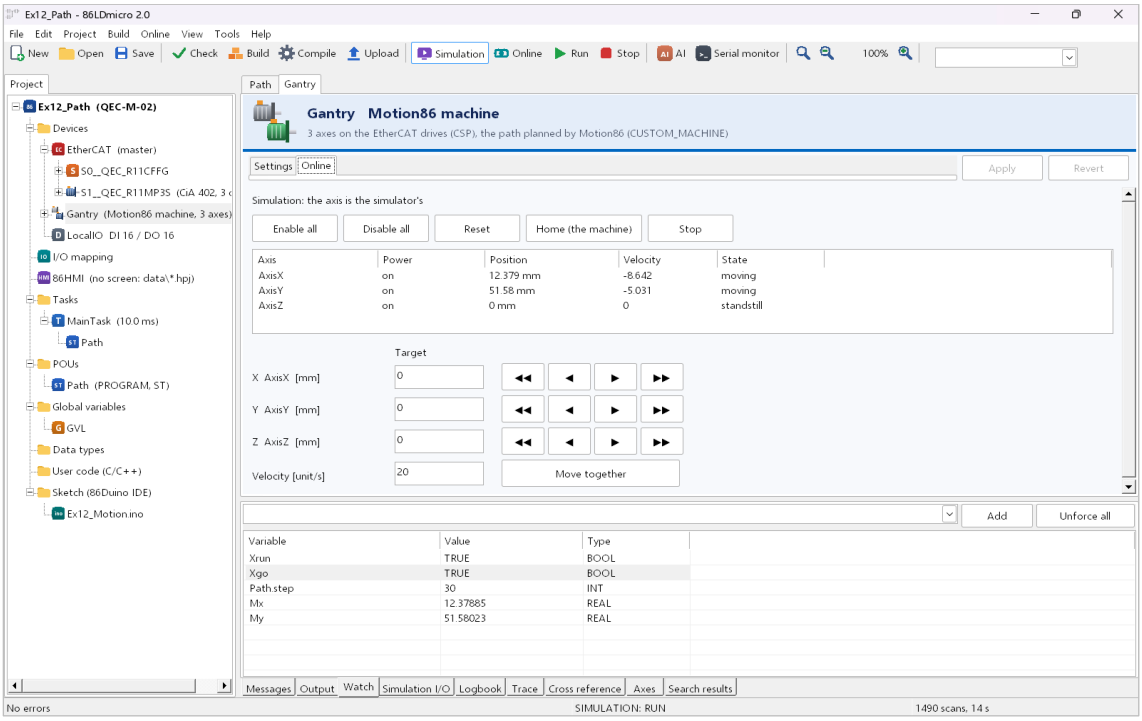


図 12-15　シミュレーション中の Path：円弧を描いている間にマシンページに表示される X、Y 位置

12.8.2　例：9 軸の同時移動

9 軸マシン Robot（AxisX～AxisC）：Xpower が TRUE の間サーボオンし、Xgo9 を押すと M86_MoveLinear9 で 9 軸を同時に目標へ到達させ、M86_ReadPosition で各軸の位置を読み返します。Xpower、Xgo9 は GVL の BOOL（初期値 FALSE）なので、プログラムをロードしただけでサーボオンや移動が行われることはありません。目標座標は各軸の単位（直線軸は mm、回転軸は deg）です。初回実行前に目標がストローク内にあることを確認し、Feedrate を下げてください。停止するには M86_Stop を使用するか、Xpower を FALSE にします。

```
PROGRAM Motion
VAR
    pw  : M86_Power;
    all : M86_MoveLinear9;
    rd  : M86_ReadPosition;
END_VAR
pw(Machine := Robot, Enable := Xpower);                    (* サーボオン：Xpower *)
all(Machine := Robot, Execute := Xgo9 AND pw.Status,     (* Xgo9 を押すと移動を 1 回送る *)
    X := 120.0, Y := 80.0, Z := 25.0, U := 90.0, V := 45.0, W := 30.0,
    A := 10.0, B := 5.0, C := 180.0, Feedrate := 6000.0);
rd(Machine := Robot, Enable := TRUE);   (* rd.X .. rd.C：現在位置 *)
END_PROGRAM
```

G-code でも U～C 軸を指定できます。例：`M86_GCode` の `Line := 'G1 X10 U90 C180 F3000'`。

12.8.3 C/C++ での軸の制御

ユーザーコード（C/C++、第 14 章）からも軸を制御できます：

関数	説明
`AXIS(AxisX)`	軸番号（以下の関数に渡します）
`mc_power(ax, on)`、`mc_is_on(ax)`	サーボオン／オフ、サーボオン済みかどうか
`mc_move_abs(ax, pos, vel)`、`mc_move_rel(ax, dist, vel)`	絶対／相対移動（vel が 0 ならデフォルト速度を使用）
`mc_jog(ax, dir, vel)`	ジョグ：dir は +1／−1／0。継続して呼び出す必要があり、最後の呼び出しから 0.6 秒後に自動停止します
`mc_stop(ax)`、`mc_home(ax)`、`mc_set_position(ax, pos)`、`mc_reset(ax)`	停止、原点復帰、位置設定、エラークリア
`mc_position(ax)`、`mc_velocity(ax)`	現在位置、速度
`mc_state(ax)`、`mc_done`、`mc_busy`、`mc_error`、`mc_moving`、`mc_error_code`	状態（0 アイドル、1 実行中、2 完了、3 エラー）
`M86_MACHINE(Gantry)`	Motion86 の `Machine *`（86LDmicro の座標オフセットとジョグ管理を経由しません）

***注意：**mc_* は `plc86_user_loop`／`plc86_user_before`／`plc86_user_after` で呼び出し、`plc86_user_ethercat`（割り込み）では呼び出さないでください。

12.9 安全機構

86LDmicro と Motion86 には次の保護機能が組み込まれていますが、これらはハードウェアの非常停止や安全回路の代わりにはなりません。

状況	動作
PLC STOP、ウォッチドッグ	プログラムの実行が止まります。すべてのマシンが停止し、ジョグも終了します。一般の出力は最後の値を保持します（10.4、10.5 節）
ウォームリスタート／コールドリスタート	ウォームリスタート：RETAIN 変数は保持され、その他の変数（出力に使う変数を含む）は初期値に戻ります。コールドリスタート：すべての変数（RETAIN を含む）が初期値に戻ります。どちらもすべての強制を解除し、RUN／STOP 状態は変わりません。次のスキャンから出力は初期値で書き出されます（RUN 中はその後プログラムが再計算します）。リスタートの前にマシンを停止してください（STOP または MC_Stop／M86_Stop）
ジョグが継続して更新されない	軸ページのボタン、MC_Jog、`mc_jog` は 600 ms 更新がないと自動的に停止します
サーボオン中にドライブが Operation Enabled から外れた（異常、Quick stop、STO、ネットワーク断）	マシン全体が停止し、**自動では再びサーボオンしません**。Logbook に `Motion86: an axis' drive left operation enabled: the machine stopped (MC_Reset, MC_Power again)` が記録されます。MC_Reset の後に MC_Power が必要です
サーボオフ	先にマシンを停止してから、ドライブをサーボオフします
サーボオフ中に軸が手で動かされた	位置の基準がずれます。Logbook に `Motion86: an axis moved while its drive was off: …` が記録されます。再度原点復帰するか MC_SetPosition を使ってください
原点復帰で原点スイッチが見つからない	Logbook に次が記録されます：`Motion86: homing found no home switch`
ソフトウェアリミット	「Limit the travel」をオンにすると、最小値／最大値を超える移動は拒否されます

***安全：MC_Stop／MC_Halt は Motion86 では即時停止で、減速ランプはありません。**実機では外部の非常停止と安全回路（ドライブの主電源の遮断または STO）を設計してください。

初めて実機で実行するときは、低速で、方向とストロークを確認し、非常停止を準備してください。

12.9.1 初回試運転チェックリスト

実機で初めてモーターを回す前に、項目ごとに確認してチェックを付けてください。2 人での作業をお勧めします。1 人がパソコンを操作し、もう 1 人が非常停止のそばに立ちます。

#	通電前	確認
A1	非常停止がハードウェアの安全回路（ドライブの主電源の遮断または STO）に接続されており、押すとモーターが確実に出力できなくなる	☐
A2	機構の可動範囲内に人、工具、異物がない。ストロークの両端にメカストッパーまたはリミットスイッチがある	☐
A3	モーター動力線、エンコーダー線、EtherCAT ケーブルがドライブのマニュアルどおりに接続され、確実に接地されている	☐
A4	ドライブのパラメーター（モーター型式、電子ギア、リミット）がドライブのマニュアルどおりに設定されている	☐

#	通電、通信	確認
B1	シリアルモニターに `EtherCAT: running, N slaves, … sync` と表示され、N が実際のサブデバイス数と一致している	☐
B2	SubDevice ページのオンライン値が更新される。ドライブの Statusword（6041h）が読める	☐
B3	ENI の PDO に 6040h／607Ah の出力と 6041h／6064h の入力が含まれている（11.3.6）	☐
B4	マシンと軸の設定（単位、1 回転あたりのパルス数、1 回転あたりの移動量、ドライブの割り当て）が正しい	☐

1. まず**シミュレーション**（F5）で、マシンページの「Online」を使って一度操作し、ボタンの位置と方向に慣れておきます。
2. プログラムをアップロードして接続し、「Axes」ウィンドウまたは軸ページの「Online」を開きます。
3. **1 軸ずつ**サーボオンし、「enabled」ランプが点灯し、位置が飛ばないことを確認します。
4. **低速でジョグ**（◀ ▶）して毎回少しずつ動かし、正方向を確認します。逆の場合はサーボオフし、軸ページの「Reverse the direction」をオンにして、再度ビルドしてアップロードします。
5. 短い距離（例：5 mm）の**相対移動**で、実際の移動量が指令と同じであることを確認します。異なる場合は 1 回転あたりの移動量を確認します。
6. **原点復帰**を実行し、正しい位置で止まることを確認します。
7. すべての軸を確認してから、マシンページで「Move together」を行い、速度を低速から段階的に上げていきます。
8. 自分の MC／M86 プログラムは最後に実行します。

#	初回動作	確認
C1	各軸の方向が正しい	☐

#	初回動作	確認
C2	各軸の移動量が正しい（指令 5 mm、実際 5 mm）	☐
C3	STOP を押す、PLC STOP、EtherCAT ケーブルを抜く、のいずれでも軸が停止する	☐
C4	ハードウェアの非常停止で全軸が停止する	☐
C5	原点復帰が正常に動作する	☐

第 13 章　86HMI ヒューマンマシンインターフェース

86HMI は 86Duino のタッチスクリーン HMI です。画面（ページ、ボタン、ランプ、数値など）は 86Duino IDE の **86HMI Editor** で作成します。86LDmicro は画面上のウィジェットを PLC 変数に**バインド**し、シミュレーション中とオンライン中に画面とプログラムを連動させます。この章では、有効化、バインド、各ウィジェットの操作方法、テキスト／トレンド／アラーム／ページ切り替え、シミュレーション、およびリソースのアップロードと言語ボタンについて順に説明します。

作業	作業場所	本章
ページの作成、ウィジェットの配置、外観と言語の設定	86HMI Editor（86Duino IDE の「Tools > 86HMI」、または 86HMI ページの **86HMI Editor**）	13.1、13.6
ウィジェット ↔ 変数のバインド	86LDmicro の 86HMI ページ、プロジェクトツリーの「86HMI」、ラダー要素の右クリック	13.2〜13.4
パソコン上で操作を試す	86HMI シミュレーターウィンドウ（F5 でシミュレーションすると自動的に開きます）	13.5
画面をコントローラーに送る	アップロード（F8）、またはリソースのみアップロード	13.6

13.1　86HMI を有効にする

86HMI の画面は sketch の `data\_project.hpj` に保存されます。86HMI Editor で **Code Generation** を押すと、`myhmi.cpp`／`myhmi.h` と画面で使う画像とフォント（`data\Resource`）が書き出され、`.ino` に `#include "myhmi.h"` と `Hmi.begin();` が追加されます。その後 86LDmicro がこの `.hpj` を読み込み、すべてのページとウィジェットを一覧表示します。

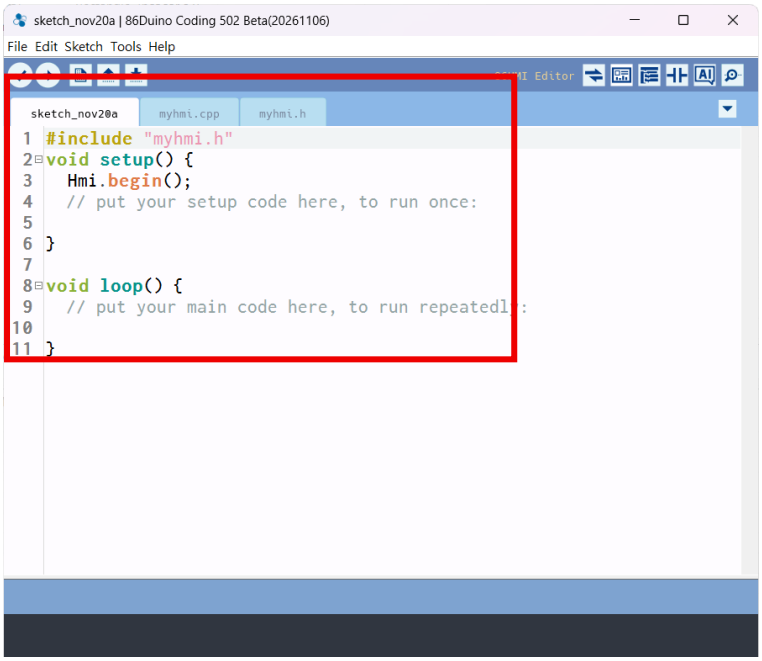


図 13-0　86HMI の Code Generation

13.1.1　86Duino IDE から 86HMI Editor を開く

86Duino IDE で sketch を開いてから「**Tools > 86HMI**」をクリックすると、86HMI Editor が開きます。これは 1.1 の頃と同じ方法です。

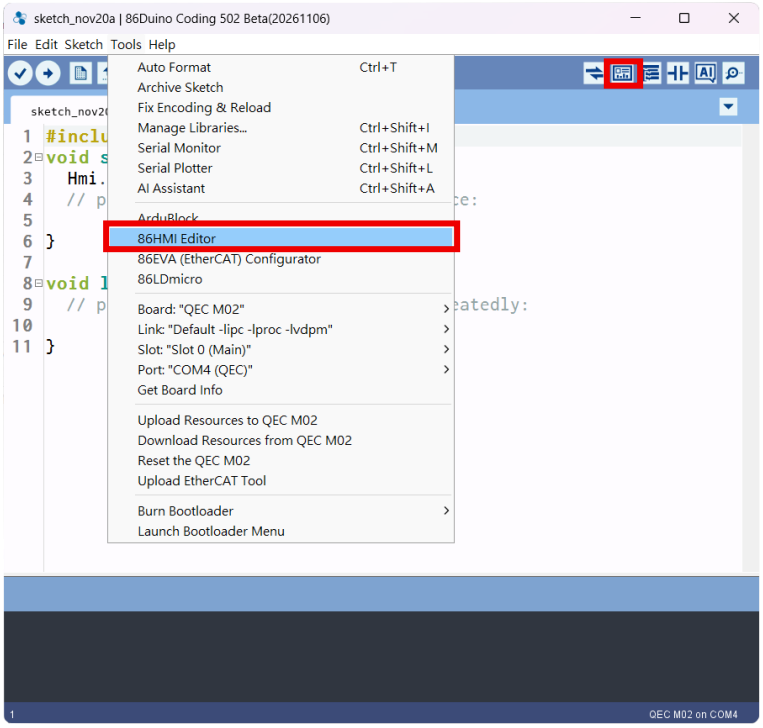


図 13-1　86Duino IDE の「Tools」メニュー：86HMI と 86LDmicro

13.1.2　86LDmicro から 86HMI Editor を開く（推奨）

1. プロジェクトツリーで「**86HMI**」をダブルクリックするか、「**Project > 86HMI screen and bindings**」で 86HMI ページを開きます。

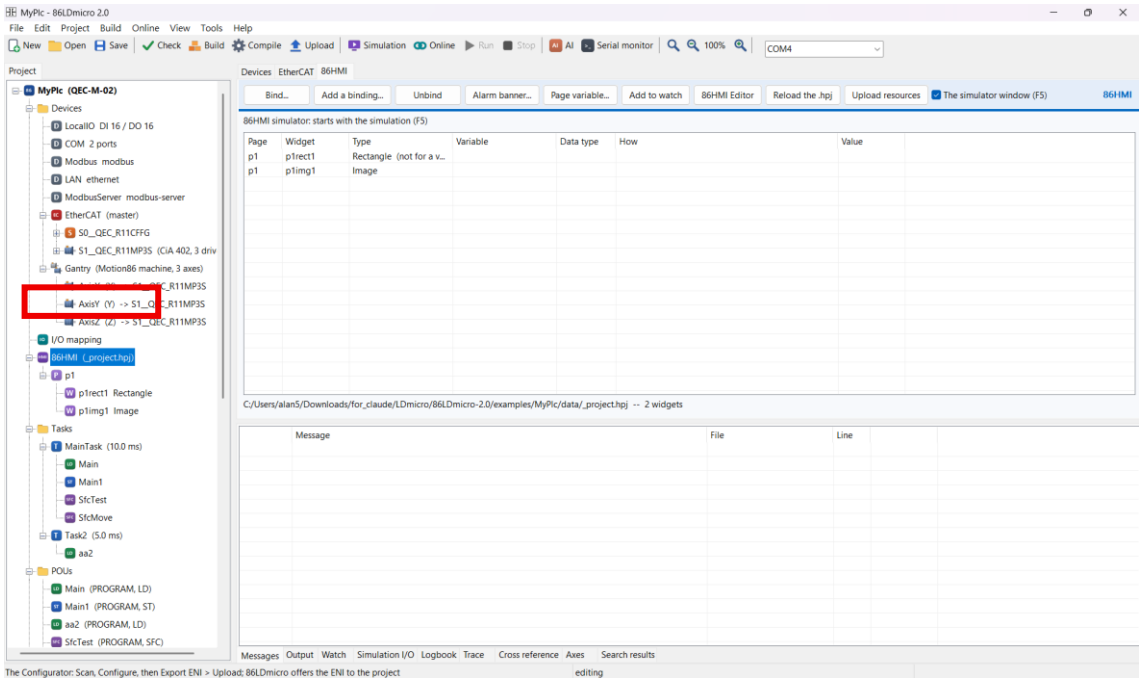


図 13-2-1　86LDmicro の Project > 86HMI screen and bindings

2. **86HMI Editor** を押します。86Duino IDE の「Tools > 86HMI」と同じ方法でエディターが開き、このプロジェクトの sketch の `data\_project.hpj` が開かれます。sketch にまだ画面がない場合は新しい画面が作成されます。

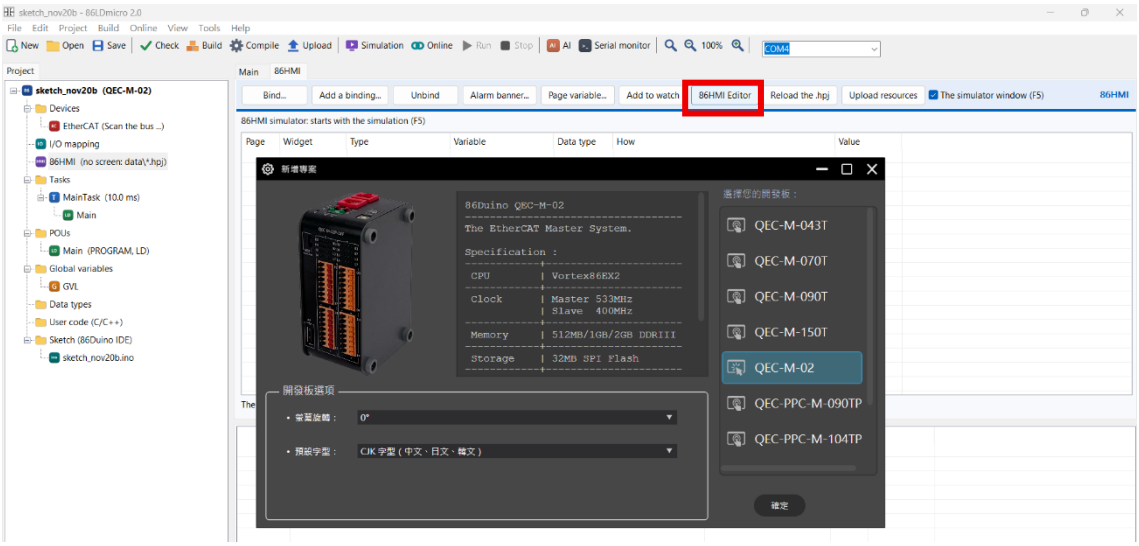


図 13-2-2　86LDmicro と 86HMI Editor、新しいプロジェクトを開く

3. エディターでページを追加し、ウィジェット（Button、Switch、LED、Number、Label、Image など）を配置して、名前（例：`p1b1`、`p1led1`）と外観を設定します。

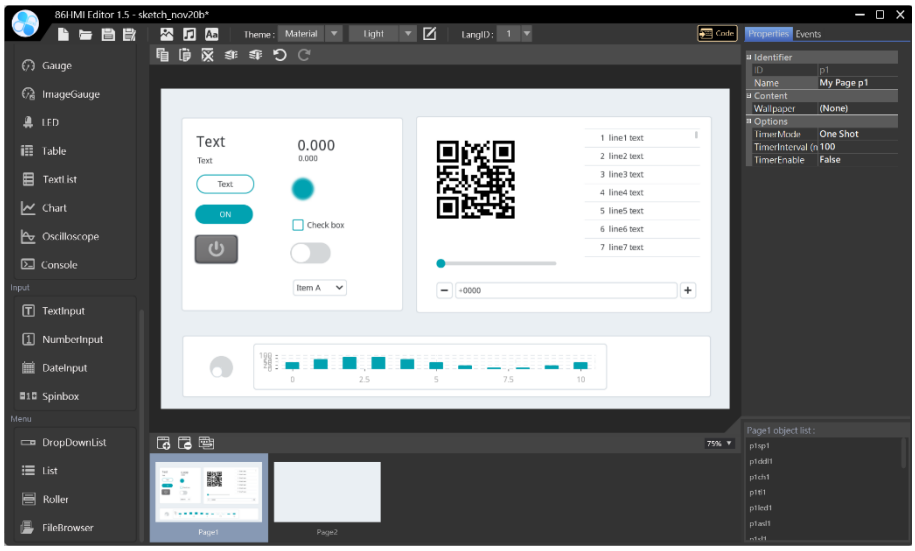


図 13-2-3　86HMI Editor

4. ボタン **Code Generation** を押します。`data\_project.hpj`、`myhmi.cpp`、`myhmi.h`、`data\Resource` が保存され、`.ino` が修正されます。

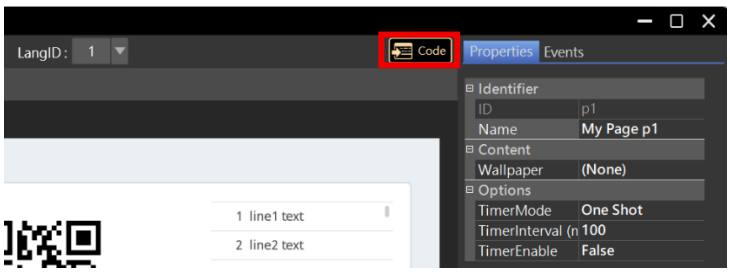


図 13-2-4　86HMI Editor4.　Code Generation を押す

5. 86LDmicro に戻ると、86HMI ページとプロジェクトツリーが自動的に再読み込みされます（ステータスバーに「86HMI: the screen is saved in the sketch (data_project.hpj, myhmi.cpp / myhmi.h): build to use it」と表示されます）。

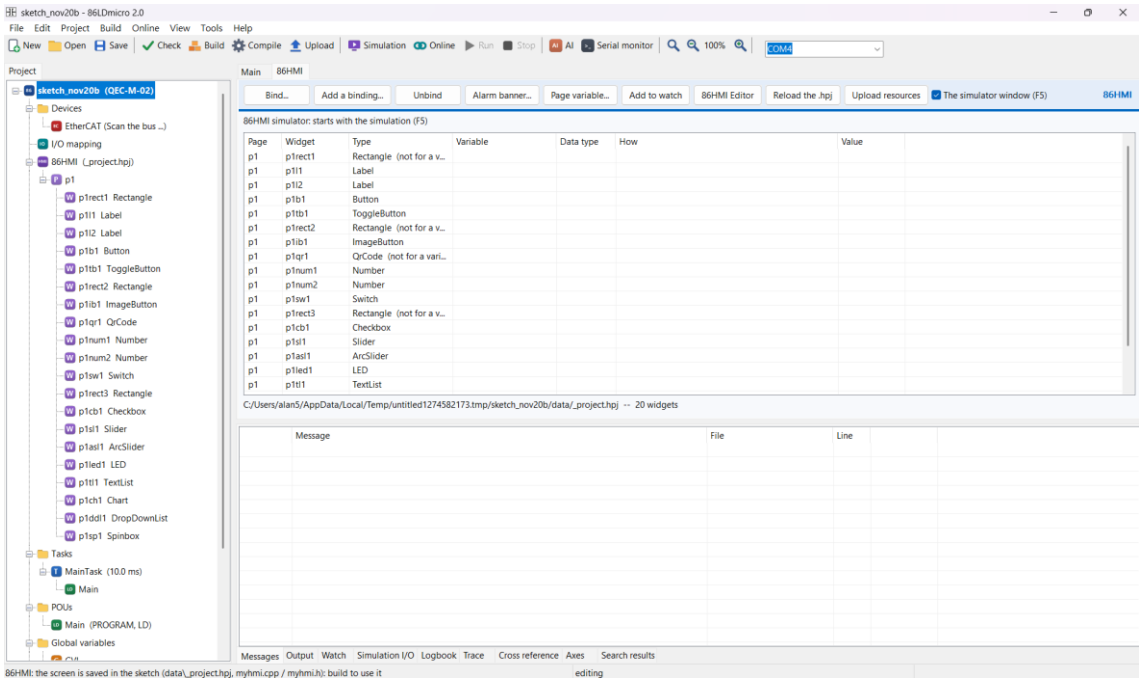


図 13-2-5　86LDmicro に戻る：86HMI ページとプロジェクトツリーが自動的に再読み込みされる

更新されない場合は **Reload the .hpj** を押します。

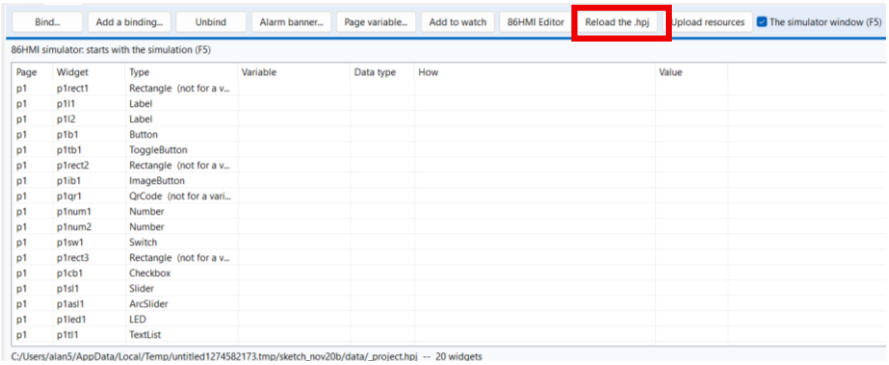


図 13-2-6　86LDmicro に戻る：86HMI が更新されない場合は Reload the .hpj を押す

*ヒント：86LDmicro が 86HMI Editor を起動するときは 86Duino IDE 付属のエディターを使うため、IDE フォルダーを正しく設定しておく必要があります（2.6 節「Options」）。86Duino IDE から 86LDmicro を起動した場合は、その IDE が自動的に使われます。

13.1.3　86HMI ページ

86HMI ページには `.hpj` 内のすべてのウィジェットが表示されます。バインド済みのウィジェットは**バインドごとに 1 行**です。上部のボタンと表の列は次のとおりです。

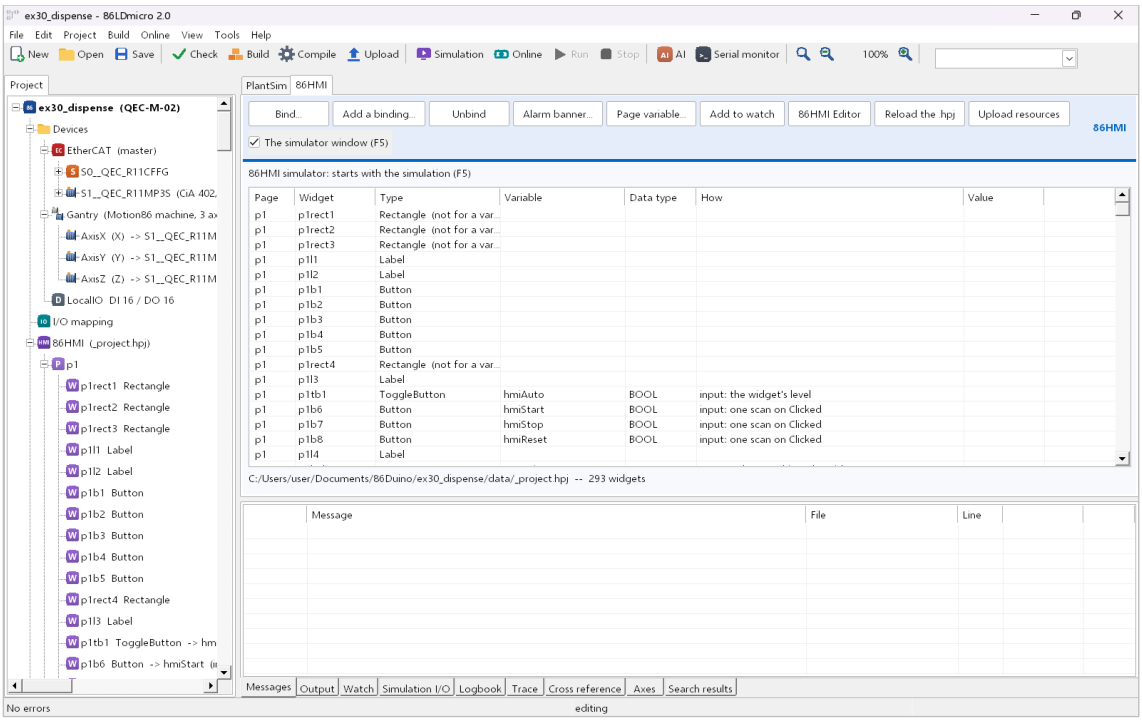


図 13-3　86HMI ページ：ボタンバーとウィジェット／バインドの一覧

ボタン	用途
Bind...	選択した行のバインドを変更します。ウィジェットがまだバインドされていない場合は新しく追加します（13.2 節）
Add a binding...	同じウィジェットにもう 1 つバインドを追加します（13.2.3 節）
Unbind	選択した行のバインドだけを削除します
Alarm banner...	BOOL 変数で制御するアラームバナー（13.4.3 節）
Page variable...	整数変数で現在表示するページを決めます（13.4.4 節）
Add to watch	選択した行の変数を「Watch」ウィンドウに追加します（9.3 節）
86HMI Editor	このプロジェクトの `data\_project.hpj` を開きます（13.1.2 節）
Reload the .hpj	画面を外部で変更した後に、ページとウィジェットを再読み込みします
Upload resources	次だけをアップロードします：`data\Resource`（13.6 節）
The simulator window (F5)（チェックボックス）	オンにすると、シミュレーション（F5）時に 86HMI シミュレーターウィンドウが自動的に開きます（13.5 節）

列	内容
Page	ウィジェットがあるページ（p1、p2 など）
Widget	86HMI Editor でのウィジェット名（例：p1b1）
Type	ウィジェットの種類（Button、Switch、LED、Number、Image (x) など）。変数をバインドできないウィジェットには「（not for a variable）」と表示されます
Variable	バインドされたグローバル変数
Data type	変数の型（BOOL、INT、REAL など）
How	入力か出力か、およびその方法（13.3 節）
Value	シミュレーション中またはオンライン中の変数のリアルタイム値

ページの最下部には `.hpj` のパスとウィジェット数が表示されます。sketch に画面がない場合、プロジェクトツリーには「86HMI（no screen: data*.hpj）」と表示されます。まず 13.1.2 節に従って画面を作成してください。

13.2 ウィジェットのバインド

バインドとは、ウィジェットをどのグローバル変数と連動させるか、そしてデータをどちらの方向に流すかを指定することです。**入力**は画面から変数を設定し（ボタン、スイッチ、数値入力）、**出力**は変数を画面に表示します（ランプ、数値、テキスト、画像の位置）。バインドは `project.86ld` の `bindings` に保存され、コントローラーにアップロードされるプログラムとシミュレーターウィンドウは同じ設定を使います。

13.2.1 バインドフォームを開く方法

- 86HMI ページでウィジェット（またはそのバインドの行）を選択して **Bind...** を押すか、その行をダブルクリックします。
- プロジェクトツリーの「86HMI > ページ > ウィジェット」をダブルクリックします。
- 変数から始める場合：ラダー要素、ウォッチの行、または ST 内の変数を右クリックし、「**86HMI binding of <変数>...**」を選んでから、ウィジェット、アラームバナー、または表示ページを選びます（13.4.5 節）。

13.2.2 バインドフォーム（ウィジェットの種類別）

フォームのタイトルは「86HMI Binding」で、1 行目にどのウィジェットか（名前、種類、ページ）が表示されます。フィールドはウィジェットの種類によって異なります。

フィールド	説明
Variable	グローバル変数。ドロップダウンから選ぶか、新しい名前を入力します。新しい名前は自動的に GVL に追加されます（ボタン／LED は BOOL、数値は INT）
How	入力（画面から変数を設定）または出力（画面に変数を表示）。選択肢はウィジェットの種類によって決まります（13.3 節）。イベントは 1 スキャンの間だけ TRUE になります
Scale	数値ウィジェット：変数 = ウィジェットの値 × スケール（出力時は ÷ スケール）。1 は換算なし
Text（Label／Table／TextList）	テンプレート（温度 `%d C`、%d は数値を表す）、または数値ごとに 1 つのメッセージを 1 行に 1 つずつ：`0`=停止、`1`=運転、`2`=アラーム（13.4.1 節）
Column, row / line（Table／TextList）	書き込むセル（Table）または行（TextList）
Channel (1..4), period (ms)（Chart／Oscilloscope）	トレンド：変数をどのチャンネルにサンプリングするか、何 ms ごとにサンプリングするか（13.4.2 節）
Enable (a BOOL, optional)	テキストとトレンドのウィジェット：空欄 = 常に更新。BOOL を指定すると、それが TRUE のときだけウィジェットを更新します

Button（ボタン）：フォームには「Variable」「How」「Scale」だけがあります。新規追加時は「Variable」が空欄で、「How」の既定値は「Input: the variable follows the widget's level」です。

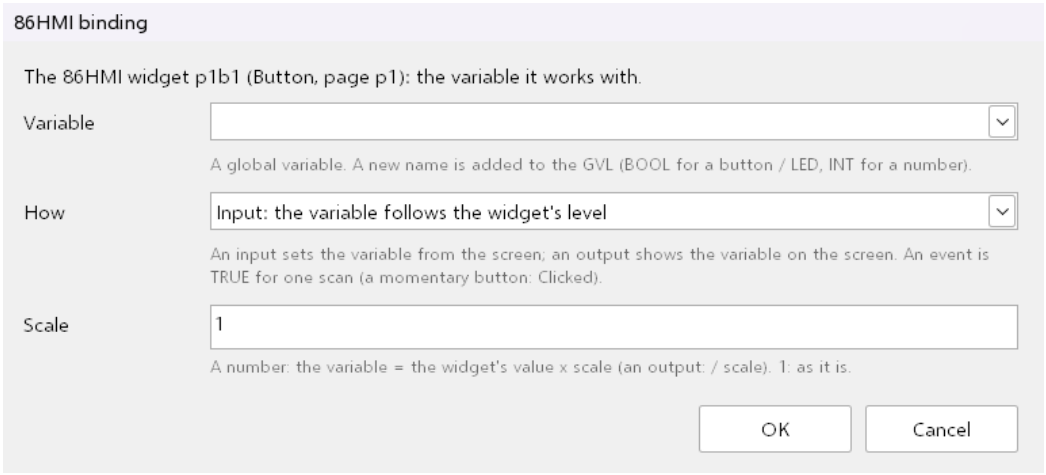


図 13-4　Button のバインド：Variable、How、Scale

Switch（スイッチ）：ON／OFF を切り替えるとその状態を保持します。通常は内部フラグ（例：`Rstart`）にバインドします。

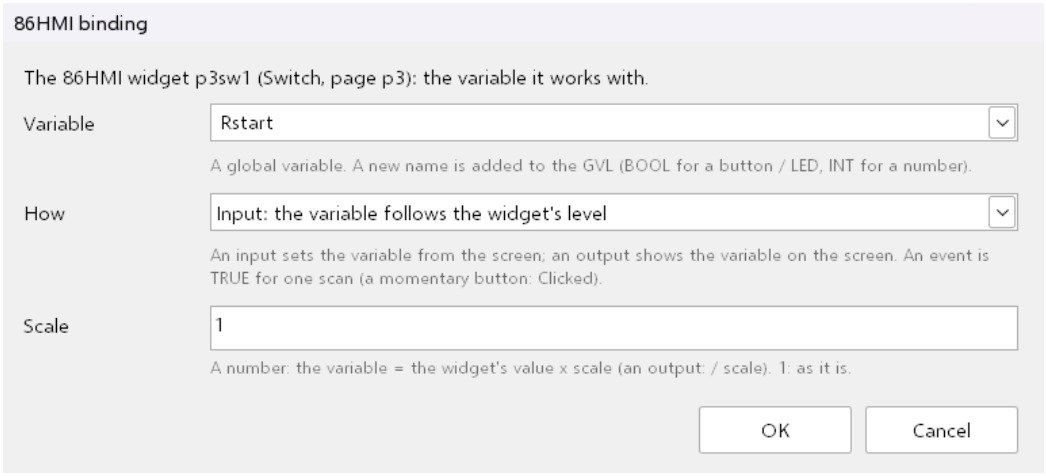


図 13-5　Switch のバインド：p1sw1 → Rstart、Input: the variable follows the widget's level

Number（数値）：出力専用で、変数の値を画面に表示します。スケールで単位換算ができます。

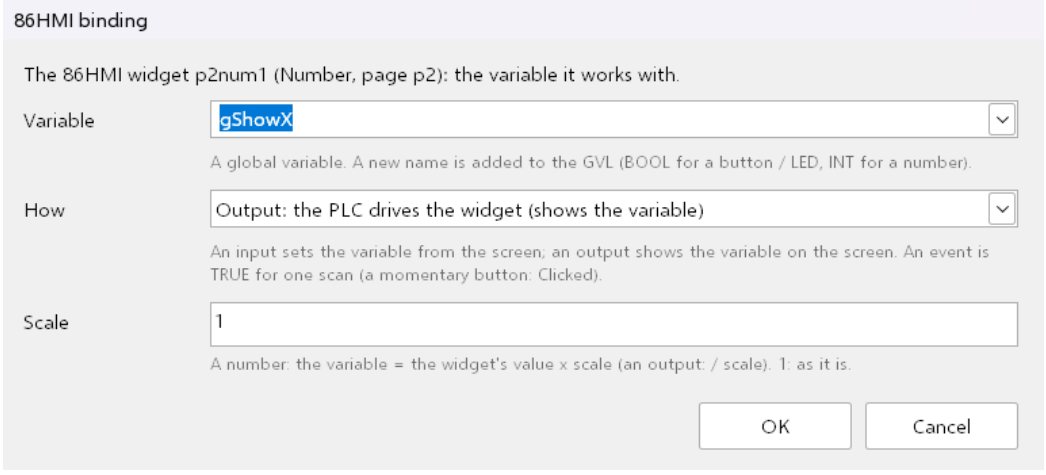


図 13-6　Number のバインド：出力、PLC がウィジェットを駆動（変数を表示）

13.2.3　図解：ボタン p1b1 を Rstop にバインドする

1. 86HMI ページでウィジェット **p1b1**（Button）を選択し、**Bind...** を押します。
2. 「Variable」に `Rstop` を入力または選択し、「How」で **Input: the variable follows the widget's level** を選びます（押している間 TRUE）。
3. **OK** を押します。表の p1b1 の「Variable」列に Rstop と表示され、プロジェクトツリーの 86HMI の下にも同様に表示されます（例：`p1b1 Button -> Rstop`）。
4. ラダー図では `Rstop` を実機の停止ボタンと一緒に使います（13.3 節の注意を参照）。

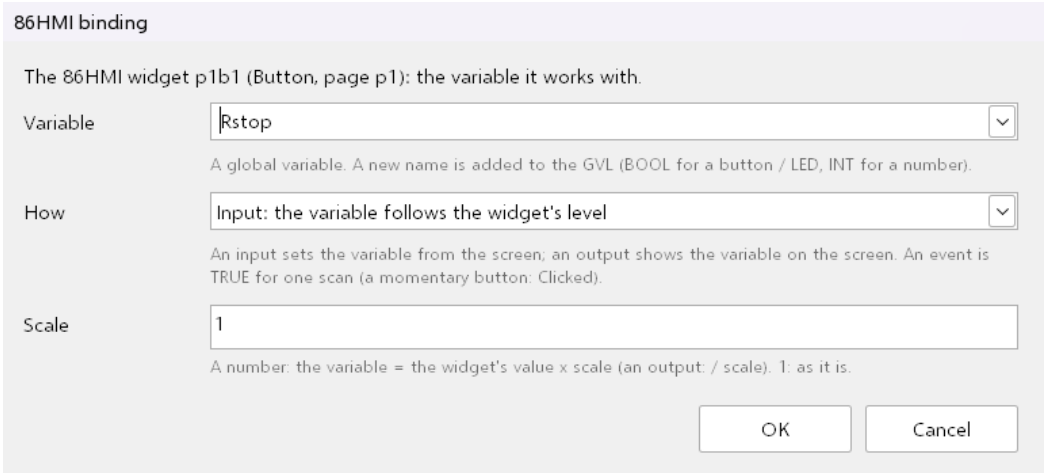


図 13-7　バインドフォーム：p1b1 → Rstop、Input: the variable follows the widget's level

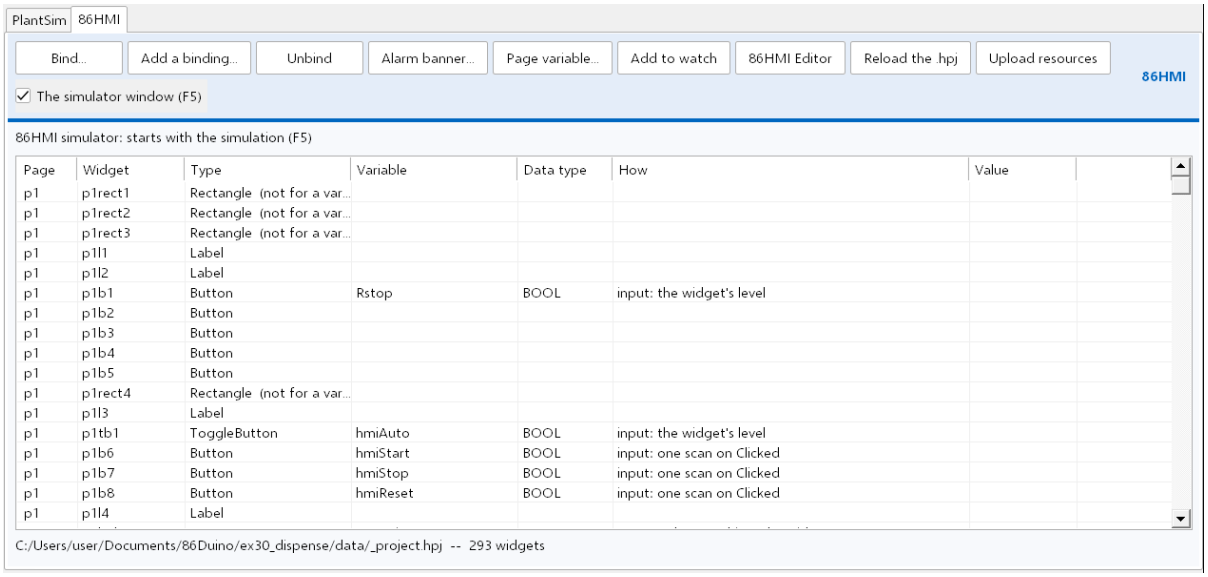


図 13-8　バインド完了：表とプロジェクトツリーの両方に p1b1 → Rstop、p1led1 → Yout、p1sw1 → Rstart、p2num1 → Mposition が表示される

13.2.4　1 つのウィジェットに複数のバインド

1 つのウィジェットを同時に複数の変数にバインドできます。例えば 1 つの Image に「表示／非表示」「x 位置」「y 位置」、Chart の各チャンネル、Table の各セル、さらに任意のウィジェットに「Enable」を 1 つ追加、などです。86HMI ページでは**バインドごとに 1 行**なので、同じウィジェットが複数行に表示されます。

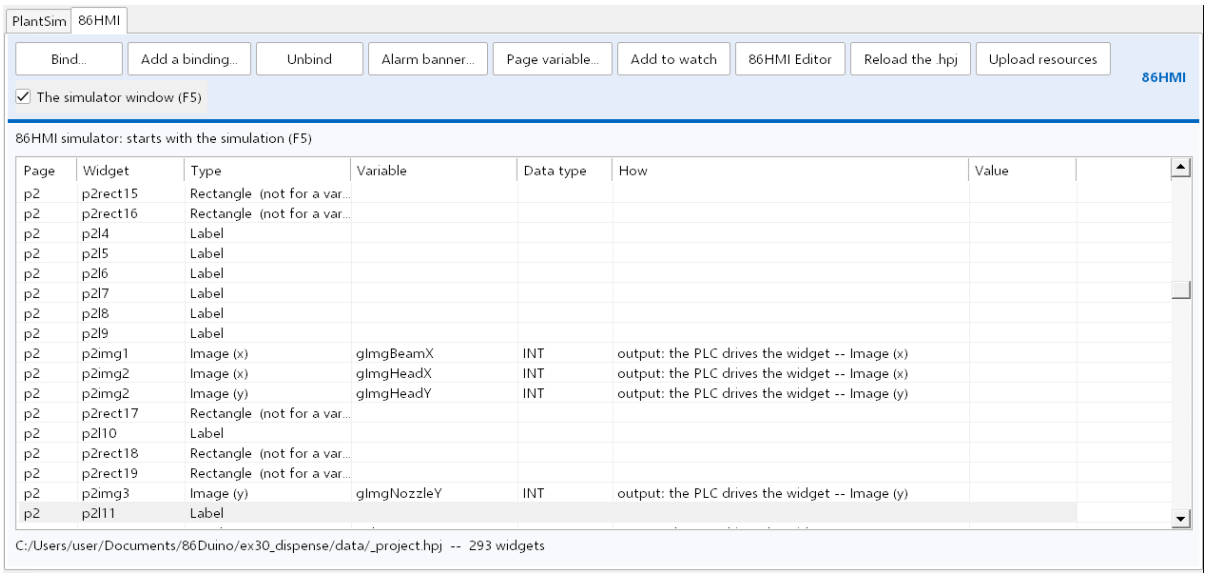


図 13-9　ex30 の 86HMI ページ：p2img1 と p2img2 にはそれぞれ Image (x) と Image (y) の 2 つのバインドがあり、バインドごとに 1 行

1. そのウィジェットのいずれかの行を選択し、**Add a binding...** を押します。
2. 「Variable」と「How」を選びます。フォームの上部に、このウィジェットにすでにあるほかのバインドの数が表示されます（「It has N other binding(s): a new one is added next to them」）。
3. **OK** を押すと一覧に 1 行増え、新しい変数は自動的に GVL に追加されます。
4. バインドを変更するには、その行を選択して **Bind...** を押します（またはダブルクリック）。削除するには、その行を選択して **Unbind** を押します。削除されるのはその 1 つだけです。

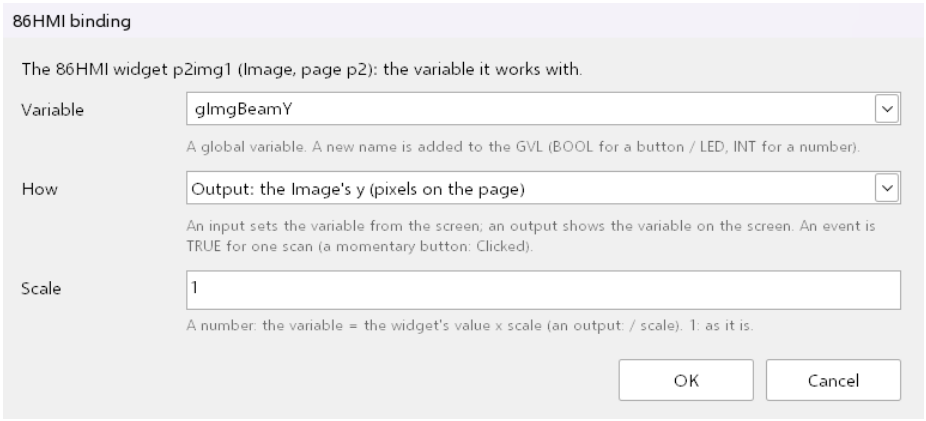


図 13-10　p2img1 に「Output: the Image's y (pixels on the page)」のバインドを追加（変数 glmgBeamY）

競合を避けるため、次の場合はブロックされ、メッセージが表示されます。

- 同じウィジェットで同じ用途が重複する（例：x 位置が 2 つ、Chart の同じチャンネル、Table の同じセル）。
- 同じ値が入力と出力の両方になる（例：Switch を入力としてバインドし、さらに出力もバインドする）。
- 変数がすでに別のウィジェットにバインドされている：先にそのウィジェットでバインドを解除してください（有効化用の BOOL は例外で、複数のウィジェットにバインドできます）。

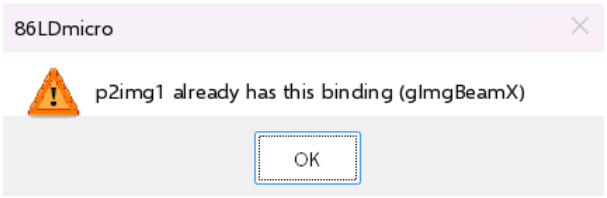


図 13-11　p2img1 にすでに x 位置のバインドがある場合、もう 1 つ追加しようとするとブロックされる

*ヒント：プロジェクトツリーでは、ウィジェットの後ろにそのすべてのバインドが表示されます。複数のバインドがあるウィジェットをダブルクリックすると、どれを変更するか、または「(a new binding)」を選べます。プロジェクトツリーで「Unbind」を押すと、すべて削除するかどうかを確認されます。

13.3 ウィジェットと操作方法

「How」の一覧には、そのウィジェットがサポートする方法だけが表示されます。次の表に、よく使うウィジェットをまとめます（ウィジェット名は 86HMI Editor での種類です）。

ウィジェット	選択できる「How」	操作
Button、ImageButton	Input: the variable follows the widget's level Input: one scan TRUE on Clicked (event) Input: one scan TRUE on Pressing (event)	状態に追従：押している間 TRUE、離すと FALSE（実機のボタンと同様。ImageButton はイベントのみ） イベント：クリックすると変数が **1 スキャン**の間 TRUE になります。「起動」「リセット」などの瞬時の指令に適しています
ToggleButton、ImageToggleButton、Checkbox	Input: the variable follows the widget's level Input: the level, inverted Input: event (Checked / Unchecked) Output: the PLC drives the widget	切り替え後に状態を保持します。PLC から状態を設定することもできます
Switch	Input: the variable follows the widget's level Input: the level, inverted (Off) Input: event (On / Off) Output: the PLC drives the widget	ON／OFF を切り替えた後に状態を保持します（例：p1sw1 → Rstart）
LED	Output: the PLC drives the widget (shows the variable)	変数が TRUE のとき点灯します（例：p1led1 → Yout）
Number	Output: the PLC drives the widget	変数の値を表示します。スケールで換算できます
NumberInput、Spinbox	Input: the variable follows the widget's level Input: event (Applied / Canceled, NumberInput only) Output: the PLC drives the widget	出力と入力を 1 つずつ同時に持てます（2 つの変数）。PLC が出力変数を変更すると画面に書き込まれ（例：レシピ値の読み込み）、オペレーターが入力した値は入力変数で読み戻されます。出力は値が変化したときだけ書き込まれるので、入力中の値を上書きしません
Slider、ArcSlider、ImageSlider	Input: the variable follows the widget's level、event (Dropped / Dragging) Output: the PLC drives the widget	ドラッグして数値を設定します
Bar、ArcBar、Gauge、ImageBar、ImageGauge	Output: the PLC drives the widget Input: the variable follows the widget's level	数値をバー、円弧、針で表示します（通常は出力で使用）
DropDownList、List、Roller	Input: the variable follows the widget's level（選択された項目、整数）、event (Selected)	オペレーターが項目を 1 つ選びます

ウィジェット	選択できる「How」	操作
Image	Output: the PLC drives the widget (show / hide) Output: the Image's x (pixels on the page) Output: the Image's y (pixels on the page)	表示：変数が TRUE のとき表示、FALSE のとき非表示 x／y：変数（INT）がページ上の画像のピクセル位置で、PLC が変数を変えると画像が移動します（例：ex30 の位置ページ）
Label、Table、TextList	Output: the value shown as text	テンプレートまたは数値ごとに 1 つのメッセージ（13.4.1 節）
Chart、Oscilloscope	Output: a trend (the variable sampled into a channel)	チャンネルごとに 1 つのバインド（13.4.2 節）
任意のウィジェット（Label、Table、Chart を含む）	Output: the widget enabled (TRUE) or greyed out (FALSE)	BOOL が FALSE のときウィジェットはグレー表示になり操作できなくなるので、現在どのボタンを押せるかを PLC が決められます。同じ BOOL を複数のウィジェットにバインドできます。値が変化したときだけ書き込まれます

13.3.1　3 種類の入力

How	変数の値	適した用途
Input: the variable follows the widget's level	ウィジェットの状態と同じ（押している間 TRUE、離すと FALSE。スイッチ ON = TRUE）	ジョグ、「押している間だけ動作」、スイッチ型の選択
Input: the level, inverted	ウィジェットの状態の反転	b 接点型のボタン（押す = FALSE）
Input: one scan TRUE on Clicked (event)	イベントが発生したときに 1 スキャンだけ TRUE になり、その後自動的に FALSE に戻ります	起動、リセット、アラーム確認などの瞬時の指令。自分で立ち上がりエッジを作る必要はありません

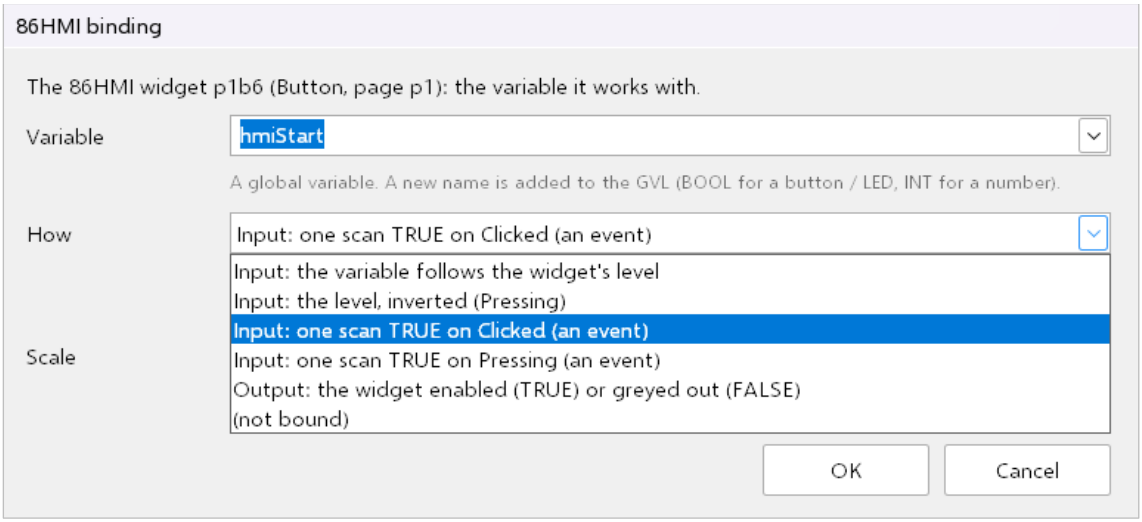


図 13-12　Button の「How」一覧：状態に追従、Clicked イベント、Pressing イベント

*ヒント：イベント（1 スキャンだけ TRUE）は**それを読むタスク**が 1 回スキャンした後にクリアされるので、遅いタスク（例：100 ms の SlowTask）でも読み取れます。同じイベントを 2 つのタスクが読む場合は最初のタスクしか保証されず、ビルド時に警告が出ます。1 つのタスクだけで読み、必要なら SET でラッチしてください。

*注意：実入力に割り当て済みの変数は HMI の入力として使えません（次のスキャンで実入力に上書きされます）。内部変数（例：`Rstart`、`Rstop`）をバインドし、プログラムの中で実機のボタンと一緒に使ってください。

例：実機の起動 `Xstart`（本体 X00）と HMI スイッチ `Rstart` を並列に、実機の停止 `Xstop` と HMI の停止ボタン `Rstop` を直列にします。

```
NETWORK 'start/stop'
    [ NO(Xstart) | NO(Rstart) | NO(Rrun) ] NC(Xstop) NC(Rstop) OUT(Rrun)
NETWORK 'lamp'
    NO(Rrun) OUT(Yrun)
```

シミュレーションで確認：`Rstart` が TRUE になると `Rrun`、`Yrun` が TRUE になって保持され、`Rstop` が TRUE になると `Rrun` が FALSE になります。

13.3.2 有効化（Enable）：どのボタンを押せるかを PLC に決めさせる

どのウィジェットにも「**Output: the widget enabled (TRUE) or greyed out (FALSE)**」のバインドを 1 つ追加できます。例えば、運転中は「原点復帰」ボタンをグレー表示にして誤操作を防ぎます。

1. ウィジェットを選択し、**Add a binding...**（ウィジェットにすでにほかのバインドがある場合）または **Bind...** を押します。
2. 「Variable」に BOOL（例：`gCanHome`）を入力し、「How」で **Output: the widget enabled (TRUE) or greyed out (FALSE)** を選んで OK を押します。
3. プログラムに `gCanHome := NOT gRunning;`（または同等のラダー図）を書きます。同じ BOOL を複数のウィジェットにバインドできます。

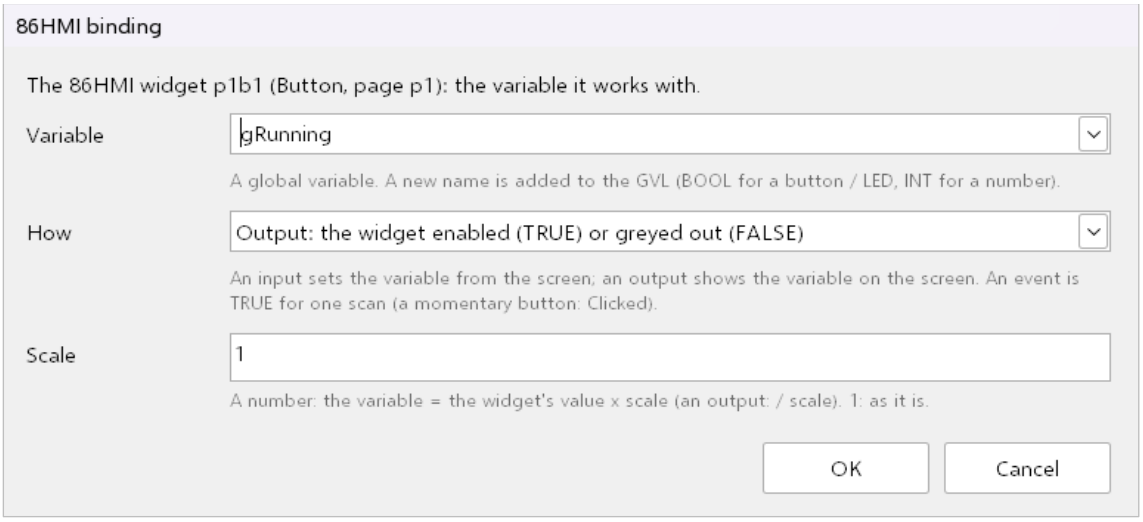


図 13-13　有効化のバインド：BOOL が FALSE のときウィジェットがグレー表示になる

13.4　テキスト、トレンド、アラーム、ページ切り替え

13.4.1　テキスト：Label、Table、TextList

テキストウィジェットの「How」は「**Output: the value shown as text**」です。「Text」欄には 2 通りの書き方があります。

書き方	例	画面の表示
テンプレート：`%d` は数値を表します	温度 `%d C`	温度 25 C

数値ごとに 1 つのメッセージを 1 行に 1 つずつ	`0=停止`
	`1=運転`
	`2=アラーム`
		変数が 1 のとき「運転」と表示

- **Table**：さらに「Column, row」でセルを指定します。セルごとに 1 つのバインドです。
- **TextList**：さらに「line」を指定します。実機のみ：86HMI simulator には TextList の API がないため、シミュレーション中は更新されません。
- 「Enable (a BOOL, optional)」：空欄 = 常に更新。BOOL を指定すると、それが TRUE のときだけテキストを更新します。

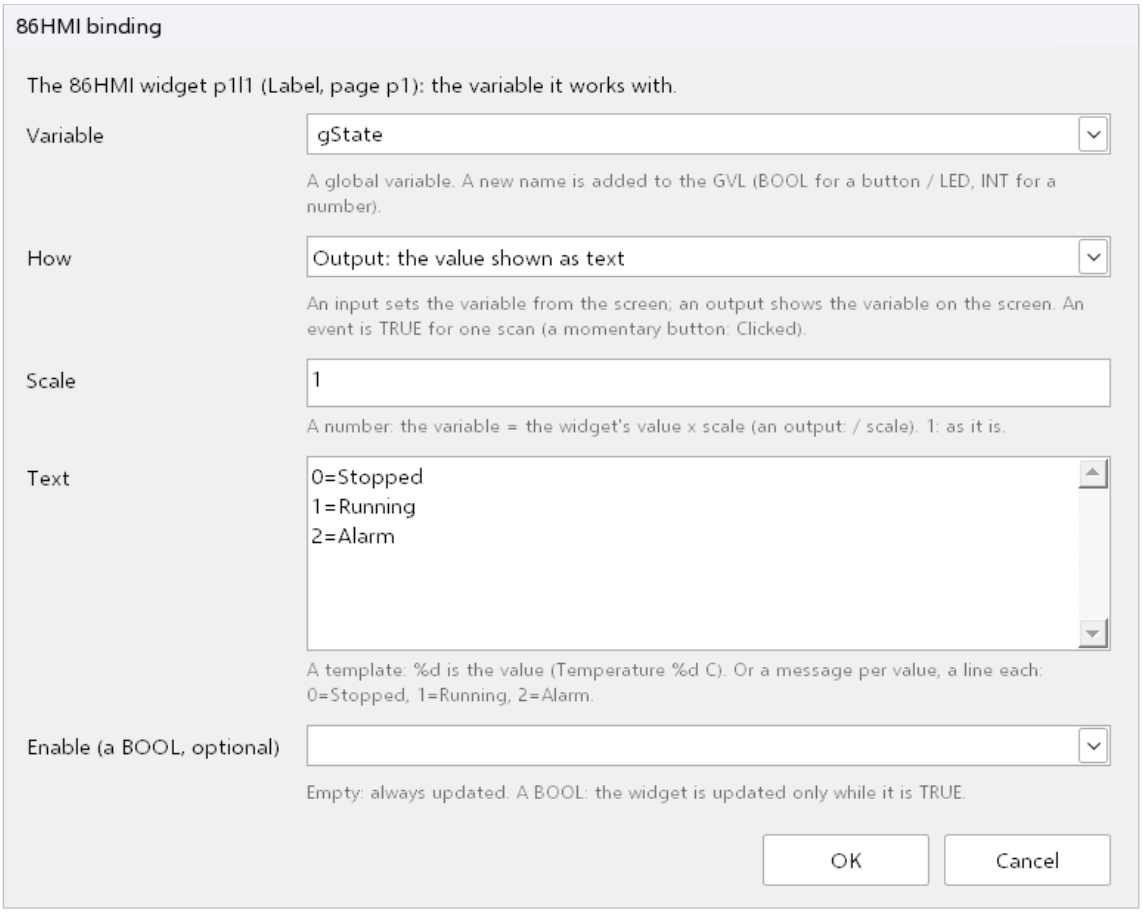


図 13-14　Label のテキストバインド：数値ごとに 1 つのメッセージ

13.4.2　トレンド：Chart、Oscilloscope

「How」は「**Output: a trend (the variable sampled into a channel)**」です。「Channel (1..4), period (ms)」に例えば `1, 100` と指定すると、100 ms ごとに変数をチャンネル 1 にサンプリングします。1 つの Chart は最大 4 チャンネルで、チャンネルごとに 1 つのバインドです（2 つ目のチャンネルは **Add a binding...** で追加します）。フォームの上部にはこの Chart のポイント数とチャンネル数が表示されます。

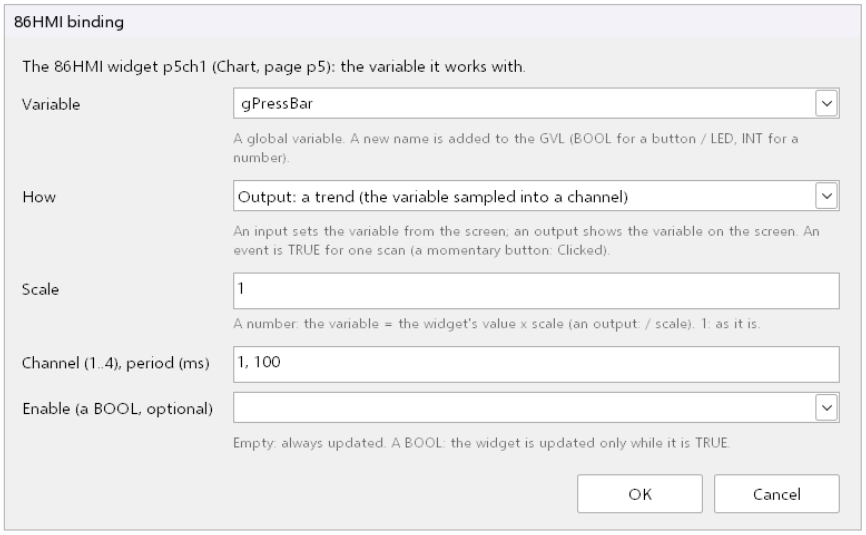


図 13-15　Chart のトレンドバインド：チャンネルとサンプリング周期

13.4.3　アラームバナー

86HMI ページの **Alarm banner...** を押します（またはプロジェクトツリーの「86HMI」を右クリック）。BOOL 変数が TRUE の間バナーが表示され、FALSE になると消えます。

フィールド	説明
Variable (a BOOL; empty: remove)	バナーを制御する変数。新しい名前は BOOL として GVL に追加されます
Bank (0..15)	バナーの位置：位置が異なるバナーは同時に表示できます
Text	バナーのメッセージ（例：「非常停止が押されました」）
Enable (a BOOL, optional)	空欄 = 常に有効

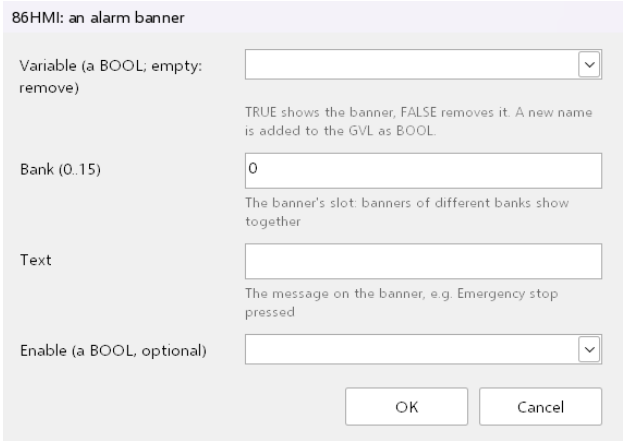


図 13-16　「86HMI: an alarm banner」：Variable、Bank、Text

13.4.4　ページ切り替え：ページ変数

Page variable... を押します。整数変数で現在表示するページを決め（1 = p1、2 = p2 …）、PLC から能動的に画面を切り替えられます（例：アラーム発生時にアラームページへ切り替える）。新しい名前は INT として GVL に追加されます。

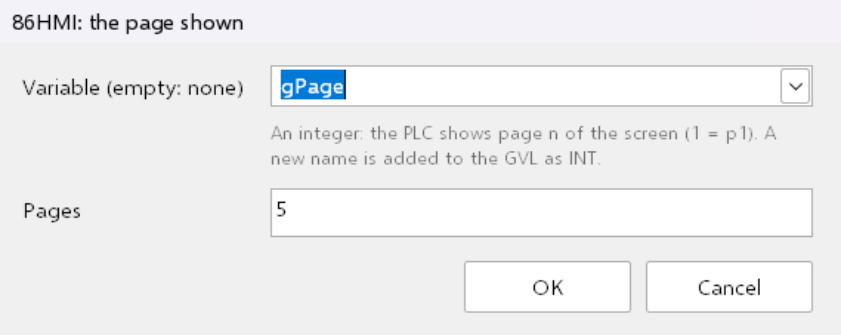


図 13-17　「86HMI: the page shown」：ページ変数とページ数

*ヒント：通常のページ切り替えには、86HMI Editor のボタンスクリプト「Page Change」を直接使うことをお勧めします（PLC を経由しません）。「Page variable」は PLC から能動的にページを切り替えるためのものです。切り替え後は変数を 0 に戻してください。そうしないと次回切り替えられません。

13.4.5　変数からのバインド

ラダー要素（またはウォッチの行、ST 内の変数）を右クリックし、「**86HMI binding of <変数>...**」を選びます。一覧からウィジェット（後ろに現在のバインドが表示されます）、「Alarm banner」または「The page shown」を選び、OK を押してから「How」を指定します。「(none: remove the 86HMI bindings of this variable)」を選ぶと、その変数の 86HMI バインドがすべて削除されます。

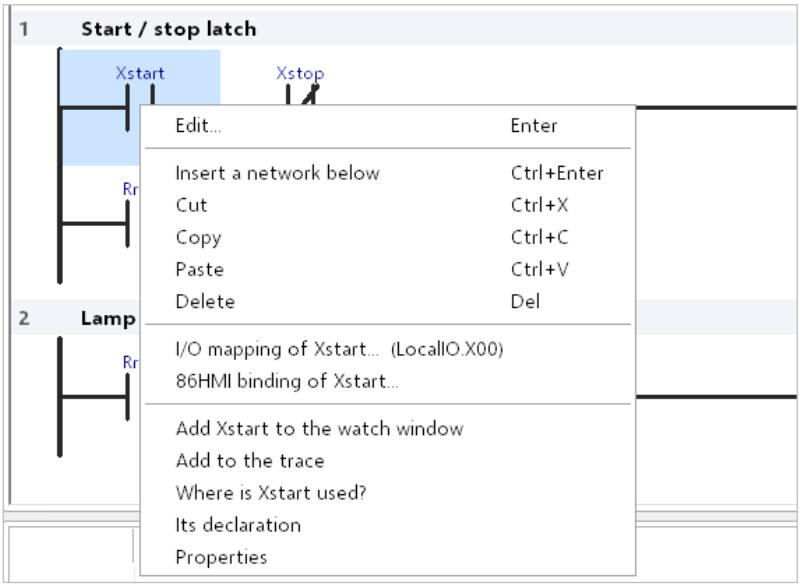


図 13-18　ラダー要素の右クリックメニュー：「86HMI binding of Rstop...」

*ヒント：86HMI 画面の更新は最初のタスクのスキャンの後に行われ、スキャン時間とウォッチドッグには含まれません（ラベルテキストのレイアウトやページ切り替えには数十 ms かかることがあります）。

13.5　シミュレーションとオンライン

シミュレーション（F5）時、sketch に 86HMI 画面があれば **86HMI シミュレーターウィンドウ**が自動的に開きます。パソコン上で直接ボタンを押したりスイッチを切り替えたりすると、PLC プログラムが即座に反応します。LED、数値、テキスト、画像の位置、アラーム、ページ、トレンドには PLC の値が表示されます。オンライン（Ctrl+F5）時は、シミュレーターウィンドウにコントローラー上の値が表示されます。



図 13-19　86HMI シミュレーターウィンドウ：パソコン上で直接画面を操作すると PLC プログラムが即座に反応する（例：加圧ステーションの監視）

13.5.1　図解：シミュレーション中に 86HMI を操作する

1. 86HMI ページの「**The simulator window (F5)**」がオンになっていることを確認します。

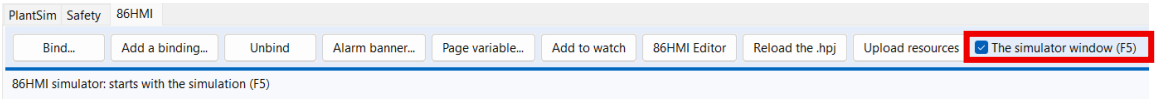


図 13-20-1　「The simulator window (F5)」がオンになっている

2. **Simulate**（F5）を押すと、シミュレーターウィンドウが自動的に開きます。86HMI ページの上部にシミュレーターウィンドウの接続状態が表示されます。

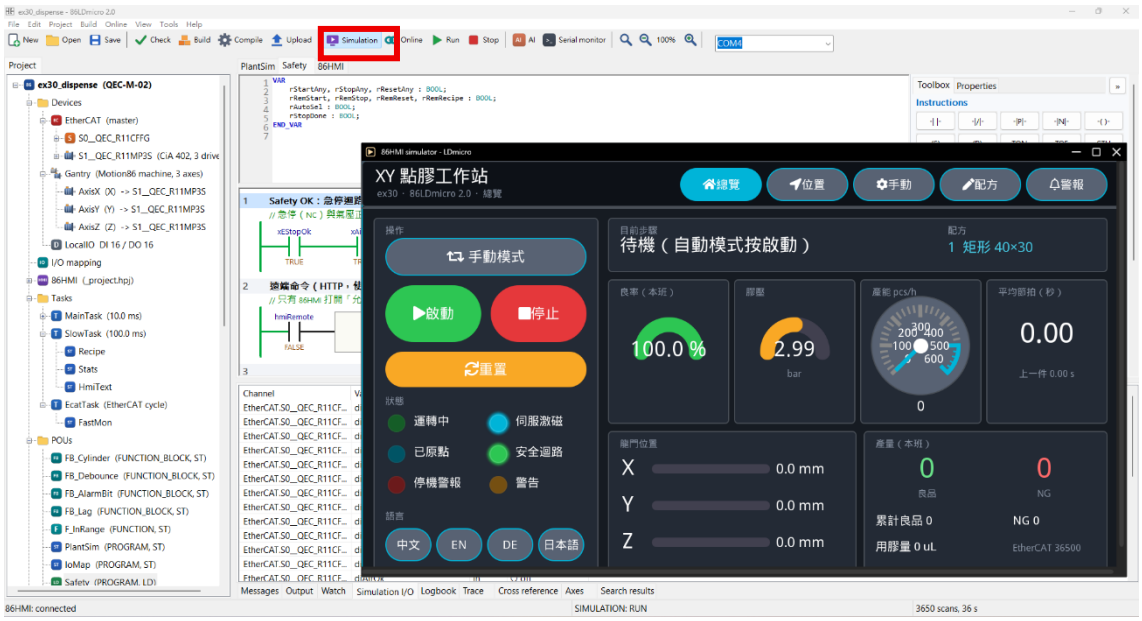


図 13-20-2　シミュレーション中：86LDmicro のラダー図と 86HMI シミュレーターウィンドウを並べて表示

3. 画面上のボタンを押し続けると、バインドされた変数が TRUE になり、ラダー図の対応する接点が緑色になります。86HMI ページの「Value」列も同時に更新されます。
4. PLC の出力変数が変化すると、バインドされた LED、数値、画像も同時に更新されます。
5. もう一度 F5 を押すとシミュレーションが終了し、シミュレーターウィンドウも一緒に閉じます。

やりたいこと	操作
シミュレーターウィンドウを閉じる（自動的に開かない）	86HMI ページの「The simulator window (F5)」をオフにする
シミュレーション中にシミュレーターウィンドウを再度開く	「Online > 86HMI simulator window (the sketch's .hpj)」
バインドを変更した	バインドは次にシミュレーション（F5）を開始したときに有効になります。先にシミュレーションを終了してから F5 を押してください
オンライン時にコントローラーの値を見る	Ctrl+F5 で接続し、シミュレーターウィンドウを開く

*注意：**既知の問題**（2.0.0）：86HMI のシミュレーションでは、ToggleButton の状態は正しく切り替わりますが、ボタン上のテキストが「押されていない」と「押されている」の 2 組のテキストの間で切り替わりません（実機では 86HMI が生成したプログラムがテキストを更新します）。PLC 変数には影響しません。

13.6　リソースのアップロードと言語

13.6.1　86HMI リソースのアップロード

86HMI Editor で「Code Generation」を実行すると、画面で使う画像、フォント、画面データが sketch の `data\Resource` に書き出されます。これらの**リソース**はプログラムとは別にコントローラー上に保存されます（一般的なアップロードの説明は 10.6 節を参照）。

状況	方法
初回のアップロード、またはプログラムと画面の両方を変更した	**Upload**（F8）を押す：プログラムと `data\Resource` が一緒にアップロードされます
画像、フォント、画面の外観だけを変更し、PLC プログラムは変更していない	**Upload resources** を押す：リソースだけを送り、コンパイルはしません。コントローラー上のプログラムは変わりません（F8 より高速）

1. ツールバーで COM ポート（例：COM4）を確認します。
2. 86HMI ページで **Upload resources** を押すか、「**Build > Upload the 86HMI resources (data\Resource)**」を使います。

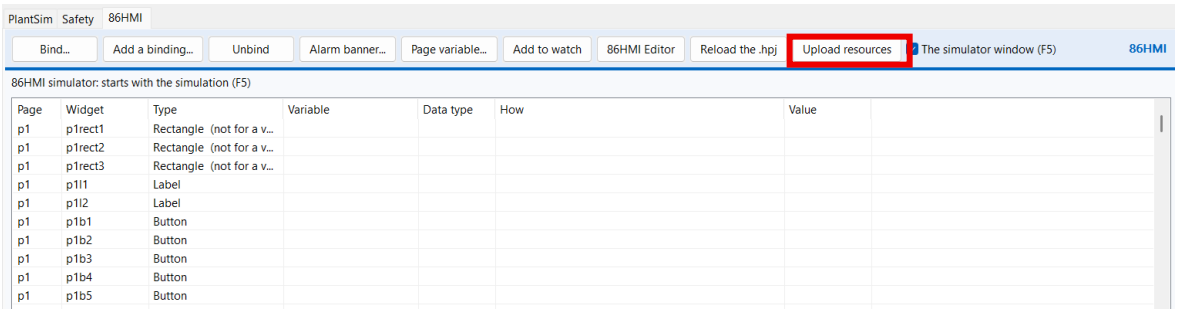


図 13-21-1　86HMI リソースの Upload resources ボタン

3. 「Output」ウィンドウに「== Uploading the resources to COM4: …\data\Resource ...」と進行状況が表示され、最後に「Resources uploaded」と表示されます。
4. オンライン中だった場合は、アップロード前に自動的にオフラインになり、完了後に自動的に再接続されます。

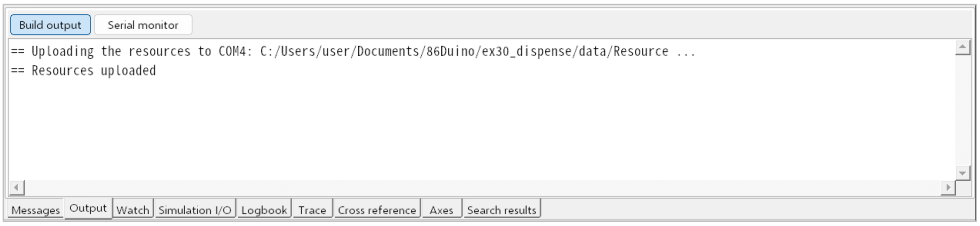


図 13-21-2　86HMI リソースのアップロード：「Output」ウィンドウの経過と「Resources uploaded」

*ヒント：この機能は 86Duino IDE の「Tools > Upload Resources to QEC M02」と同じです。コマンドライン：`86ldmicro-cli upload-res <プロジェクト> <COM>`。

「Nothing to upload: … does not exist or is empty」と表示された場合は、sketch に `data\Resource` がないか、空です。まず 86HMI Editor で .hpj を開き、「Code Generation」を押してください。

13.6.2　言語切り替えボタン

86HMI 画面には複数の言語を持たせることができます（86HMI Editor で設定）。オペレーターがコントローラー上でボタンを押して言語を切り替えられるようにするには、**86HMI Editor でボタンに切り替えスクリプトを書く必要があります**。86LDmicro のバインドは変数だけを扱い、言語は切り替えません。

1. 86HMI Editor で言語ボタン（例：「中文」）を選択し、そのイベント（Clicked）にスクリプト `Hmi.setLanguage(1);` を追加します（数字は言語番号で、エディターでの言語の順に 1、2、3 …）。
2. ほかの言語ボタンも順に `Hmi.setLanguage(2);`、`Hmi.setLanguage(3);` … と設定します
3. **Code Generation** を押し、86LDmicro に戻って **Upload**（F8）を押します。



図 13-22　86HMI Editor：言語ボタンの Clicked スクリプト Hmi.setLanguage(1);

***注意：**スクリプトが空のままだと、シミュレーターウィンドウでは正常に見えても、コントローラーにアップロードした後にボタンを押しても言語が切り替わりません。ex30 の例ではすでに正しいスクリプトを使っています。

第 14 章　C/C++ と 86Duino の統合

86LDmicro の PLC プログラムは最終的に 86Duino の C/C++ プログラム（`plc86.cpp`）になり、86Duino IDE のコンパイラーでコンパイルされます。そのため、同じプロジェクトに独自の C/C++ を追加できます。シリアル通信プロトコル、ネットワーク、特殊なアルゴリズム、86Duino／Arduino のライブラリのいずれも、LD、ST、SFC と同じ変数を共有できます。

書く場所	適した用途	本章
ユーザーコードファイル `usercode/*.cpp86`	PLC 変数と密接に関係するコード：通信、換算、EtherCAT 周期ごとの同期制御	14.1〜14.4
sketch の `.ino` とその他の `.cpp`／`.h`	PLC と関係のないコード、または既存の Arduino コード	14.5

14.1　ユーザーコードファイル

「**Project > Add a user code file (C/C++)...**」（またはプロジェクトツリーの「User code (C/C++)」を右クリック）で名前（既定値 `MyCode`）を入力すると、`usercode/<名前>.cpp86` が作成されます。ビルド時にこのファイルは `plc86.cpp` に直接組み込まれ、86Duino／Arduino のすべての API とライブラリを使えます（`#include <...>` は通常どおり書きます）。

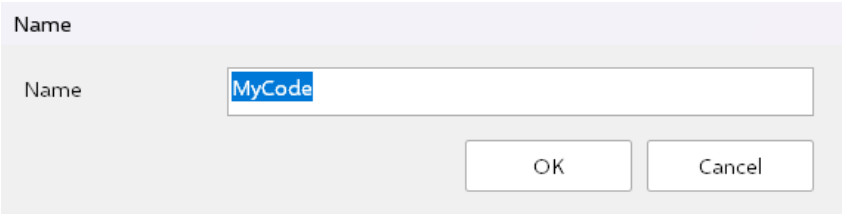


図 14-1　ユーザーコードファイルの追加：名前を入力

新しいファイルにはテンプレートが付いています。コメントで PLC 変数へのアクセス方法を説明し、各フック関数を列挙しています（14.3 節）。テンプレートは GVL の中で `X...` と `Y...` で始まる最初の BOOL 変数を例の名前として使います。プロジェクトに EtherCAT デバイスがない場合、`plc86_user_ethercat()` はコメントとして記載されます。

```
// MyCode: C/C++ of the project, compiled into plc86.cpp with the PLC (Help > C/C++ user code).
//
// The PLC's variables -- not by their bare names, through these (Ctrl+Space lists them):
//   GVL(Xstart)              a global variable (GVL), read and written: GVL(Ylamp) = true;
//   GVL(table[2]), GVL(rec.x)    an array element (the ST indexes), a structure's field
//   ST(Main.count)               a PROGRAM's variable (ST(Main.t1.Q): a function block's output)
//   FUN(Scale)(x, 10)            an ST FUNCTION of the project
//   LD_READ(x), LD_WRITE(y, v)   as in LDmicro 1.1
// The hooks: the PLC calls the ones defined here (delete the ones not needed).
// The USB Serial is the upload and online port: print to Serial1..3, not to Serial.

// in setup(), after the PLC's: begin libraries (Serial1, Ethernet, CAN, Modbus ...)
void plc86_user_setup(void) {
}
```

```
// every scan of the first task, before its programs (the inputs are read)
void plc86_user_before(void) {
}

// every scan of the first task, after its programs (before the outputs are written)
void plc86_user_after(void) {
    // e.g.: GVL(Ylamp) = GVL(Xstart);
}

// loop(), every pass: non-blocking work only (no delay())
void plc86_user_loop(void) {
}

// every EtherCAT cycle (the master's cyclic callback, an interrupt): synchronous control.
// Only with an EtherCAT device (Project > Devices). Short: no delay(), no Serial, no malloc.
// void plc86_user_ethercat(void) {
//     if(!ECAT_OK)
//         return;
// }
```

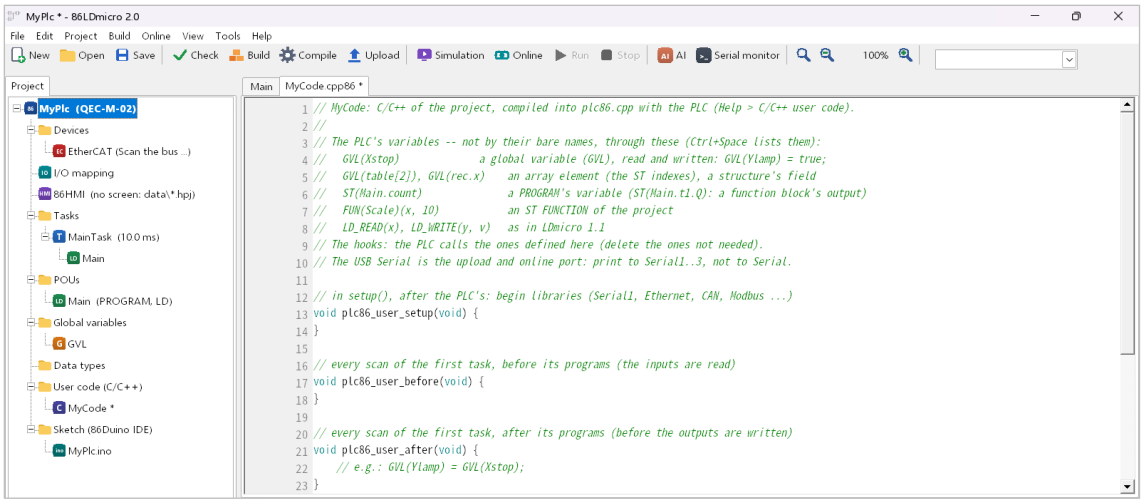


図 14-2　新規ユーザーコード MyCode.cpp86：テンプレートとプロジェクトツリーの「User code (C/C++)」

ユーザーコードエディターは ST エディターと同じく、構文の色分け、Ctrl+T による自動整形、エラーの下線表示があります。さらに 86Duino／Arduino の全 API（約 7,600 項目）の補完とパラメーターヒントを備え、まだ include していないライブラリのオブジェクトやクラスを選ぶと、ファイルの先頭に `#include` を自動的に追加します。

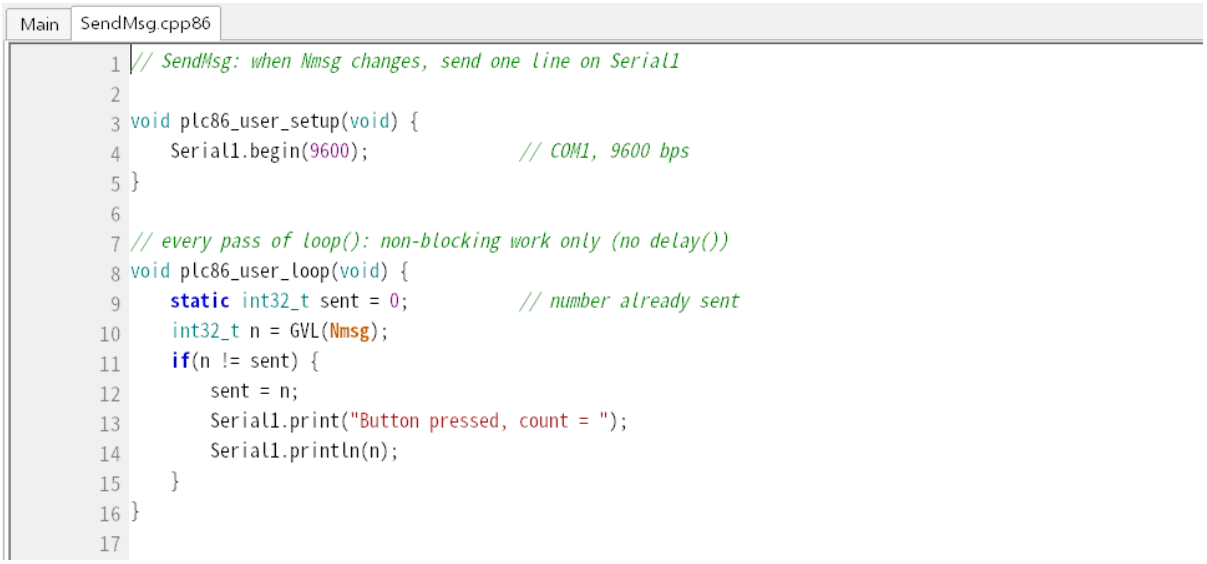


図 14-3　ユーザーコードエディター：フック関数と GVL() による PLC 変数へのアクセス

14.2 PLC 変数へのアクセス

*重要：PLC 変数は**名前をそのまま書くことはできません**（C での実際の名前は `ldst_P_GVL__Xstart` のようなものです）。`GVL(Xstart)` と書きます。名前をそのまま書くと、コンパイルエラーで正しい書き方が示されます。

書き方	意味	例
`GVL(名前)`	グローバル変数（読み書き可）	`if(GVL(Xstart)) GVL(Ylamp) = true;`
`GVL(配列[i])`	グローバル配列の要素（インデックスは ST の範囲）	`ARRAY[1..5]` の 3 番目：`GVL(Recipe[3]) = 100;`
`GVL(構造体.フィールド)`	グローバル構造体のフィールド	`GVL(Axis1.pos) = 12.5f;`
`ST(プログラム.変数)`	PROGRAM のローカル変数、FB インスタンスのメンバー、配列、構造体	`ST(Main.tLamp.Q)`、`ST(Filling.step)`
`FUN(名前)(引数)`	プロジェクトの ST FUNCTION を呼び出します	`float v = FUN(Scale)(raw, 0.0f, 100.0f);`
`LD_READ(x)`、`LD_WRITE(y, v)`、`LD(x)`	LDmicro 1.1 の書き方。`GVL()` と同じ	`LD_WRITE(Ylamp, LD_READ(Xstart));`
`ECAT_MASTER`、`ECAT_OK`、`ECAT_SLAVE(サブデバイス)`	EtherCAT メインデバイスのオブジェクト、メインデバイス起動済み、サブデバイスのドライバーオブジェクト（プロジェクトに EtherCAT デバイスがある場合）	`ECAT_SLAVE(S0__QEC_R11CFFG).digitalWrite(1, HIGH);`
`AXIS(軸名)`、`mc_*`、`M86_MACHINE(マシン)`	モーションコントロール（第 12 章）	`mc_move_abs(AXIS(AxisX), 100.0f, 50.0f);`

名前、フィールド、インデックスはビルド時にチェックされ、エラーは `.cpp86` の行番号を示します（例：`usercode/MyCode.cpp86:11: error: GVL(Xstpo): there is no such global variable`）。「Messages」でその行をダブルクリックすると、該当行にジャンプします。

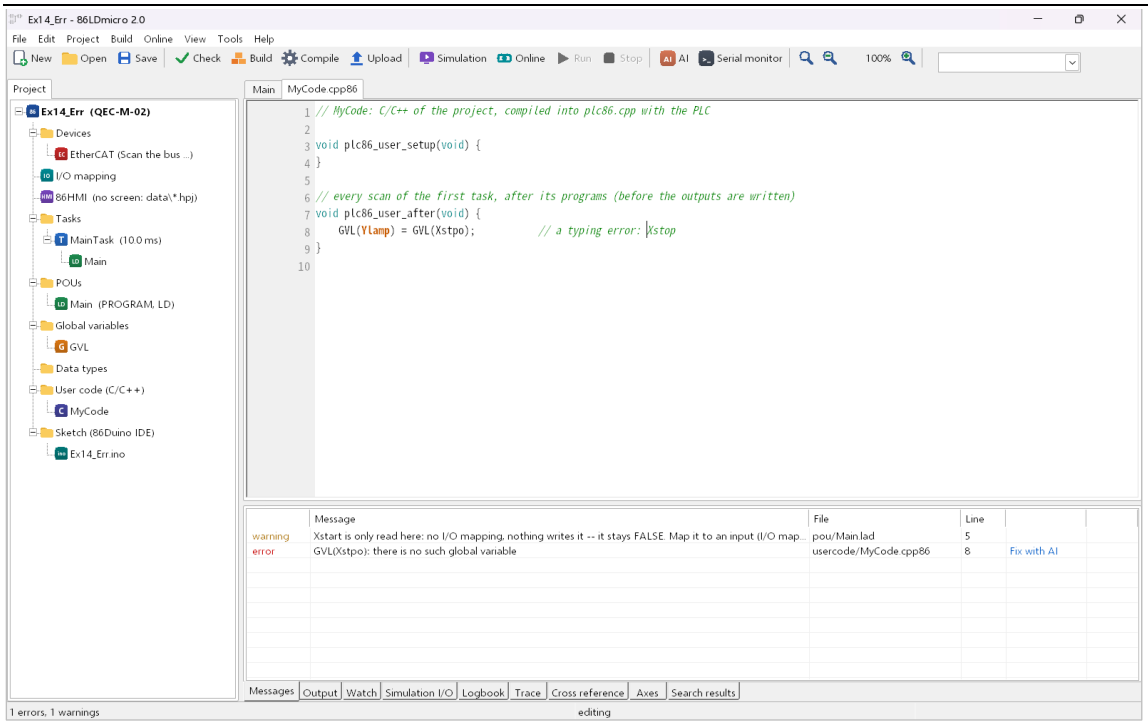


図 14-4　変数名の入力ミス：「Messages」が .cpp86 の行番号を示す

GVL(、ST(、FUN(と入力すると使用できる名前（型とコメント付き）が一覧表示され、(の後には FUNCTION のパラメーターが表示されます。名前にマウスを重ねると型が表示され、シミュレーション中やオンライン中は値も表示されます。

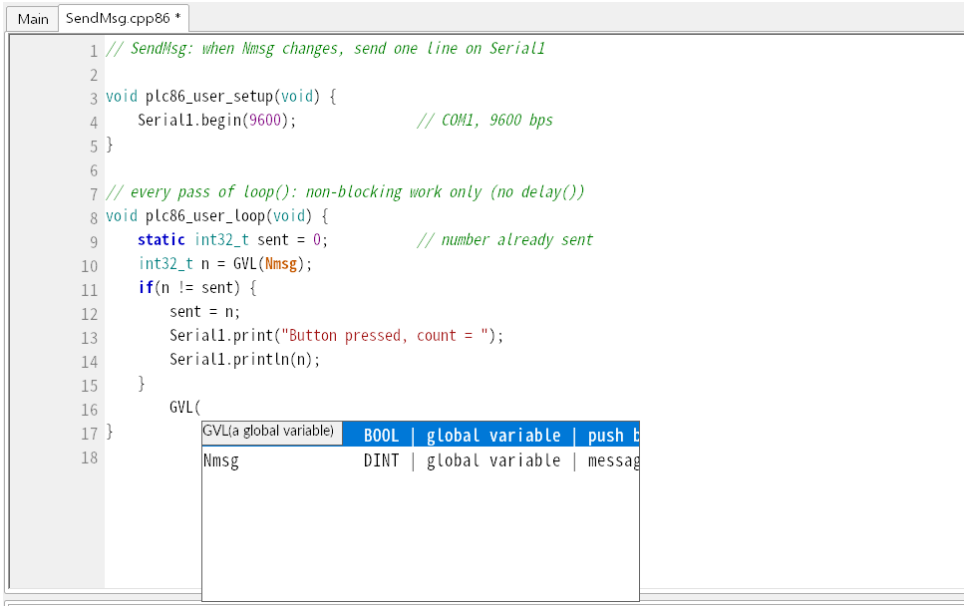


図 14-5　GVL(と入力するとグローバル変数が一覧表示される（型とコメント付き）

14.2.1 型の対応

PLC の型	C の型	PLC の型	C の型
BOOL	`bool`	REAL	`float`
SINT／USINT	`int8_t`／`uint8_t`	LREAL	`double`
INT／UINT	`int16_t`／`uint16_t`	TIME	`int32_t`（ミリ秒）
DINT／UDINT	`int32_t`／`uint32_t`	列挙型	`int32_t`
LINT／ULINT	`int64_t`／`uint64_t`	STRING[n]	`char[n+1]`（`STRING` は `char[81]`）
BYTE／WORD	`uint8_t`／`uint16_t`	DWORD／LWORD	`uint32_t`／`uint64_t`

*ヒント：STRING は C では 0 終端の文字配列です。読み取りはそのまま const char * として使います。書き込みは strncpy(GVL(Msg), "text", sizeof(GVL(Msg)) - 1); GVL(Msg)[sizeof(GVL(Msg)) - 1] = '\0'; を使い、宣告した長さを超えず、必ず 0 で終端されるようにします。

14.3　フック関数

フック関数（hook）は**定義されていれば呼び出されます**。使わないものは削除してかまいません。フック関数を使うと、PLC 実行中の決まったタイミングで独自の C/C++ を実行できます。

関数	呼び出しタイミング	適した用途
`plc86_user_setup()`	`setup()` 内、PLC の初期化の後	`Serial1.begin()`、Ethernet、CAN、Modbus、SCoop などの初期化
`plc86_user_before()`	最初のタスクの毎スキャン：入力の読み込みの後、プログラム実行の前	外部データを PLC に渡す（入力の前処理）
`plc86_user_after()`	最初のタスクの毎スキャン：プログラム実行の後、出力の書き出しの前	PLC の結果を読む（出力の後処理、状態の報告）
`plc86_user_loop()`	`loop()` の 1 周ごと（スキャンとスキャンの間）	ノンブロッキング通信。`delay()` は使わない
`plc86_user_ethercat()`	EtherCAT 周期ごと（メインデバイスの周期コールバック、割り込み内）	同期制御。短く保つこと
`plc86_user_ecat_config()`	EtherCAT サブデバイスの attach の後、start の前	SDO の初期パラメーター、メインデバイスの設定

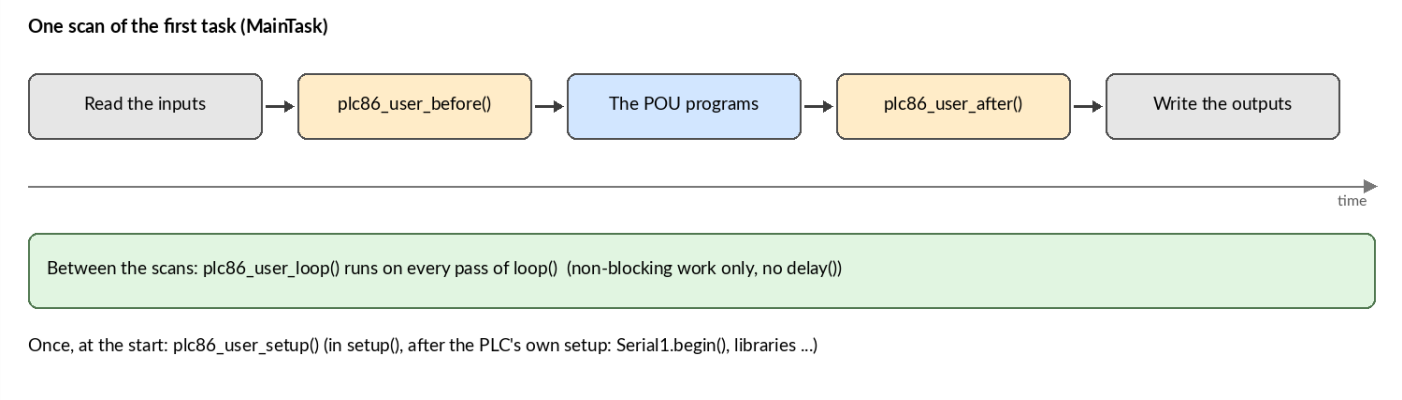


図 14-6　1 スキャン内のフック関数の位置：入力の読み込み → before → プログラム → after → 出力の書き出し。loop はスキャンの間

- `plc86_user_ethercat()`：プロジェクトに EtherCAT デバイスがあれば使えます。EtherCAT タスクを作成する必要はありません。EtherCAT タスクがある場合は、そのタスクのプログラムの後に呼び出されます。`plc86_setup()` が完了するまでは呼び出されません。
- `plc86_user_ecat_config()` はテンプレートに含まれていません。必要なときに自分で追加します（ユーザーコードと `.ino` のどちらに書いてもかまいません）。
- 補完リストから、これらの関数の空の定義を直接挿入できます。

14.4 例

14.4.1 Serial1：コマンドを受信したら起動し、状態が変わったら報告する

シナリオ：外部機器がコントローラーの COM1（RS-232／485）経由で文字 S（起動）、T（停止）を送信し、PLC の状態が変わったら RUN／STOP を返信します。実機のボタンでも引き続き操作できます。（Serial1 でメッセージを送信する入門例は 6.7 節を参照してください。）

変数	型	I/O 割り当て	用途
Xstart	BOOL	LocalIO.X00	実機の起動ボタン
Xstop	BOOL	LocalIO.X01	実機の停止ボタン
Rstart	BOOL	（なし）	Serial1 からの起動コマンド（1 スキャン）
Rstop	BOOL	（なし）	Serial1 からの停止コマンド（1 スキャン）
Rrun	BOOL	（なし）	運転中（自己保持）
Yrun	BOOL	LocalIO.Y00	運転表示灯

ラダー図（Main）：

```
NETWORK 'start/stop'
    [ NO(Xstart) | NO(Rstart) | NO(Rrun) ] NC(Xstop) NC(Rstop) OUT(Rrun)
NETWORK 'lamp'
    NO(Rrun) OUT(Yrun)
```

ユーザーコード（MyCode.cpp86）：

```
static bool cmdStart = false, cmdStop = false;

void plc86_user_setup(void)
{
    Serial1.begin(9600);                    // コントローラーの COM1
}

void plc86_user_loop(void)                  // loop() の 1 周ごと：delay() は使わない
{
    while(Serial1.available()) {            // 受信済みの文字だけを読む
        char c = Serial1.read();
        if(c == 'S') cmdStart = true;
        if(c == 'T') cmdStop = true;
    }
}

void plc86_user_before(void)                // プログラム実行前：コマンドを 1 スキャンの間 PLC に渡す
{
    GVL(Rstart) = cmdStart;
    GVL(Rstop) = cmdStop;
    cmdStart = cmdStop = false;
}

void plc86_user_after(void)                 // プログラム実行後：状態が変わったら報告する
{
    static bool last = false;
    if(GVL(Rrun) != last) {
        last = GVL(Rrun);
```

```
        Serial1.println(last ? "RUN" : "STOP");
    }
}
```

#	操作
1	上の表の変数を GVL で宣言し、4.6 節に従って Xstart、Xstop、Yrun を本体の I/O に割り当てます
2	Main でテキストモード（Ctrl+T）にしてラダー図を貼り付け、「Back to the diagram」を押します
3	「Project > Add a user code file (C/C++)...」で名前を MyCode にして、コードを貼り付けます
4	**Check**（Shift+F7）を押し、続いて **Upload**（F8）を押します
5	COM1 をパソコンの USB-RS-232／485 変換器に接続し、「Serial monitor」でそのポートと 9600 を選んで **Connect** を押し、S を送信します

期待される結果：S を送信すると Rrun、Yrun が TRUE になり、シリアルモニターが RUN を受信します。T を送信するか実機の停止ボタンを押すと、STOP を受信します。Rstart／Rstop は 1 スキャンの間だけ TRUE なので、「張り付いたまま」にはなりません。

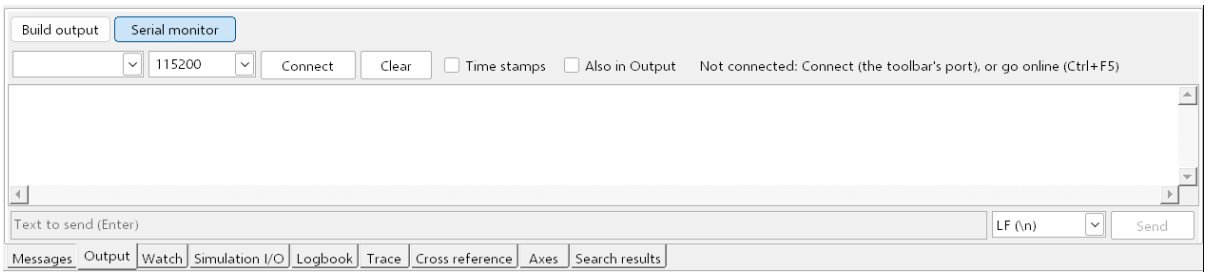


図 14-7　シリアルモニター：ポート、ボーレート、接続、タイムスタンプ、送信

14.4.2 HTTP ステータスページ（構想）

86Duino の Ethernet ライブラリで最も簡単な HTTP サーバーを作ります。ブラウザーで `http://192.168.1.50/` を開くと PLC 変数の JSON が表示されます。以下はその方法の例です。IP アドレスなどは現場のネットワークに合わせて変更してください。

```
#include <Ethernet.h>

static byte mac[] = {0x02, 0x86, 0x00, 0x00, 0x00, 0x01};
static IPAddress ip(192, 168, 1, 50);
static EthernetServer web(80);

void plc86_user_setup(void)
{
    Ethernet.begin(mac, ip);
    web.begin();
}

void plc86_user_loop(void)                  // 待たない：接続がなければすぐに戻る
{
    EthernetClient client = web.available();
    if(!client)
        return;
    while(client.available())               // リクエストを読み捨てる（ここでは解析しない）
        client.read();
    client.println("HTTP/1.1 200 OK");
    client.println("Content-Type: application/json");
    client.println("Connection: close");
    client.println();
    client.print("{\"run\":");
    client.print(GVL(Rrun) ? "true" : "false");
    client.print(",\"level\":");
    client.print(GVL(Level), 1);
    client.print(",\"count\":");
    client.print(GVL(Ncount));
    client.println("}");
    client.stop();
}
```

- 通信は plc86_user_loop() に書きます。この関数はスキャンの間に実行され、PLC タスクには入りません。ただし、かかった時間の分だけ次のスキャンが遅れるので、1 回の処理はごく短くする必要があります（接続やデータを待たないこと）。所要時間は次の項目のとおり別途測定してください。この例は方法を示すものです。ネットワークが混雑していると、print() や client.stop() にも時間がかかることがあります。
- plc86_user_loop() の時間はタスクのスキャン時間に含まれないため、ウォッチドッグにもステータスバーのスキャン時間にも、それによる遅れは現れません。plc86_user_loop() の先頭と末尾で micros() を使って測定し、最大値を GVL 変数（例：GVL(LoopMaxUs)）に書き込んで Watch ウィンドウで確認してください。タスクの周期よりも十分に短くなければなりません。
- レスポンスで読み取る変数（`Rrun`、`Level`、`Ncount`）は GVL で宣言しておく必要があります。Web ページから PLC に**書き込む**場合は、14.4.1 節と同様に、まず C の変数に保存し、`plc86_user_before()` で PLC に渡すことをお勧めします。

14.4.3 EtherCAT 周期ごとの同期出力

```
void plc86_user_ethercat(void)           // EtherCAT 周期ごと（割り込み内）
{
    if(!ECAT_OK) return;
    static uint32_t n = 0;
    n++;
    GVL(Ncycle) = (int32_t)n;                // LD／ST から参照するため
    // DQ01 はこのコードだけが書き込む（変数にも割り当てず、ほかのコードからも書き込まないこと）
    bool on = GVL(Rrun) && ((n / 500) & 1);       // 500 周期ごとに反転：周期 1 ms なら 0.5 秒ごと（点灯と消灯で約 1 秒）
    ECAT_SLAVE(S0__QEC_R11CFFG).digitalWrite(1, on ? HIGH : LOW);   // Rrun が FALSE のときは LOW を書く
}

void plc86_user_ecat_config(void)         // attach の後、start の前
{
    // ECAT_SLAVE(S0__QEC_R11CFFG).sdoDownload8(index, subindex, value);   // 例：SDO の設定。オブジェクトと有効な値はモジュールのオブジェクトディクショナリに従って入力する
}
```

サブデバイス名（例：S0__QEC_R11CFFG）は、プロジェクトツリーの「Devices > EtherCAT」の下の名前と同じです（第 11 章）。DMP の QEC モジュールはそれぞれのドライバークラス（例：EthercatDevice_QECR11CFFG）を使い、digitalWrite、digitalRead、analogRead などを呼び出せます。CiA 402 ドライブは EthercatDevice_CiA402、その他のサブデバイスは EthercatDevice_Generic（pdoGetInputBuffer()／pdoGetOutputBuffer()、SDO）です。

14.5　sketch の .ino

sketch の `.ino` に必要なのは 3 行だけで（ビルド時に自動的に追加されます）、それ以外には自分のコードを書けます。

```
#include "plc86.h"
void setup() { plc86_setup(); /* 独自の初期化 */ }
void loop()  { plc86_loop();  /* 独自のコード。delay() で長く止めないこと */ }
```

- EtherCAT メインデバイス、シリアルポート、86HMI、Modbus、オンラインモニターはすべて `plc86_setup()` がプロジェクトのデバイスに応じて起動するため、`.ino` で `EVA.begin()` や `EthercatMaster::begin()` を改めて呼び出す必要はなく、呼び出してもいけません。86HMI を使う場合は、ビルド時に `#include "myhmi.h"` と `Hmi.begin();` が追加されます。
- `.ino` と sketch の `.cpp` では、`#include "plc86.h"` の後に `GVL(名前)`、`LD_READ`、`LD_WRITE` で GVL の基本型の変数（BOOL、整数、REAL、TIME）にアクセスでき、EtherCAT オブジェクトと `mc_*` も使えます。配列、構造体、`ST()`、`FUN()`、`ECAT_SLAVE()`、`M86_MACHINE()` はユーザーコード（.cpp86）でのみ使用できます。
- プロジェクトツリーの「Sketch (86Duino IDE)」または「Project > The sketch's .ino (the 86Duino IDE's)」で、`.ino` とその他の `.cpp`／`.h` を 86LDmicro の中で直接編集できます。保存、ビルド、コンパイルの前に一緒に保存され、ビルドで `.ino` が修正されるとタブが自動的に更新されます。

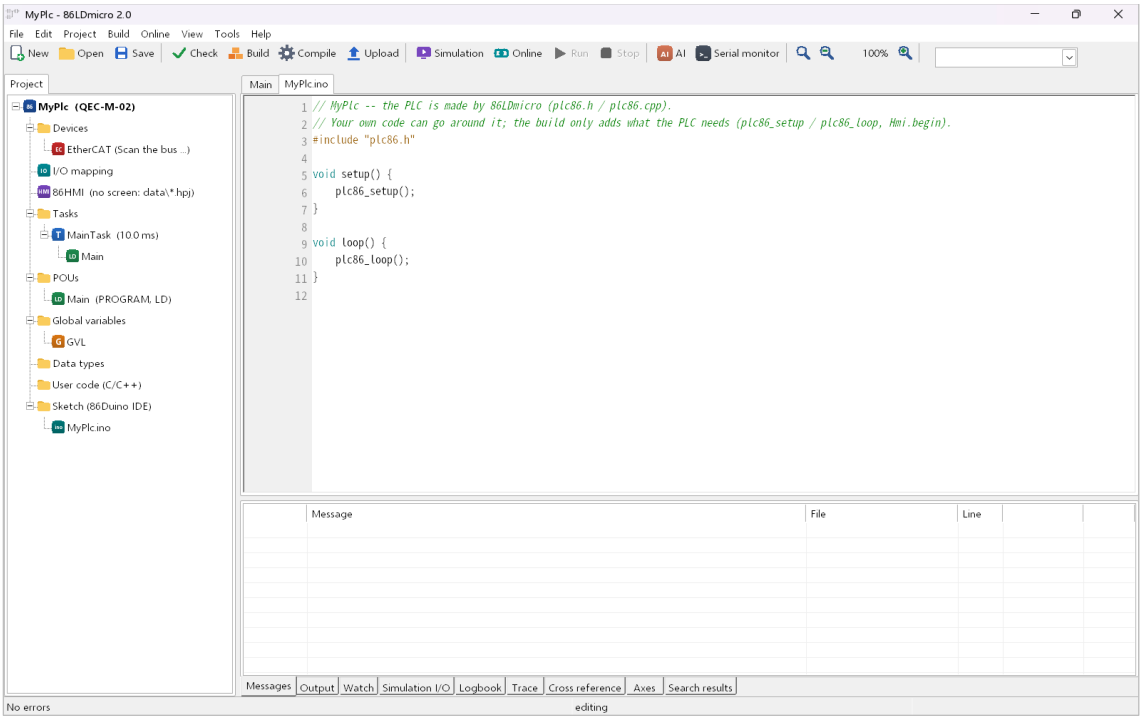


図 14-8　86LDmicro で sketch の .ino を編集する

ビルド後、sketch には次のファイルが追加されます（ビルドのたびに書き直されるので、**手で編集しないでください**）。

ファイル	内容
`plc86.h`／`plc86.cpp`	PLC プログラム：変数、POU、タスク、I/O 割り当て、86HMI バインド、ユーザーコード
`plc86_rt.h`／`plc86_rt.cpp`	ランタイム（全プロジェクト共通）：オンラインモニター（読み書き、強制、RUN／STOP）、RETAIN、Logbook
`plc86_ecat.h`／`plc86_ecat.cpp`	EtherCAT メインデバイスとサブデバイスのオブジェクト、起動手順（`begin` → 各サブデバイスの `attach` → `plc86_user_ecat_config()` → `start`）

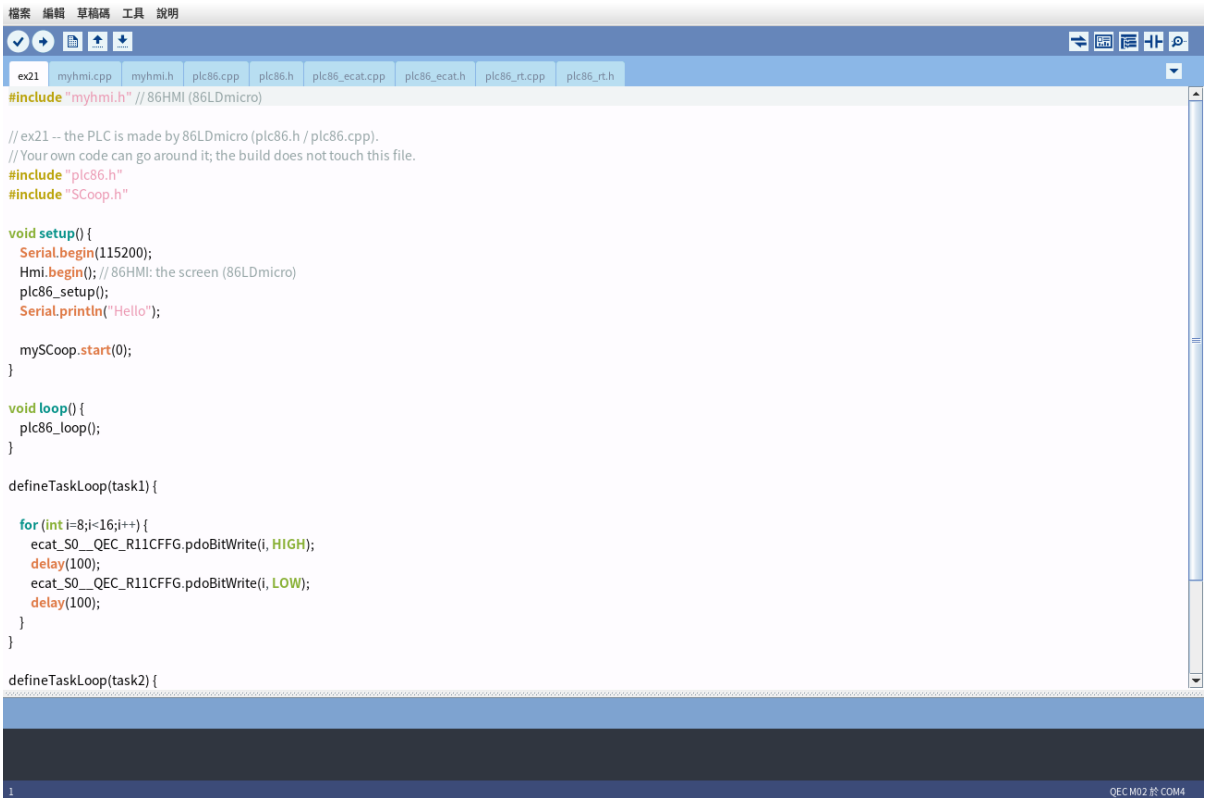


図 14-9　ビルド後の 86Duino IDE：plc86.cpp、plc86.h、plc86_ecat.cpp などのタブ

14.6 注意事項

注意	理由と対処
USB の `Serial` に出力し続けない	USB の `Serial` はアップロードとオンラインモニターに使われます。パソコン側が読み取っていないとき（86LDmicro が未接続、シリアルモニターが閉じている）、`Serial.print()` はバッファーがいっぱいになると待機し、**プログラム全体（PLC スキャンを含む）がそれに伴って停止します**。外部機器との通信には `Serial1`〜`Serial3` を使ってください。デバッグ出力は、先に `Serial.availableForWrite()` が十分かどうかを判定してから出力します
`plc86_user_ethercat()` は割り込み内で実行される	`delay()`、`Serial.print()`、動的メモリー確保（malloc／new）は使用不可。先に `ECAT_OK` を確認すること。浮動小数点演算は使えます。処理時間は EtherCAT 周期より十分に短くすること
`plc86_user_loop()` では待たない	`delay()` を使わず、`while` で外部の応答を待たないこと。1 回に処理するのは受信済みのデータだけにします
`plc86*` ファイルを手で編集しない	ビルドのたびに書き直されます。EtherCAT の起動手順を調整するには `plc86_user_ecat_config()` を使います
実入力に割り当て済みの変数	C から書き込んだ値は、次のスキャンで入力を読み込むときに上書きされます。内部変数（例：`Rstart`）に書き込み、プログラムの中で実入力と並列にしてください
C++ 標準ヘッダー	`#include <vector>` などは自動的に Arduino.h の前に配置されます（min／max マクロとの衝突を回避）
Serial の初期化	オンラインモニターが有効な場合はランタイムがすでに `Serial` を開いています。無効な場合は `.ino` の `setup()` または `plc86_user_setup()` で自分で次を実行してください：`Serial.begin(115200);`

*注意：1.1 の `.ino` に `#include "myeva.h"`、`EVA.begin()` が残っている場合は、2.0 の EtherCAT を使うときに削除し、sketch から `myeva.cpp`／`myeva.h` を移動してください（ビルド時に `EVA.begin()` が自動的にコメントアウトされ、警告が出ます。15.1 節を参照）。

第 15 章　インポート、AI、コマンドラインとプロジェクト保護

この章では、いくつかの補助機能について説明します。LDmicro 1.1 や他の IEC 61131-3 ツールのプログラムを 86LDmicro にインポートする方法、AI アシスタントと MCP の使用、コマンドラインツールによるバッチ処理、およびパスワードによるプロジェクトの保護です。

15.1　LDmicro 1.1 プロジェクトのインポート

LDmicro 1.1 のプログラム（`.ld`、通常は sketch の `data\myLD.ld`）は 86LDmicro 2.0 のプロジェクトに変換できます。プロジェクトは同じ sketch の `data\86ld` に書き込まれ、**元の `.ld` は変更されない**ため、必要なときは引き続き 1.1 で開けます。

15.1.1　86Duino IDE からのインポート

1. 86Duino IDE で `data\myLD.ld` を含む sketch を開き、「**Tools > 86LDmicro**」を選択します。
2. 86LDmicro が 1.1 のプログラムを検出し、インポートするかどうかを尋ねるので、**Yes** をクリックします。
3. インポートが完了するとプロジェクトが開きます。1.1 のラダー図は POU に、変数は GVL に、I/O ピンは I/O マッピングになります。

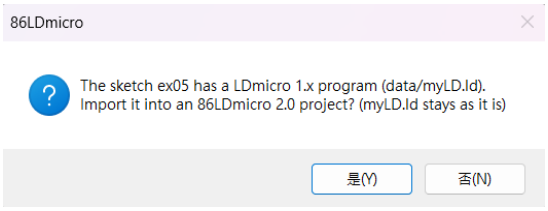


図 15-1　1.1 プロジェクトを検出したときの確認

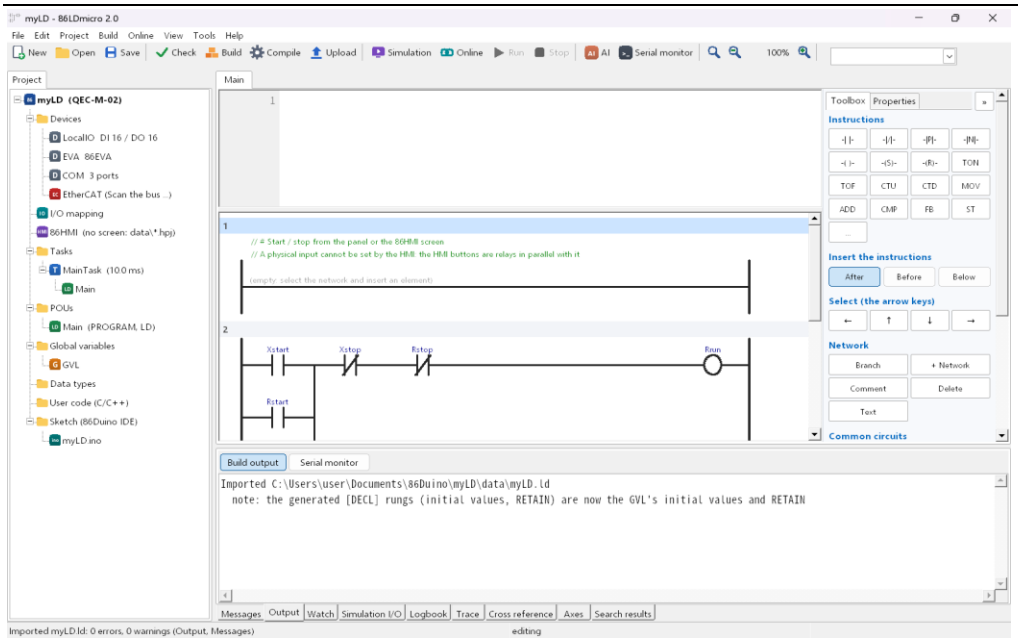


図 15-2　インポート後のプロジェクト：1.1 のラダー図と ST が POU になり、元の順序で MainTask に配置される

15.1.2　86LDmicro からのインポート

1. 「**File > Import a LDmicro 1.x program or a PLCopen XML (.ld, .xml)...**」で `.ld` ファイルを選択するか、`.ld` ファイルを 86LDmicro のウィンドウに直接ドラッグ＆ドロップします（先に「Import … as a new 86LDmicro project?」と確認されます）。
2. 「Import」フォームの「**The sketch folder**」：`.ld` が sketch の `data` フォルダーにある場合はその sketch が既定値になり、そうでない場合は `.ld` の隣にあるファイル名と同名のフォルダーになります。プロジェクトは `<sketch>\data\86ld` に書き込まれます。**OK** をクリックします。
3. ステータスバーに「Imported …: N errors, M warnings (Output, Messages)」と表示され、最初のタスクの最初のプログラムが開きます。
4. `.ino` がすでにある場合は上書きされません。その中の 1.1 の `setupPlc()`／`loopPlc()` を `plc86_setup()`／`plc86_loop()` に変更してください（14.5 節）。その他のコードはそのままです。
5. **Check**（Shift+F7）でエラーがないことを確認し、次に **Simulate**（F5）で動作を検証してから、最後にアップロードします。

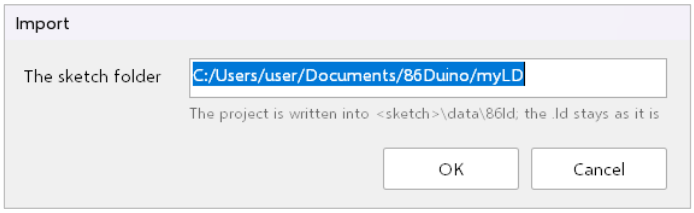


図 15-3　「Import」フォーム：sketch フォルダー（プロジェクトは <sketch>\data\86ld に書き込まれ、元の .ld ファイルは変更されない）

15.1.3 何が何に変換されるか

LDmicro 1.1	86LDmicro 2.0
ラダー図（1 つのプログラム）	`pou/Main.lad`（ST プログラムがすでに Main という名前の場合は `Ladder`）
ST プログラム（POU、before／after、task=ecat）	`pou/*.st`。ラダー図の前後の順序がタスク内の順序になり、EtherCAT 周期は EtherCAT タスクになる
ライブラリの FB、ST ブロック	`pou/*.st` の FUNCTION_BLOCK／FUNCTION
データ型	`dut/*.dut`
VAR LIST、DECLARATIONS（初期値、RETAIN、反転、フィルター、スケーリング）	`gvl/GVL.gvl`（型は 1.1 のサイズ規則に従う）＋ I/O マッピングの反転、フィルター、スケーリング
IO LIST（ピン、Modbus）	I/O マッピング（LocalIO、Pin、Modbus）
BINDINGS（EVA ポイント、サーボ、HMI）	I/O マッピング（EVA.*）とバインド（サーボ、86HMI）
ユーザーコード（global／setup／beforescan／afterscan／loop／ethercat）	`usercode/*.cpp86` のフック関数 `plc86_user_setup`、`_before`、`_after`、`_loop`、`_ethercat`。`LD_READ`／`LD_WRITE` も従来どおり使用可能（14.2 節）
86EVA、COM マッピング、Modbus、Ethernet、Modbus server	デバイス
オンラインモニター	86LDmicro の「Online」（Ctrl+F5、10.4 節）

15.1.4 注意事項

- 1.x のタイマー、カウンターなどの命令は 1.x のセマンティクス（スキャン周期単位でカウント）を保持し、1.1 と同じように動作します。新しいプログラムでは IEC ファンクションブロック（7.8 節）の使用を推奨します。
- IEC の予約語と同じ名前の変数には `_v` が付加されます（例：`TON` → `TON_v`）。`__` で始まる名前は 2.0 の内部用に予約されています。
- 2.0 の EtherCAT（第 11 章）に移行する場合は、`.ino` から 1.x の `#include "myeva.h"` と `EVA.begin()` を、sketch から `myeva.cpp`／`myeva.h` を削除してください（ビルド時に `EVA.begin()` は自動的にコメントアウトされ、通知されます）。
- `MC_ReadActualPosition.Position` は 2.0 では REAL です。DINT に接続していたプログラムは REAL に変更するか、`REAL_TO_DINT` を使用してください。
- コンパイルエラーで AI ウィンドウが自動的に開くことはなくなりました。必要なときは「Messages」で「Fix with AI」をクリックします（15.3 節）。

15.2　PLCopen XML のインポート

PLCopen XML（TC6）は IEC 61131-3 の標準交換フォーマットです。OpenPLC Editor、Beremiz、CODESYS などのツールでエクスポートした `.xml` を、新しい 86LDmicro プロジェクトとしてインポートできます。操作は 15.1.2 節と同じです。

1. 元のツールで PLCopen XML をエクスポートします（例：CODESYS では「Project > Export PLCopenXML」。OpenPLC Editor／Beremiz のプロジェクトファイル `plc.xml` はすでにこの形式です）。
2. 86LDmicro の「**File > Import a LDmicro 1.x program or a PLCopen XML (.ld, .xml)...**」で `.xml` を選択するか、`.xml` を 86LDmicro のウィンドウに直接**ドラッグ＆ドロップ**し、確認されたら **Yes** をクリックします。
3. 「Import」フォームで sketch フォルダーを選択し、**OK** をクリックします。
4. ステータスバーでエラーと警告の数を確認します。「Output」には、インポートされた POU の数（LD、ST、SFC がそれぞれいくつか）と、インポートされなかった部分が表示されます。
5. **Check** と **Simulate** で確認してから、I/O をマッピングし（4.6 節）、アップロードします。

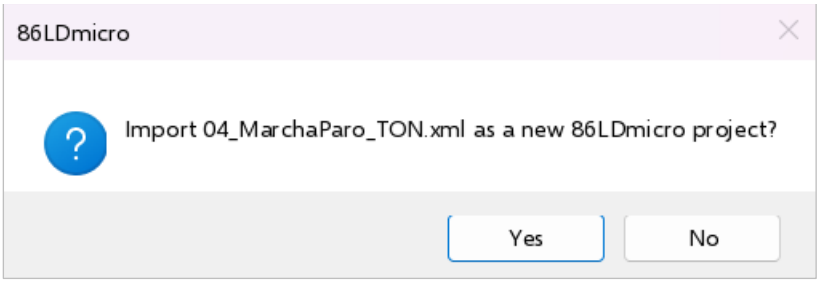


図 15-4　.xml をウィンドウにドラッグ＆ドロップ：新しい 86LDmicro プロジェクトとしてインポートするかの確認

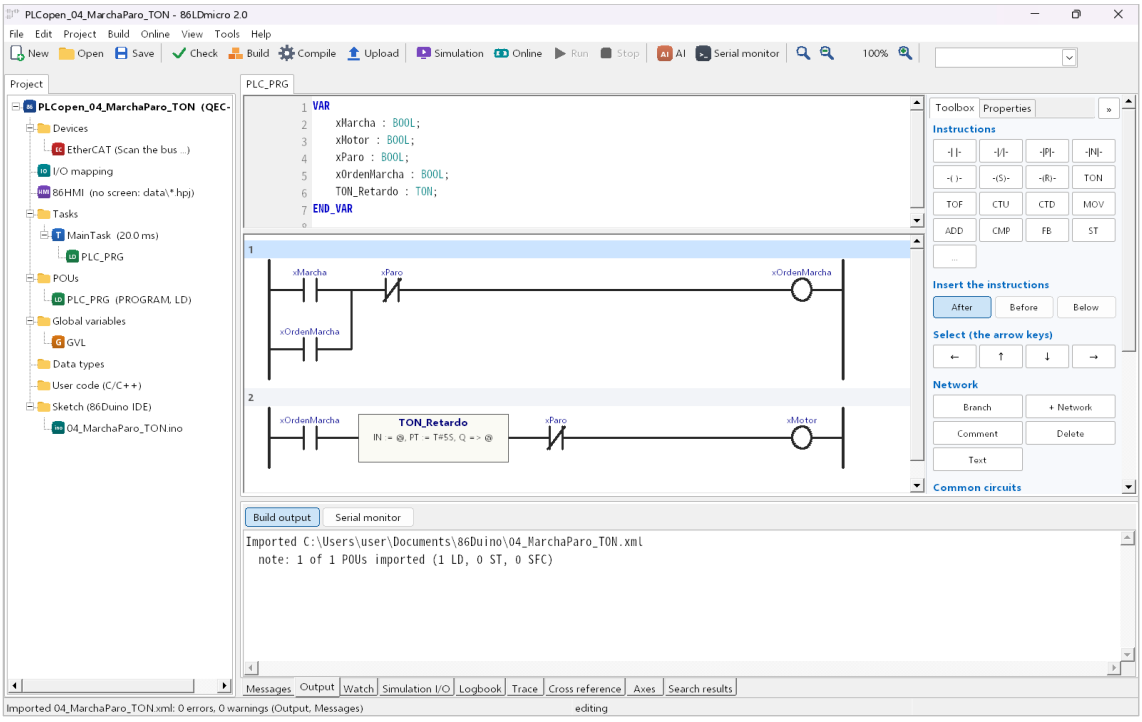


図 15-5　PLCopen XML のインポート後：POU、GVL、および「Output」のインポート概要

PLCopen の内容	インポート結果
LD（ラダー図）	`.lad` プログラム（接点、コイル、ファンクションブロック、分岐）
ST	`.st` プログラム
SFC	`.sfc` プログラム（ステップ、トランジション、アクション、分岐）
データ型（STRUCT、列挙型、配列）	`.dut`
グローバル変数（configuration／resource）	`.gvl`。`AT %IX…` のアドレスはコメントとしてのみ残るため、I/O マッピングで再度リンクする
タスクとプログラムインスタンス	プロジェクトのタスクと割り当て
PROGRAM の VAR_INPUT／VAR_OUTPUT	VAR に変更（86LDmicro の PROGRAM にはパラメーターがない）。メッセージで説明
VAR_EXTERNAL	省略（すべての POU からグローバル変数が見える）。メッセージで説明
FBD、IL	**このバージョンではインポートしない**。メッセージに一覧表示。元のツールで ST または LD に書き直してから再度エクスポートする

コマンドライン：`86ldmicro-cli import <ファイル.xml> <sketch フォルダー> [名前]`。MCP の `ld86_import_11` も PLCopen XML を受け付けます。

15.3　AI アシスタント

AI アシスタントは、ST／LD／C プログラムの作成や修正、エラーの説明、I/O の計画を行えます。**ご自身の API キー**（またはローカルのモデル）を使用し、リクエストはこのコンピューターから設定したサーバーへ直接送信されます。「Send」（または「Test connection」）を押すまで、データは一切送信されません。

15.3.1　AI Assistant タブ

ツールバーの **AI** をクリックするか、「**Tools > AI assistant (MCP)...**」を選択して、「AI Assistant」タブを開きます。

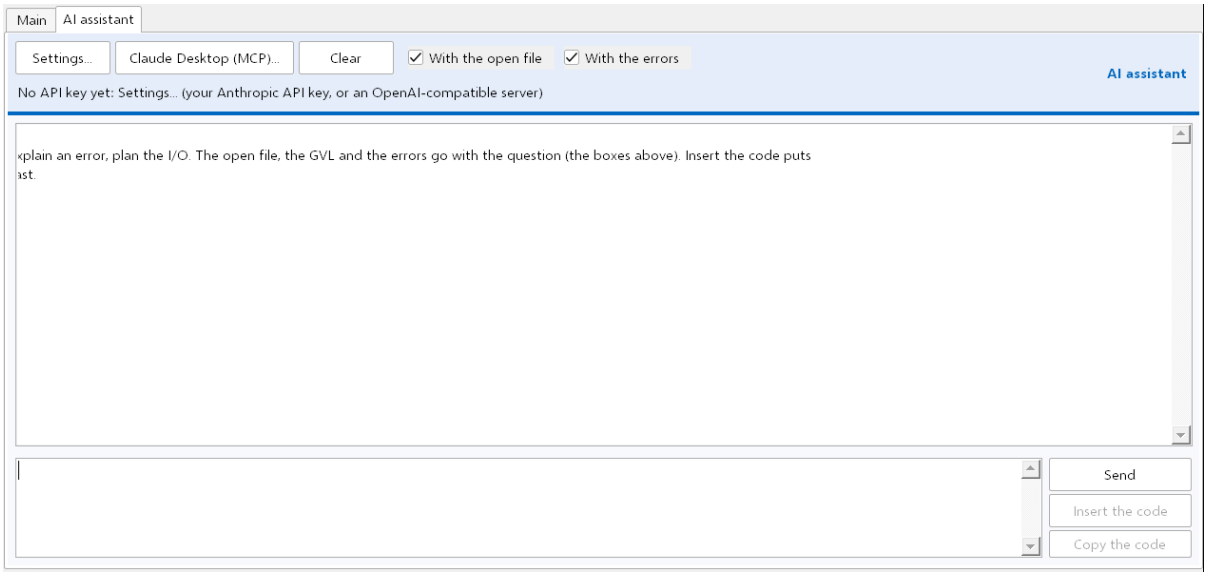


図 15-6　AI Assistant タブ：Settings...、Claude Desktop (MCP)...、Clear、開いているファイルとエラーの添付

項目	用途
Settings...	サービスを選択し、API キーとモデルを入力する（15.3.2 節）
Claude Desktop (MCP)...	MCP を設定する（15.4 節）
Clear	会話をクリアする
With the open file、With the errors（チェックボックス）	チェックすると、質問と一緒に現在開いているファイルと「Messages」のエラーが送信される（GVL とタスクは常に添付される）
Send（Ctrl+Enter）	下に入力した質問を送信する
Insert the code	最後の回答に含まれるコードをエディターに挿入する。ST と C は最後に使用したエディターのカーソル位置に、LD の NETWORK は選択したネットワークの後に挿入される（VAR ブロック内の新しい変数は自動的に宣言される）
Copy the code	回答に含まれるコードをクリップボードにコピーする

15.3.2　設定：サービスとキー

Settings... をクリックして「AI assistant: settings」を開きます。

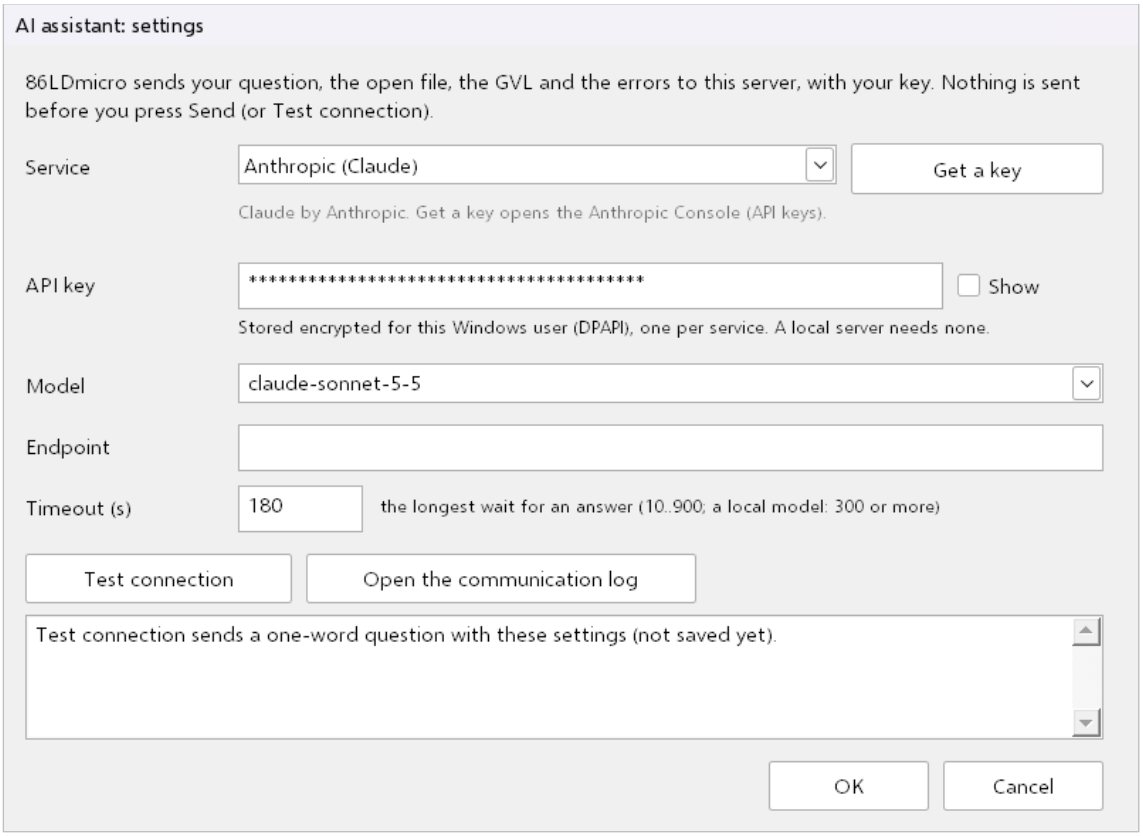


図 15-7　「AI assistant: settings」：Service、Get a key、API key、Model、Endpoint、Timeout (s)、Test connection、Open the communication log

フィールド	説明
Service	Anthropic（Claude）、OpenAI、Google Gemini、Ollama（ローカル）、LM Studio（ローカル）、その他の OpenAI 互換サーバー。選択するとエンドポイントとモデルが自動的に入力される
Get a key（ローカルサービスでは **Download**）	ブラウザーでそのサービスのキーのページ（Anthropic Console、OpenAI、Google AI Studio）またはローカルソフトウェアのダウンロードページを開く
API key	この Windows ユーザー用に暗号化して保存（DPAPI）、サービスごとに 1 つ。ローカルサーバーでは不要。「Show」をチェックすると内容が見える
Model	リストから選択するか、サーバーが表示するモデル名を入力する
Endpoint	空欄：サービスの既定値を使用。Azure、社内ゲートウェイ、vLLM などではサーバーの URL を入力する
Timeout (s)	応答を待つ最大時間（10..900、既定値 180。ローカルモデルでは 300 以上を推奨）

*ヒント：キー、モデル、エンドポイントには ASCII 文字しか使用できません。関連するエラーが出た場合は、全角文字を貼り付けていないか確認してください。

15.3.3　接続テストと通信ログ

1. 設定ウィンドウで **Test connection** をクリックすると、現在の設定（まだ保存されていない）で短い質問を送信し、バックグラウンドで実行します。テスト中はボタンが **Stop** に変わり、これをクリックすると停止できます。
2. 成功すると `OK: <モデル> answered in N ms (...)` と表示されます。失敗した場合は原因が英語で説明されます（下表参照）。
3. **Open the communication log** をクリックすると、各リクエストの URL、ステータス、所要時間、応答の冒頭（キーは含まない）を確認できます。ログファイルは `%LOCALAPPDATA%\86LDmicro\ai_comm.log` です。
4. 問題がなければ **OK** をクリックして保存します。

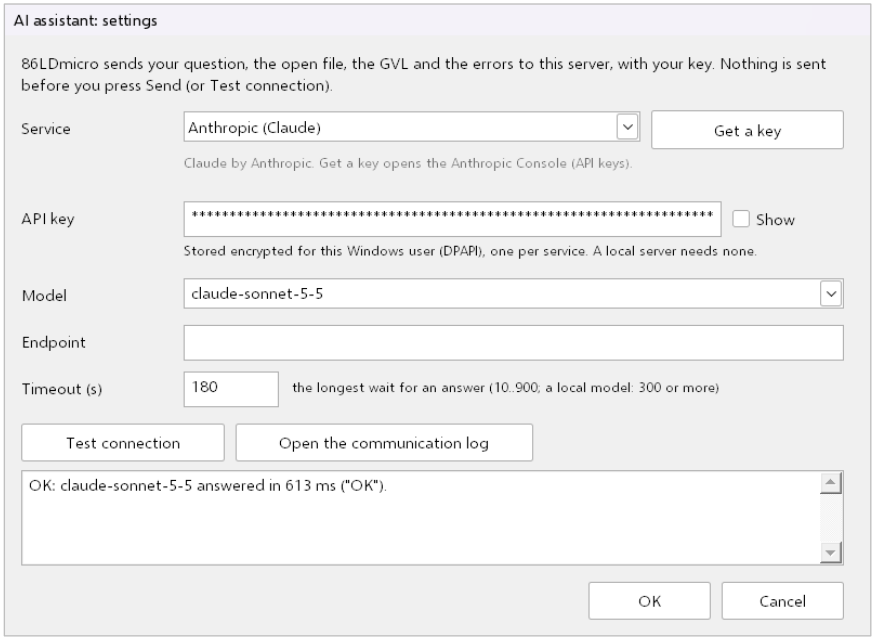


図 15-8　接続テストの結果：OK: … answered in N ms

Test connection のメッセージ	意味と対処
401 Unauthorized	API キーが間違っている、期限切れ、取り消された、または別のサービスのもの：キーを貼り直す（Get a key）
403 Forbidden	キーは有効だが、このモデルまたはエンドポイントを使用できない（プラン、組織、地域、プロジェクトの無効化）
404 Not Found	エンドポイントの URL またはモデル名が間違っている。ローカルサーバーの場合：モデルがロード／ダウンロード済みか（ollama pull）
408 Request Timeout	サーバーが待機を打ち切った：もう一度試す
429 Too Many Requests	レート制限を超えたか、利用枠を使い切った：しばらくしてから再試行するか、アカウントの課金を確認する
5xx Server error	サービス側の問題：しばらくしてから再試行する

Test connection のメッセージ	意味と対処
Timeout	設定した秒数内に応答がない：サーバーが混雑しているか、ローカルモデルがまだロード中。タイムアウトを長くするか、もう一度試す
Network: …	接続できない：ネットワークがない、DNS、エンドポイントの入力ミス、サーバーが起動していない（Ollama／LM Studio）、ファイアウォール、プロキシ、http／https の誤り、TLS 接続の失敗（HTTPS を検査するプロキシ、コンピューターの日付と時刻）

15.3.4　質問とコードの挿入

1. 検討したい POU（例：Main）を開き、AI タブの下部に質問を入力します。例えば「Rrun を、Xstart を押してから 2 秒後に起動するように変更して」と入力し、**Send**（Ctrl+Enter）をクリックします。
2. 回答は上部に表示されます。コードが含まれている場合は、**Insert the code** でエディターに挿入するか、**Copy the code** をクリックします。
3. 挿入後、ステータスバーに「The AI's code is in the editor: check it (Check, Shift+F7)」と表示されます。必ず **Check** と **Simulate** で確認してください。

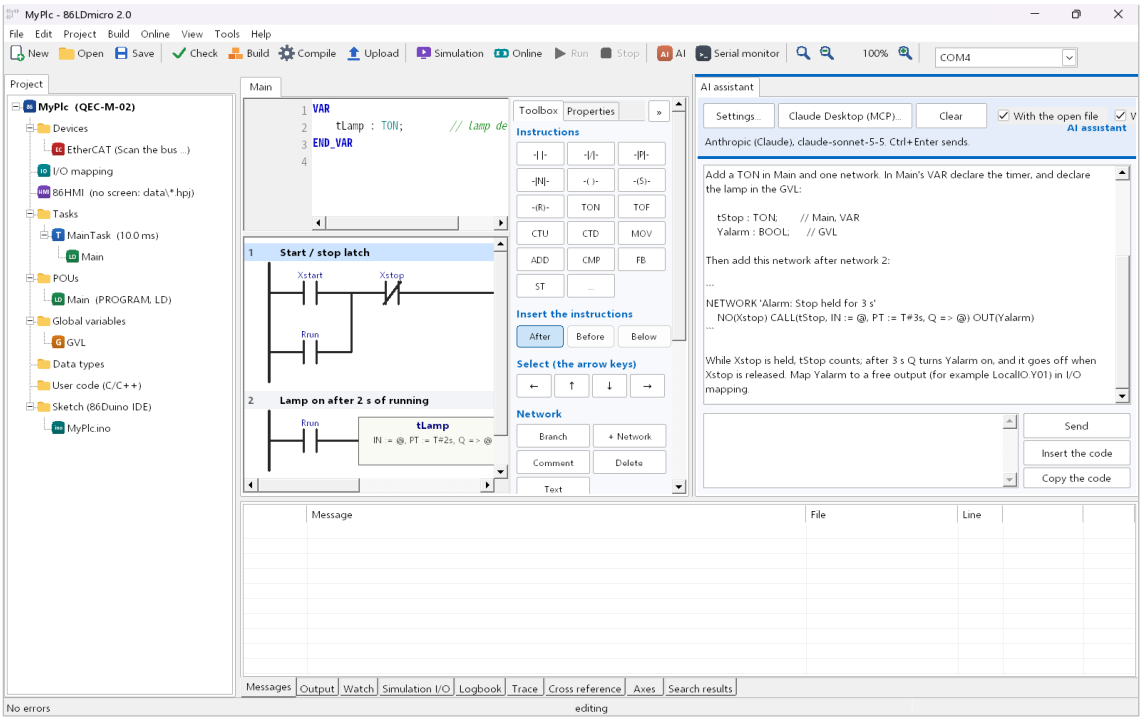


図 15-9　AI の回答と「Insert the code」

15.3.5　AI によるエラーの修正

「Messages」ウィンドウの各エラー行の右端には **Fix with AI** があります。AI が設定済みの場合は、そのエラーと周辺のコードについて AI タブで直接質問します。未設定の場合は、質問内容をクリップボードにコピーするので、他の AI ツールに貼り付けられます。

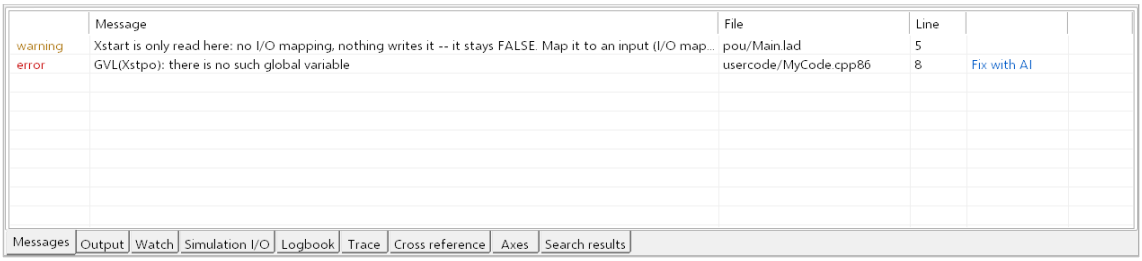


図 15-10　「Messages」ウィンドウの「Fix with AI」

***注意：**AI の回答には誤りが含まれる場合があります。挿入したコードは自分で理解し、チェックとシミュレーションを経てから実機にアップロードしてください。

15.4 MCP

MCP（Model Context Protocol）server を使うと、Claude デスクトップ版、Cowork、Claude Code など MCP に対応した AI ツールから、86LDmicro プロジェクトを直接読み取り、変更、チェック、シミュレーションできます。MCP server は 86LDmicro の `mcp` フォルダー（`mcp\ld86_mcp.py`）にインストールされており、Python 3 が必要です。

15.4.1 Claude デスクトップ版の設定

1. Python 3 をインストールします（python.org。インストール時に「Add to PATH」をチェック）。
2. AI Assistant タブで **Claude Desktop (MCP)...** をクリックし、「AI assistant (MCP)」ウィンドウを開きます。ウィンドウ上部に MCP server、Python、現在のプロジェクトの状態が表示されます。
3. 「**Set up Claude Desktop (MCP)**」をクリックします。86ldmicro が `claude_desktop_config.json` に追加されます（元のファイルは `.bak` としてバックアップされます）。
4. Claude デスクトップ版を再起動し、直接質問します。例：「<フォルダー> の 86LDmicro プロジェクトを開いて、MainTask を説明して」。
5. その他の MCP クライアント（Cowork、Claude Code…）：**Copy the MCP configuration (JSON)** をクリックし、そのツールの MCP 設定に貼り付けます。

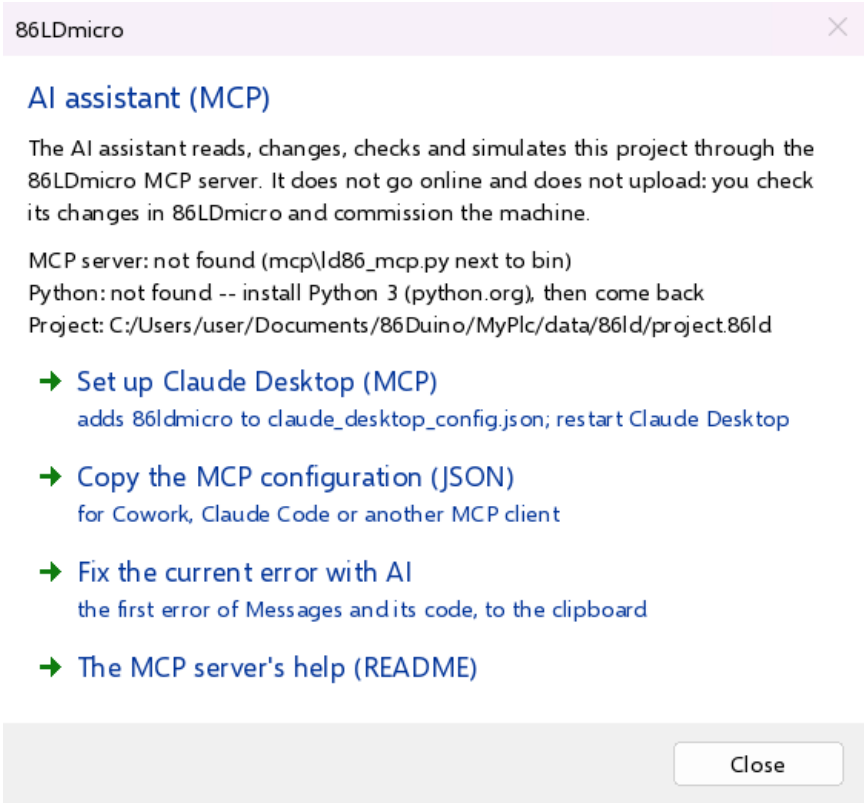


図 15-11 「AI assistant (MCP)」：Set up Claude Desktop、Copy the MCP configuration、Fix the current error with AI、MCP server の説明

ウィンドウ内の項目	用途
Set up Claude Desktop (MCP)	86ldmicro を claude_desktop_config.json に追加する。その後 Claude デスクトップ版を再起動する
Copy the MCP configuration (JSON)	Cowork、Claude Code、その他の MCP クライアント用
Fix the current error with AI	「Messages」の最初のエラーとコードをクリップボードにコピーする
The MCP server's help (README)	開く：`mcp\README.md`

15.4.2 MCP ツール

MCP ツール	用途
ld86_describe	プロジェクトの概要：タスク、POU、GVL、データ型、ユーザーコード、デバイス、I/O マッピング
ld86_read_file／ld86_write_file	POU、GVL、データ型、ユーザーコードの読み取り／追加または置換（書き込み後に自動的にコンパイルし、メッセージを返す）
ld86_set_project	タスク、デバイス、I/O マッピング、バインドなどを設定し、続けてチェックする
ld86_check／ld86_simulate	コンパイルしてエラーを一覧表示。N スキャンをシミュレーションして変数を一覧表示
ld86_build	sketch の生成ファイルを書き出す（コンパイルもアップロードもしない）
ld86_new_project／ld86_import_11／ld86_ethercat_eni	新規プロジェクト、1.x または PLCopen XML のインポート、ENI からの EtherCAT デバイスの作成
ld86_lower／ld86_instructions	LD から変換された ST、命令セットの説明

15.4.3 安全

- MCP server は**コントローラーに接続せず、アップロードも行いません。**AI が変更したプロジェクトは、86LDmicro で確認、シミュレーション、試運転を行ってから、自分でアップロードしてください。
- AI が MCP 経由でファイルを書き込んだとき、86LDmicro で同じプロジェクトを開いている場合は、「**View > Refresh: the project again from the disk**」（Ctrl+R）で変更を読み込みます。未保存の変更がある場合は、先に確認されます。
- AI による変更を比較したり元に戻したりできるよう、バージョン管理を使用するか、先に sketch の `data\86ld` をバックアップしておくことを推奨します。

15.5 コマンドラインツール

tool\bin\86ldmicro-cli.exe は 86LDmicro と同じチェック、ビルド、シミュレーション、アップロード、オンラインの機能を提供し、バッチファイルや自動テストで使用できます。引数なしで実行すると、すべてのコマンドが一覧表示されます。

```
> 86ldmicro-cli
86ldmicro-cli 2.0
  import <file.ld | file.xml> <sketch folder> [name]
  check  <project>
  search <project> <text> [--case] [--word]
  build  <project>
  lower  <project> <POU>
  new    <sketch folder> [name]
  compile <project> [--ide <86Duino IDE folder>]
  upload <project> <port> [--ide <dir>]
  upload-res <project> <port> [--ide <dir>]
  iotable export|import <project> <file.xlsx|.csv>
  format <file>
  ports
  instructions
  online <project> <port> status | read <path>... | write <path> <value> | ...
  eni    <project> <eni.xml> [name on the controller (C:/eni/)]
  sim    <project> <scans> [path...] [--set path=value]... [--diff-inputs]
```

コマンド	用途
new <sketch フォルダー> [名前]	新しいプロジェクトを作成する（MainTask 10 ms、ラダー図 Main、GVL）
import <myLD.ld | ファイル.xml> <sketch フォルダー> [名前]	LDmicro 1.x プログラムまたは PLCopen XML をインポートする（15.1、15.2 節）
check <プロジェクト>	チェック（Shift+F7 と同じ）
build <プロジェクト>	ビルド：plc86.cpp などを sketch に書き出す（Ctrl+B と同じ）
compile <プロジェクト> [--ide <フォルダー>]	コンパイルのみ、アップロードしない（F7 と同じ）
upload <プロジェクト> <COM> [--ide <フォルダー>]	コンパイルしてアップロードする（F8 と同じ）
upload-res <プロジェクト> <COM>	86HMI のリソース data\Resource のみをアップロードする（13.6 節）
sim <プロジェクト> <スキャン数> [変数...] [--set 変数=値]...	N スキャンをシミュレーションし、変数の値を一覧表示する
search <プロジェクト> <テキスト> [--case] [--word]	プロジェクト内を検索する（Ctrl+Shift+H と同じ）
lower <プロジェクト> <POU>	POU から変換された ST を一覧表示する
iotable export|import <プロジェクト> <ファイル.xlsx|.csv>	I/O マッピング表を Excel にエクスポート／Excel からインポートする
eni <プロジェクト> <eni.xml>	ENI をインポートし、EtherCAT デバイスを作成する（第 11 章）
format <ファイル>	自動整形（エディターの Ctrl+T と同じ）

コマンド	用途
`ports`	シリアルポートを一覧表示する
`instructions`	ラダー命令を一覧表示する（JSON）
`online <プロジェクト> <COM> status|read|write|force|unforce|unforce-all|run|stop|reset warm|cold|log|watch`	オンライン操作（10.4 節）
`online <プロジェクト> <COM> sdo|coe <SubDevice> ...`	EtherCAT SDO の読み書き、CoE オブジェクトディクショナリの読み取り（11.6 節）

`<プロジェクト>` には `project.86ld`、それがあるフォルダー、または `sketch` フォルダーを指定できます。例：第 5 章の `MyPlc` プロジェクト（`C:\work\MyPlc`）をチェックし、次に `250` スキャン（`MainTask 10 ms`、約 `2.5` 秒）をシミュレーションします。最初のスキャンの前に `Rrun` を `TRUE` に設定し、`Rrun` と `Ylamp` を一覧表示します。`201` 番目のスキャン（`2` 秒）から `Ylamp` が `TRUE` になります。

```
86ldmicro-cli check C:\work\MyPlc
86ldmicro-cli sim C:\work\MyPlc 250 Rrun Ylamp --set Rrun=TRUE
```

`--set` は最初のスキャンの前に一度だけ書き込むもので、強制（Force）ではありません。その後、プログラムやシミュレーション I/O によって書き換えられることがあります。入力チャンネルにマッピングされた変数（例：Xstart）はスキャンごとにシミュレーションの入力で上書きされるため、この例では内部変数 Rrun を設定しています。新しいプロジェクトを作成するには `86ldmicro-cli new C:\work\Test Test` を使用します（MainTask、空のラダー図 Main、GVL）。

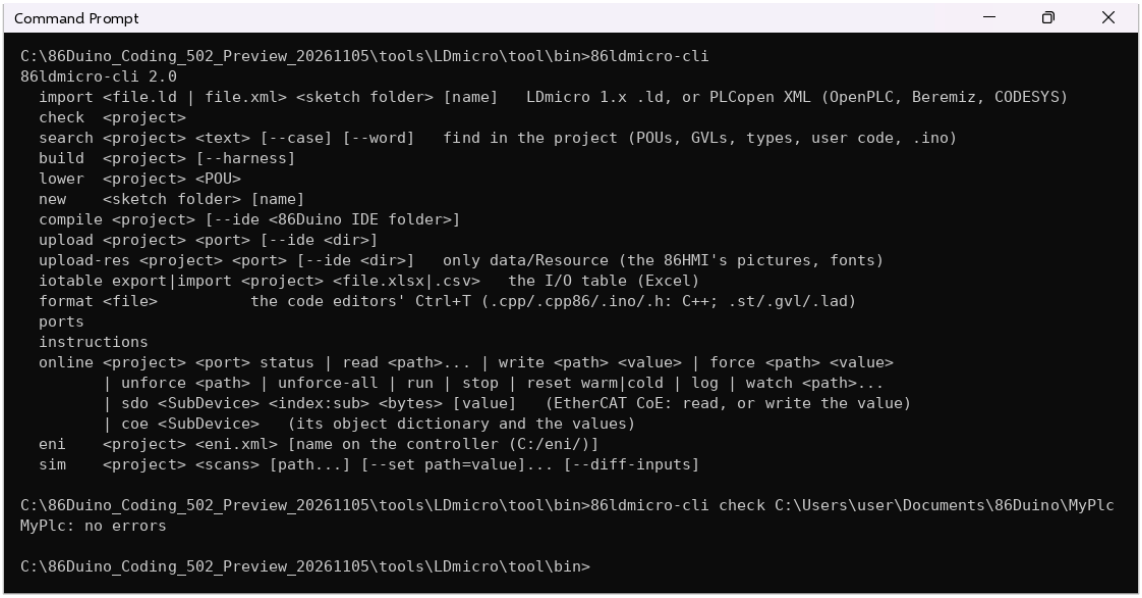


図 15-12　コマンドプロンプトで 86ldmicro-cli を実行

15.6 プロジェクト保護

顧客に納品するプロジェクトでは、一部のプログラムをパスワードで保護できます。「**Project > Protection (password)...**」でパスワードを設定してから、各 POU とユーザーコードの保護方式を個別に指定します。



図 15-13 「Protection」：パスワードと各ファイルの保護方式

保護方式	効果
none	通常のファイル
read-only	開く、シミュレーション、ビルドは可能。パスワードなしでは変更できない
hidden (encrypted)	ディスク上で AES-256 により暗号化（キーはパスワードから PBKDF2-SHA256 で導出）し、ディスク上には PROGRAM の行のみが残る。パスワードがない場合は内容が表示されず、開く、チェック、シミュレーション、ビルドもできない

15.6.1 保護の設定

1. 「Project > Protection (password)...」を選択します。
2. **Password** と **Password again** を入力します（4 文字以上、2 回とも同じ）。
3. 各ファイル（例：`Main (pou/Main.lad)`）の横で **none**、**read-only**、**hidden (encrypted)** のいずれかを選択し、**OK** をクリックします。
4. ステータスバーに「The protection is set: Project > Protection > Lock to lock it now (it is locked when the project is opened again)」と表示されます。

15.6.2　ロック解除、変更、削除

保護されたプロジェクトは、開いたときにロックされた状態です。もう一度「Project > Protection (password)...」を選択してプロジェクトのパスワードを入力するとロックが解除され（このセッション中有効）、以降同じメニューで次の操作ができます。

オプション	説明
Change the protection	各ファイルの保護方式、パスワード（新しいパスワードが空欄：変更なし）
Lock now	保護されたファイルで再びパスワードが必要になる
Remove the protection	すべてのファイルを暗号化なしに戻し、パスワードが不要になる（もう一度確認される）

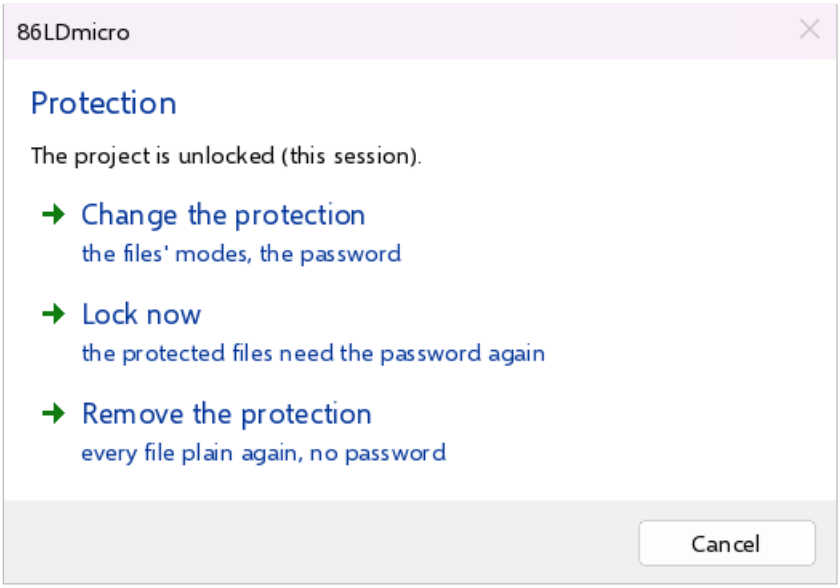


図 15-14　ロック解除後の「Protection」：Change the protection、Lock now、Remove the protection

*注意：パスワードを紛失すると復元できません。ご自身で大切に保管してください。hidden (encrypted) のファイルはパスワードなしではビルドできないため、納品前に顧客側でこれらのプログラムを変更する必要がないことを確認してください。

第 16 章　よくある質問

この章では、インストール、アップロード、プログラム、シミュレーションとオンライン、86HMI、EtherCAT とモーション制御で最もよく遭遇する問題をまとめています。エラーメッセージの完全な対照表は 9.6 節を参照してください。

Q1：86Duino IDE の「Tools」メニューに 86LDmicro がない？

86LDmicro のファイルが IDE の tools\LDmicro\tool（LDmicro.jar を含む）に展開されていることを確認してから、86Duino IDE を再起動します。実行時に Windows にブロックされる場合は、展開前に「ブロックの解除」をしていなかったことを意味します。zip ファイルを右クリック > プロパティ > ブロックの解除 を行ってから、もう一度展開してください（2.3 節）。

Q2：「Compile」または「Upload」をクリックすると「The 86Duino IDE is not found」と表示される？

86LDmicro を単独で起動した場合（86Duino IDE の「Tools > 86LDmicro」から開いたのではない場合）は、IDE の場所を指定する必要があります。「**Tools > Options...**」で「The 86Duino IDE folder」を設定してください（2.6 節）。

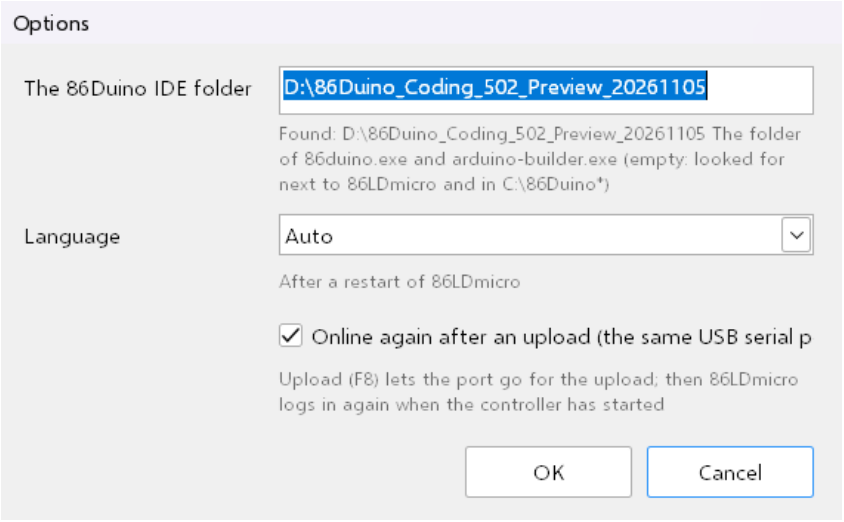


図 16-1　Options：The 86Duino IDE folder、Language、Online again after an upload

Q3：EtherCAT または Motion86 のプロジェクトのコンパイルに失敗する（sdoUpload_Begin、IKinematicsCB が見つからない、expected class-name など）？

86Duino IDE のバージョンが合っていません。**86Duino Coding 502 Preview（20261105）**の IDE パッケージ一式を使用してください。新しい Motion86（9 軸）と、それに必要な下位ライブラリが同梱されています。新しい Motion86 ライブラリだけを古い IDE にコピーすると、コンパイルに失敗します。

Q4：アップロード失敗：COMx を開けない（シリアルポートが他のプログラムに使用されている？）

同じ COM ポートを他のプログラムが開いています。最も多いのは 86Duino IDE のシリアルモニターウィンドウやターミナルプログラムです。それらを閉じてから、もう一度 **Upload**（F8）をクリックします。86LDmicro 自身の「Serial monitor」とオンラインモニターは、アップロード時に自動的にシリアルポートを解放するため、手動で閉じる必要はありません。

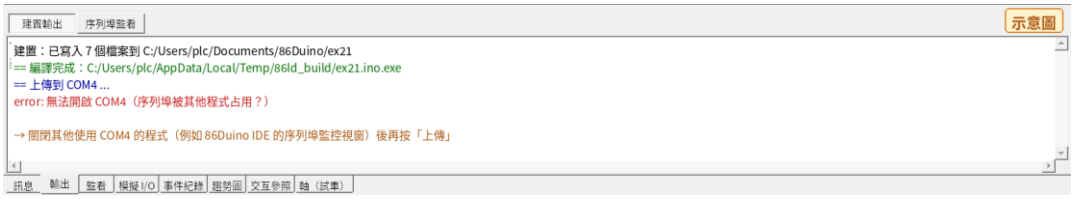


図 16-2　アップロード失敗：COM4 を開けない（シリアルポートが他のプログラムに使用されている）

「Output」のメッセージ	対処
COMx が見つからない	USB ケーブルが抜けている、ポートの選択が間違っている、またはコントローラーに電源が入っていない：2.4 節に従って COM ポートを確認し、ツールバーで選択し直す
アップロードの途中でタイムアウト	USB ケーブルやハブが不安定、またはコントローラーが再起動中：コンピューターの USB ポートに直接接続し、再度アップロードする
コンパイルエラー（.cpp のファイル名と行番号）	C/C++ ユーザーコードまたはライブラリのエラー：「Messages」でダブルクリックして .cpp86 の該当行にジャンプし、修正する

Q5：アップロードは成功したが、実際のボタンを押しても反応しない？

最も多い原因は**変数に I/O マッピングがない**ことです。PLC プログラムは変数を読み書きするだけで、変数と物理点（例：`LocalIO.X00`）とのリンクは「I/O mapping」ページで設定します（4.6 節）。ビルド時に「Messages」に次の警告が表示されます。

「Xstart はここで読み取られるだけです：I/O マッピングがなく、どこからも書き込まれないため、常に FALSE です。入力にマッピングする（I/O mapping、または EtherCAT > Variable mapping）か、プログラム／86HMI のボタンで設定してください」

1. プロジェクトツリーで「I/O mapping」をダブルクリックし、すべての入力変数と出力変数にチャンネルがあることを確認します。
2. ない場合は **Add...** をクリックし、変数とチャンネルを選択します（例：`Xstart` → `LocalIO.X00`）。
3. もう一度 **Upload**（F8）します。

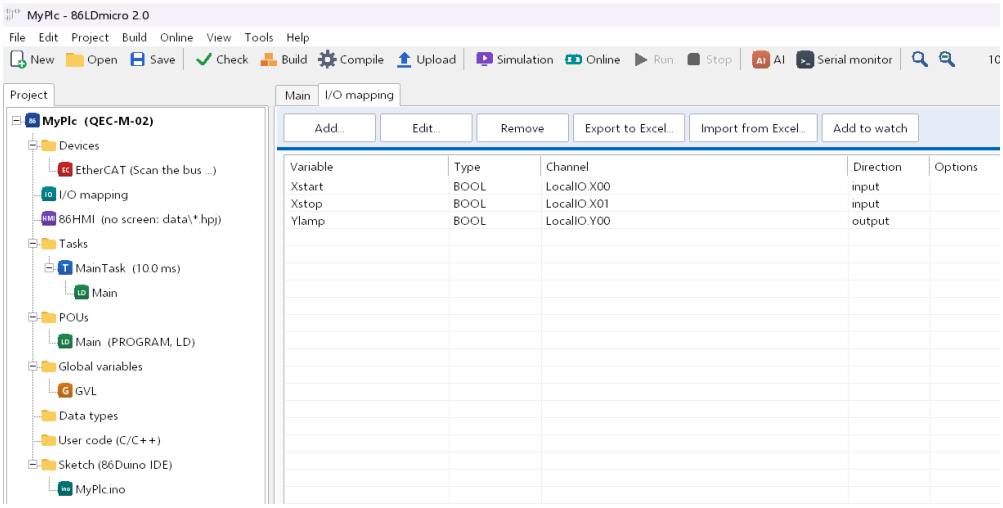


図 16-3　I/O mapping ページ：各変数を 1 つの物理チャンネルにマッピング

Q6：ラダー図の要素が赤い背景になる？

その要素の変数が宣言されていない、名前の入力ミス、または型が合っていない（例：REAL を接点として使用）可能性があります。下部の「Messages」に切り替え、エラー行をダブルクリックするとその要素が選択されます。名前を修正するか GVL／VAR で宣言してから、エラーがなくなるまで **Check**（Shift+F7）をクリックします。よくあるメッセージと対処は 9.6 節を参照してください。

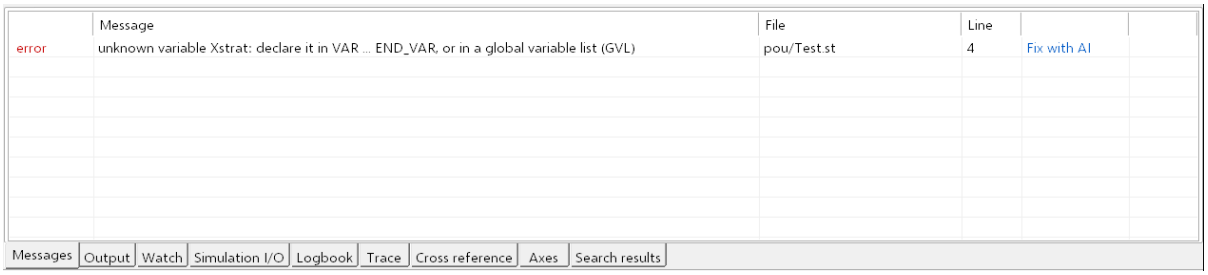


図 16-4　「Messages」ウィンドウ：1 行に 1 つのエラー。ダブルクリックでその要素にジャンプ

Q7：オンライン（Ctrl+F5）の接続が拒否される、または応答がない？

- **拒否される**（コントローラー上のプログラムが今回のビルドと異なる）：コントローラーで実行されているのは現在のプロジェクトの今回のビルドのプログラムではありません。もう一度 **Upload**（F8）してください。
- **応答がない**：「Project settings」でオンラインモニターが有効になっており、ポートが Serial（USB）であることを確認します。ユーザーコードと `.ino` で USB の `Serial` を使用しないでください。設定を変更したら、再度ビルドしてアップロードします。
- 接続が 3 秒間途切れると、コントローラーはすべての強制を自動的に解除します。

Q8：プログラムがしばらく動くと停止する（またはコンピューターを接続していないと停止する）が、86LDmicro を接続すると復帰する？

通常、プログラムが USB の `Serial` に出力し続けていることが原因です。コンピューターが読み取っていないとき（86LDmicro がオンラインでない、シリアルモニターが閉じている）、`Serial.print()` はバッファーがいっぱいになると待機し、プログラム全体（PLC スキャンを含む）も停止します。本稼働時は `Serial` に出力し続けないでください。外部機器との通信には `Serial1`〜`Serial3` を使用します。デバッグ出力の前には `Serial.availableForWrite()` が十分かどうかを確認してください（14.6 節）。

Q9：変数を強制したのに効果がない？

強制値は各スキャンの**開始時**（入力の読み込み後）と**終了時**（出力の書き出し前）に書き込まれます。スキャン中にプログラムが書き換えた変数は、その後のプログラムではプログラムが書いた値が読まれます。したがって、強制は I/O マッピングだけが書き込む変数（物理入力）と出力に対して最も効果的です。

- Cross reference（Ctrl+Shift+F）で、この変数がどのプログラムから書き込まれているかを調べます（9.5 節）。
- 例：ex30 のデモモード（`gDemo = TRUE`）では、PlantSim が `sim*` を、IoMap が `x*` を書き換えるため、これらを強制しても上書きされます。先にデモモードをオフにしてから（`gDemo = FALSE`）、`diEntry` などの物理入力を強制してください。

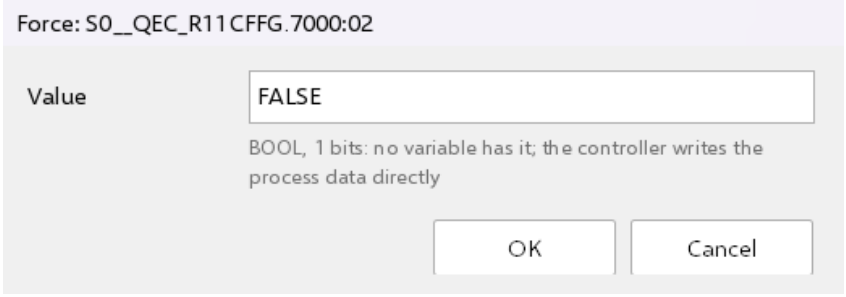


図 16-5　変数の値を強制する

Q10：シミュレーション中に Watch ウィンドウで入力変数に書き込んでも、値がすぐに元に戻る？

物理入力にマッピングされた変数は、シミュレーション中は「**Simulation I/O**」によって決まります（スキャンごとに読み込み直されます）。下部の「Simulation I/O」でそのチャンネルをダブルクリックして ON／OFF を切り替える（またはアナログ値を入力する）か、強制を使用してください（9.2、9.3 節）。

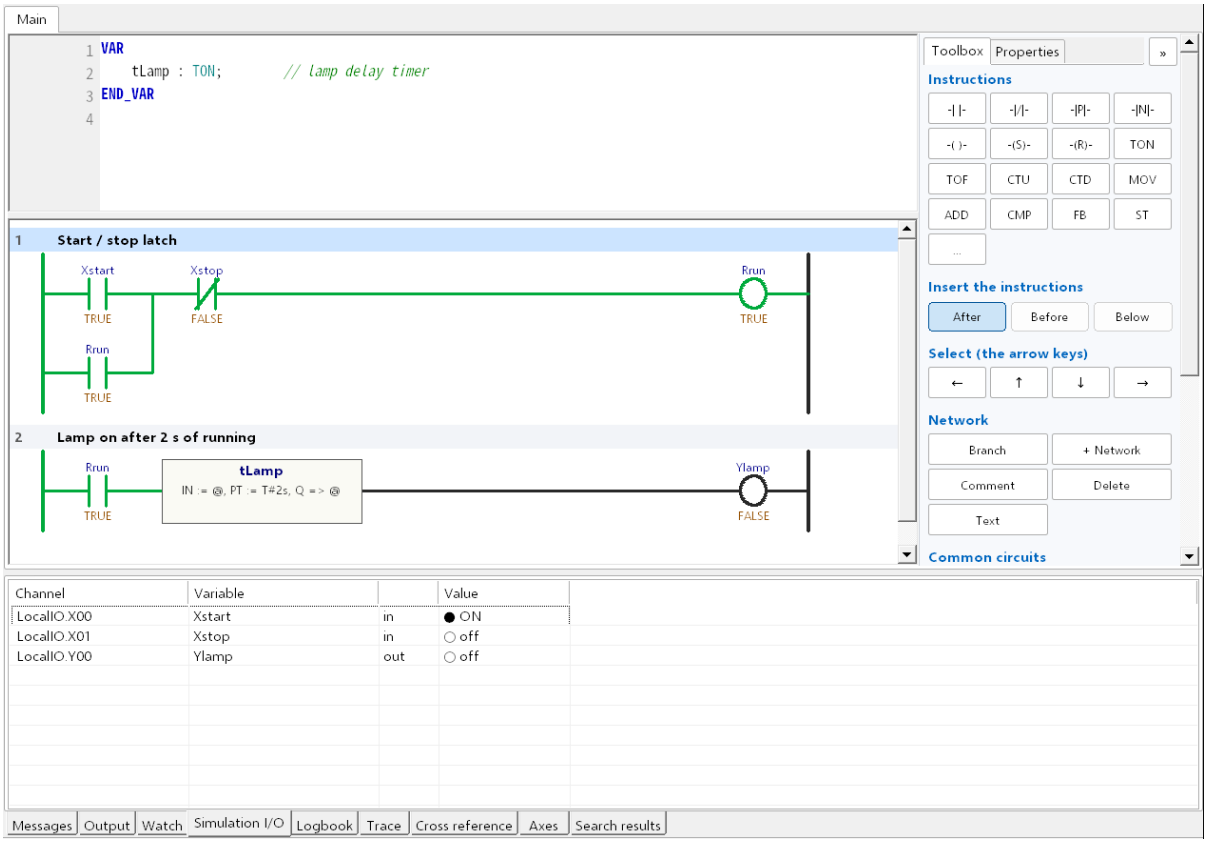


図 16-6　「Simulation I/O」：チャンネルをダブルクリックしてシミュレーションの物理入力を切り替える

Q11：86HMI 画面のボタンを押しても PLC が反応しない？

- ボタンが**物理入力にマッピングされた変数**にバインドされています。次のスキャンで物理入力によって上書きされます。内部変数（例：`Rstart`）にバインドし直し、プログラム内で実際のボタンと並列に接続してください（13.3 節）。
- バインドの使い方が「イベント」なのに、同じ変数を 2 つのタスクで読み取っている：最初のタスクが読み取れることしか保証されず、ビルド時に警告が出ます。1 つのタスクでのみ読み取り、必要に応じて SET でラッチしてください。
- シミュレーション中にバインドを変更した：バインドは次にシミュレーションを開始したとき（F5）に有効になります。いったんシミュレーションを終了してから F5 を押してください。
- 86HMI ページの表で、そのウィジェットの「Variable」列が空欄：まだバインドされていません（13.2 節）。

Q12：86HMI のシミュレーション中、ToggleButton の文字が「押されていない」と「押された」の間で切り替わらない？

これは 2.0.0 の**既知の問題**です。ToggleButton の状態は正しく切り替わり（バインドされた変数も正しい）、シミュレーションウィンドウのボタン上の文字だけが変わりません。実機では 86HMI が生成したプログラムが文字を更新するため、影響はありません。

Q13：86HMI の言語ボタンがシミュレーションでは正常なのに、コントローラーにアップロードした後は押しても言語が切り替わらない？

言語ボタンには、**86HMI Editor** で切り替えスクリプトを書く必要があります。ボタンの Clicked イベントに `Hmi.setLanguage(1);`（数字は言語番号）を追加し、「Code Generation」をクリックしてから再度アップロードします（13.6.2 節）。86LDmicro のバインドは変数のみを扱い、言語の切り替えは行いません。

Q14：EtherCAT が動作しない（サブデバイスの入出力が動かない）？

1. 「**Serial monitor**」を開きます（オンラインにする必要はありません）。正常な場合は 5 秒ごとに `EtherCAT: running, N slaves, ...` と表示され、そうでない場合は失敗の原因が表示されます。
2. `begin failed`：EtherCAT ケーブルがメインデバイスの EtherCAT ポートに接続されていない、またはサブデバイスに電源が入っていない。
3. `no slave`：ケーブルが接続されていない、サブデバイスに電源が入っていない、またはサブデバイスの IN ではなく OUT に接続されている。
4. `the ENI … cannot be used`：ENI がメインデバイスにアップロードされていない、または実際のネットワークと一致しない。再度 Scan し、ENI をエクスポートしてアップロードする（第 11 章）。
5. sketch の `.ino` では 1.x の `EVA.begin()` を呼び出してはいけません。`#include "myeva.h"` と、sketch 内の `myeva.cpp`／`myeva.h` を削除してください。
6. オンライン時に「Logbook」の EtherCAT メッセージを確認します。

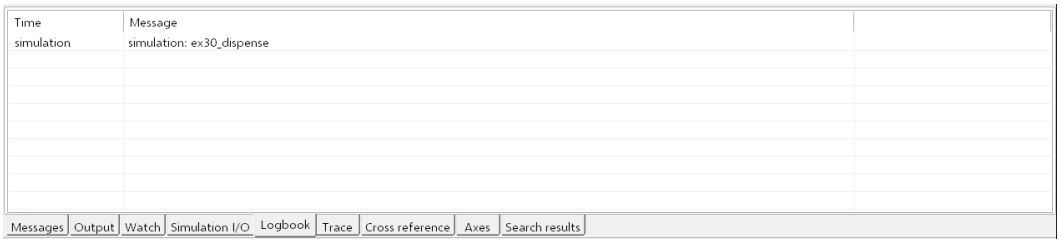


図 16-7　Logbook：EtherCAT メインデバイスの起動メッセージとプログラムの実行開始

Q15：軸が動かない、方向が逆、または原点復帰後に軸ページに表示される位置が正しくない？

現象	対処
軸が動かない	サーボオン（`MC_Power.Status`）になっていること、ドライブにアラームがないこと、ENI の PDO に 607Ah／6064h があることを確認する。マシンページの「Online」で状態を確認する（第 12 章）
方向が逆	軸ページで「Reverse the direction」をチェックし、再度ビルドしてアップロードする
原点復帰後、表示される位置が予想と一定量ずれる	軸ページの「Feedback shown」で他のソース（例：QEC-R11MP3S のエンコーダー）を使用している場合、このバージョンの原点復帰は**そのソースを再調整しません。**「Feedback shown」のオフセットで合わせてください。CSP 制御、インポジション、原点、リミットは引き続き 6064h を使用するため、影響を受けません
ドライブがサーボオフの状態で手で動かされた	位置の基準がずれる（Logbook に通知される）：もう一度原点復帰を行うか、`MC_SetPosition` を使用する

Q16：MC 移動ファンクションブロック（MC_MoveAbsolute など）が 1 回しか実行されない？

PLCopen MC ファンクションブロックは、`Execute` の**立ち上がりエッジ**でのみ起動します。もう一度実行するには、`Execute` をいったん FALSE に戻してから TRUE にする必要があります。同じマシンの移動は順番にキューに入り（Buffered）、新しい命令が実行中の移動を中断することはありません。`MC_Stop`／`MC_Halt` は即時停止で、減速ランプはありません（12.7、12.9 節）。

Q17：ST エディターで Ctrl+Space を押しても補完リストが表示されない？

このバージョンの ST エディターの補完は、**変数やキーワードの最初の数文字を入力すると自動的に一覧表示**されます（型とコメントを表示）。ST エディターでは Ctrl+Space でリストは開きません。C/C++ ユーザーコードと sketch ファイルのエディターでは Ctrl+Space を使用できます。

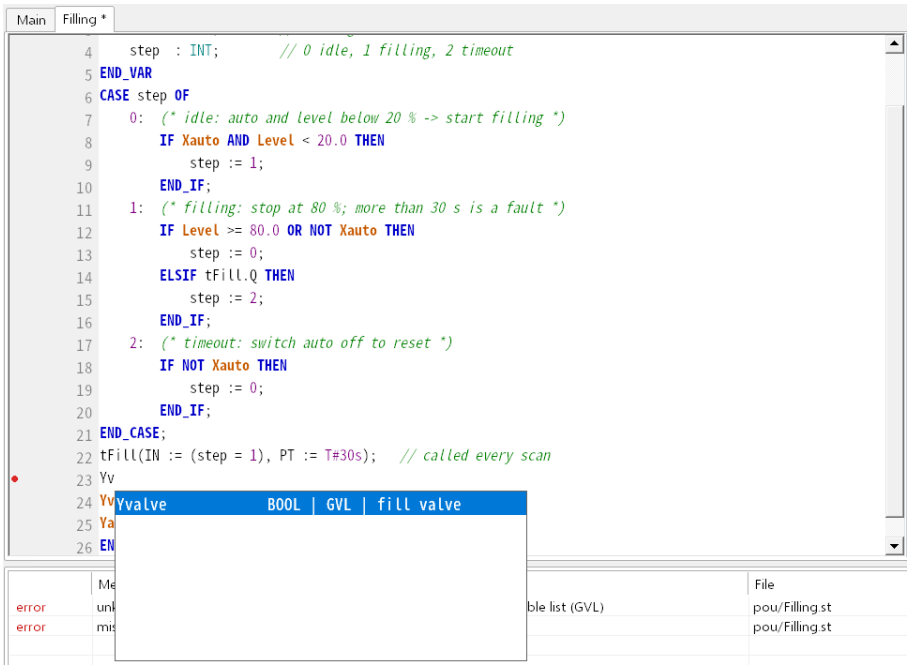


図 16-8　ST の補完：Yv と入力すると Yvalve（BOOL、GVL、コメント）が表示される

Q18：PLCopen XML をインポートした後、一部のプログラムが足りない？または外部でファイルを変更したのに、86LDmicro に反映されない？

- PLCopen XML の **FBD と IL** のプログラムはこのバージョンではインポートされず、「Output」と「Messages」に一覧表示されます。元のツールで ST または LD に書き直してから、再度エクスポートしてください（15.2 節）。
- 外部エディター、AI／MCP、またはバージョン管理でプロジェクトのファイルを変更した後は、「**View > Refresh: the project again from the disk**」（Ctrl+R）を選択します。未保存の変更がある場合は先に確認されます。シミュレーション中またはオンライン中は、先に停止してください。

問題を報告する際は、次の情報を添付してください。

- 86LDmicro のバージョン（`tool\VERSION.txt`、例：`2.0.0 2026-11-11`）とハードウェア構成（コントローラー、モジュール、ドライブの型番）。
- 操作手順と画面のスクリーンショット、「Output」ウィンドウのビルドメッセージ、「Logbook」の内容。
- 必要に応じて、プロジェクトフォルダー（sketch 内の `data\86ld`）の圧縮ファイル。連絡先は本マニュアル末尾の「お問い合わせ」を参照してください。

付録 A　LDmicro 1.1 との対照

LDmicro 1.1（ユーザーマニュアル v1.6.3）に慣れている方は、下表で 86LDmicro 2.0 での方法を対照できます。1.1 のプロジェクトのインポート方法は 15.1 節を参照してください。

項目	LDmicro 1.1	86LDmicro 2.0
起動	86Duino Coding の「Tools」メニューから LDmicro を起動	86Duino IDE の「Tools > 86LDmicro」、または IDE ツールバーの 86LDmicro ボタン（2.5 節）
プログラムファイル	1 つの `.ld` ファイル（`data\myLD.ld`）	プロジェクトフォルダー `data\86ld`：`project.86ld`、`pou`、`gvl`、`dut`、`usercode`（4.1 節）
プログラミング言語	ラダー図（ST ブロックを含む）	LD、ST、SFC（IEC 61131-3）と C/C++ ユーザーコード
MCU／ボードの選択	Settings → General（PLC Configuration）	「Project > Settings」のコントローラー（既定値 QEC-M-02）
スキャン周期	Settings → General の PLC Cycle Time	「Task」の周期（既定値 MainTask 10 ms）。複数のタスクを持てる（4.3 節）
COM ポートとボーレート	Settings → General の COM Mapping、Baudrate	「Devices」の COM デバイス（4.4 節）
Modbus	Settings → Modbus、変数の接頭辞 M／I／H	「Devices」の modbus、modbus-server と I/O マッピング
EtherCAT	Settings → Virtual Arduino ＋ 86EVA Configurator（`myeva.h`／`myeva.cpp`、`EVA.begin()`）	EtherCAT Configurator → ENI をエクスポート → 86LDmicro が ENI を読み込み、サブデバイスがデバイスになる（第 11 章）。`.ino` で `EVA.begin()` を呼び出さなくなった
変数と型	接頭辞で種類が決まる（X 入力、Y 出力、R 内部、T タイマー、C カウンター）。IO List でダブルクリックしてサイズを変更	GVL／VAR で IEC の型（BOOL、INT、REAL、TIME…）を宣言する。接頭辞は命名の慣習にすぎない（付録 C）
I/O ピン	IO List の Pin Name 列をダブルクリックしてピンを選択	「I/O mapping」ページ：変数 ↔ チャンネル（例：`LocalIO.X00`）。反転、フィルター、スケーリングを含む（4.6 節）
タイマー、カウンター	TON、TOF、CTU… 命令（周期単位でカウント）	IEC ファンクションブロック TON、TOF、TP、CTU…（`CALL`、時間単位）。1.x の命令も引き続き使用可能（第 7 章）
C コードの生成	Code Generation ボタン	**Build**（Ctrl+B）
生成されるファイル	`myplc.cpp`／`myplc.h`、`myeva.h`／`myeva.cpp`、`ladder.h`	`plc86.cpp`／`plc86.h`、`plc86_rt.*`、`plc86_ecat.*`（14.5 節）
`.ino` のエントリーポイント	`EVA.begin()` → `setupPlc()`／`loopPlc()`	`plc86_setup()`／`plc86_loop()`（ビルドで自動的に追加）

項目	LDmicro 1.1	86LDmicro 2.0
コンパイルとアップロード	86Duino Coding に戻って「Upload」をクリック	86LDmicro で直接 **Compile**（F7）、**Upload**（F8）（10.1 節）
シミュレーション	F7 シミュレーションモード、F8 リアルタイムシミュレーション、Space で単一サイクル	**Simulate**（F5）、速度 x1／x10。Simulation I/O、86HMI シミュレーションウィンドウ（第 9 章）
オンラインモニター	オンラインモニター（ldmon）	**Online**（Ctrl+F5）：パワーフロー、ウォッチ、書き込み、強制、RUN／STOP（10.4 節）
86HMI	86HMI バインドダイアログ	86HMI ページとバインドフォーム（同じ考え方）。1 つのウィジェットに複数のバインドが可能（第 13 章）
ユーザー C コード	User Code（global／setup／beforescan／afterscan／loop／ethercat）	ユーザーコードファイル `.cpp86` のフック関数。`GVL()` で変数にアクセス（第 14 章）
AI	コンパイルエラー時に AI ウィンドウが開く	AI Assistant タブ、「Messages」の「Fix with AI」、MCP（15.3、15.4 節）
ヘルプ	F1 または File → Manual...（内蔵の manual.txt）	「Help」メニューでオンラインドキュメントサイトを開く。`tool\docs` はオフライン版

付録 B　ショートカットキー

以下は 86LDmicro 2.0.0 のすべてのショートカットキー（メインウィンドウのアクセラレーターキー、各エディターのキー）です。

B.1　メインウィンドウ（どのタブでも）

キー	機能	メニュー
Ctrl+N	New project	File
Ctrl+O	Open project	File
Ctrl+S	Save	File
Ctrl+Z／Ctrl+Y	Undo／Redo	Edit
Ctrl+X／Ctrl+C／Ctrl+V	Cut／Copy／Paste	Edit
Del	Delete	Edit
Ctrl+A	Select all	Edit
Ctrl+F	Find in this tab	Edit
F3／Shift+F3	Find next／Find previous	Edit
Ctrl+Shift+F	Find a variable in the project（Cross reference）	Edit
Ctrl+Shift+H	Find in the project（Search results）	Edit
Shift+F7	Check	Build
Ctrl+B	Build (write the sketch)	Build
F7	Compile (86Duino)	Build
F8	Upload to the controller	Build
F5	Simulate（開始／終了）	Online
Ctrl+F5	コントローラーにログイン（Login、オンライン）	Online
F9／Shift+F9	RUN／STOP	Online
Ctrl+＋（またはテンキーの＋）	Zoom in	View
Ctrl+－（またはテンキーの－）	Zoom out	View
Ctrl+0（またはテンキーの 0）	100%	View
Ctrl+マウスホイール	Zoom in／Zoom out	—
Ctrl+\	Side by side：手前のタブを右側に移動／並べて表示を終了	View
Ctrl+R	Refresh: the project again from the disk	View
Ctrl+W、マウスの中ボタン	タブを閉じる	—
Ctrl+Tab／Ctrl+Shift+Tab	次／前のタブ	—

B.2　検索バーとプロジェクトツリー

キー	機能
Enter または F3	検索バー：次へ
Shift+Enter または Shift+F3	検索バー：前へ
Esc	検索バーを閉じる
Ctrl+＋／Ctrl+－（プロジェクトツリーにフォーカスがあるとき）	Expand all／Collapse all
Enter（Search results）	選択した箇所にジャンプ（ダブルクリックと同じ）

B.3　ラダー図エディター

キー	機能
←／→／↑／↓	選択を移動
Ctrl+←／Ctrl+→	選択した要素を左／右に移動
Ctrl+↑／Ctrl+↓（ネットワークを選択しているとき）	ネットワークを上／下に移動
Home／End	最初／最後のネットワークを選択
Enter、ダブルクリック	選択した要素のパラメーターを編集。ネットワークを選択しているときは、ネットワークのタイトルとコメントを編集
Del、Backspace	選択した要素またはネットワークを削除
Ctrl+Enter／Ctrl+Shift+Enter	選択したネットワークの後／前にネットワークを挿入
Ctrl+C／Ctrl+X／Ctrl+V	コピー／切り取り／貼り付け
Ctrl+Z／Ctrl+Y（または Ctrl+Shift+Z）	元に戻す／やり直し
Ctrl+T	テキストモード（.lad テキスト）
Shift＋命令キー	選択した要素の前に挿入
B	選択した要素の下に a 接点を並列に追加

命令を挿入するキー（選択した要素の後、またはサイドパネルの「Insert the instructions」に従って挿入）：

キー	命令	キー	命令	キー	命令
C	a 接点 NO	S	セットコイル SET	D	CTD
N	b 接点 NC	R	リセットコイル RST	M	MOV
P	立ち上がりエッジ接点 PT	O	TON	A	ADD
L	コイル OUT	F	TOF	E	条件接点 CMP
K	ファンクションブロック CALL	U	CTU	T	ST 文

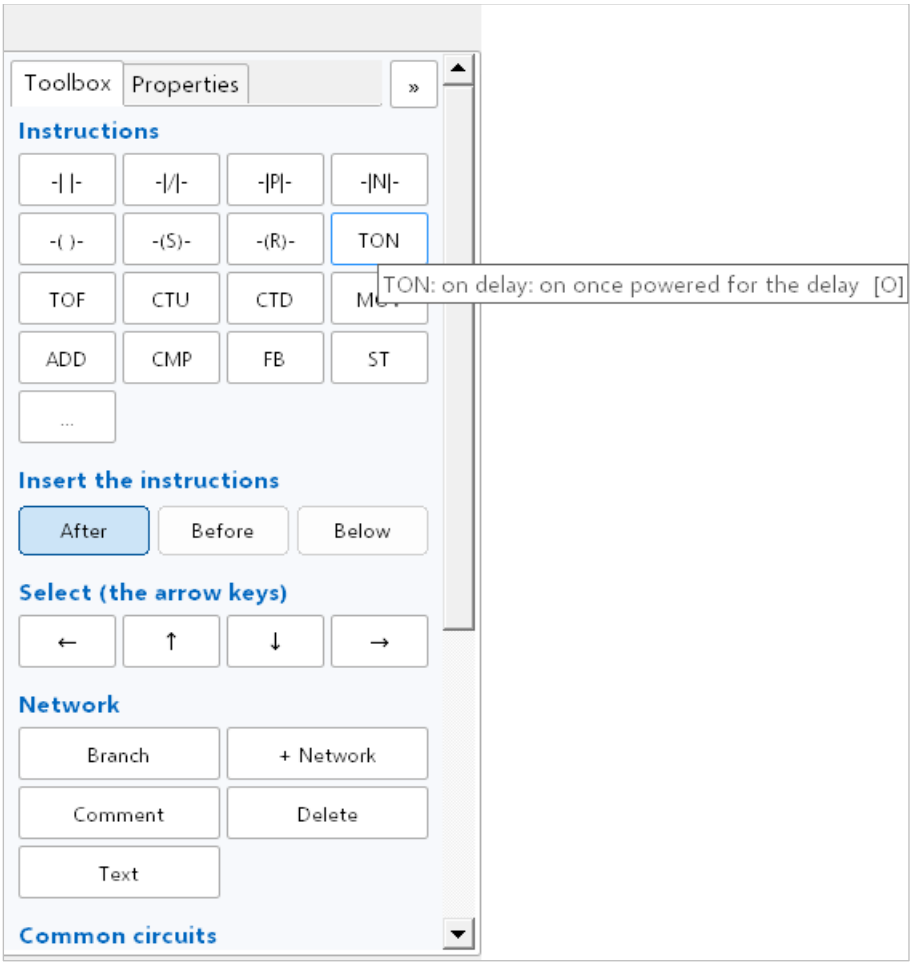


図 B-1　サイドパネルの「Toolbox」：命令ボタンにマウスを置くと説明とキーが表示される

B.4 SFC エディター

キー	機能
Insert	+ Step：選択箇所の後にトランジションとステップを 1 つ追加
Enter、ダブルクリック	選択したステップまたはトランジションを編集
Del	削除（ステップはそのトランジションと一緒に削除）
↑／↓	同じシーケンスの前／次の項目
←／→	隣の分岐
Ctrl+Z／Ctrl+Y（または Ctrl+Shift+Z）	元に戻す／やり直し
Ctrl+T	テキストモード（.sfc テキスト）

B.5 コードエディター（ST、GVL、データ型、C/C++、sketch）

キー	機能
Ctrl+T	自動整形（行頭のインデントと余分な空行のみを調整。Ctrl+Z で元に戻せる）
Ctrl+Space	補完リストを開く（C/C++ エディター）。ST エディター：最初の数文字を入力すると自動的に一覧表示される。このバージョンでは Ctrl+Space でリストは開かない
↑／↓、PgUp／PgDn	補完リスト内を移動
Enter または Tab	補完リスト：選択した項目を挿入
Esc	補完リストまたはパラメーターヒントを閉じる
Tab／Shift+Tab	4 文字分インデント／インデント解除
Enter	インデントを保って改行（{、THEN、DO などの後は 1 段深くインデント）
Ctrl+Z／Ctrl+Y	元に戻す／やり直し

B.6 AI アシスタント

キー	機能
Ctrl+Enter	質問を送信
Esc	「AI assistant: settings」を閉じる

付録 C　変数の命名と型

C.1　命名の慣習

86LDmicro 2.0 では変数の型は宣言によって決まり、名前に制限はありません。読みやすさと保守のため、LDmicro 1.1 の接頭辞の慣習を引き継ぐことを推奨します（本マニュアルの例はすべてこの命名に従っています）。

接頭辞	用途	宣言例	説明
X	入力	`Xstart : BOOL;`	物理入力にマッピング（I/O マッピング）
Y	出力	`Ylamp : BOOL;`	物理出力にマッピング
R	内部フラグ	`Rrun : BOOL;`	自己保持、HMI ボタン、ステップフラグ
T	タイマーインスタンス	`Tdelay : TON;`	タイマーごとに 1 つのインスタンス
C	カウンターインスタンス	`Cparts : CTU;`	カウンターごとに 1 つのインスタンス
M	数値	`Mposition : INT;`	位置、設定値、測定値

C.2　命名規則

- 使用できるのは英字、数字、アンダースコア _ のみで、数字で始めることはできません。大文字と小文字は区別されません（`Xstart` と `XSTART` は同じ変数です）。
- IEC の予約語（`TON`、`IF`、`END_VAR`...）と同じ名前は使用できません。1.1 からインポートする場合、このような名前には `_v` が付加されます。`__` で始まる名前は 86LDmicro の内部用に予約されています。
- **GVL** で宣言した変数は、すべての POU、86HMI、C/C++ から参照できます。POU の `VAR` で宣言した変数はその POU でのみ使用できます（他の POU からは `プログラム名.変数` で読み取ります）。
- 同じ名前は GVL 内で一度しか宣言できません。

C.3　データ型と範囲

型	サイズ	範囲	C の型
BOOL	1 ビット	FALSE／TRUE	`bool`
SINT	8 ビット	-128～127	`int8_t`
USINT	8 ビット	0～255	`uint8_t`
INT	16 ビット	-32,768～32,767	`int16_t`
UINT	16 ビット	0～65,535	`uint16_t`
DINT	32 ビット	-2,147,483,648～2,147,483,647	`int32_t`
UDINT	32 ビット	0～4,294,967,295	`uint32_t`
LINT	64 ビット	-9,223,372,036,854,775,808～9,223,372,036,854,775,807	`int64_t`
ULINT	64 ビット	0～18,446,744,073,709,551,615	`uint64_t`
BYTE／WORD／DWORD／LWORD	8／16／32／64 ビット	ビット列（符号なし）	`uint8_t`～`uint64_t`
REAL	32 ビット	IEEE 754 単精度、約 ±3.4E38、有効桁数約 7 桁	`float`
LREAL	64 ビット	IEEE 754 倍精度、約 ±1.8E308、有効桁数約 15 桁	`double`
TIME	32 ビット	ミリ秒、±2,147,483,647 ms（約 ±24.8 日）、定数の最大値は T#24d20h31m23s647ms	`int32_t`
STRING／STRING[n]	n＋1 バイト	文字列。既定値 80 文字、最大 254 文字	`char[n+1]`
列挙型（TYPE … (A, B, C)）	32 ビット	宣言した値	`int32_t`

配列（ARRAY[1..5] OF INT）、構造体（STRUCT）、列挙型は「Data types」で定義します（4.5 節）。86LDmicro 2.0.0 の IEC 実行環境は、WSTRING、DATE、TIME_OF_DAY（TOD）、DATE_AND_TIME（DT）、POINTER、REF_TO にはまだ対応していません。

***注意：**値が型の範囲を超えると、チェックで「the value … does not fit INT」のようなエラーが出ます。カウントや積算が 32,767 を超える可能性がある場合は DINT を使用してください。

C.4 定数と TIME の書き方

種類	書き方	値
TIME	`T#500ms`、`T#2s`、`T#1.5s`、`T#1m30s`、`T#1h30m`、`T#1d`	単位 d、h、m、s、ms は組み合わせ可能。数値には小数点とアンダースコアを使用できる
TIME（完全な書き方）	`TIME#2m15s`	`T#2m15s` と同じ
16 進数、2 進数、8 進数	`16#FFFF`、`2#1010_1010`、`8#777`	65535、170、511
型の指定	`INT#-5`、`REAL#1.5`	その型の定数
実数	`1.5`、`1.5E3`	1.5、1500.0
文字列	`'hello'`	シングルクォーテーション
ブール	`TRUE`、`FALSE`	

TIME と整数の変換には `TIME_TO_DINT(t)`（ミリ秒）と `DINT_TO_TIME(ms)` を使用します。時間の比較は時間同士で行います（例：`IF Fill.T >= T#5s THEN`）。宣言例（上記の書き方はすべてチェックとシミュレーションで確認済み）：

```
VAR
    tFill   : TIME := T#1m30s;     (* 90000 ms *)
    tPulse  : TIME := T#1.5s;      (* T#1s500ms *)
    Mask    : WORD := 16#FFFF;
    Pattern : BYTE := 2#1010_1010;
    Total   : LINT;
    Ratio   : LREAL := 1.5E3;
    Msg     : STRING[20] := 'hello';
END_VAR
```

お問い合わせ

本マニュアルに沿って操作する際に技術的な問題が発生した場合、または QEC シリーズ製品の購入やカスタマイズのご要望がある場合は、以下の窓口までお問い合わせください。

項目	連絡先
営業／購入	info@qec.tw（QEC 製品ライン）
技術サポート／研究開発	info@icop.com.tw（ICOP Technology Inc.）
オンラインドキュメント	https://www.qec.tw/86duino/86ldmicro/

86LDmicro の「Help」メニューからオンラインドキュメントサイトを直接開けます。最新のユーザーマニュアルと説明文書はウェブサイトを参照してください。