はじめてのIEC61131-3

―PLCの国際標準プログラミングのすすめ―

PLCopen Japan

まえがき

制御技術は，身近なエレベータから複雑な原子力発電所まで，あらゆる分野で使われている技術であり，一般にフィードバック制御とシーケンス制御に大きく分けられるといわれています。フィードバック制御は温度制御や速度制御などのように個別の物理現象を対象としているのに対して，シーケンス制御はランプの ON/OFF 制御のような単純なものから自動車の生産ラインのような大規模で複雑なシステムまでありとあらゆるものを対象にしています。このため，フィードバック制御をシーケンス制御の一部として捉える考え方もあります。

シーケンス制御は PLC（Programmable Logic Controller）という制御装置を使って行うのが一般的です。PLC はコンピュータを産業用に特化して設計された制御装置であり，現在では安価で大量に製造でき，また，その機能もコンピュータと同様に高度化・高機能化しています。このため，シーケンス制御技術は PLC 技術といっても過言ではありません。しかし，PLC のハードウェアの高度化・高機能化に比べて，現場での利用技術すなわちソフトウェアを作成するためのプログラミング技術はあまり進んでいないのが現状です。

PLC のためのプログラミング言語の国際規格は既に 20 年前に制定されており，これに対応した PLC も数多く誕生していますが，これらを有効に活用できているとは到底いえません。その原因の１つとしてはプログラミング言語の複雑さにあるといわれています。国際規格をよく理解し，PLC の機能を有効に活用し，さらに再利用やプログラムの可搬性の実現もしたいという使用者の要望が多く寄せられています。

PLCopen Japan は３年前に日本配電制御システム工業会と協力して，「やさしい国際標準 PLC」という解説本をその１つの切り口として刊行しました。制御技術とその周辺技術も日進月歩の進化を遂げていますので，日本配電制御システム工業会のご好意を得て，内容を加筆･再編集した上で本書として刊行することといたしました。

本書の刊行にあたりましては，関係各位に深謝致すとともに，読者の皆様におかれましては，本書を実践でご活用いただき，また，忌憚のないご意見やご批判をお寄せいただきますようお願い申し上げます。

2011 年 3 月 22 日

PLCopen Japan

チェアマン　宮澤 以鋼

目　　次

第 1 章 制御システム開発の技術的課題解決のために

1.1 PLC システムの市場動向

制御システムは、マイコンを搭載した機械の専用のコンロトーラによるものから、大型プラントの制御に使用する DCS(Distributed Control System)によるものなど千差万別である。制御方式は、シーケンス制御とフィードバック制御に大きく分けられ、制御の対象となるのは、ほぼ全ての機械やプラント類であるが、現在これらの制御の主力デバイスは、PLC(Programmable Logic Controller)になっている。

PLC は 1970 年代の半ばに、リレーによるシーケンス制御をミニコンやマイコンに置き換えることを目的に開発されたシステムであり、リレーシーケンスの知識で使いこなせるコンピュータシステムとして現場技術者の支持を得て普及してきた。

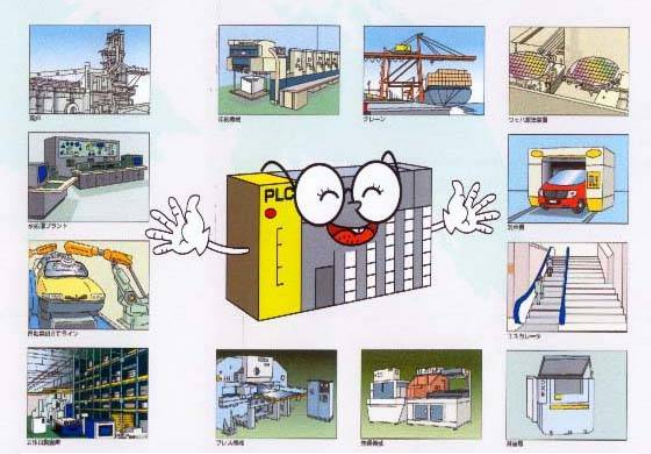


図 1.1 拡がる PLC アプリケーション
（出展:JEMA 資料より抜粋）

PLC はコンピュータの最新技術を取り込むことで年々進歩しており、小型化やローコスト化とともに、性能面では 1 命令数 ns レベルの超高速演算性能を、機能面では高級なプログラミング言語をサポートするに至っている。この結果、小は組み込み用マイコンボードの代替から、大は DCS の代替まで PLC が使われてきている。これに伴い、従来はシステムハウスや大手エンジニアリング会社の仕事であったエンジニアリング・プログラミング業務が、現在は盤メーカや機械・装置メーカに拡大してきている。

図 1.2 に(社)日本電機工業会(JEMA)発表の、1985 年から 2011 年までの国内 PLC 市場の生産量及び生産台数の推移グラフを示す（2010 年、2011 年は予測）。1990 年前半のバブル経済崩壊から 2000 年迄はほぼ横ばいであったが、その後 2007 年までは、生産高、生産台数とも急激に上昇しているのがよくわかる。2008 年度後半から 2009 年度前半には、世界的な自動車産業の低迷影響を受け急激に落込んだが、2010 年度に入り、アジア圏を主とした輸出が好調に転じ、PLC 市場は回復しつつある。別の米国の調査会社 ARC 社の発表では、今後 5 年間の世界の PLC 市場は平均年率 6.5%程度で成長すると予測されている。最近の PLC 利用の分野で特に話題となるのは、

(1)モーションコントロール・・・専用コントローラの PLC による代替
(2)計装制御分野・・・・・・・・DCS 計装の PLC による代替
(3)機能安全システム

などである。いずれの分野も高度なエンジニアリング技術力が要求され、複雑なプログラムの開発が求められる。これを裏づけるものが、図 1.3 である。1996 年度から 2006 年度までの 10 年間で、PLC 出荷台数の伸びが 1.7 倍だったのに対し、出荷メモリ容量が 3.5 倍も伸びている。ハ

ードウェアの出荷台数の伸び率の 2 倍のペースで、PLC に書き込むプログラムの量が増えていることを示しており、PLC 制御システムの機能の高度化・複雑化を反映していると見ることができる。制御システムの世界では、このような勢いで PLC 制御システムの需要が急増しているのである。

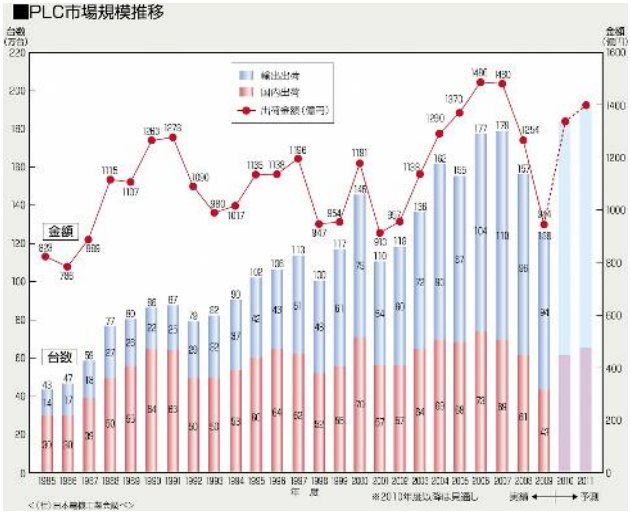


図 1.2 日本の PLC 生産動向
（出典: JEMA）

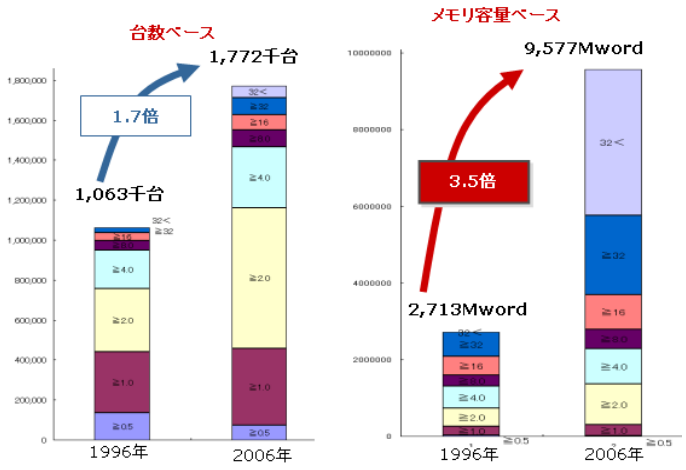


図 1.3 増え続ける PLC のプログラミング量
（出典: JEMA）

1.2 PLCopen Japan と JSIA の共同研究

前述の市場背景の中、急速に高度化、複雑化する制御システムの市場要求に応え、市場を支え成長させる為には、技術レベルの向上が不可欠である。

2006 年秋よりPLCopen Japan とPLCの国内最大ユーザ団体である社団法人日本配電制御システム工業会（以下JSIA という）は、技術レベルの向上を目指し、制御システムメーカの技術課題の調査とその解決策についての共同研究を実施した。この成果を共同研究報告書【PLC制御システムの合理化とIEC 61131-3：副題～制御システムメーカの技術課題解決の為に～】として発表し、これにJSIA会員からの意見を反映したものを一般図書「やさしい国際標準ＰＬＣ－制御システムの技術的課題解決のために－」として再編集し、2008年12月に出版した（～2010年10月完売）。

1.3 JSIA 会員企業の制御システム事業における技術的課題

JSIA の制御情報システム部会では、前段で述べたような、将来に向けて拡大基調にある制御システム事業を進めるに当り、どのような技術的な問題・課題を抱えているか、JSIA 会員企業にアンケート形式で調査を行った。その結果の要約を図 1.4 に示す。

会員の殆どは PLC を使った制御システムの大半にラダーダイアグラムを

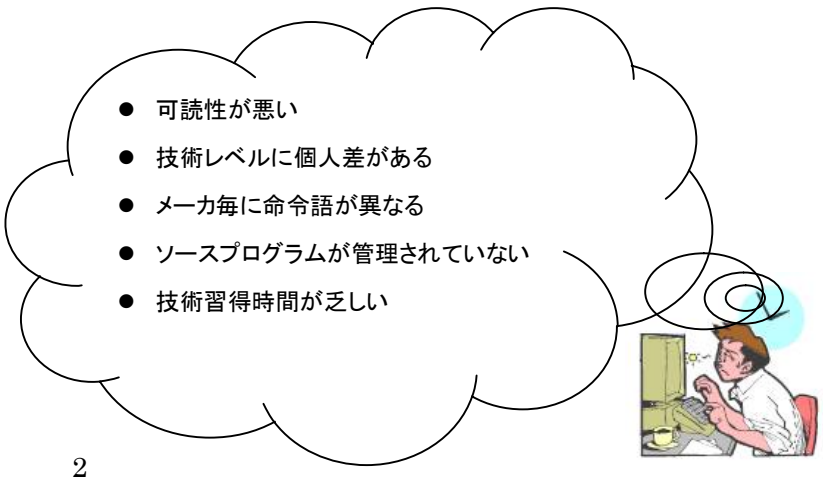


図 1.4 JSIA 会員企業の制御システム事業における技術的課題

採用し、しかも PLC メーカ固有の方式でシステムを構築している。調査結果は、この状況から生まれた問題・課題である。そして、その課題を整理し関連付けをしたのが図 1.5 である。

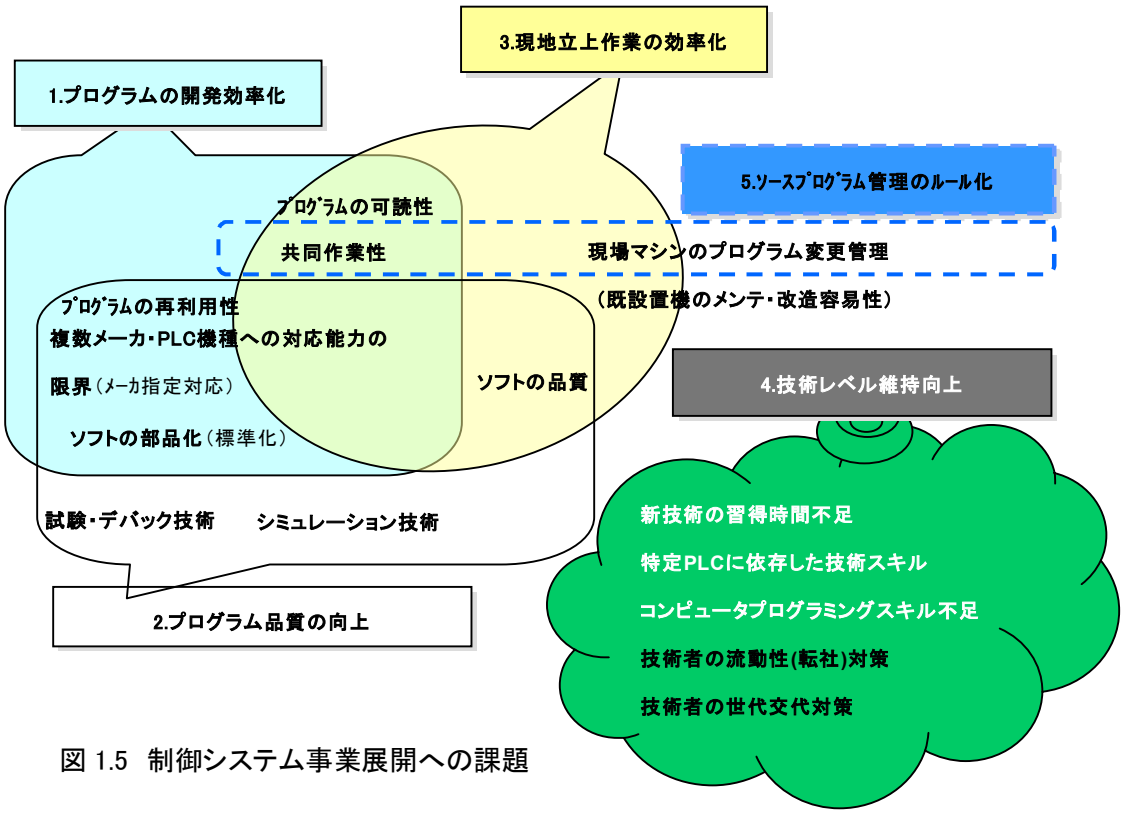


図 1.5　制御システム事業展開への課題

1.4　IEC 61131-3 規格勉強の意義

制御情報システム部会では、これらの問題を解決する手段として、PLC プログラミング言語の国際標準である IEC 61131-3 規格に着目した。そしてこの IEC 61131-3 規格の普及促進を手がける非営利国際団体 PLCopen の日本支部である PLCopen Japan の協力を得て、調査研究を実施した。

JSIA の観点から IEC 61131-3 規格に見られるポイントは、次の 3 点と考えている。

（１）言語の選択が自由

第一は、5 つの言語をサポートしていることである。プログラムの目的、プログラマの得意不得意に合わせて、自由に言語が選択でき、複数の言語を組み合わせて 1 つのシステムを構築することが可能である。日本や米国では、リレーシーケンス回路をマイコンに置換えるという発想から、リ

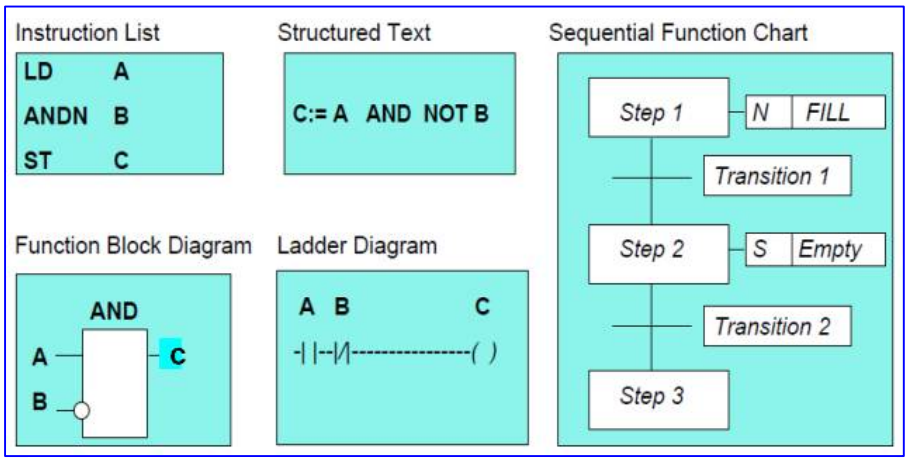


図　1.6 IEC 61131-3 で規格化されている 5 種類のプログラミング言語

レー回路図に似たラダーダイアグラム（ＬＤ言語）が一般的となった。ところが、時代が変わり、PLC に求められる機能が複雑化、大型化してくると、ＬＤ言語だけではなく、他の形式の言語が求められるようになった。つまりＬＤ言語に加えて、数値計算に都合がよいパソコンに使うパスカルや BASIC、C 言語に形式の近い ST 言語(Structured Text)、或いは機能を一つずつブロックとしてまとめ、あたかも機能部品である LSI を電子回路上で接続するようなイメージで表現してゆく FBD 言語(ファンクションブロックダイアグラム)である。更に IEC 61131-3 規格には、工程進捗を判りやすく表現できる SFC 言語(Sequential Flow Chart)や、マシン語に近い IL 言語(Instruction List)と呼ばれる言語を加えた全 5 種類のプログラミング言語が取入れられている。

（２）PLC 固有のメモリアドレスを意識しないプログラミング作業

第二の点は、変数（信号名）によるプログラミングである。変数それぞれについて、ビットやワードといたデータ型や揮発・不揮発といったメモリの機能を定義する方式を IEC では採用している。このため PLC のメーカや機種に依存しないプログラミングが可能である。また、変数には PLC システム全体で共有する「グローバル変数」と小さな特定のプログラム内（POU という）のみで使用する「ローカル変数」の 2 種類があり、これらを使い分けることによって、ソフトウェア部品を組み合わせて目的のプログラミングを簡単に行うことができる。ファンクションブロックは自社内でソフト部品を資産化（標準化）する際にとても便利であり、納期管理、生産管理の面でも大変有効である。

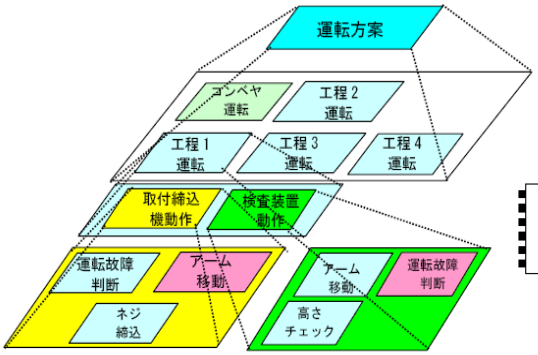


図 1.7　構造化プログラミング手法の概念

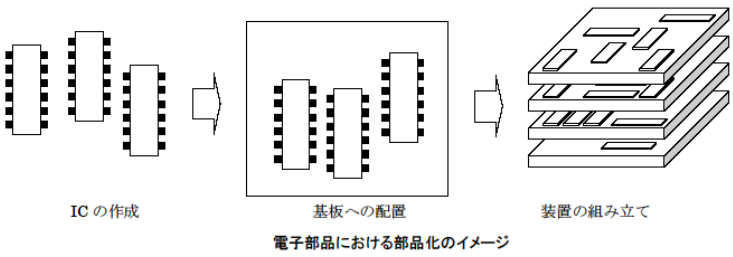


図 1.8　ソフトウェア部品化のイメージ

（３）プログラムの階層化・モジュール化が容易

そして第三の点は、POU、タスクと言った概念を使うことにより、またファンクションブロックを使ったソフトウェア部品を豊富に準備することにより、プログラムの構造化、モジュール化が容易となる。膨大なプログラミング作業を、効率よく行うため、構造化プログラミングはとても重要なプログラミング手法である。また、その構造化プログラミングのベースとなるものがソフトウェア部品であり、ソフトウェア部品はファンクションブロックにより、より作り易く・判り易く・管理しやすくなる。（図 1.7 および図 1.8 参照）

この様にソフトウェアの標準化が進み、実績のあるソフトウェアが部品として蓄積・資産化されてくると、製作工期は短縮でき、何よりもソフトウェアの品質（信頼性）が大幅に向上される。オーダーメードなソフトウェア製作の仕事であるにもかかわらず、採算性の高い事業に変えることが可能となる。

第 2 章 PLC システムのコスト構成の変化と IEC-PLC による エンジニアリングコストの低減

2.1 PLC システムのコスト構成の変化

PLC はコンピュータ技術の産業応用としてその進歩と共に発展してきた。この発展の過程は5年から10 年の遅れでコンピュータ技術が PLC に応用され実現される歴史を繰り返してきた。PLC のプロセッサの処理性能は、代表指標である命令処理時間でみるとここ 10 年で1桁短縮し(基本命令処理時間 20nS 以下)、さらに浮動小数点演算の機能も備えるまでになっている。

PLC の高性能化はその応用面で大きな変化をもたらしている。従来高価な専用コントローラで行っていたモーション制御や計装制御を、PLC のソフトウェアに置き換える動きが活発である。その結果、機械装置のコストの中に占めるソフトウェアの比率は年々上昇しており、最近の調査では装置コストの 40%に達している。機械装置のコスト削減や品質向上は「ソフトウェア」を抜きに語れない状況である。

この調査は国際的な機関によるものだが、第 1 章の技術的課題の調査結果で示したとおり、日本にも当てはまる。

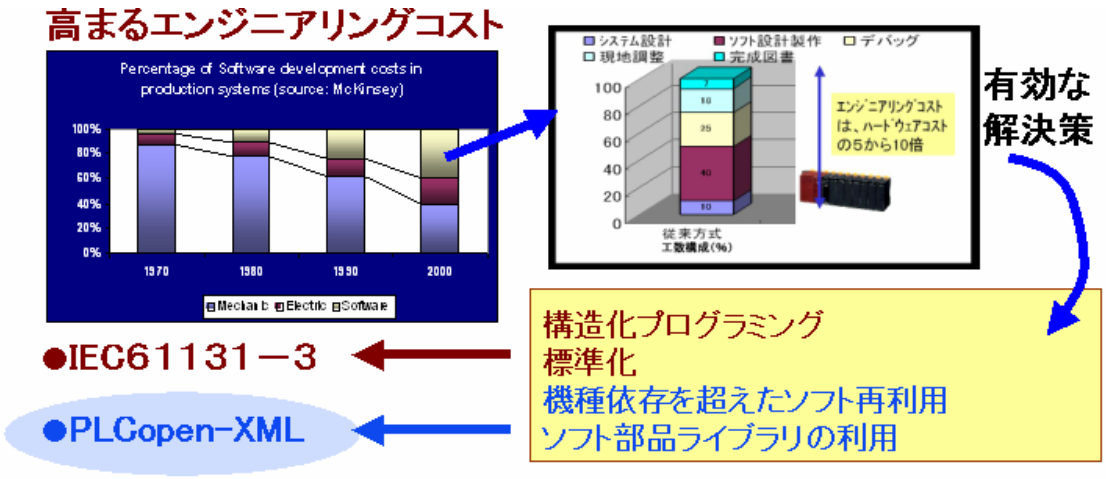


図 2.1 PLC システムのコスト構成の変化とエンジニアリングコストの低減策

2.2 PLC ソフトウェアの作成コスト削減と品質向上

コンピュータのソフトウェアでは、「標準化(部品化)・構造化はわかりやすく、高品質で、かつソフトウェア開発効率を高める効果がある」ことが広く実証されてきた。

しかし、限られたハードウェア資源から如何に高い性能を絞り出すかが「プログラマーの腕」と考えられていた日本の PLC の世界では、いまだ PLC ハードウェア依存で作成者本人以外は解読できず、再利用やメンテナンスが困難なソフトウェアが横行している。

つまり、ハードウェアの進歩にソフトウェアが追随できていないのが現状である。日本で多く使用されている PLC のラダープログラミングは、マシン語に近く極めてプリミティブなものなので、進化したコンピュータのソフトウェアの技術が導入されていない。

IEC 61131-3「PLC プログラミング言語」は PLC のプログラミングに関する唯一の標準規格で、一般にはそのタイトルから単なるプログラミング言語の規格と思われがちだが、プログラムの表記や文法などの言語と共にリソース（PLC）やプログラムの構成要素、変数などの定義を標準化したものである。

現在日本では JISB3503 として、中国では GB/T 15969.3 としてそれぞれの国家規格として制定されている。

IEC 61131-3 はコンピュータ技術の進化を予測し、「メーカや機種に非依存」、「ソフトウェアのモジュール化（部品化・再利用）」、「用途や技術者のスキルに応じて使える言語（シーケンス技術者向き LD 言語、コンピュータ技術者向き ST 言語、計装技術者向き FBD など）」をコンセプトに 1993 年に初めて制定された。

ラダーシーケンス技術者の多い日本ではその理解に多くの時間を必要としたが第 1 章で紹介したとおり、2005 年フィンランドのヘルシンキで開催された第 38 回技能五輪国際大会のメカトロニクス種目で日本のチームが、IEC 61131-3 準拠 PLC を使い、金メダルを獲得したことで、IEC 61131-3 の有用性の認識が一挙に広まった。

しかし、現場の制御システム設計者の IEC 61131-3 についての理解は進んでいるとはいえない。制御技術のスキルを持つ技術者は日々の業務に追われ、最新のコンピュータのプログラム技術や IEC-PLC の学習の機会を逸しており、効率的なプログラミングを実現する IEC-PLC を活用できていないのが実態である。

IEC-PLC の学習・導入は「IEC-PLC の導入によって自分達の仕事がどのくらい楽になるかを知ること」と「短時間で技術習得できる環境の整備」が鍵となる。

このため本書では、第 1 章の技術的課題の調査結果を基に PLC 使用技術者と PLC メーカ技術者が意見交換し、PLC 使用者の視点でできるだけわかりやすく以下の項目に分けてまとめた。

第 1 章　制御システムメーカの技術的課題解決のために
　　―――――― 今後の課題「技術者不足、技術者世代交代対応」
第 2 章　PLCシステムのコスト構成の変化と
　　IEC-PLC によるエンジニアリングコストの低減
第 3 章　従来PLCとIEC-PLCの違い　　導入編
第 4 章　IEC- PLC によるプログラムの作成
第 5 章　ソフトの部品化について
第 6 章　構造化設計と IEC-PLC によるプログラミング
第 7 章　実際のプログラム開発と共同作業　　実用編
第 8 章　デバック、試験検証の効率化
第 9 章　　PLCソフトによるモーション制御について
第 10 章　異メーカ・異機種 PLC 間でのソフト相互利用について
第 11 章　ソースプログラム管理について　―――― ISO9000 品質マネジメント対応

第 3 章　従来 PLC と IEC－PLC の違い

3.1　「従来 PLC」、「IEC－PLC」について

IEC61131-3 というと、5 言語（正しくは IL, ST, LD, FBD の 4 言語と SFC の 1 要素）が注目され、プログラミング言語の規格と思われがちだが、第 2 章に述べたとおり「プログラムの表記や文法などの言語と共に PLC やプログラムを構成する要素、変数などの定義を標準化したもの」である。本章では従来 PLC と IEC-PLC の違いについて解説する。

本書では、従来から日本で多く使われている LD 言語（ラダー言語）を中心にプログラミングを行い、あらかじめメモリマップ上に入出力、補助メモリ、タイマ・カウンタ、ファイルなど用途毎に領域が割り付けられている PLC を、「従来 PLC」と呼ぶ。

一方、「IEC-PLC」は、変数名プログラミング、ファンクションブロック、ST 言語、FBD 言語など IEC61131-3（以下、IEC 規格）を積極的に活用したプログラミングスタイルをとり、さらにパソコンアプリケーションの開発言語と同様に特定の PLC のメモリマップを意識せず（＝PLC 機種依存が少なくなる）プログラミングツールで変数（従来 PLC 用語ではラベル・信号名に相当）等を定義して使用するものを「IEC-PLC」と呼ぶことにする。

本書は、できるだけ「従来 PLC」のユーザの視点からみて記述し、「IEC-PLC」を理解しやすくなるよう務めた。

「従来 PLC」のユーザが「IEC-PLC」に触れたとき最初に戸惑う点を、プログラミングの手順（図 3.2 の左側を参照）に沿って挙げると表 3.1 のとおりになる。この表は、主に本章の内容をまとめたものであり、IEC-PLC の用語に慣れていない場合は、最後に読んでいただいても差し支えない。

従来 PLC に比べ IEC-PLC は、プログラミング作業に入る前の設定が多く使いにくいという声をよく聞くが、実は IEC-PLC の場合、特に作業が増えているわけではない。作業イメージを図 3.1 に示すが、設計者は使用する PLC のメモリマップを頭の中にイメージ（即ちこれが定義）し作業しているのに過ぎない。

したがって慣れ親しんだ PLC を使う場合は問題ないが、従来 PLC のメモリマップ・命令などは PLC 機種依存であるため機種やメーカが異なると作業が困難になる。メモリマップ・命令の仕様が異なるため、これらを設計者が全て覚えて使い分けることは難しく、また、プログラムの流用、部品化が容易ではないからである。

一方 IEC-PLC は、設計者が PLC を独自に定義するため、機種・メーカに依存度の低いプログラミングができる。さらにデータ形、真の部品化を実現するファンクションブロック、制御用途に適した言語セット等により、プログラムの開発効率と信頼性の向上が可能である。

従来 PLC の場合は、①不慣れな PLC メーカ指定があった場合の対応に困る。②経済合理性が追求できる PLC のマルチベンダ化が採用できない、などの障害があるが、IEC-PLC を使うとこれらが解決できる。

3.2 項以下に「PLC プログラミングの課題」と対比し、これらについてわかりやすく解説する。

表 3.1 最初に感じる「従来 PLC」と「IEC−PLC」の主な違い

	主な作業内容	従来 PLC での作業・機能	IEC-PLC での作業・機能
定義作業	ハードウエア構成情報の定義	設計作図作業で実施。プログラミングツール（以下ツール）としての機能はない。	ツールで構成情報を定義。プロジェクトとしてプログラムとハードウエア構成を一括管理し一意性を確保。
	I/O メモリ、変数の定義	PLC 固有のメモリマップ（絶対アドレス）毎に、ラベル・信号名を決める。これが変数の定義にあたる。プログラム内で使用する補助メモリも割付がいる。	PLC 固有のメモリマップは意識せず、変数を定義（変数名、変数の形など）。絶対アドレスは入出力などのみ定義。プログラム内で使用する変数は、メモリ割付は不要（ツールで自動処理）。
	変数の形を定義	用途に合わせて、メモリマップからメモリの種類で形を選択割付。命令種類、データにより形を常に意識する必要がある。	変数毎にツール（エクセル表などでも）で変数の形を定義する。違った形同士の演算を防ぎプログラムの信頼性が向上する。PLC の機種・ベンダ非依存。
ソフト作成	応用命令の使用	PLC のベンダ・機種ごとに提供された固有の応用命令を理解して使用。	ファンクションブロック（FB）、ファンクョン（FUN）がこれに当たる。 使用者が独自のFB、FUNを作りソフト部品のライブラリ化もできる。
	ソフトウエアのブロック化、機能単位のブロックに分けてのソフト作成	機能単位のソフトブロックをサブルーチンで処理するなど。しかし完成したプログラムは巻物状になり、作成者以外の可読性は低くなる。	最小単位のソフトブロックは POU と呼ばれる単位でソフト作成する。 POU 内プログラムで使用できる変数には内部だけで使うローカル変数と（POU の外からは見えない）、プログラム全体で共通に使えるグローバル変数がある。 ローカル変数、アドレス定義不要と相まってソフトの部品化・再利用性が向上。 さらに大きなソフトウエアのブロックは、FB、FCTをFBD言語で組み合わせて構築する。
	割り込み処理等プログラムの処理順設定	PLC のベンダ・機種ごとに提供された固有割り込み命令、定周期実行指定命令等により実行方法を定義。	タスクの概念があり、プログラムをデフォルト・定周期・イベントタスクに割り当てればよい。
	言語の選択	ラダー言語を中心としているが最近は ST 言語、FBD言語などの利用可能なものもある。	IEC5 言語をシーケンス処理、データフロー処理、数値演算など処理内容に応じて自由に使い分けられる。
転送デバッグ	PLCへの転送（ロード）	ツールでソースプログラム作成後にPLCに転送処理する。コンパイルは意識しない。STなど高級言語をサポートしている機種はツール内で転送処理中に実際にはコンパイル処理をしている。	ソースプログラム作成後にコンパイル処理（文法など各種チェックと実行コードへの変換）を行った後に、PLCに転送する。
	PLCからのアップロード	ツールでPLC内の実行コードを吸い上げて、ソースプログラムを再現できる。STなどコンパイル処理をしているものはできないが、ソースプログラムをPLC内に格納し見かけ上できるようにしている。	ツールでPLC内の実行コードを吸い上げても、ソースプログラムを再現できない。ソースプログラムを圧縮してPLC内に格納し見かけ上できるようにしているものもある。。

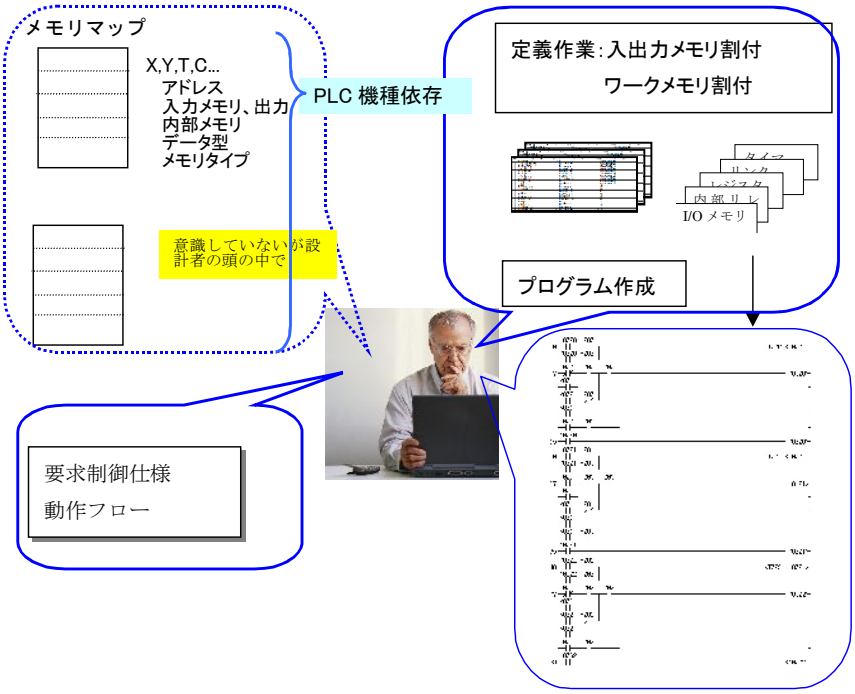


図 3.1　従来 PLC によるプログラミング

3.2　PLC プログラミングの課題

PLC のプログラミングにおいて、広く普及しているのはラダー言語である。しかしながら、ラダー言語（LD 言語：Ladder Diagram）を中心としたプログラミングスタイルでは、エンジニアリング効率という観点からいくつかの課題が指摘されている。

それらの課題は、各 PLC メーカのプログラミングツールの機能・言語仕様、プログラミングする技術者の技量により異なるが、プログラム開発の各工程で、概ね次の図 3.2 に示す項目があると考えられている。

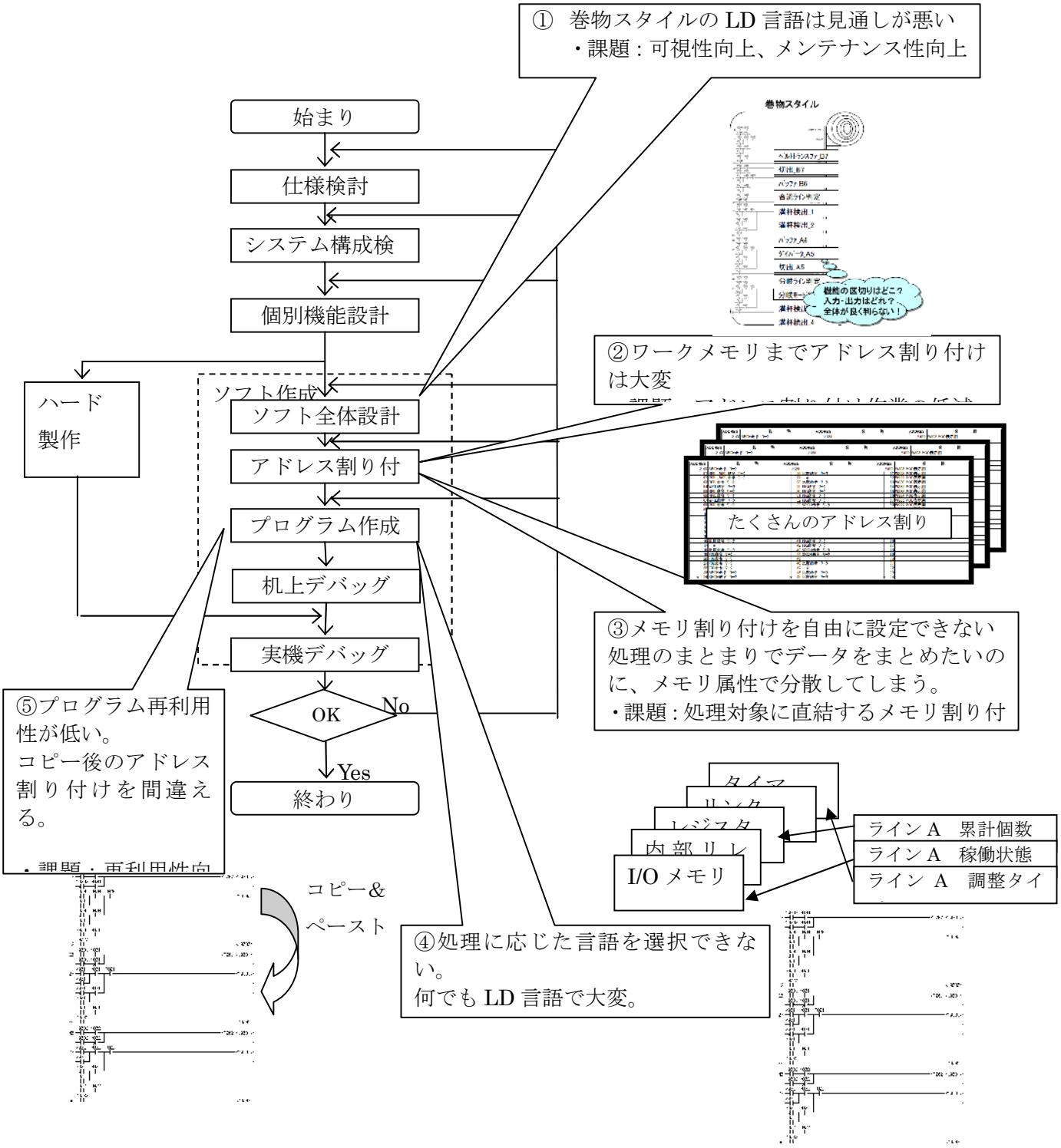


図 3.2 ソフト開発工程から見る PLC プログラミングの課題

以下、それぞれの課題について述べる。

(1)プログラムの可視性

LD 言語は図 3.3 の左側のように巻物スタイルに例えられる。機能単位、入出力処理単位でプログラミングしても、一目見て区別がつきにくく、他人が見てわかりやすいプログラムを作るのは容易ではない。設計者がソフトの部品化をしてプログラムを作っても、完成したプログラムから第三者が部品を意識して読み取ることは困難である。

処理の単位が一目で見て判別できるプログラムの表示機能があれば、プログラム全体の見通しがよくなる。例えば、下図の右側のようにプログラムを表示できれば、巻物とはちがってプログラムの全体構造が一目でわかるようになる。

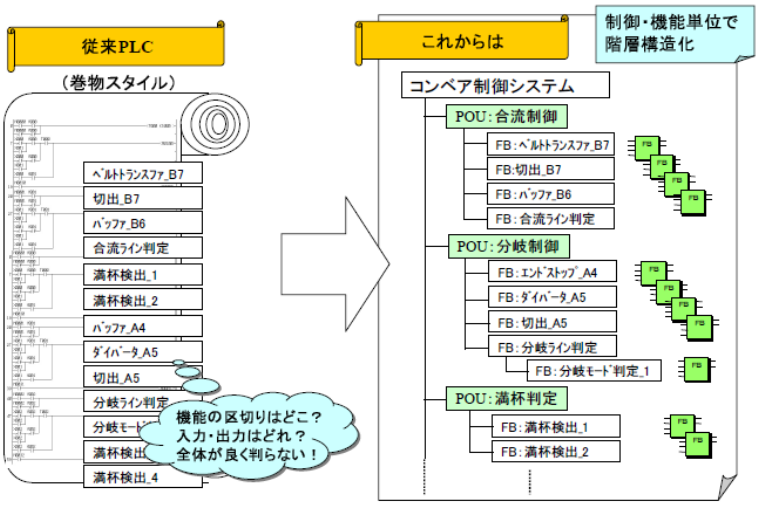
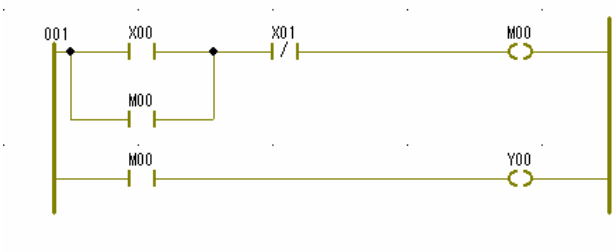


図 3.3　プログラムの見通しをよくする

(2)ワークメモリのアドレス割り付け作業

PLC は入力から信号・情報を取り込み、演算を行い、演算結果を出力する。演算の途中経過、あるいは演算結果を保持するワークメモリのために、内部リレー、レジスタ等を使うが、PLC のメモリアドレス割り付けが必要である。入出力は PLC の外界との信号授受のためにアドレスが必要なのは当然であるが、ワークメモリについては、アドレスは不要でなければならない。

例えば、図 3.4 の自己保持回路で、内部リレーM00 は自己保持を実現するためのワークメモリとして使われている。自己保持のワークメモリは入出力に直接接続されないが、PLC システムの都合で、内部リレーとしてのアドレスを割り付ける必要がある。このようなワークメモリは、メモリの目的を名称とした名前だけで、アドレスを気にせずプログラミングするのが本来の姿である。



アドレ	コメント
X００	入 PB
X０１	切 PB
Y00	駆動源
M00	自己保持ワーク

このアドレス割り付けは、本来は無駄な作業

図 3.4　自己保持回路

機械・装置の高度化・高機能化に伴い、PLC プログラムは大規模化している。これに伴い、内部リレー、データレジスタ等のメモリの使用量も大容量化している。中規模サイズの PLC は、数万点の内部リレー、データレジスタを持つ。数万点のアドレスの割り付けが不要になるだけでも、作業時間を短縮し、割り付けミスによる異常動作の防止が可能である。

(3)メモリ割り付けの自由な設計

プログラムの処理で使うメモリは、有意な固まりで存在する。例えば図 3.2③では、稼働状態(運転中、停止中など)、生産累計個数、調整タイマのような、特定のラインに関するデータが存在する。一方で、それらのデータは、電源断でも保持するもの、タイマ値を示すものといったように機能が異なる。

従来 PLC は機能の異なるデータを格納するために、メモリを区別して用意しているので(メーカお仕着せの仕様)、本来ならまとめておきたいデータが、分散してしまうのである。

この問題を解決するには、アドレスではなくデータの名前でプログラミングし、それぞれのデータに機能を割り当てればよい。例えば図 3.5 のようになる。

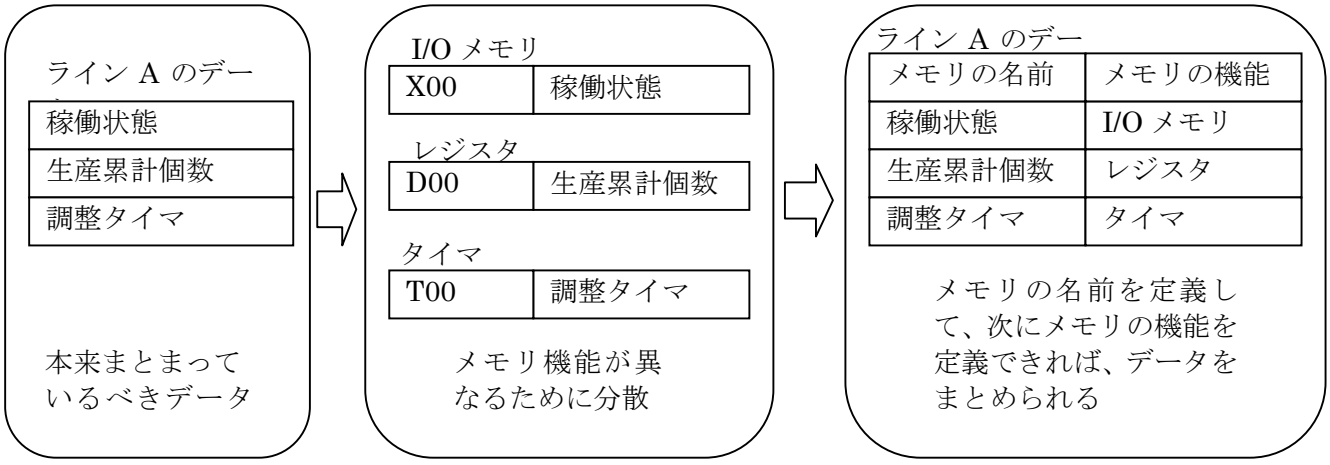


図 3.5　メモリ割り付けの問題と解決策

(4)処理に応じた最適な言語選択

PLC はシーケンス制御、データ処理、工程制御など様々な処理を行う。LD 言語はシーケンス制御向きの言語であるが、シーケンス制御以外の処理にも使われているのが実情である。図 3.6 は多項演算の例で、LD 言語で記述した場合と、データ処理に向いた ST 言語(Structured Text)で記述した場合の比較を示す。

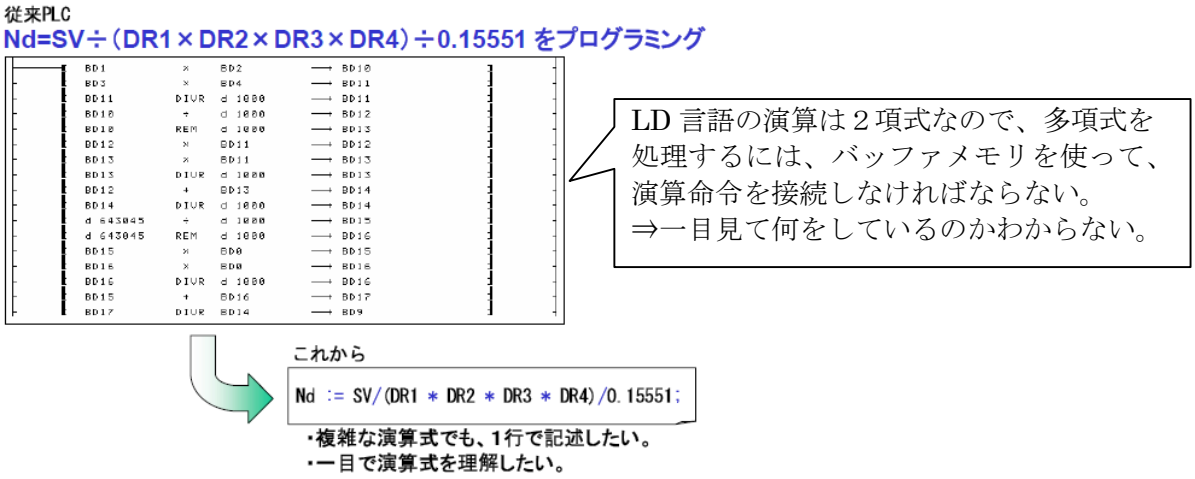


図 3.6　多項式の演算：LD 言語と ST 言語

このように用途に応じて言語を選択できれば、プログラミング効率は向上する。詳しくは 3.3 項で述べるが、IEC 規格では、用途に応じて５つの言語を選択可能である。概要を図 3.7 に示す。

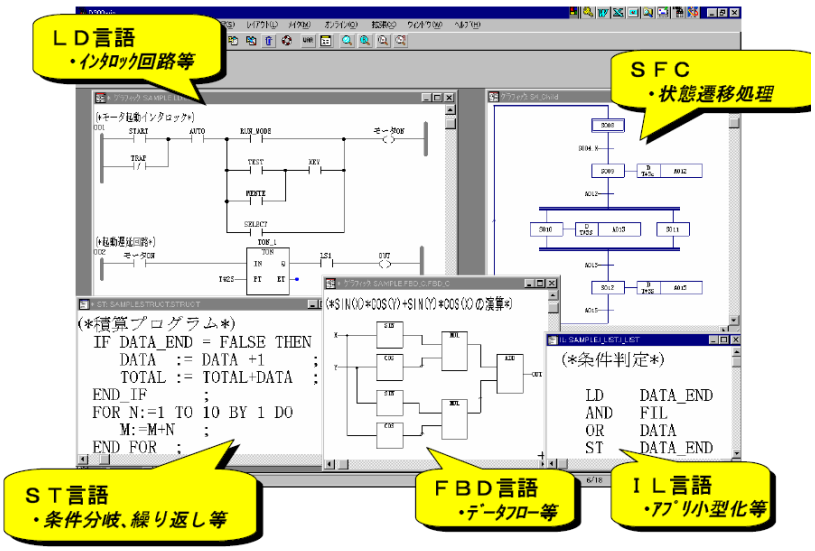


図 3.7　用途に応じ5言語が選択可能

(5)プログラム再利用性向上

プログラムには、同一パターンの回路が多数存在する場合が多い。この場合、回路を一つ作成後、残りはコピーして貼り付け、アドレスを変更するが、これがミスの元となる。

この場合、同一パターンの処理は一まとめに部品化して箱形の命令として表現し、箱の左右にそれぞれ入力、出力を割り付けるようにすれば、ミスが減る。ミスが減ることにより、プログラム再利用性が高まるのである。

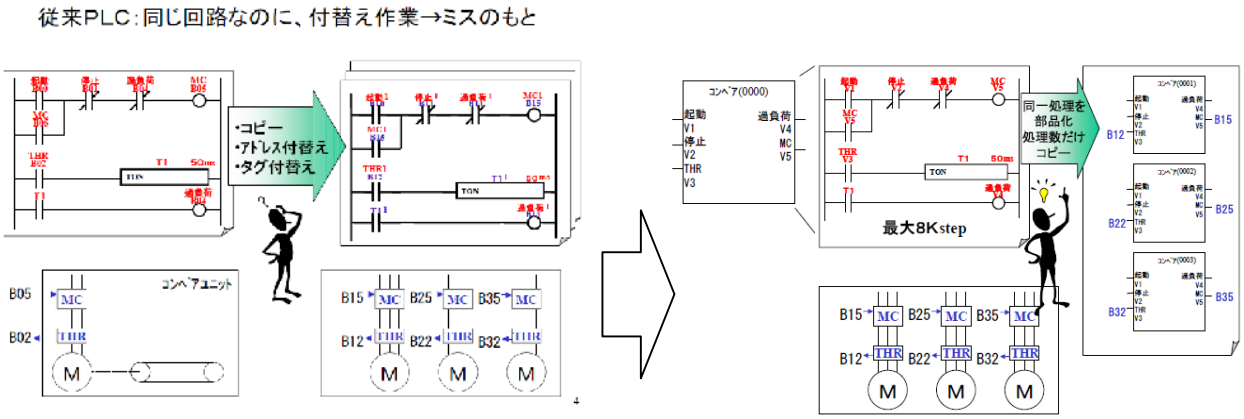


図 3.8　プログラム部品化によるプログラム再利用性の向上

3.3　IEC 言語プログラミング

IEC 規格では、処理に応じて LD 言語 、ST 言語、FBD 言語、IL 言語と SFC の5言語が用意されている。SFC は共通要素として言語とは規定されていないが、言語要素を持つので、本章では言語として扱う。

IEC－PLC では、なぜ5つもの言語が用意されているのか？

・ PLC の適用分野拡大に伴う多様な制御に対応

制御内容はシーケンス制御からモーション制御、データ処理、計装制御等多岐に渡っている。

それぞれの制御には、分野に応じた最適な言語が存在するが、IEC 言語に対応した PLC が商品化されるまで、ユーザは最適な言語を選ぶ余地が少なく、LD 言語で全てをプログラミングする状況が続いてきた。リレー接点、コイルの置き換えから発展した LD 言語は、当然のことながらシーケンス制御には向いているが、例えば多項式の演算のようなデータ処理には向いていない。機械・装置のインテリジェントが進む中で、ユーザプログラムにおけるデータ処理は増加傾向であり、LD 言語による開発効率の低さが顕在化している。

この様な状況において、IEC 規格に対応した PLC が商品化された。IEC 規格に準拠した PLC は、制御に応じたプログラミング言語が選択可能である。また、変数名プログラミング、プログラムの部品化などプログラミング効率化を実現する仕組みを持ち、LD 言語中心のプログラミングにおける課題を解決する能力を持つ。

・ 使用者の多様なスキルに対応

従来から、例えば組み立て機械分野の設計者は LD を、計測制御分野では FBD を、計算処理分野ではコンピュータ言語を使っていた。PLC の適用分野が拡大することにより、必然的にこれらに対応する必要があった。

以下、5言語について簡単に解説する。

(1)LD 言語 (Ladder Diagram)

LD 言語プログラム例を図 3.9 に示す。グラフィック上の表現は異なるところもあるが、IEC 規格の LD

言語は、従来からのLD言語と原理は同じであり、従来のLD言語と同じ考え方でプログラミング可能である。LD 言語はリレー回路と等価で、I/Oのインターロック処理などビットレベルの処理に向いている。

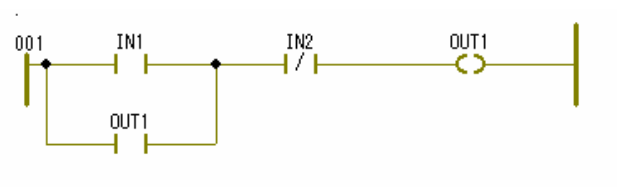


図 3.9　LD 言語プログラム例

一方でLD 言語は、システムの機能仕様の表現、例えばフローチャートで表現されるような仕様の表現ができない。フローチャートの仕様に基づく制御はできるが、実行レベルのLD 言語は、元来のフローチャートとは表現が乖離するためである。このため、プログラムを作成した人以外は理解できないといった状況に陥りやすい。

他人が理解できない属人化したプログラムは技術伝承が困難であり、技術力の向上・維持を妨げる。

(2)ST 言語 （Structured Text）

PASCAL と呼ぶコンピュータ言語をベースに設計された構造化テキスト言語である。数値演算式の表現、ネスティングした条件分岐などLD 言語が苦手とする用途で威力を発揮する。

```
if 高さ < 150.0 and 高さ > 100.0 then
   高さチェック:= False;
else
   高さチェック:= True;
end_if;
```

図 3.10　ST 言語プログラム例（条件分岐）

図 3.10 は条件分岐の例である。”高さ”が 100.0 を超え 150.0 未満であれば、”高さチェック”をFALSE（偽）とし、そうでなければ、”高さチェック”をTRUE（真）とする。”:＝”は代入を意味する。

```
RESULT:= (INPUT1 + INPUT2 ) / (INPUT3 - INPUT4);
```

図 3.11　ST 言語プログラム例

図 3.11 は、括弧付き演算の例である。”INPUT1”と”INPUT2” を加算した値を”INPUT3”から”INPUT4”を引いた値で除算し、結果を “RESULT” に代入している。LD 言語で多項式、括弧付きの式を処理するには、基本的には二項式に分解し、分解された式を、演算途中結果を一時的に格納するメモリで繋げる必要がある。このことが処理内容の可読性を阻害する。

(3)FBD 言語 （Function Block Diagram）

機能と機能間のデータや信号流れの表現が得意なグラフィカル言語である。図 3.12 にプログラム例を示す。図 3.11 の ST 言語と同じ処理である。

図の四角い箱は、ファンクションである。ファンクションの左側の変数は入力パラメータであり、右側の変数は演算結果である出力パラメータである。プログラム例では、ADD と SUB の演算結果を、結線により DIV に渡しているので ADD、SUB の出力パラメータ変数を定義していない。

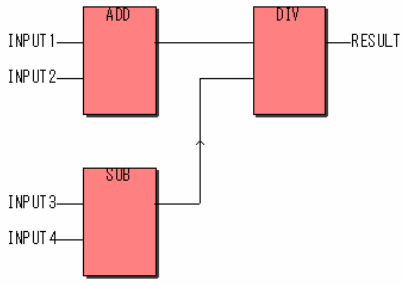


図 3.12　FBD 言語プログラム例

このように、FBD 言語は電子部品とそれを接続する配線で電子回路のようにプログラムを記述できるので、データや信号の流れが一目見てわかるという利点がある。

(4) SFC (Sequential Functional Chart)

IEC 規格以前から存在する言語であり、基本原理は同じである。工程進捗の制御表現に優れている。図 3.13 に図 3.4 の自己保持回路と等価な SFC プログラム例を示す。

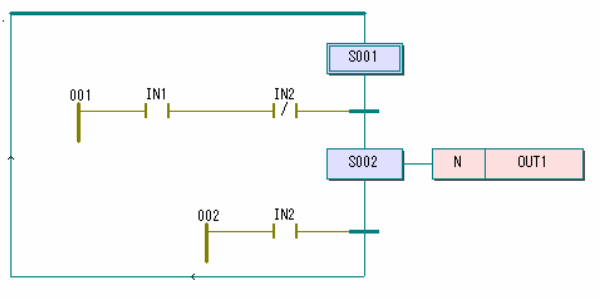


図 3.13 SFC 言語プログラム例

図 3.13 で S001、S002 はステップと呼び、工程の状態を示す。SFC 内では同時に1つだけのステップが実行される。実行状態にあることを活性化と呼ぶ。

SFC では 2 重四角の S001 から始まる。これを初期ステップと呼ぶ。ステップの下に位置する横棒はトランジションと呼び、次のステップへ移行するための遷移条件である。S001 から S002 への遷移条件は、"IN1"が TRUE かつ"IN2"が FALSE となることである。この条件が成立すると、S002 が活性化される。S002 の右横の四角はアクションと呼び、S002 で行う処理を記述する。S002 が活性化状態の時、"OUT1"に TRUEが出力される。"OUT1"へ TRUE 出力は、S002 の下に記述のトランジション、具体的には"IN2"が TRUE ににになるまで継続する。"IN2"が TRUE になると、S002 は非活性化状態となり、最初の S001 が活性化状態となる。

この例でわかるように、SFC では、工程間の遷移条件、工程内の処理を明確に分けて記載できることがメリットである。

(5) IL 言語 (Instruction List)

従来からあるニーモニックに相当する言語であり、マイコンで言えばアセンブラのような言語である。アプリケーションを小型化、高速化するポテンシャルを持つが、プログラミングの生産性、メンテナンス性が劣るため、使用機会は減っていると思われる。図 3.14 にプログラム例を示す。

```
LD    IN1
OR    OUT1
ANDN  IN2
ST    OUT1
```

図 3.14 IL 言語プログラム例

3.4 変数によるプログラミング

変数とは、データを格納する入れ物につける名前のことである。「従来 PLC」では、入れ物の名前はアドレスであり、アドレスに付属する情報としての名前＝コメントを定義することができたが、「IEC-PLC」では、最初に変数の名称があり、属性としてアドレスの定義が可能である。

変数によるプログラミングのメリットは、①変数にわかりやすい名称を付けること、②プログラムロジックの作成時に、変数のアドレスを気にする必要がないこと、③I/O 以外については、変数のアドレスを割り付ける必要がないことにより、プログラムの可読性向上、コーディング効率向上を期待できることである。

図 3.15 のプログラムは自己保持回路の例である。アドレスの代わりに変数名で記述すると図 3.16 のようになる。接点・コイルの目的を変数名で表現しているので、プログラムの処理内容が明確になっている。

第 4 章 IEC- PLC によるプログラムの作成

4.1 IEC－PLC のプログラミングと従来 PLC のプログラミング

本章では第 3 章の従来 PLC と IEC-PLC の違いで得た知識をもとに、実際に IEC-PLC を使用した簡単なプログラムの作成例を紹介する。

4.2 プログラムの作成手順

最初に基本的な PLC プログラムの作成手順を図 4.1 に示す。

客先からの要求仕様を分析して、プログラムを作成、PLC 本体へダウンロードし、デバッグを行い、動作が正しいことを検証する。ここまでの基本的な考え方は、IEC-PLC でも従来型 PLC でも大きな差を見ることはない。しかしその中身を見ると、IEC-PLC では幾つかの特徴的な作業の存在が挙げられる。

以下に、各ブロックでの作業を紹介する。

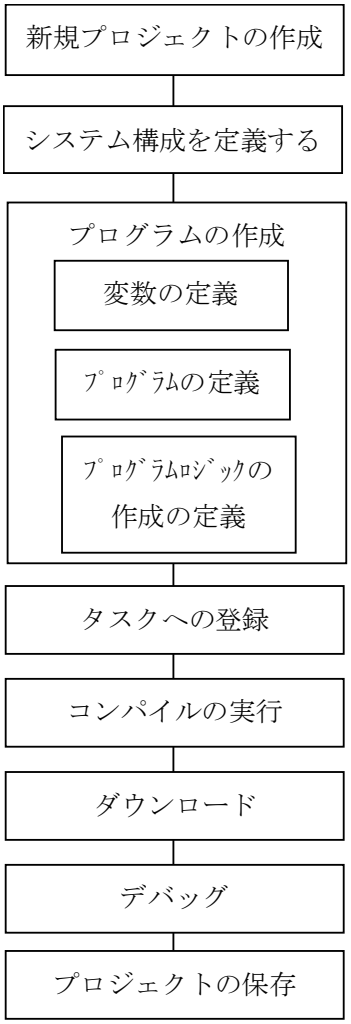


図 4.1 PLC プログラムの作成手順

(1)新規プロジェクトの作成

プロジェクト名(ファイル名)を付けて、作業を開始する。

(2)システム構成を定義する

使用する PLC ハードウェア情報を定義する。即ち CPU モジュールや電源、ベース、プロセス入出力モジュール、通信モジュールなどのシステムのハードウェア構成を登録する。

(3)プログラムの作成

ここからが実際のプログラム作成作業であるが、この作業も「変数の定義」、「プログラムの定義」という IEC-PLC としての定義作業がある。

①変数の定義

使用する変数を定義する(従来 PLC 風にいうと信号名定義と使用メモリ種類・領域の設定に相当する)。

②プログラムの定義

IEC-PLC ではプログラムを分割切り分けして複数つくる(第 5 章、6 章参照)。このため、個々のプログラム毎に名称、種類、使用言語等を定義する。

③プログラムロジックを作成

個々のプログラム毎にプログラムを作る。

(4)タスクへの登録

IEC-PLC は複数のプログラムで構成されるので、作成した個々のプログラムについてタスク登録をし、実行割り当てを行う。

(5)コンパイル

作られたソースプログラムから PLC を動かすための実行コードに変換する。この際、文法チェック等が行われ、誤り・不整合があるとエラー表示されるので(従来 PLC にない機能)、その場合は元のプログラム(ソースプログラム)を修正して再度コンパイルする。従来型 PLC に比べ、PLC プロジェクトの中で最も厄介な作業であるデバッグ作業の効率を大きく高めることが可能となる。

(6)PLC へダウンロード

完成したプログラム(実行コード)を PLC(ハードウェア)にダウンロードする。

(7)デバッグ

PLC にダウンロード後、PLC を運転状態にする。プログラミングツールのモニタ画面で回路の作動状況をモニタし、動作を確認する(図 4.11,赤表示が true＝ON, 青表示が false＝OFF)。

プログラムの変更や修正をする場合は、ソースプログラムの修正→コンパイル→ダウンロードの手順をとる。プログラム内に間違い(バグ)を見つけたときは、ソースプログラムを修正し、コンパイル、ダウンロードの手順を踏むことが必要である(実行コードをプログラミングツールで直接的に直すことはできない)。

(8)プロジェクトの保存

ソースプログラムや各種システム定義情報など、全ての情報を保存する。保存されたプログラムの管理については、第 11 章で詳細に説明する。

4.3 例題-1 : 階段上下スイッチ

ここからは実際の例題で、プログラムの作り方を具体的に紹介する。

最初の例は、2 階建ての一般家屋によくある階段の上と下についている電灯スイッチの例(三路スイッチと等価)である。階段の上からでも下からでも電灯のオンオフができるようになっている。概略を図 4.2 に示す。

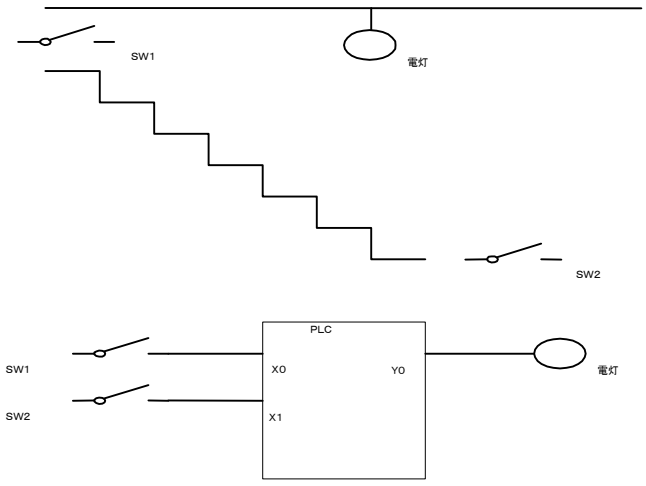


図 4.2 例題-1 階段上下スイッチ概略図とブロック

要求仕様は、階段の上下のスイッチ(SW1、SW2)のいずれかの変化(ON→OFF 又は OFF→ON)で電灯の点灯状態を変化させる(消灯→点灯又は点灯→消灯)ことである。以下具体な手順を追ってプログラミング作業方法を概説する。

4.3.1 新規プロジェクトの作成

まず、プログラミングツールを起動し、新規プロジクトとして「JSIA 編集」フォルダに「例題1」のファイルを作る(図 4.3)。

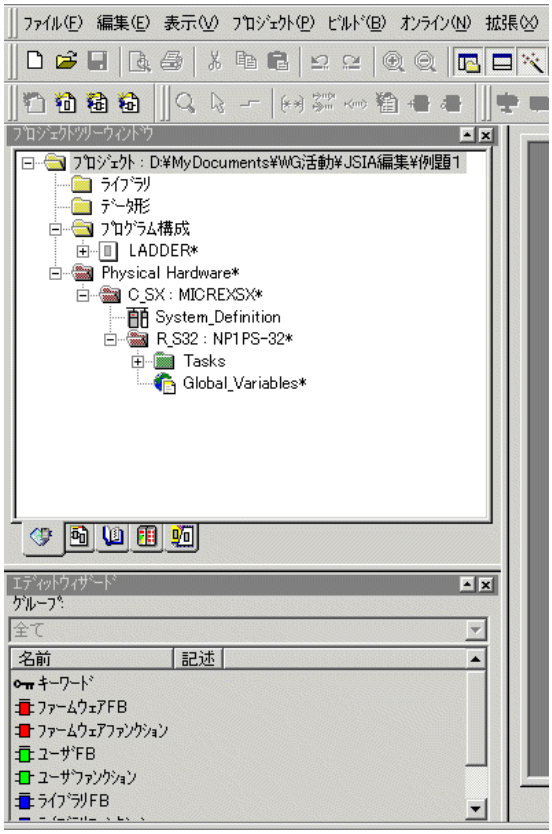


図 4.3 新規プロジェクト「例題 1」の作成

4.3.2 システム構成定義

次に使用するPLCのハードウェア情報の登録を行う。この作業は後で追加も変更も可能である。

図 4.4 はその作業の様子を、図 4.5 はシステム定義の済んだ画面を示す。

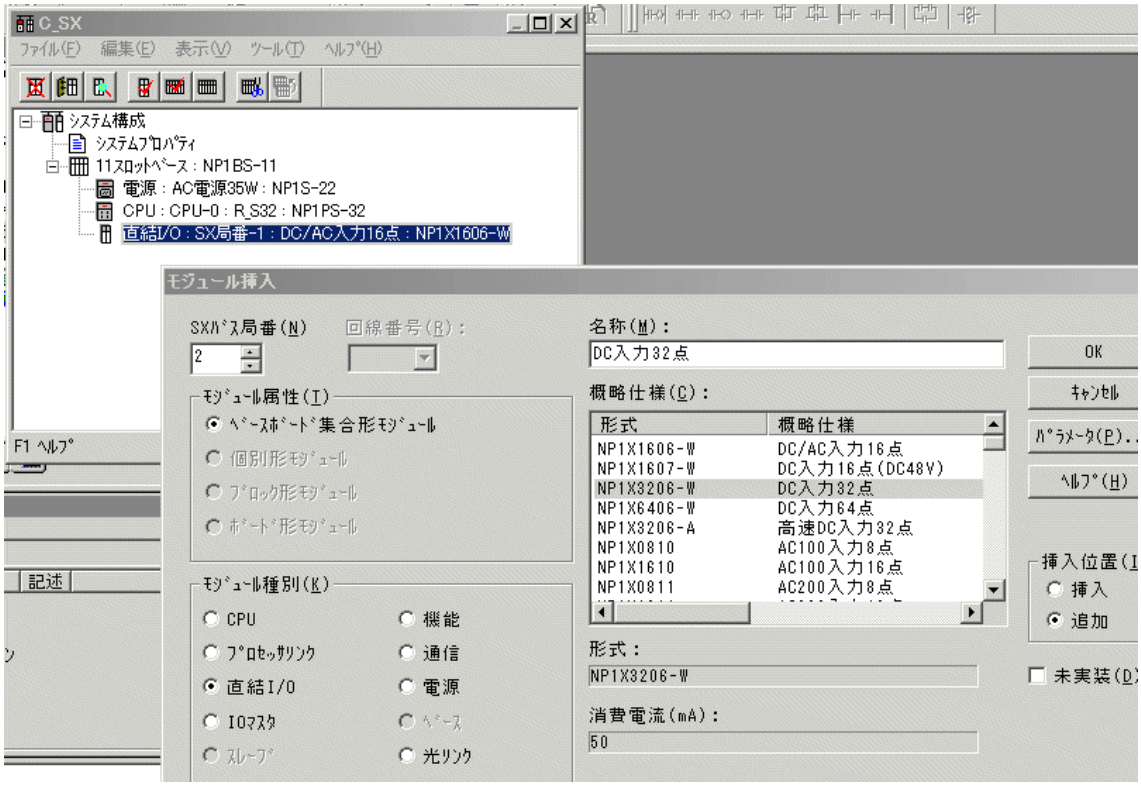


図 4.4 システム構成定義の作業

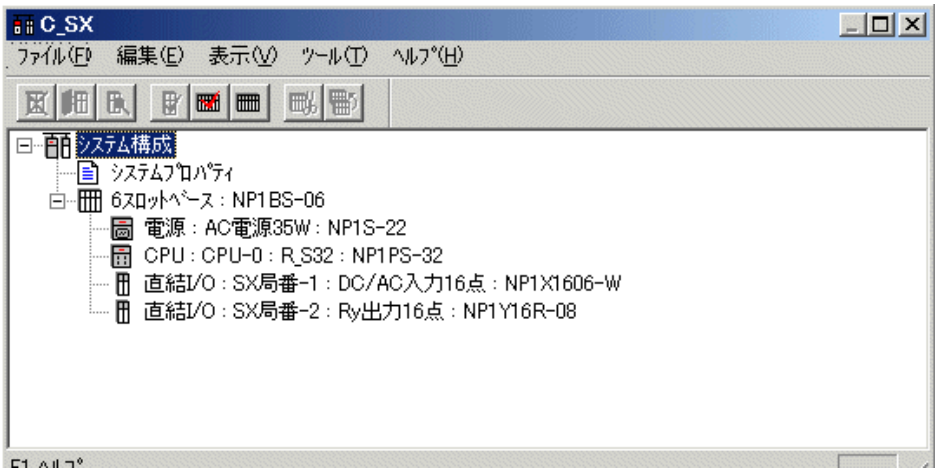


図 4.5　システム構成定義の完了

4.3.3 プログラムの作成---変数の定義

使用する変数を定義する(従来 PLC 風にいうと信号名定義と使用メモリ種類・領域の設定)。

今回は ON,OFF だけの処理で、わかりやすい小さなプログラムである。データ形は BOOL、グローバル変数と設定する(図 4.6)。

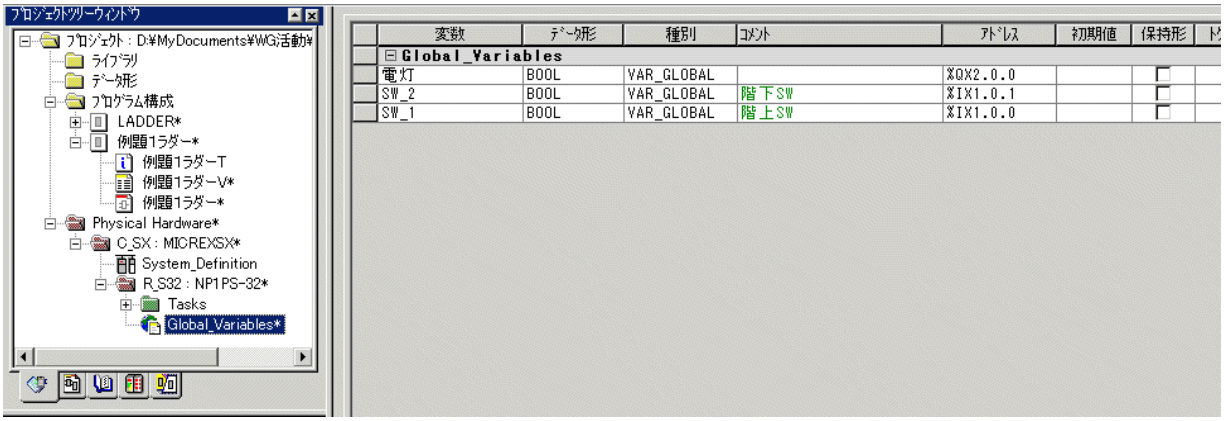


図 4.6　変数の定義

4.3.3 プログラムの作成---プログラムの定義

次に作成するプログラムの名称、種類、使用言語等を定義する(図 4.7)。

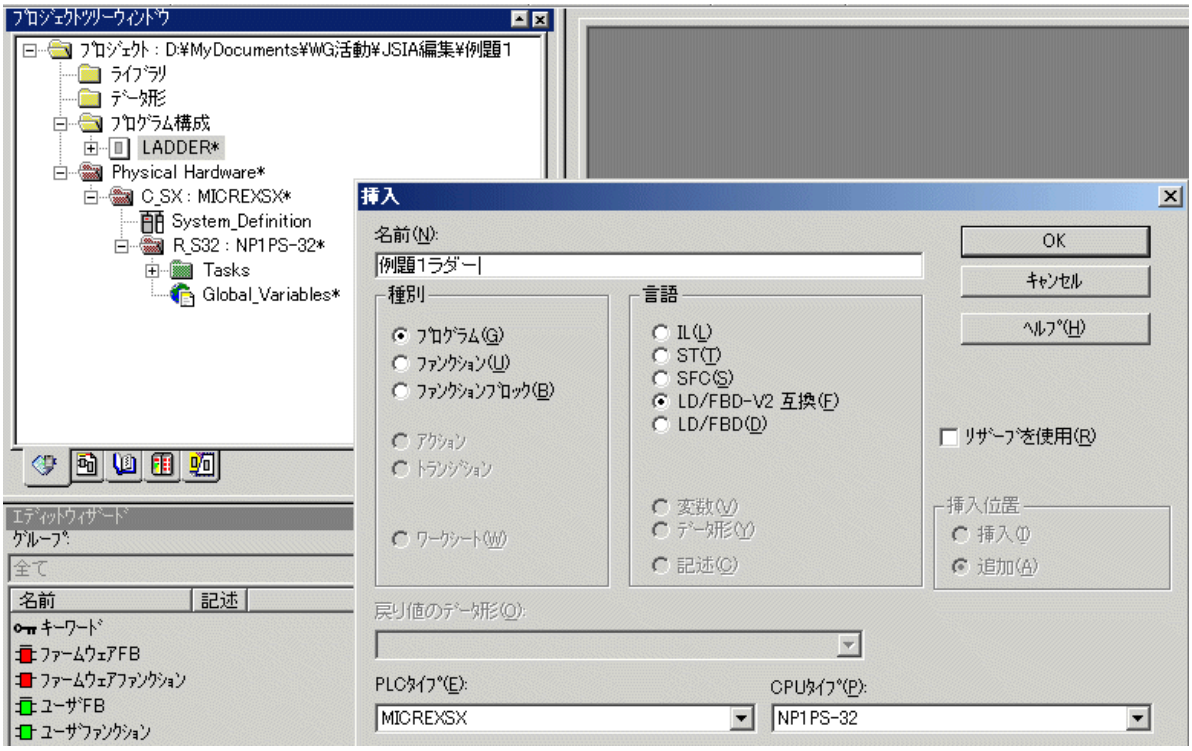


図 4.7　作成するプログラムの定義

4.3.4 プログラムの作成---プログラムロジックの作成

次にプログラムロジック（回路）を作成する（図 4.8）。

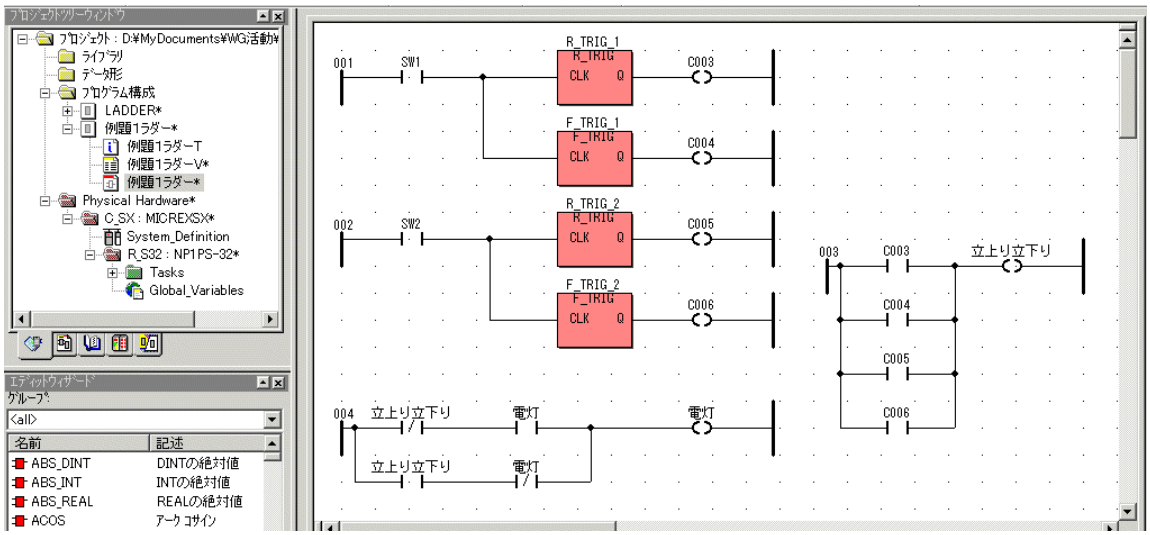


図 4.8　プログラムロジックの作成

4.3.5 タスクの登録

作成したプログラムについてタスク登録をし、実行割り当てを行う（図 4.9）。

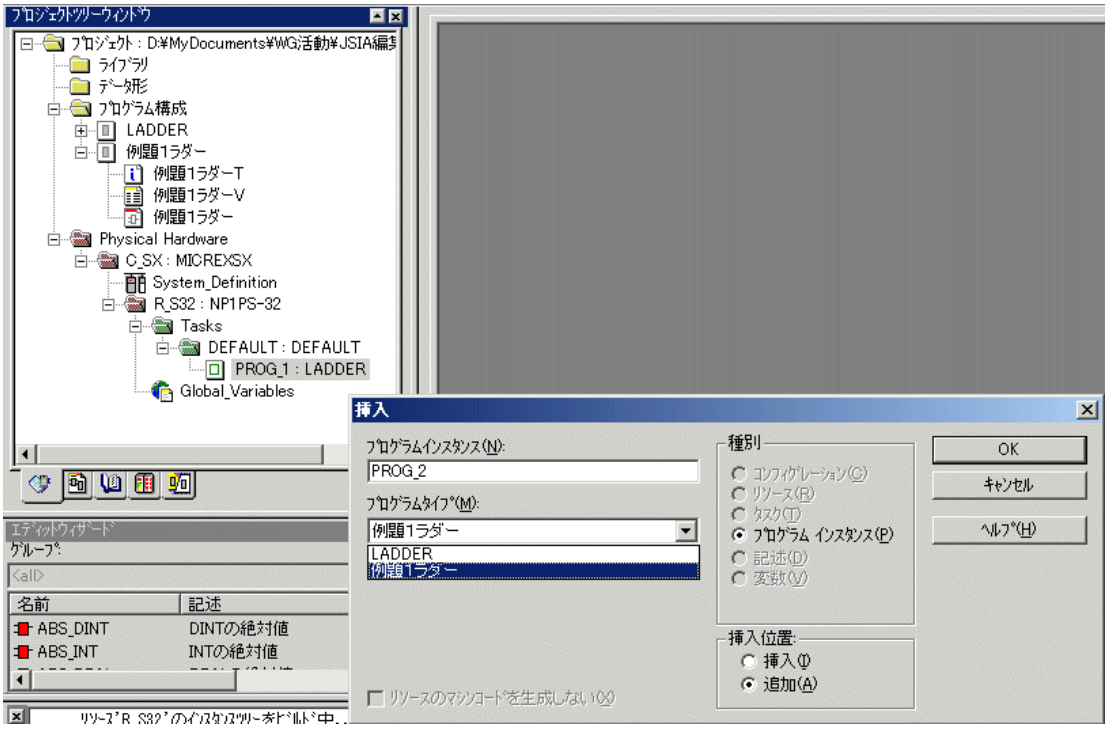


図 4.9　タスクの登録

4.3.6 コンパイルの実行

作られたソースプログラムを、実行コードに変換する。この際、文法チェック等が行われ誤り・不整合があるとエラー表示されるので、その場合はプログラムを修正して再度コンパイルする（図 4.10）。

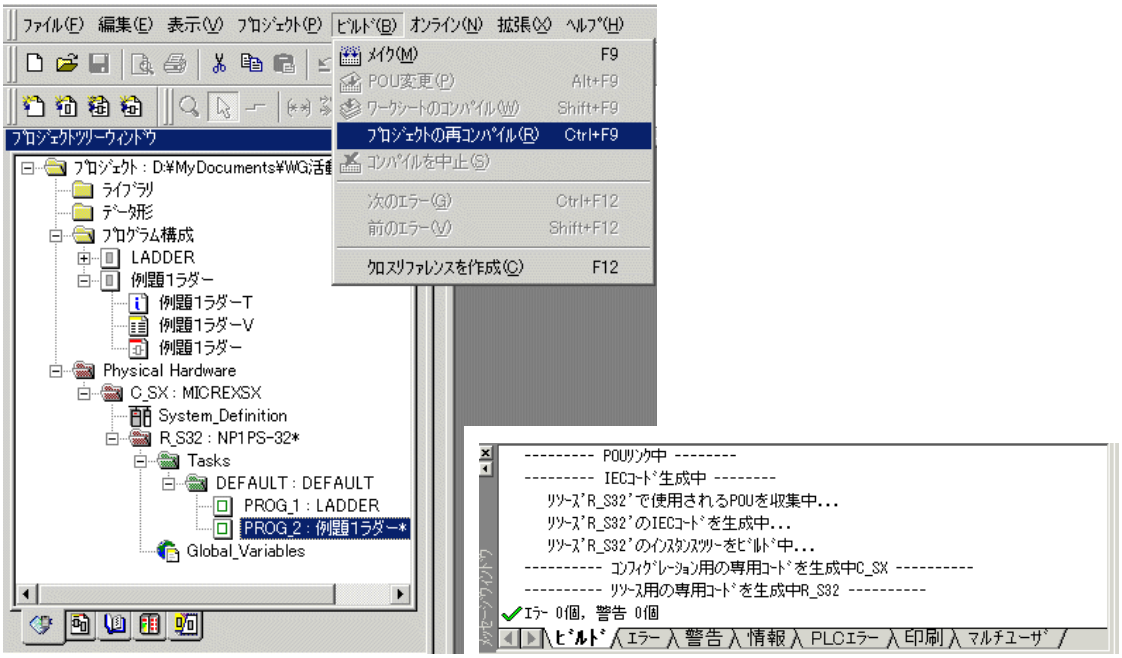


図 4.10　コンパイルの実行

4.3.7 PLCへのダウンロード

完成したプログラム(実行コード)をターゲットの PLC にダウンロードする(図 4.11)。

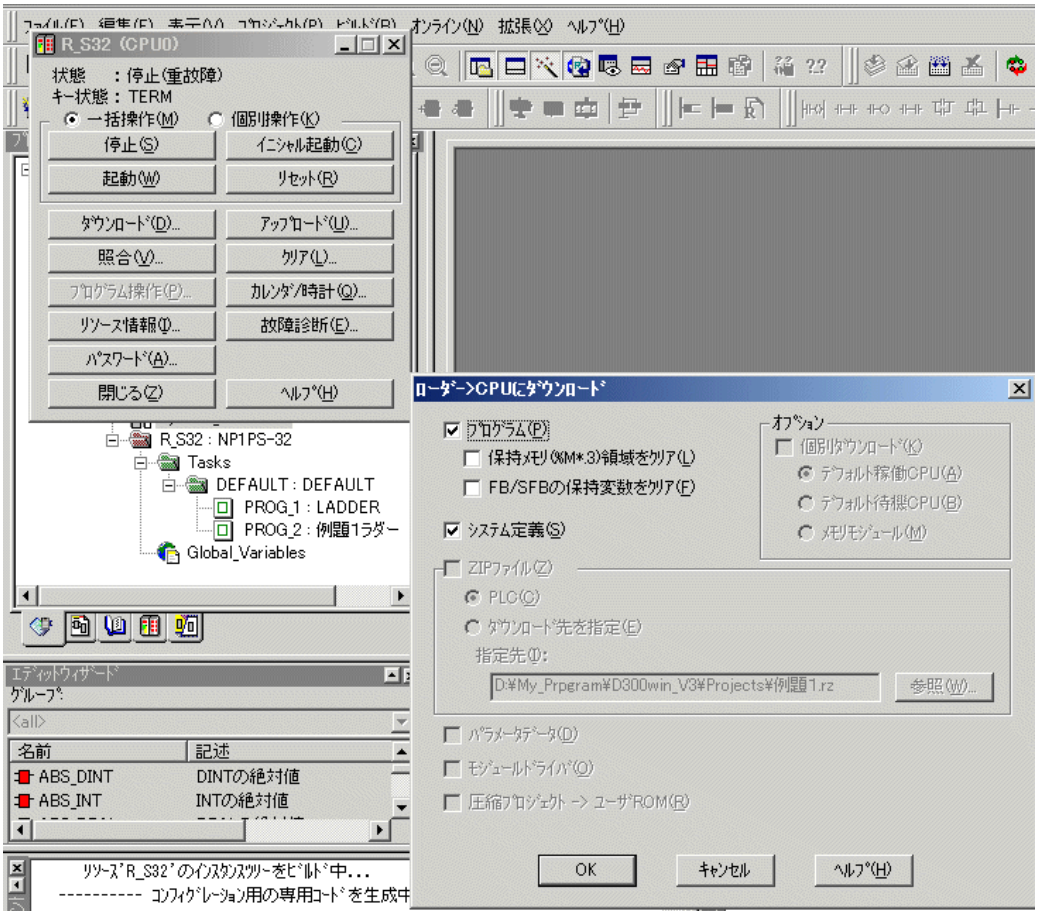


図 4.11 ダウンロードの実行操作画面

4.3.8 デバッグ

PLC にダウンロード後に PLC を運転状態にする。プログラミングツールのモニタ画面で回路の作動状況をモニタし、動作を確認する(図 4.12,赤表示が true＝ON, 青表示が false＝OFF)。

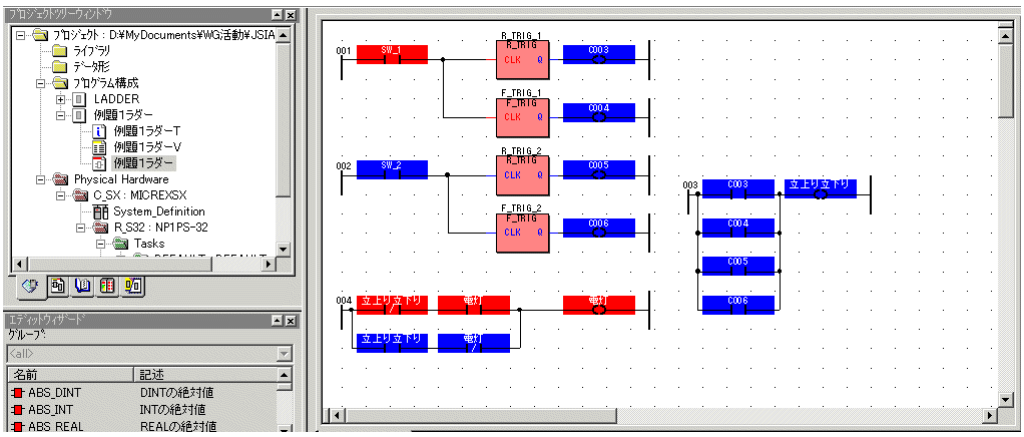


図 4.12 プログラム(回路)モニタ画面

プログラムの変更や修正をする場合は、ソースプログラムの修正→コンパイル→ダウンロードの手順をとる。プログラミングツールにオンライン書き換えの機能（ボタン）があるものは、変更部分（POU単位）のプログラムのみコンパイルとダンロード（PLC内部実行コード変更）の処理を一度に行う仕組みをもつ。実機械装置でのオンライン書き換えは危険を伴う場合が多いので、十分な安全管理体制を敷いた上で行うこと。

4.3.9 完成したプログラム

4.3.4 項では、ラダーダイアグラム（LD）によるプログラミングをしたが、その完成したものが図 4.13（A）である。また、このプログラムは、シーケンシャルフローチャート（SFC）でも記述できる。その例が図 4.13（B）である。

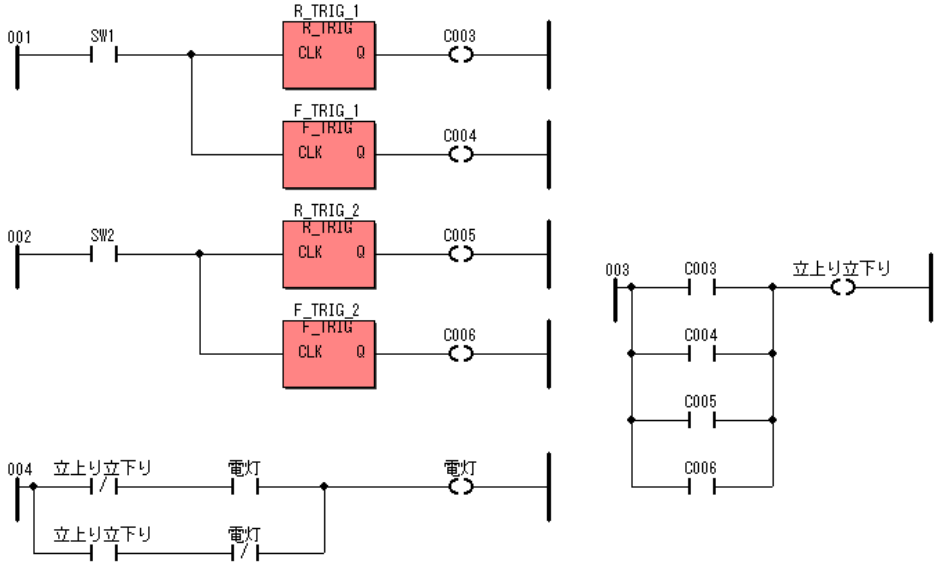


図 4.13（A） 例題-1 のラダーダイアグラムの例

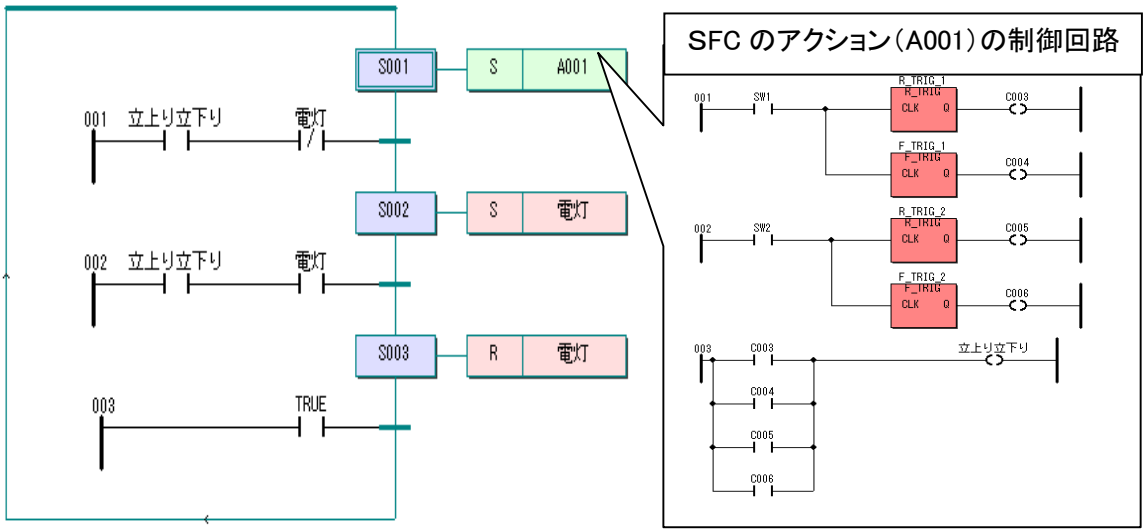


図 4.13(B) シーケンシャルフローチャート(SFC)によるプログラミング例

第 5 章　ソフトの部品化について

5.1　部品化とは何か？

IEC-PLC におけるプログラミング作業は、部品化を前提とした手順になる。よく知られている電子回路に例えて説明する。

初期には単機能の電子部品であるトランジスタや抵抗などを組み合わせ接続して、ある機能をもつ電子回路を構成していたが、現在では、よく使われる機能回路を1チップ化して IC として供給している。IC は汎用性の高い回路・機能をもつ量産部品である。

私達が装置（システム）を製作するときには、複数の IC や電子部品をプリント板上に配置し、さらに多くの機能をもつ基板を作る。この基板も最終目的のシステムからみると部品である。この基板は、IC より汎用性は低いが、反面私達で容易にカスタマイズできる大きな機能部品である。この基板を複数組み合わせて、（個別に仕様が異なる）目的のシステムを製作する。

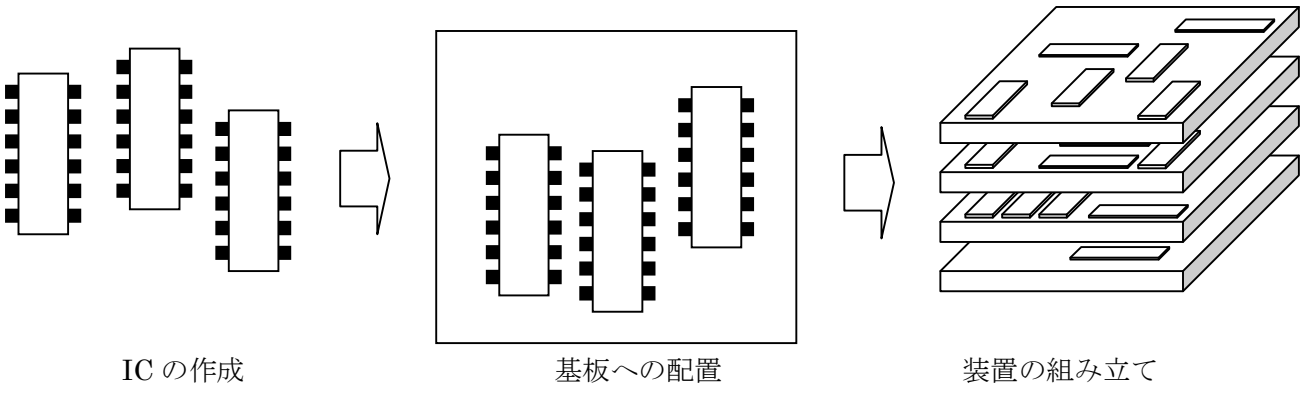


図 5.1　電子部品における部品化のイメージ

IEC-PLC におけるプログラミグを、上記の電子回路に対比すると次のようになる。

①最初に IC に相当する汎用性の高いソフト部品をつくる。

②次のステップでこのソフト部品を組み合わせて接続して、基板に相当する機能ソフト部品を作る。

③さらに、これらのソフト部品類を組み合わせ接続して、最終目的のシステムを製作する。

電子回路と同様に、①のレベルのソフト部品は、汎用性が高く多く使われ、品質も高い。誰でもその機能を理解できる。②のレベルのソフト部品は、①より汎用性は低いが、用途分野に沿った高機能な、専門的知識・ノウハウが凝縮された機能部品といえる。これも社内や業界内の共同利用により、ソフトの可視性・品質・保全性を高めることができる。③は装置ごとの個別要求対応の部分である。このように、プログラミング作業を3つのレベルに分けることができる。

IEC-PLC のプログラミグの作業、即ち小さなロジックをひとまとめにしながら、これを利用しさらに大きなロジックを作っていくエンジニアエリング作業そのものが、ソフトの部品化につながっていくのである。

IEC-PLC の部品は、内部のロジックとそのロジックに対する入出力変数で作成される。ファンクション

ブロック(FB)の場合、部品は使うときにユニークな名前を割り付けて使用する。異なる名前で使用される内部の変数は異なる領域に確保されるので、他の部品に影響しない。部品の入出力には変数が用いられる。具体的には内部のロジックで使っているいくつかの変数を入力用/出力用/入出力用に定義する。これらの変数には様々な型を用いることができる。多量のデータを受け渡す必要のある場合は、配列や構造体を用いると便利である(配列や構造体については第 3 章を参照)。

5.2 部品化による作業の効率化

部品化による利点を以下に述べる。アプリケーションには似たようなロジックがたくさん存在する。それらを体系的に部品化することにより、さまざまな利点が出てくる。

主な利点はエンジニアリング工数の削減である。最初のロジックを作成するときは大きな削減は望めないが、ロジックの資産がたまっていくことでエンジニアリング工数削減の効果は非常に高くなる。

従来 PLC では、メモリアドレスに対応したアドレスを使うために、似たようなロジックを作成するときでも、アドレスの変更を余儀なくされており、単純ミスや検査漏れにもつながっていた。IEC-PLC ではほとんどのロジックを固定的なアドレスを持たないローカル変数で作成することができるために、これらをコピーして使用した場合でも、固定されたアドレスというものを意識する必要がない。別の変数として扱われるからである。これにより、コピー後の修正作業ミスの削減や、チェックが必要な部分の削減によりエンジニアリング工数を大幅に削減できる。

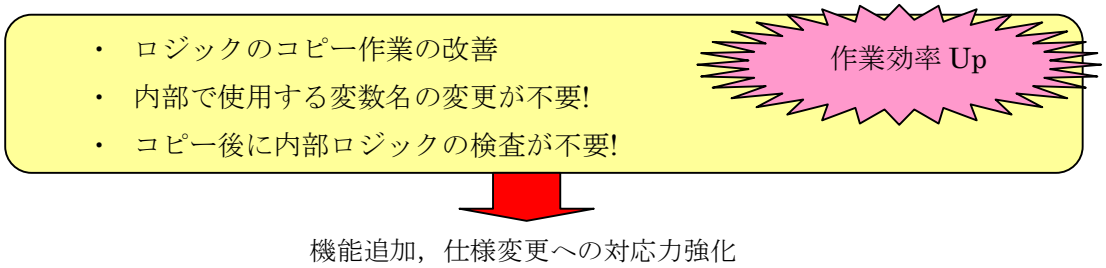


図 5.2 部品化によるメリット

次の大きな利点はロジックの可読性である。 従来 PLC も様々な技術革新により、シーケンスだけでなく、アナログ制御や難しい演算もできるようになってきている。しかし、プログラム言語は、一部の従来 PLC では言語系なども扱えるようになってきているが、いまだにラダーが中心であり、各々のエンジニアがアイディアを出しながらラダーを作成している。

IEC-PLC では、シーケンス、演算、制御などは、各々の部品にわけて、その部品単位で適した言語を扱うことになる。そうすることにより、機能毎にロジックを配置しやすくなり、ロジックを閲覧したときに、イメージで処理の全体がわかりやすい。各々の部品の詳細は、各々適した表現で記載されているために、ロジックの作成者以外でも理解するのにさほど時間を必要としない。結果的に可読性が良いことは、メンテナンスしやすいことにつながり、将来の設備改造に対する安全性やメンテナンス性の良さに結びついていく。

IEC-PLC の欠点は、規格で制定されている部品がそんなに多くないことである。機種に依存するようなロジックや実現が困難なロジックは、各 IEC-PLC メーカの独自部品が提供されている。また、それ以

外に必要な部品は、エンジニアが自由に作成することになる。いくつかの IEC-PLC メーカでは製品で提供している部品だけではなく、ユーザが変更できる環境で部品を作成し、メーカのホームページなどで部品をダウンロードできる仕組みを持っているところもある。それらを活用し、単なる工数削減だけではなく、ロジックの組み方の参考にすることが望ましい。

5.3 部品化の事例

「例題-2:ベルトコンベア搬送システム」の事例を以下に示す。この装置には検査装置があり、さらにその検査装置の中に運転故障判断ロジックが存在する。それだけでなくこの検査装置は、取付締込機にも用いている。このように同一機能のロジックを部品のように、簡単に別のロジックに流用することができる。

IEC-PLC ではこれらの部品のことを FUN(ファンクション)や FB(ファンクションブロック)と呼び、それらを活用してプログラムを作成する。

第 6 章 構造化設計と IEC-PLC によるプログラミング

6.1 構造化プログラミングとは

構造化プログラミングとは、プログラミング上の手続きをいくつかの単位に分け、メインプログラムでは大まかな処理を記述し、サブプログラムで細部を記述していく方法（複数階層）で、その概念を図 6.1 に示す。

全体的なシステム要求機能から詳細機能へと順に開発するトップダウン式と、小さな機能をまとめて大きな機能単位をつくるボトムアップ式の開発方法がある。

サブプログラムは、「同じ値を渡せば常に同じ結果が得られる」ことを満たすような部分処理を指し、データと機能は直交するもので可能な限りお互いの文脈に依存しないように分割する。

こうすることでプログラムのモジュール性（設計上の概念で、システムを構成する要素となるもの。いくつかの部品的機能を集め、まとまりのある機能を持った部品のこと。）を高めることによって設計や保守が容易となるというのが構造化プログラミングの特徴であり、サブプログラム・モジュールに相当する部分が、第5章に示したソフト部品である。

最近は、多種多様なデータを扱うためデータを中心に据えたデータフロー・オブジェクト指向が注目されているが、それも処理分割の構造化が基本になっている。

【構造化プログラミングの長所】

・プログラムがコンパクトになり可視性が増すので、プログラムの理解や検証が容易になる。

・ブロックごとに正当性や機能の確認ができ、保守や修正が容易になる。

・プログラムを部品化し、別のシステムへの再利用が容易になる。

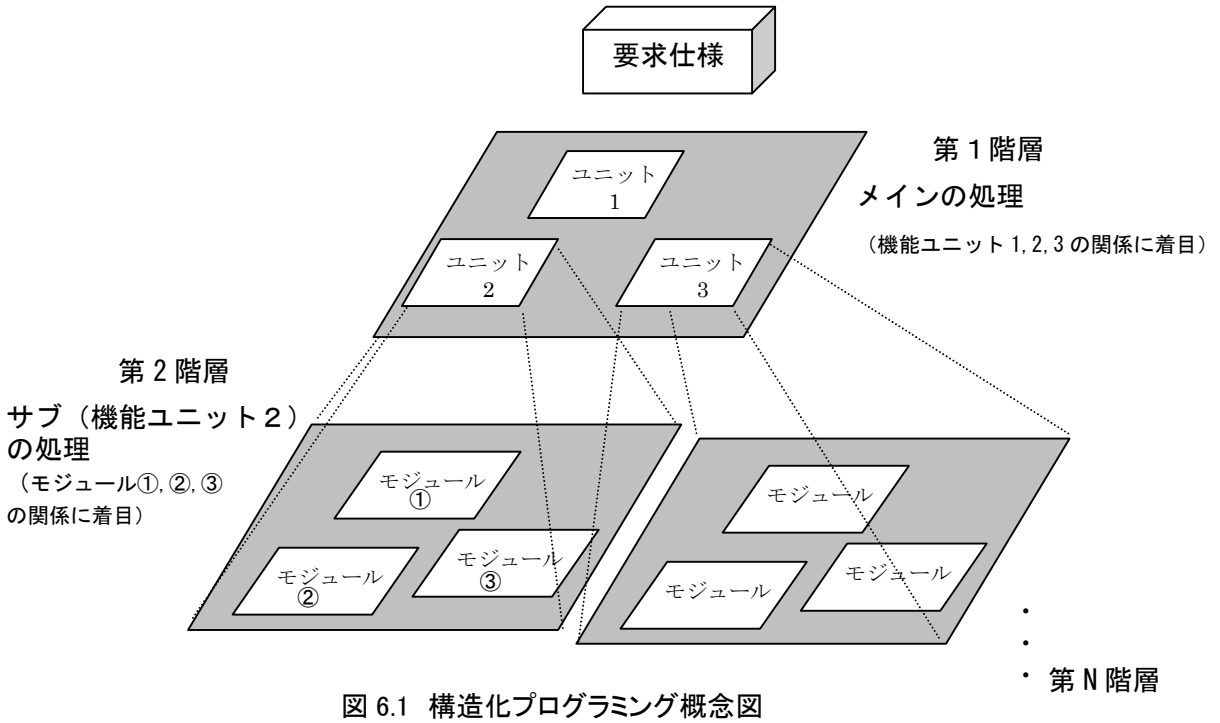


図 6.1 構造化プログラミング概念図

6.2 構造化プログラミングの手順

(1)システムの要求仕様の分析と処理分割

まず 6.1 項で示したとおり、システムの要求仕様を分析・検討し、処理内容をメイン(1 層)、サブ 1(2 層)、サブ 2(3 層)・・・というように処理分割設計を行う。その際、各処理の要求仕様を下位層の処理に求める機能を含めて、明確にしておくこと。

手動運転、部分自動運転、自動運転などの運転モードはもとより、正常な処理だけでなく異常・エラー処理等も必ず明確に定義・設計する必要がある。

(2)メインプログラムの設計とプログラミング

システムの基本となるメインプログラムを作成する。同時にメイン処理として下位層の処理に求める仕様を明確にする。

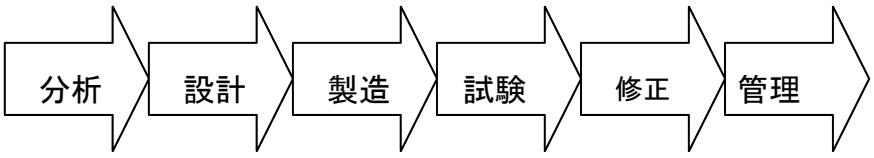
(3)システム固有のサブプログラムの設計とプログラミング

上位の処理(メインプログラム)が要求する、ターゲットのシステム固有のサブ処理プログラム群を作成する。その際、できるだけソフト部品＝共通モジュールプログラムの使用を意識した設計をする(構造化の基本)。

(4)ソフト部品(共通モジュールプログラム)の選択、新規作成

上位プログラムで要求される処理仕様を実現するために必要なソフト部品を、既存資産のソフト部品ライブラリから抽出する。該当するものがない場合は新たに作成し、将来再利用できるソフト部品は部品ライブラーの1つに加えておく。

なお、メイン・サブとも各プログラムは、通常のプログラミング開発と同様に次に示す手順で作る。



①分析・・・上位要求仕様を分析し、下位階層処理、分割・分担を意識して行う。
②設計・・・分析結果に基づいた機能単位で設計を行う。
③製造・・・プログラミングを作成する。
④試験・・・デバッグを実施する。
⑤修正・調整・・・要求仕様に沿うようプログラムを修正する。
⑥管理・・・必要なドキュメントも含め、ソースプログラムとして保存管理する。

6.3 構造化プログラミングのポイント

構造プログラミングは、6.1 項で述べたとおり

- **全体的な機能から詳細機能へと順に開発するトップダウン式** と
- **小さな機能をまとめて大きな機能単位をつくるボトムアップ式** の開発方法があり、

実際の設計作業は、

①要求システム仕様に基づくメイン処理プログラムと必要な各工程仕様など下位処理サブプログラム