平成 16 年度 修士学位論文

PLC プログラムのハードウェア変換ツールに関する研究

A Study on a Hardware Translation Tool for PLC Programs

豊橋技術科学大学大学院　工学研究科
知識情報工学専攻　　市川研究室

学籍番号：013703　　氏名：池田 亮
指導教官：市川 周一

目次

第 1 章

序論

1.1 背景

産業用機械などのシーケンス制御には，PLC（Programmable Logic Controller）[*1] と呼ばれる制御用計算機が広く利用されている．PLC は，任意の制御論理を記述した PLC プログラムを処理することでシーケンス制御を実現している．PLC を利用するメリットの一つに，制御論理をシステムに合わせて変更可能な柔軟性を挙げることができる．

しかし，生産設備の大規模化，複雑化，加工の高精度化，生産の高効率化などに伴って，PLC の処理速度不足や制御点数不足などが指摘されている．また，PLC プログラムはソフトウェアであるため解析やコピーが容易であり，貴重なノウハウの流出やコピー商品の出現などの問題を招いているということも指摘されている [1].

そこで近年，FPGA（Field Programmable Gate Array）を利用して PLC プログラムを論理回路化する手法が検討されはじめた [2] [3]．FPGA は，論理構成を何度でも変更可能な，柔軟性のある LSI である．PLC で実現していた制御論理を FPGA による論理回路で実現できれば，PLC の柔軟性を保ったまま処理速度を改善することが可能になる．

また，FPGA は 1 チップで数十から数百万ゲートを集積できるため，メインとなる制御論理と，それ以外に必要となる周辺回路まで含めて 1 チップ上に実装することが可能である．したがって，処理速度と FPGA デバイス自体が安価になっていることの他に，さらに次のメリットを期待できる．

- 部品点数の削減，スペースの縮小，消費電力の軽減による低コスト化
- 1 チップ上への集積化による制御論理のブラックボックス化，セキュリティの強化

[*1] 別名，プログラマブル・ロジック・コントローラ，プログラマブルコントローラ，シーケンサ

以上述べたように，PLC のソフトウェア面の課題であるセキュリティを向上させ，かつハードウェア面の課題である処理速度を改善する，という問題の緩和は，FPGA を適用することにより可能だと考える．

1.2 関連研究

PLC で実現していた制御論理を FPGA による論理回路で実現するというアイデアは，古くから提案されている．本節では，このアイデアと関連する先行研究について言及し，その課題を指摘する．

1995 年，Adamski らは制御用ロジックコントローラを PLD（Programmable Logic Device）[*2] で実現するシステムを報告した [4]．この研究で Adamski らは，制御論理をペトリネットに基づくルールベース言語で記述し，それを PLD 用の論理記述言語 PALASM に変換して PLD に実装する手法を検討した．

さらに 1996 年，Wegrzyn らは上述のルールベース言語をハードウェア記述言語 VHDL に変換するシステムを報告した [5]．しかし，これら二つの研究はどちらもアイデアの提案や手法の検討に留まったものである．

1998 年，Wegrzyn らはルールベース言語で記述した制御論理を論理回路に変換し，FPGA 上に実装するシステムを報告した [6]．この研究では，制御論理モデルからルールベース言語の記述を獲得する手法が検討された．また，幾つかの簡単な論理回路について，Xilinx 社の FPGA 上での実装結果が示された．

1999 年，宮澤らはラダー図表現の PLC プログラムを FPGA で実装する手法について報告した [7]．この研究では，ラダー図表現から VHDL 表現への変換手法について具体的に検討された．しかし，検討された例は非常に小さく単純で，定量的評価結果もない．

また 1999 年，池下らは SFC 表現の PLC プログラムをハードウェア記述言語 VerilogHDL に変換するツールを報告した [8]．しかし，この研究においても具体的応用に関する言及や定量的評価結果はない．

2003 年，石野らは PLC プログラムをハードウェアとソフトウェアからなる一つのシステムとして実装し，その評価結果を報告した [9]．この研究では，ハードとソフトの協調のために C ベース設計を採用し，ハードの実装には Handel–C 言語が用いられた．しかし，実装対象となったシステムや評価条件に関する具体的説明がないため，追試できないという問題がある．

[*2] PLD とは，半導体メーカが回路の要素をあらかじめ用意しておき，ユーザがそれを組み合わせて任意の回路を作る，プログラム可能なデバイスの総称である．本文では低集積度の FPGA 程度の意味で使っている．

以上説明してきたように，先行研究では，PLC を FPGA で実現するという基本的アイデアの提案やそのための手法を扱っている．しかし，それぞれ基本的な提案に留まっており，実応用の PLC プログラムを FPGA で実現する際の問題点への言及はない．また，現実の PLC を FPGA に置き換えた場合の比較評価結果もないか，あっても簡単なものであったり，評価条件等が明記されていないという課題がある．

1.3 本研究の目的

本研究では，現実の PLC プログラムについて考察し，それをもとに変換手法の検討と実装を行う．また，変換手法のみでは対応できない制限についても言及し，その対策としてのフレームワークを示す．

また，同じ制御論理を PLC 上と FPGA 上で実行した場合の各種データを測定し，それらをもとに比較評価と検討を行う．

1.4 論文の構成

本論文の構成は次のとおりである．まず 2 章では，本研究で扱う PLC プログラムについて説明し，それからハードウェアへの変換手法の検討およびフレームワークの必要性について述べる．次の 3 章では，フレームワークの概要を説明し，その構成要素として実装した各種ツールについて述べる．4 章では，フレームワークとその構成が正しいことのテストと，サンプルの PLC プログラムを利用した PLC と FPGA との比較評価を行う．最後に 5 章で本研究をまとめ，さらなる検討と今後の課題について言及する．

第 2 章

PLC プログラムから H/W への変換

2.1 PLC プログラム

PLC プログラムとは，PLC が処理する任意の制御論理を記述したプログラムである．1 章でも述べたように，記述された制御論理は制御システムに合わせて変更可能なため，生産現場での制御内容の変更に柔軟に対応することができる．

PLC はいくつかの PLC メーカ，例えば三菱電機，オムロン，日立，キーエンスなどから製品として提供されている．それぞれの PLC で用いられる命令語や用語は，少しずつ異なっているものの，機能や意味としては等価なものも多い．したがって，一つの機種について理解できれば，その他の機種の理解は容易と言える．

本節では，現実の PLC プログラム（サンプル）として，三菱電機製の PLC FX2N [10] 用の PLC プログラムについて説明する．これは，入手できたサンプルが FX2N 用だったためである．以降では，FX2N 用の PLC プログラムについて，どのように制御論理を記述するか，どのように処理が実行されるか，などを説明する．

2.1.1 PLC プログラムの記述

PLC プログラムの記述には，ラダー図と呼ばれる表現方法が広く利用されている．当然，FX2N 用の PLC プログラムを記述する際も利用可能である．下図 2.1 はラダー図の例である．

図 2.1: 簡単なラダー図の例

図 2.1 に示したラダー図は，入力スイッチ X000 と X002 がともに ON の場合のみコイル出力 Y000 が ON するような機能を持つ．図から明らかなように，ラダー図とは，PLC における入出力をリレー回路の考え方を使って制御するための記法と言える．

しかし現在の PLC はリレー回路をそのまま置き換えたようなものではなく，一種の計算機になっている．そのため，データ処理命令や分岐命令あるいはサブルーチンコールなどを直接記述することも可能である．図 2.2 は，そのような一例を示したものである．

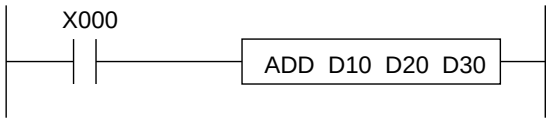

図 2.2: 命令が埋め込まれたラダー図の例

図 2.2 は，入力スイッチ X000 が ON の場合のみデータレジスタ D10 と D20 の値を加算して結果を D30 に格納する，という機能を持つラダー図の例である．このとき，X000 が OFF であれば加算命令は実行されず，D30 の値は変わらない．

2.1.2 段，条件部，処理部の定義

前節では，PLC プログラムとその記述について概要を説明した．本節では，これからの詳細な説明のために，三つの用語を定義する．

ラダー図表現における処理の最小単位を段と呼ぶ．PLC プログラムは複数の段から構成され，その段は条件部と処理部から構成される（図 2.3 参照）．条件部には処理部を制御するための条件が，処理部にはコイルへの出力命令やデータ処理命令などが設定可能である．

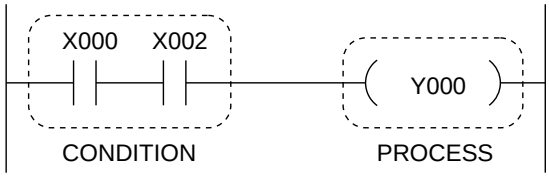

図 2.3: 段，条件部，処理部

2.1.3 様々な表現方法

PLC プログラムを記述するための表現方法として，ここまではラダー図のみを扱ってきた．しかし，この表現方法にはラダー図以外にも様々なものが許されており，PLC プログラミング言語として定義されている [11]．それらの説明を次に示す．

- シーケンシャルファンクションチャート：SFC
  - SFC はフランスの Grafcet に由来する．Grafcet は，離散事象システムを表す数学モデルのペトリネットから派生した表現法であり，ペトリネットは，非同期性と並列性を有するシステムをモデル化するために考案された表現法である [12] [13]．フランスでは，ペトリネットに幾つかの制約条件を与えたものを Grafcet として規格化し，IEC では，これを基本に制御アクションの記述を強化し SFC として規格化した．
- 命令リスト：IL
  - 命令リストはニーモニックを規格化したものであり，コンピュータのアセンブラ言語に相当するものである．PLC プログラミング作成ツールの進歩によって，ユーザ自身が IL を使用することは少くなってきた．しかし，IL は，テキスト形式の言語であるため，異なるメーカ間の PLC プログラムの移植や互換などにおいて重要な役割を持っている．命令リストは一連の命令群の羅列で，命令は修飾子をもった演算子とオペランドから構成される．
- 構造化テキスト：ST
  - ST は PASCAL 言語を基本にした言語である．ST の構文と意味論は PASCAL 言語のものを多く援用しており，IEC61131–3 の中における唯一の高級言語である．また，1970 年代に始まる構造化プログラミング手法の影響を強く受けている．一般高級言語にある命令文を使用するので，記述力がかなり高い．
- ラダー図：LD
  - LD は PLC の中で最も広く使われている言語の一つであり，そのリレーシンボルからきた接点やコイルの記号を“ はしご ”状に書くことからラダーと呼ばれている．LD はパワーフローとも言われ，電気信号の流れをそのままプログラミングすることに相当する．プログラムの特長として，システムの停止やインターロックなどが容易に記述可能な点を挙げることができる．また，安全のための異常検出回路等も容易に構築できるため，シーケンス制御などでは広く使われている．
- 機能ブロック図：FBD
  - FBD は，言語の各種機能をブロック化し，そのブロックによってプログラミングする言語である．FBD はブロック間の接続を明確にしており，さらにその接続関係が処理の前後関係を表している．FBD はデータフローと呼ばれ，データ処理の流れをプログラミングすることを意味し，順序関係が明白である．

三菱電機の PLC プログラミング環境 GX Works では，SFC と ST，LD による記述が可能である．また，それらの記述から IL への変換も，GX Converter によりサポートされる．

2.1.4 PLC プログラムの実行順序

先に述べた各種の PLC プログラミング言語のうち，SFC や LD は，並列性を内包している．つまり，それぞれの処理単位は並列に処理され得る．しかし，実際に PLC が PLC プログラムを処理する時には，基本的に順次処理となる．例えば LD の場合，その内容は一番上から下へ向かって一段ずつ逐次的に処理される（図 2.4 参照）．そして，一番下の段が処理された後は，また一番上に戻って処理を再開する．この繰り返しの周期をスキャンと呼ぶ．

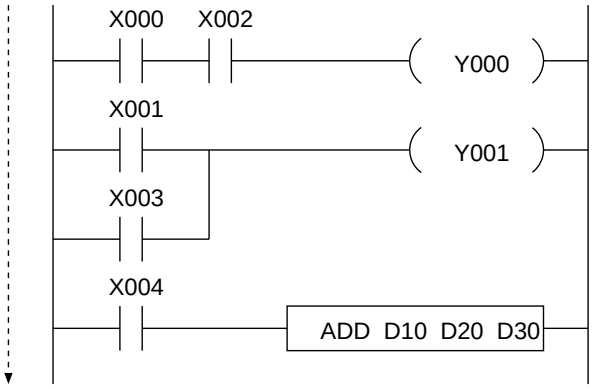

図 2.4: 上から下へ向かって逐次的に処理されるラダー図

図 2.4 に示したラダー図の例であれば，次に示す一連の処理を順次実行する．

- 一段目：X000 と X002 の論理積をとって，その ON/OFF を Y000 に出力
- 二段目：X001 と X003 の論理和をとって，その ON/OFF を Y001 に出力
- 三段目：X004 の ON/OFF にしたがって，加算命令を実行する/実行しない

PLC プログラムが逐次的に処理されるのは，PLC が一種の計算機であることによる．また，PLC の入出力インタフェースに接続された各種の周辺回路の多くが，シーケンス制御を想定していることも理由の一つに挙げられる．

2.2 H/W への変換手法

1 章では PLC プログラムを FPGA 上に実装することのメリットを，前節では PLC プログラムの記述方法と実行順序を説明した．本節では，それを踏まえて，PLC プログラムからハードウェアへの変換手法を述べる．

2.2.1 実行順序と状態遷移の対応

まず変換にあたっての基本的方針として，前節で言及した PLC プログラムの実行順序を厳密に保つような変換手法を採用する．具体的には，ラダーの一段に対して一つの状態を与える状態遷移を導入する（図 2.5 参照）．そして，それぞれの状態では，段単位の処理を行う．

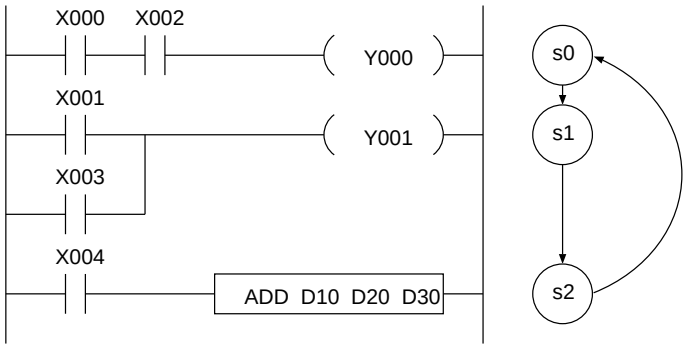

図 2.5: 実行順序と状態遷移の対応

初期状態 s0 から出発してクロック毎に状態を一つずつ進め，最後の段に対応する状態 sN に来たら次のクロックで初期状態に戻るような状態遷移である．段単位で行われる処理はそれぞれ個別にハードウェア化する．このハードウェアは容易に実現可能である．

2.2.2 変換手法の特徴と課題

この変換手法では，PLC プログラムの逐次処理をそのまま FPGA 上に実現するため，実行順序が厳密に保たれるという利点がある．しかしその一方で，ハードウェアの持つ並列性が活かされないため，性能面の向上は最小限に留まってしまう．

このとき，PLC プログラムに矛盾が生じないように，制御論理に含まれる並列性を抽出できれば，さらに性能を改善することが可能である．例えば上図 2.5 の場合，プログラムの各段には実行の依存関係が存在しない．ここに注目し，三つある状態を一つにまとめ，各段の論理をすべて並列に動作させれば，スキャン時間は短縮されて全体の処理時間も縮まると期待できる．また，これによってハードウェア面積の削減も期待できる．具体的には，逐次処理を実現するための状態遷移制御やレジスタに必要な回路の削減である．

このような性能向上のためには，並列化コンパイラ技術の導入が有効と考えられる．また，合わせて周辺回路の特性に関する検討も必要である．これら二つに関しては時間の都合上，実装まで至らなかったが，今後の重要な課題であると考えられる．

2.2.3 条件部と処理部の認識

PLC プログラムの実行順序に関しては，状態遷移を導入して対応することはすでに述べた．ここでは，ラダーの各段をどのようにハードウェアと対応させるかについて説明する．

ラダーの各段をハードウェアに対応させる際には，その段を条件部と処理部に分けて認識する．そして，条件部は AND，OR，NOT からなる論理回路に，処理部は命令に合わせた論理回路に変換する．図 2.6 に基本的なラダー図での変換例を示す．

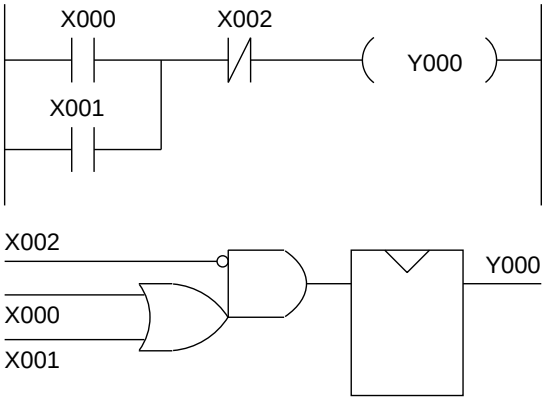

図 2.6: 実行順序と状態遷移の対応（基本的なもの）

コイル出力命令までを対象とするのであれば，AND，OR，NOT とフリップフロップで実現可能である．図 2.7 は，処理部に PLC の命令が埋め込まれたラダー図の変換例である．

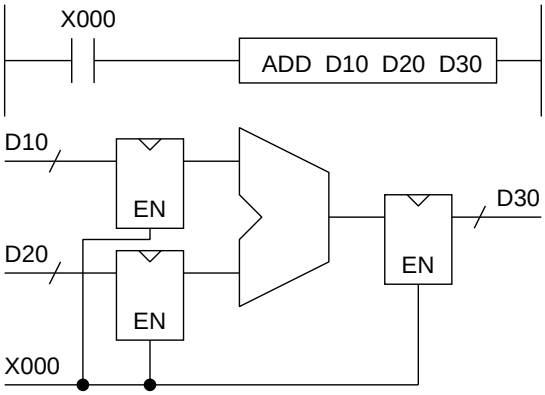

図 2.7: 実行順序と状態遷移の対応（命令が埋め込まれたもの）

図 2.7 に示したような算術演算命令の場合，命令のオペランドを演算器の入出力レジスタにつなぎ，条件部の回路をイネーブル制御につなぐような変換を行う．このように条件部と処理部を分けることで，PLC の命令が埋め込まれるようなラダーも容易に変換可能となる．

2.2.4 検討対象とした命令

本研究では，表 2.1 に示した 20 の命令語について変換手法を検討した．表 2.1 にある命令語は，サンプルの FX2N 用 PLC プログラムで利用されている最小限のものである．表に記載した機能欄より詳しい説明が必要な場合は，メーカの提供するマニュアル [14] を参照いただきたい．

表 2.1: 検討した命令語の一覧

命令語	機能
LD	論理演算開始
LDI	論理否定演算開始
AND	論理積
ANI	論理積否定
OR	論理和
ORI	論理和否定
ANB	論理ブロック間の直列接続
ORB	論理ブロック間の並列接続
NOP	無処理
END	プログラム終了
OUT	出力（コイル駆動命令）
SET	動作保持コイル
RST	動作保持コイルのリセット
PLS	立上りエッジ検出（微分）
PLF	立下りエッジ検出（微分）
MOV（DMOV）	データ転送
ADD（DADD）	加算
SUB（DSUB）	減算
MUL（DMUL）	乗算
DIV（DDIV）	除算

本研究では，FX2N 用の PLC プログラムを扱うために，三菱電機の PLC プログラミング環境 GX Works および GX Converter を使用する [15]．これらを使用することで，ラダー図表現から命令列表現への変換がサポートされる．

ラダー図表現と比べて命令列表現はテキスト形式であるため，プログラム的に扱いやすい．この点を考慮して，変換手法の実装に際しては入力に命令列表現を利用する．ラダー図表現も命令列表現も，PLC プログラムの表現方法の一つであり，内容的には等価なので問題はないと言える．

2.2.5 基本命令から H/W への変換

まず，基本命令である LD，LDI，AND，ANI，OR，ORI，ANB，ORB の論理演算系の命令に関しては，先に述べたとおり AND，OR，NOT からなる論理回路に変換する．

NOP 命令と END 命令は，基本的に論理回路には変換しない．そのため，NOP 命令を並べて時間の調整を行っているような記述は根本的に書き直す必要がある．これは，例えば OUT 命令を使ってタイマ設定を行うなどの書き直しが考えられる．

SET 命令と RST 命令は，ラッチを利用した論理回路に変換する．これは SET/RST 命令の機能が状態の保持になっているためである．具体的には下図 2.8 のように変換する．

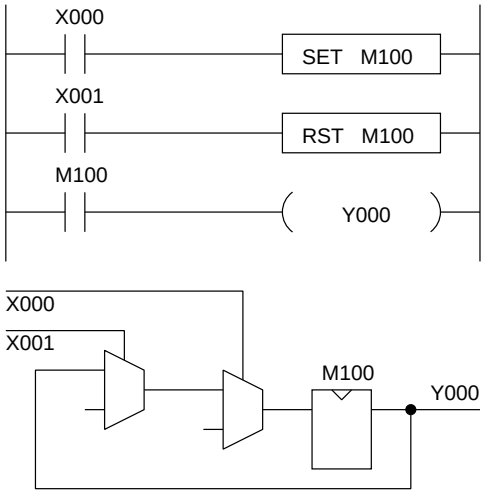

図 2.8: SET/RST 命令の変換例

PLS 命令と PLF 命令は，微分回路を利用した図 2.9 のような論理回路に変換する．

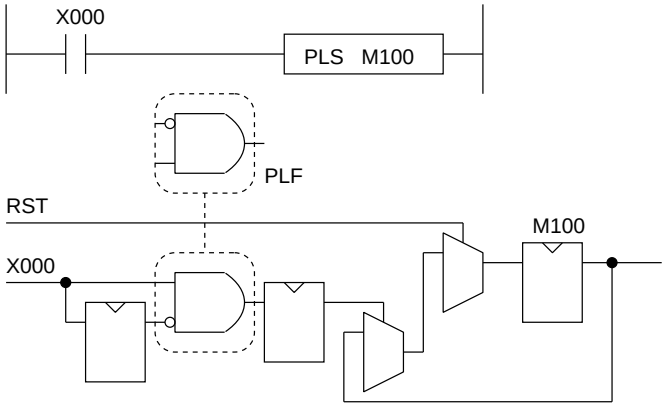

図 2.9: PLS/PLF 命令の変換例

ここで PLS/PLF 命令の変換に関する注意点について説明する．この PLS/PLF 命令は，ラダーの条件部分を入力に受けて，立上りまたは立下りのエッジ検出を出力とする．このとき，微分回路を利用している性質上，そのエッジが出力されるまでに 3 クロック必要となる．

PLS/PLF 命令の直後に，エッジ検出の結果を保持する内部リレーが参照されることもあり得るため，3 クロック待って次の段の処理に進むような変換を行う．したがって，PLS/PLF 命令に関しては，状態を 3 つ確保する．先の例を用いて表現すると図 2.10 のようになる．

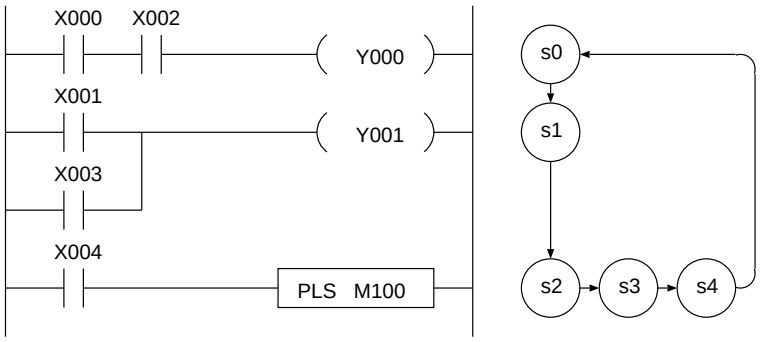

図 2.10: PLS/PLF 命令を利用した場合の，実際の状態遷移

2.2.6 応用命令から H/W への変換

MOV 命令は，転送元のデータレジスタと転送先のデータレジスタをつなぎ，そのイネーブル制御に条件部分をつなぐような回路に変換する．非常に簡単な変換なので図は省略する．

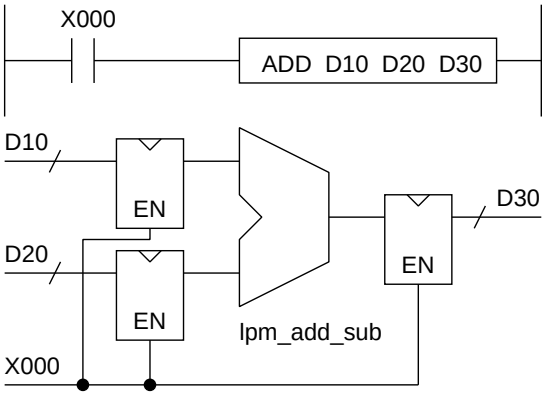

図 2.11: 算術演算系命令の変換例

最後に ADD，SUB，MUL，DIV 命令について説明する．これらの演算命令を実現するには，それぞれ加算器，減算器，乗算器，除算器が必要となる．しかし，今回はそれらを一から設計することはせずに，Altera 社の LPM 演算器 [16] を利用した（図 2.11 参照）．

また，これらの演算は，次節で説明する使い回しのため 1 クロック目で接続先の確定（切り替え）と演算を，2 クロック目で演算結果を格納するレジスタの更新を，というように動作する．したがって，演算命令は状態を 2 つ確保する．

2.2.7 演算命令に関する変更

本変換手法では，サンプルの演算命令の使用法を参考にして，論理規模を小さくするための変更を行った．具体的には，32 ビット乗算命令によって得られる 64 ビットのデータについて下位 32 ビットのみを利用し，上位ビットはテンポラリなレジスタに置くような変更を行った．除算命令によって得られる剰余データに関しても同様の変更を行った．

また，演算器のインスタンス化についても変更を行った．具体的には，演算に対応する演算器が一つインスタンス化してあれば，それを何度も使いまわすような変更である．これは，ハードウェア側で逐次的な処理を採用していることにより，同じタイミングで演算器を使う場面がないことが理由である．演算器は，特に除算器は論理規模が大きいので，除算命令が多数あるようなサンプルでは論理規模の節約が期待できる．

2.2.8 検討対象としたオペランド

本研究では，表 2.2 に示した 6 種のオペランドについて変換手法を検討した．表 2.2 にあるオペランドは，サンプルの FX2N 用 PLC プログラムで利用されている最小限のものである．表に記載した機能欄より詳しい説明が必要な場合は，メーカの提供するマニュアル [14] を参照いただきたい．

表 2.2: 検討した入出力の一覧

入出力	機能
X	外部入力，スイッチ
Y	外部出力，コイル
M	内部出力，補助リレー
K	定数
T	タイマ
D	データレジスタ

入力スイッチ X とコイル出力 Y は，ハードウェアの入出力と一対一で対応するように変換する．内部リレー M とデータレジスタ D は，ハードウェア内部のレジスタに変換する．定数 K は，ハードウェア記述中で定数としてそのまま利用される．タイマ T は，クロック用のカウンタと時間用のカウンタの二つのカウンタから構成されるタイマに変換する．

2.3 変換手法の実装

前節までに説明してきた変換手法にしたがって，PLC の命令列から論理回路を作成することができる．ここでは，それをプログラムとして実装するための方法について説明する．基本となるロジックは次のようなものである．

- 複数の命令列を受け取って，それを条件部と処理部に分けて認識する
- 条件部は，AND，OR，NOT からなる論理回路に変換する
- 処理部は，そこに設定された命令に合わせた回路に変換する

条件部と処理部の認識は，受け取る命令の次に来る命令を先読みすることで行う．つまり，次の命令を先読みしてそれがコイル出力やデータ処理を行う命令であれば，そこまでを条件部と判断し，次の命令が基本命令であれば条件部が続くと判断するような方法を用いる．

2.3.1 処理フェーズの概要

変換手法のプログラム実装は，基本的なコンパイラを作成する要領で進める．このコンパイラの入力は，PLC の命令列を記述したテキストファイルである．また出力は，ハードウェア記述言語 VHDL で記述されたハードウェアの設計情報テキストファイルである（図 2.12 参照）．

ここで，コンパイラの処理フェーズは，図 2.12 に示されるように字句解析，構文解析，コード生成に分けられる．それぞれ順を追って説明する．

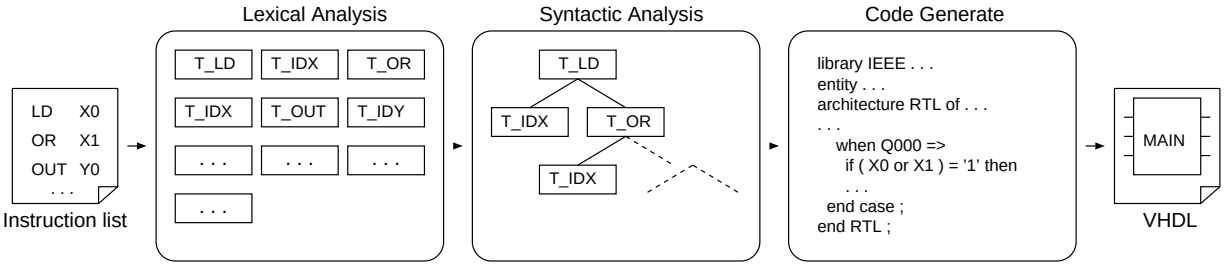

図 2.12: 変換手法の処理フェーズ

2.3.2 字句解析

PLC 命令列の記述されたテキストファイルを解析し，その解析結果に基づいて，命令語およびオペランドに対応したトークンを生成するフェーズである．

```
0       LD              X000
1       OR              X001
2       OUT             Y000
3       LD              X002
4       OUT             Y001
5       END
```

図 2.13: PLC FX2N の命令列の記述されたテキストファイルの例

例えば図 2.13 のようなテキストファイルが入力された場合，まず行番号 0 に対応するトークン，次に論理演算開始命令 LD に対応するトークン，その次に入力スイッチ X000 に対応するトークン，といったような順番でトークンを生成する．

2.3.3 構文解析

字句解析部フェーズによって生成されたトークンはそれぞれ個別に生成されたものであり，並び方などの整理は行われていない．そこで，構文解析フェーズによってそれらトークンの並び方を整理して，構文木と呼ばれる木構造のデータを構成する．

2.3.4 コード生成

構文解析フェーズによって構成された構文木を利用して，実際に VHDL のコードを出力するのが，コード生成フェーズである．

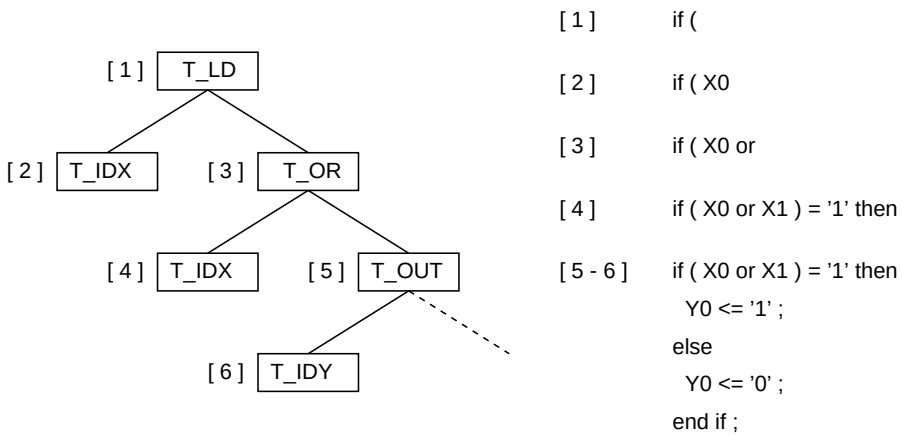

図 2.14: ノードのトレースと VHDL への言語変換

具体的には，構文木をその根（ルート）から順に深さ優先探索でトレースし，各ノードに対応した VHDL のコードを出力する処理を行う．例えば図 2.14 左に示したような構文木があった場合，まず構文木の根となっているノード T_LD から処理を開始する．

ここで，T_LD は論理演算開始命令に対応するトークンであるため，ここから先が条件部であることは明らかである．そこで，それを認識して“ if （ ”なる文字列を出力する．次に，そのノードの左の子になっている T_IDX は，論理演算開始命令のオペランドである．よって，そのオペランド“ X0 ”を出力する．このように，ノードをトレースしながら構文木から VHDL への言語変換を行う．なお，このとき条件部と処理部の分かれ目を認識するため，常に次の命令語ノードを参照する．

2.3.5 コンパイラ作成技術と VHDL 記述

コンパイラ作成技術に関しては，それだけで一冊の本が書けるようなものであるし，多くの良い教科書 [17] [18] [19] があるため詳細な説明は省略する．これらの教科書では，本節で説明した字句解析と構文解析のほかに，意味解析や最適化などについても説明されているので，詳細な説明が必要な場合は参照していただきたい．

また，コンパイラの出力となる VHDL の記述スタイルや文法に関しても，いくつかの良い教科書 [20] [21] [22] があるため詳細な説明は省略する．付録に VHDL 記述を扱った説明を記載したのでそちらを参照いただくか，またはこれらの教科書を参照していただきたい．

2.4 フレームワークの必要性

前節までに，PLC の命令列を対象とした変換手法とその実装方法について説明した．ここで，本変換手法の特徴をまとめると次のようになる．

- 現実の PLC として三菱電機製の PLC FX2N の命令セットに対応した変換手法である
- ラダー図表現から命令語表現への変換はメーカがサポートしているため，一定の制限内で，過去にラダー図で設計された PLC プログラム資産の再利用が可能である
- 制御論理については検討したが，その出力を受ける周辺回路に関しては未検討である
- 周辺回路まで含めたシステムとして実装しなければ，実用的な制御の実現は困難である
- PLC と FPGA との比較評価に関しては未検討である

そこで，制御論理のハードウェア化と周辺回路との接続を考慮し，かつ PLC と FPGA との比較評価の検討まで行えるようなフレームワークが必要となる．次章ではこれを提案する．

第 3 章

フレームワーク

3.1 フレームワークの機能

本研究で提案するフレームワークを図 3.1 に示す.

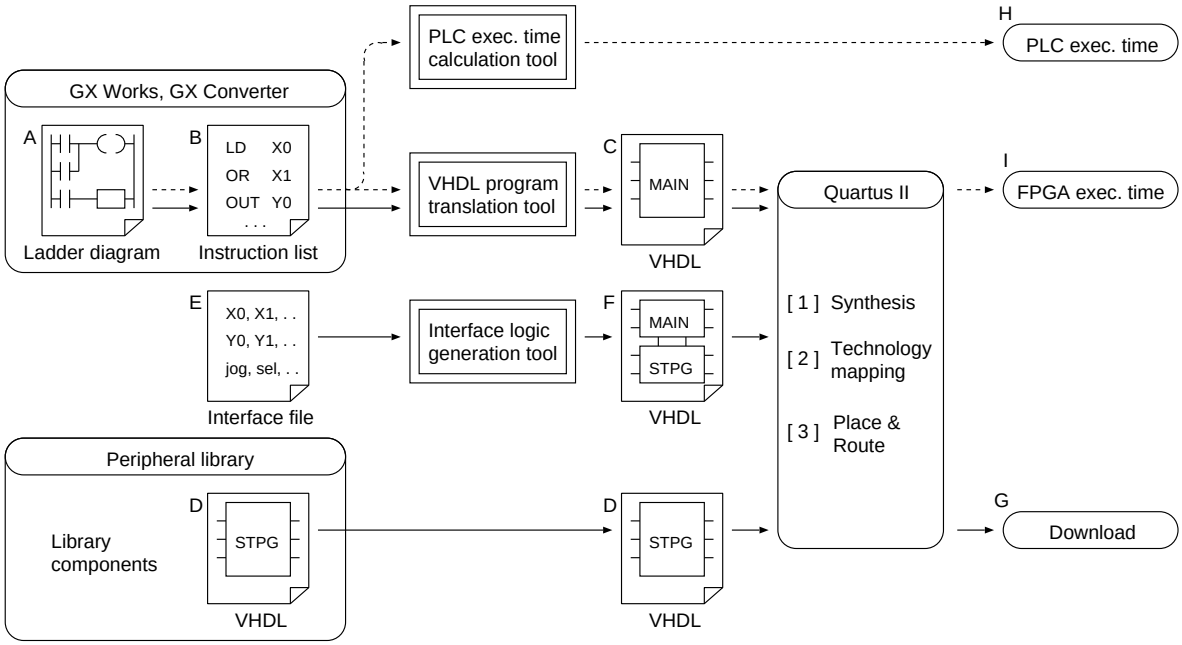

図 3.1: 提案するフレームワーク

図 3.1 のフレームワークでは，左にあるオブジェクト（A，B，D，E）を入力として与えると，右にあるオブジェクト（G，H，I）を出力として得ることができる．それぞれのオブジェクトが持つ意味は次のとおりである.

- A - ラダー図表現の PLC プログラム
- B - 命令語表現の PLC プログラム
- D - 周辺回路ライブラリから提供される，ステッピングモータ駆動機能を持つ周辺回路
- E - PLC プログラムの機能を持つ回路と周辺回路との接続情報を記述したテキスト
- G - FPGA 上へのダウンロード
- H - ある制御論理を PLC 上で実行した場合の実行時間
- I - ある制御論理を FPGA 上で実行した場合の実行時間

まず，フレームワークにおいて出力となっているオブジェクトについて説明する．オブジェクト G は，PLC プログラム本体とその周辺回路の持つ機能を実現したハードウェア（ビットストリーム）が FPGA 上にダウンロードされることを意味する．これは，PLC で実現していた制御論理を FPGA による論理回路で実現する，ということに相当する．

次に，オブジェクト H と I は，ある制御論理を PLC 上と FPGA 上でそれぞれ実行した場合の実行時間を意味する．これにより，PLC と FPGA との間で実行時間比較が可能となる．

図 3.1 において，入力から出力へ至るルートが二つあるのは，上述の二つの目的に対応するためである．以降でもう少し詳しく説明する．

3.1.1 普通線ルート：制御論理の実現

図 3.1 に示した普通線ルートは，PLC で実現していた制御論理を FPGA による論理回路で実現する，という目的を達成するためのものである．図中のオブジェクトに合わせてこのルートを辿ると，次のようになる．

1. A - ラダー図を利用して任意の PLC プログラムを作成する
2. B - ラダー図表現の PLC プログラムを命令語表現に変換したテキストファイルを得る
3. C - 上述のテキストを VHDL 変換ツールで処理して，メインとなる論理回路を得る
4. D - 周辺回路ライブラリに，目的に沿う周辺回路があることを確認する
5. E - メイン論理回路と周辺回路の接続情報を記したファイルを作成する
6. F - 接続情報をインタフェース生成ツールで処理して，最上位レベルの論理回路を得る
7. G - C，F，D の三つの設計情報を EDA ツールで処理し，FPGA 上にダウンロードする

このルートの出発点は，ラダー図を利用して任意の PLC プログラムを作成するところである．このときの PLC プログラミング環境には GX Works を利用する．次に，GX Converter を利用して，ラダー図表現の PLC プログラムから命令語表現の PLC プログラム，ここでは PLC 命令列の記述されたテキストファイルを得る．

さらに命令語表現のテキストファイルを VHDL 変換ツールで処理して，メインとなる論理回路の設計情報に変換する．このとき，PLC による制御システムの置き換えを考えると，メインの他に周辺機器を動作させるための周辺回路も必要である．そこで，本研究室で整備されつつある周辺回路ライブラリ [23] を利用する．

例えばモータを動作させる応用の場合，そのための周辺回路 STPG が用意されているので，メインと周辺回路との連携が可能である．このとき，これらの接続関係を記述した最上位レベルの論理回路が必要となる．そこで，メインと周辺回路との接続関係を記述した接続情報ファイルを作成し，それをインタフェース生成ツールで処理する．これにより，最上位レベルの論理回路を得ることができる．

最後に，メイン論理回路，周辺回路，最上位レベル回路の三つの設計情報を EDA ツール，ここでは Altera 社の Quartus II 4.0 で処理する．論理合成，テクノロジマッピング，配置配線といった一連の処理によって，FPGA 上で動作するビットストリームを得ることができる．これを FPGA 上にダウンロードして，制御論理の実現が完了する．

3.1.2 破線ルート：実行時間の比較

図 3.1 に示した破線ルートは，PLC と FPGA との間で実行時間を比較する，という目的を達成するためのものである．先の例と同様にしてこのルートを辿ると，次のようになる．

1. A - ラダー図を利用して任意の PLC プログラムを作成する
2. B - ラダー図表現の PLC プログラムを命令語表現に変換したテキストファイルを得る
3. C - 上述のテキストを VHDL 変換ツールで処理して，メインとなる論理回路を得る
4. H - 同じテキストを PLC 実行時間計算ツールで処理して，PLC の実行時間を見積もる
5. I - メインの回路を EDA ツールで処理し，その結果から FPGA の実行時間を求める

こちらのルートも PLC プログラムの作成（入手）から始まる．ここで，まず周辺回路との通信等を省略して制御論理のみを抽出する．これは，制御論理のみを用いた実行時間比較を行うためである．この PLC プログラムを，先の例と同様に命令語表現のテキストファイルに変換し，さらに VHDL 変換ツールで処理してメインの論理回路へと変換する．

同じテキストファイルを PLC 実行時間計算ツールで処理して，PLC 上での 1 スキャンにかかる実行時間を見積もる．また，メイン論理回路の設計情報を EDA ツールで処理する．Quartus II の実行結果から回路の動作可能周波数が求まるので，クロックサイクル時間と 1 スキャンに必要な状態遷移数との積をとって FPGA 上の実行時間を求める．ここまでで PLC と FPGA との実行時間比較が可能となる．

3.2 フレームワークを構成する各種ツール

先のフレームワークの図を再掲したものを下図 3.2 に示す.

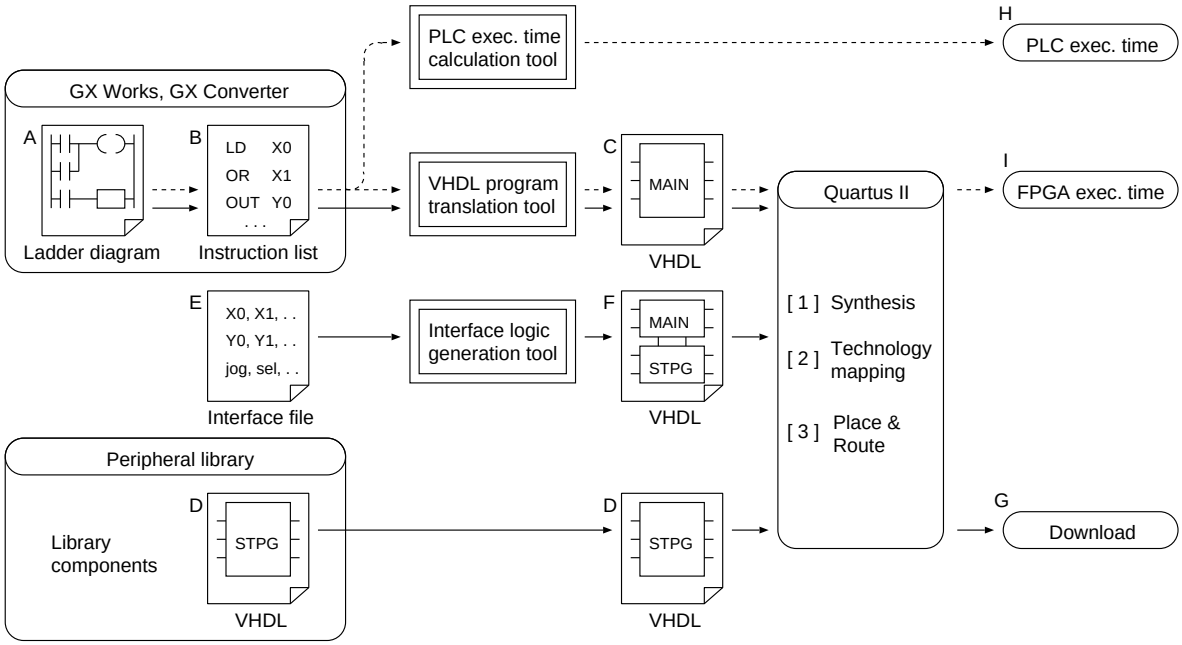

図 3.2: 提案するフレームワーク（再掲）

本節では，フレームワークを構成する各種のツールについて説明する．図 3.2 中の二重四角は，フレームワーク構成のために実装したツールである．また角丸四角は，PLC メーカと EDA ベンダから提供されるソフトウェアまたは本研究室で整備中のライブラリ群である.

3.2.1 VHDL 変換ツール

まず VHDL 変換ツール（図 3.2 の `VHDL program translation tool`）について説明する．これは，2 章で検討した変換手法をプログラムとして実装したものであり，一定の制限内で，PLC プログラムを VHDL で記述した論理回路へと変換するコンパイラである．プログラムのモジュール構成はおおまか次のようになっている.

- 字句解析
  - 命令語，オペランド（入出力信号）に対応するトークンの生成
  - 命令語専用リストのためのトークン生成
  - オペランド専用リストのためのトークン生成

- 構文解析
  - 命令語とオペランドのトークンをノードとする木構造の構成
  - 不要なトークンの解放
  - 命令語専用リストの作成/整合性チェック
  - オペランド専用リストの作成/整合性チェック
- コード生成（VHDL コードの出力）
  - パッケージ呼び出しの出力
  - エンティティ宣言の出力
  - アーキテクチャの開始部分の出力
    - PLS/PLF/タイマのコンポーネントおよびそれに必要な信号の宣言
    - LPM 演算器のコンポーネントおよびそれに必要な信号の宣言
    - 上記信号の初期値設定/信号接続
  - 状態遷移を管理する記述の出力
  - 命令語に対応する記述の出力
  - アーキテクチャの終了部分の出力
- 処理の進行を管理するメイン
  - 文法的に正しい VHDL コードが生成されるように，順番に各種のモジュールを呼ぶ
  - 各種フラグ管理などの，上記以外の雑多な処理

コンパイラの出力を VHDL 記述としたのは，筆者が VHDL を学んでいたためである．実際にハードウェアが得られるならば，VerilogHDL や Handel–C で記述しても良いと考える．

3.2.2 PLC 実行時間計算ツール

次に PLC 実行時間計算ツール（図 3.2 の `PLC exec. calculation tool`）について説明する．これは，命令語表現のテキストファイルを入力として，それを PLC 上で実行した場合の実行時間を見積もるプログラムである．このプログラムで対象とした PLC は，三菱電機製の PLC FX2N–16MT である．処理のおおまかな流れを次に示す．

1. テキストファイルに記述された命令語を上から順にフェッチする
2. プログラム内部にある実行時間テーブルを索いて，命令に対応する実行時間を得る（このとき，オペランドの種類によっても実行時間が変わることに注意する）
3. ON 時/OFF 時の実行時間を考慮して，最大，最小，中間の実行時間値を積算する
4. テキストファイルの終わりまで上記処理を繰り返す

図 3.3 は，この処理によって PLC 上の実行時間が求まる様子を示したものである．

```
テキストファイルの内容          命令の実行時間（us）          トータルの実行時間
-------------------------------------------------------------------------------
0     LD      X000              0.08                    0.08
1     OUT     Y000              0.08                    0.16
2     LD      X001              0.08                    0.24
3     AND     X002              0.08                    0.32
4     OUT     Y001              0.08                    0.40
5     END                     581.60                  582.00
```

図 3.3: PLC プログラムを入力として，PLC 上の実行時間を見積もる例

PLC の各命令語には，それぞれ実行時間が設定されている．例えば図 3.3 から明らかなように，LD という命令語には 0.08 という実行時間が設定されている．これ以外の命令の実行時間については，FX2N のプログラミングマニュアル [14] に詳しく記載されているので，そちらを参照していただきたい．ここでは，下表 3.1 に紹介する範囲で留める．

表 3.1: 基本命令の実行時間一覧表

命令語	ON 時実行時間（us）		OFF 時実行時間（us）	
	16 ビット命令	32 ビット命令	16 ビット命令	32 ビット命令
LD LDI AND ANI	0.08（M1536〜M3071 のとき，0.16） 0.08（M1536〜M3071 のとき，0.16） 0.08（M1536〜M3071 のとき，0.16） 0.08（M1536〜M3071 のとき，0.16）			
・・・	・・・			
OUT Y，M	0.08（M1536〜M3071 または M8000 以上のとき，0.16）			
・・・	・・・			
END	508+3.5X+5.7Y		−	

表 3.1 中の実行時間の定義部分には“ …のとき，… ”という但し書きが多く設定されている．そのため，プログラム側では逐一命令語とオペランドを確認し，積算する値が適切になるように注意する必要がある．

また，END 命令についても注意する必要がある．この命令の実行時間の定義部分では $508 + 3.5X + 5.7Y$ という式が設定されている．この X と Y は PLC の入出力点数を意味しており，例えば本プログラムで対象としている FX2N–16MT の場合，それぞれ 8 となる．唯一 PLC の入出力点数によって実行時間が変化する命令なので，注意する必要がある．

基本命令のみであれば以上のような計算方法でも問題はないが，PLC には応用命令と呼ばれる命令もあるため，もう少し計算方法が考えなければいけない．具体的には，ON 時実行時間と OFF 時実行時間の項目に注意する必要がある．

ON 時実行時間とは，ラダー図でいう条件部が真になり，処理部にある応用命令が実行された場合の実行時間である．一方，OFF 時実行時間はその逆で，条件部が偽のため，処理部が実行されなかった場合の実行時間である．表 3.2 にこの例を示す．

表 3.2: 応用命令の実行時間一覧表

命令語	ON 時実行時間（us）		OFF 時実行時間（us）	
	16 ビット命令	32 ビット命令	16 ビット命令	32 ビット命令
MOV	1.52	1.84	1.52	1.84
・・・	・・・			
ADD	27.6	28.9	6.4	6.4
SUB	27.6	28.9	6.4	6.4
MUL	25.2	31.4	6.4	6.4
DIV	32.0	36.4	6.4	6.4
・・・	・・・			

表 3.2 に見られるように，多くの応用命令では ON 時/OFF 時の実行時間が異なっている．そのため，PLC プログラム中の応用命令がすべて実行された場合，されなかった場合，その中間値という形で 3 種類の値を求めておくことが必要となる．なお，4 章の比較評価には中間値を採用する．

PLC 実行時間計算ツールの実装は，非常に VHDL 変換ツールに似た構成になっている．これは，入力が命令語のテキストファイルであることに起因している．

3.2.3 インタフェース生成ツール

最後にインタフェース生成ツール（図 3.2 の `Interface logic generation tool`）について説明する．これは，メインの論理回路と周辺回路の接続関係を記述したテキストファイルを入力として，それらをつないだ最上位レベルの論理回路をを出力するプログラムである．処理のおおまかな流れは次に示したとおりである．

1. テキストファイルに記述された内容からメインの入出力を確定する
2. メインの入力と周辺回路の入出力をもとに最上位レベルの入出力を確定する
3. 最上位レベルの論理回路の入出力を定義する
4. メインと周辺回路のコンポーネントを定義し，それらの間の信号接続を行う

現在の状況としては，ステッピングモータ駆動用の周辺回路 STPG のみを想定した実装になっている．具体的には，プログラムの内部に STPG のインタフェース情報をデータとして持っている．このプログラムの入力となるテキストファイルの例を下図 3.4 に示す．

```
X000,X001    // 1 行目はメインの入力
Y000,Y001    // 2 行目はメインの出力
jog,sel      // 3 行目は 2 行目の出力に対応する STPG のポート名
```

図 3.4: メインと周辺回路の接続情報を記述したテキストファイル

インタフェース生成ツールは，例えば図 3.4 のテキストファイルを入力とすると，図 3.5 のような最上位レベルの論理回路を出力する．これは実際には VHDL で記述されている．

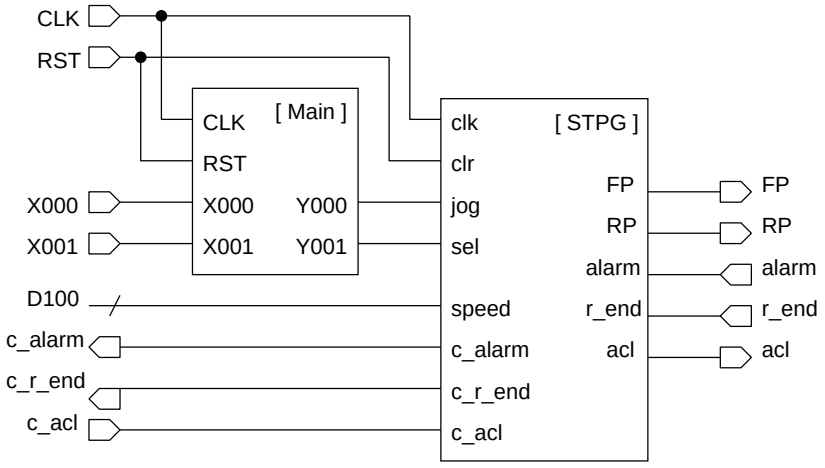

図 3.5: メインの論理回路と周辺回路 STPG をつなげた，最上位レベルの論理回路

第 4 章

実験

4.1 制御論理の実現に関するテスト

3 章で提案したフレームワークの普通線ルート，すなわち PLC で実現していた制御論理を FPGA による論理回路で実現するルートに関してテストを行った．テストには，モータの正逆転制御を行う PLC プログラムを用いた．この PLC プログラムの入出力は次のとおりである．

- 入力：動作スイッチ X000，停止スイッチ X001，回転方向選択スイッチ X002
- 出力：動作信号 Y000，回転方向決定信号 Y001

また機能は次のとおりである．

- 動作スイッチ X000 の ON/OFF にあわせて，動作信号 Y000 を ON/OFF する
- X000 が ON でも，停止スイッチ X001 が ON した場合は動作信号 Y000 を OFF する
- 回転方向スイッチ X002 の ON/OFF にあわせて，回転信号 Y001 を ON/OFF する

この PLC プログラムをフレームワークの入力として，次のような処理フローを辿り，正常にモータが動作することを確認した．

1. A - モータの正逆転制御論理を記述した PLC プログラムを作成する
2. B - ラダー図表現の PLC プログラムを命令語表現に変換したテキストファイルを得る
3. C - 上述のテキストを VHDL 変換ツールで処理して，メインとなる論理回路を得る
4. D - 周辺回路ライブラリに，モータ駆動用周辺回路 STPG があることを確認する
5. E - メイン論理回路と周辺回路の接続情報を記したファイルを作成する
6. F - 接続情報をインタフェース生成ツールで処理して，最上位レベルの論理回路を得る
7. G - C，F，D の三つの設計情報を EDA ツールで処理し，FPGA 上にダウンロードする

このとき，動作確認用のモータにはオリエンタル社のステッピングモータ [24] を用いた．また FPGA ボードには Altera 社の Nios 開発ボード [25] を用いた．ステッピングモータと FPGA ボードをつなぐインタフェース回路やモータドライバ等に関する詳細は，山本の修士論文 [23] に説明されているので，必要な場合はそちらを参照していただきたい．

4.2 PLC と FPGA との比較評価

次に，フレームワークの破線ルート，すなわち PLC と FPGA との間で実行時間を比較するルートを辿って，性能改善に関する比較評価を行った．比較対象の PLC プログラムには企業から入手したサンプルを用いた．ここで対象としたサンプルは，守秘契約を結んで入手した A 社のサンプルと，共同研究によって入手した八洲熱学 [26] のサンプルである．

本節では，これらサンプルを PLC 上と FPGA 上で実現した場合の実行時間や論理規模，そのほかのデータを測定して評価，考察する．

4.2.1 A 社のサンプル

このサンプルの特徴は次のようなものである．

- 算術演算命令が多数使われている他，PLS/PLF 命令などが使われている
- サーボモータを制御するための SVST 命令が使われている

現時点では，サーボモータを制御する周辺回路がライブラリにまだ無いため，SVST 命令を省略した制御論理のみの部分を対象とした．また，このサンプルはもともと三菱電機の PLC A シリーズの命令で記述されていたので，評価のため PLC FX2N の命令に書き直した．

4.2.2 八洲熱学のサンプル

このサンプルの特徴は次のようなものである．

- 基本的に論理演算のみで算術演算などが使われていない
- ステッピングモータの制御用周辺回路との通信のため，TO/FROM 命令が使われている

ステッピングモータの制御用周辺回路としては STPG が実装されている [23] が，これは実際の制御用周辺回路をカスタムした実装になっているため，フェアな実行時間比較を行えないという問題がある．したがって，こちらも TO/FROM 命令を省略した制御論理のみの部分を対象とした．

4.2.3 評価環境

評価環境を次の表 4.1 に示す.

表 4.1: 評価環境

項目	内容
PLC プログラムから VHDL への言語変換	Athlon XP 2600+，Memory 1 GB，Vine Linux 2.6r1, VHDL 変換ツール（自作プログラム), flex 2.5.4，bison 1.35，gcc 2.95.3
PLC 実行時間の見積もり	Athlon XP 2600+，Memory 1 GB，Vine Linux 2.6r1, PLC 実行時間計算ツール（自作プログラム), flex 2.5.4，bison 1.35，gcc 2.95.3
論理合成, テクノロジマッピング, 配置配線	Athlon XP 2600+，Memory 1 GB，WindowsXP SP1, Altera Quartus II 4.0（最適化はデフォルト設定を利用), アーキテクチャは APEX20KE, デバイスパッケージは EP20K600EBC652–3

4.2.4 評価結果

表 4.1 の評価環境のもとで測定したデータを下表 4.2 に示す.

表 4.2: 各サンプルの評価結果

		実行時間（sec）	論理規模	ステップ数	ピン数	回路生成時間（sec）
A 社	PLC	1.25×10^{-3}	—	538	—	—
	FPGA	4.36×10^{-5}	4242 LEs	—	393	748
八洲	PLC	5.89×10^{-4}	—	93	—	—
	FPGA	1.21×10^{-7}	68 LEs	—	29	46

さて，まず実行時間について説明すると，どちらのサンプルにおいても FPGA による実行時間の短縮（処理速度面での性能改善）が確認された．A 社のサンプルでは約 28 倍，八洲熱学のサンプルでは約 4800 倍高速にスキャンすることが可能だと分かった.

次に論理規模について説明する．このデータに関しては，二つのサンプルで非常に大きな差が出た．A 社のものは 4242 LEs を消費するのに比べ，八洲熱学のものは 68 LEs と非常に論理規模が小さいという結果になった．このように八洲熱学のサンプルの論理規模が小さくなるのは，次のことが理由である．すなわち，サンプルの内容が周辺回路を含まないコアの制御論理のみであり，かつ論理演算メインの小さいプログラムだからである.

A 社の論理規模に関しては，LPM 演算器の占める割合の大きいことが分かっている．具体的な内訳としては，加減算器 32 LEs，乗算器 1260 LEs，除算器 1429 LEs と，それぞれ消費している．これは，全体の約 64%に相当する．残りの 36%は，演算で利用するデータレジスタと状態遷移制御回路で消費されている．

本研究の変換方針では，LPM 演算器は 1 つのみインスタンス化し，それを使いまわす方針を採用している．そのため演算数が増えた場合，演算器の消費する LE 数は変わらず，それ以外のデータレジスタと状態遷移制御回路の消費する論理規模の割合が増えると予測できる．

4.3 考察

本節では，論理規模とピン数の観点からの考察を示す．まず，前節の評価で用いた FPGA ファミリ APEX20KE のデータを下表 4.3 に，それをグラフ化したものを図 4.1 に示す．

表 4.3: APEX20KE の論理規模，ピン数

デバイス	論理規模（LEs）	ピン数
EP20K30E	1200	92〜128
EP20K60E	2560	92〜196
EP20K100E	4160	92〜246
EP20K160E	6400	88〜316
EP20K200E	8320	136〜376
EP20K300E	11520	152〜408
EP20K400E	16640	488
EP20K600E	24320	488〜588
EP20K1000E	38400	488〜708
EP20K1500E	51840	488〜808

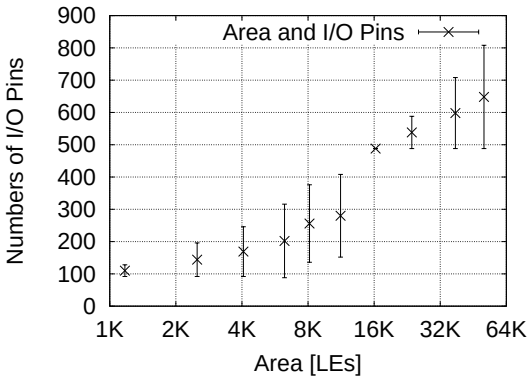

図 4.1: APEX20KE の論理規模とピン数の対応

表 4.3 と図 4.1 のピン数に幅があるのは，デバイスごとに提供されているパッケージの種類によってピン数が変わるからである．例えば，論理規模とピン数が最大レベルにあるデバイス EP20K1500E の場合，ピン数はパッケージの選択により 488 から 808 が利用可能である．

先の評価ではデバイスパッケージに EP20K600EBC652–3 を用いた．これの論理規模は 24320 LEs，ピン数は 488 である．ここで A 社サンプルの評価結果を再度参照すると，論理規模は 4242 LEs，ピン数は 393 を消費している．このとき，論理規模は全体の約 17%しか消費していないのに比べて，ピン数は全体の約 80%も消費していることが分かる．

このことは，大規模な PLC プログラム応用へ FPGA を適用する場合に，論理規模よりもピン数の制限が強く影響することを示唆する．ピン数は最高位のデバイスを選択したとしても高々 800 程度にしかならないので，デバイスの選択だけでこの問題を解決することは難しい．

この問題の緩和方法としては，次に示す二つの方法が考えられる．

- FPGA ボード上に用意されたインタフェースを利用する
- 複数デバイスを利用する

一つ目は，FPGA ボード上にあらかじめ用意されているシリアルやイーサネットのインタフェースを介してデータのやり取りを行い，ピン数を節約する方法である．この方法を採用する場合には，シリアル通信あるいは TCP/IP 通信を実現するアプリケーションを FPGA 上に構成している必要がある．FPGA 上にソフトマクロの CPU を構成し，その上でリアルタイム OS を走らせてアプリケーションを動作させる手段をとれば，比較的小さい労力で実現できる．

二つ目は，制御論理を複数デバイスに分割して実装し，ピン数をできるだけ多く確保する方法である．この方法では，論理規模の小さいデバイスを複数利用すると効果的と思われる．この方法を採用する場合には，複数デバイスに分割された機能が正しい順序で実行されるように，通信を行って管理するモジュールが別途必要となる．

第 5 章

結論

5.1 まとめ

本研究では，現実の PLC として三菱電機製の FX2N を想定した変換手法について検討した．また，変換手法そのものの制限について言及し，実用的なシステムを FPGA 上に実現するためのフレームワークを提案した．

さらにフレームワークを構成する各種のツールを実装し，テスト例を用いた動作確認と，サンプルを利用した比較評価を行った．特に PLC と FPGA との比較評価においては，処理速度面の性能改善を確認したことのほかに，論理規模とピン数に関する考察を行った．

5.2 今後の課題

フレームワークを構成する各種のツールを充実させることで，さらに比較評価を進めることが期待されるため，次に示した作業を進めたい．

- VHDL 変換ツールの対応命令の追加
- インタフェース生成ツールの対応周辺回路の追加
- PLC 実行時間計算ツールの対応 PLC の追加

また，今回は PLC プログラムのハードウェア化にあたって逐次処理を採用した．そのため，性能面の向上が最小限となってしまった．2 章でも触れたように，変換手法に対して並列化コンパイラの技術を導入し，さらなる高速化を目指したいと考える．

付録 A

VHDL 変換ツールに関する補足説明

A.1 はじめに

本節を含む付録では，VHDL 変換ツールに関して本文で省略した実装部分を説明する．図 A.1 に再掲したように，VHDL 変換ツールの処理フェーズはおおまかに字句解析，構文解析，コード生成に分けられる．それぞれ順を追って説明する．

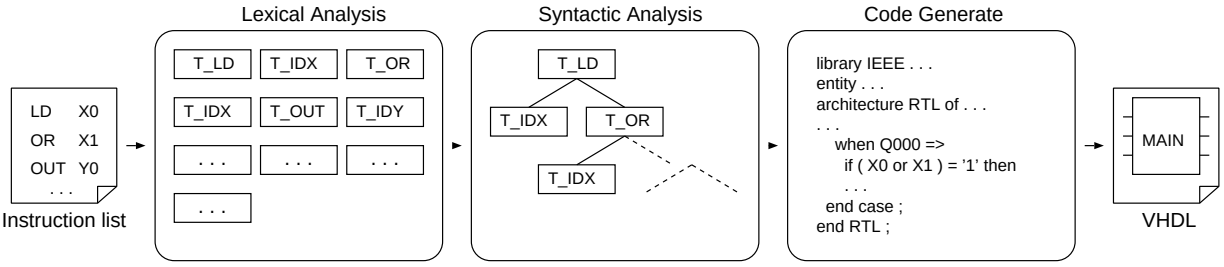

図 A.1: 変換手法の処理フェーズ（再掲）

A.2 字句解析

まず字句解析フェーズでは，入力となるテキストファイル（PLC プログラム）を受け取って，その内容をメモリ上に読み込む．これは，テキストの情報をそのまま扱った場合，字句を最初から読んでいくことしかできないため，このあとの処理につなぎにくいからである．

このとき，メモリ上のデータはトークンという単位で扱う．トークンは，C 言語の構造体を用いて実装する．図 A.2 のリストは，その構造体を定義したヘッダファイルの一部分を示したものである．

```
// ・・・
// 構文木セル
typedef struct ptree_cell {
  int               type;   // トークンタイプ
  char              *str;   // 識別子の文字列
  struct ptree_cell *next;  // 次の命令語へのポインタ
  struct ptree_cell *opr1;  // 第一オペランド
  struct ptree_cell *opr2;  // 第ニオペランド
  struct ptree_cell *opr3;  // 第三オペランド
  struct ptree_cell *opr4;  // 第四オペランド
} PTREECELL;

typedef PTREECELL     *PCELLPNT;

// 構造体の各メンバにアクセスするためのマクロ定義
#define NULLPCELL     ((PCELLPNT)NULL)
#define CellType(x)   ((x)->type)
#define CellStr(x)    ((x)->str)
#define NextCell(x)   ((x)->next)
#define CellOpr1(x)   ((x)->opr1)
#define CellOpr2(x)   ((x)->opr2)
#define CellOpr3(x)   ((x)->opr3)
#define CellOpr4(x)   ((x)->opr4)
// ・・・
```

図 A.2: トークンのデータ構造を定義したヘッダファイル

構造体のメンバには，次のような情報を持たせる．

- トークンタイプ
- 識別子の文字列
- 次の命令語へのポインタ
- オペランドへのポインタ

トークンタイプは，どういう種類の命令/オペランドに対応しているかの情報である．識別子の文字列は，命令/オペランドの実際の文字列情報を格納する．また，後述する構文木を構成するために，次の命令語へのポインタおよび命令のオペランドへのポインタも定義する．

```
%option noyywrap
%{
#include "seq2vhd_sg.h"
%}

%%
LD          { yylval = make_ptree_cell(T_LD,   yytext); return(T_LD);   }
LDI         { yylval = make_ptree_cell(T_LDI,  yytext); return(T_LDI);  }
AND         { yylval = make_ptree_cell(T_AND,  yytext); return(T_AND);  }
ANI         { yylval = make_ptree_cell(T_ANI,  yytext); return(T_ANI);  }
OR          { yylval = make_ptree_cell(T_OR,   yytext); return(T_OR);   }
ORI         { yylval = make_ptree_cell(T_ORI,  yytext); return(T_ORI);  }
ANB         { yylval = make_ptree_cell(T_ANB,  yytext); return(T_ANB);  }
ORB         { yylval = make_ptree_cell(T_ORB,  yytext); return(T_ORB);  }
NOP         { yylval = make_ptree_cell(T_NOP,  yytext); return(T_NOP);  }
END         { yylval = make_ptree_cell(T_END,  yytext); return(T_END);  }
・・・
[0-9]+      { yylval = make_ptree_cell(T_NUM,  yytext); return(T_NUM);  }

[ \r\n\t]   ;  /* 空白・改行・タブを読み飛ばす */
("<".+">")  ;  /* コメントを読み飛ばす */
%%
```

図 A.3: flex 記述による字句解析系の実装

図 A.3 のリストに，flex 記述による字句解析系の実装を示す．最初の数行は，この flex 記述を flex にかける際のオプション設定やヘッダファイルのインクルードである．

リスト中の %% で囲まれた部分が字句解析系の本体である．それぞれの行の先頭に記述されているのが一つのトークンとして切り出す文字列，つまり命令語やオペランド名の定義である．それに続いて記述されているのがトークンを生成するための処理である．トークン生成処理は次のようなものである．

1. トークン構造体の分だけメモリを確保する
2. 必要であれば，トークンタイプなど適宜メンバの値を設定する
3. その構造体へのポインタを yylval という特別な変数へ代入する
4. トークンタイプをリターンする

yylval は，flex および bison を用いてコンパイラを作成する場合に重要な変数である．この変数の役割は，字句解析系によって切り出されたトークンを構文解析系に渡すことである．

A.3 構文解析

次の構文解析フェーズでは，字句解析系によって生成されたトークンの並び方を整理して，構文木を作成する．このとき，一つ一つのトークンは構文木の各ノードに対応することになる．また，不要なトークンはこの段階で切り捨てられる．

図 A.4 と図 A.5 のリストに，bison 記述による構文解析系の実装を示す．最初の数行は，この bison 記述を bison にかける際のヘッダファイル指定と構文エラーの定義である．

```
%{
#include "seq2vhd_sg.h"
void yyerror(const char *str) { fprintf(stderr, "Error: %s\n", str); }
%}

%token T_LD T_LDI T_AND T_ANI
・・・
%%
sequences : /* empty */
          | sequences sequence
          ;
sequence  : T_NUM opecode
          {
            SetNext(ptree_end, $2);
            ptree_end = $2;
          }
          | T_NUM opecode operand
          {
            SetNext(ptree_end, $2);
            SetOpr1($2, $3);
            ptree_end = $2;
          }
          ・・・
// 次ページに続く
```

図 A.4: bison 記述による構文解析系の実装（前半）

```
// 前ページからの続き
opecode   : T_LD   { $$ = $1; }
          | T_LDI  { $$ = $1; }
          | T_AND  { $$ = $1; }
          | T_ANI  { $$ = $1; }
          ・・・

operand   : T_IDX  { $$ = $1; }
          | T_IDY  { $$ = $1; }
          | T_IDM  { $$ = $1; }
          | T_IDD  { $$ = $1; }
          ・・・
%%
```

図 A.5: bison 記述による構文解析系の実装（後半）

flex 記述と同様に，リスト中の %% で囲まれた部分が構文解析系の本体であり，その直前の %token の行に記述されているのはトークンの定義である．

本体で設定されている構文定義は次のようなものである．

- **行番号　命令語**
- **行番号　命令語　オペランド**
- **行番号　命令語　オペランド　オペランド**
- **・・・**

この構文定義によって，例えばいちばん最初のものであればオペランドをもたない NOP 命令や END 命令，ANB/ORB 命令などを認識することができる．定義された構文が見つかった場合は，構文木につなげる処理を行っている．

また，リスト中の $1 〜 $n は bison 特有の記法で，それぞれ 1 番目から n 番目のトークン構造体へのポインタを表している．この記法を利用することで，構文定義で認識できたトークンに対して任意の処理，ここでは構文木への追加などを行うことが可能になっている．

A.4　コード生成

最後にコード生成フェーズについて説明する．このフェーズでは，構文解析によって構成された構文木をもとに言語変換を行う．

具体的には，構文木をその根（ルート）から深さ優先でトレースするような読み出しポインタを利用して処理を進める．読み出しポインタはあるノードに到達すると，そのノードに対応する字句を書き出す．例えば，読みだしポインタがオペランドのノードに到達した場合は，そこに記録されている信号名を出力する．

図 A.6 のリストに，C 言語によるコード生成処理の実装を示す．

```
void print_code(FILE *fp, PCELLPNT top)
{
  if(top == NULLPCELL)
    return;

  switch(CellType(top)) {
  case T_LD :
    if(flag_cond == true) {
      fprintf(fp, ") (");
      print_code(fp, CellOpr1(top));
    }
    else if((T_OUT <= CellType(NextCell(top))) &&
            (CellType(NextCell(top)) <= T_FROM)) {
      fprintf(fp, "          when Q%03d => ", state_cnt);
      fprintf(fp, "if (");
      print_code(fp, CellOpr1(top));
      fprintf(fp, " = '1') then\n");
      flag_cond = false;
    }
    else if((T_LD <= CellType(NextCell(top))) &&
            (CellType(NextCell(top)) <= T_ORI)) {
      fprintf(fp, "          when Q%03d => ", state_cnt);
      fprintf(fp, "if ((");
      print_code(fp, CellOpr1(top));
      flag_cond = true;
    }
    print_code(fp, NextCell(top));
    break;
    ・・・
```

図 A.6: C 言語によるコード生成処理の実装

図 A.6 のリストには，論理演算開始命令 LD に対応する部分のみを記載した．ここで行っている処理は，次のトークンを確認してそれに合わせた VHDL コードを出力する，しごく簡単なものである．オペランドや次の命令語の出力は，このコード生成処理を担当する関数 `print_code` を再帰的に利用することで実現した．

A.5 Handel–C 記述への対応

本研究では，論理回路の記述方法として VHDL による記述を採用した．この理由は，本研究室での主流が VHDL だったことによるもので，実際には VerilogHDL でも Handel–C でも良い．そこで，Handel–C 記述の出力も実装したのでそのことについて説明する．

Handel–C は，ANSI–C にハードウェアの概念を取り入れて拡張した言語である．基本的に，Handel–C で記述されたプログラムは，1 ステートメントあたり 1 クロックサイクル消費して実行される．またこの他，並列処理記述やハードウェアユニット間でのインタフェース記述などの機能拡張がなされている．

Handel–C においては，逐次処理したいステートメントは `seq{ }` で囲み，並列処理したいステートメントは `par{ }` で囲むことで，任意の処理を実装することが可能である．`par{ }` で囲まれたステートメントは，それぞれ並列に動作するハードウェアとして実現される．このように，逐次処理と並列処理の書き分けが簡単にできる点が Handel–C の利点の一つと言える．

今回の実装では，逐次処理のための `seq{ }` キーワードのみを使用した変換に対応したが，それでも VHDL と比べてかなりコード的に短い出力となった．現時点では，Handel–C の開発環境である Celoxica DK3 Design Suite の論理合成とシミュレーションまで可能である．今後，さらに検討を加えて評価まで進みたい．

参考文献

[1] 市川周一：FPGA による巻取制御系の開発, 豊橋技術科学大学未来技術流動研究センター年報 2004（掲載予定） (2005).

[2] 市川周一：再構成可能論理回路のプログラマブル・ロジック・コントローラへの応用に関する研究, 豊橋技術科学大学 未来技術流動研究センター年報 2002, pp. 142–146 (2003).

[3] 市川周一：再構成可能論理回路のプログラマブル・ロジック・コントローラへの応用に関する研究, 豊橋技術科学大学 未来技術流動研究センター年報 2003, pp. 131–133 (2004).

[4] Adamski, M. A. and Monteiro, J. L.: PLD Implementation of Logic Controllers, *Proc. IEEE Int'l Symp. Industrial Electronics*, Vol. 2, pp. 706–711 (1995).

[5] Wegrzyn, M., Wolanski, P., Adamski, M. A. and Monteiro, J. L.: Field Programmable Device as a Logic Controller, *Proc. Int'l Conf. Control'96*, Vol. 2, pp. 715–720 (1996).

[6] Wegrzyn, M., Adamski, M. A. and Monteiro, J. L.: The application of reconfigurable logic to controller design, *Control Engineering Practice*, Vol. 6, pp. 879–887 (1998).

[7] Miyazawa, I., Nagao, T., Fukagawa, M., Itoh, Y., Mizuya, T. and Sekiguchi, T.: Implementation of Ladder Diagram for Programmable Controller Using FPGA, *Proc. IEEE Int'l Conf. Emerging Technologies and Factory Automation*, Vol. 2, pp. 1381–1385 (1999).

[8] Ikeshita, M., Takeda, Y., Murakoshi, H., Funakubo, N. and Miyazawa, I.: An Application of FPGA to High-Speed Programmable Controller, Development of the Conversion Program from SFC to Verilog, *Proc. IEEE Int'l Conf. Emerging Technologies and Factory Automation*, Vol. 2, pp. 1386–1390 (1999).

[9] 石野智久, 土居公司：PLC（プログラマブル・ロジック・コントローラ）の高速処理における LSI 技術応用の検証, 情報処理学会シンポジウム論文集, Vol. 2003, No. 11, pp. 225–229 (2003).

[10] 製品情報 シーケンサ FX2N,

http://www.mind.ne.jp/melsec-f/plc_fx/details/fx2n/index_j.htm.
[11] 関口隆（編）：新しいプログラマブルコントローラのプログラミング— IEC 61131-3 による効率的プログラミング—, コロナ社 (2000).
[12] 離散事象システム研究専門委員会（編）：ペトリネットとその応用 創立 30 周年記念, 計測自動制御学会 (1992).
[13] 村田忠夫：ペトリネットの解析と応用, 近代科学社 (1992).
[14] 三菱電機 FX1S/FX1N/FX2N/FX1NC/FX2NC シリーズ プログラミングマニュアル, http://wwwf3.mitsubishielectric.co.jp/members/o_manual/plc_fx/jy992d62001/jy992d62001k_3.pdf.
[15] 三菱電機 三菱統合 FA ソフトウェア, http://wwwf2.mitsubishielectric.co.jp/melsoft/nproduct/403/403.pdf.
[16] 日本アルテラ Library of Parameterized Modules (LPM), http://www.altera.co.jp/products/software/products/legacy/maxplus2/sfw-lpm.html.
[17] 佐々政孝：プログラミング言語処理系, 岩波講座ソフトウェア科学, 第 5 巻, 岩波書店 (1989).
[18] 湯淺太一：コンパイラ, 情報系教科書シリーズ, 第 9 巻, 昭晃堂 (2001).
[19] A. V. エイホ, R. セシィ, J. D. ウルマン：コンパイラ：原理・技法・ツール, サイエンス社 (1990).
[20] 長谷川裕恭：VHDL によるハードウェア設計入門：言語入力によるロジック回路設計手法を身につけよう, CQ 出版, 改訂版 (2004).
[21] 仲野巧：VHDL によるマイクロプロセッサ設計入門：パソコンによるシミュレーションから論理合成, 配置配線まで, CQ 出版 (2002).
[22] 森岡澄夫：HDL による高性能ディジタル回路設計：ソフトウェア感覚を離れてハードウェアを意識する, CQ 出版 (2002).
[23] 山本浩司：FPGA による実時間制御システムに関する研究, 豊橋技術科学大学 修士論文 (2004).
[24] ORIENTAL MOTOR 技術資料, http://search.orientalmotor.co.jp/support/technical/pdf/SteppingMotor_Tech.pdf.
[25] 日本アルテラ Nios 開発ボード & キット, http://www.altera.co.jp/products/ip/processors/nios/kits/nio-dev_kits.html.
[26] 八洲熱学株式会社, http://www.yashima-ne.co.jp/index.html.

謝辞

本研究にあたって直接のご指導をいただいた市川周一先生に，御礼申し上げます．また，研究の他に様々なところで助けていただいた研究室のみなさまにも感謝いたします．ありがとうございました．