社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

LUT リングを用いた プログラマブル・ロジック・コントローラの構成法

井口　幸洋† 　鈴木　隆広† 　笹尾　勤††

† 明治大学理工学部情報科学科，神奈川県
†† 九州工業大学情報工学部電子情報工学科，福岡県
E-mail: †{iguchi,suzuki-t}@cs.meiji.ac.jp, ††sasao@cse.kyutech.ac.jp

あらまし　LUT(Look-Up Table) の出力の一部を入力にリング状に接続した LUT リングを用いてプログラマブル・ロジック・コントローラ (PLC: Programmable Logic Controller) を構成する方法を提案する．PLC で実行する命令の多くは，論理演算である．同時に実行可能な論理演算をまとめ, 全ての組合せに対する出力値を予め計算して一つの LUT に蓄える．これらの LUT を個別に実装するかわりに, 一個のメモリを繰り返しアクセスすることにより複数の LUT として実現し, 出力の一部は外部出力レジスタに, 残りは状態変数レジスタに記憶し処理を進める．PLC のプログラミング入力としては，ラダー図を用いる．LUT 内部に格納するデータ, および, 出力を外部出力や内部記憶用に振り分けるための制御回路データを生成するツールも開発した．いくつかのベンチマーク関数に対する性能評価を示し, 従来の PLC よりも高速であることを示す.

キーワード　プログラマブル・ロジック・コントローラ, ファクトリー・オートメーション, 組み込みシステム, 再構成可能アーキテクチャ, FPGA

A Design Method for a Programmable Logic Controller Using an LUT Ring

Yukihiro IGUCHI†, Takahiro SUZUKI†, and Tsutomu SASAO††

† Department of Computer Science, Meiji University, Kawasaki-city, Kanagawa, 214-8571 Japan
†† Dept. of Electronics and Computer Science, Kyushu Institute of Technology, Iizuka-city, Fukuoka, 820-8502 Japan
E-mail: †{iguchi,suzuki-t}@cs.meiji.ac.jp, ††sasao@cse.kyutech.ac.jp

Abstract　This paper presents a method to realize a programmable logic controller (PLC) with an LUT ring. Most operations used in PLCs are logical ones. In a program for a PLC, we can often implement several operations at the same time. We can also compute the output values for all possible input combinations and store them in a page of the LUT. Instead of using many LUTs, we use an LUT to implement them. We use ladder diagrams for programming of PLCs. The LUT ring consists of memories, a state register, an output register, and a programmable interconnection network. We developed a design tool for LUT rings. We also show that the emulation by LUT ring is faster than PLC based on microprocessors.

Key words　programmable logic controller, factory automation, embedded system, reconfigurable architecture, FPGA.

1. はじめに

シーケンス制御は, 機械工業の作業工程などの制御に利用されている [1], [10]．古くは電磁リレー, センサ, スイッチ等で構成されていたが, その後は IC によるハード・ワイアード制御に変わり, 1970 年代半ばからは汎用マイクロプロセッサ (MPU) を用いたプログラマブル・ロジック・コントローラ (PLC: Programmable Logic Controller) が主流となっている [10].

PLC は, プログラマブル・コントローラ (PC: Programmable Controller) とも呼称される．PLC は, プログラムを変更することで様々な作業工程に対応できるため, ファクトリー・オートメーション (FA: factory automation) において多用されるが,

より高速な PLC が必要な場合がある．例えば, ビール瓶を高速回転しながら洗浄し, その正面にラベルを貼る作業には, 高速 PLC が必要である．

高速化のために，PLC 用の専用 MPU を用いる方法 [3], [12], [13] や, PLC のプログラム入力として用いられるラダー図から高速な機械語コードを生成する方法 [4] が提案されている．

我々の研究グループでは, LUT（Look-Up Table) を直列多段に接続した **LUT カスケード**や, LUT カスケードの個々の LUT を実装する代わりにそれらをまとめて 1 つのメモリで実現する **LUT リング**などを提案している [2], [7], [9]．本稿では, LUT リングを用いて PLC を構成する方法を提案する．PLC で使用する命令の多くは，論理演算である．同時に実行可能な論理演算をまとめ, 全ての入力組合せに対する出力値を予め計算して 1 つの LUT に蓄える．これらの LUT を多数実装する代わりに, 1 個のメモリを複数の LUT として実現した LUT リングを使う．LUT リング内のメモリ (論理関数メモリ) の各ページを順次アクセスし, 一部は外部出力に, 一部は状態変数レジスタに出力し, 処理を進める．LUT リングを用いた実現は, MPU を用いた PLC に比べ高速という特徴がある．

本文の構成は、以下のとおりである。第 2 章では, シーケンス制御と PLC の処理記述に用いられるラダー図について概説する．第 3 章では LUT カスケードと LUT リングの構造と動作について概説する．第 4 章では, LUT リングに変更を加え, PLC の実現に適したアーキテクチャを提案する．また, ラダー図より生成した LUT リングと, そこで実現された PLC で必要な論理メモリと接続メモリの内容の生成方法を提案する．第 5 章では, プロトタイプシステムについて述べ, 簡単な評価実験を示す．第 6 章ではまとめと今後の課題について述べる．

2. シーケンス制御とラダー図 [1], [10]

シーケンス制御とは, 予め定められた順序, または一定の論理によって定められた順序に従って, 制御の各段階を逐次進めていく制御である．歴史的には, この制御をリレーとスイッチ等で実現していた．そのため, 制御の記述にはリレー, スイッチ等を模した部品と, タイマやセンサなどの部品を加えたラダー図と呼ばれる特別な記法を使用することが多い．

現在では, 信頼性の点からリレー等を使用する例は少ない．高速な動作が要求される用途には専用のハードウェアを用いることもあるが, 多くは MPU（マイクロプロセッサ）を内部に搭載した PLC(Programmable Logic Controller) を用いて実現する．この場合, ラダー図で表された処理手順は, MPU の命令に置換して実行する．

ラダー図に使用する記号を図 2.1 に, ラダー図の例を図 2.2 に示す．図 2.1(a) の a 接点は, 入力 X0 が ON の時，両端が導通し, (b) の b 接点は入力 X0 が OFF の時, 両端が導通する．(c) は PLC の内部状態を記憶するレジスタへの出力であり, (d) は PLC の外部出力を表す．入力は **X**, 内部状態レジスタは **M**, 外部出力は **Y** で始まるアルファベットで記し, 個々を識別するために数字を付加する．これ以外にタイマ, カウンタ, 演算装置への指令などもあるが, ここでは扱わない．

PLC 内部には, 入力値を保持する入力レジスタ, 内部状態を保持する内部状態レジスタ, 及び, 外部出力値を保持する外部出力レジスタが存在する．本稿では, 内部レジスタを M レジスタ,

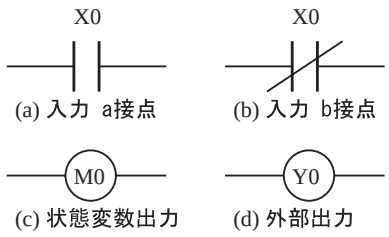

図 2.1　ラダー図で使用する部品例

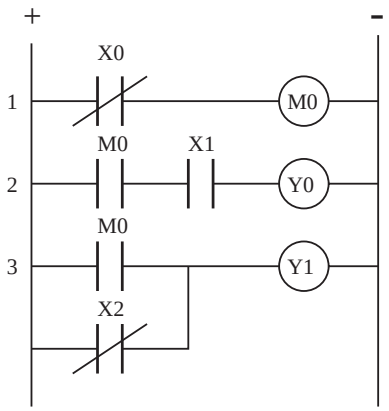

図 2.2　ラダー図の例

外部出力をレジスタを Y レジスタと呼ぶ．Y レジスタは, さらに YA レジスタと YB レジスタで構成される．

PLC は, 図 2.3 に示した動作周期で以下の動作を繰り返す．

（1） 制御システムに加えた入力値を読み取り, 入力レジスタに保持する．

（2） ラダー図で記述された所定の処理を上から順に行い, ラダー図の最終行まで各行の右端にある状態変数レジスタ, 及び, 外部出力レジスタを更新する．

（3） 外部出力レジスタに保持された値を外部に出力する．

（4） Step 1 へ戻る

1~4 までを 1 周期とし, これに要する時間を**スキャンタイム**と呼ぶ．出力値を一斉に外部に更新するためには, 出力レジスタを図 2.4 に示すようなクロックを 2 種類有するダブル・ランク構成とする．このように, ラダー図で記述されたプログラムの実行前に入力を一斉に取り込んで保持し, プログラムの最終行終了後に出力を一斉に外部に出力する本方式は, リフレッシュ方式と呼ばれる [10]．これ以外にも, スキャン中に入力値や外部出力の更新を随時行う方式（ダイレクト方式), 入力は一括で取り込み, 出力は随時更新する混合方式がある [10] が, ここでは対象としない．

図 2.2 を例として，ラダー図の動作を説明する．内部状態レジスタ M0, 外部出力レジスタ Y0, Y1 の値は, 以下の論理式で表現できる．

$$
\begin{aligned}
1 \quad & M0 = \overline{X0} \\
2 \quad & Y0 = M0 \cdot X1 \\
3 \quad & Y1 = M0 \vee \overline{X2}
\end{aligned}
$$

専用アセンブラは, ラダー図の命令をシーケンス演算の命令列に変換する．例えば, 図 2.2 のラダー図の第 1~3 行目は，以下のシーケンス命令に変換できる．ここで LD はロード命令, LDI はロード・インバース (ロードしてビットの否定をとる) 命

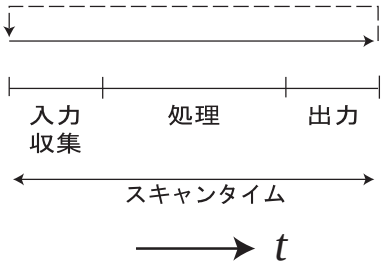

図 2.3　PLC の動作とスキャンタイム

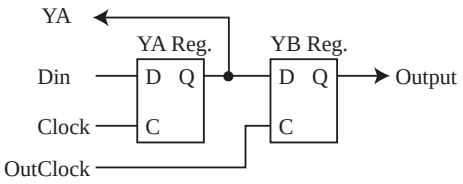

図 2.4　ダブルランク・フリップ・フロップの構造

令である．同様に ORI は, OR Inverse 命令であり, ORI X2 は, X2 の値の反転との OR をとる命令である．

```
LDI   X0    ;X0 の値を読み込み, 反転
OUT   M0    ; 上で求めた値を内部状態レジスタ M0 に出力
LD    M0    ;M0 の値を読み込む
AND   X1    ;X1 との AND を取る
OUT   Y0    ; 外部出力レジスタ Y0 に出力
LD    M0    ;M0 の値を読み込む
ORI   X2    ;X2 の NOT との OR を取る
OUT   Y1    ; 外部出力レジスタ Y1 に出力
```

PLC は, このシーケンス命令を上から順に解釈実行し，すべての命令を実行すると最初の命令に戻る．つまり, 無限ループで実行する．ラダー図の第 1 行目の出力 M0 は，第 2,3 行目の入力になっているので, 第 2, 3 行目の Y0, 及び, Y1 の値は第 1 行目の出力 M0 の結果に依存する．また, 外部への出力は 3 行全てのプログラムを実行した後に, それまでに更新された外部出力レジスタの値を外部に出力する．

3.　LUT カスケードと LUT リング

本章では，再構成可能アーキテクチャである LUT カスケードと LUT リングについて概説する [2], [7], [9].

LUT カスケードは，図 3.1 に示すように LUT (Look-Up Table) を直列多段に接続したものであり，論理関数を実現する．

FPGA では配線の自由度が高いが, 配線遅延は大きい．一方, LUT カスケードでは，隣接する LUT 間に設けた専用信号線で接続する方法をとるので，柔軟性は少ないが配線遅延は少なく，配線問題も簡単であるという特徴をもつ．

LUT カスケードでは，LUT の段数，LUT 間の接続線数 (レール数)，LUT の入出力線数を固定にすると自由度が低くなるので，レール間に高速なスイッチを設けたりする等の工夫が必要である [2]．そこで，より柔軟性の高いアーキテクチャとして, 複数の LUT を一つのメモリで実現する LUT リングを提案している [6], [7], [9]．LUT リングの構造を図 3.2 に示す．LUT をメモリの各ページに格納し，外部入力の一部と，出力の一部を，

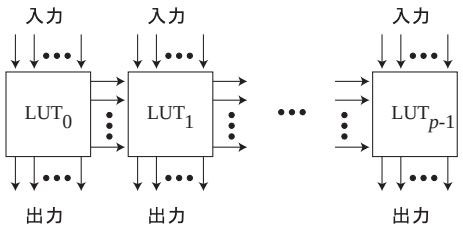

図 3.1　LUT カスケード

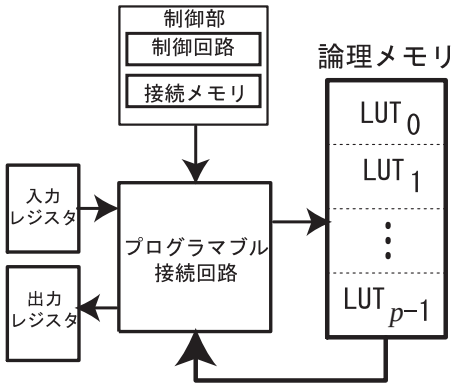

図 3.2　LUT リング

メモリのアドレス入力に加える．この操作を LUT カスケードにおける LUT の段数分繰り返すことで擬似的に LUT カスケードを実現する．

実現すべき多出力論理関数に対して，その多出力論理関数の特性関数を表す決定グラフ (Decision Diagrams for Characteristic Functions) を作成し, さらにそれを LUT カスケードや LUT リングの LUT に変換する．与えられた多出力関数に対して, LUT に格納すべき情報と LUT 間の配線を表す情報を生成する論理合成ツールが開発されている [8], [9].

本稿では PLC を実現するアーキテクチャとして LUT リングを用いるが，前述の論理合成ツールは使用せずに, ラダー図から直接 LUT にマッピングする専用ツールを開発した．

3.1　LUT リングの動作

LUT リングでは，複数の LUT を 1 個のメモリで実現する．データ出力をアドレス入力にフィードバックし，LUT の個数だけメモリ参照を繰り返すことで多出力論理関数を計算する．

図 3.3(a) のように 3 段の LUT カスケードで実現された 4 入力 4 出力の多出力論理関数をアドレス入力 4 ビット，データ出力 2 ビットのメモリを用いた LUT リングで実現する．

アドレスの 4 ビットのうち上位 2 ビットは，LUT を指定するために用いる．時刻毎の実行の流れを以下に示す．LUT_0, LUT_1, LUT_2 は, それぞれページ 0, ページ 1, ページ 2 に格納されているものとする．

時刻 0　LUT_0(ページ 0) を参照するためにアドレスの上位 2 ビットに，$(A_3, A_2) = (0, 0)$ の値を入れ，さらに下位 2 ビットに b_0 と a_0 の値を入れる．このアドレスでメモリを参照し, 得られた出力を y_1，y_0 とする．y_1 は出力 f_0 として出力レジスタに保存し, y_0 は次ページへのフィードバックとして用いる．

時刻 1　LUT_1(ページ 1) を参照するためにアドレスの上位 2 ビットに $(A_3, A_2) = (0, 1)$ の値を入れ，さらに下位 2 ビットに，前ページからのフィードバック y_0，及び，入力 b_1 を入れる．得られた出力 y_3 は出力 f_1 として出力レジスタに保存し,

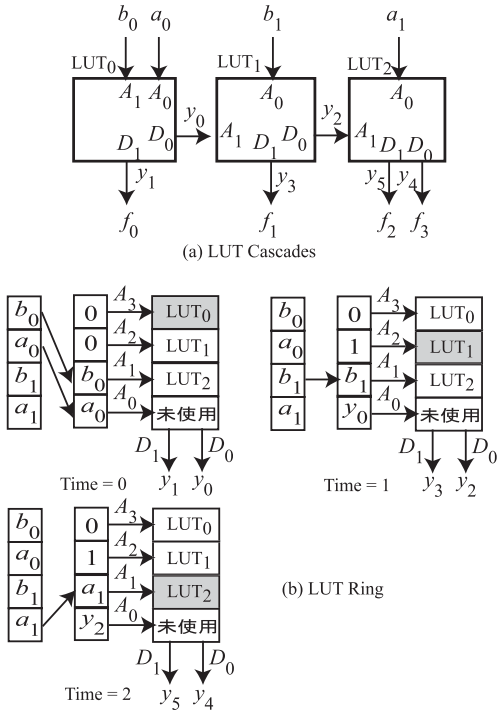

図 3.3　LUT リングの動作

y_2 は次 Page へのフィードバックとして用いる．

時刻 2　LUT_2(ページ 2) を参照するためにアドレスの上位 2 ビットに $(A_3, A_2) = (1, 0)$ の値を入れ，さらに下位 2 ビットに，前ページからのフィードバック y_2，及び，入力 a_1 を入れる．出力 y_5 を外部出力 f_2 に y_4 を外部出力 f_3 とする．

4. プログラマブル・ロジック・コントローラを模擬する LUT リング

第 2 章で述べたように, PLC はラダー図を図 2.3 に従い上から下に向って順に 1 行づつ解釈・実行する．PLC の速度は, スキャンタイムで表される．本研究では, PLC の動作を高速に模擬する LUT リングを開発する．本章では，LUT リング・アーキテクチャ，及び，LUT リング内のメモリへの実装方法を提案する．

ラダー図の各行が表現する論理関数を予め計算しておき, それを真理値表として LUT に格納する．図 4.1 に図 2.2 のラダー図の各行を各々の LUT で実現し, 合計 3 個の LUT のカスケードで実現した例を示す．LUT_0 は 1 入力 1 出力論理関数を, LUT_1 と LUT_2 は 2 入力 1 出力論理関数を実現する．LUT リングでは, これら 3 つの LUT を一つのメモリ（論理関数メモリ: memory for logic) で実現する．PLC を構成するために, LUT リングに, 内部状態を記憶する M レジスタ, 外部出力を記憶するためのレジスタとしてダブルランクのレジスタ (YA レジスタ, 及び, YB レジスタ) を追加する．[9] で述べた LUT リングでは接続回路網にはシフタを使用した．これは, 外部入力は全ての LUT 中で 1 回しか利用しないという制限を持つ論理合成法を使用するからである．しかし, PLC の場合, メモリアクセス時に M レジスタ, Y レジスタ, 入力を重複して使用する．そのため, 入力レジスタ, M レジスタ, 及び, YA レジスタの出力を論理メモリの任意のアドレスに接続する機能を追加する．また, 論理メモリの出力値を任意のレジスタに格納するための機能を追加する．

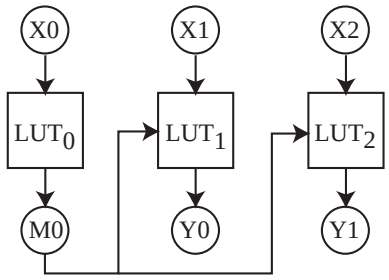

図 4.1　ラダー図から生成した LUT カスケード

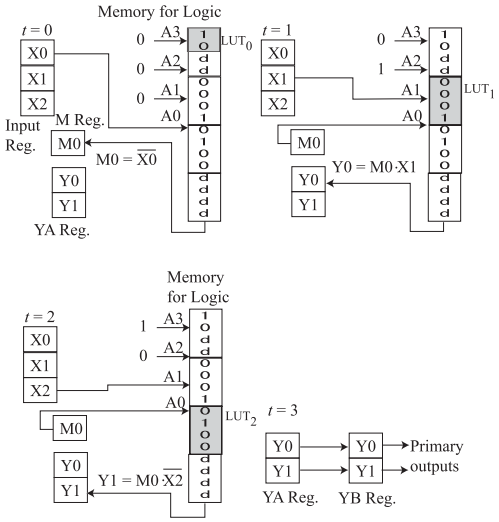

図 4.2　LUT リングによるラダープログラムの実行

以下に図 4.1 を LUT リングで PLC を模擬する方法を示す（図 4.2 参照）：

時刻 0　$M0 = \overline{X0}$ を表す真理値表は, 論理メモリの 0∼1 番地に格納されている．格納場所を指定するために, $X0$ をアドレス線 A0 に接続し, 上位 3 ビットを 0 とする．　つまり, メモリアドレスに入力 $(A3, A2, A1, A0) = (0, 0, 0, X0)$ を加える．メモリから読み出した結果を M レジスタの M0 に格納する．

時刻 1　$Y0 = M0 \cdot X1$ を表す真理値表は, 論理メモリの 4∼7 番地に格納されているので $(A3, A2, A1, A0) = (0, 1, X1, M0)$ をメモリアドレスに入力し, $Y0$ を求める．

時刻 2　$Y1 = M0 \cdot \overline{X2}$ を表す真理値表は, 論理メモリの 8∼11 番地に格納されているので $(A3, A2, A1, A0) = (1, 0, X2, M0)$ をメモリアドレスに入力し, $Y1$ を求める．

時刻 3　YA レジスタの値を一斉に YB レジスタに転送し, 出力を一斉に更新する．

4.1　PLC の高速化

図 2.2 のラダー図において, 第 1 行目で, M0 の値が更新された後にのみ第 2 行目と第 3 行目の M0 の値を使った計算が可能である．M0 の更新後, Y0 は M0 と X1 のみに, Y1 は M0 と X2 のみに依存しているので, Y0 と Y1 とは並列に計算できる．並列に計算可能な行をラダー図から検出し高速化を図るには, 変数の依存関係を調べればよい．ここでは, コンパイラなどで用いるタスクグラフを使用する．タスクグラフの例を図 4.3 に示す．この図は, “ラダー図の第 4 行目の計算は, 第 3 行目と第 2 行目の結果に依存し, 第 3 行目の計算は第 1 行目の結果に依存する” ということを示す．

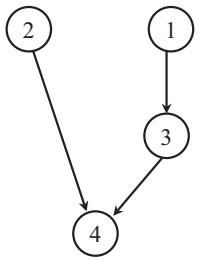

図 4.3　タスクグラフの例

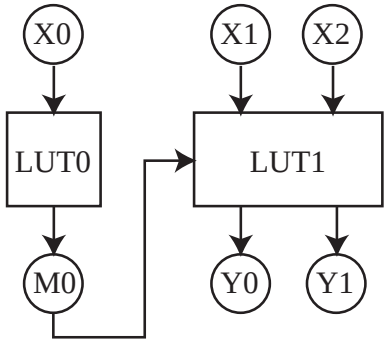

図 4.4　図 4.1 の LUT1 と LUT2 とを併合した LUT カスケード

並列に計算可能な論理関数を一つの LUT で多出力論理関数として実現する．複数出力を同時に計算するので，メモリ・アクセス数を削減でき，スキャンタイムを短縮できる．

図 4.1 と 4.2 の LUT_1 と LUT_2 とは，並列に計算可能な論理関数を表すので，これらを併合できる．併合し，段数を 3 段から 2 段に減らした LUT カスケードを図 4.4 に示す．これに対応する LUT リングを図 4.5 に示す．図 4.5 では，$t = 1$ の時に，$Y0$ と $Y1$ とを同時に計算する．

［定義 4.1］　ラダー図の q 個の行を同時に表現するために必要な真理値表の大きさ N は $N = 2^r \times q[\mathrm{bits}]$ で表される．ただし，r はこれら q 行に出現する異なる変数の個数とする．ただし，ラダー図の各行の右端にある丸で囲まれた出力変数は，r の計算では使用しない．

［アルゴリズム 4.1］　(ラダー図の並列化) ラダー図は第 1 行目から第 u 行目までとする．各行にレベル (level) 付けを行うため，第 i 行目のレベルを $level[i]$ と表記する．

```
for(i = 1; i++; i<=u) level[i] = 0;
for(i = 2; i++; i<=u) {
  for(j = 1; j++; j<i) {
    if(第 i 行目に使用している変数が第 j 行目の出力
      かつ level[i] < level[j]+1)
      level[i] = level[j] + 1;
  }
}
for(i = 1; i++; i <= u){
  level[i] が同じものを一つのグループにまとめ，
  それを一つの LUT で実現する．
}
```

■

並列実行可能な行の検出には，タスクグラフを使用するが，ここではなるべく上の行から下に向かって併合可能なものはできるだけ早期に実行するという As soon as possible 方策で並列化を行う．　アルゴリズム 4.1 は，ラダー図の並列化可能な行を検出する方法を示す．本アルゴリズムを適用したラダー図の例を図 4.6 に示す．この場合，第 1 行目と第 2 行目を並列に計算できることがわかる．

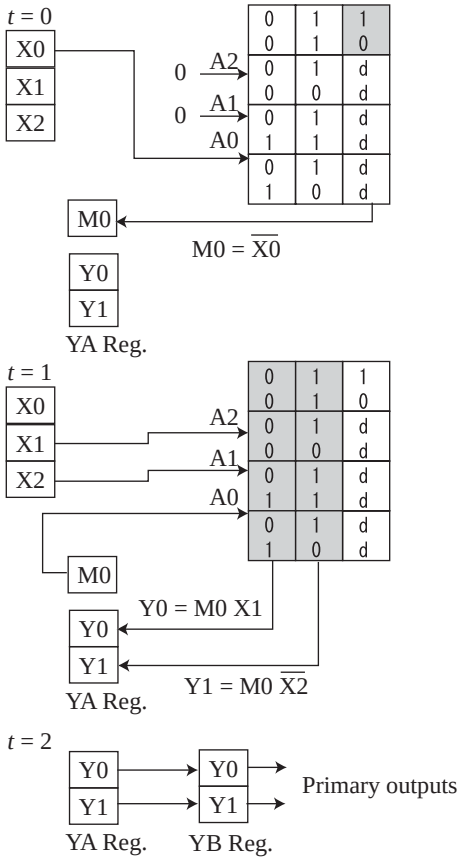

図 4.5　図 4.2 の LUT1 と LUT2 とを併合した LUT リングの動作

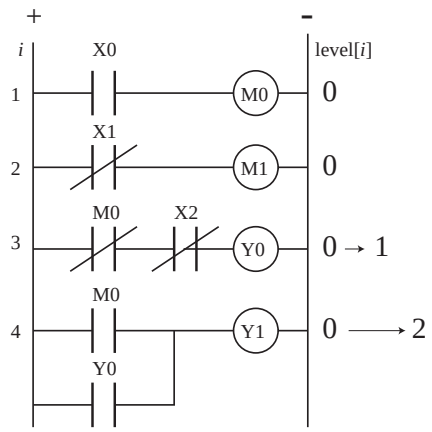

図 4.6　ラダー図と並列実行可能な行の検出例

アルゴリズム 4.1 で並列実行可能な行を併合すると，結果が実行順序に依存しない場合は演算順序が入れ替わる場合がある．図 4.7(a) のラダー図を並列化しないで LUT リングで実行すると，論理メモリの参照が 4 回必要である．これに本アルゴリズムを適用すると第 1 行目と第 3 行目，第 2 行目と第 4 行目とをそれぞれ併合できることがわかる．従って，並列化すると，論理メモリの参照が半分の 2 回ですむ．本論文では，ここまでの並

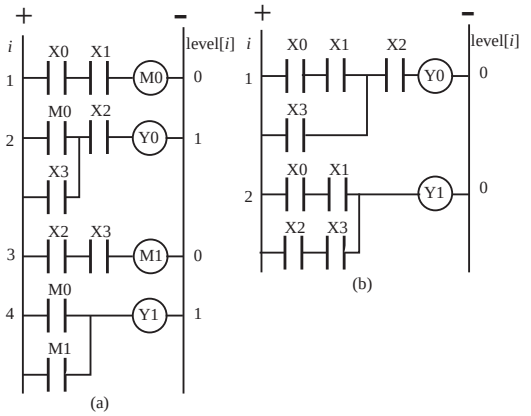

図 4.7　アルゴリズム 4.1 の効果と限界

列化は行っている.

ここで, M0, M1 は, 状態変数ではあるが, プログラム言語では一時的に使用される中間変数にあたることに注目する. これを利用し, M0 と M1 を参照している第 2 行目と第 4 行目に代入すると図 4.7(b) のラダー図を得る. 更に, これにアルゴリズム 4.1 を適用すると第 1 行目と第 2 行目とは並列に実行可能なことがわかり, 1 回の論理メモリの参照だけですむことがわかる.　しかし, 本稿ではこのような状態変数を代入は, 現時点では取り扱っていない. これは, 中間変数の代入という方法でも解決できるが, 本来は論理関数そのものを考慮することでも解決できる.

5.　LUT リングとその性能評価

5.1　PLC を模擬する LUT リング

PLC を模擬する LUT リングを設計した. 三菱電機マイコン機器ソフトウェア (株) 製の FPGA(Field Programmable Gate Array) ボード MU200-AP40 と SRAM ボード MU200-XSR 上に実現した. 使用した FPGA は, Altera 社の APEX20 シリーズ, EP20K400EBC652-3 である. LUT リングの論理メモリには, SRAM ボード (512K×24[bits]) 上のうち 512K×16[bits] を使用した. 接続メモリには FPGA 内部の組込みメモリを用いた. LUT リングのブロック図を図 5.1 に示す.

ダイレクト・アクセス・セレクタ (Direct Access Selecter) は, 論理メモリに真理値表を外部から直接格納するための回路である. 同様に接続メモリも外部から直接, 値を書き込む回路も設ける. mode 入力により, 論理メモリへの書込み, 接続メモリへの書込み, PLC 動作モードの切替を行う. また, これらのメモリに書き込むデータは, パーソナルコンピュータより RS-232C インターフェースを用いてダウンロードする. FPGA には, パーソナルコンピュータと通信するための回路も一部実現されている.

外部入力レジスタ (Input Reg.), 状態変数レジスタ (M Reg.), 出力変数レジスタ (YA Reg.), タイマ／カウンタからの出力のうち必要な信号を入力／フィードバック・セレクによって選択し論理メモリのアドレス入力に加える. どの入力を選択するかは, 接続メモリ (Memory for Interconnection) に蓄えた接続情報をセレクタのセレクト信号に加えることで行う.

LUT リングの記述は Verilog-HDL を用い, 論理合成・配置配線には Altera 社の Quartus II を使用した. ただし, 図 5.1 内部のタイマ・カウンタユニットについては今回は実装しなかった. 論理合成の結果, FPGA 内の論理回路を実現する Logic Elements は用意された LE の 26%にあたる 4483 個を使用し, 組込みメモリは 13,568-kbit で全体の 6%を使って実現できた. 開発ツールが予想した最大遅延は 21nsec で, 最大動作周波数は 47MHz であった. 本プロトタイプは, 三菱マイコン機器製のパワーメデューサ実験基板上に実現し 40MHz で動作した. なお, LUT1 個の参照には 2 クロック使用するので, ラダー図 1 行の処理に 50[nsec] 必要とする.

PLC のベンチマークに対して実験を行った結果を表 5.1 に示す. NAME はベンチマークの名前, #IN は外部入力の個数, #OUT は外部出力の個数, #STATE は状態変数の個数を示す. heatcut では, #OUT が 0 になっているが, このベンチマークは他の部分のサブルーチンになっているためで, heatcut の幾つかの状態変数の結果を他のラダー図が参照しているからである.

メモリ・アクセスの回数で実行速度を比較することにする. #N0 は, 通常の MPU で実現した場合のシーケンス命令の個数を示す. 提案した LUT リング型 PLC で, 1 度にラダー図の 1 行分だけの出力を求める場合のメモリアクセス回数を#N1, 複数行分の出力を同時に求める場合のアクセス回数を#N2 で示す. また, この場合に必要な論理メモリの容量をそれぞれ#M1, #M2 で表す. 例えば, laundry では, MPU を用いると 277 個のシーケンス命令を 1 スキャン中に行う必要がある.　一方, LUT リング型 PLC では, ラダー図の行数が 57 であり, メモリアクセス数も 57 回となる. 出力の評価値を並列化するとメモリアクセスは 20 回にまで削減できる. 参考のために 1 個のシーケンス命令が 16 ビットで表されていると仮定し, それを格納するのに必要なメモリ容量を#M0 で表す. また, この実験ではベンチマークの規模が比較的小さいのでメモリ容量は少ない.

同様に, MCNC ベンチマーク [5] に対して実験を行った結果を表 5.2 に示す.　ここで用いた MCNC ベンチマークは, PLA(Programmable Logic Array) 等での実現に適した組合せ論理関数を選んだ. この関数を PLC で実現することにし,　ラダー図に変換して実験を行った. PLC のベンチマークと異なるのは, 出力や状態変数出力を求めるのに必要なリテラル数が多いこと, 一つの出力が他の出力の入力になっている場合が少ないことが特徴である. また, 本ベンチマークは多段の組合せ回路であり, 中間変数の記憶を LUT リングの状態変数に格納することで計算を行う.

どのベンチマークに対しても $\#N0 \geqq \#N1 \geqq \#N2$ となっている. ラダー図に出現する命令の多くは論理演算命令である. メモリ参照の個数が少なければ, 1 スキャンタイムも短縮できると仮定し, さらにメモリアクセスは, MPU も LUT リングも同程度のクロック周波数で動いていると仮定するとメモリ参照の個数が少ないほど高速になる.

表 5.1 では, #N1 は#N0 の 20%〜26%, #N2 は#N0 の 3%〜13%となっている.　一方, 表 5.2 では, #N1 は#N0 の 4%〜19%, #N2 は#N0 の 1%〜8%となっている.

表 5.2 の#N1 の#N0 に対する割合が, 表 5.1 の場合に比べ小さい. 表 5.2 の関数をラダー図で表現した時, 各行の出力値を求めるのに必要な変数の個数が, 表 5.1 の PLC のベンチマークの場合に比べ多い. このため表 5.2 の#N0 のステップ数が多くなるためと考えられる. また, 表 5.2 で, #N2 の#N0 に対する比率も表 5.1 の場合に比べ小さい. これは、次の二つの性質からも推測できる。

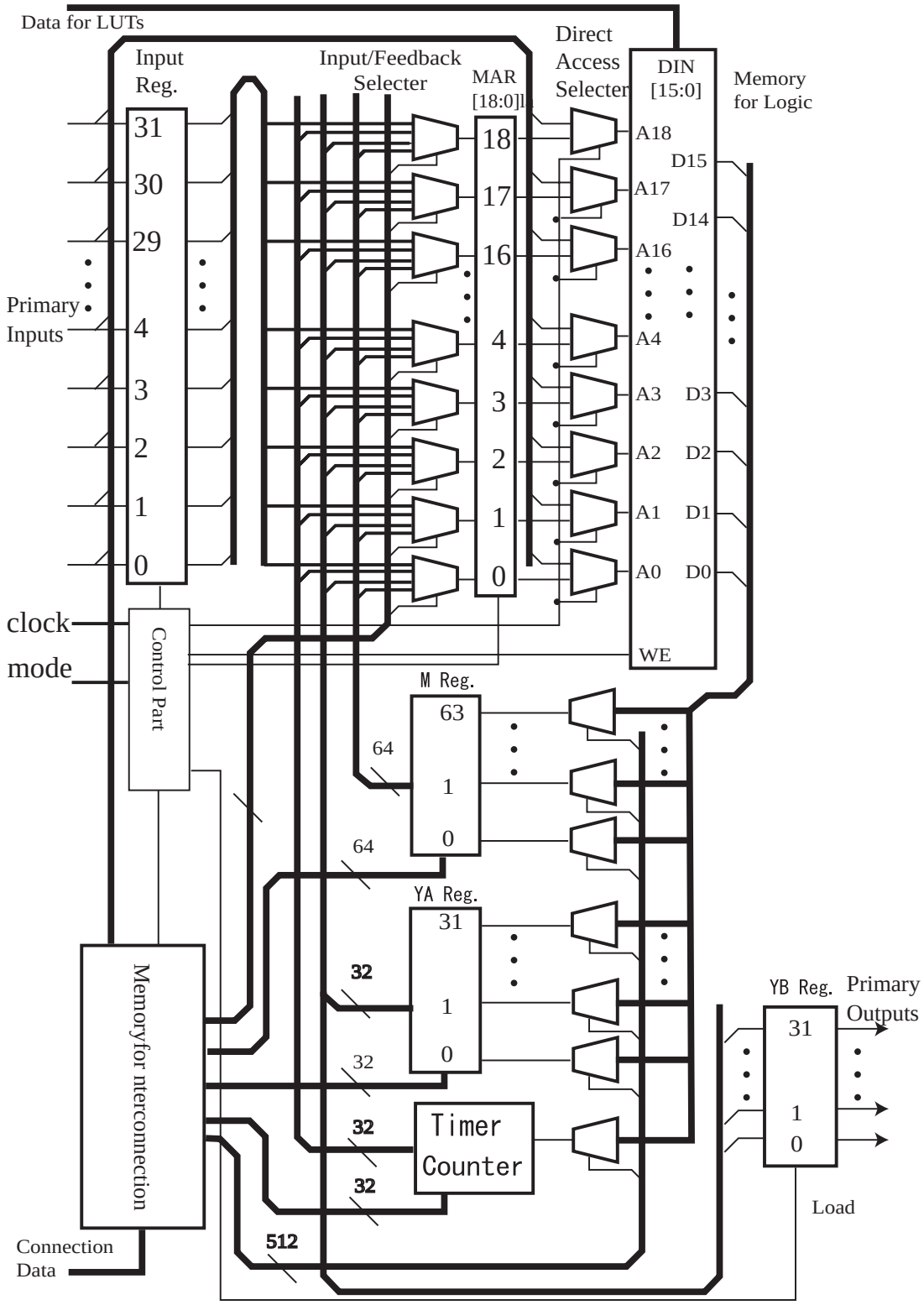

図 5.1　PLC を模擬する LUT リング

- PLC 用ベンチマークのラダー図では, ある行で求めた状態変数や出力が, 他の行で用いられることが多い.
- 一方, MCNC ベンチーマーク関数の場合はそのようなことが少なく、並列に計算可能な場合が多い。

シーケンス命令では, 情報の読み込み, 書き出し, 演算は, 1 ビット毎にしか実行できない.　一方, LUT リングで一度にラダー図 1 行分の計算を行う時は, ラダー図の 1 行に出現する異なる変数が LUT の容量に収まると仮定すると, 1 回のメモリ参照とレジスタ書込みで実行できる. つまり, MPU での実行に比べると入力変数の評価の並列化を行っていると考えられる. 次に, ラダー図複数行分を同時に実行することを許す場合は, 多出力論理関数を計算することに相当し, この場合は入力変数の評価の並列化とともに出力評価の並列化を実施していることになる. つまり, MPU の場合は変数を一つづつ処理し, LUT(single) は 1 行づつ処理し, LUT(multi) は複数行を処理しているといえる.

表 5.1　PLC ベンチマークに対するメモリアクセス数とメモリ容量比較

				steps			memory[bits]		
				MPU	Ring		MPU	Ring	
					single	multi		single	multi
NAME	#IN	#OUT	#STATE	#N0	#N1	#N2	#M0	#M1	#M2
camshaft	7	7	15	86	22	11	1376	294	616
heatcut	15	0	43	175	43	22	2800	472	2004
elevator	10	11	30	202	41	6	3232	320	1956
laundry	6	8	49	277	57	20	4432	2868	14898

表 5.2　MCNC ベンチマーク関数に対するメモリアクセス数とメモリ容量比較

				steps			memory[bits]		
				MPU	Ring		MPU	Ring	
					single	multi		single	multi
NAME	#IN	#OUT	#STATE	#N0	#N1	#N2	#M0	#M1	#M2
b12	15	9	0	257	9	2	4112	2192	69632
c8	28	18	30	515	48	12	8240	1446	340064
cc	21	20	13	172	33	5	2752	410	401440
cm163a	16	5	11	106	16	4	1696	140	6208
comp	32	3	52	311	55	11	4976	900	308772
frg1	28	3	0	912	37	36	14592	533552	557088
misex2	25	18	0	228	21	8	3648	56492	196608
pcler8	27	17	7	150	24	12	2400	552	180768
vg2	25	8	0	915	46	45	14680	598904	729104

例えば 4 入力論理積 (AND) を実現するためには, MPU では, ロード命令を 1 回, AND 命令を 3 回, OUT 命令を 1 回実行しなければならない. 一方, LUT リング (single) では 4 入力全ての値を一度にメモリに加えることで 1 回のメモリ参照とレジスタ書込みで実行できる. 更に, LUT リング (multi) では複数の出力を同時に求めることができるので高速化が達成できた.

6. ま と め

LUT リングを用いてプログラマブル・ロジック・コントローラを構成する方法を提案した. また, ラダー図より LUT リング用のコンフィギュレーションデータを作成するコンパイラを作成した. 最後に実験により本方法が従来の MPU を元にした構成法より高速であることを示した. 今後の課題としては, より多くの PLC 用ベンチマークに対する評価を行うことである.

謝辞

本研究は一部日本学術振興会 科学研究費補助金および文部科学省・北九州知的クラスタ創成事業の補助金による.

文　献

[1] W. Bolton, *Programmable Logic Controllers, 3rd Edition* Newnes, 2003.

[2] 井口, 笹尾, “LUT カスケード・アーキテクチャについて,” 平成 15 年電気学会 電子・情報・システム部門大会, MC2-4、2003 年 8 月 29 日, 秋田大学.

[3] W. H. Kwon, J. Kim, and J. Park, “Architecture of a ladder solving processor(LSP) for programmable controllers,” *Proc. IEEE IECON'91*, 1991.

[4] H. S. Kim, D. S. Kim, N. Chang, and W. H. Kwon, “A translation method of ladder diagram on PLC with application to a manufacturing process,” *IEEE Conference on Robotics and Automation '99*, Detroit, USA, May, 1999.

[5] MCNC-benchmark function set: http://www.cbl.ncsu.edu/

[6] H. Qin, T. Sasao, M. Matsuura, K. Nakamura S. Nagayama and Y. Iguchi ”A realization of multiple-output functions by a sequential look-up table cascade,” *IEICE Transactions on Fundamentals of Electronics, Vol.E87-A, (accepted for publication).*

[7] T. Sasao, M. Matsuura, and Y. Iguchi, “A cascade realization of multiple-output function for reconfigurable hardware,” *Int'l Workshop on Logic Synthesis (IWLS-2001)*, Lake Tahoe, Ca, U.S.A., pp. 225–300, June 12-15, 2001.

[8] T. Sasao, M. Kusano, and M. Matsuura, “Optimization methods in look-up table rings,” *Int'l Workshop on Logic and Synthesis (IWLS-2004)*, Temecula, Ca, U.S.A., pp. 431–437, June 2-4, 2004.

[9] T. Sasao, H. Nakahara, M. Matsuura and Y. Iguchi, “Realization of sequential circuits by look-up table ring,” The 2004 IEEE International Midwest Symposium on Circuits and Systems (MWSCAS 2004), Hiroshima, July 25-28, 2004, pp. I:517-I:520.

[10] 関口隆, 高橋浩, 青木正夫, 下川勝千, 薦田憲久, シーケンス制御工学 —新しい理論と設計法—, 電気学会, 1988.

[11] J. W. Webb and R. A. Reis, *PROGRAMMABLE LOGIC CONTROLLERS, Principles and Applications,* Pearson Education, 2003.

[12] 山口大介, 松永祐介, “プログラマブルコントローラ向けプロセッサ・アーキテクチャの検討と評価,” 信学会技術研究報告, CPSY2002-108, pp.19–24, March 2003.

[13] 山口大介, 勝木裕二, 松永祐介, “プログラマブルコントローラ向けプロセッサ・アーキテクチャの評価,” 情報処理学会研究報告, 2004-ARC-157, pp.91-96, Mar. 2004.