4H-2

CMOSゲートアレイの論理設計支援システム

石曽根　信　藤田　昌宏　田中　英彦　元岡　達

(東京大学　工学部)

1. はじめに

ハードウェアの記述より、CMOSゲートアレイ用の回路を自動的に合成する試みを、Prolog を処理系に用いて行なっている。従来のCADツールは、ユニットセル・レベルの設計からレイアウト・配線するためのものであった。この研究は、ハードウェアの仕様記述から直接ゲートアレイ用の自動設計をしようというもので、現在はハードウェア記述言語としてDDL[1]を用いている。

2. 処理の流れ

2.1 概要

ゲートアレイを開発する際には、ユーザはユニットセル・ファミリと呼ばれる基本的な論理素子を使用して回路を設計し、メーカに提出する。本研究はDDLトランスレータ[2]の出力からユニットセル・レベルの回路を合成することを目標としている。

DDLは状態遷移に基づくレジスタ・トランスファレベルのハードウェア記述言語の1つである。DDLトランスレータは、DDLの記述を受け取り、レジスタ、ターミナルの転送表、状態の遷移表などを出力する。本研究では、これを一度“マクロユニット”という仮想的なユニットに展開する。マクロユニットとは、簡単に言うと、①入力数無制限、②ファンアウト無限大、③Nビット入力－Nビット出力、という論理素子で、ユニットセルより高度なレジスタや加減算などのユニットを含むものである。このマクロユニットのレベルで種々の簡単化や、ゲート数や遅延時間の概略的な解析を行なう。この方式の利点として、①ユニットセルに比べてデータ量が少ないため扱いやすい、②機能の抽象度が高いため、ある程度ではあるがテクノロジー独立になる、などがあげられる。最後にマクロユニットをユニットセルに展開する。簡単な例で、レジスタRへの転送部分がどのように展開されていくかを図1に示す。

2.2 処理フロー

上で述べた処理を行なうにあたって、我々は処理段階を6つのフェーズに分けた(表1)。Phase1は、図の(b)で示したようなレジスタ転送表などを読んでProlog の記述に変換するものである。例えば、図1の(b)は、

```
register_trans([R,[O,O]],
    [[O,O],[[A],[CO1]],[[B],[CO2]]]).
```

と変換される。

Phase2はマクロユニットへの展開で、第1次簡単化を行なう。展開結果は and、or、registerなどの一般的な機能を使用しており、特定のテクノロジーに依存していない。

Phase3はマクロユニットの簡単化である。簡単化の詳細は2．3で述べる。

Phase4では、ゲート数、及び遅延時間の解析を行なう。ただしゲート数は、Phase5の展開時にも増減があり、遅延時間も正確にはチップ上にレイアウトしなければ分からないので、メーカの資料[3]より概略値を計算する。以上の解析で不都合な点があれば、さらに簡単化や高速化を行なう。

Phase5では、マクロユニットをユニットセルに展開する。この時簡単化、ファンアウトの制限からくるバッファの挿入も合せて行なう。

Phase6は、展開データをシミュレータなどのツールにかけるためのデータ変換である。

2.3 簡単化について

各フェーズでの簡単化について以下にまとめる。第1次簡単化は、論理式レベルでの簡単化であり、互いに似た論理式の共通部分を取り出してまとめる(図2(a))。

第2次簡単化は、重複したユニットを削除し(図2(b))、その後ルールによる置換を行なって最適化を行なう(図2(c))。ここで、CMOSゲートアレイの特徴に従った簡単化を行なう。

第3次簡単化は、ユニットセルに展開する時の簡単化で、例えば定数を加える加算器などはこの時に最適化される。

このように論理式レベル、マクロユニットレベル、ユニットセルレベルでそれぞれ簡単化を行なう。

3. 実行例

我々は例題として、高並列推論エンジンPIE[4]のユニファイ・プロセッサ(UP)をDDLで記述したもの(ソースで約1000行)を使用し、これを変換する試みを行なっている。現在PIEに実装されているUPは人手で設計されたもので、TTL・IC約500個程度のものである。

現在、Phase1及びPhase2は終了しており、Phase3もほぼ完成している。実行結果を表2に示す。Phase2での展開結果は、(基本)ゲート数約27000、Phase3の簡単化により約20000まで減少した。なお、全ての

プログラムは、Edinburgh大学で開発されたUNIX上のC－Prologを用いて書かれている。

4. おわりに

ハードウェアの記述より、CMOSゲートアレイの回路設計を自動的に行なう手法、及びその際の簡単化などについて述べた。十分実用的なシステムになる見通しもついている。今後、ゲート数が多くなった時の論理分割をどの段階で行なうか、などが重要となってくるだろう。

参考文献

[1] Duley, J.R. et al. "A Digital System Design Language (DDL)",IEEE Trans. Comput., vol.C-17,no.9,1968.
[2] Duley, J.R. et al. "Translation of DDL Digital System Specification to Boolean Equation",IEEE Trans. Comput., vol.C-18,no.4,1968.
[3] "CMOS Gate Array Design Guide", Fujitsu limited,Edition 1.0,December 1983.
[4] Moto-oka, T. et al. "The Architecture of a Parallel Inference Engine -PIE-", Proc. of FGCS,1984.

Phase１：DDLトランスレータの出力結果をPrologの記述に変換
Phase２：マクロユニットへの展開、第１次簡単化、クロス・リファレンス
Phase３：第２次簡単化
Phase４：ゲータ数・遅延時間解析とそれにともなう設計変更
Phase５：ユニットセルに展開、第３次簡単化
Phase６：展開結果のシミュレータなどへのデータ変換

表１　処理のフェーズ

	プログラム	実行時間
Phase1	900行	70分
2	900	10
3	600	80

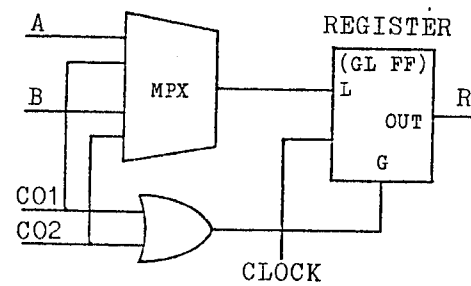
（時間はVAX11／780のCPUTIME相当）
表２　UPに対する実行結果　（最終結果ではない）

```
        .
        .
|* CO1 *|   R <- A.,
|* CO2 *|   R <- B.,
        .
        .
```

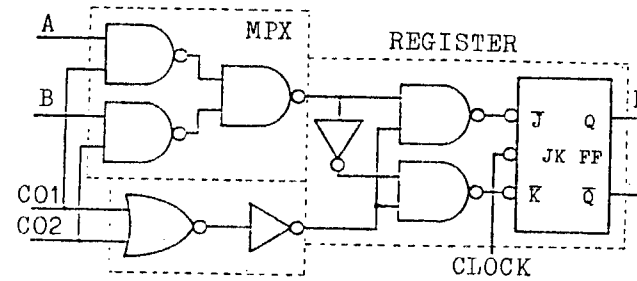
（a）DDLによる記述

R (0:0) 1 BIT REGISTER		
SINK-RANGE	SOURCE	TRANSFER CONDITION
(0:0)	A	CO1
	B	CO2

（b）DDLトランスレータ出力



（c）マクロユニットへの展開

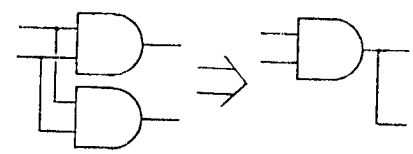


（d）ユニットセルへの展開

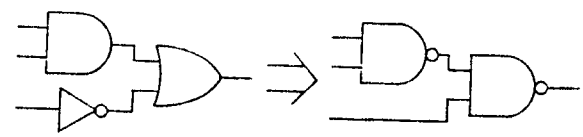
図１　処理の概要

```
T1 = A & B & C & D                COM = A & B & C
T2 = A & B & C & E        ⇒        T1 = COM & D
                                   T2 = COM & E
```

（a）共通部分をまとめる（&はAND）



（b）重複するゲートの削除



（c）ルールによる置換

図２　簡単化