設 計 自 動 化 84－1
（1997．5．23）

セレクタを効率的に利用した論理回路合成手法

鳥居 隆史　　浅田 邦博
東京大学工学部浅田研究室（VLSI センター）

従来の論理合成手法はあらゆる論理、ゲート・ライブラリに対応するために人手設計に劣る場合がある。本研究ではセレクタを利用することに注目して従来の論理合成手法では見落としてしまう最適解を求める。セレクタの利用のために論理の対称性を利用する。論理から対称性を抽出する方法、回路から抽出する方法、論理分割を行なってから対称性を抽出する方法の３つの方法を開発した。これらによって15％の論理で従来の論理合成手法より回路規模の縮小ができた。さらに論理の特性とセレクタの利用効率の関係から計算時間の短縮ができることを示した。また、マルチプレクサ・ベースのFPGA合成ツールとの比較を行ない、本研究の手法が効果があることを示した。

A Logic Synthesis Method Utilizing Selectors

Takashi Torii and Kunihiro Asada
Faculty of Eng. (VDEC), Univ. of Tokyo

We have studied a new logic synthesis algorithm by utilizing selector circuits to improve the weak point that the conventional logic synthesis algorithms synthesize using all the logics mapped to a lot of gate libraries, so that in some cases their performance is worse than human design. We have proposed three logic synthesis methods that use symmetry of logic, circuit and decomposed diagram to utilize selectors. Using these methods together, we have demostrated that about 15% of logic benchmarks are better than the conventional method, and synthesis time is minimized by efficiency of using selectors depending on the characteristics of logic. We have also shown the present method is effective compared with FPGA synthesis tools targeted to multiplexor-based designs.

1　はじめに

論理合成はすでに実用化されているが、人手設計と比べると劣る場合があるという弱点がある。この原因として２つがあげられる。１つめは論理合成がヒューリスティックな解法を用いていることである。論理合成問題はNP問題であるため、計算時間の制約から全探索は不可能であり、ヒューリスティックを用いざるを得ない。２つめは論理合成がテクノロジに応じた最適化をしていないことである。従来の論理合成はどのような論理、どのようなテクノロジ（ゲート・ライブラリ）に対しても同じ解法を用いることを目的としている。そのために、図1のようにテクノロジ非依存の合成とテクノロジ・マッピングは分割され、フィードバックは行なわれない。これはゲート・ライブラリに応じた最適化はなされないことを意味する。

本研究ではこれらの論理合成の弱点の解決方法を提案する。従来の論理合成が最大公約的な「汎用品」であるのに対し、「特定用途向き」の手法であり、従来手法の盲点となるような最適化ができる。ただしあらゆる場合に対応してはいないので、この特定用途向き

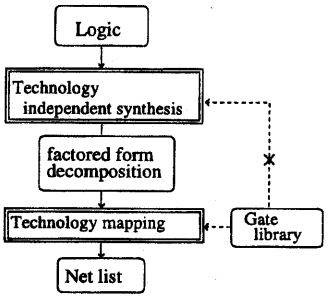

図 1: テクノロジ非依存の合成ではライブラリは考慮されない

の手法を従来手法と併用してよい方を選択する（図 2）。

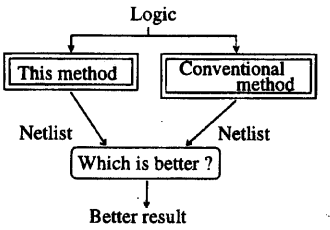

図 2: 従来手法と併用して良い方を解とする

本研究では特定用途としてセレクタの利用を考える。セレクタに注目する理由を 3 つあげる。

・セレクタは論理機能に対して面積が小さい。
図 3 のようにセレクタは CMOS パストランジスタロジックでトランジスタ 6 個で構成することができ、実現する論理に対して面積的に有利なゲートであるといえる。

・従来手法では有効利用されていない。
人手設計の特に性能の高い回路ではセレクタが使われることが多い。それに対して従来の論理合成ではセレクタはテクノロジ非依存の合成の過程ではを考慮していないため、セレクタを有効に利用しているとはいえない。

・人間の論理解法 [動的な論理変化] との類似
セレクタの論理は人間が実際に論理を解く時に用いる思考と類似している点が多いと考えられる。そのため、人間の論理解法に則した手法によってセレクタの有効利用ができると考えられる。

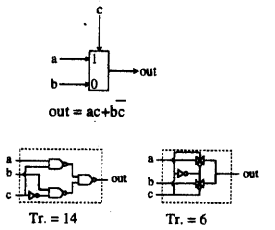

図 3: セレクタの回路構成

2 本研究で用いた回路合成手法

セレクタのコントロール信号があり、コントロール信号によって g_1 と g_2 に出力がわかれる場合、図 4 のように出力セレクタを使用して回路が構成できる。ここで、g_1、g_2 は異なる論理だが、入力セレクタを用いれば同じ論理になる場合、図 4 右のように入力セレクタを使用して g_1、g_2 をドラスティックに圧縮することが可能である。

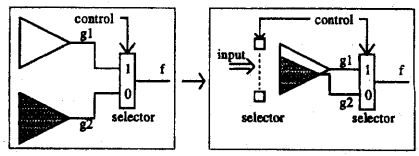

図 4: セレクタを用いた回路構成

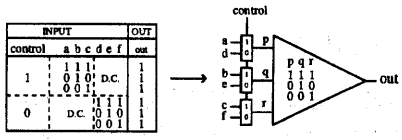

図 5: 入力の入れ換えによって同じになる論理をセレクタを用いて構成

図 5 ではコントロール信号によって区別される論理が入力 (a,b,c) と (d,e,f) を入れ換えることによって等しくなるので入力部分に

セレクタをおいて回路を構成することができる。また、図6では入力 (a,b,c) と $(\overline{b},c,d)$ を

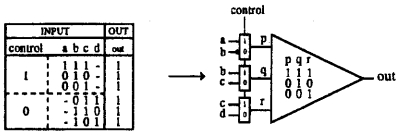

図 6: 入力の反転と入れ換えによって同じになる論理をセレクタを用いて構成

入れ換えれば論理が等しくなるので、入力セレクタとインバータをおいて回路を構成している。このように、入力セレクタを用いれば同じ論理になるような2つの論理を**対称性がある**と定義する。以降で対称性を利用してセレクタを有効利用する手法を3つ述べる。

2.1 論理からの対称性の抽出

コントロール信号を1つの入力に限定し、それによって分けられる論理の対称性を真理値表からチェックする。

1出力関数 $f(x_0,x_1,\cdots,x_{n-1})$ を考える。今回はコントロール信号として入力 x_i の場合のみを考えるため、

$$g_1 = f_x,\ g_2 = f_{\overline{x}}$$

となる。g_1 が入力の並べ変えと反転によって g_2 と同一になるかをチェックする。全ての並べ変えと反転についてチェックを行なうのは計算時間の面で実行不可能であるため、以下の点で工夫を行なった。

フィルタによる計算時間の短縮

Variable Inclusion Count[2] の利用

SYMMETRY_CHECK(図7) は行列 a_1,a_2 の対称性を検出するアルゴリズムである。

2.2 回路からの対称性の抽出

2つめとして回路を実際に合成してから回路の対称性を抽出する方法を行なった。セレクタを有効に利用する方法として、コントロール信号によって論理動作が違うような場合が考えられる。例えば図8のように一部だけがNANDゲートとNORゲートという違いがある場合、セレクタを用いて図8右のように構成することができ、劇的に回路面積が減ることが考えられる。論理をある一つの入力によってシャノン展開し、f_x,$f_{\overline{x}}$ を別々に論理合成しテクノロジ・マッピングを行なう。すると、論理 f は図9のようにセレクタを用いて実現できる。この状態から図10のようなルールを用いて回路の圧縮を行なう。 この方法のアルゴリズムを図11に示す。

```
SYMMETRY_CHECK(a1(m1, n1), a2(m2, n2)){
  if(FILTER1(a1, a2) = O.K.){
    step = 0
    while(step < 2^n1){
      a2' =VARIABLE_INVERSE(a2, step)
      if(FILTER2(a1, a2') = O.K.)
        if(VIC_CHECK(a1, a2') = O.K.)
          if(SEEK_ALL_ORDER(a1, a2') = O.K.)
            return(O.K.)
      step ++
    }
  }
  return(FALSE)
}
```

図 7: SYMMETRY_CHECK

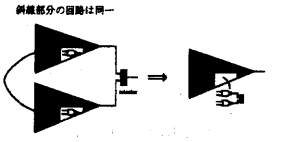

図 8: 回路合成してから対称性を抽出

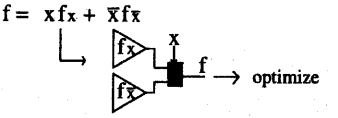

図 9: 回路合成してから対称性を抽出

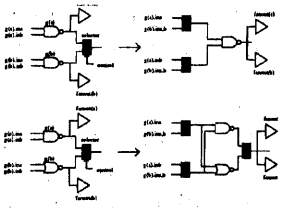

図 10: 回路圧縮ルール

```
SELECTOR_REDUCE2(network){
  repeat{
    MOVE_SELECTOR_FORWARD(network)
    CIRCUIT_REDUCE(network)
    COUNT_TRANSISTOR(network) → TABLE
  }until(no selector move forward)
}
if(TABLE ≠ NULL)
  return(MINIMUM(TABLE))
```

図 11: SELECTOR_REDUCE2 のアルゴリズム

2.3 論理分割による方法

今までの方法では論理が大きい場合には計算時間がかかる、control 信号が入力に限定されているという問題点があった。これらの問題点を解決するため、論理を小さなブロックに分割し、計算時間の短縮、入力に限定されない柔軟なコントロール信号の使用を目指す。

この方法は、対称性を先に見い出し、それを区別するコントロール信号の有無を検査する方法である。セレクタを利用できる条件は論理 g_1,g_2 に

$$\begin{cases} control = 1 \rightarrow g_2 = Don'tCare \\ control = 0 \rightarrow g_1 = Don'tCare \end{cases}$$

となる control 信号があることである。このため、分割方法には 3 つの条件がある。

1. 対称性が見出せること
2. control 信号を見出せること
3. 論理の最適構造を壊さないこと

特に 3. は面積縮小を狙うためには欠かすことのできない条件である。

以上の条件を考慮して、今回は決定木を変形したものを使用する。特徴、利点は以下のような点があげられる。

1. 各ノードに排他性が成り立つためコントロール信号を生成することが容易
2. 決定のための変数は複数の入力
3. 各ノードは多入力の論理テーブルを持つ。
4. 論理内の最大のキューブに含まれる入力で決定
5. 各ノードの論理テーブルは展開しない。

この決定木構造を Modified Dicision Diagram(MDD) と名付ける（図 12）。

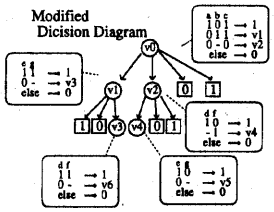

図 12: MDD による分割例

分割後、より柔軟なコントロール信号の選択のため、真理値表に変換し論理多段化と同じ形にし、セレクタによって論理を圧縮していく。セレクタによる圧縮の例を示す。図 13で

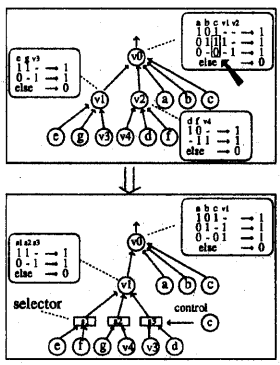

図 13: セレクタによる圧縮例

v_1 と v_2 に対称性がある。v_1,v_2 は共通の上位ノード v_0 から出ており、入力 c によって v_1,v_2 は区別され、排他性があることがわかる。そこで、v_1 の入力に対してセレクタ列をおき c

を control 信号とする。v_2 がなくなることによって上位の v_0 の論理も小さくなる。

SELECTOR_REDUCE3（図 14）はこの圧縮を行なうアルゴリズムである。v は前項で述べたように決定木の方法で分割したグラフである。圧縮を行なった後、セレクタを除いた部分は従来の論理合成ツールを利用して合成する。

```
SELECTOR_REDUCE3(v){
  table = NULL
  for all (v_i, v_j, i ≠ j){
    if(FILTER(v_i, v_j) = O.K.)
      if(SYMMETRY_CHECK(v_i, v_j) = O.K.)
        if(EXCLUSIVR_CHECK(v_i, v_j) = O.K.)
          register to table
  }
  if(table ≠ NULL){
    REDUCE(CHOICE(table))
    SELECTOR_REDUCE3(v)
  }
}
```

図 14: SELECTOR_REDECE3 のアルゴリズム

2.4 各方法の合成結果と評価

従来手法を misII の script.rugged として合成結果の比較を行なった。ベンチマークの論理は MCNC89 を使用した。単一出力の論理への適用を考えているので、全て単一出力に展開を行なった。論理の合計は 897 個である。ゲートライブラリは NAND, NOR, INV, SELECTOR、それぞれの面積は 4 : 4 : 2 : 6 で与えてある。プログラムは全て C で記述し、Sun の Sparc Station20 で実行し、計算時間の測定を行なった。

方法 1: 論理からの対称性の抽出による方法
方法 2: 回路からの対称性の抽出による方法
方法 3: 論理分割による方法
とする。3 つの方法を合わせると、トランジスタ数が従来手法よりも減った論理は 130 個 (14.5%) であった。図 15は 3 つの方法で減少した論理の重なりを示している。この図から、3 つの方法が独立した論理群に対して効果があることがわかる。図 16は sis 合成のトランジスタ数に対する比（横軸）と論理個数（縦軸）の関係である。方法 2 の減少率が大きいことがわかる。図 17は sis 合成のトランジスタ数（横軸）に対する減少する論理の割合を示している。論理規模が大きい方が減少する確率は高いことがわかる。小さい論理では従来手法で最適解が得られる場合が多いので減少する確率としては少なくなる。しかし、個数で見れば多くあり、小さい論理でも必ずしも最適解が得られてはいないといえる。本研究の方法によって従来手法の盲点をつくという目的は十分果たせたといえる。

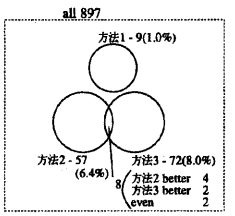

図 15: 総合結果

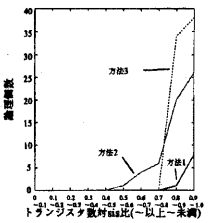

図 16: 対 sis 比別の論理個数

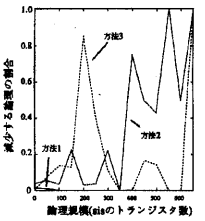

図 17: 論理規模別の減少する論理の割合

図 19から図 21はそれぞれの方法の計算時間の sis の計算時間に対する比を示している。

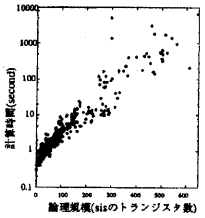

図 18: 論理規模と sis 計算時間の関係

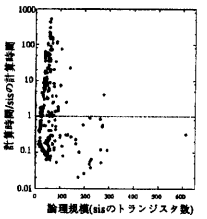

図 19: 論理規模別の計算時間対 sis 比 (方法 1)

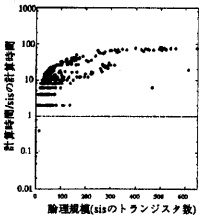

図 20: 論理規模別の計算時間対 sis 比 (方法 2)

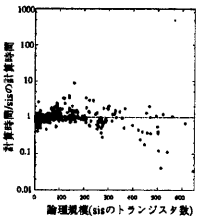

図 21: 論理規模別の計算時間対 sis 比 (方法 3)

3 論理の性質とセレクタ利用効率の関係に関する考察

論理からの情報とセレクタの利用効率との関係を利用して計算時間の短縮を狙う方法を 2 つ行なった。

3.1 コントロール入力の選択方法

どの入力をコントロール信号とするのがよいを知るために、入力 x のコントロール信号としての有効性 efficiency of$x$ を

$$\text{efficiency of } x = x f_x \overline{f_{\overline{x}}} + \overline{x} \overline{f_x} f_{\overline{x}}$$

と定義する。この式の論理規模がセレクタの利用効率と関係することが予想できる。論理規模の指標として今回は真理値表の要素 1 の数 + 要素 0 の数を使用する。この数の元の論理 f の論理規模に対する割合を横軸、トランジスタ数が減った論理の割合を縦軸にとったのが図 22である。図 23はベンチマーク論理の分布を示している。これらの図から、確率としては低いものの予想通り入力 x に依存する論理規模が大きい方がセレクタの利用効率が高いという結果が得られる。

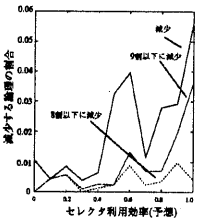

図 22: 入力の選択方法とセレクタ利用効率との相関

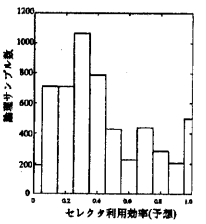

図 23: 入力の選択方法とセレクタ利用効率との相関 (サンプル数)

3.2 キューブに重なりに注目する方法

セレクタは状態空間をコントロール信号によって 2 つに分ける。この切りわけによって論理合成の最適性を崩さない場合にはセレクタが有効になる可能性が高いといえる。これを指標化するために、今回は独立なキューブ群の数に注目する。

重なっているキューブの集合を独立キューブ群と定義し、独立キューブ群が多ければセレクタによって切り分けることができる部分が多いと考え、(図 24) 独立キューブ群の数をセレクタの有効性の指標とする。図 25の横軸は独立キューブ群の数、縦軸はトランジスタ数が減った論理の割合である。図 26は横軸が独立キューブ群の数で、縦軸はベンチマークの論理数である。これらから、サンプル数は減るものの、独立キューブ群が多い方がセレクタの利用効率は高いことが示されている。

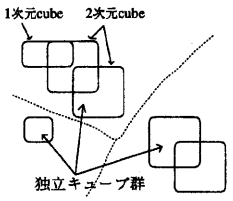

図 24: 独立キューブ群

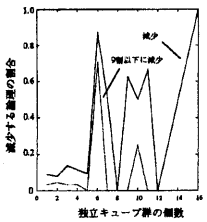

図 25: 独立キューブ群とセレクタ利用効率との相関

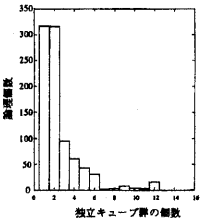

図 26: 独立キューブ群とセレクタ利用効率との相関 (サンプル数)

4 FPGA 合成ツールとの比較

FPGA の合成は初めからテクノロジを考慮した合成を行なう必要があり、本研究の発想と非常に似ている。さらに、マルチプレクサ・ベース (MB) の FPGA はセレクタをブロックとして構成するので本研究と結果の比較ができる。本章では MB の合成ツールと本研究の方法の比較を行なう。図 27は MB 構造の FPGA(

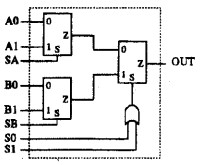

図 27: 基本ブロック

Actel's act1 architecture) の基本ブロックである。sis-1.2 ではこの構造にマッピングするコマンド act_map と比較を行なう。 [3]。

本研究との比較は回路面積 (トランジスタ数) で行なう。本研究のデータは 3 つの方法の最小値を用いる。FPGA についてはトランジスタ数に換算を行なう。合成の結果を比較すると act_map では 138 個の論理で sis よりも回路面積が縮小された。本研究では 130 個なのでほぼ同じだが、図 28のように act_map で減る論理と本研究の手法で減る論理はほとんど重複せず、セレクタを利用することを目標にしていても効果の現れ方が異なっている。図

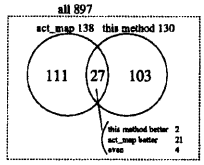

図 28: FPGA 合成コマンド act_map との比較

29から act_map のほうがやや減少率が高い論理が多いことがわかる。また、図 30、図 31から、本研究の手法では act_map に比べて大きい論理での減少が多いことが示されている。図 32は act_map の計算時間を示しており、本研究の手法と比較すると計算時間の面では act_map が優位である。しかし、 act_map でも使用できないようなセレクタの使用によって回路規模を圧縮できるため、本研究の手法の有用性が示されたといえる。

5 セレクタの利用と計算時間、論理合成手法に関する考察

act_map や本研究の手法でセレクタを利用して sis よりも小さい回路が合成できたわけだが、 sis でももっと計算時間のかかるオプションを使用すれば同等の回路が合成できるのではないかという疑問が生じる。しかし、時間をかけたとしても sis の手法ではセレクタの利用はできないのではないと我々は考える。 sis も含め、従来の論理合成手法はテクノロジ非依存の合成がテクノロジ・マッピン

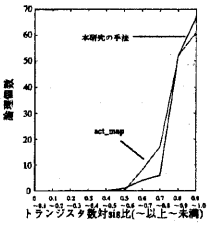

図 29: FPGA 合成コマンド act_map との比較 (対 sis 比対論理個数)

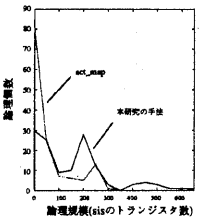

図 30: FPGA 合成コマンド act_map との比較 (論理規模対減少する個数)

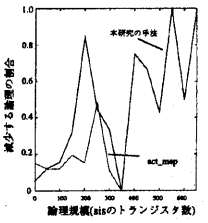

図 31: FPGA 合成コマンド act_map との比較 (論理規模対減少する個数の割合)

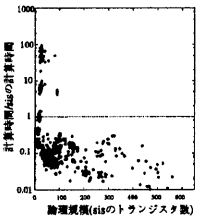

図 32: 論理規模別の act_map 計算時間対 sis 比

グと完全にわかれてしまっている。これを同じフェーズに組み入れて、マッピングからのフィードバックを行なわない限り、この弱点は解消されないのではないか。ただし、このフィードバックは膨大な計算時間を必要とすることが予想される。

自動合成の問題は従来は汎用品を求める傾向があった。これからは集積回路で ASIC が重要となってきているように、特定用途向きの自動合成が必要となるのではないか。

6 結論

- 従来の論理合成が見落としてしまう最適解を求めるため、セレクタを有効に利用して回路規模の縮小を狙う手法を 3 つ提案し、従来手法と併用することを提案した。
- 3 つの手法によりそれぞれ 1.0%,6.8%,8.0% の論理で回路規模の縮小ができた。3 つの手法を合わせることにより、14.5% の論理で回路規模の縮小ができた。
- セレクタの利用効率と論理の性質との関係について、入力のコントロール信号としての有効性を簡単に予測する方法、キューブの重なりからセレクタの利用効率を予測する方法の 2 つを提案した。これらを利用して本研究の合成手法の計算時間を短縮できることを示した。
- マルチプレクサ・ベースの FPGA 合成ツールと本研究の手法を比較し、本研究の手法が FPGA 合成ツールでは求められない最適解を求められることを示した。

本研究に関する発表

- "人間の論理解法に基づく入力セレクタを用いた回路構成手法の検討", 電子情報通信学会ソサイエティ大会,1996 年 9 月
- "セレクタを効率的に利用した回路合成手法の検討", 電子情報通信学会春季大会,1997 年 3 月

参考文献

[1] R.K.Brayton et al., "MIS: A multiple-level logic optimization system," *IEEE trans. on CAD.*, vol. 6, No. 6, Nov. 1987

[2] C.Tsai et al., "Generalized Reed-Muller Forms as a Tool to Detect Symmetries," *IEEE trans. comp.*, vol. 45, No. 1, 1996

[3] R.Murgai et al., "Logic Synthesis for Programmable Gate Arrays," *Proc. 27th Design Automation Conference*, 1990