Varchsyn (3): VHDL 記述からの演算シェアリング手法

6N−3

伊藤義行、古林紀哉、若林一敏、藤田友之、前田直孝

NEC

1 はじめに

Varchsyn [1] では、演算器のうち同時には使用されないものを共有し、演算器の入出力を切替えながら利用することによって、演算器数を削減し面積最小化を行なっている。同様の機構は、PISYN [2], Cyber [3] 等の機能合成システムでも実現されているが、Varchsyn の演算器共有部では VHDL 言語記述レベルでの演算器共有を実現している。本稿では、演算器共有に伴って生成されるマルチプレクサ (MUX) を最小にする演算器共有を VHDL 言語記述レベルで実現する手法について述べる。

2 処理手順

Varchsyn の VHDL 入力部は、入力された VHDL 言語を中間形式に変換する「構文意味解析部」と、中間形式を解釈してシェル内部の回路データを生成する「回路データ作成部」の2つの部分から成っている (図1)。

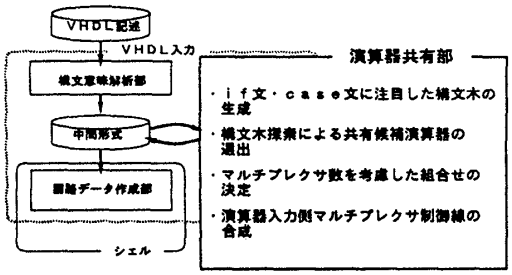

図1. VHDL ソースからのフロー

本システムでは、中間形式レベルの VHDL 記述を対象に、4章で述べる手法により演算器を組合せ、3章に述べる方法で VHDL 記述を生成している。さらには、共有結果からの VHDL 書き戻しシステムの作成により、次章で述べる利点が提供できるようになる。

3 VHDL 言語レベルでの演算器共有

演算器共有を VHDL 言語記述レベルで行ない、共有結果の VHDL 言語への書き戻しを可能にしたことにより、以下のような利点を設計者に提供できる。

1. 可読性：演算器の共有結果をユーザが容易に理解可能である。
2. 編集性：共有結果に手を加え、制約条件回避策の誘導を図ることができる。
3. 他のツールとの接続性：共有前のオリジナル・ソースに使用できるツールが、共有後も利用できる。

3.1 VHDL 記述生成上の問題点

VHDL 言語記述レベル演算器共有では、VHDL 言語から回路データを合成する前に演算器共有を行なうため、ハードウェア合成系毎に定義される VHDL 言語記述のハードウェア認識規則によって、演算器共有部が想定していない回路が合成される可能性がある。特に、演算器の被演算項を切替える MUX を記述する際、if 文・case 文等の条件式の内容による認識規則によって問題が発生する (図2)。

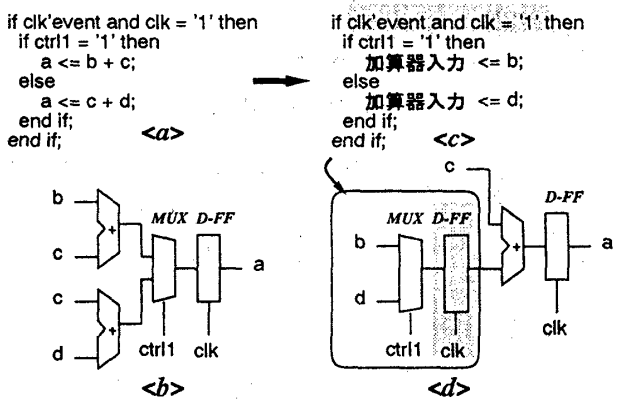

図2. ハードウェア認識規則を考慮していない共有演算器記述

図2-a の VHDL 記述例では、信号線 "a" は、回路データを作成する際、ハードウェア認識規則であるエッジ記述 (clk'event and clk = '1') によって D フリップフロップであると解釈される (図2-b)。演算器共有部が、この VHDL 記述から演算器への入力信号を生成する際、演算器が利用される条件をエッジ記述の存在を考慮せずにコピーすると (図2-c)、MUX 制御線記述にエッジ記述が混入し (図2-c 網部)、後に回路データ作成を行なう際、図2-d の網部に示される余分な D フリップフロップが生成されてしまう。

3.2 共有結果の VHDL 記述

前節の問題点に対し、本システムでは、以下の方針で解決を図ることで、ハードウェア認識規則からの独立性を保ちながら、エッジ記述の排除等を行なっている。これにより、D フリップフロップを認識してからエッジ記述を排除する手法で問題となる、言語仕様の変更やシステムの機能拡張に伴う認識規則の変化にも容易に対応できるようになっている。

演算器 共有された演算器の本体を、architecture block の最上層に独立した process として記述し、共有済みの演算器が単一のハードウェアであることを VHDL 言語レベルで明示する。

MUX 演算の共有が if 文・case 文等の構文上に明示された排他条件で行なわれている (4.1節) ため、共有される演算が存在した構文木上の節点に MUX への信号代入文を記述することで、構文木が MUX のハードウェア認識規則に一致する。その際、排他条件の判断に用いた構文木を、次節に述べる処理により縮約することで、構文木上からエッジ記述等を排除して MUX を生成する。

3.3 構文木縮約による被演算項切替え条件の選択

3.1節で述べたように、MUX 以外のハードウェア認識規則との干渉を防ぐため、エッジ記述等を MUX の制御線記述から排除しなければならない。また、入力された VHDL 記述中から、共有された演算の MUX の制御に用いるべき条件式のみを抜き出す必要がある。

本システムでは、条件式の内容に立ちいることを避け、構文木の形状により、次のように条件式を取捨選択している。

1. MUX の代入文が存在しない枝は削除する (図3-1)。
2. else 節のない if 文等、分岐のない節点は、削除して無条件実行化する (図3-2)。

Varchsyn(3): An operator resource sharing applied to VHDL descriptions.
Y. Itoh, N. Kobayashi, K. Wakabayashi, T. Fujita, N. Maeda
NEC Corporation

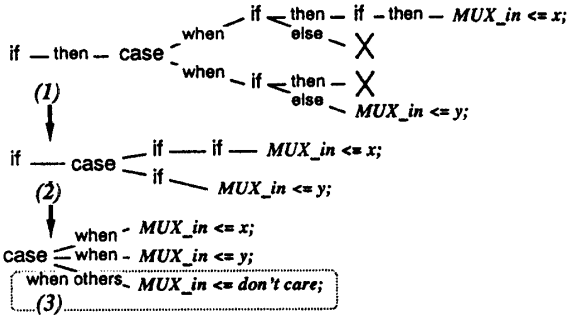

図 3. 構文木の縮約

3. other 節のない case 文には、 Don't care 相当の記述を挿入する (図 3-3)。

上記の構文木の縮約によって、 MUX 制御に関する記述のみを構文木上の節点として残すことができ、ハードウェア認識規則の解釈をせずに、他のハードウェア認識規則と干渉しない MUX 記述を生成できる。

4 演算器共有アルゴリズム

本手法は、演算器の数を最小化するように演算を組合せ、かつそれらの組合せの中で、演算器の共有に伴って生成される MUX の数をできるだけ少なくするような割り当てを選択する。

4.1 構文木探索による演算器共有

本手法は、入れ子条件分岐の葉側 (ネストの内側) から再帰的に構文木上の演算子を組み合わせる。

図 4の VHDL 記述を例に、本手法の概要を説明する。まず最も内側の if 節 (図 4-1) に対して共有処理が施される。 then 節、 else 節ともに加算演算が 1 つづつ存在しているが、この 2 つを 1 個を演算器に割り付ける。この場合、 MUX 数が少なくなるように、 else 節の加算の被演算項の入れ換えが行われる。図 5-1 が、その時の共有された演算器の様子を示している。次に、 then 節 (図 4-2) に対しては、図 5-2 のように 2 つの加算器が割り当てられる。 else 節 (図 4-3) では、演算器の共有は行われず、図 5-3 のように 3 つの加算器が割り当てられる。

最後に if 節 (図 4-3) に対して、 then 節 (図 4-2) の 2 個の加算器と、 else 節 (図 4-3) の 3 個の加算器の間で共有処理が行なわれる。この場合、 (2) と (3) を組み合わせて MUX 数が最小になるように、 3 個の加算器に割り付けられる (4.2 節参照)。図 5-4 が最終結果である。

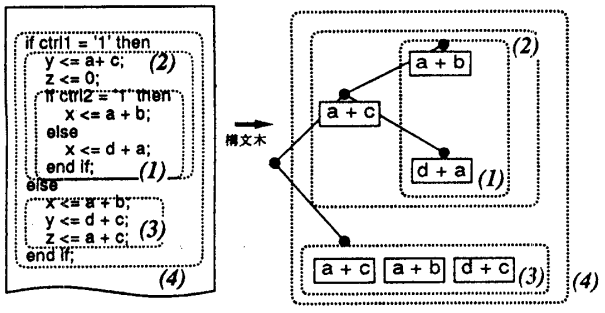

図 4. VHDL ソースとその構文木

図 6に本手法を流れを示す。本手法は、 VHDL の if 文、 case 文等の条件分岐文よる構文上での実行の排他性を利用して演算器の共有を実現している。

4.2 演算器組合せ評価基準

演算器の共有に伴って生成される MUX 数の大域的な最小化は、計算量の点で非常に困難な問題であるため、本手法では、構文木の各節点で演算を組合せる際に、被演算項の交換も含めて、その時点で生成される MUX 数が最小になるような組合せを選択する。そのような組合せは、二部グラフの最小コスト完全マッチングを求めるアルゴリズムを利用することで、多項式時間で見つけることができる。

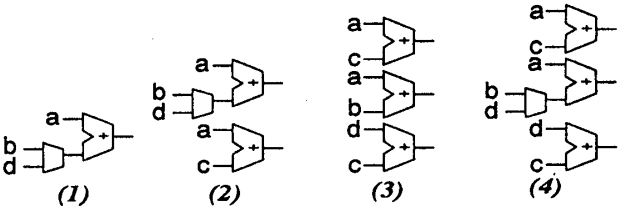

図 5. 図 4に対する演算器共有処理

```
algorithm share(node; 節点);
begin
  if(node が葉でない場合) then
    begin
      node の各々の子の部分木 child に対して
          share(child) を実行;

      if(node が条件分岐の場合) then
        /* 共有の実行 */
        部分木の演算器集合を
            演算器数が最小になるようにかつ
            MUX 数が最小になるように
            組み合わせて節点に割り付ける;
      else /* node が条件分岐でない場合 */
        部分木の演算器集合の和を節点に割り付ける;
    end;
end;
```

図 6. 演算器共有アルゴリズム

5 実験結果

いくつかの VHDL 記述に対して演算器共有を行なった結果を表 1に示す。本システムで最終的に生成される MUX 数は、 MUX 数を考慮せずに共有を行なった場合に比べ、どれも小さくなっている。

表 1. 演算シェアリング適用結果

ソース No.	VHDL ソース中の演算器の種類と数	共有処理後の演算器の数	共有処理後の MUX 数	単純共有の MUX 数
1	“+”×11	“+”×5	5	12
2	“+”×11	“+”×5	5	12
3	“+”×12	“+”×7	5	10
4	“+”×16	“+”×4	16	24
5	“+”×8 ,“−”×8	“+”×7,“−”×7	3	4
6	“+”×12,“−”×12	“+”×8,“−”×9	8	14

6 おわりに

論理合成システム Varchsyn における、 VHDL 言語レベルでの構文木を基にした、演算器の共有による演算器数削減を手法について述べた。特に、共有によって生成される MUX の数を考慮した演算器の組合せ手法、および、構文木縮約による MUX 生成手法によって、ハードウェア認識規則を解釈せずに MUX 以外のハードウェア認識規則との干渉を防ぐ方法について述べた。

今後は、構文木構造による排他的実行の検出に加え、変数・信号線の依存関係を認識した、より適用範囲の広い演算器共有を行なう予定である。

参考文献

[1] 河原林政道, 浅香俊治, 水嶋紀子, 佐藤克也, 前田直孝 : Varchsyn(2):VHDL からの合成, 情処第 46 回全国大会 (1993).

[2] Casavant, A. E., Hwang, K. S. and McNall, K. N.: PISYN — High-Level Synthesis of Application Specific Pipelined Hardware, *High-Level VLSI Synthesis*, pp. 55–78, Kluwer Academic Publishers (1991).

[3] Wakabayashi, K.: Cyber: High Level Synthesis System from Software into ASIC, *High-Level VLSI Synthesis*, pp. 127–151, Kluwer Academic Publishers (1991).