flipflop の同期部抽出による HDL シミュレーションの高速化

2B－4

長井 寛志　庄司 稔
富士通(株)

1 はじめに

ハードウェア記述言語 (HDL) を用いた回路の設計は，回路規模が大きくなるに従い重要性が増してきており，HDL のシミュレーションの高速化は設計期間を短縮する上で重要な要素といえる．シミュレーションの高速化の一手法として並列処理によるシミュレーションが挙げられる．

並列処理のシミュレーションで HDL を構成する同時処理文と順次処理文をシミュレートした場合，シミュレーション時間の長い順次処理文が並列度を低下させるため，シミュレーション高速化の妨げになる．その解決策としては処理順に依存しない組合せ回路の順次処理文を同時処理に変換 (以下，同時処理化と呼ぶ) を行なうことが考えられる．しかし，順次処理の多くを占めるフリップフロップはクロックエッジの判定と信号代入の依存関係が保てなくなるため直接同時処理化を行なうことができない．

そこで順次処理文で記述されたフリップフロップをクロックに同期した信号代入と組合せ回路に分離する方法について示す．この分離によって組合せ回路への同時処理化が可能になる．また我々が開発したハードウェア CAD アクセラレータ TP5000[1] 上で動作する VHDL シミュレータ [2] にこの変換を適用した結果を示す．

2 HDL シミュレーション

HDL は各々の文が独立して同時に処理をすることが可能な同時処理文と，記述された順序で処理をする順次処理文から構成される．VHDL による順次処理文と同時処理文の例を図 1に示す．(1) が順次処理文，(2)(3) が同時処理文である．(1)(2)(3) はそれぞれ同時に動作するが，(1) は一連の文の動作が (2)(3) の 1 文と同じ処理単位として扱われる．

並列処理のシミュレーションで同時処理文，順次処理文 1 つのシミュレーション時間を 1 として考えて (1)(2)(3) をシミュレートした場合，1 文で構成される (2)(3) は 1 で終了するが (1) は 3 つの文から成るため 3 かかり，HDL 全体としてのシミュレーション時間は 3 になる．つまり，順次処理文の処理によってシミュレーション速度が決定してしまうことになる．そのため並列処理のシミュレーションの高速化を行なうには，順次処理のシミュレーション時間を短くする必要がある．

```
process(a,b,c,d,e,f)
begin
   sig1 <= a and b;
   sig2 <= c and d;          } (1)
   sig3 <= e or f;
end process
sig4 <= c when sel2 ='1' else d;     (2)
sig5 <= sig3 and sig4;               (3)
```

図 1: 順次処理と同時処理の例

順次処理のシミュレーション時間を短くする一手法として順次処理文の同時処理化が考えられる．これは順次処理で記述された回路が図 1 の (1) のような組合せ回路であるならば，各々の接続関係を保てば同時処理をしても結果は変わらないということを用いる．順序処理にこの変換を行なえば普通は順番に処理される各文が並列に処理されるため，シミュレーション時間が 1 で済むことになる．

3 フリップフロップの抽出

ここで 2 つの回路を用意し順次処理の内容を調べた結果を表 1に示す．この表を見ると順次処理記述では組合せ回路に比べ，フリップフロップの数がかなり多いということが分かる．これらの回路の組合せ回路を同時処理へ変換しただけではそれほど効果が期待できない．このような回路のシミュレーションを高速化するには，フリップフロップを同時処理化する必要がある．しかし，フリップフロップはクロックの立ち上がりや立ち下がり (クロックエッジ) で処理を行なう順序回路であり，クロックの判定と信号代入の依存関係を

表 1: 順次処理の内訳 (%)

回路	no.1	no.2
組合せ回路	20	27
フリップフロップ	80	70
その他	0	3

Speed up of HDL simulation using flipflop recognization
Hiroshi NAGAI, Minoru SHOJI
FUJITSU LIMITED
4-1-1 , Kamikodanaka Nakahara-ku, Kawasaki 211,Japan

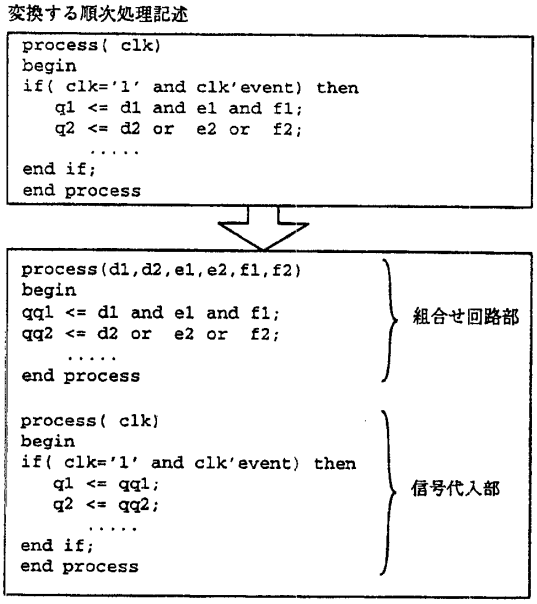

図 2: フリップフロップの変換の例

失ってしまうため単純に同時処理化を行なうことはできない.

そこで実際に依存関係を持っているのはクロックエッジの判定部と信号代入の部分であり, 信号代入同士や入力信号には依存関係がないという点に着目した. 順次処理のフリップフロップは組合せ回路とフリップフロップが組み合わさったものなので, 信号代入とそれ以外の部分 (組合せ回路) に分離することで別々の順次処理として扱えるようにする. フリップフロップの分離手順は以下のとおりである.

1. フリップフロップの動作の記述であるか調べる.
2. 信号代入している部分を探索する.
3. 分離用の信号を作成し, 信号代入の入力はその信号に行なうようにする.
4. 分離用の信号の出力を元の出力信号に代入する.
5. 信号代入とそれ以外の部分を別の順次処理にする.

以上の操作により, 順次実行文はクロックトリガの信号代入と組合せ回路に分離される. VHDL でこの変換を行なった例を図 2 に示す. qq1, qq2, … が分離用の信号である. しかし, 信号代入が 1 段増えるため, 遅延などが起きないようにシミュレーションを行なう必要がある. また, 分離した組合せ回路は同時処理化することができるため, 順次処理はクロックに同期した信号代入のみとなる. 以上の操作でフリップフロップのシミュレーション時間を短縮することができる.

4 実験及び結果

並列処理のシミュレーション環境として CAD アクセラレータ TP5000 を用いた. 今回は HDL を直接変換するのではなく, シミュレータ用の回路データベースに対して変換を行なった. また, TP5000 にはフリップフロップと同様の動作をする素子が組み込まれておりそれを利用するため, 先ほど示した手順に加え,

6. 分離した信号代入部分をフリップフロップ の素子に置き換える.

という手順を加え, 信号代入に関しても同時処理化した.

先ほどの 2 種類の回路に上記の変換を行ない, シミュレーション時間, 発生イベント数の結果を表 2に示す. また回路の大きさの目安としてアクセラレータ中で使用される素子 (プリミティブ) の数についても同様に示す. no.1 , no.2 の回路共にプリミティブ数はそれほど変化はないが, シミュレーション時間は約 2 割, イベント数は約 4 割減少している. シミュレーション時間の減少率が低い理由はイベント起こすための評価回数がそれほど減っていないためで No.1 の回路では評価回数は約 2.5 割減というシミュレーション時間の減少と近い値であった.

表 2: 変換結果の比較

circuit		no.1	no.2
simulation time[sec]	origianl	9.7	371.4
	converted	7.9	303.4
number of events	origianl	11204800	1085761263
	converted	6645512	620521476
number of primitives	origianl	30970	201620
	converted	27848	195236

5 結論

同時処理化できない順次処理文で記述されたフリップフロップを信号代入とランダムロジックに分割し, 組合せ回路を同時処理化を行なった. またフリップフロップの素子を使用し信号代入の同時処理化を行なった. 以上の変換により並列処理の HDL シミュレーションの時間を約 2 割, イベント数を約 4 割減少させることができた. このことからこの変換は順次処理記述のフリップフロップを含む HDL の並列シミュレーションに有効であると言える. これからは順序処理記述の RAM などを扱う方法について進めていく予定である.

参考文献

[1] 下郡 他, "CAD 高速化のための Thread Processor TP5000", FGCS'94 ワークショップ「並列/分散処理による LSI-CAD」, pp.17-24, 1994.

[2] M.Shoji et al., "VHDL Compiler of Behavioral Descriptions for Ultrahigh-Speed Simulation", Proc. of APCHDL'94, pp.85-88, 1994. Oct. 1994.