連載 HDLの記述スタイル

第4回 if文，case文の考え方とドント・ケアXの問題

長谷川裕恭

今回はif文やcase文の記述スタイルについて説明する．HDL設計では，可読性のよいコードからよい回路が生成されるわけではない．論理合成の結果を考慮して記述をチェックする必要がある．ここではよいif文の見分け方，並行に動作する回路とプライオリティ・ロジックの使い分け，ドント・ケアXに関する注意事項などについて解説する．なお，本連載の内容は，IPコアの再利用性や設計データの相互運用性を高めるために半導体理工学研究センター(STARC)とエッチ・ディー・ラボがまとめた『設計スタイルガイド』にもとづいている．

(編集部)

筆者らがまとめた『設計スタイルガイド』には，RTLのコーディング規定が数多く紹介されています．RTL記述は，シミュレーションが実行でき，論理合成ツールでゲート・レベルの回路を生成できればOKというわけではありません．RTLとゲート・レベルの回路は1対1に対応するものではありません．論理合成ツールはRTL記述に独自の解釈を加えてゲート・レベルの回路を生成しています．コーディング・ルールを守らないと，RTLとゲート・レベルのシミュレーション結果が異なったり，あるいは作成したFPGAやASICが正常に動作しないこともあります．なお，RTLコードを記述する際にコーディング・ルールを守らないとどのような障害が発生するのかについては，本号の特集記事(pp.37-94)にいくつかの例が紹介されています．

『設計スタイルガイド』では，最低限，守らなければならないコーディング・ルールだけでなく，生成される回路の性能を考える記述スタイルが紹介されています．藤田昌宏氏による本誌連載記事のタイトルにもあるように，論理合成ツールは魔法の道具ではありません．冗長に書かれた論理は冗長なまま残ることが多く，合成後の回路の性能を劣化させます．また，論理合成ツールは，回路のアーキテクチャを自動生成するものではありません．よく考えられたデザインを入力するのと，よく考えられていないデザインを入力するのとでは，面積(ゲート数)や動作速度などにかなりの差がでてきます．

今回は，性能に影響を与える記述のうち，if文とcase文の記述の考え方に焦点をあてて解説します．

●VHDL記述におけるよいif文の見分け方

リスト1のVHDL記述を見てください．**リスト1**(a)のif文と**リスト1**(b)のif文では，どちらのほうがよい記述例だと思いますか？

論理合成結果を示すと，**リスト1**(a)の記述が35ゲート，0.44ns，**リスト1**(b)の記述が24ゲート，0.25nsとなります(論理合成ツールはDesign Compiler 2000.05-1，ライブラリは0.18 μ mプロセスのセル・ベースIC)．この場合，**リスト1**(b)の記述のほうがよい記述だと考えてください．

では，なぜ**リスト1**(b)のほうがよい記述なのでしょうか？ よいif文の見分け方として，二つのポイントがあります．

(1) 同じ内容の条件式を減らす

第1番目に，同じ内容の条件式の登場回数に注目します．**リスト1**(a)の記述では，A='0'というデコードが4回登場しています．A='1'もA='0'を反転しただけのものと考えると，合計7回登場していることになります．こ

〔リスト1〕if文の記述例（VHDL）
(a)の記述と(b)の記述ではどちらがよいか？

(a) 記述例1

```
process (A,B,C) begin
   if( A='0' and B='0' and C='0' ) then
       Y <= "010";
   elsif(A='0' and B='0' and C='1') then
       Y <= "101";
   elsif(A='0' and B='1' and C='0') then
       Y <= "100";
   elsif(A='0' and B='1' and C='1') then
       Y <= "000";
   elsif(A='1' and B='0' and C='0') then
       Y <= "001";
   elsif(A='1' and B='0' and C='1') then
       Y <= "110";
   elsif(A='1' and B='1' and C='0') then
       Y <= "111";
   else
       Y <= "011";
   end if;
end process;
```

(b) 記述例2

```
process (A,B,C) begin
  if(A='0') then
    if(B='0') then
      if(C='0') then
        Y <= "010";
      else
        Y <= "101";
      end if;
    else
      if(C='0') then
        Y <= "100";
      else
        Y <= "000";
      end if;
    end if;
  else
    if(B='0') then
      if(C='0') then
        Y <= "001";
      else
        Y <= "110";
      end if;
    else
      if(C='0') then
        Y <= "111";
      else
        Y <= "011";
      end if;
    end if;
  end if;m
end process;
```

れに対して**リスト1**(b)の記述では，A='0'の条件式は1回しか登場しません．**リスト1**(b)の記述でB='0'の条件式は2回登場していますが，**リスト1**(a)の7回よりは少なく，またC='0'についても**リスト1**(b)のほうが少なくなっています．したがって，右側のほうがよい記述であると考えてください．

このように，if文を記述するときは，同じ内容の条件式の登場回数が少なければ少ないほどよい結果を得やすくなります．

(2) if-if，if-elsifのネストは5回まで

第2番目に，if-ifやif-elsifの登場回数に注目します．if-ifやif-elsifのネストが長くなると，それだけ直線的な（シーケンシャルな）回路が生成され，性能を悪化させやすいと考えてください．

厳密に言うと，if-ifのネストとif-elsifのネストは意味が異なります．しかし，人間が記述を見ただけで，これらの違いを判別することはかなり困難です．if-ifのネストとif-elsifのネストは，ほぼ同じと考えてください．**リスト1**(a)の記述ではif-elsifのネストは7回あります．これに対して**リスト1**(b)の記述では3回しかありません．したがって**リスト1**(b)のほうがよい記述であると考えてください．

『設計スタイルガイド』では，直線的に長いパスを発生させないため，if-ifやif-elsifのネストは5回くらいまでを推奨しています．ただし，記述によっては，もう少し多くのネストを使用しないと記述できないことがよくあります．すべてのif文のネスト回数を5回までに抑えることはできなくても，少なくとも10回を超えるネストは使わないようにしてください．

●Verilog HDLとVHDLではif文の合成結果が異なる

リスト2は，**リスト1**のVHDL記述をVerilog HDLで記述したものです．じつはDesign Compiler（米国Synopsys社製）という論理合成ツールを使用すると，VHDLとVerilog HDLの間で異なる結果が出力されます．このVerilog HDL記述に対して論理合成を適用すると，**リスト2**(a)の記述は25ゲート，0.25ns，**リスト2**(b)の記述は24ゲート，0.25nsとほぼ同じ結果になります（論理合成ツールはDesign Compiler 2000.05-1，ライブラリは0.18μmプロセスのセル・ベースIC）．

同じ結果が得られるのであれば，**リスト2**(a)の記述のほうが**リスト2**(b)よりも可読性が高く，よい記述と判断したくなるかもしれません．しかし，ちょっと待ってください．Verilog HDL記述の場合に同じ結果が得られたのは，Design CompilerのVerilog HDL版にのみauto_parallel_case機能があるためです．Design Compilerは，**リスト2**(a)の記述（if-else if）を判断するとき，各項目に重なりがなければ（プライオリティ・ロジックを付加しなくてよいと判断し），各記述が並行に動作するものとして回路を生成します．もし，並行動作と判断すると，if-else ifのネストの回数は関係なくなり，**リスト2**(a)も**リスト2**(b)もほぼ同じ性能の回路に