連載 HDLの記述スタイル

第5回

性能を考慮した記述とミスの入らない記述

長谷川裕恭

if文で記述する場合，同じ内容の条件の登場回数やif-if，if-elsifのネストの回数を考慮すべきであることは前回述べた．そのほかに注意すべき点として，余分なプライオリティを発生させないということが挙げられる．これを考慮していないif文では性能を低下させるだけでなく，デバッグにも悪影響を与える場合がある．今回はプライオリティを考慮したif文の記述例などを紹介する．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』に準じている．（編集部）

前回（本誌2001年5月号，pp140-145）で紹介したように，性能を考慮して記述するには，まずプライオリティ・ロジックを考えます．プライオリティはif文で表現します．もちろん，ここで余分なプライオリティは排除しておきます．

並行動作する回路は，andやorを単純に並べる論理式で実現します．論理式で単純に表現できない場合のみ，case文で記述します．case文は，最後に選択する文法だと考えたほうがよいでしょう．

前号で述べたように，case文において，入力された信号のすべての条件を記載しない場合，default項で出力信号にX（ドント・ケア）を代入しないと，面積や速度の極端な劣化を招いてしまいます．default項でXを代入したとしても，case文は論理合成ツールにとってあまり得意な文法ではありません．case文を使用する場合は，その規模を小さく抑えることで性能の劣化を防ぐように心がけてください．『設計スタイルガイド』では，case文の入出力のビット幅は合わせて24本以内，項目の数は100項目以内にとどめることを推奨しています．

●真理値表を考えない

従来の論理回路設計では，最初に真理値表を作成し，そこから論理回路を考えていた設計者が数多くいました．しかし現在のRTL記述による回路設計では，最初に真理値表を考えるのは好ましくありません．最初に真理値を考えると，大きいcase文を作ってしまいがちです．論理は並行動作するのか，プライオリティを付けるのか，などを考慮し，それをどうつなぐかを考えることが重要です．回路の構造さえ考えてしまえば，あとはRTL記述の文法で簡単に表現できるのです．

●if文の考え方

前号で紹介したように，if文を記述する際のポイントは以下の二つです．

⑴ 同じ内容の条件式の登場回数を少なくする

⑵ if-ifまたはif-elsifのネスト回数を少なくする（通常5段以内，単純なネストなら12段以内）

このポイントを守り，余分なプライオリティ・ロジックを発生させないように心がけてください．

リスト1(a)，**リスト2**(a)の記述例では，A，B，Cの入力信号をif文の条件項に使用しています．**リスト1**(a)，**リスト2**(a)の記述例のBの条件のif文とCの条件のif文は，**リスト1**(b)，**リスト2**(b)の記述例のようにマージしたほうがよいと考えてください．**リスト1**(b)，**リスト2**(b)の記述例ではif文をマージし，if-ifのネスト段数が一つ減ることになります．しかし，ネスト段数が減ったかわりにB=1 and C=1とif文の中に条件が並ぶことになります．if文の条件のなかで複数の信号をandあるいはor

〔リスト1〕if文のネストを削減する（Verilog HDL）

B=1，C=1のif文はマージして一つのif文にしたほうがよい．ただし，A=0のif文までマージしない．以前，筆者はif-if-ifのネストを80段ぐらいしている記述を見たことがあるが（Design Compilerがアボートするので文句がきた），このような記述で良い結果になることはない．そこまでいかなくても，else ifのネストで20段，30段ぐらいは比較的よく見かける．そのような記述はやはり性能的に問題になる場合が多い．ネスト回数は，気持ちは5段，単純なelse ifでも10段ぐらいには抑えるべきである．

```
always @ ( posedge CK or negedge RST_X )
   if( !RST_X )
      Y <= 3'b000;
   else if( A==1'b0 ) begin
         if( B==1'b1)
            if( C==2'b01 )
               Y <= DIN;
      end
   else
      Y <= 3'b010;
```

(a) 記述例1

```
always @ ( posedge CK or negedge RST_X )
   if( !RST_X )
      Y <= 3'b000;
   else if( A==1'b0 ) begin
         if( B==1'b1 && C==2'b01 )
            Y <= DIN;
      end
   else
      Y <= 3'b010;
```

(b) 記述例2

〔リスト2〕
if文のネストを削減する（VHDL）

B=1，C=1のif文はマージして一つのif文にしたほうがよい．ただし，A=0のif文までマージしない．

```
process (CLK, RST_X) begin
   if(RSTX='0') then
      Y <= "000";
   elsif(CLK'event and CLK='1') then
     if(A='0') then
       if(B='1') then
         if(C='01') then
           Y <= DIN;
         end if;
       end if;
     else
       Y <= "010";
     end if;
   end if;
end process;
```

(a) 記述例1

```
process (CLK, RST_X) begin
   if(RST_X='0') then
      Y <= "000";
   elsif(CLK'event and CLK='1') then
     if(A='0') then
       if(B='1' and C='01') then
         Y <= DIN;
       end if;
     else
       Y <= "010";
     end if;
   end if;
end process;
```

(b) 記述例2

で接続したものは並行動作します．if-ifのネストを削減したほうが，余分なプライオリティが発生しないだけ得であると考えてください．ただし，Bの条件とCの条件はマージしても，Aの条件までマージしてはいけません．なぜなら，Aの条件を記述したifにはelse項が存在するからです．Aの条件までマージすると，elseですんでいた項がelsifになってしまい，ネスト段数は2段のまま変わらないことになります．さらにelsif(C!=0)と同じ内容の条件式が再度登場することになるので逆効果です．

実際には，このような細かいところを変更したぐらいでは，論理合成した後の結果に大きな影響を与えることはありません．しかし，このような感覚をもっていないと，極端に性能の悪い記述を平気で作成するようになってしまいます．つねに正しい姿勢で記述するように心がけてください．正しい姿勢で記述する習慣が身につくと，自然に余分な論理を記述することも少なくなります．このことにより，さらに品質のよい回路が生まれると考えてください．

●中間変数

ある程度以上の論理で同じものが複数存在する場合，積極的に中間変数を利用するようにしてください．**リスト3**(a)，**リスト4**(a)の記述例では，まったく同じ関係演算が二つ存在します．この場合，中間変数でくくりだすと面積を削減することができます．if文の記述スタイルで，同じ条件式をできるだけ削除すると言いましたが，異なるalways文あるいはprocess文にまでその注意を向けてください．ただし，ある程度の論理規模がないと効果がないので，これは関係演算がある場合または4入力以上ある場合のみ有効であると考えてください．

●always文から出力する信号を少なくする

一つのalways文から出力する信号の数が多過ぎると，生成される回路の速度や面積が劣化することがあります．