基礎から学ぶ Verilog HDL & FPGA 設計

第4回
順序回路の設計 フリップフロップとカウンタ

中野浩嗣，伊藤靖朗

デバイスの記事　ビギナーズ

前回（本誌2007年8月号，pp.129-133）は，算術論理演算回路を設計した．今回は，データを記憶するためのフリップフロップとカウンタを設計する．（筆者）

順序回路は過去の信号の入力に依存して出力が決まる回路です．そのため，順序回路では過去に入力した信号を記憶しておく必要があります．D型フリップフロップ（以下では単にフリップフロップと呼ぶ）をビットのデータを記憶するのに使います．まず，フリップフロップをVerilog HDLで設計しましょう．

● フリップフロップの設計

リスト1はフリップフロップのVerilog HDL記述です．7行目のalways文のイベント・リストは，posedge clkとnegedge resetの二つからなります．ここで，posedgeは立ち上がり（0から1への変化）を，negedgeは立ち下がり（1から0への変化）を意味します．従って，clkが立ち上がった場合，またはresetが立ち下がった場合に次に続く文，この例ではif文が実行されます．このif文では，!resetが1のとき（つまり，reset が0のとき），代入文q <= 0が実行され，さもなければ，q <= dが実行されます．この意味を四つの場合に分けて考えてみましょう．

リスト1　フリップフロップのVerilog HDL記述（ff.v）

```
 1  module ff(clk,reset,d,q);
 2
 3    input clk,reset,d;
 4    output q;
 5    reg q;
 6
 7    always @(posedge clk or negedge reset)
 8      if(!reset) q <= 0;
 9      else q <= d;
10
11  endmodule
```

- posedge clkとnegedge resetが同時に発生した場合．

  イベント発生後のresetは0です．従って，q <= 0が実行され，出力値は0になります．

- posedge clkのみ発生した場合．

  resetが0のとき，出力値は0になります．resetが1ならば，q <= dが実行され，入力をラッチします．

- negedge resetのみ発生した場合．

  resetは0なので，clkの値に関係なく，出力値は0にリセットされます．

- イベントが発生しない場合．

  qに値が代入されないので，qの値は保持されます．

以上の動作を整理すると，**表1**のようになります．クロックclkの立ち上がりに同期して値をラッチするため，「同期ラッチ」と呼ばれます．リセットresetの値が0だとclkに関係なく0を保持するため「非同期リセット」と呼ばれます．また，フリップフロップは**図1**のように表されます．

表1　フリップフロップの動作

入力		出力
clk	reset	q
—	0	0（非同期リセット）
↑	1	d（入力の同期ラッチ）
↑以外	1	直前のq（値の保持）

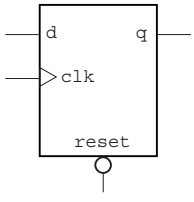

図1　フリップフロップ

KeyWord　フリップフロップ，カウンタ，非同期リセット，同期ラッチ，ノンブロッキング代入文，ブロッキング代入文，組み合わせ回路，完全同期式回路，シミュレーション実行時間，トップ・モジュール

● ブロッキング代入文とノンブロッキング代入文

リスト1の8行目と9行目の代入文では，「<=」を使っていますが，このような代入文を「ノンブロッキング代入文」と呼びます．一連の並んだノンブロッキング代入文は同時に実行されます．例えば，

```
b <= c;
a <= b;
```

は，cの値とbの値がそれぞれbとaに同時に代入されます．

一方，前回（本誌2007年8月号，pp.129-133）行った算術論理演算回路などの組み合わせ回路の設計には，ブロッキング代入文「=」を用います．ブロッキング代入文が並んで記述されている場合，上の行から順番に代入が行われます．例えば，

```
b = c;
a = b;
```

は，cの値がbに代入され，その後，bの値がaに代入されるので，aに代入されるのはcの値となります．従って，上のノンブロッキング代入文の場合とaに代入される値は異なります．一般に，組み合わせ回路の論理を設計する場合はブロッキング代入文が，順序回路の論理を設計する場合はノンブロッキング代入文が推奨されます．

表2　2ビット・カウンタの動作

入力				出力
clk	reset	load	inc	q
—	0	—	—	0（非同期リセット）
↑	1	1	—	d（入力の同期ラッチ）
↑	1	—	1	q+1（インクリメント）
↑	1	0	0	直前のq（値の保持）
↑以外	1	—	—	直前のq（値の保持）

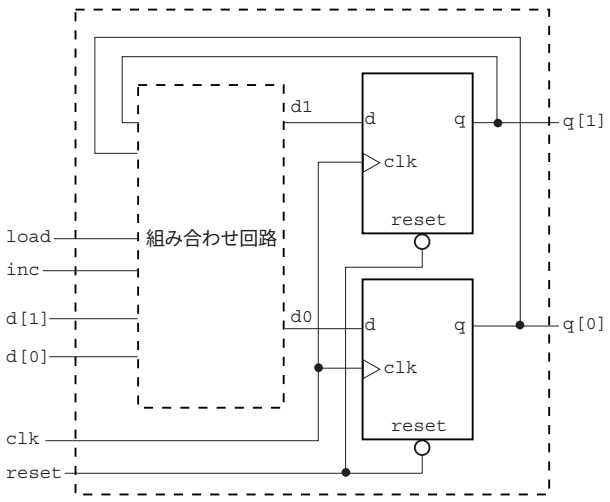

図2　2ビット・カウンタのブロック図

● 2ビット・カウンタの設計

最も簡単な順序回路の一つであるカウンタを設計しましょう．カウンタは2ビットの値を保持するものとし，入力ポートとして，それぞれ1ビットのclk，reset，load，incと，2ビットのdを用います．出力ポートqは，現在のカウンタの値を出力します．resetは，0のときにカウンタの値を0に非同期リセットします．loadが1のとき，clkの立ち上がりでdに入力されているビットの値をラッチします．incが1のとき，clkの立ち上がりでカウンタの値を1増やします（インクリメント）．従って，incが1のとき，出力される値は，00→01→10→11→00→…と変化を繰り返します．loadとincが両方とも0のときは，qの値は変わりません．loadとincが両方とも1になることはないものとします．**表2**は以上の動作をまとめたものです．

フリップフロップと組み合わせ回路を用いて，カウンタを設計できます．記憶する必要があるのは2ビットの値なので，二つのフリップフロップを用いることにします．**図2**はそのブロック図です．カウンタの入力clkとresetは，それぞれ二つのフリップフロップのclkとresetに直結します．また，カウンタの2ビットの出力q（q[0]とq[1]）は，二つのフリップフロップの出力qに接続します．二つのフリップフロップに入力する値d0とd1を，clkの立ち上がりごとにフリップフロップがラッチします．このd0とd1の値を決める組み合わせ回路を設計する必要があります．この組み合わせ回路の入力は，カウンタへの入力load，inc，d（d[0]とd[1]），および，フリップフロップの出力q[0]とq[1]です．これら6ビットの値からd0とd1が決定されます．従って，d0とd1は，load，inc，d[0]，d[1]，q[0]，q[1]の論理式で表せます．以下はその論理式です．

$$d0 = (\overline{load} \wedge \overline{inc} \wedge q[0]) \vee (load \wedge d[0]) \vee (inc \wedge \overline{q[0]})$$

$$d1 = (\overline{load} \wedge \overline{inc} \wedge q[1]) \vee (load \wedge d[1]) \vee (inc \wedge (q[0] \oplus q[1]))$$

この論理式が正しいことをd0について確認してみます．loadとincが両方とも0のとき，d0 = q[0]なので，clkの立ち上がりでq[0]の値は変わりません．loadが1のとき，d0 = d[0]なので，q[0]の値はd[0]となります．incが1のとき，d0 = $\overline{q[0]}$なので，q[0]の値は反