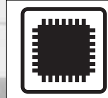
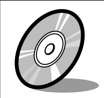
基礎から学ぶ Verilog HDL & FPGA 設計

第2回

4ビット加算器を設計しよう

中野浩嗣，伊藤靖朗





デバイスの記事　ビギナーズ　関連データ

前回（本誌2007年4月号，pp.105-114）は，全加算器をVerilog HDLで設計し，シミュレーションや，FPGAボードへの回路のダウンロードと動作確認を行った．今回は，前回設計した全加算器を用いて，4ビット加算器を設計する．（編集部）

1．4ビット加算器の設計

前回設計した全加算器は，組み合わせ回路の一つです．組み合わせ回路とは，現在の入力にのみ依存して出力が決まる回路で，基本ゲート回路（NOT，AND，OR，XORなど）の組み合わせで設計することができます．今回は，もう少し複雑な組み合わせ回路に挑戦しましょう．

● assign文とalways文

前回は全加算器を設計するのにassign文を用いましたが，Verilog HDLでは，通常，assign文よりもalways文の方がよく用いられます．**リスト1**に，always文を用いた全加算器のVerilog HDL記述を示します．

リスト1　always文を用いた全加算器のVerilog HDL記述（fa.v）

```
 1  module fa(a, b, cin, s, cout);
 2
 3     input a, b, cin;
 4     output s, cout;
 5     reg s, cout;  ← レジスタ宣言
 6
 7     always @(a or b or cin)
 8     begin
 9       s = a ^ b ^ cin;
10       cout = (a & b) | (b & cin) | (cin & a);
11     end
12
13  endmodule
```
（9～10行目：aかbかcinの値が変化するたびに実行される）

4行目までは，前回のassign文を用いた全加算器と同じです．5行目でreg文を使って変数sとcoutをレジスタ型変数として宣言しています．always文は，always @(...)という形で始まります．「…」の部分はイベント・リストと呼ばれ，ここで指定された変数などの値に変化があるたびに，後に続く文が実行されます．ここでイベント・リストは「a or b or cin」なので，入力信号のa，b，cinのいずれかの値が変化するたびに，後に続くbegin～end間の文（9行目と10行目の代入文）が実行されます．よって，変数sとcoutは常に正しい値をとることになります．

5行目のreg文は「レジスタ宣言」と呼ばれ，sとcoutがレジスタ型変数であることを宣言しています．ただし，このレジスタ型変数は，いわゆるレジスタ（論理回路で値を記憶するのに用いる）になるとは限りません．レジスタ型変数は，レジスタにも信号線にもなり得ます．実際，**リスト1**のsとcoutは信号線です．これがVerilog HDLのややこしい点なのですが，ともあれ，以下の点を覚えておくとよいでしょう．

- always文の中で用いられる代入文の左辺は，レジスタ宣言（reg）で宣言されたレジスタ型変数でなければならない．
- assign文の中で用いられる代入文の左辺は，ネット宣言（wire）で宣言されたネット型変数でなければならない．

KeyWord　Verilog HDL，FPGA，HDL，全加算器，イベント・リスト，レジスタ，モジュール・インスタンス化，オーバフロー，1の補数，2の補数，演算子

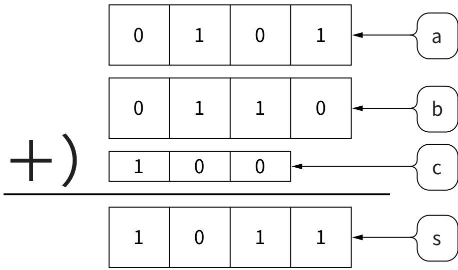


図1　4ビット加算器のイメージ
4ビットの2進数を二つ加算して，4ビットの値を出力する．けた上がりのためのビットも必要．

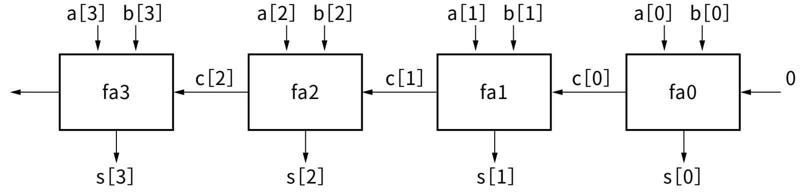


図2　4ビット加算器のブロック図
前回設計したモジュールfaを適切に接続すれば，4ビット加算器を作ることができる．

● レジスタ型変数は，値を保持するレジスタにも信号線にもなり得る．

レジスタ型変数がどのような場合に信号線になり，レジスタになるのかについては，次回以降で詳しく説明します．

● 4ビット加算器を3通りの方法で記述する

次に，4ビットの2進数を二つ足し合わせる4ビット加算器を設計してみましょう．4ビット加算器は二つの4ビットをポートaとbに入力し，その合計を4ビットでポートsに出力します(**図1**)．前回設計した全加算器を四つ並べて適切に接続することにより，4ビット加算器を作ることができます(**図2**)．

ここでは，Verilog HDLのさまざまな記述方法を知るために，以下の三つの方法で4ビット加算器を設計してみます．

● すべての信号線の論理を定義する方法(最も原始的)
● 既に設計したモジュールを部品として利用する方法(モジュール・インスタンス化)
● 算術演算子を用いる方法

● すべての信号線の論理を定義する方法

リスト2は，すべての信号線をassign文で定義することにより設計した4ビット加算器です．3行目のinput文で，aとbが，それぞれ4ビットの入力ポートであることを宣言しています．ポートのインデックスの範囲を[3:0]と指定しているので，a[3]，a[2]，a[1]，a[0]は，4ビットからなる入力ポートaの，上位ビットから並べた各ビットを表すことになります．

入力ポートと同様に，4行目のoutput文では，sが4ビットの出力ポートであることを宣言しています．

5行目のwire文では，cが3ビットのネット(信号線)であることを宣言しています．この3ビットの信号は，入力ポートと出力ポートのいずれでもないモジュール内の信号線となります．

7行目から13行目のassign文で，各信号線の接続を定義しています．信号線c[2]，c[1]，c[0]がassign文の左辺と右辺の両方にあり，これらの信号線が全加算器を接続しています．

リスト2　すべての信号線をassign文で定義することにより設計した4ビット加算器のVerilog HDL記述(adder4.v その1)

```
 1  module adder4(a, b ,s);
 2
 3     input [3:0] a, b;
 4     output [3:0] s;
 5     wire [2:0] c;
 6
 7     assign s[0] = a[0] ^ b[0];
 8     assign c[0] = a[0] & b[0];
 9     assign s[1] = a[1] ^ b[1] ^ c[0];
10     assign c[1] = (a[1] & b[1]) | (b [1] & c[0]) |
11                                  (c[0] & a[1]);
12     assign s[2] = a[2] ^ b[2] ^ c[1];
13     assign c[2] = (a[2] & b[2]) | (b [2] & c[1]) |
14                                  (c[1] & a[2]);
15     assign s[3] = a[3] ^ b[3] ^ c[2];

   endmodule
```
(7～8行目注記：最下位ビット同士の加算)

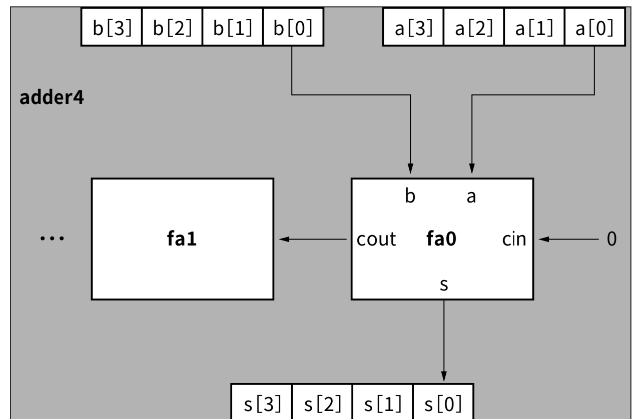


図3　モジュールadder4の信号線とモジュールfaの接続