第3章

初めてでも使えるSystemVerilog文法ガイド

回路記述を改善し，さらに検証記述を追加

近藤 洋

ここではSystemVerilogの文法について，「記述スタイル編」と「文法ガイド編」に分けて解説する．SystemVerilogでは，テストベンチやアサーションを記述するための構文（検証記述）が用意されている．また，回路記述についても，従来より記述量が少なくなったり，ミスを誘発しにくい表現をとれるようになった．（編集部）

本稿では，SystemVerilog（バージョン3.1a）で拡張された文法について説明します．前半の「記述スタイル編」では，RTL記述，テストベンチ記述，アサーション記述の三つに分けて，記述例を用いながら解説します．予約語や文法用語については，後半の「文法ガイド編」でも解説します．合わせてご覧ください．

記述スタイル編

1 RTL記述

ここでは，RTL記述向け（つまり，論理合成可能な）SystemVerilog文法について説明します．

1.1 データ・タイプの拡張

● 基本データ・タイプ

Verilog HDLでは，ネット型（wire）とレジスタ型（reg）の2種類のデータ・タイプがありました．SystemVerilogでは，これらの基本データ・タイプに加えて，表1に示すデータ・タイプが追加されました．

表1　SystemVerilogで拡張された基本データ・タイプ

データ・タイプ	ビット幅	説　明
bit	1	2値（0，1）の符号なし整数．ビット幅の指定が可能
byte	8	2値（0，1）の符号付き整数（C言語のcharと同様）
shortint	16	2値（0，1）の符号付き整数（C言語のshortと同様）
int	32	2値（0，1）の符号付き整数（C言語のintと同様）
longint	64	2値（0，1）の符号付き整数（C言語のlonglongと同様）
logic	1	4値（0，1，x，z）の符号なし論理値．ビット幅の指定が可能

bit，byte，shortint，int，longintは，‘0’，‘1’の2値をとります．これらのデータ・タイプは，おもにシステム・レベル設計（RTLより上位の設計）を対象として拡張されたものです．

logicはregと同じように‘0’，‘1’，‘x’，‘z’の4値をとり，使用方法もregとほぼ同じです．ではなぜ新たに追加されたのでしょうか？ それはregの名まえから連想する回路イメージと実際の回路に差があり，混乱を招きやすかったためです．regはレジスタ型なので，フリップフロップなどの順序回路をイメージしますが，always文で記述された組み合わせ回路の出力信号もregで宣言しなければなりません．regの代わりにlogicを使用することで，このような紛らわしさがなくなりました．

もう一つ改善された点として，ネット型とレジスタ型の使用制限の緩和があります．Verilog HDLでは，always文を使って生成する信号はレジスタ型（reg）で宣言し，assign文を使って生成する信号はネット型（wire）で宣言しなければなりません．SystemVerilogではこの制限が緩和され，表1のデータ・タイプについてはネット型とレ

リスト1　logicを使った記述

```
logic   [1:0]   SELOUT;
logic   [3:0]   DECOUT;

assign SELOUT = SEL ? A : B;

always @( A ) begin
  case ( A )
    2'b00   : DECOUT = 4'b0001;
    2'b01   : DECOUT = 4'b0010;
    2'b10   : DECOUT = 4'b0100;
    2'b11   : DECOUT = 4'b1000;
    default : DECOUT = 4'bxxxx;
  endcase
end
```

Verilog HDLでは，
`wire [1:0]SELOUT;`
`reg [3:0]DECOUT;`

リスト2　unsinged，signed，typedefの記述

```
int                 S_DATA;     // 32ビット符号付き数値
int unsigned        U_DATA;     // 32ビット符号なし数値
reg signed   [15:0] S_REGDAT;   // 16ビット符号付き数値
reg          [15:0] U_REGDAT;   // 16ビット符号なし数値

typedef  int unsigned    uint;
uint           A, B;            // 32ビット符号なし数値
```

リスト3　列挙型を使ったステート・マシンの記述

```
typedef enum logic [1:0] {
  NORMAL_ST, SEC_ST, MIN_ST, HOUR_ST
} state_type;

state_type CUR_STATE, NXT_STATE;

always @(posedge CLK, negedge RST_X) begin
  if (!RST_X)
    CUR_STATE <= NORMAL_ST;
  else
    CUR_STATE <= NXT_STATE;
end

always_comb begin
  case (CUR_STATE)
    NORMAL_ST :
     if (SW2)
        NXT_STATE = SEC_ST;
     else
        NXT_STATE = NORMAL_ST;
    SEC_ST:
     if (SW2)
        NXT_STATE = NORMAL_ST;
     else if (SW3)
        NXT_STATE = MIN_ST;
     else
        NXT_STATE = SEC_ST;
    MIN_ST:
     if (SW2)
        NXT_STATE = NORMAL_ST;
     else if (SW3)
        NXT_STATE = HOUR_ST;
     else
        NXT_STATE = MIN_ST;
    HOUR_ST:
     if (SW2)
        NXT_STATE = NORMAL_ST;
     else if (SW3)
        NXT_STATE = SEC_ST;
     else
        NXT_STATE = HOUR_ST;
  endcase
end
```

ジスタ型の両方で使用することができます．Verilog HDLでは信号宣言の際に，記述スタイルからregとwireの使い分けを判断していましたが，**リスト1**のようにすべてlogicで宣言することができます．

ただし，**表1**のデータ・タイプは，複数の値が同時に代入された場合に最終値を決定する解決機能がないため，複数のalways文やassign文などからの代入はできません．

● unsignedとsigned

Verilog HDL 1995では符号付き数値はintegerだけでしたが，Verilog HDL 2001では符号なし数値を符号付き数値として宣言する修飾子signedが追加されました．これにより，regなどで宣言された任意のビット数の信号に対して，符号付き数値として演算することが可能となりました．

さらにSystemVerilogでは，bitやintなどの符号付き数値を符号なし数値として宣言する修飾子unsignedが追加されています．

signedとunsignedはデータ・タイプの後ろに記述します．C言語の宣言方法と異なるので注意が必要です．

● ユーザ定義型（typedef）

SystemVerilogのデータ・タイプを用いて，ユーザが任意のデータ・タイプを定義できます．よく使用するデータ・タイプや，次に説明する列挙型などのように記述が長くなるものについては，typedefを使って定義しておいたほうが修正しやすくなり，また，記述も見やすくなります．

リスト2にunsigned，signed，およびtypedefを使った宣言の例を示します．

● 列挙型（enum）

列挙型は，取りうる値の集合を列挙名のリストとして宣言するデータ・タイプです．**リスト3**はステート・マシンの状態変数を列挙型で宣言した例です．列挙型を使用することで，ステート・マシンの状態名を列挙名として記述することができます．

列挙名のデータ・タイプはデフォルトでint型になりますが，**リスト3**のようにlogic[1:0]を記述することで，2ビットのlogic型にできます．

また，列挙名の値はデフォルトで指定されたデータ・タイプにより0から順番に値が割り振られますが，**リスト4**