HDLによる設計法実践講座 VHDL編 4

FPGAによるサイモン・ゲームの実現(3)

表示部，テスト・ベンチのVHDL記述

長谷川裕恭

表示部(disp.vhd)

表示部では，制御部(ctrl.vhd)で定義した状態によってLEDに何を表示させるかを決定します．

各状態でRANREGの内容，あるいはキー入力が成功したか失敗したかを一定の時間だけ表示します．RANREGのデータは，3ビットのバイナリですので，6個のLEDに表示させるためにはデコーダ回路が必要となります．このデコーダは，OUT1～OUT6，IN2からIN6までの計11回登場するので，ファンクションとして関数化しました．

この関数は，前号で紹介したパッケージMYPACの中で宣言されています．

リスト4.1の(1)が関数DECODEの宣言となります．VHDLでは，パッケージ文内にファンクション，プロシージャの宣言部しか書くことができません．本体は**リスト**4.1の(3)のように，パッケージ本体の中に記述します．

VHDLでは，戻り値はデータ・タイプしか記述できないことになっています．rで宣言されているSTD_LOGIC_VECTORの範囲は，**リスト**4.3の(2)で代入されるDISPの範囲(5 downto 0)がコピーされます．いっぽう，入力側は範囲を記述してもしなくてもよいことになっています．simonの設計では，入力側は常に3ビットの場合しか使用しません．そのため，STD_LOGIC_VECTORの範囲を規定しています．もし，範囲の規定をしなければ，何ビット長でも使用できるファンクションを作成することができます．

リスト4.2が，ビット長フリーのデコーダ・ファンクションとなります．この場合，ビット範囲が曖昧にならないようにするため，(2)のvariable宣言で戻り値の範囲を定義する必要があります．この関数は，入力の範囲が3ビットであれば，**リスト**4.1のDECODEと等価になります．

リスト4.3の(1)では，各STATEの状態で，LED出力がどのような値をとるかを記述しています．OUT1～OUT6の状態では，RANREGの1～6番目の内容を前述のDECODE関数でワンホットに変換して出力させています．IN1の状態では，いったん表示をクリアして何も表

```
library IEEE;
use IEEE.std_logic_1164.all;
package MYPAC is
 constant W  : integer := 6;
 type RANARRAY is array ( 1 to W ) of std_logic_vector(2 downto 0);


 constant HALT : std_logic_vector(4 downto 0) := "00000";
 constant RAND : std_logic_vector(4 downto 0) := "00001";
 constant OUT1 : std_logic_vector(4 downto 0) := "00010";
 constant OUT2 : std_logic_vector(4 downto 0) := "00011";
 constant OUT3 : std_logic_vector(4 downto 0) := "00100";
 constant OUT4 : std_logic_vector(4 downto 0) := "00101";
 constant OUT5 : std_logic_vector(4 downto 0) := "00110";
 constant OUT6 : std_logic_vector(4 downto 0) := "00111";
 constant IN1  : std_logic_vector(4 downto 0) := "01000";
 constant IN2  : std_logic_vector(4 downto 0) := "01001";
 constant IN3  : std_logic_vector(4 downto 0) := "01011";
 constant IN4  : std_logic_vector(4 downto 0) := "01010";
 constant IN5  : std_logic_vector(4 downto 0) := "01110";
 constant IN6  : std_logic_vector(4 downto 0) := "01100";
 constant FAIL : std_logic_vector(4 downto 0) := "10000";
 constant OK   : std_logic_vector(4 downto 0) := "10110";
 constant NXTG : std_logic_vector(4 downto 0) := "10011";
 constant SUCC : std_logic_vector(4 downto 0) := "10100";
 constant DONE : std_logic_vector(4 downto 0) := "10010";
 function DECODE (A:std_logic_vector(2 downto 0))
                  return STD_LOGIC_VECTOR;
end MYPAC;
package body MYPAC is
 function DECODE (A:std_logic_vector(2 downto 0))
                 return STD_LOGIC_VECTOR is
 begin
   case A is
    when "001" => return("000001");
    when "010" => return("000010");
    when "011" => return("000100");
    when "100" => return("001000");
    when "101" => return("010000");
    when "110" => return("100000");
    when others => return("XXXXXX");
   end case;
 end DECODE;
end MYPAC;
```

(1) DECODEファンクションの宣言

(2) パッケージ本体　ファンクション，プロシージャの本体を記述する

(3) DXECODEファンクションの本体

(4) 戻り値のデータ・タイプ returnの値が返却される

〔リスト4.1〕MYPACの記述(MYPAC2.VHD)

```
function DECODER ( A: in std_logic_vector )
                 return std_logic_vector is

 variable TMP : std_logic_vector(2**(A'length)-1 downto 0);
 variable A_INT : integer;

 begin
    TMP := (others=>'0');
    A_INT := CONV_INTEGER(A);
    TMP(A_INT) := '1';
   return TMP;
 end DECODER;
```

(1) 入力も戻り値も範囲を指定しない

(2) variable宣言で戻り値の幅を明確にする

〔リスト4.2〕ビット長フリーのデコーダ・ファンクションCODE.VHDの記述（CODE.VHD）

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use work.Mypac.all;
entity DISP is
  port( CLK,RESET,KEYDONE : in std_logic;
      HZ2L,HZ4         : in std_logic;
      STATE : in std_logic_vector(4 downto 0);
      RANREG: in RANARRAY;
      LED : out std_logic_vector(5 downto 0));
end DISP;
architecture RTL of DISP is
signal READYS,KEYDONES  : std_logic;
signal DISP : std_logic_vector(5 downto 0);
begin

 LED <= DISP;

process (CLK,RESET) begin
  if(RESET='1') then
    KEYDONES <= '0';
  elsif(CLK'event and CLK='1') then
    KEYDONES <= KEYDONE;
  end if;
end process;

process (CLK,RESET) begin
  if(RESET='1') then
    DISP <= (others =>'0');
  elsif(CLK'event and CLK='1') then
   case STATE is
    when OUT1 => DISP <= DECODE(RANREG(1));
    when OUT2 => DISP <= DECODE(RANREG(2));
    when OUT3 => DISP <= DECODE(RANREG(3));
    when OUT4 => DISP <= DECODE(RANREG(4));
    when OUT5 => DISP <= DECODE(RANREG(5));
    when OUT6 => DISP <= DECODE(RANREG(6));
    when IN2  => DISP <= DECODE(RANREG(1));
    when IN3  => if(KEYDONES='1') then
              DISP <= DISP or DECODE(RANREG(2));
            end if;
    when IN4  => if(KEYDONES='1') then
              DISP <= DISP or DECODE(RANREG(3));
            end if;
    when IN5  => if(KEYDONES='1') then
              DISP <= DISP or DECODE(RANREG(4));
            end if;
    when IN6  => if(KEYDONES='1') then
              DISP <= DISP or DECODE(RANREG(5));
            end if;
    when FAIL => if(HZ2L='1') then  DISP <= (others=>'1');
            else             DISP <= (others=>'0');
            end if;
    when SUCC => if(HZ2L='1') then  DISP <= "101010";
            else              DISP <= "010101";
            end if;
    when DONE => DISP <= "000001";
    when OK   => if(HZ4='1') then
              DISP(5 downto 1) <= DISP(4 downto 0);
              DISP(0) <= DISP(5);
            end if;
    when others => DISP <= (others=>'0');
   end case;
  end if;
end process;

end RTL;
```

(1) caseでSTATEの値によって DISP（LED表示）の値を選択

(2) DECODEファンクションの呼び出し，DISPの範囲が戻り値の範囲となる

(3) othersで残りのすべて（IN1,HALT,RAND,NXTG）が全ビット'0'（何も表示しない）

〔リスト4.3〕表示部DISP.VHDの記述（DISP.VHD）

示しない状態に戻します．記述上では，**リスト**4.3の(3)のothersが実行されます．

IN2～IN6の状態では，その前までのキー入力の値を表示させています．正解ならば，それまでの表示に加え，その順番の表示を追加します．一つの出題を全部正解した場合は，状態OKになります．この状態でLEDは1個のみの表示となり，HZ4信号により表示が1/4秒ごとにシフトされます．

すべて正解した場合は，SUCC状態になります．また，失敗した場合はFAIL状態になります．SUCC状態とFAIL状態では，HZ2L信号が'1'の場合と'0'の場合で交互に値を表示します．

シミュレーションの記述

HDL設計では，シミュレーション・パターンの作成もRTLと同様にHDLで記述します．'1'，'0'のテスト・パターンを手書きで作成したり，波形エディタを使用して作成するのにくらべて，かえってめんどうに思えるかもしれません．たしかに短いテスト・パターンを作成する場合は，単純に'1'と'0'を並べたほうが速く記述できます．しかし，1万行，2万行におよぶテスト・パターンを記述する場合に，'1'や'0'を手書きしていては全体を把握することは困難です．HDLを使用することにより，テスト・パターンを分割し階層化することができます．これにより，パターンの作成，変更が楽に行えるようになります．

さらに進んで，ハンドシェーク技法などにより，信号が出力するタイミングや値が変更されても，テスト・パターンの変更を行わなくて済むように記述することが可能になります．

リスト4.4がsimonのテスト・パターンの記述となります．テスト・パターンを作成する際には，**リスト**4.4の(1)のようにまずクロック周期を定数で宣言します．クロック周期を変更するときは，この30 μsという値を変更するだけで対応することができます．周期を正確に記述すると，