

HDLによる 設計法 実践講座

Verilog-HDL編 4

FPGAによるサイモン・ゲーム機の実現(3) 表示部，テスト・ベンチの Verilog-HDL記述

小林 優

今回は、サイモンの残りのブロックの表示部を解説した後、シミュレーションによる検証環境であるテスト・ベンチについて解説します。

表示部

表示部は、サイモンの複雑な点灯仕様を実現するブロックです。点滅や累積表示、回転などの点灯仕様があります。

ブロック図を図4.1に示します。このブロック図は、後に説明するVerilog_HDL記述とは完全には対応していません。概念を示したものです。HDL記述のほうは、Verilog_HDLの表現力を用いて簡略化して記述してあります。

ステートにより表示内容を切り替える

表示部の出力は、6ビットのレジスタLEDです。ランプの駆動出力になっています。このレジスタの入力を、レジスタ入力生成部で作成します。表示出力は、出題(OUT1～OUT6ステート)、解答入力(IN1～IN6ステート)などのステートに依存します。このブロックを含めた表示部全体が、制御部からのステート信号stateにより制御されます。

出題時と解答入力時には、乱数発生部からの入力ranreg1～ranreg6のいずれか一つをセクタで選択し、出力デコーダDECODEで、6個のランプのうち一つだけ点灯する信号を得ています。

各問正解時(OKステート)のランプ回転や、不正解時(FAILステート)のランプ点滅は、秒カウンタ(secCnt)からのタイミング信号hz2l、hz4によって実現

しています。

記述はレジスタへの代入部とデコーダ部から構成されている

表示部のVerilog_HDL記述をリスト4.1に示します。記述は大きく分けて、表示出力レジスタLEDの記述(always文)と、出力デコーダDECODEの記述(function)から構成されています。ブロック図に示したセクタや、レジスタ入力生成部は、always文中のレジスタLEDへの代入部分で実現しています。

レジスタ代入部はalways文で記述

このalways文中のcase文で、個々のステートに対応したレジスタLEDへの処理が記述されています。表示仕様にあわせて、これらの処理の詳細を説明します。

(1) 出題：OUT1～OUT6ステート

乱数発生部の出力ranreg1～ranreg6

の内容をそのままデコードして表示します。各ステートで、常に1個だけのランプが点灯します。

(2) 解答入力：IN1～IN6ステート

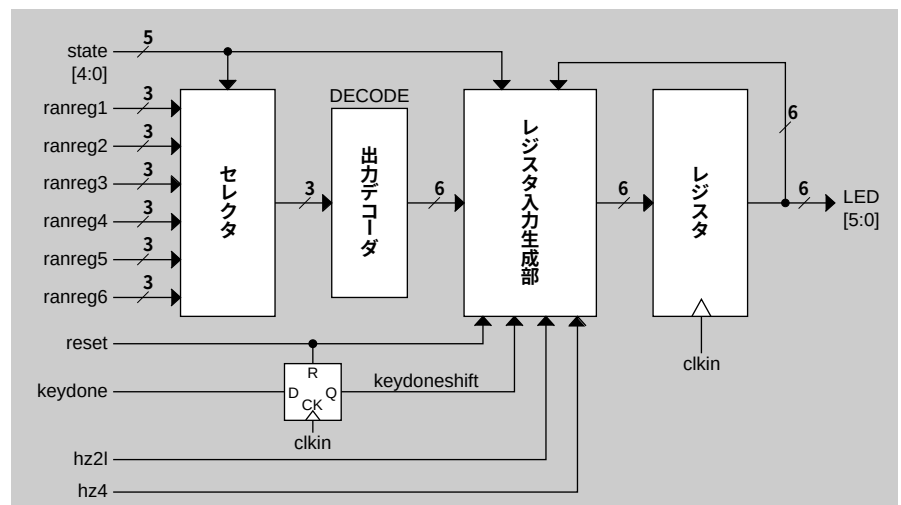
解答キーを入力するたびに累積して表示します。解答入力の各ステートは、入力待ちの状態なので、表示内容は一つ前のステートまでの累積結果です。たとえば、3個目のキー入力待ちであるIN3ステートでは、二つのキー入力を累積して表示しています。

Verilog_HDL記述もそのような回路記述となっています。たとえば

- ・IN1ステート：LEDにすべて'0'を代入
- ・IN2ステート：LEDには1個目だけを代入
- ・IN3ステート：2個目のランプをORして表示

となっています。

累積する値は、解答キー入力の値では



〔図4.1〕 表示部ブロック図

```

module disp ( clkln, reset, state, ranreg1, ranreg2,
             ranreg3, ranreg4, ranreg5, ranreg6,
             keydone, LED, hz2l, hz4 );
input        clkln, reset, keydone, hz2l, hz4;
input  [2:0]  ranreg1, ranreg2, ranreg3, ranreg4,
             ranreg5, ranreg6;
input  [4:0]  state;
output [5:0]  LED;

reg          keydoneshift;
reg  [5:0]   LED;

`include "state.h"

always @( posedge clkln ) begin
    // keydoneの1クロック遅延
    if ( reset )
        keydoneshift <= 1'b0;
    else
        keydoneshift <= keydone;
    // 表示レジスタ
    if ( reset )
        LED <= 6'b0000000;
    else
        case ( state )
            HALT : LED <= 6'b0000000;
            OUT1 : LED <= DECODE(ranreg1);
            OUT2 : LED <= DECODE(ranreg2);
            OUT3 : LED <= DECODE(ranreg3);
            OUT4 : LED <= DECODE(ranreg4);
            OUT5 : LED <= DECODE(ranreg5);
            OUT6 : LED <= DECODE(ranreg6);
            IN1  : LED <= 6'b0000000;
            IN2  : LED <= DECODE(ranreg1);
            IN3  : if ( keydoneshift )
                    LED <= LED | DECODE(ranreg2);
            IN4  : if ( keydoneshift )
                    LED <= LED | DECODE(ranreg3);
            IN5  : if ( keydoneshift )
                    LED <= LED | DECODE(ranreg4);
            IN6  : if ( keydoneshift )
                    LED <= LED | DECODE(ranreg5);
            FAIL : if ( hz2l )
                    LED <= 6'b1111111;
                    else
                    LED <= 6'b0000000;
            DONE : LED <= 6'b0000001;
            OK   : if ( hz4 )
                    LED <= { LED[4:0], LED[5] };
            NEXTGAME:
                    LED <= 6'b0000000;
            SUCC : if ( hz2l )
                    LED <= 6'b1010101;
                    else
                    LED <= 6'b0101010;
        endcase
    end

    // 出力デコーダ
    function [5:0] DECODE;
    input[2:0] ranreg;
    case ( ranreg )
        3'd1 : DECODE = 6'b0000001;
        3'd2 : DECODE = 6'b0000010;
        3'd3 : DECODE = 6'b0000100;
        3'd4 : DECODE = 6'b0010000;
        3'd5 : DECODE = 6'b0100000;
        3'd6 : DECODE = 6'b1000000;
        default: DECODE = 6'bxxxxxxx;
    endcase
    endfunction
endmodule

```

〔リスト4.1〕 表示部 disp.v

なく、解答そのもののranregの値を用いています。入力状態にあるということは、それ以前の入力は正しかったこととなりますので、解答そのものを表示する方法でも結果的には同じことです。

(3) 不正解：FAILステート

2Hzで全ランプを点滅させます。秒カウンタ(secCnt)からのhz2l信号で、LEDに対し、すべて'1'か'0'の値を代入しています。hz2l信号は、パルス信号ではなくデューティ50%の信号ですので、2Hzの点滅を実現できます。

(4) 各問正解：DONE、OKステート

6個のランプ中1個の点灯ランプを移動し、1回転させます。DONEステートは、初期化のステートです。左上のランプ1個だけを点灯させるため、

```
LED <= 6'b0000001;
```

の代入を行っています。

OKステートでは、4Hzのhz4信号でレジスタLEDの左シフト(正確にはローテート)を行っています。シフト演算に記述上の注意点があることは、本連載の2回目で述べたとおりです。

結果的には、シフト演算は、接続を用いて1行で記述する、つまり、

```
LED <= { LED[4:0], LED[5] };
とするのがベストでしょう。あえてシフト演算子"<<"を用いて、
LED <= LED << 1;
LED[0] <= LED[5];
```

のように2行で書くことは、リスク大です。たとえば、この代入がブロッキング代入='だったとすると、記述の順番通りに実行されますので、LED[5]からLED[0]への移動が正しく行われません。LED[0]は常に'0'になってしまいます(とはいっても、今回の仕様では1回転するだけですから、シフト後LED[0]が'0'のままでもよいことになりますが)。

(5) 全問正解：SUCCステート

2Hzで6個中3個のランプを交互に点滅させます。FAILステートと同様に、2Hz、デューティ50%のhz2l信号を用いてLEDへ代入する値を切り替えています。

以上の部分で、ブロック図(図4.1)のセレクト、レジスタ入力生成部、レジスタ部が実現されています。

出力デコーダ部はfunctionで記述

出力デコーダは、3ビットのランプ番号を入力とし、6個のランプ中1個だけを点灯('1')とするためのデコード回路です。functionの中のcase文を用いて記述しています。ランプ番号1から6に対応して、戻り値DECODEに1ビットだけが'1'の6ビットの値を代入しています。

defaultラベルは、入力を取り得ない値になったときの処理です。たとえば、入力が'7'になったり、不定値になったときに、出力を不定値にすることで、誤動作していることを明確にすることができます。case文中のdefaultラベルは論理合成対象ですが、この場合、不定値を戻り値としているため、論理合成ツールはこの1行を無視します。

トップ・モジュール

図4.2に示すように、サイモンは7個のモジュールから構成されています。これらを接続し、外部からのクロック、キー入力、表示出力をそれぞれ必要なブロッ