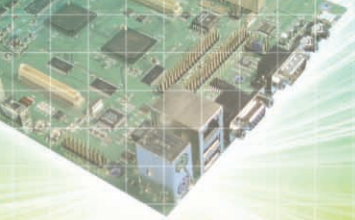

1

シフト・レジスタの基本とシリアル-パラレル変換技法

シリアルI/Oコントローラ設計入門

山武 一朗

マイコン周辺機能としてもっとも一般的かつ汎用的なものとして，シリアル・コントローラがある．ここではもっとも簡単な仕様のシリアル・コントローラを実現するまでを，シフト・レジスタを使ったシリアル-パラレル変換回路で，具体的に順を追って詳しく解説する．（編集部）

1 シリアル・コントローラ設計の前準備

● シリアル・コントローラ＝シフト・レジスタによるシリアル-パラレル変換回路

シリアル・コントローラとは，複数のビットからなる一つの情報（パラレルとして保持）を，1本の信号線を使って相手に送信（シリアル伝送）したり，逆に1本の信号線で送られてきたデータを，元の複数のビットからなる一つの情報に戻す処理をする機能をもった，マイコン周辺機能としてはもっとも一般的な汎用のコントローラです．

そのシリアル・コントローラの中身を一口で言えば，シフト・レジスタによるシリアル-パラレル変換回路であるといえます．

ここでは簡単な仕様のシリアル・コントローラを，もっとも基本的な機能を実装してその動作を確認し，少しずつ機能を追加していきながら，最後にはRS-232-Cにも使われている調歩同期式シリアル通信に対応させるまでを，本章と次章にわたって詳しく解説します．

なお，ここでのコントローラのHDL記述にはVHDLとVerilog-HDLの両方で設計しますが，本文の説明ではおもにVHDL版を使います．VHDLのソースとVerilog-HDLのソースは同じ構造になるように記述しているので，照らし合わせて見てください．VHDLおよびVerilog-HDLの基本的な言語仕様などについては詳しく説明しません．HDLの入門書などを参照してください．

● 8ビットLEDとプッシュ・ボタンを装備したFPGA評価ボードを使用

動作を試しながら学ぶということから，お手元にFPGA評価ボードを用意してください．FPGAはどこのメーカのどのようなシリーズのものでも問題ありません．ただし，動作確認をしたらさらに機能を追加して動作確認…という手順を繰り返すので，何度でも回路データをダウンロードしてテストできる必要があります．そのため，1度しか書き込みのできないデバイスやフラッシュ・メモリ・ベースのCPLD/FPGAは，学習用としては適していません．何度でもダウンロードが可能なSRAMベースのFPGAを選択してください．

また，評価ボードとして必要な機能としては，シリアル・コントローラの心臓部でもあるシフト・レジスタの動作を，目で見て理解できるようにするため，複数個のLEDとプッシュ・ボタンを装備したFPGA評価ボードを用意してください．さらに時間経過をカウントするためにクロックも必要です．そして最終的にRS-232-Cにつないで通信するところまでをテストしたい場合には，RS-232-Cドライバ ICやD-Subコネクタまでを装備したFPGA評価ボードがよいでしょう．

複数個のLEDやプッシュ・ボタン，クロック，そしてRS-232-Cが付いたFPGA評価ボードは，もっとも一般的な仕様なので，さまざまな評価ボードの中から入手が容易で安価なものを選択してください．

● 使用するFPGA評価ボードの入出力仕様の確認

ここに，種類の異なる2枚のFPGA評価ボードがあるとします．8ビットLEDとプッシュ・ボタンは，どちらも同様にボード上に装備されています．しかし回路構成上，FPGAから見たLEDやボタンの動作が異なっている場合があります．

図1に示すようにFPGAからLEDを点灯制御するには，“L”レベルを出力すると点灯するものと，“H”レベルを出力すると点灯するものの2通りがあります．スイッチも同様で，スイッチをONにするとFPGAに“L”レベルが入力されるものと，“H”レベルが入力されるものがあります．“L”レベルを出力して点灯するLEDや，スイッチがON状態で“L”レベルが入力される回路を，負論理回路と呼びます．用意した評価ボードがどちらであるかは，回路図を見たり実際にテスト回路を動作させるなどして確認しておいてください．

ちなみに，なぜ入出力回路に負論理を採用することが多いのかについては，歴史的にもそれなりに根拠があるのですが，こ

KEYWORD ―シリアル・コントローラ，シフト・レジスタ，シリアル-パラレル変換，ビット・シフト

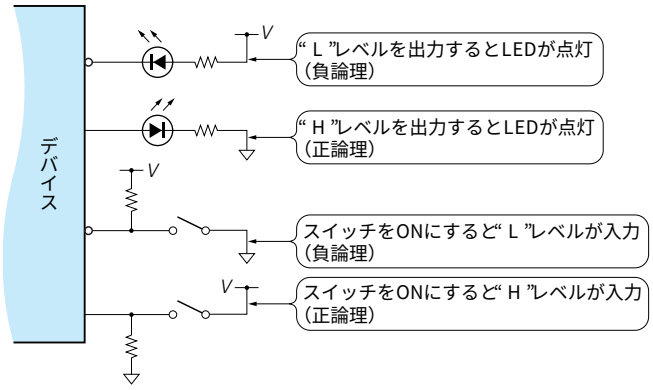

図1　正論理と負論理の違い

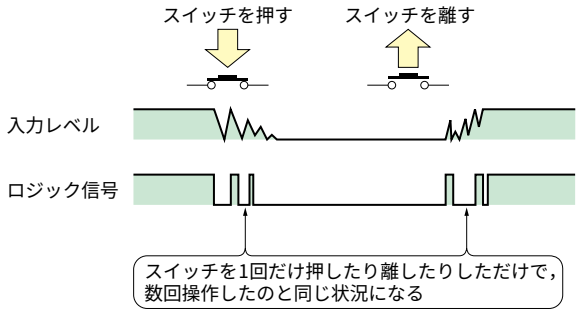

図2　スイッチ入力のチャタリング

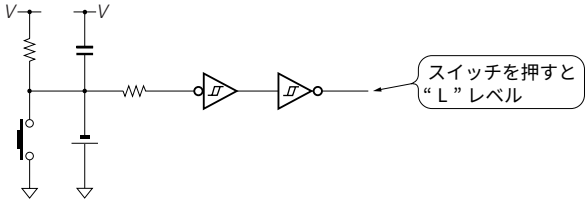

図3　チャタリング除去回路の例

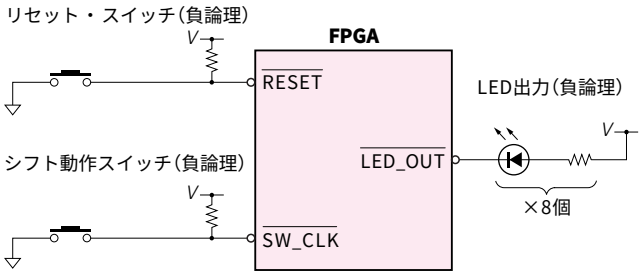

図4　リスト1のFPGA外部の回路例

こでは説明を省略します．ディジタル回路設計の入門書などを参照してください．

● スイッチのチャタリング除去回路を入れよう

評価ボードによっては，プッシュ・ボタンの入力がそのまま直接FPGAに接続されている場合もあります．その場合，スイッチを押したり離した瞬間にチャタリングと呼ばれる現象が発生し，1度しかスイッチを押していないにもかかわらず，回路としては数回のスイッチ入力があったかのような動作をしてしまう場合があります(**図2**)．

使用する評価ボードのスイッチ入力部に，**図3**のようなチャタリング除去回路が付いていない場合は，コラム1に示すような，ロジック回路によるチャタリング除去回路が必要になります．後述するシフト動作タイミングを示すスイッチ入力には，チャンタリングを除去したノイズのないクロックが必要です．

2 シフト・レジスタとシリアル-パラレル変換の回路

● パラレル→シリアル変換の動作

リスト1にシフト・レジスタによるパラレル→シリアル変換のHDLソースを，**図4**にFPGAの外部回路の例を示します．**リスト1**はLED点灯出力やスイッチ入力が負論理の場合の例です．評価ボードの回路構成の違いにより，負論理/正論理の違いがある場合は，それぞれの仕様に合わせてコメント部分を変更してください．**リスト1**では8ビットLED出力と，リセット・スイッチおよびシフト動作タイミングとして使うクロック・スイッチの2個のスイッチを必要とします．

リスト1の内容について説明します．まずentityで，FPGAの外につなげる信号を定義します(**リスト1**のⒶ)．リセット信号としてRESETを，8ビットLEDの点灯出力用としてLED_OUTを，そしてシフト動作クロック信号としてSW_CLKを定義します．

さらにFPGAの内部に必要なレジスタを定義します(Ⓑ)．ここではSHIFT_REGという8ビットのレジスタを定義し，このレジスタで8ビット・データを保持します．

実際の回路はbeginの次の行から記述します．Ⓒのprocess文がシフト・レジスタの処理をしている部分です．まずリセット・スイッチが押されると，SHIFT_REGには“10101010”というビット・パターンのデータが代入されます(Ⓓ)．するとprocess文の外のⒺでは，SHIFT_REGの値をLED_OUTという信号に出力しています．process文の外は常時並列に回路が動作するので，リセット時にSHIFT_REGに値を代入すると，それと同時にLED_OUTの出力信号もSHIFT_REGに代入した値に変わります．ソフトウェアの世界からハードウェアの世界に入門したばかりの頃は，この「同時に動作する」という点をつねに頭に入れて，それを意識しながらHDLソースを見てください．

リスト1を論理合成/配置配線し，FPGA用のコンフィグレーション・データを作成して，評価ボードにダウンロードしてください．リセット・スイッチを押して，LEDの点灯パターンがSHIFT_REGに代入した値のビット・パターンと同じになっているでしょうか？

● レジスタSHIFT_REGのシフト動作

次にシフト・レジスタをシフト動作させてみます．シフト動作スイッチを押すと，SW_CLK信号が‘1’になります．Ⓕの部

Column 1　ロジック回路によるチャタリング除去の例

図2でわかるように，スイッチ入力は安定するまでに数ms～数十ms程度の時間がかかります．そこでこの時間が経過するのを待ってからスイッチの状態を入力すれば，安定したスイッチ入力が可能です．

リストAと**リストB**にロジック回路によるチャタリング除去の例を示します．チャタリング除去の対象となるスイッチ入力と，チャタリングを除去したスイッチ状態出力のほかに，時間経過をカウントするためのクロック入力と，関連するカウンタのクリアのためにリセット入力も必要です．

通常はスイッチ入力有効期間として，スイッチの状態を常時監視します．SW_REG1とSW_REG0の二つの信号は，スイッチ入力であるSW_INの状態を保持した変数で，SW_REG1はSW_REG0より1クロック前の状態を保持しています．よって，この二つの信号を比較し状態が異なる場合は，スイッチが押された，または離されたことを検出できます．スイッチの状態が変化したことを検出したら，スイッチ入力が安定するまでの時間をカウントするCLK_COUNTERをクリアし，またスイッチ入力を無視するためのマスク信号SW_MASKを‘1’にセットします．

スイッチ入力無効期間では，時間経過をカウントするCLK_COUNTERの上位ビットを調べ，規定した時間が経過したかどうかを判定します．まだ規定時間が経過していない場合は，CLK_COUNTERをインクリメントします．もし規定時間が経過した場合は，スイッチ入力マスクをクリアし，スイッチ入力判定結果を保持する信号SW_OUT_REGに，安定したスイッチ入力の状態を代入します．

process文の外では，SW_OUT_REGの値をSW_OUT信号に代入しているので，チャタリングを除去したきれいなスイッチ入力を，ここで出力します．

時間経過をカウントするクロックとしては，33MHzのクロックを使用する場合は20数ビット程度のカウンタを使うことで，数ms～十数msの待ち時間をカウントすることができます．使用するスイッチの特性によっては，チャタリング発生期間が異なります．タクト・スイッチの類は比較的時間が短いようですが，内部が機械式バネのスイッチでは，比較的チャタリング発生期間が長いようです．

リスト1　シフト・レジスタによるLED点灯出力

```
-- ********************************************************
-- シフト・レジスタ出力のテスト
-- ********************************************************

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity SHIFT_OUT is
    port (

        RESET     : in      std_logic;
        LED_OUT   : out     std_logic_vector(7 downto 0);
        SW_CLK    : in      std_logic

    );
end SHIFT_OUT;

architecture RTL of SHIFT_OUT is

    signal SHIFT_REG    : std_logic_vector(7 downto 0);

begin

process (RESET, SW_CLK)
begin
--  if (RESET = '1') then    -- 正論理の場合
    if (RESET = '0') then    -- 負論理の場合

        SHIFT_REG <= "10101010";

--  elsif (SW_CLK'event and SW_CLK = '1') then   -- 正論理の場合
    elsif (SW_CLK'event and SW_CLK = '0') then   -- 負論理の場合

        SHIFT_REG(7) <= '0';
        SHIFT_REG(6 downto 0) <= SHIFT_REG(7 downto 1);

    end if;
end process;

--  LED_OUT <=      SHIFT_REG;     -- 正論理の場合
    LED_OUT <= not SHIFT_REG;      -- 負論理の場合

end RTL;
```

入出力信号の定義
FPGA内部で保持するシフト・レジスタ定義
シフト・レジスタ処理
リセット時代入
シフト動作
クロック・イベント文
LED点灯出力

（a）VHDL版

```
// ********************************************************
// シフト・レジスタ出力のテスト
// ********************************************************

module SHIFT_OUT
    (

        input               RESET    ,
        output    [7 : 0]   LED_OUT  ,
        input               SW_CLK

    );
    reg  [7:0]     SHIFT_REG;

//always @(posedge SW_CLK or posedge RESET)
                                    // 正論理の場合
always @(negedge SW_CLK or negedge RESET)
                                    // 負論理の場合
begin
//  if (RESET == 1'b1) begin// 正論理の場合
    if (RESET == 1'b0) begin// 負論理の場合

        SHIFT_REG <= 8'b10101010;

    end else begin

        SHIFT_REG[7] <= 1'b0;
        SHIFT_REG[6 : 0] <= SHIFT_REG[7 : 1];

    end
end

//  assign    LED_OUT   =  SHIFT_REG;  // 正論理の場合
    assign    LED_OUT   = ~SHIFT_REG;  // 負論理の場合

endmodule
```

入出力信号の定義
シフト・レジスタ処理
FPGA内部で保持するシフト・レジスタ定義
クロック・エッジで動作
リセット時代入
シフト動作
LED点灯出力

（b）Verilog-HDL版

New Products ── アダプテック，SAS製品を続々発売
アダプテックジャパン(株)は，SAS/SATAドライブのインテリジェント・エンクロージャ「Storage Enclosure 335SAS」，SAS RAIDコントローラ「Serial Attached SCSI RAID 4800SAS/4805SAS」，「Serial Attached SCSI 48300」を発売した．

リストA　ロジック回路によるチャタリング除去回路の例（VHDL版）

```
-- ********************************************************
-- スイッチ入力のチャタリング除去
-- ********************************************************

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity SWIN_CTRL is
    port (

        RESET          : in      std_logic;
        CLOCK          : in      std_logic;
        SW_IN          : in      std_logic;
        SW_OUT         : out     std_logic

    );
end SWIN_CTRL;

architecture RTL of SWIN_CTRL is

    signal CLK_COUNT    : std_logic_vector(20 downto 0);
    signal SW_REG0      : std_logic;
    signal SW_REG1      : std_logic;
    signal SW_OUT_REG   : std_logic;
    signal SW_MASK      : std_logic;

begin

-- ***** 入力スイッチ状態判定 ***** --
process(RESET, CLOCK)
begin

--  if (RESET = "1") then    -- 正論理の場合
    if (RESET = '0') then    -- 負論理の場合

        CLK_COUNT <= (others => '0') ;
        SW_REG1        <= '0';
        SW_REG0        <= '0';
        SW_OUT_REG     <= '0';
        SW_MASK        <= '0';

    elsif (CLOCK'event and CLOCK = '1') then

        if (SW_MASK = '0') then       -- スイッチ入力有効期間
            if (SW_REG1 /= SW_REG0) then
                                -- 前回と比較してスイッチ状態が変化した
                SW_MASK   <= '1';    -- スイッチ入力無効期間
                CLK_COUNT <= (others => '0');
            end if;

        else -- スイッチ入力無効期間
            if (CLK_COUNT(20) = '1') then
                                        -- 規定時間を経過した
                SW_MASK <= '0';      -- スイッチ入力有効期間
                SW_OUT_REG     <= SW_REG1;
                                        -- スイッチ入力状態確定
            else
                CLK_COUNT <= CLK_COUNT + '1';
                                        -- 経過時間カウント
            end if;
        end if;

        SW_REG1 <= SW_REG0;           -- 現在のスイッチ状態を保存
    --  SW_REG0 <=     SW_IN;         -- 正論理の場合
        SW_REG0 <= not SW_IN;         -- 負論理の場合

    end if;
end process;

    SW_OUT <= SW_OUT_REG;
                          -- チャタリング除去したスイッチ状態を出力

end RTL;
```

リストB　ロジック回路によるチャタリング除去回路の例（Verilog-HDL版）

```
// ********************************************************
// スイッチ入力のチャタリング除去
// ********************************************************

module SWIN_CTRL
    (

        input               RESET     ,
        input               CLOCK     ,
        input               SW_IN     ,
        output              SW_OUT

    );

    reg  [20: 0]   CLK_COUNT;
    reg            SW_REG0;
    reg            SW_REG1;
    reg            SW_OUT_REG;
    reg            SW_MASK;

//always @(posedge CLOCK or posedge RESET) // 正論理の場合
always @(posedge CLOCK or negedge RESET)   // 負論理の場合
begin
//  if (RESET == 1'b1) begin          // 正論理の場合
    if (RESET == 1'b0) begin          // 負論理の場合

        CLK_COUNT      <= 20'h00000;
        SW_REG1        <= 1'b0;
        SW_REG0        <= 1'b0;
        SW_OUT_REG     <= 1'b0;
        SW_MASK        <= 1'b0;

    end else begin

        if (SW_MASK == 1'b0) begin   // スイッチ入力有効期間
            if (SW_REG1 != SW_REG0) begin
                        // 前回と比較してスイッチ状態が変化した
                SW_MASK   <= 1'b1;  // スイッチ入力無効期間
                CLK_COUNT <= 20'h00000;
            end

        end else begin // スイッチ入力無効期間
            if (CLK_COUNT[20] == 1'b1) begin
                                    // 規定時間を経過した
                SW_MASK <= 1'b0; // スイッチ入力有効期間
                SW_OUT_REG <= SW_REG1;
                                    // スイッチ入力状態確定
            end else begin
                CLK_COUNT <= CLK_COUNT + 1;
                                        // 経過時間カウント
            end
        end

        SW_REG1 <= SW_REG0;      // 現在のスイッチ状態を保存
    //  SW_REG0 <=  SW_IN;       // 正論理の場合
        SW_REG0 <= ~SW_IN;       // 負論理の場合

    end
end

    assign    SW_OUT = SW_OUT_REG;
                      // チャタリング除去したスイッチ状態を出力

endmodule
```


New Products ―― サン，「Sun Java Studio Enterprise 8」を無償で提供
サン・マイクロシステムズ（株）は，Javaアプリケーション統合開発環境「Sun Java Studio Enterprise 8」をリリースし，無償での提供を開始した．本ツールは，NetBeans 4.1がベースとなって開発されている．対象となるOSは，Solarisと Windows．

Column 2
評価ボードに8ビットのディップ・スイッチがある場合

リスト1ではリセット時にHDLソース中に記述した固定値をSHIFT_REGに代入しています．もし使用するFPGA評価ボードに8ビットのディップ・スイッチが実装されている場合は，それを使ってリセット時の初期値を決めることも可能です．

リストCにリセット時の初期値をディップ・スイッチで設定可能にする例を示します．これにより，HDLソース・コードを変更して，論理合成や配置配線，さらにはダウンロードまでをやり直さなくても，ディップ・スイッチの状態を変更してからリセット・スイッチを押せば，LEDの点灯ビット・パターンをすぐに変更することができます．

リストC　リセット時の初期値をディップ・スイッチで設定可能にする例（VHDL版）

```
～中略～

entity SHIFT_OUT_b is
    port (

        RESET     : in  std_logic;
        LED_OUT   : out std_logic_vector(7 downto 0);
        DIP_SW    : in  std_logic_vector(7 downto 0);
        SW_CLK    : in  std_logic

    );
end SHIFT_OUT_b;

～中略～

process (RESET, DIP_SW, SW_CLK)
begin
--  if (RESET = '1') then    -- 正論理の場合
    if (RESET = '0') then    -- 負論理の場合

        SHIFT_REG <= DIP_SW;

--  elsif (SW_CLK'event and SW_CLK = '0') then -- 正論理の場合
    elsif (SW_CLK'event and SW_CLK = '1') then -- 負論理の場合

～中略～
```

ディップ・スイッチ入力を追加

リセット時にディップ・スイッチの内容をシフト・レジスタに代入

分は，SW_CLK信号が‘1’から‘0’になった立ち下がりのタイミングで，elsif文の内側が実行されることを示しています．

そしてSW_CLK信号の立ち下がりで動作する回路がⒼの部分です．まずSHIFT_REGの最上位ビットである7ビットに‘0’を代入しています．またすぐ下の行では，SHIFT_REGのビット7～1を同じレジスタSHIFT_REGのビット6～0に代入しています．同じレジスタから同じレジスタへ，ビット位置をずらして代入する動作がシフト動作になります．この例ではビット位置が1ビットだけずれていますが，一度に複数のビットをずらすことも可能です．もちろん移動元と移動先のビット幅が一致していなければ，論理合成時にエラーになります．

ここで，ハードウェアやHDLの初心者は，先にSHIFT_REGのビット7にゼロを代入してしまったら，そのすぐ下の行で行っているシフト動作をしても，SHIFT_REGのビット6までがゼロになってしまうと思う人もいるかもしれません．**リスト1**は一見プログラミング言語のようなソースなので，ソフトウェア畑の人はどうしてもそのような先入観を持ってしまいますが，これはれっきとしたハードウェアです．

```
SHIFT_REG(7) <= '0';
```

の行も，その下の，

```
SHIFT_REG(6 downto 0) <=
                        SHIFT_REG(7 downto 1);
```

の行も，SW_CLK信号の立ち下がりのタイミングで同時に並列に動作するのです．つまり結果的に，レジスタSHIFT_REGの内容が全体として下位ビット側に1ビットだけシフトし，最上位であるビット7はゼロになります．ちなみにシフトの直前までビット0に入っていた情報は，このシフト動作で消えてなくなります．

elsif文の内側の2行が同時に実行されるのとさらに同時に，process文の外にあるLED_OUTの出力部分も動作するので，最終的にはシフト動作スイッチを押すと，8ビットLEDの点灯パターンが1個ずつ下位ビット側にずれて表示される動作になります．またビット7にはゼロが代入されるので，その部分はLEDの点灯が消えていきます．

● LED_OUTのビット0に注目して動作を確認

ここで，LED_OUTのビット0だけに注目してみましょう．リセット時はSHIFT_REGに代入した値のビット0が反転して代入されて‘1’になります．シフト動作スイッチを押すと1ビット下位側にシフトして，ビット0は‘0’になります．さらにシフト動作スイッチを押すともう1ビット下位側にシフトして，今度はビット0は‘1’になります．つまり，リセット時にSHIFT_REGに代入した値が，シフト動作を行うたびに下位ビットから1ビットずつ出力される動作になります．これがパラレル→シリアル変換の基本動作となります．以上のシフト動作のようすを**図5**に示します．

シフト動作スイッチを8回押すと，すべてのLEDの点灯が消えることになります．ここでリセット・スイッチを押すと，レジスタSHIFT_REGにまた新しくデータが代入されるので，再度LEDが点灯し始めます．またシフト動作スイッチを押した回数が8回未満のときにリセット・スイッチを押しても，LEDの点灯状態はリセット時の状態に戻ります．

リスト1に記述しているprocess文は，先にあるif文でRESET信号の状態を調べ，次にあるif文（VHDLではelsifと記述）でSW_CLK信号の状態を調べることになるので，もし同時に信号が入力されても（ここでのSW_CLK信号はクロック・イベント文として記述しているので，SW_CLK信号の立ち上が