2013 年 6 月 26 日

FPGA による 24 時間時計回路の設計

通信処理ネットワーク研究室
10ec062　志村　貴大

1. まえがき

今回、24 時間時計回路の設計を行った理由は、 FPGA を用いた論理回路設計の基礎を学ぶにあたり、ハード及びソフト双方の基本技術を一度に習得できる題材であると推測したためである。24 時間時計を構成するモジュールの設計を終えた今、 その推測は正しかったものと自負している。

本レポートは、復習を兼ねた制作記録としてだけではなく、自分と同じ回路設計初心者が学習の参考にできるものにしたいと考えている。モジュール毎に簡単な解説を交えながら書き進めるつもりなので、興味があれば御一読願いたい。

目次

2. 製作物の仕様

2-1. 使用ボードについて

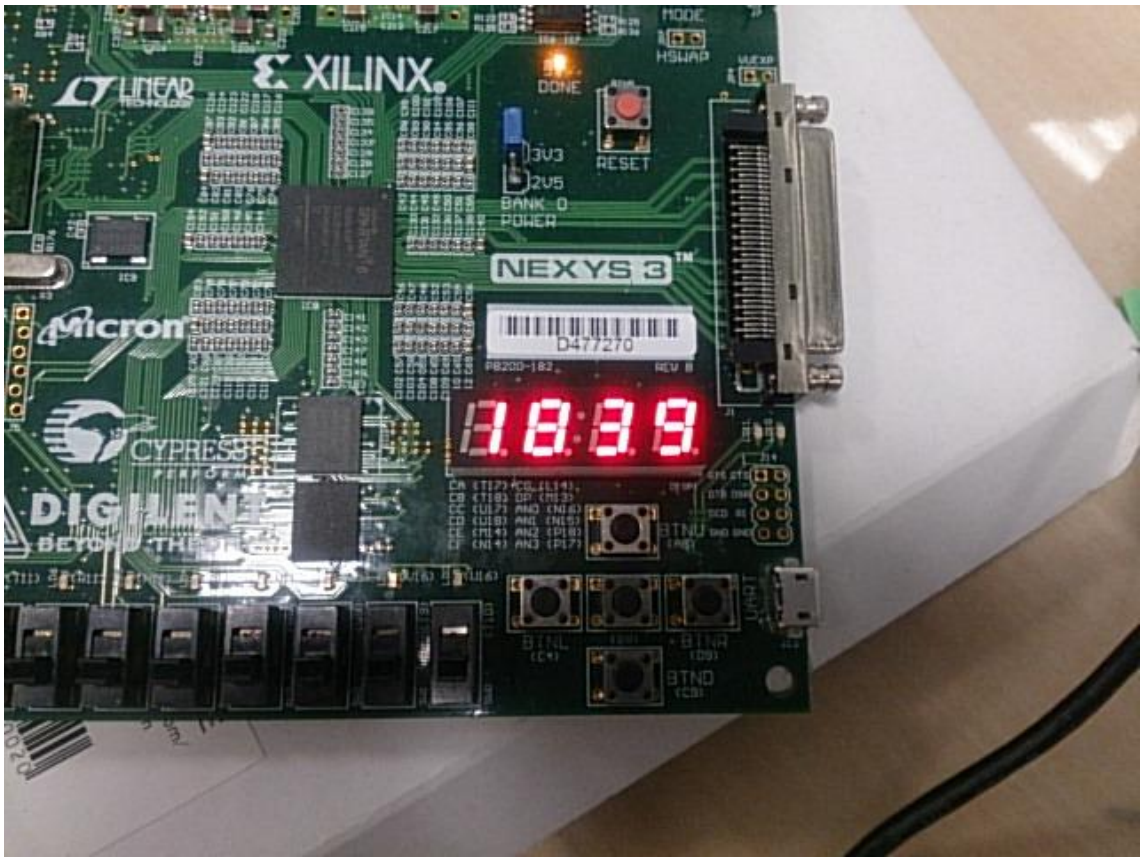
今回製作した 24 時間時計(以降、製作物と表記)を構成する論理回路は、Xilinx 製 FPGA ボード「NEXYS3」を用いて設計した。使用した言語は Verilog である。また、時計の表示部である 4 つの 7 セグメント LED、時間設定に用いる 5 つのボタン、モード(後述)切り替えに用いるスイッチは「NEXYS3」に搭載されているものを使用した。

2-2. 機能説明

製作物に実装した機能を以下に示す。

① 「24 時間時計表示」

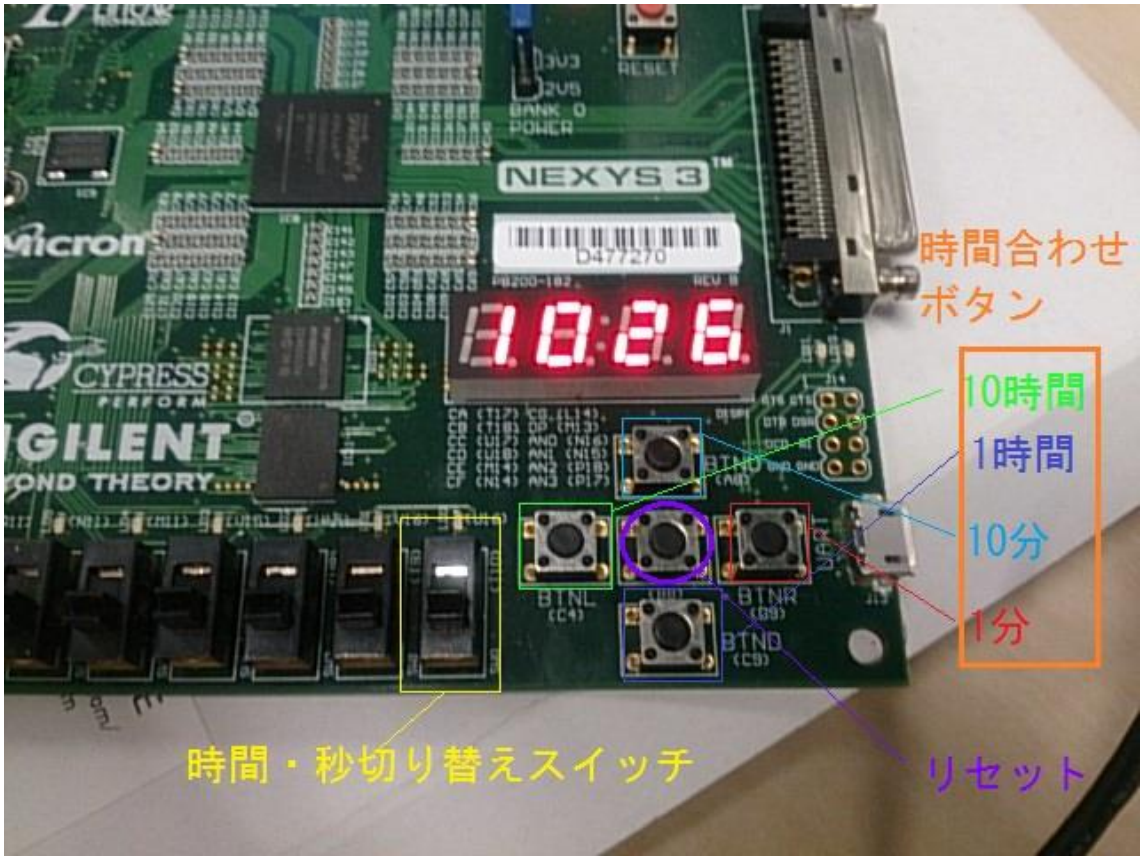
カウンタによって時間をカウントし、00:00～23:59 までの時間を、 7 セグメント LED で表示する。



(図.1 NEXYS3 上で動作する 24 時間時計　18:39 を示している。)

② 「時間設定機能」

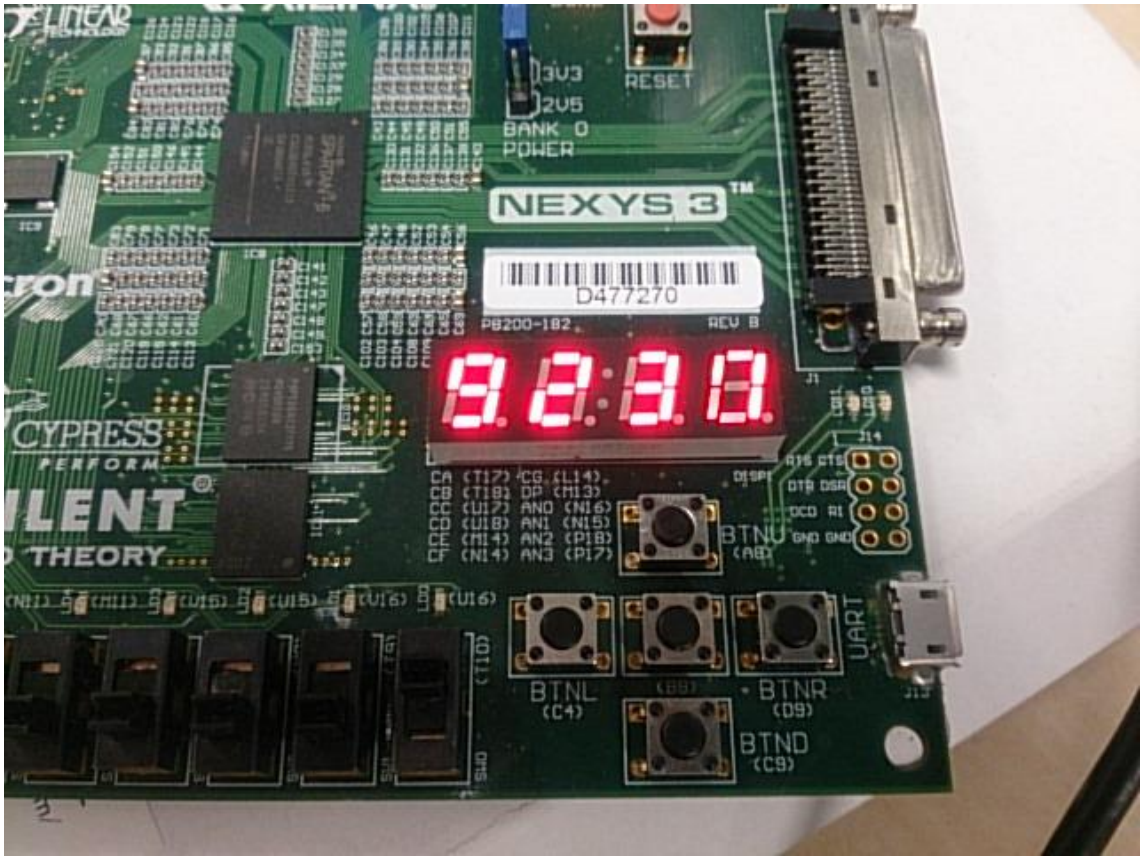
10 時間、1 時間、 10 分、 1 分の桁毎に時間の設定を行えるボタンを設けた。 ボタンを押した瞬間に、対応する桁の時間を 1 増加させる(押しっぱなしでは増加させない)。また、10 時間から 0.1 秒全ての時間カウンタを 0 にリセットするボタンも設けた。



(図.2 各種ボタン、スイッチ)

③ 「24 時間表示モード、 分・秒表示モードの切り替え」

7 セグメント LED の表示モードを切り替えるスイッチを設けた。 スイッチが OFF の時は 24 時間表示モード、ON の時は分・秒表示モードに切り替わる。分・秒表示モードでは、左から 1 分 10 秒 1 秒 0.1 秒に対応した表示になる。



(図.3 分・秒表示モード 上図は 9 分 23.7 秒を示している)

3. モジュールの説明

3-1.モジュールの全体像

製作物の論理回路は、以下に示す三つのモジュールで構成されている。

① 「カウンタ(トップモジュール)」

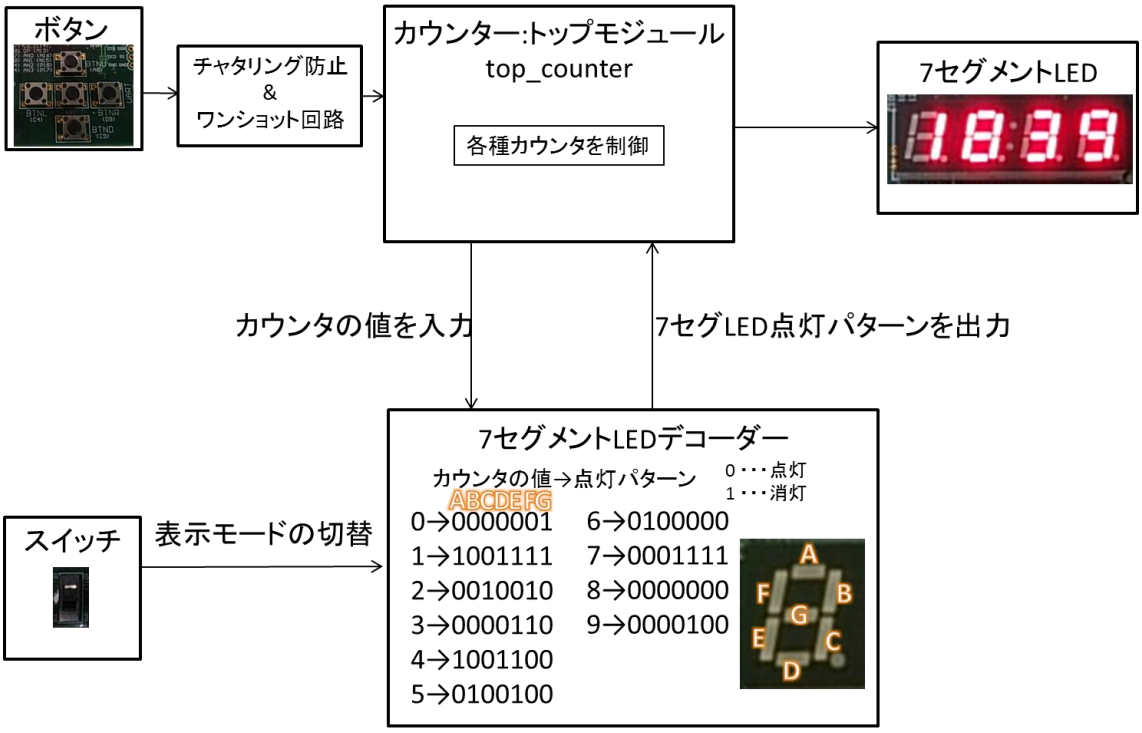
0.1 秒の桁のカウンタ～10 時間の桁のカウンタを持ち、それらの値を操作するモジュール。トップモジュールの機能としては、ボタン、スイッチ、クロック等のインプット信号を他のモジュールに渡す機能と、後述の 7 セグメント LED デコーダーより渡されたビット列を元に 7 セグメント LED を点灯させる機能をもつ。

② 「7 セグメント LED デコーダー」

トップモジュールから送られる各種カウンタの値を元に、 7 セグメント LED の点灯パターンのデータを出力するモジュール。点灯パターンのデータはトップモジュールに渡す。

③ 「チャタリング除去及びワンショット回路」

トップモジュールから受け取ったボタンのインプット信号を元に、チャタリングの除去及びワンショット処理を行う。



(図.4 モジュールの全体像)

3-2. カウンタ(トップモジュール)の解説

3-2-1.各桁のカウンタ

トップモジュールは、0.1 秒～10 時間の桁を表現するカウンタを持っている。そのため、最少の単位である 0.1 秒を表現するため、0.1 秒オーダーで立ち上がるイネーブル信号を生成する必要がある。以下は、0.1 秒オーダーイネーブル信号を生成する回路設計文である。

・0.1 秒オーダーイネーブル信号生成回路

```
reg [23:0]count;
parameter SEC0_1_MAX = 10000000;

always@(posedge clk or posedge reset)
begin
        if(reset==1'b1)
                        count <= 24'd0;
        else if(ENABLE_0_1s == 1'b1)
                        count <= 24'd0;
        else
                        count <= count + 24'd1;
end

assign ENABLE_0_1s = (count == (SEC0_1_MAX-1))? 1'b1 : 1'b0;
```

解説 : NEXYS3 に搭載されている水晶振動子の周波数は 100MHz である。水晶振動子のクロック clk が立ち上がる度、レジスタ count に 1 を加算していく。count の値が 10 進数 9999999(0 からのカウントのため 10000000-1 の値を用いる)となったとき、0.1 秒が経過したことを意味するイネーブル信号 ENABLE_0_1s を立ち上げにし、count の値を 0 に戻す。

また、0.1 秒の桁を表現するカウンタの設計文は次のようになる。

・0.1 秒の桁のカウンタ回路

```
always@(posedge clk or posedge reset)
begin
        if(reset==1'b1)
                        CNT0_1s <= 4'd0;
        else if(ENABLE_0_1s==1'b1)
                        CNT0_1s <= (CNT0_1s == 4'd9)? 4'd0 : (CNT0_1s + 4'd1);
end
assign ENABLE_1s = (ENABLE_0_1s && CNT0_1s == 4'd9)? 1'b1 : 1'b0;
```

解説：イネーブル ENABLE_0_1s が立ち上がる度、レジスタ CNT0_1s の値が桁の最大値である 10 進数 9 かどうかを判定する。真であった場合、CNT0_1s の値を 0 にする。偽であった場合、CNT0_1s の値に 1 を加算する。ENABLE_0_1s が立ち上がっており、かつ CNT0_1s の値が 10 進 9 の場合、1 秒が経過したことを表現するため ENABLE_1s を立ち上げにする。

1～10 時間のカウンタ回路も、0.1 秒の桁のカウンタ回路と同様の形でカウントアップ、リセット、イネーブル信号の立ち上げを行う。

3-2-2. 時間合わせ処理

1 分～10 時間の桁のカウンタには、時間合わせを行う際の処理を記述した。
以下に、10 時間の桁のカウンタの回路設計文を例として示す。

・10 時間の桁のカウンタ回路

```
always@(posedge clk or posedge reset)
begin
        if(reset == 1'b1)
                CNT10h <= 2'd0;
        else if(o_bt10h == 1'b1)
                CNT10h <= (CNT10h == 2'd2 | |
                        (CNT10h==2'd1 && CNT1h>4'd3))? 2'd0:(CNT10h + 2'd1);
        else if(ENABLE_10h == 1'b1)
                  CNT10h <= (CNT10h == 2'd2 && CNT1h == 4'd3 &&
                  CNT10m == 3'd5 && CNT1m == 4'd9 )? 2'd0 : (CNT10h+2'd1);
end
```

解説：チャタリング除去及びワンショット回路(後述)を通したボタン信号の立ち上がりを検出し、ボタンに対応する桁の値を増加させる。増加させた値が各桁の最大値になった場合は、桁の値を 0 にする。また、時計は 00:00～23:59 の範囲の時刻を表示させたい。よって、桁の値をボタンによって増加させたときに表示させたい範囲の数字に収まるよう処理する必要がある。例えば、1 時間の桁が 4 以上のとき、10 時間の桁は 2 に設定できなくするよう、条件(CNT10h==2'd1 && CNT1h>4'd3)を設定する。

他の桁の時間合わせの処理も、同様に記述した。

3-2-3. ダイナミック点灯用カウンタ

後述する 7 セグメント LED モジュールでのダイナミック点灯制御に使用するカウンタ。以下にその回路設計文を示す。

・ダイナミック点灯用カウンタ回路

```
reg [16:0]dcount;
always@(posedge clk)
begin
        if(ENABLE_kHz == 1'b1)
                        dcount <= 16'h0;
        else
                        dcount <= dcount + 16'h1;
end
assign ENABLE_kHz=(dcount == (kHz-1))? 1'b1 : 1'b0;

reg [1:0]AN_count;
always@(posedge clk)
begin
if(ENABLE_kHz == 1'b1)
                        AN_count <= AN_count + 2'h1;
end
```

解説：2 ビットのレジスタ AN_count に 0.001 秒オーダーで 1 を加える。0.001 秒オーダーイネーブル信号 ENABLE_kHz は、3-2-1 の「0.1 秒オーダーイネーブル信号生成回路」と同様の形で生成させる。

ENABLE_kHz は、後述のチャタリング除去回路においても再利用している。

3-3. 7segLED モジュールの解説

3-3-1. モジュールの構成

7 segLED モジュールは、以下の二つの回路で構成されている。

① デコーダー

入力されたカウンタの値に対応した 7 セグメント LED の点灯信号を出力する。

② ダイナミック点灯・モード切替回路

0.001 秒毎に点灯させる 7 セグメント LED 及び表示するカウンタの値を選択する。

3-3-2. デコーダーの解説

デコーダーの回路設計文を以下に示す。

・7segLED デコーダー回路

```
reg [3:0]CNT;
always@(CNT)
begin
                case(CNT)
                                //ABCDEFG
                0:LED <= 7'b0000001;
                1:LED <= 7'b1001111;
                2:LED <= 7'b0010010;
                3:LED <= 7'b0000110;
                4:LED <= 7'b1001100;
                5:LED <= 7'b0100100;
                6:LED <= 7'b0100000;
                7:LED <= 7'b0001111;
                8:LED <= 7'b0000000;
                9:LED <= 7'b0000100;
                default:LED <= 7'b0110000;
                endcase
end
```

解説：入力されたカウンタの値(CNT)に対応した 7 セグメント LED の点灯信号(LED)を出力する。カウンタの値と点灯信号の対応に関しては、図 4.を参照していただきたい。

3-3-3. ダイナミック点灯・モード切替回路の解説

ダイナミック点灯・モード切替回路の回路設計文を以下に示す。

・7segLED デコーダー回路

```
always@(AN_count,CNT0_1s,CNT1s,CNT10s,CNT1m,CNT10m,CNT1h,CNT10h,sw)
begin

case(AN_count[1:0])
                        2'd0:begin AN <= 4'b1110;end
                        2'd1:begin AN <= 4'b1101;end
                        2'd2:begin AN <= 4'b1011;end
                        2'd3:begin AN <= 4'b0111;end
                        default AN <= 4'b0000;
endcase

if(sw==1'b0)
begin
        case(AN_count[1:0])
                        2'd0:begin CNT <= CNT1m; end
                        2'd1:begin CNT <= {1'b0,CNT10m}; end
                        2'd2:begin CNT <= CNT1h;end
                        2'd3:begin CNT <= {2'b00,CNT10h};end
        endcase
end
else
begin
        case(AN_count[1:0])
                        2'd0:begin CNT <= CNT0_1s; end
                        2'd1:begin CNT <= CNT1s; end
                        2'd2:begin CNT <= {1'b0,CNT10s};end
                        2'd3:begin CNT <= {1'b0,CNT1m};end
        endcase
end

end

endmodule
```

解説：0.001 秒間隔で変化する AN_count の値(0～3)を用いて、ダイナミック点灯を実現した。設計文中の記述を口語的に言い換えると、点灯させる LED の位置を 0.001 秒刻みで右(AN <= 4'b1110)から左(AN <= 4'b0111)へずらす記述である、と言える。このように、高速で LED の点灯位置をずらし続け、あたかも複数の LED が同時に点灯しているかのように見せる点灯方式をダイナミック点灯と呼ぶ。

sw はモード切替用スイッチの入力信号である。sw が 0 の時は表示する桁の値を 24 時間時計のものにし、sw が 1 の時は分・秒表示のものにするよう設計した。なお、CNT0_1s, CNT1s, CNT10s ,CNT1m, CNT10m, CNT1h, CNT10h はそれぞれ 0.1 秒,1 秒,10 秒,1 分,10 分,1 時間,10 時間の桁の値を示す。

3-4.チャタリング除去及びワンショット回路モジュールの解説

3-4-1. モジュールの構成

このモジュールは以下の二つの回路で構成されている。

① チャタリング除去回路

ボタンを押した際に発生するチャタリングを除去する回路。チャタリングとはスイッチ接点の開閉が物理的要因によって繰り返される現象である。

② ワンショット回路

入力信号の立ち上がりの瞬間を検出する回路。つまり、ボタンを押した瞬間を検出する回路を指す。

3-4-2. チャタリング除去回路

チャタリング除去回路の回路設計文を以下に示す。

・チャタリング除去回路

```
always@(posedge clk)begin
        if(enable == 1'b1)
                shift <= {shift[3:0], bt};
end
assign flag = &shift;
```

解説：0.001 秒毎に、ボタン接点の開閉(0,1)を検出し、5 ビットのレズスタ shift に開閉の状態を蓄積していく。ビット列がすべて 1 で満たされた時(&shift が 1)、ボタンが押されたことを示す信号 flag を 1 にする。

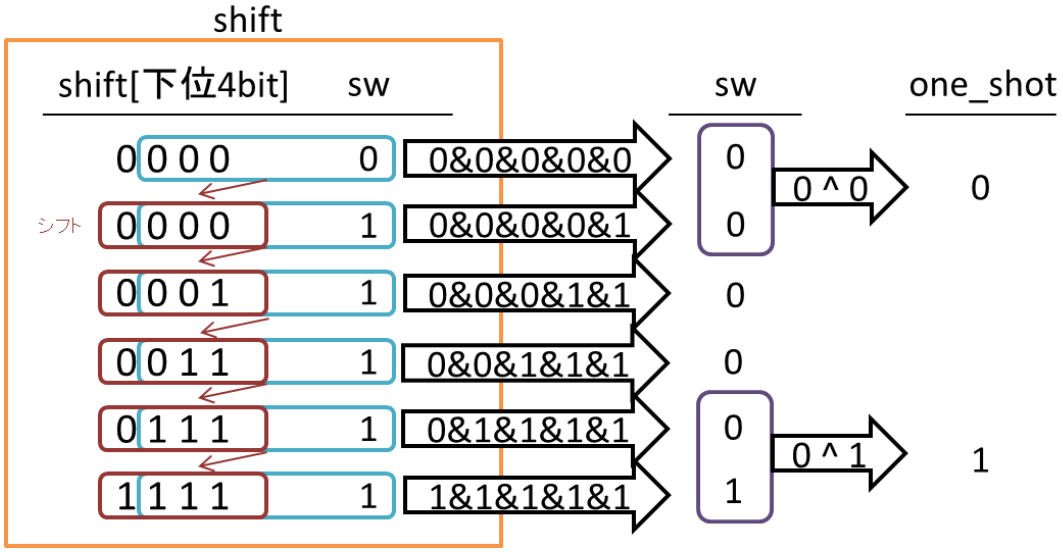
3-4-3. ワンショット回路

ワンショット回路の回路設計文を以下に示す。

・ワンショット回路

```
always@(posedge clk)begin
                btbt <= {btbt[0],flag};
end
assign one_shot = ~btbt[1] & btbt[0];
```

解説：3-4-2 のチャタリング除去回路で定義された信号 flag の状態を検出し、2 ビットのレジスタ btbt に蓄積していく。過去の flag の状態(btbt[1])と現在の flag の状態(btbt[0])を排他的論理和で比較(~btbt[1] & btbt[0])し、その結果を信号 one_shot としてトップモジュールに出力する。



(図.5 チャタリング除去及びワンショット回路モジュール ビットの返還)

4. 付録：回路設計文及び UCF ファイル

製作物の全回路設計文と UCF ファイルを以下に示す。

・top_counter.v(トップモジュール)

```
module top_counter(
input wire clk,
input wire reset,
input wire i_bt1m,
input wire i_bt10m,
input wire i_bt1h,
input wire i_bt10h,
input wire sw,
output wire [0:6]LED,
output wire [3:0]AN
);

reg [23:0]count;
reg [3:0]CNT0_1s;
reg [3:0]CNT1s;
reg [2:0]CNT10s;
reg [3:0]CNT1m;
reg [2:0]CNT10m;
reg [3:0]CNT1h;
reg [1:0]CNT10h;

parameter SEC0_1_MAX = 10000000; //0.1 秒
parameter kHz   = 100000;//0.001 秒

wire
ENABLE_0_1s,ENABLE_1s,ENABLE_10s,ENABLE_1m,ENABLE_10m,ENABLE_1h,
ENABLE_10h;//イネーブル信号
wire ENABLE_kHz;
```

```
/*----------------------------カウンタ-----------------------------------*/
/*0.1 秒の桁のカウンタ*/
always@(posedge clk or posedge reset)
begin
        if(reset==1'b1)
                        count <= 24'd0;
        else if(ENABLE_0_1s == 1'b1)
                        count <= 24'd0;
        else
                        count <= count + 24'd1;
end
assign ENABLE_0_1s = (count == (SEC0_1_MAX-1))? 1'b1 : 1'b0;

always@(posedge clk or posedge reset)
begin
        if(reset==1'b1)
                        CNT0_1s <= 4'd0;
        else if(ENABLE_0_1s==1'b1)
                        CNT0_1s <= (CNT0_1s == 4'd9)? 4'd0 : (CNT0_1s + 4'd1);
end
assign ENABLE_1s = (ENABLE_0_1s && CNT0_1s == 4'd9)? 1'b1 : 1'b0;
```

```
/*1 秒の桁のカウンタ*/
always@(posedge clk or posedge reset)
begin
                if(reset == 1'b1)
                                        CNT1s <= 4'd0;
                else if(ENABLE_1s == 1'b1)
                                        CNT1s <= (CNT1s == 4'd9)? 4'd0 :
(CNT1s+4'd1);
end
assign ENABLE_10s = (ENABLE_1s && CNT1s==4'd9)? 1'b1 : 1'b0;


/*10 秒の桁のカウンタ*/
always@(posedge clk or posedge reset)
begin
                if(reset == 1'b1)
                        CNT10s <= 3'd0;
                else if(ENABLE_10s == 1'b1)
                        CNT10s <= (CNT10s == 3'd5)? 3'd0 : (CNT10s+3'd1);
end
assign ENABLE_1m = (ENABLE_10s && CNT10s==3'd5)? 1'b1 : 1'b0;

/*1 分の桁のカウンタ*/
always@(posedge clk or posedge reset)
begin
                if(reset == 1'b1)
                        CNT1m <= 4'd0;
                else if(o_bt1m == 1'b1)
                        CNT1m <= (CNT1m == 4'd9)? 4'd0:(CNT1m + 4'd1);
                else if(ENABLE_1m == 1'b1)
                        CNT1m <= (CNT1m == 4'd9)? 4'd0 : (CNT1m+4'd1);
end
assign ENABLE_10m = (ENABLE_1m && CNT1m==4'd9)? 1'b1 : 1'b0;
```

```
/*10 分の桁のカウンタ*/
always@(posedge clk or posedge reset)
begin
        if(reset == 1'b1)
                CNT10m <= 3'd0;
        else if(o_bt10m == 1'b1)
                CNT10m <= (CNT10m == 3'd5)? 3'd0:(CNT10m + 3'd1);
        else if(ENABLE_10m == 1'b1)
                CNT10m <= (CNT10m == 3'd5)? 3'd0 : (CNT10m+3'd1);
end
assign ENABLE_1h = (ENABLE_10m &&
                        CNT10m==3'd5 && CNT1m==4'd9)? 1'b1 : 1'b0;

/*1 時間の桁のカウンタ*/
always@(posedge clk or posedge reset)
begin
        if(reset == 1'b1)
                CNT1h <= 4'd0;
        else if(o_bt1h == 1'b1)
                CNT1h <= (CNT1h == 4'd9||
                        (CNT10h==2'd2&&CNT1h==4'd3))? 4'd0 :(CNT1h+4'd1);
                else if(ENABLE_1h == 1'b1)
                CNT1h <= (CNT1h == 4'd9||
                        (CNT10h==2'd2&&CNT1h==4'd3))? 4'd0 :(CNT1h+4'd1);
end
assign ENABLE_10h = ((ENABLE_1h && CNT1h==4'd9)||
                        (ENABLE_1h && CNT10h==2'd2 && CNT1h==4'd3))? 1'b1 : 1'b0;
```

```
/*10 時間の桁のカウンタ*/
always@(posedge clk or posedge reset)
begin
        if(reset == 1'b1)
                CNT10h <= 2'd0;
        else if(o_bt10h == 1'b1)
                CNT10h <= (CNT10h == 2'd2 | |
(CNT10h==2'd1 &&CNT1h>4'd3))? 2'd0:(CNT10h + 2'd1);
                else if(ENABLE_10h == 1'b1)
                        CNT10h <= (CNT10h == 2'd2 &&
                                CNT1h == 4'd3 &&
                                CNT10m == 3'd5 &&
                                CNT1m == 4'd9 )? 2'd0 : (CNT10h+2'd1);
end
```

```
/*----------------------------------ダイナミック点灯用カウンタ--------------------------------------*/
reg [16:0]dcount;
always@(posedge clk)
begin
        if(ENABLE_kHz == 1'b1)
                        dcount <= 16'h0;
        else
                        dcount <= dcount + 16'h1;
end
assign ENABLE_kHz=(dcount == (kHz-1))? 1'b1 : 1'b0;

reg [1:0]AN_count;
always@(posedge clk)
begin
if(ENABLE_kHz == 1'b1)
                        AN_count <= AN_count + 2'h1;
end
/*--------------------------------ボタンのチャタリング除去-------------------------------------------*/
pos_one_shot i0(.clk(clk),.enable(ENABLE_kHz),.bt(i_bt1m),.one_shot(o_bt1m));
pos_one_shot i1(.clk(clk),.enable(ENABLE_kHz),.bt(i_bt10m),.one_shot(o_bt10m));
pos_one_shot i2(.clk(clk),.enable(ENABLE_kHz),.bt(i_bt1h),.one_shot(o_bt1h));
pos_one_shot i3(.clk(clk),.enable(ENABLE_kHz),.bt(i_bt10h),.one_shot(o_bt10h));
/*-------------------------------デコーダー-------------------------------------*/
seven_segment_LED d0(
.AN_count(AN_count),
.CNT0_1s(CNT0_1s),
.CNT1s(CNT1s),
.CNT10s(CNT10s),
.CNT1m(CNT1m),
.CNT10m(CNT10m),
.CNT1h(CNT1h),
.CNT10h(CNT10h),
.sw(sw),
.AN(AN),
.LED(LED));
endmodule
```

・7seg_LED.v(7 セグメント LED モジュール)

```
module seven_segment_LED(
input [1:0]AN_count,
input wire[3:0]CNT0_1s,
input wire[3:0]CNT1s,
input wire[2:0]CNT10s,
input wire[3:0]CNT1m,
input wire[2:0]CNT10m,
input wire[3:0]CNT1h,
input wire[1:0]CNT10h,
input wire sw,
output reg[3:0]AN,
output reg[0:6]LED
);

reg [3:0]CNT;
always@(CNT)
begin
                case(CNT)
                                //ABCDEFG
                0:LED <= 7'b0000001;
                1:LED <= 7'b1001111;
                2:LED <= 7'b0010010;
                3:LED <= 7'b0000110;
                4:LED <= 7'b1001100;
                5:LED <= 7'b0100100;
                6:LED <= 7'b0100000;
                7:LED <= 7'b0001111;
                8:LED <= 7'b0000000;
                9:LED <= 7'b0000100;
                default:LED <= 7'b0110000;
                endcase
end
```

```
always@(AN_count,CNT0_1s,CNT1s,CNT10s,CNT1m,CNT10m,CNT1h,CNT10h,sw)
begin

case(AN_count[1:0])
                        2'd0:begin AN <= 4'b1110;end
                        2'd1:begin AN <= 4'b1101;end
                        2'd2:begin AN <= 4'b1011;end
                        2'd3:begin AN <= 4'b0111;end
                        default AN <= 4'b0000;
endcase

if(sw==1'b0)
begin
        case(AN_count[1:0])
                        2'd0:begin CNT <= CNT1m; end
                        2'd1:begin CNT <= {1'b0,CNT10m}; end
                        2'd2:begin CNT <= CNT1h;end
                        2'd3:begin CNT <= {2'b00,CNT10h};end
        endcase
end
else
begin
        case(AN_count[1:0])
                        2'd0:begin CNT <= CNT0_1s; end
                        2'd1:begin CNT <= CNT1s; end
                        2'd2:begin CNT <= {1'b0,CNT10s};end
                        2'd3:begin CNT <= {1'b0,CNT1m};end
        endcase
end



end

endmodule
```

・pos_one_shot.v(チャタリング除去及びワンショット回路モジュール)

```
module pos_one_shot(
                        input clk,
                        input enable,
                        input bt,
                        output    one_shot
                        );

reg [4:0] shift;
reg [1:0] btbt;
wire flag;



//チャタリング除去
always@(posedge clk)begin
            if(enable == 1'b1)
                        shift <= {shift[3:0], bt};
end

assign flag = &shift;

//ワンショット回路 パルス幅は clk の幅
always@(posedge clk)begin
                        btbt <= {btbt[0],flag};
end

assign one_shot = btbt[1] ^ btbt[0];



endmodule
```

・ucf ファイル

```
NET "sw" LOC="T10";
NET "reset" LOC="B8";
NET "i_bt1m"    LOC="D9";
NET "i_bt10m" LOC="A8";
NET "i_bt1h"    LOC="C9";
NET "i_bt10h" LOC="C4";



NET "AN[0]" LOC="N16";
NET "AN[1]" LOC="N15";
NET "AN[2]" LOC="P18";
NET "AN[3]" LOC="P17";

NET "LED[0]" LOC= "T17";
NET "LED[1]" LOC= "T18";
NET "LED[2]" LOC= "U17";
NET "LED[3]" LOC= "U18";
NET "LED[4]" LOC= "M14";
NET "LED[5]" LOC= "N14";
NET "LED[6]" LOC= "L14";

NET "clk" LOC="V10";
```