論理回路II 後期中間レポート

出席番号 **31**

氏名 橋本　千聡

課題 14-3

コード

TopModule.v

```
module TopModule(
    ////////////// CLOCK //////////
    input                       CLK1,
    input                       CLK2,
    ////////////// SEG7 //////////
    output          [7:0]       HEX0,
    output          [7:0]       HEX1,
    output          [7:0]       HEX2,
    output          [7:0]       HEX3,
    output          [7:0]       HEX4,
    output          [7:0]       HEX5,
    ////////////// Push Button //////////
    input           [1:0]       BTN,
    ////////////// LED //////////
    output          [9:0]       LED,
    ////////////// SW //////////
    input           [9:0]       SW

    );

    wire a, b, c, y;

    assign a = SW[3];
    assign b = SW[2];
    assign c = SW[1];

    my_circuit u1(a, b, c, y);

    assign LED = {7'h0, a, b, c, y};

    assign HEX0=8'hff;
    assign HEX1=8'hff;
    assign HEX2=8'hff;
    assign HEX3=8'hff;
    assign HEX4=8'hff;
    assign HEX5=8'hff;

endmodule
```

MyCircuit.v

```
module my_circuit(A, B, C, Y);
    input A, B, C;
    output Y;

    assign Y = (A & C)
             | (A & ~B)
             | (C & ~B)
             | (~A & B & ~C);
endmodule
```

動作確認

真理値表どおりの光り方になることを確認した

解説

my_circuitモジュール内で、与えられた論理式をVerilogの論理演算子を用いて記述した。 assign文を用いて、Yに論理式の結果を代入している。 また、TopModuleモジュール内でSW[3:1]をそれぞれA,B,Cに対応させ、LEDの下位4ビットにA,B,C,Yを割り当てている。 これにより、SW[3:1]の組み合わせに応じてLEDが正しく点灯することを確認した。

課題 15-5

コード

TopModule.v

```
module TopModule(
    ////////////// CLOCK //////////
    input                       CLK1,
    input                       CLK2,
    ////////////// SEG7 //////////
    output          [7:0]       HEX0,
    output          [7:0]       HEX1,
    output          [7:0]       HEX2,
    output          [7:0]       HEX3,
    output          [7:0]       HEX4,
    output          [7:0]       HEX5,
    ////////////// Push Button //////////
    input           [1:0]       BTN,
    ////////////// LED //////////
    output          [9:0]       LED,
    ////////////// SW //////////
    input           [9:0]       SW

    );

    wire clk, j, k, q, rb, nq;

    assign j  = SW[3];
    assign k  = SW[2];
    assign rb = SW[1];

    m_RSFF u1(~BTN[0],~BTN[1],clk,nq);  //プッシュボタンによるクロックの生成

    m_JKFF u2(clk, j, k, q, rb);

    assign LED={7'h0, clk, j, k, rb, q };       //LEDは下位5bitを使用

    assign HEX0=8'hff;  //7segは不使用
    assign HEX1=8'hff;
    assign HEX2=8'hff;
    assign HEX3=8'hff;
    assign HEX4=8'hff;
    assign HEX5=8'hff;

endmodule
```

FlipFlop.v

```
module m_RSFF( S, R, Q, QB );
    input S, R;
    output Q, QB;

    assign Q = ~(~S & QB);
    assign QB = ~(~R & Q);

endmodule

module m_JKFF ( CK, J, K, Q, RB );
    input CK, J, K, RB;
    output Q;
    reg Q;
    always @( posedge CK or negedge RB )
    begin
        if( RB == 1'b0 )
            Q <= 1'b0;
        else
            case( {J,K} )
                2'b00: Q <= Q;
                2'b01: Q <= 1'b0;
                2'b10: Q <= 1'b1;
                2'b11: Q <= ~Q;
            endcase
    end
endmodule
```

動作確認

- ボタンの on / off でのクロック生成がLEDR4を通して確認できた
- RB (SW1)が off のときはJ (SW3)、K (SW2)がどのような状態でクロックを生成してもQ (LEDR0)が消灯していた
- RB (SW0)が on のときは以下の挙動を確認した (CLK立ち上がり時の挙動)
    - Jが on、 Kが off のときはQが on になった
    - Jが off、 Kが on のときはQが off になった
    - Jが on、 Kが on　のときはQが反転した
    - Jが off、 Kが off のときはQの変化がなかった
- RBが on の状態から off にするとどのような入力でもQは off になった

解説

JKフリップフロップは、JとKの入力に応じて出力Qが変化する記憶素子である。
TopModuleモジュール内で、SW[3]をJ、SW[2]をK、SW[1]をRBに対応させ、プッシュボタンでクロックを生成している。
FlipFlopモジュール内で、m_RSFFモジュールはRSフリップフロップを実装し、m_JKFFモジュールでJKフリップフロップを実装している。
m_JKFFモジュールでは、RBが0のときにQを0にリセットし、RBが1のときにJとKの組み合わせに応じてQを更新する。
このようにして、JKフリップフロップの動作を確認した。

課題 16-3

コード

TopModule.v

```
module TopModule(
    ////////////// CLOCK //////////
    input                       CLK1,
    input                       CLK2,
    ////////////// SEG7 //////////
    output          [7:0]       HEX0,
    output          [7:0]       HEX1,
    output          [7:0]       HEX2,
    output          [7:0]       HEX3,
    output          [7:0]       HEX4,
    output          [7:0]       HEX5,
    ////////////// Push Button //////////
    input           [1:0]       BTN,
    ////////////// LED //////////
    output          [9:0]       LED,
    ////////////// SW //////////
    input           [9:0]       SW

    );

    wire [7:0] input_0, input_1, output_0, output_1;
    wire [3:0] sum;
    wire co;

    m_seven_segment u0(SW[3:0], input_0);
    m_seven_segment u1(SW[7:4], input_1);

    add4 u2(SW[3:0], SW[7:4], 1'b0, sum, co);

    m_seven_segment u3(sum, output_0);
    m_seven_segment u4(co, output_1);

    assign LED={6'h0,SW[7:0]};
    assign HEX0=output_0;
    assign HEX1=output_1;
    assign HEX2=input_0;
    assign HEX3=input_1;
    assign HEX4=8'hff;
    assign HEX5=8'hff;

endmodule
```

SevenSegment.v

```
module m_seven_segment(input [3:0] idat,output [7:0] odat);

    function [7:0] LedHex;
      input [3:0] num;
      begin
         case (num)
           4'h0:        LedHex = 8'b11000000;  // 0
           4'h1:        LedHex = 8'b11111001;  // 1
           4'h2:        LedHex = 8'b10100100;  // 2
           4'h3:        LedHex = 8'b10110000;  // 3
           4'h4:        LedHex = 8'b10011001;  // 4
           4'h5:        LedHex = 8'b10010010;  // 5
           4'h6:        LedHex = 8'b10000010;  // 6
           4'h7:        LedHex = 8'b11111000;  // 7
           4'h8:        LedHex = 8'b10000000;  // 8
           4'h9:        LedHex = 8'b10011000;  // 9
              4'ha:        LedHex = 8'b10001000;  // A
              4'hb:        LedHex = 8'b10000011;  // b
              4'hc:        LedHex = 8'b11000110;  // C
              4'hd:        LedHex = 8'b10100001;  // d
              4'he:        LedHex = 8'b10000110;  // E
              4'hf:        LedHex = 8'b10001110;  // F
           default:     LedHex = 8'b11111111;  // LED OFF
         endcase
      end
    endfunction

    assign odat = LedHex(idat);

endmodule

module add4(input  [3:0] a, b,
            input        ci,
            output [3:0] sum,
            output       co);

  assign {co, sum} = a + b + ci;
endmodule
```

動作確認

- スイッチと対応する16進数がHEX2,3に表示された
- 足し算の結果がHEX0,1に表示された

解説

TopModuleモジュール内で、SW[3:0]とSW[7:4]をそれぞれm_seven_segmentモジュールに入力し、対応する7セグメント表示をHEX2とHEX3に割り当てている。 add4モジュールでSW[3:0]とSW[7:4]の和を計算し、その結果を再びm_seven_segmentモジュールに入力してHEX0に表示している。 また、繰り上がり(co)をm_seven_segmentモジュールに入力してHEX1に表示している

課題17-5

コード

TopModule.v

```
module TopModule(
    ////////////// CLOCK //////////
    input                       CLK1,
    input                       CLK2,
    ////////////// SEG7 //////////
    output          [7:0]       HEX0,
    output          [7:0]       HEX1,
    output          [7:0]       HEX2,
    output          [7:0]       HEX3,
    output          [7:0]       HEX4,
    output          [7:0]       HEX5,
    ////////////// Push Button //////////
    input           [1:0]       BTN,
    ////////////// LED //////////
    output          [9:0]       LED,
    ////////////// SW //////////
    input           [9:0]       SW

    );
    wire clk, res, wnq;
    wire [3:0] wq;
    wire [7:0] hex;

    m_rs_flipflop u1(BTN[0], BTN[1], clk, wnq); //clock

    assign res = SW[0];         //reset

    m_counter(clk, res, wq);    //counter

    m_seven_segment u2(wq, hex);

    assign LED={6'h0,wq};
    assign HEX0=hex;
    assign HEX1=8'hff;
    assign HEX2=8'hff;
    assign HEX3=8'hff;
    assign HEX4=8'hff;
    assign HEX5=8'hff;

endmodule
```

Counter.v

```
module m_rs_flipflop(input set,input reset,output q,output nq);
    assign q=~(set & nq);
    assign nq=~(reset & q);
endmodule


module m_counter( ck, res, q );
    input   ck, res;
    output  [3:0] q;
    reg     [3:0] q;

    always @( posedge ck or posedge res )
    begin
        if( res == 1'b1 )
            q <= 4'h0;
        else
            q <= q + 4'h1 ;
    end
endmodule
```

SevenSegment.v

```
module m_seven_segment(input [3:0] idat,output [7:0] odat);

    function [7:0] LedHex;
      input [3:0] num;
      begin
         case (num)
           4'h0:        LedHex = 8'b11000000;  // 0
           4'h1:        LedHex = 8'b11111001;  // 1
           4'h2:        LedHex = 8'b10100100;  // 2
           4'h3:        LedHex = 8'b10110000;  // 3
           4'h4:        LedHex = 8'b10011001;  // 4
           4'h5:        LedHex = 8'b10010010;  // 5
           4'h6:        LedHex = 8'b10000010;  // 6
           4'h7:        LedHex = 8'b11111000;  // 7
           4'h8:        LedHex = 8'b10000000;  // 8
           4'h9:        LedHex = 8'b10011000;  // 9
              4'ha:        LedHex = 8'b10001000;  // A
              4'hb:        LedHex = 8'b10000011;  // b
              4'hc:        LedHex = 8'b11000110;  // C
              4'hd:        LedHex = 8'b10100001;  // d
              4'he:        LedHex = 8'b10000110;  // E
              4'hf:        LedHex = 8'b10001110;  // F
           default:     LedHex = 8'b11111111;  // LED OFF
         endcase
      end
    endfunction

    assign odat = LedHex(idat);
```

```
endmodule
```

動作確認

- ボタンを押すとカウントが進み、HEX0に16進数が表示された
- SW0をあげるとリセットされた

解説

TopModuleモジュール内で、プッシュボタンでクロックを生成し、SW[0]でリセット信号を生成している。
Counterモジュール内で、クロックの立ち上がりでカウントが進み、リセット信号が1のときにカウントが0になるように実装している。
SevenSegmentモジュールで、カウント値を7セグメント表示に変換し、HEX0に表示している。
LEDの下位4ビットにカウント値を表示している。

課題18

コード

Conter.v

```
    input clk,
    input stop,
    input manual_up,
    input manual_down,
    output reg [3:0] q,
    output reg c
);
    reg [3:0] cnt = 4'd0;

    always @(posedge clk) begin
        if (!manual_up && !manual_down) begin
            if(!stop) begin
                if (cnt == 4'd9) begin
                  cnt <= 0;
                  c <= 1;
                end else begin
                  cnt <= cnt + 1;
                  c <= 0;
                end
            end
        end else begin
            if (manual_up) begin
                if (cnt == 4'd9) begin
                    cnt <= 0;
                    c <= 1;
                end else begin
                    cnt <= cnt + 1;
                    c <= 0;
                end
            end
            if (manual_down) begin
                if (cnt == 0) begin
                    cnt <= 9;
                    c <= 0;
                end else begin
                    cnt <= cnt - 1;
                    c <= 0;
                end
            end
        end
        q <= cnt;
    end
endmodule
```

```
module m_6_counter_manual(
    input clk,
    input stop,
    input manual_up,
    input manual_down,
    output reg [2:0] q,
    output reg c
);
    reg [2:0] cnt = 3'd0;

    always @(posedge clk) begin
        if (!manual_up && !manual_down) begin
            if(!stop) begin
                if (cnt == 3'd5) begin
                    cnt <= 0;
                    c <= 1;
                end else begin
                    cnt <= cnt + 1;
                    c <= 0;
                end
            end
        end else begin
            if (manual_up) begin
                if (cnt == 3'd5) begin
                    cnt <= 0;
                    c <= 1;
                end else begin
                    cnt <= cnt + 1;
                    c <= 0;
                end
            end
            if (manual_down) begin
                if (cnt == 0) begin
                    cnt <= 5;
                    c <= 0;
                end else begin
                    cnt <= cnt - 1;
                    c <= 0;
                end
            end
        end
      q <= cnt;
    end
endmodule

module m_24_counter_manual(
    input clk,
    input stop,
    input manual_up,
    input manual_down,
    output reg [3:0] q0, // units
    output reg [3:0] q1  // tens
);
    reg [4:0] cnt = 5'd0;
```

```
    always @(posedge clk) begin
        if (!manual_up && !manual_down) begin
            if (!stop) begin
                if (cnt == 5'd23) cnt <= 0;
                else cnt <= cnt + 1;
            end
        end
        if (manual_up) begin
            if (cnt == 23) cnt <= 0;
            else cnt <= cnt + 1;
        end
        if (manual_down) begin
            if (cnt == 0) cnt <= 23;
            else cnt <= cnt - 1;
        end
        q1 <= cnt / 10; // tens
        q0 <= cnt % 10; // units
    end
endmodule


module m_time_set(
    input clk,
    input mode,       // SW[9]
    input set_sec,    // SW[0]
    input set_min,    // SW[1]
    input set_hr,     // SW[2]
    input btn_up,     // BTN[0]
    input btn_down,   // BTN[1]
    input [3:0] milli_sec, // 100msカウント表示

    output reg manual_up_sec,
    output reg manual_down_sec,
    output reg manual_up_min,
    output reg manual_down_min,
    output reg manual_up_hr,
    output reg manual_down_hr,

    output reg [9:0] led_out
);

    // ------------------------
    // 長押しカウンタ
    // ------------------------
    reg [4:0] up_cnt;
    reg [4:0] down_cnt;
    parameter LONG_PRESS = 5'd5; // 適宜調整

    // m_timer_decoder インスタンス
    wire [9:0] timer_led;
    m_timer_decoder u_decoder(milli_sec, timer_led);

    always @(posedge clk) begin
```

```
        if (mode) begin
            // BTN長押し判定
            if (btn_up) begin
                up_cnt <= 0;
            end else begin
                if (up_cnt < LONG_PRESS) up_cnt <= up_cnt + 1;
                     else up_cnt <= LONG_PRESS;
            end

            if (btn_down) begin
                down_cnt <= 0;
            end else begin
                if (down_cnt < LONG_PRESS) down_cnt <= down_cnt + 1;
                     else down_cnt <= LONG_PRESS;
            end

            manual_up_sec  <= set_sec  & ((up_cnt == LONG_PRESS) | (up_cnt
== 1));
            manual_down_sec<= set_sec  & ((down_cnt == LONG_PRESS) |
(down_cnt == 1));
            manual_up_min  <= set_min  & (up_cnt == LONG_PRESS);
            manual_down_min<= set_min  & (down_cnt == LONG_PRESS);
            manual_up_hr   <= set_hr   & (up_cnt == LONG_PRESS);
            manual_down_hr <= set_hr   & (down_cnt == LONG_PRESS);

                           // 設定モード
            led_out[0] <= set_sec;
            led_out[1] <= set_min;
            led_out[2] <= set_hr;
            led_out[9] <= 1'b1;        // 光らせる

            // デバッグ用LED[3-8]
            led_out[3] <= btn_up;            // BTN[0] 押下
            led_out[4] <= btn_down;          // BTN[1] 押下
            led_out[5] <= (up_cnt == LONG_PRESS);      // BTN[0] 長押し判定中
            led_out[6] <= (down_cnt == LONG_PRESS);    // BTN[1] 長押し判定中
            led_out[7] <= manual_up_sec;              // 設定モード中
            led_out[8] <= manual_down_sec; // SW[0-2] 選択フラグ

        end else begin
            // 通常モード：0.1秒カウントをデコードして LED[0-9] に表示
            led_out <= timer_led;

            manual_up_sec  <= 0;
            manual_down_sec<= 0;
            manual_up_min  <= 0;
            manual_down_min<= 0;
            manual_up_hr   <= 0;
            manual_down_hr <= 0;

            up_cnt <= 0;
            down_cnt <= 0;
        end
```

```
    end
endmodule
```

Timer.v

```
module m_prescale50M(input clk, output c_out);
    reg [25:0] cnt;
    wire wcout;

    assign wcout = (cnt == 26'd49999999) ? 1'b1 : 1'b0;
    assign c_out = wcout;

    always @(posedge clk) begin
        if (wcout)
            cnt <= 26'd0;
        else
            cnt <= cnt + 26'd1;
    end
endmodule

// 1/5000000 PreScaler (for 50 MHz input -> 10 Hz)
module m_prescale5M(input clk,output c_out);
    reg [22:0] cnt;
    wire wcout;

    assign wcout=(cnt==23'd4999999) ? 1'b1 : 1'b0;
    assign c_out=wcout;

    always @(posedge clk) begin
        if(wcout==1'b1)
            cnt=0;
        else
            cnt=cnt+1;
    end
endmodule

module m_prescale_50M_60Hz(input clk,output c_out);
    reg [20:0] cnt;
    wire wcout;

    assign wcout=(cnt==21'd833333) ? 1'b1 : 1'b0;
    assign c_out=wcout;

    always @(posedge clk) begin
        if(wcout==1'b1)
            cnt=0;
        else
            cnt=cnt+1;
    end
endmodule
```

```
module m_prescale_50M_3600Hz(input clk, output c_out);
    reg [16:0] cnt;
    wire wcout;

    assign wcout=(cnt==17'd138888) ? 1'b1 : 1'b0;
    assign c_out=wcout;

    always @(posedge clk) begin
        if(wcout==1'b1)
            cnt=0;
        else
            cnt=cnt+1;
    end
endmodule

module m_timer_decoder(input [3:0] dcnt, output [9:0] wsec);
    assign wsec[0]=(dcnt==4'd0) ? 1'b1 : 1'b0;
    assign wsec[1]=(dcnt==4'd1) ? 1'b1 : 1'b0;
    assign wsec[2]=(dcnt==4'd2) ? 1'b1 : 1'b0;
    assign wsec[3]=(dcnt==4'd3) ? 1'b1 : 1'b0;
    assign wsec[4]=(dcnt==4'd4) ? 1'b1 : 1'b0;
    assign wsec[5]=(dcnt==4'd5) ? 1'b1 : 1'b0;
    assign wsec[6]=(dcnt==4'd6) ? 1'b1 : 1'b0;
    assign wsec[7]=(dcnt==4'd7) ? 1'b1 : 1'b0;
    assign wsec[8]=(dcnt==4'd8) ? 1'b1 : 1'b0;
    assign wsec[9]=(dcnt==4'd9) ? 1'b1 : 1'b0;
endmodule
```

SevenSegment.v

```
module m_seven_segment(input [3:0] input_data,output [7:0] output_data);
    function [7:0] LedDec;
      input [3:0] num;
      begin
         case (num)
           4'h0:        LedDec = 8'b11000000;  // 0
           4'h1:        LedDec = 8'b11111001;  // 1
           4'h2:        LedDec = 8'b10100100;  // 2
           4'h3:        LedDec = 8'b10110000;  // 3
           4'h4:        LedDec = 8'b10011001;  // 4
           4'h5:        LedDec = 8'b10010010;  // 5
           4'h6:        LedDec = 8'b10000010;  // 6
           4'h7:        LedDec = 8'b11111000;  // 7
           4'h8:        LedDec = 8'b10000000;  // 8
           4'h9:        LedDec = 8'b10011000;  // 9
           4'ha:        LedDec = 8'b10001000;  // A
           4'hb:        LedDec = 8'b10000011;  // B
           4'hc:        LedDec = 8'b10100111;  // C
           4'hd:        LedDec = 8'b10100001;  // D
           4'he:        LedDec = 8'b10000110;  // E
           4'hf:        LedDec = 8'b10001110;  // F
```

```
            default:     LedDec = 8'b11111111;  // LED OFF
          endcase
        end
    endfunction
    assign output_data=LedDec(input_data);
endmodule

module m_seven_segment_add_dot(input [7:0] input_data, output [7:0]
output_data);
    assign output_data = input_data & 8'b01111111;
endmodule
```

TopModule.v

```
module TopModule(
    ////////////// CLOCK //////////
    input                       CLK1,
    input                       CLK2,
    ////////////// SEG7 //////////
    output          [7:0]       HEX0,
    output          [7:0]       HEX1,
    output          [7:0]       HEX2,
    output          [7:0]       HEX3,
    output          [7:0]       HEX4,
    output          [7:0]       HEX5,
    ////////////// Push Button //////////
    input           [1:0]       BTN,
    ////////////// LED //////////
    output          [9:0]       LED,
    ////////////// SW //////////
    input           [9:0]       SW

    );



    // ------------------------
    // 100ms クロック生成（m_prescale5M を使用）
    // ------------------------
    wire clk_100ms;
    m_prescale5M u0(CLK1, clk_100ms);

    // ------------------------
    // 1秒パルス用カウンタ（100ms×10 = 1秒）
    // ------------------------
    wire clk_1s;
    reg [3:0] cnt_100ms;
    always @(posedge clk_100ms) begin
        if (SW[9]) begin
            cnt_100ms <= cnt_100ms; // 設定モードで停止
        end else begin
```

```
            if (cnt_100ms == 4'd9) cnt_100ms <= 0;
            else cnt_100ms <= cnt_100ms + 1;
        end
    end
    assign clk_1s = (cnt_100ms == 4'd9) && !SW[9];

    // ------------------------
    // m_time_set 出力（manual_up/down + LED）
    // ------------------------
    wire manual_up_sec, manual_down_sec;
    wire manual_up_min, manual_down_min;
    wire manual_up_hr, manual_down_hr;

    wire [3:0] milli_sec;
    assign milli_sec = cnt_100ms;

    m_time_set u_set(
        .clk(clk_100ms),
        .mode(SW[9]),
        .set_sec(SW[0]),
        .set_min(SW[1]),
        .set_hr(SW[2]),
        .btn_up(BTN[0]),
        .btn_down(BTN[1]),
        .manual_up_sec(manual_up_sec),
        .manual_down_sec(manual_down_sec),
        .manual_up_min(manual_up_min),
        .manual_down_min(manual_down_min),
        .manual_up_hr(manual_up_hr),
        .manual_down_hr(manual_down_hr),
        .milli_sec(milli_sec),
        .led_out(LED)
    );

    // ------------------------
    // 共通カウンタ（手動+自動）
    // ------------------------
    wire [3:0] sec0, sec1, min0, min1, hr0, hr1;
    wire clk_sec0, clk_sec1, clk_min0, clk_min1, clk_hr;

    // 秒
    m_10_counter_manual c_sec0(
        .clk(clk_1s),
        .stop(SW[8]),
        .manual_up(manual_up_sec),
        .manual_down(manual_down_sec),
        .q(sec0),
        .c(clk_sec0)
    );

    m_6_counter_manual c_sec1(
        .clk(clk_sec0),
        .stop(SW[8]),
        .manual_up(manual_up_sec),
```

```
        .manual_down(manual_down_sec),
        .q(sec1),
        .c(clk_sec1)
    );

    // 分
    m_10_counter_manual c_min0(
        .clk(clk_sec1),
        .stop(SW[8]),
        .manual_up(manual_up_min),
        .manual_down(manual_down_min),
        .q(min0),
        .c(clk_min0)
    );

    m_6_counter_manual c_min1(
        .clk(clk_min0),
        .stop(SW[8]),
        .manual_up(manual_up_min),
        .manual_down(manual_down_min),
        .q(min1),
        .c(clk_min1)
    );

    // 時
    m_24_counter_manual c_hr(
        .clk(clk_min1),
        .stop(SW[8]),
        .manual_up(manual_up_hr),
        .manual_down(manual_down_hr),
        .q0(hr0),
        .q1(hr1)
    );

    // ------------------------
    // 7セグメント表示
    // ------------------------
    m_seven_segment u2(sec0, HEX0);
    m_seven_segment u3(sec1, HEX1);
    m_seven_segment u4(min0, HEX2);
    m_seven_segment u5(min1, HEX3);
    m_seven_segment u6(hr0, HEX4);
    m_seven_segment u7(hr1, HEX5);


endmodule
```

動作確認

- 通常モードではLEDでミリ秒を示すことを確認した
- 通常モードでは7セグが秒、分、時間を示すことを確認した
- 設定モードにはSW[9]で入り、SW[0,1,2]で秒、分、時間の設定を選択できることを確認した

- 設定モードではBTN[0]で増加、BTN[1]で減少できることを確認した
- 設定モードではBTNの長押しで連続増減できることを確認した
- SW[8]でカウントを停止できることを確認した