課題 24-2

コード

TopModule.v

```
module TopModule(
    input            CLK1,
    input            CLK2,
    output     [7:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5,
    input      [1:0] BTN,
    output     [9:0] LED,
    input      [9:0] SW,
    input      [3:0] KEY_ROW,
    output     [3:0] KEY_COL
);

    /* ===== Clock & Key ===== */
    wire clk;                    // 分周後の100Hzクロック（全体制御用）
    wire [3:0] key_val;          // 押されたキーを4bitで表現（0～F）
    wire [15:0] key_raw;         // マトリクスキーの生データ（16キー分）
    wire pushed;                 // 何かのキーが押されたことを示す信号
    reg  pushed_d;               // pushedを1クロック遅延させた信号（エッジ検出用）

    m_prescale      u0 (CLK1, clk);                          // クロック分周器
    m_matrix_key    u1 (clk, SW[0], KEY_ROW, KEY_COL, key_raw); // マトリクスキー走査
    m_dec16to4_calc u2 (key_raw, key_val, pushed);             // キーコード変換

    always @(posedge clk)
        pushed_d <= pushed;        // pushedの前回値を保持

    /* ===== State ===== */
    reg [1:0] state;             // 状態管理（0:num1入力, 1:num2入力, 2:結果表示）
    reg [1:0] op;                // 演算子（00:+, 01:-, 10:*, 11:/）

    /* ===== Input BCD ===== */
    reg [3:0] num1 [0:5];        // 1つ目の入力値（BCD, 最大6桁）
    reg [3:0] num2 [0:5];        // 2つ目の入力値（BCD, 最大6桁）
    reg [2:0] len1, len2;        // 各入力値の現在の桁数

    /* ===== Binary ===== */
    reg [19:0] bin1, bin2;       // BCDを10進数として変換した2進表現
    reg [39:0] bin_ans;          // 演算結果（最大13桁相当）

    /* ===== Result BCD ===== */
```

```
    reg [3:0] bcd_ans [0:12];     // 演算結果をBCDで保持（13桁分）
    reg [39:0] tmp;               // BIN→BCD変換用の作業レジスタ

    /* ===== Scroll Result ==== */
    reg [7:0] cnt;                // スクロール速度制御用カウンタ
    reg scroll_tick;              // スクロール更新タイミング信号
    reg [3:0] scroll_pos;         // 表示開始位置（BCD配列のインデックス）

    always @(posedge clk or posedge SW[0]) begin
        if (SW[0]) begin
            cnt <= 0;             // リセット時にカウンタ初期化
            scroll_tick <= 1'b0;  // スクロール停止
            scroll_pos <= 4'd7;   // 初期表示位置
        end
        else if(state == 2) begin // 結果表示状態のみスクロール有効
            if (cnt == 8'd30) begin
                cnt <= 0;         // 一定周期でスクロール更新
                scroll_tick <= 1'b1;
            end
            else begin
                cnt <= cnt + 1'b1;
                scroll_tick <= 1'b0;
            end
            if (scroll_tick) begin
                scroll_tick <= 1'b0;
                if (scroll_pos == 0)
                    scroll_pos <= 4'd12; // 最後尾に戻る
                else
                    scroll_pos <= scroll_pos - 1'b1;
            end
        end
        else begin
            cnt <= 0;             // 入力状態ではスクロール無効
            scroll_tick <= 1'b0;
            scroll_pos <= 4'd7;
        end
    end

    /* ===== Reset & Input ===== */
    always @(posedge clk or posedge SW[0]) begin
        if (SW[0]) begin
            state <= 0;           // 状態初期化
            op    <= 0;           // 演算子初期化
            len1  <= 0;           // 桁数初期化
            len2  <= 0;
            bin_ans <= 0;         // 演算結果クリア

            num1[0]<=0; num1[1]<=0; num1[2]<=0;
            num1[3]<=0; num1[4]<=0; num1[5]<=0;
            num2[0]<=0; num2[1]<=0; num2[2]<=0;
            num2[3]<=0; num2[4]<=0; num2[5]<=0;
        end
        else if (pushed && !pushed_d) begin // 押下の立ち上がり検出
            if (key_val <= 9) begin         // 数字キー入力
```

```
                if (state==0 && len1<6) begin
                    // num1を左シフトして新しい桁を格納
                    num1[5] <= num1[4];
                    num1[4] <= num1[3];
                    num1[3] <= num1[2];
                    num1[2] <= num1[1];
                    num1[1] <= num1[0];
                    num1[0] <= key_val;
                    len1 <= len1 + 1'b1;
                end
                else if (state==1 && len2<6) begin
                    // num2を左シフトして新しい桁を格納
                    num2[5] <= num2[4];
                    num2[4] <= num2[3];
                    num2[3] <= num2[2];
                    num2[2] <= num2[1];
                    num2[1] <= num2[0];
                    num2[0] <= key_val;
                    len2 <= len2 + 1'b1;
                end
            end
            else begin
                case (key_val)
                    4'hA: begin op<=2'b00; state<=1; len2<=0; end // 加算
                    4'hB: begin op<=2'b01; state<=1; len2<=0; end // 減算
                    4'hC: begin op<=2'b10; state<=1; len2<=0; end // 乗算
                    4'hD: begin op<=2'b11; state<=1; len2<=0; end // 除算
                    4'hE: begin                                   // クリア
                        state <= 0;
                        op    <= 0;
                        len1  <= 0;
                        len2  <= 0;
                        bin_ans <= 0;

                        num1[0]<=0; num1[1]<=0; num1[2]<=0;
                        num1[3]<=0; num1[4]<=0; num1[5]<=0;
                        num2[0]<=0; num2[1]<=0; num2[2]<=0;
                        num2[3]<=0; num2[4]<=0; num2[5]<=0;
                    end
                    4'hF: begin                                   // イコール
                        state <= 2;
                        case (op)
                            2'b00: bin_ans <= bin1 + bin2;        // 加算
                            2'b01: bin_ans <= (bin1>=bin2)?(bin1-bin2):0;
// 減算
                            2'b10: bin_ans <= bin1 * bin2;        // 乗算
                            2'b11: bin_ans <= (bin2!=0)?(bin1/bin2):0; //
除算
                        endcase
                    end
                endcase
            end
        end
    end
```

```
    /* ===== BCD -> BIN ===== */
    always @(*) begin
        // BCD配列を10進数として結合
        bin1 =
            (((((num1[5]*10 + num1[4])*10 + num1[3])*10
              + num1[2])*10 + num1[1])*10 + num1[0]);

        bin2 =
            (((((num2[5]*10 + num2[4])*10 + num2[3])*10
              + num2[2])*10 + num2[1])*10 + num2[0]);
    end

    /* ===== BIN -> BCD ===== */
    always @(*) begin
        tmp = bin_ans;              // 作業用に演算結果をコピー

        // 10で割った余りを順にBCDとして取り出す
        bcd_ans[0]  = tmp % 10;  tmp = tmp / 10;
        bcd_ans[1]  = tmp % 10;  tmp = tmp / 10;
        bcd_ans[2]  = tmp % 10;  tmp = tmp / 10;
        bcd_ans[3]  = tmp % 10;  tmp = tmp / 10;
        bcd_ans[4]  = tmp % 10;  tmp = tmp / 10;
        bcd_ans[5]  = tmp % 10;  tmp = tmp / 10;
        bcd_ans[6]  = tmp % 10;  tmp = tmp / 10;
        bcd_ans[7]  = tmp % 10;  tmp = tmp / 10;
        bcd_ans[8]  = tmp % 10;  tmp = tmp / 10;
        bcd_ans[9]  = tmp % 10;  tmp = tmp / 10;
        bcd_ans[10] = tmp % 10;  tmp = tmp / 10;
        bcd_ans[11] = tmp % 10;
        bcd_ans[12] = 4'hF;        // 末尾は空白表示
    end

    /* ===== Display select ===== */
    reg  [3:0] d0, d1, d2, d3, d4, d5; // 7セグメント表示用データ

    always @(*) begin
        // 初期値はすべて空白
        d0 = 4'hF; d1 = 4'hF; d2 = 4'hF;
        d3 = 4'hF; d4 = 4'hF; d5 = 4'hF;

        if (state < 2) begin
            // 入力状態の表示
            if (state == 0) begin
                d0 = (len1>0)?num1[0]:4'hf;
                d1 = (len1>1)?num1[1]:4'hf;
                d2 = (len1>2)?num1[2]:4'hf;
                d3 = (len1>3)?num1[3]:4'hf;
                d4 = (len1>4)?num1[4]:4'hf;
                d5 = (len1>5)?num1[5]:4'hf;
            end
            else if (state == 1) begin
                d0 = (len2>0)?num2[0]:4'hf;
                d1 = (len2>1)?num2[1]:4'hf;
```

```
                d2 = (len2>2)?num2[2]:4'hf;
                d3 = (len2>3)?num2[3]:4'hf;
                d4 = (len2>4)?num2[4]:4'hf;
                d5 = (len2>5)?num2[5]:4'hf;
            end
        end
        else if (state == 2) begin
            // 結果表示時はスクロール表示
            d0 = bcd_ans[(scroll_pos+0)%13];
            d1 = bcd_ans[(scroll_pos+1)%13];
            d2 = bcd_ans[(scroll_pos+2)%13];
            d3 = bcd_ans[(scroll_pos+3)%13];
            d4 = bcd_ans[(scroll_pos+4)%13];
            d5 = bcd_ans[(scroll_pos+5)%13];
        end
    end

    /* ===== 7seg ===== */
    m_7segment s0(d0, HEX0); // 7セグメント表示器
    m_7segment s1(d1, HEX1);
    m_7segment s2(d2, HEX2);
    m_7segment s3(d3, HEX3);
    m_7segment s4(d4, HEX4);
    m_7segment s5(d5, HEX5);

    assign LED = {5'd0, scroll_tick, op, state}; // 動作状態をLEDに表示

endmodule
```

matrix_key.v

```


// 分周器(100Hz)
module m_prescale(input clk,output c_out);
    reg [19:0] cnt;                 // 分周用カウンタ
    wire wcout;                     // カウンタ終端検出

    assign wcout=(cnt==20'd499999) ? 1'b1 : 1'b0; // 100Hz生成
    assign c_out=wcout;             // 出力クロック

    always @(posedge clk) begin
        if (wcout)
            cnt <= 20'd0;           // カウンタリセット
        else
            cnt <= cnt + 20'd1;     // カウントアップ
    end

endmodule


module m_matrix_key(
```

```
    input             clk, rst,
    input      [3:0]  row,          // 行入力
    output reg [3:0]  col,          // 列出力
    output reg [15:0] key,          // 全キー状態
    output            tc
    ) ;

    reg [2:0]  index ;              // 走査インデックス
    reg [15:0] tmp ;                // 一時的なキー保持

    always @(posedge rst or posedge clk) begin
        if(rst == 1'b1)begin
           tmp   <= 16'hFFFF ;      // 初期状態（全未押下）
            key   <= 16'h0000 ;
            index <= 3'd0 ;
        end
        else begin
            if (index[0] == 1'b0) begin
                case (index[2:1])
                    2'd0: begin
                            col <= 4'b1110 ; // 列0をアクティブ
                            key <= ~tmp ;    // 確定したキー状態を出力
                            tmp <= 16'hFFFF ;
                        end
                    2'd1: col <= 4'b1101 ; // 列1をアクティブ
                    2'd2: col <= 4'b1011 ; // 列2をアクティブ
                    2'd3: col <= 4'b0111 ; // 列3をアクティブ
                endcase
            end
            else begin
                // 行信号を読み取りtmpに格納
                tmp[{2'd0, index[2:1]}] <= row[0] ;
                tmp[{2'd1, index[2:1]}] <= row[1] ;
                tmp[{2'd2, index[2:1]}] <= row[2] ;
                tmp[{2'd3, index[2:1]}] <= row[3] ;
            end

            index <= index + 3'd1 ; // 次の走査位置へ

        end
    end

    assign tc = (index == 3'd0) ? 1'b1 : 1'b0 ; // 1周完了検出

endmodule

// 16bit→4bitデコーダ(計算機仕様)
module m_dec16to4_calc (
    input [15:0] key,             // マトリクスキー入力
    output [3:0] out,             // デコード後のキー値
    output       pushed           // 押下検出信号
) ;

    function [4:0] f ;
```

```
        input [15:0] in ;
        case(in)
              16'h0001: f = { 1'b1, 4'h1 } ;
              16'h0002: f = { 1'b1, 4'h2 } ;
              16'h0004: f = { 1'b1, 4'h3 } ;
              16'h0008: f = { 1'b1, 4'hA } ; // +
              16'h0010: f = { 1'b1, 4'h4 } ;
              16'h0020: f = { 1'b1, 4'h5 } ;
              16'h0040: f = { 1'b1, 4'h6 } ;
              16'h0080: f = { 1'b1, 4'hB } ; // -
              16'h0100: f = { 1'b1, 4'h7 } ;
              16'h0200: f = { 1'b1, 4'h8 } ;
              16'h0400: f = { 1'b1, 4'h9 } ;
              16'h0800: f = { 1'b1, 4'hC } ; // *
              16'h1000: f = { 1'b1, 4'hE } ; // C
              16'h2000: f = { 1'b1, 4'h0 } ;
              16'h4000: f = { 1'b1, 4'hF } ; // =
              16'h8000: f = { 1'b1, 4'hD } ; // /
              default:  f = { 1'b0, 4'h0 } ; // 未押下
        endcase
    endfunction

    assign { pushed, out } = f(key) ; // 押下有無とキー値を同時出力

endmodule
```

動作確認

- 電卓を使って四則演算が可能なことを確認した
- 複数桁表示で適切に数字が流れ、区切りのスペースも表示されることを確認した
- 入力時には本物の電卓のように数字が追加されていくことを確認した

解説

マトリクスキーを用いた電卓回路を拡張し、6桁と6桁の演算結果を最大13桁として扱い、7セグメントLED上をスクロール表示する構成としている。

まずクロックは m_prescale により 100Hz に分周され、マトリクスキーの走査や入力処理を安定した周期で行っている。m_matrix_key は列を順番に駆動し、行の状態を読み取ることで、16個のキー状態を 16bit の信号として出力する。

m_dec16to4_calc ではキー配置を電卓仕様に合わせ、数字キーと演算子キーを区別しながら 4bit の key_val と打鍵検出信号 pushed を生成する。pushed は押下の立ち上がりのみを利用するため、pushed_d により1クロック遅延させてエッジ検出を行っている。

数値入力は BCD 形式で行い、num1 と num2 に最大6桁まで保存される。入力された数字は左シフトしながら格納されるため、通常の電卓と同じ感覚で桁を増やすことができる。演算子キーが押されると入力状態は num2 入力へ遷移し、イコールキーが押されると計算状態へ移行する。

計算時には BCD で保持していた数値を 10進数として bin1 と bin2 に変換し、加減乗除の演算を行う。演算結果は最大13桁となるため 40bit の bin_ans に保持し、その後 10での剰余と除算を繰り返すことで BCD 配

列 bcd_ans に展開している。

表示部では入力中は num1 または num2 をそのまま 6桁分表示し、結果表示状態では bcd_ans をスクロール位置 scroll_pos に応じて切り出すことで、13桁の計算結果が流れるように表示される。これにより、通常の1桁電卓では扱えない大きな数値でも視認可能な表示を実現している。