

# 論理回路II 後期期末レポート

---

出席番号 31

氏名 橋本 千聰

# 課題22-3

## 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
);

//=====
// 内部信号定義
//=====
reg [3:0] cnt;      // 表示位置を制御する4bitカウンタ
wire     clk_1s;   // 1秒周期クロック

//=====
// 50MHz → 1Hz 分周回路
//=====
// CLK1(50MHz)を入力し、1秒ごとに1パルス出力
m_prescale50M tim(
    .CLK1(CLK1),
    .c_out(clk_1s)
);

//=====
// 1秒ごとにカウンタ更新
//=====
always @(posedge clk_1s) begin
    cnt = cnt + 1; // 4bitなので 0~15 で循環
end

//=====
// LED表示
//=====
```

```

//=====
assign LED[0] = SW[0]; // スイッチ0の状態をLED0に表示

//=====
// ROMを使った7セグ表示
// cnt にオフセットを加えて
// 文字が横にスクロールするように表示
//=====

m_rom u1(cnt+5, SW[0], HEX0);
m_rom u2(cnt+4, SW[0], HEX1);
m_rom u3(cnt+3, SW[0], HEX2);
m_rom u4(cnt+2, SW[0], HEX3);
m_rom u5(cnt+1, SW[0], HEX4);
m_rom u6(cnt+0, SW[0], HEX5);

endmodule

```

## Rom.v

```

//=====
// 7セグ用ROMモジュール
//=====

// adr : 表示文字アドレス
// sw : 表示切替スイッチ
// dat : 7セグ出力データ
//=====

module m_rom(
    input [3:0] adr,    // 文字位置アドレス
    input      sw,     // 表示切替用スイッチ
    output [7:0] dat   // 7セグメントLED出力
);

    reg [7:0] data;    // 内部保持用レジスタ
    assign dat = data; // 出力に接続

//=====
// アドレスが変化したら即時更新
//=====

always @(adr) begin
    //
    // sw = 1 のとき
    // "GOODbyE" を表示
    //

    if(sw) begin
        case (adr)
            4'h0: data = 8'b11000010; // G
            4'h1: data = 8'b11000000; // O
            4'h2: data = 8'b11000000; // O
            4'h3: data = 8'b10100001; // d
            4'h4: data = 8'b10000011; // b
        endcase
    end
end

```

```

        4'h5: data = 8'b10010001; // y
        4'h6: data = 8'b10000110; // E
        4'h7: data = 8'b11111111; // スペース
        4'h8: data = 8'b11000010; // G
        4'h9: data = 8'b11000000; // O
        4'ha: data = 8'b11000000; // O
        4'hb: data = 8'b10100001; // d
        4'hc: data = 8'b10000011; // b
        4'hd: data = 8'b10010001; // y
        4'he: data = 8'b10000110; // E
        4'hf: data = 8'b11111111; // スペース
        default: data = 8'hff;
    endcase
end
//-----
// sw = 0 のとき
// "HELLO" を表示
//-----
else begin
    case (adr)
        4'h0: data = 8'b10001001; // H
        4'h1: data = 8'b10000110; // E
        4'h2: data = 8'b11000111; // L
        4'h3: data = 8'b11000111; // L
        4'h4: data = 8'b11000000; // O
        4'h5: data = 8'b11111111; // スペース
        4'h6: data = 8'b11111111; // スペース
        4'h7: data = 8'b11111111; // スペース
        4'h8: data = 8'b10001001; // H
        4'h9: data = 8'b10000110; // E
        4'ha: data = 8'b11000111; // L
        4'hb: data = 8'b11000111; // L
        4'hc: data = 8'b11000000; // O
        4'hd: data = 8'b11111111; // スペース
        4'he: data = 8'b11111111; // スペース
        4'hf: data = 8'b11111111; // スペース
        default: data = 8'hff;
    endcase
end
end
endmodule

//================================================================
// 50MHz → 1Hz 分周回路
//================================================================
module m_prescale50M(
    input clk,      // 50MHz クロック
    output c_out    // 1秒ごとのパルス出力
);

    reg [25:0] cnt; // 50,000,000 まで数えるカウンタ
    wire cout; // カウンタ満了フラグ

    // カウンタが 49,999,999 に達したら 1 を出力

```

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

// クロック立ち上がりでカウント
always @(posedge clk) begin
    if (wcout)
        cnt <= 26'd0;           // 1秒経過でリセット
    else
        cnt <= cnt + 26'd1;
end
endmodule
```

## 動作確認

- スイッチのON/OFFで "HELLO" と "GOODbyE" の表示を切り替え可能
- 7セグLEDに文字が左にスクロールして表示される

## 解説

m\_prescale50M を用いて 50MHz のクロックを分周し 1 秒に 1 回だけ High になるパルスを生成している。このパルスを表示更新のタイミングとして使用することで 表示速度を人間が読み取りやすい速さにしている。

cnt は 1 秒ごとに 1 ずつ増加する 4 ビットのカウンタで 0 から 15 まで循環する。この値は 現在どの文字を表示するかを決めるためのスクロール位置として使われている。

文字の表示には m\_rom を使用しており これは文字パターンを格納した ROM である。入力として 表示したい文字番号を示す adr と 表示内容を切り替えるためのスイッチ信号 sw を受け取り 出力として 7 セグメント LED の点灯パターンを生成する。

各 7 セグメント LED には cnt に異なるオフセットを加えた値を adr として与えている。1 秒ごとに cnt が増加することで 各表示器が参照する文字アドレスが順番にずれていき その結果 文字列が左方向へ流れているように見える。

# 課題22-5

## 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 w_we;           // RAM 書き込みイネーブル信号
wire w_btn0;         // チャタリング除去後の BTN[0]
wire [3:0] rdata;   // RAM からの読み出しデータ

// BTN[0] のチャタリングを除去する
m_chattering u0(CLK1, BTN[0], w_btn0);

// ボタンが押されている間だけ書き込みを有効にする
assign w_we = ~w_btn0;

// RAM モジュール
// アドレスは SW[9:4]
// 書き込みデータは SW[3:0]
m_ram u1(SW[9:4], SW[3:0], w_we, rdata);

// LED は使用しないため全消灯
assign LED = 10'h0;

// 7 セグメント LED 表示
// HEX0 には RAM から読み出したデータを表示
m_seven_segment s0(rdata, HEX0);

// HEX1 には書き込みデータを表示
m_seven_segment s1(SW[3:0], HEX1);
```

```
// HEX2 と HEX3 にはアドレスを 2 桁で表示
m_seven_segment_2 s2(SW[9:4], HEX2, HEX3);

// 使用しない 7 セグは消灯
assign HEX4 = 8'hff;
assign HEX5 = 8'hff;

endmodule
```

## RAM.v

```
module m_ram(input [5:0] adr, input [3:0] wdata, input we, output [3:0]
rdata);
    reg [3:0] mem[0:63]; // 64 ワード分の 4bit RAM

    // 指定されたアドレスの内容を常に出力
    assign rdata = mem[adr];

    // 書き込みイネーブルの立ち上がりでデータを書き込む
    always @(posedge we) begin
        mem[adr] = wdata;
    end
endmodule

// チャタリング除去回路
module m_chattering(input clk, input sw_in, output sw_out);
    reg [15:0] cnt; // 分周用カウンタ
    reg swreg; // スイッチ状態保持用レジスタ
    wire iclk; // 低速クロック

    assign sw_out = swreg;

    // カウンタを回し続ける
    always @(posedge clk) begin
        cnt = cnt + 1;
    end

    // カウンタの上位ビットを低速クロックとして使用
    assign iclk = cnt[15];

    // 一定周期ごとにスイッチ入力を取り込む
    always @(posedge iclk) begin
        swreg = sw_in;
    end
endmodule

// 7 セグメントデコーダ
module m_seven_segment(input [3:0] idat, output [7:0] odat);
    parameter dot = 1'b1;
```

```
function [7:0] LedDec;
    input [3:0] num;
    begin
        case (num)
            4'h0: LedDec = 8'b11000000;
            4'h1: LedDec = 8'b11111001;
            4'h2: LedDec = 8'b10100100;
            4'h3: LedDec = 8'b10110000;
            4'h4: LedDec = 8'b10011001;
            4'h5: LedDec = 8'b10010010;
            4'h6: LedDec = 8'b10000010;
            4'h7: LedDec = 8'b11111000;
            4'h8: LedDec = 8'b10000000;
            4'h9: LedDec = 8'b10011000;
            4'ha: LedDec = 8'b10001000;
            4'hb: LedDec = 8'b10000011;
            4'hc: LedDec = 8'b10100111;
            4'hd: LedDec = 8'b10100001;
            4'he: LedDec = 8'b10000110;
            4'hf: LedDec = 8'b10001110;
            default: LedDec = 8'b11111111;
        endcase
    end
endfunction

wire [7:0] tdat;
assign tdat = LedDec(idat);
assign odat = {dot, tdat[6:0]};
endmodule

// 7 セグメントデコーダ 2 桁表示
module m_seven_segment_2(input [5:0] idat, output [7:0] odat1, output [7:0] odat2);
    parameter dot = 1'b1;

    function [7:0] LedDec;
        input [3:0] num;
        begin
            case (num)
                4'h0: LedDec = 8'b11000000;
                4'h1: LedDec = 8'b11111001;
                4'h2: LedDec = 8'b10100100;
                4'h3: LedDec = 8'b10110000;
                4'h4: LedDec = 8'b10011001;
                4'h5: LedDec = 8'b10010010;
                4'h6: LedDec = 8'b10000010;
                4'h7: LedDec = 8'b11111000;
                4'h8: LedDec = 8'b10000000;
                4'h9: LedDec = 8'b10011000;
                4'ha: LedDec = 8'b10001000;
                4'hb: LedDec = 8'b10000011;
                4'hc: LedDec = 8'b10100111;
                4'hd: LedDec = 8'b10100001;
                4'he: LedDec = 8'b10000110;
            endcase
        end
    endfunction
```

```
        4'hf: LedDec = 8'b10001110;
        default: LedDec = 8'b11111111;
    endcase
end
endfunction

wire [7:0] tdat1;
wire [7:0] tdat2;

assign tdat1 = LedDec(idat[3:0]);
assign tdat2 = LedDec(idat[5:4]);
assign odat1 = {dot, tdat1[6:0]};
assign odat2 = {dot, tdat2[6:0]};
endmodule
```

## 動作確認

- スイッチでアドレスとデータを指定し、押しボタンを押すと RAM にデータが書き込まれることを確認した。
- 値が更新されていることが確認できた

## 解説

スライドスイッチの上位 6 ビットで RAM のアドレスを指定し、下位 4 ビットで書き込みデータを決める。押しボタンを押すと、その瞬間に指定したアドレスへデータが書き込まれる。

押しボタンにはチャタリングが発生するため、そのまま RAM の制御に使うと意図しない複数回の書き込みが起きてしまう。そこでクロックを分周し、一定周期ごとにスイッチ状態を取り込むことで入力を安定させている。この安定化された信号を反転させ、書き込みイネーブルとして使用している。

RAM は 64 ワード、1 ワード 4 ビットの構成で、アドレスが指定されると常にその内容が読み出される。読み出したデータは HEX0 に表示され、現在書き込もうとしているデータは HEX1 に表示される。また、指定しているアドレスは 2 行の 7 セグメント LED に分けて表示されるため、どのアドレスを操作しているのかを視覚的に確認できる。

このように、入力、記憶、出力の流れをすべて目で確認できる構成になっており、RAM の動作原理を理解するための実験回路として適している。

## 課題23-2

### コード

matrix\_key.v

```

module m_matrix_key(
    input      clk, rst,      // クロック , リセット
    input [3:0] row,          // 4bit入力 行
    output reg [3:0] col,     // 4bit出力 列
    output reg [15:0] key,    // 16bitキー出力
    output      tc            // 出力カウント
) ;

reg [2:0] index ;
reg [15:0] tmp ;

always @(posedge rst or posedge clk) begin
    if(rst == 1'b1)begin
        tmp  <= 16'hFFFF ;
        key  <= 16'h0000 ;
        index <= 3'd0 ;
    end
    else begin
        // LSB=0 のとき : col を1列ずつ Low にしてスキャン
        if (index[0] == 1'b0) begin
            case (index[2:1])
                2'd0: begin
                    col <= 4'b1110 ;
                    key <= ~tmp ;      // 1周期分の結果を key に反映
                end
                2'd1: col <= 4'b1101 ;
                2'd2: col <= 4'b1011 ;
                2'd3: col <= 4'b0111 ;
            endcase
        end
        // LSB=1 のとき : row を読み取り tmp に保存
        else begin
            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 ;
    end
end

```

映

```
// index が一周したタイミングを示すフラグ
assign tc = (index == 3'd0) ? 1'b1 : 1'b0 ;

endmodule

// 押下されているキーを4bitで出力 (16bit→4bitデコーダ)
module m_dec16to4 (
    input [15:0] key,           // 16bit入力
    output [3:0] out,          // 4bit出力
    output      pushed        // 打鍵検出
);

function [4:0] f ;
    input [15:0] in ;
    case(in)
        16'h0001: f = { 1'b1, 4'h0 } ;
        16'h0002: f = { 1'b1, 4'h1 } ;
        16'h0004: f = { 1'b1, 4'h2 } ;
        16'h0008: f = { 1'b1, 4'h3 } ;
        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'h7 } ;
        16'h0100: f = { 1'b1, 4'h8 } ;
        16'h0200: f = { 1'b1, 4'h9 } ;
        16'h0400: f = { 1'b1, 4'hA } ;
        16'h0800: f = { 1'b1, 4'hB } ;
        16'h1000: f = { 1'b1, 4'hC } ;
        16'h2000: f = { 1'b1, 4'hD } ;
        16'h4000: f = { 1'b1, 4'hE } ;
        16'h8000: f = { 1'b1, 4'hF } ;
        default: f = { 1'b0, 4'h0 } ; // 同時押し・未押下は無効
    endcase
endfunction

assign { pushed, out } = f(key) ;

endmodule

module m_convert_num (
    input [3:0] key,
    output [3:0] num
);
function [3:0] f;
    input [3:0] in;
    case(in)
        4'h0: f = 4'h1;
        4'h1: f = 4'h2;
        4'h2: f = 4'h3;
        4'h3: f = 4'hA;
        4'h4: f = 4'h4;
        4'h5: f = 4'h5;
        4'h6: f = 4'h6;
        4'h7: f = 4'hB;
```

```

    4'h8: f = 4'h7;
    4'h9: f = 4'h8;
    4'hA: f = 4'h9;
    4'hB: f = 4'hC;
    4'hD: f = 4'h0;
    4'hF: f = 4'hD;
    default: f = 4'hF;
endcase
endfunction

assign num = f(key);

endmodule

```

## SevenSegment.v

```

// 分周器(100Hz)
module m_prescale(input clk, output c_out);
    reg [19:0] cnt;
    wire wcout;

    assign wcout=(cnt==20'd499999) ? 1'b1 : 1'b0;
    assign c_out=wcout;

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

// マトリクスキー用7セグLED表示
module m_mat7segment(input [3:0] idat, input pushed, output [7:0] odat);

    function [7:0] LedDec;
        input [3:0] num;
        begin
            case (num)
                4'h0:     LedDec = 8'b11000000;
                4'h1:     LedDec = 8'b11111001;
                4'h2:     LedDec = 8'b10100100;
                4'h3:     LedDec = 8'b10110000;
                4'h4:     LedDec = 8'b10011001;
                4'h5:     LedDec = 8'b10010010;
                4'h6:     LedDec = 8'b10000010;
                4'h7:     LedDec = 8'b11111000;
                4'h8:     LedDec = 8'b10000000;
                4'h9:     LedDec = 8'b10011000;
                4'ha:     LedDec = 8'b10001000;
                4'hb:     LedDec = 8'b10000011;
            endcase
        end
    endfunction
endmodule

```

```

    4'hc:      LedDec = 8'b10100111;
    4'hd:      LedDec = 8'b10100001;
    4'he:      LedDec = 8'b10000110;
    4'hf:      LedDec = 8'b10001110;
    default:   LedDec = 8'b11111111;
  endcase
end
endfunction

// キーが押されているときのみ表示
assign odat= (pushed) ? LedDec(idat) : 8'b11111111;

endmodule

```

## TopModule.v

```

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

wire clk;
wire [3:0] wq;          // 押下キー(4bit)
wire [3:0] num;         // 表示用変換後データ
wire [15:0] key;        // 押下キー(16bit)
wire [7:0] dec_pat0;    // 7セグ表示パターン
wire pushed;            // 打鍵検出

// 100Hz クロック生成
m_prescale(CLK1, clk);

// マトリクスキーのスキャン
m_matrix_key(clk, SW[0], KEY_ROW, KEY_COL, key, tc);

// 16bitキーを4bitに変換
m_dec16to4(key, wq, pushed);

// 配列番号 → 表示番号変換
m_convert_num(wq, num);

```

```
// 7セグ表示
m_mat7segment(num, pushed, dec_pat0);

assign LED = {6'd0, wq};
assign HEX0 = dec_pat0;
assign HEX1 = 8'hff;
assign HEX2 = 8'hff;
assign HEX3 = 8'hff;
assign HEX4 = 8'hff;
assign HEX5 = 8'hff;

endmodule
```

## 動作確認

- m\_convert\_num モジュールのシミュレーションを行い、キー番号と表示番号の対応が正しいことを確認した。
- 実際にボタンを押して問題ないことを確認した

## 解説

4×4 マトリクスキーをスキャンして押下されたキーを検出し、その結果を 7 セグメント LED に表示している。

m\_matrix\_key は、列を 1 本ずつ Low にして行入力を読み取ることでキーの状態を取得する。index を用いて「列出力」と「行読み取り」を交互に行い、1 周期分の結果を tmp に蓄積する。1 周すると key に反映され、16bit のワンホットデータとして出力される。

m\_dec16to4 では、16bit のワンホット信号から押されているキーを 4bit の番号に変換する。同時押しや未押下の場合は pushed を 0 にし、無効として扱う。

m\_convert\_num は、マトリクスキーの物理配置と表示したい数値の対応を変換するためのモジュールで、キー番号を 7 セグ表示用の値に並び替えている。

m\_prescale は入力クロックを分周し、キー入力を安定して読み取るための低速クロック（100Hz）を生成する。

m\_mat7segment は、押下されたキーがあるときのみ対応する 7 セグメントの点灯パターンを出力し、キーが押されていない場合は消灯する。

TopModule では、これらの各モジュールを接続し、マトリクスキー入力から 7 セグメント LED 表示までの一連の処理をまとめている。

# 課題23-4

## 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,
    //////////////// Matrix Key ///////////
    input [3:0]    KEY_ROW,
    output [3:0]   KEY_COL
);

wire clk;                      // 100Hzに分周されたクロック
wire [3:0] wq;                // 押下キー(4bit)
wire [3:0] num;                // 表示用に変換されたキー値
wire [15:0] key;              // マトリクスキーの16bit出力
reg [3:0] rdata0, rdata1, rdata2, rdata3, rdata4, rdata5; // 入力履歴保存

レジスタ
wire pushed;                  // 打鍵検出信号
reg pushed_d;                 // pushedの1クロック遅延

m_prescale(CLK1, clk);        // 50MHz → 100Hz 分周

m_matrix_key(clk, SW[0], KEY_ROW, KEY_COL, key, tc); // マトリクスキー走査

m_dec16to4(key, wq, pushed); // 押下キーを4bitに変換

m_convert_num(wq, num);      // キー配列に応じた数値変換

// pushedの立ち上がり検出用遅延
always @(posedge clk) begin
    pushed_d <= pushed;
end
```

```

// 新しいキーが押されたときのみ履歴をシフト
always @(posedge clk) begin
    if (!pushed_d && pushed) begin
        rdata5 <= rdata4;
        rdata4 <= rdata3;
        rdata3 <= rdata2;
        rdata2 <= rdata1;
        rdata1 <= rdata0;
        rdata0 <= num;      // 最新の入力
    end
end

wire [7:0] dec0, dec1, dec2, dec3, dec4, dec5;

// 各履歴データを7セグ表示用にデコード
m_7segment u0 (rdata0, dec0);
m_7segment u1 (rdata1, dec1);
m_7segment u2 (rdata2, dec2);
m_7segment u3 (rdata3, dec3);
m_7segment u4 (rdata4, dec4);
m_7segment u5 (rdata5, dec5);

assign LED = {6'd0,wq};      // 押下キーをLEDで表示
assign HEX0 = dec0;
assign HEX1 = dec1;
assign HEX2 = dec2;
assign HEX3 = dec3;
assign HEX4 = dec4;
assign HEX5 = dec5;

endmodule

```

## 動作確認

- ボタンを押すと、7セグメントLEDに押したキーの番号が表示されることを確認した。
- 直近6回分の入力が左から新しい順に表示されることを確認した。
- 左にシフトしていくことを確認した。

## 解説

マトリクスキーで入力された数値を順に記録し、直近6回分の入力を7セグメントLEDに表示する構成になっている。

まず `m_prescale` により基板のクロックを 100Hz に分周し、マトリクスキーの走査や入力処理を人間にとつて安定して扱える速度にしている。`m_matrix_key` は列を順番に Low にしながら行の入力を読み取り、16個のキー状態を 16bit の `key` 信号として出力する。

`m_dec16to4` では `key` のうち、1つだけ押されている場合にその位置を 4bit の `wq` として取り出し、同時に `pushed` 信号で打鍵を検出する。`m_convert_num` はマトリクスキーの物理配置に合わせて、`wq` を実際に表

示したい数値へ変換する役割を持つ。

入力履歴の更新では `pushed` の立ち上がりのみを検出するため、`pushed_d` に1クロック遅延させた信号を用いている。`pushed` が 0 から 1 になった瞬間だけ、過去のデータを `rdata1`～`rdata5` にシフトし、最新の入力を `rdata0` に保存する。これによりキーを押し続けても同じ値が連続して記録されることを防いでいる。

最後に `rdata0` から `rdata5` に保存された6個のデータをそれぞれ7セグメントデコーダに渡し、HEX0 から HEX5 に表示する。これにより、マトリクスキーの入力履歴が左から新しい順に並んで表示される。

# 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キー一分)
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
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
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
                    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
                    state <= 2; // イコール
                    case (op)
                        2'b00: bin_ans <= bin1 + bin2; // 加算
                        2'b01: bin_ans <= (bin1>=bin2)?(bin1-bin2):0;
                end
// 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;
        end
    end
end
```

```

        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。