

# 情報工学実験 1 レポート

FPGA

AJG23055 牧野唯希

2025/07

## A. 目的

参考資料に記載のため省略

## B. 解説

参考資料に記載のため省略

## C. 使用機器

FPGA : MAX 10 10M50DAF484C7G

コンパイル環境 : Quartus Prime

設計言語 : HDL

## D. 実験

### 1. 練習 1 7 セグメントでコードの設計

Quartus Prime 上で新しいプロジェクトを作成し、「PROGRAM1」というプロジェクト名を付けて、使用する FPGA を選択し、AJG23055 フォルダ内の Project1 フォルダに保存した。

次に、デスクトップ上の「DE10-Lite pin.qsf」を用いてピンアサインを行った。

その後、「program1.v」という名前の HDL ファイルを Project1 の中で作成し、以下のように記述した。

```
module PROGRAM1 (
    input          [3:0] SW,
    output reg [6:0] HEX0
);
always @* begin
    case( SW )
        4'h0:    HEX0 = 7'b1000000;
        4'h1:    HEX0 = 7'b1111001;
        4'h2:    HEX0 = 7'b0100100;
        4'h3:    HEX0 = 7'b0110000;
        4'h4:    HEX0 = 7'b0011001;
        4'h5:    HEX0 = 7'b0010010;
        4'h6:    HEX0 = 7'b0000010;
        4'h7:    HEX0 = 7'b1011000;
        4'h8:    HEX0 = 7'b0000000;
        4'h9:    HEX0 = 7'b0010000;
        4'ha:    HEX0 = 7'b0001000;
        4'hb:    HEX0 = 7'b0000011;
        4'hc:    HEX0 = 7'b1000110;
    endcase
end
```

```

4'h0:      HEX0 = 7'b0100001;
4'h1:      HEX0 = 7'b0000110;
4'h2:      HEX0 = 7'b0001110;
4'h3:      HEX0 = 7'b1111111;
          default:
          HEX0 = 7'b1111111;

          endcase
end
endmodule

```

このプログラムでは SW の値に応じて、LED セグメントの光る場所を変化させることで 0 から f までの数字を表現している。

このファイルをコンパイルし、コンフィグレーションした結果以下のように SW0 から SW3 の 4 つのスイッチをスライドさせることで 16 進数を HEX0 上に表示することが出来た。

以下にその一部を示す。

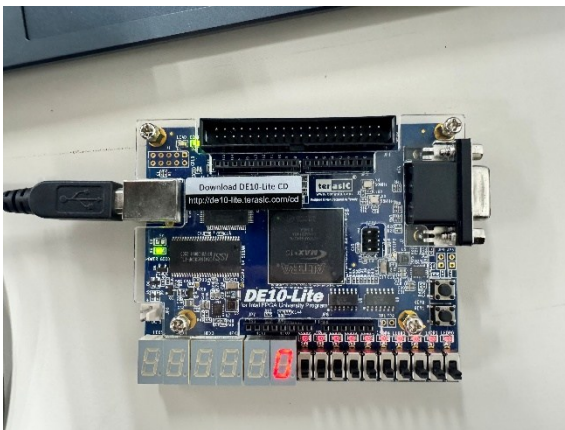


図 1 0000 の場合

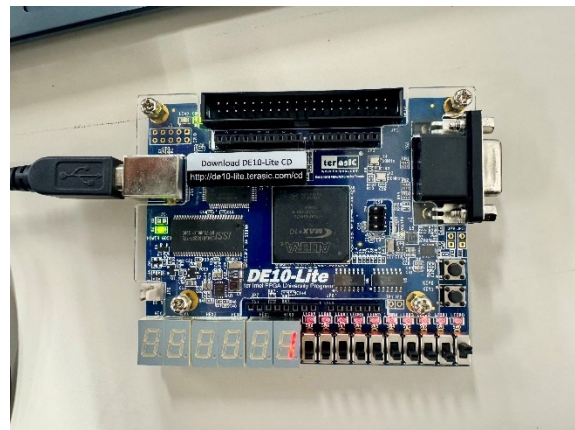


図 2 0001 の場合

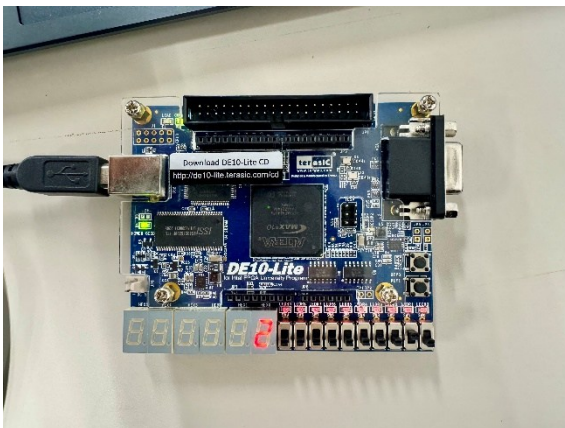


図 3 0010 の場合

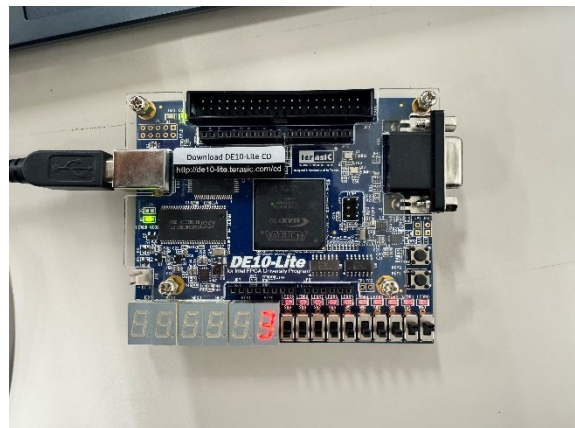


図 4 0011 の場合

## 2. 練習 2 10 秒カウンタ

先ほどと同様、Quartus Prime 上で、「SEC10」というプロジェクト名を付けて、使用する FPGA を選択し、AJG23055 フォルダ内の Project2 フォルダに保存した。

次に、デスクトップ上の「DE10-Lite pin.qsf」を用いてピンアサインを行った。

その後、「sec10.v」という名前の HDL ファイルを Project2 の中で作成し、以下のよう

```
module SEC10 (
    input          CLK, RST,
    output reg [6:0] HEX0
);

reg [25:0] cnt;
wire enlhz = (cnt==26'd49_999_999);

always @( posedge CLK ) begin
    if ( RST )
        cnt <= 26'b0;
    else if ( enlhz )
        cnt <= 26'b0;
    else
        cnt <= cnt + 26'b1;
end

reg [3:0] sec;

always @( posedge CLK ) begin
    if ( RST )
        sec <= 4'h0;
    else if ( enlhz )
        if ( sec==4'h9 )
            sec <= 4'h0;
        else
            sec <= sec + 4'h1;
end

always @* begin
    case ( sec )
        4'h0:    HEX0 = 7'b1000000;
        4'h1:    HEX0 = 7'b1111001;
        4'h2:    HEX0 = 7'b0100100;
        4'h3:    HEX0 = 7'b0110000;
        4'h4:    HEX0 = 7'b0011001;
        4'h5:    HEX0 = 7'b0010010;
        4'h6:    HEX0 = 7'b0000010;
        4'h7:    HEX0 = 7'b1011000;
        4'h8:    HEX0 = 7'b0000000;
        4'h9:    HEX0 = 7'b0010000;
        default: HEX0 = 7'bxxxxxxx;
    endcase
end

endmodule
```

このプログラムでは、50MHz のクロック周波数を持つ FPGA に対して、1Hz で動作するイネーブル信号を生成する。具体的には 50Mhz のクロック周波数のカウント値が上限に達したら、enlhz を 1 にすることで、enlhz が 1Hz で動くようにしている。また、sec が上限の 9 に達した場合は 0 に戻るように記述している。今回の回路ではクロックを用いているため、同時に制約ファイル「sec10.sdc」を作成した。そのプログラムを以下に示す。

```
create_clock -name CLK -period 20.000 [get_ports {CLK}]
derive_clock_uncertainty
set_input_delay -clock {CLK} 1 [all_inputs]
set_output_delay -clock {CLK} 1 [all_outputs]
```

このプログラムにより、コンパイル時にタイミング解析が行われ 50Mhz で動作するように回路が生成される。

「sec10.v」と制約ファイルの「sec10.sdc」を追加してコンパイルしコンフィグレーションを行うと以下のように、0 から 1 秒ごとに値が増加し、9 の次は 0 に戻る 10 秒カウンタを作成することが出来た。

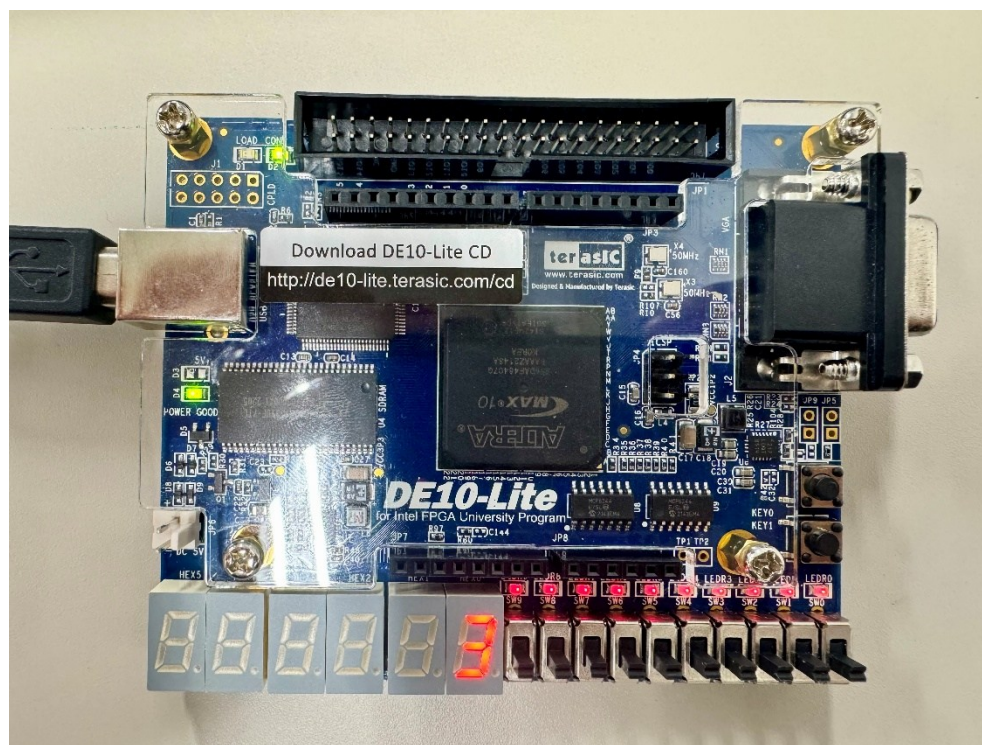


図 5 三秒時点の場合

また、SW9 を用いてリセットを行うことで値が 0 に戻ることも確認した。

### 3. 練習 3 アップダウンカウンタ

練習 1、2 と同様に、「UPDOWN」というプロジェクト名を付けて、使用する FPGA を選択し、AIG23055 フォルダ内の Project3 フォルダに保存した。

次に、デスクトップ上の「DE10-Lite pin.qsf」を用いてピンアサインを行った。

次に updown.v, BTN\_IN.v という 2 つの HDL ファイルを Project3 の中で作成し、それぞれ以下のように記述した。

<updown.v>

```
module UPDOWN(
    input    CLK,      RST,
    input    [1:0] KEY,
    output   reg [6:0] HEX0
);

wire        reset, down,      up;
BTN_IN BTN_IN(CLK, RST, KEY, {reset, down, up} );

reg [3:0] udcnt;
always @( posedge CLK, posedge RST) begin
    if(RST)
        udcnt <= 4'h0;
    else if(up)
        udcnt <= udcnt + 4'h1;
    else if(down)
        udcnt <= udcnt - 4'h1;
end

always @* begin
    case (udcnt)
        4'h0:      HEX0 = 7'b1000000;
        4'h1:      HEX0 = 7'b1111001;
        4'h2:      HEX0 = 7'b0100100;
        4'h3:      HEX0 = 7'b0110000;
        4'h4:      HEX0 = 7'b0011001;
        4'h5:      HEX0 = 7'b0010010;
        4'h6:      HEX0 = 7'b0000010;
        4'h7:      HEX0 = 7'b1011000;
        4'h8:      HEX0 = 7'b0000000;
        4'h9:      HEX0 = 7'b0010000;
        4'ha:      HEX0 = 7'b0001000;
        4'hb:      HEX0 = 7'b0000011;
        4'hc:      HEX0 = 7'b1000110;
        4'hd:      HEX0 = 7'b0100001;
        4'he:      HEX0 = 7'b0000110;
        4'hf:      HEX0 = 7'b0001110;
        default:   HEX0 = 7'bxxxxxxx;
    endcase
end

endmodule
```

< btn\_in.v >

```
module BTN_IN(
    input          CLK, RST,
    input [2:0]    nBIN,
    output reg[2:0] BOUT
);
reg[20:0] cnt;
wire en40hz = (cnt==125000-1);

always @(posedge CLK, posedge RST) begin
    if(RST)
        cnt <= 21'b0;
    else if(en40hz)
        cnt <= 21'b0;
    else
        cnt <= cnt + 21'b1;
end

reg [2:0] ff1, ff2;

always @(posedge CLK, posedge RST) begin
    if(RST) begin
        ff2 <= 3'b0;
        ff1 <= 3'b0;
    end
    else if(en40hz) begin
        ff2 <= ff1;
        ff1 <= nBIN;
    end
end

wire [2:0] temp = ~ff1 & ff2 & {3{en40hz}};

always @(posedge CLK, posedge RST)begin
    if (RST)
        BOUT <= 3'b0;
    else
        BOUT <= temp;
end

endmodule
```

btn\_in.v では、ボタンのノイズを除去し、押された瞬間だけを検出するようにし、40Hz の周期でサンプリングすることで安定したボタン入力処理を可能にしている。

また、updown.v では、KEY の入力によって、カウントがアップされたりダウンする処理を実現している。

#### 4. 任意回路の作成 タイマー

作成に当たり、タイマーの作成を試みた。最初は KEY を押すことで時間を調節できるタイマーの作成に挑んだが、間に合わず、1 分タイマーの作成を行った。

まず、これまでと同様に「PROGRAM4」というプロジェクト名を付けて、使用する FPGA を選択し、AJG23055 フォルダ内の Project4 フォルダに保存した。

次に、デスクトップ上の「DE10-Lite pin.qsf」を用いてピンアサインを行った。

その後、program4.v を以下のように作成した。

```
module PROGRAM4 (
    input          CLK, RST,
    input  [1:0]    KEY,
    input  [2:0]    SW,
    output reg [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5
);

reg  [25:0] cnt;
wire enlhz = (cnt==26'd49_999_9);

always @( posedge CLK ) begin
    if ( RST )
        cnt <= 26'b0;
    else if ( enlhz )
        cnt <= 26'b0;
    else if(SW==1)
        cnt <= cnt;
    else
        cnt <= cnt + 26'b1;
end

reg [3:0] sec;
reg [3:0] sec10;
reg [3:0] min;
reg [3:0] min10;
reg [3:0] min100;
reg [3:0] min1000;

reg [3:0] i;

always @( posedge CLK ) begin
    if(i==0) begin
        sec <= 4'h0;
        sec10 <= 5'h0;
        min <= 6'h0;
        min10 <= 7'h0;
        min100 <= 8'h1;
        min1000 <= 9'h0;
        i <= i + 4'h1;
    end
    else begin
        if ( RST ) begin
            i <= 4'h0;
            sec <= 4'h0;
            sec10 <= 5'h0;
            min <= 6'h0;
        end
    end
end
```

```

min10 <= 7'h0;
min100 <= 8'h0;
min1000 <= 9'h0;

end
else if ( enlhz )begin
    if ( sec==4'h0 )begin //4'h9
        if ( sec10==5'h0 )begin //5'h9
            if ( min==6'h0 )begin //6'h9
                if (min10==7'h0)begin
                    //7'h5

                    if(min100==8'h0)begin //8'h9

                    if(min1000==9'h0)begin //9'h9

                    sec <= 4'h0;

                    sec10 <= 5'h0;

                    min <= 6'h0;

                    min10 <= 7'h0;

                    min100 <= 8'h0;

                    min1000 <= 9'h0;

end
else
begin

    sec <= 4'h9;

    sec10 <= 5'h9;

    min <= 6'h9;

    min10 <= 7'h5;

    min100 <= 8'h9;

    min1000 <= min1000 - 9'h1;

end
end
else begin
    sec <=
    sec10
    min <=
    min10
    min100

end
end
else begin
    sec <= 4'h9;
    sec10 <= 5'h9;
    min <= 6'h9;
end
end

```

```

min10 <= min10 -
7'h1;

end
end
else begin
    sec <= 4'h9;
    sec10 <= 5'h9;
    min <= min - 6'h1;
end
end
else begin
    sec <= 4'h9;
    sec10 <= sec10 - 5'h1;
end
end
else
    sec <= sec - 4'h1;
end
end
end
end
always @* begin
    case ( sec )
        4'h0:    HEX0 = 7'b1000000;
        4'h1:    HEX0 = 7'b1111001;
        4'h2:    HEX0 = 7'b0100100;
        4'h3:    HEX0 = 7'b0110000;
        4'h4:    HEX0 = 7'b0011001;
        4'h5:    HEX0 = 7'b0010010;
        4'h6:    HEX0 = 7'b0000010;
        4'h7:    HEX0 = 7'b1011000;
        4'h8:    HEX0 = 7'b0000000;
        4'h9:    HEX0 = 7'b0010000;
        default: HEX0 = 7'bxxxxxxx;
    endcase

    case ( sec10 )
        5'h0:    HEX1 = 7'b1000000;
        5'h1:    HEX1 = 7'b1111001;
        5'h2:    HEX1 = 7'b0100100;
        5'h3:    HEX1 = 7'b0110000;
        5'h4:    HEX1 = 7'b0011001;
        5'h5:    HEX1 = 7'b0010010;
        5'h6:    HEX1 = 7'b0000010;
        5'h7:    HEX1 = 7'b1011000;
        5'h8:    HEX1 = 7'b0000000;
        5'h9:    HEX1 = 7'b0010000;
        default: HEX1 = 7'bxxxxxxx;
    endcase

    case ( min )
        6'h0:    HEX2 = 7'b1000000;
        6'h1:    HEX2 = 7'b1111001;
        6'h2:    HEX2 = 7'b0100100;
        6'h3:    HEX2 = 7'b0110000;
        6'h4:    HEX2 = 7'b0011001;
        6'h5:    HEX2 = 7'b0010010;
        6'h6:    HEX2 = 7'b0000010;
        6'h7:    HEX2 = 7'b1011000;
    endcase
end

```

```

        6'h8:          HEX2 = 7'b0000000;
        6'h9:          HEX2 = 7'b0010000;
        default:      HEX2 = 7'bxxxxxxx;
    endcase

    case ( min10 )
        7'h0:          HEX3 = 7'b1000000;
        7'h1:          HEX3 = 7'b1111001;
        7'h2:          HEX3 = 7'b0100100;
        7'h3:          HEX3 = 7'b0110000;
        7'h4:          HEX3 = 7'b0011001;
        7'h5:          HEX3 = 7'b0010010;
        7'h6:          HEX3 = 7'b0000010;
        7'h7:          HEX3 = 7'b1011000;
        7'h8:          HEX3 = 7'b0000000;
        7'h9:          HEX3 = 7'b0010000;
        default:      HEX3 = 7'bxxxxxxx;
    endcase

    case ( min100 )
        8'h0:          HEX4 = 7'b1000000;
        8'h1:          HEX4 = 7'b1111001;
        8'h2:          HEX4 = 7'b0100100;
        8'h3:          HEX4 = 7'b0110000;
        8'h4:          HEX4 = 7'b0011001;
        8'h5:          HEX4 = 7'b0010010;
        8'h6:          HEX4 = 7'b0000010;
        8'h7:          HEX4 = 7'b1011000;
        8'h8:          HEX4 = 7'b0000000;
        8'h9:          HEX4 = 7'b0010000;
        default:      HEX4 = 7'bxxxxxxx;
    endcase

    case ( min1000 )
        9'h0:          HEX5 = 7'b1000000;
        9'h1:          HEX5 = 7'b1111001;
        9'h2:          HEX5 = 7'b0100100;
        9'h3:          HEX5 = 7'b0110000;
        9'h4:          HEX5 = 7'b0011001;
        9'h5:          HEX5 = 7'b0010010;
        9'h6:          HEX5 = 7'b0000010;
        9'h7:          HEX5 = 7'b1011000;
        9'h8:          HEX5 = 7'b0000000;
        9'h9:          HEX5 = 7'b0010000;
        default:      HEX5 = 7'bxxxxxxx;
    endcase

end
endmodule

```

HEX0 から HEX5 までの 6 つの LED セグメントについて、0 の次は 9 になるようにしてカウントダウンを行い、カウント値が上限に達するタイミングを変更することで HEX0 と HEX1 はそれぞれ少数第 2 位、小数第 1 位の値を表示するようにした。初期値の値を変えることで 100 分以下の任意の時間からのタイマーを起動することが

出来るようにした。さらに、SW0 を ON にすることで、カウントが止まる機能も備えた。また、課題 2 と同じ内容の制約ファイルを読み込みコンパイルとコンフィグレーションした結果以下のように動作した。

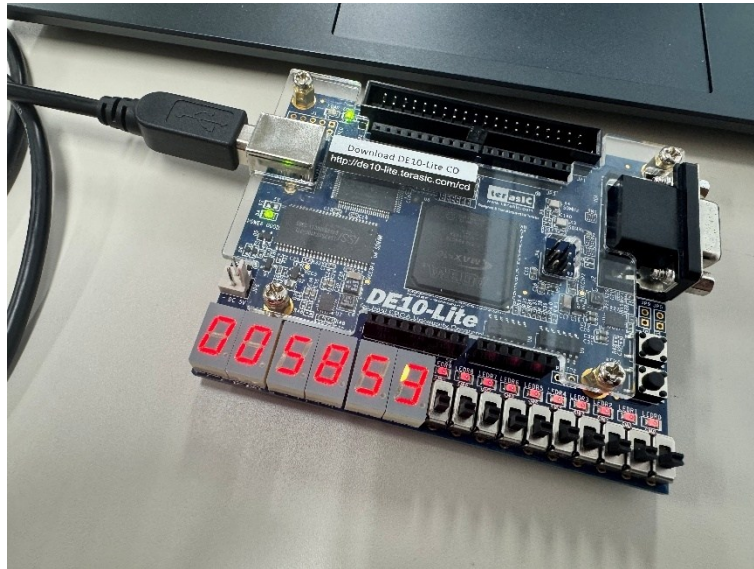


図 6 1 分タイマーの動作

ただしシャッタスピードの関係で HEX0 のライトが不自然に見える。

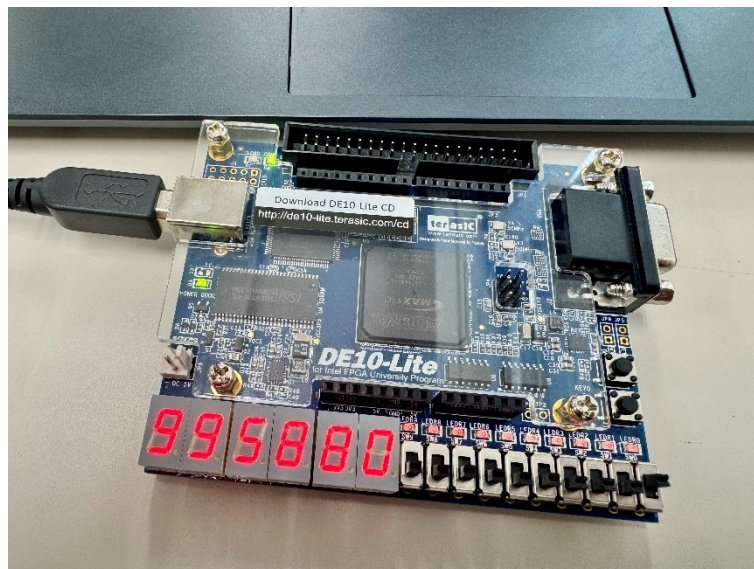


図 7 100 分タイマーの動作

また、タイマーの時間を延ばした場合も問題なく動作することが確認できた。

## E. 課題

### 1. FPGA がどのような用途に使用されているのか調べよ。

FPGA は設計した回路を電氣的に書き込むことが出来る集積回路で、自由に様々な回路を作成することが出来る。そのため、液晶パネルや通信機器、メモ리카ード、パソコンなど様々な電子機器に採用されており、特にスイッチや音、映像を制御するために組み込まれている。

### 2. 練習 1 の 7 セグメントデコーダにおいて、「reference.pdf」の表 2-2 の真理値表から、図 4 の回路図の導出を、(カルノー図等 (この他でも良い) を用いて) レポートに記すこと。

真理値表から g から a までのカルノー図を作成すると以下ようになる。

表 1 カルノー図 g

cd \ ab	00	01	11	10
00	1	1	0	0
01	0	0	1	0
11	1	0	0	0
10	0	0	0	0

表 3 カルノー図 e

cd \ ab	00	01	11	10
00	0	1	1	0
01	1	1	1	0
11	0	0	0	0
10	0	1	0	0

表 2 カルノー図 f

cd \ ab	00	01	11	10
00	0	1	1	1
01	0	0	0	0
11	0	1	0	0
10	0	0	0	0

表 4 カルノー図 d

cd \ ab	00	01	11	10
00	0	1	0	0
01	1	0	1	0
11	0	0	1	0
10	0	0	0	1

表5 カルノー図 c

cd \ ab	00	01	11	10
00	0	0	0	1
01	0	0	0	0
11	1	0	1	1
10	0	0	0	0

表7 カルノー図 a

cd \ ab	00	01	11	10
00	0	1	0	0
01	1	0	0	0
11	0	1	0	0
10	0	0	1	0

表6 カルノー図 b

cd \ ab	00	01	11	10
00	0	0	0	0
01	0	1	0	1
11	1	0	1	1
10	0	0	1	0

これらのカルノー図から以下のような論理式が求められる。

$$a = \overline{a}bcd + a\overline{b}cd + ab\overline{c}d$$

$$b = bc\overline{d} + ab\overline{d} + acd + \overline{a}b\overline{c}d$$

$$c = abc + ab\overline{d} + \overline{a}bc\overline{d}$$

$$d = \overline{a}bcd + \overline{a}b\overline{c}d + \overline{a}bcd + a\overline{b}c\overline{d} + abcd$$

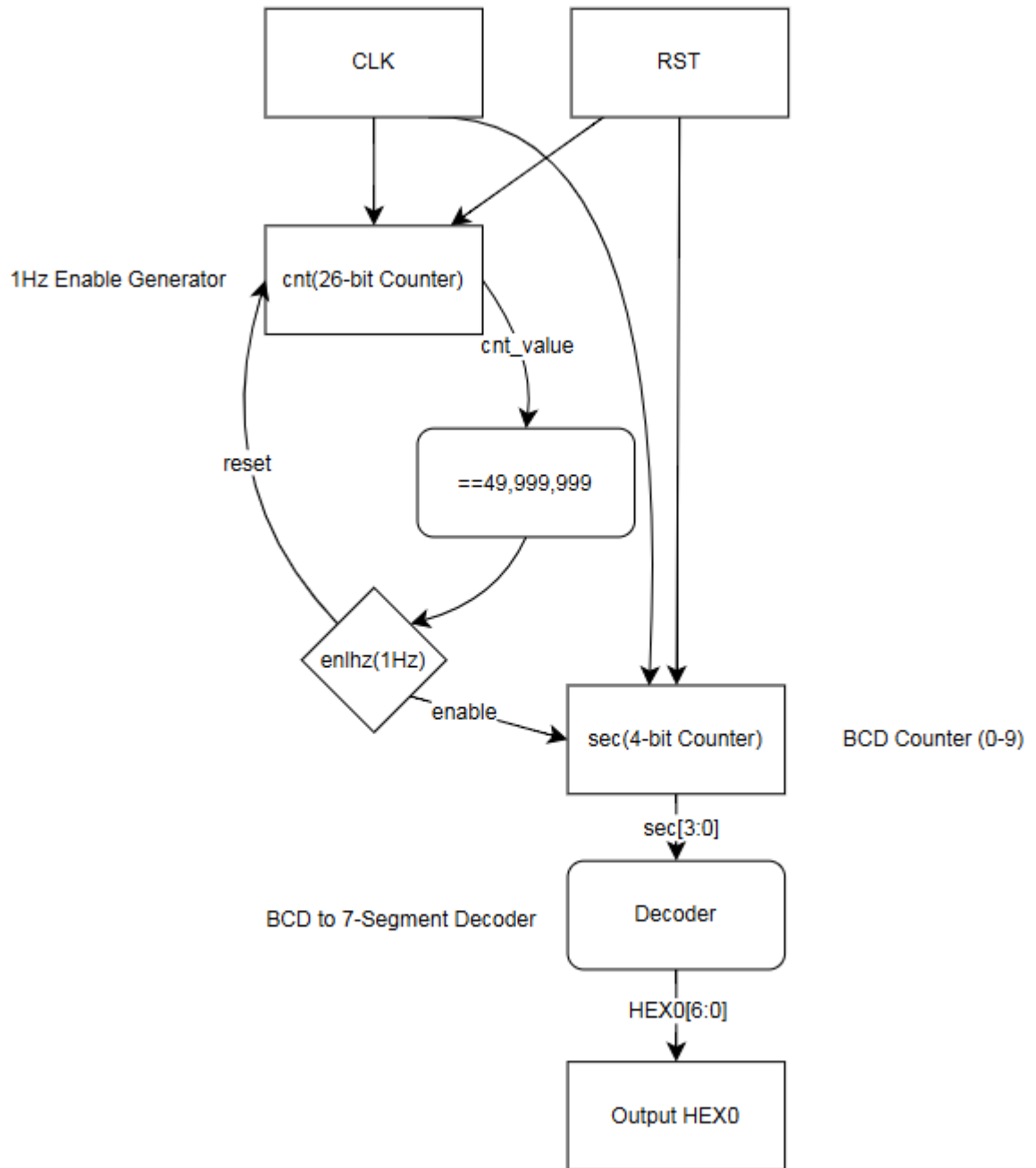
$$e = \overline{a}d + \overline{a}b\overline{c}d + a\overline{b}cd$$

$$f = \overline{a}bd + \overline{a}bc + ab\overline{c}d$$

$$g = \overline{a}bc + \overline{a}bcd + ab\overline{c}d$$

この論理式を満たすような回路図を論理和や論理積などを用いることで、回路図を作成することが出来る。ただし回路図は、参考資料に記載のため省略する。

3. 4.2 で作成した回路の想定回路図，動作確認の手順及び結果を報告せよ。  
 4.2 で作成した 1 秒カウンターが想定する回路図は以下の通りである。



動作確認の手順は、作成した `sec10.v` と制約ファイル `sec10.sdc` をコンパイルし、コンフィグレーションした。その結果 1 秒ごとにカウントがあがり、9 秒になると次は 0 秒に戻る動作が確認できた。また、ピンアサインで指定した `SW9` をオンにすると値が

0 にリセットされた。

4. 各項目において、苦労した点や工夫した点などを報告せよ。

課題 1 では、まだ Quartus Prime の使い方が分かっておらず、ファイルをうまく読み込むことが出来ていなかった。課題 2 では、制約ファイルを含めてしっかりと追加し、実装することが出来た。課題 3 では、KEY 入力を行うことでアップダウンカウンタを作成することはすぐに理解できたが、記述するコードにミスがあったため、少し時間がかかってしまった。課題 4 では、まず、桁数を増やすところから実装を行い、6 桁まで増えるカウンタを作成した。その際に、現実のタイマーのように 1 秒ごとに増えるのではなく、0.01 秒単位に値が増えるように工夫した。さらに、SW1 を押すことで時間を計ることが出来るストップウォッチ機能を実装した。

次にカウントがダウンされるように実装を変更し、最後に任意の時間を計ることが出来るタイマーを作成した。その後、アップダウンカウンタのコードを用いて計測時間を指定できるようにしようとしたが、アップダウンカウンタで指定した数字をカウンタの初期値としてうまく実装させることが出来ずに断念した。また、タイマーの表示を見やすくするために小数点を点灯させようとしたが、方法が分からず断念した。

F. 検討考察

今回の実験を通して、FPGA に HDL を書き込むことで回路が変更されていく様子を体験することが出来た。同時に、HDL の記述の複雑さも感じる事が出来た。私は今まで部活動やバイトで Micro:bit や Raspberry Pi に軽く触れたことがあったので、FPGA にも大きな抵抗がなく取り組むことが出来た。しかし、通常のプログラムと異なり、数字を表示させるだけでも、多くの記述が必要だというように煩雑さを大いに実感した。今回実装できなかったことが多数あったので、時間と機会があればまた触れてみたいと感じた。

また、FPGA について調べるにあたり、FPGA は並列処理が得意なため CPU よりもディープラーニングに適しているという記述があったので、これから AI がより活用されていく中で注目度がより増していくと感じました。

G. 参考文献

井上 勝文. FPGA の回路設計. 情報工学実験 1. 2025

きたみりゅうじ. キタミ式イラスト IT 塾応用情報技術者 令和 07 年. 2025. p156

FPGA とは？マイコン・CPU との違いやできること・使用例をわかりやすく説明. ぱーそろクロステクノロジー. 2024/09/23. <https://staff.persol->

[xtech.co.jp/hatalabo/mono\\_engineer/700.html#\\_2](https://xtech.co.jp/hatalabo/mono_engineer/700.html#_2). (参照 2025/07/12)

FPGA とは？メリットや注目される理由もわかりやすく解説. NEC ソリューションイノベーター. <https://www.nec-solutioninnovators.co.jp/sl/emb/column/04/>. (参照 2025/07/12)