目次

1　実験の目的

ハードウェア記述言語（HDL）を用いて論理回路を設計する手法の収得

2　実験器具及びツール

Quartus (Quartus Prime 20.1) Lite Edition

3　実験内容

3.1　実習 1

各入力が 3 ビット幅の 5 入力を選択する 5-to-1 マルチプレクサのモジュールを VerilogHDL で記述し、機能シミュレータにより動作を確認した。

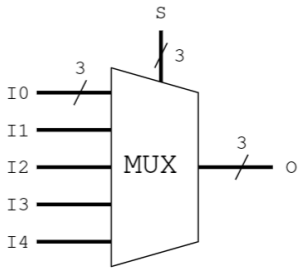

図 1　5to1 マルチプレクサ

3.2　実習 2

ラッチを 2 つ接続することでマスタスレーブ型 D-FF を VerilogHDL で記述し機能シミュレータで動作を確認した

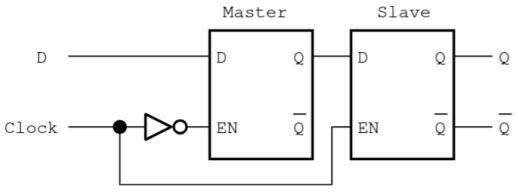

図 2　マスタースレーブ型 DFF

。

表 1　可変長とコード

文字	コード
A	00
B	01
C	100
D	101
E	110
F	1110
_	1111

3.3　実習 3

入力されたビット列を表 1 に基づいてデコードしもとの文字を 7 セグメントディスプレイに表示する回路を設計し、VerilogHDL で記述した。

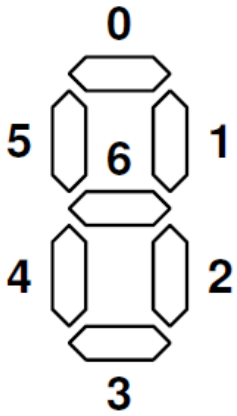

図 3　7 セグメントディスプレイ

4　実験結果

4.1　実習 1

VerilogHDL でコード 11 のように記述した。コード 1 はコード 11 から一部抜粋したものである。

ソースコード 1　実習 1 の一部抜粋

```
1  function [2:0] z1;
2      input [2:0] i_0, i_1, i_2, i_3, i_4, s1;
3    begin
4      case (s1)
5        3'b000: z1 = i_0;
```

```
6           3'b001: z1 = i_1;
7           3'b010: z1 = i_2;
8           3'b011: z1 = i_3;
9           3'b100: z1 = i_4;
10          default: z1 = 3'bxxx;
11      endcase
12    end
13  endfunction
```

図 1 のように、input は 3 ビット幅の I0 から I4 と S であり、output を Q を用意する。次に MUX で S の値によって I0 から I4 を選択し、outputQ に出力されるよう assign でつないだ。

機能シミュレータの結果は図 4 のようになった。
I0,I1,I2,I3,I4 を 0b0,0b1,0b10,0b11,0b100 と数値を設定する S を 0b0,0b1,0b10,0b11,0b100 と変化させると、Q は 0b0,0b1,0b10,0b11,0b100 となっている。このことから Q が I0,I1,I2,I3,I4 の値に変化していることがわかる。ゆえに、5-to-1 マルチプレクサを作成できたと言える。

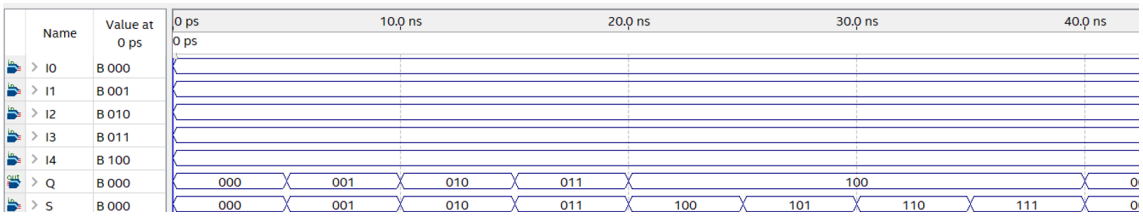

図 4　実習 1 における結果

4.2　実習 2

VerilogHDL でコード 12 のように記述した。
コード 2 はコード 12 から一部抜粋したものである。

ソースコード 2　実習 2 の一部抜粋

```
1   module Zissyu2(EN, D, Q);
2       input EN, D;
3       output Q;
4       wire W1, W2;
5
6       d_latch master_latch(~EN, D, W1);
7       d_latch slave_latch(EN, W1, W2);
8       assign Q = W2;
9   endmodule
```

まず、ラッチを定義しモジュール化を行った。つづいてそのモジュールの 2 つの実体を作成した。このとき、図 2 のように Master の input EN は Clock の反転が挿入されている。また、Slave の inputD では Master の outputQ がつながっていることがわかるので、実体を作成したときは、input 先に注意してコード

を作成した。

機能シミュレータの結果は図 5 のようになった。
EN の立ちが上がりの時にQの値がDの値に変化し、次の EN の立ち上がりまで保持されていることがこの結果から分かる。ゆえに、この回路でマスタスレーブ型 D-FF を作成できたと言える。

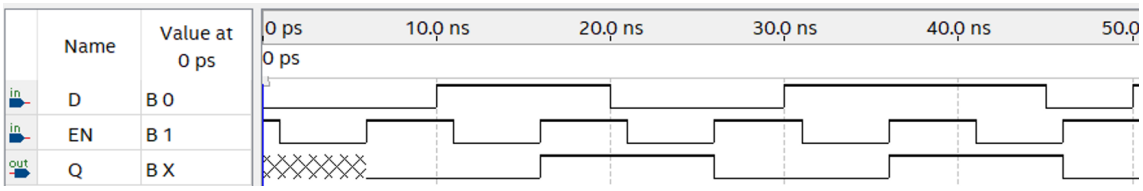

図 5　実習 2 における結果

4.3　実習 3

まず、状態遷移図を作成した。(図 6)

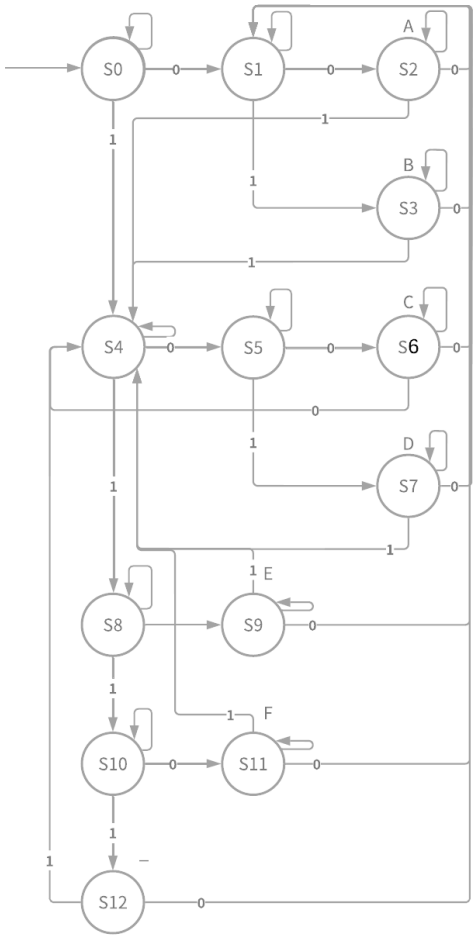

図 6　実習 3 における状態遷移図

この状態遷移図を基に VerilogHDL でコード 13 のように記述した。
コード 3 はコード 13 から一部抜粋したものである

ソースコード 3　実習 3 の一部抜粋

```
1  module Zissyu3(Clk, I, Resetn, Q);
2      input Clk, I, Resetn;
3      output [6:0]Q;
4
5      parameter S0 = 4'b0000,
6                S1 = 4'b0001,
7                S2 = 4'b0010,
8                ...
9                S12 = 4'b1100;
10     reg [3:0] cur_st, next_st;
11
12     assign Q =
13         (cur_st == S2) ? 7'b0001000 : (
```

```
14          (cur_st == S3) ? 7'b0000011 : (
15          ...
16          (cur_st == S12) ? 7'b1111110: 7'b1111111))))));
17
18
19      always @(posedge Clk)
20          if(!Resetn)
21              cur_st <= S0;
22          else
23              cur_st <= next_st;
24
25      always @(cur_st or I)
26          case(cur_st)
27              S0: next_st = (I) ? S4 : S1;
28              S1: next_st = (I) ? S3 : S2;
29              S2: next_st = (I) ? S4 : S1;
30              ...
31              S12: next_st = (I) ? S4 : S1;
32              default: next_st = S0;
33          endcase
34  endmodule
```

次に各状態を parameta として１２個用意した。そして、トリガーとして Clk のポジティブエッジのとき、Resetn が 0 のときは初期状態 S0 に遷移し、Resetn が 1 の時は状態を次状態に移すように設定した。そして状態が遷移したとき、すなわち Clk の立ち上がりタイミングもしくは inputI が変化したとき状態をパラメータで定義した状態に写した。例えば、S0 のとき次状態は I が 1 のときは S4 へ移動し、I が 0 のときは S1 に移動する。各状態についてそれぞれ遷移先をつなげてやる。最後はその状態に応じて７セグメントディスプレイへの応答へデコードさせて Q に出力させた。

機能シミュレータの結果は図７のようになった。

Clk の立ち上がりのタイミングは 6ps, 16ps, 26ps, 36ps, 46ps, 56ps, ... と 10ps 周期である。このとき、I の値は、1, 0, 1, 0, 0, ... となっていることが分かる。これを文字コード別に分けると、101, 00 となる。文字はＤとＡであり、７セグメントディスプレイで点灯、消灯を表す値がＱに出力されていることが分かる。同様に Clk の立ち上がり時の I の値は、0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, ... となっていた。これを文字コード別に分けると、01, 100, 00, 01, 01, となり、文字は B, C, A, B, B であった。Ｑの値は、0000011, 0000110, 0001000, 0000011, 0000011 となっており、これはその文字を７セグメントディスプレイに映したときの点灯、消灯位置と一致していた。このことより入力されたビット列を表に基づいてディコードし７セグメントディスプレイに表示する回路が作成できたといえる。

図７　実習３における結果

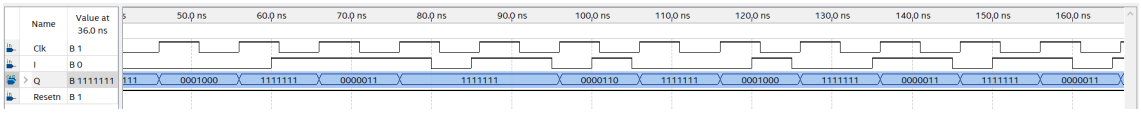

図 8　実習 3 における結果の拡大版

次に手書きで行った状態遷移図 (図 6) と Quartus Prime の機能の StateMachineViewr(図 9) の比較を行う。

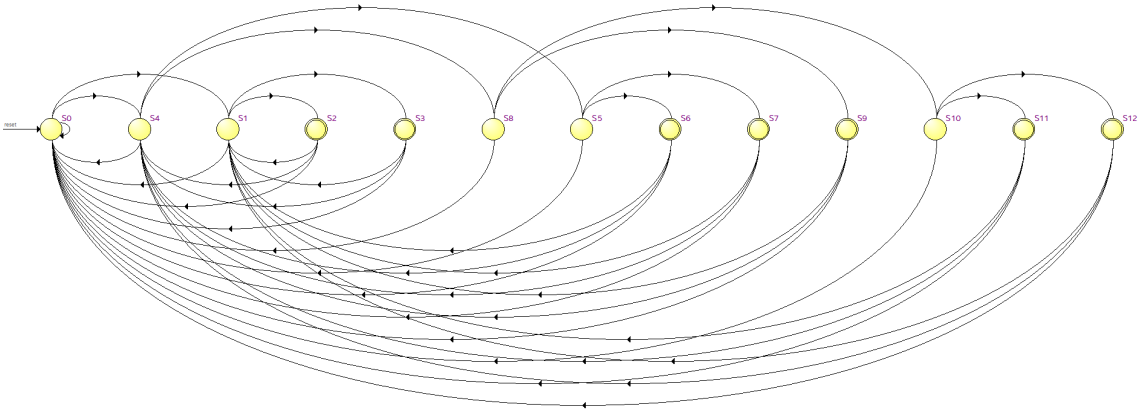

図 9　実習 3 における StateMachineViewer

まず状態数を確認すると、StateMachineViewr では S0 から S12 まで 1 3 状態あった。そして設計を行った状態遷移図でも同様に S0 から S12 まで 13 状態あったことがわかる。

次に、状態 S0 に注目してみる。S0 の遷移先は 3 つあり、S0 と S1 と S4 である。表 2 より遷移条件はそれぞれ、は Resetn が 0 のとき、inputI が 0 のとき、inputI が 1 のときである。次に、設計をおこなった状態遷移図 (図 6) を見てみると、S0 の遷移先は 3 つ存在し、S0,S1,S4 であった。遷移条件も StateMachineViewr と同じで、input が 0 の時は S1 で input が 1 のときは S4 であった.

これは S0 だけでなく、S1,S2,...S12 と以下同様に遷移先の数、遷移先の状態、遷移条件が StateMachineViewer と設計した状態戦時が一致していることが図 6 と図 9 より分かった。つまり、状態遷移図の形はことなるものの、表している内容は同じでなんら変わりはないことが分かる。

表 2　実習 3 における StateMachineViewr の遷移表

	Source State	Destination State	Condition
1	S0	S4	(I).(Resetn)
2	S0	S0	(!Resetn)
3	S0	S1	(!I).(Resetn)

5　考察

5.1　実習１の考察

実習１では module の作成方法、input, output,wire,reg の違い、function の書き方を学んだ。マルチプレクサを case 文を用いて作成できることを確認できた。また。条件演算子で書くと以下のようになる

ソースコード 4　実習１の考察

```
input [2:0] i_0, i_1, i_2, i_3, i_4, s1;
assign z1 =
  (s1 == 3'b000) ? i_0:(
  (s1 == 3'b001) ? i_1:(
  (s1 == 3'b010) ? i_2:(
  (s1 == 3'b011) ? i_3:(
  (s1 == 3'b100) ? i_4:))))
```

今回は case 文を用いるため function を用いたがその際 input を再宣言を行った。しかし条件演算子を用いることでその宣言を省くことができる。一方で、) の数が増え、可読性は低下すると考えられる。
そしてマルチプレクサの利用例も調べてみた。マルチプレクサとは、電子回路でよく MUX で省力されることがある。特に通信分野では、複数の信号のデータを 1 本の合成波にして、送信する場合に多く使用されている。一方で、一つの入力から複数の出力を行う信号器をデマルチプレクサと呼ぶ。[1]

5.2　実習２の考察

実習２で VerilogHDL でのラッチの描き方、モジュールのインスタンス化を学んだ。モジュールのインスタンス化を行うことで、可読性や流用度を上げ、作成の効率を上げることができると考えた。図 2 より考えてみると、Clock が 1 のとき Master の EN は 0 となり、Q は出力されない。Clock が 0 のときは Master は EN が 1 となり、Q が出力される。Master の output は Slave の D とつながっており、Q の値は D へ格納される。本来であれば、Clock が 0 のとき Slave の EN は 0 となり、Qは出力されることはないが、0 から 1 に立ち上がる瞬間においては一瞬 D の値が Master から入り Slave の outputQ へとつながる。このことで、master slave をつなげることで、ポジティブエッジを表すことができる。このことから、EN のタイミングを master と slave で逆にすることで、ネガティブエッジを作成できると考えられる。(コード 5)

ソースコード 5　実習２の考察

```
module Zissyu2(EN, D, Q);
    input EN, D;
    output Q;
    wire W1, W2;

    d_latch master_latch(EN, D, W1);
    d_latch slave_latch(~EN, W1, W2);
    assign Q = W2;
endmodule
```

5.3　実習３の考察

有限状態機械を状態線図をかき、VerilogHDL で書くことを学んだ。今回は電圧レベル L で点灯、電圧レベル H で消灯の指定の元行った。実験をしたときはこの仮定を間違えて、電圧レベル H で点灯、電圧レベル L で消灯としてしまった。1 と 0 を書き換えるだけであるが、自動変換できるようプログラムを組んだ。入力に 0101011 といれると 1010100 と返ってることがわかり、0 と 1 が反転できていることが確かめられる。電圧レベルが逆の仕様のときにはこのプログラムを用いるとうまくいくと考えられる。

ソースコード 6　実習 3 の考察

```
base = 0b1111111
num = input("2進数７桁を入力してください:")
num_b = int(num, 2)
print(format(num_b,'07b'))

output = base ^ num_b

print(format(output,'07b'))

/* 出力結果
2進数７桁を入力してください: 0101011
1010100
*/
```

6　演習課題

6.1　演習 1

4 ビットの値を 16 進数 1 桁で表示する 7 セグメントデコードモジュールを Verilog HDL でコード 14 のように記述した。コード 7 はコード 14 から一部抜粋したものである

ソースコード 7　演習 1 の一部抜粋

```
module Ensyu1(D,Q);
        input[3:0] D;
        output[6:0] Q;

        function [6:0] func_decoder;
                input [3:0] in;
                begin
                        case (in)]
                                4'b0000: func_decoder = 7'b0100000;//012345
                                4'b0001: func_decoder = 7'b1111001;//12
                                4'b0010: func_decoder = 7'b0100110;//01346
          ...
                                default: func_decoder = 7'b1111111;
                        endcase
```

```
15                  end
16          endfunction
17
18                  assign Q = func_decoder(D);
19  endmodule
```

input は 4 ビットの値なので 4 ビット幅で、output は 7 セグメントディスプレイに出力するので 7 ビット幅に設定した。そして、input の値に応じて出力を決めるので case 文を用いた。このとき assign でQに直接つなげることができないので、function を用いた。case 文では 7 セグメントディスプレイで点灯するところをＬレベルすなわち 0 にし、消灯するところをＨレベルすなわち 1 に設定した。

機能シミュレータの結果は図 7 のようになった。
input D の入力に応じて Q が変化していることがわかる。Dが 0001 のときはQは 1111001 となっている。これを 7 セグメントディスプレイで考えるとディスプレイの 1 番と 2 番が点灯し、他は消灯している状態である。つまり縦棒が右側に点灯している状態となるので 1 をディスプレイ上に示せたことになる。つまり 2 進数の入力 0001 をディスプレイで 1 と表示することができた。同様にDが 0010,0011,0100,... のとき Q は 0100110,0111000,0011011,... と出力された。これを 7 セグメントディスプレイ上で考えると、それぞれ、2,3,4,... という数字を表示できる。このことより入力された 4 ビットの値を 16 進数 1 桁で表示する 7 セグメントディスプレイをこの回路で作成できたといえる。

次に、記述した VerilogHDL モジュールの出力信号を FPGA ボートのどのピンに接続すればいいか見当した。出力信号とピン名の対応を表でまとめた。(表 4)。信号名に対応するピン番号は

表 3　Pin Assingment 7 Segment Display

Signal Name	*FPGA Pin No.*	*Description*	*I/O Standard*
HEX0[0]	PIN_AE26	Seven Segment Digit 0[0]	3.3V
HEX0[1]	PIN_AE27	Seven Segment Digit 0[1]	3.3V
HEX0[2]	PIN_AE28	Seven Segment Digit 0[2]	3.3V
HEX0[3]	PIN_AG27	Seven Segment Digit 0[3]	3.3V
HEX0[4]	PIN_AF28	Seven Segment Digit 0[4]	3.3V
HEX0[5]	PIN_AG28	Seven Segment Digit 0[5]	3.3V
HEX0[6]	PIN_AH28	Seven Segment Digit 0[6]	3.3V

表 4　演習 1 における出力とピン名の関係表

出力信号	ピン名
Q[0]	PIN_AE26
Q[1]	PIN_AE27
Q[2]	PIN_AE28
Q[3]	PIN_AG27
Q[4]	PIN_AF28
Q[5]	PIN_AG28
Q[6]	PIN_AH28

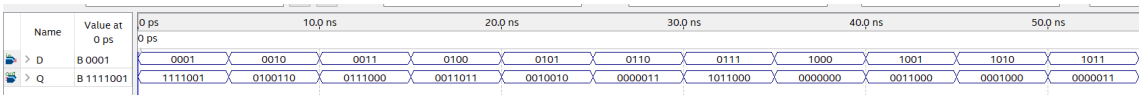

図 10　演習 1 における結果

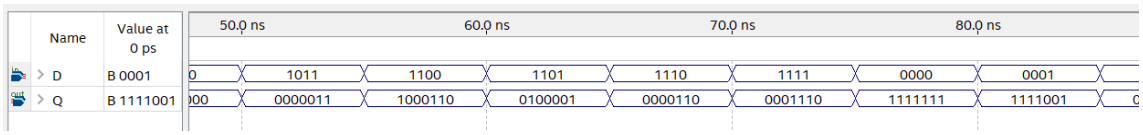

図 11　演習 1 における結果

6.2　演習 2

T-FF 及び JK-FF を VerilogHDL でコード 15, コード 16 のように記述した。

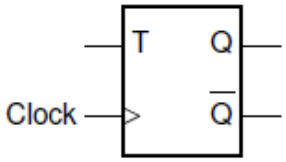

図 12　T-FF のブロック図

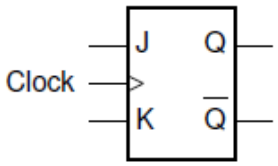

図 13　JK-FF のブロック図

まず、T-FF と JK-FF の真理値表を作成した。(表 5, 表 6)

表 5　T-FF の真理値表

T	Q	$\bar{Q}$
0	Q_0	$\bar{Q}_0$
1	$\bar{Q}_0$	Q_0

表 6　JK-FF の真理値表

J	K	Q	$\bar{Q}$
0	0	Q_0	$\bar{Q}_0$
0	1	0	1
1	0	1	0
1	1	$\bar{Q}_0$	Q_0

この表を基に、Verilog HDL を作成した。

コード 8 はコード 15 から一部抜粋したものである.

コード 9 はコード 16 から一部抜粋したものである.

- T-FF は T=0 のとき、出力は一つ前までの出力結果を保持するので、同時代入を用いてQを出力する。一方で、T=1 のときは出力波１つ前までの出力結果の反転なので反転した値を同時だ移入させる。そして、Qの値が変更するときは clk の立ち上がりタイミングとしてトリガーを用意しておくことで、T-FF を VerilogHDL で作成できた。

ソースコード 8　演習 2 の T-FF の一部抜粋

```
1  always @(posedge Clk)
2                  if(T == 0)
3                          Q <= Q;
4                  else
5                          Q <= ~Q;
```

JK-FF は JK が 0 と 1 の場合が $2^2 = 4$ 通りあるのでそれぞれ場合分けをして考える。まず JK が 0,0 のとき、JK が 1,1 のときは T-FF の T=0,T=1 の場合に相当するので同時代入を用いて作成した。続いて、JK が 0,1 のとき、Q は必ず 0 が出力される状態になるので、Qを０につなぐ。一方で JK が 1,0 の時は Q は必ず１が出力される状態になるので、Q を１につなぐ。このように４状態を場合分けを行って JK-FF を作成した。

ソースコード 9　演習 2 の JK-FF の一部抜粋

```
1  always @(posedge Clk)begin
2                  case({J,K})
3                          2'b00: Q <= Q;
4                          2'b01: Q <= 0;
5                          2'b10: Q <= 1;
6                          2'b11: Q<= ~Q;
7                          default Q <= 0;
8                  endcase
9          end
```

T-FF の機能シミュレータの結果は図 14 のようになった。

Clk の立ち上がりのタイミングは 6ps, 16ps, 26ps, 36ps, 46ps, 56ps, ... と 10ps 周期である。このとき、T の値は、1, 0, 0, 0, 0, 1, ... となっていることが分かる。そしてQの値は、1, 1, 1, 1, 1, 0, ... と T の値が０の時はQの値が保持されており、Tの値が１になったときQの値が反転していることが分かる。この法則は T-FF と一致している。ゆえに、この回路で T-FF を作成できたと言える。

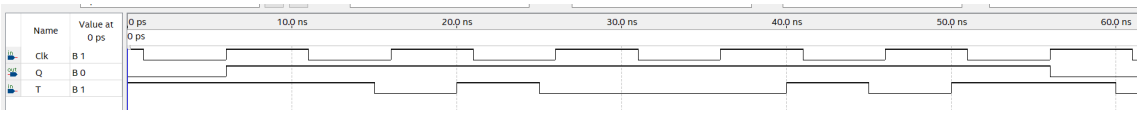

図 14　演習 2 における T-FF の結果

JK-FF の機能シミュレータの結果は図 15 のようになった。
Clk の立ち上がりのタイミングは 6ps, 16ps, 26ps, 36ps, 46ps, 56ps, ... と 10ps 周期である。このとき、(J,K) の値は (0,1), (1,0), (0,0), (1,1), (0,0), (1,1) ... となっていることが分かる。そしてQの値は、0, 1, 1, 0, 0, 1, ... となり,(J,K)=(0,1) のときはQの値は常に 0 で (J,K)=(1,0) のときはQの値は常に 1 となっていた。さらに (J,K)=(0,0) のときは Q の値は保持され、(J,K)=(1,1) のときは Q の値は反転されていた。この法則は JK-FF と一致しているゆえに、この回路で JK-FF を作成できたと言える。

図 15　演習 2 における JK-FF の結果

6.3　演習 3

EN,RST を備えた 4 ビットカウンターを Verilog HDL でコード 17 のように記述した。コード 10 はコード 17 から一部抜粋したものである. RST と EN で分岐文を作ることで、カウンターを実装した。まず、RST が 0 のときは Count を 0 にリセットする。そしてもし RST が 1 のとき EN の値をチェックする。EN が 1 のときは Count をインクリメントさせ、EN が 0 のときは Count をそのまま変更させｔないようにする。これらを Clk のポジティブエッジをトリガーに行うよう設定した。

ソースコード 10　演習 3 の一部抜粋

```
1  assign Q = Count;
2          always @(posedge Clk)
3          begin
4                  if(!RST)
5                          Count <= 4'b0000;
6                  else if(EN)
7                          Count <= Count + 1'b1;
8                  else
9                          Count <= Count;
10         end
```

まず、EN=1,RST=1 でテストを行った。結果は図 16 のようになった。Clk の立ち上がりのタイミングで outputQ はインクリメントされていることが分かる。RST=1 なのでリセットはされず、EN=1 だからインクリメントされており正しく動作したことが分かる。

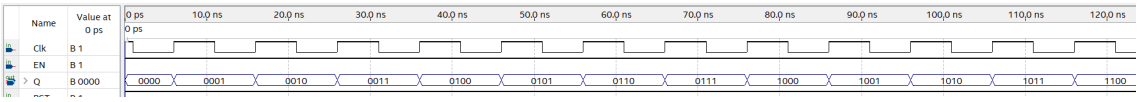

図 16　演習 3 における EN=1,RST=1 の結果

次に EN の値を変えてテストを行った。結果は図 17 のようになった。50ns のタイミングで EN=1 から EN=0 に変更した。すると、50ns までは Clk の立ち上がりのタイミングで outputQ はインクリメントされていることがわかるが、50ns 以降の Clk の立ち狩りでは outputQ はインクリメントされず、0101 のままであった。そして、100ns で EN=1 に変更すると、outputQ は再度 Clk の立ち上がりタイミングでインクリメントを始めた。この間 RST は 1 であり outputQ はリセットされることはなかった。つまり EN=1 のときは Clk の立ち上がりのタイミングでインクリメントされて、EN=0 の時にはインクリメントされないことが確かめられた。

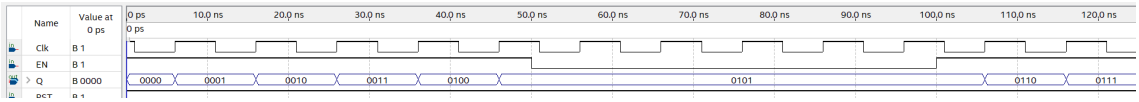

図 17　演習 3 における EN が変化したときの結果

最後に RST の値を変えテストを行った。結果は 18 のようになった。50ns のタイミングで RST=1 から RST=0 に変更した。すると、50ns までは Clk の立ち上がりタイミングで outputQ はインクリメントされている事が分かるが、50ns 以降の Clk の立ち上がりタイミングでは outputQ はインクリメントされることなく、0000 にリセットされていることが分かる。これは 106ns, すなわち RST が 1 になり Clk の立ち上がりタイミングまで続いている。このことから、RST=0 のときはリセットされることが確かめられた。

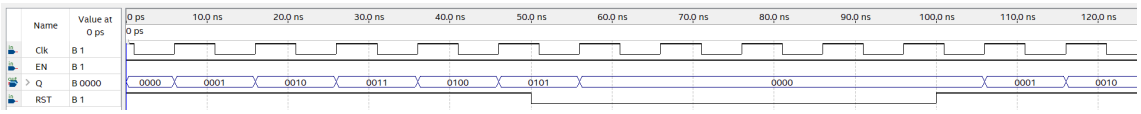

図 18　演習 3 における RST が変化したときの結果

以上のテストより EN=0 のときはインクリメントはされず、RST=0 のときは値がリセットされ、それ以外のときは Clk の立ち上がりのタイミングでインクリメントされていることが確かめられた。ゆえに、EN,RST を備えた 4 ビットカウンターが作成できたといえる。

7　ソースコード

ソースコード 11　zissyu1.v

```
1  module Zissyu1(I0,I1, I2, I3, I4, S, Q);
2      input [2:0] I0, I1, I2, I3, I4, S;
3      output [2:0] Q;
4
5      function [2:0] z1;
6          input [2:0] i_0, i_1, i_2, i_3, i_4, s1;
7                      begin
8                              case (s1)
9                                              3'b000: z1 = i_0;
10                                             3'b001: z1 = i_1;
11                                             3'b010: z1 = i_2;
12                                             3'b011: z1 = i_3;
13                                             3'b100: z1 = i_4;
14                                             default: z1 = 3'bxxx;
15                             endcase
16                     end
17     endfunction
18
19     assign Q = z1(I0, I1, I2, I3, I4, S);
20 endmodule
```

ソースコード 12　zissyu2.v

```
1  module d_latch (EN, D, Q);
2      input EN, D;
3      output Q;
4      wire R, S_g, R_g, Q1, Q2;
5
6      assign S_g = ~(D & EN);
7      assign R_g = ~(R & EN);
8      assign R = ~D;
9      assign Q1 = ~(S_g & Q2);
10     assign Q2 = ~(R_g & Q1);
11
12     assign Q = Q1;
13 endmodule
14
15 module Zissyu2(EN, D, Q);
16     input EN, D;
17     output Q;
18     wire W1, W2;
19
20     d_latch master_latch(~EN, D, W1);
21     d_latch slave_latch(EN, W1, W2);
22     assign Q = W2;
23 endmodule
```

ソースコード 13　zissyu3.v

```
1  module Zissyu3(Clk, I, Resetn, Q);
2      input Clk, I, Resetn;
3      output [6:0]Q;
4      parameter S0 = 4'b0000,
5                S1 = 4'b0001,
6                S2 = 4'b0010,
7                S3 = 4'b0011,
8                S4 = 4'b0100,
9                S5 = 4'b0101,
10               S6 = 4'b0110,
11               S7 = 4'b0111,
12               S8 = 4'b1000,
13               S9 = 4'b1001,
14               S10 = 4'b1010,
15               S11 = 4'b1011,
16               S12 = 4'b1100;
17     reg [3:0] cur_st, next_st;
18     assign Q =    (cur_st == S2) ? 7'b0001000 : (
19         (cur_st == S3) ? 7'b0000011 : (
20         (cur_st == S6) ? 7'b1000110 : (
21         (cur_st == S7) ? 7'b0100001 : (
22         (cur_st == S9) ? 7'b0000110 : (
23         (cur_st == S11) ? 7'b0001110: (
24         (cur_st == S12) ? 7'b1111110: 7'b1111111))))));
25     always @(posedge Clk)
26         if(!Resetn)
27             cur_st <= S0;
28         else
29             cur_st <= next_st;
30     always @(cur_st or I)
31         case(cur_st)
32             S0: next_st = (I) ? S4 : S1;
33             S1: next_st = (I) ? S3 : S2;
34             S2: next_st = (I) ? S4 : S1;
35             S3: next_st = (I) ? S4 : S1;
36             S4: next_st = (I) ? S8 : S5;
37             S5: next_st = (I) ? S7 : S6;
38             S6: next_st = (I) ? S4 : S1;
39             S7: next_st = (I) ? S4 : S1;
40             S8: next_st = (I) ? S10 : S9;
41             S9: next_st = (I) ? S4 : S1;
42             S10: next_st = (I) ? S12 : S11;
43             S11: next_st = (I) ? S4 : S1;
44             S12: next_st = (I) ? S4 : S1;
45             default: next_st = S0;
46         endcase
47 endmodule
```

ソースコード 14　ensyu1.v

```
module Ensyu1(D,Q);
        input[3:0] D;
        output[6:0] Q;

        function [6:0] func_decoder;
                input [3:0] in;
                begin
                        case (in)
                                4'b0000: func_decoder = 7'b0100000;//012345
                                4'b0001: func_decoder = 7'b1111001;//12
                                4'b0010: func_decoder = 7'b0100110;//01346
                                4'b0011: func_decoder = 7'b0111000;//01236
                                4'b0100: func_decoder = 7'b0011011;//256
                                4'b0101: func_decoder = 7'b0010010;//02356
                                4'b0110: func_decoder = 7'b0000011;//023456
                                4'b0111: func_decoder = 7'b1011000;//0125
                                4'b1000: func_decoder = 7'b0000000;//0123456
                                4'b1001: func_decoder = 7'b0011000;//01256
                                4'b1010: func_decoder = 7'b0001000;//012456 A
                                4'b1011: func_decoder = 7'b0000011;//23456 B
                                4'b1100: func_decoder = 7'b1000110;//0345 C
                                4'b1101: func_decoder = 7'b0100001;//12346 D
                                4'b1110: func_decoder = 7'b0000110;//03456 E
                                4'b1111: func_decoder = 7'b0001110;//0456 F
                                default: func_decoder = 7'b1111111;
                        endcase
                end
        endfunction

                assign Q = func_decoder(D);
endmodule
```

ソースコード 15　ensyu21.v

```
1  module t_ff(Clk, T, Q);
2          input Clk, T;
3          output Q;
4          reg Q;
5
6          always @(posedge Clk)
7                  if(T == 0)
8                          Q <= Q;
9                  else
10                         Q <= ~Q;
11 endmodule
```

ソースコード 16　ensyu22.v

```
1  module jk_ff(Clk, J, K, Q);
2          input Clk, J, K;
3          output Q;
4          reg Q;
5
6          always @(posedge Clk)begin
7                  case({J,K})
8                          2'b00: Q <= Q;
9                          2'b01: Q <= 0;
10                         2'b10: Q <= 1;
11                         2'b11: Q<= ~Q;
12                         default Q <= 0;
13                 endcase
14         end
15 endmodule
```

ソースコード 17　ensyu3.v

```
1  module Ensyu3 (Clk, RST, EN, Q);
2          input Clk,RST,EN;
3          output [3:0]Q;
4          wire [3:0]Q;
5          reg [3:0]Count;
6
7          assign Q = Count;
8          always @(posedge Clk)
9          begin
10                 if(!RST)
11                         Count <= 4'b0000;
12                 else if(EN)
13                         Count <= Count + 1'b1;
14                 else
15                         Count <= Count;
16         end
17 endmodule
```

8 参考文献

参考文献

[1] https://metoree.com/categories/multiplexer-ic/ metoree.com マルチプレクサ IC5 選　閲覧日 2022 年 10 月 3 日