目次

1 **実験の目的** 3

2 **実験器具及びツール** 3

3 **実験内容** 3
- 3.1 実習 1 . . . 3
- 3.2 実習 2 . . . 3
- 3.3 実習 3 . . . 4

4 **実験結果** 4
- 4.1 実習 1 . . . 4
- 4.2 実習 2 . . . 7
- 4.3 実習 3 . . . 8

5 **演習課題** 10
- 5.1 演習 1 . . . 10
- 5.2 演習 2[1] . . . 11
- 5.3 演習 3 . . . 11
- 5.4 演習 4 [2] . . . 13

6 **考察** 14
- 6.1 実習 1 . . . 14
- 6.2 実習 2 [3] . . . 14
- 6.3 実習 3 . . . 14
- 6.4 演習 1 . . . 15

7 **ソースコード** 16

8 **参考文献** 24

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

8 ビット符号付き加減算器をリプルキャリ方式により VerilogHDL で記述した。

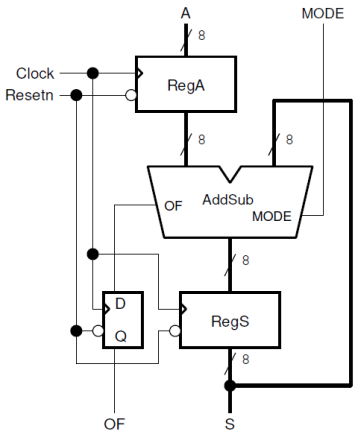

図 1　加減算器

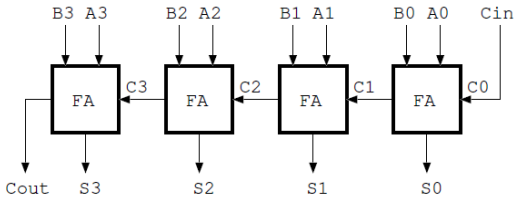

図 2　リプルキャリ加算器

3.2　実習 2

Timing Analyzer を用い、実習 1 で作成した回路が動作可能な最大周波数 fmax を調べた。また回路のどの部分がクリティカルパスになっているかを調べた

3.3　実習 3

乗算器を VerilogHDL で記述し、機能シミュレーションにより動作を確認した。また、最大動作周波数 fmax を調べた。

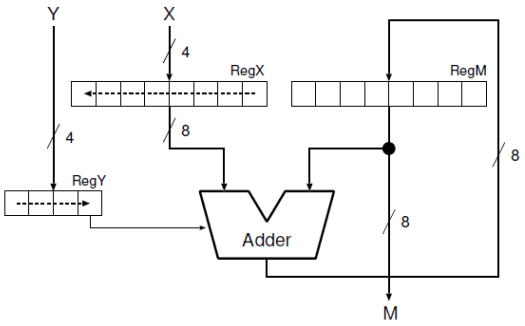

図 3　4 ビット乗算器のブロック図

4　実験結果

4.1　実習 1

加減算器の入力部を赤枠、出力部を青枠、そしてモジュールとモジュールの間のワイヤに新に名前づけをした。例えば、RegA と AddSub の間のワイヤを Aout,AddSub と RegOf の間のワイヤを Ofin,AddSub と RegS の間のワイヤを Sin とした。

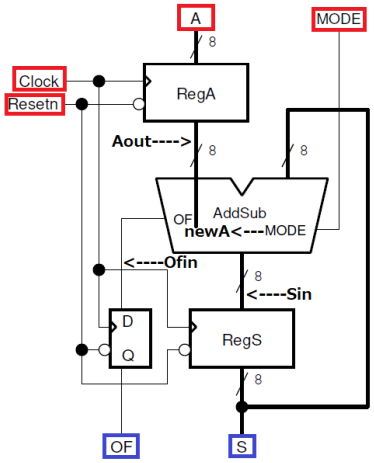

図 4　加減算器

VerilogHDL でコード 5 のように記述した。コード 1 はコード 5 から一部抜粋したものである。図 4 のようにまず、RegA と RegOf,RegS そして AddSub のモジュールのインスタンスをつくった。そして

RegA,RegS,RegOf は Clk の立ち上がり時に Rst が 1 ならば入力されたものに更新されるので D-FF を基に作製した。AddSub については、まず、mode によって加算か減算かを振り分けた。減算の場合、2 の補数をとり、Aout の入力を負の値として S と加算することで実装した。たとえば、3-2 は 3+(-2) としても変わらないように負の値を足し合わせることで機能を実現させた。そして、演習 1 で全加算器を作成したがそれらを流用して、リプルキャリ方式の機能を実現させた。最上位ビットで生じたキャリーフラグをオーバーフローとして assign で Of2 に出力させた。

ソースコード 1　実習 1 の一部抜粋

```
1  module zissyu1(A, Clk, Rst, Mode, Sin, Aout, S, Of,Ofin,newA,C);
2    Reg7 regA(A,Clk,Rst,Aout);
3    Reg7 regS(Sin,Clk,Rst,S);
4    RegOf regOf(Ofin,Clk,Rst,Of);
5    AddSub addsub(Mode,Aout,S,Sin,Ofin,newA,C);
6  ...
7  module Reg7(I,Clk,Rst,Q); //D-FF
8        always @(posedge Clk)begin
9              if(Rst)
10                   Q1 <= I;
11             else
12                   Q1 <= 0;
13       end
14       assign Q =Q1;
15 endmodule
16 ...
17 module Zenkasan(A,B,Cin,S,Cout); //from ensyu1
18       assign S = (A^B)^Cin;
19       assign Cout = (A&B)|(B&Cin)|(Cin&A);
20 endmodule
21 ...
22 module AddSub(Mode,A,B,Y,Of2,A2,C);
23       function [7:0]returnA;
24             if(mode == 1)
25                   returnA = (Ain^8'b11111111)+1'b1;
26             else
27                   returnA = Ain;
28       endfunction
29 ...
30       assign A2 = returnA(Mode,A);
31       Zenkasan zenkasan0(A2[0],B[0],0,Y[0],C[0]);
32       Zenkasan zenkasan1(A2[1],B[1],C[0],Y[1],C[1]);
33 ...
34       Zenkasan zenkasan7(A2[7],B[7],C[6],Y[7],C[7]);
35       assign Of2 = C[7];
36 endmodule
```

まず、Reset を 1,Mode を 0 にしたとき A と S の加算が Clk の立ち上がりタイミングで行われているかテストを行った。Reset1 なので、Reg に格納される値はリセットされることなく、また Mode も 0 なので、値

はAとSの加算になると予想がたてられる。実際、機能シミュレータの結果は図 5 のようになった。
実際、Clk の立ち上がりタイミング、6ns,16ns,26ns,36ns では A の値は 1,11,101,111 で、S は 0,1,100,1001 となっていた。16ns のSの値は 6ns 時点のAの値 1 と 6ns 時点の S の値 0 の和 1 のなっている。26ns のSの値は 16ns 時点のAの値 11 と 16ns 時点の S の値 1 の和 100 になっている、36ns のSの値は 26ns 時点のAの値 101 と 26ns 時点の S の値 100 の和 1001 となっていることがわかる。この和の法則は 36ns 以降も成り立っている。このことより、A と S の加算が行われていることが確かめられた。

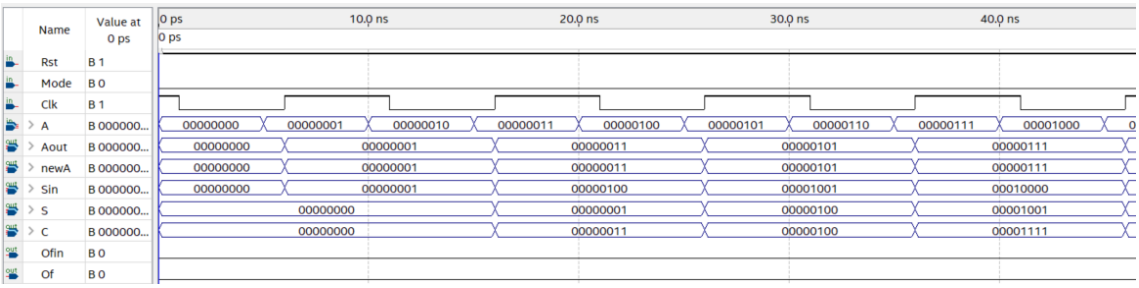

図 5　実習 1 における加算確認の結果

次に Reset が機能しているかテストを行った。Reset1 の時は A,S,Of と Clk の立ち上がりタイミングで更新されるが、Reset0 のときは inputA の値に関係なく、clk の立ち上がりタイミングで 0 にリセットされる事が予想される。実際、機能シミュレータの結果は図 6 のようになった。
Reset が 1 の時は図 5 でも確認したように加算が行われていることがわかる。一方で Reset が 0 に 25ns で変化したがそのあとの始めての clk の立ち上がりタイミング 26ns のとき、Aout の値が 0 になっていることが見てわかる。Aout とは図 4 のように RegA から出力される値である。つまり、Reset 0 のときは RegA では出力を 0 にする機能が働いていることが分かる。同様にSでも Reset1 から 0 に変わったあと、CLk の立ち上がりタイミング 26ns 時点で 0 にリセットされていることがわかる。よって Reset 機能は確かめられた。

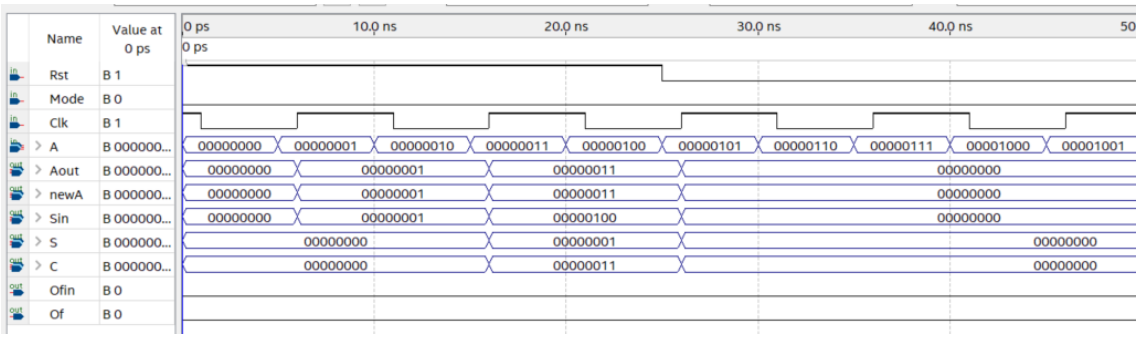

図 6　実習 1 における Reset 確認の結果

次に Mode を 1 にし減算ができるか確認をおこなった。Aが 10000101 のとき clk の立ち上がり 6ns において newA は 0 から 01111011 になっているこれはAの 2 の補数を表している。そしてこの値が Sin に入り、次の clk の立ち上がりタイミング 16ns においてSはAの二の補数が足されていることが分かる。そして次の clk の立ち上がりタイミングすなわち 26ns においてSはAの二の補数 011101011 とSの値が足しあわされて 11110110 となっている。このとき OF が 1 となっている。これはAからはいってきた値 01111011 とSの値

11110110 が足しあわされたとき最上位ビットの桁が繰り上がっているためだ。よって Mode1 の減算と OF は正しく機能していることがこの結果から分かる。

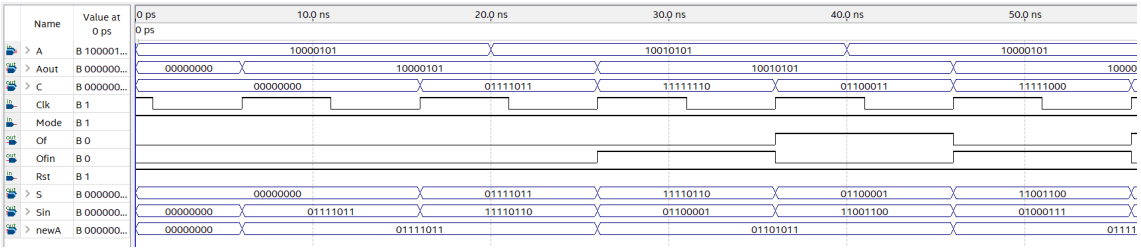

図 7　実習 1 における OF と Mode の確認の結果

4.2　実習 2

Timing Analyzer を用い、実習 1 で作成した回路が動作可能な最大周波数 fmax を調べた。また回路のどの部分がクリティカルパスになっているかを調べた
図 8 は実習 2 における最大周波数 fmax を示しているように最大周波数は 297.09MHz となった。

	Fmax	Restricted Fmax	Clock Name
1	297.09 MHz	297.09 MHz	Clk

図 8　実習 2 における最大周波数 fmax

クリティカルパスは図 9 のようになった。ここから RegA を通って、RegOf を通るまでが最も遅延時間が大きくなるクリティカルパスだと分かる。次に妥当性について考えてみる。遅延時間が最大となるときとはどいうときなのか。そもそも遅延時間は論理素子を通るときに発生する。すなわち論理素子が多いモジュールほど遅延時間が大きくなる傾向になる。今回作成したモジュールは RegA,RegOf,RegS,AddSub である。前者３つにおいては基本 D-ff を基にしたものであり出力に入力をつなげるか 0 にリセットするかの単純なものであった。一方 Addsub においては、モジュール内でさらにサブモジュールをインスタンス化をおこなっており、さらに８つもインスタンス化をおこなっている。そしてこのサブモジュールも入力を出力するまたは２の補数をとる機能をそなえている。そのため、RegA から RegOf の間、すなわち Addsub で遅延時間が最大になると考えられ、この結果は妥当だと言える。

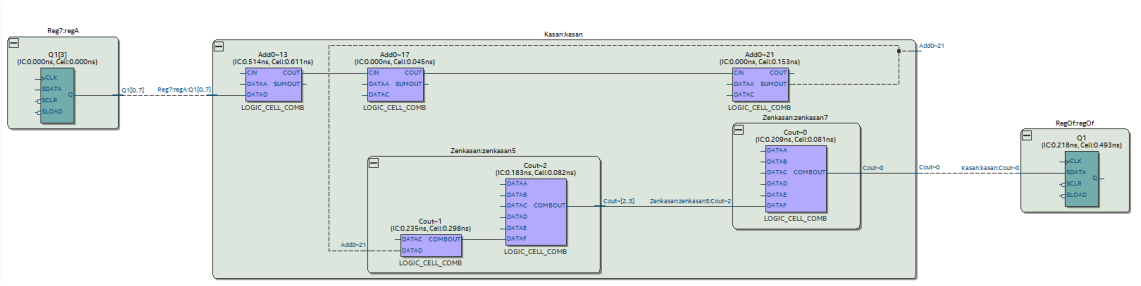

図 9　実習 2 におけるクリティカルパス

4.3　実習 3

VerilogHDL でコード 6 のように記述した。
コード 2 はコード 6 から一部抜粋したものである

ソースコード 2　実習 3 の一部抜粋

```
1  module Zissyu3(X,Y,Clk,Rst,M,Mout,Xout,Min,Yout,cur_st);
2          always @(posedge Clk) // set status by parameter
3                  else
4                          case(cur_st)
5                                  I: cur_st = A0;
6                                  A0: cur_st = A1; ...
7                                  F: cur_st = F;
8  ...
9  module RegX(I,cur_st,Clk,Rst,Q);
10         always @(posedge Clk)begin //shift x depends on status
11                 if(Rst)begin
12                         if(cur_st == 3'b000)
13                                 Q1 <= I;
14                         else if(cur_st == 3'b001)
15                                 Q1 <= (I << 1); ...
16                         else if(cur_st == 3'b100)
17                                 Q1 <= (I << 4);
18 ...
19 module RegY(I,cur_st,Clk,Rst,Q);
20         always @(posedge Clk)begin //shift y depends on status
21                         else if(cur_st == 3'b001)
22                                 Q1 <= (I >> 1); ...
23 ...
24 module Adder(x,m,y,cur_st,out);
25         function [7:0]returnOut;
26                 if(cur_st == 3'b101)
27                         returnOut = m;
28                 else
29                         returnOut = m + x*y[0];
30         endfunction
31         assign out = returnOut(cur_st,x,m,y);
32 endmodule
```

まず、状態遷移図にもとづいて、状態を I から F までパラメータを設置した。つづいて、clk の立ち上がりタイミングでそれぞれの状態パラメータを更新させた。F になったときはそれ以降も状態は F になるよう設定した。つづいて、モジュールをインスタンス化した。それぞれ、RegX, RegY, RegM, そして Adder である。それぞれ個別に見ていくと、まず、RegX は状態によってことなるものの、状態が一つ遷移するごとに x を左シフト移動させている。状態 F のときは何も変化させないようにする。同様に RegY は状態に応じて y を右シフトさせている。RegQ は D-ff と同じ働きでポジティブエッジのときにQを更新するようなモジュールになっている。そして最後に Adder は状態が F のときは入力をそのまま出力に返し、それ以外の場合は y

の 0 要素目と x の掛け合わせを足し合わせる処理を行うことでモジュールを作成させた。

機能シミュレータの結果は図 10 のようになった。まず、入力 x は 10 をいれ y を 1010 をいれた。このとき x*y = 2*10 = 20 = 10100(2) となることが予想される。実際結果 M は 10100 となっており正しい結果が出力されていることが分かる。

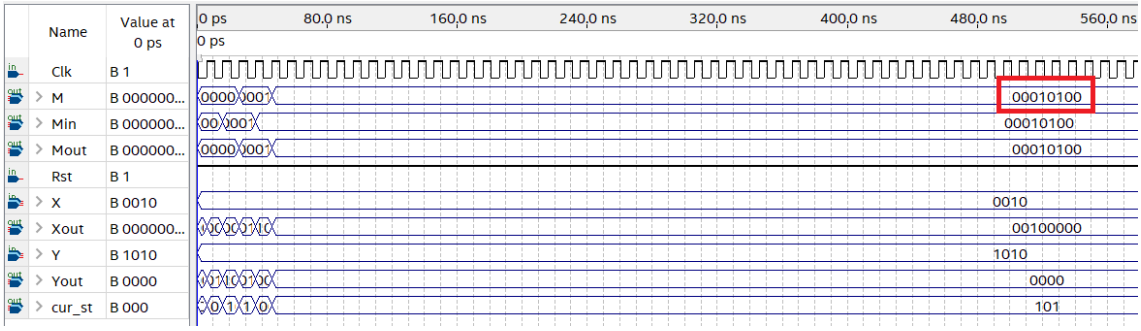

図 10　実習 3 における 4 ビット乗算の結果

実際計算課程を詳しく見てみる。(図 11) まず、始めのクロックの立ち上がりのとき状態は 000(I) であり、Xout は 10 が出力されている。次のクロックの立ち上がりでは Xout は 100、さらに次のクロックの立ち上がりでは 1000 となり右シフトがされていることがわかる。一方 Yout は最初のクロックの立ち上がりでは 1010 であったが次のクロックの立ち上がりでは 0101, そして次の立ち上がりでは 0010, 最後は 0001 となり左シフトされていることがわかる。最後に M についてみてみると、Yout が 1010,0010 のときは変化がなく、0101,0001 のときは xout の値が M に足しあわされて 100, 10100 となっていることがわかる。これらの結果から結果の値だけでなく、出力課程も予想通りの動きになっていることが分かりこの VerilogHDL で実装できたと言える。

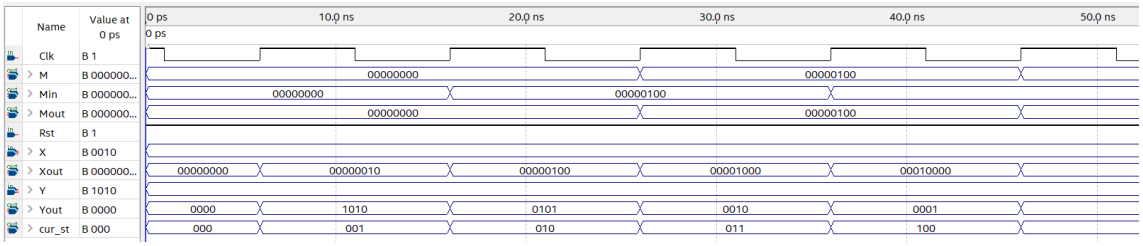

図 11　実習 3 における 4 ビット乗算の結果

最大周波数は 175.72MHz であった。(図 12)

	Fmax	Restricted Fmax	Clock Name
1	175.72 MHz	175.72 MHz	Clk

図 12　実習 3 における最大周波数の結果

RTL の結果は図 13 のようになった。

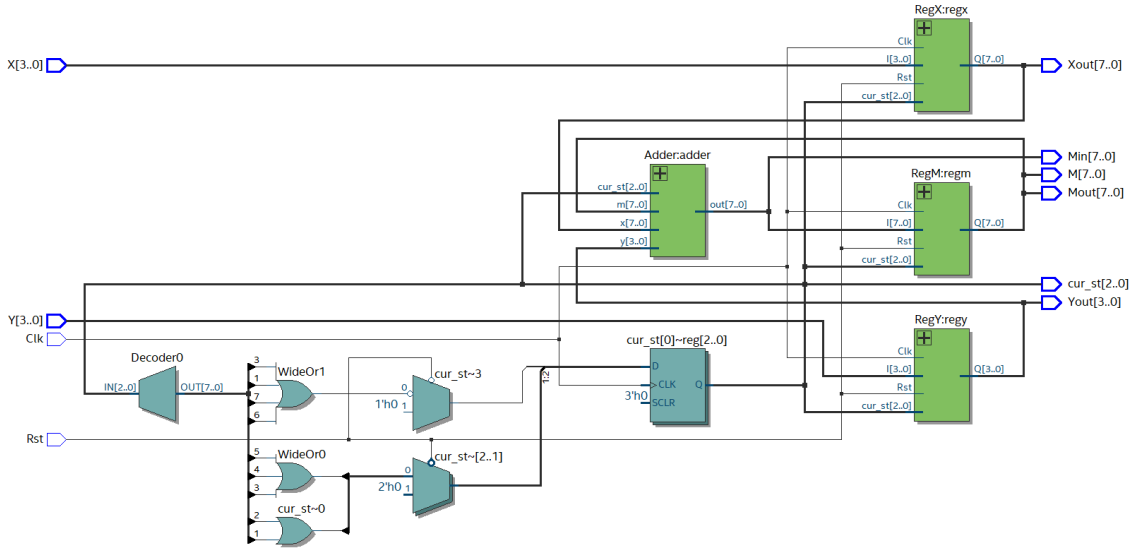

図 13　実習 3 における RLTViewer の結果

5　演習課題

5.1　演習 1

1bit の全加算器を論理ゲートを用いて構成した。表 1 に基づいて出力変数を入力変数の論理式として示し、回路図を作成した。

表 1　1 ビット全加算器の真理値表

入力			出力	
A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

まず、出力 S について考える。Cin が 0 のとき、A が 0 かつ B が 0 なら S は 0 で、A が 1 かつ B が 0 または A が 0 かつ B が 1 なら S は 1。そして、A かつ B が 1 なら S は 0 となる。すなわち、A と B が同じ値のときは 0 で異なるときは 1 となるから S は A と B の排他的論理和だと分かる。次に、Cin が 1 の時を考える。A が 0 かつ B が 0 なら S は 1 で、A が 1 かつ B が 0 または A が 0 かつ B が 1 なら S は 0。そして、A かつ B が 1 なら S は 1 となる。これは A と B の排他的論理和の結果を C と排他的論和をとった値が S となることになる。

これを数式にすると以下のようにかける。

$$S = A \oplus B \oplus C_{in} \tag{1}$$

次に Cout について考える。Cout はキャリーフラグを表す。A と B とCのうち 2 つ以上が 1 のとき Cout は 1 となる。つまり、AとB、BとC、CとAの組のうち一つ以上が 1、1 のペアがあると Cout は 1 となる。すなわち、AかつB、BかつC、CかつAの論理和でC out を判定することができる。これを数式に表すと以下のようになる

$$C_{out} = (A \wedge B) \vee (B \wedge C_{in}) \vee (C_{in} \wedge A) \tag{2}$$

これらの論理式を踏まえて、回路図を作成した。回路図は以下のようになった.(図 14)

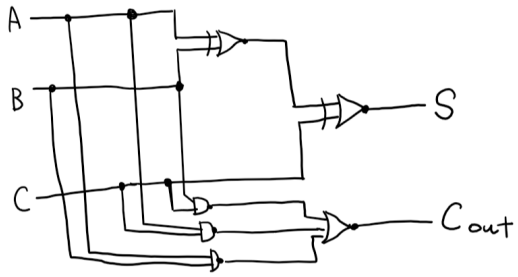

図 14　全加算器の回路図

発展として考察 1 に HDL で記述を行った。

5.2　演習 2[1]

最大周波数を高める工夫として考える手法を調べた。

- レジスタの層を増やす。(例：A*B + C*D を 1clk で処理しているところを A*B, C*D の処理を 1clk で処理し、2clk 目で二つを加算する)
- 並列に演算できる部分を切り出す (例：$8'B1010_1010 * 8'B0101_0101$ を $1010 * 0101, 1010 * 01010$ にわけ、次のクリックで足し合わせる)
- ロジックを平坦にする (例：不要な else if の削除)
- レジスタバランスの機能を使う (例：レジスタの前後にろぎっくを振り分けて動作周波数を上げる。論理合成ツールの最適化オプションで設定する)
- パスの再配置 (例：if 文の優先順位を入れ替えることで動作周波数をあげる。)

5.3　演習 3

まず、-2 の掛け算で考えてみる。-2 は 4 ビット符号付き整数のとき 1110 である。-4 は 1100 である。-6 は 1010 である。-2*3=-6 となるが、$-2*3=-2*(1+2)=-2 + -4 = 1110(2) + 1100(2) = 1010(2)$ となる。このと

き最上位ビットは符号ビットになっており、１で固定されている。つまり、実習３と同様にｘは左シフトを行うが、最上位ビットは固定されていることになる。

VerilogHDL でコード７のように記述した。
コード３はコード７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 RegX(I,cur_st,Clk,Rst,Q);
2          always @(posedge Clk)begin
3                  if(Rst)begin
4                          if(cur_st == 3'b000)
5                                  Q1 <= I;
6                          else if(cur_st == 3'b001)begin
7                                  Q1 <= (I << 1);
8                                  Q1[3] = I[3];
9                          end
10                         else if(cur_st == 3'b010)begin
11                                 Q1 <= (I << 2);
12                                 Q1[3] = I[3];
13                         end
14                         else if(cur_st == 3'b011)begin
15                                 Q1 <= (I << 3);
16                                 Q1[3] = I[3];
17                         end
18                         else if(cur_st == 3'b100)begin
19                                 Q1 <= (I << 4);
20                                 Q1[3] = I[3];
21                         end
22                 end
23                 else
24                         Q1 <= I;
25         end
26         assign Q =Q1;
27 endmodule
```

実習３からの変更点として、実習３では出力をビット幅８としたが今回はビット幅を今回は４ビットに変更した。また、RegX において、左シフトを実習３のように行ったあと、最上位ビットを入力 x の最上位ビットと合わせる処理を追記した。

機能シミュレータの結果は図 15 のようになった。入力ｘは 1100 と y は 0011 で出力 M は 1010 となった。これは x*y=1100*0011(2)=-2*3=-6=1010(2) となっており負の乗算結果が正しいことが分かる。

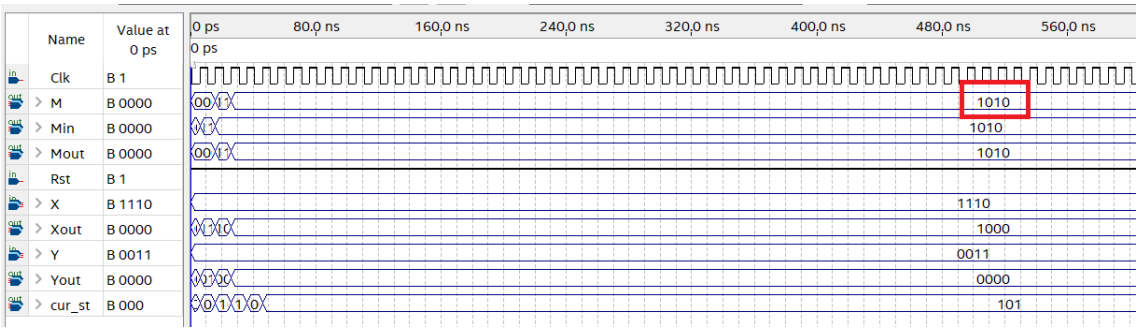

図 15　演習 3 の-2*3 の結果

次に計算処理の過程を見てみる。まず、cur_st において状態はｃｌｋの立ち上がりで遷移していることがわかる。次に、Xout は clk の立ち上がりで左シフトを行っている。そして、符号ビットである最上位ビットは常に 1 に保たれている。Yout は実習 3 同様状態遷移に合わせて右シフトが行われている。そして、Mout では clk の立ち上がりで 1110 が格納され、次の立ち上がりでは xout の 1100 が足しあわされ、1010 が格納されていることが分かる。これらの結果から結果の値だけでなく、出力課程も予想通りの動きになっていることが分かりこの VerilogHDL で実装できたと言える。

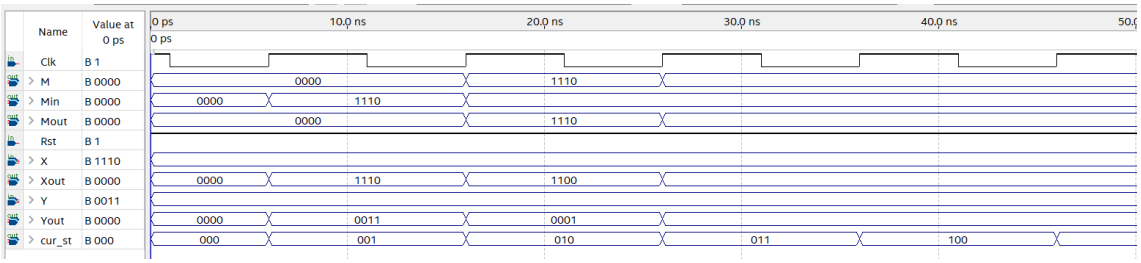

図 16　演習 3 の計算過程

5.4　演習 4 [2]

乗数Ｙが負のとき、一つの方法として、Ｙを２の補数をとって正の整数にする。そして、演算結果を２の補数をとる。例えば、-2*-3 のとき、-3=1101(2) を 0011(2) に変換して、演習 3 の回路を使う。結果は-6(1010) となる。ここに２の補数をとると 6(0110) となる。
上記の方法と異なる方法としてブースの方法がある。負の乗数Ｙが

$$Y = (-2^{n-1}) \times Y_{n-1} + 2^{n-2} \times Y_{n-2} + ... + 2^i \times Y_i + ... + 2^1 \times Y_1 + 2^0 \times Y_0$$
$$= -s^{n-1} \times (Y_{n-i} - Y_{n-2}) - 2^{n-2} \times (Y_{n-2} - Yn - 3) - ... - 2^i \times (Y_i - Y_{i-1}) - ... - 2^1 \times (Y_1 - Y_0) - Y_0$$

と変形できることを利用する。たとえば、$Y_i - Y_{i-1}$ が 0 のとき、すなわち $Y_i = 0 \&\& Y_{i-1} = 0$ または $Y_i = 1 \&\& Y_{i-1} = 1$ のとき Y には -2^i を加えない。$Y_i - Y_{i-1}$ が 1 のときすなわち $Y_i = 1 \&\& Y_{i-1} = 0$ のとき Y に -2^i を加える。$Y_i - Y_{i-1}$ が-1 のとき $Y_i = 0 \&\& Y_{i-1} = 1$ のとき Y に 2^i を加える。
m を被乗数、r を乗数とし、m のビット数を x、r のビット数を y とする。またＡの左端に m, Ｓの左端に-m を格納する。Ｐの右端には r を格納する。

P の右端 2 ビットを調べ、もし 01 のときＰ＋Ａを計算する。10 のとき P+S を計算する。00 or 11 のときは何もしない。そのあと得られた結果を右に 1 ビットシフトさせる。
たとえば、3*-4 のとき、

- m = 0011, -m = 1101, r = 1100
- A = 0011 0000 0
- S = 1101 0000 0
- P = 0000 1100 0

となり、

1. P = 0000 1100 0　となり最後の　2 ビットは 00
2. P = 0000 0110 0 となり最後の 2 ビットは 00
3. P = 0000 0011 0 となり最後の 2 ビットは 10
   P = P+S =　1101 0011 0
4. P = 1110 1001 1 となり最後の 2 ビットは 11

よって積は 1111 0100 となり-12 となる。このようにすると乗算を負の数で計算することができる。

6 考察

6.1 実習 1

実習 1 では 2 進数の和を計算する半加算とさらにキャリを考慮する全加算の違いを学んだ。全加算は 4bit リプルキャリ加算を利用することで実装できることを学習した。減算の処理も新たに減算のモジュールを作成するのではなく 2 の補数をとることで加算と同じ処理で計算できることを学んだ。

6.2 実習 2 [3]

実習 2 では not, or, and の回路素子を通過するごとに遅延が生じることを学んだ。もう少しなぜ遅延するのかについて考えてみた。論理回路の場合入力と出力は同じタイミングである。一方現実の回路は電気信号の伝搬であるため出力は必ず入力よりも遅れて現れる。この送れる時間が遅延時間になると考えられている。

6.3 実習 3

実習 3 では RTL を生成させた。実際に RegX,RegY,RegM,Adder のモジュールのインスタンスが生成されていることが分かる。そこでこの RTL についてさらに調べてみた。
RTL とは RegisterTransferLevel の略で、HDL で記述した回路を視覚的に表示することができる。そのメリットとして、HDL で記述したコードが期待通りの回路が得られるか、接続に誤りがないかを確認することができる。早い段階で特にシミュレーションを行うことで早期に設計ミスを発見できる利点がある。

6.4　演習１

演習１では論理回路を作成した。せっかくなので HDL で記述することにした。これを VerilogHDL で記述すると以下のようにできると考えた。これを基に実習 1 の加算器を作成した。

ソースコード 4　ensyu1.v

```
1 module Zissyu1v2(A,B,Cin,S,Cout);
2         input A, B, Cin;
3         output S,Cout;
4
5         assign S = (A^B)^Cin;
6
7         assign Cout = (A&B)|(B&Cin)|(Cin&A);
8 endmodule
```

7　ソースコード

ソースコード 5　zissyu1.v

```
1  module zissyu1(A, Clk, Rst, Mode, Sin, Aout, S, Of,Ofin,newA,C);
2      input [7:0]A;
3          input Clk,Rst,Mode;
4      output [7:0]Sin,Aout,S,newA,C;
5          output Of,Ofin;
6
7          Reg7 regA(A,Clk,Rst,Aout);
8          Reg7 regS(Sin,Clk,Rst,S);
9          RegOf regOf(Ofin,Clk,Rst,Of);
10         AddSub addsub(Mode,Aout,S,Sin,Ofin,newA,C);
11 endmodule
12
13 module Reg7(I,Clk,Rst,Q);
14         input [7:0]I;
15         input Clk,Rst;
16         output [7:0]Q;
17         reg [7:0]Q1;
18
19         always @(posedge Clk)begin
20                 if(Rst)
21                         Q1 <= I;
22                 else
23                         Q1 <= 0;
24         end
25         assign Q =Q1;
26 endmodule
27
28
29 module RegOf(I,Clk,Rst,Q);
30         input I;
31         input Clk,Rst;
32         output Q;
33         reg Q1;
34
35         always @(posedge Clk)begin
36                 if(Rst)
37                         Q1 <= I;
38                 else
39                         Q1 <= 0;
40         end
41         assign Q =Q1;
42 endmodule
43
44 //from ensyu1
```

```
45 module Zenkasan(A,B,Cin,S,Cout);
46         input A, B, Cin;
47         output S,Cout;
48
49         assign S = (A^B)^Cin;
50         assign Cout = (A&B)|(B&Cin)|(Cin&A);
51 endmodule
52
53
54 module AddSub(Mode,A,B,Y,Of2,A2,C);
55         input Mode;
56         input [7:0]A,B;
57         output [7:0]Y,Of2,A2,C;
58
59         function [7:0]returnA;
60                 input mode;
61                 input [7:0]Ain;
62
63                 if(mode == 1)
64                         returnA = (Ain^8'b11111111)+1'b1;
65                 else
66                         returnA = Ain;
67         endfunction
68
69         // if mode = 1 (A change negative)
70         assign A2 = returnA(Mode,A);
71
72         Zenkasan zenkasan0(A2[0],B[0],0,Y[0],C[0]);
73         Zenkasan zenkasan1(A2[1],B[1],C[0],Y[1],C[1]);
74         Zenkasan zenkasan2(A2[2],B[2],C[1],Y[2],C[2]);
75         Zenkasan zenkasan3(A2[3],B[3],C[2],Y[3],C[3]);
76         Zenkasan zenkasan4(A2[4],B[4],C[3],Y[4],C[4]);
77         Zenkasan zenkasan5(A2[5],B[5],C[4],Y[5],C[5]);
78         Zenkasan zenkasan6(A2[6],B[6],C[5],Y[6],C[6]);
79         Zenkasan zenkasan7(A2[7],B[7],C[6],Y[7],C[7]);
80         assign Of2 = C[7];
81 endmodule
```

ソースコード 6　zissyu3.v

```
1  module Zissyu3(X,Y,Clk,Rst,M,Mout,Xout,Min,Yout,cur_st);
2          input [3:0]X,Y;
3          input Clk,Rst;
4          output [7:0]M,Mout,Xout,Min;
5          output [3:0]Yout;
6          output reg [2:0] cur_st;
7          parameter I = 3'b000,
8              A0 = 3'b001,
9              A1 = 3'b010,
10             A2 = 3'b011,
11             A3 = 3'b100,
12             F = 3'b101;
13
14         always @(posedge Clk)
15                 if(!Rst)
16                         cur_st <= I;
17                 else
18                         case(cur_st)
19                                 I: cur_st = A0;
20                                 A0: cur_st = A1;
21                                 A1: cur_st = A2;
22                                 A2: cur_st = A3;
23                                 A3: cur_st = F;
24                                 F: cur_st = F;
25                                 default: cur_st = I;
26                 endcase
27
28         RegX regx(X,cur_st,Clk,Rst,Xout);
29         RegY regy(Y,cur_st,Clk,Rst,Yout);
30         Adder adder(Xout,Mout,Yout,cur_st,Min);
31         RegM regm(Min,cur_st,Clk,Rst,Mout);
32
33         assign M = Mout;
34 endmodule
35
36 module RegX(I,cur_st,Clk,Rst,Q);
37         input [3:0]I;
38         input [2:0]cur_st;
39         input Clk,Rst;
40         output [7:0]Q;
41         reg [7:0]Q1;
42
43         always @(posedge Clk)begin
44                 if(Rst)begin
45                         if(cur_st == 3'b000)
46                                 Q1 <= I;
47                         else if(cur_st == 3'b001)
```

```
48                              Q1 <= (I << 1);
49                      else if(cur_st == 3'b010)
50                              Q1 <= (I << 2);
51                      else if(cur_st == 3'b011)
52                              Q1 <= (I << 3);
53                      else if(cur_st == 3'b100)
54                              Q1 <= (I << 4);
55              end
56              else
57                      Q1 <= I;
58      end
59      assign Q =Q1;
60 endmodule
61
62 module RegY(I,cur_st,Clk,Rst,Q);
63      input [3:0]I;
64      input [2:0]cur_st;
65      input Clk,Rst;
66      output [3:0]Q;
67      reg [3:0]Q1;
68
69      always @(posedge Clk)begin
70              if(Rst) begin
71                      if(cur_st == 3'b000)
72                              Q1 <= I;
73                      else if(cur_st == 3'b001)
74                              Q1 <= (I >> 1);
75                      else if(cur_st == 3'b010)
76                              Q1 <= (I >> 2);
77                      else if(cur_st == 3'b011)
78                              Q1 <= (I >> 3);
79                      else if(cur_st == 3'b100)
80                              Q1 <= (I >> 4);
81              end
82              else
83                      Q1 <= I;
84      end
85      assign Q =Q1;
86 endmodule
87
88 module RegM(I,cur_st,Clk,Rst,Q);
89      input [7:0]I;
90      input [2:0]cur_st;
91      input Clk,Rst;
92      output [7:0]Q;
93      reg [7:0]Q1;
94
95      always @(posedge Clk)begin
96              if(Rst)
```

```
97                              Q1 <= I;
98                      else
99                              Q1 <= 0;
100             end
101             assign Q = Q1;
102     endmodule
103
104     module Adder(x,m,y,cur_st,out);
105             input [7:0]x,m;
106             input [3:0]y;
107             input [2:0]cur_st;
108             output [7:0]out;
109
110             function [7:0]returnOut;
111                     input cur_st;
112                     input [7:0]x,m;
113                     input [3:0]y;
114                     if(cur_st == 3'b101)
115                             returnOut = m;
116                     else
117                             returnOut = m + x*y[0];
118             endfunction
119
120
121             assign out = returnOut(cur_st,x,m,y);
122     endmodule
```

ソースコード 7　ensyu3.v

```
1  module Ensyu3(X,Y,Clk,Rst,M,Mout,Xout,Min,Yout,cur_st);
2          input [3:0]X,Y;
3          input Clk,Rst;
4          output [3:0]M,Mout,Xout,Min;
5          output [3:0]Yout;
6          output reg [2:0] cur_st;
7          parameter I = 3'b000,
8              A0 = 3'b001,
9              A1 = 3'b010,
10             A2 = 3'b011,
11             A3 = 3'b100,
12             F = 3'b101;
13
14         always @(posedge Clk)
15                 if(!Rst)
16                         cur_st <= I;
17                 else
18                         case(cur_st)
19                                 I: cur_st = A0;
20                                 A0: cur_st = A1;
21                                 A1: cur_st = A2;
22                                 A2: cur_st = A3;
23                                 A3: cur_st = F;
24                                 F: cur_st = F;
25                                 default: cur_st = I;
26                 endcase
27
28         RegX regx(X,cur_st,Clk,Rst,Xout);
29         RegY regy(Y,cur_st,Clk,Rst,Yout);
30         Adder adder(Xout,Mout,Yout,cur_st,Min);
31         RegM regm(Min,cur_st,Clk,Rst,Mout);
32
33         assign M = Mout;
34 endmodule
35
36 module RegX(I,cur_st,Clk,Rst,Q);
37         input [3:0]I;
38         input [2:0]cur_st;
39         input Clk,Rst;
40         output [3:0]Q;
41         reg [3:0]Q1;
42
43         always @(posedge Clk)begin
44                 if(Rst)begin
45                         if(cur_st == 3'b000)
46                                 Q1 <= I;
47                         else if(cur_st == 3'b001)begin
```

```
48                              Q1 <= (I << 1);
49                              Q1[3] = I[3];
50                      end
51                      else if(cur_st == 3'b010)begin
52                              Q1 <= (I << 2);
53                              Q1[3] = I[3];
54                      end
55                      else if(cur_st == 3'b011)begin
56                              Q1 <= (I << 3);
57                              Q1[3] = I[3];
58                      end
59                      else if(cur_st == 3'b100)begin
60                              Q1 <= (I << 4);
61                              Q1[3] = I[3];
62                      end
63              end
64              else
65                      Q1 <= I;
66      end
67      assign Q =Q1;
68  endmodule
69
70  module RegY(I,cur_st,Clk,Rst,Q);
71      input [3:0]I;
72      input [2:0]cur_st;
73      input Clk,Rst;
74      output [3:0]Q;
75      reg [3:0]Q1;
76
77      always @(posedge Clk)begin
78              if(Rst) begin
79                      if(cur_st == 3'b000)
80                              Q1 <= I;
81                      else if(cur_st == 3'b001)
82                              Q1 <= (I >> 1);
83                      else if(cur_st == 3'b010)
84                              Q1 <= (I >> 2);
85                      else if(cur_st == 3'b011)
86                              Q1 <= (I >> 3);
87                      else if(cur_st == 3'b100)
88                              Q1 <= (I >> 4);
89              end
90              else
91                      Q1 <= I;
92      end
93      assign Q =Q1;
94  endmodule
95
96  module RegM(I,cur_st,Clk,Rst,Q);
```

```
97          input [7:0]I;
98          input [2:0]cur_st;
99          input Clk,Rst;
100         output [3:0]Q;
101         reg [3:0]Q1;
102
103         always @(posedge Clk)begin
104                 if(Rst)
105                         Q1 <= I;
106                 else
107                         Q1 <= 0;
108         end
109         assign Q = Q1;
110 endmodule
111
112 module Adder(x,m,y,cur_st,out);
113         input [3:0]x,m;
114         input [3:0]y;
115         input [2:0]cur_st;
116         output [3:0]out;
117
118         function [3:0]returnOut;
119                 input cur_st;
120                 input [3:0]x,m;
121                 input [3:0]y;
122                 if(cur_st == 3'b101)
123                         returnOut = m;
124                 else
125                         returnOut = m + x*y[0];
126         endfunction
127
128
129         assign out = returnOut(cur_st,x,m,y);
130 endmodule
```

8 参考文献

参考文献

[1] Advanced FPGA Design, Steve Kilts, FPGA の動作周波数を上げる 5 つの方法

[2] コンピュータアーキテクチャの基礎, 柴山潔 著, p184 (c) ブースの方法

[3] https://edn.itmedia.co.jp/edn/articles/1203/01/news156.html 論理回路と実際のデジタル回路の違い, 閲覧日：2022 年 10 月 12 日