2022 年度 3 回生前期学生実験 HW

team02 機能設計仕様書

機能設計仕様書作成者: 伊藤舜一郎

グループメンバー：

伊藤舜一郎 (学籍番号:1029-32-7548)

植田健斗　(学籍番号:1029-32-6498)

提出期限：5 月 12 日 18 時　提出日: 2022 年 5 月 12 日

1　全体をどのようにコンポーネントに分割したか

1.1　SIMPLE/B の設計

SIMPLE/B の設計については以下の図 1 のように分割をして設計を進めた。図 2 は SIMPLE/B の全体の回路図である。

モジュール名	対応ファイル名	担当
プログラムカウンタ	PC.v	植田
インストラクションレジスタ	IR.v	
レジスタファイル	resisterFile.v	
フェーズカウンタ	phaseCounter.v	
制御部	control.v	
PCをインクリメントする組み合わせ回路	incrementer.v	
主記憶	mainMemory.v	
SZCV選択回路	SZCVJudge16.v,SZCVControl.v	
符号拡張	signExtend.v	
各マルチプレクサー	multiplexer16.v,multiplexer4.v,multiplexer3.v	
各レジスタ	register16.v,register4.v,register3.v	
全体	processser.v	
ALU	alu.v	伊藤
シフタ	shifter.v	
外部出力	externalOutput.v	

図 1: SIMPLE/B の設計の分割

1.2　機能拡張したプロセッサの設計の分割

機能拡張したプロセッサの設計については、上記の SIMPLE/B に即値命令の強化とパイプライン化を行うため、制御部を改善し、フォワーディング・パイプラインレジスタ・ハザード検知を新たに設計しなければならない。制御部、パイプラインレジスタ、ハザード検知に関しては植田が担当し、フォワーディングに関しては伊藤が担当する予定となっている（変更する可能性あり）。

2　各部品の仕様

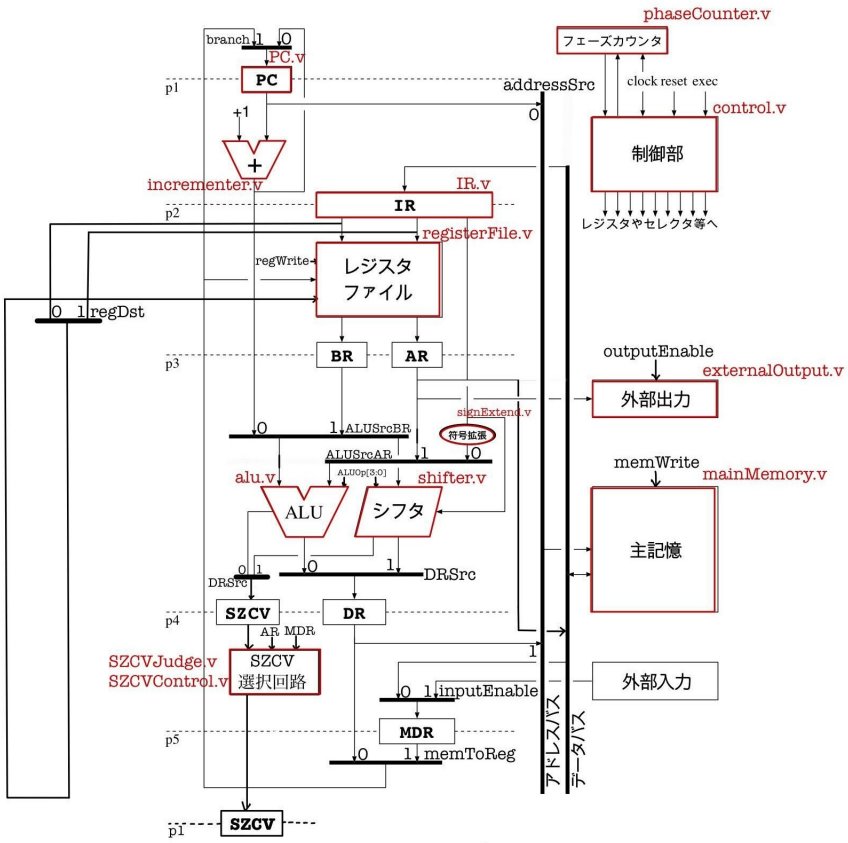
2.1　ALU

2.1.1　回路の仕様

入力・出力

- input [15:0] inA,inB
- input [3:0] op
- output [15:0] out
- output [3:0] SZCV

4 ビットの操作コード op を受け取り、それに応じて入力された 16 ビットの数値 inA,inB の値を用いて計算し、16 ビットの計算結果と対応した SZCV を出力する。



各レジスタ:register16.v,register4.v,register3.v
各マルチプレクサ:multiplexer16.v,multiplexer4.v,multiplexer3.v
全体:processor.bdf

図 2: SIMPLE/B 全体の回路図

2.1.2 回路の設計

inA,inB について、受け取った値を符号拡張して 17 ビットの数値とする。それらの数値を signExtendedInA,signExtendedInB とする。そして、signExtendedInB を符号反転し 1 を足したものを negativeInB とする。これは SUB 命令と CMP 命令に用いる。上記の計算は assign 部で行う。また assign 部にて signExtendedInA,signExtendedInB,op,negativeInB を受け取って SZCV,out を返す関数 OUT の計算を行う。

ソースコード 1: alu の assign 文

```
1 assign signExtendedInA = {inA [15], inA};
2 assign signExtendedInB = {inB [15], inB};
3 assign negativeInB = ((\~signExtendedInB) + 17'b0\_0000\_0000\_0000\_0001);
4 assign␣{SZCV,out}␣=␣OUT(signExtendedInA,signExtendedInB,op,negativeInB);
```

OUT 関数は signExtendedInA,signExtendedInB,op,negativeInB を受け取って SZCV,out を返す関数である。OUT の出力については、19 ビット目が SZCV の S,18 ビット目が SZCV の Z,17 ビット目が SZCV の C,16 ビット目が SZCV の V, 下位 16 ビットが演算結果となっている。関数内にお

いて signExtendedInA,signExtendedInB,op,negativeInB は IN_A,IN_B,OP,NEGATIVE_IN_B としている。OUT 関数では case 文を用いることで、OP によって出力を変えている。

op が AND 命令のときについて説明する。OUT[17],OUT[15:0] = IN_A + IN_B; で演算結果と C を計算している。ここで、演算結果が符号付き 16 ビットで表せる範囲を超えているか否かは、(IN_A[15] & IN_B[15] & $\tilde{\text{O}}$UT[15]) | ($\tilde{\text{I}}$N_A[15] & $\tilde{\text{I}}$N_B[15] & OUT[15]) で判定することができる。なぜなら、inA と i n B の最上位ビットが同じでかつ演算結果の最上位ビットがその値と異なるとき演算結果が 16 ビットで表せる範囲を超えているからである。よって、(IN_A[15] & IN_B[15] & $\tilde{\text{O}}$UT[15]) | ($\tilde{\text{I}}$N_A[15] & $\tilde{\text{I}}$N_B[15] & OUT[15]) は V となる。また、Z は演算結果が 0 と等しいかの真理値と同じになるので、OUT[18] = (OUT[15:0] == 0); で計算できる。また、S は演算結果の最上位ビットの値と同じになるので、OUT[19] = OUT[15]; で計算できる。

op が SUB 命令のときは AND において IN_B を NEGATIVE_IN_B に置き換えたときの演算結果と同じになる。また、op が CMP 命令のときも同じ出力結果となる。

op が AND、OR、XOR、MOV 命令の場合、演算結果はそれぞれ IN_A [15:0] & IN_B[15:0]、IN_A [15:0] | IN_B[15:0]、IN_A [15:0] $\hat{\text{I}}$N_B[15:0]、IN_B で計算できる。また、C,V は常に 0、S,Z は ADD と同じ実装で計算できる。

その他の命令を受け取った場合、演算結果と SZCV は 0 を出力するとする。

以上の仕様を実装した OUT 関数は以下のようになる。

ソースコード 2: alu の OUT 関数

```
1  function [19:0] OUT;//OUT={S,Z,C,V,out}
2          input [16:0] IN_A;
3          input [16:0] IN_B;
4          input [3:0] OP;
5          input [16:0] NEGATIVE_IN_B;
6
7          begin
8          case (OP)
9                  4'b0000:begin//ADD
10 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣{OUT[17],OUT[15:0]}␣=␣IN_A␣+␣IN_B;
11 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣OUT[16]␣=␣(IN_A[15]&IN_B[15]&~OUT[15])|(~IN_A
       [15]&~IN_B[15]&OUT[15]);//V
12 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣OUT[18]␣=␣(OUT[15:0]␣==␣0);//Z
13 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣OUT[19]␣=␣OUT[15];//S
14 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣end
15 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣4'b0001:begin//SUB
16                                 {OUT[17],OUT[15:0]} = IN_A + NEGATIVE_IN_B;
17                                 OUT[16] = (IN_A[15]&NEGATIVE_IN_B[15]&~OUT[15])|(~
                                       IN_A[15]&~NEGATIVE_IN_B[15]&OUT[15]);//V
18                                 OUT[18] = (OUT[15:0] == 0);//Z
19                                 OUT[19] = OUT[15];//S
20                                 end
21                 4'b0010:begin//AND
22 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣OUT[15:0]␣=␣IN_A␣[15:0]␣&␣IN_B[15:0];
23 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣OUT[16]␣=␣1'b0;
24                                 OUT[17] = 1'b0;
25 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣OUT[18]␣=␣(OUT[15:0]␣==␣0);//Z
26 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣OUT[19]␣=␣OUT[15];//S
27 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣end
28 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣4'b0011:begin//OR
29                                 OUT[15:0] = IN_A [15:0] | IN_B[15:0];
30                                 OUT[16] = 1'b0;
31 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣OUT[17]␣=␣1'b0;
32                                 OUT[18] = (OUT[15:0] == 0);//Z
33                                 OUT[19] = OUT[15];//S
34                                 end
35                 4'b0100:begin//XOR
36 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣OUT[15:0]␣=␣IN_A␣[15:0]␣^␣IN_B[15:0];
37 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣OUT[16]␣=␣1'b0;
```

```
                                    OUT[17] = 1'b0;
                                    OUT[18] = (OUT[15:0] == 0);//Z
                                    OUT[19] = OUT[15];//S
                                    end
                4'b0100:begin//CMP
                                    {OUT[17],OUT[15:0]} = IN_A + NEGATIVE_IN_B;
                                    OUT[16] = (IN_A[15]&NEGATIVE_IN_B[15]&~OUT[15])|(~
                                        IN_A[15]&~NEGATIVE_IN_B[15]&OUT[15]);
                                    OUT[18] = (OUT[15:0] == 0);//Z
                                    OUT[19] = OUT[15];//S
                                    end
                4'b0101:begin//CMP
                                    {OUT[17],OUT[15:0]} = IN_A + NEGATIVE_IN_B;
                                    OUT[16] = (IN_A[15]&NEGATIVE_IN_B[15]&~OUT[15])|(~
                                        IN_A[15]&~NEGATIVE_IN_B[15]&OUT[15]);
                                    OUT[18] = (OUT[15:0] == 0);//Z
                                    OUT[19] = OUT[15];//S
                                    end
                4'b0110:begin//MOV
                                    OUT[15:0] = IN_B[15:0];
                                    OUT[16] = 1'b0;
                                    OUT[17] = 1'b0;
                                    OUT[18] = (OUT[15:0] == 0);//Z
                                    OUT[19] = OUT[15];//S
                                    end
                4'b0111:begin//(reserved)
                                    OUT[15:0] = 16'b0000_0000_0000_0000;
                                    OUT[16] = 1'b0;
                                    OUT[17] = 1'b0;
                                    OUT[18] = 1'b0;
                                    OUT[19] = 1'b0;
                                    end
                4'b1000:begin//SLL
                                    OUT[15:0] = 16'b0000_0000_0000_0000;
                                    OUT[16] = 1'b0;
                                    OUT[17] = 1'b0;
                                    OUT[18] = 1'b0;
                                    OUT[19] = 1'b0;
                                    end
                4'b1001:begin//SLR
                                    OUT[15:0] = 16'b0000_0000_0000_0000;
                                    OUT[16] = 1'b0;
                                    OUT[17] = 1'b0;
                                    OUT[18] = 1'b0;
                                    OUT[19] = 1'b0;
                                    end
                4'b1010:begin//SRL
                                    OUT[15:0] = 16'b0000_0000_0000_0000;
                                    OUT[16] = 1'b0;
                                    OUT[17] = 1'b0;
                                    OUT[18] = 1'b0;
                                    OUT[19] = 1'b0;
                                    end
                4'b1011:begin//SRA
                                    OUT[15:0] = 16'b0000_0000_0000_0000;
                                    OUT[16] = 1'b0;
                                    OUT[17] = 1'b0;
                                    OUT[18] = 1'b0;
                                    OUT[19] = 1'b0;
                                    end
                4'b1100:begin//IN
                                    OUT[15:0] = 16'b0000_0000_0000_0000;
                                    OUT[16] = 1'b0;
                                    OUT[17] = 1'b0;
                                    OUT[18] = 1'b0;
                                    OUT[19] = 1'b0;
                                    end
                4'b1101:begin//OUT
                                    OUT[15:0] = 16'b0000_0000_0000_0000;
                                    OUT[16] = 1'b0;
                                    OUT[17] = 1'b0;
                                    OUT[18] = 1'b0;
                                    OUT[19] = 1'b0;
                                    end
                4'b1110:begin//(reserved)
                                    OUT[15:0] = 16'b0000_0000_0000_0000;
```

```
106 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣OUT[16]␣=␣1'b0;
107                                         OUT[17] = 1'b0;
108 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣OUT[18]␣=␣1'b0;
109                                         OUT[19] = 1'b0;
110 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣end
111 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣4'b1111:begin//halt()
112                                         OUT[15:0] = 16'b0000_0000_0000_0000;
113 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣OUT[16]␣=␣1'b0;
114                                         OUT[17] = 1'b0;
115 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣OUT[18]␣=␣1'b0;
116                                         OUT[19] = 1'b0;
117 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣end
118 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣endcase
119 ␣␣␣␣␣␣␣␣end
120 endfunction
```

2.2 shifter

2.2.1 回路の仕様

入力・出力

- input [15:0] BR
- input [3:0] d
- input [3:0] op
- output [15:0] out
- output [3:0] SZCV

4 ビットの操作コード op を受け取り、それに応じて 16 ビットの数値 BR を d ビットシフトさせ、16 ビットの計算結果と対応した SZCV を出力する。

2.2.2 回路の設計

d を 16 ビットに符号拡張したものを signExtendedD とする。16 から d を減算した値を d_SLL_S とする。これは SLL 命令を受け取ったときの C を計算する際に用いる。BR を 2 つ連結させ 32 ビットとしたものを doubleBR とし、これを d だけ左シフトさせたものを shiftedBR_SLR とする。これは循環左シフトの計算に用いる。BR を 32 ビット符号拡張したものを extendedBR とし、これを d だけ右シフトさせたものを shiftedBR_SRA とする。これらの計算は assign 部で行う。また、assign 部で BR,signExtendedD,op,d_SLL_S,shiftedBR_SLR,shiftedBR_SRA を受け取って、SZCV,out を返す関数 OUT の計算を行う。

ソースコード 3: shifter の assign 文

```
1 assign signExtendedD = {{12{1'b0}},␣d};
2 assign␣d\_SLL\_S␣=␣16'b0000\_0000\_0001\_0000 − d;
3 assign doubleBR = {BR,BR};
4 assign shiftedBR\_SLR = doubleBR << d;
5 assign extendedBR = {{16{BR[15]}}, BR};
6 assign shiftedBR\_SRA = extendedBR >> d;
7 assign {SZCV,out} = OUT(BR,signExtendedD,op,d\_SLL\_S,shiftedBR\_SLR,shiftedBR\_SRA
      );
```

OUT 関数は BR,signExtendedD,op,d_SLL_S,shiftedBR_SLR,shiftedBR_SRA を受け取って SZCV,out を返す関数である。OUT の出力については、20 ビット目が SZCV の S,19 ビット目が SZCV の Z,18 ビット目が SZCV の C,17 ビット目が SZCV の V,0〜16 ビットが演算結果となっている。関数内において BR,signExtendedD,op,d_SLL_S,shiftedBR_SLR,shiftedBR_SRA は BR_PARM,D.OP,D_SLL_S,BR_SLR,BR_SRA としている。OUT 関数では case 文を用いることで、OP によって出力を変えている。

SLL、SLR、SRL、SRA 命令を受け取ったときの説明をする。S はシフトした結果の最上位ビットなので OUT[19] = OUT[15]; で計算できる。また、Z は演算結果が 0 と等しいかの真理値と同じになるので、OUT[18] = (OUT[15:0] == 0); で計算できる。また、V は常に 0 となる。また、C はシフト桁数が 0 の時または SLR では 0 が、それ以外では最後にシフトアウトされたビットの値が設定される。

SLL 命令のとき、シフトした結果は BR_PARM ¡¡ D で計算できる。C は D が 0 でないとき BR_PARM の 16-D ビット目の値となるので、OUT[17] = BR_PARAM[D_SLL_S] で計算できる。

SLR 命令のとき、シフトした結果は BR_SLR の上位 16 ビットの値と等しいため、OUT[15:0] = BR_SLR[31:16]; で計算できる。C は仕様書より常に 0 となる。

SRL 命令のとき、シフトした結果は BR_PARAM ¿¿ D で計算できる。C は D が 0 でないとき BR_PARAM の D-1 ビット目の値となるので、OUT[17] = BR_PARAM[D-1]; で計算できる。

SRA 命令のとき、シフトした結果は BR_SRA の下位 16 ビットの値と等しいため、OUT[15:0] = BR_SRA[15:0]; で計算できる。C は SRL と同じ値となるため、OUT[17] = BR_PARAM[D-1]; で計算できる。

その他の命令を受け取ったとき、演算結果と SZCV は 0 を出力する。

以上の仕様を実装した OUT 関数は以下のようになる。

ソースコード 4: shifter の OUT 関数

```
function [19:0] OUT;//OUT={S,Z,C,V,out}
        input [15:0] BR_PARAM;
        input [15:0] D;
        input [3:0] OP;
        input [15:0] D_SLL_S;
        input [31:0] BR_SLR;
        input [31:0] BR_SRA;

        begin
        case (OP)
                4'b0000:begin//ADD
                                OUT[15:0] = 16'b0000_0000_0000_0000;
                                OUT[16] = 1'b0;
                                OUT[17] = 1'b0;
                                OUT[18] = 1'b0;
                                OUT[19] = 1'b0;
                                end
                4'b0001:begin//SUB
                                OUT[15:0] = 16'b0000_0000_0000_0000;
                                OUT[16] = 1'b0;
                                OUT[17] = 1'b0;
                                OUT[18] = 1'b0;
                                OUT[19] = 1'b0;
                                end
                4'b0010:begin//AND
                                OUT[15:0] = 16'b0000_0000_0000_0000;
                                OUT[16] = 1'b0;
                                OUT[17] = 1'b0;
                                OUT[18] = 1'b0;
                                OUT[19] = 1'b0;
                                end
                4'b0011:begin//OR
                                OUT[15:0] = 16'b0000_0000_0000_0000;
```

```
                    OUT[16] = 1'b0;
                    OUT[17] = 1'b0;
                    OUT[18] = 1'b0;
                    OUT[19] = 1'b0;
                    end
          4'b0100:begin//XOR
                    OUT[15:0] = 16'b0000_0000_0000_0000;
                    OUT[16] = 1'b0;
                    OUT[17] = 1'b0;
                    OUT[18] = 1'b0;
                    OUT[19] = 1'b0;
                    end
          4'b0101:begin//CMP
                    OUT[15:0] = 16'b0000_0000_0000_0000;
                    OUT[16] = 1'b0;
                    OUT[17] = 1'b0;
                    OUT[18] = 1'b0;
                    OUT[19] = 1'b0;
                    end
          4'b0110:begin//MOV
                    OUT[15:0] = 16'b0000_0000_0000_0000;
                    OUT[16] = 1'b0;
                    OUT[17] = 1'b0;
                    OUT[18] = 1'b0;
                    OUT[19] = 1'b0;
                    end
          4'b0111:begin//(reserved)
                    OUT[15:0] = 16'b0000_0000_0000_0000;
                    OUT[16] = 1'b0;
                    OUT[17] = 1'b0;
                    OUT[18] = 1'b0;
                    OUT[19] = 1'b0;
                    end
          4'b1000:begin//SLL
                    OUT[15:0] = BR_PARAM << D;//out
                    OUT[16] = 1'b0;//V
                         if(D==0) begin
                              OUT[17] = 1'b0;
                         end else begin
                              OUT[17] = BR_PARAM[D_SLL_S];//C
                         end
                    OUT[18] = (OUT[15:0] == 0);//Z
                    OUT[19] = OUT[15];//S
                    end
          4'b1001:begin//SLR
                    OUT[15:0] = BR_SLR[31:16];
                    OUT[16] = 1'b0;//V
                    OUT[17] = 1'b0;//C
                    OUT[18] = (OUT[15:0] == 0);//Z
                    OUT[19] = OUT[15];//S
                    end
          4'b1010:begin//SRL
                    OUT[15:0] = BR_PARAM >> D;
                    OUT[16] = 1'b0;//V
                         if(D==0) begin
                              OUT[17] = 1'b0;
                         end else begin
                              OUT[17] = BR_PARAM[D-1];//C
                         end
                    OUT[18] = (OUT[15:0] == 0);//Z
                    OUT[19] = OUT[15];//S
                    end
          4'b1011:begin//SRA
                    OUT[15:0] = BR_SRA[15:0];
                    OUT[16] = 1'b0;//V
                         if(D==0) begin
                              OUT[17] = 1'b0;
                         end else begin
                              OUT[17] = BR_PARAM[D-1];//C
```

```
                    end
                OUT[18] = (OUT[15:0] == 0);//Z
                OUT[19] = OUT[15];//S
                end
        4'b1100:begin//IN
                OUT[15:0] = 16'b0000_0000_0000_0000;
                OUT[16] = 1'b0;
                OUT[17] = 1'b0;
                OUT[18] = 1'b0;
                OUT[19] = 1'b0;
                end
        4'b1101:begin//OUT
                OUT[15:0] = 16'b0000_0000_0000_0000;
                OUT[16] = 1'b0;
                OUT[17] = 1'b0;
                OUT[18] = 1'b0;
                OUT[19] = 1'b0;
                end
        4'b1110:begin//(reserved)
                OUT[15:0] = 16'b0000_0000_0000_0000;
                OUT[16] = 1'b0;
                OUT[17] = 1'b0;
                OUT[18] = 1'b0;
                OUT[19] = 1'b0;
                end
        4'b1111:begin//halt()
                OUT[15:0] = 16'b0000_0000_0000_0000;
                OUT[16] = 1'b0;
                OUT[17] = 1'b0;
                OUT[18] = 1'b0;
                OUT[19] = 1'b0;
                end
        endcase
    end
endfunction
```

2.3 externalOutput(外部出力)

2.3.1 回路の仕様

入力・出力

- input [15:0] AR
- input outputEnable,clock,reset,changeEnable
- output reg [7:0] SEG_A,SEG_B,SEG_C,SEG_D,SEG_E,SEG_F,SEG_G,SEG_H
- output select

1 ビットの outputEnable,clock,reset,changeEnable と 16 ビットの AR を受け取り、それに応じて FPGA ボード上の 7SEGLED を点灯させる。16 ビットの数値を格納する 16 個のレジスタ A〜P を用意し、reset が 0 かつ changeEnable が 1 かつ outputEnable が 1 のときに、クロックが立ち上がったときに P が O の値を、O が N の値を、...、B が A の値を、A が AR の値を格納して各レジスタの値を更新する。

また、クロックごとに表示する 7SEGLED を変えることで見かけ上同時に各 LED が点灯しているようにする。また、各レジスタの値は 16 進数 4 ビットの数値に変換して FPGA ボード上の 7SEGLED を図 3 のように点灯させる。

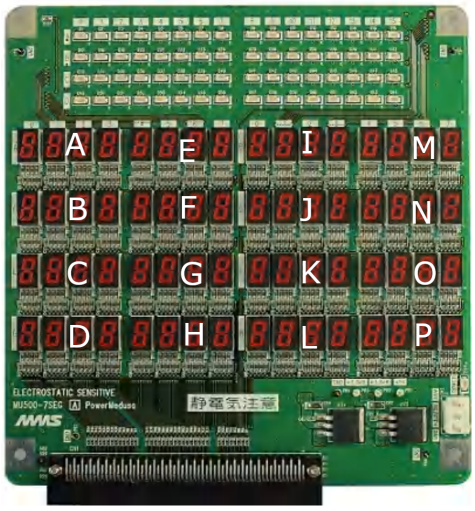


図 3: 各レジスタのボード上の割り当て

2.3.2 回路の設計

各レジスタははじめ 0 を格納するようにする。

outFunc,outFinalFunc 関数は 4 ビットの数値を受け取って、その値に応じて 16 進数の値として 7SEGLED が点灯するように 8 ビットの数値を返す。outFunc の点灯のさせ方は図 4 のようにする。outFinalFunc は outFunc の点灯に加えて dp の部分も点灯させる。この outFinalFunc は各レジスタの下位 4 ビットの値に適応させることで、表示したときに数値の境界がわかりやすくなるようにするために実装した。

A[7..0]	View	A[7..0]	View	A[7..0]	View	A[7..0]	View
1111 1100	0	0110 0110	4	1111 1110	8	0001 1010	c
0110 0000	1	1011 0110	5	1111 0110	9	0111 1010	d
1101 1010	2	1011 1110	6	1110 1110	A	1001 1110	E
1111 0010	3	1110 0000	7	0011 1110	b	1000 1110	F

図 4: 点灯のさせ方

ソースコード 5: externalOutput の outFunc・outFinalFunc

```
1  function [7:0] outFunc;
2  input [3:0] a;
3      begin
4          case (a)
5          4'b0000:outFunc = 8'b1111_1100;
6          4'b0001:outFunc = 8'b0110_0000;
7          4'b0010:outFunc = 8'b1101_1010;
8          4'b0011:outFunc = 8'b1111_0010;
9          4'b0100:outFunc = 8'b0110_0110;
10         4'b0101:outFunc = 8'b1011_0110;
11         4'b0110:outFunc = 8'b1011_1110;
```

```
12          4'b0111:outFunc = 8'b1110_0000;
13          4'b1000:outFunc = 8'b1111_1110;
14          4'b1001:outFunc = 8'b1111_0110;
15          4'b1010:outFunc = 8'b1110_1110;
16          4'b1011:outFunc = 8'b0011_1110;
17          4'b1100:outFunc = 8'b0001_1010;
18          4'b1101:outFunc = 8'b0111_1010;
19          4'b1110:outFunc = 8'b1001_1110;
20          4'b1111:outFunc = 8'b1000_1110;
21          default:outFunc = 8'b0000_0000;
22      endcase
23  end
24  endfunction
25
26  function [7:0] outFinalFunc;
27  input [3:0] b;
28      begin
29      case (b)
30      4'b0000:outFinalFunc = 8'b1111_1101;
31      4'b0001:outFinalFunc = 8'b0110_0001;
32      4'b0010:outFinalFunc = 8'b1101_1011;
33      4'b0011:outFinalFunc = 8'b1111_0011;
34      4'b0100:outFinalFunc = 8'b0110_0111;
35      4'b0101:outFinalFunc = 8'b1011_0111;
36      4'b0110:outFinalFunc = 8'b1011_1111;
37      4'b0111:outFinalFunc = 8'b1110_0001;
38      4'b1000:outFinalFunc = 8'b1111_1111;
39      4'b1001:outFinalFunc = 8'b1111_0111;
40      4'b1010:outFinalFunc = 8'b1110_1111;
41      4'b1011:outFinalFunc = 8'b0011_1111;
42      4'b1100:outFinalFunc = 8'b0001_1011;
43      4'b1101:outFinalFunc = 8'b0111_1011;
44      4'b1110:outFinalFunc = 8'b1001_1111;
45      4'b1111:outFinalFunc = 8'b1000_1111;
46      default:outFinalFunc = 8'b0000_0001;
47          endcase
48      end
49  endfunction
```

always 部について、以下のように実装することで、reset 時にレジスタの値を 0 にすることと、reset が 0 かつ changeEnable が 1 かつ outputEnable が 1 のときに、クロックが立ち上がったときに P が O の値を、O が N の値を、...、B が A の値を、A が AR の値を格納して各レジスタの値を更新することを実現している。

ソースコード 6: externalOutput のレジスタ更新部分

```
1   if(reset) begin
2           regP <= 16'b0000_0000_0000_0000;
3       regO <= 16'b0000_0000_0000_0000;
4           regN <= 16'b0000_0000_0000_0000;
5       regM <= 16'b0000_0000_0000_0000;
6           regL <= 16'b0000_0000_0000_0000;
7       regK <= 16'b0000_0000_0000_0000;
8           regJ <= 16'b0000_0000_0000_0000;
9       regI <= 16'b0000_0000_0000_0000;
10          regH <= 16'b0000_0000_0000_0000;
11      regG <= 16'b0000_0000_0000_0000;
12          regF <= 16'b0000_0000_0000_0000;
13      regE <= 16'b0000_0000_0000_0000;
14          regD <= 16'b0000_0000_0000_0000;
15      regC <= 16'b0000_0000_0000_0000;
16          regB <= 16'b0000_0000_0000_0000;
17      regA <= 16'b0000_0000_0000_0000;
18      end else begin
19          if(changeEnable) begin
20              if(outputEnable) begin
21                  regP <= regO;
22                  regO <= regN;
```

```
                    regN <= regM;
                    regM <= regL;
                    regL <= regK;
                    regK <= regJ;
                    regJ <= regI;
                    regI <= regH;
                    regH <= regG;
                    regG <= regF;
                    regF <= regE;
                    regE <= regD;
                    regD <= regC;
                    regC <= regB;
                    regB <= regA;
                    regA <= AR;
                end
            end
        end
    end
```

また、always 部について次のような実装を行うことで、LED の点灯を行う。

各 LED を点灯させる出力 SEG_A〜H とセレクタ信号である出力 select を用意する。

SEG_A〜H にレジスタ A, レジスタ E に応じた値を格納→ 1 クロック後に A,E の部分が点灯するようにセレクタ信号を更新→ 2 クロック後にセレクタ信号を 0 にする

↓

1 クロック後に SEG_A〜H にレジスタ B, レジスタ F に応じた値を格納→ 1 クロック後に B,F の部分が点灯するようにセレクタ信号を更新→ 2 クロック後にセレクタ信号を 0 にする

↓

1 クロック後に SEG_A〜H にレジスタ C, レジスタ G に応じた値を格納...

のようなループを行えるような場合分けをようにするためのカウンタ pattern を実装する。この pattern は 4 ビットであり、入力されたクロック信号が立ち上がるごとに値が増加していく。pattern による場合分けと、値に応じた動作の設定を行う。このように、SEG_A〜H の更新と select による点灯のタイミングをずらすことで正常にボード上に各レジスタの値を表示できるようにした。以上の実装を行った always 部の一部は以下のようになっている。

ソースコード 7: externalOutput の LED 更新部分

```
if(pattern == 5'b00000) begin
    SEG_A <= outFunc(regA[15:12]);
    SEG_B <= outFunc(regA[11:8]);
    SEG_C <= outFunc(regA[7:4]);
    SEG_D <= outFinalFunc(regA[3:0]);
    SEG_E <= outFunc(regE[15:12]);
    SEG_F <= outFunc(regE[11:8]);
    SEG_G <= outFunc(regE[7:4]);
    SEG_H <= outFinalFunc(regE[3:0]);
end else if(pattern == 5'b00001) begin
    select <= 8'b1000_0000;
end else if(pattern == 5'b00011) begin
    select <= 8'b0000_0000;
end else if(pattern == 5'b00100) begin
    SEG_A <= outFunc(regB[15:12]);
    SEG_B <= outFunc(regB[11:8]);
    SEG_C <= outFunc(regB[7:4]);
    SEG_D <= outFinalFunc(regB[3:0]);
    SEG_E <= outFunc(regF[15:12]);
    SEG_F <= outFunc(regF[11:8]);
    SEG_G <= outFunc(regF[7:4]);
    SEG_H <= outFinalFunc(regF[3:0]);
end else if(pattern == 5'b00101) begin
    select <= 8'b0100_0000;
end else if(pattern == 5'b00111) begin
```

```
    select <= 8'b0000_0000;
end else if(pattern == 5'b01000) begin
    SEG_A <= outFunc(regC[15:12]);
    SEG_B <= outFunc(regC[11:8]);
    SEG_C <= outFunc(regC[7:4]);
    SEG_D <= outFinalFunc(regC[3:0]);
    SEG_E <= outFunc(regG[15:12]);
    SEG_F <= outFunc(regG[11:8]);
    SEG_G <= outFunc(regG[7:4]);
    SEG_H <= outFinalFunc(regG[3:0]);
end else if(pattern == 5'b01001) begin
    select <= 8'b0010_0000;
end else if(pattern == 5'b01011) begin
    select <= 8'b0000_0000;
end else if(pattern == 5'b01100) begin
    SEG_A <= outFunc(regD[15:12]);
    SEG_B <= outFunc(regD[11:8]);
    SEG_C <= outFunc(regD[7:4]);
    SEG_D <= outFinalFunc(regD[3:0]);
    SEG_E <= outFunc(regH[15:12]);
    SEG_F <= outFunc(regH[11:8]);
    SEG_G <= outFunc(regH[7:4]);
    SEG_H <= outFinalFunc(regH[3:0]);
end else if(pattern == 5'b01101) begin
    select <= 8'b0001_0000;
end else if(pattern == 5'b01111) begin
    select <= 8'b0000_0000;
end else if(pattern == 5'b10000) begin
    SEG_A <= outFunc(regI[15:12]);
    SEG_B <= outFunc(regI[11:8]);
    SEG_C <= outFunc(regI[7:4]);
    SEG_D <= outFinalFunc(regI[3:0]);
    SEG_E <= outFunc(regM[15:12]);
    SEG_F <= outFunc(regM[11:8]);
    SEG_G <= outFunc(regM[7:4]);
    SEG_H <= outFinalFunc(regM[3:0]);
end else if(pattern == 5'b10001) begin
    select <= 8'b0000_1000;
end else if(pattern == 5'b10011) begin
    select <= 8'b0000_0000;
end else if(pattern == 5'b10100) begin
    SEG_A <= outFunc(regJ[15:12]);
    SEG_B <= outFunc(regJ[11:8]);
    SEG_C <= outFunc(regJ[7:4]);
    SEG_D <= outFinalFunc(regJ[3:0]);
    SEG_E <= outFunc(regN[15:12]);
    SEG_F <= outFunc(regN[11:8]);
    SEG_G <= outFunc(regN[7:4]);
    SEG_H <= outFinalFunc(regN[3:0]);
end else if(pattern == 5'b10101) begin
    select <= 8'b0000_0100;
end else if(pattern == 5'b10111) begin
    select <= 8'b0000_0000;
end else if(pattern == 5'b11000) begin
    SEG_A <= outFunc(regK[15:12]);
    SEG_B <= outFunc(regK[11:8]);
    SEG_C <= outFunc(regK[7:4]);
    SEG_D <= outFinalFunc(regK[3:0]);
    SEG_E <= outFunc(regO[15:12]);
    SEG_F <= outFunc(regO[11:8]);
    SEG_G <= outFunc(regO[7:4]);
    SEG_H <= outFinalFunc(regO[3:0]);
end else if(pattern == 5'b11001) begin
    select <= 8'b0000_0010;
end else if(pattern == 5'b11011) begin
    select <= 8'b0000_0000;
end else if(pattern == 5'b11100) begin
    SEG_A <= outFunc(regL[15:12]);
    SEG_B <= outFunc(regL[11:8]);
```

```
95        SEG_C <= outFunc(regL[7:4]);
96        SEG_D <= outFinalFunc(regL[3:0]);
97        SEG_E <= outFunc(regP[15:12]);
98        SEG_F <= outFunc(regP[11:8]);
99        SEG_G <= outFunc(regP[7:4]);
100       SEG_H <= outFinalFunc(regP[3:0]);
101   end else if(pattern == 5'b11101) begin
102   ␣␣␣␣select␣<=␣8'b0000_0001;
103   end else if(pattern == 5'b11111) begin
104   ␣␣␣␣select␣<=␣8'b0000_0000;
105   end
106
107   pattern <= pattern + 1;
```

2.4 フォワーディング

2.4.1 回路の仕様

まだ実装できていないため、現時点での仕様の予定を記す。データが利用可能になったら直ちに、レジスタファイルかた読みだせるようになる前に、必要とする任意のユニットにデータをフォワーディングすることを実現する部品である。データハザードが起きる条件のときにフォワーディングが動作できるようにする。

2.4.2 回路の設計

まだ実装できていないため割愛する。