計算機組織

Project1

四電機三乙

劉錕笙

B10607118

Part1：Implement a 32-bit Complete ALU

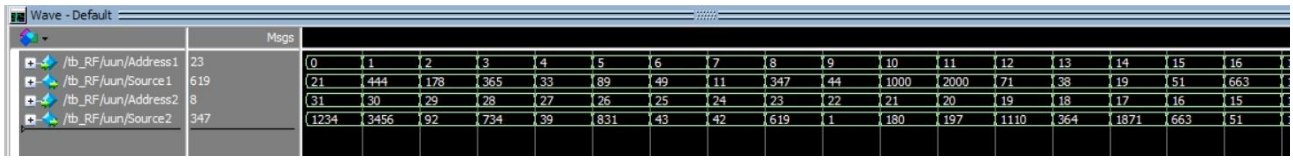
1. RF、tb_RF、波形圖

```
module RF(
        input [4:0] Address1, //Source 1 address
        input [4:0] Address2, //Source 2 address
        output [31:0] Source1, //Source 1 value
        output [31:0] Source2); //Source 2 value
FindData Src1(.Address(Address1),.Data(Source1));
FindData Src2(.Address(Address2),.Data(Source2));
endmodule

module FindData(
        input [4:0]Address,
        output reg [31:0]Data);
always @(Address)begin
        case(Address)
                5'd0:Data=21;    5'd1:Data=444;
                5'd2:Data=178;   5'd3:Data=365;
                5'd4:Data=33;    5'd5:Data=89;
                5'd6:Data=49;    5'd7:Data=11;
                5'd8:Data=347;   5'd9:Data=44;
                5'd10:Data=1000;5'd11:Data=2000;
                5'd12:Data=71;   5'd13:Data=38;
                5'd14:Data=19;   5'd15:Data=51;
                5'd16:Data=663;  5'd17:Data=1871;
                5'd18:Data=364;  5'd19:Data=1110;
                5'd20:Data=197;  5'd21:Data=180;
                5'd22:Data=1;    5'd23:Data=619;
                5'd24:Data=42;   5'd25:Data=43;
                5'd26:Data=831;  5'd27:Data=39;
                5'd28:Data=734;  5'd29:Data=92;
                5'd30:Data=3456;5'd31:Data=1234;
        endcase
end
endmodule
```

暫存器資料讀取

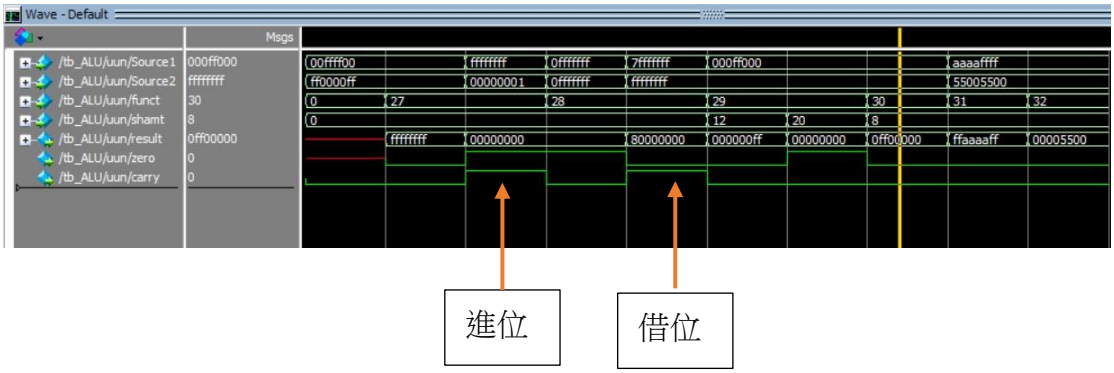
```
module tb_RF();
reg [4:0]add1,add2;
wire [31:0]Src1,Src2;

RF uun(.Address1(add1),.Address2(add2),.Source1(Src1),.Source2(Src2));

initial #200 $finish;
initial begin
add1=0;
add2=31;
end
always begin
#5
add1=add1+1;
add2=add2-1;

end
endmodule
```

每 5ns 更換一次數值，確認 RF 內資料正確



2. ALU、tb_ALU、波形圖

```
module ALU(
        input [31:0] Source1,
        input [31:0] Source2,
        input [5:0] funct,
        input [4:0] shamt,
        output reg [31:0] result,
        output zero,
        output reg carry
        );

nor ZFlag(zero,result[0],result[1],result[2],result[3],result[4],result[5],result[6],result[7]
                ,result[8],result[9],result[10],result[11],result[12],result[13],result[14],result[15]
                ,result[16],result[17],result[18],result[19],result[20],result[21],result[22],result[23]
                ,result[24],result[25],result[26],result[27],result[28],result[29],result[30],result[31]);

always @(Source1,Source2,funct,shamt)begin
        carry=0;
        case(funct)
                6'd27:{carry,result} = Source1+Source2;
                6'd28:{carry,result} = Source1-Source2;
                6'd29:result <= Source1>>shamt;
                6'd30:result <= Source1<<shamt;
                6'd31:result <= Source1^Source2;
                6'd32:result <= Source1&Source2;
        endcase
end

endmodule
module tb_ALU();
        reg [31:0] Src1;
        reg [31:0] Src2;
        reg [5:0] funct;
        reg [4:0] shamt;
        wire [31:0] result;
        wire zero;
        wire carry;

ALU uun(.Source1(Src1),.Source2(Src2),.funct(funct),.shamt(shamt),.result(result),.zero(zero),.carry(carry));

initial begin
Src1=0;Src2=0;funct=0;shamt=0;
end
initial begin
#10 funct=27;Src1=32'H00FFFF00;Src2=32'HFF0000FF;       //測試所有位元加法
#10 Src1=32'HFFFFFFFF; Src2=32'H00000001;               //測試加法進位
#10 funct=28;Src1=32'H0FFFFFFF; Src2=32'H0FFFFFFF;      //測試減法與zero flag
#10 Src1=32'H7FFFFFFF;Src2=32'HFFFFFFFF;                //測試減法借位
#10 funct=29;Src1=32'H000FF000;shamt=12;                //測試右移
#10 shamt=20;                                           //測試Data移出
#10 funct=30;shamt=8;                                   //測試左移
#10 funct=31;Src1=32'HAAAAFFFF;Src2=32'H55005500;       //測試Xor
#10 funct=32;                                           //測試And
end
endmodule
```



3. CompALU、tb_CompALU、波形圖

```
module CompALU(
        input [31:0] Instruction, //Instruction input
        output [31:0] result, //Arithmetic result
        output zero, //Zero flag
        output carry //Carry flag
        );

wire [31:0]Src1,Src2;

RF mRF(.Address1(Instruction[25:21]),.Address2(Instruction[20:16]),.Source1(Src1),.Source2(Src2));
ALU mALU(.Source1(Src1),.Source2(Src2),.funct(Instruction[5:0]),.shamt(Instruction[10:6]),.result(result),.zero(zero),.carry(carry));

endmodule
```

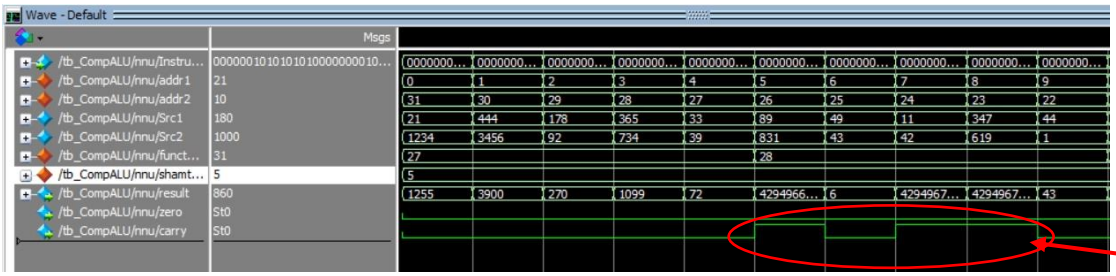
```
module tb_CompALU();
reg [31:0] Instruction;
wire [31:0] result;
wire zero;
wire carry;

CompALU nnu(.Instruction(Instruction),.result(result),.zero(zero),.carry(carry));
initial begin
Instruction=0;
Instruction[25:21]=0; //address1
Instruction[20:16]=31; //assress2
Instruction[10:6]=5;  //shamt
Instruction[5:0]=27;   //funct
end
always begin
#10
Instruction[25:21]=Instruction[25:21]+1; //address1
Instruction[20:16]=Instruction[20:16]-1; //assress2
end
initial begin
#50 Instruction[5:0]=28;   //funct
#50 Instruction[5:0]=29;   //funct
#50 Instruction[5:0]=30;   //funct
#50 Instruction[5:0]=31;   //funct
#50 Instruction[5:0]=32;   //funct
#50 Instruction[5:0]=30;   //funct
#50 Instruction[5:0]=28;   //funct
#50 Instruction[5:0]=50;   //funct
#50 Instruction[5:0]=29;   //funct
#50
$finish;
end
endmodule
```

各參數初始化
移位都使用 5

Src 內的資料用
迴圈自動讀取
用來確保不同數
值的運算皆正常

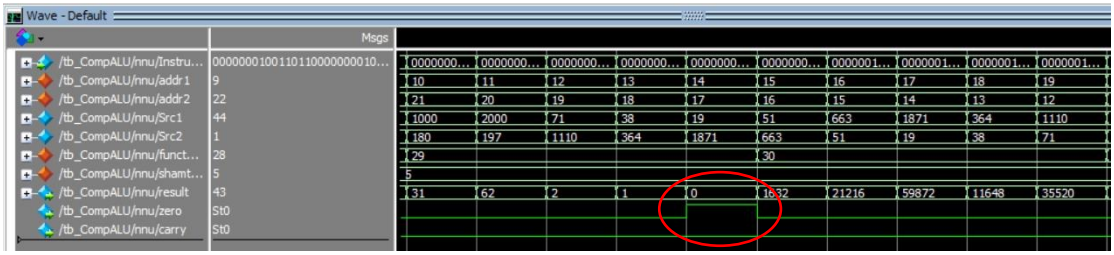
每種功能皆測試
5 種不同的 Data

Funct=27 =>加法　　Funct=28 =>減法

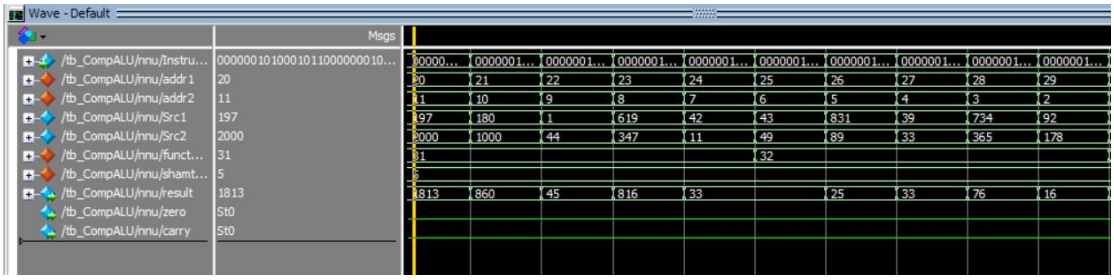


減法借位

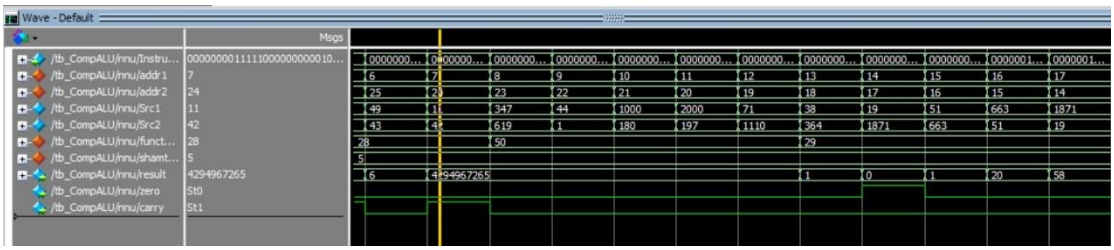
Funct=29 =>右移 5=Src1*2^5　　Funct=30=>左移 5=Src1/2^5



Funct=31 =>Xor　　Funct=32 =>And



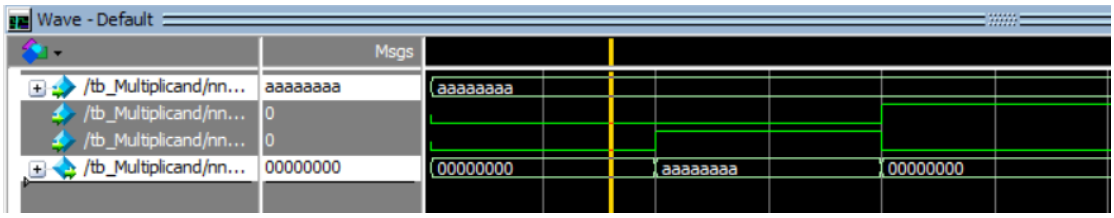
Funct=50 =>Default 不動作



Part2: Implement a 32-bits multiplier

1. ALU 同 part1，直接拿來使用。
2. Multiplicand 與 tb_Multiplicand

```
module Multiplicand(
        input [31:0] Multiplicand_input, //Multiplicand input
        input rst, //Reset multiplicand
        input wrctrl, //write the multiplicand into register
        output reg [31:0] Multiplicand_output=0 //Multiplicand output
        );
always @(posedge rst,posedge wrctrl)begin
        if(rst==1)Multiplicand_output<=0;
        else if(wrctrl==1)Multiplicand_output<=Multiplicand_input;

end
endmodule

module tb_Multiplicand();
        reg [31:0]in;
        reg rst,wr;
        wire [31:0]out;

Multiplicand nnu(.Multiplicand_input(in),.rst(rst),.wrctrl(wr),.Multiplicand_output(out));

initial begin
in=32'HAAAAAAAA;
rst=0;
wr=0;
#10 wr=1;

#10 wr=0;rst=1;


end
endmodule
```

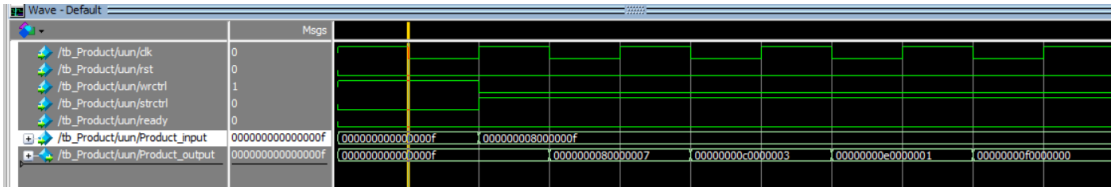


Multiplicand.v：

rst 優先權大於 wrctrl，同時只在 wrctrl 正緣觸發，因此 Control.v 在設計控制 wrctrl 時要復歸，才可再次控制。

3. Product 與 tb_Product

```
module Product(
        input [63:0] Product_input, //Product input
        input wrctrl,
        input strctrl,
        input ready,
        input rst,
        input clk,
        output reg [63:0] Product_output //Product output
        );

always @(posedge wrctrl)Product_output={32'd0,Product_input[31:0]};

always @(negedge clk,posedge rst)begin
        if(rst==1)Product_output=0;
        else if(ready!=1)begin
                if(strctrl==1)Product_output={Product_input[63:31],Product_output[31:1]};
        end
end
endmodule

module tb_Product();
         reg [63:0]Product_input;
         reg wrctrl=0;
         reg strctrl=0;
         reg ready=0;
         reg rst=0;
         reg clk=1;
         wire [63:0] Product_output;

Product uun(.Product_input(Product_input),.wrctrl(wrctrl),.strctrl(strctrl)
         ,.ready(ready),.rst(rst),.clk(clk),.Product_output(Product_output));

always begin
#5 clk=~clk;
end

initial begin
Product_input=15;
wrctrl=1;

#10 wrctrl=0;strctrl=1;Product_input[63:31]=1;

end
endmodule
```



Product.v：

Product 會在每個 clk 的負緣對 ready flag 進行一次判斷，如果 Control 沒有申明 ready(ready=1)則 Product 會執行一次右移與儲存 ALU 加法資料。直到資料 ready。

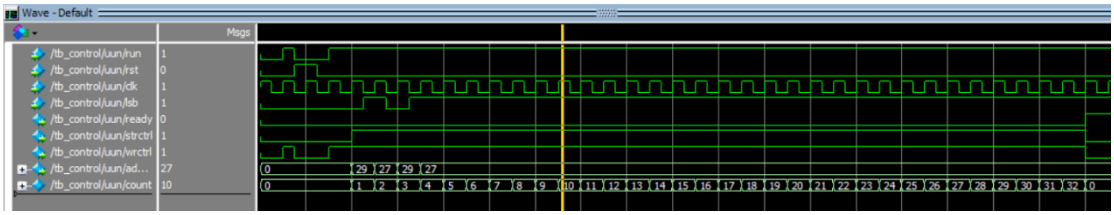
Tb_Product：

在 tb 中先給予隨機長度的乘數(15=1111)測試右側資料移出，並假設 ALU 輸出資料設計為 1(避免進位)，藉此觀察新舊資料疊放儲存的結果。

4. Control 與 tb_Control

```
module Control(
        input run,input rst,input clk,input lsb,
        output reg ready=0,output reg strctrl=0, //Signal for stc
        output reg wrctrl=0,output reg [5:0] addctrl=0);//for ALU
reg [5:0]count=0;

always @(posedge clk,posedge rst)begin
        if(rst==1)begin
                count<=0;
                ready<=0;
                strctrl<=0;
                wrctrl<=0;
                addctrl<=0;
        end
        else if(wrctrl==1)begin
                if(lsb==1)addctrl=27;
                else addctrl=29;
                strctrl=1;
                count=count+1;
        end
        if(count==33)begin
                count=0;
                wrctrl=0;
                strctrl=0;
                ready=1;
        end
end
endmodule
```

```
module tb_control();
        reg run=0;
        reg rst=0;
        reg clk=1;
        reg lsb=0;
        wire ready;
        wire strctrl; //Signal for store the ALU result into Product
        wire wrctrl;
        wire [5:0] addctrl;

Control uun(.run(run),.rst(rst),.clk(clk),.lsb(lsb),.ready(ready),.strctrl(strctrl),.wrctrl(wrctrl),.addctrl(addctrl));

always begin
#5 clk=!clk;
end

initial begin
#10 run=1;

#5 rst=1;run=0;

#10 rst=0;

#5 run=1;

#15 lsb=1;

#10 lsb=0;

#10 lsb=1;

end
endmodule
```



Control.v：

Control 設計為資料流的走向，我使用正緣觸發動作，這樣 Product 動作讀取 ALU 前會有半個 clk 週期的運算時間，避免因 t_{pd} 資料讀出錯誤的問題。

為了計算工作次數，額外設定了 Count 暫存器。Count 會在 wrctrl 動作後開始計算，程式執行 32 次後(count=33)會將 ready flag 設定為 1，並將其他 control path 復歸為 0。

乘法運算中若 lsb 為 0 時不進行運算，因為是使用 part1 的 ALU_32bits.v 做運算，因此使用的是 instruction 中的右移(funct=29)。並會在連結 ALU 時將 shamt 設定為 0。

Strctrl 會在準備好 funct 後致能，避免 product 在 wrctrl 剛寫入後立刻移位，導致資料錯誤。

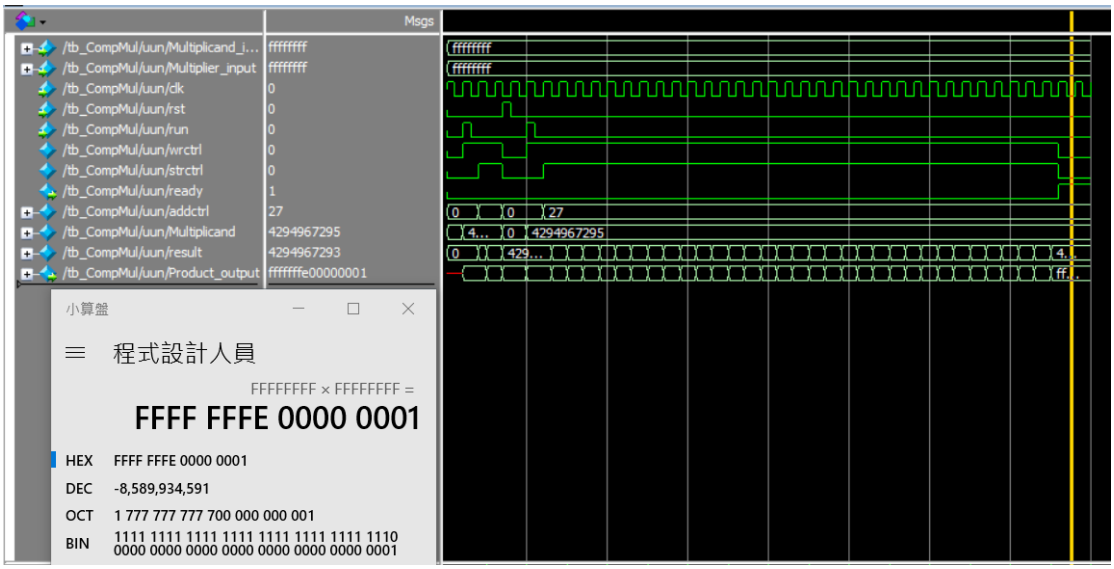
tb_Control：

主要測試 lsb=0/1 時的 addctrl 是否正確(funct=27/29)，和 count 計算次數是否正確與各 Control path 功能是否正確。

5. CompMul 與 tb_CompMul

```
module CompMul(
        input [31:0] Multiplicand_input, //Multiplicand input
        input [31:0] Multiplier_input, //Multiplier input
        input run, //Start signal
        input rst, //Reset all the control signals
        input clk, //Clock signal
        output ready, //Result ready
        output [63:0] Product_output //Arithmetic result
        );

wire strctrl,wrctrl,zero,carry;
wire [5:0]addctrl;
wire [31:0]Multiplicand;
wire  [31:0]result;
wire temp;

Control mControl(.run(run),.rst(rst),.clk(clk),.lsb(Product_output[0])
                ,.ready(ready),.strctrl(strctrl),.wrctrl(wrctrl),.addctrl(addctrl));

Multiplicand mMultiplicand(.Multiplicand_input(Multiplicand_input),.rst(rst),.wrctrl(wrctrl),.Multiplicand_output(Multiplicand));

ALU mALU(.Source1(Product_output[63:32]),.Source2(Multiplicand)
                ,.funct(addctrl),.shamt(5'b00000),.result(result),.zero(zero),.carry(carry));

Product mProduct(.Product_input({carry,result[31:1],temp,Multiplier_input[30:0]}),.wrctrl(wrctrl),.strctrl(strctrl)
        ,.ready(ready),.rst(rst),.clk(clk),.Product_output(Product_output));


assign temp=(Multiplicand==0)?Multiplier_input[31]:result[0];

endmodule
```

```
module tb_CompMul();
        reg [31:0] Multiplicand_input=0; //Multiplicand input
        reg [31:0] Multiplier_input=0; //Multiplier input
        reg run=0; //Start signal
        reg rst=0; //Reset all the control signals
        reg clk=1; //Clock signal
        wire ready; //Result ready
        wire [63:0] Product_output;

CompMul uun(.Multiplicand_input(Multiplicand_input),.Multiplier_input(Multiplier_input)
        ,.run(run),.rst(rst),.clk(clk),.ready(ready),.Product_output(Product_output));

always begin
#5 clk=~clk;
end

initial begin
Multiplicand_input=32'HFFFFFFFF;
Multiplier_input=32'HFFFFFFFF;

#10 run=1;
#5 run=0;

#20 rst=1;
#5 rst=0;

#10 run=1;
#5 run=0;


end
endmodule
```



CompMul.v：

主要將各 module 連接起來，並藉由多工器去判斷 Product_input 的共用腳位[31]，來完成接線。

Tb_ CompMul：

測試了 rst 的功能，並使用最大值進行乘法(FFFF,FFFF*FFFF,FFFF)檢測乘法器可靠度。

心得：

我覺得這次題目不算很難，唯一卡住的地方是在整合時 Product_input[31]與 result[0]共用選擇的地方，知道要使用多工器來選取，但卻找不到適合的判斷參數。一開始想使用 wrctrl，但在 Control module 設計時將 wrctrl 設定為 counter 的致能判斷，因此 wrctrl 無法用來判斷多工。也測試過使用 run 來進行判斷，後來考慮到使用者的 run data 可能和我在 testbench 中不太一樣，只好放棄。在找參數的過程中還不小心改動太多內容，導致功能爆炸，花了許多時間才調整回來。覺得 I/O 腳位事先訂好有好處也有壞處，好處是可以直接開始設計；壞處就是明明加一個腳就可以解決的問題卻花了很多時間來解套。