國立臺灣科技大學

電機工程系

計算機組織

作業報告

PA1

四電機三乙 | B10707128 | 劉杰閎

各模組程式碼截圖並說明

PART 1 乘法器(Multiplier)

a. ALU

```
module ALU(Src1, Src2,Result,Carry,Funct);      jiehong, a week ago • update from mac
input [31:0]Src1;
input [31:0]Src2;
input [5:0]Funct;
output reg [31:0]Result;
output reg Carry;

always@(Src1 or Src2 or Funct)
begin
    //  I use case to implement ALU, because ALU usually do a lot of jobs.
    case (Funct[5:0])
    6'b 001001: {Carry,Result} = Src1 + Src2;
    default: // 如果今天LSB==0→不用加的話，就不要加。
    begin
        Carry = 0;
        Result = 0;
    end
    endcase
end

endmodule
```

```
module tb_ALU();

reg [31:0]Src1;
reg [31:0]Src2;
reg [5:0]Funct;

wire [31:0]Result;
wire Carry;


ALU Arithemetic_Logical_Unit(
    .Src1(Src1),
    .Src2(Src2),
    .Funct(Funct),
    .Result(Result),
    .Carry(Carry)
);
initial #20 $finish;
initial fork
#0 Funct = 6'b 0; //' 一開始不動作

#5 Funct = 6'b 001001; // 依照HW1規定的ADDU function code
#5 Src1 = 32'h 0F0F_0F1F; // 第一個測資們，我讓他們爆炸
#5 Src2 = 32'h F0F0_F0F0;

#10 Funct = 6'b 0;// 如果funct不為MIPS所規定的function code，那Result與Carry就該為0。

#15 Funct = 6'b 001001;
#15 Src1 = 32'h 200;  // 第二個就用正常的測資
#15 Src2 = 32'h 1800;

join
endmodule
```

ALU所做的事情非常簡單，如果FUNCT[5:0] = 6’b001010(這6個bit是照著HW1所提供之MIPS function code)，就單純做unsigned的加法，因為是32bit+32bit，因此我們還需要一個Carry，來避免其overflow.若function code不是ADDU所規定之function code的話，就令Result與Carry為0。下圖為由testbench產生之模擬結果。

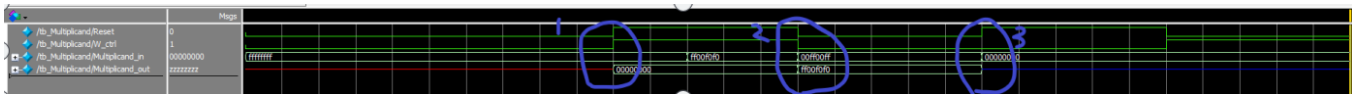
1. 為0F0F_0F1F + F0F0_F0F0，因為是設計用來overflow，因此Carry會跳1，
2. 是因為Funct = 0，所以ALU輸出為0
3. 是1800+200=1a00H

b. Multiplicand

```
module Multiplicand(Reset,W_ctrl,Multiplicand_in,Multiplicand_out);      jiehong,
input Reset;
input W_ctrl;
input [31:0]Multiplicand_in;
reg [31:0]Multiplicand_reg;
output reg [31:0]Multiplicand_out;

always@(Reset or W_ctrl or Multiplicand_in)
begin
    if(Reset == 1 && W_ctrl == 1)
        begin
            Multiplicand_reg = Multiplicand_in ;
            Multiplicand_out = 0; // 把值讀進來，但不輸出or輸出為0
        end
    // 在設計沒有錯誤的情況下，W_ctrl應該會與Reset連動，所以我將這兩個logic AND 起來
    else if (W_ctrl == 0 && Reset == 0) // means that the system is running
        begin
            Multiplicand_out = Multiplicand_reg; // let output = reg
        end
    else
        begin
            Multiplicand_out = 32'bZ; // it means that the design is failed.
        end
end


endmodule

```

```
module tb_Multiplicand();      You, 6 days ago • tb of multiplier almost finished

reg Reset,W_ctrl;
reg [31:0]Multiplicand_in;
wire [31:0]Multiplicand_out;

Multiplicand multiplicand(
    .Reset(Reset),
    .W_ctrl(W_ctrl),
    .Multiplicand_in(Multiplicand_in),
    .Multiplicand_out(Multiplicand_out)
);

initial #30 $finish;
initial fork
#0 Reset = 0; // 因為W_ctrl是由Controller所發送，所以一定會與Reset相同，那在這裡
#0 W_ctrl = 0;// 我們就依照我們設計正確的前提下，去進行test bench的測試
#0 Multiplicand_in = 32'h FFFF_FFFF; // 兩者皆為0的時候改任何值，輸出都不應該改變

#10 Reset = 1;
#10 W_ctrl = 1;
#12 Multiplicand_in = 32'h FF00_F0F0;
// 與上面兩行時間不同，是因為輸入有可能不一樣，且這個輸出應該為0

#15 Reset = 0;
#15 W_ctrl = 0;
#15 Multiplicand_in = 32'h 00FF_00FF;
// 以下測試為，系統錯誤時會發生的狀況，也就是Reset與W_ctrl不同步。
#20 Reset = 1;
#20 W_ctrl = 0;
#20 Multiplicand_in = 32'h 0;

#25 Reset = 0;
#25 W_ctrl = 1;
#25 Multiplicand_in = 32'h 0;

join

endmodule
```

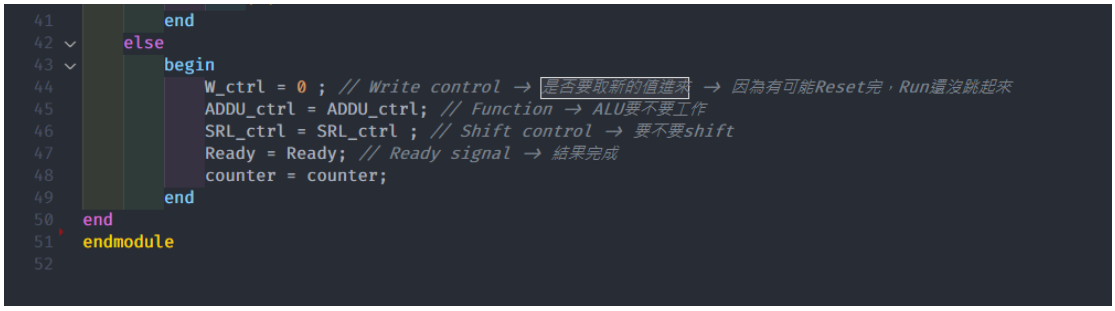
Multiplicand 主要是用來接受外部來的乘數，如果Reset訊號為1，那就代表我要讀取新的值進來了，因此我先利用一個reg來儲存這個值，到下一個clock才將值送入ALU，如果說今天輸入的狀況不如預期，我就認為是我設計錯誤而將高阻抗送入ALU。下圖為由testbench產生之模擬結果。



1之前reset=1，輸出是don’t care.1~2的時候因為reset = 1，因此輸出是0，而2~3的時候因為在1~2之間Multiplicand_in有值進來，因此此時會有reg將這個值給記錄起來在2~3的時候將他放到Multiplicand_out當作結果輸出，而在3以後，因為Reset與W_ctrl不同步，因此結果輸出高阻抗。

c. Control

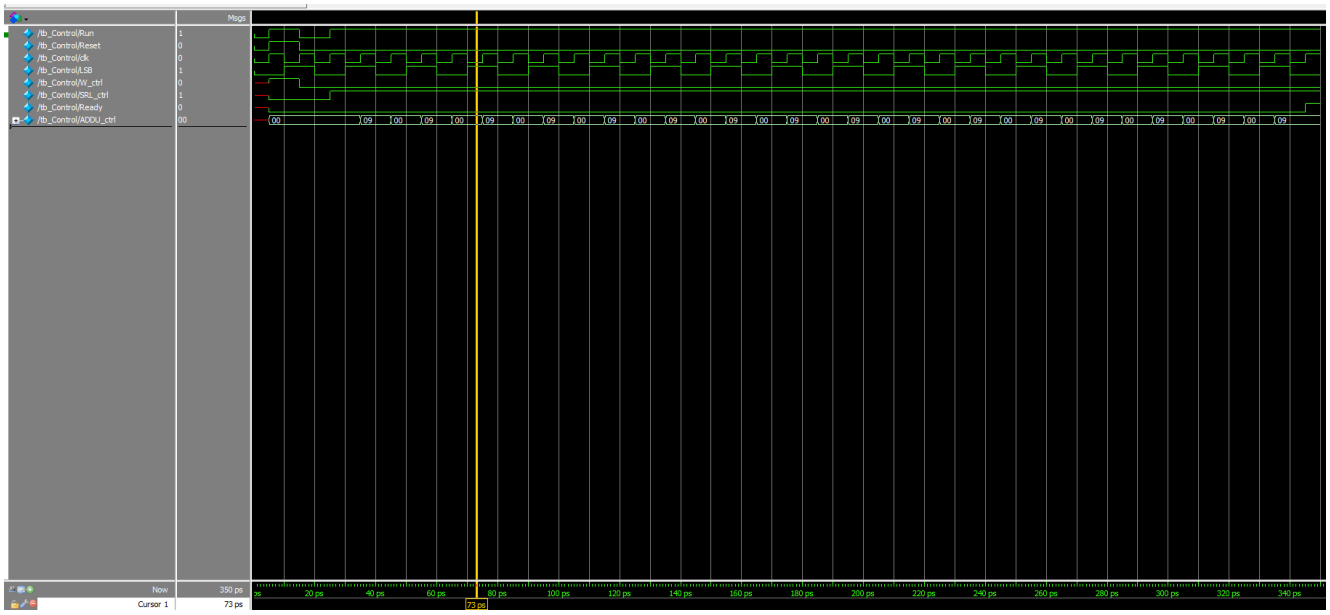
```
module Control(Run,Reset,clk,LSB,W_ctrl,ADDU_ctrl,SRL_ctrl,Ready);
input Run;
input Reset;
input clk;
input LSB;
output reg W_ctrl; // Write control
output reg [5:0]ADDU_ctrl;
output reg SRL_ctrl;
output reg Ready;
reg [5:0]counter; // counter for 32 times;
always@(posedge clk or  posedge Reset)
begin
    if(Reset == 1)
        begin
            W_ctrl = 1; // Write control → 是否要取新的值進來
            ADDU_ctrl = 6'b0; // Function → ALU要不要工作
            SRL_ctrl = 0; // Shift control → 要不要shift
            Ready = 0; // Ready signal → 結果完成
            counter = 0;
        end
    else if (Run == 1 && Ready == 0)
        begin
            if(counter == 32) // for (counter = 0 ; counter < 32; counter ++)
                begin
                    Ready = 1; // 告訴系統我們完成了
                end
            else
                begin
                    W_ctrl = 0;      // 在Run的時候不要取新的值進來      You, 5 days ago • tb of multiplier almost finished
                    SRL_ctrl = 1;   // 將結果暫存器左半累加被乘數
                    if (LSB == 1)   // Product[0] == 1 → ALU要工作，所以我讓ADDU_ctrl = 1;
                        begin
                            ADDU_ctrl = 6'b 001001;  // 並開始執行運算，
                        end
                    else if (LSB == 0)  // ALU不用工作
                        begin
                            ADDU_ctrl = 6'b0;
                        end
                    counter = counter + 1;
                end
        end
```



Control.v 代表著整個系統的controller，負責控制所有訊號的傳送，所有模組都以它為號令來決定工作與否，一開始外部訊號送Reset進來，整個系統進行初始化，代表說要讀一個新的乘數與被乘數進來了，於是將 W_ctrl = 1送出，來讀取值，ADDU_ctrl = 0送出，告訴ALU現在不用工作，SRL_ctrl=0告訴Product現在不用右移，Ready=0代表結果還沒做完，counter = 0，代表現在還沒開始做。Reset訊號送完，系統會再送一個Run訊號進來，此時我會二次確認現在Ready尚未等於1，確定系統正在執行時，我會將W_ctrl關掉，代表不可以讀新的值進來了，請把現在的值鎖住，接著將SRL_ctrl打開，因為在每次執行的時候，Product都會進行右移的動作，接著我會判斷LSB是否為1，若為1就使ALU進行加法的工作，反之則讓ALU輸出為0，做完一次counter進行+=1的動作，共計數32次，若32次就告訴系統，結果已完成，令Ready為1。若以上條件都不符合，我們就先讓W_ctrl為0，確保不會影響其他。讓其他東西都是自己的狀態，也不會影響。

```
module tb_Control();      You, 6 days ago • tb of multiplier almost finished

reg Run,Reset,clk,LSB;
wire W_ctrl,SRL_ctrl,Ready;
wire [5:0]ADDU_ctrl;

Control controller(
    .Run(Run),
    .Reset(Reset),
    .clk(clk),
    .LSB(LSB),
    .W_ctrl(W_ctrl),
    .ADDU_ctrl(ADDU_ctrl),
    .SRL_ctrl(SRL_ctrl),
    .Ready(Ready)
);

initial #350 $finish;

initial begin
    #0 clk = 0;
    forever #5 clk = ~clk;
end

initial begin
    #0 LSB = 0;
    forever #10 LSB = ~LSB;
end


initial fork
    #0 Run = 0;
    #0 Reset = 0;

    #5 Reset = 1;    // 以reset為主，所以下面為多少理論上都不會影響
    #5 Run = 1;

    #15 Reset = 0;
    #15 Run = 0;

    #25 Run = 1;     //從這裡開始計數32次 → 10*32=320，在345的時候，Ready應該要跳1
join

endmodule
```

下圖為由testbench產生之模擬結果。



動作會計數32次，從25秒開始RUN，在345秒(25+32*10(一個clk))的時候Ready會完成。

d. Product

```
module Product(SRL_ctrl,W_ctrl,Ready,Reset,clk,ALU_carry,ALU_result,Multiplier_in,Product_out);

input SRL_ctrl;
input W_ctrl;
input Ready;
input Reset;
input clk;
input ALU_carry;
input [31:0]ALU_result;
input [31:0]Multiplier_in;
output reg[63:0]Product_out;

always@(negedge clk or posedge Reset)
begin
    if (Reset == 1 && W_ctrl == 1)
        begin
            Product_out = 64'b0;
            Product_out[31:0] = Multiplier_in[31:0];
        end
    else if(SRL_ctrl == 1 && Ready == 0)
        begin
                if(Product_out[0] == 1) // means that LSB = 1;
                    begin
                        Product_out[63:32] =  ALU_result[31:0]; // 才要把Result放進Product_out
                    end
                Product_out = Product_out >> 1; // 只有shift訊號為1的時候才可以做shift
                Product_out[63] = ALU_carry;// shift 完以後把carry放到第63個bit
        end
    else if(Ready == 1)
        begin
            Product_out = Product_out ;
        end
    else;
end
endmodule
```

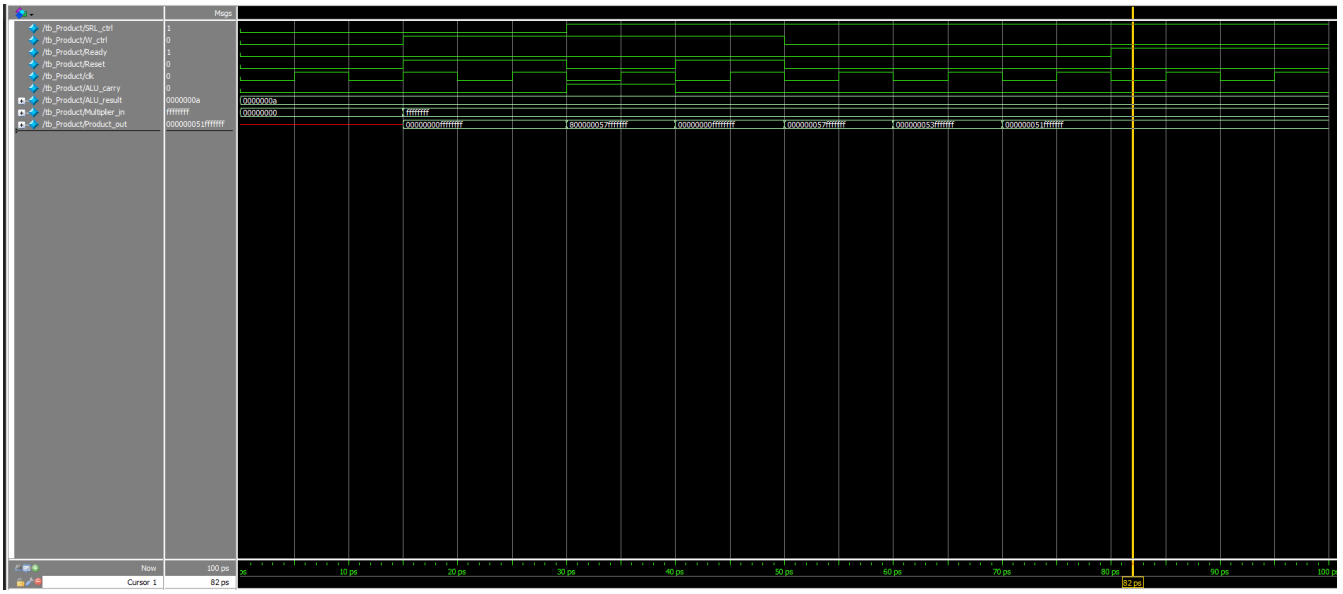
Product這個module沒什麼太複雜的工作，比較特別的是在這個地方，我是利用clk的負緣觸發，因為我的想法是clk正緣觸發後，controller會工作。從ALU正緣觸發，Product負緣觸發，我就可以在一個clk內做完一次的動作。在Reset觸發的時候，會讀乘數進來。在Ready還沒完成之前且SRL_ctrl = 1，如果此時LSB = 1，我們會把Result放進Product_out，接著，就會右移一次，接著將Carry放進我們的Product_out. 待Ready跳起時，輸出就維持不變，直到Reset訊號觸發。

```
module tb_Product();        You, 5
reg SRL_ctrl;
reg W_ctrl;
reg Ready;
reg Reset;
reg clk;
reg ALU_carry;
reg [31:0]ALU_result;
reg [31:0]Multiplier_in;
wire [63:0]Product_out;
```

```
Product pro
(
    .SRL_ctrl(SRL_ctrl),
    .W_ctrl(W_ctrl),
    .Ready(Ready),
    .Reset(Reset),
    .clk(clk),
    .ALU_carry(ALU_carry),
    .ALU_result(ALU_result),
    .Multiplier_in(Multiplier_in),
    .Product_out(Product_out)
);

initial #100 $finish;


initial begin
#0 clk = 0;
forever #5 clk = ~clk;
end
```

```
initial fork
    // 初始化
    #0 ALU_result = 32'd10;
    #0 W_ctrl = 0;
    #0 Reset = 0;
    #0 Ready = 0;
    #0 SRL_ctrl = 0;
    #0 ALU_carry = 0;
    #0 Multiplier_in = 0;

    // 系統開始初始化，product_out[31:0] = Multipler_in[31:0]
    #15 Reset = 1;
    #15 W_ctrl =1;
    #15 Multiplier_in = 32'h FFFF_FFFF;

    // 系統開始執行
    #30 Reset = 0;
    #30 SRL_ctrl = 1;
    #30 ALU_carry = 1; // 這裡的話輸出的MSB應該會是1

    #40 ALU_carry = 0; // 到這裡MSB應該要是0
    #40 Reset = 1; // 因為要把Product_out歸0，所以要重新Reset
    #40 W_ctrl = 1;

    #50 Reset = 0;
    #50 W_ctrl = 0;
    #50 Ready = 0; //這時候要讓輸出乘法器工作

    #80 Ready = 1; // 讓輸出維持不變

join

endmodule
```

下圖為由testbench產生之模擬結果，此處無特別之地方，模擬結果皆與註解所標註相同。



e. CompMultiplier

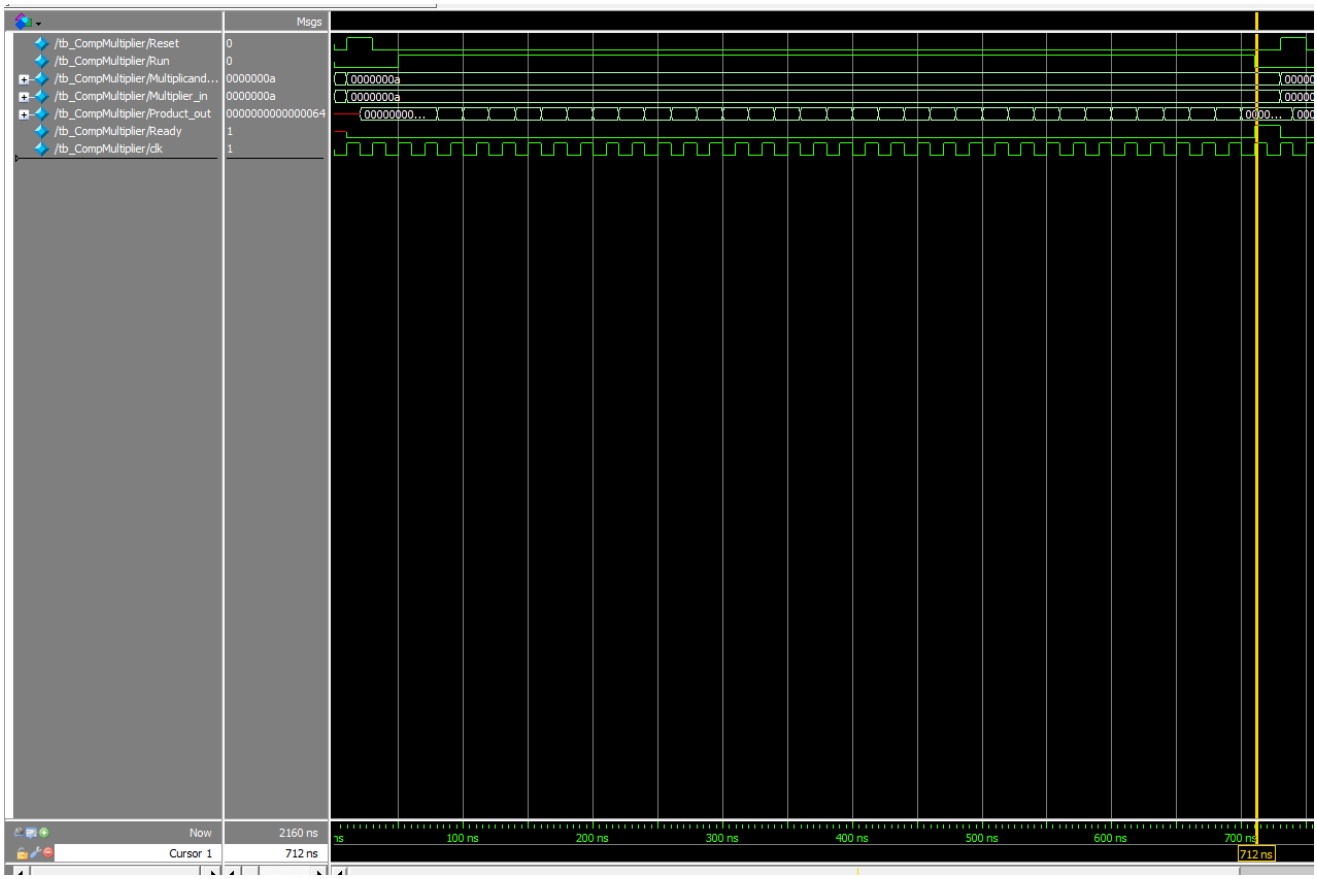
```
module CompMultiplier(clk,Reset,Run,Multiplier_in,Multiplicand_in,Product_out,Ready);
input clk;
input Reset;
input Run;
input [31:0] Multiplier_in;
input [31:0] Multiplicand_in;
output [63:0]Product_out;
output Ready;
// Multiplicand
wire W_ctrl;
wire [31:0]Multiplicand_out;
// for Product
wire SRL_ctrl,Ready,Reset,ALU_carry;
wire [31:0]ALU_result;      You, seconds ago • Uncommitted changes
wire [31:0]Multiplier_in;
wire [63:0]Product_out;
wire [5:0]ADDU_ctrl;
// control unit
Control controller
(
    .Run(Run),
    .Reset(Reset),
    .clk(clk),
    .LSB(Product_out[0]),
    .W_ctrl(W_ctrl),
    .ADDU_ctrl(ADDU_ctrl[5:0]),
    .SRL_ctrl(SRL_ctrl),
    .Ready(Ready)
);
```

```
// 被乘數
Multiplicand multiplicand
(
    .Reset(Reset),
    .W_ctrl(W_ctrl),
    .Multiplicand_in(Multiplicand_in[31:0]),
    .Multiplicand_out(Multiplicand_out[31:0])
);

// ALU
ALU Arithemetic_Logical_Unit
(
    .Src1(Product_out[63:32]),
    .Src2(Multiplicand_out[31:0]),
    .Funct(ADDU_ctrl[5:0]),
    .Carry(ALU_carry),
    .Result(ALU_result[31:0])
);

// product unit
Product product_unit
(
    .SRL_ctrl(SRL_ctrl),
    .W_ctrl(W_ctrl),
    .Ready(Ready),
    .Reset(Reset),
    .clk(clk),
    .ALU_carry(ALU_carry),
    .ALU_result(ALU_result[31:0]),
    .Multiplier_in(Multiplier_in[31:0]),
    .Product_out(Product_out[63:0])
);

endmodule
```

而最後，關於CompMultiplier這個模組，我做的則是去把Control、ALU、Multiplicand、Product這幾個模組組合起來，再藉由wire把他們從內部連接起來。Testbench直接使用助教所提供的檔案，下圖為由測試指令之in檔截圖及testbench產生之模擬結果。乘數與被乘數輸入a，Product_out經過32次的運算得到最終結果64H(10*10=100)。

PA1 > Multiplier > testbench > tb_CompMultiplier.in

```
1   0000000a_0000000a
2   000003EB_00000014
3   0fffffff_ffffffff
```


Msgs
/tb_CompMultiplier/Reset
/tb_CompMultiplier/Run
/tb_CompMultiplier/Multiplicand...
/tb_CompMultiplier/Multiplier_in
/tb_CompMultiplier/Product_out
/tb_CompMultiplier/Ready
/tb_CompMultiplier/clk
0
0
0000000a
0000000a
0000000000000064
1
1
Now
2160 ns
Cursor 1
712 ns
100 ns
200 ns
300 ns
400 ns
500 ns
600 ns
700 ns
712 ns

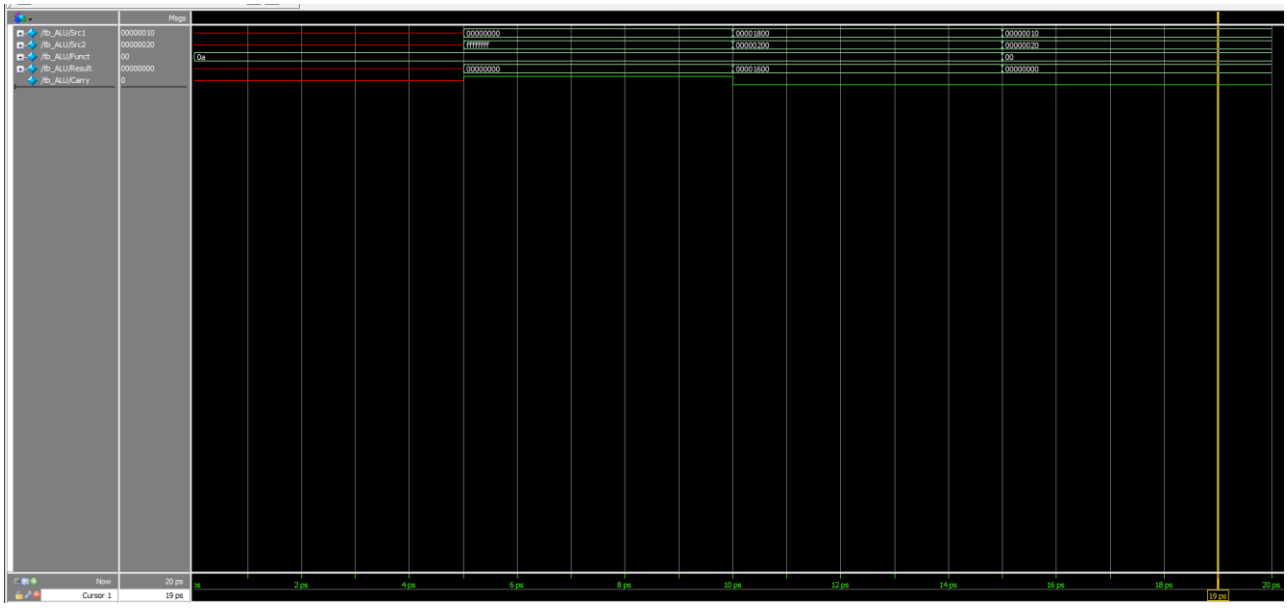
PART 2 除法器(Divider)

a. ALU

這個是ALU的模組，由case來決定是否執行減法的動作，依照除法器的要求，如果Src1 < Src2 ，輸出就是Src1，不然輸出不會改變，其他狀況輸出為0。輸出的carry可以讓Remainder知道這個輸出是否為負值。

```
module ALU(Src1, Src2,Result,Carry,Funct);      You, a week ago • tb of multiplier almost finished
input [31:0]Src1;
input [31:0]Src2;
input [5:0]Funct;
output reg [31:0]Result;
output reg Carry;

always@(Src1 or Src2 or Funct)
begin
    //  I use case to implement ALU, because ALU usually do a lot of jobs.
    case (Funct[5:0])
    // 6'b 001001: {Carry,Result} = Src1 + Src2;
    6'b 001010:
        begin
            {Carry,Result} = Src1 - Src2;
            if(Carry == 1)
                begin
                    Result = Src1;
                end
            else
                begin
                    Result = Result;
                end
        end
    default: // 如果今天Funct是別的東西就不用減，
        begin
            Carry = 0;
            Result = 0;
        end
    endcase
end

endmodule
```

```
module tb_ALU();       jiehong, 2 days ago • i think i finish the design of divider

reg [31:0]Src1;
reg [31:0]Src2;
reg [5:0]Funct;

wire [31:0]Result;
wire Carry;


ALU Arithemetic_Logical_Unit(
    .Src1(Src1),
    .Src2(Src2),
    .Funct(Funct),
    .Result(Result),
    .Carry(Carry)
);

initial #20 $finish;
initial fork
#0 Funct = 6'b 001010; // HW1 所規定的減法function code
#5 Src1 = 32'h 0; // 第一個測資們，我讓他們爆炸
#5 Src2 = 32'h FFFF_FFFF; // 因為是Src1 - Src2

#10 Funct = 6'b 001010;
#10 Src1 = 32'h 1800;
#10 Src2 = 32'h 200; // 第二個測資Result應該等於 1600h

#15 Funct = 6'b 0; // 如果funct不為MIPS所規定的function code，那Result與Carry就該為0。
#15 Src1 = 32'h 10;
#15 Src2 = 32'h 20;
join
endmodule
```



1. 在t=5，因為Src1 < Src2，因此輸出=Src1，且告訴下面的module我這個輸出是負的，所以carry=1

2. 1800-200=1600H，carry=0(結果為正)

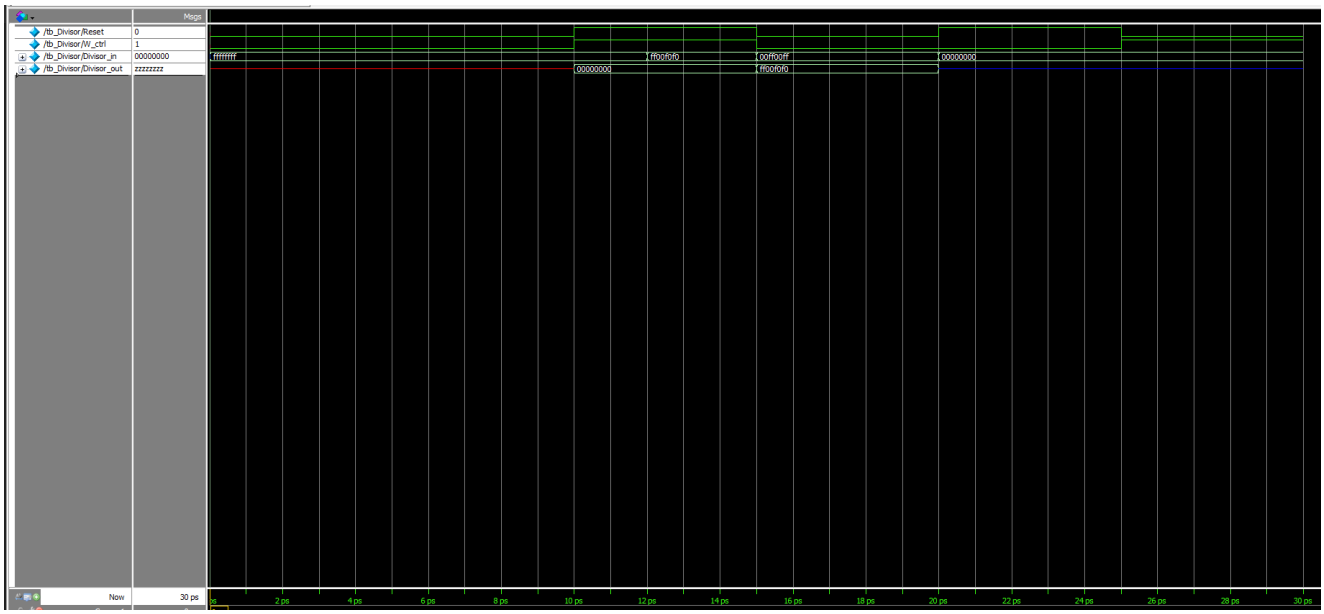
3. function = 0 ，ALU不工作。

b. Divisor

此部分根據w_ctrl來決定是否讀取除數，才讓數字進入ALU運算。但是除數有可能會在Reset訊號觸發之前進來，所以我有一個reg來儲存除數。

```
module Divisor(Reset,W_ctrl,Divisor_in,Divisor_out);      You, 3 days ago • commit from windows
input Reset;
input W_ctrl;
input [31:0]Divisor_in;
reg [31:0]Divisor_reg;
output reg [31:0]Divisor_out;

always@(Reset or W_ctrl or Divisor_in)
begin
    if(Reset == 1 && W_ctrl == 1)
        begin
            Divisor_reg = Divisor_in ;
            Divisor_out = 0; // 把值讀進來，但不輸出or輸出為0
            if(Divisor_in == 0)
            begin
                Divisor_reg = 32'bz;
            end
        end
    // 在設計沒有錯誤的情況下，W_ctrl應該會與Reset連動，所以我將這兩個logic AND 起來
    else if (W_ctrl == 0 && Reset == 0) // means that the system is running
        begin
            Divisor_out = Divisor_reg; // let output = reg
        end
    else
        begin
            Divisor_out = 32'bZ; // it means that the design is failed.
        end
end


endmodule
```

```
module tb_Divisor();      You, 2 days ago • finish the test from compDiv

reg Reset,W_ctrl;
reg [31:0]Divisor_in;
wire [31:0]Divisor_out;

Divisor divisor(
    .Reset(Reset),
    .W_ctrl(W_ctrl),
    .Divisor_in(Divisor_in),
    .Divisor_out(Divisor_out)
);

initial #30 $finish;
initial fork
#0 Reset = 0; // 因為W_ctrl是由Controller所發送，所以一定會與Reset相同，那在這裡
#0 W_ctrl = 0;// 我們就依照我們設計正確的前提下，去進行test bench的測試
#0 Divisor_in = 32'h FFFF_FFFF; // 兩者皆為0的時候改任何值，輸出都不應該改變

#10 Reset = 1;
#10 W_ctrl = 1;
#12 Divisor_in = 32'h FF00_F0F0;
// 與上面兩行時間不同，是因為輸入有可能不一樣，且這個輸出應該為0

#15 Reset = 0;
#15 W_ctrl = 0;
#15 Divisor_in = 32'h 00FF_00FF;

// 以下測試為，系統錯誤時會發生的狀況，也就是Reset與W_ctrl不同步。
#20 Reset = 1;
#20 W_ctrl = 0;
#20 Divisor_in = 32'h 0;

#25 Reset = 0;
#25 W_ctrl = 1;
#25 Divisor_in = 32'h 0;

join

endmodule
```



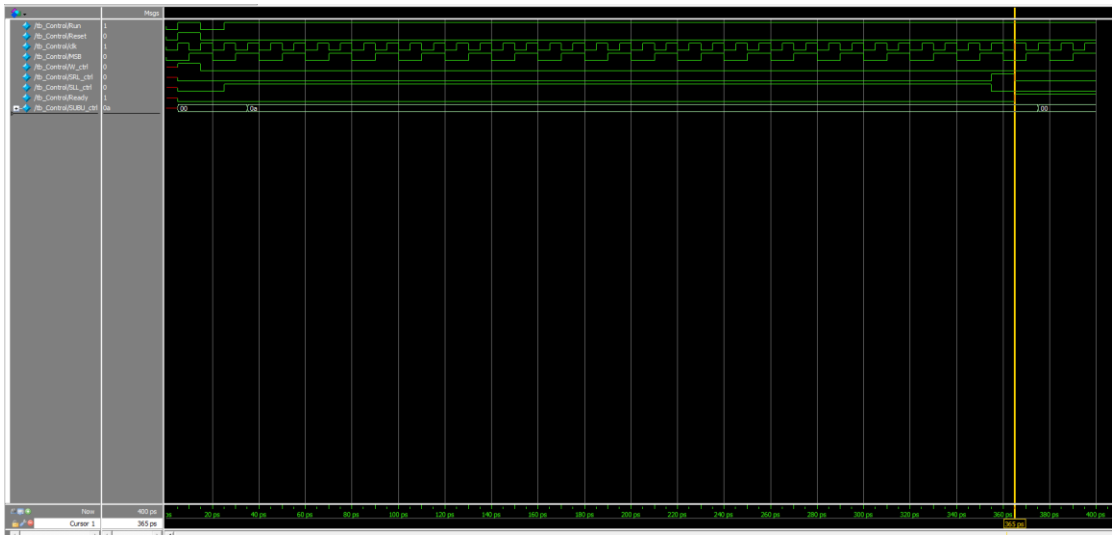
1. t=0 的時候輸出是 don’t care，因為系統剛開始沒有發送 reset=1 的初始化。
2. 在 t=10 的時候因為 reset=1，所以輸出為 0，進行整個系統的初始化。
3. T=15，輸出會吃之前的 t=12 的 divisor_in，因為 reset 訊號是在 t=10 觸發，但是在 t=12，輸入就被送進來了。
4. 因為 Reset 與 W_ctrl 不同步，所以輸出是高阻抗，如果有這種狀況代表我設計錯誤。

c. Control

```
module Control(Run,Reset,clk,W_ctrl,SUBU_ctrl,SRL_ctrl,SLL_ctrl,Ready,MSB);      You, 3 days ago • commit from windows
input Run;
input Reset;
input clk;
input MSB;

output reg W_ctrl; // Write control
output reg [5:0]SUBU_ctrl;
output reg SRL_ctrl;
output reg SLL_ctrl;
output reg Ready;
reg [5:0]counter; // counter for 32 times;
reg counting;
always@(posedge clk or  posedge Reset)
begin
    if(Reset == 1)
        begin
            W_ctrl = 1; // Write control → 是否要取新的值進來
            SUBU_ctrl = 6'b0; // Function → ALU要不要工作
            SRL_ctrl = 0; // Shift right control → 要不要shift
            SLL_ctrl = 0; // Shift left control
            Ready = 0; // Ready signal → 結果完成
            counter = 0;
            counting = 0;
        end
    else if (Run == 1 && Ready == 0)
        begin
            if(counter == 32) // for (counter = 0 ; counter < 32; counter ++)
                begin
                    if(SRL_ctrl == 0)
                        begin
                            SLL_ctrl = 0;
                            SRL_ctrl = 1;
                        end
                    else
                        begin
                            SRL_ctrl = 0;
                            Ready = 1;
                        end
                end
```

```
            else
                begin
                    if(counting == 0)
                        begin
                            SLL_ctrl = 1;
                            counting = 1;
                        end
                    else
                        begin // start to count.
                            W_ctrl = 0;    // 在Run的時候不要取新的值進來
                            SUBU_ctrl = 6'b 001010;
                            if(MSB == 0) // 代表Remainder ≥ 0
                                begin
                                    SLL_ctrl = 1; // 3.a → Remainder[0] = 1;
                                end
                            else // Remainder < 0
                                begin
                                    SLL_ctrl = 1; // 3.b → Remainder[0] = 0;
                                end
                            counter = counter + 1;
                        end
                end
        end
    else // Ready = 1;
        begin
            W_ctrl = 0 ; // Write control → 是否要取新的值進來 → 因為有可能Reset完，Run還沒跳起來
            SUBU_ctrl = 0; // Function → ALU要不要工作
            SRL_ctrl = 0 ; // Shift right control → 要不要shift
            SLL_ctrl = 0 ; // Shift left control
            Ready = Ready; // Ready signal → 結果完成
            counter = counter;
            counting = counting;
        end
end


endmodule
```

```
module tb_Control();      jiehong, 2 days ago • i think i finish the design of divider

reg Run,Reset,clk,MSB;
wire W_ctrl,SRL_ctrl,SLL_ctrl,Ready;
wire [5:0]SUBU_ctrl;

Control controller(
    .Run(Run),
    .Reset(Reset),
    .clk(clk),
    .MSB(MSB),
    .W_ctrl(W_ctrl),
    .SUBU_ctrl(SUBU_ctrl),
    .SRL_ctrl(SRL_ctrl),
    .SLL_ctrl(SLL_ctrl),
    .Ready(Ready)
);

initial #400 $finish;

initial begin
    #0 clk = 0;
    forever #5 clk = ~clk;
end

initial begin
    #0 MSB = 0;
    forever #10 MSB = ~MSB;
end


initial fork
    #0 Run = 0;
    #0 Reset = 0;

    #5 Reset = 1;    // 以reset為主，所以下面為多少理論上都不會影響
    #5 Run = 1;

    #15 Reset = 0;
    #15 Run = 0;

    #25 Run = 1;     //從這裡開始計數32次 → 10*32=320，320+20 = 340(10秒為放shift left，另一個10秒為最後做完32次要在right shift)，340 + 25 = 365 在355的時候，
join

endmodule
```



1. T=5的時候，Reset觸發，系統初始化
2. T=15的時候，Reset訊號已結束，開始等待Run訊號跳起
3. T=25的時候，Run=1，系統開始執行。

d. Remainder

這裡跟Product相同，我利用clk的負緣觸發來使整個系統可以在一個clk內做完一次，而以下分為四種部份。

1.Reset=1，系統初始化，把被除數讀進來。

2.在確認Ready=0之下，SLL_ctrl=1，如果counting=0，代表程式還未計數，這時候代表流程圖中的Shift Remainder register left 1 bit，做完後進入計數的部分，因此令counting=1。Counting=1後開始計數32次，由ALU_carry來判斷Remainder的正負。

3.line 50 因為做完32次還不代表真正結束，這時候要右移Remainder的左半部1bit.

4.Ready跳起後，令輸出維持不變。

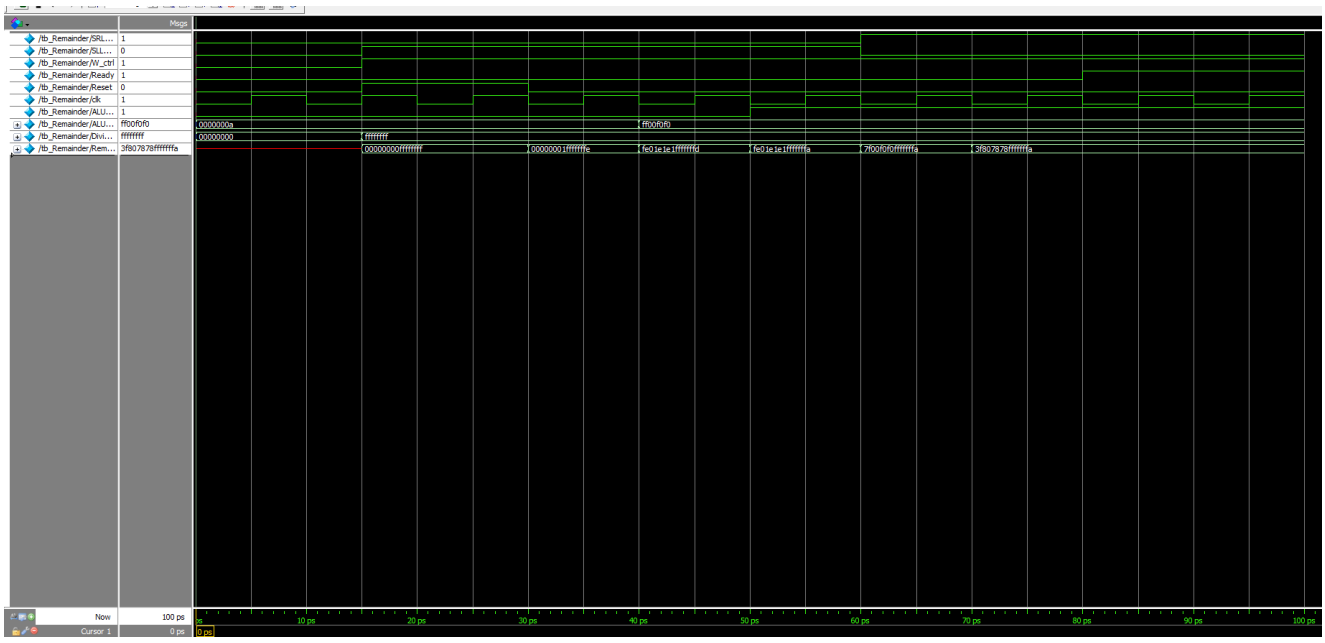
```
1  module Remainder(SRL_ctrl,SLL_ctrl,W_ctrl,Ready,Reset,clk,ALU_carry,ALU_result,Dividend_in,Remainder_out);
2
3  input SRL_ctrl,SLL_ctrl;
4  input W_ctrl;
5  input Ready;
6  input Reset;
7  input clk;
8  input ALU_carry;
9  input [31:0]ALU_result;
10 input [31:0]Dividend_in;
11 output reg[63:0]Remainder_out;
12 reg counting; // start to count or not.
```

```
13 always@(negedge clk or posedge Reset)
14     begin
15         if (Reset == 1 && W_ctrl == 1)
16             begin
17                 Remainder_out = 64'b0;
18                 Remainder_out[31:0] = Dividend_in[31:0];
19                 counting = 0;
20             end
21         else if(SLL_ctrl == 1 && Ready == 0)
22             begin
23                 if(counting == 0)
24                     begin
25                         if(SLL_ctrl == 1)
26                             begin
27                                 Remainder_out = Remainder_out << 1;
28                                 counting = 1;
29                             end
30                     end
31                 else // start to count 32 times
32                     begin
33                         Remainder_out[63:32] = ALU_result[31:0];
34                         /*
35                             Subtract Divisor register from the
36                             left half of Remainder register, and place the
37                             result in the left half of Remainder register
38                         */
39                         Remainder_out = Remainder_out << 1;
40                         if(ALU_carry == 0) // means that Remainder ≥ 0;
41                             begin
42                                 Remainder_out[0] = 1;
43                             end
44                         else
45                             begin
46                                 Remainder_out[0] = 0;
47                             end
48                     end
49             end
```

```
        else if (SRL_ctrl == 1 && Ready == 0)
            begin
                Remainder_out[63:32] = Remainder_out[63:32] >> 1;
            end
        else if(Ready == 1)
            begin
                Remainder_out = Remainder_out;
            end
        else;
    end
endmodule
```

```
module tb_Remainder();
reg SRL_ctrl;
reg SLL_ctrl;
reg W_ctrl;
reg Ready;
reg Reset;
reg clk;
reg ALU_carry;
reg [31:0]ALU_result;
reg [31:0]Dividend_in;
wire [63:0]Remainder_out;

Remainder remain
(
    .SRL_ctrl(SRL_ctrl),
    .SLL_ctrl(SLL_ctrl),
    .W_ctrl(W_ctrl),
    .Ready(Ready),
    .Reset(Reset),
    .clk(clk),
    .ALU_carry(ALU_carry),
    .ALU_result(ALU_result),
    .Dividend_in(Dividend_in),
    .Remainder_out(Remainder_out)
);
```

```
initial #100 $finish;
initial begin
#0 clk = 0;
forever #5 clk = ~clk;
end

initial fork
    // 初始化
    #0 ALU_result = 32'd10;
    #0 W_ctrl = 0;
    #0 Reset = 0;
    #0 Ready = 0;
    #0 SRL_ctrl = 0;
    #0 SLL_ctrl = 0;
    #0 ALU_carry = 0;
    #0 Dividend_in = 0;

    // 系統開始初始化，remainder_out[31:0] = Dividend_in [31:0] << 1
    #15 Reset = 1;
    #15 W_ctrl =1;
    #15 Dividend_in = 32'h FFFF_FFFF;
    #15 SLL_ctrl = 1; // start to run

    // 系統開始執行
    #30 Reset = 0;
    #30 SRL_ctrl = 0;
    #30 SLL_ctrl = 1; // Remainder_out = Remainder_out << 1;
    #40 ALU_result = 32'h FF00_F0F0; // 此時的remainder out 會等於 ALU_Result << 1
    #40 ALU_carry = 0; // remainder_out[0] = 1;

    #50 ALU_carry = 1; // remainder_out[0] = 0;

    #60 SRL_ctrl = 1; // Remainder_out[63:32] = Remainder_out[63:32] >> 1;
    #60 SLL_ctrl = 0;
    #80 Ready = 1; // 讓輸出維持不變

join

endmodule
```



1. T=15系統開始初始化
2. 直到T=30，Reset訊號不為0的時候才可以開始工作。
3. T=40的時候，因為ALU_carry=0，這時候Remainder_out[0]=1(d=1101)
4. T=50，ALU_carry=1，這時候Remainder_out[0]=0(a=1010)
5. T=80，Ready=1，之後輸出維持不變

e. CompDivider

而最後，關於CompDivider這個模組，我做的則是去把Control、ALU、Divisor、Remainder這幾個模組組合起來，再藉由wire把他們從內部連接起來。Testbench直接使用助教所提供的檔案，下圖為由測試指令之in檔截圖及testbench產生之模擬結果。

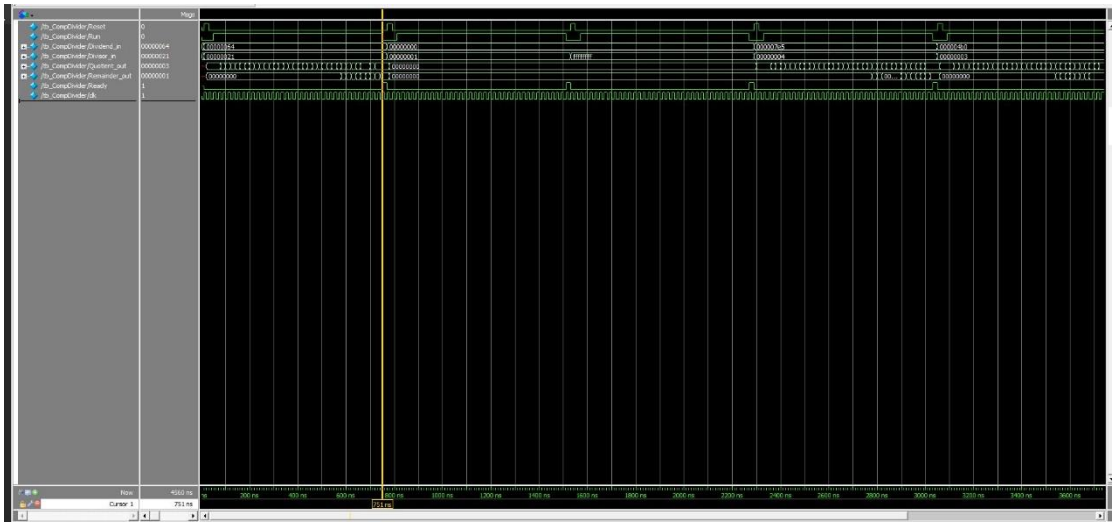
```
module CompDivider(clk,Reset,Run,Divisor_in,Dividend_in,Remainder_out,Quotient_out,Ready,ALU_result);
input clk;
input Reset;
input Run;
input [31:0] Divisor_in;
input [31:0] Dividend_in;
wire [63:0]Remainder_out_reg;
output [31:0]Remainder_out;
output [31:0]Quotient_out;
output Ready;
// Divisor
wire W_ctrl;
wire [31:0]Divisor_out;
// for Remainder
wire SRL_ctrl,Ready,Reset,ALU_carry;
output [31:0]ALU_result;
wire [5:0]SUBU_ctrl;
```

```
// control unit
Control controller
(
    .Run(Run),
    .Reset(Reset),
    .clk(clk),
    .MSB(Remainder_out_reg[63]),
    .W_ctrl(W_ctrl),
    .SUBU_ctrl(SUBU_ctrl[5:0]),
    .SRL_ctrl(SRL_ctrl),
    .SLL_ctrl(SLL_ctrl),
    .Ready(Ready)
);
```

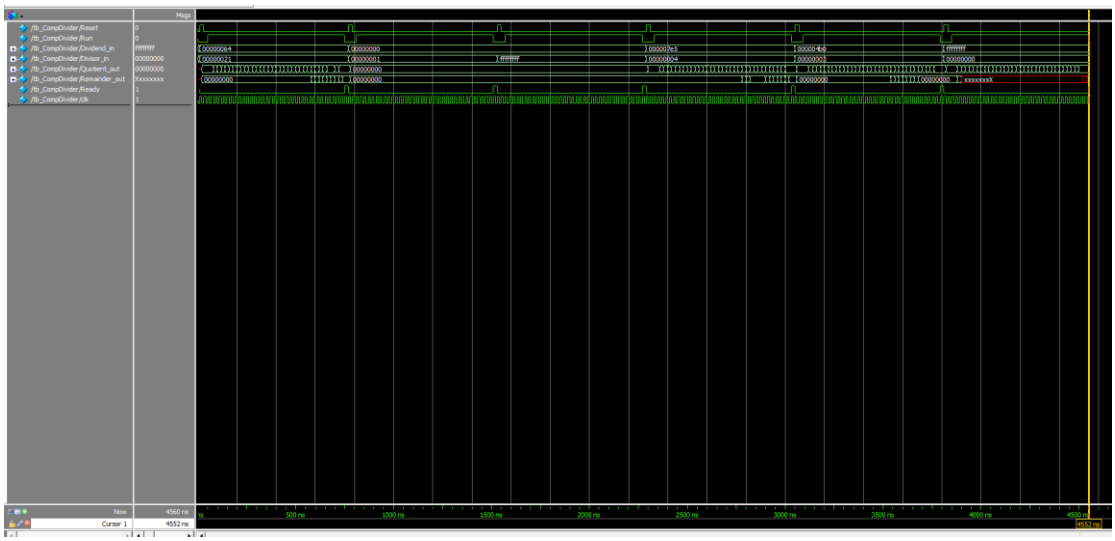
```
// 被除數
Divisor divisor
(
    .Reset(Reset),
    .W_ctrl(W_ctrl),
    .Divisor_in(Divisor_in[31:0]),
    .Divisor_out(Divisor_out[31:0])
);      You, a week ago • tb of multiplier almost finis

// ALU
ALU Arithemetic_Logical_Unit
(
    .Src1(Remainder_out_reg[63:32]),
    .Src2(Divisor_out[31:0]),
    .Funct(SUBU_ctrl[5:0]),
    .Carry(ALU_carry),
    .Result(ALU_result[31:0])
);

// remain unit
Remainder remain_unit
(
    .SRL_ctrl(SRL_ctrl),
    .SLL_ctrl(SLL_ctrl),
    .W_ctrl(W_ctrl),
    .Ready(Ready),
    .Reset(Reset),
    .clk(clk),
    .ALU_carry(ALU_carry),
    .ALU_result(ALU_result[31:0]),
    .Dividend_in(Dividend_in[31:0]),
    .Remainder_out(Remainder_out_reg[63:0])
);

assign Remainder_out[31:0] = Remainder_out_reg[63:32];
assign Quotient_out[31:0] = Remainder_out_reg[31:0];

endmodule
```

tb_CompDivider.in

```
00000064_00000021
00000000_00000001
00000000_FFFFFFFF
000007E5_00000004
000004B0_00000003
FFFFFFFF_00000000
```



在第一次被除數輸入100，除數輸入33，Remainder經過1個clk的shift再經過32次的運算，再經過1個clk的shift得到最終結果商=3，餘數=1，後面運算也都照我預期執行。而比較特別的是大約t=4500左右，我去測試任意數除上0，會有一些奇怪的結果，因此我有特別去做令他輸出為don’t care的結果。



作業總結與心得

這次的作業，雖然花了非常多的時間，大約整整兩個禮拜加上連假都在處理這次的作業，但是與之前數位系統設計不同的是，這堂課真的能讓我們運用之前所學，嘗試去整個系統，真的是與課堂名稱相呼應，我在設計每個模組的時候，都要非常清楚，這個模組是在做甚麼，及訊號要怎麼送進來，怎麼送出去。雖然很累，但我真的覺得非常值得!

還有在撰寫的時候，真的是更加驚艷，原來Controller是如此的重要，難怪在上學期修課時，無論課本抑或是老師都那麼強調他的重要性，在上學期寫作業的時候我常常忽略Controller，直接利用always暴力完成，現在發現只有小作業可以這樣子搞，遇到真正的大系統就只能藉由controller來完成了。

最後我覺得最重要的是，比起急著開始打程式，還不如先想好要怎麼設計，真的確定好邏輯是對的，都比一開始就急著打但毫無頭緒還快，也讓我了解到設計前規劃的重要，希望以後這堂課的不論是考試或是project，也可以像這次一樣順利度過。