計算機組織　作業報告

作業一(PA1)

B10807005 朱育辰

一、乘法器

a. 各模組功能描述

i. Multiplicand

1. 程式碼

```
module Multiplicand (
    input [31:0] Multiplicand_in,
    input reset,w_ctrl,
    output reg [31:0] Multiplicand_out
);
    reg [31:0] multiplicand_reg;
    always @(reset or w_ctrl or Multiplicand_in) begin
        if (reset == 1) begin
            Multiplicand_out[31:0] = 0;
            multiplicand_reg[31:0] = Multiplicand_in[31:0];
        end
        else if (reset == 0 && w_ctrl == 1) begin
            Multiplicand_out[31:0] =  multiplicand_reg[31:0];
        end
    end
endmodule

```

2. 功能

在接收到重設訊號時，將輸出歸零並將被乘數輸入放進暫存器，在歸零訊號結束後，若接收到寫入控制訊號則將暫存器輸出。

ii. ALU

1. 程式碼

```
module ALU (
    input [31:0] Src1,Src2,
    input [5:0] Funct,
    output reg [31:0] ALU_result,
    output reg ALU_Carry
);
    always @(Funct or Src1 or Src2) begin
        if (Funct == 6'b000001) begin
            {ALU_Carry,ALU_result} = Src1 + Src2;
        end
        else begin
            ALU_Carry = 0;
            ALU_result = 0;
        end
    end
endmodule

```

2. 功能

根據收到的指令，決定輸出，若輸入為正確的相加指令，則從進位腳與結果匯流排輸出兩輸入資料相加的結果，否則將兩輸出設為零。

iii. Control

1. 程式碼

```
module Control (
    input reset,run,clk,lsb,
    output reg w_ctrl,srl_ctrl,ready,
    output reg [5:0] addu_ctrl
);
    reg [5:0] counter;

    assign addu_ctrl = {5'b00000,lsb};
    always @(posedge clk or posedge reset) begin
        if(reset == 1) begin
            w_ctrl = 1;
            counter = 0;
            srl_ctrl = 0;
            addu_ctrl = 6'b0;
            ready = 0;
        end
        else if (run == 1 && ready == 0) begin
            if (counter < 32) begin
                w_ctrl = 0;
                srl_ctrl = 1;
                // if (lsb == 1) begin
                //     addu_ctrl = 6'b000001;
                // end
                // else begin
                //     addu_ctrl = 6'b000000;
                // end
                counter = counter + 1;
            end
            else if (counter == 32)begin
                ready = 1;
                counter = 0;
                srl_ctrl = 0;
            end
        end

    end
endmodule

```

2. 功能

若接收到重設訊號，則將計數器、位移控制、完成旗標與相加指令歸零，並設立寫入控制準備接收輸入。接收到運作訊號後依照乘法規則開始執行工作並移除寫入控制旗標後設立向右位移旗標，根據乘積最小位元決定加法指令，做完 32 次即為完成運算，設立完成旗標並歸零計數器與位移控制旗標。

iv. Product

1. 程式碼

```
module Product (
    input [31:0] ALU_result,
    input [31:0] Multiplier_in,
    input ALU_Carry,srl_ctrl,w_ctrl,ready,reset,clk,
    output reg[63:0] Product_out,
    output reg lsb
);
    reg[63:0] Product_reg;

    always @(posedge clk or posedge reset) begin
        if (reset == 1) begin
            Product_out = 64'b0;
            Product_reg[31:0] = Multiplier_in[31:0];
        end
        else if (w_ctrl == 1) begin
            Product_out[31:0] = Product_reg[31:0];
        end
        else if (ready == 0 && srl_ctrl == 1) begin
            if (Product_out[0] == 1) begin
                Product_out[63:32] = ALU_result[31:0];
            end
            Product_out = {ALU_Carry,Product_out[63:1]};
            if (Product_out[0] == 1) begin
                lsb = 1;
            end
            else begin
                lsb = 0;
            end
        end

    end
endmodule

```

2. 功能

接收到歸零與寫入控制訊號的動作與 Multiplicand 相同，清空輸出並等待時機輸出新值。開始運算後，根據位移控制訊號與乘積最後一位元的值，決定乘積是否右移或修改數值。

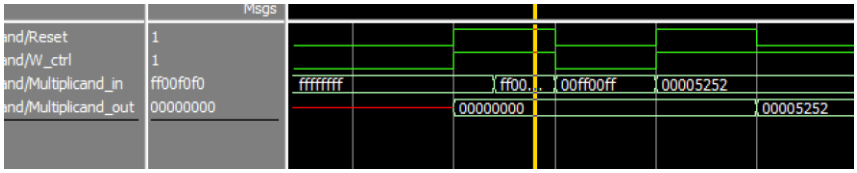
b. 測試資料與結果

i. Multiplicand

1. 測試資料

```
module tb_Multiplicand();

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_i
    .Multiplicand_out(Multiplicand_
);

initial #30 $finish;
initial fork
#0 Reset = 0;
#0 W_ctrl = 0;
#0 Multiplicand_in = 32'h FFFF_FFFF

#10 Reset = 1;
#10 W_ctrl = 1;
#12 Multiplicand_in = 32'h FF00_F0F


#15 Reset = 0;
#15 W_ctrl = 0;
#15 Multiplicand_in = 32'h 00FF_00F

#20 Reset = 1;
#20 W_ctrl = 1;
#20 Multiplicand_in = 32'h 00005252

#25 Reset = 0;

join

endmodule
```

2. 測試結果



3. 分析

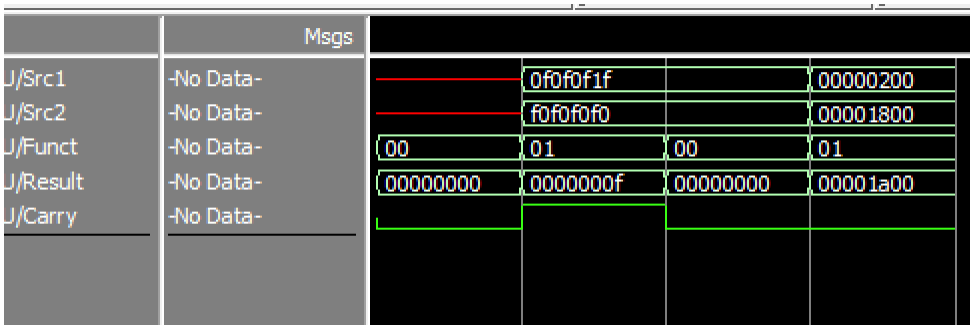
可以看見接收到重設旗標後輸出部分保持為 0，待重設結束後，根據寫入控制旗標將數值輸出。

ii. ALU

1. 測試資料

```
1   module tb_ALU();
2
3   reg [31:0]Src1;
4   reg [31:0]Src2;
5   reg [5:0]Funct;
6
7   wire [31:0]Result;
8   wire Carry;
9
10
11  ALU Arithemetic_Logical_Unit(
12      .Src1(Src1),
13      .Src2(Src2),
14      .Funct(Funct),
15      .ALU_result(Result),
16      .ALU_Carry(Carry)
17  );
18  initial #20 $finish;
19  initial fork
20  #0 Funct = 6'b 0;
21
22  #5 Funct = 6'b 000001;
23  #5 Src1 = 32'h 0F0F_0F1F;
24  #5 Src2 = 32'h F0F0_F0F0;
25
26  #10 Funct = 6'b 0;
27
28  #15 Funct = 6'b 000001;
29  #15 Src1 = 32'h 200;
30  #15 Src2 = 32'h 1800;
31
32  join
33  endmodule
```

2. 測試結果



3. 分析

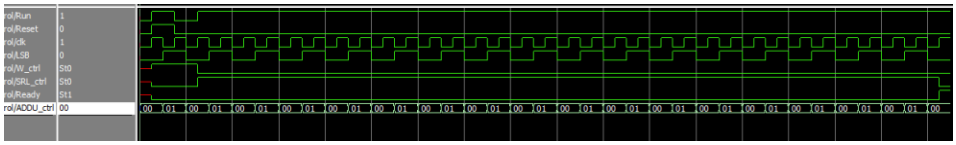
接收到正確運算指令後，將兩數值相加後輸出，否則輸出一律為 0。

iii. Control

1. 測試資料

```
module tb_Control();

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;
    #5 Run = 1;

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

    #25 Run = 1;
join

endmodule
```

2. 測試結果



3. 分析

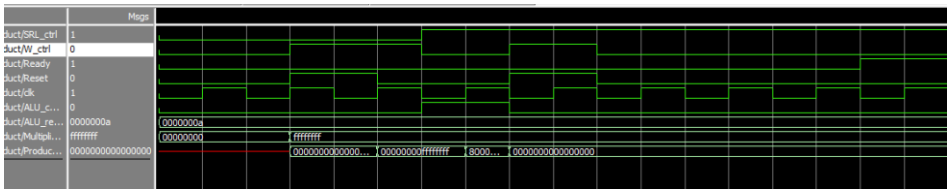
可以看到在收到重設訊號後，無論輸入任何訊號，都不會影響重設。在收到執行訊號後，開始計數 32 次後設立完成旗標。

iv. Product

1. 測試資料

```
module tb_Product();
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

    #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;


    #15 Reset = 1;
    #15 W_ctrl =1;
    #15 Multiplier_in = 32
'h FFFF_FFFF;


    #25 Reset = 0;
    #30 W_ctrl = 0;
    #30 SRL_ctrl = 1;
    #30 ALU_carry = 1;

    #40 ALU_carry = 0;
    #40 Reset = 1;
    #40 W_ctrl = 1;

    #50 Reset = 0;
    #50 W_ctrl = 0;
    #50 Ready = 0;

    #80 Ready = 1;

join

endmodule
```

2. 測試結果



3. 分析

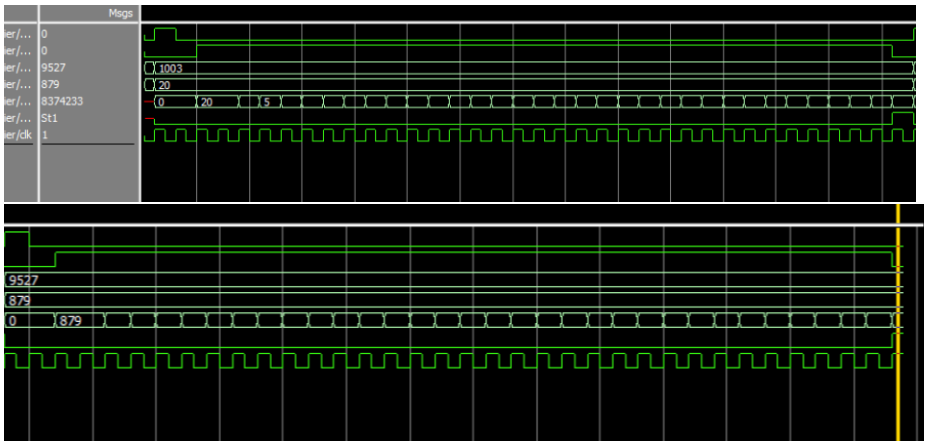
在收到重設訊號後開始重設流程，輸出為零，等待接收寫入控制訊號後輸出，接收到位移訊號也會隨之位移，接收到完成旗標後也會隨之停止。

v. CompMultiplier

1. 測試資料

```
000003EB_00000014
00002537_0000036F
```

2. 測試結果



3. 分析

此模組為整合以上所有子模組的完整乘法器，可以觀察到，不論是作業內提供的側姿或是自行輸入的數值，經過運算後都能正確輸出。

二、除法器

a. 各模組功能描述

i. Divisor

1. 程式碼

```
module Divisor (
    input [31:0] Divisor_in,
    input reset,w_ctrl,
    output reg [31:0] Divisor_out
);
    reg [31:0] Divisor_reg;
    always @(reset or w_ctrl or Divisor_in) begin
        if (reset == 1) begin
            Divisor_out[31:0] = 0;
            Divisor_reg[31:0] = Divisor_in[31:0];
        end
        if (reset == 0 && w_ctrl == 1) begin
            Divisor_out[31:0] =  Divisor_reg[31:0];
        end
    end
endmodule
```

2. 功能

與乘法器的 Multiplicand 相同，根據重設訊號與寫入控制動作。

ii. ALU

1. 程式碼

```
module ALU (
    input [31:0] Src1,Src2,
    input [5:0] Funct,
    output reg [31:0] ALU_result,
    output reg ALU_Carry
);
    always @(Funct or Src1 or Src2) begin
        if (Funct == 6'b000010) begin
            {ALU_Carry,ALU_result} = Src1 - Src2;
            if (ALU_Carry == 1) begin
                ALU_result = Src1;
            end
        end

        else begin
            ALU_Carry = 0;
            ALU_result = 0;
        end
    end
endmodule
```

2. 功能

與乘法器的 ALU 基本，接收指令並執行減法，若結果為負，則取消執行。

iii. Control

1. 程式碼

```
module Control (
    input reset,run,clk,
    output reg w_ctrl,srl_ctrl,ready,sll_ctrl,
    output reg [5:0] subu_ctrl
);
    reg [5:0] counter;
    always @(posedge clk or posedge reset) begin
        if(reset == 1) begin
            w_ctrl = 1;
            counter = 0;
            srl_ctrl = 0;
            sll_ctrl = 0;
            subu_ctrl = 6'b0;
            ready = 0;
        end
        else if (run == 1 && ready == 0) begin
            if (counter < 32) begin
                w_ctrl = 0;
                sll_ctrl = 1;
                subu_ctrl = 6'b000010;
                counter = counter + 1;
            end
            else if (counter == 32)begin
                sll_ctrl = 0;
                srl_ctrl = 1;
                counter = counter +1;
            end
            else begin
                ready = 1;
                srl_ctrl = 0;
            end
        end

    end
endmodule
```

2. 功能

同樣根據重設訊號與執行訊號作動，不同的是因應除法規則，運算開始後減法指令常設為正確指令，且計數次數為 33 次，最後一次為乘法規則中的右移。

iv. Remainder

1. 程式碼

```
module Remainder (
    input [31:0] ALU_result,
    input [31:0] Dividend_in,
    input ALU_Carry,srl_ctrl,sll_ctrl,w_ctrl,ready,reset,clk,
    output reg[63:0] Remainder_out
);

    always @(posedge clk or posedge reset) begin
        if (reset == 1) begin
            Remainder_out = 64'b0;
        end
        else if (w_ctrl == 1) begin
            Remainder_out[63:0] = {31'd0,Dividend_in[31:0],1'b0};
        end
        else if (ready == 0 && sll_ctrl == 1) begin
            Remainder_out = {ALU_result[30:0],Remainder_out[31:0],~ALU_Carry};
        end
        else if (ready == 0 && srl_ctrl == 1) begin
            Remainder_out = {1'b0,Remainder_out[63:33],Remainder_out[31:0]};
        end
    end
endmodule
```

2. 功能

   同樣根據控制模組的輸出與重設訊號作動，不同的是因為設計不同，因此可以少很多判斷，看起來更為精簡

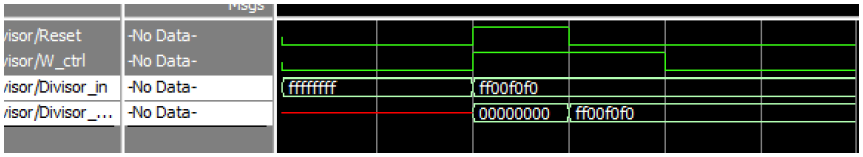
b. 測試資料與結果

i. Divisor

1. 測試資料

```
module tb_Control();

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),
    .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 fork
    #0 Run = 0;
    #0 Reset = 0;

    #5 Reset = 1;
    #5 Run = 1;

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

    #25 Run = 1;
join

endmodule
```

2. 測試結果



3. 分析

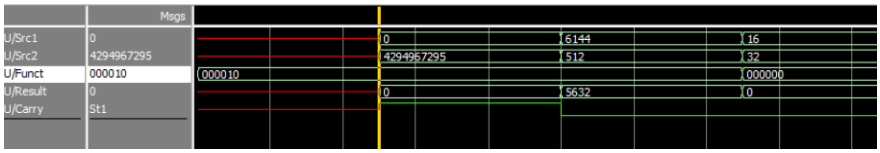
接收到重設訊號後輸出為零，後接收到寫入控制，正常輸出。

ii. ALU

1. 測試資料

```
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),
    .ALU_result(Result),
    .ALU_Carry(Carry)
);

initial #20 $finish;
initial fork
#0 Funct = 6'b 000010;
#5 Src1 = 32'h 0;
#5 Src2 = 32'h FFFF_FFFF;

#10 Funct = 6'b 000010;
#10 Src1 = 32'h 1800;
#10 Src2 = 32'h 200;

#15 Funct = 6'b 0;
#15 Src1 = 32'h 10;
#15 Src2 = 32'h 20;
join
endmodule
```

2. 測試結果



3. 分析

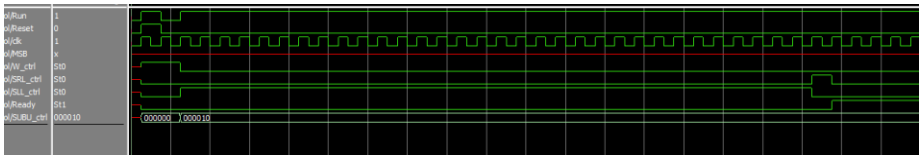
根據傳入的指令進行運算，若指令正確則進行運算，結果為負時結果為輸入資料一，進位照運算結果，若指令錯誤則皆為零。

iii. Control

1. 測試資料

```
1   module tb_Control();
2
3   reg Run,Reset,clk,MSB;
4   wire W_ctrl,SRL_ctrl,SLL_ctrl,Ready;
5   wire [5:0]SUBU_ctrl;
6
7   Control controller(
8       .run(Run),
9       .reset(Reset),
10      .clk(clk),
11      .w_ctrl(W_ctrl),
12      .subu_ctrl(SUBU_ctrl),
13      .srl_ctrl(SRL_ctrl),
14      .sll_ctrl(SLL_ctrl),
15      .ready(Ready)
16  );
17
18  initial #400 $finish;
19
20  initial begin
21      #0 clk = 0;
22      forever #5 clk = ~clk;
23  end
24
25
26  initial fork
27      #0 Run = 0;
28      #0 Reset = 0;
29
30      #5 Reset = 1;
31      #5 Run = 1;
32
33      #15 Reset = 0;
34      #15 Run = 0;
35
36      #25 Run = 1;
37  join
38
39  endmodule
```

2. 測試結果



3. 分析

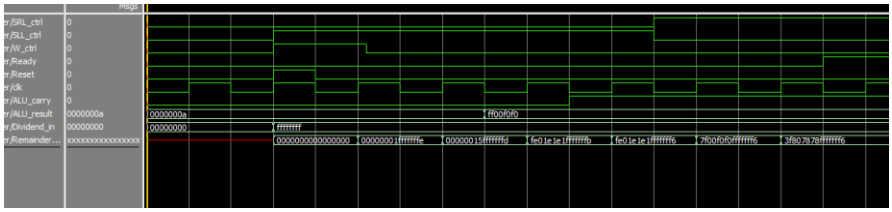
接收到重設訊號後開始重設流程，計數 32 次後右移控制旗標設立，33 次時完成旗標設立。

iv. Remainder

1. 測試資料

```
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;

    #15 Reset = 1;
    #15 W_ctrl =1;
    #15 Dividend_in = 32'h FFFF_FFFF;
    #15 SLL_ctrl = 1;


    #20 Reset = 0;
    #26 W_ctrl = 0;
    #30 SRL_ctrl = 0;
    #35 SLL_ctrl = 1;
    #40 ALU_result = 32'h FF00_F0F0;
    #40 ALU_carry = 0;

    #50 ALU_carry = 1;

    #60 SRL_ctrl = 1;
    #60 SLL_ctrl = 0;
    #80 Ready = 1;

join



endmodule
```

2. 測試結果



3. 分析

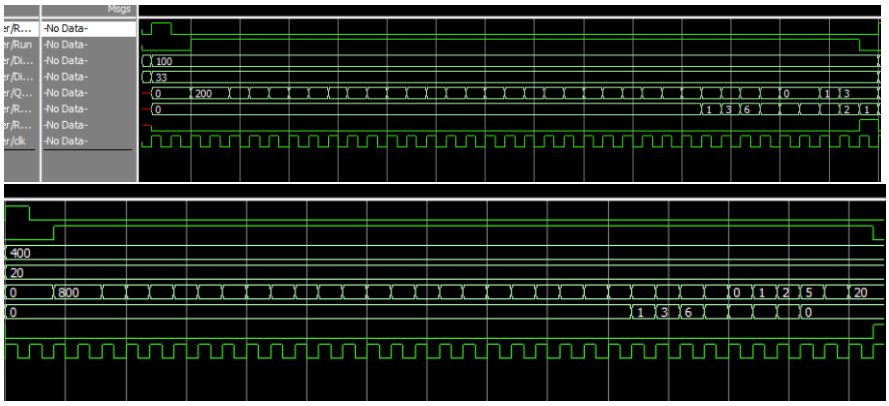
根據重設訊號與控制模組的輸出訊號作動，接收到重設與寫入控制訊號後開始重設，接收到左右移控制訊號後正常左右移。

v. CompDivider

1. 測試資料

```
1    00000064_00000021
2    00000190_00000014
```

2. 測試結果



3. 分析

可以看到不論是否整除，皆可正確計算出結果。

三、心得

這次是第一次的大作業，相較之下，上個作業基本上可以說是小打小鬧。也遇到了不少問題，與同學相互討論後成功解決了一部分，另一部分則是參考網路上他人的範例，刪除不合宜的地方並修改成可以運作的版本。深刻可以體會到老師所說這門課作業很重的意涵。