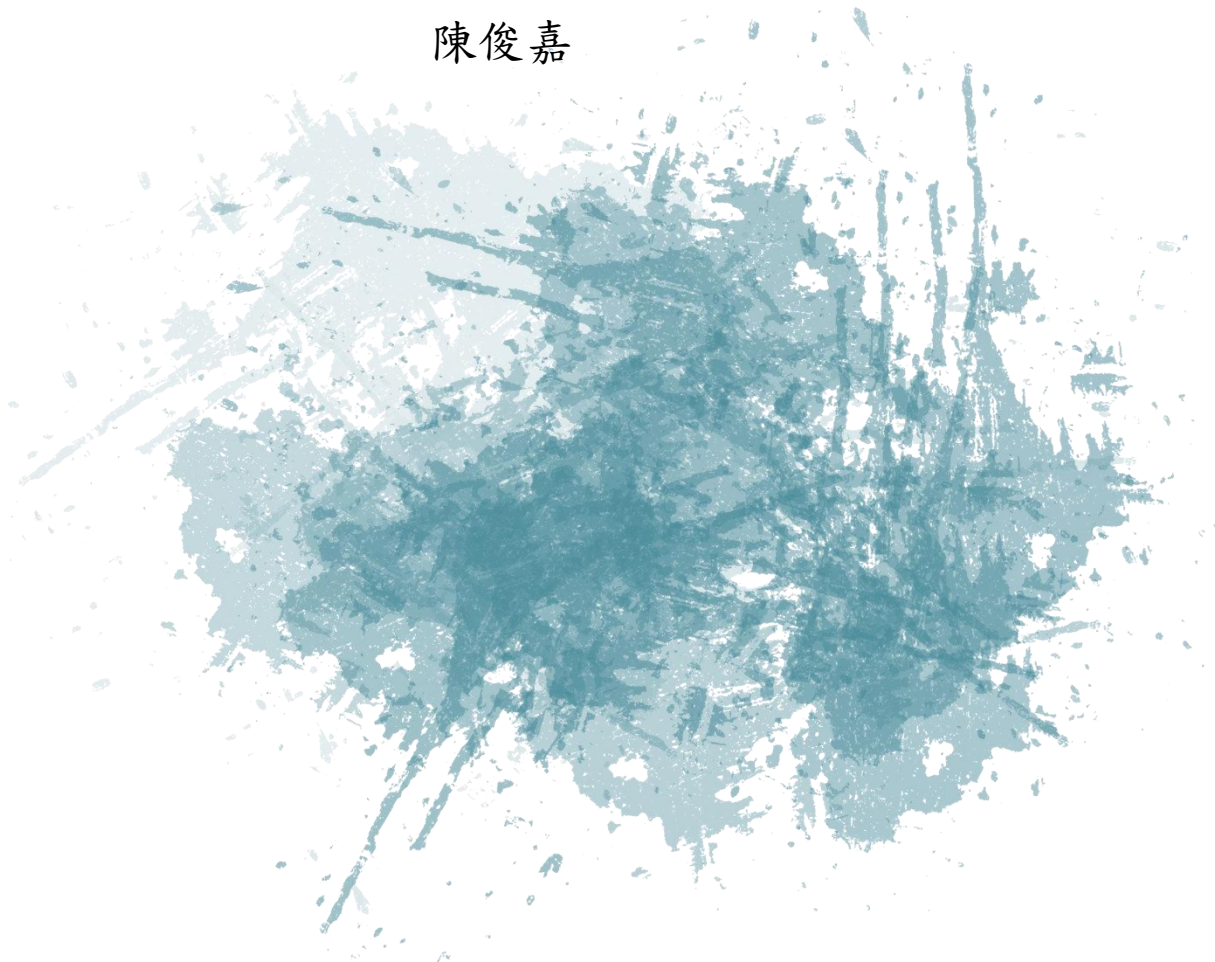


Computer Organization Project 1

四電機三甲

B10830009

陳俊嘉



Part I: Implement A 32-bit Unsigned Complete Multiplier

- Control Module

```
1  `define add      6'b001001
2
3  module Control(run, rst, clk, LSB, w_ctrl, SRL_ctrl, ready, ADDU_ctrl);
4  //inputs
5  input run, rst, clk, LSB;
6
7  //outputs
8  output reg w_ctrl, SRL_ctrl, ready;
9  output reg [5:0]ADDU_ctrl;
10
11  integer counter;
12
13  always@(posedge clk)begin
14      //reset
15      if(rst)begin
16          w_ctrl <= 0;
17          SRL_ctrl <=0;
18          ready <= 0;
19          ADDU_ctrl <= 0;
20          counter <= 0;
21      end else begin
22          //if the multiplication start -> product start shifting
23          if(run)begin
24              SRL_ctrl <= 1;
25          end else begin
26              SRL_ctrl <= 0;
27          end
28
29          //if LSB == 1 -> give add instruction, get ALU result
30          if(LSB)begin
31              w_ctrl <= 1;
32              ADDU_ctrl <= `add;
33          end
34          //if LSB == 0 -> no instruction, can't get ALU result
35          else begin
36              w_ctrl <= 0;
37              ADDU_ctrl <= 0;
38          end
39
40          //check if the loop is over
41          if(counter == 33)begin
42              ready <= 1;
43          end else begin
44              ready <= 0;
45              counter <= counter + 1;
46          end
47      end
48  end
49  endmodule
```

- Control test bench

```

1
2 // Setting timescale
3 `timescale 10 ns / 1 ns
4
5 `define DELAY 1 // # * timescale
6
7 // Declarations
8 `define LOW 1'b0
9 `define HIGH 1'b1
10
11 module tb_Control;
12
13 //inputs
14 reg run = `LOW;
15 reg reset = `LOW;
16 reg LSB = 0;
17
18 //outputs
19
20 wire w_ctrl, ready;
21 wire [5:0]ADDU_ctrl;
22
23 // Clock
24 reg clk = `LOW;
25
26 Control UUT(
27 //inputs
28 .run(run),
29 .rst(reset),
30 .clk(clk),
31 .LSB(LSB),
32 //outputs
33 .w_ctrl(w_ctrl),
34 .SRL_ctrl(SRL_ctrl),
35 .ready(ready),
36 .ADDU_ctrl(ADDU_ctrl)
37 );
38
39 // Generate Clock
40 always
41 begin : ClockGenerator
42 #1 clk <= ~clk; // toggle clock signal every one timescale delay
43 end
44
45 initial begin
46 // Wait positive edge of clock signal
47 @(posedge clk);
48
49 // Reset UUT
50 reset <= `HIGH;
51 run <= `LOW;
52
53 // End reset
54 @(posedge clk);
55 reset = `LOW;
56 //set run to 1
57 @(posedge clk);
58 run <= `HIGH;
59 // Wait 10ns and assign LSB -> 1
60 #10;
61 LSB <= 1;
62 // Wait 10ns and assign LSB -> 0
63 #10;
64 LSB <= 0;

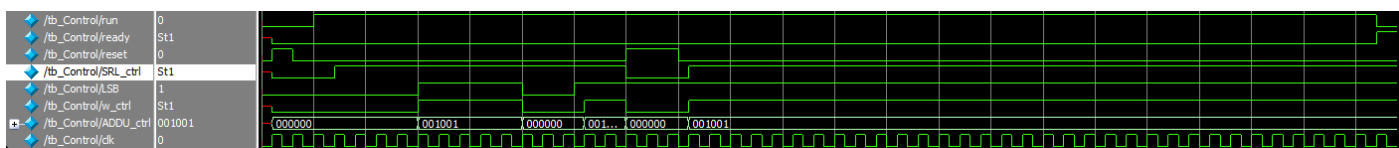
```

```

65         // Wait 10ns and assign LSB -> 1 and wait 5ns to reset
66         #5;
67         LSB <= 1;
68         #5;
69         reset <= `HIGH;
70         #5;
71         reset <= `LOW;
72         // Wait 10ns and assign LSB -> 1
73         #10;
74         LSB <= 1;
75         //Wait for multiplication iteration end
76         @(posedge ready);
77         run <= `LOW;
78
79         // Wait some time
80         #2;
81
82         // Stop the simulation
83         $stop();
84     end
85
86 endmodule

```

• Control Module Simulation



- ◆ W_ctrl、SRL_ctrl、ADDU_ctrl的postive edge trigger 正常
- ◆ Reset功能正常
- ◆ Run、ready 正常rise、fall

• Multiplicand Module

```

1  module Multiplicand(Multiplicand_in, rst, w_ctrl, Multiplicand_out);
2      input rst, w_ctrl;
3      input [31:0]Multiplicand_in;
4      output reg [31:0]Multiplicand_out;
5
6      always@(*)begin                                //any change of input will action
7          if(rst)begin                                //reset the output to zero
8              Multiplicand_out <= 0;
9          end else if(w_ctrl)begin                    //if write control on -> output = input
10             Multiplicand_out <= Multiplicand_in;
11         end
12     end
13
14 endmodule

```

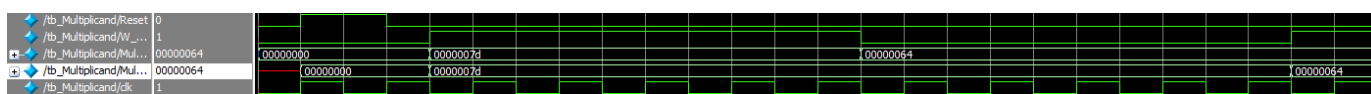
- Multiplicand test bench

```

1 // Setting timescale
2 `timescale 10 ns / 1 ns
3
4 // Declarations
5 `define LOW      1'b0
6 `define HIGH     1'b1
7
8 module tb_Multiplicand;
9
10     // Inputs
11     reg Reset = `LOW;
12     reg W_ctrl = `LOW;
13     reg [31:0] Multiplicand_in = 32'b0;
14
15     // Outputs
16     wire [31:0] Multiplicand_out;
17
18     // Clock
19     reg clk = `LOW;
20
21     // Instantiate the Unit Under Test (UUT)
22     Multiplicand UUT(
23         // Outputs
24         .Multiplicand_out(Multiplicand_out),
25         // Inputs
26         .Multiplicand_in(Multiplicand_in),
27         .rst(Reset),
28         .w_ctrl(W_ctrl)
29     );
30
31     // Generate Clock
32     always
33     begin : ClockGenerator
34         #1 clk <= ~clk;           // toggle clock signal every one timescale delay
35     end
36
37     initial begin
38         // Wait positive edge of clock signal
39         @(posedge clk);
40
41         // Reset UUT
42         Reset <= `HIGH;
43         @(posedge clk);
44         Reset <= `LOW;
45
46         // Wait negative edge of clock signal
47         @(negedge clk);
48
49         // Write data into Multiplicand Register
50         Multiplicand_in <= 32'd125;
51         W_ctrl <= `HIGH;
52
53         //Wait 10ns to off W_ctrl and change input
54         #10;
55         W_ctrl <= `LOW;
56         Multiplicand_in <= 32'd100;
57
58         //Wait 10ns turn on W_ctrl
59         #10;
60         W_ctrl <= `HIGH;
61
62         // Wait some time
63         #2;
64
65         // Stop the simulation
66         $stop();
67     end
68 endmodule
69

```

- **Multiplicand Module Simulation**



- ◆ W_ctrl控制output正常，當W_ctrl = 0時input變動，output維持不變
- ◆ Reset功能正常

- ALU Module

```

1  `define add      6'b001001
2
3  module ALU(Src_1, Src_2, ADDU_ctrl, ALU_Result, ALU_Carry);
4      //inputs
5      input [31:0]Src_1,Src_2;
6      input [5:0]ADDU_ctrl;
7
8      //outputs
9      output reg [31:0]ALU_Result;
10     output reg ALU_Carry;
11
12     always@(Src_1 or Src_2 or ADDU_ctrl)begin
13         //Perform ADDU operation
14         if(ADDU_ctrl == `add)begin
15             {ALU_Carry, ALU_Result} <= Src_1 + Src_2;
16         end else begin
17             ALU_Result <= 0;
18             ALU_Carry <= 0;
19         end
20     end
21 end
22 endmodule
23

```

- ALU test bench

```

1  // Setting timescale
2  `timescale 10 ns / 1 ns
3
4  module tb_ALU;
5
6      // Inputs
7      reg [5:0]add_ctrl = 0;
8      reg [31:0]Src_1, Src_2 = 32'b0;
9
10     // Outputs
11     wire [31:0] result;
12     wire carry;
13
14     // Clock
15     reg clk = 1'b0;
16
17     // Instantiate the Unit Under Test (UUT)
18     ALU UUT(
19         .Src_1(Src_1),
20         .Src_2(Src_2),
21         .ADDU_ctrl(add_ctrl),
22         .ALU_Result(result),
23         .ALU_Carry(carry)
24     );
25
26     // Generate Clock
27     always
28     begin : ClockGenerator
29         #1 clk <= ~clk;          // toggle clock signal every one timescale delay
30     end
31

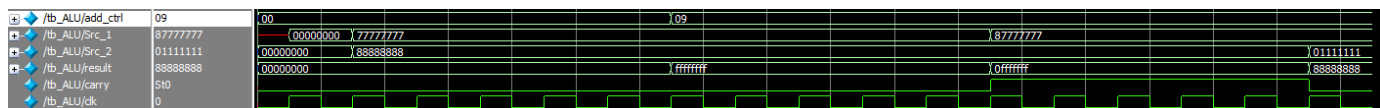
```

```

32      initial begin
33          // Wait positive edge of clock signal
34          @(posedge clk);
35
36          // Reset UUT
37          Src_1 <= 0;
38          Src_2 <= 0;
39          add_ctrl <= 0;
40
41          //assign value to input sources
42          @(posedge clk);
43          Src_1 <= 32'h7777_7777;
44          Src_2 <= 32'h8888_8888;
45
46          //Wait 10ns to give add operation code
47          #10;
48          add_ctrl <= 6'b001001;
49
50          //Wait 10ns, change inputs
51          #10;
52          Src_1 <= 32'h8777_7777;
53
54          //Wait 10ns, change inputs
55          #10;
56          Src_2 <= 32'h0111_1111;
57
58          // Wait some time
59          #2;
60
61          // Stop the simulation
62          $stop();
63      end
64
65  endmodule

```

• ALU Module Simulation



- ◆ 加法功能正常，沒有指令時輸入改變，輸出為0
- ◆ 指令判斷為加法時，輸出為兩個輸入相加
- ◆ carry正常進位

- Product Module

```

1 module Product(Multiplier_in, ALU_Carry, ALU_Result, SRL_ctrl, w_ctrl, ready, rst, clk, Product_out);
2 //inputs
3 input [31:0]Multiplier_in, ALU_Result;
4 input ALU_Carry, SRL_ctrl, w_ctrl, ready, rst, clk;
5
6 //output
7 output reg [63:0]Product_out;
8
9 reg [63:0]tempReg;
10 //the value to check if the multiplier loaded
11 reg loaded;
12
13 always@(negedge clk)begin
14     //reset the product register
15     if(rst)begin
16         Product_out <= 0;
17         tempReg <= 0;
18         loaded <= 0;
19     end else begin
20         if(~ready)begin
21             if(~loaded)begin //has not load multiplier yet -> put multiplier to the rightmost
22                 tempReg[31:0] = Multiplier_in;
23                 Product_out = tempReg;
24             end
25             else begin
26                 if(SRL_ctrl)begin
27                     //if LSB = 1 -> w_ctrl = 1 -> add the multiplicand to the leftmost and shift
28                     if(w_ctrl)begin
29                         tempReg[63:32] = ALU_Result;
30                         tempReg = tempReg >> 1;
31                         tempReg[63] = ALU_Carry;
32                         Product_out = tempReg;
33                     end
34                     //if LSB != 1 -> w_ctrl = 0 -> shift right directly
35                     else begin
36                         tempReg = tempReg >> 1;
37                         Product_out = tempReg;
38                     end
39                 end
40                 end
41                 loaded = 1;
42                 //if the loop is done -> ready = 1 -> output stays the same
43                 end else Product_out <= Product_out;
44             end
45         end
46     endmodule

```

- Product test bench

```

1 // Setting timescale
2 `timescale 10 ns / 1 ns
3
4 `define DELAY 1 // # * timescale
5
6 // Declarations
7 `define LOW 1'b0
8 `define HIGH 1'b1
9
10 module tb_Product;
11 //inputs
12 reg [31:0]Multiplier = 32'b0;
13 reg [31:0]ALU_Result = 32'b0;
14 reg ALU_Carry = 0;
15 reg SRL_ctrl = `LOW;
16 reg w_ctrl = `LOW;
17 reg reset = `LOW;
18 reg ready = `LOW;
19 //output
20 wire [63:0]Product;
21
22 //clock
23 reg clk = `LOW;
24
25 // Instantiate the Unit Under Test (UUT)
26 Product UUT(
27     .Multiplier_in(Multiplier),
28     .ALU_Carry(ALU_Carry),
29     .ALU_Result(ALU_Result),
30     .SRL_ctrl(SRL_ctrl),
31     .w_ctrl(w_ctrl),
32     .ready(ready),
33     .rst(reset),
34     .clk(clk),
35     .Product_out(Product)
36 );
37

```

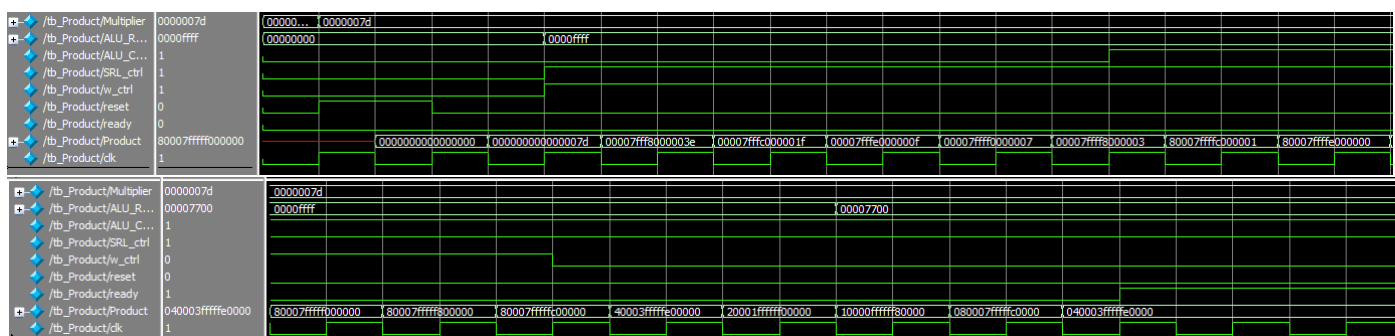


```

37
38 // Generate Clock
39 always
40 begin : ClockGenerator
41     #1 clk <= ~clk;          // toggle clock signal every one timescale delay
42 end
43
44 initial begin
45     // Wait positive edge of clock signal
46     @(posedge clk);
47
48     // Reset UUT
49     reset <= `HIGH;
50     ready <= `LOW;
51     Multiplier <= 32'd125;
52     // Wait negative edge of clock signal and end reset
53     @(posedge clk);
54     reset = `LOW;
55
56     @(posedge clk);
57     // Write data into Product Register
58     SRL_ctrl <= `HIGH;
59     w_ctrl <= `HIGH;
60     ALU_Carry <= 0;
61     ALU_Result <= 32'h0000_FFFF;
62
63     //Wait 10ns change carry to 1
64     #10;
65     ALU_Carry <= 1;
66
67     //Wait 10ns w_ctrl off
68     #10;
69     w_ctrl <= `LOW;
70
71     //Wait 5ns change ALU value
72     #5;
73     ALU_Result <= 32'h0000_7700;
74
75     #5;
76     ready <= `HIGH;
77     // Wait some time
78     #10;
79
80     // Stop the simulation
81     $stop();
82 end
83
84 endmodule

```

• Product Module Simulation



- ◆ W_ctrl on時，product左32bits正常輸入ALU結果
- ◆ Shift時，ALU的進位shift正常
- ◆ W_ctrl off時，shift正常，且不輸入ALU結果
- ◆ Ready rise後，product輸出保持

- CompMultiplier Module

```

1  module CompMultiplier(Product, ready, Multiplicand, Multiplier, run, reset, clk);
2      //outputs
3      output [63:0] Product;
4      output ready;
5      //inputs
6      input [31:0] Multiplicand;
7      input [31:0] Multiplier;
8      input run, reset, clk;
9      //wires between modules
10     wire w_ctrl, SRL_ctrl, ALU_Carry;
11     wire [31:0] Multiplicand_out, ALU_Result;
12     wire [5:0] ADDU_ctrl;
13
14     //instantiate Multiplicand Module
15     Multiplicand multiplicandReg(
16         .Multiplicand_in(Multiplicand),
17         .rst(reset),
18         .w_ctrl(w_ctrl),
19         .Multiplicand_out(Multiplicand_out)
20     );
21     //instantiate Control Module
22     Control controler(
23         .run(run),
24         .rst(reset),
25         .clk(clk),
26         .LSB(Product[0]),
27         .w_ctrl(w_ctrl),
28         .SRL_ctrl(SRL_ctrl),
29         .ready(ready),
30         .ADDU_ctrl(ADDU_ctrl)
31     );
32     //instantiate ALU Module
33     ALU ALUnit(
34         .Src_1(Product[63:32]),
35         .Src_2(Multiplicand_out),
36         .ADDU_ctrl(ADDU_ctrl),
37         .ALU_Result(ALU_Result),
38         .ALU_Carry(ALU_Carry)
39     );
40     //instantiate Product Module
41     Product ProductReg(
42         .Multiplier_in(Multiplier),
43         .ALU_Carry(ALU_Carry),
44         .ALU_Result(ALU_Result),
45         .SRL_ctrl(SRL_ctrl),
46         .w_ctrl(w_ctrl),
47         .ready(ready),
48         .rst(reset),
49         .clk(clk),
50         .Product_out(Product)
51     );
52 endmodule

```

- CompMultiplier test bench

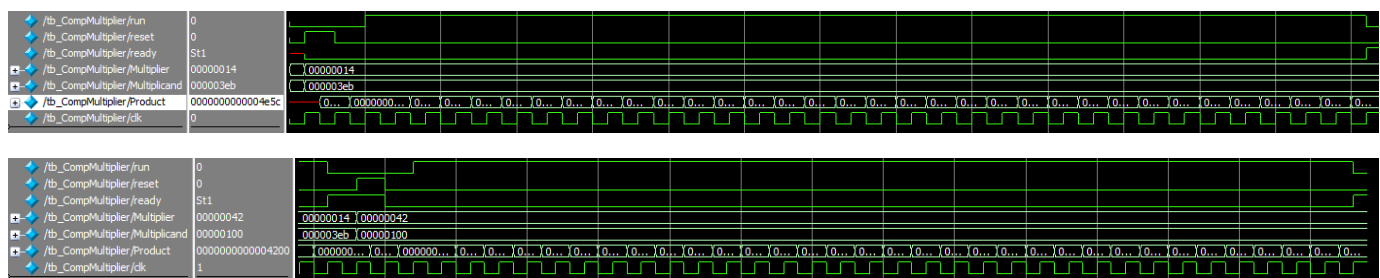
```

28 // Setting timescale
29 `timescale 10 ns / 1 ns
30
31 // Declarations
32 `define DELAY 1 // # * timescale
33 `define INPUT_FILE "testbench/tb_CompMultiplier.in"
34 `define OUTPUT_FILE "testbench/tb_CompMultiplier.out"
35
36 // Declaration
37 `define LOW 1'b0
38 `define HIGH 1'b1
39
40 module tb_CompMultiplier;
41
42 // Inputs
43 reg reset;
44 reg run;
45 reg [31:0] Multiplicand;
46 reg [31:0] Multiplier;
47
48 // Outputs
49 wire [63:0] Product;
50 wire ready;
51
52 // Clock
53 reg clk = `LOW;
54
55 // Testbench variables
56 reg [63:0] read_data;
57 integer input_file;
58 integer output_file;
59 integer i;
60
61 // Instantiate the Unit Under Test (UUT)
62 CompMultiplier UUT(
63 // Outputs
64 .Product(Product),
65 .ready(ready),
66 // Inputs
67 .Multiplicand(Multiplicand),
68 .Multiplier(Multiplier),
69 .run(run),
70 .reset(reset),
71 .clk(clk)
72 );
73
74 initial
75 begin : Preprocess
76 // Initialize inputs
77 reset = `LOW;
78 run = `LOW;
79 Multiplicand = 32'd0;
80 Multiplier = 32'd0;
81
82 // Initialize testbench files
83 input_file = $fopen(INPUT_FILE, "r");
84 output_file = $fopen(OUTPUT_FILE);
85
86 #`DELAY; // Wait for global reset to finish
87
88
89
90 always
91 begin : ClockGenerator
92 #`DELAY;
93 clk <= ~clk;
94
95 end
96
97 always
98 begin : StimuliProcess
99 // Start testing
100 while (!$feof(input_file))
101 begin
102 $fscanf(input_file, "%x\n", read_data);
103 @(posedge clk); // Wait clock
104 {Multiplicand, Multiplier} = read_data;
105 reset = `HIGH;
106 @(posedge clk); // Wait clock
107 reset = `LOW;
108 @(posedge clk); // Wait clock
109 run = `HIGH;
110 @(posedge ready); // Wait ready
111 run = `LOW;
112
113 end
114
115 #`DELAY; // Wait for result stable
116
117 // Close output file for safety
118 $fclose(output_file);
119
120 // Stop the simulation
121 $stop();
122
123 end
124
125 always @(posedge ready)
126 begin : Monitoring
127 $display("Multiplicand:%d, Multiplier:%d, Multiplicand, Multiplier);
128 $display("result:%d", Product);
129 $fdisplay(output_file, "%t,%x", $time, Product);
130
131 end
132
133 endmodule

```

- CompMultiplier Module Simulation

000003EB_00000014
00000100_00000042| ->.in file



- ◆ 兩個乘法結果正確
- ◆ Reset功能正常
- ◆ Ready訊號正常上升、下降

Part II: Implement A 32-bit Unsigned Complete Divider

- Control Module

```
1  `define add      6'b001001
2  `define sub      6'b001010
3  module Control(run, rst, clk, MSB, w_ctrl, SLL_ctrl, SRL_ctrl, ready, OP_ctrl);
4  //inputs
5  input run, rst, clk, MSB;
6  //outputs
7  output reg w_ctrl, SLL_ctrl, SRL_ctrl, ready;
8  output reg [5:0]OP_ctrl;
9
10 integer counter;
11
12 always@(posedge clk)begin
13     //reset
14     if(rst)begin
15         w_ctrl <= 0;
16         SLL_ctrl <= 0;
17         SRL_ctrl <= 0;
18         ready <= 0;
19         OP_ctrl <= 0;
20         counter <= 0;
21     end
22     else if(run) begin
23         if(counter < 64)begin
24             counter <= counter + 1;
25             //step1: Remainder minus divisor
26             if(counter % 2 == 0)begin
27                 w_ctrl <= 1;
28                 OP_ctrl <= `sub;
29                 SLL_ctrl <= 0;
30             end
31             //step2: check the value of remainder
32             else if(counter % 2 == 1)begin
33                 SLL_ctrl <= 1;
34                 //if is negative -> MSB = 1 -> add divisor back and shift
35                 if(MSB)begin
36                     w_ctrl <= 1;
37                     OP_ctrl <= `add;
38                 end
39                 //if positive -> keep it and shift
40                 else begin
41                     w_ctrl <= 0;
42                     OP_ctrl <= 0;
43                 end
44             end
45             //the last step: shift the remainder right
46         end else if(counter == 64)begin
47             SRL_ctrl <= 1;
48             SLL_ctrl <= 0;
49             counter <= counter + 1;
50         end else ready <= 1;
51     end
52 end
53 endmodule
```

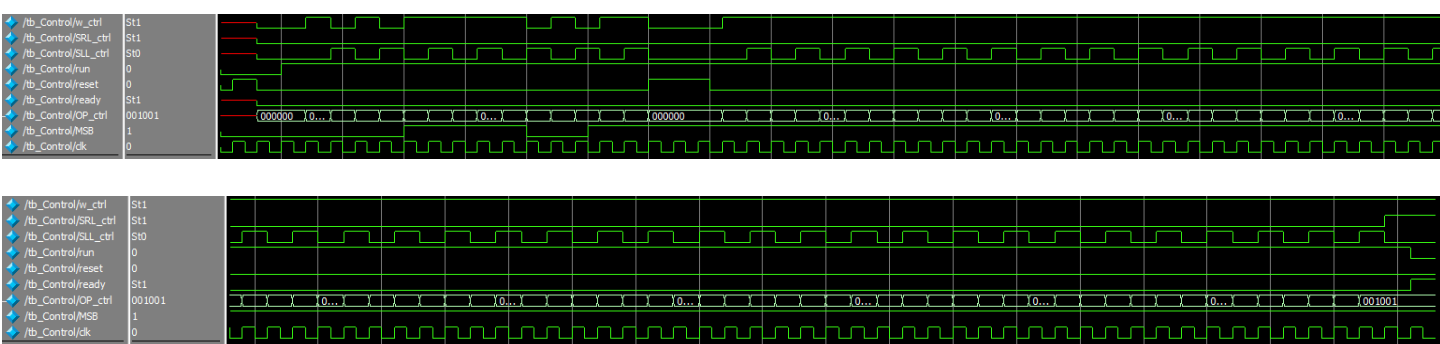
- Control test bench

```

1 // Setting timescale
2 `timescale 10 ns / 1 ns
3
4
5 `define DELAY          1          // # * timescale
6
7 // Declarations
8 `define LOW            1'b0
9 `define HIGH           1'b1
10
11 module tb_Control;
12
13     //inputs
14     reg run = `LOW;
15     reg reset = `LOW;
16     reg MSB = 0;
17
18     //outputs
19
20     wire w_ctrl, ready, SRL_ctrl, SLL_ctrl;
21     wire [5:0]OP_ctrl;
22
23     // Clock
24     reg clk = `LOW;
25
26     Control UUT(
27         //inputs
28         .run(run),
29         .rst(reset),
30         .clk(clk),
31         .MSB(MSB),
32         //outputs
33         .w_ctrl(w_ctrl),
34         .SRL_ctrl(SRL_ctrl),
35         .SLL_ctrl(SLL_ctrl),
36         .ready(ready),
37         .OP_ctrl(OP_ctrl)
38     );
39
40     // Generate Clock
41     always
42     begin : ClockGenerator
43         #1 clk <= ~clk;
44     end
45
46     initial begin
47         // Wait positive edge of clock signal
48         @(posedge clk);
49
50         // Reset UUT
51         reset <= `HIGH;
52         run <= `LOW;
53
54         // End reset
55         @(posedge clk);
56         reset <= `LOW;
57         //set run to 1
58         @(posedge clk);
59         run <= `HIGH;
60         // Wait 10ns and assign LSB -> 1
61         #10;
62         MSB <= 1;
63         // Wait 10ns and assign LSB -> 0
64         #10;
65         MSB <= 0;
66         // Wait 10ns and assign LSB -> 1 and wait 5ns to reset
67         #5;
68         MSB <= 1;
69         #5;
70         reset <= `HIGH;
71         #5;
72         reset <= `LOW;
73         // Wait 10ns and assign LSB -> 1
74         #10;
75         MSB <= 1;
76         //Wait for multiplication iteration end
77         @(posedge ready);
78         run <= `LOW;
79
80         // Wait some time
81         #2;
82
83         // Stop the simulation
84         $stop();
85     end
86
87 endmodule

```

- Control Simulation



- ◆ Reset功能正常
- ◆ Loop結束，ready訊號正常上升
- ◆ Controller每個loop的兩個步驟都正常執行

- Divisor Module

```
1 module Divisor(Divisor_in, rst, w_ctrl, Divisor_out);
2   input rst, w_ctrl;
3   input [31:0]Divisor_in;
4   output reg [31:0]Divisor_out;
5
6   always@(*)begin
7     if(rst)begin
8       Divisor_out <= 0;
9     end else if(w_ctrl)begin
10      Divisor_out <= Divisor_in;
11    end
12  end
13
14 endmodule
```

*因為跟Part1的Multiplicand Module幾乎一樣，加上報告空間不足，所以就沒有做Divisor Module的test bench

- ALU Module

```
1 `define addu    6'b001001
2 `define subu    6'b001010
3
4 module ALU(Src_1, Src_2, OP_ctrl, ALU_Result, ALU_Carry);
5   //inputs
6   input [31:0]Src_1, Src_2;
7   input [5:0]OP_ctrl;
8
9   //outputs
10  output reg [31:0]ALU_Result;
11  output reg ALU_Carry;
12
13  always@(*)begin
14    //check the operation code
15    case(OP_ctrl)
16      `addu: //perform add unsigned operation
17        {ALU_Carry, ALU_Result} <= Src_1 + Src_2;
18      `subu: //perform sub unsigned operation
19        {ALU_Carry, ALU_Result} <= Src_1 - Src_2;
20      default:
21        {ALU_Carry, ALU_Result} <= 0;
22    endcase
23  end
24
25 endmodule
```

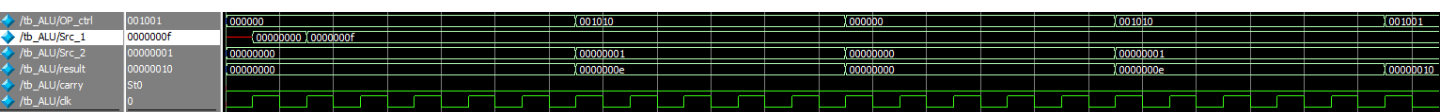
- ALU test bench

```

1 // Setting timescale
2 `timescale 10 ns / 1 ns
3 `define sub 6'b001010
4 `define add 6'b001001
5 module tb_ALU;
6
7     // Inputs
8     reg [5:0]OP_ctrl = 0;
9     reg [31:0]Src_1, Src_2 = 32'b0;
10
11     // Outputs
12     wire [31:0] result;
13     wire carry;
14
15     // Clock
16     reg clk = 1'b0;
17
18     // Instantiate the Unit Under Test (UUT)
19     ALU UUT(
20         .Src_1(Src_1),
21         .Src_2(Src_2),
22         .OP_ctrl(OP_ctrl),
23         .ALU_Result(result),
24         .ALU_Carry(carry)
25     );
26
27     // Generate Clock
28     always
29     begin : ClockGenerator
30         #1 clk <= ~clk;          // toggle clock signal every one timescale delay
31     end
32
33     initial begin
34         // Wait positive edge of clock signal
35         @(posedge clk);
36
37         // Reset UUT
38         Src_1 <= 0;
39         Src_2 <= 0;
40         OP_ctrl <= 0;
41
42         //assign value to input sources
43         @(posedge clk);
44         Src_1 <= 32'h0000_000f;
45
46         #10;
47         Src_2 <= 32'h1;
48         OP_ctrl <= `sub;
49
50         //Wait 10ns, change inputs
51         #10;
52         Src_2 <= 32'h0;
53         OP_ctrl <= 0;
54
55         //Wait 10ns, change inputs
56         #10;
57         Src_2 <= 32'h1;
58         OP_ctrl <= `sub;
59
60         #10;
61         OP_ctrl <= `add;
62
63         // Wait some time
64         #2;
65
66         // Stop the simulation
67         $stop();
68     end
69 endmodule
70

```

- ALU Simulation



- Remainder Module

```

1 module Remainder(Dividend_in, ALU_Carry, ALU_Result, SLL_ctrl, SRL_ctrl, w_ctrl, ready, rst, clk, Remainder_out);
2 //inputs
3 input [31:0]Dividend_in, ALU_Result;
4 input ALU_Carry, SLL_ctrl, SRL_ctrl, w_ctrl, ready, rst, clk;
5 //output
6 output reg [63:0]Remainder_out;
7 reg [63:0]tempReg;
8 //the value to check if the devidend loaded
9 reg loaded;
10
11 always@(negedge clk)begin
12 //reset the product register
13 if(rst)begin
14 Remainder_out <= 0;
15 tempReg <= 0;
16 loaded <= 0;
17 end else begin
18 if(~ready)begin
19 if(~loaded)begin
20 //has not load multiplier yet -> put dividend to the rightmost and shift left 1
21 tempReg[32:1] = Dividend_in;
22 Remainder_out = tempReg;
23 end
24 else begin
25 //step2 of the controller
26 if(SLL_ctrl)begin
27 //if MSB = 1 -> w_ctrl = 1 -> add the divisor back to the leftmost and shift
28 if(w_ctrl)begin
29 tempReg[63:32] = ALU_Result;
30 tempReg = tempReg << 1;
31 tempReg[0] = 1'b0; //fill right most bit with 0
32 Remainder_out = tempReg;
33 end
34 //if MSB != 1 -> w_ctrl = 0 -> move and shift
35 else begin
36 tempReg = tempReg << 1;
37 tempReg[0] = 1'b1; //fill right most bit with 1
38 Remainder_out = tempReg;
39 end
40 end
41 //step1 of the controller
42 //no shift left
43 else begin
44 if(SRL_ctrl)begin
45 Remainder_out[62:32] = tempReg[63:33];
46 Remainder_out[63] = 1'b0;
47 end
48 else if(w_ctrl)begin
49 tempReg[63:32] = ALU_Result;
50 Remainder_out = tempReg;
51 end
52 end
53 end
54 loaded = 1;
55 //if the loop is done -> ready = 1 -> output stays the same
56 end else Remainder_out <= Remainder_out;
57 end
58 end
59 endmodule

```


- Remainder test bench

```

1 // Setting timescale
2 `timescale 10 ns / 1 ns
3
4 `define DELAY 1 // # * timescale
5
6 // Declarations
7 `define LOW 1'b0
8 `define HIGH 1'b1
9
10 module tb_Product;
11     //inputs
12     reg [31:0]Dividend = 32'b0;
13     reg [31:0]ALU_Result = 32'b0;
14     reg ALU_Carry = 0;
15     reg SRL_ctrl = `LOW;
16     reg SLL_ctrl = `LOW;
17     reg w_ctrl = `LOW;
18     reg reset = `LOW;
19     reg ready = `LOW;
20     //output
21     wire [63:0]Remainder;
22
23     //clock
24     reg clk = `LOW;
25
26     // Instantiate the Unit Under Test (UUT)
27     Remainder UUT(
28         //inputs
29         .Dividend_in(Dividend),
30         .ALU_Carry(ALU_Carry),
31         .ALU_Result(ALU_Result),
32         .SLL_ctrl(SLL_ctrl),
33         .SRL_ctrl(SRL_ctrl),
34         .w_ctrl(w_ctrl),
35         .ready(ready),
36         .rst(reset),
37         .clk(clk),
38         //output
39         .Remainder_out(Remainder)
40     );
41
42     // Generate Clock
43     always
44     begin : ClockGenerator
45         #1 clk <= ~clk;
46     end
47
48 initial begin
49     // Wait positive edge of clock signal
50     @(posedge clk);
51
52     // Reset UUT
53     reset <= `HIGH;
54     ready <= `LOW;
55     SLL_ctrl <= `LOW;
56     SRL_ctrl <= `LOW;
57     w_ctrl <= `LOW;
58     Dividend <= 32'd125;
59     // Wait positive edge of clock signal and end reset
60     @(posedge clk);
61     reset = `LOW;
62
63     @(posedge clk);
64     // Write data into Product Register
65     w_ctrl <= `HIGH;
66     ALU_Carry <= 0;
67     ALU_Result <= 32'h0000_FFFF;
68
69     //Wait positive edge of clk and SLL on
70     @(posedge clk);
71     SLL_ctrl <= `HIGH;
72
73     //Wait positive edge of clk w_ctrl off
74     @(posedge clk);
75     w_ctrl <= `LOW;
76
77     //Wait positive edge of clk and SLL off
78     @(posedge clk);
79     SLL_ctrl <= `LOW;
80
81     //Wait positive edge of clk and SRL on
82     @(posedge clk);
83     SRL_ctrl <= `HIGH;
84
85     //Wait positive edge of clk and ready
86     @(posedge clk);
87     ready <= `HIGH;
88     // Wait some time
89     #10;
90
91     // Stop the simulation
92     $stop();
93 end
94
95 endmodule

```

- Remainder Simulation

Dividend	0000007d
/ALU_R...	0000ffff
/ALU_C...	00000000
/SLI_ctrl	0
/SLI_ctrl	1
/w_ctrl	0
/w_ctrl	0
/reset	0
/ready	1
/Remain...	0001ffff000003e9
/ck	0

- ◆ Reset功能正常
- ◆ ready訊號正常上升，且ready後輸出訊號保持
- ◆ Shift left & right正常
- ◆ Negative edge trigger運作正常

- CompDivider Module

```

1  module CompDivider(Quotient, Remainder, ready, Dividend, Divisor, run, reset, clk);
2      //outputs
3      output [31:0]Quotient, Remainder;
4      output ready;
5      //inputs
6      input [31:0]Dividend, Divisor;
7      input run, reset, clk;
8      //wires between the modules
9      wire [63:0]Remainder_out;
10     wire [31:0]Divisor_out, ALU_Result;
11     wire [5:0]OP_ctrl;
12     wire ALU_Carry, w_ctrl, SLL_ctrl, SRL_ctrl;
13
14     //instantiate ALU
15     ALU ALUnit(
16         .Src_1(Remainder_out[63:32]),
17         .Src_2(Divisor_out),
18         .OP_ctrl(OP_ctrl),
19         .ALU_Result(ALU_Result),
20         .ALU_Carry(ALU_Carry)
21     );
22     //instantiate Divisor register
23     Divisor DivisorReg(
24         .Divisor_in(Divisor),
25         .rst(reset),
26         .w_ctrl(w_ctrl),
27         .Divisor_out(Divisor_out)
28     );
29     //instantiate Controller
30     Control controller(
31         .run(run),
32         .rst(reset),
33         .clk(clk),
34         .MSB(Remainder_out[63]),
35         .w_ctrl(w_ctrl),
36         .SLL_ctrl(SLL_ctrl),
37         .SRL_ctrl(SRL_ctrl),
38         .ready(ready),
39         .OP_ctrl(OP_ctrl)
40     );
41     //instantiate Remainder register
42     Remainder RemainderReg(
43         .Dividend_in(Dividend),
44         .ALU_Carry(ALU_Carry),
45         .ALU_Result(ALU_Result),
46         .SLL_ctrl(SLL_ctrl),
47         .SRL_ctrl(SRL_ctrl),
48         .w_ctrl(w_ctrl),
49         .ready(ready),
50         .rst(reset),
51         .clk(clk),
52         .Remainder_out(Remainder_out)
53     );
54     //assign the value to remainder & Quotient by the value in 64-bit Remainder register
55     assign Remainder = Remainder_out[63:32];
56     assign Quotient = Remainder_out[31:0];
57     endmodule

```

• CompDivider testbench

```

28 // Setting timescale
29 `timescale 10 ns / 1 ns
30
31 // Declarations
32 `define DELAY 1 // # * timescale
33 `define INPUT_FILE "testbench/tb_CompDivider.in"
34 `define OUTPUT_FILE "testbench/tb_CompDivider.out"
35
36 // Declaration
37 `define LOW 1'b0
38 `define HIGH 1'b1
39
40 module tb_CompDivider;
41
42 // Inputs
43 reg Reset;
44 reg Run;
45 reg [31:0] Dividend_in;
46 reg [31:0] Divisor_in;
47
48 // Outputs
49 wire [31:0] Quotient_out;
50 wire [31:0] Remainder_out;
51 wire Ready;
52
53 // Clock
54 reg clk = `LOW;
55
56 // Testbench variables
57 reg [63:0] read_data;
58 integer input_file;
59 integer output_file;
60 integer i;
61
62 // Instantiate the Unit Under Test (UUT)
63 CompDivider UUT(
64 // Outputs
65 .Quotient(Quotient_out),
66 .Remainder(Remainder_out),
67 .ready(Ready),
68 // Inputs
69 .Dividend(Dividend_in),
70 .Divisor(Divisor_in),
71 .run(Run),
72 .reset(Reset),
73 .clk(clk)
74 );

```

```

76 initial
77 begin : Preprocess
78 // Initialize inputs
79 Reset = `LOW;
80 Run = `LOW;
81 Dividend_in = 32'd0;
82 Divisor_in = 32'd0;
83
84 // Initialize testbench files
85 input_file = $fopen('INPUT_FILE', "r");
86 output_file = $fopen('OUTPUT_FILE');
87
88 #`DELAY; // Wait for global reset to finish
89
90 end
91
92 always
93 begin : ClockGenerator
94 #`DELAY;
95 clk <= ~clk;
96 end
97
98 always
99 begin : StimuliProcess
100 // Start testing
101 while (!$feof(input_file))
102 begin
103 $fscanf(input_file, "%x\n", read_data);
104 @(posedge clk); // Wait clock
105 (Dividend_in, Divisor_in) = read_data;
106 Reset = `HIGH;
107 @(posedge clk); // Wait clock
108 Reset = `LOW;
109 @(posedge clk); // Wait clock
110 Run = `HIGH;
111 @(posedge Ready); // Wait ready
112 Run = `LOW;
113 end
114 #`DELAY; // Wait for result stable
115
116 // Close output file for safety
117 $fclose(output_file);
118
119 // Stop the simulation
120 $stop();
121 end
122
123 always @(posedge Ready)
124 begin : Monitoring
125 $display("Dividend_in:%d, Divisor_in:%d", Dividend_in, Divisor_in);
126 $display("Quotient_out:%d, Remainder_out:%d", Quotient_out, Remainder_out);
127 $fdisplay(output_file, "%x %x", Quotient_out, Remainder_out);
128 end
129
130 endmodule

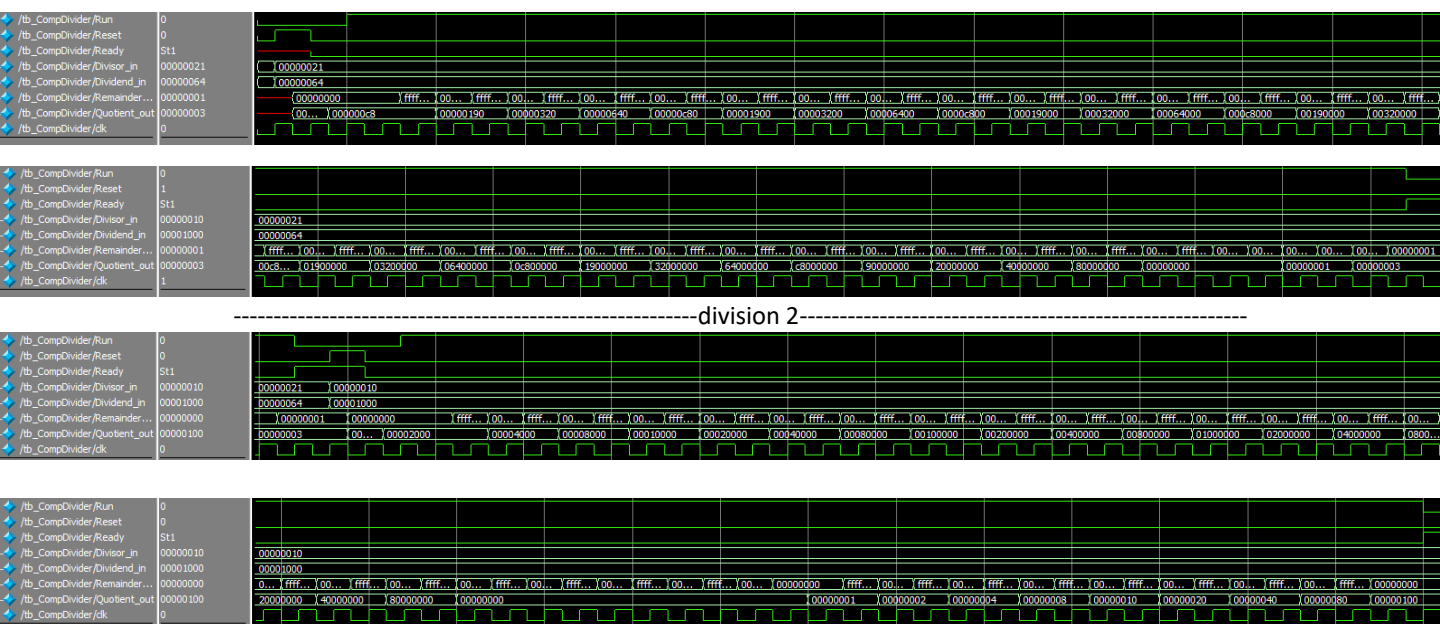
```

• CompDivider Simulation

tb_CompDivider.in - 記事本

檔案(F) 編輯(E) 格式(O) 檢視(V) ->.in file

00000064_00000021
00001000_00000010



- ◆ 兩個除法結果正確
- ◆ Reset功能正常
- ◆ Ready訊號正常上升、下降

Conclusion and insights

這次PA做的是乘法器和除法器，雖然在課堂上教的乘法和除法的概念都理解的不錯，但是真的要用verilog打出來也是有一點挑戰性，尤其是把每個module分開打的部分，因為又比上次的作業多了點判斷還有loop的部分，一開始困擾了我一下，但經過許久的思考後還是成功了，也蠻有成就感的。

乘法器的部分，因為第一個loop是把乘數放進去Product的暫存器最右邊的32個bits，所以總共是33個loop，比上課教的多了一個loop，而除法器的部分，因為每個loop的最一開始都要讓被除數去減除數，然後再去判斷結果是不是大於零，而要讓controller判斷的話，減出來的結果勢必也要讓Remainder module輸出成一個state才能給controller判斷，最後我的解決方法就是讓演算法的每一個loop在controller中分成兩個loop來進行，基數的loop會進行被除數減除數的運算，並且將結果存進暫存器，偶數的loop會判斷上一個loop進行的減法結果是不是正數，然後再進行shift，所以在controller內，原本演算法中的32個loop實際上會變成64個loop，而在最後還會再往右shift一次。在除法器的loop這邊思考很久有沒有辦法就真的只用32或多一點點的loop就能做出來，但沒想出甚麼比較不複雜的解法，還好最後採用的這個方法我自己覺得也是蠻不錯的。

這次除了最後完整的乘法器和除法器有給test bench以外，其他個別的module的test module都要自己打，原本以為要向之前的一樣要從外面讀.in檔，還好助教有給簡易版本的，雖然我還是覺得打test bench有點累有點辛苦，不過經過這次打了這麼多的test bench，也算是更加了解要怎麼去自己寫出test bench來測試各個module的運作了。

這次在打的時候會犯一些粗心的錯，像是在最後完整的除法器的module要將個別的module接起來的時候，沒填到一個參數，在compile的時候也沒有報錯，害我除錯除了很久，下次在做的時候要再細心一點。