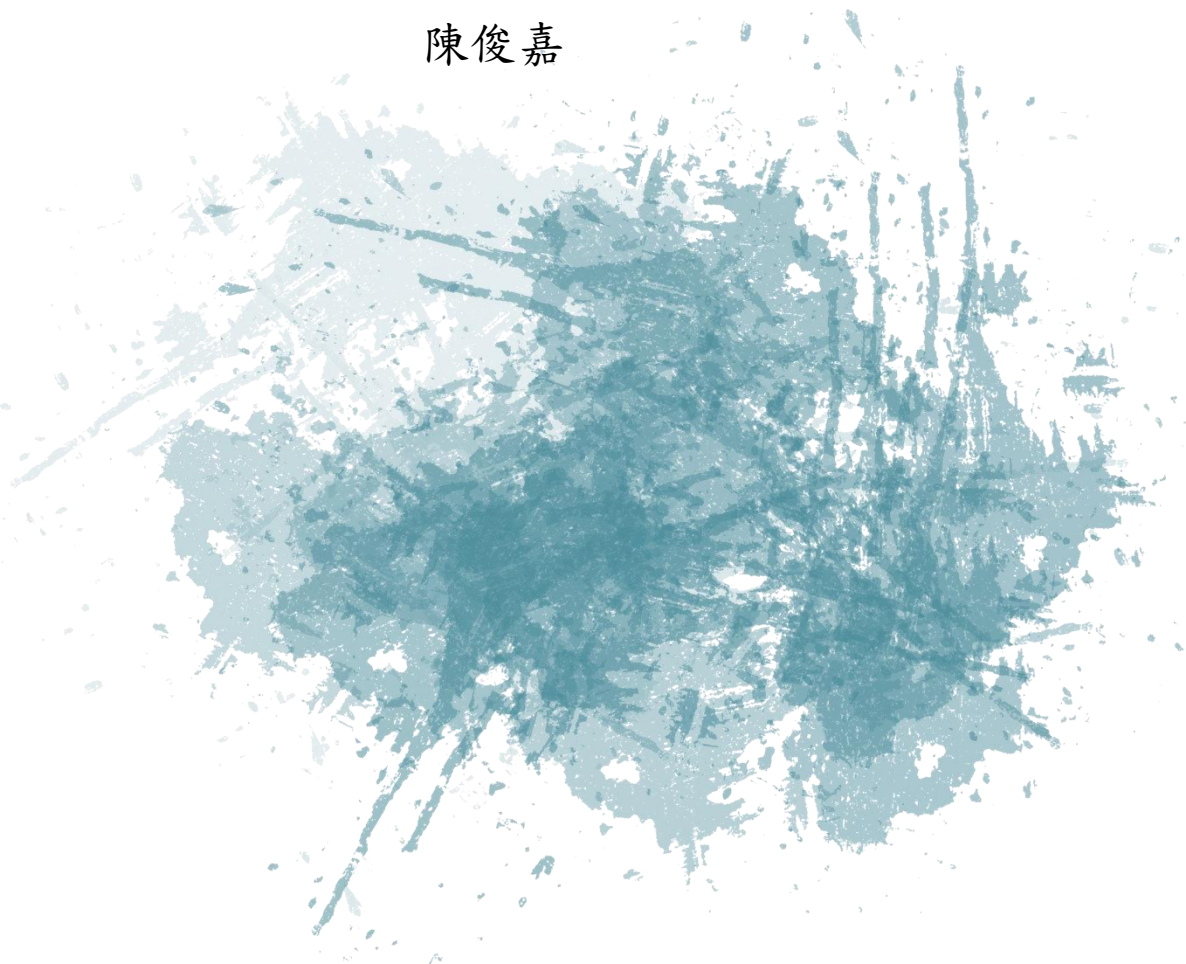
Computer Organization
Project 1

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
```

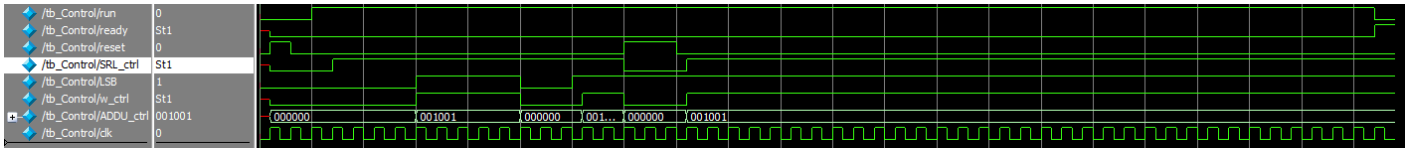
- Control test bench

```
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 evergy one timescale delay
43          end
```

```
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

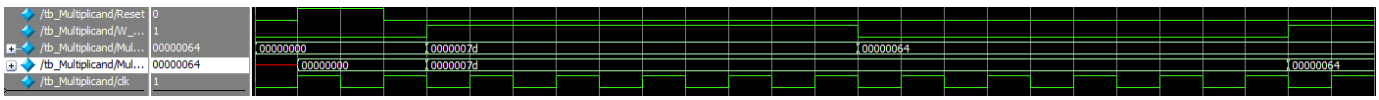
- Multiplicand Module

```
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

```
// Setting timescale
`timescale 10 ns / 1 ns

// Declarations
`define LOW     1'b0
`define HIGH    1'b1

module tb_Multiplicand;

        // Inputs
        reg Reset = `LOW;
        reg W_ctrl = `LOW;
        reg [31:0] Multiplicand_in = 32'b0;

        // Outputs
        wire [31:0] Multiplicand_out;

        // Clock
        reg clk = `LOW;

        // Instantiate the Unit Under Test (UUT)
        Multiplicand UUT(
                // Outputs
                .Multiplicand_out(Multiplicand_out),
                // Inputs
                .Multiplicand_in(Multiplicand_in),
                .rst(Reset),
                .w_ctrl(W_ctrl)
        );

        // Generate Clock
        always
        begin : ClockGenerator
                #1 clk <= ~clk;         // toggle clock signal evergy one timescale delay
        end
       initial begin
                // Wait positive edge of clock signal
                @(posedge clk);

                // Reset UUT
                Reset <= `HIGH;
                @(posedge clk);
                Reset <= `LOW;

                // Wait negative edge of clock signal
                @(negedge clk);

                // Write data into Multiplicand Register
                Multiplicand_in <= 32'd125;
                W_ctrl <= `HIGH;

                //Wait 10ns to off W_ctrl and change input
                #10;
                W_ctrl <= `LOW;
                Multiplicand_in <= 32'd100;

                //Wait 10ns turn on W_ctrl
                #10;
                W_ctrl <= `HIGH;

                // Wait some time
                #2;

                // Stop the simulation
                $stop();
        end

endmodule
```

- 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

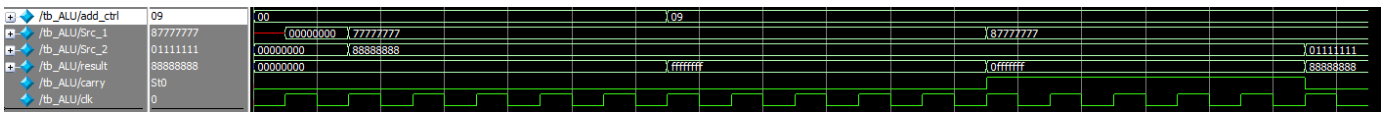
```
1   `define add     6'b001001
2
3   module ALU(Src_1, Src_2, ADDU_ctrl, ALU_Result, ALU_Carry);
4   //inputs
5   input [31:0]Src_1,Src_2;
6   input [5:0]ADDU_ctrl;
7
8   //outputs
9   output reg [31:0]ALU_Result;
10  output reg ALU_Carry;
11
12  always@(Src_1 or Src_2 or ADDU_ctrl)begin
13          //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
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 evergy one timescale delay
30          end
```

```
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

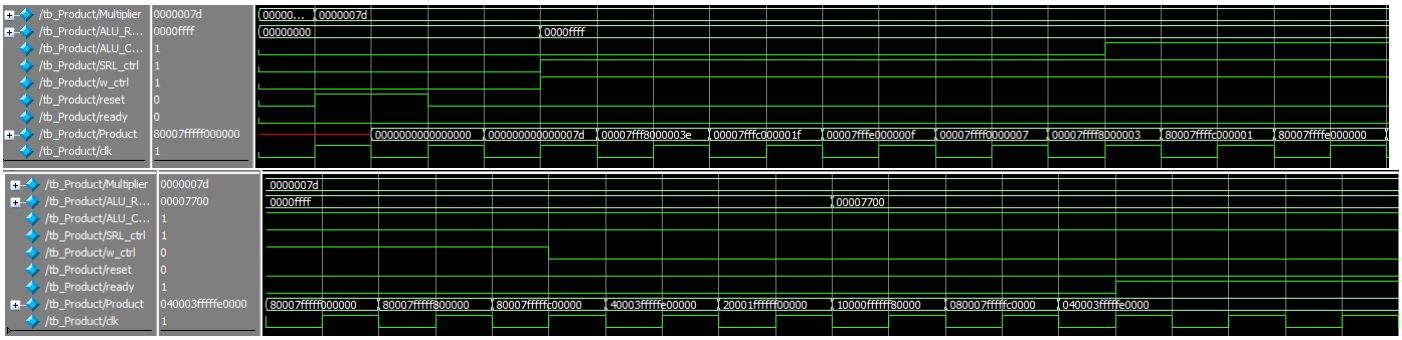
```
module Product(Multiplier_in, ALU_Carry, ALU_Result, SRL_ctrl, w_ctrl, ready, rst, clk, Product_out);
//inputs
input [31:0]Multiplier_in, ALU_Result;
input ALU_Carry, SRL_ctrl, w_ctrl, ready, rst, clk;

//output
output reg [63:0]Product_out;

reg [63:0]tempReg;
//the value to check if the multiplier loaded
reg loaded;

always@(negedge clk)begin
        //reset the product register
        if(rst)begin
                Product_out <= 0;
                tempReg <= 0;
                loaded <= 0;
        end else begin
                if(~ready)begin
                        if(~loaded)begin        //has not load multiplier yet -> put multiplier to the rightmost
                                tempReg[31:0] = Multiplier_in;
                                Product_out = tempReg;
                        end
                        else begin
                                if(SRL_ctrl)begin
                                        //if LSB = 1 -> w_ctrl = 1 -> add the multiplicand to the leftmost and shift
                                        if(w_ctrl)begin
                                                tempReg[63:32] = ALU_Result;
                                                tempReg = tempReg >> 1;
                                                tempReg[63] = ALU_Carry;
                                                Product_out = tempReg;
                                        end
                                        //if LSB != 1 -> w_ctrl = 0 -> shift right directly
                                        else begin
                                                tempReg = tempReg >> 1;
                                                Product_out = tempReg;
                                        end
                                end
                        end
                        loaded = 1;
                //if the loop is done -> ready = 1 -> output stays the same
                end else Product_out <= Product_out;
        end
end
endmodule
```

- Product test bench

```
// Setting timescale
`timescale 10 ns / 1 ns

`define DELAY                   1       // # * timescale

// Declarations
`define LOW     1'b0
`define HIGH    1'b1

module tb_Product;
        //inputs
        reg [31:0]Multiplier = 32'b0;
        reg [31:0]ALU_Result = 32'b0;
        reg ALU_Carry = 0;
        reg SRL_ctrl = `LOW;
        reg w_ctrl = `LOW;
        reg reset = `LOW;
        reg ready = `LOW;
        //output
        wire [63:0]Product;

        //clock
        reg clk = `LOW;

        // Instantiate the Unit Under Test (UUT)
        Product UUT(
                .Multiplier_in(Multiplier),
                .ALU_Carry(ALU_Carry),
                .ALU_Result(ALU_Result),
                .SRL_ctrl(SRL_ctrl),
                .w_ctrl(w_ctrl),
                .ready(ready),
                .rst(reset),
                .clk(clk),
                .Product_out(Product)
        );
```

```
38          // Generate Clock
39          always
40          begin : ClockGenerator
41                  #1 clk <= ~clk;         // toggle clock signal evergy one timescale delay
42          end
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

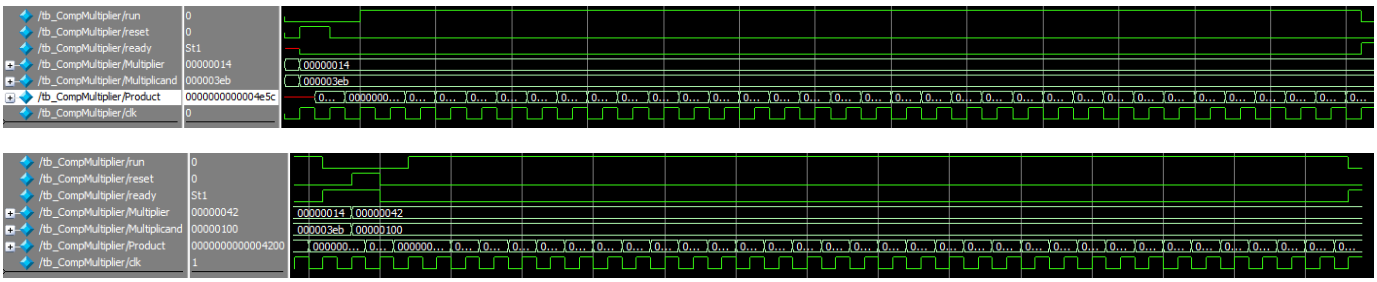
```
module CompMultiplier(Product, ready, Multiplicand, Multiplier, run, reset, clk);
//outputs
output [63:0] Product;
output ready;
//inputs
input [31:0] Multiplicand;
input [31:0] Multiplier;
input run, reset, clk;
//wires between modules
wire w_ctrl, SRL_ctrl, ALU_Carry;
wire [31:0]Multiplicand_out, ALU_Result;
wire [5:0]ADDU_ctrl;

//instantiate Multiplicand Module
Multiplicand multiplicandReg(
        .Multiplicand_in(Multiplicand),
        .rst(reset),
        .w_ctrl(w_ctrl),
        .Multiplicand_out(Multiplicand_out)
);
//instantiate Control Module
Control controler(
        .run(run),
        .rst(reset),
        .clk(clk),
        .LSB(Product[0]),
        .w_ctrl(w_ctrl),
        .SRL_ctrl(SRL_ctrl),
        .ready(ready),
        .ADDU_ctrl(ADDU_ctrl)
);
//instantiate ALU Module
ALU ALUnit(
        .Src_1(Product[63:32]),
        .Src_2(Multiplicand_out),
        .ADDU_ctrl(ADDU_ctrl),
        .ALU_Result(ALU_Result),
        .ALU_Carry(ALU_Carry)
);
//instantiate Product Module
Product ProductReg(
        .Multiplier_in(Multiplier),
        .ALU_Carry(ALU_Carry),
        .ALU_Result(ALU_Result),
        .SRL_ctrl(SRL_ctrl),
        .w_ctrl(w_ctrl),
        .ready(ready),
        .rst(reset),
        .clk(clk),
        .Product_out(Product)
);
endmodule
```

- CompMultiplier test bench

```
// Setting timescale
`timescale 10 ns / 1 ns

// Declarations
`define DELAY                   1       // # * timescale
`define INPUT_FILE              "testbench/tb_CompMultiplier.in"
`define OUTPUT_FILE             "testbench/tb_CompMultiplier.out"

// Declaration
`define LOW             1'b0
`define HIGH    1'b1

module tb_CompMultiplier;

        // Inputs
        reg reset;
        reg run;
        reg [31:0] Multiplicand;
        reg [31:0] Multiplier;

        // Outputs
        wire [63:0] Product;
        wire ready;

        // Clock
        reg clk = `LOW;

        // Testbench variables
        reg [63:0] read_data;
        integer input_file;
        integer output_file;
        integer i;

        // Instantiate the Unit Under Test (UUT)
        CompMultiplier UUT(
                // Outputs
                .Product(Product),
                .ready(ready),
                // Inputs
                .Multiplicand(Multiplicand),
                .Multiplier(Multiplier),
                .run(run),
                .reset(reset),
                .clk(clk)
        );

        initial
        begin : Preprocess
                // Initialize inputs
                reset           = `LOW;
                run             = `LOW;
                Multiplicand    = 32'd0;
                Multiplier      = 32'd0;

                // Initialize testbench files
                input_file      = $fopen(`INPUT_FILE, "r");
                output_file     = $fopen(`OUTPUT_FILE);

                #`DELAY;        // Wait for global reset to finish
        end

        always
        begin : ClockGenerator
                #`DELAY;
                clk <= ~clk;
        end

        always
        begin : StimuliProcess
                // Start testing
                while (!$feof(input_file))
                begin
                        $fscanf(input_file, "%x\n", read_data);
                        @(posedge clk); // Wait clock
                        {Multiplicand, Multiplier} = read_data;
                        reset = `HIGH;
                        @(posedge clk); // Wait clock
                        reset = `LOW;
                        @(posedge clk); // Wait clock
                        run = `HIGH;
                        @(posedge ready);       // Wait ready
                        run = `LOW;
                end

                #`DELAY;        // Wait for result stable

                // Close output file for safety
                $fclose(output_file);

                // Stop the simulation
                $stop();
        end

        always @(posedge ready)
        begin : Monitoring
                $display("Multiplicand:%d, Multiplier:%d", Multiplicand, Multiplier);
                $display("result:%d", Product);
                $fdisplay(output_file, "%t,%x", $time, Product);
        end

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

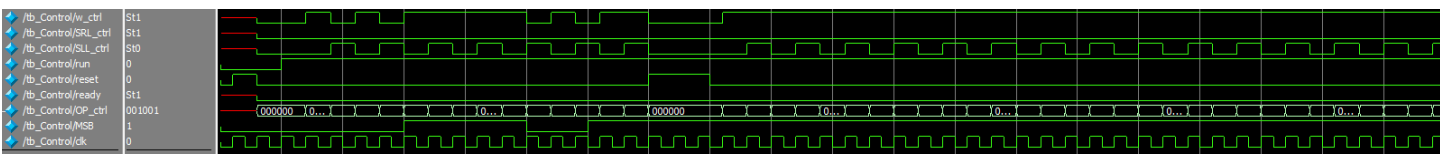
```
`define add     6'b001001
`define sub     6'b001010
module Control(run, rst, clk, MSB, w_ctrl, SLL_ctrl, SRL_ctrl, ready, OP_ctrl);
//inputs
input run, rst, clk, MSB;
//outputs
output reg w_ctrl, SLL_ctrl, SRL_ctrl, ready;
output reg [5:0]OP_ctrl;

integer counter;

always@(posedge clk)begin
        //reset
        if(rst)begin
                w_ctrl <= 0;
                SLL_ctrl <= 0;
                SRL_ctrl <=0;
                ready <= 0;
                OP_ctrl <= 0;
                counter <= 0;
        end
        else if(run) begin
                if(counter < 64)begin
                        counter <= counter + 1;
                        //step1: Remainder minus divisor
                        if(counter % 2 == 0)begin
                                w_ctrl <= 1;
                                OP_ctrl <= `sub;
                                SLL_ctrl <= 0;
                        end
                        //step2: check the value of remainder
                        else if(counter % 2 == 1)begin
                                SLL_ctrl <= 1;
                                //if is negative -> MSB = 1 -> add divisor back and shift
                                if(MSB)begin
                                        w_ctrl <= 1;
                                        OP_ctrl <= `add;
                                end
                                //if positive -> keep it and shift
                                else begin
                                        w_ctrl <= 0;
                                        OP_ctrl <= 0;
                                end
                        end
                //the last step: shift the remainder right
                end else if(counter == 64)begin
                        SRL_ctrl <= 1;
                        SLL_ctrl <= 0;
                        counter <= counter +1;
                end else ready <= 1;
        end
end
endmodule
```

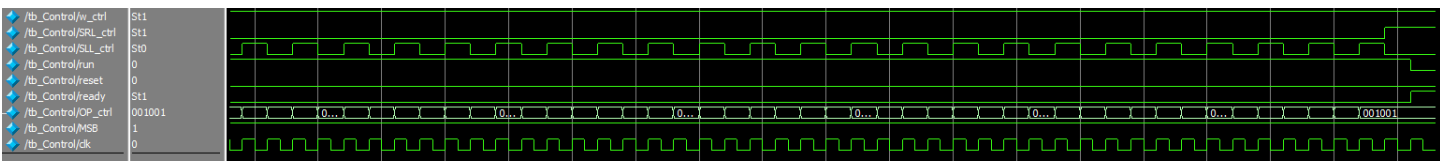
- Control test bench

```
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 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
```

```
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

```
module Divisor(Divisor_in, rst, w_ctrl, Divisor_out);
input rst, w_ctrl;
input [31:0]Divisor_in;
output reg [31:0]Divisor_out;

always@(*)begin                         //any change of input will action
        if(rst)begin                    //reset the output to zero
                Divisor_out <= 0;
        end else if(w_ctrl)begin        //if write control on -> output = input
                Divisor_out <= Divisor_in;
        end
end

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

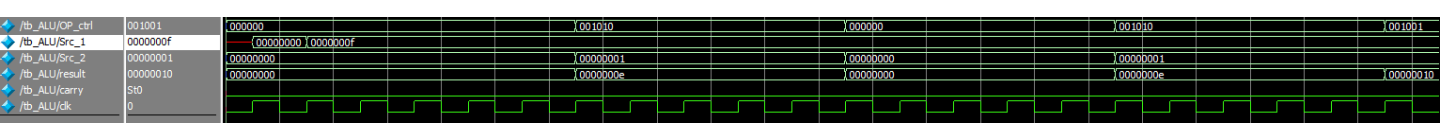
- ALU Module

```
`define addu    6'b001001
`define subu    6'b001010

module ALU(Src_1, Src_2, OP_ctrl, ALU_Result, ALU_Carry);
//inputs
input [31:0]Src_1, Src_2;
input [5:0]OP_ctrl;

//outputs
output reg [31:0]ALU_Result;
output reg ALU_Carry;

always@(*)begin
        //check the operation code
        case(OP_ctrl)
                `addu:  //perform add unsigned operation
                        {ALU_Carry, ALU_Result} <= Src_1 + Src_2;
                `subu:  //perform sub unsigned opoeration
                        {ALU_Carry, ALU_Result} <= Src_1 - Src_2;
                default:
                        {ALU_Carry, ALU_Result} <= 0;
        endcase

end
endmodule
```

- ALU test bench

```
// Setting timescale
`timescale 10 ns / 1 ns
`define sub     6'b001010
`define add     6'b001001
module tb_ALU;

        // Inputs
        reg [5:0]OP_ctrl = 0;
        reg [31:0]Src_1, Src_2 = 32'b0;

        // Outputs
        wire [31:0] result;
        wire carry;

        // Clock
        reg clk = 1'b0;

        // Instantiate the Unit Under Test (UUT)
        ALU UUT(
                .Src_1(Src_1),
                .Src_2(Src_2),
                .OP_ctrl(OP_ctrl),
                .ALU_Result(result),
                .ALU_Carry(carry)
        );

        // Generate Clock
        always
        begin : ClockGenerator
                #1 clk <= ~clk;         // toggle clock signal evergy one timescale delay
        end

        initial begin
                // Wait positive edge of clock signal
                @(posedge clk);

                // Reset UUT
                Src_1 <= 0;
                Src_2 <= 0;
                OP_ctrl <= 0;

                //assign value to input sources
                @(posedge clk);
                Src_1 <= 32'h0000_000f;

                #10;
                Src_2 <= 32'h1;
                OP_ctrl <= `sub;

                //Wait 10ns, change inputs
                #10;
                Src_2 <= 32'h0;
                OP_ctrl <= 0;

                //Wait 10ns, change inputs
                #10;
                Src_2 <= 32'h1;
                OP_ctrl <= `sub;
                #10;
                OP_ctrl <= `add;

                // Wait some time
                #2;

                // Stop the simulation
                $stop();
        end

endmodule
```

- ALU Simulation



- Remainder Module

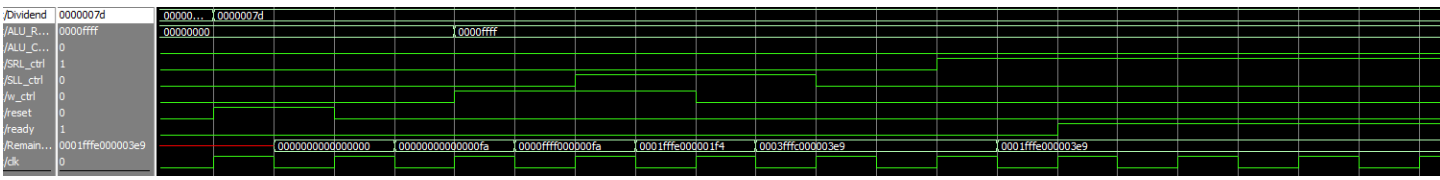
```
module Remainder(Dividend_in, ALU_Carry, ALU_Result, SLL_ctrl, SRL_ctrl, w_ctrl, ready, rst, clk, Remainder_out);
//inputs
input [31:0]Dividend_in, ALU_Result;
input ALU_Carry, SLL_ctrl, SRL_ctrl, w_ctrl, ready, rst, clk;
//output
output reg [63:0]Remainder_out;
reg [63:0]tempReg;
//the value to check if the devidend loaded
reg loaded;

always@(negedge clk)begin
        //reset the product register
        if(rst)begin
                Remainder_out <= 0;
                tempReg <= 0;
                loaded <= 0;
        end else begin
                if(~ready)begin
                        if(~loaded)begin
                                //has not load multiplier yet -> put dividend to the rightmost and shift left 1
                                tempReg[32:1] = Dividend_in;
                                Remainder_out = tempReg;
                        end
                        else begin
                                //step2 of the controller
                                if(SLL_ctrl)begin
                                        //if MSB = 1 -> w_ctrl = 1 -> add the divisor back to the leftmost and shift
                                        if(w_ctrl)begin
                                                tempReg[63:32] = ALU_Result;
                                                tempReg = tempReg << 1;
                                                tempReg[0] = 1'b0;      //fill right most bit with 0
                                                Remainder_out = tempReg;
                                        end
                                        //if MSB != 1 -> w_ctrl = 0 -> move and shift
                                        else begin
                                                tempReg = tempReg << 1;
                                                tempReg[0] = 1'b1;      //fill right most bit with 1
                                                Remainder_out = tempReg;
                                        end
                                end
                                //step1 of the controller
                                //no shift left
                                else begin
                                        if(SRL_ctrl)begin
                                                Remainder_out[62:32] = tempReg[63:33];
                                                Remainder_out[63] = 1'b0;
                                        end
                                        else if(w_ctrl)begin
                                                tempReg[63:32] = ALU_Result;
                                                Remainder_out = tempReg;
                                        end
                                end
                        end
                        loaded = 1;
                //if the loop is done -> ready = 1 -> output stays the same
                end else Remainder_out <= Remainder_out;
        end
end
endmodule
```

- Remainder test bench

```
// Setting timescale
`timescale 10 ns / 1 ns

`define DELAY                   1       // # * timescale

// Declarations
`define LOW     1'b0
`define HIGH    1'b1

module tb_Product;
        //inputs
        reg [31:0]Dividend = 32'b0;
        reg [31:0]ALU_Result = 32'b0;
        reg ALU_Carry = 0;
        reg SRL_ctrl = `LOW;
        reg SLL_ctrl = `LOW;
        reg w_ctrl = `LOW;
        reg reset = `LOW;
        reg ready = `LOW;
        //output
        wire [63:0]Remainder;

        //clock
        reg clk = `LOW;

        // Instantiate the Unit Under Test (UUT)
        Remainder UUT(
                //inputs
                .Dividend_in(Dividend),
                .ALU_Carry(ALU_Carry),
                .ALU_Result(ALU_Result),
                .SLL_ctrl(SLL_ctrl),
                .SRL_ctrl(SRL_ctrl),
                .w_ctrl(w_ctrl),
                .ready(ready),
                .rst(reset),
                .clk(clk),
                //output
                .Remainder_out(Remainder)
        );

        // Generate Clock
        always
        begin : ClockGenerator
                #1 clk <= ~clk;
        end
```

```
        initial begin
                // Wait positive edge of clock signal
                @(posedge clk);

                // Reset UUT
                reset <= `HIGH;
                ready <= `LOW;
                SLL_ctrl <= `LOW;
                SRL_ctrl <= `LOW;
                w_ctrl <= `LOW;
                Dividend <= 32'd125;
                // Wait positive edge of clock signal and end reset
                @(posedge clk);
                reset = `LOW;

                @(posedge clk);
                // Write data into Product Register
                w_ctrl <= `HIGH;
                ALU_Carry <= 0;
                ALU_Result <= 32'h0000_FFFF;

                //Wait positive edge of clk and SLL on
                @(posedge clk);
                SLL_ctrl <= `HIGH;

                //Wait positive edge of clk w_ctrl off
                @(posedge clk);
                w_ctrl <= `LOW;

                //Wait positive edge of clk and SLL off
                @(posedge clk);
                SLL_ctrl <= `LOW;

                //Wait positive edge of clk and SRL on
                @(posedge clk);
                SRL_ctrl <= `HIGH;

                //Wait positive edge of clk and ready
                @(posedge clk);
                ready <= `HIGH;
                // Wait some time
                #10;

                // Stop the simulation
                $stop();
        end

endmodule
```

- Remainder Simulation



- 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

```
module CompDivider(Quotient, Remainder, ready, Dividend, Divisor, run, reset, clk);
//outputs
output [31:0]Quotient, Remainder;
output ready;
//inputs
input [31:0]Dividend, Divisor;
input run, reset, clk;
//wires between the modules
wire [63:0]Remainder_out;
wire [31:0]Divisor_out, ALU_Result;
wire [5:0]OP_ctrl;
wire ALU_Carry, w_ctrl, SLL_ctrl, SRL_ctrl;

//instantiate ALU
ALU ALUnit(
        .Src_1(Remainder_out[63:32]),
        .Src_2(Divisor_out),
        .OP_ctrl(OP_ctrl),
        .ALU_Result(ALU_Result),
        .ALU_Carry(ALU_Carry)
);
//instantiate Divisor register
Divisor DivisorReg(
        .Divisor_in(Divisor),
        .rst(reset),
        .w_ctrl(w_ctrl),
        .Divisor_out(Divisor_out)
);
//instantiate Controller
Control controller(
        .run(run),
        .rst(reset),
        .clk(clk),
        .MSB(Remainder_out[63]),
        .w_ctrl(w_ctrl),
        .SLL_ctrl(SLL_ctrl),
        .SRL_ctrl(SRL_ctrl),
        .ready(ready),
        .OP_ctrl(OP_ctrl)
);
//instantiate Remainder register
Remainder RemainderReg(
        .Dividend_in(Dividend),
        .ALU_Carry(ALU_Carry),
        .ALU_Result(ALU_Result),
        .SLL_ctrl(SLL_ctrl),
        .SRL_ctrl(SRL_ctrl),
        .w_ctrl(w_ctrl),
        .ready(ready),
        .rst(reset),
        .clk(clk),
        .Remainder_out(Remainder_out)
);
//assign the value to remainder & Quotient by the value in 64-bit Remainder register
assign Remainder = Remainder_out[63:32];
assign Quotient = Remainder_out[31:0];
endmodule
```

- CompDivider testbench

```
// Setting timescale
`timescale 10 ns / 1 ns

// Declarations
`define DELAY                   1       // # * timescale
`define INPUT_FILE              "testbench/tb_CompDivider.in"
`define OUTPUT_FILE             "testbench/tb_CompDivider.out"

// Declaration
`define LOW             1'b0
`define HIGH    1'b1

module tb_CompDivider;

        // Inputs
        reg Reset;
        reg Run;
        reg [31:0] Dividend_in;
        reg [31:0] Divisor_in;

        // Outputs
        wire [31:0] Quotient_out;
        wire [31:0] Remainder_out;
        wire Ready;

        // Clock
        reg clk = `LOW;

        // Testbench variables
        reg [63:0] read_data;
        integer input_file;
        integer output_file;
        integer i;

        // Instantiate the Unit Under Test (UUT)
        CompDivider UUT(
                // Outputs
                .Quotient(Quotient_out),
                .Remainder(Remainder_out),
                .ready(Ready),
                // Inputs
                .Dividend(Dividend_in),
                .Divisor(Divisor_in),
                .run(Run),
                .reset(Reset),
                .clk(clk)
        );
```

```
        initial
        begin : Preprocess
                // Initialize inputs
                Reset           = `LOW;
                Run             = `LOW;
                Dividend_in     = 32'd0;
                Divisor_in      = 32'd0;

                // Initialize testbench files
                input_file      = $fopen(`INPUT_FILE, "r");
                output_file     = $fopen(`OUTPUT_FILE);

                #`DELAY;        // Wait for global reset to finish
        end

        always
        begin : ClockGenerator
                #`DELAY;
                clk <= ~clk;
        end

        always
        begin : StimuliProcess
                // Start testing
                while (!$feof(input_file))
                begin
                        $fscanf(input_file, "%x\n", read_data);
                        @(posedge clk); // Wait clock
                        {Dividend_in, Divisor_in} = read_data;
                        Reset = `HIGH;
                        @(posedge clk); // Wait clock
                        Reset = `LOW;
                        @(posedge clk); // Wait clock
                        Run = `HIGH;
                        @(posedge Ready);       // Wait ready
                        Run = `LOW;
                end

                #`DELAY;        // Wait for result stable

                // Close output file for safety
                $fclose(output_file);

                // Stop the simulation
                $stop();
        end

        always @(posedge Ready)
        begin : Monitoring
                $display("Dividend_in:%d, Divisor_in:%d", Dividend_in, Divisor_in);
                $display("Quotient_out:%d, Remainder_out:%d", Quotient_out, Remainder_out);
                $fdisplay(output_file, "%x_%x", Quotient_out, Remainder_out);
        end

endmodule
```

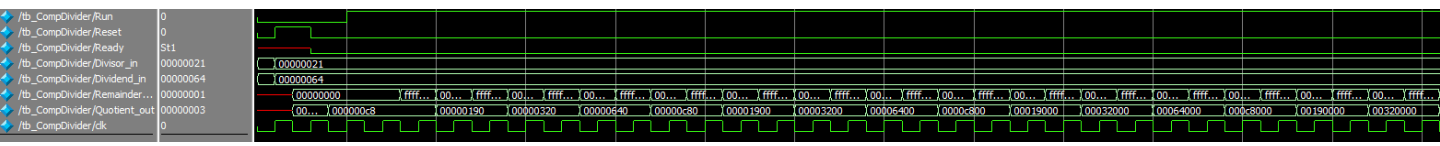
- CompDivider Simulation

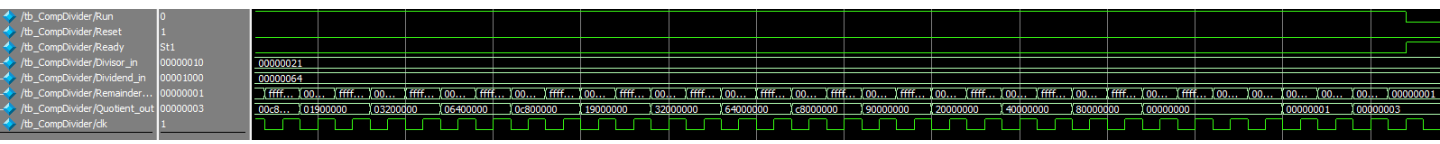
tb_CompDivider.in - 記事本

檔案(F) 編輯(E) 格式(O) 檢視(V)

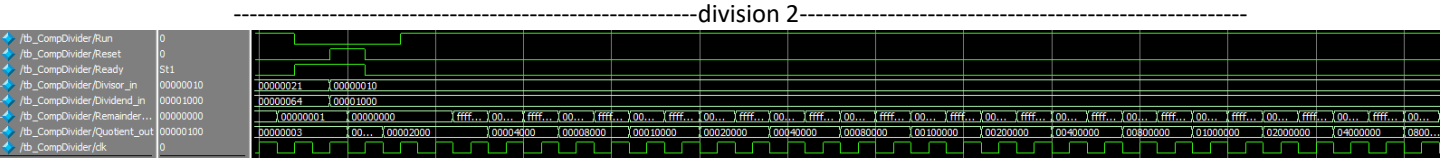
00000064_00000021
00001000_00000010

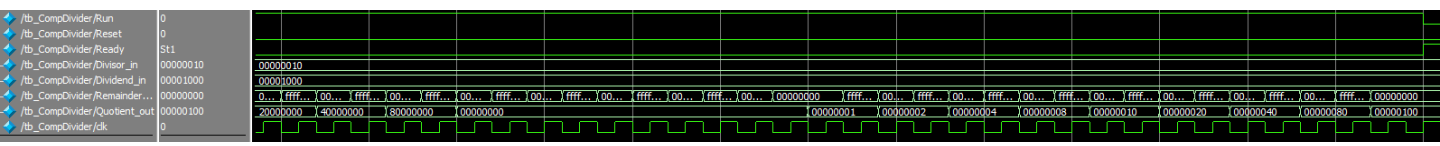
->.in file





----------division 2----------





- 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。