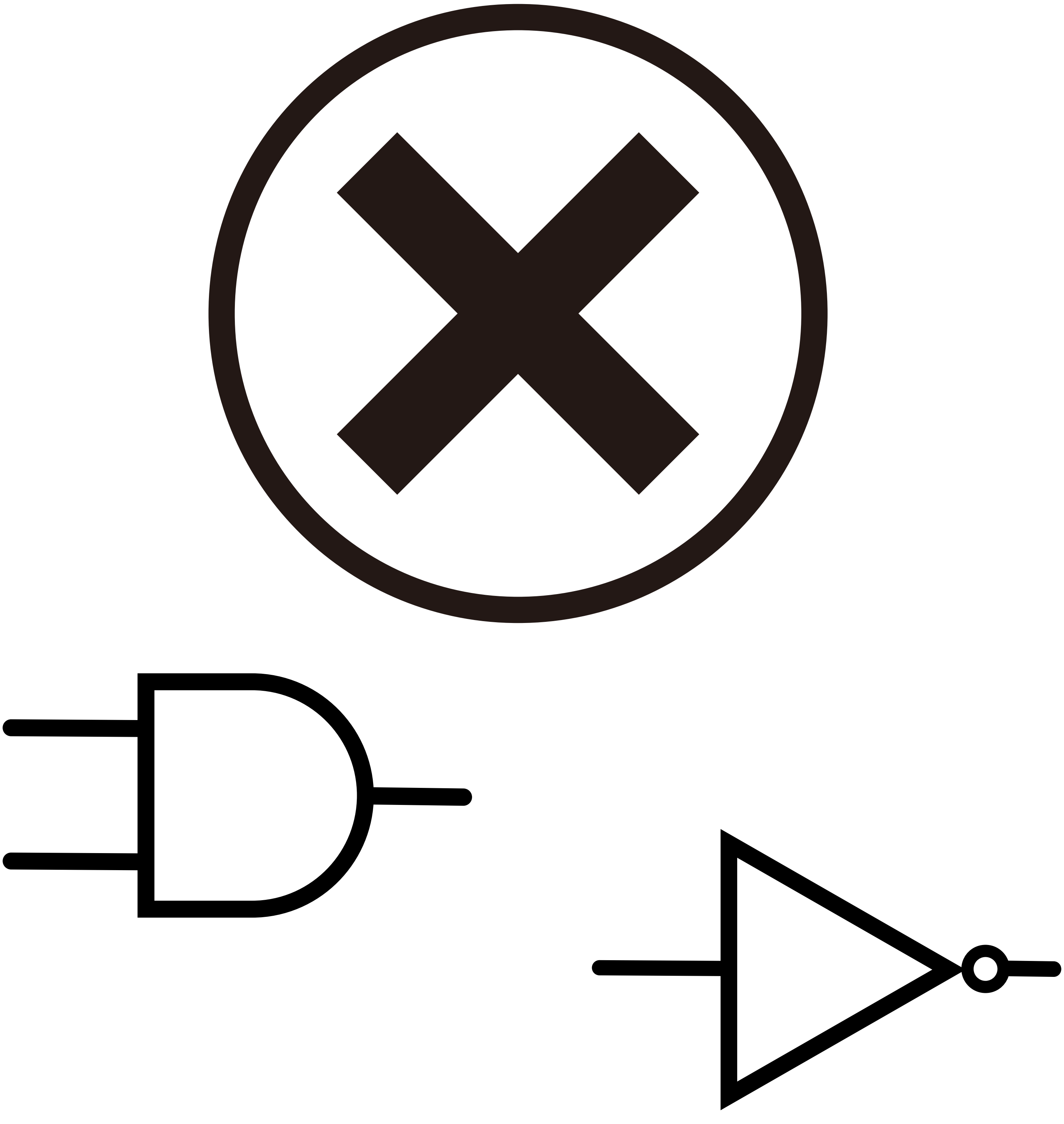
PA1
Unsigned Multiplier & Unsigned Divider

四電機三乙

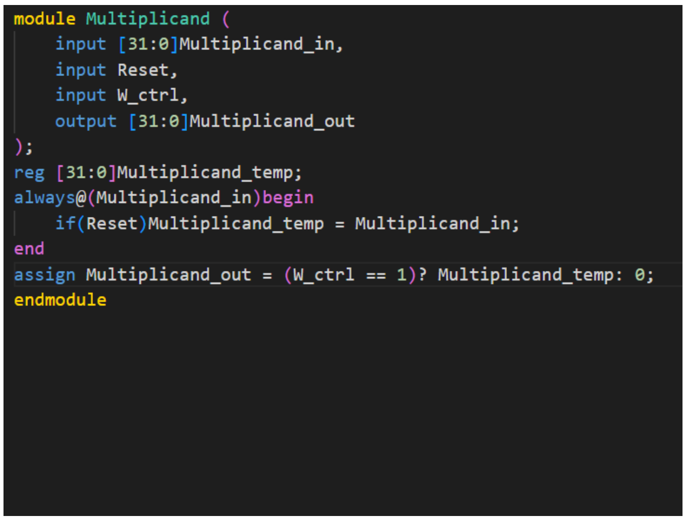
B10932010 游兆暄

Part 1
Unsigned Multiplier

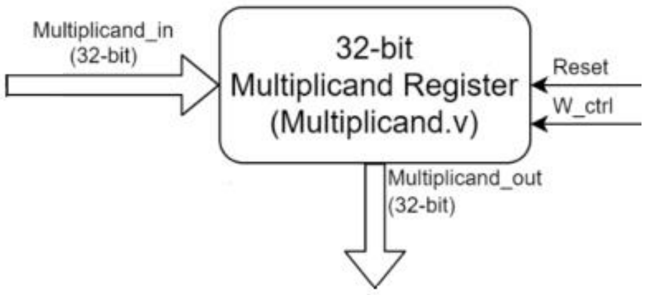


Multiplicand、ALU、Control、Product、CompMultiplier

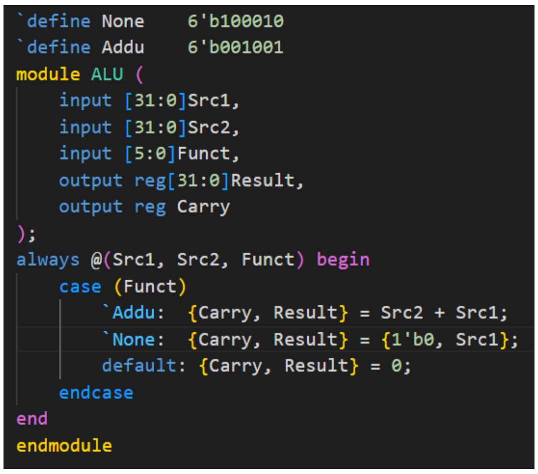
(Screenshots of each program, and description of the process.)



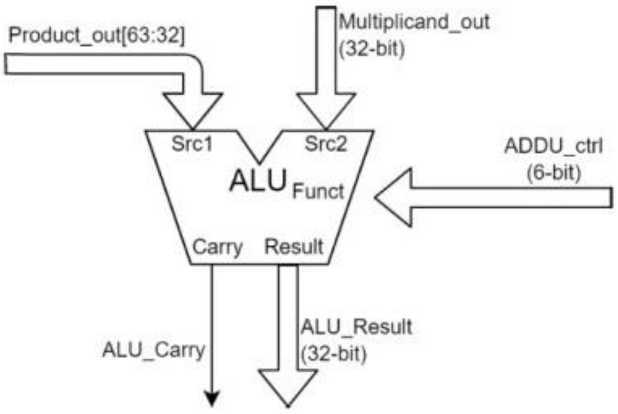
```
module Multiplicand (
    input [31:0]Multiplicand_in,
    input Reset,
    input W_ctrl,
    output [31:0]Multiplicand_out
);
reg [31:0]Multiplicand_temp;
always@(Multiplicand_in)begin
    if(Reset)Multiplicand_temp = Multiplicand_in;
end
assign Multiplicand_out = (W_ctrl == 1)? Multiplicand_temp: 0;
endmodule
```



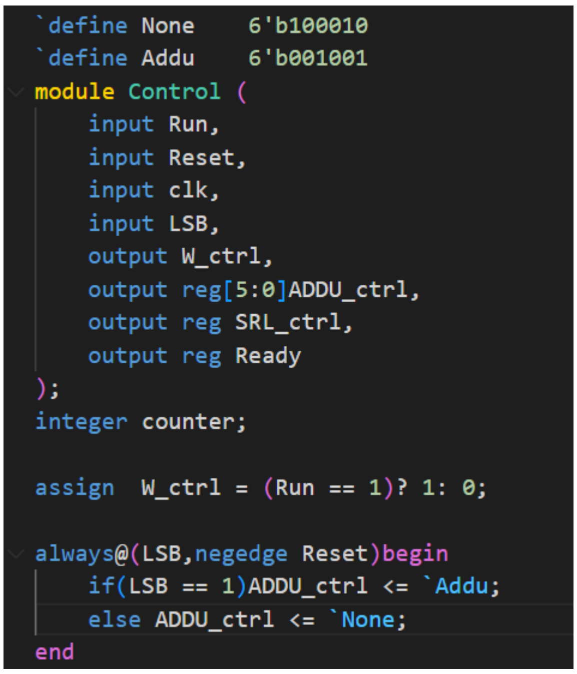
有一個 32bit 的 Multiplicand_in、Reset、W_ctrl，當有新的 Multiplicand_in，但是 W_ctrl 為 1，不會將值傳進 Multiplicand_out，而是暫存在 Multiplicand_temp 中。當 W_ctrl 為 1 時，Multiplicand_out 才會更新為新的值，否則就維持原狀。



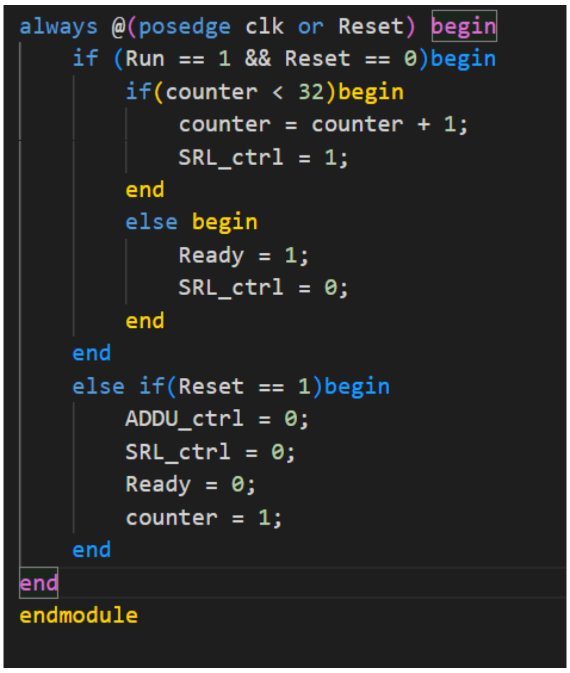
```
`define None    6'b100010
`define Addu    6'b001001
module ALU (
    input [31:0]Src1,
    input [31:0]Src2,
    input [5:0]Funct,
    output reg[31:0]Result,
    output reg Carry
);
always @(Src1, Src2, Funct) begin
    case (Funct)
        `Addu:  {Carry, Result} = Src2 + Src1;
        `None:  {Carry, Result} = {1'b0, Src1};
        default: {Carry, Result} = 0;
    endcase
end
endmodule
```



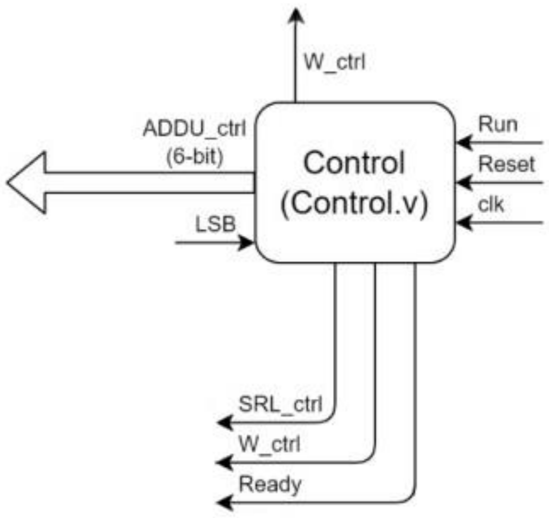
定義每個功能的 Funct，將 Multiplicand_Register 所取到的資料 Multiplicand_out 以及 Product_out 傳到 ALU 中並判斷 Funct 進行相對應的運算，若有進位則會使 Carry 變為 1，最後將運算完的資料傳到 Result。若 Funct 為 None，則將 Carry 設為 0，Src1 的值不做運算直接輸出到 Result。若 Funct 為 0，Carry 及 Result 會歸零。



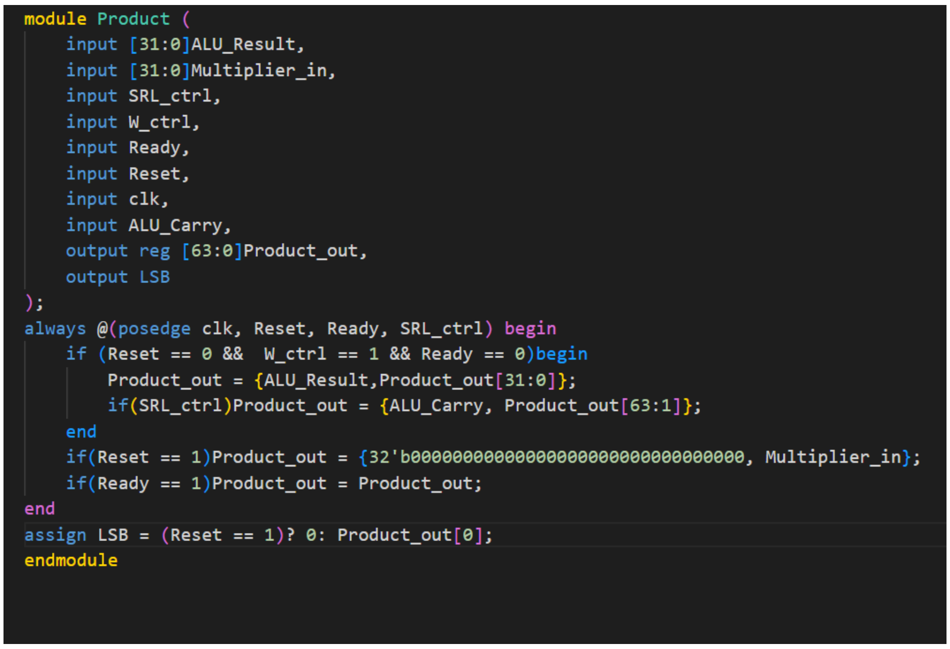
```
`define None    6'b100010
`define Addu    6'b001001
module Control (
    input Run,
    input Reset,
    input clk,
    input LSB,
    output W_ctrl,
    output reg[5:0]ADDU_ctrl,
    output reg SRL_ctrl,
    output reg Ready
);
integer counter;

assign  W_ctrl = (Run == 1)? 1: 0;

always@(LSB,negedge Reset)begin
    if(LSB == 1)ADDU_ctrl <= `Addu;
    else ADDU_ctrl <= `None;
end
```



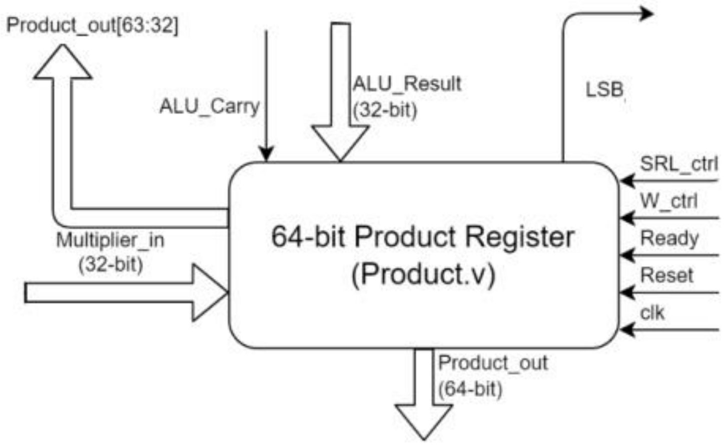
```
always @(posedge clk or Reset) begin
    if (Run == 1 && Reset == 0)begin
        if(counter < 32)begin
            counter = counter + 1;
            SRL_ctrl = 1;
        end
        else begin
            Ready = 1;
            SRL_ctrl = 0;
        end
    end
    else if(Reset == 1)begin
        ADDU_ctrl = 0;
        SRL_ctrl = 0;
        Ready = 0;
        counter = 1;
    end
end
endmodule
```



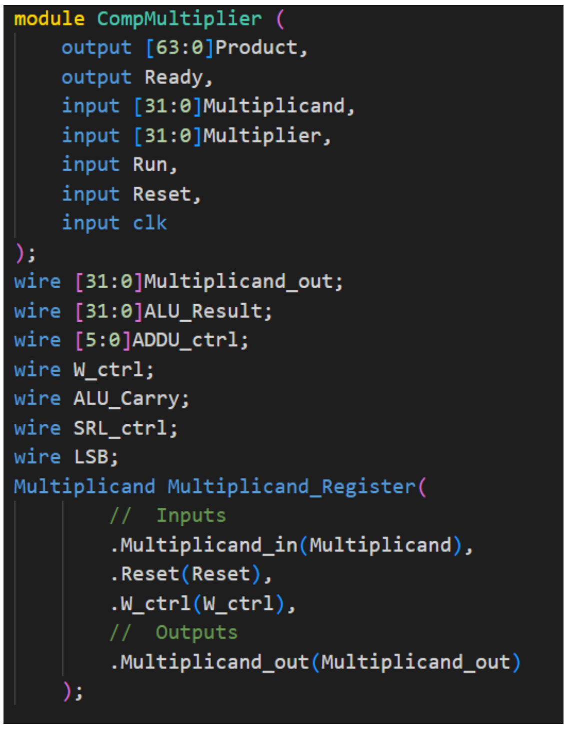
定義每個功能的 ADDU_ctrl，當 Run 為一時，將 W_ctrl 設為 1，反之為 0。另外判斷 LSB 的值來決定 ADDU_ctrl 要執行哪種指令，若為 1 就執行加法，若不是則讓值直接通過(None)。在 Reset 為 1 時，將所有值歸零，counter 恢復為 1。另外若 Run 為 1 且 Reset 為 0 時，代表還在計算階段，因此計數器 counter 每經過一個正緣 clk 就加 1，另外，因為計算階段需要持續地向右位移，SRL_ctrl 也為 1。當計數器 counter 為 32 時代表計算結束，此時將 SRL_ctrl 改為 0，以及 Ready 改為 1，表示計算結束，避免其他元件繼續運作影響到答案。



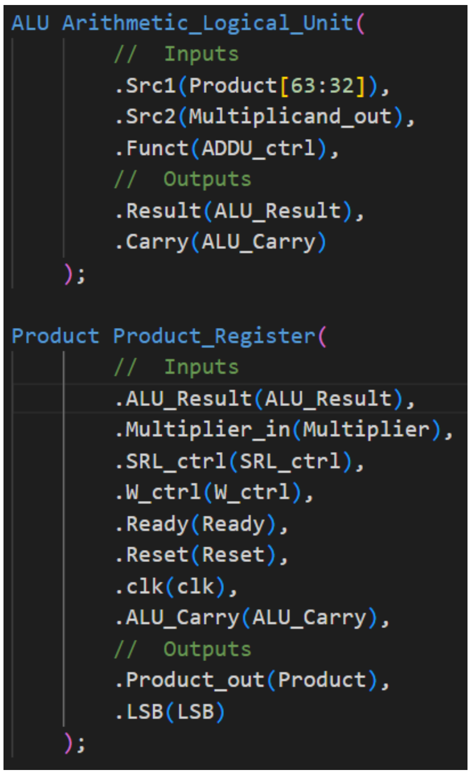
```
module Product (
    input [31:0]ALU_Result,
    input [31:0]Multiplier_in,
    input SRL_ctrl,
    input W_ctrl,
    input Ready,
    input Reset,
    input clk,
    input ALU_Carry,
    output reg [63:0]Product_out,
    output LSB
);
always @(posedge clk, Reset, Ready, SRL_ctrl) begin
    if (Reset == 0 &&  W_ctrl == 1 && Ready == 0)begin
        Product_out = {ALU_Result,Product_out[31:0]};
        if(SRL_ctrl)Product_out = {ALU_Carry, Product_out[63:1]};
    end
    if(Reset == 1)Product_out = {32'b00000000000000000000000000000000, Multiplier_in};
    if(Ready == 1)Product_out = Product_out;
end
assign LSB = (Reset == 1)? 0: Product_out[0];
endmodule
```



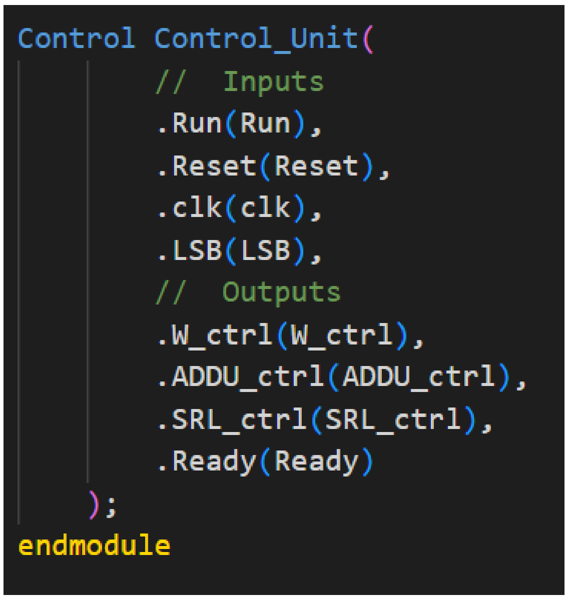
在 Reset 為 1 時，將 Multiplier_in 的值放進 Product_out 的右半 32bit，左半 32bit 歸零。當 Reset、Ready 為 0 且 W_ctrl 為 1 時，表示正在計算階段，因此每經過一個正緣 clk，就將 ALU 的計算結果 ALU_Result 放進 Product_out，接下來判斷 SRL_ctrl 是否為 1，若是，則將 ALU_Carry 放進向右位移 1bit 後的 Product_out 的最左邊 1 個 bit。當 Ready 為 1，表示計算結束，因此維持 Product_out 避免答案出錯。



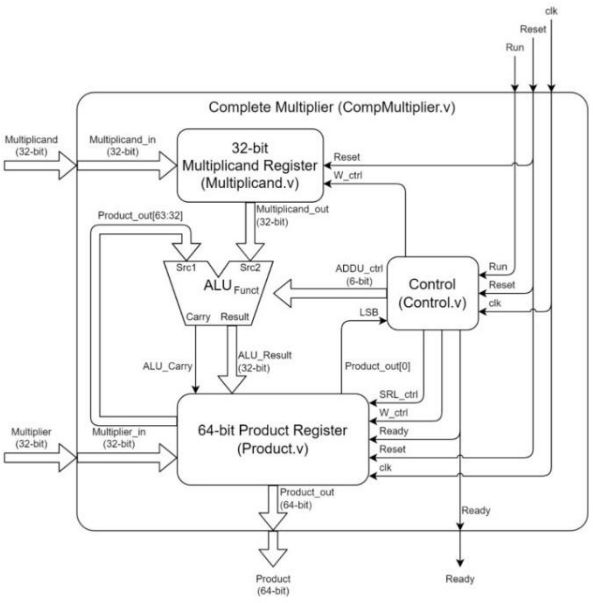
```verilog
module CompMultiplier (
    output [63:0]Product,
    output Ready,
    input [31:0]Multiplicand,
    input [31:0]Multiplier,
    input Run,
    input Reset,
    input clk
);
wire [31:0]Multiplicand_out;
wire [31:0]ALU_Result;
wire [5:0]ADDU_ctrl;
wire W_ctrl;
wire ALU_Carry;
wire SRL_ctrl;
wire LSB;
Multiplicand Multiplicand_Register(
        //  Inputs
        .Multiplicand_in(Multiplicand),
        .Reset(Reset),
        .W_ctrl(W_ctrl),
        //  Outputs
        .Multiplicand_out(Multiplicand_out)
    );
```



```verilog
ALU Arithmetic_Logical_Unit(
        //  Inputs
        .Src1(Product[63:32]),
        .Src2(Multiplicand_out),
        .Funct(ADDU_ctrl),
        //  Outputs
        .Result(ALU_Result),
        .Carry(ALU_Carry)
    );

Product Product_Register(
        //  Inputs
        .ALU_Result(ALU_Result),
        .Multiplier_in(Multiplier),
        .SRL_ctrl(SRL_ctrl),
        .W_ctrl(W_ctrl),
        .Ready(Ready),
        .Reset(Reset),
        .clk(clk),
        .ALU_Carry(ALU_Carry),
        //  Outputs
        .Product_out(Product),
        .LSB(LSB)
    );
```



```verilog
Control Control_Unit(
        //  Inputs
        .Run(Run),
        .Reset(Reset),
        .clk(clk),
        .LSB(LSB),
        //  Outputs
        .W_ctrl(W_ctrl),
        .ADDU_ctrl(ADDU_ctrl),
        .SRL_ctrl(SRL_ctrl),
        .Ready(Ready)
    );
endmodule
```

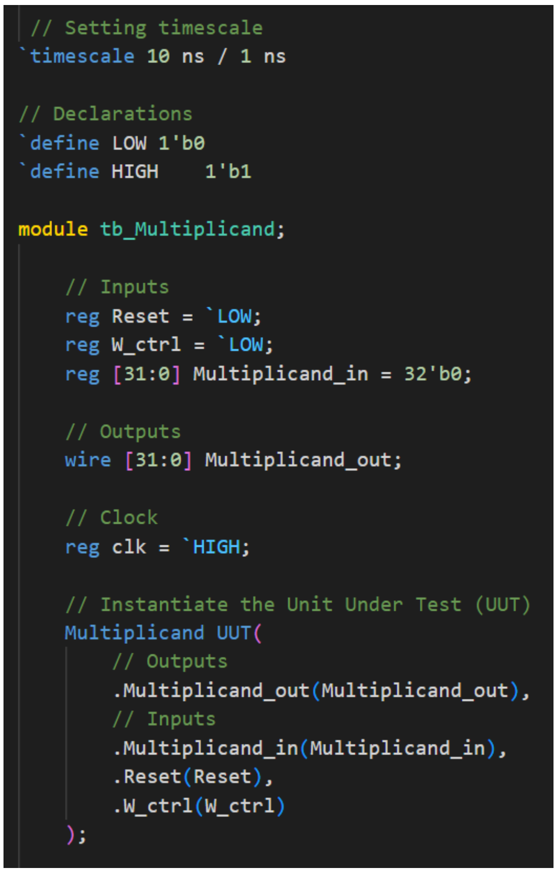


將 Multiplicand、ALU、Control、Product 串在一起成為一個大模組，按照途中的接線將相對應的輸出輸入接好，另外用 Multiplicand_out、ALU_Result、ADDU_ctrl、W_ctrl、ALU_Carry、SRL_ctrl、LSB 內部接線串接除法器內各元件。需要注意的是 ALU 的 Product_out 要輸入的應該是 Product 的左半 32bit 進去做運算。

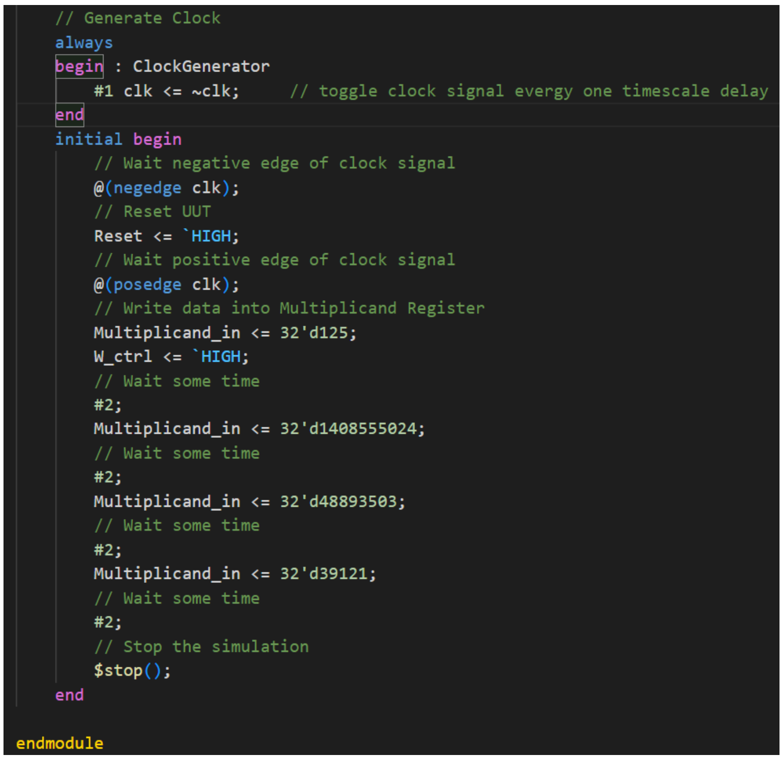
Multiplicand、ALU、Control、Product、CompMultiplier

(Screenshots of each testbench, and description of the test functions.)

Multiplicand：



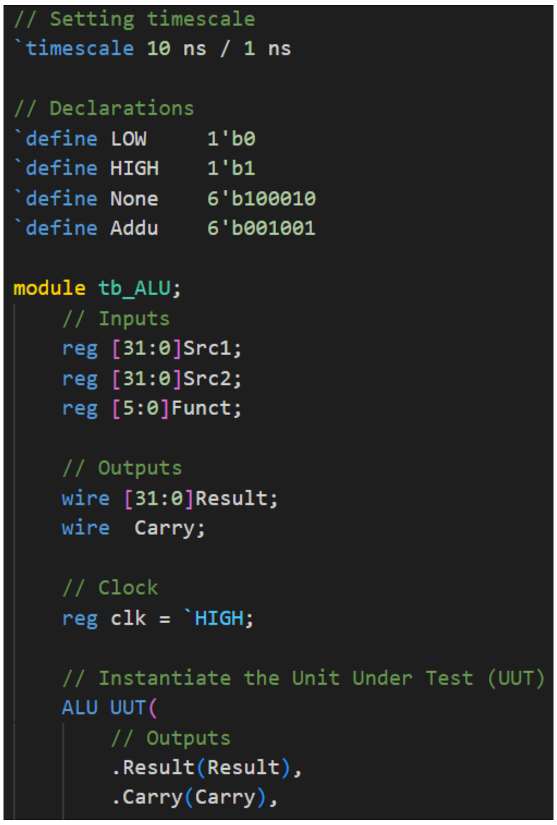
```verilog
// Setting timescale
`timescale 10 ns / 1 ns

// 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 = `HIGH;

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



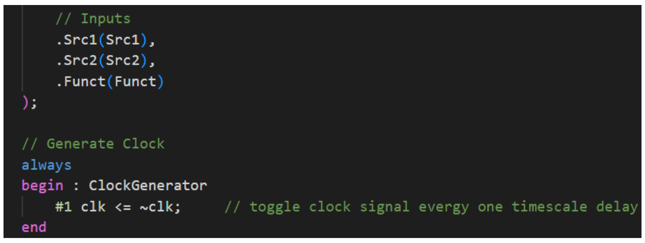
```verilog
    // Generate Clock
    always
    begin : ClockGenerator
        #1 clk <= ~clk;     // toggle clock signal evergy one timescale delay
    end
    initial begin
        // Wait negative edge of clock signal
        @(negedge clk);
        // Reset UUT
        Reset <= `HIGH;
        // Wait positive edge of clock signal
        @(posedge clk);
        // Write data into Multiplicand Register
        Multiplicand_in <= 32'd125;
        W_ctrl <= `HIGH;
        // Wait some time
        #2;
        Multiplicand_in <= 32'd1408555024;
        // Wait some time
        #2;
        Multiplicand_in <= 32'd48893503;
        // Wait some time
        #2;
        Multiplicand_in <= 32'd39121;
        // Wait some time
        #2;
        // Stop the simulation
        $stop();
    end

endmodule
```

程式中先測試了 Reset 為 1 但是 W_ctrl 不為 1 時，Multiplicand_out 是否維持原值。再測試 Reset 和 W_ctrl 都為 1 時，是否 Multiplicand_out 也會隨著 Multiplicand_in 一起變動。

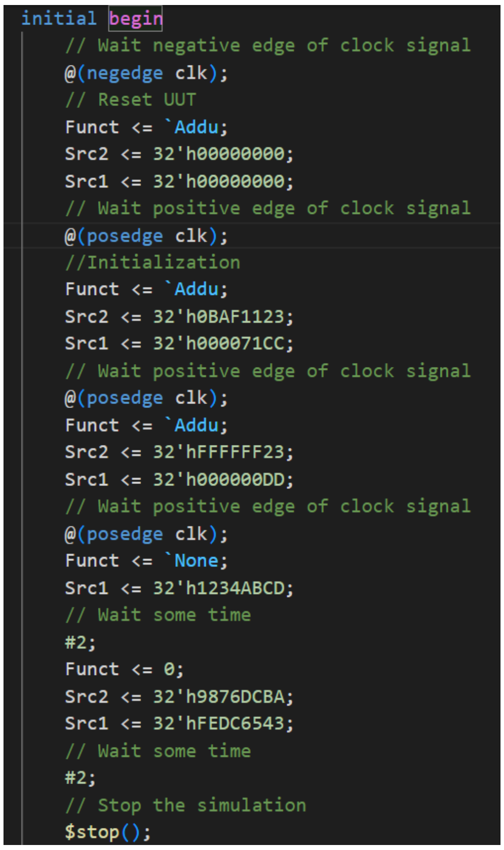
ALU：



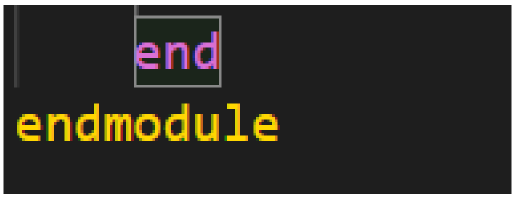
```verilog
// Setting timescale
`timescale 10 ns / 1 ns

// Declarations
`define LOW     1'b0
`define HIGH    1'b1
`define None    6'b100010
`define Addu    6'b001001

module tb_ALU;
    // Inputs
    reg [31:0]Src1;
    reg [31:0]Src2;
    reg [5:0]Funct;

    // Outputs
    wire [31:0]Result;
    wire  Carry;

    // Clock
    reg clk = `HIGH;

    // Instantiate the Unit Under Test (UUT)
    ALU UUT(
        // Outputs
        .Result(Result),
        .Carry(Carry),
```



```verilog
        // Inputs
        .Src1(Src1),
        .Src2(Src2),
        .Funct(Funct)
    );

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



```verilog
initial begin
    // Wait negative edge of clock signal
    @(negedge clk);
    // Reset UUT
    Funct <= `Addu;
    Src2 <= 32'h00000000;
    Src1 <= 32'h00000000;
    // Wait positive edge of clock signal
    @(posedge clk);
    //Initialization
    Funct <= `Addu;
    Src2 <= 32'h0BAF1123;
    Src1 <= 32'h000071CC;
    // Wait positive edge of clock signal
    @(posedge clk);
    Funct <= `Addu;
    Src2 <= 32'hFFFFFF23;
    Src1 <= 32'h000000DD;
    // Wait positive edge of clock signal
    @(posedge clk);
    Funct <= `None;
    Src1 <= 32'h1234ABCD;
    // Wait some time
    #2;
    Funct <= 0;
    Src2 <= 32'h9876DCBA;
    Src1 <= 32'hFEDC6543;
    // Wait some time
    #2;
    // Stop the simulation
    $stop();
```



```verilog
    end
endmodule
```

Case1:

Funct=001001(Addu)，Src2=32'h0BAF1123，Src1=32'h000071CC，可以測試加法，沒有進位，因此可以測試 Carry 是否正常。

Case2:

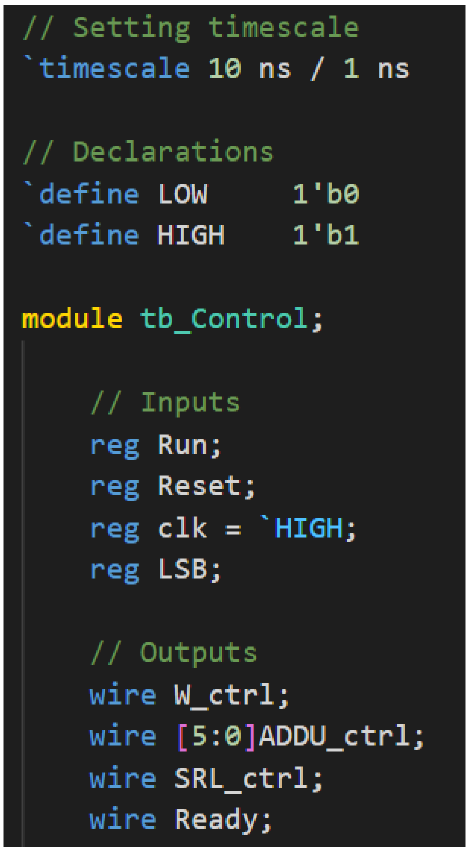
Funct =001001(Addu)，Src2=32'hFFFFFF23，Src1=32'h000000DD，可以測試加法，有進位，因此可以測試 Carry 是否正常。

Case3:

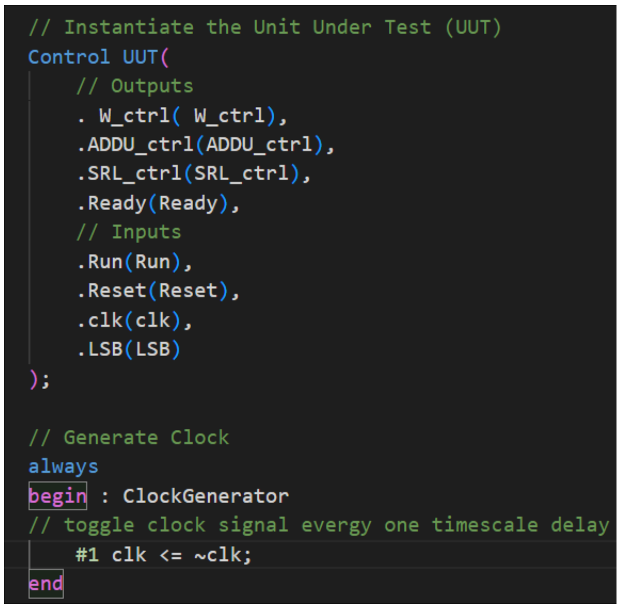
Funct =100010(None)，Src1=32'h1234ABCD，可以測試 Src1 中的值是否不做任何運算直接傳送到 Result。

Case4:
Funct =00，可以測試是否會將 Result 以及 Carry 都歸零。

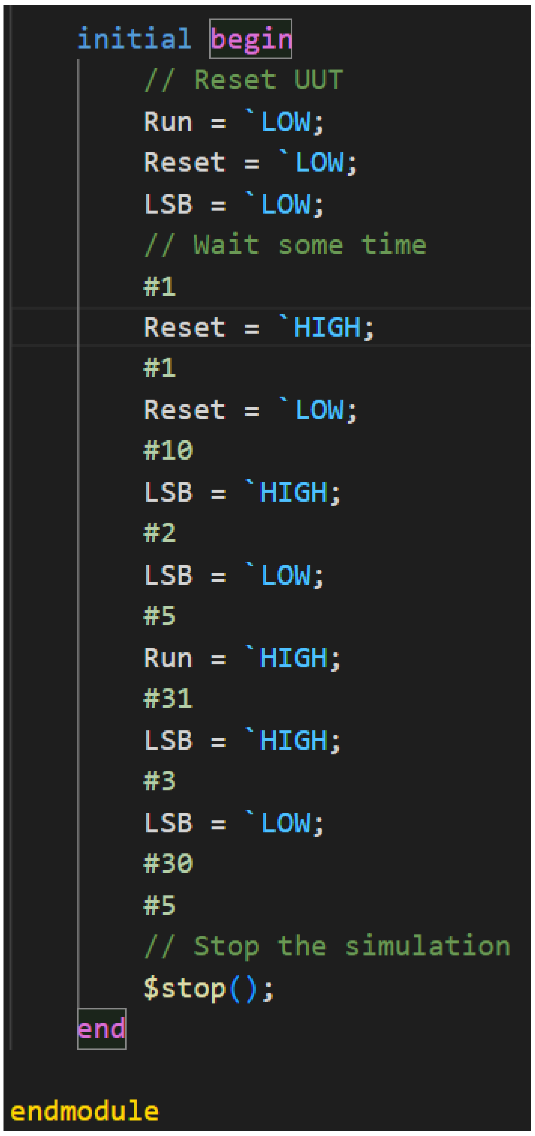
Control：



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

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

module tb_Control;

    // Inputs
    reg Run;
    reg Reset;
    reg clk = `HIGH;
    reg LSB;

    // Outputs
    wire W_ctrl;
    wire [5:0]ADDU_ctrl;
    wire SRL_ctrl;
    wire Ready;
```



```verilog
// Instantiate the Unit Under Test (UUT)
Control UUT(
    // Outputs
    . W_ctrl( W_ctrl),
    .ADDU_ctrl(ADDU_ctrl),
    .SRL_ctrl(SRL_ctrl),
    .Ready(Ready),
    // Inputs
    .Run(Run),
    .Reset(Reset),
    .clk(clk),
    .LSB(LSB)
);

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



```verilog
    initial begin
        // Reset UUT
        Run = `LOW;
        Reset = `LOW;
        LSB = `LOW;
        // Wait some time
        #1
        Reset = `HIGH;
        #1
        Reset = `LOW;
        #10
        LSB = `HIGH;
        #2
        LSB = `LOW;
        #5
        Run = `HIGH;
        #31
        LSB = `HIGH;
        #3
        LSB = `LOW;
        #30
        #5
        // Stop the simulation
        $stop();
    end

endmodule
```

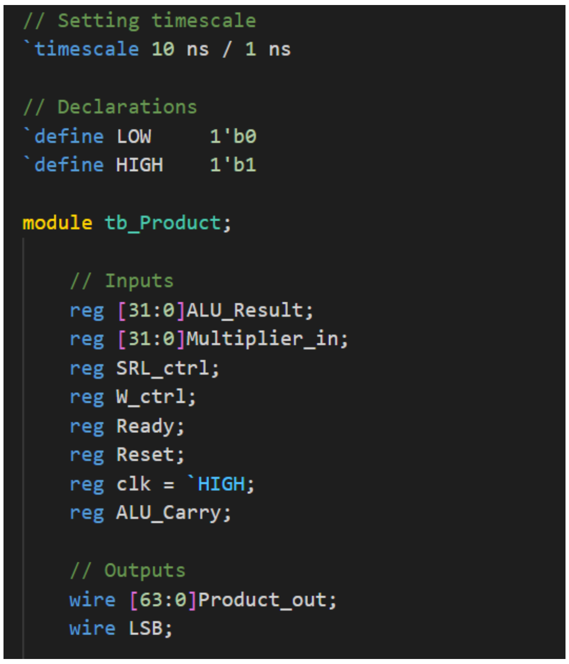
Case1:

Run = 0、Reset = 0、W_ctrl = 0 時，變動 LSB 的值，觀察 ADDU_ctrl 是否有跟著變動。另外因為 W_ctrl、Run 都為 0，因此可以觀察 counter 是否維持，不隨 clk 變動而增加。

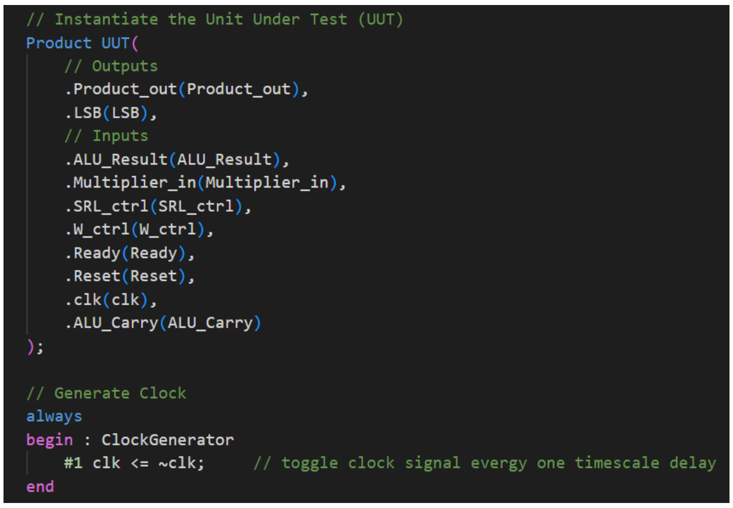
Case2:

Run =1、Reset = 0、W_ctrl = 1 時，變動 LSB 的值，觀察 ADDU_ctrl 是否有跟著變動。另外因為 W_ctrl、Run 都為 1，因此可以觀察 32 個 clk 之後，Ready 是否由 0 變為 1。也可以觀察 counter 是否隨 clk 增加。

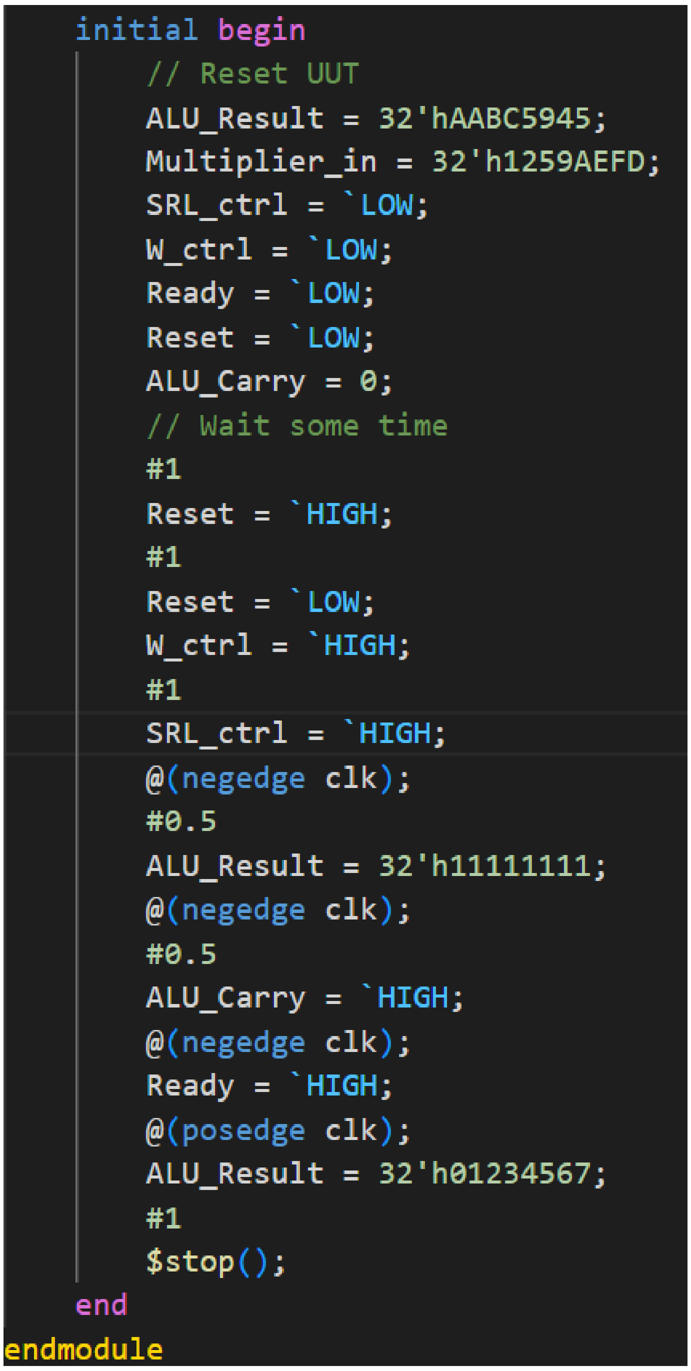
Product：



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

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

module tb_Product;

    // Inputs
    reg [31:0]ALU_Result;
    reg [31:0]Multiplier_in;
    reg SRL_ctrl;
    reg W_ctrl;
    reg Ready;
    reg Reset;
    reg clk = `HIGH;
    reg ALU_Carry;

    // Outputs
    wire [63:0]Product_out;
    wire LSB;
```



```
// Instantiate the Unit Under Test (UUT)
Product UUT(
    // Outputs
    .Product_out(Product_out),
    .LSB(LSB),
    // Inputs
    .ALU_Result(ALU_Result),
    .Multiplier_in(Multiplier_in),
    .SRL_ctrl(SRL_ctrl),
    .W_ctrl(W_ctrl),
    .Ready(Ready),
    .Reset(Reset),
    .clk(clk),
    .ALU_Carry(ALU_Carry)
);

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



```
    initial begin
        // Reset UUT
        ALU_Result = 32'hAABC5945;
        Multiplier_in = 32'h1259AEFD;
        SRL_ctrl = `LOW;
        W_ctrl = `LOW;
        Ready = `LOW;
        Reset = `LOW;
        ALU_Carry = 0;
        // Wait some time
        #1
        Reset = `HIGH;
        #1
        Reset = `LOW;
        W_ctrl = `HIGH;
        #1
        SRL_ctrl = `HIGH;
        @(negedge clk);
        #0.5
        ALU_Result = 32'h11111111;
        @(negedge clk);
        #0.5
        ALU_Carry = `HIGH;
        @(negedge clk);
        Ready = `HIGH;
        @(posedge clk);
        ALU_Result = 32'h01234567;
        #1
        $stop();
    end
endmodule
```

Case1:

ALU_Result = 32'hAABC5945，Multiplier_in = 32'h1259AEFD，Reset = 1、SRL_ctrl=0、W_ctrl=0、Ready=0、ALU_Carry=0 時，可以觀察 Product_out=是否將 Multiplier_in 放進右半 32bit。

Case2:

ALU_Result = 32'hAABC5945，Multiplier_in = 32'h1259AEFD，Reset = 0、SRL_ctrl=0、W_ctrl=1、Ready=0、ALU_Carry=0 時，可以觀察 Product_out=是否將 Multiplier_in 放進右半 32bit、ALU_Result 放進左半 32bit。

Case3:

ALU_Result = 32'h11111111，Multiplier_in = 32'h1259AEFD，Reset = 0、SRL_ctrl=1、W_ctrl=1、Ready=0、ALU_Carry=0 時，可以觀察 Product_out=是否將 Multiplier_in 放進右半 32bit、ALU_Result 放進左半 32bit 後，向右位移 1 bit。

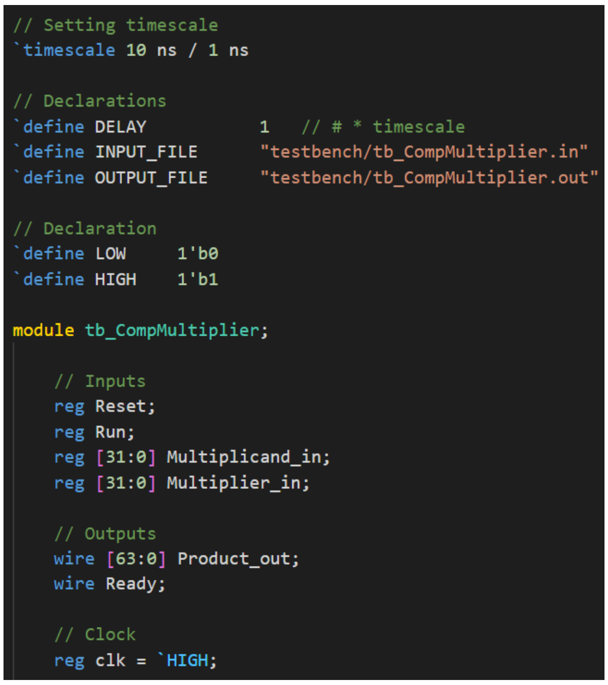
Case4:

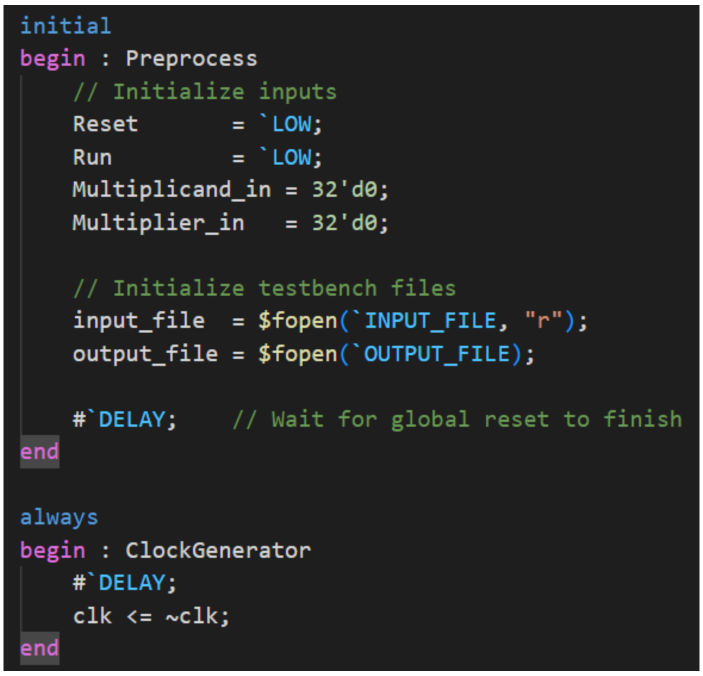
ALU_Result = 32'hAABC5945，Multiplier_in = 32'h1259AEFD，Reset = 0、SRL_ctrl=1、W_ctrl=1、Ready=0、ALU_Carry=1 時，可以觀察 Product_out=是否將 Multiplier_in 放進右半 32bit、ALU_Result 放進左半 32bit 後，向右位移 1 bit 並將 1 放進左邊第一個 bit。

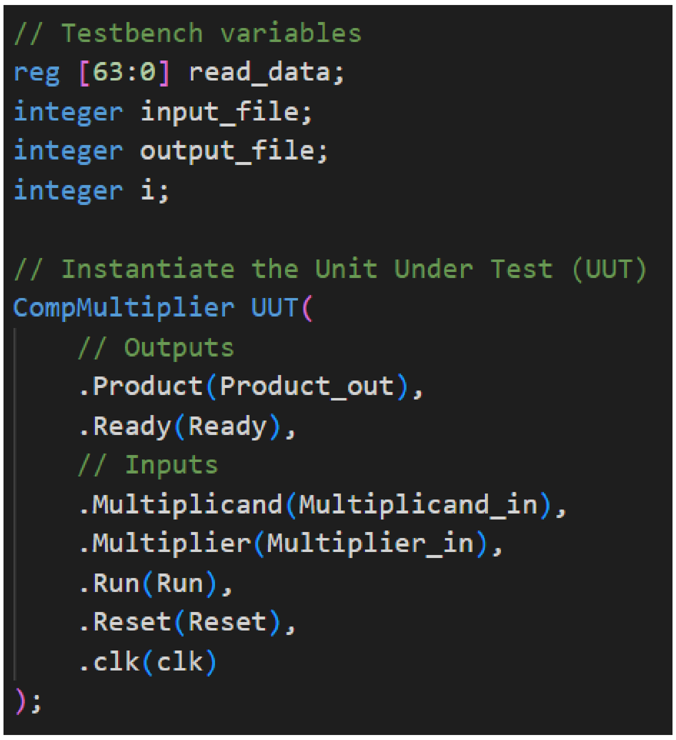
Case5:

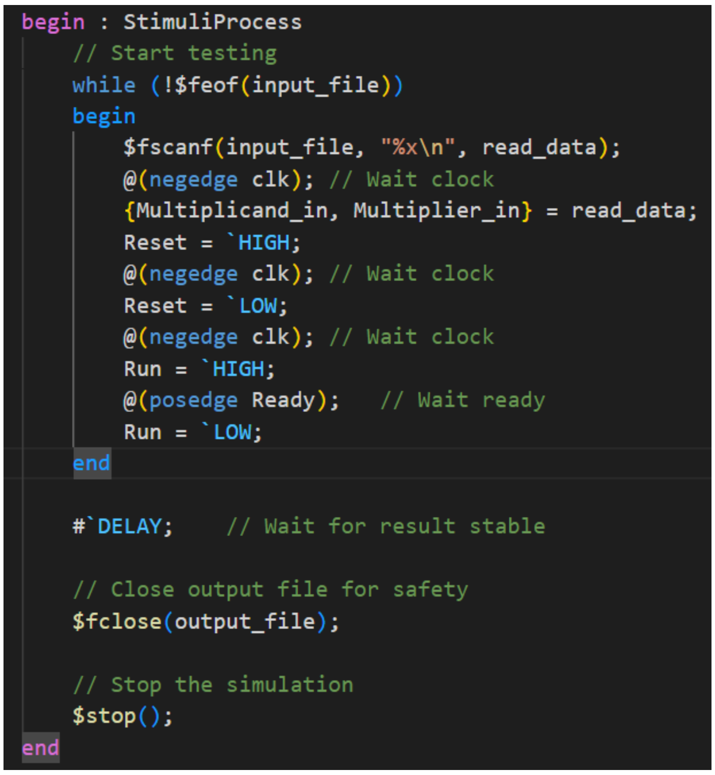
Ready = 1 時，Product_out 是否不受 ALU_Result 改變為 32'h01234567 所影響，繼續維持原本的值。

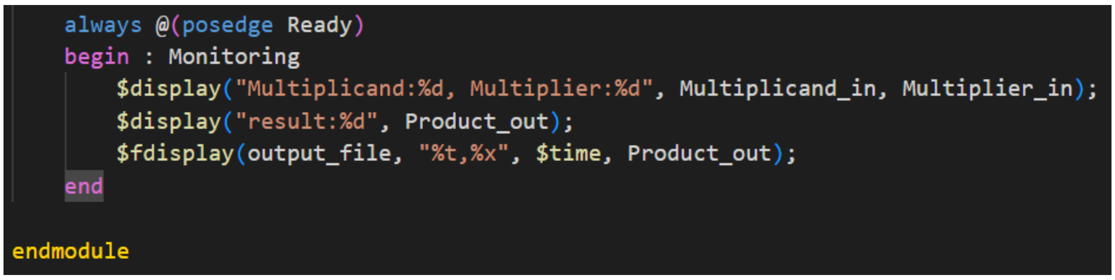
CompMultiplier：

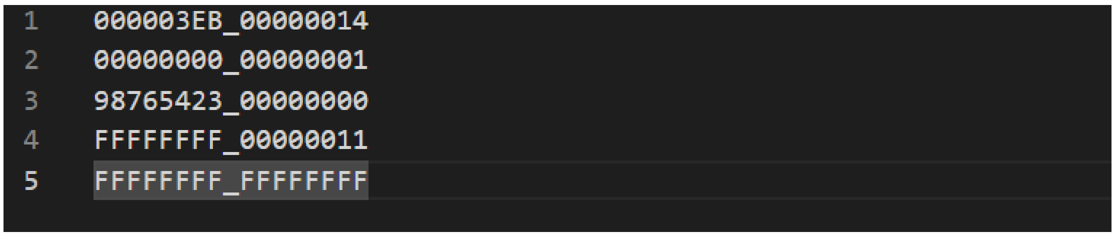

```verilog
// Setting timescale
`timescale 10 ns / 1 ns

// 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_in;
    reg [31:0] Multiplier_in;

    // Outputs
    wire [63:0] Product_out;
    wire Ready;

    // Clock
    reg clk = `HIGH;
```


```verilog
initial
begin : Preprocess
    // Initialize inputs
    Reset         = `LOW;
    Run           = `LOW;
    Multiplicand_in = 32'd0;
    Multiplier_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
```


```verilog
// 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_out),
    .Ready(Ready),
    // Inputs
    .Multiplicand(Multiplicand_in),
    .Multiplier(Multiplier_in),
    .Run(Run),
    .Reset(Reset),
    .clk(clk)
);
```


```verilog
begin : StimuliProcess
    // Start testing
    while (!$feof(input_file))
    begin
        $fscanf(input_file, "%x\n", read_data);
        @(negedge clk); // Wait clock
        {Multiplicand_in, Multiplier_in} = read_data;
        Reset = `HIGH;
        @(negedge clk); // Wait clock
        Reset = `LOW;
        @(negedge 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
```


```verilog
    always @(posedge Ready)
    begin : Monitoring
        $display("Multiplicand:%d, Multiplier:%d", Multiplicand_in, Multiplier_in);
        $display("result:%d", Product_out);
        $fdisplay(output_file, "%t,%x", $time, Product_out);
    end

endmodule
```



```
000003EB_00000014
00000000_00000001
98765423_00000000
FFFFFFFF_00000011
FFFFFFFF_FFFFFFFF
```

Case1:

測試計算過程中沒有任何進位時，計算結果是否正確。

Case2:

測試被乘數為 0 時，計算結果是否正確。

Case3:

測試乘數為 0 時，計算結果是否正確。

Case4:

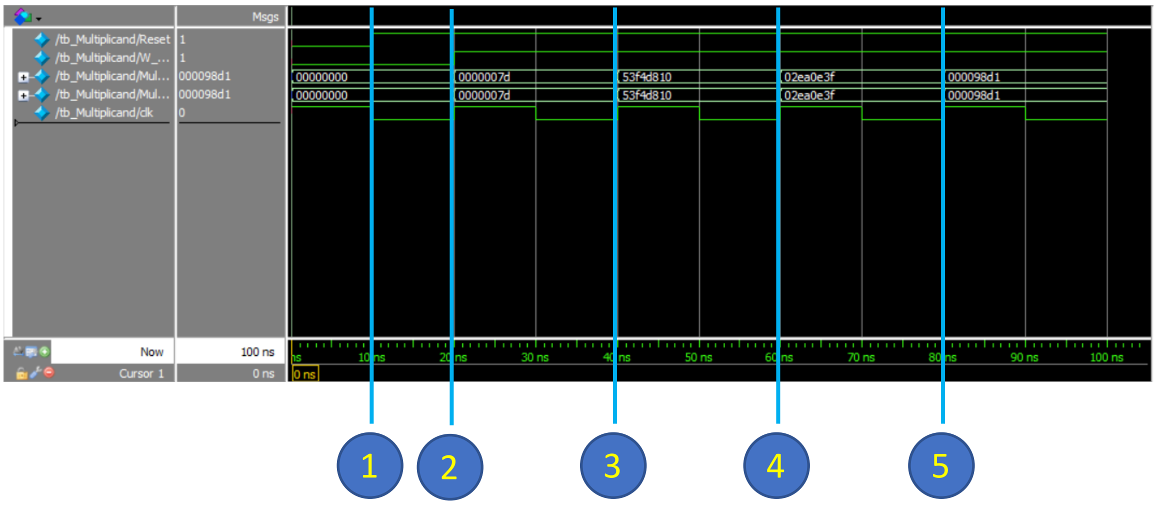
測試計算過程中若出現少量進位時，計算結果是否正確。

Case5:

測試計算過程中若出現大量進位時，計算結果是否正確。

Multiplicand

(Screenshots and Explanations of the test results)



Case 1:

$$Reset \ = \ 0 \rightarrow \text{歸零}$$
$$Multiplicand\_in \ = \ 32'h00000000$$
$$Multiplicand\_out \ = \ 32'h00000000$$

Case 2:

$$Reset \ = \ 1 \text{、} W\_ctrl = 1$$
$$Multiplicand\_in \ = \ 32'h0000007D$$
$$Multiplicand\_out \ = \ 32'h0000007D$$

Case 3:

$$Reset \ = \ 1 \text{、} W\_ctrl = 1$$
$$Multiplicand\_in \ = \ 32'h53F4D810$$
$$Multiplicand\_out \ = \ 32'h53F4D810$$

Case 4:

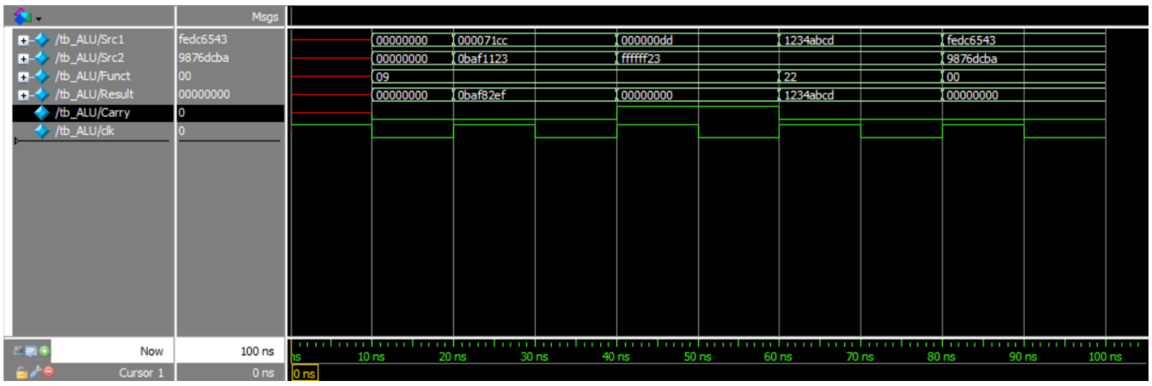
$$Reset \ = \ 1 \text{、} W\_ctrl = 1$$
$$Multiplicand\_in \ = \ 32'h02EA0D3F$$
$$Multiplicand\_out \ = \ 32'h02EA0D3F$$

Case5:

$$Reset \ = \ 1 \text{、} W\_ctrl = 1$$
$$Multiplicand\_in \ = \ 32'h000098D1$$
$$Multiplicand\_out \ = \ 32'h000098D1$$

Alu

(Screenshots and Explanations of the test results)



```
/tb_ALU/Src1    fedc6543   00000000 | 000071cc | 000000dd | 1234abcd | fedc6543
/tb_ALU/Src2    9876dcba   00000000 | 0baf1123 | ffffff23 | 9876dcba
/tb_ALU/Funct   00         09 | 22 | 00
/tb_ALU/Result  00000000   00000000 | 0baf82ef | 00000000 | 1234abcd | 00000000
/tb_ALU/Carry   0
/tb_ALU/clk     0
```

Case 1(Addu[Funct = 001001]):

$$\begin{array}{rll} & 0000\ 0000\ 0000\ 0000\ 0111\ 0001\ 1100\ 1100 & (0000\_71CC) \\ + & 0000\ 1011\ 1010\ 1111\ 0001\ 0001\ 0010\ 0011 & (0BAF\_1123) \\ \hline & 0000\ 1011\ 1010\ 1111\ 1000\ 0010\ 1110\ 1111 & (0BAF\_82EF) \end{array}$$

Case 2(Addu/Carry[Funct =001001]):

$$\begin{array}{rll} & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1101\ 1101 & (0000\_00DD) \\ + & 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 0010\ 0011 & (FFFF\_FF23) \\ \hline 1 & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000 & (0000\_0000) \end{array}$$

Case 3(None[Funct =100010]):

$$Src1\ 0001\ 0010\ 0011\ 0100\ 1010\ 1011\ 1100\ 1101\ (1234\_ABCD)$$

$$\downarrow$$

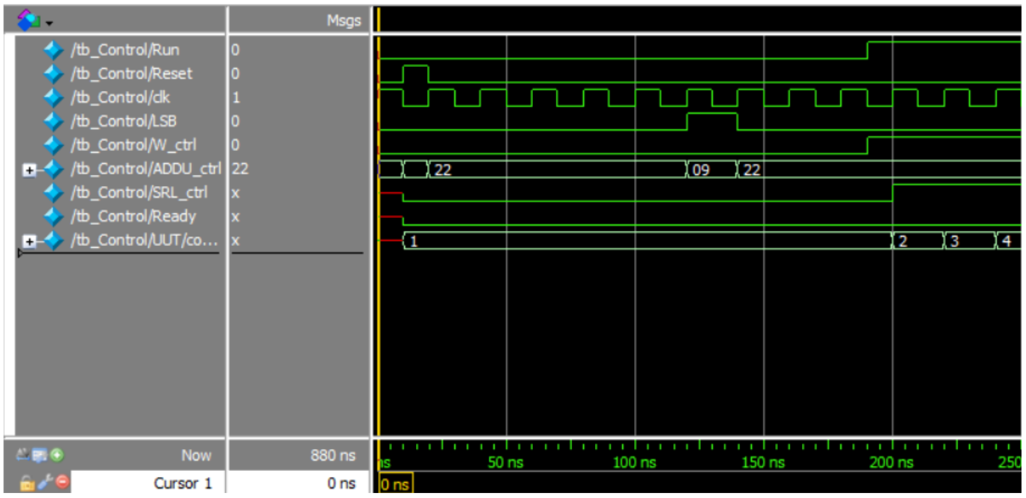
$$Result\ 0001\ 0010\ 0011\ 0100\ 1010\ 1011\ 1100\ 1101\ (1234\_ABCD)$$

Case 4(Funct =00):

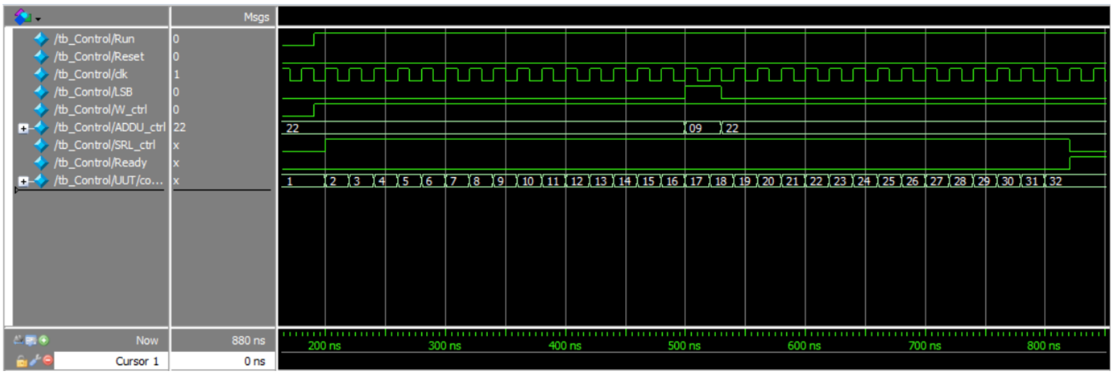
$$Result\ =\ 32'b0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000$$
$$Carry\ =\ 0$$

Control

(Screenshots and Explanations of the test results)



Case1



Case2

Case 1(Run = 0 、W_ctrl = 0 、SRL_ctrl = 0[非計算階段]):

$$\begin{array}{rc} LSB & 0 \rightarrow 1 \\ ADDU\_ctrl & None \rightarrow Addu \\ Counter & Fixed \end{array}$$

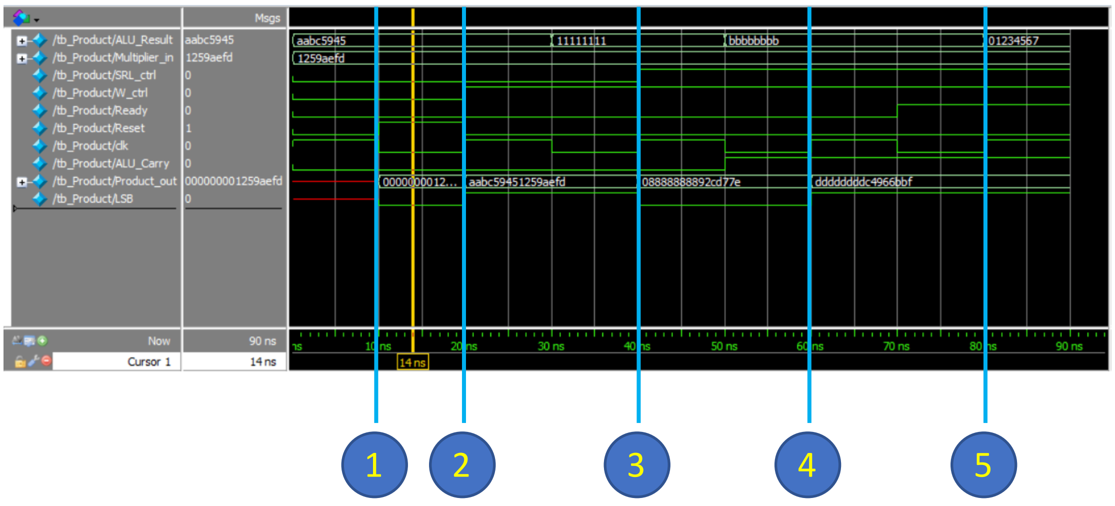
Case 2(Run = 1 、W_ctrl = 1 、SRL_ctrl = 1[計算階段]):

$$\begin{array}{rc} LSB & 0 \rightarrow 1 \\ ADDU\_ctrl & None \rightarrow Addu \end{array}$$

$$\boldsymbol{when} \quad Counter = 32$$
$$\boldsymbol{Calculation\ Complete}\ Ready\ 0 \rightarrow 1$$

Product

(Screenshots and Explanations of the test results)



```
                               Msgs
/tb_Product/ALU_Result         aabc5945
/tb_Product/Multiplier_in      1259aefd
/tb_Product/SRL_ctrl           0
/tb_Product/W_ctrl             0
/tb_Product/Ready              0
/tb_Product/Reset              1
/tb_Product/clk                0
/tb_Product/ALU_Carry          0
/tb_Product/Product_out        000000001259aefd
/tb_Product/LSB                0

Now                            90 ns
Cursor 1                       14 ns
```

Case 1:

$$Reset\ =\ 1 \rightarrow \text{歸零}$$
$$Product\_out[63:32]\ =\ 32'h00000000$$
$$Product\_out[31:0]\ =\ Multiplier\_in(32'h1259AEFD)$$

Case 2:

$$Reset\ =\ 0\ \text{、} W\_ctrl = 1\ \text{、} SRL\_ctrl = 0$$
$$Product\_out[63:32]\ =\ ALU\_Result(32'hAABC5945)$$
$$Product\_out[31:0]\ \ Fixed(32'h1259AEFD)$$

Case 3:

$$Reset\ =\ 0\ \text{、} W\_ctrl = 1\ \text{、} SRL\_ctrl = 1\ \text{、} \mathrm{ALU\_Carry}\ =\ 0$$
$$Product\_out[63:32]\ =\ ALU\_Result(32'h11111111)$$
$$Product\_out[31:0]\ \ Fixed(32'h1259AEFD)$$
$$Product\_out\ =\ Product\_out\ >>\ 1$$

Case 4:

$$Reset\ =\ 0\ \text{、} W\_ctrl = 1\ \text{、} SRL\_ctrl = 1\ \text{、} \mathrm{ALU\_Carry}\ =\ 1$$
$$Product\_out[63:32]\ =\ ALU\_Result(32'hBBBBBBBB)$$
$$Product\_out[31:0]\ \ Fixed(32'h1259AEFD)$$
$$Product\_out\ =\ Product\_out\ >>\ 1$$
$$Product\_out[63]\ =\ 1'b1$$

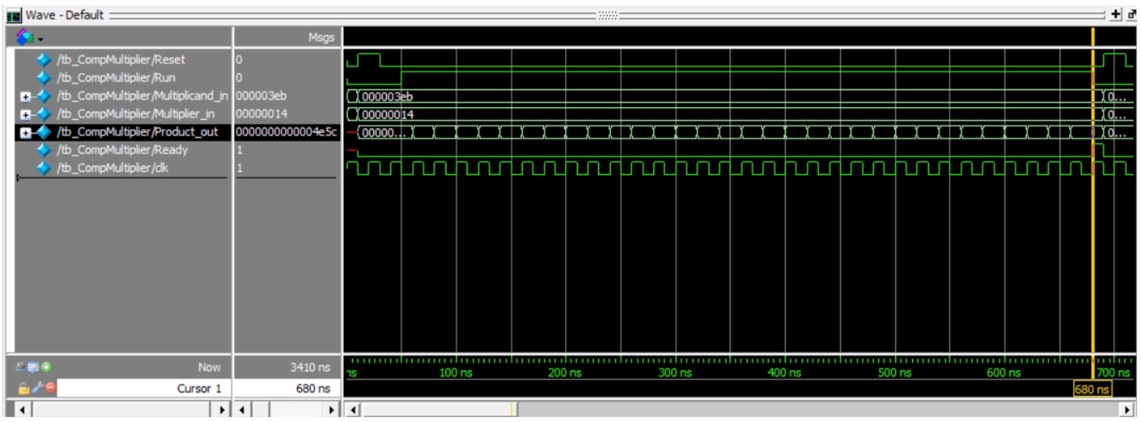
Case5:

$$Ready = 1\ (Calculation\ Complete)$$
~~$ALU\_Result(32'h01234567)$~~

$$Product\_out\ Fixed(64'hDDDDDDDDC4966BBF)$$

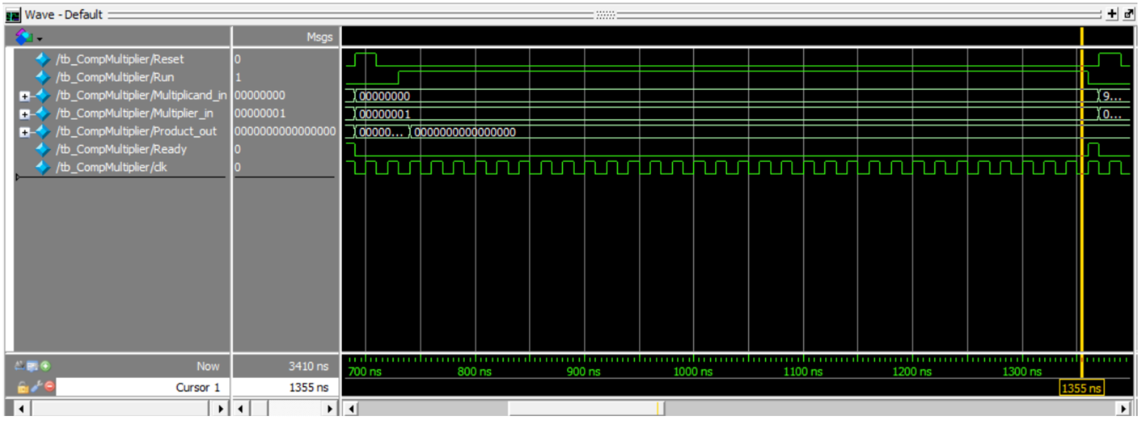
CompMultiplier

(Screenshots and Explanations of the test results)



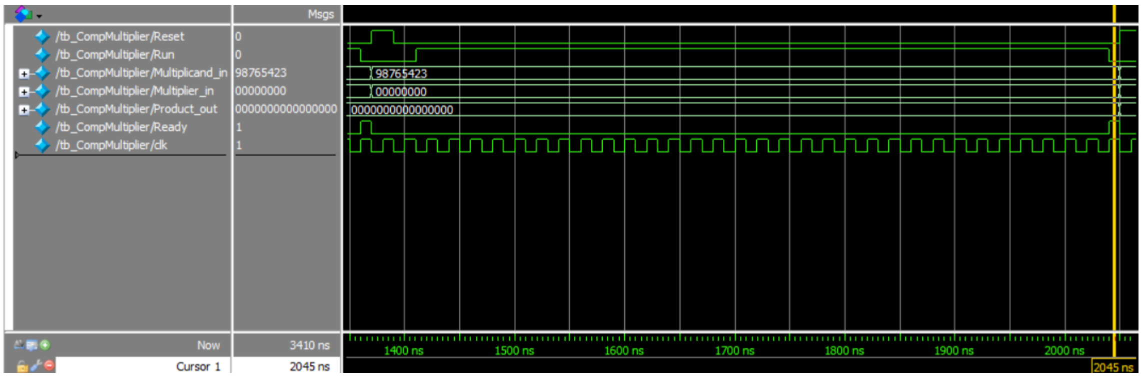
```
Wave - Default
                                      Msgs
/tb_CompMultiplier/Reset              0
/tb_CompMultiplier/Run                0
/tb_CompMultiplier/Multiplicand_in    000003eb
/tb_CompMultiplier/Multiplier_in      00000014
/tb_CompMultiplier/Product_out        0000000000004e5c
/tb_CompMultiplier/Ready              1
/tb_CompMultiplier/clk                1

Now          3410 ns
Cursor 1     680 ns
```

Case1



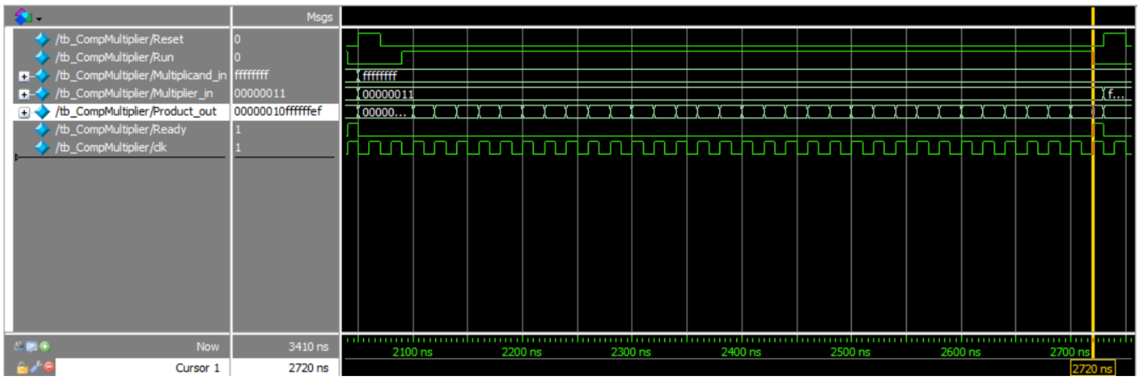
```
Wave - Default
                                      Msgs
/tb_CompMultiplier/Reset              0
/tb_CompMultiplier/Run                1
/tb_CompMultiplier/Multiplicand_in    00000000
/tb_CompMultiplier/Multiplier_in      00000001
/tb_CompMultiplier/Product_out        0000000000000000
/tb_CompMultiplier/Ready              0
/tb_CompMultiplier/clk                0

Now          3410 ns
Cursor 1     1355 ns
```

Case2



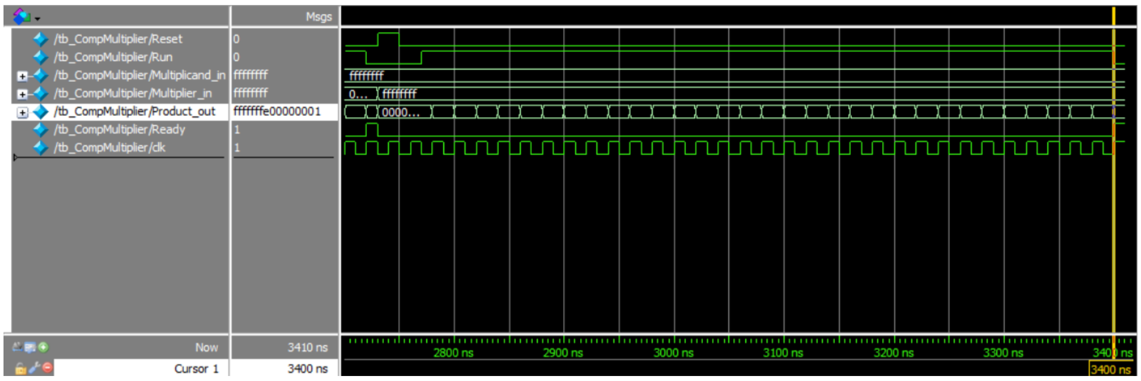
```
                                      Msgs
/tb_CompMultiplier/Reset              0
/tb_CompMultiplier/Run                0
/tb_CompMultiplier/Multiplicand_in    98765423
/tb_CompMultiplier/Multiplier_in      00000000
/tb_CompMultiplier/Product_out        0000000000000000
/tb_CompMultiplier/Ready              1
/tb_CompMultiplier/clk                1

Now          3410 ns
Cursor 1     2045 ns
```

Case3



```
/tb_CompMultiplier/Reset            0
/tb_CompMultiplier/Run              0
/tb_CompMultiplier/Multiplicand_in  ffffffff
/tb_CompMultiplier/Multiplier_in    00000011
/tb_CompMultiplier/Product_out      00000010ffffffef
/tb_CompMultiplier/Ready            1
/tb_CompMultiplier/clk              1
Now       3410 ns
Cursor 1  2720 ns
```

Case4

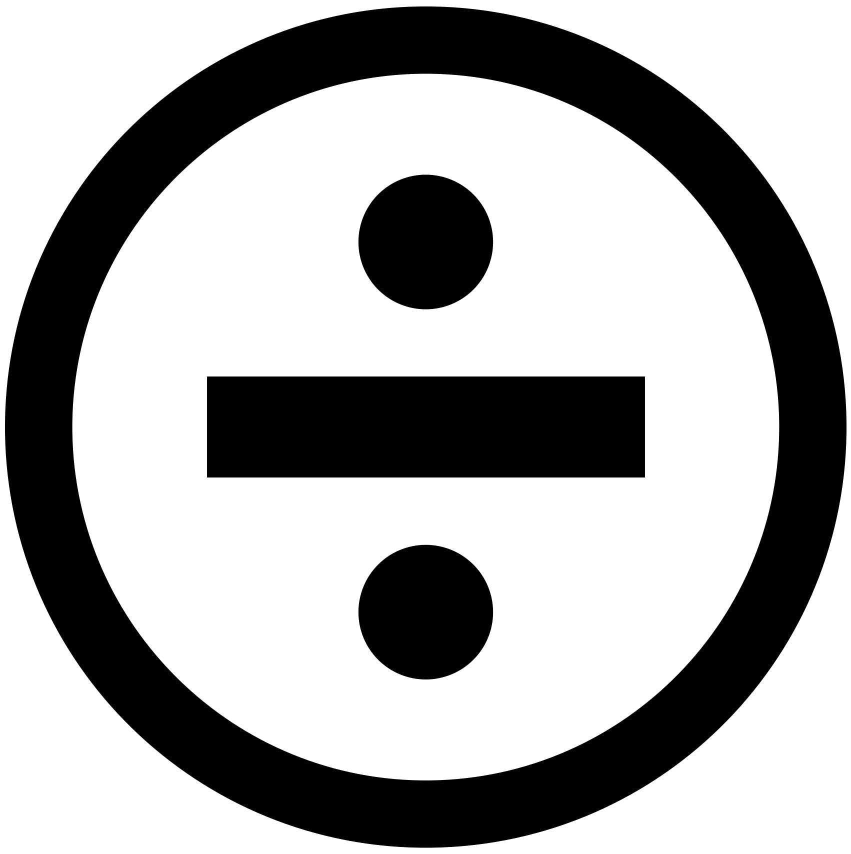


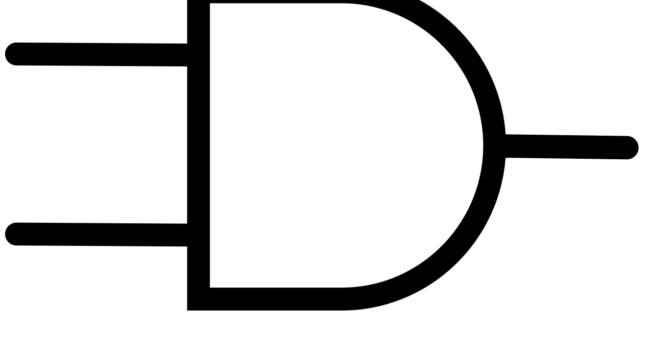
```
/tb_CompMultiplier/Reset            0
/tb_CompMultiplier/Run              0
/tb_CompMultiplier/Multiplicand_in  ffffffff
/tb_CompMultiplier/Multiplier_in    ffffffff
/tb_CompMultiplier/Product_out      fffffffe00000001
/tb_CompMultiplier/Ready            1
/tb_CompMultiplier/clk              1
Now       3410 ns
Cursor 1  3400 ns
```

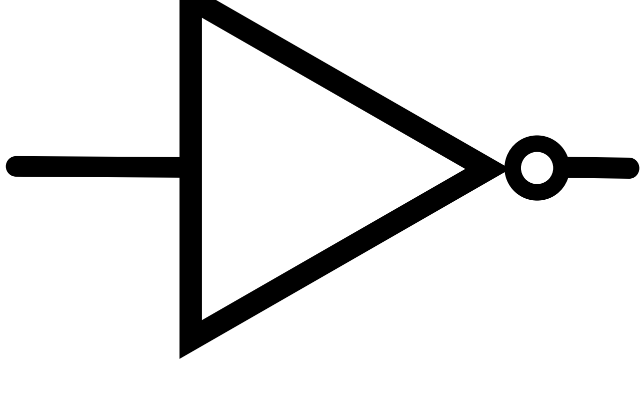
Case5

	進制	被乘數	乘數	積(預想)
Case1	Dec	1003	20	20060
	Hex	0000 03EB	0000 0014	0000 0000 0000 4E5C
Case2	Dec	0	1	0
	Hex	0000 0000	0000 0001	0000 0000 0000 0000
Case3	Dec	2557891634	0	0
	Hex	9876 5423	0000 0000	0000 0000 0000 0000
Case4	Dec	4294967295	17	73014444015
	Hex	FFFF FFFF	0000 0011	0000 0010 FFFF FFEF
Case5	Dec	4294967295	4294967295	18446744065119620000
	Hex	FFFF FFFF	FFFF FFFF	FFFF FFFE 0000 0001

Part 2
Unsigned Divider

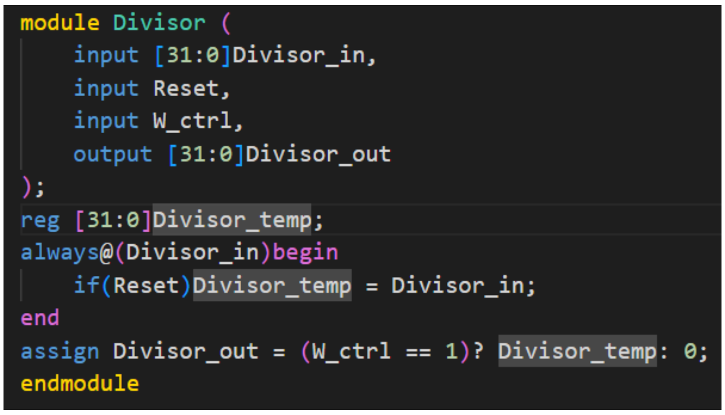




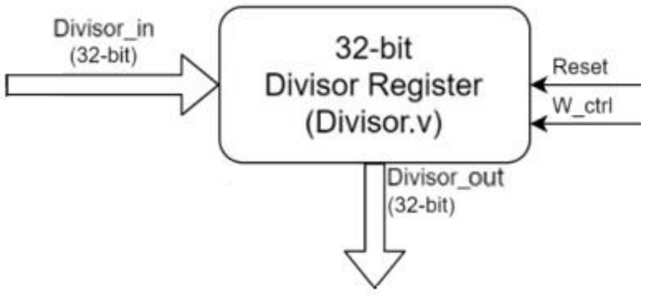


Divisor、ALU、Control、
Remainder、CompDivider

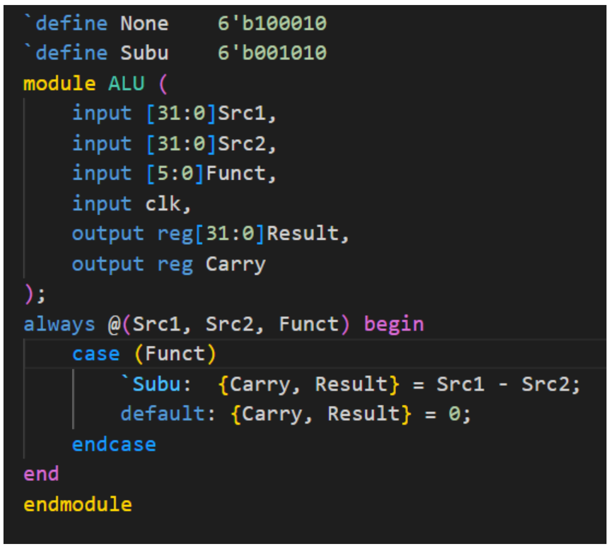
(Screenshots of each program, and description of the process.)



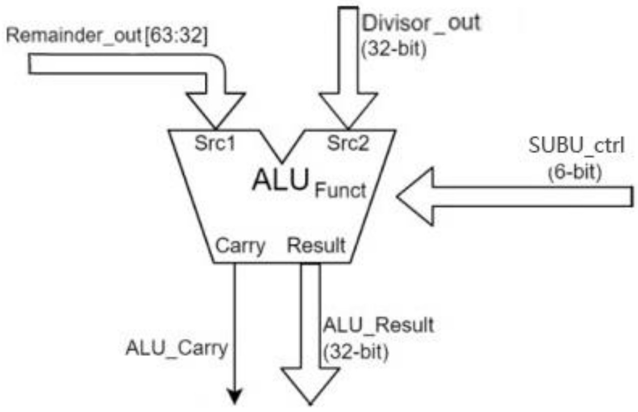
```
module Divisor (
    input [31:0]Divisor_in,
    input Reset,
    input W_ctrl,
    output [31:0]Divisor_out
);
reg [31:0]Divisor_temp;
always@(Divisor_in)begin
    if(Reset)Divisor_temp = Divisor_in;
end
assign Divisor_out = (W_ctrl == 1)? Divisor_temp: 0;
endmodule
```



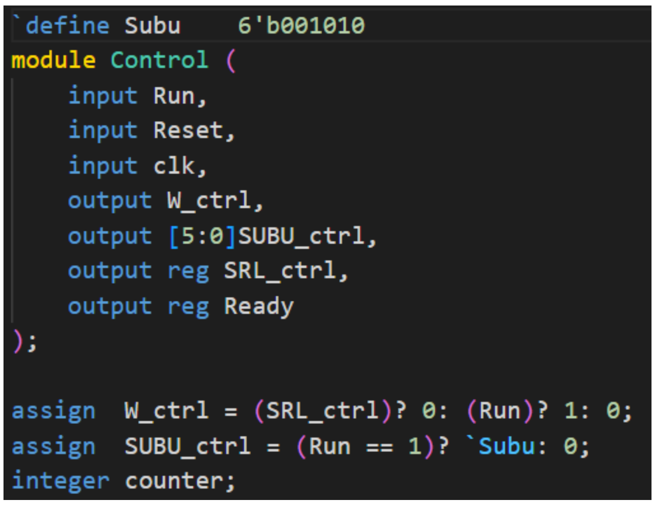
有一個 32bit 的 Divisor_in、Reset、W_ctrl，當有新的 Divisor_in，但是 W_ctrl 為 1，不會將值傳進 Divisor_out，而是暫存在 Divisor_temp 中。當 W_ctrl 為 1 時，Divisor_out 才會更新為新的值，否則就維持原狀。



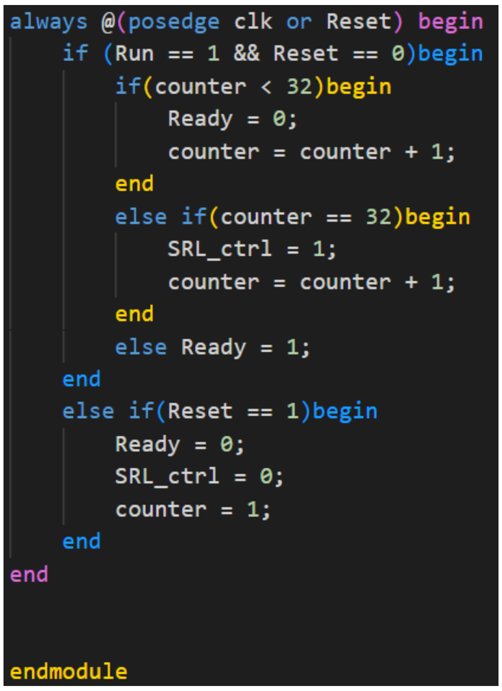
```
`define None    6'b100010
`define Subu    6'b001010
module ALU (
    input [31:0]Src1,
    input [31:0]Src2,
    input [5:0]Funct,
    input clk,
    output reg[31:0]Result,
    output reg Carry
);
always @(Src1, Src2, Funct) begin
    case (Funct)
        `Subu:  {Carry, Result} = Src1 - Src2;
        default: {Carry, Result} = 0;
    endcase
end
endmodule
```



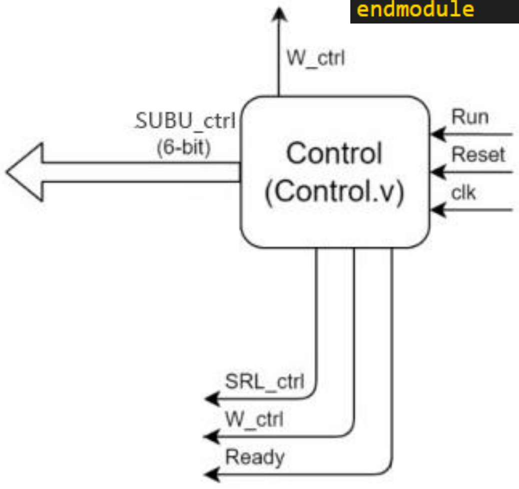
定義每個功能的 Funct，將 Divisor_Register 所取到的資料 Divisor_out 以及 Remainder_out 傳到 ALU 中並判斷 Funct 進行相對應的運算，若相減後負會使 Carry 變為 1，最後將運算完的資料傳到 Result。若 Funct 為 0，Carry 及 Result 會歸零。



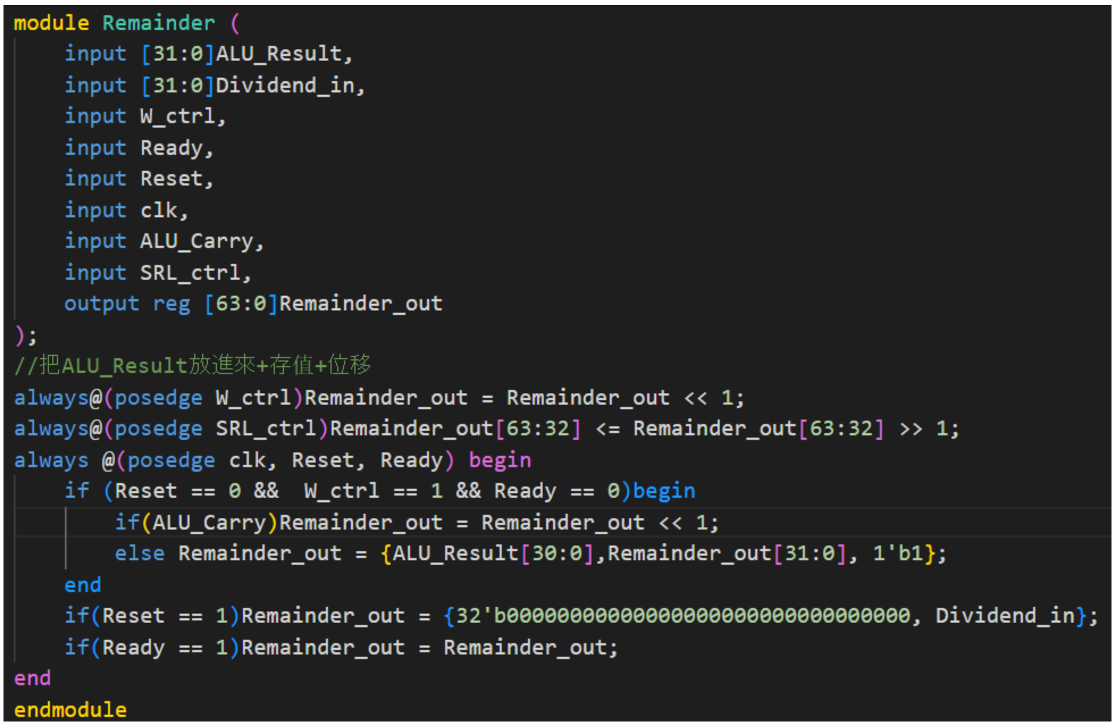
```
`define Subu     6'b001010
module Control (
    input Run,
    input Reset,
    input clk,
    output W_ctrl,
    output [5:0]SUBU_ctrl,
    output reg SRL_ctrl,
    output reg Ready
);

assign  W_ctrl = (SRL_ctrl)? 0: (Run)? 1: 0;
assign  SUBU_ctrl = (Run == 1)? `Subu: 0;
integer counter;
```



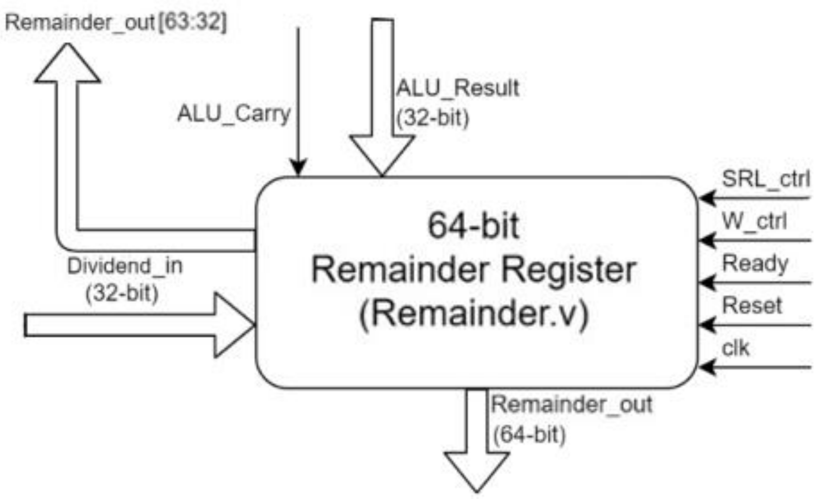
```
always @(posedge clk or Reset) begin
    if (Run == 1 && Reset == 0)begin
        if(counter < 32)begin
            Ready = 0;
            counter = counter + 1;
        end
        else if(counter == 32)begin
            SRL_ctrl = 1;
            counter = counter + 1;
        end
        else Ready = 1;
    end
    else if(Reset == 1)begin
        Ready = 0;
        SRL_ctrl = 0;
        counter = 1;
    end
end


endmodule
```



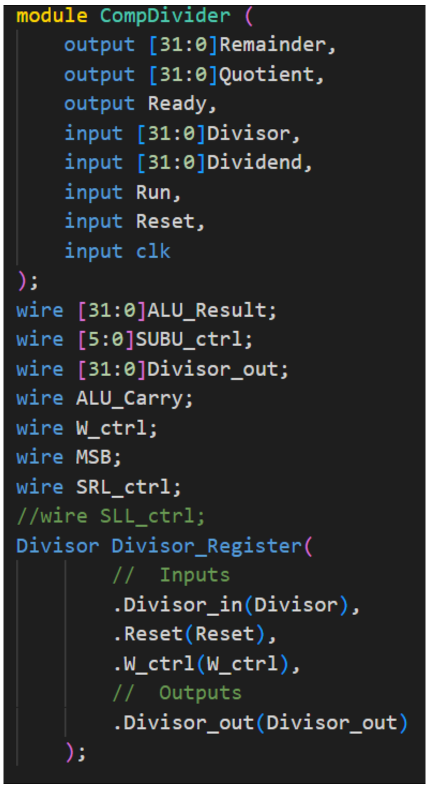
定義 Subu，當 Run 為一時，將 W_ctrl 設為 1，反之為 0。當 Run 為 1 時，SUB_ctrl 設為 Subu，進行減法計算。在 Reset 為 1 時，將所有值歸零，counter 恢復為 1。另外若 Run 為 1 且 Reset 為 0 時，代表還在計算階段，因此計數器 counter 每經過一個正緣 clk 就加 1，另外 SRL_ctrl 也為 1，因為計算階段需要持續地向右位移。當計數器 counter 為 32 時代表計算結束，此時將 SRL_ctrl 改為 1，進行餘數右移，結束後 counter 加 1，使 Ready 變為 1，表示答案已經正確，停止計算，避免其他元件繼續運作影響到答案。



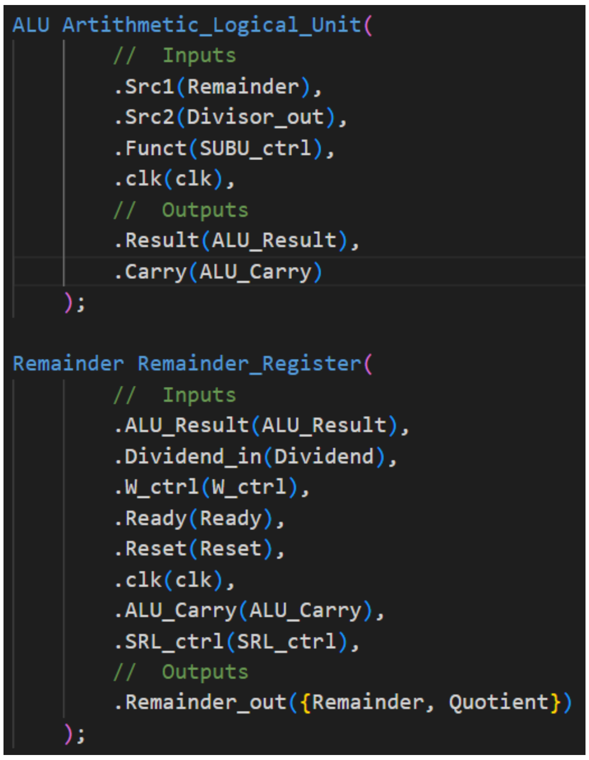
```
module Remainder (
    input [31:0]ALU_Result,
    input [31:0]Dividend_in,
    input W_ctrl,
    input Ready,
    input Reset,
    input clk,
    input ALU_Carry,
    input SRL_ctrl,
    output reg [63:0]Remainder_out
);
//把ALU_Result放進來+存值+位移
always@(posedge W_ctrl)Remainder_out = Remainder_out << 1;
always@(posedge SRL_ctrl)Remainder_out[63:32] <= Remainder_out[63:32] >> 1;
always @(posedge clk, Reset, Ready) begin
    if (Reset == 0 &&  W_ctrl == 1 && Ready == 0)begin
        if(ALU_Carry)Remainder_out = Remainder_out << 1;
        else Remainder_out = {ALU_Result[30:0],Remainder_out[31:0], 1'b1};
    end
    if(Reset == 1)Remainder_out = {32'b00000000000000000000000000000000, Dividend_in};
    if(Ready == 1)Remainder_out = Remainder_out;
end
endmodule
```



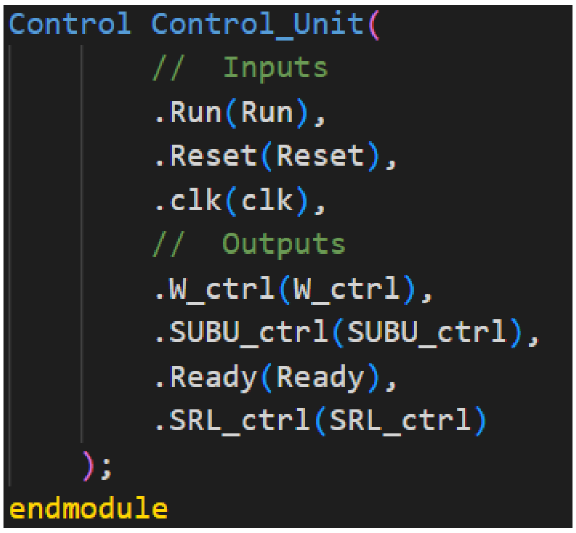
在 Reset 為 1 時，將 Dividend_in 的值放進 Remainder_out 的右半 32bit，左半 32bit 歸零。當 W_ctrl 變為 1 的瞬間表示計算開始，因此 Remainder_out 先向左位移 1bit。當 SRL_ctrl 為 1，表示減法計算結束，因此 Remainder_out 左邊 32bit 向右位移 1bit。當 Reset、Ready 為 0 且 W_ctrl 為 1 時，表示正在計算階段，因此每經過一個正緣 clk，就判斷 ALU_Carry 是否為 1，若是，則將 Remainder_out 向左位移 1bit，反之則將 ALU_Result 放入 Remainder_out 左半 32bit，並將整個 Remainder_out 左移 1bit，最右邊 1 個 bit 放進 1。當 Ready 為 1，表示計算結束，因此維持 Product_out 避免答案出錯。



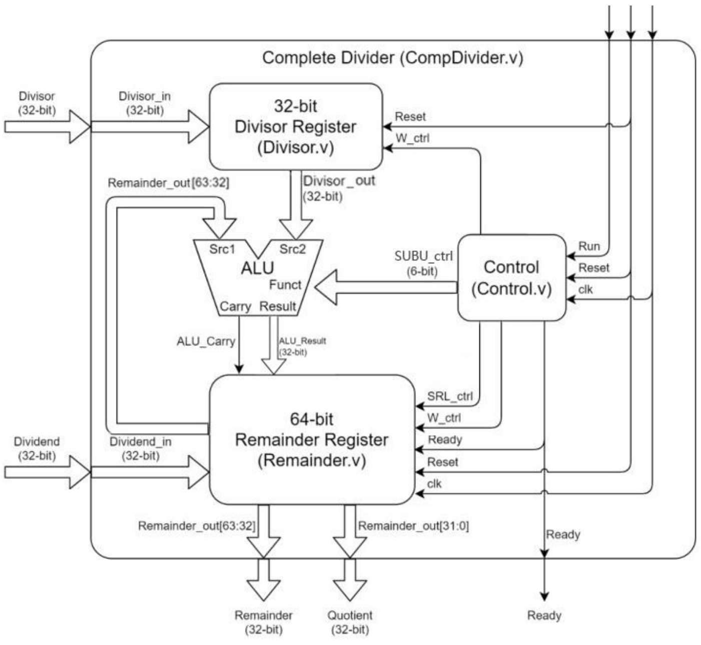
```
module CompDivider (
    output [31:0]Remainder,
    output [31:0]Quotient,
    output Ready,
    input [31:0]Divisor,
    input [31:0]Dividend,
    input Run,
    input Reset,
    input clk
);
wire [31:0]ALU_Result;
wire [5:0]SUBU_ctrl;
wire [31:0]Divisor_out;
wire ALU_Carry;
wire W_ctrl;
wire MSB;
wire SRL_ctrl;
//wire SLL_ctrl;
Divisor Divisor_Register(
        //  Inputs
        .Divisor_in(Divisor),
        .Reset(Reset),
        .W_ctrl(W_ctrl),
        //  Outputs
        .Divisor_out(Divisor_out)
    );
```



```
ALU Artithmetic_Logical_Unit(
        //  Inputs
        .Src1(Remainder),
        .Src2(Divisor_out),
        .Funct(SUBU_ctrl),
        .clk(clk),
        //  Outputs
        .Result(ALU_Result),
        .Carry(ALU_Carry)
    );

Remainder Remainder_Register(
        //  Inputs
        .ALU_Result(ALU_Result),
        .Dividend_in(Dividend),
        .W_ctrl(W_ctrl),
        .Ready(Ready),
        .Reset(Reset),
        .clk(clk),
        .ALU_Carry(ALU_Carry),
        .SRL_ctrl(SRL_ctrl),
        //  Outputs
        .Remainder_out({Remainder, Quotient})
    );
```



```
Control Control_Unit(
        //  Inputs
        .Run(Run),
        .Reset(Reset),
        .clk(clk),
        //  Outputs
        .W_ctrl(W_ctrl),
        .SUBU_ctrl(SUBU_ctrl),
        .Ready(Ready),
        .SRL_ctrl(SRL_ctrl)
    );
endmodule
```



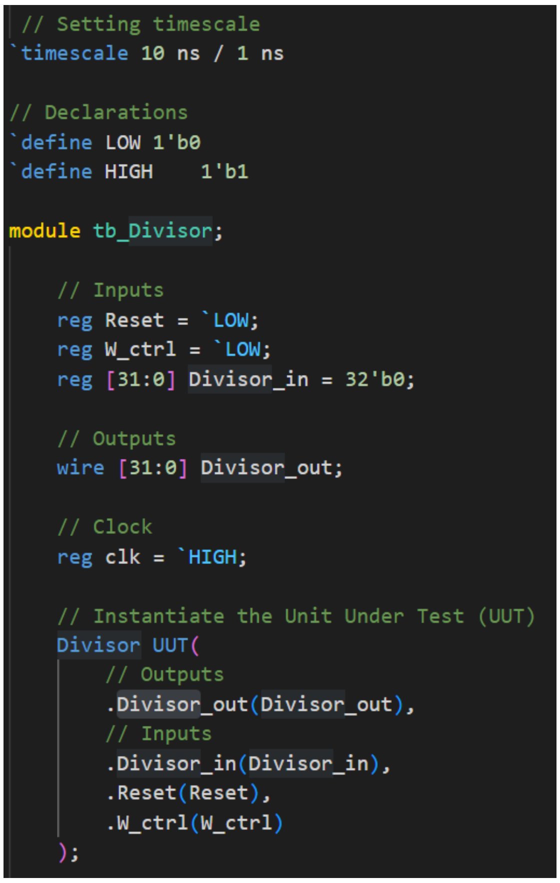
將 Divisor、ALU、Control、Remainder 串在一起成為一個大模組，按照途中的接線將相對應的輸出輸入接好，另外用 Divisor_out、ALU_Result、SUBU_ctrl、W_ctrl、ALU_Carry、SRL_ctrl 內部接線串接除法器內各元件。需要注意的是 ALU 的 Remainder_out 要輸入的應該是 Remainder 進去做運算。另外 Remainder_out 的左 32bit 為 Remainder、右 32bit 為 Quotient。

Divisor、ALU、Control、

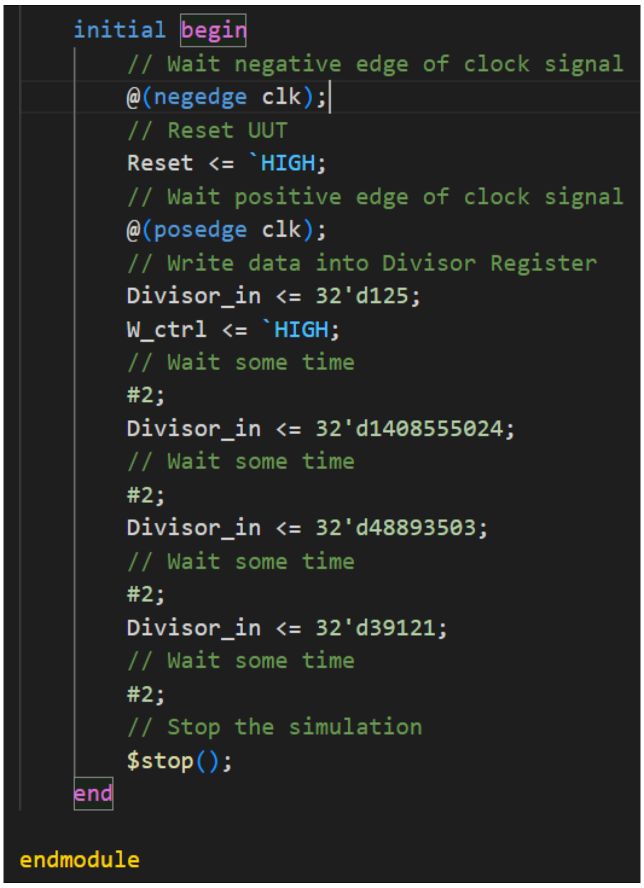
Remainder、CompDivider

(Screenshots of each testbench, and description of the test functions.)

Divisor：



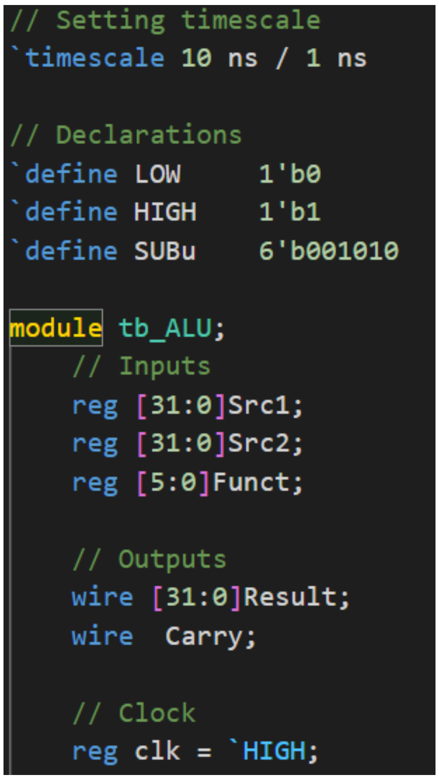
```
// Setting timescale
`timescale 10 ns / 1 ns

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

module tb_Divisor;

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

    // Outputs
    wire [31:0] Divisor_out;

    // Clock
    reg clk = `HIGH;

    // Instantiate the Unit Under Test (UUT)
    Divisor UUT(
        // Outputs
        .Divisor_out(Divisor_out),
        // Inputs
        .Divisor_in(Divisor_in),
        .Reset(Reset),
        .W_ctrl(W_ctrl)
    );
```



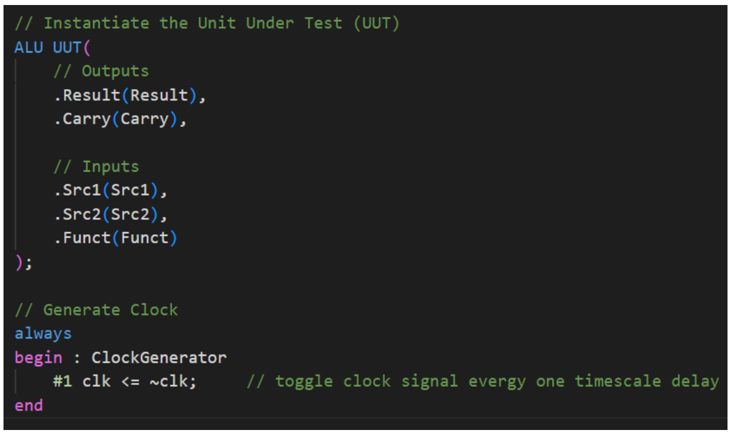
```
    initial begin
        // Wait negative edge of clock signal
        @(negedge clk);
        // Reset UUT
        Reset <= `HIGH;
        // Wait positive edge of clock signal
        @(posedge clk);
        // Write data into Divisor Register
        Divisor_in <= 32'd125;
        W_ctrl <= `HIGH;
        // Wait some time
        #2;
        Divisor_in <= 32'd1408555024;
        // Wait some time
        #2;
        Divisor_in <= 32'd48893503;
        // Wait some time
        #2;
        Divisor_in <= 32'd39121;
        // Wait some time
        #2;
        // Stop the simulation
        $stop();
    end

endmodule
```

程式中先測試了 Reset 為 1 但是 W_ctrl 不為 1 時，Divisor_out 是否維持原值。再測試 Reset 和 W_ctrl 都為 1 時，是否 Divisor _out 也會隨著 Divisor _in 一起變動。

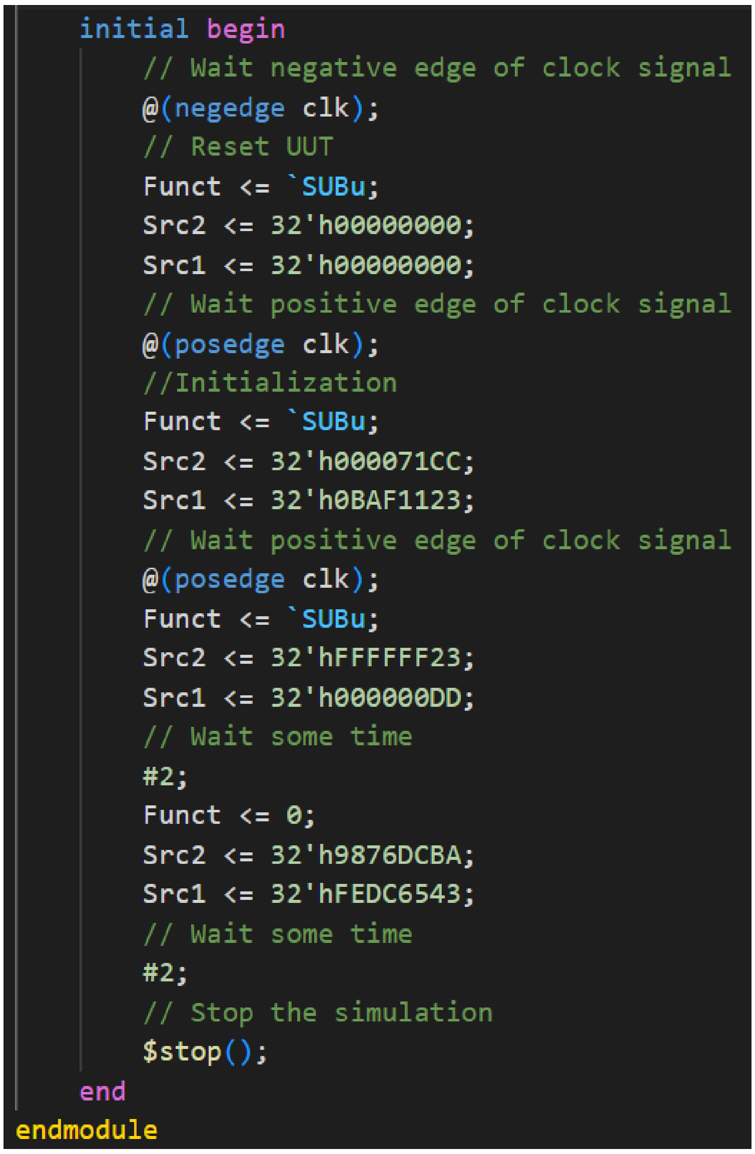
ALU：



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

// Declarations
`define LOW     1'b0
`define HIGH    1'b1
`define SUBu    6'b001010

module tb_ALU;
    // Inputs
    reg [31:0]Src1;
    reg [31:0]Src2;
    reg [5:0]Funct;

    // Outputs
    wire [31:0]Result;
    wire  Carry;

    // Clock
    reg clk = `HIGH;
```



```
// Instantiate the Unit Under Test (UUT)
ALU UUT(
    // Outputs
    .Result(Result),
    .Carry(Carry),

    // Inputs
    .Src1(Src1),
    .Src2(Src2),
    .Funct(Funct)
);

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



```
    initial begin
        // Wait negative edge of clock signal
        @(negedge clk);
        // Reset UUT
        Funct <= `SUBu;
        Src2 <= 32'h00000000;
        Src1 <= 32'h00000000;
        // Wait positive edge of clock signal
        @(posedge clk);
        //Initialization
        Funct <= `SUBu;
        Src2 <= 32'h000071CC;
        Src1 <= 32'h0BAF1123;
        // Wait positive edge of clock signal
        @(posedge clk);
        Funct <= `SUBu;
        Src2 <= 32'hFFFFFF23;
        Src1 <= 32'h000000DD;
        // Wait some time
        #2;
        Funct <= 0;
        Src2 <= 32'h9876DCBA;
        Src1 <= 32'hFEDC6543;
        // Wait some time
        #2;
        // Stop the simulation
        $stop();
    end
endmodule
```

Case1:

Funct = 001010(Subu)，Src2=32'h000071CC，Src1=32'h08AF1123，可以測試減法，相減後為正，因此可以測試 Carry 是否正常。

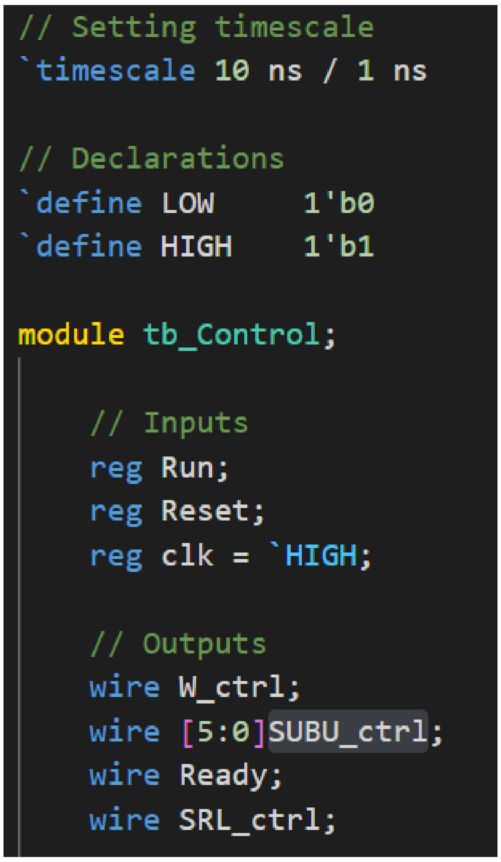
Case2:

Funct =001010(Subu)，Src2=32'hFFFFFF23，Src1=32'h000000DD，可以測試減法，相減後為負，因此可以測試 Carry 是否正常。

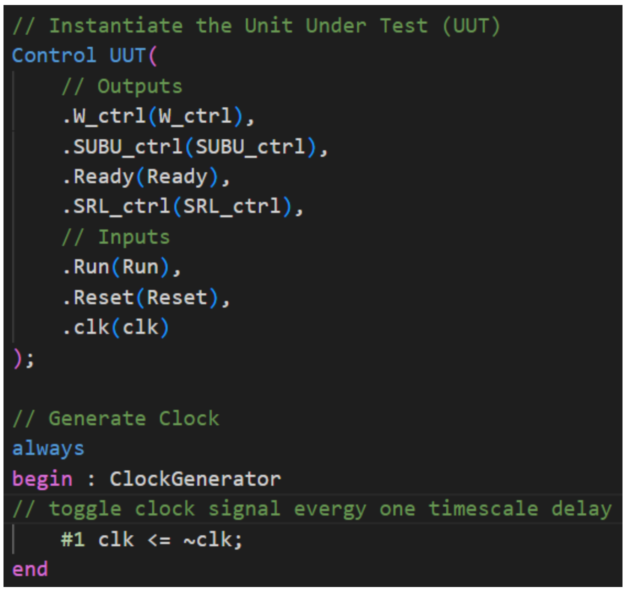
Case4:

Funct = 00，可以測試是否會將 Result 以及 Carry 都歸零。

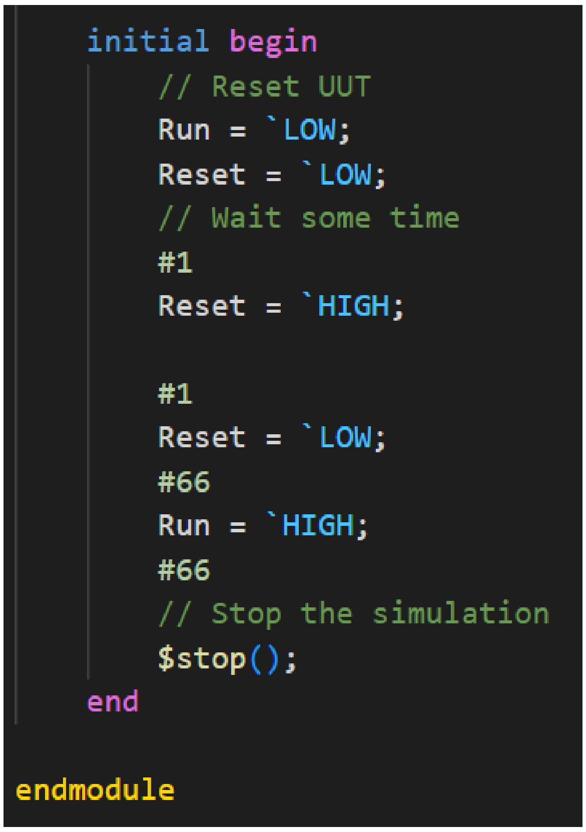
Control：



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

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

module tb_Control;

    // Inputs
    reg Run;
    reg Reset;
    reg clk = `HIGH;

    // Outputs
    wire W_ctrl;
    wire [5:0]SUBU_ctrl;
    wire Ready;
    wire SRL_ctrl;
```



```verilog
// Instantiate the Unit Under Test (UUT)
Control UUT(
    // Outputs
    .W_ctrl(W_ctrl),
    .SUBU_ctrl(SUBU_ctrl),
    .Ready(Ready),
    .SRL_ctrl(SRL_ctrl),
    // Inputs
    .Run(Run),
    .Reset(Reset),
    .clk(clk)
);

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



```verilog
    initial begin
        // Reset UUT
        Run = `LOW;
        Reset = `LOW;
        // Wait some time
        #1
        Reset = `HIGH;

        #1
        Reset = `LOW;
        #66
        Run = `HIGH;
        #66
        // Stop the simulation
        $stop();
    end

endmodule
```

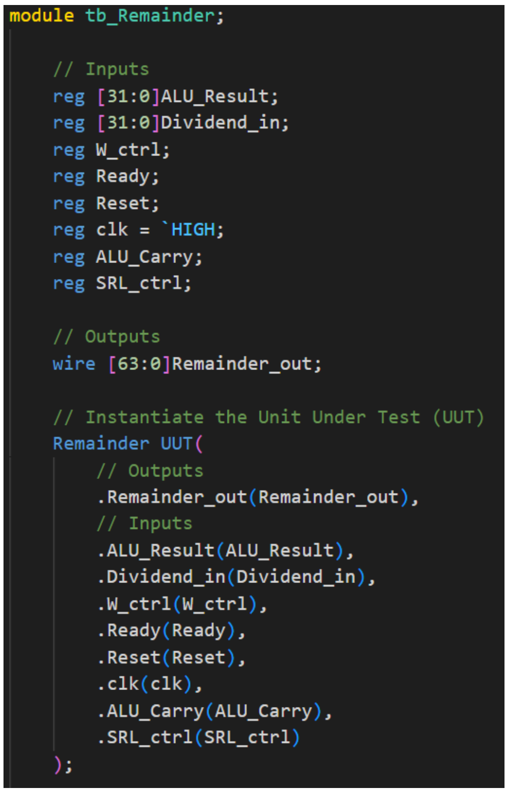
Case1:

Run = 0、Reset = 0、W_ctrl = 0 時為非計算階段，因此可以觀察 counter 是否維持，不隨 clk 變動而增加。

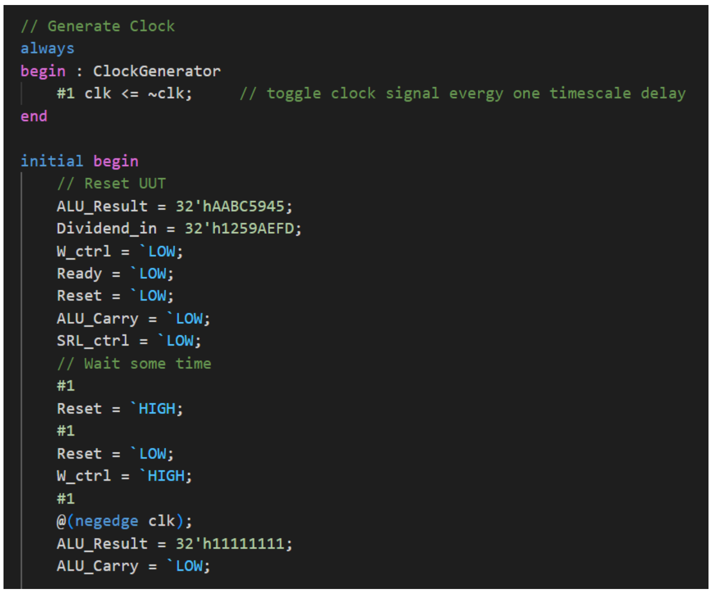
Case2:

Run =1、Reset = 0、W_ctrl = 1 時為計算階段，因此可以觀察 32 個 clk 之後，SRL 是否由 0 變為 1，第 33 個 clk 時 Ready 是否變為 1。也可以觀察 counter 是否隨 clk 增加。

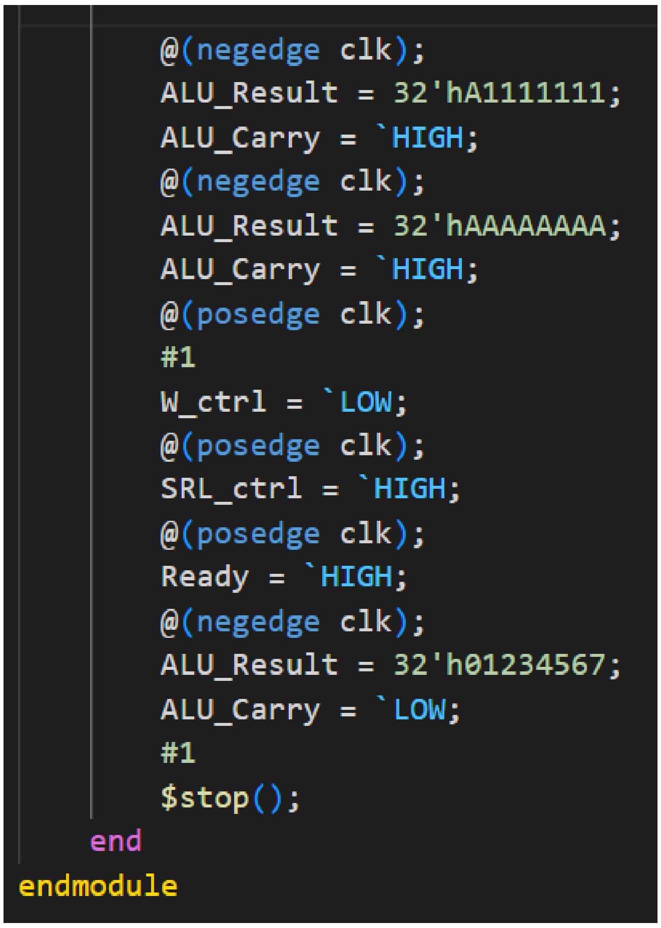
Remainder：



```
module tb_Remainder;

    // Inputs
    reg [31:0]ALU_Result;
    reg [31:0]Dividend_in;
    reg W_ctrl;
    reg Ready;
    reg Reset;
    reg clk = `HIGH;
    reg ALU_Carry;
    reg SRL_ctrl;

    // Outputs
    wire [63:0]Remainder_out;

    // Instantiate the Unit Under Test (UUT)
    Remainder UUT(
        // Outputs
        .Remainder_out(Remainder_out),
        // Inputs
        .ALU_Result(ALU_Result),
        .Dividend_in(Dividend_in),
        .W_ctrl(W_ctrl),
        .Ready(Ready),
        .Reset(Reset),
        .clk(clk),
        .ALU_Carry(ALU_Carry),
        .SRL_ctrl(SRL_ctrl)
    );
```



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

initial begin
    // Reset UUT
    ALU_Result = 32'hAABC5945;
    Dividend_in = 32'h1259AEFD;
    W_ctrl = `LOW;
    Ready = `LOW;
    Reset = `LOW;
    ALU_Carry = `LOW;
    SRL_ctrl = `LOW;
    // Wait some time
    #1
    Reset = `HIGH;
    #1
    Reset = `LOW;
    W_ctrl = `HIGH;
    #1
    @(negedge clk);
    ALU_Result = 32'h11111111;
    ALU_Carry = `LOW;
```



```
        @(negedge clk);
        ALU_Result = 32'hA1111111;
        ALU_Carry = `HIGH;
        @(negedge clk);
        ALU_Result = 32'hAAAAAAAA;
        ALU_Carry = `HIGH;
        @(posedge clk);
        #1
        W_ctrl = `LOW;
        @(posedge clk);
        SRL_ctrl = `HIGH;
        @(posedge clk);
        Ready = `HIGH;
        @(negedge clk);
        ALU_Result = 32'h01234567;
        ALU_Carry = `LOW;
        #1
        $stop();
    end
endmodule
```

Case1:

ALU_Result = 32'hAABC5945，Dividend_in = 32'h1259AEFD，Reset = 1、SRL_ctrl=0、W_ctrl=0、Ready=0、ALU_Carry=0 時，可以觀察 Product_out=是否將 Multiplier_in 放進右半 32bit。

Case2:

ALU_Result = 32'h11111111，Dividend_in = 32'h1259AEFD，Reset = 0、SRL_ctrl=0、W_ctrl=1、Ready=0、ALU_Carry=0 時，可以觀察 Remainder_out 是否將 Multiplier_in 放進右半 32bit、ALU_Result 放進左半 32bit 後，向左位移 1 bit 並將 1 放進右邊第一個 bit。

Case3:

ALU_Result = 32'hA1111111，Dividend_in = 32'h1259AEFD，Reset = 0、SRL_ctrl=1、W_ctrl=1、Ready=0、ALU_Carry=1 時，可以觀察 Remainder_out 是否向左移 1bit。

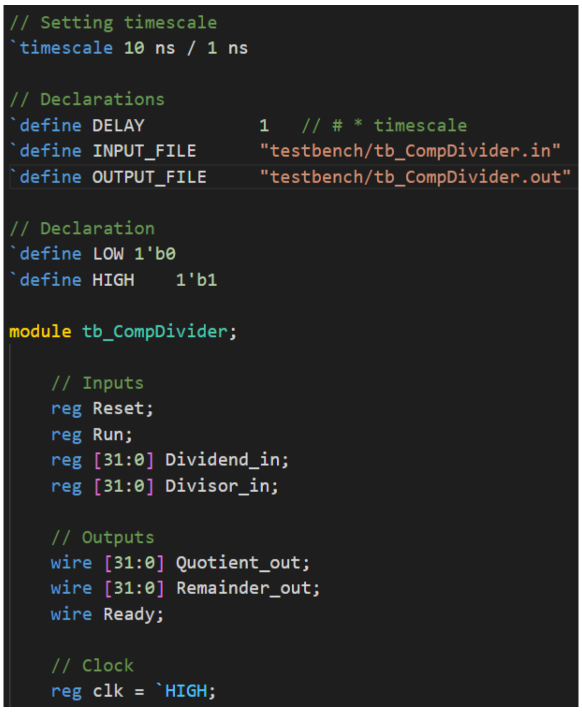
Case4:

ALU_Result = 32'hAAAAAAAA，Dividend_in = 32'h1259AEFD，Reset = 0、SRL_ctrl=1、W_ctrl=1、Ready=0、ALU_Carry=1 時，可以觀察 Remainder_out 的左半 32bit 是否向右位移 1bit。

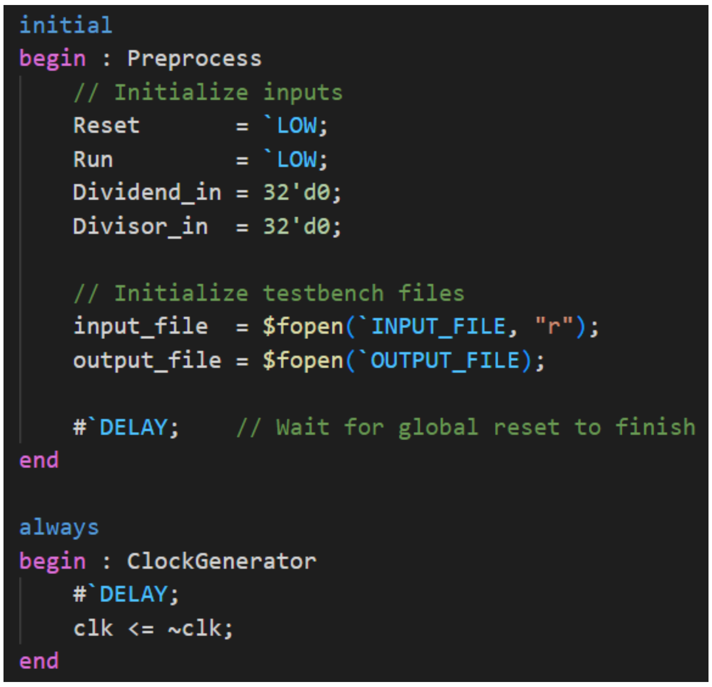
Case5:

Ready = 1 時，Remainder_out 是否不受 ALU_Result 改變為 32'h01234567 所影響，繼續維持原本的值。

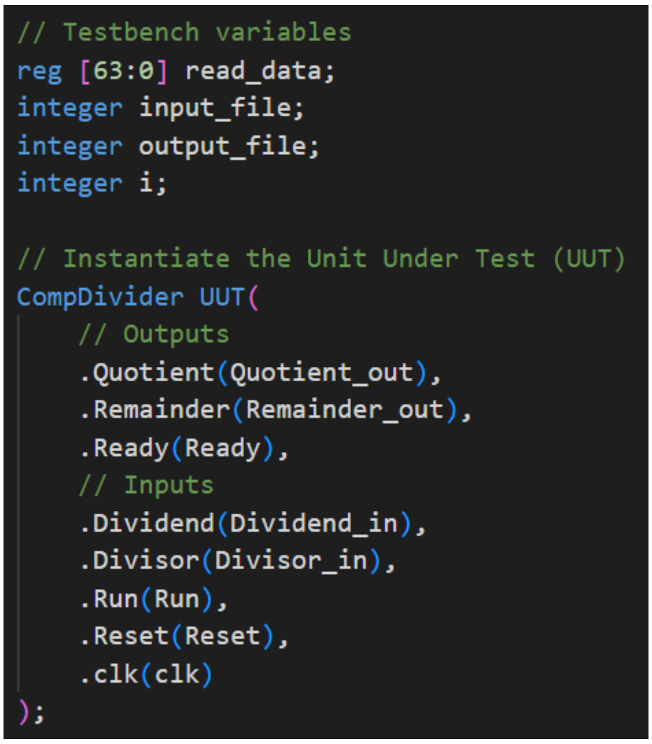
CompDivider：



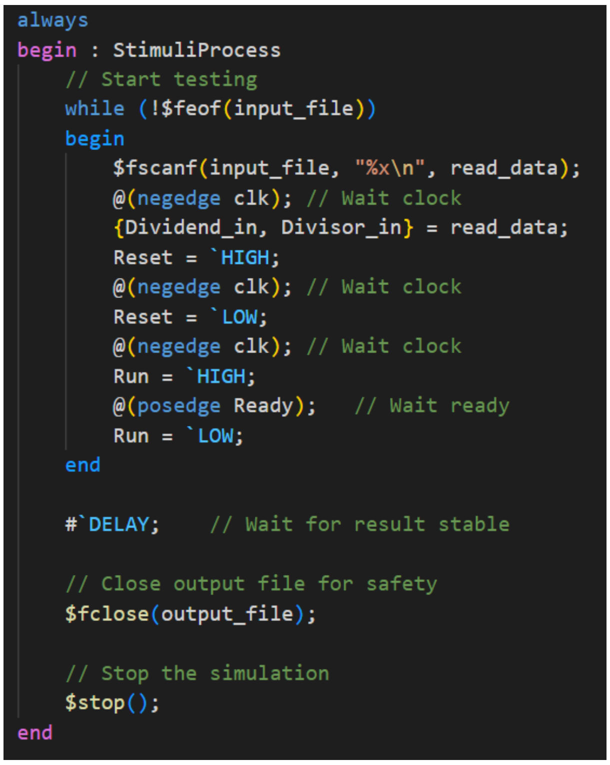
```
// Setting timescale
`timescale 10 ns / 1 ns

// 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 = `HIGH;
```



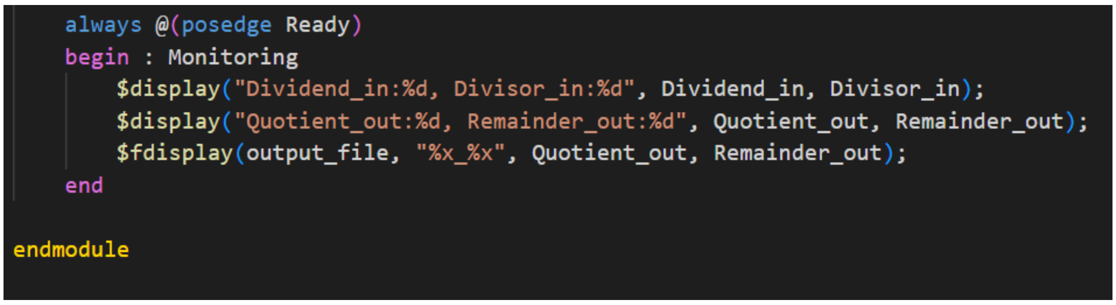
```
initial
begin : Preprocess
    // Initialize inputs
    Reset       = `LOW;
    Run         = `LOW;
    Dividend_in = 32'd0;
    Divisor_in  = 32'd0;

    // 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
```



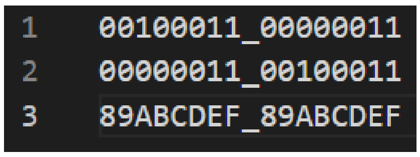
```
// Testbench variables
reg [63:0] read_data;
integer input_file;
integer output_file;
integer i;

// 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)
);
```



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



```
00100011_00000011
00000011_00100011
89ABCDEF_89ABCDEF
```

Case1:

測試被除數大於除數且有餘數時，計算結果是否正確。

Case2:

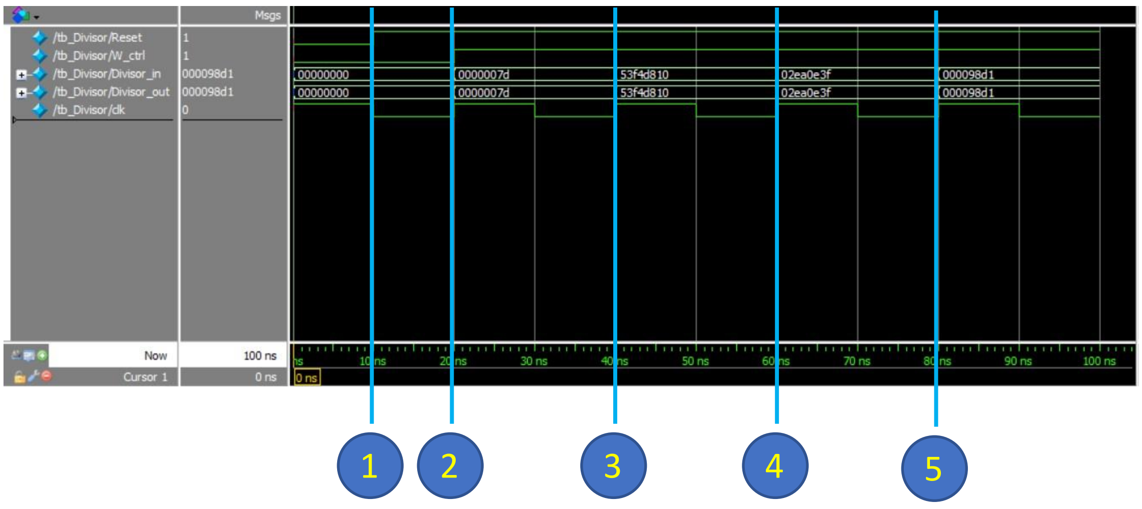
測試被除數小於除數且測試整除時，計算結果是否正確。

Case3:

測試整除時，計算結果是否正確。

Divisor

(Screenshots and Explanations of the test results)



Case 1:

$$Reset \ = \ 0 \rightarrow 歸零$$
$$Divisor\_in \ = \ 32'h00000000$$
$$Divisor\_out \ = \ 32'h00000000$$

Case 2:

$$Reset \ = \ 1 、 W\_ctrl = 1$$
$$Divisor\_in \ = \ 32'h0000007D$$
$$Divisor\_out \ = \ 32'h0000007D$$

Case 3:

$$Reset \ = \ 1 、 W\_ctrl = 1$$
$$Divisor\_in \ = \ 32'h53F4D810$$
$$Divisor\_out \ = \ 32'h53F4D810$$

Case 4:

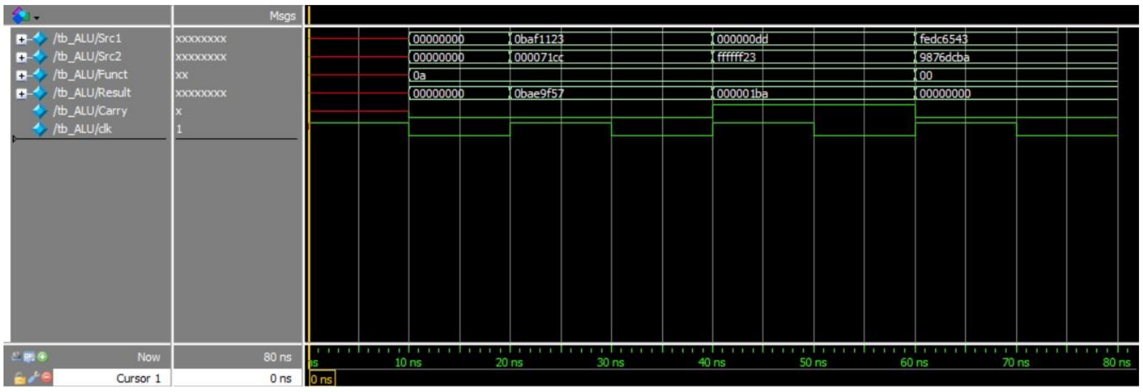
$$Reset \ = \ 1 、 W\_ctrl = 1$$
$$Divisor\_in \ = \ 32'h02EA0D3F$$
$$Divisor\_out \ = \ 32'h02EA0D3F$$

Case5:

$$Reset \ = \ 1 、 W\_ctrl = 1$$
$$Divisor\_in \ = \ 32'h000098D1$$
$$Divisor\_out \ = \ 32'h000098D1$$

Alu

(Screenshots and Explanations of the test results)



```
                 Msgs
/tb_ALU/Src1     xxxxxxxx    00000000  0baf1123  000000dd  fedc6543
/tb_ALU/Src2     xxxxxxxx    00000000  000071cc  ffffff23  9876dcba
/tb_ALU/Funct    xx          0a                            00
/tb_ALU/Result   xxxxxxxx    00000000  0bae9f57  000001ba  00000000
/tb_ALU/Carry    x
/tb_ALU/clk      1
Now              80 ns
Cursor 1         0 ns
```

Case 1(Subu[Funct = 001010]):

$$
\begin{array}{rrr}
 & 0000\ 1011\ 1010\ 1111\ 0001\ 0001\ 0010\ 0011 & (0BAF\_1123) \\
- & 0000\ 0000\ 0000\ 0000\ \ 0111\ 0001\ 1100\ 1100 & (0000\_71CC) \\
\hline
 & 0000\ 1011\ 1010\ 1110\ 1001\ 1111\ 0101\ 0111 & (0BAE\_9F57)
\end{array}
$$

Case 2(Subu/Carry[Funct =001010]):

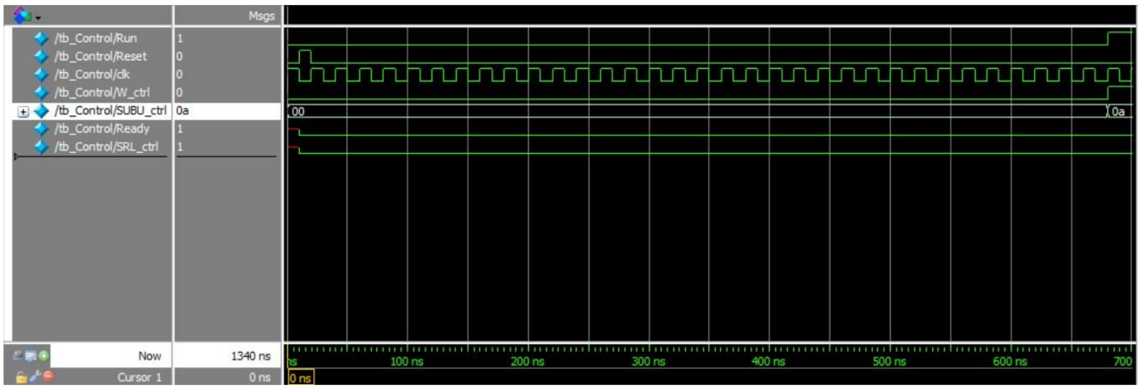
$$
\begin{array}{rrr}
 & 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1101\ 1101 & (0000\_00DD) \\
- & 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 0010\ 0011 & (FFFF\_FF23) \\
\hline
{\color{red}1(Carry)} & < 0 & (XXXX\_XXXX)
\end{array}
$$

Case 3(Funct = 00):

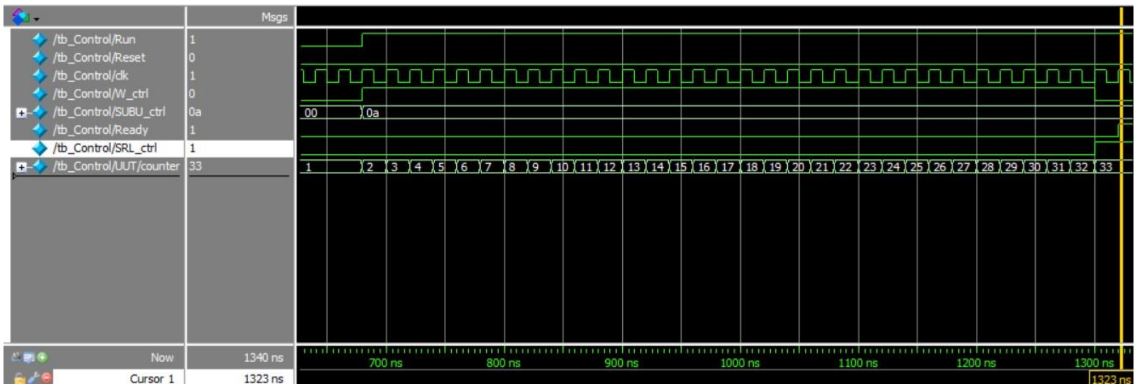
$$
Result\ =\ 32'b0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000
$$
$$
Carry\ =\ 0
$$

Control

(Screenshots and Explanations of the test results)



Case1



Case2

Case 1(Run = 0 、 W_ctrl = 0 [非計算階段]):

$$Counter \qquad Fixed$$

Case 2(Run = 1 、 W_ctrl = 1 [計算階段]):

$$LSB \qquad 0 \rightarrow 1$$

$$SUBU\_ctrl \qquad 00 \rightarrow Subu$$

$$\boldsymbol{when} \quad Counter = 32$$

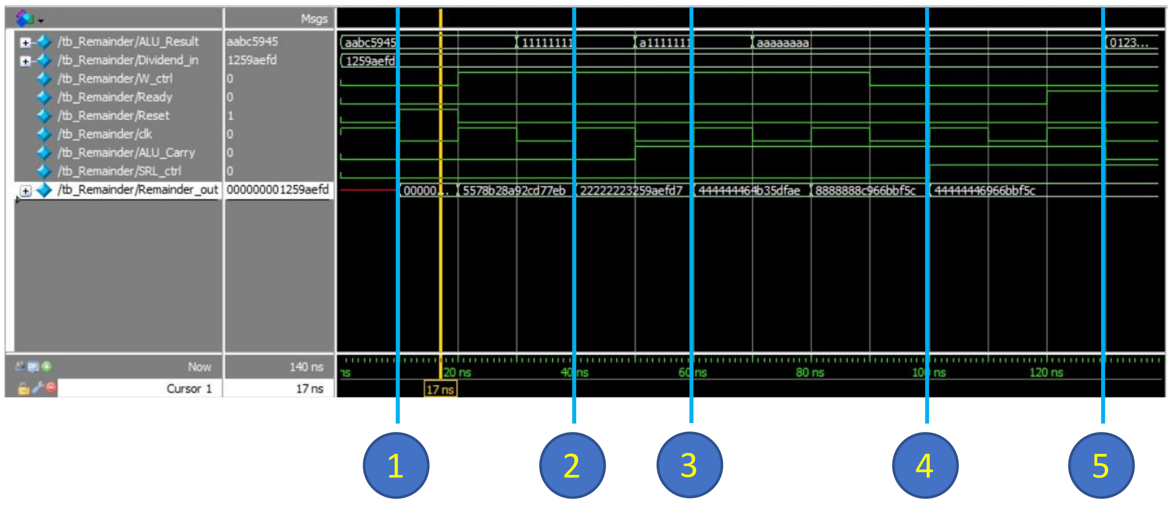
$$SRL\_ctrl \qquad 0 \rightarrow 1$$

$$\boldsymbol{when} \quad Counter = 33$$

$$\boldsymbol{Calculation\ Complete}\ Ready\ 0 \rightarrow 1$$

Remainder

(Screenshots and Explanations of the test results)



Case 1:

$$Reset\ = 1 \rightarrow 歸零$$
$$Remainder\_out[63:32]\ = 32'h00000000$$
$$Remainder\_out[31:0]\ = Dividend\_in(32'h1259AEFD)$$

Case 2:

$$Reset\ = 0 \text{、} W\_ctrl = 1 \text{、} ALU\_Carry = 0$$
$$Remainder\_out[63:32]\ = ALU\_Result(32'h11111111)$$
$$Remainder\_out[31:0]\ Fixed(32'h1259AEFD)$$
$$Remainder\_out = Remainder\_out \ll 1$$
$$Remainder\_out[0]\ = 1'b1$$

Case 3:

$$Reset\ = 0 \text{、} W\_ctrl = 1 \text{、} ALU\_Carry = 1$$
$$Remainder\_out\ = Remainder\_out \ll 1$$

Case 4:

$$SRL\_ctrl\ = 1$$
$$Remainder\_out[63:32] = Remainder\_out[63:32] \gg 1$$

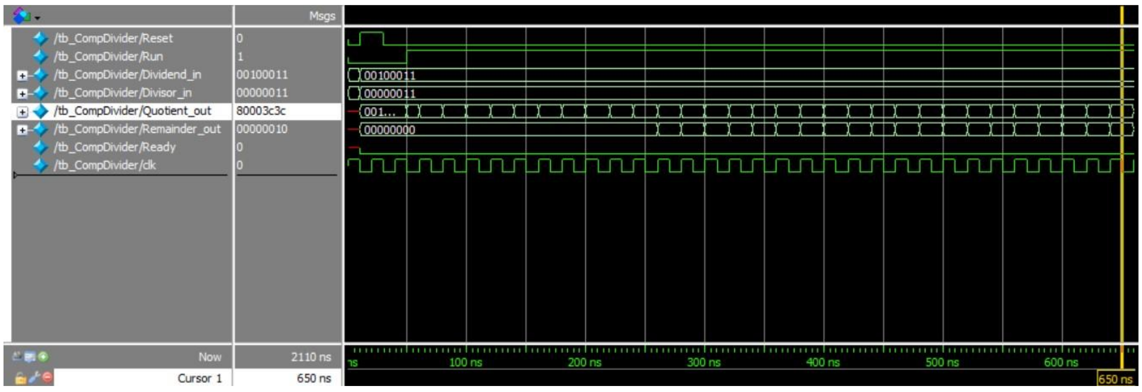
Case5:

$$Ready = 1\ (Calculation\ Complete)$$
$$\cancel{ALU\_Result(8'h01234567)}$$

$$Remainder\_out\ Fixed(64'h44444446966BBF5C)$$

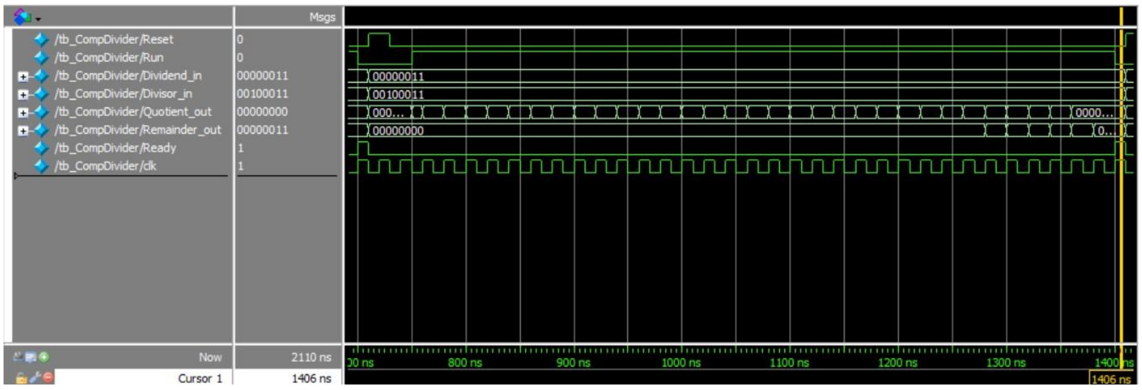
CompDivider

(Screenshots and Explanations of the test results)



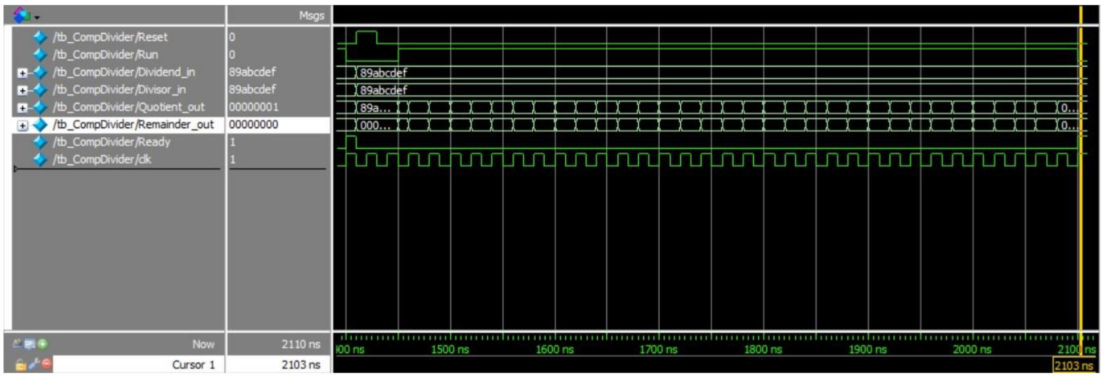
```
/tb_CompDivider/Reset          0
/tb_CompDivider/Run            1
/tb_CompDivider/Dividend_in    00100011
/tb_CompDivider/Divisor_in     00000011
/tb_CompDivider/Quotient_out   80003c3c
/tb_CompDivider/Remainder_out  00000010
/tb_CompDivider/Ready          0
/tb_CompDivider/clk            0
Now                            2110 ns
Cursor 1                       650 ns
```

Case1



```
/tb_CompDivider/Reset          0
/tb_CompDivider/Run            0
/tb_CompDivider/Dividend_in    00000011
/tb_CompDivider/Divisor_in     00100011
/tb_CompDivider/Quotient_out   00000000
/tb_CompDivider/Remainder_out  00000011
/tb_CompDivider/Ready          1
/tb_CompDivider/clk            1
Now                            2110 ns
Cursor 1                       1406 ns
```

Case2



```
/tb_CompDivider/Reset          0
/tb_CompDivider/Run            0
/tb_CompDivider/Dividend_in    89abcdef
/tb_CompDivider/Divisor_in     89abcdef
/tb_CompDivider/Quotient_out   00000001
/tb_CompDivider/Remainder_out  00000000
/tb_CompDivider/Ready          1
/tb_CompDivider/clk            1
Now                            2110 ns
Cursor 1                       2103 ns
```

Case3

	進制	被除數	除數	商(預想)	餘數(預想)
Case1	Dec	1048593	17	61681	16
	Hex	0010 0011	0000 0011	0000 f0f1	0000 0010
Case2	Dec	17	1048593	0	17
	Hex	0000 0011	0010 0011	0000 0000	0000 0011
Case3	Dec	2309737967	2309737967	1	0
	Hex	89AB CDEF	89AB CDEF	0000 0001	0000 0000

Conclusion and insight

這次的作業是要讓我們做出乘法器和除法器。在上課的時候，我覺得乘除法器的運作方式很簡單，應該不會花太久的時間。但是實際在寫的時候就會發現其實還會遇到很多問題，例如：有兩個變數都是 CLK 正緣觸發，其中一個值在正緣觸發時會去取另一個的值，而另一個值是在正緣觸發時會改變值，這時就會發現其中一個會延遲一個 CLK。這次實驗中讓我學到最多的就是在設定觸發條件的時候一定要先想清楚，不然很容易會取到不是自己想要的值，或者我有遇到因為不同的觸發條件而一次出發兩遍的情形，讓我 Debug 很久。