國 立 台 灣 科 技 大 學

計 算 機 組 織

指 導 教 授：陳雅淑 教 授

計算機組織作業PA1報告

班 級　：　四電機三甲
學 生　：　呂佳祐
學 號　：　B11107055
乘法器　：　Area: 2408.630、Slack: 4.3850
除法器　：　Area: 2570.358、Slack: 4.5649

Part1. Multiplier

A. 32-bit Multiplicand Register

```
module Multiplicand(
    output reg [31:0] Multiplicand_out,
    input [31:0] Multiplicand_in,
    input clk,
    input Reset,
    input W_ctrl
);
    always @(*) begin
        if (Reset)
            Multiplicand_out = 32'b0;
        else if (W_ctrl)
            Multiplicand_out = Multiplicand_in;
    end
endmodule
```

Multiplicand.v 程式碼

Multiplicand Register 用於暫存外部輸入之被乘數值。若 Reset 為 1 時，則輸出至 ALU 的 Multiplicand_out 重設為 0；若 W_ctrl 為 1 時，則暫存器會將被乘數值寫入進 ALU。

B. Arithmetic Logic Unit

```
module ALU(
    output reg [31:0] Result,
    output reg Carry,
    input [31:0] Src_1, Src_2,
    input [5:0] Funct,
    input clk
);
    assign Carry = 0;
    always @(*) begin
        case(Funct)
            6'b000001: {Carry, Result} = Src_1 + Src_2;
            6'b000000: {Carry, Result} = {1'b0, Src_1};
            default: {Carry, Result} = {Carry, Result};
        endcase
    end
endmodule
```

ALU.v 程式碼

ALU 將輸入訊號 Src_1 與 Src_2 根據 Funct 值做對應的運算。當 Funct 為”000000”時，將輸出結果 Result 重設為 Src_1 的值，並將

Carry 重整為 0；若 Funct 為”000001”時，將 Src_1 與 Src_2 相加，並將加法結果傳入 Result，進位位元傳入 Carry。

C. 64-bit Product Register

```
module Product(
    output reg [63:0] Product_out,
    output reg LSB,
    input [31:0] Multiplier_in,
    input [31:0] ALU_result,
    input ALU_carry,
    input SRL_ctrl,
    input W_ctrl,
    input Ready,
    input Reset,
    input clk
);
    reg pre_carry = 0;
    assign LSB = Product_out[0];
    always @(posedge clk) begin
        pre_carry <= ALU_carry;
        if (Reset) begin
            pre_carry <= 0;
            Product_out[31:0] <= Multiplier_in;
            Product_out[63:32] <= 32'b0;
        end
        else if(!Ready) begin
            if(W_ctrl) begin
                Product_out[63:32] <= ALU_result;
            end
            else if(SRL_ctrl) Product_out <= {pre_carry, Product_out[63:1]};
        end
    end
endmodule
```

Product.v 程式碼

Product Register 用於儲存乘積。為了節省元件，將乘數合併進暫存器的前 32-bit 裡，並透過 Controller 的控制，經過一系列演算後，會得到正確乘積。暫存器提供乘積的第一位元給控制器參考，使其得以決策。當 Reset 為 1 時，初始化暫存器，將旗標重設為 0，並將輸入的乘數傳進乘積的前 32-bit，並將後 32-bit 設為 0。程式演算未完成時，若 W_ctrl 為 1，則將 ALU 的結果寫進暫存器；若 SRL_ctrl 為 1 時，則將乘積向左移一位元。

D. Controller

```
module Control(
    output reg [5:0] Addu_ctrl,
    output reg SRL_ctrl, W_ctrl, Ready,
    input Run, Reset, clk, LSB
);
    integer count = 1;
    reg has_W_ctrl = 0;
    always @(posedge clk) begin
        if(Reset) begin
            Addu_ctrl <= 6'b0;
            SRL_ctrl <= 0;
            W_ctrl <= 0;
            Ready <= 0;
        end
        else if(Run) begin
            if(count > 32) Ready <= 1;
            else begin
                W_ctrl <= 1;
                SRL_ctrl <= 0;
                if (has_W_ctrl) begin
                    count <= count + 1;
                    Addu_ctrl <= (LSB) ? 6'b000001 : 6'b000000;
                    W_ctrl <= 0;
                    SRL_ctrl <= 1;
                    has_W_ctrl <= 0;
                end else has_W_ctrl <= 1;
            end
        end else count <= 0;
    end
endmodule
```

Control.v 程式碼

Controller 根據輸入、現在狀態與旗標，決定其輸出，使其控制各模組以完成乘法演算。控制器在每一 repetition 會先檢查乘積的第一位元是否為 1，若是，則向 ALU 下加法指令，令被乘數與乘積左半部相加，並將相加結果放回乘積左半部，再將整個乘積向右移；若不是 1，則直接將整個乘積向右移。每次 repetition 都會先將 W_ctrl 設為 1，使 Product Register 都能先重載上回計算後，ALU 輸出的結果。每次 repetition 都會計數迴圈次數，若超過 32 次，則代表運算完成，將 Ready 設為 1。

E. 32-bit Unsigned Complete Multiplier

甲、 主程式

```
module CompMultiplier(
    // Outputs
    output  [63:0]  Product,
    output      Ready,
    // Inputs
    input   [31:0]  Multiplicand, Multiplier,
    input       Run, Reset, clk
);
    wire [31:0] Src_1, Src_2, Result;
    wire [5:0] Funct;
    wire W_ctrl, LSB, SRL_ctrl, Carry;
    assign Src_1 = Product[63:32];
    Multiplicand MultiplicandRegister (
        .Multiplicand_out(Src_2),
        .Multiplicand_in(Multiplicand),
        .clk(clk),
        .Reset(Reset),
        .W_ctrl(W_ctrl)
    );
    Product ProductRegister (
        .Multiplier_in(Multiplier),
        .Product_out(Product),
        .LSB(LSB),
        .ALU_result(Result),
        .ALU_carry(Carry),
        .SRL_ctrl(SRL_ctrl),
        .W_ctrl(W_ctrl),
        .Ready(Ready),
        .Reset(Reset),
        .clk(clk)
    );
    ALU ALU (
        .Result(Result),
        .Carry(Carry),
        .Src_1(Src_1),
        .Src_2(Src_2),
        .Funct(Funct),
        .clk(clk)
    );
    Control Controller (
        .Addu_ctrl(Funct),
        .SRL_ctrl(SRL_ctrl),
        .W_ctrl(W_ctrl),
        .Ready(Ready),
        .Run(Run),
        .Reset(Reset),
        .clk(clk),
        .LSB(LSB)
    );
endmodule
```

CompMultiplier.v 程式碼

CompMultiplier 將各 module 透過內部接線，使對應接腳相連，組合成一個功能完整的乘法器。

乙、 Testbench and Simulation

```
`timescale 10 ns / 1 ns
// Declarations
`define DELAY           3   // # * 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, Run;
    reg [31:0] Multiplicand, 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)
    );
```

tb_CompMultiplier.v 程式碼—初始化

設定時間單位 10ns、時間精度 1ns。設定各常數與變數，並接線至 CompMultiplier 模組。

```
35      initial
36      begin : Preprocess
37          // Initialize inputs
38          Reset       = `LOW;
39          Run         = `LOW;
40          Multiplicand    = 32'd0;
41          Multiplier  = 32'd0;
42          // Initialize testbench files
43          input_file  = $fopen(`INPUT_FILE, "r");
44          output_file = $fopen(`OUTPUT_FILE);
45          #`DELAY;    // Wait for global reset to finish
46      end
47      always
48      begin : ClockGenerator
49          #`DELAY;
50          clk <= ~clk;
51      end
```

tb_CompMultiplier.v 程式碼—檔案處理與時鐘

Preprocess 負責將各變數設為 0，並開啟 tb_CompMultiplier.in、tb_CompMultiplier.out 檔案。

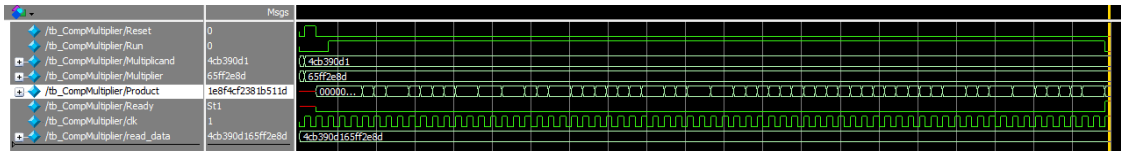
ClockGenerator 產生週期為 6 個時間單位的時鐘。

```
52      always
53      begin : StimuliProcess
54          // Start testing
55          while (!$feof(input_file))
56          begin
57              $fscanf(input_file, "%x\n", read_data);
58              @(posedge clk); // Wait clock
59              {Multiplicand, Multiplier} = read_data;
60              Reset = `HIGH;
61              @(posedge clk); // Wait clock
62              Reset = `LOW;
63              @(posedge clk); // Wait clock
64              Run = `HIGH;
65              @(posedge Ready);   // Wait ready
66              Run = `LOW;
67          end
68          #`DELAY;    // Wait for result stable
69          // Close output file for safety
70          $fclose(output_file);
71          // Stop the simulation
72          $stop();
73      end
74      always @(posedge Ready)
75      begin : Monitoring
76          $display("Multiplicand:%d, Multiplier:%d", Multiplicand, Multiplier);
77          $display("result:%d", Product);
78          $fdisplay(output_file, "%x", Product);
79      end
80  endmodule
```

tb_CompMultiplier.v 程式碼—模擬測試與輸出結果

StimuliProcess 將從 tb_CompMultiplier.in 讀取的測試向量，輸入至 CompMultiplier。每筆輸入資料都會等到 Controller 將 Ready 設為 1，才會測試下筆資料。

Monitoring 將運算結果輸出至終端，並一一儲存於 tb_CompMultiplier.out 檔。



tb_CompMultiplier.v 模擬波型

可以從波型看到乘積在演算過程中變化了很多次，直到最後算出結果為止。從結果來看 4CB390D1 * 65FF2E8D 確實是 1E8F4CF2381B511D，而且 Ready 也有在運算結束後變為 1，可以斷定乘法器運算功能正常。

Part2. Divider

A. 32-bit Divisor Register

```
1  module Divisor(
2      output reg [31:0] Divisor_out,
3      input reg [31:0] Divisor_in,
4      input Reset, W_ctrl, clk
5  );
6      always @(*) begin
7          if(Reset) Divisor_out = 32'b0;
8          else if(W_ctrl) Divisor_out = Divisor_in;
9      end
10 endmodule
```

Divisor.v 程式碼

Divisor Register 用於暫存來自輸入的除法數訊號，並將訊號傳遞給 ALU 運算。當 Reset 為 1 時，Divisor_out 會被重設為 0；當 W_ctrl 為 1 時，則 Divisor_out 會被設為輸入訊號 Divisor_in 的值。

B. Arithmetic Logic Unit

```v
module ALU(
    output reg [31:0] Result,
    output reg Carry, Neg_Rem,
    inout reg [31:0] Src_1,
    input [31:0] Src_2,
    input [5:0] Subu_ctrl,
    input clk
);
    assign Carry = 0, Neg_Rem = 0;
    assign Result = Src_1;
    reg signed [63:0] Result_signed;
    assign Result_signed = Result;
    always @(*) begin
        case(Subu_ctrl)
            6'b000000: {Carry, Result} = {Carry, Result};
            6'b000001: begin
                Carry = 0;
                Result = Src_1 + Src_2;
                Result_signed = Result;
            end
            6'b000010: begin
                Carry = 1;
                Result = Src_1 - Src_2;
                Result_signed = Src_1 - Src_2;
            end
            6'b000011: begin
                Carry = 0;
                Neg_Rem = 0;
                Result = Src_1;
                Result_signed = Result;
            end
            default: {Carry, Result} = {Carry, Result};
        endcase
        if (Result_signed < 0) begin
            Neg_Rem = 1;
        end
        else begin
            Neg_Rem = 0;
        end
    end
endmodule
```

ALU.v 程式碼

ALU 用於對兩輸入 Src_1、Src_2 的值做運算，並依接收到的 Funct 值，對應其指定的運算功能。此外，為使 Remainder Register 與 Controller 得知運算結果是否為負數，以判斷應對措施，而輸出 Neg_Rem 旗標。以下表格整理出各 Funct 值對應功能。

Funct (Binary)	對應功能
000000	使輸出結果與 Carry 保持原本狀態。
000001	相加兩輸入訊號，並輸出結果。使 Carry 保持為 1。
000010	相減兩輸入訊號，並輸出結果。使 Carry 保持為 0。
000011	重設所有輸出，使其回到初始狀態。

C. 64-bit Remainder Register

```
module Remainder(
    output reg [63:0] Remainder_out,
    input [31:0] ALU_result,
    input [31:0] Dividend_in,
    input ALU_carry, SLL_ctrl, SRL_ctrl, W_ctrl, Ready, Reset, clk, Neg_Rem
);
    reg signed [31:0] ALU_result_signed;
    reg SLL_Edge_Carry = 0;
    assign ALU_result_signed = ALU_result;
    always @(posedge clk) begin
        if (Reset) Remainder_out <= {32'b0, Dividend_in};
        else if(Ready) Remainder_out <= Remainder_out;
        else if (W_ctrl) Remainder_out <= {ALU_result, Remainder_out[31:0]};
        else if (SLL_ctrl) {SLL_Edge_Carry, Remainder_out} <= {Remainder_out[63:0], ALU_carry};
        else if (SRL_ctrl) Remainder_out <= {SLL_Edge_Carry, Remainder_out[63:33], Remainder_out[31:0]};
    end
endmodule
```

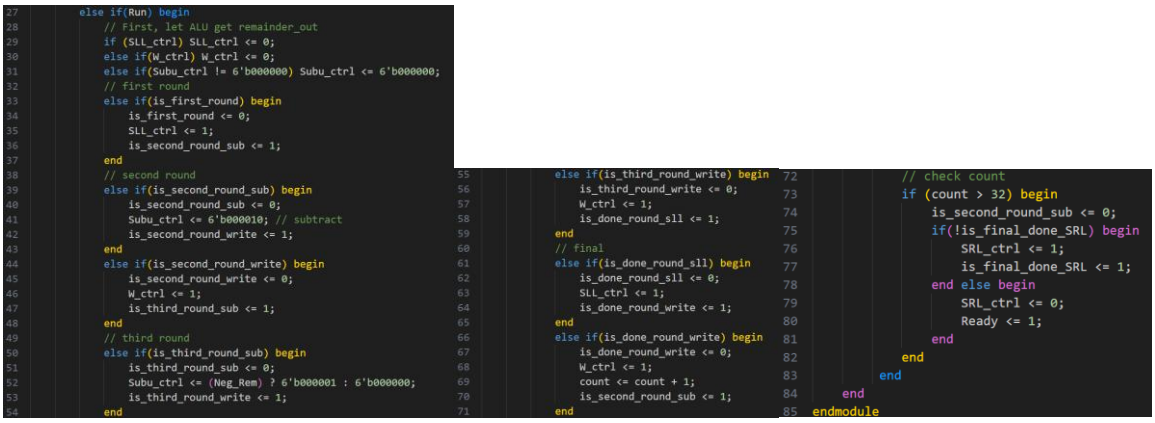
Remainder.v 程式碼

Remainder Register 用於暫存餘數與商的值，為節省空間，一開始會將被除數放入暫存器的後 32-bit 中。當 Reset 為 1 時，Remainder_out 會被重設至左半部為 0、右半部為被除數。若 Ready 為 1 時，代表整體除法運算完成，則 Remainder_out 保持原本狀態。若 W_ctrl 為 1 時，暫存器讀取 ALU 運算結果，並放入 Remainder_out 左半部，右半部則維持原狀。若 SLL_ctrl 為 1，則整體 Remainder_out 向左移；最右一位元會被移入 Carry，當 ALU 相減後結果為負數，則 Carry 為 0，若為正數，則 Carry 為 1。且左移後會保存被移出的位元，以確保最後右移時不會出錯。若 SRL_ctrl 為 1，則僅 Remainder_out 左半部往右移。

D. Controller

```
module Control(
    output reg [5:0] Subu_ctrl,
    output reg W_ctrl, SLL_ctrl, SRL_ctrl, Ready,
    input Run, Reset, clk, Neg_Rem
);
    reg is_initialize = 0, is_first_round = 1, is_second_round_sub = 0,
        is_second_round_write = 0, is_third_round_sub = 0,
        is_third_round_write = 0, is_done_round_sll = 0,
        is_done_round_write = 0, is_final_done_SRL = 0;
    integer count = 1;
    always @(posedge clk) begin
        if (Reset) begin
            Subu_ctrl <= 6'b000000;
            W_ctrl <= 1;
            SLL_ctrl <= 0;
            SRL_ctrl <= 0;
            Ready <= 0;
            is_initialize <= 0;
            is_first_round <= 1;
            is_second_round_sub <= 0;
            is_second_round_write <= 0;
            is_third_round_sub <= 0;
            is_third_round_write <= 0;
            is_done_round_sll <= 0;
            is_done_round_write <= 0;
            is_final_done_SRL <= 0;
            count <= 1;
        end
```

Control.v 程式碼



```
        else if(Run) begin
            // First, let ALU get remainder_out
            if (SLL_ctrl) SLL_ctrl <= 0;
            else if(W_ctrl) W_ctrl <= 0;
            else if(Subu_ctrl != 6'b000000) Subu_ctrl <= 6'b000000;
            // first round
            else if(is_first_round) begin
                is_first_round <= 0;
                SLL_ctrl <= 1;
                is_second_round_sub <= 1;
            end
            // second round
            else if(is_second_round_sub) begin
                is_second_round_sub <= 0;
                Subu_ctrl <= 6'b000010; // subtract
                is_second_round_write <= 1;
            end
            else if(is_second_round_write) begin
                is_second_round_write <= 0;
                W_ctrl <= 1;
                is_third_round_sub <= 1;
            end
            // third round
            else if(is_third_round_sub) begin
                is_third_round_sub <= 0;
                Subu_ctrl <= (Neg_Rem) ? 6'b000001 : 6'b000000;
                is_third_round_write <= 1;
            end
            else if(is_third_round_write) begin
                is_third_round_write <= 0;
                W_ctrl <= 1;
                is_done_round_sll <= 1;
            end
            // final
            else if(is_done_round_sll) begin
                is_done_round_sll <= 0;
                SLL_ctrl <= 1;
                is_done_round_write <= 1;
            end
            else if(is_done_round_write) begin
                is_done_round_write <= 0;
                W_ctrl <= 1;
                count <= count + 1;
                is_second_round_sub <= 1;
            end
            // check count
            if (count > 32) begin
                is_second_round_sub <= 0;
                if(!is_final_done_SRL) begin
                    SRL_ctrl <= 1;
                    is_final_done_SRL <= 1;
                end else begin
                    SRL_ctrl <= 0;
                    Ready <= 1;
                end
            end
        end
    end
endmodule
```

Control.v 程式碼

Controller 根據輸入值、當前狀態，判斷其應輸出的值，相當於個模組的樞紐。當 Reset 為 1 時，所有旗標及變數初始化。當 Reset 結束，開始執行程式時，會分各狀態一一執行。且每次 W_ctrl、SLL_ctrl 設為 1 後，會等待 1 個時鐘週期的時間，才進入下個狀態，這麼做是為了能讓 Remainder 及 ALU 正常讀入運算後的值。所有狀態執行完後，代表一個 repetition 的結束，這時會將計數器加一，並判斷迴圈次數有無大於 32。若有，則會先將 Remainder 左半部向右移，再將 Ready 設為 1；若無，則繼續進入下個循環，並從狀態二(second round)開始。

E. 32-bit Unsigned Complete Divider

甲、 主程式

```
module CompDivider(
    // Outputs
    output  [31:0]  Quotient,
    output  [31:0]  Remainder,
    output          Ready,
    // Inputs
    input   [31:0]  Dividend,
    input   [31:0]  Divisor,
    input           Run,
    input           Reset,
    input           clk
);
wire [63:0] Remainder_out;
wire [31:0] Divisor_out, ALU_result;
wire [5:0] Subu_ctrl;
wire W_ctrl, SLL_ctrl, SRL_ctrl, Neg_Rem, ALU_Carry;
assign Quotient = Remainder_out[31:0];
assign Remainder = Remainder_out[63:32];
Divisor DivisorRegister(
    .Divisor_out(Divisor_out),
    .Divisor_in(Divisor),
    .Reset(Reset),
    .W_ctrl(W_ctrl),
    .clk(clk)
);
Remainder RemainderRegister(
    .Remainder_out(Remainder_out),
    .Neg_Rem(Neg_Rem),
    .ALU_result(ALU_result),
    .Dividend_in(Dividend),
    .ALU_carry(ALU_Carry),
    .SLL_ctrl(SLL_ctrl),
    .SRL_ctrl(SRL_ctrl),
    .W_ctrl(W_ctrl),
    .Ready(Ready),
    .Reset(Reset),
    .clk(clk)
);
ALU ALU(
    .Result(ALU_result),
    .Neg_Rem(Neg_Rem),
    .Carry(ALU_Carry),
    .Src_1(Remainder_out[63:32]),
    .Src_2(Divisor_out),
    .Subu_ctrl(Subu_ctrl),
    .clk(clk)
);
Control Control(
    .Subu_ctrl(Subu_ctrl),
    .W_ctrl(W_ctrl),
    .SLL_ctrl(SLL_ctrl),
    .SRL_ctrl(SRL_ctrl),
    .Ready(Ready),
    .Run(Run),
    .Reset(Reset),
    .clk(clk),
    .Neg_Rem(Neg_Rem)
);
endmodule
```

CompDivider.v 程式碼

CompDivider 將各 module 以內部接線連接，組成一個完整的除法器。

乙、 Tesetbench and Simulation

```
`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;
    reg [31:0] Divisor;
    // Outputs
    wire [31:0] Quotient;
    wire [31:0] Remainder;
    wire Ready;
    // Clock
    reg clk = `LOW;
    // Testbench variables
    reg [63:0] read_data;
    integer input_file;
    integer output_file;
    integer i;
    CompDivider UUT(
        // Outputs
        .Quotient(Quotient),
        .Remainder(Remainder),
        .Ready(Ready),
        // Inputs
        .Dividend(Dividend),
        .Divisor(Divisor),
        .Run(Run),
        .Reset(Reset),
        .clk(clk)
    );
    initial
    begin : Preprocess
        // Initialize inputs
        Reset       = `LOW;
        Run         = `LOW;
        Dividend    = 32'd0;
        Divisor = 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
```

tb_CompDivider.v 程式碼—初始化、連接、預處理與時鐘設定

Testbench 一開始將時間單位設為 10ns、時間精度 1ns。初始化各常數與變數，並接線至 CompDivier。

Preprocess 將輸入至 CompDivider 的訊號初始化，並開啟 input 檔與 output 檔。

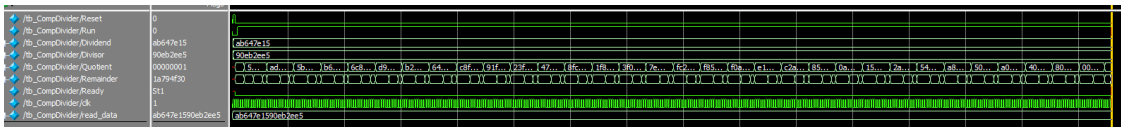
ClockGenerator 設定時鐘週期為兩個時間單位。

```
    begin : StimuliProcess
        // Start testing
        while (!$feof(input_file))
        begin
            $fscanf(input_file, "%x\n", read_data);
            @(posedge clk); // Wait clock
            {Dividend, Divisor} = 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:%d, Divisor:%d", Dividend, Divisor);
        $display("Quotient:%d, Remainder:%d", Quotient, Remainder);
        $fdisplay(output_file, "%x_%x", Quotient, Remainder);
    end
endmodule
```

tb_CompDivider.v 程式碼—模擬程序與輸出結果

StimuliProccess 將 input 檔的測試向量一一輸入至除法器中，在每個資料運算時，都會等到 Ready 變為 1，才會結束，送入下個訊號。

Monitoring 會將結果輸出至終端並記錄於 output 檔中。



tb_CompDivider.v 模擬波型

可以看到輸入的被除數為 90EB2EE5、除數為 AB647E15，其模擬後所得的商為 1、餘數為 1A794F30，跟理論值相符，且 Ready 也有在運算完成後被設為 1，可以判斷此除法器功能正常。

Part3. Conclution and Insight

乘法器與除法器的運作方式，因為要考慮到電路的限制與晶片的面積，所以沒有那麼的直覺。但仔細觀察這些算法的步驟，對比實際上用手去算，可以發現算法有它本身的邏輯在，在此可以感嘆能想出這些算法的人，他們是多麼的厲害。