PA1:Unsigned Multiplier and Unsigned Divider

Multiplier:

Area: $1125.978\mu m^2$ slack: 4.3324

Divider:

Area: $1250.2\mu m^2$ slack: 4.2589

學生：李勁磊

學號：B11107048

一、Screenshots and descriptions of each module:

1. 乘法器:

Multiplicand:

```
1  module Multiplicand(input clk, input Reset, input [31:0] Multiplicand_in,
2                      output wire [31:0]Multiplicand_out);
3      assign Multiplicand_out = (Reset == 1) ? Multiplicand_in : Multiplicand_out;
4  endmodule
5
```

1 ~ 2	設定基本的接腳
3	讓輸出在 Reset = 1 時設定 output 刷新

ALU:

```
1  module ALU(input [31:0] Src_1, input [31:0] Src_2,
2             output wire Carry, output wire [31:0] Result);
3      //assign {Carry, Result} = (flag == 1) ? (Src_1 + Src_2) : 33'b0;
4      // Internal wires
5      wire [8:0] Carry_internal;
6      wire [31:0] Sum;
7
8      assign Carry_internal[0] = 1'b0;
9
10     // Generate 8 CLA4 blocks
11     genvar i;
12     generate
13         for (i = 0; i < 8; i = i + 1) begin : CLA4_BLOCK
14             wire [3:0] A = Src_1[i*4 +: 4];
15             wire [3:0] B = Src_2[i*4 +: 4];
16             wire [3:0] G, P, C;
17
18             assign G = A & B;
19             assign P = A ^ B;
20
21             assign C[0] = Carry_internal[i];
22             assign C[1] = G[0] | (P[0] & C[0]);
23             assign C[2] = G[1] | (P[1] & G[0]) | (P[1] & P[0] & C[0]);
24             assign C[3] = G[2] | (P[2] & G[1]) | (P[2] & P[1] & G[0]) | (P[2] & P[1] & P[0] & C[0]);
25             assign Carry_internal[i+1] = G[3] | (P[3] & G[2]) | (P[3] & P[2] & G[1]) |
26                                          (P[3] & P[2] & P[1] & G[0]) | (P[3] & P[2] & P[1] & P[0] & C[0]);
27             assign Sum[i*4 +: 4] = P ^ C;
28         end
29     endgenerate
30
31     assign Result = Sum;
32     assign Carry = Carry_internal[8];
33
34 endmodule
```

1 ~ 2	宣告接腳
13 ~ 32	使用 CLA 電路來組成 32bit adder

Control:

```
1   module Control(input Run, input Reset, input clk,
2                  output reg Ready, output reg pre_finish);
3       reg [6:0] step_count;
4       initial begin
5           step_count = 0;
6           Ready = 0;
7       end
8       always@(*) begin
9           if(Reset == 1) begin
10              Ready = 0;
11              pre_finish = 0;
12          end
13          else if(step_count == 32) begin
14              pre_finish = 1;
15          end
16          else if(step_count == 33) begin
17              Ready = 1;
18          end
19      end
20      always@(posedge clk) begin
21          if(Reset == 1) begin
22              step_count = 0;
23          end
24          else if(Run == 1 && step_count < 33) begin
25              step_count = step_count + 1;
26          end
27      end
28  endmodule
```

1 ~ 3	宣告一些基本的接腳
8 ~ 28	輸出總共有 2 之接腳，Ready 跟 pre_finish，pre_finish 是用來穩定 output 訊號的，因為有些時候輸出會讀到上一個 state 的狀態

Product:

```
1  module Product(input [31:0]Multiplier_in, input pre_finish, input ALU_carry
2                 ,input [31:0] ALU_result, input Reset, input clk, input Run,
3                 input Ready, output reg [63:0] Product, output wire add_flag);
4      initial begin
5          Product = 64'b0;
6      end
7      assign add_flag = (Product[0] == 1) ? 1: 0;
8
9      always@(posedge clk) begin
10         if(Reset == 1) begin
11             Product <= {32'b0, Multiplier_in};
12         end
13         if(Run == 1 && pre_finish == 0) begin
14             if(Product[0] == 1) begin
15                 Product[63:31] <= {ALU_carry, ALU_result};
16                 Product[30:0] <= Product[31:1];
17             end
18             else begin
19                 Product[63:0] <= {1'b0, Product[63:1]};
20             end
21         end
22     end
23 endmodule
```

1 ~ 6	宣告一些基本的接腳
7	輸出是否需要進行加法的訊號，讓 ALU 可以先進行加法
9 ~ 22	如果在步數在 32 之前執行加法的功能，如果是 32 步以上就保值結果

2. 除法器：

Divisor:

```
1   module Divisor(input Reset, input [31:0] Divisor_in,
2       output wire [31:0] Divisor_out);
3       assign Divisor_out  = (Reset == 1) ? Divisor_in : Divisor_out;
4   endmodule
```

1 ~ 2	宣告一些基本的接腳
3	如果輸入 Reset = 1，那就刷新輸出，反之保持先前的結果

ALU:

```
1   module ALU(input [31:0] Src_1, input [31:0] Src_2,
2           output wire Carry, output wire [31:0] Result);
3       wire [31:0] B_invert;
4       wire [8:0] Carry_internal;
5       wire [31:0] Sum;
6
7       assign B_invert = ~Src_2;         // Step 1: 取反 B
8       assign Carry_internal[0] = 1'b1;  // Step 2: 加上 +1，完成 2's complement
9
10      // Look-ahead Adder 結構：Src_1 + ~Src_2 + 1
11      genvar i;
12      generate
13          for (i = 0; i < 8; i = i + 1) begin : CLA4_BLOCK
14              wire [3:0] A = Src_1[i*4 +: 4];
15              wire [3:0] B = B_invert[i*4 +: 4];
16              wire [3:0] G, P, C;
17
18              assign G = A & B;
19              assign P = A ^ B;
20
21              assign C[0] = Carry_internal[i];
22              assign C[1] = G[0] | (P[0] & C[0]);
23              assign C[2] = G[1] | (P[1] & G[0]) | (P[1] & P[0] & C[0]);
24              assign C[3] = G[2] | (P[2] & G[1]) | (P[2] & P[1] & G[0]) | (P[2] & P[1] & P[0] & C[0]);
25              assign Carry_internal[i+1] = G[3] | (P[3] & G[2]) | (P[3] & P[2] & G[1]) |
26                                           (P[3] & P[2] & P[1] & G[0]) | (P[3] & P[2] & P[1] & P[0] & C[0]);
27
28              assign Sum[i*4 +: 4] = P ^ C;
29          end
30      endgenerate
31
32      assign Result = Sum;
33      assign Carry  = ~Carry_internal[8];  // 若結果為正（MSB = 0），Carry = 1
34  endmodule
```

1 ~ 2	宣告接腳
10~33	使用 CLA 電路來組成 32bit 減法

Control:

```
1  module Control(input Run, input Reset, input clk,
2                 output reg Ready, output reg pre_finish);
3      reg [6:0] step_count;
4      initial begin
5          step_count = 0;
6          Ready = 0;
7          pre_finish = 0;
8      end
9      always@(*) begin
10         if(Reset == 1) begin
11             Ready = 0;
12             pre_finish = 0;
13         end
14         else if(step_count == 32) begin
15             pre_finish = 1;
16         end
17         else if(step_count == 33) begin
18             Ready = 1;
19         end
20     end
21     always@(posedge clk) begin
22         if(Reset == 1) begin
23             step_count = 0;
24         end
25         else if(Run == 1 && step_count < 33) begin
26             step_count <= step_count + 1;
27         end
28     end
29 endmodule
```

1 ~ 8	宣告一些 output 接腳跟 reg 的 initial 狀態
9 ~ 29	如果 Reset = 0 重置輸出接腳，根據次數輸出對應的接腳

Remainder:

```
1  module Remainder(input pre_finish, input Reset, input clk, input Ready,
2                   input Run, input [31:0] ALU_result, input [31:0] Dividend_in,
3                   input ALU_carry, output reg [63:0]Remainder_out);
4      reg step_count;
5      always@(posedge clk) begin
6          if(Reset == 1) begin
7              Remainder_out <= {31'b0, Dividend_in, 1'b0};
8              step_count <= 0;
9          end
10         else if(Run == 1 && pre_finish == 0) begin
11             if(ALU_carry == 0) begin
12                 Remainder_out[63:32] <= {ALU_result[30:0], Remainder_out[31]};
13                 Remainder_out[31:0] <= {Remainder_out[30:0], 1'b1};
14             end
15             else begin
16                 Remainder_out[63:0] <= {Remainder_out[62:0], 1'b0};
17             end
18         end
19         else if(pre_finish == 1 && Ready == 0) begin
20             if(step_count == 0) begin
21                 Remainder_out[63:32] <= {1'b0, Remainder_out[62:33]};
22                 step_count <= 1;
23             end
24         end
25     end
26 endmodule
```

1 ~ 4	宣告一些基本的接腳

5 ~ 25	如果 Run = 1 執行正常的除法，如果 pre_finish = 1 Ready = 0 執行餘數的左移，如果 Ready = 1 保持結果

二、Descriptions test commands for each module:

Multiplier test bench:

測試不同種 input 的情況：有正常的輸入、由 F 乘到最大值、乘零、最大乘最大的所有情況

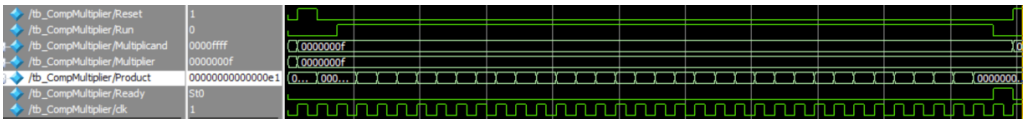
Divider test bench:

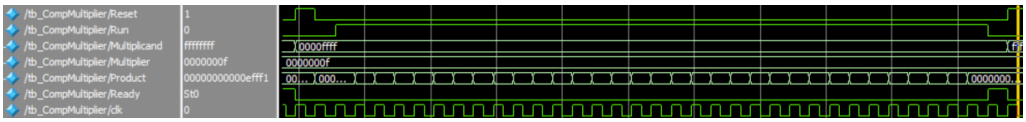
測試不同種 input 的情況：除數大於被除數、被除數大於除數、隨機數值的除法

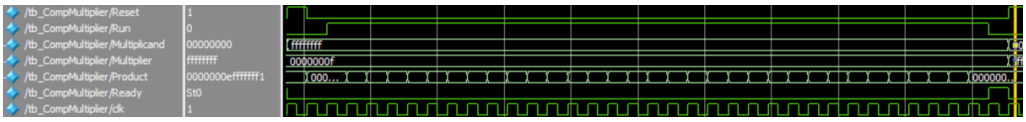
Multiplier Test Bench	Divider Test Bench
0000000F_0000000F 0000FFFF_0000000F FFFFFFFF_0000000F 00000000_FFFFFFFF FFFFFFFF_FFFFFFFF 12345678_12345678	FFFFFFFE_FFFFFFFF 0000000F_0000000F 0000FFFF_0000000F FFFFFFFF_0000000F 00000000_FFFFFFFF 0000000F_FFFFFFFF FFFFFFFF_FFFFFFFF 87654321_12345678

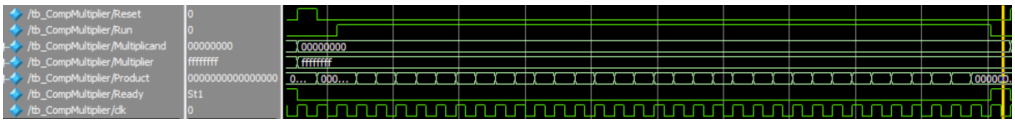
三、Stimulation Result:

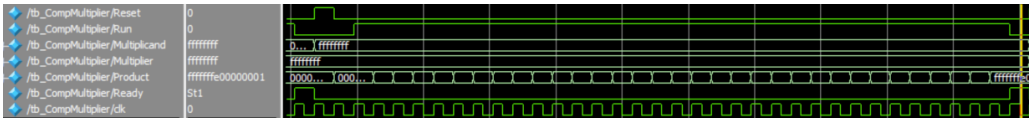
Multiplier:

Input Signal	Ideal output
0000000F_0000000F	00000000000000e1
實際波形輸出	
	

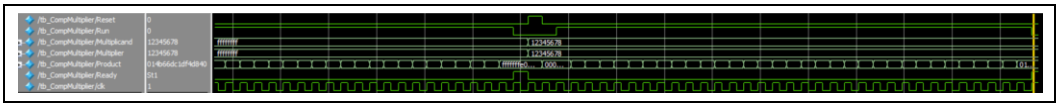
Input Signal	Ideal output
0000FFFF_0000000F	00000000000efff1
實際波形輸出	
	

Input Signal	Ideal output
FFFFFFFF_0000000F	0000000effffff1
實際波形輸出	
	

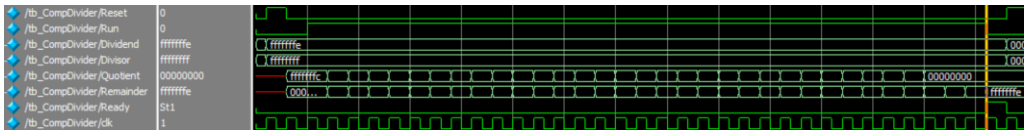
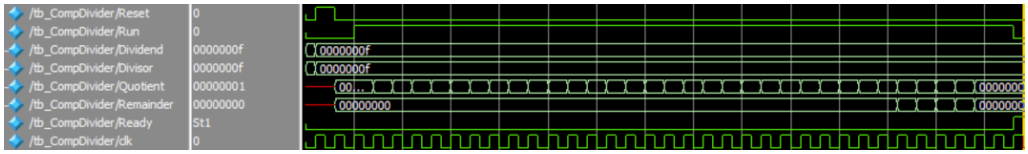
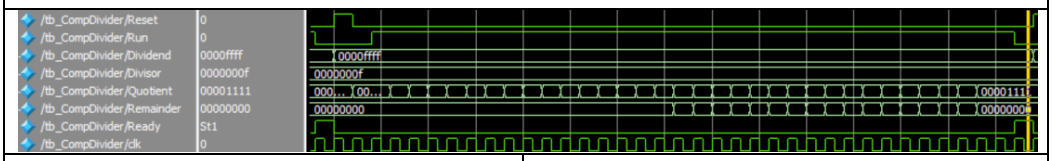
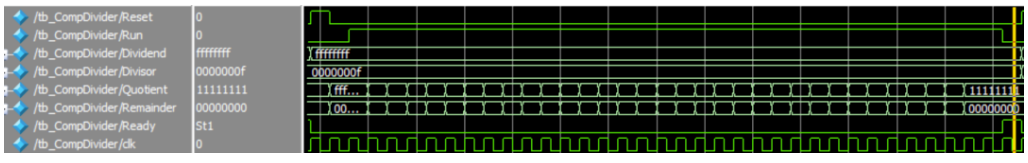
Input Signal	Ideal output
00000000_FFFFFFFF	0000000000000000
實際波形輸出	
	

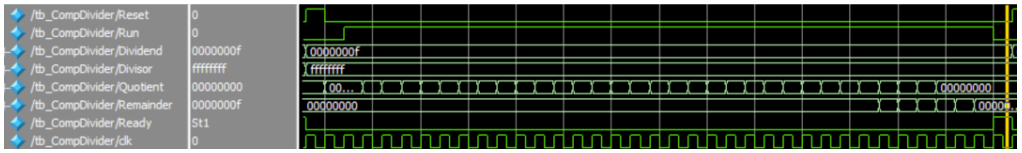
Input Signal	Ideal output
FFFFFFFF_FFFFFFFF	fffffffe00000001
實際波形輸出	
	

Input Signal	Ideal output
12345678_12345678	014b66dc1df4d840
實際波形輸出	



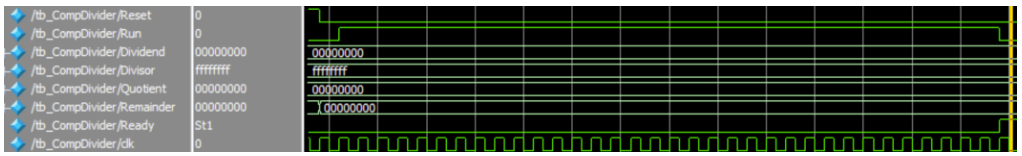
Divider:

Input Signal	Ideal output
FFFFFFFE_FFFFFFFF	00000000_fffffffe
實際波形輸出	
	
Input Signal	Ideal output
0000000F_0000000F	00000001_00000000
實際波形輸出	
	
Input Signal	Ideal output
0000FFFF_0000000F	00001111_00000000
實際波形輸出	
	
Input Signal	Ideal output
FFFFFFFF_0000000F	11111111_00000000
實際波形輸出	
	
Input Signal	Ideal output
0000000F_FFFFFFFF	00000000_0000000f
實際波形輸出	



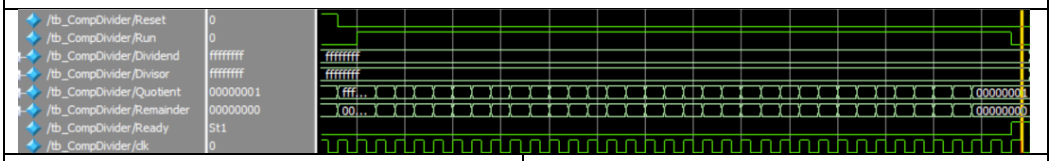
Input Signal	Ideal output
00000000_FFFFFFFF	00000000_00000000

實際波形輸出



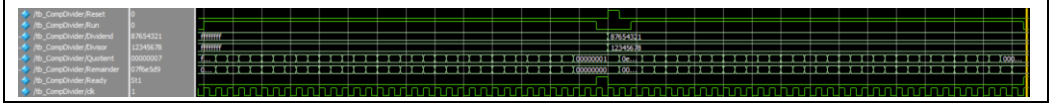
Input Signal	Ideal output
FFFFFFFF_FFFFFFFF	00000001_00000000

實際波形輸出



Input Signal	Ideal output
87654321_12345678s	00000007_07f6e5d9

實際波形輸出



四、Conclusion and insight on this homework.:

在這次做報告的過程中發生很多小插曲，我原本是打算使用 finite state machine 來進行編寫，然後已經寫好之後才知道原來會限制 clk 的數量，所以我之前的程式直接不能使用，後來我全部從新寫，花了很多時間才完成。

另外一個問題就是有些時候 Ready 訊號跳起來時，output 的數值也是正確的，但是在 tb_out 那個檔案中卻會發現輸出卻是上一個狀態的數值，所以最後我只好多一個 clk 來穩定輸出，讓 tb 讀到的數值是正確的。

最後我發現如果直接用 A+B 或是 A-B，ALU 的面積會變得很大，所以後來我詢問其他同學有沒有甚麼方法，所以我救參考他們的經驗加上 chatGPT，成功的寫出 look ahead 的加減法器。