9 RTL Verilog Modeling

9.1 實驗目的

- 練習使用 RTL Verilog modeling 語法，利用 16-bit Adder 完成 signed/unsigned 8-bit (V1) serial 乘法器設計。
- 練習使用 RTL Verilog modeling 語法，利用 8-bit Adder 完成 signed/unsigned 8-bit (V3) serial 乘法器設計。

9.2 實驗工具

名稱	說明
 Icarus Verilog	**Icarus Verilog** 是 Verilog 硬體描述語言的實現工具之一，其內建 GTKWave 可顯示輸出波型。它支持 Verilog 對應 IEEE 1995、IEEE 2001 和 IEEE 2005 三個不同的版本，並對 SystemVerilog 的部分內容提供支持。 Icarus Verilog 可以配置在 Linux、FreeBSD、OpenSolaris、AIX、Microsoft Windows 以及 OS X 環境中。該軟體以 GNU 通用公共許可協議發布，是一個自由軟體。 詳細內容請參考下列網址: http://iverilog.icarus.com/

9.3 實驗內容

同學應在先前 Lab 知道 RTL Verilog modeling 語法，本實驗則是要讓同學熟悉 RTL Verilog modeling 語法， RTL 是 Register Transfer Level 的縮寫，也就是暫存器轉換語言，這種寫法與 C、Java 等高階語言非常相似。本次實驗為使用 RTL Verilog modeling 實作 signed/unsigned 8-bit serial 乘法器，詳細的實驗內容如下：

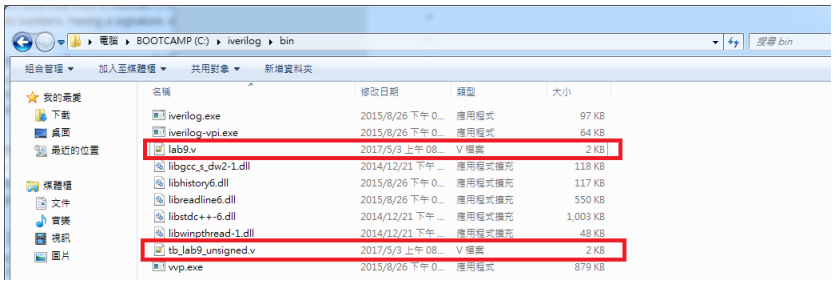
1. 使用 RTL Verilog modeling 語法實作 **unsigned** 8-bit serial 乘法器(使用 16-bit 加法器)。
2. 使用 RTL Verilog modeling 語法實作 **signed** 8-bit serial 乘法器(使用 16-bit 加法器)。

本次實驗目的是藉由實作 signed/unsigned 8-bit serial 乘法器熟悉 RTL modeling 語法，接下來我們將依序介紹本次實驗。

實驗一

本次實驗讓同學使用 RTL Verilog modeling 語法實作 unsigned 8-bit serial 乘法器 (V1 unsigned serial 乘法器)。首先介紹範例提供之 Verilog 程式，藉此讓同學了解 V1 unsigned serial 乘法器的架構，之後會講解如何依照 block diagram 的功能完成 V1 unsigned serial 乘法器(使用 16-bit 加法器)。

步驟一：下載助教在 e-course 上提供的範例檔案 Lab9，並將 lab9.v 及 tb_lab9_unsigned.v 複製到 C:\iverilog\bin 資料夾內。



步驟二：範例程式主要分為 4 個區塊，藍色區塊為執行初始化的動作，當 reset 信號觸發時需將所有的 register 清除為 0，黃色區塊為載入乘數及被乘數，紅色區塊在範例程式中還未填入程式碼，同學需填入加法與 shift 的指令，8-bit 的乘法器總共要做 8 次的加法及 shift，灰色區塊為做完運算後讓暫存器保持原態。

```
28  //Product
29  always @(posedge CLK or posedge RST)
30  begin
31          if(RST) begin
32                  Product    <=16'b0;
33                  Mplicand <= 16'b0;
34                  Mplier     <= 8'b0;
35          end
36          else if(Counter == 6'd0) begin
37                  Product    <=16'b0;
38                  Mplicand <= {8'b0, in_a};
39                  Mplier     <= in_b;
40          end
41          else if(Counter <=6'd8) begin
42                  /* write down your design below */
43
44
45
46                  /* write down your design upon */
47          end
48          else begin
49                  Product         <= Product;
50                  Mplicand        <= Mplicand;
51                  Mplier          <= Mplier;
52          end
53  end
```

觸發 reset 信號時，將儲存乘數、被乘數及積數的暫存器清除為 0。

將 input 的資料(in_a、in_b)接到暫存器中，之後便可使用暫存器做運算。

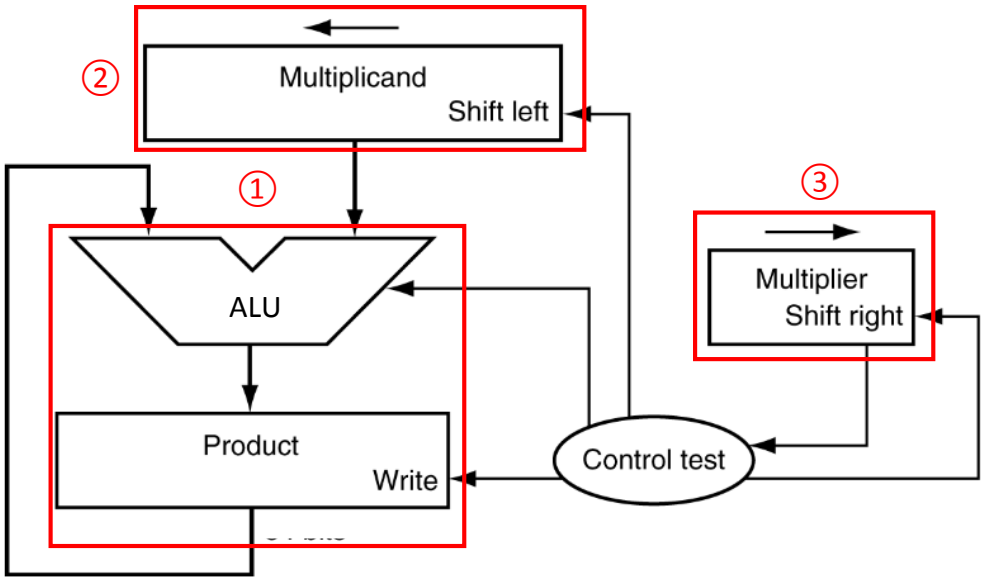
在此填入 shift 及加法等動作。

執行完乘法的動作後使暫存器維持原態。

在紅色區塊加入 V1 unsigned serial 乘法器(使用 16-bit 加法器)，首先判斷乘數的 LSB 位元是否為 1，如果是的話將被乘數與積數相加，之後被乘數左移 1-bit，乘數右移 1-bit。

```
41          else if(Counter <=6'd8)
42          begin
43                  /* write down your design below */
44                  if(Mplier[0] == 1'b1)
45              ①   Product <= Mplicand + Product;
46              ②   Mplicand <= Mplicand << 1;
47              ③   Mplier <= Mplier >> 1;
48                  /* write down your design upon */
49          end
```

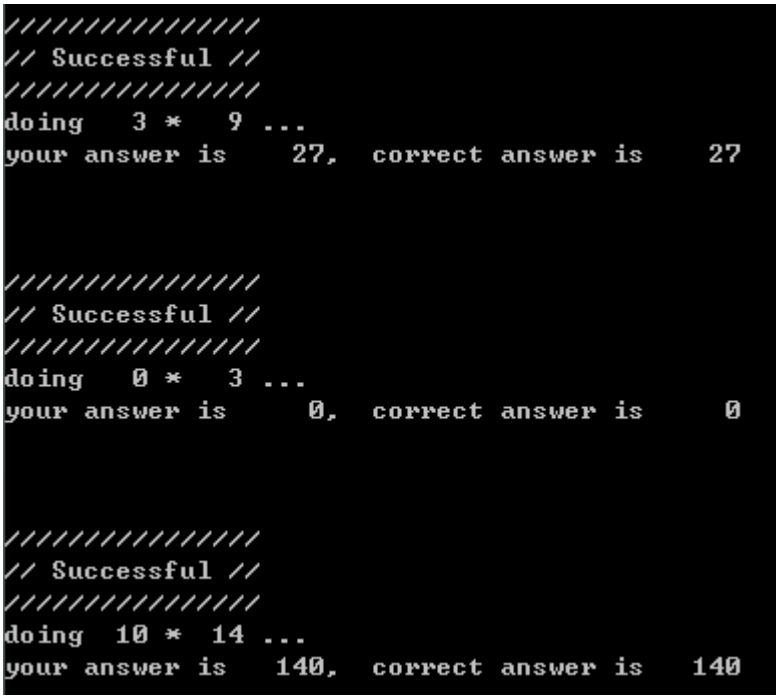
下圖為 V1 unsigned serial 乘法器的 block diagram，①為被乘數累加到積數，②被乘數左移，③每次都是以乘數的 LSB 來判斷是否執行 ALU 的動作，所以需右移乘數。



步驟三：使用 iverilog 指令編譯 tb_lab9_unsigned 與 lab9.v，之後使用 vvp 指令執行編譯完的結果，指令及執行結果如下：

iverilog –o lab9.out tb_lab9_unsigned.v

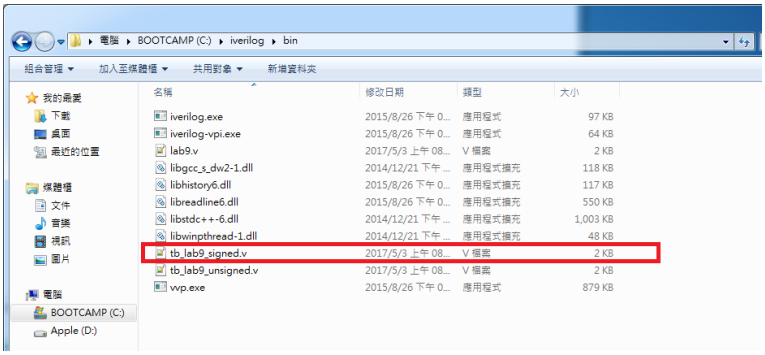
vvp lab9.out



實驗二

本次實驗為實作 signed 8-bit serial 乘法器(V1 signed serial 乘法器)，方法是在 V1 signed serial 乘法器的 input 加入正負數的判斷式，並根據判斷的結果決定是否對積數取補數。接下來會講解如何修改實驗一的 Verilog 的程式並完成 V1 signed serial 乘法器。

步驟一：下載助教在 e-course 上提供的範例檔案 Lab9，並將 tb_lab9_signed.v 複製到 C:\iverilog\bin 資料夾內。



步驟二：在實驗一的設計加入 signed 判斷，在乘法器的 input 與 output 加上正負數轉換，將 input 的數值全部轉成正數做乘法計算，並判斷計算後的積數為正數或負數，依照判斷結果修改 output 的數值。下圖為修改後的程式碼，按照藍色及黃色區塊修改程式。

```
36          else if(Counter == 6'd0) begin
37                  Product         <=16'b0;
38                  if(in_a[7] == 1'b1)
39                          Mplicand <= {8'b0,(~in_a) + 8'b1};
40                  else
41                          Mplicand <= {8'b0,in_a};
42                  if(in_b[7] == 1'b1)
43                          Mplier <= (~in_b) + 8'b1;
44                  else
45                          Mplier <= in_b;
46                  sign <= in_a[7] ^ in_b[7];
47          end
```

輸入的數值若為負數，則取 2 的補數。

判斷計算後結果的正負號。

使用 sign 變數來判斷積數的正負，若 sign 等於 0 表示積數為正，若 sign 等於 1 表示積數為負，則必須將積數取補數。下圖為修改後的程式碼，按照紅色區塊修改程式。

```
57          else begin
58                  if(sign)
59                  begin
60                          Product <= ~Product + 16'b1;
61                          sign    <= 1'b0;
62                  end
63                  else
64                          Product <= Product;
                    Mplicand<= Mplicand;
                    Mplier <= Mplier;
            end
```

依照 sign 來決定積數是正數或是負數。

步驟三：使用 iverilog 指令編譯 tb_lab9_signed.v 與 lab9.v，之後使用 vvp 指令執行編譯完的結果，指令及執行結果如下：

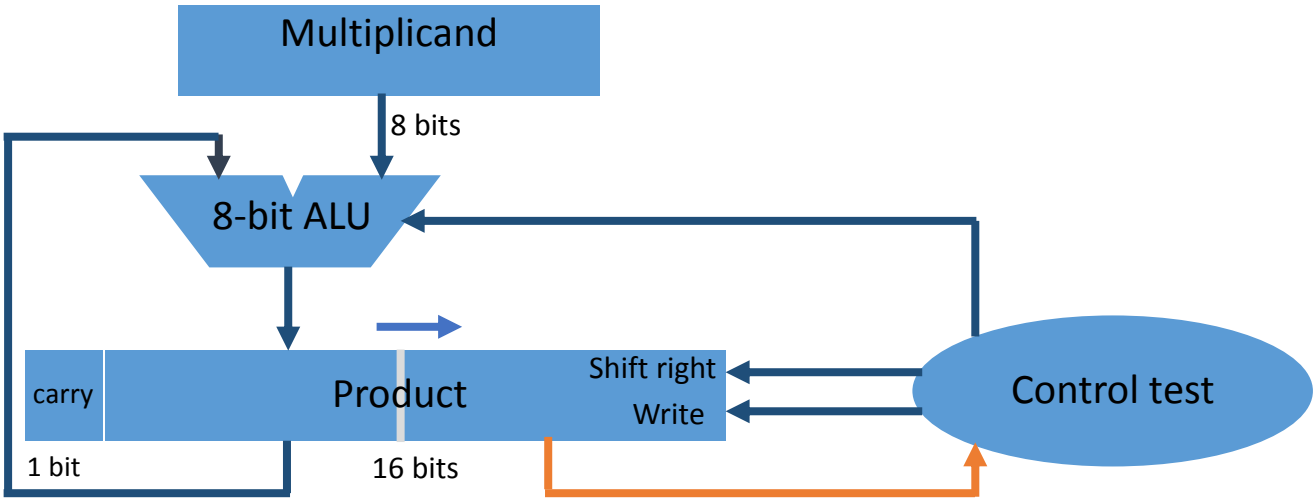
```
iverilog  –o  lab9.out  tb_lab9_signed.v
vvp                     lab9.out
```

```
////////////////
// Successful //
////////////////
doing   -3 *    9 ...
your answer is    -27,  correct answer is    -27



////////////////
// Successful //
////////////////
doing    0 *    3 ...
your answer is      0,  correct answer is      0



////////////////
// Successful //
////////////////
doing   10 *  -14 ...
your answer is   -140,  correct answer is   -140
```

9.4 練習題

本次壓縮檔的資料夾(v3_serial_mpy)中含有 tb_lab9_unsigned.v、tb_lab9_signed.v 與 lab9.v。

下圖為另一個版本的 serial multiplier(V3 serial multiplier)架構，此架構將 8-bit multiplier 合併至 16-bit product 的右半部，並使用 8-bit multiplicand 及 8-bit ALU 做乘法運算，此版本較省硬體資源。參考下圖，將 lab9.v 檔內 serial multiplier behavior 程式區塊改寫成此架構。



判斷 Product[0]是否為 1，如果為 1 則把 Multiplicand(4-bit)跟 Product[7:4]相加並右移;如果為 0，則直接右移，如下列範例所示。

V3 serial multiplier example (4-bit)

iteration	Step	Multiplicand	Product
0	Initial value	0010	0000_0011
1	1a: 1→Prod = Prod + Mcand	0010	0010_0011
	2: shift right Product	0010	0001_0001
2	1a: 1→Prod = Prod + Mcand	0010	0011_0001
	2: shift right Product	0010	0001_1000
3	1:0 →no operation	0010	0001_1000
	2: shift right Product	0010	0000_1100
4	1:0 →no operation	0010	0000_1100
	2: shift right Product	0010	0000_0110

執行結果如下:

```
// Successful //
////////////////
doing   32 *    9 ...
your answer is    288,  correct answer is    288



////////////////
// Successful //
////////////////
doing  100 *    0 ...
your answer is      0,  correct answer is      0



////////////////
// Successful //
////////////////
doing -101 *   14 ...
your answer is  -1414,  correct answer is  -1414



////////////////
// Successful //
////////////////
doing  123 *   -7 ...
your answer is   -861,  correct answer is   -861



////////////////
// Successful //
////////////////
doing   -1 *  -60 ...
your answer is     60,  correct answer is     60
```