HDL HW4 報告

B023040050 顏義洋

第一部分：撰寫組合語言將 RGB 轉為 YUV

檔案：YUV.v

R*0.299：

```
MOV r0, r_pixel
r3 = r0 >> 2
r4 = r0 >> 5
r5 = r0 >> 6
r6 = r0 >> 8
ADD r3,r3,r4
ADD r5,r5,r6
ADD r3,r3,r5
STR r3,data_memory[12]
```

G*0.589：

```
MOV r0, g_pixel
r3 = r0 >> 1
r4 = r0 >> 4
r5 = r0 >> 6
r6 = r0 >> 7
ADD r3,r3,r4
ADD r5,r5,r6
ADD r3,r3,r5
STR r3,data_memory[16]
```

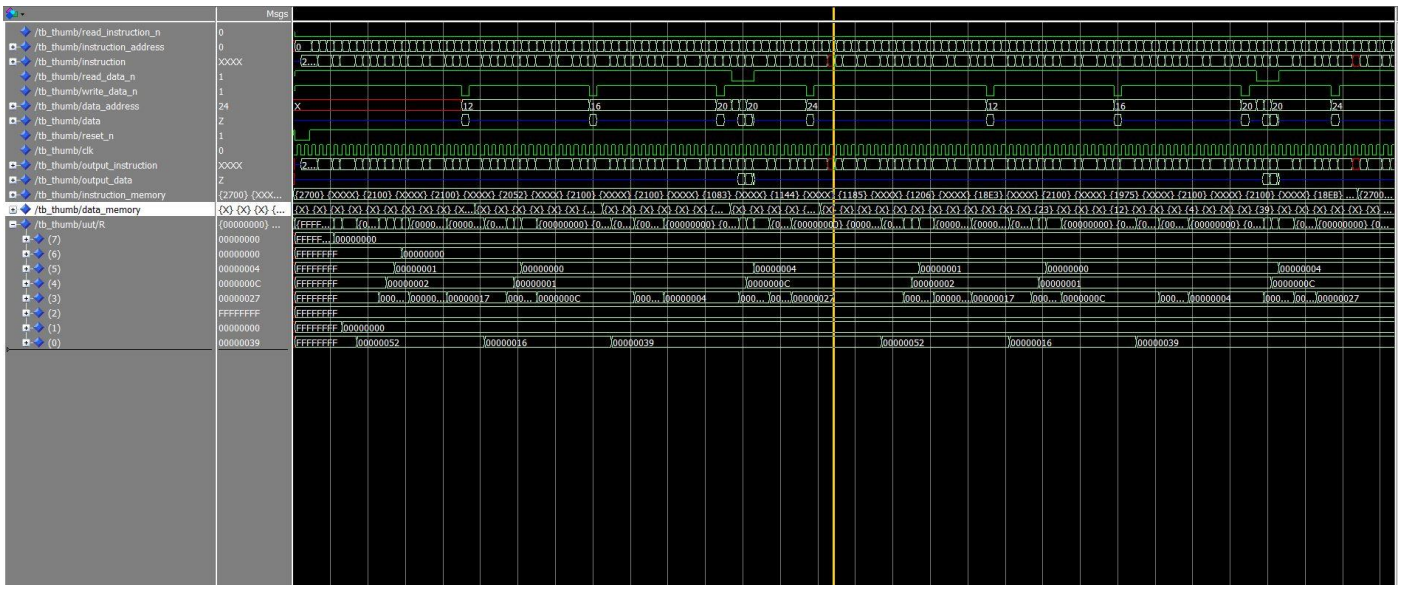
B*0.114：

```
MOV r0, b_pixel
r3 = r0 >> 4
r4 = r0 >> 5
r5 = r0 >> 6
r6 = r0 >> 7
ADD r3,r3,r4
ADD r5,r5,r6
ADD r3,r3,r5
STR r3,data_memory[20]
```

Y=R+G+B：

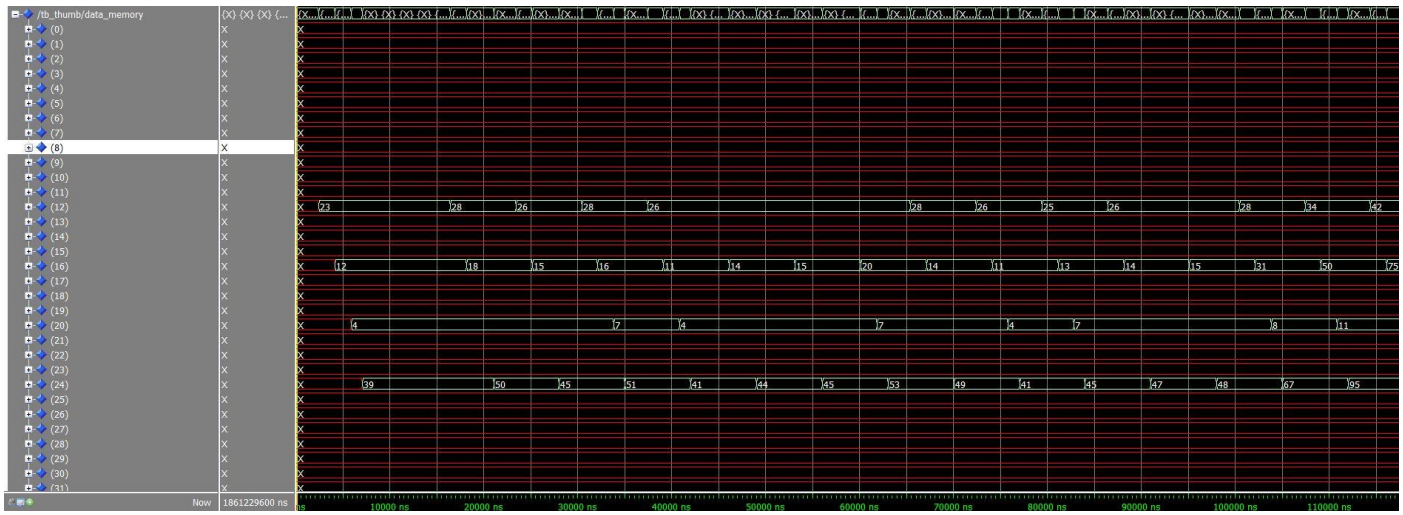
LDR r3, data_memory[12]
LDR r4, data_memory[16]
LDR r5, data_memory[20]
ADD r3,r3,r4
ADD r3,r3,r5
STR r3,data_memory[24]

B #0; 回到 pc=0 繼續跑



▲圖一

▲圖二



▲圖三

運算流程：

(1)把 R mov 到 r0，再對 r0 作數個 shift 分別存到不同 reg，最後把總合存到 r3

(2)把 R*0.299 的近似值 store 到 data_memory[12]

(3)G*0.589、B*0.114 的方法同(1)(2)，結果分別 store 到 data_memory[16]，data_memory[20]

(4)計算 Y = R*0.299 + G*0.589 + B*0.114 存到 data_memory[24]

Testbench 流程：

(1) 開啟 input 的 BMP，讀取 header 取得 bmp 的長跟寬還有檔案大小和 pixel data 的起始點。一次放 3 個 byte：把 R、G、B 分別 mov 到 r0

(2) 此時 Y 值已經在 data_memory[24]

(3) 一次寫 3 個 byte 的 Y 值進圖片檔

(4) LOOP 以上(2)~(4)的步驟直到讀檔結束

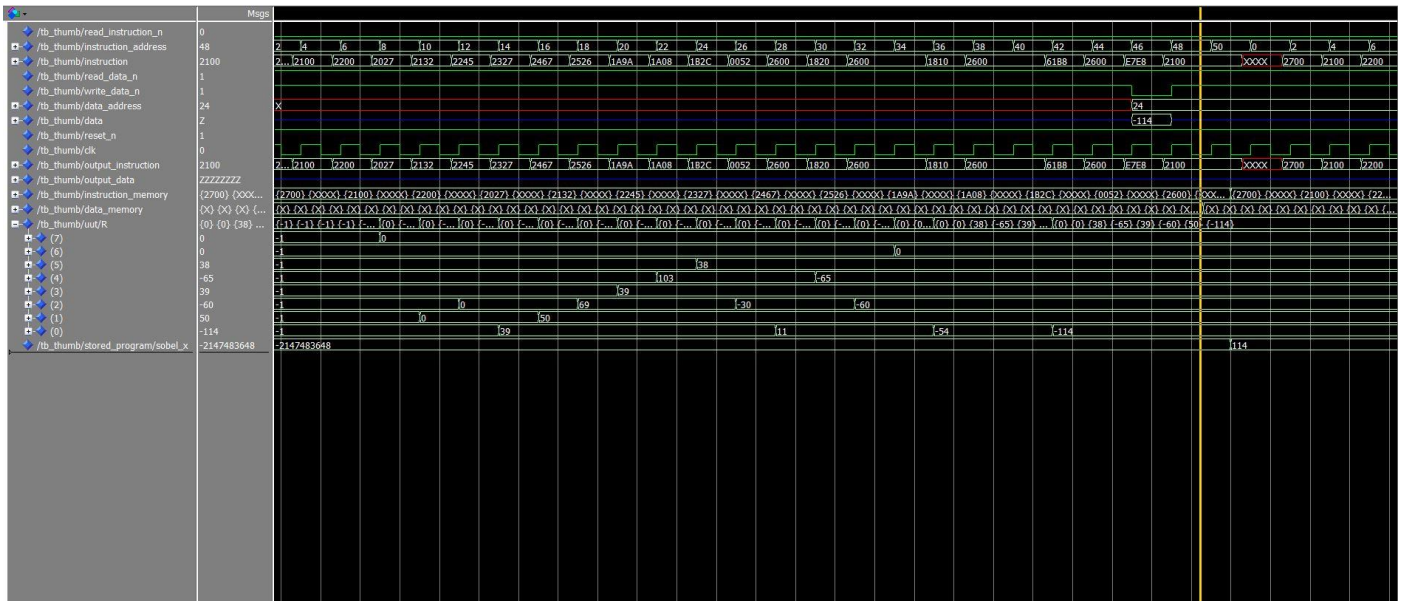
以上流程可以參考圖 1,2,3。

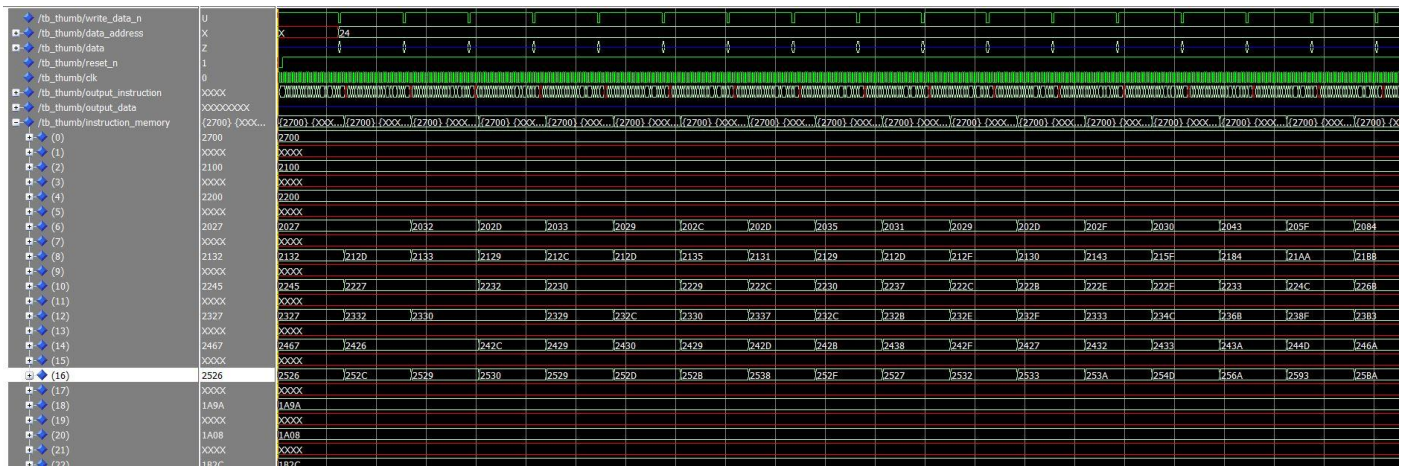
第二部分：撰寫組合語言執行 sobel 運算

檔案：sobel.v

mov r0, bmp_data(3*(bmp_width*(y-1)+(x-1)))
mov r1, bmp_data(3*(bmp_width*(y-1)+(x+1)))
mov r2, bmp_data(3*(bmp_width*y+(x-1)))
mov r3, bmp_data(3*(bmp_width*y+(x+1)))
mov r4, bmp_data(3*(bmp_width*(y+1)+(x-1)))
mov r5, bmp_data(3*(bmp_width*(y+1)+(x+1)))
SUB r2,r3,r2;
SUB r0,r1,r0;
SUB r4,r5,r4;
LSL r2,#2
ADD r0,r0,r4
ADD r0,r0,r2

B #0; 回到 pc=0 繼續跑





運算流程：

$$\begin{bmatrix} r0 & 0 & r1 \\ r2 & 0 & r3 \\ r4 & 0 & r5 \end{bmatrix}$$

(1) 將 sobel 所需要的 Y 值分別從 mov 進 r0~r5 內

(2) 利用加減和 shift 運算，算出所需要的梯度值

(3) 把算出來的梯度值放入 data_memory[24]

Testbench 流程：

(1) 開啟 input 的 BMP，讀取 header 取得 bmp 的長跟寬還有檔案大小和 pixel data 的起始點。

(2) 開個很大個陣列把所有的 BMP pixel 的 data 先讀進來

(3) 最外圍的那一圈 pixel 不做 sobel 運算，寫檔時直接寫黑色

(4) 把相對應的 pixel 放進 data_memory

(5) 此時 sobel 算出來的梯度已經在 data_memory[24]

(6) 因為在 thumb 內沒有類似 arm conditional execution 的指令，所以不方便在 thumb 內就直接算出要寫白色還是黑色的 pixel 進圖片

(7) 將出來的梯度取絕對值

(8) 選定 64 為標準，梯度大於 64 寫白色(255,255,255)進圖片，反之填入黑色(0,0,0,)

(9) 用兩層 LOOP 執行以上(3)~(8)的步驟直到所有的 pixel 都算完

結果展示：

寫作業遇到的所有問題 or 心得：

(1) VHDL 的資料型態不太好用花很多時間研究，特別是型態轉換的部分。

(2) VHDL 讀取 binary 方法網路上找的都是用 std_logic_vector，但是這份作業比較適合用 char 讀取。

(3) 因為老師給的 thumb 的 data_memory 因為是 signal 的關係，所以在 testbench 內無法直接塞值，因為這樣好像會 multi assign 造成無法使用，所以餵資料進 thumb 都是採用 mov 的方式。

(4) Thumb VHDL 的版本 同樣的組語 算出來的值可能會差 1~2 單位。

(5) 由於是用 mov 丟資料的關係，整體 testbench 的 delay 要抓得很剛好，不然可能會從 data_memory 抓出來的資料是錯的。

(6) 原本 verilog 版本的 thumb 我都是用 reset 讓 pc 歸 0 但是 VHDL 版本不行，所以組語要改用 branch 指令跳回 pc=0