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]

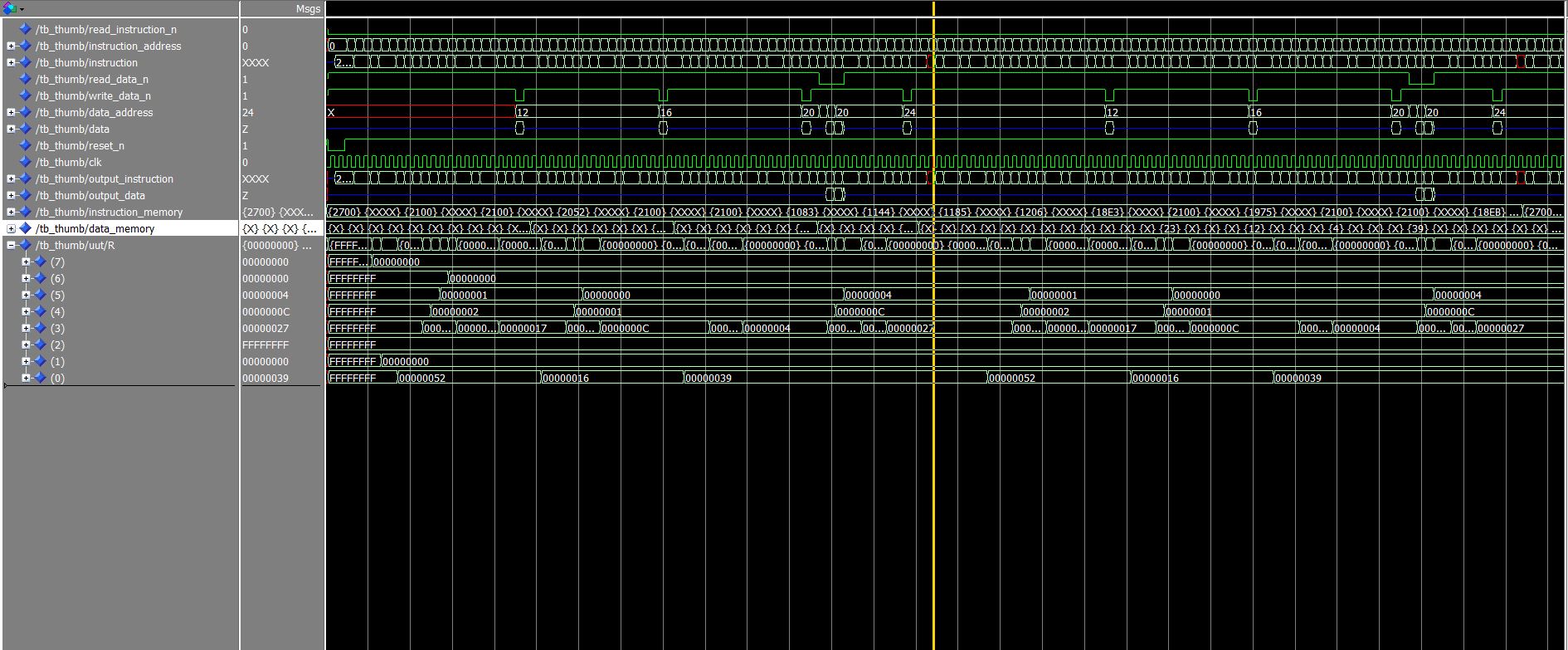
LDR r5, data\_memory[20]

ADD r3,r3,r4

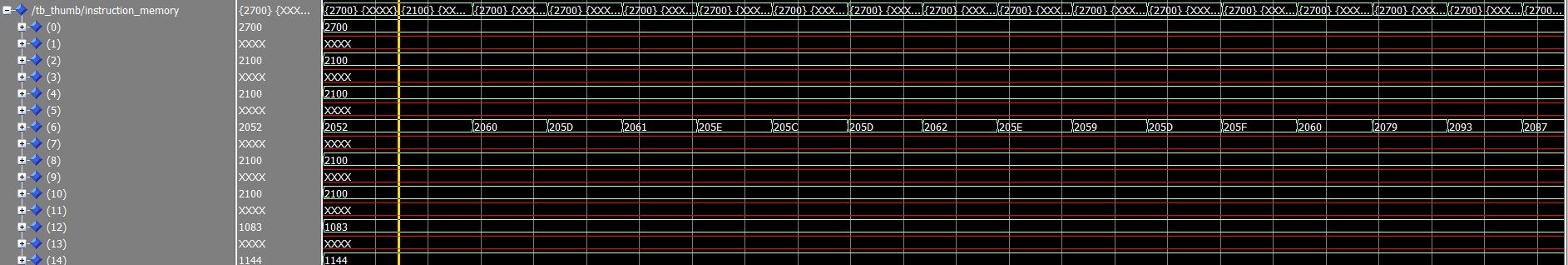
ADD r3,r3,r5

STR r3,data\_memory[24]

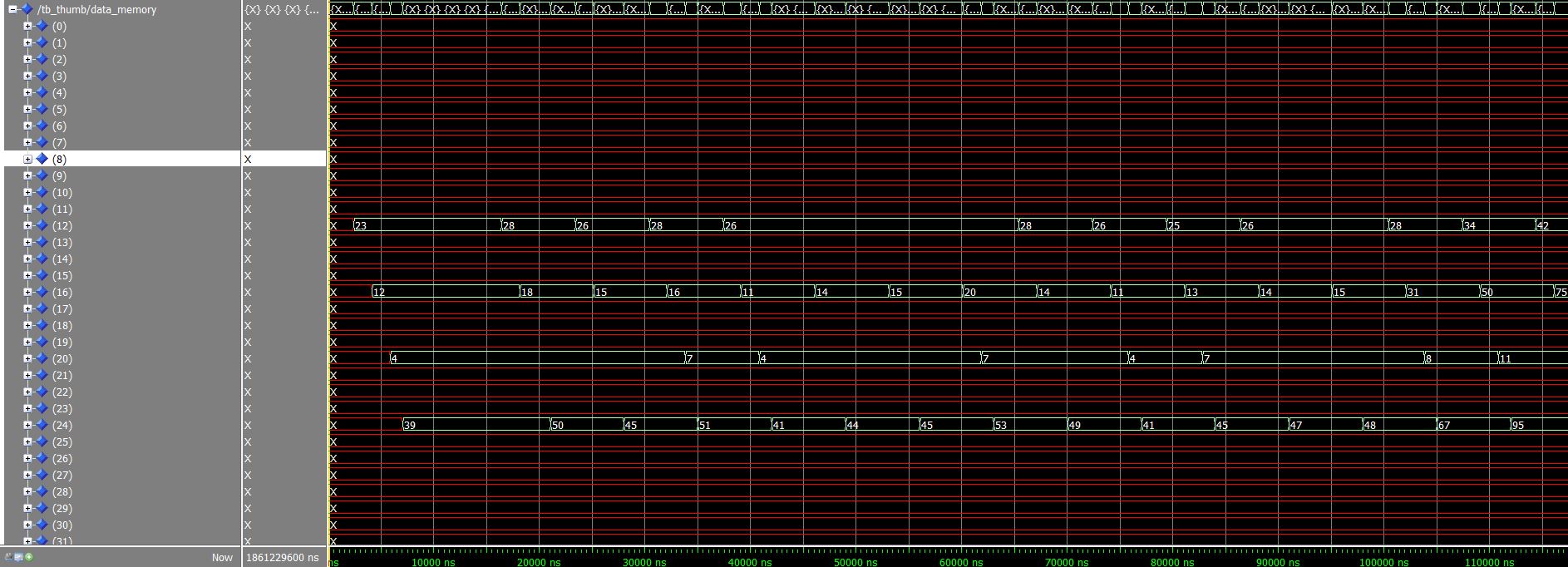
B #0; 回到pc=0繼續跑



▲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;

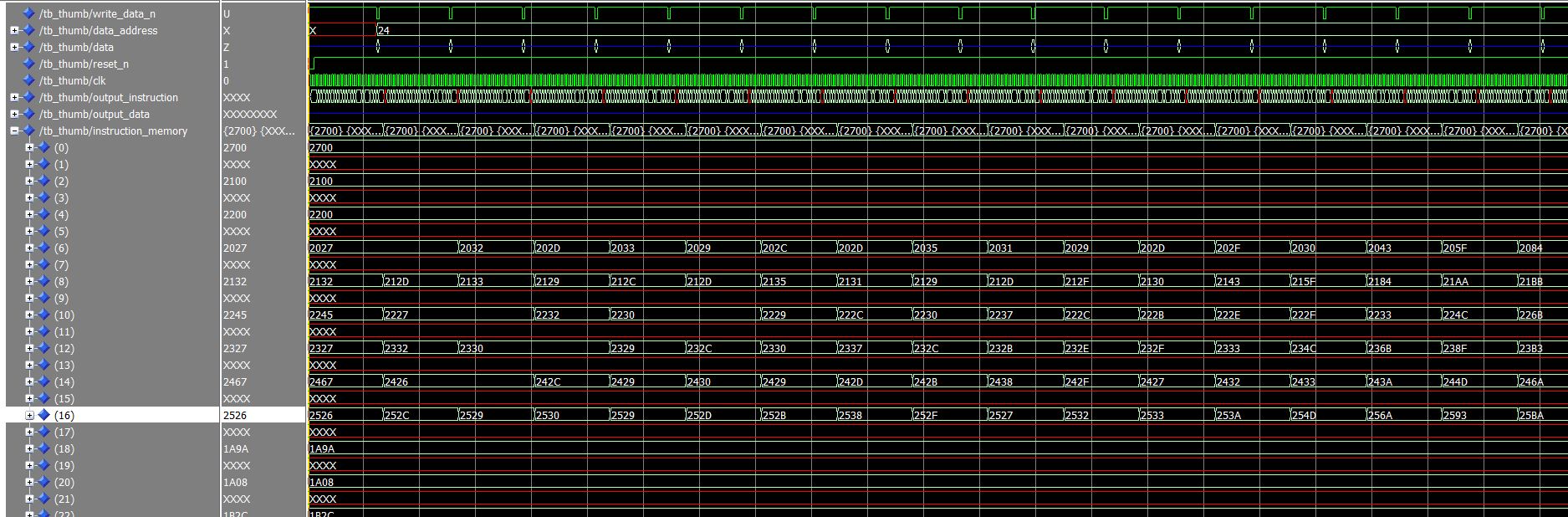
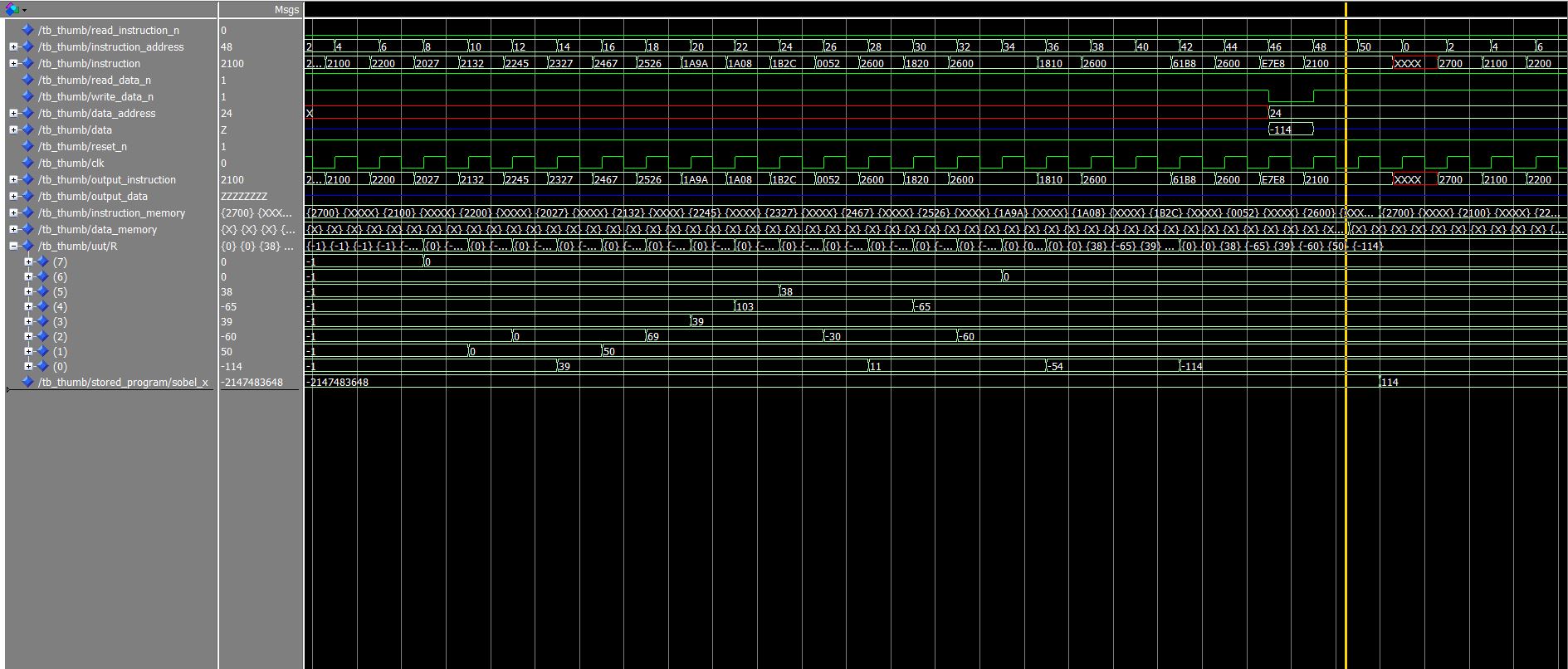
SUB r4,r5,r4;

LSL r2,#2

ADD r0,r0,r4

ADD r0,r0,r2

B #0; 回到pc=0繼續跑



運算流程：



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