Digital System Design
Final Project – Sobel Edge Detection

104062229 毛羿宣

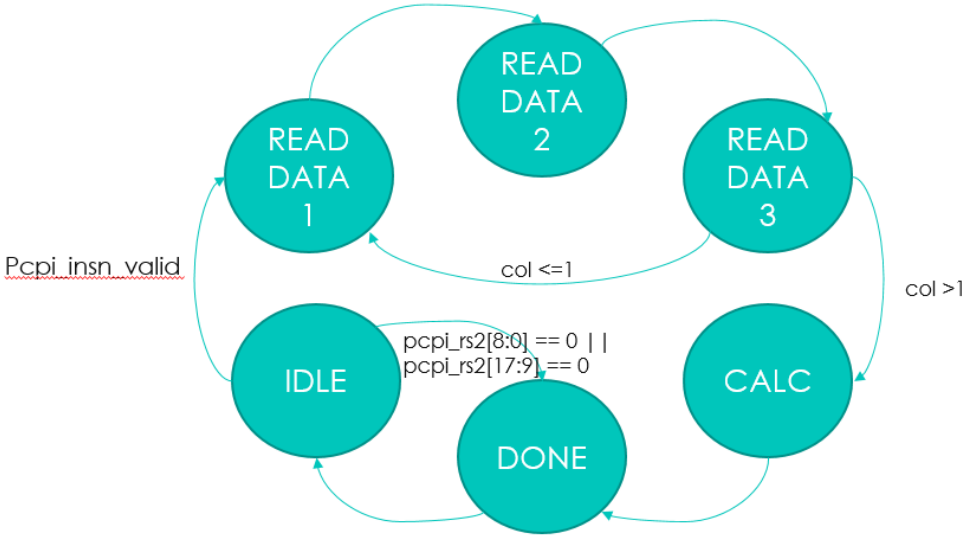
一、題目說明:

Final 我所要實作的是 Sobel Edge Detection。Sobel Edge 是利用兩個 kernel Gx、Gy 算出圖片的水平梯度與垂直梯度，分別取絕對值相加，來求出最後圖片的 edge。

$$Gx = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \qquad Gy = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

$$G = |Gx| + |Gy| \text{ (或 } \sqrt{Gx^2 + Gy^2} \text{)}$$

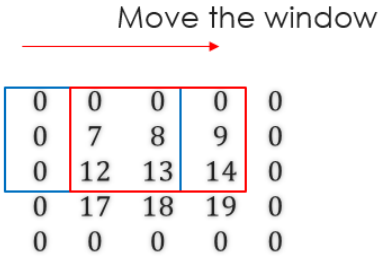
二、實作過程:



↑ Finite State Machine

A. Data 處理:

由於我的 kernel size 為兩個 3*3 的 window，所以資料讀取的時候需要讀進來九個 data。



由圖中可以看出來，每一次滑動 window 的時候，右邊兩排的數值會往左移動一格，利用這個性質，每次計算 pixel 時不需要再重新讀取九個 data，只需要再讀取下一排的三個 data 即可。

在軟體的部分，我先將圖片(bmp 檔)轉成 .bin 檔，再放到工作站上存到 memory 中。為了方便之後的計算，我將圖片最外圈的 RGB value 設成 0，如此可以避免 window 滑動的時候會造成 boundary 的問題。

```
READ_DATA1:begin
//shift data
    next_data[0] = data[1];
    next_data[1] = data[2];
    next_data[3] = data[4];
    next_data[4] = data[5];
    next_data[6] = data[7];
    next_data[7] = data[8];
//get the first data
    next_data[2] = mem_rdata;
    mem_valid = 1;
    next_row = row + 9'd1;
    next_addr = {14'd0, row,col};
    next_state = READ_DATA2;
end
```

```
READ_DATA3:begin
    next_data[8] = mem_rdata;
    mem_valid = 1;
    if(col <=1)begin
        next_row = row - 9'd2;
        next_col = col+1;
        next_addr = {14'd0, row,col};
        next_state = READ_DATA1;
    end
    else begin
        next_state = CALC;
    end
end
```

如果是第一個 column 代表要重讀 9 個 data

B. 計算:

軟體:

因為在 memory 中取位置時，實質上為一維陣列，故在軟體的部分我用兩層 for 迴圈，將 row、column 的值用 pcpi_rs2 傳進去 hardware。又因為我的圖片維 512*512 = 2^9*2^9，則傳入的值可以被 decode 成 9 bit 的 row 跟 9 bit 的 column，模擬成二維陣列。

```
while(test < 2){
        for(i = 0;i < 512;i++){
            for( j = 0;j<512;j++){
                color[i][j]=hard_sobel_pcpi(test,i*512+j);
        }
    }
    test++;
}
```

硬體:

當 state 到 CALC 時會將 window 與 Gx、Gy 對應位置相乘相加，並傳到 abs 這個 module 中做絕對值相加，到 DONE 的時候計算結束，回傳結果。最後輸出的時候，我設置了一個 threshold 來過濾一些顏色，讓邊界變的更明顯。當輸出的值 > threshold，則維持原本的顏色，反之則設為 0 (黑色)。

```
module abs (input[31:0] A,
            input[31:0] B,
            output[31:0] ans);
    wire[31:0] a = (A[31])?(~A[31:0]+1):A;
    wire[31:0] b = (B[31])?(~B[31:0]+1):B;
    assign ans = a+b;
```

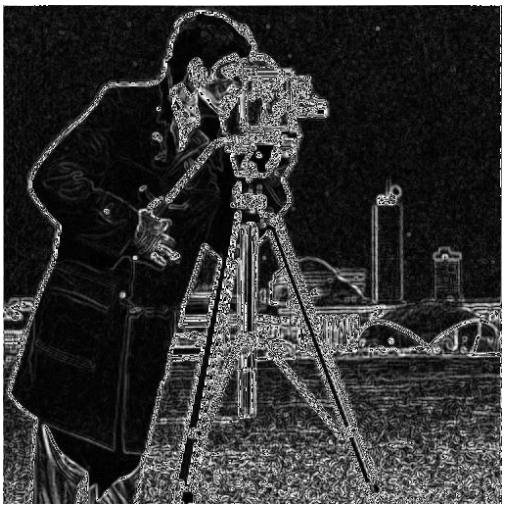
↑ abs module

```
answer <= (next_answer>`THERHOLD)?next_answer:32'd0;
```

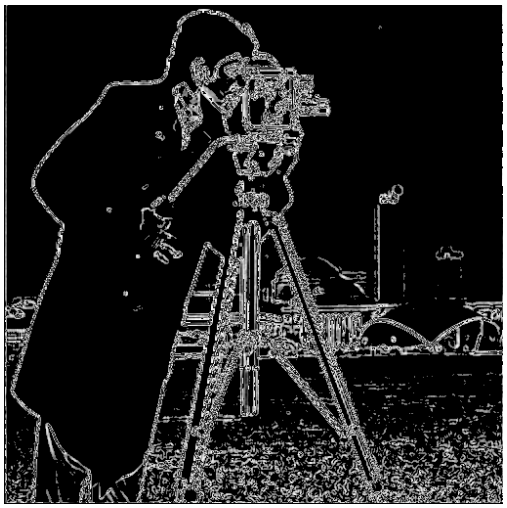
C. 檔案輸出:

為了能夠顯示圖片，我在 DONE 的時候將 data 放入 .bin 檔輸出，當讀到最後一個 pixel 時，檔案就可以關閉。最後將圖片利用 image_gen 轉成 bmp 圖檔。

三、實作結果:

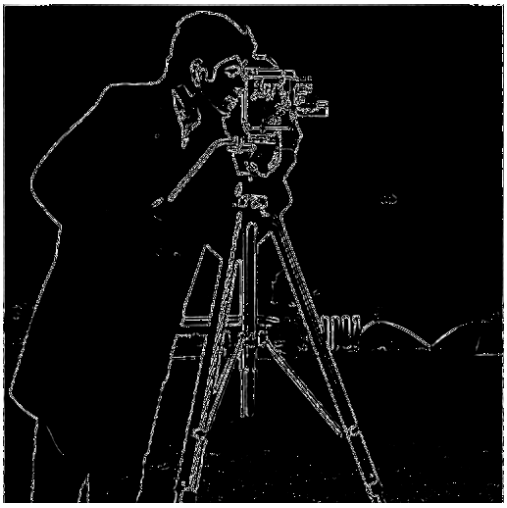


↑ No Threshold



↑ Threshold = 100

↑ Threshold = 175



↑ Threshold = 250

計算時間:

```
test_image: 0
Elapsed: 4264531387
test_image: 1
Elapsed: 4264531387
```

上圖為硬體計算，軟體控制 row、column 的結果。由於在工作站上實作軟體部分的時候發現光要將值放入 512*512 大小的陣列就需要花到很多時間，故最後選擇只用硬體計算。

四、心得:

Final project 中最難的部分我覺得就是處理 data 的地方。因為在 picoRV 裡面，並不能使用平常 C 有的 library，故在讀檔與輸出檔案的地方只能先用 testbench 放到 memory 中，計算時在從 memory 讀取 (這樣也會使要在工作站上做軟體計算 data 的話，會變非常的慢)，而輸出檔案的地方也只能用 Verilog 的 fwrite，算完 data 後直接輸出，與軟體的互動就變少了。

一開始讀取 data 的時候，一直在想怎麼解決邊界的問題，後來發現可以先把圖檔再轉 .bin 的時候，就將外圍的 pixel 設成 0，也就是其實圖片是 511 * 511，就從 (1,1) 開始計算，就不會讀到奇怪的 data 了。