HDL Digital Design (Graduate Level)
Spring 2024

HOMEWORK
REPORT

Must do self-checking before submission:

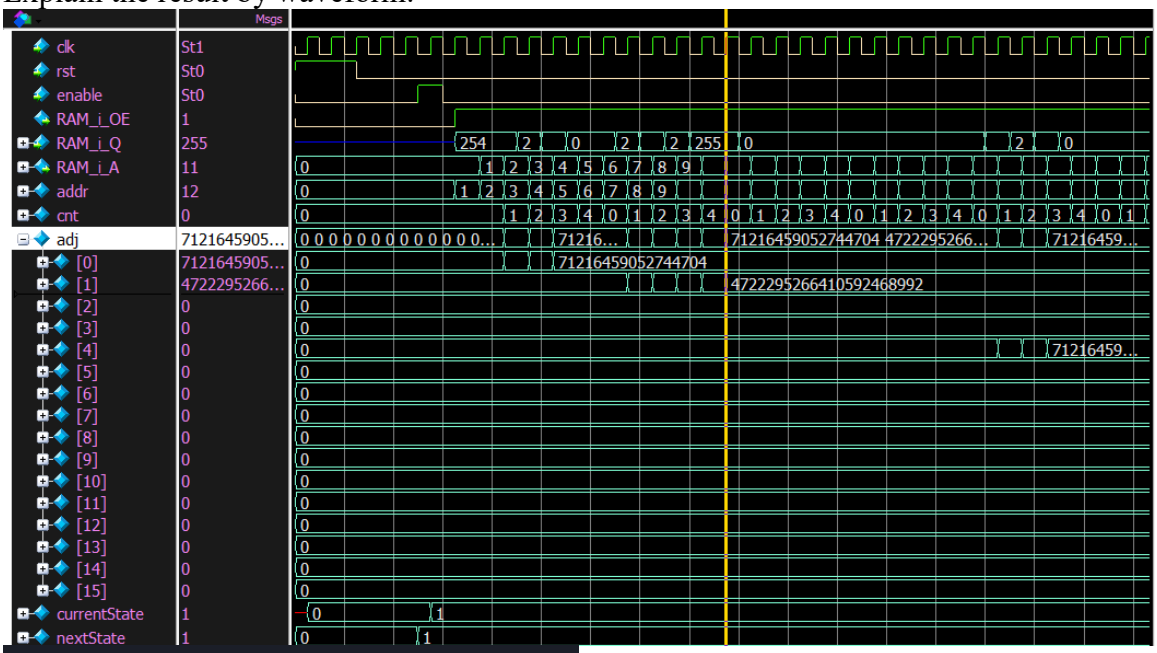
☐ Compress all files described in the problem into one zip file.
☐ All files can be compiled under ModelSim environment.
☐ All port declarations comply with I/O port specifications.
☐ Organize files according to File Hierarchy Requirement
☐ No waveform files or project file in deliverables

Student name: 蔡承哲
Student ID: Q36111150

1. Your simulation result on the terminal (Transcript) .

```
# ********************************
# ============ Pattern 1 PASS !!! ============
# ============ Pattern 2 PASS !!! ============
# ============ Pattern 3 PASS !!! ============
#
#
#  ******************************
#  **                          **        |__||
#  **  Congratulations !!      **       / O.O  |
#  **                          **     /_____   |
#  **  Simulation PASS!!       **    /^ ^ ^ \  |
#  **                          **   |^ ^ ^ ^ |w|
#  ******************************    \m___m__|_|
#
#
# ** Note: $finish    : D:/00_second_under/StudentID_Lab7/tb_PDC.sv(166)
#    Time: 21630155 ns  Iteration: 1  Instance: /tb_PDC
# 1
# Break in Module tb_PDC at D:/00_second_under/StudentID_Lab7/tb_PDC.sv line 166
```

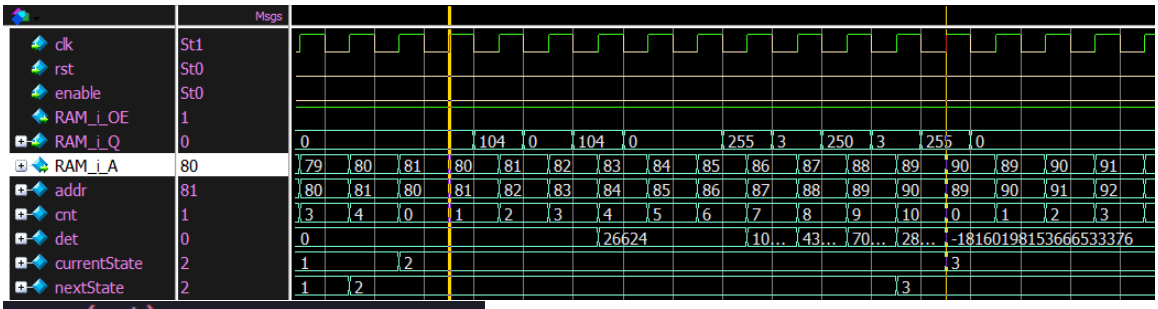
2. Explain the result by waveform.




```
case (cnt)
  4'd0: adj[index][39:32] <= RAM_i_Q;
  4'd1: adj[index][47:40] <= RAM_i_Q;
  4'd2: adj[index][55:48] <= RAM_i_Q;
  4'd3: adj[index][63:56] <= RAM_i_Q;
  4'd4: adj[index][71:64] <= RAM_i_Q;
endcase
```

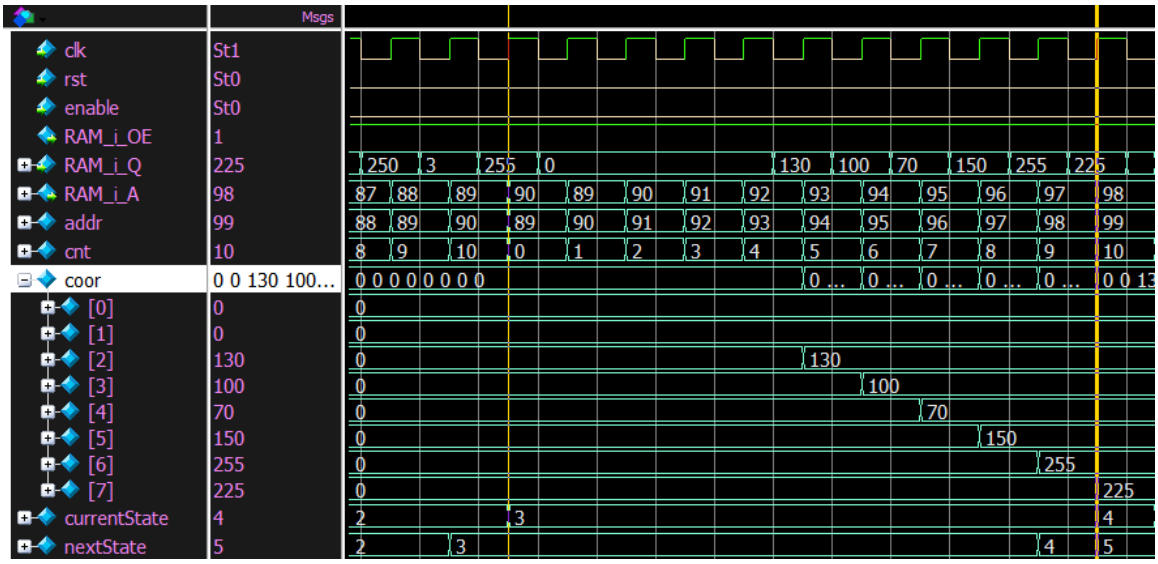
在 READ_ADJ state 主要就是把 Adjoint matrix 讀進來，並存進 array 裡，但因為

每個 element 都是 40 個 bits，而讀進來的每筆 data 都是 8bits，所以額外利用 cnt 來控制。會把讀進來的資料從 32 開始放到 71 是因為後面的(0~31)都是小數部分。

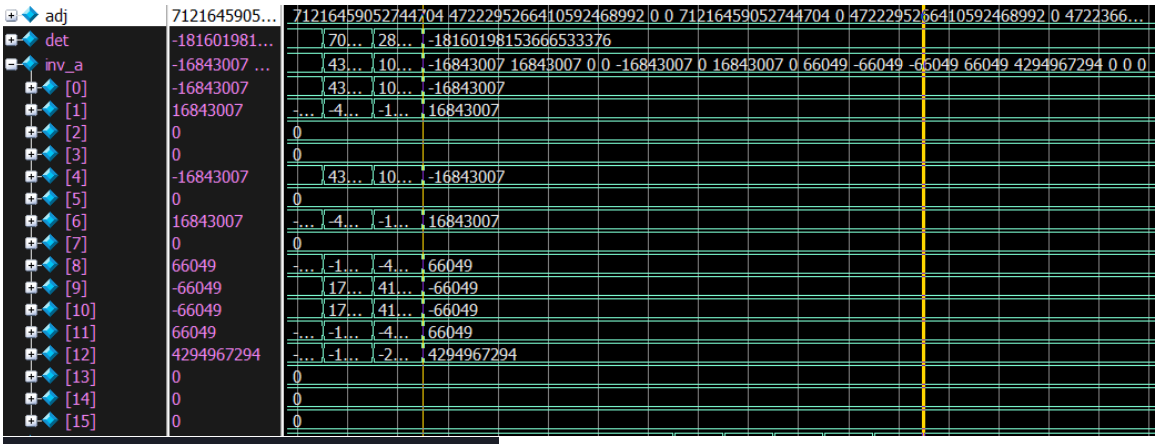



```
case (cnt)
  4'd2: det[7:0] <= RAM_i_Q;
  4'd3: det[15:8] <= RAM_i_Q;
  4'd4: det[23:16] <= RAM_i_Q;
  4'd5: det[31:24] <= RAM_i_Q;
  4'd6: det[39:32] <= RAM_i_Q;
  4'd7: det[47:40] <= RAM_i_Q;
  4'd8: det[55:48] <= RAM_i_Q;
  4'd9: det[63:56] <= RAM_i_Q;
  4'd10: det[71:64] <= RAM_i_Q;
endcase
```

在 READ_DET state 把 determinant A 讀進來，因為 determinant A 為 72bits，如上所述，一樣利用 cnt 來控制。

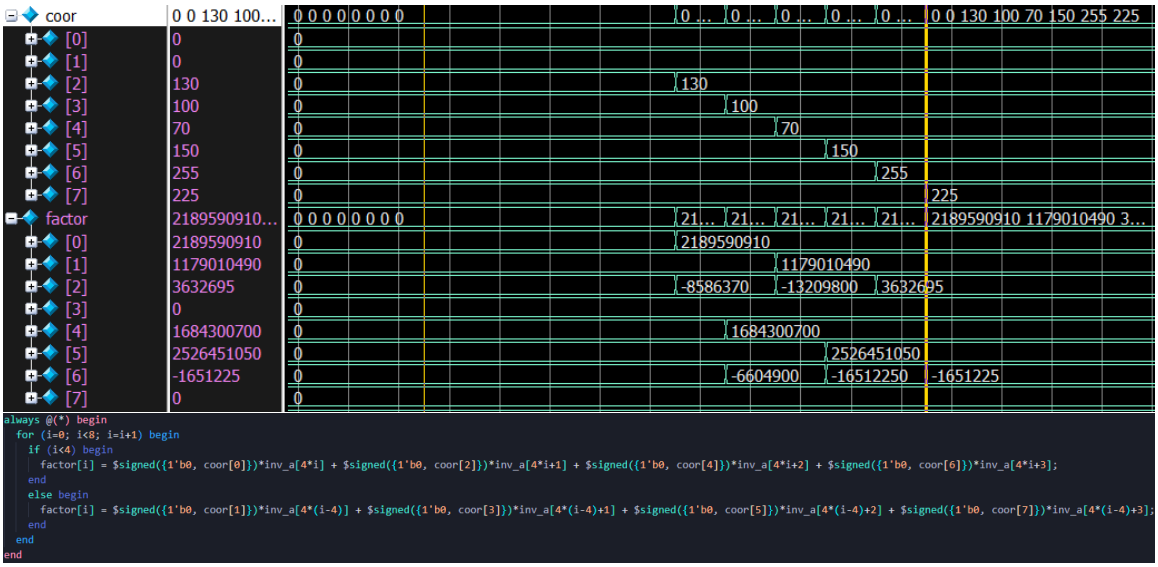


在 READ_COOR state 把 4 個座標點讀進來放在 coor 的 array 裡。



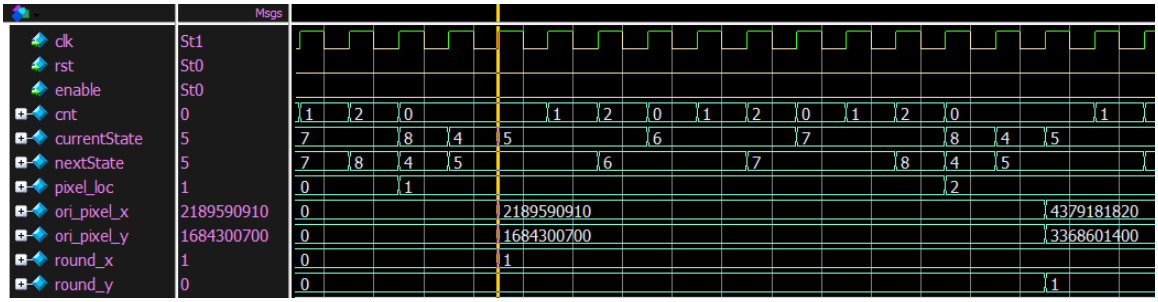

```
always @(*) begin
  for (i=0; i<16; i=i+1) begin
    inv_a[i] = (adj[i]<<32)/det;
  end
end
```

其實在讀完 determinant A 後就可以算出 inverse A，這裡使用組合電路去實現。




```
always @(*) begin
  for (i=0; i<8; i=i+1) begin
    if (i<4) begin
      factor[i] = $signed({1'b0, coor[0]})*inv_a[4*i] + $signed({1'b0, coor[2]})*inv_a[4*i+1] + $signed({1'b0, coor[4]})*inv_a[4*i+2] + $signed({1'b0, coor[6]})*inv_a[4*i+3];
    end
    else begin
      factor[i] = $signed({1'b0, coor[1]})*inv_a[4*(i-4)] + $signed({1'b0, coor[3]})*inv_a[4*(i-4)+1] + $signed({1'b0, coor[5]})*inv_a[4*(i-4)+2] + $signed({1'b0, coor[7]})*inv_a[4*(i-4)+3];
    end
  end
end
```

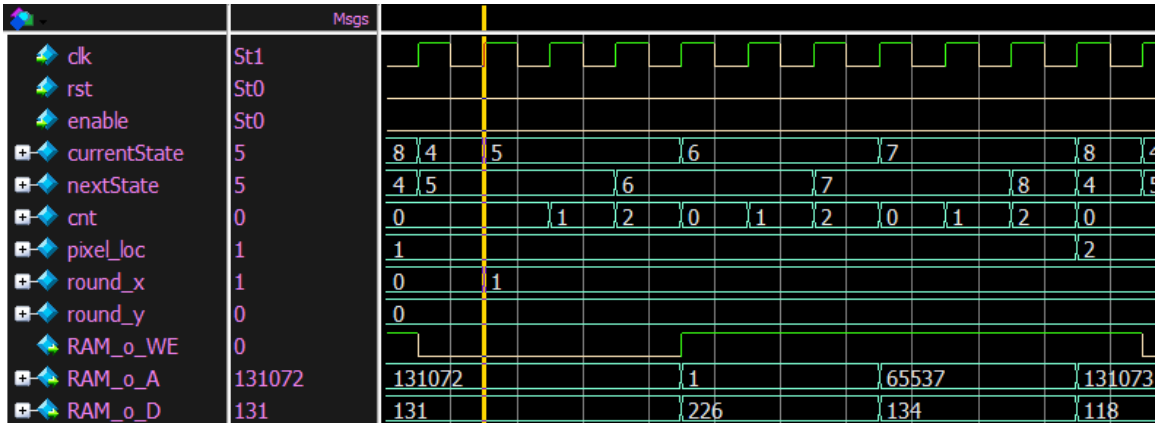
同理，這裡一樣食用組合電路去算出 a, b, c, d, e, f, g, h，並存進 factor 這個 array 裡。



```
FIND_P: begin
  cnt <= 0;
  RAM_i_OE <= 0;
  ori_pixel_x <= factor[0]*$signed({1'b0,pixel_loc[7:0]}) + factor[1]*$signed({1'b0,pixel_loc[15:8]}) + factor[2]*$signed({1'b0,pixel_loc[7:0]})*$signed({1'b0,pixel_loc[15:8]}) + factor[3];
  ori_pixel_y <= factor[4]*$signed({1'b0,pixel_loc[7:0]}) + factor[5]*$signed({1'b0,pixel_loc[15:8]}) + factor[6]*$signed({1'b0,pixel_loc[7:0]})*$signed({1'b0,pixel_loc[15:8]}) + factor[7];
end
```

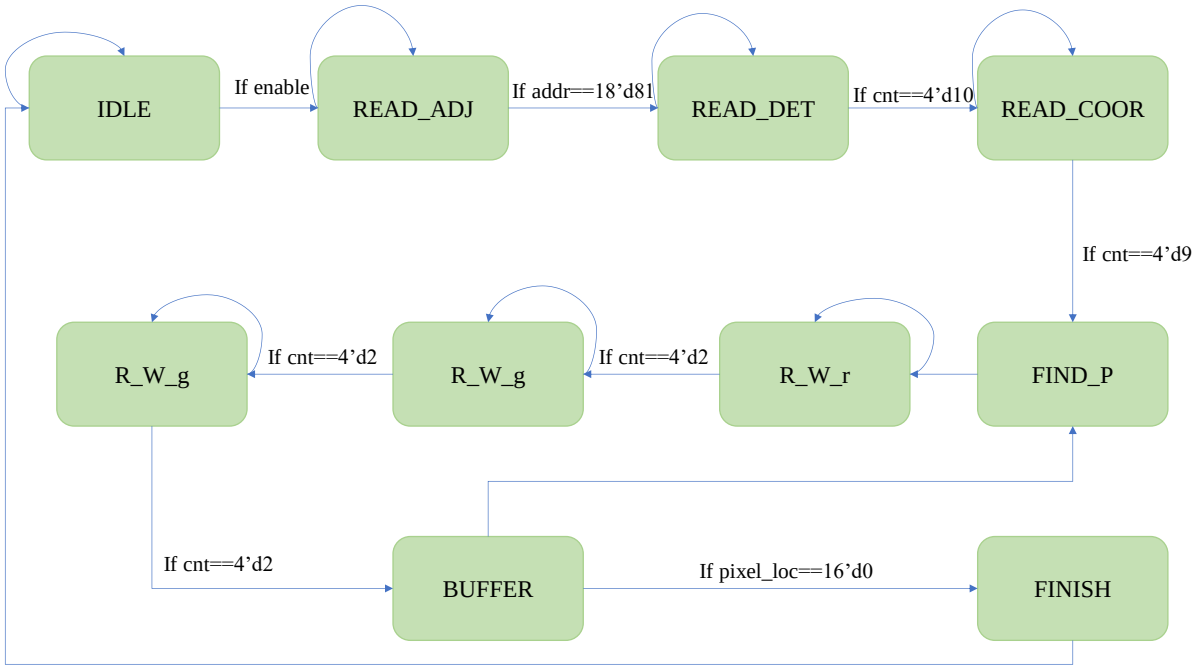
```
always @(*) begin
  round_x = ori_pixel_x[39:32] + ori_pixel_x[31];
  round_y = ori_pixel_y[39:32] + ori_pixel_y[31];
end
```

在 FIND_P，就是利用剛剛找出的 factor 以及 pixel_loc 來找出新的點在原圖的哪的位置，之後會利用這個位置來讀出 R, G, B 值，並寫入 Ram_o。算出來的 ori_pixel_x 與 ori_pixel_y 需要四捨五入，這邊一樣使用組合電路去實現。



在經過 rounding 後，就可以得知要去原圖的哪個位置取出 pixel 值。我的作法是先取出 R channel 的值並寫入 RAM_o，接著換讀取 G channel 的值並寫入 RAM_o，最後就是 B channel。寫完後會進 BUFFER state，判別是否已經處理完整張 image，如果還沒，就會回到 FIND_P，重複以上動作直到結束。

3. Draw the flowchart for your Finite State Machine (FSM).



4. At last, please write the lesson you learned from Lab7.
這次寫的狀態機又比之前的作業還多，所以需要更加注意換狀態的條件。另外此次作業遇到比較大的難點是如何設計狀態，及讀取資料的部分。