Final Project 實驗報告

題目:Image Histogram Equalization

姓名:羅名志

學號:0813228

繳交日期:2022/6/17

(1)實驗程式碼:

主程式碼:

```
module FPGA_FINAL_PAK (

	//////////// CLOCK //////////
	input 		          		CLOCK_50,
//	input 		          		CLOCK2_50,
//	input 		          		CLOCK3_50,
//	input 		          		CLOCK4_50,

	//////////// SEG7 //////////
	output		     [6:0]		HEX0,
	output		     [6:0]		HEX1,
	output		     [6:0]		HEX2,
	output		     [6:0]		HEX3,
	output		     [6:0]		HEX4,
	output		     [6:0]		HEX5,

	//////////// KEY //////////
	input 		     [3:0]		KEY,

	//////////// LED //////////
	output		     [9:0]		LEDR,

	//////////// SW //////////
	input 		     [9:0]		SW,

	//////////// HPS //////////
//	inout 		          		HPS_CONV_USB_N,
	output		    [14:0]		HPS_DDR3_ADDR,
	output		     [2:0]		HPS_DDR3_BA,
	output		          		HPS_DDR3_CAS_N,
	output		          		HPS_DDR3_CK_N,
	output		          		HPS_DDR3_CK_P,
	output		          		HPS_DDR3_CKE,
	output		          		HPS_DDR3_CS_N,
	output		     [3:0]		HPS_DDR3_DM,
        inout           [31:0]          HPS_DDR3_DQ,
        inout            [3:0]          HPS_DDR3_DQS_N,
        inout            [3:0]          HPS_DDR3_DQS_P,
        output                          HPS_DDR3_ODT,
        output                          HPS_DDR3_RAS_N,
        output                          HPS_DDR3_RESET_N,
        input                           HPS_DDR3_RZQ,
        output                          HPS_DDR3_WE_N,
        output                          HPS_ENET_GTX_CLK,
//      inout                           HPS_ENET_INT_N,
        output                          HPS_ENET_MDC,
        inout                           HPS_ENET_MDIO,
        input                           HPS_ENET_RX_CLK,
        input            [3:0]          HPS_ENET_RX_DATA,
        input                           HPS_ENET_RX_DV,
        output           [3:0]          HPS_ENET_TX_DATA,
        output                          HPS_ENET_TX_EN,
//      inout            [3:0]          HPS_FLASH_DATA,
//      output                          HPS_FLASH_DCLK,
//      output                          HPS_FLASH_NCSO,
//      inout            [1:0]          HPS_GPIO,
//      inout                           HPS_GSENSOR_INT,
//      inout                           HPS_I2C_CONTROL,
//      inout                           HPS_I2C1_SCLK,
//      inout                           HPS_I2C1_SDAT,
//      inout                           HPS_I2C2_SCLK,
//      inout                           HPS_I2C2_SDAT,
//      inout                           HPS_KEY,
//      inout                           HPS_LED,
        output                          HPS_SD_CLK,
        inout                           HPS_SD_CMD,
        inout            [3:0]          HPS_SD_DATA,
//      output                          HPS_SPIM_CLK,
//      input                           HPS_SPIM_MISO,
//      output                          HPS_SPIM_MOSI,
//      inout                           HPS_SPIM_SS,
        input                           HPS_UART_RX,
        output                          HPS_UART_TX
```

```
//     input                                   HPS_USB_CLKOUT,
//     inout                  [7:0]            HPS_USB_DATA,
//     input                                   HPS_USB_DIR,
//     input                                   HPS_USB_NXT,
//     output                                  HPS_USB_STP
);

//=======================================================
//    REG/WIRE declarations
//=======================================================


wire    [1:0]           state;
wire                              R_W_done;

//indicator to see if we are done prepare or done process
reg                     pre_done        ;
reg                     pro_done        ;

//ports to TA's preset RAM
reg   [16:0]      ram_TA_add;
reg                     ram_TA_wen;
reg   [7:0]       ram_TA_data;
reg                     ram_TA_ren;
wire        [7:0]       ram_TA_q;

//=======================================================
//    UI assign
//=======================================================


assign LEDR[1:0]      = state;

assign LEDR[2]        = R_W_done;

//turn off unused indicator
//assign LEDR[9:3]   = i[6:0];
//assign LEDR[9] = clk_n[16];
```

```
//assign LEDR[] =
//assign          HEX0       = 7'h7f;
//assign          HEX1       = 7'h7f;
//assign          HEX2       = 7'h7f;
//assign          HEX3       = 7'h7f;
//assign          HEX4       = 7'h7f;
//assign          HEX5       = 7'h7f;

//=======================================================
//    Structural coding
//=======================================================
reg [3:0]trans_done,sigg,all_done,sec_done;
reg [7:0]min;
reg [7:0]max;

reg [16:0] add,clk_n;
reg[3:0] clk_2;
reg[16:0] nk[256];
reg[16:0] ck[256];
reg[16:0] dk[256];
reg[8:0] sk[256];

integer idx, idx_2, ini;
//wire [16:0] r, s, t;
//assign r=dk[242];
//assign s=dk[8];
//assign t=ck[242];
reg count;
wire[24:0]    quo;
wire[16:0]    rem;
reg[24:0] b;
//reg [8:0] t,s,r;
//wire[8:0]i;
//assign i=nk[12];
//parameter size=17'b00000000011111111;
//hexto7segment s1(t[3:0],HEX0);
//hexto7segment s2(t[7:4],HEX1);
//hexto7segment s3(s[3:0],HEX2);
```

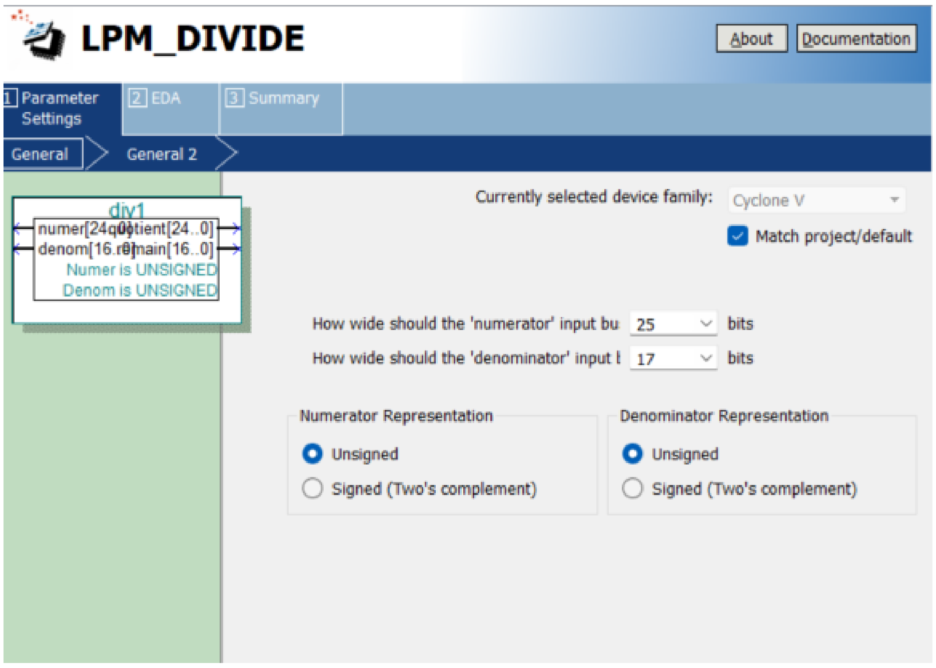
```
//hexto7segment s4(s[7:4],HEX3);
//hexto7segment s5(r[3:0],HEX4);
//hexto7segment s6(r[7:4],HEX5);
div1(dk[max],b,quo,rem);

always@ (posedge CLOCK_50) begin
        if(clk_2==11) begin
                clk_2=0;

        //initialization
        if(state==2'b00) begin
                min=8'b11111111;
                max=0;
                trans_done=0;
                clk_n=0;
                pre_done=1;
                all_done=0;
                add=0;
                sec_done=0;
                //sigg=0;
                sec_done=0;

                for(ini=0; ini<256; ini=ini+1) begin
                        nk[ini]=0;
                        ck[ini]=0;
                        dk[ini]=0;
                        sk[ini]=0;
                end
        end

        else if(state==2'b10) begin

                //part1:calculate nk, max, min value
                if(trans_done==0 & add<=17'b10100000000000000) begin
                        ram_TA_ren =    1'b1;
                        ram_TA_add = add[16:0];
                        add=add+1;
                        clk_n=clk_n+1;
                        if(clk_n>=17'b00000000000000010) begin

                                if(ram_TA_q < min)begin
                                        min = ram_TA_q;
                                end
                                if(ram_TA_q > max) begin
                                        max = ram_TA_q;
                                end
                                nk[ram_TA_q]=nk[ram_TA_q]+1;

                        end
                end
                //part 2 :calculate ck
                else if (trans_done==0 & add > 17'b10100000000000000)begin
                        ram_TA_add=0;
                        add=0;
                        trans_done=1;
                        clk_n=0;
                        for(idx=0; idx<=255; idx=idx+1)begin
                                if(idx>0) begin
                                        ck[idx]=ck[idx-1]+nk[idx];
                                end
                                else begin
                                        ck[idx]=nk[idx];
                                end
                        end
                end

                //calculate dk for transition
                else if(sec_done==0 & trans_done==1) begin
                        sec_done=1;
                        for(idx_2=0; idx_2<=255; idx_2=idx_2+1)begin
                                if(idx_2>min && idx_2>0 ) begin
                                        //sigg=1;
                                        dk[idx_2]=dk[idx_2-1]+nk[idx_2];
                                end
                        end
                        end

                        //part3:calculate sk as LUT table
                        else if(sec_done==1 & all_done==0) begin

                            if(count ==0) begin
                                b<=255*dk[add];
                                count=1;
                            end

                            else begin
                                count=0;
                                sk[add]=quo[7:0];
                                add=add+1;
                                if(add==255) begin
                                    sigg=1;
                                    add=0;
                                    all_done=1;
                                end
                            end
                        end

                        //part4:write result
                        else if(sec_done==1 & all_done==1 & add<=17'b10100000000000011)
                begin
                            if(count ==0) begin
                                ram_TA_ren =    1;
                                ram_TA_wen=0;

                                ram_TA_add=add;
                                add=add+1;
                                //ram_TA_q;
                                count=1;
                            end
                            else begin
                                count=0;
                                ram_TA_ren =    0;
                                if(add>=17'b00000000000000000 &
                clk_n<17'b10100000000000000) begin
                                    ram_TA_add = clk_n;
                                    ram_TA_wen=1;
                                    ram_TA_data = sk[ram_TA_q];
                                    clk_n=clk_n+1;
                                end

                            end
                        end

                        else begin
                            pro_done <= 1'b1;
                            ram_TA_wen <= 0;
                            //sigg=1;
                        end
                    end


                    else begin
                        //sigg=0;
                        ram_TA_add<=0;
                        add=0;
                        trans_done <= 0;
                        ram_TA_wen <= 0;
                        pre_done <= 1'b0;
                        pro_done <= 1'b0;
                        ram_TA_ren    <=    1'b0;
                    end
                    end

                    else begin
                        clk_2=clk_2+1;
                    end
                end

                /*============================================================
                =======
                // Below is TA's module, don't modify it unless you know what are you doing
```

```
=================================================================
=====*/

V_TA                                    u0 (
        /////////// CLOCK //////////
        .CLOCK_50                  (CLOCK_50),

        //================================================================
=======
        //                                                      Student's IO
        //================================================================
=======
        //////// Change state ///////
        .r_2_pro                   (KEY[1]),
        .w_2_pre                   (KEY[2]),
        //////////// ACT ////////////
        .act                       (KEY[0]),
        /////////// State ///////////
        .state_w                   (state),
        /////////    R W done /////////
        .R_W_done                  (R_W_done),
        //////// Choose frame ///////
        .frame_number              (SW[0]),
        ////// Is state done? ///////
        .pre_done                  (pre_done),
        .pro_done                  (pro_done),
        //// Student Control RAM ////
        .stc_ram_add          (ram_TA_add),
        .stc_ram_wen          (ram_TA_wen),
        .stc_ram_data         (ram_TA_data),
        .stc_ram_ren          (ram_TA_ren),
        .stc_ram_q                 (ram_TA_q),
        //================================================================
=======
        //                                               End of Student's IO
        //================================================================
=======
	///////////// HPS ///////////
	.HPS_DDR3_ADDR          (HPS_DDR3_ADDR),
	.HPS_DDR3_BA            (HPS_DDR3_BA),
	.HPS_DDR3_CAS_N (HPS_DDR3_CAS_N),
	.HPS_DDR3_CK_N          (HPS_DDR3_CK_N),
	.HPS_DDR3_CK_P          (HPS_DDR3_CK_P),
	.HPS_DDR3_CKE           (HPS_DDR3_CKE),
	.HPS_DDR3_CS_N          (HPS_DDR3_CS_N),
	.HPS_DDR3_DM            (HPS_DDR3_DM),
	.HPS_DDR3_DQ            (HPS_DDR3_DQ),
	.HPS_DDR3_DQS_N         (HPS_DDR3_DQS_N),
	.HPS_DDR3_DQS_P (HPS_DDR3_DQS_P),
	.HPS_DDR3_ODT           (HPS_DDR3_ODT),
	.HPS_DDR3_RAS_N (HPS_DDR3_RAS_N),
	.HPS_DDR3_RESET_N       (HPS_DDR3_RESET_N),
	.HPS_DDR3_RZQ           (HPS_DDR3_RZQ),
	.HPS_DDR3_WE_N          (HPS_DDR3_WE_N),
	.HPS_ENET_GTX_CLK       (HPS_ENET_GTX_CLK),
	.HPS_ENET_MDC           (HPS_ENET_MDC),
	.HPS_ENET_MDIO          (HPS_ENET_MDIO),
	.HPS_ENET_RX_CLK(HPS_ENET_RX_CLK),
	.HPS_ENET_RX_DATA       (HPS_ENET_RX_DATA),
	.HPS_ENET_RX_DV (HPS_ENET_RX_DV),
	.HPS_ENET_TX_DATA       (HPS_ENET_TX_DATA),
	.HPS_ENET_TX_EN  (HPS_ENET_TX_EN),
	.HPS_SD_CLK             (HPS_SD_CLK),
	.HPS_SD_CMD             (HPS_SD_CMD),
	.HPS_SD_DATA            (HPS_SD_DATA),
	.HPS_UART_RX            (HPS_UART_RX),
	.HPS_UART_TX            (HPS_UART_TX)
);
//===========================================================
=
// Above is TA's module, don't modify unless you know what are you doing
//===========================================================
=

endmodule
```

```
module hexto7segment (
      input [3:0] iDIG,
      output reg [6:0] oSEG
          );

  always@(iDIG) begin
      case(iDIG)
          4'h1: oSEG = 7'b1111001;
        4'h2: oSEG = 7'b0100100;    // ---t----          4'h2: oSEG = 7'b0100100;    // |
|
          4'h3: oSEG = 7'b0110000;    // lt        rt
          4'h4: oSEG = 7'b0011001;    // |            |
          4'h5: oSEG = 7'b0010010;    // ---m----
          4'h6: oSEG = 7'b0000010;    // |            |
          4'h7: oSEG = 7'b1111000;    // lb        rb
          4'h8: oSEG = 7'b0000000;    // |            |
          4'h9: oSEG = 7'b0011000;    // ---b----
          4'ha: oSEG = 7'b0001000;
          4'hb: oSEG = 7'b0000011;
          4'hc: oSEG = 7'b1000110;
          4'hd: oSEG = 7'b0100001;
          4'he: oSEG = 7'b0000110;
          4'hf: oSEG = 7'b0001110;
          4'h0: oSEG = 7'b1000000;
      endcase
  end

endmodule
```

LPM_divider

```
`timescale 1 ps / 1 ps
// synopsys translate_on
module div1 (
        denom,
        numer,
	quotient,
	remain);

	input	[16:0]	denom;
	input	[24:0]	numer;
	output	[24:0]	quotient;
	output	[16:0]	remain;

	wire [24:0] sub_wire0;
	wire [16:0] sub_wire1;
	wire [24:0] quotient = sub_wire0[24:0];
	wire [16:0] remain = sub_wire1[16:0];

	lpm_divide	LPM_DIVIDE_component (
				.denom (denom),
				.numer (numer),
				.quotient (sub_wire0),
				.remain (sub_wire1),
				.aclr (1'b0),
				.clken (1'b1),
				.clock (1'b0));
	defparam
		LPM_DIVIDE_component.lpm_drepresentation = "UNSIGNED",
		LPM_DIVIDE_component.lpm_hint = "LPM_REMAINDERPOSITIVE=TRUE",
		LPM_DIVIDE_component.lpm_nrepresentation = "UNSIGNED",
		LPM_DIVIDE_component.lpm_type = "LPM_DIVIDE",
		LPM_DIVIDE_component.lpm_widthd = 17,
		LPM_DIVIDE_component.lpm_widthn = 25;


endmodule
```



Currently selected device family: Cyclone V

[✓] Match project/default

How wide should the 'numerator' input bu 25 bits

How wide should the 'denominator' input b 17 bits

Numerator Representation

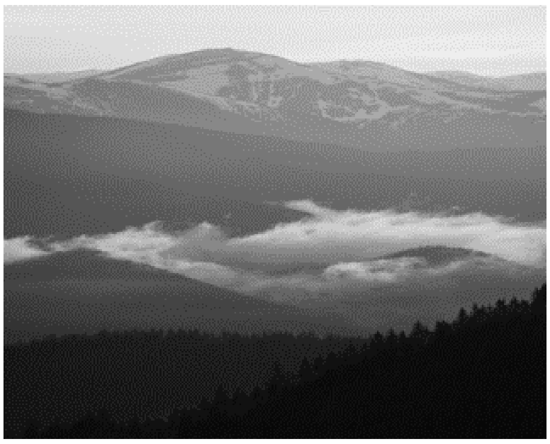
[✓ Unsigned] Signed (Two's complement)

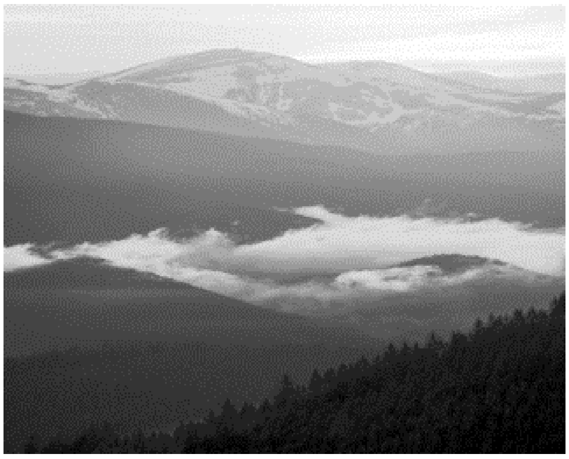
Denominator Representation

[✓ Unsigned] Signed (Two's complement)

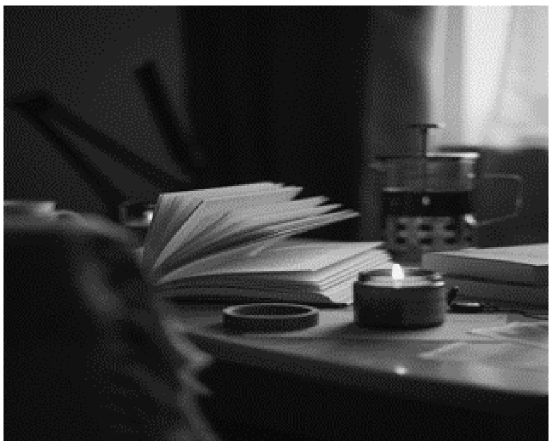
(2)實驗結果:

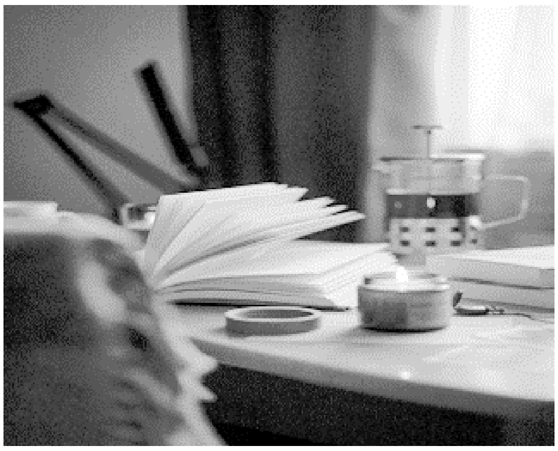
//input0 -> input1



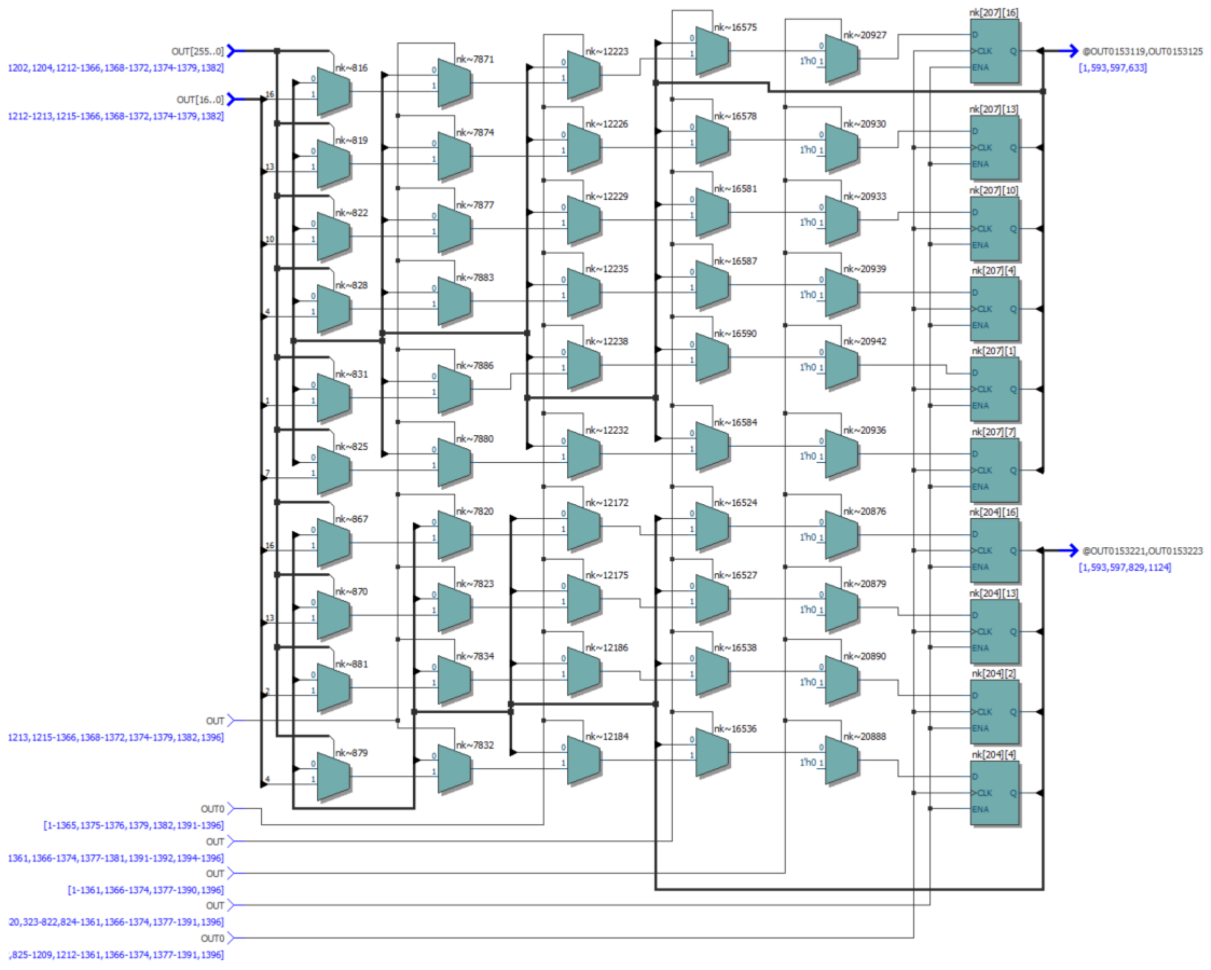


//output0 -> output1





(3)RTL 佈局:



(4)問題與討論:

首先先簡介一下我的作法，我將 state=00 作為初始化階段，將所有的 register 給定初始值以免在執行第一張照片後會有出錯的情形，而在 state=10 時，大致上分為 5 步驟:第一步為循序讀入 ram 內每個 address 的 gray level，並用 if-else 判斷是否為最大或最小的灰階值，同時計算 n[k];第二步則是透過 n[k] 的結果並利用 for 迴圈計算 c[k];第三步則是為了降低整個程式的 longest path，因此先計算每個 c[k]-c[min]的值(定義為 d[k]，也就是 c[k]的 k 從 min+1 的地方開始累加)，第四步則是透過 d[k]的結果計算 s[k](即 255*d[k]/d[max])，最後透過 s[k]作為 LUT table 將轉換後的灰階值寫回 ram 中，不過為了避免同時讀出和寫入問題，因此我將最後一步又細分兩部分處理。

在完成這份期末專題的時候我遇到許多問題，首先從理解 sd 卡、FPGA、RAM 間的 I/O 就花了許多時間，另外熟悉如何使用 putty 也花了一段時間，不過最難且最花時間理解的是助教所提到的時脈問題，一開始沒意識到時脈的大

小會影響實驗結果，只知道程式邏輯都看似無誤同時編譯也都有過，但利用七段顯示器顯示計算的值都是錯的，為此花了許多時間修正和理解，另外，register 和 wire 的宣告和使用也是這次實驗中看似簡單但充滿細節的地方。

由於一開始圖片的格式錯誤和自己理解速度慢，導致花很久的時間做許多無意義的改動，因此最後時間不夠無法繼續優化結果，不過我還是有一些能讓這次專題更進步的想法，像是其實不用計算 c[k]的值即有辦法得出 s[k]，就如同我上述所提到的，因為轉換公式的分子和分母都減掉 c[min]，所以可以在累加時直接從最小值的下一 address 累加，但礙於題目要求所以仍要計算 c[k]，結果讓我因此多花了將近 6 個 clock(從 6/50M 到 12/50M)。另外，register 的部分如果能更加善用 lpm 架構或大小設計上更精簡應該都能有效提升性能和減少memory 空間浪費。

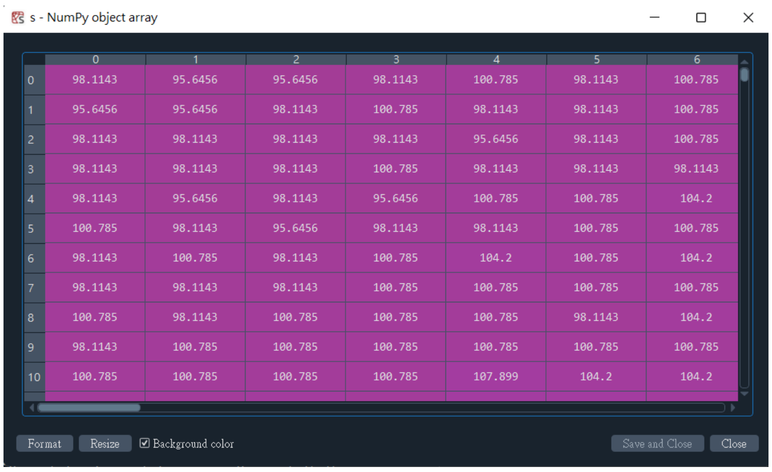
最後，我原本真的很想考期末考，因為比較好準備，但藉由這次的期末project 學到非常多，除了讓我學到許多編寫 verilog 上的技巧和觀念，也讓我更加深前十次 lab 助教所教導的各個觀念，算是收穫滿滿，對於一個外系且沒有修過邏輯設計的我來說，真的因為這堂課和這次的作業對數位設計和軟韌體有更進一步的了解和認知。

(5)補充:檢查正確性的程式(python)

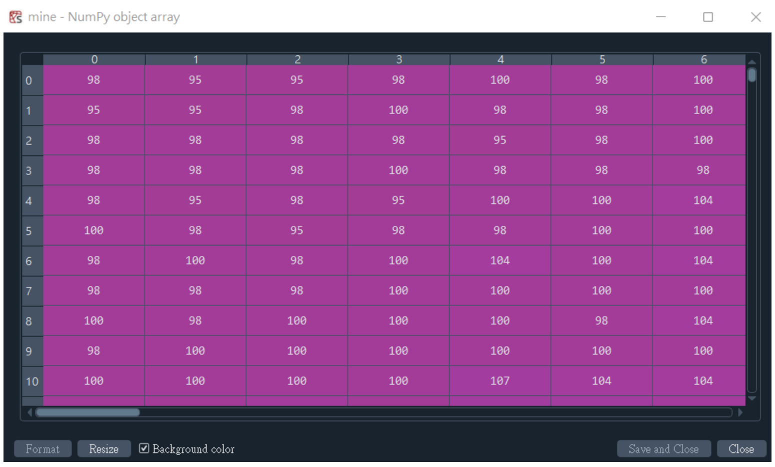
```
import numpy as np
import cv2
from matplotlib import pyplot as plt
img =
cv2.imread("C:/Users/max2306/Desktop/FPGA_project_student/4.MATLAB/input1.b
mp", cv2.IMREAD_GRAYSCALE)
mine=plt.imread("C:/Users/max2306/Desktop/output1.bmp",
cv2.IMREAD_GRAYSCALE)
cmin = 255
cmax = 0
n=np.zeros((256,1))

for i in range(256):
    for j in range(320):
        # print(img[i][j][0])
        if(img[i][j]<cmin):
            cmin=img[i][j]
```

```
                if(img[i][j]==242):
                        print(str(i)+', '+str(j))

                if(img[i][j]>cmax):
                        cmax=img[i][j]
                n[img[i][j]]+=1

c = np.zeros((256,1))
c[0] = 0
for i in range(1,256):
        c[i]=c[i-1]+n[i]

s = np.zeros((256,320))

for i in range(256):
        for j in range(320):
                s[i][j] = ((c[img[i][j]]-c[cmin])/(c[cmax]-c[cmin]))*255

plt.imshow(img,cmap='gray')
plt.imshow(s,cmap='gray')
plt.imshow(mine,cmap='gray')
```

正確性:


s - NumPy object array

	0	1	2	3	4	5	6
0	98.1143	95.6456	95.6456	98.1143	100.785	98.1143	100.785
1	95.6456	95.6456	98.1143	100.785	98.1143	98.1143	100.785
2	98.1143	98.1143	98.1143	98.1143	95.6456	98.1143	100.785
3	98.1143	98.1143	98.1143	100.785	98.1143	98.1143	98.1143
4	98.1143	95.6456	98.1143	95.6456	100.785	100.785	104.2
5	100.785	98.1143	95.6456	98.1143	98.1143	100.785	100.785
6	98.1143	100.785	98.1143	100.785	104.2	100.785	104.2
7	98.1143	98.1143	98.1143	100.785	100.785	100.785	100.785
8	100.785	98.1143	100.785	100.785	100.785	98.1143	104.2
9	98.1143	100.785	100.785	100.785	100.785	100.785	100.785
10	100.785	100.785	100.785	100.785	107.899	104.2	104.2


mine - NumPy object array

	0	1	2	3	4	5	6
0	98	95	95	98	100	98	100
1	95	95	98	100	98	98	100
2	98	98	98	98	95	98	100
3	98	98	98	100	98	98	98
4	98	95	98	95	100	100	104
5	100	98	95	98	98	100	100
6	98	100	98	100	104	100	104
7	98	98	98	100	100	100	100
8	100	98	100	100	100	98	104
9	98	100	100	100	100	100	100
10	100	100	100	100	107	104	104

//左圖為程式計算而得 s(k)，右圖則是經 FPGA 後轉換的圖片灰階值
//n(k)、c(k)同理藉由此方法觀察其正確性