RTL Design of RGB to YUV

Use VCS and Verdi to implement RGB to YUV

實驗日期：2024.05.06

學號姓名：B092040016 陳昱逢

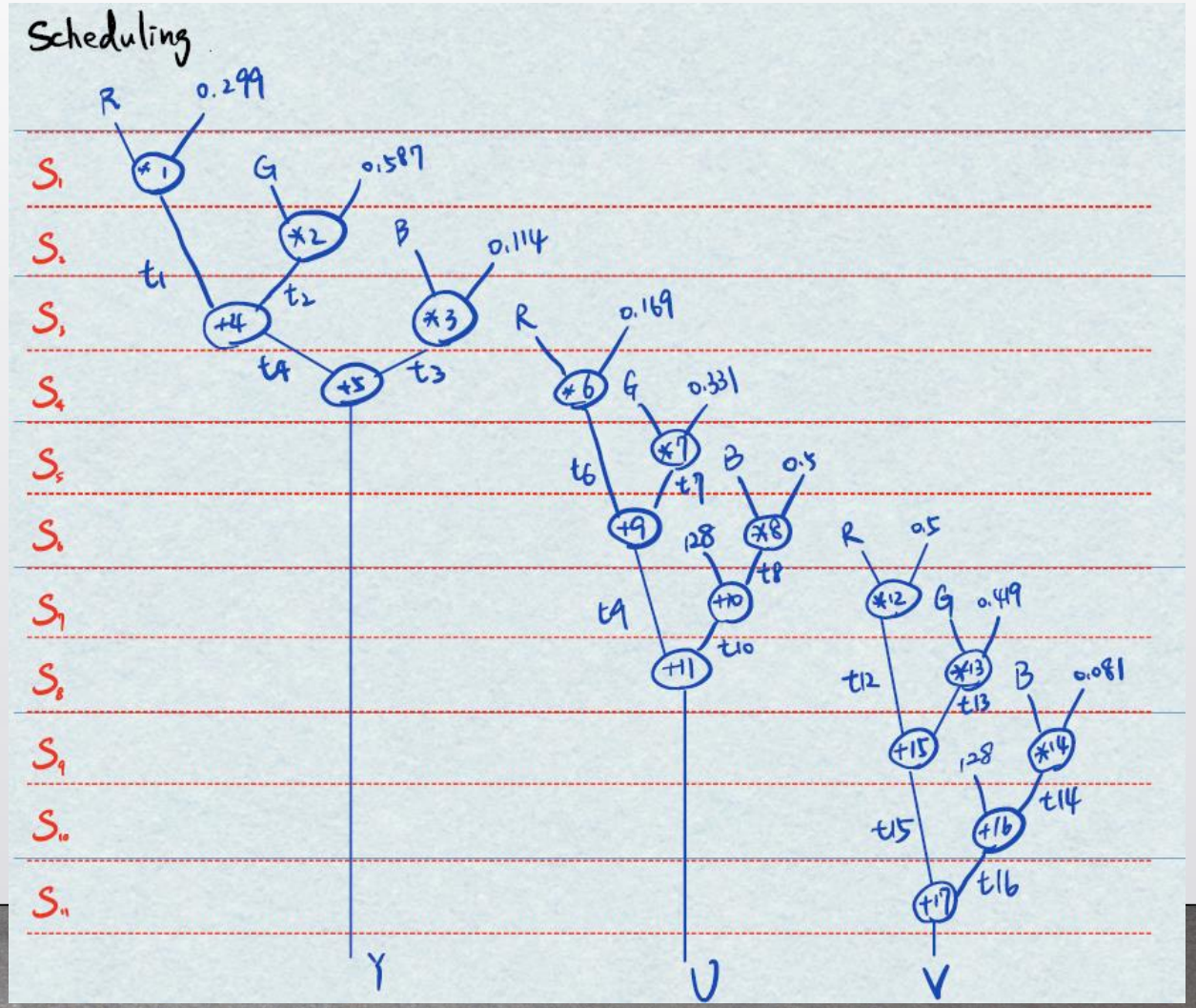
實驗內容及過程

- **主要內容：**使用 Verilog 撰寫 RGB to YUV 電路，並使用提供的 testbench.v 測試實作是否正確。

- **主要過程：**利用前一次的實驗的排程結果，我先分別畫出 Scheduling, Allocation, Datapath 以及 STG and State Table 等圖片，接著參考提供的 Verilog code 一步步實作出完整電路並測試成功。

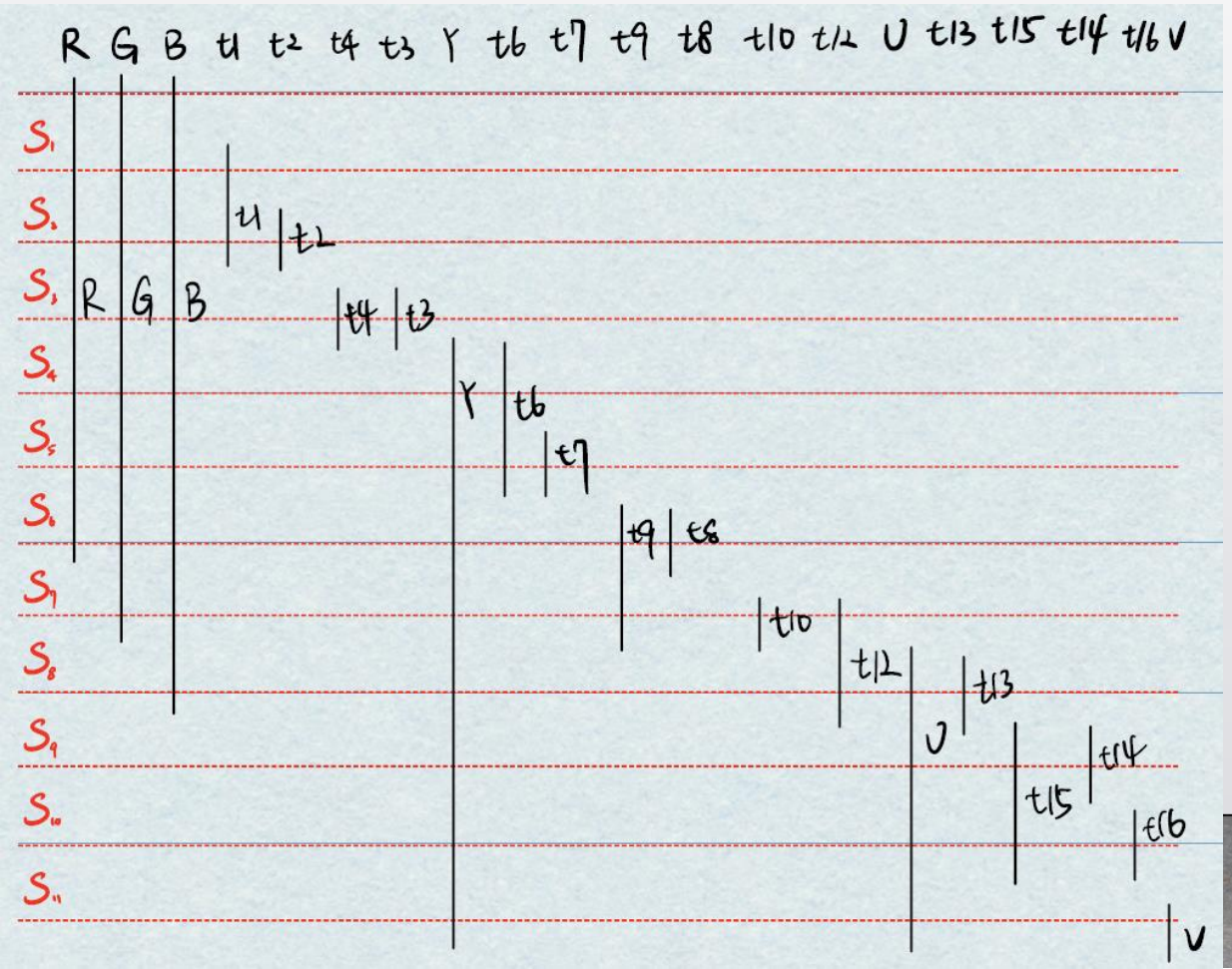
實驗過程-Scheduling

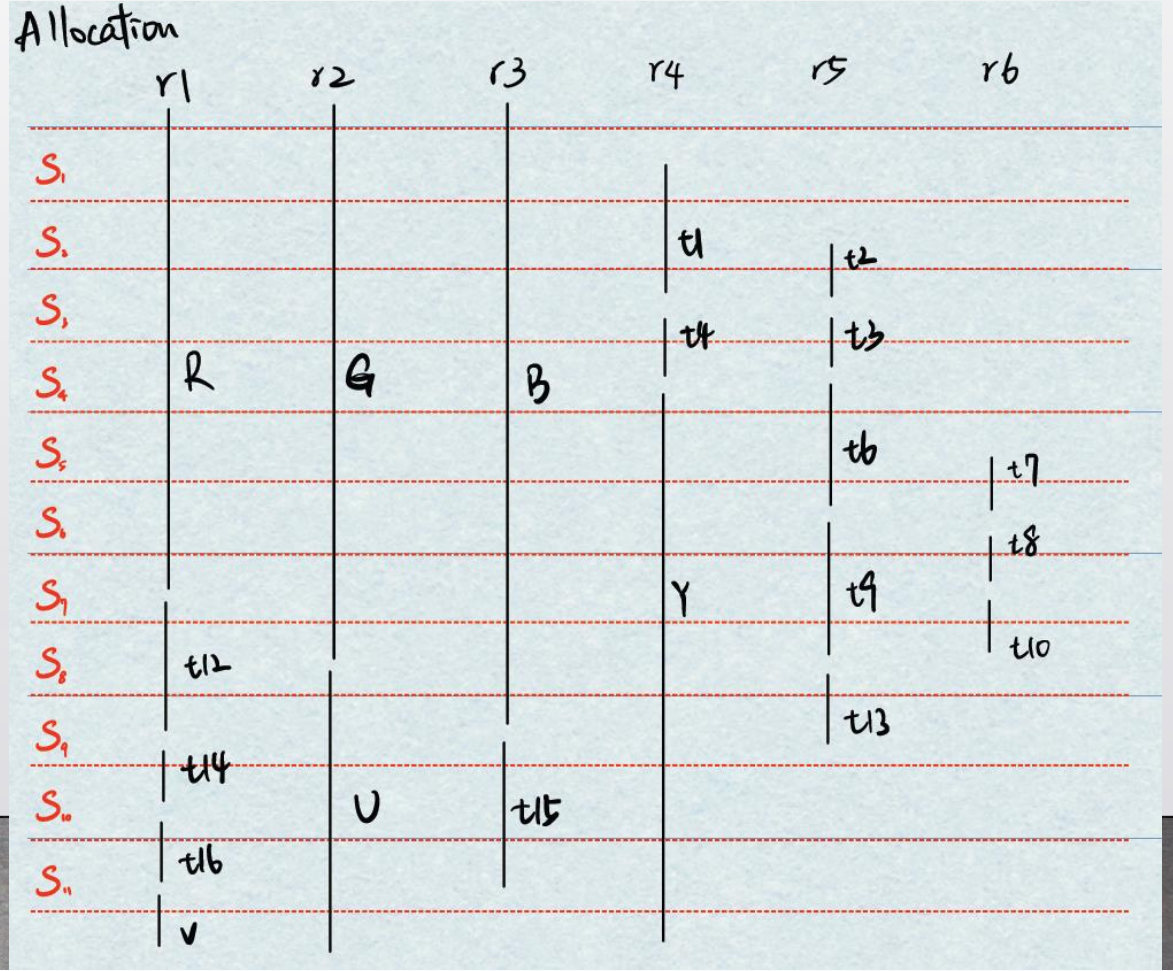
首先根據前一次的實驗排程結果，畫出我的 Scheduling 圖：



實驗過程-Allocation

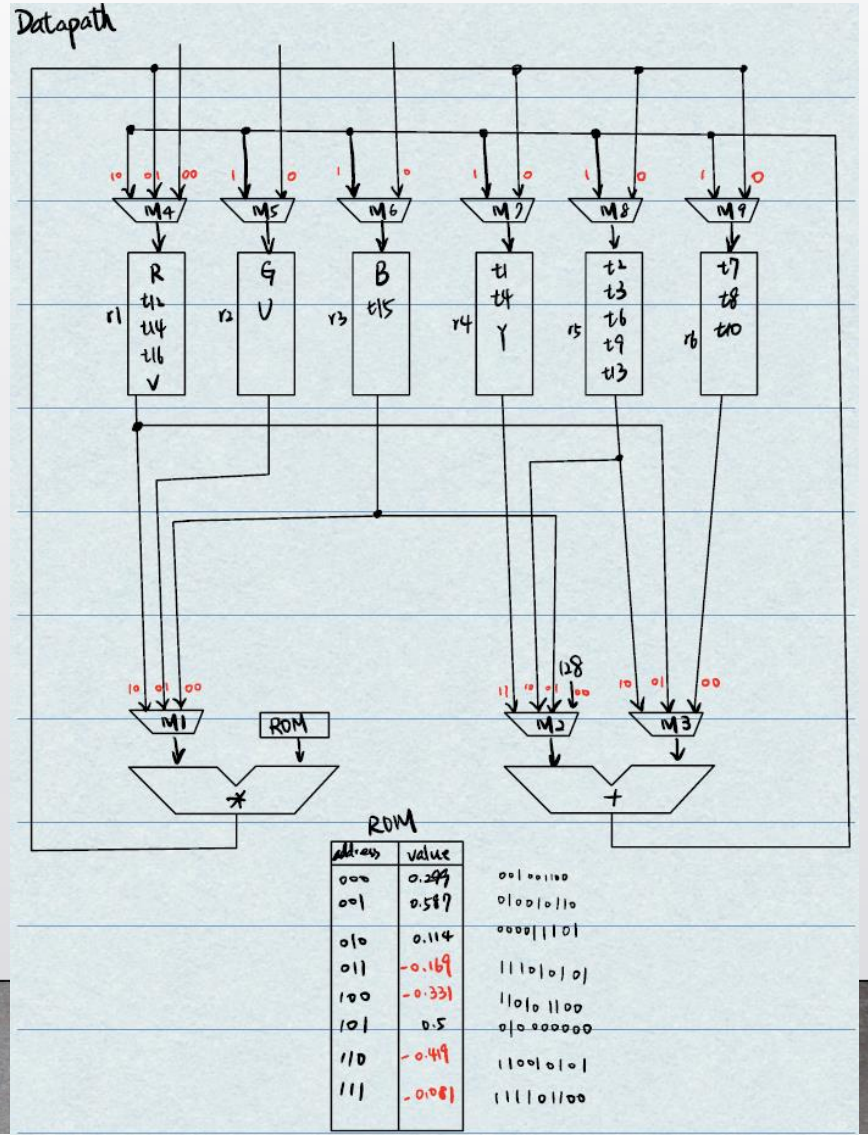
接著做 Allocation ，最後整理得出右圖：





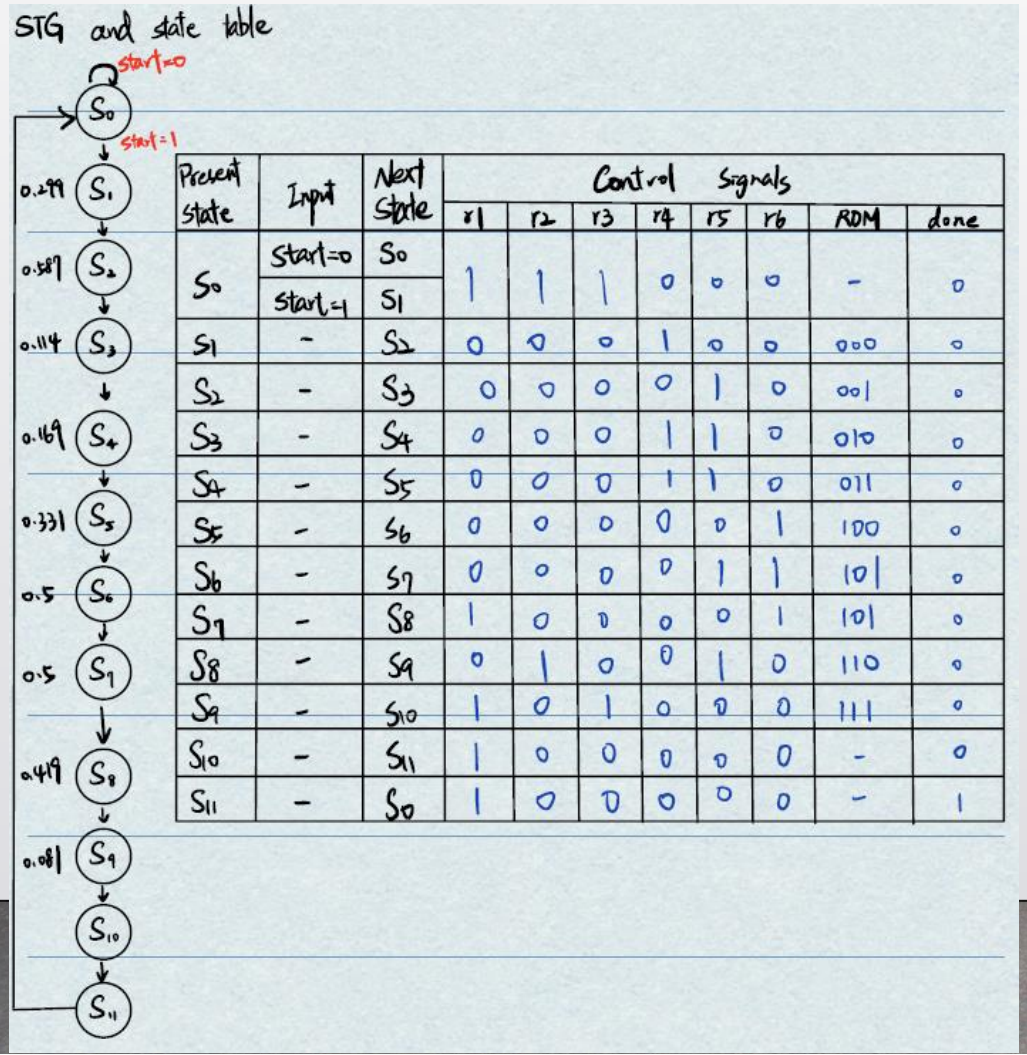
實驗過程-Datapath

根據 Allocation 結果，畫出 Datapath：



實驗過程-STG and state table

接著畫出 STG and state table, 以便寫控制訊號：



Present state	Input	Next State	Control Signals r1	r2	r3	r4	r5	r6	RDM	done
S_0	Start=0	S_0	1	1	1	0	0	0	-	0
	Start=1	S_1								
S_1	-	S_2	0	0	0	1	0	0	000	0
S_2	-	S_3	0	0	0	0	1	0	001	0
S_3	-	S_4	0	0	0	1	1	0	010	0
S_4	-	S_5	0	0	0	1	1	0	011	0
S_5	-	S_6	0	0	0	0	0	1	100	0
S_6	-	S_7	0	0	0	0	1	1	101	0
S_7	-	S_8	1	0	0	0	0	1	101	0
S_8	-	S_9	0	1	0	0	1	0	110	0
S_9	-	S_{10}	1	0	1	0	0	0	111	0
S_{10}	-	S_{11}	1	0	0	0	0	0	-	0
S_{11}	-	S_0	1	0	0	0	0	0	-	1

Present state	Input	Next State	Control Signals M1	M2	M3	M4	M5	M6	M7	M8	M9
S_0	Start=0	S_0	-	-	-	00	0	0	-	-	-
	Start=1	S_1									
S_1	-	S_2	10	-	-	-	-	-	0	-	-
S_2	-	S_3	01	-	-	-	-	-	-	0	-
S_3	-	S_4	00	11	10	-	-	-	1	0	-
S_4	-	S_5	10	11	10	-	-	-	1	0	-
S_5	-	S_6	01	-	-	-	-	-	-	-	0
S_6	-	S_7	00	10	00	-	-	-	-	1	0
S_7	-	S_8	10	00	00	01	-	-	-	-	1
S_8	-	S_9	01	10	00	-	1	-	-	0	-
S_9	-	S_{10}	00	10	01	01	-	1	-	-	-
S_{10}	-	S_{11}	-	00	01	10	-	-	-	-	-
S_{11}	-	S_0	-	01	01	10	-	-	-	-	-

實驗過程-程式碼說明

參考提供的程式碼，先定義基本模組，其中我設計的 ROM 的值如最右邊的圖，轉換成二進制寫入在 ROM 的 module 中，基本模組的 Verilog code 如下:

Adder:

Add.v
```
1  `define bits 9
2  module Add(A, B, Add);
3      input signed [`bits-1:0] A, B;
4      output [`bits-1:0] Add;
5      assign Add = A + B;
6  endmodule
7
8
9
```

Multiplier:

Multiplier.v
```
1  `define bits 9
2  module Mul(A, B, Mul);
3      input signed [`bits-1:0] A, B;
4      output [`bits-1:0] Mul;
5      wire s;
6      wire [7:0] N;
7      assign {s, Mul, N} = A * B;
8  endmodule
```

Register:

Register.v
```
1  `define bits 9
2  module Register (
3      input [`bits-1:0] D,
4      input reset, clk, load,
5      output reg [`bits-1:0] Q
6  );
7      always @(posedge clk or negedge reset) begin
8          if(!reset)
9              Q <= 9'b0;
10         else if(load)
11             Q <= D;
12     end
13 endmodule
```

ROM:

ROM.v
```
1  module ROM (clk, addr, data);
2      input clk;
3      input [2:0] addr;
4      output reg [8:0] data;
5      always @(*)
6      begin
7          case(addr)
8          3'b000: data <= 9'b001001100;
9          3'b001: data <= 9'b010010110;
10         3'b010: data <= 9'b000011101;
11         3'b011: data <= 9'b111010101;
12         3'b100: data <= 9'b110101100;
13         3'b101: data <= 9'b010000000;
14         3'b110: data <= 9'b110010101;
15         3'b111: data <= 9'b111101100;
16         endcase
17     end
18 endmodule
```

ROM 設計方式:

ROM

address	value
000	0.299
001	0.587
010	0.114
011	-0.169
100	-0.331
101	0.5
110	-0.419
111	-0.081

實驗過程-程式碼說明

我的電路設計中有接收 2 個訊號的多工器，因此參考老師提供的 MUX3、MUX4 的程式碼，撰寫出 MUX2 ，Multiplexer 的邏輯蠻類似的，其中 S 為控制訊號，控制要使用哪一個輸入作為輸出。以下為 Multiplexer 接收不同數量輸入的程式碼：

MUX2:

MUX2.v

```
1  `define bits 9
2  module MUX2 (
3      input [`bits-1:0] A, B,
4      input S,
5      output reg [`bits-1:0] Y
6  );
7      always @(*) begin
8          case (S)
9              1'b0: Y = A;
10             1'b1: Y = B;
11             default : Y = A;
12         endcase
13     end
14 endmodule
```

MUX3:

MUX3.v

D:\Desktop\IC_LAB\Lab3\MUX3.v

```
2  module MUX3 (
3      input [`bits-1:0] A, B, C,
4      input [1:0] S,
5      output reg [`bits-1:0] Y
6  );
7      always @(*) begin
8          case (S)
9              2'b00: Y = A;
10             2'b01: Y = B;
11             2'b10: Y = C;
12             default : Y = A;
13         endcase
14     end
15 endmodule
```

MUX4:

MUX4.v

```
1  `define bits 9
2  module MUX4 (
3      input [`bits-1:0] A, B, C, D,
4      input [1:0] S,
5      output reg [`bits-1:0] Y
6  );
7      always @(*) begin
8          case (S)
9              2'b00: Y = A;
10             2'b01: Y = B;
11             2'b10: Y = C;
12             2'b11: Y = D;
13         endcase
14     end
15 endmodule
```

實驗過程-程式碼說明

使用先前定義的基本模組，根據自己設計出的 Datapath 撰寫程式碼,這裡主要要注意的是，輸入是接到哪裡的輸出，或是輸出又是輸出到哪裡。

Datapath:

Datapath.v

```
1  `timescale 1ns/1ps
2  `define bits 9
3  module Datapath ( inportR, inportG, inportB, control, clk, rst_n, outportY, outportU, outportV, done );
4      input [`bits-1:0] inportR, inportG, inportB;
5      input [21:0] control;
6      input clk, rst_n, done;
7      output [`bits-1:0] outportY, outportU, outportV;
8      wire [`bits-1:0] M1_OUT, M2_OUT, M3_OUT, M4_OUT, M5_OUT, M6_OUT, M7_OUT;
9      wire [`bits-1:0] M8_OUT, M9_OUT, Fadd, Fmul, R1, R2, R3, R4, R5, R6, data;
10     Register r1( .D(M4_OUT), .reset(rst_n), .clk(clk), .load(control[21:21]), .Q(R1) );
11     Register r2( .D(M5_OUT), .reset(rst_n), .clk(clk), .load(control[20:20]), .Q(R2) );
12     Register r3( .D(M6_OUT), .reset(rst_n), .clk(clk), .load(control[19:19]), .Q(R3) );
13     Register r4( .D(M7_OUT), .reset(rst_n), .clk(clk), .load(control[18:18]), .Q(R4) );
14     Register r5( .D(M8_OUT), .reset(rst_n), .clk(clk), .load(control[17:17]), .Q(R5) );
15     Register r6( .D(M9_OUT), .reset(rst_n), .clk(clk), .load(control[16:16]), .Q(R6) );
16     MUX3 M1 ( .A(R3), .B(R2), .C(R1), .S(control[12:11]), .Y(M1_OUT) );
17     MUX4 M2 ( .A(9'b010000000), .B(R3), .C(R5), .D(R4), .S(control[10:9]), .Y(M2_OUT) );
18     MUX3 M3 ( .A(R6), .B(R1), .C(R5), .S(control[8:7]), .Y(M3_OUT) );
19     MUX3 M4 ( .A(inportR), .B(Fmul), .C(Fadd), .S(control[6:5]), .Y(M4_OUT) );
20     MUX2 M5 ( .A(inportG), .B(Fadd), .S(control[4:4]), .Y(M5_OUT) );
21     MUX2 M6 ( .A(inportB), .B(Fadd), .S(control[3:3]), .Y(M6_OUT) );
22     MUX2 M7 ( .A(Fmul), .B(Fadd), .S(control[2:2]), .Y(M7_OUT) );
23     MUX2 M8 ( .A(Fmul), .B(Fadd), .S(control[1:1]), .Y(M8_OUT) );
24     MUX2 M9 ( .A(Fmul), .B(Fadd), .S(control[0:0]), .Y(M9_OUT) );
25     Mul FU1 ( .A(M1_OUT), .B(data), .Mul(Fmul) );
26     Add FU2 ( .A(M2_OUT), .B(M3_OUT), .Add(Fadd) );
27     ROM FU3 ( .clk(clk), .addr(control[15:13]), .data(data) );
28     Register r7(R4, rst_n, clk, done, outportY);
29     Register r9(R2, rst_n, clk, done, outportU);
30     Register r8(R1, rst_n, clk, done, outportV);
31 endmodule
```

實驗過程-程式碼說明

Controller:

此圖為 Controller 的程式碼，主要根據 state table 分別去設定每個 state 所需的控制訊號，以確保運算正確。

Controller.v

```
`timescale 1ns/1ps
`define bits 9
`define S0 4'b0000
`define S1 4'b0001
`define S2 4'b0010
`define S3 4'b0011
`define S4 4'b0100
`define S5 4'b0101
`define S6 4'b0110
`define S7 4'b0111
`define S8 4'b1000
`define S9 4'b1001
`define S10 4'b1010
`define S11 4'b1011
module Controller ( start, rst_n, clk, done, control);
    input start, rst_n, clk;
    output reg done;
    output reg [21:0] control;
    reg [3:0] Current_State, Next_State;
    always @(posedge clk or negedge rst_n) begin
        if(!rst_n) Current_State <= `S0;
        else Current_State <= Next_State;
    end
    always @(Current_State or start)
    begin
        case (Current_State)
            `S0:
            begin
                control = 22'b1_1_1_0_0_0_000_00_00_00_00_0_0_0_0_0;
                done = 1'b0;
                if(~start) Next_State = `S0;
                else Next_State = `S1;
            end
            `S1:
            begin
                control = 22'b0_0_0_1_0_0_000_10_00_00_00_0_0_0_0_0;
                done = 1'b0;
                Next_State = `S2;
            end
            `S2:
            begin
                control = 22'b0_0_0_0_1_0_001_01_00_00_00_0_0_0_0_0;
                done = 1'b0;
                Next_State = `S3;
            end
            `S3:
            begin
                control = 22'b0_0_0_1_1_0_010_00_11_10_00_0_0_1_0_0;
                done = 1'b0;
                Next_State = `S4;
            end
            `S4:
            begin
                control = 22'b0_0_0_1_1_0_011_10_11_10_00_0_0_1_0_0;
                done = 1'b0;
                Next_State = `S5;
            end
            `S5:
            begin
                control = 22'b0_0_0_0_0_1_100_01_00_00_00_0_0_0_0_0;
                done = 1'b0;
                Next_State = `S6;
            end
            `S6:
            begin
                control = 22'b0_0_0_0_1_1_101_00_10_00_00_0_0_0_1_0;
                done = 1'b0;
                Next_State = `S7;
            end
            `S7:
            begin
                control = 22'b1_0_0_0_0_1_101_10_00_00_01_0_0_0_0_1;
                done = 1'b0;
                Next_State = `S8;
            end
            `S8:
            begin
                control = 22'b0_1_0_0_1_0_110_01_10_00_00_1_0_0_0_0;
                done = 1'b0;
                Next_State = `S9;
            end
            `S9:
            begin
                control = 22'b1_0_1_0_0_0_111_00_10_01_01_0_1_0_0_0;
                done = 1'b0;
                Next_State = `S10;
            end
            `S10:
            begin
                control = 22'b1_0_0_0_0_0_000_00_00_01_10_0_0_0_0_0;
                done = 1'b0;
                Next_State = `S11;
            end
            `S11:
            begin
                control = 22'b1_0_0_0_0_0_000_00_01_01_10_0_0_0_0_0;
                done = 1'b1;
                Next_State = `S0;
            end
            default:
                begin
                    control= 22'b0_0_1_0_0_0_000_00_01_01_0_00_1_0_0_0;
                    done = 1'b1;
                    Next_State = `S0;
                end
        endcase
    end
endmodule
```

實驗過程-程式碼說明

最後一步參考提供的程式碼完成整個電路的 Verilog code, 接著再用 testbench 測試是否實作正確。

RGB2YUV:

RGB2YUV.v

```
1  `timescale 1ns/1ps
2  `define bits 9
3
4  module RGB2YUV (start, clk, rst_n, inportR, inportG, inportB, done, outportY, outportU, outportV);
5      input start, clk, rst_n;
6      input [`bits-1:0] inportR, inportG, inportB;
7      output done;
8      output [`bits-1:0] outportY, outportU, outportV;
9      wire [21:0] control;
10     Controller Controller( .start(start), .rst_n(rst_n), .clk(clk), .done(done), .control(control) );
11     Datapath Datapath( .inportR(inportR), .inportG(inportG), .inportB(inportB), .control(control), .clk(clk),
12     .rst_n(rst_n), .outportY(outportY), .outportU(outportU), .outportV(outportV) ,.done(done) );
13 endmodule
```

實驗結果及分析

模擬結果後，原本 RGB 的圖片生成 3 張圖片，分別對應 Y,U,V 的輸出。

mountain256.bmp:

mountain256Y.bmp:

mountain256U.bmp:

mountain256V.bmp:

實驗心得

一開始看到這次實驗的時候覺得很難，不過還好此次實驗有提供大部分的程式碼，讓我們只需要專注在最重要的部分就好。我首先畫了 Scheduling, Allocation, Datapath 以及 STG and state table，雖然過程偏繁複，但是實際自己畫過一遍，遍歷整個電路的流程的過程中，更清楚了其中的邏輯也了解實際在設計電路過程中會遇到的問題，撰寫程式碼也有幫我找回之前數位系統碰過一些 Verilog 的記憶，此次實驗中，我不僅了解其中的原理，也更深入地理解電路實際在跑所需要注意的地方，雖然此次實驗較為複雜，但看到結果出來的那一刻真的很值得也很有成就感。