RTL Design of RGB to YUV

2017/6/20

范真瑋

RTL Design of RGB to YUV

實驗內容：

撰寫 RGB to YUV 電路的 Verilog code

以提供的 testbench.v 測試 RGB to YUV 電路的 Verilog code 是否正確

透過 ISE，將 RGB to YUV 電路下載至 FPGA 並進行軟硬體溝通及驗證

程式碼：

Multipliper.v

```
1  `timescale 1ns/1ps
2  `define bits 9
3
4  module Mul(A, B, Mul);
5   input signed [`bits-1:0]A, B;
6   output [`bits-1:0] Mul;
7   wire s;
8   wire [7:0] N;
9
10  assign {s, Mul, N} = A * B;
11
12 endmodule
```

Add.v

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

MUX4.v

```
`timescale 1ns/1ps
`define bits 9

module MUX4(
 input [`bits-1:0]A, B, C, D,
 input [1:0]S,
 output reg[`bits-1:0]Y
);

always @(*)begin
 case(S)
   2'b00: Y = A;
   2'b01: Y = B;
   2'b10: Y = C;
   2'b11: Y = D;
 endcase
end

endmodule
```

MUX3.v

```
`timescale 1ns/1ps
`define bits 9

module MUX3(
 input [`bits-1:0]A, B, C,
 input [1:0]S,
 output reg[`bits-1:0]Y
);

always @(*)begin
 case(S)
   2'b00: Y = A;
   2'b01: Y = B;
   2'b10: Y = C;
   default : Y = A;
 endcase
end

endmodule
```

MUX2.v

```
`timescale 1ns/1ps
`define bits 9

module MUX2(
input [`bits-1:0]A, B,
input [0:0]S,
output reg [`bits-1:0]Y
);

always @(*)begin
 case (S)
   1'b0: Y = A;
   1'b1: Y = B;
   default : Y = A;
 endcase
end

endmodule
```

Register.v

```
`timescale lns/lps
`define bits 9

module Register(
 input [`bits-1:0] D,
 input reset, clk, load,
 output reg [`bits-1:0]Q
);

always @(posedge clk or negedge reset) begin
  if(!reset)
      Q <= 9'b0;
  else if(load)
      Q <= D;
end

endmodule
```

ROM.v

```
`timescale lns / lps

module ROM(clk, addr, data);
 input       clk;
 input [2:0] addr;
 output reg[8:0] data;

 always @(*)
 begin
    case(addr)
       3'b000: data <= 9'b001001100;
       3'b001: data <= 9'b010010110;
       3'b010: data <= 9'b111010101;
       3'b011: data <= 9'b110101100;
       3'b100: data <= 9'b010000000;
       3'b101: data <= 9'b110010101;
       3'b110: data <= 9'b111101100;
       3'b111: data <= 9'b000011101;
    endcase
 end

endmodule
```

Datapath.v

```
`timescale lns/lps
`define bits 9

module Datapath(inportR, inportG, inportB, control, clk, rst_n, outportY, outportU, outportV, done);
 input [`bits-1:0]inportR, inportG, inportB;
 input [21:0]control;
 input clk, rst_n, done;
 output [`bits-1:0] outportY, outportU, outportV;

 wire [`bits-1:0] M1_OUT, M2_OUT, M3_OUT, M4_OUT, M5_OUT, M6_OUT, M7_OUT;
 wire [`bits-1:0] M8_OUT, M9_OUT, Fadd, Fmul, R1, R2, R3, R4, R5, R6, data;

 Register r1( .D(M4_OUT), .reset(rst_n), .clk(clk), .load(control[21:21]), .Q(R1));
 Register r2( .D(M5_OUT), .reset(rst_n), .clk(clk), .load(control[20:20]), .Q(R2));
 Register r3( .D(M6_OUT), .reset(rst_n), .clk(clk), .load(control[19:19]), .Q(R3));
 Register r4( .D(M7_OUT), .reset(rst_n), .clk(clk), .load(control[18:18]), .Q(R4));
 Register r5( .D(M8_OUT), .reset(rst_n), .clk(clk), .load(control[17:17]), .Q(R5));
 Register r6( .D(M9_OUT), .reset(rst_n), .clk(clk), .load(control[16:16]), .Q(R6));
 MUX3 M1( .A(R3), .B(R2), .C(R1), .S(control[12:11]), .Y(M1_OUT) );
 MUX4 M2( .A(9'b010000000), .B(R5), .C(R4), .D(R1), .S(control[10:9]), .Y(M2_OUT) );
 MUX4 M3( .A(R6), .B(R3), .C(R2), .D(R5), .S(control[8:7]), .Y(M3_OUT) );
 MUX2 M4( .A(inportR), .B(Fadd), .S(control[6:6]), .Y(M4_OUT) );
 MUX3 M5( .A(inportG), .B(Fmul), .C(Fadd), .S(control[5:4]), .Y(M5_OUT) );
 MUX2 M6( .A(inportB), .B(Fadd), .S(control[3:3]), .Y(M6_OUT) );
 MUX2 M7( .A(Fmul), .B(Fadd), .S(control[2:2]), .Y(M7_OUT) );
 MUX2 M8( .A(Fmul), .B(Fadd), .S(control[1:1]), .Y(M8_OUT) );
 MUX2 M9( .A(Fmul), .B(Fadd), .S(control[0:0]), .Y(M9_OUT) );
 Mul FU1( .A(M1_OUT), .B(data), .Mul(Fmul) );
 Add FU2( .A(M2_OUT), .B(M3_OUT), .Add(Fadd) );
 ROM FU3( .clk(clk), .addr(control[15:13]), .data(data) );

 Register r7(R2, rst_n, clk, done, outportY);
 Register r9(R1, rst_n, clk, done, outportU);
 Register r8(R3, rst_n, clk, done, outportV);
endmodule
```

Controller.v

```
`timescale lns/lps
`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_0_00_0_0_0_0;//control = 22'br1_r2_r3_r4_r5_r6_ROM_M1_M2_M3_M4_M5_M6_M7_M8_M9;
      done = 1'b0;
      if(~start)
         Next_State = `S0;
      else
         Next_State = `S1;
   end
```

```
43
44      `S1:
45      begin
46         control = 22'b0_0_0_1_0_0_000_10_00_00_0_00_0_0_0_0;
47         done = 1'b0;
48         Next_State = `S2;
49      end
50
51      `S2:
52      begin
53         control = 22'b0_0_0_0_1_0_001_01_00_00_0_00_0_0_0_0;
54         done = 1'b0;
55         Next_State = `S3;
56      end
57
58      `S3:
59      begin
60         control = 22'b0_0_0_1_1_0_010_10_10_11_0_00_0_1_0_0;
61         done = 1'b0;
62         Next_State = `S4;
63      end
64
65      `S4:
66      begin
67         control = 22'b0_0_0_0_0_1_011_01_00_00_0_00_0_0_0_0;
68         done = 1'b0;
69         Next_State = `S5;
70      end
71
72      `S5:
73      begin
74         control = 22'b0_0_0_0_1_1_100_00_01_00_0_00_0_0_0_1;
75         done = 1'b0;
76         Next_State = `S6;
77      end
78
79      `S6:
80      begin
81         control = 22'b1_0_0_0_1_0_100_10_00_11_1_00_0_0_0_0;
82         done = 1'b0;
83         Next_State = `S7;
84      end
```

```
85
86      `S7:
87      begin
88         control = 22'b1_1_0_0_0_0_101_01_11_00_1_01_0_0_0_0;
89         done = 1'b0;
90         Next_State = `S8;
91      end
92
93      `S8:
94      begin
95         control = 22'b0_1_0_0_1_0_110_00_01_10_0_01_0_0_1_0;
96         done = 1'b0;
97         Next_State = `S9;
98      end
99
100     `S9:
101     begin
102        control = 22'b0_1_1_0_0_0_111_00_00_10_0_01_1_0_0_0;
103        done = 1'b0;
104        Next_State = `S10;
105     end
106
107     `S10:
108     begin
109        control = 22'b0_1_0_0_0_0_000_00_10_10_0_10_0_0_0_0;
110        done = 1'b0;
111        Next_State = `S11;
112     end
113
114     `S11:
115     begin
116        control = 22'b0_0_1_0_0_0_000_00_01_01_0_00_1_0_0_0;
117        done = 1'b1;
118        Next_State = `S0;
119     end
120
121     default:
122     begin
123        control = 22'b0_0_1_0_0_0_000_00_01_01_0_00_1_0_0_0;
124        done = 1'b1;
125        Next_State = `S0;
126     end
127     endcase
128 end
129
130 endmodule
```

RGB2YUV.v

```
`timescale 1ns / 1ps
`define bits 9

module RGB2YUV(start, clk, rst_n, inportR, inportG, inportB, done, outportY, outportU, outportV);
input start, clk, rst_n;
input [`bits-1:0] inportR, inportG, inportB;
output done;
output [`bits-1:0] outportY, outportU, outportV;
wire [21:0] control;

Controller Controller( .start(start), .rst_n(rst_n), .clk(clk), .done(done), .control(control) );
Datapath Datapath( .inportR(inportR), .inportG(inportG), .inportB(inportB), .control(control), .clk(clk),
   .rst_n(rst_n), .outportY(outportY), .outportU(outportU), .outportV(outportV) ,.done(done) );

endmodule
```

testbench.v

```
`timescale 1ns/1ns
`define bits 9

module testbench();
  parameter half_clk = 20;
  parameter clk_period = 2 * half_clk;
  integer imageIN,imageOUTY,imageOUTU,imageOUTV, i, cc, j;
  integer bmp_width, bmp_hight, data_start_index, bmp_size;
  reg [7:0]  bmp_data [0:200000];
  reg rst,clk,start;
  wire [8:0] outY,outU,outV;
  wire done;
  reg[8:0] tempY;
initial begin
  clk = 1'b0;
  rst = 1'b1;
  #(clk_period) rst = ~rst;
  #(clk_period) rst = ~rst;
end

always
  #(half_clk) clk = ~clk;

RGB2YUV rgbtuv(start,clk,rst,{1'b0,bmp_data[(i-data_start_index)*3+data_start_index+2]},{1'b0,bmp_data[(i-data_start_index)*3+data_start_index+1]},
              {1'b0,bmp_data[(i-data_start_index)*3+data_start_index]},done,outY,outU,outV);

  initial begin
    start= 1'b0;
    imageIN = $fopen("mountain256.bmp","rb");
    imageOUTY = $fopen("mountain256Y.bmp","wb");
    imageOUTU = $fopen("mountain256U.bmp","wb");
    imageOUTV = $fopen("mountain256V.bmp","wb");
    cc = $fread(bmp_data,imageIN);

```

```
35      bmp_width = {bmp_data[22],bmp_data[21],bmp_data[20],bmp_data[19]};
36      bmp_hight = {bmp_data[26],bmp_data[25],bmp_data[24],bmp_data[23]};
37      data_start_index = {bmp_data[14],bmp_data[13],bmp_data[12],bmp_data[11]};
38      bmp_size  = {bmp_data[6],bmp_data[5],bmp_data[4],bmp_data[3]};
39      i = data_start_index;
40
41      for(j = 1; j < 55; j = j + 1) begin
42        #(clk_period*1)
43        tempY = bmp_data[j];
44        $fwrite(imageOUTY,"%c",bmp_data[j]);
45        $fwrite(imageOUTU,"%c",bmp_data[j]);
46        $fwrite(imageOUTV,"%c",bmp_data[j]);
47      end
48
49       #(clk_period*1)
50        start= 1'b1;
51        #(clk_period*1)
52      for(i = data_start_index; i < bmp_width*bmp_hight+data_start_index; i = i + 1) begin
53        // start= 1'b1;
54        #(clk_period*12)
55        $fwrite(imageOUTY,"%c%c%c",outY[7:0],outY[7:0],outY[7:0]);
56        $fwrite(imageOUTU,"%c%c%c",outU[7:0],outU[7:0],outU[7:0]);
57        $fwrite(imageOUTV,"%c%c%c",outV[7:0],outV[7:0],outV[7:0]);
58      end
59
60      #(clk_period*2)
61      $fclose(imageOUTY);
62      $fclose(imageOUTU);
63      $fclose(imageOUTV);
64      $fclose(imageIN);
65      $finish;
66
67    end
68
69  endmodule
```

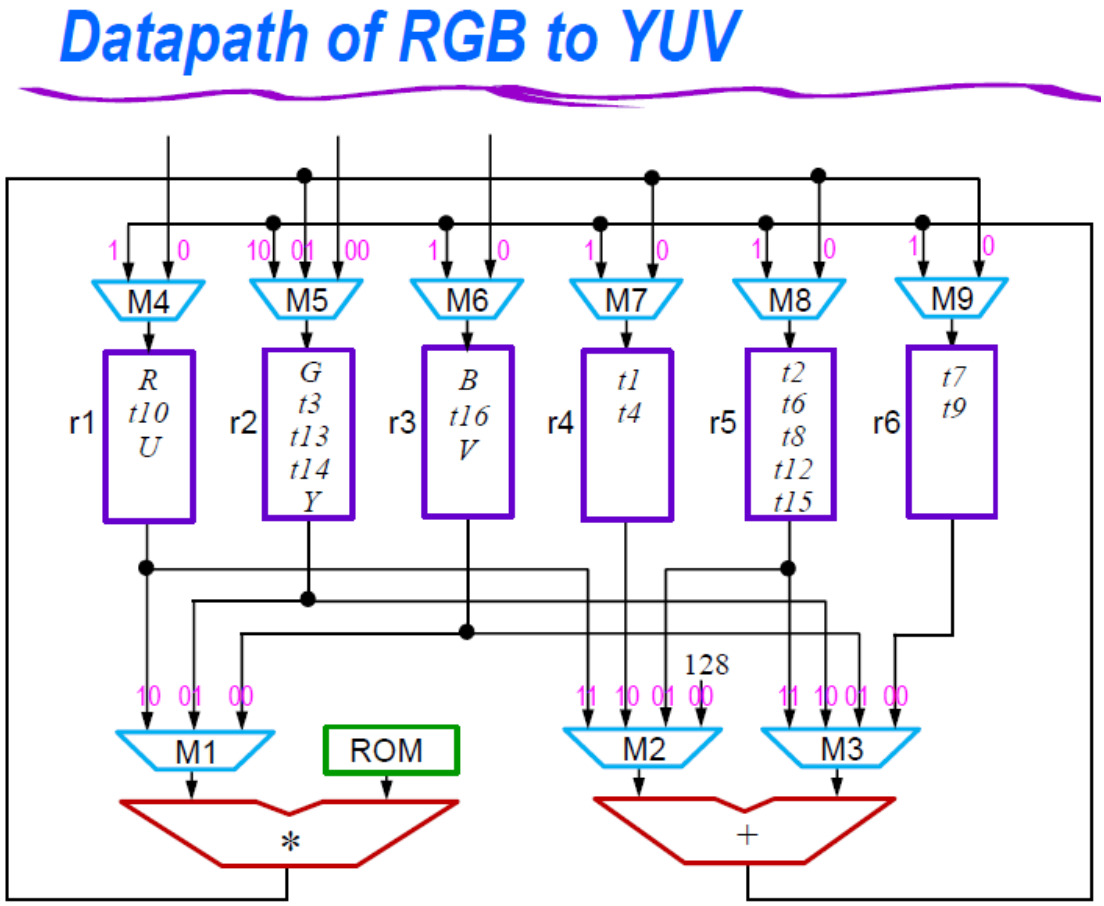
實驗結果及分析：

ROM.v 參考下圖，將小數轉為二進位，負數則使用二補數的方法表示。

ROM

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

Datapath.v 參考下圖，建立 MUX 對應的輸入與輸出。



Controller.v 參考以下兩表格，設定對應的 control。

//control = 22'br1_r2_r3_r4_r5_r6_ROM_M1_M2_M3_M4_M5_M6_M7_M8_M9;

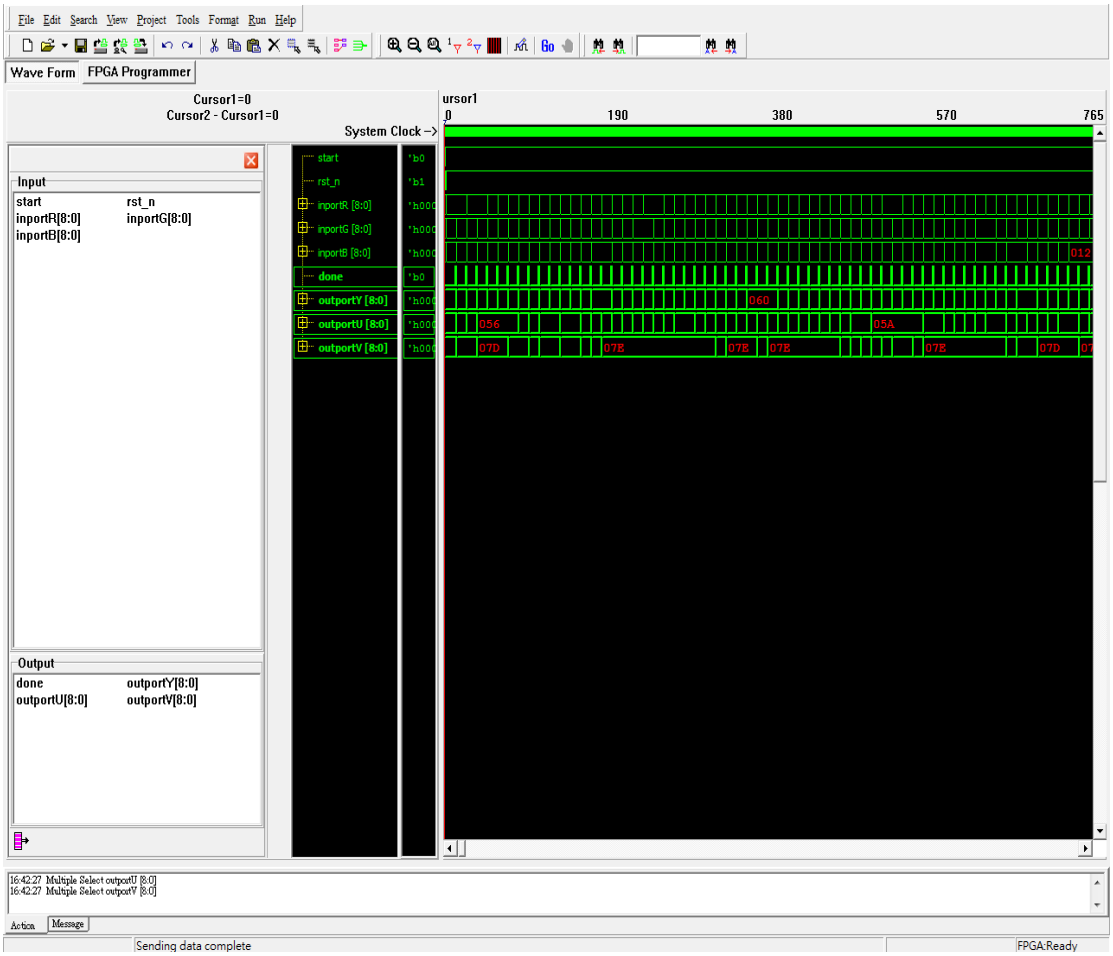
Present State	Input	Next State	Control Signals							
			r1	r2	r3	r4	r5	r6	ROM	done
S0	Start = 0	S0	1	1	1	0	0	0	-	0
	Start = 1	S1								
S1	-	S2	0	0	0	1	0	0	000	0
S2	-	S3	0	0	0	0	1	0	001	0
S3	-	S4	0	0	0	1	1	0	010	0
S4	-	S5	0	0	0	0	0	1	011	0
S5	-	S6	0	0	0	0	1	1	100	0
S6	-	S7	1	0	0	0	1	0	100	0
S7	-	S8	1	1	0	0	0	0	101	0
S8	-	S9	0	1	0	0	1	0	110	0
S9	-	S10	0	1	1	0	0	0	111	0
S10	-	S11	0	1	0	0	0	0	-	0
S11	-	S0	0	0	1	0	0	0	-	1

Present State	Input	Next State	Control Signals								
			M1	M2	M3	M4	M5	M6	M7	M8	M9
S0	Start = 0	S0	-	-	-	0	00	0	-	-	-
	Start = 1	S1									
S1	-	S2	10	-	-	-	-	-	0	-	-
S2	-	S3	01	-	-	-	-	-	-	0	-
S3	-	S4	10	10	11	-	-	-	1	0	-
S4	-	S5	01	-	-	-	-	-	-	-	0
S5	-	S6	00	01	00	-	-	-	-	0	1
S6	-	S7	10	00	11	1	-	-	-	0	-
S7	-	S8	01	11	00	1	01	-	-	-	-
S8	-	S9	00	01	10	-	01	-	-	1	-
S9	-	S10	00	00	10	-	01	1	-	-	-
S10	-	S11	-	10	10	-	10	-	-	-	-
S11	-	S0	-	01	01	-	-	1	-	-	-

Verilog code 完成後，使用 HDL Auto Assign Pin 產生 ucf 檔，並將 ucf 檔加入專案，把程式碼燒錄至 FPGA 實驗板。之後使用 VeriComm 產生 out 檔。再來將 mountain256.bmp、FPGA 訊號輸出的檔案和程式放在同一個資料夾下，最後執行程式，即可產生圖片。

FPGA

VeriComm


File Edit Search View Project Tools Format Run Help
Wave Form
FPGA Programmer
Cursor1=0
Cursor2 - Cursor1=0
System Clock –>
ursor1
0
190
380
570
765
Input
start
rst_n
inportR[8:0]
inportG[8:0]
inportB[8:0]
Output
done
outportY[8:0]
outportU[8:0]
outportV[8:0]
start
rst_n
inportR [8:0]
inportG [8:0]
inportB [8:0]
done
outportY [8:0]
outportU [8:0]
outportV [8:0]
16:42:27 Multiple Select outportU [8:0]
16:42:27 Multiple Select outportV [8:0]
Action
Message
Sending data complete
FPGA:Ready

mountain256.bmp

mountain256_Y

mountain256_U

mountain256_V