

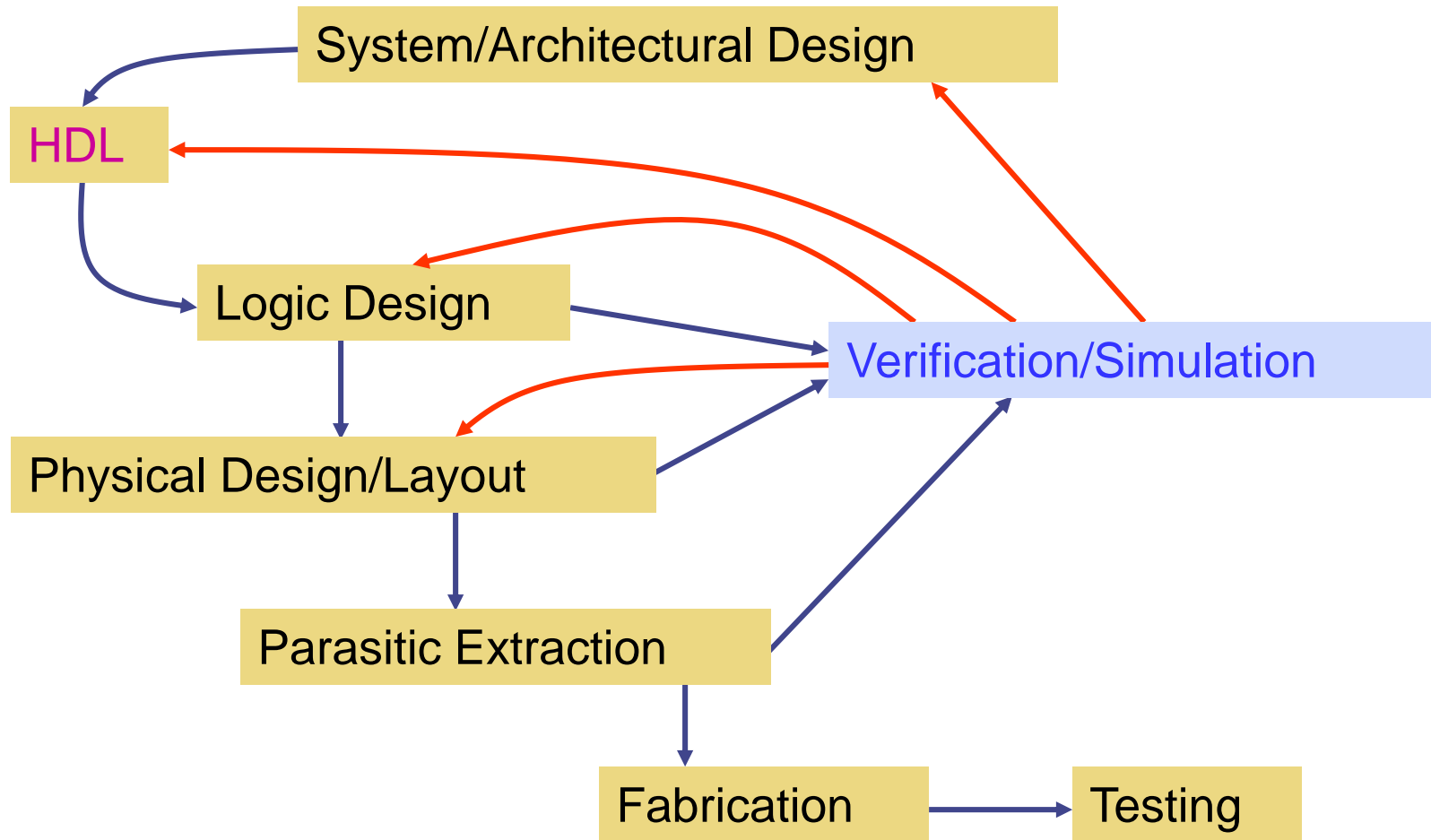


Lab 3: RTL Design of RGB to YUV

Lab 3: RTL Design of RGB to YUV

- 使用VCS, Verdi，並且參考以下的投影片撰寫 RGB to YUV 電路的Verilog code
 - ◆ 包含Multiplier.v, Add.v, MUX2.v, MUX3.v, MUX4.v, Register.v, ROM.v, Datapath.v, Controller.v以及RGB2YUV.v
- 以提供的testbench.v測試RGB to YUV 電路的Verilog code是否正確！
- 繳交RGB to YUV 電路的Verilog code及實驗報告至“中山大學網路大學-作業評量區”

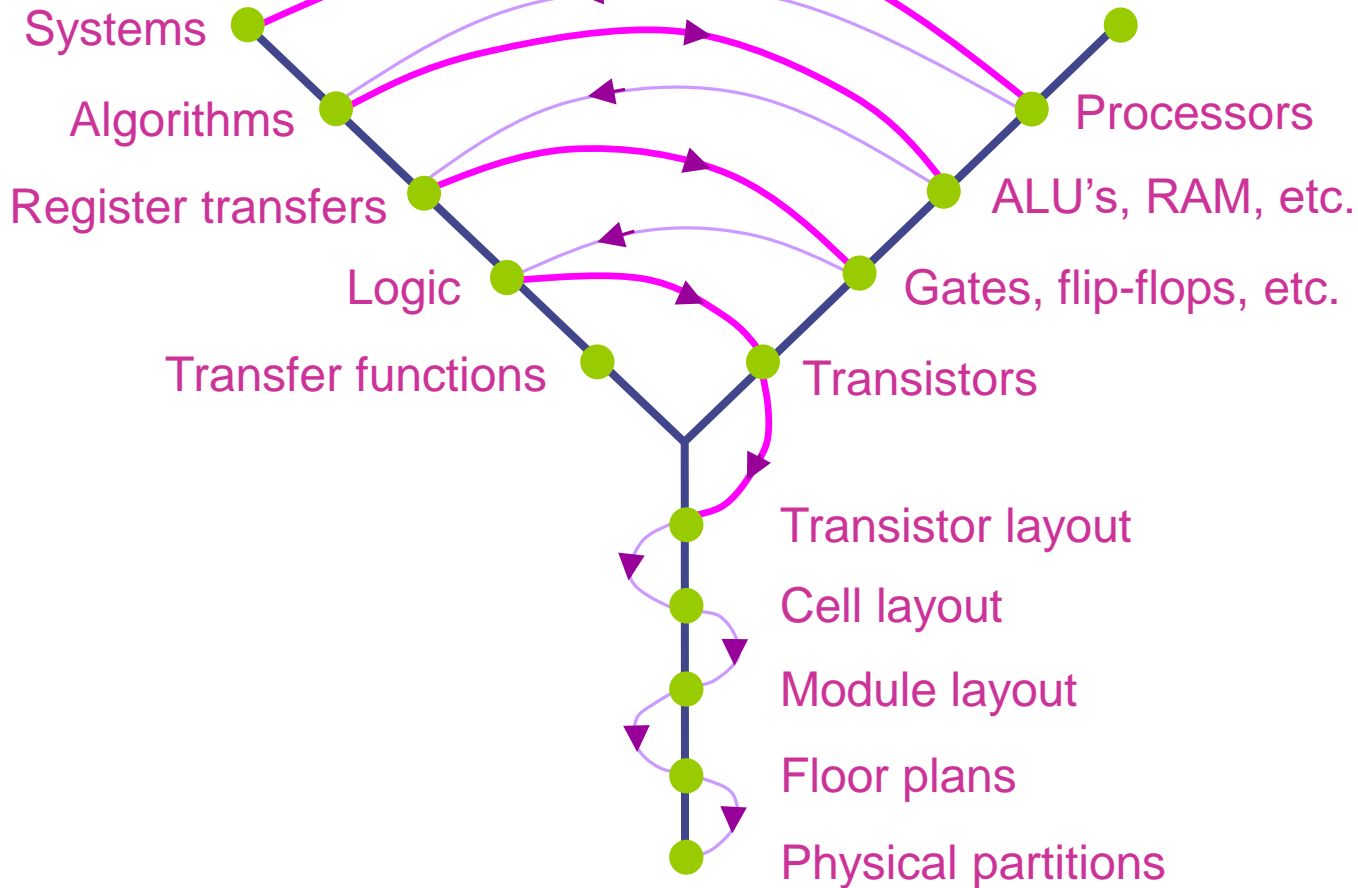
Basic Design Flow



Design Flow

BEHAVIORAL DOMAIN

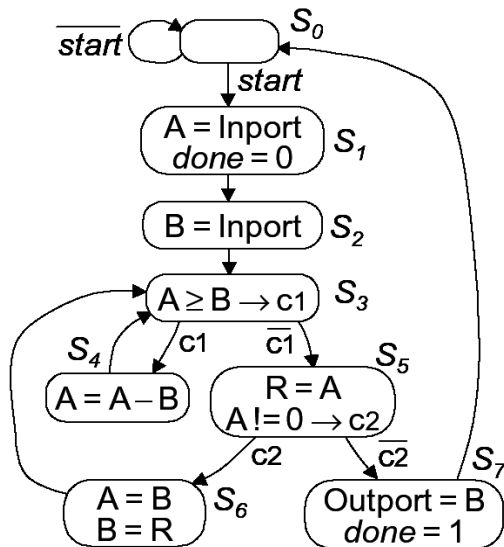
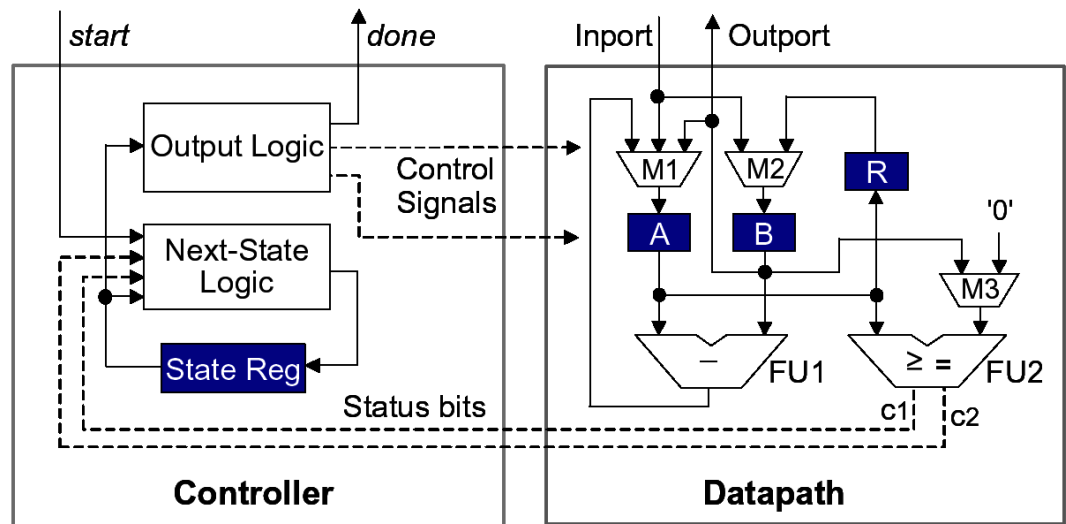
STRUCTURAL DOMAIN



Behavioral Synthesis

```

A = Inport;
B = Inport;
done = 0;
Repeat: while (A ≥ B)
    A = A - B;
    R = A;
    if (A != 0) {
        A = B;
        B = R;
        goto Repeat;
    }
    else goto End;
End:   Output = B;
      done = 1;
  
```



| PS | Input | NS | Control signals | | | | | | |
|-------|-------------|-------|-----------------|---|---|--------|--------|-----|------|
| | | | A | B | R | M1 | M2 | M3 | done |
| S_0 | $start = 0$ | S_0 | - | - | - | - | - | - | 0 |
| | $start = 1$ | S_1 | - | - | - | - | - | - | 0 |
| S_1 | - | S_2 | 1 | - | - | Inport | - | - | 0 |
| S_2 | - | S_3 | 0 | 1 | - | - | Inport | - | 0 |
| S_3 | $c1 = 1$ | S_4 | 0 | 0 | 0 | - | - | B | 0 |
| | $c1 = 0$ | S_5 | 0 | 0 | 0 | - | - | - | 0 |
| S_4 | - | S_3 | 1 | 0 | 0 | FU1 | - | - | 0 |
| S_5 | $c2 = 1$ | S_6 | 0 | 0 | 1 | - | - | '0' | 0 |
| | $c2 = 0$ | S_7 | 0 | 0 | 1 | - | - | - | 0 |
| S_6 | - | S_3 | 1 | 1 | 0 | B | R | - | 0 |
| S_7 | - | S_0 | 0 | 0 | 0 | - | - | - | 1 |

RGB to YUV

$$Y = 0.299 \times R + 0.587 \times G + 0.114 \times B$$

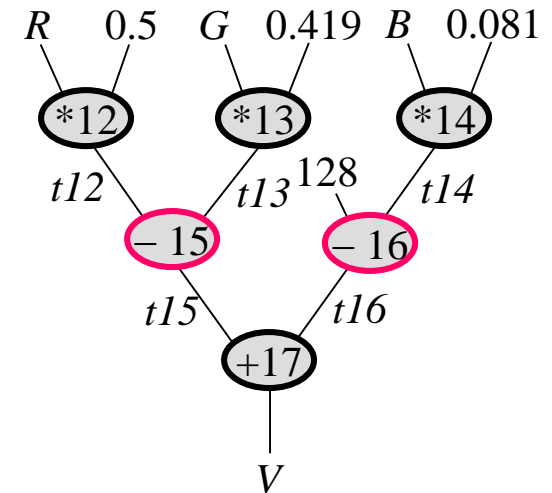
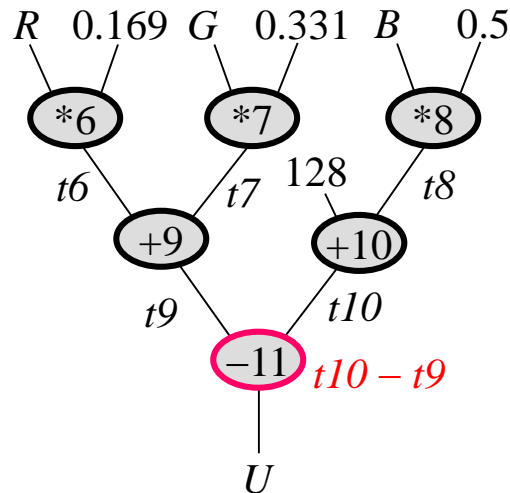
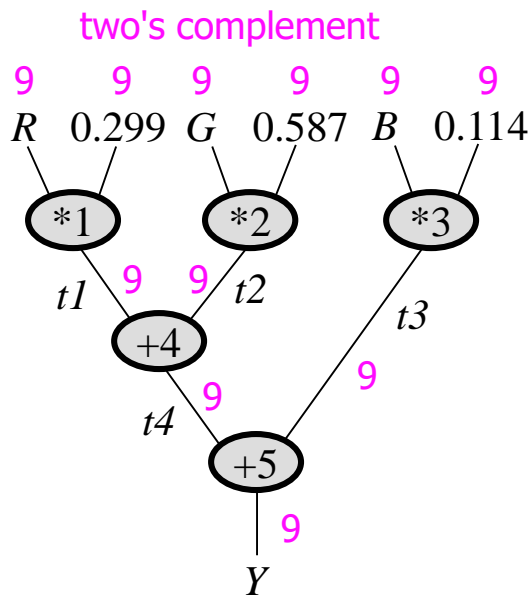
$$Y \in [0, 255]$$

$$U = -0.169 \times R - 0.331 \times G + 0.5 \times B + 128$$

$$U \in [0, 255]$$

$$V = 0.5 \times R - 0.419 \times G - 0.081 \times B + 128$$

$$V \in [0, 255]$$

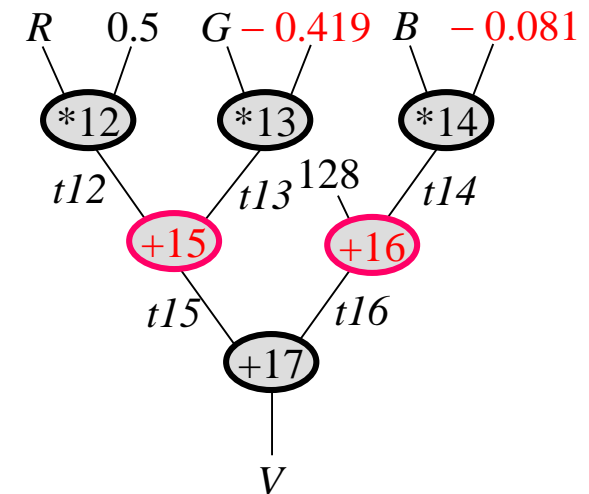
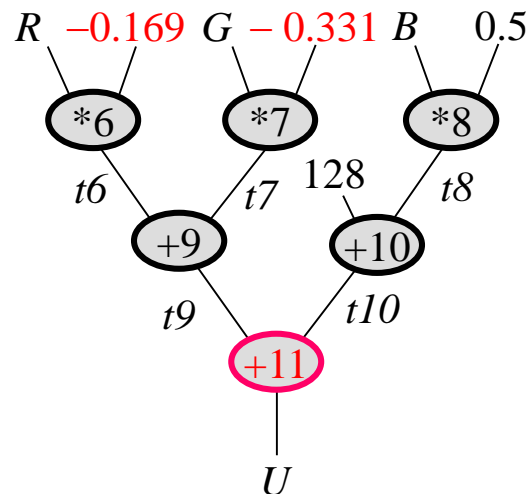
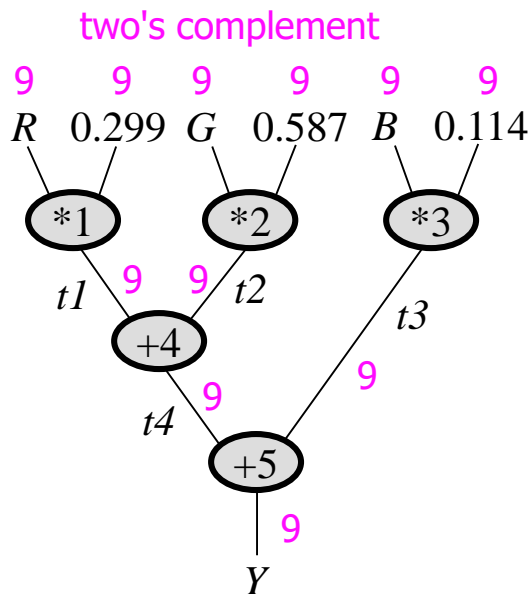


High-level Transformations

$$Y = 0.299 \times R + 0.587 \times G + 0.114 \times B \quad Y \in [0, 255]$$

$$U = \underline{(-0.169)} \times R + \underline{(-0.331)} \times G + 0.5 \times B + 128 \quad U \in [0, 255]$$

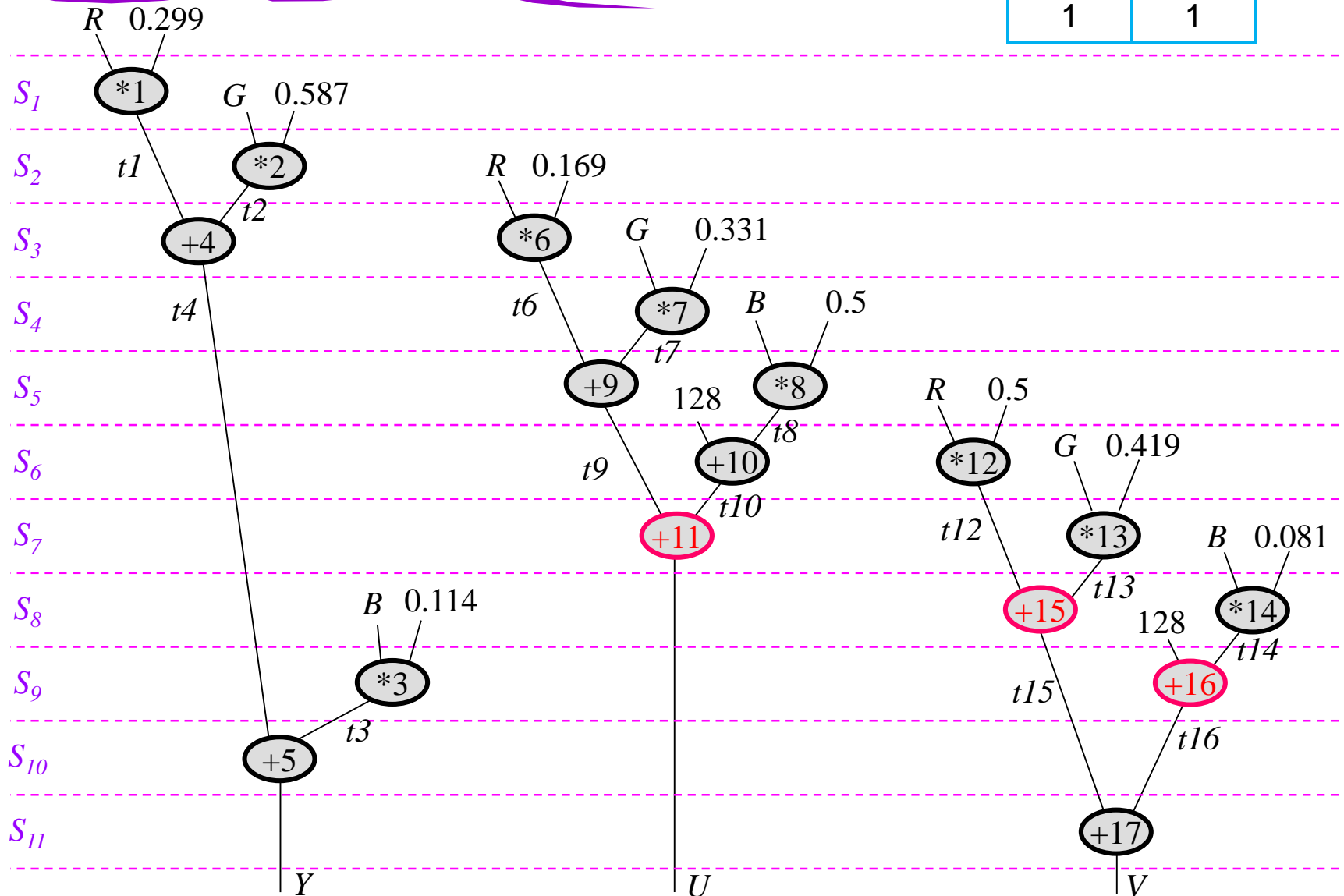
$$V = 0.5 \times R + \underline{(-0.419)} \times G + \underline{(-0.081)} \times B + 128 \quad V \in [0, 255]$$



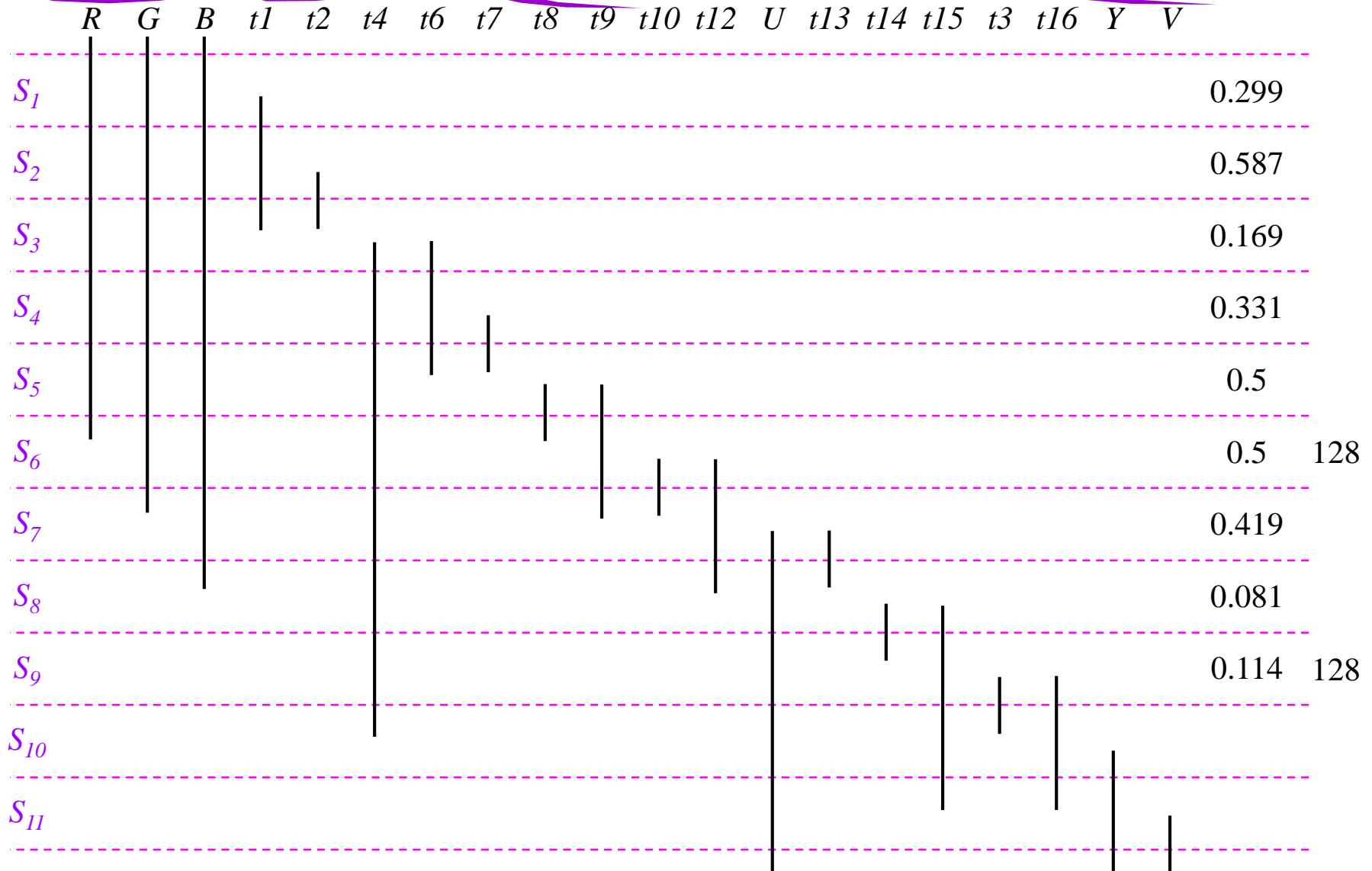
Scheduling of RGB to YUV

Resource constraint:

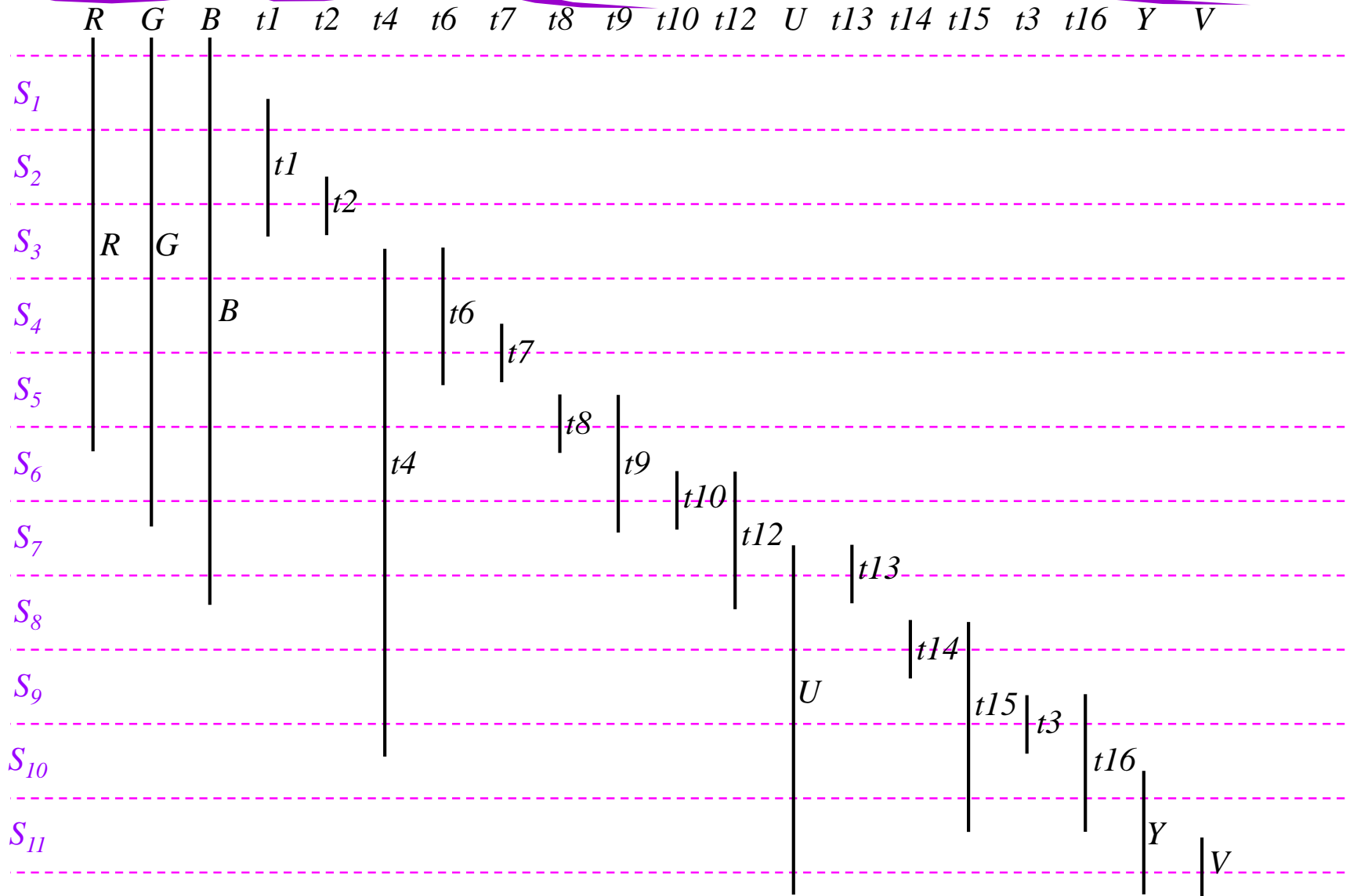
| | |
|---|---|
| * | + |
| 1 | 1 |



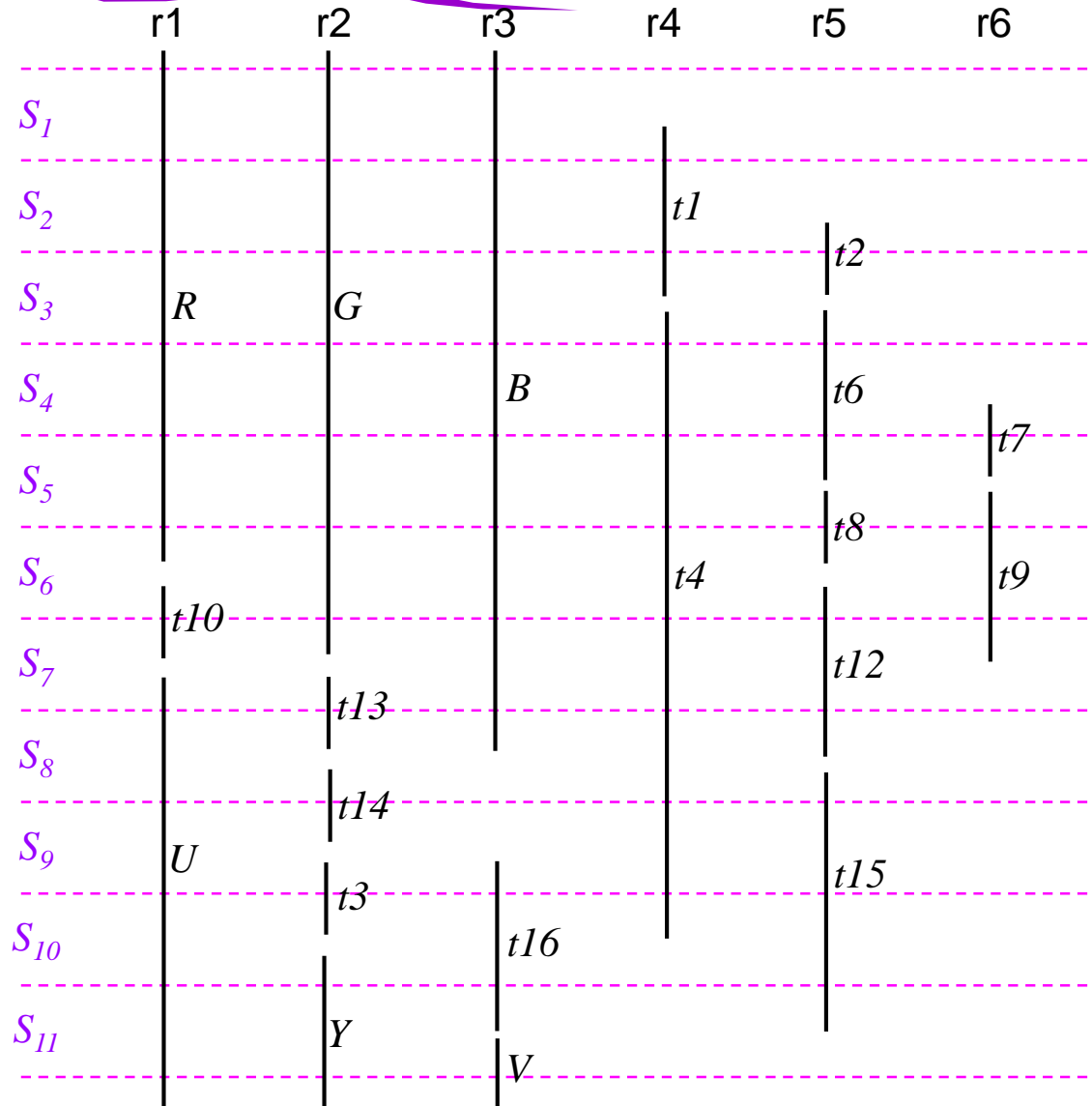
Allocation of RGB to YUV (1/3)



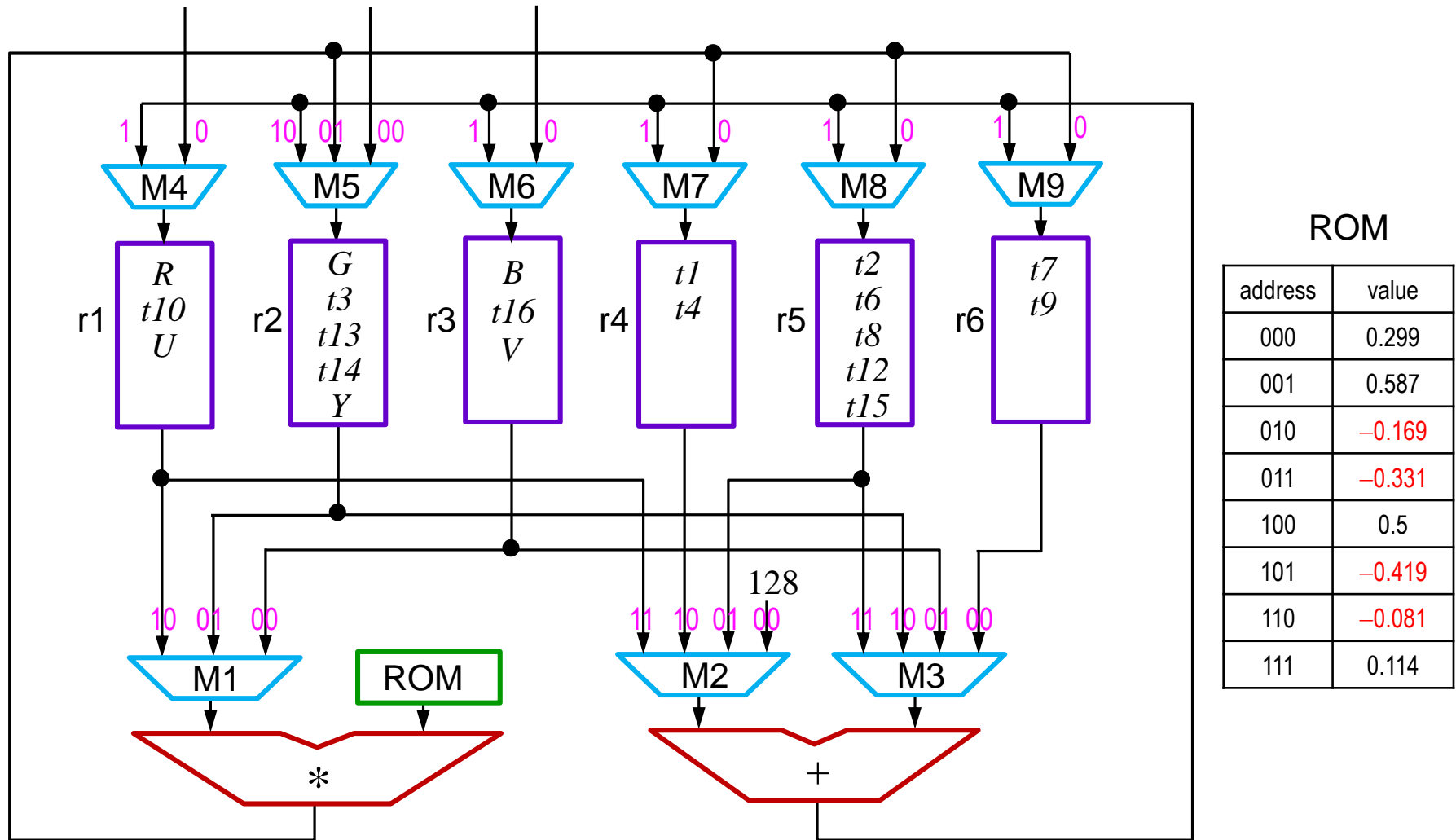
Allocation of RGB to YUV (2/3)



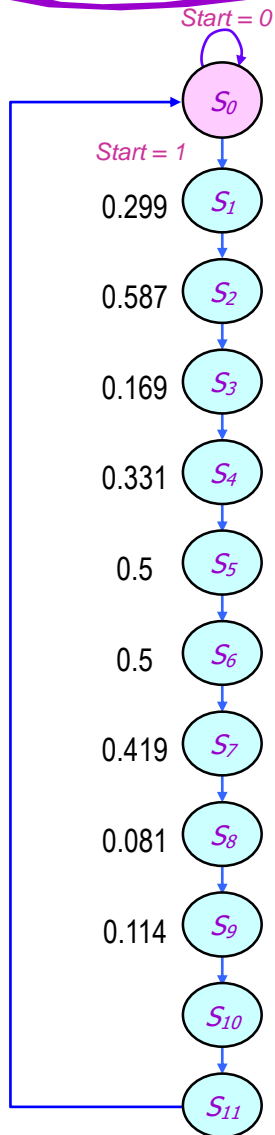
Allocation of RGB to YUV (3/3)



Datapath of RGB to YUV



STG and State Table of RGB to YUV (1/2)



| 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 |

STG and State Table of RGB to YUV (2/2)

| 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 of RGB to YUV (1/8)

``define bits 9` Multiplier.v

```
module Mul(A, B, Mul);  
  input signed [`bits-1:0] A, B;  
  output [`bits-1:0] Mul;  
  wire s;  
  wire [7:0] N;
```

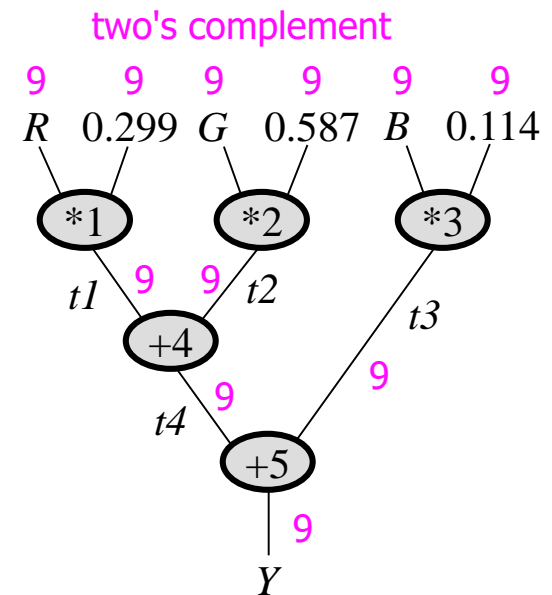
```
  assign {s, Mul, N} = A * B;
```

```
endmodule
```

```
module Add(A, B, Add);      Add.v  
  input signed [`bits-1:0] A, B;  
  output [`bits-1:0] Add;
```

```
  assign Add = A + B;
```

```
endmodule
```



Verilog of RGB to YUV (2/8)

`define bits 9 **MUX4.v**

```
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
```

`define bits 9 **MUX3.v**

```
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
```


Verilog of RGB to YUV (3/8)

```
`define bits 9    Register.v
```

```
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
```

Verilog of RGB to YUV (4/8)

```
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 <= ... ;  
            ...  
        endcase  
    end  
endmodule
```

ROM.v

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 |

Verilog of RGB to YUV (5/8)

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

Datapath.v

```
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 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 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
```

Verilog of RGB to YUV (6/8)

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_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_0_0_0_0;
                done = 1'b0;
                Next_State = `S2;
            end
        ...
        `S11:
            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
        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
```

Verilog of RGB to YUV (7/8)

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

RGB2YUV.v

```
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
```

Verilog of RGB to YUV (8/8)

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

testbench.v

```
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 [0:2000000];
  reg rst,clk,start;
  wire [8:0] outY,outU,outV;
  wire done;

  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,...,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);
    ...

    for(j = 0; j < 54; j = j + 1) begin
      $fwrite(imageOUTY,"%c",bmp_data[j]);
      $fwrite(imageOUTU,"%c",bmp_data[j]);
      $fwrite(imageOUTV,"%c",bmp_data[j]);
    end

    ...

    #(clk_period*2)
    $fclose(imageOUTY);
    $fclose(imageOUTU);
    $fclose(imageOUTV);
    $fclose(imageIN);
    $finish;

  end

endmodule
```