電子系統層級設計與驗證

Homework 2

Complete the SystemC program of

RGB to YUV with

sc_fixed or sc_ufixed data type

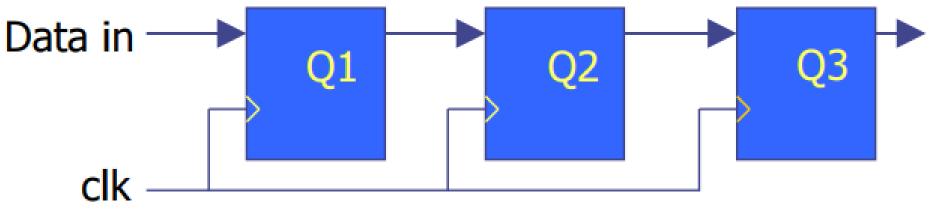
學號：B103090046

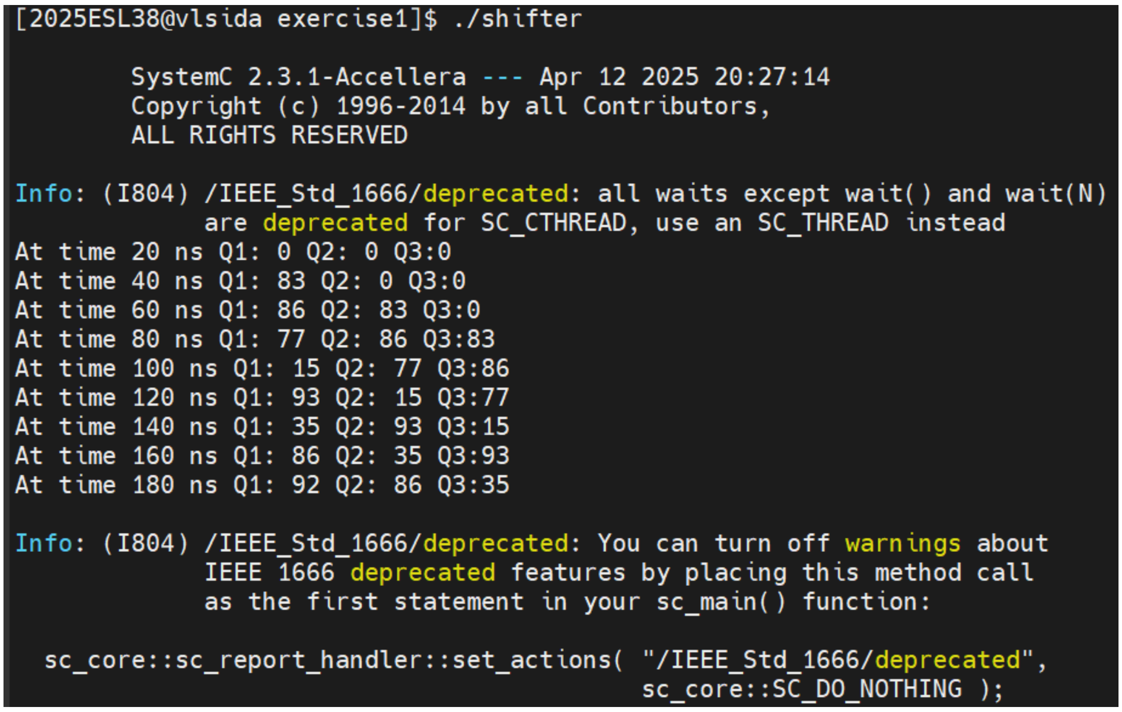
姓名：宋易柔

日期：113/04/15

（一）實驗過程

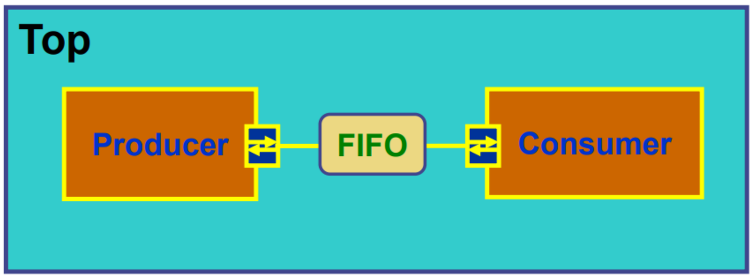
(1) Exercise 1: Shifter

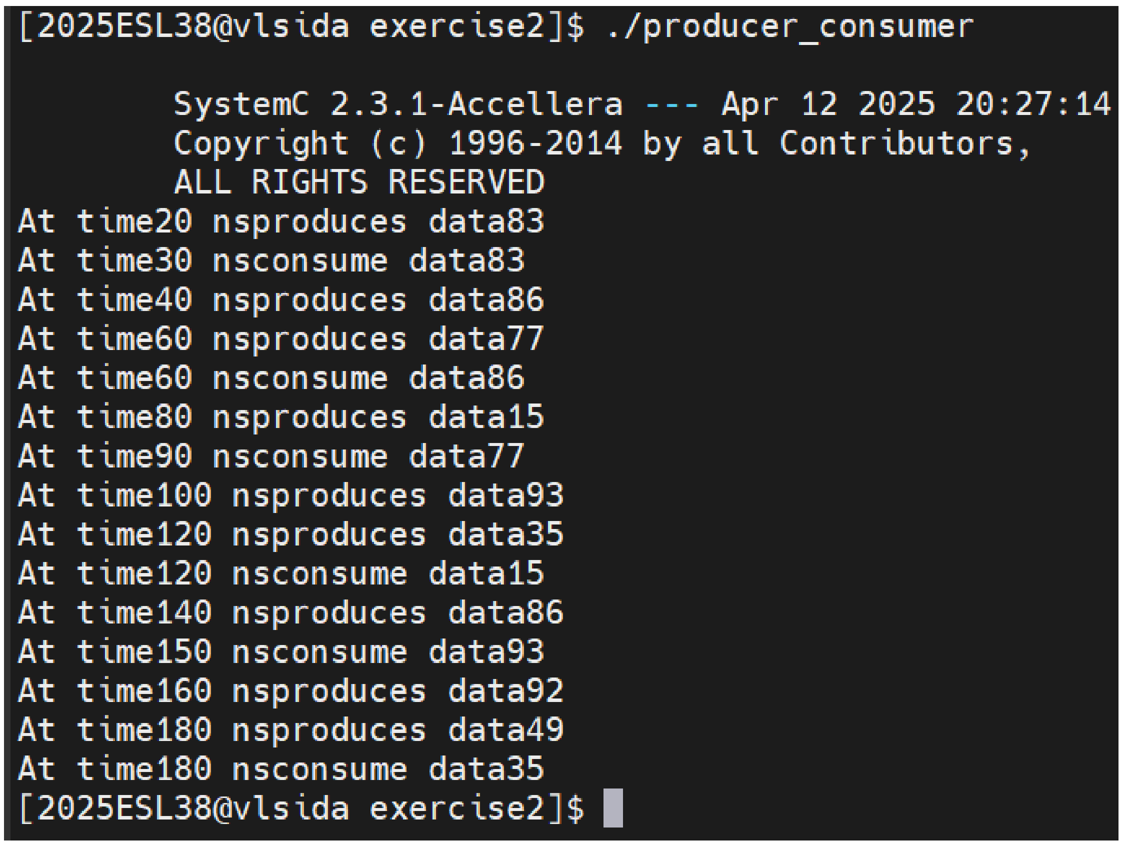




```
[2025ESL38@vlsida exercise1]$ ./shifter

        SystemC 2.3.1-Accellera --- Apr 12 2025 20:27:14
        Copyright (c) 1996-2014 by all Contributors,
        ALL RIGHTS RESERVED

Info: (I804) /IEEE_Std_1666/deprecated: all waits except wait() and wait(N)
             are deprecated for SC_CTHREAD, use an SC_THREAD instead
At time 20 ns Q1: 0 Q2: 0 Q3:0
At time 40 ns Q1: 83 Q2: 0 Q3:0
At time 60 ns Q1: 86 Q2: 83 Q3:0
At time 80 ns Q1: 77 Q2: 86 Q3:83
At time 100 ns Q1: 15 Q2: 77 Q3:86
At time 120 ns Q1: 93 Q2: 15 Q3:77
At time 140 ns Q1: 35 Q2: 93 Q3:15
At time 160 ns Q1: 86 Q2: 35 Q3:93
At time 180 ns Q1: 92 Q2: 86 Q3:35

Info: (I804) /IEEE_Std_1666/deprecated: You can turn off warnings about
             IEEE 1666 deprecated features by placing this method call
             as the first statement in your sc_main() function:

  sc_core::sc_report_handler::set_actions( "/IEEE_Std_1666/deprecated",
                                           sc_core::SC_DO_NOTHING );
```

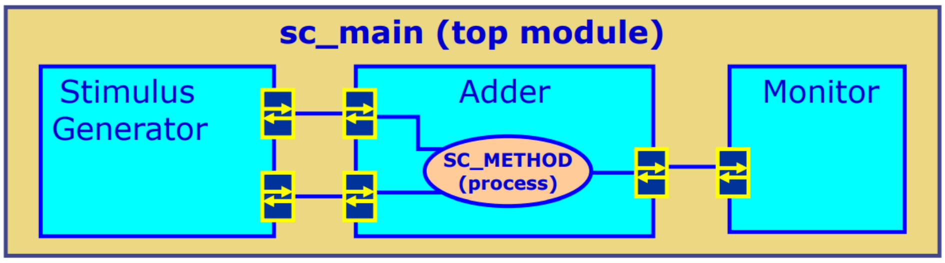
(2) Exercise 2: Producer and Consumer

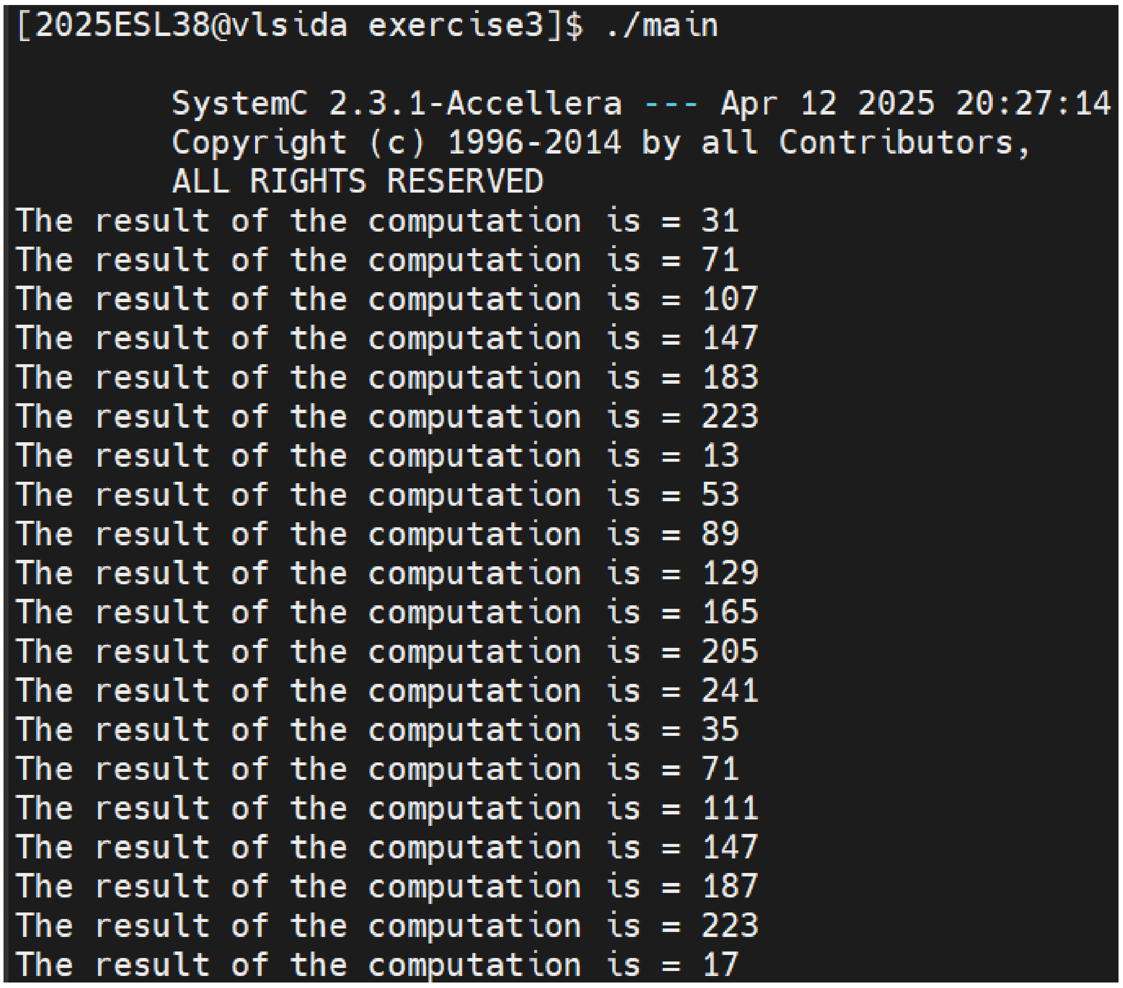




```
[2025ESL38@vlsida exercise2]$ ./producer_consumer

        SystemC 2.3.1-Accellera --- Apr 12 2025 20:27:14
        Copyright (c) 1996-2014 by all Contributors,
        ALL RIGHTS RESERVED
At time20 nsproduces data83
At time30 nsconsume data83
At time40 nsproduces data86
At time60 nsproduces data77
At time60 nsconsume data86
At time80 nsproduces data15
At time90 nsconsume data77
At time100 nsproduces data93
At time120 nsproduces data35
At time120 nsconsume data15
At time140 nsproduces data86
At time150 nsconsume data93
At time160 nsproduces data92
At time180 nsproduces data49
At time180 nsconsume data35
[2025ESL38@vlsida exercise2]$
```

(3) Exercise 3: Test in SystemC





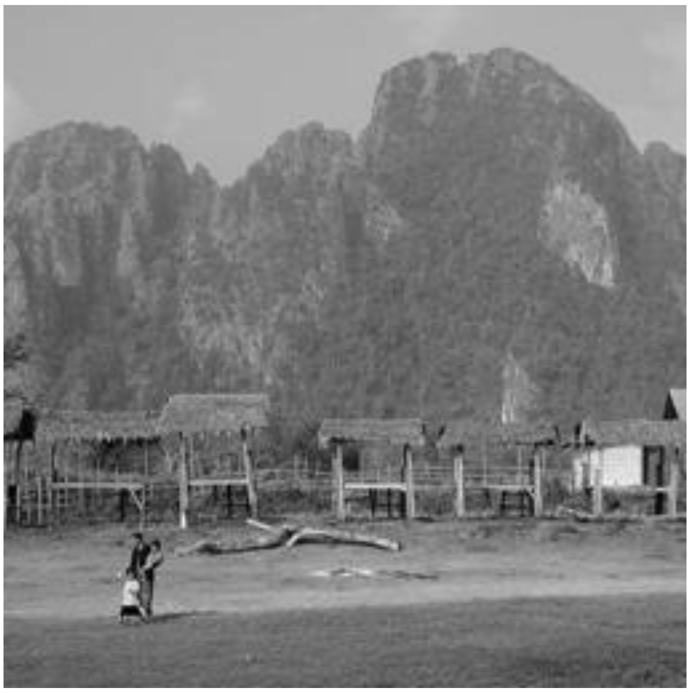
```
[2025ESL38@vlsida exercise3]$ ./main

        SystemC 2.3.1-Accellera --- Apr 12 2025 20:27:14
        Copyright (c) 1996-2014 by all Contributors,
        ALL RIGHTS RESERVED
The result of the computation is = 31
The result of the computation is = 71
The result of the computation is = 107
The result of the computation is = 147
The result of the computation is = 183
The result of the computation is = 223
The result of the computation is = 13
The result of the computation is = 53
The result of the computation is = 89
The result of the computation is = 129
The result of the computation is = 165
The result of the computation is = 205
The result of the computation is = 241
The result of the computation is = 35
The result of the computation is = 71
The result of the computation is = 111
The result of the computation is = 147
The result of the computation is = 187
The result of the computation is = 223
The result of the computation is = 17
```

（二）Exercise 4: RGB to YUV

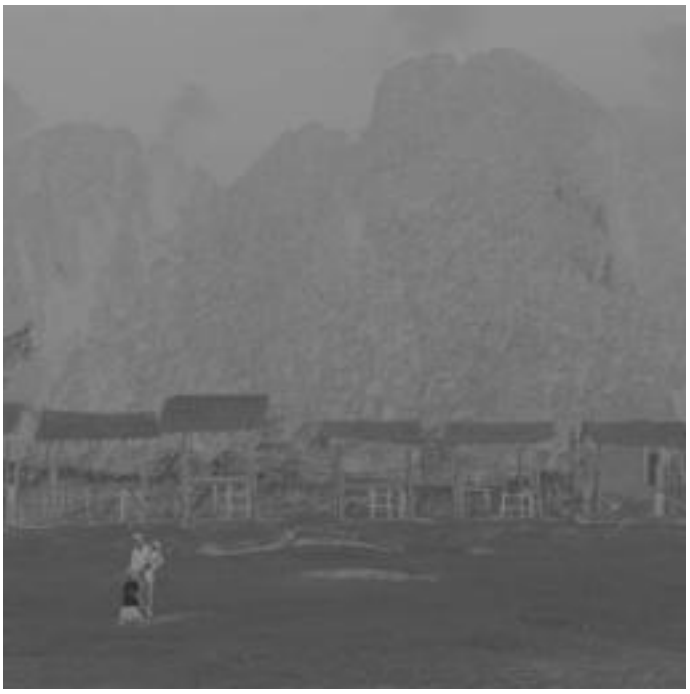
(1) 圖片

1. 原圖

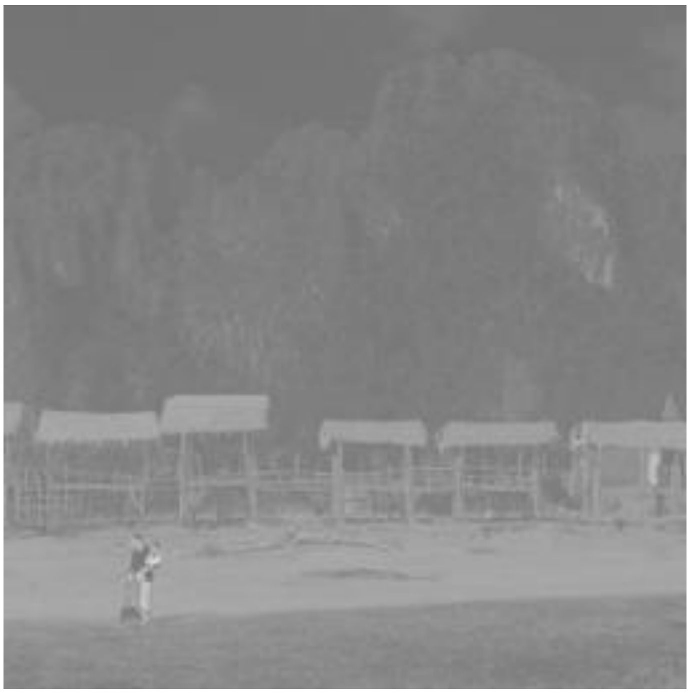
2.Y



3. U



4.V



(2) Y value 誤差表格

RGB2YUV	Maximum Absolute Error	Average Absolute Error
Version 1 vs. Version 2	1	0.00
Version 1 vs. Version 3	18	3.63
Version 1 vs. Version 4	18	3.60

p.s. Version 1 vs. Version 2 的 Average Absolute Error 是 0.00 的原因是 Sum of Absolute Error 除以分母(256*256)後，小數後兩位皆 0，並不是代表無誤差存在。

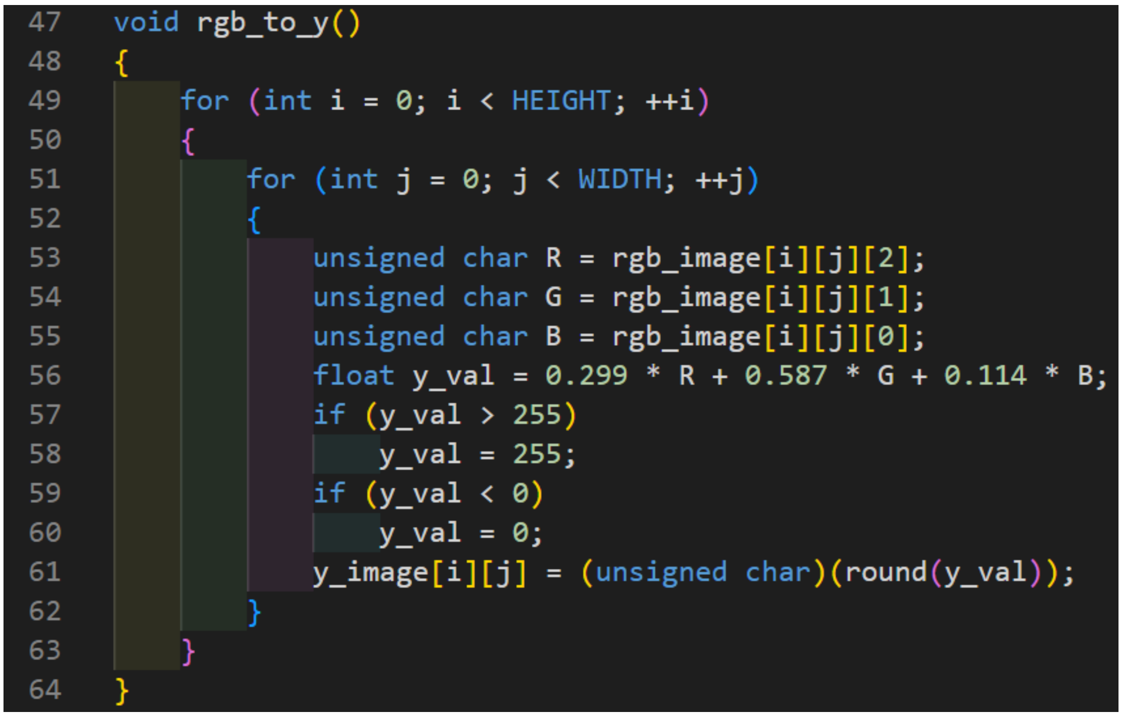
(3) 作法

A. Version 1

語言：C

RGB to YUV 公式 (Formula 1) ：

$$\begin{aligned} Y &= \quad 0.299 \times R + 0.587 \times G + 0.114 \times B \\ U &= -0.169 \times R - 0.331 \times G + 0.5 \times B + 128 \\ V &= \quad 0.5 \times R - 0.419 \times G - 0.081 \times B + 128 \end{aligned}$$


```c
void rgb_to_y()
{
    for (int i = 0; i < HEIGHT; ++i)
    {
        for (int j = 0; j < WIDTH; ++j)
        {
            unsigned char R = rgb_image[i][j][2];
            unsigned char G = rgb_image[i][j][1];
            unsigned char B = rgb_image[i][j][0];
            float y_val = 0.299 * R + 0.587 * G + 0.114 * B;
            if (y_val > 255)
                y_val = 255;
            if (y_val < 0)
                y_val = 0;
            y_image[i][j] = (unsigned char)(round(y_val));
        }
    }
}
```

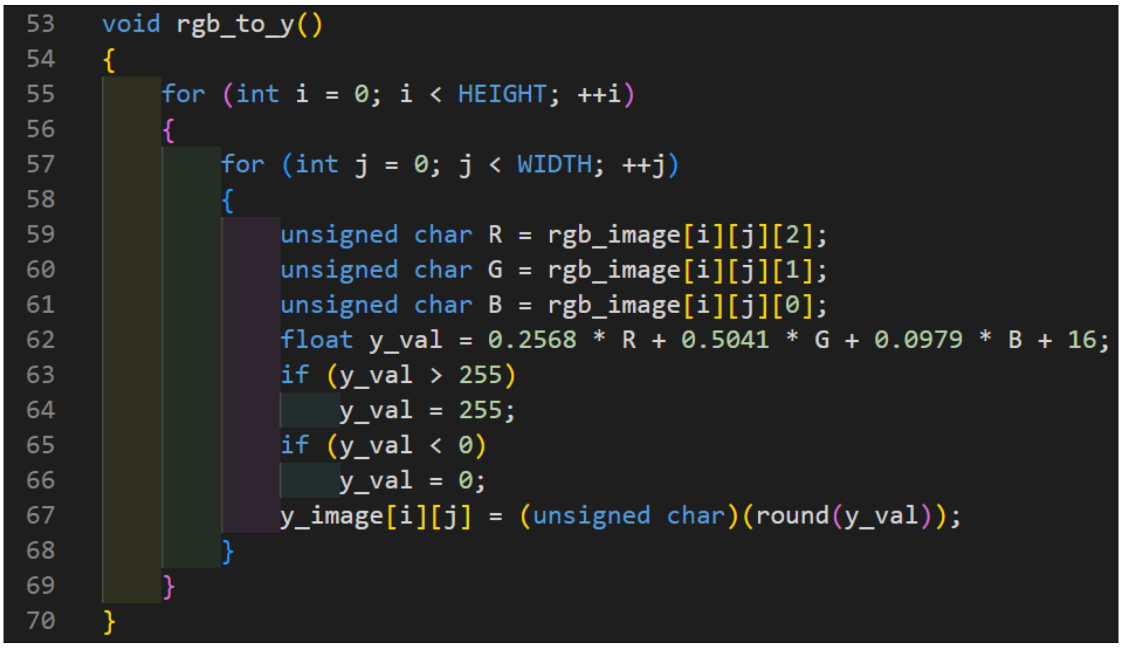
Y 值設有最大(255)與最小(0)值。

B. Version 2

語言：C

RGB to YUV 公式 (Formula 2) ：

$$\begin{aligned} Y &= 0.2568 \times R + 0.5041 \times G + 0.0979 \times B + 16 \\ U &= -0.1482 \times R - 0.2910 \times G + 0.4392 \times B + 128 \\ V &= 0.4392 \times R - 0.3678 \times G - 0.0714 \times B + 128 \end{aligned}$$



```c
void rgb_to_y()
{
    for (int i = 0; i < HEIGHT; ++i)
    {
        for (int j = 0; j < WIDTH; ++j)
        {
            unsigned char R = rgb_image[i][j][2];
            unsigned char G = rgb_image[i][j][1];
            unsigned char B = rgb_image[i][j][0];
            float y_val = 0.2568 * R + 0.5041 * G + 0.0979 * B + 16;
            if (y_val > 255)
                y_val = 255;
            if (y_val < 0)
                y_val = 0;
            y_image[i][j] = (unsigned char)(round(y_val));
        }
    }
}
```

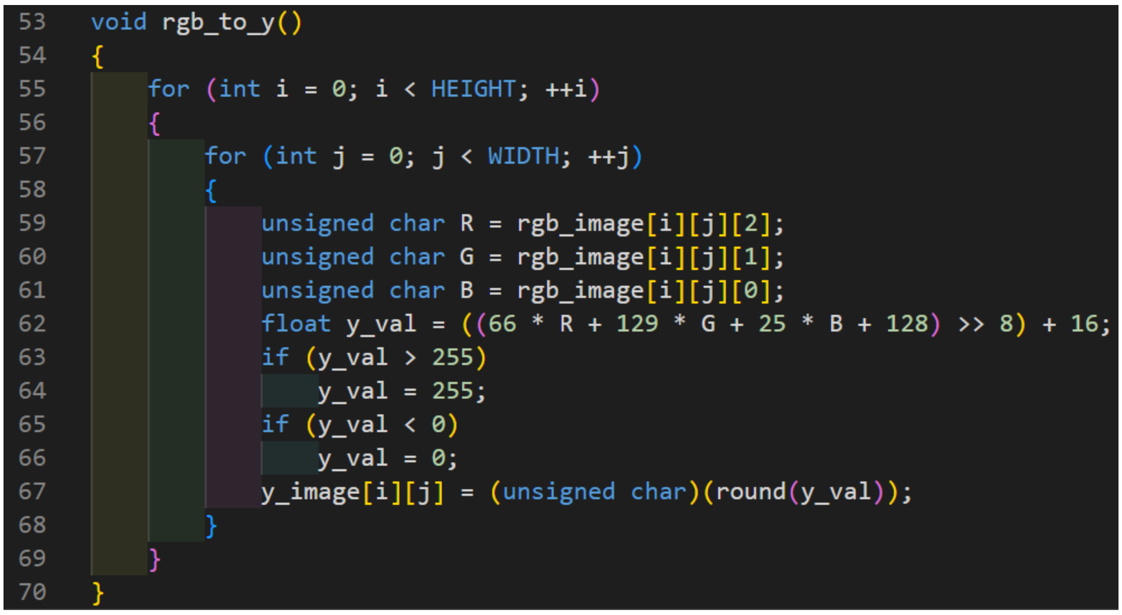
Y 值設有最大(255)與最小(0)值。

C. Version 3

語言：C

RGB to YUV 公式 (Formula 2_2) ：

$$\begin{aligned} Y &= (66 \times R + 129 \times G + 25 \times B + 128) \gg 8) + 16 \\ U &= (-38 \times R - 74 \times G + 112 \times B + 128) \gg 8) + 128 \\ V &= (112 \times R - 94 \times G - 18 \times B + 128) \gg 8) + 128 \end{aligned}$$



```
void rgb_to_y()
{
    for (int i = 0; i < HEIGHT; ++i)
    {
        for (int j = 0; j < WIDTH; ++j)
        {
            unsigned char R = rgb_image[i][j][2];
            unsigned char G = rgb_image[i][j][1];
            unsigned char B = rgb_image[i][j][0];
            float y_val = ((66 * R + 129 * G + 25 * B + 128) >> 8) + 16;
            if (y_val > 255)
                y_val = 255;
            if (y_val < 0)
                y_val = 0;
            y_image[i][j] = (unsigned char)(round(y_val));
        }
    }
}
```

Y 值設有最大(255)與最小(0)值。

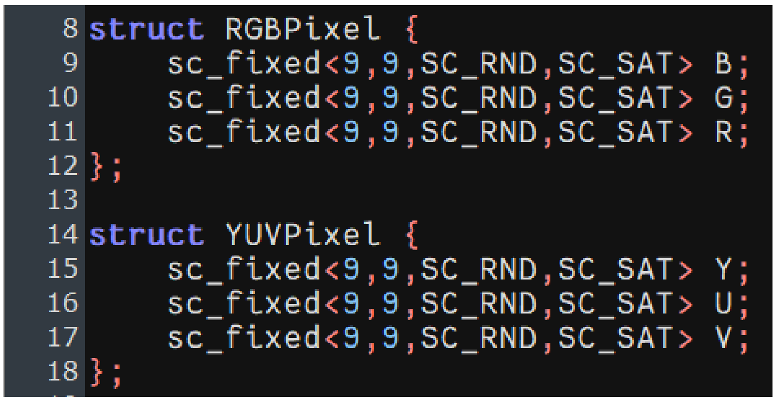
D. Version 4

語言：SystemC

RGB to YUV 公式 (Formula 1)：

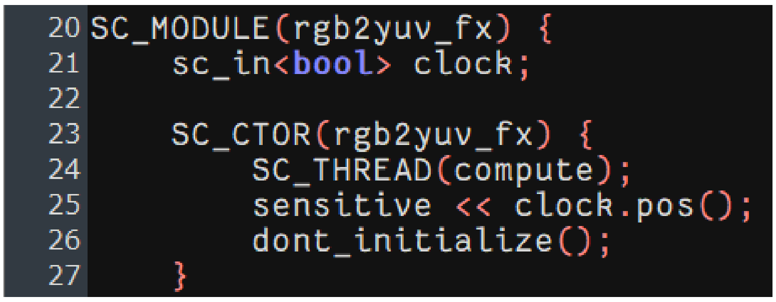
$$\begin{aligned} Y &= \quad 0.299 \times R + 0.587 \times G + 0.114 \times B \\ U &= -0.169 \times R - 0.331 \times G + 0.5 \times B + 128 \\ V &= \quad 0.5 \times R - 0.419 \times G - 0.081 \times B + 128 \end{aligned}$$

Data type：sc_fixed



```
struct RGBPixel {
    sc_fixed<9,9,SC_RND,SC_SAT> B;
    sc_fixed<9,9,SC_RND,SC_SAT> G;
    sc_fixed<9,9,SC_RND,SC_SAT> R;
};

struct YUVPixel {
    sc_fixed<9,9,SC_RND,SC_SAT> Y;
    sc_fixed<9,9,SC_RND,SC_SAT> U;
    sc_fixed<9,9,SC_RND,SC_SAT> V;
};
```

定點數表示法 sc_fixed<wl,iwl,q_mode,o_mode>，sc_fixed 的參數是靜態的，對照程式，RGB 與 YUV 整數部分有 9 位元，沒有小數部分，SC_RND 的意思是將捨去的 MSB 加到保留位元的 LSB。SC_SAT 因為有 9 位元，所以確保最大值是+255，最小值是−256。

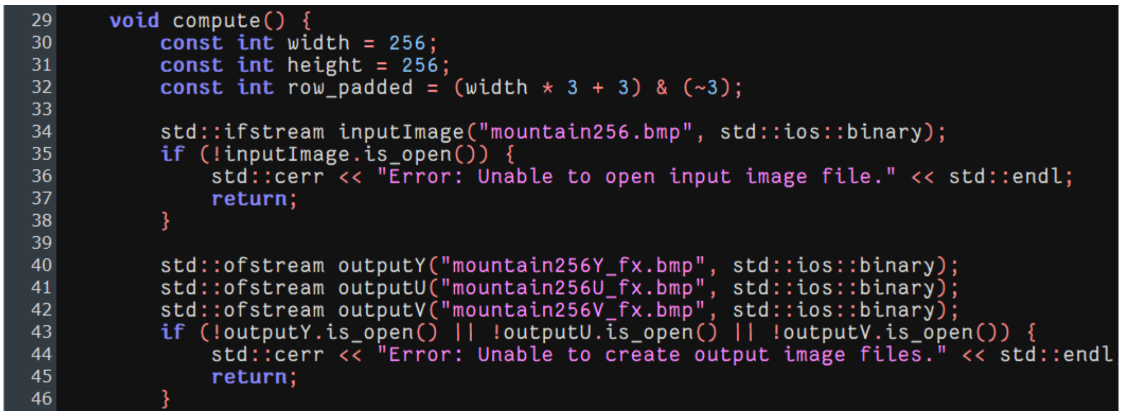


```
SC_MODULE(rgb2yuv_fx) {
    sc_in<bool> clock;

    SC_CTOR(rgb2yuv_fx) {
        SC_THREAD(compute);
        sensitive << clock.pos();
        dont_initialize();
    }
```

宣告一組元件定義 rgb2yuv_fx、輸入訊號 clock。

整組元件具有 process(SC_THREAD)，執行 compute。

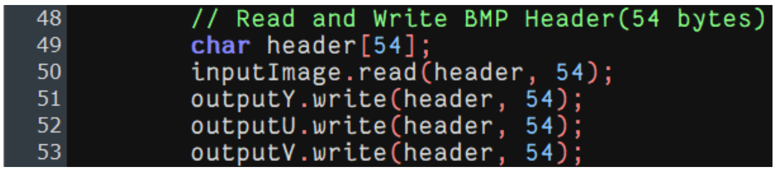
Process 會經由 clock 正緣觸發，且不要初始化。



```
    void compute() {
        const int width = 256;
        const int height = 256;
        const int row_padded = (width * 3 + 3) & (~3);

        std::ifstream inputImage("mountain256.bmp", std::ios::binary);
        if (!inputImage.is_open()) {
            std::cerr << "Error: Unable to open input image file." << std::endl;
            return;
        }

        std::ofstream outputY("mountain256Y_fx.bmp", std::ios::binary);
        std::ofstream outputU("mountain256U_fx.bmp", std::ios::binary);
        std::ofstream outputV("mountain256V_fx.bmp", std::ios::binary);
        if (!outputY.is_open() || !outputU.is_open() || !outputV.is_open()) {
            std::cerr << "Error: Unable to create output image files." << std::endl
            return;
        }
```

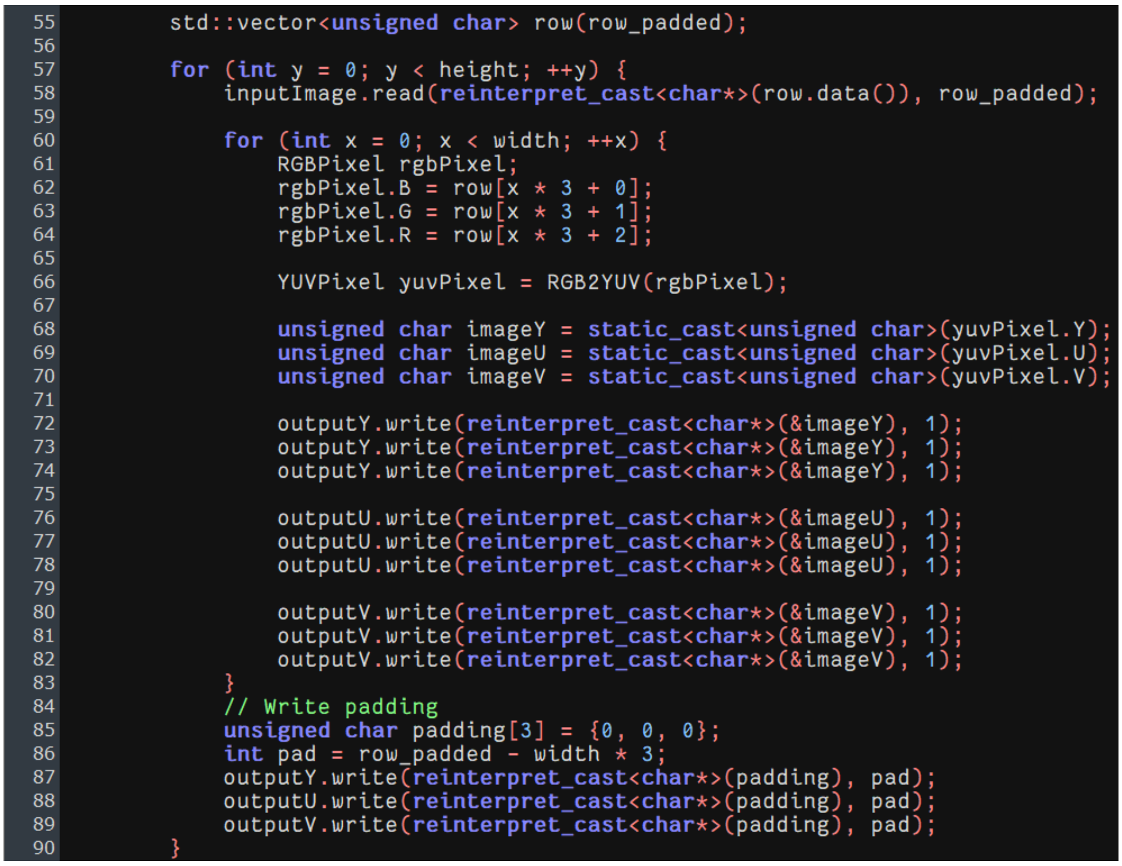
圖片長寬皆為 256。每列皆 4 byte。

讀取 RGB 原圖，創建 YUV 三張圖檔。

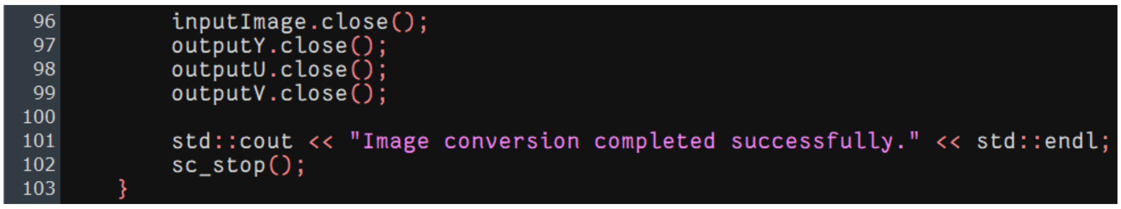


```
        // Read and Write BMP Header(54 bytes)
        char header[54];
        inputImage.read(header, 54);
        outputY.write(header, 54);
        outputU.write(header, 54);
        outputV.write(header, 54);
```

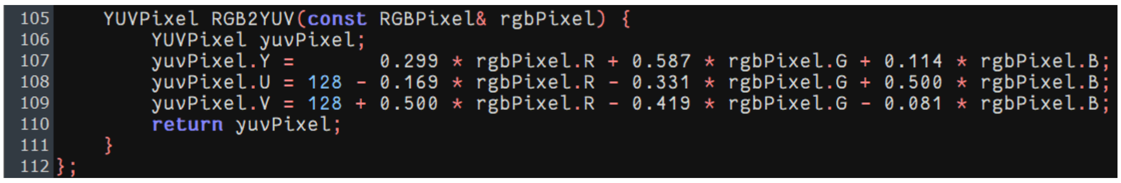
.bmp 檔案格式前 54 byte 是標頭，標示圖片資訊，設定輸出圖檔與原圖檔相同。


```
        std::vector<unsigned char> row(row_padded);

        for (int y = 0; y < height; ++y) {
            inputImage.read(reinterpret_cast<char*>(row.data()), row_padded);

            for (int x = 0; x < width; ++x) {
                RGBPixel rgbPixel;
                rgbPixel.B = row[x * 3 + 0];
                rgbPixel.G = row[x * 3 + 1];
                rgbPixel.R = row[x * 3 + 2];

                YUVPixel yuvPixel = RGB2YUV(rgbPixel);

                unsigned char imageY = static_cast<unsigned char>(yuvPixel.Y);
                unsigned char imageU = static_cast<unsigned char>(yuvPixel.U);
                unsigned char imageV = static_cast<unsigned char>(yuvPixel.V);

                outputY.write(reinterpret_cast<char*>(&imageY), 1);
                outputY.write(reinterpret_cast<char*>(&imageY), 1);
                outputY.write(reinterpret_cast<char*>(&imageY), 1);

                outputU.write(reinterpret_cast<char*>(&imageU), 1);
                outputU.write(reinterpret_cast<char*>(&imageU), 1);
                outputU.write(reinterpret_cast<char*>(&imageU), 1);

                outputV.write(reinterpret_cast<char*>(&imageV), 1);
                outputV.write(reinterpret_cast<char*>(&imageV), 1);
                outputV.write(reinterpret_cast<char*>(&imageV), 1);
            }
            // Write padding
            unsigned char padding[3] = {0, 0, 0};
            int pad = row_padded - width * 3;
            outputY.write(reinterpret_cast<char*>(padding), pad);
            outputU.write(reinterpret_cast<char*>(padding), pad);
            outputV.write(reinterpret_cast<char*>(padding), pad);
        }
```

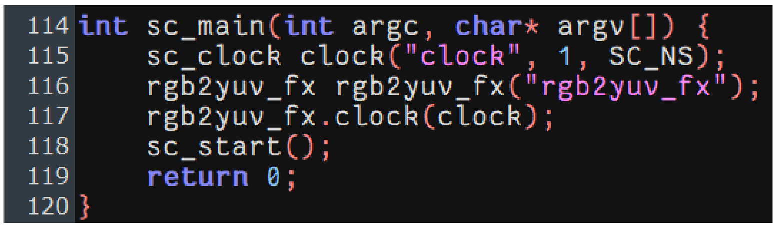
由下而上、由左而右將原圖每個 Pixel 的 B、G、R 依序分開，經由 RGB2YUV 的函式得到 Y、U、V，轉成 char 後分別輸出至各自的圖檔。


```
        inputImage.close();
        outputY.close();
        outputU.close();
        outputV.close();

        std::cout << "Image conversion completed successfully." << std::endl;
        sc_stop();
    }
```

關閉輸入與輸出圖，確認轉變 YUV 成功後即停止模擬。


```
    YUVPixel RGB2YUV(const RGBPixel& rgbPixel) {
        YUVPixel yuvPixel;
        yuvPixel.Y =       0.299 * rgbPixel.R + 0.587 * rgbPixel.G + 0.114 * rgbPixel.B;
        yuvPixel.U = 128 - 0.169 * rgbPixel.R - 0.331 * rgbPixel.G + 0.500 * rgbPixel.B;
        yuvPixel.V = 128 + 0.500 * rgbPixel.R - 0.419 * rgbPixel.G - 0.081 * rgbPixel.B;
        return yuvPixel;
    }
};
```

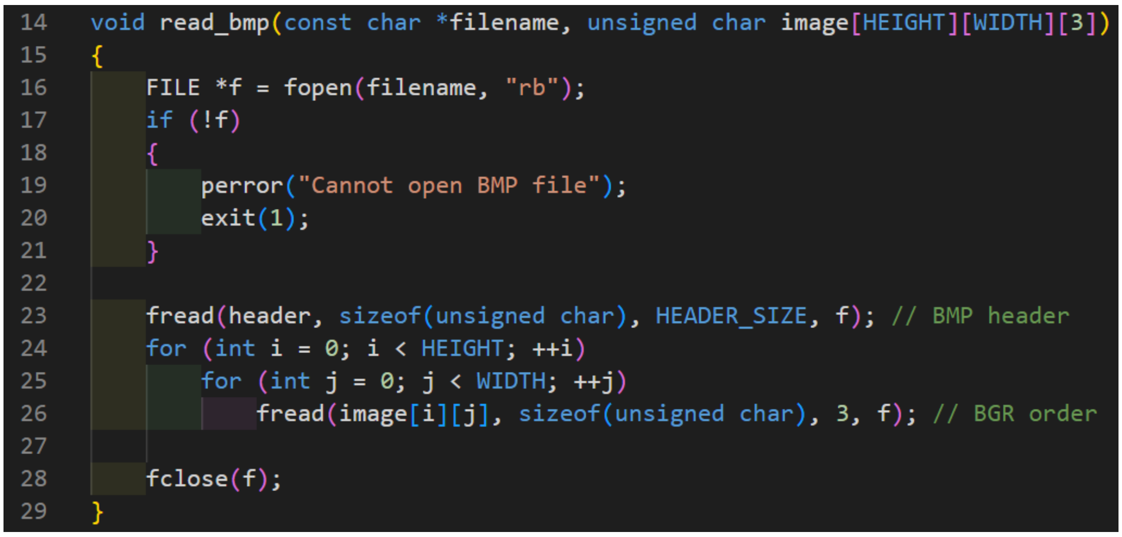
RGB2YUV 函式使用講義上提供的 Formula 1。



```
int sc_main(int argc, char* argv[]) {
    sc_clock clock("clock", 1, SC_NS);
    rgb2yuv_fx rgb2yuv_fx("rgb2yuv_fx");
    rgb2yuv_fx.clock(clock);
    sc_start();
    return 0;
}
```

主函式，將 module 接上周期 1 ns 的 clock，並開始模擬。

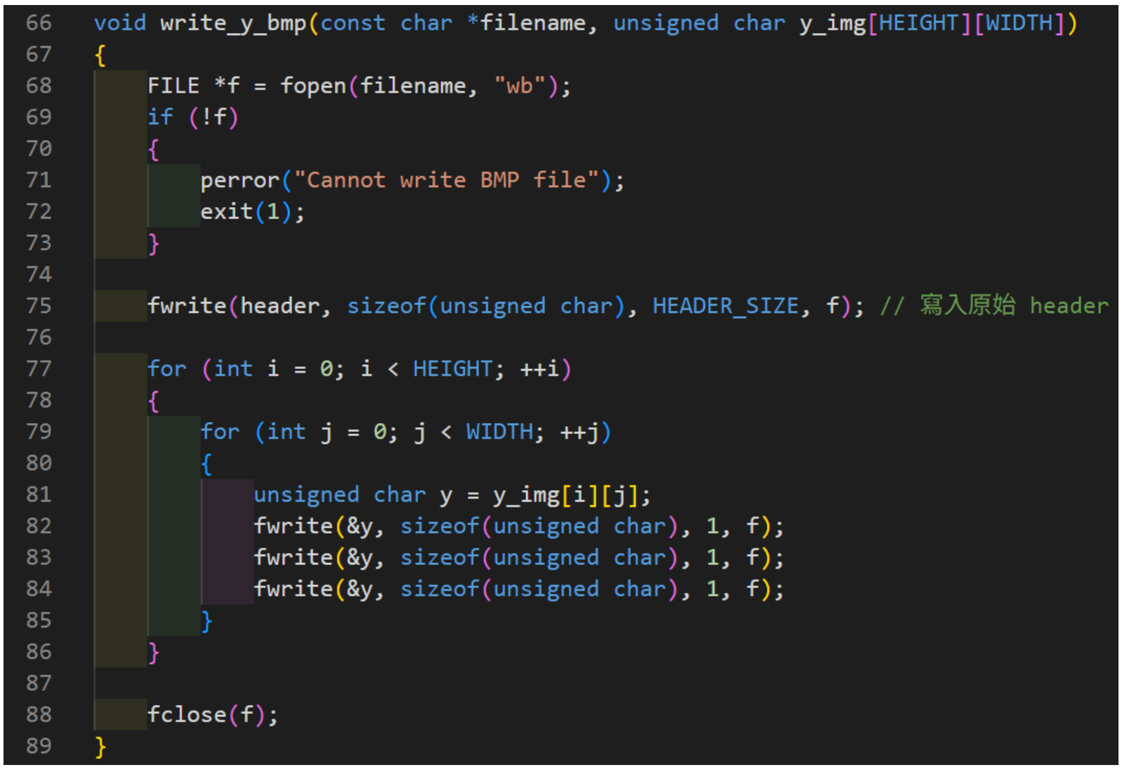
E. 比較誤差方法



```
void read_bmp(const char *filename, unsigned char image[HEIGHT][WIDTH][3])
{
    FILE *f = fopen(filename, "rb");
    if (!f)
    {
        perror("Cannot open BMP file");
        exit(1);
    }

    fread(header, sizeof(unsigned char), HEADER_SIZE, f); // BMP header
    for (int i = 0; i < HEIGHT; ++i)
        for (int j = 0; j < WIDTH; ++j)
            fread(image[i][j], sizeof(unsigned char), 3, f); // BGR order

    fclose(f);
}
```

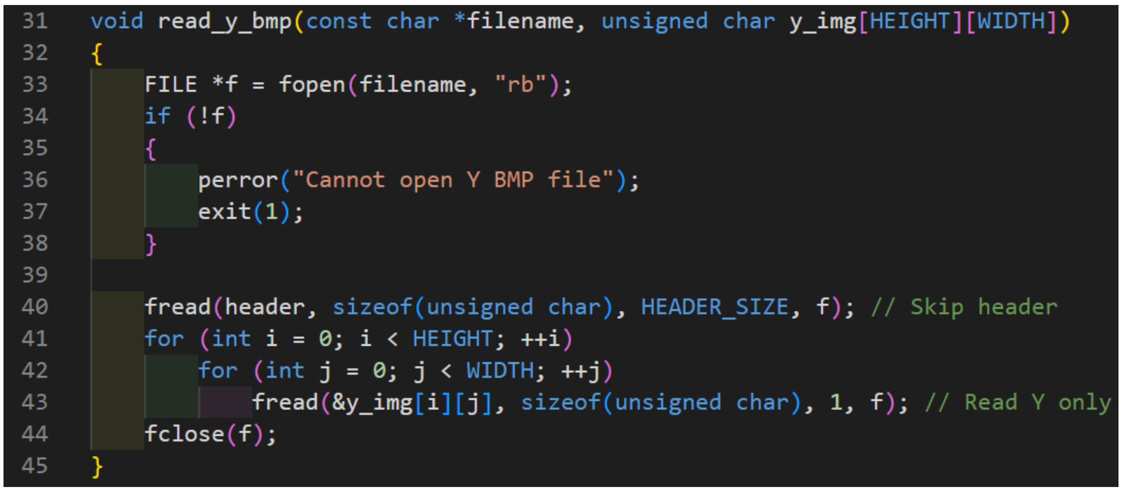
首先讀取原圖檔。

再分別經由上述 A.~C.的函式轉變成 YUV。



```c
void write_y_bmp(const char *filename, unsigned char y_img[HEIGHT][WIDTH])
{
    FILE *f = fopen(filename, "wb");
    if (!f)
    {
        perror("Cannot write BMP file");
        exit(1);
    }

    fwrite(header, sizeof(unsigned char), HEADER_SIZE, f); // 寫入原始 header

    for (int i = 0; i < HEIGHT; ++i)
    {
        for (int j = 0; j < WIDTH; ++j)
        {
            unsigned char y = y_img[i][j];
            fwrite(&y, sizeof(unsigned char), 1, f);
            fwrite(&y, sizeof(unsigned char), 1, f);
            fwrite(&y, sizeof(unsigned char), 1, f);
        }
    }

    fclose(f);
}
```

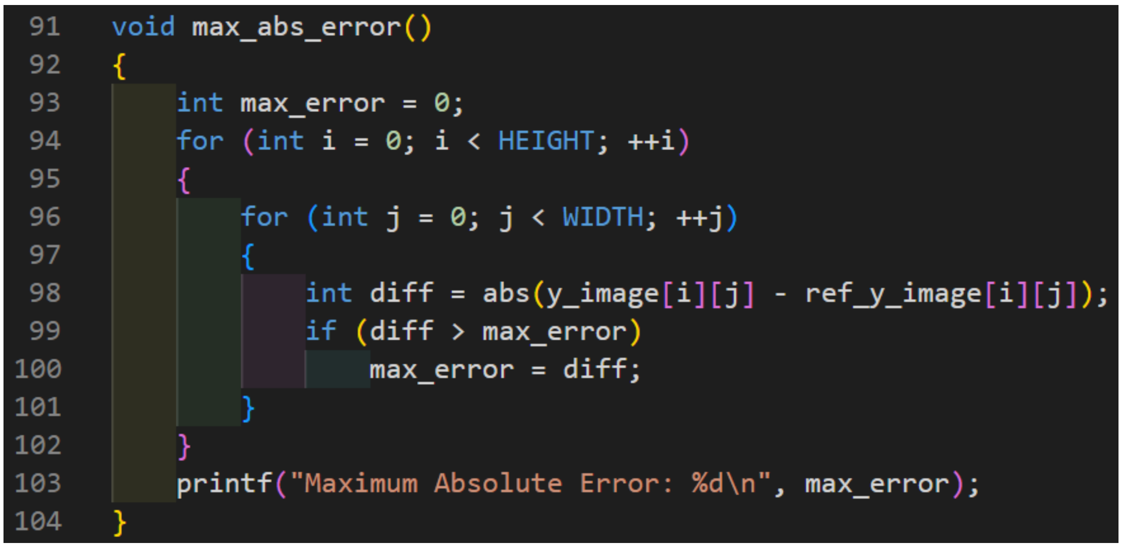
將 Y 寫入新的.bmp 檔。



```c
void read_y_bmp(const char *filename, unsigned char y_img[HEIGHT][WIDTH])
{
    FILE *f = fopen(filename, "rb");
    if (!f)
    {
        perror("Cannot open Y BMP file");
        exit(1);
    }

    fread(header, sizeof(unsigned char), HEADER_SIZE, f); // Skip header
    for (int i = 0; i < HEIGHT; ++i)
        for (int j = 0; j < WIDTH; ++j)
            fread(&y_img[i][j], sizeof(unsigned char), 1, f); // Read Y only
    fclose(f);
}
```

這個函式可以讀取 version 1~ version 3 的 Y 圖與 version 4 輸出的 Y 圖。

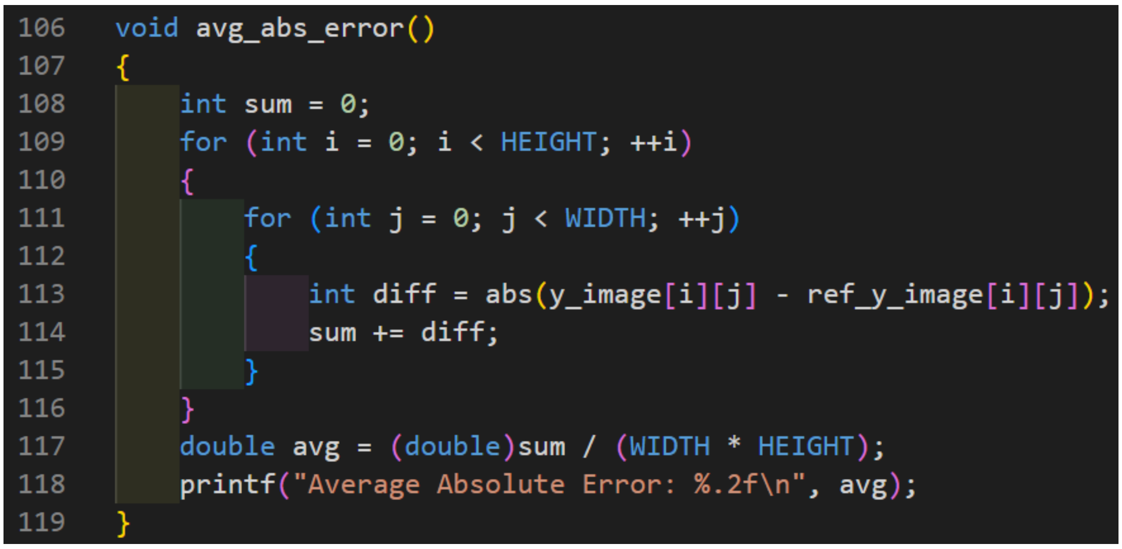
F. 最大絕對誤差



```c
void max_abs_error()
{
    int max_error = 0;
    for (int i = 0; i < HEIGHT; ++i)
    {
        for (int j = 0; j < WIDTH; ++j)
        {
            int diff = abs(y_image[i][j] - ref_y_image[i][j]);
            if (diff > max_error)
                max_error = diff;
        }
    }
    printf("Maximum Absolute Error: %d\n", max_error);
}
```

設最大絕對誤差初始值設為 0，再用迴圈跑每個 Pixel，計算 version 4 與其他 version 每個 Pixel 絕對誤差(diff)，如果該絕對誤差(diff)大於當前最大絕對誤差值(max_error)，則更新，最終的 max_error 即為答案。

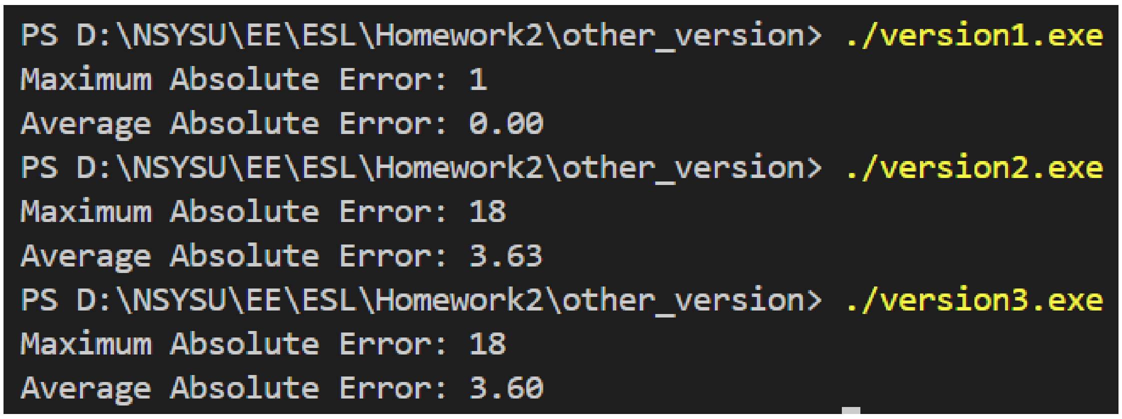
G. 平均絕對誤差



```c
void avg_abs_error()
{
    int sum = 0;
    for (int i = 0; i < HEIGHT; ++i)
    {
        for (int j = 0; j < WIDTH; ++j)
        {
            int diff = abs(y_image[i][j] - ref_y_image[i][j]);
            sum += diff;
        }
    }
    double avg = (double)sum / (WIDTH * HEIGHT);
    printf("Average Absolute Error: %.2f\n", avg);
}
```

設絕對誤差加總(sum)初始值為 0，再用迴圈跑每個 Pixel，計算 version 4 與其他 version 每個 Pixel 絕對誤差，並累加在 sum 上，最終將 sum 除以(長*寬)即為平均絕對誤差。

H. 結果



```
PS D:\NSYSU\EE\ESL\Homework2\other_version> ./version1.exe
Maximum Absolute Error: 1
Average Absolute Error: 0.00
PS D:\NSYSU\EE\ESL\Homework2\other_version> ./version2.exe
Maximum Absolute Error: 18
Average Absolute Error: 3.63
PS D:\NSYSU\EE\ESL\Homework2\other_version> ./version3.exe
Maximum Absolute Error: 18
Average Absolute Error: 3.60
```

（三）實驗心得

以前對硬體描述語言的認知只有 VHDL、Verilog，現在我多學會了 SystemC，是用一種比較像寫軟體的方式描述電路，幾乎都可以與我比較熟悉的 Verilog 對得起來，只有語法不太一樣，不過我覺得 SystemC 會需要有資料結構的觀念，我沒有學過，所以一開始的時候不知從何下手，是看著講義上的範例參考，一步一步兜起來的，花了一些時間，不過也透過這個機會對 C++更加熟悉一點，像是指標、迴圈等等。

而因為此次實驗用的是.bmp，所以要先了解這種檔案的形式，像是前 54 byte 是標頭檔，RGB 調色盤的順序是 B、G、R，而每一個 row 都要是 4 byte，如果忽略了這個特性，輸出的圖檔可能就會看起來怪怪的，即使 RGB 轉 YUV 的公式沒錯。

RGB 轉 YUV 的公式簡直是五花八門，但是條條大路通羅馬，都可以轉成亮度與色度的圖，雖然用肉眼看起轉出的圖都差不多，但實際上還是有差，像我們

做的 Exercise 4，就是用很多種公式，但是從輸出的結果圖比對就可以知道，每個像素的數值還是略有差異。

另外，SystemC 的 Simulation 因為是 event-accurate，所以應該沒辦法看波形去一個一個 cycle 檢查有沒有做錯，只能從 simulation result 檢查，不過有弊就有利，這也是為什麼 SystemC 可以模擬比較快的原因。