Hw1

112061612 蔡慶翰

1. Gaussian blur filter with FIFO channels

```
const int MASK_N = 1;
const int MASK_X = 5;
const int MASK_Y = 5;
```

```
// gaussianblur mask
const int mask[MASK_N][MASK_X][MASK_Y] = {{{1, 4, 7, 4, 1}, {4, 16, 26, 16, 4}, {7, 26, 41, 26, 7}, {4, 16, 26, 16, 4}, {1, 4, 7, 4, 1}}};
```

將 filter 大小以及各位置權重設好

```
double total = 0;
for (unsigned int i = 0; i != MASK_N; ++i)
{
  total += val[i];
}
int result = (int)(total / 273);
```

且 output 取平均要除以 273

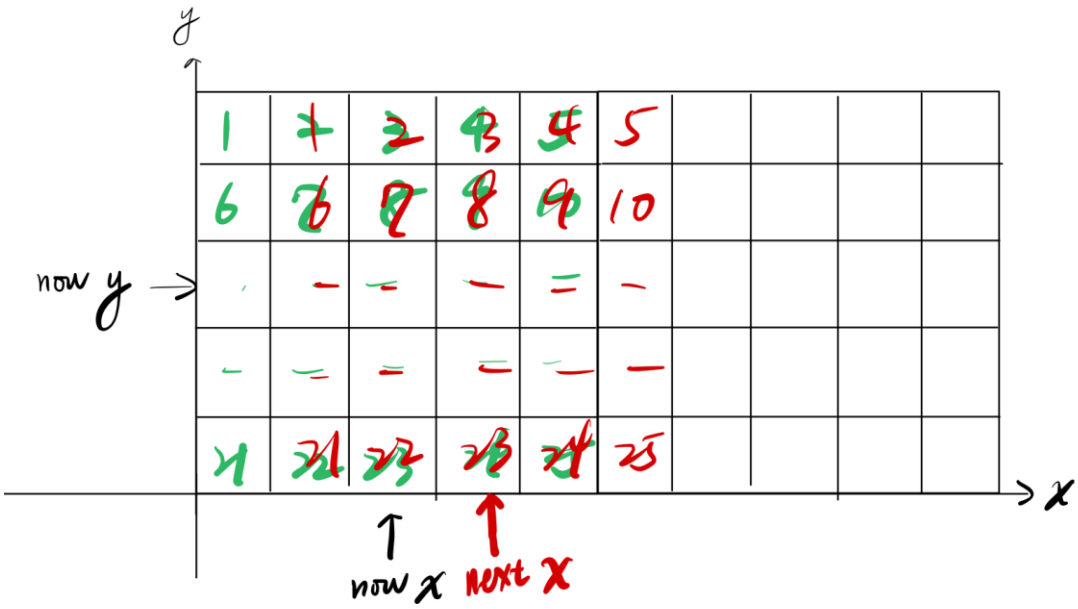
```
*(target_bitmap + bytes_per_pixel * (width * y + x - MASK_X) + 2) = total;
*(target_bitmap + bytes_per_pixel * (width * y + x - MASK_X) + 1) = total;
*(target_bitmap + bytes_per_pixel * (width * y + x - MASK_X) + 0) = total;
```

將原本判斷 total 是否超過 threshold 而定義 output 為黑或白的機制改為直接賦予 total。

2. Data buffers

原本在 initiator 端傳送 data 的機制如下：

```
for (y = 0; y != height; ++y)
{
  for (x = 0; x != width; ++x)
  {
    adjustX = (MASK_X % 2) ? 1 : 0; // 1
    adjustY = (MASK_Y % 2) ? 1 : 0; // 1
    xBound = MASK_X / 2;            // 1
    yBound = MASK_Y / 2;            // 1

    for (v = -yBound; v != yBound + adjustY; ++v)
    { //-1, 0, 1
      for (u = -xBound; u != xBound + adjustX; ++u)
      { //-1, 0, 1
        if (x + u >= 0 && x + u < width && y + v >= 0 && y + v < height)
        {
          R = *(source_bitmap +
                bytes_per_pixel * (width * (y + v) + (x + u)) + 2);
          G = *(source_bitmap +
                bytes_per_pixel * (width * (y + v) + (x + u)) + 1);
          B = *(source_bitmap +
                bytes_per_pixel * (width * (y + v) + (x + u)) + 0);
        }
        else
        {
          R = 0;
          G = 0;
          B = 0;
        }
        o_r.write(R);
        o_g.write(G);
        o_b.write(B);
        wait(1); // emulate channel delay
      }
    }

    if (i_result.num_available() == 0)
      wait(i_result.data_written_event());
    total = i_result.read();
    // cout << "Now at " << sc_time_stamp() << endl; //print current sc_time

    *(target_bitmap + bytes_per_pixel * (width * y + x) + 2) = total;
    *(target_bitmap + bytes_per_pixel * (width * y + x) + 1) = total;
    *(target_bitmap + bytes_per_pixel * (width * y + x) + 0) = total;
  }
}
```



傳送順序及重複傳送 data 如上圖所示，沒有 input data reuse，總共 transfer　512*512*(5*5)*3 Byte = 19660800 Byte。

(Byte per pixel = 3，因有 r、g、b 各 1byte)

```
        SystemC 2.3.3-Accellera --- Mar 17 2022 13:55:26
        Copyright (c) 1996-2018 by all Contributors,
        ALL RIGHTS RESERVED
Image width=512, height=512

Info: /OSCI/SystemC: Simulation stopped by user.
Simulated time == 6553600 ns
[100%] Built target run
```

Simulated time == 6553600ns

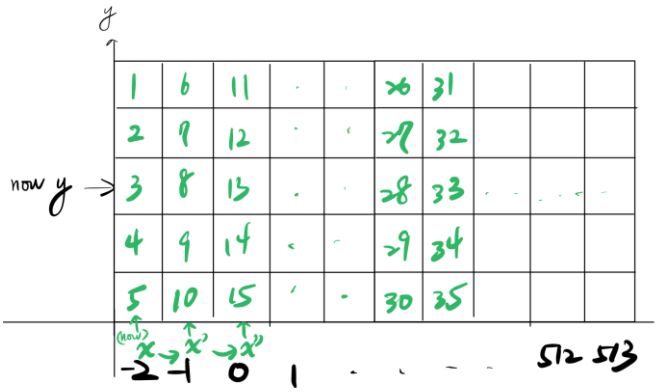
加上 input buffer(grey[5][5])在 GaussianblurFilter.cpp

```
unsigned char grey[5][5] = {0};
while (true)
{

  for (unsigned int x = 0; x < MASK_X; ++x)
  {
    grey[x][MASK_Y - 1] = (i_r.read() + i_g.read() + i_b.read()) / 3;
  }

  for (unsigned int i = 0; i < MASK_N; ++i)
  {
    val[i] = 0;
  }
  for (unsigned int v = 0; v < MASK_Y; ++v)
  {
    for (unsigned int u = 0; u < MASK_X; ++u)
    {
      for (unsigned int i = 0; i != MASK_N; ++i)
      {
        val[i] += grey[u][v] * mask[i][u][v];
        if (v > 0 && v < MASK_Y)
        {
          grey[u][v - 1] = grey[u][v];
        }
      }
    }
  }

  double total = 0;
  for (unsigned int i = 0; i != MASK_N; ++i)
  {
    total += val[i];
  }
  int result = (int)(total / 273);
  o_result.write(result);
  wait(10); // emulate module delay
}
```

做法是每次讀取新的 15 筆 input data，轉換成 5 筆新的灰階 data 後，依序存放於 grey[][]最右邊的 column，而每次做完運算的 grey[u][v]就往左 shift 一個 column，這麼做有一點須注意就是在 initiator 端每次 x=-2~2 輸入時，運算的 total 是不能用的，要等到 buffer 裡的 input data 都填滿新的才能使用，而針對這項問題的處理在接收 total 的 initiator 端解決。

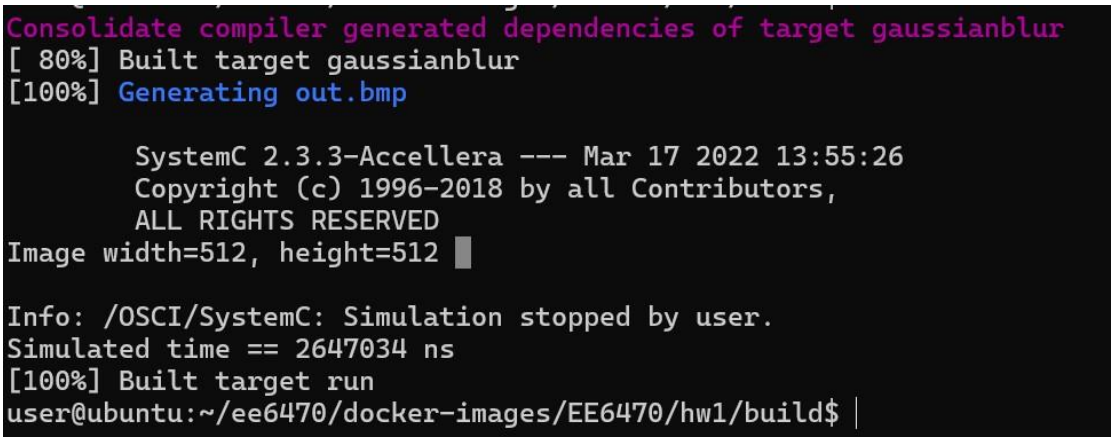
以下是更改為不傳送重複的 input data 的方法

```
for (y = 0; y != height; ++y)
{
  for (x = -2; x < width + 2; ++x)
  {
    // adjustX = (MASK_X % 2) ? 1 : 0; // 1
    adjustY = (MASK_Y % 2) ? 1 : 0; // 1
    // xBound = MASK_X / 2;            // 1
    yBound = MASK_Y / 2; // 1
    for (v = -yBound; v != yBound + adjustY; ++v)
    { //-1, 0, 1
      if (x >= 0 && x < width && y + v >= 0 && y + v < height)
      {
        R = *(source_bitmap +
              bytes_per_pixel * (width * (y + v) + (x)) + 2);
        G = *(source_bitmap +
              bytes_per_pixel * (width * (y + v) + (x)) + 1);
        B = *(source_bitmap +
              bytes_per_pixel * (width * (y + v) + (x)) + 0);
      }
      else
      {
        R = 0;
        G = 0;
        B = 0;
      }
      o_r.write(R);
      o_g.write(G);
      o_b.write(B);
      wait(1); // emulate channel delay
    }

    if (i_result.num_available() == 0)
      wait(i_result.data_written_event());
    total = i_result.read();
    // cout << "Now at " << sc_time_stamp() << endl; //print current sc_time
    if (x > 2)
    {
      *(target_bitmap + bytes_per_pixel * (width * y + x - 2) + 2) = total;
      *(target_bitmap + bytes_per_pixel * (width * y + x - 2) + 1) = total;
      *(target_bitmap + bytes_per_pixel * (width * y + x - 2) + 0) = total;
    }
  }
}
```



上圖是傳送 input data 的順序且不重複的示意圖，有 input data reuse，總共 transfer　512*(512+2+2)*(5)*3 Byte = 3962880 Byte。

與原本沒有 input buffer 的設計，少傳輸了 15697920 Byte



Simulated time == 2647034ns

Performance = 6553600ns/2647034ns = 2.4758，較原本設計快了近 2.5 倍，雖說 transfer input data 少了近 5 倍的量，但運算以及 output 的 transfer 都沒改變，因此只較原本設計快了近 2.5 倍，離最理想的 5 倍還有些距離。