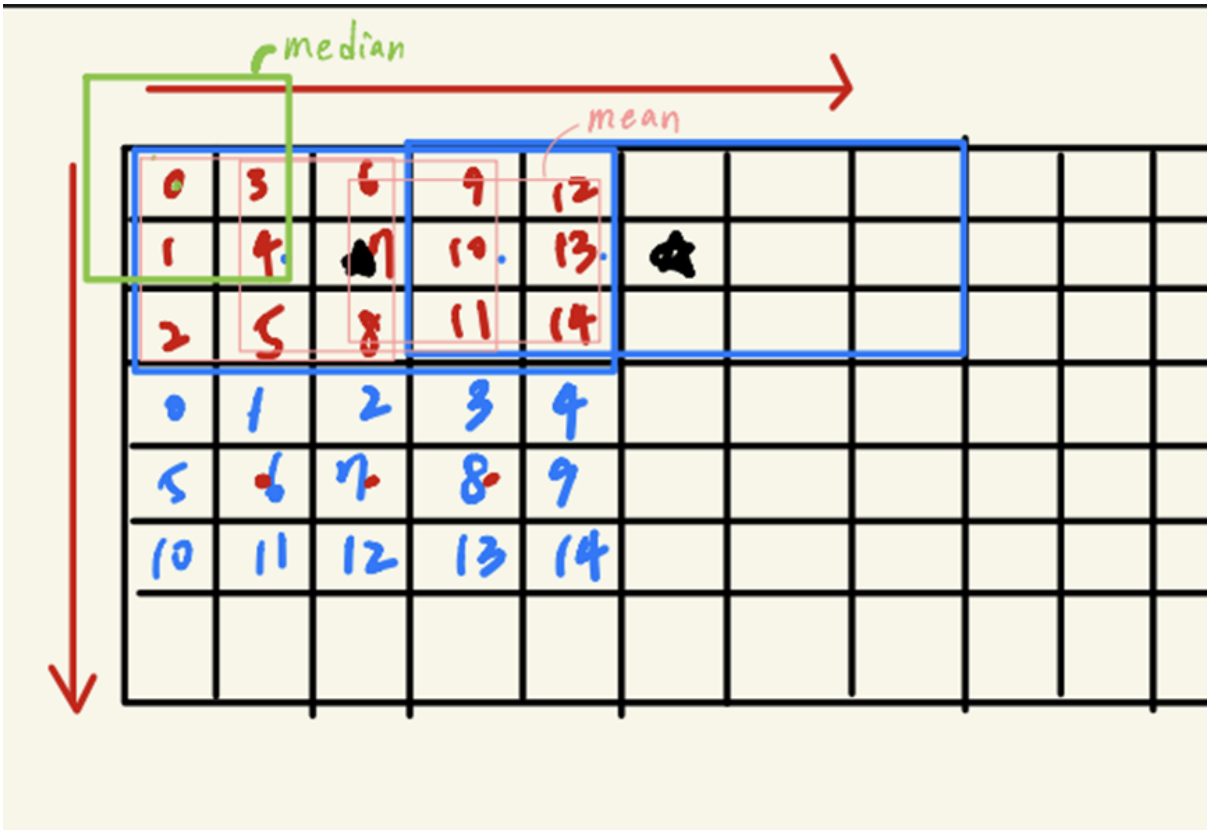
E111064503 通訊所 吳紹齊 Final project

https://github.com/cowboy35927/ESL/tree/main/Final%20project

演算法介紹:



圖四

上圖是示意圖綠色框是做median filter，粉色框是mean filter，粉色框移動順序是做完mean filter後向右移一個，藍色框是buffer儲存pixel的數量，一共存15個pixel，藍色框移動順序是中心點向右移3格，做完整排後，往下移一格。

因為systemC module 先做median filter再做mean filter，所以我設計需要先算出9個經過median filter所得出的median value pixel，再將這9個pixel加起來平均後，得到1個mean value pixel後，輸出到Testbench。但我至這邊與上一題的儲存順序不太一樣，buffer的index是由上到下後由左到右的順序去儲存，因讀入pixel是由左到右後由上到下，舉例來說:讀入pixel為

先讀第一行buffer index為:0、3、6、9、12

再讀第二行buffer index為:1、4、7、10、13

再讀第三行buffer index為:2、5、8、11、14

當讀到第三行的8時，可以看到第一個mean filter出現了可以算第一個mean pixel，可以直接讀buffer[0]~buffer[8]的mean pixel。

當讀到第三行的11時，可以看到第二個mean filter出現了可以算第二個mean pixel，可以直接讀buffer[3]~buffer[11]的mean pixel。

當讀到第三行的14時，可以看到第三個mean filter出現了可以算第三個mean pixel，可以直接讀buffer[6]~buffer[14]的mean pixel。

每做出一個mean pixel 就輸出一次，每輸出三個pixel需要讀135個pixel。

```
for (y = 0; y != height; ++y)
{
  int count = 0;
  for (x = 2; x < width; x = x + 3)
  {
    adjustX = 1; // 1
    adjustY = 1; // 1
    xBound = 1;  // 1
    yBound = 1;  // 1

    for (int count_y = 0; count_y < 3; count_y++)
    {
      for (int count_x = -2; count_x < 3; count_x++)
      {
        for (v = -yBound; v != yBound + adjustY; ++v)
        { //-1, 0, 1
          for (u = -xBound; u != xBound + adjustX; ++u)
          { //-1, 0, 1
            if (x + count_x + u >= 0 && x + count_x + u < width && y + count_y + v >= 0 && y + count_y + v < height)
            {
              R = *(source_bitmap +
                    bytes_per_pixel * (width * (y + count_y + v) + (x + count_x + u)) + 2);
              G = *(source_bitmap +
                    bytes_per_pixel * (width * (y + count_y + v) + (x + count_x + u)) + 1);
              B = *(source_bitmap +
                    bytes_per_pixel * (width * (y + count_y + v) + (x + count_x + u)) + 0);
            }
            else
            {
              R = 0;
              G = 0;
              B = 0;
```

圖五(Testbench)

可以看到上面的迴圈最外面的兩層是控制藍色框的mean filter的移動順序，X方向是移動3格，Y是一格。

中間兩個迴圈是控制藍色框裡的pixel點，順序如圖四粉紅色數字的順序。

最內層是控制綠色框的median filter的順序。

中間四個迴圈總共會寫入135個pixel得到三個mean pixel

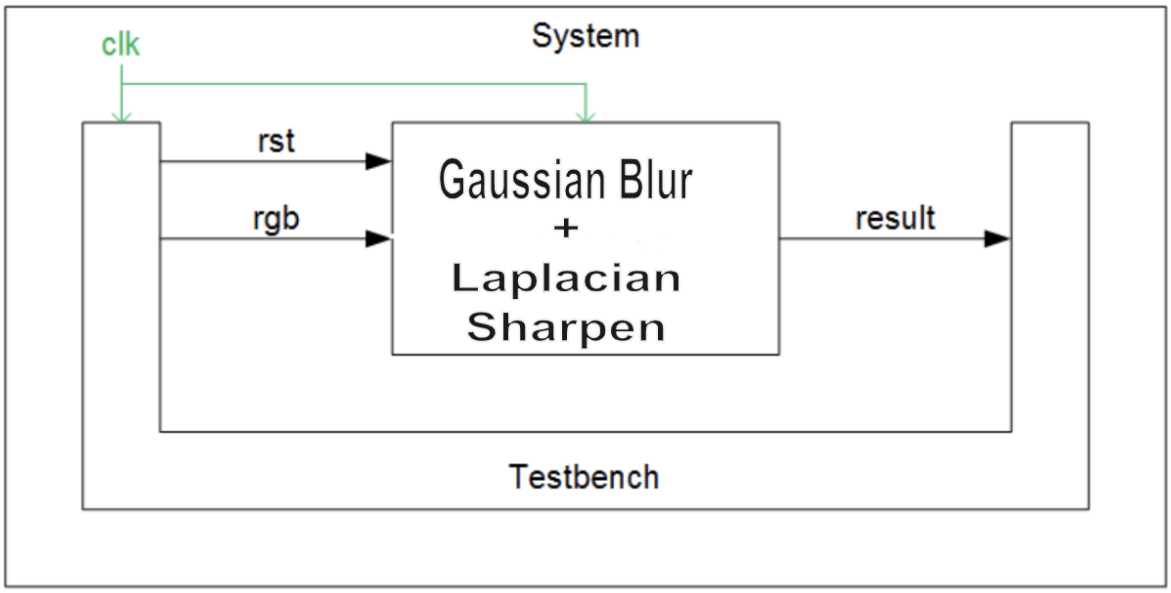
```
vector<vector<unsigned char>> buffer(3, vector<unsigned char>(15, 0));
```

上式是我所加的buffer，一共可以存15個pixel。

1.(50 pt) Implement HLS accelerator PEs**

1. Please choose and implement a SystemC-based HLS accelerator PE.
2. Please also implement test bench from the corresponding software to validate the function and timing of a PE.
3. The SystemC PEs should be synthesized with Stratus.
    1. PE codes should be simplified for math expressions and bit width.
    2. PE should be optimized with loop pipelining, unrolling, etc.
    3. Please compare and research the area and performance of different accelerator PE versions. The focus should be placed on the tradeoff of micro-architecture designs vs. area/performance.

HLS structure:



這裡是High level systhesis的架構圖，我這裡採用的是先經過Gassian Blur，進行數據平滑處理，可以有效的將噪聲去除和平滑邊界，再經過Laplacian sharpen去進行銳化處理，可以使圖片看起來更清晰。

HLS-Cynw:

```
#include <cynw_p2p.h>
#include <cynw_fixed.h>
```

```
private:
  void do_filter();
  cynw_ufixed<24,12> val[3][9];
```

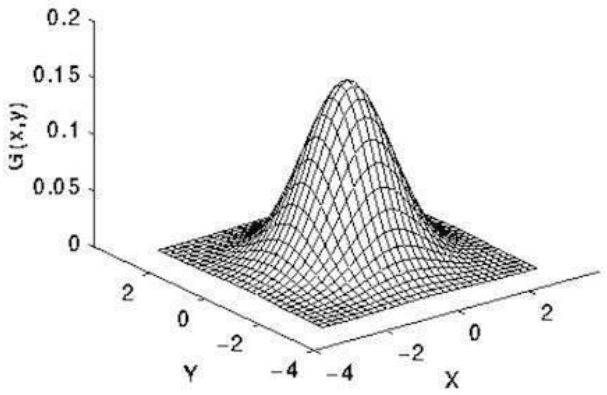
```
              ################################################################
              #                                                              #
  FATAL 00698:# SystemC fixed-point classes are not synthesizable            #
              #                                                              #
              ################################################################

        01445: Summary of messages of severity WARNING or greater:
        01193:   SEVERITY MSGID CNT
        01198:      FATAL 00698   1
        01198:    WARNING 90177   1

stratus_hls failed with 1 error and 1 warning.
```

因為高斯模糊的計算需要用到不動點，而不動點類是不可合成的，所以必須包含cynw_fixed header，這樣不動點才可以合成。

Gaussian Blur:

高斯模糊，也叫高斯平滑，是圖像處理中常用的一種技術，主要用於降低圖像噪聲和減少圖像細節。 在數值上，這是一種“平滑”。 從圖形上看，相當於產生了“模糊”的效果，“中間點”失去了細節。 顯然，在計算平均值時，取值範圍最大，“模糊效果”更強。 模糊半徑越大，圖像越模糊。 從數值上看，數值越平滑。 如果採用簡單的平均，顯然不是很合理，因為圖像是連續的。 點越近，關係越近，點越遠，關係越遠。 所以加權平均比較合理，越近的點權重越大，越遠的點權重越小。

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$



```
  cynw_ufixed<24,12> sum_r=0;
  cynw_ufixed<24,12> sum_g=0;
  cynw_ufixed<24,12> sum_b=0;
  for (int x=0;x<9;x++){
      sum_r+=  Gaussian[x]*val[0][x] ;
      sum_g+=  Gaussian[x]*val[1][x] ;
      sum_b+=  Gaussian[x]*val[2][x] ;
  }

  if (i < 5)
  {
    buffer[0][i * 3] =  sum_r.range(20,12);
    buffer[1][i * 3] =  sum_g.range(20,12);
    buffer[2][i * 3] =  sum_b.range(20,12);
  }
  else if (i >= 5 && i < 10)
  {
    buffer[0][(i - 5) * 3 + 1] =  sum_r.range(20,12);
    buffer[1][(i - 5) * 3 + 1] =  sum_g.range(20,12);
    buffer[2][(i - 5) * 3 + 1] =  sum_b.range(20,12);
  }
  else
  {
    buffer[0][(i - 10) * 3 + 2] =  sum_r.range(20,12);
    buffer[1][(i - 10) * 3 + 2] =  sum_g.range(20,12);
    buffer[2][(i - 10) * 3 + 2] =  sum_b.range(20,12);
  }
unsigned int buffer [3][15];
cynw_ufixed<24,12> sigma;
cynw_ufixed<24,12> constant;
int radium;
cynw_ufixed<24,12> Gaussian[9];
sigma=0.3*((3-1)*0.5 - 1) + 0.8;
constant=1/(2*3.141592*sigma*sigma);
radium=1;
int i=0;
cynw_ufixed<24,12> total_sum=0;
for (int i=0;i<3;i++){
  for(int j=0;j<3;j++){
        int x,y;
        cynw_ufixed<24,12> gaussian;
        x=i-radium;
        y=j-radium;
        gaussian = constant*expon[i][j];
        //cout<<gaussian<<endl;
        total_sum+=gaussian ;
        Gaussian[i*3+j]= gaussian ;
  }
}

for (int i=0;i<9;i++){
  Gaussian[i]=  Gaussian[i]/total_sum ;
}
{
```

上上圖是Gaussian blur的參數計算，因為需要用到fixed point ，所以datatype採用 cynw_ufixed，kernel 尺寸為3乘3，上圖就是去做Gaussian blur，它本質就是進行convolution，做完後，只取整數部分。

Laplacian Sharpen:

$$\nabla^2 f = 4f(x,y) - f(x-1,y) - f(x,y+1) - f(x+1,y) - f(x,y-1)$$

這樣可以找到一個模板矩陣:

$$\begin{matrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{matrix}$$

```
int sum_r=0;
int sum_g=0;
int sum_b=0;
for(int u=0;u< 3;u++){
    for(int j=0;j<3;j++){
        sum_r+=buffer[0][u * 3 + j]*sharpening[u][j];
        sum_g+=buffer[1][u * 3 + j]*sharpening[u][j];
        sum_b+=buffer[2][u * 3 + j]*sharpening[u][j];
    }
}
if(sum_r>255){
    sum_r=255;
}else if(sum_r<0){
    sum_r=0;
}else{
    sum_r=sum_r;
}
if(sum_g>255){
    sum_g=255;
}else if(sum_g<0){
    sum_g=0;
}else{
    sum_g=sum_g;
}

if(sum_b>255){
    sum_b=255;
}else if(sum_b<0){
    sum_b=0;
}else{
    sum_b=sum_b;
}
```

```
const int sharpening[MASK_X][MASK_Y]
```

```
= {{0,-1,0},{-1,5,-1},{0,-1,0}};
```

這部分是進行Laplacian Sharpen，將sharpening mask去做convolution後，截去低於0或高於255的數值。

```
### 4. Define your HLS configuration (arbitrary names, BASIC and DPA in this example).
define_hls_config SobelFilter BASIC
define_hls_config SobelFilter DPA       --dpopt_auto=op,expr
define_hls_config SobelFilter PIPELINE -DII=2
define_hls_config SobelFilter UNROLL_ALL --flatten_arrays=all -post_elab_tcl {
    unroll_loops [find -loop "*_loop" ]
    constrain_latency -max_lat $HLS::ACHIEVABLE [find -loop "while_1"]
}

set IMAGE_DIR           ".."
set IN_FILE_NAME        "${IMAGE_DIR}/lena_std_short.bmp"
set OUT_FILE_NAME       "out.bmp"

### 5. Define simulation configuration for each HLS configuration
### 5.1 The behavioral simulation (C++ only).
define_sim_config B -argv "$IN_FILE_NAME $OUT_FILE_NAME"
### 5.2 The Verilog simulation for HLS config "BASIC".
define_sim_config V_BASIC "SobelFilter RTL_V BASIC" -argv "$IN_FILE_NAME $OUT_FILE_NAME"
### 5.3 The Verilog simulation for HLS config "DPA".
define_sim_config V_DPA "SobelFilter RTL_V DPA" -argv "$IN_FILE_NAME $OUT_FILE_NAME"
###
define_sim_config V_PIPELINE "SobelFilter RTL_V PIPELINE" -argv "$IN_FILE_NAME $OUT_FILE_NAME"
###
define_sim_config V_UNROLL_ALL "SobelFilter RTL_V UNROLL_ALL" -argv "$IN_FILE_NAME $OUT_FILE_NAME"
```

這裡是project.tcl，我做了BASIC、DPA、PIPELINE、UNROLL等指令。

```
#ifndef NATIVE_SYSTEMC
  HLS_FLATTEN_ARRAY(val);
#endif
```

這裡是flatten array

```
while_1:
  while (true) {
    for (unsigned int v = 0; v<MASK_Y; ++v) {
      for (unsigned int u = 0; u<MASK_X; ++u) {
        sc_dt::sc_uint<24> rgb;
        #if defined (II)
            HLS_PIPELINE_LOOP(SOFT_STALL, II , "Pipeline" );
        #endif
#ifndef NATIVE_SYSTEMC
        {
          HLS_DEFINE_PROTOCOL("input");
          rgb = i_rgb.get();
          wait();
        }
#else
        rgb = i_rgb.read();
#endif

        val[0][v * 3 + u] = rgb.range(7, 0);
        val[1][v * 3 + u] = rgb.range(15, 8);
        val[2][v * 3 + u] = rgb.range(23, 16);
      }
    }
```

上面的while_1是做unroll，迴圈裡的是做pipeline。

Result:

V_BASIC:

```
        SystemC 2.3.3-Accellera --- Jun 11 2021 12:51:14
        Copyright (c) 1996-2018 by all Contributors,
        ALL RIGHTS RESERVED
NOTE: Cadence Design Systems Hub Simulation Platform : version 21.20-p100
start time:19460 ns,end time:21380 ns,Latency:1920 ns
start time:39950 ns,end time:41870 ns,Latency:1920 ns
start time:60440 ns,end time:62360 ns,Latency:1920 ns
start time:80930 ns,end time:82850 ns,Latency:1920 ns
```

```
done

Info: /OSCI/SystemC: Simulation stopped by user.
Simulation stopped via $stop(1) at time 445842805100 PS + 0
./bdw_work/sims/top_V_BASIC.v:67    #100 $stop;
xcelium> quit
Total run time = 445842740 ns
TOOL: xrun(64)  22.03-s003: Exiting on May 26, 2023 at 08:39:52 CST  (total: 01:34:24)
    01442: |                   Reg bits by type:                                                  |
    01442. |                      EN SS SC AS AC                                                  |
    00809: |                       0  0  1  0  0      2     5.5(1)       1.4                     |
    00809: |                       0  1  0  0  0      1     5.5(1)       1.4                     |
    00809: |                       1  0  0  0  0    286     7.5(1)       0.0                     |
    00809: |                       1  0  1  0  0    126     7.5(1)       1.4                     |
    00809: |                       1  1  0  0  0      2     7.5(1)       1.4                     |
    00809: |                   all register bits    417     7.5(1)       0.4          3310.6     |
    02604: |                     estimated cntrl      1                460.0           460.0     |
    00811: | ------------------------------------------------------------------------------------ |
    00812: |                           Total Area         4704.6(620)  21636.8   0.0  26341.3     |
           |                                                                                      |
           +--------------------------------------------------------------------------------------+
```

V_DPA:

```
        SystemC 2.3.3-Accellera --- Jun 11 2021 12:51:14
        Copyright (c) 1996-2018 by all Contributors,
        ALL RIGHTS RESERVED
NOTE: Cadence Design Systems Hub Simulation Platform : version 21.20-p100
start time:10890 ns,end time:11910 ns,Latency:1020 ns
start time:21930 ns,end time:22950 ns,Latency:1020 ns
start time:32970 ns,end time:33990 ns,Latency:1020 ns
start time:44010 ns,end time:45030 ns,Latency:1020 ns
start time:55050 ns,end time:56070 ns,Latency:1020 ns
Info: /OSCI/SystemC: Simulation stopped by user.
Simulation stopped via $stop(1) at time 240220235100 PS + 0
./bdw_work/sims/top_V_DPA.v:67    #100 $stop;
xcelium> quit
Total run time = 240220170 ns
TOOL: xrun(64)  22.03-s003: Exiting on May 26, 2023 at 14:11:01 CST  (total: 01:31:21)
    00808: |                           registers     83                                           |
    01442: |                   Reg bits by type:                                                  |
    01442. |                      EN SS SC AS AC                                                  |
    00809: |                       0  0  1  0  0      2     5.5(1)       1.4                     |
    00809: |                       0  1  0  0  0      1     5.5(1)       1.4                     |
    00809: |                       1  0  0  0  0    231     7.5(1)       0.0                     |
    00809: |                       1  0  1  0  0     93     7.5(1)       1.4                     |
    00809: |                       1  1  0  0  0      2     7.5(1)       1.4                     |
    00809: |                   all register bits    329     7.5(1)       0.4          2603.3     |
    02604: |                     estimated cntrl      1                350.1           350.1     |
    00811: | ------------------------------------------------------------------------------------ |
    00812: |                           Total Area         4042.4(532)  22605.1   0.0  26647.6     |
           |                                                                                      |
           +--------------------------------------------------------------------------------------+
```

V_PIPELINE:

```
        SystemC 2.3.3-Accellera --- Jun 11 2021 12:51:14
        Copyright (c) 1996-2018 by all Contributors,
        ALL RIGHTS RESERVED
NOTE: Cadence Design Systems Hub Simulation Platform : version 21.20-p100
start time:9690 ns,end time:10910 ns,Latency:1220 ns
start time:19800 ns,end time:21020 ns,Latency:1220 ns
start time:29910 ns,end time:31130 ns,Latency:1220 ns
start time:40020 ns,end time:41240 ns,Latency:1220 ns
start time:50130 ns,end time:51350 ns,Latency:1220 ns
```

```
done

Info: /OSCI/SystemC: Simulation stopped by user.
Simulation stopped via $stop(1) at time 219984295100 PS + 0
./bdw_work/sims/top_V_PIPELINE.v:67     #100 $stop;
xcelium> quit
Total run time = 219984230 ns
TOOL: xrun(64)  22.03-s003: Exiting on May 29, 2023 at 22:00:10 CST  (total: 01:28:37)
   01442: |                    Reg bits by type:                                                  |
   01442. |                       EN SS SC AS AC                                                  |
   00809: |                        0  0  1  0  0      3     5.5(1)        1.4                    |
   00809: |                        0  1  0  0  0      1     5.5(1)        1.4                    |
   00809: |                        1  0  0  0  0    298     7.5(1)        0.0                    |
   00809: |                        1  0  1  0  0    668     7.5(1)        1.4                    |
   00809: |                        1  1  0  0  0      2     7.5(1)        1.4                    |
   00809: |                    all register bits    972     7.5(1)        0.9         8227.2     |
   02604: |                      estimated cntrl      1                  448.6          448.6     |
   00811: | ------------------------------------------------------------------------------------- |
   00812: |                           Total Area        8878.3(1175)  26819.6   0.0  35697.9      |
          |                                                                                       |
          +---------------------------------------------------------------------------------------+
```

V_UNROLL_ALL:

```
xcelium>
xcelium> run

        SystemC 2.3.3-Accellera --- Jun 11 2021 12:51:14
        Copyright (c) 1996-2018 by all Contributors,
        ALL RIGHTS RESERVED
NOTE: Cadence Design Systems Hub Simulation Platform : version 21.20-p100
start time:4600 ns,end time:4900 ns,Latency:300 ns
start time:8920 ns,end time:9220 ns,Latency:300 ns
start time:13240 ns,end time:13540 ns,Latency:300 ns
start time:17560 ns,end time:17860 ns,Latency:300 ns
start time:21880 ns,end time:22180 ns,Latency:300 ns
start time:26200 ns,end time:26500 ns,Latency:300 ns
done

Info: /OSCI/SystemC: Simulation stopped by user.
Simulation stopped via $stop(1) at time 93999465100 PS + 0
./bdw_work/sims/top_V_UNROLL_ALL.v:67     #100 $stop;
xcelium> quit
Total run time = 93999400 ns
TOOL: xrun(64)  22.03-s003: Exiting on May 26, 2023 at 06:47:22 CST  (total: 00:36:44)
   01442: |                    Reg bits by type:                                                  |
   01442. |                       EN SS SC AS AC                                                  |
   00809: |                        0  0  1  0  0      2     5.5(1)        1.4                    |
   00809: |                        0  1  0  0  0      1     5.5(1)        1.4                    |
   00809: |                        1  0  0  0  0    509     7.5(1)        0.0                    |
   00809: |                        1  0  1  0  0   2296     7.5(1)        1.4                    |
   00809: |                    all register bits   2808     7.5(1)        1.1        24266.3     |
   02604: |                      estimated cntrl      1                  294.2          294.2     |
   00811: | ------------------------------------------------------------------------------------- |
   00812: |                           Total Area        22694.4(3011)  74162.2   0.0  96856.6      |
          |                                                                                       |
          +---------------------------------------------------------------------------------------+
```

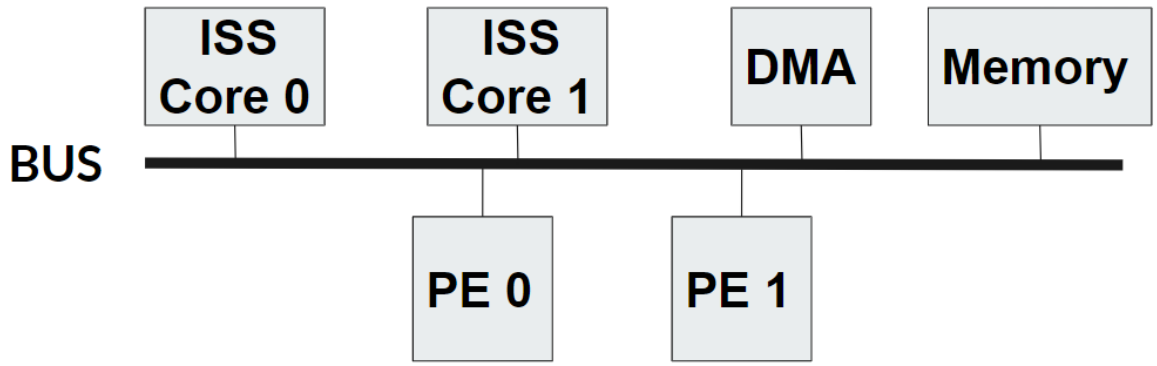
Method	Simulation time(ps)	Latency(ns)	PE Area	Total Area
BASIC	445842805100.00	1920	26232.9	26341.3
DPA	240220235100.00	1020	26634.9	26647.6
PIPELINE	219984295100.00	1220	40805.4	35697.9
UNROLL_ALL	93999465100.00	300	106148.7	96856.6

這裡是結果，可以看到Unroll的simulation time最少，因為loop都展開合成了，所以速度最快但缺點是面積非常大。

2.** (50 pt) Implement multiple accelerator PEs with a multi-core riscv-vp platform**

4. Please implement the SystemC PE TLM model for riscv-vp platform.
    1. This version can be none-synthesizable, e.g., use sc_fifo.
    2. Please use timing (cycles of latency) from synthesized and optimized PE in the Sys temC model
5. Please also implement the software on the multi-core processors.
    1. Please apply simple data partitioning to allocate tasks to different CPU-PE pairs.
    2. The main.cpp will use hartid, mutex and barrier to synchronize tasks from different CPU-PE pairs.
6. Please compute the DMA data amount for input and output data.
    1. All memory transactions between memory and hardware PEs should be done with DMA.
    2. Please convert DMA data amount to cycles. Each DMA transaction cycle can be m odeled = initial cycle + (sent data in bytes/DMA bandwidth). For example, assume a DMA transaction moves 256 bytes and initial cycle=2, DMA bandwith=4 bytes/c ycle. Then total DMA cycle=2+ 256/4=66.
7. Please add DMA cycles and PE computation cycles as the total application cycles.
    1. We can implement a status register in PE that notifies if the current work is done o r not.
    2. Software can use a loop to check the register to determine if the task of PE is finis hed and calculate the total latency for PE.

RISCV-VP:



RISCV-VP-main.cppRISCV-VP-main.cpp

```
addr_t Filter_0_start_addr = 0x73000000;
addr_t Filter_0_size = 0x01000000;
addr_t Filter_0_end_addr = Filter_0_start_addr + Filter_0_size - 1;
addr_t Filter_1_start_addr = 0x74000000;
addr_t Filter_1_size = 0x01000000;
addr_t Filter_1_end_addr = Filter_1_start_addr + Filter_1_size - 1;
```

```
SimpleBus<4, 14> bus("SimpleBus");
Filter Filter_0("Filter_0");
Filter Filter_1("Filter_1");
bus.ports[12] = new PortMapping(opt.Filter_0_start_addr, opt.Filter_0_end_addr);
bus.ports[13] = new PortMapping(opt.Filter_1_start_addr, opt.Filter_1_end_addr);
// connect TLM sockets
iss0_mem_if.isock.bind(bus.tsocks[0]);
iss1_mem_if.isock.bind(bus.tsocks[1]);
bus.isocks[12].bind(Filter_0.tsock);
bus.isocks[13].bind(Filter_1.tsock);
```

這裡是RISCV-VP的main.cpp裡，我這邊多加的PE，給了新的address，再將新的PE接上BUS

```
ISS core0(0, opt.use_E_base_isa);
ISS core1(1, opt.use_E_base_isa);
```

```
CombinedMemoryInterface iss0_mem_if("MemoryInterface0", core0);
CombinedMemoryInterface iss1_mem_if("MemoryInterface1", core1);
MemoryDMI dmi = MemoryDMI::create_start_size_mapping(mem.data, opt.mem_start_addr, mem.size);
InstrMemoryProxy instr_mem0(dmi, core0);
InstrMemoryProxy instr_mem1(dmi, core1);
core0.init(instr0_mem_if, data0_mem_if, &clint, entry_point, opt.mem_end_addr - 3);
core1.init(instr1_mem_if, data1_mem_if, &clint, entry_point, opt.mem_end_addr - 32767);
sys.init(mem.data, opt.mem_start_addr, loader.get_heap_addr());
sys.register_core(&core0);
sys.register_core(&core1);

if (opt.intercept_syscalls) {
    core0.sys = &sys;
    core1.sys = &sys;
}
```

這裡是2個cores，我一共使用兩個core來計算。

```
plic.target_harts[0] = &core0;
plic.target_harts[1] = &core1;
clint.target_harts[0] = &core0;
clint.target_harts[1] = &core1;
threads.push_back(&core0);
threads.push_back(&core1);

core0.trace = opt.trace_mode;  // switch for printing instructions
core1.trace = opt.trace_mode;  // switch for printing instructions
```

```
if (opt.use_debug_runner) {
    auto server = new GDBServer("GDBServer", threads, &dbg_if, opt.debug_port);
    new GDBServerRunner("GDBRunner", server, &core0);
    new GDBServerRunner("GDBRunner", server, &core1);
} else {
    new DirectCoreRunner(core0);
    new DirectCoreRunner(core1);
}
```

here is connect interrupt signals and switch for printing instructions。

Software:

```
//Total number of cores
//static const int PROCESSORS = 2;
#define PROCESSORS 2
//the barrier synchronization objects
uint32_t barrier_counter=0;
uint32_t barrier_lock;
uint32_t barrier_sem;
//the mutex object to control global summation
uint32_t lock;
uint32_t dma_lock;
//print synchronication semaphore (print in core order)
uint32_t print_sem[PROCESSORS];
```

here is the barrier synchronization objects and the mutex object to control global summation 。

```
if (hart_id == 0) {
  // create a barrier object with a count of PROCESSORS

  sem_init(&barrier_lock, 1);
  sem_init(&barrier_sem, 0); //lock all cores initially
  for(int i=0; i< PROCESSORS; ++i){
    sem_init(&print_sem[i], 0); //lock printing initially
  }
  // Create mutex lock
  sem_init(&lock, 1);
  sem_init(&dma_lock, 1);
}
```

create mutex lock and create a barrier object

Software-DMA:

```c
void write_data_to_ACC(char* ADDR, unsigned char* buffer, int len, int hart_id){
  if(_is_using_dma){
    // Using DMA
    sem_wait(&dma_lock);
    *DMA_SRC_ADDR = (uint32_t)(buffer);
    *DMA_DST_ADDR = (uint32_t)(ADDR);
    *DMA_LEN_ADDR = len;
    *DMA_OP_ADDR  = DMA_OP_MEMCPY;
    //calculate DMA cycle
    //Please convert DMA data amount to cycles. Each DMA transaction cycle can be
    //modeled = initial cycle + (sent data in bytes/DMA bandwidth).
    //For example, assume a DMA transaction moves 256 bytes and initial cycle=2, DMA bandwith=4 by
    // Then total DMA cycle=2+ 256/4=66.
    write_data_cycles += initial_cycles + (len/DMA_Bandwidth);
    sem_post(&dma_lock);


  }else{
    // Directly Send
    memcpy(ADDR, buffer, sizeof(unsigned char)*len);

  }
}
```

使用DMA來傳送資料，並使用dma lock來控制 global summation，還有計算DAM 的 total cycle

```c
if (hart_id==0){
  unsigned char  buffer[4] = {0};
  word data;
  int adjustX, adjustY, xBound, yBound;
  int total;
  sem_wait(&lock);
  printf("Start processing...(%d, %d),core%d\n", width, height,hart_id);
  read_bmp("lena_std_short.bmp");
  printf("====================================\n");
  printf("\t  Reading from array\n");
  printf("====================================\n");
  printf(" input_rgb_raw_data_offset\t= %d\n", input_rgb_raw_data_offset);
  printf(" width\t\t\t\t= %d\n", width);
  printf(" height\t\t\t\t= %d\n", height);
  printf(" bytes_per_pixel\t\t= %d\n",bytes_per_pixel);
  printf("====================================\n");
  sem_post(&lock);
  int start = (hart_id == 0 ? 2 : 2+height/2);
  int end = (hart_id == 0 ? 2+height/2 : height);
  for(int i = 0; i < width; i++)
  {
    int count = 0;
    //printf("y_0:%d,hart_id=%d\n",i,hart_id);

    for(int j = start ; j < end; j=j+3)
    {
      //printf("pixel (%d, %d); \n", i, j);
      adjustX = 1 ;// 1
      adjustY = 1 ; // 1
      xBound = 1;            // 1
      yBound = 1;            // 1
```

這裡是讀取bmp初始狀態，並根據hart_id也就是core 0或core 1去區分要做哪部分，我這裡將圖片的高分成兩部分給不同core去計算。

```
const SOBELFILTER_START_ADDR[2] = {reinterpret_cast<char* const>(0x73000000),reinterpret_cast<char* const>(0x74000000)};
const SOBELFILTER_READ_ADDR [2] = {reinterpret_cast<char* const>(0x73000004),reinterpret_cast<char* const>(0x74000004)};
const SOBELFILTER_CYCLE_ADDR[2] = {reinterpret_cast<char* const>(0x73000008),reinterpret_cast<char* const>(0x74000008)};
```

```
write_data_to_ACC(SOBELFILTER_START_ADDR[hart_id], buffer, 4, hart_id);
read_data_from_ACC(SOBELFILTER_READ_ADDR[hart_id], buffer, 4, hart_id);

memcpy(data.uc, buffer, 4);
total = (data).sint;
*(target_bitmap + bytes_per_pixel * (width * i + j ) + 2) =data.uc[0];
*(target_bitmap + bytes_per_pixel * (width * i + j ) + 1) =data.uc[1];
*(target_bitmap + bytes_per_pixel * (width * i + j ) + 0) =data.uc[2];
read_data_from_ACC(SOBELFILTER_CYCLE_ADDR[hart_id], buffer, 4, hart_id);
memcpy(data.uc, buffer, 4);
unsigned int latency=(data).sint;

PE_cycles += 1;
PE_latency =PE_latency + latency;
printf("0:core:%d,:data.uint:%d,PE_latency:%d,PE_cycles:%d\n",hart_id,latency,PE_latency,PE_cycles);
```

根據不同的core 來去寫入不同的address，並在輸出的時候計算PE的cycles和PE的latency

```
Start processing...(0, 0),core1
[sys_open] lena_std_short.bmp, 0 (translated to 0), 438
=====================================
          Reading from array
=====================================
 input_rgb_raw_data_offset      = 54
 width                          = 256
 height                         = 256
 bytes_per_pixel                = 3
=====================================
Start processing...(256, 256),core0
[sys_open] lena_std_short.bmp, 0 (translated to 0), 438
=====================================
          Reading from array
=====================================
 input_rgb_raw_data_offset      = 54
 width                          = 256
 height                         = 256
 bytes_per_pixel                = 3
=====================================
width:0,core1
width:0,core0
width:1,core1
width:1,core0
width:2,core1
width:2,core0
```

可以看到右圖的core在一直交換輸出

Result:

single core without DMA:

```
=[ core : 0 ]==========================
simulation time: 18975999240 ns
zero (x0) =        0
ra   (x1) =    100b2
sp   (x2) =  1ffffec
gp   (x3) =    218f8
tp   (x4) =        0
t0   (x5) =     232a
t1   (x6) =        0
t2   (x7) = 23289efc
s0/fp(x8) =        0
s1   (x9) =        0
a0  (x10) =        0
a1  (x11) =        0
a2  (x12) =       12
a3  (x13) =    21798
a4  (x14) =    21d60
a5  (x15) =        0
a6  (x16) =        0
a7  (x17) =       5d
s2  (x18) =        0
s3  (x19) =        0
s4  (x20) =        0
s5  (x21) =        0
s6  (x22) =        0
s7  (x23) =        0
s8  (x24) =        0
s9  (x25) =        0
s10 (x26) =        0
s11 (x27) =        0
t3  (x28) =        3
t4  (x29) =    825be
t5  (x30) =        0
t6  (x31) =        0
pc = 1c608
num-instr = 472174105
```

single core with DMA:

```
=[ core : 0 ]==========================
simulation time: 18015078320 ns
zero (x0) =        0
ra   (x1) =    100b2
sp   (x2) =  1ffffec
gp   (x3) =    218f8
tp   (x4) =        0
t0   (x5) =     232a
t1   (x6) =        0
t2   (x7) = 2a3724e8
s0/fp(x8) =        0
s1   (x9) =        0
a0  (x10) =        0
a1  (x11) =        0
a2  (x12) =       12
a3  (x13) =    21798
a4  (x14) =    21d60
a5  (x15) =        0
a6  (x16) =        0
a7  (x17) =       5d
s2  (x18) =        0
s3  (x19) =        0
s4  (x20) =        0
s5  (x21) =        0
s6  (x22) =        0
s7  (x23) =        0
s8  (x24) =        0
s9  (x25) =        0
s10 (x26) =        0
s11 (x27) =        0
t3  (x28) =        3
t4  (x29) =    825be
t5  (x30) =        0
t6  (x31) =        0
pc = 1c608
num-instr = 418122283
user@ubuntu:~/ee6470/docker-images/EE6470/riscv-vp/sw/basic-Gussian-Sharpen-singlecore$
```

multi-core without DMA:

```
=[ core : 0 ]============================
simulation time: 12765253090 ns
zero (x0) =        0
ra   (x1) =    11938
sp   (x2) =    19a00
gp   (x3) =    32ae8
tp   (x4) =        0
t0   (x5) =  2010000
t1   (x6) =        1
t2   (x7) =        1
s0/fp(x8) =        0
s1   (x9) =        0
a0  (x10) =        0
a1  (x11) =    32dd0
a2  (x12) =       27
a3  (x13) =       27
a4  (x14) =        1
a5  (x15) =        0
a6  (x16) =        0
a7  (x17) =       5d
s2  (x18) =        0
s3  (x19) =        0
s4  (x20) =        0
s5  (x21) =        0
s6  (x22) =        0
s7  (x23) =        0
s8  (x24) =        0
s9  (x25) =        0
s10 (x26) =        0
s11 (x27) =        0
t3  (x28) =        3
t4  (x29) =    32df6
t5  (x30) =        0
t6  (x31) =        0
pc = 11964
num-instr = 323133965
```

```
=[ core : 1 ]============================
simulation time: 12765253090 ns
zero (x0) =        0
ra   (x1) =    11938
sp   (x2) =    21a00
gp   (x3) =    32ae8
tp   (x4) =        0
t0   (x5) =    21a00
t1   (x6) =        1
t2   (x7) =        1
s0/fp(x8) =        0
s1   (x9) =        0
a0  (x10) =        0
a1  (x11) =    32dd0
a2  (x12) =       27
a3  (x13) =       27
a4  (x14) =        1
a5  (x15) =        0
a6  (x16) =        6
a7  (x17) =       40
s2  (x18) =        0
s3  (x19) =        0
s4  (x20) =        0
s5  (x21) =        0
s6  (x22) =        0
s7  (x23) =        0
s8  (x24) =        0
s9  (x25) =        0
s10 (x26) =        0
s11 (x27) =        0
t3  (x28) =        3
t4  (x29) =    32df6
t5  (x30) = 8da38651
t6  (x31) =        0
pc = 1194c
num-instr = 330508381
```

multi-core with DMA:

```
Info: /OSCI/SystemC: Simulation stopped by user.
=[ core : 0 ]============================
simulation time: 12277354520 ns
zero (x0) =        0
ra   (x1) =    11938
sp   (x2) =    19a00
gp   (x3) =    32ae0
tp   (x4) =        0
t0   (x5) =  2010000
t1   (x6) =        1
t2   (x7) =        1
s0/fp(x8) =        0
s1   (x9) =        0
a0  (x10) =        0
a1  (x11) =    32dc8
a2  (x12) =       27
a3  (x13) =       27
a4  (x14) =        1
a5  (x15) =        0
a6  (x16) =        0
a7  (x17) =       5d
s2  (x18) =        0
s3  (x19) =        0
s4  (x20) =        0
s5  (x21) =        0
s6  (x22) =        0
s7  (x23) =        0
s8  (x24) =        0
s9  (x25) =        0
s10 (x26) =        0
s11 (x27) =        0
t3  (x28) =        3
t4  (x29) =    32dee
t5  (x30) =        0
t6  (x31) =        0
pc = 11964
num-instr = 295256245
```

```
=[ core : 1 ]============================
simulation time: 12277354520 ns
zero (x0) =        0
ra   (x1) =    11938
sp   (x2) =    21a00
gp   (x3) =    32ae0
tp   (x4) =        0
t0   (x5) =    21a00
t1   (x6) =        1
t2   (x7) =        1
s0/fp(x8) =        0
s1   (x9) =        0
a0  (x10) =        0
a1  (x11) =    32dc8
a2  (x12) =       27
a3  (x13) =       27
a4  (x14) =        1
a5  (x15) =        0
a6  (x16) =        6
a7  (x17) =       40
s2  (x18) =        0
s3  (x19) =        0
s4  (x20) =        0
s5  (x21) =        0
s6  (x22) =        0
s7  (x23) =        0
s8  (x24) =        0
s9  (x25) =        0
s10 (x26) =        0
s11 (x27) =        0
t3  (x28) =        3
t4  (x29) =    32dee
t5  (x30) = 8da38651
t6  (x31) =        0
pc = 1194c
num-instr = 302709920
```

Single Core	Simulation time(ns)	Num-instr
DMA-Open	18015078320.00	418122283
DMA-Close	18975999240.00	472174105

```
====================================
Total PE  cycles= 21760
Total PE  latency= 1727409 us
Total application cycles= 3046400
Write data cycle= 2937600,Read data cycle=87040,and total DMA cycles= 3024640
====================================
```

Multi- Core 0	Simulation time(ns)	Num-instr	Multi- Core 1	Simulation time(ns)	Num-instr
DMA-Open	12277354520.00	295256245	DMA-Open	12277354520.00	302709920
DMA-Close	12765253090.00	330508381	DMA-Close	12765253090.00	330508381

最上圖是single core result，中圖是DMA cycles PE computation cycles ,and total application cycles，最下圖是 multi-core result，有使用DMA的simulation and Num-instr比較少一點，然後使用multi-core的simulation and Num-instr比使用single core少了1/3倍，會少的原因不外乎就是使用了兩core去進行計算。

可以看到直接進行銳化處理的圖片看起來噪點很多，但有先經過高斯模糊再銳化的圖片看起來就更平滑了，而且比原圖更多細節。