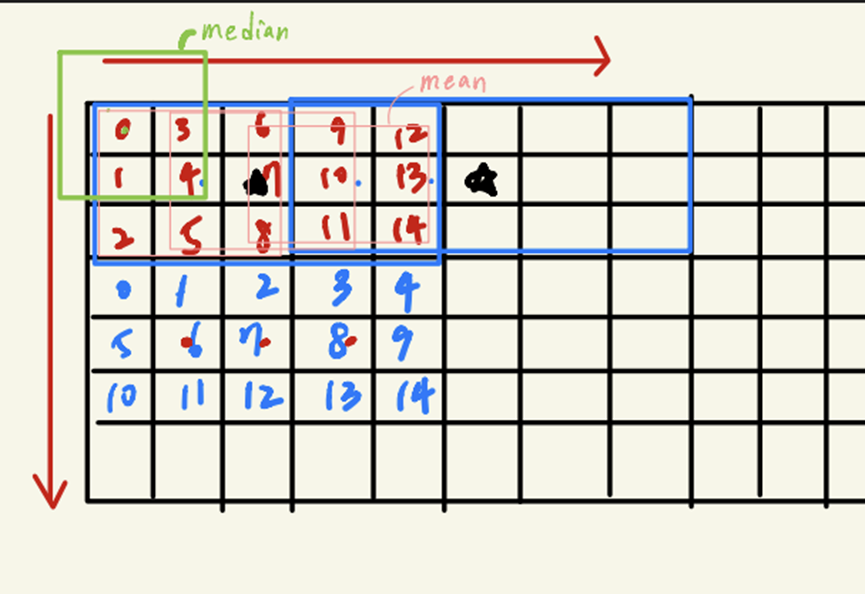
E111064503 通訊所 吳紹齊 Final project

<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**

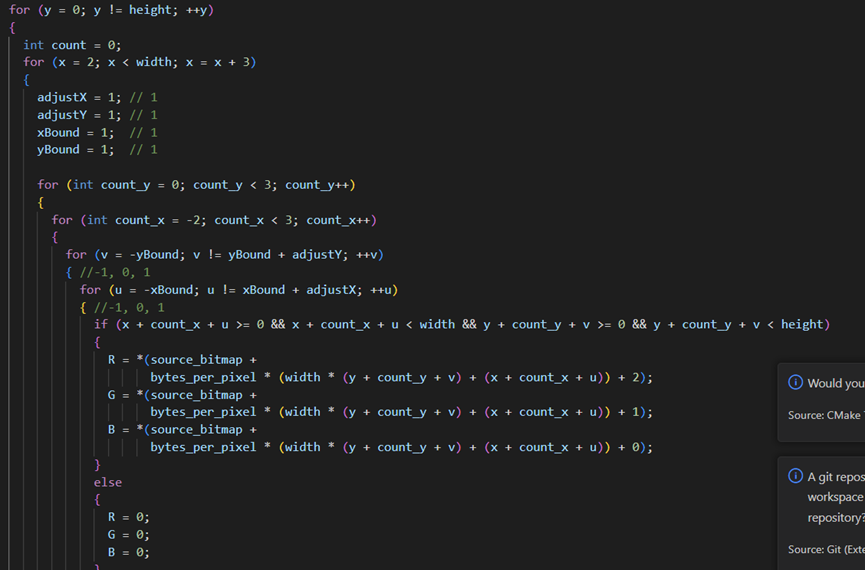
**再讀第三行buffer index為:2、5、8、11、14**

**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。**

**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。**

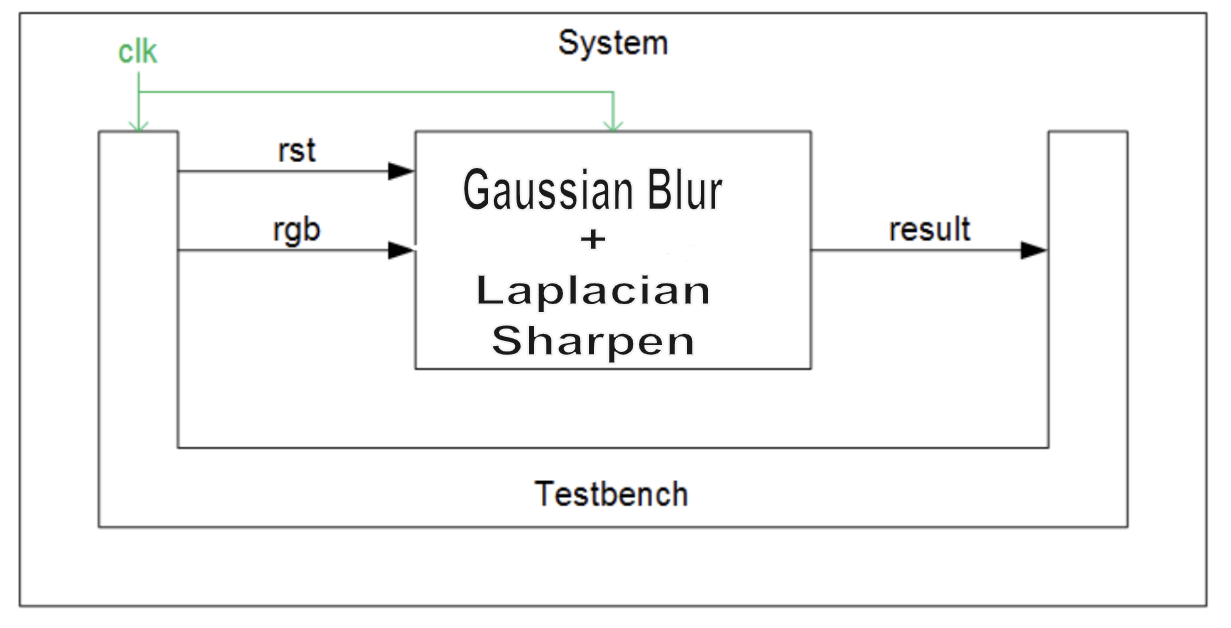
**中間四個迴圈總共會寫入135個pixel得到三個mean pixel**

****

**上式是我所加的buffer，一共可以存15個pixel。**

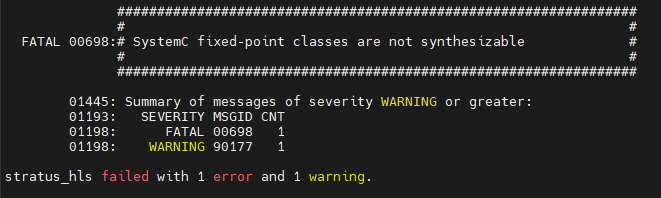
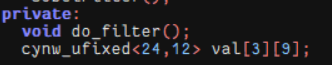
1.(50 pt) Implement HLS accelerator PEs\*\*

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。

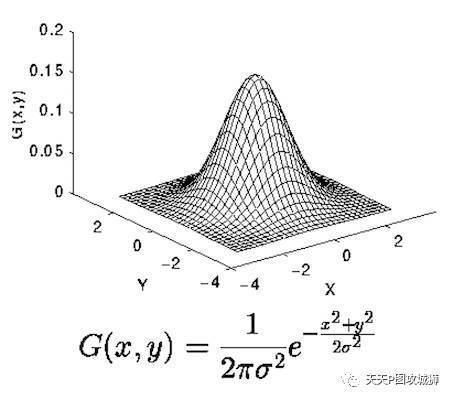
**HLS-Cynw:**

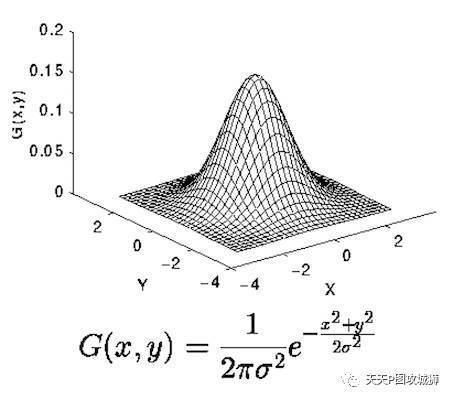
****

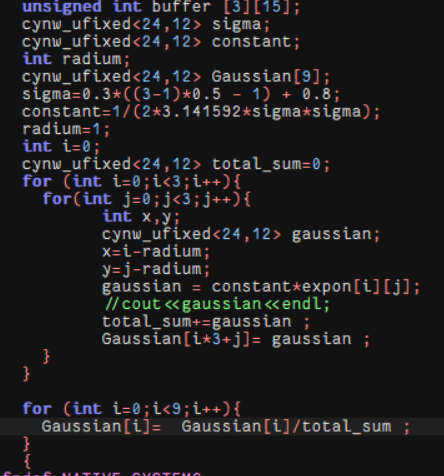
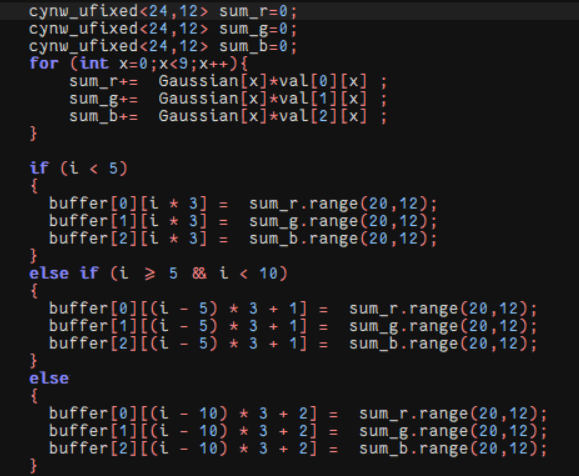
因為高斯模糊的計算需要用到不動點，而不動點類是不可合成的，所以必須包含cynw\_fixed header，這樣不動點才可以合成。

**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。

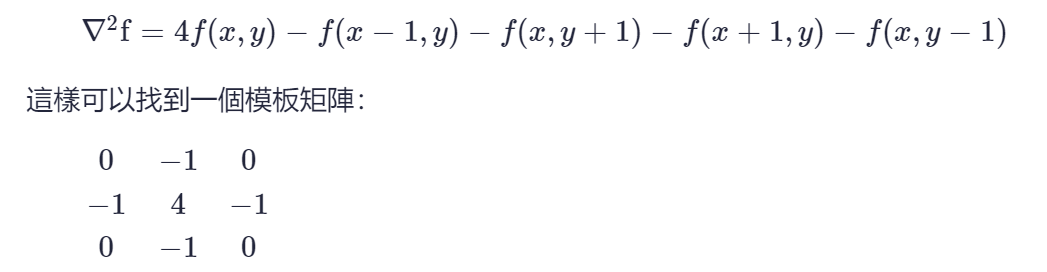
****

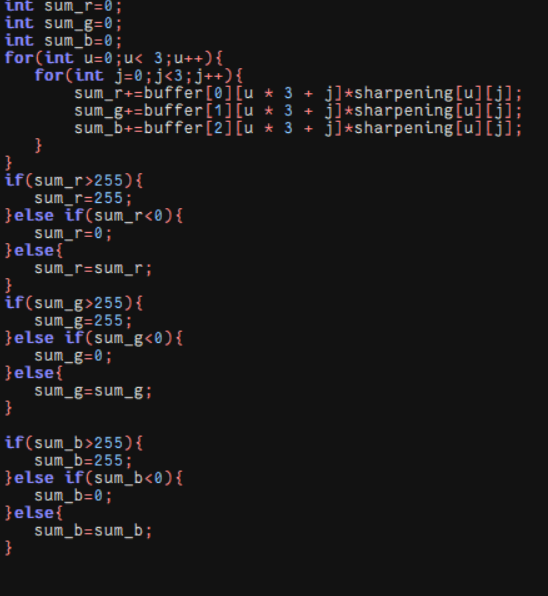
****

****

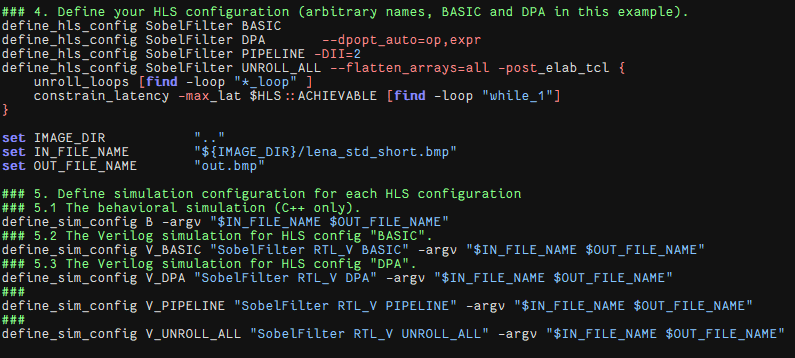
上上圖是Gaussian blur的參數計算，因為需要用到fixed point ，所以datatype採用 cynw\_ufixed，kernel 尺寸為3乘3，上圖就是去做Gaussian blur，它本質就是進行convolution，做完後，只取整數部分。

**Laplacian Sharpen:**

****

****

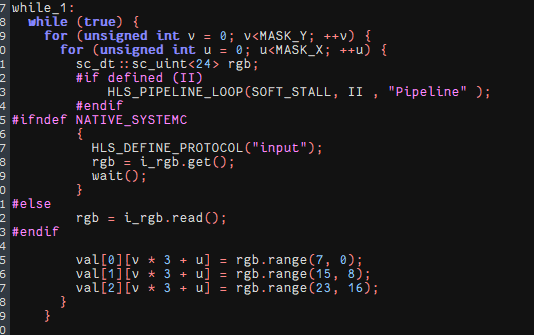
這部分是進行Laplacian Sharpen，將sharpening mask去做convolution後，截去低於0或高於255的數值。



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



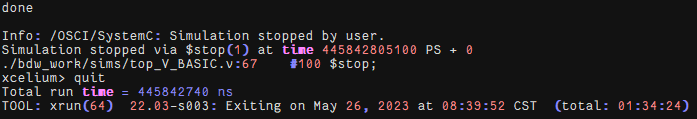
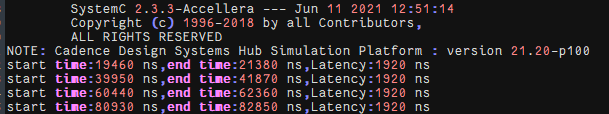
這裡是flatten array

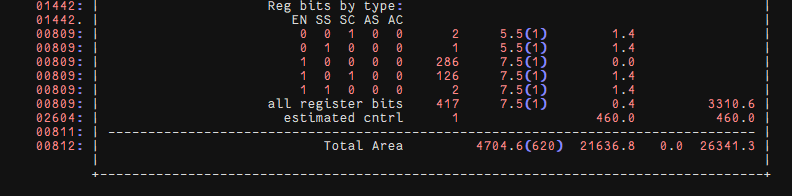


上面的while\_1是做unroll，迴圈裡的是做pipeline。

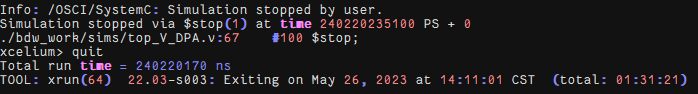
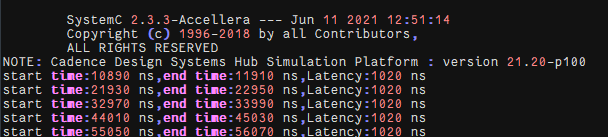
**Result:**

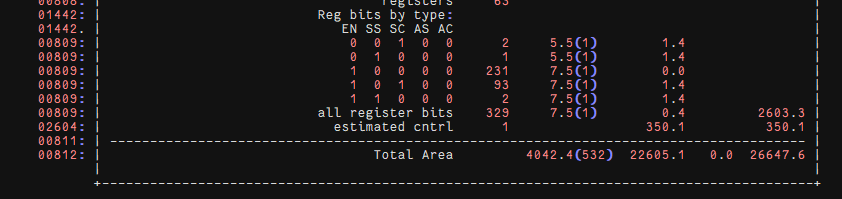
**V\_BASIC:**

****

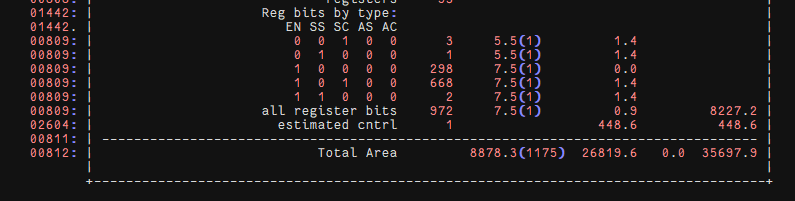
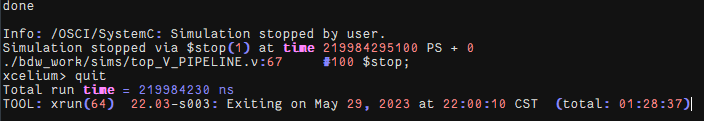
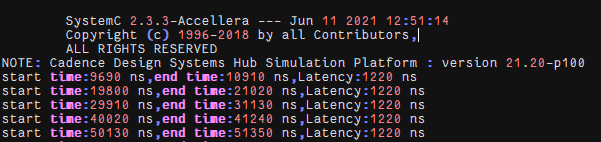
****

**V\_DPA:**

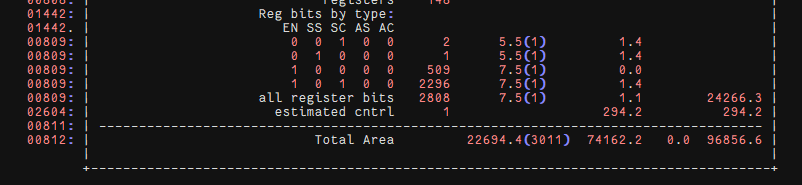
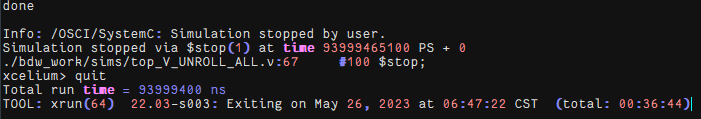
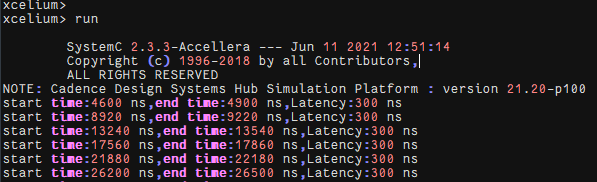
****

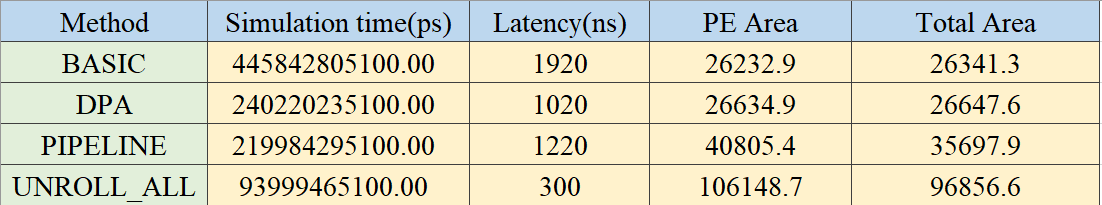
****

**V\_PIPELINE:**

****

**V\_UNROLL\_ALL:**

****

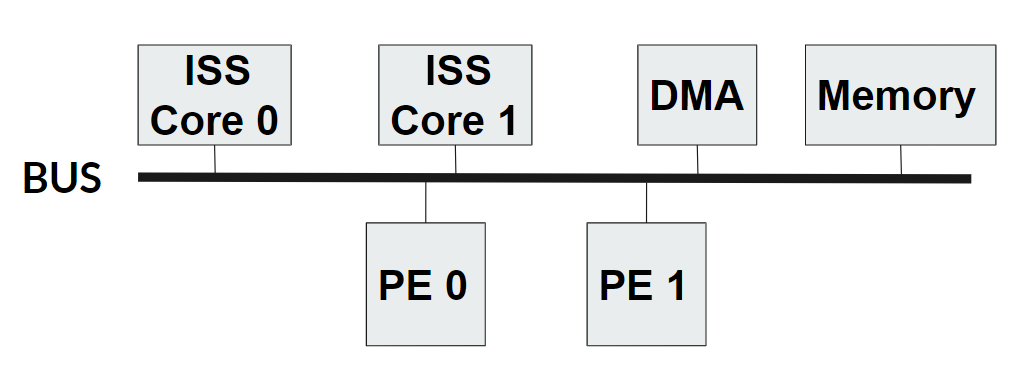
****

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。

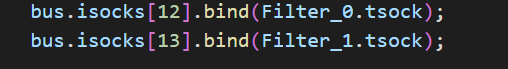
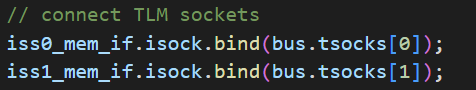
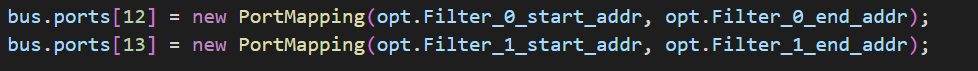
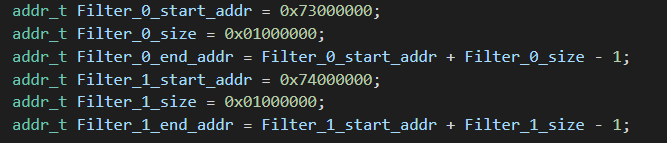
2.\*\* (50 pt) Implement multiple accelerator PEs with a multi-core riscv-vp platform\*\*

1. 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 SystemC model
2. 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.
3. 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 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 bytes/cycle. Then total DMA cycle=2+ 256/4=66.
4. 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 or not.
   2. Software can use a loop to check the register to determine if the task of PE is finished and calculate the total latency for PE.

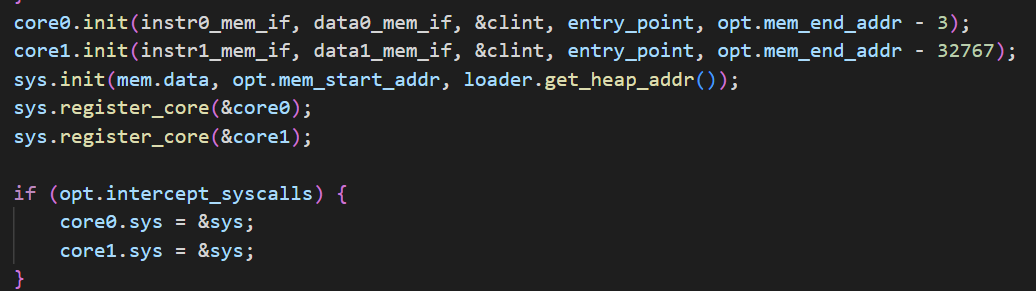
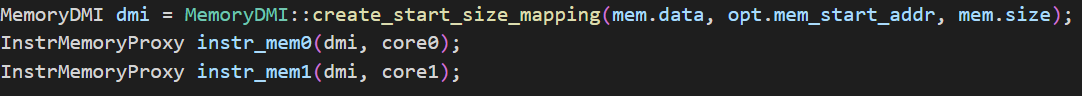
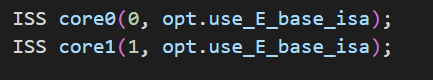
**RISCV-VP:**

****

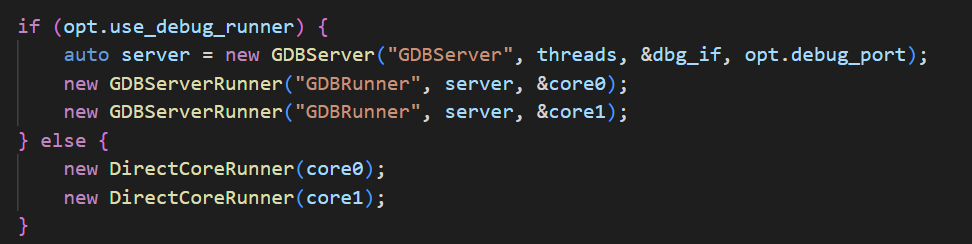
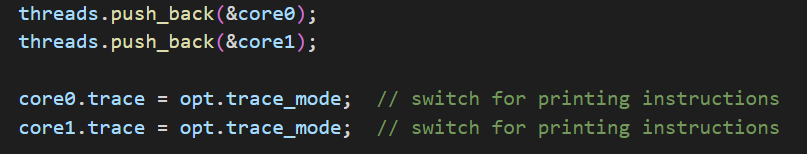
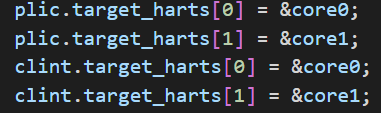
**RISCV-VP-main.cppRISCV-VP-main.cpp**

****

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

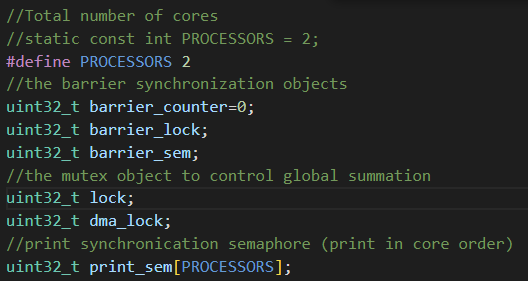
****

這裡是2個cores，我一共使用兩個core來計算。

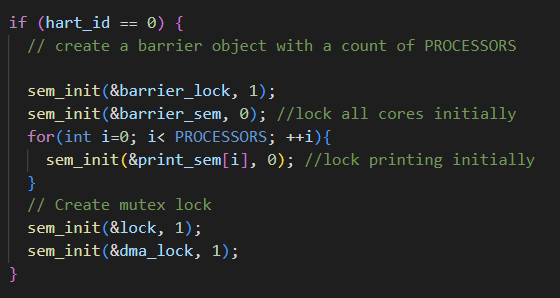


here is connect interrupt signals and switch for printing instructions。

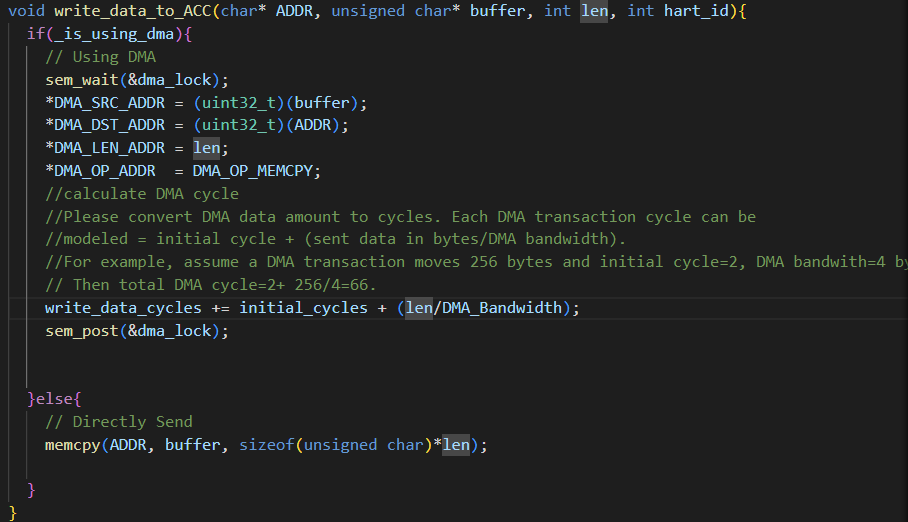
**Software:**

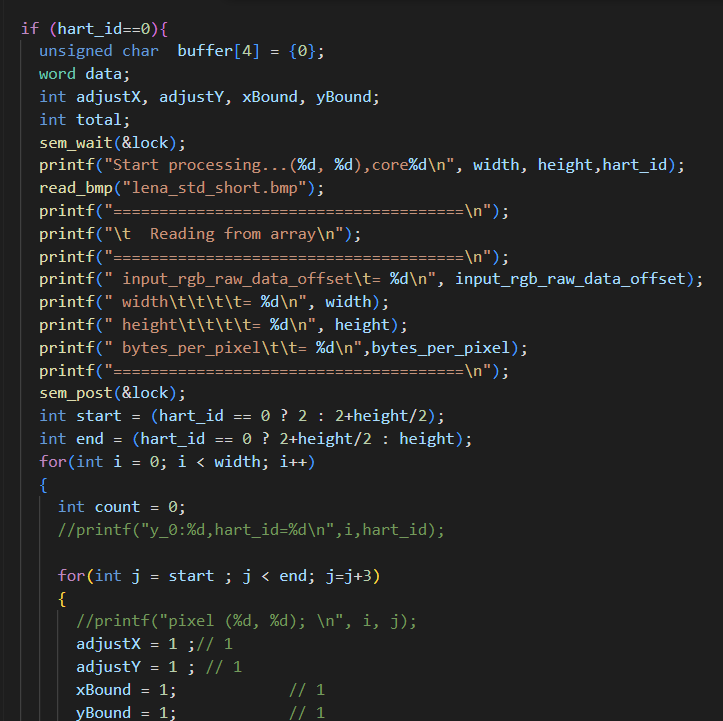
****

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

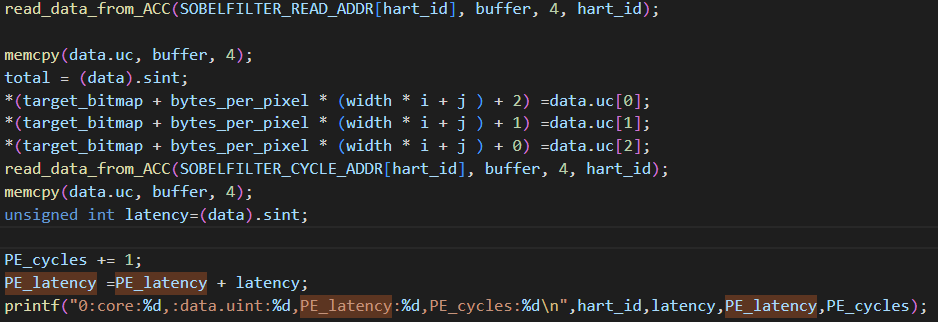
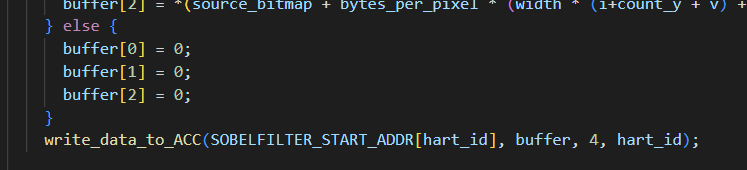
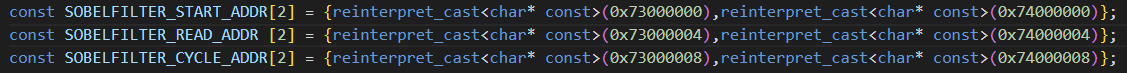


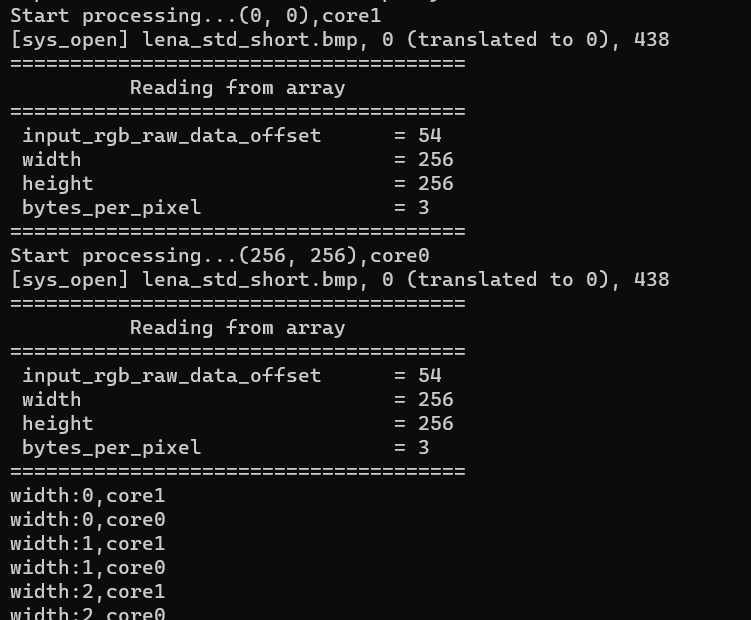
create mutex lock and create a barrier object

**Software-DMA:**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

****

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。

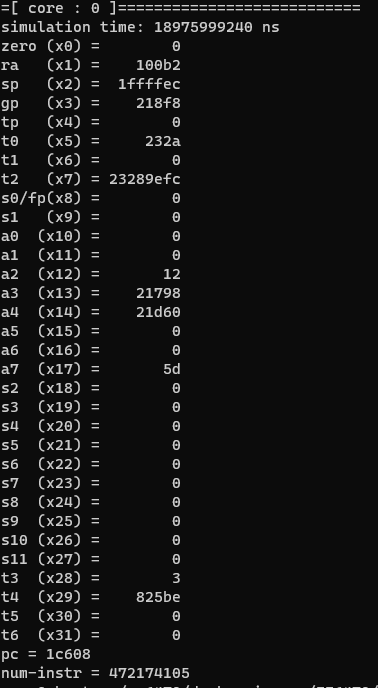
****根據不同的core 來去寫入不同的address，並在輸出的時候計算PE的cycles和PE的latency



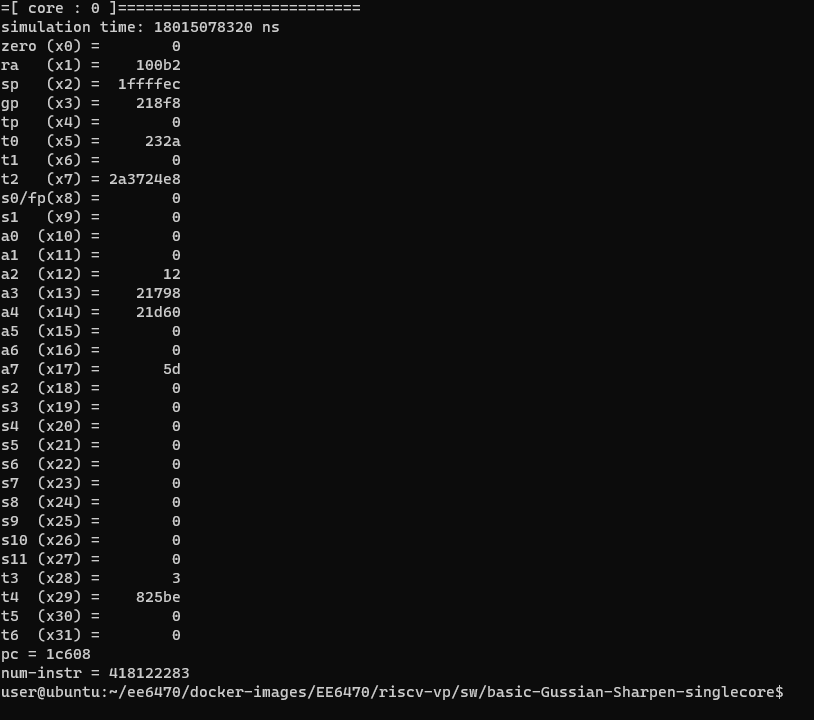
可以看到右圖的core在一直交換輸出

**Result:**

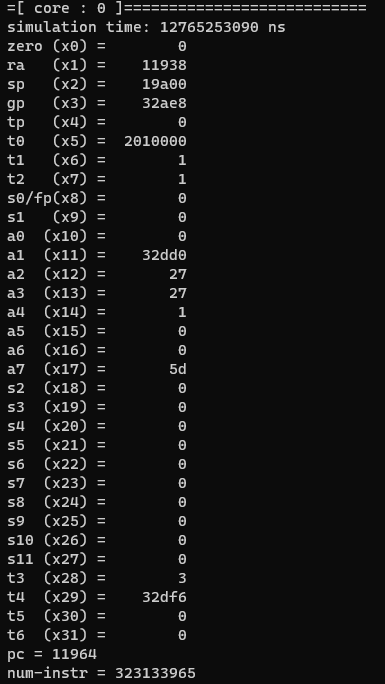
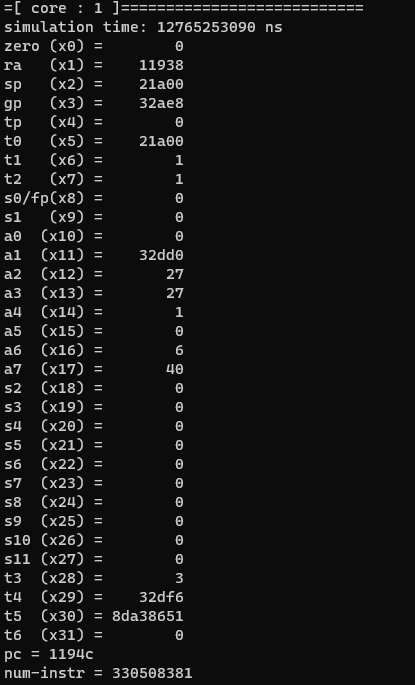
**single core without DMA:**

****

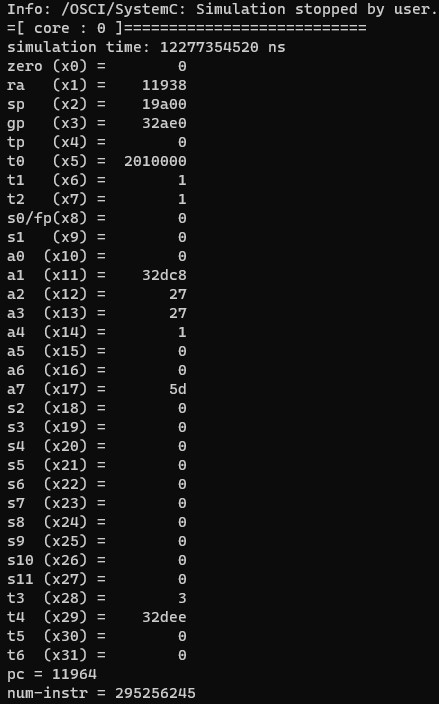
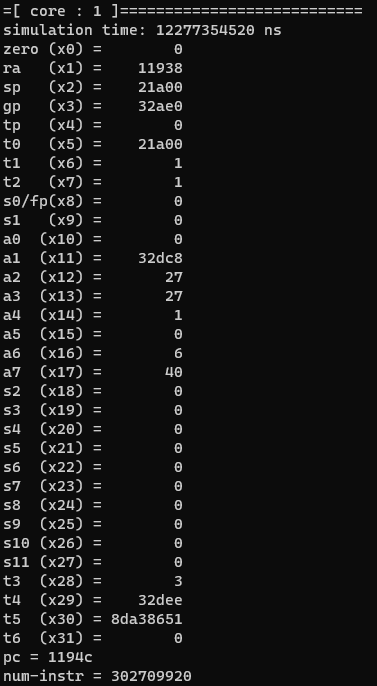
**single core with DMA:**

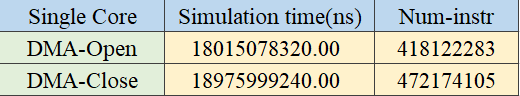
****

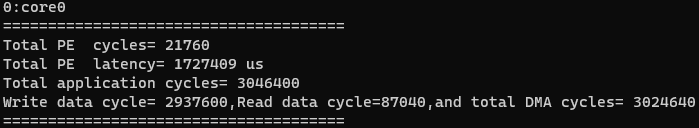
**multi-core without DMA:**

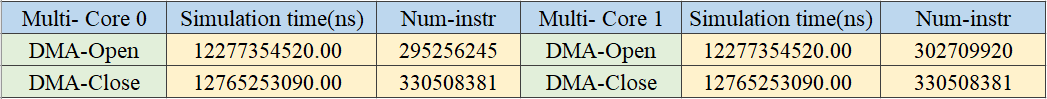
** **

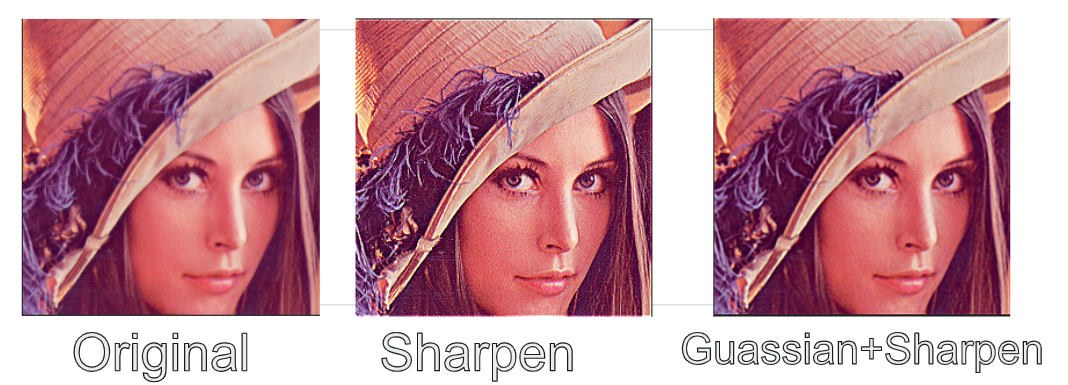
**multi-core with DMA:**

** **





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。