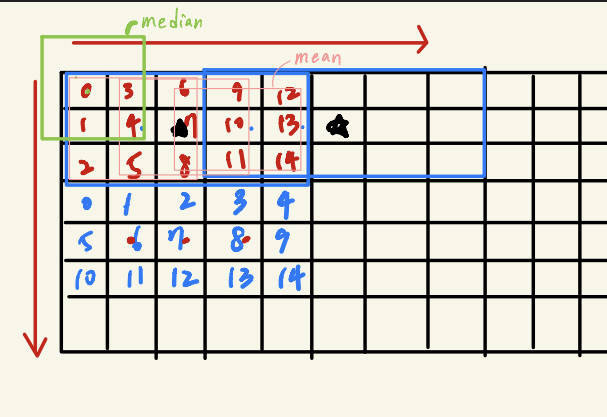
E111064503 通訊所 吳紹齊 HW2 <https://github.com/cowboy35927/ESL/tree/main/Hw2>

**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

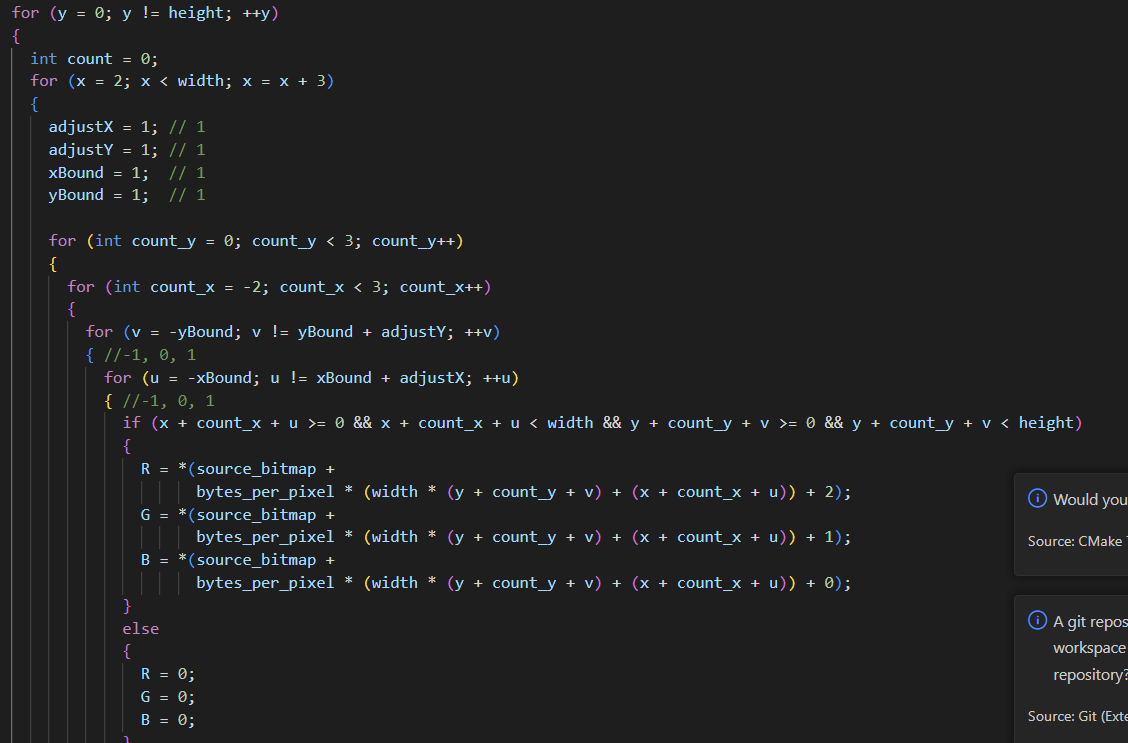
再讀第三行buffer index為:2、5、8、11、14

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。

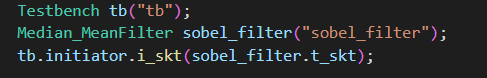
最內層是控制綠色框的median filter的順序。

中間四個迴圈總共會寫入135個pixel得到三個mean pixel

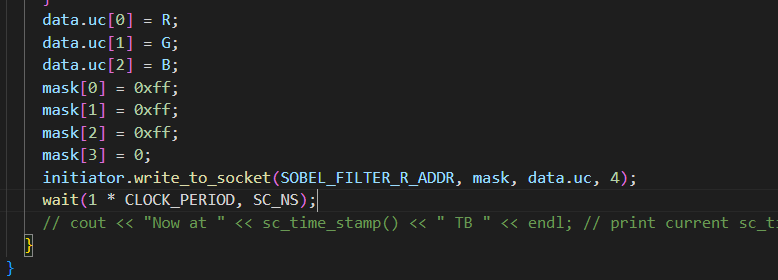


上式是我所加的buffer，一共可以存15個pixel。

1. **Median and mean filters with TLM interface**



**上圖是main.cpp 初始化，最下行是bind initiator socket to target socket。**

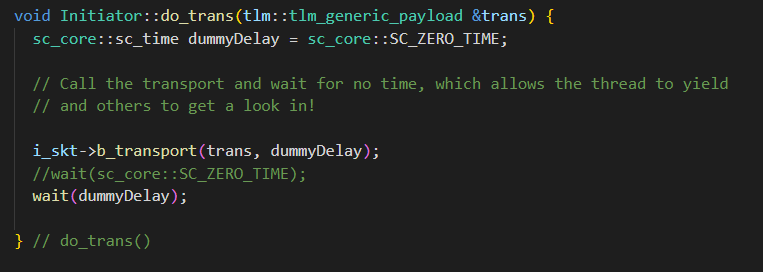


一張含有 文字 的圖片

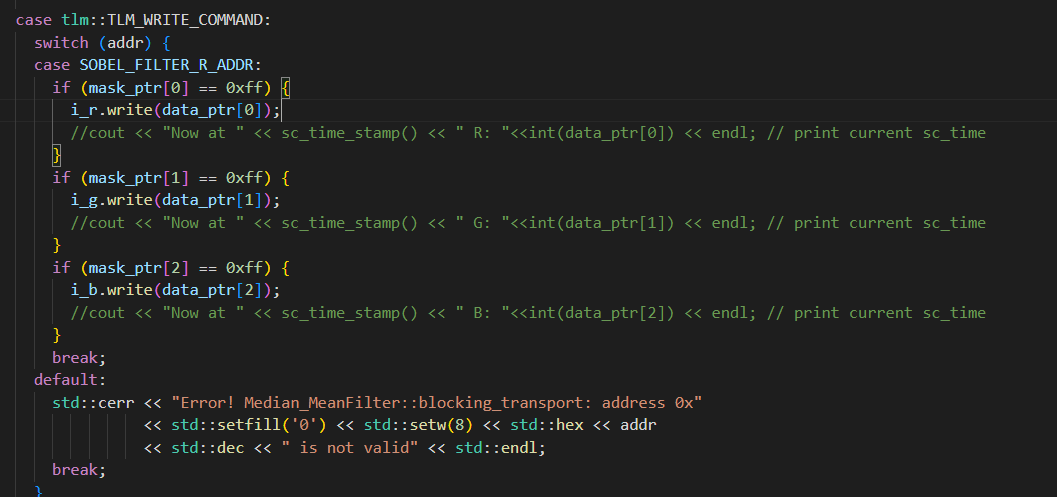
自動產生的描述

**write\_to\_socket函式會我們設置了**payload**。 寫入**

**SOBEL\_FILTER\_R\_ADDR地址。**



**準備好有效payload後，我們調用阻塞傳輸函數來發送數據。do\_trans中的b\_transport()，然後我們調用 wait(delay) 以 target 返回的 dummyDelay 值提前 Systemc 時間。**



**上圖就是當address是SOBEL\_FILTER\_R\_ADDR會執行tlm::TLM\_WRITE\_COMMAND的指令，會將i\_r、i\_g、i\_b分別寫入資料。**

一張含有 文字, 筆記型電腦, 螢幕擷取畫面, 屏幕、螢幕 的圖片

自動產生的描述

**read\_from\_socket函式會我們設置了**payload**。 寫入**

**SOBEL\_FILTER\_R\_ADDR地址。**

一張含有 文字 的圖片

自動產生的描述

**write\_to\_socket函式會我們設置了**payload**。 寫入**

**SOBEL\_FILTER\_CHECK\_ADDR地址。**一張含有 文字 的圖片

自動產生的描述

**上圖就是當address是SOBEL\_FILTER\_R\_ADDR會執行tlm::TLM\_WRITE\_COMMAND的指令，如果o\_result.num\_available()不=0時，就會跳出迴圈，**

一張含有 文字 的圖片

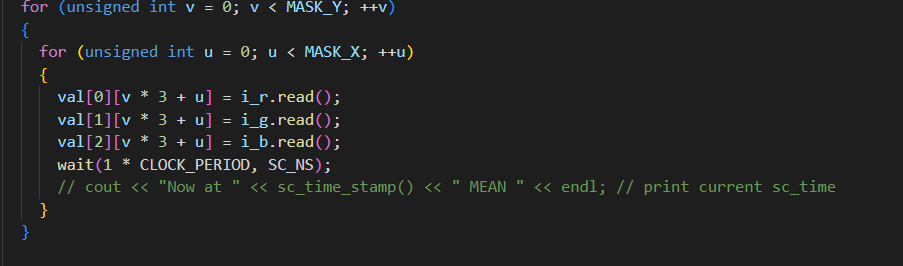
自動產生的描述

**寫資料進到buffer.uint後，在寫入target\_bitmap。**

一張含有 文字, 筆記型電腦, 螢幕擷取畫面, 屏幕、螢幕 的圖片

自動產生的描述

**Target socket module的wait:**



**在每read完rgb三個pixel後，wait一次**

一張含有 文字 的圖片

自動產生的描述

**在每write完o\_result後，wait一次**

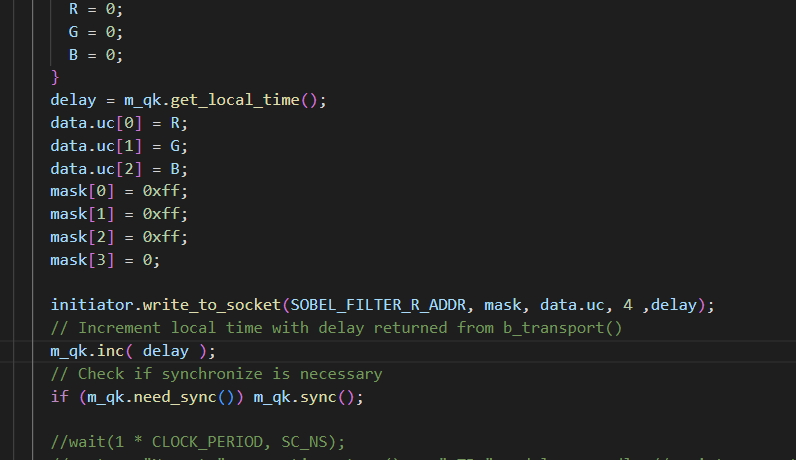
1. **Median and mean filters with quantum keeper**

****

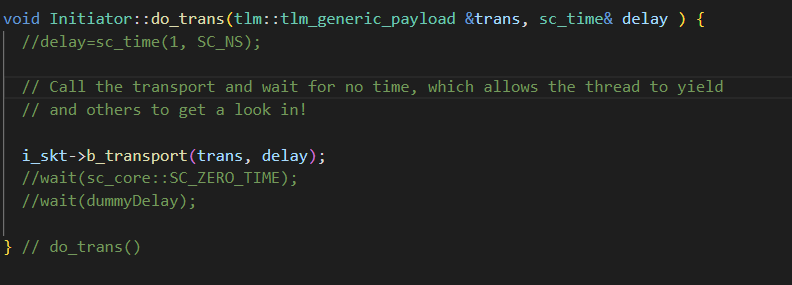
**定義**quantum keeper

****

**b\_transport() 返回的每個延遲值，我們只需通過調用 inc(delay) 來增加本地時間並檢查本地時間是否超過上述量程 (100ns)。 當本地時間達到量程（need\_sync() 為真）時，我們調用 sync() ，它將調用 wait(local time) 和 reset() 本地時間為 0。**

****

**當本地超過100ns後，會進行同步。**

****

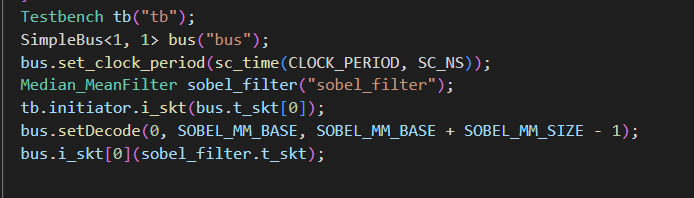
**Initiator.cpp的do\_trans()可以把wait去掉。**

**還有Median\_MeanFilter.cpp的wait都可以拿掉。**

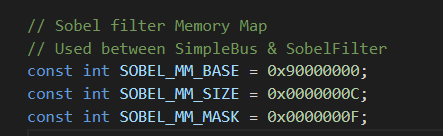
**所以原本每1ns就會call wait，現在是每100ns call wait大幅減少call wait的次數，以達到節省simulation的時間。**

1. **Median and mean filters with TLM interconnect**

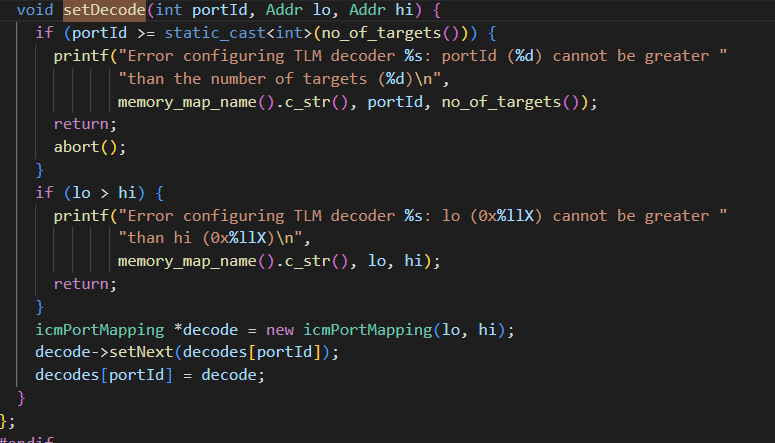
**唯一的區別是，現在 Testbench 通過bus模塊 SimpleBus 向 Median\_MeanFilter 寫入像素和讀取結果。 我們只能關注bus如何將transactions from an initiator to a target module.。**



**在代碼中，我們首先通過“SimpleBus<1, 1>”模板參數實例化一個具有一個target socket和一個initiator socket的bus。 initiator socket通過“tb.initiator.i\_skt(bus.t\_skt[0]);”連接到 Testbench，其中“bus.t\_skt[0]”是bus的第一個（也是唯一一個）initiator socket。 initiator socket通過“bus.i\_skt[0](sobel\_filter.t\_skt);”連接到 Median\_MeanFilter，其中“bus.i\_skt[0]”是bus上的第一個（也是唯一一個）initiator socket。**

****

**記憶體位置和大小**

****

**使用“bus.setDecode()”將全局內存映射地址“SOBEL\_MM\_BASE”設置為端口 ID“0”。 這意味著對該地址的任何blocking transport調用都將被發送到端口 ID 為“0”的initiator socktet。 同樣在將總線blocking transport調用轉發到端口 ID“0”時，將從轉發地址中減去“SOBEL\_MM\_BASE”。 例如，如果我們想通過Bus調用“SOBEL\_MM\_BASE+0x04”處的blocking transport，目標 (Median\_MeanFilter) 將僅接收到“0x04”地址**

****

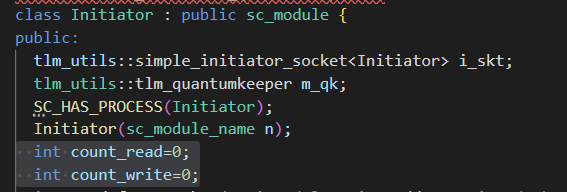
****

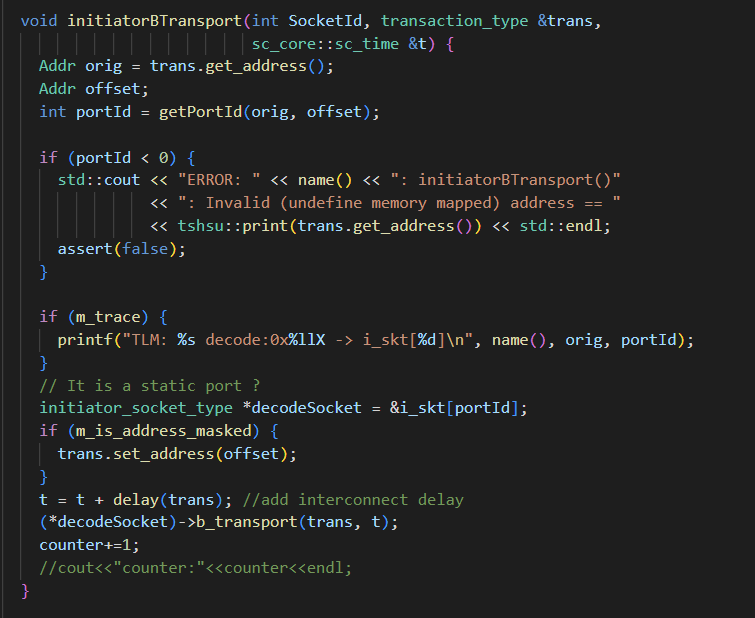
****

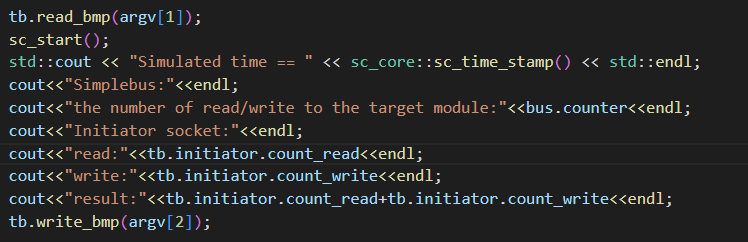
**Initiator的讀寫的地址需是SOBEL\_MM\_BASE+指令地址。**

**下兩圖是計算對目標模塊的read/write次數，我在Simplebus.h的initiatorBTransport放置counter，來計算read/write總數。**

**我還有在initiator.cpp分別在read\_from\_socket和write\_to\_socket放置counter去分別計算read和write的次數來做驗證。**

****

****

****

**最後印出結果。**

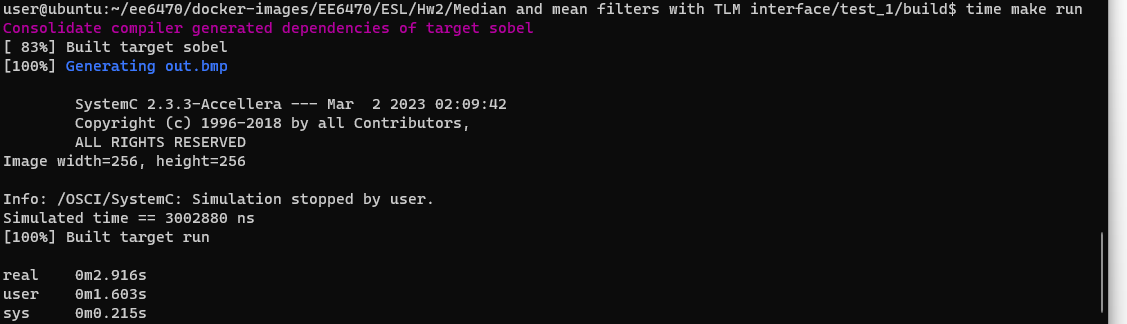
**結果和比較:**



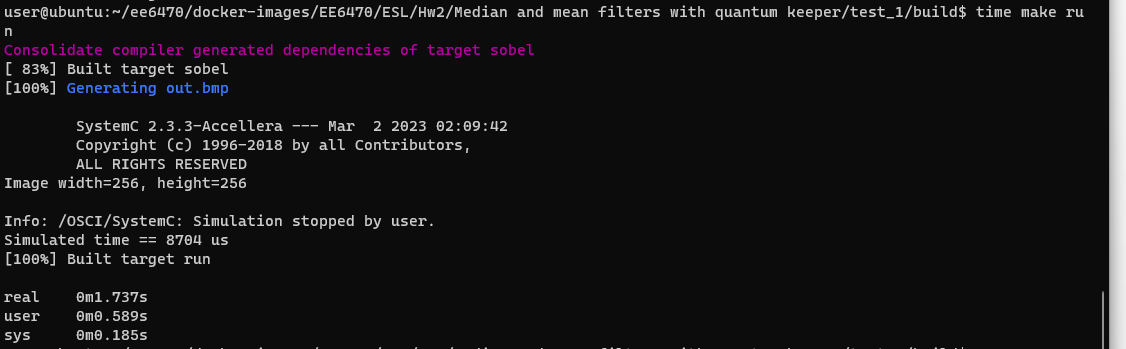


**256\*256的圖片:**

**(1)**

****

**(2)**

****

**(3)**

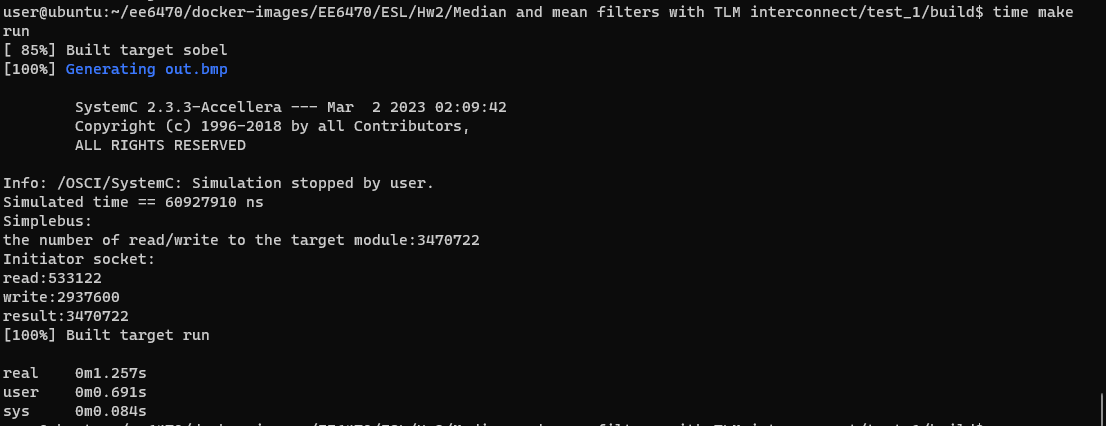
**Simplebus的read/write總和次數為3470722**

**Initiator的 read次數為533122**

**Initiator的 write次數為2937600**

**Initiator的 read0/write次數為3470722**

**兩者一致**

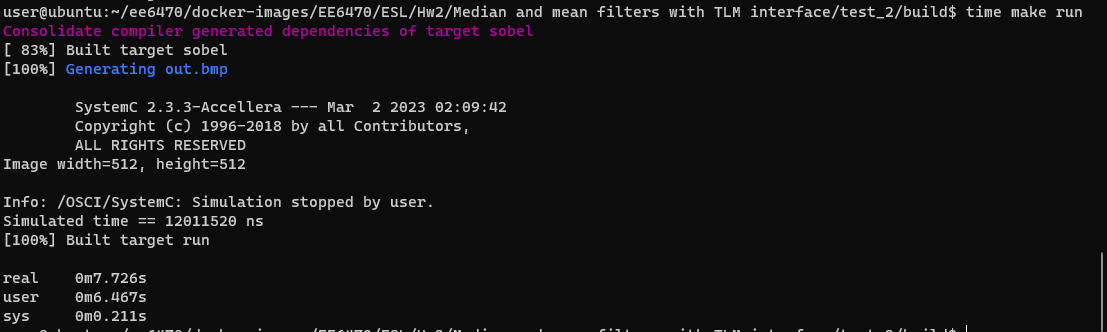
****

**執行時間: (1) > (2) >= (3)**

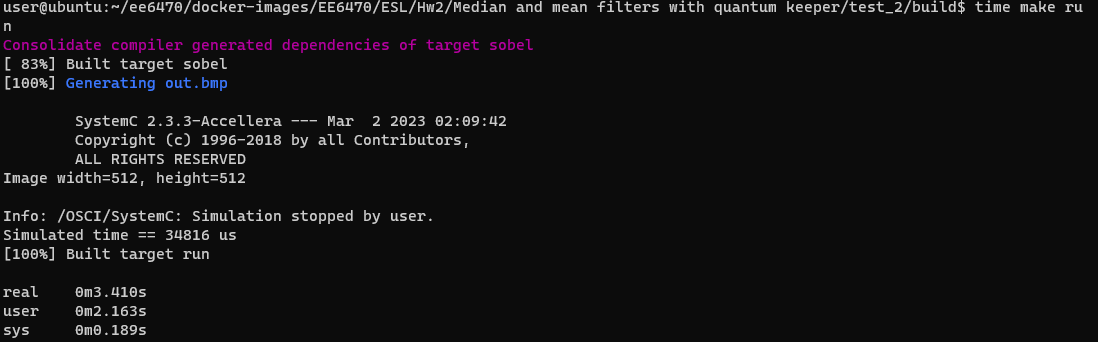
**(1)需要一直call wait，而(2)使用quantum keeper減少了call wait的次數達成減少執行時間的功效，最後(3)使用quantum keeper減少call wait的次數還有指定address。**

**512\*512的圖片:**

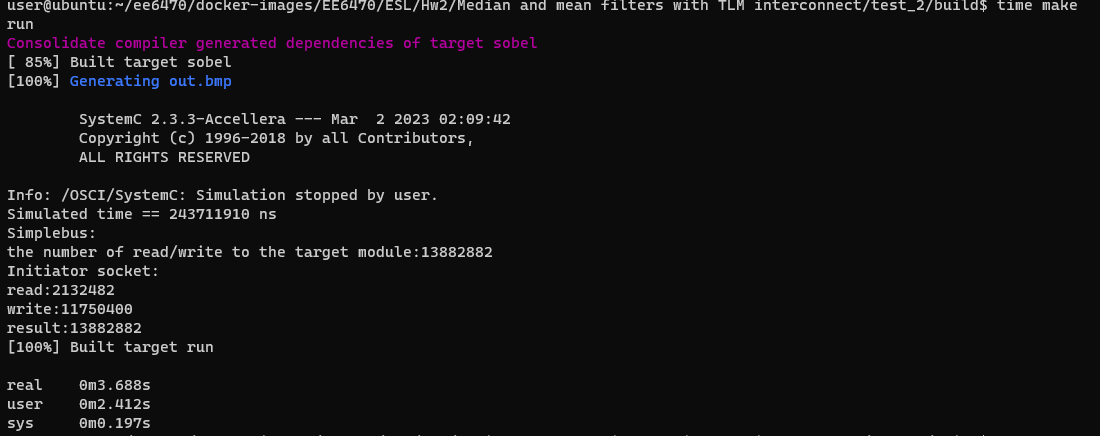
**(1)**

****

**(2)**

****

**(3)**

****

**執行時間: (1) > (2) >= (3)**

**(1)需要一直call wait，而(2)使用quantum keeper減少了call wait的次數達成減少執行時間的功效，最後(3)使用quantum keeper減少call wait的次數還有指定address。**