Jini Intelligent Computing
Workbook of Lab. #3

Preamble

Lab. #3 在 AlveoU50 的 Platform 專案目錄下有四個 Applications 專案目錄：

- vts_Opt1Baseline
- vts_Opt2KernelParallel
- vts_Opt3DataBurst
- vts_Opt4ArrayPartition

以上目錄中皆包含該專案的原始碼檔。

1. Introduction

本實驗為 Vitis OpenCL/XRT 實作，以 Xilinx Alveo U50 PCIe 加速卡為基礎。Xilinx Alveo U50 為 PCIe 介面之 FPGA 加速卡，以 Linux server 為平台透過建置 Xilinx XRT runtime 架構，再以 OpenCL 語言開發 host program，將 bitstream（.xclbin）檔案下載至 Xilinx Alveo U50 加速卡，並運行 host program 的流程控制。

此外本實驗就算沒有 Alveo U50 加速卡，仍然可以在使用者 PC 做到 Software/ Hardware Emulation，Hardware Emulation 的模擬結果與真實在 FPGA 運行相近。

Note：

因 Windows 版本的 Vitis 不支援 Alveo U50 等 PCIe 介面之 FPGA 加速卡，本次實驗將全部在 Linux 系統上實作，若無 Linux PC 亦可在 Windows 上以 Oracle VM VirtualBox 等 virtual machine 運行。採用 VM 請特別注意分配給 VM 的記憶體不要太少，建議分配 8 GB 以上，CPU 也可以多分配一些以加快模擬速度。

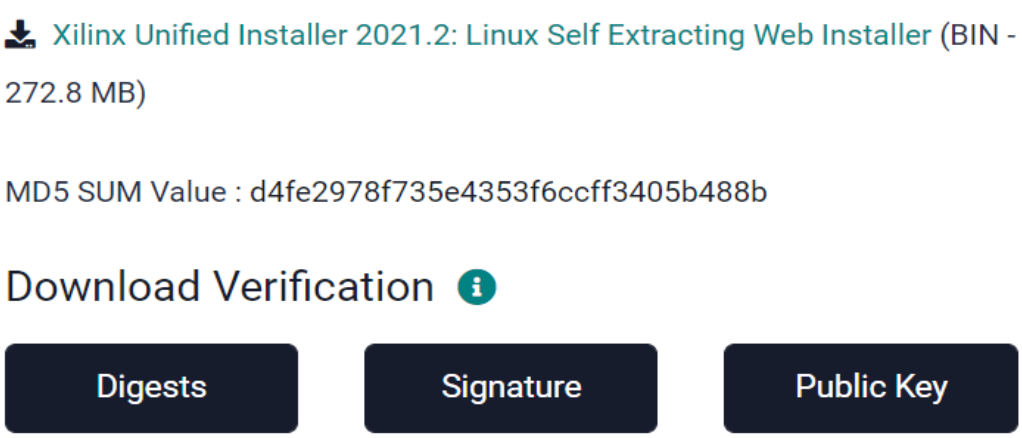
2. Installation

【施作環境為在使用者 PC/laptop/notebook（Linux Base)。】

2.1. Vitis/Vivado/Vitis HLS Software Installation

首先至 Xilinx 官方下載頁面下載 Linux 版本安裝檔：Xilinx Unified Installer 2021.2: Linux Self Extracting Web Installer。

連結網址：https://www.xilinx.com/support/download.html



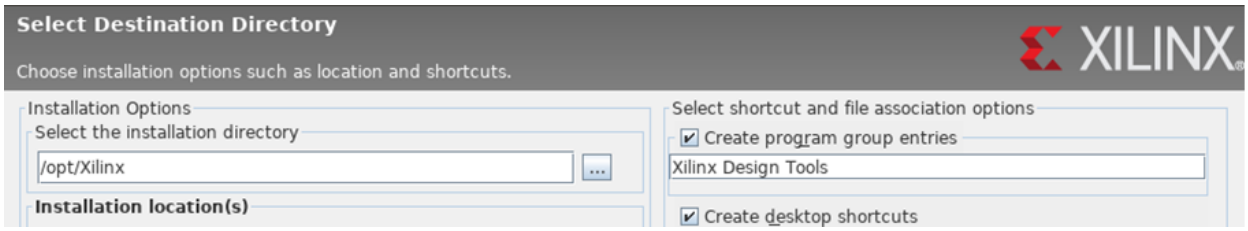
下載完成後，請於檔案資料夾輸入以下指令（[filename]為剛下載好的安裝檔）來將安裝檔更改權限為可執行：

$> chmod +x [filename]

接著執行安裝檔。

$> sudo ./[filename]

安裝過程可參考 Tools Installation Guide 內Windows 版本安裝說明，另外一樣須注意安裝過程需要約 240 GB 的硬碟空間。



安裝完成後輸入以下指令開啟 Vivado License Manager（/opt/Xilinx/

為安裝路徑）：

$> cd /opt/Xilinx/Vitis/2021.2/bin

$> sudo ./vlm

接著同 Tools Installation Guide 中說明匯入下載的 license，若點選 Connect Now

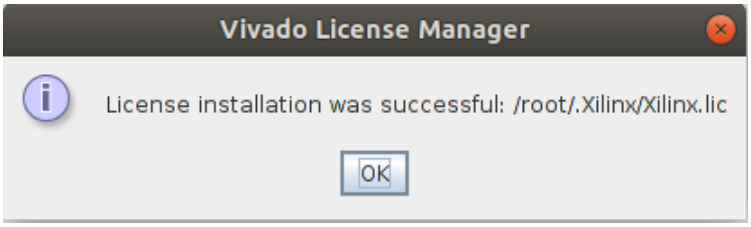
沒有反應請直接上 Xilinx 官網下載 license。

連結網址：

https://www.xilinx.com/support/licensing_solution_center.html

進入後點選 Xillinx Product Licensing Site 並登入下載 license。

匯入成功會看到以下通知。

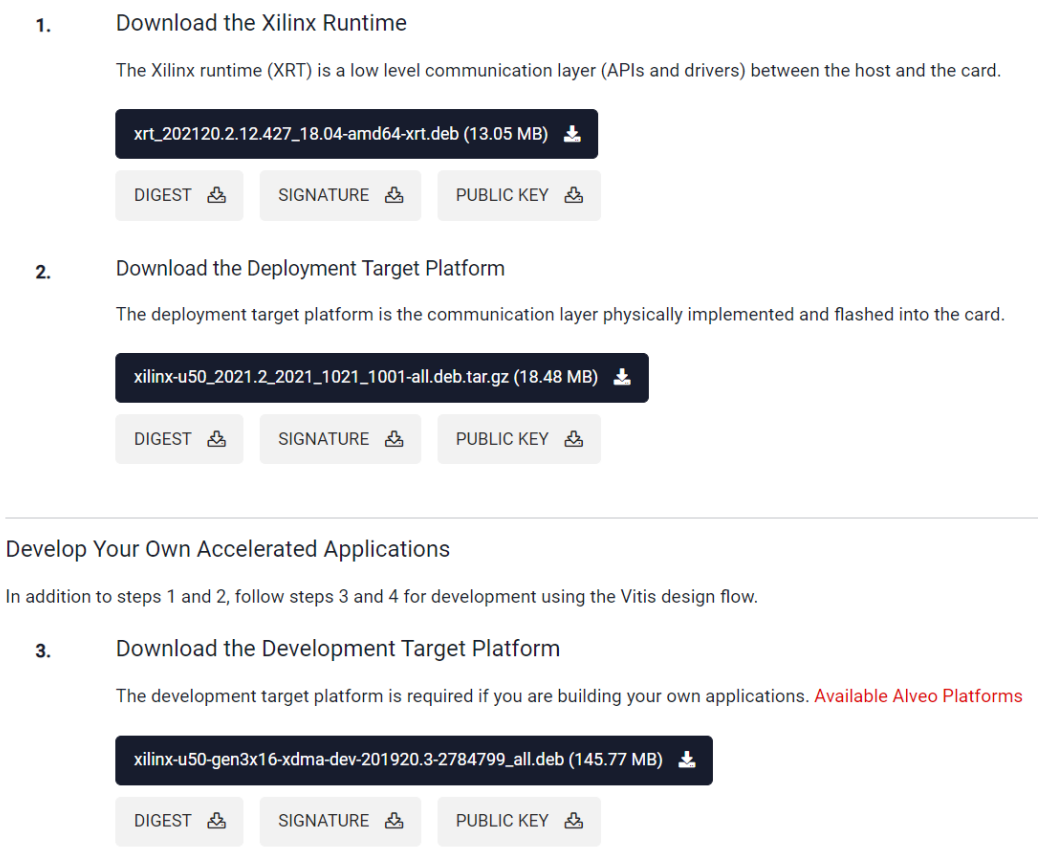


2.2. U50 Platform/XRT Installation

要在Vitis 中使用U50，須安裝U50 platform，請至以下連結下載 package： https://www.xilinx.com/products/boards-and-kits/alveo/u50.html#gettingStarted首先選擇使用的tools 及OS 版本。

下載下方顯示的三個檔案，解壓縮後一一執行安裝，可用以下指令進行安裝：

$> sudo apt install [filename]



2.3. Bash Shell Setting

在Vitis 運行的Linux server 上需要設定環境 bash shell，在個人帳戶

下.bashrc 檔加入環境設定 (nano ~/.bashrc)

```
#source Xilinx Vitis/XRT
source /opt/Xilinx/Vitis/2021.2/settings64.sh
source /opt/xilinx/xrt/setup.sh
```

設定完成後重新開啟 Terminal，確定有順利 source 到。

```
XILINX_XRT        : /opt/xilinx/xrt
PATH              : /opt/xilinx/xrt/bin:/tools/Xilinx/Vitis_HLS/2021.2/bin:/tool
s/Xilinx/Model_Composer/2021.2/bin:/tools/Xilinx/Vitis/2021.2/bin:/tools/Xilinx/
Vitis/2021.2/gnu/microblaze/lin/bin:/tools/Xilinx/Vitis/2021.2/gnu/arm/lin/bin:/
tools/Xilinx/Vitis/2021.2/gnu/microblaze/linux_toolchain/lin64_le/bin:/tools/Xil
inx/Vitis/2021.2/gnu/aarch32/lin/gcc-arm-linux-gnueabi/bin:/tools/Xilinx/Vitis/2
021.2/gnu/aarch32/lin/gcc-arm-none-eabi/bin:/tools/Xilinx/Vitis/2021.2/gnu/aarch
64/lin/aarch64-linux/bin:/tools/Xilinx/Vitis/2021.2/gnu/aarch64/lin/aarch64-none
/bin:/tools/Xilinx/Vitis/2021.2/gnu/armr5/lin/gcc-arm-none-eabi/bin:/tools/Xilin
x/Vitis/2021.2/tps/lnx64/cmake-3.3.2/bin:/tools/Xilinx/Vitis/2021.2/aietools/bin
:/tools/Xilinx/Vivado/2021.2/bin:/tools/Xilinx/DocNav:/home/hplab/bin:/home/hpla
b/.local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/g
ames:/usr/local/games:/snap/bin
LD_LIBRARY_PATH   : /opt/xilinx/xrt/lib:
PYTHONPATH        : /opt/xilinx/xrt/python:
```

2.4. Install Additional Packages

建置專案時需要有 opencl-headers 以及gcc-multilib 套件，輸入以下指令安裝：

$> sudo apt install opencl-headers

$> sudo apt install gcc-multilib

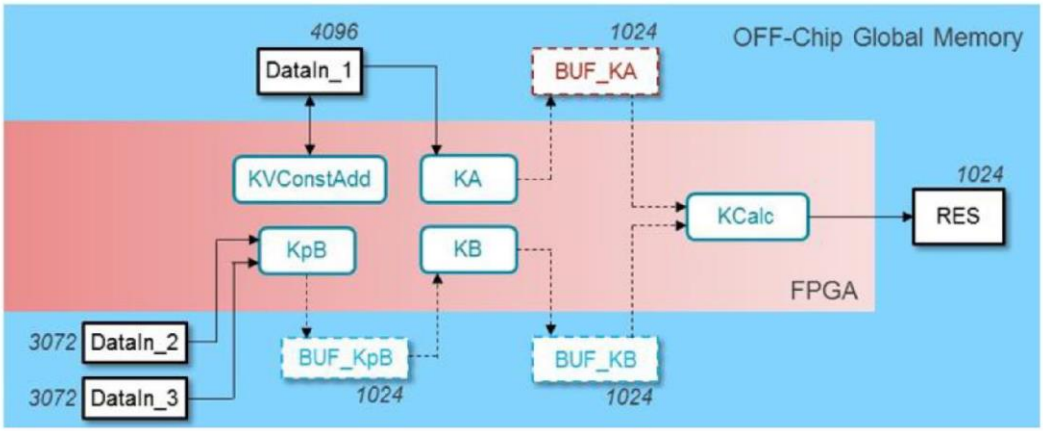
3. Vitis Application Acceleration

【施作環境為在使用者 PC/laptop/notebook (Linux Base)。】

本實驗共有四個專案，分別對應 Baseline、Kernel Parallel、Data Burst 及 Array Partition 四種不同的組態。

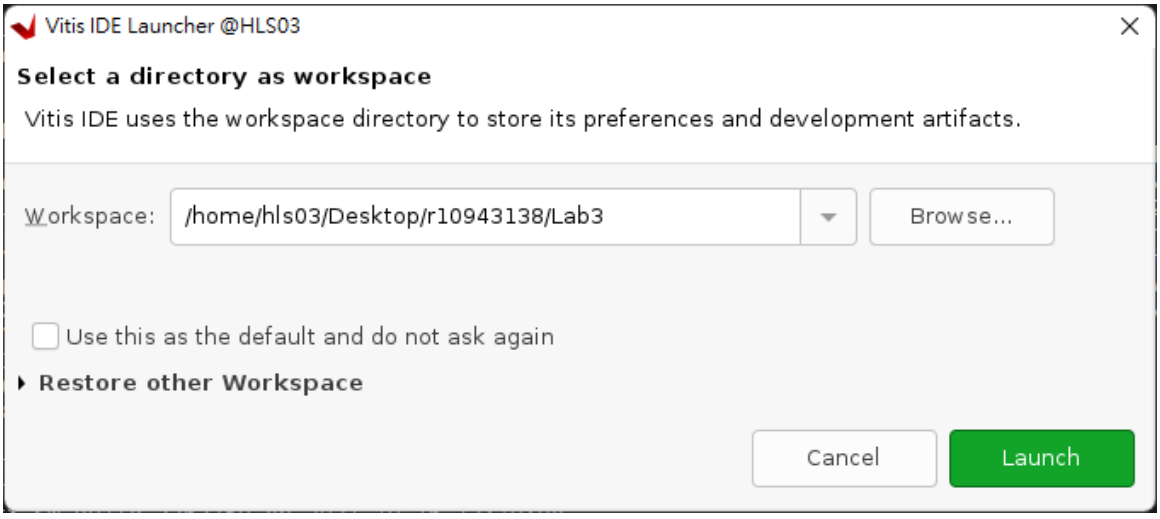
四個組態的實驗步驟皆相同，**請仔細比較各組態 source code 及產生的 Application Timeline 與 Profile Summary 的差異。**下方步驟以 Baseline 作為範例。

下圖為本次實驗的架構，由五個 kernel function 及七個在 global memory 中的 data buffer 組成。

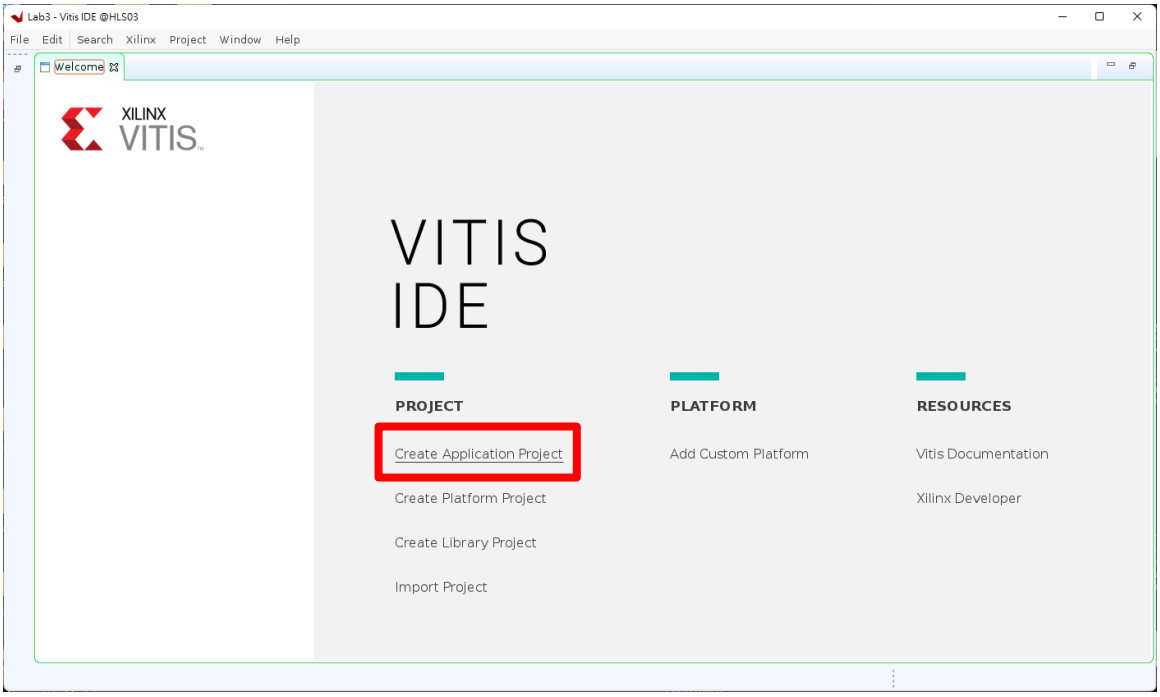


3.1. Create and Setup Project

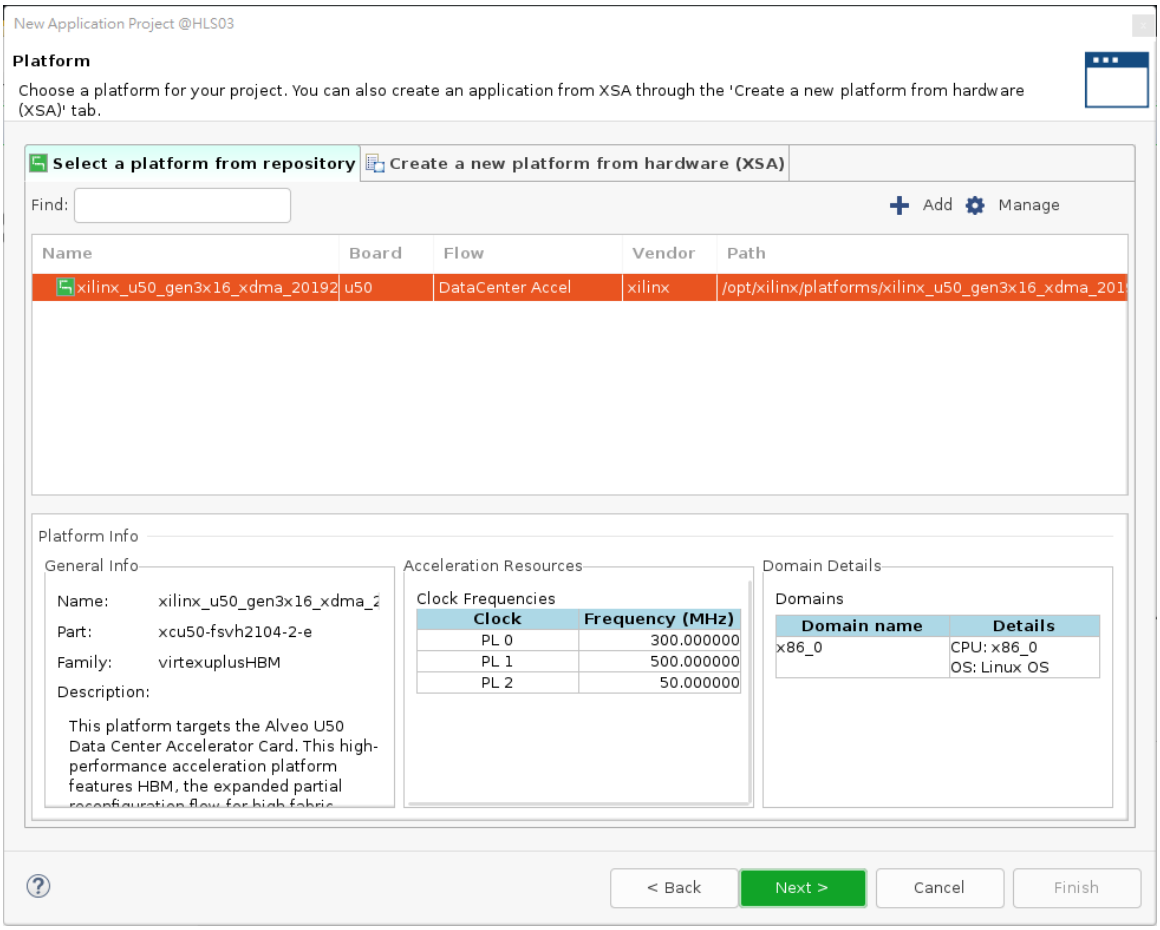
登入先前註冊好的帳號，在 Terminal 輸入 vitis 開啟 Vitis 程式，並設定好運行的 Launch directory。



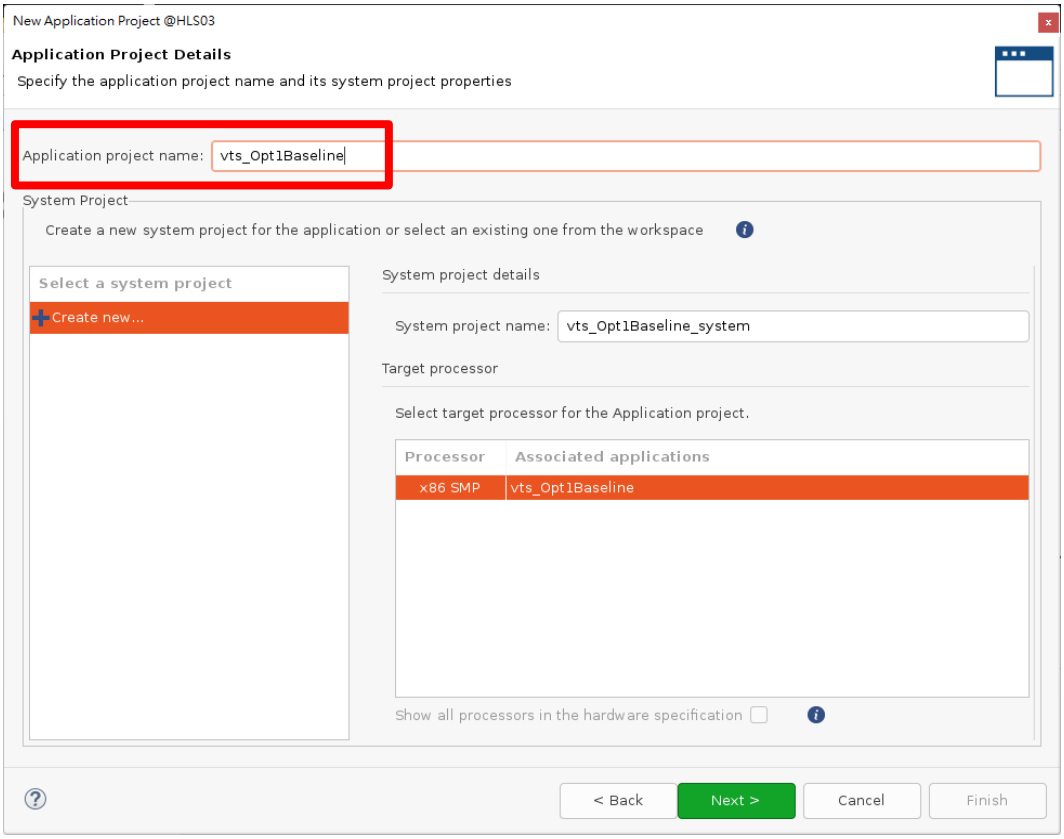
開啟主畫面後，點選 Create Application Project。



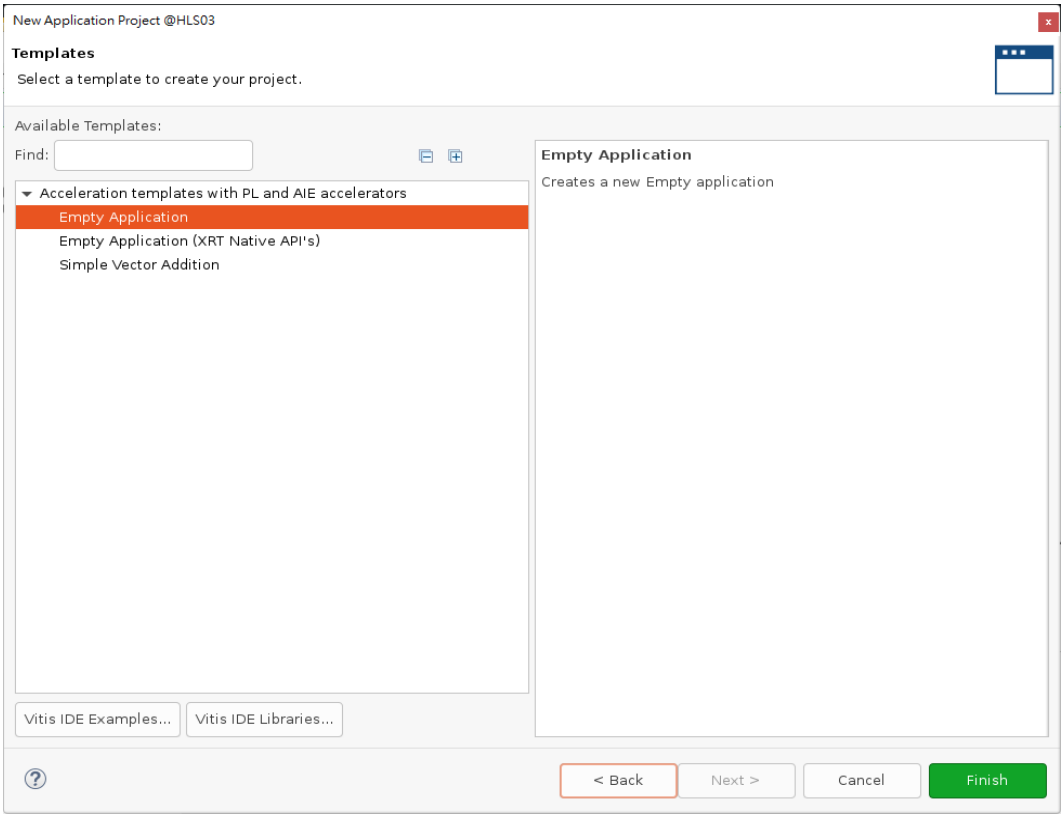
選擇先前安裝好的 U50 platform。

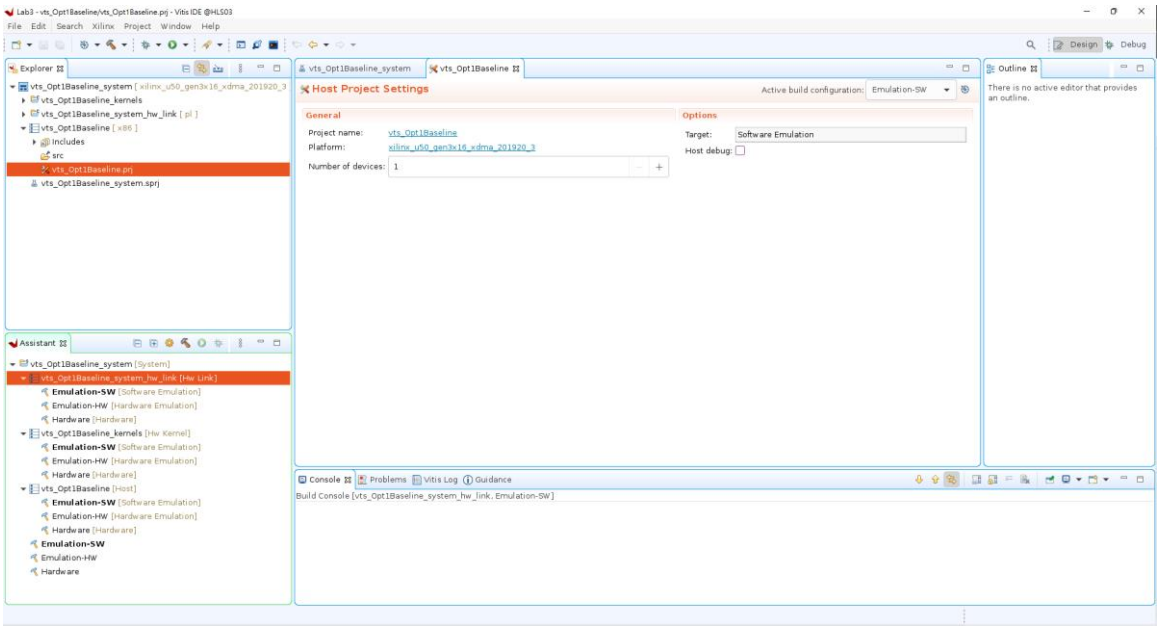


為專案命名，下方 System project name 會自動填上毋須修改。



選擇 Empty Application，點選 Finish 建立專案。





左上方 Explorer 內可看到在 vts_Opt1Baseline_system 專案底下有三個專案：

1. vts_Opt1Baseline_kernels 專案負責 compile kernel function。
2. vts_Opt1Baseline_system_hw_link 專案負責將 kernel link 起來產生 bitstream file（.xclbin）。
3. vts_Opt1Baseline 專案負責 host program 的部分。

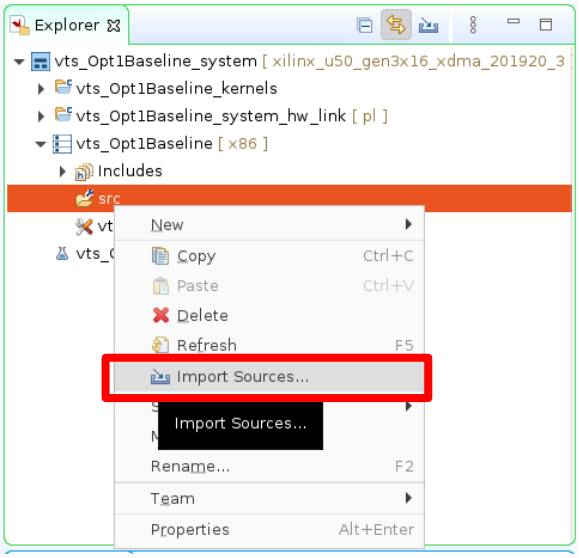
左下方 Assistant 內顯示了各個專案的建置和模擬狀態，以及各項工作產生的 report。

中間的 Project Editor 顯示專案部分屬性，且可以直接對各專案進行設定。

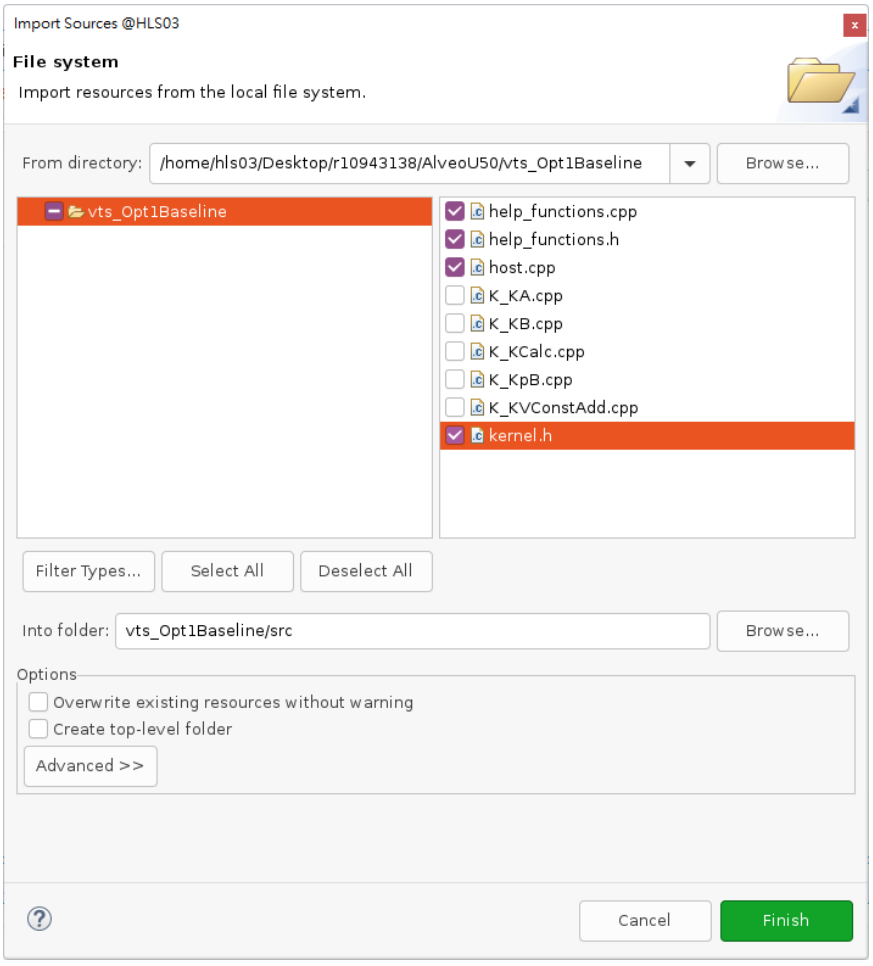
下方有 Console 顯示工作狀態，且可以在各專案不同組態的 console 間切換。

建立好專案後第一步要在專案裡加入 source code。

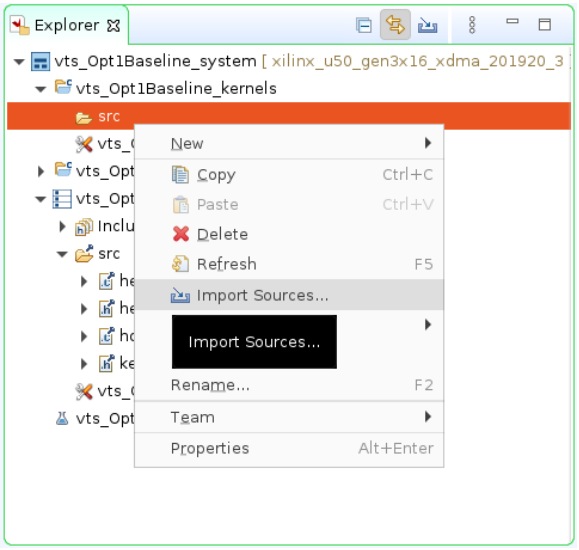
首先在 Explorer 中右鍵點選 vts_Opt1Baseline 專案底下的 src 資料夾，點選 Import Sources 加入 host program 的 source code。



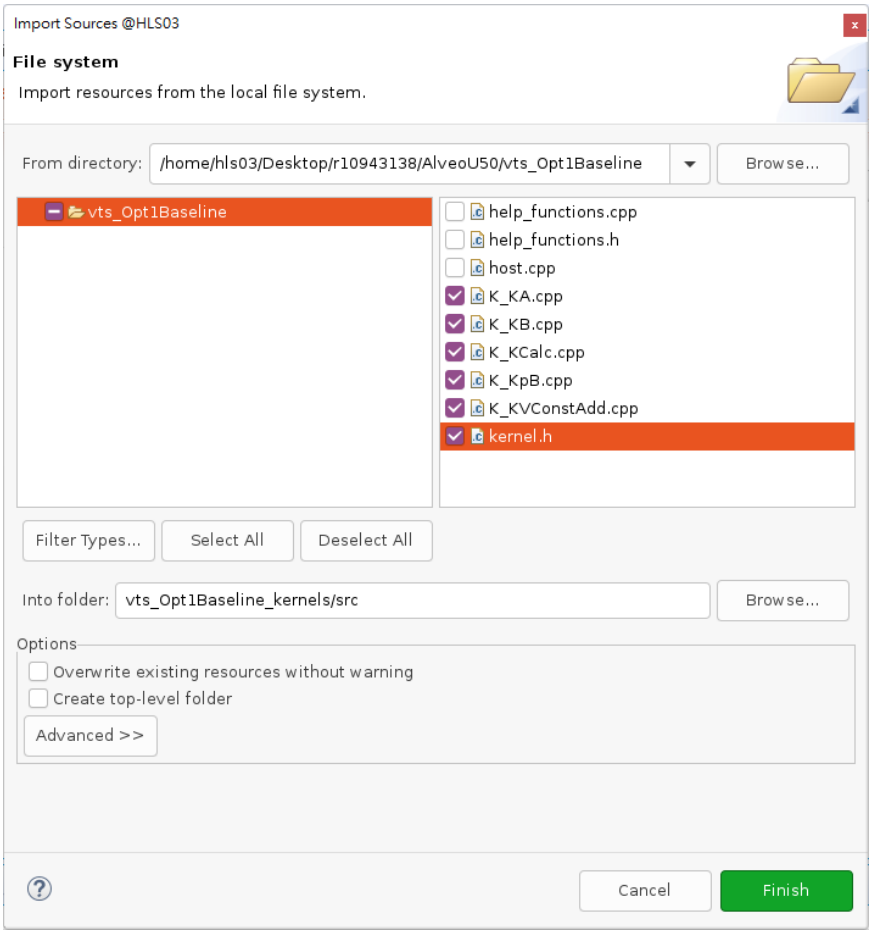
選擇提供的 source code 資料夾， 勾選 **help_functions.cpp**、**help_functions.h**、**host.cpp** 以及 **kernel.h**。



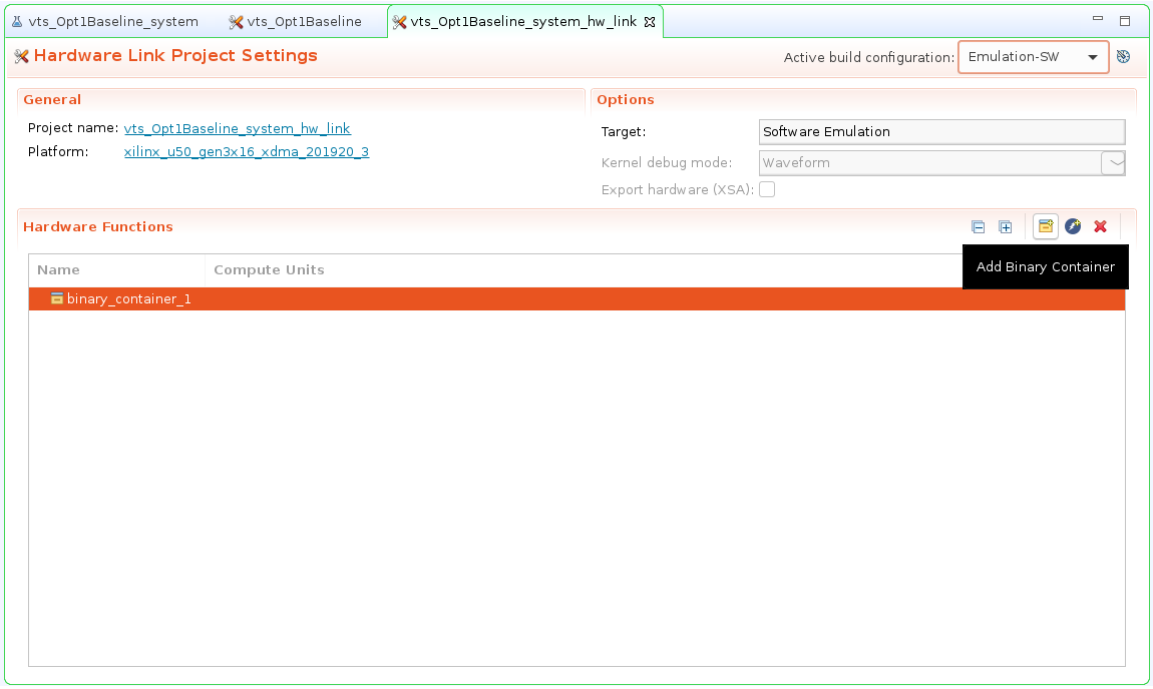
接著在 vts_Opt1Baseline_kernels 專案底下加入 kernel function 的 source code。



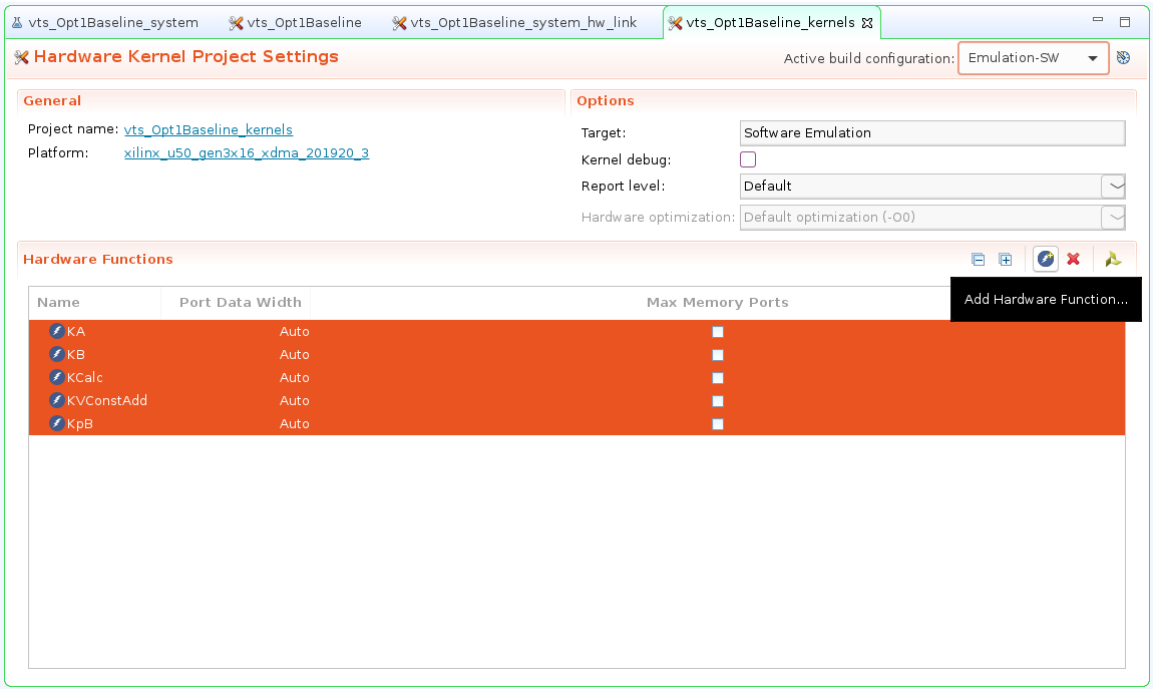
選擇提供的 source code 資料夾， 勾選 **K_KA.cpp、K_KB.cpp、K_Kcalc.cpp、K_KpB.cpp、K_KVConstAdd.cpp 以及 kernel.h**。

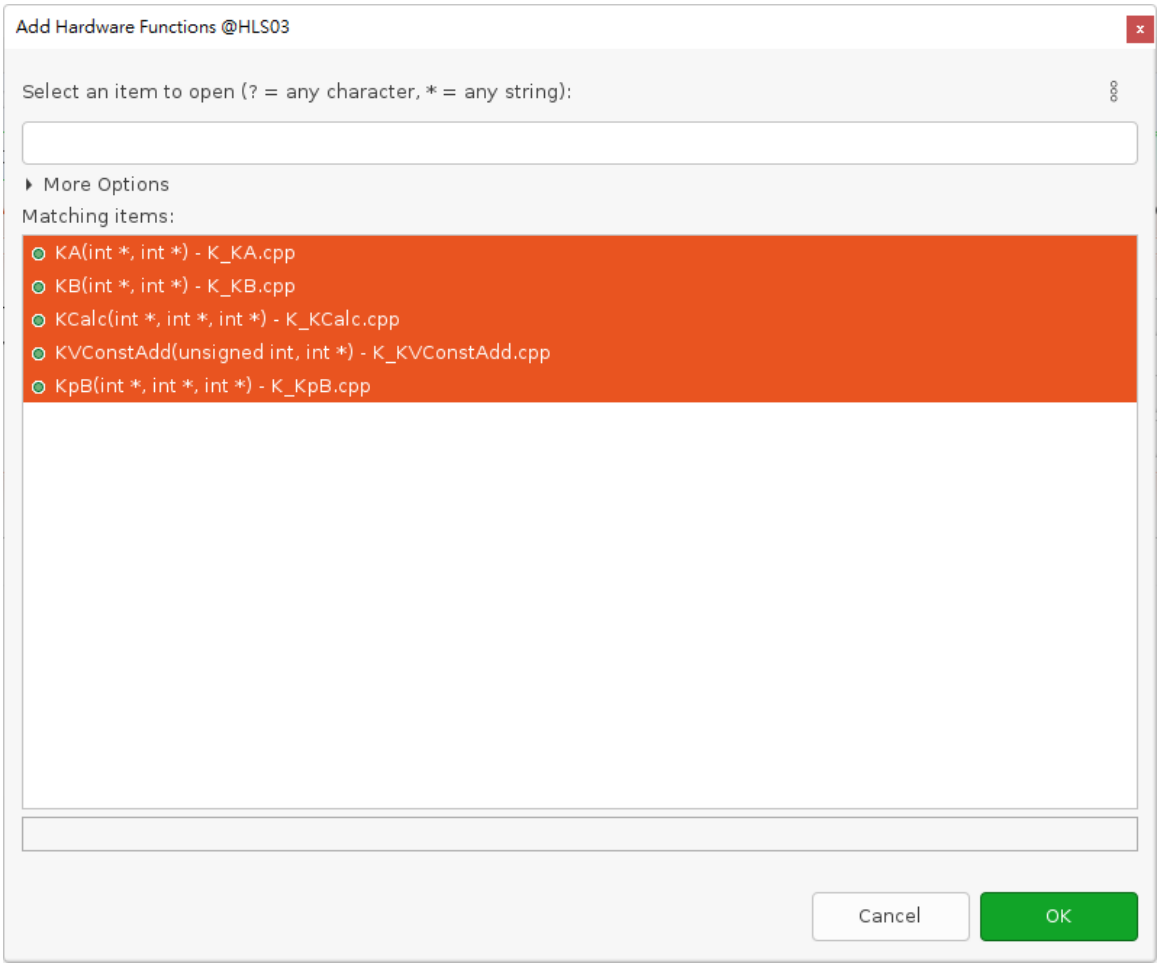


下一步要加入 binary container，打開 vts_Opt1Baseline_system_hw_link 專案(雙擊 hw_link.prj)，在 Project Editor 點選 Add Binary Container 後，底下會出現 binary_container_1。



最後要加入 hardware functions，打開 vts_Opt1Baseline_kernels 專案(kernels.prj)，在 Project Editor 點選 Add Hardware Functions，並在彈出的視窗中選擇 KA、KB、KCalc、KVConstAdd 以及 KpB 加入。完成後底下會列出所有 kernel functions。





3.2. Software Emulation

Software emulation 是以軟體函式形式直接傳遞引數來模擬結果，類似於 Lab. #1 中的 C simulation。

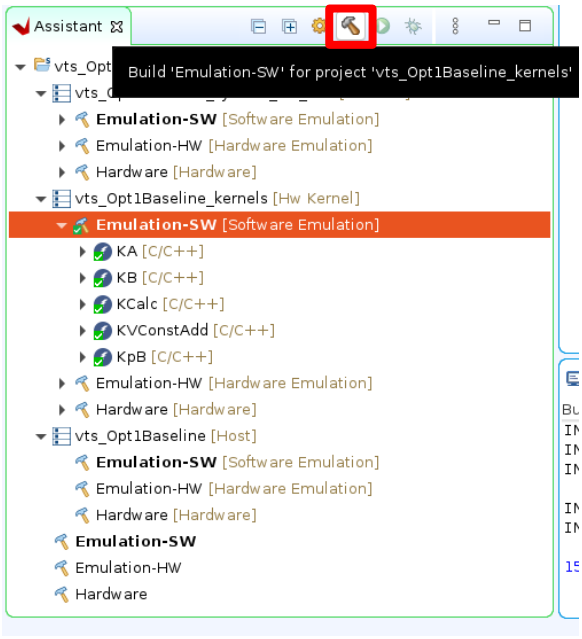
3.2.1. Build Project

要執行 emulation 前要先建置專案，產生模擬需要的執行檔及 bitstream file。

IMPORTANT：

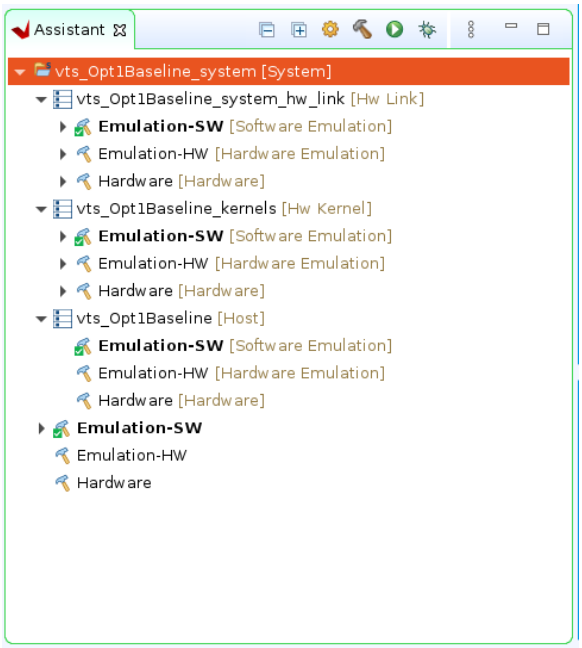
必須依照 kernel → hw_link → host → system 的順序來建置專案！

首先在 Assistant 中選擇 kernel 專案底下的 Emulation-SW，接著按下上方 Build project 按鈕進行建置。



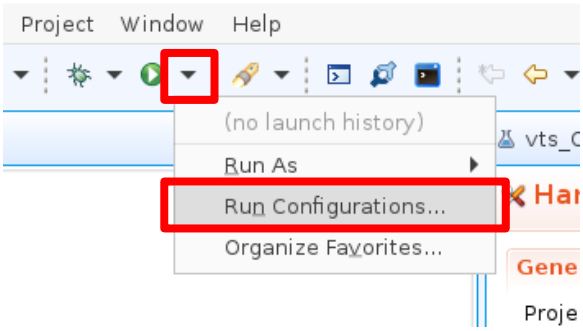
kernel 建置完成後再依前述順序點選其他專案的 Emulation-SW 並進行建置，system 的部分請直接點選 vts_Opt1Baseline_system[System]來建置。

完成建置後 Assistant View 顯示如下，建置成功會有綠色打勾標示。

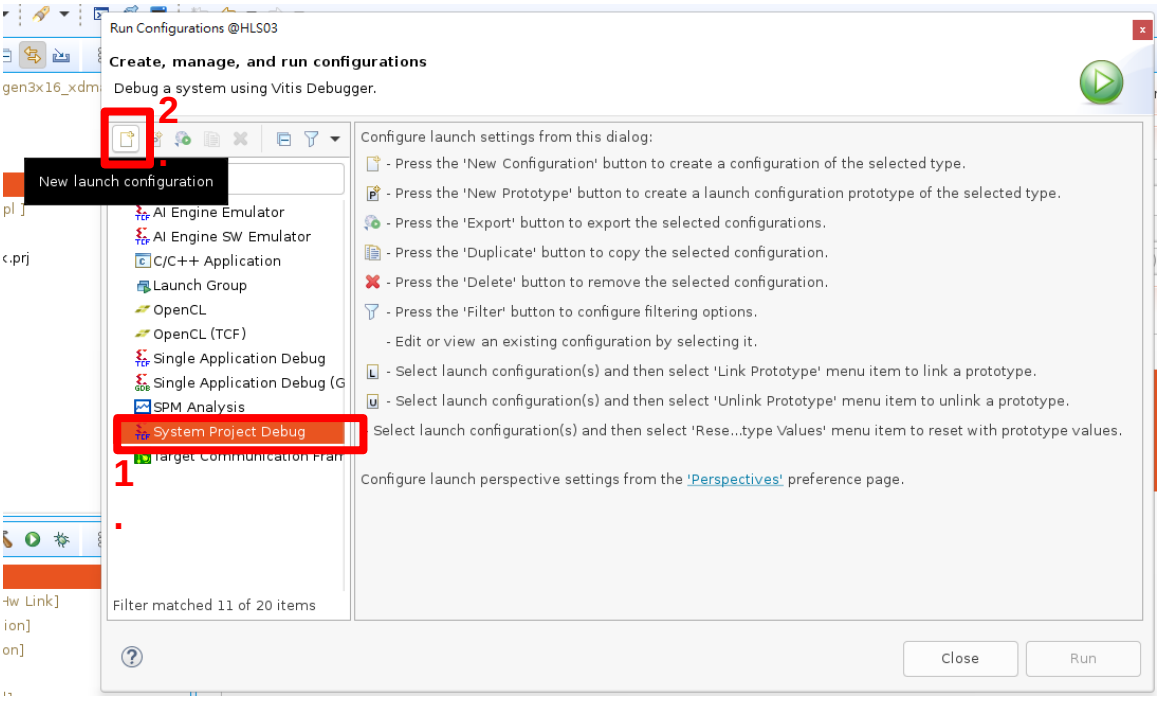


3.2.2. Run Emulation

執行 emulation 前要先設定其組態。點選 Run Configurations



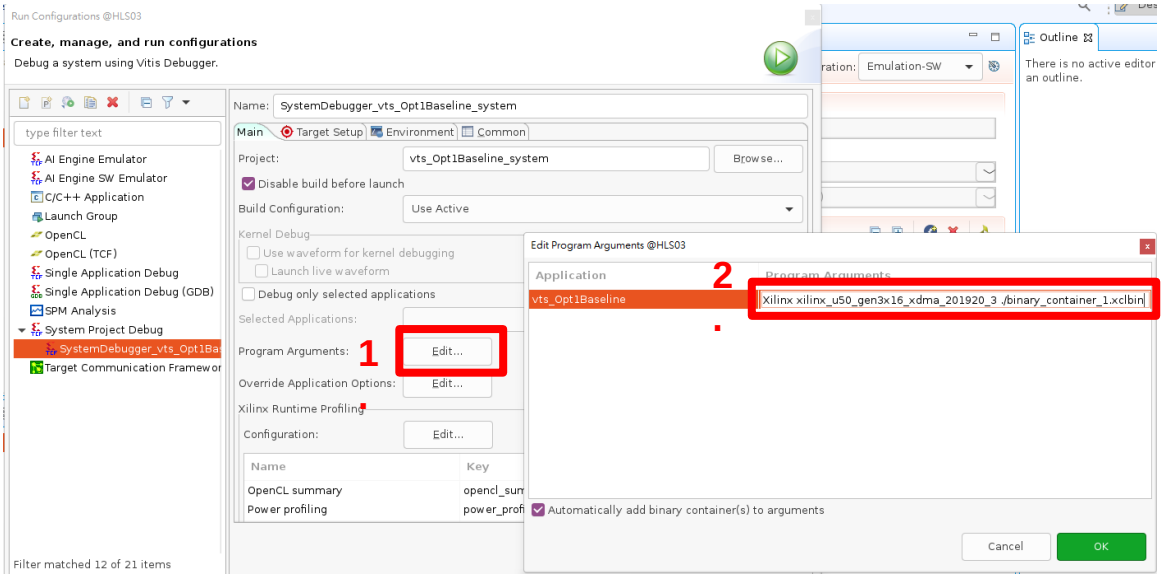
新增一個 System Project Debug 組態。



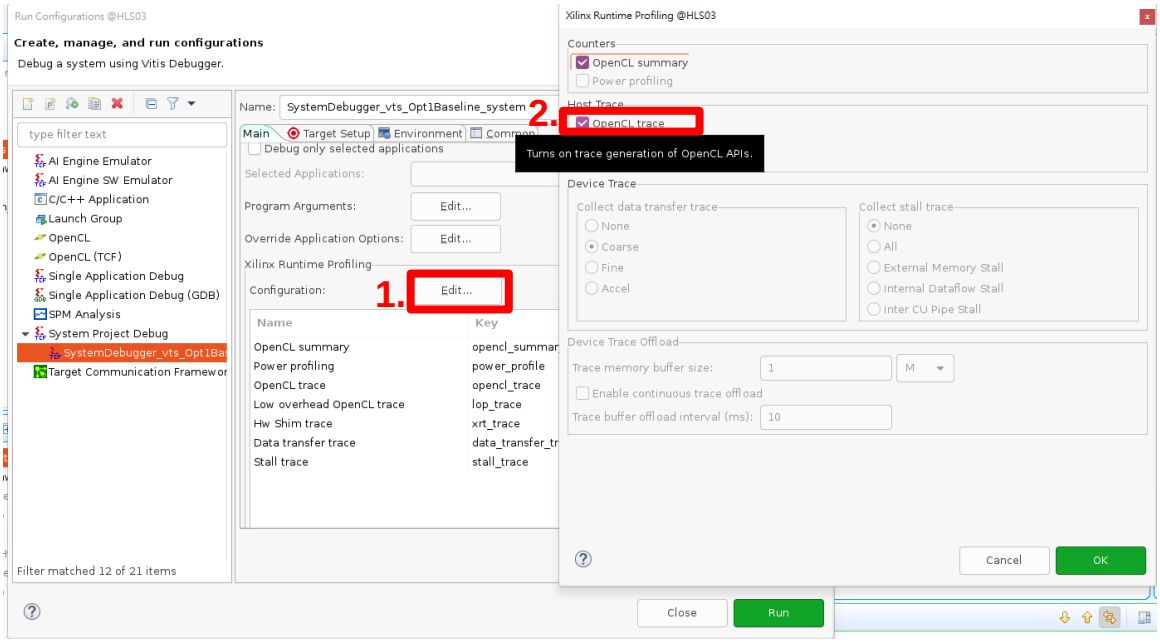
編輯 Program Arguments，提供 host program 所需的三個 arguments：

Xilinx xilinx_u50_gen3x16_xdma_201920_3 ./binary_container_1.xclbin

(用空白隔開這三項)

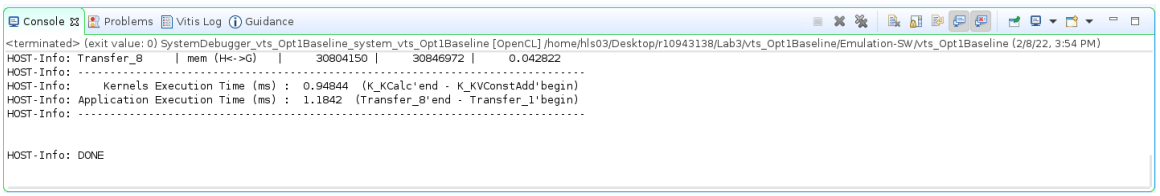


編輯 Xilinx Runtime Profiling 的 Configuration，設定為 OpenCL summary and Open CL trace。



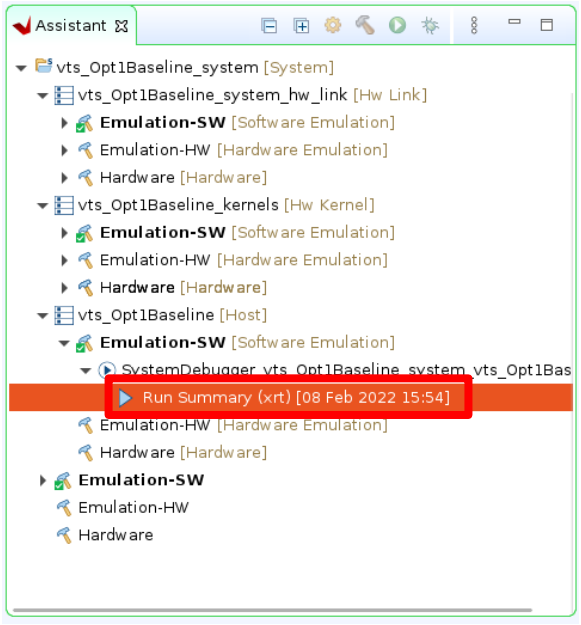
點選 Apply，再點選 Run 開始執行 software emulation。

Emulation 完成後，會在 Console 看到 DONE 訊息。

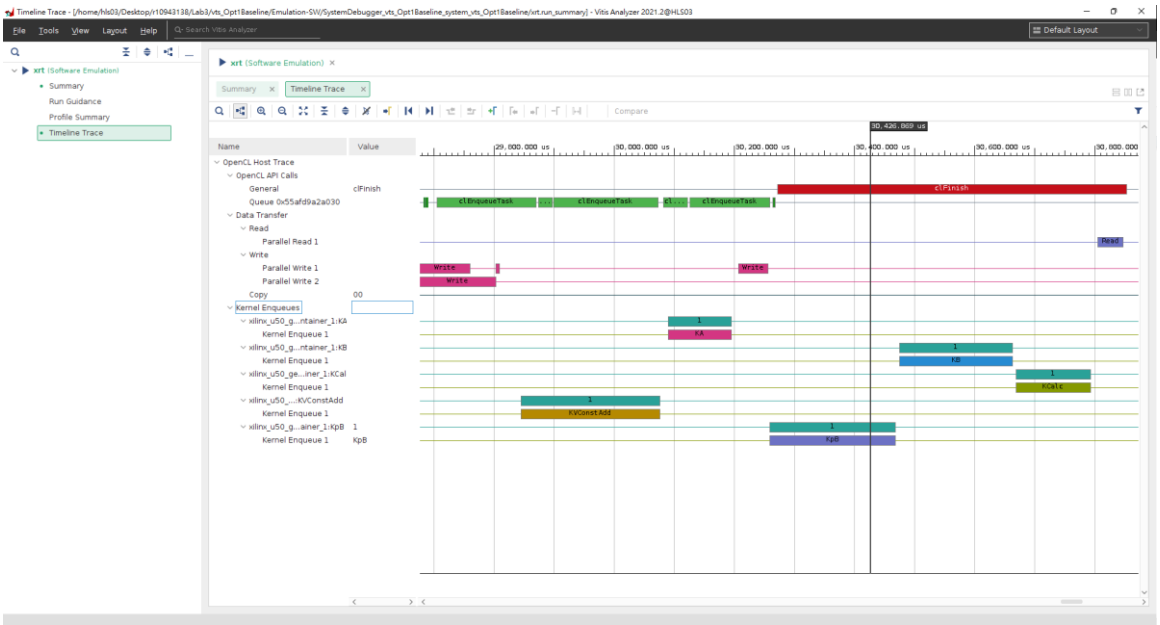


3.2.3. Analysis

Emulation 執行完畢後，在 Assistant View 的 vts_Opt1Baseline 專案底下的 Emulation-SW 中會產生一個 Run Summary，可以雙擊打開 Vitis Analyzer 查看各項 report 進行分析。



在 Vitis Analyzer 中，點選 Application Timeline 可查看 host program 以及 kernel 運行的時序。



3.3. Hardware Emulation

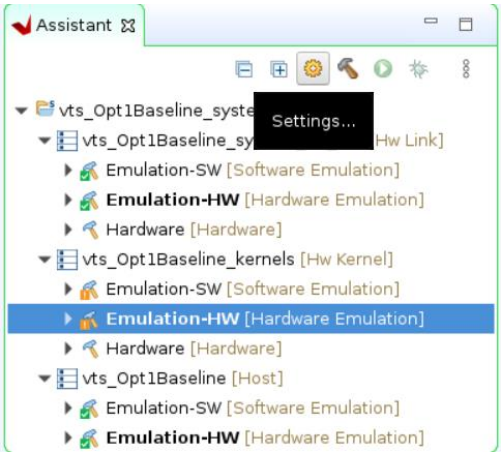
Hardware emulation 是以軟體模擬 XRT runtime 到 kernel FPGA 的行為，類似於 Lab. #1 中的 Co-simulation。

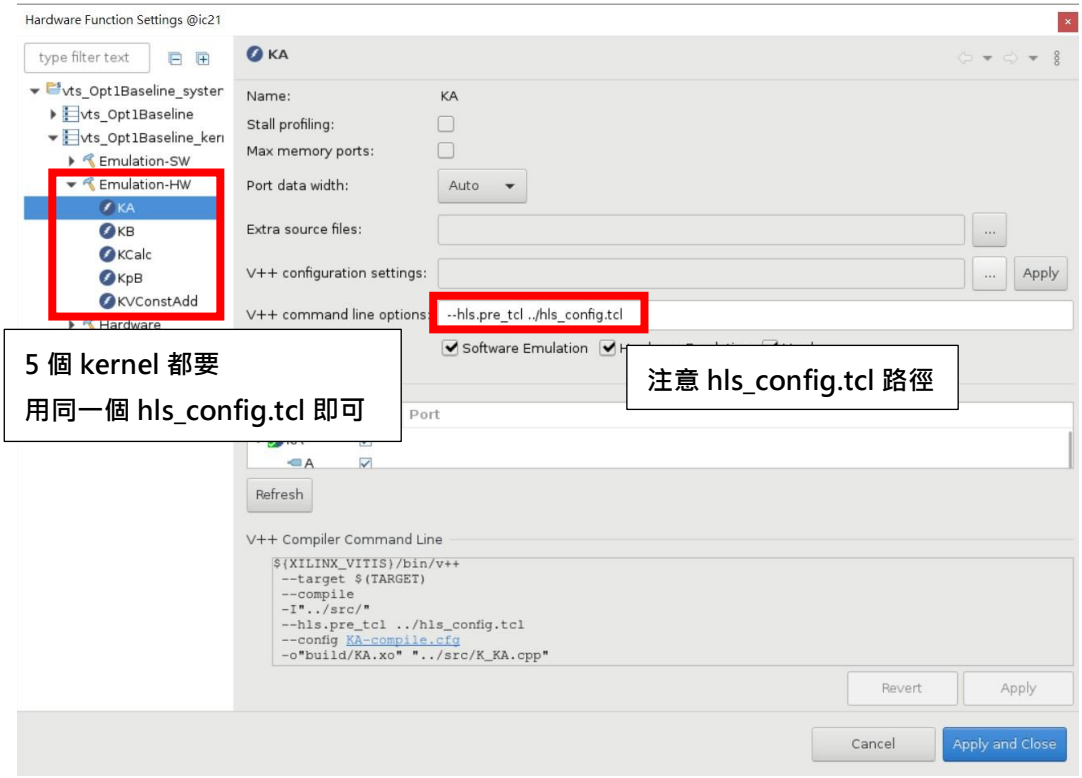
3.3.1. Build Project

步驟同 Software Emulation，請參考前述步驟，改為點選 Emulation-HW。過程視電腦配備可能需要數十分鐘。

(kernel → hw_link → host → system)

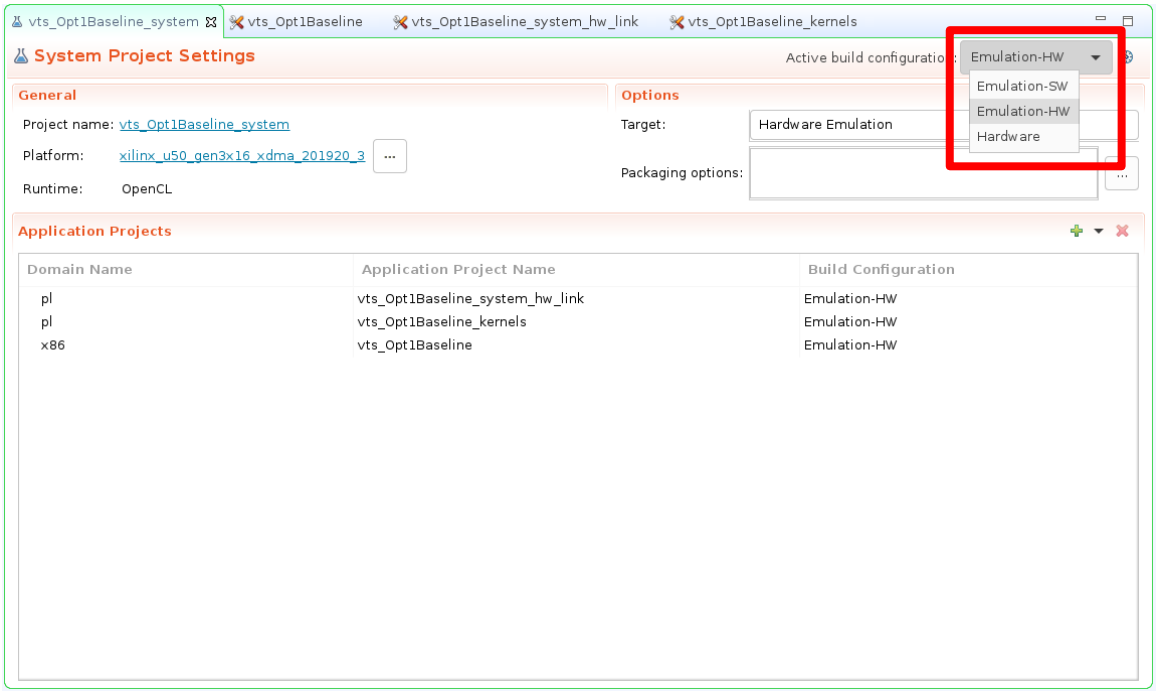
※在 build hw_link 時，在 setting 加上 V++ 參數





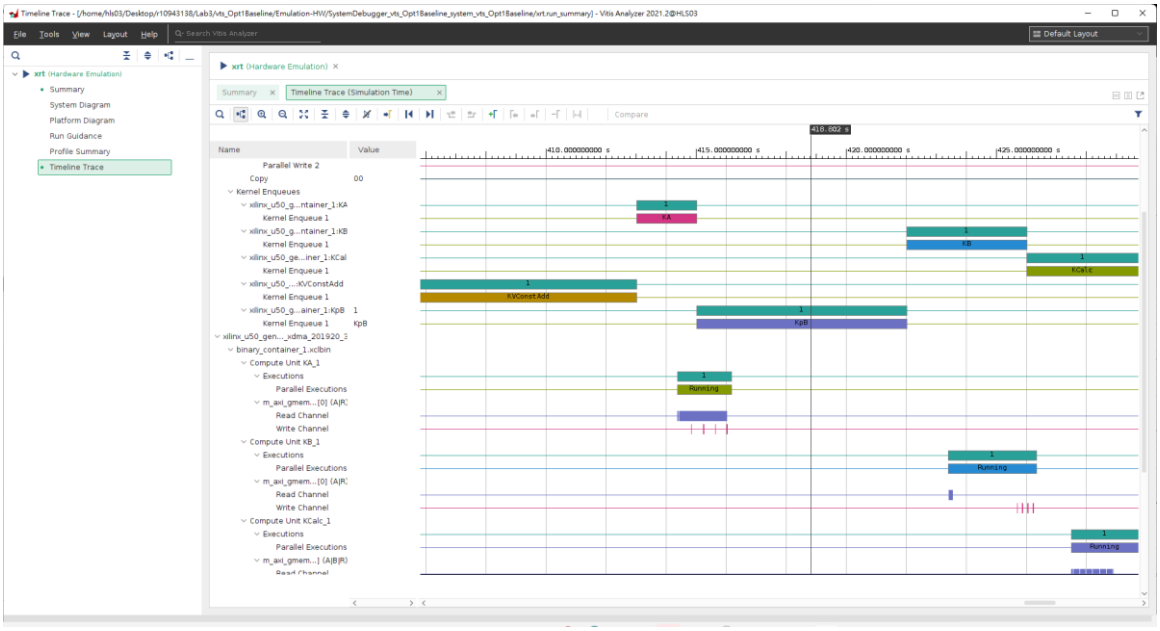
3.3.2. Run Emulation

請將 Active build configuration 設定為 Emulation-HW，即可直接使用相同的 Run Configuration 毋須修改，其餘步驟與 Software Emulation 相同，請參考前述步驟。過程視電腦配備可能需要數十分鐘。



3.3.3. Analysis

步驟同 Software Emulation，請參考前述步驟，打開在 Emulation-HW 中的 Run Summary 查看各項 report 進行分析。


Timeline Trace - [/home/hls03/Desktop/r10943138/Lab3/vts_Opt1Baseline/Emulation-HW/SystemDebugger_vts_Opt1Baseline_system_vts_Opt1Baseline/xrt.run_summary] - Vitis Analyzer 2021.2@HLS03
File
Tools
View
Layout
Help
Search Vitis Analyzer
Default Layout
xrt (Hardware Emulation)
Summary
System Diagram
Platform Diagram
Run Guidance
Profile Summary
Timeline Trace
xrt (Hardware Emulation)
Summary
Timeline Trace (Simulation Time)
Compare
Name
Value
410.602 s
410.000000000 s
415.000000000 s
420.000000000 s
425.000000000 s
Parallel Write 2
Copy
00
Kernel Enqueues
xilinx_u50_g...ntainer_1:kA
Kernel Enqueue 1
xilinx_u50_g...ntainer_1:kB
Kernel Enqueue 1
xilinx_u50_ge...iner_1:KCal
Kernel Enqueue 1
xilinx_u50_...:kVConstAdd
Kernel Enqueue 1
xilinx_u50_g...ainer_1:KpB
1
Kernel Enqueue 1
KpB
xilinx_u50_gen-..._xdma_201920_3
binary_container_1.xclbin
Compute Unit KA_1
Executions
Parallel Executions
m_axi_gmem...[0] (A|R
Read Channel
Write Channel
Compute Unit KB_1
Executions
Parallel Executions
m_axi_gmem...[0] (A|R
Read Channel
Write Channel
Compute Unit KCalc_1
Executions
Parallel Executions
m_axi_gmem...] (A|B|R)
Read Channel
1
KA
KB
KCalc
kVConstAdd
KpB
Running