Vitis HPC Library: Multi-Layer Perceptron (MLP)

Lab #C Vitis library

107012045 郭柏辰 2022 / 5 / 1

一、Background introduction

1. HPC library

   High-Performance Computation: provides an acceleration library for applications with high computation workload. This library depends on the Xilinx BLAS and SPARSE library to implement some components.[1]

2. Multi-Layer Perceptron (MLP) introduction

   其實 MLP 就是一個 fully connected 的 feedforward 人工神經網路 (ANN)[2]。多層感知器廣義來說就是如前面的定義，狹義來說是定義一個多層網路架構並且包含 threshold activation function 的網路架構。通常一個 MLP 最少需要有 3 層。包含一個 input layer，一個 hidden layer，與一個 output layer。其架構大概如下圖。圖示為一個 FCN 的架構。這邊每一層使用到的 fcn function 為$C = sigm(A * B + Xbias)$，其中 A 和 B 是輸入矩陣，X 為 Bias 矩陣，而輸出為 C 矩陣，這邊在後面定義 fcn 的架構會講到。

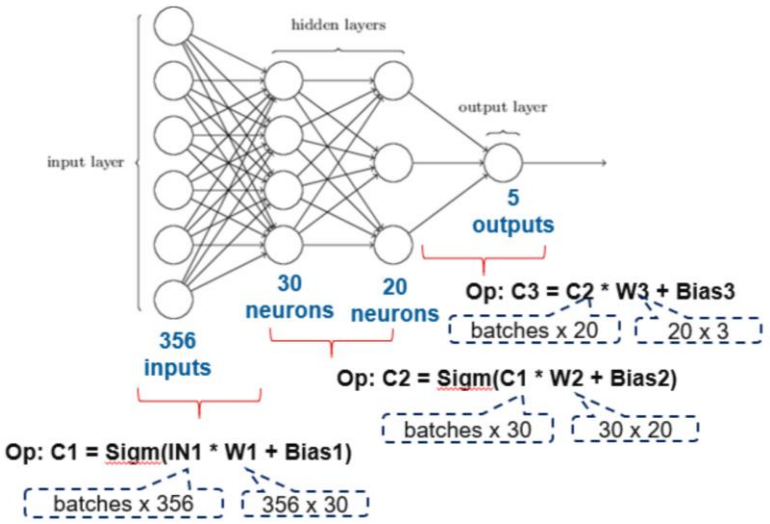


Figure 1. FCN example [3]

[1] https://xilinx.github.io/Vitis_Libraries/hpc/2020.2/index.html
[2] https://en.wikipedia.org/wiki/Multilayer_perceptron
[3] FCN : https://xilinx.github.io/Vitis_Libraries/hpc/2021.2/user_guide/L1/mlp_intr.html

3. Why need acceleration?
   因為 MLP 中有大量高複雜度的矩陣乘法計算，以及 MACs 的使用，這些都是高平行度的資料，因此若是使用傳統 CPU 計算的話，會消耗相當多的 cycle 在做運算以及需要更多的 buffer 來做儲存，其資料流存取量會變得相當多，導致效能變差。因此將 MLP 做加速。

4. What function inside MLP?
   其中一個最重要在 fully connected network 之中會用到的$C = sigm(A * B + Xbias)$，這邊是直接使用 BLAS library(basic linear algebra subroutines)定義的 GEMM kernels (General matrix multiply)，將 A、B 輸入矩陣相乘之後再加上 X bias 矩陣。其架構如下圖，主要由 data mover, transpose 和 buffer 以及 systolic array 完成，其中 systolic array 是一個脈動陣列，讓輸入與輸出資料只在邊界的乘法加法器(MAC)，核心概念就是讓數據在運算單元(PE)的陣列中進行流動，減少訪存的次數，並且使得結構更加規整，佈線更加統一，提高頻率[4]。主要作用就是平衡 I/O 的讀寫。

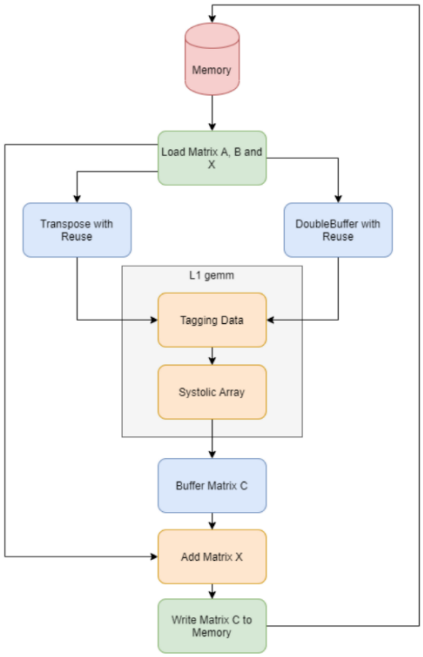


Figure. GEMM architecture

[4] https://www.twblogs.net/a/5b877cc82b71775d1cd76ad3

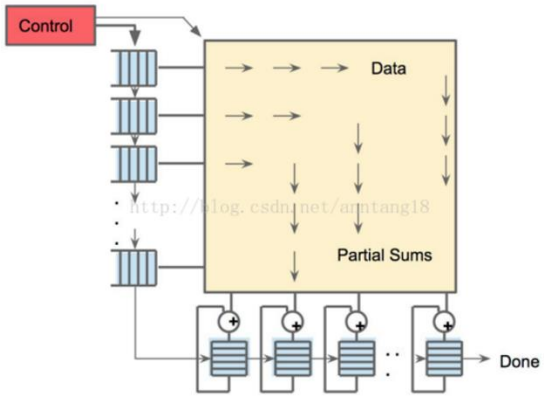


Figure. systolic array architecture

```
namespace blas {

/**
 * @brief Gemm class, implement C = A*B+X
 * t_aColMemWords defines number of memwords in the columns of one row of buffer_A. Due to the reusability, the
 height of buffer_A is only one memwords. For buffer_B, t_aColMemWords defines number of memwords in the rows of one
 column in buffer_B, t_bColMemWords defines number of memwords in the cols of one row in buffer_B. t_aRowMemWords and
 t_bColMemWords define the height and width of buffer_C in terms of memwords.
 *
 * @tparam t_FloatType matrix A, B entry data type
 * @tparam t_XDataType matrix X entry data type
 * @tparam t_DdrWidth number of matrix elements in one memory word
 * @tparam t_XDdrWidth number of matrix X elements in one memory word
 * @tparam t_aColMemWords  number of memory words in one row of the matrix A buffer
 * @tparam t_aRowMemWords  number of memory words in one column of the matrix A buffer
 * @tparam t_bColMemWords   number of memory words in one row of the matrix B buffer
 *
 */
template <typename t_FloatType,    // matrix A, B entry data type
          typename t_XDataType,    // matrix X entry data type
          unsigned int t_DdrWidth, // number of matrix elements in one memory word
          unsigned int t_XDdrWidth,
          unsigned int t_aColMemWords = 1, // number of memory words in one row of the matrix A buffer
          unsigned int t_aRowMemWords = 1, // number of memory words in one column of the matrix A buffer
          unsigned int t_bColMemWords = 1  // number of memory words in one row of the matrix B buffer
          >
```

Figure. gemmKernel.hpp partial code

這邊的 kernel 在輸入矩陣已有做相當多的優化，如 pipeline…等。

```
90      *
91      * @param l_aAddr  the base address of matrix A in external memory
92      * @param l_aColBlocks the No. blocks along matrix A cols
93      * @param l_aRowBlocks the No. blocks along matrix A rows
94      * @param l_bColBlocks the No. blocks along matrix B cols
95      * @param l_aWordLd the matrix A word leading dimention
96      * @param p_As the output stream
97      *
98      */
99      void GemmReadMatA(DdrIntType* l_aAddr,
100                       unsigned int l_aColBlocks,
101                       unsigned int l_aRowBlocks,
102                       unsigned int l_bColBlocks,
103                       unsigned int l_aWordLd,
104                       DdrStream& p_As) {
105       assert(t_DdrOverXDdr != 0);
106       assert(t_DdrOverXDdr * t_XDdrWidth == t_DdrWidth);
107       for (int l_aRowBlock = 0; l_aRowBlock < l_aRowBlocks; ++l_aRowBlock) {
108           for (int l_bColBlock = 0; l_bColBlock < l_bColBlocks; ++l_bColBlock) {
109               for (int l_aColBlock = 0; l_aColBlock < l_aColBlocks; ++l_aColBlock) {
110                   for (int i = 0; i < t_aMH; ++i) {
111                       //#pragma HLS PIPELINE II = t_aColMemWords
112                       for (int j = 0; j < t_aColMemWords; ++j) {
113   #pragma HLS PIPELINE
114                           unsigned int l_aSrcOffset =
115                               l_aWordLd * t_aMH * l_aRowBlock + l_aColBlock * t_aColMemWords + i * l_aWordLd + j;
116                           DdrIntType l_word = l_aAddr[l_aSrcOffset];
117                           p_As.write(l_word);
118                       }
119                   }
120               }
121           }
122       }
123   }
```

這邊assert主要用來檢查ddr錯誤的

a col->b col -> a row

Figure. gemmKernel.hpp partial code read MatA

了解其核心架構後，這邊介紹 Fcn.hpp 中定義的內容，在 FCN class 中定義了 add preScale， pRelu ，postScal operations to the gemm results 和 gemm 的乘法。其中 FCN block 包含了完整 FCN 的架構與資料流。

```
146     void FcnBlocks(DdrIntType* p_aAddr,
147                    DdrIntType* p_bAddr,
148                    DdrIntType* p_cAddr,
149                    DdrIntType* p_xAddr,
150                    unsigned int p_aColBlocks,
151                    unsigned int p_aRowBlocks,
152                    unsigned int p_bColBlocks,
153                    unsigned int p_aLd,
154                    unsigned int p_bLd,
155                    unsigned int p_cLd,
156                    unsigned int p_xLd,
157                    unsigned int p_transpBlocks,
158                    t_FloatType (*f_act)(t_FloatType),
159                    FcnArgsType& p_Args) {
160 #pragma HLS DATAFLOW
161
162         GemmKernel<t_FloatType, t_XDataType, t_DdrWidth, t_XDdrWidth, t_aColMemWords, t_aRowMemWords, t_bColMemWords>
163             l_gemm;
164         DdrStream p_C2ScalePRelu;
165         DdrStream p_Cs;
166
167 #pragma HLS STREAM variable = p_C2ScalePRelu depth = t_DdrWidth * t_aRowMemWords * t_bColMemWords
168 #pragma HLS RESOURCE variable = p_C2ScalePRelu core = fifo_uram
169
170         l_gemm.GemmReadAndMult(p_aAddr, p_bAddr, p_xAddr, p_aColBlocks, p_aRowBlocks, p_bColBlocks, p_aLd, p_bLd, p_xLd,
171                                p_transpBlocks, p_Args.m_postScale, p_C2ScalePRelu);
172         FcnActivation(p_C2ScalePRelu, p_Cs, p_aRowBlocks * p_bColBlocks, f_act, p_Args.m_PReluVal);
173         l_gemm.GemmWriteDdrStream(p_cAddr, p_Cs, p_aRowBlocks, p_bColBlocks, p_cLd);
174     }
175
```

the base address of external memory

the No. blocks along matrix X cols/rows

the matrix word leading dimention

做好dataflow與引用GemmKernel進來

這邊用fifo_uram，針對大量的資料做儲存並且減少資料流

read資料與mult相乘並輸出結果到C

將C結果做activation

最後回傳資料

Figure. fcnBlocks source code

這邊其中有使用到 FcnActivation，在這邊有定義，整體的 activation function 要怎麼被包裝執行，並且可選擇不同的 function，包含 relu，sigmoid，tansig。

```
94      /** @brief FcnActivation applies activation function to the FCN output
95       *
96       *  @param p_inS is the input stream from FCN
97       *  @param p_outS is the output stream after applying activation
98       *  @param p_blocks is the number of blocks to be processed
99       *  @param f_act is the activation function
100      *  @param p_args is the arguments passed to the activation function
101      */
102     void FcnActivation(
103         DdrStream& p_inS, DdrStream& p_outS, unsigned int p_blocks, t_FloatType (*f_act)(t_FloatType), int16_t p_args) {
104         unsigned l_count = p_blocks * t_aRowMemWords * t_DdrWidth * t_bColMemWords;
105         if ((p_args & 0x01) == 1) {
106             for (int c = 0; c < l_count; ++c) {
107 #pragma HLS PIPELINE
108                 DdrIntType l_val = p_inS.read();
109                 p_outS.write(l_val);
110             }
111         } else {
112             for (int c = 0; c < l_count; ++c) {
113 #pragma HLS PIPELINE II = t_aColMemWords
114                 DdrWideType l_val = p_inS.read();
115                 DdrWideType l_valOut;
116                 for (int w = 0; w < t_DdrWidth; ++w) {
117                     l_valOut[w] = f_act(l_val[w]);
118                 }
119                 p_outS.write(l_valOut);
120             }
121         }
122     }
```

t_dataType選擇要使用的function (reLU, sigmoid, tansig)

這邊選擇不做activation 直接讀取與匯出資料，因此PIPELINE II=1

這邊要逐行做activation，因此PIPELINE要等column words讀完才可以在進來，因此II=t_aColMemWords

將inS資料依序做activation並匯出

Figure. FcnActivation source code

像是在 activation.hpp 中就定義了 relu，如下圖。

```
60  template <typename t_DataType>
61  t_DataType relu(t_DataType x) {
62      if (x > 0)
63          return x;
64      else
65          return 0;
66  }
```

以上定義好整的 FCN functions 之後將其用 fcnkernel 包裝起來

```
#include "ddr.hpp"
#include "ddrType.hpp"
#include "fcn.hpp"

// Compute engine types

#if BLAS_runFcn == 1
typedef xf::hpc::mlp::Fcn<BLAS_dataType,
                          BLAS_XdataType,
                          BLAS_ddrWidth,
                          BLAS_XddrWidth,
                          BLAS_gemmKBlocks,
                          BLAS_gemmMBlocks,
                          BLAS_gemmNBlocks
#if BLAS_CACHE == 1
                          ,
                          512 * 32,
                          32
#endif
                          >
    FcnType;
#endif

/**
 * @brief fcnKernel defines the kernel top function, with DDR/HBM as an interface
 *
 * @param p_DdrRd is DDR/HBM memory address used for read
 * @param p_DdrWr is DDR/HBM memory address used for write
 */
extern "C" {
void fcnKernel(DdrIntType* p_DdrRd, DdrIntType* p_DdrWr);
} // extern C

#endif
```

以上就完成整體 kernel 的設計。

5. Host 端設計

在最底層定義一個 handle 來做指令的包裝，然後在定義一個 FCN Host 將整個 kernel 包裝起來，然後再用一個 mlp_wrapper.h 定義好 xfhpc 中 mlp 會用到的所有指令如下，提供 fcn_example.cpp 所使用。

kernel functions (mlp_wrapper.h)

- bool xfhpcCreate(); // 初始化XFHPC和建立handle
- bool xfhpcMalloc(); // 在FPGA allocates memory
- bool xfhpcSetMatrix(); // 複製host端memory到FPGA端
- bool xfhpcGetMatrix(); // 複製FPGA端memory到host端
- bool xfhpcFree(); // free memory in FPGA
- void xfhpcFreeInstr(); // free instruction
- void xfhpcDestroy(); // release handle
- bool xfhpcFcn(); // 呼叫使用FCN (以instruction)

最後 fcn_example.cpp 完成 host 要做的控制指令。

Host function fcn_example.cpp (Pseudocode)

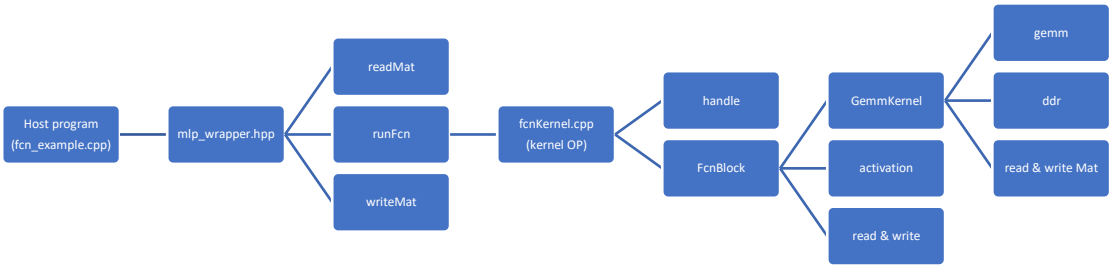
```cpp
/// host
int main(int argc, char** argv) {
    // This function initializes the XFHPC library and creates a handle for the specific engine.
    // It must be called prior to any other XFHPC library calls
    bool check = xfhpcCreate(l_xclbinFile.c_str(), l_numKernel);
    // prepare input data
    float *a, *b, *c, *bias;
    posix_memalign((void**)&a, 4096, m * k * sizeof(float));
    memset(bias, 0, 1 * n * sizeof(float));
    // setup golden data
    float* goldenC = getGoldenMat(a, b, c, true);
    // allocates memory for host row-major format matrix on the FPGA device.
    check = xfhpcMalloc(m, k, sizeof(*a), a, k, l_numKernel - 1);
    // This function copies a matrix in host memory to FPGA device memory
    check = xfhpcSetMatrix(a, l_numKernel - 1);
    //execute
    check = xfhpcFcn(m, n, k, 1, a, k, b, n, 1, c, n, bias, n, 1, 0, 1, 0, l_numKernel - 1);
    // This function copies a matrix in FPGA device memory to host memory
    check = xfhpcGetMatrix(c, l_numKernel - 1);
    // compare and show results
    if (compareFCN(c, goldenC)) cout << "Test passed!\n";
    else cout << "Test failed!\n";
    // This function frees memory in FPGA device
    xfhpcFree(a, l_numKernel - 1);
    // This function frees instrution
    xfhpcFreeInstr(l_numKernel - 1);
    // free memory
    free(a);
    // This function releases handle used by the XFHPC library.
    xfhpcDestroy(l_numKernel);

    return EXIT_SUCCESS;
}
```

1.Setup environment
2.prepare data
3.send it into FPGA

Execute and compare result

Clean memory used

其整體 system 架構大概如下



而其最簡單得核心內容就在於 FCN 所需要完成的$C = fcn(A * B + X)$為基礎。

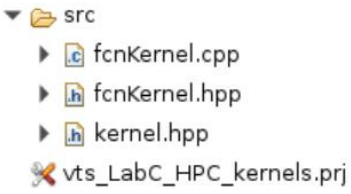
二、Findings from the lab work (design flow)

其 design flow 與 lab 3 大致相同，使用 makefile 的話，首先先在 terminal 端輸入 scl enable devtoolset-9 bash 建立一個 bash 環境之後可以直接指令如下。選定 device=xilinx_u50_gen3x16_xdma_201920_3。為我們所有的板子。並且 host_arch=x86。

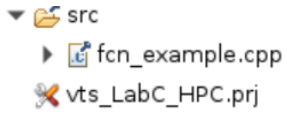
```
help::
  $(ECHO) "Makefile Usage:"
  $(ECHO) "  make all TARGET=<sw_emu/hw_emu/hw> DEVICE=<FPGA platform> HOST_ARCH=<aarch32/aarch64/x86>"
  $(ECHO) "      Command to generate the design for specified Target and Shell."
  $(ECHO) "      By default, HOST_ARCH=x86. HOST_ARCH is required for SoC shells"
  $(ECHO) ""
  $(ECHO) "  make clean "
  $(ECHO) "      Command to remove the generated non-hardware files."
  $(ECHO) ""
  $(ECHO) "  make cleanall"
  $(ECHO) "      Command to remove all the generated files."
  $(ECHO) ""
  $(ECHO) "  make TARGET=<sw_emu/hw_emu/hw> DEVICE=<FPGA platform> HOST_ARCH=<aarch32/aarch64/x86>"
  $(ECHO) "      By default, HOST_ARCH=x86. HOST_ARCH is required for SoC shells"
  $(ECHO) ""
  $(ECHO) "  make run TARGET=<sw_emu/hw_emu/hw> DEVICE=<FPGA platform> HOST_ARCH=<aarch32/aarch64/x86>"
  $(ECHO) "      Command to run application in emulation."
  $(ECHO) "      By default, HOST_ARCH=x86. HOST_ARCH required for SoC shells"
  $(ECHO) ""
  $(ECHO) "  make build TARGET=<sw_emu/hw_emu/hw> DEVICE=<FPGA platform> HOST_ARCH=<aarch32/aarch64/x86>"
  $(ECHO) "      Command to build xclbin application."
  $(ECHO) "      By default, HOST_ARCH=x86. HOST_ARCH is required for SoC shells"
  $(ECHO) ""
  $(ECHO) "  make host HOST_ARCH=<aarch32/aarch64/x86>"
  $(ECHO) "      Command to build host application."
  $(ECHO) "      By default, HOST_ARCH=x86. HOST_ARCH is required for SoC shells"
  $(ECHO) ""
  $(ECHO) "  NOTE: For SoC shells, ENV variable SYSROOT needs to be set."
  $(ECHO) ""
```

而使用 GUI 的話，打開 terminal>> vitis >>new application project >> xilinx_u50_gen3x16_201920_3 >> new empty project。

在 kernel project 匯入以下檔案。



然後在 host project 匯入以下檔案

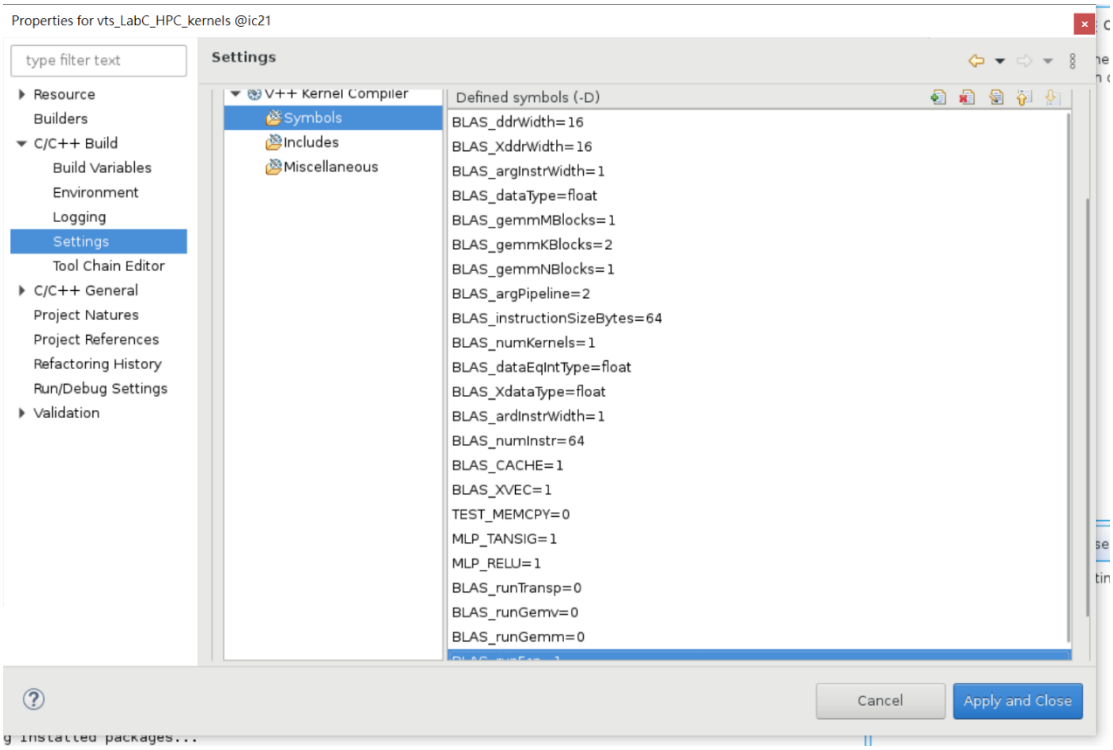


然後在 hw_link project 中點選 add binary container

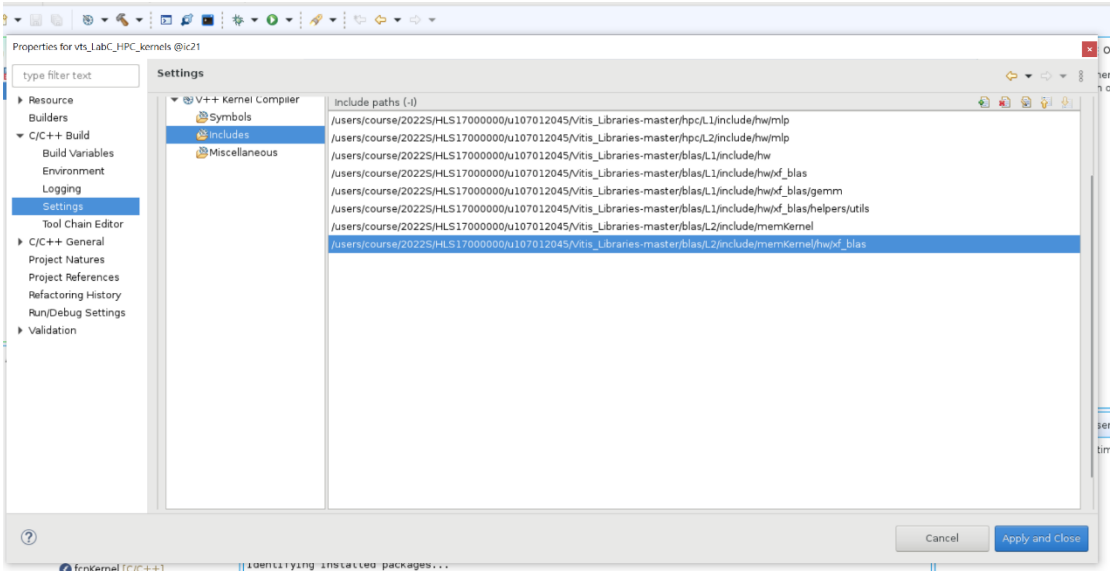
並且在 kernel project 中點選 add hardware function，選擇 fcnkernel。

再來點選 kernel project 點擊>>file>properties 進入以下畫面

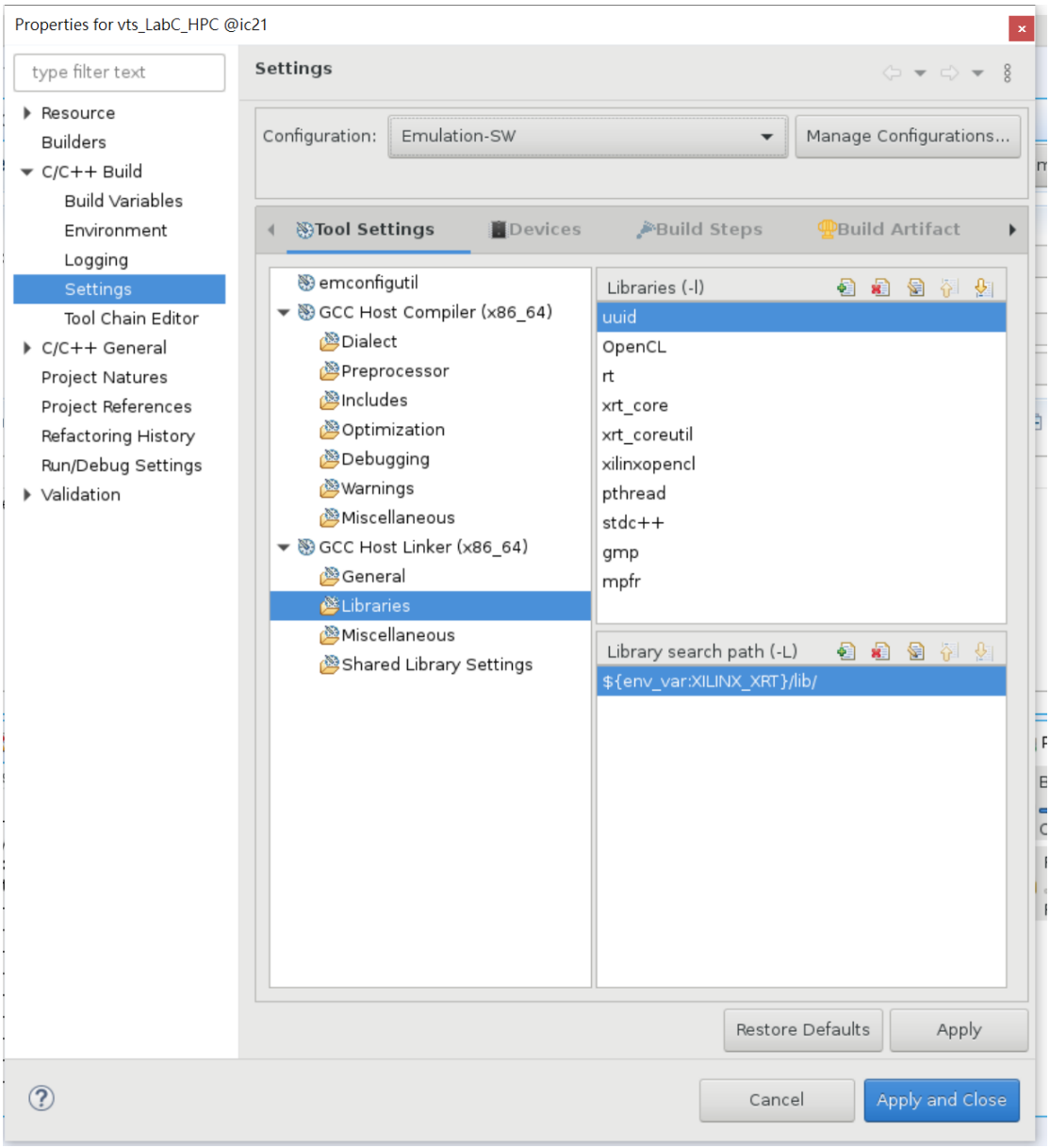
首先添加 symbols ，這邊在 makefile.param 中有定義環境變數。



添加完環境變數後點選 include 添加路徑，這些路徑在 makefile 中有。



最後在 GCC host linker> libraries 添加以下資料(uuid, OpenCL,xrt_coreutil)



確認完成後即可進行 Emulation-SW build。

依序從 kernels>>hw-link>>host>>system 做 build。

這邊要注意，如果出現以下錯誤

```
21:34:58 **** Incremental Build of configuration Emulation-SW for project vts_LabC_HPC ****
make -j64 all
g++ -std=c++1y -DVITIS_PLATFORM=xilinx_u50_gen3x16_xdma_201920_3 -D__USE_XOPEN2K8 -DBLAS_dataType=float -DBLAS_XdataType=float -DBLAS_XddrWidth=16 -
DBLAS_gemmKBlocks=2 -DBLAS_gemmMBlocks=1 -DBLAS_gemmNBlocks=1 -DBLAS_CACHE=1 -DBLAS_runFcn=1 -DBLAS_numInstr=64 -DBLAS_argInstrWidth=1 -
DBLAS_argPipeline=2 -DBLAS_ddrWidth=16 -I/opt/xilinx/xrt/include/ -I/opt/Xilinx/Vitis_HLS/2021.2/include/ -I/users/course/2022S/HLS17000000/u107012045/Vitis_Libraries-
master/hpc/L3/include/sw/mlp -I/users/course/2022S/HLS17000000/u107012045/Vitis_Libraries-master/blas/L3/include/sw/xf_blas -O0 -g -Wall -c -fmessage-length=0 -o
"src/fcn_example.o" "../src/fcn_example.cpp"
g++ -o "vts_LabC_HPC" src/fcn_example.o -luuid -lOpenCL -lrt -lxrt_core -lxrt_coreutil -lxilinxopencl -lpthread -lstdc++ -lgmp -lmpfr -L/opt/xilinx/xrt/lib/ -Wl,-rpath-link,/opt/xilinx/xrt/lib
src/fcn_example.o: In function `xf::blas::XFpga::XFpga(char const*, int*, unsigned int)':
/users/course/2022S/HLS17000000/u107012045/Vitis_Libraries-master/blas/L3/include/sw/xf_blas/host.hpp:76: undefined reference to `xrt::device::load_xclbin(std:
:__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&)'
src/fcn_example.o: In function `xf::blas::XHost::XHost(char const*, xf::blas::xfblasStatus_t*, unsigned int, unsigned int)':
/users/course/2022S/HLS17000000/u107012045/Vitis_Libraries-master/blas/L3/include/sw/xf_blas/host.hpp:144: undefined reference to `xrt::kernel::kernel(xrt::device const&, xrt::uuid
const&, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> > const&, xrt::kernel::cu_access_mode)'
collect2: error: ld returned 1 exit status
make: *** [vts_LabC_HPC] Error 1

21:35:00 Build Finished (took 2s.438ms)
```

則需要在 host 端 dialect 添加 -std=c++z

完成之後則可以進行 Emulation-SW>> run configuration>> 添加 program argument

Usage: ./binary_container_1.xclbin 1

這邊要在環境配置下添加

export LC_ALL=C

即可執行，但由於不確定的原因導致執行結果會計算到一半跳出如下

```
****** configutil v2021.2 (64-bit)
  **** SW Build 3363252 on 2021-10-14-04:41:01
    ** Copyright 1986-2020 Xilinx, Inc. All Rights Reserved.

INFO: [ConfigUtil 60-895]   Target platform: /opt/xilinx/platforms/xi
linx_u50_gen3x16_xdma_201920_3/xilinx_u50_gen3x16_xdma_201920_3.xpfm
INFO: [ConfigUtil 60-1578]   This platform contains Xilinx Shell Arch
ive '/opt/xilinx/platforms/xilinx_u50_gen3x16_xdma_201920_3/hw/hw.xsa
'
INFO: [ConfigUtil 60-1032] Extracting hardware platform to build_dir.
sw_emu.xilinx_u50_gen3x16_xdma_201920_3
emulation configuration file `emconfig.json` is created in build_dir.
sw_emu.xilinx_u50_gen3x16_xdma_201920_3 directory
cp -rf build_dir.sw_emu.xilinx_u50_gen3x16_xdma_201920_3/emconfig.jso
n .
XCL_EMULATION_MODE=sw_emu build_dir.sw_emu.xilinx_u50_gen3x16_xdma_20
1920_3/fcn_example.exe build_dir.sw_emu.xilinx_u50_gen3x16_xdma_20192
0_3/fcn.xclbin
Software emulation of compute unit(s) exited unexpectedly
make: *** [Makefile:188: run] Terminated
Terminated
```

而在 Emulation-HW 執行結果上可以正常執行，但其結果會與 FPGA 計算得到的結果不同，如下

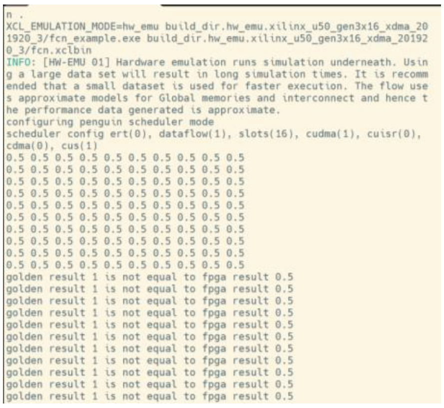
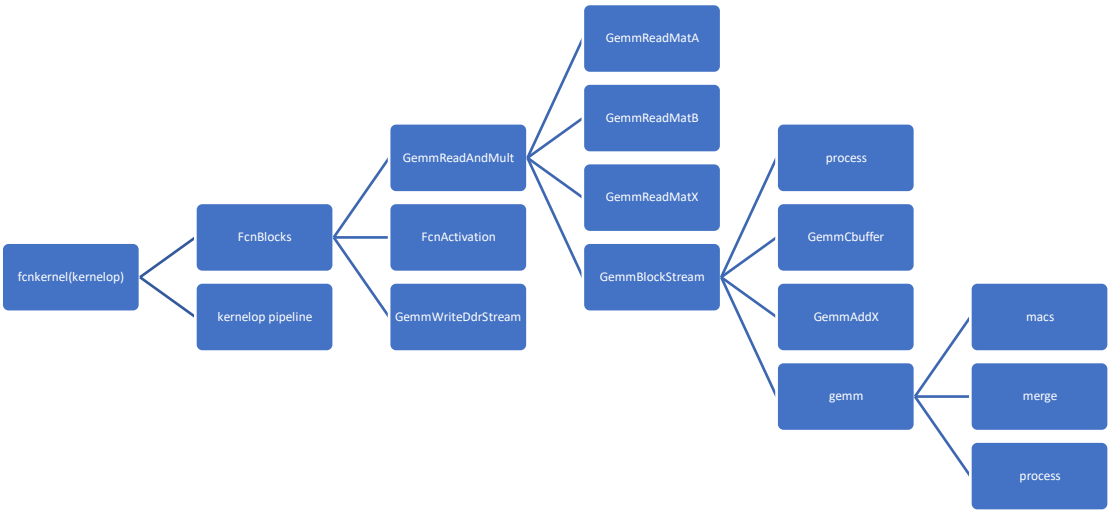

```
n .
XCL_EMULATION_MODE=hw_emu build_dir.hw_emu.xilinx_u50_gen3x16_xdma_20
1920_3/fcn_example.exe build_dir.hw_emu.xilinx_u50_gen3x16_xdma_20192
0_3/fcn.xclbin
INFO: [HW-EMU 01] Hardware emulation runs simulation underneath. Usin
g a large data set will result in long simulation times. It is recomm
ended that a small dataset is used for faster execution. The flow use
s approximate models for Global memories and interconnect and hence t
he performance data generated is approximate.
configuring penguin scheduler mode
scheduler config ert(0), dataflow(1), slots(16), cudma(1), cuisr(0),
cdma(0), cus(1)
0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5
golden result 1 is not equal to fpga result 0.5
golden result 1 is not equal to fpga result 0.5
golden result 1 is not equal to fpga result 0.5
golden result 1 is not equal to fpga result 0.5
golden result 1 is not equal to fpga result 0.5
golden result 1 is not equal to fpga result 0.5
golden result 1 is not equal to fpga result 0.5
golden result 1 is not equal to fpga result 0.5
golden result 1 is not equal to fpga result 0.5
golden result 1 is not equal to fpga result 0.5
golden result 1 is not equal to fpga result 0.5
```

Figure. hw emulation result

三、Analysis

我們針對 HLS 合成結果進行分析

其架構大致如下



我們再來看其消耗的資源

Kernel 的部分

	BRAM	DSP	FF	LUT	Slack
fcnkernel	46(1%)	1272(21%)	442929(25%)	278898(31%	-1.22
FcnBlocks	0(0%)	1264(21%)	4435563(24%)	273324(31%)	-1.22

再來是 Gemm 的部分，可以看到 Gemm 裡面沒有使用到 BRAM，另外其整體 DSP 的部分都是 Mac 乘加法器所使用到的，單一一個 Macs 需要 72 個 DSP。再來觀看整體 fabric 的部分，可以看到 GemmReadandMult，大部分的硬體都實作在 Macs 上了，幾乎就佔了 66%左右。但整體 timing violation 的部分是實現在 merge 的地方，這邊是將 16 MAC 組合起來並且做資料傳遞與連接的部分。

	BRAM	DSP	FF	LUT	Slack
GemmReadAndMult	0	1255	430025	269497	-1.22
GemmReadMatA	0	21	3157	1971	0
GemmBlockStream	0	1221	421050	263926	-1.22
gemm	0	1152	360046	203783	-1.22
Macs(1 個)	0	72	17552	10372	0
Macs(總共 16 個)	0	1152	284032	165952	0
merge_4_2	0	0	36067	14130	-1.22

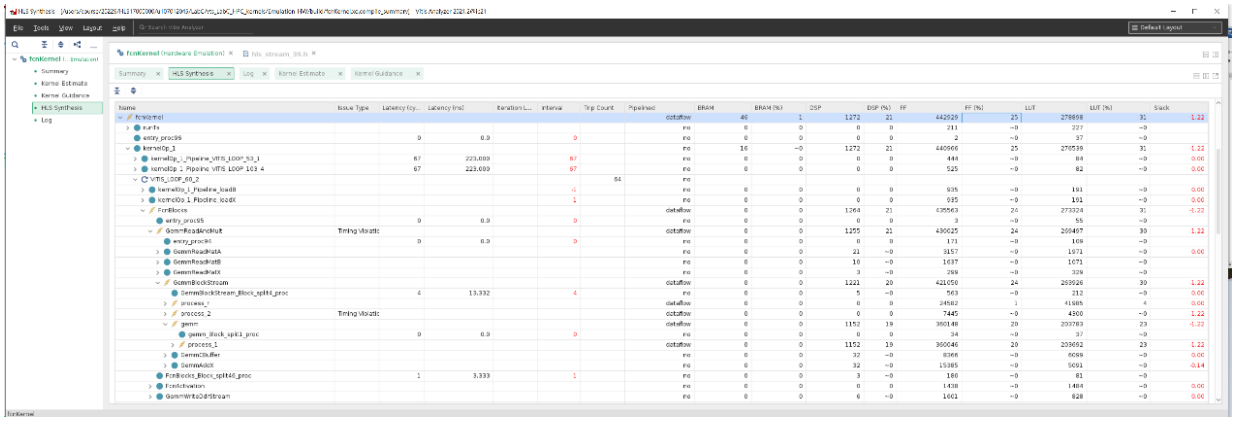
因此我們可以推估整體的 critical timing 受限於 gemmKernel.hpp 的 runGemm function 之下。

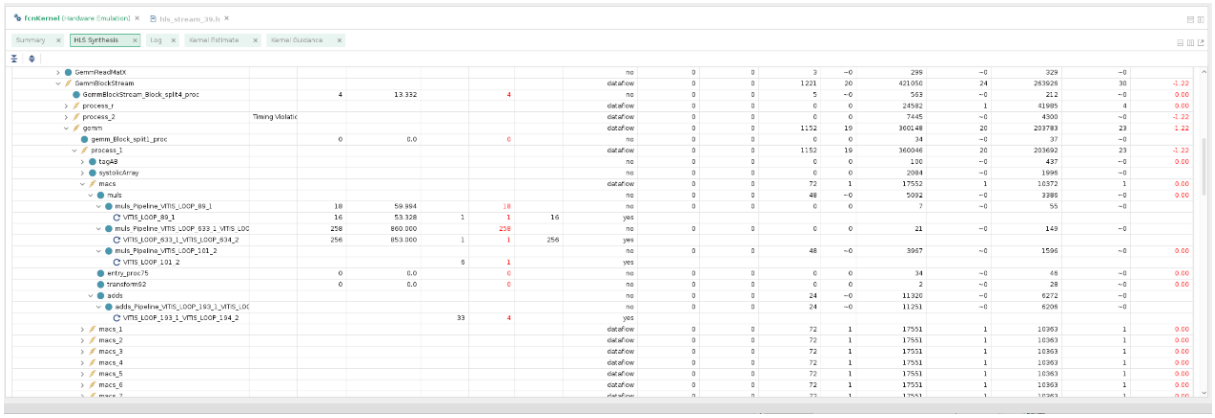
```
502     */
503     void runGemm(DdrIntType* p_DdrRd, // base DDR/memory address for matrix A and B
504                  DdrIntType* p_DdrWr, // base DDR/memory address for matrix C
505                  GemmArgsType& p_Args // GEMM argument that stores the address offset of matrix A, B and C, sizes of
506                  // matrix dimensions (M, K and N) and lead dimension sizes for matrix A, B and C
507                  ) {
508         DdrIntType* l_aAddr = p_DdrRd + p_Args.m_Aoffset * DdrWideType::per4k();
509         DdrIntType* l_bAddr = p_DdrRd + p_Args.m_Boffset * DdrWideType::per4k();
510         DdrIntType* l_xAddr = p_DdrRd + p_Args.m_Xoffset * DdrWideType::per4k();
511         DdrIntType* l_cAddr = p_DdrWr + p_Args.m_Coffset * DdrWideType::per4k();
512
513         const unsigned int l_aColBlocks = p_Args.m_K / (t_DdrWidth * t_aColMemWords);
514         const unsigned int l_aRowBlocks = p_Args.m_M / (t_DdrWidth * t_aRowMemWords);
515         const unsigned int l_bColBlocks = p_Args.m_N / (t_DdrWidth * t_bColMemWords);
516         const unsigned int l_aLd = p_Args.m_Lda / t_DdrWidth;
517         const unsigned int l_bLd = p_Args.m_Ldb / t_DdrWidth;
518         const unsigned int l_cLd = p_Args.m_Ldc / t_DdrWidth;
519         const unsigned int l_xLd = p_Args.m_Ldx / t_XDdrWidth;
520         const int32_t l_postScale = p_Args.m_postScale;
521         unsigned int l_transpBlocks = l_aColBlocks * l_aRowBlocks * l_bColBlocks * t_aRowMemWords;
522         GemmBlocks(l_aAddr, l_bAddr, l_cAddr, l_xAddr, l_aColBlocks, l_aRowBlocks, l_bColBlocks, l_aLd, l_bLd, l_cLd,
523                    l_xLd, l_transpBlocks, l_postScale);
524     }
525
```

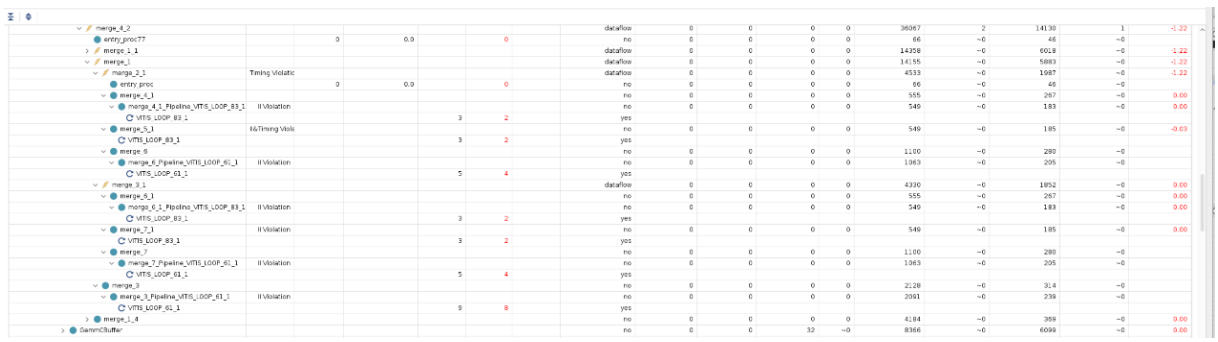
這邊有包含 Macs 的部分以及 merge 的部分。可以看到這邊有做相當多的乘法與除法，相當的耗資源，再往下看到 Gemmblocks 中裡面有包含 gemmblockstream。

```
397     void GemmBlockStream(DdrStream& p_As,
398                          DdrStream& p_Bs,
399                          XDdrStream& p_Xs,
400                          DdrStream& p_Cs,
401                          unsigned int p_aColBlocks,
402                          unsigned int p_aRowBlocks,
403                          unsigned int p_bColBlocks,
404                          unsigned int p_transpBlocks,
405                          int32_t p_postScale) {
406         unsigned int l_cBlocks = p_aRowBlocks * p_bColBlocks;
407         unsigned int l_abBlocks = l_cBlocks * p_aColBlocks;
408     #pragma HLS DATAFLOW
409                                 這邊的架構因為資料是固定依照function樹續走的，因此使用dataflow
410         DdrStream p_Bs1, p_AoutS, p_CBufferS;
411         EdgeStream p_AEdgeS0, p_BEdgeS0;
412         WideMacBitStream p_CEdgeS, p_COutS;
413     #pragma HLS STREAM variable = p_CEdgeS depth = t_DdrWidth * t_aRowMemWords * t_bColMemWords
414     #pragma HLS RESOURCE variable = p_CEdgeS core = fifo_uram
415
416         /*
417           Transp<t_FloatType, t_DdrWidth, t_aColMemWords, 1> l_transp;
418           l_transp.processWithReuse(p_As, p_AoutS, p_transpBlocks, t_bColMemWords);
419           */
420                                         讓A做transpose
421         Transpose<t_FloatType, t_aColMemWords, t_DdrWidth> l_transp(p_transpBlocks, t_bColMemWords);
422         l_transp.process(p_As, p_AoutS);
423                                                         讓Matrix做buffer緩存 b，等A做完transpose
424         MatrixBuffer<typename DdrWideType::t_TypeInt, t_DdrWidth * t_aColMemWords, t_bColMemWords, true, false>()
425             .process(p_Bs, p_Bs1, l_abBlocks, t_aRowMemWords);
426
427         Gemm<t_FloatType, t_bKD, t_DdrWidth>::gemm(p_AoutS, p_Bs1, p_CEdgeS,        實際呼叫Gemm的指令
428                                                    l_abBlocks * t_aRowMemWords * t_bColMemWords);
429
430         GemmCBuffer(p_CEdgeS, p_aColBlocks, l_cBlocks, p_COutS);
431         GemmAddX(p_COutS, p_Xs, l_cBlocks, p_postScale, p_Cs);
```

最後是整體資源的分析

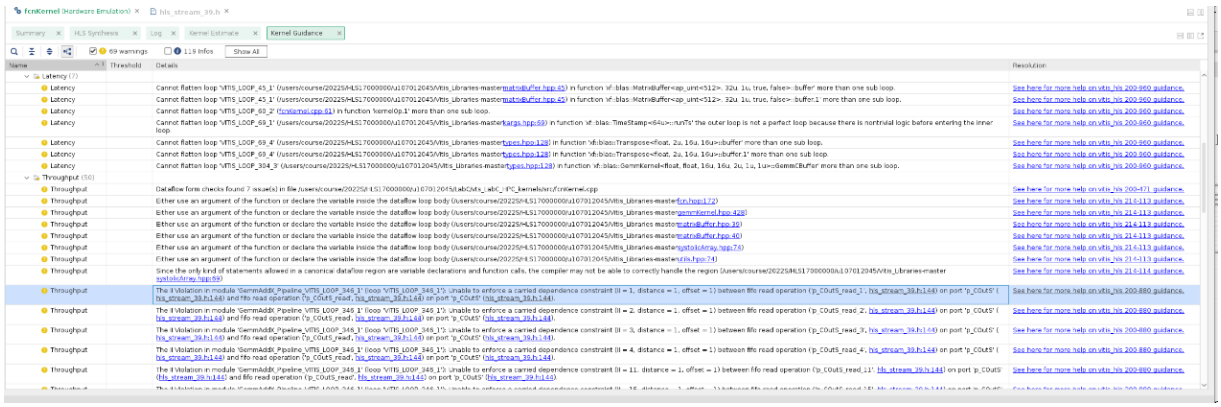






最後這邊有出現一些 warning

大部分都是在講在 gemm 中的 II violation，以及在 mixmatrix 中無法 flatten loop 導致 latency 大。



四、Suggestion for improvement

1. 希望能夠修改 matrix 的大小已符合可以讓 u50 板子可以執行 Emulation-SW。
2. 希望能夠修改讓 Emulation-HW 可以執行預期的結果。
3. 原先以為可以優化的部分應該在 fcn_kernel 的地方，但經過合成之後發現應該優化的地方應該是在 gemm 的部分，屬於 blas libraries 的部分。希望能夠優化 gemm 來提升效能。